An Implementation of Curvilinear Reformatting on VTK
Wallace Loos, David Gobbi (Intern Supervisor) and Wu, Shin-Ting (Adviser)
Department of Computer Engineering and Automation (DCA)
School of Electrical and Computer Engineering
University of Campinas (Unicamp)
P.O box 6101, 13083-970 – Campinas, SP, Brazil
{wloos,ting}@dca.fee.unicamp.br
dgobbi@ucalgary.ca

Abstract – This report presents an implementation of curvilinear reformatting tool using VTK and OpenGL on the platform Mac. This task was performed in the Calgary Image Processing and Analysis Centre (CIPAC) at University of Calgary within the cooperation program between CIPAC and the Interactive Visualization Group (IVG) at Unicamp.

Keywords – VTK, Curvilinear Reformatting, Mesh.

1. Introduction
The curvilinear reformatting is a non-invasive tool that allows exploring the human brain. It consists in peeling the brain in the direction perpendicular to the scalp [8]. The utility of this tool for analyzing the cortical surface and the opportunity of learning a new development environment made us to set as a one-time project its implementation on top of VTK and OpenGL running on OS X Mavericks/Mac during my three month Internship Program at the Calgary Image Processing and Analysis Centre (CIPAC).

An implementation of the curvilinear reformatting algorithm has been accomplished by the Interactive Visualization Group (IVG) at University of Campinas (Unicamp) [2]. It is built on top of the wxWidgets GUI library [10] and the open source Grassroots DICOM library is used for reading and parsing DICOM medical files [3]. GPU resources are extensively explored for real-time interactive renderings. The C++ code is runnable on both Windows and Linux platforms which are the major brazilian desktop platforms. The software development environment at CIPAC differs greatly from the brazilian environment. Their desktops run Apple Macintosh and their applications are built on top of the cross-platform visualization toolkit VTK [9] and the Qt GUI library [1].

This short report details a learning experience at CIPAC for migrating our curvilinear reformatting implementation to the Mac platform. It is organized as follows. A brief description of the algorithm and some differences between the implementation are described in Section 2. Result of the implementation is presented in Section 3. Finally, the conclusion and future works are presented in Section 4.

2. Curvilinear Reformatting Algorithm
Curvilinear reformatting algorithm can be divided into four pre-processing steps:

1. to select the region of interest (ROI) on the scalp;
2. to build the mesh from the samples of ROI;
3. to build a series of intersection-free offset meshes from the mesh;
4. to voxelize the offset meshes and the voxels are tagged with their depth with respect to the scalp.

A user selects ROI by brushing with mouse on the head’s surface (Figure 1(a)). When the interaction is finished, the region is sampled (Figure 1(b)) and a mesh is built. The mesh is a collection of vertices, edges and faces that approximate the scalp’s geometry. It plays an important role in the curvilinear reformatting, because it allows us to apply known mesh oriented geometric procedures for computing intersection-free laminar layers. A boundary-based data structure is employed for managing the mesh topology.

In fact, the mesh is displaced in the direction of the normal vector until a previsouly specified depth is reached. And for each displacement it is voxelized in the same resolution of the raw brain volume. The voxelization is used to find the voxels which the mesh intersects, and these voxels are tagged with their depth relative to the scalp. The voxelization algorithm is an adaptation of the depth map based algorithm proposed in [5]. We explore
the framebuffer object (FBO) architecture available in OpenGL [7] to off-screen render the depth maps of a mesh from three orthogonal views and to reconstruct from these maps the 3D coordinates of the mesh.

At the end we have a set of depth-tagged voxels from which we may modify the visibility of regions of a data volume and create the visual effect that a curvilinear reformatting is performed on the brain data. Because of distinguishing infrastructures, the way that these depth-tagged voxels are organized at Unicamp for displaying curvilinear reformed volume should be changed in the CIPAC. In the following subsections we explain the reasons and how these changes were performed.

2.1. IVG’s Implementation

The tagged voxels are organized as a selection volume [4] and this selection volume is represented by a 3D texture similar to that for the brain volume. Each entry corresponds to the voxel’s depth level. Exploring the multitexturing facilities of GPUs, both the original data volume and the selection volume are sent to GPU. The standard rendering algorithm has been slightly modified by including a comparison of the depth value from the selection volume and the user-defined depth value in a fragment shader. If a voxel has depth value less than the user-defined one, the voxel is considered invisible and its value does not contribute to the final fragment color. A slider widget is employed for interactively controlling the cropping depth.

A requirement for porting the curvilinear reformatting algorithm to the image analysis and data management environment in the CIPAC is to explore all facilities already available in the VTK library. VTK supports several volume rendering techniques including the standard GPU-based ray-casting one. How to adapt in a three-month internship program the IVG’s implementation to the VTK-based implementation? The answer to this question is presented in the next subsection.

2.2. Implementation in CIPAC

Under the infra-structure constraint, three problems arose after a careful analysis: management of mesh topology, creation of the FBO-based depth maps, and reuse of the GPU-based ray-casting rendering available in the VTK library.

Although it is easy to port the module that manages the mesh topology, we decide, for learning purpose, to use the 2D Delaunay triangulation class available in the VTK library. From a list of input samples of the painted samples, a 2D Delaunay triangulation is built and further processings can benefit from the underlying data structure.

To reuse the voxelization algorithm implemented by IVG, we should create FBO. Although VTK supports FBO, no researcher in the CIPAC had experienced its usage. Hence, we decided to directly access FBO via the GL API as in the original Linux/Windows implementation. The problem is thus reduced to the creation of an OpenGL graphics context on a Mac platform. A quick search among the references shows that the MacOS X operating system offers Apple-specific OpenGL APIs for creating OpenGL rendering contexts and for associating them with the OS X windowing system [6].
only need to take care about the context integrity when we switch between the VTK and OpenGL contexts.

The GPU-based ray-casting algorithm available in the VTK is a standard one. It does not access an extra user-specified texture but the brain data volume. To overcome this limitation, we store the tagged voxels as an adjacency list on the CPU side and use them to “retouch” the brain data volume that is sent to GPU. The retouching consists simply in setting to zero the tagged voxels.

3. Results
Two curvilinear reformattings of a data volume are shown in the Figures 2 and 3. In Figure 2 the cropping mesh has 546 faces, while in Figure 3 the mesh size is of 1121 faces. Table 1 presents the time spent to perform the construction of offset meshes and their voxelization. The evaluation platform was a desktop Intel Core i5 3,2GHz CPU with 8GB RAM 1600MHz DDR3 and a NVIDIA GeForce GTX 675MX with 1024 VRAM. Note that these times are dependent on the mesh size.

The complexity to traverse the adjacency list is $O(D + V)$, where $D$ is the maximum depth of the crop and $V$ is the number of labeled voxels. In other words, the complexity to traverse the adjacency list is linear and the interactivity of the curvilinear reformatting is ensured even the retouching occurs on CPU and the brain volume must be resent to GPU.

4. Conclusion and Future Works
A learning experience in the CIPAC has shortly reported. In three months the programmed internship task has been successfully accomplished. A version of curvilinear reformatting algorithm has been implemented on top of VTK in the Mac platform. During this period we contribute with an implementation of a curvilinear reformatting algorithm developed in our laboratory, while the CIPAC contributes with the knowledge in the software development on the Mac platform.

This internship program provides us a good opportunity to learn from practice the difference between building an application with VTK, as in the CIPAC, and building one with a set of basic data and graphics management tools available in separate libraries, as in the IVG. VTK provides a wide range of algorithms that satisfies demand of most applications. The great advantage of using VTK is saving the development time. Once it is based on an object-oriented programming model, most of concepts and implementations that are difficult to understand are hidden behind the prebuilt objects. Therefore, no detailed knowledge or skills are required to construct a standard application.

However, the VTK’s programming approach does not offer us flexibility in creating completely novel solutions as the curvilinear reformatting algorithm. Combining our own code with VTK can be very challenging, because it requires that graphics pipelines are re-written according to VTK’s rules. Because of steep learning curve for coding with these rules in mind, we constructed “artifacts” among the VTK’s prebuilt objects to make it works as we expected: multiple transfers of the brain volume to GPU for ray-casting rendering and another graphics context to have immediate-mode access to the framebuffer objects for efficient off-screen rendering.

Since we are always looking for new technologies and needing complete flexibility for evaluating different ideas we plan to keep the raw material development approach in IVG. Nevertheless, facing the increasing number of Mac desktops, we plan to include the Mac in our cross-platform code development in mnext code revision.

References


[4] Markus Hadwiger, Joe M. Kniss, Christof Rezk-salama, Daniel Weiskopf, and Klaus En-

<table>
<thead>
<tr>
<th>Mesh (number of faces)</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>546</td>
<td>0.296014</td>
</tr>
<tr>
<td>1121</td>
<td>0.488908</td>
</tr>
</tbody>
</table>

Table 1. Curvilinear Reformatting’s timing.
Figure 2. Curvilinear reformatting.

Figure 3. Curvilinear reformatting.


