




GPU-accelerated cartoon representation for interactive flexible docking

K.R.Holdsworth¹ , G.Iakovou^{1,2}, S.Hayward¹  and S.D.Laycock¹ 

¹School of Computing Sciences, University East Anglia, Norwich, NR4 7TJ, UK

²Digital Engineering, Aviva Plc, Norwich, Norfolk NR1 3NS, U.K.

Abstract

Molecular docking involves simulating the binding of two proteins and is widely used for structure-based drug design. There are two main types of docking tools: automated and interactive. While automated tools are useful for reducing the search space of ligands and identifying potential binding sites, interactive tools allow the user to guide the docking process and observe what happens during docking. High computation speeds are required for interactive docking to handle both protein deformation and user interaction in real time. The cartoon representation, not to be confused with cartoon style rendering, is a protein representation that shows an abstracted view and is frequently used by structural biologists to not only identify a protein, but also to identify key regions within a protein of interest. There are examples of GPU-accelerated methods to construct the cartoon representation in real time. However, none of these methods achieve real-time assignment of secondary structure and construction of the cartoon representation for a flexible molecule. This paper presents our method to achieve this, integrated into an interactive docking tool with receptor flexibility. The methods outlined in this paper produced some promising results, with proteins of up to 3,300 amino acid residues being constructed and rendered at 70 fps.

CCS Concepts

• **Applied computing** → **Computational biology**; • **Computing methodologies** → **Modeling and simulation**; **Computer graphics**;

1. Introduction

Molecular docking tools are designed to simulate protein binding in silico. Protein binding is the process by which two proteins join together, this can cause or inhibit biochemical reactions, making it ideal for structure-based drug design.

There are two types of molecular docking tools, automated and interactive. Automated tools such as AutoDock and RosettaLigand [MHL*09, LM12] use computational techniques to predict the biological binding pose, amongst a vast set of possibilities. This is useful as a first stage, but there are few automated tools that take user expertise into account or are able to model conformational change on both a local and global scale during docking. Interactive tools enable the user to refine the results from an automated tool and to inspect different docking scenarios in detail. When incorporating receptor flexibility the user can see how the receptor changes shape in relation to a given position and orientation of a ligand. As they interact the user can learn about the change in shape and which parts of the molecules are key to causing receptor deformation during binding.

This research contributes to an interactive docking tool, DockIT [ILH22]. DockIT is designed to run with either keyboard and mouse or VR device, and simulates flexible-receptor docking in real time. This is vital for simulating protein binding accurately,

as proteins are naturally flexible. During the binding process, the proteins undergo conformational change, where the protein can assume a range of different poses.

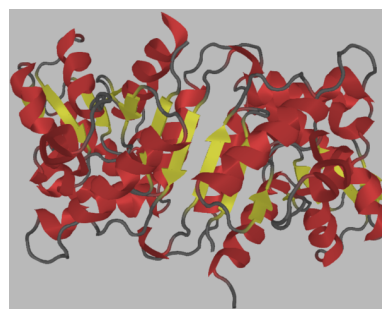


Figure 1: Cartoon representation of a protein, produced with DockIT.

For simulation on a screen, the minimum frame-rate needed is 30 fps but for VR this increases to 90 fps. To achieve real time deformation at these speeds, DockIT uses the results of a molecular dynamics (MD) simulation to generate a trajectory of atomic fluctuations. Using linear response theory this MD trajectory is used

to model, in real time, deformation of the receptor in response to interaction with a ligand [MKLH19]. Furthermore, the use of spatial decomposition and GPU parallel processing allows DockIT to handle docking scenarios with over 5000 atoms at 150 fps [ILH22].

DockIT already includes a number of different protein representations, such as space-fill, surface and ball-and-stick. These representations all show the protein's atoms, and in some cases bonds, explicitly. The cartoon representation [Ric81] is more abstract and shows the secondary structure of a protein. It is widely used for protein identification and highlighting key regions. There are three types of secondary structure used for the cartoon representation, sheets which are represented by flat arrows, helices which are represented by spirals and coil, represented by thin tubes. The placement of the helices and sheets is defined by hydrogen bonding patterns [KS83], which require re-assigning as the protein deforms.

DockIT is one of few interactive docking tools that can simulate docking with full flexible receptor. The novelty of this paper is the real-time definition and construction of the cartoon representation within a docking tool. To achieve this, the hydrogen bonds that are used to assign the secondary structure must be updated and the geometry must be reconstructed every frame. Note, in this context, the term cartoon representation does not refer to a style of rendering, such as toon shading.

2. Background

The cartoon representation of protein structure indicates the three main secondary structure types, α -helix, β -sheet and coil, and was first proposed by Richardson in the early 1980s [Ric81]. Hand-drawn sketches of the cartoon representation were common in textbooks and scientific articles, until the widespread use of molecular visualisation tools replaced these sketches with digitally rendered images. The relative positions and orientations of these secondary structure types are displayed, providing a high level, abstracted view. Using molecular graphics software, structural biologists use the cartoon depiction to recognise individual proteins and, perhaps more importantly, to identify key regions within their protein of interest. Proteins are made up of chains of amino acids, linked together by peptide bonds. The structure of a protein, and consequently its function, is largely determined by its sequence of amino acids along the chain. The secondary structure of a protein is defined by patterns of main-chain hydrogen bonds.

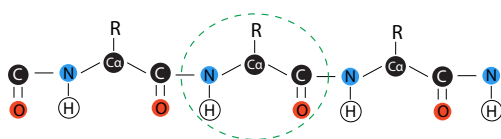


Figure 2: Diagram of a segment of a protein chain, showing the main-chain atoms as filled circles and the side chains as “R”. The large dotted circle indicates an individual amino acid residue.

Figure 1 shows the result of the approach we present here to construct and render the cartoon representation within DockIT. α -helices are represented by coiled ribbons. β -sheets, which comprise β -strands, can be either parallel, if both strands are in the same direction, or anti-parallel otherwise. β -strands are represented by

flat arrows. Connecting loop or coil regions are usually depicted by thin tubes.

Although secondary structures can be defined based on the main-chain ϕ and ψ angles, they are better defined by the hydrogen-bonding properties of the main chain. The first algorithm to rigorously define secondary structures was called “Define Secondary Structure of Proteins” (DSSP) [KS83]. It first finds main-chain hydrogen bonds which it then uses to identify elemental hydrogen-bonded motifs, such as the “n-turn” and the “bridge”. Overlapping 4-turns are used to assign α -helices, and bridges are used to assign β -sheets. The DSSP approach has proved to be the most popular method for assigning secondary structures and is still used today for proteins deposited in the Protein Data Bank (PDB) [BWF*00]. Since the development of DSSP, there have been many different examples of rendering the cartoon representation [HOF04, KBE08, HGVPVA15].

The geometry involved in showing the cartoon representation is much more complex compared to other representations. One effective way to reduce computation time is to use shaders to construct the geometry. Krone et al [KBE08] developed a method for real-time construction of the cartoon representation. Three versions of this algorithm were developed, one for the CPU, one for the GPU and a hybrid approach. The geometry shader was used to construct the cartoon representation, using two B-Splines to approximate the backbone shape and vectors around the control points to define the helix and sheet surfaces. Separate geometry shaders were used for helices, sheets and coils. Overall, the results show that the hybrid was the fastest at the time.

Hermosilla et al. [HGVPVA15] built on this work, with a GPU-based method for instant construction of the cartoon representation. A B-Spline was also used for the backbone approximation. However, instead of using vectors to define the geometry, they defined two surface patches oriented in opposite directions. Two surface patches are generated on the CPU for each section of the backbone, then they are displaced within the tessellation shader depending on the secondary structure assignment. Different subdivision levels were used depending on how far the geometry was from the viewer. This is an effective method for reducing construction times.

While previous approaches are effective at constructing and rendering the cartoon representation at real time speeds, they fail to take into account conformational changes that can occur during docking. One of the key aims of this work is to integrate the real time construction of the cartoon representation into an interactive docking tool. This involves more overhead, as DockIT not only renders the proteins, but also simulates flexible-receptor docking in real time.

3. Methods

The geometry construction and secondary structure assignment for the cartoon representation is based solely on the backbone atom positions. Data on a protein's structure, such as atomic positions and secondary structures, are stored in PDB-formatted files [BWF*00]. These files are loaded into DockIT by the user, but secondary structure assignments would only apply to the protein in its initial, undeformed conformation that appears upon loading into DockIT.

Figure 3 shows a diagram of how the code is structured between

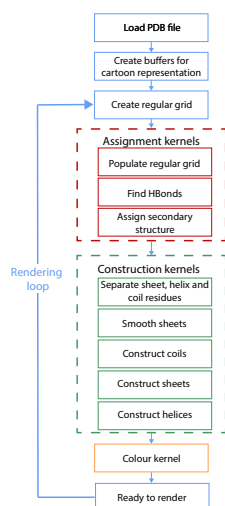


Figure 3: Flowchart outlining the method of constructing the cartoon representation in DockIT. Assignment kernels shown in red, construction kernels shown in green. Blue indicates parts run on the CPU.

CPU and GPU processing. This begins with loading the PDB file and extracting the atom information, which is stored in a common buffer used by many different representations within DockIT. Once all the atoms in the current molecule are loaded, the buffers for the cartoon representation can be created. This only needs to happen once, as the number of residues cannot change, so enough memory is reserved for the entire molecule. These buffers are then only written to and read from by the GPU kernels where possible, to minimise the amount of memory the CPU needs to read and write.

Within the rendering loop there are three main sections of GPU kernels: assignment, construction and colour (see Figure 3). The assignment kernels assign either sheet, helix or coil to each residue. This decision is made based on the DSSP rules [KS83], which use hydrogen bonding patterns to assign the secondary structure. This requires first assigning intramolecular hydrogen bonds, which is done by calculating the electrostatic interaction energy between hydrogen-bond donor atoms and hydrogen-bond acceptor atoms in two residues. To reduce the number of residues that are tested for hydrogen bonds, a regular grid is used to partition the molecule. Using a grid means that residues that are far apart on the chain but spatially close will be compared, without the need to check residues that are distant.

Once the hydrogen bonds have been assigned, patterns of bonds can be analysed for the existence of helices and sheets. To do this, each thread checks a single residue for n-turns, bridges or high curvature. This needs to be done on a per residue basis to avoid memory conflicts. Once each residue has been categorised, consecutive runs of n-turns and bridges can be identified and used to assign helices and sheets.

The construction kernels handle different aspects of the geometry construction. Before the geometry itself is constructed, each residue is written to either the helix, sheet or coil buffer, as they are each constructed by separate kernels. This is carried out at the start to prevent too many branches within the geometry kernels. After

this, the sheet smoothing kernel is run. This uses the method described by Preistle [Pri88] to reduce the appearance of pleats in the sheets over multiple iterations. Sheets are naturally pleated but it is a matter of choice whether to depict them with a pleat and this is an option in DockIT.

The geometry construction kernels all have similar features as the geometry is constructed using Catmull-Rom splines or surfaces. Catmull-Rom splines use four data control points but only connect the middle two, using them as end points for the resulting curve. This is similar for Catmull-Rom surfaces, as they require 16 control points and connect the central four. Catmull-Rom has been chosen over B-Spline as the resulting geometry will pass through every C α position exactly.

The geometry kernels are each launched with a block of 8×8 threads, equal to the number of points interpolated between the start and end of the surface patch. Each block of threads works on constructing one surface patch, using the atom positions from four consecutive residues. The helix, sheet and coil buffers contain the first residue position from this group to save memory. Each kernel writes the vertex and triangle information to common buffers which are then used for rendering.

3.1. Coil construction

The construction of coil segments uses a Catmull-Rom spline as a base. The first stage of the kernel only uses thread 0, which is treated as a set up thread and populates the geometry matrix used for interpolation. This matrix only needs to be created once, as each block of threads creates one surface patch. Then a set of 8 threads interpolate the central points using this geometry matrix. This creates a thin spline which is used as a centre for the tube.

A ring of points is then created around each interpolated spline position, each thread creating a single point which is then written to the vertex buffer. The ring of points is created using a simple circle equation, and is oriented to be perpendicular to the direction of the coil. After this, the triangles are indexed, again using each thread for one triangle.

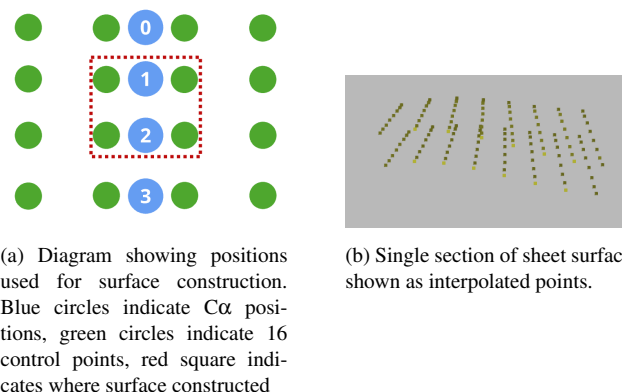


Figure 4: Helix and sheet construction diagrams

3.2. Helix and Sheet construction

The surface patches for helices and sheets both use Catmull-Rom surfaces as the base. To get the 16 control points needed, the orig-

inal 4 C α positions are duplicated. The direction between the carbonyl carbon and oxygen of each residue is used as an axis, and the C α position is duplicated twice, in both the positive and negative direction (see Figure 4a). The distance between the original position and the duplicated points is constant, so all the surface patches have the same width. These 16 points are then used to populate three geometry matrices, for each dimension, that can be used by each thread to interpolate a single point.

The resulting Catmull-Rom surface will span between the middle two C α positions, but is very thin. For some molecular visualisation tools [Jmo], thin surfaces like these are used. However, to match with Richardson's original drawings and most modern visualisation tools, 3D surface blocks are used. Each of the 8 \times 8 threads work on a single interpolated point, and displace it up and down along the normal resulting in two surface patches (see Figure 4b). For the sheets, the displacement value is constant in order to create a box shape, but helices require bevelled edges, so smaller displacement values are used at the edges.

4. Results

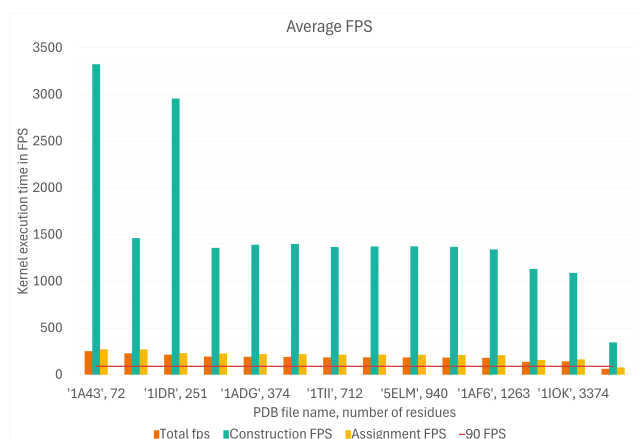


Figure 5: Graph showing the results of timing experiments for helix, sheet and coil construction kernels.

The main goal of this research was to achieve real-time flexible receptor docking with the cartoon representation. To quantify the success of this implementation, the time taken for each kernel to execute was recorded and averaged over 1000 runs. These tests were run on an Intel i7 processor running at 3.8GHz and an Nvidia GeForce RTX 2080. The OpenGL rendering pipeline was not modified for these tests, and standard back-face culling was applied. Overall, the molecules shown in Figure 5 achieved between 70 and 100 fps whilst performing all of the steps required for DockIT, including force calculations, deformation, cartoon construction and rendering.

Figure 5 also shows the execution times for the geometry construction kernels, which are generally stable. However, the near constant kernel execution times are caused by the size of the files used. The smaller files do not require enough work to reach maximum GPU occupancy. The larger files used show slightly longer execution times as this occupancy barrier has been passed. VR compatibility requires rendering speeds of 90 fps, which the majority of the current files tested achieve.

On average, the DSSP kernel was the least efficient, taking around 3.5ms for the smaller PDB files. This is in part caused by global memory reads of the regular grid. Each thread reads the contents from the current cell and neighbouring cells. This will be addressed by utilising shared memory.

5. Conclusion

The methods outlined in this paper produced some promising results, with proteins of up to 3,300 residues being constructed and rendered at 70 fps. Further work in this area will include optimisation of the DSSP kernel, to increase computation speeds even further, and visual enhancements to highlight the deformation of the backbone during docking. The end goal is to construct and render the cartoon representation fast enough to be used with the molecular surface during docking.

References

- [BWF*00] BERMAN H. M., WESTBROOK J., FENG Z., GILLILAND G., BHAT T. N., WEISSIG H., SHINDYALOV I. N., BOURNE P. E.: The protein data bank. *Nucleic acids research* 28, 1 (2000), 235–242. URL: www.rcsb.org. 2
- [HGVPA15] HERMOSILLA P., GUALLAR V., VINACUA PLA Á., VÁZQUEZ ALCOCER P. P.: Instant visualization of secondary structures of molecular models. In *Vcbm 15: eurographics workshop on visual computing for biology and medicine* (2015), European Association for Computer Graphics (Eurographics), pp. 51–60. 2
- [HOF04] HALM A., OFFEN L., FELLNER D.: Visualization of complex molecular ribbon structures at interactive rates. In *Proceedings. Eighth International Conference on Information Visualisation, 2004. IV 2004.* (2004), pp. 737–744. doi:10.1109/IV.2004.1320224. 2
- [ILH22] IAKOVOU G., LAYCOCK S. D., HAYWARD S.: Interactive flexible-receptor molecular docking in virtual reality using dockit. *Journal of Chemical Information and Modeling* (2022). URL: <https://doi.org/10.1021/acs.jcim.2c01274>, doi:10.1021/acs.jcim.2c01274. 1, 2
- [Jmo] Jmol DEVELOPMENT TEAM: Jmol. <http://jmol.sourceforge.net/> 2016. URL: <http://jmol.sourceforge.net/>. 4
- [KBE08] KRONE M., BIDMON K., ERTL T.: GPU-based visualisation of protein secondary structure. *TPCG* (2008), 1–8. 2
- [KS83] KABSCH W., SANDER C.: Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers: Original Research on Biomolecules* 22 (1983), 2577–2637. 2, 3
- [LM12] LEMMON G., MEILER J.: *Rosetta Ligand docking with flexible XML protocols*. Springer, 2012, pp. 143–155. 1
- [MHL*09] MORRIS G. M., HUEY R., LINDSTROM W., SANER M. F., BELEW R. K., GOODSELL D. S., OLSON A. J.: Autodock4 and autodocktools4: Automated docking with selective receptor flexibility. *Journal of Computational Chemistry* 30 (2009), 2785–2791. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21256>, doi:https://doi.org/10.1002/jcc.21256. 1
- [MKLH19] MATTHEWS N., KITAO A., LAYCOCK S., HAYWARD S.: Haptic-assisted interactive molecular docking incorporating receptor flexibility. *Journal of Chemical Information and Modeling* 59 (10 2019), 2900–2912. doi:10.1021/acs.jcim.9b00112. 2
- [Pri88] PRIESTLE J.: RIBBON: a stereo cartoon drawing program for proteins. *Journal of Applied Crystallography* (1988), 572–576. 3
- [Ric81] RICHARDSON J. S.: *The Anatomy and Taxonomy of Protein Structure*, vol. 34. Academic Press, 1981. URL: <https://www.sciencedirect.com/science/article/pii/S0065323308605203>, doi:https://doi.org/10.1016/S0065-3233(08)60520-3. 2