

EASYOGL: FAST PROTOTYPING PLATFORM FOR 3D MULTIMEDIA INTEGRATION USING TCL/TK AND OPENGL

S. Sidney Fels and Matt Yedlin

Dept. of Electrical and Computer Engineering, UBC, Vancouver, BC, Canada, {ssfels, matt}@ece.ubc.ca

ABSTRACT

EasyOGL is a Tcl/Tk extension that adds three new dynamically loadable packages: a new Tk 3D graphics widget, an OpenGL package and an OpenGLU package. The 3D graphics widget provides a context for drawing 3D graphics. The OpenGL/GLU packages provide interpretive access to the OpenGL/GLU libraries so that developers can script interactive 3D graphics that appear inside an active 3D drawing context. Since EasyOGL is embedded in Tcl, it provides an extensible, easy-to-use, fast prototyping environment for 3D multimedia applications.

1. INTRODUCTION

Scripting languages such as Tcl (Tool Command Language) [10] offer enormous advantages over lower level languages such as C++ and C for prototyping multimedia interfaces. The interpretive nature of these languages permit developers to focus on the functional requirements of the system being created rather than on dealing with compiler issues which slow down the process. Most modern scripting languages allow the developer to embed multimedia libraries inside the language allowing scriptable access to the library's functionality¹. Associated with Tcl is Tk: a toolkit that provides easy-to-use 2D widgets such as buttons, sliders, and entry widgets for building 2D graphical user interfaces (GUIs). Tk has emerged as one of the defacto standard 2D GUI toolkits for scripting languages including Perl [13], Tcl/Tk, and Python [8]. One deficiency in Tk has been the lack of strong support for a 3D graphics widget to provide easy creation of 3D interfaces and 3D visualization techniques. EasyOGL is an extension to Tcl that embeds the OpenGL [4] and OpenGLU libraries, as well as providing a 3D graphics widget in Tk.

EasyOGL is derived from two previous extensions for embedding OpenGL/GLU in Tcl/Tk: Tkogl1.0 [3] and *swiggl* [2]. Tkogl1.0 provides a 3D graphics widget for Tk. In Tkogl1.0, OpenGL commands are associated with the 3D widget. Many of the main OpenGL/GLU commands are supported through a modified OpenGL syntax passed to the 3D widget as a single command argument. Tkogl1.0 relies on display lists heavily to provide 3D graphics functions. The strength of this system is that the 3D graphics widget behaves like any Tk 2D widget, making it easy to integrate 3D graphics in a GUI. The weakness of the system is that it ties the OpenGL/GLU commands to the widget itself, destroying one of the main advantages of OpenGL/GLU: window

system independence. Another weakness is that its reliance on display lists makes code reuse very difficult. Further, not all of the OpenGL/GLU commands are supported and the syntax has been modified from standard OpenGL/GLU.

The second embedded system is called *swiggl* which comes as part of the Simplified Wrapper and Interface Generator (SWIG) [2]. *Swiggl* comes from "wrapping" OpenGL, OpenGLU and the glaux libraries inside of Tcl so that users can access the libraries as script commands. Glaux provides functions to create a 3D graphics context into which OpenGL and OpenGLU commands render graphics. The strength of this approach is that the complete OpenGL/GLU libraries are accessible from the interpreter. The main disadvantage is that the 3D graphics context provided with glaux does not integrate at all with Tk, making the creation of a 2D GUI difficult.

EasyOGL takes the best of both systems and glues them together. The 3D widget is taken from Tkogl1.0 but the OpenGL/GLU commands are removed. This provides the 3D drawing context for OpenGL/OpenGLU commands. As in *swiggl*, SWIG wraps the OpenGL/GLU libraries. Additional glue code manages the redrawing of windows, supporting multiple 3D widgets, supporting 2D text in the 3D widget, and providing additional convenience functions. Thus, the 3D graphics context is kept separate from any OpenGL/GLU commands as originally intended with these libraries.

Creating a 3D widget with the appropriate graphics context desired, such as double buffered, RGB modes, alpha plane size, etc., provides a simple way to use the system. Once created, a user specified procedure is called whenever the widget needs to be redrawn. This redraw procedure contains OpenGL/GLU scripted code to draw the desired 3D graphics. Sample code and results are shown in figures 2 and 3.

Tcl is a glue language, allowing applications to be combined together easily. Additionally, Tk provides an easy to use 2D GUI builder. EasyOGL is a dynamically loadable extension to Tcl/Tk that provides a 3D widget compatible with Tk as well as Tcl command access to the full set of functions in the OpenGL/GLU libraries. By embedding video and audio libraries, and other multimedia libraries (as well as multimodal interface libraries, such as instrumented gloves, joysticks, and MIDI libraries) in a similar manner, users have a foundation for fast prototyping interactive 3D multimedia environments.

2. RELATED WORK

EasyOGL derives from Tkogl1.0 [3] and uses the SWIG to wrap the OpenGL/GLU libraries as discussed in section 1. The OpenGL and OpenGLU libraries are function-oriented rather than object-oriented, thus, fit naturally inside a function-oriented scripting language such as Tcl/Tk. In contrast, InvenTcl [5] provides an em-

This work was supported by a grant from ATR MI&C Research Laboratories, Kyoto, Japan and TLEF grant from the University of British Columbia with N. Burlinson.

¹Tools such as SWIG [2] perform the embedding automatically using the library's header file.

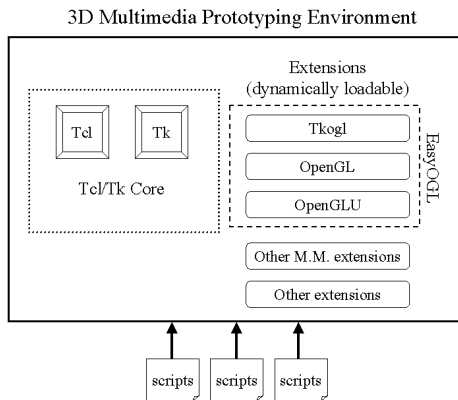


Figure 1: Overview of the EasyOGL packages.

bedding of Open Inventor inside of [incr Tcl] [9]. [incr Tcl] is an object oriented extension to Tcl/Tk. Open Inventor is a C++ class library for creating 3D graphics that sits on top of OpenGL. It uses a scene graph representation for 3D scenes. Since it is object-oriented, it is more suited for embedding in an object-oriented scripting language such as [incr Tcl] or Python. The main disadvantage of InvenTcl is that it depends upon the object-oriented extension to Tcl/Tk, [incr Tcl]. One solution to this problem is to embed Open Inventor inside of Tcl/Tk and provide command structures that mimic object-oriented behaviour. This is the approach taken in Ishells [6]. Other systems that embed 3D graphics inside of scripting languages include Alice [12], togl [11]², and tkSm [7].

3. EASYOGL: SYSTEM STRUCTURE

EasyOGL is a set of three independent packages that are dynamically loadable inside of the Tcl interpreter. Figure 1 shows the relationship of the packages to Tcl/Tk. EasyOGL (*Tkogl3.0*) is the modified version of *Tkogl1.0* [3] discussed in section 3.1. It provides the 3D graphics context as a tk widget. The other two packages, *OpenGL* and *OpenGLU*, are the wrapped OpenGL and OpenGLU libraries respectively. SWIG is used to wrap the libraries, as discussed in section 3.2.

3.1. 3D Graphics Widget

One of the main contributions of EasyOGL is the separation of 3D drawing commands from the window manager/drawing context. This was accomplished by extracting the 3D widget code from *Tkogl1.0* and adding glue code to manage drawing control of the windows. A 3D graphics widget is created using the following syntax:

```
OpenGLwin <path> [configuration parameters]
```

²toGl provides a 3D graphics context for graphics to be rendered from OpenGL commands called from C code.

-accumsize < size >	specifies that <i>size</i> bit planes should be supported for red, green and blue
-alphasize < size >	specifies <i>size</i> bit planes for the alpha component
-aspectratio < ratio >	sets the aspect ratio.
-context < path2 >	makes the current OpenGL context share display lists with another OGLwin graphics widget
-depthsize < size >	specifies the number of bit planes for the depth buffer
-doublebuffer < flag >	specifies whether or not to use double buffered mode
-height < height >	specifies the height of the window in pixels
-refresh < TCLexpr >	< TCLexpr > is executed whenever a redraw window event occurs (see subsection 3.1.1)
-stencilsz < size >	specifies the number of bit planes requested for the stencil buffer
-width < width >	specifies the width of the window in pixels

Table 1: Configuration parameters for a 3D widget. All but -*refresh* are the same as in *Tkogl1.0*.

where <widget path> is the Tk path name the user wants to give to the 3D widget being created. The configuration parameters specify attributes of the 3D widget (indicated using a dash - in Tk syntax). Figure 2 shows an example of the OGLwin command being used. Once created the 3D widget becomes the current graphics context. Any executed graphics commands will appear in this window. The following sections describes each of the properties of the 3D widget and how it differs from the implementation in *Tkogl1.0*. The main changes are new 1) configuration parameters, 2) commands for drawing control, 3) commands for 2D text and 4) convenience functions.

3.1.1. Configuration Parameters

All but one of the 3D widget configuration parameters come from *Tkogl1.0*. The parameters are listed in table 1. One configuration parameter was added to the 3D graphics widget from *Tkogl1.0*: *refresh*. The *refresh* parameter takes a Tcl expression as its argument. This parameter effectively sets up a call back so that the refresh expression is executed whenever the widget needs to be redrawn, such as when it receives a window expose event. Typically, users put procedure calls or OpenGL/GLU commands in the expression, as shown in figure 2.

The call back mechanism is crucial for providing flexible code reuse. Users can write generic drawing functions that will change their drawing behaviour based on passed arguments. Thus, different 3D widgets can use the same drawing functions and need only pass arguments to get the desired rendered graphics. Thus, libraries of 3D utility scripts can be developed and reused easily.

3.1.2. 3D Widget Commands for Drawing Control

Three commands are associated with controlling drawing in the 3D widget:

- *makeCurrent*
- *swapBuffers*
- *redraw*

The *makeCurrent* command makes the 3D widget the current active 3D drawing context. This means that any subsequent 3D drawing commands will render in this window. The *swapBuffers* command will swap the front and back buffers of the current 3D widget. The *redraw* command will force a window redraw event to occur which in turn causes the 3D widget to execute the *-refresh* expression. This causes the current 3D widget to be redrawn. These commands are implemented in Tcl/Tk by wrapping the equivalent C code accessible through the library according to the Tcl/Tk protocol. For example the *swapBuffers* command will call *gluSwapBuffers* in the OpenGLU library to swap the front and back buffers.

3.1.3. 2D Text Drawing Commands

There are two techniques for putting 2D text labels in a 3D widget. The first technique uses the normal OpenGL commands for creating raster fonts, then draws each character in the window at the specified raster position (by the *glRaster* command). This is the technique currently in place for running EasyOGL on a Unix platform. It is cumbersome, especially when many fonts are required.

The second technique uses calls to a font server to generate the raster fonts automatically. For EasyOGL, we "wrapped" the Microsoft API to the font server found in Microsoft's extended OpenGL library for Windows, *wgl*. The font commands are part of the 3D graphics widget, since the *wgl* API require a 3D graphics context to create the fonts. This is available only with a 3D widget. These are the commands for 2D text based on the *wgl* API:

- `createFont < typeface > < height > < weight > < italic > < underline > < strikethrough >`
- `deleteFont < font >`
- `putString < font > < string >`

The *createFont* command creates a raster font with the specified characteristics. The *deleteFont* deletes a font created by the *createFont* command. The *putString* command draws the specified string in the specified font at the current 2D raster location. The raster location is set with the *glRaster2d* command. See the example in section 4 and shown in figure 3. Notice the text commands are associated explicitly with a 3D graphics context called *.gl*.

3.1.4. Convenience Commands

The 3D widget provides a window for graphics functions from the OpenGL/GLU libraries to draw into. As these functions are wrapped in EasyOGL, anything possible with OpenGL/GLU can be scripted. However, for some commands the scripting code is syntactically complicated, thus, we have implemented convenience commands associated with the 3D widget. Use of these commands is optional. Most of the convenience commands originally come from Tkogl1.0 and are part of the OpenGLU library functions. These include OpenGLU commands: *cylinder*, *gencyl*, *get*, *load3ds*, *nurbssurface*, *partialdisk*, *project*, *tessellate*, *sphere* and *unproject*. (Refer to [3] for a description of each command's arguments.) The only command different from Tkogl1.0 is *texImage2D*. This command provides a convenient mechanism to pass a Tk image to the OpenGL *glTexImage2D* function.

3.2. 3D Drawing Commands (OpenGL and OpenGLU)

Creating the 3D widget provides a 3D graphics context for drawing. Any function executed from the OpenGL/GLU libraries ren-

ders in the 3D widget. We use SWIG to automatically wrap both libraries and make them into dynamically loadable packages.

SWIG is an automatic code generation program. It takes as input an interface file³ (derived from the library's header file) and generates C wrapper code which is then linked to the library (either statically or dynamically). This wrapper is responsible for binding the library's functions, variables, and constants to commands in a scripting language such as Tcl, Perl, or Python. The benefits of SWIG include rapid code development and the ability to glue different C libraries together within a scripting language.

Adding the functionality of a library of C functions (or any linkable library) to a scripting language generally requires three steps. First, the developer creates a header file containing function prototypes, variable declarations, directives, and documentation. This file is called an interface file⁴. Next, SWIG uses this interface file and the C header files along with any required additional C code to create a wrapper file. This wrapper file is more C code connecting the C library to the desired scripting language. SWIG also produces documentation in HTML and plain text that lists the function names, return types, and comments for the library functions included in the wrapper code forming the basis of documentation for EasyOGL. Finally, the wrapper code is compiled and becomes a package. Refer to [2] for details of automatically wrapping libraries.

4. EXAMPLE

Figure 2 illustrates some of the basic features of EasyOGL. This example creates a window with a lit sphere with some text and a 2D button (shown in figure 3). It depends upon a number of default settings optionally set by the developer. Refer to [4] for more complex OpenGL/GLU commands. Refer to the EasyOGL [1] distribution for additional examples. Code could be added easily to allow selecting objects in the scene, moving view points, editing the scene and other typical OpenGL/GLU functions.

5. SUMMARY AND FUTURE WORK

The Tcl/Tk scripting language and GUI toolkit provide the means to quickly and easily create 2D graphical user interfaces. The addition of the packages in EasyOGL extend the capabilities of Tcl/Tk to allow scriptable 2D/3D GUI development with 2D and 3D visualization. The EasyOGL extension includes three packages: Tkogl, OpenGL and OpenGLU. The Tkogl package provides a command to create 3D graphics widgets which behave like normal Tk widgets. These 3D graphics are drawn into the 3D widget using rendering functions from the OpenGL/GLU libraries. The OpenGL and the OpenGLU packages are Tcl embeddings of the C based OpenGL and OpenGLU libraries. Thus, the full set of library functions are accessible from the Tcl command line, and draw into the current 3D graphics widget.

For portability, we plan to support Unix/Linux platforms, including access to the font server, to simplify adding text to 3D scenes. Further, we plan to port the extensions to other scripting languages that use Tk, such as Perl and Python. We also plan to embed audio and video players inside of Tcl/Tk. This will allow

³SWIG may also need some additional C code for input.

⁴For OpenGL and OpenGLU the header files are used as the interface file.

```

package require Tkogl
package require OpenGL
package require OpenGLU

pack [OGLwin .gl -doublebuffer 1 -refresh draw]
pack [button .b -text "PUSH ME!"]

set font [.gl createFont Times 28 700 1 0 0]

proc draw {} {
    global font
    glClear GL_COLOR_BUFFER_BIT
    glClear GL_DEPTH_BUFFER_BIT
    gluSphere [gluNewQuadric] 0.75 20 20
    glRasterPos3f -0.8 -0.6 -0.75
    .gl putString $font "Hello World!"
    .gl swapBuffer
}

glEnable GL_LIGHT0
glEnable GL_LIGHTING
glEnable GL_DEPTH_TEST

glLightfv GL_LIGHT0 GL_AMBIENT [newfv4 1 1 1 1]
glLightfv GL_LIGHT0 GL_SPECULAR [newfv4 1 1 1 1]
glLightfv GL_LIGHT0 GL_POSITION [newfv4 1 1 -1 0]

```

Figure 2: Sample EasyOGL code for drawing a sphere with 2D text and a 2D button widget



Figure 3: Resulting drawing from the sample code.

us to create multimedia tutorials for teaching 3D concepts. We intend to use EasyOGL to create a 3D drag and drop demonstration builder for Nuclear Magnetic Resonance (NMR) lecturers. Learning NMR techniques requires significant understanding of vector rotations in 3 dimensions. Often, a significant understanding 3D vector rotations is difficult to acquire when learning NMR concepts. However, having 3D multimedia demonstrations of NMR phenomena that can be easily created by NMR lecturers will be a great asset. Finally, we will make EasyOGL a plug-in so it is web-accessible.

Tcl/Tk with the EasyOGL extension provides an easy to use, extensible platform to facilitate fast prototyping 3D multimedia interfaces.

6. ACKNOWLEDGEMENTS

The authors thank contributors to this project: Joel Joyner, Simon Sun, Behnoosh Rahmatian, and the many other students who worked on the project.

7. REFERENCES

- [1] EasyOGL available at: <http://www.ece.ubc.ca/~hct/easyogl>, 2001.
- [2] D. M. Beazley. Swig: An easy to use tool for integrating scripting languages with C and C++. In *Proceedings of Tcl/Tk Workshop, Monterey, CA*, July 6-10, 1996.
- [3] C. Esperanca. A tk opengl widget. In *USENIX 5th Annual Tcl/Tk Workshop*, July 1997.
- [4] M. Woo et. al. *OpenGL(r) 1.2 Programming Guide, Third Edition*. Addison-Wesley, New York, 1999.
- [5] S. S. Fels, A. Bruderlin, S. Esser, and K. Mase. Inventcl: Interpretive 3d graphics using open inventor and Tcl/[incr Tcl]. In *USENIX'97 Tcl/Tk Workshop Proceedings*, pages 185–186, Jul 1997.
- [6] C. Geiger, V. Paelke, and W. Rosenbach. Design of interactive 3d illustrations using ishells. <http://www.c-lab.de/vis/software/ishells/>, 2000.
- [7] I. Hsu. Tksm a mesa/opengl 3d modeling widget extension for tcl 7.[45]/tk. In <http://sal.kachinatech.com/E/0/TKSM.html>, 1997.
- [8] M. Lutz. *Programming Python*. O'Reilly, 1996.
- [9] M. McLennan. [incr Tcl]: Object-oriented programming in Tcl. In *Proc. 1st Tcl/Tk Workshop*, University of Berkeley, CA, USA, 1993.
- [10] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, New York, 1994.
- [11] Brian Paul and Ben Bederson. Togl - a tk opengl widget. <http://togl.sourceforge.net/>, 2000.
- [12] R. Pausch, T. Burnette, A.C. Capeheart, M. Conway, D. Cosgrove, R. DeLine, J. Durbin, R. Gossweiler, S. Koga, and J. White. Alice: Rapid prototyping system for virtual reality. In *IEEE Computer Graphics and Applications*, May 1995.
- [13] L. Wall, T. Christiansen, and R. L. Swartz. *Programming Perl 2nd Edition*. O'Reilly, 1996.