TkOGL – Integrating Tcl and Open GL

Version 3.0

Submitted by:

Joel Joyner

December 1, 2000

Submitted to:

Dr. Sidney Fels

Supervisor, EECE 496

Professor, Department of Electrical and Computer Engineering

UBC

ABSTRACT

This document discuses the revisions made to TkOGL and its features. Specifically, it provides a brief introduction to Tcl, Open GL, and SWIG, it discuses the revisions made to TkOGL, it provides a summary of the method of implementation of this tool, and acts as a guide to its usage. TkOGL is a library function that links itself to the Tcl scripting language, and provides a means to access two- and three-dimensional Open GL graphics.

Tcl is a scripting language that allows the user to create custom user interfaces simply by creating a plain text file that contains a list of Tcl strings. These strings represent a list of commands and arguments to the Tcl interpreter. Open GL is a power graphics-rendering engine developed by Silicon Graphics and is used in many applications worldwide. SWIG is a code automation tool that automatically generates wrapper code. When this wrapper code is compiled, it generates a library that allows the functions it wraps to be accessed within Tcl.

TkOGL allows Open GL graphics to be used in Tcl by providing a canvas upon which these graphics are displayed. It does this through the OGLwin command. Each of the Open GL commands has been linked separately into the Tcl language and thus, is accessed in the same way as any other Tcl command. TkOGL also provides other functionality that is accessed through the Open GL canvas.

Since it's original conception, TkOGL has undergone a several revisions. The first set of revisions included adding support for multiple Open GL windows, adding a command that allows the user to explicitly swap buffers in double buffer mode, and eliminating the need for display lists by providing a separate command for each Open GL function. The next set of revisions involved adding support for drawing fonts in the Open GL canvas, separating the libraries for the Open GL and Open GLU functions, and removing the unnecessary code from TkOGL.

Using Open GL graphics in a custom graphical user interface is a great way to spruce up an otherwise dull application. The TkOGL library provides a simple and fast way to add Open GL graphics to an application, without the hassle that would normally be accompanied with utilizing this powerful graphics engine. Hopefully, TkOGL will soon be used a variety of applications.

TABLE OF CONTENTS

LIST OF TABLES AND ILLUSTRATIONS

LIST OF ABBREVIATIONS

API	Application Programming Interface
GUI	Graphical User Interface
GL	Graphics Library
GLU	Graphics Library Utility
HTML	Hypertext Markup Language
OpenGL	Open Graphics Library
Perl	Practical Extraction and Report Language
SWIG	Simplified Wrapper and Interface Generator
Tcl	Tool Command Language
Tk	Toolkit
TkOGL	Tk Open GL Widget

1. INTRODUCTION

The purpose of this document is to discuss the revisions made to TkOGL and to provide a summary of its features. TkOGL is an extension to the Tcl scripting language that provides a means to access the powerful open GL library within a Tcl script. Thus, through TkOGL, the user is able to include two and three-dimensional graphics in a custom application through an easy to use interface.

This document will provide a brief introduction to the world of open GL and Tcl before moving on to the details of TkOGL. It will discuss the details of the usage, internal functioning, and implementation of TkOGL. Although a brief introduction to Tcl and open GL is included, readers who lack basic knowledge of those topics are encouraged to review additional material before reading this document.

The following conventions will be observed in this document. Text that is part of a script or code is written in `mono-spaced courier` font. If part of a script or code listing must be replaced with something else, it will be surrounded in angled brackets and italicized, like this: `set <variable-name> 8`.

This document begins by providing a brief overview of the Tcl scripting language, Open GL, and the SWIG code automation tool. Next the features and revision history of TkOGL are discussed. The implementation details of the TkOGL library are discussed next. Finally, the usage of TkOGL is examined before the conclusion of the document.

2. TCL, TK, OPEN GL, AND SWIG FUNDAMENTALS

This section will provide the reader with a basic overview of the Tcl scripting language and the Open GL graphics library. In addition, it introduces a tool known as SWIG, which was used to link the Open GL library with Tcl. Readers who are already familiar with these topics may wish to skim through this section or skip it entirely.

2.1 Tcl/Tk

Tcl/Tk is a scripting language, created by John K. Ousterhout at the University of California, Berkley in the early 1990s. It enables its users to design and develop custom Graphical User Interfaces (GUI's) via a simple, easy to learn scripting language. It is comprised of two components: the Tool Command Language (Tcl), which is the scripting language and its interpreter, and the Tool Kit (Tk), which is a widget generating toolkit.

2.1.1 Tcl

Tcl is a convenient scripting language that allows developers to easily test and debug their applications without the need to re-compile the application. It does this by interpreting the script instead of compiling it. A given Tcl script is composed of strings separated by new-line characters. The first word of every string is interpreted as the command to be invoked, and the remainder of the string is assumed to be a space separated list of arguments to pass to that command.

One of the best features of Tcl is that it is extendible. It allows users to add new functions to the Tcl scripting language, simply by loading a library. Tcl provides C interface functions to allow developers to easily integrate new commands into the Tcl language. One of the extensions to the Tcl language is the Tool Kit (Tk).

2.1.2 Tk

The Tool Kit (Tk) is a set of additional commands to the Tcl language that allow users to create graphical interface elements, such as buttons and scrollbars. These graphical interface elements are known as widgets. A geometry manager places these widgets in a window. This geometry manager is responsible for arranging the widgets in the parent window according to the parameters of the widget, the properties of the master window, and other rules depending upon the platform being used. Invoking the geometry manager is referred to as ***packing*** a widget. Tk also allows the user to define the action to be performed when the user acts upon a widget by ***binding*** the widget to some Tcl code or a procedure.

2.2 Open GL and Open GLU

The Open Graphics Library (Open GL) is an application programming interface (API) that consists of several hundred library functions that render two- and three-dimensional graphics using a Z-buffer or depth-buffer algorithm. Open GL originates from the Silicon Graphics GL library.

Instead of working with pixels, OpenGL allows its users to work with higher order primitives such as points, lines, polygons, images, and bitmaps. Rendering images with Open GL involves high-level mathematical operations. A given object is positioned in the real window after a set of transformation matrices has been applied to its world co-ordinates. The user is given the freedom to specify the window size, virtual camera position, orientation, field of view, and the position of the object to be drawn.

Each Open GL object is composed of multiple geometric primitives that are arranged and drawn to produce the final image. A technique called rasterization translates information such as vertex color, surface normal, and texture coordinates to real world pixels and co-ordinates. Open GL is able to operate on any windowing system including X Windows and Windows NT. This flexibility has made Open GL one of the foremost API's for developing portable three-dimensional graphics applications.

Open GL and Open GLU are essentially two parts of the same graphics library. They have been separated into two libraries so that the Open GLU library is not loaded if it is not needed. The difference between the two is that Open GLU is primarily used to rendering specific solid objects, such as spheres, while Open GL is used for more general graphics rendering.

2.3 SWIG

The Simplified Wrapper and Interface Generator (SWIG) is a code automation program. It takes in an interface file and some C code as input and generates C wrapper code. This wrapper is responsible for binding the user's C functions, variables, and constants to a scripting language such as Tcl, Perl, or Python. The benefits of SWIG include rapid code development and the ability to glue several different C libraries together with a scripting language.

Adding the functionality of C code to a scripting language requires three steps. First the user creates a header file containing function prototypes, variable declarations, directives, and documentation. This file is called an interface file. Next, SWIG uses this interface file and the C header files for the necessary C code to create a wrapper file. This wrapper file is more C code that connects the C code 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. Finally, the wrapper code is compiled, and a library is produced.

3. FEATURES AND HISTORY OF TKOGL

TkOGL is a library that provides a platform for rendering Open GL graphics in a Tcl scripting environment. TkOGL allows users to create three-dimensional graphic renderings more quickly by including Open GL commands in a Tcl. Furthermore, TkOGL provides a high-level abstraction freeing the user from the details of embedding Open GL into a windowing system.

TkOGL accomplishes these tasks primarily through the OGLwin command. This command creates an Open GL canvas for the user. The results of any Open GL commands are displayed upon this canvas. In addition, this library includes a helpful toolbox of commands that can be passed to this canvas, both upon creation, and during rendering. Examples of these functions include a utility to print text on the canvas, and to request an immediate refresh of the Open GL window.

3.1 Package Overview

As already discussed, Tcl and Open GL have been packed together through the SWIG tool and the TkOGL library. This was accomplished by extending the Tcl language to include commands that allow access to the Open GL library, and providing a means to produce a canvas to draw these graphics upon. The figure below illustrates the hierarchy of this package.

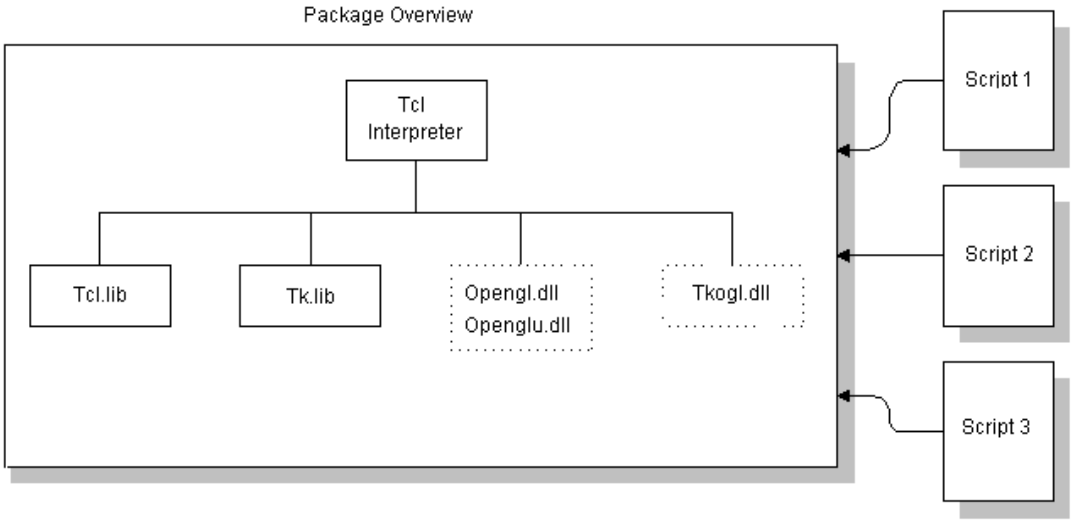

Figure 1: An overview of the entire package.

There are two methods of extending the Tcl language. One is to register a new command into the Tcl scripting language itself, and the other is to register a new command into TkOGL. Commands registered into the Tcl language can be accessed on a line by themselves, and are treated the same as any other Tcl command. Commands registered into TkOGL can only act upon a canvas, but have the advantage of much less restricted access to low level window functions and data.

The Open GL and Open GLU libraries were integrated into this package as Tcl commands. Thus, Open GL library functions may be accessed directly from Tcl. The SWIG tool automated this process by automatically generating wrappers for all the Open GL commands. Other commands have been added to the TkOGL package (and some have been removed). These commands are passed as parameters to the OGLwin widget instead of being passed to the Tcl interpreter. See section 4 for more information about the implementation of these libraries.

3.2 Changes made to TkOGL

TkOGL has undergone many revisions to satisfy its requirements. The table below provides a brief history of the changes made to TkOGL. The remainder of this sections discuses these changes in more detail.

Revision Number	Description of CHANGES
1.0	Original version provided by Claudio Esperanca
2.0	Group 4 of the second semester of the winter 1999 EECE 285 class made the following changes: • The Open GL command calling structure was changed so that the Open GL commands could maintain their original structure. • The need for display lists was eliminated • Support for multiple TkOGL windows was implemented. • Functionality was added so that users working in double buffered mode could explicitly swap buffers.
3.0	In the first semester of the winter 2000 EECE 496 class, I made the following changes: • Support for drawing fonts in the TkOGL window was added. • The `opengl.dll` was divided into two libraries. `Opendl.dll` and `Openglu.dll`. Open GL's AUX functions were eliminated from the libraries. • Some of the code in the original TkOGL was no longer required because revisions to the TkOGL code replaced it. This code was eliminated from TkOGL.

Table 1: The History of TkOGL

3.2.1 The Origin of TkOGL

TkOGL was originally conceived by Claudio Esperança to provide a means of integrating Open GL graphics to Tcl. It accomplished this goal, but it was decided that additional features should be added, and the scripting style needed to be changed. The primary

feature of this library is the OGLwin command. As stated earlier, this command creates a canvas widget, which provides a means of displaying Open GL graphics on the screen. This widget also accepts additional commands. The table below provides a brief description of the commands that the original implementation of OGLwin accepted.

Command	**Description**
Eval	Executes a provided list of Open GL commands.
Configure	Query or modify the configuration options of the Open GL Canvas.
Mainlist	Creates a display list that contains a provided list of Open GL commands to be executed every time the Open GL canvas needs to be updated.
Newlist	Creates a display list that contains a provided list of Open GL commands.
Deletelist	Deletes a display list created by newlist.
Redraw	Forces the Open GL canvas to redraw itself.
Select	A command used to determine what objects exist within a certain provided area.
Project	Provides a means of converting from world co-ordinates into window co-ordinates.
Unproject	Provides a means of converting from window co-ordinates into world co-ordinates.
Sphere	Renders a sphere.
Partialdisk	Renders a partial disk.
Disk	Renders a disk.
Cylinder	Renders a cylinder.
Nurbssurface	Creates a complex curved surface, and returns a display list containing the Open GL instructions needed to generate this surface.
Tessellate	Creates a complex polygon, and returns a display list containing the Open GL instructions needed to generate this polygon.

Table 2: A Summary of the Original TkOGL Commands.

Each one of the commands above requires additional parameters to perform its tasks. Additional information about these commands can be found in the TkOGL documentation and in section 5 of this document.

3.2.2 The Next Revision of TkOGL

Group 4 of the second semester of the winter 1999 EECE 285 class was charged with the task of adding additional features into TkOGL and revising the Open GL command structure in Tcl. This group included Henry Chan, Geoffrey Clark, Frank Ho, Erin Lim, Ivan Sun, Nicholas Tsang, Herbert Wong, and Kin Yeung.

Specifically, they changed the syntax of the Open GL commands so that it more closely matched the original Open GL syntax, eliminated the need for display lists, and added a means to manage multiple Open GL canvases. Using the SWIG tool, they created wrapper code that registered each Open GL command as an individual command in Tcl. In the original version of TkOGL, Open GL commands were only accessible via TkOGL display lists. They also created the refresh, makeCurrent and swapBuffer commands. These commands are discussed in more detail in section 5.

3.2.3 The Current Version of TkOGL

At the time of writing of this document, the current version of TkOGL is version 2.3. This version was enhanced with the ability to draw text on the Open GL canvas. Any font that is installed on the system can be used. The commands to create fonts and display text are discussed in more detail in section 5. In addition, the mainlist, newlist, deletelist, eval, and select commands were removed from TkOGL, in favour of the new Open GL command structure.

4. IMPLEMENTATION DETAILS

This section provides a description of how the `opengl.dll`, `openglu.dll`, and `tkogl.dll` libraries were built. These libraries are dynamic link libraries. A script loads them at run time if they are needed.

4.1 The Open GL and Open GLU libraries

The Open GL and Open GLU libraries were built using very similar techniques and tools. The SWIG automatic wrapper generation tool did almost all of the work. It even came with an example that provided the basis for the implementation of `opengl.dll` and `openglu.dll`. Example interface files provided with SWIG were modified so that the Open GL and Open GLU libraries would be separated.

Next, the interface files, and the required header file, `gl.h`, were used to generate the wrapper code. This wrapper code essentially acts as in intermediary in between the Tcl scripting language and the Open GL libraries. It checks and reformats the parameters passed to it, calls the Open GL command, and reformats and passes the output back to the Tcl interpreter. The wrapper code was then compiled with the *Microsoft Visual C++ Professional Edition, Version 6.0 compiler*. This process was the same for both the Open GL and Open GLU libraries, with the exception that the interface files were different. The diagram below illustrates this.

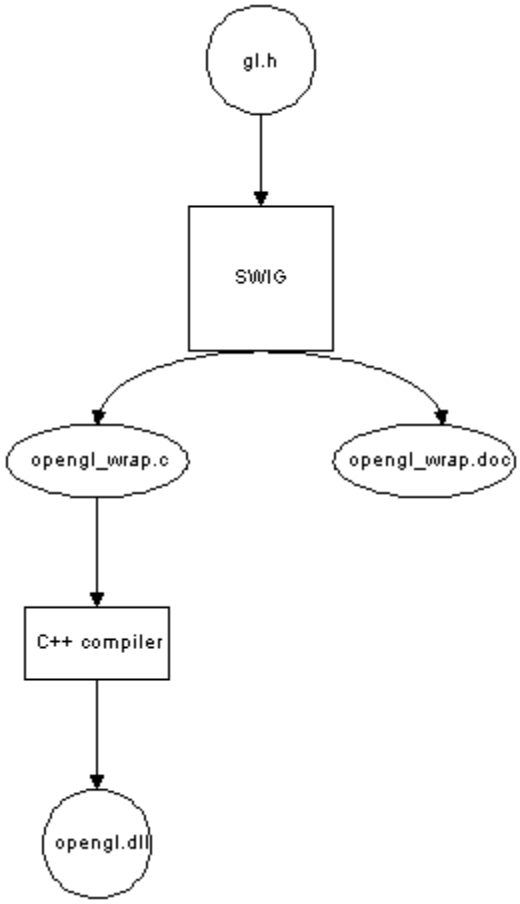

Figure 2: How the Open GL and Open GLU libraries were built

4.2 The TkOGL library

The `tkogl.dll` dynamic link library was created by compiling the necessary C source code and header files using the *Microsoft Visual C++ Professional Edition, Version 6.0 compiler*. The original source files came from Claudio Esperança's original version of TkOGL. They were modified to include the revision discussed in section 3 and compiled.

In addition to `tkogl.dll`, the compile process also produced `pkgIndex.tcl`. This file allows the user to load the TkOGL library simply by invoking the command `package require Tkogl`, when the `tkogl.dll` and `pkgIndex.tcl` files are placed in the Tk library sub-directory of the Tcl directory. The diagram below illustrates this.

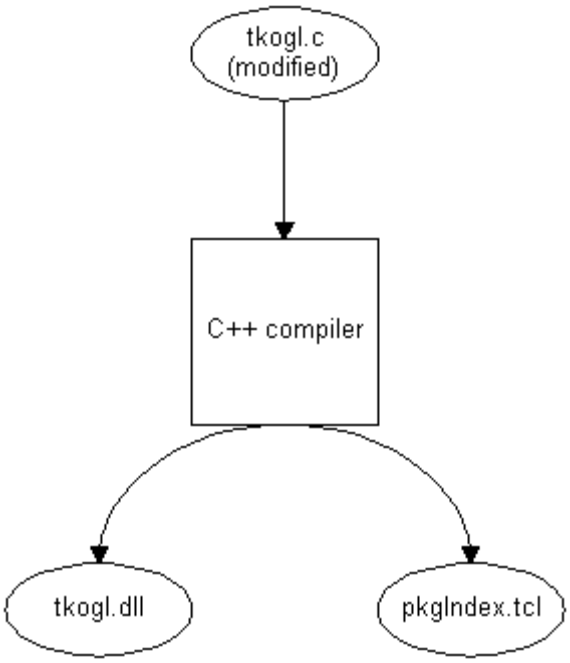

Figure 3: How the TkOGL library was built.

5. TKOGL AND OPEN GL USAGE

This section describes how to use TkOGL and Open GL in Tcl scripts. Calling the name of the command as the first word in the Tcl string, and specifying the parameters on the rest of the string executes Open GL library functions. The TkOGL library provides the OGLwin command, which creates the canvas for drawing Open GL graphics. Other commands can be passed to the Open GL canvas to perform other functions.

Although this section provides some of the basic information needed to use Open GL and TkOGL, it is not intended as a reference to Open GL or Tcl. Readers are encouraged to seek additional references for this material. See the list of references at the end of this report for a list of suggested references.

5.1 Loading the libraries

To load the Open GL, Open GLU, and TkOGL libraries, the header of the script must contain these entries:

```
Package require Tkogl
Load “opengl.dll” opengl
Load “openglu.dll” openglu
```

If any one of the libraries is not required, the entry to load that library may be omitted. The libraries should be loaded in the order they are listed above.

5.2 Accessing the Open GL and Open GLU libraries

Open GL and Open GLU library functions can be accessed simply by entering the name of the library function as the first word of a Tcl string, and the parameters as the rest of the string. The syntax is as shown below:

<gl-command-name> <parameter 1> <parameter 2> …

The parameters should be entered in the same order as they would in a C source file; however, they do not need to be separated by commas, only spaces.

Some of the parameters that are passed to Open GL libraries are arrays. Thus, a method has been created to generate, populate, and read these arrays. A series of Tcl commands have been created for this purpose. The syntax of these commands is as follows:

`Array_<type> <size>`

Generates an array of type `<type>` with `<size>` elements.

`Set_<type> <array-name> <index> <value>`

Sets the element `<index>` of the array called `<array-name>` to `<value>`.

`Get_<type> <array-name> <index>`

Returns the value of the element `<index>` of the array called `<array-name>`.

The types of arrays supported are: integer (`int`), double precision floating point (`double`), floating point (`float`), byte (`byte`), and strings of characters (`string`). In addition, to simplify the creation of floating point arrays with four or few elements two additional commands have been created. These are as follows:

`Newfv4 <value-1> <value-2> <value-3> <value-4>`

Creates a new floating-point array with four elements and populates it with the provided values.

`Setfv4 <array-name> <value-1> <value-2> <value-3> <value-4>`

Sets the values of an already existing four-element floating-point array to the provided values.

These commands can be used as shortcuts to send constant valued arrays to Open GL commands. If fewer than four elements are needed, populate the remainder of the elements with zeros.

5.3 Creating a new Open GL Canvas

The OGLwin command is used to create a new Open GL canvas. The syntax of the command is as follows:

`OGLwin <path-name> <options>`

where the `<options>` are specified as follows:

- `-accumsize <accumSize>` specifies that `<accumSize>` bit planes should be supported for each of the red, green and blue components by the accumulation buffer. The same number of bit planes is also requested for the alpha component of the accumulation buffer if an alpha component is requested for the color buffer. No accumulation buffer is requested by default.
- `-alphasize <alphaSize>` specifies that `<alphaSize>` bit planes for the alpha component should be supported by the color buffer. By default, no alpha bit planes are requested.
- `-aspectratio <ratio>` forces the viewport of the window to the width/height ratio `<ratio>`. The parameter should be a positive floating-point number. The viewport will be defined as the biggest possible rectangle having the specified aspect ratio centered inside the window. By default `<ratio>` is 0.0. In this case, no aspect ratio is enforced, meaning that the viewport will always take the same shape as the window.
- `-context <pathName2>` makes the current Open GL context share display lists with that of `<pathName2>`. `<pathName2>` should be the name of another OGLwin canvas widget.
- `-depthsize <depthSize>` specifies the number of bit planes for the depth buffer or *z-buffer*. This number is 16 by default. A `<depthSize>` of 0 will indicate that no depth buffer is desired.

`-doublebuffer <doubleFlag>` specifies whether or not a double buffered visual is needed. This is true by default. A value of one is interpreted as true, and a value of zero is false.

`-height <height>` specifies the height of the window in pixels. The default is 300.

`-refresh <refresh-commands>` Specifies the procedure or list of commands that are executed every time the Open GL canvas needs to be updated. The Tcl interpreter interprets the commands specified by refresh-commands. Any commands may be used, including other Open GL commands.

`-stencilsize <stencilSize>` specifies the number of bit planes requested for the stencil buffer. This value is set to zero by default.

`-width <width>` specifies the width of the window in pixels. The default is 300.

5.4 Using the Canvas

Once an Open GL canvas has been created, commands may be issued to draw graphics or re-configure the canvas window. Regular Open GL library functions are used as described in section 5.2. The output of these commands is displayed in the current selected Open GL canvas. Additional commands registered to the TkOGL library are called as follows:

<path-name> <command-name> <parameters>

The commands may be any of the following:

Command Name: `configure`

Parameters: *<option-1> <value-1> <option-2> <value-2>* …

Description: Configure queries or modifies the configuration options of the widget. If no options are specified, it returns a list describing all of the available options for <*pathname*>. If an option is provided with no value, then the command returns a list

describing the one mentioned option. If one or more option-value pairs are specified, the command modifies the given widget option(s). Note: only the `-width` and `-height` options can be modified by the `configure` command; all others can only be specified at the time the widget was created and cannot be modified afterwards.

Command Name: `createFont`

Parameters: *<typeface> <height> <weight> <italic> <underline> <strikethrough>*

Description: This command creates a new font according to the options provided. It returns the numerical value corresponding to the newly created font. The *<typeface>* parameter is the plain text name of any font installed on the system. The *<height>* and *<weight>* parameters are integers that define the height and weight of the text. A weight of 700 or greater will make text appear bold. The *<italic>*, *<underline>*, and *<strikethrough>* parameters are simply true or false options. If any of the options are omitted, they are set to the system default, with the exception of the *<italic>*, *<underline>*, and *<strikethrough>* options, which, if omitted, are set to false.

Command Name: `cylinder`

Parameters: `-displaylist` *<dlist>* `-normals` *<normals>* `-drawstyle` *drawStyle* `-orientation` *<orientation>* `-texture` *<texture>* *<baseRadius>* *<topRadius>* *<height>* *<slices>* *<stacks>*

Description: Renders a cylinder using the GLU facilities for quadrics. By default, the rendering is compiled into a new display list. The number of the display list is returned as the result of the widget command. The `-displaylist` option can be used to specify a display list *<dlist>*. Rendering is performed immediately and no display list is generated or overwritten if *<dlist>* is set to `none`. The remaining options can be used to specify rendering styles that would otherwise be implemented

by the functions **gluQuadricNormals**, **gluQuadricDrawStyle, gluQuadricOrientation** and **gluQuadricTexture**, respectively. Lowercase strings derived from corresponding symbolic constants specify option values. For example, `-normals flat` corresponds to calling **gluQuadricNormals** with the **GLU_FLAT** argument. The <*baseRadius*>, <*topRadius*>, <*height*>, <*slices*>, and <*stacks*> parameters are required. Refer to the **gluCylinder** function for more information.

Command Name: `deleteFont`

Parameters: <*font*>

Description: Deletes a font created by the createFont command. The required <*font*> option specifies which font is to be deleted.

Command Name: `disk`

Parameters: `-displaylist <dlist> -normals <normals> -drawstyle <drawStyle> -orientation <orientation> -texture <texture> <innerRadius> <outerRadius> <slices> <loops>`

Description: Renders a disk using the GLU facilities for quadrics. The format of the options is similar to those of the cylinder command. The <*innerRadius*>, <*outerRadius*>, <*slices*>, and <*loops*> parameters are required. Refer to the **gluDisk** function for more information.

Command Name: `gencyl`

Parameters: `-cross <x> <y> <z> <x> <y> <z> ... –identity –plot -polygon <radius> <nSides> -rotate <angle> <x> <y> <z> -scale <x> <y> <z> -shade <shadeModel> -stitch <stitchStyle> -texgen <minS> <maxS> <mint> <maxT> -translate <x> <y> <z>`

Description: This command provides access to the generic cylinder extension. This extension provides a relatively straightforward way of rendering arbitrary surfaces. It's return value and parameters are fairly complicated and are not specified in this

document for sake of conciseness. Refer to the original TkOGL documentation for more detail.

Command Name: `get`

Parameters: `<info>`

Description: This command returns a specific piece of information about the Open GL canvas depending upon what information is requested. Valid parameters for `<info>` are `currentcolor`, `currentnormal`, `modelviewmatrix`, `projectionmatrix`, and `viewport`. These parameters return the current drawing color, the current normal vector, current model view mode, current projection mode, and the current viewport respectively.

Command Name: `load3ds`

Parameters: `<filename>`

Description: This command loads a 3DStudio file and pipes any triangles it may find into the GL pipeline. Its parameter is the name and path of the 3DStudio file. It returns a display list to draw the image in stored in the file.

Command Name: `makeCurrent`

Parameters: None

Description: This command permits the user to specify which Open GL canvas should be the current active window. Any OpenGL commands executed thereafter will be drawn in that OGL window.

Command Name: `nurbssurface`

Parameters: `-uknots <knot-1> <knot-2> ... -vknots <knot-1> <knot-2> ... controlpoints <coord-1> <coord-2> ... -type <type> -uorder <order> -vorder <order> -samplingtolerance <tol> -displaymode <mode> -culling <cull>`

Description: This command creates a nurbs surface as a display list. The display list number is returned as the result of the command. The `-uknots` and `-vknots` arguments are mandatory. They specify the sequence of knots in the u and v directions, respectively. The `–controlpoints` argument is also mandatory, and specifies the coordinates of the control points. The remaining parameters of **gluNurbsSurface** and rendering parameters usually set with **gluNurbsProperty** are specified by the remaining options. The default values for these options are as follows. `-type map2vertex3` (**GL_MAP2_VERTEX_3**); `-uorder 4 -vorder 4` (cubic polynomials); `-samplingtolerance 50` (pixels); `-displaymode fill` (**GLU_FILL**); `-culling no` (**GL_FALSE)**. The command takes care of computing remaining parameters such as <*uStride*> and <*vStride*> by counting the number of knot values and control point coordinates provided.

Command Name: `partialdisk`

Parameters: `-displaylist` <*dlist*> `-normals` <*normals*> `-drawstyle` <*drawStyle*> `-orientation` <*orientation*> `-texture` <*texture*> <*innerRadius*> <*outerRadius*> <*slices*> <*loops*> <*startAngle*> <*sweepAngle*>

Description: Renders a partial disk using the GLU facilities for quadrics. The format of the options is similar to those of the cylinder command. The <*innerRadius*>, <*outerRadius*>, <*slices*>, <*loops*>, <*startAngle*>, and <*sweepAngle*> are required parameters. Refer to the **gluPartialDisk** function for more information.

Command Name: `project`

Parameters: <*worldX*> <*worldY*> <*worldZ*>

Description: This command provides access to the **gluProject** OpenGL function. The <*worldX*>, <*worldY*> and <*worldZ*> parameters are the world coordinates of a point. A list containing the corresponding three window coordinates is the return

value of this command. The command takes care of the other arguments that would normally be required for **gluProject**, such as the viewport, projection and modelview matrices. It also corrects for the value for the *y* coordinate. This value is offset due to the fact that the *y*-axis runs from the bottom to the top of the screen in OpenGL, and from the top to the bottom in X windows and MS Windows.

Command Name: `putString`

Parameters: `<font> <string>`

Description: Draws a string of text specified by <*string*> on the canvas using the font specified by <*font*>. The parameters are all required.

Command Name: `redraw`

Parameters: None

Description: This command forces the window to be refreshed. First, Tcl executes the procedure defined by the refresh parameter passed to OGLwin when the canvas was created. Then, **glFlush** is called to flush all pending commands, and the front and back buffers are swapped if the image is a double-buffered visual.

Command Name: `tesselate`

Parameters: `-displaylist <dlist> -noedgeflags <x> <y> <z> ... -contour <x> <y> <z> ...`

Description: Uses GLU Tesselator to render a complex polygon. The <*x*>, <*y*> and <*z*> coordinates specify the vertices of the polygon. Different polygon boundaries or "holes" can be specified with the `-contour` option. The rendering is compiled into a new display list by default. The number of this display list is returned by the widget command. The `-displaylist` option can be used to specify a different display list <*dlist*> instead. If <*dlist*> is set to `none`, no display list is generated, and the rendering takes place immediately. Unless the option `-noedgeflags` is specified, the rendering will flag internal triangle edges. This is useful if the polygon is rendered using a `line` style.

Command Name: `texImage2D`

Parameters: `<level> <border> <image>`

Description: Passes a TK image to the Open GL **glTexImage2D** command. This command defines a 2D texture image in Open GL. The <*level*> argument specifies the texture image's level of detail. The <*border*> argument specifies the number of border pixels. The name of the image is passed to the <*image*> argument. All of the arguments are required.

Command Name: `sphere`

Parameters: `-displaylist <dlist> -normals <normals> -drawstyle <drawStyle> -orientation <orientation> -texture <texture> <radius> <slices> <stacks>`

Description: Renders a sphere using the GLU facilities for quadrics. The options are the same as those of the `cylinder` command. The <*radius*>, <*slices*>, and <*stacks*> parameters are required. Refer to the **gluSphere** function for more information.

Command Name: `swapBuffer`

Parameters: None

Description: This command allows a user who is working in double buffer mode to explicitly swap the buffers. More specifically, the back buffer of the active OGL window will be switched with the front buffer of the window once the drawing in the back buffer has been completed.

Command Name: `unproject`

Parameters: <*windowX*> <*windowY*> <*windowZ*>

Description: This command provides access to the **gluUnProject** Open GL function. The <*windowX*>, <*windowY*> and <*windowZ*> parameters are the window coordinates of a point. A list with the corresponding world coordinates is returned by

this command. The remaining values necessary to the **gluUnProject** function are obtained in the same way described for the `project` command.

5.5 An example

To clarify and summarize the information provided in this section, a simple example is provided. This example creates a window with a triangle drawn in it.

```
package require Tkogl
load "opengl.dll" opengl

pack [OGLwin .gl -refresh draw]

proc draw {} {
     glClear GL_COLOR_BUFFER_BIT
     glColor3f 1 0 0.5
     glBegin GL_TRIANGLES
          glVertex3f 0.1 0.7 0.1
          glVertex3f -0.7 -0.7 0.1
          glVertex3f 0.7 -0.7 0.1
     glEnd
}
```

The first two commands load the required libraries into the Tcl environment. A third command `load “openglu.dll” openglu` could also have been required, but was omitted because this script does not contain any Open GLU commands. The next line uses the OGLwin command to create an Open GL canvas. The refresh parameter specifies that the draw procedure is the procedure that is executed by the Tcl interpreter every time a refresh is requested. The Open GL canvas is packed in the same line.

The draw procedure itself contains the Open GL commands used to generate the triangle. These commands clear the canvas, specify the colour, and specify the vertices of the

triangle. Refer to an Open GL reference for more information about these commands. When this script is executed, the triangle appears on the screen.

6. CONCLUSION

The Tcl scripting language provides a means to quickly and easily create a graphical user interface that would have otherwise required more labour intensive coding and debugging. Through TkOGL, the powerful Open GL graphics engine can be accessed in this scripting environment. By using Tcl and Open GL together, the user can create dazzling two- and three-dimensional graphics in a custom user interface.

The primary purpose of the TkOGL library is to provide the OGLwin widget, which provides a canvas for the output of Open GL commands. This canvas accepts a variety of parameters when it is created, and additional commands after it has been created. This canvas widget also abstracts away the complicated details of drawing graphics in a windowing environment.

TkOGL has undergone several revisions since Claudio Esperança's original version of the tool. It has been updated to allow for multiple Open GL canvases, manual refresh requests and double buffer mode buffer swaps, and to allow system fonts to be used in the canvas. Since TkOGL display lists are no longer needed, some of the code was removed.

Embedding Open GL graphics in a custom graphical user interface is a sure fire way to impress end users and add life to an otherwise mundane application. TkOGL provides a simple and straightforward method of accomplishing this with minimal time and effort. With any luck, it will soon be used for many applications.

REFERENCES

The following documents and web pages were used as resources in the creation of this report.

1. Ousterhout, John K, *Tcl and the Tk Toolkit* (Berkley, CA: Addison-Wesley Publishing Company Inc., 1994).

2. David M. Beaszley, *SWIG Users' Manual, Version 1.1* (Salt Lake City Utah: The University of Utah, Department of Computer Science, 1997), http://www.swig.org.

3. Esperança, Claudio. *A Tk OpenGL Widget* 1997, http://www.usenix.org/publications/library/proceedings/tcl97/full_papers/esperanca.

4. Wright, Richard S. Jr. and Sweet, Michael, *Open GL Superbible* (Corte Madera, CA: Waite Group Press, 1996.

5. Henry Chan, Geoffrey Clark, Frank Ho, Erin Lim, Ivan Sun, Nicholas Tsang, Herbert Wong, and Kin Yeung, *Software Design Document for TkOGL, Version 2.0*, (Vancouver, BC: The University of British Columbia, Department of Computer and Electrical Engineering, 2000)