# TSIPP Workbench - Working Widgets Without Code

Paul Welton
Nortel Networks
PO Box 3511, Station C
Ottawa, Ontario
K1Y 4H7
pdw@nortelnetworks.com

## Abstract

This submission describes an interactive tool for 3D/4D drawing named the “TSIPP Workbench”. It is based on, and derives its name from, the established Tcl extension TSIPP, a 3-D image specification and rendering toolkit for use with Tcl and Tk developed by Mark Diekhans, described in “TCL/TK TOOLS” [ref. 1]. The main goal of TSIPP is visualization; the understanding of data in science or engineering. To do this many calls to routines in the Toolkit are needed to construct a scene, with dimensions and positions of each component being passed in as numerical values. This can be a laborious task, and an interactive tool is particularly appropriate here, where the objective is a visual image, and immediate feedback on the appearance is invaluable. The TSIPP Workbench makes it possible to create 3-D images through interactive drawing, and encapsulate them as widgets which appear as visual scenes with photographic realism which can vary with time (animation) or other control stimulus. During an interactive session, the image of the widget is defined, as well as the methods to be supported and their effect on the image. These widgets can then be used to create graphical applications with minimal code.

The animation of the image encompasses more than just movements of the objects in the scene. Camera positions, positions of light sources, and the intensity and characteristics of the image can all be controlled by methods of the widget.

Implementation has followed the “Tcl Style Guide” [ref. 3] and the recommendations on “Namespace and Package Use” [ref. 4].

# 1 Features

The objective of the *TSIPP Workbench* is to give the user easy and productive access to all of the features of TSIPP.

## 1.1 Orthogonal Views

While 3-D drawing on a computer may appear difficult, 2-D drawing is a familiar extension to word processing packages. The GUI of the TSIPP Workbench presents three orthogonal 2-D views of the scene representing elevation, plan and side views. Behavior in the three orthogonal views resembles conventional 2-D drawing tools, except that they are linked together, and always remain consistent. When a selected object is moved by the user forward in the plan view for example, the object will be moved in the side view in sympathy.

Three types of manipulation of selected components are supported: movement, resizing and rotation. The three operations can be applied in an arbitrary order. Figure 2 on page 2 shows an example view of a plate-like cuboid which has been rotated slightly in each of the three views.

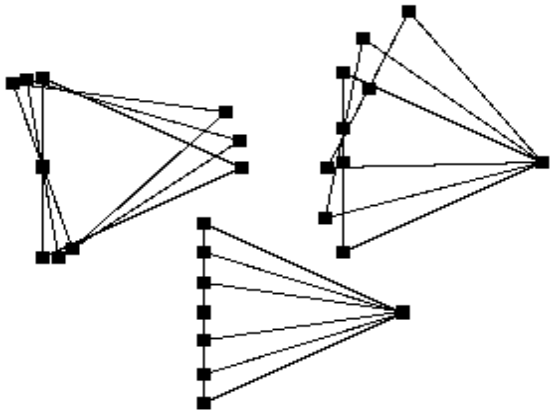
### 1.1.1 Resizing

Resizing of a selected object is invoked by dragging the mouse with M1 depressed on “handles” attached to each vertex (vertex of a bounding cuboid in the case of the ellipsoid). The motion is constrained in a manner appropriate for the type of component.

**For cuboids**, when one vertex is moved, the adjacent vertices must move so that the object remains a cuboid, with perpendicular faces.

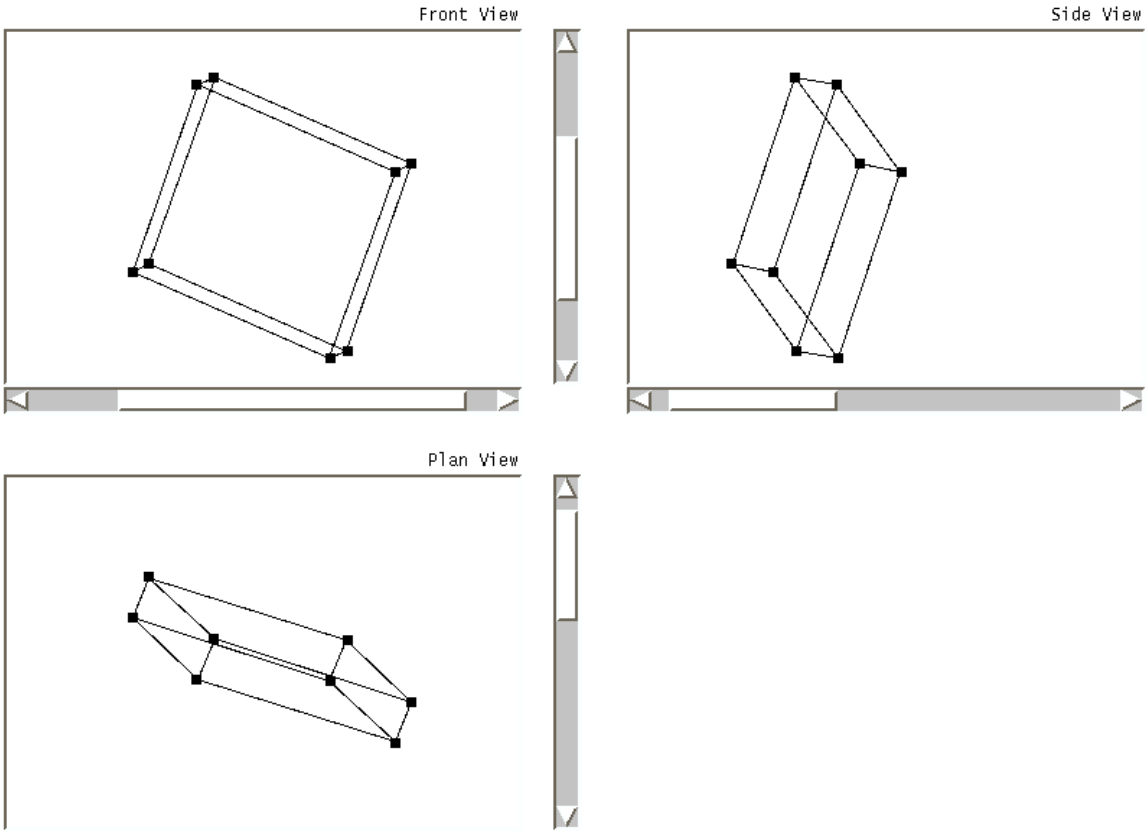
**For spotlights and cameras**, the representation in each canvas is constrained to be an isosceles triangle, with the object located at the vertex opposite the side of unequal length. Manipulation is limited as shown in figure 1 below. Any further degree of freedom of motion would only generate confusion for the user, as it would convey the impression that the underlying TSIPP object has parameters other than position, orientation and “divergence”, the degree of spread of the beam or field of view.

**Figure 1. Constrained Movement of Spotlights and Cameras**



The handle at the vertex at which the object is located, and an additional handle at the centre of the opposite side can be dragged arbitrarily, and the triangle remains fixed at the

**Figure 2. Linked Orthogonal Views in the MultiView Widget**




other such vertex. The remaining two handles control the divergence only, and can only revolve about the object, with their motion locked together so that they cannot affect the orientation of the object.

As a consequence of the constraints in handle motion, the handle does not necessarily follow the mouse pointer.

### 1.1.2 Rotation

Rotation of Cuboid and Cylinder objects is invoked by dragging the mouse with M3 depressed on a handle at a vertex. The Component will be rotated about an arbitrary centre of rotation, if this has been defined by user command, or if not, about an implicit centre having each coordinate equal to the mean of the corresponding coordinates of all the vertices.

### 1.1.3 Movement

Movement of all selected objects is invoked by dragging the mouse with M3 depressed on the canvas background.

### 1.1.4 Avoiding loss of accuracy through quantization to pixel resolution

When an object is created, each vertex will have coordinates that are an integral number of pixel units. On rotation, the handle that is dragged will be moved to another location with the same constraint, but this will usually imply that the other handles should be moved to a location with non-integral coordinates. If this was approximated by rounding to the nearest pixel, then a series of rotations would result in unacceptable cumulative errors. Keeping soft copies of the coordinates in floating-point format is not desirable, as the implementation of the “movement” manipulation (see section 1.1.3 above) is implemented using the ability of the canvas widget to

move whole groups of items, so that the real coordinates would become inconsistent with the soft coordinates.

The solution is to maintain the error differences between the pixel coordinates and the accurate indices. When resizing or rendering is to take place, the error differences and the current pixel coordinates are combined. When the operation is complete the error differences are recovered.

## 1.2 Saving the state of the Tool and Work in Progress

The state of the TSIPP Workbench can be saved as a plain-text incr-Tcl source file, which may be reloaded to restore the state of the tool. This Tcl source file can be thought of as the “source” for the image. As a human readable incr-Tcl program it provides the potential for interfacing to other image processing systems, or of implementing recovery from tool problems which make the file unreadable, which would be significantly more difficult if the image source was in binary format.

## 1.3 Defining Colors and Textures

A custom widget is provided for defining colors and textures. TSIPP provides a range of shader types, which define the color and texture of objects. The colors are modified using a color wheel taken from “Effective Tcl/Tk Programming” [ref. 2]. The GUI options are customized for the shader type selected, since some shaders require multiple colors and each has a different set of parameters.

## 1.4 Animation

Animation of the scene is achieved by defining two or more reference frames, and allowing the tool to interpolate and then render a sufficient series of intermediate frames for smooth motion. The frames may differ in terms of the position of objects in the scene, camera position and orientation, and the characteristics of the light sources. Thus the methods of the widget may show objects heating up and cooling down for example.

When a reference frame has been defined and rendered, the “fix” button is used to store that image in the “frameStore”, a data structure indexed by frame number. This contains a copy of the set of public variables of each of the objects overlaying one of the TSIPP objects in the scene, and a copy of the coordinates of the representation of the component in each orthogonal view. Thus it is a two dimensional structure indexed by “frame number” and “parameter name”.

When the tool interpolates a particular frame, it applies the interpolation algorithm in turn to each “parameter name” represented in the frameStore. The parameters may be numerical values, textual values such as Shader type, or lists of either of these scalar values, extending to an arbitrary degrees of lists-of-lists. For example, the Wood Shader contains a list of two colors, each of which is a list of hsb[1] values. Textual parameters, such as Shader type must not change between reference frames, but numerical values embedded in lists may change. Interpolation is performed for each parameter by testing it against each of the following cases in turn and using the first one that applies:

- If the reference point values are identical, then the interpolated value is the common value for the reference points.
- If the reference point values are lists, then the interpolated value is a list of the results of applying the algorithm recursively to each element in turn.
- If the reference point value is not a list then it is interpreted numerically. At present, only linear interpolation between the two closest points is available, but it is intended to add polynomial fit to a larger number of reference frames, and the option to link in a custom algorithm. Suppose the frame numbers of the reference frames are $r_0$ and $r_1$, and the values at these points are $v_0$ and $v_1$ respectively, then the interpolated value, v for frame r would be:

$$v = (v_0 * (r_1 - r) + v_1 * (r - r_0)) / (r_1 - r_0)$$

The result of the interpolation process depends on how the parameter is represented. For example, the fact that a color is represented in HSB rather than RGB format means that changing from pure BLUE to pure RED proceeds via GREEN and maintains full saturation; while with RGB representation it would proceed via magenta. Similar choices arise with positional information. If the angle of a block is interpolated, this results in rotational motion, while if the coordinates of the vertices are interpolated then this results in the block deforming when moving between the reference states.

## 1.5 Automatic Rendering

When this feature is turned on, any button or mouse event will cause the image to be rendered. Double-buffering of the image is used so that the image being rendered is not made visible unless the rendering completes. Any mouse or button event occurring before rendering completes aborts the rendering and initiates a fresh attempt, which will, if it completes, reflect any change to the image which has been made. To the user, the effect approximates to the rendered image tracking any editing done in the multiView, although there will be a noticeable delay. TSIPP offers four levels of rendering algorithm, trading rendering time against quality of image. Automatic Rendering is most

1. hue, saturation and brightness.

useful with the “LINE” rendering algorithm, which is the fastest.

If any error occurs, the “bgerror” procedure has been wrapped so as to cancel the Automatic Rendering mode. This is necessary because otherwise an infinite loop may develop in which any attempt to exit from the dialog entered on an error may trigger a rendering attempt and a further error.

# 2 Class Structure

The incr-Tcl classes used for this program fall into four groups:

- Megawidgets, representing a function of the program as well as representing an area of the visible GUI;
- Classes overlaying TSIPP objects. These form a hierarchy of which class Component is the root, and the most specific classes correspond to TSIPP objects.
- Classes associated with RLE files which are used in both the TSIPPwb program and the widgets that it generates for use in other applications.
- Utility classes, such as “DeepCopy”.

The relationship between these classes can be seen in figure 3 on page 5. The remainder of this section describes the classes in more detail.

## 2.1 Megawidgets

The TSIPPwb program is partitioned into a small number of mega-widgets[1] with two objectives:

- A highly structured program results with low coupling between modules.
- The megawidgets are fully defined in their own terms and may be reused in other applications.

The top-level layout of mega-widgets is shown in figure 4 on page 6.

### 2.1.1 MultiView

An object of this class is a widget to support interactive editing of 3D images, based on entry of information in orthogonal views.

A single instance “.v” is created. Objects of all classes, including Component and its specializations refer to this name. The public view of this class is described in more detail in section 3 on page 7.

### 2.1.2 VisualEdit

Several component types supported by TSIPP have similar sets of properties which need to be configured. These include the many different “Shaders” and “Light Sources” which are available, each requiring one or more colors to be defined together with a number of miscellaneous parameters, which are each either floating point values or enumerated types. An object of class VisualEdit is a widget to support the editing of these TSIPP objects.

All component types to be edited with the VisualEdit widget are overlaid by a specialization of the class “Visual”, which is itself a specialization of the class Component. The VisualEdit widget is not explicitly aware of the nature of the specializations, so that in principal, a new variant such as “DirectedLight” which is not presently supported, could be added as another specialization of Visual, and the VisualEdit widget would provide custom editing support for it without modification.

The MultiView widget creates a single instance of a VisualEdit widget with a pathname that is not available outside of the MultiView widget. The VisualEdit widget can be referred to outside using the method named “visualEditEval”. The public view of the class VisualEdit is described in more detail in section 4 on page 9.

### 2.1.3 GlobalEdit

This custom widget allows global options, file access and operations on selected objects to be controlled. A single instance “.g” is created. Objects of all classes, including Component and its specializations refer to this name.

### 2.1.4 MotionEdit

This custom widget allows the animation of the scene to be controlled. The tool can be placed in a “record” mode in which each frame of the animated scene is rendered and written to an RLE file, or into playback mode in which the resulting RLE file is displayed frame-by-frame. This widget is also used to revisit existing reference frames, delete them or to define new reference frames.

A MultiView widget creates a single instance of a MotionEdit widget with a pathname that is not available outside of the MultiView widget. No objects other than “.v” refer to it, and in fact, it has no public methods or procs.

## 2.2 Classes overlaying TSIPP objects

Each TSIPP object is overlaid by an object of class Component, described more fully in section 5 on page 10. The

1. Currently they are not actually Mega Widgets in the iTk sense - this is being worked on.

**Figure 3. Class Diagram**

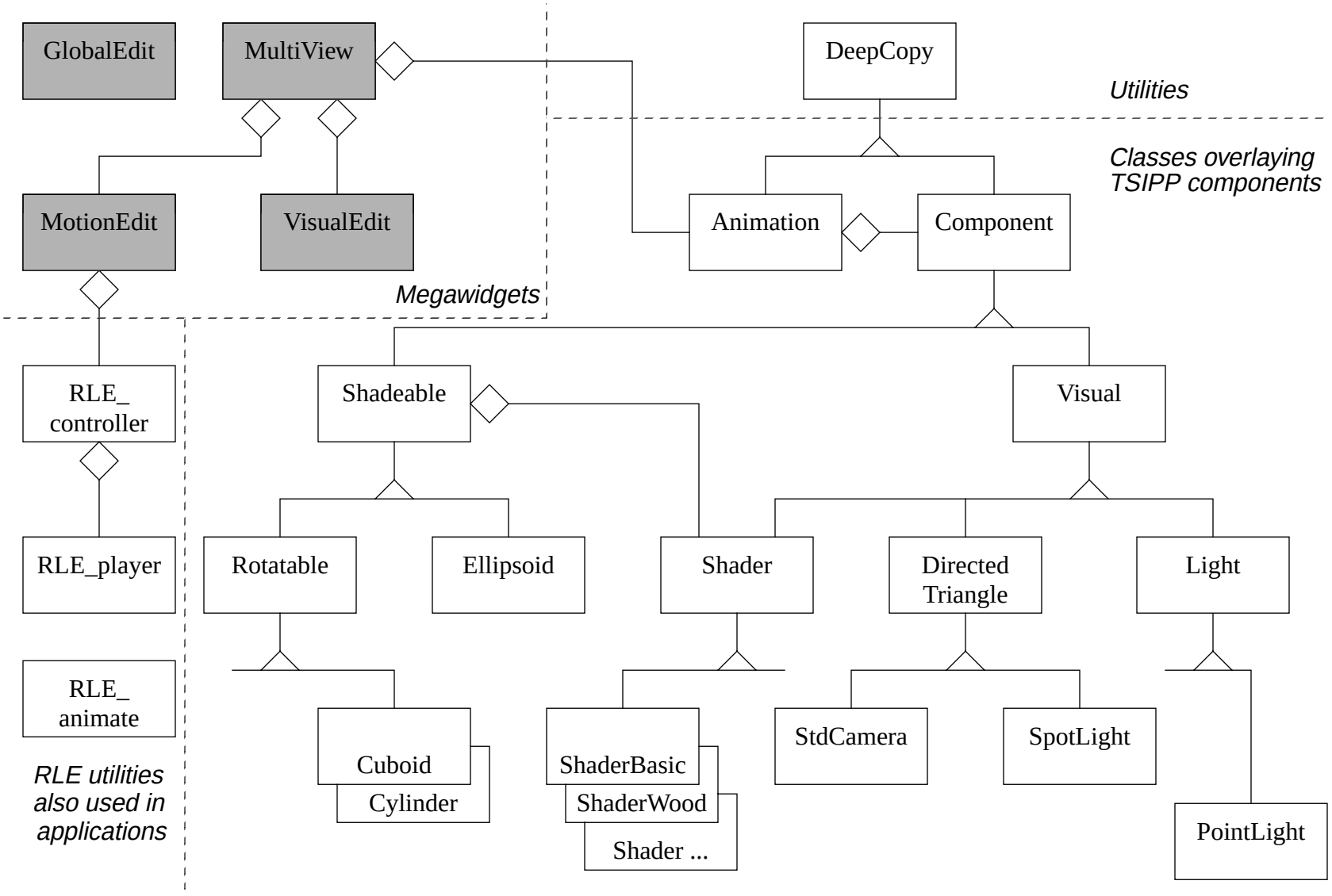
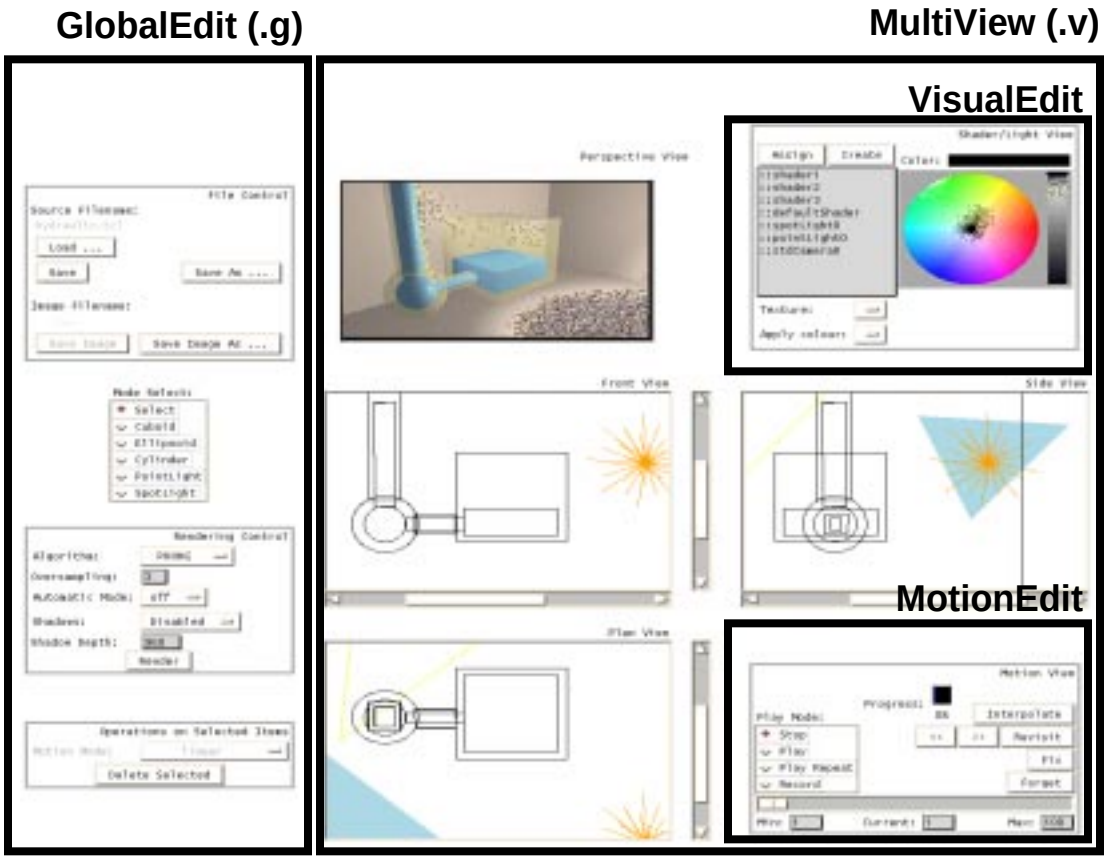


**Figure 4. Widget Hierarchy**




collection of Components comprising the stationary picture or an animation that is being edited are overlaid by an object of class Animation. A MultiView widget creates and manages an object of class Animation with a fixed name “c” which is known publicly[1].

## 2.3 RLE File Managers

### 2.3.1 RLE_player

This is the most basic object overlaying an RLE file. An object of this class would normally be used in any application incorporating a widget generated using the TSIPP Workbench.

### 2.3.2 RLE_controller

This object is a GUI to control an RLE_player object which it owns. An object of this class is used inside of the MotionEdit object of the TSIPP Workbench, but can also be used in standalone applications.

### 2.3.3 RLE_animate

An RLE_player that loads all images from the RLE file into separate photo widgets. This potentially allows faster random access to frames compared with serial reading of an RLE file. In practice, access to sequential frames is slower. Since the number of frames that can be loaded in this way is constrained by the memory available, this approach has limited application.

1. This implies that at present, only one MultiView widget can be created.

## 2.4 Utilities

### 2.4.1 DeepCopy

Objects inheriting this class may call putObject to write code to a file that when executed, will recreate the object.

# 3 Class MultiView

In this section, a more in-depth description is given of the public interface to objects of this class

This widget resembles a conventional widget in that it is created by the command “multiView <win>”, but an integrated Mega-widget is not created - an object <win> of class MultiView is created but the widget name has the suffix “_win” added. The extended name is returned by the command and should be referenced in pack or grid commands, while the original name is used to access the specific MultiView methods.

## 3.1 Public Methods

### 3.1.1 addItemBindings

This method adds bindings for M1 and M3 mouse-press events to the representation of the component with a specified tag in a specified canvas of the MultiView widget.

Arguments:

canvas - the canvas on which the bindings are to be created.

tag - the tag identifying the item to which the binding should be applied.

Results: No return value.

### 3.1.2 skipCanvasBindings

This method is typically used by an item binding in a canvas of a MultiView object that wishes to suppress any canvas binding that may also be invoked. It is analogous to using a “break” command in a binding script to suppress further processing of the binding tags. However, using break in an item binding does not suppress widget bindings.

Arguments:

canvas - the canvas on which the bindings are to be suppressed.

Results: No return value.

### 3.1.3 transform2d3d

This method is used in the construction of binding scripts to be used in a binding for a canvas of a MultiView widget, or an item in such a canvas. Suppose that the binding is to invoke a method which takes 3D coordinates; x, y, z as arguments. Two of those coordinates can be obtained from the standard binding substitutions %x and %y, while the third is indeterminate. The mapping between %x, %y and the indeterminate value depends on the canvas. This method returns the list of three elements, {%x %y *} permuted so that so that the substituted values are mapped to the correct arguments, and “*” is passed to the indeterminate argument.

Arguments:

canvas - the canvas to which the 2D coordinates relate.

Results: Returns {%x %y *}, permuted appropriately.

### 3.1.4 visualEditEval

A MultiView object contains an embedded widget of class VisualEdit. The pathname of this widget is not known publicly, but the embedded VisualEdit widget may be asked to execute methods using this method of the MultiView that owns it. “.v visualEditEval <method> <args>” is equivalent to “<name of VisualEdit> <method> <args>”.

Arguments:

args - the command and arguments which the embedded VisualEdit widget is to perform.

Results: The result of the supplied command.

### 3.1.5 clearSelection

This method deselects all selected items in all canvases.

Arguments: None.

Results: No return value.

### 3.1.6 safeSource

To reload a source file, it must be sourced from within the MultiView object so that it can use variables, such as those holding the canvas pathnames. In addition, it is valuable to source the file completely, even if such commands fail. This will allow backwards compatibility with old source files that, for example, attempt to configure variables that no longer exist.

This method is equivalent to invoking the standard source procedure within the context of the object (so that it can

access public and private variables of the object directly), except that if any of the commands result in an error then the error message is written out but the source command is not aborted. Note: this procedure is not related to “safe interpreters”.

Arguments:

filename - name of a file containing Tcl source code which will reconstruct the MultiView to a former state.

Results: No return value.

### 3.1.7 setAutoRender

This method is used to turn the “auto-rendering” feature on or off. See section 1.5 on page 3.

Arguments:

autoRender - new state of this option, which may be “off” or “on”.

Results: No return value.

### 3.1.8 Operations on Selected Objects

The following public methods perform operations on the objects of class Component which are represented as “selected” in the MultiView object.

### 3.1.9 setMotionMode

The motion mode for all selected objects are set to the selected value, unless “various” is selected, in which case the settings are not changed.

Arguments: None.

Results: No return value.

### 3.1.10 commonMotionMode

This method is invoked whenever the selection is reduced, in which case it is necessary to scan all remaining selected objects. If the existing motionMode is “various” then it may become an explicit value as a result of the deletion. If there are no remaining selected items, then the state of the motionMode is regarded as disabled.

Arguments:

pState - Name of a variable in the context of the caller which will be set to “disabled” if there are no selected items, and “normal” otherwise.

pValue - Name of a variable in the context of the caller which will be set to the common motion mode of all of the selected items, or “various” if there is a mixture.

Results: No return value.

### 3.1.11 deleteSelected

This method causes all selected items, associated handles, and the associated objects of class Component to be deleted. A check is made for any items which are tagged “noCasualDeletion”, and if this is found, then an error results and no items are deleted. This tag is used for the component StdCamera.

Arguments: None.

Results: No return value.

## 3.2 Public Procs

### 3.2.1 getCanvasVariables

This proc returns the constant list {Front Plan Side} of the names of public variables which contain the pathnames of each canvas. The reason for returning the names of the variables rather than the pathnames themselves is that this routine is called by routines which write out the source files for the animation being edited. It is desirable that these source files can be sourced by any object of class MultiView, which may have a different pathname from that of the MultiView object that created the file. Therefore, the file should contain the variable names only, not the pathnames. As the variables are public, they can of course be de-referenced with the standard method “cget”.

## 3.3 Public Data

### 3.3.1 Front, Plan, Side, Perspective

These public variables contain the pathnames of the respective canvas widgets within the MultiView object.

### 3.3.2 canvasMargin

This public variable specifies the margin in pixels which is left around canvas items when a canvas is automatically resized in pixels, and defaults to 100.

## 3.4 Behavior

In this section the behavior of the widget, as defined with bindings for various events is specified.

### 3.4.1 Canvas

On creation, the following bindings are created in each canvas:

**Button-1** - If the mode selected in the associated GlobalEdit widget (.g) is “select”, then a “rubberband selection” mode is entered, in which a rectangle can be dragged with the mouse, and on release, all fully enclosed items will be selected in addition to any existing selection. The “rubberband selection” is maintained as a rectangle with one vertex at the initial pointer coordinates, and the opposite vertex tracking the mouse coordinates.

In any other mode, an item corresponding to that mode is created. The new item is selected, and items already selected are deselected. The new item may be resized in the same way as a selected item in select mode.

**Button-3** - All selected items are dragged with the mouse.

### 3.4.2 All Items

When addItemBindings is called, the following bindings are created for the item. They take precedence over the canvas bindings described above.

**ButtonPress-1** - If the mode selected in the associated GlobalEdit widget (.g) is “select”, then all currently selected items are deselected, and then the item to which the mouse is pointing is selected.

**ButtonPress-3** - If the mode selected in the associated GlobalEdit widget (.g) is “select”, then the selection state of the item to which the mouse is pointing is toggled; i.e. it becomes selected if it was formerly deselected, and vice-versa.

### 3.4.3 Selected Items

When a component is selected the following bindings are created for the handles. They take precedence over the canvas bindings and the item bindings described above.

**Button-1** - The handle may be dragged with the mouse to re-dimension the component. The effect on the other handles is dependent on the component type.

**Button-3** - For items that support rotation, the Component will be rotated about its centre of rotation, if defined, or if not, about an implicit centre with each coordinate equal to the mean of the corresponding coordinates of each vertex.

# 4 Class VisualEdit

## 4.1 Public Methods

### 4.1.1 add

This method adds the name of a specified object to the scrolled listbox of Visual objects.

Arguments:

visual - Name of an object of class Visual.

Results: No return value.

### 4.1.2 remove

This method removes a specified item from the listbox, and if this is the selected item, then the entry widgets of the VisualEdit are disabled.

Arguments:

visual - Name of an object of class Visual.

Results: No return value.

### 4.1.3 select

This method forces the selection in the scrolled listbox to the specified visual, and updates the colordial subwidget to reflect the settings of the current visual. This procedure is called when a single component is selected in one of the orthogonal views, and this is the visual associated with it to be selected here.

Arguments:

visual - Name of an object of class Visual.

Results: No return value.

### 4.1.4 currentSelection

This method returns the identity of the object currently selected by the scrolled listbox of Visual objects.

Arguments: none.

Results: returns the name of the object currently being edited.

### 4.1.5 setColorTargets

This method is used to configure the names of the colors used for the particular Visual being edited. For example, the “Wood Shader” requires two colors; “base” and “ring”.

Arguments:

lColors - list of color identifiers.

Results: No return value.

### 4.1.6 visualToColorDial

This method updates the ColorDial to reflect the selected Visual.

Arguments: None.

Results: No return value.

### 4.1.7 visualToAttributeEntries

This method is a more comprehensive version of the previous method which updates all widgets within the VisualEdit widget to reflect the selected Visual.

Arguments: None.

Results: No return value.

# 5 Component

Each TSIPP object is overlaid by an object of class Component, or a specialization.

## 5.1 Public Methods

### 5.1.1 File Transfer

putObject - writes itself to a specified file pointer. The code generated consists of calls to one of the following:

addRotationCentre - adds an indicator to rotatable objects to indicate the position about which they will rotate.

components - makes the Component owned by the master animation.

configure - loads scalar variables of the Component.

coordStore_Load - loads data into the coordinate store. The coordinate store is part of the frameStore, but is used to restore the Multiview displays rather than public variables of the Component.

display - displays the object in each canvas of the MultiView with the same appearance as it had when it was saved.

error_load - loads data into the error vectors.

frameStore_Load - loads data into the frameStore.

lower - StdCamera is lowered in each view that it is represented, as it is represented by a filled item and would obscure other items if left on top.

### 5.1.2 FrameStore Operations

framePut - stores the current state for a specified frame.

frameGet - recovers the state for a specified frame.

frameInterpolate - compute the state for a specified frame.

frameForget - forget the state for a specified frame.

frameSearch - look for a remembered frame, from a specified starting position in a specified direction.

frameExists - return whether the state is known for a certain frame.

### 5.1.3 Canvas Operations

offCanvasSelect - This default method does nothing, but in specializations it performs the actions necessary when the component is selected, other than those actions specific to individual canvases of the MultiView widget.

display - draw the object on a specified canvas.

defaultDisplay - create the initial display of an object.

addRotationCentre -

activateRotationCentre - used by action procedures of widgets in “.g”.

select - adds handles to all vertices of the representation of this component in the specified canvas and makes it appear to be selected.

deselect - removes handles from all vertices of the representation of this component in the specified canvas and returns it appearance to an unselected state.

### 5.1.4 3D image Creation

multiView_to_TSIPP - creates the parameters required by TSIPP to create objects from the canvas representations in the MultiView and the VisualEditor.

construct - call the TSIPP primitives to create the object using the current TSIPP parameters.

### 5.2 Public Procs

windowToCanvas - Canvas utility for converting between coordinates of the visible area, and coordinates of the canvas.

mouseStart - Utility for noting the current mouse coordinates.

mouseRelative - Returns the change in mouse coordinates since the last call to mouseStart or mouseRelative.

### 5.3 Public Data

#### 5.3.1 motionMode

The value of this variable defines how the motion of the object will be interpolated, and must be one of the following:

**Linear** - Each value is found by linear interpolation of the two closest points to the frame number being interpolated.

**Rotate in Front/Plan/Side** - Each value is interpolated so that it describes a point moving in a circle about the centre of rotation at a constant velocity.

## 6 Summary of Work Done

The TSIPP Workbench has been implemented as an incr-Tcl program which demonstrates the generation of 3D rendered drawings, and of animating them. The full capabilities of TSIPP are not yet fully accessible; only blocks, cylinders, ellipsoids and either PointLights or SpotLights are available at present. However with this limited palette, several effective and realistic animations have been created in the form of an RLE[1] file. A widget template is available to allow the animation to be used as a widget in another Tk application. The widget may be commanded to display a particular frame or to “play” the animation.

## 7 Software

You may access the software from the Tcl Contributed Sources Archive:

```
http://www.NeoSoft.com/tcl/ftparchive/
                    sorted/graphics/TSIPPwb
```

At the time of writing, version 1.3 is the latest available.

As the TSIPP Workbench has dependencies on many packages, complete binaries have been provided (for the Linux Operating System only).

Please refer to the README file, and the HTML documentation `html/index.html` and in particular to the list of known deficiencies in `html/knownProblems.html`.

## 8 Conclusion

The power of the TSIPP 3-D image specification and rendering toolkit can be appreciated far more readily through a point-and-click GUI than through the Tcl command interface directly. Incr-Tcl provides the means to implement such a GUI rapidly and effectively.

Smooth motion animation has heavy requirements for both CPU power and disk storage capacity. With the TSIPP Workbench, the images are pre-rendered and written to disk. Today, there will be many applications where scenic widgets are desirable, but the computer resources are either not available or not justified. However, it must be remembered that during the last 20 years, the computers available to the public for home or business use have increased in disk capacity from 10 MByte to 10 GByte, and CPU speeds from 1 MHz to 1 GHz. Although this rate of progress may not be continued, applications development must not underestimate future capabilities which may make 3D animation commonplace.

## 9 References

[1] “TCL/TK TOOLS”, Mark Harrison and other contributors, O’Reilly, 1997, ISBN 1-56592-218-2.

[2] “Effective Tcl/Tk Programming”, Mark Harrison, Michael McLennan, Addison Wesley Longman, 1997, ISBN: 0-201-63474-0.

[3] “Tcl Style Guide”, Ray Johnson, Sun Microsystems, Inc., August 22nd, 1997.

[4] “Namespaces and Packages”, William H Duquette, 2000.

1. Run-Length Encoding. This is the multi-frame storage format used by the Utah Raster Toolkit, provided as part of TSIPP.