Design and modular self-assembly of nanostructures

Joakim Bohlin

Balliol College
University of Oxford

A thesis submitted for the degree of Doctor of Philosophy

Michaelmas 2021

Abstract

As nucleic acid nanostructures grow larger and more complex, new tools and methods are needed to facilitate their design. DNA origami structures, for example, are limited by the lengths of their scaffolds, but can they be joined together into larger multi-component designs. This thesis presents two projects, each approaching the design of such modular structures at a different level of abstraction.

Project 1 presents the *polycube* self-assembly model, where building blocks assemble stochastically using complementary patches. The assembly of both 2D polyominoes, as well as 3D polycubes, is considered. First, the mapping between input rules (defining the set of available species) and output shapes is investigated, revealing a clear bias toward low-complexity structures. The frequent shapes also tend to be highly modular and symmetric. Secondly, the reversed mapping is explored, presenting a method to find the minimal rule that assembles a provided output shape.

Project 2 presents a more detailed approach to the design of modular structures and individual modules; the *oxView* toolkit for the design, analysis, and visualisation of DNA, RNA and protein nanostructures. While many other design tools exist, oxView makes it easy to import and connect their designs into complete assemblies. Furthermore, oxView allows for free-form editing and rigid-body manipulation. Designs can then be interactively simulated using the oxDNA model, providing a more intuitive understanding of the dynamics.

In conclusion, nanostructures self-assembly design has been investigated on both an abstract a more detailed level. The presented projects have resulted in tools and methods for creating, simulating and analysing self-limiting modular structures with minimal complexity, potentially containing building blocks created in different design software.

Design and modular self-assembly of nanostructures

Joakim Bohlin

Balliol College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2021

This thesis is dedicated to
Ellen ♡ʖ
for joining me on this adventure

Acknowledgements

First and foremost, I would like to thank my supervisors, Professor Andrew Turberfield and Professor Ard Louis, for giving me this opportunity to begin with, and for all their help, knowledge, and support. Andrew has provided sound and rational advice whenever I have needed it, helping me to ensure the scientific relevance of my work and making sure I focus on what is important. Likewise, Ard has provided much appreciated inspiration and ideas for new research directions and collaborations.

I would also like to thank my fellow members of the Turberfield lab. For their help, for a friendly working environment and for excellent beta testing; Dr. Jonathan Bath, Rafael Carrascosa Marzo, Dr. Erik Benson, Dr. Seham Helmi, Dr. Antonio Garcia Guerra, Behnam Najafi, Emma Silvester, Robert Oppenheimer, Catherine Fan, Alma Chapet–Batlle, Sing Ming Chan, Qian Zhang, and Dr. Rana Abdul Razzak.

I am also very thankful to Professor Petr Šulc, at Arizona State University, for his advice and guidance concerning oxView development, patchy particle simulation, as well as oxRNA. The secondment I spent in the Šulc group was a very valuable and productive time. My fellow oxView developers Erik Poppleton, Michael Matthies, Jonah Procyk, and Aatmik Mallya deserve gratitude for their hard and excellent work.

Similarly, I would also like to thank Prof. Ebbe Andersen at Aarhus University for my secondment in his group and for introducing me to RNA origami. Also, thank you Néstor Sampedro for all the help and valuable discussions.

Finally, I appreciate the help and input I have received from Prof. Jonathan Doye and his group, including Hannah Fowler, Domen Prešern and others.

On a more personal note, for her undying support (despite my endless ramblings about polycubes and simulations), my beloved wife Ellen Bohlin. Thank you for joining me on yet another adventure and for enduring the times I still had to leave you behind. I simply cannot thank you enough.

Of course, also want to thank the rest of my family. My father Ulf, who I am certain would also have loved to study here had he been given the opportunity. My mother Okki, for her creativity, her care, and encouragement. My two sisters, Ann-Marie and Ingela, for inspiring me to travel and to study.

I am also grateful for the support and encouragement I have received from Ellen's side of the family. My parents-in-law Karl-Johan and Ann-Louise are like an

extra set of parents, but I want to specifically acknowledge Ellen's late grandfather, Anders Bohlin. A botanist in the spirit of Linnaeus, he

Gunnar, Emily, Asami, Wheatley Ryobukai

Bruce, Helmer, Harriet, Peter

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 765703

Abstract

As nucleic acid nanostructures grow larger and more complex, new tools and methods are needed to facilitate their design. DNA origami structures, for example, are limited by the lengths of their scaffolds, but can they be joined together into larger multi-component designs. This thesis presents two projects, each approaching the design of such modular structures at a different level of abstraction.

Project 1 presents the *polycube* self-assembly model, where building blocks assemble stochastically using complementary patches. The assembly of both 2D polyominoes, as well as 3D polycubes, is considered. First, the mapping between input rules (defining the set of available species) and output shapes is investigated, revealing a clear bias toward low-complexity structures. The frequent shapes also tend to be highly modular and symmetric. Secondly, the reversed mapping is explored, presenting a method to find the minimal rule that assembles a provided output shape.

Project 2 presents a more detailed approach to the design of modular structures and individual modules; the *oxView* toolkit for the design, analysis, and visualisation of DNA, RNA and protein nanostructures. While many other design tools exist, oxView makes it easy to import and connect their designs into complete assemblies. Furthermore, oxView allows for free-form editing and rigid-body manipulation. Designs can then be interactively simulated using the oxDNA model, providing a more intuitive understanding of the dynamics.

In conclusion, nanostructures self-assembly design has been investigated on both an abstract a more detailed level. The presented projects have resulted in tools and methods for creating, simulating and analysing self-limiting modular structures with minimal complexity, potentially containing building blocks created in different design software.

Contents

List of Figures

List of Tables

Glossary

2D, 3D . . . . Two- or three-dimensional, referring in this thesis to spatial dimensions of a self-assembly structure.

DNA . . . . . Deoxyribonucleic acid.

RNA . . . . . Ribonucleic acid.

PDB . . . . . The Protein Data Bank. Used in this thesis to refer to the file format used to save atomic structures.

SAT . . . . . Boolean satisfiability.

Species . . . . Used in this thesis to denote a type of cube allowed by the polycube rule.

Polycube . . . A structure consisting of multiple cubes connected by their sides.

Far out in the uncharted backwaters of the unfashionable end of the western spiral arm of the Galaxy lies a small unregarded yellow sun. Orbiting this at a distance of roughly ninety-two million miles is an utterly insignificant little blue green planet whose ape-descended life forms are so amazingly primitive that they still think digital watches are a pretty neat idea.

— D. Adams, The Hitchhiker's Guide to the Galaxy

1 Introduction

Contents

How do you design and assemble something on the nanoscale? Imagine building something with lego bricks; you choose the building blocks you want and use your hands to attach them where you want them to be. However, for nanoscale objects, it is difficult to have that level of control. Instead, more success has been had by imitating nature and letting the building blocks assemble themselves. This thesis will cover novel methods and tools to design such self-assembling nanostructures.

1.1 Thesis structure

This thesis covers two related projects, both concerning the design and modular self-assembly of nanostructures. Each project is introduced by a separate chapter on its relevant history and background.

The first project, introduced by Chapter 2, covers an abstract self-assembly model called *polycubes*. Chapter 3 details the model and the shapes that assemble

when randomly sampling the input space. Chapter 4 presents the results on the reverse problem; given a polycube shape, what input rules will assemble it and can you minimise input complexity?

The second project takes a more detailed view of self-assembly design. Chapter 5 gives a background on computer-aided design tools for nucleic acid structures, followed by an introduction of the models used to simulate such designs. Chapter 6 then presents my contributions to *oxView*, a web-based tool for the visualisation, design, and integration of DNA, RNA and protein structures.

This chapter serves as an introduction to both projects, providing background on DNA and RNA self-assembly.

1.2 DNA design

Deoxyribonucleic acid (DNA) is a string-like molecule used to encode the genes of living systems [1]. These strings are made up of units called *nucleotides*, consisting of a sugar-phosphate backbone as well as one of four possible bases: *adenine* (A), *thymine* (T) *cytosine* (C), and *guanine* (G).

Through Watson-Crick base-pairing, A forms two hydrogen bonds with T, while G forms three with C, making DNA double-stranded. Each strand has a directionality, conventionally represented as going from the 3' to the 5' end of the strand, making the duplex anti-parallel.

The bases of the duplex are hydrophobic, while the sugar-phosphate backbone is hydrophilic, which means that the bases "hide" on the inside of the duplex to avoid contact with water molecules. However, the backbone distance is about 6 Å (0.6 nm), while the bases would need to be at a distance of 3.3 Å (the thickness of the base) to not leave any room for water [1]. To solve this, the DNA duplex forms a double helix structure with a radius of about 9 Å, placing everything at an energetically comfortable distance.

While a single double-helix is the most natural confirmation, it is possible for strands to branch into multiple junctions. For example, the Holliday junction

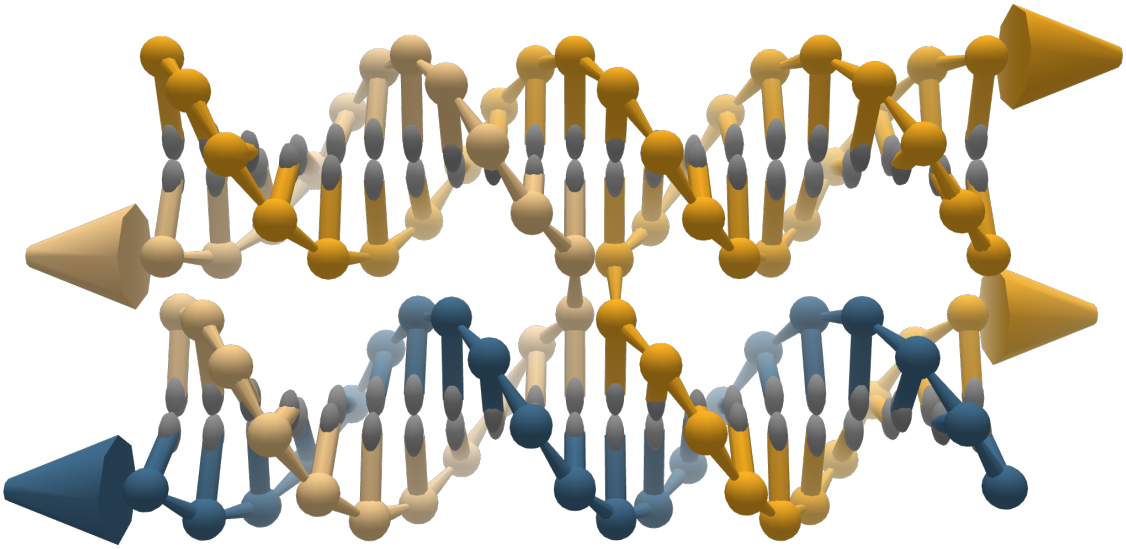


Figure 1.1: Holliday junction, designed in oxView. Cones at the end indicates the 3' end of each of the four strands.

is a junction between four double-helical arms, shown in Figure 1.1 in one of its possible configurations.

By designing sequences with complementary domains for the intended duplex regions, it is possible to create many different DNA motifs and structures [2].

A breakthrough in the field of structural DNA nanotechnology was the DNA origami technique [3] a now popular and proven method for creating larger irregular structures using DNA. The principle behind it, as illustrated in Figure 1.2, is to use short staple strands to fold one long viral scaffold strand into the desired structure.

Using design tools such as caDNAno[5], it has become relatively easy to design structures of any given form. See Section 5.1 for an introduction to additional such tools. However, the size of the origami is limited by the length of the scaffold. This motivates researchers to investigate modular approaches, some of which will be described in Section 2.1.

1.3 RNA design

Ribonucleic acid (RNA) is very similar to DNA, but with the *thymine* base replaced by *uracil* (U). Just like DNA, it is possible to use it as a self-assembling building material.

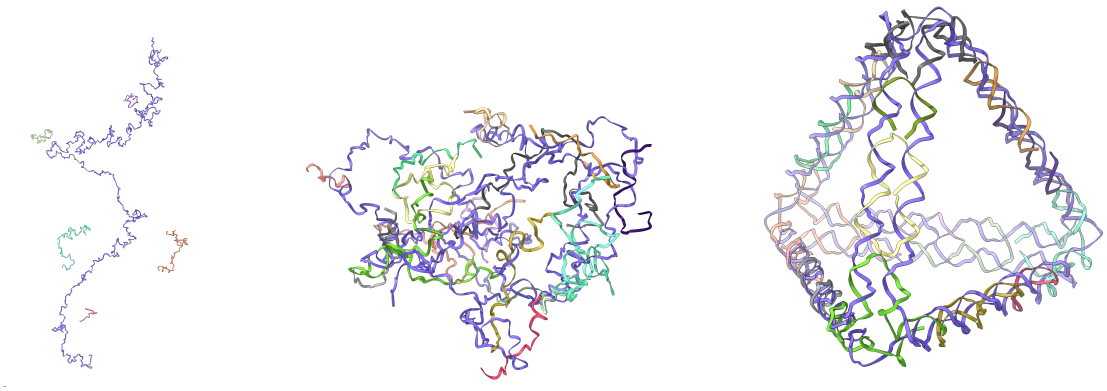


Figure 1.2: Illustration of DNA origami self-assembly of a tetrahedron. A long scaffold strand (purple), obtained from a virus, is folded into the desired shape by multiple short staple strands binding to complementary domains of the scaffold. Tetrahedron design obtained from `https://cando-dna-origami.org/examples/` and melted using oxDNA simulation [4].

Biologically, DNA is transcribed into RNA by the RNA polymerase enzyme as part of gene expression. While DNA folding is easier to predict, the fact that RNA is more reactive than DNA also offers the possibility of a more useful structure; for example, by incorporating aptamers, enzymes and other such functionalities[6].

Geary et al., from the Andersen lab in Aarhus, demonstrated a method[7–9] for co-transcriptionally folded RNA origami in 2014, which also enables folding *in vivo*. The design used a set of tertiary RNA motifs, such as kissing hairpins and double crossovers, to fold the transcribed RNA strand into the desired structure. The Andersen lab is one of the network partners, and I have spent a two-month secondment there working with their RNA origami method.

1.4 Algorithmic Information Theory and input-output maps

If you have a monkey pressing random keys on a typewriter, you would expect it to produce every string of length N with equal probability (assuming the keystrokes were indeed truly random). With k keys on the keyboard, the probability for any string of length N is then k^{-N}. For example, the title of this thesis, while unlikely to appear randomly, would be equally as probable as any other 50-character string, see the three example strings below:

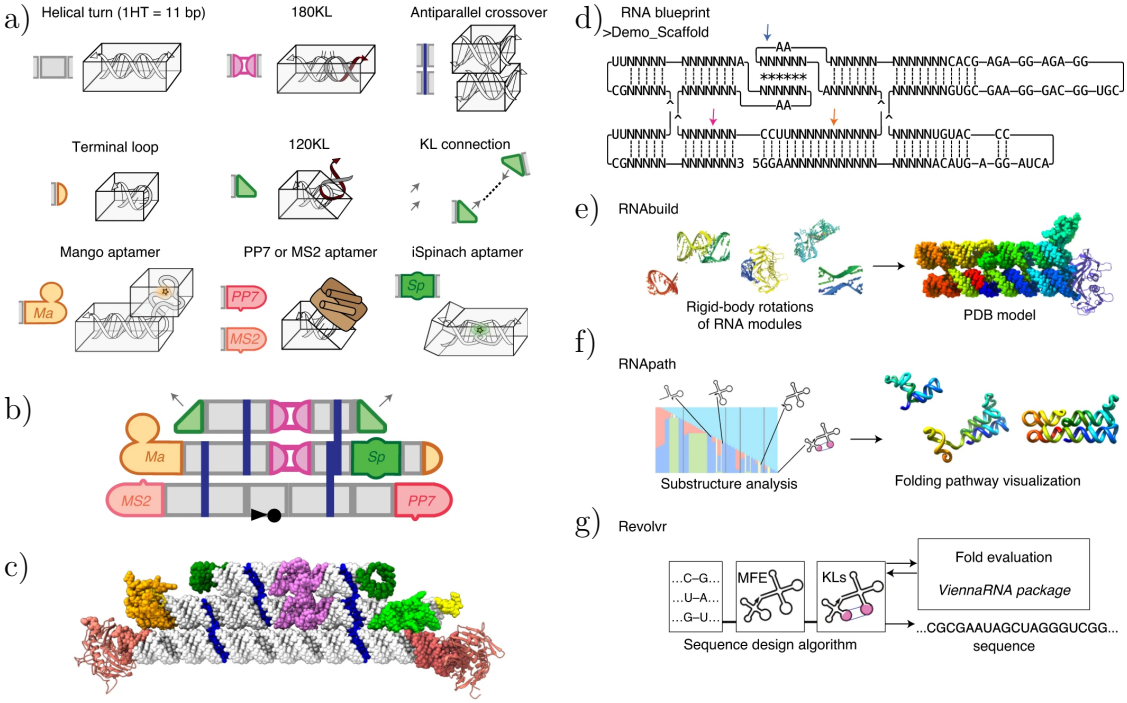


Figure 1.3: Co-transcriptional folding of RNA origami, adapted from [9]. a) The set of RNA motifs used as modular building blocks. b) Schematic of the modules connected to form a single strand. c) Atomistic model of the design in b). d) Shows the text-based blueprint used to create designs, while e), f), and g) shows scripts developed to aid the visualisation and preform sequence design for the origami.

```
1   DESIGN AND MODULAR SELF-ASSEMBLY OF NANOSTRUCTURES
2   SHWDRVWKFORWJDOEXOZLSBNREKC Z VSDJJF ROKFYRVMIUI
3   AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

However, some strings can have a description shorter than a full listing of the letters it contains. For example, string number three above could be described simply as "Print `A`, 50 times", or if we use the C programming language:

```
for(int i=50; i--;) printf("A");
```

There are also a number of possible valid variations of the code above, with different variable names and coding conventions, all producing the same output. In other words, not only is the description shorter than writing out the full string but multiple inputs map to the same output, increasing the probability of the output further.

The concept of using the shortest possible computer program that can describe an object (for example, a binary string) to determine its complexity is central within the field of Algorithmic Information Theory (AIT) and is called Kolmogorov complexity (or Solomonoff–Kolmogorov–Chaitin to give full credit) [10]. More specifically, the Kolmogorov complexity $K(x)$ of an output x is the length of the shortest program that generates x on a Universal Turing Machine (UTM).

AIT includes the *coding theorem*, introducing lower and upper bounds for the probability $P(x)$ of generating a binary string x as $2^{-K(x)} \leq P(x) \leq 2^{-K(x)+\mathcal{O}(1)}$. In other words, low-complexity outputs are exponentially more likely to generated by random input compared to high-complexity outputs. This could be compared to how, intuitively, there are likely more programs generating the "simple" string number 3 above compared to the randomly generated number 2.

However, finding the shortest program for an output is far from trivial (in general, it is in fact *uncomputable*, due to the *halting problem*). Fortunately, Dingle at al [11] were able to derive an upper bound to the probability using a computable approximation $\widetilde{K}(x)$ of the Komologrov complexity:

$$P(x) \lesssim 2^{-a\widetilde{K}(x)-b}$$

where a and b are constants that depend on the input-output map used (but are independent of x).

Simplicity is prerequisite for reliability.

— Edsger W. Dijkstra

2

An introduction to modular assembly

Contents

From the start, the field of structural DNA nanotechnology has been interested in modular assemblies, although initially in the form of infinite and uniform crystal structures. In fact, Professor Ned Seeman was inspired to pioneer the field after seeing the woodcut *Depth* by M.C. Escher [2], where fish are depicted organised into a neat crystalline structure.

However, components that self-assemble into chrystals can also assemble bounded structures, if a way can be found to make them self-limiting. The question is how many unique components would be required. This chapter will provide an

overview of the background of self-assembled multicomponent structures, starting with experimental results of ever-increasing size and followed by a selection of theoretical models.

2.1 Experimental applications

From small tiles made from a handful of strands to megadalton-scale structures made from multiple origami designs, modular self-assembly has seen considerable experimental research. This section will detail some results of particular interest to the polycube model presented in Chapter 3.

2.1.1 DNA tiles

Following early nucleic acid multi-arm junctions and lattices suggested by Seeman [12], Winfree [13, 14] used double-crossover (DX) motifs, shown in Figure 2.1.a, to self-assemble 2D DNA crystals. As can be seen in figure 2.1.b, the lattices could be made with varying complexity, exemplified using either two or four different species of tiles.

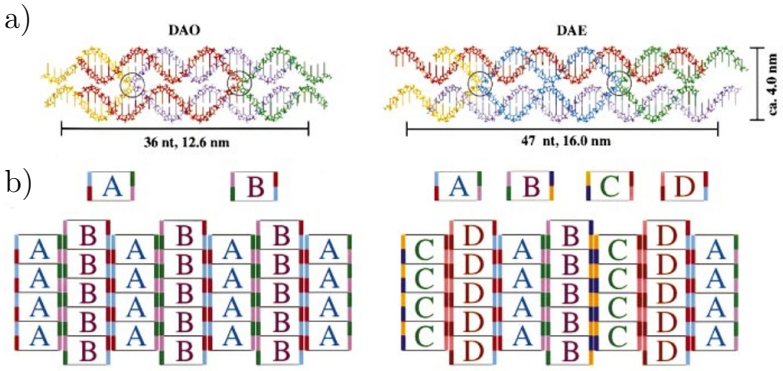


Figure 2.1: DX tiles forming 2D lattices, adapted from [14]. a) Examples of DAO and DAE tile designs. b) Lattices made from two and four tile species respectively.

2.1.2 DNA bricks

A three-dimensional DNA lattice was created in 2017 by Ong et al. [15], called DNA bricks. Structures are assembled from up to about 30,000 unique components,

as seen in Figure 2.2. Since each component consists of a unique strand, custom shapes can be "sculpted" by leaving out the voxels (3D pixels) not required.

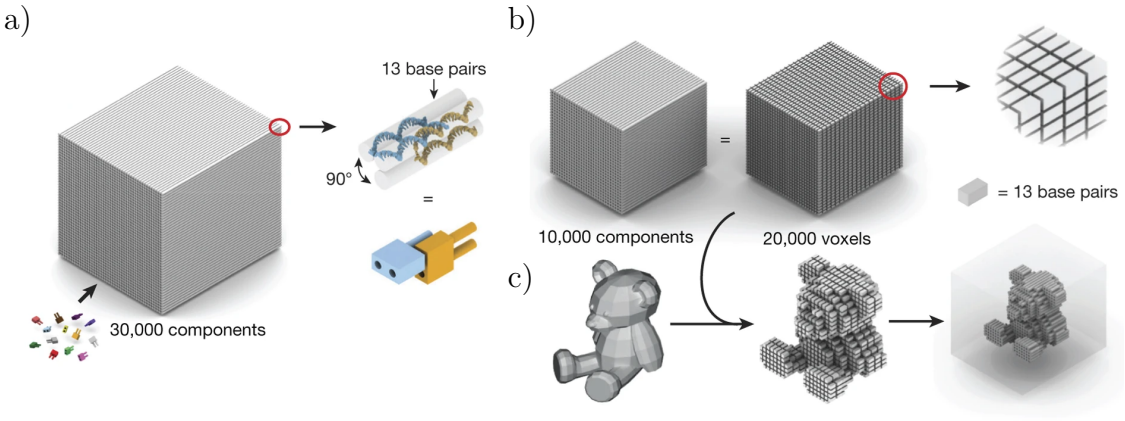


Figure 2.2: DNA bricks, adapted from [15]. a) DNA brick structure, where each of the up to 30'000 unique components is a 52 nucleotide DNA strand. The strands connect through a 13 base pair complementary domain at a 90 degree dihedral angle. b) A cuboid, here shown with 10,000 components, corresponds to a 20,000 voxel canvas. c) Approximating the shape of a teddy bear by removing a subset of the voxels from the canvas.

2.1.3 RNA tiles

As covered in Section 1.3, it is possible to co-transcriptionally fold *RNA origami* [7]. This was first shown by Geary et al. in 2014, who folded RNA tiles that connect through complementary 120-degree kissing loop interactions, as seen in Figure 2.3, forming a hexagonal lattice.

2.1.4 Finite DNA origami arrays

In 2017, Tikhomirov et al.[16, 17] demonstrated two-dimensional patterns assembled on the micrometre-scale using square origami tiles connecting through their complementary edges, as seen in Figure 2.4. The patterns could either be hierarchically assembled from unique tiles [16], or assembled into random patterns from a small number of tile types [17]. The binding stregth could be calibrated using a variable amount of overhangs on the edges, as seen in Figure 2.4.b). While the random tilings were generally unbounded, Tikhomirov et al also showed how to program a

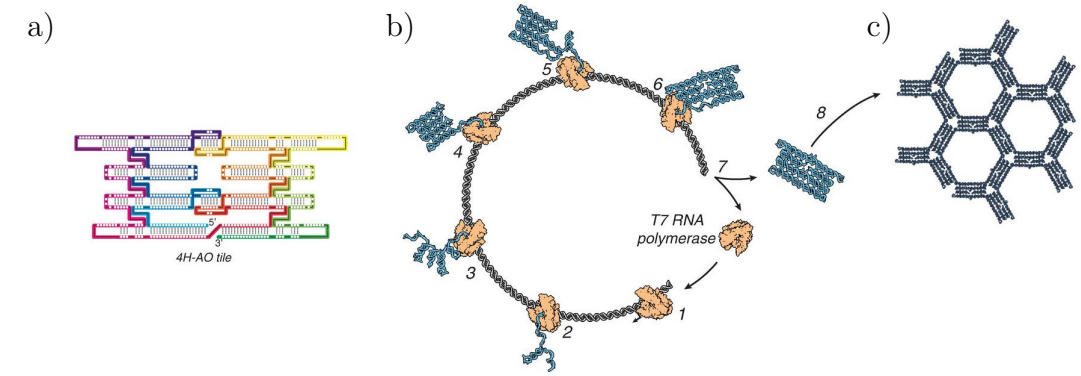


Figure 2.3: Co-transcriptional folding RNA origami tiles, adapted from [7]. The tiles connect through 120-degree kissing loop interactions, forming a hexagonal lattice. a) Detailed scematic of the four-helix 4H-AO tile. b) Co-transcriptional folding, where the RNA tile folds as it is transcribed from a DNA template. c) Hexagonal lattice formed by folded tiles.

finite grid (see Figure 2.4.d). Arrays up to 8×8 tiles were successfully produced, although larger arrays had a much smaller yield (Figure 2.4.c).

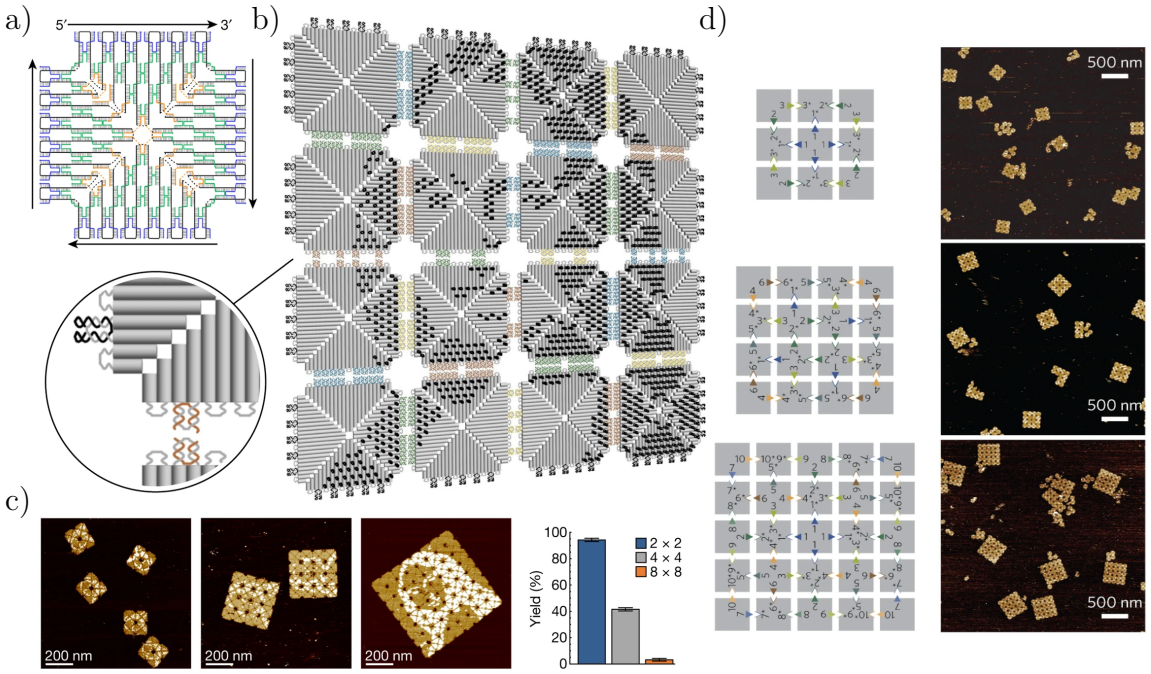


Figure 2.4: DNA origami arrays, adapted from [16, 17]. a) Strand-level diagram of a 12×12 version of the the origami tile (actual size is 22×22 helices). b) 4×4 tile "Mona Lisa" pattern. c) AFM image of patterned assemblies of different sizes (left) with their respective yields (right). d) Abstract design diagrams (left) and AFM images (right) of finite origami arrays, designed to different sizes [17].

2.1.5 Shape-complementary origami

Also, in 2017, Wagenbauer et al. [18] used shape-complementarity to assemble DNA origami components into three-dimensional polyhedral shapes up to 450 nanometers in diameter. Later, in 2021, Sigl et al. assembled large shells from shape-complementary origami triangles, as seen in Figure 2.5. The triangular sides attach through helix protrusions and indentations of complementary shape, as seen in figure 2.5.a), binding together through blunt-end stacking interactions, a method first shown by Woo et al. [19].

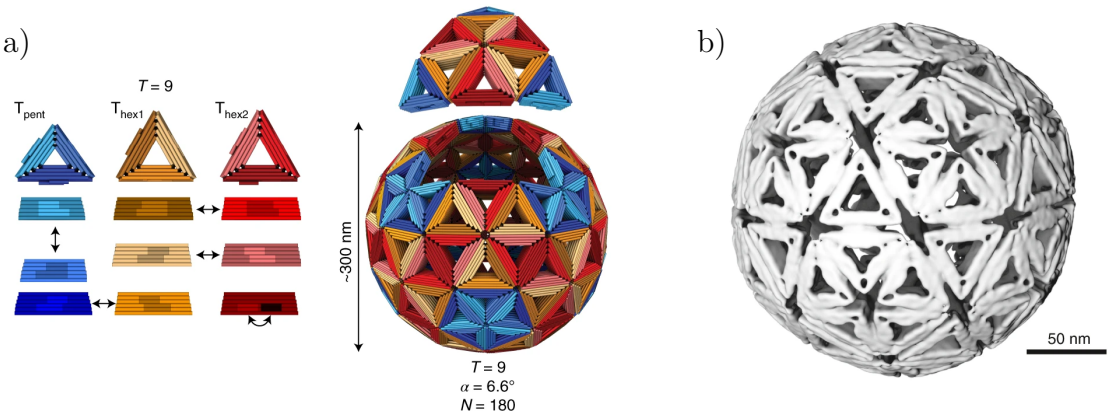


Figure 2.5: Shape-complementary triangles assembling polyhedral shells. Adapted from [20]. a) Polyhedral shell design for T=9. N is the triangulation number (the number of unique edges required for assembly), α is the bevel angle of the triangle sides, and N is the number of triangles required for a full shell. b) Cryo-EM reconstruction of the individual sub-units (top) and the assembled shell (bottom) for a T=4 icosahedral shell.

2.1.6 DNA origami nanochambers

In 2020, Lin et al.[21] presented cubic DNA origami "nanochambers". The chambers have sticky-end overhangs on every side of the cube, allowing it to assemble in 1D, 2Dm and 3D, as can be seen in Figure 2.6. However, since the shape is only rotationally symmetric around the z-axis, the assembly is a limited subset of the polycube model described in Chapter 3.

2.1.7 Octahedral DNA origami frames

In 2020, Wang et al. [22] showed how octahedral DNA origami frames could be used as building blocks in limited and unlimited programmed assemblies, as seen

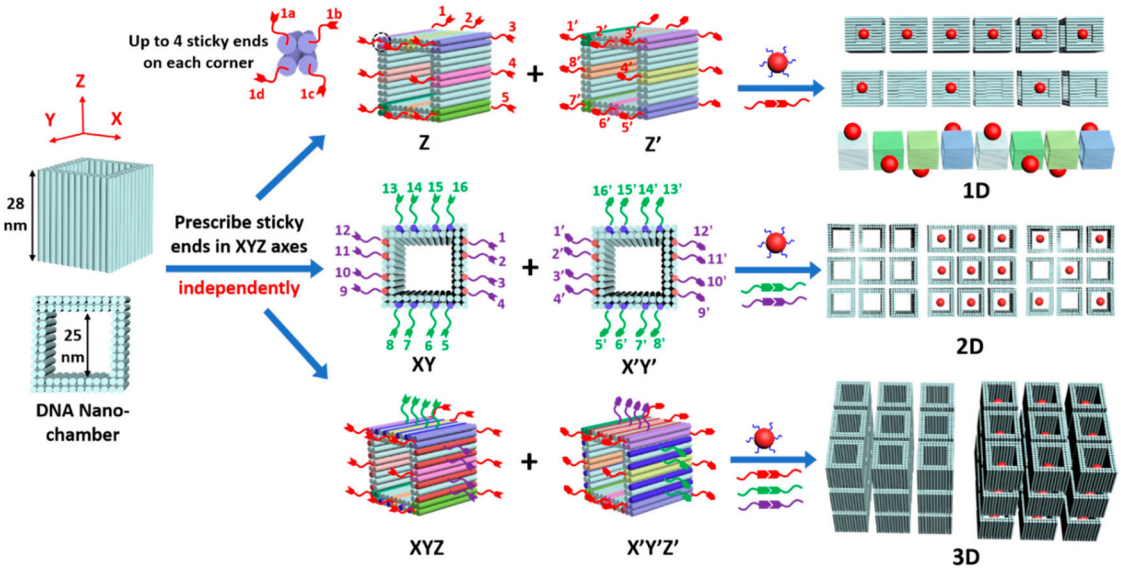


Figure 2.6: DNA origami nanochambers, adapted from [21]. a) Concept illustration, where building blocks with *polychromatic* bonds (differentiated though different single-stranded sequences), assemble into 1D, 2D, and 3D structures. b) Schematic of DNA nanochamber programmable assembly, showing sticky end overhangs applied in 1D, 2D, and 3D assemblies.

in Figure 2.7. Using sticky-end overhangs at the octahedral vertices, the building blocks have the connectivity, as well as the rotational symmetry, of a cube.

Due to the flexibility of the single-stranded connections, the connections are less torsionally rigid than assumed in the polucube model, later described in Chapter 3. However, as can be seen in Figure 2.7, shapes such as the cube can still be assembled with good yield.

2.2 Modular assembly models

It would be computationally unreasonable to simulate module assembly at the level of individual nucleotides. A simpler approach is to use an abstract model to predict the assembly process with the modules treated as rigid bodies or discrete tiles. This section will present earlier such models as a background to my own polycube model, described in Chapter 3.

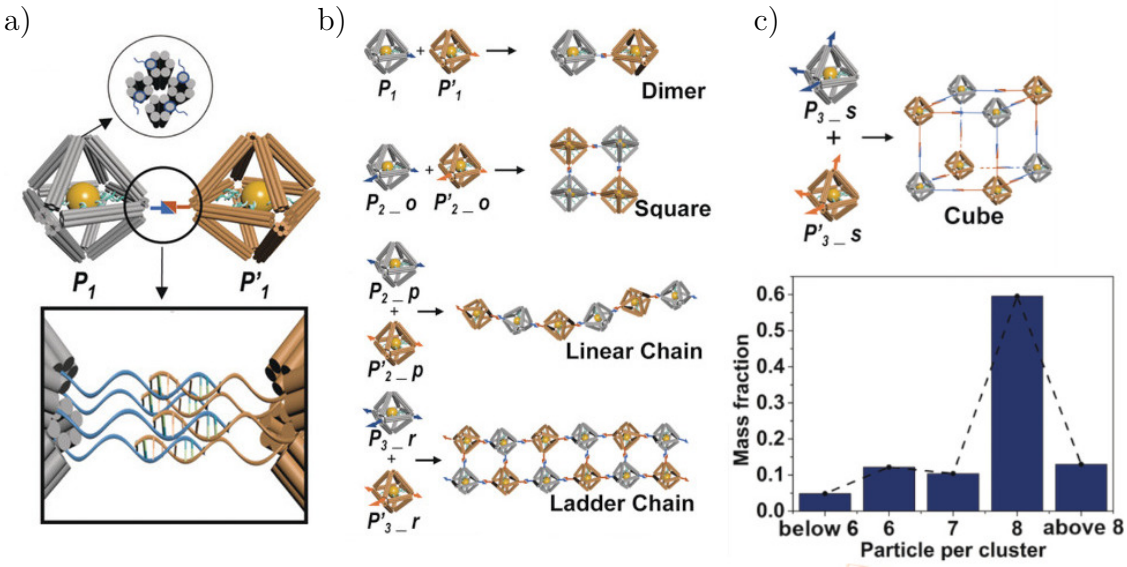


Figure 2.7: Octahedral DNA origami frames, adapted from [22]. a) Two octahedra with complementary sticky ends binding together to form a dimer. The edges consist of six-helix bundles. b) nanoclusters assembled from different sets of building blocks. c) $2 \times 2 \times 2$ cube nano cluster (top) and histogram of the mass fraction, where the intended design of eight components per cluster is the most common.

2.2.1 Wang tiles

Introduced by Hao Wang in 1961 [23], so-called *Wang tiles* are square tiles with a colour on each of their four edges. Without rotating or reflecting the tiles, they assemble so that adjacent edges have the same colour.

The DNA tiles by Winfree et al. [14], presented in Section 2.1.1, behave like Wang tiles by design and do not allow rotations or reflections. Winfree investigated the possibility of using such tiles for computation [13], which led to aTAM: the algorithmic Tile Assembly Model.

2.2.2 The algorithmic tile assembly model

The algorithmic Tile Assembly Model (aTAM), shown in Figure 2.8, models the dynamic behaviour of the DNA tiles introduced by Winfree et al. [14]. Each tile has four patches, one on each edge. Furthermore, the patches can have different strengths, with a global temperature variable determining the total connection strength required for a tile to attach [24].

In the example seen in Figure 2.8, the pattern grows from the initial bottom-right seed into the blue horizontal bottom row and the rightmost vertical column. This is because these tiles have "strength-2" glues with enough binding strength to attach by themselves [24]. The additional tiles have weaker "strength-1" glues (illustrated as thinner black connectors), so they need at least two complementary patches to achieve the binding strength treshold (temperature). Because of this so-called co-operative binding, the tiles can be seen as logic gates performing computation; given the bottom and right patches as input bits, the matching tile attaches and produces two computed output bits (top and left).

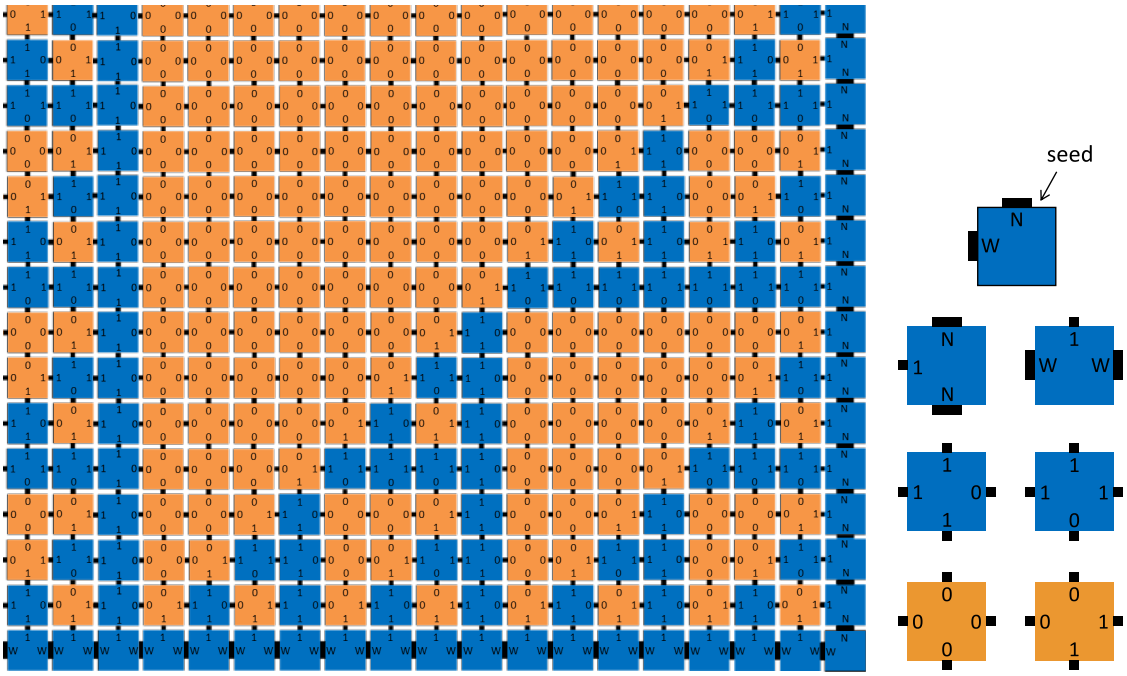


Figure 2.8: Algorithmic self-assembly of a Sierpiński triangle. Adapted from [25]. A tile set (right) grows from an initial seed by co-operatively attaching self-complementary edges (without rotation). The 0 and 0 "glues" are weaker and require two matching bonds to attach (co-operative binding), compared to the W (west) and N (north) glues that are strong enough to bind alone.

2.2.3 The polyomino model

The main inspiration for the polycube model presented in Chapter 3 is the polyomino model [26, 27]. The 2D model is similar in assembly to the later experimental micro-meter scale tile designs by Tikhomirov [17] shown in Figure 2.4.d). Compared

to the abstract tile assembly model, described in Section 2.2.2, polyomino tiles are allowed to rotate; creating further possibilities for symmetries. Also, the edge binding is not self-complementary, using complementary color pairs instead. Finally, polyominoes have a constant binding strength (in other words, it is a temperature-1 model), avoiding co-operative binding.

See Figure 2.9 for an illustration of the model, where an input *genotype* genotype (describing four possible tile types) is mapped into an assembled output polyomino by stochastically growing the shape from an initial seed. The growth stops if, as in the example in Figure 2.9, no more tiles can attach (since the color 0 does not bind to anything).

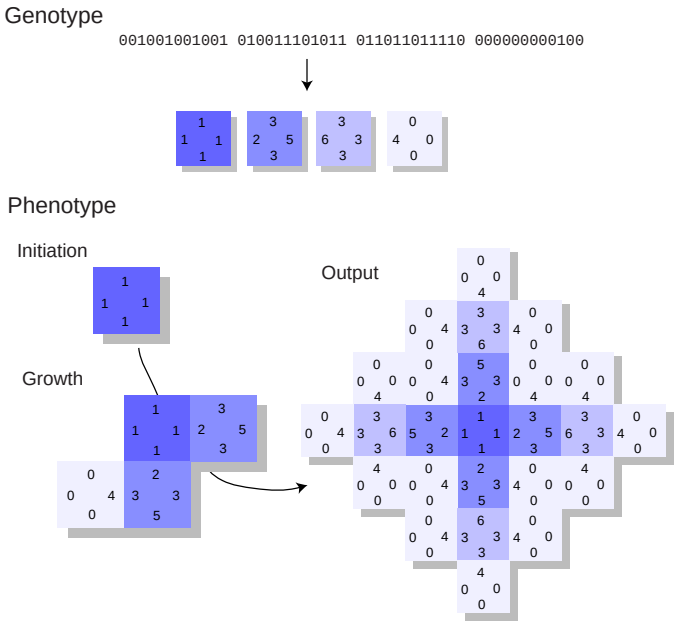


Figure 2.9: Illustration of the polyomino assembly model, adapted from [27]. A *genotype*, in the form of a ruleset of possible tiles, encodes for a polyomino *phenotype*, grown stochastically from an initial seed tile.

2.2.4 Patchy particles

3

Modular self-assembly of polycubes

Contents

As introduced in the previous chapter, there is an increasing interest within the field of DNA nanotechnology to create finite-sized multi-component objects. While some coarse-grained tile models exist, there remains a need for methods to quickly explore the assembly of multi-component 3D structures. This chapter describes my polycube model and details how it can be used to sample a large number of assembly rules, showing that some polycube shapes are significantly more common than others. The following chapter (Chapter 4) will show how to obtain the simplest assembly rule for any given polycube shape.

The model supports both 3D polycubes and 2D polyominoes, both explained in the following sections.

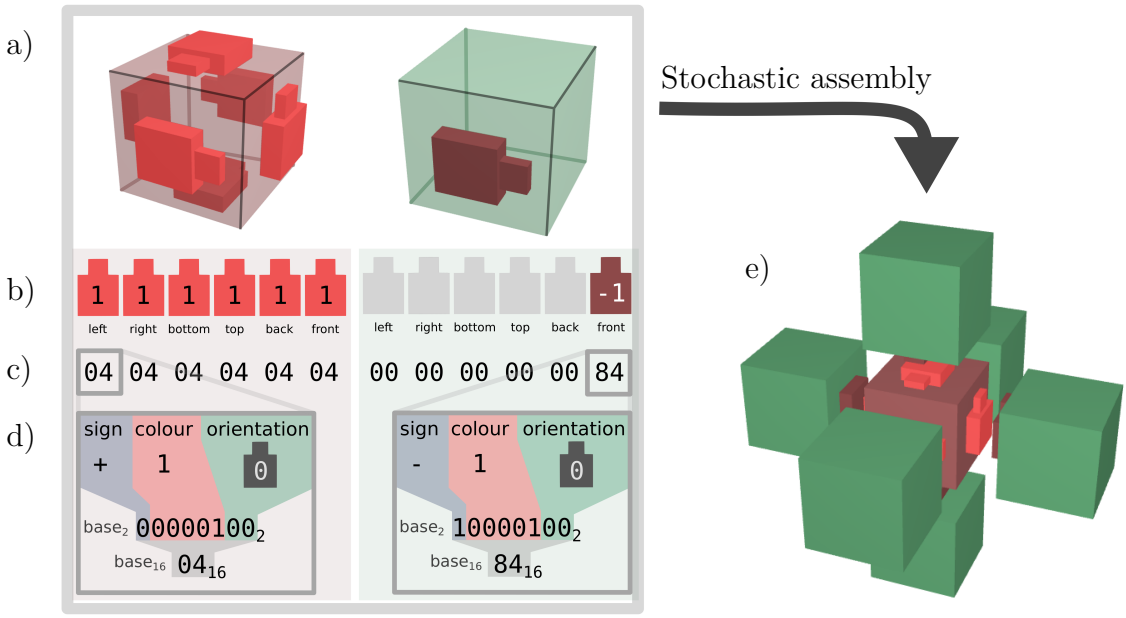


Figure 3.1: Illustration of the polycube model and notation, examplified with the rule 040404040404000000000084. Compare this to the polyomino model in Figure 2.9. **a)** 3D representation of the species in the rule. **b)** Rule depicted as a list of the patches in each species. The empty patches (colour 0) in the green species are just shown with their orientations. All orientations are 0 in this rule, since changing them would not change the output. **c)** Hexadecimal representation of the rule, shown decoded in **d)**, where every 2-digit hexadecimal number represents a patch. Converted to a 8-bit binary number, first bit encodes the sign, the next five bits the colour ($[0, 31]$), and the final two bits encode the orientation ($[0, 3]$). **e)** Fully assembled polycube output. The assembly used one copy of the first species (red) and six copies of the second (green). The assembly finished since no further cubes could be added.

3.1 The polycube model

A *polycube* consists of multiple equally-sized cubes connected by their neighbouring faces (a three-dimensional analogue to how polyominoes are squares connected by their neighbouring edges). In the model presented here, a polycube is stochastically self-assembled according to a specified rule, defining a set of available cube species. Each species describes a type of cube that can be present in the polycube, so cubes belonging to the same species are always identical. See, for example, Figure 3.1, where an input rule with two species assembles into a double-cross output polycubes.

Each species has six patches; one on each face of the cube, and each patch has

a "colour" and an orientation. The colour is indicated by a signed integer and the orientation is one of four possible rotations:  (0),  ($\frac{\pi}{2}$),  (π) or  ($\frac{3\pi}{2}$). A patch can bind to another patch if and only if they have the opposite colour and the same (global) orientation. In Figure 3.1, the patch color 1 is shown as bright red while the complementary -1 is a darker red colour. If the patch color is zero, the patch is shown as empty and will not bind to anything.

The model can be expanded to more complicated colour interaction matrices, or changed to pair odd integers with each subsequent integer as in the polyomino model[26, 27] described in Section 2.2.3.

By constraining the input space not to use the back and front patches, while orienting the remaining patches to point towards the front, the output space will instead be 2D polyominoes.

Polycube rules can be described in a hexadecimal representation, as seen in the top left of Figure 3.1. Each

The stochastic self-assembly of a polycube starts by placing a cube from one of the available species as a seed at the origin. If the assembly mode is *seeded*, it will always use the first species in of rule. If the assembly mode is *stochastic*, the seed species is instead chosen at random. Once a cube has been added to the assembly, all available neighbouring positions are added to a list of possible moves.

Moves are then processed until the list of moves is empty, or the polycube grows beyond a specified size, at which point it is considered unbounded. Figure 3.2.a) shows an example of an unbounded structure tiling the plane using two species.

While the list of moves is not empty, a random move is choosen at each step. The input rule is then randomly searched for a species fitting the move. Cubes can be rotated to fit, and if a fitting species is found, the corresponding cube is added to the assembly. If there is no fit to be found, the move is discarded.

To determine if the rule is deterministic, the assembly is repeated n_{times} times (default 100) and the outputs are compared for equality (allowing rotation). Figure 3.2.b) shows a non-deterministic structure, where the blue "neck" species can bind to itself. Thus, the output depends on how many cubes from the blue species bind

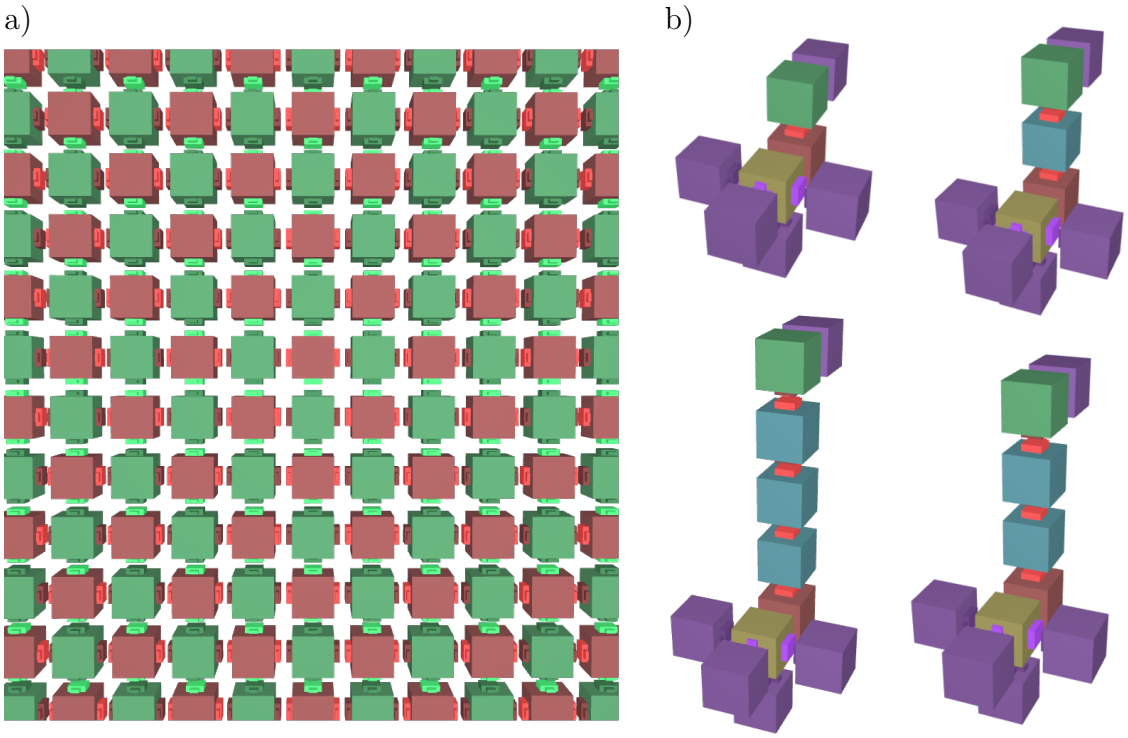


Figure 3.2: Examples of undefined assemblies. **a)** Unbounded assembly that tiles the plane using two species (05050a08000085858a880000), **b)** An undeterminisic assembly of a "giraffe duck" with a neck that can have a different length each time it is assembled (00000006008b00008600000c000000028c00080c0c000c0c048600000000).

before a green species cube stops the growth. Both species are as likely to bind, so the probability of assembling a neck with l blue cubes is 2^{-l}.

It could be argued, instead of first picking a random move and then randomly trying all available species to find a fit, that one should pick both a move and a species at random until a fit is found. While this would take a longer time, it would avoid biasing the assembly toward unlikely assembly results, where a move is picked that would otherwise usually be blocked by more likely surrounding cubes. However, since only deterministic and bounded rules are of interest, this would only affect the end result in the cases where the bias is strong enough and n_{times} is low enough to incorrectly make the rule seem deterministic.

For an illustration of the model, let us return to Figure 3.1. The example in the figure is a three-dimensional "cross" structure created from a rule of size 2. The initial seeding cube belongs to the first species, enabling six additional cubes, all belonging to the second species, to bind at each patch. The patches bind since their colours, 1

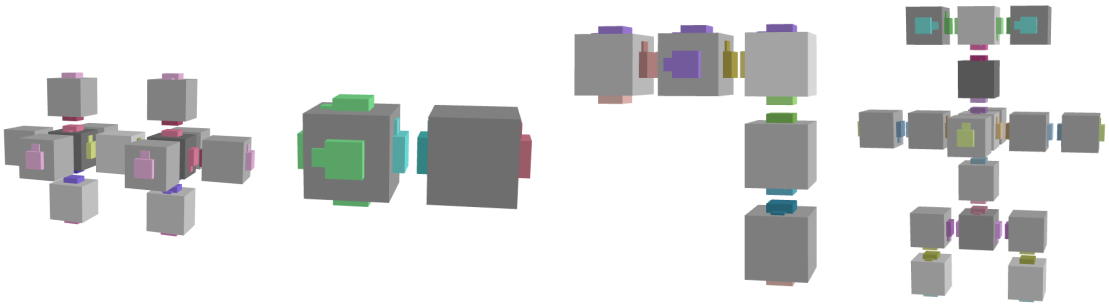


Figure 3.3: Polycube versions of the conceptual DNA Robotics designs. From left to right, the size of the rule required to specify each polyomino is: 4, 2, 5 and 11. Note how the third polyomino (L-shaped) requires a larger rule than the first (double cross), although it is smaller in size. This is because each cube in the L-shaped nanobot needs to be unique, while the double-cross-shaped first nanobot consists of two identical parts. If we only wished to reproduce the polycube shape, the rulesets could be minimised further.

and -1, are opposites. After all six outer cubes have bound, there are no remaining possible moves, and thus the polycube stops growing. Since the growth stops, this particular polycube is bounded at a size of seven cubes. Furthermore, since the rule gives the same polycube every time it is evaluated, the polycube is deterministic.

The polycube assembly model is implemented in two versions: one browser-based implementation in JavaScript for outreach activities and accessible visualisation, and one C++ implementation for fast rule evaluation. The C++ code also includes a Python binding for simplified analysis. More details on the code can be found in Appendix A.

3.2 Sampling the space of assembly rules

Trying all inputs in a brute-force approach is impossible for. For a 3D polycube space with n_s species and n_c colours, four possible patch orientations, and six patches per species, we get:

$$I^3_{n_c,n_s} = (4 \times (1 + 2n_c))^{6n_s}$$

Even for a relatively small values of $n_s = 3$ species and $n_c = 2$ colours, we get $I^3_{2,3} \approx 2.62 \times 10^{23}$.

For 2D it is a bit easier, since there are only four patches per species and a fixed patch orientation:

$$I^2_{n_c,n_s} = (1 + 2n_c)^{4n_s}$$

But even if the space of possible input rules is too large to explore fully, we can still get an idea of how likely it is for an input to map to a certain output thorugh uniform sampling.

This was done by sampling and assembling one billion random rules from $I_{31,8}$. Rules growing larger than 100 cubes were discarded as unbounded, while those remaining bounded were re-assembled 15 times to ensure they assembled deterministically. Deterministic and bounded output was then grouped by their shapes, counting the number of times each given phenotype occurs. For each rule found to produce a given phenotype, the number of colours is multiplied by the number of species in the rule, producing a measure of the rule size. The smallest such rule size is then used as a proxy for the phenotype complexity.

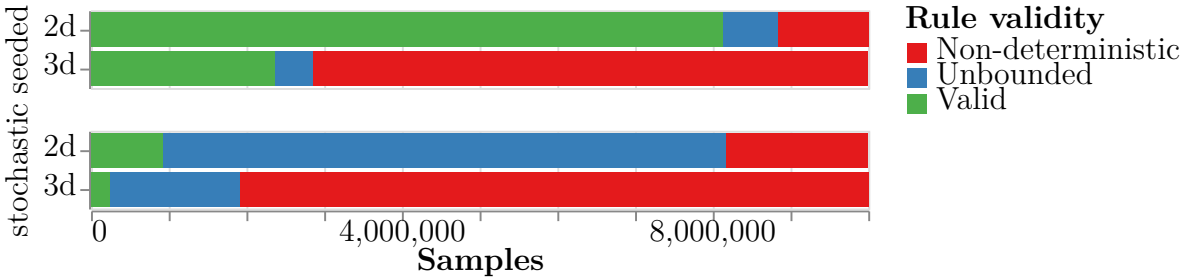


Figure 3.4: Proportion of valid rules in different sampling datasets

3.3 Complexity bias

As can be seen in Figure 3.6, there is a clear log-linear relationship between the probability of finding a rule that assembles into a particular structure and the information needed to specify the structure, as predicted in [11, 28].

Stochastic sampling tends to have a constant high number of cube types. Since any of the cube types can be used as a seed, they all need to grow into the same structure every time. Thus they all need to be included.

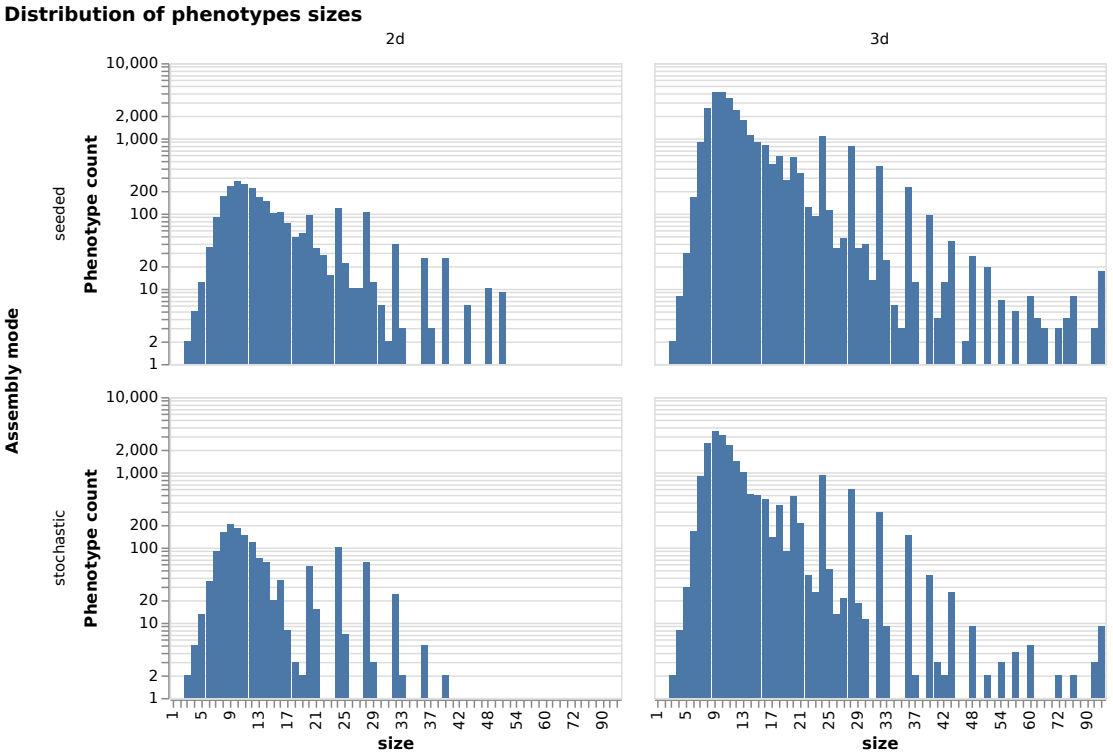


Figure 3.5: Distribution of phenotype sizes.

3.4 Modularity

Modularity index, polycube size divided by number of species, plot vs. frequency (and vs complexity)

3.5 Symmetry

For n most common 16-mers, how many of them are symmetrical?

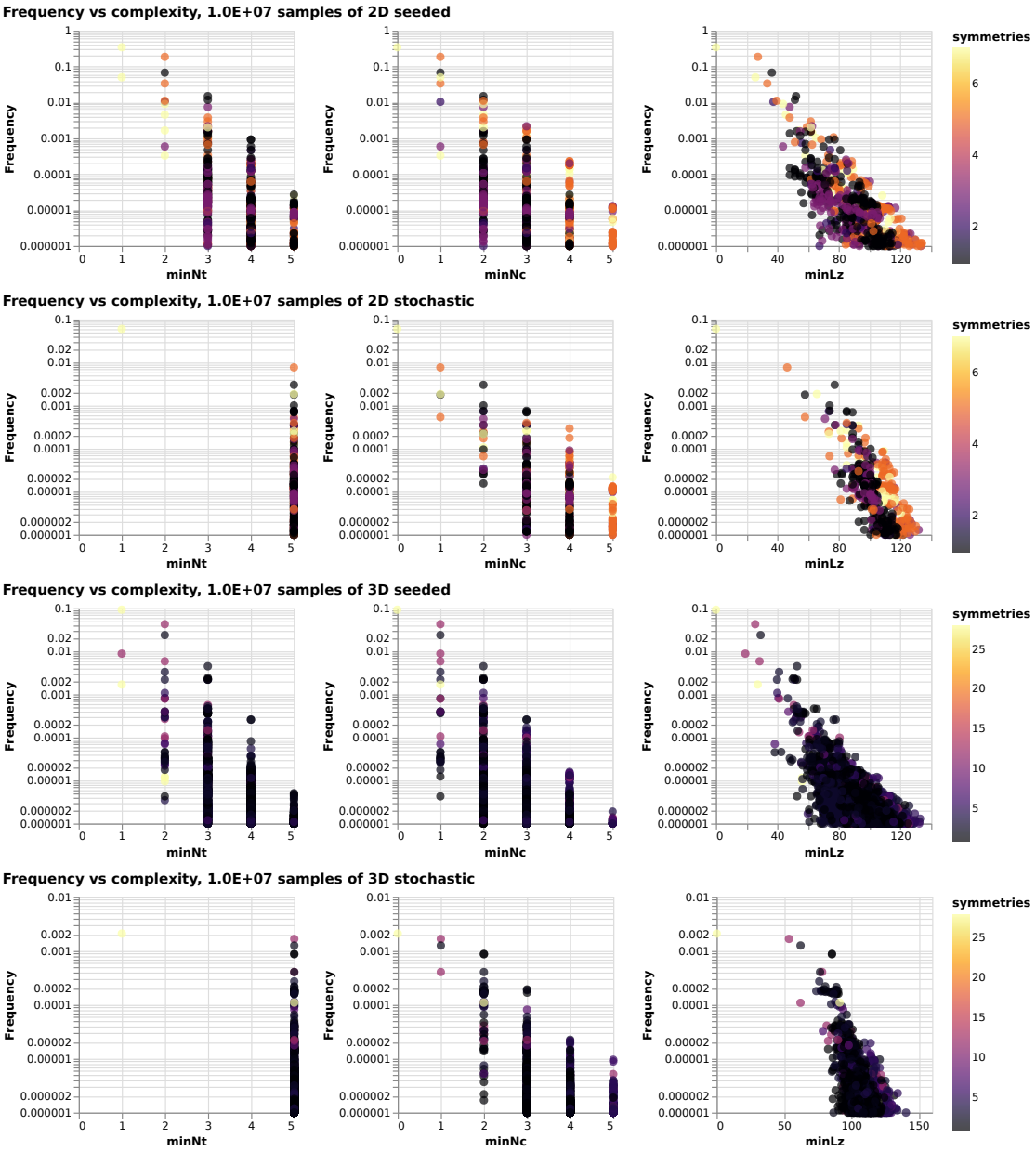


Figure 3.6: Frequency vs complexity

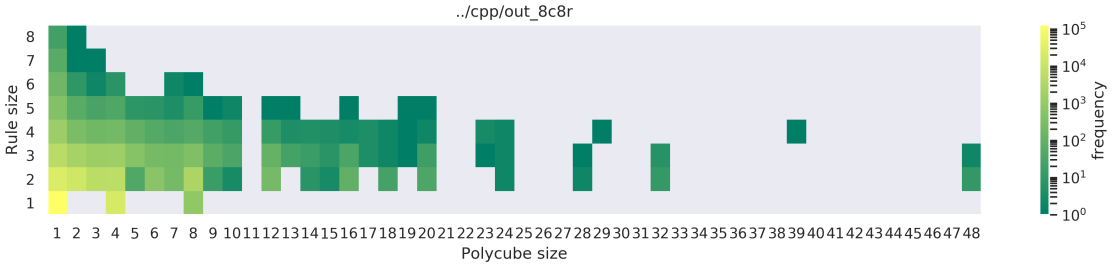


Figure 3.7: Heat map of rule size vs polycube size, for a simulation with a maximum of 8 species and a limit of 8 colours, evaluating 2.5 million random rules. Note that the frequency scale is logarithmic.

Figure 3.8: Example of polycubes grown from 1, 2 and 3 species. Although any polycube can be encoded into a rule, the larger polycubes that have small rules tend to be symmetrical. This agrees with Johnston's polyomino model in that they have low complexity

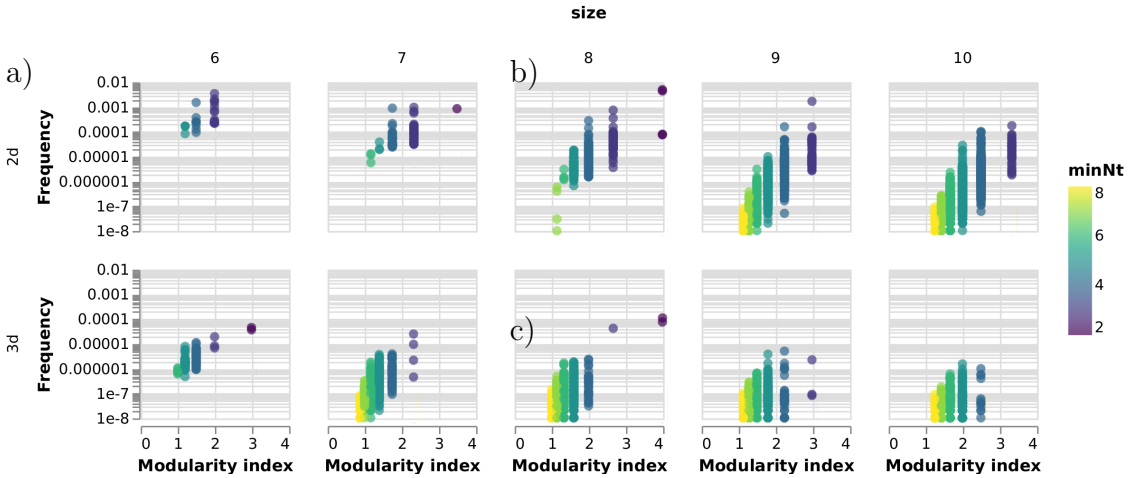


Figure 3.9: Frequency vs modularity for a selection of polycube and polyomino sizes. From 1e8 samples of $n_s = 8$, $n_c = 31$

4

Designing polycube assembly rules

Contents

In the previous chapter, we saw how to map an input rule into an output polycube shape. However, the reverse problem is just as significant; given a target shape, how do you find a rule that assembles it?

A trivial solution would be to use fully addressable assembly, simply assign a unique species to each cube, and a unique colour to each pair of adjacent patches. This is similar to what was done for DNA bricks (Section 2.1.1). However, as was seen in Chapter 3, many shapes have alternative solutions requiring significantly fewer unique components. See Figure 4.1, where a square tetromino is shown to have a range of possible inputs assembling it, from the minimal solution with just a single species and one colour, up to the fully adressable solution with four species and four colours.

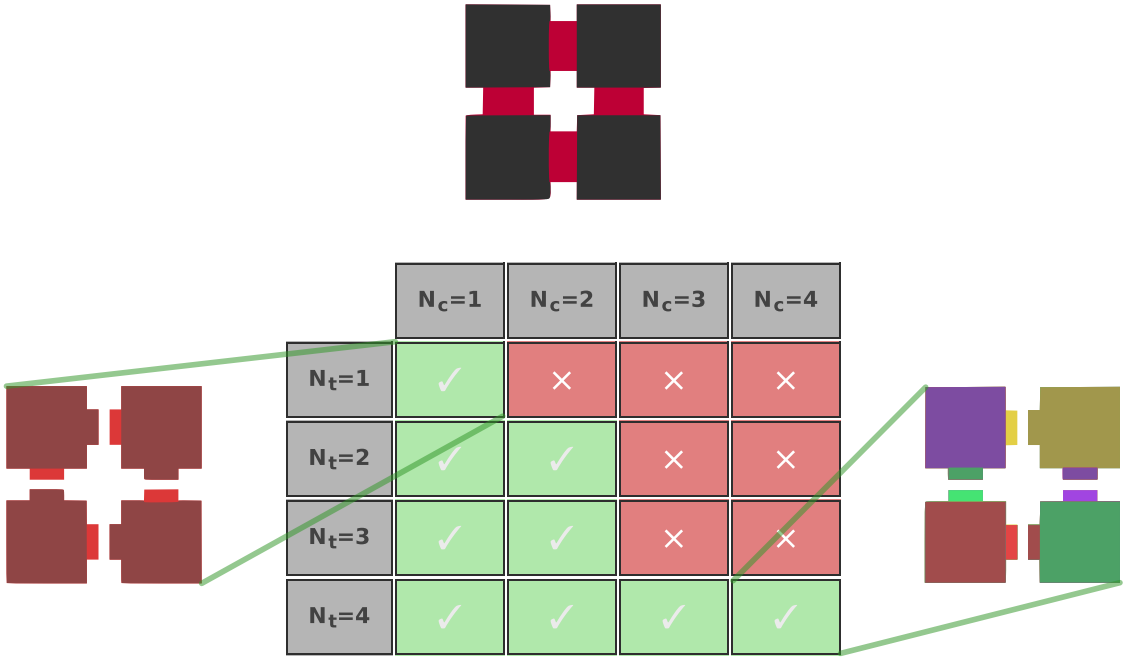


Figure 4.1: 2x2 polyomino assembled with different levels of complexity.

So, is there a better way to find these simpler input rules than to sample the space of all rules? This chapter presents an approach where satisfiability solving is used to determine if a shape can be assembled from a given number of colours and species. Furthermore, a complementary approach of substituting similar species is also described.

4.1 Satisfiability solving

Building upon a method for determining patchy particle interactions for unbounded structures [29], it is possible to formulate and solve satisfiability problems for the bounded polycube structures.

In essence, we formulate a boolean expression that, if true, means it is possible to assemble a given polycube topology using a given number of colours and species. We can then use a satisfiability solver to check if the expression is indeed solvable and, if it is, extract an assembly rule from the solution.

4.1.1 Boolean expressions

The boolean expression is written in conjunctive normal form (CNF), where variables are composed into clauses using NOT ($\neg$) and OR ($\vee$) operators and where the clauses are joined by AND ($\wedge$) operators. As a simple example, see the expression below:

$$(\neg x_{rain} \vee x_{umbrella} \vee \neg x_{walk}) \wedge (\neg x_{rainbow} \vee x_{rain}) \wedge (\neg x_{rainbow} \vee x_{sunny})$$

The first clause is $true$ for all values except when $x_{rain} = true$, $x_{umbrella} = false$ and $x_{walk} = true$, so the solution of taking a walk in the rain without an umbrella is forbidden (allthough I would prefer a good coat). This could also be written as an *implication*: $x_{rain} \wedge x_{walk} \implies x_{umbrella}$.

The following two clauses are the CNF form of $(x_{rainbow} \implies x_{rain} \wedge x_{sunshine})$, stating that a rainbow implies that we have both rain and sunshine. We cannot have a rainbow without rain or without sunshine. The full expression is satisfiable, for example, if we set $x_{sun} = true$,$x_{rain} = false$, $x_{walk} = false$, $x_{umbrella} = false$, and $x_{rainbow} = false$; ignoring the sunshine and remaining inside to write.

4.1.2 Polycube formulation

For the polycube problem, we introduce the following variables:

$x^A_{l,p,o}$ (patch p at position l has orientation o)

$x^B_{c_i,c_j}$ (colour c_i is compatible with colour c_j)

$x^C_{s,p,c}$ (patch p on cube type s has colour c)

$x^D_{p_1,o_1,p_2,o_2}$ (patch p_1 with orientation o_1 binds to patch p_2 with orientation o_2)

$x^F_{l,p,c}$ (patch p at position l has colour c)

$x^O_{s,p,o}$ (patch p on cube type s has orientation o)

$x^P_{l,s,r}$ (position l is occupied by cube type s rotated by r)]

We then formulate clauses to constrain the problem, seen in Table 4.1. Clauses (i)-(vii) are the same as in [29] while the remaining are added, together with variables x^D, x^A and x^O, to include torsional restrictions, meaning that patches need to bind at a compatible orientation (complared to being allowed to rotate freely).

	Clause	Boolean expression
(i)	$C^B_{c_i,c_j,c_k}$	$\neg x^B_{c_i,c_j} \vee \neg x^B_{c_i,c_k}$
(ii)	C^C_{s,p,c_k,c_l}	$\neg x^C_{s,p,c_k} \vee \neg x^C_{s,p,c_l}$
(iii)	C^P_{l,s_i,r_i,s_j,r_j}	$\neg x^P_{l,s_i,r_i} \vee \neg x^P_{l,s_j,r_j}$
(iv)	$C^{BF}_{l_i,p_i,c_i,l_j,p_j,c_j}$	$\left(x^F_{l_i,p_i,c_i} \wedge x^F_{l_j,p_j,c_j}\right) \Rightarrow x^B_{c_i,c_j}$
(v)	$C^{rotC}_{l,s,r,p,c}$	$x^P_{l,s,r} \Rightarrow \left(x^F_{l,p,c} \Leftrightarrow x^C_{s,\phi_r(p),c}\right)$
(vi)	C^{alls}_{s}	$\bigvee_{\forall l,r} x^P_{l,s,r}$
(vii)	C^{allc}_{c}	$\bigvee_{\forall s,p} x^C_{s,p,c}$
(iix)	C^O_{s,p,o_k,o_l}	$\neg x^O_{s,p,o_k} \vee \neg x^O_{s,p,o_l}$
(ix)	$C^{DA}_{l_i,p_i,c_i,l_j,p_j,c_j}$	$\left(x^A_{l_i,p_i,c_i} \wedge x^A_{l_j,p_j,c_j}\right) \Rightarrow x^D_{p_i,c_i,p_j,c_j}$
(x)	$C^{rotO}_{l,s,r,p,o}$	$x^P_{l,s,r} \Rightarrow \left(x^A_{l,p,o} \Leftrightarrow x^O_{s,\phi_r(p),o}\right)$

Table 4.1: SAT clauses. (i) Each colour is compatible with *exactly one* colour. (ii) Each patch has *exactly one* colour. (iii) Each lattice position contains a single cube type with an assigned rotation. (iv) Adjacent patches in the lattice must have compatible colours. (v) Patches at a lattice position are coloured according to the (rotated) occupying cube type. (vi) All N_t cube types are required in the solution. (vii) All N_c patch colours are required in the solution. (iix) Each patch is assigned *exactly one* orientation. (ix) Adjacent patches in the target lattice must have the same orientation. (v) Patches at a lattice position are oriented according to the (rotated) occupying cube type.

4.1.3 Bounded structures

An important difference to [29] is that the method presented here allows for bounded structures. This is done by adding species of type “empty” as a "shell" around the shape to ensure that empty patches remain unbound. Adding a clause $x^B_{0,1}$ ensures that colour 0 always binds to 1.

We then add clauses $x^F_{l,p,1}$ to constrain every boundary patch p at lattice position l to have the colour 1 and thereby not bind anything else. For the example in Figure 4.2, these boundary patches are seen coloured white, so there we get:

$$
\begin{aligned}
&x^F_{0,0,1} \wedge x^F_{0,1,1} \wedge x^F_{0,3,1} \wedge \\
&x^F_{1,0,1} \wedge x^F_{1,2,1} \wedge x^F_{1,3,1} \wedge \\
&x^F_{3,0,1} \wedge x^F_{3,1,1} \wedge x^F_{3,2,1} \wedge \\
&x^F_{4,1,1} \wedge x^F_{4,2,1} \wedge x^F_{4,3,1}
\end{aligned} \tag{4.1}
$$

Note that for 3D polucubes, there are six patches per species, instead of the four seen in the 2D polyomino in Figure 4.2. This also introduces 27 possible cube rotations, compared to the 4 square rotations defined for 2D.

The topology of the shape is enforced by clause (iv), $\left(\neg x^F_{l_1,p_1,c_1} \vee \neg x^F_{l_2,p_2,c_2} \vee x^B_{c_1,c_2}\right)$ in CNF, making sure that if patch p_1 on lattice position l_1 binds to p_2 on lattice position l_2, their colours are compatible. Similarly, clause (ix) ensures that the patches have the same orientation.

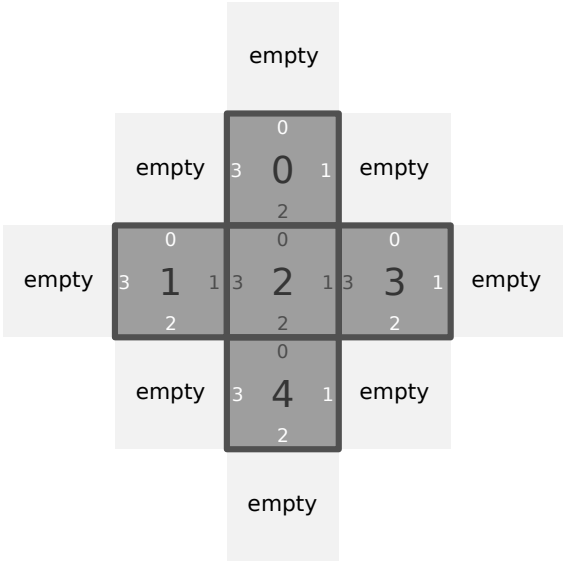


Figure 4.2: Bounded shape topology for satisfiability solving. Patches at the boundary of the shape (white) are constrained to only bind to "empty". 3D shapes are specified the same way, but with six patches per species.

Interaction matrix is fixed (compared to [29]).

Figure of topology graph.

Add giraffe-duck-like example of when SAT solver gives UND solution

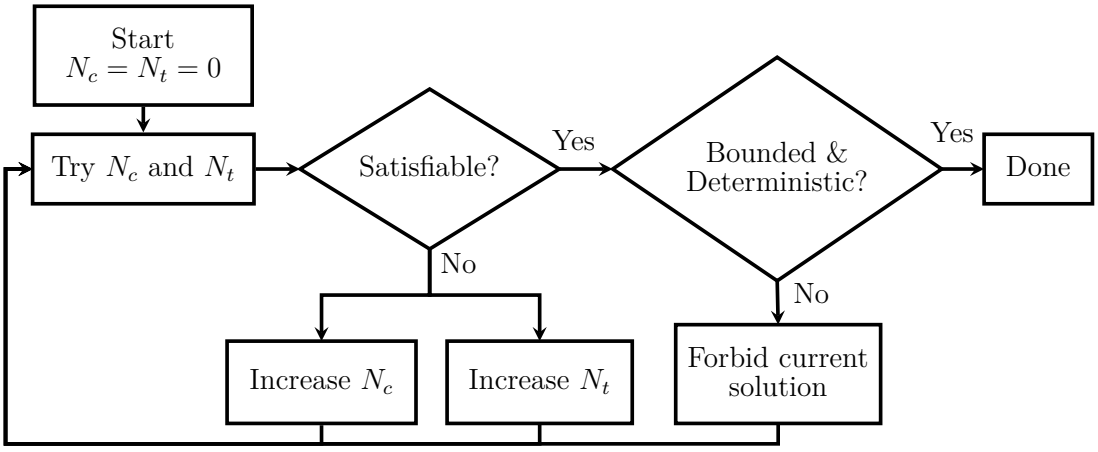


Figure 4.3: Algorithm for finding the minimal solution using SAT. Even if a solution is found to be satisfiable it might not assemble correctly every time. Additional solutions for a given N_c and N_t are found by explicitly forbidding the current solution. Alternatively, it is possible to use a solver like relsat to obtain multiple solutions.

4.2 Finding the minimal assembly rule

The method presented here uses a boolean satisfiability (SAT) solver to determines if a provided polycube shape is satisfiable for a given number of cube types N_t and colours N_c. By iteratively ruling out lower values of N_t and N_c, a minimal solution can be found, as detailed in Figure 4.3. It is also possible to generate and compare alternative solutions of varying complexity.

4.3 Simplification by substitution

An alternative approach is to substitute species in the input that are similar, removing duplicates. Starting from a fully addressable solution, any pair of species s_1 and s_2 with the same configuration of patches are tested. If we can remove s_1 and replace the patches complementary to it with ones complementary to the patches of s_2 and still get the correct output, we have successfully simplified the rule. This continues until all pairs have been tried.

4.4 Example solves

4.5 Patchy particle simulation

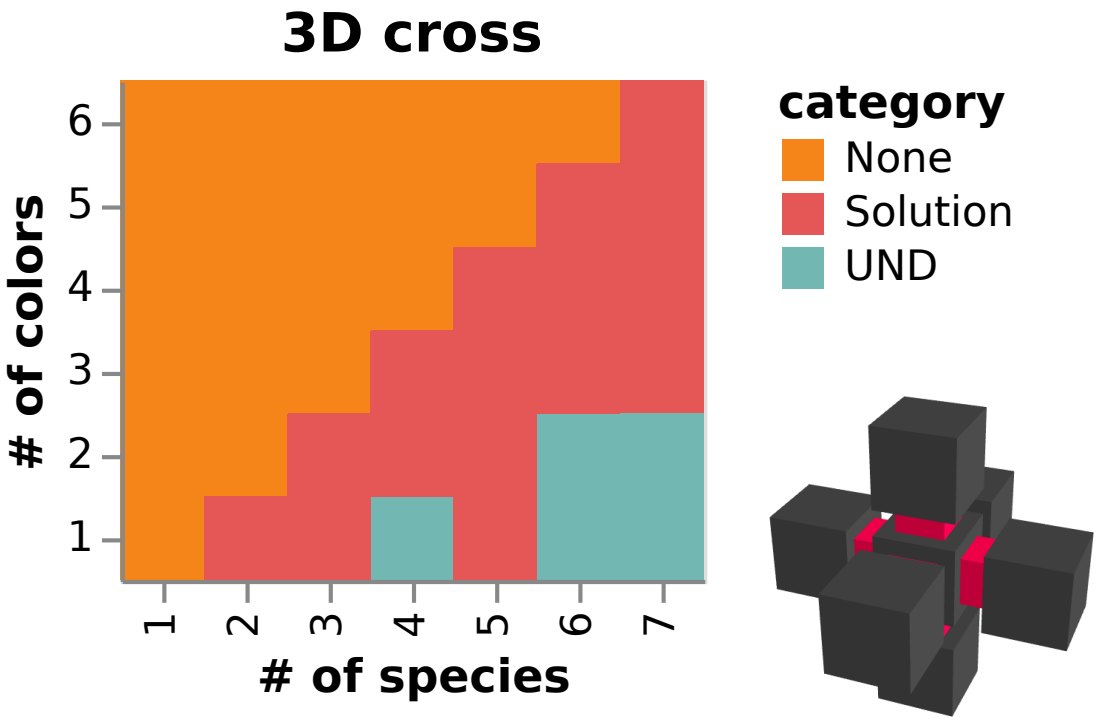


Figure 4.4: 3D-cross

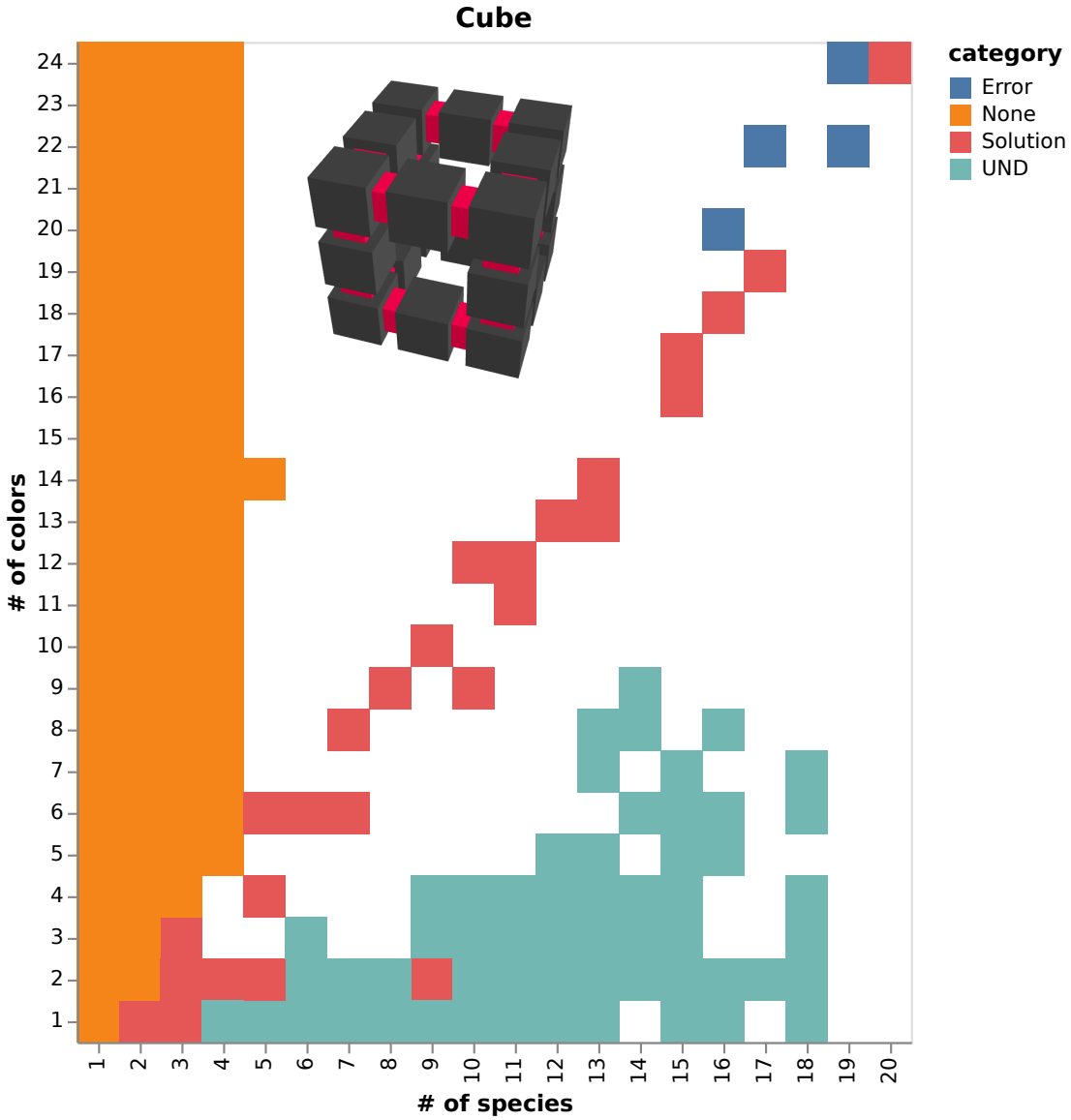
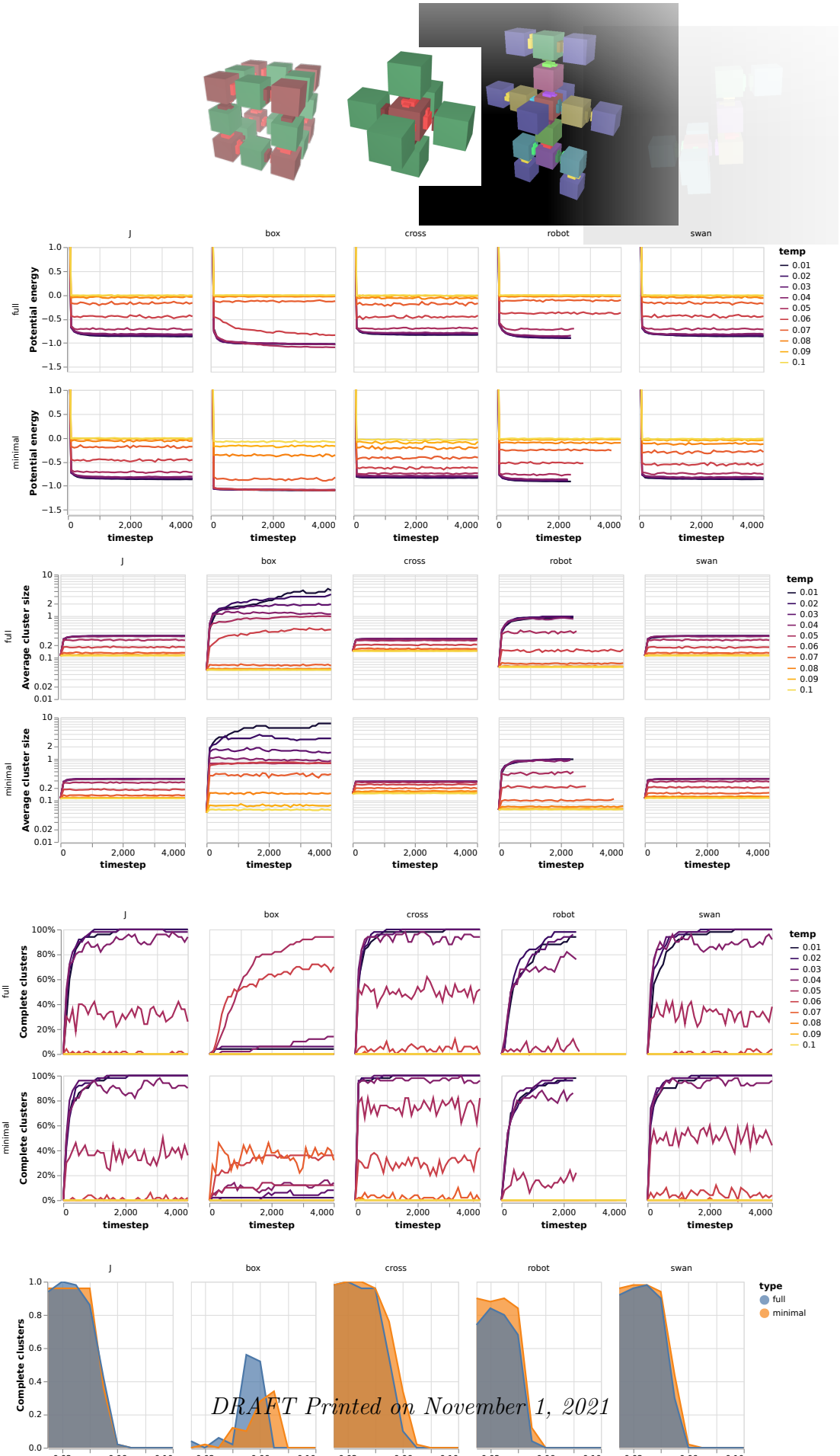


Figure 4.5: Cube


J
box
cross
robot
swan
full
minimal
Potential energy
Average cluster size
Complete clusters
timestep
temp
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
type
full
minimal

A process cannot be understood by stopping it. Understanding must move with the flow of the process, must join it and flow with it. (First Law of Mentat)

— Frank Herbert, Dune

5

An introduction to tools for the design and simulation of nucleic acid structures

Contents

While previous chapters have covered modular self-assembly on a very abstract level, approximating the modules as simple cubes or patchy particles, this chapter will introduce tools and methods for designing and simulating individual structures or modules folded using DNA (or RNA).

The following sections will cover a selection of practical design and simulation tools that have been developed over the years, providing context for the presentation of my contributions to the *oxView* tool in Chapter 6.

5.1 Design tools

Designing a DNA origami structure by hand would be very laborious for anything but the most simple design. As such, a host of computer-aided design tools have been introduced over the years to make things easier. This section will cover some of the more common examples.

5.1.1 Lattice-based design tools

The caDNAno design tool [5] and the web-based scadnano[30] it inspired, allows the user to design DNA origami on a lattice of parallel helices.

caDNAno

CaDNAno [5] was introduced in 2009 as a way to simplify 2D and 3D DNA origami designs. It has a graphical user interface with multiple panels, seen in Figure 5.1. In the slice panel, the designer can place virtual helices on a lattice (either hexagonal or square), seen in the leftmost panel of the figure. The helices can then be filled in with strands and connected using crossovers in a path panel, seen in the middle of the figure.

Finally, caDNAno is also available as a plugin to the Autodesk Maya software, which enables a 3D visualisation of the design as seen in the render panel to the right in Figure 5.1. However, caDNAno does not support Maya versions after 2015 [31].

Scadnano

Scadnano [30] (scriptable caDNAno) is a relatively new design tool, independent from but inspired by caDNAno version 2. The main difference is that Scadnano is entirely web-based (thus not requiring any installation). The python code base is also designed to make it easier to write scripts generating DNA designs. Scadnano can be found at `https://scadnano.org/`.

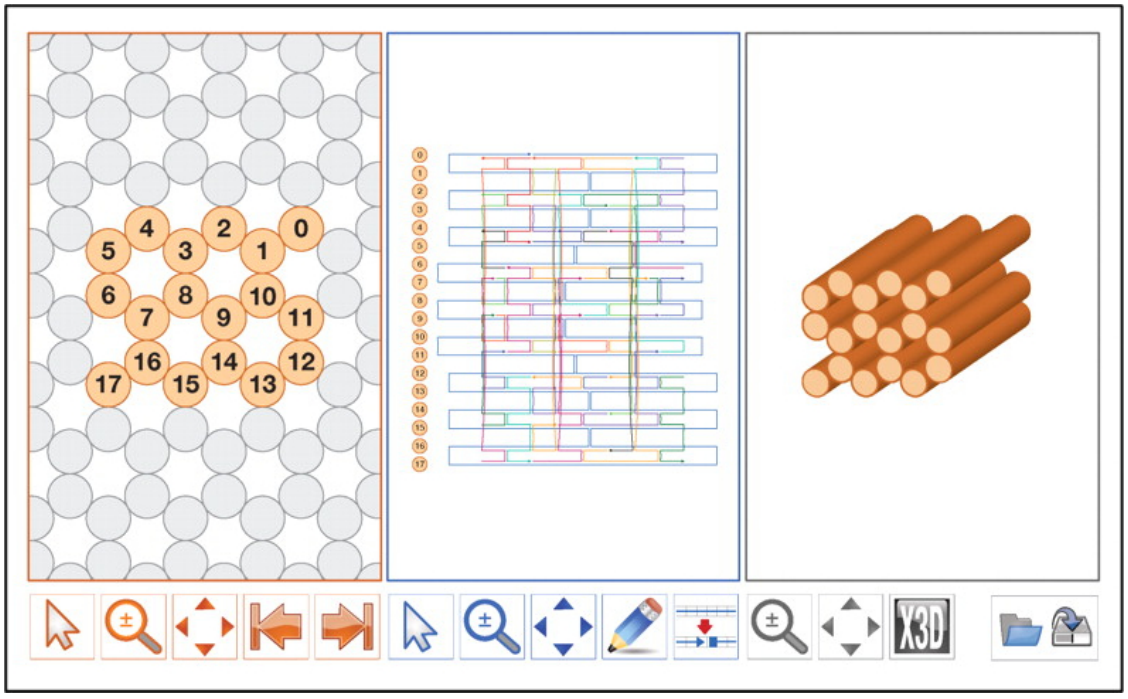


Figure 5.1: The caDNAno design interface, adapted from figure 1.(a) in [5]

5.1.2 Top-down shape converters

While tools like caDNAno simplify bottoms-up design, where the user builds structures from individual strands and nucleotides, a top-down tool can take a polyhedral target shape as input and provide a suitable origami design as output.

BSCOR

In 2015, Benson et al. published a method for converting arbitrary mesh designs into a DNA origami mesh [32]. Figure 5.2 shows a set of example polyhedral shapes, with the designed shape in a), the output DNA design in b), and microscopy characterisations in c)-d). A follow-up paper in 2016 also introduced the ability to design flat-sheet meshes [33]. BSCOR uses single DNA duplex edges, with double edges added whenever topologically necessary.

ATHENA

ATHENA is a recently published tool for automatic design wireframe origami shapes. As seen in Figure 5.3, earlier software such as PERDIX, METIS, DAEDAULS and

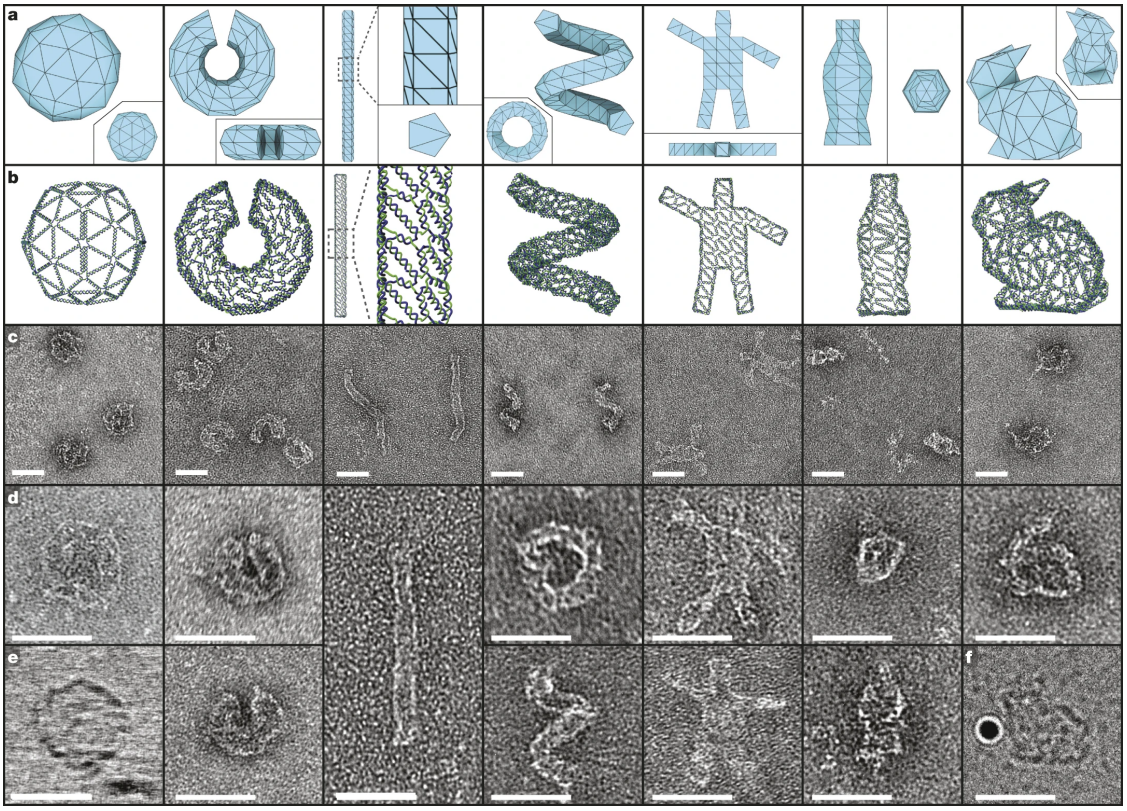


Figure 5.2: 3D meshes rendered in DNA origami using BSCOR. Adapted from [32].

TALOS have facilitated design for 2D and 3D wireframes respectively, using different edge designs, but ATHENA aims to bring them all toghether as a single package.

Triangulated truss structures

5.1.3 Free-form or hybrid tools

Tiamat

Tiamat is an early free-form design tool introduced in 2009.

vHelix

Adenita

Adenita [35] is a free-form editing tool developed as a plugin to the SAMSON toolkitb

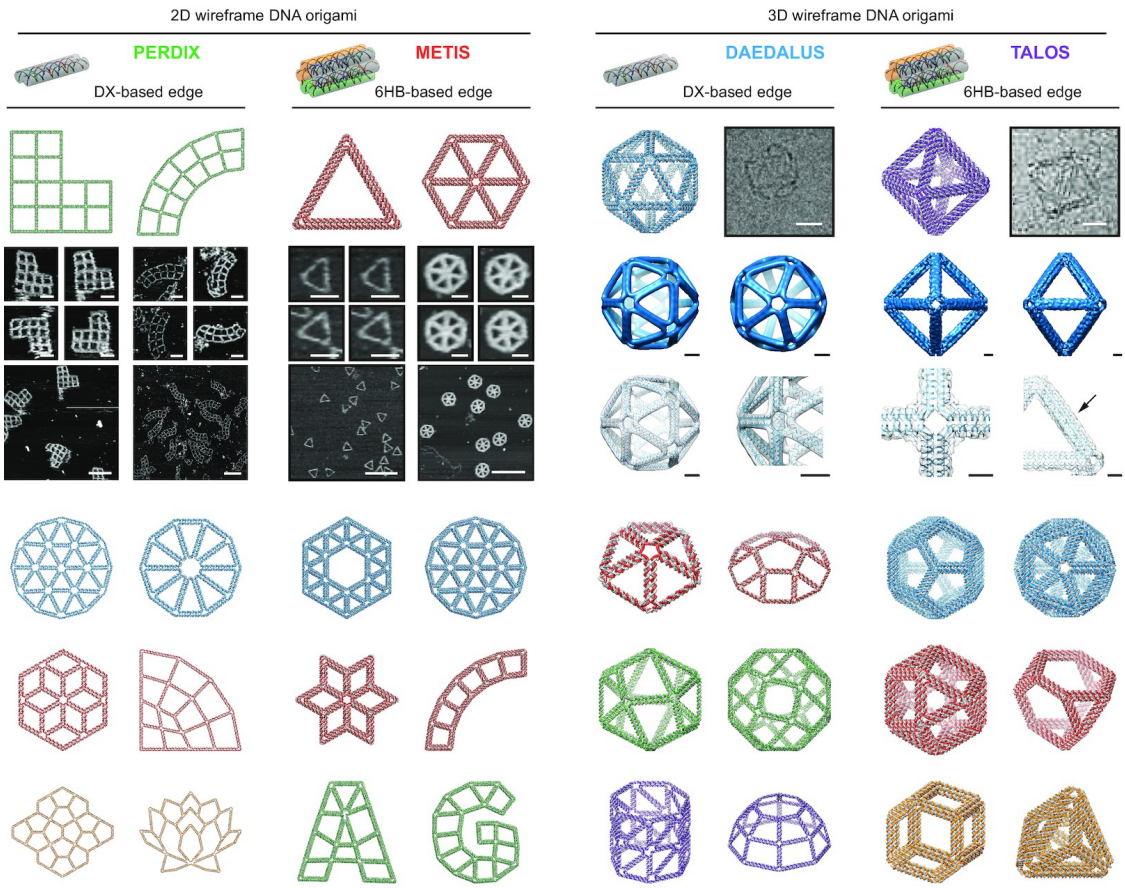


Figure 5.3: Automatic wireframe origami shapes using ATHENA. Adapted from [34].

MagicDNA

Another method of solving topological issues through rigid-body manipulation was used by Chao-Min Huang [36]. MagicDNA, as seen in Figure 5.7, has an computer-aided design workflow where geometry can be specified from helix cross-sections or imported from a part library, then assembled into an integrated structure (using multiple scaffolds if neccesary).

MagicDNA is written as a plugin to the Matlab suite.

oxView

The oxView application was developed as part of this thesis project and will be described more in Chapter 6.

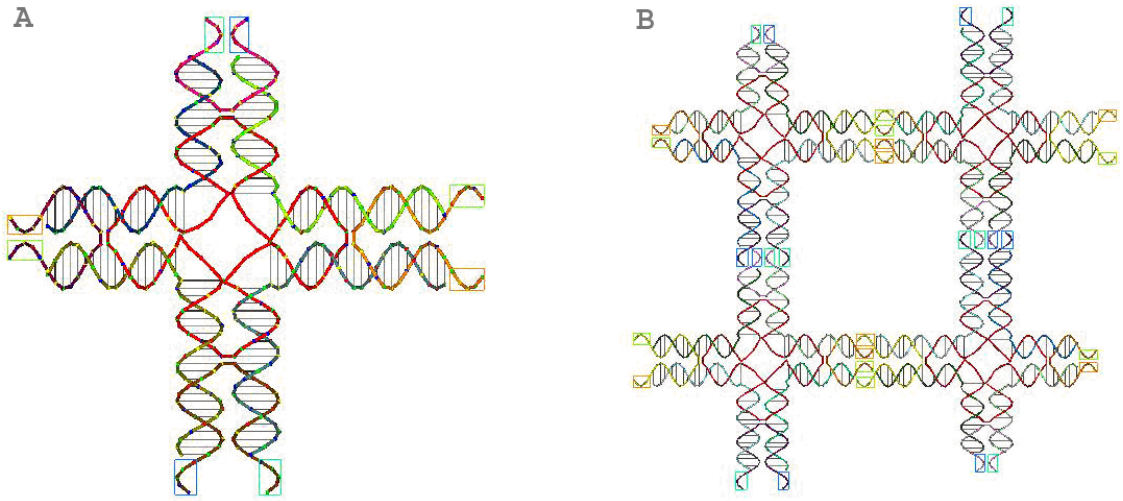


Figure 5.4: Tiamat

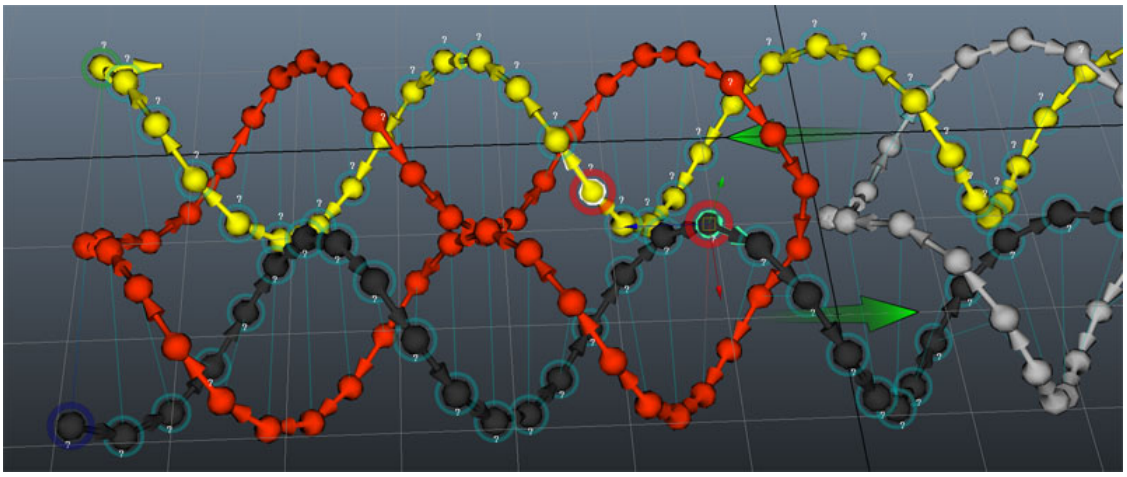


Figure 5.5: vHelix

5.2 Simulation tools

Simulating a structure can provide insight to understand experimental results but can also guide decisions at the design stage. Different models

5.2.1 All-atom simulation

Simulation tools such as NAMD[37], use force fields such as AMBER[38] and CHARMM [39] that model interactions between individual atoms. While it is possible to perform atomistic simulations of large DNA origami structures[40],

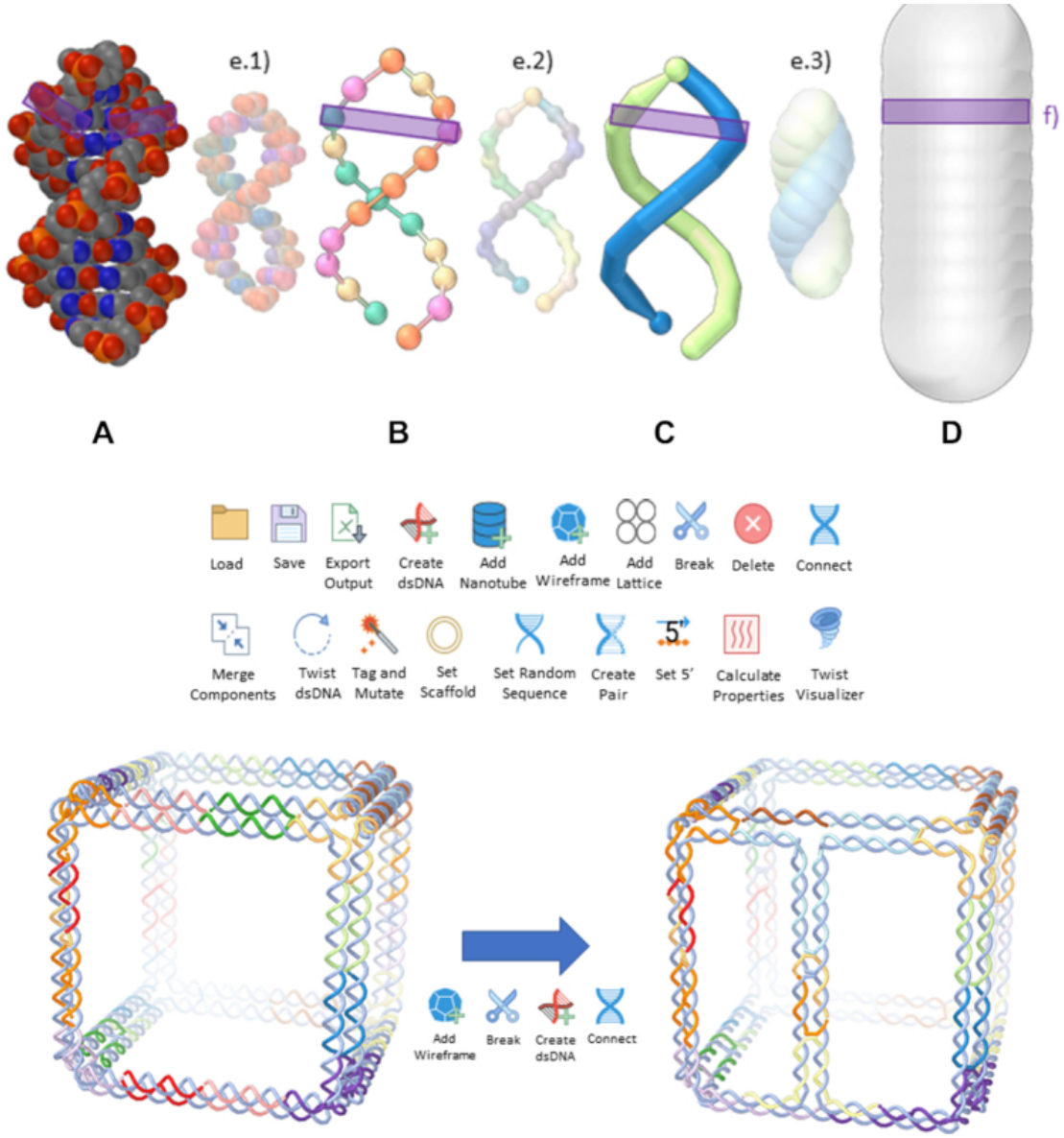


Figure 5.6: Adenita

the simulations take a long time to run, and it is unknown how well the models represent DNA thermodynamics [41].

5.2.2 oxDNA/RNA

In 2010, a coarse-grained simulation software called oxDNA was introduced by Thomas Ouldridge [4]. It simulates DNA on the level of nucleotides and has been shown to model complex origami devices with a generally good agreement with

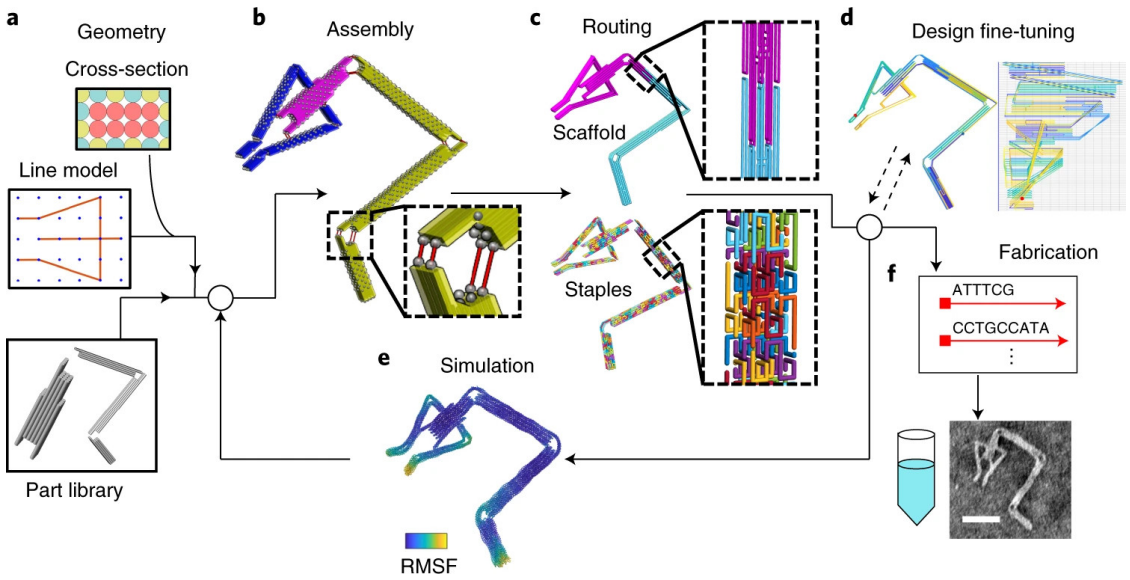


Figure 5.7: MagicDNA adapted from [36].

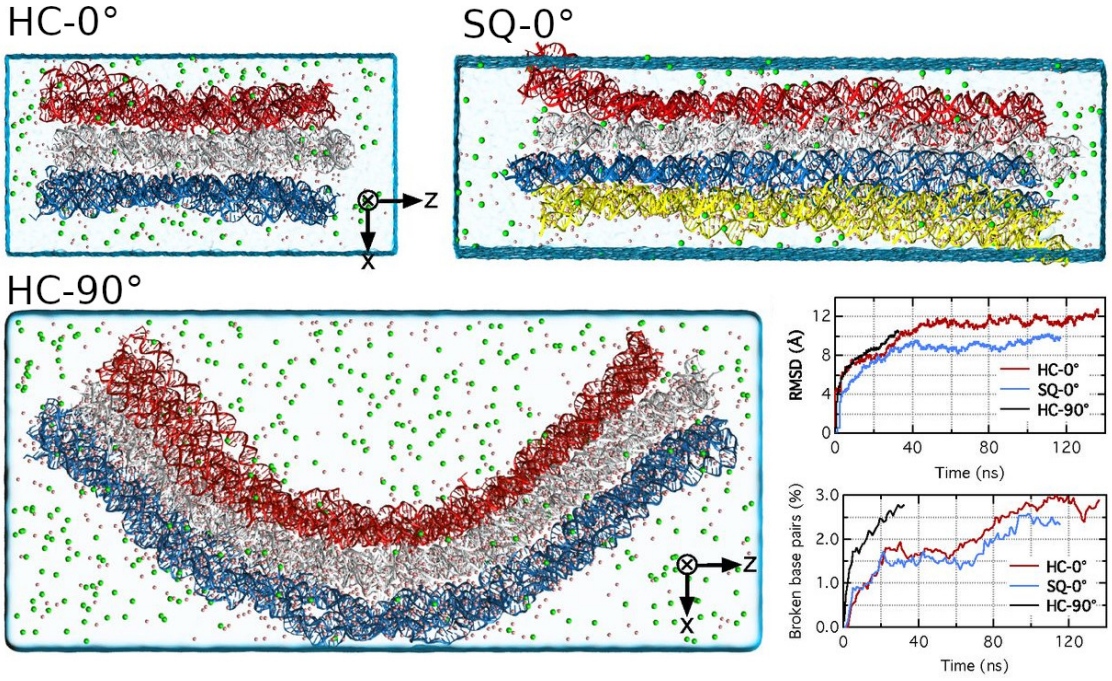


Figure 5.8: All-atom simulation

experimental data [42]. In 2014, the DNA model was extended to include RNA by Petr Šulc [43], showing its ability to model a set of common RNA motifs.

Molecular Dynamics (MD) and Monte Carlo (MC) simulation techiniques.

OxDNA was joined by cogli1 for trajectory visualisation.

While oxDNA can be very useful for modelling a structure, it has traditionally not been very accessible for experimentalists.

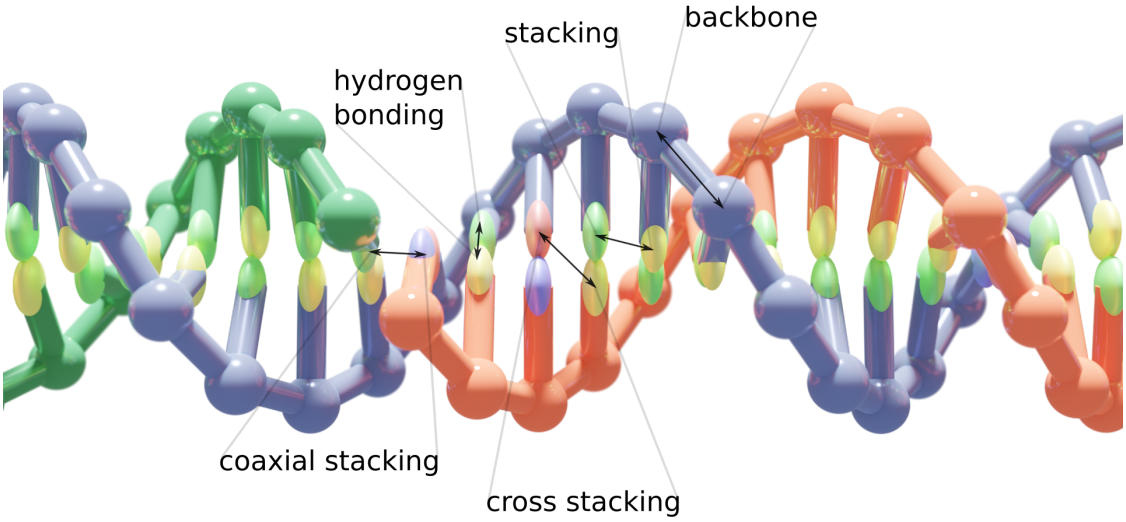


Figure 5.9: The oxDNA model

5.2.3 mrDNA

Another promising feature of mrdna is that it has a spline-based helix representation and is implemented in python. This enables users to easily script edits to the structure, translating and rotating parts of it before starting the simulation. Thus, topological issues or over-stretched bonds can, with some skill, be resolved even before starting to simulate. I did, based on this, also create a rudimentary interactive editor interface to mrdna, but it would need a lot of refinement to be externally usable.

A selection of the DNA designs I have relaxed are shown in Figure 5.11. The first two examples, adopted from [44] and [45] and illustrated in Figure 5.11.a and 5.11.b, are both quite straightforward to relax in oxDNA, although the relaxation is much faster using mrdna.

The tensegrity kite structure, adopted from [46] is harder to relax since, as seen in the first image in Figure 5.11.c, the two helix bundles are drawn parallel

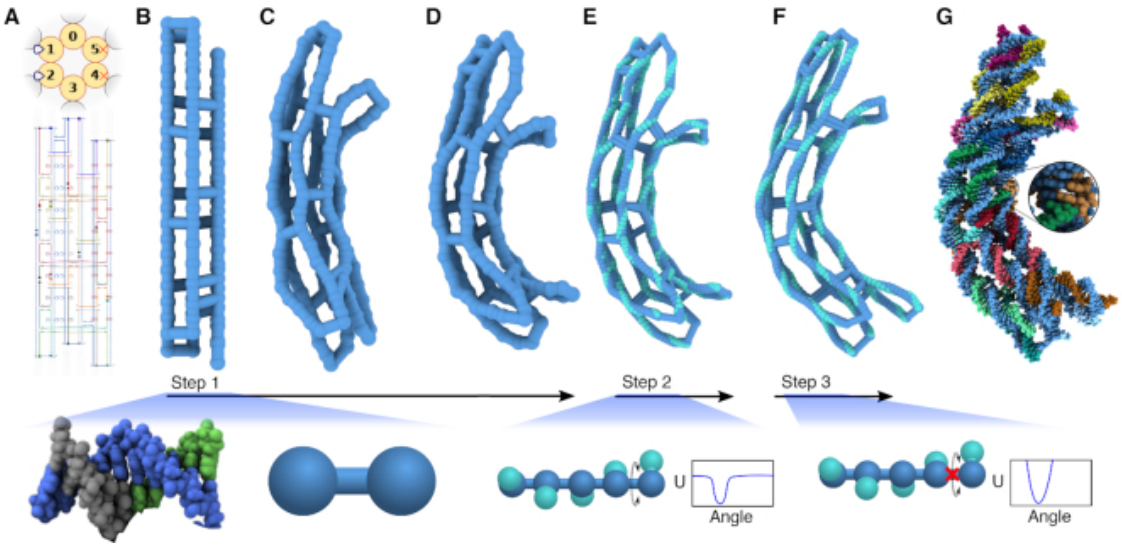


Figure 5.10: MrDNA

to each other in caDNAno. Given enough time to relax, they should still become orthogonal, but a much more efficient way is to write a mrdna script to rotate one helix bundle so that it is orthogonal from the start, as seen in the middle image of 5.11.c. The remaining overstretched bonds are then quickly relaxed using mrdna.

Finally, the Möbius strip, adopted from [47], is particularly tricky to relax, since the caDNAno design have all helices drawn in the same plane, with bonds from each end stretching through the whole structure and intersecting at a single point, as can be seen in the first image of Figure 5.11.d. With some help from Chris Maffeo, however, I was able to use a mrdna script to edit the structure into a configuration much easier to relax, as seen in the second image of Figure 5.11.d. Since the caDNAno design does not make it clear if the Möbius strip should be left-handed or right-handed, this is also decided in the script; changing the rotational direction will produce a mirrored version of the structure, as seen in the third image of Figure 5.11.d.

5.2.4 Cando

Cando is a finite element modelling framework[48] available through a web server at `https://cando-dna-origami.org`. DNA double helices are modelled as elastic

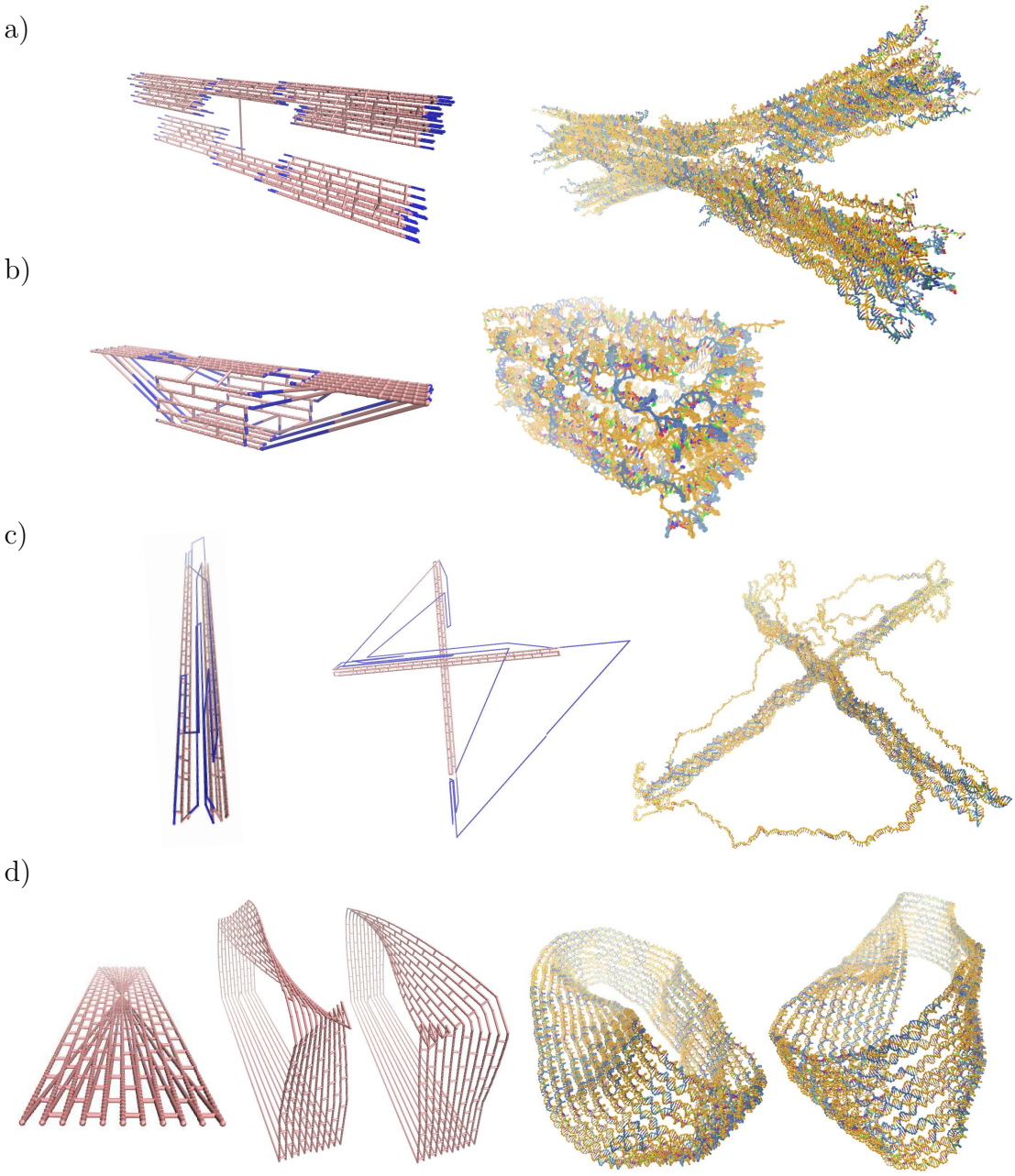


Figure 5.11: Relaxation results for various DNA designs. Each row depicts a new design, with the left-hand side showing the structure as it was drawn in caDNAno (and parsed by mrdna), while the right-hand side is the relaxed structure in oxDNA. Intermediate images are edits done in mrdna. While the switch design[44] in **a)** and the small DNA origami box[45] in **b)** relaxed without any required editing, the tensegrity kite structure [46] in **c)** and the Möbius strip[47] in **d)** benefited greatly from moving selected helices to a position off the lattice before starting the simulation.

rods (connected by rigid crossovers) that stretch, twist and bend in line with experimental measurements.

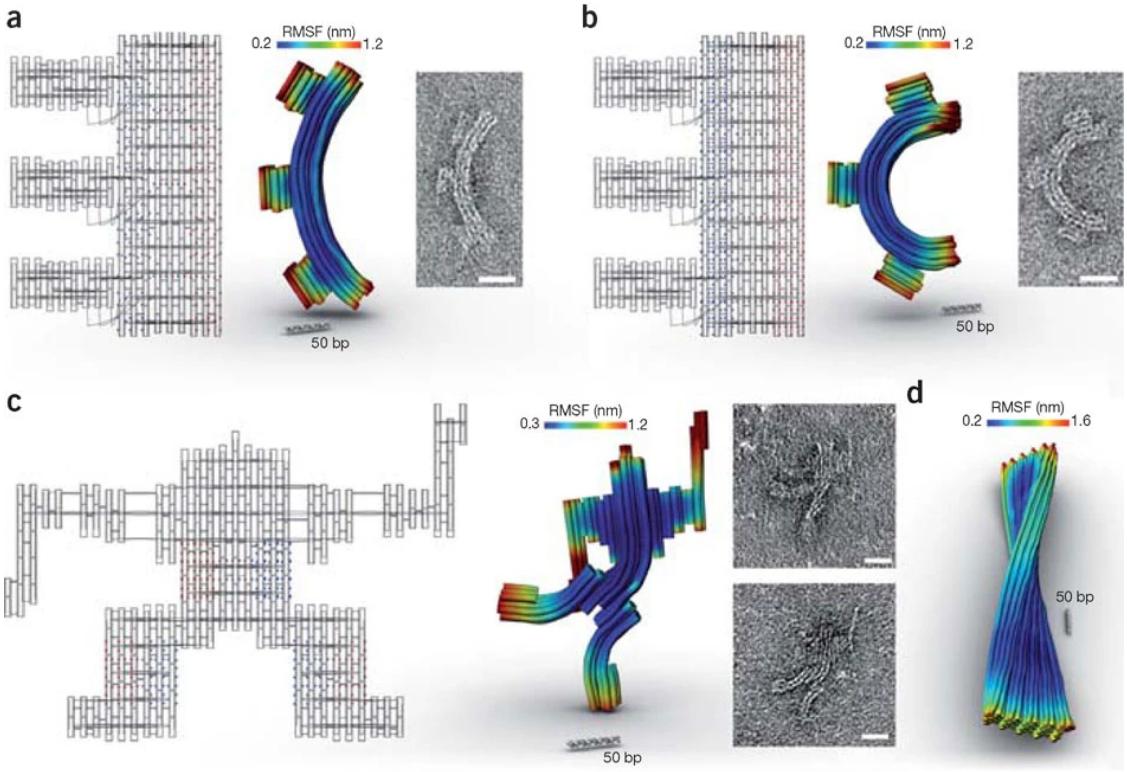


Figure 5.12: Cando

6

Structure design and analysis in oxView

Contents

This chapter contains my results on how to design, simulate and analyse DNA and RNA nanostructures.

As I started this DPhil, I was tasked with the issue of converting origami designs created in caDNAno so that they could be correctly simulated in oxDNA. I soon started collaborating with Hannah Fowler from the Doye group at Theoretical Chemistry, who simulated an extensive collection of old DNA designs. This was a good opportunity to investigate why some structures were more problematic

to convert than others.

At this time, the available method for converting a caDNAno design into the oxDNA format was to use an old python script included in the UTILS directory of the oxDNA repository. However, it was not easy to use, and it failed for many structures. At the end of 2018, the tacoxDNA webserver [49] was launched, updating the conversion script and making it more accessible.

Still, since caDNAno structures are drawn on a lattice, where all helices have to be parallel to each other, the resulting oxDNA configurations often had unnaturally extended backbone bonds, requiring time-consuming relaxation.

During a secondment within the Šulc group at Arizona State University in 2019, I contributed to the development of a web-based oxDNA viewer called oxView[50]. Among the main early features that I added to oxView was a cluster-level rigid-body dynamics option (detailed in Section 6.2) that in many cases speed up the relaxation with orders of magnitude compared to oxDNA relaxation alone. Since then, I have collaborated with the Šulc group to add more features and to make the tool more accessible as a visualiser and editor. For our second oxView publication, I rewrote the main parts of the taxoxDNA codebase into typescript, resulting in the *taxoxdna.js* library (`https://github.com/Akodiat/tacoxdna.js`) which oxView uses to import various standard design formats (including caDNAno) automatically.

This section will present my own contributions to oxView unless otherwise stated. However, I also want to acknowledge the work done by Erik Poppleton and Michael Matthies; without them, this tool would not exist. Michael was the original oxView developer and has done great work in supporting live oxDNA relaxations through the *ox-serve* webserver. Erik enabled oxView to render and analyse systems with over a million nucleotides and has created a large set of handy analysis scripts.

6.1 Importing designs

There are a lot of different formats available for DNA origami design; some of the main design tools producing them are covered in Section 5.1. This section will describe how to use oxView to import different designs.

6.1.1 Basic import

Thanks to the *tacoxdna.js* library, importing designs is now generally straightforward. Any loaded structure can then be exported for oxDNA simulation, as described in Section 6.5. The formats listed below can all be imported by simply clicking the *Import* button in oxview. However, some additional formats still require the use of the external Tacoxdna webserver[51].

Importing caDNAno files

JSON files created using caDNAno (described in Section 5.1.1) can now be imported directly into oxView. Select the file to import and make sure to also select *caDNAno* as the file format. Next, choose the correct lattice-type; either *Square* or *Hexagonal*. Optionally, input a sequence to assign to the origami scaffold (which will otherwise be random).

Another option is to use tacoxdna to convert the caDNAno design into oxDNA files and to then load those into oxView. However, designs loaded directly into oxView have the benefit of including correct colouring, clustering and base pairing information, which would otherwise have been lost.

Importing rpoly files

Rpoly files are the output from the BSCOR[32] tool (described in Section 5.1.2) for converting polyhedral meshes into DNA origami. Select the rpoly file to import, making sure that *rpoly* is selected as the file format. Optionally, input a sequence to assign to the origami scaffold (which will otherwise be random).

Importing Tiamat files

Select the tiamat *.dnajson* file to import, making sure that *tiamat* is selected as file format. Binary *.dna* Tiamat files need to be reopened in Tiamat and saved to the text-based *.dnajson*. Select Tiamat version (1 or 2), then select nucleic acid type (DNA or RNA).

By default, nucleotides without assigned base types will be given a random type. However, it is also possible to select a fixed default base.

Importing PDB files

While I did include the DNA PDB to oxDNA converter from Tacoxdna in *tacoxdna.js*, a more versatile PDB import was created by Jonah Procyk to support his ANM-oxDNA model [52]. Simply drag and drop (or load) a PDB file into an oxView window and the DNA, RNA and/or protein it contains will be automatically converted and loaded.

6.1.2 Multi-component designs

Designs spread across multiple files (or even multiple design tools) can be easily combined in oxView by simply importing them all and using the editing tools to arrange and connect them properly.

6.1.3 Far-from-physical caDNAno designs

Some structures, while converted without failure to the oxDNA format, will have a very far-from-physical configuration due to the way they are drawn in caDNAno. As mentioned above, since it is only possible to draw all helices parallel to each other, on a lattice, backbone bonds may be very elongated, creating high energies and/or topological problems.

The oxDNA software already includes relaxation procedures for such structures, bringing them together in a slow and controlled manner using a specified maximum backbone force. However, for large structures, this can take a very long time, even while using GPU simulation.

In such cases, another software I have investigated, called mrdna (Multi-resolution DNA nanotechnology), proves very useful[53]. It is a python package using ARBD (Atomic Resolution Brownian Dynamics), both being developed by Chris Maffeo, to simulate double- and single-stranded DNA structures at multiple levels of coarse-graining. Since these simulations are more coarse-grained than oxDNA, they run significantly faster. Furthermore, after I have been in contact with Chris, mrdna can now also be configured to output and simulate oxDNA

configurations as the final simulation step. Because of this, mrdna can also be useful for converting simple structures if tacoxDNA fails to parse the caDNAno file.

6.2 Rigid-body manipulation and dynamics

Rapid relaxation of converted cadnano designs

Rigid-body manipulation [54]. The translation and rotation tools in oxView allow users to select and rearrange blocks of nucleotides as rigid bodies.

Dynamics [55]

Manually, on import or through DBSCAN algorithm [56].

Clusters held together with spring forces at each shared backbone bond, with a magnitude of

$$f_{\text{spr}} = c_{spr}(l - l_r),$$

where c_{spr} is a spring constant, l is the current bond length and l_r is the relaxed bond length. To avoid overlaps, a simple linear repulsive force, of magnitude

$$f_{\text{rep}} = max\left(c_{rep}\left(1 - \frac{d}{r_a + r_b}\right), 0\right)$$

is added between the centre of each group, where c_{rep} is a repulsion constant, d is the distance between the two centres of mass, and $r_a + r_b$ is the sum of the group radii (the greatest distance they can be while still overlapping).

6.3 Editing designs

While oxView started as a visualisation tool, it has since been extended with a number of editing features. One of the earliest was the ability to perform rigid-body manipulation of selected nucleotides by dragging them with the mouse. I contributed by adding tranformation gizmos that simplify translation and rotation of selections using on-screen arrows and arcs for the user to manipulate. See Table 6.1 for a complete list of the editing tools currently available in oxView.

With the ability to create, remove, connect, and disconnect nucleotides, oxView users can now design structures from scratch. See, for example, Figure 6.2, where

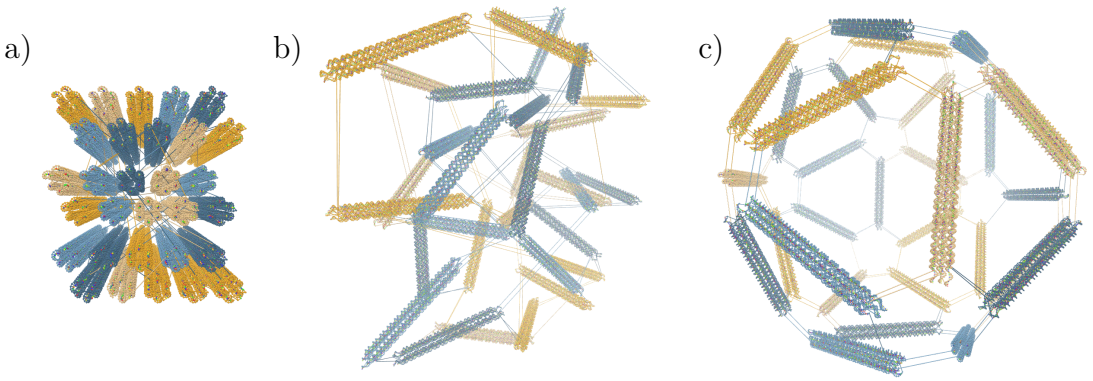


Figure 6.1: Rigid-body dynamics of clusters. Snapshots from the automatic rigid-body relaxation of an icosahedron, starting with the configuration converted from caDNAno **a)**, through the intermediate **b)** where the dynamics are applied, and **c)** the final resulting relaxed state.

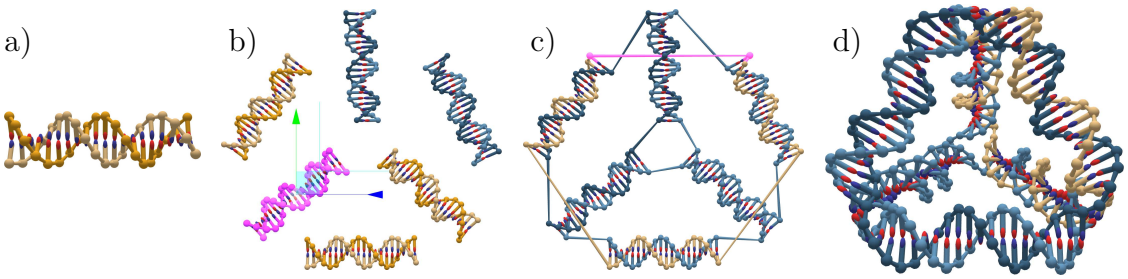


Figure 6.2: Designing the DNA tetrahedron from [57] using the oxView editing tools. a) An initial 20 base pair helix created. b) Duplicated helices being rotated and translated into place. c) Strands ligated together. d) The resulting 3D tetrahedron shape, as seen after applying rigid-body dynamics.

the DNA tetrahedron from [57] is created. The user first creates an initial helix (Figure 6.2.a) by typing a 20-base sequence and clicking the "Create" button in the "Edit" tab. Note that the "Duplex mode" need to be active in order for the complementary strand to be created automatically. Next, the user can copy and paste the helix repeatedly, using the "Translate" and "Rotate" tools to position the helices as seen in 6.2.b).

The do/undo system

6.4 Visualization options

Depending on the design, a user of oxView might want to modify the visualisation settings to make certain features of the structure more or less visible.

Tool	Description
	**Create** a new strand from a given sequence. Select *duplex mode* to instead create a helix.
	**Copy** the selected elements (Ctrl+C).
	**Cut** the selected elements (Ctrl+X).
	**Paste** elements from clipboard (Ctrl+V to paste in original position, or Ctrl+Shift+V to paste in front of camera).
	**Delete** all currently selected elements (delete).
	**Ligate** two strands by selecting the 3' and 5' endpoint elements to connect (L).
	**Nick** a strand at the selected element (N)
	**Extend** strand from the selected element with the given sequence. Select *duplex mode* to also extend the complementary strand.
	**Insert** (add) elements within a strand after the selected element.
	**Skip** (remove) selected elements within a strand.
	**Rotate** selected elements around their center of mass (R).
	**Translate** currently selected elements (T).
	**Move to**. Move other selected elements to the position of the most recently selected element.
	**Connect 3' duplex**. Connects the 3' ends of two selected staple strands with a duplex, generated from the sequence input.
	**Connect 5' duplex**. Connects the 5' ends of two selected staple strands with a duplex, generated from the sequence input.
	**Set** the sequence of currently selected elements. Select duplex mode to also set the complementary sequence on paired elements.
	**Get**. Assigns the sequence of selected bases to the sequence input.
	**Reverse complement**. Generates the reverse complement of a provided sequence.
	**Search**. Highlights the position the provided sequence in each strand, if present.

Table 6.1: Editing tools available in oxView

Centring with periodic boundary conditions. A useful utility when visualising an oxDNA trajectory is to keep the structure centred at the origin, stopping it from drifting out of view. The method described in [58] was used to achieve centring while taking periodic boundary conditions into account.

In essence, for each coordinate $p_j = \left(p_x^j, p_y^j, p_z^j\right)$ in the centring set of size n,

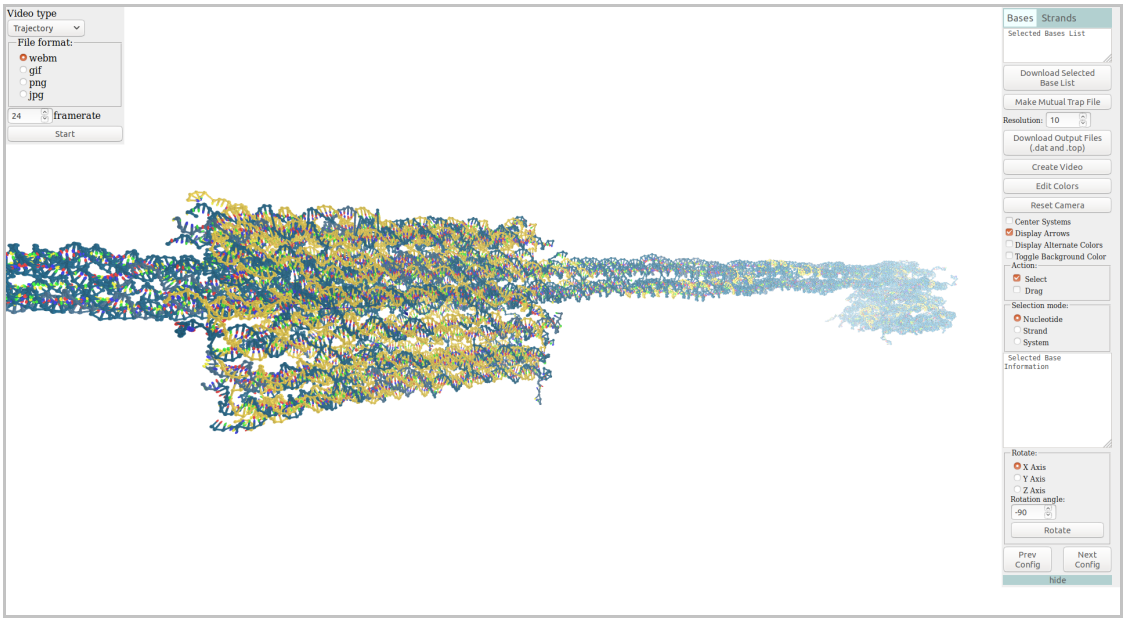


Figure 6.3: Screenshot of the online oxView tool, exporting a video of a slider on a rail from a simulated trajectory.

each dimension $i \in \{x, y, z\}$ gets averaged in its own variable α_i, representing its 1D interval as a 2D circle (where the circumference b_i is the bounding box side length:

$$c_i = \frac{1}{n} \sum_{j=1}^{n} \left[\cos\left(\alpha_i^j\right), \sin\left(\alpha_i^j\right) \right]$$

Where $\alpha_i^j = \frac{2\pi}{b_i} p_i^j$ is the angle on the unit circle and c_i is the average 2D position representing dimension i. Finally, the averages are converted back to cartesian coordinates:

$$cm_i = \pi + \frac{b_i}{2\pi} atan2(-c_{i,y}, -c_{i,x})$$

Change component sizes , change colours, fog, virtual reality.

6.5 Exporting designs

Once a structure has been created in or loaded into oxView, it can be exported in a variety of formats.

6.5.1 Exporting oxDNA simulation files

OxView can export topology, configuration, and external force files for oxDNA simulation. Simply click the "Export oxDNA" button in the "File" menu and select the required file types.

One important thing to note is that the oxDNA format requires nucleotides to be correctly sorted with consecutive indices. Furthermore, this sorting is done, against convention, in 3' to 5' order. Meanwhile, to facilitate editing, oxView nucleotides keep their indices even if the topology changes. Thus, nucleotides indices may be reassigned on oxDNA export, so it is important to export all files (topology, configuration, and forces) if the design has been edited.

6.5.2 Exporting other 3D formats

File formats such as glTF and STL contain geometrical information that can be 3D printed or imported into other 3D software such as Blender. OxView exports the scene as it is at the moment of export (at the current frame if a simulation trajectory is loaded), so it is important to configure intended component sizes and colours beforehand.

STL is an old and common standard for 3D shapes, containing only vertex coordinates (no colours). It is a popular input for 3D printing, but the file size is relatively large.

The glTF format (or glb if binary) is a modern standard for 3D scenes, storing geometry, hierarchy, and even material properties.

6.5.3 Exporting sequence files

Strand sequences can be exported as standard CSV files using the "Sequence file" export button in the "File" tab. The designed sequences can then be ordered and assembled experimentally.

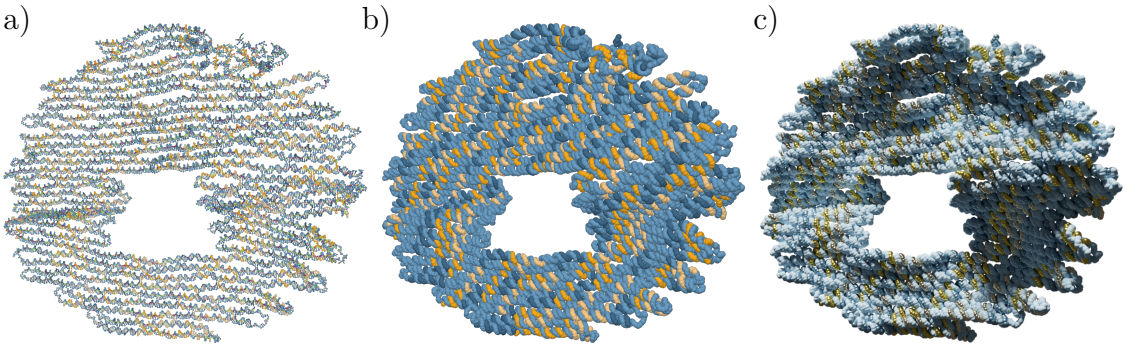


Figure 6.4: Component scaling and image export. a) Default oxView visualisation. b) Custom component scale and visibility. Using the"Visible components" dropdown in the "View" menu, the backbone spheres have been scaled up by a factor of 4.18 while all other nucleotide components have been hidden. c) The scaled scene in b) exported as glTF and rendered using Cycles in Blender.

6.5.4 Saving image files

It is possible to save an image of the current oxView view by simply clicking the "Save image" button in the "File" tab. This is a preferred option over a screenshot for two reasons. First, is possible to increase the resolution by a scaling factor, found in the "Image size" dropdown (rescale the browser window to change the aspect ratio of the image). Secondly, the background of the saved image will be transparent, simplifying further editing and composition.

For cover art and photo-realistic renders, it is also possible to export and load a glTF file into (for example) Blender, as seen in Figure 6.4. Note that in current Blender versions (2.9), large structures take a long time to import. So, make sure to disable any components not needed before export.

6.5.5 Creating videos

One of the earlier features I implemented in oxView was the ability to create videos from oxDNA simulation trajectories. It is also possible to create leminiscate videos of single configurations, where the camera moves around the structure in a leminiscate-shaped loop. The video export uses the `CCapture.js` library, grabbing the `Three.js` canvas and outputing either "webm" or "gif" animations, or each frame as separate "jpg" or "png" images.

Click "Create video" in the "File" menu, choose video type (trajectory or leminiscate), file format, and frame rate. For leminiscate videos, it is also possible to set a video duration.

For trajectory videos, the camera can be manually moved while the video is being recorded, thus showing the simulation from different angles.

Another option for video creation is to follow the steps described in Section 6.5.4. When the structure is loaded into Blender, the camera can be animated (or the design rotated) using keyframes to render high-quality videos. To render an oxDNA simulation, use the "traj2blender.py" script, found at `https://github.com/Akodiat/traj2blender`, to automatically load keyframes corresponding to simulation steps.

6.6 Converting RNA origami designs

During my secondment at the Andersen lab in Aarhus, I worked with converting RNA structures designed using their ASCII-based blueprint format into oxRNA simulation files. Examples of converted structures are shown in Figure 6.5. The Andersen lab already has scripts, soon to be published, for parsing their blueprint files and building the corresponding PDB structures (the first two columns of Figure 6.5). However, the resulting PDB files are not relaxed and would take a long time to relax using all-atom simulation. As such, I modified the tacoxDNA [49] PDB parser to enable PDB-to-oxRNA conversion. I have contacted Lorenzo to let him know about these additions, and hopefully, they will be included in a future version of tacoxDNA. One issue I noticed with the conversion was that the asterisk (*) and prime (') characters were used interchangeably in the RNA motif library to name atoms of the sugar group, causing problems with the tacoxDNA script, which only recognises the prime. This has already been fixed in tacoxDNA. The third column of Figure 6.5 shows the structures relaxed and simulated in oxRNA, some significantly different from the previously available PDB models in the second column.

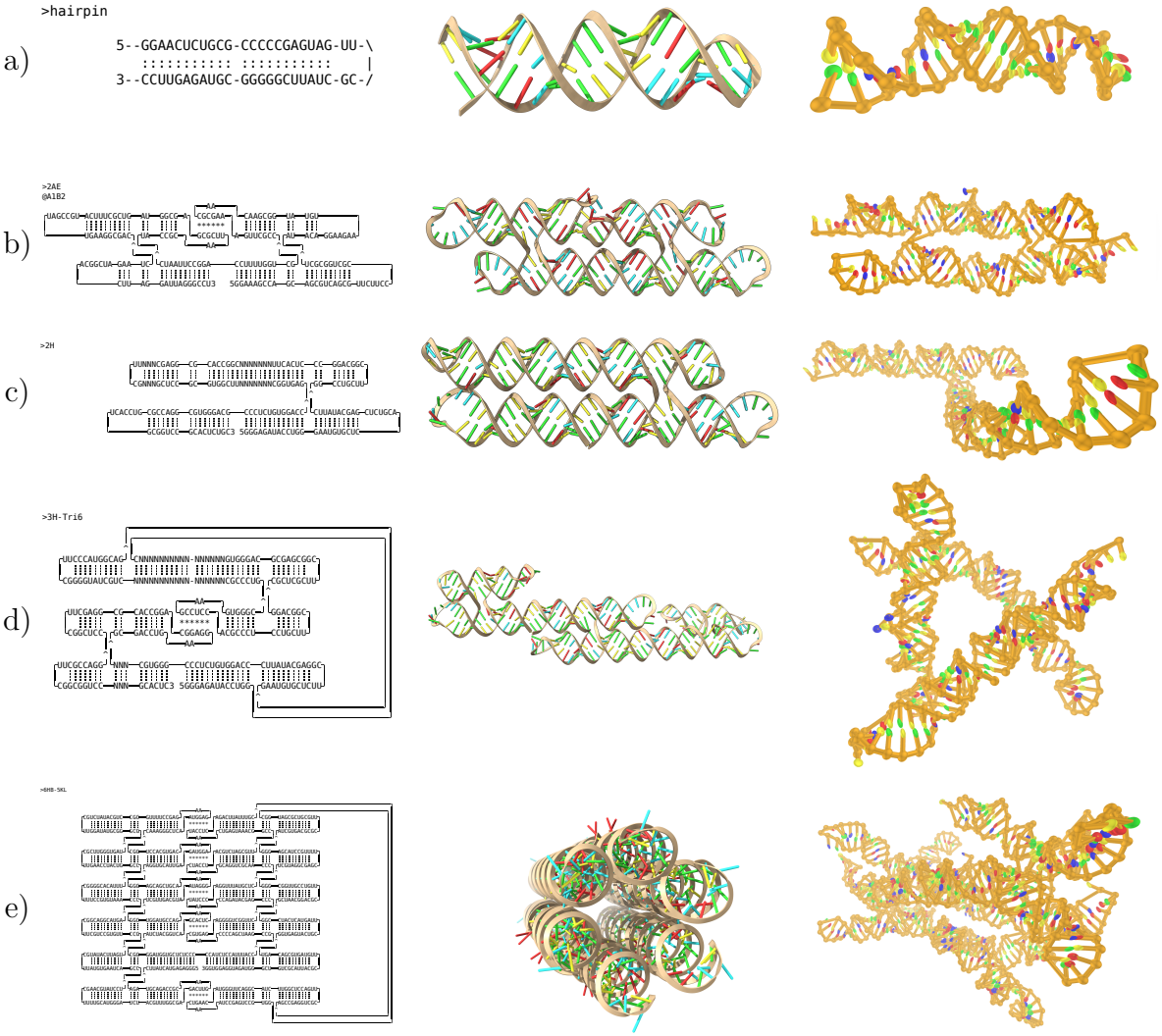


Figure 6.5: Conversion and simulation of various RNA designs. Each row, from left to right, shows the ASCII blueprint design, the PDB model (visualised using ChimeraX), and a frame from the simulated structure (visualised using oxView). **a)** Is a simple hairpin loop. **b)** is a two-helix bundle tile used in [7]. **c)** is two helices connected by a double crossover, analysing the flexibility of such a motif. **d)** is a possible design for a tensegrity triangle. **e)** is a siz-helix bundle.

You live and learn. At any rate, you live.

— Douglas Adams, Mostly Harmless

7 Conclusion

In conclusion, nanostructures self-assembly design has been investigated on both an abstract and a more detailed level. The presented projects have resulted in tools and methods for creating, simulating and analysing self-limiting modular structures with minimal complexity, potentially containing building blocks created in different design software.

7.1 Individual module design

7.2 Modular assembly

7.3 Future work

Staged assembly

Given a maximum complexity - for example, a limited number of possible colours - what is the largest bounded structure that can deterministically assemble?

Appendices

A

The polycube codebase

Contents

The code used for polycube assembly can be found at `https://github.com/Akodiat/polycubes`.

A.1 Stochastic assembly code

A.1.1 C++

A.1.2 JavaScript

A.2 Polycube solver

A.2.1 Python

A.2.2 JavaScript

A.3 Analysis

B

The oxView codebase

The code for oxView can be found at `https://github.com/sulcgroup/oxdna-viewer`. It is written in TypeScript and compiled into JavaScript. The 3D visualisation is done with the help of the Three.js library, while the graphical user interface uses the Metro 4.

To compile, you need both typescript and Node.js installed. Typing `npm install` should install all required node modules.

Type `tsc` to compile.

References

[1] Chris R Calladine and Horace Drew. *Understanding DNA: the molecule and how it works*. Academic press, 1997.

[2] Nadrian C. Seeman. *Structural DNA Nanotechnology*. Cambridge University Press, 2016.

[3] Paul WK Rothemund. "Folding DNA to create nanoscale shapes and patterns". In: *Nature* 440.7082 (2006), p. 297.

[4] Thomas E Ouldridge, Ard A Louis, and Jonathan PK Doye. "DNA nanotweezers studied with a coarse-grained model of DNA". In: *Physical review letters* 104.17 (2010), p. 178101.

[5] Shawn M. Douglas et al. "Rapid prototyping of 3D DNA-origami shapes with caDNAno". In: *Nucleic Acids Research* 37.15 (2009), pp. 5001–5006.

[6] Peixuan Guo. "The emerging field of RNA nanotechnology". In: *Nature nanotechnology* 5.12 (2010), p. 833.

[7] Cody Geary, Paul WK Rothemund, and Ebbe S Andersen. "A single-stranded architecture for cotranscriptional folding of RNA nanostructures". In: *Science* 345.6198 (2014), pp. 799–804.

[8] Steffen L Sparvath, Cody W Geary, and Ebbe S Andersen. "Computer-aided design of RNA origami structures". In: *3D DNA Nanostructure*. Springer, 2017, pp. 51–80.

[9] Cody Geary et al. "RNA origami design tools enable cotranscriptional folding of kilobase-sized nanoscaffolds". In: *Nature chemistry* 13.6 (2021), pp. 549–558.

[10] Ming Li and P. M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications [electronic resource]*. eng. Fourth edition. Texts in computer science. Cham, 2019.

[11] Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. "Input–output maps are strongly biased towards simple outputs". In: *Nature communications* 9.1 (2018), p. 761.

[12] Nadrian C Seeman. "Nucleic acid junctions and lattices". In: *Journal of theoretical biology* 99.2 (1982), pp. 237–247.

[13] Erik Winfree. "Algorithmic self-assembly of DNA". PhD thesis. California Institute of Technology, 1998.

[14] Erik Winfree et al. "Design and self-assembly of two-dimensional DNA crystals". In: *Nature* 394.6693 (1998), p. 539.

[15] Luvena L Ong et al. "Programmable self-assembly of three-dimensional nanostructures from 10,000 unique components". In: *Nature* 552.7683 (2017), pp. 72–77.

[16] Grigory Tikhomirov, Philip Petersen, and Lulu Qian. “Fractal assembly of micrometre-scale DNA origami arrays with arbitrary patterns”. In: *Nature* 552.7683 (2017), p. 67.

[17] Grigory Tikhomirov, Philip Petersen, and Lulu Qian. “Programmable disorder in random DNA tilings”. In: *Nature nanotechnology* 12.3 (2017), pp. 251–259.

[18] Klaus F Wagenbauer, Christian Sigl, and Hendrik Dietz. “Gigadalton-scale shape-programmable DNA assemblies”. In: *Nature* 552.7683 (2017), p. 78.

[19] Sungwook Woo and Paul WK Rothemund. “Programmable molecular recognition based on the geometry of DNA nanostructures”. In: *Nature chemistry* 3.8 (2011), pp. 620–627.

[20] Christian Sigl et al. “Programmable icosahedral shell system for virus trapping”. In: *Nature Materials* 20.9 (2021), pp. 1281–1289.

[21] Zhiwei Lin et al. “Engineering Organization of DNA Nano-Chambers through Dimensionally Controlled and Multi-Sequence Encoded Differentiated Bonds”. In: *Journal of the American Chemical Society* 142.41 (2020). PMID: 32902966, pp. 17531–17542. eprint: https://doi.org/10.1021/jacs.0c07263. URL: https://doi.org/10.1021/jacs.0c07263.

[22] Mingyang Wang et al. “Programmable Assembly of Nano-architectures through Designing Anisotropic DNA Origami Patches”. In: *Angewandte Chemie International Edition* 59.16 (2020), pp. 6389–6396. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/anie.201913958.

[23] Hao Wang. “Proving theorems by pattern recognition—II”. In: *Bell system technical journal* 40.1 (1961), pp. 1–41.

[24] David Doty. “Theory of algorithmic self-assembly”. In: *Communications of the ACM* 55.12 (2012), pp. 78–88.

[25] David Doty. “DNA tile self-assembly”. Tutorial presented at the 23rd International Conference on DNA Computing and Molecular Programming. Sept. 2017. URL: https://web.cs.ucdavis.edu/~doty/papers/dna23-tile-assembly-tutorial.pdf.

[26] SE Ahnert et al. “Self-assembly, modularity, and physical complexity”. In: *Physical Review E* 82.2 (2010), p. 026117.

[27] Iain G Johnston et al. “Evolutionary dynamics in a simple model of self-assembly”. In: *Physical Review E* 83.6 (2011), p. 066105.

[28] Kamaludin Dingle, Guillermo Valle Pérez, and Ard A Louis. “Generic predictions of output probability based on complexities of inputs and outputs”. In: *Scientific reports* 10.1 (2020), pp. 1–9.

[29] Flavio Romano et al. “Designing patchy interactions to self-assemble arbitrary structures”. In: *Physical Review Letters* 125.11 (2020), p. 118003.

[30] David Doty, Benjamin L Lee, and Tristan Stérin. "scadnano: A browser-based, scriptable tool for designing DNA nanostructures". In: *DNA 2020: Proceedings of the 26th International Meeting on DNA Computing and Molecular Programming*. Ed. by Cody Geary and Matthew J. Patitz. Vol. 174. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 9:1–9:17. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12962.

[31] Nick Conway and Shawn Douglas. *Windows Installation - caDNAno*. URL: https://cadnano.org/windows-installation.html (visited on 08/20/2021).

[32] Erik Benson et al. "DNA rendering of polyhedral meshes at the nanoscale". In: *Nature* 523.7561 (2015), pp. 441–444.

[33] Erik Benson et al. "Computer-Aided Production of Scaffolded DNA Nanostructures from Flat Sheet Meshes". In: *Angewandte Chemie International Edition* 55.31 (2016), pp. 8869–8872.

[34] Hyungmin Jun et al. "Rapid prototyping of arbitrary 2D and 3D wireframe DNA origami". In: *Nucleic Acids Research* (Sept. 2021). gkab762. eprint: https://academic.oup.com/nar/advance-article-pdf/doi/10.1093/nar/gkab762/40347509/gkab762.pdf. URL: https://doi.org/10.1093/nar/gkab762.

[35] Haichao Miao et al. "Multiscale Visualization and Scale-adaptive Modification of DNA Nanostructures". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018). URL: https://www.cg.tuwien.ac.at/research/publications/2018/miao_tvcg_2018/.

[36] Chao-Min Huang et al. "Integrated computer-aided engineering and design for DNA assemblies". In: *Nature Materials* (2021), pp. 1–8.

[37] James C Phillips et al. "Scalable molecular dynamics with NAMD". In: *Journal of computational chemistry* 26.16 (2005), pp. 1781–1802.

[38] Wendy D Cornell et al. "A second generation force field for the simulation of proteins, nucleic acids, and organic molecules J. Am. Chem. Soc. 1995, 117, 5179-5197". In: *Journal of the American Chemical Society* 118.9 (1996), pp. 2309–2309.

[39] Bernard R Brooks et al. "CHARMM: a program for macromolecular energy, minimization, and dynamics calculations". In: *Journal of computational chemistry* 4.2 (1983), pp. 187–217.

[40] Jejoong Yoo and Aleksei Aksimentiev. "In situ structure and dynamics of DNA origami determined through molecular dynamics simulations". In: *Proceedings of the National Academy of Sciences* 110.50 (2013), pp. 20099–20104.

[41] Aditya Sengar et al. "A primer on the oxDNA model of DNA: When to use it, how to simulate it and how to interpret the results". In: *arXiv preprint arXiv:2104.11567* (2021).

[42] Rahul Sharma et al. "Characterizing the motion of jointed DNA nanostructures using a coarse-grained model". In: *ACS nano* 11.12 (2017), pp. 12426–12435.

[43] Petr Šulc et al. "A nucleotide-level coarse-grained model of RNA". In: *The Journal of chemical physics* 140.23 (2014), 06B614_1.

[44] Thomas Gerling et al. “Dynamic DNA devices and assemblies formed by shape-complementary, non–base pairing 3D components”. In: *Science* 347.6229 (2015), pp. 1446–1452.

[45] Reza M Zadegan et al. “Construction of a 4 zeptoliters switchable 3D DNA box origami”. In: *ACS nano* 6.11 (2012), pp. 10050–10053.

[46] Tim Liedl et al. “Self-assembly of three-dimensional prestressed tensegrity structures from DNA”. In: *Nature nanotechnology* 5.7 (2010), p. 520.

[47] Dongran Han et al. “Folding and cutting DNA into reconfigurable topological nanostructures”. In: *Nature nanotechnology* 5.10 (2010), p. 712.

[48] Do-Nyun Kim et al. “Quantitative prediction of 3D solution shape and flexibility of nucleic acid nanostructures”. In: *Nucleic acids research* 40.7 (2012), pp. 2862–2868.

[49] Antonio Suma et al. “TacoxDNA: A user-friendly web server for simulations of complex DNA structures, from single strands to origami”. In: *Journal of Computational Chemistry* (2019). URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.26029.

[50] Erik Poppleton et al. “Design, optimization and analysis of large DNA and RNA nanostructures through interactive visualization, editing and molecular simulation”. In: *Nucleic acids research* 48.12 (2020), e72–e72.

[51] Antonio Suma et al. “TacoxDNA: A user-friendly web server for simulations of complex DNA structures, from single strands to origami”. In: *Journal of computational chemistry* 40.29 (2019), pp. 2586–2595.

[52] Jonah Procyk, Erik Poppleton, and Petr Šulc. “Coarse-grained nucleic acid–protein model for hybrid nanotechnology”. In: *Soft Matter* 17.13 (2021), pp. 3586–3593.

[53] Chris Maffeo and Aleksei Aksimentiev. *Multi-resolution simulations of self-assembled DNA nanostructures*. 2018. URL: https://gitlab.engr.illinois.edu/tbgl/tutorials/multi-resolution-dna-nanotechnology/tree/master.

[54] Chao-Min Huang et al. “Uncertainty quantification of a DNA origami mechanism using a coarse-grained model and kinematic variance analysis”. In: *Nanoscale* 11.4 (2019), pp. 1647–1660.

[55] David Baraff. *An introduction to physically based modeling: Rigid Body Simulation I — Unconstrained Rigid Body Dynamics*. 1997.

[56] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *Kdd*. Vol. 96. 34. 1996, pp. 226–231.

[57] Russell P Goodman et al. “Rapid chiral assembly of rigid DNA building blocks for molecular nanofabrication”. In: *Science* 310.5754 (2005), pp. 1661–1665.

[58] Linge Bai and David Breen. “Calculating Center of Mass in an Unbounded 2D Environment”. In: *Journal of Graphics Tools* 13.4 (2008), pp. 53–60. URL: https://doi.org/10.1080/2151237X.2008.10129266.