

The RADIUS Common Development Environment*

Aaron J. Heller and Lynn H. Quam

Artificial Intelligence Center, SRI International
333 Ravenswood Ave., Menlo Park, CA 94025
E-MAIL: {heller,quam}@ai.sri.com

Abstract

The RADIUS Common Development Environment (RCDE) pulls together many diverse functions into an integrated whole. The main goal of the environment is to provide a system to carry out interactive cartographic modeling of three-dimensional scenes from multiple images, as well as to provide an infrastructure to support the research in, and implementation of, image-understanding-based algorithms for this and other tasks. In addition, the RCDE provides the foundation for the RADIUS Testbed System.

1 Introduction

The RCDE contains facilities for CAD-system-like three-dimensional (3-D) modeling, image processing, electronic-light-table image viewing and mensuration, frame and non-frame camera photogrammetry, and photo-realistic rendering.

The cartographic features and data that can be entered include multiple images, camera models, digi-

tal terrain elevation data, point, line, and area cartographic features, and a wide assortment of 3-D objects. Interactive capabilities include free-hand feature entry, altering features while constraining them to conform to the terrain and lighting geometry, adjustment of feature parameters, and the adjustment of the camera model to display the scene features from arbitrary viewpoints.

The major achievement of the system is the seamless integration and high level of interoperability between and among these facilities. A key realization that enables this is that every entity represented in the RCDE has an associated local coordinate system. This includes cartographic and cultural features, images and subimages, text annotations, graphical user interface (GUI) elements, photogrammetric conjugate points and even the earth itself. These entities are tied together through a flexible and efficient network of coordinate transformations. This allows each type of data to be represented, manipulated, and displayed in the most convenient and precise form, without sacrificing functionality or generality, in addition to enabling the fusion of different types of geometric data.

2 History

The RCDE occupies that rarefied realm of software systems that are simultaneously uncompromising in theoretical rigor and completeness without sacrificing efficiency, that are sufficiently flexible to be amenable for use as a rapid prototyping environment in support of experimental and exploratory work, and yet of such "industrial strength" to be

*This work was sponsored by Lockheed-Martin Corporation under contracts RRM881032 and RRM960506, the Advanced Research Projects Agency under contract DACA76-92-C-034 monitored by the U.S. Army Topographic Engineering Center, Fort Belvoir, VA, and SRI International under various internal research and development projects. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency, the United States Government, or SRI International. *ImagCalc*, *TerrainCalc*, *The Cartographic Modeling Environment*, and *3DIUS* are trademarks of SRI International. All other products and company names mentioned in this paper are the trademarks of their respective holder.

applied to “real-world” problems and data sources. This achievement is in no small part due to the insight, programming skills, and tireless efforts of Lynn Quam, the RCDE’s architect and primary implementor, and the use of Lucid Common Lisp¹ and the Common Lisp Object System (CLOS) for the system’s implementation.

The RCDE runs on both Sun Microsystems and Silicon Graphics RISC-based workstations, under the UNIX operating system and uses the X Window System and OSF/Motif for user interface and display functions. The current implementation comprises over 200,000 lines of Lisp/CLOS and 6,000 lines of C,² and makes direct use of the X and Motif libraries via Lucid Common Lisp’s foreign-function interface. The system represents over 20 person-years of continuous development of software environments for image understanding, cartographic feature extraction, and visualization at SRI’s Artificial Intelligence Center. It has been honed through use as the primary vehicle for image understanding research at SRI and dozens of other academic and industrial research laboratories throughout the world, as well as in more applied work such as the infrastructure for the RADIUS Testbed System [Gerson and Wood, 1994].

A rough chronology of the system’s evolution follows.

2.1 *ImagCalc*TM (1982-1984)

ImagCalc is an image analysis system that provides flexible access to two-dimensional image processing tools, including displays at multiple resolutions, perspective projections, and a wide range of image operators. This was the first system that provided interactive image processing and manipulation tools on a single-user-workstation computer with a high-resolution bit-mapped display.

2.2 *TerrainCalc*TM (1984-1986)

TerrainCalc [Quam, 1985] is an interactive system for synthesizing realistic sequences of perspec-

tive views of real-world terrain as derived from a database consisting of geometric and photometric models. TerrainCalc introduced the idea of the *image perspective transformation* and the *fly-through*, created by texture-mapping aerial imagery onto digital terrain models.

2.3 *The Cartographic Modeling Environment*TM (1986-1990)

The Cartographic Modeling Environment [Hanson *et al.*, 1987, Hanson and Quam, 1988], commonly referred to as CME, is an environment for deriving, editing, viewing, and rendering 3-D cartographic models. This system merged the capabilities of ImagCalc and TerrainCalc with additional interactive 3-D modeling facilities of its own. CME was the first environment to make rigorous photogrammetry tools available to the IU research community. It was the original “proof of concept” for the paradigm of Model-Supported Exploitation that is basis of the RADIUS program and served as the main inspiration for DARPA’s Image Understanding Environment [Mundy *et al.*, 1992a], GE’s TargetJr. [Mundy *et al.*, 1992b], and CDI’s GLMX system.

2.4 *The RADIUS Common Development Environment* (1991-present)

At the beginning of the RADIUS program, it was decided that the only way that research results and technology from the IU research community could be successfully transitioned into a quasi-operational testbed system was to provide a common development environment that could not only satisfy the needs of the research community, but could also provide the infrastructure for the full testbed system. The government selected SRI’s Cartographic Modeling Environment as the software system, but decided that the hardware platform would be a UNIX workstation instead of the Symbolics 3600-series Lisp Machine on which the system was implemented.

A full translation of the system into C++ was originally contemplated, but a study conducted by GE Military and Data Systems Operations³ and SRI In-

¹now called *Liquid Common Lisp*TM, Harlequin’s proprietary Common Lisp product.

²An additional 98,000 lines of Lisp and 75,000 lines of C were written by SRI specifically for the RADIUS Testbed System. [Heller *et al.*, 1996]

³now Lockheed-Martin Corporation, Management and Data Systems in King of Prussia, PA

ternational estimated a cost and schedule for a C++ implementation which the government determined to be prohibitive.⁴

After some investigation, SRI determined that a high-quality Lisp compiler (e.g., Lucid Common Lisp's production compiler), if carefully used, could provide performance on par with C++, while maintaining the significant advantages of Lisp (i.e. excellent support for rapid prototyping, automatic storage management, easy maintenance and debugging, etc.).

This work culminated in the release of the RCDE version 1.0 in July 1993. This system included nearly all of capabilities of the CME, as well as the ability to utilize appropriate sensor models and imagery obtained through National Technical Means (NTM), a Lisp-to-C/C++ interface developed by GE M&DSO, and two volumes of documentation.⁵ Since then, two additional major releases of the system have been made, which, in addition to bug fixes and modifications requested by the RCDE user community, have incorporated major improvements in the system's capabilities (e.g., new image representations, improved photogrammetry facilities, including multi-image bundle adjustment, enhanced user interface) and performance (e.g., a 20-fold increase in 3-D object drawing speed, streamlined coordinate transform machinery).

⁴In retrospect, the authors believe that the estimated cost for a C++ implementation was probably too low by a factor of at least two, and, given the primitive state of C++ compilers and development tools in 1991, could not have been delivered in a timely fashion, and furthermore, would have required extensive maintenance over the intervening years to keep in sync with the evolving C++ language definition, compilers, and support libraries. In fact, "coding around" the bugs and shortcomings of current C++ compilers and support libraries (e.g. STL) has proven to be a major headache for the implementors of the DARPA Image Understanding Environment [Boult *et al.*, 1994].

⁵The *RCDE User's Manual* and *The RCDE Programmer's Reference Manual*, written by GE M&DSO, are available on the WWW through the URL <http://www.ai.sri.com/~radius/rcde/support.html>. Unfortunately they are somewhat out of date.

3 Overview of RCDE Facilities

3.1 Electronic Light Table and Image Processing

One of the basic operations of the RCDE is viewing and manipulating images. A great deal of effort has gone into making these operations fast and efficient. Image sizes as large as 2^{50} pixels are accommodated, meaning that the effective limit is determined by the amount of high-speed disk storage accessible by the system. Specifically, images can be larger than the virtual address space of the machine and larger than the maximum disk partition size supported by the host machine's operating system.

The system provides all the standard geometric image viewing functions: zoom, pan, rotate, mirror, windowing. All geometric adjustments are related back to the original image data through what is called the *2d-world*. The 2d-world is simply the coordinate system of the original imaging event.⁶ For an image taken on film with a photogrammetrically calibrated camera, film plane measurements in microns are the natural units for the 2d-world. For an image acquired with a charge-coupled device (CCD) array, line and sample pixel indices are the obvious choices. Each image generated by a geometric transformation of the original image data has an *image-to-2d-transform* that provides the mapping from pixels in the derived images back to the original image event.

Photometric adjustments such as contrast stretching and inversion are also supported, and the relationship of images produced through these operations is maintained by a *photometric-transform* that specifies the pixel value mapping between the derived image and the original.

In addition, a variety of image operators are available, such as gauss smoothing, difference of gaussians, edge and zero-crossing detectors, fast-Fourier transform, and a full complement of unary and binary pixel arithmetic and conversions.

Figure 1 shows an example user interface to the electronic light table functionality.

⁶Some systems call this the *fiducial* coordinate system.

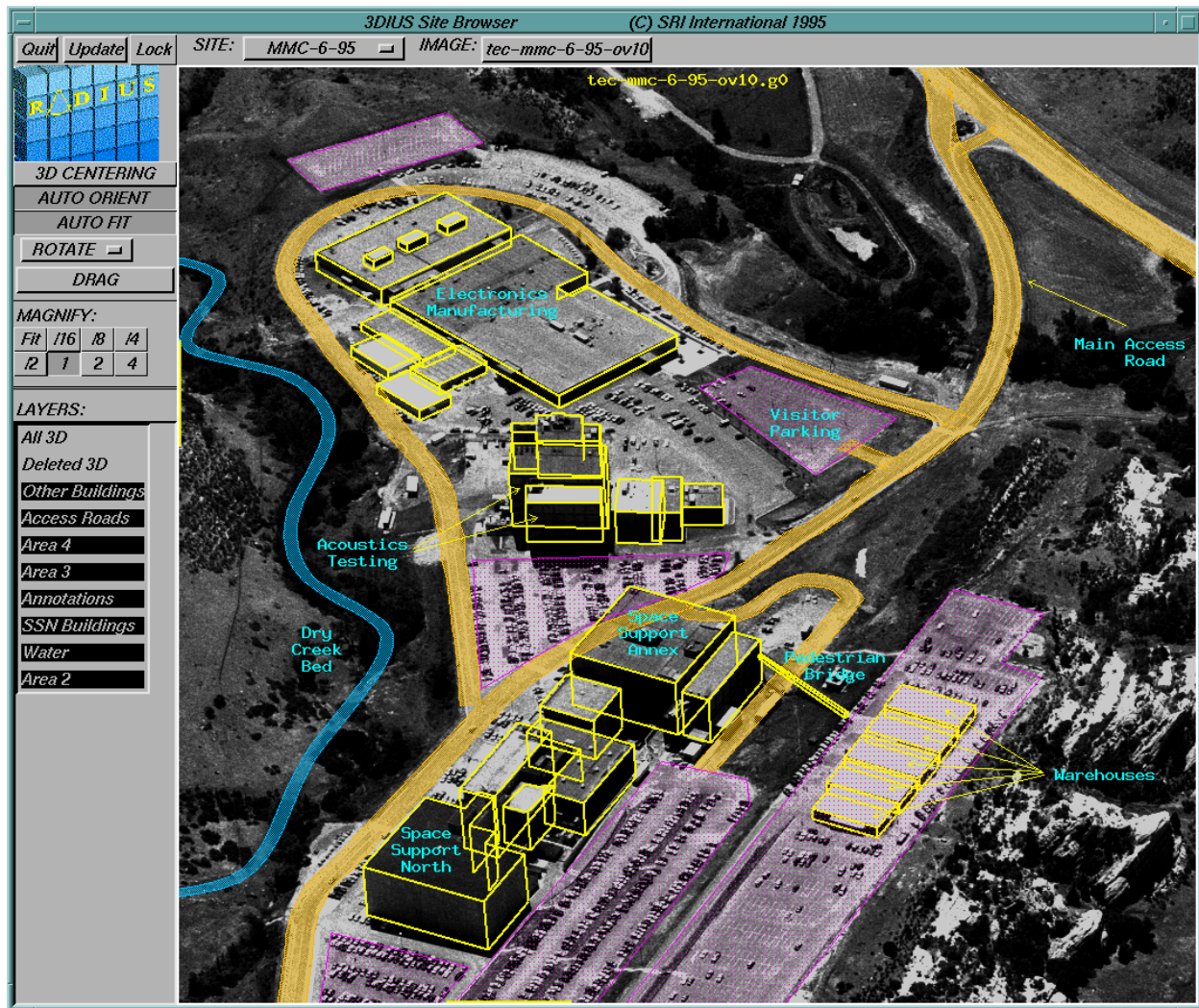


Figure 1: The Electronic Light Table (ELT) user interface. Sites and images are selected with the pull-down menus along the top. Orientation, magnification, and other viewing parameters are adjusted with the buttons on the side. The wireframe overlays can be switched on and off under control of the *Layers* menu.

3.2 Image Types and Storage Management

A variety of image types can be represented in the RCDE. While the most commonly used type is unsigned 8-bit, the system can also manipulate and display binary, 16- and 32-bit signed and unsigned, and IEEE short and long floating point images. An arbitrary number of these can be combined to form color, complex and vector images, as well as resolution pyramids. Images (and collections of images) are also used to store other types of array data such as digital elevation models.

Image pixels are stored in a blocked order (as opposed to raster order) so that the pixels in both hor-

izontal and vertical neighborhoods are likely to be stored on the same memory page. Ideally, the block size is chosen to optimize performance by matching it to the size of the memory pages used by host's operating system and hardware. Other block sizes (e.g., 1024×1024 pixel blocks) can be accommodated, but with reduced workstation performance.

Four management schemes are implemented:

Memory Mapped In this scheme, all the image storage resides in the address space of the RCDE process. This is typically used for images that are smaller than 4Mb.

File Mapped In this scheme, blocks of the image are read from a file into memory on demand. A common pool of memory pages is allocated to all images of this type, allowing an arbitrary number of images that are larger than the virtual address space of the machine to be manipulated. A number of heuristics are used by the page-pool manager to optimize the size of the pool and minimize page thrashing.⁷ IU Testbed images and certain types of NITF images can be used directly by file-mapped images. Other formats must be converted.

Lazy This is similar to the file-mapped scheme, but the blocks of the image are computed on demand. Rather than being associated with a file, this image has a filling function and one or more source images. For example, it has been used to calculate rotated images and image gradients, to apply edge-sharpening filters, and to perform on-the-fly resampling and mosaicking. The advantage is that the only those pixels actually referenced are computed.⁸ An arbitrary number of these images can be strung together to create an on-demand image processing pipeline.

Indirect These images contain no pixel memory themselves, but rather are an integer-ratio pixel mapping (i.e. no resampling) into the memory of another image. This is used to implement subimages, regions of interest, multiple-of-ninety-degree rotations, mirroring, pixel-replicated expansion, and pixel-sampled image reduction.

Each image also has a 2-D bounding polygon that indicates the actual pixel area of an image in relation to the image's 2d-world coordinate system. This feature is necessary because

- Some lenses (e.g., fish-eye) produce images that are not rectangular.

⁷The authors eagerly await widespread adoption of true 64-bit operating systems and terabyte filesystems, so that this functionality can be handled by the host machine's operating system and dropped from the RCDE.

⁸This property has been used to allow the SRI-developed Model-Based Optimization (MBO) System [Fua, 1996a, Fua, 1996b] to operate efficiently on very large images by calculating image gradients on demand and therefore only where they are actually needed by the optimization process.

- Arbitrary rotations of rectangular images can produce "no data" areas in the resulting image.⁹

3.3 Solid Modeling

The RCDE shares many features with any capable 3-D computer-aided design (CAD) system. For example, it permits the entry of 3-D objects and allows the user to interact with them in a simulated 3-D world; it allows the creation of high-quality rendered images to show how the simulated 3-D world would appear from a real camera. An example is shown in Figure 2.

The distinguishing features that set the RCDE apart from more conventional CAD systems include:

- Registration of multiple data sources, including stereographic or multiple images, terrain elevation models, and 3-D object models, to the same world coordinate system. This capability permits object model entry to be driven by *sensor* data such as actual images. A typical modeling sequence is shown in Figure 3.
- Use of lighting models, terrain elevation data, and other geometric knowledge to constrain and facilitate data entry. The exploitation of *constraints* in the interactive modeling process potentially increases the efficiency of the human operator.
- Registration of local coordinate systems to Universal Transverse Mercator (UTM), latitude-longitude, and other cartographic coordinate representations. The use of real-world coordinate systems enables the system to exploit specific world knowledge, for example, by computing the sun position for a particular location at a particular time of day.

3.3.1 3-D Shapes and Models

The RCDE supports a variety of primitive shapes for modeling cultural features. The geometric parameters (e.g., position, orientation, height, width) and presentation parameters (e.g., color, stippling) of these primitives can be interactively adjusted.

⁹In fact, through a long series of rotations and cropping operations, one could produce a round image.



Figure 2: A synthetic view of a site constructed with the RCDE. The texture maps used on the models were automatically extracted from the appropriate images. The terrain model was created by F. Raye Norvelle of the U.S. Army Topographical Engineering Center using his Iterative Ortho-Refinement (IOR) technique.

Point and Crosshair These simply specify a 3-D location. A point is drawn as a dot and the crosshair as a cross.

Cubes and Houses These shapes are rectangular prism-like structures, and include boxes, buildings, and houses. Boxes and buildings have three degrees of freedom. Houses are boxes with one (shed), two (gable), or four (hip) sloping roof surfaces in place of the top face. A roof overhang may also be specified.

Cylinders and Quonsets A cylinder is an extruded ellipse; the major and minor axis dimensions

are controlled by the x and y size parameters. A quonset is a half-cylinder, with the extruded axis lying parallel to the ground.

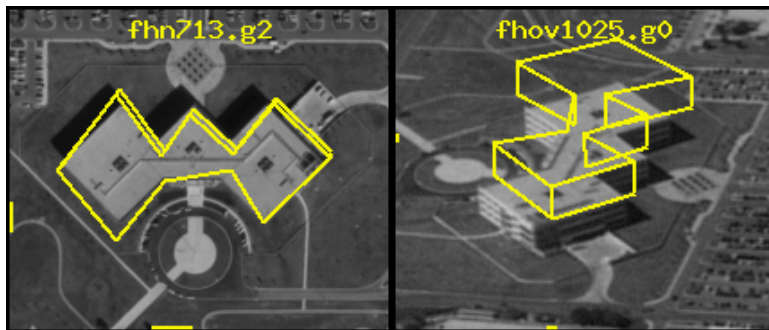
Superellipses and Superquadrics are 3-D shapes described by the equation $ax^r + by^s + cz^t = 1$. In the superellipse the constraint $r = s$ is enforced. By adjusting the parameters, a large number of shapes varying smoothly from rectangular solids, through cylinders and ellipsoids, to very “cuspy” hyperbolic shapes can be produced.



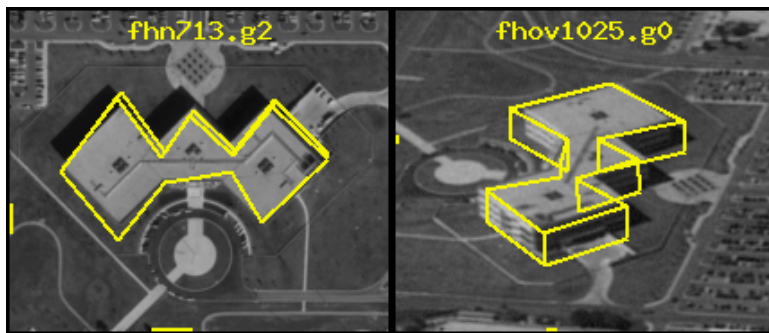
The original images.



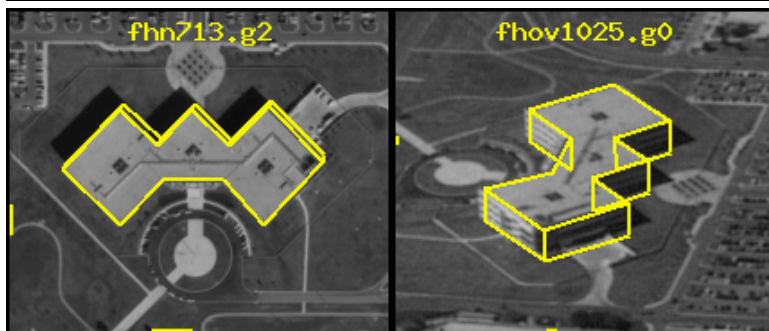
Sketch roof-line (*Add Vertex*).



Done with roof-line (*Drop*).



Correct elevation (*MBO-Z-Search*).



Optimize Shape (*MBO-Opt*)... Done!

Figure 3: The sequence of steps used to model a complex-shaped building with the extrusion primitive and the SRI-authored Model-Based Optimization system. This entire sequence typically takes less than 1 minute of elapsed time.

Polyhedra are represented as a collection of vertices and a network of edges that connect them. Closed cycles of edges can be grouped into a face and assigned independent properties, such as texture maps.

Open and Closed Curves are space curves specified by a list of 3-D vertices. The location and number of the vertices can be interactively adjusted. They are drawn either as a polyline or a spline whose tension can be adjusted. If the curve is closed, the last vertex is connected back to the first.

Extrusions are closed curves that have an additional z value associated with each vertex. This sets the length or height of the extrusion at each point.

Ribbons and Curtains are open curves that have a width associated with each vertex and are used for modeling roads, rivers, fences, and so forth. For ribbons, the parallel lines are separated horizontally; for curtains, they are separated vertically. They may be drawn directly as entered, or have a spline fit to them.

DTM Mesh This is a regular network of four- (quad-mesh) or six- (tri-mesh) connected vertices whose z values are derived from a digital terrain model.

Contour Map This object displays a set of contours of constant elevation computed from a digital terrain model. The elevation spacing can be set to any convenient quantity.

Map Grid This object displays lines of constant latitude and longitude or UTM coordinates.

Text displays a string of text at a 3-D location in a site.

Composite Objects This object controls a collection of other objects belonging to the same coordinate system. When user interface operations (such as motion, rotation, or cloning) are performed on composite objects, the operation is performed on each object in the collection.

Many of these objects have 2-D counterparts that can be used for annotation of a particular image or

to represent and display the output of 2-D feature extraction processes, such as an edge detector.

3.4 Coordinates, Coordinate Transforms, and Coordinate Systems

A uniform structure for handling coordinates, coordinate transforms, and coordinate systems simplifies the implementation, increases the generality, and improves the performance of the RCDE.

The RCDE is required to efficiently perform linear, nonlinear, and projective coordinate transformations between a large variety of coordinate systems, as shown in Figure 4. In this network, nodes (ellipses) represent coordinate systems, and directed arcs represent coordinate-transforms. Note that all the transforms shown are reversible except for the projection from local 3d-world coordinates to sensor coordinates, which will be discussed later.

3.4.1 Coordinate Vectors

Spatial coordinates are represented by vectors (one-dimensional arrays) of double floats. In a preliminary implementation of the RCDE, it was found extremely awkward to pass coordinates to functions as groups of individual scalar arguments. This was due to both the number of arguments to functions and the local variable bindings needed to hold the returned coordinates, as well as the awkward constructs needed to perform sequences of coordinate transforms.

3.4.2 Coordinate Transforms

Coordinate transforms are represented in the RCDE by using an object-oriented approach in which each coordinate transform is represented by an instance of a coordinate-transform class with slots as shown in Figure 5.

Coordinate transforms are invoked from programs by calling the generic function `transform-vector`.

```
(transform-vector transform
 from-vector &optional to-vector)
⇒ result-vector
```

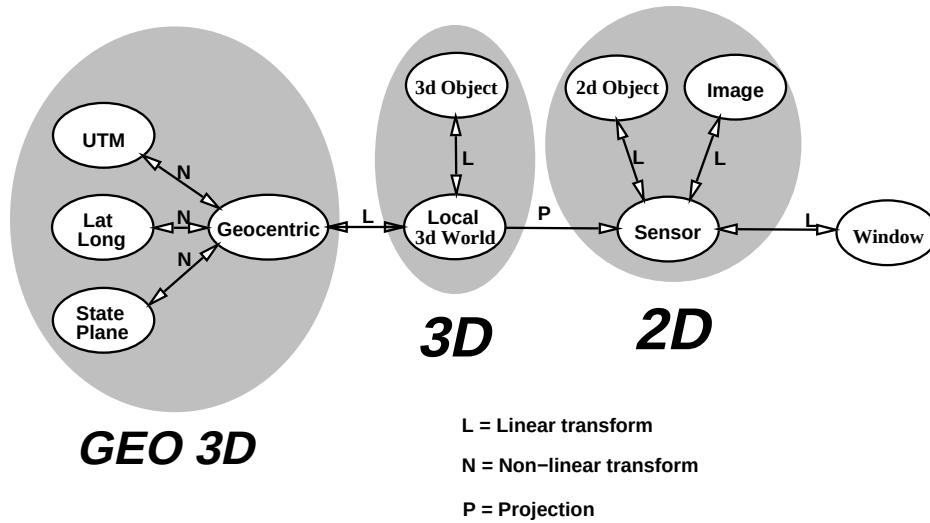


Figure 4: Coordinate Systems and Coordinate Transforms.

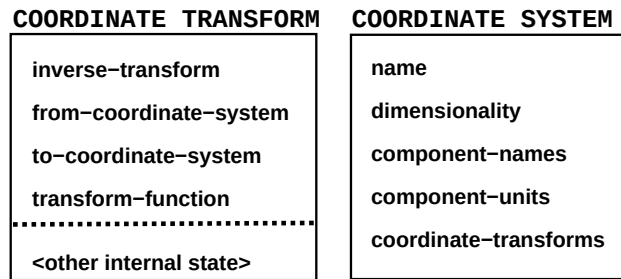


Figure 5: Coordinate Transform and Coordinate System Classes.

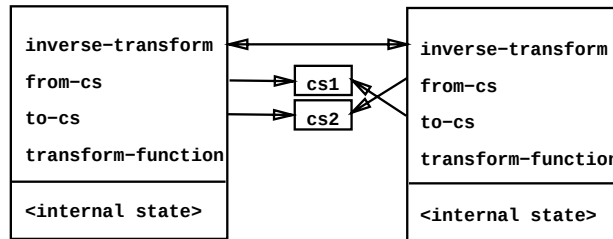


Figure 6: Coordinate Transform Pair.

The behavior of transform-vector has been designed to satisfy the following:

- **Polymorphic** — The caller is not required to know anything about the particular transform being invoked. In principle, the caller does not even have to know the dimensionality of the coordinate vectors.
- **Domain Restricted** — Any coordinate-transform is allowed to fail and return nil

(null-vector) when *from-vector* is outside the domain of the transform. For example, a central perspective camera model is allowed to fail when *from-vector* designates a point behind the camera or outside the view frustum of the camera.

- **Composition Rules** — Sequences of transforms must be easily composed. In particular, the form

(transform-vector t_n

```

    ...
    (transform-vector t2
      (transform-vector t1
        from-vector)) ...)

```

must be guaranteed to apply the sequence of transforms ($t_1 t_2 \dots t_n$) to *from-vector* and return *nil* if any of the transforms returns *nil*. (This implies that *transform-vector* returns *nil* whenever *from-vector* is *nil*).

For convenience, the form

```

(transform-vector
 transform-list from-vector)

```

where *transform-list* = ($t_1 t_2 \dots t_n$) is equivalent to the previous set of nested calls to *transform-vector*.

Invertible Coordinate Transforms Nonsingular linear transforms from n-space to n-space are invertible. Likewise, many nonlinear transforms, such as from geocentric to latitude-longitude coordinates, are invertible for vectors within the domain of the transforms. This relationship is illustrated in Figure 6.

Coordinate Projections A coordinate projection is a special type of coordinate transform that projects from a coordinate system to a lower-dimensional coordinate system. Thus, without additional information, a coordinate projection cannot be inverted. Central perspective projection (pin-hole camera) and orthographic projection are the most common forms of coordinate projection used in computer graphics.

Inverse Coordinate Projections An inverse coordinate projection transforms a coordinate vector from a coordinate system (2-D) to a higher dimensional coordinate system (3-D) by ray-tracing to the nearest point on a surface specified as an internal state variable of the inverse projection. In the RCDE, the surface is usually either a terrain model (DTED) or a plane of constant elevation. In the most general case, the ray is the locus of points in a 3-D space that project to the point of interest in the 2-D sensor coordinate system. For electro-optical

(E-O) sensors the ray is a straight line, possibly perturbed by atmospheric refraction; for synthetic aperture radar (SAR) sensors, the ray is an arc of a circle, possibly perturbed by refraction.

3.4.3 Coordinate Systems

Coordinate system objects are selectively used by the RCDE for representing the network that connects coordinate systems by coordinate transforms, for user interface access to information such as the coordinate system name and the coordinate component units and names, and for run-time error checking.¹⁰

The generic function *find-transform* searches the network accessible from *from-coordinate-system* to find a path to *to-coordinate-system*.

```

(find-transform
 from-coordinate-system
 to-coordinate-system) ⇒ transform

```

When multiple paths exist between the coordinate systems, the choice can be resolved based on minimal number of transforms in the sequence, minimal transform error, or minimal computation time.

All the commonly used cartographic coordinate systems, reference spheroids, geoids, and datums, and appropriate transformations between them, are available in the RCDE and can be manipulated with the machinery described here.

Each 3-D model object is parameterized in its own local rectangular coordinate system and has an *object-to-world-transform* that encodes its relationship to the site's *local-vertical-coordinate-system* (LVCS). The LVCS is a right-handed, Cartesian coordinate system (usually oriented so that z is up and y points north) that is established for each site before it is modeled. The LVCS in turn, has an *LVCS-to-geocentric-transform* that is used to relate it to a geocentric coordinate system, and ultimately to geographic coordinate systems such as latitude, longitude, and elevation in WGS-84 (the geographic coordinates used by the Global Positioning System).

¹⁰The implementation of *transform-vector* does not require the use of coordinate system objects. If omitted, the run-time error checking is disabled.

Through this chain of transformations, every point of every model object is fully georeferenced.

3.4.4 The RCDE Coordinate Transform Pipeline

The RCDE implements the coordinate transform pipeline shown in Figure 7. The shaded ellipses indicate the major points of contact between the RCDE user interface and the coordinate transforms and object representations.

3.5 Sensor Models

The dominant use of a sensor model in the RCDE is to project 3-space coordinates to image coordinates required during wire-frame model rendering. To support smooth interaction with the object models, this projection must be fast (i.e., under $50\mu s$). For central projection cameras, the 3-D to 2-D projection can be accomplished with as few as 20 floating-point operations, taking less than $5\mu s$ on a Sun SPARCstation.

The RCDE can also accommodate images generated by dynamic sensors, in which the image formation process involves movement of parts of the sensor or the entire sensor platform itself, such as the strip cameras used for aerial mapping or the imaging systems used in photo-reconnaissance satellites [McDonald, 1995]. Full mathematical models of sensors of this type typically require two orders of magnitude more computation than for a central projection camera. To address this problem, the RCDE has facilities to fit and use a piecewise polynomial approximation (commonly referred to as the *fast block-interpolation projection* or FBIP) to the full mathematical model, as well as being able to import and make use of rational polynomial function approximations (RPCs) generated by photogrammetric workstations such as GDE's SOCET SET.

Since the RCDE must deal with a variety of image and sensor types, a very general interface framework was developed. The only explicit knowledge assumed about the sensor model is obtained from either its world-to-sensor projection function $P(X)$ or its sensor-to-world projection function $P^{-1}(U)$. The actual implementation of the projection func-

tions is arbitrary, so long as specific mathematical properties are maintained. In particular, the projection functions P and P^{-1} are expected to be well behaved such that the local differential geometry expressed by the Jacobian matrix of the projection function characterizes a fairly large local neighborhood.

In general, there is no explicit knowledge of sensor position, sensor orientation, focal length, and so forth. We have found that most calculations that use these parameters in graphical user interfaces and IU algorithms can be replaced by counterparts derived from only the projection function and its Jacobian matrix. Furthermore, given two sensor models, there is no global concept of epipoles or epipolar geometry. However, in local neighborhoods, the epipolar relationship between two images can be computed from their Jacobian matrices.

The two primary application program interface (API) functions are `project-to-view` and `project-to-world`. `Project-to-view` takes a 3-D point and returns the image plane coordinates of the projection of the point. Every time a 3-D object is drawn on top of an image, this function is called for each vertex of the object.¹¹

`Project-to-world` is somewhat more complicated. It first computes the ray or curve¹² in space that is defined by the locus of points that project to the given point in the image plane, and then computes the intersection of that ray or curve with a surface, such as a terrain model or a plane of constant elevation. If this intersection fails, the function returns `nil`; otherwise, it returns the point of intersection. This functionality can be exercised by using the mouse to click on an image being displayed by the RCDE. In response, the RCDE displays both the image coordinates and apparent 3-D coordinates of the selected point.

One of the main advantages of this generic, sensor-model-independent API is that code can be developed and tested in an unclassified environment with unclassified imagery, and then used in classified

¹¹A caching scheme, described later, eliminates recomputing the calls to `project-to-view` except when 3-D coordinates or camera models are modified.

¹²This occurs in the case of SAR imagery or when the 3-D coordinate system is non-Cartesian, as is the case with geographic coordinates.

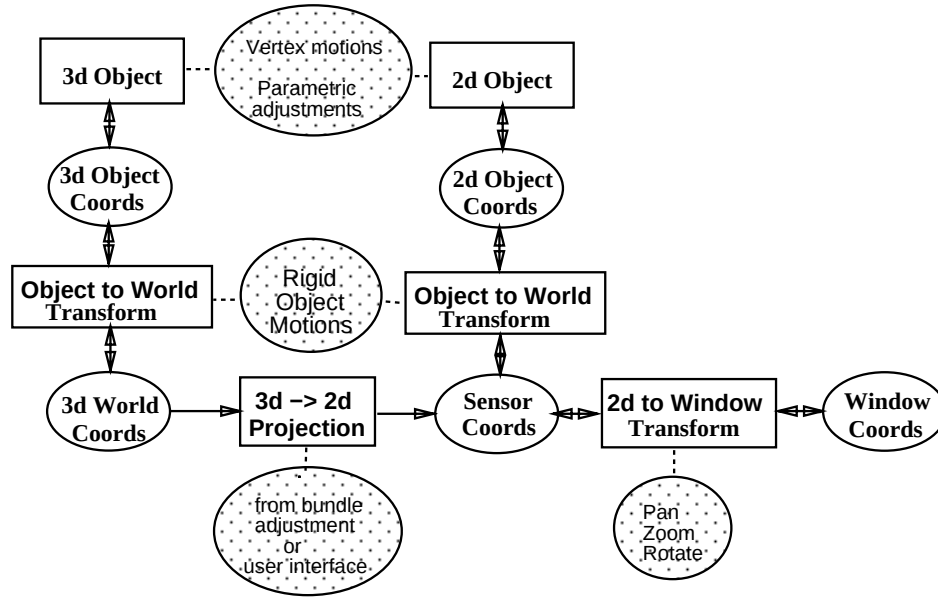


Figure 7: Standard Coordinate Transform Pipeline.

imagery environments without significant modification. This ability has been key to the transfer of IU technology from universities into the RADIUS Testbed System.

3.5.1 Central Perspective Camera

This is the standard eleven-parameter “pinhole” camera model. It can be directly instantiated from the internal and external parameters ($x, y, z, \omega, \phi, \kappa$, *principal-point-u*, *principal-point-v*, *focal-length*, *skew*, *aspect-ratio*),¹³ from an orthonormal 4×3 projection matrix, or as the result of a space resection of an image. The RCDE also includes facilities for decomposing arbitrary projection matrices (such as those arising from a direct linear transform resection) into standard parameters. In addition, there are a number of user interface functions that can be used to dynamically adjust projections to create synthetic views of the modeled scene.

3.5.2 Orthographic Projection

The RCDE handles orthographic projections as a special case of the central perspective projection,

¹³For most applications, *skew* is fixed at zero and *aspect-ratio* is fixed at 1.

with the inverse of the focal-length, $1/f$, set to zero. These projections are commonly used to import existing maps into the RCDE or to produce 2-D “campus maps” from 3-D site models.

3.5.3 Fast Block Interpolation Projection

As mentioned earlier, the full math models for some sensor types are computationally so complex as to make them unsuitable for use in an interactive system. The solution used in the RCDE is to employ an existing (but too slow) implementation as a “black box” for generating tables for the new, faster (arbitrarily precise) approximation. In all of the known existing math models, it is relatively inexpensive to compute the ray in space corresponding to a given pixel. Projection from a point in 3-space to the image may be much more expensive.

The multiplane fast block projective model divides the image into rectangular blocks of some specified size, and defines a set of planes (usually three planes) of constant elevation in 3-space that span the range of elevations of the terrain. Thus 3-space is divided into volume cells defined by the paths of the camera rays in 3-space at each of the block corners in the image, and the elevation planes in

3-space. The approach makes no explicit assumption about the sensor and, furthermore, the coordinate system is not required to be Cartesian, allowing the direct projection from WGS-84 <longitude, latitude, elevation> to image coordinates.

For each elevation plane of each block, eight parameters, (a, b, c, d, e, f, g, h) , define the projection of a 3-space point in that plane into image coordinates:

$$u = \frac{ax + by + c}{gx + hy + 1} \quad (1)$$

$$v = \frac{dx + ey + f}{gx + hy + 1} \quad (2)$$

An arbitrary point in 3-space point $\langle x, y, z \rangle$ is projected to image coordinates in the following steps:

- Determine which block contains the 3-space point. In general, this involves a search over many or all image blocks and the following nontrivial computation within each image block.
 - For each block corner, interpolate the corner position at the elevation of the given 3-space point. The interpolation formula is of the form

$$x = x_0 \frac{(z - z_1)(z - z_2)}{(z_0 - z_1)(z_0 - z_2)} + x_1 \frac{(z - z_0)(z - z_2)}{(z_1 - z_0)(z_1 - z_2)} + x_2 \frac{(z - z_0)(z - z_1)}{(z_2 - z_0)(z_2 - z_1)} \quad (3)$$

$$y = y_0 \frac{(z - z_1)(z - z_2)}{(z_0 - z_1)(z_0 - z_2)} + y_1 \frac{(z - z_0)(z - z_2)}{(z_1 - z_0)(z_1 - z_2)} + y_2 \frac{(z - z_0)(z - z_1)}{(z_2 - z_0)(z_2 - z_1)} \quad (4)$$

- Perform a point-in-polygon test to determine if the given 3-space point is within this block.
- For each elevation $\langle z_i \rangle$ in the block, compute the $\langle u_i, v_i \rangle$ projection of the 3-space point.

- Perform an n^{th} -order interpolation (n being the number of elevation planes) of the $\langle u_i, v_i \rangle$ image locations with z being the variable of interpolation. For three elevation planes we have

$$u = u_0 \frac{(z - z_1)(z - z_2)}{(z_0 - z_1)(z_0 - z_2)} + u_1 \frac{(z - z_0)(z - z_2)}{(z_1 - z_0)(z_1 - z_2)} + u_2 \frac{(z - z_0)(z - z_1)}{(z_2 - z_0)(z_2 - z_1)} \quad (5)$$

$$v = v_0 \frac{(z - z_1)(z - z_2)}{(z_0 - z_1)(z_0 - z_2)} + v_1 \frac{(z - z_0)(z - z_2)}{(z_1 - z_0)(z_1 - z_2)} + v_2 \frac{(z - z_0)(z - z_1)}{(z_2 - z_0)(z_2 - z_1)} \quad (6)$$

The multiplane block projective interpolation scheme is expensive to compute, especially when the image block containing the 3-space point is not known. To improve performance, the following heuristics have been implemented:

- Spatial and Temporal Coherence: When projecting a new 3-space point, first try the image block of the last point projected. If that fails, re-project the point using the parameters from the current block. Repeat until the point projects into the correct block. This process usually converges in two iterations.
- Vertex Cache: When projecting a vertex of a CME object, save the projected image position in a cache (hash table) associated with that vertex. Subsequent projections of that vertex are avoided by using the cached image coordinates. Of course, the cache must be flushed if the 3-space coordinates of the vertex or the camera parameters are changed.

3.5.4 Composite

Any one of the preceding projections can have an arbitrary sequence of 3-D-to-3-D transformations prepended to it and an arbitrary sequence of 2-D-to-2-D transformations appended to it to form a composite projection. Where possible, elements of the

sequence are automatically collapsed to increase the computational efficiency of the overall projection. 3-D-to-3-D transformations are typically used to adjust nonparametric projections (e.g., FBIP) during image resections and bundle adjustments or to create dynamic views that retain the overall geometric qualities of the original projection. Nonlinear 2-D-to-2-D transformations are typically used to model the optical distortions in lenses. Linear 2-D-to-2-D transformations are used to define the transformation from 2d-world (original image) coordinates to displayed image coordinates, and from displayed image coordinates to screen coordinates.

3.6 Image Registration

Image registration is the process of determining and/or refining the internal and external parameters (e.g., position, orientation) of the sensor used to acquire the image. The RCDE has facilities to perform single-image space resection, closed-form direct linear transform resection, and multiple-image bundle adjustment in a covariance-weighted least-squares framework [Mikhail, 1976, Slama, 1980]. If accurate, georeferenced ground control (possibly augmented by image-to-image tie points) is available, absolute parameters are derived. If only image-to-image tie points are available, relative parameters are computed.

The primary interface to this functionality is the Registration Tool. The tool adds no new underlying functionality to the RCDE, but rather packages existing operations in a much more user-friendly fashion. The tool is designed to lead the user through the necessary steps to register a new image, without detailed knowledge of the intricacies of photogrammetry.

The Registration Tool is shown in Figures 8, 9, and 10. The main panel contains three image display panes as well as a few menu items. They show the full image being operated on and enlarged views of the current conjugate point on both the image being registered and a previously registered reference image.

The primary object used for image registration is the conjugate point. This object is unique in the RCDE in that it has both 2-D and 3-D components. It is used to enter both image-to-ground correspon-

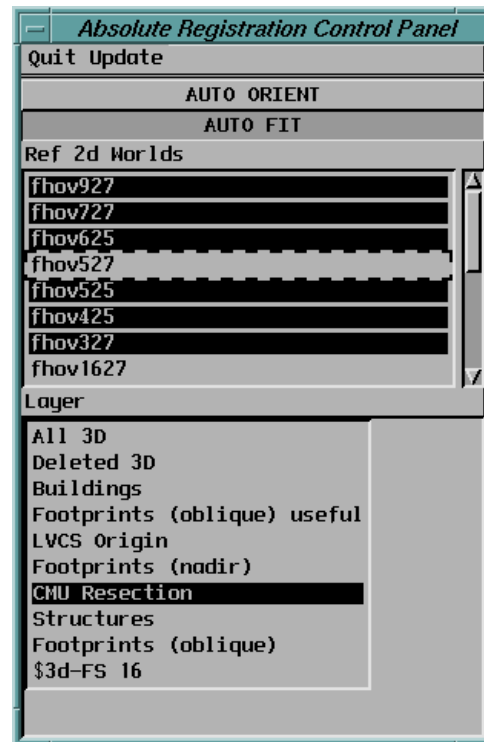


Figure 9: The Registration Tool Control Panel. This panel is used to select the reference images and overlay layers for the display of the ground control points.

dences (ground control points) and image-to-image correspondences (tie points) and to specify the *a priori* error estimates for these quantities.

The 3-D component of the conjugate point is drawn as a “+”-shaped crosshair whose image overlay position is computed by projecting its 3-D coordinates through the image’s current camera model into the coordinate system of the image. The 2-D component is drawn as an “×”-shaped crosshair whose image overlay position is set manually through mouse interactions. When the projection error is large, a line is drawn between the crosshair and the × to visually group the two components together. When the mouse is placed on a conjugate point, the projection error for that point is displayed. Each conjugate point has a single 3-D coordinate and may have a 2-D coordinate for each image in which its 3-D location is visible. A recent modification to the RCDE allows conjugate point information to be associated with any vertex of a planar faceted object. This was done to support entry of error estimates in modeled objects, as well as to support experiments



Figure 8: The Registration Tool Main Panel

in using extended features for controlling images. If the Model-Based Optimization system [Fua, 1996a, Fua, 1996b] is loaded into the RCDE, it can be used to refine the 2-D position of the conjugate point.

3.7 The Eval-Cache

Another key feature of the RCDE is a memoization facility called the *eval-cache*. It is implemented as a relational network with data objects as the nodes and functions and methods as the edges. This provides a uniform method for caching the result of “expensive” computations. For example, if the Fast-Fourier Transform (FFT), gaussian smoothing, or decimation of an image has already been computed, subsequent invocations of these operations with identical parameters return the cached image instead of recomputing it. It is also possible to declare that a particular function is the inverse of another function. For example, the if a user invokes

the inverse FFT function on an image which resulted from the application of a forward FFT, the original image is immediately returned.¹⁴ The eval-cache can also be directly queried for a data object’s history. This returns the sequence of operations and parameters that were applied to the object and how the object was created or loaded into the system. In addition, when a given function is recompiled (a common occurrence during debugging), the cache is automatically flushed of all the results produced by that function.

Another subtler, but perhaps more important use of the eval-cache comes into play when a function consists of a number of sub-functions. If each of the sub-functions is cached, repeated invocations of the main function with different parameters results in the recomputation of only the intermediate results

¹⁴This can be a bit unnerving to the novice user, who may be unaware of the existence of the eval-cache.

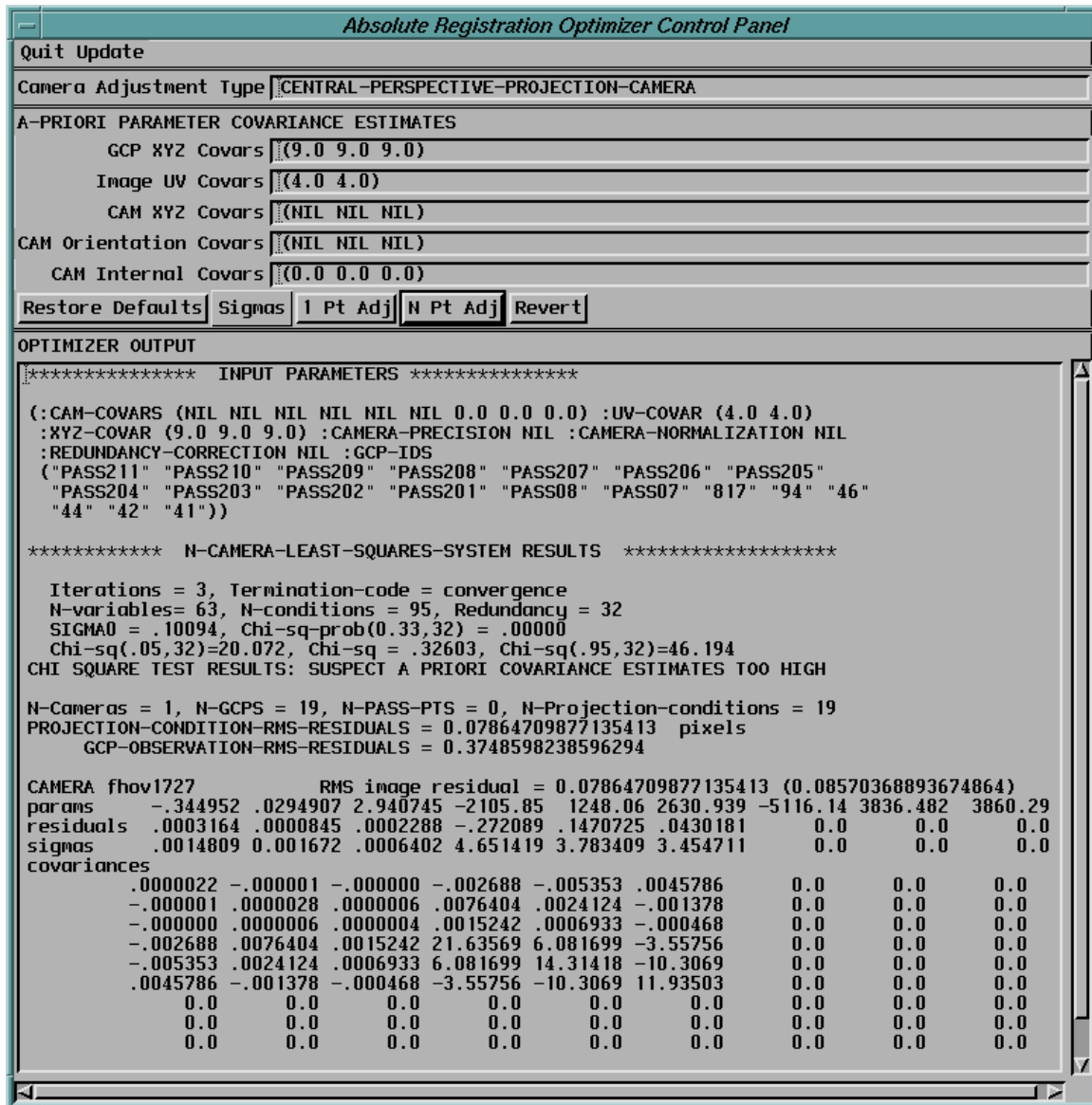


Figure 10: The Registration Tool Optimizer Panel. A typical result of a single-image resection is shown.

that are affected by the changed parameters. The programmer or user just specifies the sequence of operations and the system automatically performs the optimizations.

When very-large images are being used with the RCDE, this facility, in combination with the lazy-image operations, can result in tremendous savings of both computation time and memory.

3.8 System Support

A number of utilities to support code development, debugging, distribution, loading, creation of site-

independent disksaves, and datasets, etc. that were part of Genera (the Symbolics Lisp Machine operating system) but not available in Lucid Common Lisp or the UNIX operating system were developed for the RCDE. These include:

- A forward and reverse pathname translation facility that allows pathnames to be read from and written to persistent storage with embedded environment variables. This is essential to facilitate code, image, and site-model distribution to other organizations using the RCDE to allow for varying file system names, network topologies, and constraints on storage of large

amounts of data.

- A patch facility that allows bug fixes and enhancements to the system to be easily distributed without creation and distribution a new RCDE executable. It enables the changes to be packaged into small files for distribution via FTP that are automatically incrementally linked into the executable when the system is started. This capability proved to be invaluable for providing “rapid turnaround” modifications to the RADIUS Testbed System. These patches can also be installed into a running copy of the RCDE (see discussion below).
- A graphic data structure browser.
- A system construction facility that allows the specification of loading and compilation procedures for layered subsystems of code (e.g., MBO, photogrammetry, NITF) as well as interdependencies among the subsystems.
- A facility that streamlines the creation of distribution tapes for both code and data sets.

In addition, a development environment was build around the Lucid Common Lisp System, consisting of GNU Emacs, Ilisp, CVS, and other GNU utilities, that is distributed along with the RCDE. This allows us to keep a number of sets of the RCDE and RTS source code synchronized. Combined with the facilities already present in the Lucid Common Lisp System, it provides excellent support for rapid prototyping, code optimization, and debugging, through the use of the source code tracking, profiling, and tracing tools, data structure browsers, and incremental compilers and linkers.

One of the unique capabilities of high-performance Common Lisp systems in general, and Lucid Common Lisp in particular, is the support for incremental compilation and linking of both Lisp and C. The value of this in a system that deals with very large datasets, such as the RCDE, cannot be overemphasized. By allowing patches to be installed into and experimental changes to be made to a live, running copy of the RCDE, without corrupting the dynamic state of the system execution (thereby, among other things, avoiding the need for the the data being used

to be reloaded), programming and debugging productivity is increased by at least an order of magnitude over conventional “segfault-modify-compile-relink-restart” types of code development environments.¹⁵

4 Conclusions

Throughout the development of the RCDE and the systems that preceded it, a number of innovative ideas and techniques were introduced. Many of these have become adopted as “standard practice” and now appear in commercial systems.

- Creation of an extensible, interactive, object-oriented, workstation-based image processing system.
- Tiled memory layout for image storage.
- The use of a memoization facility in an image processing system (the eval-cache).
- The use of texture-mapping to perform image perspective transformations and produce “fly-through” sequences.
- Joining CAD-system-like interactive 3-D modeling facilities and and image processing facilities in a photogrammetrically rigorous and georeferenced framework.
- Object-oriented implementation of coordinate-system representations and coordinate-system transformation operations.
- The design and implementation of the generic sensor-model API.
- Lazy-evaluation image storage and operators.

In a paper of this length, we could not possibly cover all the functionality present in the RCDE and have therefore concentrated on what we feel are some of the key ideas in the system. A wealth of additional functionality is available (e.g., vector, matrix, and quaternion math, and computational geometry libraries, an astronomical ephemeris, GUI

¹⁵With the possible exception of some implementations of Smalltalk and Dylan, as well as some experimental, purely-functional languages such as Erlang, this capability is unique to Common Lisp systems, and in particular is absent from all known implementations of C++ and Java.

libraries, an I/O sub-system for a variety of image and data formats, nonlinear systems solvers). In fact, one of the main challenges in assembling the RADIUS Testbed System was building simple user interfaces that allow the functionality in the RCDE to be used by someone who does not have a deep understanding of all the underlying theory and technology.

Very few IU projects have had the level of success of RADIUS in moving code from a research environment into a quasi-operational environment where it could be directly evaluated by image analysts on operational imagery. It is safe to say that without the RCDE there would be a great deal less code from the IU research community in the RADIUS Testbed System and much more time would have been spent in getting the code to interoperate. While a number of systems, both commercial and research, provide various subsets of the functionality of the RCDE, none provide the breadth and depth of facilities or can match it in the ease with which it can be extended and adapted to new applications.

The system continues to be developed, and we anticipate producing a version that runs on the Sun UltraSPARC under Solaris in the near future, as well as making use of OpenGL and new high-performance hardware for much of the graphics.

Acknowledgments

The authors wish to thank Christopher Connolly, Marty Fischler, and Oscar Firschein for reading preliminary drafts of this paper and making many suggestions that improved it. We would also like to thank Thomas Strat and Donald Gerson for encouraging us to write it.

References

- [Boult *et al.*, 1994] Terrance E. Boult, Samuel D. Fenster, and Jason W. Kim. Dynamic Attributes, Code Generation and the IUE. In *Proceedings of the DARPA Image Understanding Workshop*, pages 405–422, 1994.
- [Fua, 1996a] P. Fua. Cartographic Applications of Model-Based Optimization. In *Proceedings of the DARPA Image Understanding Workshop*, Palm Springs, CA, February 1996. Morgan Kaufmann.
- [Fua, 1996b] P. Fua. Model-Based Optimization: Accurate and Consistent Site Modeling. In *XVIII ISPRS Congress*, Vienna, Austria, July 1996.
- [Gerson and Wood, 1994] D.J. Gerson and S.E. Wood. RADIUS Phase II: The RADIUS Testbed System. In *Proceedings: Image Understanding Workshop*, volume I, pages 231–237, Monterey, CA, 1994. ARPA, Morgan Kaufman.
- [Hanson and Quam, 1988] A.J. Hanson and L.H. Quam. Overview of the SRI Cartographic Modeling Environment. In *DARPA Image Understanding Workshop*, pages 576–582. Morgan Kaufman, 1988.
- [Hanson *et al.*, 1987] A.J. Hanson, A.P. Pentland, and L.H. Quam. Design of a prototype interactive cartographic display and analysis environment. In *DARPA Image Understanding Workshop*, pages 475–482. Morgan Kaufman, 1987.
- [Heller *et al.*, 1996] A. J. Heller, P. Fua, C. Connolly, and J. Sargent. The Site-Model Construction Component of the RADIUS Testbed System. In *Proceedings of the DARPA Image Understanding Workshop*, pages 345–355, 1996.
- [McDonald, 1995] R.A. McDonald. CORONA: Success for Space Reconnaissance, a Look into the Cold War, and a Revolution for Intelligence. *Photogrammetric Engineering and Remote Sensing*, 61(6):689–720, 1995.
- [Mikhail, 1976] E.M. Mikhail. *Observations and Least Squares*. University Press of America, Lanham, New York, London, 1976.
- [Mundy *et al.*, 1992a] J. Mundy, T. Binford, T. Boult, A. Hanson, R. Beveridge, R. Haralick, V. Ramesh, C. Kohl, D. Lawton, D. Morgan, K. Price, and T. Strat. The Image Understanding Environments Program. In *Proceedings of the DARPA Image Understanding Workshop*, pages 185–226, 1992.
- [Mundy *et al.*, 1992b] J.L. Mundy, A. Noble, C. Marinos, V.D. Nguyen, A.J. Heller, J. Farley, and A.T. Tran. An object-oriented approach to template guided inspection. In *CVPR92*, pages 386–392, 1992.

- [Quam, 1985] L.H. Quam. The Terrain-Calc System. In *DARPA Image Understanding Workshop*, pages 327–330. Morgan Kaufman, 1985.
- [Slama, 1980] C.C. Slama, editor. *Manual of Photogrammetry*. American Society of Photogrammetry, fourth edition, 1980.