The following paper was originally published in the
Proceedings of the Fifth Annual Tcl/Tk Workshop
Boston, Massachusetts, July 1997

# InvenTcl: Interpretive 3D graphics using Open Inventor and Tcl/[incrTcl]

Sidney Fels, Silvio Esser, Armin Bruderlin and Kenji Mase
ATR MI&C Research Laboratories
Kyoto, Japan

# InvenTcl: Interpretive 3D graphics using Open Inventor and Tcl/[incr Tcl]

Sidney Fels, Silvio Esser, Armin Bruderlin and Kenji Mase,
*ATR MI&C Research Laboratories*
*Seika-cho, Soraku-gun, Kyoto, 619-02, JAPAN*
*fels@mic.atr.co.jp  +81 774 95 1448  fax: +81 774 95 1408*

## Abstract

Open Inventor is an object oriented 3D graphics toolkit written in C++. Because Open Inventor is written in C++, typical user code development consists of a program/compile/debug iteration cycle. This paper introduces *InvenTcl* which is an interpretive version of Open Inventor using Tcl/Tk [4] and [incr Tcl] [3]. The advantages of InvenTcl include: script-able and direct manipulation of 3D objects in an Open Inventor scene, easy prototyping of 3D graphics and animation, and low-bandwidth communication of 3D scenes and animations (using scripts).

## Discussion

There are three command types provided by InvenTcl which correspond to the main functions available in Open Inventor: object creation commands, object interaction commands and animation commands. For object creation there are command names for instantiating objects. These commands have the same name as the class names in Open Inventor, e.g. in InvenTcl there is a command called SoSeparator[1] which creates a new separator [incr Tcl] object with corresponding methods to access the public methods defined in Open Inventor for an SoSeparator. For interaction, InvenTcl has binding mechanisms to allow Tcl procedures to be called when objects in the 3D scene are selected. For animation commands, InvenTcl provides access to animation functions found in Open Inventor (i.e. engines and sensors). To illustrate creation commands with an example, the following code shows how a simple scene graph is created interpretively:

```
SoSeparator >root>separator1
SoMaterial >root>separator1>material1
SoCube >root>separator1>myCube
```

This series of commands adds an SoSeparator node *separator1* to the root node, an *SoMaterial* node to separator1 and an *SoCube* node to separator1. These commands coupled with other Open Inventor commands would display a cube with the material properties specified by material1.

Notice, that we use the '>' notation to specify parent/child relationships.

There are four main technical issues to deal with to make Open Inventor interpretive:

1. accessing Open Inventor's object functions and the object's public methods and variables from the Tcl interpreter,
2. Open Inventor event management within Tcl/Tk,
3. binding Open Inventor objects to Tcl procedures and interaction modes,
4. synchronization of Open Inventor and Tcl processing.

To implement access to Open Inventor objects from Tcl we write command *wrappers* around the C++ instantiation functions for the Open Inventor library classes. Thus, a Tcl command is created with the same name as an Open Inventor class for instantiation. To access public variables such as fields, we provide command-line options like width, length, or radius. Alternatively, fields can be set after creation analogous to the Tk *configure* commands using Inventor's callback functionality with the appropriate *ClientData* pointer set. Our current implementation does not work effectively with the inheritance relationships specified in the Open Inventor class libraries and has commands which are tailored to our project [1]. In particular, we do not have convenient access to all the public methods for our classes. To remedy this we plan to use a utility called Itcl++[2] to convert Open Inventor object's methods into [incr Tcl] classes. Itcl++ is a utility which converts C++ class hierarchies into [incr Tcl]. It provides access to public methods inside each class and preserves the inheritance hierarchy. In [2], 32 classes and 190 member functions from the Open Inventor library were converted to [incr Tcl]. Using this utility we plan to convert the remainder of the class hierarchy of Open Inventor and integrate it with our notation, event handling, binding and synchronization

[1] An SoSeparator is a common object used in Open Inventor programming. It is used to isolate the effects of nodes in a group from other nodes in a scene graph.

techniques (see below). On top of the class hierarchy created with Itcl++, we will add the operator '>' for specifying parent/child relationships. Having this operator is useful for integrating Open Inventor scene graph descriptions with path specifications in Tcl/Tk. Itcl++ does not provide access to any public variables. We are currently addressing this issue by altering our approach for providing configure style access to public variables to work with Itcl++.

An Open Inventor event handler is installed as an asynchronous handler in Tcl to manage any Open Inventor events. Implementation of event management within Tcl does cause some performance penalty. Here is the Open Inventor event manager we use:

```
XtInputMask m;
XEvent event;
XtAppContext t;
t = SoXt::getAppContext();
if( m = XtAppPending(t ) ) {
  if ( m & XtIMXEvent ){
    SoXt::nextEvent(t, &event);
    SoXt::dispatchEvent(&event);
  } else   {
      XtAppProcessEvent( t, m );
  }
}
```

The main role of this handler is to find events which are related to Open Inventor and dispatch them to Open Inventor.

The main interaction command in InvenTcl is the `Ibind` command. The `Ibind` command allows users to bind an event and a callback procedure to objects in the Open Inventor scene graph. When used with objects in the scene graph this command is analogous to the canvas widget bind method for binding to objects on the canvas. However, our current version of InvenTcl allows only one scene graph and the `Ibind` command implicitly binds to objects in the one and only scene graph. Thus its syntax resembles the Tk bind command even though the objects that are bound are more like 3D versions of the canvas widget. Further, we do not provide support for Tcl binding to some of the interaction buttons and sliders that are provided by Open Inventor. Here is an example of the `Ibind` command:

```
Ibind >root>player>p5 <Ctrl-Button-1-Up>
      {puts "here"}
```

The binding mechanism is implemented using a combination of the event callback mechanism provided in Open Inventor and a Tcl_HashTable addressed by each object in the scene graph which is bound. Each object (referenced by a path in the scene graph) in the hash table has a structure associated with it to keep track of the bound callback function and any user call back data. In the example above, the path specified is >root>player>p5 and the callback function is {puts "here"}. The event which triggers the callback is a <Ctrl-Button-1-Up> event. When an event occurs, i.e. <Ctrl-Button-1-Up>, an Open Inventor callback node in the scene graph calls a generic callback handler. This generic callback handler looks in the hash table to see if the scene graph path which was selected is in it. If so, the generic callback gets the associated structure from the hash table and calls the bound callback function with the user data.

Synchronization between autonomous event driven Open Inventor activities and Tcl/Tk is performed using a global semaphore.

## Summary

In summary, to date, the following is working:

- Implementation of a small subset of the Open Inventor library with limited access to public variables and methods.
- Integration of event management of Open Inventor and Tcl/Tk events.
- Implementation of a 3D binding mechanism allowing selection of 3D objects to call Tcl procedures.
- Implementation of synchronization between Open Inventor and Tcl/Tk.

Our current work is focussed on enhancing Itcl++ to translate all of Open Inventor's class library. To do this, we are modifying Itcl++ to allow access to public variables. Further, the classes created using Itcl++ are being integrated with our mechanisms for event handling, binding and synchronization. Once complete, we plan to use InvenTcl to create a 3D mega-widget canvas using the Tk [4] canvas widget as a model for design.


[1] Bruderlin, A., Fels, S., Esser, S., and Mase, K. Hierarchical agent interface for animation. In *Animated Interface Agents IJCAI workshop, to appear in Workshop Proc. of the International Joint Conference on Artificial Intelligence (IJCAI'97)* (August 1997).

[2] Heidrich, W., and Slusallek, P. Automatic generation of Tcl bindings for C and C++ libraries. In *Proc. of the Tcl/Tk Workshop* (July 1995).

[3] McLennan, M. [incr Tcl]: Object-oriented programming in Tcl. In *Proc. 1st Tcl/Tk Workshop* (University of Berkeley, CA, USA, 1993).

[4] Ousterhout, J. K. *Tcl and the Tk Toolkit.* Addison-Wesley, New York, 1994.