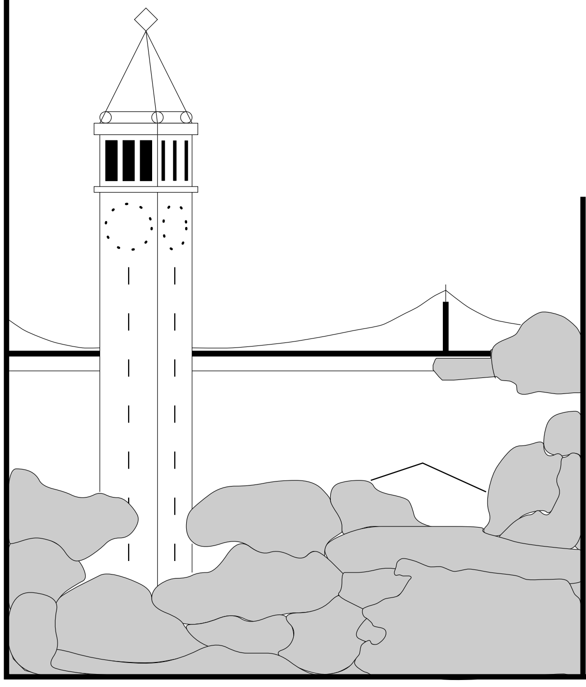
Ugmovie — Interactive UNIGRAFIX Movie Previewing

Gregory S. Couch



Report No. UCB/CSD 94/831

May 1994

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

Ugmovie — Interactive UNIGRAFIX Movie Previewing

Gregory S. Couch

Computer Science Division
University of California, Berkeley

ABSTRACT

This report describes the interface and implementation of the **ugmovie** program. Ugmovie displays views of UNIGRAFIX scene descriptions [Couch94] that have been optionally augmented with the bump [Oakland87] animation extensions. Ugmovie also provides interactive controls to manipulate time, edit scene variables, and modify viewing and rendering parameters. Ugmovie uses OSF's MOTIF [OSF91a] for its user interface and SGI's GL [GLX91] for the 3D rendering.

1. Introduction

In the fall of 1987, Andy Oakland wrote an ''animation-generating preprocessor'' for UNIGRAFIX called **bump**, the Berkeley UNIGRAFIX Movie Package [Oakland87]. The salient feature of the original bump is that it knew how to minimize the work for successive frames of an animation. Bump was successfully used to produce a short videotape and was mostly forgotten after Andy left. In the fall of 1992, Professor Séquin needed an interactive tool to animate UNIGRAFIX scenes on a Silicon Graphics (SGI) Iris workstation for his geometric modeling class. The bump animation extensions were resurrected, enhanced, and incorporated into an interactive program named **ugmovie**. Ugmovie provides interactive controls for previewing animations by manipulating time or scene variables and by providing controls for modifying viewing and rendering parameters.

The following sections review the bump animation extensions, describe the user interface, and describe the implementation.

2. Bump Extensions to the UNIGRAFIX Language

Bump embeds animation instructions in the UNIGRAFIX scene description through the extension mechanism called escape statements. Ugmovie understands the exact same set of escape statements. The following is a synopsis of the original bump extensions along with the new **vardef** statement. In total, there are six statements added to the UNIGRAFIX language: **move**, **mdef**, **mapply**, **mexecute**, **meval**, and **vardef**. The syntax of the statements is shown in a variant of BNF (Backus-Naur Form) where literals are in boldface, non-literals are in italics, square brackets ([and]) denote optional parts, curly brackets ({ and }) denote one or more repetitions, and the vertical bar (|) denotes either or.

2.1. mdef

The motion definition statement, **mdef**, builds a set of time-dependent motions that can be referred to by name. The syntax is:

> **(** **mdef** *new-motion-name local-start-time local-duration*
> { **(** *motion-type transformation* { **(** *time param(s)* **)** } **)**
> | **(** *motion-name start-time duration* [*repetitions*] **)** }
> **)**

There are four motion types: **step**, **linear**, **cspline** (cardinal spline), and **bspline** (B-spline). The *transformation* is any of the UNIGRAFIX transformations that takes at least one parameter (*i.e.,* all except **–mx**, **–my**, **–mz**, and **–ma**). The *time param(s)* tuples of time and transformation parameter values, are the control points for the motion. The **step** motion stays at a particular value until the next control point occurs. The **linear** motion linearly interpolates between adjacent control points. The **cspline** and **bspline** motions do cubic interpolation using groups of four control points.

Motions may refer to other motions. The *start-time* and *duration* of a nested motion (its *local-start-time* and *local-duration*) are scaled to match. An optional number of *repetitions* may be given to repeat the motion.

A motion results in an affine transformation matrix that is the concatenation of the UNIGRAFIX transformations in the order listed. The exception to the above are two special *motion-name*s, **appear** and **disappear**, which cause an instance to appear and disappear, respectively. By default, instance's are visible, so an initial **appear** is unnecessary.

2.2. mapply

(mapply *instance-name*
  { **(** *motion-name start-time duration* [*repetitions*] **)** }
)

Once you have a motion defined, the **mapply** statement applys the motion to an instance. The semantics of each motion component of the **mapply** statement match that of nested motions in the **mdef** statement.

2.3. move

(move *instance-name start-time duration*
  { **(** *motion-type transformation* { **(** *time param(s)* **)** } **)**
  | (*motion-name start-time duration* [*repetitions*] **)** }
)

The **move** statement is a shorthand for a combination **mdef** and **mapply** where the motion definition is not reused.

2.4. vardef

(vardef *new-variable-name value* [*min-value* [*max-value*]] **)**

The **vardef** statement defines a global variable that can be referred to in expressions (see below). Note that the *value* may be an expression, if it is enclosed within parenthesis. The optional *min-value* and *max-value* can only be single numbers. They are unused except as a hint for setting the initial slider boundaries in ugmovie.

2.5. meval and mexecute

(meval
  text to be fed back into UNIGRAFIX language parser after evaluating
  the expressions between pairs of dollar signs ($).
)

(mexecute
  text to be used as a UNIX command that generates UNIGRAFIX language
  after evaluating the expressions between pairs of dollar signs ($).
)

The animator should be aware that these statements cost more computationally than the other animation statements. The expense comes from the (resulting) text having to be reparsed for each frame rather than only once when the scene is read from a file, and from needing to restore the enclosing definition after each frame because of the modifications made by the evaluated text. Due to the latter expense, there is one implementation restriction on the use of **meval**'s and **mexecute**'s — they must occur within a **def**/**end** pair. See below for the expression syntax.

2.6. Bump Expressions

Bump expressions are computed in floating point and follow the C language syntax. Operators are: **+**, **-**, *****, **/**, **%**, **? :**, **&&**, **||**, **==**, **!=**, **>**, **>=**, **<=**, **<**, and **( )**. There is one predefined variable, **t**, that corresponds to the current frame's time. There are two predefined constants: **pi** and **e**. The available functions are: **sqrt**, **exp**, **log**, **pow**, **floor**, **ceil**, **sin**, **cos**, **tan**, **asin**, **acos**, **atan**, **atan2**, **sind**, **cosd**, **tand**, **dasin**, **dacos**, **datan**, **datan2**, **ramp**, and **sawtooth**. The trigonometric functions that end or start with a **d** take their arguments in degrees or return their results in degrees, respectively. The **ramp** function is a periodic function that returns the periodically increasing fractional part of a positive number. The **sawtooth** function is a symmetric periodic function that slopes from zero to one and back to zero within each integral interval.

2.7. Elevator Example

Below are four versions of a naive elevator consisting of the floor of the elevator and one wall of the elevator shaft (see Figure 1). Topological consistency is maintained in two of them by having the floor explicitly intersect the wall. The height variable is visible to ugmovie and can be modified interactively. The first version uses the straight forward technique of putting everything inside one **meval**. The second version minimizes the amount of text within the **meval**, and as a result, is faster and only slightly more complicated with an additional definition. The third version is much faster because it neither uses any variables nor any **meval**'s, but it loses topological consistency. Of course, once we give up topological connectivity, we may as well go all the way and have a single fixed polygon replacing the wall. In the forth version, the vertical position of the elevator floor is the only thing that needs to be evaluated, resulting in the fastest implementation.

```
(vardef height 5 0 10)
c c_wall1 0.5 0;
c c_wall2 0.5 30;
c c_floor 0.5 180;
def obj;
(meval
        v m0 0 $ height $ 1;
        v m1 0 $ height $ 0;
        v m2 1 $ height $ 0;
        v m3 1 $ height $ 1;
        f floor (m0 m1 m2 m3) c_floor;
        v b0 0 0 0;
        v b1 0 0 1;
        f bot (m1 m0 b1 b0) c_wall1;
        v t0 0 10 1;
        v t1 0 10 0;
        f top (m0 m1 t1 t0) c_wall2;
)
end;
i (obj);
```

The slower way

```
(vardef height 4 0 10)
c c_wall1 0.5 0;
c c_wall2 0.5 30;
c c_floor 0.5 180;
def floor;
        v m0 0 0 1;
        v m1 0 0 0;
        v m2 1 0 0;
        v m3 1 0 1;
        f floor (m0 m1 m2 m3) c_floor;
end;
def obj;
        v b0 0 0 0;
        v b1 0 0 1;
        v t0 0 10 1;
        v t1 0 10 0;
(meval
        i f (floor -ty $ height $);
        f bot (f.m1 f.m0 b1 b0) c_wall1;
        f top (f.m0 f.m1 t1 t0) c_wall2;
)
end;
i (obj);
```

The faster way

```
c c_wall1 0.5 0;
c c_wall2 0.5 30;
c c_floor 0.5 180;
def panel;
        v m0 0 0 1;
        v m1 0 0 0;
        v m2 1 0 0;
        v m3 1 0 1;
        f (m0 m1 m2 m3);
end;
i floor (panel c_floor);
i bot (panel c_wall1 -rz 90);
i top (panel c_wall2 -rz 90);
(move floor 0 100
        (linear -ty (0 0) (100 10)))
(move bot 0 100
        (linear -sy (0 0) (100 10)))
(move top 0 100
        (linear -sy (0 10) (100 0))
        (linear -ty (0 0) (100 10)))
```

An even faster way

```
c c_wall 0.5 0;
c c_floor 0.5 180;
def panel;
        v m0 0 0 1;
        v m1 0 0 0;
        v m2 1 0 0;
        v m3 1 0 1;
        f (m0 m1 m2 m3);
end;
i floor (panel c_floor);
i wall (panel c_wall -rz 90 -sy 100);
(move floor 0 100
        (linear -ty (0 0) (100 10)))
```

The fastest way

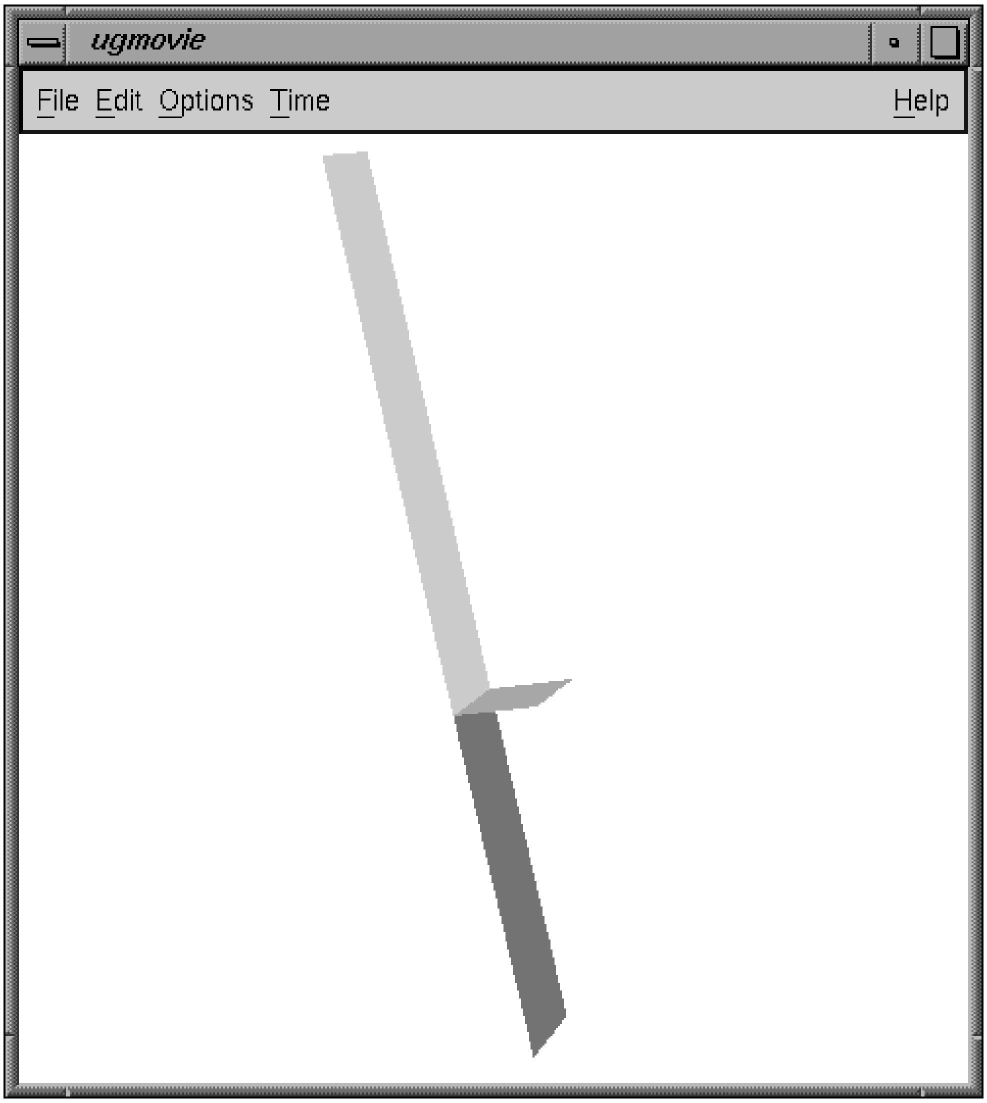


Figure 1 — Main Window with elevator example

3. Interface

Ugmovie owes its physical appearance to the MOTIF user interface toolkit [OSF91a]. The initial window that appears has all of the characteristics of a normal MOTIF application, the MOTIF keyboard shortcuts for menus work, *etc.*. The MOTIF interface was selected to leverage a user's experience with other applications on the SGI workstations (and because SGI plans to remove windowing support from its graphics library in the near future).

3.1. Command Line Arguments

Since ugmovie was written using the MOTIF toolkit that is based on the X Intrinsics toolkit, Xt [McCormack91], it understands all standard X application command line options (*e.g.*, −display, −geometry, −iconic, −synchronous, *etc*. — see the ''OPTIONS'' section of a ''X'' manual page or your favorite X book), in addition to its own options.

For the X savvy user, the **−ResFile** *resource-file* option lets one substitute a different application default resource file. **−rf** is a synonym for **−ResFile**. This command line argument is from the WCL toolkit [Smyth92] (see the implementation section for details).

Ugmovie provides three options. The –**movie** *scene-file* option specifies the initial scene to display. The –**mintime** *timeval* specifies the initial minimum frame number. The –**maxtime** *timeval* specifies the initial maximum frame number.

3.2. Main Window

The main window (see Figure 1) is broken up into two areas, the menu bar and the scene display area.

3.2.1. Menu Bar

The menu bar holds the File, Edit, Options, Time, and Help menus. The underlined characters in the menu bar are clues for accessing menus with the keyboard. When typed simultaneously with the Alternate key (ALT), the underlined characters in the menu bar invoke the associated menu. Once the menu is posted, a menu entry is invoked by typing the underlined character in the menu entry. To cancel a menu, press the Escape key (ESC). (The window manager menu can be accessed by typing Alternate-Escape.)

The File menu has Open, Save As, Save Frame As, Close, Print, and Exit entries. There should be no surprises about what each entry does. The Save As entry is disabled because the bump library currently removes variable and motion definitions from the scene's data structure — this is a bug and when the bump library is fixed, this option will added back in. The Print entry is disabled because to implement it correctly would be difficult and because the user can work around the problems more easily than the program. The problems are: 1) windows can be obscured and there is no off-screen rendering using the graphics hardware, and 2) SGI workstations support two different print spooling systems. The user can work around this by making the window completely visible, saving the window contents with an interactive screen saver (*e.g.* /usr/sbin/snapshot or /usr/sbin/imgsnap), converting it to PostScript (*e.g.* /usr/sbin/tops), and printing it using whichever print spooling system is appropriate.

The Edit menu is currently just a place holder for cut and paste functionality. Cut and paste does work for text, but not for three dimensional objects.

The Options menu provides hooks for items that don't fit the other menus. The Reset View entry causes the viewing rotation and translation transformations to be reset to the identity matrix and the zero vector respectively. The Viewing Parameters entry brings up a panel for controlling how an object is rendered. The Variables entry brings up the Variables Panel (see below).

The Time menu lets one control animations. The Freeze entry stops any animation. The Single Step entry advances time to the next frame. The Sequence entry sequences time from its minimum to maximum frame. The Cycle entry cycles time from its minimum to maximum frame for an animation loop. The frame time can also be changed interactively by manipulating the **t** variable as described in the Variables Panel section below. The Variable Editing Panel (also described below) can change the minimum and maximum time values (*i.e.* the values of **t**).

The Help menu is also just a place holder. We do not understand yet how the Motif help support works and consequently have not implemented it.

3.2.2. Scene Display Area

The current scene comes from either the file given on the command line with the –**movie** option, or from the Open File Panel. The scene is displayed below the menu bar in the main window. To be compatible with other UNIGRAFIX display programs on SGI workstations, the middle mouse button controls view manipulation. Moving the middle mouse button rotates the scene using a virtual sphere [Chen88]. Pressing the shift key with the middle mouse button translates the scene in z. Pressing the control key with the middle mouse button translates the scene in x and y.

3.2.3. Variables and Variable Editing Panels

The Variables Panel allows one to change variables that have been declared with the **vardef** statement described in the language extensions section above (§ 2.4). In addition to the declared variables, the built-in time variable, **t**, is presented. Each variable has a slider associated with it that can be set or dragged interactively. Every time a variable is changed, the value is propagated back to the scene and the scene is

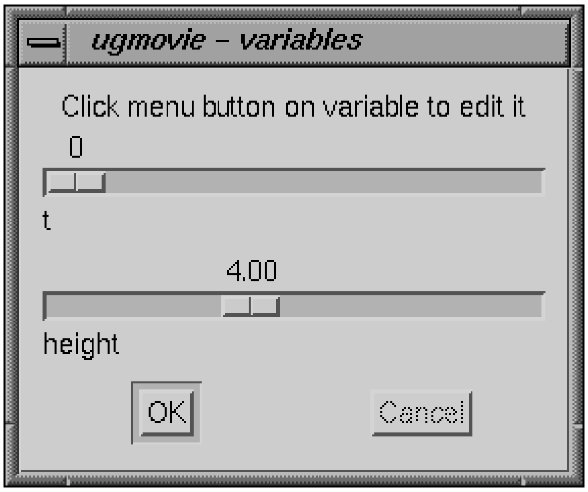


Figure 2 — Variables Panel

re-evaluated with the new value. Using the slider implicitly edits the definition of the variable to be the constant value given by the slider. More sophisticated editing of a variable's definition may be initiated by pressing the right mouse button over the variable's name in the Variable Panel — which causes the Edit Variable Panel to pop up.

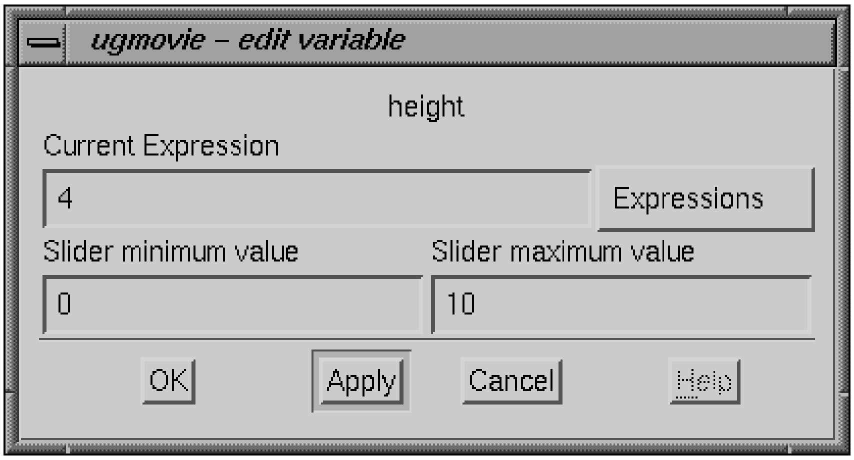


Figure 3 — Edit Variable Panel

The Edit Variable Panel has type-in fields for the *minimum* value and the *maximum* value for the slider and for the value expression of the variable (the expression may be a constant). Several prefabricated expressions are available by pressing the *Expressions* button. See the language extensions section above (§ 2.6) to find out what is legal in expressions. Due to an implementation restriction, only one variable may be edited at a time.

4. Implementation

Much of the structure of ugmovie comes from it being a ‘‘mixed-model’’ GLX program [GLX91]. What the above means is that all the 3-D graphics is done using the IRIS GL graphics language in conjuction with the graphics hardware and that all the windowing and input event handing are done using the X windowing system. Choosing to implement ugmovie as a mixed-model program was done to increase its lifespan as SGI plans to drop the windowing support in GL and replace it with an output-only graphics library called OpenGL. To change ugmovie from using GL to OpenGL will be trivial since ugmovie does not use any of the discontinued GL features. Most of the rest of the structure of ugmovie comes from its support of the bump [Oakland87] language extensions to the UNIGRAFIX language — including the new **vardef** statement.

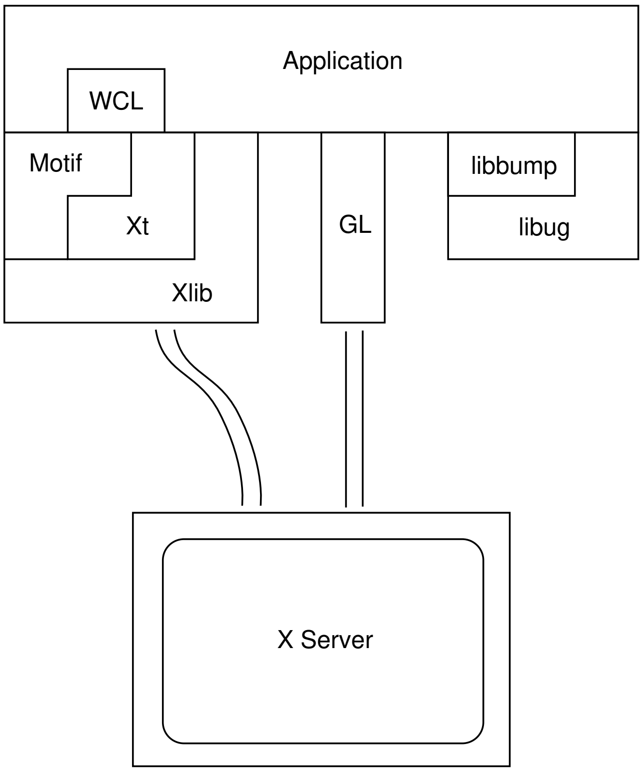


Figure 4 — Implementation Organization

4.1. X Contributions

The advantage of programming with X and with various X libraries is that your code should work on a wide variety of platforms and exhibit a standard look and feel. The disadvantage is that X is so low-level and customizable that your program gets bloated quickly when you use the look and feel libraries. A related disadvantage is the weight of the documentation that has to be read, especially when one considers the libraries used; it must be at least 40 pounds — probably higher. Code bloat and voluminous documentation seem to be the price to pay for portability at this time.

Following the SGI party line, we selected the Motif toolkit (Xm) [OSF91b] for the user interface toolkit. Consequently, the look and feel should match applications written by SGI and third-party vendors. Selecting the Motif toolkit implies using the X toolkit (Xt) [McCormack91], and the X library (Xlib), as each is layered on top of its successor. The X library is the only layer that talks directly to the X server. The Motif and X toolkits provide a widget interface to the programmer. A widget is a visual element of the user interface along with its behavior. For example, a slider is a widget, a menu is a widget, and menu buttons are widgets. The X toolkit also provides a mechanism for user customization of widget behavior through resource files. For example, the background and foreground colors or the font could be changed with a resource file. Applications typically have a default resource file that resides in */usr/lib/X11/app-defaults/*, and user's can make small modifications in the .Xdefaults file in their home directory.

To help manage the widgets in the user interface, we decided to use the Widget Creation Library [Smyth92]. WCL extends the resources in the X resource file to include resources that control the layout of the user interface. This speeds up prototyping the application and reduces the amount of hardwired user interface code in the program. WCL is a general solution and works with user interface toolkits other than Motif.

The interface between widgets and the application code is through callback functions. Each widget provides a set of conditions under which it will invoke a callback function. If a callback function is not provided, it is as if that condition were never present in the widget. Callback functions are also called for input events that occur within a widget. Many events are handled automatically by the widget, for example, the slider motion when moved by the mouse.

4.2. Bump and UNIGRAFIX Contributions

Bump was originally a stand-alone program, written using an earlier version of the current UNIGRAFIX data structure [Séquin90]. Part of the work behind the development of ugmovie was the conversion of the UNIGRAFIX data structure library and bump to ANSI C. The update of the UNIGRAFIX data structure library included merging changes that were made at SGI for the walk-through application. The major change was that colors were updated to include material properties and texture, and the internal color triple was changed from HSV to RGB. At the same time, the switch from double precision arithmetic to single precision was made since ANSI C does not promote single precision expressions to double precision unlike earlier versions of C.

After bump was updated to ANSI C, it was then divided into two parts: an object code library so that any application could use the bump functionality, and a driver that mimicked the original bump command line interface. Then the bump library was modified to understand the new **vardef** statement discussed above. Many bugs were fixed along the way, in particular, all line length limits were removed, expression recognition was enchanced with additional operators and functions. and it is now also possible to instance a definition that was defined outside of a **meval** from within the **meval**.

Earlier in section 2.5 on **meval** and **mexecute**, several reasons why there is an implementation restriction on their use where given. If they were allowed at the top level of the scene hierarchy, then the whole scene's data structure would have to be saved and restored for each frame. The requirement that they be in enclosing definitions is a significant time-saving optimization. The bump library interface implicitly enforces that restriction by only having the ability to modify the scene given to it, not to replace it completely.

4.3. GL Contributions

As little work as possible was done on the GL portion of ugmovie. We started with the GL renderer from **ugiris**, which only renders flattened scenes, and modified it to walk the instance tree using the hardware matrix stack. Walking the instance tree is important in ugmovie because it avoids the expense of reflattening the scene for each frame as instances move. Ugiris was able to flatten the scene because it only displays static scenes, so the flattening reduced the amount of time needed to display a scene. An optimization in ugmovie would be to flatten the instances in the scene that are not animated.

4.4. Future Directions

The GL renderer should be extended to handle the full UNIGRAFIX language — especially the new color attributes (material properties and texture) and the light and camera statements. Lights and cameras should be selectable, both on the command line as well as interactively.

The user interface can be extended to have UNIGRAFIX editing capabilities. Currently only the middle mouse button is used so the other buttons are free for other uses. Possibilities include picking, cut and paste, and shape editing. Adding mouse actions is as simple as registering a callback function and placing a reference to the callback function in the application defaults file.

With the new IRIS Inventor 3D Interchange File Format [Inventor93] and with more knowledge of the Motif toolkit, it should be possible to implement 3D cut and paste. The format does not maintain the topological information that the UNIGRAFIX data structure does, but it would allow UNIGRAFIX scenes to be placed in other SGI applications.

References

[Chen88] Michael Chen, S. Joy Mountford, Abigail Sellen, ‘‘A Study in Interactive 3-D Rotation Using 2-D Control Devices,’’ *Computer Graphics*, 22(4):121-129, August 1988.

[Couch94] Gregory S. Couch, ‘‘Berkeley UNIGRAFIX 3.1 Language and Data Structure,’’ May 1994.

[GLX91] *IRIX 4D1-4.0 Transition Guide, Programming Environment*, Silicon Graphics Inc., Mountain View, California, 1991. ch. 4 & 5.

[Inventor93] *How to Write an IRIS Inventor™ File Translator Release 1.0*, Silicon Graphics Inc., Mountain View, California, 1993.

[McCormack91] Joel McCormack, Paul Asente, Ralph R. Swick, *X Toolkit Intrinsics — C Language Interface*, MIT X Consortium, August 1991.

[Oakland87] Steven Anders Oakland, ‘‘BUMP, a Motion Description and Animation Package,’’ *Master's Report*, U. C. Berkeley, September 1987.

[OSF91a] Open Software Foundation, ed., *OSF/Motif™ Style Guide Release 1.1*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1991.

[OSF91b] Open Software Foundation, ed., *OSF/Motif™ Programmer's Reference Release 1.1*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1991.

[Séquin90] Carlo H. Séquin, Kevin P. Smith, ‘‘Introduction to the Berkeley UNIGRAFIX Tools Version 3.0,’’ Berkeley, California, November 1990.

[Smyth92] David E. Smyth, *WCL 2.02 — Widget Creation Library*, via anonymous ftp, June 1992. export.lcs.mit.edu:contrib/Wcl-2.2.tar.Z.

NAME

ugmovie — interactive viewing of bump animation extensions

SYNOPSIS

ugmovie [–**movie** *scene-file*] [–**mintime** *mintimeval*] [–**maxtime** *maxtimeval*] [*standard-X11-options*]

DESCRIPTION

Ugmovie is a tool for interactive viewing of the bump animation extensions to the UNIGRAFIX language on Silicon Graphics workstations. The –**movie** *scene-file* option lets one specify the initial scene to display. The –**mintime** *mintimeval* option specifies the initial minimum frame number. The –**maxtime** *maxtimeval* option specifies the initial maximum frame number.

Ugmovie's user interface was written using the Motif toolkit and follows the Motif conventions as much as it reasonably possible. All of the standard Motif keyboard menu accelerators work.

Mouse

To be compatible with other interactive UNIGRAFIX applications, only the middle mouse is used. Pressing the middle mouse button invokes a virtual sphere to rotate the scene. Pressing the control key at the same time as the middle mouse button invokes x and y translation. Pressing the shift key at the same time as the middle mouse button invokes z translation.

Menu Bar

The File menu is a standard Motif File menu. The Edit menu is also standard, but effectively disabled as there is no cut and paste of UNIGRAFIX scene elements. The Options menu leads to the bump variable editing and view parameter panels. And the Time menu provides for single stepping as well as continuous updating of the frame number.

ENVIRONMENT

UGHOME — root of installed UNIGRAFIX binaries

FILES

$UGHOME/lib/ugmovie — actual binary
$UGHOME/lib/Ugmovie.ad — X11 application defaults file for ugmovie

SEE ALSO

bump(1), bump(3), ug(5)
Ugmovie — Interactive UNIGRAFIX Movie Previewing

BUGS

Should allow the specification of different cameras and lights, both on the command line and interactively. The rendering code should be updated to understand material definitions and draw wires correctly.

AUTHOR

Greg Couch