Interactive Visualization of RNA and DNA Structures

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

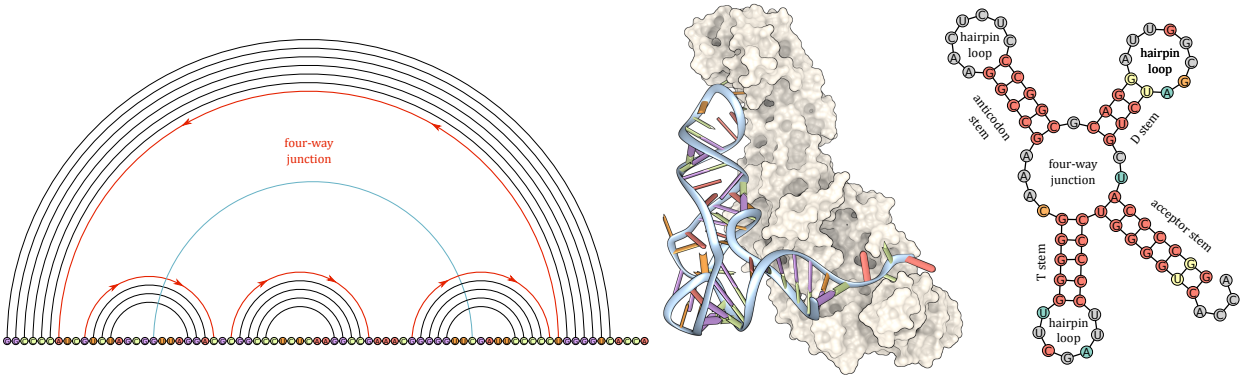

Fig. 1. Glutamyl-tRNA (PDB ID: 1G59). *Left:* 2D linear model showing the nucleotide sequence at the bottom line and the base pairings by circular arcs. Nucleotides are colored by type. *Middle:* 3D visualization of the glutamyl-tRNA as ribbon-stick model complexed with a glutamyl-tRNA synthetase shown as solvent excluded surface. Again, nucleotides are colored by type. *Right:* 2D graph model, with color highlighting the different base pair types. The four-way junction can also be seen in the left image with highlighted red lines.

Abstract— The analysis and visualization of nucleic acids (RNA and DNA) is playing an increasingly important role due to their fundamental importance for all forms of life and the growing number of known 3D 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. We present a ray casting-based visualization of RNA and DNA secondary and tertiary structures, which enables for the first time 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.

Index Terms—Ribonucleic acids, DNA, RNA, secondary & tertiary structures, interactive rendering, ray casting, brushing & linking.

1 Introduction

Nucleic acids are macromolecules that play a major role in many cellular processes [42]. 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 [43]. 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 [40], see Fig. 1. The tertiary structure of RNA is formed by additional base pairs, which typically are farther 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 pseudoknots and kissing hairpins [40].

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 [30]. To elucidate their structures, sophisticated tools are required that allow the visual exploration of the primary, secondary and tertiary structures. For almost 40 years [48], 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 understood well by 3D visualizations that show the primary structure as backbone, and the secondary and tertiary structures through the bases and base pairs embedded in 3D space. To avoid visual clutter, 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 [22, 23, 27, 28, 34, 47] to detect the wide variety of base pairing motifs [29]. 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 the lack of support for specialized RNA and DNA visualization in standard molecular visualization packages, as recently stated by Hanson and Lu [10]. To overcome this shortcoming, they extended *Jmol* [10] to support the visualization of base pairs and secondary structures by coupling it to the nucleic acid analysis tool DSSR [27]. Interactive exploration, however, is still lacking since no

• *Norbert Lindow, Daniel Baum, Morgan Leborgne, and Hans-Christian Hege are with Zuse Institute Berlin. E-mail: {norbert.lindow, baum, leborgne, hege}@zib.de*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

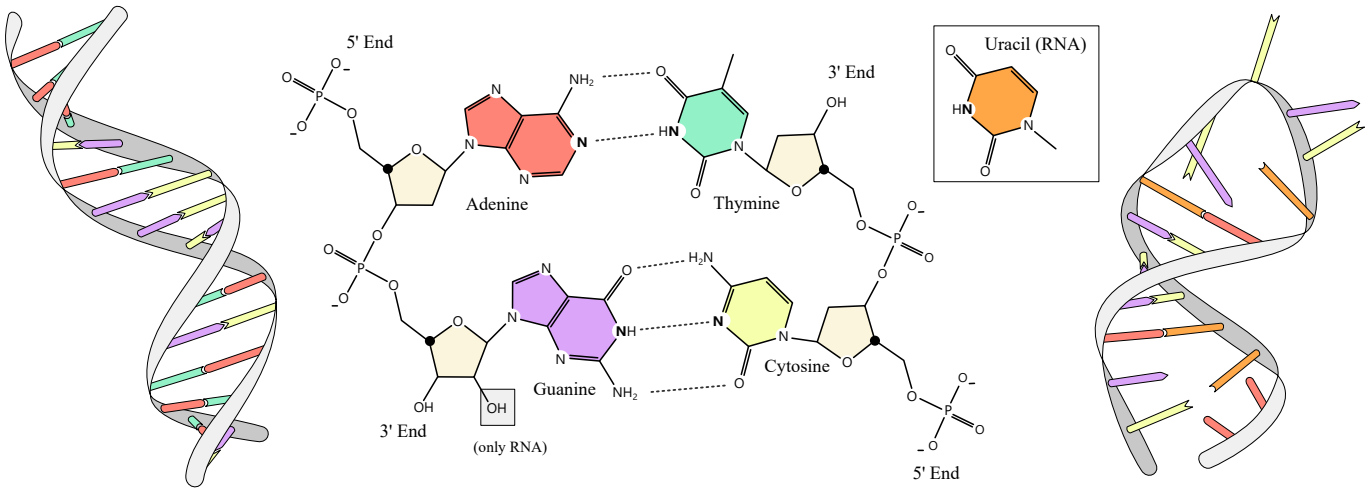

Fig. 2. The double-stranded helix structure of DNA (left) and the single-stranded structure of RNA (right). Both nucleic acids are composed of nucleotides that consist of a backbone part and a nucleobase. The backbone contains sugar (beige ring) and a phosphate group. In contrast to RNA, DNA has no OH-group 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.

2D visualizations are provided in *Jmol*, which limits the selection of secondary structures to command line interaction. This problem is overcome by *Assemble2* [18] in combination with *UCSF Chimera* [33]. 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 work is to show that accelerated algorithms for computation and visualization allow interactive visual exploration even for very large nucleic acids. To this end we make the following sub-contributions:

1. 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. (Sect. 4)

2. 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. (Sect. 5)

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

4. 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. (Sects. 7 & 8)

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 [43], 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 [19], 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 [46]. 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 [22,23], the purine bases are rotated by 180° w.r.t. the classical Watson-Crick base pairs. Thus, Hoogsteen base pairs build different hydrogen bonds. Another variation is the Wobble pair [22, 23], 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 [22] take the first approach. Different methods exist for adding hydrogen atoms [5, 44]. They use the force field-based approach by Cornell et al. [5]. 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. [47] and Lu, Olson, and co-workers [27, 28]. For the computation of base pairs, they use the standard reference frame for nucleic acid base pairs defined by Olson et al. [32], 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. [35]. 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 [36], starting their previously presented algorithm [35] from the circle visualization by Zuker et al. [48]. Though this does not guarantee overlap-free visualizations in all cases, in general, it works very well. A few years later, Bruccoleri and Heinrich [3] 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 [14, 15, 26].

Since automatic methods rarely produced fully satisfying results, in particular for very complex molecular structures, De Rijk and De Wachter [8] 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 *RnaViz 2*, an improved version [9], that included support of more file formats as well as better annotations. The *VARNA* tool by Darty et al. [7] provides four classical representations for 2D secondary structures and allows editing and annotation. Furthermore, it can be run on a web-server.

In 2013, Hecker et al. [12] 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. [36], it starts from the circle plot [48]. To arrive at visualizations that are similar to the standard drawing [35], they run 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 [10], support of specialized 3D visualizations for RNA and DNA in standard molecular visualization packages is largely missing. Even tools like Genome3D [24] and GMOL [31], 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* [28]. It visualizes the bases as blocks that describe the standard reference frame [32] 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* [47]. *UCSF Chimera* also offers support for the visualization of bases. These visualizations include filled rings, boxes, ellipses and elliptical sticks [6]. 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. [17] is to enable users to model nucleic acids. For this purpose, they combine several tools including *RNAplot* [14] for the generation of 2D secondary structure visualizations and *RNAview* [47] for the annotation of 3D structures. *Assemble2* [16, 18] represents an improved version of the previous modeling tool that uses *Chimera* [6, 33] 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* [27], a very powerful tool to analyze RNA structures that uses *Jmol* for the 3D visualization. Recently, Hanson and Lu described this integration [10], 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 [13, 21, 41]; also see the survey paper on molecular visualization by Krone et al. [20]. Already in 2008, Krone et al. [21] investigated several GPU implementations to compute the ribbon geometry on the fly, hereby exploiting the geometry shader. Three years later, Whale and Birmanns [41] presented an approach utilizing vertex shaders. Compared to Krone et al., with their hybrid GPU/CPU-based approach they achieved a three times faster rendering. Hermosilla et al. [13] exploited the tesselation 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 [27], for example, N1, C2, N3, C4, C5, C6 etc. We found that this information is not always reliable. Therefore, we reduce the requirements even further and assume only that for each atom the Cartesian coordinates and atomic number are given, and that all atoms belonging to a nucleotide are grouped in 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

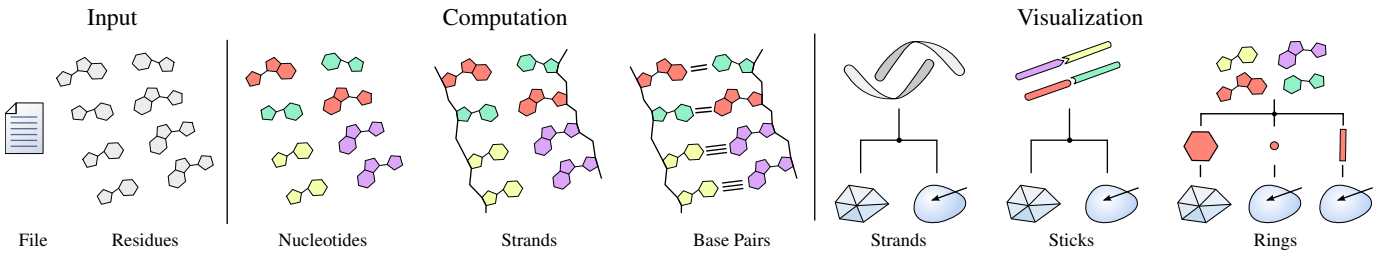

Fig. 3. Processing pipeline for DNA/RNA visualization. First, all residues are read from the input file. The atom positions and element types must be given. Then, the subset of nucleotides and their types are detected. Afterward, the strands and the base pairs are computed. Finally, the different geometric components are constructed and either triangulated or directly ray casted.

computation of covalent bonds can be done based on the atom positions, the atomic numbers, and the covalent atom radii [39].

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, $\psi : V \longrightarrow \mathbb{N}$ 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 $\forall (a_i, a_j) \in E_X \Longrightarrow (m(a_i), m(a_j)) \in E$. 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. [27]).

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 needs 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. [4, 45, 47]). 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

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

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.

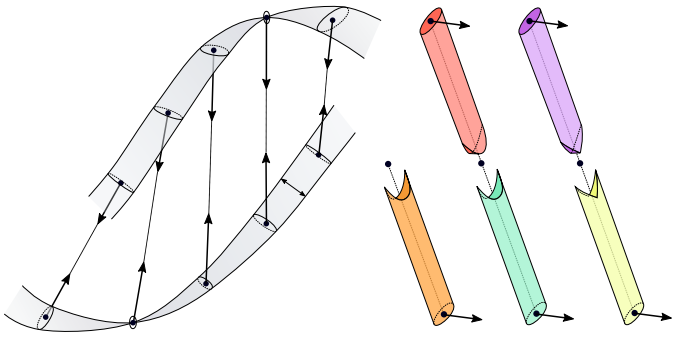

Fig. 4. Illustration of the ladder visualization model. *Left:* backbone strand, given by one point and one orientation per nucleotide. *Right:* base sticks with round end patterns for A, T, and C, and peaked ends for C and G.

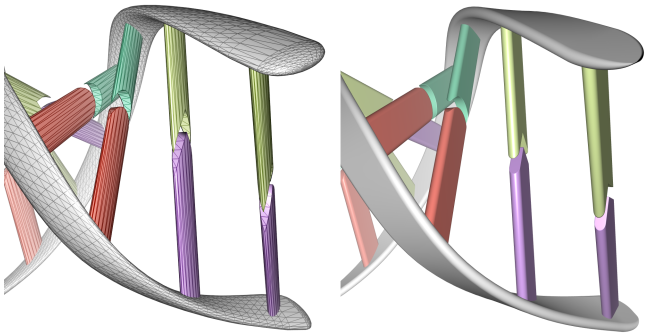

Fig. 5. Close-up of triangulation (left) and the GPU-based ray casting (right) of the ladder model of the end of a DNA strand.

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 (Figs. 3,5). 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. It is partially based on the approach by Sigg et al [37]. 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 $\vec{o}_i \in \mathbb{R}^3$. 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 $\vec{o}_i \times \vec{t}_i$. 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 ith 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).

For ray casting the ribbons, we use a capsule-shaped cross-section instead of an elliptical one, because the distance to a capsule can be computed faster 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 [11]. 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 e_{i-1}, 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. [25]. Finally, the sphere tracing is performed in the fragment

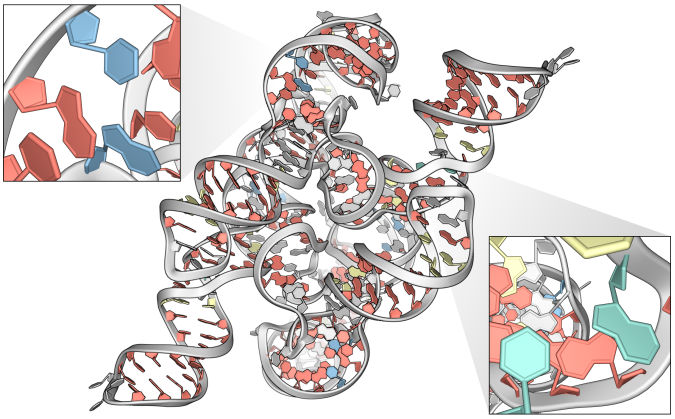

Fig. 6. A group I ribozyme domain (PDB ID: 1GID). Bases are visualized using the ring model. Colors denote base pair types: Watson-Crick (red), Wobble (yellow), Hoogsteen (blue), reverse Hoogsteen (green).

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 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 ε or we reach a certain number of iterations, the algorithm stops. Otherwise, p is moved along the ray by $0.85 \cdot d$. We don't move the full distance because we use a heuristic to detect the closest capsule, see also Suppl. Fig. 5. In practice, we use $\varepsilon = 0.001$ Å and a maximum of 50 steps. If d is still larger than ε 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, here we use the tangents at the four corners of the segment: Let $\vec{t}_{i_{1,2}}$ be the normalized vectors of $p_{i+1} - p_{i-1} \mp w(e_{i+1} - e_{i-1})$. 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 $(1-c)\vec{t}_a + c\vec{t}_b$. Then, 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 segments.

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 $0.8w$ and $0.8t$ 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 jth vertex by $\pm 0.8w\left|\sin(\alpha_j)\right|$ 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 $1.2w$. 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

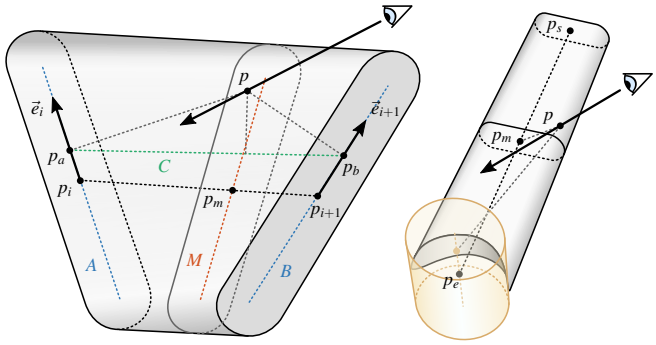

Fig. 7. A single ray casting step for a ribbon segment (left) and a stick (right). *Left:* First, the closest points p_a on A and p_b on B to the current position p on the ray are computed. Then, the closest point to the connection C of p_a and p_b is used to define the closest capsule with axis M. *Right:* 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.

the base types. Let p_s and p_e be the start and end points of the stick and let $\vec{o}$ 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 $\vec{o} \times (p_e - p_s)$. Again we use $0.8w$ and $0.8t$ as width and thickness values. For the round patterns, we also compute the distance 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, with the advantage of giving an impression of slightly rounded edges and avoiding pixel artifacts at sharp edges.

5.3 Sugar & Base Rings

We use classic GPU-based ray casting to render spheres and cylinders of ring structures and bonds between sugar and base rings. Except for some minor improvements, the approach is equal to the one presented by Sigg et al. [37]. 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 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 visualization models used 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 whose vertices are the nucleotides and whose edges represent connections between adjacent nucleotides along the strand as well as Watson-Crick and Wobble base pairs. We call the first type of edges 'backbone

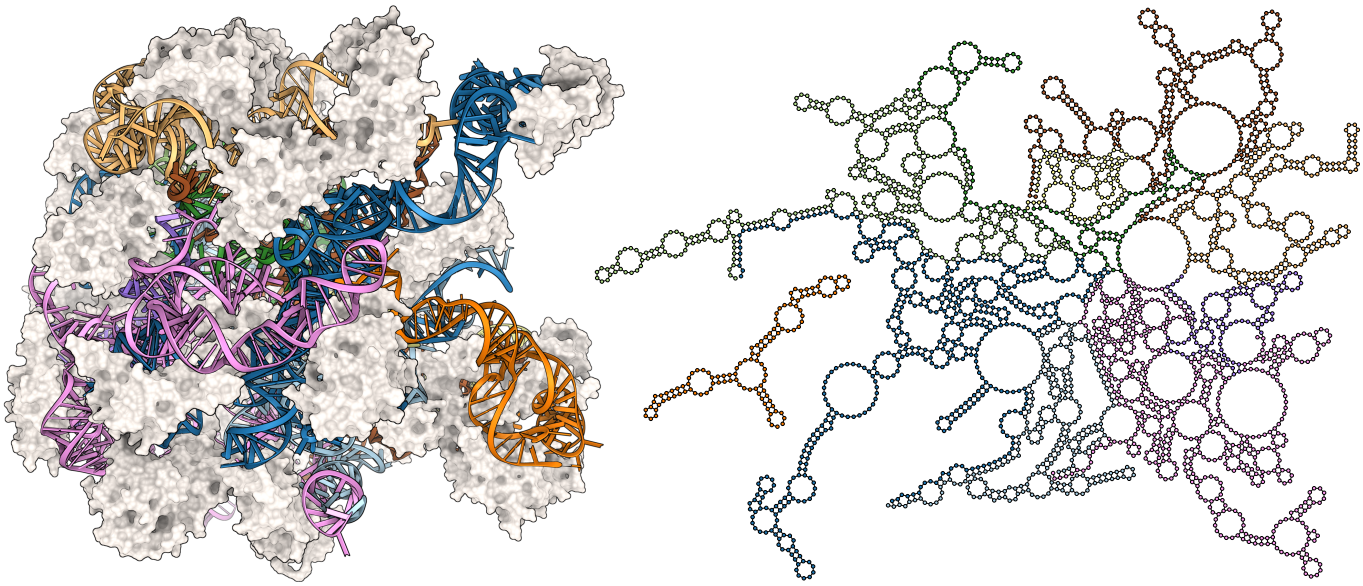

Fig. 8. Large ribosomal subunit (PDB ID: 1S72). *Left:* The ribosomal RNA with 2876 nucleotides is shown by the ray casted ladder model, colored by strands, while the ribosomal protein is depicted by the molecular surface. *Right:* 2D graph model of the ribosomal RNA.

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. [38] 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 pseudoknot edges. The idea is to compute a graph representation that best preserves the local neighborhood of the 3D structure in the 2D graph. To do so, the nucleotides are arranged in such a way that the edges have constant lengths, the base edges are 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. To eliminate intersections, we must deform the graph, accepting minor distortions. Various researchers 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. [12]. 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 [48], 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 cost function that quantifies the difference between the current layout and the ideal solution. It 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, which could easily take half a day for large molecules, we propose two major improvements.

An expensive part is the evaluation of the repulsion forces, since for every base all other bases and edges need to be processed. For repulsion, however, only nearby adjacent bases and edges are important. For this reason, we restrict the repulsion to a certain spatial neighborhood that allows us to collect the relevant bases and edges in constant time using a 2D grid data structure.

Another issue is the initialization of the circle visualization, which is not optimal because it can produce 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 bulge the circle locally in the following way: First, all base edges of the graph are sorted by their length in ascending order. Then, the edges are processed iteratively. Each edge divides the circle into two parts. The part 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 adjusting the view over time. In particular, the user may 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 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

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).

PDB ID	#Res	Computation			Geometry		UR
		#Nuc.	Nuc.	Sec.	Mesh.	Ray.	
D 5L6L	996	54 (0)	3.5	0.1	1.4	0.1	> **200**
R 1EHZ	245	76 (14)	2.0	0.2	1.7	0.2	> **200**
R 2GIS	189	95 (1)	1.8	0.3	2.3	0.2	> **200**
R 3DS7	771	136 (0)	3.9	0.4	3.1	0.3	> **200**
R 4RGE	277	167 (3)	2.5	0.5	4.0	0.4	> **200**
H 4OO8	2846	236 (0)	13.2	0.5	5.8	0.6	**159 / 952**
D 5JRG	1322	289 (2)	8.5	0.8	7.2	0.6	**125 / 719**
R 1GID	350	316 (0)	3.4	0.9	7.8	0.7	**116 / 628**
R 3BWP	373	356 (7)	3.7	1.1	9.0	0.9	**99 / 513**
R 1S72	14705	2876 (5)	93.4	9.4	72.5	6.2	**12 / 64**
R 5AFI	11467	4801 (53)	76.7	18.0	124.5	11.6	**7 / 34**
R 4U4O	35860	10398 (1)	168.8	43.8	279.8	24.5	**3 / 15**

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

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 these 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.

8 Results

To measure the performance and to evaluate the algorithms, we used datasets from the PDB [2] that differ in size and requirements.

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 [27]. 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 investigating 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 that tests a default or previous successful mappings (see Sect. 4.1). The performance is mainly influenced by the number of residues and their complexities. 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 widely (see the dark green bars in Fig. 9). This is not a problem since for molecular dynamics trajectories, this computation must 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 mesh of strands and sticks consists of $\sim$3 Mio. vertices and $\sim$5 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 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$30 Mio. triangles. Yet, we still achieved almost 13 fps. Thus, for this rendering approach, the vertex processing is probably the bottleneck. In contrast, for the ray casting of the ladder model, the amount of work per fragment is much higher. Thus, the rendering performance depends highly on the overall number of fragments that need to be processed, which in turn depends on the data, the screen resolution, and the camera setup. For our screen resolution, the worst case frame rate for the largest dataset was 15 fps. For the medium-sized datasets ($\sim$3 000 nucleotides, Fig. 8) we achieved at least 30 fps and for the smaller ones ($\sim$350 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 rRNA molecule with $\sim$3 000 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

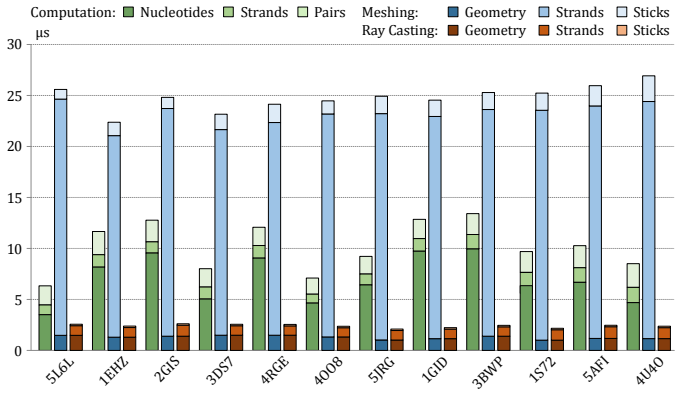

Fig. 9. Computation times for the nucleotides, backbone strands, and base pairs as well as for the geometry construction of the meshing and the ray casting. The geometry construction is subdivided into the modeling part and the generation of the strands and sticks. Except for the nucleotide computation, which is given per residue, all other timings are given per nucleotide.

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 work, 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 structures, determination of the nucleotides, computation of strands and base pairs to the visualization with tightly linked 2D and 3D renderings. We focused on visual quality and performance with the aim of enabling 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 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 in terms of both 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.

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 used by most base-pair detection algorithms. Nevertheless, our evaluation (Sect. 8.1) shows that with the proposed approaches in terms of quality we get very similar results to the ones obtained by tools like DSSR. 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 [27], 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. [12] 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 running times for very large ones (like 4U4O). By executing the computation in parallel, interaction with the 2D visualization is possible

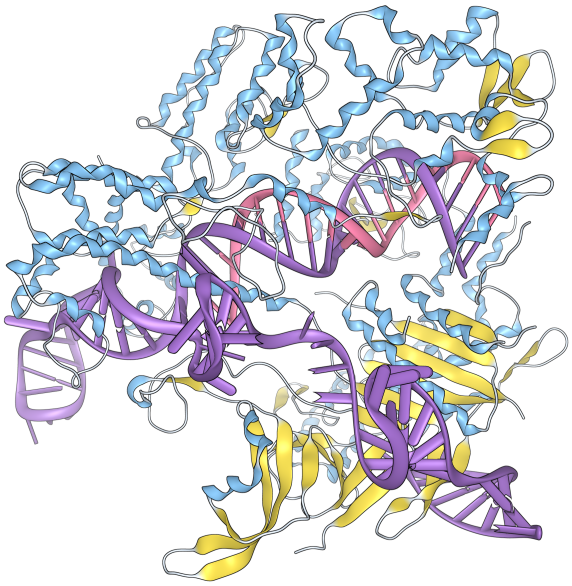

Fig. 10. Streptococcus pyogenes Cas9 complexed with guide RNA and target DNA as triangulated ladder model (PDB ID: 4OO8).

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.

Among the various results of this paper, we see as a major contribution the first ray casting approach for rendering the backbone and base pairs of nucleic acids. The backbone rendering can also 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, as only a small amount of information needs to be computed on the CPU and sent to the graphics card, which allows high update rates. On the other hand, in our meshing approach, we focused on visual quality rather than performance. Using geometry and/or tessellation shaders to generate the mesh on the GPU [13], 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.

In addition to fast rendering by means of ray casting, we have implemented for the first time a 3D ladder model to depict base pairs with complementing ends, allowing an easy identification of the nucleotides. Instead of the often used block visualization of the base pairs, we propose a ring model that offers comparable information but better represents the atomic structure. Our last contribution is the tight linking between 2D and 3D visualizations – as a proof of concept, especially for highlighting and filtering substructures.

For future work, we will develop further interaction techniques in collaboration with domain experts. We also plan to look in detail at the base pairings. In particular, it could be interesting to explore the suitability of machine learning 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.

Acknowledgements

The authors wish to thank Vedat Durmaz from Zuse Institute Berlin for providing DNA and RNA molecular dynamics trajectories. In addition, the authors would like to thank the 3 anonymous domain experts for their user feedback.

REFERENCES

[1] F. H. Allen, S. Bellard, M. Brice, B. A. Cartwright, A. Doubleday, H. Higgs, T. Hummelink, et al. The Cambridge Crystallographic Data Centre: computer-based search, retrieval, analysis and display of information. *Acta Cryst.*, 35(10):2331–2339, 1979.

[2] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucl. Acids Res.*, 28(1):235–242, 2000.

[3] R. E. Bruccoleri and G. Heinrich. An improved algorithm for nucleic acid secondary structure display. *Bioinformatics*, 4(1):167–173, 1988.

[4] M. Coll, C. A. Frederick, A. Wang, and A. Rich. A bifurcated hydrogen-bonded conformation in the d (AT) base pairs of the DNA dodecamer d (CGCAAATTTGCG) and its complex with distamycin. *Proc. Natl. Acad. Sci.*, 84(23):8385–8389, 1987.

[5] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, et al. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *J. Am. Chem. Soc.*, 117(19):5179–5197, 1995.

[6] G. S. Couch, D. K. Hendrix, and T. E. Ferrin. Nucleic acid visualization with UCSF Chimera. *Nucl. Acids Res.*, 34(4):e29–e29, 2006.

[7] K. Darty, A. Denise, and Y. Ponty. VARNA: Interactive drawing and editing of the RNA secondary structure. *Bioinformatics*, 25(15):1974, 2009.

[8] P. De Rijk and R. De Wachter. RnaViz, a program for the visualisation of RNA secondary structure. *Nucl. Acids Res.*, 25(22):4679–4684, 1997.

[9] P. De Rijk, J. Wuyts, and R. De Wachter. RnaViz 2: an improved representation of RNA secondary structure. *Bioinformatics*, 19(2):299–300, 2003.

[10] R. M. Hanson and X.-J. Lu. DSSR-enhanced visualization of nucleic acid structures in Jmol. *Nucl. Acids Res.*, 45(W1):W528–W533, 2017.

[11] J. C. Hart. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *Vis. Comput.*, 12(10):527–545, 1996.

[12] N. Hecker, T. Wiegels, and A. E. Torda. RNA secondary structure diagrams for very large molecules: RNAfdl. *Bioinformatics*, 29(22):2941–2942, 2013.

[13] P. Hermosilla, V. Guallar, Á. Vinacua Pla, and P. P. Vázquez Alcocer. Instant visualization of secondary structures of molecular models. In *VCBM '15: Eurographics Workshop on Visual Computing for Biology and Medicine*, pp. 51–60. Eurographics, 2015.

[14] I. L. Hofacker. Vienna RNA secondary structure server. *Nucl. Acids Res.*, 31(13):3429–3431, 2003.

[15] I. L. Hofacker. RNA secondary structure analysis using the vienna rna package. *Curr. Protoc. Bioinformatics*, 4(1):12–2, 2009.

[16] F. Jossinet. Assemble2: an interactive graphical environment dedicated to the study and construction of RNA architectures. In *Virtual and Augmented Reality for Molecular Science (VARMS@ IEEEVR), 2015 IEEE 1st International Workshop on*, pp. 37–38. IEEE, 2015.

[17] F. Jossinet, T. E. Ludwig, and E. Westhof. Assemble: an interactive graphical tool to analyze and build RNA architectures at the 2d and 3d levels. *Bioinformatics*, 26(16):2057–2059, 2010.

[18] F. Jossinet and E. Westhof. S2S-Assemble2: a semi-automatic bioinformatics framework to study and model RNA 3d architectures. In *Handbook of RNA Biochemistry: Second, Completely Revised and Enlarged Edition*, pp. 667–686. Wiley Online Library, 2014.

[19] E. T. Kool. Hydrogen bonding, base stacking, and steric effects in DNA replication. *Ann. Rev. Bioph. Biom. Struct.*, 30(1):1–22, 2001.

[20] B. Kozlíková, M. Krone, M. Falk, N. Lindow, M. Baaden, D. Baum, I. Viola, J. Parulek, and H.-C. Hege. Visualization of biomolecular structures: State of the art revisited. *Comput. Graph. Forum*, 36(8):178–204, 2017.

[21] M. Krone, K. Bidmon, and T. Ertl. GPU-based visualisation of protein secondary structure. In I. S. Lim and W. Tang, eds., *Theory and Practice of Computer Graphics*. The Eurographics Association, 2008. doi: 10.2312/LocalChapterEvents/TPCG/TPCG08/115-122

[22] S. Lemieux and F. Major. RNA canonical and non-canonical base pairing types: a recognition method and complete repertoire. *Nucl. Acids Res.*, 30(19):4250–4263, 2002.

[23] N. B. Leontis and E. Westhof. Geometric nomenclature and classification of RNA base pairs. *RNA*, 7(4):499–512, 2001.

[24] T. E. Lewis, I. Sillitoe, A. Andreeva, T. L. Blundell, D. W. Buchan, C. Chothia, D. Cozzetto, J. M. Dana, I. Filippis, J. Gough, et al. Genome3D: exploiting structure to help users understand their sequences. *Nucl. Acids Res.*, 43(D1):D382–D386, 2014.

[25] N. Lindow, D. Baum, S. Prohaska, and H.-C. Hege. Accelerated visualization of dynamic molecular surfaces. *Comput. Graph. Forum*, 29(3):943–952, 2010.

[26] R. Lorenz, S. H. Bernhart, C. H. Zu Siederdissen, H. Tafer, C. Flamm, P. F. Stadler, and I. L. Hofacker. ViennaRNA package 2.0. *Algorithms Mol. Biol.*, 6(1):26, 2011.

[27] X.-J. Lu, H. J. Bussemaker, and W. K. Olson. DSSR: an integrated software tool for dissecting the spatial structure of RNA. *Nucl. Acids Res.*, 43(21):e142–e142, 2015.

[28] X.-J. Lu and W. K. Olson. 3DNA: a versatile, integrated software system for the analysis, rebuilding and visualization of three-dimensional nucleic-acid structures. *Nat. Protoc.*, 3(7):1213–1227, 2008.

[29] P. B. Moore. Structural motifs in RNA. *Annu. Rev. Biochem.*, 68(1):287–300, 1999.

[30] H. F. Noller. RNA structure: reading the ribosome. *Science*, 309(5740):1508–1514, 2005.

[31] J. Nowotny, A. Wells, O. Oluwadare, L. Xu, R. Cao, T. Trieu, C. He, and J. Cheng. GMOL: an interactive tool for 3D genome structure visualization. *Sci. Rep.*, 6:20802, 2016.

[32] W. K. Olson, M. Bansal, S. K. Burley, R. E. Dickerson, M. Gerstein, S. C. Harvey, U. Heinemann, X.-J. Lu, S. Neidle, Z. Shakked, et al. A standard reference frame for the description of nucleic acid base-pair geometry. *J. Mol. Biol.*, 313(1):229–237, 2001.

[33] E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin. UCSF Chimera – a visualization system for exploratory research and analysis. *J. Comput. Chem.*, 25(13):1605–1612, 2004.

[34] M. Sarver, C. L. Zirbel, J. Stombaugh, A. Mokdad, and N. B. Leontis. FR3D: finding local and composite recurrent structural motifs in RNA 3d structures. *J. Math. Biol.*, 56(1):215–252, 2008.

[35] B. A. Shapiro, L. E. Lipkin, and J. Maizel. An interactive technique for the display of nucleic acid secondary structure. *Nucl. Acids Res.*, 10(21):7041–7052, 1982.

[36] B. A. Shapiro, J. Maizel, L. E. Lipkin, K. Currey, and C. Whitney. Generating non-overlapping displays of nucleic acid secondary structure. *Nucl. Acids Res.*, 12(1):75–88, 1984.

[37] C. Sigg, T. Weyrich, M. Botsch, and M. H. Gross. GPU-based ray-casting of quadratic surfaces. In *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, pp. 59–65, 2006.

[38] S. Smit, K. Rother, J. Heringa, and R. Knight. From knotted to nested RNA structures: a variety of computational methods for pseudoknot removal. *RNA*, 14(3):410–416, 2008.

[39] J. G. Speight et al. *Lange's Handbook of Chemistry*, vol. 1. McGraw-Hill New York, 2005.

[40] I. Tinoco and C. Bustamante. How RNA folds. *J. Mol. Biol.*, 293(2):271–281, 1999.

[41] M. Wahle and S. Birmanns. GPU-accelerated visualization of protein dynamics in ribbon mode. In *Visualization and Data Analysis 2011*, vol. 7868, p. 786805. International Society for Optics and Photonics, 2011.

[42] Y. Wan, M. Kertesz, R. C. Spitale, E. Segal, and H. Y. Chang. Understanding the transcriptome through RNA structure. *Nat. Rev. Genet.*, 12(9):641–655, 2011.

[43] J. D. Watson and F. H. Crick. Molecular structure of nucleic acids. *Nature*, 171(4356):737–738, 1953.

[44] J. M. Word, S. C. Lovell, T. H. LaBean, H. C. Taylor, M. E. Zalis, B. K. Presley, J. S. Richardson, and D. C. Richardson. Visualizing and quantifying molecular goodness-of-fit: small-probe contact dots with explicit hydrogen atoms. *J. Mol. Biol.*, 285(4):1711–1733, 1999.

[45] Z. Wu, A. Ono, M. Kainosho, and A. Bax. H···N hydrogen bond lengths in double stranded DNA from internucleotide dipolar couplings. *J. Biomol. NMR*, 19(4):361–365, 2001.

[46] P. Yakovchuk, E. Protozanova, and M. D. Frank-Kamenetskii. Base-stacking and base-pairing contributions into thermal stability of the DNA double helix. *Nucl. Acids Res.*, 34(2):564–574, 2006.

[47] H. Yang, F. Jossinet, N. Leontis, L. Chen, J. Westbrook, H. Berman, and E. Westhof. Tools for the automatic identification and classification of RNA base pairs. *Nucl. Acids Res.*, 31(13):3450–3460, 2003.

[48] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucl. Acids Res.*, 9(1):133–148, 1981.