Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Power Sources 138 (2004) 187–193

JOURNAL OF POWER SOURCES

www.elsevier.com/locate/jpowsour

Short communication

Fuel-cell simulator interface

Andrei V. Smirnov[1], Hanzhou Zhang, B. Sowers, A. Burt, I. Celik*

West Virginia University, Morgantown, WV 26506, USA

Received 29 April 2004; accepted 24 May 2004
Available online 31 July 2004

Abstract

A 3D drawing methodology based on voxel-graphics was applied to the design of multi-component engineering systems, such as fuel-cells. Using this methodology and Java-technology a graphics user interface (GUI) for a fuel-cell simulator program was developed and used in simulations of large fuel-cell stacks. The GUI is capable to setup, run and monitor simulations remotely from a web-browser. The geometric design module was implemented using 3D voxel sculpting methodology and data visualization, which is prototyped after 2D pixel graphics systems. The developed approach was primarily aimed at the design of complex multi-component engineering systems. However, the flexibility of voxel-based geometry representation enables one to use this technique for both 3D geometric design and visualization of unstructured volume data. Examples of both applications are presented, with the focus on fuel-cell stack simulations.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Fuel-cells; Geometric design; Graphical user interface; Voxel-based sculpturing

1. Introduction

Distributed memory computer platforms, such as Beowulf clusters are increasingly used for complex scientific simulations of physical processes and engineering systems. Fuel cells offer a way of using the capabilities of distributed processing for efficient simulation of single fuel cells and fuel-cell stacks. The modularity of fuel-cell stacks can be exploited on computer clusters by running the simulation of each fuel cell on a separate processor. In earlier work the authors reported on the results of simulations of fuel-cell stacks using continuum solvers and distributed simulation techniques [1]. In this study, we present further developments of these techniques and focus on the issues of efficient simulation control on remote clusters and modeling of a single fuel cell as a multi-component system.

Until recently it was common to consider two basic geometries for fuel-cells: tubular and planar. Currently we witness a proliferation of various designs aimed at increased efficiency and power density. But even in the domain of simple planar designs there are multitudes of configurations of different components, such as anode, cathode, electrolyte, air/fuel channels, interconnect, separator plates, seals, current collectors, etc. Each component is typically represented by its own physical model. Many geometrical designs are employed, resulting in co-flow, counter-flow and cross-flow configurations [2]. Consequently, there are two issues that arise in the design of these complex multi-component, multi-physics systems: geometric design and physical modeling. This article gives a brief outline of the basic principles of general physical modeling used in our fuel-cell simulations, but is primarily concerned with geometric design of a single fuel cell. In particular, for typical fuel cells configurations we found that the design can be simplified by adopting a relatively straightforward method of voxel sculpting [3–5].

Another aspect of simulating fuel-cell stacks concerns simulation control on a remote cluster. The simulation solver has to be specifically implemented for execution on a distributed memory system, using domain decomposition techniques and message-passing interfaces (MPI, PVM). After such solver has been implemented, to perform a simulation one has to go through the stages of setup, execution, data processing and visualization. All the stages face challenges associated with the distributed nature of computations, especially when geometrically complex 3D systems are involved. The task becomes extra difficult when the cluster has to be accessed through the Internet from a remote workstation. In this case, the user of the cluster would greatly benefit from an accessibility to a graphical user interface (GUI), which could provide for remote control of the simulation. In this study we developed such a GUI, and used it in simulations

* Corresponding author.
E-mail address: ismail.celik@mail.wvu.edu (I. Celik).
[1] Tel.: +1-304-293-3111; fax: +1-304-293-6689.

of fuel-cell stacks on Beowulf clusters. The GUI performed functions of (1) simulation setup, including complex 3D geometric design, (2) monitoring and runtime control of the simulation, and (3) distributed data sampling and visualization.

2. Method

2.1. Client–server model

In order to effectively monitor and control the execution of a parallel application running on a remote cluster we implement a client–server scheme (Fig. 1). In this approach there is a single client running on a local workstation and server process running on a remote cluster. The client process enables the user to setup and remotely control the simulation, as well as retrieve and visualize data samples.

The server process controls the execution of a parallel solver running on the cluster. The server process is initiated as a part of the remote parallel application, and its purpose is to respond to client's requests. The are different channels by which the control information can be passed between the client and the server. In the simplest version the information can be exchanged through configuration files. Files containing user requests are created by the client and periodically read by the server, and server control information is written into files read by the client. In a more dynamic scheme the communications can be accomplished via MPI interface. The server is implemented as part of the solver, which is executed on each node the multi-processor simulation.

2.2. Modeling framework

The server running on the cluster is implemented as a set of functions that can be linked with a variety of continuum or discrete solvers, thus enabling one to control the simulation performed by the solver from a remote client. Client and server exchange information on parameters, variables and

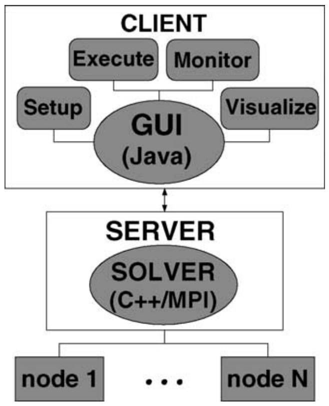


Fig. 1. Simulation setup with a remote GUI control based on a client–server model.

domains, which represent generic data types used by most continuum and discrete dynamics solvers.

Each parameter stores a single value attributed to the given model or to the simulation as a whole. Examples of parameters are total current through the system, ambient temperature, number of processors, data sampling interval, etc.

Variables are represented by a set of multi-dimensional values (scalars, vectors, etc.) with each element of the set attributed to one element of the domain. For example, distributions of temperature, current, chemical species, etc.

Domain is a connected region of space assigned to a specific physical model. For the purpose of numerical integration each domain is discretized by decomposing it into smaller and geometrically simple regions (elements), where physical laws are considered to be homogeneous and isotropic. The group of connected elements represents a grid, which can be of a structured type (global connectivity information) or unstructured type (local connectivity), which is also called *mesh*. Thus, a domain consists of a mesh, a set of variables and the solution procedure. The introduction of the domain data class provides the basis for muti-physics simulations, where different physical models can be assigned to the different regions of space.

2.3. User interface

The purpose of the local client process is to initiate the following actions through the user interface:

(1) problem setup on a local workstation,
(2) transfer of the data and source-code files to the remote cluster,
(3) building of the application executables and input files on the cluster,
(4) submitting the remote application for execution,
(5) monitoring of the remote run,
(6) sampling of data from cluster nodes,
(7) terminating the run,
(8) collecting the data,
(9) visualizing.

The most difficult among these tasks are related to geometric design, data retrieval and visualization. In particular, the following aspects of this work received the most of the development effort: (1) designing 3D drawing tools – 3D sculpting, (2) implementing 3D surface and volume rendering, (3) compression of voxel data, and (4) creating grid-independent representation of geometry.

2.3.1. Geometric design

An important issue in geometric design is defining complex 3D shapes. In most engineering CAD applications 3D drawing is realized by using combinations of pre-defined shapes: spheres, cones, surfaces of evolution, combined with planar drawing and extrusion operations. These 3D modeling techniques still lack the flexibility of general pixel graphics used in conventional 2D drawing programs (PaintBrush,

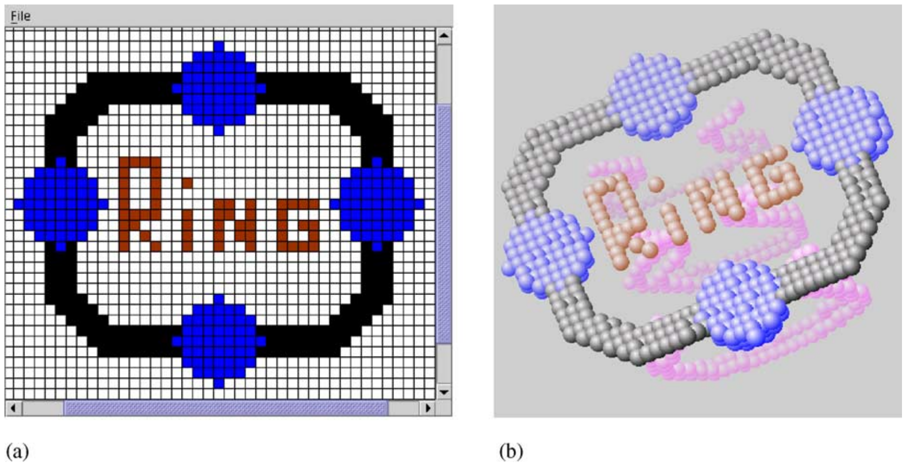


Fig. 2. Voxel sculpting of 3D shapes. (a) 2D drawing area and (b) 3D representation.

XPaint, etc.). Pixel graphics offer the possibility to easily create and alter complex geometrical shapes. The extension of a pixel to 3D is also known as volume pixel (*voxel*) [6]. Using voxel graphics creates additional advantages over surface representations (vector graphics), since they avoid completely topological complexities of surface transformations. This is because surfaces do not exist as objects in voxel representation. Another advantage of voxel graphics is their versatility and power combined with a great algorithmic simplicity.

The price to pay for this flexibility is seemingly inefficient usage of space, which has to be uniformly filled by the voxels. However, the very uniformity of voxel distribution opens the possibility to use efficient compression algorithms, so that the overall storage requirement for a surface or a compressed voxel representation becomes about the same. Unfortunately, efficient compression schemes can only be applied to voxel graphics for storage and communication purposes. Real time graphics manipulation by the drawing algorithm would require at least partially uncompressed image. This is the reason why voxel graphics were not seriously approached until recently. Today the situation may be changing. A complex multi-color scene described by a 1000^3-pixel cube, can already easily fit into a workstation with a 1 GB of RAM[2]. This opens an opportunity to revisit simple and robust pixel graphics techniques for engineering design and scientific applications that involve dynamic 3D geometries and complex scene transformations. In applications to fuel cells design the approach offers a simple technique for geometric design of these multi-component systems. We found it particularly useful in simulations of fuel-cell stacks on distributed memory computer platforms, where remote setup and control of the simulation can speed up the analysis.

In this work, we pursued the approach to geometric design based on *3D sculpting* and *voxel-graphics* [5,4,3]. The task of extending 2D pixel graphics to 3D can be accomplished on two levels of generality: (1) extending the drawing plane to 3D, and (2) introducing 3D paint-tool controls.

A simple extension of planar drawing plane to 3D is relatively straightforward. It requires the introduction of the third dimension into the pixel-array and identifying the position and orientation of the drawing plane. In a simplified case the drawing plane can have three different orientations with respect to Cartesian coordinates. The image drawn in the plane can then be *extruded* into the third dimension, analogously to the operation done in conventional CAD applications (Fig. 2). This approach is adequate for the purposes of designing many engineering systems, such as planar fuel-cells. A more general approach to *voxel-sculpting* [5,4] is being currently pursued with the introduction of 3D sculpting tools.

Introducing 3D drawing tool controls can be as simple as changing the position and orientation of the drawing plane. However, to alleviate the frustration of dealing with hundreds of drawing planes in case of high-resolution 3D scenes, more advanced 3D drawing tools and motion controls should be introduced. In essence, each 2D drawing tool can have its 3D counterpart, with extra spatial dimension added to the tool. For example, the drawing pen can be represented by a color-filled ball of a certain radius. The user selects the color and the radius of the ball. Positioning of the ball in space can be done by selecting the direction vector of ball's motion and then advancing it along this direction. Since there are three parameters required for this operation, it can be done, using only mouse controls: two mouse-position coordinates to set the direction vector, and mouse-wheel motion to set the position along that direction. More sophisticated 3D navigation tools can be developed using prototype controls of a flight-simulator application.

[2] We consider a 256 color scheme where one pixel can be represented by one byte in a computer memory

2.3.2. Visualization

Effective 3D visualization of the drawn scene is the key supplement to successful drawing capabilities. An almost trivial feature in 2D graphics, visualization and surface rendering become a major effort in 3D. For most purposes of engineering design a simple wireframe rendering mode is usually enough. This can be accomplished in a number of ways, and in a manner consistent with the resolution of the image, i.e. the ratio of the image size to the grid-cell size. Three wireframe rendering models were implemented. One of them is based on a relatively versatile and fast method of constructing cutting-plane contours. The number of planes, their orientations and separations can be set by the user, thus, adjusting the rendering to high versus low resolution scenes. Another wireframe model is based on a direct rendering of surface edges of every boundary voxel as a set of segments. This representation is most memory consuming and preserves all the information contained in the voxel format. It can be used for low-resolution scenes with a small number of voxels. The third wireframe representation is grid-independent type, where the surface is stored as a triangulated mesh, with mesh properties independent of the three axes directions of the original voxel-grid. The method of constructing such surface is based on boundary surface reconstruction algorithm, which filters out the main wavenumber associated with the underlying discrete grid, and which was specifically developed in the course of this study.

The advantages of wireframe rendering are that it is relatively simple to handle algorithmically and sufficiently fast to work well even without accelerated graphics. It also provides one with the depth perspective. Nevertheless, for an accomplished drawing and design package a more advanced surface and texture rendering is needed. For this purpose Java3D graphics library was used, which works best with hardware accelerated graphics. In this combination both wireframe and surface rendering modes are possible.

2.3.3. Data compression

Even though scenes of relatively high resolution can be created on a modern workstation, when used remotely, pixel graphics can still present a problem because of the necessity to transfer bulky voxel representations over relatively slow networks. Fortunately, most of the scenes of practical interest can be effectively compressed to many times less than the size of raw voxel representation. This is due to the fact that the amount of information contained in a scene is independent of whether pixel or vector graphics representation is used to describe it. This information is rather related to the positions and shapes of the few objects populating the scene. From this perspective, the efficient compression algorithms applied to a voxel-graphics representation will eliminate the inherent redundancy of voxel format and convert it into a high-entropy format which will be comparable in size to any other compact format with the degree of compression close to theoretical maximum.

2.3.4. Simulation control

After the physical model has been setup the client can initiate the transfer of necessary files to the cluster and schedule the simulation for execution. Once started, the simulation can be monitored by periodic data sampling from the cluster nodes and displaying them in numerical or graphical format. The data sampling strategy is set from the considerations of bandwidth and problem size. One dimensional (vector) data can be displayed as 2D plots.

Simulation parameters represent the input data of the problem, which are not affected by the simulation, such as initial/boundary conditions, number of processors, the duration of the run, etc. Almost all the simulation parameters can be changed dynamically during program execution. This enables one to change simulation conditions in real-time.

Table 1 displays the list of some parameters used to control the sampling sizes and frequencies as well as several physical parameters of a fuel-cell model. Some fields in the parameter table can be set by the user, and others are fixed. Each parameter is identified by several properties. *Scope* determines if the parameter represents a variable, defined on the nodes of computational mesh, such as temperature or concentration, or a single value valid for all the simulation, such as total current or the ambient temperature. Parameters which belong to parameter-scope are not modified by the solver, and can be changed by the user during the simulation. Parameters of the variable-scope are subdivided into *control variables* and *variables*. Variables are modified by the solver during the run, and thus can only be set as initial parameters of the simulation, whereas the control variables can be set by the user during the run. The type of the parameter identifies its numerical representation as an integer or a real number. The parameter *dimension* identifies it as a scalar (0), a vector (1), or a general *n*-rank tensor (*n*). The *value* and *monitor* fields are set by the user, where the latter

Table 1
Simulation control parameters for a fuel-cell application

Name	Scope	Type	Dimension	Value	Monitor
NP	Parameter	Int	0	10	False
MonitorPlane	Parameter	Int	0	10	False
TotCurrent	Parameter	Real	0	600.0	False
TemperatureAmb	Parameter	Real	0	1250.0	False
StopTime	Parameter	Real	0	9000.0	False
PrintCntStep	Parameter	Int	0	1E9	False
PrintTimeStep	Parameter	Real	0	100.0	False
CathodeInletVel	Parameter	Real	0	1.214	False
AnodeInletVel	Parameter	Real	0	0.407	False
CathodeInletT	Parameter	Real	0	1073.0	False
AnodeInletT	Parameter	Real	0	1073.0	False
TimeStep	Controlvar	Real	0	0.0	False
TemperaturePEN	Variable	Real	0	1200.0	True
TemperatureAir	Variable	Real	0	1200.0	True
TemperatureFuel	Variable	Real	0	1200.0	True
TemperatureSep	Variable	Real	0	1200.0	True
TemperatureTop	Variable	Real	0	1200.0	True
CurrentDensity	Variable	Real	0	0.0	True

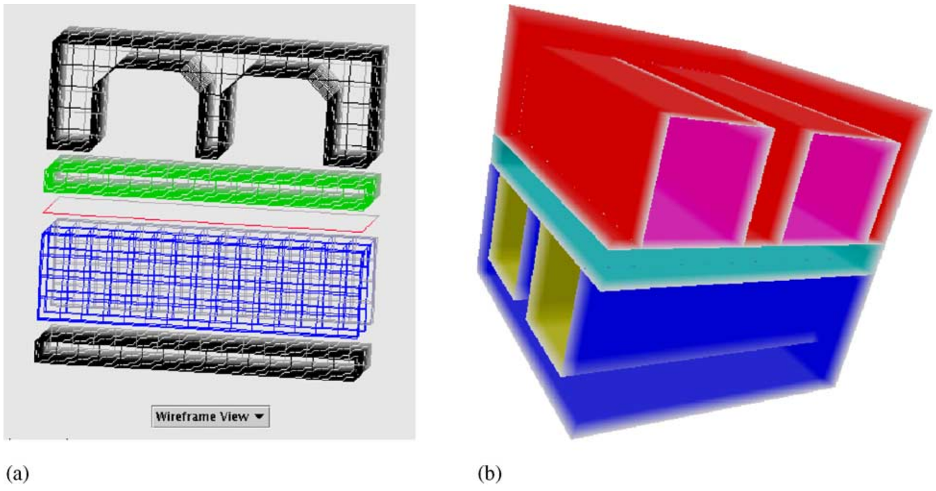
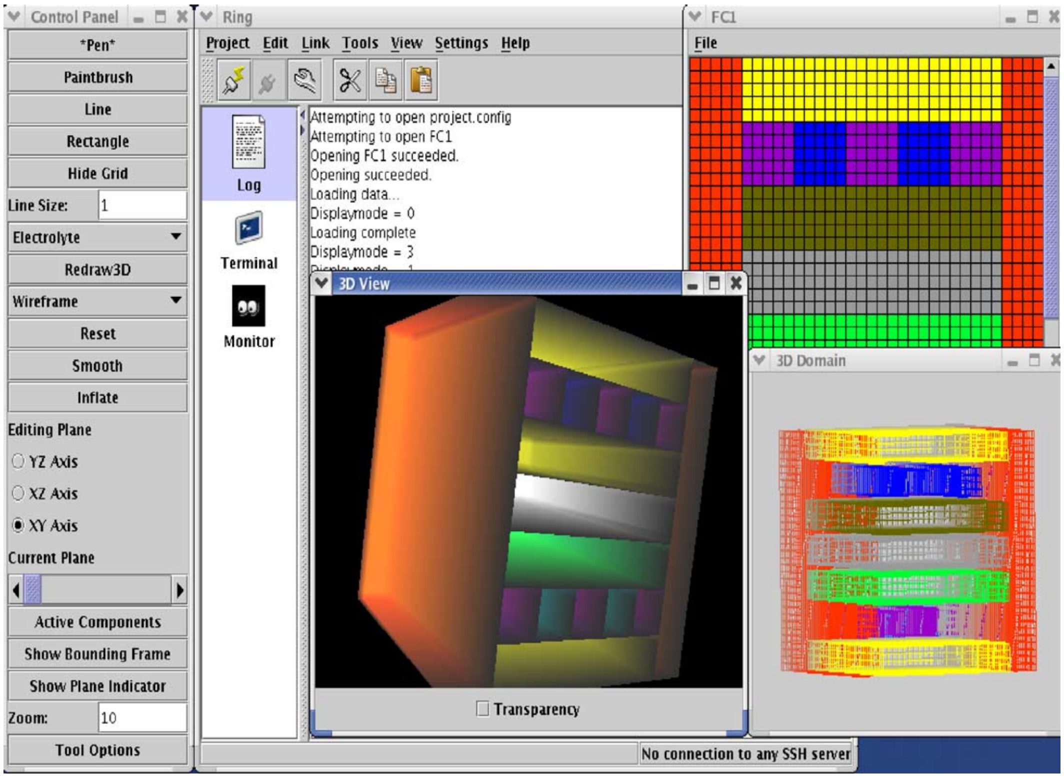


Fig. 3. Geometric design of fuel cells: surface representations. (a) Wireframe representation and (b) surface representation.

indicates if the parameter's values will be monitored during the run.

It should be noted that the flexibility of setting up the control parameters enables one not only to start/stop the execution but also to change model parameters during the simulation, i.e. ambient temperature, total current, etc.

In addition to providing visualization capabilities, remote data monitoring, and control of the simulation, the interface essentially hides from the user the intricacies of the underlying operating system running on the cluster. Some of the interface menu functions can in fact be developer-defined. Thus, it is possible for the code developer to assign different Unix-type commands for the user to execute on the cluster without requiring proficiency with Unix. These commands can be changed or implemented without the need to recompile the interface executable itself.

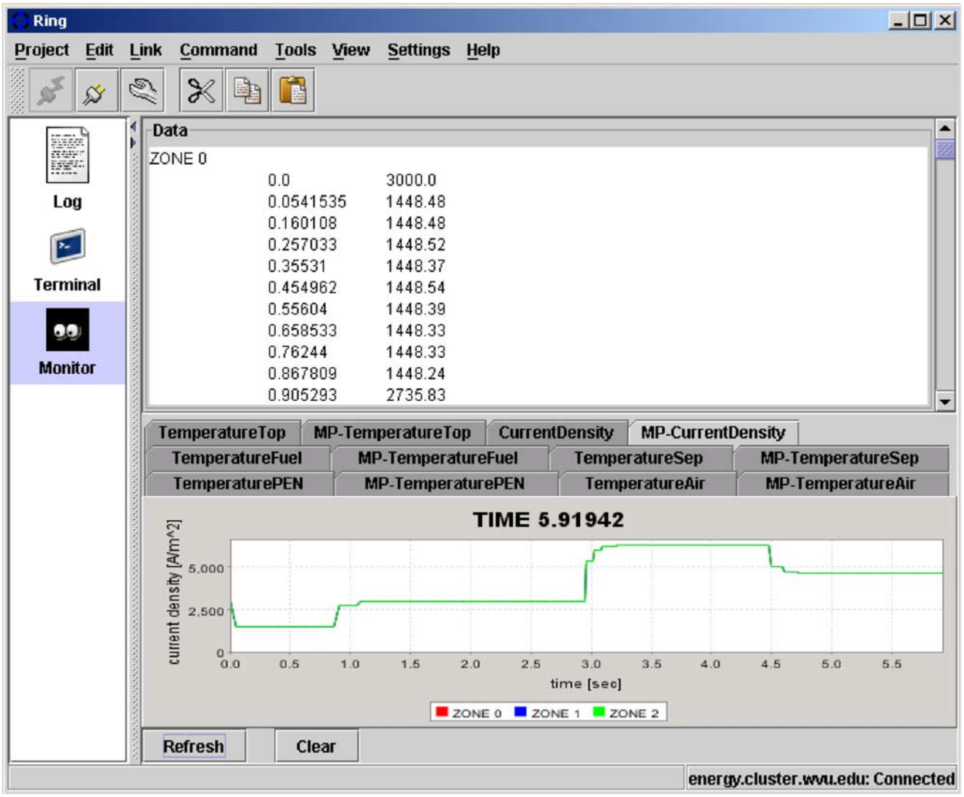


Line Size: 1
Zoom: 10

```
Attempting to open project.config
Attempting to open FC1
Opening FC1 succeeded.
Opening succeeded.
Loading data...
Displaymode = 0
Loading complete
Displaymode = 3
```

No connection to any SSH server

Fig. 4. GUI control and monitoring windows: control panel (left), 3D view panel (middle bottom), 2D drawing pane (right top), 3D wireframe view (right bottom), main panel (in the back).


Ring
Project Edit Link Command Tools View Settings Help
Log
Terminal
Monitor
Data
TemperatureTop MP-TemperatureTop CurrentDensity MP-CurrentDensity
TemperatureFuel MP-TemperatureFuel TemperatureSep MP-TemperatureSep
TemperaturePEN MP-TemperaturePEN TemperatureAir MP-TemperatureAir
TIME 5.91942
current density [A/m^2]
time [sec]
ZONE 0 ZONE 1 ZONE 2
Refresh
Clear
energy.cluster.wvu.edu: Connected

```
ZONE 0
          0.0           3000.0
          0.0541535     1448.48
          0.160108      1448.48
          0.257033      1448.52
          0.35531       1448.37
          0.454962      1448.54
          0.55604       1448.39
          0.658533      1448.33
          0.76244       1448.33
          0.867809      1448.24
          0.905293      2735.83
```

Fig. 5. Remote monitoring of transient temperatures.

3. Simulation of fuel-cell stacks

The methodology of integrated simulation setup and control based on voxel-graphics and Java-technology was applied to simulations of fuel-cell stacks on Beowulf clusters. In this case the geometric design of a fuel cell is done on a local workstation by means of voxel-based graphics tools implemented in Java. The geometric information and the setup parameters are then transferred to the cluster. After the simulation is started it can be monitored from on the workstation by periodically retrieving data samples and displaying them in graphics format.

Fig. 2 shows the example of voxel-based sculpting of arbitrary 3D shapes. Application of this voxel-based sculpting to a cross-flow fuel-cell geometry is shown in Fig. 3. The geometry can be displayed either in wireframe representation or using surface rendering.

A screen-shot of the GUI is shown in Fig. 4 where the main components, such as the main panel, the control panel, the 3D view panel, the 2D drawing pane, and the 3D wireframe view are displayed. Fig. 5 shows the screenshot of the monitoring window where the transient temperature retrieved from the remote cluster nodes are displayed in a graphical format. Considering small time-steps that are required for electrical and chemical sub-models of the solver, such simulation may take large computer resources in terms of time and memory. Thus, it is important to realize a flexible system of simulation control which enables one to ad-

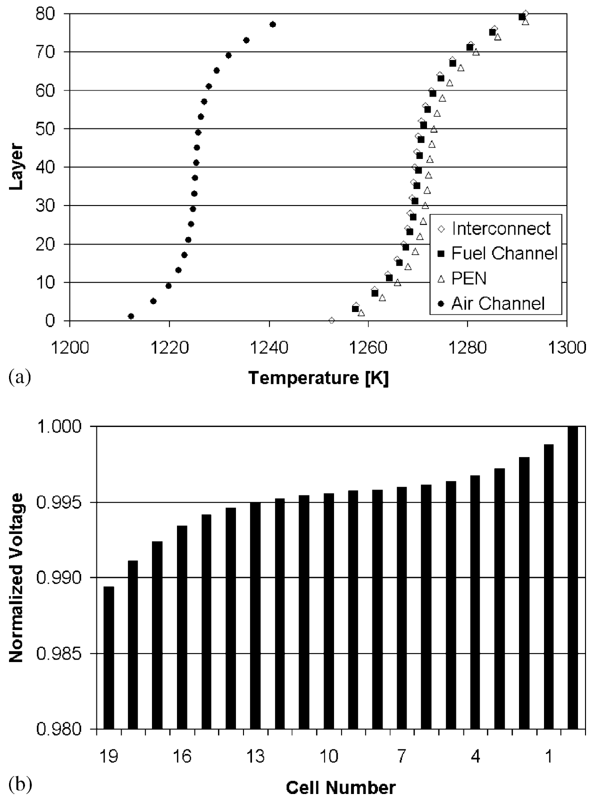


Fig. 6. Distributions of physical parameters within the stack. (a) Temperature, (b) voltage.

just parameters during the run. Thus, in Fig. 5 thermal responses to changes in total current are observed. It should also be noted that large fuel-cell stacks may exhibit unexpected temperature and voltage variations, depending on the performance of separate cells. The developed system can be effectively used to simulate various stack operation scenarios, where the failure of one of several cells may affect the overall stack performance.

This system of remote setup and monitoring was successfully applied to the simulation of large stacks of up to 40 solid oxide fuel cells [1,7,8]. Fig. 6 shows sample distributions of temperature and voltage within a 20-cell stack. This simulation was done under uniform stack conditions with respect to fuel and oxidizer supply. However, non-uniform variations of temperature and voltage can clearly be observed for the bottom and top group of cells in the stack.

4. Conclusions

An approach to 3D graphics based on voxel-representation was successfully applied to the setup of typical fuel-cell geometries. The approach offers considerable simplicity and flexibility. It also enables one to combine geometric design and data visualization in a single framework.

Making use of Java-technology and a client–server model enables one to design web-based user interfaces for remote control and monitoring of scientific and engineering simulations on Beowulf clusters.

Because of inherent modularity of fuel-cell stacks these systems can be effectively simulated on distributed memory platforms, such as workstation clusters. These simulations can benefit from remote interfaces with graphical capabilities, such as the one developed in this study. An extra advantage of the interface is the flexible control of the simulation, which provides the possibility of playing out different operation scenarios.

Based on the results of this study we conclude that voxel-graphics is a promising technique for applications in grid and cluster computing, related to 3D geometric design and data visualization.

Acknowledgements

This work was jointly sponsored by the US Department of Energy, National Energy Technology Laboratory and West Virginia University.

References

[1] A. Burt, I. Celik, R. Gemmen, A. Smirnov, A numerical study of cell to cell variations in a SOFC stack, J. Power Sources 126 (2004) 76–87.

[2] E. Services, I. Parsons, S.A.I. Corporation, Fuel Cell Handbook, fifth ed., Tech. Rep. DOE/NETL-2000/1110, US Department of Energy, Office of Fossil Energy, Federal Energy Technology Center, 2000.

[3] H. Chen, H. Sun, Real-time haptic sculpting in virtual volume space, in: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, ACM Press, 2002, pp. 81–88.

[4] S.W. Wang, A.E. Kaufman, Volume sculpting, in: Proceedings of the 1995 Symposium on Interactive 3D Graphics, ACM Press, 1995, pp. 151–158.

[5] T.A. Galyean, J.F. Hughes, Sculpting: an interactive volumetric modeling technique, in: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, 1991, pp. 267–274.

[6] A. Kaufman, D. Cohen, R. Yagel, Volume graphics, Computer 26 (7) (1993) 51–64.

[7] A. Burt, I. Celik, R. Gemmen, W. Smirnov, Cell to cell performance variations within a stack, in: Proceedings of the Eighth International Symposium on Solid Oxide Fuel Cells (SOFC VIII), Paris, France, 2003, pp. 217–223.

[8] A. Burt, I. Celik, R. Gemmen, A. Smirnov, Influence of radiative heat transfer on variation of cell voltage within a stack, in: Proceedings of the First International Conference on Fuel Cell Science, Engineering, and Technology, Rochester, NY, 2003, pp. 1487–1500.