Image Display System

Hassan Hanino

Table of Contents

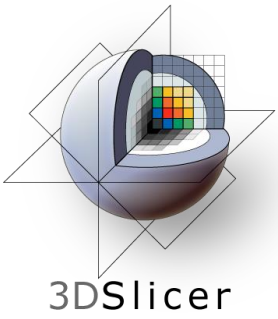
Introduction and Background

This software focuses on creating a simple **image display system** which examines and manipulates raw data of a volumetric image (a series of 2D frames). It first loads and then renders raw data of a volumetric image of a head onto four panes for manipulation. Manipulations currently include: rotation, alpha blending, a change in depth from which it is viewed, zooming in and out, applying a thickness function, and changing the axis of rotation. This application consists of a basic foundation and functionality normally seen in DICOM viewers and 3D visualization systems which can navigate and manipulate medical images.

Here is an image of a modern 3D visualization tool for medical images made by Object Research Systems (ORS), also approved by Health Canada and the FDA.

The image viewing system is formed by combining a framework containing C++ code along with many different libraries. The main resource used to load and render the image is the API OpenGL (Open Graphics Library). OpenGL is a cross-platform and language application programming interface (API) for rendering 2D/3D graphics which is supported by hardware. A few real-world applications of OpenGL are not limited to these technologies but include: simulations, visualizations, virtual reality, and video games.

There are numerous reasons for using OpenGL to create an image viewing system, rather than using an intensive high level open-source package such as VTK (Visualization Tool Kit) or 3D Slicer. Even though these high-level packages are easy to use when implementing the same functionality as demonstrated in this project, they are not used commercially as they cannot be approved by the FDA (Food and Drug Administration) or Health Canada. The main reason is that there are rigorous guidelines and standards which any medical device deployed in the EU or US medical industry must obey. One of the most important standards which basically characterizes a medical device software is the IEC (International Electrotechnical Commission) 62304. This standard defines the life cycle of a medical device software which outlines everything between the development process to the maintenance stage once the software is deployed. There are many other standards that must be upheld for a software to be used in a medical device such as ISO 13485 (Quality System for medical devices), ISO 14971 (Risk Management), etc. Clearly, high-level packages such as VTK or 3D Slicer cannot meet these guidelines as both of their code were open-source since the 1990's. It would be extremely difficult to ensure every feature and update made since the creation of both programs passed all documented standards that are required for all medical software devices. Additionally, packages as such are too constraining as they are forced to use specific data structures and concepts for specific functionalities which limits the choices of customization, or even sometimes prevents the task from being accomplished due to such constraints. Although it is more difficult to use, OpenGL is used commercially within medical imaging systems. It is also much more flexible regarding what it can achieve, as it is able to draw complex geometric shapes through simpler geometric shapes which signifies OpenGL's drawing capability. Additionally, OpenGL is low-level as it depends on hardware support to render 2D/3D graphics, making it highly compatible with many platforms such as Windows, Linux, OS X (specific implementations of OpenGL are usually not cross-platform).



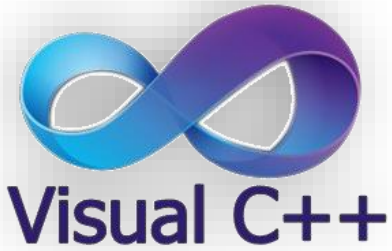
The code is written in managed C++ which is a based on the Common Language Infrastructure (CLI) language specification created by Microsoft that allows C++ code to be executed using in their .NET framework. The .NET framework contains a large library known as Framework Class Library which provides a graphical user interface (GUI) known as Windows Forms (WinForms) to navigate and manipulate the image.

Approach

Initially, I downloaded 3D Slicer examining the user interface, functionality, and documentation, to get a more precise idea as to what my display system must possess. The display system in 3D Slicer contains the basic functionality and controls I expected to create (four panes where each of them render a different point of view of the image, entries and buttons which specifies an angle of rotation at each axis, an alpha blending function, etc.), along with vast amounts of other functionalities that were beyond the scope of this project if it was to be implemented.

I began to learn and use OpenGL which was very challenging at first, but slowly learned the basics of the graphics library and what was required to simply render an image onto the screen. OpenGL is an evolving API that is regularly updated by the Khronos Group, expanding on the API to support new features. OpenGL is more capable with associated libraries such as OpenGL Utility Library (GLU) which provides useful features that were not incorporated as a functionality in OpenGL, such as tessellating (the use of 3D surfaces and curves are created by faces, edged, and vertices), mipmapping (manipulating level of detail through pre-calculated sequences of images that progressively have lower resolution), and higher-level drawing routines built on primitive types from OpenGL (vertices, lines, polygons). Clearly, the capability and customization OpenGL offers is significant.

The integrated development environment (IDE) chosen was Visual Studio. Visual Studio is one of the most used IDEs that is created by Microsoft. It is used to develop computer programs, websites, and mobile apps. It can produce native and managed code. The language of choice for this project was chosen to be C++, which I also had no experience with. It was challenging but valuable experience to learn a new API and a language at the same time (although, C++ is very similar to C and Java making it easier to transition to). As described before, managed C++ is a language specification created by Microsoft which allows developers to compile C++ code (that is slightly different in terms of syntax and memory allocation) with their .NET framework. Their .NET framework provides a way to create a GUI easily, specifically through Windows Forms in this case. Windows Forms is a GUI library included in .NET which provides a clean slate which can be enhanced with controls and code together to create a user interface when creating applications for desktop, laptop, and tablets.



Functionality

The application can load and convert raw data of a series of sequential 2D frames which is what the volumetric image is composed of, to RGBA data. A texture variable is then created to hold

the image data of the head which was created using the RGBA data. The texture variable can be referenced when the image data is needed to be drawn on the scene which can then be manipulated in various ways. Manipulations currently include: rotation, alpha function, a change in depth from which it is viewed, zooming in and out, applying a thickness function, and changing the axis of rotation/plane from which we view the head.

Rotation: The image can be rotated 360 degrees. Using quaternions in OpenGL, we are able to represent rotation from all 3 axis (x,y,z) and the angle of rotation which defines how much the image is to be rotated on a specific axis in a simpler way. Once the parameters are set, the quaternion is converted to a matrix which is then used rotate our image.

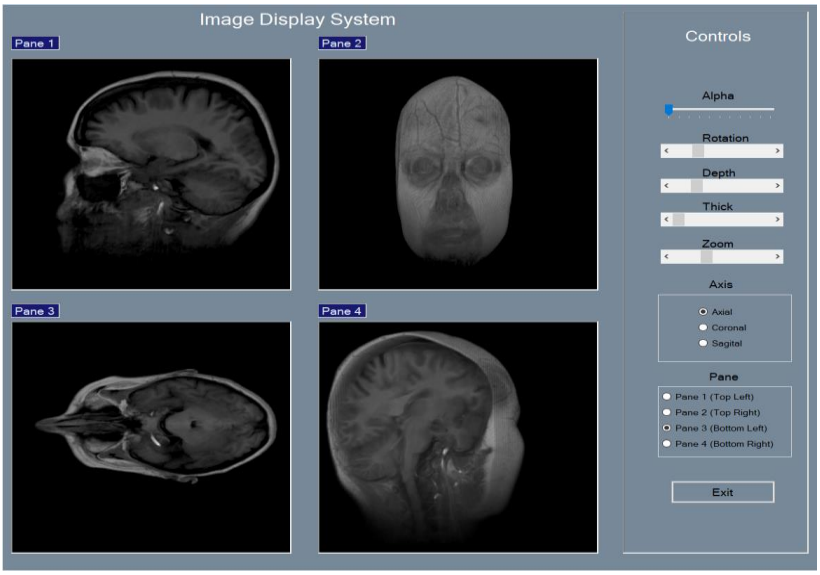
Alpha Blending Function: The alpha function provides an option to change opacity/translucency/transparency of a material by blending colours. OpenGL can enable blending of colours by enabling an alpha function.

Change in Depth: To examine the image at a different depth, a clipping plane is created using the specified depth value which can change the position of the camera/viewing frustum in the scene.

Zoom In/Out: The image can be zoomed in and out by applying a scaling function with a parametrized scaling value to the current image. The scaling function will increase/decrease the size of the image which mimics the ability to zoom in and out.

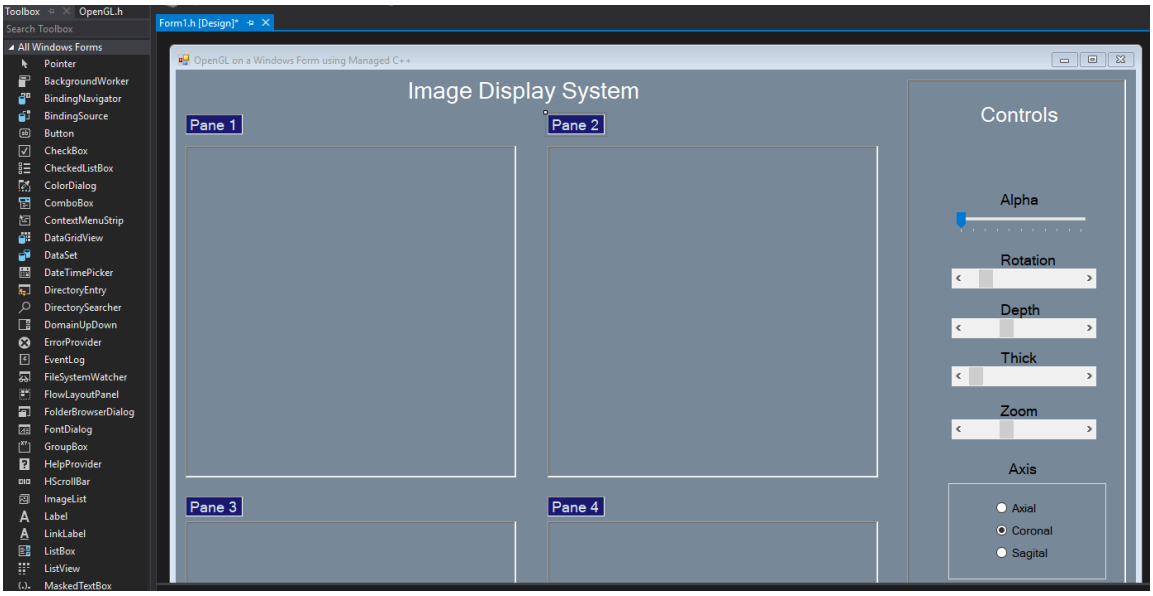
Thickness Function: The thickness function can be used to display parts of the image that exceed the threshold of a specified thickness. This is accomplished by incorporating the value of the required thickness to be shown into an equation of a plane (which is how OpenGL specifies a clipping plane) which all geometric shapes on that plane are clipped.

Changing Projection Plane: The plane from which we view the head can be changed. The three different projection planes are coronal, axial, and sagittal. This is accomplished using OpenGLs clipping planes.



Here is an image of my Image Display System.

Window forms are also used to create the GUI of the application. Windows Forms can be easily created in Visual Studio as it provides a form editor which is a visual surface to design your windows forms by adding and dragging features from Visual Studios toolbox. The toolbox displays icons for controls and other items that can be added into the windows form by simply clicking on it.



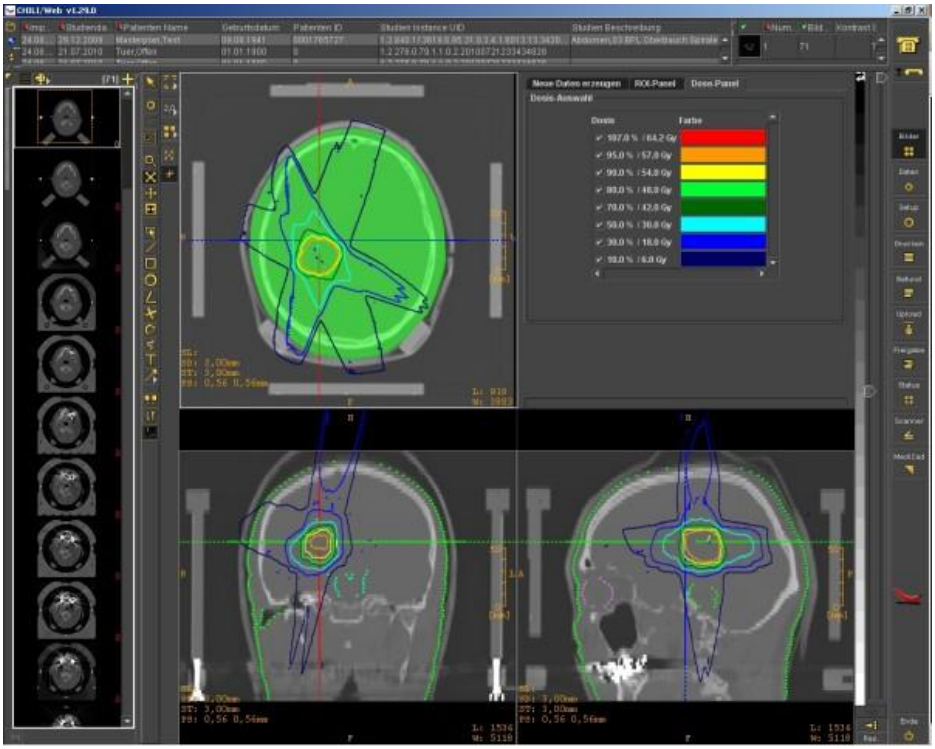
Visual Studios form editor which adds features from the toolbox shown on the left side.

Lessons Learned and Future Work

Unfortunately, the time needed to incorporate the functionality and basics of an image display system was much more than I thought. Learning OpenGL can be quite challenging as the learning curve is steep. Graphics programming is very different from the programming I have done through out my academic career. An extensive and complicated API that is mostly based off mathematical properties that can range in complexity is needed to set up the scene and apply certain effects. Most of my effort in this project was learning how OpenGL works and how to use it properly. There are strict routines to accomplish numerous types of image rendering, but to do it correctly requires a solid background as to how OpenGL works. To create the desired scene, understanding how to use the proper OpenGL functions with the correct parameters is essential as errors can easily occur if not careful. Although it was very challenging, the experience of using OpenGL to create an image display system was valuable. I had to adapt to understanding very different concepts and software I was used to using in school. Even though challenging, OpenGLs superb rendering capability caught my interest and motivated me to learn more of it.

Many image display systems can have plans created for a more thorough and effective examination on the rendered image which can then be saved. The planning consists of creating a point of interest that is defined by an (X, Y) coordinate, setting a trajectory within the 3D scene that can be set anywhere along the image, outlining an area of interest, and extracting a feature or an area of interest which is to be rendered on a separate pane for viewing and manipulation.

Additionally, I plan on adding a button to save the current state of the program which can be reopened with the same plan and manipulations applied from before, but this will require much more work and knowledge on OpenGL before it can be successfully implemented. Many more complex functionalities can be referenced from massive open-source image display and planning systems such as 3D Slicer, VTK, or MicroDicom (a massive free DICOM viewer for Windows).



ResearchGates DICOM viewer which can create points and areas of interests.

References

1. https://www.khronos.org/opengl/wiki/Primitive
2. https://www.slicer.org/wiki/Documentation/4.8/Announcements
3. https://www.vtk.org/Wiki/VTK/FAQ#What_is_the_Visualization_Toolkit.3F
4. https://www.iso.org/standard/38421.html
5. https://docs.microsoft.com/en-us/dotnet/framework/winforms/
6. http://nehe.gamedev.net/tutorial/creating_an_opengl_window_(win32)/13001/
7. https://www.khronos.org/opengl/wiki/Creating_an_OpenGL_Context_(WGL)
8. https://www.opengl.org/archives/resources/faq/technical/miscellaneous.htm
9. http://www.cse.chalmers.se/edu/year/2017/course/TDA362/redbook.pdf
10. https://www.brainlab.com/userguides/en/guides/dicom-viewer/en/3.1/idPDMG50b342a5/