On Encouraging Coupled Views for Visualization Exploration

Jonathan C. Roberts

University of Kent at Canterbury, Computing Laboratory, Canterbury, England, UK.

ABSTRACT

Scientific visualization, especially visualization exploration, enables information to be investigated and better understood. Exploration enables hands-on experimentation with the displayed visualizations and the underlying data. Most exploration techniques, by their nature, generate multiple realizations and many data instances.

Thus, to best understand the information in coincident views, the manipulation information within one view may be 'directed' to other related views. These multiple views may be described as being closely coupled.

Within this paper we advocate the use of coupled views for scientific visualization exploration. We describe, some key concepts of coupled views for visualization exploration and present how to encourage their use. The key concepts include: the scope of the correlation (between two specific views or many realizations), who initiates the correlation (whether the user or the system) and issues about 'what is correlated' (objects within a view, or the whole viewport).

Keywords: Coupled views, multiple views, linking, multiform, visualization, visualization system

1. INTRODUCTION AND MOTIVATION

Data visualization may be achieved to present a result, explain a phenomenon and to discover new insight through exploration techniques. Indeed, visualization exploration requires highly interactive visualizations that may be manipulated, dissected and interrogated to explore and investigate the underlying information. Thus the user is *looking* for interesting or unusual anomalies and features within the realizations. The investigator starts to ask questions of 'what is that' and 'why is that happening', and they zoom and focus on *interesting features*.

Such investigation techniques may provide multiple *realizations* of the same dataset. These multiple views may be generated from different selections (filterings) and investigations on the dataset, or by depicting the same information in different visualization methods (multiform). Within a visualization exploration environment it is prudent to have available the views (or the values that created the views) as an experiment history, so a user may backtrack and replay previous experiment results. The use, generation and management of such multiple views should be actively encouraged.[1]

Other than this multiple-view technique the new experimental results may *replace* current realizations or *overlay* onto the previous realization. Many visualization systems, especially the module-builders,[2–6] provide the replace and overlay method. Where realizations are generated by loading the datasets, selecting parts that are of interest and displaying and manipulating the information. Further experiments may be generated by changing parameter values on any of the modules, an operation that *automatically* updates the data and visualization components in the 'down stream' modules to *replace* the current visualization with the new realization. Likewise, multiple objects may be *overlayed* and added to the current display, created from hybrid renderers.

However, when the *replacement* strategy is used information about previous experimentations is lost. Some visualization systems overcome the transient nature of the display by storing past visualization commands (as data or variable values) in a database, see GRASPARC[7] and Tioga,[8] for example. In the case of GRASPARC, or HyperScribe[9] as implemented as a module in IRIS Explorer, the user can 'roll back' to a predefined state and re-visualize the data with the 'old' parameters.

Thus, exploration techniques necessitate intuitive and powerful manipulation techniques, with the ability to realize multiple results and representations of the experiment process. Moreover, to best understand the information in coincident views, the manipulation information within one view may be 'directed' to other related views. These multiple views may be described as being closely coupled.

E-mail: j.c.roberts@ukc.ac.uk http://www.cs.ukc.ac.uk/people/staff/jcr/

Indeed, over the years, especially within information visualization, such 'focus and linking' methods have been widely advocated.

These 'focus and linking' methods allow information in one view to be selected, and thus highlighted, with the corresponding action being applied to other related views[10]; this is often utilized for multivariate visualization. These operations may be described as a graphical spreadsheet, with multiple views representing the many cells.[11,12]

We reiterate that *Multiple views* are important. First, to give a better understanding and representation of the experiment process. Second, to allow multiple forms of the same data to be presented, that enhances the users appreciation and understanding of the information.[1,13] Moreover, we advocate that the multiple views should be closely coupled together, enabling and enhancing complex manipulation and interrogation to be easily achieved.

Within this paper we:

1. categorize "what is being correlated" based on the dataflow model, section 3;
2. present what we believe to be, some key concepts of coupled views for visualization exploration, section 4. The key concepts include: the scope of the correlation (between two specific views or many realizations), who initiates the correlation (whether the user or the system) and issues about 'what is correlated' (objects within a view, or the whole viewport).
3. then explain how to encourage their use and how they are being implemented into the Waltz visualization system,[13] section 5.

2. AN ANALOGY

In essence, the coupling of views may be described as 'information sharing'. For example, to coincidentally rotate two three-dimensional objects, information regarding the viewport is shared, such that, both views use the same transformation information and the views are similarly manipulated. Therefore, this operation is analogous to program variables; if two expressions use the same variable 'a', then when 'a' is changed a different result would be generated by both expressions, when they were next evaluated.

Thus, like the variables, the links may have an equivalent type, scope, lifetime, and rules for creation and initialization. However, unlike the variables, the linkages tend to actively share the information, as the objects attached to the shared information, are actively updated when the shared information is altered.

2.1. Model View Controller

The above analogy may be similarly explained using the 'Model View Controller' (MVC) *pattern*. This pattern originated from Smalltalk architecture and is described in many books (for example[14]). This pattern (see Figure 1) describes a triad of objects, where the Controller (an interface object) is the input mechanism that allows the user to modify the Model. Once the model has been modified it notifies (attached) views that a change has occurred. Each attached View then displays the updated information. In theory, the interactions between objects are simplified and the *roles* of each object are clear.

Multiple views of the same information, using the same Model, may be easily represented. Indeed, the whole idea of separating the code within these objects is to provide highly extensible code, where new *views* may be programmed and added to the system.

Ideally, the Controller never directly interacts with the view, neither does the View change parts of the model. Thus, as the model does not hold any detailed information about the views, multiple views of the same information may be dynamically added to the system. Then, when the Controller updates the Model the attached views are automatically updated with the same information: they may be said to be closely coupled.

When using *direct manipulation* the Controller and the View become one and the same: the View is also the Controller of the model. This then generates mutually dependent classes and a storm of 'updates' may occur; in practice the Controller needs to know about the View. Thus, like in the Java Swing GUI component kit[15] the Controller and View objects are collapsed into a single UI object (Figure 2).

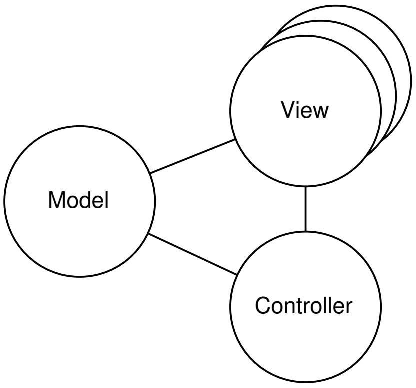

Figure 1. Diagram showing the Model View Controller pattern. The Model holds the state of the process and publishes notifications when the state changes. The View gets notifications from the Model change and displays the visualization. The Controller handles requests from external events and updates the model appropriately.

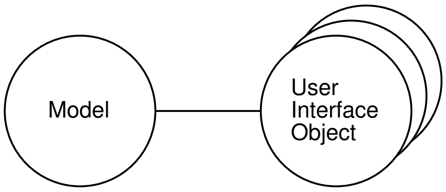

Figure 2. Figure depicting the View and Controller as one object.

3. LINKED VIEWS WITHIN VISUALIZATION

There are many reasons for requiring linked views within visualization, from coincidentally rotating multiple three-dimensional views to highlighting like elements within those views.

Within this section we categorize the linked correlation based on the dataflow model.[5,16] This dataflow model describes one method of understanding the visualization process; the data is *filtered*, to create a subset of the data, which is then *mapped* into a representation which can be *displayed*. We propose that the linkages may 'couple' different aspects of the visualization process. Figure 3 depicts the dataflow model annotated with different *data spaces* and brief descriptions of what may be linked.

To understand what is being *coupled*, using the MVC method, we need to think that each View of this data is coupled to the same Model at a particular Layer. For example, within the *Viewport Layer* the *Image* Model holds information about the orientation, scaling and projection characteristics of the Abstract Visualization Object; so this coupling enables Views to be coincidentally rotated, zoomed and moved. In essence, it is the function (the visualization process) that is being coupled, that in turn changes the Model and sends notification of the *change* to the View.

Within the visualization process, a change to a Model at a particular level may have an effect on the processes, and thus the datasets, further down the data-flow. In the module building visualization systems the down-stream modules automatically fire when the data becomes out-of-date due to a change in the visualization parameters. Thus, coupled views may generate a cascade of 'notifications' to update other dependent views and modules.

In the following sections, we describe the coupled operations at different layers of the model, using the dataflow model in Figure 3, and reference some visualization systems in which the techniques are used.

3.1. Window Coupling

Here the management and display of the end-user windows is coupled together. This is perhaps an unusual concept, however, Ben Shneiderman[17] talks about tightly coupled windows, that enable multiple windows to be jointly re-scaled, minimised and restored. Taking this further, manipulations such as positioning the windows and scrolling elements displayed within the windows could be coupled.

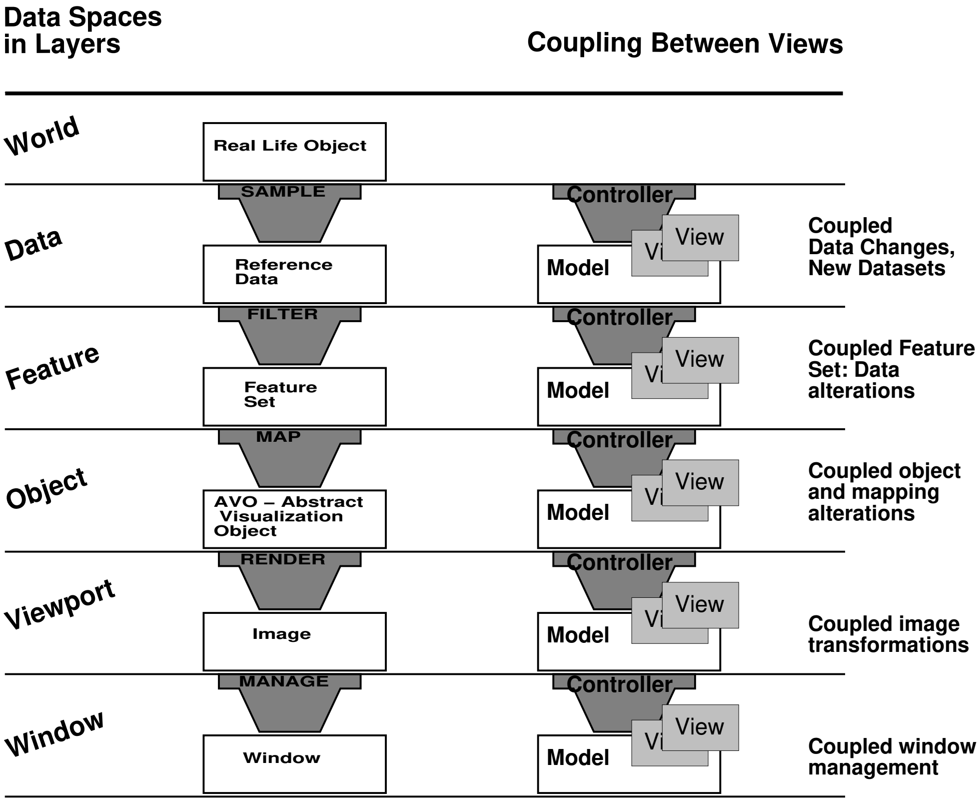

Figure 3. Schematic depicting the dataflow model. The layers in the model show the intermediate data-sets between the visualization processes. Coupled views may interact between objects within the same layers.

3.2. Viewport Coupling

This operation provides methods to coincidentally transform the viewport between multiple windows. Direct manipulation may be used to (say) jointly rotate multiple three-dimensional views of some visualization.

Many visualization systems allow this style of coupling. Indeed, within the module building environments, such as IRIS Explorer[18] the camera position and orientation may be redirected to other modules, including another *Render* module. Thus, transformation operations in one view are tightly *coupled* to other coincident views.

Separate views of the same or similar information are useful to both individual users and multiple participants. Although, the focus in this paper is the linking of multiple views for a single user, similar issues arise in collaborative visualization, with many participants, and multiple views being displayed and controlled across disparate clients, see for example, Wood et al.[19] Here, the problem becomes more of 'how to send the information to other views, without an information cascade taking place'. Solutions exist from 'sending only delta changes' to using multicasting approaches.

The identical control and manipulation of objects may further extended when higher dimensions exist. For example, in HyperSlice[20] each view represents different dimensions of space, so when one variable is moved the other grid views are updated.

3.3. Object Coupling

The individual object (the Abstract Visualization Object) within the realization may be coupled. This enables *like* mappings to be applied to different views. For example, it is often useful to use the same colourmap organisation across a series of views; this may be easily performed using systems such as IRIS Explorer.

Moreover, changing the attributes of an object to highlight a feature, may be represented in this category. For example, the Spyglass[21] suite of software tools (Spyglass Plot and Data) can work interactively to allow, for example, a point to be selected in one window and synchronously highlighted and related to points in other windows.

Other than the appearance of an object, the size of a selected object may be dynamically changed and coupled to other objects. For example, the SDM (Selective Dynamic Manipulation) system provides interactors to change the physical appearance of graphical objects; each feature set has an attached interactor that is responsible for changing the physical appearance of graphical objects in that set.

Other objects, such as pointers, annotations or meta-information, may be coupled. For example, the Visualization Input Pipeline (VIP)[22] provides a backward control path, from the image to the data, that is used to transport control information backwards up the flow pipeline. They describe an example that displays several views of the data along the pipeline with a cursor pointer in each view. The position of the cursors are linked together and the translation of one pointer moves the position of each corresponding cursor.

3.4. Feature Coupling

Feature coupling enables the same feature operation to be applied to like views.

For example, Klinker[23] describes a method using 'Data Probing' that allows the data to be graphically filtered by defining areas of interest on the screen using polygon areas. They use a 'Cursor Linking' technique that allows the probed information to be linked between modules (and views). They describe an example that creates two displays of the data, the first window displaying a three-dimensional rendering of the data and the second window a three-dimensional Red-Green-Blue colour scatter plot of a two dimensional selected region from the first window; as the region of interest is altered so the scatter plot alters.

This 'Data Probing' fits in with the MVC representation, where the View and Controller are separate objects. Often within visualization this is called the 'Display Filter, coined by Schroeder, Lorensen, Montanaro and Volpe.[24] This technique allows the output from one display to be the input to another display. Schroeder et al describe one example that outputs streamlines within a fluid flow data set, the Display Filter of this object allowing spheres created with a connected module, (scaled and coloured according to input scalar magnitude and vector value) to be mapped along the length of each streamer.

The XmdvTool[25,26] generates multiple representations of multivariate data and provides a Brushing Technique,[27] so that when one element is *brushed* in one view the corresponding elements in other views are also brushed. Thus, the brushing could be setup to generate a specified feature-set that is displayed in correlated views.

3.5. Data Coupling

It is possible and relevant to couple at the data level. These operations are similar to those at the 'Feature Set' level, but apply to operations on the reference-data. For example, a data archive may consist of multiple datasets of the same or different data objects. It would be useful to apply the same filters and sub-sampling operations to each.

4. KEY COUPLING CONCEPTS

Close correlation, through coupling, allows multiple views to be linked together in such a way that any manipulation and change of values in one view creates a similar change in the linked view (vice versa).[23,20,25] In theory, any view could be linked to any other, but in practice the explorer would wish to correlate only certain views and operations.

Moreover, it may be, especially in visualization exploration, that some links do not make sense and in fact may confuse the user. If view 1 is a histogram and view 2 is a three dimensional view of multiple isosurfaces, then 'Linked Highlight' – that automatically highlights corresponding data elements between linked views – may be valid but 'Linked Rotation' would be meaningless.

The process of distinguishing the valid and invalid connection types is not simple: the dimension of the output can be used to a certain extent but this is not a generic decision parameter because, for example, it could be useful to correlate the cursor between a three dimensional view of an object and a two dimensional view of an X-Ray rendering of the same view. So as the three dimensional image is rotated so the resultant X-Ray image would change; however, the opposite three-dimensional control would not be valid!

Within the MVC model it is the responsibility of the View object to *use* the useful information, and to ignore the other *irrelevant* information.

We now describe some 'key coupling concepts', these concepts are based on the analogy in Section 2.

4.1. Type

The 'type' of the link expresses the method by which the views are linked, e.g. selection, highlight and the transformation operations, such as rotation, translation and scale. From a low level point-of-view each view-coupling is a manipulation of numbers and data-structures. From a more abstract level the 'type' is represented and determined by operations at the different Layers of the visualization paradigm, as expressed above in Section 3.

4.2. Scope

The 'scope' of control defines which objects or views are coupled; the linked communication enacts upon the views 'in scope'. This is similar to the variable scoping, such as local or global variables. For example, the linked communication may couple specific objects between:

1. any user specified view; described as a local variable on a set of views;
2. views from the same data Model; described as a local variable on a set of views, where the set is determined by the data origin, e.g. a feature-set.
3. all the views of a particular reference-dataset;
4. every current view of this session; described as a global variable.

Thus, one way to determine the scope of each coupling is to scope the views within different sub-parts of the exploration process itself. For example, coupling could occur between views of the same Feature-set, or derived from the same reference-dataset. This simplifies the above stated problem of valid/invalid correlations, by guaranteeing that only views from the same derived-dataset are coupled. However, it may be possible (and relevant) to couple views from different data-origins; in this case, unless a correlation between data-models is given, only the Viewport and Window coupling Layers may be relevant.

4.3. Lifetime

Each linked manipulation has a 'lifetime'. For example, two views may always be linked together whatever is being displayed, alternatively, the objects in the views may be linked for a specific operation at a given time. Incidentally, like for program variables, lifetime and scope are inherently related.

4.4. Initialization

The links may be created and 'initialized' by a variety of methods. For example, the user may graphically point and define which variables and objects are linked, as in the LinkWinds[11] system; by drag-and-dropping viewport icons into controller windows, as in SciAn[28]; by naming specific viewers for particular operations, as in the spreadsheet system of Levoy[12]; or using the experimentation process itself to determine the coupling, as in the Waltz visualization system[13]).

In the spreadsheet system of Levoy[12] individual cells may be named as viewers for particular operations. The LinkWinds[11] system enables specific (user defined) views to be joined together. SciAn[28] enables viewports to be linked by dropping viewport icons into the controller windows.

4.5. Updating

Finally, each of the views attached to the model being controlled should be *instantaneously* updated when the Model has been changed. However, especially in visualization some views require extensive processing, thus, a new thread should be spawned to do the work. In Levoy's spreadsheet system[12] the out-of-date views are shaded-out whilst they are updating.

Moreover, other views and data-Models may depend on the action from some coupled-views; there may be multiple dependencies within the visualization pipeline. There are several methods to update these dependencies, including:

1. *User initiated* provides the user with full control, to precisely initiate when the data is passed from one module to the next. If the user changes the values at a module then the user is required to re-export the data;

2. *Semi-Automatic* operates like the User initiated, but provides a routine to update the path. Occurring locally (from the current selected module down stream) or globally (updating every module in the session);

3. *Global Transfer* acts like the data flow method: when the data has been specialized it automatically flows to the following module and updates the display and that can automatically flow to the next module, see[18];

4. *Demand (or Output) Driven* requests the data through the network of modules as the user controls a display, see.[5]

5. METHODS THAT 'ENCOURAGE'

We believe that the visualization system itself should encourage the user to perform 'coupling' of related views. Thus, to motivate their use the system should provide appropriate support and manage their use. We use a classroom as a metaphor, to describe some 'methods that encourage'.

In the classroom the pupil is exploring knowledge and understanding: they are being encouraged to learn and solve problems. The teacher in the classroom may 'encourage' this process using many methods, including: breaking the problem into individual tasks - even to automate or remove some of the tasks, provide a sense of consistency and comfort, force people to do the tasks and group people to learn and work together.

Similarly, the coupling concept may be encouraged by:

1. Managing the exploration – by dividing the visualization exploration process into smaller tasks and enabling views to automatically generate appropriate views of the Model when, for example, the views are first attached; and

2. Managing the correlation – by, automating parts of the linkage operation, providing methods that keep each views consistent and up-to-date, forcing the user to follow a visualization model and grouping operations together.

5.1. The Waltz Visualization system

The methods described in this paper are being further integrated within the Waltz visualization system. The Waltz visualization system[13] is designed to aid the user to explore and generate a correct interpretation of the underlying information. The three parts of the system enable the user to, first, display the data in different forms – *Multiform* visualization. Second, display the information in simplified *Abstract* views. Third, provide a high degree of interaction and *Direct Manipulation*. Waltz enables the direct manipulation to be coupled between related views.

5.1.1. Managing the Exploration

Waltz implements a subset exploration hierarchy, Figure 4. This strategy allows the data to be initially viewed as one whole dataset, and the user refines the data selections on this set to *specialize* the information into specific sets of interesting features. Each new subset may be left in the exploration hierarchy – generating a history of previous experimentations. The root node of the hierarchy contains the *reference data*, subsequent nodes (named render-groups) contain user-specified sub-sets of the information.

Figure 4 shows that at each subset the user may attach multiple View methods. On attaching a new view, an update is sent from the Module-Controller to attach a new view and automatically display a realization of the information. Each view has a style and the representations are displayed determined by the style; volume rendering, isosurface and statistics, for example. Figure 5 shows the result from a Waltz exploration. A new view may be added to the render-group C, for example, to automatically display another depiction of that same feature-set; the window automatically generates a default rendering of the data.

Waltz implements the Semi-Automatic update strategy, see section 4.5. Thus, when a particular subset (in the exploration hierarchy) is altered, the coupled views of this display-group are automatically updated. Child subsets may be updated by initiating a request, however, the user has an option to generate a new subset of the data and not replace the old views. If the child Model is replaced with the new data, then the attached views are automatically updated. This enables the user to control the exploration hierarchy, and encourages the user to experiment. For example, if in Figure 5 the data-filtering from module A is altered then the view in A is automatically updated and the user has the option to generate a new render-group or force B and C to update with the new information.

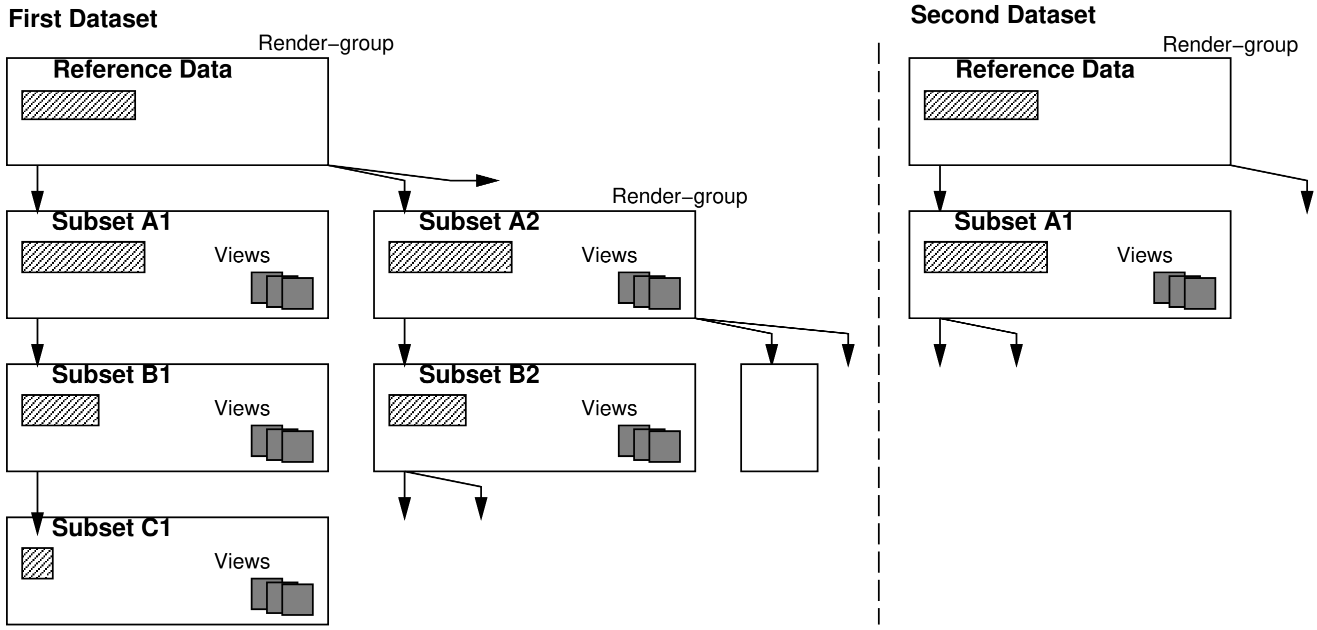

Figure 4. Schematic depicting the Waltz exploration hierarchy, where the data is focussed and specialized by selecting feature-sets of interest.

5.1.2. Managing the Correlation

The Exploration strategy is an important factor for determining what is coupled. The Waltz exploration hierarchy provides a convenient method to *scope* and *type* the coupled manipulation.

To encourage the coupled views, Waltz automatically correlates each view in the render-group. This allows views, of the same feature-set, to be coupled together, for example, in Figure 5 each of the elements at C are automatically correlated. Indeed the example shows that the three-dimensional views have been jointly rotated.

Any or all of the correlations may be turned off (or on) with the views being automatically updated. In Figure 5 we see that the appearance of the objects solely within view C2 have been altered.

Thus, these render-groups provide an elegant method to 'scope' the correlation at the Window, Viewport, Object and Feature Coupling layers. Data-coupling is only appropriate at the root node of this exploration hierarchy, but is not implemented in the current version of Waltz.

The current Waltz version supports coupling at the

- Window layer, by labelling each window with a unique identifier, that relates each window to the experiment canvas; and by automatically organising the layout of the modules on the experiment-canvas to represent the data-subset hierarchy;
- Viewport layer, with transformation and direct manipulations);
- Object layer, by highlighting and enabling appearance changes to be coupled between objects in multiple views;
- Feature layers, by coupling feature-set changes.

Waltz has been designed in C++ with classes representing the data, subsets of the data and Views. The system is implemented over X and Xlib using the Motif widget set, Inventor[29] and OpenGL[30] libraries.

The correlation is implemented in an MVC method. Where a Views are attached to a display-group method, that also holds a copy of the data-model, and the Controller (also part of the View) notifies the display-group of the 'type' of correlation and which View initiated the Control information. In Object-Oriented terms this is more tightly coupled than a *pure* MVC method. However, this allows such notifications of change to only update appropriate methods of Views that are (1) in *scope*, (2) not the initiating the control information and (3) of the correct *type*.

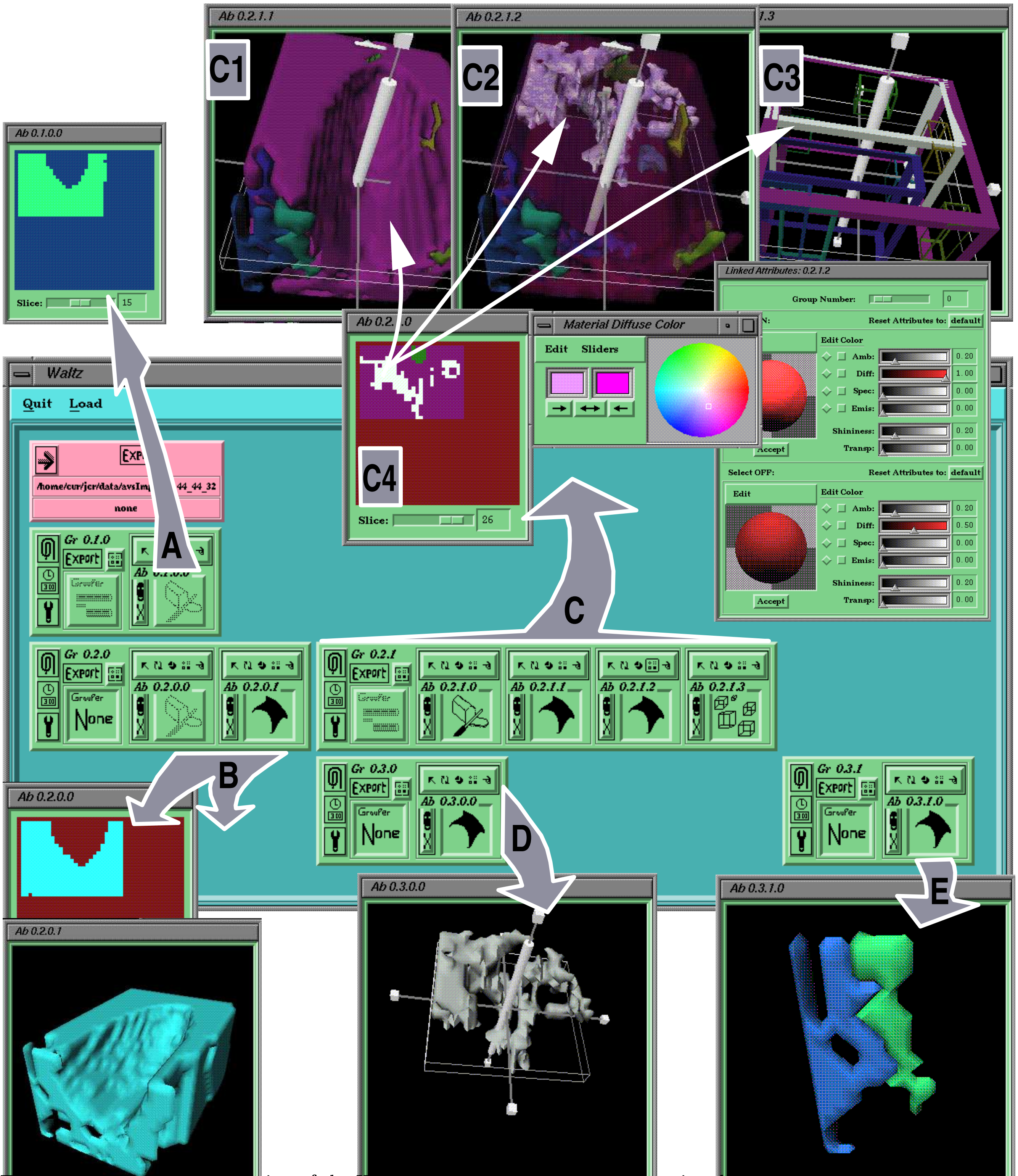

Figure 5: Images from a session of the Waltz visualization system. The session depicts a Space Impact Simulation. **A** depicts the slice viewer; **B** a surface representation; **C** coupled investigations, C1-3 showing Viewport coupling and attached jack manipulator, C4 shows Object Coupling of an object being selected and likewise highlighted in views C1-3, C2's outer object has been un-coupled and made to be semi-transparent. **D** is a subset of the highlighted object in C. **E** is another subset of a different selection on feature-set C.

6. DISCUSSION

By providing the visualization exploration hierarchy, the user is forced to follow a strict pattern of control. This enables and encourages the user to explore the information and easily generate multiple representations of their data. Direct manipulation is important to visualization exploration, and if the user is to be encouraged to generate multiple views the system needs to manage the views and the coupling of information between views. Thus, automatically coupling views together, determined by the exploration hierarchy, is an important and powerful method.

However, the coupling of views, determined by the exploration method, restricts which views may be coupled, and any-to-any schemes may be useful. Such a strategy could be implemented in Waltz, but would generate multiple inter-dependencies within the exploration hierarchy.

Also, the any-to-any schemes couple between views of different data Models, hence, it is easy to couple at the Viewport and Window levels and more difficult to *generally* correlate data-objects at the Object and Feature levels. In general, the more diverse the Models become the less the coupling makes sense; and the views *ignore* more control information.

7. SUMMARY

We have presented many issues regarding the coupling of views, categorizing the ideas using the dataflow model. This forms a convenient method to explain 'what information is being coupled'.

We have presented some 'key coupling concepts' based on an analogy of data variables. Thus, the coupling has a type, a scope, a lifetime, an initializing method, and methods to update dependent parts of the visualization process.

We stated that 'to encourage close coupling' the visualization system must provide methods that enable the operations to be easily achieved, such as automating the initiation of the coupling depending on the exploration hierarchy; and have shown that they may be integrated within a visualization system such as Waltz.

REFERENCES

1. J. C. Roberts, "On Encouraging Multiple Views for Visualization," in *Information Visualization IV'98*, E. Banissi, F. Khosrowshahi, and M. Sarfraz, eds., pp. 8–14, IEEE Computer Society, 29-31 July 1998.
2. G. Abram and L. Treinish, "An extended data-flow architecture for data analysis and visualization," in *Proceedings Visualization '95 – sponsored by the IEEE Computer Society*, pp. 263–270, 1995.
3. D. S. Dyer, "A dataflow toolkit for visualization," *IEEE Computer Graphics and Applications* **10**(4), pp. 60–69, 1990.
4. J. Walton, "Data Visualisation with IRIS Explorer - What's New?," Tech. Rep. TR10/96 (NP3070), Numerical Algorithms Group Ltd., 1996. (http://www.nag.co.uk/doc/TechRep/NP1513.html).
5. C. Upson, T. Faulhaber, D. Kamins, D. Schlegel, D. Laidlaw, F. Vroom, R. Gurwitz, and A. van Dam, "The application visualization system: A computational environment for scientific visualization," *IEEE Computer Graphics and Applications* **9**(4), pp. 30–42, 1989.
6. M. Young, D. Argiro, and S. Kubica, "Cantata: Visual programming environment for the Khoros system," *Computer Graphics* **29**, pp. 22–24, May 1995.
7. K. Brodlie, A. Poon, H. Wright, L. Brankin, G. Banecki, and A. Gay, "GRASPARC – a problem solving environment integrating computation and visualization," in *Proceedings Visualization '93 – sponsored by the IEEE Computer Society*, pp. 102–109, 1993.
8. M. Stonebraker, J. Chen, N. Nathan, C. Paxon, A. Su, and J. Wu, "Tioga: A database-oriented visualization tool," in *Proceedings Visualization '93 – sponsored by the IEEE Computer Society*, pp. 86–93, 1993.
9. H. Wright, "The HyperScribe Data Management Facility," Tech. Rep. TR1 96, NAG, 1996. http://www.nag.co.uk/doc/TechRep/PS/tr1_96.ps.
10. A. Buja, J. A. McDonald, J. Michalak, and W. Stuetzle, "Interactive data visualization using focusing and linking," in *Proceedings Visualization '91 – sponsored by the IEEE Computer Society*, pp. 156–163, 1991.
11. A. S. Jacobson, A. L. Berkin, and M. N. Orton, "LinkWinds: Interactive scientific data analysis and visualization," *Communications of the ACM* **37**, pp. 43–52, April 1994.
12. *Spreadsheets for images*, ACM SIGGRAPH, July 1994.

13. J. C. Roberts, "Waltz: an exploratory visualization tool for volume data using multiform abstract displays," *Visual Data Exploration and Analysis V – Proceedings of SPIE Vol. 3298* , January 1998.
14. M. Cohn, B. Morgan, M. Morrison, and M. T. Nygard, *Java Developer's Reference*, Sams, Net, October 1996.
15. M. Andrews, "Getting started with Swing." http://java.sun.com/products/jfc/tsc/getting_started/getting_started.html, 1998.
16. R. B. Haber and D. A. McNabb, "Visualization idioms: A conceptual model for scientific visualization systems," in *Visualization in Scientific Computing*, B. Shriver, G. M. Nielson, and L. J. Rosenblum, eds., pp. 74–93, IEEE Computer Society Press, 1990.
17. B. Shneiderman, "Advanced graphic user interfaces: elastic and tightly coupled windows," *ACM Computing Surveys* **28**, pp. 144–144, Dec. 1996.
18. Silicon Graphics Computer Systems – Silicon Graphics Inc., *IRIS Explorer Technical Overview*, April 1992. (http:// www.nag.co.uk/visual/IE/iecbb/docs/Technical_Report.ps).
19. J. D. Wood, H. Wright, and K. W. Brodlie, "Collaborative visualization," in *IEEE Visualization '97*, R. Yagel and H. Hagen, eds., pp. 253–260, IEEE, Nov. 1997.
20. J. J. van Wijk and R. van Liere, "HyperSlice – visualization of scalar functions of many variables," in *Proceedings Visualization '93 – sponsored by the IEEE Computer Society*, pp. 119–125, 1993.
21. F. O'Neill and P. Goosens, "Bringing data to life," in *Technical Computing*, C. Rand, ed., pp. 20–21, Published by Adept Scientific, Summer 1995.
22. W. Felger and F. Schröder, "The visualization input pipeline – enabling semantic interaction in scientific visualization," in *Eurographics '92 (Computer Graphics Forum Volume 11 No. 3) – Alistair Kilgour and Lars Kjelldahl Eds.*, pp. 139–151, Blackwell Publishers, 1992.
23. G. J. Klinker, "An environment for telecollaborative data exploration," in *Proceedings Visualization '93 – sponsored by the IEEE Computer Society*, pp. 110–117, 1993.
24. W. J. Schroeder, W. E. Lorensen, G. D. Montanaro, and C. R. Volpe, "VISAGE: an object-oriented scientific visualization system," in *Proceedings Visualization '92 – sponsored by the IEEE Computer Society*, pp. 219–226, 1992.
25. M. O. Ward, "XmdvTool: Integrating multiple methods for visualizing multivariate data," in *Proceedings Visualization '94 – sponsored by the IEEE Computer Society*, pp. 326–333, 1994.
26. A. R. Martin and M. O. Ward, "High dimensional brushing for interactive exploration of multivariate data," in *Proceedings Visualization '95 – sponsored by the IEEE Computer Society*, pp. 271–278, 1995.
27. R. A. Becker, W. S. Cleveland, and A. R. Wilks, "Dynamic graphics for data analysis," *Statistical Science* **2**(4), pp. 355–395, 1987.
28. E. Pepke and J. Lyons, *SciAn: User's Manual.* Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida, 1993. (http://www.scri.fsu.edu/~lyons/scian).
29. J. Wernecke, *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, Release 2*, Addison-Wesley, 1994.
30. J. Neider, T. Davis, and M. Woo, *OpenGL Programming Guide – The Official Guide to Learning OpenGL, Release 1*, Addison-Wesley, 1994.