Interactive Visualization of RNA and DNA Structures
Interactive Visualization of RNA and DNA Structures

Norbert Lindow, Daniel Baum, Morgan Leborgne & Hans-Christian Hege

July 31, 2018

Abstract

The analysis and visualization of nucleic acids (RNA and DNA) play an increasingly important role due to the growing number of known 3-dimensional structures of such molecules. The great complexity of these structures, in particular, those of RNA, demands interactive visualization to get deeper insights into the relationship between the 2D secondary structure motifs and their 3D tertiary structures. Over the last decades, a lot of research in molecular visualization has focused on the visual exploration of protein structures while nucleic acids have only been marginally addressed. In contrast to proteins, which are composed of amino acids, the ingredients of nucleic acids are nucleotides. They form structuring patterns that differ from those of proteins and, hence, also require different visualization and exploration techniques. In order to support interactive exploration of nucleic acids, the computation of secondary structure motifs as well as their visualization in 2D and 3D must be fast. Therefore, in this paper, we focus on the performance of both the computation and visualization of nucleic acid structure. For the first time, we present a ray casting-based visualization of RNA and DNA secondary and tertiary structures, which enables real-time visualization of even large molecular dynamics trajectories. Furthermore, we provide a detailed description of all important aspects to visualize nucleic acid secondary and tertiary structures. With this, we close an important gap in molecular visualization.

1 Introduction

Nucleic acids are macromolecules that play a major role in many cellular processes [47]. While deoxyribonucleic acid (DNA) is mainly responsible to carry genetic information, different types of ribonucleic acids (RNA) are necessary to turn this genetic information into new molecular structures. Mainly three types of RNA are involved in this process: (i) messenger RNA (mRNA); (ii) transfer RNA (tRNA); and (iii) ribosomal RNA (rRNA). DNA consists of two strands
of nucleotides that are interconnected by base pairs and form the typical double helix structure discovered by Watson and Crick in 1953 [48]. In RNA molecules, on the other hand, single strands usually fold back onto themselves by building pairings of bases from the same strand. Typical secondary structure elements of RNA are stems, hairpin loops, internal loops, bulges, and junctions [45], see Fig. 1. The tertiary structure of RNA is formed by additional base pairs that are usually further apart along the nucleotide sequence (see the blue arc in Fig. 1, left) or are even built between different strands. Examples for tertiary structure elements are pseudo-knots and kissing hairpins [45].

The types of RNA differ immensely in size and complexity. While tRNA has a very small conserved structure, mRNA and some rRNA molecules can contain several hundreds to thousands of nucleotides with an enormous degree of complexity [34]. To elucidate their structures, sophisticated tools are required that allow the visual exploration of the primary, secondary and tertiary structures. For almost 40 years [53], researchers have used 2D visualizations that depict the primary and secondary structures as a graph embedded in 2D. The tertiary structure, however, can only be well understood by 3D visualizations that show the primary structure as backbone, and the secondary and tertiary structures by the bases and base pairs embedded in 3D space. To avoid visual clutter, these 3D visualizations usually do not show the atomic structure but simplified representations of the nucleotides.

For all such tools, the determination of base pairs plays a crucial role. Many algorithms have been proposed [26, 27, 31, 32, 39, 52] to detect the wide variety of base pairing motifs [33]. However, these algorithms are not fast enough to allow for interactive visualization. Hence, there is a clear need for a base pair detection algorithm that is both fast and robust. This is one motivation for our work.

A second motivation is a lacking support of specialized visualizations for RNA and DNA in standard molecular visualization packages as recently stated by Hanson and Lu [13]. To overcome this shortcoming, they extended Jmol [13] to support the visualization of base pairs and secondary structures by coupling...
Figure 2: The double-stranded helix structure of DNA (left) and the single-stranded structure of RNA (right). Both nucleic acids are composed of nucleotides that consist of a backbone part and a nucleobase. The backbone contains sugar (beige ring) and a phosphate group. In contrast to RNA, DNA has no OH-group at the sugar (deoxy-), see the small gray box. While the nucleotides in DNA contain the bases adenine (A), cytosine (C), guanine (G), and thymine (T) with pairs A-T and G-C, in RNA uracil (U) replaces thymine with pairs A-U and G-C.

Interactive exploration, however, is still lacking since no 2D visualizations are provided in Jmol, which limits the selection of secondary structures to command line interaction. This problem is overcome by Assemble2 [22] in combination with UCSF Chimera [37]. Here, Assemble2 provides the computation and visualization of 2D secondary and tertiary structures while Chimera is used for the 3D visualization. Thus, the selection of the secondary structure done in Assemble2 can be directly visualized in Chimera. The other direction, however, does not seem to be possible at the moment. In our work, we go one step further by tightly linking the 2D visualizations with a fast and high-quality 3D rendering, allowing for a mutual selection in all visualizations. The overall contribution of this paper, however, is to show that interactive visual exploration even for very large nucleic acids is feasible with accelerated algorithms for computation and visualization. To this end we make the following sub-contributions:

1. In Sect. 4: We describe an efficient algorithm for the identification of standard and modified bases and the computation of their base pairings. This algorithm only needs very little information like the atomic number and the grouping of atoms into residues.

2. In Sect. 5: We present fast and high-quality rendering techniques for two 3D visualization models of nucleic acids. In particular, we developed the first ray casting approach that can also be used for protein secondary structure visualization. With this rendering technique, real-time exploration of even large dynamic trajectories becomes possible.

3. In Sect. 6: We improve a previous algorithm for the visualization of the 2D secondary structure by providing a better initialization that allows a faster optimization.

4. In Sects. 7 & 8: We show that a seamless exploration of secondary and
tertiary molecular structures is possible by the mutual interaction of the 2D and 3D visualizations described in this paper.

2 Nucleic Acids

Deoxyribonucleic acid (DNA) usually appears as double helix of two twisted strands of nucleotides, see Fig. 2 (left). A nucleotide consists of a backbone part and a nucleobase. The backbone part contains the sugar deoxyribose (beige ring) and a phosphate group that connects the nucleotides along the strand. While the backbone part is equal for all nucleotides, four different nucleobases can appear in DNA: adenine (A), cytosine (C), guanine (G), and thymine (T). These bases can be divided into two groups: the single-ring pyrimidines C and T, and the double-ring purines A and G. Typically, each nucleobase of one strand forms a pair with a nucleobase of the other strand by creating hydrogen bonds. In most cases, these pairs are Watson-Crick base pairs [48], where adenine prefers to bind to thymine (A-T) with two hydrogen bonds, and guanine binds to cytosine (G-C) with three hydrogen bonds (dashed lines in Fig. 2). Due to the additional hydrogen bond, G-C pairs are stronger than A-T pairs. Thus, the number of G-C pairs and their distribution along the helix has an impact on the stability of the DNA structure. Furthermore, its stability is also greatly influenced by the base stacking [23], which is the result of a dipole-dipole interaction being induced by the aromatic rings of the heterocyclic bases when the same base pairs are stacked along the strand [51]. This stacking is also responsible for the base rings being parallel and the small offset that creates the twisting. Notice that G-C/G-C stacks add more stability than A-T/A-T stacks.

Ribonucleic acids (RNA) mainly occur as single-stranded structures that partially fold back onto themselves by forming base pairs between nucleotides of the same strand. The chemical structures of the RNA nucleotides are quite similar to those of DNA. In fact, the backbone part has the same structure except for an additional OH-group at the second position of the sugar (beige ring in Fig. 2). This additional OH-group is the main cause that makes RNA less stable than DNA. The bases are the same except for thymine, which is replaced by uracil (U). The common base pairs are A-U and G-C which mainly form two types of secondary structure elements: stems and loops. Loops can be further classified into hairpin loops, internal/bulge loops, and junctions.

In addition to the Watson-Crick base pairs, many variations and modifications can appear. For example, in Hoogsteen base pairs [26, 27], the purine bases are rotated by 180° w.r.t. to the classical Watson-Crick base pairs. Thus, Hoogsteen base pairs build different hydrogen bonds. Another variation is the Wobble pair [26, 27], where G usually pairs with U, which mainly appears in RNA, see Suppl. Fig. 2. Besides the base pair variations, the chemical structure of the nucleotides can also be modified, either in the backbone or in the base part.
3 Related Work

3.1 Base Pair Computation

The first step in computing base pairs is the determination of hydrogen bonds that typically form between the bases of the nucleotides. Since in most molecular structures the hydrogen atoms are missing, one can either explicitly add the hydrogen atoms to the molecular structure, or one can consider the distance between donor and acceptor, thereby ignoring the exact positions of the hydrogen atoms.

Lemieux and Major [26] take the first approach. Different methods exist for adding hydrogen atoms [6, 49]. They use the force field-based approach by Cornell et al. [6]. Then, the probabilities of all potential hydrogen bonds are computed and a bipartite graph is constructed on which the maximum flow is computed to identify the best hydrogen bonds. The pairs of bases with the largest sum of hydrogen bond probabilities are then defined as base pairs.

The second approach is followed, for example, by Yang et al. [52] and Lu, Olson, and co-workers [31, 32]. For the computation of base pairs, they use the standard reference frame for nucleic acid base pairs defined by Olson et al. [36], based on standard bases and base pairs from the Cambridge Structure Database [1]. The approach consists of several steps. First, standard bases together with the reference frame are aligned to the real bases. Then, based on the reference frame, several measurements are taken for potential base pairs, like distances and angles. If these measurements are within certain bounds, two bases are considered to build a base pair.

We developed a simpler approach that leads to very similar results but is much faster (Sects. 4.2, 8.1). This improvement is required to enable base pair computation in real-time for dynamic nucleic acids.

3.2 2D Visualization

2D visualizations depict the primary and secondary structures as a graph in the 2D plane (see Fig. 1). They are mainly interesting for RNA molecules, since these build complex secondary structures, whereas DNA basically only shows the well-known double helix. One of the first algorithms to generate 2D visualizations of RNA molecules was described already in 1982 by Shapiro et al. [40]. The visualization they presented can still be considered as the standard to depict 2D secondary structures. Here, loops are represented by circles whereas stems are depicted by ladders (Fig. 1, right). This first approach, however, needed manual interaction to resolve overlaps. Therefore, in 1984 they presented an algorithm that generates overlap-free visualizations [41], starting from the circle visualization by Zuker et al. [53]. In order to generate the standard visualization [40], they used this new visualization as starting point to their previous approach [40]. Though this does not guarantee overlap-free visualizations in all cases, in general, it works very well. A few years later, Brucoleri and Heinrich [4] presented an improved algorithm, called NAView, to further reduce the number of overlaps and to create more pleasing visualizations. This algorithm is also used, for example, in the ViennaRNA package [17, 18, 30].

Since automatic methods rarely produced fully satisfying results, in particular for very complex molecular structures, De Rijk and De Wachter [9] followed
a different approach. with their RnaViz tool. They did not aim at creating overlap-free visualizations in the first place but designed RnaViz to allow quick manual editing to arrive at pleasing publication-ready visualizations. Later they presented an improved version [10], called RnaViz 2, that included the support of more file formats as well as better annotations. The VARNA tool by Darty et al. [8] provides four classical representations for 2D secondary structure and allows editing and annotation. As an additional property, it can be run on a web-server.

In 2013, Hecker et al. [15] presented their RNAfdl tool, which is able to automatically create overlap-free visualizations even for very large RNA structures like ribosomes. Similar to the algorithm by Shapiro et al. [41], it starts from the circle plot [53]. To arrive at visualizations that are similar to the standard drawing [40], they ran an optimization algorithm using gradient descent. They themselves describe their algorithm as slow, but since it produces very good visualizations, we use it in our approach, too. To accelerate it, we use a spatial data structure and start the computation from a slightly modified initial layout that allows a faster optimization (Sect. 6.2).

3.3 3D Visualization

As pointed out by Hanson and Lu in their recent paper [13], support of specialized 3D visualizations for RNA and DNA in standard molecular visualization packages is largely missing. Even tools like Genome3D [28] and GMOL [35], which are dedicated to the analysis of the genome using multi-scale visualization functionality and which support a great variety of functionality, do not seem to offer base-pair visualization. One visualization package that does offer special visualizations of bases and base pairings is 3DNA [32]. It visualizes the bases as blocks that describe the standard reference frame [36] aligned to the bases in the molecular structure. In the ideal case, the blocks of paired bases are in-plane and have opposite orientations. The deviation from this ideal allows the user to obtain information about the type and quality of a base pairing. A similar visualization is used in the BPviewer [52], UCSF Chimera also offers support for the visualization of bases. These visualizations include filled rings, boxes, ellipses and elliptical sticks [7]. Later the authors added a ladder visualization depicting the base pairs as one cylinder connecting the backbone atoms of the paired bases. However, those pairings are not bijective, that is, one base can be paired with two or even more other bases.

The main purpose of the Assemble tool presented by Jossinet et al. [21] is to enable users to model nucleic acids. For this purpose, they combine several tools including RNAplot [17] for the generation of 2D secondary structure visualizations and RNAview [52] for the annotation of 3D structures. Assemble2 [20, 22] represents an improved version of the previous modeling tool that uses Chimera [7,37] to visualize the 3D structure. Selection of secondary structures, which can also be shown in Chimera, can be done using RNAplot. The other direction, however, does not seem to be possible.

Built on their experience from 3DNA, Lu et al. developed DSSR [31], a very powerful tool to analyze RNA structures that uses Jmol for the 3D visualization. Recently, Hanson and Lu described this integration [13], which is based on a JSON-interface that directly couples DSSR and the 3D visualization of Jmol. This is a great improvement, but still missing is the integration of 2D secondary
structure visualizations and brushing & linking techniques to enable simple selection with and exploration of the 3D molecular structure. One contribution of this paper is to show how a full linking between 3D and 2D visualizations can be done and what benefits arise from such a tight coupling (see Sects. 8 and 9).

Closely related to the rendering of nucleic acid molecular structures is the rendering of secondary structures of proteins. The most interesting rendering mode in this respect is the ribbon. In terms of efficiency this has mainly been looked at by three groups [16, 25, 46]; also see the survey paper on molecular visualization by Krone et al. [24]. Already in 2008, Krone et al. [25] investigated several GPU implementations to compute the ribbon geometry on the fly, hereby exploiting the geometry shader. Three years later, Whale and Birmanns [46] presented an approach utilizing vertex shaders. Compared to Krone et al., with their hybrid GPU/CPU-based approach they achieve a three times faster rendering. Hermosilla et al. [16] exploited the tessellation shader to generate the ribbons. This allows the rendering of even large molecular structures in ribbon mode and is also suitable for the rendering of molecular dynamics trajectories. With our ray casting-based approach (Sect. 5) we present an alternative that generates visualizations of even higher quality while being comparable in speed.

4 Secondary Structure Computation

The secondary structure of nucleic acids is generally not stored in the files containing the molecular structure information (for example, pdb files). Thus, in order to be able to visualize the secondary structure, we need to compute it from the nucleotide information. This requires the computation of all base pairs of the nucleotide sequence, which again requires the determination of the hydrogen bonds of the nucleobases (Sect. 4.2). However, since the names of the nucleotides do not always follow a unique convention, we also need to determine the type of each nucleotide including its base type and modification (Sect. 4.1). For this purpose, other approaches rely on the unique naming and ordering of the atoms [31], for example, N1, C2, N3, C4, C5, C6 etc. We found that we cannot always rely on this information. Therefore, we reduced the requirements even further and only assume that for each atom the Cartesian coordinates and the atomic number are given and that all atoms belonging to one nucleotide are grouped into the same residue. Note that in molecular file formats, each atom is assigned to exactly one residue. See Fig. 3 for an overview of the processing pipeline.
4.1 Nucleotide Type Detection

In this section, we describe the detection of nucleotides and their types. We ignore hydrogen atoms since they are often not given in the molecular structure file and we do not need them for further processing. As the first step, all covalent bonds within this residue are computed. Note that we do not distinguish between single, double and triple bonds. The computation of covalent bonds can be done based on the atom positions, the atomic numbers, and the covalent atom radii [44].

As result of this preprocessing, each residue is represented by an undirected vertex-labeled graph $G = (V, E, \psi)$, with vertices $a_i \in V$ representing the atoms, and edges $(a_i, a_j) \in E$ representing covalent bonds between atoms $a_i, a_j \in V$. Furthermore, $\psi : V \rightarrow \mathbb{N}$ maps each atom to its corresponding chemical element, that is, $\psi(a_i) \mapsto e_{a_i}$. Next, we try to find a mapping $m$ from a standardized nucleotide graph $G_X \in \{A,C,G,T,U\}$ to the residue graph $G$. This mapping must satisfy that $\forall a_i \in V_X$, $m(a_i) \in V$ and $\psi_X(a_i) = \psi(m(a_i))$, and $\forall (a_i, a_j) \in E_X \implies (m(a_i), m(a_j)) \in E$. Since the nucleotides differ in their structure, only a single mapping can exist that is an isomorphism. However, due to modifications that can occur in the nucleotide structure, not for all residues an isomorphism can be found. For this reason and to improve the performance, we compute the mapping in two steps.

First, we try to find a mapping for the backbone part. For this, we use two standardized backbone structures. The first one is the full DNA backbone part, while the second one contains only the sugar ring with the oxygen atom that connects the backbone with the next backbone. The smaller backbone part without the phosphate group sometimes appears at the end of a strand. We start to compute a mapping for the large backbone structure. If such a mapping does not exist, we try the smaller one. If both backbones cannot be mapped, the residue is not a nucleotide and we proceed to the next. Otherwise, we check for an additional oxygen in $G$ at the second position of the sugar, indicating that the backbone belongs to an RNA strand instead of a DNA one.

In the second step, we try to find a mapping for each of the five different nucleobases to the remaining atoms in $G$. The type is then given by a valid mapping. If more than one mapping exists, we select the mapping for the largest nucleobase. Note that we set the type of the nucleotide to unknown if none of the nucleobases can be mapped. If the overall mapping is not an isomorphism, which means $G$ contains unmapped atoms, the nucleotide is a modified one and is marked as such. Other modifications that can be detected include thymine with an RNA backbone or uracil with a DNA backbone. The connection between nucleobase and backbone is also an indicator of a modification. For example, pseudouridine is connected to the backbone by a carbon atom of the pyrimidine ring instead of the nitrogen. Further modifications can be detected by geometric analysis. In particular, we check the flatness of the rings by fitting a plane to the corresponding atom positions using least squares fitting. For errors larger than 0.05 Å, we mark the nucleotide as modified (also compare to Ref. [31]).

In general, the subgraph isomorphism problem is NP-complete. In practice, however, since the graphs are very small and due to the restrictions in the possibilities by the given bonds and elements, it can be solved efficiently with a brute-force approach that tries each combination by a depth-first search with backtracking (see Tab. 1). In addition, we accomplish the search such that it first
tries some default mappings, which often accelerates the computation such that we obtain linear complexity. Another optimization exploits the fact that within a molecular file the order of the atoms of equal residues is often the same. Thus, once we have found a mapping for a certain nucleotide, we can test this mapping for other nucleotides before starting the expensive search. Furthermore, since the number of atoms and covalent bonds in classical biomolecular simulations is usually constant, the mapping for a molecular trajectory needs to be computed only for a single time step.

4.2 Strands and Base Pairs

In the following, we consider only residues that represent nucleotides. First, we compute the strands of connected nucleotides \( S = \{ S_1, ..., S_k \} \), where \( k \) is the number of strands. Each strand \( S_i \subset \mathbb{N} \) consists of a set of residue indices in the order of their appearance in the strand. We start by computing the covalent bonds between the nucleotides. Due to the previously computed mappings of the nucleotides, only a single pair of atoms between two nucleotides \( i \) and \( j \) needs to be analyzed. Using a 3D grid data structure, the detection of valid neighbors for each nucleotide can be done in constant time.

Then, to compute the base pairs for each nucleotide \( n_i \), all nucleotides \( n_j, i \neq j \) in the close spatial neighborhood need to be investigated. To quickly detect these nucleotides, again a grid data structure is used. After a trivial distance check between \( n_i \) and \( n_j \), the pair types are checked. To do so, we compute all potential hydrogen bonds \( H_{ij} \) between the bases. For each pair type, a subset \( T(H_{ij}) \) of the hydrogen bonds need to be investigated. We allow Watson-Crick, Hoogsteen (A-T, G-C, and A-U) and Wobble base pairs (G-U), see Suppl. Fig. 2. Using the mappings, we can directly analyze whether a certain hydrogen bond exists. Fig. 2 depicts the hydrogen bonds for classical Watson-Crick pairs by dotted lines. The strength of a hydrogen bond can be measured by its length and linearity. Typical hydrogen bond lengths range from 2.8 Å to 4.3 Å (see Refs. [5, 50, 52]). The linearity measures how close the hydrogen atom lies to the line connecting the donor and acceptor atoms. However, we usually do not know the position of the hydrogen atoms. For this reason, we measure for each potential hydrogen bond \( h \) the distance \( d(h) \) between the donor and acceptor atoms as well as the mean of the out-of-plane angles \( \angle(h) \) between the donor-acceptor vector and the planes through the base rings. With \( \angle(h) \) we can roughly approximate the linearity, but more important, we get a measure for the parallelism and the offset between the pairs. In the ideal case, \( \angle(h) = 0 \)°. For \( d(h) > 4.8 \) Å and \( \angle(h) > 60 \)°, we consider the hydrogen bond as too weak and ignore the corresponding base pairs. For all remaining pair types \( T \), we compute an uncertainty value \( \omega_{ij}^T \in [0, 1] \) given by

\[
\omega_{ij}^T = \frac{1}{2|T(H_{ij})|} \sum_{h \in T(H_{ij})} \left( \frac{\max(d(h), 2.8) - 2.8}{4.8 - 2.8} + \frac{\angle(h)}{60} \right),
\]

where 0 corresponds to a strong pairing and 1 to a weak one. For all examples in this paper, we only considered base pairs with an uncertainty value smaller than 0.5.

Note that a base can be paired only with a limited number of other bases. For example, A and G can create at most two pairs at once, a Watson-Crick
and a Hoogsteen pair, while C, T, and U can form only a single one. For this reason, we select the most probable pairs with a Greedy approach. Let \( P \) be the final set of valid base pairs, which is initialized with \( P = \emptyset \). Then, we sort the uncertainty values of all potential pairs of all nucleotides in ascending order. In the next step, we iterate over the sorted pairs and investigate if the pair does not conflict with a pair in \( P \). Two pairs are in conflict if their hydrogen bonds share an acceptor atom or a donor atom that cannot form a hydrogen bond with both pairs at once. In case of a conflict, the pair is rejected, otherwise, it is added to \( P \).

5 3D Visualization

Our 3D visualization of the RNA and DNA structures consists of two parts, the backbone visualization and the base visualization. We represent the backbone by ribbon-shaped strands that are parametrized by \( w \) and \( t \), representing half the width and half the thickness, respectively. The ribbon geometry is defined by a set of 3D sampling points with a 3D orientation at each point as normal of the ribbon (Fig. 4). Each nucleotide creates one sampling point, given by the position of the representative backbone atom (black circle in Fig. 2). The orientation is then defined as the vector from this point to the position of the atom on the base ring that lies farthest away (bold atoms in Fig. 2). However, for Watson-Crick and Wobble pairs, the orientation is adapted to the sampling point of the opposite paired base. To get a smooth representation of the ribbon, we create further samples using cubic spline interpolation for the sampling points and SQUAD interpolation for the normals. For \( n \) nucleotides in the strand and a subsampling degree \( s \), we use \( s(n - 1) + 1 \) samples per strand. We use \( s = 10 \) as default.

For the rendering of the bases, we implemented two visualization models. The first is a classical ladder visualization, where in addition to the backbone
the bases are rendered as ribbon-shaped sticks with a specific pattern at the end of the sticks to easily distinguish the base types. The patterns of the base pairs are complementary, emphasizing the pairing. For the nucleotides A and T/U, round ends are used, and for C and G peaked ends (Figs. 4, 5). Each stick of the ladder is given by a start and an endpoint, a single normal direction and, again, \( w \) and \( t \). The start point is the sampling point of the corresponding backbone ribbon and the end point is given by the ribbon orientation, such that paired bases point into exactly opposite directions. Furthermore, the normal direction of the base is the normal of the plane through the base ring. In case the base is part of a Watson-Crick or Wobble pair, the normal direction is adapted to the mean of both normals.

The second base visualization, called ring model, shows the sugar rings of the backbone and the ring structures of the purines and pyrimidines as filled polygons, surrounded and connected by the ball-and-stick model (Fig. 6). The geometry is directly given by the atom positions of the ring atoms and the covalent bonds.

For the ribbon and the stick visualizations, we developed a triangulation-based and a ray casting-based approach (Fig. 3). While triangular meshes can be efficiently rendered by the GPU and can also be exported to other tools, there is a tradeoff between the number of triangles for smooth structures and the performance for the mesh creation and the rendering. To overcome these limitations, we developed a ray casting approach, which, to the best of our knowledge, is the first such approach. The ray casting is partially based on the approach by Sigg et al [42]. One difference is that we create for each surface patch only a single vertex. The vertex is expanded to an object-aligned quad in the geometry shader for the generation of fragments. This is faster and creates fewer fragments, which makes it more efficient. Another difference is that we need specific ray casting techniques for the ribbon and stick structures. One challenge, for example, is the construction of smooth transitions between two neighboring segments of the ribbon-shaped backbone. In the following, we detail the rendering techniques for the backbone, the base sticks, and the sugar and base rings.

5.1 Backbone

To generate the triangular mesh for the ribbon structure, we use an elliptic cross-section as illustrated in Fig. 4, left. Consider a single backbone strand that was prepared as described above. This strand consist of \( n \) sampling points \( p_i \in \mathbb{R}^3, i = 1, ..., n \), and \( n \) orientations \( \vec{\alpha}_i \in \mathbb{R}^3 \). At each sampling point \( p_i \), we derive an approximated tangent vector \( \vec{t}_i = p_{i+1} - p_{i-1}, \) where \( i + 1 \) and \( i - 1 \) are clamped to the range \([1, ..., n]\). Furthermore, we compute the normalized vectors \( \vec{e}_i \) of \( \vec{\alpha}_i \times \vec{t}_i \). At each point \( p_i \), we create an elliptic sampling of vertices for the ribbon surface. The number of samples steers the smoothness of the ribbon shape but also affects the requirements for the creation, storage, and rendering. In our implementation, we used \( k = 20 \) as default number of vertices per ellipse. The \( i \)th ellipse is created by uniformly rotating \( \vec{e}_i \) around \( p_i \) with the rotation axis \( \vec{t}_i \). The scaling of \( \vec{e}_i \) creates the elliptic shape and is given by \( r_j = \sqrt{w^2 \sin^2 \alpha_j + t^2 \cos^2 \alpha_j}, j = 1, ..., k \), with \( \alpha_j = (j - 1)2\pi/k \). Note that the values of \( r_j \) can be precomputed and used for all ellipses. In the second step, the triangles between two neighboring ellipses are created as a triangle strip.
Finally, at each end, a cap is created. To do so, further ellipses are created, where \( w \) and \( t \) are elliptically reduced until the last ellipse degenerates to a single point. We use \( k/2 \) ellipses, which results in the same smoothness as for the ellipses themselves. A close-up can be seen in Fig. 5 (left).

Instead of an elliptic cross-section, for the ray casting of the ribbons, we use a capsule-shaped cross section because it is faster to compute the distance to a capsule than to an ellipsoid. Consider a capsule at each sampling point with center \( p_i \) and axis \( \vec{e}_i \) (Fig. 7). The length of the capsule equals \( 2w \) and the radius is \( t \). The surface of a ribbon segment is created by infinitely many capsules that result from interpolating the capsules at the sampling points. The idea is to ray cast each segment separately such that they finally fit together without a visual seam. Since a direct computation of the intersection point with a ray is difficult, we use the iterative sphere tracing [14]. In each step, the minimal distance from the current position to the surface is computed. The current position can be moved this distance along the ray without intersecting the surface. After a certain number of steps, the distance either converges to 0, because the ray hits the surface, or the distance increases to infinity. In practice, only a maximal number of steps can be processed and a distance threshold is selected for a valid intersection. For the ray casting of a ribbon segment, in each step, we try to detect the closest capsule and then compute the distance to this capsule. While the latter part can be done analytically, for the detection of the closest capsule we use a heuristic.

In detail, for each segment between the samples \( i \) and \( i + 1 \), a single vertex is created with the vertex attributes \( p_{i-1}, p_i, p_{i+1}, p_{i+2} \) and \( \vec{e}_{i-1}, \vec{e}_i, \vec{e}_{i+1}, \vec{e}_{i+2} \). For the first and the last segment, \( p_{i-1}, p_{i+2} \) and \( \vec{e}_{i-1}, \vec{e}_{i+2} \) are linearly extrapolated. In the geometry shader, for each vertex, the bounding cylinder of the corresponding segment is computed and then a quad is spanned that encloses the cylinder after projection. The computation of the quad is done as described by Lindow et al. [29]. Finally, the sphere tracing is performed in the fragment shader. Consider the line segments \( A = \{ p_i + (2a - 1)w\vec{e}_i | a \in [0, 1] \} \) and \( B = \{ p_{i+1} + (2b - 1)w\vec{e}_{i+1} | b \in [0, 1] \} \) (Fig. 7, left). In each iteration, we com-
Figure 6: A group I ribozyme domain (PDB ID: 1GID). Bases are visualized using the ring model. Colors denote base pair types: Watson-Crick (red), Wobble (yellow), Hoogsteen (blue), reverse Hoogsteen (green).

compute the closest points $p_a$ on $A$ and $p_b$ on $B$ to the current position $p$. Then, we compute the closest point to the line segment $C = \{ p_a + c(p_b - p_a) | c \in [0, 1] \}$. The parameter $c$ for the closest point is used to linearly interpolate the capsules. We assume that the capsule with position $p_m = (1 - c)p_i + c(p_{i+1})$ and axis $\vec{e}_m = (1 - c)\vec{e}_i + c(\vec{e}_{i+1})$ is the closest capsule to the current position $p$. In the last step, the distance $d$ to this capsule is computed, which is the distance from $p$ to the line segment $M = \{ p_m + (2m - 1)w\vec{e}_m | m \in [0, 1] \}$ minus the thickness $t$, see Fig. 7. If $d$ becomes smaller than a threshold $\epsilon$ or we reach a certain number of iterations, the algorithm stops. Otherwise, $p$ is moved along the ray by $0.85 \cdot d$. We move not the full distance because we use a heuristic to detect the closest capsule, see also Suppl. Fig. 5. In practice, we use $\epsilon = 0.001 \text{Å}$ and a maximum of 50 steps. If $d$ is still larger than $\epsilon$ after these steps, we assume that the ray does not intersect the ribbon segment. In case of an intersection, we compute the normal for the shading in $p$ and the correct depth value. The normal $\vec{n}$ of the capsule in $p$ is simply given by the vector from the closest point on $M$ to $p$. Although this normal computation is in principle correct and smooth within a segment, the normals do not change continuously in the region between two segments. Hence, one would easily observe the single segments. To let the ribbon appear as one smooth structure, we apply a similar trick as is used to create a smooth impression of triangular meshes by vertex normal interpolation. However, in our case, we use the tangents at the four corners of the segment. Let $\vec{t}_{i,1,2}$ be the normalized vectors of $p_{i+1} - p_{i-1} = w(e_{i+1} - e_{i-1})$. Furthermore, let $\vec{t}_a$ be the normalized vectors of $(1 - a)\vec{t}_{i,1} + a\vec{t}_{i,2}$ in point $p_a$ and $\vec{t}_b$ of $(1 - b)\vec{t}_{i+1,1} + b\vec{t}_{i+1,2}$ in point $p_b$. Finally, let $\vec{t}_c$ be the normalized vector of $(1 - c)\vec{t}_a + c\vec{t}_b$. To create continuous normals, we project $\vec{n}$ into the plane with normal $\vec{t}_c$. Note that $\vec{t}_c$ changes continuously for all points $p$ even between two
Figure 7: A single ray casting step for a ribbon segment (left) and a stick (right). Left: First, the closest points \( p_a \) on \( A \) and \( p_b \) on \( B \) to the current position \( p \) on the ray are computed. Then, the closest point to the connection \( C \) of \( p_a \) and \( p_b \) is used to define the closest capsule with axis \( M \). Right: The closest point \( p_m \) on the stick axis to the current position \( p \) defines the closest capsule for which the distance \( d_s \) is computed. In combination with the distance \( d_c \) to the cylinder (yellow), the stick pattern is created by moving \( p \) iteratively by \( \max(d_s, -d_c) \) along the ray.

### 5.2 Base Sticks

The base sticks are triangulated in a similar way as the ribbons. At both ends of a stick, an ellipse is sampled orthogonally to the stick axis. We use the same number of samples as for the ribbons but only 0.8\( w \) and 0.8\( t \) as parameters. Then, all vertices of the first ellipse are moved along the stick axis until they lie in the plane that is spanned by \( \vec{t}_i \) and \( \vec{e}_i \) of the corresponding ribbon sample \( p_i \). To create the characteristic patterns, the vertices of the other ellipse are also moved along the stick axis. For the round ends, we move the \( j \)th vertex by \( \pm 0.8w |\sin(\alpha_j)| \) for the convex or concave shape, respectively. For the peaked ends, we use a piece-wise linear zigzag function with a similar shape but scaled by 1.2\( w \). In addition to the triangle strip between the two ellipses, a further strip is used to create a cap at the end of the pattern (Fig. 5).

The ray casting of the sticks is also similar to the one of the ribbons. It becomes a bit easier because we do not have to deal with twisting or multiple segments. On the other hand, we have to create the patterns for the base types. Let \( p_s \) and \( p_e \) be the start and end points of the stick and let \( \vec{d} \) be the normal orientation. Again we use sphere tracing, see Fig. 7, right. For the distance \( d_s \) to the stick shaft, we first compute the closest point \( p_m \) on the line segment given by \( p_s \) and \( p_e \). Then we compute the distance to the capsule with center \( p_m \) and axis \( \vec{d} \times (p_e - p_s) \). Again we use 0.8\( w \) and 0.8\( t \) as width and thickness values. For the round patterns, we also compute the distance...
Figure 8: Large ribosomal subunit (PDB ID: 1S72). Left: The ribosomal RNA with 2876 nucleotides is shown by the ray casted ladder model, colored by strands, while the ribosomal protein is depicted by the molecular surface. Right: 2D graph model of the ribosomal RNA.

d_c to a cylinder with axis \( \vec{o} \) through \( p_e \) and with radius \( 0.8w \). The concave round shape is achieved by the difference of these two shapes, which means the overall distance is \( \max(d_s, -d_c) \), see Fig. 7. For the convex round shape, we need an additional step. Let \( d_l \) be the distance to the elongated shaft by \( 0.8w \) into the direction of the pattern. The intersection of the elongated shaft and the cylinder creates the pattern. Then, the union of this intersection with the shaft creates the overall shape of the stick, which means the overall distance is \( \min(d_s, \max(d_l, d_c)) \). For the peaked patterns, we create two planes that cut out the triangular shape. The normals of the planes are orthogonal to \( \vec{o} \) and the base of the triangle is constructed such that it contains \( p_e \) and lies orthogonal to the stick axis. For the concave shape, the planes are oriented such that the triangle points to \( p_s \), otherwise into the opposite direction. In the convex case, we use again the elongated shaft. Let \( d_p \) and \( d_q \) be the distances to the two planes, then the overall distance is \( \max(d_l, \max(d_p, d_q)) \). In the concave case, the overall distance is \( \max(d_s, \min(d_p, d_q)) \). In contrast to the ribbons, we compute the normals by measuring the changes of the distance in \( x \), \( y \), and \( z \)-direction around the intersection point. This is an approximation of the analytical normal but it has the advantage that an impression of slightly round edges is created and pixel artifacts at sharp edges are avoided.

5.3 Sugar & Base Rings

We use classical GPU-based ray casting to render spheres and cylinders of the ring structures and the bond that connects the sugar and base rings. Except for some minor improvements, the approach is equal to the one presented by Sigg et al. [42]. Since the rings are convex and usually flat, we use triangles to render the inner parts of the rings. To make the rings look nicer, we use two layers of triangles that have been moved apart from each other along the ring normal. Fig. 6 shows the ring visualization for a group I ribozyme domain.
6 2D Visualization

To support interactive visual exploration of DNA and RNA structures, we implemented the two most important 2D visualizations models to study nucleic acids. This is of particular interest for large RNA datasets, where the 3D visualization is often cluttered.

To create these visualizations, we consider an edge-labeled graph the vertices of which are the nucleotides and the edges represent connections between neighboring nucleotides along the strand as well as Watson-Crick and Wobble base pairs. We call the first type of edges backbone edges and the second type base edges. This graph directly results from the computations described in Sect. 4.2.

6.1 Linear Model

The first visualization is often called linear model (Fig. 1, left). Here, the nucleotides are placed on a straight line in the order of their appearance along the nucleic acid strand and the base pairings are depicted by arcs, that is semicircles, connecting the respective nucleotides. In this visualization, stems can be easily identified as bundles of neighboring arcs, whereas loops are represented by cycles in this graph. Intersections of arcs only occur when the molecular structure contains pseudoknots. Often, these are formed by only a few pairs (see the blue arc in Fig. 1, left). For several applications, it is important to mark or even remove pseudoknots. We use a greedy heuristic that starts with the most critical base pair whose arc has the most intersections. We mark the edge in the graph and consider it as removed. Then we proceed with the next most critical pair until no remaining critical arc can be found. See the work by Smit et al. [43] for other approaches.

6.2 Graph Model

The second visualization shows the RNA as a planar graph model without any intersections. Examples are shown in Fig. 1, right, and Fig. 8, right. Such a graph always exists after removing all pseudo-knot edges. The idea is to compute a graph representation that tries to preserve the local neighborhood of the 3D structure in the 2D graph representation. To do so, the nucleotides are arranged in such a way that the edges have constant lengths, the base edges lie orthogonal to the backbone edges, and the nucleotides of a loop are arranged on circles. As result, one obtains the classical ladder visualization that allows the user to quickly analyze stem and loop structures (Fig. 1, right). However, it is not guaranteed that such a graph is free of intersections by just applying the rules. In such cases, we need to deform the graph to get rid of the intersections by accepting minor distortions. Several people have worked on this problem for more than 40 years (see Sect. 3.2).

In principle, our approach is based on the work by Hecker et al. [15]. In contrast to other algorithms, this one generates visually appealing graphs without intersections. However, this comes at the price of high computational costs. The algorithm starts from a circle visualization [53], where all nucleotides are arranged on a circle and the base edges build chords that do not intersect. Then, a gradient descent approach is applied with a penalty function that quantifies the difference between the current layout and the ideal solution. The penalty
function uses the squared difference \((d_{ij} - \tilde{d}_{ij})^2\) for bases \(i,j\) connected by a backbone or base edge and for all bases inside a loop. While \(d_{ij}\) is the current distance, \(\tilde{d}_{ij}\) represents the ideal distance which is constant for backbone and base edges and has the ideal chord length of the corresponding circle for bases inside loops. In addition, a quartic term is used for the repulsion between close bases and bases close to edges. To accelerate the computation that could easily take half a day for large molecules, we propose two major improvements.

One expensive part is the evaluation of the repulsion forces because for each base all other bases and all edges need to be processed. However, only close neighboring bases and edges are important for the repulsion. For this reason, we restrict the repulsion to a certain spatial neighborhood, which allows us to collect the relevant bases and edges in constant time using a 2D grid data structure.

Another issue is the initialization from the circle visualization, which is not optimal since it can create very small distances between bases and edges, especially for large molecules. In order to avoid intersections during relaxation, the step size needs to be rather small, which leads to long running times. We solve this problem by modifying the circle initialization before running the gradient descent approach. To get rid of the small distances we locally bulge the circle in the following way. First, all base edges of the graph are sorted according to their length in ascending order. Then, the edges are iteratively processed. Each edge divides the circle into two parts. The smaller part, which means the one with fewer nucleotides, is then realigned on a half circle using the base edge as diameter. This creates bulges that remove the small distances without creating intersections (Suppl. Fig. 9, 17-25 and Suppl. Video).

To achieve high interactivity, the gradient descent approach for the 2D graph visualization is computed in parallel with a permanent update of the view. This allows the user to quickly get an overview of the structure while aligning the view over time. In particular, the user can decide to stop the optimization at any time.

7 Interactive Exploration

To explore complex DNA and RNA structures, we linked all described visualization techniques: the 2D line model, the 2D graph model, and the 3D visualization. We implemented several interaction techniques for selection and filtering that can be handled by all of them instantaneously. In particular, the user can move the mouse over nucleotides to highlight them in all models or to show properties stored for the corresponding residue. Moreover, highlighted nucleotides can be permanently marked for further investigations. Within the 2D visualizations, the user can filter nucleotides either with a lasso tool or a special circular selection tool. The filtering can then be modified with the current selection using Boolean operations. To link the different views, we implemented a data structure that maintains the current state at a single position. The data structure stores, for example, the currently highlighted, selected or filtered nucleotides. All view models are registered at this data structure and can either send or receive signals about modifications of the current state. When a model receives signals, the corresponding changes are immediately applied. On the other hand, a module that sends signals for modifications does not apply them
Table 1: Performance measures for computing all information necessary for the 3D rendering, up to sending the information to the graphics card. All timings are given in ms. The table contains the following information: (1) PDB ID plus the type of the molecule (D - DNA, R - RNA, H - hybrid); (2) number of residues (#Res); (3) number of nucleotides including number of modified bases given in parentheses (#Nuc); (4) computation of the nucleotides (Nuc.); (5) strand and base pair computation (Sec.); (6) preprocessing for mesh rendering (Mesh.); (7) preprocessing for ray casting (Ray.); (8) Update rates (UR) per second for mesh geometry (first number) and ray casting (second number).

<table>
<thead>
<tr>
<th>PDB ID</th>
<th>#Res</th>
<th>Computation</th>
<th>Geometry</th>
<th>UR</th>
</tr>
</thead>
<tbody>
<tr>
<td>D 5L6L</td>
<td>996</td>
<td>54 (0)</td>
<td>3.5</td>
<td>0.1</td>
</tr>
<tr>
<td>R 1EHZ</td>
<td>245</td>
<td>76 (14)</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>R 2GIS</td>
<td>189</td>
<td>95 (1)</td>
<td>1.8</td>
<td>0.3</td>
</tr>
<tr>
<td>R 3DS7</td>
<td>771</td>
<td>136 (0)</td>
<td>3.9</td>
<td>0.4</td>
</tr>
<tr>
<td>R 4RGE</td>
<td>277</td>
<td>167 (3)</td>
<td>2.5</td>
<td>0.5</td>
</tr>
<tr>
<td>H 4OO8</td>
<td>2846</td>
<td>236 (0)</td>
<td>13.2</td>
<td>0.5</td>
</tr>
<tr>
<td>D 5JRG</td>
<td>1322</td>
<td>289 (2)</td>
<td>8.5</td>
<td>0.8</td>
</tr>
<tr>
<td>R 1GID</td>
<td>350</td>
<td>316 (0)</td>
<td>3.4</td>
<td>0.9</td>
</tr>
<tr>
<td>R 3BWP</td>
<td>373</td>
<td>356 (7)</td>
<td>3.7</td>
<td>1.1</td>
</tr>
<tr>
<td>R 1ST2</td>
<td>14705</td>
<td>2876 (5)</td>
<td>93.4</td>
<td>9.4</td>
</tr>
<tr>
<td>R 5AF1</td>
<td>11467</td>
<td>4801 (53)</td>
<td>76.7</td>
<td>18.0</td>
</tr>
<tr>
<td>R 4U4O</td>
<td>35860</td>
<td>10398 (1)</td>
<td>168.8</td>
<td>43.8</td>
</tr>
</tbody>
</table>

System: Intel Core i5-4690K 3.50 GHz, 4 Cores.

directly but waits until it gets the same signal as the other views by the maintainer data structure. Thus, we ensure that all views are always synchronized. Whenever possible, the maintainer data structure tries to send only the relevant changes to achieve low update times.

8 Results

To measure the performance and to evaluate our algorithms, we used datasets of different size and with different challenges from the PDB [3].

8.1 Evaluation

In Tab. 1, for each dataset we report the number of nucleotides that were detected by our method as well as those that were found to be modified or incomplete. We compared these results with the latest version of DSSR [31]. Our approach is able to correctly detect all regular nucleotides and most of the modified and undefined nucleotides. In the following, we describe the minor differences. For dataset 4RGE, we detected 3 modified uracil nucleotides that were not labeled as modified by DSSR. These nucleotides have a DNA backbone instead of an RNA one. Dataset 3BWP contains 7 nucleotides that only consist of the backbone part without bases. While our approach marks these
as undefined, in DSSR they are not detected at all. Furthermore, in 5AFI we mark 3 nucleotides as undefined, while these are detected as a modified uracil by DSSR. This is due to the base containing sulfur instead of oxygen, so they possibly are sulfur analogs of uracil.

We also compared the results of our base pair detection (Suppl. Tab. 1). We determined all Watson-Crick, Hoogsteen, and Wobble pairs, and the reverse versions of the first two. For most of the datasets, our method returned the same results as DSSR. In particular, both approaches never created contradicting results, which means all common base pairs had identical pair type. In general, our geometrical approach generates slightly more base pairs compared to DSSR. However, when investigated both, the base pairs determined by DSSR but not by our approach and vice versa, we found that most of these pairs are borderline cases, where the decision was made depending on the threshold of the geometrical heuristic. Only in a few cases, the differences were not clear for both approaches, see Suppl. Fig. 3.

8.2 Performance

To measure the performance, we used a system with an Intel Core i5-4690K (3.50 GHz, 4 cores) with 8 GB RAM and an NVIDIA Geforce GTX 1080 graphics card. All CPU algorithms were parallelized using OpenMP. The default screen resolution for the rendering was 3440 x 1440. We switched on FXAA anti-aliasing in the driver, which improves the visual quality, particularly for ray casting.

The performance of the 3D visualization mainly depends on three parts: (1) the detection of the nucleotides and the computation of the strands and base pairs; (2) the construction of the geometric information for the rendering; and (3) the rendering of the geometry itself. The computation times for the first two parts are given in Tab. 1, where the last column provides the update rates for dynamic data. Further details can be seen in Fig. 9. The meshing and ray casting was performed with the default parameters given in the corresponding sections.

For the performance analysis of the nucleotide detection, we used the worst case, which means we switched off the optimization for testing a default or previous successful mappings (see Sect. 4.1). Besides the number of residues, their complexities mainly influence the performance. While small residues like water or most ions can be quickly rejected, larger ones require more investigations. Thus, the computation time per residue can vary a lot, see the dark green bars in Fig. 9. This is not a problem since for molecular dynamics trajectories, this computation needs to be done only once. The computation of the strands and base pairs is usually much faster. Both parts scale almost linearly with the number of nucleotides, see the green and light green bars in Fig. 9.

The second part, the geometry construction, is further split into a modeling part and the generation of the backbone strands and the sticks or rings. All steps scale linearly with the number of nucleotides both for ray casting and meshing (see Fig. 9). The modeling part computes the analytical description of the geometrical model and is equal for meshing and ray casting (see Sect. 5). Here, the most expensive step is the interpolation of the sampling points and directions. Since the meshing approach needs to discretize the geometrical model, much more data is generated. For the largest dataset (4U4O), the complete
Figure 9: Computation times for the nucleotides, backbone strands, and base pairs as well as for the geometry construction of the meshing and the ray casting. The geometry construction is subdivided into the modeling part and the generation of the strands and sticks. Except for the nucleotide computation, which is given per residue, all other timings are given per nucleotide.

mesh of strands and sticks consists of \( \sim 3 \text{ Mio. vertices and } \sim 5 \text{ Mio. triangles.} \)

Hence, ray casting outperforms meshing (see Fig. 9).

The third part comprises the rendering on the GPU. For the classical meshing of the ladder model, we did not observe any limitations. Even the largest dataset with \( \sim 5 \text{ Mio. triangles} \) can be rendered with more than 60 fps, independent of the camera setup. To approach the limitations, we duplicated the dataset 6 times such that it contained \( \sim 30 \text{ Mio. triangles} \). Yet, we still achieved almost 13 fps. Thus, for this rendering approach, the vertex processing is probably the bottleneck. In contrast, for the ray casting of the ladder model, the amount of work per fragment is much higher. Thus, the rendering performance depends highly on the overall number of fragments that need to be processed, which in turn depends on the data, the screen resolution, and the camera setup. For our screen resolution, the worst case frame rate for the largest dataset was 15 fps. For the medium-sized datasets (\( \sim 3000 \text{ nucleotides, Fig. 8} \)) we achieved at least 30 fps and for the smaller ones (\( \sim 350 \text{ nucleotides} \)) even more than 60 fps. In addition, we also tested the performance for the 6-times-increased largest dataset and still obtained 10 fps. Hence, the ray casting seems to scale slightly better with the size of the data. When using the ring visualization instead of the sticks, we did not observe any noticeable performance changes, neither for the mesh-based nor the ray casting approach.

For several small dynamic datasets, the overall performance was always far beyond 60 fps. The visualization of a larger dynamics simulation of an RNA molecule with \( \sim 3000 \text{ nucleotides} \), was achieved with 11 fps for the meshing and between 30-50 fps for the ray casting. This confirmed our static performance analysis. Dynamic datasets are shown in the accompanying video (Suppl. Mat.).
While the pure rendering of the 2D plots does not affect the overall performance, the optimization of the graph model is clearly the bottleneck. We are able to generate the plots for small tRNA molecules on demand within 2-3 s. For mid-sized molecules with $\sim 500$ nucleotides, the plot can be still generated within 20-30 s. However, for larger molecules with thousands of nucleotides, reasonable plots can require up to an hour, see Suppl. Tab. 3. Nevertheless, usually, the plot does not need to be fully optimized in order to start the analysis. Furthermore, as for the nucleotide detection, we need to compute the layout only once and can quickly update it on demand, because the changes are restricted to appearing and disappearing base edges.

9 Discussion & Conclusion

In this paper, we have investigated how far we can push the performance of 2D and 3D DNA and RNA visualizations. To this end, we have covered the whole pipeline starting from importing molecular structure files, the determination of the nucleotides, the computation of strands and base pairs up to the visualization of tightly linked 2D and 3D renderings. We focused on visual quality and performance with the aim of allowing interactive visualization of even large molecular structures and dynamics trajectories. Our approach detects all standard and many modified nucleotides as well as the most common base pairs. Further special cases could be easily added. Yet, the system that we developed should not be seen as a replacement for well established tools like DSSR. Rather, it shows what can be achieved with modern techniques both in terms of computation and rendering. We believe that interactivity plays an important role in the visual exploration of DNA and RNA structures and that domain experts would benefit from the improvements presented in this paper.

Concerning the detection of nucleotides and the computation of base pairs, we have chosen approaches that are both fast and have very few requirements. For the nucleotide detection, we only need the element type and the grouping of atoms into residues. For the base pair computation, we follow an almost purely geometrical approach that does not need information about the steric constellations of typical base pairings stored in databases and used by most base-pair detection algorithms. Nevertheless, our evaluation (Sect. 8.1) shows that with the proposed approaches we get very similar results to the ones obtained by tools like DSSR in terms of quality. In terms of speed, DSSR needs much longer run times. For example, for 4U4O, DSSR needed $\sim 15$ min for the secondary and tertiary structure analysis [31], while our algorithm only needs $\sim 0.2$ s (see Tab. 1). From our tests, we could not deduce that more complicated algorithms provide more information. However, this has to be evaluated more extensively in future work.

In terms of 2D visualizations, we implemented two popular models, the linear and the graph model. While the first is trivial to create and can be rendered quickly, the second, which is usually favored, requires more effort to generate. From the known algorithms, we identified the one by Hecker et al. [15] to produce the best results. With minor modifications (Sect. 6.2), we obtained a speed-up that leads to almost instantaneous results for small molecular structures and still acceptable ones for very large ones (like 4U4O). By running the computation in parallel, interaction with the 2D visualization is possible at any time even when
the optimization has not yet converged. Since we basically use the algorithm by Hecker et al. with a better starting condition, similarly it might happen that the algorithm gets stuck in a local minimum. Yet, for the purpose of interactive visualization, this is not a problem.

While making several contributions, we see the main contribution of this paper in the presentation of the first ray casting approach to render the backbone and base pairs of nucleic acids. Furthermore, the backbone rendering can be applied directly to the ribbon rendering of proteins. Our ray casting creates a high visual quality and is particularly well-suited for dynamic data since only little information needs to be computed on the CPU and sent to the graphics card, thus allowing for high update rates. On the other hand, for our meshing approach, we focused on visual quality instead of performance. Using geometry and/or tessellation shaders to generate the mesh on the GPU [16], probably achieves the same performance as ray casting. However, correctly handling all patterns, such as the ends of the strands or the different stick patterns, without visual seams can be quite complicated, especially if level-of-detail is involved. While thoroughly testing the ray casting of single segments, we observed artifacts when the bending within the segment was too high, see Suppl. Fig. 4. However, for real data, such artifacts never appeared, see Suppl. Fig. 6.

Besides the fast rendering using ray casting, for the first time, we implemented a 3D ladder model for depicting base pairs with complementing ends to allow an easy identification of the nucleotides. Instead of the often used block visualization of the base pairs, we provide a ring model that gives comparable information but is closer to the atomic structure. Our last contribution is the tight linking between 2D and 3D visualizations. We did this as a proof of concept, especially for highlighting and filtering substructures. Using the proposed functionalities as a basis, further interaction techniques will be developed in collaboration with domain experts.

For future work, we plan to look in detail at the base pairings. In particular, it could be interesting to explore the suitability of machine learning algorithms to either learn the parameter thresholds for the geometric heuristics or to completely train the pair types based on pure structural and geometrical molecular information. Furthermore, for dynamic data, we plan to compute and visualize the uncertainty of base pairs.
Figure 10: Streptococcus pyogenes Cas9 complexed with guide RNA and target DNA as triangulated ladder model (PDB ID: 4O08).
References


Supplementary Material

Figure 11: Secondary structure of Caulobacter crescentus (blue and yellow) bound to the operator DNA double helix (PDB ID: 5L6L).

Introduction
Here we present material to support the statements made in the paper. In particular, we show images of all molecular structures that we mentioned in the main manuscript but that did not fit into it. All of these structures are from the Protein Data Bank (PDB [3]), which we do not explicitly state for each molecular structure. With these images, we also show the beauty and variety of DNA and RNA.

The supplementary material is organized in eight sections:

- Base pair computation: comparison with DSSR
- Limitations of ribbon ray casting
- Triangulation: complexity in terms of number of vertices and triangles
- Comparison of 3D visualizations created with 6 other widely used tools and our tool
- 2D Graph Model: timings compared to RNDdlf, multiple strands
- Pseudo code for ray casting of the 3D ladder model
- Collection of further results
- Informal user feedback
Table 2: Base pairs computed with our approach compared to those of DSSR [31]. For each molecule, the PDB ID is given with a leading character that identifies DNA (D), RNA (R), or a hybrid (H). Besides the overall number of nucleotides (#Nuc), the number of modified and undefined base pairs is given in brackets. This is followed by the number of Watson-Crick (WC), reverse Watson-Crick (rWC), Hoogsteen (Hoog.), reverse Hoogsteen (rHoog.), and Wobble base pairs. The red numbers indicate pairs that were only detected by DSSR and the green numbers pairs that were only detected by our approach.

<table>
<thead>
<tr>
<th>PDB ID</th>
<th>#Nuc</th>
<th>#WC</th>
<th>#rWC</th>
<th>#Hoog.</th>
<th>#rHoog.</th>
<th>Wobble</th>
</tr>
</thead>
<tbody>
<tr>
<td>D 5L6L</td>
<td>54 (0)</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R 1EHZ</td>
<td>76 (14)</td>
<td>20</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>R 4P5J</td>
<td>84 (1)</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>R 2GIS</td>
<td>95 (1)</td>
<td>28</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>R 3DS7</td>
<td>136 (0)</td>
<td>46</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>R 4RGE</td>
<td>167 (3)</td>
<td>60</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>H 4O08</td>
<td>236 (0)</td>
<td>84</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>D 5J0N</td>
<td>476 (0)</td>
<td>232 (1,5)</td>
<td>0</td>
<td>3 (3)</td>
<td>1 (1)</td>
<td>0</td>
</tr>
<tr>
<td>D 5JR</td>
<td>289 (2)</td>
<td>142</td>
<td>0</td>
<td>1 (1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R 1GID</td>
<td>316 (0)</td>
<td>84</td>
<td>0</td>
<td>6 (2)</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>R 3BWP</td>
<td>356 (7)</td>
<td>100</td>
<td>2 (2)</td>
<td>0</td>
<td>5 (1)</td>
<td>14 (3)</td>
</tr>
<tr>
<td>R 1S72</td>
<td>2876 (5)</td>
<td>737 (1,1)</td>
<td>10 (1)</td>
<td>12 (3)</td>
<td>40 (4)</td>
<td>77 (1,1)</td>
</tr>
<tr>
<td>R 1YJW</td>
<td>2876 (5)</td>
<td>735 (2)</td>
<td>10 (1)</td>
<td>11 (2)</td>
<td>40 (4)</td>
<td>81 (1,2)</td>
</tr>
</tbody>
</table>

Base Pair Computation

Here we present a more detailed comparison with DSSR [31] concerning base pair computation. Tab. 2 clearly supports our claim that we obtain very similar results. Fig. 12 shows all types of base pairs that we are currently able to detect and that we used for the comparison with DSSR. Additionally, examples for differences between the approaches are given in Fig. 13.

Ribbon Ray Casting

For the ray casting of the ribbons, we provide further details on the artifacts, which may occur in principle due to the geometric approximations, but which we have never observed in practice. Fig. 14 shows different setups of bending and twisting on a single ribbon segment. Furthermore, in Fig. 15, we analyzed the influence of the scale of the step size on the visual result. Finally, Fig. 16 demonstrates the effect of our tangent interpolation and shows again practical situations with strong bending and twisting.

Triangulation

In Tab. 3, we provide further information about the triangulation, such as the number of vertices and the number of triangles. The number of vertices and triangles scale linearly with the number of nucleotides plus a constant overhead for the smooth ends of the strands. For a dataset that contains $m$ strands with
Table 3: Triangulation timings and mesh information. All timings are given in ms. The table contains the following information: (1) PDB ID plus the type of the molecule (D - DNA, R - RNA, H - hybrid); (2) number of nucleotides (#Nuc.); (3) computation of the geometric model (Geo.) (4) triangulation of the strands (Strands); (5) triangulation of the sticks (Sticks); (6) sum of all timings (Overall); (7) number of vertices in the mesh (#Vert.); (8) number of triangles in the mesh (#Tri.).

<table>
<thead>
<tr>
<th>PDB ID</th>
<th>#Nuc</th>
<th>Computation</th>
<th>Geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Geo.</td>
<td>Strands</td>
</tr>
<tr>
<td>D 5L6L</td>
<td>54</td>
<td>0.1</td>
<td>1.2</td>
</tr>
<tr>
<td>R 1EHZ</td>
<td>76</td>
<td>0.1</td>
<td>1.5</td>
</tr>
<tr>
<td>R 2GIS</td>
<td>95</td>
<td>0.1</td>
<td>2.1</td>
</tr>
<tr>
<td>R 3DS7</td>
<td>136</td>
<td>0.2</td>
<td>2.7</td>
</tr>
<tr>
<td>R 4RGE</td>
<td>167</td>
<td>0.3</td>
<td>3.5</td>
</tr>
<tr>
<td>H 4OO8</td>
<td>236</td>
<td>0.3</td>
<td>5.2</td>
</tr>
<tr>
<td>D 5JRG</td>
<td>289</td>
<td>0.3</td>
<td>6.4</td>
</tr>
<tr>
<td>R 1GID</td>
<td>316</td>
<td>0.4</td>
<td>6.9</td>
</tr>
<tr>
<td>R 3BWP</td>
<td>356</td>
<td>0.5</td>
<td>7.9</td>
</tr>
<tr>
<td>R 1S72</td>
<td>2876</td>
<td>3.0</td>
<td>64.7</td>
</tr>
<tr>
<td>R 5AFO</td>
<td>4801</td>
<td>6.7</td>
<td>109.4</td>
</tr>
<tr>
<td>R 4U4O</td>
<td>10398</td>
<td>12.1</td>
<td>241.6</td>
</tr>
</tbody>
</table>

System: Intel Core i5-4690K 3.50 GHz, 4 Cores.

\[ n_1, \ldots, n_m \text{ nucleotides per strand, the overall number of vertices } v \text{ is given by} \]
\[
v = k \cdot \sum_{i=1}^{m} \left( n_i - 1 \right) s + 1 \quad + \quad m \cdot \left( k \cdot (k - 2) + 2 \right) \quad + \quad \sum_{i=1}^{m} n_i \cdot 3k. \quad (1)\]

Recall that \( s \) defines the detail for the interpolation of the strand curve, where we used a default of \( s = 10 \), and that \( k \) is the detail of the elliptical cross-section of the strands and the sticks, where we used a default value of \( k = 20 \). The overall number of triangles \( t \) is given by
\[
t = 2k \cdot \sum_{i=1}^{m} \left( n_i - 1 \right) s \quad + \quad m \cdot 2(k^2 - k) \quad + \quad \sum_{i=1}^{m} n_i \cdot (3k - 2). \quad (2)\]

**Comparison with 3D Visualizations in Other Tools**

A thorough comparison of our 3D visualizations with other approaches is difficult since we do not have detailed timings for the other tools and are also lacking information about the algorithms. Therefore, in this section, we only do an informal evaluation by giving some information about our experience with six tools that are widely used for molecular visualization and that provide DNA
and RNA visualizations which are most closely related to our proposed models. The tools are VMD [19], Chimera [37], PyMol [11], Jmol [12], NGL Viewer [38] from the PDB, and the Autodesk molecule viewer [2].

While the NGL viewer and the Autodesk molecule viewer run completely in the browser, Jmol, PyMol, Chimera, and VMD are stand-alone tools. Furthermore, with JSmol a web version of Jmol is provided that also runs in the browser. However, Jmol is approximately 3-6 times faster than JSmol. From the investigated six tools, only the Jmol versions in combination with DSSR and Chimera provide a real ladder visualization where the base sticks are aligned according to the pairs. This is of particular interest because the user can easily identify stems and helices that are usually more stable than other parts of the structure. Furthermore, none of the tools provide patterns for the different base types although this is quite often used in hand-drawn depictions to easily detect base types even without coloring them. In addition, most of the tools do not adequately close the strands with a cap, which is of course only a minor visual drawback. In the following, we will go into the details about each tool. In Fig. 17, we provide renderings of the same molecular structure for all six tools. In comparison, Fig. 18 shows renderings of the same structure with our approach.

The Autodesk molecule viewer represents nucleic acids by a triangulation approach. The backbones are depicted as ribbon-shaped strands and the bases as cylindrical sticks with round caps. The viewer achieves a real-time performance for static structures, even with thousands of nucleotides. However, the viewer doesn’t seem to be designed to work with dynamic data, at least we were not able to load molecular dynamics trajectories. Furthermore, the visual quality could be improved by a higher mesh resolution and smooth ends at the strands. The circular shape of the sticks does not allow the user to estimate the orientation of the base rings, which is important to detect potential stacks. In addition, we detected some shading artifacts at the silhouettes of the mesh. Finally, the viewer does not detect modified nucleotides. For this reason, they are depicted by the ball-and-stick model, see Fig. 17.

We tested the NGL viewer directly from the PDB website. It is quite similar to the Autodesk molecule viewer but provides a better visual quality with a higher mesh resolution and no shading artifacts. In addition, it is able to detect modified nucleotides. Again, however, the NGL viewer does not seem to support dynamic data.

VMD provides the same model as the Autodesk molecule viewer and the NGL viewer. It has a similar quality and performance as the NGL viewer but supports dynamic data. However, as the other two tools, it does not compute the base pairs and, hence, does not align the sticks according to this information. For a similar mesh quality, the performance for dynamic datasets is comparable to our triangulation approach. Note, however, that we have to compute the pair information in each time step, which is not done by VMD.

In contrast to the previous tools, PyMol has a cylindrical shape for both the backbone strands and the sticks. The visual quality and the performance for large static datasets is similar to VMD and our approach for the triangulation. It also supports dynamic data but the performance is slightly worse compared to VMD and our triangulation. In combination with DSSR, PyMol also provides the reference frame visualization, however, the base pairs are not depicted.

Chimera has a ladder model for nucleic acids that shows the backbones
as ribbon-shaped strands and the bases as cylindrical sticks that are aligned according to the detected base pairs. This is realized by a triangular mesh with high resolution. However, the pairs seem to be approximated based on a quite simple heuristic that leads to many false positive pairs. Chimera also supports several other DNA and RNA visualizations such as the ring model or the reference frame model. The rendering performance is similar to our triangulation-based approach.

DSSR provides a plugin for Jmol to generate a ladder model that is similar to Chimera and our approach. In addition, DSSR can also visualize the ring model of nucleic acids. However, the visual quality, especially of the strands and sticks, is worse compared to the other tools and our approach. The strands are composed by many spheres to achieve an impression of a cylindrical shape. Due to the many spheres, the rendering performance is also worse compared to the other techniques. Furthermore, all potential hydrogen bonds between bases are visualized as pairs. In contrast, our approach aligns only the Watson-Crick and Wobble pairs that really build the ladder stacking in the DNA and RNA molecular structures.

### 2D Graph Model

In the paper, we only showed the result of the 2D graph model generation but neither the original starting position used by Hecker et al. [15] nor our modified version of it. Both starting positions as well as the final result for the RNA structure of 3BWP are shown in Fig. 19.

In Tab. 4, we present timings for the generation of the 2D graph layout compared to RNAdlf, the original approach by Hecker et al. [15]. We took several datasets of different size from the PDB and ensured that both algorithms used the same base pairs. We started with the optimization approach by RNAdlf and measured the time until the result was satisfying. Afterwards, we optimized the same structures with our approach until the result was as similar as possible to the result of RNAdlf. Note that we used the Linux version of RNAdlf which seems to be much faster than the windows version, based on our experience. In Figs. 27-35, we provide details about the progress of the plot generation with

**Table 4: Timings for the generation of the 2D graph layout compared to RNAdlf [15].** The timings are given in hours, minutes, and seconds (hh:mm:ss).

<table>
<thead>
<tr>
<th>PDB ID</th>
<th>#Nuc</th>
<th>Type</th>
<th>RNAdlf [15]</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>2NOQ</td>
<td>317</td>
<td>Internal Ribosome Entry Site</td>
<td>00:00:41</td>
<td>00:00:19</td>
</tr>
<tr>
<td>2OM7</td>
<td>460</td>
<td>Ribosomal RNA</td>
<td>00:02:18</td>
<td>00:00:26</td>
</tr>
<tr>
<td>1GRZ</td>
<td>494</td>
<td>Ribozyme</td>
<td>00:02:26</td>
<td>00:00:45</td>
</tr>
<tr>
<td>1FFZ</td>
<td>500</td>
<td>Ribosomal RNA</td>
<td>00:03:48</td>
<td>00:02:37</td>
</tr>
<tr>
<td>2NZ4</td>
<td>608</td>
<td>Ribozyme</td>
<td>00:04:52</td>
<td>00:00:35</td>
</tr>
<tr>
<td>2RKJ</td>
<td>964</td>
<td>not specified RNA</td>
<td>00:38:00</td>
<td>00:01:00</td>
</tr>
<tr>
<td>1X8W</td>
<td>968</td>
<td>Ribozyme</td>
<td>00:35:00</td>
<td>00:00:46</td>
</tr>
<tr>
<td>1N36</td>
<td>1505</td>
<td>30S Ribosomal RNA</td>
<td>01:20:00</td>
<td>00:10:00</td>
</tr>
<tr>
<td>1YL4</td>
<td>1726</td>
<td>30S Ribosomal RNA</td>
<td>02:15:00</td>
<td>00:15:00</td>
</tr>
</tbody>
</table>

System: Intel Core i5-4690K 3.50 GHz, 4 Cores.
RNAIdlf (left column) and our approach (right column) for each dataset in the table. Especially for very large datasets, we found that it can be superior for both approaches to not use the fully optimized graph because it can become too dense.

Note that most RNA datasets consist of multiple strands. Sometimes, strands are even completely independent, which means they do not form any base pair with another strand. In our tool, we allow to create the 2D graph layout for each strand separately but also for connected components based on the pairs. In RNAIdlf, and most other tools, all strands are connected in the 2D graph to a single strand. The connection is based on their order in the file, even if the strands do not share a base pair at all or lie at completely different positions in 3D. To compare the layouts we implemented the same approach but we did not visually connect the strands in the plots to show the different components. In Figs. 19, 10, 25, and 26, we colored the graphs according to the different strands.

**Pseudo Code for Ray Casting of 3D Ladder Model**

In the following, we provide some pseudo code for the ray casting of the ribbon segments and the sticks. Note that the pseudo code is written to understand the functionality and to reuse as much code as possible. To get the full performance it needs to be slightly restructured. Especially, the differentiation between the stick patterns should be avoided in each loop pass of the distance computation.

We start with a helper function to compute the parameter on a line segment, given by \( l_s \) and \( l_e \), that represents the closest point to a given point \( p \):

1: function \( \text{LINEVALUE}(p, l_s, l_e) \)
2:     \( \vec{n} = l_e - l_s \)
3:     return \( \max(0, \min(1, \langle p - l_s, \vec{n} \rangle / \langle \vec{n}, \vec{n} \rangle)) \)
4: end function

The following function provides the ray casting algorithm to get the intersection point \( p \) with surface normal \( \vec{n} \) for the \( i \)th ribbon segment with width \( w \) and thickness \( t \) based on the view ray with start \( r_s \) and normalized direction \( \vec{r}_d \).

1: function \( \text{CASTRIBBON}(p_{i-1}, p_i, p_{i+1}, p_{i+2}, \vec{e}_{i-1}, \vec{e}_i, \vec{e}_{i+1}, \vec{e}_{i+2}) \)
2:     \( s_{\text{max}} := 50 \)
3:     \( \epsilon := 0.001 \)
4:     \( p := r_s \)
5:     \( p^+ := p_i + w\vec{e}_i \)
6:     \( p^- := p_i - w\vec{e}_i \)
7:     \( p^+_e := p_{i+1} + w\vec{e}_{i+1} \)
8:     \( p^-_e := p_{i+1} - w\vec{e}_{i+1} \)
9:     for \( i = 1 \ldots s_{\text{max}} \) do
10:        \( a := \text{LINEVALUE}(p, p^+_e, p^-_e) \)
11:        \( b := \text{LINEVALUE}(p, p^+_e, p^-_e) \)
12:        \( p_0 := p^+_e + a(p^-_e - p^+_e) \)
13:        \( p_0 := p^+_e + b(p^-_e - p^+_e) \)
14:        \( c := \text{LINEVALUE}(p, p_0, p_0) \)
15:        \( p_m := (1 - c)p_i + c p_{i+1} \)
16:        \( \vec{e}_m := (1 - c)w\vec{e}_i + cw\vec{e}_{i+1} \)
\[ m := \text{lineValue}(p, p_m + e_m, p_m - e_m) \]
\[ q := p_m + e_m - m2e_m \]
\[ d := \|q - p\| - t \]
\[ \text{if } d < \epsilon \text{ then break } \]
\[ p := p + 0.85 \cdot d \vec{r}_d \]
\[ \text{end for} \]
\[ \text{if } d \geq \epsilon \text{ then return } \]
\[ \vec{t}_+ := p^+_{t} - (p_{t-1} + w\vec{e}_{t-1}) \]
\[ \vec{t}_- := p^-_{t} - (p_{t-1} - w\vec{e}_{t-1}) \]
\[ \vec{t}_c := (p_{t+2} + w\vec{e}_{t+2}) - p^+_s \]
\[ \vec{t}_e := (p_{t+2} - w\vec{e}_{t+2}) - p^-_s \]
\[ \vec{t}_a := \text{normalize}((1 - a)\vec{t}_e^+ + a\vec{t}_e^-) \]
\[ \vec{t}_b := \text{normalize}((1 - b)\vec{t}_c^+ + b\vec{t}_c^-) \]
\[ \vec{t} := \text{normalize}(p - q) \]
\[ \vec{m} := \text{normalize}(p - \langle \vec{t}, \vec{m} \rangle \vec{t}) \]
\[ \vec{n} := \text{normalize}(\vec{m} - \langle \vec{t}, \vec{m} \rangle \vec{t}) \]
\[ \text{end function} \]

For the ray casting of the sticks, we first introduce a function to compute the minimal distance from a point \( p \) to the shaft with start \( p_s \), end \( p_e \), and normal orientation \( \vec{o} \).

1. **function distShaft** \( (p, p_s, p_e, \vec{o}) \)
2. \( m := \text{lineValue}(p, p_s, p_e) \)
3. \( p_m := p_s + m(p_e - p_s) \)
4. \( e_m := 0.8w \cdot \text{normalize}(\vec{o} \times (p_e - p_s)) \)
5. \( s := \text{lineValue}(p, p_m + e_m, p_m - e_m) \)
6. \( q := p_m + e_m - s2e_m \)
7. **return** \( \|p - q\| - 0.8t \)
8. **end function**

Next, we provide the distance functions for the different stick types.

1. **function distAdenine** \( (p, p_s, p_e, \vec{o}) \)
2. \( \vec{e} := \text{normalize}(p_e - p_s) \)
3. \( e := \text{lineValue}(p, p_e - 0.8t\vec{o}, p_e + 0.8t\vec{o}) \)
4. \( p_e := p_e + 0.8t\vec{o}(2c - 1) \)
5. \( p_t := p_e + 0.8(w + t)\vec{e} \)
6. \( d_e := \|p - p_t\| - 0.8(w + t) \)
7. \( d_s := \text{distShaft}(p, p_s, p_e, \vec{o}) \)
8. \( d_l := \text{distShaft}(p, p_s, p_t, \vec{o}) \)
9. **return** \( \min(d_s, \max(d_l, d_e)) \)
10. **end function**
1: function distCytosine(p, p_s, p_e, \vec{\sigma})
2:  \vec{c}_1 := \text{normalize}(p_e - p_s)
3:  \vec{c}_2 := \text{normalize}(\vec{\sigma} \times \vec{c}_1)
4:  \vec{n}_t_1 := \text{normalize}(-0.5(\vec{c}_1 + \vec{c}_2))
5:  \vec{n}_t_2 := \text{normalize}(-0.5(\vec{c}_1 - \vec{c}_2))
6:  p_t := p_e + 0.8(w + t)\vec{c}_1
7:  d_{t_1} := \langle \vec{n}_t_1, p - p_t \rangle
8:  d_{t_2} := \langle \vec{n}_t_2, p - p_t \rangle
9:  d_s := \text{distShaft}(p, p_s, p_e, \vec{\sigma})
10: \text{return } \max(d_s, \min(d_{t_1}, d_{t_2}))

11: end function

1: function distGuanine(p, p_s, p_e, \vec{\sigma})
2:  \vec{c}_1 := \text{normalize}(p_e - p_s)
3:  \vec{c}_2 := \text{normalize}(\vec{\sigma} \times \vec{c}_1)
4:  \vec{n}_t_1 := \text{normalize}(0.5(\vec{c}_1 + \vec{c}_2))
5:  \vec{n}_t_2 := \text{normalize}(0.5(\vec{c}_1 - \vec{c}_2))
6:  p_t := p_e + 0.8(w + t)\vec{c}_1
7:  d_{t_1} := \langle \vec{n}_t_1, p - p_t \rangle
8:  d_{t_2} := \langle \vec{n}_t_2, p - p_t \rangle
9:  d_l := \text{distShaft}(p, p_s, p_t, \vec{\sigma})
10: \text{return } \max(d_l, \max(d_{t_1}, d_{t_2}))

11: end function

1: function distThymine(p, p_s, p_e, \vec{\sigma})
2:  \vec{c}_1 := \text{lineValue}(p, p_e - 0.8t\vec{\sigma}, p_e + 0.8t\vec{\sigma})
3:  p_c := p_e + 0.8t\vec{\sigma}(2c - 1)
4:  d_s := \text{distShaft}(p, p_s, p_c, \vec{\sigma})
5:  d_c := \|p - p_c\| - 0.8(w + t)
6:  \text{return } \max(d_s, -d_c)
7: end function

1: function distStick(p, p_s, p_e, \vec{\sigma}, \text{type})
2:  \text{if } \text{type} = \text{adenine} \text{ then return } \text{distAdenine}(p, p_s, p_e, \vec{\sigma})
3:  \text{if } \text{type} = \text{cytosine} \text{ then return } \text{distCytosine}(p, p_s, p_e, \vec{\sigma})
4:  \text{if } \text{type} = \text{guanine} \text{ then return } \text{distGuanine}(p, p_s, p_e, \vec{\sigma})
5:  \text{if } \text{type} = \text{thymine} \text{ then return } \text{distThymine}(p, p_s, p_e, \vec{\sigma})
6:  \text{if } \text{type} = \text{uracil} \text{ then return } \text{distThymine}(p, p_s, p_e, \vec{\sigma})
7: end function

Finally the full ray casting of a stick, that provides the intersection position \(p\) and the surface normal \(\vec{n}\) is given by
function castStick(pₛ, pₑ, ⃗o, type)

sₘₐₓ := 50

ϵ := 0.001

p := rₛ

for i = 1...sₘₐₓ do

d := distStick(p, pₛ, pₑ, ⃗o, type)

if d < ϵ then break

p := p + d⃗rₜₐₜ
enď for

if d ≥ ϵ then return

η := 0.02

n₊ₓ := distStick(p + (η, 0, 0)ᵀ, pₛ, pₑ, ⃗o, type)

n₋ₓ := distStick(p − (η, 0, 0)ᵀ, pₛ, pₑ, ⃗o, type)

n₊ᵧ := distStick(p + (0, η, 0)ᵀ, pₛ, pₑ, ⃗o, type)

n₋ᵧ := distStick(p − (0, η, 0)ᵀ, pₛ, pₑ, ⃗o, type)

n₊𝑧 := distStick(p + (0, 0, η)ᵀ, pₛ, pₑ, ⃗o, type)

n₋𝑧 := distStick(p − (0, 0, η)ᵀ, pₛ, pₑ, ⃗o, type)

⃗n := normalize((n₊ₓ − n₋ₓ, n₊ᵧ − n₋ᵧ, n₊𝑧 − n₋𝑧)ᵀ)

end function

Further Results

Fig. 20 shows a tRNA that is very similar to the one shown in the main manuscript. This is typical for tRNA: they only differ by a few nucleotides but their main structure determined by the secondary structures is quite conserved. Fig. 21 shows part of an mRNA that contains several internal loops. In the linear plot, one can also see several base pairs that lead to tertiary structure elements. Fig. 22 shows a molecular structure that contains three different types of structure: RNA, DNA, and protein. In the 2D graph model, the DNA double helix can easily be identified by the long ladder structure. Fig. 23 again shows a complex of DNA structure and protein structures. This structure is called nucleosome. Here, the DNA wraps around the protein structures like thread that is wrapped around a spool. Fig. 24 shows a beautiful Holliday junction of a DNA. Here, again, the DNA is bound to several protein structures. In Figs. 25 and 26, we show the largest structure that we investigated in our paper. It is a ribosomal structure 80S containing 10399 nucleotides. The 2D graph model consist of two large connected components.

User Feedback

We collected some user feedback about our tool, in particular the tight linking of 2D and 3D visualizations, from three experts working in the field of molecular modeling.

User 1

For 20 years I have been in touch with RNA modeling in the context of aptamer and ribozyme investigations. The introduced techniques give a real new perspective for modeling 3D structures in a very efficient way. Starting with a reduced
knowledge of the molecule (atomic type and residue membership), the tools enable a fast access to nearly all databases. The brushing and linking of 2D and 3D data are impressive and the fast ray casting visualization makes real-time modeling even for large structures possible. Therefore, the challenging tasks of real 3D modeling, the combination of 2D analysis, 3D model, and thermodynamical exploration of molecular states, all benefit from the fast visualization techniques presented in this paper.

User 2

I am an associate research fellow at a major pharmaceutical company, with 27 years of experience. I have significant experience in protein representations, in 1D (sequence), 2D (interaction diagram), and 3D (atomic or ribbons rendering), and a strong background in structure visualization, structure-based drug design, and interactive computer graphics (both programming and using). I have less experience with nucleic acids, but the experience should carry over; I am confident that I can offer reasonable feedback.

I was asked to comment on the interactive display of 2D and 3D views specifically. I will say without reservation that this interactivity between 2D and 3D representations is extremely helpful for drug design, and helps tie together various pieces of experimental data that cannot otherwise be correlated easily. For example, mutation data will show that changing one particular residue has a profound effect on the activity of a particular bound molecule. You can look at that on a 2D plot and understand the relationship between that change and interacting residues; but only the 3D view provides enough detail to understand the effects on an atomic level. Likewise, changes to a bound molecule can subtly change interactions, but the effect of those interactions on a more global sense are much more readily observed on the 2D plot. The two views are synergistic. Also, the efforts by the authors to have the 2D and 3D views be significantly similar makes this synergy stronger.

The methods seem to be a significant step forward in the state of the art, especially with respect to rendering speed and clarity of visualization. Also, it seems to be a very significant step to offer both optimized 2D and 3D representations with interactivity between them.

User 3

I have been working in the field of molecular modeling and simulation for almost 10 years. From this experience, I can say that the interaction between 2D and 3D visualization is a beneficial yet largely missing feature in most visualization tools. While 2D drawings give you a very good overview of the molecular structure and allow you to easily grasp and navigate through it, a 3D visualization is necessary to obtain details about the molecular structure and to determine atomic interactions. Only the above mentioned link between 2D and 3D allows a quick visual insight into the relative positions of spatially interacting parts on the primary sequence level. This feature is not only important for protein structures but even more for complex RNA molecules. The proposed methods, in particular the fully interactive linking of 2D and 3D visualizations are a big step forward for interactive molecular structure analysis.
Figure 12: All types of base pairs that will be detected by our approach. The dotted lines indicate the responsible hydrogen bonds.
Figure 13: Examples of base pairs detected by DSSR and not by our approach (left column) and vice versa (right column). For each pair an almost frontal view onto the base rings (top) and a view from above (bottom) is provided. Except for the Wobble pair (bottom left), the pairs can be seen as borderline cases that could be accepted or rejected due to base torsions or the distance between donor and acceptor atom. However, the hydrogen bonds for classical G-U Wobble pairs seem to be quite unrealistic for the bottom left pair. Either this is a limitation of DSSR or it is some kind of specific Wobble pair with other hydrogen bonds than the depicted ones that our approach does not detect.
Figure 14: Different setups for bending and twisting angles (in degree) on a single ribbon segment with a step width scale of 0.85. The length of the segment was chosen to particularly highlight the artifacts. They appear due to the approximation for the detection of the closest capsule, the error of which increases much more with the bending angle than with the twisting angle. However, these artifacts never appeared in practical cases, see Fig. 16.

Figure 15: Influence of the scale of the step size on the sphere tracing result for a single ribbon segment. We observed these artifacts only for twisting but never for bending cases. With a step size of 0.85, these artifacts do not appear at all.
Figure 16: The effect of the tangent interpolation to create smooth normals between neighbored ribbon segments (top: disabled, bottom: enabled). Furthermore, several examples of strong bending and twisting examples in practical examples for the strand detail of $s = 10$. We also tested lower details of $s = 5$ and $s = 2$ without detecting any artifacts.
Figure 17: Comparison of the 3D ladder model visualization by different tools for the tRNA structure PDB ID: 1EHZ. The tools are (a) Autodesk molecule viewer, (b) NGL viewer, (c) PyMol, (d) VMD, (e) Chimera, (f) Jmol.
Figure 18: Our ladder model using triangulation (top) and ray casting (bottom) of the structure PDB ID: 1EHZ.
Figure 19: 2D graph model visualization for a structure of a self-spliced group II intron (PDB ID: 3BWP). The top image shows the starting position used by Hecker et al. [15]. The middle image shows our modified starting position. The bottom image shows the result after gradient-descent optimization of the start position shown in the middle. Color denotes the different strands.
Figure 20: Yeast phenylalanine tRNA (PDB ID: 1EHZ) showing the typical structure of a tRNA. The nucleotides are colored according to the base pair type: Watson-Crick (red), reverse Watson-Crick (orange), Wobble (yellow), and reverse Hoogsteen (green). Non-paired bases are depicted in gray.
Figure 21: Structure of the S-adenosylmethionine riboswitch mRNA regulatory element (PDB ID: 2GIS). All nucleotides are colored according to their type. Backbone and sticks are raycasted.
Figure 22: Streptococcus pyogenes Cas9 complexed with guide RNA and target DNA (PDB ID: 4OO8). Colors denote different strands. Top: 2D linear model. Left: 2D graph model showing RNA and DNA. The latter can be easily identified by the long ladder built by the two strands. Right: RNA and DNA are rendered using ray casting. The DNA part can be seen in the foreground while the RNA part is mostly hidden. The protein part is visualized using the solvent excluded surface.
Figure 23: Molecular structure of a nucleosome containing DNA with tetrahydrofuran (PDB ID: 5JRG). The DNA, again rendered using our ray casting approach, wraps around the protein structures. Top: The proteins are depicted by their secondary structures. Bottom: The protein is shown by the molecular surface.
Figure 24: Structure of a Holliday junction complex (PDB ID: 5J0N). Top: Only the Holliday junction built by the DNA is depicted using the triangulated ribbon-stick visualization. Bottom: The DNA is shown using the same visualization as above but this time in complex with several protein structures bound to the DNA.
Figure 25: Structure of Geneticin bound to the yeast 80S ribosome (PDB ID: 4U4O). The structure consists of two identical parts (left and right, see bottom image), which altogether contain 10399 nucleotides. For clarity, the 2D visualization (top) displays only one component of strands of the left part of the RNA structure. The other component of the left RNA structure is shown in Fig. 26. The 3D visualization shows the RNA as ray casted ribbon-stick rendering while the protein structures are depicted by their molecular surface. Colors denote strands affiliation.
Figure 26: Structure of Geneticin bound to the yeast 80S ribosome (PDB ID: 4U4O). The 2D graph model visualization shows the second part of the RNA structures shown in Fig. 25. Again, colors denote strand affiliation.
Figure 27: Structure of ribosome-bound cricket paralysis virus IRES RNA (PDB ID: 2NOQ). The structure consists of 5 strands. Left: Original approach by Hecker et al. [15]. Right: Our approach.
Figure 28: Structural basis for interaction of the ribosome with the switch regions of GTP-bound elongation factors (PDB ID: 2OM7). Only the rRNA part is depicted by the 2D graph, which consist of 14 strands. Left: Original approach by Hecker et al. [15]. Right: Our approach.
Figure 29: A proorganized active site in the crystal structure of the Tetrahymena ribozyme (PDB ID:1GRZ). The structure consists of two strands. Left: Original approach by Hecker et al. [15]. Right: Our approach.
Figure 30: Large ribosomal subunit complex with R(CC)-DA-Ruromycin (PDB ID: 1FFZ). The structure consists of 3 strands. Left: Original approach by Hecker et al. [15]. Right: Our approach.
Figure 31: Structural investigation of the GlmS ribozyme bound to its catalytic cofactor (PDB ID: 2NZ4). The structure consists of 4 equal independent parts of 12 strands. This leads to the 4-fold symmetry in the 2D graph layout. Left: Original approach by Hecker et al. [15]. Right: Our approach.
Figure 32: Cocrystal structure of a tyrosyl-tRNA synthetase splicing factor with a group I intron RNA (PDB ID: 2RKJ). Similar to 2NZ4, the structure consists of 4 equal independent parts of 16 strands. Again, this leads to a 4-fold symmetry in the 2D graph layout. Left: Original approach by Hecker et al. [15]. Right: Our approach.
Figure 33: Structure of the Tetrahymena ribozyme: Base triple sandwich and metal ion at the active site (PDB ID: 1X8W). Similar to 2NZ4 and 1x8W, the structure consists of 4 equal independent parts of 12 strands. Again, this leads to a 4-fold symmetry in the 2D graph layout. Left: Original approach by Hecker et al. [15]. Right: Our approach.
Figure 34: Structure of the Thermus thermophilus 30S ribosomal subunit in the presence of crystallographically disordered codon and near-cognate transfer RNA anticodon stem-loop mismatched at the second codon position (PDB ID: 1N36). The RNA structure consists of 6 strands. Left: Original approach by Hecker et al. [15]. Right: Our approach.
Figure 35: Crystal structure of the 30S subunit of a 70S ribosome with thrS operator and tRNAs (PDB ID: 1YL4). The RNA structure consists of 7 strands. Left: Original approach by Hecker et al. [15]. Right: Our approach.