Tcl3D: Doing 3D with Tcl

1 Introduction

Tcl3D enables the 3D functionality of OpenGL and various other portable 3D libraries at the Tcl scripting level.
It's main design requirement is to wrap existing 3D libraries without modification of their header files and with minimal manual code writing. The Tcl API should be a direct mapping of the C/C++ based library API's, with a "natural" mapping of C types to according Tcl types.

This is accomplished with ***SWIG*** [12], the Simplified Wrapper and Interface Generator.

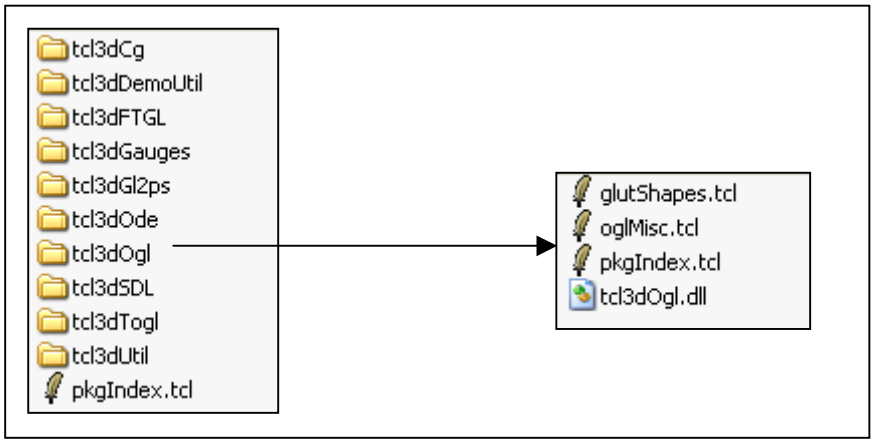
Tcl3D is based on ideas of Roger E. Critchlow, who formerly created an OpenGL Tcl binding called Frustum [2].

1.1 Architecture overview

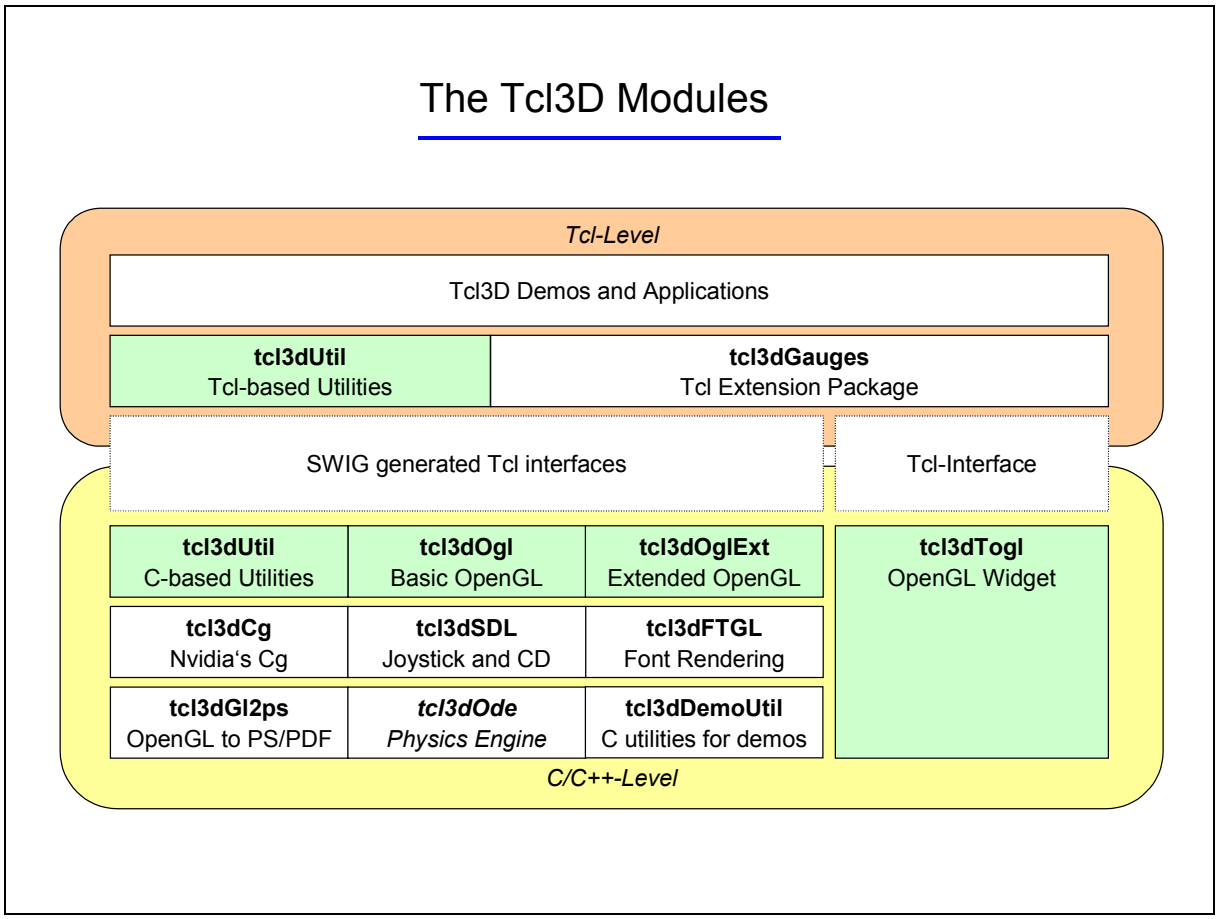
The ***Tcl3D*** package currently consists of the following building blocks, also called modules throughout the manual:

Tcl3D core modules	
tcl3dTogl	Enhanced Togl widget, a Tk widget for displaying OpenGL content.
tcl3dUtil	Tcl3D utility library (math functions, shapes, stop watch, et al).
tcl3dOgl	Wrapper for basic OpenGL functionality (GL Version 1.1, GLU Version 1.2).
Tcl3D optional modules	
tcl3dOglExt	Wrapper for enhanced OpenGL functionality (GL Version 1.2 through 2.0) and OpenGL extensions.
tcl3dCg	Wrapper for NVidia's Cg shading language.
tcl3dSDL	Wrapper for the Simple DirectMedia Library.
tcl3dFTGL	Wrapper for the OpenGL Font Rendering library.
tcl3dGl2ps	Wrapper for the OpenGL To Postscript library.
tcl3dOde	Wrapper for the Open Dynamics Engine.
tcl3dGauges	Tcl3D package for displaying gauges.
tcl3dDemoUtil	C/C++ based utility functions for some of the demo applications.

Each module is implemented as a separate Tcl package, similar to the Tcl standard library Tcllib. All Tcl3D subpackages can be loaded with a single `package require tcl3d`.



The next figure shows the currently available modules of Tcl3D.



1.2 Modules overview

This chapter gives you a short overview of the modules available in Tcl3D.

1.2.1 tcl3dTogl: Enhanced Togl widget

This module is an enhanced version of the Togl [3] widget, a Tk widget for displaying OpenGL graphics.
The following enhancements are currently implemented:

- Callback functions in Tcl.
- Better bitmap font support.
- Multisampling support.
- Swap Interval support.

A detailled description of this module can be found in chapter 4.1.

1.2.2 tcl3dUtil: Tcl3D utility library

This module implements C/C++ and Tcl utilities offering functionality needed for 3D programs. It currently contains the following submodules:

- 3D vector and transformation matrix module
- Information module
- Color names module
- Large data module (tcl3dVector)
- Image utility module

- Screen capture module
- Timing module
- 3D-model and shapes module
- Virtual trackball module

A detailled description of this module can be found in chapter 4.2.

1.2.3 tcl3dOgl: Wrapper for basic OpenGL functionality

This module wraps OpenGL functionality based on OpenGL Version 1.1, as well as the GLU library functions based on Version 1.2. This is due to the fact, that Windows still does not support newer versions of OpenGL. OpenGL features defined in newer versions have to be accessed via the OpenGL extension mechanism on Windows.
Standard shapes (box, sphere, cylinder, teapot, …) with a GLUT compatible syntax are supplied here, too.

A detailled description of this module can be found in chapter 4.3.

1.2.4 tcl3dOglExt: Wrapper for enhanced OpenGL functionality

This module wraps OpenGL functionality based on versions 1.2 till 2.0, lots of OpenGL extensions not contained in the OpenGL core, as well as Windows specific extensions. It is implemented with the help of the ***OglExt*** [24] library.
The files of this module are contained in the same directory as the basic OpenGL wrapper files for practial compilation reasons.

This is an optional module.

A detailled description of this module can be found in chapter 4.4.

1.2.5 tcl3dCg: Wrapper for NVidia's Cg shading language

This module wraps NVidia's Cg [18] shader library based on version 1.5.0015 and adds some Cg related utility procedures.

This is an optional module.

A detailled description of this module can be found in chapter 4.5.

1.2.6 tcl3dSDL: Wrapper for the Simple DirectMedia Library

This module wraps the SDL [19] library based on version 1.2.9 and adds some SDL related utility procedures.
Currently only the functions related to joystick and CD-ROM handling have been wrapped and tested.

This is an optional module.

A detailled description of this module can be found in chapter 4.6.

1.2.7 tcl3dFTGL: Wrapper for the OpenGL Font Rendering Library

This module wraps the FTGL [20] library based on version 2.1.2 and adds some FTGL related utility procedures.

The following font types are available:
- Bitmap font (2D)
- Pixmap font (2D)
- Outline font
- Polygon font
- Texture font
- Extruded font

This is an optional module.

A detailled description of this module can be found in chapter 4.7.

1.2.8 tcl3dGl2ps: Wrapper for the OpenGL To Postscript Library

This module wraps the GL2PS [22] library based on version 1.3.2 and adds some GL2PS related utility procedures.

GL2PS is a C library providing high quality vector output (PostScript, PDF, SVG) for any OpenGL application.

This is an optional module.

A detailled description of this module can be found in chapter 4.8.

1.2.9 tcl3dOde: Wrapper for the Open Dynamics Engine

This module wraps the OpenSource physics engine ODE [23] based on version 0.7 and adds some ODE related utility procedures.

This is an optional module.

Note This module is still work in progress. It's interface may change in the future.

A detailled description of this module can be found in chapter 4.9.

1.2.10 tcl3dGauges: Tcl3D package for displaying gauges

This package implements the following gauges as a pure Tcl package: airspeed, altimeter, compass, tiltmeter.

This is an optional module.

A detailled description of this module can be found in chapter 4.10.

1.2.11 tcl3dDemoUtil: C/C++ based utilities for demo applications

This package implements several C/C++ based utility functions for some of the demo applications.

This is an optional module.

A detailled description of this module can be found in chapter 4.11.

1.3 Supported platforms

The following table gives an overview, which modules are available on the supported operating systems. It also tries to give an indication on the quality of the module.

	Windows (32-bit)		Linux (32-bit)		Mac OS X (Intel)		IRIX 6.5 (n32)	
Module	**Wrap**	**Test**	**Wrap**	**Test**	**Wrap**	**Test**	**Wrap**	**Test**
tcl3dTogl	++	++	++	++	++	+	++	+
tcl3dUtil	++	++	++	++	++	++	++	+
tcl3dOgl	++	++	++	++	++	+	++	+
tcl3dOglExt	++	++	++	++	++	+	++	+
tcl3dCg	++	++	++	++	++	+	-	-
tcl3dSDL	+	++	+	++	+	0	+	+
tcl3dFTGL	++	+	++	+	++	0	++	+
tcl3dGl2ps	++	+	++	+	++	+	++	+
tcl3dOde	+	0	+	0	+	0	+	0
tcl3dGauges	++	+	++	+	++	+	++	+
tcl3dDemoUtil	++	++	++	++	++	++	++	+

Legend:

Column Wrap
- ++ Interface of module fully wrapped.
- + Interface of module partially wrapped.
- 0 Module not yet wrapped.
- - Module not available for the platform.

Column Test
- ++ Module extensively tested. No errors known.
- + Module tested. Minor errors known.
- 0 Module in work.
- - Module not available for the platform.

Short summary:
The Windows and Linux ports are supported best and are regularly tested on different hardware combinations.
On IRIX every module (except Cg, which is not available for SGI) has been wrapped and seems to be running fine, but no extensive tests are made.
The OS X port is in it's first stage, and needs another iteration of extensive testing.

1.4 Getting started

The easiest way to get started, is using a Tcl3D starpack. Starpacks for Windows, Linux, IRIX and Mac OS X (Intel based) can be downloaded from http://www.tcl3d.org/. See chapter 2 for a detailled information about all available Tcl3D packages.

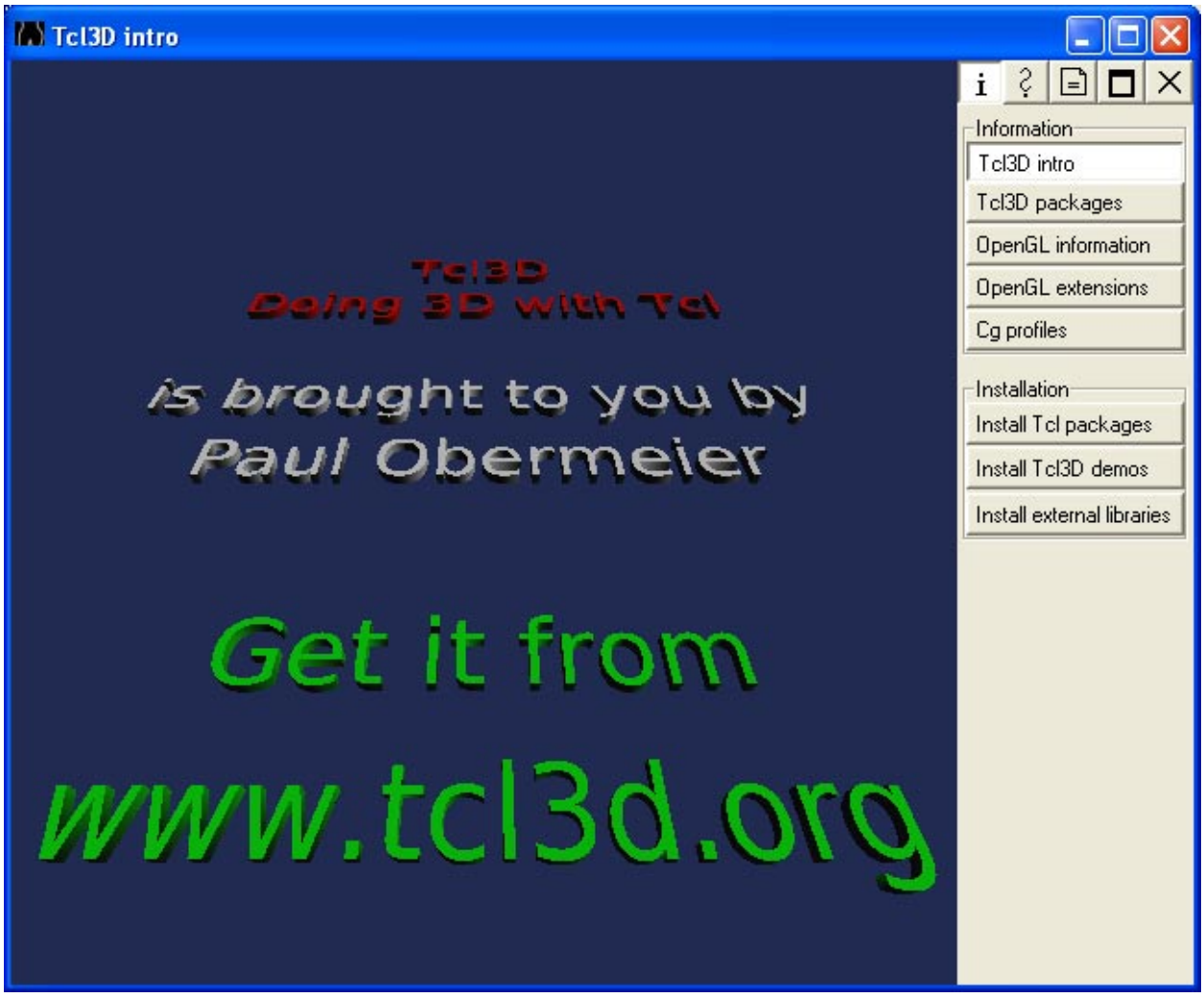
The only prerequisite needed for using the Tcl3D starpack distribution is an installed OpenGL driver. Everything else - even the Tcl interpreter - is contained in the starpack.

The starpacks are distributed as a ZIP-compressed file. Unzipping this file creates a directory containing the starpack `tcl3dsh-OS-VERSION`. Distributions for Unix systems contain an additional shell script `tcl3dsh-OS-VERSION.sh`, which should be used for starting the Tcl3D starpack.

After starting the starpack, a toplevel Tk window labeled `Tcl3D` as well as a console window labeled `Tcl3D Console` should appear, similar to starting a wish shell.
The console window should contain the following two message lines as well as a tcl3d prompt:

```
Type "pres" to start Tcl3D presentation.
Type "inst" to write the Tcl3D installation packages to disk.
tcl3d>
```

Typing `pres` in the console window, starts the Tcl3D presentation showing an introductionary animation as shown in the screenshot below. The available key and mouse bindings are shown in the console window.



Binding	Action
Key-Escape	Exit the program
Key-Left	Move text to the left
Key-Right	Move text to the right
Key-i	Increase distance from viewer
Key-d	Decrease distance from viewer
Key-Up	Increase speed
Key-Down	Decrease speed
Key-plus	Rotate text
Key-minus	Rotate text (other direction)
Key-space	Set speed of text to zero
Key-r	Reset speed and position of text
Mouse-1	Start animation

Mouse-2	Stop animation

The presentation can be started alternatively by using `-pres` as a command line parameter to the Tcl3D starpack.

Description of the Tcl3D starpack

The Starpack ***tcl3dsh*** can be used as
- a standalone executable like wish with builtin Tcl3D
- a test and presentation program for Tcl3D
- an installer for the Tcl3D specific libraries, the external libraries and demo programs

The Tcl3D presentation is divided into 3 sections:
- Information and installation
- Help and documentation
- Demos and tutorials

The information menu gives you access to different types of information (OpenGL, Tcl3D, ...), which are shown as animated OpenGL text. More detailed information can be obtained by using the ***tcl3dInfo.tcl*** script located in the demos directory in category Tcl3DSpecific.

The demo and tutorials menu has lots of sample programs, divided into 3 categories:
- ***Library specific demos*** contains scripts showing features specific to the wrapped library.
- ***Tutorials and books*** contains scripts, which have been converted from C to Tcl3D, coming from the following sources:
  OpenGL Red Book
  NeHe tutorials
  Kevin Harris CodeSampler web site
  Vahid Kazemi's GameProgrammer page
- ***Tcl3D specific demos*** contains scripts demonstrating and testing Tcl3D specific features.

Some notes about the demos contained in the Starpack:

Depending on your operating system, graphics card and driver, some of the programs may raise an error message or will not work properly.
As the demos contained within the Starpack were written to be standalone programs, no error recovery was implemented. The programs typically just quit. This is, why you may get a confirmation window from time to time, asking you, if you want to quit the show.
In most cases, you may proceed with other demos, but be warned, that strange effects may occur.

2 Installation

Precompiled packages for Windows, Linux, Intel based Mac OS X and IRIX, the source code of the Tcl3D package as well as test and demonstration programs can be retrieved from the download section of the Tcl3D home page [14].

Please report problems or errors to info@tcl3d.org.

The following distribution packages are currently available. Which packages are needed, depends on the proposed usage. See the next chapters for detailed information.

Documents	
`Tcl3D-Manual-VERSION.pdf`	Tcl3D user manual (this document).
`Tcl3D-DemoRef-VERSION.pdf`	Tcl3D demo programs reference.
Demos	
`tcl3d-demos-VERSION.zip`	Tcl3D demo sources.
`tcl3d-demoimgs-VERSION.zip`	Screenshots of all Tcl3D demo programs.
Starpacks	
`tcl3dsh-win32-VERSION.zip`	Tcl3D Starpack for Windows.
`tcl3dsh-Linux-VERSION.zip`	Tcl3D Starpack for Linux.
`tcl3dsh-Darwin-VERSION.zip`	Tcl3D Starpack for Mac OS X.
`tcl3dsh-IRIX64-VERSION.zip`	Tcl3D Starpack for SGI IRIX.
Binary packages	
`tcl3d-win32-VERSION.zip`	DLL's of external libraries and Tcl3D package for Windows.
`tcl3d-Linux-VERSION.zip`	DSO's of external libraries and Tcl3D package for Linux.
`tcl3d-Darwin-VERSION.zip`	DSO's of external libraries and Tcl3D package for Mac OS X.
`tcl3d-IRIX64-VERSION.zip`	DSO's of external libraries and Tcl3D package for SGI IRIX.
Sources	
`tcl3d-src-VERSION.zip`	Tcl3D source distribution.
`tcl3d-starpack-VERSION.zip`	Tcl3D sources for creating Starpacks.

The term VERSION is a template for the Tcl3D version number, i.e. for the currently available version it must be replaced with 0.3.2.

2.1 Installation of a binary distribution

There are two possibilities to install a Tcl3D binary distribution onto your computer.

2.1.1 Installation from a Tcl3D starpack

The following prerequisites are needed when installing from a Tcl3D starpack:

- An ***OpenGL*** driver suitable for your graphic card. I recommend to download and install an actual OpenGL driver from the manufacturer of your graphic card, especially if intending to write shader programs in GLSL or Cg.

Download, unzip and start a Tcl3D starpack presentation as described in chapter 1.4.

In the right menu pane, you will see 3 buttons in the Installation and Information menu.
These allow you to extract the Tcl3D packages, the external libraries and the demo programs onto the file system, so you can use Tcl3D from tclsh or wish.

- The Tcl3D package folder (***tcl3d0.3.2***) should be copied into the library section of your Tcl installation (ex. ***C:\Tcl\lib***). If write access to this Tcl directory is not permitted, you can copy the ***tcl3d0.3.2*** directory somewhere else, eg. ***C:\mytcl3d*** or ***/home/user/mytcl3d***. To have Tcl look for packages in this location, you must set the `TCLLIBPATH` environment variable with the above specified directory name as value. Note, that on Windows the path must be written with slashes (not backslashes): `set TCLLIBPATH = C:/mytcl3d`
- The files contained in the external libraries folder (***extlibs***) should be copied into a directory, which is listed in your PATH environment variable (Windows) or your LD_LIBRARY_PATH environment variable (Unix).
- The demonstration programs folder (***demos***) can be copied to any convenient place of your file system.

Now you are ready for using Tcl3D from standard Tcl by starting a `tclsh` or `wish` program and issuing the following command: `package require tcl3d`.

Alternatively you can extract the 3 installation folders with one of the following methods:
- Start the Tcl3D starpack and issue the command `inst` in the console.
- Start the Tcl3D starpack with command line parameter `-inst`.

Both steps will copy the 3 above described package folders into the directory containing the starpack.

2.1.2 Installation from a binary package

The following prerequisites are needed when using a Tcl3D binary package:
- An ***OpenGL*** driver suitable for your graphic card. I recommend to download and install an actual OpenGL driver from the manufacturer of your graphic card, especially if intending to write shader programs in GLSL or Cg.
- A ***Tcl/Tk*** version greater or equal to 8.4.
- The ***Img*** extension is needed to have access to various image formats, which are used as OpenGL textures.
- For some demos the ***snack*** extension is used.
- To generate screenshots from the Tcl3D presentation, the ***Twapi*** extension is needed on Windows.

I therefore recommend to use an actual ActiveTcl distribution [17], which contains all of the above listed Tcl extensions.

Download and unzip the following distribution packages suitable for your operating system:
- `tcl3d-OS-0.3.2.zip`
- `tcl3d-demos-0.3.2.zip`

Then copy the resulting folders into the appropriate directories as described in the previous chapter.

2.2 Installation of a source distribution

This chapter outlines the general process of compiling, customizing and installing the Tcl3D package. See the file ***Readme.txt*** in the source code distribution for additional up-to-date information.

2.2.1 Step 1: Prerequisites

The following prerequisites are needed when using a Tcl3D source package:

- An ***OpenGL*** driver suitable for your graphic card. I recommend to download and install an actual OpenGL driver from the manufacturer of your graphic card, especially if intending to write shader programs in GLSL or Cg.
- A ***Tcl/Tk*** version greater or equal to 8.4.
- The ***Img*** extension is needed to have access to various image formats, which are used as OpenGL textures.
- For some demos the ***snack*** extension is used.
- To generate screenshots from the Tcl3D presentation, the ***Twapi*** extension is needed on Windows.

I therefore recommend to use an actual ActiveTcl distribution [17], which contains all of the above listed Tcl extensions.

To build the Tcl3D from source, you also need the following tools installed and accessable from the command line:

Tool	Version	URL
GNU make	>= 3.79	http://www.gnu.org/
SWIG	>= 1.3.19	http://www.swig.org/

Note

- A binary version of SWIG version 1.3.24 for IRIX is available from my private home page http://www.posoft.de/.
- Tcl3D is currently generated and tested with versions 1.3.24 and 1.3.29. These versions are recommended.
- See chapter 5.4 for known bugs with other SWIG versions.

Download and unzip the following distribution packages suitable for your operating system:

- `tcl3d-src-0.3.2.zip`
- `tcl3d-OS-0.3.2.zip`
- `tcl3d-demos-0.3.2.zip`
- `tcl3d-starpack-0.3.2.zip`

Example installation procedures

Version 1: Tcl3D-Basic: OpenGL support, no external libraries

Needed: `tcl3d-src-0.3.2.zip`
Recommended: `tcl3d-demos.0.3.2.zip`

Unzip `tcl3d-src-0.3.2.zip` in a folder of your choice. This creates a new folder ***tcl3d*** containing the sources. Unzip `tcl3d-demos.0.3.2.zip` into the new folder ***tcl3d***.
If only basic OGL support is needed, comment all WRAP_* macros in file ***make.wrap***.
For extended OpenGL support, leave the macro WRAP_OGLEXT uncommented. See the chapter 2.2.3 Customization for details.
The presentation framework ***presentation.tcl*** works, but the texts are displayed as 2D bitmaps only. Most OpenGL only demos should work.

Version 2: Tcl3D-Complete: OpenGL support plus optional external libraries

Needed: `tcl3d-src-0.3.2.zip`
Needed: `tcl3d-OS-0.3.2.zip`
Recommended: `tcl3d-demos.0.3.2.zip`

Unzip `tcl3d-src-0.3.2.zip` in a folder of your choice. This creates a new folder tcl3d containing the sources. Unzip `tcl3d-demos.0.3.2.zip` into the new folder tcl3d. Unzip `tcl3d-OS-0.3.2.zip` into a temporary folder. Then copy the dynamic libraries contained in subfolder ***extlibs/OS*** into a directory, which is listed in your PATH environment variable (Windows) or your LD_LIBRARY_PATH environment variable (Unix).

If you want to build the ***tcl3dCg*** module, you have to download the Cg toolkit version 1.5.0015 from [18]. After installation, copy all the Cg header files into the ***tcl3dCg/Cg*** directory. These files are not included because of license issues. The dynamic libraries of Cg are included in the Tcl3D distribution package `tcl3d-OS-0.3.2.zip`.
If you want to wrap only a sub-set of the supported optional modules, edit the ***make.wrap*** file appropriately. See the chapter 2.2.3 Customization for details.

Version 3: Tcl3D-Star: Tcl3D-Basic or Tcl3D-Complete with Starpack support
Needed: `Installation of Version 1 or 2`
Needed: `tcl3d-starpack-0.3.2.zip`

Perform the steps as described for Version 1 or 2. Additionally copy the folder ***extlibs*** contained in distribution package `tcl3d-OS-0.3.2.zip` into the source code folder ***tcl3d***. Then unzip `tcl3d-starpack-0.3.2.zip` into the source code folder ***tcl3d***.

Note
The starpack distribution package contains Tclkits for all supported operating systems, as well as supporting Tcl packages needed for the Tcl3D demonstration programs.

2.2.2 Step 2: Configuration

Before compiling, edit the appropriate ***config_**** file to fit your platform/compiler combination:

Operating system	Compiler	Configuration file
Windows	Visual C++ 6.0, 7.1, 8.0	***config_win32***
Windows	CygWin (gcc)	***config_cygwin***
Windows	MinGW (gcc)	***config_msys***
Linux	gcc	***config_Linux***
Mac OS X	gcc	***config_Darwin***
SGI IRIX 6.5	gcc, MIPS Pro 7.3	***config_IRIX64***

Note For Unix systems, the name after the underscore is the name derived from the Unix command `uname`.

The following lines in the ***config_**** files may be edited:

`WITH_DEBUG`	If you don't want debug information, remove ALL characters after the equal sign.
`INSTDIR`	Set to your prefered installation directory.
`TCLDIR`	Set to where your Tcl installation is.
`TCLMINOR`	Set to your installed Tcl version.

Examples:

Compile with debugging information. The Tcl installation is located in ***/usr/local***. We install the Tcl3D package into the same location as the Tcl distribution. The installed Tcl version is 8.4.

```
WITH_DEBUG = 1
INSTDIR  = /usr/local
TCLDIR   = /usr/local
TCLMINOR = 4
```

Compile without debugging information. The Tcl installation is located in ***C:\Programme\Tcl***. We install the Tcl3D package into a separate directory. The installed Tcl version is 8.4.

```
WITH_DEBUG =
INSTDIR  = C:\Programme\Tcl
TCLDIR   = C:\Programme\poSoft
TCLMINOR = 4
```

Instead of editing the configuration file, you may alternatively create a file called ***make.private*** in the top level directory of Tcl3D and add lines according to your needs.

```
ifeq ($(MACHINE),win32)
INSTDIR = F:\Programme\poSoft
TCLDIR  = F:\Programme\Tcl
endif
ifeq ($(CONFIG),mingw)
INSTDIR = F:/Programme/poSoft
TCLDIR  = F:/Programme/Tcl
endif
```

2.2.3 Step 3: Customization

The optional modules can be included or excluded from the comilation step by setting the following macros in file ***make.wrap*** in the top level directory of the Tcl3D source tree.

Macro name	Description	Additional check file
WRAP_OGLEXT	Customize support for tcl3dOglExt	***OglExt/glext.h***
WRAP_CG	Customize support for tcl3dCg	***Cg/cg.h***
WRAP_SDL	Customize support for tcl3dSDL	***include/SDL.h***
WRAP_FTGL	Customize support for tcl3dFTGL	***include/FTGL.h***
WRAP_GL2PS	Customize support for tcl3dGl2ps	***gl2ps.h***
WRAP_ODE	Customize support for tcl3dOde	***ode/ode.h***

Note
Do not set a macro to 0, but comment the corresponding line (i.e. undefine), as shown in the following example:

```
WRAP_FEATURE = 1       enables the feature
# WRAP_FEATURE = 1     disables the feature
```

Each Makefile of an optional module additionally checks for the existence of an important include file (as listed in column "Additional check file") to enable extension support for Tcl3D.

2.2.4 Step 4: Compilation and installation

The following commands should compile and install the Tcl3D package:

```
> gmake
> gmake install
```

The make process prints out lines about the success of wrapping optional modules:

```
Tcl3D built with Cg support
Tcl3D built without ODE support
...
```

The starpack is not generated by default. If you installed the starpack distribution package, you have to go into the directory ***starpack*** and call `make` there.

Note
To test the generated starpack, copy it into a temporary directory and start it from there.

First installation tests

Start a ***tclsh*** or ***wish*** shell and type `package require tcl3d`.
Use the procedures `tcl3dShowPackageInfo` for a graphical package information or `tcl3dGetPackageInfo` for textual package information.

If these procedures fails, you may try the low level information supplied in the Tcl array `__tcl3dPkgInfo`:

```
> parray __tcl3dPkgInfo
__tcl3dPkgInfo(tcl3dcg,avail)         = 0
__tcl3dPkgInfo(tcl3dcg,version)       = Cg library not wrapped
__tcl3dPkgInfo(tcl3ddemoutil,avail)   = 1
__tcl3dPkgInfo(tcl3ddemoutil,version) = 0.3.2
```

Version ***Tcl3D-Basic*** should print out the following lines, when calling `tcl3dGetPackageInfo`:

```
{tcl3dcg 0 {Cg library not wrapped} {}}
{tcl3ddemoutil 1 0.3.2 {}}
{tcl3dftgl 0 {FTGL library not wrapped} {}}
{tcl3dgauges 1 0.3.2 {}}
{tcl3dgl2ps 0 {gl2ps library not wrapped} {}}
{tcl3dode 0 {ODE library not wrapped} {}}
{tcl3dogl 1 0.3.2 {}}
{tcl3dsdl 0 {SDL library not wrapped} {}}
{tcl3dtogl 1 0.3.2 {}} {tcl3dutil 1 0.3.2 {}}
```

Version ***Tcl3D-Complete*** should print out the following lines, when calling `tcl3dGetPackageInfo`:

```
{tcl3dcg 1 0.3.2 1.5.0015}
{tcl3ddemoutil 1 0.3.2 {}}
{tcl3dftgl 1 0.3.2 2.1.2}
{tcl3dgauges 1 0.3.2 {}}
{tcl3dgl2ps 1 0.3.2 1.3.2}
{tcl3dode 1 0.3.2 0.7.0}
{tcl3dogl 1 0.3.2 {}}
{tcl3dsdl 1 0.3.2 1.2.9}
{tcl3dtogl 1 0.3.2 {}}
{tcl3dutil 1 0.3.2 {}}
```

2.3 Extending Tcl3D

TODO

2.3.1 Extending with a Tcl utility

2.3.2 Extending with a C/C++ utility

2.3.3 Extending with a newer version of an external library

2.3.4 Extending with a new external library

3 Wrapping in detail

This chapter explains, how parameters and return values of the C and C++-based library functions are mapped to Tcl command parameters and return values. The intention of the wrapping mechanism was to be as close to the C interface and use Tcl standard types wherever possible:

- C functions are mapped to Tcl commands.
- C constants are mapped to Tcl global variables.
- Some C enumerations are mapped to Tcl global variables and are inserted into a Tcl hash table for lookup by name.

3.1 Wrapping description

Conventions used in this chapter:

- Every type of parameter is explained with a typical example from the OpenGL wrapping.
- The notation TYPE stands for any scalar value (char, int, float, enum etc. as well as inherited scalar types like GLboolean, GLint, GLfloat, etc.). It is **not** used for type void or GLvoid.
- The notation `STRUCT` stands for any C struct.
- The decision how to map C to Tcl types was mainly inspired to fit the needs of the OpenGL library best. The same conventions are used for the optional modules, too.

3.1.1 Scalar input parameters

The mapping of most scalar types is handled by SWIG standard typemaps.

Scalar types as function input parameter must be supplied as numerical value.

Input parameter	**TYPE**
C declaration	`void glTranslatef (GLfloat x, GLfloat y, GLfloat z);`
C example	`glTranslatef (1.0, 2.0, 3.0);` `glTranslatef (x, y, z);`
Tcl example	`glTranslatef 1.0 2.0 3.0` `glTranslatef $x $y $z`

The mapping of the following enumerations is handled differently (see file ***tcl3dConstHash.i***). They can be specified either as numerical value like the other scalar types, or additionally as a name identical to the enumeration name.

- `GLboolean`
- `GLenum`
- `GLbitfield`
- `CGenum`
- `CGGLenum`
- `CGprofile`
- `CGtype`
- `CGresource`
- `CGerror`

The mapping is explained using the 3 OpenGL types. The Cg types are handled accordingly.

`GLenum` as function input parameter can be supplied as numerical value or as name.

Input parameter	GLenum
C declaration	`void glEnable (GLenum cap);`
C example	`glEnable (GL_BLEND);`
Tcl example	`glEnable GL_BLEND` `glEnable $::GL_BLEND`

`GLbitfield` as function input parameter can be supplied as numerical value or as name.

Note
A combination of bit masks has to be specified as a numerical value like this:
`glClear [expr $::GL_COLOR_BUFFER_BIT | $::GL_DEPTH_BUFFER_BIT]`

Input parameter	GLbitfield
C declaration	`void glClear (GLbitfield mask);`
C example	`glClear (GL_COLOR_BUFFER_BIT);`
Tcl example	`glClear GL_COLOR_BUFFER_BIT` `glClear $::GL_COLOR_BUFFER_BIT`

`GLboolean` as function input parameter can be supplied as numerical value or as name.

Input parameter	GLboolean
C declaration	`void glEdgeFlag (GLboolean flag);`
C example	`glEdgeFlag (GL_TRUE);`
Tcl example	`glEdgeFlag GL_TRUE` `glEdgeFlag $::GL_TRUE`

3.1.2 Pointer input parameters

The mapping of `const TYPE` pointers is handled in file ***tcl3dPointer.i***.

Constant pointers as function input parameter must be supplied as a Tcl list.

Input parameter	const TYPE[SIZE], const TYPE *
C declaration	`void glMaterialfv (GLenum face, GLenum pname,` `const GLfloat *params);`
C example	`GLfloat mat_diffuse = { 0.7, 0.7, 0.7, 1.0 };` `glMaterialfv (GL_FRONT, GL_DIFFUSE, mat_diffuse) ;`
Tcl example	`set mat_diffuse { 0.7 0.7 0.7 1.0 }` `glMaterialfv GL_FRONT GL_DIFFUSE $mat_diffuse`

Note

- This type of parameter is typically used to specify small vectors (2D, 3D and 4D) as well as control points for NURBS.
- Unlike in the C version, specifying data with the scalar version of a function (ex. `glVertex3f`) is faster than the vector version (ex. `glVertex3fv`) in Tcl.
- Tcl lists given as parameters to a Tcl3D function have to be flat, i.e. they are not allowed to contain sublists. When working with lists of lists, you have to flatten the list, before supplying it as an input parameter to a Tcl3D function. One way to do this is shown in the example below.

```
set ctrlpoints {
      {-4.0 -4.0 0.0} {-2.0 4.0 0.0}
      { 2.0 -4.0 0.0} { 4.0 4.0 0.0}
}
glMap1f GL_MAP1_VERTEX_3 0.0 1.0 3 4 [join $ctrlpoints]
```

The mapping of `const void` pointers is handled by SWIG standard typemaps.

Constant void pointers as function input parameter must be given as a pointer to a contiguous piece of memory of appropriate size.

Input parameter	const void[SIZE], const void *
C declaration	`void glVertexPointer (GLint size, GLenum type,` `  GLsizei stride, const GLvoid *ptr);`

C example	`static GLint vertices[] =`
	`  { 25,  25, 100, 325, 175, 25,`
	`  175, 325, 250,  25, 325, 325};`
	`glVertexPointer (2, GL_INT, 0, vertices);`

Tcl example	`set vertices [tcl3dVectorFromArgs GLint \`
	`  25  25  100 325  175  25 \`
	`  175 325  250  25  325 325]`
	`glVertexPointer 2 GL_INT 0 $::vertices`

Note

- The allocation of usable memory can be accomplished with the use of the `tcl3dVector` command, which is described in chapter 4.2.
- This type of parameter is typically used to supply image data or vertex arrays. See also the description of the image utility module in chapter 4.2.

3.1.3 Output parameters

The mapping of non-constant pointers is handled by the SWIG standard typemaps.

Non-constant pointers as function output parameter must be given as a pointer to a contiguous piece of memory of appropriate size (`tcl3dVector`). See note above.

Output parameter	TYPE[SIZE], void[SIZE], TYPE *, void *

C declaration	`void glGetFloatv (GLenum pname, GLfloat *params);`
	
	`void glReadPixels (GLint x, GLint y, GLsizei width,`
	`  GLsizei height, GLenum format,`
	`  GLenum type, GLvoid *pixels);`

C example	`GLfloat values[2];`
	`glGetFloatv (GL_LINE_WIDTH_GRANULARITY, values);`
	
	`GLubyte *vec = malloc (w * h * 3);`
	`glReadPixels (0, 0, w, h, GL_RGB, GL_UNSIGNED_BYTE, vec);`

Tcl example	`set values [tcl3dVector GLfloat 2]`
	`glGetFloatv GL_LINE_WIDTH_GRANULARITY $values`
	
	`set vec [tcl3dVector GLubyte [expr $w * $h * 3]]`
	`glReadPixels 0 0 $w $h GL_RGB GL_UNSIGNED_BYTE $vec`

3.1.4 Function return values

The mapping of return values is handled by the SWIG standard typemaps.

Scalar return values are returned as the numerical value.
Pointer to structs are returned with the standard SWIG mechanism of encoding the pointer in an ASCII string.

Function return	TYPE, STRUCT *
C declaration	**GLuint** glGenLists (GLsizei range); **GLUnurbs*** gluNewNurbsRenderer (void);
C example	GLuint sphereList = glGenLists(1); GLUnurbsObj *theNurb = gluNewNurbsRenderer(); gluNurbsProperty (theNurb, GLU_SAMPLING_TOLERANCE, 25.0);
Tcl example	set sphereList [glGenLists 1] set theNurb [gluNewNurbsRenderer] gluNurbsProperty $theNurb GLU_SAMPLING_TOLERANCE 25.0

The next lines show an example of SWIG's pointer encoding:

```
% set theNurb [gluNewNurbsRenderer]
% puts $theNurb
_10fa1500_p_GLUnurbs
```

The returned name can only be used in functions expecting a pointer to the appropriate struct.

3.1.5 Exceptions from the standard rules

The GLU library as specified in header file ***glu.h*** does not provide an API, that is using the `const` specifier as consistent as the GL core library. So one class of function parameters (`TYPE*`) is handled differently with GLU functions. Arguments of type `TYPE*` are used both as input and output parameters in the C version. In GLU 1.2 most functions use this type as input parameter. Only two functions use this type as an output parameter.
So for GLU functions there is the exception, that `TYPE*` is considered an input parameter and therefore is wrapped as a Tcl list.

Input parameter	TYPE * (GLU only)
C declaration	void gluNurbsCurve (GLUnurbs *nobj, GLint nknots, **GLfloat *knot**, GLint stride, **GLfloat *ctlarray**, GLint order, GLenum type);
C example	GLfloat curvePt[4][2] = {{0.25, 0.5}, {0.25, 0.75}, {0.75, 0.75}, {0.75, 0.5}}; GLfloat curveKnots[8] = {0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0}; gluNurbsCurve (theNurb, 8, curveKnots, 2, &curvePt[0][0], 4, GLU_MAP1_TRIM_2);
Tcl example	set curvePt {0.25 0.5 0.25 0.75 0.75 0.75 0.75 0.5} set curveKnots {0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0} gluNurbsCurve $theNurb 8 $curveKnots 2 $curvePt 4 GLU_MAP1_TRIM_2

The two aforementioned functions, which provide output parameters with `TYPE*` are `gluProject` and `gluUnProject`. These are handled as a special case in the SWIG interface file ***glu.i***. The 3 output parameters are given the keyword OUTPUT, so SWIG handles them in a special way: SWIG builds a list consisting of the normal function return value, and all parameters marked with that keyword. This list will be the return value of the corresponding Tcl command.

Definition in glu.h	Redefinition in SWIG interface file glu.i
extern GLint gluUnProject (GLdouble winX, GLdouble winY, GLdouble winZ,	GLint gluUnProject (GLdouble winX, GLdouble winY, GLdouble winZ,

const GLdouble *model, const GLdouble *proj, const GLint *view, GLdouble* objX, GLdouble* objY, GLdouble* objZ);	const GLdouble *model, const GLdouble *proj, const GLint *view, GLdouble* OUTPUT, GLdouble* OUTPUT, GLdouble* OUTPUT);

Example usage (see Redbook example ***unproject.tcl*** for complete code):

```
glGetIntegerv GL_VIEWPORT $viewport
glGetDoublev  GL_MODELVIEW_MATRIX  $mvmatrix
glGetDoublev  GL_PROJECTION_MATRIX $projmatrix
set viewList [tcl3dVectorToList $viewport 4]
set mvList   [tcl3dVectorToList $mvmatrix 16]
set projList [tcl3dVectorToList $projmatrix 16]

set realy [expr [$viewport get 3] - $y - 1]
set winList [gluUnProject $x $realy 0.0 $mvList $projList $viewList]
puts "gluUnProject return value: [lindex $winList 0]"
puts [format "World coords at z=0.0 are (%f, %f, %f)" \
     [lindex $winList 1] [lindex $winList 2] [lindex $winList 3]]
```

Note The above listed exceptions are only valid for the GLU library. The optional modules have not been analysed in-depth regarding the constness of parameters.

3.2 Wrapping reference card

- The notation `TYPE` stands for any scalar value (`char`, `int`, `float`, etc. as well as inherited scalar types like `GLboolean`, `GLint`, `GLfloat`, etc.). It is not used for type `void` or `GLvoid`.
- The notation `STRUCT` stands for any C struct.

C parameter type	Tcl parameter type
Input parameter	
`TYPE`	Numerical value.
`GLboolean`	Numerical value or name of constant.
`GLenum`	Numerical value or name of constant.
`GLbitfield`	Numerical value or name of constant.
`CGenum`	Numerical value or name of constant.
`CGGLenum`	Numerical value or name of constant.
`CGprofile`	Numerical value or name of constant.
`CGtype`	Numerical value or name of constant.
`CGresource`	Numerical value or name of constant.
`CGerror`	Numerical value or name of constant.
`const TYPE[SIZE]`	Tcl list.
`const TYPE *`	Tcl list.
`const void *`	tcl3dVector
Output parameter	
`TYPE *`	tcl3dVector
`void *`	tcl3dVector
Return value	
`TYPE`	Numerical value.
`STRUCT *`	SWIG encoded pointer to struct.

4 Modules in detail

This chapter explains in detail the different modules, Tcl3D is currently built upon:

- tcl3dTogl: Enhanced Togl widget
- tcl3dUtil: Tcl3D utility library
- tcl3dOgl: Wrapper for basic OpenGL functionality
- tcl3dOglExt: Wrapper for enhanced OpenGL functionality
- tcl3dCg: Wrapper for NVidia's Cg shading language
- tcl3dSDL: Wrapper for the Simple DirectMedia Library
- tcl3dFTGL: Wrapper for the OpenGL Font Rendering Library
- tcl3dGl2ps: Wrapper for the OpenGL To Postscript Library
- tcl3dOde: Wrapper for the Open Dynamics Engine
- tcl3dGauges: Tcl3D package for displaying gauges
- tcl3dDemoUtil: C/C++ based utilities for demo applications

4.1 tcl3dTogl: Enhanced Togl widget

Togl [3] is a Tk widget with support to display OpenGL graphics. The original version only supported issuing drawing commands in C. To be usable from the Tcl level, it has been extended with configuration options for specifying Tcl callback commands: ***tcl3dTogl***.

Requirements for this module: None, all files are contained in the Tcl3D distribution.

4.1.1 Togl commands

The following is a list of currently available Togl commands. The commands changed or new in Tcl3D are marked bold and explained in detail below. For a description of the other commands see the original Togl documentation.

```
configure
render
swapbuffers
makecurrent
postredisplay
loadbitmapfont
unloadbitmapfont
```

Bitmap fonts

Specifying bitmap fonts can be accomplished with the `loadbitmapfont` command.
The font can either be specified in XLFD format or Tk-like with the following options:

```
-family courier|times|...
-weight medium|bold
-slant  regular|italic
-size   PixelSize
```

Examples:

```
$toglwin loadbitmapfont -*-courier-bold-r-*-*-10-*-*-*-*-*-*-*
$toglwin loadbitmapfont -family fixed -size 12 -weight medium -slant regular
```

See the ***tcl3dToglFonts.tcl*** and ***tcl3dFont.tcl*** demos for more examples, on how to use fonts with Togl.

4.1.2 Togl options

The following is a list of currently available Togl options. The options changed or new in Tcl3D are marked bold and explained in detail below. For a description of the other options see the original Togl documentation.

-height	-width	-setgrid	
-rgba	-redsize	-greensize	-bluesize
-double	-depth	-depthsize	-accum
-accumredsize	-accumgreensize	-accumbluesize	-accumalphasize
-alpha	-alphasize	-stencil	-stencilsize
-auxbuffers	-privatecmap	-overlay	-stereo
-cursor	-time	-sharelist	-sharecontext
-ident	-indirect	-pixelformat	
-swapinterval	**-multisamplebuffers**	**-multisamplesamples**	
-createproc	**-displayproc**	**-reshapeproc**	

These configuration options behave like standard Tcl options and can be queried as such:

```
% package require tcl3d ; # or just package require tcl3dtogl
0.3.2
% togl .t
% .t configure
{-height height Height 400 400} …
{-displayproc displayproc Displayproc {} {}} …
% .t configure -displayproc tclDisplayFunc
% .t configure -displayproc
-displayproc displayproc Displayproc {} tclDisplayFunc
```

Callback procedures

To be usable from the Tcl level, it has been extended to support 3 new configuration options for specifying Tcl callback procedures:

-createproc TclCommandName	Called when a new widget is created.
-reshapeproc TclCommandName	Called when the widget's size is changed.
-displayproc TclCommandName	Called when the widget's content needs to be redrawn.

Default settings are:

```
{-createproc  createproc  Createproc  {} {}}
{-displayproc displayproc Displayproc {} {}}
{-reshapeproc reshapeproc Reshapeproc {} {}}
```

Display options

-swapinterval	Enable/disable synchronization to vertical blank signal
-multisamplebuffers	Enable/disable the multisample buffer
-multisamplesamples	Set the number of multisamples

Default settings are:

```
{-swapinterval swapInterval SwapInterval 1 1}
{-multisamplebuffers multisampleBuffers MultisampleBuffers 0 0}
{-multisamplesamples multisampleSamples MultisampleSamples 2 2}
```

Note

Multisampling was not supported by the Togl widget till version 0.3.2. If working with older version of Tcl3D, you may enabling multisampling outside of Tcl3D as follows:

With NVidia cards, you can enable multisampling under Windows via the NVidia driver GUI. Under Linux you can set the environment variable `__GL_FSAA_MODE` to 1.

4.1.3 A simple Tcl3D template

A template for a Tcl3D application looks like follows:

```
package require tcl3d

proc tclDisplayFunc { toglwin } {
    # Clear color and depth buffer
    glClear [expr $::GL_COLOR_BUFFER_BIT | $::GL_DEPTH_BUFFER_BIT]

    glLoadIdentity                  ; # Reset the current modelview matrix

    glTranslatef 0.0 0.0 -5.0       ; # Transformations
    glRotatef $::xrot 1.0 0.0 0.0
    glRotatef $::yrot 0.0 1.0 0.0
    glRotatef $::zrot 0.0 0.0 1.0

    drawGeometry                    ; # Draw the actual geometry

    $toglwin swapbuffers            ; # Swap front and back buffer
}

proc tclCreateFunc { toglwin } {
    glShadeModel GL_SMOOTH          ; # Enable smooth shading
    glClearColor 0.0 0.0 0.0 0.5    ; # Black background
    glClearDepth 1.0                ; # Depth buffer setup
    glEnable GL_DEPTH_TEST          ; # Enable depth testing
}

proc tclReshapeFunc { toglwin w h } {
    glViewport 0 0 $w $h            ; # Reset the current viewport
    glMatrixMode GL_PROJECTION      ; # Select the projection matrix
    glLoadIdentity                  ; # Reset the projection matrix

    # Calculate the aspect ratio of the window
    gluPerspective 45.0 [expr double($w)/double($h)] 0.1 100.0

    glMatrixMode GL_MODELVIEW       ; # Select the modelview matrix
    glLoadIdentity                  ; # Reset the modelview matrix
}

frame .fr
pack .fr -expand 1 -fill both
# Create a Togl widget with a depth buffer and doublebuffering enabled.
togl .fr.toglwin -width 250 -height 250 \
                 -double true -depth true \
                 -createproc  tclCreateFunc \
                 -displayproc tclDisplayFunc \
                 -reshapeproc tclReshapeFunc
grid .fr.toglwin -row 0 -column 0 -sticky news
```

Note
Option `-createproc` is not effective, when specified in the configure subcommand. It has to be specified at widget creation time.

4.2 tcl3dUtil: Tcl3D utility library

This module implements several utilities in C and Tcl offering functionality needed for 3D programs. It currently contains the following submodules:

- 3D vector and transformation matrix module
- Information module

- Color names module
- Large data module (tcl3dVector)
- Image utility module
- Screen capture module
- Timing module
- 3D-model and shapes module
- Virtual trackball module

Requirements for this module: None, all files are contained in the Tcl3D distribution.

The master SWIG file for wrapping the utility library is ***tcl3dUtil.i***.

4.2.1 3D vector and transformation matrix module

This module provides miscellaneous 3D vector and 4x4 transformation matrix functions.

Implementation files: ***tcl3dVecMath.c, tcl3dVecMath.tcl***
Header files: ***tcl3dVecMath.h***
Wrapper files: ***util.i***

Tcl command	Description
`tcl3dVec3fPrint`	Print the contents of a 3D vector onto standard output.
`tcl3dVec3fIdentity`	Fill a 3D vector with (0.0, 0.0, 0.0).
`tcl3dVec3fCopy`	Copy a 3D vector.
`tcl3dVec3fLength`	Calculate the length of a 3D vector.
`tcl3dVec3fNormalize`	Normalize a 3D vector.
`tcl3dVec3fDistance`	Calculate the distance between two 3D vectors.
`tcl3dVec3fDotProduct`	Calculate the dot product of two 3D vectors.
`tcl3dVec3fCrossProduct`	Calculate the cross product of two 3D vectors.
`tcl3dVec3fAdd`	Add two 3D vectors.
`tcl3dVec3fSubtract`	Subtract two 3D vectors.
`tcl3dVec3fScale`	Scale a 3D vector by a scalar value.

Tcl command	Description
`tcl3dMatfPrint`	Print the contents of a matrix onto standard output.
`tcl3dMatfIdentity`	Build the identity transformation matrix.
`tcl3dMatfCopy`	Copy a transformation matrix.
`tcl3dMatfTranslatev`	Build a translation matrix based on a 3D vector.
`tcl3dMatfTranslate`	Build a translation matrix based on 3 scalar values.
`tcl3dMatfRotate`	Build a rotation matrix based on angle (°) and axis.
`tcl3dMatfRotateX`	Build a rotation matrix based on angle (°) around x axis.
`tcl3dMatfRotateY`	Build a rotation matrix based on angle (°) around y axis.
`tcl3dMatfRotateZ`	Build a rotation matrix based on angle (°) around z axis.
`tcl3dMatfScalev`	Build a scale matrix based on a 3D vector.
`tcl3dMatfScale`	Build a scale matrix based on 3 scalar values.
`tcl3dMatfTransformPoint`	Transform a point by a given matrix.
`tcl3dMatfTransformVector`	Transform a 3D vector by a given matrix.
`tcl3dMatfMult`	Multiply two transformation matrices.
`tcl3dMatfInvert`	Invert a transformation matrix.
`tcl3dMatfTranspose`	Transpose a transformation matrix.

See the test programs ***matmathtest.tcl and vecmathtest.tcl*** for examples, on how to use these procedures. Also take a look at the demo program ***ogl_fps_controls.tcl*** for a real-world example.

4.2.2 Information module

This module provides miscellaneous functions for querying OpenGL related information.

Implementation files: ***tcl3dInfo.tcl***
Header files: None
Wrapper files: None

Tcl command	Description
`tcl3dHavePackage`	Check, if a Tcl package is available in a given version.
`tcl3dGetLibraryInfo`	Return the library version corresponding to supplied Tcl3D package name.
`tcl3dGetPackageInfo`	Return a list of sub-lists containing Tcl3D package information. Each sub-list contains the name of the Tcl3D sub-package, the availability flag (0 or 1), the sub-package version as well as the version of the wrapped library.
`tcl3dShowPackageInfo`	Display the version info returned by `tcl3dGetPackageInfo` in a toplevel window.
`tcl3dHaveExtension`	Check, if a given OpenGL extension is provided by the OpenGL implementation.
`tcl3dHaveCg`	Check, if the Cg library has been loaded successfully.
`tcl3dHaveSDL`	Check, if the SDL library has been loaded successfully.
`tcl3dHaveFTGL`	Check, if the FTGL library has been loaded successfully.
`tcl3dHaveGl2ps`	Check, if the GL2PS library has been loaded successfully.
`tcl3dHaveOde`	Check, if the ODE library has been loaded successfully.
`tcl3dHaveVersion`	Check, if a specific OpenGL version is available.
`tcl3dGetVersions`	Query the OpenGL library with the keys `GL_VENDOR`, `GL_RENDERER`, `GL_VERSION`, `GLU_VERSION` and return the results as a list of key-value pairs.
`tcl3dGetExtensions`	Query the OpenGL library with the keys `GL_EXTENSIONS` and `GLU_EXTENSIONS` and return the results as a list of key-value pairs.
`tcl3dGetStates`	Query all state variables of the OpenGL library and return the results as a list of sub-lists. Each sublist contains a flag indicating the sucess of the query, the querying command used, the key and the value(s).

Note
The functions `glGetString` and `gluGetString` as well as the corresponding high-level functions `tcl3dGetVersions` and `tcl3dGetExtensions` only return correct values, if a tcl3dTogl window has been created, i.e. a rendering context has been established.

Examples:

The following code snippet shows how to call `tcl3dGetVersions`.

```
foreach glInfo [tcl3dGetVersions] {
    puts "[lindex $glInfo 0]: [lindex $glInfo 1]"
}

GL_VENDOR: NVIDIA Corporation
GL_RENDERER: GeForce FX Go5600/AGP/SSE2
```

```
GL_VERSION: 1.4.0
GLU_VERSION: 1.2.2.0 Microsoft Corporation
```

The following code snippet shows how to call `tcl3dGetExtensions`.

```
foreach glInfo [tcl3dGetExtensions] {
    puts "[lindex $glInfo 0]:"
    foreach ext [lsort [lindex $glInfo 1]] {
        puts "\t$ext"
    }
}

GL_EXTENSIONS:
     GL_ARB_depth_texture
     GL_ARB_fragment_program
     GL_ARB_imaging
     …
GLU_EXTENSIONS:
     GL_EXT_bgra
```

The following code snippet shows how to call `tcl3dGetStates`.

```
foreach glState [tcl3dGetStates] {
    set msgStr "[lindex $glState 2]: [lrange $glState 3 end]"
    if { [lindex $glState 0] == 0 } {
        set tag "(Unsupported)"
    } else {
        set tag ""
    }
    append msgStr $tag
    puts $msgStr
}

GL_VERTEX_ARRAY_SIZE: 4
GL_VERTEX_ARRAY_TYPE: 5126
GL_VERTEX_ARRAY_STRIDE: 0
GL_VERTEX_ARRAY_POINTER: --(Unsupported)
GL_NORMAL_ARRAY: 0
GL_NORMAL_ARRAY_TYPE: 5126
```

See the demo program ***tcl3dInfo.tcl*** for other examples, on how to use these procedures.

4.2.3 Color names module

This module provides miscellaneous OpenGL related functions.

Implementation files:	***tcl3dColors.tcl***
Header files:	None
Wrapper files:	None

Tcl command	Description
`tcl3dGetColorNames`	Return a list of all supported Tcl color names.
`tcl3dFindColorName`	Check, if supplied color name is a valid Tcl color name.
`tcl3dName2rgb`	Convert a Tcl color specification into the corresponding OpenGL representation. OpenGL colors are returned as a list of 3 unsigned bytes: r g b
`tcl3dName2rgbf`	Convert a color specification into the corresponding OpenGL representation. OpenGL colors are returned as a list of 3 floats in the range [0..1]: r g b

`tcl3dName2rgba`	Convert a color specification into the corresponding OpenGL representation. OpenGL colors are returned as a list of 4 unsigned bytes: r g b a
`tcl3dName2rgbaf`	Convert a color specification into the corresponding OpenGL representation. OpenGL colors are returned as a list of 4 floats in the range [0..1]: r g b a

See the test program ***colorNames.tcl*** for examples, on how to use these procedures.

4.2.4 Large data module

This module provides miscellaneous functions for handling large data like images used for textures and vertex arrays.

Implementation files: ***tcl3dVector.tcl***
Header files: None
Wrapper files: ***tcl3dArrays.i, bytearray.i***

Low level access

As stated in chapter 3.1.2, some of the OpenGL functions need a pointer to a contiguous block of allocated memory. SWIG already provides a feature to automatically generate wrapper functions for allocating and freeing memory of any type. This SWIG feature `%array_functions` has been extended and replaced with 2 new SWIG commands: `%baseTypeVector` for scalar types and `%complexTypeVector` for complex types like structs. It not only creates setter and getter functions for accessing single elements of the allocated memory, but also adds functions to set ranges of the memory.

There are wrapper functions for these scalar types defined in file ***tcl3dArrays.i***:

Array of	is mapped to
`char`	`char`
`short`	`short`
`int`	`int`
`float`	`float`
`double`	`double`
`GLenum`	`unsigned int`
`GLboolean`	`unsigned char`
`GLbitfield`	`unsigned int`
`GLbyte`	`signed char`
`GLshort`	`short`
`GLint`	`int`
`GLsizei`	`int`
`GLubyte`	`unsigned char`
`GLushort`	`unsigned short`
`GLuint`	`unsigned int`
`GLfloat`	`float`
`GLclampf`	`float`
`GLdouble`	`double`
`GLclampd`	`double`
`GLchar`	`char`
`GLcharARB`	`char`

The generated wrapper code looks like this (Example shown for GLdouble):

```
static double *new_GLdouble(int nelements) {
  return (double *) calloc(nelements,sizeof(double));
}
```

```
static void delete_GLdouble(double *ary) {
  free(ary);
}

static double GLdouble_getitem(double *ary, int index) {
    return ary[index];
}

static void GLdouble_setitem(double *ary, int index, double value) {
    ary[index] = value;
}

static void GLdouble_setarray(double *ary, double value,
                              int startIndex, int len) {
    int i;
    int endIndex = startIndex + len;
    for (i=startIndex; i<endIndex; i++) {
        ary[i] = value;
    }
}

static void GLdouble_addarray(double *ary, double value,
                              int startIndex, int len) {
    int i;
    int endIndex = startIndex + len;
    for (i=startIndex; i<endIndex; i++) {
        ary[i] += (double) value;
    }
}

static void GLdouble_mularray(double *ary, double value,
                              int startIndex, int len) {
    int i;
    int endIndex = startIndex + len;
    for (i=startIndex; i<endIndex; i++) {
        ary[i] *= (double) value;
    }
}

static double *GLdouble_ind(double *ary, int incr) {
    return (ary + incr);
}
```

These low level functions are typically not used directly. They are accessible via the Tcl command `tcl3dVector`, with the exception of the `TYPE_ind` functions.
An example for the usage of `GLfloat_ind` for optimized access to vectors can be found in NeHe demo ***Lesson37.tcl***.

File ***bytearray.i*** provides the implementation and wrapper definitions to convert Tcl binary strings (ByteArrays) into Tcl3D Vectors (`tcl3dByteArray2Vector`) and vice versa (`tcl3dVector2ByteArray`).

High level access

The file ***tcl3dVector.tcl*** contains additional Tcl commands for encapsulation of these low-level accessor functions. See the Tcl implementation file for a detailed explanation of the available procedures and its parameters.

Tcl command	Description
`tcl3dVector`	Create a new Tcl3D Vector by calling the memory allocation routine new_TYPE and create a new Tcl procedure. (See example below).

tcl3dVectorPrint	Print the contents of a Tcl3D Vector onto standard output.
tcl3dVectorFromArgs	Create a new Tcl3D Vector from given aruments.
tcl3dVectorFromList	Create a new Tcl3D Vector from given Tcl list.
tcl3dVectorFromString	Create a new Tcl3D Vector from given Tcl string. *Very slow.*
tcl3dVectorFromByteArray	Create a new Tcl3D Vector from given Tcl binary string.
tcl3dVectorFromPhoto	Create a new Tcl3D Vector containing the data of a Tk photo image. See next chapter for detailled description.
tcl3dVectorToList	Copy the contents of a Tcl3D Vector into a Tcl list.
tcl3dVectorToString	Copy the contents of a Tcl3D Vector into a string. *Very slow.*
tcl3dVectorToByteArray	Copy the contents of a Tcl3D Vector into a Tcl binary string.

Note

- The tcl3dFromString and tcl3dVectorToString commands can be replaced with the corresponding ByteArray commands, which are much faster.
- For functions converting photos into vectors and vice versa, see the next chapter about image manipulation.

The tcl3dVector command creates a new Tcl procedure with the following subcommands, which wrap the low-level vector access functions described above:

Subcommand	Description
get	Get vector element at a given index. (TYPE_getitem)
set	Set vector element at a given index to supplied value. (TYPE_setitem)
setvec	Set range of vector elements to supplied value. (TYPE_setarray)
addvec	Add supplied value to a range of vector elements. (TYPE_addarray)
mulvec	Muliply supplied value to a range of vector elements. (TYPE_mularray)
delete	Delete a tcl3dVector. (delete_TYPE)

The following example shows the usage of the tcl3dVector command.

```
set ind 23
set vec [tcl3dVector GLfloat 123] ; # Create Vector of size 123 GLfloats
$vec set $ind 1017.0              ; # Set element at index 23 to 1017.0
set x [$vec get $ind]             ; # Get element at index 23
$vec addvec 33 2 10               ; # Add 33 to ten elements starting at index 2
$vec delete                       ; # Free the allocated memory
```

Note

Indices start at zero.

See the demo program ***bytearray.tcl*** and ***vecmanip.tcl*** for examples, on how to use the ByteArray procedures for generating textures in Tcl.

4.2.5 Image utility module

This module provides access to photo images as supplied by Tk. The Img extension is recommended to have access to lots of image formats.

Implementation files: ***tkphoto.i***
Header files : None
Wrapper files: ***tkphoto.i***

In file ***tkphoto.i*** the following C functions are implemented and wrapped to provide access to the Tk photo image functionality.

Tcl command	Description
`tcl3dPhotoChans`	Return the number of channels of a Tk photo.
`tcl3dPhoto2Vector`	Copy a Tk photo into a `tcl3dVector` in OpenGL raw image format. The `tcl3dVector` must have been allocated with the approriate size and type.
`tcl3dVector2Photo`	Copy from OpenGL raw image format into a Tk photo. The photo image must have been initialized with the appropriate size and type.
`tcl3dVectorFromPhoto`	Create a new Tcl3D Vector containing the image data of a Tk photo image. Only GL_UNSIGNED_BYTE currently supported.

Example 1: Read an image into a Tk photo and use it as a texture map.

Note Texture map images must have width and height, that are powers of 2.

```
set texture [tcl3dVector GLuint 1] ; # Memory for 1 texture

proc LoadImage { imgName } {
    set retVal [catch {set phImg [image create photo -file $imgName]} err1]
    if { $retVal != 0 } {
        error "Error reading image $imgName ($err1)"
    } else {
        set numChans [tcl3dPhotoChans $phImg]
        if { $numChans != 3 && $numChans != 4 } {
            error "Error: Only 3 or 4 channels allowed ($numChans supplied)"
        }
        set w [image width  $phImg]
        set h [image height $phImg]
        set texImg [tcl3dVectorFromPhoto $phImg $numChans]
        image delete $phImg
    }
    return [list $texImg $w $h]
}

proc CreateTexture {} {
    # Load an image into a tcl3dVector.
    set imgInfo [LoadImage "Wall.bmp"]
    set imgData   [lindex $imgInfo 0]
    set imgWidth  [lindex $imgInfo 1]
    set imgHeight [lindex $imgInfo 2]

    # Create the texture identifiers.
    glGenTextures 1 $::texture

    glBindTexture GL_TEXTURE_2D [$::texture get 0]
    glTexParameteri GL_TEXTURE_2D GL_TEXTURE_MIN_FILTER $::GL_LINEAR
    glTexParameteri GL_TEXTURE_2D GL_TEXTURE_MAG_FILTER $::GL_LINEAR
    glTexImage2D GL_TEXTURE_2D 0 3 $imgWidth $imgHeight \
                 0 GL_RGBA GL_UNSIGNED_BYTE $imgData

    # Delete the image data vector.
    $imgData delete
}
```

Example 2: Read an image from the OpenGL framebuffer and save it with the Img library.

```
proc SaveImg { imgName } {
    set w $::toglWidth
    set h $::toglHeight
    set numChans 4
```

```
    set vec [tcl3dVector GLubyte [expr $w * $h * $numChans]]
    glReadPixels 0 0 $w $h GL_RGBA GL_UNSIGNED_BYTE $vec
    set ph [image create photo -width $w -height $h]
    tcl3dVector2Photo $vec $ph $w $h $numChans
    set fmt [string range [file extension $imgName] 1 end]
    $ph write $imgName -format $fmt
    image delete $phImg
    $vec delete
}

proc tclReshapeFunc { toglwin w h } {
    set ::toglWidth  $w
    set ::toglHeight $h
    ...
}
```

The actual size of the Togl window (`::toglWidth, ::toglHeight`), which is needed in command `SaveImg`, can be saved in a global variable when the reshape callback is executed.

See the NeHe demo program ***Lesson41.tcl*** or any demo using textures for examples, on how to use photo utilities.

4.2.6 Screen capture module

This module implements functions for capturing window contents into an image, file or the clipboard.

Note
All of the functionality requires the help of the ***Img*** extension.
Some of the functionality requires the help of the ***Twapi*** extension and is therefore available only on Windows.

Implementation files:	***tcl3dCapture.tcl***
Header files :	None
Wrapper files:	None

In file ***tcl3dCapture.tcl*** the following Tcl procedures are implemented:

Tcl command	Description
`tcl3dWidget2Img`	Copy contents of a widget and all of its sub-widgets into a photo image.
`tcl3dWidget2File`	Copy contents of a widget and all of its sub-widgets into a photo image and save the image to a file.
`tcl3dCanvas2Img`	Copy the contents of a Tk canvas into a photo image.
`tcl3dCanvas2File`	Copy the contents of a Tk canvas into a photo image and save the image to a file.
`tcl3dClipboard2Img`	Copy the contents of the Windows clipboard into a photo image.
`tcl3dClipboard2File`	Copy the contents of the Windows clipboard into a photo image and save the image to a file.
`tcl3dImg2Clipboard`	Copy a photo into the Windows clipboard.
`tcl3dWindow2Clipboard`	Copy the contents of the top level window (Alt-PrtSc) into the Windows clipboard.
`tcl3dWindow2Img`	Copy the contents of the top level window (Alt-PrtSc) into a photo image. (Windows only)
`tcl3dWindow2File`	Copy the contents of the top level window (Alt-PrtSc) into a photo image and save the image to a file. (Windows only)

See the demo program ***presentation.tcl*** for an example, on how to use these procedures to save screenshots of all available Tcl3D demos by right-clicking on the demo name.

4.2.7 Timing module

This module provides functions for timing purposes.

Implementation files:	***tcl3dStopWatch.c***
Header files :	***tcl3dStopWatch.h***
Wrapper files:	***util.i***

The tcl3dStopWatch.* files implement a stop watch with the following commands :

Tcl command	Description
`tcl3dNewSwatch`	Create a new stop watch and return it's identifier.
`tcl3dDeleteSwatch`	Delete an existing stop watch.
`tcl3dStopSwatch`	Stop a running stop watch.
`tcl3dStartSwatch`	Start a stop watch.
`tcl3dResetSwatch`	Reset a stop watch, i.e. set the time to zero seconds.
`tcl3dLookupSwatch`	Lookup a stop watch and return the elapsed seconds.

See the demo program ***spheres.tcl*** for an example, on how to use these procedures to measure the rendering frame rate.

4.2.8 3D-Model and shapes module

This module provides functions for reading 3D models from files and creating basic shapes.

Implementation files:	***tcl3dModel.c, tcl3dModelFmtObj.c, tcl3dShapes.c***
Header files:	***tcl3dModel.h, tcl3dModelFmtObj.h, tcl3dShapes.h***
Wrapper files:	***util.i***

The ***tcl3dModel.* and tcl3dModelFmtObj.**** files provide a parser for reading model files in Alias/Wavefront format. The code to read and draw the models is a modified version of the parser from Nate Robin's OpenGL tutorial [7].

Tcl command	Description
`glmUnitize`	"Unitize" a model by translating it to the origin and scaling it to fit in a unit cube around the origin.
`glmDimensions`	Calculates the dimensions (width, height, depth) of a model.
`glmScale`	Scales a model by a given amount.
`glmReverseWinding`	Reverse the polygon winding for all polygons in this model.
`glmFacetNormals`	Generates facet normals for a model.
`glmVertexNormals`	Generates smooth vertex normals for a model.
`glmLinearTexture`	Generates texture coordinates according to a linear projection of the texture map.
`glmSpheremapTexture`	Generates texture coordinates according to a spherical projection of the texture map.
`glmDelete`	Deletes a GLMmodel structure.
`glmReadOBJ`	Reads a model description from a Wavefront .OBJ file.
`glmWriteOBJ`	Writes a model description in Wavefront .OBJ format to a file.
`glmDraw`	Renders the model to the current OpenGL context using the mode specified.
`glmList`	Generates and returns a display list for the model using the mode

	specified.
`glmWeld`	Eliminate (weld) vectors that are within an epsilon of each other.

See the demo program ***gaugedemo.tcl*** for an example, on how to use these procedures.

The ***tcl3dShapes.**** files implement a sphere based on an algorithm found at Paul Bourke's excellent pages [10] as well as a cube and a helix based on algorithms found in the NeHe tutorials 23 and 36 [4].

Tcl command	Description
`tcl3dCube`	Draw a textured cube with given center and size.
`tcl3dHelix`	Draw a helix with given center, radius and number of twists.
`tcl3dSphere`	Draw a sphere with given radius precision.

See NeHe demo program ***Lesson23.tcl*** for an example, on how to use `tcl3dCube`.
See NeHe demo program ***Lesson36.tcl*** for an example, on how to use `tcl3dHelix`.
See demo program ***ogl_benchmark_sphere.tcl*** for an example, on how to use `tcl3dSphere`.

Note The standard GLUT shapes are implemented in module ***tcl3dOgl***, see chapter 4.3.

4.2.9 Virtual trackball module

This module provides functions for emulating a trackball.

Implementation files: ***tcl3dTrackball.c, tcl3dTrackball.tcl***
Header files: ***tcl3dTrackball.h***
Wrapper files: ***util.i***

The trackball module implements the following commands:

Tcl command	Description
`tcl3dTbInit`	Call this initialization procedure before any other trackball procedure.
`tcl3dTbReshape`	Call this procedure from the reshape callback.
`tcl3dTbMatrix`	Get the trackball matrix rotation.
`tcl3dTbStartMotion`	Begin trackball movement.
`tcl3dTbStopMotion`	Stop trackball movement.
`tcl3dTbMotion`	Call this procedure from the motion callback.
`tcl3dTbAnimate`	Call with parameter `1` (or `$::GL_TRUE`), if you want the trackball to continue spinning after the mouse button has been released. Call with parameter `0` (or `$::GL_FALSE`), if you want the trackball to stop spinning after the mouse button has been released.

See the demo program ***ftglDemo.tcl*** for an example, on how to use the trackball procedures.

4.2.10 Virtual ArcBall module

This module provides functions for emulating an ArcBall, which is the same as a trackball.

Implementation files: ***tcl3dArcBall.c***
Header files: ***tcl3dArcBall.h***
Wrapper files: ***util.i***

The ArcBall module implements the following commands:

Tcl command	Description
`tcl3dNewArcBall`	Create new ArcBall with given width and height.
`tcl3dDeleteArcBall`	Delete an ArcBall.
`tcl3dSetArcBallBounds`	Update mouse bounds for ArcBall. Call this procedure from the reshape callback.
`tcl3dArcBallClick`	Update start vector and prepare for dragging.
`tcl3dArcBallDrag`	Update end vector and get rotation as Quaternion.

See the NeHe demo program ***Lesson48.tcl*** for an example, on how to use the ArcBall procedures.

4.3 tcl3dOgl: Wrapper for basic OpenGL functionality

This module wraps ***OpenGL*** functionality based on OpenGL Version 1.1, as well as the GLU library functions based on Version 1.2. This is due to the fact, that Windows still does not support newer versions of OpenGL. OpenGL features defined in newer versions have to be accessed via the OpenGL extension mechanism on Windows.
The shapes of the GLUT library (box, sphere, cylinder, teapot, …) with a GLUT compatible syntax are supplied here, too.

Requirements for this module: An OpenGL 1.1 compatible library. OpenGL header files are contained in the Tcl3D distribution.

The master SWIG file for wrapping the basic OpenGL library is ***tcl3dOgl.i***.

Basic OpenGL library

Implementation files: ***tcl3dOglUtil.tcl***
Header files: ***gl.h, glu.h***
Wrapper files: ***gl.i, glu.i***

The wrapping for this module is based on the unmodified header files ***gl.h*** and ***glu.h***.

The following commands are implemented in file ***tcl3dOglUtil.tcl***:

Tcl command	Description
`tcl3dOglGetVersion`	Get the version of the wrapped OpenGL library.
`glMultiDrawElements`	Procedure to implement the OpenGL function `glMultiDrawElements`.
`tcl3dGetGlError`	Procedure to find out, if an OpenGL error has been occurred.

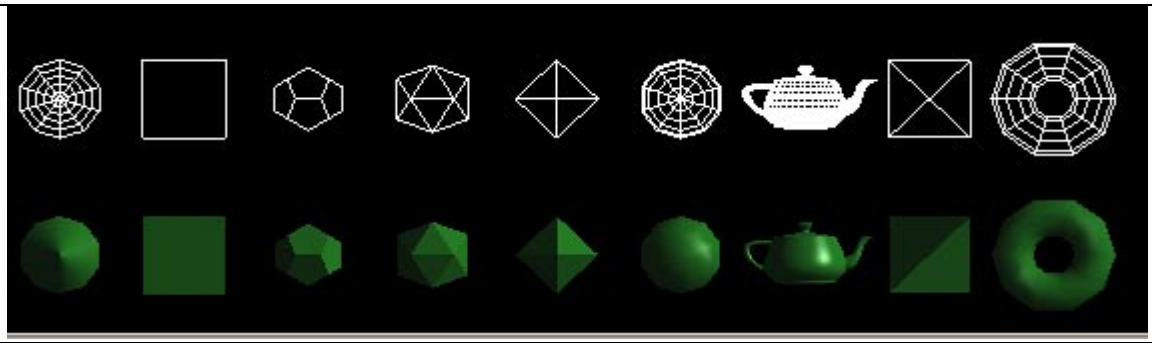
GLUT shapes library

Implementation files: ***glutShapes.c, glutTeapot.c, glutShapes.tcl***
Header files: ***glutShapes.h***
Wrapper files: ***tcl3dOgl.i***

The shapes library consists of the C files (***glutTeapot.c*** for the teapot, ***glutShapes.c*** for all other shapes and the common header file ***glutShapes.h***) and the Tcl file ***glutShapes.tcl***.

The GLUT shape objects are available under identical names for porting test and demonstration programs to Tcl3D. These shapes are used extensively in the examples of the OpenGL redbook [1]. See there for a description of the functions and its parameters.

Solid shapes	Wire shapes
glutSolidCone	glutWireCone
glutSolidCube	glutWireCube
glutSolidDodecahedron	glutWireDodecahedron
glutSolidIcosahedron	glutWireIcosahedron
glutSolidOctahedron	glutWireOctahedron
glutSolidSphere	glutWireSphere
glutSolidTeapot	glutWireTeapot
glutSolidTetrahedron	glutWireTetrahedron
glutSolidTorus	glutWireTorus



All supported GLUT shapes (Demo ***glutShapes.tcl***)

4.4 tcl3dOglExt: Wrapper for enhanced OpenGL functionality

This module wraps OpenGL functionality based on versions 1.2 till 2.0, lots of OpenGL extensions not contained in the OpenGL core, as well as Windows specific extensions.
The files of this logical building block are contained in the same directory as the basic OpenGL wrapper files for practial compilation reasons.

This is an optional module.
Requirements for this module: An OpenGL compatible library. OpenGL header files are contained in the Tcl3D distribution. To have access to all wrapped features, the OpenGL library should support Version 2.0.

The master SWIG file for wrapping the enhanced OpenGL library is ***tcl3dOgl.i***.

Implementation files:	See subdirectory ***OglExt***
Header files:	***glext.h, glprocs.h***
Wrapper files:	***glext.i, wglext.i***

The wrapping for OpenGL functions greater 1.1 and the OpenGL extensions is defined in file ***glext.i*** and based on the header file ***glext.h***. This header file is part of ***OglExt*** [24]***,*** an OpenGL extension library from the research center caesar. It has been slightly modified to fit the Tcl3D needs.
The wrapping of Windows specific OpenGL functions is defined in file ***wglext.i*** and based on the header file ***glprocs.h*** from Intel's ***GLsdk*** [25] library. The GLsdk is an extension library similar to the OglExt library. It has been stripped down to only use the Windows specific OpenGL functions.

Note If using functions from this module, be sure to add a call to `tcl3dInit` in the create callback. This initialization is necessary due to a bug in the ***OglExt*** library.

See the demo program ***extensions.tcl*** for an example, on how to use OpenGL extensions.

4.5 tcl3dCg: Wrapper for NVidia's Cg shading language

This module wraps NVidia's ***Cg*** [18] library based on version 1.5.0015 and adds some Cg related utility procedures.

This is an optional module.
Requirements for this module: The Cg library and header files.
Libraries are included in distribution.

The master SWIG file for wrapping the Cg library is ***tcl3dCg.i***.

Implementation files:	***tcl3dCgUtil.tcl***
Header files:	All files in subdirectory ***Cg***
Wrapper files:	***cg.i***

The wrapping for this module is based on the unmodified Cg header files.

Cg utility module

Tcl command	Description
`tcl3dCgGetVersion`	Get the version of the wrapped Cg library.
`tcl3dGetCgError`	Check, if a Cg related error has occured.
`tcl3dGetCgProfileList`	Get a list of Cg profile names.
`tcl3dFindCgProfile`	Find the first profile supported by the Cg implementation from the supplied profile names.
`tcl3dFindCgProfileByNum`	Find a profile name by it's numerical value.
`tcl3dPrintProgramInfo`	Print the Cg program information onto standard output.

See the demo programs contained in directory ***LibrarySpecificDemos/tcl3dCg*** for examples, on how to use the Cg functions.

4.6 tcl3dSDL: Wrapper for the Simple DirectMedia Library

This module wraps the ***SDL*** [19] library based on version 1.2.9 and adds some SDL related utility procedures.
Currently only the functions related to joystick and CD-ROM handling have been wrapped and tested.

This is an optional module.
Requirements for this module: The SDL library and header files.
Libraries and header files are included in distribution.

The master SWIG file for wrapping the Simple DirectMedia library is ***tcl3dSDL.i***.

Implementation files:	None
Header files:	All files in subdirectory ***include***
Wrapper files:	***sdl.i***

The wrapping for this module is based on the unmodified SDL header files.

SDL utility module

Tcl command	Description
tcl3dSDLGetVersion	Get the version of the wrapped SDL library.
tcl3dSDLGetFocusName	Convert a SDL focus state bitfield into a string representation.
tcl3dSDLGetButtonName	Convert a SDL button state bitfield into a string representation.
tcl3dSDLGetHatName	Convert SDL hat related enumerations into a string representation.
tcl3dSDLGetEventName	Convert SDL event related enumerations into a string representation.
tcl3dSDLFrames2MSF	Convert CD frames into minutes/seconds/frames.
tcl3dSDLGetTrackTypeName	Convert SDL CD track type enumerations into a string representation.
tcl3dSDLGetCdStatusName	Convert SDL CD status enumerations into a string representation.

See the demo programs contained in directory ***LibrarySpecificDemos/tcl3dSDL*** for examples, on how to use the SDL functions.

4.7 tcl3dFTGL: Wrapper for the OpenGL Font Rendering Library

This module wraps the ***FTGL*** [20] library based on version 2.1.2 and adds some FTGL related utility procedures.
The FTGL library depends on the ***Freetype2*** library [21].

This is an optional module.
Requirements for this module: The FTGL and Freetype2 library and header files.
Libraries and header files are included in distribution.

The master SWIG file for wrapping the OpenGL Font Rendering library is ***tcl3dFTGL.i***.

Implementation files:	None
Header files:	All files in subdirectory ***include***
Wrapper files:	***ftgl.i***

The wrapping for this module is based on the unmodified FTGL header files.

FTGL utility module

Tcl command	Description
tcl3dFTGLGetVersion	Get the version of the wrapped FTGL library.

See the demo programs contained in directory ***LibrarySpecificDemos/tcl3dFTGL*** for examples, on how to use the FTGL functions.

4.8 tcl3dGl2ps: Wrapper for the OpenGL To Postscript Library

This module wraps Christophe Geuzaine's ***GL2PS*** [22] library based on version 1.3.2 and adds some GL2PS related utility procedures.
Note Gl2PS does not support textures.

This is an optional module.

Requirements for this module: None, all files are contained in the Tcl3D distribution.

The master SWIG file for wrapping the Simple DirectMedia library is ***tcl3dGl2ps.i***.

Implementation files:	***gl2ps.c, tcl3dGl2psUtil.tcl***
Header files:	***gl2ps.h***
Wrapper files:	***gl2ps.i***

The wrapping for this module is based on the unmodified GL2PS implementation and header files.

Gl2ps utility module

Tcl command	Description
`tcl3dGl2psGetVersion`	Get the version of the wrapped GL2PS library.
`tcl3dGl2psCreatePdf`	Create a PDF file from current Togl window content.

See NeHe demo ***Lesson02.tcl*** or the benchmarking demo ***sphere.tcl*** in directory ***LibrarySpecificDemos/tcl3dOgl*** for an example, on how to use the GL2PS functions for PDF export.

4.9 tcl3dOde: Wrapper for the Open Dynamics Engine

This module wraps the ***ODE*** [23] library based on version 0.7 and adds some ODE related utility procedures.

Note This module is still work in progress. It's interface may change in the future.

This is an optional module.
Requirements for this module: The ODE library and header files.
Libraries and header files are included in distribution.

The master SWIG file for wrapping the Open Dynamics Engine library is ***tcl3dOde.i***.

Implementation files:	None
Header files:	All files in subdirectory ***ode***
Wrapper files:	***ode.i***

The wrapping for this module is based on the unmodified ODE header files.

ODE utility module

Tcl command	Description
`tcl3dOdeGetVersion`	Get the version of the wrapped ODE library.

See the demo programs contained in directory ***LibrarySpecificDemos/tcl3Ode*** for examples, on how to use the ODE functions.

4.10 tcl3dGauges: Tcl3D package for displaying gauges

This package implements the following gauges: airspeed, altimeter, compass, tiltmeter.

This is an optional module.

Requirements for this module: None, all files are contained in the Tcl3D distribution.

The gauge package has been implemented by Victor G. Bonilla.

See the demo programs ***gaugedemo.tcl*** and ***gaugetest.tcl*** for examples, on how to use the gauges.

4.11 tcl3dDemoUtil: C/C++ based utilities for demo applications

This package implements several C/C++ based utility functions for some of the demo applications.

This is an optional module.
Requirements for this module: None, all files are contained in the Tcl3D distribution.

The master SWIG file for wrapping the demo utility library is ***tcl3dDemoUtil.i***.

The following submodules are contained in this module:

Name:	***tcl3dOglLogo***
Implementation files:	***tcl3dOglLogo.c***
Header files:	***tcl3dOglLogo.h***
Wrapper files:	***demoutil.i***

tcl3dOglLogo implements an animated 3-dimensional OpenGL logo.
It is used in demo ***animlogo.tcl*** in directory ***LibrarySpecificDemos/tcl3dOgl***.

Name:	***tcl3dReadRedBookImg***
Implementation files:	***tcl3dReadRedBookImg.c***
Header files:	***tcl3dReadRedBookImg.h***
Wrapper files:	***demoutil.i***

tcl3dReadRedBookImg implements a parser for the simple image file format used in some of the RedBook demos.
It is used in demos ***colormatrix.tcl***, ***colortable.tcl***, ***convolution.tcl***, ***histogram.tcl*** and ***minmax.tcl*** in directory ***TutorialsAndBooks/RedBook***.

Name:	***tcl3dHeightmap***
Implementation files:	***heightmap.i, tcl3dHeightMap.tcl***
Header files:	***None***
Wrapper files:	***heightmap.i***

tcl3dHeightmap implements a photo image to heightmap converter.
It is used in NeHe demo ***Lesson45.tcl*** in directory ***TutorialsAndBooks/NeHe***.

5 Miscellaneous Tcl3D information

This chapter contains various information about Tcl3D.

5.1 License information

The Tcl3D utility library files (see below for exceptions) are copyrighted by Paul Obermeier and distributed under the BSD license.
The following files of the Tcl3D utility library have differing copyrights:

- The original Wavefront parser code is copyrighted by Nate Robins.
- The original GLUT shape code is copyrighted by Mark Kilgard.
- The original code of tcl3dSphere is copyrighted by Paul Bourke.
- The original code of tcl3dHelix is copyrighted by Dario Corno.
- The original code of tcl3dArcBall is copyrighted by Tatewake.com.
- The original code of tcl3dTrackball is copyrighted by Gavin Bell et al.

The Tcl3D gauge library is copyrighted by Victor G. Bonilla and distributed under the BSD license.

The original Togl widget is copyrighted by Brian Paul and Benjamin Bederson. The modified Tcl3D version is copyrighted by Paul Obermeier and distributed under the BSD license.

The SWIG wrapper files and supporting Tcl files of all modules are copyrighted by Paul Obermeier and distributed under the BSD license.

See the homepages of the wrapped libraries for their license conditions.

5.2 Programming hints

Hint 1:
Some OpenGL functions expect an integer or floating point value, which is often given in C code examples with an enumeration, as shown in the next example:

```
extern void glTexParameteri ( GLenum target, GLenum pname, GLint param );
```

It is called in C typically as follows:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

As the 3rd parameter is not of type `GLenum`, you have to specify the numerical value here:

```
glTexParameteri GL_TEXTURE_2D GL_TEXTURE_WRAP_S $::GL_REPEAT
glTexParameteri GL_TEXTURE_2D GL_TEXTURE_MAG_FILTER $::GL_NEAREST
```

If called with the enumeration name:

```
glTexParameteri GL_TEXTURE_2D GL_TEXTURE_WRAP_S GL_REPEAT
```

you will get an error message like this: expected integer but got "GL_REPEAT"

Hint 2:
Most OpenGL examples written in C use the immediate mode. As Tcl is a scripted language and each OpenGL call has to go through the wrapper interface, it's almost always a bad idea (in terms of speed) to translate these examples one-by-one. Using display lists or vertex arrays does not add much complexity to your Tcl3D program, but enhances performance significantly. Try the ***Spheres.tcl*** or ***ogl_benchmark_sphere.tcl*** demo for an example, how display lists or vertex arrays can speed up your Tcl3D application.

Hint 3:
Do not use global variables GL_VERSION_X_Y (ex. [info exists GL_VERSION_1_3]) to check the OpenGL version supported on your computer. This does not work, because these variables are defined in the range 1_1 till 2_0 in Tcl3D. Use the utility function tcl3dHaveExtension instead.

5.3 Open issues

- GLU callbacks are currently not supported. This implies, that tesselation does not work, because this functionality relies heavily on the usage of C callback functions.
- There is currently no possibility to specify a color map for OpenGL indexed mode. As color maps depend on the underlying windowing system, this feature must be handled by the Togl widget.

5.4 Known bugs

- The tiltmeter widget from the tcl3dGauge package is not working correctly with Tcl versions less than 8.4.7, because of a bug in the namespace implementation.
- Picking with depth values does not work correctly, as depth is returned as an unsigned int, mapping the internal floating-point depth values [0.0 .. 1.0] to the range [0 .. 2^{32} –1]. As Tcl only supports signed integers, some depth values are incorrectly transferred into the Tcl commands.
- SWIG versions up to 1.3.24 had an annoying (but not critical) bug in the Tcl library file ***swigtcl8.swg***: Please check, if your version has a line `"printf ("Searching %s\n", key);"` in function `SWIG_Tcl_GetConstant`, and delete this line, if existent. ***swigtcl8.swg*** can be found in ***/usr/lib/swig1.3/tcl*** or ***/usr/share/swig/VERSION/tcl*** on Linux or in the ***lib/tcl*** subdirectory of your SWIG Windows installation.
- SWIG version 1.3.21 (as delivered with SuSE 9.3) does not correctly wrap the ODE library.

5.5 Starpack internals

For an introduction to Tclkits, Starkits and Starpacks see Jean-Claude Wippler's homepage at http://www.equi4.com/.

5.5.1 Starpack problem 1

If shipping external libraries with your Starpack, you have to copy them to the file system, before they can be used. Best place is the directory containing the Starpack.

```
# Check if all necessary external libraries exists in the directory
# containing the Starpack. Copy them to the filesystem, if necessary.
set __tcl3dExecDir [file dirname $::starkit::topdir]
set __tcl3dDllList [glob -nocomplain -dir [file join $starkit::topdir extlibs] \
                 *[info sharedlibextension]*]

foreach starkitName $__tcl3dDllList {
    set osName [file join $__tcl3dExecDir [file tail $starkitName]]
    if { ! [file exists $osName] } {
        set retVal [catch { file copy -force -- $starkitName $__tcl3dExecDir }]
        puts "Copying DLL $starkitName to directory $__tcl3dExecDir"
        if { $retVal != 0 } {
            error "Error copying DLL $starkitName to directory $__tcl3dExecDir"
        }
    }
}
```

This aforementioned solution seems to be the best possible solution today, but has the following two disadvantages:

- Windows user will typically place the Starpack onto the desktop. Starting the Starpack inflates the desktop with lots of DLL's.
- On Linux/Unix the current directory typically is not included in the LD_LIBRARY_PATH variable.

That's why the starpacks are distributed in it's own folder, and the Unix distributions come with an additional start shell script: `tcl3dsh-OS-VERSION.sh`

```
#!/bin/sh
# Startup script for tcl3dsh, the Tcl3D Starpack.

LD_LIBRARY_PATH=".:$LD_LIBRARY_PATH"
LD_LIBRARYN32_PATH=".:$LD_LIBRARYN32_PATH"
export LD_LIBRARY_PATH
export LD_LIBRARYN32_PATH

./tcl3dsh-Linux-0.3.2
```

5.5.2 Starpack problem 2

Some of the external libraries need files for initialization, ex. the FTGL library needs the name of a TrueType font file to construct it's OpenGL commands. This font file has to be on the real filesystem, so that the FTGL library can find it, and not in the virtual filesystem of the starpack.
Tcl3D supports a utility procedure `tcl3dGetExtFile`, which you should use, if intending to use a Tcl3D script - depending on such a library - in a Starpack. See file ***tcl3dFile.tcl*** in directory ***tcl3dUtil/tclfiles*** for the code of the procedure and more inline comments.

A typical usage is shown in the following code segment:

```
set fontfile [file join [file dirname [info script]] "Vera.ttf"]
# tcl3dGetExtFile is available only in versions 0.3.1 and up.
# You may check availability of command first, if running scripts with older
# Tcl3D versions.
if { [info proc tcl3dGetExtFile] eq "tcl3dGetExtFile" } {
    # Get the font file in a Starpack independent way.
    set fontfile [tcl3dGetExtFile $fontfile]
}
```

6 Demo applications

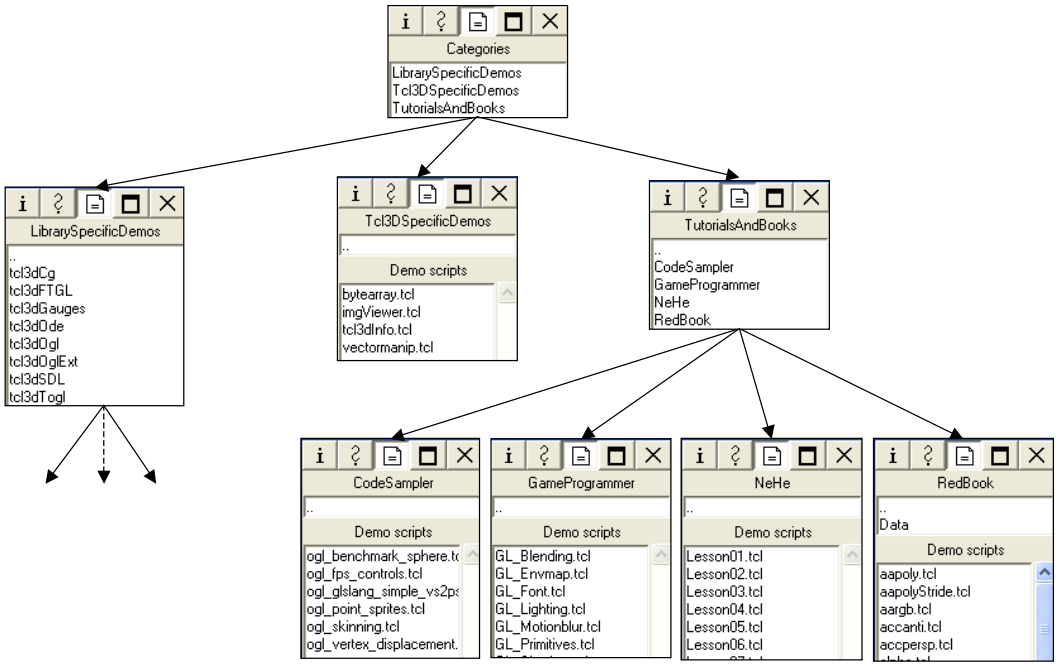
More than 100 Tcl3D applications for testing and demonstration purposes are currently available. Most of these applications were converted from existing demonstration programs written in C/C++ found on the web. A detailed list of all demos is available online on the Tcl3D homepage at http://www.tcl3d.org/demos/ or in the Tcl3D Demo Manual.

The Tcl3D demo applications are divided into 3 categories:

- Category **Tutorials and books** contains scripts, which have been converted from C/C++ to Tcl3D, coming from the following sources:
  - OpenGL Red Book [8]
  - NeHe tutorials [4]
  - Kevin Harris CodeSampler web site [5]
  - Vahid Kazemi's GameProgrammer page [6]
- Category **Library specific demos** contains scripts showing features specific to the wrapped library.
- Category **Tcl3D specific demos** contains scripts demonstrating and testing Tcl3D specific features.

The next figure shows an excerpt from the demo hierarchy.

Tcl3D Demo Hierarchy

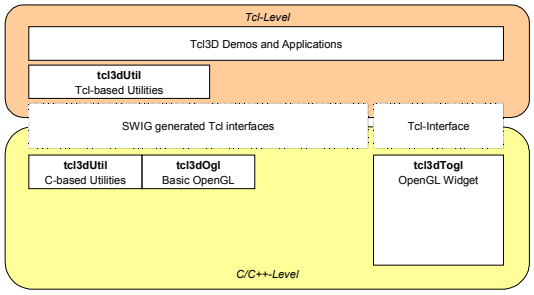


7 Release notes

This chapter shows the release and feature history of Tcl3D both graphically and in text form.

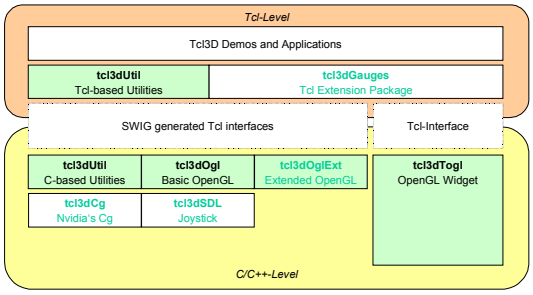
Tcl3D Version 0.1

Released 2005/05/29 as TclOgl: Basic OpenGL wrapping, Togl widget with Tcl callbacks.



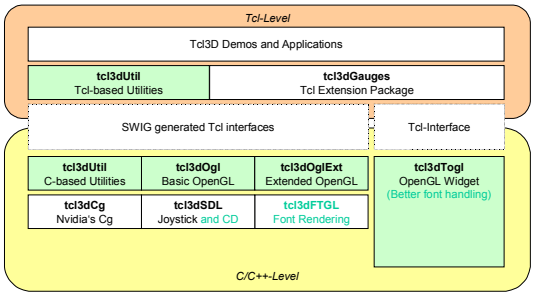
Tcl3D Version 0.2

Released 2006/01/07: Major rewrite and support of new libraries: OpenGL 2.0, OpenGL extensions, Cg, SDL, gauges. Domain www.tcl3d.org created.



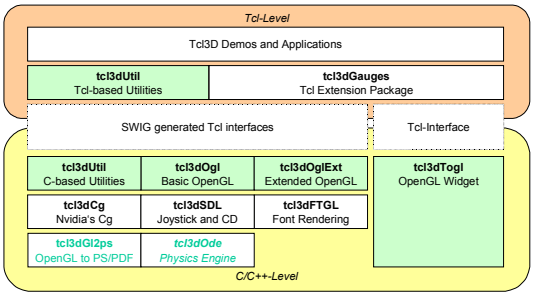
Tcl3D Version 0.3

Released 2006/02/12: Enhanced font handling in Togl. Library FTGL added. Mac OS X support supplied by Daniel Steffen.



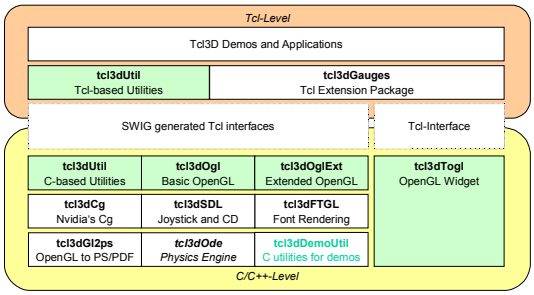
Tcl3D Version 0.3.1

Released 2006/06/16: Support for GL2PS and ODE (alpha) added. Starpack versions.



Tcl3D Version 0.3.2

Released 2007/02/25: Demo cleanup and first official Mac OS X support. Windowing system specifics incorporated into Togl widget. New module tcl3dDemoUtil.



Date	Version	Release information
2007/02/25	0.3.2	**Demo cleanup and first official Mac OS X support:** ➢ Unification of demo applications and presentation framework. ➢ New module tcl3dDemoUtil for C/C++ based utility functions needed by some of the demos for speed issues. ➢ More NeHe tutorials added: Lessons 14, 22-24, 26, 28, 33, 36, 37, 41, 45-48. ➢ Nine demos from www.GameProgrammer.org added. ➢ Updated Tcl3D manual. Created separate demo reference document. ➢ Added support to capture screenshots (Module tcl3dCapture).

		➢ Added new functionality to tcl3dUtil: ArcBall emulation. ➢ Added windowing system specifics (SwapInterval, Multisampling) to the tcl3dTogl widget. ➢ Added support for Visual Studio 2003 (7.1) and 2005 (8.0). ➢ Enhanced tcl3dVector functionality. • Utility functions for manipulation of image data stored in tcl3dVectors: tcl3dVectorCopy, tcl3dVectorCopyChannel, tcl3dVectorManip, tcl3dVectorManipChannel • tcl3dVector member functions for content independent manipulation: setvec, addvec, mulvec ➢ tcl3dOde now uses ODE version 0.7 and is available for Windows, Linux, Mac OS X and IRIX. Wrapper still in alpha version and not complete. ➢ tcl3dGl2ps now uses GL2PS version 1.3.2. ➢ tcl3dCg now uses Cg version 1.5.0015. The 1.4 versions of Cg did not work with OS X on Intel platforms.
2006/06/19	0.3.1	**Starpack support for Tcl3D:** ➢ Starpack version of Tcl3D, including demos and external libraries. First shown at TclEurope 2006. ➢ New optional module tcl3dGl2ps, wrapping the OpenGL To Postscript library. Thanks to Ian Gay for idea and first implementation. ➢ New optional module tcl3dOde, wrapping the Open Dynamics Engine. Very alpha preview, Windows only !!! ➢ More NeHe tutorials added: Lessons 19-21.
2006/02/12	0.3	**Bug-fixes and enhancements:** ➢ Support for Mac OS X added. (Thanks to Daniel A. Steffen for supplying Darwin patches and binaries) ➢ New optional module tcl3dFTGL, wrapping the OpenGL font rendering library FTGL, based on freetype fonts. ➢ Corrected and enhanced font handling under Windows in the tcl3dTogl widget. No more private Tcl header files needed. ➢ Added new font related demo programs: tcl3dFont.tcl, tcl3dToglFonts.tcl, ftglTest.tcl, ftglDemo.tcl. ➢ Added new SDL demo related to CD-ROM handling: cdplayer.tcl ➢ Added some of NeHe's OpenGL tutorials. ➢ If an optional library is not installed, no error message is created. New procedures to check existence of optional modules: tcl3dHaveCg, tcl3dHaveSDL, tcl3dHaveFTGL. ➢ Get information on Tcl3D subpackages with tcl3dGetPackageInfo and tcl3dShowPackageInfo. ➢ Information program tcl3dInfo.tcl enhanced to support commands and enums of SDL and FTGL modules. ➢ Added new functionality to tcl3dUtil: Simple, scrollable Tk widgets for demo programs, virtual trackball (used in FTGLdemo.tcl). ➢ Added new functionality to tcl3dUtil: tcl3dVectorFromByteArray, tcl3dVectorToByteArray. Convert Tcl binary strings to tcl3dVectors and vice versa (see demo bytearray.tcl). ➢ Bug fix in OglExt wrapping: Parameters of type `float *` and `double *` were wrapped incorrectly.
2006/01/07	0.2	**Major rewrite and additional support of several new 3D libraries:** ➢ OpenGL extensions ➢ Cg shader ➢ SDL ➢ Gauge widgets (Thanks to Victor G. Bonilla for supplying this library) ➢ Utility library

		➢ Renamed from tclogl to Tcl3D. ➢ Created domain tcl3d.org.
2005/05/29	0.1	**First version called tclogl:** ➢ Introduced at the Tcl Europe 2005 conference. ➢ Supported features include basic OpenGL wrapping.

A note for users of the first version 0.1 (called tclogl).
Usage of tclogl is not recommended anymore.
The following Tcl procedures have different names in the newer versions. It is recommended to update your scripts to the new naming scheme. The following Tcl lines make your old scripts run with the new releases:

```
rename ::Vector ::tcl3dVector
rename ::VectorPrint ::tcl3dVectorPrint
rename ::VectorFromList ::tcl3dVectorFromList
rename ::VectorFromArgs ::tcl3dVectorFromArgs
rename ::VectorFromString ::tcl3dVectorFromString
rename ::VectorToString ::tcl3dVectorToString
rename ::VectorToList ::tcl3dVectorToList
rename ::CharToNum ::tcl3dCharToNum
rename ::Photo2Vector ::tcl3dPhoto2Vector
rename ::Vector2Photo ::tcl3dVector2Photo
rename ::PhotoChans ::tcl3dPhotoChans
```

8 References

[1] Woo, Neider, Davis: OpenGL Programming Guide, Addison-Wesley, **"The Redbook"**

[2] Roger E Critchlow's Frustum: http://www.elf.org/pub/frustum01.zip

[3] Togl page at SourceForge: http://sourceforge.net/projects/togl/

[4] NeHe's tutorials: http://nehe.gamedev.net/

[5] Kevin Harris' code samples: http://www.codesampler.com/oglsrc.htm

[6] Vahid Kazemi's GameProgrammer page: http://www.gameprogrammer.org/

[7] Nate Robins OpenGL tutorials: http://www.xmission.com/~nate/tutors.html

[8] The Redbook sources: http://www.opengl-redbook.com/source/

[9] OpenGL GLUT demos: http://www.opengl.org/resources/code/samples/glut_examples/demos/demos.html

[10] Paul Bourke's textured sphere: http://local.wasp.uwa.edu.au/~pbourke/texture/spheremap/

[11] OpenGL Wiki page: http://wiki.tcl.tk/2237

[12] SWIG (Simplified Wrapper and Interface Generator): http://www.swig.org/

[13] Paul Obermeier's Portable Software: http://www.posoft.de/

[14] Tcl3D homepage: http://www.tcl3d.org/

[15] Tcl3D page on the Tclers Wiki: http://wiki.tcl.tk/15278

[16] Tcl3D discussion page on the Tclers Wiki: http://wiki.tcl.tk/16057

[17] Tcl download: http://www.activestate.com/

[18] Cg download: http://developer.nvidia.com/object/cg_toolkit.html

[19] SDL download: http://www.libsdl.org/

[20] FTGL download: http://homepages.paradise.net.nz/henryj/code/index.html

[21] Freetype download: http://www.freetype.org/

[22] GL2PS download: http://www.geuz.org/gl2ps/

[23] ODE download: http://www.ode.org/

[24] OglExt Julius Caesar: http://www.julius.caesar.de/index.php/OglExt

[25] GLsdk library: http://oss.sgi.com/projects/ogl-sample/sdk.html

[26] OpenGL Extension Registry: http://www.opengl.org/registry/

[27] Starpack Wiki page: http://wiki.tcl.tk/3663