Electrical Distribution Grid Visualization using Programmable GPUs

Javier Novo Rodríguez[1],
Mariano Cabrero Canosa[2], and **Elena Hernández Pereira**[3], Non-members

ABSTRACT

Modern graphics cards enable applications to process big amounts of graphical data faster than CPUs, allowing high-volume parallelizable data to be visualized in real-time. In this paper, we present an approach to enable a power grid planning Computer-Aided-Design application to use this processing power to visualize electrical distribution grids in the fastest possible way. Specifically, the GPU has been programmed using Vertex Shaders in order to generate wide lines representing the power lines carrying a given power load. Furthermore, this kind of shaders was also used to translate the coordinates of the power grid from the geographic to the graphic domain, thus offloading this work from the CPU. As a result, the aforesaid application became able to render the visualization of power grids much faster and to offer highly responsive interactions with them.

Keywords: GPGPU, Shaders, Power Grid Visualization

1. INTRODUCTION

Power grid visualization has traditionally been focused on transport networks as shown in [1, 5]. While these networks configure the backbone of the power grid, distribution systems account for up to 90% of all customer reliability problems [10]. Therefore, electric power utilities need their Computer-Aided-Design (CAD) applications to be able to manage distribution networks as well. These networks – also known as medium-voltage networks – are more complex than the transport ones and require advanced computing power to be visualized. The rendering of such visualizations has traditionally taken too long, making them only suitable for off-line tasks such as plotting. The high level of parallelism and computing power offered by modern Graphical Processing Units (GPUs) have opened the window for such a visualization to be done in a reasonable amount of time, allowing the electrical engineers to interact in real-time with the visualizations and simulations offered by CAD tools.

Manuscript received on July 31, 2010 ; revised on May 1, 2011.

[1,2,3] The authors are with Department of Computer Science University of A Coruña, Spain , E-mail: javier.novo@udc.es, mariano.cabrero@udc.es and elena.hernandez@udc.es

Traditionally, one-line diagram models – which provide schematic representations – have been used as the main tool for mathematical analysis in electric networks design. Their main tradeoff is the absence of any spatial attributes which also play an important role in the networks design. Thanks to the progressive adoption of Geographic Information Systems (GIS) by the utilities, CAD applications can now access very precise data about the spatial relations within the networks and how they relate to external elements in the real world [6, 7]. Several power grid visualization techniques that consume this information have been developed over the years [2, 3, 8], the most common being the Geographic Data Views (GDVs) which overlay technical power data such as loads and flows over the geographic representation of the grid [4].

While the technical data represented in GDVs require some computing power – more notably for advanced visualizations such as contours – it is the processing and rendering of the topological data that conform the bottleneck of the distribution grid visualization. The processing of the grid data involved is divided in two stages: translating the geographic data to the graphic domain and rendering the visual representation of the grid topology. Beyond the basic geographic information, the width of the lines linking the different nodes of the grid is commonly used to represent the load of the power line being displayed. Even with techniques using modern hardware rendering pipelines to draw the representation, the performance falls short – making the process unsuitable for real-time human interaction with the CAD applications – unless the data domain translation takes place in the GPU as well. Such functionality can be introduced by using the programmable graphics pipelines available in modern GPUs [18].

The main two high-performance graphics technologies currently available are OpenGL and Direct3D, both of which act as a hardware abstraction layer offering developers a flexible and homogenous way of interacting with graphics hardware. Both technologies expose Application Programming Interfaces (APIs) that follow a similar design and offer three kinds of basic primitives – points, lines and triangles – which are combined to form complex three-dimensional structures. The main primitive required for a topological representation of the grid is the line with a given width. While lines are available as basic primitives, a width can not be specified for them, yielding the need to draw rectangles formed by two triangles. As a result, both the complexity and the memory and processing requirements are increased. The solution proposed, implemented as an electric grid visualization library, uses Direct3D and its High Level Shading Language (HLSL) to perform both the required data domain conversion and the drawing of the grid topology efficiently.

This paper is organized as follows: Section 2. presents a characterization of the data used, introducing the most relevant concepts of electric distribution networks and describing the datasets selected to test the implementation. Section 3.describes the proposed solution, and then the performance results of a test application using this solution to visualize the datasets are presented in Section 4. Finally, the conclusions obtained from this work are exposed in Section 5.

2. DISTRIBUTION POWER GRIDS CHARACTERIZATION

Power management comprises disparate areas that conform a very complex system known as the electric utility system. Its typical structure is introduced here before presenting the distribution networks datasets used in this work.

2.1 Components of the electric utility system

The electric utility system is usually divided into three stages: generation, transmission and distribution. In complex systems, another stage called subtransmission may be introduced [9]. Since it usually overlaps with the transmission and distribution stages it is ignored here for simplicity.

Figure 1 shows a top-level depiction of the electric utility system. Classical power generation systems, such as fossil fuel or nuclear power plants, generate voltages typically between 11 and 30 kV. Generation plants are connected to transmission lines through generation substations, where a step-up transformer increases the voltage to transmission levels – generally well over 110 kV. Transmission lines are used to transport the electricity over long distances through different transmission substations to distribution substations where electric energy enters the last stage before being retailed. Distribution systems can be split into:

1. Primary distribution systems, which consist of feeders that deliver power from distribution substations to distribution transformers through distribution lines carrying voltages usually between 1 and 35 kV. In Europe, these systems are often called medium-voltage networks in contrast to high-voltage networks which is the name given to transmission networks.

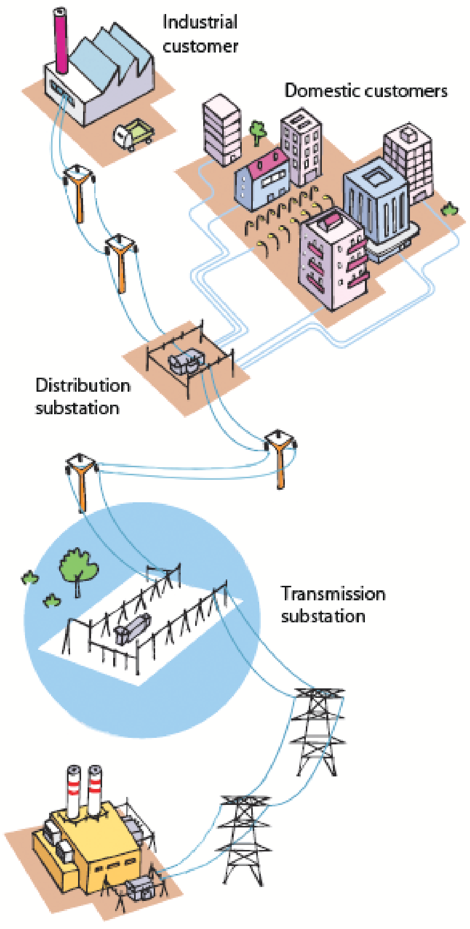

Fig.1: *The electric utility system*

2. Secondary distribution systems, which connect distribution transformers and customers. The complexity of these systems vary greatly between regions: in the US it is common to find less than 10 customers per transformer whereas in Europe the number may go up to several hundreds [10]. Voltages usually range between 220 and 240 V. Because of these values, in Europe these systems are also known as low-voltage networks.

2.2 Primary distribution network datasets

Several published works have studied the topological structure of power grids in order to analyze the structural risks within US, Italian, French and Spanish networks [11–13]. Furthermore, some techniques to generate random power grid topologies for simulation purposes have been developed [14]. In this work, six real distribution primary networks extracted from different regions over the world have been analyzed and used to test the visualization performance:

- Three different Central American countries: Guatemala, with a high population density, and Nicaragua and Panama, with low population densities.
- One Eastern European country, Moldova, which

has a medium population density.
• Two different regions of Spain: Galicia, the northwesternmost area of Spain, formed by 4 provinces with a scattered, medium-density population; and Central Spain which comprises 9 provinces in the center of Spain, including Madrid – the province with the highest population density in the country – and most of the less densely-populated provinces of the country.

Table 1: *Population density by region*

Region	**Population**	**Area (km^2)**	**Density**
Nicaragua	5,677,771	130,373	43.55
Panama	3,394,528	75,517	44.95
Central Spain	8,856,615	115,777	76.50
Galicia	2,783,100	29,574	94.11
Moldova	3,633,369	33,846	107.35
Guatemala	13,675,714	108,890	125.59

Table 1 summarizes some population characteristics of the mentioned regions. They have been selected to cover different populations and densities [15]. The largest covered area has over 130,000 squared kilometers which should already be too large and too highly detailed to interact with through visualization in a CAD application.

For each region, a dataset including the geographic and topological data of the network has been created. This data can be divided into two categories:
• Knots: points where three or more branches converge on. Typically, distribution substations, transformers and feeders are located at these points.
• Branches: segments connecting knots and composed by one or more nodes linked by power lines. Besides power lines interconnections, the nodes may also be switchgear or medium-to-medium voltage transformers.

Table 2 shows the different datasets, detailing the number of knots and branches for each network. Furthermore, the number of lines and nodes that compose the branches are shown for each dataset.

Table 2: *Distribution networks datasets*

Region	**Knots**	**Branches**	**Nodes**	**Lines**
Moldova	59,726	43,769	160,717	116,948
Nicaragua	34,912	95,771	245,136	149,365
Panama	85,990	85,319	262,319	177,000
Guatemala	142,427	142,507	345,645	203,138
Galicia	23,812	91,959	301,118	209,159
Central Spain	35,522	147,651	493,121	345,470

In order to simplify the performance tests and to focus on the bulk of the topological data, the time required by a test application using the implemented solution to draw the lines that conform the networks is measured. Before the drawing can take place, the nodes that form the different lines must be processed to translate their coordinates from the GIS domain to the graphic domain. While the main focus is on visualizing network topologies, GDV visualization may also affect the performance. A fundamental example is the use of the width of the lines to visualize the load on the power line. Another example of such visualization not considered here is representing the different substations as squares with their size varying as a function of a given electrical parameter. Hence, the primitives drawn to test the performance are lines with an arbitrary width for the lines forming the network branches.

3. IMPLEMENTATION

The proposed solution has been implemented as an electric grid visualization library [16]. Direct3D was chosen as the graphics API over OpenGL due to its state of art rendering technology, even though both technologies have a similar design and the same three basic graphical primitives: points, lines and triangles. Given that the line primitive has a unitary width, its usage is usually limited to represent wireframed three-dimensional models. A wide line is essentially a rectangle and thus, in order to be drawn, two triangles must be used. For instance, for the Galicia dataset, almost half a million triangles must be drawn. While this is a number that can be easily handled by modern GPUs, it must be taken into account that such number only involves the topological visualization. Other graphical representations such as geographic contours or satellite imaging usually need to be performed at the same time.

Since the library is to be integrated with a CAD application, several design considerations must also be taken into account regarding the specifics of event-driven applications. These issues along with those related with the GPU management are developed in the following paragraphs.

3.1 Graphical primitives pre-processing

The three vertices that compose each triangle will be processed by the GPU to transform its coordinates and optional attributes such as color. The more vertices are involved, the more memory and processing time are required. Thus, it is essential to minimize the number of vertices used by reusing adjacent vertices. For instance, two triangles forming a rectangle have two of their vertices located in the same position. Consequently, these two vertices can be shared, requiring only four instead of six vertices to define the rectangle. Indexed primitives lists offer this reuse capability and they have been available for a while in the main graphic APIs. They are used to define primitives as a list of pointers to vertices stored in a separated memory buffer. This way, since different primitives can point to the same vertices, they are effectively shared and redundancy is avoided.

3.2 Vertex shaders

Traditionally, in what is known as the fixed-function rendering pipeline, the transformation performed by the GPU over each vertex was a simple matrix multiplication to translate, rotate and/or scale the coordinates, and then apply projections and viewports. With the introduction of the programmable pipeline, this operation is replaced by small programs called vertex shaders which are executed for every vertex conforming the graphic primitives [19]. Several copies of a vertex shader run at the same time on the different vertex shader units available in the GPU, allowing the parallel execution of the shader over different vertices. The matrix multiplication usually still needs to be performed, but new and more complex operations can be included in the shader code.

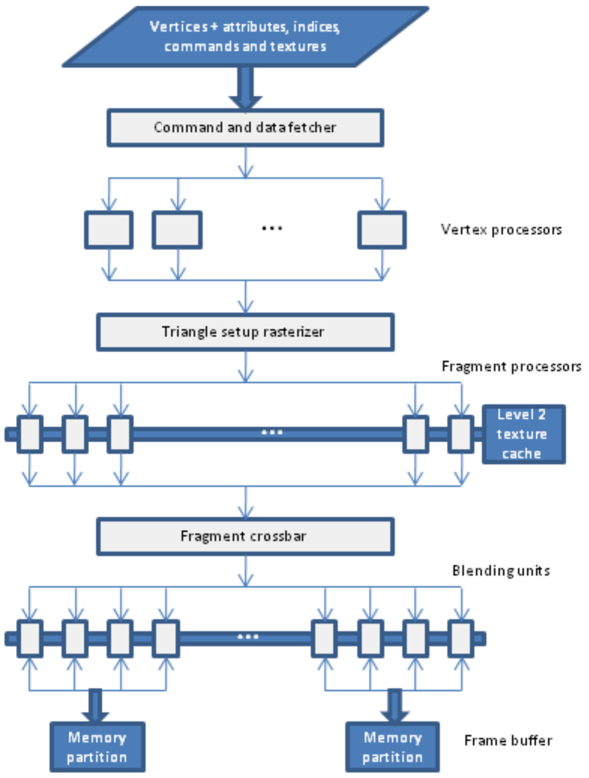

Fig.2: *Programmable pipeline block diagram*

Figure 2 shows a block diagram of a third-generation GPU – also referred to as Shader Model 3 or Direct3D 9 hardware. Primitives are defined by specifying their vertices and how they relate to one another to form primitives through the indexed primitives lists. After being passed into the GPU from the CPU, the rendering commands and the data involved is fetched and supplied to the vertex processors or vertex shader units. Because vertices are independent and the data fetcher assigns incoming work to any idle processor, the parallel utilization is nearly perfect. Each processor executes the vertex shader for the assigned vertex. Once processed, the vertices are combined into the required primitive, which is in turn rasterized – i.e. filled with pixels. These pixels are known as fragments and are passed to the fragment processors – also known as pixel shader units – where a pixel shader can be executed in parallel over them. The result of this processing is passed to the different blending units, where raster operations are performed before writing the final pixels to the frame buffer. Last generation hardware unifies the different kinds of shading units or processors under a general one which performs one task or another depending on the kind of dominant workload. This results in a better load balancing and a best utilization of the hardware since the number of idle units is greatly diminished.

Our implementation takes advance of this architecture to define two vertex shaders, each one performing a different task:

1. Translate the coordinates from the GIS to the graphic domain. This task can be totally parallelized given the independent nature of the coordinates, fitting the GPU architecture better. Hence, many CPU cycles per vertex can be saved by moving this task to the GPU.
2. Draw wide lines efficiently as rectangles representing the available topological data.

Since only one vertex shader can be used at the same time, the second one subsumes the coordinate translation before generating the rectangles representing the lines with a given width. Following, each shader implementation is detailed:

3.2.1 Coordinates domain translation

To use the GPU to translate each vertex from the GIS domain to the graphic domain, all vertices are declared using the geographic coordinates. A vertex shader is then used to operate on every single vertex and to translate those coordinates to the graphic domain. Once this conversion is done, the coordinates are multiplied by the classic three-dimensional world-view-projection matrix. Given the parallel nature of the GPUs, closer to vector processors than to CPUs, these operations are performed much more efficiently and the CPU is freed from this workload. As a result, the overall performance is improved, most notably for critical CPU-dependant areas such as graphical user interfaces. Given that the translation only requires basic mathematical operations, even the most primitive vertex shaders are able to accomplish this task. Therefore, no special hardware needs arise from the use of this functionality. This approach follows the trend of using the high-performance parallel architecture present in modern GPUs for non-graphical applications – namely general-purpose computation on graphics hardware (GPGPUs) [17].

3.2.2 Wide lines

These lines are defined by a desired width and its beginning and end points. With these three parameters two triangles forming a rectangle must be generated. This rectangle has the given width and it is bisected by the line connecting the beginning and end points. It must be noted that creating a rectangle from these parameters is more complex than doing so from the traditional declaration embodying one point plus height and width, which only requires additions to calculate the coordinates of the four vertices of the rectangle. The approach presented in [20] was used: for each line, these coordinates are calculated using the normalized normal vector from each given point of the bisector line to the opposite point. In order for each vertex shader to be able to perform these operations, a new vertex type is required. This vertex must include the coordinates and color information as well as the extra information involved in the calculus: the coordinates of the opposite point and the desired width. For each line, two vertices located at the beginning point coordinates and two vertices at the end point are created. As Fig. 3 shows, after the vertex shader processes the four vertices, each one will be located in a corner of the desired rectangle.

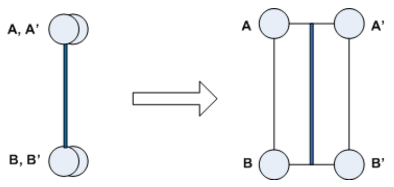

Fig.3: *Rectangle generation using the width and its bisector line*

An important optimization can be introduced by using geometry shaders instead of vertex shaders to generate the wide lines. Geometry shaders were introduced by Direct3D 10 through its Shader Model 4.0. After the vertex processing, a new stage is introduced where a shader operating on primitives as a whole can be used to modify their geometries by adding or removing vertices or even to create new primitives [21]. By using a geometry shader, half the vertices are needed in the vertex processing stage since the other half can be generated by the geometry shader. The downside of this approach is that it requires fourth-generation graphics hardware supporting Direct3D 10, which may not be yet available in the workstations of a utility. The first GPU supporting geometry shaders was the GeForce 8800 released by NVIDIA in 2007 targeting the high-end gaming market.

3.3 Integration in a event-driven application

Applications using Direct3D are usually real-time rendering applications in which the same scene is rendered several times per second. In such applications, the execution flow loops around the rendering routines that accomplish such work. On the other hand, applications with graphical user interfaces such as CAD programs, are event-driven applications which react to events received from the system who carries the control of the execution.

Integrating Direct3D into a Windows application along with common user-interface controls can be done relatively easily thanks to inter-operation technologies available. Nevertheless, special care must be put into the design of the application so it is able to render through Direct3D without impacting the responsiveness of the user interface. The application must use multi-threading techniques since the graphical user interface and the Direct3D rendering routines should run on different threads to prevent the user interface from being unresponsive during long renderings. Only one entry point for the render process should be provided since Direct3D devices can not be used from more than one thread at the same time.

Drawing operations must be grouped instead of being dispatched right away, resulting in what is known as retained graphics mode as opposed to the immediate mode. Drawing calls and state changes in the rendering pipeline must be minimized in order to optimize the performance. Synchronization techniques must also be introduced to present the render result into its share of the graphical user interface real estate every time a requested render is completed. This approach yields a Direct3D rendering subsystem in the application that acts more like an on-demand rendering server.

4. PERFORMANCE TESTS RESULTS

Besides the library that implements our Direct3D solution, another one using GDI+ was created. This technology is the modern version of the classic immediate mode graphics library known as Windows Graphics Device Interface which has been available in Microsoft Windows since its first versions [22]. While Microsoft has put a lot of effort over the years to evolve it accordingly to the changes in the hardware, it has been finally deprecated by Direct3D as the main graphics technology for Windows. However, because of the complexity of the Direct3D API, GDI+ is still a widely adopted solution for windowed applications and thus, it has been employed to establish a comparison with the proposed solution.

A test application was created to measure the times required to draw the different datasets with both technologies. This application loads the datasets and then uses both the GDI+ and Direct3D solutions in order to visualize the datasets in a window. Figure 4 shows a capture of such visualization for the Galicia dataset. For each dataset, the application measures the time required to draw the power lines as fixed-width lines. Using fixed-width lines ben-

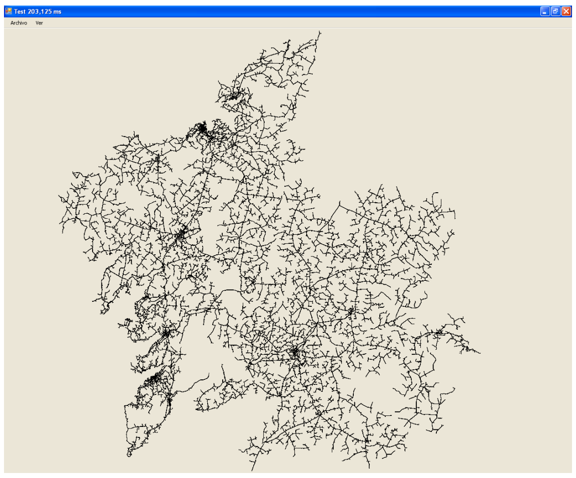

Fig.4: *Visualization of the Galicia dataset*

efits GDI+ since changing the width involves state changes, which have a great impact on performance. In the case of Direct3D, changing the width does not involve any extra operation or state changes and, therefore, the performance is not affected by varying widths. To be able to evaluate the complexity added by generating the wide lines, the times needed to perform the same task using only simple lines – i.e. with unitary-width – are also shown.

Several tests were executed in a workstation equipped with an Intel Core2 Q6600 2.4 GHz CPU, 2 GBs of memory, a nVidia GeForce 8500 GT GPU with 512 MBs DDR2 video memory and Windows XP SP3 operative system. This workstation has been chosen keeping in mind that workstations used by the utilities do not usually have cutting-edge hardware. As a result, both the graphic card and the operative system constrained the supported version of Direct3D to 9.0c.

Table 3: *Dataset rendering times (in milliseconds)*

Dataset	**Unitary-width lines**		**Wide lines**	
	GDI+	Direct3D	GDI+	Direct3D
Moldova	265.6	62.5	656.3	156.3
Nicaragua	1,359.4	78.1	2,890.6	203.1
Panama	375.0	93.8	1,203.1	218.8
Guatemala	531.3	101.6	1,890.6	250.0
Galicia	2,265.6	117.2	4,062.5	265.6
Central Spain	3,093.8	140.6	5,578.1	421.9

The results in Table 3 show that the Direct3D implementation is between 4.20 and 16.25 times faster than the GDI+ version for the selected datasets. Furthermore, the more primitives that must be drawn the bigger the gap between both technologies gets: Direct3D scales much better, as can be seen in Fig. 5. This figure shows that GDI+ drawing times do not exhibit a linear increase as the number of primitives grows. This is due to the different averages of lines composing the branches of each dataset. For instance,

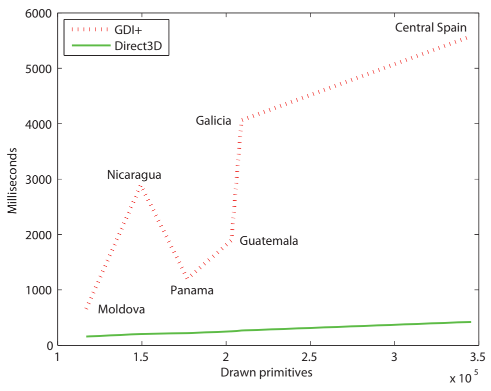

Fig.5: *GDI+ and Direct3D drawing times*

the Nicaragua dataset has an average of only 1.56 lines per branch while for the other datasets this factor goes over 2. This factor influences the optimizations that are automatically performed by GDI+ by reusing common points in the same way our Direct3D implementation does with vertices. Unlike GDI+, Direct3D performance is only influenced by the volume of the data and not by how this data is organized.

5. CONCLUSION

In this paper, we present an approach to enable a power grid planning CAD application to use the processing power available in modern GPUs. The implemented solution has used Direct3D to manage the programmable rendering pipeline found in these GPUs. Results show that it performed an average of 10.09 times faster for the selected datasets than a reference implementation using GDI+ which is a simpler but very widespread fixed-function rendering pipeline technology. Furthermore, the higher the data volumes, the better Direct3D performed compared to GDI+.

Vertex shaders have been used not only to process the graphical primitives more efficiently but also to translate the GIS coordinates of distribution network nodes to the graphic domain. Therefore, the geographic processing is carried by the GPU instead of the CPU resulting in an overall performance boost and better responsiveness of graphical user interfaces. This approach, which uses the high-performance parallel architecture present in modern GPUs for non-graphical applications, follows the GPGPU trend. The vertex processing performance is highly dependant on the number of vertex units present in the hardware but these details are hidden by the graphics APIs. As a result, applications using shaders scale with the hardware without any tweaking or code changes.

While this work is focused on the use of the programmable pipeline to enhance distribution electric grid visualization, other optimizations should also be introduced such as tiling or level of detail techniques.

The latter is a current line of work: using a multiscale architecture in spatially-enabled databases to pre-process the data and adjust the number of primitives required by the visualization of each different scale. Additionally, as soon as workstations equip fourth-generation graphics hardware, visualization applications should use technologies supporting geometry shaders such as Direct3D 10 or OpenGL 3.2.

The performance gain derived from using programmable GPUs allows CAD applications to offer real-time interaction with high-volume data visualizations such as electrical distribution networks. Furthermore, these applications have their performance automatically boosted whenever the graphics hardware is upgraded. As the GPUs available in workstations evolve, technologies like Direct3D and OpenGL which take advance of their increasing programmability features are becoming the standard for any application requiring high-performance visualization.

References

[1] P. M. Mahadev and R. D. Christie, *Case Study: Visualization of an Electric Power Transmission System*, Proceedings of the conference on Visualization, pp. 379–381, 1994.

[2] T. J. Overbye and J. D. Weber, *New Methods for the Visualization of Electric Power System Information*, Proc. IEEE Symposium on Information Visualization, pp. 131–136, 2000.

[3] P. C. Wong, K. Schneider, H. Foote, G. Chin Jr., R. Guttromson and J. Thomas, *A Novel Visualization Technique for Electric Power Grid Analytics*, IEEE Transactions on Visualization and Computer Graphics, vol. XV, pp. 410–423, 2009.

[4] T. J. Overbye, E. M. Rantanen and S. Judd, *Electric Power Control Center Visualization using Geographic Data Views*, Bulk Power System Dynamics and Control-VII. Revitalizing Operational Reliability, 2007 iREP Symposium, pp. 1–8, 2007.

[5] T. J. Overbye, *Wide-area power system visualization with geographic data views*, IEEE Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, pp. 1–3, 2008.

[6] T. Mohar, K. Bakic and J. Curk, *Advanced Planning Procedure and Operation of Distribution Network Supported by SCADA and GIS* IEEE Power Engineering Society Winter Meeting, vol. IV, pp. 2780–2785, 2000.

[7] P. A. Parikh and T. D. Nielsen, *Transforming Traditional Geographic Information System to Support Smart Distribution Systems*, Power Systems Conference and Exposition, pp. 1-4, 2009.

[8] Y. Liu and J. Qiu, *Visualization of Power System Static Security Assessment Based on GIS*, 1998 International Conference On Power System Technology, vol. II, pp. 1266–1270, 1998.

[9] J. J. Burke, *Power Distribution Engineering: Fundamentals and Applications*, Marcel Dekker Inc., pp. 1–27, 1994.

[10] R. E. Brown, *Electric Power Distribution Reliability*, Marcel Dekker Inc., pp. 1–38, 2002.

[11] P. Hines, S. Blumsack, E. C. Sanchez and C. Barrows *The Topological and Electrical Structure of Power Grids*, Proc. 43rd Hawaii International Conference on System Sciences, 2010, in press.

[12] P. Crucitti, V. Latora and M. Marchiori, *A topological analysis of the Italian electric power grid*, Physica A: Statistical Mechanics and its Applications, vol. 338, pp. 92–97, Elsevier, 2004.

[13] V. Rosato, S. Bologna and F. Tiriticco, *Topological properties of high-voltage electrical transmission networks*, Electric Power Systems Research, vol. 77, pp. 99-105, Elsevier, 2007.

[14] Z. Wang, R. J. Thomas and A. Scaglione, *Generating Random Topology Power Grids*, Proc. 41st Annual Hawaii International Conference on System Sciences, p. 183, 2008.

[15] *World Development Indicators database*, World Bank, September 2009.

[16] J. Novo Rodrguez, *Diseño e implementación de un motor de dibujo acelerado por hardware para una herramienta de planificación de redes elctricas*, Master's Thesis, Department of Computer Science, University of A Corua, 2009.

[17] D. Luebke et al., *GPGPU: general-purpose computation on graphics hardware*, Proc. ACM/IEEE conference on Supercomputing, pp. 208–241, 2006.

[18] D. Luebke and G. Humphreys, *How Gpus Work*, Computer-IEEE Computer Society, vol. 40, pp. 96-100, 2007.

[19] E. Lindholm, M. J. Kilgard and H. Moreton, *A User-Programmable Vertex Engine*, Proc. 28th annual conference on Computer Graphics and Interactive Techniques, pp. 149–158, 2001.

[20] T. Lorach, *Fake Volumetric lines*, NVIDIA Corporation, 2005, unpublished.

[21] D. Blythe, *The Direct3D 10 System*, International Conference on Computer Graphics and Interactive Techniques, pp. 724–734, 2006.

[22] M. Chand, *Graphics programming with GDI+*, Addison-Wesley Professional, pp. 1–13, 2004.

Javier Novo Rodrguez received the B.Eng. and M.Sc. in Computer Science degrees from University of A Corua, Spain, where he has been working in the *LIDIA* R&D group for the last 3 years under a contract with the power utility *Gas Natural Fenosa*, developing Computer-Aided-Design solutions for Power Grids planning, mainly in the fields of computer graphics and visualization. He is currently writing his Ph.D. dissertation in Computer Science.

Mariano Cabrero Canosa graduated in Computer Science from the University of A Coruña, Spain, in 1992. In 1998, he received his Ph.D. degree for work in the area of intelligent monitoring systems applied to intensive care. He is currently *Profesor Titular de Universidad* in the Department of Computer Science, University of A Coruña, Spain. His main current research areas are Expert Systems, Intelligent Monitoring Systems and Multiagent Systems with application to medicine and education. It also works in specific applications in the area of patient traceability with RFID technology and computer graphics and visualization in power grids.

Elena Hernndez Pereira graduated in Computer Science from the University of A Coruña, Spain, in 1995. In 2000, she received her Ph.D. degree for work in the area of the application of Artificial Intelligent techniques to sleep apnea diagnosis. Her main current research areas are: Expert Systems, Machine Learning, and Validation and Usability of Intelligent Systems. She is currently a *Profesora Contratada Doctora* in the Department of Computer Science, University of A Coruña, Spain.