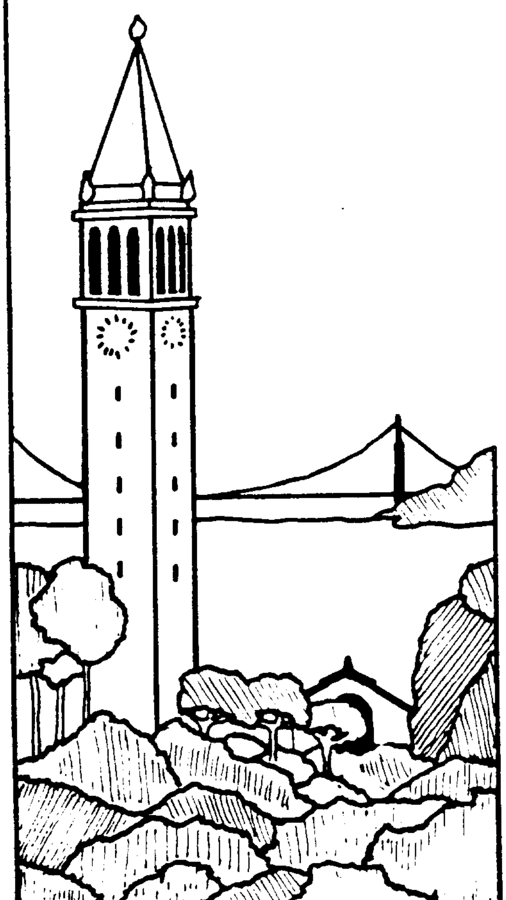
JESSIE:
AN INTERACTIVE EDITOR FOR UNIGRAFIX

H. B. Siegel



Report No. UCB/CSD 86/279

December 1985

Computer Science Division (EECS)
University of California
Berkeley, California 94720

JESSIE:
AN INTERACTIVE EDITOR FOR UNIGRAFIX

H.B. Siegel

Master's Project Report
Under the Direction of
Prof. Carlo H. Séquin

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
December, 1985

ABSTRACT

Jessie, (Geometric Construction Editor) is a tool for the creation and modification of *UNIGRAFIX* objects and scene descriptions. It is an interactive 3-D editor running under SunTools. *Jessie* has operations to incrementally create and maintain a hierarchical scene representation. It supports a large set of transformation and alignment operators to transform portions of the scene tree, both by eye and by exact placement. The command language includes some geometric operators to help the user construct exact representations, that are not limited by the screen resolution. *Jessie* has an easily expandable input language and flexible menu structure.

This report consists of several parts: As an introduction, one should read first the User's Guide. For further information on maintaining, debugging, or customizing *Jessie* see *The Jessie Design Manual.*

This work was supported in part by
the Semiconductor Research Corporation.

Jessie: An Interactive Editor for Unigrafix
The User's Guide

H.B. Siegel

Master's Project Report
Under the Direction of
Prof. Carlo H. Séquin

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
December, 1985

1. INTRODUCTION TO JESSIE

This manual assumes a working knowledge of:

[1] The BSD Unix operating system as implemented on a Sun workstation [1],

[2] The *UNIGRAFIX* ascii scene representation format, including hierarchical models and transformations

Potential users unfamiliar with one or both should spend some time learning the peculiarities of BSD and *UNIGRAFIX* before attempting to learn *Jessie*.

Jessie is tool for the creation and modification of *UNIGRAFIX* objects and scenes. It fulfills two roles in the *UNIGRAFIX* universe. *Jessie* may be used as a scene compositor, assembling smaller scenes and objects into a cohesive whole. Because *Jessie* is also an editor, it allows creating and modifying individual parts of the scene. The command language is a superset of *UNIGRAFIX*, so *Jessie* can edit any scene created by any other *UNIGRAFIX* tool.

Jessie 1.0 currently runs only on the Sun workstations under SunTools, similar to other Sun tools such as Icontool and Gremlin. It is memory and processor intensive and runs best on a single-user workstation. *Jessie* may be opened, closed, or even cloned on the same workstation.

The tool window is broken up into several subwindows. The graphics subwindows make up most of the tool area, with a smaller area for command confirmation and menus. *Jessie* 1.0 supports four independent views on the same scene tree and a flexible menu/prompting scheme for the user interface. All picking and selection of *UNIGRAFIX* objects takes place in one of the four graphics areas, or *views*. Command entry is through the menu panel or alternatively by typing directly into the command line. The user may mix and match command entry styles - a fast typist may prefer to type commands as opposed to most novices who prefer the safety of a prompted menu. *Jessie* may even be used non-interactively by feeding it scripts containing commands.

1.1. Buttons and Cursors

Mouse Buttons	
Left	Used for all command menu buttons, *selection* of vertices and points on the graphics screens, positive valuator values, and confirmation queries.
Middle	Used for *extraction* of coordinates for **contour** and **face** commands.
Right	Used for negative valuator values, and pull-down menu selection on the command menus.

All menu commands and most picks are through the left-hand mouse button. The middle and right-hand buttons are occasionally used, as described in the above table.

Cursors and Icons	
Pointing Hand	The *Command Selection* cursor informs user that the system is waiting for for a command to be selected from the menu.
Coffee Cups	The *Patience* cursor informs the user that a relative slow operation is currently executing, e.g. reading a large file. The animated steam rising from the coffee cups shows that the system is still running.
Quill Pen	The *Write* cursor informs the user that a file is being written.
Gnomon	The *Dials* cursor is displayed while the cursor is in the dials area.
Target	The *Target* cursor is the default cursor in the *views*. Used for selection of vertices.
Eyeglasses	The *View* icon is displayed in the upper left-hand corner of the current view. It is also the cursor in the graphics area while the system is waiting for the user to select a view via the pick *view* command.
Map	The *Map* cursor informs the user that the system is in *map* mode. The cursor represents a ray penetrating a face.
O.K.?	The *confirm* cursor informs the user that the system is waiting to confirm the execution of the current command. Hit the left mouse button to confirm and middle or right to abort.

The cursor that tracks the mouse informs the user of the current function of the mouse: a small crosshairs when *Jessie* wants the user to pick an object vertex, a pointing hand for picking commands, a mouse image for the confirmation signal, etc. The status line, error line, prompt line, and current cursor work together to inform the user of the current status.

1.2. The Use of Color

Screen Objects and Colors	
Vertices	Green
Wires	Magenta
Faces	Blue
curInst	*White*
Other Instances	Red
Select Set	Green
New Contours	Yellow
Gnomon	Magenta
Background	Gray

Jessie uses color heavily to key the object types; this may preclude use of a monochrome workstation, depending on the perceptual skill of the user. Like MAGIC, the Berkeley VLSI design system [3], *Jessie* uses a highlit/lowlit scheme for identifying the currently editable objects among all other objects. The rule is simple and works well on a color display: The bright objects can be changed through various commands and the dim objects are untouchable. The bright and dim objects make it easy to tell the current position in the tree.

Jessie runs on monochrome Suns, but currently there is no support for special keying on these devices.

2. BASIC OPERATIONS

The core features are simply the *UNIGRAFIX* commands that describe scenes. There is a duality between the *UNIGRAFIX* descriptive format and the *Jessie* command language. All *UNIGRAFIX* object descriptions are a subset of *Jessie* commands. The following statements are valid *UNIGRAFIX* input as well as *Jessie* commands:

v *vAlpha 0 1 2* ;
v *vBeta 3 4 5* ;
w *wirename* (*vAlpha vBeta);*

2.1. Primitive Constructors

A naive user could theoretically use just the standard *UNIGRAFIX* statements to build scenes of arbitrary complexity. This set of commands is called the *Primitive Constructors*. They allocate and name new structures that are available for use later in the session. Any object assembled with the primitive constructors may be referred to by its name or by picking it in a view.

Unlike *UNIGRAFIX*, which ignores the identifiers for everything except vertices, *Jessie* uses them for uniquely identifying the object within a definition.

Primitive Constructors (standard *UNIGRAFIX* statement)
vertex [id]
wire [id] (vertex-list) { (vertex-list) }* [colorId];
face [id] (vertex-list) { (vertex-list) }* [colorId] [illum];
instance [id] (defid {transform-list}) ;
def id;
end;
color [id] intensity [hue [saturation [translucency]]] ;
light [id] intensity [x y z [h]];

A careful reader might have noticed that, unlike standard *UNIGRAFIX*, the identifiers for vertices are optional. In *Jessie*, all identifiers are optional, and the parser will supply an internally generated, unique, cryptic identifier to any new, unnamed object. If the user prefers more sensible names, he should name the object as it is created or use the **rename** command to replace the internal identifier. The internally generated names are of the form g#x#y, where *x* is the number of symbols generated since the beginning of the session and *y* is a unique number associated with the session process id. The exact identifier is unimportant, suffice it to say that it is unique.

All Primitive and Advanced Constructors return the identifier of the object that they create. This allows a LISP-like nesting of these commands, greatly extending the scope and power of standard *UNIGRAFIX*. *Jessie* takes advantage of this by constructing commands like:

f fid (v a 0 1 0 v b 1 0 0 v c 2 3 0) ;

which is equivalent to:

v a 0 1 0;
v b 1 0 0;
v c 2 3 0;
f fid (a b c);

The vertices that are constructed on-the-fly are syntactically and semantically the same as regular *UNIGRAFIX* vertices. They may be named explicitly or implicitly by letting *Jessie* generate internal names. *Jessie* will generate statements like the one above in response to a command that creates a face out of new vertices, e.g. creating a new face from points that generated from some pointing action.

The generation of vertex coordinates is much more flexible than illustrated above. As described in later sections, the coordinates for a vertex may also be extracted from any other vertex, or generated by mapping a mouse-pick onto an existing face. Several commands use the *vertex-list* structure so it is worth noting here. The literals are in italics.

vertex-list ::= vertexId vertex-list
| vertex-command vertex-list
| ***map*** **faceId vertex-command** ***end_map*** **vertex-list**

The first alternative should be familiar to *UNIGRAFIX* users - it is a list of previously specified vertex identifiers. The second alternative is the direct interpolation of vertex commands, as described in the example above. The third choice allows mapping of vertices onto an existing face. This is useful for drawing contours (cutting holes). The **contour** command, described later, uses this command to assure the new vertices lie on the prescribed face.

Often the user will want to work on one portion of the scene tree for a while before moving onto another portion of the tree. The commands that change the subtree of interest are described in a later section, but the concepts are important for describing the rest of the constructor commands.

The scene tree is rooted at *World*. The *World* definition has some special properties: it has no parent and it cannot be deleted. Otherwise, it is a simply another definition. There is one implied instance of the *World* in each view. When *Jessie* starts up, the current definition is set to be *World*, which means that all additions and deletions affect only the *World*. Several commands change the current definition to another definition in the scene. This changes the focus of the various construction and modifying commands to the new current definition. This current definition is commonly referred to as ***curDef*** in the remaining documentation.

One instance, if there is one, in the ***curDef*** is specially designated - a favorite child, so to speak. This is the current instance, or ***curInst***. All transformations and some modifying commands work only with the current instance. Both *Jessie* and *UNIGRAFIX* treat definitions as mere prototypes or templates of an object, whereas instances are the real, physical interpretation of the object in the world. The instance of a definition may be squeezed, stretched, translated, rotated or even deleted with no effect on the definition itself. Of course, any change to ***curInst*** ***will*** affect ***curDef*** because ***curDef*** is the parent of ***curInst***. Any change in ***curDef*** will be reflected in any instance that refers to this definition, just as in standard *UNIGRAFIX*.

The designated ***curDef*** and ***curInst*** save a lot of typing and repicking. All construction and transformation operators affect only ***curDef***, but some modifiers have side effects, e.g. **copy**, which will affect other specified definitions as well.

Jessie input is normally completely prefix, that is, the operation is specified before the parameters or the elements to be operated upon are supplied. However, the commands that use ***curDef*** and ***curInst*** have a postfix feel, since ***curDef*** and ***curInst*** are set implicitly before the commands are invoked. The original *Jessie* was completely postfix and this is one the few vestiges of that version. The tree traversal commands use and modify ***curInst*** and ***curDef*** with each movement through the tree.

The status line always displays the ***curDef*** and ***curInst***. If the ***curDef*** is a leaf node, there is no ***curInst***.

2.2. Advanced Constructors

Advanced Constructors
contour faceId (vertex-list); reverse contour faceId; reverse face faceId;
build def defId; copy def newDefId; copy def select newDefId; rename instance instanceId; rename def defId; replace def newDefId;
delete def defId; delete instance; delete vertex vertexId; delete face faceId; delete wire wireId; delete contour faceid; delete select;
flatten instance; join instance instId [defId];
clear def; clear all;

The **contour** command adds a contour to an existing face. The most useful contours are holes, but it is also acceptable to *Jessie* to add a co-planar face to an existing face. However, such contructs may not be properly interpreted by some of the other *UNIGRAFIX* programs. The vertex-list of a contour is formed exactly the same way as the vertex list in face statement. A useful option for drawing contours is *map* mode, which takes a face identifier as a parameter. *Map* will interpret each mouse-click as a ray from the viewer's eye to the map face, generating a point at the intersection of the face and the ray. This is useful for guaranteeing that each vertex lies on the selected face. Each click of the left mouse button will create a new vertex at the intersection point and interpolate the vertex identifier into the vertex list.

The *end_map* parameter in the vertex list informs *Jessie* to leave the *map* mode and go back to picking vertices. An example **contour** command could be:

contour faceId (map faceId
v g#0 3 4 5 v g#1 9 4 3 v g#2 9 4 5 end_map);

This format appears wordy, but since it is entirely generated through menu selections, some of which contribute several parts each to the final command, the actual issuance of the command is quick.

The direction of the contour determines whether it is a hole or a face. Reversing a contour through the **reverse** command is a convenient way to change the direction of a contour. Reversing a face reverses all the contours of a face.

The **build** command is a combination of several simpler *UNIGRAFIX* commands that are often used together. **Build** creates a new empty definition called *defId,* instantiates it in the current definition, and then sets the *curDef* to be the new definition.

A Unigrafix Scene Tree

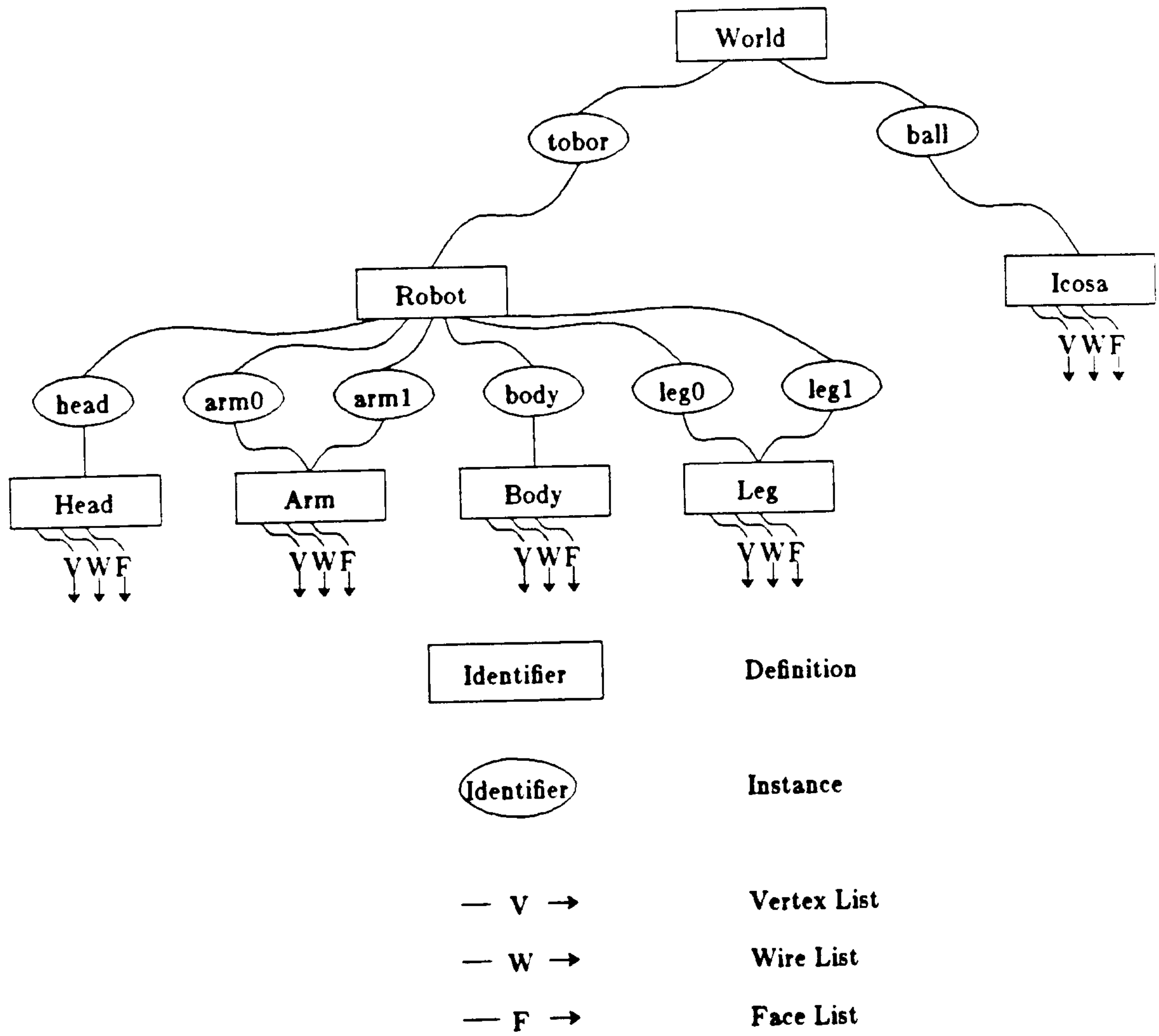


Figure 1.

A typical scene tree created by Jessie. The scene tree is rooted at the World definition. Each definition may have several instances, each of which points to another definition.

The current definition may be copied using the **copy** *def* command. The new definition is an exact replica of the current definition. All vertices, wires, faces, and instances in the new definition are unique, but it is important to remember that the definitions to which the duplicate instances refer are the same as before.

The **copy** *def select* command is useful for extracting only the members of the select set into a new definition. The select set mechanism is explained in the chapter **Picking and Selection**.

The **rename** command changes the internal identifier for a *UNIGRAFIX* object. The current instance and current definitions may be renamed with the *instance* and *def* parameters, respectively. *Jessie* 1.0 has no provision to rename vertices, wires, or faces.

The **replace** *def* comand changes the definition to which the current instance refers. This is occasionally useful for exchanging simple object for a more complicated object.

UNIGRAFIX is hierarchy-oriented, so *Jessie* provides several operators to help maintain and rearrange the scene tree.

The **delete** *type* operator will remove an object in *curDef*. **Delete** accepts each object type as the *type* parameter, including *contour* and *select*. **Delete** *contour* will delete the last added inside contour of a face, but not the first (outside) contour. Additional **delete** *contour* commands will delete the remaining inside contours, one at a time. Only **delete** *face* will remove the entire face, including the outside contour. The **delete** *select* command deletes all currently selected items in the select set inside the current definition.

The **delete** *def defId* command will delete the definition called *defId,* but only if it is not in use by any other instance. It is impossible to delete the world definition, but the same net effect may be obtained through the **clear** *all* command.

The **flatten** command "flattens" the current instance by copying the transformed elements of *curInst's* definition into the current definition. This does not affect the definition to which *curInst* refers. The *curInst* is deleted, and a new *curInst* is chosen among the remaining instances in *curDef*.

The last of the hierarchy modifying commands is **join. Join** asks the user to pick any instance in the scene. This picked instance is inserted into the current definition with the same relative transformation as it had to its former parent. The instance is then removed from its old parent definition, effectively moving the instance to the current definition. This is the only *Jessie* command that affects an object outside the current definition. *UNIGRAFIX* does not allow a recursive definition, so *Jessie* does not allow the picked instance to a paternal relative of the current definition.

The **clear** *def* command removes all vertices, wires, faces and instances from the current definition. It is equivalent to a series of delete commands. The parameter *all* will erase *every* existing definition, clearing the entire database. This is useful for restarting a session, or starting afresh on a new scene.

3. TREE TRAVERSAL, PICKING, AND SELECTION

Tree Traversal, Picking, Selection Commands
walk_down; walk_up; walk_right; walk_left;
pick instance iId; pick vertex vId; pick wire wId; pick face fId;
select {closure} { pick i {iId}+ }* { pick v {vId}+ }* { pick f {fId}+ }* { pick w {wId}+ }* ;

The **walk** commands change the *curInst* and *curDef* by moving around the scene tree. Using the **walk** commands may or may not be faster than simply picking the appropriate instance with **pick** *instance*. It is sometimes more convenient, especially if the user is very familiar with the scene tree.

The **walk_down** command changes the *curDef* to the one referenced by *curInst*. The first instance in this definition, if any exist, becomes the new *curInst*. If the **walk_down** command is issued in a definition with no instances, and therefore no *curInst, Jessie* displays a mild error message.

The **walk_up** command changes the current definition and current instance by moving one level closer to the root, i.e. *World*. The new *curInst* is reset the previous *curInst* before the last *walk_down* command. The new *curDef* is the definition that contains the new *curInst*. Since instances in many definitions may refer to a particular definition, *Jessie* keeps track of the specific one that was used to get to current definition. Several *Jessie* commands, such as **pick** *instance* and **walk_down,** set the path through the scene tree.

Walk_right and **walk_left** set the new current instance to be the right or left sibling, respectively, of the current instance. Practically, right and left have no physical meaning, except that the right sibling is alway in the opposite direction as the left sibling. The sibling instance list is circular within *curDef,* so a walk in the wrong direction will eventually end up with the desired instance.

Each command knows what type of objects it needs in order to complete a syntactically correct command. The prompt line and menus inform the user of the choices for next valid steps. Often the command needs an identifier from the database of some *UNIGRAFIX* object. Although there are about a dozen different types of objects the pick handler can return, there are a few common denominators: the left-hand mouse button picks, and each pick is always at a vertex or at a corner of a bounding box.

The *vertex-list* pick is special. The middle button may be used to *extract* the coordinates of any point in the scene, including the boundary of a bounding box. A new vertex is created at that world location, but it bears no topological relationship to the picked point. It just occupies the same position in space.

The **pick** command takes the goal object type as a parameter. This type may be any of the standard objects: *vertex, wire, face* or *instance*. Definitions may not be picked, as they never appear on the display. However, the instantation of a particular definition may be selected.

Notice that each of the **pick** commands also takes an identifier as a parameter. The user never types this; it is supplied by *Jessie* after the pick is successfully completed. If for some reason the pick fails, *Jessie* will keep asking the user to re-pick the object until the pick succeeds or the user decides to **abort.** *Jessie* will return the full path name to the instance in a **pick** *instance* command. The **pick** *vertex,* **pick** *wire,* and **pick** *face* commands are local to the current definition. Some commands allow inter-definition picks, but in these cases the pick will return a value, such as the coordinates of a vertex rather than a direct reference to the vertex.

Because names of objects in definitions are local, the name of a *UNIGRAFIX* object does not uniquely specify it. It is possible to uniquely specify an object by giving its full path name, but this is often extremely cumbersome and certainly is not useful within an interactive editor. *Jessie*, like most 3D hierarchical editors, uses a picking device to bypass the hierarchical naming problem and to allow the user to directly point to the desired object.

Sometimes even this is not enough and a 3D view may be ambiguous. This is easily solved by shifting the view slightly, an advantage which a 3D editor has over a 2D editor in a similar situation. The early versions of *Jessie* had a sophisticated picking algorithm for distinguishing among these ambiguous picks. The simpler system turned out to be satisfactory.

Jessie finesses the ambiguous pick problem by allowing picks from any graphics subwindow within the same command. At least one view should contain a satisfactory eyepoint on the scene.

Occasionally the user will want to directly type in the name of an object. The path from the root to the destination must be fully specified. This format resembles the way UNIX specifies directory paths, e.g. /rootinst/b/c/destobj. Each slash and string represents one instance of the *UNIGRAFIX* scene tree. The full specification of the path will uniquely specify one object in the destination definition. This representation is cumbersome, but complete.

Picking, by itself, does nothing more than return the name of the picked object. Often the user wants to do something with a picked object, so *Jessie* uses the **pick** command to help build a ***select set***.

The select set is a group of objects within a definition that are temporarily linked together so that a *Jessie* command may operate on all of them in parallel. Any set of vertices, wires, faces and instances in a definition may be grouped into a select set. Most commands that operate on *UNIGRAFIX* object can also take the select set as a parameter. Select sets remain with the definition (they are implemented as boolean flags on each item) so it is possible to have several simultaneous select sets in several different definitions.

In effect, the select set may be used as a temporary instance, without any permanent changes of the data structure. The selected objects may be deleted, copied, or transformed. Using select sets for transformations is extremely useful and often saves tedious picking and retransformations.

Selecting a face or a wire implicitly selects all its vertices. Since *UNIGRAFIX* is a vertex-oriented boundary representation, selecting a face would be somewhat meaningless if its vertices were not also selected.

The **select** command is primitive in *Jessie* 1.0. Each **select** command may select of any combination of objects in the current definition, but the select set in the definition is cleared for each new **select** command. This means that the entire select set must be re-entered if the user wants to add or delete any objects in the set. Selected objects must be in the current definition. Despite these drawbacks, the mechanism is useful. **Select** has a *closure* option which computes the transitive closure of a selection. With *closure* it is easy to point to one vertex of a group of connected faces and implicitly select the entire group. Unconnected objects (in the *UNIGRAFIX* sense) remain unselected. The *closure* parameter has no effect on the selection of instances since they are never connected to any other object.

4. INCREMENTAL TRANSFORMATIONS

The transformation operators do not change the scene tree topology, but rather the transformations between portions of the scene tree. Rotations, translations, and scaling are examples of commands, that transform some portion of the scene tree. The viewing transformations change the relationship between the *Jessie* scene and the viewer's eye.

The **use_dials** command attaches the incremental transforming operations to the dial area on the command panel. This allows easy interactive modification of the selected objects without having to know the exact transformations. These dials may be hooked up to any view or object in the scene tree and may be interpreted as rotations, translations or scaling around various important axes in the scene.

The Sun does not have any valuator devices, so the mouse and virtual dials replace physical dials. The dial buttons on the dial area are valuators. A left mouse-button click on the left side of the dial represents a "small positive" value and a left mouse-button click on the right side of the dial is a "large positive" value. The "largeness" or "smallness" of the value depends on the coarseness of the dial mode. Similarly, the right mouse-button returns a "negative" value of the same magnitude. A left-click followed by a right-click will cancel each other out, leaving a net transformation of zero units.

The coarseness of the dials may be set to *autoscale* so that the value returned is a small fraction of the bounding box of the current instance. The semantics of the dial value should be interpreted only as a convenient analog number with no particular significance. If the analog dials are not suitable, the user may enter an exact numerical distance or angle for a transformation.

There is a wide variety of transformation options available from the keyboard and dials. Often the user will want to fiddle with the dials freely, moving the selected object in space until it reaches the desired position.

Each definition has an implicit coordinate system. This is the coordinate system that we use to build the object, e.g. the vertices of a cube may lie at (+-1, +-1, +-1). The most common type of transformation is to rotate or translate an instance in this coordinate system, where the x,y,z axis are interpreted in the natural manner. Such operations are called *instance* transformations and correspond to transforming the instance around its own axes. The *parent* transformation, on the other hand, transforms the instance and its axes with respect to the coordinate frame of the parent definition.

While designing an object, the user may want to shift the axes of the definition to a more convenient position. Having the axes in a good position simplifies future transformations of the instance with an *instance* transformation. The *axes* transformation changes the frame of reference, i.e., the position of the axes within the current instance. If the **show** *xyz true* option is set, the user sees the axes of the current instance move or rotate. These algorithms are described in more detail in the *The Jessie Design Manual* [4].

After the type of transformation (*instance,parent,* or *axes)* is specified, *Jessie* attaches the dials to the specified object or objects. The user has the option of restricting the transformation further or continuing with a *general* transform. The *general* transform allows free movement around and along the x,y,z axes of the specified coordinate frame, as well as symmetrical scaling with respect to the origin of the frame.

Incremental Transforms
use_dials instance general transform-list;
use_dials axes general transform-list;
use_dials parent general transform-list;
use_dials vertex general { vertexId transform-list }*;
use_dials select general transform-list;
use_dials view general transform-list;
use_dials instance vec_select point point transform-list;
use_dials axes vec_select transform-list;
use_dials vertex vec_select point point { vertexId transform-list}*;

The user may type in the transform list directly, but it is much easier to use the dials as described above to generate the sequence of transforms. As each transform is parsed, *Jessie* will update the display by moving the appropriate object in real time. Although *Jessie* update speed is not blazingly fast, the feedback is acceptable.

The transformation parameters *instance, axes,* and *parent* refer to the current instance. *Jessie* does not have any concept of a current vertex, so the **use_dials** *vertex* command operates on picked vertices within the current definition. As each vertex is picked, the entire set of accumulated transformations from the start of the current **use_dials** command is applied to the new vertex. This makes it easy to transform a series of vertices with the same set of transformations.

The **use_dials** *select* command transforms every selected object within the definition with the same series of transformations. This is the preferred way to transform a face, such that it remains planar, because a series of **use_dials** *vertex* commands could create non-planar faces. *Jessie* doesn't care about non-planar faces because it deals almost exclusively in wire-frames, but the algorithms in *ugdisp* or *ugplot* rely on planar faces. The **use_dials** *select* is especially useful for moving a group of instances simultaneously while keeping the relationship between those instances constant.

Occasionally the user begins to transform the object, but, perhaps through indecision or carelessness, regrets ever touching the object. At any time the user may *abort* the transform and return the object to its original state. If the user wants to keep the transform, hitting *accept* will end the **use_dials** command. The user may even change his mind after *accept*ing the transform (see **undo**). The ability to abort a transformation in mid-stream makes transforming objects painless and easy.

Sometimes it is inconvenient to transform objects in terms of their instance or parent coordinate system, so *Jessie* also supports rotation and translation around arbitrary vectors in the scene. With the *vec_select* option, the user may pick any two points in the scene, such as an edge of a face or corners of a bounding box, and then rotate around or translate along that vector.

The incremental transform that operates on the scene as a whole, **use_dials** *view general,* is so useful that it has been given a special place on the main menu (abbreviated to just **view xform**). This operation transforms the view, allowing the user to move around inside the scene. All the transformation options that transform an instance, e.g. translation along a vector, could also be used to transform the view. However, the most common method of transforming the view is still the **use_dials** *view general* command.

5. ABSOLUTE TRANSFORMATIONS

All the transformation operators described so far are incremental. Each transformation is applied on top of the previous transformations to produce a new transformation. The incremental transformation is intuitively obvious for interactive tweaking of the scene, but often the user has a clear idea of exactly where he wants the object to be. Moving the latter to its final position via the incremental transforms would be tedious, error-prone, and unexact, even with the use of the various coordinate frames. To ease this task, *Jessie* supports a set of ***absolute transforms***.

Absolute Transforms
move instance pt_select pt0 pt1;
move instance vec_select vec0 vec1;
move instance plane_select plane0 plane1;
move select pt_select pt0 pt1;
move select vec_select vec0 vec1;
move select plane_select plane0 plane1;
move instance norm transform;
move instance normrot transform;
move instance align transform;
move instance abut x transform;
move instance abut y transform;
move instance abut z transform;
move instance abut xyz transform;

The syntax and semantics for absolute transforms are similar to that of incremental transforms. All transformed instances and vertices must be in the current definition. Except **move** *select,* all the absolute transforms change only the current instance. **Move** *select,* like **use_dials** *select* changes the transform of all the selected objects inside the current definition.

The *pt_select, vec_select* and *plane_select* parameters are extremely useful for aligning objects exactly. These alignment operators are a major feature of another 3-D scene composition system, called*SCOT* [5].

Pt_select asks the user for two *points* in the scene. These points may be vertices, or points on a bounding box. *Jessie* calculates a transform that will map the first point onto the second point with a simple translation. This translation is applied to the current instance or to the select set for the commands **move** *instance* and **move** *select,* respectively; position angles remain unchanged.

Vec_select asks the user for two *vectors* in the scene. A vector is defined by any two points, as described above. The vector is extracted by subtracting the first picked point (the tail), from the second point (the head). Vector alignment starts by moving the tail of the first vector onto the tail of the second vector, like *pt_select,* then rotating the altered first vector onto the second vector until the vectors are aligned. The combined transform that describes this alignment process is applied to the current instance or the select set.

Plane_select asks the user for two *planes* in the scene. A plane is defined by any three points. For plane alignment, the lines defined by the first *two* points of each triple are aligned, as described in vector alignment, then the altered first plane is rotated about this line until the planes coincide.

The order of picked points for *pt_select, vec_select, plane_select* is critical. Different picking order will have different results. This is intentional and useful. In summary, for all three cases, the first point (of the first vector or plane) is translated to the first point (of the second vector or plane). The first vector (of the first plane) is rotated into the first vector (of the second plane). Finally, the new first plane is rotated into the second plane.

Most of the absolute transforms accept a parameter *transform*. Usually this parameter is extracted from an existing transformation somewhere in the scene tree. For example, if the user wants to *align* instance A with with instance B anywhere in the scene tree, he could select **move** *instance align,* then pick instance B in a view. The user does not care what the transform was, but simply wants to set align A with B.

The pick handler extracts the transform from the picked instance and substitutes the transform for the picked instance, without the user ever knowing what the transform was.

The *norm* parameter sets the total transforms with respect to the *World,* between the current instance and the picked instance to be the same. Afterwards, the current instance will have the same rotation, translation and scale relative to the *World* as the picked object.

The *normrot* parameter sets the total *rotational* transforms with respect to the *World,* between the current instance and the picked instance to be the same, but leaves the translation component intact. The selected object will have the same rotation and scale relative to the world as the picked object, but its origin will remain fixed.

The *abut* xyz parameter leaves *curInst's* rotation intact, but translates the coordinate frame such that it directly coincides with the picked instance. *Abut* also accepts just *x,y* or *z,* which translates the frame such that *curInst* and the picked instance's frame coincide along the specified axis.

The *align* transform is particularly useful. *Align* looks at the current instance's coordinate frame and the picked instance's coordinate frame and aligns them by matching the closest axes of the current instance to the picked instance. This algorithm chooses the smallest rotational transform that will align any combination of the axis pairs. This operation is useful for making the coordinate systems of two objects orthogonal to each other.

The command sequence *abut* x, *abut* y, *abut* z, has the same effect as just *abut xyz*. A *normrot* followed by an *abut xyz* would have the same effect as one *norm* command. Except for *align,* all these transforms ignore *curInst's* current transformation.

The normalization and abutting operations are described in depth in [6].

6. DRAWING AND VIEWING COMMANDS

Drawing and Viewing Commands
draw {clear | grid | contour | select}* ; disp {clear | sa | in | ho | ab}*;
eyepoint x y z; eyedirect x y z; viewrotangle r; viewinfo; use_dials view general transform-list; perspective { true | false }; pick view viewIndex;
show open { true | false }; show bbox { true | false }; show xyz { true | false };

Jessie has a small, but useful set of operators to help view the scene as it is being constructed.

Jessie 1.0 does not place a strong emphasis on intelligent screen updating. Where possible, *Jessie* attempts to redraw the screen to reflect the current state of the world, but once in a while the user would like to force a redraw. The **Draw** command redraws the scene in all current views. The **Draw** command accepts four parameters, *grid, contour, select* and *clear*. The *Grid* parameter draws a grid on the screen which the user may find helpful for alignment. The *Contour* parameter draws small arrow-like lines on all edges in the scene so the user can visualize the direction that a contours faces and easily distinguish holes from faces. The contour lines lie on the plane of the face at a forty-five degree angle to the edge. The contour line intersects the edge two-thirds of the way along the edge in the direction it points, so the contour lines on adjacent faces are easily distinguished. Each contour line is scaled so it is relatively "small" compared to the edge on which it lies. The *select* parameter highlights the current select sets. All of the drawing parameters stay on until the next **draw** *clear* command.

Jessie has a simple, yet workable interface to *ugdisp* [7], using an intermediate ascii file. The **disp** command creates a *ugdisp*-rendered version of the current scene in the current view. Unfortunately, *ugdisp* automatically rescales the picture, although the view rotation and eyepoint will remain the same. This will eventually be fixed as *ugdisp* is enhanced witha the new *XFORM* standard [8]. The **Disp** command accepts all the standard *ugdisp* options. Like **draw,** all **disp** options stay in effect until the next **disp** *clear*.

The **eyepoint,** *eyedirect,* and *viewrotangle* need little explanation. They correspond to the standard *UNIGRAFIX* description. Like *UNIGRAFIX*, **eyepoint** implies perspective viewing which may be useful as a depth cue. **Viewdirect** turns off the perspective viewing as well as setting a new view direction. Perspective viewing may be explicitly set (or unset) with the **perspective** command.

Jessie 1.0 supports up to four simultaneous views on the same scene. The **pick** *view* command, followed by a pick anywhere within the four views sets the view for future viewing commands. This view is marked as the "current" view by a small pair of eyeglasses in its upper-left hand corner. Working with multiple views is sometimes necessary, and often helpful in visualizing complex scenes. The main view defaults to an eye direction of (1,0,0), which corresponds to looking along the X axis. The three smaller, remaining views default to the following view directions: (from the top of the tool downward):

(1,0,0), (0,1,0), (0,0,1). These correspond to looking along the X,Y,Z axes, respectively.

The most useful of the viewing commands is the **use_dials** *view general* command, which is conveniently abbreviated to **view xform** on the main menu. This command attaches the dials (see incremental transforms) and allows to manipulate the viewing transform that instantiates the *World*. The interface is simple and intuitive and fits nicely into the way *Jessie* transforms other parts of the *UNIGRAFIX* scene tree.

The **viewinfo** commands prints a table of current viewing information, which includes the current settings of the eyepoint, eyedirect, perspective and view rotation.

The **show** options allow the user to control the level of information displayed on the screen. These options are applied to the scene tree itself, and are independent of the current view.

The **show** *open true* command will open the current instance's bounding box so that the insides may be viewed. Unlike **walk_down,** the current definition and current instance *do not change*. **Show** *open false* has the opposite effect, that is, it hides the contents of the current instance and displays only its bounding box. This command is useful for suppressing scene details that would slow down redrawing, or make picking or visualization difficult.

The **show** *bbox* command shows or hides the bounding box of the current definition. This command is useful on occasion, but can cause confusion if used improperly. Note that the *bbox* and *open* flag are independent.

The last of the **show** options sets or unsets the display of a *gnomon* or labeled xyz axis for the current instance. This option is useful for choosing axes for the various transform commands. *Jessie* will scale the gnomon so it fits the scale of the bounding box. The **show** *axes* command often prefaces the **use_dials** *axes* command, so the user can see how the axes are being transformed.

7. MISCELLANEOUS

Miscellaneous
include pathname;
source pathname;
write pathname;
set identifier { = value };
unset identifier;
quit;
abort;
undo;

The **include** command reads a *UNIGRAFIX* source file, just like the standard *UNIGRAFIX* command. The **source** command reads a previously generated *Jessie* script. Since *UNIGRAFIX* commands are a subset of *Jessie* commands, any file that can be **included** may alternatively be **sourced**, but not the other way around. If the included file has any error, however small, all internal changes generated by this include file are undone. In effect, the database returns to the state which existed before the **include** command was issued. The **source** command ignores errors and continues reading until the end of the file. There is also another semantic distinction between the two commands described in the section on the **undo** command.

The **write** command writes the entire scene tree to a file in *UNIGRAFIX* format so that it may be included in another session or sent to another *UNIGRAFIX* program.

The **set** command sets the particular environment variable by entering it in the symbol table along with an optional value. The system maintainer uses these special variables for debugging sections of *Jessie*. The **unset** command removes the variable from the symbol table.

The **quit** command ends the *Jessie* session after first asking for confirmation. Obviously, this should be used with care.

Abort will stop any pending command. Every menu contains the **abort** command in the lower left corner. The current command is flushed from the input buffer, and any intermediate effects of the command are undone.

Jessie supports the elusive and desirable **undo** command. The effects of the last database command, which includes additions, deletions and transformations, may be quickly undone by hitting **undo.** The undo trail goes all the way back to the begging of the session, and by repeatedly hitting **undo,** the user can erase any number of previous changes in the scene. The availability of the **undo** command makes it painless and easy to experiment with unfamiliar commands.

Jessie can even undo an entire **include** command. After including a file, a touch of the **undo** button will undo the effects of the include. As far as undo is concerned, the **include** command, plus all the commands read from the file are one *group* of commands to be undone together. The **source** command is simply a set of commands read from a different input stream; they are therefore undone individually.

The only caveat is that *Jessie* only undoes *changes to the database*. This means that simple commands, like moving around the tree, or setting a show option is not undoable.

Jessie 1.0 does not contain a redo command. It would be straightforward to implement and is discussed in the *The Jessie Design Manual.*

8. THE ENVIRONMENT

A file in the user's home directory, called .jessierc contains a set of commands to be issued at bootup time. The default file includes the gnomon file and sets some environment variables. The user may customize this file to include other commonly used definitions, scenes, etc.

Jessie maintains a script of all commands issued in the last session. This script is called *.jessielog*. It is placed in the user's home directory and overwritten at the beginning of each new session. The user may save old scripts, rename them, then **source** them to restart a session if the tool faults for any reason.

Jessie has several environment variables that are useful for maintaining the system. Each of the these variables may be set during a session with the **set** command, or *Jessie* may be started with the environment variable already set by using the appropriate command flag.

Environment Variables		
Name	Startup Flag	Function
ErrLog	-e	If set, *Jessie* will keep a log of all error messages in a file called *.jessieerror* in the user's home directory.
Flashy	-f	While in a script, *Jessie* will display every command on the command line and incrementally display new contours.
LexDebug	-l	Trace scanner by printing tokens as they are accepted.
YYDebug	-p	Trace grammar by printing out rules as they are reduced.
IoDebug	-o	Trace each character as it is recieved by the scanner. Trace characters placed on the *unput* stack.
UndoDebug	-u	Trace strings as they are placed on the undo stack.
PathDebug	-t	Trace calculation of the path through the scene tree. Useful for debugging hierarchical commands.
SymDebug	-s	Enable symbol table debugging

References

1. Sun Microsystems, Programmer's Reference Manual for the Sun Window System, Sun Microsystems, Inc., Mountain View, CA, 1982.

2. Mark Segal, Carlo H. Séquin, and Paul R. Wensley, *UNIGRAFIX* 2.0 User's Manual and Tutorial, CS-Technical Report 83/161,, UCB/CSD, December 1983.

3. John Ousterhout, Gordon Hamachi, Robert Mayo, and Walter Scott, *Magic VLSI Design System,* CS-Technical Report, UCB/CSD, 1984.

4. H.B. Siegel, *Jessie: An Interactive Editor For UNIGRAFIX / The Design Guide* , CS-Technical Report, UCB/CSD, December 1985.

5. Steve Upstill, Tony DeRose, and John Gross, *SCOT: Scene Composition Tool,* CS-Technical Report, UCB/CSD, 1983.

6. Eric Allan Bier and K. R. Sloan, Jr., *Pointing and Placing with Homogeneous Transforms,* Xerox Palo Alto Research Center, July 24, 1984.

7. Nachshon Gal, *Hidden Feature Removal and Display of Intersecting Objects in UNIGRAFIX,* CS-Technical Report, UCB/CSD, 1985.

8. Mark Segal, *XFORM Geometry Package,* In Preparation, CS-Technical Report, UCB/CSD, 1986.