Instant visualization of secondary structures of molecular models

P. Hermosilla[1,2] & V. Guallar[2] & A. Vinacua[1] & P.P. Vázquez[1]

[1]Universitat Politècnica de Catalunya, Spain
[2]Barcelona Supercomputing Center, Spain

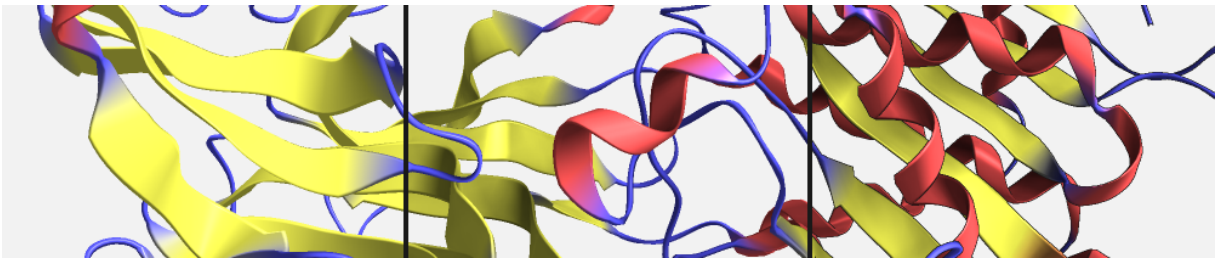

Figure 1: *The results of our algorithm. On the left, the the basic ribbons visualization for secondary structures generated with our adaptive method. The high framerates let us include additional effects, namely Ambient Occlusion (middle) and silhouettes (right), that enhance the perception of shapes while still maintaining realtime rendering times.*

Abstract

Molecular Dynamics simulations are of key importance in the drug design field. Among all possible representations commonly used to inspect these simulations, Ribbons has the advantage of giving the expert a good overview of the conformation of the molecule. Although several techniques have been previously proposed to render ribbons, all of them have limitations in terms of space or calculation time, making them not suitable for real-time interaction with simulation software. In this paper we present a novel adaptive method that generates ribbons in real-time, taking advantage of the tessellation shader. The result is a fast method that requires no precomputation, and that generates high quality shapes and shading.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

In many fields such as pharmacology, researchers are interested in the visualization of molecular simulations for example for drug design. Commonly, these simulations happen offline in high profile systems such as the Anton machine [DDG*12]. Nowadays, the simulation times of Molecular Dynamics software are still not realtime, but they are getting close. Thus, in order to better understand the process, the visualization of the results in realtime is becoming more and more important. Among the many rendering modes, ribbons lets the user easily understand the conformation of the molecules because it provides a higher abstraction for two types of structures inside the proteins: α-helices and β-sheets. These structures are visualized using a standard representation that consists of ribbons (thus originating their name) possibly with arrows. These ribbons are generated when concrete combinations of atoms are close to each other. As we will see later, these are complex geometries that change over time. Therefore, rendering such elements requires continuous changes over time that are not easy to handle. Previous techniques mainly use precomputed data. Other, more recent approaches generate the ribbons on-the-fly on the GPU [KBE08, WB11]. Despite its good results,

unfortunately, this strategy does not scale well, and may not be fast enough for large proteins.

In this paper we present an adaptive system that generates high quality shapes using the tessellation shader, with the following advantages:

- No preprocess: We generate the ribbons at each frame, without any precomputed data.
- Adaptive geometry generation: The geometry is generated with only the required detail, view dependently.
- Fully hardware accelerated: The geometry is completely generated in the GPU.
- High frame rate: Due to the previous two points, we achieve faster frame rates than other methods.

We achieve this by using a novel algorithm that makes intensive use of tessellation shaders. Our contributions are:

- A fast algorithm that uses the tessellation shader for the generation of ribbons.
- An adaptation of a previous Ambient Occlusion algorithm that generates high quality shading in real-time.

The rest of the paper is organized as follows: In Section 2 we give a brief introduction to biochemistry and analyze the related work. We describe our technique in Section 3 and in Section 4 we describe how the algorithm is implemented on modern GPUs. Section 5 deals with the ambient occlusion algorithm. We conclude the paper in Section 6 with a discussion of the results and in Section 7 with the conclusions and future work.

2. Background

2.1. Biomolecular Background

Protein are large biomolecules crucial for life as they carry out a large number of functions on the organism. Proteins replicate the DNA, catalyze metabolic reactions, transport molecules from one location to another and so on. They are composed by one or more chains or sequences of amino acid residues —figure 2. These amino acids are connected between them by peptide bonds and the order, type and length of the residue chains define the characteristics of the protein, as they are unique.

There are 23 types of amino acids which are part of the proteins and all of them have a common structure —H, N, C_α, C and O in figure 2— and a residue that is different for each type of amino acid —R in figure 2. The atoms of the common part of the amino acids on a protein are known as backbone, as they connect one amino acid to the next in the chain.

The peptide bonds between amino acids are not rigid and they can fold. This tridimensional structure may cause two non-consecutive amino acids to create a hydrogen bond between them leading to characteristic structures. These structures are known as secondary structures and the most common are:

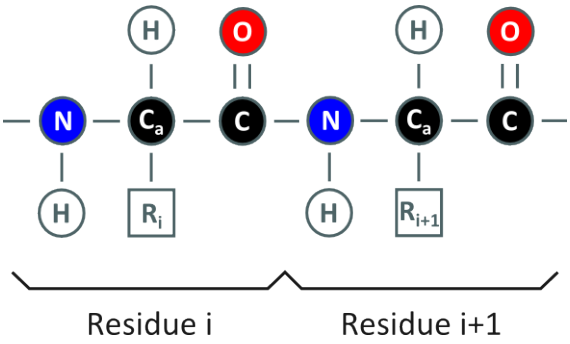

Figure 2: *Protein basics, amino acid chain structure.*

- α-helix: A part of the backbone acquires a form of an helix.
- β-sheet: Two distant parts of the backbone are interconnected in parallel.

2.2. Related Work

Molecular visualization is a relevant topic in visualization, as it helps researchers to understand the molecular processes. There are many representation methods used in molecular visualization. The most important are Space Filling [TCM06a, HOF05], where the molecule is represented at atom level using spheres for each one; Ball & Sticks [TCM06a, HOF05], similar to Space Filling, but where the bonds are represented with cylinders; Surfaces [PTRV12, SAMG14, SKR*14, LBH14], which try to highlight the accessible areas of the molecule to a small atom or molecule; and Ribbons representation. We address the reader to the state of the art presented by Kozlikova et al. [KKL*15] in molecular visualization to have a more accurate description of these rendering techniques.

The Ribbons representation method was popularized by Richardson [Ric81] and it was used later in molecular visualization as an elegant way to visualize the secondary structures of the molecules. Two main approaches have been used to visualize the secondary structures in ribbons mode, using geometry and using impostors. The method developed by Carson [Car91] to generate the ribbons is the most commonly used to extract the 3D information from the molecule, as it is fast to compute and produces a smooth representation. He used a B-Spline to interpolate the positions of the backbone and extract the representation from it. This work was extended by Halm et al. [HOF04] by adaptively triangulating the generated 3D geometry depending on the user point of view. Despite the number of triangles used in this representation is reduced considerably, all the computations are performed on the CPU and it still requires a big data transmission between CPU and GPU. Zamborsky et al. [ZSK09] proposed a method to reduce the amount of data stored for a molecular dynamics animation by interpolating

the backbone and then using a predefined segment representation for each segment. A method to reduce the data transmission between CPU and GPU was described by Krone et al. [KBE08], where the calculation of the final geometry was performed by the geometry shader. Wahle et al. [WB11] also proposed a method to perform the generation of the Ribbon structure on the GPU, but they only use the basic features to be able to execute it in old systems.

A completely different approach was developed by Bajaj et al. [BDST04], where they used ray-casting of implicit surfaces to represent the secondary structures. Despite it is a fast visualization method, the resulting quality is not as good as that obtained with previous methods. Other similar method was proposed by Bagur et al. [BSN12], where they used impostors to visualize molecules using different representation methods.

Two works related with ribbons visualization, but not with their generation, are the ones published by Weber [Web09] and by Van Der Zwan et al. [VDZLBI11]. Weber [Web09] developed a framework to visualize ribbons using illustrative rendering, and Van Der Zwan et al. [VDZLBI11] proposed a continuous transition between different visualization modes.

In spite of all the available algorithms to visualize Ribbons in real-time, commercial packages –as PyMol, RasMol or VMD– calculate the 3D geometry of the representation in a pre-process step and store it into main memory to render it later. Although this method is able to run in old computers, it is not suitable for large and/or dynamic simulations.

3. Algorithm

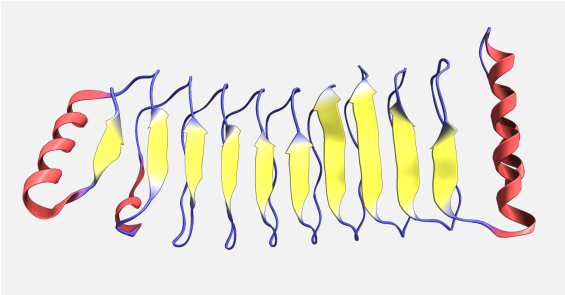

Figure 3: *Protein visualized using Ribbons mode. α-helices are colored in red, β-sheets are colored in yellow and the parts of the backbone that do not belong to any type of secondary structure are colored in blue.*

Popular molecular rendering packages as VMD [HDS96] or Maestro [Rel15] have different modes to visualize the secondary structures of a protein, but the most commonly used is the Ribbons mode. In this mode the α-helix secondary structures are represented by a ribbon helix, the β-sheet secondary structures are represented by a flat arrow and the parts of the backbone that do not belong to any type of secondary structure are represented by a tube. The figure 3 shows an example of a protein represented using this approach.

In order to obtain this smooth representation, we use a B-Spline to model the backbone of the protein. This algorithm was proposed by Carson [Car91], where he defined the control points of a B-Spline segment as the positions of the C_α atoms of four consecutive residues in the protein backbone, see Figure 5. To determine the orientation of the sheets we define a vector d for each control point as in [KBE08]. This vector is the direction between the C_α atom and the O atom of each residue. We use this approach instead of the the hydrogen bond direction, as they are virtually equal but the first one can be easily computed from the atoms of each aminoacid [CB86]. Using these vectors as the orientation of the segments can lead to flips in the resulting sheets if two of these consecutive directions have an angle greater than 90 degrees between them. In that case we flip one of them to avoid visual artifacts. We also define another vector n for each control point as the vector perpendicular to the spline direction and the d vector.

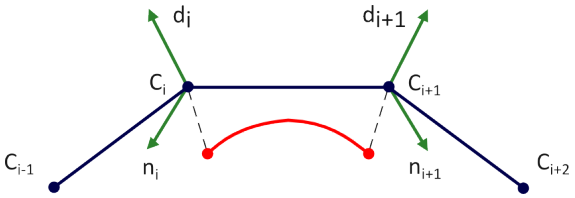

Figure 5: *Segment of the B-Spline. Control points are illustrated in blue, with their vectors used to align the sheet in green and the interpolated B-Spline segment is illustrated with the red curve.*

With this information —Figure 5— for each segment of the backbone we are able to generate the geometry needed to represent the protein secondary structures. We have designed a unified algorithm for all the secondary structure types based on rectangular patches. Each segment of the backbone is represented by two rectangular patches with opposite orientations —see figure 4. These patches are aligned with the backbone segment and oriented using the d and n vectors calculated previously. Then, they are subdivided according to the distance to the observer. The B-Spline is used to determine the new position of the vertex along the segment and the d and n vectors are interpolated using the B-Spline too. The new vertex is then displaced by the interpolated d and n vectors to its final position —see figure 7. This algorithm produces a round sheet that follows a smooth path along the backbone of the protein.

Despite this algorithm generates the geometry for the α-helices, we also need to generate the geometry for the β-sheets and the rest of the backbone, the arrows to represent the β-sheets and the tubes to represent the backbone. To do

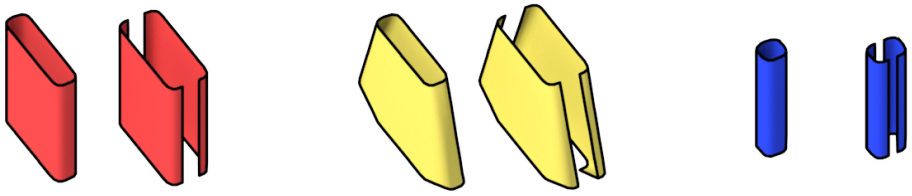

Figure 4: *Geometry used to represent a segment of different secondary structure types.* α*-helix on the left,* β*-sheet on the center and simple backbone segment on the right.*

so, we introduce a little modification on the algorithm: we modify the distance used to displace the vertex along the *d* vector for each type of secondary structure. Thus, we can generate sheets of different widths and, in the case of the backbone, we can generate tubes. These tubes are generated defining the displacements along the *d* vector equal to the displacement along the *n* vector, giving as a result a prism that represents the backbone —see figure 4-right.

The arrows are a bit more complicated to generate, since the displacements along the *d* vector have to be different along the same patch. These arrows have to be rendered in the last segment of each β-sheet, so we have to identify these last segments and define their displacement as a linear function. Figure 6 shows the plot of this function and the relation with the final geometry.

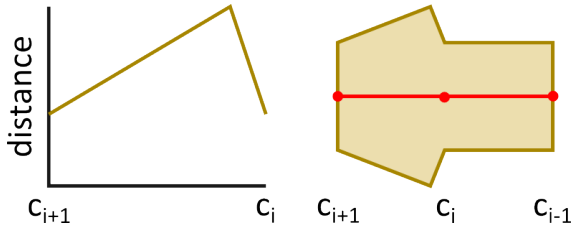

Figure 6: *Function used to determine the displacement on the end segments of the* β*-sheet secondary structures.*

At this point the algorithm is able to generate the geometry for each type of secondary structure, but we still have to define how to generate those segments where each endpoint belongs to a different type of secondary structure. At these segments a linear interpolation is performed in the colors and displacement distances used by the algorithm, creating a smooth transition between consecutive secondary structures —see figure 3.

4. Implementation

We have implemented this algorithm fully on the GPU in order to meet our requirements, reduce the amount of information to transfer between CPU and GPU and generate geometry with adaptive resolution for the current point of view. An overview of the process is shown in figure 8.

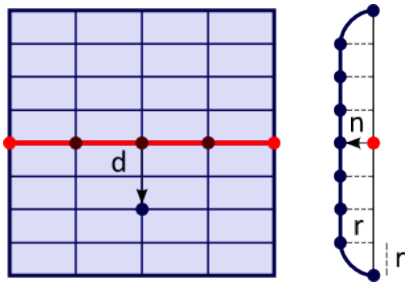

Figure 7: *Patch deformation to obtain the ribbon representation. The vertices of the patch are displaced using the d and n vectors of the B-Spline segment.*

We have divided the tasks in the four different stages in such a way that none of them requires a complex algorithm. In the following, we describe those stages in depth.

4.1. Tasks distribution

We describe here how the load is distributed into the different tasks. Note that we generate a rendering from scratch for each new position of atoms that is computed by the Molecular Dynamics software. We call each of these positions *frame*. The Molecular Dynamics software also provides the identification of the secondary structures to be rendered, also for each frame.

CPU

When the first frame arrives to our visualization software a vertex buffer and an index buffer are created. In the vertex buffer we store a vertex for each residue of our protein. A vertex in the buffer has the following information: C_α atom position, the direction of the *d* vector —see section 3, and an integer that encodes the type of secondary structure the residue belongs to. In the index buffer, we store four indexes for each segment of the B-Spline, which are the four control points used to evaluate the B-Spline at this segment. Note that only the vertex buffer has to be updated with new information for the next frames as the protein backbone structure remains the same.

With this information on the buffers the algorithm gen-

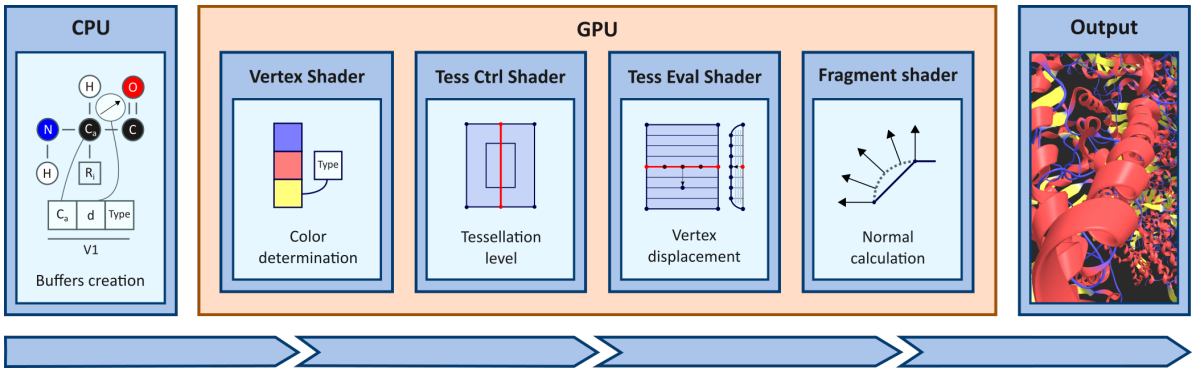

Figure 8: *Rendering pipeline: the load is balanced among the different stages of the algorithm. The CPU creates the data buffers, the vertex shader determines the colors, the tessellation stage is in charge of calculating the subdivision level required, and performing the tessellation, and finally, the fragment shader recomputes normals to provide a better shading effect.*

erates a patch for each segment of the backbone, but each segment has to be represented also by another one oriented in the opposite direction. To avoid introducing extra logic into the shaders we send the patches again with the order of the control points inverted, so that the algorithm automatically generates the patch with the new orientation. These new indexes are pushed at the end of the index buffer so all the patches can be rendered with a single drawcall.

Although this is a good and simple solution for the orientation problem, it introduces a new one, for now we cannot identify the end segments of the β-sheets where we ought to draw an arrow. The sense of direction is lost as now some segments are following one direction and the others are following the opposite. To solve this, we add an extra float for each control point that increases its value along the backbone, so the shader can determine the original direction of the backbone and which is the last segment of every β-sheet.

These buffers are sent to the graphics pipeline where each segment is interpreted as a patch of four vertexes.

Vertex shader

A vertex shader is executed for each vertex of the patch. The main duty of this stage is to pass all the information to the next stage and determine the color used to render the vertex. To do so, the vertex shader obtains the vertex color from an array using the secondary structure type as an index. The array holds the color chosen for each secondary structure type. The colors we have chosen to render the secondary structure types are red for α-helices, yellow for β-sheets and blue for the rest of the backbone —figure 4.

Tessellation control shader

The main task of this stage is to determine the tessellation level used to subdivide the patch.

The shader first calculates a tessellation factor for each vertex of the segment, C_i and C_{i+1}. These tessellation factors are in the range [0,1], where 0 is the minimum tessellation level and 1 is the maximum, and are calculated using the distance from the points towards the camera with the following equation:

$$Tess(C) = \frac{distance(C, camera) - min\_distance}{max\_distance}$$

The result of this formula is clamped to the range [0,1]. We have tested several distances to define the minimum and maximum tessellation factors and we found that 160Å and 10Å respectively produce good results without visual artifacts.

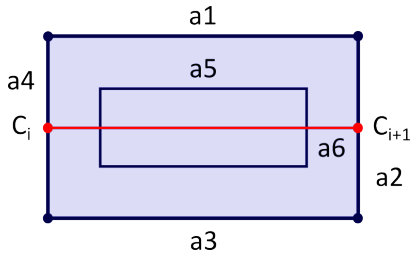

Figure 9: *Configuration of a tessellated patch on the GPU. The tessellation control shader determines the tessellation level for each edge of the patch.*

Hardware tessellated patches have a configuration as the one illustrated in figure 9, and the subdivision level is configured by the inner and outer tessellation factors. The outer tessellation factors define the subdivision level of the edges of the patch –a1, a2, a3 and a4 in figure 9– and the inner tessellation factors define the number of internal subdivisions of the patch –a5 and a6 in figure 9. Figure 10 shows an example of a possible patch subdivision. These factors are defined int the tessellation control shader using the following formulas:

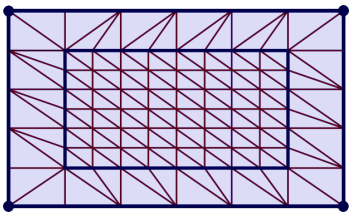

Figure 10: *Subdivided patch with 5 as tessellation factor for the edges a2 and a4, 6 for the edges a1, a3 and a6, and 8 for the edge a5.*

$$Tess(a1) = max(Tess(C_i), Tess(C_{i+1})) * SMax + SMin$$

$$Tess(a2) = Tess(C_{i+1}) * SSTMax_{i+1} + SSTMin_{i+1}$$

$$Tess(a3) = max(Tess(C_i), Tess(C_{i+1})) * SMax + SMin$$

$$Tess(a4) = Tess(C_i) * SSTMax_i + SSTMin_i$$

$$Tess(a5) = max(Tess(C_i), Tess(C_{i+1})) * SMax + SMin$$

$$Tess(a6) = max(Tess(a2), Tess(a4))$$

SMax and SMin are used to define the maximum and minimum subdivision level of the B-Spline, which in our case are 12 and 2. However, for the segments that generate the arrows of the β-sheets, the tessellation level of the B-Spline is always the maximum. If an adaptive tessellation level is allowed on these segments visual artifacts will appear, as the arrow will be changing its shape with the geometric resolution.

The values SSTMax and SSTMin are used to define the maximum and minimum subdivision level of the patch in the d direction. These values are different for each type of secondary structure, since their displacement is different. The maximum and minimum subdivision level for α-helices and β-sheets are 8 and 2 respectively and for the rest of the backbone both are set to 2. These values have been chosen empirically as they produce good results for all the tested view distances. Note that with this subdivision scheme there are no discontinuities between two consecutive patches as the secondary structure type of the corresponding vertex is used to subdivide the edges of the patch.

Besides the tessellation level calculation, the tessellation control shader also calculates the n vectors used to displace the new generated vertexes. These values are calculated by the cross product between the d vectors and the segment direction. To avoid discontinuities, the n vector for the control point i is calculated using the direction of the segment $i-1$ and the segment i, and then averaged.

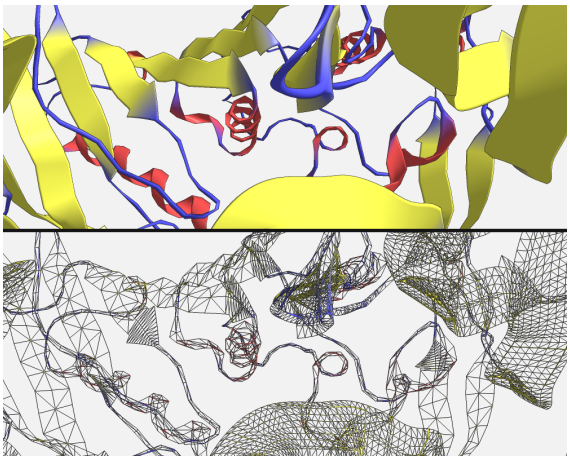

Figure 11: *Effect of the progressive tessellation. Note the different geometric resolution between near and far secondary structures —this effect has been exaggerated in this image in order to show the difference.*

Tessellation evaluation shader

The tessellation shader executes two important tasks: the evaluation of the resolution level required, and the actual subdivision of the geometry. The tessellation evaluation shader is executed for each new vertex created by the tessellation pipeline. With every execution a pair of coordinates are available which indicate the position of the new vertex inside the patch. Using these coordinates we move the new vertex to its corresponding position as described in section 3. The x coordinate is used to evaluate the B-Spline and interpolate the d and n, and the y coordinate is used to move the vertex along the interpolated vectors d and n as in figure 7.

As a result, we have different subdivision levels for the geometry that is close, and the geometry that is far from the observer. This has the main benefit of preserving the scalability of the algorithm. We show the different levels of subdivision in Figure 11. To provide the close view we have exaggerated in this case the subdivision levels, so that the different levels appear at closer distances than in normal operation of the algorithm. Note how our system allows changing the number of triangles depending on the depth. In order to illustrate the effect of the adaptive tessellation in a real case, we also provide a comparison of a fully tessellated representation and an adaptive one in Figure 12. The image on the right contains only triangles, that are really small since all the ribbons are created using the level that ensures they are correctly visible for near views. On the contrary, the left image shows how we save a lot of geometry with our adaptive method.

Fragment shader

The last stage of the pipeline, the fragment shader, is in charge of performing the lighting calculations to shade the

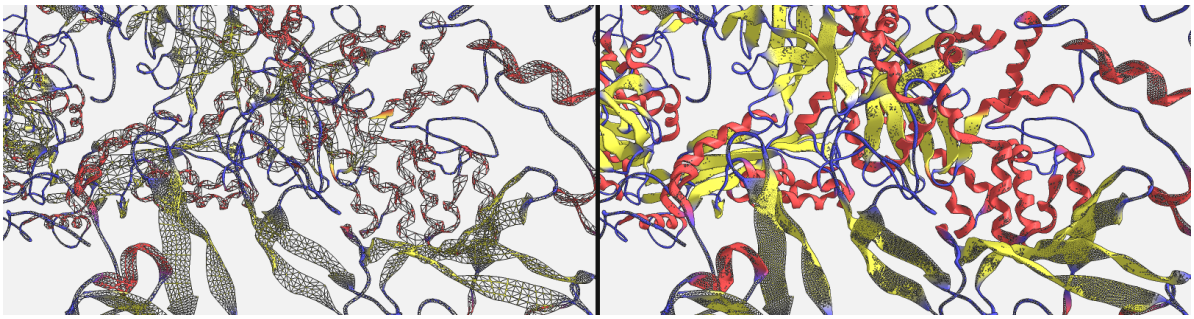

Figure 12: *Triangle reduction thanks to our adaptive tessellation –left– versus the regular tessellation that would guarantee correct images upon zooming –right. Note how the number of triangles is reduced in the left figure.*

surface of the sheet, but to avoid artifacts produced by the adaptive tessellation of the geometry, we calculate the normal also on this stage at pixel level. If the interpolated vertex normal is used instead, the lighting at the borders of the surfaces would change with the movement of vertexes carried out by the tessellation evaluation shader.

5. Ambient occlusion

Ambient occlusion is an algorithm that simulates the ambient light that arrives to a certain point from the rest of the scene.

There are many real-time approximations that can be divided in two types, the ones that approximate the ambient occlusion factor in screen-space and the ones that approximate it in object-space. The first ones [ESH13] are fast to compute since they only take into account the visible information, but they are not very accurate and suffer from artifacts with the movement of the camera —the information used to compute the ambient occlusion factor differs from one frame to another. In contrast, algorithms that work on object space [SGG15, GKSE12, TCM06b] are more accurate, as they take all the scene information into account to perform the ambient occlusion factor approximation. The problem with these algorithms is that their complexity grows with the size of the scene. We prefer to use an object space algorithm since it does not produce visual artifacts with the movement of the camera, which can be distracting to the scientists, and is more coherent with the geometry of the scene.

Our algorithm is based on a previous work of the same authors of this paper [HGVV15]. We give here a brief description of it for completeness. This algorithm was designed to approximate the ambient occlusion factor for molecules rendered using the space-filling or the ball & stick methods. The algorithm works as follows:

- First we create a coarse representation of the scene that is stored as an occupancy pyramid. Each level of this pyramid is a voxelization of the scene at a different resolution that stores an approximation of the occupancy of each voxel. This occupancy pyramid is updated at every frame —using the compute shader— by performing an intersection test between the scene primitives —spheres and cylinders— and the voxels and approximating the overlap between them.
- In a second pass the ambient occlusion factor is calculated for each pixel using the voxel cone tracing algorithm [CNS*11] through the occupancy pyramid.

To adapt this algorithm to our scene we approximate the ribbons geometry by a set of boxes —3 for each segment— with different sizes, and then compute their overlap against all levels of the occupancy pyramid. An intersection test is performed between the box volume and the volume of the voxels —approximated by spheres— and the overlap with each voxel is then estimated as in the original paper. Since we have a more sparse scene, we scale this overlap approximation by a factor that can be modified by the user in order to adjust the intensity of the generated shadows. Attempting to approximate each segment by a variable number of boxes —as the geometry of the segments is adaptive too— would be counter-productive, since it would lead to visible popping artifacts on the shadowed areas.

We show an example of our implementation of ambient occlusion for ribbons models in Figure 13.

6. Results

In this section we provide some results for the rendering of different molecules and compare with other methods.

We run a series of tests for molecules occupying a central part of the screen (we call it *far* or *F*) and with a zoom-in of the same molecule (called *Near*, or *N*) —where the maximum tessellation level is achieved by the near geometry— to see the effect of the screen coverage. All these tests have been executed on a Intel Core i7 PC, running at 3.5GHz, with 16Gb of RAM, and a GeForce 770 GTX, and rendering in a 1280×720 viewport. The different molecules go from a simple molecule of 249 residues to a large example of

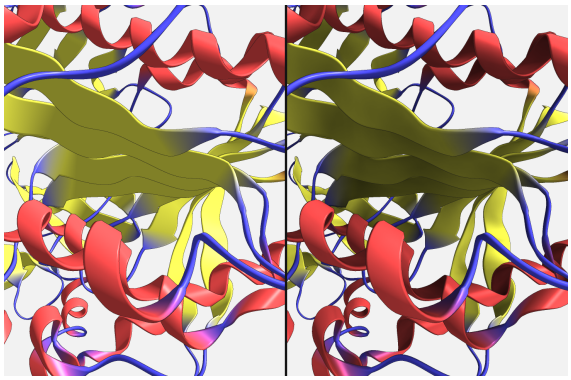

Figure 13: *Extra effects added to the typical molecule shading in order to improve the perception of its elements. On the left, the molecules is shaded using only Phong shading and silhouettes. The right image also incorporates ambient occlusion to the Phong shading and silhouettes.*

Mol name	A	1CWP	3IYJ	3IYN
#residues	249	29220	171720	749340
NoAO	2582.18	726.41	244.17	70.01
AO	436.83	340.64	117.18	35.06

Table 2: *Performance measured in frames per second for different molecules without ambient occlusion (NoAO) and with it (AO). Note that even with ambient occlusion activated, our system is able to render complex molecules in realtime. Even molecules whose size exceeds what is commonly needed for the pharmacological simulations we deal with.*

749K residues —figure 15. Note that the latter case is exceptionally large, orders of magnitude larger than the ones used in the pharmacological simulations we are addressing. We can see in Table 1 how the framerates we achieve are larger than the method that uses the geometry shader to create the secondary structures –Figure 14. Note how we achieve realtime framerates even with the largest molecule. Moreover, the method scales very well with the size of the molecule. There is also hardly any difference between the version that uploads the buffers to the GPU at every frame —shown as *U*— and the method that does not —*NU* in the table, as the data transmitted from CPU to GPU consists only of the backbone information.

We also provide a comparison of the performance of our algorithm when adding ambient occlusion. Roughly, the framerates decay to half —except for the first model, that was actually so fast to render that the AO calculation dominates the cost—. In Table 2 we can see those framerates. Note that we still maintain real-time framerates for the most complex molecules.

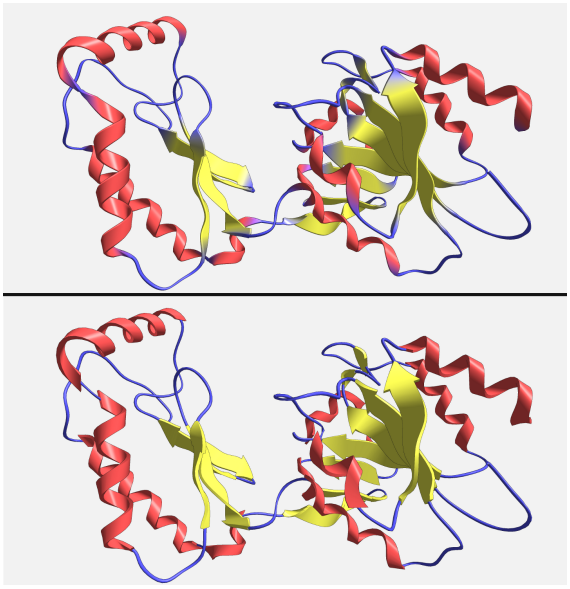

Figure 14: *Visual comparison of our method (top) with the method proposed by Krone et al. [KBE08] (bottom).*

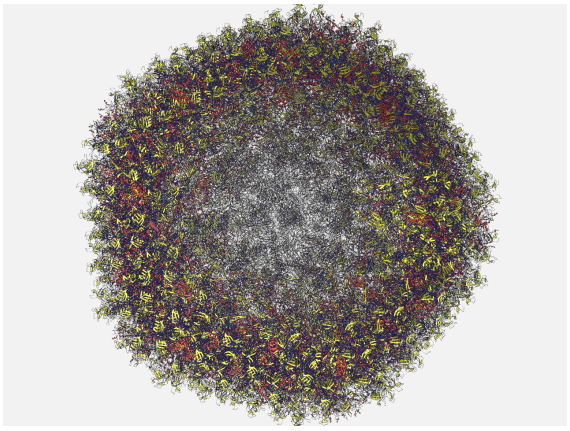

Figure 15: *Far view of the molecule 3IYN with 749K residues. With this complex molecule, with far more atoms than the required for common pharmacology simulations, we still obtain framerates of around 50 fps.*

7. Conclusions and Future work

We have presented an algorithm capable of generating the secondary structure geometry of very complex proteins on-the-fly using the tessellation stage of the GPU. While other existing algorithms use the GPU to generate the geometry on-the-fly, ours is capable of generating only the geometry needed for the current point of view, allowing the interaction with bigger molecules in real-time and the use of other rendering effects in order to increase the visual quality of the

Molecule name		A		3EXG		1CWP		3IYJ		3IYN	
#residues		249		10781		29220		171720		749340	
VBO update		NU	U	NU	U	NU	U	NU	U	NU	U
Our	N	2594.21	2504.29	1198.53	1059.86	760.75	667.96	246.47	198.72	68.98	48.41
	F	3596.47	3388.34	2375.83	2093.71	1330.46	1078.64	290.35	222.94	72.51	50.19
[KBE08]	N	1531.91	1388.87	263.57	254.99	86.84	85.44	14.48	14.31	3.52	3.36
	F	1946.81	1838.98	287.75	274.18	90.06	89.51	14.46	14.56	3.54	3.39

Table 1: *Performance measured in frames per second for different molecules (N)ear the camera (where the near geometry has the maximum tessellation level) and (F)ar from the molecule (where all the geometry has the minimum tessellation levels). Ambient occlusion has been computed at the highest quality (nine cones per point). "Our" indicates the performance measured with our method and " [KBE08]" are the frames per second measured with the method proposed by Krone et al. [KBE08] –using 5 segment for each curve section and 6 edges of the tube's front and back faces. Framerates have been measured without uploading the buffers to the GPU (NU) and uploading them (U).*

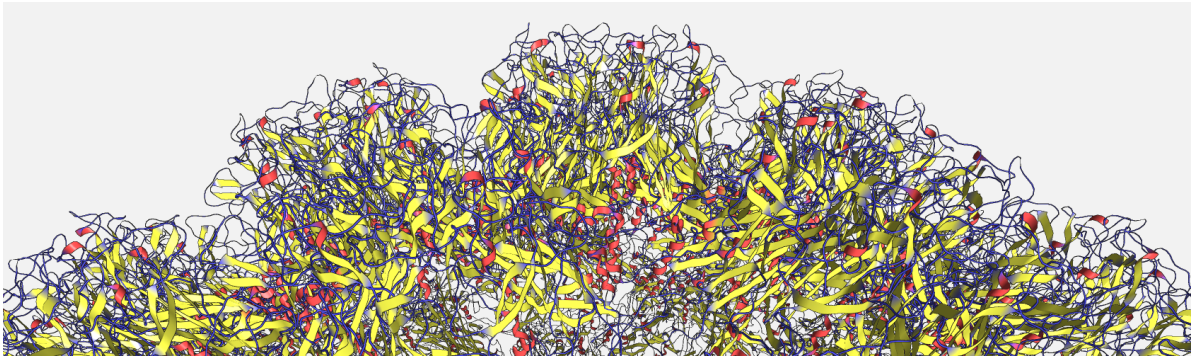

Figure 18: *Close view of the molecule 3IYJ with 171K residues. The rendering framerates are well above 200 fps for the regular views, and closer views oscillate between 198 and above for the different configurations.*

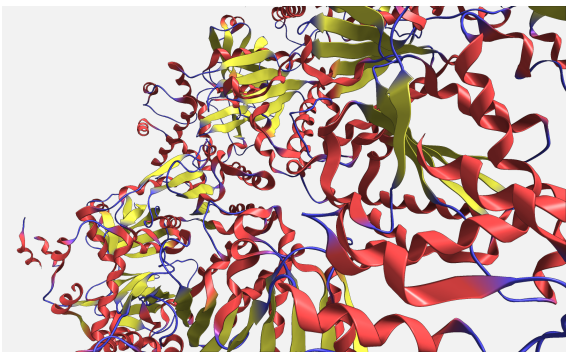

Figure 16: *Molecule 1S3S with 2940 residues.*

rendering. We have also adapted an existing method to calculate the ambient occlusion factor in object space, obtaining real-time frame rates even for macromolecules.

As future work we are planning to carry out a formal user study to determine which is the most useful render method for the scientists and also to compare our method to other approaches. We are also planning to explore adapting this method in order to improve the rendering in other areas of scientific visualization. The algorithm used to represent the parts of the backbone that do not belong to any secondary structure could be used to represent high quality lines and brain fibers.

Acknowledgements

This work has been supported by the projects TIN2013-47137-C2-1-P and TIN2014-52211-C2-1-R of the Spanish Ministerio de Economía y Competitividad, and the project 2014-SGR 146 from the Catalan Government.

References

[BDST04] BAJAJ C., DJEU P., SIDDAVANAHALLI V., THANE A.: Texmol: Interactive visual exploration of large flexible multi-component molecular complexes. In *Proceedings of the conference on Visualization'04* (2004), IEEE Computer Society, pp. 243–250. 3

[BSN12] BAGUR P. D., SHIVASHANKAR N., NATARAJAN V.:

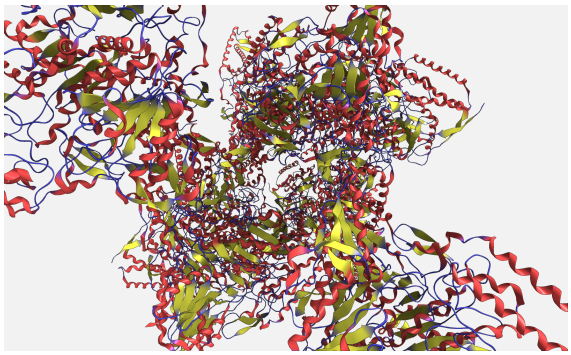

Figure 17: *Molecule 3EXG with 10781 residues. Molecules with this complexity level are rendered with our algorithm at framerates between 1000 and 2000 fps, even when using ambient occlusion, like in this example, the framerates are above 400 fps.*

Improved quadric surface impostors for large bio-molecular visualization. In *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing* (2012), ACM, p. 33. 3

[Car91] CARSON M.: Ribbons 2.0. *Journal of Applied Crystallography 24*, 5 (1991), 958–961. 2, 3

[CB86] CARSON M., BUGG C.: Algorithm for ribbon models of proteins. *J.Mol.Graphics*, 4 (1986), 121–122. 3

[CNS*11] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN E.: Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum 30*, 7 (2011), 1921–1930. 7

[DDG*12] DROR R. O., DIRKS R. M., GROSSMAN J., XU H., SHAW D. E.: Biomolecular simulation: a computational microscope for molecular biology. *Annual review of biophysics 41* (2012), 429–452. 1

[ESH13] EICHELBAUM S., SCHEUERMANN G., HLAWITSCHKA M.: PointAO - Improved Ambient Occlusion for Point-based Visualization. In *EuroVis - Short Papers* (2013), Hlawitschka M., Weinkauf T., (Eds.), The Eurographics Association, pp. 013–017. 7

[GKSE12] GROTTEL S., KRONE M., SCHARNOWSKI K., ERTL T.: Object-space ambient occlusion for molecular dynamics. In *Pacific Visualization Symposium, 2012 IEEE* (2012), IEEE, pp. 209–216. 7

[HDS96] HUMPHREY W., DALKE A., SCHULTEN K.: VMD: visual molecular dynamics. *Journal of molecular graphics 14*, 1 (1996), 33–38. 3

[HGVV15] HERMOSILLA P., GUALLAR V., VINACUA A., VÁZQUEZ P.: High quality illustrative effects for molecular rendering. *Computers & Graphics* (2015), –. URL: http://www.sciencedirect.com/science/article/pii/S009784931500120X, doi:http://dx.doi.org/10.1016/j.cag.2015.07.017. 7

[HOF04] HALM A., OFFEN L., FELLNER D.: Visualization of complex molecular ribbon structures at interactive rates. In *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on* (2004), IEEE, pp. 737–744. 2

[HOF05] HALM A., OFFEN L., FELLNER D. W.: Biobrowser: A framework for fast protein visualization. In *EuroVis* (2005), Brodlie K., Duke D. J., Joy K. I., (Eds.), Eurographics Association, pp. 287–294. 2

[KBE08] KRONE M., BIDMON K., ERTL T.: Gpu-based visualisation of protein secondary structure. *TPCG 8* (2008), 115–122. 1, 3, 8, 9

[KKL*15] KOZLÍKOVÁ B., KRONEY M., LINDOW N., FALK M., BAADEN M., PARULEK J., HEGE H.-C.: Visualization of Molecular Structure: The State of the Art. In *EuroVis 2015 State of the Art Reports* (2015). 2

[LBH14] LINDOW N., BAUM D., HEGE H.-C.: Ligand excluded surface: A new type of molecular surface. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 2486–2495. 2

[PTRV12] PARULEK J., TURKAY C., REUTER N., VIOLA I.: Implicit surfaces for interactive graph based cavity analysis of molecular simulations. In *Biological Data Visualization (BioVis), IEEE Symposium on* (2012), IEEE, pp. 115–122. 2

[Rel15] RELEASE S.: 1: Maestro. *Schrödinger, LLC, New York, NY* (2015). 3

[Ric81] RICHARDSON J. S.: The anatomy and taxonomy of protein structure. In *Advances in Protein Chemistry*, C.B. Anfinsen J. T. E., Richards F. M., (Eds.), vol. 34 of *Advances in Protein Chemistry*. Academic Press, 1981, pp. 167 – 339. 2

[SAMG14] SARIKAYA A., ALBERS D., MITCHELL J., GLEICHER M.: Visualizing validation of protein surface classifiers. *Computer Graphics Forum 33*, 3 (2014), 171–180. 2

[SGG15] STAIB J., GROTTEL S., GUMHOLD S.: Visualization of Particle-based Data with Transparency and Ambient Occlusion. In *Computer Graphics Forum* (2015). 7

[SKR*14] SCHARNOWSKI K., KRONE M., REINA G., KULSCHEWSKI T., PLEISS J., ERTL T.: Comparative visualization of molecular surfaces using deformable models. *Computer Graphics Forum 33*, 3 (2014), 191–200. 2

[TCM06a] TARINI M., CIGNONI P., MONTANI C.: Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (Sept. 2006), 1237–1244. doi:10.1109/TVCG.2006.115. 2

[TCM06b] TARINI M., CIGNONI P., MONTANI C.: Ambient occlusion and edge cueing for enhancing real time molecular visualization. *Visualization and Computer Graphics, IEEE Transactions on 12*, 5 (2006), 1237–1244. 7

[VDZLBI11] VAN DER ZWAN M., LUEKS W., BEKKER H., ISENBERG T.: Illustrative molecular visualization with continuous abstraction. *Computer Graphics Forum 30*, 3 (2011), 683–690. 3

[WB11] WAHLE M., BIRMANNS S.: Gpu-accelerated visualization of protein dynamics in ribbon mode. In *IS&T/SPIE Electronic Imaging* (2011), International Society for Optics and Photonics, pp. 786805–786805. 1, 3

[Web09] WEBER J. R.: Proteinshader: illustrative rendering of macromolecules. *BMC structural biology 9*, 1 (2009), 19. 3

[ZSK09] ZAMBORSKY M., SZABO T., KOZLIKOVA B.: Dynamic visualization of protein secondary structures. In *Proceedings of the 13th Central European Seminar on Computer Graphics (CESCG)* (2009), pp. 147—-152. 2