**CLAS NOTE 95-023**

Nov 2, 1995

# Hv: A Graphical User Interface Library for Scientific and Engineering Applications

D. P. Heddle

*Christopher Newport University and the Continuous Electron Beam Accelerator Facility, Newport News, VA 23606*

Electronic mail: heddle@cebaf.gov

## Abstract

We have developed Hv, a library based on Motif and X-Windows for use in scientific and engineering applications. Through more than 500 public functions, Hv provides for multiple "views" with independent world and pixel coordinate systems. Hv will perform all necessary coordinate transformations and window maintenance and will automatically provide Hv applications with the ability to print their views to postscript printers or to save their contents as encapsulated postscript files for inclusion into other documents. Applications may draw upon a rich suite of predefined graphical items or easily add their own. Examples of applications written in Hv include the single event-display for the CEBAF Large Acceptance Spectrometer (CLAS), the graphical level-one "trigger" programming software for experiments using CLAS, a scientific plotter, and a simulation of theater missile defense architectures.

## I Introduction

Hv is a library that greatly simplifies the development of applications with a sophisticated graphical user interface. It is layered on top of X, Xt (R4 or later), and OSF Motif (version 1.1or later); however, the developer is completely insulated from those libraries. Applications can be developed solely through calls to the Hv library. Hv is *not* an interface builder for creating Motif applications; it is a library whose use results in applications with a unique appearance and feature set. While Hv is in part a tool for simplifying Motif and X, it has extended capabilities not available in either package. In particular, Hv is useful for scientific and engineering applications including modeling and simulation.

Hv is also to be distinguished from data visualization packages, such as the AVS and Explorer systems. Hv is used, for the most part, to display and interact with representations of physical objects rather than complex data sets, although visualization at the level of scientific plotting (usually of the *results* of a simulation) is an integrated Hv feature. This will become clear as we discuss example Hv applications.

As a simple but important example of why Hv is particularly useful for scientific and engineering applications, we note that Hv maintains a floating-point *world* coordinate system as well as the pixel based *local* system used by X and Motif. The world system is sized to accommodate the dimensions of what is actually being drawn. For example, a display of detectors in a nuclear physics laboratory might require a world system that is several meters in extent. With Hv, there is no need to artificially pre-scale physical objects to match a screen-based pixel system. The graphical items carry along the actual dimensions and positions of the objects they represent.

Another example of an Hv feature that is especially useful for technical applications is *pointer-tracking* (also called *feedback*). As the user moves the pointer over graphical items, the application will display information about the item. Returning to the detector example, when the user points at a detector component, the application will display in real-time the ID and location of the component, as well as any data presently available that pertain to the item. Hv automates this as much as possible for the developer by continuously reporting both which item is being pointed at and the pointer's present pixel *and world* coordinates.

Hv also provides a mechanism for *registering* multiple simulations. One application can have many simulations running simultaneously, each being updated at a different (user controllable) rate. By registering a simulation, the developer is guaranteed that a private simulation process will be called at programmable time intervals. For example, the demo application *hvmap* included with the Hv distribution displays a rotating globe. The rotation is handled by registering a simulation, called once per second, that recalculates the map configurations based on the earth's rotation.

Hv requires

- the Motif library (not just the Motif window manager, which in fact is not required), version 1.1 or higher.
- a color monitor that supports 256 colors or gray scales. Hv takes full advantage of color, and we lack the manpower to insure that it does something reasonable on monochrome monitors.

Hv has been tested on a variety of UNIX platforms including Dec (Ultrix and OSF), Sun, SGI, IBM, HP, and also under linux. Hv was written in K&R "C" and contains about 45,000 lines of code and more than 500 "public" functions. Space will not permit using this forum to publish a programming manual.

Hv is freely available without warranty. Developers may use Hv providing they give proper attribution and citation. A complete manual[1], source code, installation instructions, and demo applications are available via anonymous ftp from two sites: the Super Computing Research Institute at Florida State University: *ftp.scri.fsu.edu* in the directory *pub/CLASsoft/files/Hv*, and from *america.pcs.cnu.edu* in the directory *pub/heddle/Hv*. All necessary files are compressed into one tar file named *Hv.1XXX.tar.Z* where *XXX* is a number representing the latest version.

## II The Hv Paradigm

An Hv application looks something like a cross between a Motif and a Macintosh application. It contains a single X window, (henceforth called the Hv main window) with a single main menubar, much like the Macintosh desktop. However, unlike the Macintosh desktop the Hv main window can be moved, resized, and iconified.

Within the Hv main window are additional window-like objects called *views*. Again, these views are like the windows on the Macintosh desktop, except they have increased functionality: dragging, resizing, pointer tracking (feedback), scrolling, zooming, zoom-to-fill, controls, etc. The views have a 3D sculptured appearance that provides an elegant look-and-feel to Hv applications. An example of a Hv main window containing multiple views is shown in Figure 1.

Contained in each Hv view are Hv *items*. These include but are not limited to sliders, buttons, rainbow scales, wheels, text, and, most importantly, *user-defined* items that represent real objects, such as detectors. Each item has a rich suite of *attributes* that can be used to control its behavior. To reiterate: the Hv main window contains one or more Hv views; each Hv view contains many Hv items.

An Hv view has three distinct areas, each being optional but typically all three being present. The *HotRect* is the primary "drawing" area where the bulk of the graphical items will live. The *Control* area is where sliders, buttons (of various flavors), and other control items are found. Finally the *Feedback* area is where updates associated with tracking the pointer are displayed.

In Figure 2 we display a typical view arrangement, designating the controls, HotRect, and feedback areas. Also in Figure 2 we point out a familiar set of manipulators for dragging, resizing, hiding, and scrolling an Hv view.

It is essential to understand the purpose of each area. The HotRect is the "canvas" for the view. Items, usually very application-specific representations of actual objects, will be dragged, resized and rotated within the HotRect. Results may be represented graphically and displayed within the HotRect. It is the area affected by scrolling and zooming. In most applications it will change frequently.

The Control area is relatively static, and it is also the only of the three areas that need not be rectangular. It gives the view its personality in the sense that the controls are specific to that view, as opposed to the main menubar which applies to all views. The Control area is not affected by scrolling or zooming. A typical application will have a small family of view types, each with a specialized set of controls. *In Hv applications the bulk of the power to control the state of an application resides in the* *views' controls, not in the main menubar, which tends to remain relatively simple.*

The Feedback area is where the results of pointer tracking are displayed. In most Hv applications, as the pointer is dragged through the HotRect, the text in the feedback area will be rapidly changing. The user stops at a point of interest and reads the information in the feedback area. Pointing at items in the HotRect and getting instant feedback is perhaps the most popular Hv feature. In any of the figures in this report that display Hv views you can find examples Hv's feedback mechanism.

The size and placement of these areas is at the discretion of the developer. For example, the view shown in Figure 3 has a very different layout than the view in Figure 2, since its parent application is less demanding in its need for controls and feedback. The arrangement of Figure 3 minimizes wasted screen real estate.

Some discussion of the figures in this document is appropriate. Hv provides for postscript and encapsulated postscript rendering of its views. In fact, that is how the figures of views used in this paper were produced. Thus, any representation of a view is faithful to its actual appearance on the screen.

One more point about the figures. In Figures 1-3, the view is shown as it appears on the screen, namely with three areas on a gray 3D sculptured background. Figure 4, however, shows only the contents of a view's HotRect on a clean "white" background. This is almost always what users want to print, just the contents of the HotRect. Printing the entire view is useful when producing a user's manual, so that one can refer to the controls when discussing their use. When a print request is made from an Hv application, the user can choose whether to print the entire view with all adornments or just the

HotRect.

Below we present an additional (but partial) list of features that come "for free" when programming with the Hv library:

- *Drag and Drop.* Hv will allow you to create items that can be dragged, resized and rotated, usually within the confines of the HotRect.

- *Drawing Tools.* Hv provides a set of drawing tools that can be optionally added to any view. Thus any view can be annotated with text, lines, rectangles, etc. prior to printing.

- *Scientific Plotting.* Hv provides a scientific plotter that can be integrated into any Hv application. The plotter includes spline smoothing and curve fitting of various types, such as to a sum of Legendre polynomials.

- *Simulations.* Each Hv view in an application can register a simulation by providing a *simulation process* and a *simulation time step* in milliseconds. Hv will automatically call the simulation process at intervals of the time step, for example every 100 ms. Thus an application could have the same simulation running independently in multiple cloned Hv views, or a completely different simulation in each Hv view.

- *Postscript printing.* Any application will be able to print its views to a postscript printer or to create postscript (or encapsulated postscript) files for inclusion into other documents.

- *Balloon-Help.* Any item created by an application can include a string that will appear in a balloon when the balloon-help feature is enabled (via an automatically provided `Help` menu) and the pointer is placed over the item.

• *On-line help.* When the user selects "Help" from the `Help` menu, a scrollable list of topics (automatically alphabetized) is presented. Clicking on an item brings up any available help. Developers only need to maintain a simple ASCII file containing the help text. This feature, along with the balloon-help, enables developers to easily add a substantive help capability to their applications.

• *Fonts and colors.* Hv provides 37 predefined fonts for use in Hv functions that take "font" as an argument, and 80 colors for use in Hv functions that take "color" as an argument. Some color manipulation routines are also provided.

• *Animated Items.* Any Hv item can be assigned a redraw time, for example it can be told to redraw itself every 500 ms, from which the developer can effect simple animation.

## III Typical Hv Applications

Hv was developed for scientific and engineering applications wherein part or even *most* of the purpose of the application is to produce *and interact with* some sort of graphical output. The interaction can occur in the usual way with controls (buttons, etc.) but also in a less familiar way: by placing the pointer over (and perhaps clicking) non-control items such as pieces of a physics detector or a map of the world. This concept might be clarified by a synopsis of some existing Hv applications, views from which are used for the figures in this document.

• ***ced.*** The original Hv application, *ced* displays multiple representations of the Continuous Electron Beam Accelerator Facility (CEBAF) Large Acceptance Spectrometer (CLAS), overlaying the raw and/or reconstructed data. The user can place the pointer on a detector component and get immediate feedback about the data for that component, such as raw electronic data, or perhaps reconstructed information such as the energy deposited.

• ***hvplot.*** Included as a demo with the Hv installation, *hvplot* is actually the most widely distributed and used Hv application. Scientific plots are generated from ASCII data files. The user can then interact with the plots in a variety of ways, such as zooming individual plots or double clicking to bring up various graphical editors of the plot attributes.

• ***tigris.*** Another CLAS related application. Like *ced*, *tigris* provides a graphical representation of the detector. In this case, however, the user clicks on various components to program a "trigger", which is a template for the data acquisition system. In effect the trigger instructs the data acquisition system regarding what class of events are considered interesting enough to record in an ensuing experiment.

• ***caps.*** In a commercial use of the Hv library, SPARTA, Inc. of Mclean VA has developed a theater missile defense simulation by combining a FORTRAN "engine" with an Hv graphical interface. Objects such as radars and interceptors are placed on maps and can then be dragged and rotated. Once the Hv interface has established a "scenario", it sends the information to the FORTRAN simulation and, upon completion of the simulation, graphically displays the results. The user can then interact with the results through the feedback mechanism and in some cases by clicking on their graphical representation.

These applications share the feature of requiring, or at the very least providing for, extensive user interaction with *complex, user-defined* graphical objects. Nuclear physics detector components and radar fans are not part of any standard widget set. With Hv, the developer can turn such objects into dynamic graphical entities in a straightforward manner.

For applications similar to one of the four described above, where users click and in some sense probe and/or modify specialized graphical items, or interact with the graphical representation of the data resulting from an analysis or simulation, then Hv should be an appropriate tool. For applications requiring only normal controls, such as buttons, file-selection boxes, sliders, etc., well known control toolkits such as Tcl/Tk might better serve the developer.

## IV Hv Programming

As mentioned, we cannot present a programming manual in this report. Instead, we will give a top-level overview of how an Hv application is developed. No actual code is presented nor is any actual Hv procedure described. Such details are available in the manual included in the Hv distribution. The purpose here is outline the basic steps used in constructing an Hv application and in doing so to provide further elucidation of the nature of Hv.

The design that must be carried out prior to the development of an Hv application involves deciding how many different types of views are required and then deciding on a view-by-view basis what control items are needed, what HotRect (usually user-defined application-specific) items are needed, and what manner of pointer tracking (feedback) would be informative.

Let us return to the CLAS event display (*ced*) and recreate part of the process of this preliminary design phase. A slice through the middle and showing the entire detector is presented in Figure 4. It is not important for the reader to have any knowledge of detectors, only to recognize that we want to represent a large object (CLAS) that is comprised of independent components (the various detector packages).

We decided to construct one view that shows the entire detector -- the view shown in Figure 4. We also recognized the need for several additional views, each showing only one component of CLAS, and a final view displaying the toroidal magnetic field that gives CLAS its spectroscopic capability. Some of these views can be found in Figure 1. Thus we designed *ced* to contain a handful of *view types*, each with its own controls, user-defined items, and feedback but all behaving consistently in terms of the user interface. Finally we note that each view type can be cloned indefinitely: We could have any number of magnetic field views, each a carbon copy in regards to the controls and feedback, but independently positioned, sized, and zoomed.

Having a design in mind, we are ready to begin coding. Our discussion is now completely generic. The main program in an Hv application generally consists of three lines: one to initialize Hv, one to call the application's private initialization, and a final call to enter an event dispatching loop.

The first step, the Hv initialization, performs all the underlying X initialization, creates the Hv main window, allocates the fonts and colors, and creates the main menubar. The third step, the event loop, is where X events such as pointer motion, pointer clicks and keystrokes are dispatched. *The developer does not need to know any of the details of the Hv initialization or how Hv processes X events.*

All the development goes into the second step, the private initialization. Here is where an application's initial views are created, along with the controls, user-defined items, and entries into the feedback area. Any routine initialization (such as initializing global variables) and additions to or modifications of the main menubar are also done here. This is represented schematically in Figure 5.

When creating the views and items, the developer will attach to them procedures that are called in response to various actions, entirely analogous (but simpler than) the "callbacks" of X and Motif. For example, when a view is created, a feedback procedure is (optionally) attached. This is the procedure that Hv will invoke as the pointer is moved within the view's HotRect. The user has to provide the code that will take the information passed to it by Hv (*i.e.*, the present pointer position) and convert it to a meaningful string which Hv will then display in the appropriate position in the feedback area.

As for the user-defined items, there are two methods available for the developer. One is to create *truly* user-defined items that Hv knows nothing about. Here the developer has to provide a procedure for drawing the item as well as various support procedures that tell Hv how the object should respond to scrolling, resizing, rotation, etc. A simpler method recommended whenever applicable is to "piggy-back" onto an existing Hv item. For example, many items (such as the detectors in CLAS) can often be represented by world-based polygons, *i.e.*, polygons with vertices stored as world rather than pixel locations. Hv has a predefined world-based polygon, so creating an item based on it has the advantage that Hv already knows how to handle all the basic drawing and item maintenance. The user only provides a customized drawing procedure that Hv calls after its basic polygon drawing that converts the item from generic to specific.

## V Summary

The Hv library is an option for graphics-intensive applications, especially scientific modeling and simulation. It transparently handles mundane tasks such as refreshing the window after it was occluded and then exposed (a task many novice programmers assume is done "by the system") and the administration of printing, fonts, and colors. More importantly, it provides a suite of tools such as coordinate transformations that are tailored as to their utility for such developments but are not provided by Motif and X.

ACKNOWLEDGEMENTS

The author is delighted to acknowledge the early users and critics of Hv, including Dave Doughty at CNU, Mick Guidal, Etienne Burtin, Mikhail Kossov and Kevin Beard at CEBAF, Tim Smith at the University of New Hampshire, Larry Dennis, Greg Riccardi and Peter Dragovitsch at Florida State University, Barry Preedom at The University of South Carolina, and Martin Durbin, Rick Addelson, Wai Lee and Laura Lee at SPARTA. Special acknowledgments are reserved for Mike Finn at INRI and CNU who contributed much code to a precursor development, and to James Muguira of CEBAF and CNU. This work was supported by The Southeastern Universities Research Association (SURA), which operates the Continuous Electron Beam Accelerator Facility (CEBAF) for the United States Department of Energy under contract DE-AC05-84ER40150.

REFERENCES

1) D. P. Heddle, *Hv Programming Manual* (unpublished), included in the Hv distribution or as a technical note: CLAS 95-024 from CEBAF, 12000 Jefferson Avenue, Newport News VA 23606.

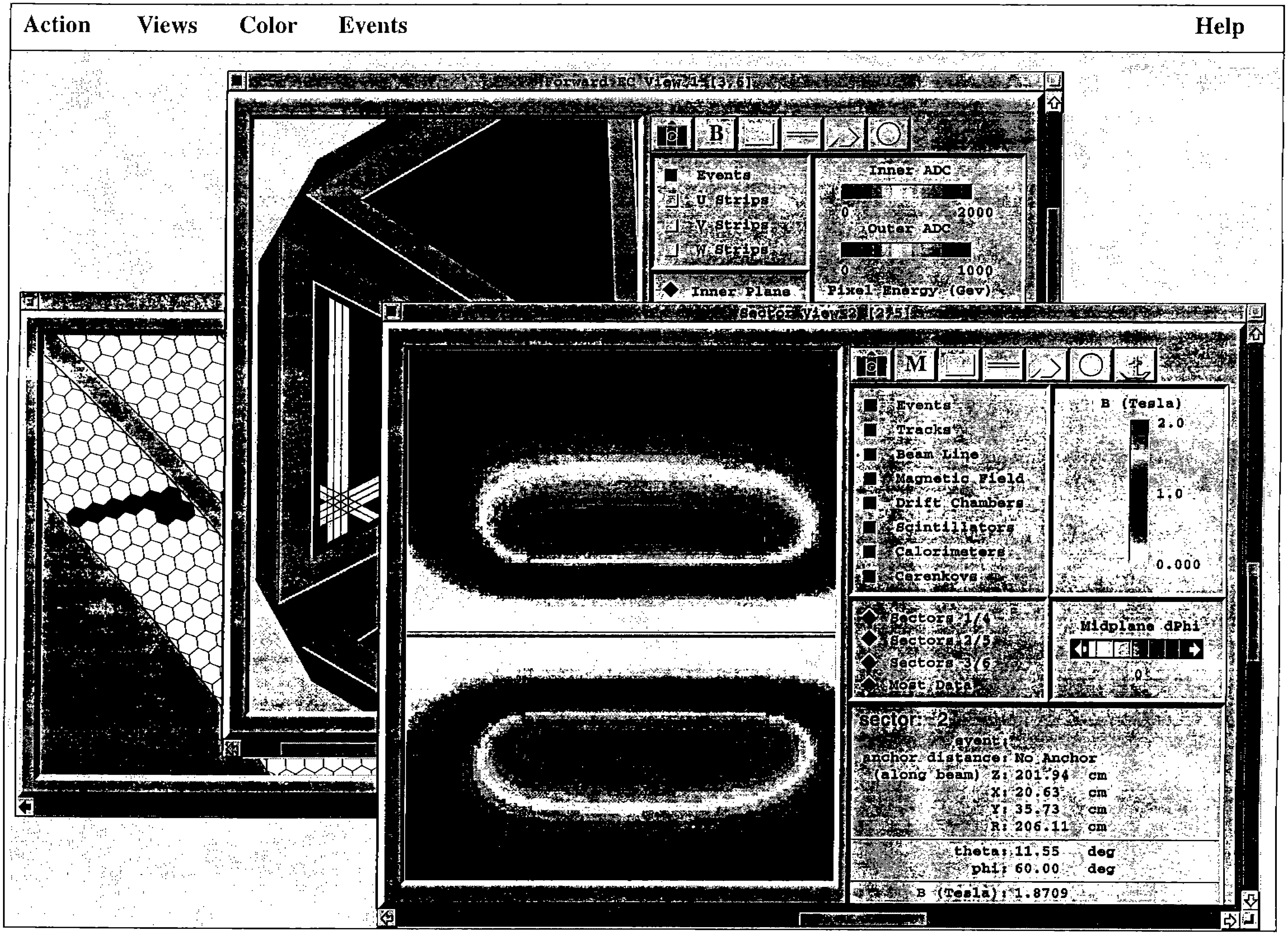
*Figure 1. A example of a main window from an Hv application. Three over-lapping views are present.*

*Figure 2. The anatomy of an Hv view. This view could be described as following the canonical Hv style: drawing tools (if desired) to left of the HotRect, textual buttons along the top, picture buttons and other controls on the top right and the feedback area on the bottom right.*

*Figure 3. An example of an Hv view with very different layout from the one shown in Figure 2. Here only minimal controls and feedback are required and are placed along the top. This allows the maximal allocation of screen real estate to the HotRect.*

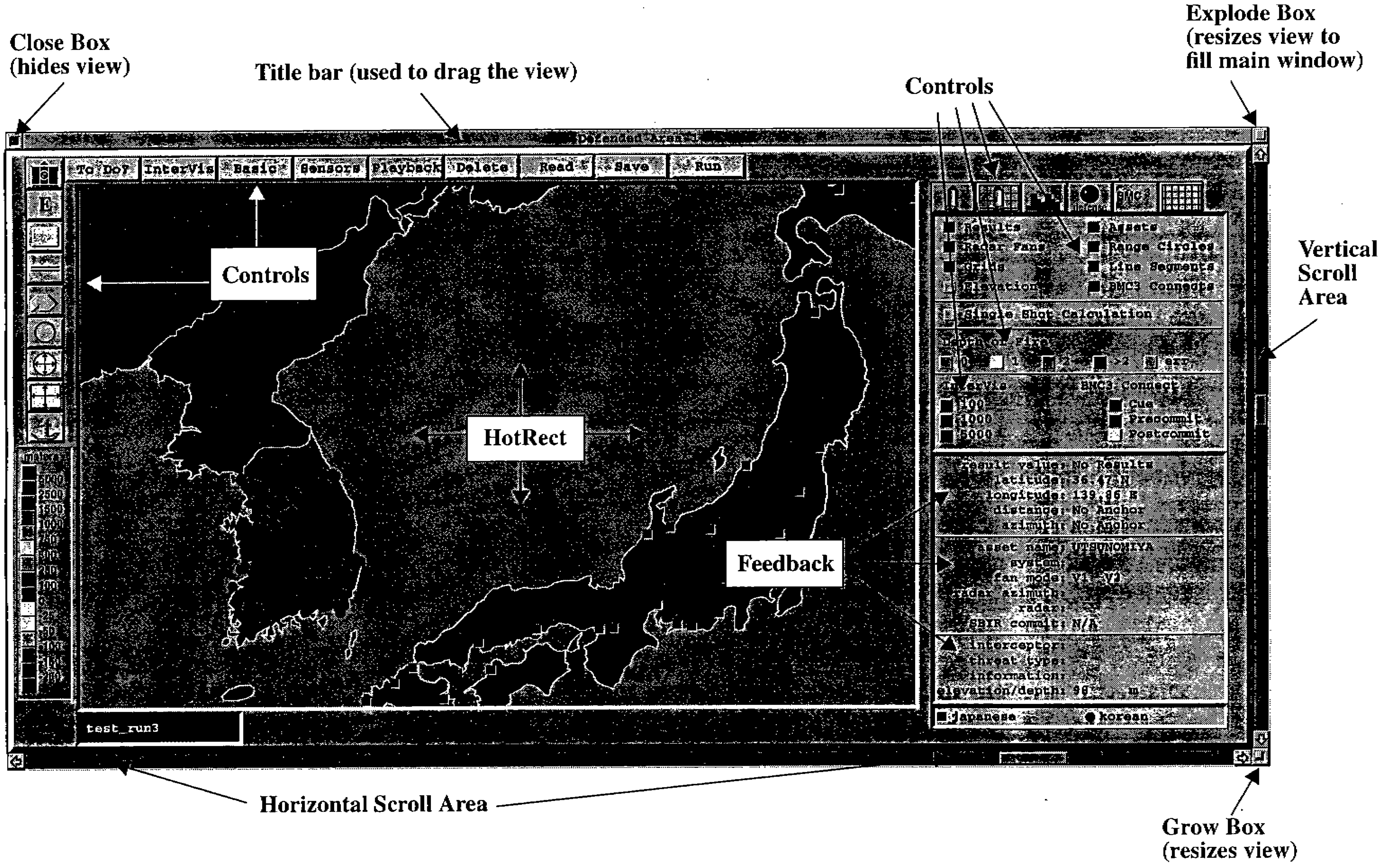
*Figure 4. A slice of the CLAS detector, which is comprised of concentrically (with the target at the center) arranged component packages. A knowledge of detectors is not important here, only the notion that we are representing a large, composite object.*

*Figure 5. A top level overview of Hv programming. The user's main program typically consists of three lines: one to initialize Hv, one to call the application specific initialization and a final call to an event processing loop. The user's initialization is further expanded into the standard sequence of steps, the most important of which is the creation of the application specific views.*



```
sector: 2
     event:
anchor distance: No Anchor
 (along beam) Z: 201.94  cm
              X: 20.63   cm
              Y: 35.73   cm
              R: 206.11  cm
     theta: 11.55   deg
       phi: 60.00   deg
   B (Tesla): 1.8709
```

Figure 1



Results
Radar Fans
Grids
Elevation
Assets
Range Circles
Line Segments
BMC3 Connects

Single Shot Calculation

Depth of Fire
0 1 2 >2 err

InterVis
100
1000
5000

BMC3 Connect
Cue
Precommit
Postcommit

result value: No Results
latitude: 36.47 N
longitude: 139.86 E
distance: No Anchor
azimuth: No Anchor

asset name: UTSUNOMIYA
system:
fan mode: V1 V2
radar azimuth:
radar:
SBIR commit: N/A

interceptor:
threat type:
information:
elevation/depth: 98 m

Japanese
korean

Figure 2

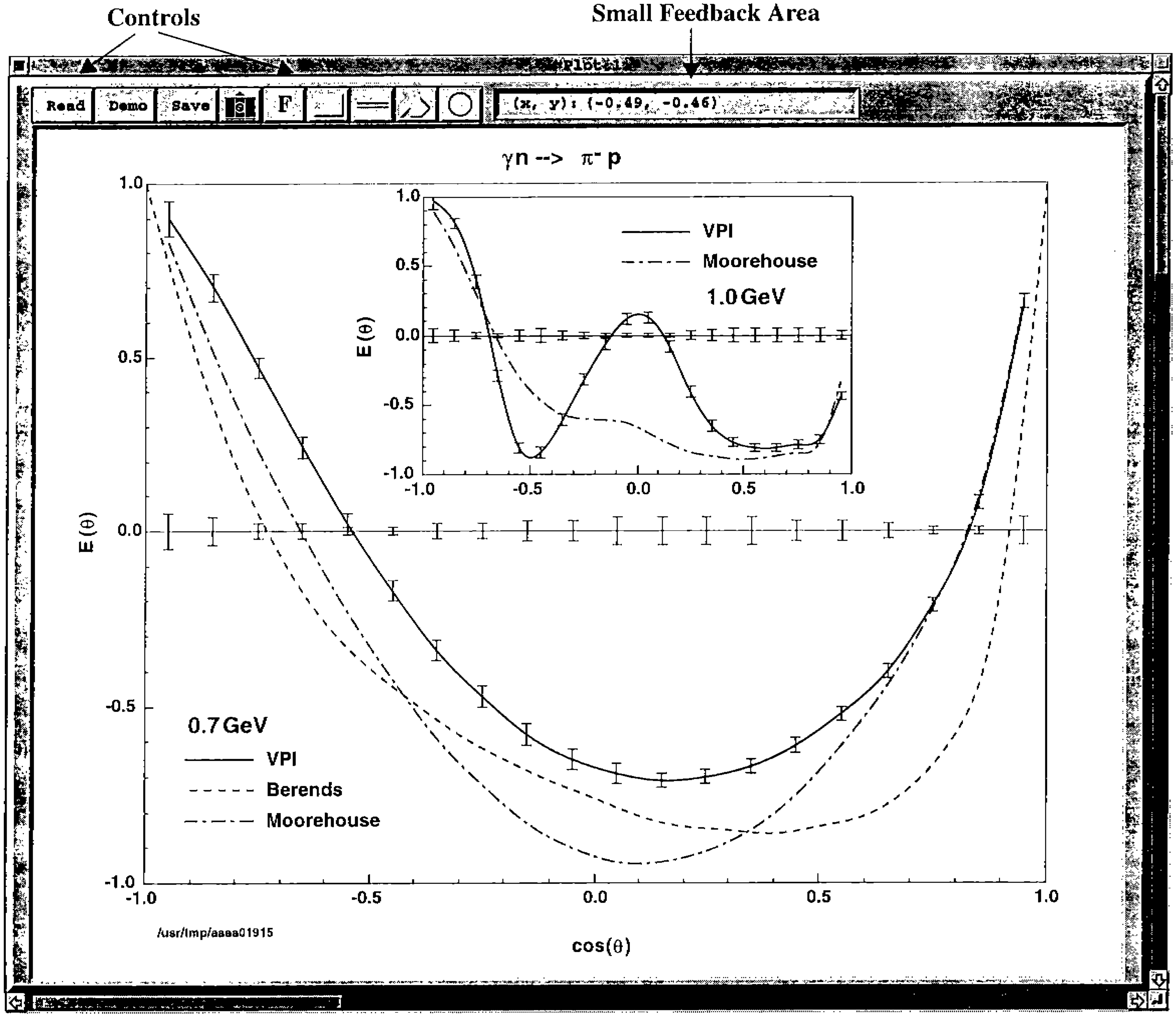

Controls
Small Feedback Area
Read
Demo
Save
F
(x, y): (-0.49, -0.46)
γn --> π⁻ p
E (θ)
cos(θ)
0.7 GeV
VPI
Berends
Moorehouse
1.0 GeV
VPI
Moorehouse
/usr/tmp/aaaa01915

Figure 3

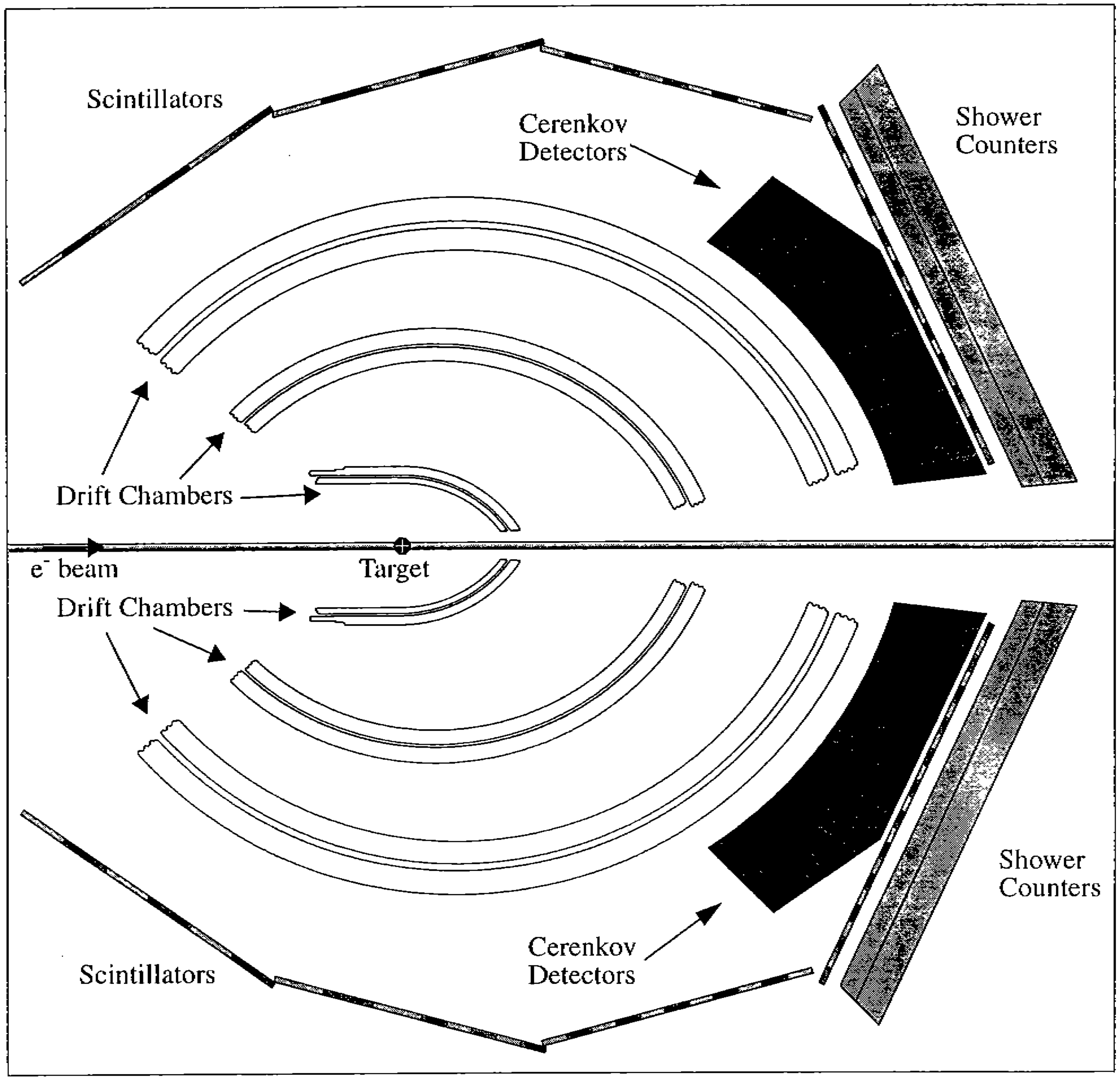

Scintillators
Cerenkov Detectors
Shower Counters
Drift Chambers
e⁻ beam
Target
Drift Chambers
Shower Counters
Cerenkov Detectors
Scintillators

Figure 4

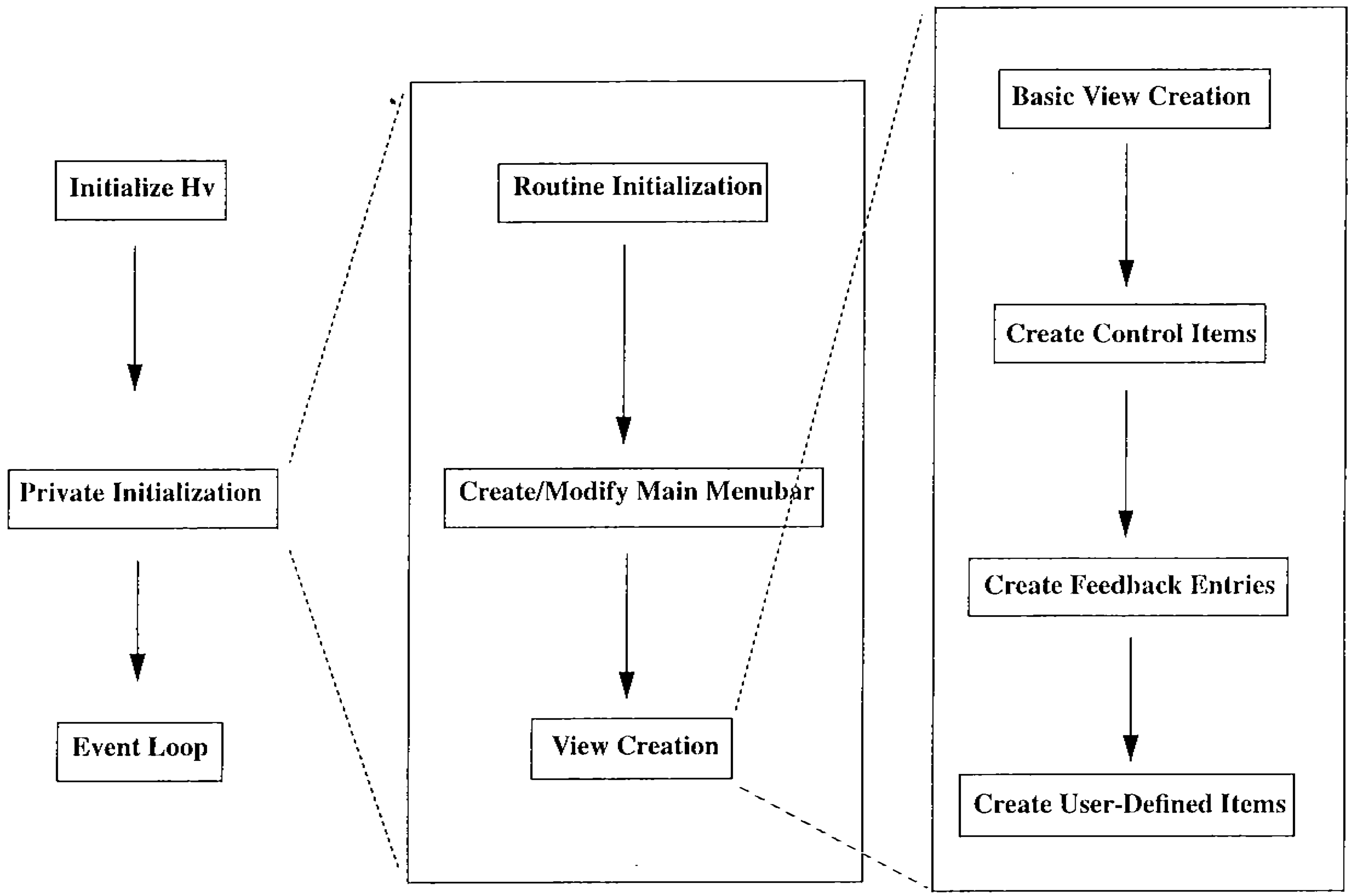

Initialize Hv
Private Initialization
Event Loop
Routine Initialization
Create/Modify Main Menubar
View Creation
Basic View Creation
Create Control Items
Create Feedback Entries
Create User-Defined Items

Figure 5