Visualization Multi-Pipeline for Communicating Biology

Peter Mindek, David Kouřil, Johannes Sorger, Daniel Toloudis, Blair Lyons,
Graham Johnson, M. Eduard Gröller, and Ivan Viola

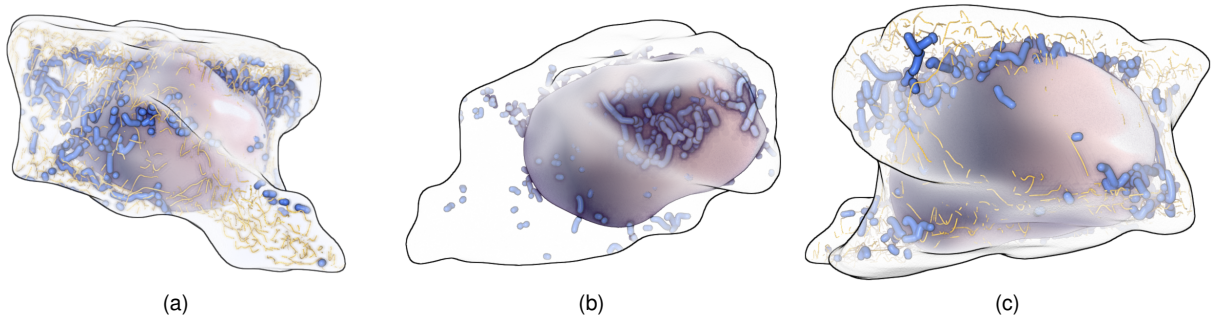


Fig. 1: Several datasets integrated into a single visualization of a human induced pluripotent stem cell. Three renderings with different visualization parameters are shown.

Abstract—We propose a system to facilitate biology communication by developing a pipeline to support the instructional visualization of heterogeneous biological data on heterogeneous user-devices. Discoveries and concepts in biology are typically summarized with illustrations assembled manually from the interpretation and application of heterogenous data. The creation of such illustrations is time consuming, which makes it incompatible with frequent updates to the measured data as new discoveries are made. Illustrations are typically non-interactive, and when an illustration is updated, it still has to reach the user. Our system is designed to overcome these three obstacles. It supports the integration of heterogeneous datasets, reflecting the knowledge that is gained from different data sources in biology. After pre-processing the datasets, the system transforms them into visual representations as inspired by scientific illustrations. As opposed to traditional scientific illustration these representations are generated in real-time - they are interactive. The code generating the visualizations can be embedded in various software environments. To demonstrate this, we implemented both a desktop application and a remote-rendering server in which the pipeline is embedded. The remote-rendering server supports multi-threaded rendering and it is able to handle multiple users simultaneously. This scalability to different hardware environments, including multi-GPU setups, makes our system useful for efficient public dissemination of biological discoveries.

Index Terms—Biological visualization, remote rendering, public dissemination

1 Introduction

Biologists generate large amounts of data every day. Through analysis of these data, scientists make new discoveries, which are further communicated to a broad spectrum of audiences, from peer-experts, through students in training, up to the general public. Especially for the latter two, this is commonly done through illustrations designed to reduce distractions, focus attention, and clarify content, since 3D image data is difficult for non-biologists to interpret.

In order to create an illustration conveying a novel discovery, several steps are necessary. First, the discovery, as well as the underlying data have to be explained to the illustrators. Subsequently, the illustrators proceed to create an illustration, which can be either in the form of an image or an animation. Finally, labels and descriptions are added so that the illustration conveys the desired information, and the illustration is distributed to the target audience.

There are several problems with this approach. In cell biology, the process of creating illustrations can take much longer than the average time in which new discoveries are made. Therefore, after the illustration is made public, it might already be obsolete. The only way to update it is to remake it from scratch. Therefore, this model is not suitable for public dissemination of state-of-the-art knowledge in biology.

Another problem is that the illustrations and animations are typically not interactive. They cannot support the learning process as efficiently as an interactive environment could. They can tell a single story, but the possibilities to adapt them to tell different stories are very limited.

We propose a novel system, called *Marion*, which addresses both of these problems. It offers means for creating a pipeline which connects the underlying measurement and imaging data directly with the target audience. The system includes tools for illustrators to design interactive visualizations created automatically from the underlying data, and to deliver the results directly to users over the internet. Whenever the data change, or new findings are made, the illustrations can simply be updated and the results are visible to everyone immediately. The system can either allow users to freely interact with the illustration, or it can take them on guided tours through cells or microorganisms.

We designed the system to fulfill the need for public dissemination of discoveries in cell science. The goal of the system is to informatively display human cells, in a similar way to visualizations designed by illustrators. The renderings are required to be interactive and deliverable over the internet, so that they can be used for communicating various scientific discoveries to wide audiences. The renderings have to be directly derived from the underlying data, so that the pipeline does not have to change when the data are updated.

In the first step, the system loads and filters the data. The system has to support various data types, such as meshes, volumes, as well

- *P. Mindek, D. Kouřil, J. Sorger, I. Viola are with TU Wien. M. E. Gröller is with TU Wien and the VRVis Research Center.*
  E-mail: {mindek | dvdkouril | sorger | meister | viola}@cg.tuwien.ac.at.
- *D. Toloudis, B. Lyons, G. Johnson are with Allen Institute for Cell Science.*
  E-mail: {danielt | blairl}@alleninstitute.org and graham@grahamj.com.

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

as curves describing tubular structures, since individual organelles are described with different data types. Subsequently, each of the data types has to be rendered in a unique way, while maintaining interactive framerates. For instance, the tubular structures are displayed as sets of screen-space metaballs, which can be rendered in real-time while providing an organic look of the organelles they are representing, such as mitochondria. Finally, the rendered images of the organelles, which are to be visualized, are composited by means of an order-independent transparency technique.

In our system, this pipeline is implemented as a stand-alone dynamically-linked library, which can be reused in different host applications without any changes. To demonstrate this, we implemented a desktop application with a simple user interface displaying the cell data and supporting various story-telling operations, as well as a remote-rendering server, which delivers the renderings over the internet to a web-based client. Our system ensures that both GPU shader code, as well as native host code is reloaded automatically upon change. This allows the developers to quickly prototype new rendering methods, as well as maintain the code without the need of restarting the application and reloading all the data.

Our system solves an existing problem of the *Allen Institute for Cell Science (AICS)*. The goal of AICS is to engage the public in cell biology with informative 3D illustrations originating from microscopy data of human cells. The data are frequently updated, e.g., new organelles are scanned, or higher resolution scans are produced. Our system ensures that the changes made to the data are immediately incorporated in the produced visualizations. Our remote-rendering solution ensures that these interactive visualizations are delivered to the users with the only requirement that they have an internet connection on their devices.

Our remote-rendering solution serves as a reference implementation of a public dissemination mechanism for biological data. It is scalable to utilize multiple graphics processing units through the use of a stateless multi-threaded rendering architecture. This ensures that multiple clients can simultaneously issue rendering requests to the server. The requests are distributed amongst the active rendering threads based on a load-balancing mechanism tailored to the rendering pipeline, automatically adapting to its possible changes. The load-balancing works independently from the actual visualization pipeline, and it is therefore easily reusable for different applications.

The contributions of this paper are:

- A novel system for fast prototyping and deployment of illustrative visualizations.
- A pipeline implemented using our system designed for public dissemination of cell biology.
- A scalable remote-rendering system for the delivery of rendered images over the internet, implemented as another application of our system.

2 Related Work

Visualization of biological structures and processes is a major part of visualization research. It has been shown that students understand concepts easily and in-depth if the material is supplemented by realistic renderings [11]. Iwasa [6] argues for the importance of animated content for science communication and teaching. As a consequence there has been a significant effort put into creating scientific illustrations and animations. Sharpe et al. [27] provide a comprehensive overview of techniques for creating biology-based models using modern 3D software like *Maya*. Arguably, interactive visualizations can convey the information and help with the learning process even more than a static image or a noninteractive animation.

2.1 Biological visualization systems

Rendering large biological structures at their atomic level on consumer level hardware has only been possible in the last few years. Initially, Lindow [16] rendered large scenes containing microtubules made of several billion atoms. Since then, new rendering techniques have been developed to improve the performance and appearance of rendering large and complex molecular scenes [15].

MegaMol [5] provides a framework that serves as a prototyping tool for creating visualizations of large, particle-based datasets. Using a packing tool like cellPACK [7], is is possible to create models of HIV down to their atomic detail [8]. The actual interactive visualization of cellPACK outputs have been made possible by cellVIEW [14].

Different approaches have to be taken when acquiring and displaying larger organisms. Our system uses microscopy image data to render organelles of a human cell, which cannot be feasibly modelled and rendered in its entirety on the atomic level of detail because of hardware limitations.

Walter et al. [33] reviews the many different modern acquisition techniques which scientists are using to generate enormous amounts of data these days. Moreover, the authors introduce a plethora of tools for visualizing multidimensional image data.

VTK [26] is a prevalent framework for computer graphics, used extensively in visualization. Multiple other popular tools for showing multidimensional images exist [3, 19, 20, 22–24, 30].

UCSF Chimera [19] is a popular viewer for displaying and manipulating atom-resolution macromolecule structures.

ParaView [1] is an open-source solution initially focused on visualizing simulation results. It uses VTK as its foundation layer and implements methods to process, analyze, and visualize large datasets.

Web-based tools play an important role, as it is the most widespread form of media used by general audience. Online Anatomical Human [29] is a web based project which provides an interactive way to study human anatomy. With our system, we want to enable interactive exploration of human cells, based on various microscopy and simulation datasets.

2.2 Remote rendering systems

The idea of remote rendering is not new and has been used in several applications. Martin [18] establishes three basic approaches to rendering in a server-client environment: client-side methods, server-side methods, and hybrid methods. The server-side methods are the focus of this paper and are based on rendering all the content on a high performance server and delivering the results as images to the clients.

The most obvious reason to use remote rendering is to leverage accelerated graphics capabilities of a powerful server machine and deliver the high-quality rendering results to the client, which might be a less powerful machine, such as a tablet or a mobile phone. This has been done for example in case of intensive volume rendering [4]. In this work, the server renders into an off-screen buffer, compresses the result and transfers it to the client. The client on the other hand notifies the server about user interactions.

Koller et al. [10] use remote rendering as a way of securing the intellectual property of scanned high-fidelity models. The authors provide a downloadable client program by which users can view 3D models of Michelangelo's statues. The difference to our system is that the client is implemented as a web page, thus eliminating the barrier of downloading and installing the client software itself.

Ma et al. [17] explore remote rendering as a way to solve the problem of both high performance and storage requirements. The issue of latency is discussed and several image compression formats are compared in order to minimize the lag observed by the user. JPEG is chosen as the most suitable compression format, therefore we also use JPEG as the format of resulting images.

Krone et al. [12] used a server-client architecture to leverage the performance of a server in order to deliver high-quality visualization results. The main use case of their solution is to employ mobile devices to control a large high-resolution display. The main emphasis does not lie in the client visualization quality. Our approach instead aims to provide the best possible final visualization on the client side.

Lamberti et al. [13] propose an architecture for a remote visualization system based on the Chromium software and test the performance of such a system on two model use cases: surface (mesh) rendering and volume rendering.

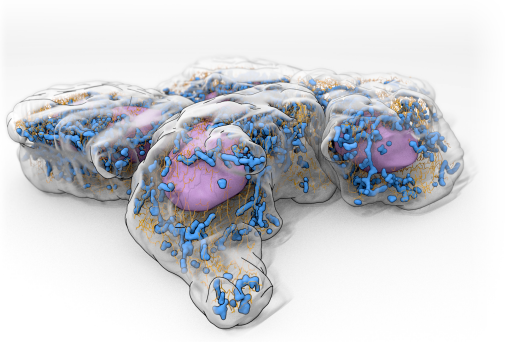


Fig. 2: An illustration of a few human induced pluripotent stem cells. It was created manually in 3D modelling software, based on the measured data. The rendering took several minutes. Illustration by Graham Johnson.

Jomier et al. [9] demonstrate the usage of ParaViewWeb, a JavaScript library that uses the processing and rendering power of either VTK or ParaView on the back-end to deliver visualizations in the browser.

3 System Overview: Visualization Multi-Pipeline

The motivation for this work is the task to make human-cell confocal-microscopy data accessible to the general public. The raw data, in the form of intensity volumes, do not convey much on their own to laypersons. Their relatively low resolution, and multiple channels encoding individual organelles, make the data unsuitable to standard volume-rendering approaches. Therefore, illustrators manually create abstracted three dimensional depictions of the cells based on the observed data, such as the one shown in Figure 2. Here, the scanned volumes are simplified and discrete surfaces are introduced to give the viewers an idea about the shapes and spatial configuration of the cell. Such information would be very difficult to grasp from simply visually exploring the volume-rendered images of the original data.

The manually created illustrations can explain the data very efficiently. However, there are several problems with this approach.

P1: the underlying data are continuously refined, as better scanning techniques are introduced relatively frequently. The algorithms used for extracting surfaces from the volume data are also continuously improved. With each update to the data, the illustration has to be manually updated as well. This takes a significant amount of time for the illustrator, which makes it expensive to produce the updates to the illustration. After the creation of the illustration, it is also necessary to deliver it to the target audience through a variety of channels. This results in the situation that by the time the illustration reaches the target audience, it might be already outdated.

P2: with each scanned cell, a new illustration has to be created. Biologists also use shape-modeling tools to examine all possible shapes of the cells by generating synthetic data. To visualize all these synthetic cells, an automated approach of creating the illustrations is necessary.

P3: the manually created illustrations do not provide any kind of interactivity. All the rendering parameters are fixed and tailored towards a single use-case. It is difficult to use such illustrations as a presentation environment for novel discoveries, such as protein interactions. The illustrations created in this way are mostly single-purpose.

Our goal is to address all of these problems with a system, which connects the underlying data directly with the target audience by an automated pipeline creating interactive, real-time 3D visualizations. The system should allow for easy creation of thick pipelines which simulate workflows of illustrators producing multiple different images from the underlying biological data. This means the individual datasets are processed into various types of data, rendered with different visualization techniques, composited together into an illustration communicating certain message, and delivered to the target audience. We refer to such pipelines as *visualization multi-pipelines*.

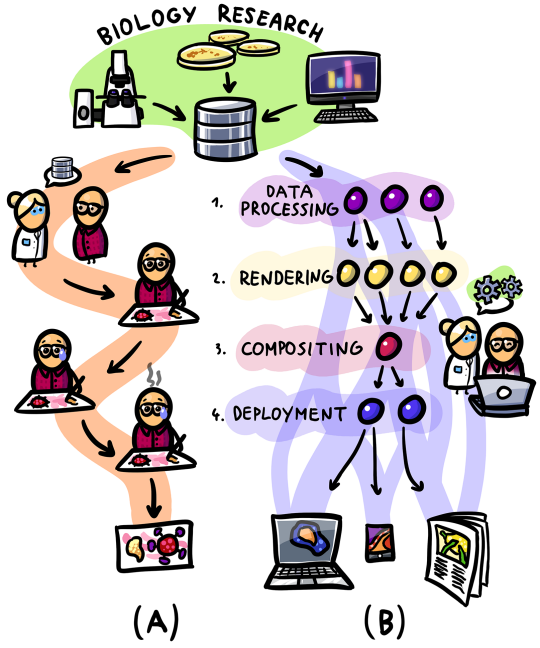


Fig. 3: (a) Traditional scientific illustration. It is necessary to transfer the knowledge to the illustrator, who then, through a thin pipeline, produces a single illustration. Any new illustration, e.g., reflecting changes in the underlying data, has to be created from scratch. (b) Visualization multi-pipeline. Our system allows the illustrators to design thick pipelines, which can automatically convert the underlying data to the desired visualizations in multiple ways, producing illustrations conveying different aspects of the data. All data refinements are instantaneously reflected in all generated visualizations.

3.1 Visualization Multi-Pipeline

The visualization multi-pipeline extends what is commonly known as *visualization pipeline* by the final stage which we refer to as *deployment stage*. In this stage, the entire process of generating the illustrations is embedded into various host applications, which ensures the delivery of the illustrations to the target audience. This step is essential when the illustrations are intended for public dissemination purposes. The support for the deployment stage constitutes the central novelty of our system. In comparison to the existing visualization systems, our system is specifically designed so that the visualizations developed within it can be easily embedded into external applications. Thus, supporting the deployment stage of the visualization multi-pipeline.

Figure 3 shows the workflow of traditional illustrations (orange path) as opposed to our proposed system (blue path).

Both cases start with biology researchers gathering data through observations, measurements, and simulations (green). In a traditional, manual approach, the process continues with a knowledge-transfer, where the scientists carefully explain the underlying dataset to the illustrators. The illustrators then process the dataset in various ways, and work on abstracted depictions of the structures and processes, which need to be communicated. This requires a significant effort, and in general the result can only convey information about a single dataset - the one that has been explained to the illustrator by the scientists.

When our system is used, there are three user-roles involved - a domain expert (a biologist), an illustrator, and a programmer. Rather than explaining a single dataset that should be illustrated, the domain expert explains to the illustrator the biology processes in general. The illustrator assesses which visualization and rendering techniques will be required in order to illustrate any dataset that is supposed to convey these processes. Subsequently, the illustrator communicates with the programmer, who prepares the skeleton of the visualization multi-

pipeline. The API of the system is designed in such a way that common operations, which illustrators would typically be doing, can be implemented as single-command statements. With this approach, the programmer can efficiently implement any illustrator's requests.

After the programmer translates the illustration process into the multi-pipeline, the illustrator can customize it by changing the parameter settings. Our system automatically exposes all the parameters through a user interface so that the illustrator can influence the rendering without looking into the code. Thanks to the deployment stage, all the changes made to the multi-pipeline and its parameters are automatically reflected in all applications build for this particular multi-pipeline. This makes our system suitable for public dissemination of biology, where it is necessary to illustrate large numbers of datasets on different platforms.

In contrast to the traditional approach of illustration, as well as existing visualization systems, Marion is able to produce interactive illustrations, which can be immediately deployed to different devices. The interactiveness of the generated illustrations ensures that they can be used as a platform to tell stories about cell biology, as well as to communicate new discoveries made in this field to the public, all based on the same underlying data. Alternatively, the generated illustrations can be used for inter-disciplinary communication, for instance as figures in papers, while they can be highly customized to highlight desired features of the data. Whenever the underlying data change, all the generated visualizations can be updated immediately, which is a significant advantage over the traditional approach.

The first stage, *Data Processing*, ensures that the data are loaded, and adequately filtered and simplified so that they are ready to be visualized. Here it is important that the complexity of the data is reduced as much as possible, while maintaining all the necessary structural information about the data. In general, multiple datasets are processed in this stage, and each of them can be converted into a different format or data type, since this might be necessary for applying different illustrative techniques.

In the second stage, the data are *rendered* using various illustrative-visualization techniques. Different techniques are needed, since the underlying data are of various data types, and it might be required to convey multiple aspects of each of them. For instance, the mitochondria might be displayed with volume rendering as probability clouds to show the original measured data, or they can be displayed as extracted surfaces, which are not conveying the measured data, but give a simplified idea to the laypersons what mitochondria *might* look like.

The third stage produces the final image by compositing all the renderings into a single visualization. Here various visualization parameters are applied to achieve desired results.

Most notably, the visualization multi-pipeline extends the traditional visualization pipeline by the last, fourth stage, which we refer to as *deployment*. This stage describes the application environments, into which the previous stages can be integrated. Marion aids this integration on the prototyping, development, as well as deployment level. Thanks to the inclusion of this stage in our system, it is easily possible to create applications delivering visualizations in the form of interactive content to large audiences.

The colored circles in each stage depicted in Figure 3 represent the modules of our system. Marion is built in a way that it is easy to add new modules to each of the stages, and to arbitrarily combine them. To demonstrate our system, we implemented all the modules necessary to create a pipeline for public dissemination of cell biology based on confocal-microscopy data.

In the following sections, we discuss the individual stages, and the modules we implemented to realize our multi-pipeline.

3.2 Data Processing

The first step in creating an illustration is to acquire the necessary data. The data processing stage has to be able to process all acquired data types, and prepare them for the rendering. Confocal laser scanning microscopy (CLSM) creates a 3D volume of the observed specimen, such as a human cell. By modifying the cells, so that some of their organelle systems produce fluorescent light when excited by the laser beam, it is possible to scan each organelle system separately. This results in several volumes, which need to be further processed: First, the organelle structures are segmented. Some of the scanned structures, such as the cell membrane, can be smoothed and converted to *meshes*, since they do not have to convey any volumetric information. Organelles such as microtubules or mitochondria, which form tubular structures, are in general segmented with a backbone segmentation, and only the *backbones* are used for the visualization. Finally, in some situations, it is also necessary to show the *original data* in the volumetric form. We implemented modules for loading and smoothing of volumes, meshes, and backbone segmentations.

3.3 Rendering

After the data processing, the datasets are rendered. Since the input data are diverse (volumes, meshes, backbones), there has to be a separate renderer for each data type.

For rendering volumes and meshes, we use well known methods, such as directional occlusion shading [25] or volume raycasting. For both mesh and volume rendering, we provide the same set of shaders to mimic the look of the visual appearance created by illustrators.

Each renderer is realized as a standalone module, which renders the input dataset into an off-screen buffer. The renderers have to output the depth-buffer for each image too, so that the individual images can be later composed together in image-space with the depth testing implemented as a post-processing effect. Thanks to this design it is possible to split the rendering tasks between separate GPUs if necessary, since before compositing the renderers are independent of each other.

3.4 Compositing

After the rendering, all the images are composited into a single visualization. For that purpose, all the renderers have to produce images with the same size and the same format of the depth buffer. Based on the depth information, we are able to composite the images by means of order-independent transparency. For semi-transparent volumetric renderings, an approximation has to be taken, such as assuming a solid surface where the opacity accumulates above a certain threshold. In our application area, we are working with volumes obtained by 3D fluorescent microscopy, where each volume contains only a single object, and so there are no nested structures. Therefore, for our purposes, this is a satisfactory approximation, as shown in Section 5.

The order-independent transparency technique ensures that the individual organelles can be rendered semi-transparently, or turned off completely. Additionally, this way of rendering allows for applying certain post-processing effects on individual layers, such as contours or halos. This is necessary to achieve those visual effects which illustrators use for particular storytelling or highlighting purposes.

3.5 Deployment

To finalize the pipeline between the original data and target audience, Marion implements the fourth stage - Deployment. In this stage, the visualization created in the previous stages is placed within a suitable software environment so that it can be delivered to the final users. This is done in a fast and flexible manner which supports fast prototyping as well as deploying of the finalized solutions.

The modules within this stage are in fact applications, in which the previous stages are embedded. With its internal architecture and helper tools , Marion ensures that this embedding is easily possible, and the same visualization pipeline is reusable across multiple applications. This means that in case one of the previous stages is modified, these changes are automatically passed on to in all applications within the deployment stage. This happens without the necessity to change any application code, or even to restart running applications. This is advantageous in case the application is a running server, since it is possible to update its visualization pipeline without any downtime. Additionally, this design supports fast prototyping of the visualization code, since it avoids all the data having to be reloaded on every change of the visualization pipeline.

We implemented two applications within the deployment stage to demonstrate the possibilities of our system. The first one is a desktop application, which provides a user interface for setting the visualization parameters. The application is scriptable and it can be used to create story-telling animations using the cell-visualization pipeline.

The second application consists of a remote-rendering server and thin client handling the interaction and displaying of the rendered images. This application is used to deliver the visualized content in an interactive manner to the target users over the internet. All the rendering happens on the server, where a load-balancing mechanism is in place. All the interaction is implemented on the client side, and it allows the users to set the desired rendering parameters. The client part can be embedded in any web-page, while the server can handle multiple users at the same time. The server is independent of the actual visualization pipeline, and its load-balancing mechanism automatically adjusts to it . Therefore, no server code has to be changed when the visualization pipeline is updated.

Both applications implemented within the deployment stage of our system support the public dissemination of biology knowledge based on measured or simulated data. The remote-rendering solution makes the interactive visualization available also on devices which lack the necessary processing power to create such visualizations on their own.

4 System Architecture

The purpose of our system is to make a database of cell-microscopy scans accessible to laypersons in the form of interactive illustrations. The database is constantly being updated with higher quality scans, as well as by adding new datasets. Additionally, it is often necessary that the created illustrations are deployed to new platforms, or reused in different applications. This is typically very difficult to achieve with existing visualization systems, where the visualization pipelines are integral parts of a specific application.

Therefore, the contribution of this paper is twofold: an architecture of a system that allows development of visualization pipelines in such a dynamic environment, which is described in this section; the second contribution is an actual pipeline implemented within the system, which creates interactive illustrations of cells from the microscopy data, which is described in Section 5.

From the software architecture point of view, Marion consists of two main components, i.e., the *visualization layer* and the *application layer*. The visualization layer is used to implement the first three stages of the visualization multi-pipeline as shown in Figure 3. The application layer implements the deployment stage. Within the deployment stage, standalone applications, which use Marion, are written.

4.1 Visualization Layer

The visualization layer of our system is implemented as a standalone library. In this layer, all the modules necessary for data processing, rendering, and compositing are implemented. Additionally, the library provides functionality for fast prototyping and deployment of the modules. In this layer, the whole visualization is designed. This layer is not concerned with any sort of interaction. Its sole task is to render an image according to the given visualization parameters and input data.

4.1.1 System

The system part of the visualization layer is responsible for handling the code and the resources of the visualization multi-pipeline. On its own, this part of the visualization layer can be used for general purpose graphics applications and it is not restricted only to visualization tasks.

Our system is build in a way that it supports both fast low-level prototyping of illustrative rendering techniques and quick deployment to different platforms. This mean the outputs of our system can be quickly adapted to new types of data, directly within a running infrastructure without any significant down-times. Existing toolkits, such as VTK or ParaView, do not offer such functionality. They provide means for fast prototyping, but unlike in our system, the deployment of these prototypes to different platforms is tedious.

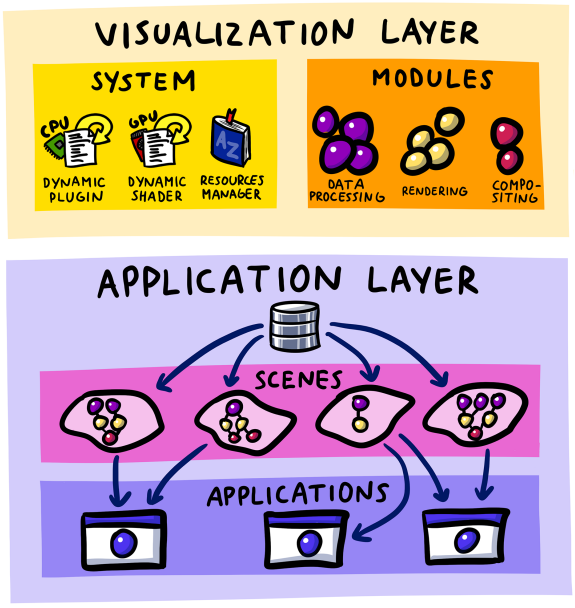


Fig. 4: The architecture of our system.

Code For the development of complex, extensible visualizations, it is necessary to have a possibility to structure the code into modules in an intuitive way. In scientific visualization, two types of code are being used - CPU (or host) code, and GPU (or device) code. The visualization layer offers an encapsulating structure for both types of code. For the CPU code the structure is referred to as *dynamic plugin*, while for the GPU code it is referred to as *dynamic shader*. The developer can create an arbitrary number of dynamic plugins and dynamic shaders. Both types of the code modules have in common that the visualization layer automatically reloads them when the files in which they are stored are modified. For the GPU code this is trivial, since the GPU shaders are always compiled during the application run-time. For the CPU code, this is achieved through the use of dynamically-linked libraries. More details about this mechanism are given in Section 6.

Thanks to the dynamic reloading of the code modules, it is possible to observe the implementation changes immediately, without restarting the host application and reloading all the data, which might otherwise take a significant amount of time. In this way, the visualization layer ensures fast prototyping of visualization multi-pipelines.

Resources In the visualization of scientific data, various types of resources are frequently used. These are code modules (dynamic shaders and dynamic plugins), and buffers, such as render targets, textures, or datasets. In order to enable fast prototyping and development of visualization algorithms, the usage of these resources has to be simplified as much as possible.

The visualization layer implements a resource manager. All the resources are stored in dictionaries, implemented as hash maps, where the keys are string IDs. In this way, all the code modules, datasets, and GPU buffers are stored. They are created with a unique ID, which can be later used to retrieve them. The creation, as well as referencing a particular resource, is reduced to a single function call, where the unique ID is passed as a parameter.

In the background, the visualization layer handles these resources in specific ways. For instance, it ensures that the dynamic shaders and the dynamic plugins are reloaded whenever they are modified, e.g., after recompilation. Specific types of GPU buffers, the frame-buffer objects, which are used as rendering targets, are automatically re-sized when a re-size signal is received by the visualization layer. Therefore, the creation and usage of all types of resources is comfortable, since these mechanisms are hidden from the programmer.

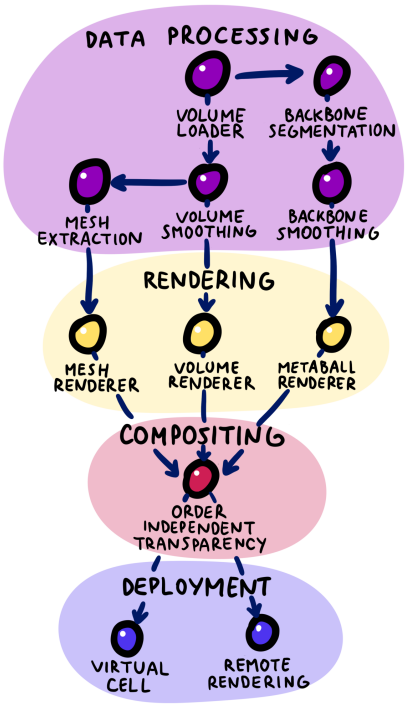


Fig. 5: Overview of the modules implemented within our system in the individual stages of the visualization multi-pipeline for illustrative visualization of the cell microscopy data.

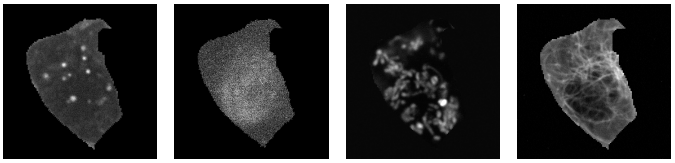
4.1.2 Modules

The modules within the visualization layer implement individual algorithms for loading, processing, and rendering data. Typically, they are implemented as either dynamic shaders, dynamic plugins, or an arbitrary combination thereof. However, it is also possible to implement them as static libraries, compiled together with the rest of the visualization layer. In this case, the developer trades the flexibility of dynamic shaders and dynamic plugins for a possible performance increase, or compactness of the final application.

Data processing For the purpose of the data processing modules, the visualization layer contains data structures for holding meshes, volumes, and fiber structures (sets of one-dimensional poly-lines, i.e., a generalization of point clouds) . These structures are stored in a paged memory layout, which means that for each dataset a certain number of continuous memory blocks, or pages, is allocated. This ensures a relative data locality. However, it is scalable to large datasets that might otherwise not fit in the memory in a completely continuous way.

Rendering The rendering modules are based on *OpenGL*. Generally, they only render into off-screen buffers, which are managed by the visualization layer. It is the task of the host application to display the final rendering on screen, if desired.

Compositing For the purpose of the compositing of the final images from the individual renderings, the visualization layer is designed in a way that all the resources are global, and they are accessible to all the modules. This is realized by automatically providing a pointer to the visualization layer instance to every module. Through this pointer, the modules can access the resource dictionaries, which contain all the frame-buffer objects that are used by the rendering modules as render targets. Therefore, the compositing modules can access previously rendered images, as well as their depth buffers, and implement various compositing schemes, such as order-independent transparency. Similarly to the rendering modules, the output of a compositing module is also an off-screen buffer, which can be used by the host application in a flexible way.



(a) (b) (c) (d)

Fig. 6: The original data as scanned by the confocal microscope. (a) Cell membrane, (b) nucleus, (c) mitochondria, (d) microtubules.

4.2 Application Layer

The application layer of the Marion system consists of host applications, which embed a visualization multi-pipeline implemented through the visualization layer. This layer ensures that the visualizations can be easily deployed to the target audiences across different software and hardware platforms.

4.2.1 Scenes

In order to make the visualization multi-pipelines easily reusable across multiple applications, we introduce the concept of *scenes*. Each scene implements a single multi-pipeline by accessing the visualization layer of the Marion system. The scenes are implemented as dynamic plugins. Therefore, each scene contains an independent code, that produces one or more images stored in the off-screen buffers managed by the Marion's resource manager. The scenes carry additional information, such as the starting time and the duration. This information can be used when the final visualization consists of several animated sequences, where each sequence is implemented as a single scene.

4.2.2 Applications

The final step of connecting the underlying data with the target audience is to implement a host application. These applications can use any functionality of the visualization layer of Marion. However, the intended usage of the system is that all the rendering functionality is implemented within the scenes, and the host application is only using existing scenes. In this way, the visualization code is centralized and shared amongst all the applications that might be developed. The host applications are then either showing the rendered images, or sending them over the network to their target audience. Changes made to the scenes, either because of the changes in the nature of the underlying data, or because of an aesthetic choice, are automatically reflected in all applications.

5 System in Action: Communicating Cell Biology

In this section, we describe the visualization multi-pipeline which we implemented within Marion for creating illustrations of cells, which are visually comparable to the professional illustration shown in Figure 2. However, the requirements are that the cells are visualized in real-time, and delivered to the target audience over the internet or in a form of video, this avoiding the necessity of the users owning a high-end graphics hardware. The modules we implemented to achieve this goal are shown in Figure 5.

5.1 Data processing

At the current stage, four different organelle systems of a human induced pluripotent stem cell are being scanned by our collaborators in the form of image stacks forming 3D volumes. A single slice from each volume is shown in Figure 6. A direct volume rendering of these datasets would not result in a comprehensible visualization of the organelle systems, since they are too noisy and their resolution is too low. Therefore, we implement several modules to process these data and extract models suitable for creating illustrative visualizations.

In the final visualization of the cell, it is necessary to communicate the approximate shape of the cell membrane and the nucleus, as well as the relative size and position of the nucleus within the cell. All

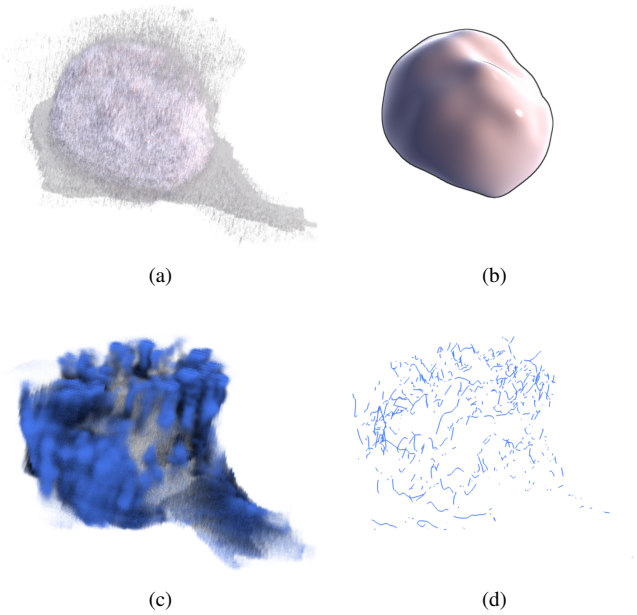


Fig. 7: (a) Direct volume rendering of the raw nucleus data: due to the high level of noise in the original data, direct volume rendering does not convey the shape of the nucleus well. (b) Visualization of the extracted nucleus surface. (c) Direct volume rendering of the raw mitochondria data. (d) Extracted center-lines for the individual mitochondria.

the other information present in the acquired data can be abstracted away, since they would reduce the clarity of the resulting visualization without providing any added value. Direct volume rendering of the raw nucleus data is shown in Figure 7a.

To extract an approximate shape of the membrane and the nucleus from the original data, a segmentation mask is created for both structures. For the membrane, seeded watershed [21] is used. The segmentation of the nucleus is the result of a combination of adaptive local normalization, 3D level set segmentation, and weighted seed merging. This results in noise-free volumes describing the shapes of the structures. However, their resolution is still too low.

To extract smooth surfaces from the generated segmentation masks, the masks are filtered with a three-dimensional Gaussian kernel. Subsequently, iso-surfaces are extracted. The result of this process, a smooth surface suitable for an illustration, is shown in Figure 7b.

The same approach cannot be used on much finer organelles, namely the mitochondria and the microtubules, since they are too fuzzy for meaningful surfaces to be extracted from the data. Instead, center-lines are extracted for these organelles. For this purpose, MitoGraph [32] is used. We up-sample the extracted center-lines and filter them with a 1D Gaussian kernel, since the low resolution of the original data causes staircase artifacts. Up-sampling and filtering produces smooth curves. Figure 7c shows how the mitochondria dataset looks like when directly visualized, Figure 7d shows the extracted center-lines.

5.2 Rendering

In the next step, each of the organelle system is rendered into a separate image. We have implemented several shaders useful for achieving the desired illustrative visual style. For the membrane (Figure 8a) and the nucleus (Figure 8b), contours are used together with a Fresnel shader to indicate the 3D shape of these structures. The contours are calculated by applying a Sobel operator to the depth buffer, which ensures that the inner contours are included as well. The nucleus additionally uses tone-mapping to achieve a less synthetic look.

To achieve an organic look of the mitochondria (Figure 8c) and microtubules (Figure 8d), for which the center-lines have been extracted, metaballs are used. The metaballs technique has originally been introduced by Blinn [2] for rendering molecular data. We place a single metaball on every vertex of every center-line. We implemented a screen-space approximation, which is able to render a dataset consisting of approximately 65000 vertices, in real-time. However, the approach is scalable to a much larger number of vertices, since it is performed in screen-space.

The algorithm renders sphere impostors in place of each vertex, which is realized through billboarding. Separate color, depth, and normal buffers are rendered. The normal buffer is blurred, and deferred shading is applied to compose the final image approximating the visual appearance of metaballs.

In order to adequately illustrate the organelle surfaces, it is necessary that the metaballs belonging to different mitochondria or microtubules do not blend into each other. Therefore, for the blurring step, we use a trilateral filter, which calculates the kernel weights based on the pixel distance, the differences in depth, and whether pixels belong to the same mitochondrion or not. This ensures that the screen-space edges between different mitochondria are not blurred, while the depth component ensures that the different parts of a single mitochondrion, which are not next to each other, are not blurred either. The results of this rendering method are shown in Figure 9.

Even though the trilateral filter is not separable, we approximatively divide it into a vertical and a horizontal pass. This causes some undesired artifacts, however, since the center-lines are densely sampled in the data processing step, only a small blur radius is required to compensate this through the metaball effect and the amount of visual artifacts is therefore negligible.

5.3 Compositing

After all the organelle systems are rendered, they have to be composited. For this we use a simple order-independent transparency method. For the nucleus, mitochondria, and microtubules, the corresponding depth-values for each pixel are sorted and the color-values are composited in a back-to-front order using alpha blending. The membrane is then simply blended on top of this image, since the membrane is always the outermost structure. The final image is shown in Figure 8e.

Blending the semi-transparent membrane on top of the other organelles in screen-space allows us to use screen-space effects which depend on the depth information, such as screen-space ambient occlusion, depth-of-field, or fog. Compositing the membrane with other organelles in object-space would still allow us to use semi-transparency for each organelle system, but we would not be able to apply these effects. Figure 10 shows a cell rendered without and with the screen-space effects. Applying the effects enhances the depth perception of the individual objects, while their screen-space implementations ensures the scalability of the visualization for large datasets.

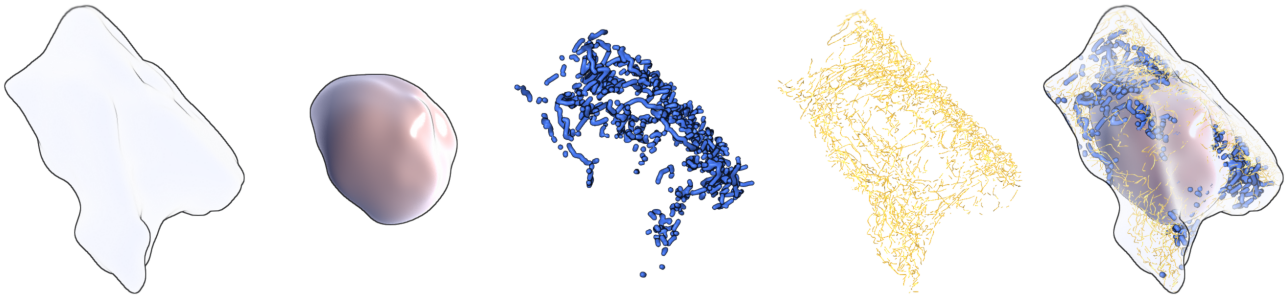
5.4 Deployment

The final step of making the cell data accessible to the general public is the deployment of the visualization designed in the previous steps. With the requirement of reaching as large an audience as possible, we implemented the previous steps into two separate *scenes*, as described in Section 4.2 so that they can be reused in different applications. The first scene contains the data loading and processing, while the second scene contains the rendering and compositing. The reason for this is that we can modify the visualization pipeline while the host application is still running. If the data loading and processing would be part of this scene, it would be executed every time we change the visualization pipeline, which is not desirable.

We implemented two distinct applications within the deployment stage, both reusing these scenes. These applications are described in the following subsections.

5.4.1 Virtual Cell - A Storytelling Application

We implemented a desktop application which uses both scenes mentioned in previous section to display the cell on the screen. The added value of the application is a graphical user interface, through which the visualization parameters of individual stages can be specified. The parameter settings can be stored in so-called *presets*. The presets can



(a) (b) (c) (d) (e)

Fig. 8: Individual organelle systems extracted from the original data. (a) Cell membrane, (b) nucleus, (c) mitochondria, (d) microtubules, (e) final composed image.

be timed using a simple scripting interface. The application then interpolates all the parameters in accordance with the created script. In this way, it is possible to create animations within the entire parameter space of the visualization. Another use case is to expose the user interface and allow the users to freely explore the visualization parameter space.

5.4.2 Remote-Rendering Server

To be able to deliver the cell visualizations to large audiences, we implemented a remote-rendering server. The server uses both scenes implementing the visualization, but does not display it on the screen. Instead, it acts as an application server listening to connections from clients. Each client can request an image rendered with given parameters. The remote-rendering server executes each request by rendering the image in an off-screen buffer, and sending the result back to the client over a *WebSocket* interface encoded as a *JPEG* image.

The server supports multi-threaded rendering with a load-balancing mechanism for the utilization of multiple GPUs, if these are present. The multi-threaded rendering works by starting several threads and creating a separate OpenGL context for each of the threads. All the data are loaded separately within each of the contexts, so that the rendering can be done independently in each thread. The XYZ extension can be used to specify the affinity of each thread to a separate GPU, if this operation is supported in the given system.

The load-balancing mechanism chooses which thread is used to render a request. For this, an estimate of the rendering duration of each request is necessary. The rendering threads have request queues, where the incoming requests are placed. The sum of the estimated rendering durations of all requests in a queue gives an estimated load of the given thread. The load-balancing mechanism always places an incoming request into the queue with the lowest estimated load.

For this mechanism to work, it is essential that the system can realistically calculate estimated rendering durations of the incoming requests. To achieve this, upon startup the server samples the parameter space of the visualization by creating permutations of the parameter settings. Subsequently, it renders the scene a predefined number of times for each parameter setting and from different viewpoints. The duration of each rendering is recorded, and the durations for all the renderings from different viewpoints for the same parameter setting are averaged. These averages are stored. Whenever a request arrives, the server looks up the averaged rendering times for the parameter settings of this request. Subsequently it is used as an estimation of the rendering duration. In our use-case, we take the visibilities of the individual organelle systems as the relevant parameter settings, since turning on or off organelle systems has the most significant influence on the duration of the rendering.

The server uses the visualization multi-pipeline for rendering samples of the parameter settings without making any assumptions on how the multi-pipeline actually works. Therefore, the load-balancing mechanism automatically adapts to a particular implementation of the visualization multi-pipeline.

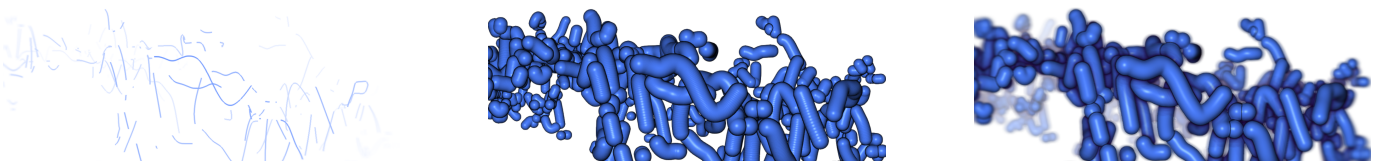
5.4.3 Remote-Rendering Client

Our remote-rendering solution utilizes a thin client, which implements a user interface to set the visualization parameters, as well as a camera widget based on ArcBall [28] rotations. The client does not have any rendering capabilities, it only shows the images received from the server. The client is implemented in JavaScript. With this design, it is possible to run the client on a large variety of devices, which are not required to be equipped with a powerful GPU.

6 Implementation

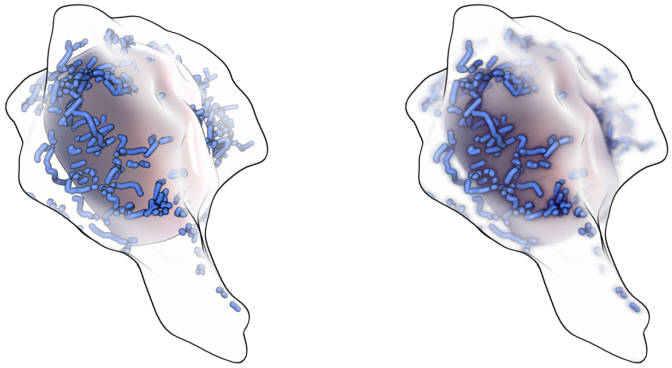
Our system is implemented in C++ and it uses the Qt library [31]. The rendering is based on OpenGL. Besides that, there are no other dependencies. The entire visualization layer, as well as the individual scenes, are implemented as dynamically-linked libraries. Therefore, it is possible to use the system without static linking.

An important aspect of our system is that it is able to reload CPU and GPU code without restarting the host application. For the GPU code this is done by watching changes of the shader files, and upon their modification, the respective shaders are recompiled in the runtime. For the CPU code, it is required that the code is compiled into a dynamically-linked library. Instead of loading it directly, the system



(a) (b) (c)

Fig. 9: (a) Center-lines extracted from the mitochondria dataset. (b) Every vertex is replaced by an impostor sphere. (c) The normal buffer is blurred with a trilateral filter, which makes individual mitochondria appear as smooth 3D objects. The filter prevents individual mitochondria to blend into each other. Additionally, screen-space ambient occlusion, depth of field, and fog effects have been applied in a post-processing step to provide sufficient depth cues for the scene. The screen-space implementation ensures interactive frame rates.



(a) (b)

Fig. 10: (a) No post-processing effects are used. (b) Screen-space ambient occlusion, screen-space volumetric fog, and screen-space depth-of-field are applied. Since the semi-transparent membrane is blended in image-space on top of the rest of the scene, it is possible to use these screen-space effects together with the semi-transparent membrane.

makes a copy of the library file and loads the copy. In this way, it is still possible to modify the original library file, e.g., through a recompilation of the particular program. Marion watches the changes of the original library file. Whenever a modification is detected, it unlinks the library and erases the copy from which the library has been loaded. It then proceeds to create a copy of the new version of the library, and loads it.

7 Discussion and Future Work

Our remote-rendering solution is going to be deployed at the AICS to support the public dissemination of their microscopy data of human cells. After the deployment, we plan to evaluate the performance of the system on the large user group composed of AICS website visitors.

The illustrators and visualization tool software engineers at the AICS specified a set of requirements for the rendering of cells, so that the datasets are easily understandable by the general public. The illustrators also confirmed that our system fulfills their requirements. These include the possibility to use depth-cueing effects, such as fog and depth of field; illustrative effects, such as Fresnel shaders applicable to any channel, contours, and metaball rendering; composition of the individual organelles by means of order-independent transparency; volume rendering of noisy data, which we achieved by gradient-free shading. Additionally, the system exposes all the necessary parameters through the user interface, so that the illustrators have full control over such aspects as the level of simplification of the datasets.

In the future, we intend to present certain discoveries based on nanoscale models of the cellular structures at atomic detail. Such discoveries might be, for instance, protein colocalizations, interactions or other functional aspects derived from structural changes in the cells.

Our system already supports this multiscale aspect, as we already implemented a molecular rendering module within Marion, which can currently be used for rendering microtubules on molecular scale (as demonstrated in Figure 11). We plan to further extend the nanoscale rendering capabilities with more complex mechanisms to support dynamic scenes of molecular interactions.

The interactions with the current single-scale visualizations can be handled by the standard features of the underlying Qt library. Therefore, no additional abstraction level was needed. For a proper support of interactions with multiscale data, however, our system will need to be extended by special interaction modules. One of these modules is a multiscale camera, which adapts the zoom speed semantically, based on the target of a zoom operation.

Another aspect we would like to explore in the future, are multi-user guided tours within the cell. Such guided tours can be used for teaching through interactive storytelling. This will allow users to virtually experience new biological discoveries through interactive tours guided by narration.We are also working on tools for comparing

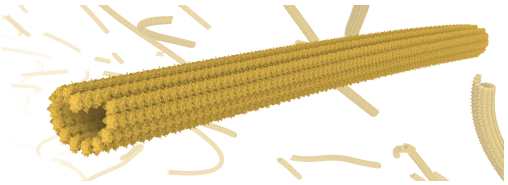


Fig. 11: Detail of a microtubule rendered at a nanoscale resolution.

two or more cells both visually, statistically, and methods to integrate these two approaches.

Since the discoveries of the Allen Institute will be publicly announced, it might happen that the remote-rendering server load will peak shortly after the announcements. Our load-balancing mechanism can already efficiently utilize the resources available to the server, but this might not be enough during the peak times. Therefore, we plan to investigate further scalability options. One idea is to introduce a smart caching system, where the images, which have been already rendered, are stored. Whenever a similar image is requested, the respective image is taken from the cache, transformed in image-space so that it fits the request, and is sent to the user. The system would avoid caching images that are too similar, for instance with respect to the camera settings. In this way, a lot of processing power can be saved, and reasonable storage requirements can be maintained.

While our system is tailored to biological visualization, it includes modules covering many illustrative-rendering tasks to help focus the attention of different types of end users. It contains functionality for volume and mesh rendering, various illustrative effects such as contours and depth-of-field, and several simple filters like Gaussian and bilateral blur. These can be arbitrarily combined in order to use our system in different applications. Besides the data structures for storing volumes, meshes, and point clouds, the system also includes low-level memory structures for holding arbitrary data. This makes it convenient to extend the system to support new data types.

8 Conclusions

We presented a system for the dissemination of knowledge in cell science by means of interactive illustrative visualizations. Our system is powered by a *visualization multi-pipeline* that specifically targets two aspects that are associated with this task. First, it is capable of handling the illustrative visualization of volumetric data of various origins and degrees of quality. In order to achieve this, the multi-pipeline offers several modules for the processing and rendering of heterogeneous data sources that can be combined sequentially and in parallel. Secondly, the multi-pipeline allows the deployment of the generated visualizations at interactive frame rates to a wide range of software environments and output devices. Our system is therefore specially tailored to the quickly changing and accumulating data types and data entries of the short lived research results that are produced by cutting edge cell science. The flexibility of our system enables scientists to explore their data in the raw volumetric and in the refined illustrative representations. At the same time, guided illustrations with simple interactions can be produced for public knowledge dissemination, by allowing the partial masking the system's features and by supporting scripted changes of parametrizations. We demonstrated the capabilities of our multi-pipeline based on a real world application that will be publicly available. We are excited about the potential of our system with the goal of integrating all available data about human cells in a single framework.

Acknowledgements

This project has been funded by the Vienna Science and Technology Fund (WWTF) through project VRG11-010 and also supported by EC Marie Curie Career Integration Grant through project PCIG13-GA-2013-618680 and ManyViews, OeAD Scientific & Technological Agreement SK 14/2016. Johannes Sorger has been partially supported in the scope of the FWF-funded project P24597-N23 (VISAR) and the COMET K1 program of the Austrian Funding Agency (FFG).

REFERENCES

[1] J. Ahrens, B. Geveci, and C. Law. Paraview: An end-user tool for large data visualization. *The Visualization Handbook*, 717, 2005.

[2] J. F. Blinn. A generalization of algebraic surface drawing. In *Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '82, pp. 273–. ACM, New York, NY, USA, 1982. doi: 10.1145/800064.801290

[3] S. Bruckner, I. Viola, and M. E. Gröller. Volumeshop: Interactive direct volume illustration. In *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05. ACM, New York, NY, USA, 2005. doi: 10.1145/1187112.1187183

[4] K. Engel, P. Hastreiter, B. Tomandl, K. Eberhardt, and T. Ertl. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *Proceedings Visualization 2000. VIS 2000 (Cat. No.00CH37145)*, pp. 449–452, Oct 2000. doi: 10.1109/VISUAL.2000.885729

[5] S. Grottel, M. Krone, C. Muller, G. Reina, and T. Ertl. MegaMol–A Prototyping Framework for Particle-Based Visualization. *IEEE Trans Vis Comput Graph*, 21(2):201–214, Feb 2015.

[6] J. H. Iwasa. Animating the model figure. *Trends Cell Biol.*, 20(12):699–704, Dec 2010.

[7] G. T. Johnson, L. Autin, M. Al-Alusi, D. S. Goodsell, M. F. Sanner, and A. J. Olson. cellPACK: a virtual mesoscope to model and visualize structural systems biology. *Nat. Methods*, 12(1):85–91, Jan 2015.

[8] G. T. Johnson, D. S. Goodsell, L. Autin, S. Forli, M. F. Sanner, and A. J. Olson. 3D molecular models of whole HIV-1 virions generated with cellPACK. *Faraday Discuss.*, 169:23–44, 2014.

[9] J. Jomier, S. Jourdain, U. Ayachit, and C. Marion. Remote visualization of large datasets with midas and paraviewweb. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pp. 147–150. ACM, New York, NY, USA, 2011. doi: 10.1145/2010425.2010450

[10] D. Koller, M. Turitzin, M. Levoy, M. Tarini, G. Croccia, P. Cignoni, and R. Scopigno. Protected interactive 3d graphics via remote rendering. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pp. 695–703. ACM, New York, NY, USA, 2004. doi: 10.1145/1186562.1015782

[11] I. M. Kramer, H. R. Dahmani, P. Delouche, M. Bidabe, and P. Schneeberger. Education catching up with science: preparing students for three-dimensional literacy in cell biology. *CBE Life Sci Educ*, 11(4):437–447, 2012.

[12] M. Krone, C. Müller, and T. Ertl. Remote rendering and user interaction on mobile devices for scientific visualization. In *Proceedings of the 8th International Symposium on Visual Information Communication and Interaction*, VINCI '15, pp. 21–26. ACM, New York, NY, USA, 2015. doi: 10.1145/2801040.2801057

[13] F. Lamberti and A. Sanna. A streaming-based solution for remote visualization of 3d graphics on mobile devices. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):247–260, March 2007. doi: 10.1109/TVCG.2007.29

[14] M. Le Muzic, L. Autin, J. Parulek, and I. Viola. cellview: A tool for illustrative and multi-scale rendering of large biomolecular datasets. In *Proceedings of the Eurographics Workshop on Visual Computing for Biology and Medicine*, VCBM '15, pp. 61–70. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2015. doi: 10.2312/vcbm.20151209

[15] M. Le Muzic, J. Parulek, A. Stavrum, and I. Viola. Illustrative visualization of molecular reactions using omniscient intelligence and passive agents. *Comput. Graph. Forum*, 33(3):141–150, June 2014. doi: 10.1111/cgf.12370

[16] N. Lindow, D. Baum, and H.-C. Hege. Interactive rendering of materials and biological structures on atomic and nanoscopic scale. *Computer Graphics Forum*, 31(3pt4):1325–1334, 2012. doi: 10.1111/j.1467-8659.2012.03128.x

[17] K.-L. Ma and D. M. Camp. High performance visualization of time-varying volume data over a wide-area network. In *Supercomputing, ACM/IEEE 2000 Conference*, pp. 29–29, Nov 2000. doi: 10.1109/SC.2000.10000

[18] I. M. Martin. Adaptive rendering of 3d models over networks using multiple modalities. Technical report, 2000.

[19] E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin. Ucsf chimera - a visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, 25(13):1605–1612, 2004.

[20] S. Pieper, M. Halle, and R. Kikinis. 3d slicer. pp. 632–5, 04 2004.

[21] A. Pinidiyaarachchi and C. Wählby. *Seeded Watersheds for Combined Segmentation and Tracking of Cells*, pp. 336–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. doi: 10.1007/11553595_41

[22] A. Rosset, L. Spadola, and O. Ratib. OsiriX: an open-source software for navigating in multidimensional DICOM images. *J Digit Imaging*, 17(3):205–216, Sep 2004.

[23] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J. Y. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, and A. Cardona. Fiji: an open-source platform for biological-image analysis. *Nat. Methods*, 9(7):676–682, Jun 2012.

[24] J. Schindelin, C. T. Rueden, M. C. Hiner, and K. W. Eliceiri. The ImageJ ecosystem: An open platform for biomedical image analysis. *Mol. Reprod. Dev.*, 82(7-8):518–529, 2015.

[25] M. Schott, V. Pegoraro, C. Hansen, K. Boulanger, and K. Bouatouch. A directional occlusion shading model for interactive direct volume rendering. In *Computer Graphics Forum*, vol. 28, pp. 855–862. Wiley Online Library, 2009.

[26] W. Schroeder, K. M. Martin, and W. E. Lorensen. *The Visualization Toolkit (2Nd Ed.): An Object-oriented Approach to 3D Graphics*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[27] J. Sharpe, C. J. Lumsden, and N. Woolridge. *In Silico: 3D Animation and Simulation of Cell Biology with Maya and MEL*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[28] K. Shoemake. Arcball: a user interface for specifying three-dimensional orientation using a mouse. In *Graphics Interface*, vol. 92, pp. 151–156, 1992.

[29] N. Smit, C.-W. Hofstede, A. Kraima, D. Jansma, M. deRuiter, E. Eisemann, and A. Vilanova. The Online Anatomical Human: Web-based Anatomy Education. In B. S. Santos and J.-M. Dischler, eds., *EG 2016 - Education Papers*. The Eurographics Association, 2016. doi: 10.2312/eged.20161025

[30] E. Sundén, P. Steneteg, S. Kottravel, D. Jönsson, R. Englund, M. Falk, and T. Ropinski. Inviwo - An Extensible, Multi-Purpose Visualization Framework. Poster at IEEE Vis, 2015.

[31] The Qt Company. Qt.

[32] M. P. Viana, S. Lim, and S. M. Rafelski. Quantifying mitochondrial content in living cells. *Methods in cell biology*, 125:77–93, 2015.

[33] T. Walter, D. W. Shattuck, R. Baldock, M. E. Bastin, A. E. Carpenter, S. Duce, J. Ellenberg, A. Fraser, N. Hamilton, S. Pieper, M. A. Ragan, J. E. Schneider, P. Tomancak, and J.-K. Hériché. Visualization of image data from cells to organisms. *Nat Methods*, 7(3 Suppl):S26–41, 2010 Mar 2010. doi: 10.1038/nmeth.1431