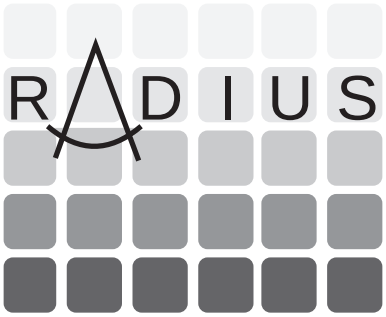
RADIUS
Common Development Environment

User's Manual



Version 1.0

Sponsored by the
Office of Research and Development
Under Contract 91-F 133700-000

Developed by
SRI International and **Martin Marietta**

Acknowledgements
The primary architect of the RCDE is Lynn Quam of SRI International, who developed the Cartographic Modeling Environment over the last decade. The other members of the team shaped the final product, which includes documentation and the Lisp-C/C++ interface:

Bill Bremner	Martin Marietta	Aaron Heller	SRI International
Bill Deriscavage	Martin Marietta	Tom Strat	SRI International
Doug Hackett	Martin Marietta		
Anthony Hoogs	Martin Marietta	Joseph Mundy	General Electric
Mark Horwedel	Martin Marietta		
Phil Rossomando	Martin Marietta	Richard Welty	Infologic
Janis Tomko	Martin Marietta		

Our thanks to the Office of Research and Development and ARPA for funding this effort.

Version 1.0 Edition, July 1993

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose

This document is intended as a user's manual for the RADIUS Common Development Environment (RCDE), which is an interactive workstation environment and standard toolset for manipulating imagery and constructing 3-D site models. The User's Manual will provide a description and several examples of the various stages of constructing site models. In addition, this document describes an interface between CLOS and C++ that permits a Lisp or C++ programmer to extend the functionality of the RCDE.

This User's Manual is addressed principally to researchers within the Image Understanding (IU) community, and is sponsored by the Office of Research and Development (ORD) and the Advanced Research Projects Agency (ARPA). The environment will be made available to all industry and university contractors who perform IU research for the U.S. Government, and is to be used for developing and comparing image understanding algorithms.

1.2 Scope of the Document

This manual is intended to aid IU researchers in understanding the RCDE user interface, data representations, and programming requirements. The beginning chapters of this document will assume that the user wishes to manipulate objects from the interactive window display and is not familiar with a Lisp or C++ programming environment. In order to extend the capabilities of the RCDE, the user must understand how to use the programmer's interface, which will be described in sufficient detail so that the interested reader/programmer can add additional functionality to the RCDE. A description of the RCDE at the functional level is supplied in the *RCDE Programmer's Reference Manual.*

The RCDE, when utilized to its full potential, will provide the user with the following capabilities;

Interactivity: Commands and menus, with mouse and context sensitivity.

Extensibility: Large and small scale system integration support.

Standard Operations and Representations: Functionality that is widely used by the IU community.

Reusable Concepts: Encouraged through object-oriented methodologies and a consistent data exchange format.

Seamless and Consistent Integration: Intuitively usable basic operations for all entities (*e.g.*, Images, Objects and Graphs) in the user interface.

Integrated Support for Imagery and 3-D Models: The ability to create a site model from imagery. The system provides interactive, three-dimensional features capable of being associated with imagery. Camera models are used to establish the relation between the imagery and the site model on a mathematical basis.

1.3 Applicable Documents

The following documents provide supplementary information:

- *RCDE Programmer's Reference Manual*
- *RCDE Installation Guide*
- Tutorial on image understanding environments, presented by D.T. Lawton at 1990 DARPA IU Workshop
- Sun Microsystem OpenWindows 3.0 Manual
- Sun Systems User Guide - (Desktop SPARCstation)

1.4 Document Organization

This document is organized into chapters that are summarized below. In the earlier part of the document, chapters are presented in pairs; the first contains a step-by-step scenario of how to use the RCDE, and the second lists and describes the menu items individually. The user is first exposed to application-specific examples before the details of each function are fully discussed. This approach is intended to serve those users who like a tutorial approach and those users who prefer to use the manual only as a reference.

- **Chapter 1 - Introduction** – This chapter describes the intended purpose and scope of the *RCDE User's Manual.*

- **Chapter 2 - RCDE Overview** – This chapter describes the RCDE architecture and user interface at a systems level.
- **Chapter 3 - Stack Usage Scenario** – This chapter describes a short scenario that introduces the user to the RCDE screen layout. It will also introduce the concept of data stacks and how to remove, copy, inspect, delete, and undelete the underlying data that is manipulated and viewed from the RCDE windows.
- **Chapter 4 - Stack Manipulation** – This chapter describes each stack manipulation option – what it does and how to invoke it.
- **Chapter 5 - Geometric View Transform Scenario** – This chapter describes a short scenario that demonstrates zooming, repositioning, and other window operations.
- **Chapter 6 - Geometric View Transforms** – This chapter describes the geometric operations that can be performed on images and site models.
- **Chapter 7 - Arithmetic Image Transform Scenarios** – This chapter describes a short step by step example of Boolean image operations such as xor, intersection, and negation. The scenario demonstrates the arithmetic effects in visual manner that are recognizable to the user.
- **Chapter 8 - Arithmetic Image Transforms** – This chapter describes each Arithmetic Image Photometric Transform menu option, what it does, and how to invoke it.
- **Chapter 9 - Enhancement Image Transforms** – This chapter describes pixel intensity manipulation and image window function menu options, what they do, and how to invoke them. No scenarios are required. By following each procedure, the user can execute the function. The user is introduced to the notion of an object with menu options (e.g., the view-object-hacking tool)
- **Chapter 10 - Graph, I/O, and Miscellaneous Operations** – This chapter describes other operations such as how to create histograms, plot a horizontal or vertical line of pixel intensities (line amplitude profile), save site models, load and save images, and adjust color maps.
- **Chapter 11 - Introduction to 3-D Modeling** – This chapter introduces the RCDE's 3-D site modeling capabilities and presents a simple scenario for creating and manipulating model objects.
- **Chapter 12 - The RCDE Objects** – This chapter presents how objects are created, grouped, and viewed within a pane. It introduces the concept of 2-D and 3-D worlds, 3-D objects and camera objects via a site model scenario.

- **Chapter 13 - The Camera Model** – This chapter describes the RCDE camera model in detail, including its creation and adjustment.
- **Chapter 14 - Terrain Data Integration** – This chapter describes the integration and manipulation of terrain data within the RCDE site model.
- **Chapter 15 - Building A Site Model** – This chapter integrates the previous four chapters by presenting a comprehensive view of the site modeling process.
- **Chapter 16 - Data Exchange** – This chapter discusses the file formats and data requirements necessary to transfer site models between systems.
- **Chapter 17 - Lisp-C/C++ Interface** – This chapter discusses the architecture supporting the programmer's interface between Lisp and C/C++. A scenario describing the interface is presented, as well as an overview of its operation and sufficient detail to allow the user to integrate C/C++ code with the RCDE.
- **Chapter 18 - On-line Documentation** – This chapter presents the RCDE's hypertext documentation reference system.

Chapter 2

RCDE Overview

2.1 Introduction

The RCDE is based on the Cartographic Modeling Environment (CME) software developed by SRI International, which allows a user to interactively derive 3-D models of the world using imagery and geometric constraints. The RCDE incorporates a Lisp-C/C++ (LCI) programming interface, developed at Martin Marietta, that permits IU researchers to develop new algorithms in C or C++ and integrate them into the baseline RCDE architecture. A robust language bridge allows data interchange and parameter passing between C/C++, Lisp/CLOS, and the RCDE baseline. The RCDE is currently hosted on the Sun SPARCstation platform under the UNIX operating system. Figure 2.1 illustrates the basic RCDE concept.

2.1.1 Historical Overview

The Cartographic Modeling Environment, developed at SRI, serves as the basis for the RCDE System. Figure 2.2 supplies a historical perspective of CME's development. ImagCalc was initially developed as a stand-alone system, providing facilities for image display, "electronic light-table" functionality, image processing, and a robust user interface. On top of ImagCalc, the Object system was designed to generate three-dimensional models (wire frame models), incorporating the pinhole camera model and geographic coordinate systems. A third component, TerrainCalc, was incorporated to render and display terrain features, including tiling of DTM data and texture mapping of digital imagery onto the terrain grid. When combined with Terrain-Calc, the Object system allows the 3-D model to be melded with the terrain model (built with TerrainCalc) and then rendered, including simulated fly-through sequences over arbitrary flight paths.

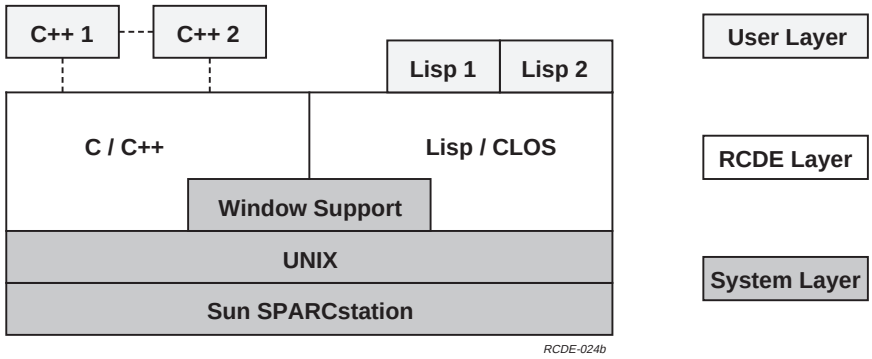


Figure 2.1: An Overview of the RCDE System – In the user layer, the C/C++ applications may be coupled to each other via Inter-Process Communication (IPC), while the applications are coupled to the RCDE via the Lisp-C/C++ Interface (LCI). Lisp applications may be loaded directly into the Lisp/CLOS process. The RCDE layer includes facilities and libraries accessible to Lisp, C and C++ programmers.

2.1.2 Architecture

The RCDE architecture, illustrated in Figure 2.3, is similar to CME, but contains two additional elements. The System Utilities are used by the RCDE to execute on the Sun platform and windowing system. The Programming Interface allows both Common Lisp Object System (CLOS) and C++ programmers to access RCDE objects when developing their own algorithms.

The RCDE System is divided into subsystems, which are defined as groups of objects that share similar functionality. Figure 2.4 lists the subsystems of the RCDE, grouping them functionally as defined in Figure 2.3.

2.2 Design Philosophy and Overview

This discussion should provide insight into the usage of the RCDE. It discusses the main design areas associated with the RCDE: data usage, representations, user interaction, and the programming environment.

2.2.1 Data Manipulation

The RCDE is designed to combine information from a variety of sources to support the construction of site models: imagery, terrain data, wire frame models, and collateral con-

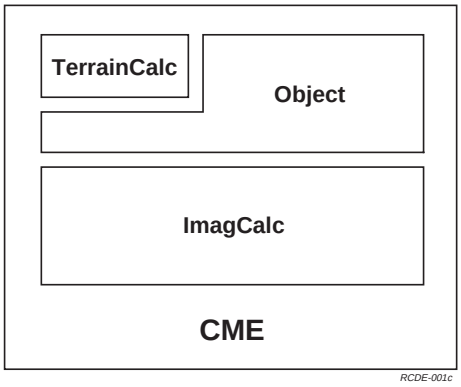


Figure 2.2: The CME System Developed at SRI

straints. In doing so, the system incorporates a mix of manual and automatic construction methods.

Figure 2.5 illustrates a typical approach for building a site model using the RCDE. Many paths through the system are possible depending on which site parameters are known; the diagram illustrates a few alternate paths. The user starts by loading an image of interest into a blank pane. Multiple image operations (e.g., contrast stretch, enhance) can be performed on the image to suit the viewer. When the user is ready to begin building a model, he creates a view transform, which creates a 3-D World (which includes a coordinate system and camera model). This transform data structure can be blank or oaded from another source such as a file. The user then creates model objects, overlaying their wire frames on the image. From this state, the user can perform image operations and model operations in each subsequent state; the notations are deleted from later states for convenience.

After the view transform is created, the user must load and align the Digital Terrain Model (DTM) with the image to ensure that the terrain data and the image occurs at the correct coordinates (e.g., Latitude/Longitude) of the image. The final step in creating a fully aligned model is to correct the default camera model by resectioning the camera with the image and DTM. Manual Adjustment of the objects in the scene produces a fully registered site model whose objects are in the correct 3-D location. The image then represents a snapshot of the model (objects and DTM) as taken from the given camera.

Multiple images of a site can be aligned to a single site model to generate multiple views from multiple cameras. If the camera parameters are already known, a new view transform (camera) can be created and specified explicitly, as illustrated in the right fork of Figure 2.5. This produces a second view of the scene that is already aligned; by copying selected collections of model objects (feature sets) into the new view, a fully aligned, multiple

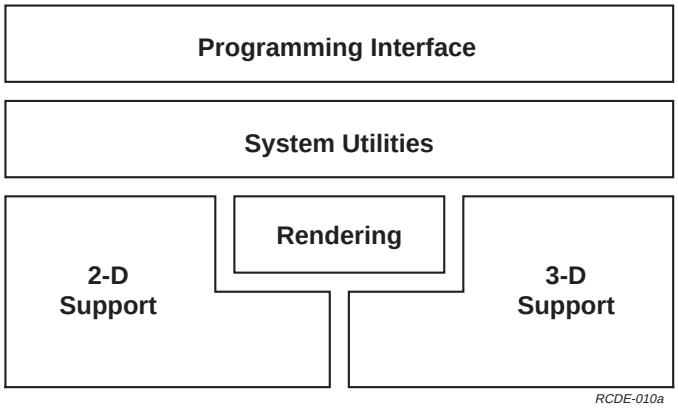


Figure 2.3: The RCDE Architecture – The programming interface enables C, C++ and Lisp applications to access the RCDE Capabilities.

view site model is produced. If the camera parameters are not known, a new blank view is created and the model feature sets are copied into it. The camera is then aligned either by manipulating it manually through the user interface or by using a resection technique, as illustrated in the left fork of Figure 2.5. More views of the scene can be generated by repeating the above procedure.

Finally, simple model rendering is available as a verification aid. The texture of each model face can assume a default shading to produce a realistic effect. The user then specifies an arbitrary number of rendering viewpoints (through the user interface) and the model is rendered. Viewing the model from multiple locations allows the user to review his model for consistency.

2.2.2 Data Representations

The VIEW is a fundamental component of the 3D-WORLD, which unifies the RCDE data structures for viewing on the display. The world contains objects that are viewed through a simple camera, which transforms the world coordinates to image coordinates. Objects in the scene are mapped to the image plane (film plane) of the camera.

The major components of a view are related as illustrated in Figure 2.6. The white boxes represent data objects, which include model elements (e.g., house, image), groupings of objects (e.g., feature sets), and transformations. The shaded boxes denote coordinate systems, which are numerical values stored in a separate container object.

From the left side of the figure, we can see that a view is associated with a number of 3-D feature sets, 2-D feature sets, tool feature sets, images, and view stacks. Each type of

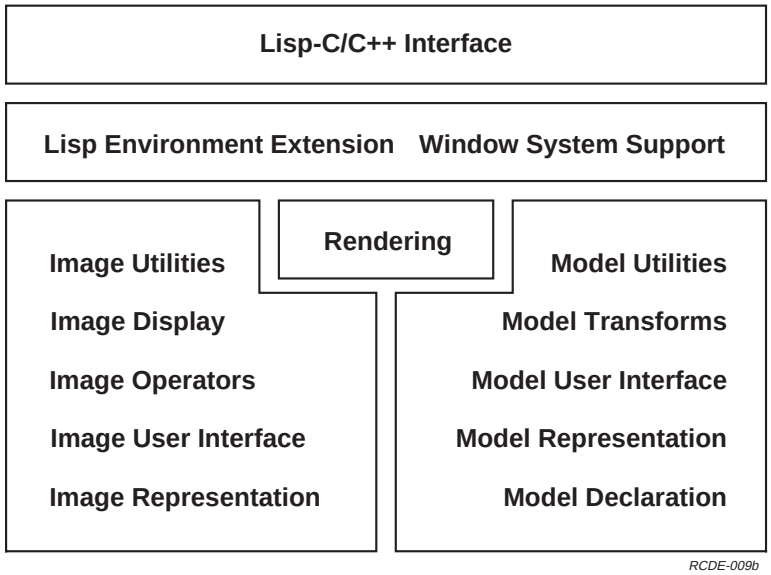


Figure 2.4: The RCDE Subsystems

feature set (3-D, 2-D, tool) is associated with a number of objects. Each object has its own internal coordinate system (object coordinates) and a transform matrix that maps object coordinates to some other coordinate system.

For example, follow the figure from the bottom, moving toward the right. Three-D models (e.g., houses) have their own object-centered coordinate system and a 4X4-TRANSFORM that maps object coordinates into a set of 3-D world coordinates. The 3-D world coordinates are stored by a 3D-WORLD object as a local vertical coordinate system (LVCS), whose origin is defined relative to a given site. A 3-D world has additional transformations to convert the LVCS to a universally recognized World Coordinate system, such as Universal Transverse Mercator (UTM), . Latitude-Longitude-Elevation, or geocentric coordinates.

The view is associated with a sensor (camera) model that maps 3-D world coordinates to 2-D world coordinates. The camera model is labeled "3-D→2-D Projection" in the figure. Chapter 13 presents more details of the camera model.

Views are also composed of images and 2-D feature sets, which contain 2-D objects. Both have transformations to transform local object or image coordinates to 2-D world coordinates, in common with the output of the camera model.

The 2-D world coordinates are then mapped into window (pane) coordinates by another 2-D transformation ("2-D Transform" in the figure). This transformation is necessary for

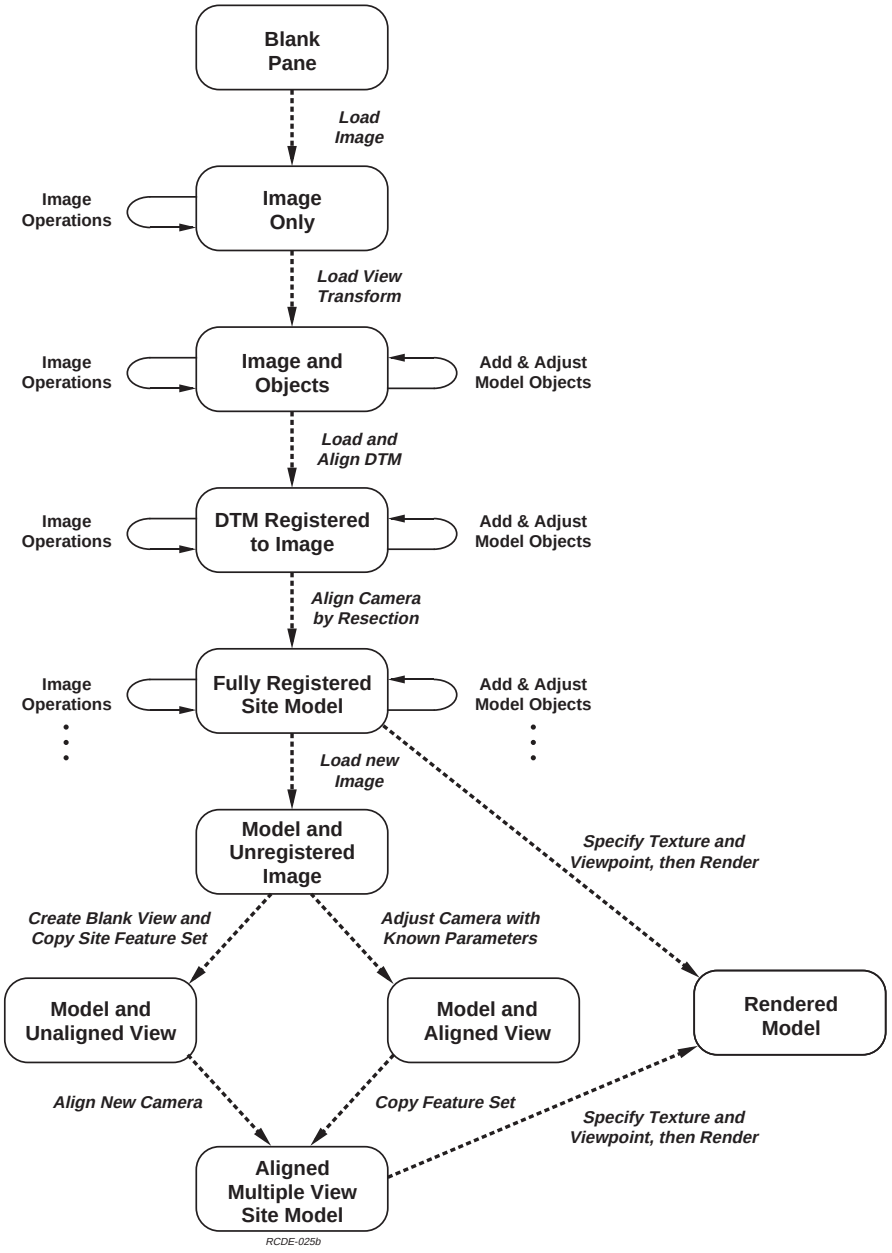


Figure 2.5: Example RCDE Functional Flow – Illustrates how the RCDE is typically used to manipulate images and to construct site models

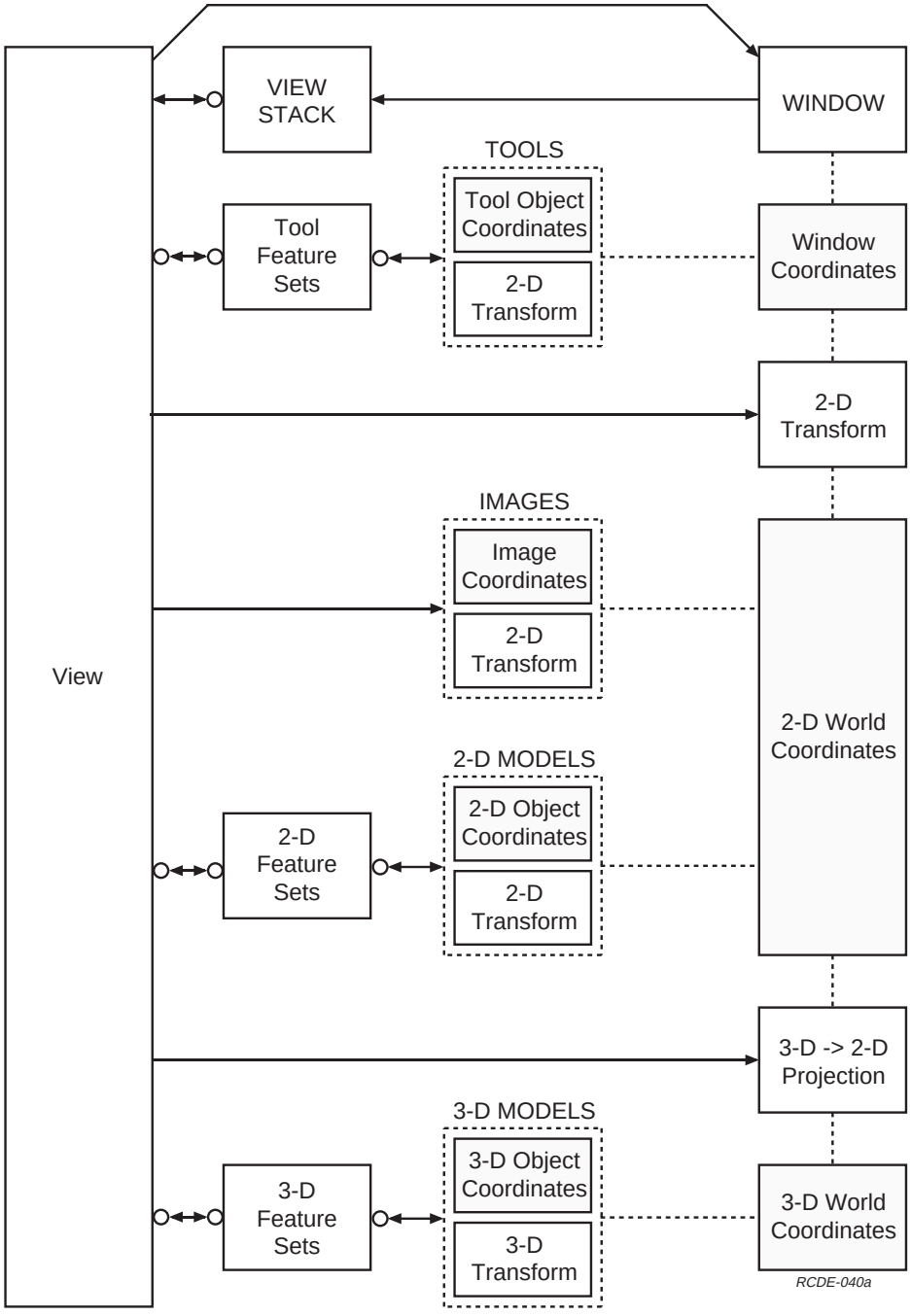


Figure 2.6: RCDE Data Representations

translations of the 2-D data within the window such as panning and zooming. The view may also contain Tool objects, which are designed to remain fixed in the pane even when the underlying data is panned or zoomed. All types of objects are therefore mapped to window coordinates in a similar way. A WINDOW-WORLD object is responsible for storing the window coordinate data; note that the window has a stack associated with it, which contains a number of views.

2.2.3 User Interface Overview

The RCDE was designed as a development environment; as such, the user is able to access visual elements of a scene and then perform new operations on them using his own code. For example, each display pane contains a stack of views, which are mouse sensitive in a variety of ways. When the mouse "points" at a particular object, it becomes highlighted. The object can then be manipulated directly or programmatically because an actual pointer to the object is made available in the Lisp Listener.

The RCDE user interface was originally developed as a standard pulldown menu driven interface. A command key (shortcut) interface was then added to use keyboard and mouse button interactions — the Bucky keys. The object-oriented nature of the user interface implies that each object instance contains its own particular set of functions. The Bucky key system therefore evolved from the need to create multiple identical objects which may react differently to the same system stimulus (e.g., camera motion). Hence, a function selection capability was given to each object instance.

When an object is highlighted, a set of functions are available that are specialized to that object. This context-sensitive environment is enhanced by the persistent nature of the Lucid Lisp environment. The most striking example of this is the Eval Cache facility, which is a mechanism to control the execution of display operations. It avoids replicating identical computations by storing a list of previously executed operational sequences and reusing the result, if available. The Eval Cache greatly increases system performance, especially for image operations.

Both keyboard and mouse input are used to drive the interface components, which are detailed in the following sections and figures.

2.2.3.1 Windows

The RCDE screen provides visual feedback to the user and provides a means of human interaction and control at various phases of the site modeling process. Through interaction with the RCDE, the user is able to control, manipulate, and view site model objects (houses, terrain models, buildings, images) within the 2-D and 3-D worlds associated with appropriate coordinate systems. The RCDE screen layout provides an efficient tool to study the site modeling process.

By default, the RCDE dedicates most of the screen to a regular array of display areas called panes, each of which contains a stack of displayable objects called views. The top view

on each stack is displayed on the pane. This array of panes is grouped in a display window called a frame. The panes are used to display all of the RCDE displayable objects, including images, graphs, geometric objects, and textual messages. Figure 2.7 illustrates an example frame, labeled **CME 2x2**. The frame contains four panes that each display a different view of a common site model. It also shows that multiple overlapping windows are available.

Menu Bar and Message Area The RCDE Menu Bar appears by default at the top of the screen. Just above the main menu buttons are two critical documentation areas. The top line, usually referred to as the Documentation Line, performs the following functions:

- **Object Name and Mouse Commands** – If the mouse is within the sensitivity radius of a sensitive object, the name of that object and the current choices for the left, middle, and right mouse buttons are described in the Documentation Line.
- **Image Pane Mouse Button Commands** – If the mouse is not in an object context, the choices for the left, middle, and right mouse buttons are described in the Documentation Line.
- **Menu Item Description** – When the **Main Menu** or any other pop-up menu is on the screen, the Documentation Line displays a one-line description of the menu choice that is under the mouse.
- **User Prompt** – Most main menu selections prompt the user for more information, such as which image or object to use for the operation and which pane to use for the result. The Documentation Line is used to display the prompt messages.

The second text line in the Menu bar displays important information about an object or pane whenever a selection occurs. The information depends on the context, but generally includes the the pixel location in image coordinates, the pixel value selected, the pixel location in 2-D (u, v) coordinates, and the 3-D coordinates of a point on the terrain, if one exists.

On the bottom of the Menu Bar is a row of pulldown menu buttons, which can be selected depending on the window manager being used. Selecting a menu item can result in three actions: 1) the function will be immediately executed, 2) a popup menu will appear that requires the user to supply some auxiliary information, or 3) another submenu will appear that requires the user to select an option. In Figure 2.7, the Create Object menu has been pinned and the Geom menu is in the process of being pulled down. The RCDE menu options will be described later in the context of views, transforms, and objects.

2.2.4 Object Sensitivity

Each object instance has its own mouse sensitivity method which defines how that object responds to the presence of the mouse. Usually, an object sensitivity method computes the

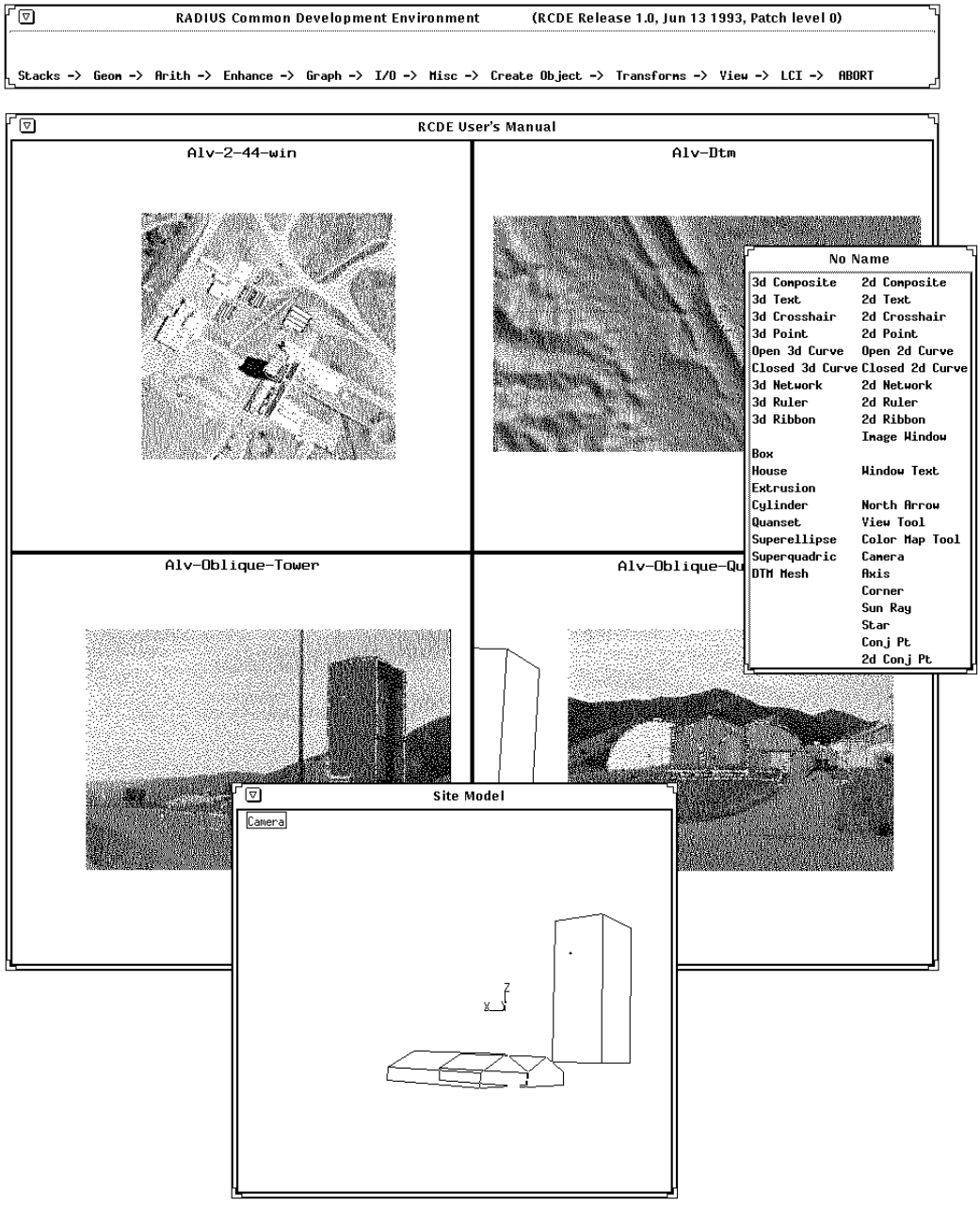


Figure 2.7: The RCDE User Interface

distance from the mouse cursor to visible parts of the wire frame depiction of the object in the window. When the mouse is positioned within the sensitivity radius of a sensitive object, the object is highlighted on the view and the documentation line is updated to indicate the name of the object and the operations accessible via the left, middle, and right mouse buttons for the current state of the RCDE Control, Meta, Super, and Hyper keys (sometimes referred to as the Bucky keys). When the mouse is within the sensitivity radius of more than one object, the nearest object is chosen. When no object is selected, the pane Bucky menu is active.

The sensitivity of entire feature sets of objects may be turned on and off using the View menu Feature Set commands. The mouse sensitivity of individual objects cannot be controlled from the user interface directly, but can be changed through Lisp.

2.2.4.1 Bucky Keys and Menus

Certain key functions are supported by a second menu system called the Bucky menus. On a pane, this Bucky menu can be invoked by depressing the right mouse button. A Bucky menu will appear, allowing the user to simply left mouse click on the functions that are identified within the menu system. The Bucky menus support Objects,Views and Image functions. Please refer to the *RCDE Installation Guide* for Bucky key mappings. These Bucky keys are intended to be a shortcut alternative to menu selections. In order to use the Bucky keys, ensure that you have correctly modified your environment variables as described in the *RCDE Installation Guide*. These variables change the functionality of the keys listed above.

Bucky commands provide a very efficient graphical interface to objects. To invoke a Bucky menu command, the user moves the mouse to select a particular object, then invokes the Bucky menu command by depressing some combination of the Bucky keys and then clicking one of the three mouse buttons (while the Bucky keys are still depressed).

Each object has a Bucky menu that specifies the menu of operations appropriate to objects of that object class. The current Bucky menu may be invoked for a given object by holding down the Hyper and Control keys and clicking the right mouse button while pointing to the object. A shorthand for that action is [H--C$\rightarrow$--R].

2.2.4.2 Lisp Interaction Window

The Lisp Interaction window, illustrated in Figure 2.8, is an Emacs buffer in a separate window that will accept any Lisp form and pass it to the Lucid interpreter. This Lisp form may be viewed as a command or instruction to the RCDE, and therefore the window can be used as a command line interface. Most operations, whether initiated through the command line interface or by other means, return pointers (and printed representations) to the Lisp Interaction Window. These pointers are useful as a means of referencing specific intermediate results.

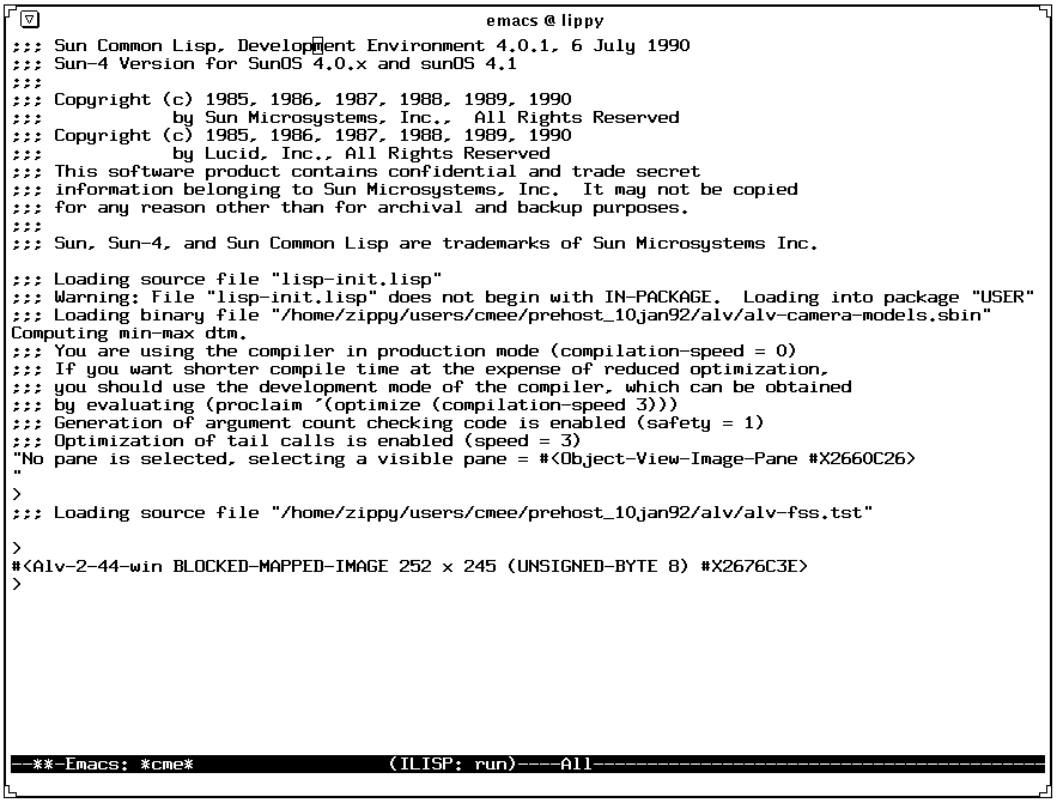


Figure 2.8: The Lisp Interaction Window

The Lisp Interaction window provides a means of controlling the RCDE via Lisp command functions. The Lisp interactive window also provides a means of presenting the user with text output as part of the RCDE feedback when the user invokes RCDE operations via the user interface or as part of a Lisp form.

When the RCDE is invoked, a separate Emacs buffer is created that interfaces to the Lucid interpreter. Any runtime error that is detected is passed along to the user in the form of an abort alert. The user may then choose to "inspect" the stream of Lisp commands in order to determine the source of the error. This interface with the Lucid debugger allows the user to quickly prototype new algorithms and reload code without tedious compilation.

2.2.5 Development Environment

The RCDE was prototyped as an object-oriented system over a period of more than 15 years. As a prototype, it was a research tool, designed for the user to develop code while using the system. As a result, the RCDE supplies a seamlessly coupled set of tools for application development. The coupling is based on the fine grained code-reuse philosophy of the object-oriented paradigm. The user interface provides direct access to some of these objects. Those objects are in turn supported by more abstract objects in the RCDE environment.

2.2.5.1 Extensible Object Hierarchies

The RCDE provides a comprehensive hierarchy of object classes for program development. It is relatively easy to navigate through that hierarchy and attach new objects to extend the hierarchy for specialized applications. Specifically, the RCDE includes hierarchies for the following application support areas:

- Images
- 3-D Wire Frame Models, Terrain Objects
- Coordinate Systems, Transforms between Coordinate Systems
- 2-D Display objects, Views, Feature Sets
- Menu objects

2.2.5.2 Lisp/C++ Interface

As part of the RCDE, Martin Marietta has developed a robust Lisp-C/C++ Interface (LCI). This interface allows the user to compile, link, load, run, and debug code written in C/C++. As a result, the LCI provides a mechanism that allows previously developed RCDE code to be reuseable by the C/C++ programmer. Conversely, code written in C/C++ can be reused by the Lisp programmer via this interface.

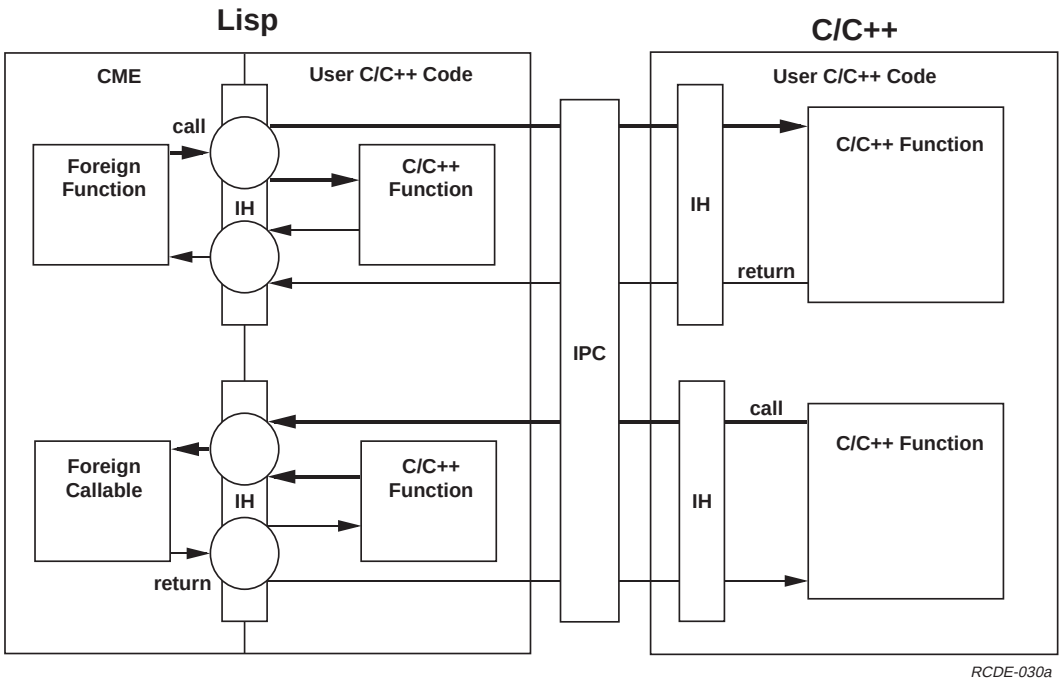


Figure 2.9: A Top-Level View of the Lisp/C++ Interface Design

The RCDE system provides a library of interface modules and a means of generating the Lisp code to couple the C/C++ module with Lisp. The LCI menu option provides a menu interface to the development and debugging interface and is fully described in Chapter 17. This functionality is provided using the Lucid Foreign Function Interface, which provides for the conversion of C/C++ argument parameters into equivalent Lisp arguments and *vice-versa*.

The LCI provides two modes of communication. The first mode is a performance mode in which user application code is loaded into the same process as the RCDE process. The code is linked to the RCDE and can be run in that process. The second mode is a debugging mode, in which the application code is loaded into a separate process from the RCDE process. In this mode, the user may run the application code in any chosen debugger.

Figure 2.9 illustrates the overall design of the LCI. The figure illustrates both modes of operation, which do not occur at the same time. In Performance mode, the shaded functions are loaded into the Lisp address space and execute through the Lisp foreign function facilities. In Debugging mode, the user code executes in a separate process (on the far right) and an interprocess communications module transfers data between the two processes.

Chapter 3

Stack Usage Scenario

The *RCDE Installation Guide* contains a confidence test scenario that should be executed by every user. If you have successfully executed the confidence test scenario, you will notice that you can move a graphic object (house model) by simply left clicking on the object (selecting it), and moving the cursor. The house object will track (move with) the cursor movement within the pane. Having experienced some degree of success with the RCDE user interface, you are now ready to move on to the next phase which guides you through some practical scenarios that explore the RCDE in greater detail.

3.1 Taking Charge

To check the response of Lisp, move the cursor to the Lisp Interaction window and select the ***cme*** buffer. Find the Lisp prompt (a greater-than sign), which is at the end of the buffer. You can go to the end of the buffer using **M->**. Type

```
()
```

followed by a carriage return. The Lisp Interaction window should return `NIL`.

If the Lisp debugger window responds, but the menu items appear to be unresponsive, type the following command in the Lisp Interaction window (followed by a carriage return):

```
> (ic::repl)
```

Note — If you are sure that the system is configured properly and if you select a menu item and the system does not respond, you probably need to respond to a Lisp function via the Lisp Interaction window. Look to either the Documentation Line or the Lisp Interaction window for prompts that can assist you in resolving this condition. Further error recovery measures are documented in the *RCDE Installation Guide*.

3.2 Scenario Initialization

Begin the scenario by loading the ALV site model, which will be used throughout the document:

1. Select the Load Site Model function from the I/O menu. A popup menu of available site models will appear.

2. Select the ALV entry from the menu. The popup menu will disappear, and a frame with four panes will appear. The frame contains four views of Martin Marietta's Autonomous Land Vehicle site in Denver. (The upper right pane is an image representation of the site's terrain.)

   During the course of any scenario, you may make a mistake which alters the appearance of a display pane beyond that of the scenario description. You can always get back to the initial starting position and restart by clearing the frame and reloading the stored site model. To do so, follow the procedures detailed below:

3. Select the Misc menu option from the Menu Bar by positioning the cursor on the menu option and clicking with the left mouse button. A menu should appear.

4. From the pulldown menu choices, place the cursor on the menu choice labeled Clear all Panes and left mouse click.

5. Select the frame, by placing the cursor in one of the panels and left mouse click. Four blank panes should appear in the **Alv 2x2** frame, and the Misc menu should disappear.

6. Select the I/O menu option from the horizontal menu bar by positioning the cursor on the menu and clicking with the left mouse button. A pull down menu should appear.

7. From the pulldown menu choices, place the cursor on the menu choice labeled Load Site Model and left mouse click. A menu should appear.

8. From the pulldown menu choices, place the cursor on the menu choice labeled ALV and left mouse click. The I/O submenu should disappear and the ALV site model should appear in the four frames.

3.3 Stack Scenario

This introductory scenario illustrates how objects and viewable data are managed by the RCDE environment. Each of the four panes in the CME 2x2 frame is a stack of views that may contain 3-D objects, 2-D objects, tool objects, and imagery. Operations for manipulating these stacks are found in the Stacks menu. As the mouse moves over the menu items, context-sensitive documentation appears in the Documentation Line. In addition to

moving views from pane to pane, the user can get information on the stack contents as well as refresh the panes if necessary.

This scenario is not exhaustive, but it does give the user a feel for the kind of functionality the Stack operations provide. The next chapter discusses all of the stack operations in detail, and is intended to be used as reference. In summary, this chapter conveys that:

- Each pane has a stack of views
- The views in a stack can be copied, moved, or deleted from stacks.
- The views in a stack can be described using the Lisp environment.
- The views can be moved between stacks, added, or deleted.
- Some heavily used functions may be invoked by Bucky menus, Bucky keys or the Stacks menu.
- The Bucky key functions serve as "command accelerators", often assuming default parameters.
- The Bucky menu can be used as "cue card" for identifying Bucky key combinations as Bucky keys are the preferred way to initiate actions.

Begin the scenario as follows. Since a site model is already loaded, model, you can get text descriptions of the views within the panes:

1. Select the upper left pane (titled **Alv-2-44-win**) with the left mouse button. This performs the Select function on the image in that view. Notice that the border of the pane is outlined to indicate that it is selected. Also the Lisp Interaction window will print out something similar to this:

```
>
#<Alv-2-44-win BLOCKED-ARRAY-MAPPED-IMAGE 252 x 245 (UNSIGNED-BYTE
8) #X3D7725E>
>
```

   This function describes the image and retuns a Lisp pointer to it. The Documentation Line will also display information about the selected pixel in the image: its location in image coordinates, the intensity value, the pixel location in 2-D (u, v) coordinates, and the 3-D coordinates of a point on the terrain, if one exists.

You can also get Lisp descriptions of pane items:

1. Select the Describe Image function from the Stack menu. Note that the Documentation Line describes the menu functions when the cursor is moved over the menu items.

2. Next, select the lower left pane (titled **Alv-Oblique-Tower**) with the left mouse button. This description appears in the Lisp Interaction window, as follows:

```
>
(LOAD-IMAGE "$CMEHOME[1]/alv/alv-oblique-tower.pic")
>
```

   This indicates that the pane contains an image that has been loaded with the **load-image** function. If you try this command later after a number of other functions have been performed on a pane, you will see that the consecutive Lisp function calls are saved by the Eval Cache mechanism.

The panes of a frame are stacks that keep the results of many operations. The lower panes of the ALV site contain other views below the visible ones, which can be retrieved with stack operations:

1. Select the Fwd Cycle command from the Stacks menu, then select the lower left pane with the left mouse button. The lower left pane should change to reflect a new view, labeled **Alv-3-42**.

There may be multiple ways to select the same function. Place the cursor on a pane and click the right mouse button. Another menu will appear, which is called the Bucky menu. Functions on images or panes can be performed by selecting the entries of the menu. Notice that many of the functions in the menu also appear in the Stacks submenu.

1. Select Cycle Stack from the Bucky menu, then select the lower right pane (titled **Alv-Oblique-Quanset**). The pane should reveal the **Alv-3-41** view. In this case, the Bucky menu item Cycle Stack and the menu function Fwd Cycle perform the same functions. In other cases, the menu is intended as a short cut and some default function arguments are supplied.

2. Now explore how the Bucky keys work. Notice the column named "Bucky keys" on the left side of the Bucky menu. This designates the key combinations Hyper, Super, Meta, and Control[2]. The top of the Bucky menu contains the words Left, Middle, and Right, which refer to the mouse buttons. When the keys are pressed in conjunction with a mouse click, the function is executed. The Bucky keys implement a short cut to selecting functions on the main menu, and more functions are available there.

3. Depress and hold the Meta and Control keys simultaneously and click the left mouse button with the cursor in the lower right pane. A shorthand for this sequence which specifies the Bucky key/mouse combination is [--M C→ L--]. (The letters on the left

[1] Substitute the appropriate path for $CMEHOME at your site

[2] Refer to the *RCDE Installation Guide* for details of the Bucky key locations

side of the arrow represent the Bucky keys, and the letters on the right represent the mouse buttons. Dashes represent the keys not selected.) With each mouse click you should see the view change, indicating the stack is cycling.

You can copy the views from stack to stack:

1. Select the Copy View function from the Stacks menu item. The Documentation Line will prompt you to pick a source view.
2. Select the pane titled **Alv-2-44-win**. The Documentation Line will prompt you to pick a destination view.
3. Select the lower left pane, which contains the view **Alv-3-42**. The view in the upper left pane should now appear in the lower left pane. Both left panes should contain the **Alv-2-44-win** image. You have copied a view from one pane to another.
4. Note the Bucky menu option Copy to Here. This Bucky menu function is identical to the Stacks menu option Copy View, except that the currently active (highlighted) pane is assumed to be the source pane.

Suppose at this time, we wish to inspect the stack in the lower left pane, without cycling through it.

1. Invoke the Inspect Stack function from the Stacks submenu, then select the desired stack.
2. A message similar to the following will appear within the Lisp interaction window:

```
> #<List 3CAE899>
[0] (LOAD-IMAGE "$CMEHOME/alv/alv-2-44-win.g0")
[1] (LOAD-IMAGE "$CMEHOME/alv/alv-oblique-tower.pic")
[2] (LOAD-IMAGE "$CMEHOME/alv/alv-3-42.g0")
>>
```

3. After you have reviewed the stack contents, move the cursor to the Lisp Interaction window and type **:q** followed by a carriage return.

Try invoking the same function from the Bucky keys.

1. Simultaneously, hold down the Hyper, Super, and Meta keys, move the cursor to the lower left pane, and depress the left mouse button [H S M-$\rightarrow$ L--].
2. The same message as before should appear in the Lisp Interaction window.
3. Move the cursor to the Lisp Interaction window and type **:q** followed by a carriage return when you have reviewed the stack contents.

Now explore some of the differences between menus and Bucky keys. We will move the view in the lower left corner to the pane in the upper right.

1. Select the lower left corner pane by moving the cursor to this pane and left mouse clicking in it. Note that the selected pane is highlighted.

2. Next, position the cursor in the upper right corner pane. Hold down the Meta button and click the right mouse button [--M-→--R]. The view in the lower left corner should move to the upper right.

3. Cycle through the stack on the upper right corner pane by holding down the Meta and Control keys and clicking the left mouse button [--MC→L--]. Make sure the cursor is within the pane you wish to cycle.

4. Now move the cursor to the lower left pane, hold down Meta and Control simultaneously, and left mouse click several times to cycle through this stack. You will notice that the view has moved panes.

Now move the view back to the lower left corner using a function from the Menu Bar.

1. Select the Move Object menu item. Notice that you are able to select a source and destination pane when using the command from a menu, but you must have previously selected the source pane when you use the Bucky keys. That is, Bucky operations often assume a default source and/or destination pane.

2. Following the prompt, left mouse click in the upper right corner, then move the cursor to the lower left corner and left mouse click again. The view will again be moved to the lower left corner pane.

You can pop the stack, which removes the view from the pane. The view is not permanently removed, however, and can be recovered with the Unkill Top function.

1. Select the Pop Stack function from the Stacks menu, then select the view in the lower left pane. The view will be removed from the stack.

2. You can view the stack via the Fwd Cycle, Rev Cycle or Inspect Stack commands to ensure that you have indeed removed the view.

3. To get the view back, select the Unkill Top function from the Stacks menu, then select the pane to restore.

4. You can perform more drastic kill operations, but please read the next chapter before you Pop Expunge, or Kill Stack.

Chapter 4

Stack Manipulation

The Stacks menu provides a list of functions directed toward pane and pane display management. Each pane contains a storage mechanism called a stack, which is a list of views. Conventional stack operations such as push, pop, and copy are can be performed with respect to the views on the stack.

Name Image – Associate a name with the selected image.

Menu Bar: Select the Name Image function from the Stacks menu. Following the Documentation Line prompt, select the pane containing the desired image. A popup menu will appear; enter the reference name followed by a carriage return. The named image can now be referenced by the specified name.

Inspect Image – Display the instance variables (slots) associated with the selected image using the Lisp Inspector.

The Lisp Inspector is a tool for examining Lisp data structures, as illustrated in Figure 4.1. Since Lisp is engaged while in the Inspector, subsequent attempts to use other RCDE menu selections without exiting the Inspector will queue up menu selections. Therefore, the user is cautioned to type **q:** followed by a carriage return to exit the Lisp inspector and return control to the Lisp top level.

Menu Bar: Select the Inspect Image function from the Stacks menu. Following the Documentation Line prompt, select the pane containing the desired image. The Lisp Interaction window should appear similar to the one listed in Figure 4.1. To exit the Lisp Inspector, type **:q** followed by a carriage return at the ">>" prompt.

Bucky Keys: Holding down the Hyper, Super, and Meta Bucky keys, click the left mouse button in the pane of choice [H S M-→ L--]. window will enter the Lisp

Inspector. To exit the Lisp Inspector, type **:q** followed by a carriage return at the ">>" prompt.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the function Inspect Image from the Bucky menu. Following the Documentation Line prompt, select the pane containing the desired image. The Lisp Interaction window should appear similar to the one listed in Figure 4.1. To exit the Lisp Inspector, type **:q** followed by a carriage return at the ">>" prompt.

Describe Image – Display a description of the form stored for the given image in the Lisp Interaction window. A Lisp form will be displayed that describes the series of operations performed on that stack item, such as:

```
(LOAD-IMAGE ''$CMEHOME/alv/alv-oblique-tower.pic'')
```

Menu Bar: Select the Describe Image function from the Stacks menu. Following the Documentation Line prompt, select the pane containing the desired image. A Lisp form similar to the above will appear in the Lisp Interaction window.

Bucky Keys: Holding down the Hyper, Super, and Meta Bucky keys, select the pane containing the desired image with the middle mouse button. [H S M C→ - M -]. A Lisp form similar to the above will appear in the Lisp Interaction window.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Describe Image function from the Bucky menu. Following the Documentation Line prompt, select the pane containing the desired image. A Lisp form similar to the above will appear in the Lisp Interaction window.

Inspect Stack – Display a description of each view on the stack in the Lisp Inspector.

The Lisp Inspector is a tool for examining Lisp data structures, as illustrated in Figure 4.1. Since Lisp is engaged while in the Inspector, subsequent attempts to use other RCDE menu selections without exiting the Inspector will queue up menu selections. Therefore, the user is cautioned to type **q:** followed by a carriage return to exit the Lisp inspector and return control to the Lisp top level.

Menu Bar: Select the Inspect Stack function from the Stacks menu. Following the Documentation Line prompt, select the pane containing the desired image. All views on the stack will be described in the Lisp Interaction window. To exit the Lisp Inspector, type **:q** followed by a carriage return at the ">>" prompt.

Bucky Keys: Holding down the Hyper, Super, and Meta Bucky keys, select the pane containing the desired image with the left mouse button [H S M C→ - - R]. All views on the stack will be described in the Lisp Interaction window. To exit the Lisp Inspector, type **:q** followed by a carriage return at the ">>" prompt.

```
[0: CLASS] #<Standard-Class IC::BLOCKED-ARRAY-MAPPED-IMAGE>

  Slots:
  [1: PROPERTY-LIST] (:3D-WORLD #<Alv 3d World 3D-WORLD #X416CB66>
  :IMAGE-TO-2D-TRANSFORM
   #<4X4-COORDINATE-TRANSFORM (NIL to NIL)
   #X416FDFE> :SUN-VECTOR (0.6082805 -0.1486257 0.7796828)
   :PATHNAME
  #P"/home/lippy/toolset/RCDE/alpha2/alv/alv-2-44-win.g0" :NAME "Alv-2-44-win" ...)
  [2: IC::INTERNAL-PROPERTY-LIST]
  (:XIMAGE 33083248 :INFERIORS NIL :TIME-TAG 4
   :TIME-OF-LAST-ACCESS 2934116255 :TIME-OF-CREATION 2934116255)
  [3: IC::X-DIM] 252
  [4: IC::Y-DIM] 245
  [5: IC::ELEMENT-TYPE] (UNSIGNED-BYTE 8)
  [6: IC::ELEMENT-SIZE] 8
  [7: IC::INITIAL-VALUE] 0
  [8: IC::WRITE-ACTION] IC::WRITE-LOCK-ERROR
  [9: IC::WRITE-LOCK] T
  [10: IC::X-MAP] #<Simple-Vector T 252 21BF8CE>
  [11: IC::Y-MAP] #<Simple-Vector T 245 21BFCCE>
  [12: IC::BLOCK-X-DIM] 128
  [13: IC::BLOCK-Y-DIM] -62
  [14: IC::PADDED-BLOCK-X-DIM] 128
  [15: IC::BLOCK-SIZE] 7936
  [16: IC::BLOCK-RADICES] NIL
  [17: IC::BLOCKS-WIDE] 2
  [18: IC::BLOCKS-HI] 4
  [19: IC::BITBLTABLE] T
  [20: IC::BITBLT-X-OFFSET] 0
  [21: IC::BITBLT-Y-OFFSET] 0
  [22: IC::IMAGE-STRUCT] #S(IC::ARRAY-IMAGE-STRUCT X-DIM 252 Y-DIM 245
  X-MAP #<Simple-Vector T 252 21BF8CE> Y-MAP #<Simple-Vector T 245 21BFCCE>
  IREF-FN #<Compiled-Function IC::ARRAY-IMAGE-UNSIGNED-8BIT-IREF 6C304E>
  ISET-FN #<Compiled-Function IC::ARRAY-IMAGE-UNSIGNED-8BIT-ISET 6C307E>
  SIMPLE-ARRAY #<Simple-Vector (UNSIGNED-BYTE 8) 63488 21B00BE>
  ARRAY #<Vector (UNSIGNED-BYTE 8) 63488 416BC3E> EXTRA1 NIL EXTRA2 NIL)
  [23: IC::IMAGE_STRUCT] #<Foreign-Pointer 1F89E04 (:POINTER IC::IMAGE_STRUCT)>
  [24: IC::ELEMENT-TYPE-CODE] 3
  [25: ARRAY] #<Vector (UNSIGNED-BYTE 8) 63488 416BC3E>
  >>
```

Figure 4.1: An Example of the Lisp Inspector

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Describe Image function from the Bucky menu. Following the Documentation Line prompt, select the pane containing the desired image. All views on the stack will be described in the Lisp Interaction window. To exit the Lisp Inspector, type **:q** followed by a carriage return at the ">>" prompt.

Refresh Pane – Refresh the display of the selected pane and return a Lisp pointer to the pane in the Lisp Interaction window.

Menu Bar: Select the Refresh Pane function from the Stacks menu. Following the Documentation Line prompt, select the desired pane. The pane will be refreshed and a pointer to the pane will appear in the Lisp Interaction window.

Bucky Keys: Select the desired pane with the middle mouse button [----→ -M-]. The pane will be refreshed and a pointer to the pane will appear in the Lisp Interaction window.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Describe Image function from the Bucky menu. Following the Documentation Line prompt, select the desired pane. The pane will be refreshed and a pointer to the pane will appear in the Lisp Interaction window.

Select – Select a pane for future operations, dropping a pointer to the pane's image or graph into the Lisp Interaction window.

Menu Bar: Select the Select function from the Stacks menu. Following the Documentation Line prompt, select the desired pane. The pane will be refreshed and a pointer to its contents will appear in the Lisp Interaction window.

Bucky Keys: Select the desired pane with the left mouse button [----→ L--]. The pane will be refreshed and a pointer to its contents will appear in the Lisp Interaction window.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Select function from the Bucky menu. Following the Documentation Line prompt, select the desired pane. The pane will be refreshed and a pointer to its contents will appear in the Lisp Interaction window.

Fwd Cycle – Cycle the selected stack in the forward direction (top of stack to bottom).

Menu Bar: Select the Fwd Cycle function from the Stacks menu. Following the Documentation Line prompt, select the desired pane. The pane will be cycled and a pointer to its contents will appear in the Lisp Interaction window.

Bucky Keys: Holding down the Meta and Control Bucky keys, select the pane containing the desired image with the left mouse button [--MC→ L--]. The pane will be cycled and a pointer to its contents will appear in the Lisp Interaction window.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Cycle Stack function from the Bucky menu. Following the Documentation Line prompt, select the desired pane. The pane will be cycled and a pointer to its contents will appear in the Lisp Interaction window.

Rev Cycle – Cycle the selected stack in the reverse direction (bottom of stack to top).

Menu Bar: Select the Rev Cycle function from the Stacks menu. Following the Documentation Line prompt, select the desired pane. The pane will be cycled and a pointer to its contents will appear in the Lisp Interaction window.

Bucky Keys: Holding down the Super, Meta, and Control Bucky keys, select the pane containing the desired image with the left mouse button [- S M C → L - -]. The pane will be cycled and a pointer to its contents will appear in the Lisp Interaction window.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Rev Cycle function from the Bucky menu. Following the Documentation Line prompt, select the desired pane. The pane will be cycled and a pointer to its contents will appear in the Lisp Interaction window.

Move View – Move the view from the top of one pane stack to another pane.

Menu Bar: Select the Move View function from the Stacks menu. Following the Documentation Line prompt, select the source and destination panes. The view will be popped from the top of the source stack and pushed onto the destination stack.

Bucky Keys: Holding down the Meta Bucky key, select the destination pane with the right mouse button [- - M - → - - R]. The source pane will default to the previously selected pane (the highlighted pane), which will be copied onto the destination stack.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Move To Here function from the Bucky menu. Following the Documentation Line prompt, select the source and destination panes. The view will be copied from the top of the source stack and pushed onto the destination stack.

Copy View – Copy a view from one pane stack to the top of another.

Menu Bar: Select the Copy View function from the Stacks menu. Following the Documentation Line prompt, select the source and destination panes. The view will be copied from the top of the source stack and pushed onto the destination stack.

Bucky Keys: Holding down the Meta Bucky key, select the destination pane with the middle mouse button [--M-→-M-]. The source pane will default to the previously selected pane (the highlighted pane), which will be copied onto the destination stack.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Copy To Here function from the Bucky menu. Following the Documentation Line prompt, select the source and destination panes. The view will be copied from the top of the source stack and pushed onto the destination stack.

Pop Stack – Remove the top view from the selected stack. The removed view is stored in an invisible temporary storage stack and can be restored (see Unkill Top).

Menu Bar: Select the Pop Stack function from the Stacks menu. Following the Documentation Line prompt, select the desired pane. The top view will be removed from the chosen stack.

Bucky Keys: Holding down the Meta and Control Bucky keys, select the desired pane with the middle mouse button [--M-→-M-]. The top view will be removed from the chosen stack.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Pop Stack function from the Bucky menu. Following the Documentation Line prompt, select the desired pane. The top view will be removed from the chosen stack.

Unkill Top – Restore the most recently popped view to the selected pane.

Menu Bar: Select the Unkill Top function from the Stacks menu. Following the Documentation Line prompt, select the destination pane. The most recently popped view is retrieved from the invisible temporary storage stack and pushed onto the selected pane.

Pop Expunge – Permanently remove the view from the top of the selected stack. The expunged object is permanently destroyed and can not be recovered.

Menu Bar: Select the Pop Expunge function from the Stacks menu. Following the Documentation Line prompt, select the desired pane. The top view will be permanently removed from the chosen stack.

Bucky Keys: Holding down the Meta and Control Bucky keys, select the desired pane with the right mouse button [--M-→--R]. The top view will be permanently removed from the chosen stack.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Expunge Top function from the Bucky menu. Following the Documentation Line prompt, select the desired pane. The top view will be removed from the chosen stack.

Copy Stack – Copy all views from one pane stack to the top of another. selected stack.

Menu Bar: Select the Copy Stack function from the Stacks menu. Following the Documentation Line prompt, select the source and destination panes. The entire source pane stack will be cleared of views, which will be pushed onto the destination pane stack.

Kill Stack – Remove all views from the selected stack. The removed views are stored in an invisible temporary storage stack and can be restored (see Unkill Stack).

Menu Bar: Select the Kill Stack function from the Stacks menu. Following the Documentation Line prompt, select the desired pane. The entire stack will be cleared of views. The killed stack views will be pushed together onto the temporary storage stack, so that they can be restored in their entirety using Unkill Stack.

Unkill Stack – Restore the most recently killed stack of views to the selected pane.

Menu Bar: Select the Unkill Stack function from the Stacks menu. Following the Documentation Line prompt, select the destination pane. The most recently killed stack will be restored to the selected pane. Multiple Unkill Stack operations will restore previously killed stacks. Unkill Stack.

Chapter 5

Geometric Image Transform Scenario

The user may be interested in multiple areas of multiple images, which can be accessed by zooming and scrolling. The Zoom commands (In and Out) use an interpolated zoom algorithm, while Fast Zoom uses pixel replication instead of interpolation. Related commands include % Reposition, which enables the user to move anywhere in a given image, and Recenter which moves the selected point to the center of the pane. As previously stated, the viewing area is divided up into panes. Each pane contains a stack of views. The top of the stack view is what the user sees in each pane. A view can contain wire frame models as well as imagery, so be sure to position the cursor on a section of imagery free of the models, otherwise the wrong function will be executed. The desired context is the image, not an object.

The scenarios listed below illustrate how to load a site model and/or imagery into a pane. The imagery is zoomed in, zoomed out, repositioned, scrolled about the screen and partitioned into an image chip via the window function.

1. First, load the ALV site model by selecting the Load Site Model from the I/O menu. A second submenu should appear.
2. Select the ALV field from the menu. After a few moments the ALV site model should appear on the screen, and the site model menu should disappear.
3. Select the Zoom In function from the Geom menu. Following the Documentation Line prompt, select the pane containing the desired image to zoom.
4. Select the view in the upper left pane, labeled **Alv-2-44-win**.
5. Following the Documentation Line prompt, select the destination pane for the zoomed image; choose the view in the lower left pane, labeled **Alv-Oblique-Tower**. Note that

the selection point will become the zoom focus. An image twice the size of the original will appear in lower left corner of the pane.

6. Now visually inspect the stack of the lower left corner pane by simultaneously holding down the Meta and Control keys while left clicking the mouse key. You will observe that this pane contains three images as the a new zoomed image view was copied on the lower left pane.

7. Now let us use the Bucky keys to perform the same function. Move the cursor to the upper left corner pane and hold down the Control key and depress the middle mouse button. You will notice that the original has been replaced by the zoomed image if you inspect the upper left corner pane stack.

We will now re-zoom the upper left image to its original resolution. Move the cursor to the upper left corner pane and hold down the Control key and depress the right mouse button. You will notice that the zoomed out image has been replaced by the original image, if you inspect the upper left pane stack. Notice that if the function is selected by menu, you have the option to select a source and destination pane. When using the Bucky commands, the selected (highlighted) pane is used as the default source pane.

For optimal use of the % Reposition function, it is best to use the cursor in conjunction with the Bucky keys:

1. Hold down the Control key, press and hold the left mouse button, and move the cursor about in the pane. You will observe the image scrolling along with the cursor.

2. To place the image in the desired position, release the mouse button.

The Window command supplies an interactive interface for cutting one image out of another or viewing a zoomed in image with respect to the original image. We will first use the window function to view the same image in two resolutions.

1. Create an Image Windowing Tool by selecting the Window function from the Geom menu.

2. Then position the cursor in the lower left corner pane. The top pane should contain the original image and the bottom pane directly underneath it should contain the zoomed image.

3. Left mouse click in the zoomed image. A yellow box (the Window Tool) should appear in the both panes. Notice that as you move the cursor, the window object moves in both the zoomed pane and the original image pane. Drop the window by clicking the left mouse button in the pane.

4. Place the cursor on the upper horizontal window edge. The window edge should turn green. Then resize the box by holding down the middle mouse button and moving the horizontal edge upward or downward to change the size of the window. Let go of the mouse button to drop the window edge.

5. Note that you can use one of two move modes: 1) move an edge by pressing and holding the mouse button, and letting go when you are done, or 2) clicking (pressing and letting go immediately), moving the edge as needed, then clicking the same mouse button again.

6. You can move two edges of the window at the same time by placing the cursor near a corner so that two of the edges turn green.

7. Note that as you touch and resize the object in one pane, it turns green and is resized in the other pane, indicating that we have two views of the same window. Because the window is proportional to the image no matter what resolution, you can use this technique to locate expanded portions of a large image.

8. You can delete this Window object at any time by placing the cursor on the window edge until it turns green, holding the Hyper key, and pressing the middle mouse button. (Note that the Documentation line reveals that this function is Delete).

You can also use the Windowing Tool to cut out a portion of an image.

1. Use the Bucky keys this time to create a new Window Tool: hold down the Hyper Bucky key and click with the left mouse button in the lower right pane. Click with the left mouse button to place the window, then adjust the size of the tool as before.

2. Place the cursor on the window box edge until the edge turns green, then press the right mouse button. The window will crop the image to the dimensions of the outlined in the box.

Another scenario allows the user to scroll and view several images at different resolution levels. First, load an image into the upper left pane using the procedures listed below:

1. Select the Load Image function from the I/O menu. A popup menu should appear.

2. Left click in the field labeled Directory and type **$CMEHOME/alv/** followed by a carriage return. (The trailing slash is optional).

3. Next, left click in the Pathname field and type **alv-3-42.g0** followed by a carriage return. The Format field should read **IU-TEST-BED-IMAGE-FILE**. If you do not see this message in the format field, rekey in the directory and pathname entry, ensuring that there are no errors and that entries have been accepted by the system. If the path name is incorrect, the field should say something like "File not found".

4. Then push the Load Image button at the bottom of the menu. Following the Documentation Line prompt, select the destination pane for the image being loaded; choose the view in the upper left pane. The image should appear in the pane.

Now copy the views between adjacent panes by the following procedures.

1. Place the cursor in the upper left pane and depress the left mouse button. Notice that this pane now has a highlighted border indicating it has been selected.

2. Next, move the cursor to the pane where you want the image to be copied; choose the lower left pane. While pressing the Meta key, hit the middle mouse button. The image will be copied into that pane.

3. Release the Meta key and left mouse click to select the pane.

4. Hold down the Control key and click the middle mouse button; the image should be zoomed.

5. Release the mouse key and Meta key, then select the upper left pane again.

6. Move the cursor to the pane where you want the next image to be copied; choose the upper right pane.

7. Depress the Meta key and hit the middle mouse button. The image will be copied into that pane.

8. Select another pane. Hold down the Control key and depress the right mouse key twice. The image should have gotten smaller indicating that you "zoomed out".

9. Now reposition the "zoomed out" image so that it is in the middle of the screen. To do this, position the cursor on the zoomed out image, and depress the Control key and hold it down. Now move the mouse cursor; you will find that the image moves with the cursor. When you are satisfied with the position of the image within the pane, release the Control key.

Now that we have copies of the same image in three different panes at different resolutions, we will tandem the images. Tandem implies that one image view is a master while the other is slave. Thus, when a window is moved in a master it is also position in the slave. We will make the zoomed out image the master and the other two images the slave.

1. Select the Tandem function from the Misc menu, and pin the resulting menu to the screen.

2. Select the Set Tandem View function from the Tandem menu. The Documentation Line will prompt you to select a master view.

3. Select the image with the smallest resolution (upper right) with the left mouse button. The Documentation Line will ask you to pick a slave view.

4. Move the cursor to the pane that contains the original image (upper left) and select it with a left mouse click.

5. Select the Set Tandem View function from the pinned Tandem menu. The documentation window will prompt you to select a master view.

6. Select the image with the lowest resolution with the left mouse button. The documentation window will ask you to pick a slave view.

7. Move the cursor to the pane that contains the zoomed image (lower left) and select it via a left mouse click.

8. Now go to the Tandem menu and select Tandem On.

9. Create a window to view the tandem images. Place the cursor on Geom and select it by left mouse click.

10. Create an Image Windowing Tool by selecting the Window command from the Geom submenu.

11. Select Window via left mouse click then position the cursor in the lower left corner pane. A yellow box should appear in the pane. Notice that as you move the cursor, the window object moves in the image pane.

12. You can resize the window by placing the cursor on the upper horizontal window edge. The window edge should turn green.

13. Now you can resize the box by holding down the middle mouse button and moving the horizontal edge upward or downward, thus changing the length of the window. You can resize the width of the window by placing the cursor on the left or right vertical edge and depressing and holding the middle mouse button. Make sure the edge turns green before you move the cursor while holding down the middle mouse button. Releasing the middle mouse button will set the width of the window. Notice how the vertical edges and horizontal edges can be positioned to resize the shape of the window. You have now created a window object.

14. Each object has a feature set. Move the cursor to the View horizontal menu bar and select View by left mouse clicking on the View menu item.

15. A second submenu will appear. Move the cursor to the menu item named Feature Sets and left mouse click. You will be prompted to select an image.

16. Select the image in the upper right hand corner.

17. Another pop up menu will appear. There are three horizontal fields labeled PRES, SENS, and SEL. PRES indicates that the feature set is present in the view. SENS indicates that the feature set is mouse sensitive in the view. SEL indicates that the feature set is selected for object creation in the current world. Since you have created a 2-D feature, move the mouse cursor to the PRES field and left mouse click on it. A yellow box will appear.

18. Move the cursor to the box edge. You will notice that it turned green indicating you have selected it. Now move the mouse cursor back to the menu and mouse click on the SENS field. Move the cursor back to the box. You have desensitized the window to mouse selection. When you place the cursor on the PRES or SEL, you hide or turn on the box for view.

19. In a similar manner, go to the pane that that does not have a window outline visible and make its feature set visible and mouse selectable as previously illustrated. Close the view menu and feature set menu choices.

20. Now select the window in the smallest image window view by placing the cursor on the window outline and left mouse click.

21. Notice that you can move the window around the screen as you move the cursor. The window goes out of view sometimes in the larger images. A way to remedy this situation is via the Bucky key command TScroll, which copies regions of the larger image and displays those portions within the pane as the window moves. Thus you always capture the position of the window in the larger images that occupy more space than a pane.

22. To invoke TScroll place the mouse cursor on the window edge until it turns green.

23. Then depress the Control key and left mouse click. As you move the cursor the window will follow, repainting the screen with the proper image region in the panes where the imagery exceeds the pane size.

24. To "drop" the object, simply left mouse click twice or depress Hyper and right mouse click once.

In summary, this chapter has presented a simple scenario to load imagery, then manipulate it geometrically using zooming, scrolling, and windowing operations.

Chapter 6

Geometric View Transforms

While most of the operations in the previous chapters manipulate image data, the functions in this chapter apply to the view as a whole. They perform geometric operations on the view, including:

Recenter	%Reposition
Zoom in	Zoom out
Fast Zoom in	Fast Zoom out
Mirror X	Mirror Y
Rotate CW	Rotate CCW
Scale Rotate	Window

Recenter – Move the view to put the selected point at the center of the pane. If this function is accessed through the Bucky keys, dragging the mouse with the button pressed causes the view to scroll interactively.

Menu Bar: Select the Recenter function from the Geom menu. Following the Documentation Line prompt, select the desired center point; the view will be repositioned to center this point in the pane.

Bucky Keys: Holding down the Meta Bucky key, select the desired center point with the left mouse button [--M-→ L--]; the view will be repositioned to center this point in the pane. Dragging the mouse with the button pressed causes the view to scroll interactively.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Recenter function from the Bucky menu. Following the Documentation Line prompt, select the desired center point; the view will be repositioned to center this point in the pane.

% Reposition – Recenter the view based on the percentage of the cursor's position in the pane.

This function is useful for panning around a large image without having to zoom. For example, if the image is much larger than the pane, you can move to the top left corner of the image by selecting the top left corner of the pane. If this function is accessed through the Bucky keys, dragging the mouse with the button pressed causes the pane to scroll interactively.

Menu Bar: Select the % Reposition function from the Geom menu. Following the Documentation Line prompt, select a point in the pane to reposition the view based on percentage.

Bucky Keys: Holding down the Control Bucky key, select a point in the pane to reposition the view with the left mouse button [---C→L--]. Dragging the mouse with the button pressed causes the view to scroll interactively.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the % Reposition function from the Bucky menu. Following the Documentation Line prompt, select a point in the pane to reposition the view based on percentage.

Zoom In – Enlarge the view by a factor of two, using pixel interpolation.

Menu Bar: Select the Zoom In function from the Geom menu. Following the Documentation Line prompt, select a point on the source pane to be the zoom focus, then select the destination pane. The view will be enlarged by a factor of two.

Bucky Keys: Holding down the Control Bucky key, select a point in the view to be the zoom focus with the middle mouse button [---C→-M-]. The view will be enlarged by a factor of two.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Zoom In function from the Bucky menu. Following the Documentation Line prompt, select a point on the source pane to be the zoom focus, then select the destination pane. The view will be enlarged by a factor of two.

Zoom Out – Reduce the view by a factor of two, using Gaussian blur as anti-aliasing filter prior to subsampling.

Menu Bar: Select the Zoom Out function from the Geom menu. Following the Documentation Line prompt, select a point on the source pane to be the zoom focus, then select the destination pane. The view will be reduced by a factor of two.

Bucky Keys: Holding down the Control Bucky key, select a point in the view to be the zoom focus with the right mouse button [---C→--R]. The view will be enlarged by a factor of two.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Zoom Out function from the Bucky menu. Following the Documentation Line prompt, select a point on the source pane to be the zoom focus, then select the destination pane. The view will be enlarged by a factor of two.

Fast Zoom In – In-place zoom in by pixel replication, without interpolation.

Menu Bar: Select the Fast Zoom In function from the Geom menu. Following the Documentation Line prompt, select a point on the view to be the zoom focus. The view will be enlarged by a factor of two.

Bucky Keys: Holding down the Super and Meta Bucky keys, select a point in the view to be the zoom focus with the middle mouse button [- - - C → - M -]. The view will be enlarged by a factor of two.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Fast Zoom In function from the Bucky menu. Following the Documentation Line prompt, select a point on the pane to be the zoom focus. The view will be enlarged by a factor of two.

Fast Zoom Out – In-place zoom out without antialiasing filter.

Menu Bar: Select the Fast Zoom Out function from the Geom menu. Following the Documentation Line prompt, select a point on the view to be the zoom focus. The view will be enlarged by a factor of two.

Bucky Keys: Holding down the Super and Meta Bucky keys, select a point in the view to be the zoom focus with the right mouse button [- - - C → - - R]. The view will be enlarged by a factor of two.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Fast Zoom In function from the Bucky menu. Following the Documentation Line prompt, select a point on the pane to be the zoom focus. The view will be enlarged by a factor of two.

Mirror X – Reverse the view left to right about the selected axis.

Menu Bar: Select the Mirror X function from the Geom menu. Following the Documentation Line prompts, select a point on the source pane to determine the mirror axis, then select the destination pane. The view will be rotated about a vertical axis.

Bucky Keys: Holding down the Hyper and Super Bucky keys, select a point in the view to be mirror axis with the left mouse button [H S - - → L - -]. The view will be rotated about a vertical axis.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Flip X function from the Bucky menu. Following the Documentation Line prompts, select a point on the source pane to determine the mirror axis, then select the destination pane. The view will be rotated about a vertical axis.

Mirror Y – Reverse the view top to bottom about the selected axis.

Menu Bar: Select the Mirror Y function from the Geom menu. Following the Documentation Line prompts, select a point on the source pane to determine the mirror axis, then select the destination pane. The view will be rotated about a horizontal axis.

Bucky Keys: Holding down the Hyper and Super Bucky keys, select a point in the view to be mirror axis with the left mouse button [H S - - → L - -]. The view will be rotated about a horizontal axis.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Flip Y function from the Bucky menu. Following the Documentation Line prompts, select a point on the source pane to determine the mirror axis, then select the destination pane. The view will be rotated about a horizontal axis.

Rotate CW – Rotate the view 90° in the clockwise direction about the selected point.

Menu Bar: Select the Rotate CW function from the Geom menu. Following the Documentation Line prompts, select a point on the source pane to determine the rotation point, then select the destination pane. The view will be rotated 90° in the clockwise direction about the selected point.

Bucky Keys: Holding down the Hyper and Super Bucky keys, select a point in the view to be mirror axis with the left mouse button [H S - - → - - R]. The view will be rotated 90° in the clockwise direction about the selected point.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Rotate function from the Bucky menu. Following the Documentation Line prompts, select a point on the source pane to determine the rotation point, then select the destination pane. The view will be rotated 90° in the clockwise direction about the selected point.

Rotate CCW – Rotate the view 90° in the counter-clockwise direction about the selected point.

Menu Bar: Select the Rotate CCW function from the Geom menu. Following the Documentation Line prompts, select a point on the source pane to determine the rotation point, then select the destination pane. The view will be rotated 90° in the counter-clockwise direction about the selected point.

Scale Rotate – Scale the X and Y dimensions of an view by independent scale factors, then rotate by a specified number of degrees.

Menu Bar: Select the Scale Rotate function from the Geom menu, which invokes a pop-up menu for specifying the view, x scale, y scale, degrees of rotation (clockwise), and pixel intensity value of the background. Default values are taken from the last time Scale Rotate was invoked. On the first invocation, default values are $x\ scale = 1$, $y\ scale = 1$, and rotation = 0, making it an identity operation. Scales with magnitude less than one shrink the view, while those with magnitude greater than one magnify the view. Negative scale values cause axis reversal. For eight-bit images, the background (the area vacated after the view has been rotated) pixel intensity can be set to an integer between 0 (black) and 255 (white).

The following items appear on the Scale Rotate popup menu:

- Status - This field displays information and prompts that are relevant to the operation.
- Image - To set a view for the operation, select the button to the right of the **IMAGE:** label, then select the desired pane. The image ID should be displayed on the button.
- X scale - Enter the scale factor for the X dimension.
- Y scale - Enter the scale factor for the Y dimension.
- CW Rotate - Enter the amount of rotation in degrees.
- Background - When a view with a rectangular image is rotated, it is bounded by a larger rectangle. The four empty triangular areas that result from this bounding rectangle can be set to a gray scale level between 0 and $2^n - 1$, where n is the number of bits per pixel. The default is zero (black).
- DOIT - Push this button to perform the operation. Following the Documentation Line prompt, select a destination pane for the scaled, rotated view.

Window – Cut out a subimage from the given view.

This function creates an Image Window tool, which is really a 2-D object placed in the view for performing a specialized set of window operations. Refer to Chapter 12 for more details of using this object.

The Image Window tool can be created three ways:

Menu Bar: Select the Window function from the Geom menu. Following the Documentation Line prompt, select the view to place the Image Window tool, which will be created at the selected point.

Bucky Keys: Holding down the Hyper Bucky key, select the point to place the Image Window tool with the left mouse button [H---→L--].

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Window function from the Bucky menu. Following the Documentation Line prompt, select the view to place the Image Window tool, which will be created at the selected point.

When the tool is created, a green window outline should appear on the pane. Move the mouse cursor and you will notice that the newly created window follows the movement of the cursor. Click the left mouse button to place the window. The window outline should turn yellow. You can resize the window by placing the cursor on the a horizontal or vertical yellow line until it turns green. Note the status bar – there are three choices:

Left: Move Middle: Move Edges Right: Make Window

Hold down the middle mouse button and move the edge to resize the window. Release the mouse button when you have finished. You must resize the horizontal and vertical edges separately. To move the box about the window, simply place the cursor on the box outline until the box turns green. Left mouse click to pick the box. the box should turn green and should follow the mouse movement. To deselect the box outline, left mouse click again.

If a parent view is duplicated, manipulated, and moved to another pane, a window outline should be seen in all other views of that scene, since the Window is a 2-D object in that view. For example, this allows one to view zoomed views with respect to the original view.

To crop an image to the size of the window, select the window by placing the cursor on the box outline. When the box turns green, depress the right mouse button. An image the size of the box outline will be produced in the pane. Note that the view objects outside the window still appear after the windowing operation.

If the Tandem function (Chapter 10) is set up properly, the Tandem Scroll (TScroll) is enabled from the Bucky keys for the Image Window. Holding down the Control Bucky key and pressing the left mouse button while the window is selected will scroll the tandem views. Note that the view is scrolled behind the window, which remains fixed in the pane.

Note — If you have created a window in a view and the window does not appear in other panes of the same scene, it is probably because that feature set of the newly created view has not been enabled (refer to Chapter 10 for details). To do this, select the Feature Sets command from the View menu. The Documentation Line will prompt you to select a view, which invokes a new panel. Select the Sel fields with the left mouse button until the window appears in the desired pane.

Chapter 7

Arithmetic Image Transform Scenarios

The next scenario illustrates the usefulness of arithmetic functions in image exploitation and processing. These functions include boolean operators, such as union(or), intersection(and), exclusive or, Complement, and thresholding (clipping to a binary value within specified limits), 2-D fast fourier transforms, pixel data conversions (truncate, round, convert to float, change pixel integer bit size), linear transform the image in the form $ax + b$, add two images, subtract two images, or linear combine two images.

The rules for Arithmetic operations are:

- The dimensions of the smallest image dimensions (x and y direction) will also be the size of the resultant image view operation in the event that the images are of different sizes.
- The lower left corner of the image defines the first pixel of each image.

It is easy to see how logical functions can be used to process images and extract information. Several scenarios illustrate the power of arithmetic operations in terms of image operators. In order to demonstrate the Boolean capabilities of the RCDE system, a reference image needs to be generated. Included on the distribution tape is a C program that writes a Sun rasterfile image to a file; the image is a pattern consisting of a small black square (50 x 50 pixels of intensity zero) within a larger block of white pixels (252 x 245 pixels of intensity 255). The generated test pattern is then loaded into a pane and several operations are boolean demonstrated.

1. To generate the test pattern, first compile and execute the C file

   ```
   $CMEHOME/lci/examples/square.c
   ```

This program generates a Sun rasterfile called **image_pattern.pix** with an image pattern to be used as an example.

2. Next, load the ALV site model by positioning the cursor on the the I/O menu choice and left mouse clicking. A second submenu should appear.

3. Position the cursor on the Load Site Model menu option and left mouse click.

4. Another Submenu will appear. Position the cursor over the ALV choice and left mouse click. After a few moments the ALV site model should have been loaded on the screen.

5. Bring the I/O menu option to the RCDE foreground by left mouse clicking in the I/O vertical pulldown menu near the I/O lettering.

6. Choose the Load Image menu option. Another text entry menu will appear. You will need to fill in the directory and the Name of the file to be loaded.

   (a) If the there is text in the directory field you may need to delete it. To delete the text, place the cursor to the right of the text. click on the arrow entry until the last letter of the directory is found and left mouse click. Continue to press the "Delete key on the keyboard until the text has disappeared.

   (b) Enter the directory where the **image_pattern.pix** file resides (you generated this with the executable code from the C source above). Be sure to end the directory path with a slash character "/".

   (c) Similarly, enter the filename in the Pathname entry field. You do not need to end the file name with a slash character "/".

   (d) Place the mouse cursor on the Load image button and left mouse click. Choose the upper right hand pane as the destination of the test pattern.

As an example of a boolean operation, let us exclusive-or an image with the test pattern you loaded. Recall that a binary one "xor'ed" with an unknown binary value, will invert the unknown binary value. Now imagine a pixel as a vector n-tuple of binary values of length n. If we exclusive or an 8 bit per pixel image of equal height and width with the test pattern image, the white shaded area (length 8 – all ones) when "xored" an image should invert the corresponding pixels, while the dark shaded pixels (length 8 – all zeros) should preserve the original value of the pixels.

1. Place the cursor on the Arith option and left mouse click. A Second submenu should appear. Place the cursor on the Boole option and left mouse click. A third menu choice should appear with the following choices.

   To perform the Union, Intersection, and Xor functions, select the function by placing the cursor on this menu option and left mouse click. Then pick a first image by placing the cursor in the pane of choice, followed by a left mouse click. Do the same for the

second image of choice. Then select a destination pane for the result. by left mouse clicking in the destination pane. The procedure for Complement is similar, except that only one image is involved.

- Union - Or the pixel values of two images.
- Intersection - and the pixel intensity values of two images.
- Xor - Exclusive or pixel intensity values of two images.
- Complement – Complement pixel intensity values an image.
- Threshold - Creates a binary image, having value 0 for all pixel intensities below the specified threshold and 1 for all other pixels. This function invokes a separate popup menu. Select the button marked **IMAGE:**, then select an image with the left mouse button. Use the left mouse button to move the slider to the desired image threshold, push the DOIT button, and select a result pane with the left mouse button.

2. Take the image view in the lower left pane and exclusive or it with the test pattern image in the upper right hand corner pane. Place the result in the lower right pane.

3. You should see a photographic negative of the image except for the 50 x 50 pixel area which has remained the same intensity as the original image.

4. Select the intersection function. Take the image view in the lower left pane and select the intersection of it with the test pattern image in the upper right hand corner pane. Place the result in the lower right pane.

5. The resultant image should look similar to the original image in the lower left pane except for the 50 x 50 pixel section which will appear as a black square. The black square represents an all zero pattern.

6. Select the Union function. Take the image view in the lower left pane and select the Union of it with the test pattern image in the upper right hand corner pane. Place the result in the lower right pane.

7. The resultant image should appear as an all white square except for the 50 x 50 pixel section on which will appear as the 50 x 50 pixel intensities of the original image.

8. Select the Complement function, using the view in the lower left pane as the source image. Place the result in the lower right pane. The resulting image should look like a photographic negative of the original image.

The following scenario will demonstrate how to add two images and subtract two images. This scenario is useful when processing multiple images (e.g., multispectral imagery).

1. Reload the ALV site model as described previously.

2. Select the Add function on the Arith menu.
3. Add the upper right view to the lower left view and place the result in lower right pane. (Follow the documentation line prompts if you are not sure of how to select source images and destination pane). Notice that the resultant image is about the same size as the upper right image view (in terms of width and height), but looks like the lower left view.
4. Select the Subtract function from the Arith function.
5. Choose the lower right view first, then the lower left, putting the result in the upper left pane.
6. Notice that the composite image before the subtraction did not even remotely resemble the image in the upper right hand corner, but yet we were successfully able to recover the upper right image.

Note: for the above addition/subtraction scenario to operate properly, it is imperative that at least one image view is in floating point format. Note that that the result of the addition and/or subtraction is a floating point image; you can tell this by selecting a pixel in the image, and noting that its pixel value in the Documentation Line is a floating point number. Integer images can be converted to floating point by utilizing the Float menu option.

Another scenario

1. Select the FFT function from the Arith submenu.
2. Select the image in the upper left hand corner pane to be the source of the FFT data.
3. Select the lower right hand corner pane for the destination.
4. Select the Magnitude Squared and the current pane as the destination(lower right pane).
5. Place the cursor within the pane, left mouse click and observe the documentation line.
6. Notice the magnitude of the pixel intensities. You will have to scale them.
7. Select the Linear Xform menu option and use the lower right pane as the source and destination pane.
8. A submenu will appear. Mouse click left in image button.
9. Enter the number 10000 in the scale factor field.
10. Next select the Round menu option and again select the lower right pane as the destination.

11. Select the center zero option from the menu choices.

12. Choose the lower right pane for the results.

13. The view you see should be the familiar 2-D FFT plot.

The above steps were necessary for you to view the FFT data in a meaningful way. To calculate an FFT you just need to invoke the FFT function with an image as its argument. The resultant complex image is placed on the stack.

Chapter 8

Arithmetic Image Transforms

These functions perform arithmetic operations on image intensities. The menu choices are:

Boole	Linear Transform
FFT	Clip
Float	Threshold
Fix	Add
Round	Subtract
Resize Pixel	Linear Combine
Negate	

Boole – Produce a submenu of boolean image operations.

These operations perform logical (Boolean) operations on one-bit images or multi-bit images. Each bit position within a pixel is treated on a bitwise basis with respect to Boolean operations. The Boolean operations include:

- Union - Perform the pixelwise union (logical OR) of two images, where a '1' is an indication of set membership (logical TRUE) and a '0' is an indication of set exclusion (logical FALSE). Rules for handling images of differing sizes are the same as for binary arithmetic operations (Missing bit positions are zero extended.)

  Select the Union function from the Boole submenu. Following the Documentation Line prompts, select the two source panes and the destination pane. The resulting image will appear in the destination pane.

- Intersect - Perform the pixelwise intersection (logical AND) of two images, where a '1' is an indication of set membership (logical TRUE) and a '0' is an indication

of set exclusion (logical FALSE). Rules for handling images of differing sizes are the same as for binary arithmetic operations.

Select the Intersect function from the Boole submenu. Following the Documentation Line prompts, select the two source panes and the destination pane. The resulting image will appear in the destination pane.

- Xor - Perform the pixelwise exclusive OR operation of two images, where '1' is logical TRUE and '0' is logical FALSE. Rules for handling images of differing sizes are the same as for binary arithmetic operations. The truth table for XOR is

a	b	$a \otimes b$
0	0	0
0	1	1
1	0	1
1	1	0

  Select the Xor function from the Boole submenu. Following the Documentation Line prompts, select the two source panes and the destination pane. The resulting image will appear in the destination pane.

- Complement - Pixelwise complement (logical NOT) of a single image. This function performs negation on a pixel by pixel basis by binary complementing each bit in the pixel width.

  Select the Complement function from the Boole submenu. Following the Documentation Line prompts, select the source and and destination panes. The resulting complemented image will appear in the destination pane.

- Threshold - Create a binary image from a numeric image by thresholding.

  Select the Threshold function from the Boole menu item to invoke a submenu. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. Next, adjust the threshold by moving the slider bar labeled Threshold. To perform the operation, push the button DOIT, then select the desired destination pane. The Status field displays relevant information and prompts throughout the process.

FFT – Produce a submenu of Fast Fourier Transform operations.

- FFT: - Perform a two-dimensional Fast Fourier Transform on an image. The resulting image will have a complex floating point data type. The width and length of the data set will be rounded upward to the nearest 2^n; those values that are added as a result of the upward rounding will be padded with zeros prior to the FFT operation. Select the FFT function from the FFT submenu. Following the Documentation Line prompts, select the source and destination panes. The resulting FFT is pushed onto the destination pane.

- Inverse FFT - Perform a two dimensional Inverse Fast Fourier Transform on an image. The input image will have a complex floating point data type. The width and length of the data set will be rounded upward to the nearest 2^n; those values that are added as a result of the upward rounding will be padded with zeros prior to the Inverse FFT operation. Select the Inverse FFT function from the FFT submenu. Following the Documentation Line prompts, select the source and destination panes. The resulting inverse FFT is pushed onto the destination pane.
- Magnitude - Produce a magnitude image from a complex image. Complex pixel values are defined as $a + bi$, where a is the real part and b is the imaginary part. The magnitude is defined as $\sqrt{a^2 + b^2}$. On a real image, this is the identity operation. Select the Magnitude function from the FFT submenu. Following the Documentation Line prompts, select the source and destination panes. The resulting magnitude image is pushed onto the destination pane.
- Magnitude Squared - Produce a magnitude squared image from a complex image. Complex pixel values are defined as $a + bi$, where a is the real part and b is the imaginary part. The magnitude squared is defined as $a^2 + b^2$. Select the Magnitude Squared function from the FFT submenu. Following the Documentation Line prompts, select the source and destination panes. The resulting magnitude image is pushed onto the destination pane.
- Center Zero Rotate the image toroidally (divide the image into quarters, and swap the blocks diagonally) to center the outermost pixels in the image. Usually, this function is performed on data that has undergone an FFT transformation to move the DC component to the center of the image. Select the Magnitude Squared function from the FFT submenu. Following the Documentation Line prompts, select the source and destination panes. The resulting image is pushed onto the destination pane.

Float – Create a new image whose pixel values are converted to floating point numbers.

Select the Float function from the Arith menu. Following the Documentation Line prompts, select the source and destination panes. The resulting floating point image is pushed onto the destination pane.

Fix – Create a new image whose pixel values are converted to integers by truncating values below the decimal point.

Select the Fix function from the Arith menu. Following the Documentation Line prompts, select the source and destination panes. The resulting fixed-representation image is pushed onto the destination pane.

Round – Coerce floating point pixel values to integers by adding 1 to the integer part if the decimal part is greater than or equal to 0.5; otherwise truncate the decimal part.

Select the Round function from the Arith menu. Following the Documentation Line prompts, select the source and destination panes. The resulting fixed-representation image is pushed onto the destination pane.

Resize Pixel – Create a new image having the same numerical values at all pixels (where possible), but having different pixel data types.

Select the Resize Pixel function from the Arith menu, which causes a popup window to appear. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. Select the desired new pixel size from the row of buttons; choices for number of bits/pixel are 1, 2, 4, 8, 16, and 32. The default value is the number used on the last invocation of this function. When decreasing the number of bits per pixel, intensities above the range of the new representation are set to the maximum value of the new representation. To perform the operation, push the button DOIT, then select the desired destination pane.

Negate – Perform a "photographic negative" operation.

On floating-point images, this corresponds to a numerical negation of each pixel. On fixed-point images, this corresponds to subtracting each pixel value from the maximum representable pixel value.

Select the Negate function from the Arith menu. Following the Documentation Line prompts, select the source and destination panes. The resulting fixed-representation image is pushed onto the destination pane.

Menu Bar: Select the Negate function from the Arith menu. Following the Documentation Line prompts, select the source and destination panes. The resulting negated image is pushed onto the destination pane.

Bucky Keys: Holding down the Hyper and Meta Bucky keys, select the destination pane with the right mouse button [H-M-→--R]. The resulting negated image is pushed onto the destination pane.

Bucky Menu: Invoke the Bucky menu with a right mouse click and select the Negate function from the Bucky menu. Following the Documentation Line prompts, select the source and destination panes. The resulting negated image is pushed onto the destination pane.

Linear Xform – Replace each pixel value x by $ax + b$, where a and b are user-specifiable scale and offset values, respectively.

Default values are taken from the last invocation of this function. On the first invocation, default values are $a = 1$ and $b = 0$, making this the identity operation.

Select the Linear Xform function from the Arith menu, which causes a popup window to appear. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. Enter a numeric value in the Scale Factor

and Offset fields representing a and b in the equation $ax + b$. X is the pixel value, a is scale factor and b is the offset. To perform the operation, push the button DOIT, then select the desired destination pane. The Status field displays relevant information and prompts throughout the process.

Clip – Perform a "hard-limit" function at each pixel location.

This function performs an identity mapping on pixels which fall between the limiting values, maps all of the pixels above the top limiting value to the top limiting value, and maps all of the pixels below the bottom limiting value to the bottom limiting value. Menu options specify the two limiting values. Default values are taken from the last invocation of this function. On the first invocation, default values are 0 and 255, making this the identity operator for 8-bit images.

Select the Clip function from the Arith menu, which causes a popup window to appear. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. Adjust the Lower Clip value and Upper Clip Value fields by moving the slider bars. To perform the operation, push the button DOIT, then select the desired destination pane. The Status field displays relevant information and prompts throughout the process.

Threshold – Create a binary image by setting each pixel below a threshold to the pixel value zero and those pixel values above the threshold to 1.

Select the Threshold function from the Arith menu, which causes a popup window to appear.

Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. Next, adjust the threshold by moving the slider bar labeled Threshold. To perform the operation, push the button DOIT, then select the desired destination pane. The Status field displays relevant information and prompts throughout the process.

Add – Add corresponding pixel intensities in two images.

Numeric overflows cause saturation at the maximum value of the pixel representation. If the pixel representations are different, the output image will have the "more expressive" of the two input images pixel representations, where floating point representations are considered "more expressive" than fixed point representations, and a fixed point representation is "more expressive" than other fixed point representations of fewer bits per pixel. If the images to be added are of differing sizes, the output will be of size $min(l_1, l_2)$ by $min(w_1, w_2)$, where l_i is the length of the ith image in pixels and w_i is the width of the ith image in pixels. The area added will be the area of overlap if the two images were put on top of each other with the bottom left corners coinciding.

Select the Add function from the Arith menu. Following the Documentation Line prompts, select the source and destination panes. The resulting sum image is pushed onto the destination pane.

Subtract – Subtract corresponding pixel intensities in two images.

Numeric underflows cause saturation at 0. If the pixel representations are different, the output image will have the "more expressive" of the two input images pixel representations, where floating point representations are considered "more expressive" than fixed point representations, and a fixed point representation is "more expressive" than other fixed point representations of fewer bits per pixel. If the images to be subtracted are of differing sizes, the output will be of size $min(l_1, l_2)$ by $min(w_1, w_2)$, where l_i is the length of the ith image in pixels and w_i is the width of the ith image in pixels. The area subtracted will be the area of overlap if the two images were put on top of each other with the bottom left corners coinciding.

Select the Subtract function from the Arith menu. Following the Documentation Line prompts, select the source and destination panes. The resulting difference image is pushed onto the destination pane.

Linear Combine – Weighted addition of two images, with user specifiable weights.

The initial defaults for this algorithm are 1 and -1, making this the subtraction operation. The rules for sizing and handling overflows and underflows are the same as for other binary arithmetic operations.

Select the Linear Combine function from the Arith menu, which causes a popup window to appear. Designate one of the source views by pushing the button labeled Image A, then selecting the pane containing the image. Repeat for Image B. Adjust the Factor A and Factor B fields by moving the labeled slider bars. Enter the Offset value by typing the desired value in the field. To perform the operation, push the button DOIT, then select the desired destination pane. The Status field displays relevant information and prompts throughout the process.

Chapter 9

Enhancement Image Transforms

This chapter describes a set of operations that "enhance" the visual display of an image by emphasizing either the contrast or the edges of the image. The functions include:

Contrast Stretch	Zero Cross Image
Diff of Gauss	Sobel Edges
Gauss Blur	Canny Edges
Zero Cross Overlay	

Contrast Stretch – Perform a pixelwise linear remap of existing intensity values to the full range of the pixel representation.

Pixel values outside the maximum range are mapped to the closest range extreme. This function is similar to the Linear Transform function discussed in Chapter 8 but is an interactive function. The gain and offset of each pixel is defined by the relation:

$$(gain) * (pixelvalue) + (offset)$$

This operation is similar to the Image Window operation since it performs the operations using a special tool, the view-hacking-object. Since more general capabilities of the RCDE objects will be presented further in Chapter 12, this discussion will concentrate on simple contrast manipulation.

Select the Contrast Stretch function from the Geom menu. Following the Documentation Line prompt, select the view to place the Image Window tool. The tool will be created at the selected point, appearing as a small box with the text designation

"View". This icon is an object having a set of associated methods, and is called a view-hacking-object.

A number of operations can be performed on the selected view using this object. For example, the Brt & Cont function allows interactive adjustment of the brightness and contrast of the view's appearance. Enter this mode by by moving the cursor into the object (the icon must turn green) and clicking the middle mouse button. The brightness (gain) and contrast(offset) will change as a linear function of mouse movement, using the following convention:

- Mouse vertically up - increase gain
- Mouse vertically down - decrease gain
- Mouse horizontally to the right - increase offset
- Mouse horizontally to the left - decrease offset
- Mouse movement in both directions - increase/decrease both parameters.

The Documentation Line will display brightness and contrast parameters interactively as the operation is performed. Note that the entire screen appears to be affected while the operation is in progress, but only the desired pane is changed when the operation is finished. This is because the colormap is directly manipulated for fast interactive feedback, but a new image is calculated and the colormap restored when the operation is completed.

To see the other operations available from the view-hacking-object, press the Hyper and Control Bucky keys while the cursor is on the object (it must be highlighted), then click the right mouse button. This invokes the object Bucky menu, which can not be used to execute operations but serves as documentation for the Bucky key commands available on the object.

Diff of Gauss (Difference of Gaussians) – Subtract two smoothed versions of the same image.

This operator functions as an edge detector, which approximates the Laplacian of a Gaussian ($\nabla^2 G$) operator for an appropriate selection of parameters. The menu parameters specify the pyramid levels for the images to be differenced. This convolution uses the technique of hierarchical convolution developed by Peter Burt of SRI[1]. Each level in the hierarchy approximately doubles the standard deviation of the kernel. The coefficients of the kernel are approximately:

$$\text{coeff}_{x,y} = \frac{\exp(-.5(\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}))}{2\pi\sigma_x\sigma_y}$$

[1]P. Burt, "Fast Filter Transforms for Image Processing," *Computer Graphics and Image Processing*, No. 16, pp. 20-51, 1981)

Default values are taken from the last invocation of this function. Initial invocation defaults are level 0 and level 1.

Select the Diff of Gauss function from the Enhance menu, which causes a popup window to appear. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. Next, select one of nine levels in each of the fields marked Level-1 and Level-2, which determine the size of the Gaussian kernels used (a larger number implies a larger standard deviation). Keeping Level 2 larger than Level 1 avoids reversing the image. To perform the operation, push the button DOIT, then select the desired destination pane. The Status field displays relevant information and prompts throughout the process.

Gauss Blur – Convolve with a Gaussian to get a low-pass filtered version of the image.

Select the Gauss blur function from the Enhance menu, which causes a popup window to appear. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. Next, select one of nine levels in the field marked Level, which determine the size of the Gaussian kernel used (a larger number implies a larger standard deviation). To perform the operation, push the button DOIT, then select the desired destination pane. The Status field displays relevant information and prompts throughout the process.

Zero Cross Overlay – Overlay the image with "zero crossings" of the image. The crossing level parameter is the pixel intensity to be taken as zero.

Select the Zero Cross Overlay function from the Enhance menu, which causes a popup window to appear. Designate the desired view by pushing the button labeled View, then selecting the desired pane. Note that the pane must contain a view, not just an image, because the zero crossings are marked as an overlay. Next, enter the pixel intensity to be used as zero by typing a value into the Crossing Level field. This threshold value will determine the boundary state for pixel intensity (e.g., equal to or above threshold a binary one, below the threshold a binary zero). To perform the operation, push the button DOIT, and the results of the operation will be displayed in the selected view. The Status field displays relevant information and prompts throughout the process.

Zero Cross Image – Compute zero crossings of an image, returning a Boolean image showing where the zero crossings occur. The crossing level parameter is the pixel intensity to be taken as zero.

Select the Zero Cross Image function from the Enhance menu, which causes a popup window to appear. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. Next, enter the pixel intensity to be used as zero by typing a value into the Crossing Level field. This threshold value will determine the boundary state for pixel intensity (e.g., equal to or above threshold a binary one, below the threshold a binary zero). To perform the operation, push

the button DOIT, then select the desired destination pane. The Status field displays relevant information and prompts throughout the process.

Sobel Edges – Perform edge detection on a given image using the Sobel operator.

The Sobel Edge Operator approximates the two directional derivatives at each point in the image. The pixelwise output of the Sobel operator is the gradient magnitude

$$\sqrt{\frac{\partial I}{\partial x}^2 + \frac{\partial I}{\partial y}^2},$$

where I is the image intensity at a point.

Select the Sobel Edges function from the Enhance menu. Following the Documentation Line prompts, select the source and destination panes. The resulting edge image is pushed onto the destination pane.

Canny Edges – Perform edge detection on a given image using the a Canny edge operator.

This edge operator is useful for feature extraction and site modeling as it produces a binary image whose line curves are of a uniform width.

Select the Canny Edges function from the Enhance menu, which causes a popup window to appear. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. Three sliders control the algorithm parameters. As implemented, the Canny algorithm uses the following steps:

1. **Smoothing** is the first step of the algorithm, and is accomplished with a Gaussian blur. The kernel of the Gaussian is determined by the Filt Size slider on the popup panel.
2. The **Gradient** operator is then applied to the smoothed image. Since ∇ is a vector operator, the result can be expressed as magnitude and direction images. The Mag button displays the magnitude image as an intermediate result.
3. The third stage is NMS, which performs Non-Maximum Suppression. Its results are fed into a **Threshold** operation to determine which pixels of the strength image are selected as edges. Both the Lo Thresh and Hi Thresh sliders are used to specify the high and low thresholds. The resulting edge image can be viewed using the Edges button. Selecting the Strengths button produces an image that displays values of the magnitude image for every edge pixel identified in the final edge image, setting all other pixels to zero.

Chapter 10

Graph, I/O, and Miscellaneous Operations

This chapter summarizes the Graph, I/O, and Miscellaneous menus, which provide assorted graphing capabilities, input/output operations, and system level services.

10.1 The Graph Menu

The functions performed by this menu selection display image intensity information in a graphical form.

X Slice – Plot pixel intensity as a function of position for a given horizontal line in the image.

The X Slice and Y Slice functions are also called line amplitude profiles. Only the portion of the image visible in the pane will be graphed.

Menu Bar: Select the X Slice function from the Graph menu. After the Documentation Line prompts to select a horizontal line to graph, select any pixel in the desired image line. When the Documentation Line prompts for a destination pane, select the desired pane. The plot of pixel intensity versus pixel position in the horizontal direction is then displayed in the destination pane.

Bucky Keys: Depress the Super Bucky key and select any pixel in the desired image line within the desired image using the left mouse button. When the Documentation Line prompts for a destination pane, select the desired pane. The plot of pixel intensity versus pixel position in the horizontal direction is then displayed in the destination pane.

Bucky Menu: Invoke the Bucky menu with a right mouse click on the image, then select the X Slice function from the Bucky menu options. After the Documentation Line prompts to select a horizontal line to graph, select any pixel in the desired image line. When the Documentation Line prompts for a destination pane, select the desired pane. The plot of pixel intensity versus pixel position in the horizontal direction is then displayed in the destination pane.

Y Slice – Plot pixel intensity as a function of position for a given vertical line in the image.

The Y Slice and Y Slice functions are also called line amplitude profiles. Only the portion of the image visible in the pane will be graphed.

Menu Bar: Select the Y Slice function from the Graph menu. After the Documentation Line prompts to select a vertical line to graph, select any pixel in the desired image line. When the Documentation Line prompts for a destination pane, select the desired pane. The plot of pixel intensity versus pixel position in the vertical direction is then displayed in the destination pane.

Bucky Keys: Depress the Super Bucky key and select any pixel in the desired image line within the desired image using the middle mouse button. When the Documentation Line prompts for a destination pane, select the desired pane. The plot of pixel intensity versus pixel position in the vertical direction is then displayed in the destination pane.

Bucky Menu: Invoke the Bucky menu with a right mouse click on the image, then select the Y Slice function from the Bucky menu options. After the Documentation Line prompts to select a vertical line to graph, select any pixel in the desired image line. When the Documentation Line prompts for a destination pane, select the desired pane. The plot of pixel intensity versus pixel position in the vertical direction is then displayed in the destination pane.

Histogram – Plot a histogram of the pixel intensities of the selected image.

The menu version of the command allows extra control of the histogram display. The histogram is calculated for the entire image, not just the part visible in the pane. Note that the image representation in the pane is used for the histogram; image operations such as zooming may affect the statistics of the resulting histogram.

Menu Bar: Select the Histogram function from the Graph menu, which causes a popup window to appear. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. To display the histogram, push the button labeled Window, then select the desired pane for the result.

The amount of the histogram to be displayed can be controlled by moving the slider bars labeled Left start value, Right end value, and Tail factor. The left

mouse button is typically used to manipulate the sliders. If the Tail factor is zero, a histogram will be displayed between the grey level values specified by Left start value and Right end value. The Tail factor is a number between 0 and .5. It represents the fraction of data points that will be eliminated from each end of the histogram display. Note that adjusting the Tail factor affects the start and end values, but not vice versa.

The resolution of the histogram can be controlled by the Number of Buckets field, which specifies the number of bins for the histogram calculation. After adjusting the histogram display parameters, the histogram can be redisplayed by pushing the Window button and selecting a result pane. The Status field displays relevant information throughout the process.

Bucky Keys: Depress the Super Bucky key and select the desired image using the right mouse button. (Be sure not to select another object inadvertently.) When the Documentation Line prompts for a destination pane, select the desired pane. The histogram will be plotted without benefit of the tail control features.

Bucky Menu: Invoke the Bucky menu with a right mouse click on the image, then select the Histogram function from the Bucky menu options. Following the Documentation Line prompts, select the image source and destination panes. The histogram will be plotted without benefit of the tail control features.

10.2 The I/O Menu

This section describes the RCDE system utilities for loading/saving feature sets, images, and site models to/from the RCDE environment. Hardcopy and directory capabilities are also described.

Save Image – Select an image and save it to a file.

Select the Save Image function from the I/O menu, which causes a popup window to appear. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. Next, specify the UNIX path and filename for storing the image in the Pathname field. Finally, press the Save Image button to perform the operation. The Status field displays relevant information throughout the process.

Load Image – Specify an image file and load it into a pane.

Select the Load Image function from the I/O menu, which causes a popup window to appear. Two fields are available for entering file names: the Directory field and the Pathname field. Enter the directory path where the desired file resides in the Directory field, then enter the file name in the Pathname field. Although it is not required, entering a carriage return after entering the filename will cause the RCDE to search

the directory for the file, read the image header, and determine what type of image is to be read. Press the Load Image button to perform the operation; following the Documentation Line prompt, select the desired destination pane.

If the image format is unrecognized, the other fields in the panel may be used to specify the appropriate image parameters. The Format field allows the image file format to be specified directly, and the Element Type sets the pixel type. Each is a multiple-choice popup menu that is selected with the left mouse button. The remaining fields allow numeric image parameters to be specified, which is useful for images without file headers. The fields are X Dimension, Y Dimension, Block X Dimension, Block Y Dimension, and Header Length. Finally, the Status field displays relevant information throughout the image loading process.

Dired Image – Search a directory for image files, select an image from the directory list, and load it into a pane.

Select the Dired Image (DIRectory EDit) function from the I/O menu, which causes a popup menu to appear. Type a directory path name into the Filter field, which is used to search the directory for files. Enter a carriage return or push the Filter button to initiate the file search. The search will terminate with the last slash ("/"), so end the path with a slash to go into the deepest directory.

The directories and files matching the search string will appear in the Directories and Files scrolling lists. The Directory list may be used to move around the directory structure by double-clicking with the left mouse button on one of the entries. Single click on one of the file entries to select an image for input and display its full path name in the Selection field. Press the Load Image button to load the image; following the Documentation Line prompt, select the desired destination pane.

To display the image header fields (e.g., Block X Dimension), push the Read Header button. It is not necessary to read the header information to load an image. Note that the image header fields in this panel are read-only, and do not affect the header values themselves. All of the text fields in this panel can be scrolled using the arrow keys on the keypad.

Alternately, double-click on one of the file entries to completely select the image for input, which includes reading the image header, displaying the image header values in the fields below (e.g., Block X Dimension), and displaying the full path name of the selection in the Selection field.

Hardcopy Image – Select an image and send it to a hardcopy device, such as a Kodak or PostScript printer.

This command writes a PostScript file to the temporary file directory[1], which can saved for later printing. Note that the PRINTER environment variable is used to specify the print device.

[1] Defined by ***host-default-mapped-file-allocation-directory***.

Select the Hardcopy Image function from the I/O menu, which causes a popup window to appear. Designate the desired image by pushing the button labeled Image, then selecting the pane containing the image. The image label and its pixel type should be displayed in the Title and Status fields, respectively. The Width and Height fields specify the size of the image on the page, and the Orient field specifies the orientation of the output. Finally, press the Hardcopy Image button to perform the operation. The Status field displays relevant information throughout the process.

Save Feature Sets – Selectively save feature sets from a view.

Select the Save Feature Sets function from the I/O menu. Following the Documentation Line prompt, select a view containing the desired feature sets. A popup panel will be invoked. The top of the panel will contain a row of buttons, one for each feature set in the selected view. Press any number of buttons to select the feature sets to be saved. As the buttons are pressed, the Count field will display the sum of the objects within the selected feature sets. Likewise, the Status field will display how many feature sets have been selected. Specify the UNIX path and filename for storing the image in the Pathname field, then press the Save Feature Sets button to save the selected feature sets.

Load Feature Sets – Load a feature set into a pane from a file.

Select the Load Feature Sets function from the I/O menu, which causes a popup window to appear. Two fields are available for entering file names: the Directory field and the Pathname field. Enter the directory path where the desired file resides in the Directory field, then enter the file name in the Pathname field. Although it is not required, entering a carriage return after entering the filename will initiate a search for the file, displaying information about the file in the Date, Author, and File Length fields of the panel. Press the Load Feature Sets button to perform the operation; following the Documentation Line prompt, select the desired destination view. The Status field displays relevant information throughout the process.

Save Site Model – Save a site model to a file.

Select the Save Site Model function from the I/O menu, which causes a popup window to appear. Designate the desired model by pushing the button labeled Push Me, then selecting the pane containing the model. The Count field will display the number of objects found in the selected model. Next, specify the UNIX path and filename for storing the site in the Pathname field. Finally, press the Save Site button to perform the operation. The Status field displays relevant information throughout the process.

Load Site Model – Load a site model from a file.

Select the Load Site Model function from the I/O menu, which causes a small popup panel to appear. The popup presents a list of site models [2] that can be loaded directly

[2] The list is stored in the variable ***cme-site-model-list***, which the user can augment.

by selecting the button, plus the entries None and Other. The None entry dismisses the panel and aborts the Load Site Model operation, while the Other command invokes a second popup menu for loading site models from a file. Two fields are available for entering file names: the Directory field and the Pathname field. Enter the directory path where the desired file resides in the Directory field, then enter the file name in the Pathname field. Although it is not required, entering a carriage return after entering the filename will initiate a search for the file, displaying information about the file in the Date, Author, and File Length fields of the panel. Press the Load Site Model button to perform the operation; following the Documentation Line prompt, select the desired destination pane. The Status field displays relevant information throughout the process.

10.3 The Misc Menu

This section contains a collection of miscellaneous functions for controlling the user interface.

Tandem – Synchronize multiple views for performing simultaneous operations.

The Tandem functions allows the user to slave two or more views to a master view so that operations performed in the master view are simultaneously executed in each of the slave views.Select the Tandem function from the Misc menu, which causes a pullright menu to appear. The menu, which can be pinned, presents a list of functions for implementing the Tandem mode:

- Tandem On – Enable the tandem function.
- Tandem Off – Disable the tandem function.
- Set Tandem View – Establish a master/slave relationship between two views. The default position used for the slave operation is the 3-D point selected in the master view.
- Clear Tandem View – Clear the master/slave relationship.
- Show Tandem View – Indicate (via a flashing square) which views are associated.
- Set Tandem Pane – Establish a master/slave relationship between two panes.
- Clear Tandem Pane – Clear the master/slave relationship.
- Show Tandem Pane – Indicate (via a flashing square) which panes are associated.

For the Tandem On and Tandem Off functions, select the appropriate menu item. For the Set Tandem View, Clear Tandem View, Set Tandem Pane, and Clear Tandem Pane commands, select the appropriate menu item, select the master pane, then select the slave pane. For the Show Tandem View and Show Tandem Pane commands, select the appropriate menu item, then select the master pane.

CME Frames – Select existing RCDE frame or create/delete a frame.

Select the Pick CME Frame function from the Misc menu, which causes a small popup panel to appear. The popup presents a list of active CME frames and the buttons Make Frame and Kill Frame. Selecting one of the active frame entries causes the frame to be opened, if necessary, and brought to the foreground. The Make Frame and Kill Frame buttons allow the user to kill existing frames and create new ones, respectively.

The Make Frame command causes a popup panel to appear, which presents a list of parameters for specifying how the new frame will appear:

- N panes horiz/vert - These two buttons together determine the tiling of panes within the frame.
- Window Type - Determines the type of graphics operations that can be performed within the pane. An X11 pane is used for normal RCDE site models. XGL frames are required to render model scenes (using the XGL library).
- Screen - Specifies on which screen the new frame will appear, if the workstation has multiple screens and/or screen types. These buttons are determined by the system configuration; for example, a two monitor system having one 8-bit color screen and one one-bit monochrome screen might have the options 8-bit :0.1 | 1-bit :0.0.
- Frame Name - Labels the frame on its window decoration with the specified string. The default label describes the window type and tiling arrangement.

If no defaults appear, a value must be chosen for all parameters except Frame Name. Press the DOIT button to perform the operation.

Globals Control panel – Set a number of user interface display options.

Select the Globals Control Panel function from the Misc menu, which causes a popup panel to appear. The popup present a list of parameters affecting the RCDE user interface:

- Enable Multi Colors - Enables the display of wire-frame objects in colors other than the default yellow. Object colors are set in the object parameters menu. Default is No.
- Continuous Object Menu Update - Enables continuous updating of values on the object parameters menu, such as the position of the selected point or the object dimensions. Default is Yes.
- Screen for Menu Popup - Designates the screen on which the Bucky menus are drawn. Options are Mouse, the screen on which the object resides, or Other, a second display (if any). Default is Mouse.
- Tandem Enabled - Defines the default state of the Tandem command. If the Tandem is enabled, Tandem mode defaults to Tandem On. Default is No.

- Sensitive Arcs and Faces - Enables the mode where individual arcs and faces of polyhedral objects are sensitive to mouse selection. Default is No. In addition to setting this toggle, the vertices of the object must be opened with the Open Verts [H - M - → L - -] command.
- Undo History Length - Determines the number of object manipulation operations that can be undone with the Undo operation [H - - C → L - -]. Default is 5.

Import Public Color Map – Copy the colors added to the public color map into the RCDE color map.

Select the Import Public Color Map function from the Misc menu; if enough space is available, the public color map colors will be transferred to the RCDE color map in an attempt to minimize colormap flashing.

Reset Color Map – Reset the RCDE color map to default values.

Select the Reset Color Map function from the Misc menu; the color map will be reset to its default state, which is defined by the variable **ic::*color-map-grey-ramp-defaults***.

Clear All Panes – Clear all panes in the selected frame.

Select the Clear All Panes function from the Misc menu, then select the desired frame. All panes in the selected frame will be cleared.

Command Apropos – Initiate a key word search on Documentation Line parameter or sub-menu command.

Select the Command Apropos function from the Misc menu. The Lisp Interaction Window will enter a new mode; enter a text string, and the RCDE menu documentation strings will be searched for a command that contains that string. A list of commands associated with the keyword will be displayed in the Lisp interaction window. Enter **q** at the prompt to quit.

Set Emacs Window – Instruct the RCDE to direct keystrokes to a given Emacs window. RCDE frames or Menu Bar.

When keystrokes are typed while the mouse cursor is on an RCDE frame, the RCDE can direct them to the Lisp Interaction Window. This command enables that mode.

Select the Set Emacs Window function from the Misc menu. Move the cursor to the Lisp prompt, then type **T** and a carriage return. Keystrokes should then be directed to that window whenever the mouse cursor is positioned over an RCDE frame. Note that this function is disabled, as are all RCDE functions, whenever Lisp enters the debugger.

Quit CME – Exit the RCDE.

Select the Quit CME function from the Misc menu, which causes a popup menu to appear. Select Yes to exit the RCDE, or No to abort the exit.

Chapter 11

Introduction to 3-D Modeling

In the preceding chapters, the reader was introduced to the RCDE user interface in the context of manipulating images. The true power of the RCDE is in its ability to use imagery to create and manipulate 3-D site models. The next several chapters are dedicated to describing the RCDE's three-dimensional capabilities. The RCDE data structures are sufficiently complex, however, that this document will only summarize some important details. For more information, use the *RCDE Programmer's Reference Manual* together with the Lisp Listener to explore the RCDE representations directly.

11.1 Representations

As introduced in Chapter 2, the RCDE 3-D world is composed of the following components:

- **Images** – a set of data derived from some sensing event.
- **Views** – fundamental RCDE data structures that allow the user to observe the 3-D world on the display.
- **Feature Sets** – a collection of objects, logically grouped for some purpose (e.g., common manipulation).
- **Illumination Model** – a representation of site illumination, used for shading the faces of rendered polyhedral objects, and for computing values based on the sun position (e.g., building height from shadows).
- **Objects** – 2-D, 3-D, or Tool visual representations that populate or help build a site model.

- **Camera Model** – a view component that specifies a transformation from 3-D world coordinates to 2-D view coordinates.
- **Terrain Model** – terrain data associated with a given 3-D world. Objects are often manipulated relative to the ground level.
- **Site** – a collection of feature sets in a common geographic area. All feature sets for a given site share a common local coordinate system referenced to the site.

Image operations have already been addressed in earlier chapters. This chapter will present views, feature sets, and the illumination model. Subsequent chapters describe the RCDE cameras, terrain, and site models.

11.2 Views

The VIEW is a fundamental component of the 3D-WORLD, which unifies the RCDE data structures for viewing on the display. The world contains objects that are viewed through a simple camera, as illustrated in Figure 11.1. The camera transforms the world coordinates (x, y, z) to image coordinates (u, v). Objects in the scene are mapped to the image plane (film plane) of the camera as shown in the drawing.

The major components of a view are related as illustrated in Figure 11.2. The white boxes represent data objects, which include model elements (e.g., house, image), groupings of objects (e.g., feature sets), and transformations. The shaded boxes denote coordinate systems, which are numerical values stored in a separate container object.

The solid lines in the figure represent association (a "has-a" relation) between object classes, while dotted lines indicate a logical connection between the objects and related coordinate representations. The arrows on the solid lines indicate some interaction between the objects in the direction of the arrows, but do not imply a specific implementation. The small, filled circles indicate a multiplicity for a given relation. The association notation is commonly used for describing object and database systems, while the arrow notation has been added to imply object interaction.

From the left side of the figure, we can see that a view (class OBJECT-VIEW) is associated with a number of 3-D feature sets, 2-D feature sets, tool feature sets, images, and view stacks. Each type of feature set (3-D, 2-D, tool) is associated with a number of objects. Each object has its own internal coordinate system (object coordinates) and a transform matrix that maps object coordinates to some other coordinate system.

For example, follow the figure from the bottom, moving toward the right. Three-D models (e.g., houses) have their own object-centered coordinate system and a 4X4-TRANSFORM that maps object coordinates into a set of 3-D world coordinates.

The 3-D world coordinates are stored by a 3D-WORLD object as a local vertical coordinate system (LVCS), whose origin is defined relative to a given site. A 3-D world has additional transformations to convert the LVCS to a universally recognized World Coordinate system,

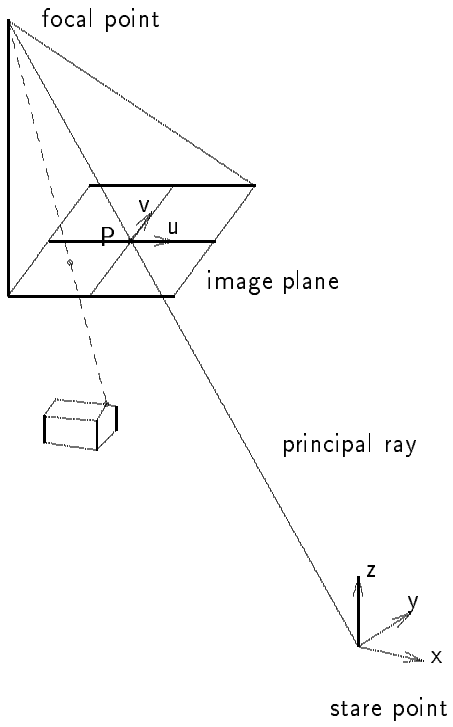


Figure 11.1: A simple 3-D world. The world is viewed through a camera aimed through the principal point **P**, which lies at the intersection of the image plane and the principal ray. The stare point is the intersection of the principal ray with the terrain model, or the $z = 0$ plane if the site does not have a terrain model. Objects in the world are projected onto the image plane through the camera, which maps world coordinates *(x, y, z)* to image coordinates *(u,v)*.

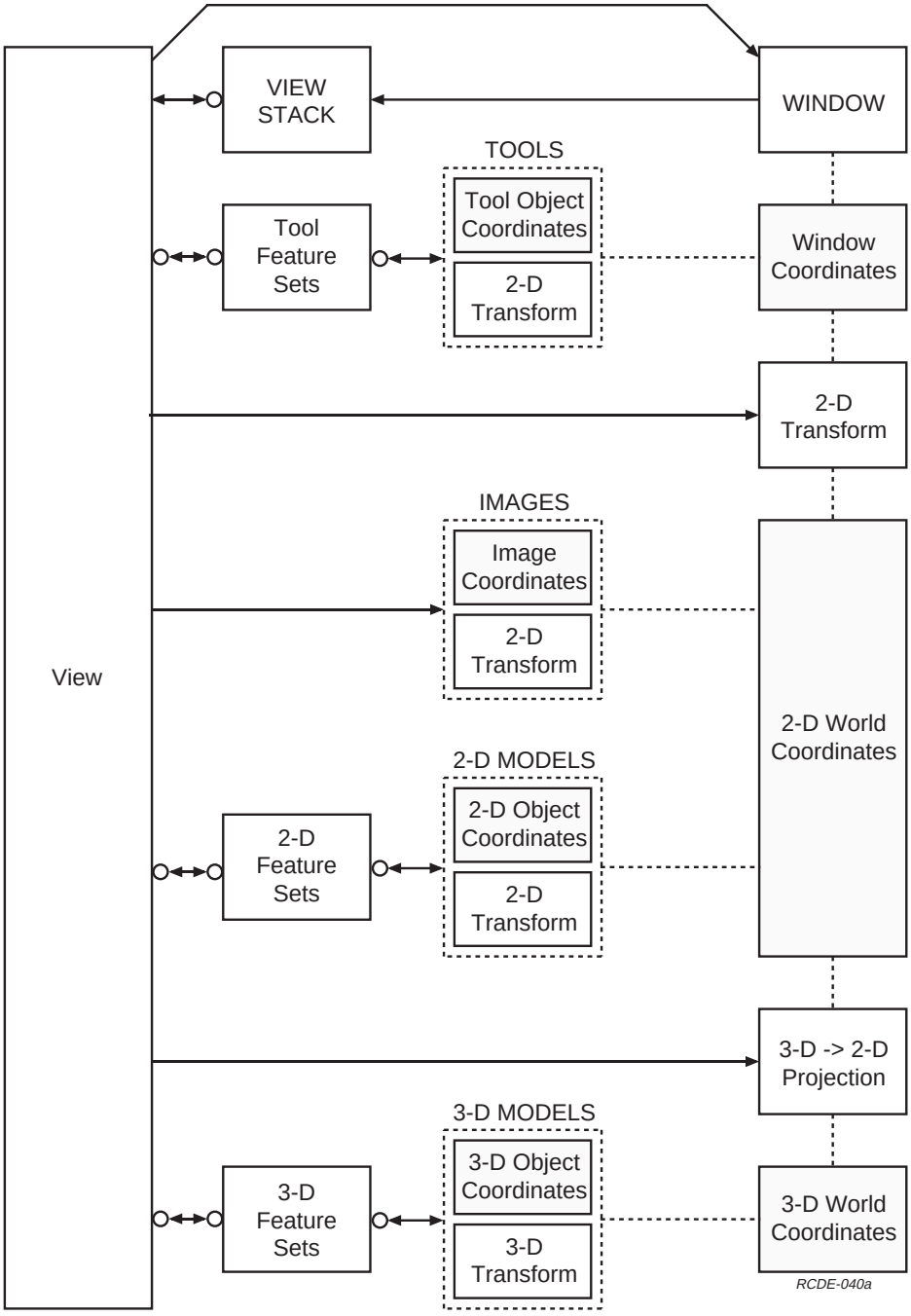


Figure 11.2: RCDE Data Representations

such as Universal Transverse Mercator (UTM), Latitude-Longitude-Elevation, or geocentric coordinates.

The view is also associated with a sensor (camera) model that maps 3-D world coordinates to 2-D world coordinates. The camera model is labeled "3-D→2-D Projection" in the figure. Chapter 13 presents more details of the camera model,

Views are also composed of images and 2-D feature sets, which contain 2-D objects. Both have transformations to transform local object or image coordinates to 2-D world coordinates, in common with the output of the camera model. The 2-D world coordinate values are stored by a 2D-WORLD object.

The 2-D world coordinates are then mapped into window (pane) coordinates by another 2-D transformation ("2-D Transform" in the figure). This transformation is necessary for translations of the 2-D data within the window such as panning and zooming. The view may also contain Tool objects, which are designed to remain fixed in the pane even when the underlying data is panned or zoomed. All types of objects are therefore mapped to window coordinates in a similar way. A WINDOW-WORLD object is responsible for storing the window coordinate data. To complete the figure, note that the window has a stack associated with it, which contains a number of views.

11.3 The Transforms Menu

The Transforms menu supplies operations to create different types of views, either from scratch or from an existing view.

Exact Copy – Create a copy of an existing view with the same camera but without an image.

This function copies all elements of a view except the image into the specified pane: the feature sets, the association with terrain and illumination models, and the camera model. To copy a view entirely, including the image, use the Stacks→Copy View command.

Select the Exact Copy function from the Transforms menu. Following the Documentation Line prompt, select the source and destination panes. The view will be copied from the top of the source stack and pushed onto the destination stack.

New View Transform on Image – Create a new view using the specified image.

Select the New View Transform on Image function from the Transforms menu. Following the Documentation Line prompt, select the desired pane. A popup menu will appear for selecting the 3d-world into which the view will be inserted. If you wish to use an existing 3d-world, select the desired world from the list. If no world exists or you wish to make a new one, select the Make 3d World button, which invokes another popup panel. Enter a string to label the new world, then press Do It to create the new world and new view transform. The new camera has a default set of parameters and

places the stare point so that it appears in the bottom left corner of the image. The transform is adjustable.

Make Transform Adjustable – Allow the selected view's camera to be adjusted.

Some camera models, such as in the ALV model, have purposely been created so that the camera model cannot be adjusted. This command allows the user to make the transform adjustable so that the camera parameters can be changed.

Select the Make Transform Adjustable function from the Transforms menu. Following the Documentation Line prompt, select the desired pane. The existing view transform will be made adjustable, and a camera icon and a stare point object will be created in the frame. Note that you may have to refresh the pane if the objects do not appear in the pane.

Sun View – Create a new view with an orthographic camera model at the sun position.

To create a new view from the sun position, a SUN RAY object must be present and set in the view to define the illumination model. The new view will not have an associated image.

Select the Sun View function from the Transforms menu. Following the Documentation Line prompt, select the center point of the new transform (where the stare point will be located), then select the destination pane. The new view will be created in the selected pane.

Vertical View – Create a new nadir (vertical) view.

The new view will use a perspective camera with the same focal length and camera elevation as the specified view. The new view will not have an associated image.

Select the Vertical View function from the Transforms menu. Following the Documentation Line prompt, select the center point of the new transform (where the stare point will be located), then select the destination pane. The new view will be created in the selected pane.

11.4 Feature Sets

Feature sets are a mechanism to group modeling objects within the RCDE. Objects in a common feature set can be made visible or invisible as a group in a single view. They can also be *sensitized* or *desensitized* as a whole; sensitized objects can be selected by the mouse, whereas desensitized objects ignore mouse events.

When an object is created, it is automatically placed into the *selected* feature set. Though the user may not have explicitly created or selected a feature set, both actions are automatically executed when a view is created.

In addition to resection and displaying selected groups of objects, feature sets are useful for other purposes, such as saving groups of objects to file and reading them back into the RCDE.

11.5 The View Menu

This menu allows access to a view's feature sets and controls its rendering parameters.

Feature Sets – Configure or create feature sets for a given view.

Select the Feature Sets function from the View menu, which causes a popup panel to appear. Designate the desired view by pushing the button labeled View, then selecting the pane containing the view. A list of the feature sets that currently exist in the world will be displayed in the panel. If no image nor a view exists in a pane, then no feature sets will appear. Once an image is loaded, the pane will contain at least one tool and one 2-D feature set. After a view transform is created, the pane will contain at least one 3-D feature set.

Each feature set is associated with three buttons:

Pres – Toggles whether the objects in a given feature set are displayed (present) in the view.

Sens – Toggles whether the objects in a given feature set can be modified (sensitive) in the view.

Sel – Selects the feature set into which newly created objects will be placed. Only one each of 3-D, 2-D, and tool (window) feature sets can be selected at one time.

To create a new feature set, press the button for the desired type at the bottom of the panel. A new feature set will be created and added to the list.

3D Feature Sets – Configure or create 3-D feature sets for a given view.

Select the 3D Feature Sets function from the View menu, which causes a popup panel to appear. Designate the desired view by pushing the button labeled View, then selecting the pane containing the view. A list of the 3-D feature sets that currently exist in the world will be displayed in the panel, and the name of that world appears in the 3D World field. If a view transform has not yet been created for that view, no world nor feature sets will be listed.

Each feature set is associated with three buttons:

Pres – Toggles whether the objects in a given feature set are displayed (present) in the view.

Sens – Toggles whether the objects in a given feature set can be modified (sensitive) in the view.

Sel – Selects the feature set into which newly created objects will be placed. Only one 3-D feature set can be selected at one time.

The Other Views toggle controls the scope of the feature set buttons:

None – Changes made to the feature set buttons affect the selected view only.

3d World – Changes made to the feature set buttons affect all views in the selected world.

Frame – Changes made to the feature set buttons affect all views in the same frame having the selected world.

To create a new 3-D feature set, press the New3DFS button at the bottom of the panel. A new feature set will be created and added to the list.

Render – Control the rendering parameters for a view.

Select the Render function from the View menu, which causes a popup panel to appear. Designate the desired view by pushing the button labeled Pick View, then selecting the pane containing the view. Specify which type of rendering is desired using the Mode menu:

Wire Frame – Draw the site objects as wire frame objects, performing hidden line removal within each object but not between objects. This mode, the default, is available in all types of frames.

HLR Wire Frame – Draw the site objects as wire frame objects, performing hidden line removal within each object and between objects. This mode is enabled only in an XGL frame.

HSR Shaded Surface – Draw the site objects using shaded surfaces, performing hidden surface removal within each object and between objects. This mode is enabled only in an XGL frame.

Select which mode of refresh is desired using the When button. Choosing Done implies that wire frames are used to indicate the objects' positions when they are being moved, and the scene is re-rendered whenever the object is dropped. Continuous allows the rendering process to be performed continuously as objects are modified. The Status field displays relevant information throughout the process.

The following steps summarize the process of rendering an existing site model:

1. Create a new XGL frame using Misc→CME Frames→Make Frame.
2. Copy an existing view into the new frame.
3. Create a SUN RAY object in the scene using the Create Object menu.
4. Define the illumination model by using the Set Sun Ray command [--M-→L--] on the Sun Ray.

5. Invoke the rendering panel (View→Render), choose a view, then choose the HSR Shaded Surface mode.
6. If the view is not rendered automatically, refresh the pane [- - - - → - M -]. The view should be displayed in rendered form. To adjust the illumination of a shaded rendering, move the vertices of the Sun Ray object, Set Sun Ray again, then refresh the pane.

Chapter 12

The RCDE Objects

The RCDE contains a set of object primitives that are useful for building site models and annotating imagery. This chapter first presents a simple object manipulation scenario, then details the specific properties of each object and its functionality.

12.1 Site Model Update Scenario

A camera's image plane contains a picture that is a 2-D representation of a given scene. The purpose of the RCDE is to use imagery to support building site models. That is, the user must reconstruct the 3-D model given one or more 2-D views of a scene plus some additional information such as the camera parameters or ground truth with image correspondences. The following scenario illustrates the process of adding model objects to an existing scene and modifying the scene.

12.1.1 Creating a New View

To get a feel for the 3-D modeling system, the user can copy an existing view into a pane which does not contain an image. This is not normally done when creating a site model, but is instructive.

1. Select the CME Frames option from the Misc menu, which invokes a small popup menu.
2. Select Make Frame, which invokes another panel.
3. Select two horizontal panes and one vertical pane, and other options if you wish (be sure that one button is pressed in each row), then select DOIT to make a new frame. After the frame is created, you can reduce its size using standard window manager functions.

4. Invoke a new version of the ALV frame from the Load Site Model command on the I/O menu.

5. Select Exact Copy on the Transforms menu, then left click on the bottom left ALV pane as the source pane and the left pane of your new frame as the destination pane. You have created a new view of the ALV site in your frame, but the view does not have an image.

6. If you wish, you can use the % Reposition or Recenter commands to center the buildings in the pane, so that your left pane looks like the left pane in Figure 12.1 (without the ball-shaped object). Note how the box with the word "Camera" returns to a fixed point in the pane; it is a tool for manipulating the camera parameters.

7. Now create a second view in the right-hand pane using Exact Copy on the Transforms menu again, and perform the rest of the scenario using the right pane.

12.1.2 Instantiating Model Object Primitives

The next step is to place model objects in the scene. The available objects can be found in the Create Object menu, and placed in the scene by selecting the object and choosing its position in the pane.

Each object serves as a separate context, and as such has a set of functions associated with that context. These functions are available only through the Bucky keys in combination with the mouse. Chapter 12 details the functions available for each object. Although each object has functions unique to it, many functions are common to all objects. For example, the House object has resizing options that are common to all 3-D objects (such as X-Y Scale), but also allows the user to adjust the angle of the roof using Roof Pitch [-S-C→--R].

1. Select Superquadric from the Create Object menu and choose a point near the botton of the pane [1] to place the object. Note how the Documentation Line changes – the object name appears at the far left, followed by the Bucky key documentation. After the object appears in the scene, it remains selected for manipulation; the default Bucky function is Move UV@Z, allowing you to move the object around while keeping its Z position constant. To drop the object, either left click again or choose Drop/Obj from the Bucky menus [H---→--R].

2. Experiment with changing the size of the object using:

   - Z Size [--M-→--R] – Scale the Z dimension of the object with respect to the selected point, keeping that point fixed in site coordinates.

[1] By default, objects are placed onto the terrain. Therefore, select the point to place the object so that a ray from the viewpoint intersects the terrain, otherwise an error will be generated.

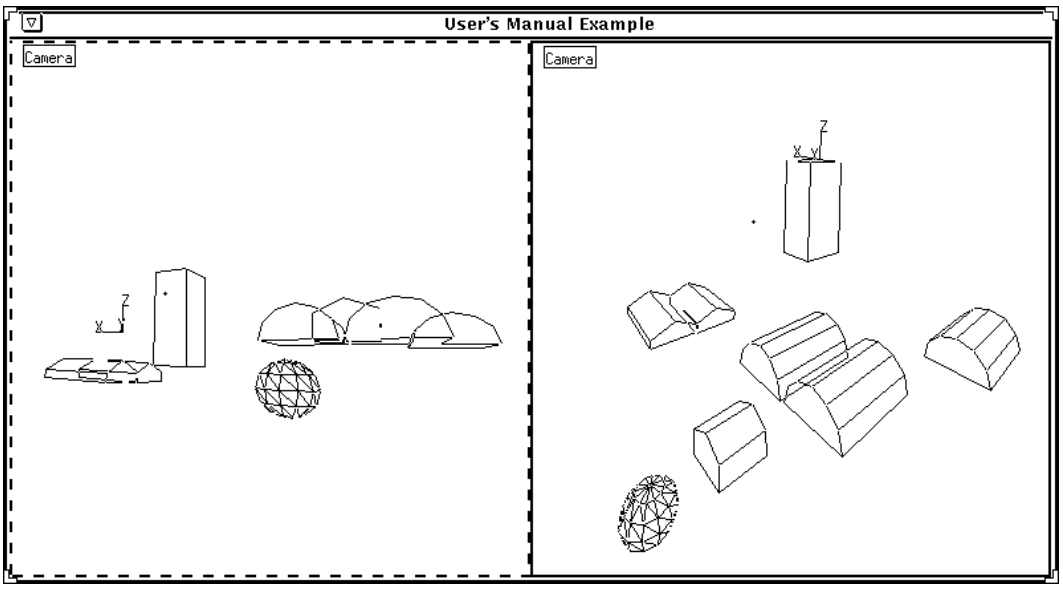


Figure 12.1: A simple scene of model objects. The left pane shows the objects as created, while the right pane shows the same scene from another camera.

- Rotate/Scale [--M-→-M-] – Change the object's size and orientation, keeping the selected point fixed in site coordinates. Left-right mouse motions perform one action, up-down motions perform the other action.
- Move Z [----→-M-] – Move the object in the site coordinate Z direction, keeping the selected point fixed in site coordinate X and Y.

12.1.3 Moving Model Objects

To understand how objects are moved in the scene, it is necessary to understand the camera model viewing the scene. Figure 11.1 shows that the U and V axes lie in the image plane and describe coordinates in 2-D. The W axis, not shown in the figure, is perpendicular to the image plane (orthogonal to u and v). Many of the operations for moving model objects are performed with respect to these axes. Some examples include:

- Move UV @ Z – Move the selected point on the object in the image (UV) plane while keeping the object's Z value (in site coordinates) constant.
- Move W – Move the object along a ray from camera's W axis (toward or away from the user in the given view).
- UV Roll – Rotate the object about the U and V axes, keeping the selected point fixed in site coordinates. Up-down motions roll the object around the U (horizontal) axis, while left-right mouse motions roll around the V axis (vertical).

More details regarding the RCDE camera model can be found in Chapter 13.

12.1.4 Adjusting the Model View

The camera parameters can be adjusted to view the model objects from another angle. The camera itself may be manipulated either from its icon (the small box with the word "Camera") or via the STARE POINT object (a particular type of (x, y, z) axis representation):

1. Move the cursor to the camera icon and invoke the Bucky menu panel [H--C→--R].
2. Similarly, invoke the Bucky menu for the stare-point.
3. Note the differences in the two menus. The camera icon allows you to adjust the camera parameters from camera-centered coordinates, while the stare point supplies a point in the view around which the camera can be rotated. The camera icon is a tool and remains fixed in the pane, while the stare point can be positioned in 3-D. In particular, the Az-Elev command allows the user to adjust both the azimuth and the elevation of the camera simultaneously, with respect to the stare point.
4. Move the cursor to highlight the stare point object, and invoke the Az-Elev command [-SM-→L--]. Move the cursor *slowly* to see the view change. After repositioning, your right pane may look something like the right pane of Figure 12.1.

12.2 Object Classification

The RCDE has several categories of objects, which are loosely grouped here for convenience.

12.2.1 2-D Objects

The following objects exist only in the image plane, associated with a given point in the image. They appear only in the pane where they are created, and as such are not part of the site model *per se*.

- **2d Composite** — An object that serves to group other 2-D objects, allowing them to be manipulated together. Note that this object does not have a wire frame representation *per se*.
- **2d Conj Pt** — A 2-D conjugate point object is used to establish a correspondence between 2-D points in images where there is no 3-D world (e.g., stereo correspondences).
- **2d Crosshair** — A cross-shaped object used to mark locations in the image plane.
- **2d Network** — A collection of line segments and vertices that can be manipulated as a group in the image plane.
- **2d Point** — A zero-dimensional object used to mark locations in the image plane.
- **2d Ribbon** — A pair of parallel connected line segments in the image plane.
- **2d Ruler** — An open 2-D curve used for performing mensuration in the image plane.
- **2d Text** — A text object typically used as a label in the image plane.
- **Closed 2d Curve** — A set of connected line segments, forming a closed region, in the image plane; commonly used to delineate region boundaries.
- **Open 2d Curve** — A set of connected line segments in the image plane.

12.2.2 3-D Feature Objects

The following objects, although not necessarily three-dimensional themselves, are associated with a given point in 3-space. They appear in all views of a given scene.

- **3d Composite** — An object that serves to group other 3-D objects, allowing them to be manipulated together. Note that this object does not have a wire frame representation *per se*.
- **3d Crosshair** — A cross-shaped object used to mark locations in 3-space.

- **3d Network** — A collection of line segments and vertices that can be manipulated as a group in 3-space, and whose vertices can be moved individually in 3-space.
- **3d Point** — A zero-dimensional object used to mark locations in 3-space.
- **3d Ribbon** — A pair of parallel open curves for representing lines of communication such as roads and rivers in a scene.
- **3d Ruler** — An open 3-D curve used for performing mensuration in 3-space.
- **3d Text** — A text object typically used as a label in 3-space.
- **Closed 3d Curve** — A set of connected 3-D line segments that form a closed region; commonly used to delineate 3-D boundaries.
- **DTM Mesh** — A rectangular grid of lines used to reveal the DTM data underlying a given site.
- **Open 3d Curve** — A set of connected line segments in 3-space.
- **Ribbon** — pairs of near parallel lines used for modeling roads, rivers, and other ribbon-like objects; each vertex may take a separate 3-D position.
- **Star** — A set of 3-D line segments sharing a common end point. This object exists in 3-space and is visible in all views.

12.2.3 3-D Face Objects

The following objects are fully three-dimensional and are associated with a given point in 3-space. They appear in all views of a given scene, and are capable of being shaded or having texture mapped to their faces.

- **Box** — A rectangular, prism-like 3-D object having three degrees of freedom (X, Y, and Z).
- **Cylinder** — A cylindrical 3-D object having two degrees of freedom (height and radius).
- **Extrusion** — A 3-D object whose base is a closed 3-D curve, used for extruding buildings with complex geometry.
- **House** — A box with two sloping roof surfaces in place of the upper face whose roof angle can be specified separately.
- **Quanset** — a half-cylindrical 3-D object having two degrees of freedom (height and radius); its canonical direction is flat side down.

- **Superellipse** — A closed 3-D object having two degrees of freedom that is described by the equation

$$\left(\frac{x}{a}\right)^m + \left(\frac{y}{b}\right)^m + \left(\frac{z}{c}\right)^n = 1$$

- **Superquadric** — A closed 3-D object having three degrees of freedom that is described by the equation

$$\left(\frac{x}{a}\right)^l + \left(\frac{y}{b}\right)^m + \left(\frac{z}{c}\right)^n = 1$$

12.2.4 Tool Objects

The following objects exist only in the two-dimensional window space. That is, they are attached to a given point in a pane, independent of image or model projections behind them. They are intended to remain fixed in the work space, and appear only in the pane where they are created.

- **Color Map Tool** — A rectangular object used to manipulate the global color map (pixel values); stationary in the pane.
- **Image Window** — A rectangular 2-D object used to manipulate rectangular regions of imagery.
- **North Arrow** — A linear object indicating the direction defined as North within the given view.
- **View Tool** — A rectangular pane object used to manipulate the grey-level values associated with a given view; stationary in the pane.
- **Window Text** — A text object that remains stationary in the pane.

12.2.5 Miscellaneous Objects

The following objects are mixed in functionality, attachment, and visibility.

- **Axis** — A set of orthogonal directed line segments that intersect at a point, used to represent coordinate frame orientation and location. This object exists in 3-space and is visible in all views.
- **Camera** — A wire frame object that illustrates the position of another view's camera within the current view. This object exists in 3-space but is only visible in the view where it was created.
- **Conj Pt** — A conjugate point object is used to establish a correspondence between a point in space and a location in one or more images. Therefore, one end of the object is attached to an image point, and the other end is attached to a 3-D point. Only the 3-D attachment point is displayed in other views.

- **Corner** — A wire frame object to indicate the location and direction of a trihedral corner in 3-space.
- **Scroll Bar** — A scroll bar object is automatically created in a pane whenever the image in a pane is larger than or scrolled outside of the pane. This object cannot be instantiated from the Create Object menu, but has a number of Bucky commands available to it.
- **Sun Ray** — A pair of 3-D line segments sharing a common end point; used to specify the direction of illumination. This object exists in 3-space and is visible in all views.

12.3 Methods on Objects

Because the RCDE is an object-oriented system, each object has a number of operations that are inherited from its predecessors. This section describes the current list of operations (culled from the system) that can be performed using the Bucky keys on the objects. Please also refer to Appendix A for a mapping of these functions to the objects. Some attempt has been made to organize the following methods from general to specific.

12.3.1 Basic Methods

The following methods can be performed on all objects.

- **Blank** — Temporarily remove the object from all views, until the pane is refreshed. Note that the object still exists and is still sensitive to selection.
- **Bucky Menu** — Invoke the Bucky menu for the selected object (to be used only as a mnemonic for objects).
- **Clone** — Create an identical copy of an object.
- **Delete** — Delete the object.
- **Drop/Obj** — Stop modifications to the selected object, and return a pointer to the object instance in the Lisp Interaction Window.
- **Edit Hier** — Edit the composite object hierarchy of the selected object.
- **Menu** — Bring up a panel to adjust object parameters menu for the selected object.
- **Redo** — Redo the last action. Note that the Undo History Length can be changed using the Globals Control Panel.
- **Undo** — Undo the last action. Note that the Undo History Length can be changed using the Globals Control Panel.

12.3.2 Common Methods

The following methods are common to many objects.

- **Rotate/Scale** — Change the object's size and orientation, keeping the selected point fixed in site coordinates. Left-right mouse motions perform one action, up-down motions perform the other action.
- **Scale** — Scale the object with respect to the selected point, keeping that point fixed in site coordinates.
- **XY Sizes** — Change the object's size with respect to site coordinate X and Y direction, keeping the selected point fixed in site coordinates. Left-right mouse motions perform one action, up-down motions perform the other action.

12.3.3 3-D Methods

The following additional methods may be performed in a 3-D world.

- **@Vertex** — Move the indicated vertex to the selected point. For conjugate point objects, move the 3-D part of the conjugate point to the selected vertex.
- **Az-Elev** — Rotate the object in two dimensions: left-right mouse motions rotate the object about the site Z axis, and up-down motions rotate the object about the object's X axis. If specialized to STARE POINT objects, mouse motions rotate the scene about the STARE POINT, keeping that point centered in the view.
- **Close Object** — Desensitize the selected object from mouse sensitivity, and open its superior for operations (for a simple object, enter Feature Set selection).
- **Drop W** — Move the object to the DTM along a ray from the camera's focal point through the selected point. The base of the object is placed on the DTM.
- **Drop Z** — Move the object along a ray from the camera's W axis (toward or away from the user in the given view). This is valid for 3-D objects and cameras.
- **Move UV on DTM** — Move the object in the image (UV) plane while keeping the object on the Digital Terrain Model (DTM).
- **Move UV@Z** — Move the selected point on the object in the image (UV) plane while keeping the object's Z value (in site coordinates) constant. This is valid for 3-D objects and cameras.
- **Move W** — Move the object to the DTM along a ray from the camera's focal point through the selected point. The base of the object is placed on the DTM.

- **Move Z** — Move the object in the site coordinate Z direction, keeping the selected point fixed in site coordinate X and Y. This is valid for 3-D objects and cameras.
- **Re Orient** — Reset to Canonical orientation, in which the object coordinate system is aligned with the site coordinate system. The selected point is kept in site coordinates.
- **Sun Z** — Adjust the object's height using its shadow, which is a ray drawn from the selected point to the DTM in the direction specified by the illumination model. The illumination model must be defined for the specified view (see Set Sun Ray
- **Taper Rate** — Change the object's shape to be tapered around an axis that is parallel to the object's Z axis and intersects the object's center. Neither the Z dimension nor the site position of the object are affected, and the object's cross-sectional shape is unchanged.
- **UV-Roll** — Rotate the object about the U and V axes, keeping the selected point fixed in site coordinates. Up-down motions roll the object around the U (Horizontal) axis, while left-right mouse motions roll around the V axis (vertical).
- **W Rot** — Rotate object about a ray from the camera's focal point through the selected point, keeping that point fixed in site coordinates.
- **Z Rotate** — Rotate the object about the world Z.
- **Z Size** — Scale the Z dimension of the object with respect to the selected point, keeping that point fixed in site coordinates.
- **Z' Rotate** — Rotate the world about the object Z. Set the selected point at vertex.

12.3.4 2-D Methods

The following additional methods may be performed for 2-D objects.

- **Move UV** — Move the selected point on the object in the image (UV) plane.
- **Rotate** — Change the object's orientation, keeping the selected point fixed in site coordinates.

12.3.5 Vertex Manipulation Methods

The following additional methods may be used to manipulate object vertices.

- **Open Verts** — Open the vertices of an object to allow for vertex modification.
- **Close Verts** — Close vertex modification of opened objects.
- **Add Vert** — Add a vertex to a curve or network object.

- **Del Vert/Arc** — Delete a vertex or an arc in a network object.
- **Del Vert** — Delete a vertex in a curve or network object.
- **Merge Vert** — Merge vertices of a basic network.
- **Reset Verts** — Restore the vertices to default positions for basic curves and extruded objects.
- **Split Vertex** — Split the vertex in a basic network.
- **Vert UV on DTM** — Move the vertex of an extruded or 3-D curve object in the image (UV) plane onto the Digital Terrain Model (DTM).
- **Vert UVXY** — Moves a selected vertex of an object in the X-Y plane (at constant Z). Note that a vertex cannot be selected unless the vertices for the object have been "opened" using the Open Verts command.
- **Vert W** — Move a selected vertex of an object in the direction of the camera (W) axis.
- **Vert Z** — Moves a selected vertex of an object in the direction of the local vertical (Z) axis.

12.3.6 Color Mapping Methods

The following additional methods are available for COLOR-MAP-HACKING objects.

- **-Overlays** — Remove overlays for an object.
- **Blue** — Change blue range of colormap.
- **Brt&Cont** — Change brightness and contrast of view or pane.
- **Gain** — Change attenuation (contrast) of view or pane.
- **Gamma** — Change gamma of colormap.
- **Green** — Change green range of colormap.
- **Hue** — Change hue of colormap.
- **Intensity** — Change intensity of colormap.
- **Move Me** — Move a tool object around in the pane.
- **Offset** — Change offset(brightness) of view or pane.
- **Red** — Change red range of colormap.

- **Reset** — Reset object to initial setting. For colormap tools, reset the colormap values. For curve objects, reset to one vertex.
- **Saturation** — Change saturation of view or pane.

12.3.7 Conjugate Point Methods

- **Move Conj UV** — Move the conjugate point UV position (X). The following additional methods are available for CONJUGATE POINT objects.
- **@Image UV** — Set the conjugate point UV position in the given view.
- **Delete UV** — Delete the 2-D conjugate point constraint.
- **Resection Improve** — Improve the resection conjugate point.
- **Resection Menu** — Display the resection conjugate point menu.

12.3.8 View Tool Methods

The following additional methods are available for VIEW-HACKING objects.

- **Auto Stretch** — Perform automatic contrast stretch for the given view.
- **Neg Contrast** — Perform automatic contrast stretch for the given view using a negative ramp.
- **Reset G&O** — Normalize stretch limits of the given view's colormap.
- **Threshold** — Interactively threshold the given view.

12.3.9 Super Methods

The following additional methods are available for SUPERELLIPSE and/or SUPERQUADRIC objects.

- **XY Exponent** — Change the X and Y exponents of a superellipse.
- **X Exponent** — Change the X exponent of a superquadric.
- **Y Exponent** — Change the Y exponent of a superquadric.
- **Z Exponent** — Change the Z exponent of either a superellipse or superquadric.

12.3.10 Camera Methods

The following additional methods are available for CAMERA objects.

- **Move Cam W** — Move the camera along its axis (W axis). This method is also available for PERSPECTIVE TRANSFORM objects.
- **Focal Length** — Change the focal length of a camera. This method is also available for PERSPECTIVE TRANSFORM objects.
- **Force Z Up** — Reset the camera to its canonical orientation.. This method is also available for PERSPECTIVE TRANSFORM objects.
- **Princ Pt** — Move the principal point of a camera. This method is also available for PERSPECTIVE TRANSFORM objects.

12.3.11 House Methods

The following additional methods are available for HOUSE objects.

- **Roof Pitch** — Adjust the pitch of a house's roof.

12.3.12 Cylinder Methods

The following additional methods are available for CYLINDER objects.

- **XY Scale** — Adjust the radius of a cylinder.

12.3.13 Rectangle Methods

The following additional methods are available for RECTANGLE objects.

- **Change Size** — Change the size of a rectangle object.
- **Move Edge** — Move the edge of a rectangle object. If a corner of the object is selected, two edges can be moved simultaneously.
- **Move** — Move a rectangle object in the pane.

12.3.14 Scroll Bar Methods

The following additional methods are available for SCROLL BAR objects.

- **Scroll** — Scroll the indicated view.
- **Scroll2d** — Scroll the view in 2-D.

12.3.15 Window Methods

The following additional methods are available for WINDOW and IMAGE WINDOWING objects.

- **Make Window** — Make a new view whose image is the subimage inside the selected window.
- **TScroll** — Scrolls the view in Tandem mode.

12.3.16 Feature Set Methods

The following additional methods are available for FEATURE SET objects.

- **Desensitize** — Desensitize the feature set, making all objects in the set insensitive to the mouse.
- **Hide** — Hide the feature set, making the feature set not present in that view.
- **Sensitize** — Sensitize the feature set, making all objects in the set sensitive to the mouse.

12.3.17 Composite Methods

The following additional methods are available for composite objects.

- **Desel Superiors** — Deselect the superiors of an object.
- **Open Inferiors** — Open the inferior objects of an object.
- **Sel Superiors** — Select the superiors of an object.

12.3.18 Perspective Transform Methods

The following additional methods are available for PERSPECTIVE TRANSFORM and STARE POINT objects.

- **Range/Focal Length** — Set the range/focal length of the camera.
- **Stare Pt UV** — Move the stare point in the UV plane of the camera.
- **UV Aspect** — Change the UV aspect ratio of the image plane.
- **UV Skew** — Skew the image plane.

12.3.19 Curve Methods

The following additional methods are available for open and closed 2-D and 3-D curve objects.

- **Every Z to Ground** — Move every vertex in Z down to the terrain.
- **Extrude Object** — Extrude a curve into a 3-D object. See also the EXTRUDE object.
- **Object UVXY** — Move the entire object in the image plane.

12.3.20 Half Cylinder

The following additional methods are available for HALF CYLINDER objects.

- **Radius/Length** — Change the radius and length of a half cylinder object simultaneously.
- **Rotate/Scale** — Change the orientation and size of a half cylinder object simultaneously.

12.3.21 Sun Ray Methods

The following additional methods are available for SUN RAY objects.

- **Set Sun Ray** — Set the illumination model to use the current configuration of the sun ray, so that rendering and Sun Z operations work properly.

Chapter 13

The Camera Model

A critical step in constructing a site model is the determination of accurate or approximate camera models for the imagery to be used in the model construction process. This chapter describes RCDE's representation of transforms and projections in detail, followed by the default RCDE camera model and techniques to adjust its parameters.

13.1 Coordinate Transforms

Coordinate transformations are the fundamental representations of geometric relationships between objects in the RCDE. Each object in a site has its own coordinate system in which the locations of its components (vertices, faces, etc.) are defined. In turn, the site itself has a coordinate system that serves as a reference system for all objects and images related to that site. The site coordinate system may be registered to a location on the Earth with a geographic transform. The rotation and position of an object coordinate system with respect to the site coordinate system is expressed as the matrix that transforms the world coordinate system into the object coordinate system.

Homogeneous coordinate transforms for geometric objects in the RCDE are often maintained in the **transform-matrix** slot of the class 4X4-COORDINATE-TRANSFORM or any of its subclasses. Each 3D object in a site has a slot **object-to-world-transform** for an instance of 4X4-COORDINATE-TRANSFORM, which has its **transform-matrix** slot filled with the appropriate 4×4 homogeneous transformation matrix defining the geometric relation between the object and site coordinate systems.

Projections defining the relationship between an image and the site are represented by the class COORDINATE-PROJECTION or any of its subclasses, or 4X4-COORDINATE-PROJECTION. These classes are described in the section on the RCDE pinhole camera.

13.1.1 The Transform Matrix

Coordinate transforms have two main components, a rotation and a translation. In three dimensions, the translation can be represented by a simple three-dimensional vector. The rotational component can be represented by a 3×3 matrix, in which the three columns contain the components of the unit vectors of the new coordinate system with respect to the reference coordinate system in the x, y and z directions, respectively.

The two components are usually combined into a single 4×4 *homogeneous transformation matrix* that can be applied in a single matrix multiplication. The structure of the matrix is defined as

$$\begin{bmatrix} u_x & v_x & w_x & t_x \\ u_y & v_y & w_y & t_y \\ u_z & v_z & w_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where the transformed coordinate system has axes defined by the unit vectors u, v and w with respect to the reference coordinate system, and (t_x, t_y, t_z) is the origin of the transformed coordinate system with respect to the reference coordinate system. This transform matrix converts points in the transformed coordinate system into their equivalent coordinates in the reference coordinate system when the matrix is premultiplied to the point.

The RCDE represents the rotational component of coordinate transforms according to the photogrammetry standard, the *omega-phi-kappa* (ω-ϕ-κ) matrix. Conceptually, the matrix describes the rotation from some coordinate system A to a second coordinate system B (in the context of photogrammetry, A is the world coordinate system, and B is the camera coordinate system) as a sequential set of three rotations. Beginning with the B coordinate system aligned with the A coordinate system, the B system is first rotated by ω about the A x-axis, then rotated by ϕ about the resulting y-axis, and lastly rotated by κ about the resulting z-axis. The final matrix rotates points expressed in B coordinates into points expressed in A coordinates when the matrix is premultiplied to the point.

Mathematically, this matrix $R_{\omega,\phi,\kappa}$ can be computed as

$$\begin{bmatrix} \cos\phi\cos\kappa & \cos\omega\sin\kappa + \sin\omega\sin\phi\cos\kappa & \sin\omega\sin\kappa - \cos\omega\sin\phi\cos\kappa \\ -\cos\phi\sin\kappa & \cos\omega\cos\kappa - \sin\omega\sin\phi\sin\kappa & \sin\omega\cos\kappa + \cos\omega\sin\phi\sin\kappa \\ \sin\phi & -\sin\omega\cos\phi & \cos\omega\cos\phi \end{bmatrix}$$

There are a number of top-level functions available to the user for the creation and adjustment of transformation matrices (see the services relating to transformations in the Programmers' Reference Manual). In particular, the function **make-orientation-matrix** creates a 4×4 matrix with its rotational component set according to the input values of ω, ϕ and κ.

13.2 The Pinhole Camera

The RCDE contains a flexible implementation of a pinhole camera model that allows both perspective and orthographic projection to be represented by the same equations. With traditional pinhole camera models, the equations for perspective projection are

$$u = x \frac{f}{z} + u_0$$

$$v = y \frac{f}{z} + v_0$$

where (x, y, z) are the world coordinates of a point, (u, v) are the projected image coordinates, (u_0, v_0) are the displacement in the image plane and f is the focal length.

Orthographic projection, on the other hand, is defined by the condition $f = \infty$. Therefore, the general orthographic projection equations state that a point's (u, v) coordinates do not depend on the point's z coordinate. Instead, (x, y) and (u, v) are related by a simple proportion,

$$u = \mu\, x + u_0$$

$$v = \mu\, y + v_0$$

where μ is a constant, often related to the focal length of the sensor.

Because the perspective projection equations depend on f, but $f = \infty$ in orthographic projection, the perspective equations cannot be used directly to model orthographic projection. However, if the singularity at $f = \infty$ is removed, it is possible to use the same equations for the two projection models.

The RCDE accomplishes this unification of projection models by defining the camera coordinate system to have its origin at an arbitrary distance r along the principal ray of the camera, rather than at the camera's center of projection (the center of the lens) as the standard projection equations imply (see Figure 13.1). The RCDE projection equations then become

$$u = x \frac{f}{z - r} + u_0$$

$$v = y \frac{f}{z - r} + v_0$$

which can be written as

$$u = x \frac{1}{z\frac{1}{f} - \frac{r}{f}} + u_0$$

$$v = y \frac{1}{z\frac{1}{f} - \frac{r}{f}} + v_0$$

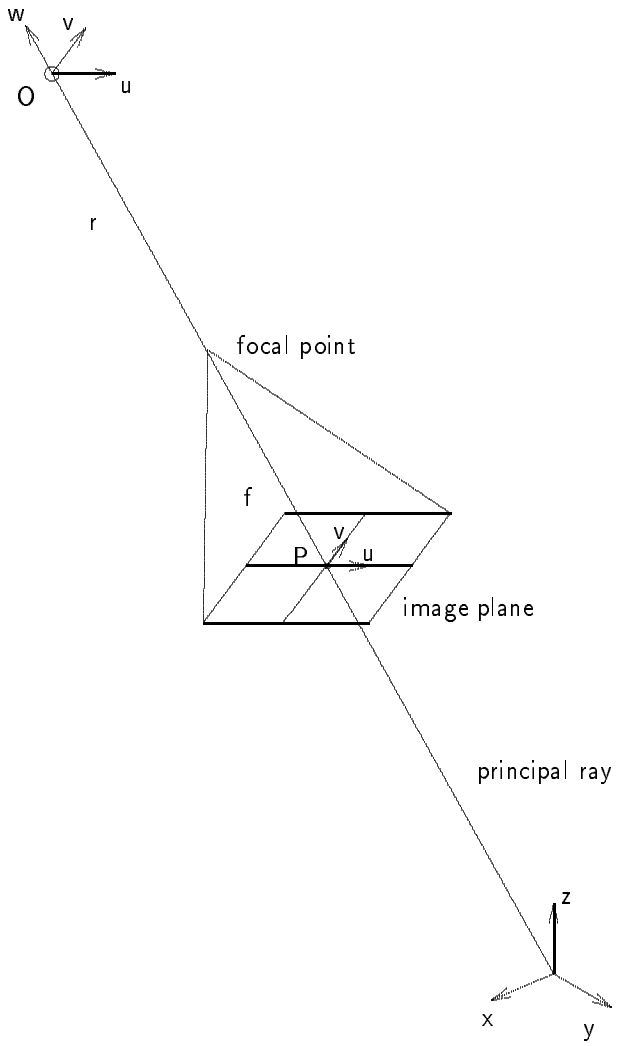


Figure 13.1: The pinhole camera model. The focal length f is the distance from the image plane to the focal point. The origin of the camera coordinate system, O, lies on the principal ray at a distance r from the focal point; the axes **u** and **v** are drawn on the image plane to show that the **u**-**v** plane is parallel to the image plane. The **w** axis is perpendicular to the image plane, and the principal point **P** lies at the intersection of the principal ray and the image plane.

As $f \to \infty$ these equations still present a singularity, unless the ratio r/f is known to be constant. If that is assumed, then at $f = \infty$, or the case of orthographic projection, the equations are

$$u = x \frac{1}{0 - \frac{r}{f}} = -x \frac{f}{r}$$

$$v = y \frac{1}{0 - \frac{r}{f}} = -y \frac{f}{r}$$

which is equivalent to the orthographic equations above when $\mu = -f/r$ and f/r is constant. Note that $-1/(r/f)$ is the scale factor between coordinate frames; this is particularly useful when using orthographic projection to represent a simple transform from one 3-D system to another (no actual projection is performed). In this case, the w coordinate is computed in a symmetric manner:

$$w = -z \frac{f}{r}$$

For perspective projection, the parameter r also behaves as a scaling parameter, but in a different way. Since the w coordinates of points visible in a view are defined to be negative, positive values of r will decrease the scaling; in effect, the larger the value of r, the smaller the object will be in the projection. Setting r to negative values is not advisable, because that moves the origin of the camera coordinate system toward objects in the scene. As the values of r and the w coordinates of objects approach equality, the projection equations degenerate toward infinity.

This mathematical representation thus allows a continuous transition from a perspective projection model to an orthographic projection model as the focal length increases, without an undefined condition as the focal length approaches infinity. Orthographic projection is modeled by setting $1/f = 0$ and r/f to some nonzero value. Perspective projection is modeled by using the focal length of the camera for f, and setting r to a nonnegative value based on the desired scaling of the projection.

In a more general sense, the equations governing many forms of camera projection can be formulated as a matrix multiplication:

$$\begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

where the matrix $\mathbf{T}$ contains the transformation to camera coordinates, the modeled projection effects and scaling. In its default camera model, the RCDE maintains homogeneous transformation matrices without projection or scaling. These two effects are applied after a point has been transformed to the camera coordinate system.

In implementing the RCDE camera model, the 4X4-COORDINATE-PROJECTION class does not have a slot for the focal length; rather, it contains a slot for the multiplicative inverse of

the focal length, **1/f**. Similarly, the distance of the origin of the camera coordinate system along the principal ray, r, is represented only as the ratio **r/f**, since it must be constant.

The 4X4-COORDINATE-PROJECTION class also has a **transform-matrix** slot for the ω-ϕ-κ transformation matrix describing the relation between the camera coordinate system and the world coordinate system. This matrix transforms points in the camera coordinate system into points in the world coordinate system; the inverse matrix is used to transform points in the other direction. The translation embedded in the **transform-matrix** is the location of the focal point of the camera (the center of the lens) with respect to the site coordinate system.

The camera coordinate system in the RCDE is defined to have its U axis horizontal and pointing to the right as the user looks at the associated image, or along increasing column indices in the image. The V axis is vertical and pointing up as the user looks at the image, or along increasing row indices. The w axis is perpendicular to the $U - V$ image plane, and points *out* of the image, toward the user, to maintain a right-handed camera coordinate system. Thus w camera coordinates of objects visible in the view are always negative.

By default, the origin in the image plane is at the lower left corner of the image. By definition, the projection of the principal point into the image plane gives the camera origin's location in the u and v dimensions. Recall that the camera origin's location on the w axis is determined by the parameter r.

The RCDE requires that the transformation matrix contained in the **transform-matrix** slot of the class 4X4-COORDINATE-PROJECTION has a rotational component consisting of *unit* vectors. If there is a change in scale between two coordinate systems in addition to the change prescribed by the transformation matrix, then each coordinate of points being transformed from one coordinate system to the other must also be multiplied by the scale factor. If the coordinate transform is defined by an orthographic 4X4-COORDINATE-PROJECTION, then the scale factor is simply **r/f**.

If a coordinate transform is not defined by a 4X4-COORDINATE-PROJECTION, then the scale factor must be included directly in the matrix representation. For example, the class BASIC-4X4-TRANSFORM does not have a specific slot for a scale parameter; it assumes that the scaling is incorporated into the matrix. Mathematically, this is accomplished by multiplying each element of the 3×3 rotation matrix embedded in the upper left portion of the transform matrix by the scale factor. Points transformed by multiplying them with this matrix will be rotated, scaled and translated in one matrix operation, saving a significant amount of floating point computation.

13.3 Camera Refinement

A critical step in constructing the site model is building an accurate camera model for each image of the scene. If the position, orientation and focal length of the camera used to photograph a particular image are known, the RCDE provides a simple menu interface, the camera object menu, for specifying camera parameters directly.

For many images, however, the focal length of the camera used to photograph a particular image is known but the position and orientation of the camera when the image was taken are only approximated at best. For cameras with unknown rotation and orientation parameters, the RCDE provides *resectioning* techniques to automatically recover the position and orientation of the camera relative to a special set of 3-D points, referred to as *conjugate points*.

13.3.1 Entering a Known Camera

The camera position and orientation can be described by the six parameters, three angles and three scalars, used to calculate the transform matrix. These parameters can be entered by the user directly on the screen, once the associated image is loaded and a camera object has been initialized.

- Load the ALV image "$CMEHOME/alv/alv-3-42.g0" with I/O→Load Image.

- Create an unregistered camera for the loaded image with Transforms→New View Transform on Image. Since a view must be contained within a site, a pop-up menu giving a choice of existing world objects appears. Select the ALV world to indicate that the view belongs in this particular site (the world object must have been created already by loading the ALV site).

  A box containing the text string "Camera" should appear in the upper left corner of the pane – this is the camera object associated with the camera model in that view.

- Move the mouse onto the camera object and click the right mouse button. The camera object menu should appear on the screen.

- Enter the six camera pose parameters and the camera focal length f by clicking the mouse on each and typing in the appropriate numbers. For this camera, the parameters have been calculated to be

ω	0.00286348
ϕ	0.0015922
κ	−1.563783
x_{cam}	5194.016
y_{cam}	4149.6226
z_{cam}	11993.813
$1/f$	−0.00054202342

- Exit the menu. The camera model and the view are automatically updated, possibly changing the apparent orientation of any visible features.

If the camera parameters were not derived by the RCDE, then the rotation angles may not follow the same convention for transforming coordinate systems. However, if the complete transform matrix is available, the RCDE contains functions to recover the three angles ω, ϕ and κ by matrix decomposition. See the Programmers' Reference Manual for details and related functionality.

13.3.2 Manually Adjusting Camera Positions

The RCDE provides two resectioning algorithms. The first assumes that the initial camera position is unknown, and attempts to converge from a selected set of initial viewpoints. The second assumes that the existing camera model has been placed in approximately the correct location by the user. While the second method requires more user interaction, it is more robust, and should yield better results in cases where the approximate camera position is known.

The initial manual camera positioning process is more of an art than a science – there is no standard procedure to follow. Since the resectioning routines are based on mathematical methods that can fail if the solution is in too distant a configuration from the initial parameter set, it is important in some cases for the user to set a model that is roughly accurate.

When adjusting a camera's position, it is important to remember that any modeled objects in the site are fixed relative to the site coordinate system while the camera is being moved about the site. The goal of resection, manual or automatic, is to move the camera into the position from which the image was taken, so that the modeled objects correctly overlay their locations in the image.

To understand the motions of the camera, the user must understand the functions of the components of the system camera model, shown in Figure 13.2. The camera has its own coordinate system, with the u axis in the horizontal direction and the v axis in the vertical direction as the user looks at the pane. The w axis of the camera is perpendicular to the image plane, and points out of the image toward the user. The origin of the camera coordinate system lies on the principal ray, at a distance r as explained in the above section on camera models.

The *principal point* defines the direction of the optical axis, or where the camera is pointing relative to the site – more rigorously, it is the intersection of the image plane and the principal ray (the optical axis). Since the principal point can be anywhere on the image plane, it does not necessarily have to be visible in the image; the user may move the principal point beyond the visible part of the image plane with the Princ Pt menu button on the camera menu. The principal ray is always perpendicular to the image plane, however, so that moving the principal point also moves the camera location because the image plane is kept constant.

The *stare point* is, by default, the location of the intersection of the principal ray with the DTM, or the $z = 0$ plane in the site coordinate system if no DTM is registered. Graphically, it is identical to an **axis** object, which shows the directions of the site coordinate axes. The

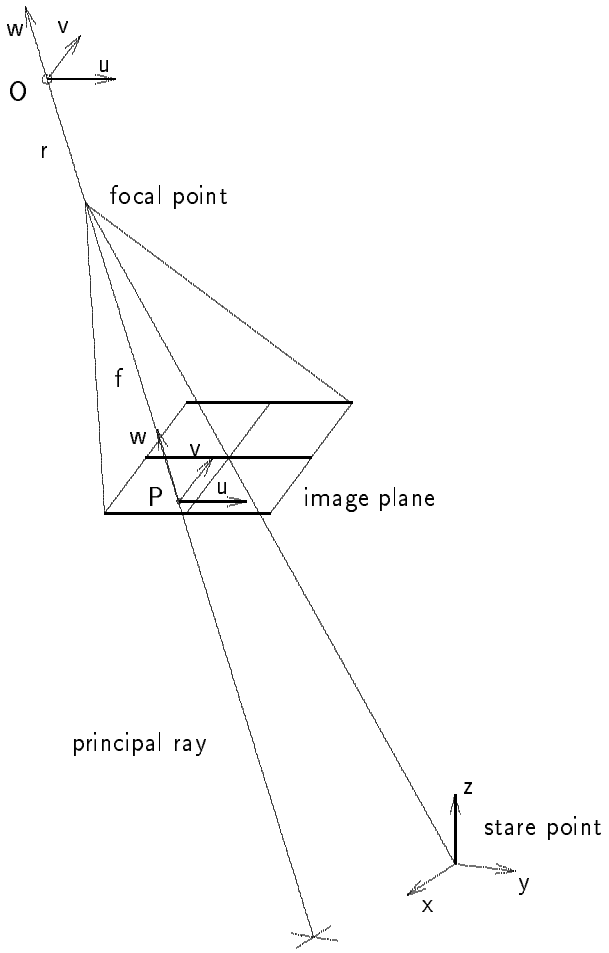


Figure 13.2: The pinhole camera model when the stare point and the principal point are separated. The focal length f is the distance from the image plane to the focal point. The origin of the camera coordinate system, O, lies on the principal ray at a distance r from the focal point. The principal point $\mathbf{P}$ lies at the intersection of the principal ray and the image plane. The axes u and v are drawn on the image plane to show that the $u-v$ plane is parallel to the image plane. Note that the w axis is, by definition, always perpendicular to the image plane, implying that the ray from the focal point to the stare point is not perpendicular to the image plane.

primary purpose of the stare point is to provide a mechanism for moving the camera relative to an arbitrary point in the model, resulting in a more intuitive set of camera manipulation tools. When a blank view is first created with the New View Transform on Image option on the Transforms menu, the projection of the stare point and the principal point are at the same location in the image (recall that the principal point is defined in the camera coordinate system, while the location of the stare point in the image is the projection of its location in the site).

The stare point can be moved in the image plane without adjusting the camera by using Stare Point→Stare Pt UV. Once it is moved, though, the stare point will no longer be on the principal ray, and the letter "P" will appear on the image, next to a "+" indicating the new position of the principal point. The camera can be rotated and translated with respect to the site by selecting the stare point object (the visible set of coordinate axes that appears when the view is created), and using the Move UV @Z, Move Z, Drop Z, Move W, Drop W and Az-El commands, all of which keep the stare point at the same position on the pane while moving the modeled objects on the screen. Contrary to appearances, though, it is actually the camera position that is being adjusted with respect to the site coordinate system.

Moving the principal point with the Princ Pt option causes a more complex interaction with the model that is not immediately obvious. As the principal point is moved the camera location is also adjusted, according to two constraints. The first requires that the stare point remains in the same u, v position in the view. The second constraint forces the principal ray to remain perpendicular to the image plane; in effect, the focal point of the camera is moved to be directly above the principal point while the image plane is held constant. The orientation of the camera remains unchanged (see Figure 13.3).

Geometrically, these two conditions cause the view to rotate about the model and the perspective distortion to increase as the principal point is moved further away from the image center, because the optical axis is being moved away from the visible portion of the image plane. This is useful when modeling a camera that has a high perspective distortion, or when the image has been windowed from a region near the border of a larger image (and hence has significant perspective distortion).

The following steps describe the beginning of a scenario to perform camera resection. To obtain an initial camera model, the ALV site is used. A view from the site is copied, then slightly adjusted so that it is out of alignment. Resection is employed to adjust the camera parameters of the copy, using the original view as a reference. On completion, the copied view should once again have the same camera parameters as the original.

- Load the ALV site with I/O→Load Site Model.
- Copy the view containing the image **$CMEHOME/alv/alv-3-42.g0** to a new pane using Stacks→Copy View. Make the copied view adjustable using Transforms→Make Transform Adjustable. This should automatically set the copied camera model to be adjustable. Visually, the camera object, or a box containing the text string "Camera",

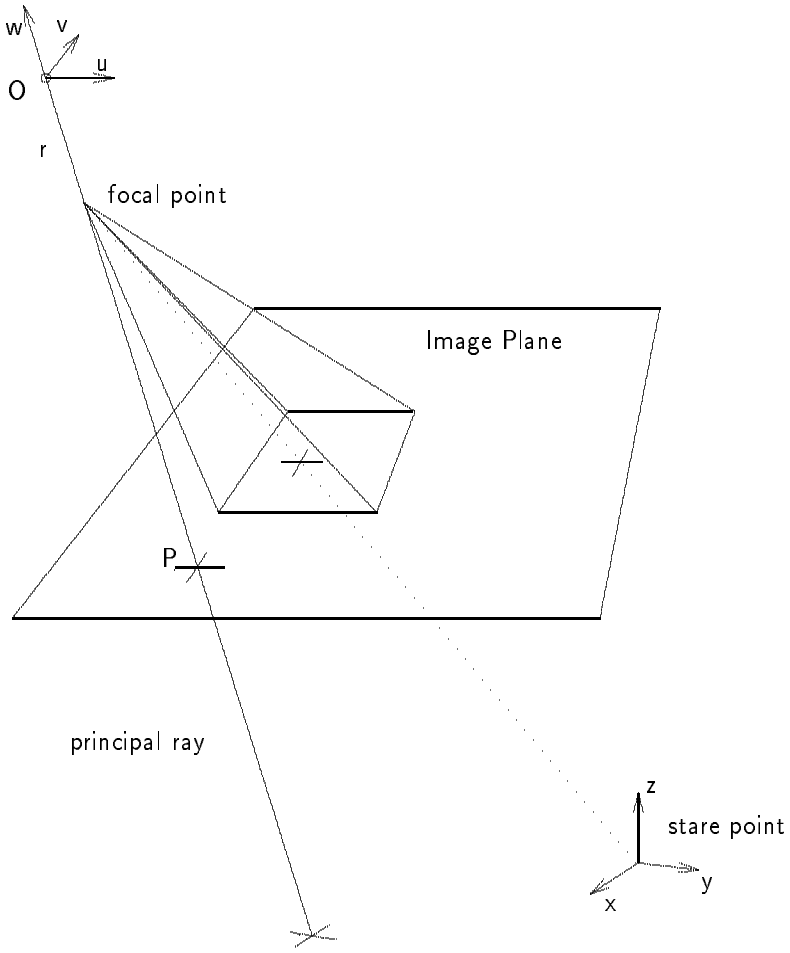


Figure 13.3: The pinhole camera model when the stare point is visible in the view, but the principal point has been moved beyond the visible portion of the image plane (the rectangular region inside the larger region labeled "Image Plane"). The principal ray is perpendicular to the image plane, so that the perspective distortion in the view will increase as the principal point is moved farther away from the visible portion of the image plane.

should appear in the upper left corner of the new view. The new view will be a view of the same site, and hence will have a registered terrain model.

- Create an Axis object on the registered image (i.e., the original view from which the copy was made). This set of axes shows the orientation of the site coordinate system relative to the camera in its view. It serves as a reference between the images, showing a common point as seen from both views.

- If the Axis is not visible in the unregistered image, zoom out until it can be seen. Be careful not to confuse the axis object with the camera's stare point; although they appear identical on the screen, only the axis should be highlighted in green in *both* images when selected.

- Place the mouse on the camera object or the stare point in the unregistered image, and move the camera about using Stare Point→Az-El (azimuth-elevation) or Camera→UV Roll. When the axis object is approximately in the same position in both images, click the left mouse button twice to deselect the camera object. The axis should now be on the same image feature in both images, but its orientation in the two images will not be the same. The new camera is now ready for automatic adjustment with the resection routine.

13.4 Applying Camera Resection

Resection is usually applied to camera models, rather than terrain data or maps. In most cases the terrain data transform is known, and data taken from maps can be accurately placed in the site coordinate system. In some cases the camera parameters are known for an image (the image is said to be *registered*); additional camera models for more images of the site can be resected using conjugate points visible in the registered image and the unregistered images. In either situation, resection of an unknown camera is performed with respect to data from a known (or simply fixed) source, such as another camera or the DTM.

The resection process has four major steps, which are detailed in this chapter (see Figure 13.4). First, the camera to be resected must be manually placed in a position close to its actual location (this step may not be necessary if adequate results can be obtained from the resection algorithm using random starting locations). Second, conjugate points are created and placed at their known locations in the site, using the DTM and a correctly registered image. Since the camera to be adjusted has a view of the site, the conjugate points should be visible in its view, but they are probably not in the correct positions. Next, the points are manually moved to their proper locations using a special menu option that moves the points only in the view of the camera to be resected. This difference in locations is the critical input for the resection routine. Finally, the resection algorithm is called to find the best set of camera parameters that aligns the projected locations of the conjugate points with the specified correct locations.

Load an unregistered image.	I/O→Load Image
Create a view on the image.	Transforms→New View Transform On Image
Adjust the new camera into approximate position.	Camera→UV Roll, etc.
Create conj. pts. in reference image.	Create Object→Conjugate Pt
Position conj. pts. in second image.	Conjugate Pt→Move Conj Pt
Call the resection function.	Conjugate Pt→Resect

Figure 13.4: The Steps in Executing Automatic Camera Resection.

13.4.1 Conjugate Points

In the simplest terms, conjugate points are 3-D locations that can be identified and correlated between multiple views of the same site. For example, the corners of a building that are visible in two images could be used as conjugate points between the two images. More generally, conjugate points are points whose locations can be identified in multiple sources of information about the same site, such as images, terrain data and maps. For example, if a map of a site shows the location of a building, and this same building is visible in an image of the site, then the visible corners of the building may serve as conjugate points between the map and the image.

Conjugate points are assumed to be known and fixed relative to the site coordinate system and one data source. This implies that the transform between the data source and the site is fixed and assumed to be correct. The resection process uses conjugate points between the known data source and a second data source to adjust the transform parameters defining the location of the second data source relative to the first.

In the best case the positions of conjugate points are known relative to some absolute coordinate system, such as the geographic coordinate system. If the position of the DTM in geographic coordinates is known, then any of the points on the DTM can serve as conjugate points, provided that they can be identified in another data source. When camera models are resected using points on the DTM, the cameras will then be in positions at which geographic coordinates can be computed. In many cases, however, accurate absolute data is not available. While this situation does not prevent resection, it does limit the camera models to be solved only with respect to one another; they may not be properly scaled in absolute units (such as meters).

13.4.1.1 Creating Conjugate Points

Once the camera to be resected is in approximately the correct position, the conjugate points must be set in the image associated with that camera. For this example, the alignment and resection will be between two images, one of which is assumed to be correctly registered

to the ALV world. As mentioned above, accurate registration of one image is important, because it provides precise 3-D locations in the site coordinate system for the conjugate points. If no images were registered, the 3-D coordinates could be obtained from the DTM or a map.

The purpose of conjugate points is to quantize the error in a camera model (referred to as the unregistered camera) relative to a known camera model, so that this error can be minimized by adjusting the position of the unknown camera. The user creates conjugate points in the image associated to the fixed camera model with the Conjugate Point option on the Create Object menu. This will place a new conjugate point at the position pointed to by the mouse when the left button is pressed. However, for this point to be correct in three dimensions, a DTM must be registered to the same site as the registered image.

It is essential that conjugate points are given correct coordinates in three dimensions. If no DTM is registered to the site, the default z coordinate will be set to zero. This will lead to a set of coplanar conjugate points, which will most likely cause the resection algorithm to fail. Without a registered DTM, the z coordinate can be manually set for each point by entering the appropriate value in the Conjugate Pt→Menu.

The conjugate points must be at locations that are visible in both images, such as corners of buildings or roads that are clearly displayed from both viewpoints. For reasonable resection performance, at least six conjugate points should be defined for an image pair. The points should not lie in the same plane, and they should span the registered image so that a reasonable perspective model can be approximated. The conjugate points relating the two images must also be included in the same Feature Set.

13.4.1.2 Aligning Conjugate Points

Unless the unregistered camera is already in perfect position, the conjugate points will not appear to be in exactly the correct locations in the unregistered image. For example, a conjugate point on an upper corner of an imaged building in the registered image may be projected onto the center of the roof of the same building in the unregistered image. Some points will have greater apparent error than others because of the nonlinear nature of the projective model.

With the conjugate points visible in the unregistered image, the user can see where the points are relative to world coordinates and also where they should be according to the image. Using the Move Conj Pt option on the Conjugate Point menu, the correct location in the image plane for each conjugate point is specified with the mouse by moving the mouse from the crosshair to the proper location. Once the mouse moves away from the crosshair, a small "x" should appear and move interactively. Clicking the left mouse button will secure the "x" in place.

All conjugate points that are created (and are in the same feature set) must also be corrected, or else they will be taken to be properly positioned in both images (i.e. the error in the uncorrected points will be taken as zero).

- Conjugate points are created interactively using the Create Object menu option *Conjugate Point*. By observing the point locations in the registered views, create and position a number of conjugate points on prominent site features that are also visible in the unregistered image. The DTM must be registered to the site, and both the registered view and the unregistered view must be views of the site.

  With the Conjugate Pt→Move Conj Pt option, mark the correct location of each conjugate point in the unregistered image. The corrected locations, marked by an "x", will only be visible in the unregistered view.

13.4.2 Executing Camera Resection

When all the conjugate points are prepared, the resection algorithm is executed by selecting the Resection Menu option on the Conjugate Pt menu of any of the created conjugate points. All of the conjugate points in the same feature set as that point will be applied in the resection, while existing conjugate points in other feature sets will be ignored. This constraint imposed by the resection routine allows multiple sets of conjugate points, perhaps pertaining to different images, to coexist simultaneously if they are grouped into independent feature sets.

The execution may take a few seconds, but the camera model, and the positioning of objects in its view, will be updated automatically when the new camera parameters have been computed.

Resection can be run multiple times to achieve more accurate refinements of the same camera model as more conjugate points are created. When new images are added to the site, previous camera models may be updated because of improved views of common features in the new image.

- Create the resection menu by clicking on Conjugate Pt→Resection Menu for a conjugate point in the view to be resected, or its equivalent in the registered image. Only conjugate points in the same Feature Set as the selected conjugate point will be used for resection.

  Two resection search modes are available, depending on the desired initial state of the camera model. "Random" mode searches for a resection solution beginning from a number of viewpoints distributed over the viewing hemisphere. This number is entered in the menu item "Number of Random Starting Estimates=". "Improve" mode begins with the existing camera model as its only starting estimate. The menu entries for focal length and principal point coordinates allow the user to enter initial estimates for these values. The convergence threshold menu item allows the user to specify a desired accuracy in pixels to be achieved by the iterative resection algorithm.

- Execute resection by setting the "Search Mode" to "Random" and clicking on the DOIT button. When the resection has completed, the view should be updated auto-

matically with the new camera model. The crosshairs of the actual conjugate point locations should be noticeably closer to their correct positions.

Change the Search Mode to "Improve" and click on DOIT again. This time, the final camera position should be more accurate than with random starting positions.

Chapter 14

Terrain Data Integration

At any time during the model-building process the user may choose to integrate digital terrain data with the current site model. A digital terrain grid is a set of known elevations spaced at regular intervals over a specified region (usually square). Once the digital terrain model (DTM) is registered to the site, the user may take advantage of the knowledge of ground-truth when placing objects in the site (Move UV on DTM, Drop Z), creating additional camera models by resection and rendering complete scenes. The DTM points are not usually shown in a view, but by creating a DTM Mesh object the user can see a movable grid overlaid on a portion of the DTM. It is suggested that the terrain model integration should be completed as soon as possible in the site modeling process, so that subsequent additions to the model can take advantage of the DTM features.

Currently there is no menu support for registering terrain data to the local coordinate system. In general, the user must supply terrain data in a known image format, and must register the terrain image to the model by applying the RCDE functions that rely on *a priori* knowledge of the relationship of the site to the terrain data. Because of the possible complexity of this process and the variety of forms of input terrain data, it is recommended that the user must make use of the Lisp command line, or a file of Lisp or C/C++ code, to load the data into the system's DTM object. Once created, however, this object is essentially just an image containing elevation data instead of intensity values.

Direct support is provided for US Geological Survey Digital Elevation Map (DEM) files, which have a standardized format. Functionality exists to read DEM file headers and terrain image data into an RCDE structure and an RCDE image, respectively. See the *Programmer's Reference Manual*, Section USGS-DEM for details.

Whichever format of terrain image is used, the DTM registration process can be divided into the following steps:

1. Load the terrain image as a standard RCDE floating-point image.

2. Define the DTM coordinate transform mapping the terrain image to the site coordinate

system, including appropriate scaling.

3. Create an RCDE terrain model object with the DTM transform and the DTM image.

4. Link the created terrain model to a world.

The next section of this chapter contains an explicit description of the complete process required to integrate USGS DEM files. This is included as a brief scenario to illustrate some of the concepts and functionality used to register terrain data to a site. The remaining sections detail how a user might integrate a terrain file format that cannot be directly read by the RCDE.

14.1 Registering USGS DEM Files

Because it can read the header format, the RCDE can extract image data and other information, such as scaling and latitude-longitude position, from USGS DEM Files. This collateral information provides most of the data needed to integrate a DEM file into a site; the only remaining piece that must be supplied by the user is the transform describing the mapping between the terrain data location and the site coordinate system.

For example purposes, a DEM file, **$CMEHOME/alv/usgs.dem**, is provided with the ALV data set. This file is loaded and registered in this scenario. Although programming commands in this section are given in Lisp syntax, it should be noted that all functions called here may be called from C/C++.

Step 1 above, loading the terrain image, is accomplished by a pair of function calls. The first loads the header of a DEM file, and saves the result in a temporary variable **usgs-header**:

```
(setq dem-header
(cme::read-usgs-dem-header "$CMEHOME/alv/usgs.dem"))
#<Usgs-Utm-Dem-Header #X2F6A3CE>
>
```

The second loads the corresponding DEM image data into an RCDE image:

```
(setq dem-image (cme::make-dem-image dem-header))
#<BLOCKED-ARRAY-MAPPED-IMAGE 402 x 466 SINGLE-FLOAT #X30BF306>
>
```

The terrain image should appear as an image of type **SINGLE-FLOAT** in the selected pane.

Step 2 requires the user to provide information regarding the mapping of the terrain image to the site coordinate system. Since this information is dependent on the user's

choice of the position and orientation of the site coordinate system, it cannot be provided by the RCDE. The information must be encoded in a transform object, of the class **4x4-coordinate-transform** or one of its subclasses. See Section14.3 below for more details.

Step 3 consists of a single function call to a function specialized to work on USGS DEM images. If the DEM transform is pointed to by a variable (or *c_handle*) named **dem-transform**, then the following function call creates an RCDE terrain model:

```
(setq terrain-model
(cme::make-usgs-terrain-model dem-image dem-transform "USGS DEM CS"))
#<Non-Linear-Mapped-Regular-Grid-Terrain-Model #X37390DE>
>
```

The string argument is a textual name given to the DEM coordinate system.

Step 4, linking the resulting terrain model into a world, is accomplished for any terrain model object (not just USGS-derived ones) by setting the terrain model property of the chosen world. If the world is named ***alv-3d-world***, then the following function call adds the terrain model to the world:

```
(setf (get-prop *alv-3d-world* :terrain-model) terrain-model)
#<Non-Linear-Mapped-Regular-Grid-Terrain-Model #X37390DE> >
```

The terrain model should now be completely integrated into the world. The elevation of points visible in registered should be available by clicking with the left mouse button, buildings may be moved on the DTM, and so on.

14.2 Preparing a DTM Image

With terrain data in a file format unknown to RCDE, the user must supply certain information that RCDE automatically extracts from known file formats, such as the actual distance between data points, the units of distance used, and so on.

Even with this information, the elevation data must be read into RCDE. It is the user's responsibility to supply the elevation image in a form that the RCDE understands. This can be accomplished in two ways. The first involves independently creating an elevation image file in a format that the RCDE can read, such as a Sun raster file, which is then loaded into the RCDE in the same manner as a normal intensity image. The second method, which can be applied to input file formats not supported by the RCDE, involves writing a C, C++ or Lisp function to read the specific input format, creating an internal RCDE image structure of the appropriate size, and copying the data into the RCDE image. It can then be saved in a standard RCDE format, so that the RCDE I/O functions can be used to load it in later. Since the second method uses internal RCDE structures, it must be coupled (loosely

or tightly) to the Lisp process; the first method can be an independent program, since it outputs a file.

With either technique, before the RCDE can use the elevation data it must be transformed such that the units of elevation are in the same scale as the indices of the array (i.e., the x, y, and z axes of the elevation data are all on the same scale). Because of the variety of elevation image file formats, the RCDE does not provide functional support to accomplish this step for file formats other than USGS. To demonstrate the concepts involved, though, an example of how a standard DMA Digital Terrain Elevation Data (DTED) Level 1 file would be prepared is presented here.

14.2.1 Loading a DTED File as a RCDE Image

DTED files have a header containing information on the latitude and longitude of the position of the lower left corner of the elevation data image, and all Level 1 files cover an area of one degree by one degree on the Earth's surface. In this context it is assumed that the DTED file contains data at the maximum Level 1 resolution, or one data point every 3 arc-seconds in both latitude and longitude. Since the elevation is given in meters, while the x and y coordinates are in the geographic coordinate system, the x and y coordinates will be converted into meters with the local origin at the lower left corner of the elevation image. Then the elevation values will be scaled to be expressed in units equal to the size of x-y pixels.

First, an RCDE floating-point image is created to store the terrain data. In this example, the image is 1201 by 1201 pixels, because there are 1200 3-second intervals in one degree, and DTED files have an extra row and column on two of the image borders. Floating-point data values are used for greater accuracy during interpolation between known data points. The image can be created, with all pixels initialized to zero, and stored in the variable **dtm-image** with the command

```
(setq dtm-image
(ic::make-image '(1201 1201) :element-type 'single-float))
```

The symbol **dtm-image** now points to an allocated image structure prepared to receive floating-point values.

Next, the scaling factor for the elevation data is calculated. Each elevation value must be multiplied by this factor before it is entered into the elevation image. DTED files, being delineated by latitude and longitude lines, do not actually cover square regions on the Earth's surface, as their symmetric structure would indicate. However, approximating the DTED area as a square does not introduce large error if the latitude is relatively near the equator. Assuming this, the 3-second spacing between data points on the same latitude corresponds to about 100 meters on the Earth's surface. As the latitude approaches the poles, this distance decreases to zero. The 3-second spacing between points of the same

longitude is also approximately 100 meters, but this distance remains constant for DTED at any latitude.

For the DTED file, then, the z values must be expressed in terms of units of 100 meters, the spacing in the x-y plane. Programmatically, this corresponds to dividing each elevation in the DTED file by 100 before it is entered in the RCDE DTM image.

Though the programming details are not given here, in pseudo-code the central routine to transfer the points from DTED format into the RCDE image looks something like this:

For each column *c* of DTED points

    For each row *r* in *c*

        value = dted(*c*,*r*) / 100;
        (iset value **dtm-image** *c r*)

The function dted(r,c) returns the value of the elevation at the appropriate point in the DTED file. The RCDE function **iset**, which sets the value of a pixel in an image, is given in exact Lisp syntax. The corresponding C/C++ *iset()* function call would be

```
iset_float(value,dtm_image,c,r);
```

where *dtm_image* is a *c_handle* that points to the RCDE image.

Reading the DTED file by column, then row from the beginning of the DTED input ensures that the image will have the correct orientation in the RCDE. The origin of the image coordinate system is at the lower left corner of the image, with the u-axis pointing to the right and the v-axis pointing up. Since columns of DTED data are ordered from west to east, and rows are ordered from south to north, reading the DTED file as shown above will create an image with its v-axis pointing north, and its u-axis pointing east.

It is not necessary to align the DTM along geographic lines, but doing so simplifies the transformation from the DTM to the site, if the site is also aligned along geographic lines. The DTM transform can have a significant impact on performance, since it is often used in calculations for real-time graphics.

14.2.2 Loading a RCDE-Compatible DTM Image

Elevation data stored in an image file format that RCDE supports can be read into the system using the **load-image** command. No special syntax is required to accommodate elevation images, but the file header information indicates that the data is in floating point format.

- To load the ALV terrain data image, which is in IU Testbed image format, enter the following expression into the Lisp Lisp command line:

```
(setq alv-dtm (load-image "$CMEHOME/alv/alv-dtm.g0"))
```

This will create a variable **alv-dtm** which points to the image structure automatically created and filled by **load-image**.

For easier identification, the image can be given the literal name "Alv-Dtm" with the Lisp statement

```
(setf (image-prop alv-dtm :name) "Alv-Dtm")
```

It should be emphasized that once the image is loaded in floating-point format, the original format of the image file is no longer significant. This allows any floating point image to be used as a terrain map, regardless of how the data is stored on disk.

14.3 The DTM Transform

With the terrain data loaded into an RCDE image, the user must define the spatial relationship between the terrain and the site. This transformation from the DTM coordinate system to the site coordinate system can be established using a priori knowledge, or may be estimated if no absolute is available. In either case, the transform may be updated at any later time as the site develops; however, objects placed according to the DTM's previous position will not be automatically adjusted so that they remain on the DTM. Thus it is strongly recommended that the DTM transform be fixed early in the model construction process.

The spatial relationship between the DTM and the site coordinate system is encapsulated in the **transform-matrix** slot of a 4X4-COORDINATE-TRANSFORM object associated with the DTM. Both the translation and the rotation of the DTM are encoded in the matrix.

The translation component describes the position of the lower left corner of the DTM image relative to the site (or world) coordinate system. Referring to the notation used in Chapter 13 for transformation matrices, the matrix translation elements t_x, t_y and t_z are x_{dtm}, y_{dtm} and z_{dtm}, respectively, where the point $(x_{dtm}, y_{dtm}, z_{dtm})$ is the location in world coordinates of the lower left corner of the DTM image.

The rotational component of the DTM transform matrix describes the orientation in 3D of the DTM image relative to the site coordinate frame. Although the x-y planes of the DTM and site are usually set to be parallel, it is not required by the RCDE; the transform matrix may contain rotations about all three axes. In general it is advisable to align the DTM and site coordinate systems as much as possible, since there is heavy computational dependence on the DTM matrix.

The DTM-to-site axis alignment can be accomplished in two ways, depending on whether the DTM is available before the site modeling process is begun. In that case, the site coordinate frame can simply be defined to be in alignment with the DTM coordinate frame,

i.e. the two frames would have the same axes, and the origin of the site coordinate frame would be at the lower left corner of the DTM image. If DTM data is acquired after the site coordinate frame has been established, then axis alignment in the x-y plane can be achieved by rotating the DTM image using the RCDE Rotate operator, before the DTM image is registered as the elevation image for the site.

Given that the geometric relation between the DTM and the site is known, the user can encode this relationship in a transform matrix created by using RCDE matrix functionality (see the *Programmer's Reference Manual*).

For the ALV data set, the DTM transform matrix may be defined with the Lisp statement

```
(make-and-fill-2d-array
 `((0.9999981 -0.0013416967 -0.0014063651 -7890.5878999999995)
   (0.0013461612 0.9999933 0.0016050119 176.01)
   (0.0011754258 -0.0037573292 0.9992737 4.999999999883585)
   (0.0 0.0 0.0 1.0)))
```

This matrix implies that the lower left corner of the DTM is located at the world point $(-7890.5879, 176.01, 5.0)$, in world units. The rotational component indicates that the axes are closely aligned with the world coordinate system, since the unit vectors embedded in the transform are approximately $(1,0,0)^T$, $(0,1,0)^T$ and $(0,0,1)^T$.

This matrix should then be used to construct an instance of 4X4-COORDINATE-TRANSFORM or 4X4-COORDINATE-PROJECTION, which will be the DTM transform object. For example, the Lisp form

```
(MAKE-INSTANCE '4X4-COORDINATE-PROJECTION
               :R/F
               0.0
               :1/F
                0.0
               :PRINCIPAL-POINT-U
               0.0
               :PRINCIPAL-POINT-V
               0.0
               :POSITIVE-W-CLIP-PLANE
               0.0
               :TRANSFORM-MATRIX
               (make-and-fill-2d-array
                `((0.9999981 -0.0013416967 -0.0014063651 -7890.5878999999995)
                  (0.0013461612 0.9999933 0.0016050119 176.01)
                  (0.0011754258 -0.0037573292 0.9992737 4.999999999883585)
                  (0.0 0.0 0.0 1.0)))
               :PROPERTY-LIST
               nil)
```

creates an instance of the class 4X4-COORDINATE-PROJECTION containing the ALV DTM transform.

14.4 Creating a DTM Object

With the DTM image created and the DTM transform defined, the next step in the DTM registration process is to create the RCDE terrain model object that associates the transform and DTM image (Step 3). Because the RCDE contains functionality specific to the DTM, the system must be explicitly told which image contains the elevation data and which transform defines the registration of that image. An instance of the class REGULAR-GRID-TERRAIN-MODEL, or one of its subclasses, is created to provide a framework for these DTM system components.

The function **make-terrain-model** takes an elevation image **dtm-image** and a transform **dtm-frame-or-matrix**, which is a matrix (two-dimensional array) or an instance of the class 4X4-COORDINATE-TRANSFORM. It creates a REGULAR-GRID-TERRAIN-MODEL containing the parameters. The elevation image becomes the DTM data set, while the input transform describes the relationship between the DTM image and the site.

The DTM input matrix must have scaling incorporated into the rotation component (this also provides a more efficient DTM transform). The translation component is unchanged, as it is originally specified in site units. This scaled transform matrix is used in most computations relating to the DTM, such as those required when moving objects about on the DTM surface, or interpolating elevation values for selected pixels.

14.5 Linking the DTM to the Site

The final step in the terrain registration process is associating the DTM image and its transform to a world (Step 4). A 3D-WORLD object may contain a terrain model as an element on its property list. Adding the terrain model to the world's property list establishes the terrain model as ground truth for that site. This is accomplished with standard property-list syntax in Lisp, and with the function *put_prop* in C/C++.

For example, in Lisp syntax the following form establishes ***alv-windowed-terrain-model*** as the terrain model for *alv-3d-world*:

```
(setf (get-prop *alv-3d-world* :terrain-model) *alv-windowed-terrain-model*)
```

In addition, the terrain image may be registered to the world as an image itself, so that it may be displayed as a normal image in the world (i.e. any modeled objects in the site will appear on the DTM image). The function **setup-image-worlds** creates the proper entries in the various slots of the image and the world objects so that the view containing the image is included as part of the given world. This function can be used to associate any image-transform pair with an existing world, thereby creating a view of that world.

For example, the Lisp call to link the ALV terrain image into the ALV world ***alv-3d-world*** is

```
(setup-image-worlds alv-dtm :3d-to-2d-projection alv-dtm-projection
                            :3d-world *alv-3d-world*
                            :2d-world nil)
```

These variables (and this function call) are defined in the ALV file **$CMEHOME/alv/alv-camera-models.lisp**.

The terrain model registration for the ALV data set should now be complete. From any view of the site, it should be possible to move objects on the DTM, and to find the elevation at any visible point in a view with DTM coverage. The terrain image can also be viewed in a pane by using the **push-image** function (this must be performed *after* the call to **setup-image-worlds**).

Chapter 15

Building a Site Model

This chapter contains a comprehensive overview of the complete process of building a site model in the RCDE. While other chapters have detailed each part of the RCDE covered in this chapter, an end-to-end scenario illustrating the precise steps necessary to build site models may be helpful to those unfamiliar with the RCDE and Lisp programming.

For those who prefer to examine Lisp code directly, the files containing site model "definitions" are helpful. For the ALV site, the file "$CMEHOME/alv/alv-camera-models.lisp" contains definitions and function calls to build the ALV site from a few files of Lisp code.

There are six major stages of site model construction covered in the following sections:

1. Creating a new site object;
2. Registering images to the site;
3. Registering a terrain model to the site;
4. Constructing 3-D models of site objects; and
5. Saving the site model; and
6. Reloading the site model.

Figure 2.5 provides a graphical overview of the site model construction process.

15.1 Creating a New Site Object

The **site** object, or **3d-world**, provides a means for grouping objects within a session of the RCDE into distinct sites. All objects within a site have a pointer to the site object containing them.

Initializing the Frame, View and Image For a site to be visible to the user, it must have at least one **view** of the site loaded into a pane. Make a frame using the menu option Misc→CME Frames→Make Frame. This frame will contain the views of the new site, although they could also be placed in any existing frame. Frames are purely I/O devices, and are not associated to worlds directly.

Load an image containing a non-nadir view of a building using I/O→Load Image, placing it in a pane in the new frame.

Initializing the Site and Camera Model The next step involves creating a new 3d-world object, and a default camera model for the loaded image.

Select Transforms→New View Transform on Image. Look for a cue on the documentation line to "Pick Image For New Transform". Select the pane containing the recently loaded image, and a menu of all existing 3d-worlds will appear. At the base of the menu, there is an option to Make 3d World. Select this button; another menu should appear with a string editor to enter the name of the new 3d-world object. Type in the name "Outland" (or whatever is appropriate) and press Do It. When the menu disappears, refresh the selected pane by clicking the middle mouse button while the mouse is on the pane. The camera icon and a coordinate system axis will then appear in the pane containing the image.

This process explicitly creates a new 3d-world, but it also creates a new camera model for the image and a new 2d-world object to associate the new camera model with the image. The site coordinate system and model-to-image transform (camera model) are "default" values at this point. The user must specify that information later, either by typing in known camera parameters or by solving for a camera model using the resectioning functionality described in Chapter 13.

15.2 Registering Images

The details of operating the resectioning functionality within the RCDE are described in Chapter 13. Rather than repeating them here, this section focuses on how the first image of a site may be resected, so that it can be used as the reference camera model for resecting additional images later.

It is assumed in this example that terrain data or other absolute information is not available. Thus the camera models produced will be relative to one another, but will not be specified in world units such as meters.

In the view containing the loaded image, choose a prominent cultural feature, such as a large rectangular building, that has easily identifiable corners. Using Create Object→Box, create a building in the site near the stare point. Select the box by placing the mouse on one of its lower vertices. Using the box's object menu, set the (x, y, z) location of the box to be $(0, 0, 0)$. This operation will place a ground-level corner of the building at the site origin, providing a convenient landmark for the site coordinate system.

For convenience, the building's local coordinate system may be aligned with the site coordinate system. This is the default configuration, and does not need to be changed.

Using the same selected vertex of the box, resize it so that it is approximately shaped like the chosen building in the image. The image may now be resected by using the proportions of the building dimensions as a reference.

First, create a new 3D feature set using View→Feature Sets→New3DFS. Select this feature set, so that newly created objects will be grouped into it automatically. Next, for each corner of the building visible in the image, create a conjugate point directly on the corresponding corner of the building model. Note that the 3-D positions of the conjugate points must be specified – this can be done precisely by using the conjugate point object menus, or by using the @Vertex bucky option for the conjugate point. For each conjugate point, identify the image location of the correct position of the point using Conjugate Point→Move Conj Pt. The "X" for each conjugate point should be placed on the image location of the building corner corresponding to its model corner. Conjugate points should only be created for visible building corners, since conjugate points without adjusted 2D image locations will be assumed to be in their correct image location.

Resection may now be applied to the camera model through the menu option Conjugate Point→Resection Menu. The feature set containing the conjugate points may not contain any other objects, or an error will result. Initially, the "Random" search mode should be used, since no effort was made to manually position the camera at the correct orientation. Once a reasonable solution has been found, the "Improve" mode should be used to avoid throwing away the approximate solution.

More buildings may be added to the scene, and more conjugate points may be added on them to improve the camera model. Since ground truth is not known in this scenario, the distance between buildings must be judged relative to the size of the buildings as estimated from the imagery. The chosen buildings and their conjugate points should span the image, so that a more accurate camera solution can be derived.

When additional images are introduced, they may be resected with respect to the first image. New images are added to the site by using the same functionality described above – the image is loaded, a new view transform is created on the image, and the created site is selected as the world containing the new view. If site objects are not immediately visible in the new view, then the menu View→Feature Sets may be used to make the site feature sets visible in the new view by selecting the new view and pressing the Pres option for each 3-D feature set. The camera model for the first image may also be improved as additional images are introduced to the site.

15.3 Registering a Terrain Model

This step in site model construction is covered in detail in Chapter 14. The main difficulty in associating terrain to an existing site is finding the proper transform relating the position of the terrain grid to the site coordinate system, including the proper scaling factor. However,

there is no systematic means of doing this within RCDE; some a priori knowledge must be used, such as the latitude and longitude coordinates of the terrain grid in coordination with a geographically referenced site coordinate system.

15.4 Constructing 3-D Models

The world data structure is composed of collections of objects placed on the image pane. These collections are contained in the world as a slot called a Feature Set. Feature sets can be created using the menu interface; View→Feature Sets prompts for a view, and then provides a menu of feature sets associated with the world containing the view with options to create new feature sets. Create a new 3d feature set, or select an existing 3d feature set that does not contain the conjugate points created for resection.

The new view is now prepared to allow the creation of 3d objects using the menu interface. By default, objects created through the menus are automatically placed in the selected feature set. Only one 3d feature set may be selected per site.

Select the Create Object menu, and choose the House option to create a default house object. Look for a cue on the documentation line to "Pick a Pane and Position for New Object". Move the mouse to the pane and click left at the position where you wish to place the object; wait for the object to appear on the pane. Click left to drop the object. There may be a warning that the feature set is not visible on the view, but this may be ignored. Then select View→Feature Sets to toggle the visibility of the feature sets. Note that the house was placed in the feature set named "3d home," which is the selected feature set.

Repeat this creation process for other object types as desired. A feature set of the site may be completely populated in this fashion. New feature sets may be created using the View→Feature Sets menu, and any new feature sets may be selected to group additional objects separately.

15.5 Saving a Site Model

To save the site that has been created, use the menu option I/O→Save Site Model option. The Save Site menu that pops up allows a mouse selection of the chosen site to save by pushing the site slot. Look for a cue on the documentation pane to select the site by mouse or keyboard. Specify the site by mouse clicking left in the image where the site has been connected. Note the number of objects created; it should match with with total number of objects in the union of all the feature sets for the site. Specify the full pathname of the site with the file name ending with extension site, if you wish, then click on the SAVE SITE button. A message will indicate when the save operation is complete. The Lisp Listener will provide a message similar to the following:

```
Saving
   #<Outland 3D-WORLD #X3228056>
```

```
to File /home/zippy/RCDE/users/wsb/outland.site

Finished writing /home/zippy/RCDE/users/wsb/outland.site
>
```

The file containing the site is in FASD format, which is ASCII Lisp code that will reconstruct the site objects when evaluated by loading it back into Lisp. For more information on FASD files, see Chapter 16.

15.6 Loading a Site Model

Currently, menu support for loading site models is incomplete. While the FASD files output by the RCDE contain sufficient information for recreating the site and objects in the site, no information about the display of the site is explicitly stored. For example, the site files do not contain data giving the frame dimensions, which views are present in which panes, etc. The available menu options allow the loading of a site FASD file, but do not provide for display of the site.

For purposes of this scenario, it is assumed that this is a different session of the RCDE, so that the created world and its objects are not present in the system. If this is not so, there is no reason to load the site, because the associated data structures will be cached in the RCDE (even if they are explicitly deleted). If a copy of the stored world is to be loaded into the same session of the RCDE in which the world was created (or previously loaded), a new world may be created by editing the site file; just change the string names of the world objects, i.e. by appending "-new" onto all string names ("Outland" becomes "Outland-new").

15.6.1 Initialize The Frame

Create a new frame to hold the image and world that is stored, using the Misc→New CME Frame option. Configure the window with the desired number of panes and push the DOIT button. Select a pane where the site model image will appear.

15.6.2 Initialize The Site Model

Select the menu option I/O→Load Site Model. Use the choice of Other to get a file load menu called Load Site Model. Specify the directory and pathname of the site model file. Push the LOAD SITE MODEL button. This will load the site into the environment but not display it, because it has not been associated with a view. The Lisp buffer should contain a message similar to the following in response to the load command:

```
;;; Loading source file "/home/zippy/RCDE/users/wsb/outland.site"
```

15.6.3 Display the Site Model

Each image of the site may be displayed with site object overlays by pushing the 2d-world containing the image onto a pane. Initially, however, the images must be directly loaded into the RCDE. To load the images associated with a site, use the function **load-site-images** as follows:

```
> (load-site-images (get-3d-world-named "Outland"))
NIL
```

This function loads all the images contained in the site "Outland". It does not make them visible in any pane, however.

To display an image and its associated view of the site, it is necessary to return the image from a function call. This may be accomplished with the function **base-image**:

```
> (base-image (first (2d-worlds (get-3d-world-named "Outland"))))
#<BLOCKED-ARRAY-MAPPED-IMAGE 362 x 253 (UNSIGNED-BYTE 8) #X35531EE>
```

The image will be displayed in the selected pane, with its name at the top of the pane. The image is fully integrated with the "Outland" site. If any other 2d-worlds exist in the site, this procedure may be repeated for each of them, replacing "first" in the Lisp statement with "second", "third", etc. They may be displayed in separate panes by selecting different panes between calls to **base-image**.

To see the site objects in the loaded view, select the menu option View→Feature Sets and click on the pane containing the view. Click on the Pres button for the 3d feature sets in the site to see all of the 3d objects created for this site. Each 3-D feature set created in the original site should have an entry on the menu.

Chapter 16

Data Exchange

This chapter describes methods for capturing objects within a file for the purpose of subsequent restoration or insertion into the RCDE. The methods described are extensible and may be used on objects native to the RCDE Lisp environment or on data structures defined in a C or C++ process not native to the RCDE. Methods for handling such non-native objects are described in Section 16.2.

The following sections examine three approaches to RCDE data exchange. The first is a direct transfer mechanism of native RCDE data. The second uses the RCDE's Lisp-C/C++ interface to allow the direct programmatic transfer of such objects within the RCDE, while the third approach uses file-to-file transfer external to the RCDE.

16.1 Transfer Between RCDE Sites: FASD Files

Currently, the fundamental means of permanently storing native RCDE objects is FASt-Dump (FASD). Given a set of RCDE object instances, this Lisp facility creates Lisp forms that are saved to a file. The Lisp forms can then be loaded back into the system at a later date. The term FASD is a holdover from the Symbolics environment, where the concept of a FASD was integral to how the Lisp compiler worked[1]. The basic idea behind FASD is that every object should be able to create a textual expression (S-expression), that, when evaluated, produces a deep copy (i.e., no common sub-components) of the original object.

16.1.1 Creating FASD Files

A FASD file may be programmatically created using a command script illustrated below, but the RCDE environment also provides a menu interface to load and save particular FASD

[1] FASD-FORM is the Lisp command which is used to invoke the FASD functionality.

files. Specifically, the menu functions Save/Load Feature Sets and Save/Load Site Model provide the user interface to the RCDE's FASD facility.

The RCDE currently implements the FASD facility. When a user executes

```
(fasd-form thing)
```

the FASD command outputs Lisp forms which, when executed, restore the object (i.e., thing) within the RCDE. To create an ASCII file containing these Lisp forms for subsequent object restoration, the user must explicitly save the FASD output to file using the RCDE menus, editor commands, Lisp commands, or C function calls. If the RCDE is being run under Emacs, then the user may simply employ editor commands to manually cut the text of a FASD form from the RCDE Emacs buffer and paste it into another buffer. Lisp programmers may use Lisp syntax to redirect FASD output to a file, since FASD produces a Lisp expression. For C/C++ programmers, the **fasd-form** command can be called from C directly; it returns a character string containing the FASD form, which can then be routed to file using C/C++ syntax. This again saves objects in the Lisp environment.

For instance, a simple Lisp statement that saves the FASD form of thing to the file **filename.lisp** is

```
(with-open-file (stream "filename.lisp") (write `(setq *restored-thing*
,(fasd-form thing)) :stream stream))
```

C/C++ FASD Formatting FASD files may also be produced using an independent program written in any language, such as C or C++. The file format must be written so that Lisp can effectively execute the data as code, but the Lisp loader allows some flexibility, in that whitespace is ignored and the order of slots within a FASD class instance is not specified.

16.1.2 Loading a FASD File Into The RCDE

An ASCII file containing FASD forms may be loaded during a subsequent interactive RCDE session to reproduce the environment in virtually the same state as it was saved. This ability can significantly reduce development time and improve research productivity. The RCDE does not require any special naming conventions for these files, so the developer is free to choose a name which best describes its content[2].

Since a FASD file is an ASCII file of Lisp executable commands, it can be loaded directly into the Lisp environment as Lisp code. This is done by executing the the following command sequence on the Lisp command line:

```
(load "filename.lisp")
```

Alternatively, a C/C++ programmer could call the C version of the RCDE function:

[2]The use of the .lisp suffix is recommended.

```
(in-package 'cme)
(append cme::*object-feature-sets* (list
(MAKE-INSTANCE '3D-FEATURE-SET    ;; Feature Set - a collection of related
                                  ;; Objects. There are currently 3 types:
                                  ;;                3D-FEATURE-SET
                                  ;;                2D-FEATURE-SET
                                  ;;                WINDOW-FEATURE-SET
               :WORLD (GET-3D-WORLD-NAMED "Alv 3d World")
               :INFERIORS
 (LIST
  (MAKE-INSTANCE HOUSE_OBJECT ...)   ;; see related tables
  (MAKE-INSTANCE HALF-CYLINDER ...)  ;; see related tables
  (MAKE-INSTANCE SUPERQUADRIC ...)   ;; see related tables
  (MAKE-INSTANCE RIBBON-CURVE ...)   ;; see related tables
  :PROPERTY-LIST
  (LIST ':DRAWING-COMPLEXITY 'NIL)))
))
```

Figure 16.1: Generic FASD Example

```
load_file("filename.lisp");
```

If a developer has many such files to load, it is recommended that a command script be created containing the names of the files. This command can then be executed within the Lisp environment to load selected FASD files.

16.1.3 FASD File Format

The following sections detail the contents of a FASD file, and how to re-instantiate its contents within the RCDE. The file format of each object is a Lisp form that begins with the symbol **make-instance**. This is the CLOS command for creation of class instances[3]. Figure 16.1 shows the form for making multiple instances (the actual object code has been elided for clarity).

The **make-instance** form creates class instances. Tables 16.1 through 16.4 illustrate the FASD representations of several basic 3-D objects, along with a description of the object slots. Note that the Lisp forms should be inserted into Figure16.1 for completeness. Each word preceded with a colon corresponds to a slot within that class. The values shown are instance-dependent and will change between applications. The order of the slots within a class is also arbitrary. The **make-instance** Lisp form is followed by the symbol **quote**,

[3] For a deeper insight into CLOS, it is suggested that the text Understanding CLOS The Common Lisp Object System by J.A. Lawless, and Molly M. Miller and published by Digital Press be examined.

then the Lisp class name in parentheses. This is the textual representation of the Lisp **quote** function and is used to prevent the Lisp interpreter from attempting to evaluate what follows as a function, a variable, or a constant expression.

16.1.4 FASD Limitations

The FASD facility does not handle data structures with circular references in a general way. In the version of Common Lisp described in the 2nd Edition of Steele's *Common Lisp: the Language*, there is a function **make-load-form** that addresses the whole issue with a more complete perspective. Additionally, Lisp data representations not native to the RCDE will not be loadable through this mechanism. It will be necessary for Lisp users to implement the definitions and FASD Generators of non-native RCDE representations in the RCDE environment, or Lisp users will have to implement code translators to native RCDE representations.

16.2 Transferring Objects To and From C/C++ Systems

There are two methods to transfer data from C/C++ systems into the RCDE: file-to-internal and file-to-FASD. Transfer efficiency and code reuse are deciding factors for which method to use.

File-to-internal Typically, the user's C/C++ system already contains routines to read and write a specific non-FASD file format. In this case, the existing user routine to read the user's file format may be loaded directly into the RCDE, with the appropriate files including the user's internal data structures (*.h files).

The user then adds code to copy the internal data representations into native RCDE data structures, this process will have to happen regardless of the transfer mechanism, if the data are going to be used by the RCDE. The reverse process to copy from the RCDE to internal user representations may also be implemented to allow the exporting of data. The coding efficiency and reuse are maximized with this technique if the internal data transfer is easier to implement than a direct routine to produce FASD output from C data files. In general, the representational transformation is being performed between internal representations, rather than through the FASD file format mechanism.

File-to-FASD The user could write specialized data translators to transform user data files directly into a FASD file. The FASD file could then be loaded directly into the RCDE. This alternative may be more expedient than the File-to-internal method. However, it is clear that this method is not as efficient because of its file-to-file nature, and it requires the production of "Lisp-Like" data files which are foreign to C and C++.

16.2.1 Restoring Objects from C/C++ Files or Programs

The RCDE will permit two alternative approaches to loading image files or on-line objects produced in C-based systems into the RCDE environment. One method, called *file-to-FASD*, involves the C developer writing a program which creates a FASD file with the format and content described in Section 16.1.3. The C program may read an existing file and translate it into FASD format, or it may be a subroutine attached to an image or modeling system containing the data to be transferred to the RCDE. The FASD file so produced can than be loaded into the RCDE environment. For IU researchers not working in Lisp, the RCDE supports object archival and restoration by calling the **fasd-form** function from C or C++.

An alternative to the file-to-file transfer strategy is the direct link strategy, called *file-to-internal* method. With this approach, the C user would still be expected to supply interface software, but would not be required to create a FASD file with its associated Lisp command structure. More specifically, the user must supply C or C++ code to copy his internal data structures into the RCDE's internal data structures. This is a standard data transfer between different representations, but since the C code is loaded into Lisp it takes place *within the Lisp process,* without file overhead.

Once this is done, normal C calls to routines within the RCDE environment will be sufficient to load all existing C based files and/or system resident data into the RCDE for display and analysis. To assist in the implementation of this strategy, the RCDE provides both hard and on-line documentation of each interface routine and its associated calling sequence. Using this strategy, the Lisp nature of the RCDE environment will be made totally transparent to the C developer.

The advantages to this scheme include performance and code reuse. While the data-intensive nature of image processing can cause file-to-file transfer to be unacceptably slow, the inter-process connection created by the LCI Loosely-Coupled mode allows approximately average Ethernet speeds (2 MBits/sec) for image transfer between Lisp and C.

Table 16.1: House FASD Example

```
;;; FASD EXAMPLE
(MAKE-INSTANCE (QUOTE HOUSE-OBJECT)
     :ROOF-PITCH (QUOTE 0.5)
     :ALBEDO (QUOTE 1.0)
     :X-SIZE 19.837037984153995
     :Y-SIZE 19.52734631894257
     :Z-SIZE 15.064418246990208
     :TAPER-RATE (QUOTE NIL)
     :OBJECT-TO-WORLD-TRANSFORM
     (MAKE-INSTANCE (QUOTE 4X4-COORDINATE-TRANSFORM)
                    :TRANSFORM-MATRIX
                    (MAKE-AND-FILL-4X4-MATRIX
                      0.86086789054  0.5083565589 0.02191081935 -123.2596374
                     -0.50869196365  0.8608361500 0.01391433934  4820.974567
                     -0.01178817971 -0.0231242656 0.99966309682  6072.100011
                      0.0            0.0          0.0            1.0      ))
     :WORLD (GET-3D-WORLD-NAMED (QUOTE "Alv 3d World")))
```

ROOF-PITCH	Roof angle in radians
ALBEDO	Ratio of Reflection to Incidence (value between 0 and 1)
X-SIZE	Length of side X measured in site units
Y-SIZE	Length of side Y measured in site units
Z-SIZE	Length of side Z measured in site units
TAPER-RATE	Rate of object taper moving along the z axis. Default is no taper (i.e., taper rate = 0)
OBJECT-TO-WORLD-TRANSFORM	The rotation and position of an object coordinate system with respect to a world coordinate system expressed in matrix form. This matrix is used to transform the given object instances coordinate system to that of its attached world.
WORLD	Name of the site model containing the object

Table 16.2: Half-Cylinder FASD Example

```
(MAKE-INSTANCE (QUOTE HALF-CYLINDER)
               :ORDER (QUOTE (3 12))
               :TESSELLATION-MODE (QUOTE :LAT-LONG)
               :X-SIZE 35.212728995926014
               :Y-SIZE 46.29628560002381
               :Z-SIZE 17.606364497963007
               :TAPER-RATE (QUOTE NIL)
               :ALBEDO (QUOTE 1.0)
               :OBJECT-TO-WORLD-TRANSFORM
               (MAKE-INSTANCE (QUOTE 4X4-COORDINATE-TRANSFORM)
                              :TRANSFORM-MATRIX
                              (MAKE-COORD-FRAME-MATRIX
                               -223.90225672421684
                               4729.971732020333
                               6068.8284088525015
                               :Z-ROT -0.5333900014026516))
               :WORLD (GET-3D-WORLD-NAMED (QUOTE "Alv 3d World")))
```

X-SIZE	Length of side X measured in site units
Y-SIZE	Length of side Y measured in site units
Z-SIZE	Length of side Z measured in site units
TAPER-RATE	Rate of object taper moving along the z axis. Default is no taper (i.e., taper rate = 0)
ALBEDO	Ratio of light reflection to incidence (value between 0 and 1)
OBJECT-TO-WORLD-TRANSFORM	The rotation and position of an object coordinate system with respect to a world coordinate system expressed in matrix form. This matrix is used to transform the given object instances coordinate system to that of its attached world.
WORLD	Name of the site model containing the object.

Table 16.3: Superquadric FASD Example

```
(MAKE-INSTANCE (QUOTE SUPERQUADRIC)
               :X-EXPT (QUOTE 1.0)
               :Y-EXPT (QUOTE 1.0)
               :Z-EXPT (QUOTE 1.0)
               :ORDER (QUOTE 6)
               :TESSELATION-MODE (QUOTE :LAT-LONG)
               :X-SIZE 20.0
               :Y-SIZE 20.0
               :Z-SIZE 20.0
               :TAPER-RATE (QUOTE NIL)
               :ALBEDO (QUOTE 1.0)
               :OBJECT-TO-WORLD-TRANSFORM
               (MAKE-INSTANCE (QUOTE 4X4-COORDINATE-TRANSFORM)
                              :TRANSFORM-MATRIX
                              (MAKE-COORD-FRAME-MATRIX
                               -127.43829409356406
                               4711.981430409669
                               6051.898915464119))
               :WORLD (GET-3D-WORLD-NAMED (QUOTE "Alv 3d World")))
```

X-EXPT	Exponent of X component of superquadric equation. (default = 1.0)
Y-EXPT	Exponent of Y component of superquadric equation. (default = 1.0)
Z-EXPT	Exponent of Z component of superquadric equation. (default = 1.0)
ORDER	The number of faces used to approximate a complex 3-D object. The greater the value of order, the more accurate the representation.
TESSELATION-MODE	A mapping of rectangular graph coordinates into the coordinate points of a surface. (Possible Values: :LAT-LONG, :GEODESIC) (default LAT-LONG)
X-SIZE	Magnitude of the X coefficient of the superquadric equation measured in site units
Y-SIZE	Magnitude of the Y coefficient of the superquadric equation measured in site units
Z-SIZE	Magnitude of the Z coefficient of the superquadric equation measured in site units
ALBEDO	Ratio of light Reflection to Incidence (value between 0 and 1)
OBJECT-TO-WORLD-TRANSFORM	The rotation and position of an object coordinate system with respect to a site/world coordinate system expressed in matrix form. This matrix is used to transform the given object instances coordinate system to that of its attached world.
WORLD	Name of the site model containing the object.

Table 16.4: Ribbon-Curve FASD Example

```
(MAKE-INSTANCE (QUOTE RIBBON-CURVE)
               :VERTICES (MAKE-RIBBON-VERTEX-ARRAY-FROM-LIST
                          (QUOTE ((0.0 0.0 0.0 24.0)
                                  (87.07990264892578
                                     -32.76338577270508
                                     -15.647439956665039
                                     24.0))))
               :CLOSED-P (QUOTE NIL)
               :FILL-P (QUOTE NIL)
               :OPEN-FOR-VERTEX-MODIFICATION (QUOTE T)
               :OBJECT-TO-WORLD-TRANSFORM (MAKE-INSTANCE
                                           (QUOTE 4X4-COORDINATE-TRANSFORM)
                                           :TRANSFORM-MATRIX
                                           (MAKE-COORD-FRAME-MATRIX
                                            -294.8896620587175
                                            4556.413122432717
                                            6074.948873798788))
               :WORLD (GET-3D-WORLD-NAMED (QUOTE "Alv 3d World"))
               :PROPERTY-LIST (LIST (QUOTE RENDER-MODE) (QUOTE FILL)))
```

VERTICES	Points within a ribbon corresponding to the extreme points at either end of the ribbon and points of inflection within the ribbon
CLOSED-P	Whether or not the given ribbon is closed (i.e., the end points are joined) (default T)
FILL-P	Whether the given ribbon is graphically filled; default is nil.
OPEN-FOR-VERTEX-MODIFICATION	Whether or not the given ribbon can have its shape changed by mouse actions on its vertices (default T)
OBJECT-TO-WORLD-TRANSFORM	The rotation and position of an object coordinate system with respect to a site/world coordinate system expressed in matrix form. This matrix is used to transform the given object instances coordinate system to that of its attached world.
WORLD	Name of the site model containing the object.

Chapter 17

Lisp-C/C++ Interface

A primary goal of the RCDE is to support the two major programming language families used within the IU community: Lisp/CLOS and C/C++. Code developed in one language must be shared with the other language in a natural way. In doing so, the system must also supply developers with their accustomed level of debugging support. Since the RCDE is an object-oriented system, cross-language support is available for both the object-oriented and structured programming paradigms.

This chapter supplies an overview of the RCDE Lisp-C/C++ Interface (LCI) from a user's and programmer's perspective. That is, details of the interface's construction are provided only where necessary for an understanding of executing and programming the environment. More programming details may be found in the Lisp/C++ Interface chapter of the *Programmer's Reference Manual* and in the *Lucid Lisp Advanced User's Guide*.

17.1 LCI Overview

The LCI is based on the Lucid Common Lisp Foreign Function Interface[1] (FFI), which provides a means for interacting with other programming languages. The FFI gives Lisp the capability to load and execute C/C++ functions within the Lisp process, to dynamically link C/C++ modules and libraries, and to directly declare and access C data structures in Lisp. Using the FFI, C/C++ functions may be called from Lisp, and Lisp functions may be called from C/C++. The main limitations of the FFI are that it does not directly support C++ classes, although it will execute C++ code, and it does not provide support for debugging C or C++ code executing within the Lisp process.

The LCI provides three major enhancements to the FFI. The first is the ability to execute C/C++ code in a separate process, allowing the use of standard C/C++ debuggers. The user may seamlessly switch between execution in a separate debugging process and execution

[1] The FFI is detailed in the *Lucid Common Lisp Advanced User's Guide*.

within the Lisp process. The second is the introduction of several special data types to support the natural programming style in both language families. The third is special support for passing large blocks of data between Lisp and C/C++ at average Ethernet rates when C/C++ code is executed in a separate process.

To allow execution of C/C++ code in a separate process, the LCI provides two main code development modes and a way to switch between the modes without any code changes. In either mode, the LCI allows a natural programming style for C/C++ as well as for Lisp. Both modes support the data types introduced to enhance the FFI.

The first LCI mode, Performance mode, executes the user's C/C++ code within the Lisp process. This is accomplished by loading C/C++ code compiled into object form directly into the Lisp process. Compilation is first performed by standard C and C++ compilers, then all linking is provided by the Lisp system during the loading of object files. Since Lisp performs dynamic linking on C/C++ code, individual C/C++ files can be compiled, reloaded, and executed independently.

The second LCI mode, Debugging mode, executes the user's code in a user-controlled process separate from the Lisp process. Compilation and linking of C and C++ is performed by standard C and C++ compilers, while the Lisp process initiates control of these operations through interprocess communication. Because C/C++ code is executed in a separate process, standard C/C++ debuggers or development environments may be used for code development.

A system-level view of the LCI is illustrated in Figure 17.1. In the figure, the arrows indicate the direction of a function call or data communication. The terms used in the figure are explained in the following sections.

Both of the main modes allow the user to program a fine-grained interaction between languages or coarse-grained interaction between languages. The "graininess" of the interaction is the programmer's option, as it is unrestricted by the mode of coupling between the languages. In fine-grained interactions, a program makes frequent calls across the language interface, as one might use to access and modify single pixels of an image, one at a time. In coarse grained interactions, an entire image might be passed across the interface and be processed entirely in one language before it is passed back.

In either execution mode, C and C++ files may be compiled and linked or loaded together. K&R syntax is expected of C files, while ANSI C and AT&T C++ syntax is expected of C++ files. Function calls and data structures that are managed by the LCI do not require any syntax that is different from the language standards. To accommodate the differences among Lisp, K&R C, ANSI C, and C++ function calling paradigms, a set of conventions have been introduced and are itemized in this chapter and in the *RCDE Programmer's Reference Manual*. These conventions remain consistent with the proper programming styles of each of the languages.

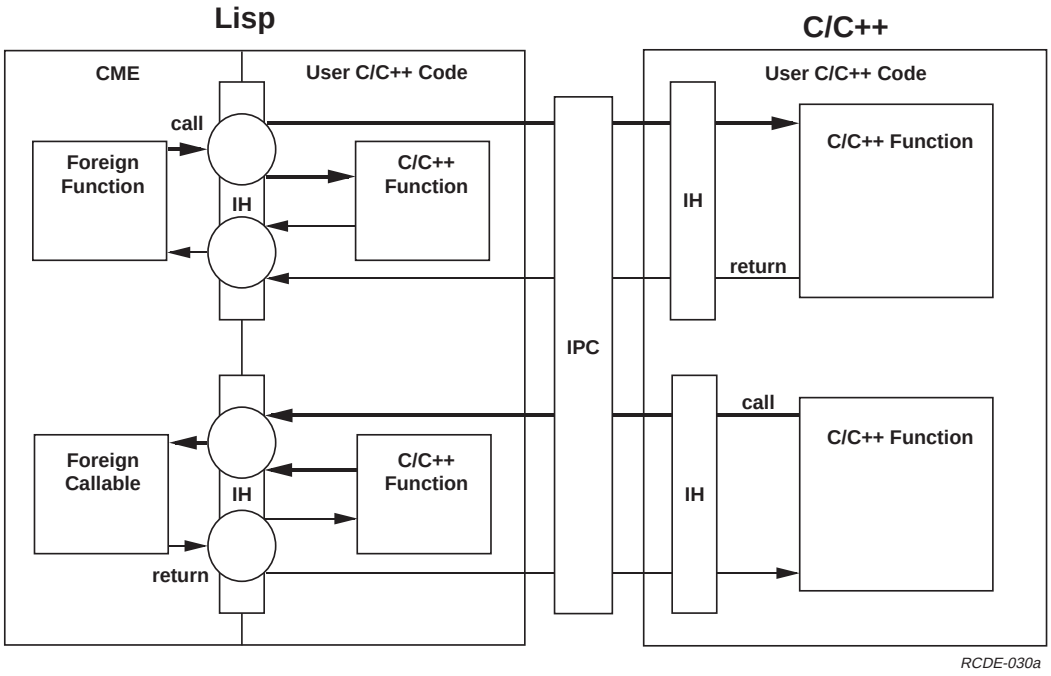


Figure 17.1: System Level View of Lisp C/C++ Interface Architecture. The figure shows both LCI modes and the switch between modes. C/C++ code can exist within the Lisp process and within a separate user process.

17.1.1 Executing C and C++ within Lisp: Performance Mode

The first LCI mode, more commonly called the *Tightly-Coupled mode*, makes use of the LCI to link Lisp and C/C++ code at the compiled-code level. The C/C++ code shares the same address space with Lisp, thereby providing a highly efficient way to couple the two languages from a run-time perspective. The Lisp environment supplies the linking facilities normally available through Unix. In addition, Lisp controls the execution of C/C++ code, replacing the standard C function *main()* with its own generic version. The user must rename the *main()* function in his source code to prevent a Lisp loading error.

It is difficult to debug tightly-coupled code because Lucid Lisp does not support an integrated debugger at this time. The existing Lisp debugger does not provide access to the C run-time stack, nor to C data that is not explicitly declared in Lisp. Therefore, the Lucid debugger does not offer much help in debugging C/C++ code. Run-time errors within the user's C/C++ code send the system into the Lisp debugger with very limited C information. Standard C debuggers such as **xdbx** cannot be used directly since tightly-coupled C/C++ code is embedded within the Lisp address space.

One potential solution is to debug the entire Lisp process with a standard C/C++ debugger. This strategy is currently infeasible because existing C/C++ debuggers cannot decipher how Lisp maintains C data in its foreign symbol table. In addition, a huge amount of debugging overhead would created; sorting through all of the Lisp process information to find the actual foreign C code and symbol table information would require detailed coding within the core Lisp implementation.

17.1.2 Executing C and C++ Outside Lisp: Debugging Mode

A more viable solution for debugging is to execute Lisp and C/C++ code in separate address spaces with some *invisible* means of communication between address spaces. This type of execution is provided by the LCI in the RCDE, and is usually called the *Loosely-Coupled* mode. C/C++ code that will be tightly coupled in its final form is temporarily executed outside of the Lisp process at the programmer's request. Control of the normal execution flow is maintained by Lisp through interprocess communication (IPC), but when the C/C++ process is executing it can be interrupted by the user for debugging purposes. The Lisp process waits, undisturbed by the interruption in program sequence; when execution is resumed, the program control flow continues normally.

The main advantage of this approach is that C/C++ code can be linked and debugged using standard UNIX tools, providing a familiar environment to the non-Lisp programmer. In addition, errors in the C/C++ process do not crash the Lisp process, and performance may benefit from multithreaded architectures. The Loosely-Coupled mode provides a trade-off between debugging functionality and reduced execution speed. However, the Loosely-Coupled mode may provide satisfactory performance for most development activities. It is assumed that the typical user will switch to the Tightly-Coupled mode for final execution and demonstration purposes.

17.1.3 Switching Between Modes

A key feature of the Lisp-C++ interface is that the user can switch easily and invisibly between Tightly-Coupled and Loosely-Coupled modes without modification of the user's code. The LCI either provides or automatically generates the interface functions necessary for communicating between the Lisp and C/C++ processes. Although there is significant overhead in communicating between the processes, the Loosely-Coupled mode is designed primarily for debugging. After the user finishes debugging, the C/C++ modules are then tightly coupled for performance. The switch actually takes place by selecting a menu option to change between Performance mode and Debugging mode. That switch simply changes a global Lisp variable causing the interface to change its mode of communication.

17.1.4 Programming C and C++ in the RCDE

The enhancements made to the basic Lucid FFI include: handles, Lisp types that C does not support directly (*symbols* and *forms*), and syntax for automatically passing arrays. Because of these enhancements, several new concepts were defined for the LCI. This section briefly describes these concepts and related terminology. At this point it should be clear that there are two ways to refer to the coupling modes: the Tightly-Coupled mode is also called the Performance mode in the menu options, while the Loosely-Coupled mode is also called the Debugging mode in the menu options.

Code Reuse: Proxies In either coupling mode, variables, functions, methods, and classes may be declared in one language that serve only to communicate with corresponding *real* variables, functions, methods and classes in another language. Such *stand-in* functions and classes are called *Proxies*, and support the concept of **code reuse**. A C++ proxy class looks and acts like a real C++ class. A Lisp proxy class looks and acts like a real Lisp class. Such class proxies may be used in the same manner as any normal class in the language. To access this reuse capability, the declaration header (*.h) files for these classes are included in the code, as any external library would be. In Lisp these definitions are loaded as standard Lisp definitions.

Proxy classes in either language have similar structure and format. All methods of a proxy class are functions that communicate through the interface; proxies derive their functionality from a real function in the other language. Proxy classes have only one direct slot, a pointer to the corresponding object in the other language, and access data slots through accessor functions that communicate through the interface.

Callables and Foreign Functions Callable functions are functions that have an interface function defined specially for communication between languages; they provide direct access from one language to a function written in the other. As described above, callable functions are one kind of proxy (proxy functions). Lisp-Callable functions allow the user to make a

Lisp function call to a C/C++ function. Each C/C++ system integrated into Lisp must have at least one Lisp-callable defined to initiate C/C++ execution from Lisp.

C-Callable functions allow the user to make a C or C++ function call to a Lisp function. The *RCDE Programmers Reference Manual* describes the RCDE C-Callables and conventions for calling them. These C-Callables are described as "manual page" entries and may be viewed as a library of C functions providing the basis of C/C++ access to RCDE. More generally, any Lisp function (even non-RCDE) may be called if a C-Callable form is created for it. The RCDE provides a menu interface to extend (by automatic code generation) the language interface by adding any new Callables the user needs to access functionality in the other language.

When Callable forms are created, they are evaluated by Lisp. Their textual textual forms may be extracted from the Lisp buffer (when running the RCDE through Emacs) and placed in a file for later loading as part of the interface layer.

A System Level view of the LCI Performance mode is illustrated in Figure 17.2. The diagram graphically illustrates the major concepts discussed in this section. In the figure, the arrows indicate the direction of a function call while the oval tails indicate where the return value is deposited. The return value may be a symbolic handle or actual data that is shared depending upon the return type defined for that function call.

Returned Data: Handles or Shared Structures In Figures 17.2, 17.3 and 17.4 the oval tails on the arrows indicate where returned values are deposited from a function call. The LCI provides the user with the ability to manage returned data in two ways, using *handles* and *shared structures*. A handle is a symbolic reference to a data structure or an object instance that resides in the other language. The language maintaining the handle does not have an explicit representation for the data structure in the other language. Instead, the handle provides a way to uniquely identify a data object in the other language. When this identifier is passed as a parameter to a function in the other language, the object specified by the handle is automatically retrieved and may be used normally in the function. When a handle is specified as a return value from a Callable function, a unique identifier (handle instance) for the returned object is created and returned to the calling language.

Handles are defined in both C/C++ and Lisp to point to objects in the two languages. When a C-Callable has an argument or return value that is a handle, its type is identified in C/C++ as a *c_handle*. The Lisp type *lisp-handle* declares a pointer to a C structure. Such a handle is used to pass C structures to Lisp-Callables or as a return value for a Lisp-Callable function.

A handle "points" to a single copy of a data structure, object or symbol. The data content is changed by providing a function or writer method to call the other language with new values to replace the old values. Since a single copy of the data exists, no question of consistency between copies arises.

Shared Structures are explicitly declared types provided by the LCI to allow large or complex data structures to be passed between languages easily, in a single function call.

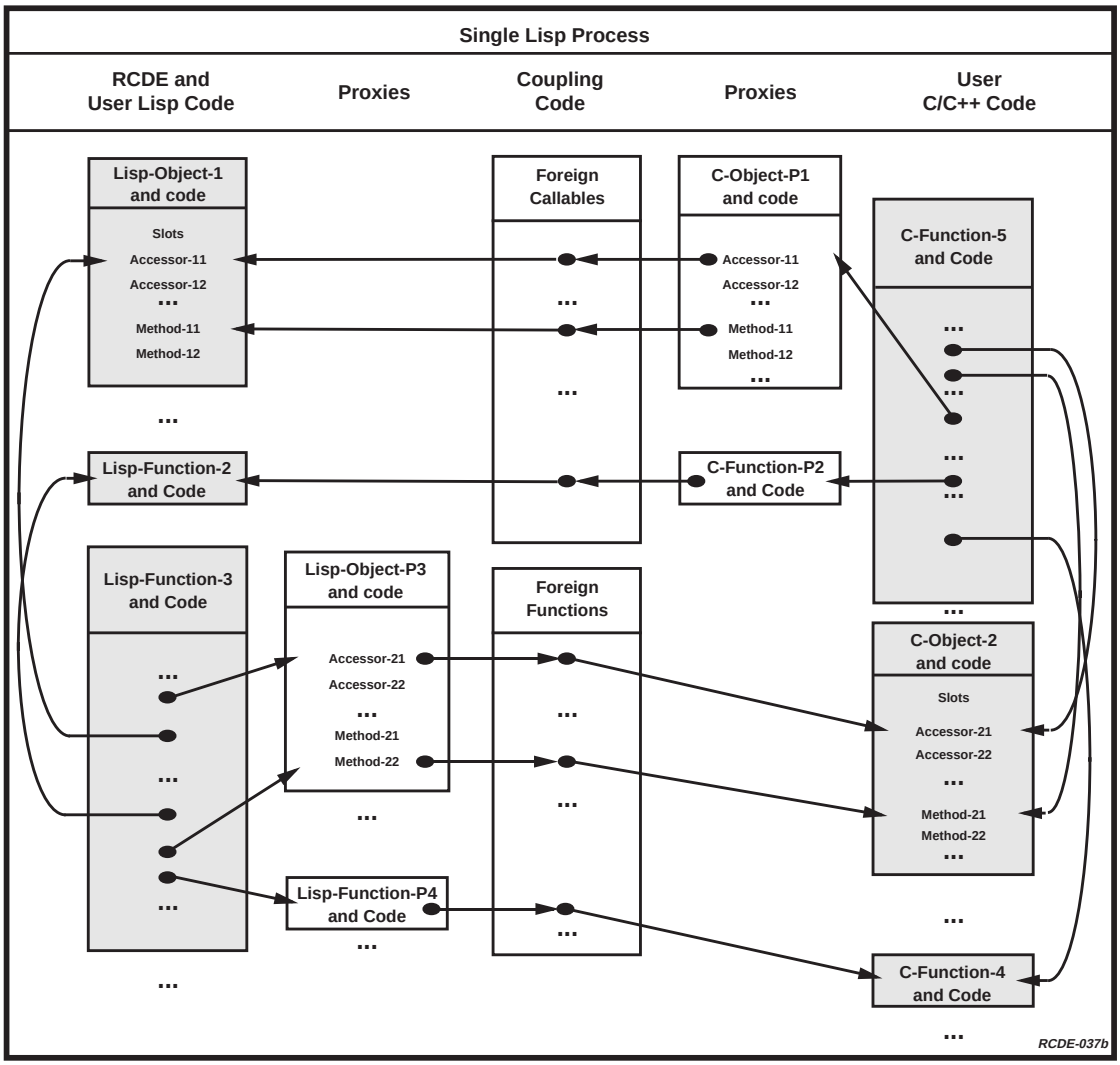


Figure 17.2: System-level view of Performance (Tight Coupling) Mode

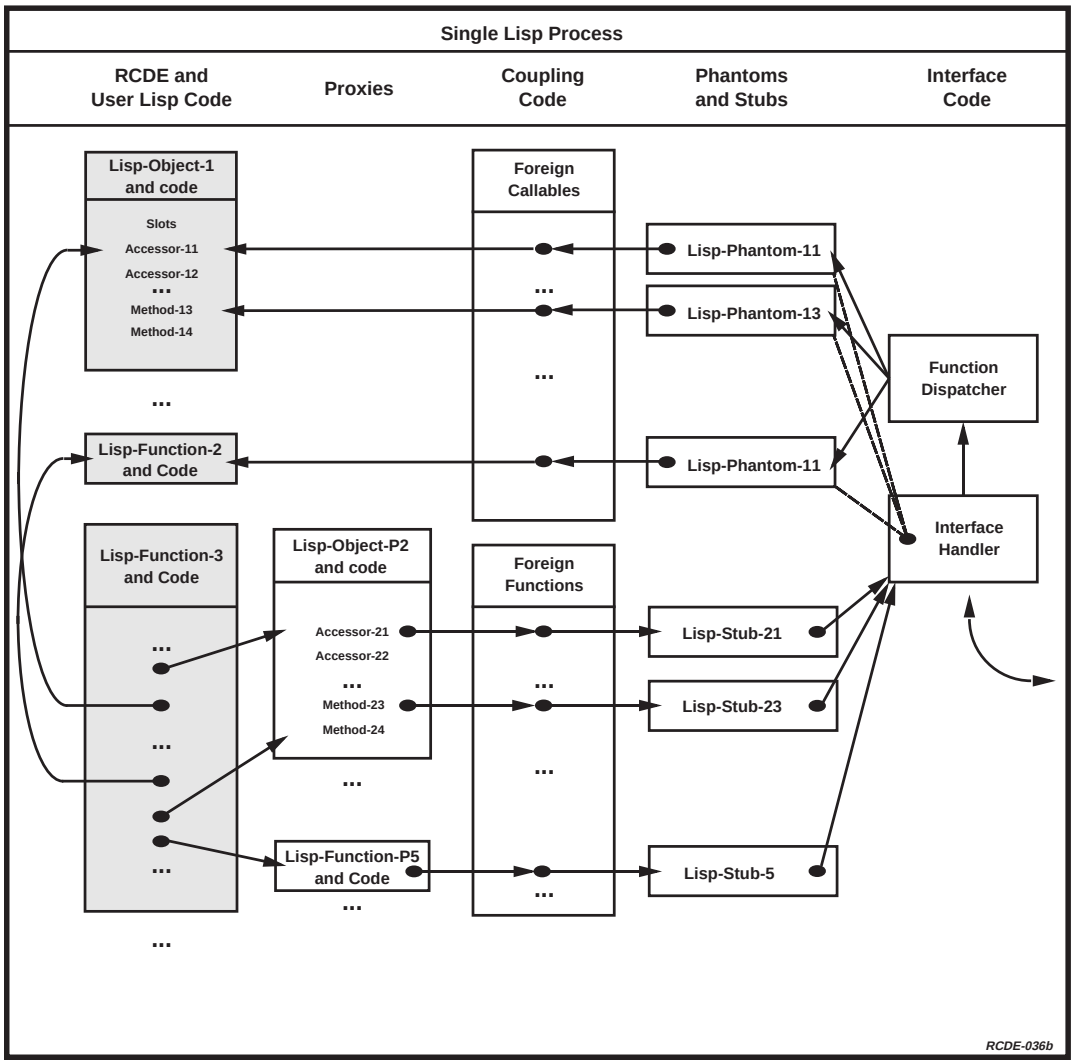


Figure 17.3: System-level view of Debugging Mode (Loose Coupling) *in the Lisp Process*

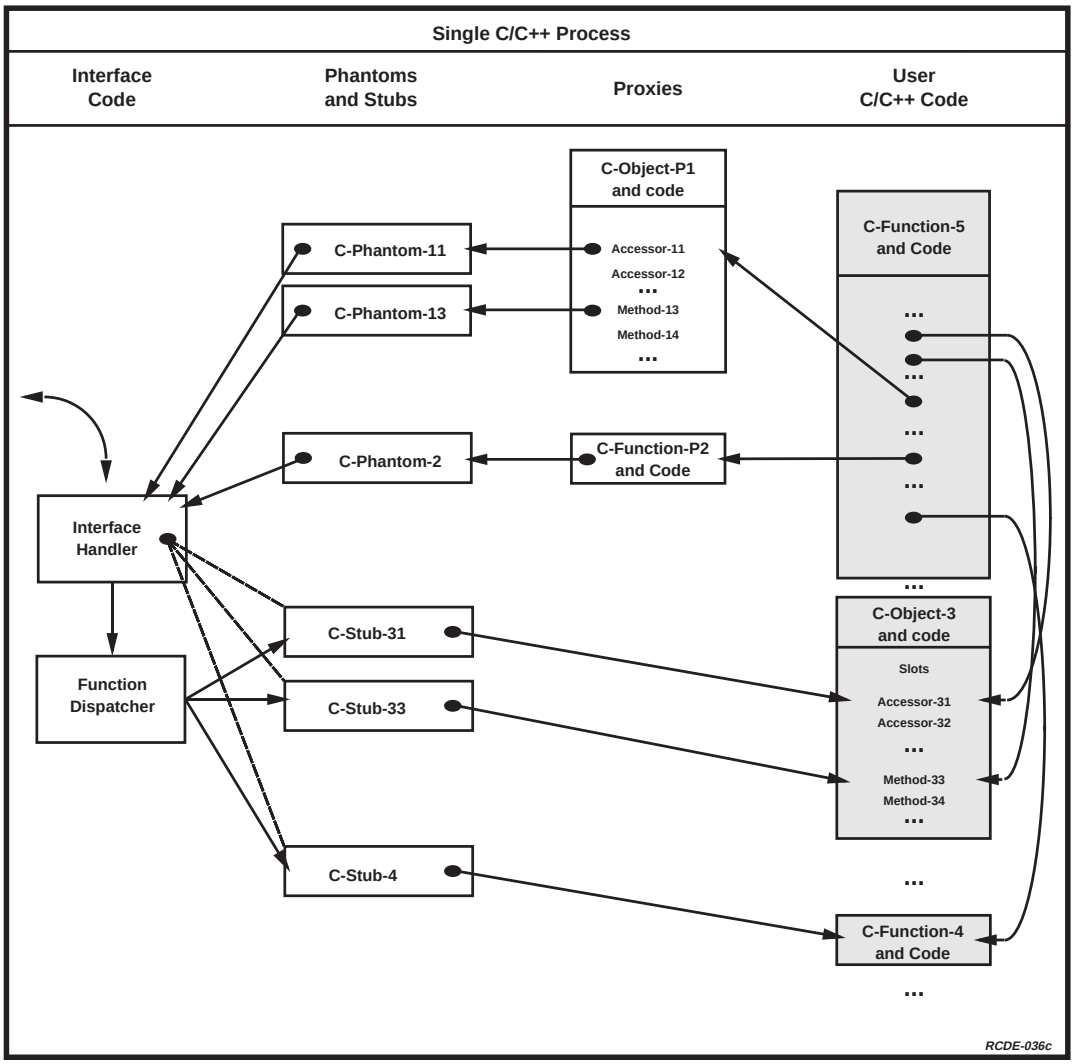


Figure 17.4: System Level View of Debugging Mode (Loose Coupling) *in the C/C++ Process*

They are declared as C structures, so that C and C++ may access their slots (data fields) using standard C operators. Through the FFI, Lisp may also directly access slots of shared structures using only Lisp function calls. The FFI provides the capability for shared structures to be passed between languages in Tightly Coupled mode, while the LCI adds this capability for Loosely Coupled mode. They are supported as parameter and return types for functions calls in both directions between languages.

Unlike handles, shared structure representations are completely declared in each language, so that both Lisp and C/C++ have detailed knowledge of the structures. This allows entire structures to be exchanged between languages, rather than only basic types or pointers (handles). Many shared structures are defined in the RCDE; most are for returning multiple values from RCDE functions, but there are also shared structures for transferring images, transforms, and polyhedral vertices between languages. In addition to the supplied shared structures, the user may add his own shared structures to the LCI.

In Tightly-Coupled mode, shared structures are passed by reference between functions. Changes made to the data structure by C/C++ are *immediately* visible from Lisp, since only one copy of the data structure exists, and both languages have direct access to it. In Loosely-Coupled mode, shared structures are passed by value. For an instance of shared structure passed between languages, two copies of the data exist, one in Lisp and one in C/C++. When the content of the shared structure is changed in the native language, no corresponding change takes place in the other language. It is necessary to return the updated copy of the content to the original language to maintain consistency. *It is the user's responsibility to maintain this consistency.*

Loose Communication Layer: Phantoms and Stubs When the user switches from Performance mode to Debugging mode, direct calls to functions across the language barrier must be redirected through the interprocess communication layer set up by the LCI as illustrated in Figures 17.3 and 17.4. The following sections explain the new terms in the figures in more detail. The notation is the same as that of Figure 17.2 where the arrows are function calls and the ovals are the returns of the calls.

In Debugging Mode, any C/C++ function loaded into the Lisp process is no longer executed when its Lisp-Callable function is called (although its definition is not deleted). Instead, each C/C++ function call is routed to a *stub* function that transfers the function call and parameter data through the IPC to the C/C++ process. Similarly, calls to Lisp functions from C/C++ are routed to *phantom* functions resident in the C/C++ process to manage data transfer and program control flow to and from the Lisp process. Together, the phantoms and stubs provide the Loosely-Coupled IPC communication layer. Stub functions manage calls from Lisp to C/C++, and phantom functions manage calls from C/C++ to Lisp.

The communication layer needs to be generated and loaded before the user executes C/C++ code in Debugging mode. As long as the Callable function names and argument lists are not changed, or Callable functions are not added or deleted, the communication

layer (phantoms and stubs) does not have be regenerated, recompiled or reloaded. But if the user changes the function names being used, or the parameter lists of the functions, or adds calls to new functions between the languages, then the compilation process must also recreate the phantoms and stubs. The phantoms and stubs are generated automatically, and stored in the user's directory as C and object files.

The Loose Communication Layer communicates with the C/C++ process through a socket created by an RCDE-supplied *main()* function in the file debug-main.c. This socket port must be established by executing the C/C++ process before Lisp tries to make a communication link with the C/C++ process. Once Lisp triggers the communication flow, The *main()* function acts as a server, awaiting C/C++ calls from Lisp (client requests) and executing them as they are received. Calls to Lisp from C/C++ code, or *callbacks*, are managed as they are encountered. The LCI can handle any level of nesting of calls between Lisp and C/C++.

17.2 Operations Concept

This section is a description of how the user sets up the LCI and its parameter values as a development environment for C/C++ code. The LCI provides a menu interface to manage C/C++ code compilation, loading and execution. All LCI parameters, such as execution mode, lists of user source files, name of user's makefile, etc., may be set using the menus. All LCI operations, such as C/C++ code compilation and C/C++ function execution, may be performed by pressing LCI menu buttons. The details of the menu interface and LCI programming are described in later sections; this section is concerned with the conceptual process of integrating C/C++ into the RCDE.

The general steps for code development and execution using the LCI are:

1. Write C/C++ code that calls C-callable Lisp functions (e.g., any RCDE function) or uses the C++ proxy hierarchy (described below).

2. Specify the C/C++ (and possibly Lisp) files containing user source code, including compiled library files.

3. Create Lisp-callable functions for any C/C++ functions to be called from Lisp, or specify a Lisp file containing previously-created Lisp-callable definitions.

4. Set the LCI parameters as desired for compilation mode, working directory, makefile, etc.

5. Compile code; compiled files are automatically loaded according to compilation mode.

6. Execute code by calling specific C/C++ functions.

7. Edit code as desired.

8. Repeat to 5.

The user's C/C++ code should contain calls to the RCDE functionality, which can be viewed as a large library of functions, classes and type declarations. The RCDE functions and classes available to C/C++ are given in the *RCDE Programmer's Reference Manual*. The user may also supply Lisp source code to be integrated with the RCDE and/or the user's C/C++ code. Details of C/C++ and Lisp programming with the LCI are specified in Section 17.5 below.

The names of all source files are provided by the user through the LCI→Files menu, either relative to the working directory or with a complete pathname. The working directory may be specified on the LCI→Parameters menu. When using Performance mode, all C/C++ libraries (except *libc* and *libm*) must be specified for loading into the Lisp process.

Before user code is compiled through the LCI, Callable forms (C-callables and Lisp-callables) must be generated for any C/C++ functions to be called from Lisp, and any user-supplied Lisp functions to be called from C/C++. The C-callable forms for the RCDE functions listed in the *RCDE Programmer's Reference Manual* are distributed with the RCDE and do not need to be regenerated by the user. Both types of Callable forms may be created using the LCI→Functions menu. Generally, C/C++ users create Lisp-callable forms, which allow Lisp to call their C/C++ functions. Each C/C++ program integrated with the RCDE must have at least one Lisp-callable, so that C/C++ execution can be initiated (Lisp does not allow C/C++ code containing a *main()* function to be loaded).

When C-callables and Lisp-callables are created with the menu interface, the output Lisp form is printed in the Lisp buffer. If the RCDE is being run through Emacs, these textual Callable forms may be saved in a file by pasting the text into another Emacs buffer using standard Emacs editing commands. The file(s) of Callable forms should then be specified as Lisp source files, and loaded into the RCDE instead of recreating the Callable forms – i.e., Step 3 becomes the loading of a Lisp source file (LCI→Control Panel→Load Lisp Files). In any case, Callable forms must be defined before compilation by creating them or loading them from file.

The next step involves specifying the LCI parameters controlling compilation and execution. With the LCI→Parameters menu, the user determines the execution mode (Debugging or Performance) and compilation options. These parameters may be set or changed at any time. For details, see the LCI Parameters section below.

With the source files specified, Callable forms defined and parameter values set, the user is prepared for compilation (LCI→Control Panel→Compile). In Performance mode, only user C/C++ files are compiled, and the resulting object files are automatically loaded into Lisp. In Debugging mode, the LCI (optionally) generates and compiles four C files from the defined Callable forms, compiles user C/C++ code, and links them with other static object files into the external C/C++ executable. Compilation feedback output to *stdout* and *stderr* are displayed in the Lisp buffer. The compilation process will abort if a failure is encountered at any stage (such as a syntax or linking error).

When compiling in Performance mode and C/C++ libraries have been specified, the

libraries should be explicitly loaded after compilation has completed with the menu option LCI→Control Panel→Load C/C++ Files. This need only be done once in any RCDE session (unless the libraries are recompiled).

Once compilation has successfully completed, the LCI is prepared to execute C/C++ code. If executing in Debugging mode, two extra steps must be performed before any C/C++ functions are called: the C/C++ process must be started by running the generated executable, either in a debugger or from a Unix shell; and the link between the RCDE and the C/C++ process must be created by pressing LCI→Control Panel→Establish Interface. These two steps must be repeated each time the LCI compiles in Debugging mode.

Each C-callable function may be called from the menus using LCI→Functions→Lisp-Callable C/C++ Functions→Execute. C-callables may be called in any order, with arguments entered by the user. When an error occurs in user C/C++ code, the LCI response depends on the execution mode. In Performance mode, a C/C++ error will cause the RCDE to trap into the Lisp debugger (with printed instructions on how to restore Lisp to its normal operation). In debugging mode, the error may occur in the Lisp process or the C/C++ process. In the former case, the RCDE again traps into the Lisp debugger. In the latter case, the C/C++ process will halt (possibly with a core dump), returning to Unix or to an enclosing debugger.

The user may edit source code and recompile, completing the development subcycle. The LCI makes no restrictions on the user's choice of editor (although Emacs is recommended because of its ability to run the RCDE as an inferior process). The execution mode may be switched at any time, but this may require recompilation before executing functions in the new mode if Callable functions have changed.

17.2.1 Projects

An LCI *project* is a set of source files and LCI parameters defining the integration of a C/C++ program with the RCDE. It is intended as a convenient means to package lists of source files and other information, so that the user does not have to explicitly enter this data each time a new RCDE session is begun. Projects also provide a quick way to switch between C/C++ programs integrated with the RCDE without starting a new RCDE process.

Project files may be created by manually setting all source files and LCI parameters, then saving a project file using LCI→Projects→Save Project. With a project file, steps 2 and 4 above are replaced by loading the project file (LCI→Projects→Load Project). Note that loading the project does not load the source files into Lisp, however – this must be done during compilation or by using LCI→Control Panel→Load C/C++ Files and LCI→Control Panel→Load Lisp Files.

17.2.2 Executing without Compiling

Once user C/C++ code has reached a certain level of maturity, it may be desirable to execute it without recompiling in a new RCDE session. This requires a slightly different sequence of operations than those listed above. Assuming that a project file is defined and that Callable forms have been stored in a Lisp source file, the following steps may be used to execute C/C++ code:

1. Specify and load the project file (LCI→Projects).
2. Load all source files (C/C++ and Lisp).
3. Execute code by calling specific C/C++ functions.

17.3 LCI Scenarios

This section contains a set of examples of how to use the LCI, with emphasis on menu interactions. A running example is intended to guide the user through the menu structure. The menu structure is detailed in Section 17.4.

17.3.1 Preparation and Example 1

Example 1: Image handles, Image creation, Image modification, Simple Shared Structures

Example Overview To get started by using an example, copy the file

$CMEHOME/lci/examples/create_and_change.c[2]

into your working directory and then load it into an Emacs buffer. The header on the file should identify it as Example 1. The sample code contains programming examples that demonstrate usage of the LCI and the RCDE C library. Notice that the programming style does not reveal any calls to Lisp and adheres to conventional C syntax. The include file *rcde_types.h* contains definitions for all RCDE C types.

The first type declarations are c_handles for two images and two panes. The next declarations are a keyword argument pair for the C-Callable function **make_image** which will set all the initial pixel values for the image being created. Notice that the *initializer* sets all pixel values to the gray level 128. When *make_image* runs it returns a c_handle to the image in the RCDE.

After it runs, a function to choose where to display the image is called. This is a C function which in turn calls an RCDE function that prompts the user to select an RCDE window. Upon selecting a pane to display the image the RCDE call *push_image* puts the

[2]Substitute the path at your site for $CMEHOME

image on the pane stack and displays the image. Note: the second keyword, *refresh*, is used to make the pane refresh itself. Its boolean value was set to true in the declarations.

Next the C-Callable function **image_scale_rotate** is called with keywords one and two. This function produces more than just scaling and rotation when called with these arguments. Try calling it without the final y_scale keyword pair.

In the last two calls to *push_image* notice that the *duplicate* keyword is used to make sure that a copy of the image is pushed onto the stack. Try running this example without the third keyword pair for duplication of the image.

Finally, notice the second C function in this file, *pause_and_select_pane*. Above it is an *extern* declaration of a shared structure. That shared structure is returned by the C-Callable RCDE function **pick_a_pane_and_x_y**. That function returns a pointer to a shared structure containing a single handle to the pane chosen by mouse click, and two integers at the mouse location at the time of click and a third integer, hence the name *c_handle_1_int_3*. Notice that the handle component of the shared structure is called *h1* and is obtained using conventional C syntax. This example should be run with the variations suggested in both Performance and Debugging mode.

Preparation The following preparatory steps are independent of the interface mode:

1. Write the desired C, C++, and Lisp code, keeping it in one working directory. The *RCDE Programmer's Reference Manual* defines the functions that interface to the RCDE. This chapter uses example code from the directory

   $CMEHOME/lci/examples

   You may substitute your own code in place of the examples.

2. To use RCDE-defined types such as handles, user C and C++ code must contain the declaration

   ```
   #include "rcde_types.h"
   ```

   Any files containing this include statement should be compiled with the following option:

   ```
   -I$CMEHOME/lci
   ```

3. If you are using a Makefile, you must add a few lines to it to compile the interface-generated code and to link the object modules. Details of this process can be found in the file

   $CMEHOME/examples/make-c-debug

Set the Run Makefile switch accordingly.

4. Select LCI on the Menu Panel, which invokes the main interface panel.

17.3.2 Tightly-Coupled Example

Now load and run code in the Tightly-Coupled mode to demonstrate the basics of the interface:

1. Set compilation and execution parameters (be sure to hit enter on all menu string entries:
   - Select Parameters on the main interface panel to invoke the Parameters menu, which is described in Section 17.4.2.
   - On the Parameters menu, set the working directory to reflect where your code is located.
   - Set the Execution Mode to Performance (Tightly-Coupled mode).
   - Set the Run Makefile button to No.
2. Specify the user files to be compiled:
   (a) Choose Files on the main interface panel to invoke the Files menu, which is described in Section 17.4.1.
   (b) Choose C Files on the IF Files menu.
   (c) Type the file name

   ```
   create_and_change.c
   ```

   (d) Select Add File to register the file name. The file name should appear in the large window on the menu.
3. Compile and load code:
   (a) Invoke the Control Panel from the main interface menu.
   (b) Compile the code by selecting Compile on the Control Panel. The resulting object code is automatically loaded into Lisp.
4. Create Lisp-Callable functions:
   (a) Select Functions from the main interface menu.
   (b) On the resulting menu, select Lisp-Callable C/C++ Functions since we want to access user C code.

(c) The resulting menu should have a place for function names on it. Since we do not have any yet, the entry is blank.

(d) On the menu, type the function name:

```
create_and_change_image
```

Then select the Create option to get creation menu.

(e) Enter the function name and specify whether you have given a mangled name (No) and the number of parameters (zero, or you can leave it blank).

(f) Push the button labeled Create Entry Panel

(g) On the resulting menu, specify the types of each function argument and return type. In this case, there are no arguments and the return type is *int*.

(h) Create the interface function by selecting the button marked Create Lisp-Callable.

(i) Note that a Lisp form was generated in the ***cme*** Emacs buffer and loaded into Lisp. You may yank the Lisp-callable into another Emacs buffer and save it to a Lisp file containing all Callable functions for later reference, but it is not necessary.

5. The foreign function has the same name as you specified in the Functions menu above. To execute the compiled C function, select the Lisp-Callable C/C++ Functions to see the list of functions in operation. Push the button by the function name; after the name appears on the line, select Execute to invoke another menu. Select it and watch for the prompts. Or you may run the function from the Lisp prompt using a Lisp form:

```
(create_and_change_image)
```

Follow the instructions in the RCDE Documentation Line to finish the example. (This example will ask you to select an RCDE pane twice.) If your Menu Bar is covered, you will not see the prompts; just select a pane twice to see the resulting images. Expose the menu bar, then repeat the Lisp call.

6. To repeat the cycle, edit your code in an editor, recompile it with the Compile button, and run it again from Lisp. The editing can be done based upon the suggested changes to the code made above. Try other variations of your own invention.

17.3.3 Loose Example

Once the Tight Example is running smoothly, try compiling and executing the same program in Debugging mode:

1. Reconfigure the Parameters menu:

(a) Set the Execution Mode to Debugging.

(b) Set the Load User Object Code button to Yes.

(c) Set the Run Executable button to No.

2. Compile again using the Compile button on the Control Panel. In this case, four files are generated: *c-stubs.c*, *lisp-stubs.c*, *c-phantoms.c*, and *lisp-phantoms.c*. These are then compiled, and the two *lisp-*.c* files are loaded into Lisp. The two *c-*.c* files are linked into the executable created from the user's own code, plus some other (static) LCI files.

3. In an independent shell, run the executable produced by the compilation (the default name is *debug-main*, as defined on the Parameters menu). This can be done under any debugger. A message should tell you that a socket was created by giving you the port number (you do not need this information, however). You can even run the function without a debugger. Just type *debug-main* at a prompt in a UNIX shell.

4. Click on Establish Interface. Some handshaking messages should be printed in the ***cme*** buffer and in the shell telling you that the connection was safely established. This gives you evidence that everything is still operating as it should.

5. To execute the compiled C function, select the Lisp-Callable C/C++ Functions button to see the list of functions in operation. Push the button by the function name; after the name appears on the line, select Execute to invoke another menu. Select it and watch for the prompts. Or type

   ```
   (create_and_change_image)
   ```

   on the Lisp command line to run the same function as earlier. Once the IPC connection is made, the execution is the same at the top level as it is in tight mode. After running this smoothly in Debugging mode, simply select Performance mode and run it again, and back to Debugging mode and repeat. Do this sequence to verify that you can switch back and forth with no other changes and get the same functionality.

   You may then try the suggested edits to change the code. Each time you recompile, you must re-run *debug-main*, and push the Establish Interface button again. You can avoid the killing/restarting cycle by setting the Run Executable option, but then the C/C++ process will not be run inside a debugger, and its printed output will be dumped to the ***cme*** buffer.

   The Lucid FFI occasionally has trouble with C's standard output. Often a print statement in C code does not display any text, but in fact it is buffered by Lisp and never displayed. To see what has been collected this way, type **(if::flush-em)** in Lisp. This dumps what is left of the C stdout buffer.

17.3.4 Example 2: Image I/O, Shared Image Array Structures, and Image Modification

The second example demonstrates how Lisp images are managed from C in both coupling modes. Copy the file

$CMEHOME/lci/examples/load_and_blur.c

into your working directory and load the copy into an Emacs buffer. The header on the file should identify it as Example 2. The include file *rcde_types.h* contains definitions for all RCDE C types, including the shared image structure *array_image_struct*.

Notice the *extern* statement for *get_image_struct()* containing the *array_image_struct* pointer. This is the data structure used by the RCDE for array images, that is, images small enough to reside entirely in RAM. In this example the C-Callable function *get_image_struct()* is used to access a shared image structure.

The first function definition in the file, *get_image_pixel()*, is not a C-Callable function. It is a pure C function to set the pixel values of the image array. A copy of the image structure is passed by reference to this function and the pixel values at (x,y) are set to the value. In addition, adjoining pixels are re-set to the given value. Note that values from the *xmap* and *ymap* arrays must be right-shifted by two bits; this is necessary because of bit-level differences between the C and Lisp representations of integer data. It is not necessary, however, for the character data contained in the image array itself.

The primary function (in this file) is called *access_and_blur_image*. The first step is to provide a pathname to the C-Callable **load_image**. Note that even though no keywords are used in this call, since the function does have keywords, it is necessary to add a numeric zero at the end of the argument list. The return value is a c_handle pointing to the Lisp image. It is then displayed using the C-Callable from the source file *create_and_change.c*. It is necessary to load this file into Lisp by adding it to the C files list using LCI→Files→C Files. However, if you are continuing from the first example, it is already loaded.

The second step is an image manipulation call to **gauss_image** using the c_handle pointing to the loaded image. The result is then displayed on the chosen pane. Note that there are no optionals parameters used by this call, though the function accepts them. Thus the final argument is a numeric zero (just as a keyword list must end with a numeric zero).

The third step is to access the shared structure using the function *get_image_struct*, which takes only a c_handle and returns a copy of the complete image data structure. The next few lines of the example examine some of the components of the structure. Then a *for* loop is used to set the diagonal pixels of the image to 255. Note that the C function to set pixel values utilizes the xmap and ymap of the RCDE to quickly access the correct elements of the image array.

Next, the original image is displayed by pushing it onto the stack. In Tight Coupling, the white diagonal stripe will be visible in this image, since the original image data is modified. In Loose Coupling, however, this image will not have the stripe, since the pixel data in the local copy in the C process is modified, but not the original data in the Lisp process.

The two final steps copy the image data from the local image structure into a single-dimensional character array, then pass this array to Lisp to be converted into a Lisp image. In both coupling modes this creates a copy of the modified image with the diagonal stripe. In Loose Coupling, though, the data is passed between processes as a means of returning the modified image data to Lisp. Note that the nested *for* loops indicate how indexing must be done on the image array to produce a linear array in row-major order.

Initialize Interface and Execute Add the file *load_and_blur.c* to the list of C files. Add the file *create_and_change.c* also if it is not already there. Create a Lisp-callable form for the function *access_and_blur_image* using the same process as in the previous example. Compile the files in Performance mode.

To execute the compiled C function, select the Lisp-Callable C/C++ Functions button to see the list of functions in operation. Push the button by the function name; after the name appears on the line, select Execute to invoke the execution menu for that function. Then press the Execute button. The C function may also be executed from Lisp by typing

```
(access_and_blur_image)
```

into the Lisp Listener. Wait for and respond to the prompts for mouse selection of image panes in the RCDE documentation window. The instructions may be missed, in which case you may try to click a RCDE pane and look for a response.

Once the function executes in Tight Coupling, switch to Loose Coupling as you did with Example 1, then recompile and execute. Note that you must start the C process in a separate shell unless Parameters→Run Executable is set to "Yes".

17.3.5 Example 3: Object Creation and Manipulation in C++

Example Overview To continue with the scenario, copy the file

```
$CMEHOME/lci/examples/c_proxy_house.C
```

into your working directory and load the copy into an Emacs buffer. The header on the file should identify it as Example 3. The ALV Site must be loaded before proceeding further with this example. *The sample code contains calls to RCDE using both C and C++ syntax.* Now notice the two include files. The second, house_object_proxy.h, is necessary to include part of the C++ Proxy Class Hierarchy. It contains nested include statements defining other applicable classes in the hierarchy. This file, and any C++ files using the C++ Proxy Class Hierarchy, must be compiled with the include option "-I$CMEHOME/lci/proxy-objects".

The next area to notice is the *extern* statement areas. All function calls and structures that are not part of the include files must be externed in C++. These extern functions were automatically generated using a Lisp function called

```
(if::create-extern-statement 'lisp-name-without-package "v")
```

However, these extern statements are available in the file **$CMEHOME/lci/extern-statements**, which contains extern statements for all the RCDE C-Callable functions. This is a text file; it is not intended to be included in source code, but rather for textual cutting and pasting of selected statements.

The next area is the C call *pause_and_select_pane* modified to work in C++. The only differences are the argument list and the documentation string delimiters. In this example, this code is being used to pause between example steps.

The next area is the main routine, *create_and_change_house*. This function shows the steps in accessing the programming elements of the RCDE. Once these elements are present, any one of several experiments can be performed:

1. A pane, the top view and from it the ALV world are extracted from the ALV frame, after the ALV site data have been loaded.

2. The coordinate system and a feature set (**$3d-FS 1**) is extracted from the world.

3. A coordinate transform is constructed which transforms from a object centered coordinate system (for the house) to the world coordinate system, placing the origin of the house at (-35, 4567, 6050).

4. Next, a C++ house object is created associated with the world and the coordinate transform. The dimensions of the house and the roof pitch are supplied as keywords. The constructor is named the same as its class: HOUSE_OBJECT_PR. Note that the object has _pr appended to signify that it is a proxy object and constructor. At this point, the object will not appear on the display because it is not part of a feature set.

5. So the next step is to put the object into a feature set using the RCDE function call *add_object*. Note that the first argument is a Lisp symbol accessor for the **house1** object. It gets the Lisp pointer to the Lisp house object. The moment this function is executed, the object created appears on the screen. The pause and select option is placed before this call to allow you to look at this display at the time that is happens.

6. To rotate the object, a rotation matrix is created using a call to *make_orientation_matrix*. The only argument is the eleventh keyword, which allows us to specify the rotation angle about the z axis in degrees. In this case, a rotation of 90 degrees is specified.

7. Finally, the C++ house Class indirect method *house1.rotate_relative_to_world_by_pr* is called to rotate by the specified angle and axis. Note that the C function to accomplish this same activity is called *rotate_relative_to_world_by* and takes one more argument first. That argument is the Lisp symbol as in the call to *add_object*, which is a C call.

Initialize Interface and Execute Add the *c_proxy_house.C* file to the interface list of C++ files. Compile the file in Performance mode. Create a Lisp-callable form for the C++ function *create_and_change_house__Fv*. Note that Fv is added as the mangled name of the

interface function. To execute the compiled C++ function, select the Lisp-Callable C/C++ Functions button to see the list of functions in operation. Push the button by the function name; after the name appears on the line, select Execute and invoke another menu. Or run the C code by typing

```
(create_and_change_house_Fv)
```

into the Lisp Listener. Wait for and respond to the prompts for mouse selection of image panes in the RCDE documentation window. The first prompt is just to select a pane and nothing happens. The instructions may be missed, in which case you may try to click a RCDE pane and look for a response. At this point you can proceed with the Loose Coupling preparation as you did with Example 1. Recall that you must interrupt the Lisp process and the Debugger process, rerun the debug-main, and re-establish the interface.

17.4 The Menu Interface

This section summarizes the Lisp-C/C++ Interface menus for quick reference. Note: all of the string editor options on the menus require careful attention. First, the user must select the string editor line. Then the user may type and edit the string into the menu buffer. Finally, to enter the string, the user must use a carriage return.

Menu panels are invoked by selecting the main interface menu, LCI, as detailed in the following sections.

17.4.1 Control Panel

This menu contains buttons that perform loading and execution functions. Although the Establish Interface button is only applicable in Debugging Mode, all other buttons function in both execution modes.

Establish Interface – Establishes socket communications between Lisp and an external C or C++ process for Loosely-Coupled mode. If Parameters→Run Executable is set to "No", then this button should be pressed only after the external process is started manually (possibly within a debugger). If Parameters→Run Executable is set to "Yes", then the C/C++ process will be started by Lisp automatically, directing standard output to the Lisp buffer. In this case, the value of Parameters→Executable should give the name of the C/C++ executable in a pathname that is complete or relative to the current working directory given in Parameters→Working Directory. In either coupling mode, this button must be pressed before any C/C++ code is executed from Lisp.

Compile – Compiles a predefined set of user files using the compiler and compile options specified on the Parameters menu, then loads those files into Lisp according to execution mode. Additionally, a set of interface files are generated, compiled, and loaded

when Loosely-Coupled mode is in effect. Note that compilation depends heavily on the parameters specified in the Parameters menu.

Load C/C++ Files – Loads the set of user C and C++ files specified on the Files menu. The files must be compiled into object code before loading. If Parameters→Execution Mode is set to "Performance", then all C and C++ files and libraries are loaded. Otherwise, only the generated files *lisp-phantoms.o* and *lisp-stubs.o* are loaded from the current working directory, if they exist. This option allows execution of previously generated projects without regenerating or recompiling source code.

Load Lisp Files – Loads the set of user Lisp files specified on the Files menu into the Lisp environment. These files may include the files of Callable functions created by the user. The pathnames must be complete, or specified relative to the current working directory.

Create a c-handle – Creates a c-handle object pointing to a Lisp object to be used as a parameter or argument in a C call. The Lisp object is the last value returned by a function call, or the last object "dropped" into the Lisp buffer. The name of the c-handle object and the printed representation of the object pointed to are printed in the Lisp buffer. The name of the c-handle may then be entered as an argument to a Lisp-Callable taking a c-handle parameter, either on a function execution menu or at the Lisp command line.

17.4.2 Parameters Menu

This menu allows the user to control the interface modes and specify the parameters for compilation and execution. The Run Executable and Executable parameters affect the Establish Interface button on the Control Panel, the Execution Mode parameter affects the Load C/C++ Files control button, and all other parameters (except *Run Executable*) affect code compilation.

The parameters Compilation Level, Load User Object Code, Run Executable and Executable only apply in Debugging Mode.

.

Execution Mode – A button for specifying the coupling mode of the interface: "Debugging" for Loosely-Coupled mode, "Performance" for Tightly-Coupled mode.

Compilation Level – A button for specifying the level of compilation. The Regenerate Interface option causes the files containing phantom and stub functions to be automatically generated and compiled. The User Code Only option indicates that only user code should be compiled, omitting the code generation stage. Once the interface functions have been generated, they do not need to be regenerated again unless new Callable functions are added or existing Callable function parameter or return types

are changed. Although it is the user's responsibility to monitor these changes, setting the User Code Only option can significantly decrease compile time.

Load User Object Code – A button for specifying whether user object code is loaded immediately after compilation. In Tightly-Coupled mode, code is automatically loaded into Lisp after compilation. In Loosely-Coupled mode, loading can be disabled by setting this button to "No". This option may be useful when switching between modes for performance testing.

Run Makefile – A button for specifying whether the user has supplied a Makefile. If set, then the compile routine looks for the file specified by the Makefile parameter and executes the UNIX **make** command using the given makefile name.

In Performance Mode, the makefile must provide instructions for compiling each user source file; however, all linking and compilation of generated files is done by the LCI. In Debugging Mode, the makefile must produce a complete executable named by the Executable parameter. The makefile must compile all user source files into object or library files, then link user files with the generated files *c-stubs.o* and *c-phantoms.o*, and the LCI files *socket-lib.o*, *debug-main.o*, *rcde-c-phantoms.o* and *rcde_types_xdr.o* from the directory $CMEHOME/lci. See the example makefile $CMEHOME/lci/make-c-debug for further details.

If Run Makefile is not set, then each user C/C++ source file is compiled using the specified C or C++ compiler commands and options. The resulting user object files are linked with the generated object files and the LCI object files listed above to produce the executable named by the Executable parameter.

Makefile – A string editor for specifying a Makefile filename. If no makefile name is specified, **make** is executed with the default file *Makefile* when Run Makefile is set.

C Compile Command – A string editor for specifying the command to be used for C compilation (e.g., cc, gcc). Note that only **cc** is currently supported by the interface.

C Compile Options – A string editor for specifying the command-line options to be used for C compilation (e.g., -lm, -I$CMEHOME/lci). The options are copied directly onto the compilation command line. The -lm option is required for linking the LCI object files.

C++ Compile Command – A string editor for specifying the command to be used for C++ compilation (e.g., CC). Note that only AT&T C++ Version 2.1 is currently supported by the interface.

C++ Compile Options – A string editor for specifying the command-line options to be used for C compilation (e.g., -lm, -I$CMEHOME/lci). The options are copied directly onto the compilation command line. The -lm option is required for linking the LCI object files.

Working Directory – A string editor for specifying the default directory where the interface reads the user files and writes the generated interface files and compiled object files. All user source files are expected to be in this directory, specified by a pathname relative to this directory, or specified by a complete pathname.

Run Executable – A button for specifying if the executable (specified by the Executable parameter) is run automatically by the LCI when in Debugging Mode (this parameter has no effect in Performance Mode). If the option is "No", the user must run the C/C++ executable as a separate UNIX process, possibly under a debugger, before pressing Control Panel→Establish Interface. If the option is "Yes", then the LCI executes the executable and connects it with the Lisp process when Control Panel→Establish Interface is pressed. In this case, printed output from the C/C++ process will appear in the Lisp buffer. This mode is useful when executing code in Debugging Mode without using a debugger, because the user does not need to initiate the C/C++ process by hand.

Executable – A string editor for specifying the executable file name. This file is produced by compiling user and LCI files; it performs socket communications and implements the *main()* function. The default value for the file name is "debug-main".

17.4.3 Files Menu

Each entry in the Files Menu is a button that invokes a panel for specifying source file names. The source file pathnames must be defined relative to the current working directory (this also applies to loading operations) or as complete pathnames. For the C/C++ Libraries option, the complete or relative pathname of the archive (*.a) file must be given.

After the name is entered in the File Name option, the Add File button may be pressed to append the file name to the current list of files. A file may be deleted from the list by clicking on the line in the scrollable window containing the file name, then pressing the Delete File button. The file name should appear in the File Name slot when the file name is selected.

These menus are used to set the source files to be compiled, linked and loaded during LCI compilation. No immediate action is taken by these menu options.

For C and C++ files, the names of source files should be entered. If a Lisp file is specified and no extension is used, Lisp will first look for a *.sbin* extension with the name of the file; otherwise it will look for a *.lisp* extension. If the *.lisp* extension is used, it may not use the most recent version of the Lisp code, because it uses the **simple-load** command for loading Lisp files.

17.4.4 Functions Menu

.

This menu has three options: Lisp-Callable C/C++ Functions for Lisp-Callable creation, C-Callable Lisp Functions for C-Callable creation, and Lisp Variables for Variable Accessor creation. The first two set up the communications necessary for inter-language function calls. The menus display the forms currently defined, and also can be used to create new forms (which are automatically evaluated and printed to the Emacs buffer). The forms can be pasted into a file for future reloading.

Lisp-Callable C/C++ Functions – Manages Lisp-callable forms through a menu/entry panel sequence. The button creates the Lisp-Callable C/C++ Functions menu with the options to Create, Edit, Delete, Execute.

C-Callable Lisp Functions – Manages C-callable forms through a menu sequence. The button creates the C/C++-Callable Lisp Functions menu with the options to Create, Create Target Header, Edit, Delete.

Lisp Variables – Creates an entry panel to specify the name and type for a reader (get-...) and a writer (set-...) C-callable to manage the Lisp global variables used by the RCDE.

17.4.4.1 Lisp-Callable C/C++ Functions Menu

The first menu item is a list of Lisp-callables currently defined within the Lisp environment. Each Lisp-callable shown here must have a corresponding C/C++ function definition, or an error will result when compiling in Debugging mode. When a new Lisp-callable is created or loaded, this menu must be reinvoked to display the new entry.

The next item is a string edit line to enter the name of a new Lisp-callable. If a Lisp-callable is selected from the list, that Lisp-callable name will appear on string edit line. A Lisp-callable must be selected for the Edit, Delete and Execute operations.

Create – brings up an entry panel, Create New Function Header, to structure the Lisp-callable function creation panel which follows in this menu sequence. The first line is the Function Name string edit line. The name of the Lisp-callable function appears on this line and can be edited. The second line is a boolean switch to specify if the name on the function line is a mangled name. If "yes", the entry panel created will generate the argument list as specified in the mangled name. However, the Sun demangling utility *dem* must be on your Unix path (*before* the RCDE is started) to use this option. If "no", the entry panel created will generate the argument list from the Number of Parameters item that follows the mangled name entry. Selecting the Create Entry Panel button at the bottom of this menu will generate a customized entry panel for the Lisp-callable.

Function Header Entry Panel – opens a panel for entering information defining a Lisp-callable form. The first line of the panel is the Function Name (C/C++)

string edit line. The specified name must be in proper C/C++ syntax, and should exactly match the name visible to the C/C++ linker. If the function is a C++ function, then the mangled name must be used; it may be found by running the Sun utility *nm* or *nm++* on the compiled object file containing the function definition.

The second menu item is a pull-right menu specifying the Lisp-callable return type. A list of possible types appears for user selection when the item button is pressed.

The remaining entries are pairs of parameter name and parameter type. If a mangled function name is used, the number of parameters and their types are set according to the mangled name. Otherwise, default argument types are assigned. Although the names of Callable function parameters are not important, their types must be correctly identified. The parameter types may be set using the same pull-right menu as for the return type.

The Create Lisp-Callable button will create a Lisp-callable function, evaluate it, and print it in the Lisp buffer. The textual form may be pasted into another Emacs buffer, creating a Lisp source file of Callable functions. This file may be loaded rather than regenerating each Callable function for each RCDE session.

Edit – allows an existing Lisp-callable to be edited. The selected Lisp-callable is used to create a menu identical to the Function Header Entry Panel for changing the name, return type, parameter names and parameter types. The changes will not take effect, however, until the Create Lisp-Callable button is pressed.

Delete – removes the Lisp-callable function definition from the LCI, enabling recompilation in Debugging mode without the deleted function. This is useful for removing invalid functions when switching to a new set of C/C++ files (i.e. a new project).

Execute – provides a means to execute an existing C/C++ Function (through its Lisp-callable). Selection of this option brings up a menu containing one string edit line for each parameter of the function to be called. The user may type in parameter values, and may execute the function by pressing the Execute button. Parameters whose names begin with a colon are keyword parameters. If the function does not exist or is not identified correctly, the entry panel is not created.

17.4.4.2 C-Callable Lisp Functions Menu

The first menu item is a list of C-callables currently defined within the Lisp environment. Each C-callable shown here is a Lisp function that may be called from C/C++. When a new C-callable is created or loaded, this menu must be reinvoked to display the new entry.

The next item is a string edit line to enter the name of a C-callable. If a C-callable is selected from the list, that C-callable name will appear on string edit line. A C-callable must

be selected for the Edit and Delete operations, and a Lisp function name (not a C-callable name) must be entered for the Create Target Header option.

Create – invokes the Create New Function Header panel, which is described in the above section on the Lisp-Callable Functions Menu. The same sequence of operations is followed to create a C-callable form, except that the final step is to press the Create C-Callable button instead of Create Lisp-Callable.

However, a C-callable produced through the Create menu will not have a body, since there is no specification of what the function should do. C-callables are complete Lisp functions, and so cannot be easily defined through a menu interface. The resulting form is only intended to be enhanced with functionality by direct editing, rather than being executed immediately. This option should only be used by programmers familiar with Lisp and LCI programming.

Create Target Header – opens an entry panel for creating a C-callable function that calls an existing Lisp function (such as an RCDE function). The first line of the panel is the Function Name (C/C++) string edit line. The second menu item is a pull-right menu specifying the C-callable return type. A list of possible types appears for user selection when the return type button is pressed. The remaining entries are parameter types that may be specified by using the same pull-right menu as for the return type. The names and number of the parameters are extracted from the target Lisp function. The body of the C-callable consists of a call to the target Lisp function, and should not be edited.

Edit – allows an existing C-callable to be edited. The selected C-callable is used to create a menu identical to the Create Target Header menu for changing the name, return type, and parameter types of the C-callable. The changes will not take effect, however, until the Create Header button is pressed.

Delete – removes the C-callable function definition from the LCI, enabling recompilation in Debugging mode without the deleted function. This is useful for removing invalid functions when switching to a new set of C/C++ files (i.e. a new project).

17.4.4.3 Lisp Variables

The variables menu creates two accessor functions, a reader and a writer, that are callable from C. They both operate on the specified variable. These functions are named the same as the corresponding Lisp global variable with the asterisks removed, hyphens changed to underscores, and a "get-" and "set-" prefix. With these functions, the C/C++ user can access and change the Lisp globals. Lisp globals usually establish the default behavior of the environment. These functions are created when the Create C-Callable Accessors is selected.

17.4.5 Project Files Menu

The Project Files menu manages the loading, storing and deleting of project files. Projects are a convenient means of storing LCI parameters, such as all the items on the Parameters menu and all lists of source files and libraries. The Projects menu contains the following options:

Load Project – loads the project file named by the Project File: item at the top of the menu. This file must have been created earlier using the Save Project option. All LCI parameters are set to their values stored in the project file, and any existing Lisp-callable and C-callable forms are deleted. Source files are not loaded into the Lisp process, however.

Save Project – creates and saves the current state of the LCI in the project file named by the Project File: item at the top of the menu. Current values of all LCI parameters and lists of source files and libraries are saved.

Delete Project – removes any settings within the LCI that may overlap different projects. In particular, all Lisp-callable, C-callable and source file list entries are deleted.

17.5 C/C++ Programming Details

Most of the LCI concepts presented in this chapter apply to both C and C++; many of the differences have been demonstrated in the examples. The menu options are clearly marked where C and C++ are handled differently. Programming enhancements and conventions for C and C++ added to the Lucid FFI by the LCI are listed in this section. Additional details may be found in the LCI sections of the *Programmer's Reference Manual.*

17.5.1 Parameter Passing

Because of differences in function calling syntax and paradigms between Lisp and C/C++, it is necessary to establish special conventions to define equivalent parameter passing options. The LCI establishes the following requirements and capabilities:

- **Use of C-handles** — C-handles are defined as the type *c_handle* in C/C++ for declaring parameter and return types or C/C++ variables. Although a c_handle is simply an integer in C, a c_handle instance may uniquely point to a Lisp construct or object through the LCI. Some Lisp constructs have fixed addresses (static allocation) and some constructs are relocated by the Lisp garbage collection process. The c_handle system manages these differences without the need for user intervention.

- **Shared Structures** — Any C-Callable function with shared structures as parameters or return values must pass pointers to the structures, rather than instances of the structures. In Performance mode, these pointers refer to the same object in memory.

In Debugging mode, though, only an exact copy of the structure is passed between languages.

The user may add shared structures to the system, but this requires three steps:

1. The structure must have a Lisp foreign-structure definition created using the Lucid FFI macro **def-foreign-struct**, in addition to its C/C++ definition. This Lisp structure definition must match the C/C++ structure definition exactly.
2. The structure must have an associated XDR routine that formats the structure in binary form for transfer between the processes. This function may be automatically generated from the C *struct* definition using the Sun utility *rpcgen*.
3. The structure must be registered in the LCI. This is accomplished with a call to the LCI function **register-shared-type**. See the file **$CMEHOME/lci/shared-types.lisp** for examples.

During LCI Debugging mode compilation and loading, the user C/C++ source file(s) containing his XDR routines must be specified as a C source file, and *must be loaded into the Lisp process*. Unlike other source files, those containing XDR routines used to pass data between Lisp and C/C++ must be linked into both processes when compiling in Debugging mode. For a compiled object file named *my_xdr_routines.o* containing XDR routines, this may be explicitly accomplished by typing

```
(load-foreign-files "my_xdr_routines.o" nil)
```

at the Lisp prompt after normal Debugging mode compilation has completed. Once these steps are accomplished, then the structure may be passed between the languages as a single parameter or return value.

- **Optional Arguments** — C-Callable functions may have optional parameters. Each such C-Callable function has a parameter named **optionals_used**, which follows the list of required arguments. The optionals_used parameter contains a value giving the number of optional parameters to be used on a given function call. If no optional parameters are used, then optionals_used must be zero, i.e., the optionals_used parameter itself is required. C-Callable Lisp functions may not have both optional and keyword parameters.

- **Keyword Arguments** — C-Callable functions may have keyword parameters, which are arguments specified by an identifier-value pair in the argument list. Although standard Lisp allows keywords, C and C++ do not explicitly support any equivalent function-calling syntax. Any keyword in a keyword list is specified by two arguments. The first, the *keyword identifier*, is an integer specifying the ordinal position in the keyword list (e.g., the first keyword position is numeric one) given in the *Programmer's Reference Manual*, or in the Lisp definition of the C-Callable function. The second is a pointer to the actual value to be passed according to the type in the keyword list.

The utility of the keyword syntax is that it allows any order among the keyword arguments (as long as the identifier-pair order is maintained for each keyword pair), and any number of them (including zero) may be used on a given function call. To achieve this polymorphism, though, all keywords must be passed as pointers. However, string parameters may be passed by using char* types, not char** types. Unless all keyword parameters are used, the argument list must be terminated by an additional numeric zero argument, whether any keyword arguments are used or not. Within the body of the C-Callable function, keywords are automatically bundled into a property list. See the *Programmer's Reference Manual* entry on **def-foreign-callable-switch** for more details.

Because of the unusual syntax for keyword arguments, the C/C++ calling convention differs from what may be assumed given the function listings in the *Programmer's Reference Manual*. Each keyword pair must be listed as an integer parameter, followed by a *void** parameter. Thus each keyword given in the C-Callable documentation becomes two parameters in the C/C++function declaration. For examples, examine the C++ proxy class header files, which contain extern statements for each C-Callable used in the classes.

When programming the RCDE, a situation may arise where the C/C++ programmer receives a *c_handle* return value when he desires a vector of data. The LCI provides a set of data conversion utilities to convert between *c_handle* and value representations, as well as routines to create and access simple Lisp arrays and lists. These functions are described in the *Interface Utilities* section of the *Programmer's Reference Manual*.

17.5.2 LCI Compilation and Linking

The LCI supplies a few files for C/C++ compilation and linking, and a library of include files defining the C++ proxy hierarchy.

Most of the RCDE types available to C and C++ programs are defined in the file **$CMEHOME/lci/rcde_types.h**. This file contains definitions for most of the RCDE shared structures, and includes the files:

- **$CMEHOME/ic/c/image.h**, containing the shared structure definition for images, *array_image_struct*;
- **$CMEHOME/cme/c/transform.h**, containing the shared structure definition for transforms, *transform_4x4_struct*; and
- **$CMEHOME/cme/c/poly.h**, containing the shared structure definition for polyhedral vertices, *vertex_struct*.

When a file including rcde_types.h is compiled, the include option "-I$CMEHOME/lci" should be used.

The file **$CMEHOME/lci/define-statements** contains *#define* statements for keyword identifier arguments of the RCDE C-callable library; this file may be included in user source code, but is not included in **$CMEHOME/lci/rcde_types.h** to avoid name conflicts. Since many RCDE keyword parameters have the same name, but different keyword identifier values, the function name is prepended to the keyword name when this occurs. Using these defined constants instead of integer arguments should yield code that is easier to understand.

Any C-Callable function must be declared in an *extern* statement in the file using the C-Callable. Any C-Callable function that is not part of an include file must be externed in C++ (standard C++ practice). The file **$CMEHOME/lci/extern-statements** contains *extern* statements for all of the functions in the RCDE C-callable library. Since extern statements for C-callables taking keyword and optional arguments are not straightforward (as explained above), this file serves as a reference for all defined C-callables. It may be included in user C/C++ files, or used as a text source for individual *extern* statements in user source code.

The C++ proxy classes defining the C++ proxy class hierarchy are stored in three directories:

$CMEHOME/lci/proxy-images contains proxies for the image, pane, view and frame hierarchies

$CMEHOME/lci/proxy-objects contains proxies for all geometric and graphical objects, and

$CMEHOME/lci/proxy-transforms contains proxies for coordinate transforms and coordinate systems.

See Section 17.5.6 for details on the class hierarchy.

The user may also recompile the LCI source files coded in C. This may be desirable if the value of certain static system parameters needs to be changed. To recompile the C source files, execute the following line:

```
make -f make-debug-source-files
```

at a Unix shell prompt in the directory $CMEHOME/lci. The RCDE must be restarted for the compiled changes to take effect.

One system parameter that may need to be adjusted on-site is the size of the XDR buffers, *XDR_BUFFER_LENGTH*, declared in $CMEHOME/lci/debug.h. This parameter defines the upper limit in bytes of the total size of all the arguments that may be passed between Lisp and C/C++ in Debugging mode. Two buffers of this size are allocated in each process when they are started. By default, *XDR_BUFFER_LENGTH* is set to 2 MB. If large images are being passed between languages, or if memory allowances must be reduced, this value may be inadequate. To change this setting, edit the file $CMEHOME/lci/debug.h and remake the LCI source files as described just above.

17.5.3 Notational Conventions

The conversion of Lisp symbol names into C/C++ syntax requires that the result be in legal C/C++ syntax. The LCI defines the following naming conventions:

- **Leading Integers in Lisp function names** — Leading digits in Lisp names are converted to use the textual name of the integer in C/C++. For example, *2d-object* becomes *two_d_object*.

- **Structure Component Names** — Typical RCDE-provided structure types (shared structures) are given names which contain the types of the components followed by the number of such types (e.g., *c_handle_2_int_2_double_2*). The handle components would be called *h1, h2,* etc. Likewise, the integer components would be called *i1,i2* and the double components *d1, d2*.

- **C versus C++ Arguments** — Each C-Callable Lisp Function has a typed argument list, as listed in the *Programmer's Reference Manual*. The corresponding C++ member function (within a C++ proxy class) has the same argument list with the first entry removed. This entry is the pointer to the object itself, and is inherently supplied by the proxy class.

- **Proxy Names in C++** — Any C-Callable Lisp function name has a "_pr" appended when called as a member function in C++. C++ proxy class names are the same as their RCDE counterparts, except that the suffix "_pr" is appended, and any hyphens in the Lisp names are converted to underscores.

17.5.4 Calling C/C++ from Lisp

To call from Lisp to a C or C++ function, a Lisp-Callable function must be created using the menu sequence indicated above (i.e., LCI→ Functions→Lisp-Callable C/C++ Functions→ Create→Create Entry Panel→Create Lisp-Callable). The result will be the creation of a communication layer Lisp function called a DFFS (Def-Foreign-Function-Switch), or Lisp-callable function. In this sequence of menus, you must specify the name of the Lisp function, the name of the corresponding C or mangled C++ name and number and types of the arguments and the return type. The supported types include the following:

- *int, unsigned int, char, char*, void, void*, string, float, double, boolean* — All of these *basic types* are familiar C or C++ data types. Arrays of the non-pointer types are also supported.

- *c_handle (C/C++), c-handle (Lisp)* — A special LCI type that is an index or pointer to a Lisp construct. It is used to manipulate Lisp objects without moving the object through the interface.

- **lisp-handle** — A special LCI Lisp type that is effectively a pointer to a C construct. It is used to provide Lisp with a pointer to a C structure or C++ class.

If the handle classes are used properly, there is no need for the programmer to understand the underlying representation or implementation. The general principle in using handles is to allow the LCI to manage all encoding and decoding, while the programmer simply passes handles between languages as if they were objects. If a Lisp function returns a **c-handle**, then the C program only needs to know to expect a return type of *c_handle*. The returned value can then be used as a parameter to other C-callable Lisp functions that use a c-handle parameter, without the C program ever examining or otherwise using the c_handle directly.

A facility to pass C/C++ arrays between Lisp and C/C++ is also provided. C arrays are often represented as pointers, with no direct reference to the number of elements in the array. Similarly, Lisp can accept foreign pointers that point to the first of an unspecified number of array elements. For automatic transfer of these data types between processes, the RCDE defines a syntactic link between the pointer variable and a length variable giving the number of elements in the array. This syntax is defined such that a pointer will be treated as an array if the next parameter in the parameter list has the same name as the pointer, but with the suffix "_length" appended. For example, the foreign pointer *foreign-array* will be passed as an array if the next parameter is named *foreign-array_length*. This length parameter gives the number of array elements to pass on a given function call.

After the Lisp-callable has been created, the user may execute the function by using the LCI→Functions→Lisp-callable C/C++ Functions menu as described in the Menu Interface section above.

17.5.5 Calling Lisp from C/C++

To call from C or C++ to Lisp, a C-Callable function must be created using the menu sequence indicated above (i.e., LCI→Functions→C-Callable Lisp Functions→Create→Create Entry Panel→Create C-callable Header). The result will be the creation of a communication layer function called a DFCS (Def-Foreign-Callable-Switch), or C-callable. In this sequence of menus, you must specify the name of the Lisp function (only used in the Lisp symbol table), the name of the corresponding C or mangled C++ function, the return type, and the number and types of the arguments. C-callable forms have been created for most of the RCDE functions in the public interface; see the *Programmers Reference Manual*. In addition to the data types supported by Lisp-callable functions, C-callable functions support three more data types:

- **lisp-symbol** (Lisp), *lisp_symbol* (C/C++) — A special LCI type that is interpreted as a Lisp symbol. The lisp-symbol may be given as a parameter or return type in a C-callable. Equivalent to char* variables in C/C++, lisp-symbol parameters are decoded into a lisp symbol automatically via one level of Lisp evaluation.
- **lisp-form** (Lisp), *lisp_form* (C/C++) — A special LCI type that evaluates its value as a complete Lisp form. The lisp-form is only valid as a parameter type. It enables a

C/C++ programmer to pass a string argument (char*) containing a Lisp expression. It may be used as an argument to Lisp macros or functions which require such forms.

- **simple-vector-type** (Lisp), *char** (C/C++) — This return type allows Lisp simple one-dimensional arrays (vectors) to be returned to C directly. The corresponding Lisp array, however, must be *static*, or immune to garbage collection. Otherwise, the array might be moved, and the pointer would no longer be valid in tight coupling, and garbage collection might move the array during the copying phase in loose coupling. For the RCDE C-callables, any returned vectors are statically allocated.

  The length of the result array in bytes is returned in the value of the global C integer variable *sv_return_length*, while the type of the elements of the array is returned in the integer value of *sv_return_type*. The decoding of the value of *sv_return_type* is given in the file $CMEHOME/lci/sv_return.h. Both variables are automatically linked with user's C/C++ files, and should be declared as *extern* variables. Each vector transfer rewrites the value of these variables. The type of the returned data is not statically declared, since any function may return vectors with different element types. Therefore, C must cast the result appropriately for proper indexing, using *sv_return_type* to obtain the array type. **Simple-vector-type** is only valid as a return type in C-callables.

17.5.5.1 Conventions Used in Creating Target Headers for RCDE Functions

C-callable Lisp functions can be created in two different ways, with an existing Lisp function and without an existing Lisp function. When there is an existing Lisp function, it is referred to as the *Target* function. After the menu sequence Functions → C-Callable Lisp Functions, there is a choice to either Create a DFCS without a Lisp Function or Create Target Header DFCS with a Lisp Function.

Opting to Create a DFCS will produce a "empty" DFCS form. The user is expected to yank the form into a buffer and edit it to contain a desired sequence of Lisp expressions.

Opting to Create Target Header requires the specification of the name of an existing Lisp function along with its package prefix into the menu. The menu that follows reveals a C name for the function being called. It also shows a list of all of the arguments to the Lisp function, including keywords and optionals. It is the user's responsibility to specify the C types for each, including the return type for the function. The created form has a Lisp body consisting of a call to the target function and possibly some optional or keyword parameter decoding.

Special naming is useful for struct type specifiers. Typical RCDE-provided structure types are given names which contain the types of the components followed by the number of such types (e.g., *c_handle_2_int_2_double_2*).

While Lisp functions may return multiple values, C/C++ functions do not have the ability to collect these values after a call to Lisp. Instead, multiple return values are bundled into C structures defined for this purpose and returned as shared structures. The C/C++ function must then extract the individual values from the C structure.

Lisp functions have no restriction on the data types of their return values, allowing a Lisp function to return a value of any type on a given call. Since C/C++ is more strongly typed, RCDE functions returning multiple return types are proxied with multiple C-Callable functions, one for each type. The return type of a particular C-Callable function is included in its name, as a suffix. The C/C++ program must specifically select one of the C-Callable versions, based on the desired return type.

Similarly, Lisp functions have no restrictions on the types of their arguments (except in CLOS, which is used to some advantage where possible). RCDE functions that can take multiple parameter types for the same parameter are also proxied with multiple C functions, one for each possible type. Again, the parameter type of a particular C-Callable function is included in its name, as a suffix.

17.5.6 C++ Proxy Classes

The RCDE provides C++ proxy classes for many of the RCDE CLOS classes comprising the basic objects in the system. While C++ programmers may call the C-Callable library functions directly, as C functions, most of those functions are included as C++ member functions in the C++ proxy classes. The RCDE provides a full hierarchy of proxies necessary to utilize the RCDE Classes as a C++ Class Library. The C++ inheritance and function dispatching mirror that of the class hierarchies in the RCDE. Since the C++ Proxy Classes are full-fledged C++ classes, member functions with native C++ code may be added to the classes. Also, new classes may be attached to the inheritance hierarchy.

Each proxy class consists of standard C++ components, but these generally refer to Lisp for their true functionality.

- Data Slots — Each proxy class has at least one data slot, of type c_handle, that points to a corresponding instance of the Lisp class being proxied. Additional data slots may be added, as in the case of *blocked_array_mapped_image*, for local copies of data to be accessed directly from C++. In most cases, however, all data slots are maintained in Lisp, while data accessors are provided as member functions in C++. This is consistent with data hiding techniques, with the addition that the data is kept in another language.

- Member Functions — Proxy classes contain member functions that correspond to direct methods on the equivalent RCDE classes. If an RCDE generic function in the public interface has a method on a particular RCDE class, then the proxy class for that RCDE class will have a corresponding member function which calls the method.

  The member functions have a consistent functionality, since they simply call through the LCI to execute the corresponding Lisp class method. Each has an argument list that is the same as the C-Callable's argument list, except that the first argument, which is the class object, is removed. The body of the function is a call to the C-Callable function, with the proxy's local c_handle slot inserted as the first argument.

There are a few exceptions to this format, since Lisp does not require the first argument to be the class object. The syntax given in the header files should be noted before member functions are used.

Where possible, member function overloading has been used to accommodate RCDE generic functions that accept multiple parameter types. In these cases, C++ dispatching is used to select the proper C-Callable function, rather than forcing the programmer to explicitly determine which C-Callable must be used (e.g., the *iref* function). The equivalent technique cannot be applied to return types, however, since C++ does not allow overloading based solely on return types.

- Constructors — Each proxy class has at least two constructors, a default constructor which is empty and a constructor that accepts a c_handle argument. The latter may be used when a C++ class is instantiated to proxy a Lisp object already in existence. The local c_handle slot of the proxy is set to the given c_handle, so that calls to member functions will operate on the given Lisp object.

  Many C++ proxy classes have a third constructor that calls the equivalent Lisp constructor for that class. In most cases, the argument list given to the constructor is exactly the same as the argument list of the corresponding C-Callable for the constructor. Occasionally, however, the Lisp constructor has no required parameters, but only keyword arguments. In this situation, the Lisp-calling constructor cannot be distinguished from the default constructor, since keyword parameters are implemented as optional parameters in C++. To resolve this ambiguity, each Lisp-calling constructor with only keyword parameters has one added required argument, of type *LCI_CONSTRUCTOR_TYPE*, to distinguish it from the default C++ constructor. The Lisp-calling constructor may be called by giving the value *LCI_CONSTRUCTOR* as the first argument.

  The two variables *LCI_CONSTRUCTOR_TYPE* and *LCI_CONSTRUCTOR* are defined in *rcde_types.h*.

17.5.7 C/C++ Programming Hints and Suggestions

While considerable effort has been made to facilitate C/C++ code development using the LCI, some aspects of compiling, linking and executing C/C++ code through Lisp can be problematic. C++ code, in particular, may cause difficulty when executed within the Lisp process because of its memory management techniques. Debugging mode can introduce extra complications, because parameter typing errors in user code may lead to apparent errors in the automatically generated C code, which is unfamiliar to the programmer.

Some common errors and idiosyncracies of executing C/C++ though Lisp can be anticipated. Here is a list of problems or potential problems the C/C++ programmer might encounter:

Undefined Foreign Symbols: When the RCDE is started, the LCI is loaded last. A warning may be printed, stating that there are linking symbols that have not been resolved:

```
;;; Warning: The following foreign symbols are undefined: (_dlsym
_lisp_function_table _dlclose _dlopen)
```

The three *dl_** symbols are derived from the loading of C libraries, and can be ignored. The *lisp_function_table* symbol is created by the LCI, but it should be resolved when the user compiles anything in Debugging mode. Initially, though, it is supposed to be undefined.

The undefined foreign symbols list is the primary feedback from the Lisp linker. Since Lisp attempts to resolve all linking symbols in C/C++ code as it is loaded, undefined foreign symbols often mean that modules are missing or function names are incorrect. In particular, if the C/C++ name given to a Lisp-callable or C-callable is not accurate, i.e. it does not correspond to the function name in the C/C++ code, then compilation will fail, and the misnamed symbol will be added to the undefined foreign symbols list.

In some situations, symbols may be added to the undefined foreign symbols list during a successful compilation. When a Lisp-callable form is evaluated in Lisp, two foreign symbols are created – one for Performance mode execution, and one for Debugging mode execution. If the user executes in Debugging mode without loading his C/C++ code into the Lisp process (which is not necessary and probably not desirable), then the symbols created for Performance mode execution are not resolved. In this case, the C/C++ name given in the Lisp-callable will appear on the undefined foreign symbols list. If the user never compiles in Debugging mode, then each Lisp-callable will produce one unresolved symbol – the C/C++ name given in the Lisp-callable with the suffix "_stub" appended. These symbols are expected to be unresolved, and will not cause errors.

c_function_table **Compiler Error:** When compiling in Debugging mode, the LCI expects to find at least one defined Lisp-callable. If one does not exist, a linking error will appear, reporting that *c_function_table* is an unresolved symbol. This is easily fixed by creating (or loading) a Lisp-callable function for the C/C++ function called to initiate C/C++ execution.

C++ Libraries: The RCDE loads the standard C libraries *libc.a* and *libm.a* when it is initialized. The standard C++ libraries are not loaded, however. To execute C++ code in Performance mode, any referenced C++ libraries must be loaded directly into the Lisp process, including *libC.a* and *libOstream.a*. This can be accomplished by adding the complete pathnames of the archive files to the C/C++ libraries list, and loading C/C++ files while in Performance mode.

C++ Initializations: When C++ is compiled, it is generally translated into C, either symbolically or textually. During this process, some default initializations are added to the *main()* function in the user's code. However, since the *main()* function cannot be in user code compiled under the LCI, these initializations are never encoded by the C++ compiler, and hence are never executed. This can lead to seemingly inexplicable bugs and crashes when executing C++ programs in both execution modes.

One essential initialization is for the default C++ output stream *cout*. Without an initialization statement, a program may hang or simply crash at the first attempt to direct output to *cout*. To avoid this, the user should place the C++ statement

```
Iostream_init init;
```

in a function that is called before *cout* is used. The variable *init* is not itself important, but serves to open the necessary streams.

Other initialization errors may be encountered, depending on the behavior of the C++ compiler being used. The user may identify them by running the C++ compiler and directing its output to a C file, rather than an object file. The *main()* function in this C file may be inspected for initialization statements placed there by the compiler, not the user.

Chapter 18

On-line Documentation

18.1 RCDE's Online Documentation System

Indices to all of the RCDE documentation of interest to users of the system are available on-line and accessible by using the Info facility of GNU Emacs. These Info indices are organized in a hierarchy that reflects the available documents. The top node of the Info index is a menu selection when the Info facility is invoked. Each Info index contains the same information as the corresponding index in the printed version. This Info facility allows one to jump directly to any page number in an **xdvi** text previewer containing the corresponding document by doing anyone of several string searches in the on-line Info index, and then invoking the corresponding page search in the **xdvi** viewer. This facility when coupled with the **xdvi** viewer is extremely useful in quickly accessing the available information about a given function, concept, class, or variable.

18.2 The GNU Emacs Info facility

The on-line index system makes use of the Info facility of GNU Emacs. To enter the on-line index, use the key sequence C-h i, (*control-h i*). The *control-h* is the standard Emacs key for accessing any help facility, and the "i" gets one the Info facility within Emacs. To exit the on-line , type "q". This will restore your original Emacs Buffer. Alternatively, enter meta-x help-for-help, then enter "i" to get the Info facility.

If you have not previously accessed the Info facility in your current editor invocation, perform this command to create an Emacs buffer that looks like Figure 18.1. Note: Some of the table has been left out to fit on the figure.

If what you see now does not include the last line of Figure 18.1, type "g(dir)" to get the appropriate menu. If this does not work, see your systems administrator to review the installation of the GNU Emacs Info system and the RCDE installation. What you

```
File: dir Node: Top This is the top of the INFO tree
  This (the Directory node) gives a menu of major topics.
  Typing "d" returns here, "q" exits, "?" lists all INFO commands,
  "h" gives a primer for first-timers,
  "mTexinfo<Return>" visits Texinfo topic, etc.

  Clicking middle on a highlighted word follows that crossreference.
  Clicking right anywhere brings up a menu of commands and references to
  follow.
  --- PLEASE ADD DOCUMENTATION TO THIS TREE. (See INFO topic first.) ---

* Menu: The list of major topics begins on the next line.

* Info:     (info). Documentation browsing system.

* Emacs:    (emacs). The extensible self-documenting text editor.
 This manual is for Lucid GNU Emacs 19.3.

                                      ⋮

* CVS:      (pcl-cvs).  The front end to CVS.

* RCDE:     (rcde.info). RCDE Documentation.
```

Figure 18.1: The GNU Emacs Top-Level Directory

```
File: rcde.info Node: Top

*Menu:

* prm:  (idd.info).  The Programmer's Reference Manual.
* ig:   (ig.info).   The Installation Guide.
* ugd:  (ugd.info).  The User's Manual.
```

Figure 18.2: The Menu of RCDE Documents

are looking at is the top-level directory into the GNU Emacs Info facility. At the top is a section giving a brief synopsis of commands which are available in Info mode. Note that this section says that by typing "h," one can take the Info mode tutorial. The tutorial introduces most of the commands which are available to navigate through on-line documentation, and gives one enough practice in using them to enable one to use the facility with some level of confidence.

The rest of the Info top-level directory is a menu of Info topics. Most of these topics correspond to documents for software products put out by the Free Software Foundation (FSF), which developed GNU Emacs. One of the manuals on the list is for the Info system. This is convenient should you forget one of the commands you learned in taking the tutorial, or you want more detail on one of the commands.

The RCDE Info File If your system has had the RCDE installed according to the instructions in the *RCDE Installation Guide*, one of the menu items should read "* RCDE: (rcde.info)." To get to the menu of The RCDE documentation, type "m RCDE." The "m" is the command for accessing a menu item, and the "RCDE" accesses the menu item giving the directory of RCDE documentation. The same thing could have been accomplished by pointing to the RCDE menu item with the mouse and clicking, then typing "m < *return* >."

When you have selected the RCDE menu, you should see the menu of Figure 18.2.

To get to **this** text in the *RCDE User's Manual*, point to the "User's Manual" entry on the RCDE menu with the mouse, click left, then type "m< *return* >." This gets you to a menu for the index and table of contents of the *RCDE User's Manual.* Select the table of contents menu item and type "m < *return* >." String search for GNU Emacs. The index will then reveal the page number. Search for that page in the **xdvi** previewer containing this manual.

Note — The Version 1.0 release does not have the complete text of the documents available on-line. Tables of Contents and Indices are available for the Version 1.0 release. We suggest that you use this facility to assist you in finding where to look for information, and use a viewer such as **xdvi**, which is distributed with LaTeX. To view a document using **xdvi**, type

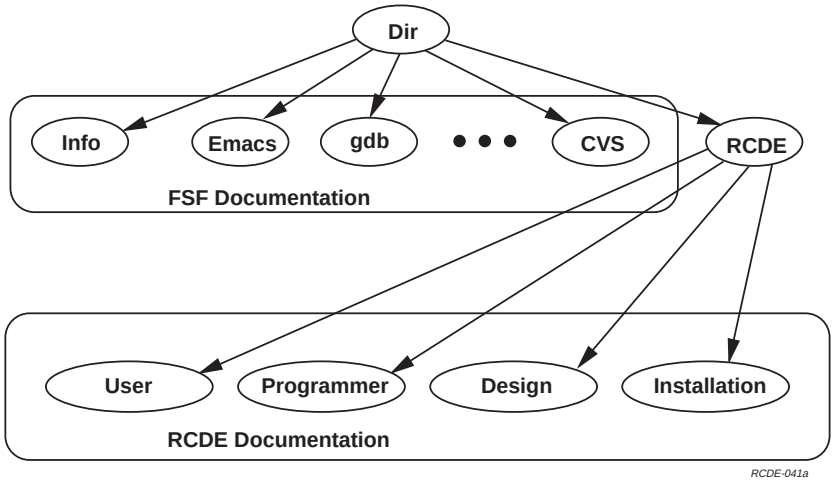


Figure 18.3: Top Level Nodes of RCDE Documentation

xdvi < *filename* >, substituting the filename of the document for < *filename* >. The spacebar and del keys can be used to page through the document. Once you are viewing a numbered page, you can set the pagenumber by typing the number, then "P", the **set page number** command. Once this is done, you can jump to any desired page by typing in the page number, then "g", the **goto page** command. This could be a page number obtained from one of our Info indices or tables of contents, or it could be one of your choosing. The Emacs **list-matching-lines** command, executed by typing **M-x list-matching-lines**, is useful for getting a subset of an index or table of contents containing a search target.

18.3 Organization of RCDE Nodes

Figure 18.3 shows the GNU Info document hierarchy graphically, and within the figure the first level of the RCDE document hierarchy is illustrated. Each menu and index is referred to as a node in the hierarchy.

The directory node supplied by the FSF has been modified during the RCDE installation procedure to add a menu item pointing to the RCDE documentation. This menu item selects the next lower level node, a node which gives a menu of the RCDE indices in an Info menu. Selection of one of these menu items will provide access to a particular RCDE index, table of contents or menu.

Within an Info hierarchy, the node at the top will be a menu or "Table of Contents", which is a menu allowing one to directly access any node of the Info hierarchy, since it con-

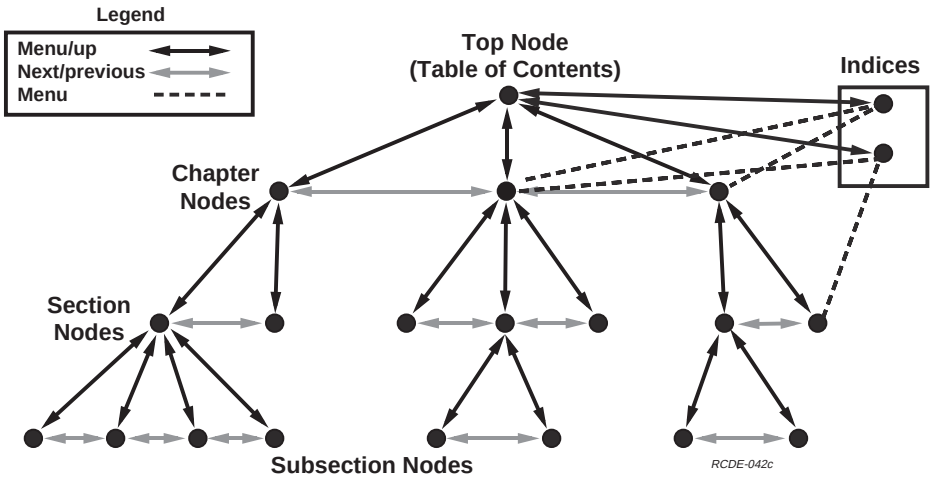


Figure 18.4: Documentation Node Relationships

tains all the nodes. Figure 18.4 shows these relationships graphically. The figure illustrates another set of key nodes, indices, corresponding to menus allowing direct access to the node containing the selected index entry. Other document nodes are arranged into a hierarchy which reflects the sectioning of the Info hierarchy. A menu is provided to access the subordinates of any node that has subordinates, and "previous," "next," and "up" nodes are provided to access the previous and next nodes at the same level, and the parent node, respectively. If the document has "Parts," the *part* nodes will be at the level above the chapter level in the hierarchy. Experiment using the menus to get to deeper nodes of the Info hierarchy, "u," the "Up" command to return to higher nodes, and "p" and "n" to get to the previous and next node, respectively, at the same level.

18.4 Synopsis of Info Commands

The RCDE key Info commands are:

m: menu Select the menu item indicated. The indication can be chosen by typing in the menu item name (or enough to disambiguate it from others), or by taking as default the menu line in which the Emacs cursor resides. Partial item names are a handy way to find other menu items in a long list, because a <space> bar will cause a list of completions to appear in a pop-up buffer. It will disappear when a specific choice is typed in.

p: previous Select the "previous" node to a given node. In the RCDE, the previous node is a node at the same hierarchical level as the current node, having the same parent.

n: next Select the "next" node to a given node. In the RCDE, the next node is a node at the same hierarchical level as the current node, having the same parent.

u: up Select the "up" node to a given node. In the RCDE, the up node is always the parent of the node in the sectioning hierarchy.

f: follow Follow a cross reference. If a cross reference is present in the document, it will show up as a "See such and such" item within the document. It can be selected in the same way as a menu item.

l: last Return to the node being visited just prior to the current node. A reliable way to retrace your hypertext travel path.

< *space* >: **page down** Scroll down slightly less than a screenful of text. Note: < *space* > is context sensitive. As an Emacs command in a mini buffer it usually means try to complete this partial string.

< *delete* >: **page up** Scroll up slightly less than a screenful of text.

h: help Bring up the Info tutorial.

?: What? Give list of Info commands.

g: goto Go directly to the named node.

b: beginning Go to the beginning of the current node.

q: quit Exit from Info, restoring previously viewed buffer in window currently used by Info. Next entry into Info will be to the same current node.

18.5 Other Means of Obtaining Online Information

The Common Lisp environment has its own means of providing information about the system and user code. One such means is through the association of document strings with the key components of the system. These document strings can be viewed by calling the Lisp function **documentation**. For instance, to get the documentation string for the function **image-difference**, type

```
(documentation 'ic:image-difference 'function)
```

at the Lisp prompt. Documentation types that are available are

1. Function,

2. Macro,
3. Variable, and
4. Class.

If you are using GNU Emacs with ILisp to run your RCDE process, the same thing can be done by typing

```
(ic:gauss-convolve-image
```

followed by <control-d> either at the Lisp prompt or in a Lisp mode editing buffer. Similarly, argument lists can be obtained using the **arglist** Lisp function or the Emacs command **lisp-arglist**, which is bound to <control-c a> in the Lisp and Ilisp modes. Much of the *Programmers' Reference Manual* is automatically generated by functions which use calls to **documentation** and **arglist** to ensure that the printed manuals are consistent with the documentation strings and true argument lists.

The Lisp function **apropos** is another means of gathering information about the system. If you know part of the name of a Lisp symbol, **apropos** will give a list of matching names. For instance, the call

```
(apropos 'house 'cme)
```

gives a list of all symbols in package **cme** containing the string "house" in their names. A similar function is available to ILisp users through the **lisp-complete** command <meta-TAB>, which will complete typing a partially-typed symbol name.

Appendix A

Bucky Menu Functionality

Each RCDE object is manipulated by a number of commands that are accessed from the Bucky keys or Bucky menus. Because the requirements of objects vary, each kind of object will have a different set of available commands. This is a result of the object-oriented nature of the RCDE; each object inherits characteristics from its ancestors.

The following tables summarize the Bucky menu commands available on each of the RCDE objects. Refer to the *Programmer's Reference Manual* for details on programming with these functions. Note that each table ends with a double horizontal line, which helps the reader distinguish where multiple-page tables end.

Table A.1: Bucky Menu Functionality for Class 3d-Curve

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Close Verts	Close Vertex Modification	H M	M
Reset Verts	Restore Vertices to Default Positions for Object	H M	L
Scale	Change x,y,z sizes.	C	R

Add Vert	Add a new Vertex	M	L
Del Vert	Delete Vertex	M	M
Reset	Delete All Vertices Except the First	M	R
Vert UVXY	—		L
Vert Z	—		M
Vert UV On Dtm	—	C	L
Vert W	—	C	M
Object UVXY	—	C	R
Every Z to Ground	—	SMC	M

Table A.2: Bucky Menu Functionality for Class 3d-Closed-Curve

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Close Verts	Close Vertex Modification	H M	M
Reset Verts	Restore Vertices to Default Positions for Object	H M	L
Scale	Change x,y,z sizes.	C	R

Add Vert	Add a new Vertex	M	L
Del Vert	Delete Vertex	M	M
Reset	Delete All Vertices Except the First	M	R
Vert UVXY	—		L
Vert Z	—		M
Vert UV On Dtm	—	C	L
Vert W	—	C	M
Object UVXY	—	C	R
Every Z to Ground	—	SMC	M
Every Z to Ground	—	SMC	M
Extrude Object	—	SMC	L

Table A.3: Bucky Menu Functionality for Class 3d-Ruler-Object

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Close Verts	Close Vertex Modification	H M	M
Reset Verts	Restore Vertices to Default Positions for Object	H M	L
Scale	Change x,y,z sizes.	C	R

Add Vert	Add a new Vertex	M	L
Del Vert	Delete Vertex	M	M
Reset	Delete All Vertices Except the First	M	R
Vert UVXY	—		L
Vert Z	—		M
Vert UV On Dtm	—	C	L
Vert W	—	C	M
Object UVXY	—	C	R
Every Z to Ground	—	SMC	M
Set Sun Ray	—	M	M

Table A.4: Bucky Menu Functionality for Class Image-Windowing-Tool

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV	—		L
	—		M
	—	C	L
	—	C	M
	—	MC	L
	—	MC	M
	—	MC	R
	—	SM	L
Rotate	Rotate	SM	M
	—	SM	R
	—	S	L
	—	S	M
	—	S	R
Move	—		L
Change Size	—	C	M
Move Edge	—		M
Make Window	—		R
Scroll	—	C	M
TScroll	Scroll Tandem Views	C	L

Table A.5: Bucky Menu Functionality for Class House-Object

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Open Verts	Allow Vertex Modification	H M	L
Close Object	Open Superior to Object	H M	M
Scale	Change x,y,z sizes.	C	R
XY sizes	—	M	L

Rotate/Scale	—	M	M
Z size	—	M	R
Taper rate	—	S C	R
Roof Pitch	—	S C	R

Table A.6: Bucky Menu Functionality for Class Extruded-Object

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Close Verts	Close Vertex Modification	H M	M
Reset Verts	Restore Vertices to Default Positions for Object	H M	L
Scale	Change x,y,z sizes.	C	R

XY sizes	—	M	L
Rotate/Scale	—	M	M
Z size	—	M	R
Taper rate	—	S C	R
Vert UVXY	—		L
Vert Z	—		M
Vert UV On Dtm	—	C	L
Vert W	—	C	M
Object UVXY	—	C	R
Add Vert	Add a new Vertex	M	L
Del Vert	Delete Vertex	M	M

Table A.7: Bucky Menu Functionality for Class Cylinder

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Open Verts	Allow Vertex Modification	H M	L
Close Object	Open Superior to Object	H M	M
Scale	Change x,y,z sizes.	C	R
XY sizes	—	M	L

Rotate/Scale	—	M	M
Z size	—	M	R
Taper rate	—	S C	R
XY Scale	—	M	L
	—	M	M

Table A.8: Bucky Menu Functionality for Class Half-Cylinder

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Open Verts	Allow Vertex Modification	H M	L
Close Object	Open Superior to Object	H M	M
Scale	Change x,y,z sizes.	C	R
XY sizes	—	M	L

Rotate/Scale	—	M	M
Z size	—	M	R
Taper rate	—	S C	R
XY Scale	—	M	L
	—	M	M
Radius/Length	—	M	L
Rotate/Scale	—	M	M

Table A.9: Bucky Menu Functionality for Class View-Hacking-Object

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move XY	—		L
Offset	—	C	L
Gain	—	C	M
	—		L
Brt&Cont	—		M
Reset G&O	—	C	R
Neg Contrast	—	M	R
Auto Stretch	—	M	M
Threshold	—	M	L
Move Me	—	SM	L

Table A.10: Bucky Menu Functionality for Class Superellipse

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Open Verts	Allow Vertex Modification	H M	L
Close Object	Open Superior to Object	H M	M
Scale	Change x,y,z sizes.	C	R
XY sizes	—	M	L

Rotate/Scale	—	M	M
Z size	—	M	R
Taper rate	—	S C	R
XY exponent	—	SMC	L
Z exponent	—	SMC	M
Exponents	—	SMC	R

Table A.11: Bucky Menu Functionality for Class Color-Map-Hacking-Object

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move XY	—		L
Red	—	SM	L
Green	—	SM	M
Blue	—	SM	R
Intensity	—	S	L
Hue	—	S	M
Saturation	—	S	R
Offset	—	C	L
Gain	—	C	M
Reset	—	C	R
Gamma	—		L
Brt&Cont	—		M
-Overlays	—	M	L
Move Me	—	SM	L

Table A.12: Bucky Menu Functionality for Class Superquadric

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Open Verts	Allow Vertex Modification	H M	L
Close Object	Open Superior to Object	H M	M
Scale	Change x,y,z sizes.	C	R
XY sizes	—	M	L

Rotate/Scale	—	M	M
Z size	—	M	R
Taper rate	—	S C	R
X exponent	—	SMC	L
Y exponent	—	SMC	M
Z exponent	—	SMC	R

Table A.13: Bucky Menu Functionality for Class Camera-Model-Object

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Move UV @Z	—		L
Move Z	—		M
Move Cam W	—	C	R

Princ Pt	—	M	L
Focal Length	—	M	M
Drop Z	Drop Object Vertically to Surface.	MC	L
	—	C	M
	—	SM	L
	—	H	L
	—	C	L
	—	MC	M
	—	MC	R
UV Roll	—	S	L
Force Z Up	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	—	S	R
Show Image	—	HS	M

Table A.14: Bucky Menu Functionality for Class Sun-Ray-Object

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Close Verts	Close Vertex Modification	H M	M
Reset Verts	Restore Vertices to Default Positions for Object	H M	L
Scale	Change x,y,z sizes.	C	R

Add Vert	Add a new Vertex	M	L
Del Vert	Delete Vertex	M	M
Reset	Delete All Vertices Except the First	M	R
Vert UVXY	—		L
Vert Z	—		M
Vert UV On Dtm	—	C	L
Vert W	—	C	M
Object UVXY	—	C	R
Every Z to Ground	—	SMC	M
Set Sun Ray	—	M	L
Scale	—	M	M

Table A.15: Bucky Menu Functionality for Class Conjugate-Point-Object

Bucky Label	Documentation Line	Keys	Mouse
Menu	Menu of Object Parameters		R
Blank	Temporarily Remove Object from Views.	HS	L
Close Object	Close Object to Mouse Sensitivity, Open Superior.	H M	M
Edit Hier	Edit Composite Object Hierarchy	H M	R
Undo	—	H C	L
Redo	—	H C	M
Bucky Menu	—	H C	R
Clone	Create an Identical Copy of Object.	H	L
Delete	Delete this Object.	H	M
Drop/Obj	Stop Modifications to this Object, return Object Instance to Listener.	H	R
Move UV @Z	Move Object by Projecting UV Mouse Motion to World XY @ Constant Z.		L
Move Z	Move Object along World Z.		M
Move UV on DTM	—	C	L
Move W	Move Object Along Ray from View.	C	M
Drop Z	Drop Object Vertically to Surface.	MC	L
Sun Z	Adjust Object Height using Sun Model and Shadow on DEM	MC	M
Drop W	Place Object at Intersection of Ray from View with DEM	MC	R
Az-Elev	Mouse X Rotates about World Z, Mouse Y Rotates about Object X	SM	L
Z Rot	Rotate about World Z	SM	M
Z' Rot	Rotate about Object Z	SM	R
UV Roll	Rotate Object as rolling ball	S	L
Re Orient	Reset to Canonical Orientation (Z axis up).	S	M
W Rot	Rotate Object about the Ray from View	S	R
@Vertex	—	SMC	R
Move Conj UV	—	C	R
@Vertex	—	M	M
@Image UV	—	M	R

Resection Improve	—	M	L
Resection Menu	—	SMC	L
Delete UV	—	SMC	M

Appendix B

Glossary

ARPA *Advanced Research Projects Agency.* One of the sponsors of the RADIUS program and other basic IU research that will be the primary source of technology for RADIUS.

Bucky Keys A set of four keys (Hyper, Super, Meta, and Control) on the RCDE keyboard that serve to modify system key codes.

Bucky Menus A feature of the RCDE that employs the Bucky keys to alter the command menu structure, allowing access to a wide range of functions with few keystrokes or mouse actions.

CASE *Computer-Aided Software Engineering*

Class A Class is a description of a group of *Objects* with similar properties, common behavior, common relationships, and common semantics.

CME *Cartographic Modeling Environment.* A software package developed at SRI for producing three-dimensional models of areas of interest using image data. CME is the basis of the RCDE system.

CLOS *Common Lisp Object System.* An extension to the Common Lisp computer language specification to support programming in an object-oriented style.

DEM *Digital Elevation Model.* A format for digital elevation data supported by the USGS.

DMA *Defense Mapping Agency.*

DTED *Digital Terrain Elevation Data.* A specific DTM product line supported by the DMA.

DTM *Digital Terrain Map.*

FASD *FASt Dump.* Mechanism to produce a Lisp form from an object that produces a copy of the object when evaluated.

FFI *Foreign Function Interface.* Basic functionality supplied by Lucid Lisp allowing single-process interfacing between Lisp and C.

FFT *Fast Fourier Transform.*

FSF *Free Software Foundation.*

IPC *InterProcess Communication.*

IU *Image Understanding.* The science and engineering endeavor of attempting to imbue machines with sight-related cognitive abilities. Also an ARPA research program that has image understanding in this sense as one of its goals.

LCI *Lisp-C/C++ Interface.* The language interface module built into RCDE allowing C and C++ programs to access RCDE functionality and data.

LVCS *Local Vertical Coordinate System.* In standard photogrammetric usage, refers to the local interpretation of altitude with respect to a reference (height=0) datum. In the RCDE, LVCS refers to a local 3-D coordinate frame attached to the site being modeled.

Mixin A class defined for the purpose of contributing properties via inheritance, which is not instantiated directly. For example, a class named SHADED-HOUSE might inherit from SHADED-FACE-MIXIN, which gives the house the ability to shade its faces.

Module A generic term for a software component (e.g., *Class*, *Service*, or *Subsystem*).

Object A concept, abstraction, or thing with crisp boundaries and meanings for the problem at hand; an instance of a *Class*.

OMT *Object Modeling Tool.* A CASE tool developed at GE to support the discipline of object-oriented analysis and design, particularly of software.

ORD *Office of Research and Development.* The primary sponsor of the RADIUS and RCDE programs.

RADIUS *Research and Development for Image Understanding Systems.* A program administered by ORD and ARPA for investigating Model-Supported Exploitation (MSE).

RAM *Random Access Memory.*

RCDE *RADIUS Common Development Environment.* An interactive workstation environment and standard toolset for supporting Image Understanding research. It is the development environment upon which the RADIUS Phase 2 testbed will be built.

Resection The process of simultaneously adjusting camera parameters to resolve discrepancies between the apparent image locations and known world locations of a set of conjugate points.

Service A group of related functions (or operations) that work together to provide a functional capability.

Subsystem A major component of a system organized around some coherent theme.

System An organized collection of components that interact. Specifically, the RCDE itself.

USGS *Unites States Geologic Survey.*

UTM *Universal Transverse Mercator.* A standard set of map projections onto cylinders that are tangent to longitude circles (meridians).

XDR *eXternal Data Representation.* A machine-independent, byte-level representation defined by Sun Microsystems for basic data types. Intended to be duplicated by other vendors to facilitate data passing between differing architectures.

Index

M

T