Visualization of Plasma Edges using VTK and Adobe Flash

By

LAVESH KUMAR GUPTA

A thesis submitted to the

Graduate School-New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

written under the direction of

Prof Deborah Silver

and approved by

____

____

____

____

New Brunswick, New Jersey

May, 2011

Abstract of the Thesis

Visualization of Plasma Edges using VTK and Adobe Flash

By Lavesh Kumar Gupta

Thesis Director:
Professor Deborah Silver

Visualization is a very useful tool in the study and analysis of huge experimental or simulation datasets. It converts vast amount of numerical data into graphical images, extracts important hidden information and aids scientists in hypothesis building. This thesis focuses on migrating visualization module of Plasma Edge from commercial visualization software AVS to open source visualization library VTK. The VTK visualization module will be integrated in new plasma edge simulation code package at the Center for Plasma Edge Simulation for the study of plasma edge region relevant to both existing magnetic fusion facilities and next generation burning plasma experiments. VTK visualization module generates two set of images: 2D images representing cross-sectional view of 3D torroidal plane, and 3D images representing whole torroidal plane.

Two Adobe Flash modules were also developed using open source Adobe Flex to integrate the VTK visualization module with eSimMon dashboard, a front-end tool for simulation monitoring. Using these modules, virtual 3D visualization capability was provided in the dashboard for viewing generated plasma images.

Acknowledgement

I would like to thank my advisor, Prof. Deborah Silver, for her support and encouragement while writing this thesis. Also, I would like to thank my parents and family who provided me with a strong educational foundation and supported me in all my academic pursuits. I also acknowledge the help of VIZLAB at Rutgers.

Table of Contents

Abstract ii

Acknowledgement iii

Table of Contents iv

Table of Figures viii

1. Introduction 1

 1.1 Motivation for thesis 2

2. Scientific Visualization 5

 2.1 Data File Formats 8

  2.1.1 .vtk file format 8

  2.1.2 .obj file format 10

  2.1.2 HDF5 file format 12

 2.2 Scientific Visualization tools 12

  2.2.1 OpenGL 13

  2.2.2 VTK 14

3. Background 16

 3.1 Importance of Plasma Edge Research 16

 3.2 EFFIS Workflow 17

 3.3 eSimMon Dashboard 18

 3.4 Vizlab Contribution 20

3.4.1. 2D Time Visualization....................21

3.4.2. 3D Torus Visualization....................23

4. HDF5....................26

4.1 Need for a standard file format....................26

4.2 Data challenges solved using HDF5 technology....................27

4.3 HDF5 data organization....................30

4.4 HDFView – GUI tool to modify HDF5 files....................31

4.5 Data Files....................33

4.5.1. Mesh HDF5 File....................33

4.5.2 Intensity HDF5 File....................35

4.5 HDF5 Reader....................36

4.5.1. Information required for 2D time visualization....................36

4.5.2. Information required for 3D time visualization....................36

4.5.3. Flowchart for HDF5 reader....................37

4.6 Summary....................39

5. Visualization Toolkit (VTK)....................40

5.1. Introduction....................40

5.2 VTK Object Models....................41

5.2.1 Visualization Model....................42

5.2.2 Graphics Model....................43

5.3 VTK Visualization Modules 44

5.3.1. VTK module for 2D Time Visualization 44

5.3.2. VTK Module for 3D Torus Visualization..46

5.4. Summary 47

6. Adobe Flash..48

6.1. Introduction 48

6.2 Advantages of using Flex and Flash...49

6.3 MXML and ActionScript..51

6.3.1. MXML 52

6.3.2. ActionScript 53

6.4 Flex Modules..54

6.4.1. Flash module – 2D Time Visualization 55

6.4.2. Flash Module – 3D Torus Visualization..56

6.5. Summary 56

7. Results...57

7.1 Output of 2D Time Visualization Flash module 57

7.2 Output of 3D Torus Visualization Module 60

8. Conclusion 63

9. References 64

Appendix – I..66

Building VTK on Windows using CMake and Visual Studio 2008 66

Appendix II 69

Building HDF5 library on Windows using Visual Studio 2008 69

Table of Figures

Fig 1: The visualization process. 2

Fig 2: 2D Time Visualization 3

Fig 3: 3D Torus Visualization 4

Fig 4: Visualization of numerical dataset to an image. 5

Fig 5: Visualization pipeline 7

Fig 6: Overview of five parts of VTK data file format [7] 9

Fig 7: OpenGL Graphics Pipeline 14

Fig 8: The Plasma Edge [3] 16

Fig 9: Schematic diagram of EFFIS components [1] 18

Fig 10: eSimMon Dashboard [1] 19

Fig 11: Cross-sectional view of Plasma torus 21

Fig 12: 2D Time Visualization 21

Fig 13: 2D Time Visualization workflow 22

Fig 14: 3D Plasma Torus 23

Fig 15: 3D Torus visualization 24

Fig 16: 3D Torus Visualization Workflow 24

Fig 17: HDF5 Reader in Workflow 26

Fig 18:- Challenges faced by HDF adopters 28

Fig 19: HDF5 group 30

Fig 20: GUI Interface of HDFView 33

Fig 21: Mesh HDF5 File View in HDFView 34

Fig 22: Intensity HDF5 File View in HDFView 35

Fig 23: HDF5 reader flowchart 37

Fig 24: VTK module in 2D time visualization workflow..40

Fig 25: VTK module in 3D torus visualization workflow...40

Fig 26: VTK Object Models 42

Fig 27: Visualization model – Process objects..43

Fig 28: VTK 2D Time Visualization module Flowchart..45

Fig 29: VTK 3D Torus Visualization Module Flowchart...46

Fig 30: Flash module in 2D Time Visualization Workflow 48

Fig 31: Flash module in 3D Torus Visualization Workflow 48

Fig 32: Flash 2D Time Visualization Module flowchart...55

Fig 33: Flash 3D Torus visualization module flowchart 56

Fig 34: 2D preprocessed images loaded 57

Fig 35: Rotation around Y-Axis...58

Fig 36: Controlling Space between the images..58

Fig 37: Rotating Images across Z-Axis 59

Fig 38: Zooming Operation...59

Fig 39: Image at 0 degree tilt and 0 degree rotation angle loaded at startup 60

Fig 40: Images loaded according to tilt angle 61

Fig 41: Images loaded according to rotation angle 62

1. Introduction

Process of converting raw data into a form which is viewable and understandable by human being is called visualization and the field of science which represents such complex and massive scientific data through visual images is called Scientific Visualization. Such scientific data, referred to as ‘dataset’, can vary from an output of a scientific simulation or a laboratory experiment or it can be collected by sensors in fields of geology, medicine, physics etc. These datasets can be of various sizes and structures. Visualization of these datasets creates images providing a more intuitive interpretation of the dataset for the process of hypothesis building and modeling. These hypothesis building and modeling can range for observing desired features or region of interest, certain anomalies and change in patterns.

Scientific visualization can be applied to various fields such as medicine in the form of Computer Tomography (CT) and Magnetic Resonance Imaging (MRI), various physical processes such as simulating ballistic trajectories, fluid flow (plasma) and structural mechanics, and predicting weather with help of weather maps. More recently, visualization has found its application in finance, marketing and business to generate graphs for investment portfolios, risk versus return and many other areas. All these applications have one thing in common: to extract important hidden information from vast amount of numerical data which represents various parameters depending on experiment such as pressure, electrical field, potential difference, wind speed, stock price etc.

Visualization of data can be divided into four major steps as shown in Fig 1:

- Acquiring data from different sources
- Applying various methods to transform data, for example to convert 2D dataset to 3D dataset
- Mapping transformed data to appropriate form
- Displaying or rendering the result

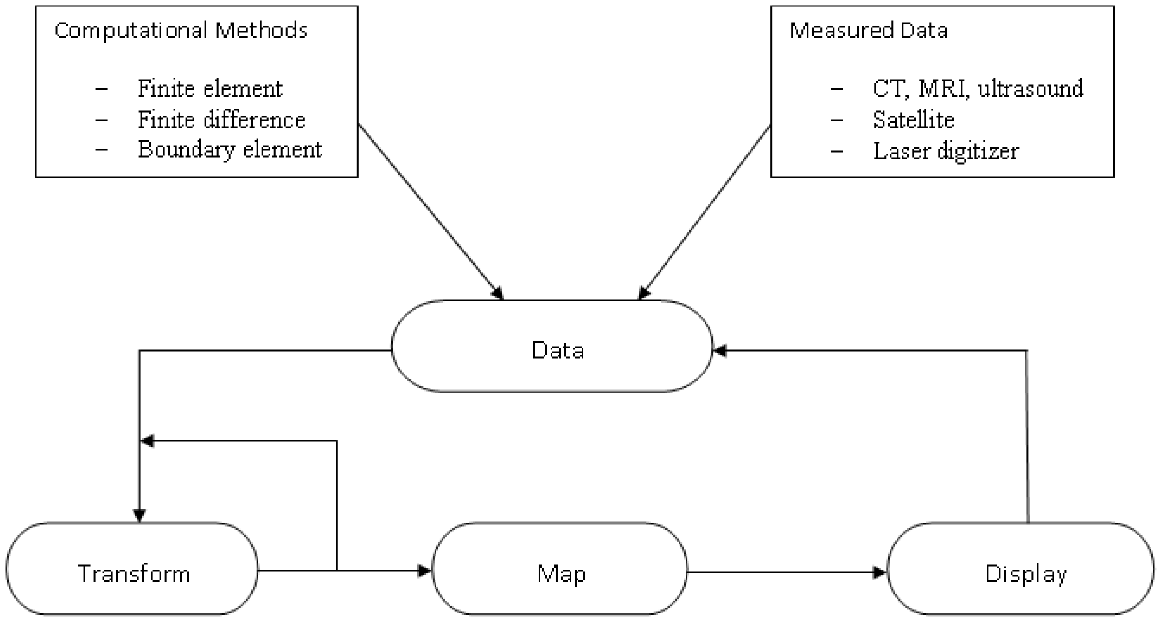


Fig 1: The visualization process.
Data from various sources is repeatedly transformed to extract, derive, and enhance information. The resulting data is mapped to a graphical system for display.
(Image Source: - Chapter 1, Page 6 – The Visualization toolkit – 2nd edition)

1.1 Motivation for thesis

The Center for Plasma Edge Simulation is developing a novel integrated predictive plasma edge simulation framework applicable in existing magnetic fusion facilities and next generation burning plasma experiments. This will help scientists in understanding new models for the plasma edge in kinetic regime with complex geometry [1]. As a part of Rutgers initiative, a visualization module for plasma edge simulation was developed using commercial visualization software, Advanced Visual Systems (AVS). This thesis focuses on migration of the AVS module to an open source visualization library, VTK, for generating plasma edge images, thus saving licensing costs.

After visualization is complete, it becomes difficult to move huge amounts of data from server to local machines for analysis. To avoid that CPES developed a web based dashboard – eSimMon, which provides scientists the tools for performing analysis on the results. In order to integrate VTK module in the workflow and to enable scientists to perform analysis on the graphical images produced as a result of visualization - two different Flash modules were developed to give a sense of 2D in time (Fig 2) and 3D in space (Fig 3). In Flash module depicting 3D in time, 2D slices of plasma torus were loaded in Flash and various transformation were applied to these images interactively. In second flash module which displays 3D in space, 2D images of plasma torus were loaded in flash and based on mouse movements current image displayed was changed. More details about CPES workflow is presented in chapter 3.

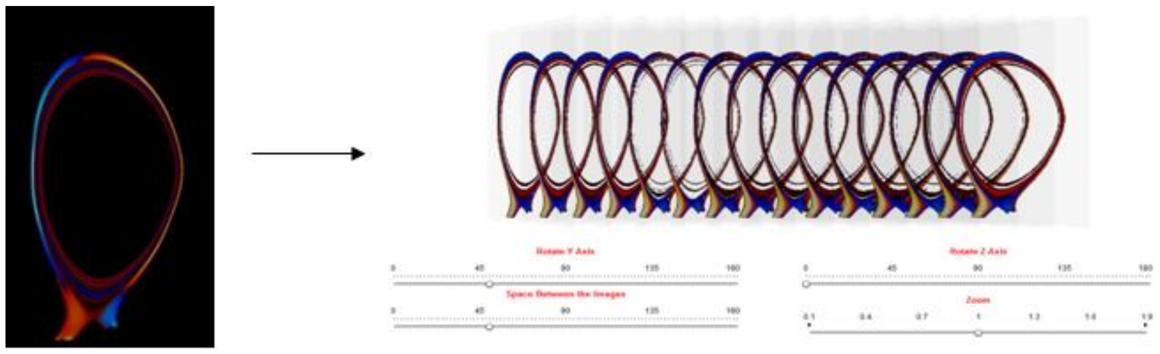


Fig 2: 2D Time Visualization

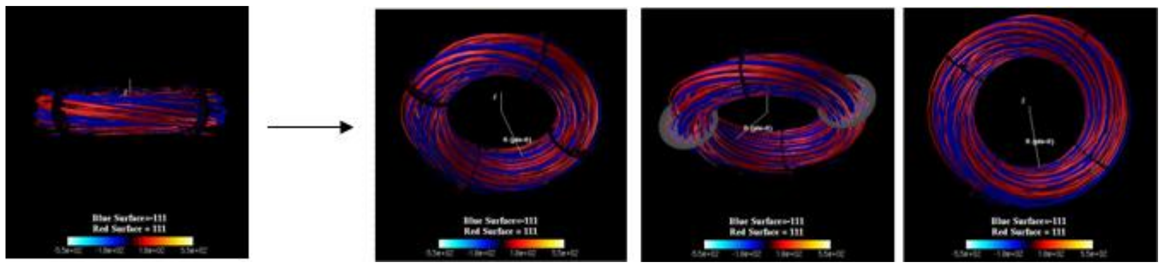
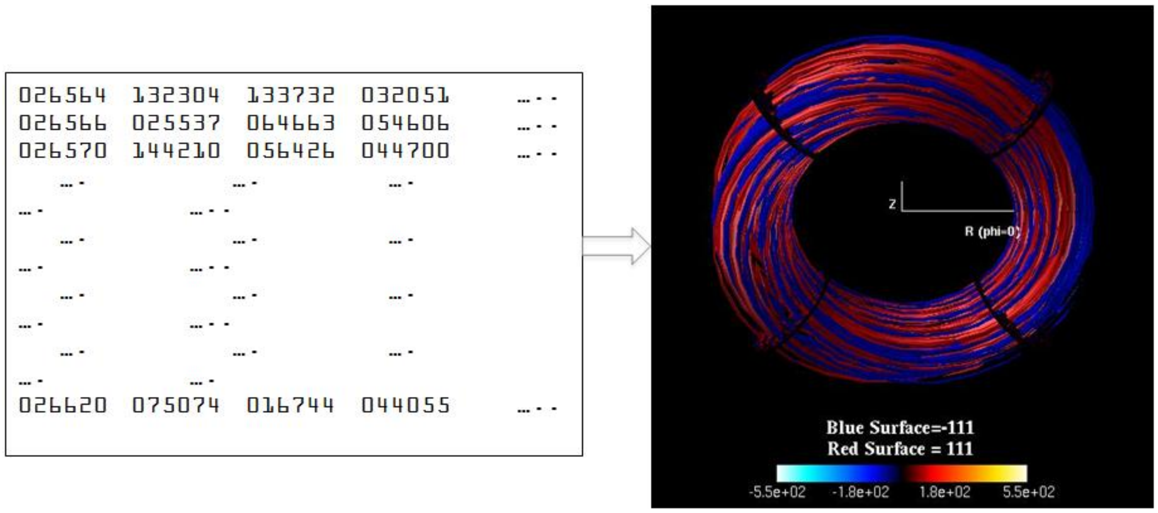


Fig 3: 3D Torus Visualization

The organization of the thesis is as follows: Chapter 2 provides an insight into Scientific Visualization, describes dataset file formats, some visualization software languages. Chapter 3 gives background information about Plasma Edge research going on at CPES. Chapter 4 provides detailed information about HDF5 file format and its applications in Visualization world. Chapter 5 gives information of VTK concepts used in this thesis. In Chapter 6, we discuss about Adobe Flash and its usefulness as visualization tool to group the results produced by previous steps and present them in an interactive manner. Chapter 7 details the results obtained by the visualization process and the interactive module developed using Flash. Also, Appendix A elaborates the steps required to install VTK using a Windows system and Visual Studio Environment. Appendix B provides information to setup HDF5 environment using Windows and Visual Studio.

2. Scientific Visualization

Visualization is the process of taking in raw data and presenting it in a form which allows understanding relationships easily otherwise not readily evident from raw data [4]. Figure 4, represents numerical data collected from experiments or simulation. It is very difficult to ascertain relationship or to interpret the meaning of these numbers, but after visualizing these numbers in the form of an image, we get a much better idea of what the data represents. In this case, the data represents electric potential of plasma torus inside the cyclotron.



```
026564  132304  133732  032051     ....
026566  025537  064663  054606     ....
026570  144210  056426  044700     ....
   ...      ...      ...
...        ....
   ...      ...      ...
...        ....
   ...      ...      ...
...        ....
   ...      ...      ...
...        ...
026620  075074  016744  044055     ....
```

Fig 4: Visualization of numerical dataset to an image.

As stated in [5], “Scientific visualization is an interdisciplinary branch of science, concerned with visualization of three dimensional phenomena (architectural, meteorological, medical, fluid flow etc), where emphasis is on realistic rendering of volumes, surfaces, illumination sources and so forth, perhaps with a dynamic component of time”. Scientific Visualization uses data driven computer graphics to aid in understanding of scientific data. Does it mean scientific visualization is same as computer graphics? No, Computer graphics is just the process to create images using a computer. This can include 2D paint-and-draw techniques as well as more sophisticated 3D drawing (or rendering) techniques. Visualization of the other hand is the process of exploring, transforming and viewing data as images to gain understanding and insight into the data [6]. Visualization can be also differentiated from computer graphics and image processing on the basis of dimensionality, data transformation and interactiveness.

- Dimensionality - Computer graphics provides us with variety of tools to view data of two dimensions or less, but visualization serves best when applied to data of higher dimension
- Data Transformation – Visualization works on the principle of data filtering and feedback, which leads to data transformation. That is, information gets repeatedly created and modified to enhance meaning of the data
- Interactiveness – Through visualization a set of tools can be provided to make output interactive with the user. It aids in better understanding of the data

Thus, visualization uses graphics as a tool to represent data during rendering phase.

Scientific visualization can be considered analytically as a transfer function between data and graphical images. It involves a barrage of procedures, each influencing the final outcome and ability to convey meaningful information. These procedures can be roughly categorized using Figure 5.

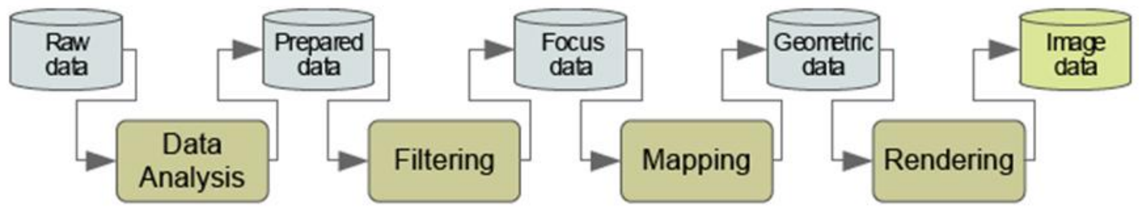


Fig 5: Visualization pipeline
(Image Source: **-** IS&T Scientific Visualization Tutorial – Summer 2010
www.bu.edu/tech/files/2010/06/VizTutSummer2010-DataWrangling.ppt)

- Data Analysis – It includes loading data from a file into memory and inspecting, cleaning, transforming, and modeling it with the goal of highlighting useful information
- Filtering – It can lead to further cleaning and processing of data to yield meaningful results. Examples can be removing noise, replacing missing values, clamping values in a certain range or producing new numeric forms leading to greater insights
- Mapping – It involves transforming filtered data and then mapping to a form appropriate for presentation to the user. To achieve that, it is advisable to provide information about the simulation itself e.g., coordinate system, scale information, resolution of computation in addition to just providing numerical data
- Rendering – Finally the mapped data gets rendered or displayed on the screen

Sometimes, the visualization process might also include an additional step of feedback. Feedback helps scientists to question the accuracy and the validity of the information presented to them. Feedback helps in making changes to the process, thus making the process more accurate and generating better results for the intended audience.

2.1 Data File Formats

Representing large amounts of data obtained from different simulations and experiment environments is a tedious task. In order to have a consistent data representation scheme for a variety of dataset types and to provide a simple method for exchanging data between groups, various file formats have been developed.

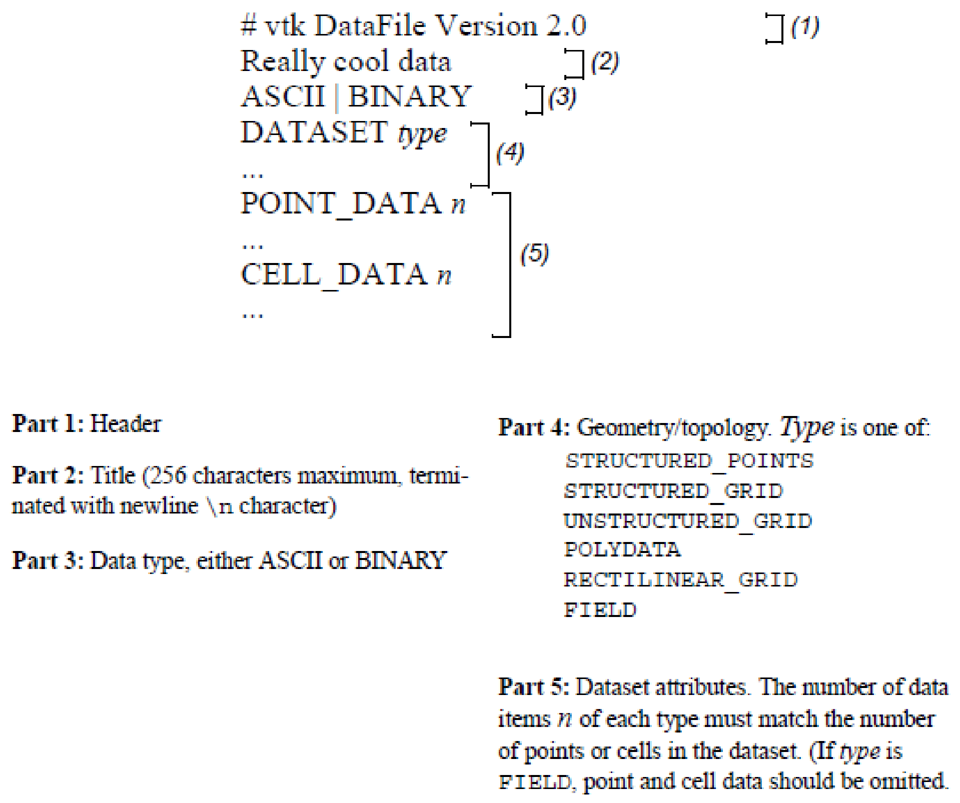
Common file formats to represent data are .vtk, .obj and .hdf5. These are described below in more detail

2.1.1 .vtk file format

VTK file format consists of five basic parts [7]:-

1. First part is file version and identifier
2. Second part is the header
3. Next part is the file format, either ASCII or binary
4. Fourth part is the dataset structure
5. Final part describes the dataset attributes (POINT_DATA or CELL_DATA)

Figure 6 provides an overview of the various components that make up a VTK file.



```
# vtk DataFile Version 2.0          (1)
Really cool data                    (2)
ASCII | BINARY                      (3)
DATASET type                        (4)
...
POINT_DATA n                        (5)
...
CELL_DATA n
...
```

Part 1: Header

Part 2: Title (256 characters maximum, terminated with newline `\n` character)

Part 3: Data type, either ASCII or BINARY

Part 4: Geometry/topology. *Type* is one of:

```
STRUCTURED_POINTS
STRUCTURED_GRID
UNSTRUCTURED_GRID
POLYDATA
RECTILINEAR_GRID
FIELD
```

Part 5: Dataset attributes. The number of data items *n* of each type must match the number of points or cells in the dataset. (If *type* is `FIELD`, point and cell data should be omitted.

Fig 6: Overview of five parts of VTK data file format [7]

VTK file format example

vtk DataFile Version 2.0

Cube example

ASCII

DATASET POLYDATA

POINTS 8 float

0.0 0.0 0.0

1.0 0.0 0.0

1.0 1.0 0.0

0.0 1.0 0.0

0.0 0.0 1.0

```
1.0 0.0 1.0

1.0 1.0 1.0

0.0 1.0 1.0

POLYGONS 6 30

4 0 1 2 3

4 4 5 6 7

4 0 1 5 4

4 2 3 7 6

4 0 4 7 3

4 1 2 6 5
```

2.1.2 .obj file format

OBJ file format is a text file format, which means OBJ files can be edited in a text editor. OBJ files are read and parsed from top to bottom where first character of each line specifies the type of command. As an OBJ file only allows sequential access, this possesses a major limitation if user wants to randomly access data in a file. In the descriptions of commands that follow, first character represents the command followed by arguments. Anything shown in brackets is optional parameters [8]. Various commands in this file format are:-

- # - a comment line

  This represents a comment line and is always ignored.

- v x y z – Vertex command

This specifies a vertex by its three coordinates. Point to remember- the vertex is implicitly named by the order it is found in the file. For example: the first vertex in file is referenced as ‘1’, the second as ‘2’ and so on.

- vt u v [w] – Vertex texture command

  It specifies the UV (and optionally W) mapping.

- vn x y z – Vertex Normal command

  It specifies a normal vector.

- f v1[/vt1][/vn1] v2[/vt2][/vn2] v3[/vt3][/vn3] ... – Face command

  It specifies a polygon made from the vertices listed. One can use as many vertices as one likes. For referencing a particular vertex one just gives its index in the file, for example ‘f 10 11 12 13’ means that a face is built using the vertices 10 – 13.

OBJ file example

```
# Cube

v 3.716360 2.343387 0.000000

v 4.126565 0.642027 0.000000

v 3.454971 2.169877 0.000000

v 3.929251 0.411689 0.000000

v 3.247406 2.073333 0.000000

v 3.731689 0.186391 0.000000

f 0 1 2 3

f 0 4 7 3

f 0 1 5 4
```

f 1 5 6 2

f 5 6 7 4

f 6 7 3 2

2.1.2 HDF5 file format

HDF5 is an acronym for Hierarchical data format. It can be categorized as a data model, library and a file format for storing and managing data. It can be used to represent unlimited variety of datasets and is designed for flexible and efficient I/O and for high volume and complex data. A more detailed description of this file format is presented in Chapter 4 of this thesis.

2.2 Scientific Visualization tools

A number of tools can be found these days for visualizing information. These can be broadly classified into:-

- Plotting libraries – These libraries were developed to help researchers generate charts, graphs and plots without the need to reinvent the graphics themselves. Here, interaction was achieved using programming which gives it limited interactivity. For ex:- Bluff, PLplot etc
- Turn-key packages – These packages are developed specifically for visualization and contain controls (widgets) for most options users would exercise when visualizing data. These controls are available in form of pull down menus or popup windows. Examples are PV-Wave, GnuPlot etc.
- Dataflow packages – These packages employ dataflow concept of breaking down the tasks into small programs, each of which does one task. Each task is

represented using its own module. These packages offer feature to add custom dataflow network depending on the requirement. Examples are AVS, Vizit etc.

- Graphics and Visualization languages – These languages are used to write programs customized to perform a particular task at hand. When the visualization effort required creating own dataflow network vs. program is less, it's preferable to go for visualization languages and create custom programs. Examples include OpenGL, VTK.

As this thesis focuses on visualization of plasma edges using VTK, we will discuss two visualization languages that are commonly used – OpenGL and VTK.

2.2.1 OpenGL

OpenGL stands for Open Graphics Library. It provides a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. It can be considered as a software interface to graphics hardware. OpenGL standard specifies the graphics processing pipeline for geometric primitives and image data, leaving the implementation of the pipeline for the provider [9]. It has an API that provides access to large number of graphics functions including rendering, texture mapping. For displaying an image on screen which can range from a 2D scene to a complex 3D scene, models are built from small set of geometric primitives – points, lines and polygons and then they are rendered into a form suitable for display by the graphics hardware. Standardization of OpenGL ensures that all applications will produce consistent results on any OpenGL compliant platform regardless of the operating system or the windowing system in use.

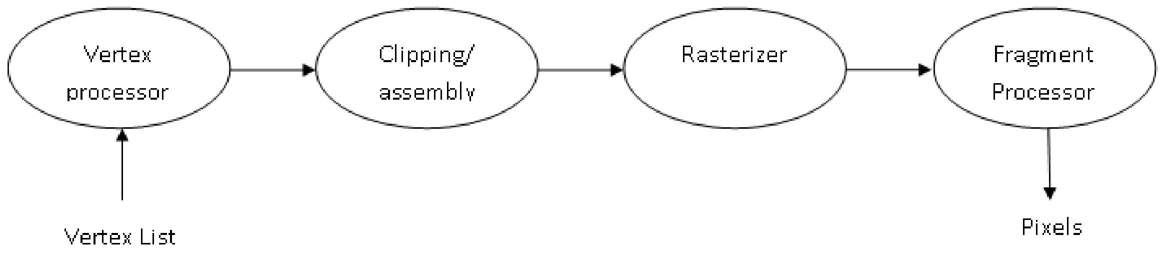


Fig 7: OpenGL Graphics Pipeline

Steps involved in visualizing data using OpenGL can be broadly divided into four steps as shown in figure 7 and explained below:

- Vertex Processor – Role of vertex processor is to assemble vertices into primitives such as lines, curves, triangle, polygons and surfaces. Here, face list is used to index into the vertex list.
- Clipping Assembly – The role of clipping assembly is to remove primitives which are outside camera's view frustum. It also removes surfaces which are facing away from camera. Thus it ensures that any object which can't be seen is not rendered.
- Rasterizer – It converts each primitive into a fragment where set of pixels for that primitive are taken, each having its own RGB color and depth. Vertex colors are then interpolated over the whole fragment.
- Fragment Processor – Finally the fragments are assembled into final frame buffer where algorithms such as Z-buffer will run to remove hidden-surfaces.

2.2.2 VTK

The Visualization toolkit is an open source, freely available software system for 3D computer graphics, image processing and visualization. VTK is specifically developed for visualization and has a higher level of abstraction than OpenGL. Due to this it becomes much easier to create graphics application and visualization using VTK. It has been developed according to object-oriented methodology and consists of C++ class library which makes it closer to real world.

VTK supports a large number of visualization algorithms including scalar, vector, tensor, texture and volumetric methods. It can be also used in advanced modeling techniques such as implicit modeling, polygon reduction, mesh smoothing, cutting, contouring and Delaunay triangulation. Detailed description of VTK can be found in chapter 4 of this thesis.

3. Background

The Department of Energy (DOE) recently funded a prototype Fusion Simulation Project, known as The Center for Plasma Edge Simulation (CPES). CPES has been actively researching advanced predictive modeling capabilities in the area of magnetically confined fusion plasmas. These advanced modeling capabilities will help scientists investigate a range of physics issues deemed crucial in the understanding and efficient design and operation of next-generation tokamak fusion reactor devices such as the International Thermonuclear Experimental Reactor (ITER). CPES was specifically tasked with developing a new integrated and predictive plasma edge simulation package due to strong correlation of confinement properties of tokamak plasmas with plasma edge conditions.

3.1 Importance of Plasma Edge Research

“The plasma edge includes the region from the top of the pedestal to the scrape-off layer and divertor region bounded by a material wall” [3], see Figure 8.

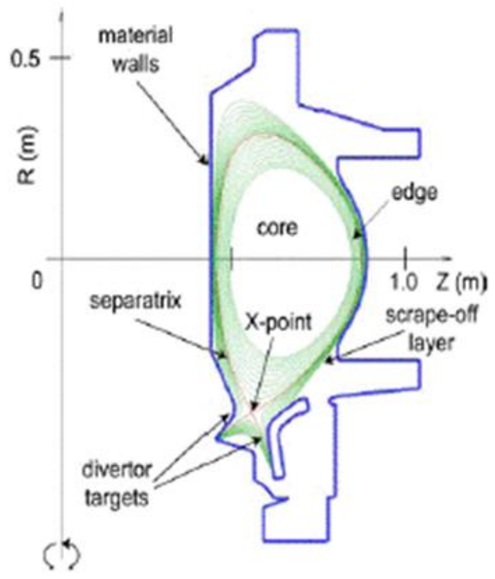


Fig 8: The Plasma Edge [3]

As plasma edge has strong correlation to confinement properties of tokamak plasmas, scientists need to study multitude of non-equilibrium physical processes on different temporal scales present in the edge region. Thus, CPES focuses its physics research to study the interplay of neoclassical transport, micro-turbulence and magnetohydrodynamic (MHD) instabilities in the plasma edge. CPES uses kinetic approach such as a particle-in-cell (PIC) simulation to model neoclassical transport and micro-turbulence, while MHD modes are more efficiently studied with a fluid code [1]. Hence, CPES developed a new integrated code framework EFFIS (An End to end Framework for Fusion Integration Simulation) to couple newly developed edge kinetic simulation capabilities with existing state-of-the-art MHD models. This framework enabled physicists to understand the dynamic interaction of kinetic effects that cause a buildup of the edge pedestal in plasma density and temperature profiles and large bootstrap currents with so-called Edge Localized Modes (ELMs).

3.2 EFFIS Workflow

CPES developed EFFIS framework to automate the integrated simulation process of kinetic edge pedestal buildup, stability boundary check and edge localized mode crash for multiple ELM cycles. EFFIS has also been used recently to monitor long running micro-turbulence simulation in the plasma edge. To perform the above tasks EFFIS uses highly advanced inter-operating computer science tools such as fast adaptable I/O, workflow management, fast presentation of image data on a web-based dashboard, the collection and management of provenance metadata, and wide-area data transfer as seen in Fig 9.

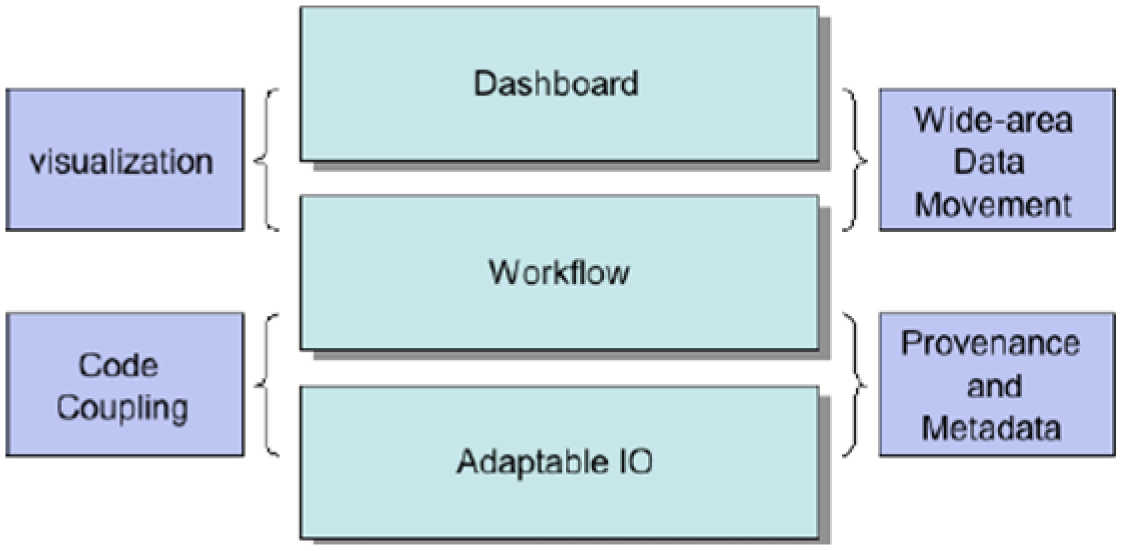


Fig 9: Schematic diagram of EFFIS components [1]

One of the prime requirements of CPES research environments was the support for variety of data transport mechanisms and data formats. Thus, ADaptable I/O System (ADIOS) was developed. Second most important piece of this framework was development of Kepler workflow system [1] used to orchestrate many parallel tasks on multiple computer platforms. The final and one of the most important component of this framework is the eSimMon dashboard, using which users can monitor and analyze long running simulations of the web.

3.3 eSimMon Dashboard

The eSimMon Dashboard is a front-end tool for simulation monitoring. It is used by physicists to analyze, manage, visualize and share data produced by the simulations. With the development of this component, researchers has been able to focus entirely on their research instead of worrying about underlying IT details, such as file locations and formats. Thus, eSimMon Dashboard provides scientists with ease of use, interactivity and responsiveness.

eSimMon dashboard comes under the realm of Rich Internet Applications (RIAs), where dynamic web pages are created on the fly with local interaction and asynchronous communication with the server. So among a plethora of technology choices available for developing dashboard, Adobe Flash was chosen for performing client side operations, whereas the server side programming is done using combination of PHP and MySQL queries.

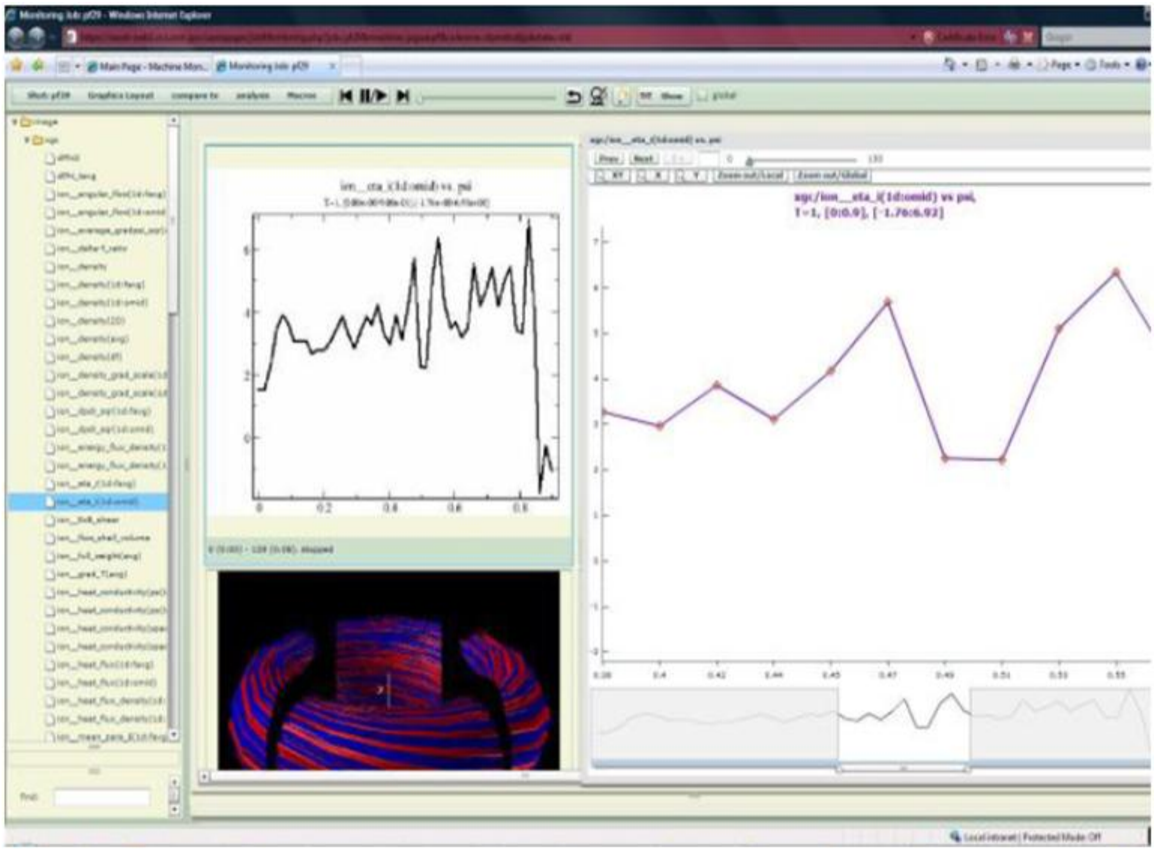

Monitoring Job: pf29 - Windows Internet Explorer
Main Page - Machine Mon...
Monitoring Job: pf29
Graphics Layout
compare to
analysis
Prev
Next
Zoom out/Local
Zoom out/Global
Local intranet | Protected Mode: Off

Fig 10: eSimMon Dashboard [1]

With the help of Adobe Flash, scientists can play videos in dashboard, allowing them to scrutinize data at each time step. In dashboard scientists can also perform vector graphics with zooming and panning capabilities which are is natively supported by Adobe Flash.

On the server side, the eSimMon dashboard uses PHP and MySQL database for linking user requests on the interface to raw data files, thus enabling dashboard to hide IT details from the user and raise the focus from files to scientific variables.

3.4 Vizlab Contribution

eSimMon dashboard uses Adobe Flash for displaying simulation results to its users in the form of images and videos and allows them to perform basic vector graphics operations such as panning and zoom. It provides scientists to perform 2D visualization but lack in the capability to perform 3D visualization. Thus, our goal was to introduce 3D visualization capabilities in the dashboard without actual real 3D, for viewing plasma edge simulation results, thus aiding scientists in better understanding of their simulation. In order to achieve this, two different workflows were developed

- 2D Time Visualization
- 3D Torus Visualization

3.4.1. 2D Time Visualization

In this module, 2D slices of cross-sectional view of plasma torus (Fig 11) were generated using VTK, and these generated images were then loaded in Flash for performing virtual 3D vector graphics operations (Fig 12) in the dashboard.

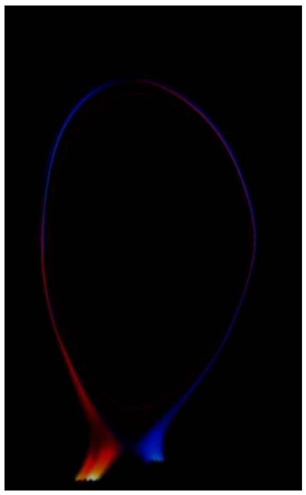


Fig 11: Cross-sectional view of Plasma torus

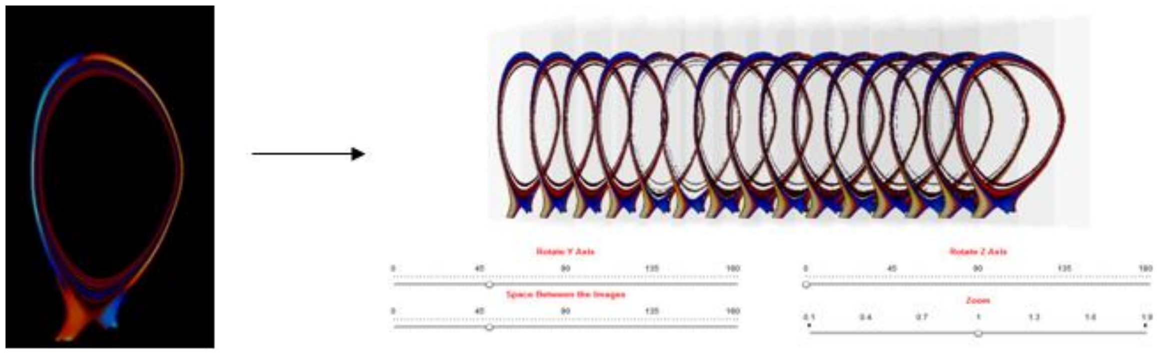


Fig 12: 2D Time Visualization
(Virtual 3D graphics operations on Plasma Edge)

The workflow for performing 2D time visualization is shown in Figure 13.

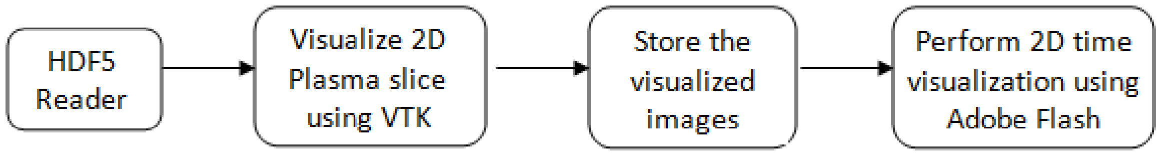

HDF5 Reader
Visualize 2D Plasma slice using VTK
Store the visualized images
Perform 2D time visualization using Adobe Flash

Fig 13: 2D Time Visualization workflow

The 2D time visualization was developed using three technologies – HDF5, VTK and Adobe Flash as explained below

- HDF5 reader – Plasma dataset was provided to us by the CPES group in the form of HDF5 files – Mesh Files and Intensity Files. In order to read data, a reader was created using HDF5 library and C++. Mesh File provided information about the geometry and topology of the plasma edge region and Intensity Files gave information about the intensities associated with each vertex of the plane. Generally, there used to one mesh file provided and for every time-step there was a corresponding intensity file associated with it. We were provided information about 1000 time-steps and visualizing that gave users an idea how plasma edge simulation progresses in time. More information about the HDF5 reader is presented in Chapter 4 – HDF5.
- VTK visualization module – Once the dataset is loaded from HDF5 files, it was visualized into images using VTK library and C++. With the help of VTK library object, 2D coordinates were connected using triangle mesh, were assigned intensity values, mapped to a color range and saved as 2D images. Chapter 5 – VTK provides more information about the VTK workflow.
- Generated Images – Once the images were generated, their information was stored in XML file, which served as an input for Flash module. Image location,

Image name, starting image and end image information was stored in the XML file. Use of XML file provided us with two major advantages. First, I can create custom tags using which I can store the above mentioned information. Secondly, new images can be loaded in Flash by changing their information in XML file, without recompiling Flash code.

- Flash Module – In this module, all the 2D plasma images listed in XML file were loaded in Flash, processed and displayed to the user. With help of controls provided to them using sliders, they can perform 3D vector operations on the images such as Rotation around Y and Z axis, zooming and varying the space between images. More information is provided in Chapter 6 – Flash.

3.4.2. 3D Torus Visualization

In this module, 2D images representing 3D plasma torus (Fig 14) were generated using VTK, and the generated images were loaded in Flash for performing virtual 3D visualization (Fig 15) in the dashboard.

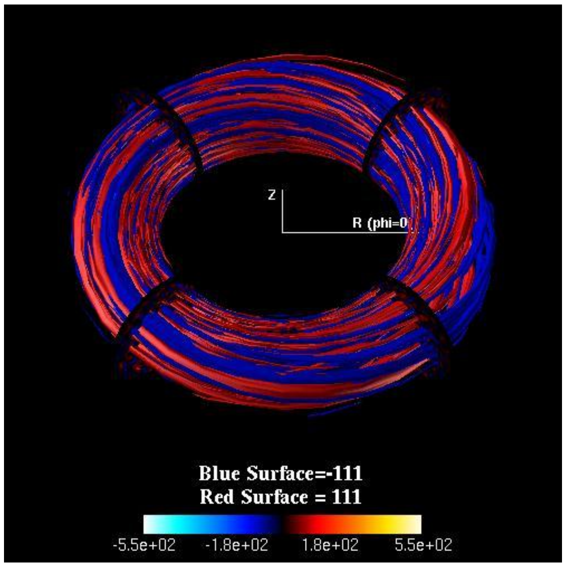


Fig 14: 3D Plasma Torus

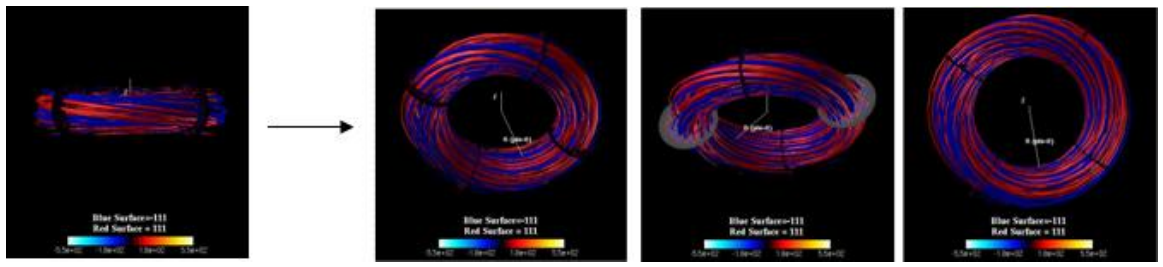


Fig 15: 3D Torus visualization

The workflow for performing 3D Torus Visualization is shown below in Figure 16:-

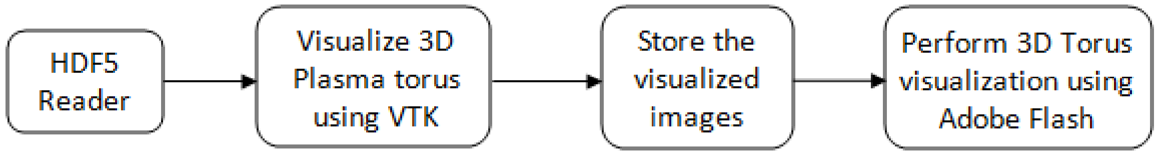

HDF5 Reader
Visualize 3D Plasma torus using VTK
Store the visualized images
Perform 3D Torus visualization using Adobe Flash

Fig 16: 3D Torus Visualization Workflow

The 3D torus visualization was developed using three technologies – HDF5, VTK and Adobe Flash as explained below

- HDF5 reader – Plasma dataset was provided to us by the CPES group in the form of HDF5 files – Mesh Files and Intensity Files. In order to read data, a reader was created using HDF5 library and C++. Mesh File provided information about the geometry and topology of the plasma edge region and Intensity Files gave information about the intensities associated with each vertex of the plane. Generally, there used to one mesh file provided and for every time-step there was a corresponding intensity file associated with it. In dataset provided to us, each intensity files contained intensity information about thirty-three 2D planes. More information about HDF5 reader is presented in Chapter 4 – HDF5.
- VTK visualization module – Once the dataset is loaded from HDF5 files, it was visualized into images using VTK library and C++. For this module, first 2D coordinates were converted to 3D coordinates and connected to form a torroidal

mesh. Then, with the help of VTK library object, this torroidal mesh was assigned intensity values, mapped to a color range and visualized. After visualization is completed, camera angles were varied in rotational (horizontal) and tilt (vertical) axis to generate plasma torus images at various orientations and were saved. Chapter 5 – VTK provides more information about the VTK workflow.

- Generated Images – Once the images were generated, their information was stored in XML file, which served as an input for Flash module. Image location, Image name, starting image, minimum and maximum rotation and tilt angles, interval between rotation and tilt angles was stored in the XML file. Use of XML file provided us with two major advantages. First, custom tags custom tags can be created using which I can store the above mentioned information. Secondly, new images can be loaded in Flash by changing their information in XML file, without recompiling Flash code.
- Flash Module – In this module, all the 3D plasma torus images listed in XML file were loaded in Flash and displayed to the user. With help of mouse events, direction of cursor movement was tracked and accordingly, rotation and tilt angles were changed and plasma torus image pertaining to that rotation and tilt angle was displayed. More information is provided in Chapter 6 – Flash.

4. HDF5

This chapter discusses about HDF5 technology and the steps involved for building a HDF5 reader using HDF5 library and C++.

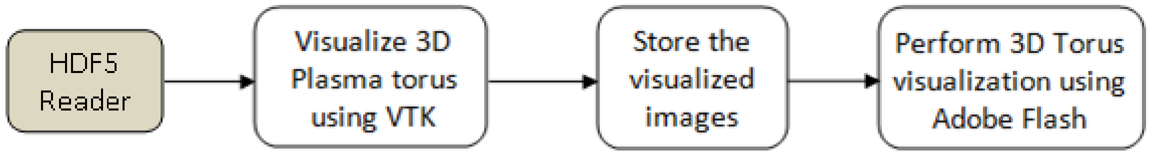

HDF5 Reader
Visualize 3D Plasma torus using VTK
Store the visualized images
Perform 3D Torus visualization using Adobe Flash

Fig 17: HDF5 Reader in Workflow

4.1 Need for a standard file format

One of the major challenges scientists face is how to represent data from their experiments for visualizing it. In the past, scientists used non-standardized data formats for external representation of data. This posed a major challenge as it greatly limited the possibility to interchange data between packages and hence, between research groups and projects. Groups who wish to work together have to overcome the problem of differing file formats first to be able to work on the same data. This becomes a tedious and time consuming task and sometimes it becomes nearly impossible if the file format is not documented, which is the case with most proprietary file formats.

Secondly, data is represented both internally and externally to an application which puts extra demands on representing data. Data resides internal to an application when that particular application is executed, which makes its important to achieve optimal computational performance. Whereas external representation of data should allow easy interchangeability and be as self-explanatory as possible and at the same time allow compact storage and fast read-write access.

To overcome lack of a standard data representation, National Center for Supercomputing Applications came up with a standardized file format and library to represent data, called as Hierarchical Data Format (HDF5).

"HDF5 is a general purpose library and file format for storing scientific data. [10]"

HDF5 system provided scientists with the following features which makes it one of the widely used formats:-

- Versatile data model – HDF5 is capable of representing complex data objects and a wide variety of metadata.
- Portable file format – It provides users with a file format that pose no limit on number and size of data objects in the collection
- Software library – It gives access to a software library making it runnable on a range of computational platforms. HDF5 also implements a high level API with C, C++ and Java interfaces.
- Tools – various tools and applications are provided which can be used to manage, manipulate, view and analyze the data in a collection.

4.2 Data challenges solved using HDF5 technology

Most important advantage of using HDF technology is that it is an open-source software, distributed free of charge. Thus, potential users can take advantage of this technology without any financial investment. Also, projects that adopt HDF are assured they are not relying on a proprietary file format to manage, store their data and in future they won't face any financial challenges of licensing fees and support.

Visualization has huge data requirements due to sheer amount of data which needs to be processed to generate graphical output. These huge dataset needs to represent a wide variety of data representing the results of scientific experiment and simulations. Thus, these huge data demands impose a lot of challenges on the format used to manage data. Some of the key challenges that face HDF5 adopters are shown in the Figure 18.

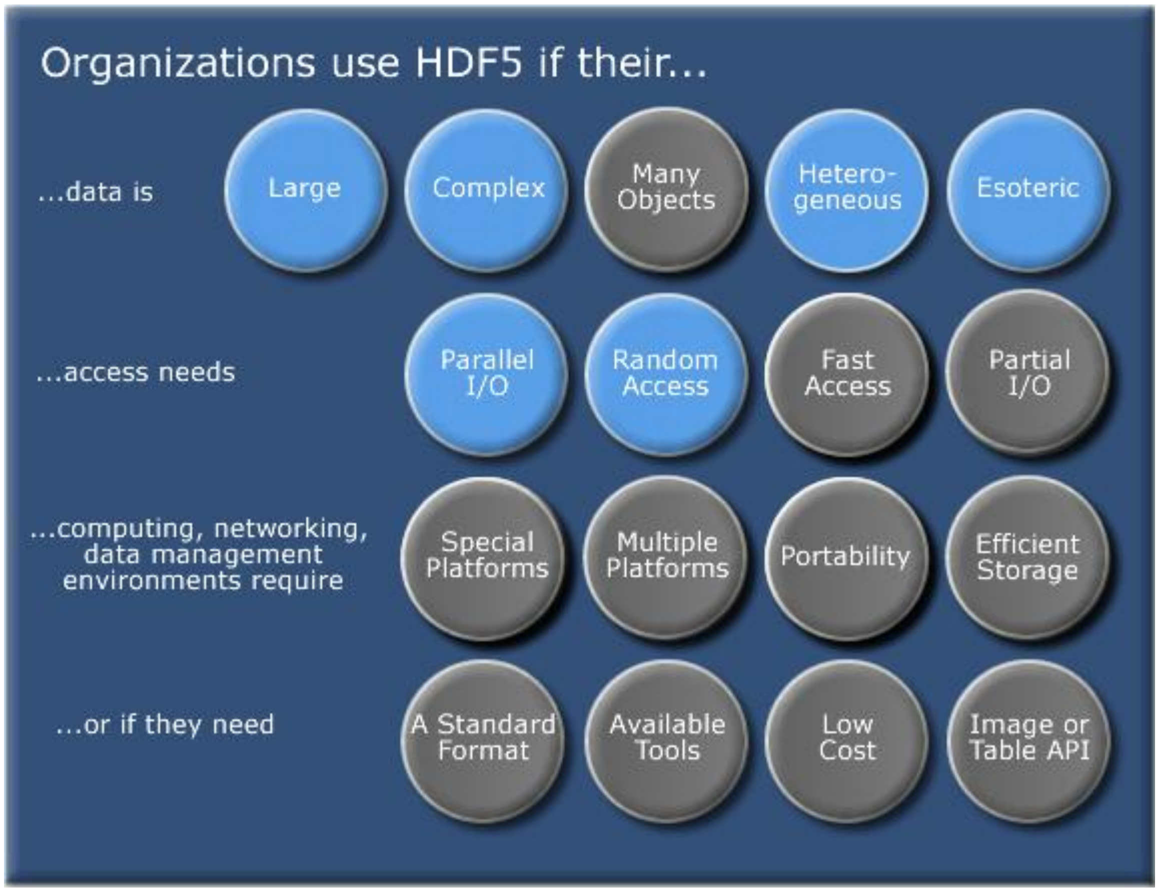


Fig 18:- Challenges faced by HDF adopters
(Image Source: http://www.hdfgroup.org/why_hdf/)

- Data is large – HDF5 format and software are designed specifically to store and access large datasets providing features such as chunking, compression and external object storage which mitigates most problems associated with large datasets. HDF5 also makes it possible for applications to create composite structures, such as tables and indexes, which provide performance-enhancing (fast I/O, compression) and access features (partial I/O).

- Data is complex – Grouping structure in HDF5 enables applications to organize objects in HDF5 to reflect complex relationships among objects. HDF5 provides users with rich collection of HDF5 datatypes, including datatypes that points to data in other objects, and it also includes the ability to define custom datatypes which helps applications build sophisticated structures that match their complex data.
- Heterogeneous Data – HDF technologies allows one to mix and match any kind of data within a file, thus allowing applications to create coherent repositories of highly heterogeneous collections. Users can mix images, tables, and structured data all in the same file. HDF library also provides us with the capability to ass new objects to existing HDF file, even though the file weans originally not created to handle new objects.
- Random and parallel I/O access – Previously mentioned file formats .vtk and .obj only also serialize access to the data. HDF technology on the other hand, provides application the capability to specify subset of elements within the dataset's array that are to be read or written.

  HDF5 reference datatype lets application store information about regions of interest in a dataset, then applying this information for partial I/O operations.

Due to above mentioned features, HDF5 has made inroads in visualizations application developed by various organizations. HDF5 has been used for representing Earth Observing System's HDF-EOS format, DNA sequencing analysis experiment data, test flight data and much more [11].

4.3 HDF5 data organization

HDF5 data model is a directed graph structure with one root node. It consists of three main components – groups, datasets and links. Groups and dataset are nodes in the graph. Links are edges between these nodes. Files, attributes, data types and data spaces form additional components that are not directly related to graph structures but useful in providing additional information [12].

- File – It serves as a container for all hdf5 objects that build up the complete data structure. It holds the root node, a group object type with name "/" and meta data information. This file is stored on disk with an extension ".h5".
- Group – It is an hdf5-named-object. It consists of a group header containing group name, list of attributes, dataset and list of other hdf5- named objects.

  A group is similar to a directory entry in a UNIX or folder in a Windows file system, but group may contain cycles.

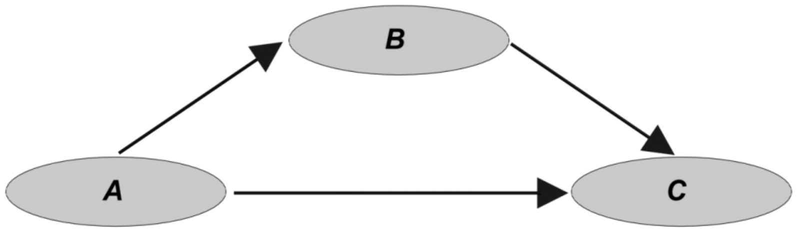


Fig 19: HDF5 group

- Data Set – It consists of a header and a data array. The header contains information required to read the array and metadata. Data set includes the name of data, list of attributes, data type of all the data array elements, data space of array and the layout data is stored in.
- Data Space – One data space object is required in a data set or in an attributes. It provides us with rank of the dataset, which specifies number of dimensions of the data array and maximum number of elements in each dimension. Data spaces can also be used to specify sub selections on the data-sets.
- Attribute – Attributes can be associated to groups, data-set and named data-types. They provide a way to store additional user defined and meta data to an object. Attributes consists of a name, a data-space and a data-type. They are used to store small amount of data of the associated objects. No sharing, compression, chunking or sub-selection can be applied to attributes. An attribute is directly written into the header of the object it is associated with.

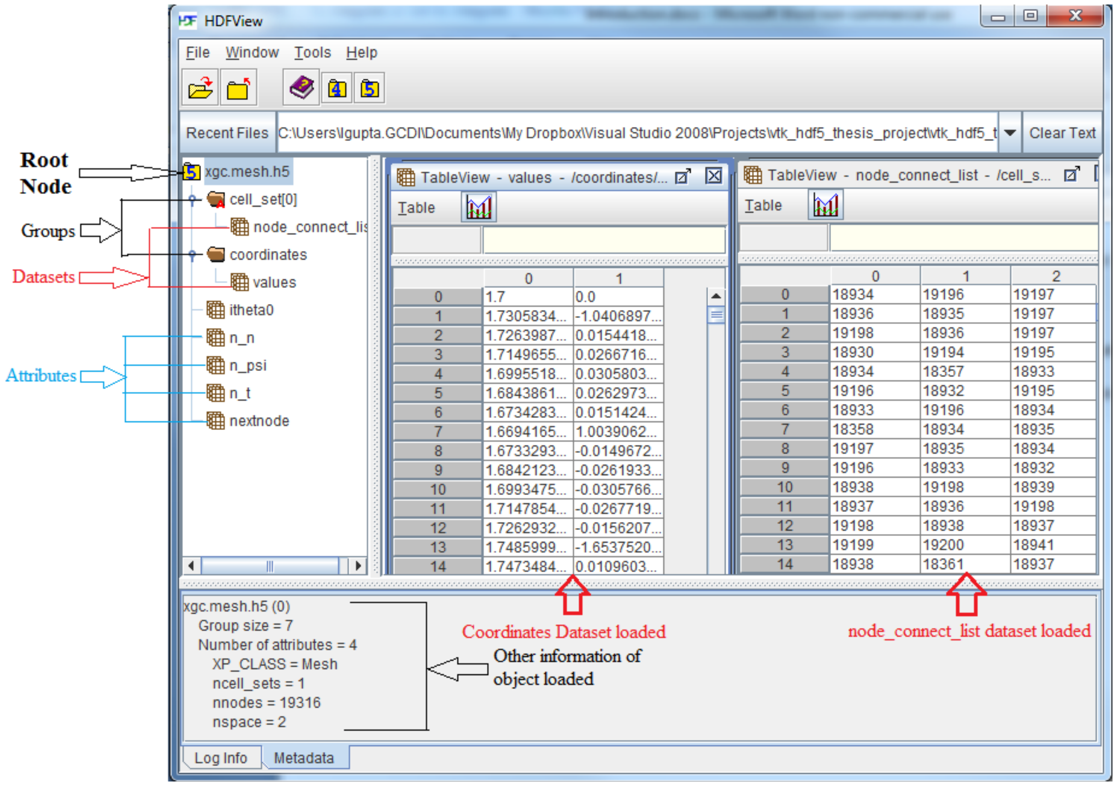
4.4 HDFView – GUI tool to modify HDF5 files

HDFView is a GUI tool for browsing and editing HDF4 and HDF5 files. It allows users to browse any given HDF4 and HDF5 files, starting with a tree view of all top-level objects in an HDF file's hierarchy. Using this tool user can descend through the file hierarchy and navigate among data objects present in file. Most prominent feature of HDFView is that it loads contents of an object only when that object is selected, providing interactive and effective access to HDf4 and HDF5 files. It also contains editing functions using which user can create, delete and modify the value of HDF object and attributes.

There are three most prominent features of HDF5View:-

- It is implemented using Java platform, which make it machine independent. It can run on any machine which has Java virtual machine installed on it. Due to this, GUI components have the same look and feel for all machines.
- Secondly, it uses conventional folders and icons to display groups and datasets in a tree structure. Users can easily expand and collapse folders to navigate the hierarchical structure of an HDF file [13].
- Third, it shows data content as a text or as an image.

Figure 20 gives an illustration of GUI interface of HDFView


HDFView
File Window Tools Help
Recent Files
C:\Users\lgupta.GCDI\Documents\My Dropbox\Visual Studio 2008\Projects\vtk_hdf5_thesis_project\vtk_hdf5_t
Clear Text
Root Node
Groups
Datasets
Attributes
xgc.mesh.h5
cell_set[0]
node_connect_lis
coordinates
values
itheta0
n_n
n_psi
n_t
nextnode
TableView - values - /coordinates/...
TableView - node_connect_list - /cell_s...
Table
Coordinates Dataset loaded
node_connect_list dataset loaded
Other information of object loaded
Log Info
Metadata

	0	1
0	1.7	0.0
1	1.7305834...	-1.0406897...
2	1.7263987...	0.0154418...
3	1.7149655...	0.0266716...
4	1.6995518...	0.0305803...
5	1.6843861...	0.0262973...
6	1.6734283...	0.0151424...
7	1.6694165...	1.0039062...
8	1.6733293...	-0.0149672...
9	1.6842123...	-0.0261933...
10	1.6993475...	-0.0305766...
11	1.7147854...	-0.0267719...
12	1.7262932...	-0.0156207...
13	1.7485999...	-1.6537520...
14	1.7473484...	0.0109603...

	0	1	2
0	18934	19196	19197
1	18936	18935	19197
2	19198	18936	19197
3	18930	19194	19195
4	18934	18357	18933
5	19196	18932	19195
6	18933	19196	18934
7	18358	18934	18935
8	19197	18935	18934
9	19196	18933	18932
10	18938	19198	18939
11	18937	18936	19198
12	19198	18938	18937
13	19199	19200	18941
14	18938	18361	18937

```
xgc.mesh.h5 (0)
  Group size = 7
  Number of attributes = 4
    XP_CLASS = Mesh
    ncell_sets = 1
    nnodes = 19316
    nspace = 2
```

Fig 20: GUI Interface of HDFView
(Loaded dataset – Mesh.h5)

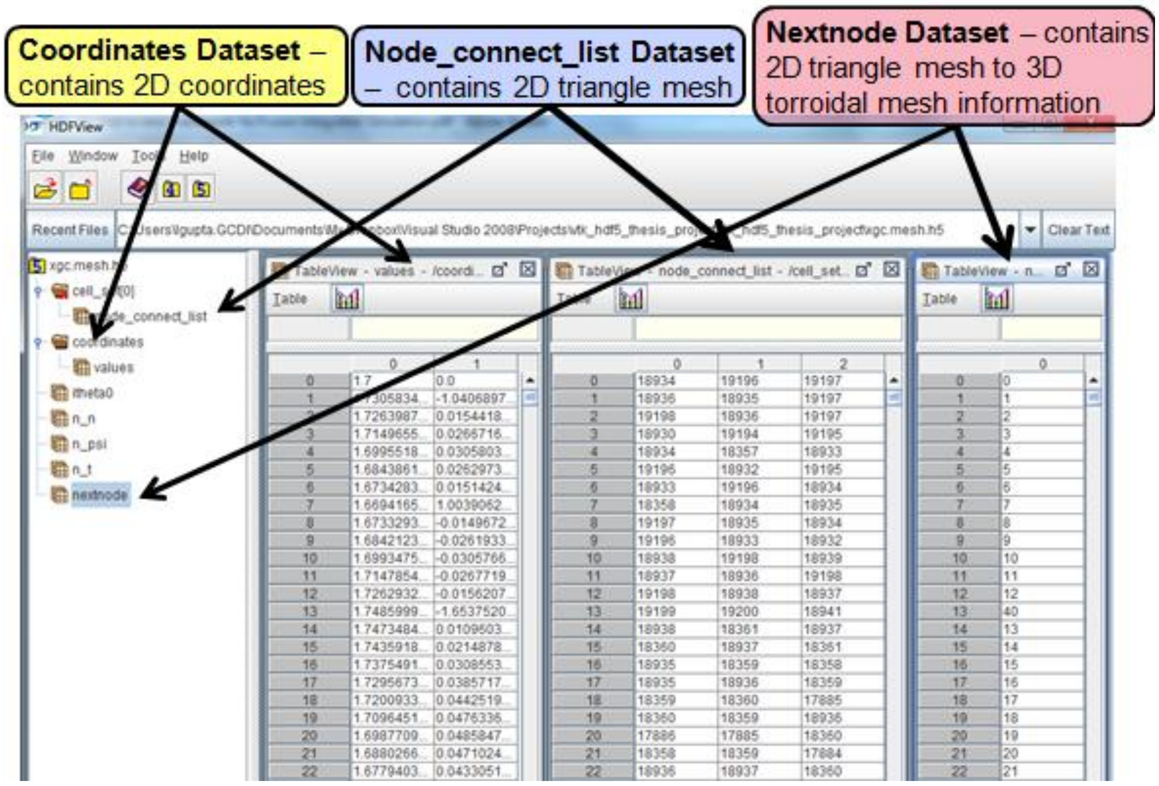
4.5 Data Files

Data for performing both 2D time visualization and 3D torus visualization was provided in form of two HDF5 files

- Mesh File
- Intensity File

4.5.1. Mesh HDF5 File

This file provided information about the geometry and topology of the Plasma Edges. Information contained in mesh file is shown in Figure 21


Coordinates Dataset – contains 2D coordinates
Node_connect_list Dataset – contains 2D triangle mesh
Nextnode Dataset – contains 2D triangle mesh to 3D torroidal mesh information
HDFView
File Window Tools Help
Recent Files
Clear Text
xgc.mesh.h5
cell_set[0]
node_connect_list
coordinates
values
rtheta0
n_n
n_psi
n_t
nextnode
TableView - values - /coordi...
TableView - node_connect_list - /cell_set...
TableView - n...
Table

	0	1
0	1.7	0.0
1	1.7305834...	-1.0406897...
2	1.7263987...	0.0154418...
3	1.7149655...	0.0266716...
4	1.6995518...	0.0305803...
5	1.6843861...	0.0262973...
6	1.6734283...	0.0151424...
7	1.6694165...	1.0039062...
8	1.6733293...	-0.0149672...
9	1.6842123...	-0.0261933...
10	1.6993475...	-0.0305766...
11	1.7147854...	-0.0267719...
12	1.7262932...	-0.0156207...
13	1.7485999...	-1.6537520...
14	1.7473484...	0.0109603...
15	1.7435918...	0.0214878...
16	1.7375491...	0.0308553...
17	1.7295673...	0.0385717...
18	1.7200933...	0.0442519...
19	1.7096451...	0.0476336...
20	1.6987709...	0.0485847...
21	1.6880266...	0.0471024...
22	1.6779403...	0.0433051...

	0	1	2
0	18934	19196	19197
1	18936	18935	19197
2	19198	18936	19197
3	18930	19194	19195
4	18934	18357	18933
5	19196	18932	19195
6	18933	19196	18934
7	18358	18934	18935
8	19197	18935	18934
9	19196	18933	18932
10	18938	19198	18939
11	18937	18936	19198
12	19198	18938	18937
13	19199	19200	18941
14	18938	18361	18937
15	18360	18937	18361
16	18935	18359	18358
17	18935	18936	18359
18	18359	18360	17885
19	18360	18359	18936
20	17886	17885	18360
21	18358	18359	17884
22	18936	18937	18360

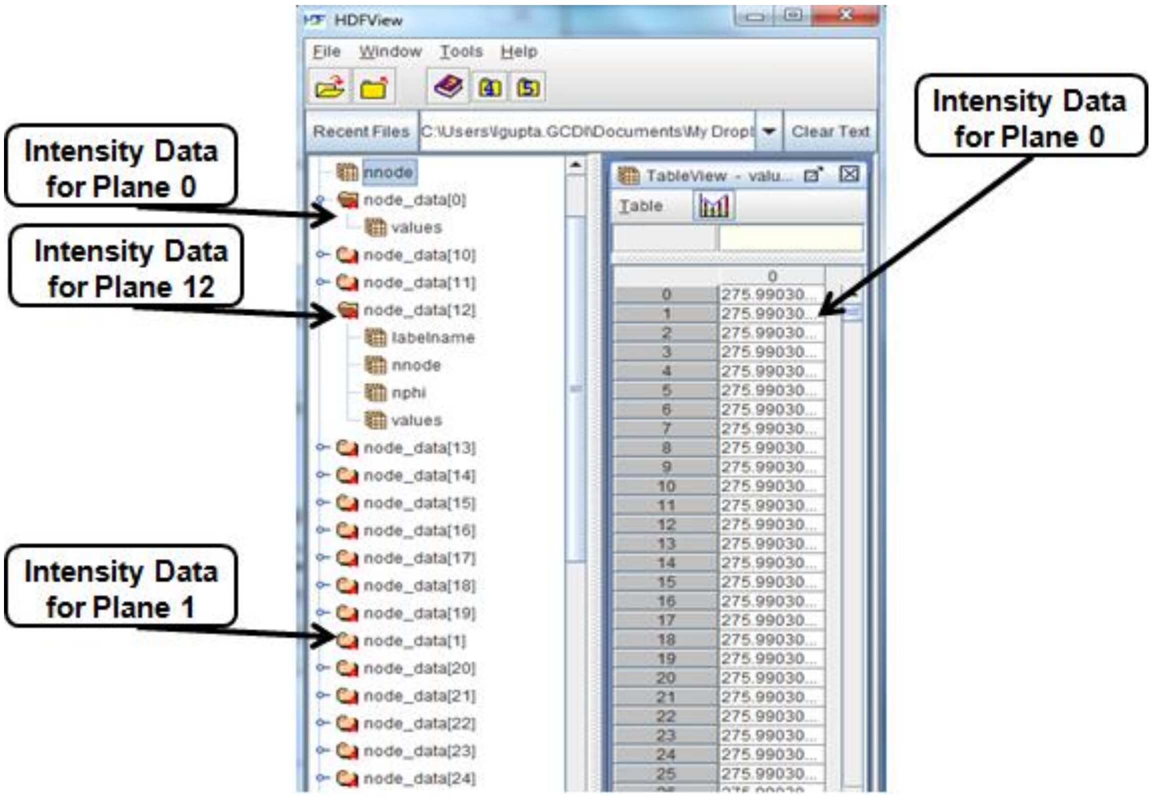
	0
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	40
14	13
15	14
16	15
17	16
18	17
19	18
20	19
21	20
22	21

Fig 21: Mesh HDF5 File View in HDFView

- 2D coordinates – As HDF5 files are organized in groups, Coordinates dataset inside the Mesh file provided information about number of 2D coordinates and their values. In the dataset provide to us, number of 2D coordinates were 19316.
- 2D triangular meshes – “node_connect_list” dataset present in Mesh file provided information on how to form a 2D triangular mesh using the above coordinates. In the dataset, there were 38331 triangular meshes that were formed
- Information to form 3D torroidal Mesh – “node_connect_list” dataset provided information on how to form a 3D torroidal mesh from 2D triangular mesh. This dataset provided information on how to join vertex of one plane to corresponding vertex of another plane.

4.5.2 Intensity HDF5 File

This file provided information about the intensity associated with each vertex of each plane. Generally, there would be many planes information present inside a single file, representing information for a particular time-step. For example: - Figure 22 shows a snapshot of the intensity file which contains intensity values for different 2D planes. In the dataset given to us, it contained intensity information for thirty-three planes in each intensity file belonging to each time-step.



	0
0	275.99030...
1	275.99030...
2	275.99030...
3	275.99030...
4	275.99030...
5	275.99030...
6	275.99030...
7	275.99030...
8	275.99030...
9	275.99030...
10	275.99030...
11	275.99030...
12	275.99030...
13	275.99030...
14	275.99030...
15	275.99030...
16	275.99030...
17	275.99030...
18	275.99030...
19	275.99030...
20	275.99030...
21	275.99030...
22	275.99030...
23	275.99030...
24	275.99030...
25	275.99030...

Fig 22: Intensity HDF5 File View in HDFView

- Number of planes – "nphi" dataset present in the intensity file provides information about the number of 2D planes information present in the file.
- Intensities – "node_data[plane_number]" dataset provided intensity information associated with each vertex of that particular 2D plane.

In the dataset intensity value represent turbulent electrostatic potential.

4.5 HDF5 Reader

For both the workflow – 2D time visualization and 3D torus visualization, process of reading data from HDF5 files is the same, with difference in the information which is read from the file.

4.5.1. Information required for 2D time visualization

Using this workflow, scientists study how the behavior of plasma edges as time progresses. They are interested in viewing 2D slices of particular plane, say plane number 20, from time-steps 30 to 60. In order to generate 2D plasma plane number across time-steps 30 to 60 information read is: -

- Coordinates dataset from the mesh file
- Node_connect_list dataset from the mesh file
- Node_data[19] from intensity files Intensity 30 to Intensity 60. Note_data[19] represents intensity value information for plane 20

When the VTK module (next step in workflow), gets the following information it forms a triangle mesh using coordinates and Node_connect_list dataset info and maps intensity values for each plane using Node_data[19] onto the triangle mesh. The output is set of thirty-one images from time-step 30 to 60.

4.5.2. Information required for 3D time visualization

Using this workflow, scientists study the whole plasma torus at various orientations (different rotation and tilt angles). For forming a plasma torus, all the 2D planes present in a given time-step are combined together to form a torus shape. In order to achieve this information read from HDF5 files is:-

- Coordinates dataset from the mesh file
- Node_connect_list dataset from the mesh file
- Next_node dataset from mesh file
- All Node_data dataset present in intensity file for a particular time-step that interests scientists

Using the next_node dataset, all the 2D plane triangle meshes formed using node_connect_list, are converted to form a toroidal mesh. After this, intensity values are mapped onto the planes that form torroidal mesh using Node_data dataset

4.5.3. Flowchart for HDF5 reader

Flowchart for reading information for HDF5 file is presented in Figure 23.

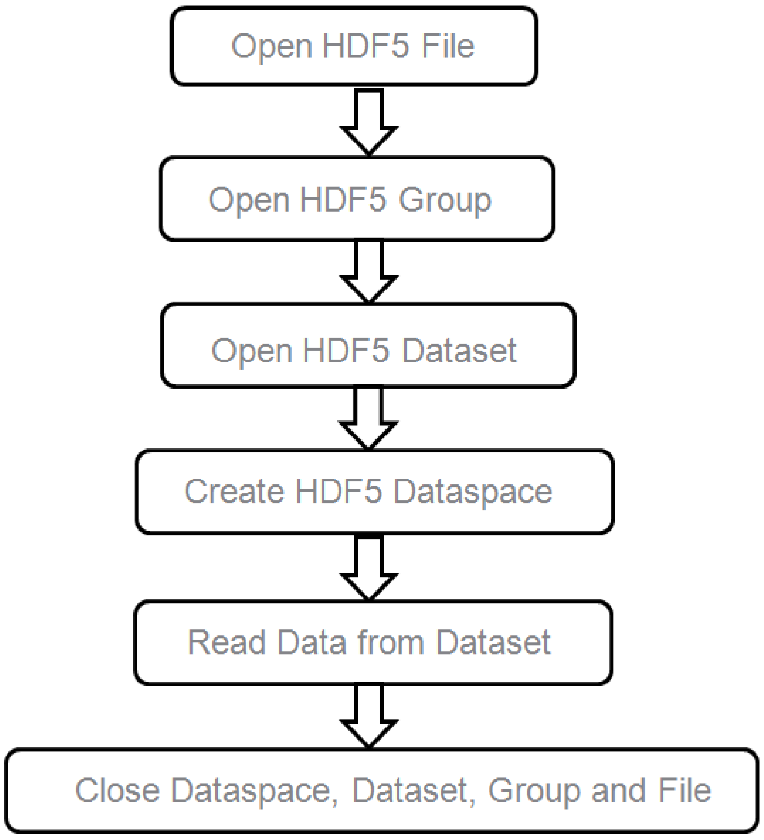


Fig 23: HDF5 reader flowchart

- Open HDF5 file – This is achieved using an H5 file constructor call. Arguments of this function call include H5Filename and the flags used to specify what operation needs to be performed on the file. If the function executes correctly then, a pointer of H5 file is returned, else null is returned back.

  H5File* file = new H5File(const char* H5Filename, unsigned int flag);

  Argument –

  H5Filename – file which needs to be opened

  Flag – what operation needs to be performed on the file. Here, value of flag used is "H5F_ACC_RDONLY".

  Return –

  If the function call is successful then a pointer of type H5File is returned, else null is returned.

- Open Group – This is achieved using a call to "openGroup" function using a file pointer. The result is passed to a group constructor, which returns a group pointer if everything is successful, else a null pointer is returned.

  Group* group = new Group(file->openGroup(groupIntensityName));

- Open HDF5 Dataset – This is done by calling "openDataSet" function using a group pointer. The result is then passed to Dataset constructor. Constructor call returns a pointer of dataset type.

  Dataset* dataset = new DataSet(group->openDataSet(datasetIntensityName));

- Open HDF5 Dataspace – This is achieved by calling “getSpace()” function using dataset pointer. Here an object of type Dataspace is returned which can be further used to get rank and dimension of the dataset.

  ```
  DataSpace dataspace = dataset->getSpace();
  ```

  Rank of a dataspace is obtained by calling “getSimpleExtentNdims()” function.

  ```
  int rank = dataspace.getSimpleExtentNdims();
  ```

4.6 Summary

This chapter provided information about the dataset, information read from the each HDF5 file and the flowchart for HDF5 reader. The output of the HDF5 reader serves as an input for the next step of the workflow – VTK module.

5. Visualization Toolkit (VTK)

This chapter provides information about the next step in the workflow – visualizing 2D plasma images (for 2D time visualization, Fig 24) and 3D plasma torus images (for 3D torus visualization, Fig 25).

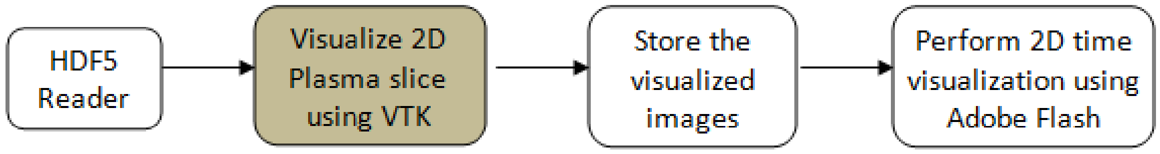

HDF5 Reader
Visualize 2D Plasma slice using VTK
Store the visualized images
Perform 2D time visualization using Adobe Flash

Fig 24: VTK module in 2D time visualization workflow

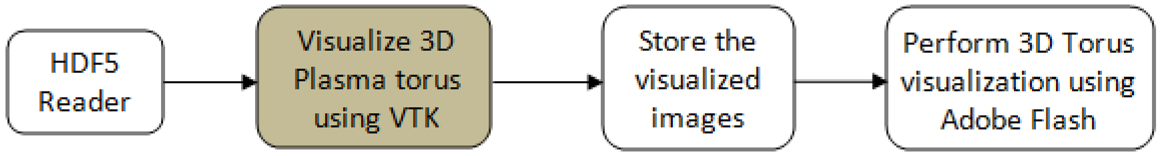

HDF5 Reader
Visualize 3D Plasma torus using VTK
Store the visualized images
Perform 3D Torus visualization using Adobe Flash

Fig 25: VTK module in 3D torus visualization workflow

5.1. Introduction

The Visualization Toolkit (VTK) is an open source, freely available software system for 3D computer graphics, image processing and visualization developed by Kitware Inc [14]. VTK has been developed keeping object-oriented philosophy in mind. It consists of C++ libraries as well as several interpreted interface layers including Tcl/Tk, Java and Python. These layers sit on top of the C++ libraries and simply the creation of application using language of user's choice. Presence of several interpreted interface languages reduces the development time as code written using interpreted languages is written at a higher level than compiled languages which makes the code simpler, more compact which is faster to write and debug.

VTK supports a large number of visualization algorithms ranging from scalar, vector, tensor, texture and volumetric methods to advanced modeling techniques such as: implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation [15]. It also has an extensive information visualization framework containing a suite of 3D interaction widgets, providing parallel processing and capability to integrate with various databases on GUI toolkits such as Qt and Tk. VTK has also integrated various imaging algorithms directly into it, mixing them with 3D graphics algorithms and data. VTK is cross-platform and runs easily on different environments such as Linux, Windows, Mac and UNIX environments. VTK is predominantly used world-wide in development of commercial visualization application, research and development. Most common commercial or advanced visualization application developed using VTK are ParaView, VisTrails, VisIt, 3DSlicer and GPlates etc.

5.2 VTK Object Models

Visualization process employed by VTK can be split into two parts:

- Visualization Model – This model takes in raw information and turns it into a geometric representation.
- Graphics Model – This model takes that geometric representation and converts it into an image

Both these models can be further described by Figure 26

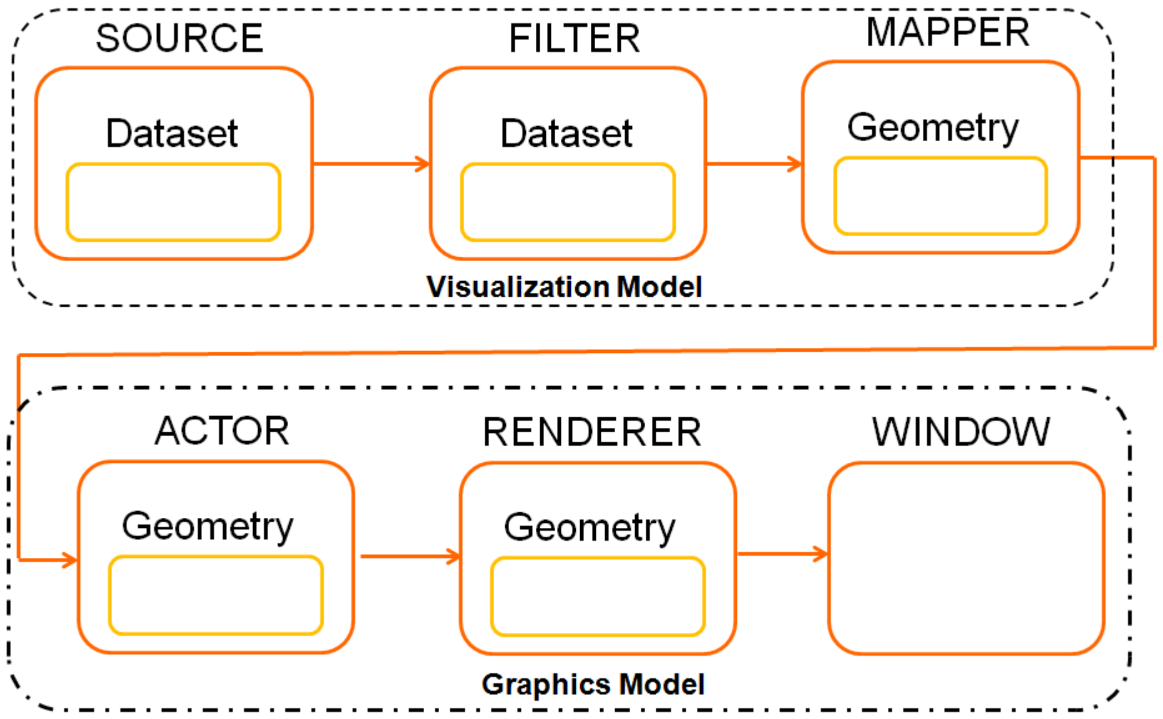


Fig 26: VTK Object Models

5.2.1 Visualization Model

VTK visualization model is based on the data-flow paradigm adopted by many commercial software systems. In data flow paradigm as the name suggests, various modules are connected to form a network or a pipeline for data. These modules perform algorithmic operations on data as it flows through the network. The execution of visualization network is controlled in response to demands for data (demand-driven) or in response to user input (event-driven). Main advantage of this model is that it is flexible and it allows new algorithms to be introduced into the process.

Within the framework of VTK object oriented design, visualization model can be divided into two types of objects as shown in Figure 27.

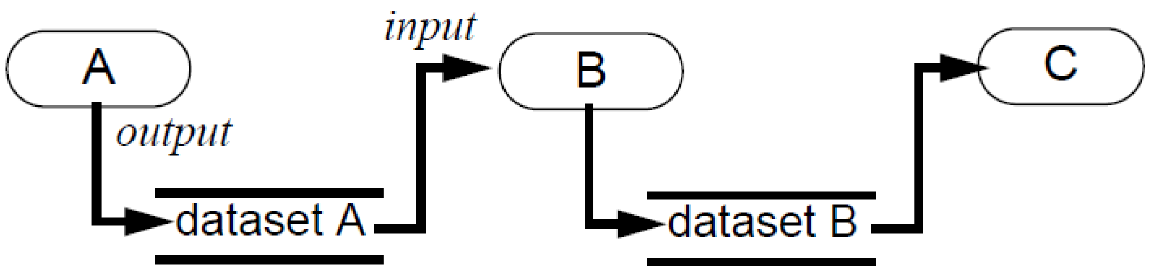


Fig 27: Visualization model – Process objects.
Process objects A, B, C input and/or output one or more data objects. Data objects represent and provide access to data; process objects operate on the data. Objects A, B and C are source, filter and mapper objects respectively. [13]

- Process objects – These are the modules, or algorithmic portions of the visualization network. These are classified into – Sources, Filters and Mappers
- Data Objects – This consists of the data on which process objects act on.

5.2.2 Graphics Model

VTK graphics model is an abstract model for 3D graphics. This abstraction is based on the movie-making industry, with influences from current graphical user interface (GUI) windowing systems. Graphics models are represented by nine basic objects in the model:

- Render Master – It coordinates device-independent methods and creates rendering windows.
- Render Window - manages a window on the display device. One or more renderers draw into a render window to generate a scene (i.e., final image).
- Renderer - coordinates the rendering of lights, cameras, and actors.
- Light – illuminates the actors in a scene.
- Camera – define the view position, focal point and other camera characteristics.

- Actor – It is an object drawn by a renderer in the scene. They are defined in terms of mapper, property and transform objects.
- Property - represents the rendered attributes of an actor including object color, lighting (e.g., specular, ambient, diffuse), texture map, drawing style (e.g., wireframe or shaded), and shading style.
- Mapper - represents the geometric definition of an actor and maps the object through a lookup table. More than one actor may refer to the same mapper.
- Transform - an object that consists of a 4x4 transformation matrix and methods to modify the matrix. It specifies the position and orientation of actors, cameras, and lights.

5.3 VTK Visualization Modules

In both the workflow, HDF5 reader serves as the input. HDF5 reader provides corresponding information to the respective VTK module as mentioned under topic 4.5.

5.3.1. VTK module for 2D Time Visualization

Figure 28 represents the flowchart for converting data coming from HDF5 reader to 2D Plasma images.

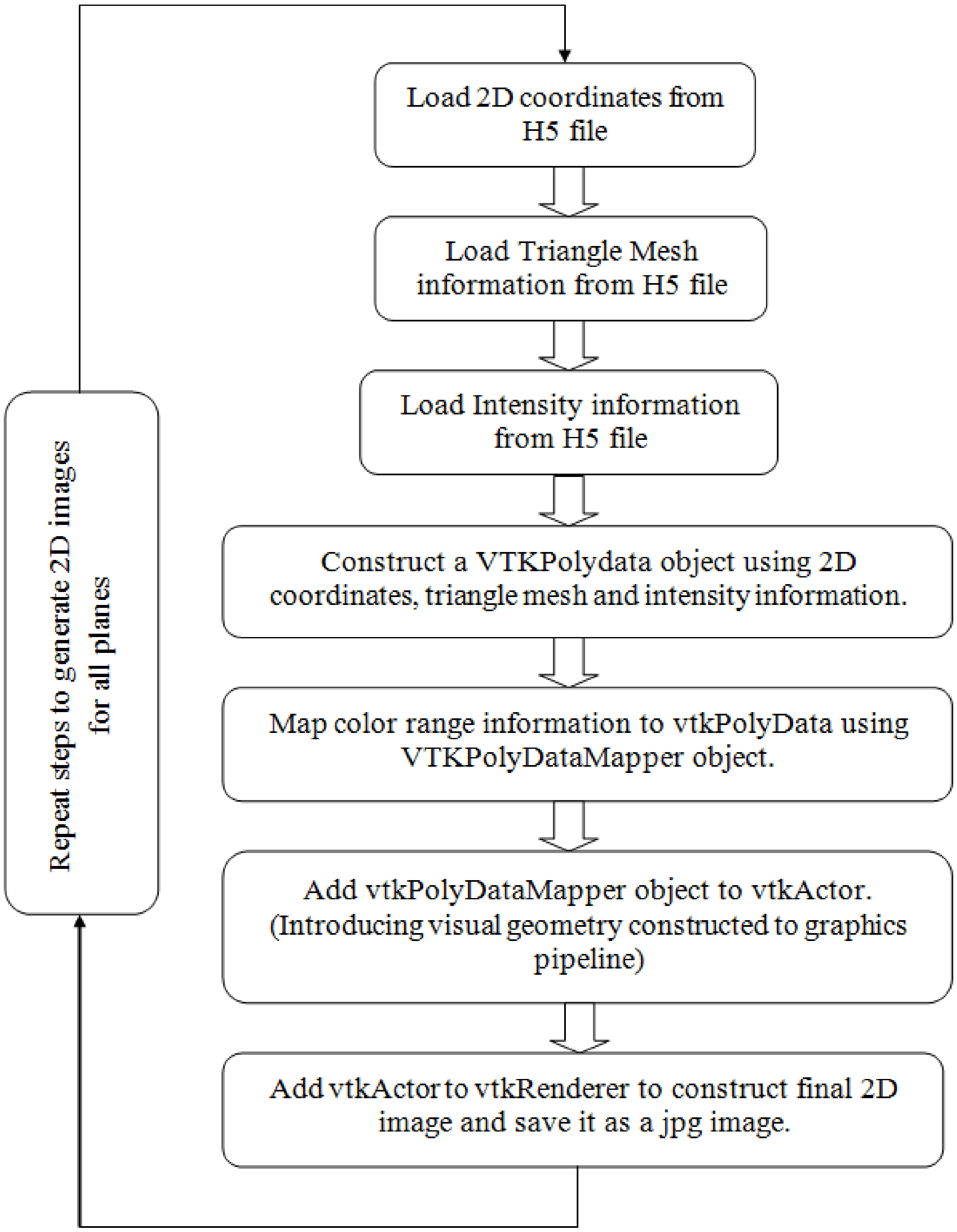

Load 2D coordinates from H5 file
Load Triangle Mesh information from H5 file
Load Intensity information from H5 file
Construct a VTKPolydata object using 2D coordinates, triangle mesh and intensity information.
Map color range information to vtkPolyData using VTKPolyDataMapper object.
Add vtkPolyDataMapper object to vtkActor. (Introducing visual geometry constructed to graphics pipeline)
Add vtkActor to vtkRenderer to construct final 2D image and save it as a jpg image.
Repeat steps to generate 2D images for all planes

Fig 28: VTK 2D Time Visualization module Flowchart

5.3.2. VTK Module for 3D Torus Visualization

Figure 29 shows the process for generate 3D torus images.

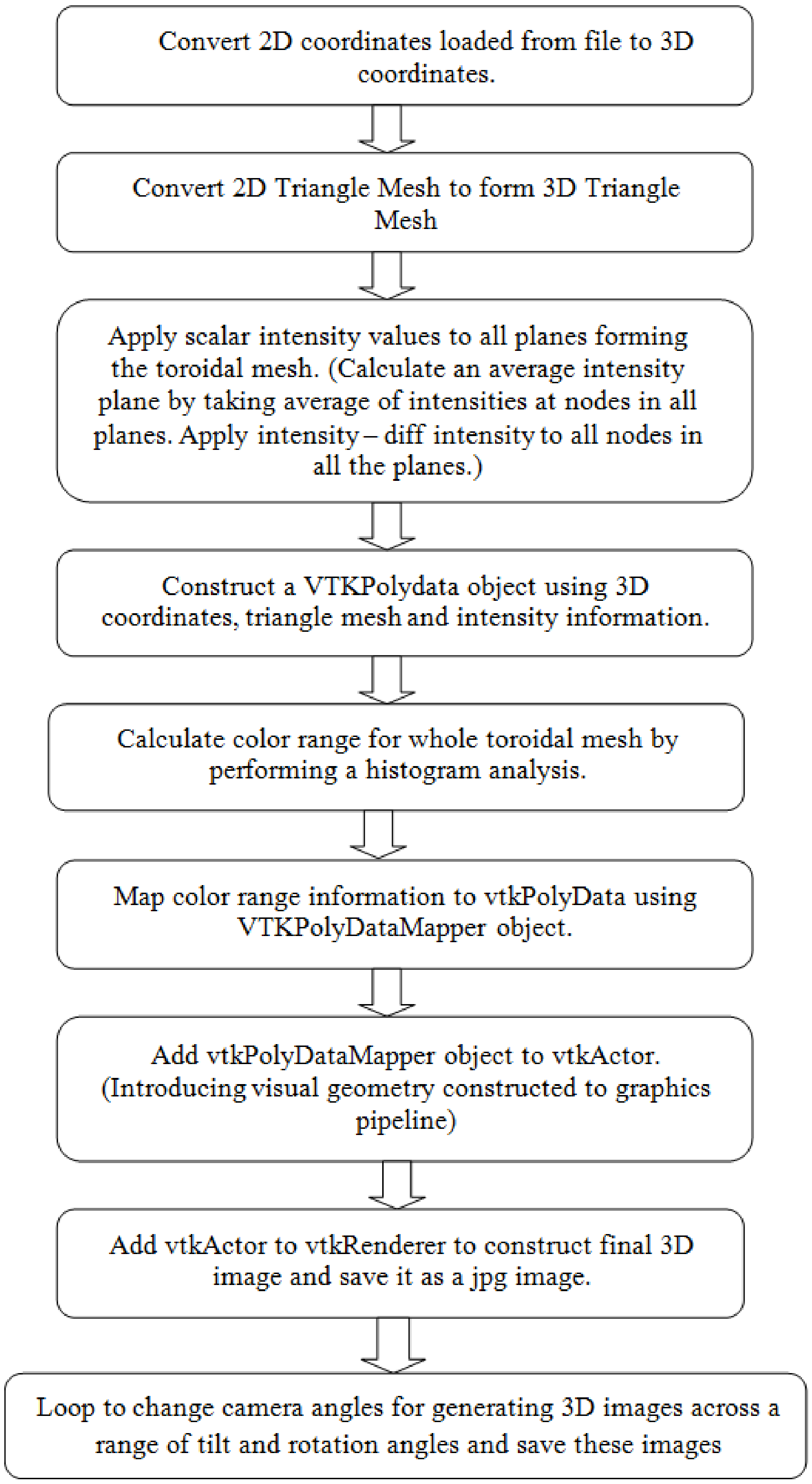

Convert 2D coordinates loaded from file to 3D coordinates.
Convert 2D Triangle Mesh to form 3D Triangle Mesh
Apply scalar intensity values to all planes forming the toroidal mesh. (Calculate an average intensity plane by taking average of intensities at nodes in all planes. Apply intensity – diff intensity to all nodes in all the planes.)
Construct a VTKPolydata object using 3D coordinates, triangle mesh and intensity information.
Calculate color range for whole toroidal mesh by performing a histogram analysis.
Map color range information to vtkPolyData using VTKPolyDataMapper object.
Add vtkPolyDataMapper object to vtkActor. (Introducing visual geometry constructed to graphics pipeline)
Add vtkActor to vtkRenderer to construct final 3D image and save it as a jpg image.
Loop to change camera angles for generating 3D images across a range of tilt and rotation angles and save these images

Fig 29: VTK 3D Torus Visualization Module Flowchart

5.4. Summary

This chapter provided information about VTK, its models and some information about various components forming its models. It also provided information on using VTK to generate both 2D as well as 3D images of plasma edges as required in both the workflows. Images generated in this module serve as an input for Flash modules used for displaying the images on the dashboard.

6. Adobe Flash

This chapter provides information on Adobe Flash (Flash) modules which are used to display plasma images in the dashboard. 2D Time Visualization images are displayed in Flash module (Fig 30) allowing users to perform 3D vector operations on the images. For 3D Torus visualization, images generated from previous step are loaded and displayed in Flash (Fig 31), by keeping track of mouse movements. With change in mouse cursor movements, correspondingly rotation and tilt angles are changed and 3D torus plasma image corresponding to those angles is loaded and displayed giving a sense of virtual 3D visualization.

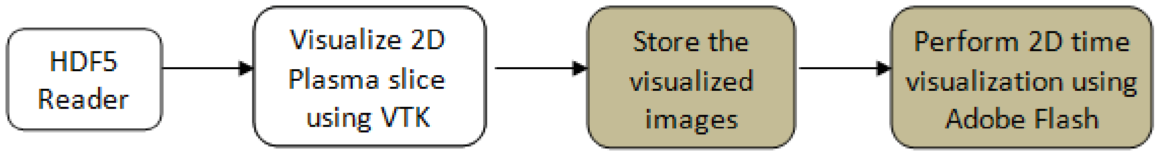

HDF5 Reader
Visualize 2D Plasma slice using VTK
Store the visualized images
Perform 2D time visualization using Adobe Flash

Fig 30: Flash module in 2D Time Visualization Workflow

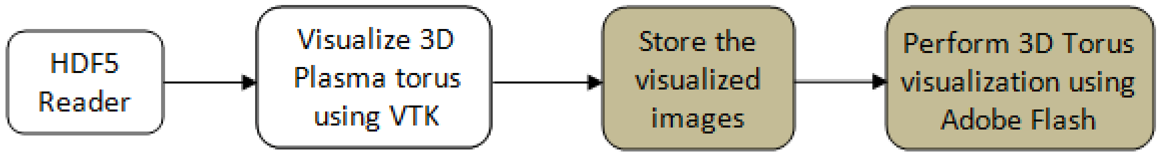

HDF5 Reader
Visualize 3D Plasma torus using VTK
Store the visualized images
Perform 3D Torus visualization using Adobe Flash

Fig 31: Flash module in 3D Torus Visualization Workflow

6.1. Introduction

Rapid growth of web led to emergence of new kind of applications called Rich Internet Applications (RIA). It is a web application having many characteristics of a desktop application. RIAs combine the responsiveness and richness of desktop software with the broad reach of web applications to deliver a more effective user experience. Users can access these applications with the help of site-specific browser, via a browser plug-in or virtual machines. Three most common platforms are Adobe Flash, Java and Microsoft

Silverlight. In this thesis, focus is on developing modules using Adobe Flash as eSimMon dashboard used Flash to handle front-end operations. To develop RIA applications using Flash, one needs to use Flex as the development environment.

Adobe Flex has been developed by Adobe to serve as a Software Development Kit for developing flash RIA application. Flex SDK provides an integrated set of tools and technology which enables developers to build and deploy scalable rich internet applications. Flex provides a modern, standards-based language that supports common design patterns. It also includes a client runtime environment, a programming model, a development environment and advanced data services. Applications developed using Flex take advantage of Adobe Flash Player which enables developers to seamlessly extend the capabilities of the browser. Using Flash Player, developers are able to deliver richer, more responsive client – side applications as well as more robust integration with server – side functionality and service-oriented architecture can be achieved.

In this thesis, Flex was used to develop flash modules for two workflows – 2D Time Visualization and 3D Torus Visualization for displayed images on the web and providing a virtual 3D environment for performing vector operations.

6.2 Advantages of using Flex and Flash

- Enhanced user experience – Flex helps in building application that provide an engaging user experience. This is made possible with various set of controls, widgets available in Flex SDK. For example: - To apply geometric transformation to 2D images loaded in Flash, event listeners were used. With help of these event listeners, Flash internally keeps track of input changes and applies these changes to the images.

- A complete environment – Flex is a powerful application development solution for creating and delivering RIAs within the enterprise and across the web.
- Common deployment environment – Output of a Flex application (.swf) file executes on Flash Player. As Flash Player is platform independent, there is no need for users to install custom client software. Also, Flash player runs consistently in all the browsers and platform, developers need not worry about inconsistent behavior in different client environments. For example: - Web pages developed using HTML, DHTML and CSS behave differently on different browsers, due to fact that each browser has their own implementation of engine to layout these images. Thus, developer has to take care of all the scenarios when developing web application using other technologies.
- Enterprise-class features – Flex provides developers with gamut of Data Services options for transparently synchronizing data and supporting real-time data push. One can developed Client-side only applications to applications which provides data access using HTTPService and WebService to applications providing data access with Flex data services. In this thesis, both modules developed take advantage of data access using HTTPService to dynamically load XML file and images.
- Direct User feedback – A good developer must make sure to provide feedback to user, if he makes an input error or enters invalid information. Flex ensures the quality of input data using various formatters and validators.

- Eliminating page load – As output of Flex application is a compiled .swf file that contains all the components such as CSS files, graphics library objects etc. Due to this applications running in Flash Player behave like desktop applications, instead of series of linked pages. Flash player manages the client interface as a single, uninterrupted flow, thus not requiring a page load from the server when the client moves from one section of the application to another.
- User Interactiveness – Flex uses MXML and ActionScript for developers programming needs. Both these languages provide an event model with the help of various event listeners to provide interactivity.

- Standards-based architecture – Flex using two types of languages to implement the desired functionality – MXML and ActionScript. All three Flex, ActionScript and MXML are complying with existing standards. MXML for example is XML compliant; ActionScript is am ECMAScript-based language that provides support for object oriented development. Flex server uses J2EE Java technology for its execution.

These above mentioned features make Flex environment popular among web and graphics developers.

6.3 MXML and ActionScript

Flex provides developers the flexibility to develop their code using either MXML or ActionScript. MXML is an XML markup language that developer can use to lay out user-interface components. MXML is very similar to HTML and it provides a much richer tag set than HTML. ActionScript on the other hand is object-oriented programming language similar to core JavaScript. ActionScript has built-in objects and functions, and also provides flexibility to developers to create custom objects and functions.

By doing this, Flex has tried to make Flash Player popular among various groups such as web developers and graphics developers. Web developers, who used to code previously using HTML, can take advantage of MXML to develop web components which they use to develop earlier using HTML. Similarly, graphics developers who used to code previously using other object oriented graphics languages such as OpenGL can take advantage of ActionScript to develop applications. Time required to migrate to these languages is very less, as both of them are very close to languages previously used by the developers.

6.3.1. MXML

MXML is an XML markup language, which can be used to lay out user-interface components. It can be also used to declaratively define non-visual aspects of an application, such as data sources on the server and data bindings between user-interface components and these data sources.

To achieve above mentioned utilities MXML provides various tags to developer. These tags are very similar to tags available in HTML, which makes MXML very close to HTML with their share of differences. MXML provides developers with much richer tag set and is more structured than HTML. Also, MXML defined applications are compiled into SWF files and they use Flash player for rendering purposes, thus providing a richer and more dynamic user interface than page – based HTML applications do.

Various tags provided by MXML can be used to serve variety of purposes such as: -

- Creation of visual components such as data grids, trees, tab navigators, accordions and menus.
- For using non-visual components in the application. These components include web – service connections, data bindings and animation effects.
- Use of various event listeners. Flex us to use various event listeners. Event listeners are objects which listen for any activity on the associated object. When any activity happens on the associated object, event listeners catch that event and perform the desired operation.
- MXML can also be extended with custom components that can be referenced in future using MXML tags.

In addition, Flex provides us with the ability to write whole code in one MXML file or to break it down in multiple files. This helps is much better organization of code.

6.3.2. ActionScript

Adobe Flex developers are provided with ActionScript, which helps in extending the functionality of their Flex applications. It is an object oriented programming language for Adobe Flash Player. It is very similar to the core JavaScript programming language. ActionScript has built-in objects and functions, and it also allows for creation of custom objects and functions. Main use of ActionScript in a Flash application is to provide flow control and object manipulation features that are not available under strict MXML.

Various areas where ActionScript can be used inside a Flex application are:-

- Define event listeners – ActionScript is used to provide define functions which operates on data after an event has occurred.

- Getting or setting component properties.
- Handling callback functions
- Creating of new classes, packages and components.
- Referencing external classes and packages to handle more complex tasks. By doing this, one takes advantage of object-oriented programming concepts such as code reuse and inheritance.

Based on above analysis of MXML and ActionScript, most of the Flex applications will use a mixed version of design using both MXML and ActionScript. MXML is used generally to layout components and ActionScript is used to handle programming part of these components.

6.4 Flex Modules

In both workflows, images were loaded dynamically in Flash module using XML as the source input. The XML file contains information about the name of the images, location of images. More information is available under topic 2.4 of this thesis.

For 2D Time visualization module, various geometric transformations were applied to loaded 2D images (representing cross-section of plasma toroid) giving an illusion of 3D in time. These geometric transformations were applied using various event listeners available in Flash.

For 3D Torus visualization module, 3D images (representing torroidal 3D plasma mesh) at different tilt and rotation angles were loaded. Current image displayed on Flex stage was changed using mouse movements giving an illusion of 3D in space.

Output of both flex modules was a compiled flash file which helped scientists to view these modules remotely.

6.4.1. Flash module – 2D Time Visualization

Figure 32 shows the flowchart developing Flash 2D Time visualization module.

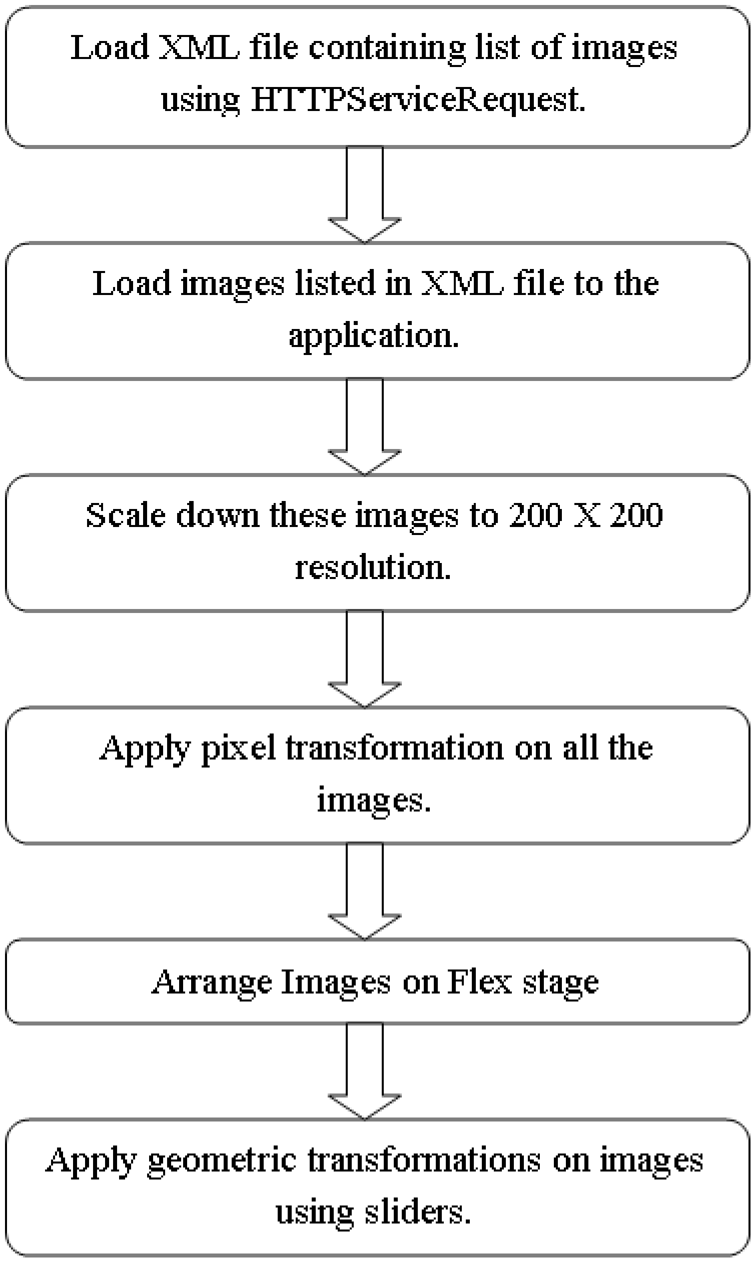

Load XML file containing list of images using HTTPServiceRequest.
Load images listed in XML file to the application.
Scale down these images to 200 X 200 resolution.
Apply pixel transformation on all the images.
Arrange Images on Flex stage
Apply geometric transformations on images using sliders.

Fig 32: Flash 2D Time Visualization Module flowchart

6.4.2. Flash Module – 3D Torus Visualization

Figure 33 shows the flowchart developing Flash 3D Torus visualization module.

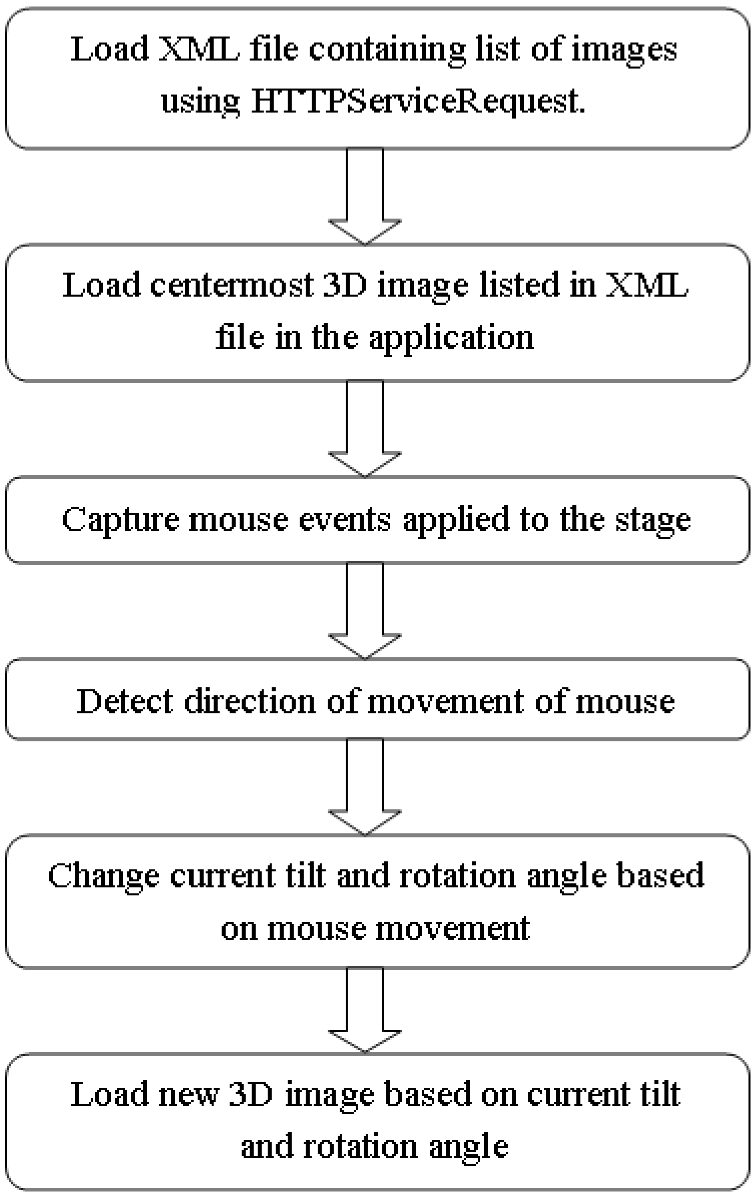

Load XML file containing list of images using HTTPServiceRequest.
Load centermost 3D image listed in XML file in the application
Capture mouse events applied to the stage
Detect direction of movement of mouse
Change current tilt and rotation angle based on mouse movement
Load new 3D image based on current tilt and rotation angle

Fig 33: Flash 3D Torus visualization module flowchart

6.5. Summary

These Flash modules complete both the workflows and next chapter shows the result obtained at the end of these workflows.

7. Results

7.1 Output of 2D Time Visualization Flash module

2D images representing cross-sectional view of plasma toroidal plane were loaded using XML file in Flash. After loading was complete, images were then pre-processed to remove black pixels from the images and replace them by alpha induced pixels. This added transparency to the images. After completion of this stage, these images were controlled by four different sliders, each performing a specific task.

- Rotating images across Y – axis.
- Rotating images across Z – axis.
- Increasing or decreasing the space between images.
- Performing zoom in or zoom out operation on images.

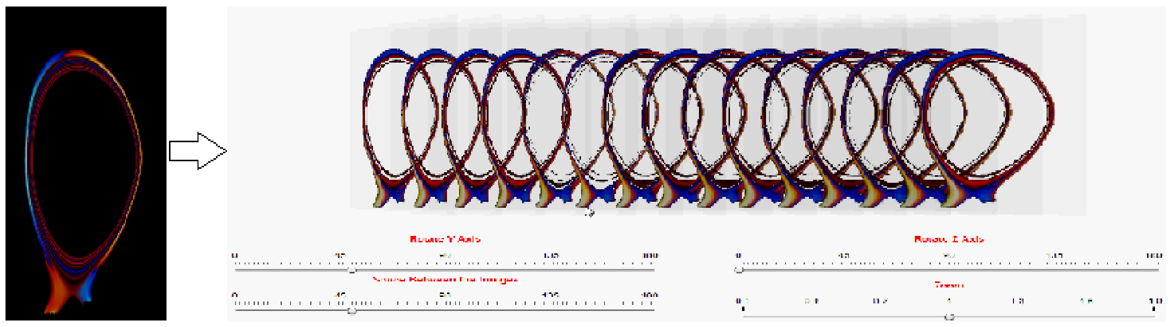


Fig 34: 2D preprocessed images loaded

Figure 34 shows 2D images loaded and preprocessed in flash module. It can be see that black pixels in image at left hand are converted to white and alpha value has been added to these images. This alpha value makes the images transparent which helps in much better analysis.

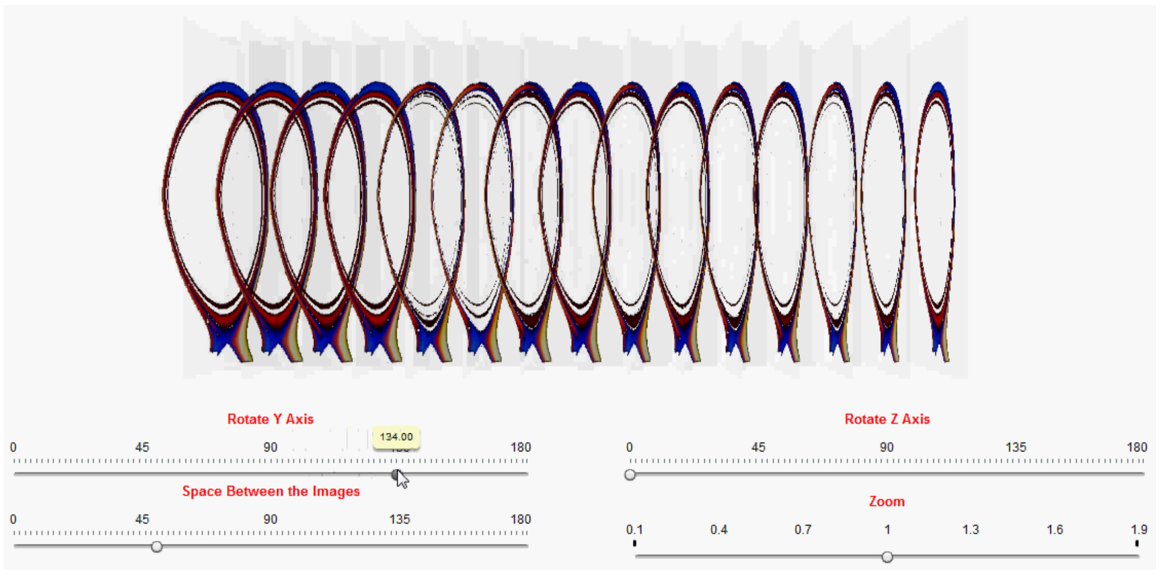


Fig 35: Rotation around Y-Axis

Figure 35 represents rotation of images across Y axis which slider events originating from slider ‘Rotate Y Axis’.

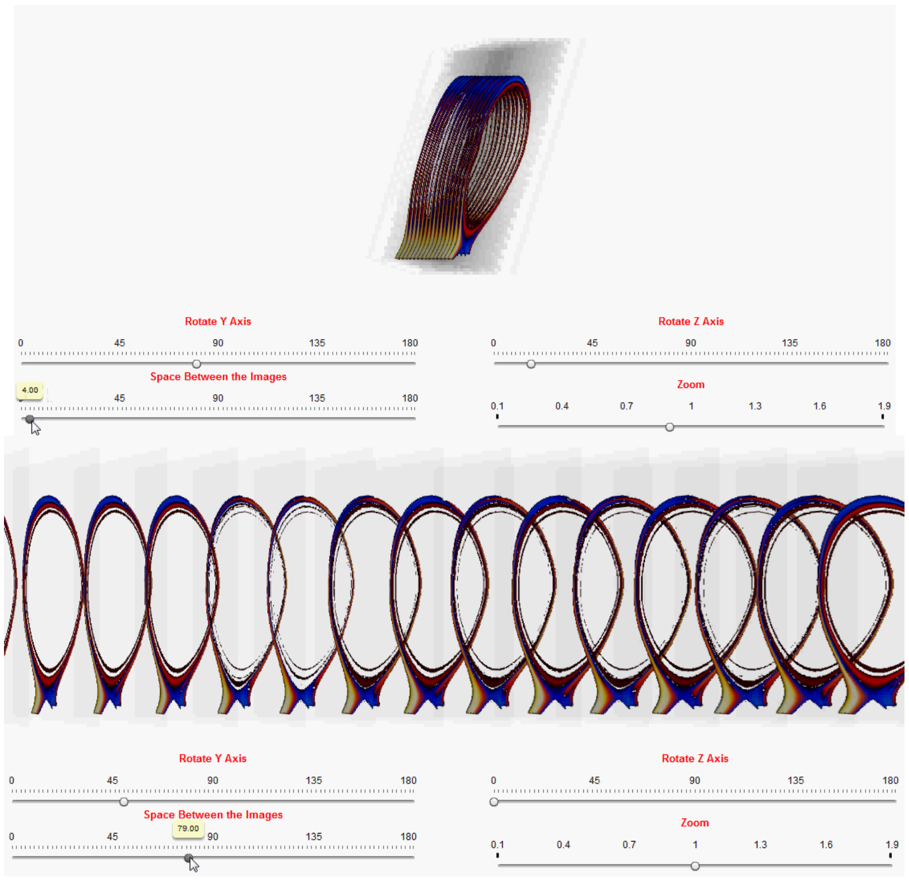


Fig 36: Controlling Space between the images

Figure 36 represents action of slider “Space between the images” which controls spacing between the images.

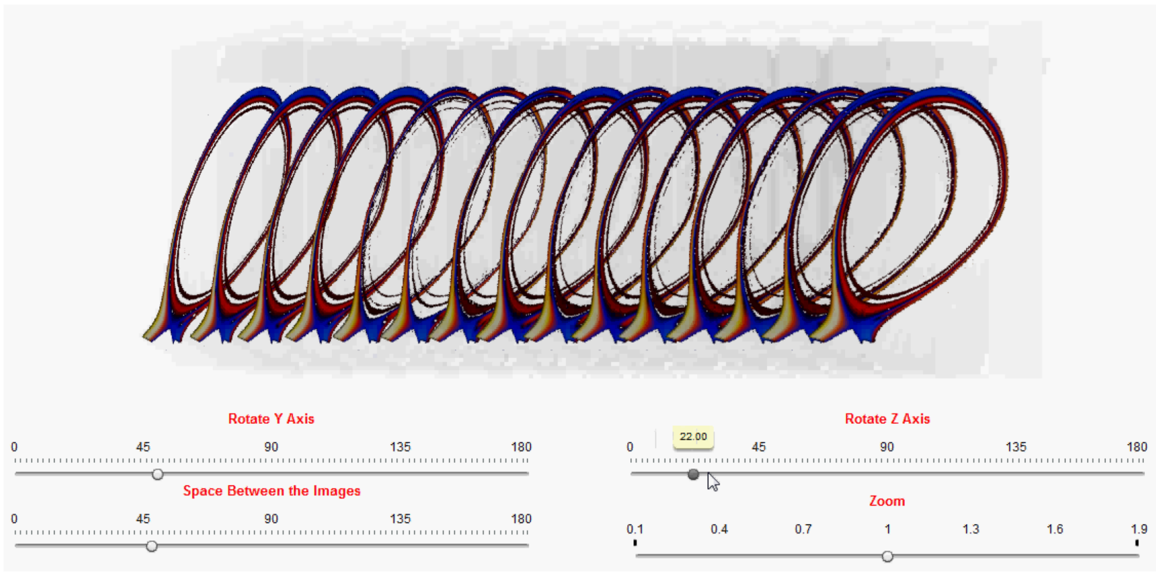


Fig 37: Rotating Images across Z-Axis

Figure 37 shows rotation of images across Z axis with help of slider “Rotate Z Axis”.

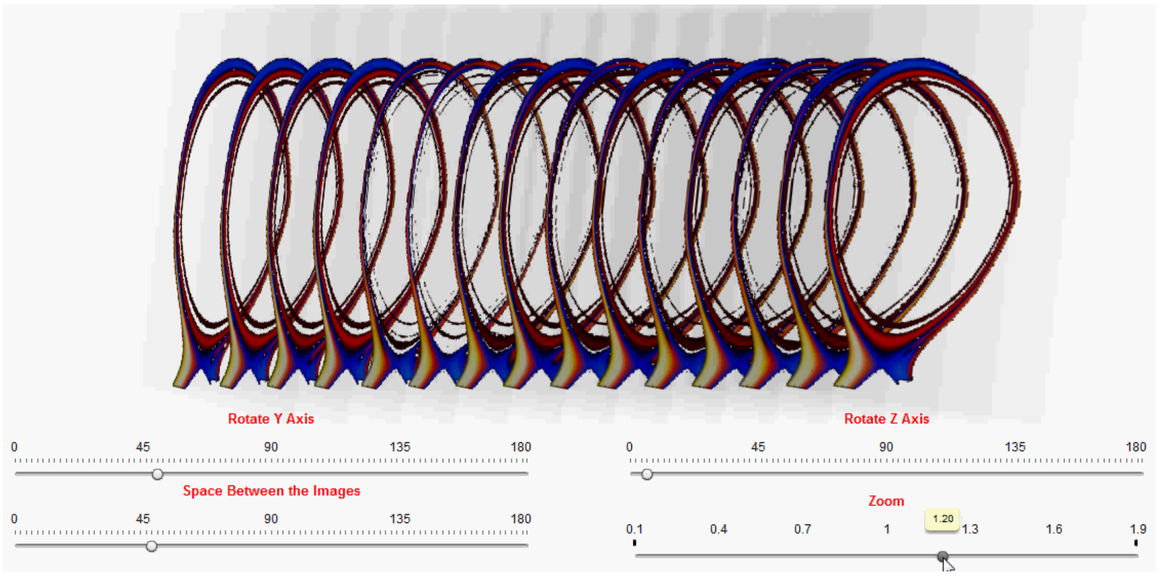


Fig 38: Zooming Operation

Figure 38 show zoom operation. Events generated from slider “Zoom” control magnification level of images.

7.2 Output of 3D Torus Visualization Module

In this module 3D images of plasma toroid were loaded from xml file. These images are 2D images which through perspective projection represent 3D toroidal plasma image. To create a virtual 3D environment, a set of images were generated at various rotation and tilt angles. Rotation angles are along the horizontal plane and they have a range from 0 to 350 degree. Tilt angles are along vertical plane and they have a range from -90 to 90 degree. Based on mouse cursor movement, rotation and tilt angles were correspondingly changed. These changes in rotation and tilt angles, correspondingly loads up torus image pertaining to those angles, thus giving an illusion of 3D.

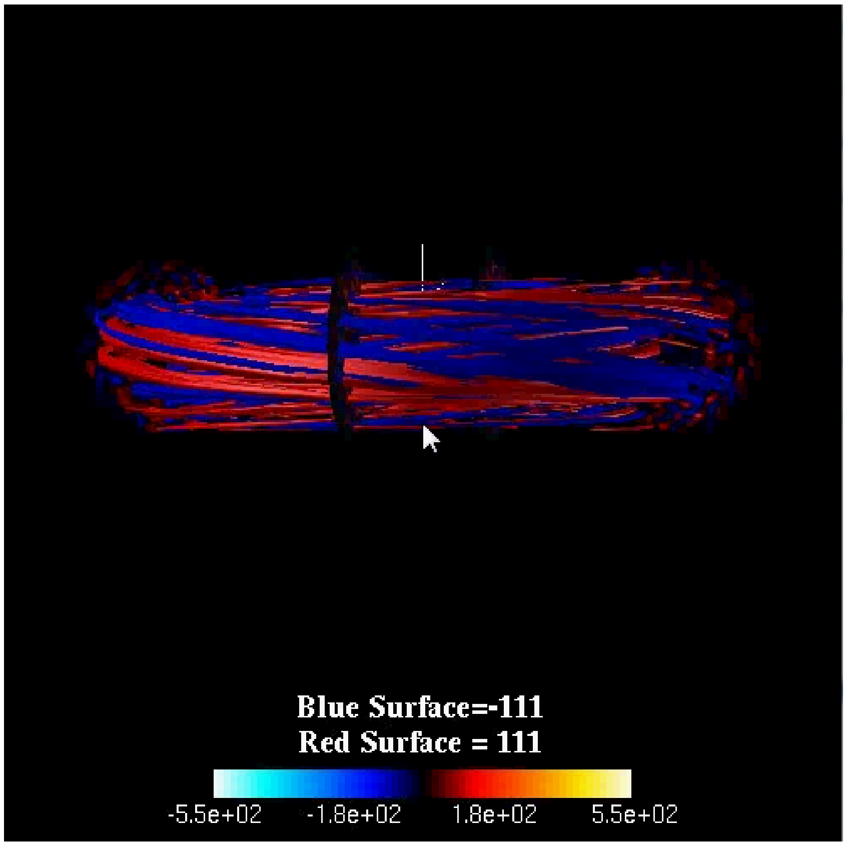


Fig 39: Image at 0 degree tilt and 0 degree rotation angle loaded at startup

Figure 39 shows image at 0 degree and 0 degree rotation angle is loaded when module starts.

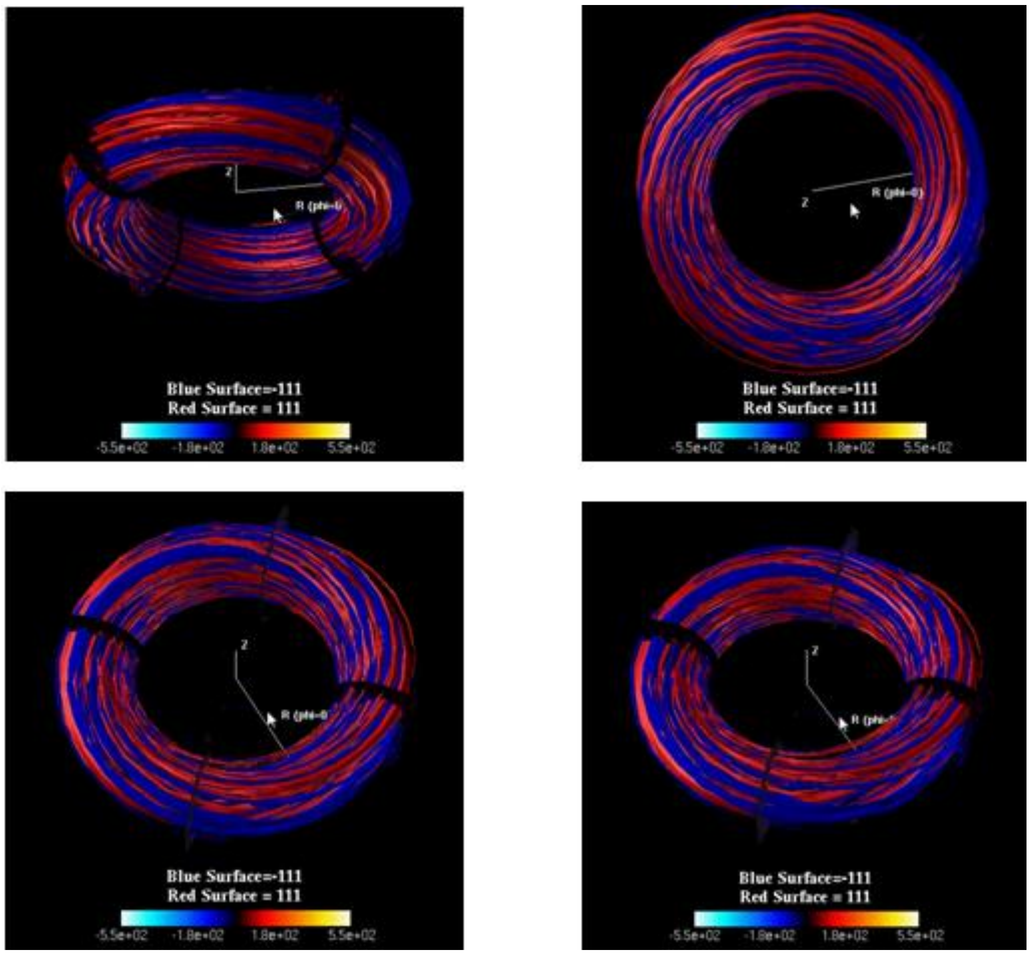


Fig 40: Images loaded according to tilt angle
(Tilt angle changes on up and down movement of mouse cursor)

Figure 40 shows current image changing as per change in tilt angle, due to mouse cursor up and down movements

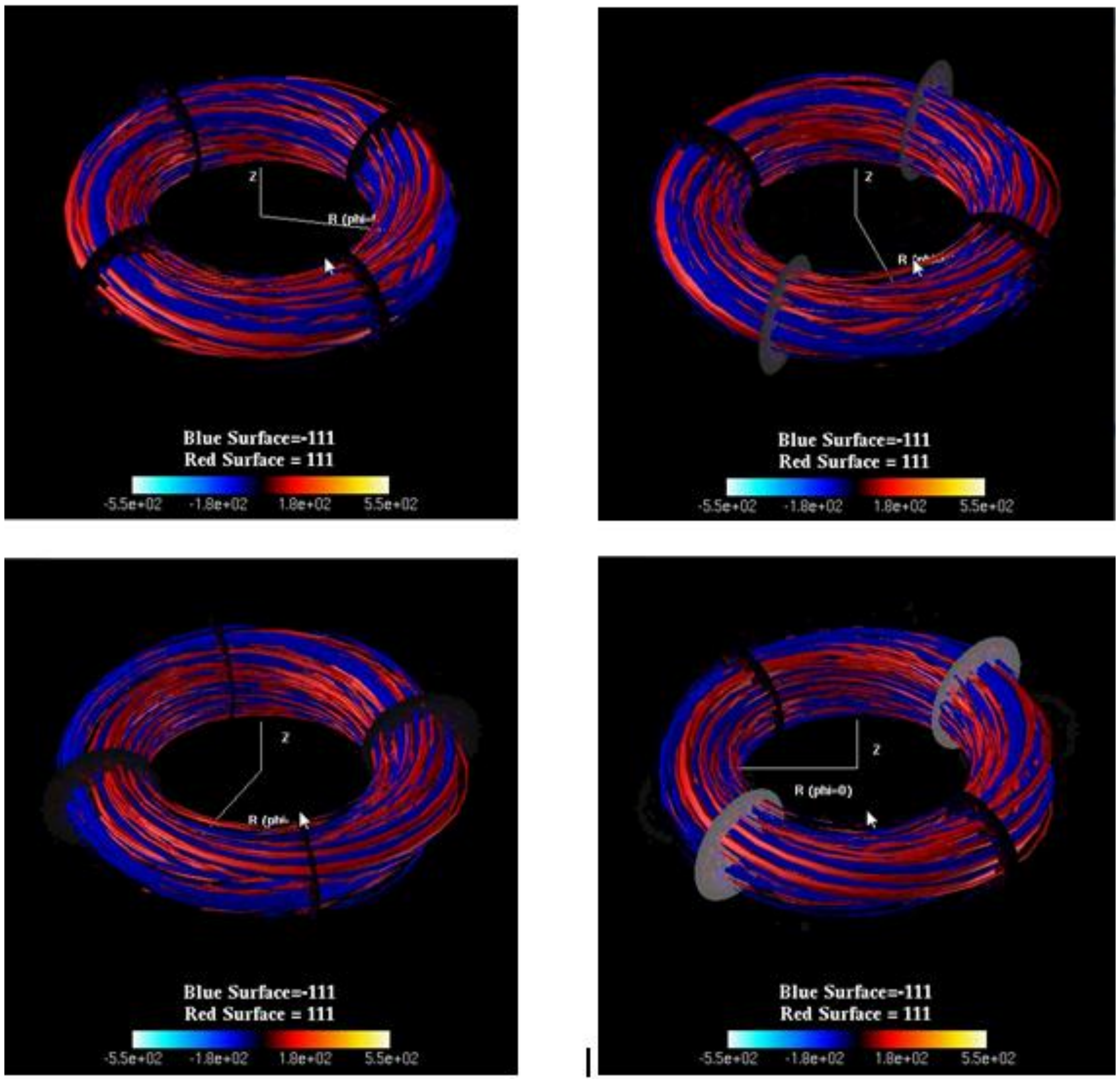


Fig 41: Images loaded according to rotation angle
(Rotation angle changes on sideways movement of mouse cursor)

Figure 41 shows how images are changed based on rotation angles values getting changed. Rotation angle values are changed based on sideways movement of mouse cursor.

8. Conclusion

Visualization is a very powerful tool to help scientists in the analysis of numerical data. VTK provides us with a rich library of visualization tools and functions aiding in visualizing both 2D as well as 3D data. In this thesis, a module developed for visualizing plasma data was ported from AVS to VTK. This helped in avoiding huge licensing costs involved for using AVS. VTK module was a standalone dataflow visualization module producing both 2D and 3D images using given 2D dataset. To bring this dataset to memory a custom HDF5 reader was developed using C++.

In addition to that, to help scientists more in analysis of data and provide them with the option to view these images over web, two modules were developed using Flex for each type of image. For both flash modules, images were loaded using HttpListener object and XML. For 2D images, Flash module was capable of applying various geometric transformations to these loaded images. Using 3D image Flash module, a virtual realization of 3D in space was created where a set of 3D images based on tilt and rotation angles were loaded. Tilt and rotation were changed depending on mouse movements and corresponding 3D image was displayed on screen.

Thus, this thesis showcases how visualization can be done by using visualization libraries of VTK and data representation libraries of HDF5. It also shows a way of how real time interactivity can be provided over web using Flash.

9. References

1) Julian Cummings, Jay Lofstead and Karsten Schwan, Alexander Sim and Arie Shoshani, Ciprian Docan and Manish Parashar, Scott Klasky, Norbert Podhorszki and Roselyne Barreto. EFFIS: an End-to-end Framework for Fusion Integrated Simulation, submitted to PDP 2010, Pisa, Italy, February 2010

2) Cummings, J. and Pankin, A. and Podhosrzki, N. and Park, G. and Ku, S. and Barreto, R. and Klasky, S. and Chang, C. S. and Strauss, H. and Sugiyama, L. and Snyder, P. and Pearlstein, D. and Ludäscher, B. and Bateman, G. and Kritz, A. (2008) Plasma Edge Kinetic-MHD Modeling in Tokamaks Using Kepler Workflow for Code Coupling, Data Management and Visualization. Communications in Computational Physics, 4 (3). pp. 675-702. ISSN 1815-2406

3) Scott Klasky, Bertram Ludaescher and Manish Parashar, The Center for Plasma Edge Simulation Workflow Requirements, ICDE Workshops pp.73, 2006

4) Naveen Atmakuri. Feature Tracking & Visualization in VisIt. Master's Thesis, Rutgers The State University of New Jersey, Electrical and Engineering Department.

5) M. Friendly, "Milestones in the History of Thematic Cartography, Statistical Graphics, and Data Visualization", 2008, www.math.yorku.ca/SCS/Gallery/milestone/

6) Will Schroeder, Ken Martin and Bill Lorensen. Visualization Toolkit – An Object- Oriented Approach to 3 D Graphics

7) Vtk File Formats – Extract from VTK User's Guide - http://www.vtk.org/VTK/img/ file-formats.pdf

8) OBJ File Format – Extract describing obj file format - http://www.royriggs.com/obj.html

9) Jeremy Nowell. Using VTK and OpenGL Graphics Libraries on HPCx, The University of Edinburgh, 2005

10) HDF5 User's Guide, 2011 - http://www.hdfgroup.org/HDF5/doc/UG/index.html

11) Why HDF - http://www.hdfgroup.org/why_hdf/

12) Marcel Ritter, Introduction to HDF5 and F5, Bachelor Thesis, Institute of Computer Science, Leopold-Franzens-Universität Innsbruck

13) HDF5View home page - http://www.hdfgroup.org/hdf-java-html/hdfview/UsersGuide/ug01introduction.html#ug01installation

14) Kitware home page - http://www.kitware.com

15) Visualization toolkit (VTK) homepage – www.vtk.org

16) William J. Schroeder, Kenneth M. Martin and William E. Lorensen, The Design and Implementation of an Object-Oriented Toolkit for 3D graphics and Visualization, GE Corporate Research & Development

Appendix – I

Building VTK on Windows using CMake and Visual Studio 2008

To install VTK, one needs to download vtk source, vtk data and CMake files from the following links.

- Vtk source – download it from http://www.vtk.org/VTK/resources/software.html. Current version of source file is vtk-5.6.1.zip. Download and extract the files to a folder named ‘vtk’.
- VTK data – download it from http://www.vtk.org/VTK/resources/software.html. Current version of data file is vtkdata-5.6.1.zip. Download and extract the files to folder named ‘vtkdata’.
- CMake – download 32-bit CMake windows installer from http://cmake.org/cmake/resources/software.html. Current version of CMake 32-bit installer is cmake-2.8.4-win32-x86.exe. After downloading the executable, click on it to install CMake.

After you have installed CMake, start CMake (CMake-gui) from Windows start menu. In the textbox “where is the source code” enter the path of folder named ‘vtk’, let say is C:\vtk. Enter build path for the binaries as vtkbuild, (C:\vtkbuild). If this folder is not created, CMake will prompt you to create a folder “vtkbuild”. Click on configure button and choose “Visual Studio 2008” as the build platform.

After configuration has been done, various options will get displayed. Out of all the options, choose the corresponding values for the below mentioned options and leave others as it is.

BUILD_EXAMPLES – ON

BUILD_SHARED_LIBS – ON

CMAKE_INSTALL_PREFIX – Path of folder called ‘vtk’. I choose it as C:\vtk.

VTK_DATA_ROOT – Path of folder called ‘vtkdata’. I choose C:\vtkdata.

VTK_USE_RENDERING – ON

VTK_USE_VIEWS – ON

After entering values for the above mentioned options click on the ‘Generate’ button. CMake will create files and folders in ‘vtkbuild’ (C:\vtkbuild) directory. After building process of CMake is over, go to ‘vtkbuild’ folder and search for file called ‘vtk.sln’. Open this file using Visual Studio 2008. (Note: - If you are using Windows Vista or XP, make sure you run Visual Studio 2008 as administrator during the build process). Once ‘vtk.sln’ is opened in Visual studio, right click on ‘ALL_BUILD’ project and choose build solution option. Normally a build process takes half an hour to one hour depending on speed of the machine.

If the build process is unsuccessful, delete the folders and repeat the whole process. A successful build will create new files will be created in these three folders. Copy all the ‘dll’ files from ‘bin\debug’ subfolder of ‘vtkbuild’ (C:\vtkbuild\bin\debug) to ‘C:\Windows\System32’.

After this has been done, open visual studio and create and empty C++ project. In order to make sure that Visual studio environment is able to find VTK libraries, one need to make the following changes.

Click on Tools -> Options -> Projects and Solutions -> VC++ Directories. On right side upper corner of the dialog box, a "Show Directories" drop-down menu will appear. Select 'include files' option from the drop-down menu entries and add following directories to it: -

- C:\vtkbuild
- C:\vtk-5.0.4\Rendering
- C:\vtk-5.0.4\Parallel
- C:\vtk-5.0.4\IO
- C:\vtk-5.0.4\Imaging
- C:\vtk-5.0.4\Hybrid
- C:\vtk-5.0.4\Graphics
- C:\vtk-5.0.4\Filtering
- C:\vtk-5.0.4\Common

Now, select 'Library files' option from the same drop-down menu and add below mentioned entries: -

- C:\vtkbuild\bin
- C:\vtk\lib

VTK is now installed and ready to be used in the projects.

Appendix II

Building HDF5 library on Windows using Visual Studio 2008

I will be building HDF5-1.8.5 version of the HDF5 libraries. To build this library you need to download HDF5-1.8.5 source code and compiled versions of zlib and szip libraries. These can be easily obtained from the following links: -

- HDF5 Source code – It can be downloaded from http://www.hdfgroup.org/ftp/HDF5/prev-releases/hdf5-1.8.5/src/hdf5-1.8.5.tar. Extract the content of this archive to folder named “hdf5”. Let say the location of this folder is “C:\hdf5”.
- Szip and Zlib Compiled code - It can easily obtain from ‘http://www.hdfgroup.org/ftp/HDF5/prev-releases/hdf5-1.8.5/bin/windows/hdf5-1.8.5-win32.zip’. Extract the contents of the archive to holder called “hdf5-win32”. Now, create a folder called “compress” (C:\compress) and inside it create two sub-directories called ‘include’ and ‘lib’. Copy all the ‘.h’ files not starting with ‘H5’ from the include folder of ‘hdf5-win32’ to ‘C:\compress\include’. Similarly, copy all the dll and lib files, not starting with hdf5 from dll and lib folders of “hdf5-win32” to “C:\compress\lib”.

After this, one needs to create two system variables in Windows by going to:

Start -> Control Panel -> System -> Advanced -> Environment Variables and then, create two user variables with following values: -

- HDF5_EXT_SZIP = szip.lib

- HDF5_EXT_ZLIB = zlib1.lib

Copy both szip.dll and zlib1.dll to System32 folder “C:\Windows\System32”.

Now, Invoke Microsoft Visual Studio and go to "Tools" and select "Options", find "Projects", and then "VC++ Directories". In the drop-down menu “Show Directories for”, choose “Include Files” and add the path for include sub-folder of compress directory (C:\compress\include) to the list of included directories.

Similarly, for “Library Files” option in “Show Directories for” drop-down, add the path for lib sub-folder of compress directory (C:\compress\lib) to the library directories.

Go to your HDF5 folder (C:\HDF5) and run copy_hdf.bat. This batch process copies all the necessary batch files the necessary batch files, Windows-specific source code and text files saved under c:\MyHDFstuff\hdf5\windows directory to the corresponding directories under hdf5.

Open “all.sln” project using Visual Studio available under the windows directory. (c:\ hdf5\windows\proj\all\all.sln). Select “Build”, and then select “Configuration Manager”. Chose ‘Active Solution Configuration’ as ‘Debug’ and then, Select "Build" -> "Build Solution" or "Rebuild Solution" to build debug version of project "all".

After build process is completed, run the batch file ‘installhdf5lib.bat’ under ‘C:\HDF5’ directory. This batch file relocates all HDF5 libraries build in previous step under one folder named ‘hdf5lib’ available under “C:\HDF5”.

The <debug> layout of <hdf5lib> should be:

```
  debug\include   --  HDF5 header files
  debug\bin       --  HDF5 static tool executables
```

debug\bindll	-- HDF5 DLL tool executables
debug\lib	-- HDF5 static libraries
debug\dll	-- HDF5 DLLs