A Graphical Framework for Constructing and Executing Computational Networks

Christopher L. Hall
Consortium Research Fellows Program
Dayton, OH, USA

Vincent A. Schmidt
US Air Force Research Laboratory
Wright Patterson Air Force Base, OH, USA

Abstract—Research in multispectral data visualization frequently consists of experimenting with combinations of diverse fusion and visualization algorithms. This paper describes the design and implementation of a flexible graphically-based software utility for rapidly constructing sequences and networks of algorithms, drawn from a predefined and expandable library of algorithmic building blocks.1

Keywords-model execution; simulation

I. INTRODUCTION

Research in multispectral data visualization frequently consists of experimenting with combinations of diverse fusion and visualization algorithms. Researchers commonly program custom user interfaces to display the intermediate and final results of algorithm sequences and networks, or must work with software developers to generate the desired graphical interface. Furthermore, it is often the case that the principle difference between successive experiments is merely the addition of a new algorithm, the modification of algorithm parameters, or a change in the execution order or connectivity of a network of algorithms. Considerable time and effort could be saved if the experimenter could easily and quickly alter such characteristics independently, without heavily relying on software development personnel to make the modifications.

In general, it would be beneficial for a researcher to be able to rapidly construct and execute a sequence or network of algorithmic building blocks, each with its own set of parameters, quickly and easily in a graphical environment, with minimal technical assistance. The constructed networks could be executed as complete self-contained tests or experiments, or used as a preliminary test prior to full-scale study development.

The graphical utility described by this research provides this capability. A collection of predesigned algorithmic “blocks” are available for use in a drag-and-drop graphical user interface (GUI) window. Non-programmer experimenters can use the blocks to graphically build their own network design, and can execute network configurations without requiring additional software support. Programmers are still needed to lay down the initial framework of the models, but no extra time is spent on creating a whole new interface for each experiment that might vary from the previous study by mere parameter settings.

Flexibility is the key to this utility because it not only suits the original purpose of aiding multispectral visualization research, but is also able to promote numerous other uses based on the library of blocks used. The utility’s base design functionality and execution behavior is determined by the model identified at startup — using the multispectral visualization library presents the user with multispectral visualization algorithms and operations, but using the logic library yields blocks of low-level logic operators. Other libraries could also be easily added. In this way, the utility can be considered as a sort of Computer-Aided Design (CAD) tool, where the designs are also executable.

Note that the purpose of this engineering research is to continue to develop and test the utility described in order to:

1. Determine if such a GUI utility would be valuable for the design and evaluation of multispectral fusion and visualization research experiments in the laboratory environment,

2. Determine if such a GUI utility would be useful for non-programmer data analysts for rapidly prototyping networks performing certain information processing operations,

3. Provide a simple framework for adding new visualization research operations,

4. Collect multispectral fusion and visualization algorithms into a single reusable library, vs. the current configuration which distributes the implementation of visualization and fusion algorithms by storing them in an experiment-by-experiment fashion,

5. Determine the value of such a GUI utility in the more general CAD-like sense, where substituting one library for another completely and easily redefines the functionality offered by the utility.

This paper describes the utility framework and concepts, discusses the implementation details, and gives examples (with figures) of how networks can be designed and executed. The concluding section identifies near-term future work and sets expectations for the utility’s initial operational capability.

---

1 Distribution A: Approved for public release; distribution unlimited, PA# 88ABW-2010-3356
II. FRAMEWORK AND CONCEPT DESCRIPTION

The research concept is the design and implementation of a flexible GUI-based software utility to rapidly graphically design, manipulate, and execute networks of functional algorithms (“blocks”). The software utility operates as both a design and execution framework.

As a design framework, algorithms are represented as graphical objects, or blocks. A layout is constructed by selecting blocks from a collection of available blocks, and connecting them together to form a network. The network can be a trivial linear sequence, leading from a single input to one or more successive blocks, or it can be a collection of inputs leading to blocks with multiple inputs and multiple outputs. Fig. 1 shows a simple layout of boolean logic blocks.

A model defines the set of available blocks that can be used to construct a layout. The dialog box displayed in Fig. 1 illustrates the blocks currently available from the logic model. Each block defines a specific function or operation. Block and model implementation are described in the next section.

In design mode, the user must specify the model to use when starting the program. If a new layout is being constructed, the user specifies the number and type of inputs, and selects and places the desired blocks on the screen. The model defines the blocks available for use.

As blocks are placed on the screen (design board), a graphical representation of the block is drawn. These representations can be defined by drawing commands within the model, or simply loaded as predesigned graphic files (such as .gif or .jpg). Fig. 1 shows an example of both graphic files and constructed drawings. The input values shown as triangles are drawn with the constructor while the logic gates are all created from image files.

When the application operates as a GUI design program, the user is able to add, delete, select, move, and edit each individual block and the associated connections by using the mouse. The mouse is context-dependent, so clicking over a block does something different than clicking on the design board.

The same dialog box that identifies functional blocks used to build layouts also contains functions for moving and connecting blocks. To move objects, click the Select option from the dialog box and move the desired object using the mouse. To connect two objects, click the Wire option use the mouse to drag and drop a connection from the source object to the destination object.

Blocks can be connected at any time during network construction. Network connectivity between blocks is very general. Blocks are able to have multiple inputs and outputs, allowing them to be connected a variety of ways.

Our preliminary execution mechanism did not allow for recurrent or cyclic networks; only feedforward networks are currently supported. However, we are already in the process of updating our execution algorithm to support recurrent networks.

The display in Fig. 1 shows the application’s main screen, with an example of a network using the digital logic model. This model contains a set of blocks describing basic digital logic functions. All blocks offered by the model are block designs that are dynamically populated from the model's XML [1]. All blocks are placed on the display by clicking the desired selection from the dialog box, then clicking on the design board. The block details and specifications can be altered through the menu options when right clicking on an individual block.

In all layouts, start blocks are also considered to be specific blocks because they represent inputs into the network when execution occurs, and should be compatible with the model that is being used. For example, if the logic model is being used, 1’s and 0’s should be executable inputs, but an imaging model network should use forms of imagery as inputs.

In execution mode, the user identifies a layout, a model, and an input source. The model is loaded into the application, the layout is constructed and displayed on the screen, and the inputs are fed to the layout for execution.

Inputs are parsed from the input file and fed to the input blocks. Each vector of inputs is passed in succession to the network, and moved through the layout. Each block takes its inputs, performs the defined algorithmic operation, and then passes the result on to the next connected block (or blocks). Execution continues until all inputs have been processed.

One particularly useful feature is the capability to view intermediate results during execution. The GUI allows the user to select individual displays for all blocks within the layout. These selections are used to show intermediate results as data passes through the network. This means that each block has the ability to display its individual result as they are computed, instead of just seeing the end result of the entire network.

Intermediate data displayed by a block depends on the context and type of data. Examples of intermediate results might include displays of numeric values, boolean values, imagery, histogram, EXIF image data, etc., selected from a list of options.

Final execution results can be written to file(s), or simply displayed to the user. The same layout can be reused as desired to process other input datasets. For example, a barrel shifter layout (implemented from the digital logic model) could be
used on multiple sets of binary logic inputs to test the shifter design.

The layout files can be created from scratch as desired, or an existing layout can be imported. A layout can be saved at any time for later examination or use.

There is also the option to step through the execution of the layout to see results for each block as the algorithm is applied. The button for taking individual steps is shown with its tooltip in Fig. 1. This functionality can be used to ensure validity and applicability of inputs to each block as incremental processing occurs. The ability to step through functions facilitates block debugging and proper execution of the network.

Each block is able to provide dynamic type checking in order to validate that the inputs are the right format and outputs are displayed in an appropriate fashion. Therefore if a block requires an image as an input then placing values other than the specified types will be rejected.

In many ways, using this system is similar to operating Labview [2] and Simulink [3], where functions are graphically manipulated and connected based on the required needs and definitions of inputs and outputs each block. The key difference is that the models in our design are fully customizable, and are not based on sets of predetermined functions. The implementation section describes how additional functionality can be added and how completely new models can be developed.

There are many cases where this application would be very useful. Some of the areas of interest include:

- An experimenter with little or no software development experience could use the building blocks to graphically construct and test a specific layout to design experiments. (Once the experiment is designed and validated, the experimenter could execute it from the application, or software personnel could use the design as a template for a custom software component.)

- In an intelligence organization, the graphically-based application may also be used to construct interesting and useful algorithmic networks for examining individual frames of multispectral imagery.

- A person could send inputs of music through a network that would allow them to manipulate the sounds such as amplifying the bass or separating the vocals from instrumentals.

- A movie or image editor could construct a layout to alter the display of the images, such as reducing size and increasing contrast; additional blocks could extract or add captioning.

- A student learning about logic gates and basic CAD operations could use this utility to construct logic circuits and test the results of their designs.

The program can be used in a multitude of CAD-like ways, and is therefore a blank slate, completely defined by the model loaded into the application. The utility provides a GUI drag-and-drop interface, but the model provides all of the functionality.

Our primary purpose is to facilitate the ease of use for multispectral visualization experimenters through a graphical display. We are also interested in saving the computer programmer's time. Many of the experimenters' tasks involve applying specific algorithms, with certain parameters assigned, in a specific order. Data is moved through the algorithms, and the results are evaluated to determine if changes should be made to the parameters, or if additional filters and other algorithms should be added to the network. This type of scenario would require a programmer to re-code the application for each different parameter or network connection.

Instead, the utility described in this paper allows an experimenter to autonomously interact with the program. Multiple inputs are fed through various blocks in order to visibly see the results, and parameters can be changed at will, with minimal programmer intervention.

We are very interested in supporting experiments with 3D visualization and inputs from multiple viewpoints. Another key area we are interested in is the production of high-level image computations and the creation of appropriate display visualizations for blocks. We already have a collection of image processing and visualization algorithms, and we'd like to integrate these seamlessly into the utility.

Fig. 2 shows the basic usage of the application in order to edit a single image. The first image displayed on the left in the figure is the starting image. The input is connected to the decreasing arrow which is a symbol for downsizing the picture. The result of this operation is shown below the block itself; a smaller image is produced. The last block's functional is to add noise and shift the picture, represented by an icon with a mass of colors. The final result from beginning to end is a smaller, shifted picture with noise added. This is just a small example of functions that can be utilized to edit image features. As long as the code can access the necessary features of the picture, the image can be changed and altered to the meet specifications. The intermediate results displayed in this example are the images themselves, but the system is capable of displaying anything the user desires that is meaningful, since the intermediate display visualizations are also user-selectable.
III. IMPLEMENTATION

The entire application (including block blocks and display functions) is implemented as a collection of functions written in the cross-platform Python programming language [4]. Python is a byte-code-compiled, object-oriented scripting language with a robust set of development features and libraries. Our application is compatible with any other operating system supporting Python and the GTK graphics libraries [5] such as Microsoft Windows and Linux. If needed, algorithms written in other computing languages can be directly integrated into the application using the ctypes library and mechanisms available through Python.

GTK is used as the main GUI library for creating windows, buttons, and dialog boxes throughout the program. GTK is also useful because it houses a generic drawing area, allowing the Cairo interfaces to be used to create custom graphics. While GTK is good for basic template window creation, the Cairo 2D graphics library specializes in supporting custom drawing requirements, such as drawing done in this application. Cairo is a low level drawing tool, which means that items are drawn on the screen based on a set of coordinates. It can be programmed to draw nearly anything, including entire pictures and text [6].

The models consist of a set of files: a descriptive XML file and a collection of (generally Python) implementation code. All functional blocks are listed in the model's XML file. The model's XML file defines all blocks for the model, including each block's property information (name, number and type of input and output, drawing instructions, and executable function name). The content of this file determines the list of blocks available to the user for layout construction.

Drawing instructions for the blocks are very simple, since the screen representation of the block is not intended to be a complex part of the system design. The drawing instructions can consist of an ordered collection of points, which are passed to Cairo to draw as line segments. Alternatively, the drawing instruction section can identify a graphic to be loaded and displayed, such as a .gif or .jpg file, along with size information used to scale the graphic for display.

The model's implementation file only contains the code performing the implementation for the corresponding block's function identified in the model's XML file. Our models currently only include pure Python implementations, but the Python ctypes library allows the program to directly import compiled shared libraries from other computer languages. We anticipate adding implementations from our vast precompiled C and C++ libraries of multispectral visualization routines soon.

The way we've designed the model files clearly encourages model extension, and new blocks can be added by simply inserting a segment of XML into the model's XML file and generating appropriate implementation code into the model's Python code file. The source code for new blocks can be copied from existing code, imported from binary libraries, or created from scratch.

Layouts are stored as separate XML files, and contain descriptions of user-designed configurations of model blocks and their connectivity. The user also typically generates an input file containing vectors of inputs to send through the layout. For example, the file may identify sets of binary values for use with a digital logic layout, or it may simply contain the names of directories of imagery to process with a multispectral layout. The format of the input file is user-defined.

When the user initiates the program with a model, a layout, and an input dataset, the software is able to execute the layout. The program statically displays the layout, and selecting the "run" buttons executes the layout for all inputs. Alternatively, selecting the "step" button reads a single input and process the input through exactly one link of the layout. Additional processing is performed each time "step" is selected, with a new input vector read and processed once the current vector has completed.

During execution, values from the input file are fed to the input nodes, then passed along the connections to successive blocks. Each block executes its function on the input, and passes its result to the next connected block(s). The layout is traversed in this manner until the output nodes are reached, and processing begins again with the next input vector. This procedure continues until all inputs have been computed through the layout.

Each block is capable of displaying the value it computes, where all values except for the final output are considered to be intermediate values. The way intermediate values are displayed is user-selectable, and may include displaying imagery, histograms, statistics, or other options.

The current execution function is a recursive procedure (which does not allow cyclic connections in the layout), but a new execution function is already being developed to correctly compute values for cyclic configurations.

The software design allows the tool to import a variety of models without loss of generality. As a proof of concept, the original models were developed as simple binary logic design operations (AND and OR gates) and were used to demonstrate
logical operators on input files containing lists of binary vectors. Within a single day, a small set of block functions and models were developed for basic manipulation of JPEG image files, demonstrating the versatility of this application for use in multispectral image analysis. Additional operations are being added to both models (the digital logic model facilitates fast and easy testing).

As described in the previous section, the development of additional model code further increases the base of available building blocks to apply to any network. The application’s ability to easily integrate blocks coded in other languages encourages the block library to be increased quickly and efficiently. Display options can also be easily expanded as desired, in order to show intermediate results in a different ways. Display functions are not model-specific, and are available to all models once they are added.

IV. EXAMPLES

Fig. 3 is a very basic example of a logic gate design, developed as a proof of concept of the flow of the application, and is easily expanded to larger layouts as needed. This specific layout has three inputs, two entering the AND gate and one directly connected to the OR gate. The AND gate sends its output to the OR gate to use as an input. In this configuration, the final result is the block that does not have an output and therefore the end of the network. The specific display options shown are very basic drawings of 1’s and 0’s for results (directly to the right of each block), but could be just as easily be represented as a meter displaying full and empty or a display that shows red and green stoplights.

Fig. 4 is a simple layout example of image fusion. The two different images of the same scene were obtained online from OCTEC Intelligent Platforms (now part of GE). The first source image is of a LWIR (Long Wave Infrared Radar) view of a small gathering of houses. The second is what is visible to the human eye. Both of these images provide useful details about the environment. The visible image provides small details of the background and the houses in general. The LWIR is able to see past the smoke (which limits the human eye) and can see the people running between the houses. However, the LWIR image loses details such as color and fine lines. The result of fusing these images allows for the best of both worlds: the view of the people running behind the smoke screen as well as the detail and color that is normally seen in the surrounding area. Again, this display is only showing the images themselves as results but could display just the human-like figures in the picture, edge detection of all the houses, or any other selected display function, on a block-by-block basis.

![Figure 3: Logic gates, results drawn to the right](image1.png)

![Figure 4: Layout combining visual and LWIR images (humans boxed)](image2.png)
It would also be possible to incorporate the fusion of SWIR, MWIR, and LWIR, covering several different infrared waveforms, to allow an analyst to see differing views of the environment. This can then be easily changed to view different combinations and outcomes by graphically modifying the layout, not source code. Results would be available instantly.

Several examples and uses have been mentioned, note that there are more functional options that blocks could perform. Blocks are not limited to single inputs (such as binary values or individual JPEG files), but can also be programmed to accept multiple data inputs. For example, a block that accepts three frames of multispectral imagery simultaneously might implement a data fusion algorithm, or a block that takes two JPEG image frames as inputs might output the frame with a higher information content. As another example, consider a block that takes multiple images of the same scene (from different viewpoints) as inputs and generates a 3D model as an output.

Specific blocks could also be designed to operate on data streams. For example, a block might accept a stream of UAV imagery and perform video stabilization operations, outputting the stabilized stream. Another block might simply read an image stream and output individual graphics frames, or input a stream of audio and output the stream with noise reduction applied. Blocks could even operate on computer files, such as performing system administrative functions on user directories, or doing data mining operations on textual data. This utility is obviously a versatile analysis tool.

V. CONCLUSIONS AND FUTURE WORK

This application is a very broad and open ended project, which means most of the functionality has to be very dynamic, and still be user friendly. Some of the areas of the application are not up to their full potential and are being significantly increased. This is especially true with respect to the usability of the program. Some of the additional items being addressed are increasing execution speed, enhancing user-friendliness (more GUI functionality), adding more options to initial libraries, and adding convenience features for the user.

Display functions are currently limited to very basic image and logic results, but additional display functions (with greater flexibility) are being added quickly. For example, display options are needed to show results for intermediate data that is not imagery or simple values. The display list will continuously grow, so these options will be numerous and cover a broad spectrum of display needs.

Along with adding more displays, it would be good to have the ability to detect what types of inputs are being fed through the blocks in order to either accept or deny the signal. It would also be useful to make the list of available display functions context-sensitive, selectable based on the output a block is generating.

GUI functionality is somewhat limited now, and could be vastly expanded to allow users to customize the blocks with menus providing options for changing the default models. Many details are not editable in the current menu system; it would be easier for non-programmers to edit blocks from menus instead of making changes directly to the XML files.

Adding tooltips to display not only text, but images and information based on what block or item the user is currently hovering over is a worthwhile feature. Along with tooltips, full expansion of keyboard shortcuts and commands would allow a user to access functionality faster, increasing productivity while using the program.

Once a layout is created it can be reloaded into the system from a saved XML file, but the entire design is brought in with each individual block. Adding an object-oriented approach would allow the user to import a layout as a single executable block. This promotes reusability, and makes it easier to manage large scale layouts.

The assumption is that the user is not always tech savvy, so there is a graphical interface to all of this functionality. However, if a user becomes more familiar with the program and knows what it should output with given inputs, then the entire application should be able to be executed in batch mode, performing the functionality without the display. This operating mode would reduce overhead.

Finally, we are in the process of taking advantage of parallelism by dividing processes across available CPUs. Additional testing will be required to determine when multiprocess is advantageous.

We are enthusiastic about the application of this utility within our organization. We believe that the experimenters will find tremendous value with their ability to generate studies graphically, instantly, and without relying on software support personnel for all aspects of their design. The model library is also an outstanding place to capture our multispectral visualization and fusion algorithms.

VI. REFERENCES