Development of Multiplatform Adaptive Rendering Tools to Visualize Scientific Experiments

Konstantin Ryabinin¹ and Svetlana Chuprina¹

¹Perm State University, Bukireva Str. 15, 614990, Perm, Russia
kostya.ryabinin@gmail.com, chuprinas@inbox.ru

Abstract
In this paper, we propose methods and tools for multiplatform adaptive visualization system development adequate to the specific visualization goals of the experiments in the different fields of science. Approach proposed was implemented and we present a client-server rendering system SciVi (Scientific Visualizer) which provides multiplatform portability and automated integration with different solvers based on ontology engineering methods. SciVi is developed in Perm State University to help scientists and researchers acquire the necessary multidisciplinary skills and to solve real scientific problems.

Keywords: Scientific visualization, Multiplatform portability, Ontology, Mobile devices, Adaptivity

1 Introduction

The high variety and big volume of data is one of the main problems in scientific visualization. Nowadays almost every scientific problem is related to measuring of the parameters of real objects or to modeling the processes that involve these objects. In both cases there are hardware and software systems called solvers that manage the scientific experiment and produce output data in a special format. These data are to be analyzed by other systems or by humans. Usually the data intended to be analyzed by humans have to be visualized. However the formats of output data as well as the interfaces of solvers differ significantly from one particular solver to the other due to the high variety of research objects in different domains. Often, this leads to the fact that it becomes impossible to reuse the visualizers across different solvers, even in the same field of science. The scientists have to develop a new visualizer for each new solver or to convert the data obtained from the new solver to the existing visualizers manually. Therefore there is an actual need to develop the visualization systems that could be easily adapted to the different solvers, because this will reduce the cost of program development and make the research process more efficient.

Very often the scientific solvers require a huge amount of computational power or special input devices like MRI scanners or genome sequencers. Usually the solvers run on supercomputers or on special hardware systems and the visualizers cannot be a part of them in these cases. To visualize data
obtained from such kind of solvers client-server technology can be used. The visualizer as a client can run on the scientist's desktop computer or even on the mobile device and it is necessary to establish the connection to the solver via a local network or Internet.

Many scientific problems require 3D visualization. Since the rendering is based on the discrete process of rasterization, the aliasing problem is inevitable. To improve the quality and realism of the image produced by 3D graphics rendering systems anti-aliasing is used. Although there are a lot of anti-aliasing algorithms, there are still challenges in this filed. The main ones are listed below:

1. Any kind of anti-aliasing reduces the performance. For example, the multisampling (3D Center, 2003) anti-aliasing that is used as a standard built-in technique in many rendering systems and has hardware support on the most graphics cards reduces the performance in average in 3 times on mobile devices running under iOS and Android.

2. Very often the anti-aliasing algorithms produces artifacts on the image like unwanted blur of small details.

The goal of our work is to develop an adaptive multiplatform scientific visualization system based on client-server architecture that allows automated integration with different solvers, provides rendering distributed between client and server and eliminates the aliasing problems. The client can run on the desktop computers and on mobile devices, the server – on the desktop computers and high-performance computing systems. The highlights of our work include the following:

1. To provide the integration of any solver with visualization system we suggest the approach based on domain-specific ontologies. We have implemented our approach in the visualization system adaptable to different solvers in unified way including solvers for experiments of very different scientific fields.

2. To solve the problem of multiplatform portability we have developed the framework that provides an abstraction layer between the operating and visualization systems that allows building graphical user interface for different platforms based on the high-level declarative description.

3. To improve the performance of rendering we propose the special heuristics to adaptive distribution of rendering between client and server.

4. To solve the aliasing problems we have implemented the adaptive algorithm of anti-aliasing that ensures the high quality of image rendered without hitting the interactivity and smoothness of animations.

5. The results of our research are proved by solving the real interdisciplinary scientific tasks, such as visualization of:
   - magnetic moments of nanoparticles (physics);
   - prices on the stock market (economy);
   - genetic sequences and phylogenetic trees (genomics).

2 Background

The analysis of the most popular scientific visualization systems and tools (Ryabinin & Chuprina, 2013) showed that there are only a few multiplatform solutions and they are not flexible enough to integrate with arbitrary third-party solvers. In our opinion the most versatile and powerful tools for scientific visualization are the VTK (Visualization Toolkit) and the systems based on VTK class library.

VTK is an open-source, freely available software system for 3D computer graphics, image processing and visualization: VTK for desktop computers, pVTK for high-performance computing systems and VES for mobile devices.

VTK is a family of tools designed to render complex 3D-objects using different techniques for rendering surfaces, volumes, slices, stereo images, etc. There are two most popular scientific
visualization systems based on VTK: ParaView for desktop computers and KiwiViewer for mobile devices. They allow rendering complex data but have no means for automatic integration with arbitrary solvers.

We decided to use the libraries from VTK family as a rendering subsystem for our visualization system. But we also have our own rendering subsystem that allows switching between different low-level graphical API. This is important when running on MS Windows, because of the choice between OpenGL and DirectX. Support of DirectX ensures high performance on MS Windows and compatibility with MS WinRT.

A lot of popular interactive visualization systems use native operating systems' tools to build graphical user interface (GUI), but in the case of multiplatform visualization system portable solution is needed. As shown in (Ryabinin & Chuprina, 2013), the most popular solution for building multiplatform GUI is Web application development. All the other technologies and tools have so-called “double design” problem. The developer has to design interface twice, once for desktop computer and once for mobile device. Web applications partially solve this problem due to the use of a set of high-level primitives independent from the concrete operating systems. However the Web applications themselves are not enough efficient for tackling scientific visualization challenges, especially on the mobile devices. While on the desktop WebGL is available as a tool to render complex 3D graphics, there is still no support of WebGL under iOS and very limited support under Android.

Since there is no procurable software for efficient and high-level building of multiplatform GUI we have developed our own GUI framework based on the same rendering engine as we use to draw the main scene. Our scientific visualization system SciVi is developed in co-working with Computer Science Department (Perm State University, Russia) and IT-Company Nulana LTD (Russia).

The details of our research’s background can be found in (Ryabinin & Chuprina, 2013). It describes how to eliminate the drawbacks of analyzed scientific visualization systems by developing the tools to adaptive distribution of rendering process, automated integration with different solvers and multiplatform portability.

3 SciVi Architecture

Server of SciVi is written in C++ and partially in Python, client of SciVi is written in C++ to ensure the multiprocessor portability. The desired platforms for the client are Windows, GNU/Linux, Mac OS X, iOS and Android. The server runs under GNU/Linux, but it can easily be ported to any other desktop or supercomputer operating system. C++ code can be executed directly under all the desired platforms except Android. Android uses Java Native Interface to execute C++ code from within Java applications. We have developed the lightweight shell for each platform to initialize the application using native operating system API and pass the control to the multiprocessor core.

3.1 Server Architecture

The stack of server's frameworks is presented in Figure 1.

![Figure 1: Stack of SciVi server's frameworks](image)
NFoundation framework provides all the necessary abstractions to work with processes, resources and the network for server and client building. The rendering of the intermediate data is achieved with the help of NGraphics and pVTK. pVTK is used in SciVi without any modification to provide efficient routines to visualize complex data in parallel. This library is distributed under the terms of BSD license like other libraries from VTK family.

On the top of the stack is the framework that is responsible for the logic of server. Its architecture is demonstrated in Figure 2.

The server needs the Solver XML Description (1) to manage visualization of scientific experiment results and to adapt to the specific features of the Solver (10). To generate this description the special module named Solver Description Generator (5) based on SciVi Ontology (4) is used. Web Interface (11) to the module (5) enables the user to register the corresponding solver in the special repository on the server-side. That means that solver’s metadata are stored and used in the unified way. The module (5) generates and stores the XML Description (1) in repository and returns the authentication code to access the given Solver (10).

After successful authentication of the client, the Solver (10) is started on the server-side and controlled by Solver Communication Module (7). Next to the solver’s finish, the module (7) obtains the corresponding Solver Output File (9) and transmits it to the Templater (6). Templater uses the Solver XML Description (1) to transform the output data to the Scene Description (2) acceptable for rendering by SciVi Client (12) as well as by Renderer (3). The partial rendering on the server-side is used if the client is not powerful enough. The Renderer (3) states the scene parts that are to be rendered on the server-side and on the client-side and perform the partial rendering. Moreover, it applies the necessary preprocessing to the data like simplification or splitting. To get an optimal decision, special heuristics about the performance of the client, the load of the server and the speed of the network connection are used.

All the prepared data are sent to the client via SciVi Client Communication Module (8). This module is used to communicate with clients for submitting and retrieving jobs. The server can communicate with the clients running on different platforms, such as iOS, Android, Windows, GNU/Linux and OS X.

![Figure 2: Architecture of the SciVi server](image-url)
3.2 Client Architecture

The stack of SciVi client's frameworks is presented in Figure 3.

![Stack of SciVi client's frameworks](image)

The basic framework called NFoundation provides an abstraction layer from the operating system. It is similar to Boost including all necessary abstractions to work with dynamic memory, resources, network, threads, callbacks and delegates, containers, mathematical functions and optimization algorithms.

The framework called NGraphics is used as a main rendering subsystem. It provides an abstraction layer from the graphics API and supports OpenGL, OpenGLES, DirectX9 and DirectX11 as the rendering backend. It is used to draw complex 3D scene as well as 2D elements of GUI. It supports vertex and fragment shaders to unify the visual effects applied to the different types of objects. Now it does not support geometry shaders because they are not supported on mobile devices yet, and the main goal of NGraphics is to stay completely portable. While OpenGL(ES) is not thread-safe, NGraphics includes the software mechanism to provide the transactional memory. It allows logging the changes of objects' properties from-within separate thread and applies them in the rendering thread just before the visualization of the next frame. This increases the performance by running rendering of the image and handling of the user interactions in different threads, because almost all modern devices have at least double-core CPUs and can execute two (or more) threads simultaneously. NGraphics also provides scene graph to build the scene hierarchy and all the necessary geometry entities like affine transform matrices, quaternions and bounding boxes.

The framework called NChart3D is used to draw charts and graphs. It is based on NGraphics and developed in Nulana LTD as a stand-alone library. It provides a lot of different 2D and 3D chart types that are suitable for scientific and analytical data visualization.

To solve the problem of GUI design tools we have developed our own framework called GUIBuilder. It parses the XML-description of GUI and automatically builds the interface appropriate for the platform where it is running. Our solution of double design problem is to describe the GUI with the set of high-level primitives like “button”, “slider” or “text label”, etc. The appearance as well as the interaction model of these primitives is obtained from the built-in database, because they differ on the desktop computers and mobile devices. We reuse NGraphics mentioned above to draw the GUI. There is no switching of graphics API and no compositing of images rendered by different APIs, thereby increasing of the system performance is achieved. GUIBuilder was successfully tested under Windows, GNU / Linux and Max OS X as well as under iOS and Android.

VTK/VES library is used to achieve the rendering of complex scenes using special techniques like slice-based rendering and rendering of volumes.

SciVi Client framework on top of the stack in Figure 3 is designed to implement the logic of the client. Its architecture is demonstrated in Figure 4.
Figure 4: Architecture of SciVi client

It has been developed according to the model-view-controller pattern. The Controller (5) is intended to manage the logic by using the data obtained from the communication modules (6, 7) and commands obtained from the GUI (2) through the GUI Manager (4). The controller transmits data and commands to the Scene Manager (3). The first action of client is the request of available servers’ list from the special Web Hosting (8) by means of the Web Hosting Communication Module (6). The user chooses the server he is interested in from this list. After that, the SciVi Server Communication Module (7) tries to access the chosen server, and if this server is available, the connection is established. The chosen SciVi Server (9) transmits the description of graphical user interface for its solver through the communication module (7). The Controller (5) transmits this description to the GUI Manager (4). The GUI Manager (4) builds the interface according to this description with help of GUIBuilder tools. The user controls the solver by means of the generated GUI interface and the commands are transmitted to the solver via the server. The data generated by the solver are processed by the server automatically and transmitted to the client. The Controller (5) passes them to the Scene Manager (3) and after all the necessary preparations lets Renderer (1) visualize the final image. After the scene is visualized, the user can interact with it through the GUI of client. User’s commands are obtained with the GUI Manager (4) and transmitted to the Scene Manager (3) to achieve interactivity.

4 Adaptation to different solvers

We present an ontology-based approach to development of the adaptive multiplatform rendering tools to visualize scientific experiments so the current restrictions can be eliminated in the process of development and improvement of the ontology. Due to SciVi ontology-based capability to generate automatically the description of input/output files format and structure of files have no restrictions. There are no restrictions for solver’s programming language. Of course, it is not necessary to do any changes in solver’s input/output files and source code.

The following requirements must be fulfilled for the logic of the solver:
- Solver should have single input and single output file.
- All objects, which characteristics are evaluated in the process of scientific experiment should be the objects of the same type.
The process of visualization starts only after the solver’s finish according to the concrete number of objects and keyframes. There are a lot of problems to integrate scientific visualization systems with various solvers. Actually different solvers have various formats of input data, control commands and outputs. That is why very often the visualizers are written from scratch for each solver manually.

To adapt visualization systems to specific of concrete solver we offer quick and easy adaptation tools based on domain-specific ontologies. The set of XML-based descriptions including the specification of solver's input and output for each solver are generated automatically. Graphical user interface for solver is also generated automatically in the notation suitable for GUIBuilder framework.

The output description is a template of the scene in the notation of SciVi scene manager. It specifies how objects look that represents the results of the solver’s work. And the actual visual parameters of the objects (for example, position, rotation, color, etc.) are filled at runtime automatically from solver’s output data.

For adaptation for solver’s specific we have designed two ontologies:
1. Ontology, which contains the knowledge about syntax of input/output statements to extract the descriptions of input and output variables from source code in Fortran and C (it seems to us, that these languages are the most popular for solvers’ implementation).
2. The ontology of typical visual objects that can represent the results of solver’s work.

We developed a special tool called SciVi Solver Description Generator that helps to automate creation the solver's descriptions and integration of SciVi with third-party solvers. If the source code of the solver is available, SciVi parses it according to the ontology (1) and shows all the extracted declarations of input/output variables. User can mark the required parameters and choose the corresponding visual objects for the output (see Figure 5).

In the future we plan to extend the ontologies to take into account features of other programming languages.

5 Adaptive anti-aliasing

A lot of the modern graphics accelerators support built-in anti-aliasing. Very often (for example, under iOS and Android) anti-aliasing technique is implemented as multisampling (MSAA) (3D Center, 2003). However built-in anti-aliasing has some problems:
1. Significantly lowers the performance (we experienced the decrease of the frame rate in 3 times on the mobile devices).
2. Can produce unwanted artifacts on the image.
3. Depends on the platform and thereby offends the portability.
For example, to test the performance and quality of visualization we used the scene that contains approximately 800,000 vertices connected in 1,600,000 triangles rendered with the simple Phong's shading model and occupying the whole screen. The results of this scene testing (with and without anti-aliasing) are demonstrated in the Figure 6. iPad 3 is able to render this scene 15 times per second, which is nearly acceptable to ensure good response to the user's interactions and to play smooth animations. But test with the build-in MSAA lowers the performance down to 5 FPS. Such a frame rate is not acceptable anymore, because the movements become jagged and the user feels the significant lag between his actions and the responses of the system.

The obvious solution of the first problem is to switch off the anti-aliasing for the periods when the scene is in motion. This solution has almost no negative effect on the visual quality, because the aliasing is less noticeable when the objects are moving. However it takes up to 1,000 milliseconds to toggle the system anti-aliasing, because we have to change the rendering context and to synchronize the rendering thread with the main thread where the user's interactions are handled. Such a huge lag ruins the smoothness of motions completely.

The second problem arises because the system anti-aliasing affects the entire scene. This means, each object on the scene is anti-aliased. However some objects require no anti-aliasing. For example, the vertical and horizontal grid lines should not be anti-aliased because it is aligned to pixels and any attempt of smoothing its edges will end up in losing the sharpness and thereby the visual quality.

The third problem is the obvious necessity to extract the code responsible for the enabling anti-aliasing to the platform-dependent part of the project. It will increase the cost of program development in case of porting the program system to other platforms.

To solve all the problems mentioned above we decided to implement custom anti-aliasing. There are many algorithms of anti-aliasing based on the different approaches. The main set of approaches includes the algorithms applied while rendering the scene; the post-processing algorithms and the algorithms that use multiple rendering passes.

As usual, these algorithms are used separately, but we propose to use the superposition of the supersampling (SSAA) (Lizandra, 2000) and fast approximate anti-aliasing (FXAA) (Lottes, 2009). These two algorithms have sufficient performance. Also they are compatible with mobile devices (preserving portability) and their combination ensures high visual quality of the result image.

We proved that the theoretical average-case complexity of the superposition FXAA and SSAA is equal to the complexity of SSAA. According to our estimation it is $O(n + (s^2 + 1) \cdot w \cdot h)$, where $n$ is the number of vertices on the scene, $s$ is the ratio of supersampling, $w$ is the width and $h$ is the height of the result image (that should be presented to the user). In practice the performance on the test scene (1,600,000 triangles) is 4.5 FPS on the iPad 3. The result of the superposition is demonstrated in the Figure 6 (b).
The suggested approach to anti-aliasing is adaptive to the hardware and to the scene changes. We optimized the FXAA for the usage on the mobile devices so, that its performance increases from 8 to 10 FPS. We also improved SSAA so, that the supersampling ratio is calculated dynamically according to the characteristics of the hardware. The anti-aliasing itself is automatically switched off when the user interacts with the scene or animation is played. The toggling of antialiasing is as fast as the rendering of the single frame. So we achieved the response within 60 milliseconds on our test scene (1 600 000 triangles).

The suggested approach is also adaptable, because the programmer can combine objects with and without anti-aliasing on the same scene, eliminating the artifacts of unwanted blur. For this we implemented the layered rendering. It means that the scene consists of an array of layers and each layer is a group of objects. The layers are rendered one by one and each of them can either be anti-aliased or not.

6 Testing of SciVi

We tested our system on three scientific problems representing different fields of science: physics, economy and genomics. The first problem is about modeling the magnetic moments of nanoparticles in the magnetic field. The third-party solver for this problem is called MagnetoDynamics-F (Demenev, Belozerova, Kharebov, & Khenner, 2012) and is implemented in Fortran. The input data for this program describe the characteristics of magnetic field, the structure of the grid containing nanoparticles and the amount of time quants. The output file contains the array of 3D orientation vectors, which represent magnetic moments of nanoparticles in each time quant. All the modeling results including the states of each particle in each moment of time are stored in the scene description and visualized on the client with animated rotation. The result of visualization is presented in Figure 7 (a).

The second problem is the visualization of the data from Bitcoin stock market BTC-e. The solver in this case is implemented as Java application that connects to BTC-e and requests the necessary data. The input data describe the period of time. The output data are the opening and closing trades and highest and lowest traded prices of the cryptocurrency over the given period. To visualize the price changed, the candlestick chart is used as demonstrated in Figure 7 (b).

The third problem came from genomics and is about the alignment of genetic sequences. The solver used for this problem is called Clustal (Larkin, et al., 2007). The input data are the genetic sequences of bacteria that are obtained from the hardware sequencer in the Institute of Ecology and Genetics of Microorganisms (Perm, Russia). They are represented as array of characters, where each character corresponds to the particular nitrogenous base. The output data are the aligned sequences and the description of phylogenetic tree (the representation of inferred relationships among the sequences). The visualization of the aligned sequences and phylogenetic tree is shown in Figure 7 (c, d).

![Image](a.png) ![Image](b.png) ![Image](c.png) ![Image](d.png)

Figure 7: SciVi visualization: magnetic moments of nanoparticles (a), candlestick representing the changing of prices on the stock market (b), genetic sequences (c) and phylogenetic tree (d)
As the results of tests conducted we can conclude that SciVi is adequate for multidisciplinary collaboration projects. For example, it can help scientists to present their research results in the way that enhance the complex understanding to discuss, interpret and verify experiments. Also SciVi can be used in learning as a part of pedagogical environment to achieve complex higher-order skills.

7 Conclusion

Building scientific rendering systems adequate to the specific visualization goals of the experiments in different fields of science is a huge problem. In this paper, we have proposed the ontology-based approach to be used to automate the integration scientific visualization systems with different solvers. We have demonstrated our approach by implementing a system called SciVi.

The rendering process in SciVi is adaptively distributed between client and server. This enables high interactivity and optimal load of the computing system at the same time. Thanks to this SciVi is not only adaptable but also adaptive to the performance of the client, speed of the connection and load of the server. The high quality of image is ensured by using custom adaptive anti-aliasing algorithm that enables to smooth the edges of the objects without offending the speed of animations and responsiveness of the system. The multiphase performance portability of the system allows it to run on the desktop computers as well as on the mobile devices.

In our opinion, the distinctive features of the system designed are automated integration with different solvers, multiphase portability, high interactivity ensured by adaptive distribution of the rendering process between client and server, high visual quality of result image ensured by the adaptive anti-aliasing algorithm.

In the future, we plan to extend the SciVi ontology of programming languages, to enhance the heuristics used for distribution of rendering, to implement new data prepossessing algorithms on the server-side and to add new types of visualization like multidimensional rendering.

References


