

Christoph Müller¹ / Michael Krone¹ / Markus Huber¹ / Verena Biener¹ / Dominik Herr^{2,3} / Steffen Koch² / Guido Reina¹ / Daniel Weiskopf^{1,2} / Thomas Ertl^{1,2}

Interactive Molecular Graphics for Augmented Reality Using HoloLens

¹ Visualisation Research Centre (VISUS), University of Stuttgart, 70569 Stuttgart, Germany, E-mail: christoph.mueller@visus.uni-stuttgart.de

² Institute for Visualisation and Interactive Systems (VIS), University of Stuttgart, 70569 Stuttgart, Germany

³ Graduate School Advanced Manufacturing Engineering (GSaME), University of Stuttgart, 70569 Stuttgart, Germany

Abstract:

Immersive technologies like stereo rendering, virtual reality, or augmented reality (AR) are often used in the field of molecular visualisation. Modern, comparably lightweight and affordable AR headsets like Microsoft's HoloLens open up new possibilities for immersive analytics in molecular visualisation. A crucial factor for a comprehensive analysis of molecular data in AR is the rendering speed. HoloLens, however, has limited hardware capabilities due to requirements like battery life, fanless cooling and weight. Consequently, insights from best practises for powerful desktop hardware may not be transferable. Therefore, we evaluate the capabilities of the HoloLens hardware for modern, GPU-enabled, high-quality rendering methods for the space-filling model commonly used in molecular visualisation. We also assess the scalability for large molecular data sets. Based on the results, we discuss ideas and possibilities for immersive molecular analytics. Besides more obvious benefits like the stereoscopic rendering offered by the device, this specifically includes natural user interfaces that use physical navigation instead of the traditional virtual one. Furthermore, we consider different scenarios for such an immersive system, ranging from educational use to collaborative scenarios.

Keywords: Bioinformatics, HoloLens, Augmented reality, Molecular visualisation, GPU, Scientific visualisation, Immersive Analytics


DOI: 10.1515/jib-2018-0005

Received: January 22, 2018; **Revised:** April 19, 2018; **Accepted:** May 9, 2018

1 Introduction

The software tools developed in bioinformatics to support the understanding of data, for instance, in structural biology or computational chemistry traditionally make use of scientific visualisation to analyse data, for example simulated protein interactions. Popular desktop tools for molecular visualisation like VMD [1] or PyMOL [2] often support stereoscopic rendering, for example using head-mounted virtual reality (VR) displays like Oculus Rift¹, or powerwalls [3] and CAVE-like systems [4]. This rendering mode helps to convey the complex spatial structure of molecular data. Augmented reality (AR) has so far not been used extensively, which is probably due to the lack of convenient and affordable hardware. Microsoft's HoloLens² is a head-mounted see-through AR display which offers much functionality out of the box, especially the registration of the environment, which considerably reduces the effort required by developers for implementing AR software, making it thus feasible to quickly create various augmented experiences. Furthermore, the availability of such affordable commodity hardware broadens the potential audience. However, technical restrictions of the hardware prevent direct reuse of desktop software, which usually requires much processing power like the massively parallel architecture of modern high-performance GPUs. HoloLens features an Intel Atom x5-Z8100P CPU (1.04 GHz), 2 GB RAM, 64 GB eMMC and a custom-made Holographic Processing Unit (HPU) coprocessor exclusively used for spatial tracking. Overall, very little is known about the graphics capabilities of the whole system. So far, it has not yet been thoroughly investigated how HoloLens can handle molecular visualisation using state-of-the-art rendering methods tailored to modern desktop hardware. Since fast, high-quality rendering is crucial for AR, this is an important first step towards an immersive analytics application for molecular visualisation. As a rule of thumb, software for AR headsets should maintain at least 60 fps to ensure user comfort and judder-free rendering that stays in sync with the registered environment. Due to the limited processing power of HoloLens, Microsoft

Christoph Müller is the corresponding author.

 ©2018, Christoph Müller et al., published by DeGruyter.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.

provides extensive performance recommendations to developers³, which suggest that the typical rendering approaches for desktops are not a good match, as discussed in Section 5.

In this paper, we evaluate the rendering capabilities of HoloLens for molecular graphics and present possible solutions and guidelines for scientific visualisations. We use the space-filling model for our scaling tests, because the fact that it is the computationally least expensive representation should be beneficial with respect to the limited computational power of the HoloLens. This model uses a simple sphere to represent each atom, which of the radius corresponds to the atomic radius (or the van der Waals radius) of the chemical element. Figure 1 shows a small protein rendered as space-filling model, where the atom spheres are coloured by element (so-called CPK colouring).

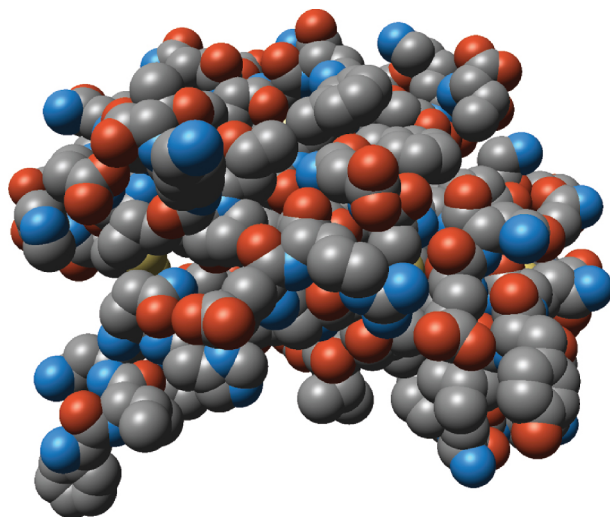


Figure 1: Rendering of the space-filling model for a small protein (PDB ID: 1RWE) coloured by element.

In traditional computer graphics, objects like spheres have to be tessellated as triangle meshes prior to rendering. Today, GPU-based raycasting [5], [6] is the fastest way to render large numbers of implicitly defined objects of low polynomial degree (e.g. quadrics) [7]. Here, a small proxy geometry covering the whole object is generated and rendered. The pixel shader (PS) then computes a ray from the camera through the current fragment and tests it for intersection with the implicit object – the sphere in our case. This way, the technique cannot only guarantee the best visual representation of a sphere, but it is also the fastest method to render large numbers of particles on desktop hardware [8], [9]. Figure 2 shows a schematic of this rendering method, which is, for example, also implemented in the molecular visualisation tools VMD [1] and MegaMol [10].

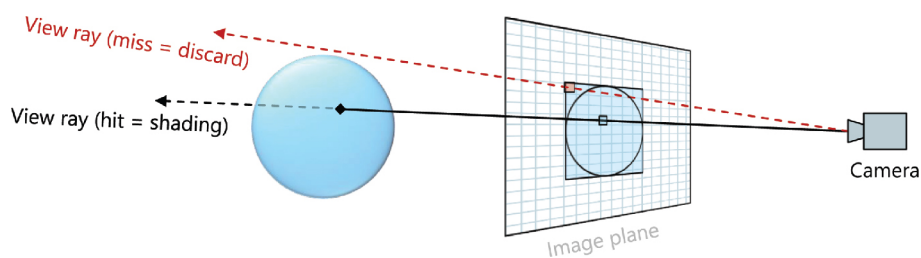


Figure 2: Schematic drawing of GPU-based raycasting: for each sphere, a proxy geometry that covers the whole sphere is rendered (blue quad in the image plane). For each fragment of this quad, a view ray is computed in the pixel shader and a ray-sphere intersection determines whether the actual sphere is visible through this pixel. Our objective is to assess the rendering speed of the custom GPU used by HoloLens for such shader-based methods. We also compare this approach to traditional rendering methods based on triangle meshes.

As test environment, we developed a prototypical molecular visualisation software specifically tailored to HoloLens. Our custom prototype is a Universal Windows Platform (UWP) app written in C++ using the native Direct3D API and HLSL shaders. We use real-world molecular data sets obtained from the RCSB Protein Data Bank (PDB) [11] to evaluate the performance. As the most popular and convenient way to develop software for the HoloLens is the Unity game engine⁴, we also implemented a Unity prototype that can load and visualise PDB data sets and compared the frame rates with our custom prototype.

2 Related Work

Our work is about the evaluation of rendering performance and the scalability of Microsoft's HoloLens for molecular visualisation. The rendering performance is an essential factor for the usability of HoloLens for immersive analytics, i.e. the combination of immersive techniques like AR or VR, natural user interfaces and visualisation to foster data analysis. For a concise definition of immersive analytics, we refer to the recent overview paper by Chandler et al. [12].

Molecular visualisation itself is a very diverse field in bioinformatics. In this work, we focus mainly on visualisation for structural biology, i.e. on three-dimensional visualisations of molecules with atomistic detail. There is a large variety of established representations used in this field; an in-depth overview of visualisation methods for biomolecular data can be found in the recent survey by Kozlíková et al. [13]. We use the space-filling model, which we expect to be best suitable for HoloLens thanks to the simplicity of its visual primitives.

As mentioned in the introduction, stereoscopic rendering is widely used for molecular visualisation. Beside established tools like PyMol or VMD, a range of specialised immersive visualisation tools for molecular data have recently emerged that make use of the capabilities of modern VR head-mounted displays (HMDs) like Oculus Rift or HTC Vive⁵. Examples are Molecular Rift [14], which is intended for drug designers, or the Caffeine molecular viewer [15], [16], which can also visualise the results of quantum mechanics computations. Recently, Molecular Rift has been extended by adding support for the Leap Motion⁶ controller [17], which offers hand tracking via an infrared camera. The black-and-white image of the Leap attached to the front of the HMD is also used as background for the visualisation, giving users a sense of their surroundings, thus moving in the direction of an AR application.

AR has been used for molecular visualisation, too. However, the technical approaches differ widely. Early work by Gillet et al. [18] uses a camera to capture a 3D-printed molecule. The position and orientation of the molecule is obtained via markers that are attached to the 3D print. Thus, the molecule in the captured video can be enhanced with additional information such as the electrostatic field. A similar, camera-based AR setup has more recently been used by Berry and Board [19]. Their tool, which is intended for educational use, also recognises printed markers in a live-captured video and renders different molecular models at this location. The user can investigate the virtual molecule by moving the camera or the markers. Similar tools were developed for mobile devices like smartphones or tablet PCs, for example Miew⁷. An alternative to such camera-based setups are modern see-through AR HMDs like HoloLens. Zheng and Waller [20] presented ChemPreview, an AR molecule viewer that works with the Meta 1, an AR HMD that is similar to HoloLens, but has to be connected to a desktop PC that handles the rendering. Hoffman and Provance [21] described how to create an AR application based on the Unity game engine for HoloLens. However, in contrast our work, they only used precomputed models of small data sets and did not evaluate the performance or technical limitations of the hardware. Another software similar to our work is the HoliMoli app⁸ by Sobhani, which uses the tessellation shader to create triangulated spheres for the space-filling representation. We included this rendering method in our performance tests (cf. Section 4). The app Holocule, which is available from the Microsoft Store⁹, renders different representations of a molecule, including ball-and-stick and cartoon. The techniques used in this app are unknown, but from its performance and visual appearance, we assume that at least the ball-and-stick representation uses a raycasting-based method.

As mentioned above, none of the existing work on molecular visualisation using HoloLens contains a rigorous testing of the rendering performance, which, however, plays an important role for many applications. Therefore, it comes at no surprise that this topic is touched upon in a variety of works from other areas. Voinea et al. [22] mention the problem of predicting CPU and GPU performance due to power/thermal throttling for biomechanical simulations. The requirements for rendering reconstructed three-dimensional models for telepresence applications that are transmitted to the HoloLens using a WiFi connection are discussed by Joachimczak et al. [23]. They propose sending models with a low number of polygons and counteract the loss in model detail with higher resolution of texture data to realise acceptable frame rates using the HoloLens hardware. Chen et al. [24] compare the performance of different wearable devices including the HoloLens with respect to latencies occurring when running different edge computing applications. These previous works mention the importance of, and problems with, rendering performance of the HoloLens, but quantitative evaluations dedicated to this specific aspect are still missing. With our approach, we would like to shed first light on what can be expected in terms of HoloLens' rendering performance and how this impacts the choice of application scenarios.

3 HoloLens Prototype Implementations

3.1 Standalone UWP Application

HoloLens is based on Windows 10 and can therefore run most universal Windows apps in a floating window that can be placed somewhere in the real world. Immersive applications like ours, which allow the user to place virtual objects (holograms) in the real world, differ in that they use a fullscreen window and the Direct3D API for rendering their 3D content. In contrast to standard 3D applications, HoloLens takes care of all low-level tasks like managing the viewport, the coordinate systems and transformations. That is, the device chooses an appropriate coordinate system based on the surroundings it discovers with its sensors and provides the view and projection transformations for rendering. This enables the programmer to place objects relative to the user's current physical position, and HoloLens makes sure that they stay there as the user moves. All coordinates in the software development kit (SDK) are given in metres, i.e. if an object is placed 2 m in front of the virtual camera, it appears 2 m in front of the user. Likewise, the size of the object specified in virtual units directly matches its apparent size in metres. HoloLens additionally provides the user with a three-dimensional model of the surroundings allowing virtual objects to be occluded by real-world ones.

Our prototype is a Direct3D-based UWP app, which renders molecules from the Protein Data Bank using different techniques. On the GPU, we try to use the minimal representation required to render an atom, which is its position, radius and colour. Using this input, we implemented several variants of two basic approaches: the first one is generating primitives resulting in a screen-space bounding geometry of each atom (sprite or billboard) and raycasting a sphere for each pixel covered by the sprite. This is currently the fastest approach on desktop-class GPUs and the most suitable for dynamic data as it minimises the amount of data to be streamed to the GPU (a float4 vector for a sphere without properties beside its position and radius). There are different ways of generating the sprites for raycasting, which are described later. However, all of them eventually yield the same image, which is a pixel-perfect sphere, regardless of the distance between the camera and the object. The second and fundamentally different approach is rasterising and shading the triangle representation of a sphere for every atom, which is the traditional interactive computer graphics approach. Its visual quality depends on how many triangles are used per sphere and how close the camera is. At large distances, differences to the raycasted spheres might be barely noticeable, but if the camera is close to the sphere, individual triangles will become visible around its silhouette.

The original approach for raycasting spheres on the GPU suggests using point sprite primitives, i. e. point primitives with a variable extent in screen space [5], [6]. However, such point sprites are unsupported starting with Direct3D 10. Therefore, our first approach expands the single vertex representing an atom into the four corners of a sprite using the geometry shader (GS), which was introduced in Direct3D 10. A similar effect can be achieved using dynamic tessellation capabilities, which are available starting with Direct3D 11. Dynamic tessellation is separated in three phases: first, the hull shader (HS) computes the control points of a patch (a quad in our case) from the control points of a lower-order representation (the single vertex representing an atom in our case) as well as per-patch constants like the subdivision level for the following second stage, the non-programmable tessellation. The third and final domain shader (DS) stage moves the vertices created in the tessellation stage to their final location. In our case, the polygon generating the pixels for raycasting is resized to match the radius of the atom and oriented towards the camera. The number of vertices of the polygon is a trade-off between the computational load of transforming additional vertices and the number of pixels that fail the hit test of the sphere: in case of a quad, there are more pixels for which the pixel shader is executed, but which are eventually empty (see Figure 2). A higher number of vertices can better approximate the shape of the sphere, but high geometry load also has a massively negative impact on the performance of the HoloLens. In our tests, a pentagon proved to be the best compromise between high geometry load and high overhead in the pixel shader stage.

A third way to obtain the primitives for raycasting is instancing, i.e. emitting each vertex multiple times. It is possible to draw an empty vertex buffer four times per atom and then compute the vertex position solely from the instance ID and a structured buffer view holding the parameters of each atom. Therefore, the data transferred between CPU and GPU is the same as for techniques reading the atom positions from a vertex buffer. The instancing-based approach needs to be aware of the fact that HoloLens uses instancing to minimise the overhead of rendering for the left and the right eye. Therefore, any number of instances needs to be doubled to render one time to the left and the other time to the right image.

Instancing is also the basis for one of our geometry-based approach: in this case, we prepare a vertex buffer for a unit-sized geodesic sphere comprising 128 triangles, which is scaled and moved by the parameters of each atom stored in a structured buffer view. This approach has the advantage to use only two shader stages, the vertex shader (VS) performing all geometry transformations and the pixel shader computing simple Blinn-Phong shading.

Finally, the tessellation stage can be used to generate the geometry for a sphere from the centre point of each atom. This approach is also used by the HoliMoli app (cf. Section 2). We use a hull shader requesting a tessellated quad which is conceptually “wrapped around” the sphere like cloth in the domain shader, which is also responsible for applying all transformations. The shader uses an adaptive tessellation factor computed in the hull shader using exactly the same method as in HoliMoli. The pixel shader is only performing shading computations in this approach. We have also implemented an optimised variant of this approach, which only renders a hemisphere that is oriented towards the viewer in the domain shader. This way, we can avoid generating and transforming the vertices, which would be dropped during back face culling anyway.

During normal operation, the user can place a molecule 2 m in front of the device and then start wandering around. However, in order to obtain consistent results during benchmarking, we eliminated all AR-related influences on the rendering speed by ignoring HoloLens’ tracking-based view transform. The result is the molecule always floating 2 m in front of the user regardless of how the HoloLens is oriented. Furthermore, we do not perform any clipping against the model of real-world objects such that all atoms of the molecule are drawn in any case.

3.2 Unity-Based Prototype

Microsoft provides the Mixed Reality Toolkit for Unity¹⁰ for the Unity game engine, which facilitates content creation for HoloLens and other immersive headsets (Microsoft summarises their AR and VR efforts under the Mixed Reality term). As Unity is mainly targeted at game development, it provides extensive scripting support. Using the natively supported C# interface, we implemented the routines to load molecular data into a Unity scene. In terms of rendering, our Unity prototype uses instancing. Unity provides a so-called Prefab asset type that acts as a template object. In our prototype, the Prefab asset is a triangle mesh representing a sphere with 136 faces. We assigned the built-in standard shader of Unity to render opaque spheres (see Figure 3). Secondary lighting effects like shadows or reflections were disabled. For each atom, an instance of the Prefab is created which is translated and scaled according to the atom’s parameters.

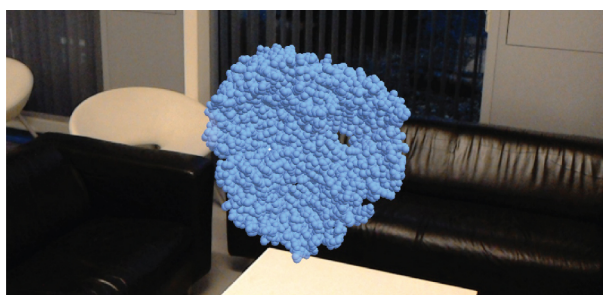


Figure 3: Live capture from the Unity prototype showing one of the protein data sets used for testing (PDB ID: 1AF6).

4 Results

Most of the following performance numbers were obtained using our UWP-based prototype, which gives us most flexibility in isolating influencing factors. We also compared the performance of the UWP-based prototype to the Unity-based one. We measured frame rates while moving the camera on five scripted paths: rotating the molecule around its y-axis and around its x-axis, respectively, while being either 2 or 1 m away from the user. Each of the rotations took 10 seconds. Furthermore, we moved the camera along the z-axis through the whole molecule starting again at 2 m distance. This camera path required 30 seconds to complete. All frame rates were measured by averaging over 500 ms periods. Following each of these periods, we additionally obtained statistics from the graphics pipeline comprising the geometry load and shader invocations of the last frame of the period. Table 1 shows the data sets (Figure 4) from the PDB we used for our tests. All data sets were scaled such that they fit a box of 0.5 m side length, i.e. the data set always covered approximately the same fraction of the image, but as the data set grows, the individual atoms get smaller. The last column in Table 1 shows how the holograms were scaled in comparison to the real size of the molecules.

Table 1: Data sets from the Protein Data Bank used for performance evaluation.

Data set	PDB ID	# of atoms	Scaling of bounding box
Insulin	1RWE	826	1:1.2E+08
Enterotoxin	1TII	5475	1:6.8E+07
Maltoporin	1AF6	10,052	1:5.9E+07
Plasmid coupling protein	1GKI	19,541	1:4.8E+08
<i>T. thermophilus</i> 30S ribosomal subunit	4KVB	50,952	1:2.2E+07

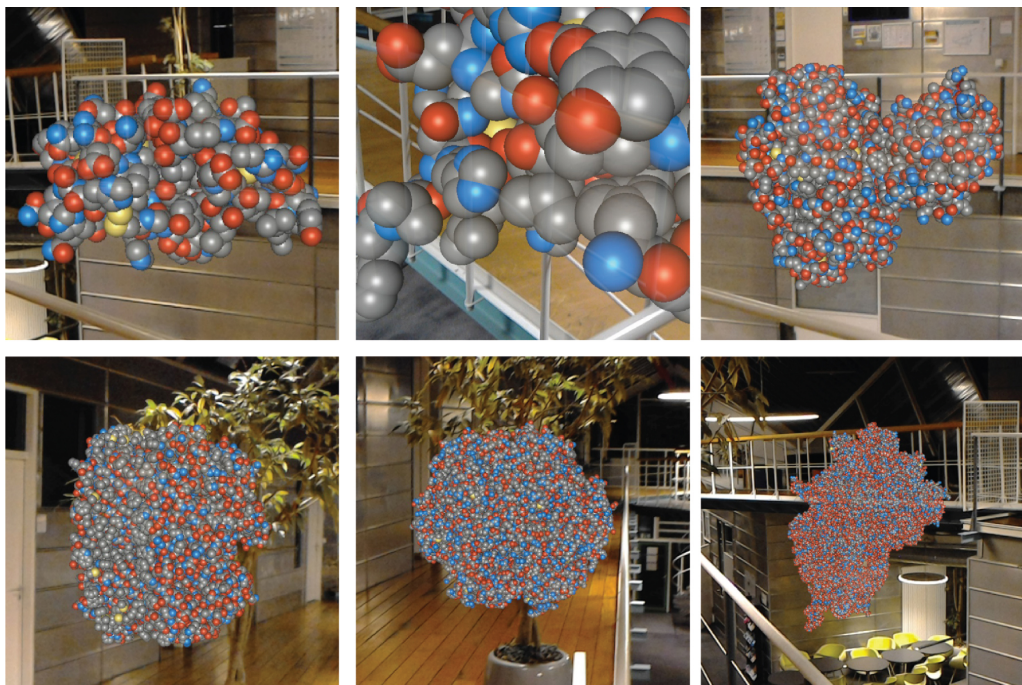


Figure 4: Live captures from our standalone UWP application rendering the data sets used in our benchmarks using raycasting on instances sprites. Upper row: 1RWE (826 atoms), close-up of 1RWE, 1TII (5475 atoms). Lower row: 1AF6 (10,052 atoms), 1GKI (19,541 atoms), 4KVB (50,952 atoms).

In the following, we show the frame rates for the rotation around the y-axis at 2 and 1 m. The numbers we obtained for the rotation around the x-axis are almost identical and show only slight variations for data sets which of the bounding box strongly deviates from a cube. In this case, the on-screen footprints differ in size, which causes noticeable differences in the performance.

As can be seen in Figure 5, only one technique reaches the target frame rate of 60 fps for the approximately 1000 atoms of the 1RWE data set: the instanced geodesic spheres. The tessellated full spheres and the hemispheres follow with an average of 35 fps and 33 fps respectively, while all three raycasting approaches reach only around 31 fps. We attribute this to the complex pixel shader for raycasting, which requires about 60 instructions and is invoked over 6 million times on average, whereas for the instancing and tessellation cases, we see only about 1.9 million invocations of the pixel shader (having only 23 instructions). Remember that 1RWE is relatively small, so the atoms are large, and in turn the number of pixels per sprite that are not part of the sphere (cf. Figure 2) is also large, which explains the large difference in pixels being filled.

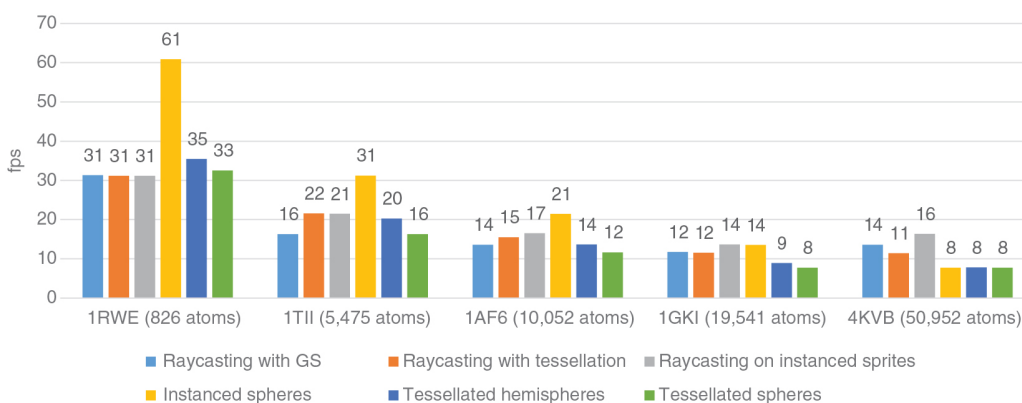


Figure 5: Average frame rates during a full rotation around the data set at 2 m distance.

Table 2 shows the approximate number of shader instructions for all techniques and stages. As the number of vertex shader invocations is solely dependent on the number of atoms (four vertices per atom for instanced sprites, 384 for each instanced sphere and one per atom for all other cases), the larger vertex shader for the instanced spheres has no negative impact for the small 1RWE data set. A special case are the tessellated hemispheres and spheres: these use an adaptive tessellation factor, i. e. the number of vertices per sphere increases as the camera gets closer (see also Figure 8). When rotating at 2 m distance, on average 423062 vertices are eventually generated for the hemispheres and 741,156 for the full spheres. As we move closer to the data set (Figure 6), the number rises to an average of 637,408 and 1,150,470 vertices for the both cases. Please note that the adaptive tessellation factors need to be rounded to an integer number, which causes the number of vertices for hemispheres being more than 50 % of the full sphere case. Additionally, we use “fractional odd” partitioning to achieve the same behaviour as HoliMoli, which generates slightly more excess vertices than “integer”, specifically for small tessellation factors. We also use the same algorithm as HoliMoli to compute the dynamic tessellation factors to achieve a faithful comparison. Note that the total number of triangles generated by this algorithm for the hemispheres is about the same as for the instanced spheres at a distance of 1 m. At 2 m, the number of triangles is much lower (~85 vs. 128). The visual quality, however, is much higher, since more triangles are distributed over the visible hemisphere compared to the instanced spheres.

Table 2: Approximate number of assembler instructions as reported by the HLSL compiler for the vertex shader (VS), geometry shader (GS), hull shader (HS), domain shader (DS) and pixel shader (PS) stage.

Rendering technique	VS	GS	HS	DS	PS
Raycasting with GS	6	127	–	–	60
Raycasting with tessellation	6	–	28	66	60
Raycasting on instanced sprites	77	–	–	–	60
Instanced spheres	21	–	–	–	23
Tessellated hemispheres	6	–	28	66	23
Tessellated spheres	6	–	27	35	23

A dash indicates that the shader stage is not used for the respective technique. Note that the tessellated hemispheres use a significantly more complex domain shader than the tessellated spheres, because additional code is required to orient the hemispheres towards the user, whereas the orientation of the spheres is irrelevant.

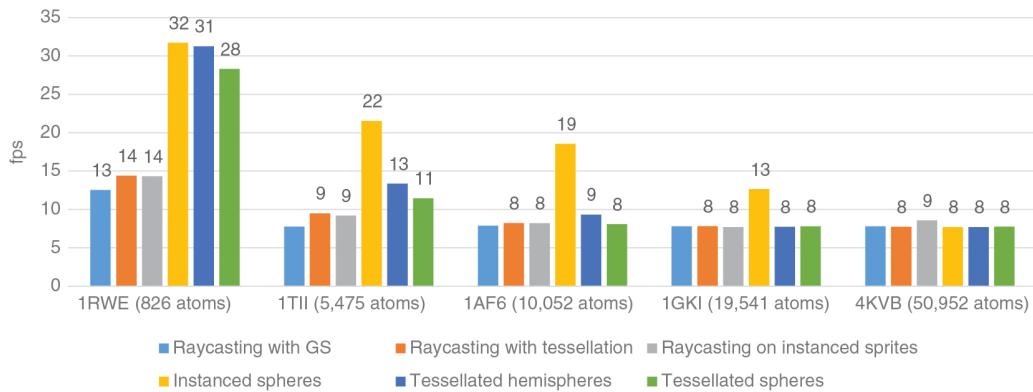


Figure 6: Average frame rates during a full rotation around the data set at 1 m distance.

Comparing Figure 5 and Figure 6 also reveals that the cost of rendering at close distances is rasterisation-bound: Although the vertex load is the same for the instanced spheres, the frame rate of the instanced spheres drops to 32 fps if a larger fraction of the screen needs to be filled. To examine this further, we turned off the per-pixel shading for this case and returned a constant colour instead, but the frame rate stayed at 32 fps. That is, the cost of the shading in the pixel shader is negligible. Consequently, only the fragment generation and the resulting invocation of the pixel shaders causes the drop in frame rate. These now need to process around 6 million pixels per frame, while for raycasting, almost 21 million need to be computed. However, due to the lower number of instructions per pixel, the geometry-based methods only drop to 30 fps while the number of pixels triples, whereas the raycasting-based ones drop by more than half and only reach 11–15 fps. This is backed by the observation that the raycasting reaches 60 fps when turning off the intersection test in the pixel shader and returning just a colour instead.

As the number of atoms increases, more differences between the techniques emerge. In case of 1TII, instanced spheres still perform best, but the distance of the raycasting-based techniques becomes smaller at 2 m

distance. We attribute this to two factors: first, as the number of atoms becomes larger, the performance impact on methods with large number of vertices per atom becomes also larger. This can also be observed with both dynamic tessellation techniques, which achieve a visual quality close to the raycasting techniques thanks to adaptive tessellation. Second, the previously mentioned scaling of the data set to the same bounding box decreases the screen-space footprint of each atom, which in turn reduces the relative total cost of the pixel shader. For instance, during the rotation at 2 m distance, we see an average of 3712 pixels per atom for 1RWE compared to 1166 for 1TII. At 1 m distance, having again larger per-atom footprints, instanced spheres clearly outperform all other techniques. A very similar behaviour as for the approximately 5000-atom 1TII data set can be observed for the almost double-sized 1AF6. The only significant difference is that the performance advantage of using the tessellated sprites over the ones from the geometry shader becomes smaller. This trend continues for 1GKI, in which case both techniques are already even, while for 4KVB, the geometry shader even performs better. Again, we attribute this to a combination of two factors: first, as the tessellation-shader technique uses a pentagon to avoid pixels without a hit during raycasting, the transformation load in the domain shader grows faster with the size of the data set than the one in the geometry shader. Second, as the footprints of the atoms become smaller, the benefit of avoiding misses during raycasting vanishes because the number of these problematic pixels per atom decreases. Therefore, the additional effort on the geometry side does not pay off any more for large data sets like 4KVB. Interestingly, the absolute frame rate we can achieve with raycasting is higher for the approximately 50,000 atoms of 4KVB than for the just below 20,000 atoms of 1GKI. Again, we think this is due to the smaller footprint of each atom: on average, 603 pixels for 1GKI and only 114 for 4KVB. The difference is so large that the average number of pixel shader invocations during the rotation at 2 m distance is lower for 4KVB (14,294,200) than for the much smaller 1GKI (23,574,700) – an effect that is not least due to the shape of the data sets: while 1GKI has a largely cubical bounding box, 4KVB has a relatively larger height, which causes the scaling to have a larger effect, because it is computed based on the longest edge of the bounding box (cf. Figure 4).

The effect of the image-space footprint of the data can be investigated when flying through the data set along one of the main axes. Figure 7 shows how the frame rate changes while flying through 1RWE. All raycasting-based methods behave very similarly in this test: as the camera gets closer to the data set, the frame rate first decreases with the increasing number of pixel shader invocations (Figure 8).

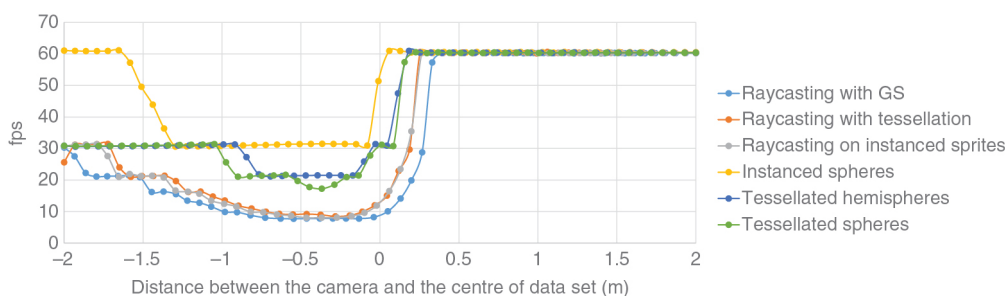
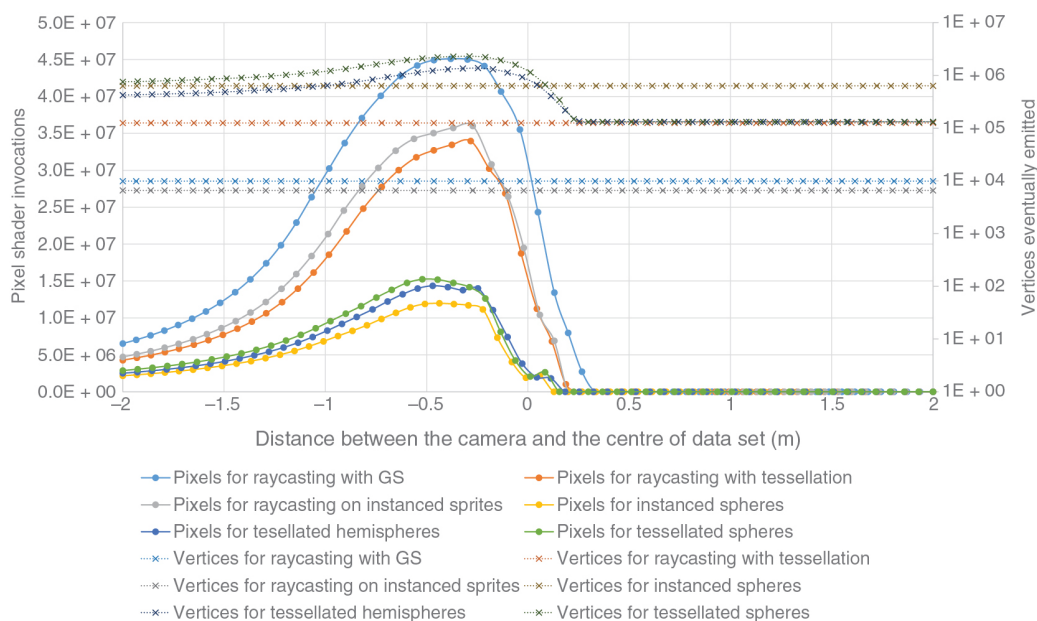


Figure 7: Frame rate over time while flying through 1RWE starting at 2 m distance and ending 2 m behind the data set.



Automatically generated rough PDF by ProofCheck from River Valley Technologies Ltd

Figure 8: Pipeline statistics collected while flying through the 1RWE data set along the z-axis. The solid lines denote the number of pixel shader invocations. The dotted ones designate the number of vertices emitted in the last stage producing geometry.

As the camera moves through the data, more and more atoms are behind the camera, which causes the number of pixel shader invocations to decrease again around the centre of the molecule. At some point around 0.2 m behind the centre, all of the atoms are behind the camera, such that all vertices still need to be transformed, but none of them results in pixel shader invocations, which are the limiting factor for this technique. In this situation, all raycasting-based methods reach 60 fps. As mentioned before, the geometry-based methods, namely using instancing and hemispheres, also suffer from large footprints due to the increasing rasterisation cost. The yellow line in Figure 7 shows that this happens abruptly after approximately half a metre on the path, although the number of pixel shader invocations increases smoothly (Figure 8). We therefore hypothesise, that we hit some limit in hardware parallelisation at this point. As the tessellated hemispheres (blue line) and spheres (green line) use an adaptive tessellation factor, more vertices are generated (Figure 8), which causes a drop in the frame rate at about 0.7 m before the centre of the data set. These techniques also reach 60 fps once the atoms are behind the camera, because this can be detected in the hull shader, i.e. no subdivision needs to be applied, which obviously drastically reduces the vertex load.

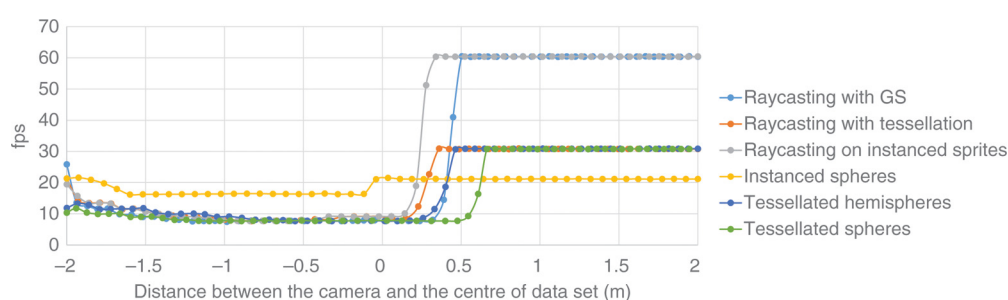


Figure 9: Frame rate over time while flying through 1AF6 starting at 2 m distance and ending 2 m behind the data set.

The diagram for the same fly-through using 1TII looks quite similar. As the data sets become larger, the constant cost of vertex transformations is already too high to achieve 60 fps even if nothing is visible. From the pool of the data sets we tested, this is the case for A1F6 and all larger ones. Figure 9 shows that all techniques that require more than one vertex per atom and/or need more than three shader stages cannot reach more than 30 fps even if nothing is visible.

In summary, we found that reaching the 60 fps target is possible for smaller data sets when using geometry-based rendering. However, this requires using coarsely tessellated spheres to keep the triangle count sufficiently low. The spheres we used in our tests are sufficiently tessellated for a good representation at 2 m distance. At a closer distance, the triangles become visible at the silhouettes, but the visual quality is still acceptable. Although the tessellation-based techniques did not reach 60 fps in the results presented above, this can be achieved by reducing the tessellation factors computed in hull shader. Raycasting renders only faster for the largest of our data sets, which is, however, too complex to reach acceptable frame rates with any technique on the HoloLens.

As described in Section 3.2, we also created a Unity prototype since this is the most popular way of developing apps for HoloLens. Our prototype uses Unity 2017.3.0f3 with the Microsoft Mixed Reality Toolkit for Unity 2017.2.1f1 and a custom sphere consisting of 136 triangles, which is close to the 128 triangles we used in our standalone prototype. The small difference of eight triangles per sphere is a result of the different ways of constructing a subdivision sphere in Unity and our prototype. As for the measurements above, we rotated the data set around the x-axis and the y-axis at a distance of 2 m. For both axes, the average frame rates obtained through HoloLens' device portal were similar. For 1TII, we reached 11.7 fps during the rotation and for 1AF6 6.2 fps. At a distance of only 1 m, performance significantly increases: during the rotation, the frame rate for 1TII was 19.9 fps, whereas for 1AF6, we measured 11.5 fps.

Table 3 shows a summary of all techniques we implemented in our UWP and Unity prototypes. While the performance of the smallest data set using the same rendering mode (instanced spheres) is comparable at 2 m distance, Unity can keep the frame rate when getting closer, whereas our prototype cannot. For the larger data sets, our standalone application reaches about thrice the frame rate at 2 m distance. At 1 m distance, the Unity implementation is between 7 % and 38 % slower than our hand-tailored application. We believe that this difference in performance cannot be explained by the above-mentioned slightly higher triangle count, but must be ascribed to additional computations performed by the Unity game engine. However, these computations apparently allow the engine to achieve better performance in cases where view frustum culling is possible, e.g. if the viewer is close to the molecule. In case of the of the smallest data set with 1000 atoms, this allows Unity to almost keep the frame rate. Nevertheless, with an increasing amount of objects, the advantage of this

optimisation diminishes. This is not least caused by the application case – viewing a molecule from outside – which causes most atoms being visible in most cases.

Table 3: Comparison of average frame rates for the rotation around the y-axis between the UWP and the Unity prototype.

Technique	1RWE		1TII		1AF6		1GKI		4KVB	
	2 m	1 m	2 m	1 m	2 m	1 m	2 m	1 m	2 m	1 m
Raycast. w/ GS	31.3	12.5	16.3	7.8	13.6	7.9	11.8	7.8	13.6	7.8
Raycast. w/ tessellation	31.1	14.4	21.6	9.5	15.5	8.2	11.6	7.8	11.4	7.7
Raycast. on inst. sprites	31.1	14.3	21.5	9.2	16.5	8.2	13.7	7.7	16.4	8.6
Instanced spheres	60.8	31.7	31.2	21.5	21.4	18.6	13.5	12.6	7.8	7.7
Tess. hemispheres	35.5	31.3	20.3	13.4	13.6	9.3	8.9	7.8	7.8	7.7
Tess. spheres	32.5	28.3	16.3	11.5	11.6	8.1	7.7	7.8	7.8	7.8
Unity	55.6	52.6	11.7	19.9	6.2	11.5	3.0	6.7	3.0	4.4

5 Discussion

As mentioned in the introduction, our current prototype is only the first step on the way to a comprehensive immersive analytics environment for molecular data. The ability of HoloLens to share scenes between devices is not only useful for collaborative data analysis, but also for educational scenarios. For instance, as all users are able to walk independently around the virtual molecule, they can choose their own view without influencing those of others – in contrast to a scenario of multiple users in front of a (large) screen where only one user can control the camera. The AR approach still allows users to point out locations of interesting aspects of the data to each other in a natural way. Consequently, it offers all benefits of physical navigation, which has been shown to be advantageous for comparison and search tasks in two-dimensional data like maps on large displays [25]. However, results of our previous experiments in this direction [26] suggest that the effect is less pronounced when working with three-dimensional data on a large display, which still requires virtual navigation to investigate from all sides. We believe that this is due to the fact that three-dimensional content also requires three-dimensional navigation, which is possible in the AR scenario.

Another inherent benefit common to AR and VR when working with three-dimensional data are the stereoscopic rendering capabilities. Stereoscopic rendering is routinely used for molecular visualisation to make the highly complex structures of molecules comprehensible. Therefore, many available popular molecular viewers like VMD or PyMol offer support for traditional stereoscopic output devices like 3D displays, powerwalls or VR headsets. The latter have the same properties as AR with respect to stereo rendering and three-dimensional physical navigation and would lend themselves also to create a collaborative immersive analytics environment. However, we see several advantages of AR over VR in our context: obviously, a see-through AR HMD like HoloLens leads to minimal detachment of the user from the environment. Consequently, natural communication among collaborators is possible, which could only be approximated by the use of avatars in VR. The connection to the real world makes it possible to interact with physical objects like models of proteins and augment them with visualisation, as proposed by Gillet et al. [18], who rendered the electrostatic field around a 3D-printed protein (cf. Section 2). During development, we noticed that HoloLens is less obtrusive than we initially expected. It is not only possible to use the device for a longer period of time, but also to perform short non-AR tasks while wearing it (taking notes on paper, reading from screen or textbook and even making code changes while testing our prototype). This is naturally not possible with a VR headset. One benefit of a VR environment would be the larger field of view offered by current devices. We therefore think that VR might be advantageous in a remote collaboration scenario where each user is in a separate location until the advent of improved AR headsets.

Besides the benefits of stereoscopic rendering, which makes complex three-dimensional molecular structures easier to grasp, and the natural physical navigation, an immersive analytics environment can offer an engaging way of viewing and interacting with molecular visualisation. This advantage must not be underestimated, especially for educational scenarios. An example for such an educational AR application from the field of medicine is *Insight Heart* by AnimaRes¹¹, which teaches the user about common conditions of the heart. We envision that a collaborative AR application could also be used in a classroom. Students could investigate the same molecular data sets together, which fosters discussion. In this scenario, the natural communication between students and teachers while working in the AR environment would be especially useful. As mentioned above, it is also still possible to take notes or read from a whiteboard while wearing HoloLens, which is often

necessary in a classroom. Since a smooth user experience is an important factor for an engaging experience, fast rendering should be ensured for educational applications.

As stated in the introduction, we expected complex rendering methods like GPU-based raycasting to perform poorly on HoloLens given the performance guidelines for the device. Specifically, the method requires significantly more instructions per pixel than the recommended maximum, the compact structure of the molecules usually causes much overdraw and makes frustum culling superfluous since all atoms are typically within the frustum, and the raycasting requires the unfavourable high-precision depth buffer. For the smaller data sets, this assumption proved to be correct, as raycasting reaches only 31 fps for the smallest data set, which renders at 60 fps using instanced geometry. If the data sets significantly exceed 1000 atoms, none of the techniques are able to reach the 60 fps target. Since we cannot expect that further micro-optimisation of the rendering will lead to a significantly increased frame rate, large data sets need to be rendered based on drastically simplified representations. Replacing the spheres with pre-lit, flat imposters reduces the per-atom cost compared to the techniques we used. Another simplification could be to pre-compute clusters of spheres and render only one representative for the cluster similar to the work of Parulek et al. [27]. Alternatively, one could resort to using other representations like low-polygon molecular surfaces or ribbon models. For larger data sets, these become easier to render than the spheres. An unsystematic black-box-test of the HoloLens app using the frame rates reported by the HoloLens device portal showed that no technique was able to reach the 60 fps goal for our test data sets, but for the large 4KVB, the supposedly more complex cartoon representation was even faster than ball-and-stick (19 fps in contrast to 11 fps at an estimated distance of 3 m). When targeting HoloLens, using spatial structures on the CPU to reduce the overdraw of raycasting-based techniques, which performed better for large data sets, could also be a way to extend the size of data sets the hardware can handle. Finding the optimal strategy requires, however, further empirical investigation. The same is true for different and more powerful hardware like the DAQRI Smart Glasses¹², which sport an Intel Core m7 CPU with Intel HD Graphics 515. Solutions like Meta²¹³, which are tethered to a desktop PC can, of course, leverage its full processing power. Since this restricts the physical navigation, especially when multiple users are present, we rate this as unsuitable for collaborative immersive analytics. The increasing number of different VR and AR systems does not only promise more capable hardware, but confronts developers with a variety of different native SDKs. Adapting software to different environments for reaching the best possible performance on every platform naturally increases the cost of development and might deter users from trying another platform. Therefore, developing reusable rendering strategies for Unity, which has become the common denominator of all recent VR and AR platforms, that adapt the engine to the requirements of bioinformatics applications might be a way to quickly leverage new hardware in the field.

6 Summary and Future Work

In this paper, we have presented a prototypical AR molecular visualisation application for HoloLens. We implemented several different rendering methods for the space-filling model (sphere rendering) and compared their performance to identify the technology that works best on the rather limited HoloLens hardware. As input, we used publicly available protein data sets obtained from the RSCB PDB [11], hence, our results are reproducible and meaningful for real-world data. In our tests, we found that different methods give maximum performance, depending on data set size and image footprint of the final rendering. We further found that GPU-based raycasting, which is currently the fastest method on desktop hardware, is only advisable for very large data sets. For data sets below 50,000 atoms, triangulated spheres rendered via instancing give much better performance. This is due to the high number of shader instructions needed for raycasting, which is not a good match for the GPU used by HoloLens. It is also noteworthy that only the smallest of our test data sets with 1000 atoms reached the target frame rate of 60 fps. Consequently, we recommend the use of level-of-detail techniques or reducing the visual fidelity of the representations for larger data sets to ensure a smooth, judder-free user experience, which is important for AR.

In the future, we want to extend our prototype to a collaborative immersive analytics application for molecular structures and other three-dimensional data from different areas of bioinformatics. Therefore, we also discussed different application scenarios for such a system and examined the technical feasibility using the available hardware. Besides implementing additional representations like molecular surfaces such that they can be rendered interactively on HoloLens, we also want to complement the current speech-based user interface with a graphical user interface (GUI). While voice input is recommended by Microsoft as one of the three key forms of input on HoloLens¹⁴, it could become infeasible in a collaborative environment with multiple people talking and giving voice commands at the same time. This GUI should also support typical collaborative features like placing text labels for annotations, which can either be private or shared with other users. We

also want to investigate the low performance of the Unity prototype further and find ways to speed it up as well, e.g. by employing level-of-detail rendering similar to the one used by Le Muzic et al. [28] in their desktop Unity-based tool cellVIEW.

Acknowledgement

This work was partially funded by German Research Foundation (DFG) as part of Collaborative Research Centres SFB 716 (projects D3 and D4), SFB Transregio 161 (projects A02 and INF), and SFB 1244 (project B04).

Conflict of Interest Statement: Authors state no conflict of interest. All authors have read the journal's publication ethics and publication malpractice statement available at the journal's website and hereby confirm that they comply with all its parts applicable to the present scientific work.

Notes

- 1 <https://www.oculus.com/rift/> (online, last accessed 21/12/2017)
- 2 <https://www.microsoft.com/hololens> (online, last accessed 21/12/2017)
- 3 https://developer.microsoft.com/en-us/windows/mixed-reality/performance_recommendations_for_hololens_apps (online, last accessed 21/12/2017)
- 4 <http://www.unity3d.com/> (online, last accessed 21/12/2017)
- 5 <https://www.vive.com/> (online, last accessed 10/01/2018)
- 6 <https://www.leapmotion.com/> (online, last accessed 10/01/2018)
- 7 <https://www.epam.com/ideas/videos/vr-ar-molecular-visualization-apps> (online, last accessed 10/01/2018)
- 8 <https://github.com/arminms/HoliMoli> (online, last accessed 10/01/2018)
- 9 <https://www.microsoft.com/en-us/store/p/holecule/9nblggh513z0> (online, last accessed 18/04/2018)
- 10 <https://github.com/Microsoft/MixedRealityToolkit-Unity> (online, last accessed 09/01/2018)
- 11 <https://animares.com/> (online, last accessed 10/01/2018)
- 12 <https://daqri.com> (online, last accessed 10/01/2018)
- 13 <https://www.metavision.com> (online, last accessed 10/01/2018)
- 14 https://developer.microsoft.com/en-us/windows/mixed-reality/voice_input (online, last accessed 10/01/2018)

References

- [1] Humphrey W, Dalke A, Schulten K. VMD: visual molecular dynamics. *J Mol Graph.* 1996;14:33–8.
- [2] Schrödinger L. The PyMOL molecular graphics system, Version 1.8; 2016.
- [3] Ni T, Schmidt GS, Staadt OG, Livingston MA, Ball R, May R. A survey of large high-resolution display technologies, techniques, and applications. In: *IEEE Virtual Reality Conference*; 2006. p. 223–36.
- [4] Cruz-Neira C, Sandin DJ, DeFanti TA, Kenyon RV, Hart JC. The CAVE: audio visual experience automatic virtual environment. *Commun ACM.* 1992;35:64–72.
- [5] Gumhold S. Splatting illuminated ellipsoids with depth correction. In: *Vision, modeling, and visualization*; 2003. p. 245–52.
- [6] Klein T, Ertl T. Illustrating magnetic field lines using a discrete particle model. In: *Vision, modeling, and visualization*; 2004. p. 387–94.
- [7] Falk M, Grottel S, Krone M, et al. Interactive GPU-based Visualization of Large Dynamic Particle Data. San Rafael: Morgan & Claypool Publishers, 2016.
- [8] Reina G, Ertl T. Hardware-accelerated glyphs for mono- and dipoles in molecular dynamics visualization. In: *EG/IEEE VGTC symposium on visualization*; 2005. p. 177–82.
- [9] Grottel S, Reina G, Ertl T. Optimized data transfer for time-dependent, GPU-based glyphs. In: *IEEE pacific visualization symposium*; 2009. p. 65–72.
- [10] Grottel S, Krone M, Müller C, Reina G, Ertl T. MegaMol – a prototyping framework for particle-based visualization. *IEEE Trans Vis Comput Graph.* 2015;21:201–14.
- [11] Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, et al. The protein data bank. *Nucleic Acids Res.* 2000;28:235–42. Available from: <http://www.rcsb.org>.
- [12] Chandler T, Cordeil M, Czauderna T, Dwyer T, Glowacki J, Goncu C, et al. Immersive analytics. In: *2015 Big Data Visual Analytics (BDVA)*; 2015. p. 1–8.
- [13] Kozlíková B, Krone M, Falk M, Lindow N, Baaden M, Baum D, et al. Visualization of biomolecular structures: State of the art revisited. *Comput Graph Forum.* 2017;36:178–204.
- [14] Norrby M, Grebner C, Eriksson J, Boström J. Molecular rift: virtual reality for drug designers. *J Chem Inf Model.* 2015;55:2475–84.
- [15] Salvadori A, Brogni A, Mancini G, Barone V. Moka: designing a simple scene graph library for cluster-based virtual reality systems. In: *Augmented and virtual reality*. Cham: Springer; 2014. p. 333–50.

- [16] Salvadori A, Del Frate G, Pagliai M, Mancini G, Barone V. Immersive virtual reality in computational chemistry: applications to the analysis of QM and MM data. *Int J Quantum Chem.* 2016;116:1731–46.
- [17] Grebner C, Norrby M, Enström J, Nilsson I, Hogner A, Henriksson J, et al. 3D-Lab: a collaborative web-based platform for molecular modeling. *Future Med Chem.* 2016;8:1739–52.
- [18] Gillet A, Sanner M, Stoffer D, Goodsell D, Olson A. Augmented reality with tangible auto-fabricated models for molecular biology applications. In: *Proceedings of the IEEE conference on visualization*; 2004. p. 235–42.
- [19] Berry C, Board J. A protein in the palm of your hand through augmented reality. *Biochem Mol Biol Educ.* 2014;42:446–9.
- [20] Zheng M, Waller MP. ChemPreview: an augmented reality-based molecular interface. *J Mol Graph Model.* 2017;73:18–23.
- [21] Hoffman M, Provance J. Visualization of molecular structures using HoloLens-based augmented reality. *AMIA Jt Summits Transl Sci Proc.* 2017;2017:68–74.
- [22] Voinea A, Moldoveanu A, Moldoveanu F. Bringing the augmented reality benefits to biomechanics study. In: *Proceedings of the 2016 workshop on multimodal virtual and augmented reality*; 2016. p. 9:1–6.
- [23] Joachimczak M, Liu J, Ando H. Real-time mixed-reality telepresence via 3D reconstruction with HoloLens and commodity depth sensors. In: *Proceedings of the ACM international conference on multimodal interaction*; 2017. p. 514–5.
- [24] Chen Z, Hu W, Wang J, Zhao S, Amos B, Wu G, et al. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In: *Proceedings of the ACM/IEEE symposium on edge computing*; 2017. p. 14:1–14.
- [25] Ball R, North C, Bowman D. Move to improve: promoting physical navigation to increase user performance with large displays. In: *Proceedings of the SIGCHI conference on human factors in computing systems*; 2007. p. 191–200.
- [26] Müller C, Krone M, Scharnowski K, Reina G, Ertl T. On the utility of large high-resolution displays for comparative scientific visualisation. In: *International symposium on visual information communication and interaction.* vol. 8; 2015. p. 131–6.
- [27] Parulek J, Jönsson D, Ropinski T, Bruckner S, Ynnerman A, Viola I. Continuous levels-of-detail and visual abstraction for seamless molecular visualization. *Comput Graph Forum.* 2014;33:276–87.
- [28] Le Muzic M, Autin L, Parulek J, Viola I. cellVIEW: a tool for illustrative and multi-scale rendering of large biomolecular datasets. In: *EG workshop on visual computing for biology and medicine*; 2015. p. 61–70.