Real-time Image Based Rendering Using Limited Resources

Esmir Pilav*
Belma Ramić-Brkić†

Sarajevo School of Science and Technology
Sarajevo / Bosnia and Herzegovina

Abstract

We are living in an age when computers are becoming powerful enough that real-time graphics are experiencing fewer computational issues. One of the areas where limited computational power still prevails is that of hand-held devices which usually have very limited memory and computational power. In this paper, we present a solution for a real-time data visualisation application. The application is to be applied to a hand-held medical device that assesses patient's needs for further examination in the case of concussion, stroke or other brain disorder. The scalp electric potential difference is measured via electrodes and is visualised on the display of the device. In this paper we describe the first stage in the development of a 3D visualisation solution for this device.

Keywords: Real-time CG, Image based rendering, Data visualisation, Optimization

1 Introduction

A number of authors have explored visualisation with respect to devices with the problem of limited resources. In [7] authors explored how to accurately display computer graphics application colours on different devices. In [6] authors developed an optimized Linux-based system whose capabilities are in line with the requirements for our project. A framework for rendering 3D graphics on a hand-held device is presented in [3].

The advance of technology has given rise to widespread use of hand-held devices, in particular, the use in medical diagnostics ([4]). It is important to be able to determine the type and severity of injuries immediately. Equipping ambulances with different devices, improves the ability of medical professionals to act in the best possible way and improves the victim's chances of quick recovery.

The particular device addressed in this paper is displayed in Figure 1. It is somewhat bigger that an iPod, is battery operated and provides immediate information to carry out timely medical intervention in cases of brain disorders, such as stroke or concussion. It can be used by physicians or nurses, Emergency Medical Teams, military medics and other first responders. It is not intended to determine the final diagnosis, but can assist in making triage decisions.

It measures the difference in a electroencephalogram(EEG) and a magnetoencephalogram(MEG) at different points on the scalp. This information describes the approximate electric neuronal activity at different points on the scalp. It is very useful to medical personnel but its usefulness is also determined by the quality of information visualisation.

Figure 1: Hand-held device for brain function assessment

The solution that was used on a full size static device applied LORETA([10]) imaging to solve the visualisation problem. The full size device uses 19 EEG electrodes and a PC for computation. The neural activity is estimated and a great efforts are made to preserve its original location. The activity was described with a blurred coloured image of the location of the head where the activity was measured. The colour depended on the difference of EEG and MEG measurements. This system is too large to be installed on a hand-held device. Hand-held device uses only 6 electrodes and has a much smaller computational power.

*esmirpilav@yahoo.com
†belma.ramic@ssst.edu.ba

Also, the hand-held device is meant to be used for immediate but not for final decision making. The visualisation solution for the full-size device (in Figure 2) was used in our work to make an assessment of the approximate positions of the 6 measuring electrodes to be use on a hand-held device.

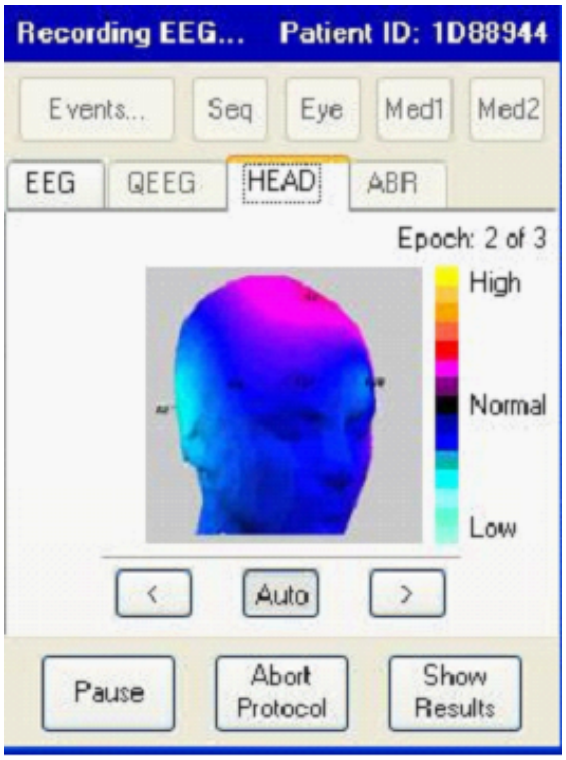

Figure 2: Loreta image of a 3D head with electrodes

In the first phase of development we concentrated on maximizing the computer graphics quality of the application.

We started with a 3D model of the head([12]), which was then divided into sectors. Every sector was in charge of presenting information from one electrode. In the rest of the paper, the terms *sector* and *cluster* are used interchangeably. According to the strength of the difference in EEG and MEG signals reported by the electrode, the sector colour needed to be changed. The number of colours which each sector can take will be definite and specified in advance. Every sector would be given a chance to change its colour regardless of other sectors, but according to the strength of the signal received from the electrode(Figure 3). To the end user, the interface will offer the opportunity of seeing all sectors from different viewpoints.

One simple solution for this problem is to pre-compute all the possible images and then, once the values from electrodes are read, display an image in accordance with these values. There are 128 different values coming from each electrode and, having in mind that there are 6 electrodes on the handheld device, the number of data visualisation combinations is 128^6. To compute the number of images that need to be pre-computed and stored we also need to multiply this number with the number of views for each data visualisation. After computing these numbers it is obvious that we cannot pre-render all the combinations; we have to do some real-time rendering.

The application was intended to run on the Linux operating system and to be programmed in C programming language with the OpenGL ([13],[2]) library. Since the device has limitations concerning memory size and computational power, it is essential to obtain the highest possible optimization in reading the 3D model of the head and appropriate sectors for each view.

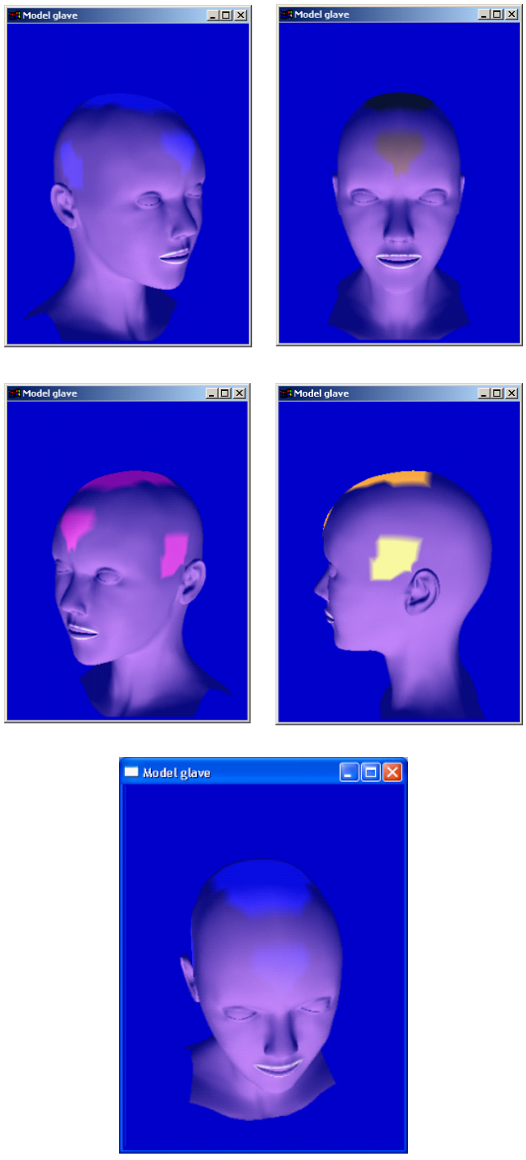

Figure 3: Example views with sectors of interest

2 Background

To develop an application which will fulfil the requested conditions, it is necessary to use an appropriate 3D model. There are various formats for storing 3D objects. A view of the head model used in the application is given in Figure 4. In the future, we plan to optimize this model before putting it into our rendering pipeline(Figure 5).

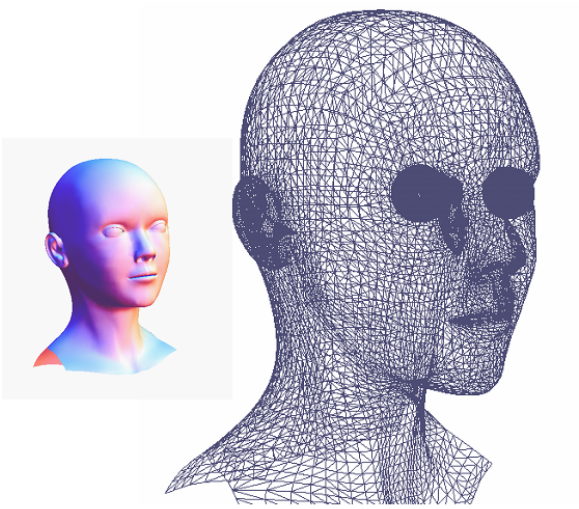

Figure 4: 3D head model

For this application, we used the .obj format ([11]), which is actually a text file that stores information about edges and the connections between them. We chose this format since it contains just enough information needed in this case. This is important to ensure the optimal final solution. To read the model, we needed to develop a function which would also be used later for reading and drawing individual sectors over image views.

We have already mentioned that due to space limitations we had to do some real-time rendering, but due to computational limitations, we also needed to limit the real-time rendering as well. To optimize this time-space problem we came to a compromise. Since we need only a few views of the 3D model of the head, we applied the following solution: we extract sectors in individual .obj files from the 3D model, and for all the views of the head used images which were pre-processed according to the viewpoint specifications.

The underlying images are to be changed from one view to another while sectors are read from .obj files and coloured according to the input data in real time. The sectors are also appropriately adjusted (rotated) for each view (Figure 6).

Since the head model alone is about 37 MB, and has a few thousand of vertices and polygons, reading and displaying it is much slower than showing all 6 sectors on the head model combined with view images. Another important choice was the format in which the images are stored.

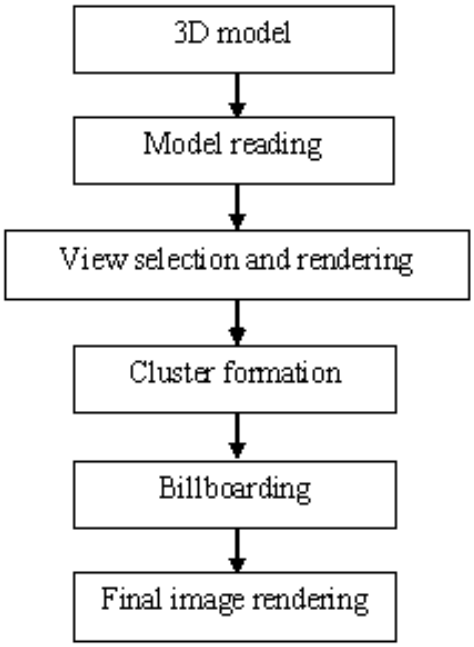

Figure 5: Rendering pipeline

For this application we used the .tga image format([1],[9]).

One view image of 400 x 300 with a .tga extension takes up about 315 KB. Taking all images in consideration, about 2 MB of storage is needed. In the further development of the application, we shall need to do further optimizations and possibly change the .tga format with another format which occupies even less memory space.

2.1 Visualisation Method

Even though we started with a 3D model in which we wanted to visualise the data, the final desired product of our application is a series of images. Since we needed a computationally efficient application, we decided to use image-based rendering (IBR). The advantage of using this paradigm is that computational complexity is measured in the number of pixels that need to be present in the image, and not, for example, in the number of vertices in the geometrical model.

In particular, the IBR technique used is billboarding ([8]). In this technique the image is rendered onto a polygon facing the viewer. The positioning of that polygon regarding the viewpoint is called billboarding. The polygon that is rendered is called a billboard. If we want to represent a different view, we need to reposition the polygon.

There are several popular billboarding techniques. What is in common to all is that a surface normal (n) and an up direction (u) are determined to orientate the polygon which is usually a quadrilateral. When these two vectors

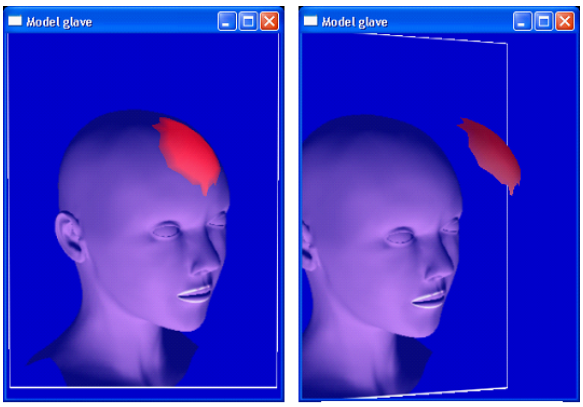

Figure 6: Sector and rotated sector combined with an image

are found, we can create an orthonormal basis for the surface. That is basically saying that with these two vectors, we have enough information to describe the rotation matrix needed to rotate the polygon to its final position. The position of the quadrilateral is taken to be its centre and is noted as an anchor location. The desired surface normal n and up vector u are often not perpendicular.

One of these two vectors is designated as a fixed vector. It needs to be maintained in the given direction. We then need the other vector to be perpendicular to the fixed vector. First, a vector pointing toward the right edge of the quadrilateral is created and designated by r. This vector is a cross product of u and n.

Since this vector is going to be used as an axis of the orthonormal basis for the rotation matrix, it needs to be normalized. If r has length zero, u and n must be parallel and another technique needs to be used.

The unfixed vector (the one that remains - either n or u) is modified by taking the cross product of the fixed vector and r. This creates a vector that is perpendicular to both.

If we fix the normal n (as most billboarding techniques do), then the new up vector u' is:

$$u' = nxr$$

This process is shown in Figure 8. If, instead, we fix the up direction (true for axially aligned billboards such as trees on landscape), then the new normal vector n' is:

$$n' = rxu$$

The new vector normalized, together with n and u, is used to form a rotation matrix. For a fixed normal n and adjusted up vector u' the matrix is:

$$M = (r, u', n)$$

This matrix is used to transform a quadrilateral in the xy plane to the appropriate orientation. Then a translation matrix is applied to move the quadrilateral's anchor point

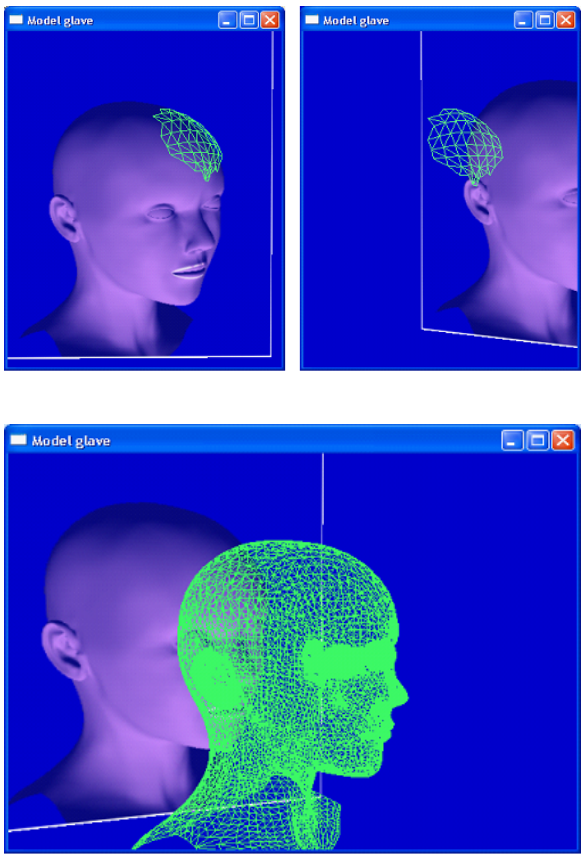

Figure 7: Combining sector network with images using rotation

to the appropriate location. The remaining task is then to decide what surface normal (n) and up vector (u) are used to define the billboard's orientation. Based on this decision different billboarding techniques are defined. The two that we used are *Screen-Aligned Billboard* and *World-Oriented Billboard*.

Screen-Aligned Billboard is the simplest form of billboarding. The surface normal determined here is the negation of the view planes normal. The up vector u is a vector in the view plane that defines the cameras up direction.

In World-Oriented Billboarding, the normal n is still the negation of the view plane normal, but up vector u is world up vector.

3 Implementation and Results

To develop this application we used combined Screen-Aligned and World-Oriented billboard techniques. A fixed number of views was chosen. Those views were then used to construct the billboard of the head model together with the appropriate sector. Constructed rotation matrices were used to position the sectors appropriately.

The functions for reading the model and manipulate sectors and combining them with the appropriate model view

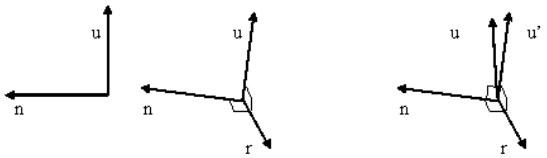

Figure 8: Given a desired surface normal direction n and an approximate up vector direction u, we wish to establish a set of three mutually perpendicular vectors to orient the billboard. In the middle figure, the ”right” vector r is found by taking the cross product of u and n, and so is perpendicular to both of them. In the right figure, the fixed vector n is crossed with r to give a mutually perpendicular up vector uf.

are programmed. The function for the display of the final image was also programmed.

It was also necessary to create an engine which rotates and adjusts the coordinate system with the model coordinates. This engine also fetches the appropriate views of the head model for later use.

Another function that is implemented groups a set of individual triangles or faces into one sector. Since the model has 26944 triangles, the manual selection of triangles would have been impossible. The only way to do this successfully is to automate this process. For this purpose, we created an application which, for a chosen set of triangles that form the centre of a sector, selects the faces neighbouring the previously selected faces by recursion. The faces that form the centre were selected beforehand. With this form of selection, it is necessary to set an additional variable, which would indicate at which level of the selection to stop. The selected faces form a cluster. In order to get a smooth transition between the parts of the head model and the coloured sectors that are added to the head, given clusters need to define edges which form a border. In order to construct the border, we had to take into account the breadth of each face sector (Figure 9). Based on the information of sector breadth, each vertex common to a sector and the rest of the model were coloured with the colour of the model. This ensured smooth transition between areas with and without coloured sectors.

This approach had a few flaws due to program speed and reactions to events from the outside world in real time. To enable optimization in this case (since a definite number of model views suffice the visualisation), images of the model from a definite number of viewpoints are extracted. The model now becomes static and the only thing that changes is the designated sectors. This makes an excellent optimisation since about 90% of the data becomes pre-computed.

To accomplish this idea, in the pre-computation stage, sectors were extracted into separate .obj files. This was accomplished by a function that selected levels of triangles with an algorithm mentioned above. The extracted sectors occupy a few hundred kilobytes of memory. This is a significant memory saving, compared to the memory required to store the entire model.

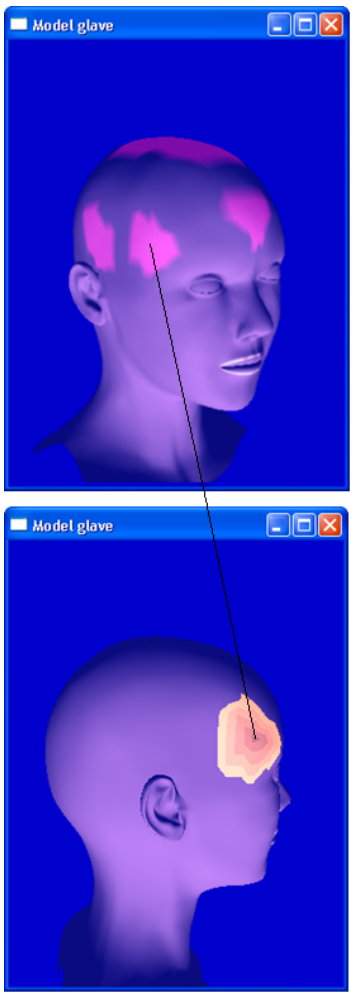

Figure 9: a) The result after adding a separated sector to already existing ones. b) The result of implementing a function for separating a sector with a given centre and breadth. A separated mesh is composed of more levels base on distance from the sector’s centre which are later used for gaining smooth transition between view image and the sector.

In order to combine head images from specific viewpoints with coloured sectors, we needed to remember all relevant parameters that define a given viewpoint. When moving from one viewpoint to another, the background image of the head model related to that viewpoint is read. Then the parameters which define that viewpoint, as well as the sector position, are set.

The application at this project stage displays the head model with five viewpoints and four defined sectors with electrode centres. Sectors are displayed dynamically. Colours that define the sectors are defined and their number is fixed.

To implement these functionalities we used the C++ programming language because of greater flexibility and ability to use STL ([5]) library. To manipulate the graphical objects, we used the OpenGL library routine. To manipulate the windows, we used the glut library which can be used both on Windows and Linux operating systems. The only thing that needs to be changed is the process of compiling a different executable for a different operating systems.

4 Conclusions and Future Work

In this paper we presented an approach for medical data visualisation on the 3D model of the head. The approach is designed for hand-held devices that have limited memory and computation power. The solution that such devices had been using thus far used only pre-computed images and did not have the ability to view the visualized data from several points of view.

In the second stage of the solution development we plan to concentrate on testing this application on the actual device and coming up with further optimization solutions. One image compression algorithm that can be used to further reduce the memory space needed for this application is JPEG ([1],[9]).

In the second stage of the project, we also need to take into closer account the following measurements in order to optimize the application even further: display dimension, display resolution, essential colour of the head model, the number of possible sector colours, the position of the head model on the display, better approximation of sector centres, approximation of preferable sector breadth, the input data interface.

With this additional data it is possible to carry out further optimization in the sense of: reducing the amount of space used by images of every viewpoint; decreasing the speed of the application's reaction to the change of input parameters; and reducing the space occupied by the application itself.

5 Acknowledgement

We would like to thank the INSPIRE D.O.O Company in cooperation with BrainScope Inc. funded this research as the part of the Beta Prototype of BrainScope ED Triage project.

References

[1] C.Wayne Brown and Barry J. Sheperd. *Graphics File Formats: Reference and Guide*. Manning Publications, 1995.

[2] Samuel R. Buss. *3D Computer Graphics : A Mathematical Introduction with OpelGL*. Cambridge University Press, 2003.

[3] Chun-Fa Chang and Shyh-Haur Ger. Enhancing 3d graphics on mobile devices by image-based rendering. In *PCM '02: Proceedings of the Third IEEE Pacific Rim Conference on Multimedia*, pages 1105–1111, London, UK, 2002. Springer-Verlag.

[4] Dan F. Criswell and Michael L. Parchman. *Handheld Computer Use in U.S. Family Practice Residency Programs*. Journal of American Medical Informatics Association, 2000.

[5] Nicolai M. Josuttis. *The C++ Standard Library: A Tutorial and Reference*. Addison Wesley, 1999.

[6] Narayanaswami C.and Kamijoh N.and Raghunath M.and Inoue T.and Cipolla T.and Sanford J.and Schlig E.and Venkiteswaran S.and Guniguntala D.and Kulkarni V.and Yamazaki K. *IBM's Linux watch, the challenge of miniaturization*. Computer, IEEE Computer Society, 2002.

[7] Bruce J. Lindbloom. *Accurate color reproduction for computer graphics applications*. Proceedings of the 16th annual conference on Computer graphics and interactive techniques p 117-126, 1989.

[8] Tomas Moller, Eric Haines, and Tomas Akenice-Moller. *Real-Time Rendering*. AK Peters, 2002.

[9] James D. Murray and William van Ryper. *Encyclopedia of Graphics File Formats*. O'Reilly, 1996.

[10] R.D. Pascual-Marqui, M. Esslen, K. Kochi, and D. Lehmann. *Functional imaging with low resolution brain electromagnetic tomography (LORETA): a review.* Methods and Findings in Experimental and Clinical Pharmacology, 2002.

[11] Keith Rule. *3D Graphics File Formats: A Programmer's Reference*. Addison-Wesley, 1996.

[12] Sidow. *3D Woman's head*. http://www.turbosquid.com/ FullPreview/Index.cfm/ID/275671, 2007.

[13] Richard S. Wright, Benjamin Lipchak, and Nicholas Haemel. *OpenGL(R) SuperBible: Comprehensive Tutorial and Reference (4th Edition) (OpenGL)*. Addison- Wesley, 2007.