

9-1-2006

# Hub-based Simulation and Graphics Hardware Accelerated Visualization for Nanotechnology Applications

Wei Qiao

Michael McLennan  
*Purdue University - Main Campus*

Rick Kennell  
*Purdue University - Main Campus*

David S. Ebert  
*Purdue University - Main Campus*

Gerhard Klimeck  
*Purdue University - Main Campus, gekco@purdue.edu*

Follow this and additional works at: <http://docs.lib.purdue.edu/nanodocs>

---

Qiao, Wei; McLennan, Michael; Kennell, Rick; Ebert, David S.; and Klimeck, Gerhard, "Hub-based Simulation and Graphics Hardware Accelerated Visualization for Nanotechnology Applications" (2006). *Other Nanotechnology Publications*. Paper 94. <http://docs.lib.purdue.edu/nanodocs/94>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact [epubs@purdue.edu](mailto:epubs@purdue.edu) for additional information.

# Hub-based Simulation and Graphics Hardware Accelerated Visualization for Nanotechnology Applications

Wei Qiao, *Student Member, IEEE*, Michael McLennan, *Member, IEEE*, Rick Kennell, *Member, IEEE*, David S. Ebert, *Senior Member, IEEE*, and Gerhard Klimeck, *Senior Member, IEEE*

**Abstract**—The Network for Computational Nanotechnology (NCN) has developed a science gateway at nanoHUB.org for nanotechnology education and research. Remote users can browse through online seminars and courses, and launch sophisticated nanotechnology simulation tools, all within their web browser. Simulations are supported by a middleware that can route complex jobs to grid supercomputing resources. But what is truly unique about the middleware is the way that it uses hardware accelerated graphics to support both problem setup and result visualization. This paper describes the design and integration of a remote visualization framework into the nanoHUB for interactive visual analytics of nanotechnology simulations. Our services flexibly handle a variety of nanoscience simulations, render them utilizing graphics hardware acceleration in a scalable manner, and deliver them seamlessly through the middleware to the user. Rendering is done only on-demand, as needed, so each graphics hardware unit can simultaneously support many user sessions. Additionally, a novel node distribution scheme further improves our system's scalability. Our approach is not only efficient but also cost-effective. Only a half-dozen render nodes are anticipated to support hundreds of active tool sessions on the nanoHUB. Moreover, this architecture and visual analytics environment provides capabilities that can serve many areas of scientific simulation and analysis beyond nanotechnology with its ability to interactively analyze and visualize multivariate scalar and vector fields.

**Index Terms**—remote visualization, volume visualization, flow visualization, graphics hardware, nanotechnology simulation.

## 1 Introduction

Nanoscience is the study of matter at the scale of a nanometer—one billionth of a meter, the scale of atoms and small molecules. Structures and materials at such scale often exhibit unique phenomena and properties fundamentally different from that of macroscopic structures. These properties can be harnessed to make novel devices. Such work is often interdisciplinary in nature, involving molecular biologists, electrical engineers, computer scientists, and others to simulate, optimize, and engineer a working nanoscale device.

To address these challenges, the National Science Foundation (NSF) has formed the Network for Computational Nanotechnology (NCN), a network of eight universities with expertise in nanoelectronics, nanoelectromechanical systems (NEMS), and nanomedical devices. The NCN has created a science gateway for nanotechnology exploration at nanoHUB.org. This web site contains a large collection of online seminars, web-based courses, animations, and other educational materials. All of these resources are coupled with interactive simulations that users can access from any web browser. Its supporting middleware can route simulation jobs to supercomputing resources, including the NSF TeraGrid and the Open Science Grid.

The nanoHUB is becoming a national resource for the nanotechnology community. Last year, more than 10,000 users viewed online seminars, courses, animations and publications related to nanoscience. Among those users, over 1,800 ran more than 54,000 simulation jobs, consuming over 28,500 solid hours of CPU time.

Acquiring simulation data is only part of scientific study. Visualization and data analysis serve as the critical pathways to insights and discovery. Providing scientists with matching computation and visualization power is an essential part of the nanoHUB effort. As the

facility grew, its visualization capability became the weakest link, primarily due to the lack of state of the art visualization systems that utilize graphics hardware acceleration. In this work, we report our progress in designing and integrating such visualization systems into the nanoHUB architecture, and we show their utility and performance.

## 2 System Requirements

Assisting remote users who may not be equipped with sufficient computational resources has become an important and practical aspect of scientific visualization. Such tasks become significantly more difficult for facilities like nanoHUB.org, where visualization systems serve a large number of users whose hardware and software capabilities vary widely, to the extent that only minimal assumptions can be made about their computing profiles. We have identified the following important requirements that direct the design and implementation of our framework:

1. **Transparency:** The delivery of hardware acceleration should be seamless and transparent to remote users. The requirements on the user's system should be minimal.
2. **Scalability:** Our framework should support many remote user sessions simultaneously. The performance should scale under an increased rendering load as hardware resources are added.
3. **Responsiveness:** The system should be responsive to user commands even when the number of open visualization sessions is large.
4. **Flexibility:** The rendering engine should be flexible enough to handle a variety of simulation data, providing capabilities of quickly developing and deploying new tools.
5. **Extensibility:** The system software and hardware architecture should be easily extensible, to satisfy new visualization needs for future applications.

## 3 Related Work

Most nanoelectronics simulations generate scalar volumes, vector fields or a combination of the two. To deliver interactive visual analytics of a variety of nanoelectronics simulations to remote scientists, our work involves techniques related to molecular dynamics visualization, flow visualization and remote visualization. In this section we will review relevant work separately.

- Wei Qiao is with Purdue University, E-mail: qiaow@purdue.edu.
- Michael McLennan is with Purdue University, E-mail: mmclennan@purdue.edu.
- Rick Kennell is with Purdue University, E-mail: kennell@purdue.edu.
- David S. Ebert is with Purdue University, E-mail: ebertd@purdue.edu.
- Gerhard Klimeck is with Purdue University, E-mail: gekco@purdue.edu.

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.

For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

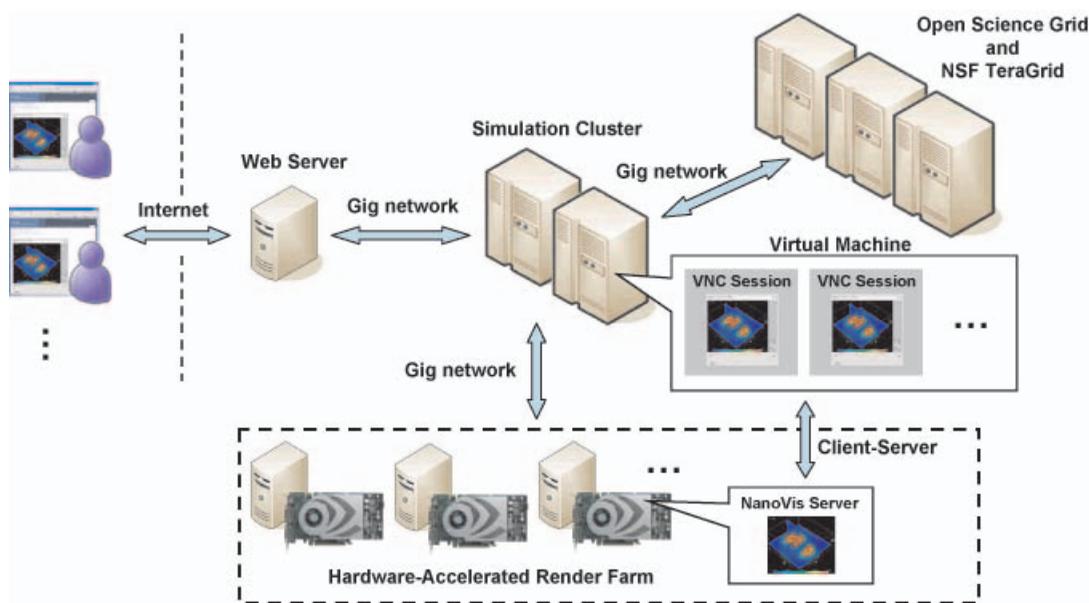


Fig. 1. nanoHUB architecture

### 3.1 Molecular Dynamics Visualization

Nanoscale devices frequently consist of molecular structures, and their models often compute scalar fields of molecular dynamics, e.g. electron density, potential and charge density, etc. Published molecular visualization work mainly focuses on two aspects of molecular dynamics: structural information and field properties.

For structural information, most published visualization techniques display molecular assemblies with two types of representations: structural rendering and surface rendering. Structural rendering emphasizes the underlying skeletal construction of the molecule, and often displays models using primitives like spheres for atoms and cylinders for bonds (Ball-and-Stick model). Surface rendering mainly presents molecules as space-filling surfaces defined by the van der Waal radii of the atoms or isosurfaces of molecular dynamics properties (e.g., electronic potential). Surface rendering has been adopted by many areas of study, for instance protein-folding [5] and drug design [27]. Visualization platforms like Visual Molecular Dynamics (VMD) [15] and Protein Explorer [32] are examples that handle both structural and surface rendering.

Volume visualization is quite suitable for rendering scalar dynamics fields. Among volume visualization techniques a popular class is the texture-based volume rendering approach [7, 47]. Additionally, cell projection approaches are often used for visualizing unstructured meshes (e.g. [41, 38]). Many nanoHUB simulations are computed on the Cartesian lattice, or an alternative regular lattice called Face Centered Cubic grid (FCC) [36]. Thus, for scalar visualization, we choose to base our system on texture-based volume rendering. In addition, a convenient and efficient method is available to visualize FCC grid using a set of staggered Cartesian grids [36].

### 3.2 Flow Visualization

Vector flow field visualization has been an active research area in recent years. Popular techniques can usually be categorized into two classes: texture synthesis based and particle tracing. Early work in texture synthesis [8, 45] has led to various extensions [33, 19] including GPU-accelerated approaches [12, 18, 46, 43]. Texture synthesis techniques are very effective in visualizing 2D vector flow. However, their effectiveness is reduced substantially when visualizing 3D flow fields due to visual cluttering. On the other hand, particle tracing techniques in essence utilize numerical integration schemes to advect particles through the vector field [39]. Recently, particle engines have also been implemented on the GPU where the integration is computed

using fragment shaders [24, 26]. Besides point primitives, particle tracing techniques are also able to generate stream ribbons and tubes [14, 6, 44].

### 3.3 Remote Visualization

Remote visualization systems are often preferable when the data involved is excessively large to transmit over the network [11], or such size is unsuitable to visualize using local workstation and cluster or super computing resources are utilized to accelerate rendering [16, 1, 30, 35, 9, 20]. Remote visualization is also helpful in bringing graphics hardware resources to remote users [29, 10, 42, 2] and facilitating distance collaboration [13, 31]. The motivation of this work incorporates all of the above aspects. Among these systems, VirtualGL [2] and its predecessor [42] intercept and reroute GLX commands to an X display with hardware-accelerated graphics. The advantage of such an approach is that it requires no modification to the original program and achieves excellent transparency. However, due to the architecture of the nanoHUB, we could not take advantage of VirtualGL (see Section 4.2). In this work, we strive to achieve such transparency using an extremely cost effective approach specifically suited to the nanoHUB software and hardware architecture.

## 4 nanoHUB Architecture

### 4.1 Web-based Interface

A large part of the nanoHUB popularity comes from a focus on ease-of-use. nanoHUB users can launch a simulation simply by clicking a button within their web browser. This brings up an interactive graphical interface, such as the one shown in Figure 2. This interface appears to be a Java applet running within the browser, but the implementation mechanism behind the scene is much more robust with both simulation and rendering conducted on remote servers. When a simulation is launched, nanoHUB allocates a session on a virtual machine running on a supporting cluster, then sends the image of that session back to the viewer inside the user's web browser via the Virtual Network Computing (VNC) paradigm [37]. The user sees an interactive image of the session which runs remotely on nanoHUB hardware, as shown in Figure 1. This unique architecture has a few important advantages. First, because the simulation is hosted on nanoHUB hardware, the state of the simulation is maintained even when the user connection is lost. A user can simply reconnect using web browser to resume his previous simulation session. Second, each session can transparently launch simulations on a vast array of hardware, to which most users would

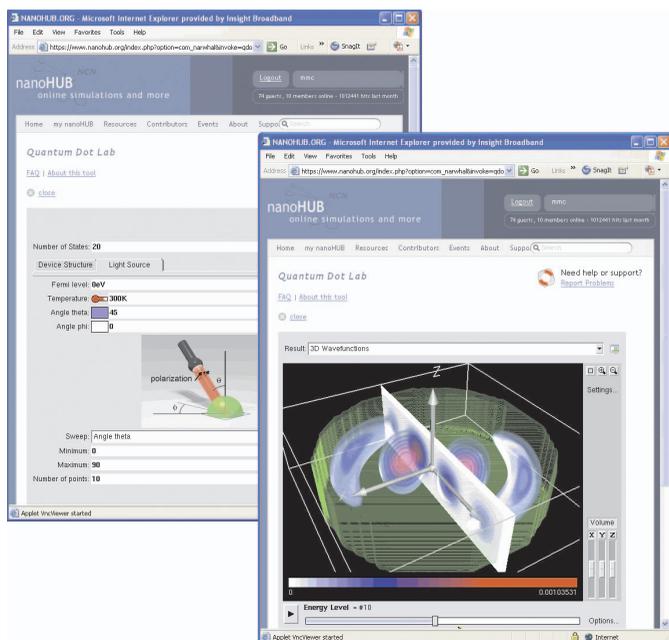


Fig. 2. nanoHUB web-based interface: simulation (back) and visualization (front).

not ordinarily have access. For example, complex jobs can be sent off to other compute clusters on the NCN network, or to other national grids, including the Open Science Grid and the NSF TeraGrid.

## 4.2 Visualization Challenges

There is, however, one disadvantage to the nanoHUB architecture: The graphical environment for each tool runs within a VNC session on cluster nodes that have no graphics hardware acceleration. As a result, visualization of non-trivial simulations (such as the 3D volume shown in Figure 2) can be very slow. Additionally, the VNC server nodes are rack mounted machines with neither AGP nor PCI Express interfaces to install graphics hardware. Even if graphics cards were installed on the simulation nodes, they would not be directly accessible through nanoHUB's virtual machine layer. A VirtualGL-based solution would require another VNC connection to the machines with hardware acceleration. A VNC-inside-VNC approach is conceptually confusing and difficult to implement, and has prevented us from taking advantage of VirtualGL.

The use of virtual machines is an important element in the nanoHUB middleware. Virtual machines can be configured independently of the underlying physical machines, so they can be tuned to provide extra security preventing users walled off within one machine from attacking another. They can also authenticate users against a customized Lightweight Directory Access Protocol (LDAP), so nanoHUB users can be managed separately from users at the institutions supporting nanoHUB. The design and delivery of hardware accelerated remote visualization in the nanoHUB environment should take its unique architecture into consideration and not compromise the advantages mentioned in Section 4.1.

## 5 Hardware-accelerated Remote Visualization

Our strategy for nanoHUB remote visualization is a client-server architecture. We have created nanoVIS, a visualization engine library that can flexibly handle a variety of nanoscience simulations involving vector flows and multivariate scalar fields. Acting as the server end of the remote visualization, nanoVIS services run on a Linux cluster equipped with hardware acceleration. The render engine has been integrated to NCN's Rappture Toolkit (see Section 5.2.1). The development and deployment of new visualization tools becomes an extremely quick process.

### 5.1 Hardware

Our render farm is composed of a Linux cluster equipped with nVIDIA Geforce 7800GT graphics acceleration. Because our visualization tasks are inherently GPU intensive, CPU speed is not a dominant factor. Thus, modestly configured machines are sufficient, making such a cluster extremely economical to build. Our render nodes are 1.6GHz Pentium 4 with 512MB of RAM running RedHat Linux. The render farm is directly connected with the VNC server cluster. To reduce the network latency that can delay client-server interaction, our render farm is also physically placed in close proximity to the VNC servers. All interconnects in our setup are fast gigabit network as shown the lower portion of Figure 1.

Utilizing commodity cluster and consumer graphics hardware for render services has several advantages. First, such an approach is extremely cost effective due to the high performance/price ratio offered by modern consumer graphics hardware. As shown in Section 6.3, only a few render nodes are required to serve many remote users. Second, a Linux cluster is flexible to upgrade and expand as needs grow. Our rendering software also facilitates such scalability. Third and foremost, a client-server approach running on cluster PCs integrates tightly into the nanoHUB existing architecture, and such a strategy fits the philosophy of hub-based computing well.

### 5.2 Software

The nanoVIS library and necessary client-server network modules have become an integral part of the Rappture toolkit. This integration has made the development of remote simulation and visualization tools a largely automated process, where code for the GUI client, the server and client-server communication is generated from the user's description of the input parameters and output formats. Thus, the deployment of new tools is greatly expedited.

#### 5.2.1 Rappture Toolkit

As part of the nanoHUB development, the NCN has created the Rappture Toolkit [3], the Rapid Application Infrastructure Toolkit, which provides the basic infrastructure for a large class of scientific applications, accelerating the deployment of new tools. Rappture is available as open source, and it includes language bindings for C/C++, Fortran, Matlab, Python, and Tcl, so it is easily integrated into various simulation codes. Instead of inventing their own input/output interfaces, researchers declare the parameters associated with their simulator by describing Rappture objects stored in an Extensible Markup Language (XML) format. Rappture has a variety of input/output objects, ranging from simple elements (such as numbers, choices, Booleans, and text entries) to more complex elements (such as molecules, curves, meshes, and scalar/vector fields). Once a researcher has defined the interface for a tool, Rappture reads the interface description and generates the graphical user interface (GUI) automatically. The tool shown in Figure 2 is an example of graphical interface generated automatically by Rappture.

#### 5.2.2 nanoVIS

The nanoVIS library, as part of the Rappture Toolkit is also open source. The render engine takes advantage of graphics hardware acceleration and can flexibly handle a variety of nanotechnology simulations computed on Cartesian and FCC grids. For multivariate scalar fields, we choose to base our system on texture-based volume rendering and adopt a multi transfer function approach [36] to simultaneously visualize various electromagnetic properties. The isosurfaces of these dynamics fields are also visualized using transfer functions without explicitly generating geometry. In addition, nanoVIS also supports textured cutting planes and geometric primitives which are primarily used to illustrate the simulation geometry.

For vector field visualization, our engine implements a completely GPU-accelerated particle system similar to [24] and [26]. The particle position is governed by the ordinary differential equation (ODE):

$$\frac{\partial \vec{x}}{\partial t} = \vec{v}(\vec{x}(t), t) \quad (1)$$

Where  $\vec{x}(t)$  is the particle position at time  $t$ , and particle tracing amounts to numerically solving Equation 1. Similar to [26], we implemented particle advection with fragment shaders using a Eulerian integration scheme:

$$\vec{x}_{n+1} = \vec{x}_n + (t_{n+1} - t_n) \cdot \vec{x}_n \quad (2)$$

The Framebuffer Object (FBO) feature of OpenGL 2.0 [40] greatly assisted our implementation. In our implementation, two off-screen FBOs with floating point texture attachments act as source and destination memory. During time step  $n$  an FBO storing particle information (position and lifetime) of the previous time step  $n - 1$  acts as the input texture, and the other FBO acts as the output target to which the updated particles are written to using a fragment shader. We then bind the output target FBO as a Vertex Buffer Object (VBO) to render opaque particles. In the next time step  $n + 1$ , FBOs are flipped in terms of source and target. Such an implementation requires practically no CPU computation. Particles always stay in the graphics memory and are never transferred through the system bus. Additionally, 2D Line Integral Convolution (LIC) is also integrated into our system to complement the 3D particle advection and illustrate per slice flow field.

### 5.2.3 Client-Server Interaction

The client side of a simulation and visualization application is composed of a Rappture GUI (see Section 5.2.1) running within a virtual machine. This GUI drives the whole interaction of the tool on nanoHUB as shown in Figure 1. The remote user interacts with the GUI, entering values to set up the problem, and eventually presses the Simulate button. At that point, Rappture substitutes the current value for each input parameter into the XML description, and launches the simulator with this XML description as a driver file. The simulator computes and writes results to files. In scenarios where hardware-accelerated rendering is not required, Rappture loads the results into the output analyzer for the user to explore.

As soon as the user selects a simulation result or mode that requires 3D graphics, the Rappture client connects to one of the rendering nodes selected using a node selection algorithm (see Section 6.2). We created a special Linux daemon, nanoSCALE, to monitor a dedicated network port for client communication. When a client connects the communication port, nanoSCALE spawns an instance of our nanoVIS server. Once the socket connection is established, the client and server will synchronize through a network message passing interface.

As an initial setup step, the client controlling interface sends over one or more scalar and/or vector fields, along with a default camera position. The visualization server loads the data into graphics memory, and then generates and sends back the desired image at the specified camera position. The image coming back from the visualization server is loaded into the GUI running within the VNC session. To save network bandwidth, we utilize a simple run-length encoding method to compress the images. Such a compression scheme is inexpensive to compute using our modest CPUs and works fairly well, since the visualization result typically contains regions of black space. VNC then transmits the screen change to the remote user. To conserve GPU computation resources, the nanoVIS server is designed to quickly deliver rendering upon request and wait for further instructions instead of actively looping and generating the same frame. In the case of particle animation, 20 frames of particle renderings are quickly generated and transmitted to the client side. The interface then plays back this animation without requesting further service. Thus, in our architecture, as long as the user is still, the visualization server is sleeping. This behavior is very important in a hub-based simulation environment, since we need to support not one, but many hundreds, or perhaps someday thousands, of simulation sessions at any given time.

Various communication commands have been defined in our framework, such as: transfer function, particle seeding, cut plane, zoom, and rotate. Each of these actions will trigger a request to the nanoVIS server, which then updates the view and sends back another image. We use a simple protocol parsed by Tcl interpreters on both ends for most of the communication. For example, the camera can be adjusted by a command such as the following: camera 0.0 45.0 0.0, where the

three parameters represent the Euler angles of the camera. Meshes and fields are defined by a similar Tcl-based command, but the actual data is sent and received in binary format, as produced by objects in the Rappture C++ library. So the transmission of control messages is simple and human-readable, but the transmission of data (meshes, fields, and the resulting image) is handled more efficiently. This makes the system easy to debug, while at the same time providing the throughput needed for large datasets.

## 6 Performance and Optimization

As an integral component of the nanoHUB multiuser environment, it is important that the visualization facility should support many user rendering sessions simultaneously. Our nanoVIS engine makes extensive use of GPU acceleration, whereas the CPU is only responsible for network communication and rendering setup (see Section 5.2.3). Thus, the CPU workload is very light compared to that of the GPU. This aspect has allowed us to optimize rendering performance by choosing render hosts based on the GPU workload only.

### 6.1 GPU Load Estimation

The DirectX9 SDK [4] exposes various useful graphics hardware performance queries including pipeline timings, pixel timings, etc. The nVIDIA NVPerfKit and NVPerfHUD are also powerful tools to access low-level performance counters. However, these queries and tools are not available on Linux. Therefore, we developed a fairly straightforward GPU workload estimation model for our system based on test data. This model relies on the fact that GPU fragment processing is the dominating factor of performance in our system, and its cost is determined by the number of rasterized fragments and how much computation is required per fragment. However, a complication is the integration of a particle system. Thus, a unified workload measurement is needed.

It is, in general, difficult to compare the GPU workload of fixed-pipeline geometry rendering (particles and simulation geometry) with programmable per-pixel shading (volume visualization). The geometric elements of simulations typically include a few simple primitives for illustration purpose, for instance, computation domain bounding boxes. These primitives are trivial to render compared to the massive amount of particles in our flow visualization. Thus, we only consider the cost of particle rendering in our model. Additionally, our tests have shown that the time required to render the particles is only about a factor of 0.2 of what is required to advect them due to the efficient use of render to vertex array. This enables us to express the GPU workload of the particle system using only the fragment processing cost of the advection step. Now we can develop a unified cost model since the advection shader is comparable to a texture-based volume visualization shader in the way it samples a vector field.

Our model reflects the fact that the primary cost of the shader execution is texture access, where a particle update requires one texture fetch and a multivariate scalar field may need a few texture samples per fragment. We can query the number of rasterized fragments using the OpenGL ARB\_occlusion\_query extension [21]. The workload is computed at each time interval of  $t$ . For example, there have been  $s$  nanoVIS servers on render host  $i$  in the  $n$ th time interval. We define the GPU workload  $L_i^n$  of this interval as a function of the total number of rasterized fragments  $f$ , the number of visualized scalar fields  $m$ , and the number of advected particles  $p$  (0, if the particle system is disabled):

$$L_i^n = \sum_{j=1}^s (2 * \text{ceil}(\frac{m_j}{4}) * f_j + (1 + \alpha) * p_j) \quad (3)$$

where  $\alpha$  is tunable parameter denoting the cost of particle rendering in relation to advection. To minimize the number of texture fetches, we pack 4 scalar fields into one volume texture. Thus the number of texture accesses for sampling a multivariate volume of  $m$  fields is  $\text{ceil}(\frac{m}{4})$ . The factor 2 is due to a transfer function table lookup following each scalar sampling.

## 6.2 Node Selection

In addition to the estimation model, GPU memory usage is also factored in to prevent graphics memory thrashing. Thus, the best render node choice is one with the least amount of GPU workload that can fit the data sets of the new client. The nanoSCALE daemon resides on each render node and is responsible for monitoring estimated local GPU workload, tracking local graphics memory usage and starting new render services.

The nanoSCALE daemon communicates with the started render services through Linux pipes. Whenever a render server receives a render request, it estimates the GPU workload of the current request and pushes the estimate to the local nanoSCALE daemon through its pipe. Additionally, to include a historical bias in the load estimates, the workload at interval  $w$  is a decayed sum of the workloads of the previous  $n$  intervals with a decay factor of  $\gamma$ , where  $n$  and  $\gamma$  are tunable parameters:

$$L_i = \sum_{k=w-n+1}^w \gamma^{(w-k)} * L_i^k \quad (4)$$

The nanoSCALE daemon broadcasts this decayed sum to all peer render nodes. Since the per node workload statistics are made aware to all render nodes, a new client can contact a random render node in the initial step. The initial host subsequently chooses a host with the lightest workload and enough graphics memory to start the render service. The initial host itself is also eligible in this selection. The initial host then updates its record of the target host's load average factoring in this new job. Such an approach keeps the initial host from redirecting to the same target when clients rapidly connect before the next time interval expires. When the target host finally broadcasts its most recent workload, its record at the initial host is updated.

And finally, a load redirection threshold factor  $\epsilon$  is defined to further guide redirection decisions. The initial host with workload average  $L_{initial}$  only redirects a task to a target host with workload average  $L_{target}$ , if:

$$L_{target} < \epsilon * L_{initial}$$

This is needed due to the lag time in the broadcast of workload updates. Different render nodes may have different ideas about the lowest workload. The parameter  $\epsilon$  can also be tuned to achieve more accurate redirection. In theory, a client could cycle through several redirections until the load records stabilizes enough for it to settle on a host. However, such an approach would involve many iterations of network communication.

## 6.3 Performance

To measure the performance and scalability of our system, we recorded real user interaction sessions visualizing a 128x128x128 scalar field using the nanoVIS engine in a 512x512 window. The user interaction includes rotation, zoom, transfer function modification, movement of cutting planes and change of lighting/material parameters, each of which causes the engine to re-render the data. A typical user generates bursts of events, where sets of closely spaced events are interlaced with periods of inactivity when the user examines the rendering result. On average, a user initiates less than 5 events per second.

Based on recorded events, we then simulated a series of simultaneous user visualization sessions. These sessions were started using our node selection scheme in Section 6.2. Our performance metric is the turnaround time, from the time the client issues a command to the time an updated image is received. We measured it on 1, 3 and 5 render nodes to demonstrate how the performance scales as the number of simultaneous render sessions increases. The timing does not include the simulation time, since such computation is not part of the rendering cost. Additionally, our tests are conducted on the nanoHUB production network with competing traffic from users in the same subnet. Thus, the test results are a realistic reflection of our system's performance in the production environment.

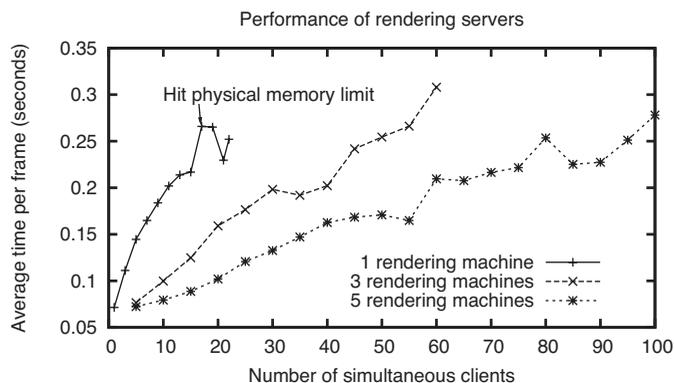


Fig. 3. Frame times: one, three and five render nodes.

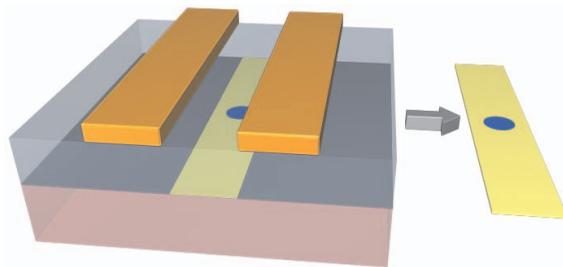


Fig. 4. Electron gas simulation. Electrodes are positioned on the top. A narrow channel constraining the electrons is between the top GaAs and bottom AlGaAs layers. An impurity is planted in the middle of the channel to split the electron flow.

As shown in Figure 3, the frame times show a favorable sub-linear scaling instead of linear scaling as the number of render sessions increases. This is due to the inherent randomness in the distribution of the user events and their spacing. In other words, the clients rarely request image updates all at the same time. As described in Section 5.2.3, our nanoVIS server is designed to deliver upon request. Thus, the GPU can service the active clients while the others are idle. Such a strategy fits the nanoHUB particularly well, since a typical user will interact with the data generating a series of requests, then examine the results, leaving the client idle for a period of time. As demonstrated in Figure 3.b, with only 5 render nodes, our system is able to support 100 active sessions at more than 3 frames per second. With a more casual user base we expect to support a much larger number of sessions.

Our tests also revealed that when the number of clients exceeds 17 on a single node, the 512MB of system memory becomes insufficient and swapping occurs. Thus, the performance become less predictable. However, such problem can be easily solved by installing more system memory. Additionally, even under the severe system memory stress, the performance still degrades gracefully.

## 7 Case Studies

Utilizing our remote simulation and visualization framework, developers successfully created several new nanotechnology tools, SQUALID-2D, Quantum Dot Lab, BioMOCA and Nanowire. More than 1100 user simulation and visualization sessions have been conducted within the very first two weeks of their deployment at nanoHUB.org. In this section we introduce three of these tools. More existing tools are being built to take advantage of nanoVIS graphics hardware acceleration.

### 7.1 2-D Electron Gas Simulator

A 2-dimensional electron gas (2DEG) is created by trapping electrons in a very thin quantum well at the boundary between GaAs and  $\text{Al}_x\text{Ga}_{1-x}\text{As}$  layers in a semiconductor heterostructure. Quantum de-

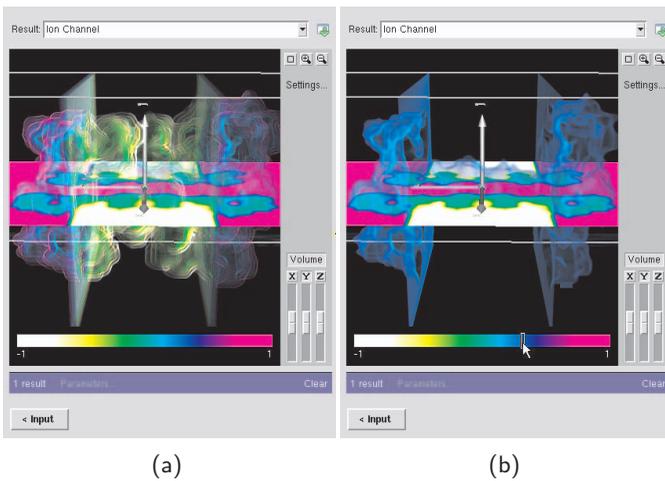


Fig. 5. Ion flow simulation through a pore in a cell membrane: (a) default view, (b) with a single isosurface.

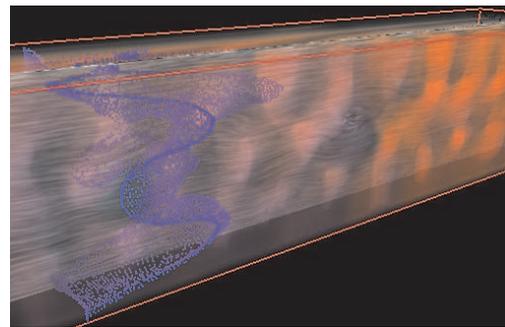
vices can be created by further constraining electrons in the remaining two dimensions. For instance, by adding electrodes on top of the 2DEG to create potential barriers defining channels, as shown in Figure 4. Splitting the path of electrons as they flow down the channel creates an opportunity for interesting quantum interference effects. A magnetic field, for example, can be used to tune the interference of electrons. This is the Aharonov-Bohm effect, which can be used to create many interesting nanodevices [17, 23].

However, even a single impurity within the device is enough to split the flow of electrons, creating unintentional interference effects [28]. We used a simulator called SQUALID-2D (Semiconductor QUantum AnaLysis Including Dissipation) [34] to study such effects in a nanowire with an impurity in the middle of the channel. By varying the strength of the influencing magnetic field, a series of 2D electron flows and electron potential scalar fields are generated. Using the magnetic field as the third dimension, these slices are stacked into a volume, which provides a convenient environment to study the electron properties under different magnetic conditions.

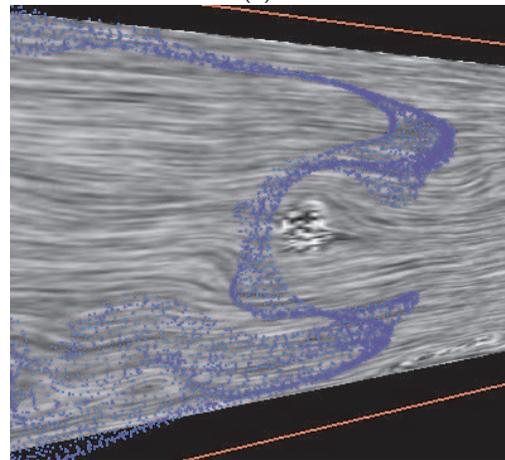
As shown in Figure 6.a, the particle simulation illustrates the conceptual flow of electrons in the device based on a current density that was computed quantum-mechanically. Near the front of the device, the strong magnetic field causes electron stream to curl and flow around the impurity on either side (see Figure 6.b), coupling the edge states as explained in [28]. Additionally, the electrochemical potential field, which is a measure of the average energy of electrons as they propagate down the channel, can be volume-rendered to further illustrate the strength and spatial features of the energy distribution. As shown in Figure 6.c, a sharp drop in electrochemical potential shows areas of resistance within the device as a result of an impurity.

## 7.2 BioMOCA

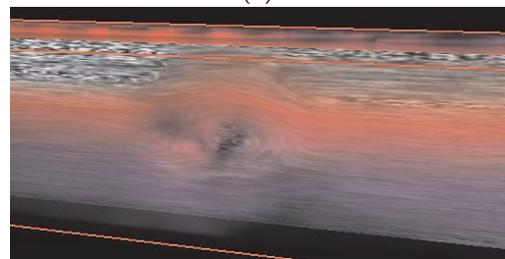
BioMOCA simulates the flow of ions through a pore in a cell membrane. This is normally a daunting task in molecular dynamics simulation, but BioMOCA borrows the Monte Carlo technique from traditional semiconductor device simulation and applies it to the biological realm. It computes random walks of ions through a channel with a fixed geometry within a cell membrane. The channel is shown in Figure 5 as it appears in the client window. The two vertical planes illustrates the walls of the cell membrane. The cut plane shows a 2-dimensional slice with the channel clearly highlighted as the magenta region running from left to right. Users can interactively explore the geometry of the channel by clicking and dragging on the legend as shown in Figure 5.b, to highlight a single isosurface. Each click or drag operation sends a command to the nanoVIS server, which responds instantly with an updated view.



(a)



(b)



(c)

Fig. 6. Electron gas simulation: (a) hybrid electron flow and potential visualization, (b) impurity causing electrons to swirl around, (c) sharp drop of electron potential surrounding the impurity.

## 7.3 Quantum Dot Lab

A quantum dot is a tiny chunk of conductor or semiconductor material surrounded on all sides by an insulator. Electrons inside are trapped by the insulator, and if the interior dimensions are small enough (perhaps a few dozen atoms in any direction), the electrons exhibit quantum effects, even at room temperature. Electrons sit in quantized states which resemble the wavefunctions of electrons bound to an atom. Because of this, quantum dots are sometimes referred to as “artificial atoms”, and they can be exploited in the same manner as atoms to create lasers and detectors that operate at very specific wavelengths of light [22].

The Quantum Dot Lab is based on the NEMO-3D simulator, which can be used to study various configurations of quantum dots [25]. This tool has a graphical interface generated by the Rappture toolkit, letting the user select the size and shape of the quantum dot, the material, the incident light source, and other parameters. Pressing the Simulate button launches a NEMO-3D simulation, which reports the electronic wavefunctions for the various quantized states, along with the absorption spectrum for incident light. Researchers can then tweak the size and shape of the quantum dot to achieve sensitivity to a particular wavelength of light.

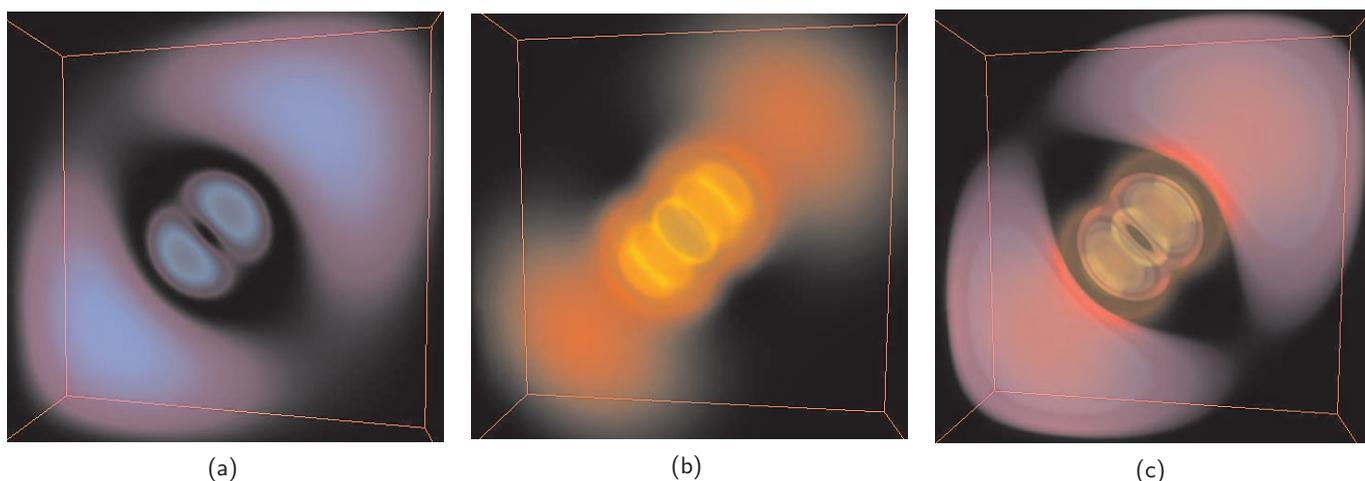


Fig. 7. Two million atom quantum dot simulation: (a) s electron orbital, (b) p electron orbital, (c) s and p electron orbital combined.

Visualization of the electronic wavefunction within the volume of the quantum dot is particularly helpful in this analysis. Figure 7 shows s- and p-orbitals for electrons confined within a rectangular quantum dot. We have found *visually* that a single wavefunction in a highly excited state can have significantly different nodal symmetries for its component orbitals. In this case, it is the s-orbital, rather than the p-orbital, that has most of its distribution concentrated near the boundary of the quantum dot.

## 8 Conclusion and Future work

We have described our remote visualization hardware and client-server software architecture for nanoHUB.org. As an integral component of the nanoHUB, our framework is capable of seamlessly delivering hardware accelerated nanotechnology visualization to remote simulation scientists with only minimal requirements on their computing environments. Our nanoVIS render server incorporates texture-based volume visualization and flow visualization techniques to flexibly handle a variety of nanoscience simulation data. As a component of NCN's Rappture Toolkit, the nanoVIS engine enables rapid development and deployment of new simulation tools. Additionally, we demonstrated that coupled with our GPU load estimation model and render node selection scheme, our approach is both efficient and scalable. Our system design can also be adopted to economically deliver accelerated graphics to other hub-based multi-user environments.

Future work on our system encompasses several directions. First, a more advanced compression algorithm may help reduce the network bandwidth usage under heavy render requests. Second, the nanoVIS render server can be further optimized to adaptively reduce the rendering quality during rapid user interaction sequences. Such an approach can further enhance our system's interactivity without degrading the user experience. Third, we can extend the nanoVIS engine to handle a richer set of grid topologies including unstructured meshes for finite element simulations.

## Acknowledgements

The authors would like to thank Martin Kraus, Nikolai Svakhine, Ross Maciejewski, Xiaoyu Li, the anonymous reviewers for many helpful discussions and comments, and nVIDIA for providing graphics hardware for testing. This material is based upon work supported by the National Science Foundation under Grant No. EEC-0228390.

## References

- [1] ParaView <http://www.paraview.org>.
- [2] VirtualGL <http://virtualgl.sourceforge.net/>.
- [3] Rappture Toolkit <http://rappture.org>.
- [4] DirectX9 <http://msdn.microsoft.com/directx/>.
- [5] C. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–230, 1973.
- [6] M. Brill, H. Hagen, H.-C. Rodrian, W. Djatschin, and S. V. Klimenko. Streamball techniques for flow visualization. In *Proceedings IEEE Visualization 1994*, 1994.
- [7] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. *ACM Symposium on Volume Visualization*, 1994.
- [8] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of SIGGRAPH 93*, pages 263–272, 1993.
- [9] H. R. Childs, E. Brugger, K. S. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max. A contract based system for large data visualization. In *Proceedings IEEE Visualization 2005*, 2005.
- [10] K. Engel, O. Sommer, and T. Ertl. A framework for interactive hardware accelerated remote 3d-visualization. In *Proceedings of EG/IEEE TCVF Symposium on Visualization VisSym'00*, pages 167–177, 2000.
- [11] J. Gao, J. Huang, C. R. Johnson, and S. Atchley. Distributed data management for large volume visualization. In *Proceedings IEEE Visualization 2005*, 2005.
- [12] W. Heidrich, R. Westermann, H.-P. Seidel, and T. Ertl. Applications of pixel textures in visualization and realistic image synthesis. In *ACM Symposium on Interactive 3D Graphics*, pages 127–134, 1999.
- [13] W. Hibbard. Visad: connecting people to computations and people to people. *Computer Graphics*, 32(3), 1991.
- [14] J. P. M. Hultquist. Constructing stream surfaces in steady 3d vector fields. In *Proceedings IEEE Visualization 1992*, 1992.
- [15] W. Humphrey, A. Dalke, and K. Schulten. Vmd: Visual molecular dynamics. *Journal of Molecular Graphics*, 14:33–38, February 1996.
- [16] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 693–702, 2002.
- [17] Y. Imry and R. A. Webb. Quantum interference and the aharonov-bohm effect. *Scientific American*, 260(4), 1989.
- [18] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Hardware accelerated texture advection for unsteady flow visualization. In *Proceedings IEEE Visualization 2000*, 2000.
- [19] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):211–222, 2002.
- [20] R. Kaehler, S. Prohaska, A. Hutanu, and H.-C. Hege. Visualization of time-dependent remote adaptive mesh refinement data. In *Proceedings IEEE Visualization 2005*, 2005.
- [21] M. J. Kilgard. *NVIDIA OpenGL Extension Specifications*. NVIDIA Corporation, 2001.
- [22] G. Klimeck, F. Oyafuso, R. Bowen, and T. Boykin. 3-d atomistic nanoelectronic modeling on high performance clusters: multimillion atom simulations. *Superlattices and Microstructures*, 31(2–4), 2002.

- [23] K. Kobayashi, H. Aikawa, S. Katsumoto, and Y. Iye. Tuning of the fano effect through a quantum dot in an aharonov-bohm interferometer. *Physical Review Letters*, 85(25), 2002.
- [24] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *Proceedings SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware 2004*, 2004.
- [25] M. Korkusinski, F. Saied, H. Xu, S. Lee, M. Sayeed, S. Goasguen, and G. Klimeck. Large scale simulations in nanostructures with nemo3-d on linux clusters. In *6th International Conference, Linux Clusters: The HPC Revolution 2005*, April 2005.
- [26] J. Krüger, P. Kipfer, P. Kondratieva, and R. Westermann. A particle system for interactive visualization of 3d flows. *IEEE Transactions on Visualization and Computer Graphics*, 11(6), 11 2005.
- [27] C. H. Lee and A. Varshney. Computing and displaying intermolecular negative volume for docking. *Scientific Visualization: Extracting Information and Knowledge from Scientific Datasets*, 2004.
- [28] Y. Lee, M. McLennan, and S. Datta. Anomalous rxn in the quantum hall regime due to impurity-bound states. *Physical Review B*, 43(17), 1991.
- [29] E. J. Luke and C. D. Hansen. Semotus visum: a flexible remote visualization framework. In *Proceedings IEEE Visualization 2002*, 2002.
- [30] K.-L. Ma and D. M. Camp. High performance visualization of time-varying volume data over a wide-area network status. In *Proceedings of Supercomputing 2000*, 2000.
- [31] R. Macleod, D. Weinstein, J. de St. Germain, D. Brooks, C. Johnson, and S. Parker. Scirun/biopse: Integrated problem solving environment for bioelectric field problems and visualization. In *Proceedings of the Int. Symp. on Biomed. Imag.*, pages 640–643, April 2004.
- [32] E. Martz. Protein explorer: Easy yet powerful macromolecular visualization. *Trends in Biochemical Sciences*, 27(2):107–109, February 2002.
- [33] N. Max and B. Becker. Flow visualization using moving textures. In *Proceedings of ICASW/LaRC Symposium on Visualizing Time-Varying Data*, pages 77–87, 1995.
- [34] M. McLennan, Y. Lee, and S. Datta. Voltage drop in mesoscopic systems: A numerical study using a quantum kinetic equation. *Physical Review B*, 43(17), 1991.
- [35] S. Muraki, E. B. Lum, K.-L. Ma, M. Ogata, and X. Liu. A pc cluster system for simultaneous interactive volumetric modeling and visualization. In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, 2003.
- [36] W. Qiao, D. S. Ebert, A. Entezari, M. Korkusinski, and G. Klimeck. Volqd: Direct volume rendering of multi-million atom quantum dot simulations. In *Proceedings of IEEE Visualization 2005*, pages 319–326, 2005.
- [37] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.
- [38] S. Röttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and iso-surface rendering based on cell-projection. In *Proceedings IEEE Visualization 2000*, pages 109–116, 2000.
- [39] A. Sadarjoen, T. v. Walsum, A. J. S. Hin, and F. H. Post. Particle tracing algorithms for 3d curvilinear grids. Delft Univeristy of Technology Technical Report DUT-TWI-94-80, 1994.
- [40] M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification (Version 2.0 - October 22, 2004)*. Silicon Graphics, Inc., 2004.
- [41] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. In *Proceedings of the 1990 workshop on Volume visualization*, pages 63–70, 1990.
- [42] S. Stegmaier, M. Magallon, and T. Ertl. A generic solution for hardware-accelerated remote visualization. In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 87–ff, 2002.
- [43] A. Telea and J. J. van Wijk. 3d ibfv: Hardware-accelerated 3d flow visualization. In *Proceedings IEEE Visualization 2003*, 2003.
- [44] S. Ueng, K. Sikorski, and K. Ma. Efficient streamline, streamribbon, and streamtube constructions on unstructured grids. *IEEE Transactions on Visualization and Computer Graphics*, pages 100–110, 1996.
- [45] J. J. van Wijk. Spot noise - texture synthesis for data visualization. In *Proceedings of SIGGRAPH 91*, pages 309–318, 1991.
- [46] D. Weiskopf, G. Erlebacher, and T. Ertl. A texture-based framework for spacetime-coherent visualization of time-dependent vector fields. In *Proceedings IEEE Visualization 2003*, 2003.
- [47] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *Proceedings of SIGGRAPH 98*, pages 169–177, 1998.