Tcl3D – Doing 3D with Tcl

1 Introduction

Tcl3D enables the 3D functionality of OpenGL and various other 3D libraries at the Tcl scripting level on Windows and Linux platforms.

Its main design requirement is to wrap existing 3D libraries without modification of their header files and with minimal manual code writing. The Tcl API shall be a direct wrapping of the C/C++ based library API's, with a natural mapping of C types to according Tcl types.

This is accomplished mostly with the help of SWIG, the Simplified Wrapper and Interface Generator.

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

The Tcl3D homepage is at https://www.tcl3d.org/
Tcl3D sources are available at https://sourceforge.net/projects/tcl3d/.
Tcl3D Windows binaries are available via ***BAWT*** (Build Automation With Tcl) at https://www.tcl3d.org/bawt/index.html

Tcl3D is distributed in two files: One file containing source code and documentation, the other file contains several demo programs showing the use of the Tcl3D functionality. The distribution files are available at https://www.tcl3d.org/download.html.

Additional reference documentation is available at https://www.tcl3d.org/documentation.html.

1.1 Architecture Overview

The ***Tcl3D*** package currently consists of the following building blocks, also called modules throughout the manual. The modules of distribution type Tcl3D-Basic do not need external libraries, while the modules of distribution type Tcl3D-Full need external libraries like SDL or OpenSceneGraph.

Distribution Type	Module	Sub-Module	Description
Tcl3D-Basic	tcl3dOgl	Togl	Enhanced Togl widget, a Tk widget for displaying OpenGL content.
		OpenGL	Wrapper for core OpenGL functionality and OpenGL extensions.
		Util	Tcl3D utility library: Math functions, standard shapes, stop watch, demo support.
Tcl3D-Basic	tcl3dGauges		Pure Tcl package for displaying gauges.
Tcl3D-Basic	tcl3dGl2ps		Wrapper for the OpenGL to Postscript library.
Tcl3D-Full	tcl3dFTGL		Wrapper for the OpenGL Font Rendering Library.
Tcl3D-Full	tcl3dOsg		Wrapper for the OpenSceneGraph library.
Tcl3D-Full	tcl3dSDL		Wrapper for the Simple DirectMedia Library.

Table 1: Overview of Tcl3D modules

The next figure shows the currently available modules of the Tcl3D package.

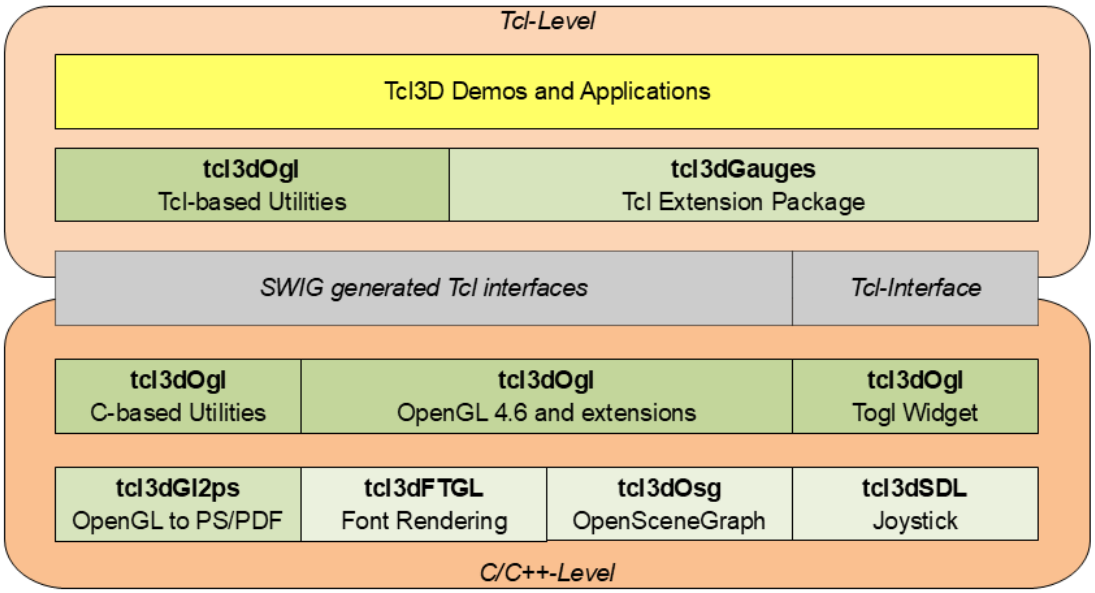


Figure 1: Overview of Tcl3D modules

Each module is implemented as a separate Tcl package and can be loaded explicitly with the Tcl package command, ex. `package require tcl3dsdl`. All available Tcl3D modules can be loaded with a single command: `package require tcl3d`.

Note: Package names are all lower case.

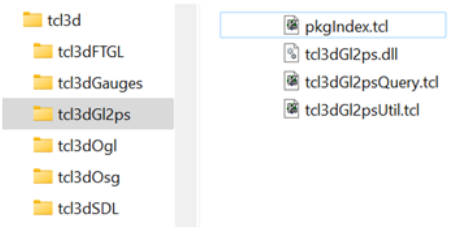


Figure 2: Tcl3D module layout

1.2 License Information

The ***Tcl3D*** package is copyrighted by Paul Obermeier and distributed under the 3-clause BSD license. See file `Tcl3D-License.txt` in the Tcl3D source distribution for the license text.
Parts of Tcl3D rely on **OpenSource** libraries. See below for license information of these libraries.

The Tcl3D utility library files 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 and distributed under the BSD license (see https://sourceforge.net/projects/togl). The modified Tcl3D version of Togl is copyrighted by Paul Obermeier and distributed under the BSD license.

See the following table of wrapped, unmodified libraries for their license conditions.

Library	License	More information
GLEW	Modified BSD license	https://github.com/nigels-com/glew/blob/master/LICENSE.txt
FTGL	LGPLv2	https://sourceforge.net/projects/ftgl/
Freetype	Freetype License (BSD style)	https://freetype.org/license.html
GL2PS	LGPL	See file COPYING.GL2PS at https://gitlab.onelab.info/gl2ps/gl2ps
OSG	OpenSceneGraph Public License (LGPL style)	https://openscenegraph.github.io/openscenegraph.io/about/license.html
SDL	zlib	http://www.libsdl.org/license.php

Table 2: License information of wrapped libraries

1.3 Modules Overview

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

1.3.1 Module tcl3dOgl: Sub-module Togl

This sub-module is an enhanced version of the Togl widget, a Tk canvas for displaying OpenGL graphics.

The following enhancements are currently implemented:

- Callback functions in Tcl.
- Better bitmap font support.
- Multisampling support.
- Swap Interval support.
- OpenGL Core and Compatibility profile support.

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

A detailed description of this sub-module can be found in chapter *3.1 Module tcl3dOgl: Sub-module Togl.*

1.3.2 Module tcl3dOgl: Sub-module OpenGL

This sub-module wraps OpenGL functionality up to OpenGL Version 4.6, the GLU library functions based on Version 1.2 and most of the available OpenGL extensions.

It is implemented with the help of the GLEW library.

Standard shapes (box, sphere, cylinder, teapot, ...) with a GLUT compatible syntax are supplied here, too.

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

A detailed description of this sub-module can be found in chapter *3.2 Module tcl3dOgl: Sub-module OpenGL*.

1.3.3 Module tcl3dOgl: Sub-module Util

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

- 3D vector and transformation matrix module.
- Information module.
- File utility module.
- Color names module.
- Large data module.
- Image utility module.
- Screen capture module.
- Timing module.
- Random number module.
- 3D model and shapes module.
- Virtual trackball and arcball module.
- C/C++ based utility functions for some of the demo applications.

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

A detailed description of this sub-module can be found in chapter *3.3 Module tcl3dOgl: Sub-module Util*.

1.3.4 Module tcl3dGauges

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

This is an optional module and contained in the Tcl3D-Basic distribution.

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

A detailed description of this module can be found in chapter *3.4 Module tcl3dGauges*.

1.3.5 Module tcl3dGl2ps

This module wraps the GL2PS library and adds some GL2PS related utility procedures.

GL2PS is a C library providing high quality vector output (PostScript, PDF, SVG) for any OpenGL application. It does not support textures.

This is an optional module and contained in the Tcl3D-Basic distribution.

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

A detailed description of this module can be found in chapter *3.5 Module tcl3dGl2ps*.

1.3.6 Module tcl3dFTGL

This module wraps the FTGL library 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 and contained in the Tcl3D-Full distribution.

Requirements for this module: The FTGL and Freetype2 library and header files.

A detailed description of this module can be found in chapter *3.6 Module tcl3dFTGL*.

1.3.7 Module tcl3dOsg

This module wraps the OpenSceneGraph library and adds some OSG related utility procedures.

This is an optional module and contained in the Tcl3D-Full distribution.

Requirements for this module: The OSG library and header files.

A detailed description of this module can be found in chapter *3.7 Module tcl3dOsg*.

1.3.8 Module tcl3dSDL

This module wraps the Simple DirectMedia Layer library and adds some SDL related utility procedures.

Currently only the functions related to joystick handling have been used and tested.

This is an optional module and contained in the Tcl3D-Full distribution.

Requirements for this module: The SDL library and header files.

A detailed description of this module can be found in chapter *3.8 Module tcl3dSDL*.

1.4 Getting Started

The easiest way to get started on Windows, is using a BAWT Batteries Included Installation file, which contains a ***Tcl3D-Basic*** distribution. After installation of the Tcl/Tk distribution, download the Tcl3D demo ZIP file and start the `presentation.tcl` script to try the demos.

If you want to use the ***Tcl3D-Full*** distribution on Windows or use ***Tcl3D*** on Linux, follow the steps in the next chapter.

1.5 Build and Installation

The build environment of Tcl3D is based on CMake since version 0.9.3. In previous versions it was based on GNU Makefiles, which are not supported anymore.

The easiest way to build Tcl3D, is using the ***BAWT*** framework. You may use the Setup file `Tcl_Extended.bawt` to build the ***Tcl3D-Basic*** distribution or use Setup file `Tcl3D.bawt` to build the ***Tcl3D-Full*** distribution.
See https://www.tcl3d.org/bawt/documentation.html for information on how to use the BAWT framework.

Note:

- To compile Tcl3D with Tcl 9, you have to use at least SWIG version 4.2.1.
- Recommended versions (as used in BAWT) for externals libraries are:
  - FTGL 2.1.3
  - SDL 2.26.2
  - OpenSceneGraph 3.4.1 or 3.6.5

The following Tcl3D specific CMake variables can be adapted to enable compilation of specific modules. The default values are set for a ***Tcl3D-Basic*** distribution.

Variable	Description	Default
TCL3D_BUILD_OGL	Compile base module tcl3dOgl	ON
TCL3D_BUILD_GAUGES	Compile optional module tcl3dGauges	ON
TCL3D_BUILD_GL2PS	Compile optional module tcl3dGl2ps	ON
TCL3D_BUILD_FTGL	Compile optional module tcl3dFTGL	OFF
TCL3D_BUILD_SDL	Compile optional module tcl3dSDL	OFF
TCL3D_BUILD_OSG	Compile optional module tcl3dOsg	OFF
TCL3D_OPTIMIZE_OSG	Compile tcl3dOsg wrapper with optimization	ON

Table 3: CMake variables for Tcl3D configuration

The following variables must be specified to find the Tcl and Tk header and library files.

Variable	BAWT Example (MSys/MinGW)
TCL_INCLUDE_PATH	Development/opt/Tcl/include

TK_INCLUDE_PATH	Development/opt/Tcl/include
TCL_STUB_LIBRARY	Development/opt/Tcl/lib/libtclstub86.a
DTK_STUB_LIBRARY	Development/opt/Tcl/lib/libtkstub86.a

Table 4: CMake variables for Tcl/Tk

As the modules of ***Tcl3D-Full*** depend on external libraries, the location of the header and library files must be specified.

Variable	**BAWT Example (MSys/MinGW)**
FTGL_INCLUDE_DIR	Development/include
FTGL_LIBRARY	Development/lib/libftgl.dll.a
FREETYPE_LIBRARY	Development/lib/libfreetype.a
FREETYPE_INCLUDE_DIR_freetype2	Development/include/freetype/config
DFREETYPE_INCLUDE_DIR_ft2build	Development/include/freetype
SDL_INCLUDE_DIR	Development/include/SDL
SDL_LIBRARY	Development/lib/libSDL2.dll.a
OSGDB_INCLUDE_DIR	Development/include
OSGDB_LIBRARY_RELEASE	Development/lib/libosgDB.dll.a

Table 5: CMake variables for external libraries

The OpenSceneGraph library consists of several packages, which have their own environment variables. The easiest way to specify all OSG related CMake variables is to set the `OSGDIR` environment variable pointing to the root directory of the OSG header and library files.

Installation tests

Start a ***tclsh*** or ***wish*** shell and type the following two commands:

```
> package require tcl3d
> togl .t
```

Now use either the command `tcl3dShowPackageInfo` for graphical package information or `tcl3dGetPackageInfo` for textual package information.

Version ***Tcl3D-Basic*** should print out information similar to the lines listed below, when calling `tcl3dGetPackageInfo`:

```
{tcl3dftgl   0 {can't find package tcl3dftgl} {}}
{tcl3dgauges 1 1.0.0 {}}
{tcl3dgl2ps  1 1.0.0 1.4.2}
{tcl3dogl    1 1.0.0 {4.6.0 NVIDIA 512.36}}
{tcl3dosg    0 {can't find package tcl3dosg} {}}
{tcl3dsdl    0 {can't find package tcl3dsdl} {}}
```

Version ***Tcl3D-Full*** should print out information similar to the lines listed below, when calling `tcl3dGetPackageInfo`:

```
{tcl3dftgl   1 1.0.0 2.1.3-rc5}
{tcl3dgauges 1 1.0.0 {}}
{tcl3dgl2ps  1 1.0.0 1.4.2}
{tcl3dogl    1 1.0.0 {4.6.0 NVIDIA 512.36}}
{tcl3dosg    1 1.0.0 3.4.1}
{tcl3dsdl    1 1.0.0 2.26.2}
```

If the above mentioned procedures fail, you may try the low-level information supplied in Tcl array `__tcl3dPkgInfo`:

```
> parray __tcl3dPkgInfo
__tcl3dPkgInfo(tcl3dftgl,avail)     = 0
__tcl3dPkgInfo(tcl3dftgl,version)   = can't find package tcl3dftgl
__tcl3dPkgInfo(tcl3dgauges,avail)   = 1
__tcl3dPkgInfo(tcl3dgauges,version) = 1.0.0
__tcl3dPkgInfo(tcl3dgl2ps,avail)    = 1
__tcl3dPkgInfo(tcl3dgl2ps,version)  = 1.0.0
__tcl3dPkgInfo(tcl3dogl,avail)      = 1
__tcl3dPkgInfo(tcl3dogl,version)    = 1.0.0
__tcl3dPkgInfo(tcl3dosg,avail)      = 0
__tcl3dPkgInfo(tcl3dosg,version)    = can't find package tcl3dosg
__tcl3dPkgInfo(tcl3dsdl,avail)      = 0
__tcl3dPkgInfo(tcl3dsdl,version)    = can't find package tcl3dsdl
```

Please report problems or errors to info@tcl3d.org or open a ticket on SourceForge.

Use the following script when sending bug reports or questions to supply me with information about your environment.

```
catch { console show }

package require tcl3d
togl .t

# Print information about the OS.
parray tcl_platform

# Print information about the Tcl3D modules.
puts [tcl3dGetPackageInfo]

# Print information about the OpenGL driver.
puts [tcl3dOglGetVersions]

# If it is a problem with an OpenGL extension, you should also
# include the output of the following statement:
puts [tcl3dOglGetExtensions]

# If Tcl3D procedures are not found
# (ex. invalid command name "tcl3dShowPackageInfo")
# print out some low-level information.
parray __tcl3dPkgInfo
```

2 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.

The mapping described in this chapter is consistently applied to all libraries wrapped with Tcl3D. It is optimized to work best with the OpenGL interface.

2.1 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.
- Function parameters declared as const pointers are interpreted as input parameters. Parameters declared as pointer are interpreted output parameters.

2.1.1 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`

Table 6: Wrapping of a scalar input parameter

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`
- `osg::StateAttribute::GLMode`
- `osg::StateAttribute::GLModeValue`
- `osg::StateAttribute::OverrideValue`
- `osg::Fog::Mode`

The mapping is explained using the 3 OpenGL enumeration types.

`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`

Table 7: Wrapping of a GLenum input parameter

`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`

Table 8: Wrapping of a GLbitfield input parameter

`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`

Table 9: Wrapping of a GLboolean input parameter

2.1.2 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`

Table 10: Wrapping of a pointer input parameter

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`

Table 11: Wrapping of a void pointer input parameter

Note:

- The allocation of usable memory can be accomplished with the use of the `tcl3dVector` commands, which are described in chapter 3.3.5.
- 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 3.3.6.

2.1.3 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`

Table 12: Wrapping of a pointer output parameter

2.1.4 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`

Table 13: Wrapping of a function return value

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.

2.1.5 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`

Table 14: Wrapping of GLU functions

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,`
`const GLdouble *model,`
`const GLdouble *proj,`
`const GLint *view,`
`GLdouble* objX,`
`GLdouble* objY,`
`GLdouble* objZ);`
	`GLint gluUnProject (`
	`GLdouble winX, GLdouble winY,`
	`GLdouble winZ,`
	`const GLdouble *model,`
	`const GLdouble *proj,`
	`const GLint *view,`
	`GLdouble* OUTPUT,`
	`GLdouble* OUTPUT,`
	`GLdouble* OUTPUT);`

Table 15: Wrapping exceptions for GLU

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.

- Tcl3D 0.4.3 adds two new functions `tcl3dOglProject` and `tcl3dOglUnProject` with the same signature as `gluProject` and `gluUnProject` with the following exception: The parameters for the viewport, the modelview matrix and the projection matrix can be specified as a `tcl3dVector`. See Redbook demo *`unprojectVec.tcl`* for an example.

2.2 Wrapping Reference Table

- 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.
`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.

Table 16: Tcl3D wrapping reference

3 Modules in Detail

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

3.1 Module tcl3dOgl: Sub-module Togl

Togl is a Tk widget with support to display graphics in an OpenGL context. 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.

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

3.1.1 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.

```
cget
configure
extensions
postredisplay
render
swapbuffers
loadbitmapfont
unloadbitmapfont
width
height
```

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.

3.1.2 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
-createcommand    -displaycommand     -reshapecommand
-coreprofile      -major              -minor
```

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

```
% package require tcl3d ; # or just package require tcl3dogl
1.0.0
% togl .t

% .t configure
{-height height Height 400 400} …
{ -displaycommand displayCommand CallbackCommand {} {}} …

% .t configure -displaycommand DisplayCallback
% .t configure -displaycommand
-displaycommand displayCommand CallbackCommand {} DisplayCallback
```

Callback procedures

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

-createcommand ProcName Procedure is called when a new widget is created.
-reshapecommand ProcName Procedure is called when the widget size is changed.
-displaycommand ProcName Procedure is called when the widget content needs to be redrawn.

Default settings are:

```
{-createcommand  createCommand  CallbackCommand {} {}}
{-displaycommand displayCommand CallbackCommand {} {}}
{-reshapecommand reshapeCommand CallbackCommand {} {}}
```

The callback procedures must have the following signatures:

```
proc CreateProc  { toglwin } { ... }
proc ReshapeProc { toglwin } { ... }
proc DisplayProc { toglwin } { ... }
```

Note:
Starting with Tcl3D version 0.5.0 the Togl version used in Tcl3D is based on version 2.0. The advantages are `Tcl_Obj` based callback functions and a cleaner code base. All callbacks now have the same signature, i.e. only the Togl widget identifier. This introduces an incompatibility with previous Tcl3D versions, where the Reshape callback function had additional parameters for the new width and height of the Togl window. See the Tcl3D demos or the example below for backward compatible changes to the Reshape callback.

Display options

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

Default settings are:

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

Note:
Multisampling was implemented for the Togl widget in Tcl3D version 0.3.2. If working with older versions of Tcl3D, you may enable 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.

The default value for `-swapinterval` was changed in version 0.4.0 from 1 to 0, i.e. if this option is not specified, a Tcl3D program does not wait for the vertical blank signal, but runs at maximum speed.
This functionality relies on the presence of the following extensions:
Windows: `WGL_EXT_swap_control`
Linux: `GLX_EXT_swap_control` , `GLX_MESA_swap_control` or `GLX_SGI_swap_control`
You can check the existence of these extensions in the OpenGL driver of your system using the OpenGL Information Center.

Profile options

-coreprofile	Enable/disable the OpenGL core profile.
-major	Set the OpenGL major number for the core profile.
-minor	Set the OpenGL minor number for the core profile.

Default settings are:

```
{-coreprofile coreProfile CoreProfile false 0}
{-major major Major 1 1}
{-minor minor Minor 0 0}
```

3.1.3 A simple Tcl3D template

A template for a Tcl3D application looks like follows:

```
package require tcl3d

proc CreateCallback { 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 ReshapeCallback { toglwin { w -1 } { h -1 } } {
    set w [$toglwin width]         ; # Get Togl window width
    set h [$toglwin height]        ; # Get Togl window height
    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
}

proc DisplayCallback { 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
}

frame .fr
pack .fr -expand 1 -fill both
# Create a Togl widget with a depth buffer and double buffering enabled.
togl .fr.toglwin -width 250 -height 250 \
                 -double true -depth true \
                 -createcommand  CreateCallback \
                 -reshapecommand ReshapeCallback \
                 -displaycommand DisplayCallback
grid .fr.toglwin -row 0 -column 0 -sticky news
```

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

3.2 Module tcl3dOgl: Sub-module OpenGL

This module wraps OpenGL functionality up to OpenGL Version 4.6, GLU library functions based on Version 1.2 and several OpenGL extensions.
It is implemented with the help of the GLEW library.
Standard shapes (box, sphere, cylinder, teapot, …) with a GLUT compatible syntax are supplied in this module, too.

Requirements for this module: An OpenGL driver suitable for your graphics card. It is recommended to download and install an up-to-date OpenGL driver from the manufacturer of your graphics card, especially if intending to write shader programs in GLSL.

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

OpenGL library

Implementation files:	tcl3dOglQuery.tcl, tcl3dOglUtil.tcl, tcl3dOglHelp.txt
Header files:	glew.h, glu.h
Wrapper files:	glew.i, glewautogen.i, glu.i

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

Note:
The original GLEW header file is not usable for direct wrapping with Swig, so its information is used for generating the wrapper files `glewdefs.i` and `glewfuncs.i` during the build process with Tcl script `createSwigAndHelpFile.tcl`.
File `tcl3dOglHelp.tcl` is automatically generated by script `createSwigAndHelpFile.tcl`.
See the `ReadmeGlew.txt` file in subdirectory `GLSpec` for more detailed information about the GLEW wrapping and update process.

The next figure shows the dependencies of the build process.

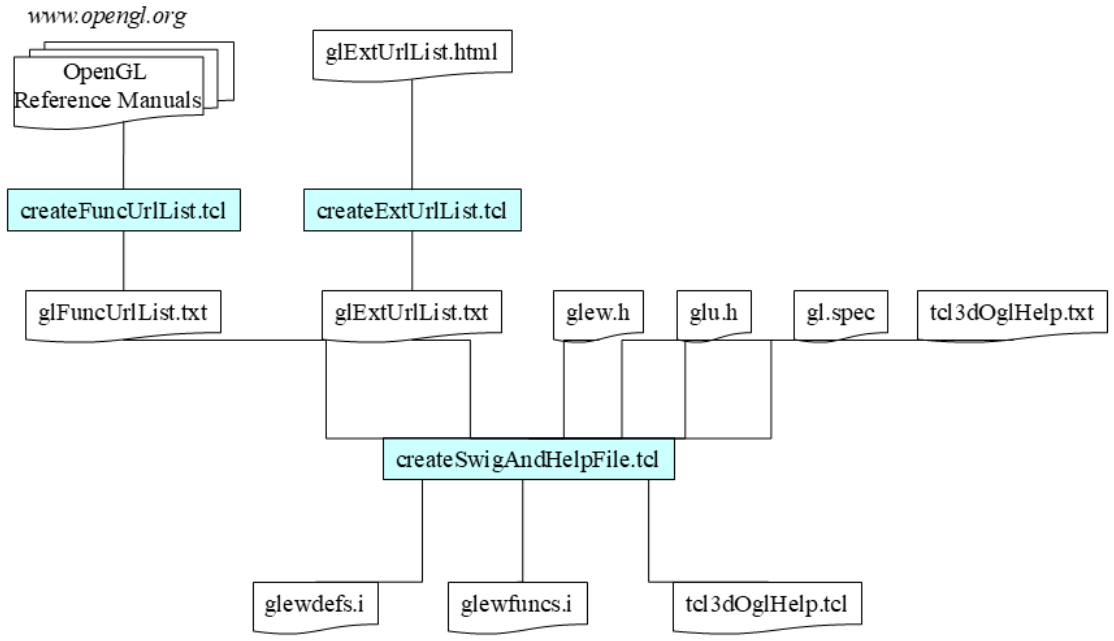


Figure 3: Build dependencies of the tcl3dOgl module

The following Tcl3D specific commands are implemented in this module:

Tcl3D command	Description
`tcl3dOglGetVersion`	Get the OpenGL version string.
`tcl3dOglGetVersionNumber`	Get the OpenGL version number as a dictionary.
`tcl3dOglGetGlewVersion`	Get the version of the used GLEW library.
`tcl3dOglGetProfile`	Get OpenGL profile settings.
`tcl3dOglHaveFunc`	Check availability of an OpenGL function in the OpenGL driver.
`tcl3dOglHaveExtension`	Check, if a given OpenGL extension is provided by the OpenGL implementation.
`tcl3dOglHaveVersion`	Check, if a specific OpenGL version is available.
`tcl3dOglGetVersions`	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.
`tcl3dOglGetExtensions`	Query the OpenGL library with the keys `GL_EXTENSIONS` and `GLU_EXTENSIONS` and return the results as a list of key-value pairs.
`tcl3dOglGetStates`	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). *Deprecated.*

Tcl3D command	Description
tcl3dOglGetIntState	Get OpenGL integer state variable.
tcl3dOglGetFloatState	Get OpenGL float state variable.
tcl3dOglGetDoubleState	Get OpenGL double state variable.
tcl3dOglGetMaxTextureSize	Get maximum texture size.
tcl3dOglGetMaxTextureUnits	Get maximum number of texture units.
tcl3dOglGetViewport	Get current viewport as a 4-element Tcl list.
tcl3dOglGetShaderState	Utility function for easier use of OpenGL function glGetShaderiv.
tcl3dOglGetProgramState	Utility function for easier use of OpenGL function glGetProgramiv.
tcl3dOglGetShaderInfoLog	Utility function for easier use of OpenGL function glGetShaderInfoLog.
tcl3dOglGetProgramInfoLog	Utility function for easier use of OpenGL function glGetProgramInfoLog.
tcl3dOglGetShaderSource	Utility function for easier use of OpenGL function glGetShaderSource.
tcl3dOglGetInfoLogARB	Utility function for easier use of OpenGL function glGetInfoLogARB.
tcl3dOglShaderSource	Utility function for easier use of OpenGL function glShaderSource.
tcl3dOglReadShaderFile	Read a shader file.
tcl3dOglCompileProgram	Compile a shader program.
tcl3dOglLinkProgram	Link a shader program.
tcl3dOglBuildProgram	Build a shader program.
tcl3dOglDestroyProgram	Destroy a shader program.
glMultiDrawElements	Procedure to implement the OpenGL function glMultiDrawElements.
tcl3dOglGetGlError	Check, if an OpenGL related error has been occurred.
tcl3dOglGetFuncList	Return a list of the names of all wrapped OpenGL functions.
tcl3dOglGetFuncSignatureList	Return a list of the C-signatures of all wrapped OpenGL functions. *Deprecated*. Use tclOglGetFuncSignature instead.
tcl3dOglGetFuncVersionList	Return a list of the OpenGL versions or extensions of all wrapped OpenGL functions. *Deprecated.* Use tclOglGetFuncVersion instead.
tcl3dOglGetVersionList	Get list of wrapped OpenGL versions and extensions.
tcl3dOglGetExtensionList	Get list of all OpenGL extensions.
tcl3dOglIsFuncWrapped	Check if OpenGL or GLU function is wrapped.
tcl3dOglGetFuncSignature	Get the signature of an OpenGL or GLU function.
tcl3dOglGetFuncVersion	Get the version or extension name of an OpenGL function.
tcl3dOglGetEnumVersion	Get the version or extension name of an OpenGL enumeration.
tcl3dOglGetFuncDeprecated	Get the OpenGL version, an OpenGL function has been declared deprecated.
tcl3dOglGetUrl	Get the URL of the official documentation of an OpenGL item.
tcl3dOglGetVersionFuncs	Get the function names of an OpenGL version or extension.

Tcl3D command	Description
tcl3dOglGetVersionEnums	Get the enumeration names of an OpenGL version or extension.
tcl3dOglGetExtSuffixes	Get list of allowed OpenGL extension suffixes.
tcl3dOglFindFunc	Find an OpenGL core or extension function.
tcl3dOglSetNormalMode	Set the execution mode of OpenGL functions to normal.
tcl3dOglSetSafeMode	Set the execution mode of OpenGL functions to safe.
tcl3dOglSetDebugMode	Set the execution mode of OpenGL functions to debug.
tcl3dOglSetMode	Set the execution mode of OpenGL functions.

Table 17: tcl3dOgl helper commands

Note:
The functions `glGetString` and `gluGetString` as well as the corresponding high-level functions `tcl3dOglGetVersions` and `tcl3dOglGetExtensions` only return correct values, if a Togl window has been created, i.e. a rendering context has been established. This holds true for function `tcl3dOglHaveFunc`, too.

GLUT shapes library

Implementation files:	tcl3dShapesGlut.c, tcl3dShapesTeapot.c, tcl3dShapesGlut.tcl
Header files:	tcl3dShapesGlut.h
Wrapper files:	tcl3dOgl.i

The shapes library consists of C files (*tcl3dShapesTeapot.c* for the teapot, *tcl3dShapesGlut.c* for all other GLUT shapes and the common header file *tcl3dShapesGlut.h*) and the Tcl file *tcl3dShapesGlut.tcl*.

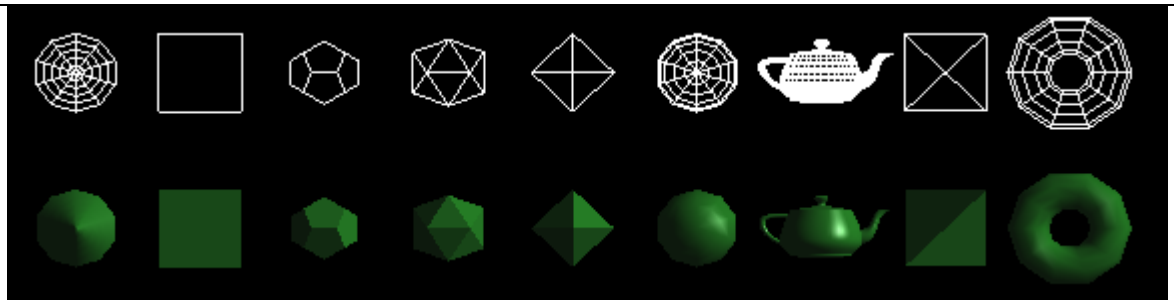
The GLUT shape objects are available under identical names for porting test and demonstration programs to Tcl3D. These shapes are used extensively in the examples of the OpenGL RedBook. See there for a description of the functions and its parameters.

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

Table 18: tcl3dOgl GLUT shape commands

Note:
The teapot implementation differs in the original GLUT and the freeglut implementation.
If using the teapot in a benchmark application, note that:

- Freeglut uses 7 for the grid parameter.
- Original GLUT and Tcl3D use 14 as grid parameter.



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

Examples

The following code snippet shows how to call `tcl3dOglGetVersions`.

```
package require tcl3d
togl .t

foreach glInfo [tcl3dOglGetVersions] {
    puts "[lindex $glInfo 0]: [lindex $glInfo 1]"
}

GL_VENDOR: NVIDIA Corporation
GL_RENDERER: NVIDIA GeForce RTX 3080 Laptop GPU/PCIe/SSE2
GL_VERSION: 4.6.0 NVIDIA 512.36
GLU_VERSION: 1.2.2.0 Microsoft Corporation
GL_SHADING_LANGUAGE_VERSION: 4.60 NVIDIA
GLEW_VERSION: 2.2.0
```

The following code snippet shows how to call `tcl3dOglGetExtensions`.

```
package require tcl3d
togl .t

foreach glInfo [tcl3dOglGetExtensions] {
    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
```

Note:
`tcl3dOglGetExtensions` lists the extensions supported by the running OpenGL driver. Use `tcl3dOglGetExtensionList` to get a list of all specified OpenGL extensions.

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

3.3 Module tcl3dOgl: Sub-module Util

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

- 3D vector and transformation matrix module
- Information module
- File utility module
- Color names module
- Large data module
- Image utility module
- Screen capture module
- Timing module
- Random number module
- 3D model and shapes module
- Virtual trackball and arcball module
- C/C++ based utilities for demo applications

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

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

3.3.1 3D vector and transformation matrix module

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

Overview

The following tables list the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

Tcl3D command	Description
tcl3dVec3Print	Print the contents of a 3D vector onto standard output.
tcl3dVec3fCompare	Compare two 3D vectors.
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	Normalise 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.
tcl3dVec3fPlaneNormal	Create a plane normal defined by three points.

Table 19: tcl3dUtil: 3D vector commands

Note:
All above listed 3D vector commands are available for single-precision (prefix `tcl3dVec3f`) and double-precision (prefix `tcl3dVec3d`) floating point numbers.

Tcl3D command	Description
tcl3dMatPrint	Print the contents of a matrix onto standard output.
tcl3dMatfCompare	Compare two transformation matrices.
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.
tcl3dMultMatrixf	Replacement function for glMultMatf.
tcl3dRotatef	Replacement function for glRotatef.
tcl3dScalef	Replacement function for glScalef.
tc3dTranslatef	Replacement function for glTranslatef.

Table 20: tcl3dUtil: Matrix commands

Note:
All above listed 4x4 matrix commands are available for single-precision (prefix `tcl3dMatf`) and double-precision (prefix `tcl3dMatd`) floating point numbers.

Tcl3D command	Description
tcl3dOrtho	Replacement function for glOrtho.
tcl3dFrustum	Replacement function for glFrustum.
tcl3dPerspective	Replacement function for gluPerspective.
tcl3dLookAt	Replacement function for gluLookAt.

Table 21: tcl3dUtil: View commands

Examples

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.

Implementation details

The functionality of this module is implemented in the following files:

Implementation files:	tcl3dVecMath.c, tcl3dViewMath.c, tcl3dVecMath.tcl
Header files:	tcl3dVecMath.h, tcl3dViewMath.h
Wrapper files:	util.i

3.3.2 Information module

This module provides convenience functions for querying Tcl3D package related information.

Overview

The following table lists the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

Tcl3D 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.
`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.
`tcl3dHaveOsg`	Check, if the OSG library has been loaded successfully.

Table 22: tcl3dUtil: Information commands

Examples

The following code snippet shows how to call `tcl3dGetPackageInfo`.

```
package require tcl3d
togl .t

foreach pkgInfo [tcl3dGetPackageInfo] {
    puts $pkgInfo
}

tcl3dftgl   1 1.0.0 2.1.3-rc5
tcl3dgauges 1 1.0.0 {}
tcl3dgl2ps  1 1.0.0 1.4.2
tcl3dogl    1 1.0.0 {4.6.0 NVIDIA 512.36}
tcl3dosg    1 1.0.0 3.4.1
tcl3dsdl    1 1.0.0 2.26.2
```

Implementation details

The functionality of this module is implemented in the following files:

Implementation files:	tcl3dUtilInfo.tcl
Header files:	None
Wrapper files:	None

3.3.3 File utility module

This module provides miscellaneous functions for file related tasks: Handling of temporary directories and file access from a starpack.

Overview

The following table lists the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

Tcl3D command	Description
`tcl3dGetTmpDir`	Get the name of a temporary directory.
`tcl3dCreateTmpDir`	Create a unique temporary directory.
`tcl3dGenExtName`	Create a name on the file system. Use this function, if writing to a file from a script, which may be running from within a starpack.
`tcl3dGetExtFile`	Get a name on the file system. Use this function, if a file is needed for reading from an external Tcl3D library, like font files used by FTGL, or shader files, and the script may be executed from within a starpack.

Table 23: tcl3dUtil: File utility commands

Examples

See the demo program *`Lesson02.tcl`* for an example usage of `tcl3dGenExtName`, and demo *`ftglTest.tcl`* for an example usage of `tcl3dGetExtFile` .

Implementation details

The functionality of this module is implemented in the following files:

Implementation files:	tcl3dUtilFile.tcl
Header files:	None
Wrapper files:	None

3.3.4 Color names module

This module provides miscellaneous functions for handling color specifications in Tcl and OpenGL style.

Overview

The following table lists the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

Tcl3D command	Description
`tcl3dGetColorNames`	Return a list of all supported Tcl color names.
`tcl3dFindColorName`	Check, if supplied color name is a valid Tcl color name.
`tcl3dName2Hex`	Convert a Tcl color name into the corresponding hexadecimal representation: #RRGGBB
`tcl3dName2Hexa`	Convert a Tcl color name into the corresponding hexadecimal representation: #RRGGBBAA

Tcl3D command	Description
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
tcl3dRgb2Name	Convert an OpenGL RGB color representation into a hexadecimal Tcl color name string. OpenGL colors are specified as unsigned bytes in the range [0..255].
tcl3dRgba2Name	Convert an OpenGL RGBA color representation into a hexadecimal Tcl color name string. OpenGL colors are specified as unsigned bytes in the range [0..255].
tcl3dRgbf2Name	Convert an OpenGL RGB color representation into a hexadecimal Tcl color name string. OpenGL colors are specified as floats in the range [0..1].
tcl3dRgbaf2Name	Convert an OpenGL RGBA color representation into a hexadecimal Tcl color name string. OpenGL colors are specified as floats in the range [0..1].

Table 24: tcl3dUtil: Color utility commands

Examples

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

```
[tcl3dName2Hex white]  returns "#FFFFFF"
[tcl3dName2Hexa white] returns "#FFFFFFFF"

[tcl3dName2rgb white] returns {255 255 255}
[tcl3dRgb2Name 255 255 255] returns "#FFFFFF"

[tcl3dName2rgba white] returns {255 255 255 255}
[tcl3dRgba2Name 255 255 255 255] returns "#FFFFFFFF"

[tcl3dName2rgbf white] returns {1.0 1.0 1.0}
[tcl3dRgbf2Name 1.0 1.0 1.0] returns "#FFFFFF"

[tcl3dName2rgbaf white] returns {1.0 1.0 1.0 1.0}
[tcl3dRgbaf2Name 1.0 1.0 1.0 1.0] returns "#FFFFFFFF"

[tcl3dName2rgb "#0a0c0e"] returns {10 12 14}
```

Implementation details

The functionality of this module is implemented in the following files:

Implementation files:	tcl3dUtilColors.tcl
Header files:	None
Wrapper files:	None

3.3.5 Large data module

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

Overview

The following table lists the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

Tcl3D command	Description
`tcl3dVector`	Create a new Tcl3D Vector by calling the low-level memory allocation routine new_TYPE and create a new Tcl procedure. (See example below).
`tcl3dVectorInd`	Get index of a Tcl3D Vector.
`tcl3dVectorPrint`	Print the contents of a Tcl3D Vector onto standard output.
`tcl3dVectorFromArgs`	Create a new Tcl3D Vector from a variable argument list.
`tcl3dVectorFromList`	Create a new Tcl3D Vector from a Tcl list.
`tcl3dVectorFromString`	Create a new Tcl3D Vector from a Tcl string.
`tcl3dVectorFromByteArray`	Create a new Tcl3D Vector from a Tcl binary string.
`tcl3dVectorFromPhoto`	Create a new Tcl3D Vector containing the data of a Tk photo image. See next chapter for detailed description.
`tcl3dVectorFromLinspace`	Create a new linearly spaced Tcl3D Vector.
`tcl3dVectorToList`	Copy the contents of a Tcl3D Vector into a Tcl list.
`tcl3dVectorToString`	Copy the contents of a Tcl3D Vector into a string.
`tcl3dVectorToByteArray`	Copy the contents of a Tcl3D Vector into a Tcl binary string.

Table 25: tcl3dUtil: tcl3dVector utility commands

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`)
`setrgb`	Set 3 vector elements starting at given index. (`TYPE_setrgb`)
`setrgba`	Set 4 vector elements starting at given index. (`TYPE_setrgba`)
`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`)
`elemsize`	Determine the size of an element in bytes. (`TYPE_elemsize`)

Table 26: tcl3dUtil: tcl3dVector subcommands

Examples

The following example shows the usage of the `tcl3dVector` command.

```
set ind 23
set vec [tcl3dVector GLfloat 123] ; # Create Vector of 123 GLfloats
$vec set $ind 1017.0              ; # Set element at index 23 to 1017.0
set x [$vec get $ind]             ; # Get element at index 23
puts [$vec elemsize]              ; # Prints out 4
$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.

The following example shows the usage of the `tcl3dVectorFromLinspace` command.

```
# Create a GLdouble vector of length 5 with values from 0 to 0.1

> set v [tcl3dVectorFromLinspace GLdouble 0 0.1 5]
> tcl3dVectorPrint $v 5

   0:  0.000
   1:  0.025
   2:  0.050
   3:  0.075
   4:  0.100
```

See also the test program *`vectorlinspace.tcl`* for more examples.

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

Implementation details

The functionality of this module is implemented in the following files:

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

As stated in chapter *2.1.2 Pointer input parameters*, 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 allocated memory.

Wrapper functions for the following scalar types are defined in file *`tcl3dVectors.i`*:

Array of	**is mapped to**
`short`	`short`
`int`	`int`
`ushort`	`unsigned short`
`uint`	`unsigned int`

Array of	is mapped to
float	float
double	double
GLenum	unsigned int
GLboolean	unsigned char
GLbitfield	unsigned int
GLbyte	signed char
GLshort	short
GLint	int
GLint64	int
GLintptr	int
GLsizei	int
GLsizeiptr	int
GLubyte	unsigned char
GLushort	unsigned short
GLuint	unsigned int
GLfloat	float
GLclampf	float
GLdouble	double
GLclampd	double

Note:
tcl3dVectors of type `char`, `unsigned char`, `GLchar` and `GLcharARB` are not supported, because the corresponding typemaps would collide with the standard SWIG mapping for C strings. Use types `GLbyte` and `GLubyte`, if you need tcl3dVectors with element sizes of 1 byte.

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 int GLdouble_elemsize(double *ary) {
    return sizeof (ary[0]);
}

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_setrgb(double *ary, int index,
                            double r, double g, double b) {
    ary[index]   = r;
    ary[++index] = g;
    ary[++index] = b;
}

static void GLdouble_setrgba(double *ary, int index,
                             double r, double g, double b, double a) {
    ary[index]   = r;
    ary[++index] = g;
    ary[++index] = b;
    ary[++index] = a;
}

static void GLdouble_setvector(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_addvector(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_mulvector(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);
}

static double *GLdouble_cast(void *ary) {
    return (double *)ary;
}
```

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 optimised 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`).

Comparison of the different vector methods

There are four different methods of setting vectors.

Method 1: `$vec set $index $val`
Set the elements with the tcl3dVector object method `set`. Most elegant way, but also the slowest. Only useful for small vectors.

Method 2: `${type}_setitem $vec $index $val`
Set the elements with the tcl3dVector low-level function `setitem`. Not so elegant, because you need to know the type of the vector, but much faster than method 1.

Method 3: `tcl3dListToVector_$type $list $vec $len`
Set the elements with the low-level functions `tcl3dListToVector_TYPE` introduced in Tcl3D 0.3.3. Not so elegant, because you need to know the type of the tcl3dVector and you have to build a Tcl list before setting the tcl3dVector. This is the fastest way.

Method 4: `set vec [tcl3dVectorFromList $type $list]`

Set the elements with the utility function `tcl3dVectorFromList`, which internally calls the low-level functions `tcl3dListToVector_TYPE`. You do not have to care about allocating a tcl3dVector of appropriate size. This is only slightly slower than Method 3.

The test program *`vectorspeed.tcl`* implements the above mentioned four different methods and shows output similar to the following lines:

```
D:\tcl3d\tcl3dOgl\tests> tclsh vectorspeed.tcl
Number of runs : 100
Size of vectors: 1000
Setting 100000 elements per method.
SetMethod1: 2163.2 microseconds per iteration
SetMethod2:  289.6 microseconds per iteration
SetMethod3:   67.6 microseconds per iteration
SetMethod4:   76.5 microseconds per iteration
```

3.3.6 Image utility module

This module provides access to Tk photo images.

Overview

The following table lists the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

Tcl3D command	Description
`tcl3dPhotoChans`	Return the number of channels of a Tk photo.
`tcl3dVectorToPhoto`	Copy from OpenGL raw image format into a Tk photo. The photo image must have been initialized with the appropriate size and type.
`tcl3dPhotoToVector`	Copy a Tk photo into a `tcl3dVector` in OpenGL raw image format. The `tcl3dVector` must have been allocated 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.

Table 27: tcl3dUtil: Image utility commands

Note:
The ***Img*** extension is recommended to have access to lots of image formats.

Examples

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

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

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]
    tcl3dVectorToPhoto $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 ReshapeCallback { toglwin } {
    set ::toglWidth  [$toglwin width]
    set ::toglHeight [$toglwin height]
    ...
}
```

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 the photo image utilities.

Implementation details

The functionality of this module is implemented in the following files:

Implementation files:	tcl3dVector.tcl
Header files:	None
Wrapper files:	tkphoto.i

3.3.7 Screen capture module

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

Overview

The following table lists the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

Tcl3D 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.
`tcl3dDesktop2Clipboard`	Copy the contents of the whole desktop (PrtSc) into the Windows clipboard.
`tcl3dWindow2Img`	Copy the contents of the top-level window (Alt-PrtSc) into a photo image.
`tcl3dWindow2File`	Copy the contents of the top-level window (Alt-PrtSc) into a photo image and save the image to a file.

Table 28: tcl3dUtil: Capture commands

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.

Examples

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

Implementation details

The functionality of this module is implemented in the following files:

Implementation files:	tcl3dUtilCapture.tcl
Header files:	None
Wrapper files:	None

3.3.8 Timing module

This module provides functions for timing purposes.

Overview

The following table lists the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

Tcl3D command	Description
`tcl3dNewSwatch`	Create a new stop watch and return its 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.

Table 29: tcl3dUtil: Stop watch commands

Examples

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

Implementation details

The functionality of this module is implemented in the following files:

Implementation files:	tcl3dUtilStopWatch.c
Header files:	tcl3dUtilStopWatch.h
Wrapper files:	util.i

3.3.9 Random number module

This module provides functions to generate random numbers.

Overview

The following table lists the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

Tcl3D command	Description
`tcl3dNewRandomGen`	Initialize a new random number generator.
`tcl3dDeleteRandomGen`	Delete a random number generator.
`tcl3dGetRandomInt`	Generate a pseudo-random integer number.
`tcl3dGetRandomFloat`	Generate a pseudo-random floating-point number.

Table 30: tcl3dUtil: Random number commands

Examples

See the demo program *`mandelbrot.tcl`* for an example, on how to use these procedures to set up random colors for fractal generation.

Implementation details

The functionality of this module is implemented in the following files:

Implementation files:	tcl3dUtilRandom.c
Header files:	tcl3dUtilRandom.h
Wrapper files:	util.i

3.3.10 3D model and shapes module

This module provides functions for reading 3D models in Wavefront format and creating basic shapes.

Overview

The following tables list the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

Tcl3D 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.

Table 31: tcl3dUtil: Wavefront reader commands

Tcl3D 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.
`tcl3dCameraModel`	Draw a model of a simple camera.
`tcl3dTeapotModel`	Draw a teapot with quads.

Table 32: tcl3dUtil: Shape commands

Examples

See the demo program *`gaugedemo.tcl`* for an example on how to use the Wavefront parser functions.
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`.

Implementation details

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

Implementation files:	tcl3dModel.c, tcl3dModelFmtObj.c, tcl3dShapesMisc.c
Header files:	tcl3dModel.h, tcl3dModelFmtObj.h, tcl3dShapesMisc.h
Wrapper files:	util.i

Note:
The standard GLUT shapes are described in chapter *3.2 Module tcl3dOgl: Sub-module OpenGL*.

3.3.11 Virtual trackball and arcball module

This module provides functions for emulating a trackball and an arcball.

Overview

The following tables list the available functions of this module. For a detailed description of the functions see the Tcl3D Reference Manual or the source code files as listed in section *Implementation details* at the end of this chapter.

The trackball module implements the following commands:

Tcl3D 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.

Table 33: tcl3dUtil: Trackball commands

The ArcBall module implements the following commands:

Tcl3D 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.

Table 34: tcl3dUtil: ArcBall commands

Examples

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

Implementation details

The functionality of the trackball module is implemented in the following files:

Implementation files:	tcl3dUtilTrackball.c, tcl3dUtilTrackball.tcl
Header files:	tcl3dUtilTrackball.h
Wrapper files:	util.i

The functionality of the ArcBall module is implemented in the following files:

Implementation files:	tcl3dUtilArcBall.c
Header files:	tcl3dUtilArcBall.h
Wrapper files:	util.i

3.3.12 C/C++ based utilities for demo applications

This sub-module implements C/C++ based utility functions for some of the demo applications.

Overview

tcl3dDemoOglLogo implements an animated 3-dimensional OpenGL logo.
It is used in demo *animlogo.tcl* in directory *LibrarySpecificDemos/tcl3dOgl*.

tcl3dDemoReadRedBookImg 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*.

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

Implementation details

The functionality of the OpenGL logo animation is implemented in the following files:

Implementation files:	tcl3dDemoOglLogo.c
Header files:	tcl3dDemoOglLogo.h
Wrapper files:	util.i

The functionality of the RedBook image parser module is implemented in the following files:

Implementation files:	tcl3dDemoReadRedBookImg.c
Header files:	tcl3dDemoReadRedBookImg.h
Wrapper files:	util.i

The functionality of the heightmap module is implemented in the following files:

Implementation files:	heightmap.i, tcl3dDemoHeightMap.tcl
Header files:	None
Wrapper files:	heightmap.i

3.4 Module tcl3dGauges

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.

3.5 Module tcl3dGl2ps

This module wraps the GL2PS library based on version 1.4.2 and adds some GL2PS related utility procedures.
GL2PS is a C library providing high quality vector output (PostScript, PDF, SVG) for any OpenGL application. It does not support textures.

This is an optional module and contained in the Tcl3D-Basic distribution.

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

The master SWIG file for wrapping the Gl2ps library is *tcl3dGl2ps.i*.

Implementation files:	gl2ps.c, tcl3dGl2psQuery.tcl, tcl3dGl2psUtil.tcl
Header files:	gl2ps.h
Wrapper files:	gl2ps.i

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

Gl2ps utility module

Tcl3D command	Description
`tcl3dGl2psGetVersion`	Get the version of the wrapped GL2PS library.
`tcl3dGl2psCreatePdf`	Create a PDF file from current Togl window content.

Table 35: tcl3dGl2ps utility commands

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.

3.6 Module tcl3dFTGL

This module wraps the FTGL library based on version 2.1.3 RC5 and adds some FTGL related utility procedures. The FTGL library itself depends on the Freetype2 library.

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 and contained in the Tcl3D-Full distribution.

Requirements for this module: The FTGL and Freetype2 library and header files.

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

Implementation files:	tcl3dFTGLQuery.tcl, tcl3dFTGLUtil.tcl
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

Tcl3D command	Description
tcl3dFTGLGetVersion	Get the version of the wrapped FTGL library.
tcl3dFTGLGetBBox	Get bounding box of a string.

Table 36: tcl3dFTGL utility commands

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

3.7 Module tcl3dOsg

This module wraps the OpenSceneGraph library based on version 3.4.1 and adds some OSG related utility procedures.

This is an optional module and contained in the Tcl3D-Full distribution.

Requirements for this module: The OSG library and header files.

Implementation files:	tcl3dOsg*.tcl
Header files:	All files in OpenSceneGraph subdirectory include
Wrapper files:	osg*.i, tcl3dOsg*.i

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

OSG utility module

Tcl3D command	Description
tcl3dOsgGetVersion	Get the version of the wrapped OSG library.
tcl3dOsgKeysym	Convert a keysym into decimal and vice versa.

Tcl3D command	Description
`tcl3dOsgGetBitmap`	Get the bitmap image of a node type.
`tcl3dOsgVecPrint`	Print the contents of an osg::Vec* class.
`tcl3dOsgMatPrint`	Print the contents of an osg::Matrix* class.
`tcl3dOsgBBoxPrint`	Print the contents of an osg::BoundingBox* class.
`tcl3dOsgBSpherePrint`	Print the contents of an osg::BoundingSphere* class.
`tcl3dOsgVecArrayPrint`	Print an array of vectors.
`tcl3dOsgScalarArrayPrint`	Print an array of scalars.
`tcl3dOsgObjectArrayPrint`	Print an array of objects.
`tcl3dOsgGetVisitorTypeName`	Get visitor type name.
`tcl3dOsgGetTraversalModeName`	Get traversal mode name.

`tcl3dOsgSendButtonPress`
`tcl3dOsgSendButtonRelease`
`tcl3dOsgSendMouseMotion`
`tcl3dOsgSendKeyPress`
`tcl3dOsgSendKeyRelease`
`tcl3dOsgSendWindowResize`
	Procedures to transfer the corresponding Tk event to the OSG event queue (osgGA::EventQueue).

`tcl3dOsgAddTrackballBindings`	Add OS independent mouse bindings for trackball usage.

Table 37: tcl3dOsg utility commands

See the demo programs contained in directory *`demos/OpenSceneGraph`* for examples, on how to use the OSG functions.

3.8 Module tcl3dSDL

This module wraps the Simple DirectMedia Layer library based on version 2.26.2 and adds some SDL related utility procedures.

Note:
Currently only the functions related to joystick handling have been wrapped and tested.

This is an optional module and contained in the Tcl3D-Full distribution.

Requirements for this module: The SDL library and header files.

The master SWIG file for wrapping the SDL library is *`tcl3dSDL.i`*.

Implementation files:	tcl3dSDLQuery.tcl, tcl3dSDLUtil.tcl
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

Tcl3D command	Description
`tcl3dSDLGetVersion`	Get the version of the wrapped SDL library.
`tcl3dSDLGetFocusName`	Convert an SDL focus state bitfield into a string representation.
`tcl3dSDLGetButtonName`	Convert an SDL button state bitfield into a string representation.

Tcl3D command	Description
`tcl3dSDLGetHatName`	Convert SDL hat related enumerations into a string representation.
`tcl3dSDLGetEventName`	Convert SDL event related enumerations into a string representation.

Table 38: tcl3dSDL utility commands

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

4 Miscellaneous Tcl3D Information

This chapter contains miscellaneous information about ***Tcl3D***.

4.1 Programming Hints

Hint 1:

Most OpenGL examples written in C use the OpenGL immediate mode. As Tcl is a scripted language and each OpenGL call has to go through the wrapper interface, it is 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.
Also note, that immediate mode and display lists are marked deprecated in OpenGL 3.1.

Hint 2:

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 all defined independently of the underlying OpenGL implementation. Use the utility functions `tcl3dHaveVersion` and `tcl3dHaveExtension` instead.

Hint 3:

```
Error: expected integer but got "GL_REPEAT"
```

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 in Tcl:

```
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 the above error message.

Hint 4:

```
Error: expected floating-point number but got "_08201905_p_float".
```

This error message indicates, that a `tcl3dVector` has been specified as parameter to a function, which expects a Tcl list. This often happens, when using one of the `glMultMatrix` commands. Use a sequence like shown below to convert the `tcl3dVector` into a Tcl list before supplying it to the function:

```
set matAsList [tcl3dVectorToList $mat 16]
glMultMatrixf $matAsList
```

Hint 5:

```
Error: Package tcl3dftgl: couldn't load library
"C:/Tcl/lib/tcl3d/tcl3dFTGL/tcl3dFTGL.dll": this library or a dependent library could
not be found in library path
```

This typically indicates that the dependent library or libraries (ex. *ftgl.dll*) are not found, i.e. they are not in a directory contained in your Path environment variable.

```
Error: Package tcl3dftgl: couldn't load library
"C:/Tcl/lib/tcl3d/tcl3dFTGL/tcl3dFTGL.dll": permission denied
```

This typically indicates that the dependent library or libraries (ex. *ftgl.dll*) were found, but you do not have the permissions to execute the library.

These errors may occur with the following Tcl3D modules:

Tcl 3D module	Affected libraries
tcl3dFTGL	*ftgl.dll libfreetype-6.dll*
tcl3dOsg	*osg.dll, osgDB.dll, ...*
tcl3dSDL	*SDL.dll*

Although the examples shown in this hint use Windows specific library names, the above-mentioned errors may occur on Unix systems as well.

Hint 6:

The OpenGL extension library ***OglExt*** used in Tcl3D versions before 0.4 for wrapping OpenGL functions and the currently used ***GLEW*** library have an important difference:
OpenGL functions not available in the installed OpenGL driver have been ignored by the ***OglExt*** library, i.e. transformed into a no-op. The disadvantage of this behaviour was, that you did not get any feedback about not available functions in your OpenGL driver implementation.
With ***GLEW*** you will get a core dump, when trying to use such a function, because the function pointer is NULL. You should therefore always check either the OpenGL version implemented in your driver (`tcl3dOglHaveVersion`), the availability of the extensions you intend to use (`tcl3dOglHaveExtension`), or to be absolutely sure, check the availability of each OpenGL function in your initialization code (`tcl3dOglHaveFunc`).
Starting with Tcl3D version 0.4.1 the utility procedure `tcl3dSetSafeMode` can be used to avoid core dumps and to get information about which OpenGL functions are not available in your driver.
Use the OpenGL Information Center program `OglInfo.tcl` to get information about the supported OpenGL functions of your installed OpenGL driver.

Hint 7:

The demo applications are using a procedure called `Cleanup` to remove or unset demo data. This procedure is only needed when running the demo from the presentation framework. Do not put `Cleanup` into the exit procedure `ExitProg`, because an `after` handler might still be running in the presentation framework. `Cleanup` is not needed if running a demo application as a standalone script.

4.2 Open Issues

- GLU callbacks are currently not supported. This implies, that tessellation 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.

Note:
The open issues list is considered obsolete with OpenGL versions greater than 3.2:
- Tessellation can be done better and faster with Geometry Shaders.
- Indexed color mode has been declared deprecated.

4.3 Known Bugs

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.

4.4 Starpack Issues

This chapter describes some issues, if ***Tcl3D*** should be used inside of a Starpack.

4.4.1 Starpack issue #1

If shipping external libraries with your Starpack (ex. *ftgl.dll*), you have to copy them to the file system, before they can be used. A convenient place is the directory containing the Starpack.

```
# Check if all necessary external libraries exist 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 a Starpack onto the desktop. Starting the Starpack inflates the desktop with possibly lots of DLL files.
- On Unix the current directory typically is not included in the LD_LIBRARY_PATH variable.

4.4.2 Starpack issue #2

Some of the external libraries need files for initialization, ex. the FTGL library needs the name of a TrueType font file to construct its 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 chapter *3.3.3 File utility module* for a description of the Starpack related file utilities.

A typical usage is shown in the following code segment:

```
set fontfile [file join [file dirname [info script]] "Vera.ttf"]
# Get the font file in a Starpack independent way.
set fontfile [tcl3dGetExtFile $fontfile]
```

4.5 OpenGL Deprecation Mode

The next table shows the release history of OpenGL versions with short comments regarding the introduction of new features.

Version	Release date	New features
OpenGL 1.0	01/1992	First release based on SGI's IRIS GL.
OpenGL 1.1	01/1997	Textures and texture formats on GPU hardware.
OpenGL 1.2	03/1998	Volume textures, packed pixels, normal rescaling, image processing.
OpenGL 1.2.1	10/1998	Multi-textures.
OpenGL 1.3	08/2001	Cube map texture, multi-sampling, texture unit combine operations.
OpenGL 1.4	07/2002	Hardware shadowing support, fog coordinates, automatic mipmap generation.
OpenGL 1.5	07/2003	Vertex buffer objects, occlusion queries, extended shadowing functions.
OpenGL 2.0	09/2004	User-programmable shaders. OpenGL Shading Language (**GLSL**).
OpenGL 2.1	07/2006	Pixel buffer objects, sRGB textures, non-square matrices. GLSL 1.20.
OpenGL 3.0	07/2008	Frame buffer objects, hardware instancing, vertex array objects. GLSL 1.30. Introduction of a **deprecation mechanism**.
OpenGL 3.1	03/2009	Texture Buffer Objects, Uniform Buffer Objects. GLSL 1.40. Removal of **legacy functionality.**
OpenGL 3.2	08/2009	Geometry Shader, Sync and Fence objects. GLSL 1.50.
OpenGL 3.3	03/2010	Backport of OpenGL 4.0 functionality for use on previous GPU HW. GLSL 3.30.
OpenGL 4.0	03/2010	Shader subroutines, 64-bit floating point support. GLSL 4.00.
OpenGL 4.1	07/2010	OpenGL ES compatibility. GLSL 4.10.
OpenGL 4.2	08/2011	Shaders with atomic counters, draw transform feedback instanced, shader packing, performance improvements. GLSL 4.20.
OpenGL 4.3	08/2012	Compute shaders. GLSL 4.30.
OpenGL 4.4	07/2013	Buffer placement control, Efficient asynchronous queries and more. GLSL 4.40.

Version	Release date	New features
OpenGL 4.5	08/2014	OpenGL ES 3.1 API and Shader compatibility. DX11 emulation features. GLSL 4.50.
OpenGL 4.6	07/2017	Support of SPIR-V and anisotropic filtering.. GLSL 4.60.

Table 39: OpenGL release timeline

Starting with OpenGL 3.1 the functionality of the fixed-function pipeline was declared deprecated. About 65% of all existing OpenGL functions have been declared deprecated, as can be seen in the following table.

Note:
The following table only takes the core functions into account, no OpenGL extension functions. The table is based on information contained in file *glew.h*, which may not be identical with the OpenGL specification.

OpenGL version	Number of functions	Number of deprecated functions
OpenGL 1.1	336	272
OpenGL 1.2	4	1
OpenGL 1.3	46	37
OpenGL 1.4	47	38
OpenGL 1.5	19	0
OpenGL 2.0	93	36
OpenGL 2.1	6	0
OpenGL 3.0	55	20
OpenGL 3.1	4	0
OpenGL 3.2	3	0
OpenGL 3.3	0	0
OpenGL 4.0	5	0
OpenGL 4.1	0	0
OpenGL 4.2	0	0
OpenGL 4.3	0	0
OpenGL 4.4	0	0
OpenGL 4.5	4	0
OpenGL 4.6	5	0
Total	**618**	**404**

Table 40: Deprecated OpenGL functions

The deprecated functions can be easily determined with the Tcl3D utility procedure `tcl3dOglGetFuncDeprecated`. The following example script prints out all deprecated functions:

```
foreach f [tcl3dOglGetFuncList] {
    set depr [tcl3dOglGetFuncDeprecated $f]
    if { $depr ne "0.0" } {
        puts "Deprecated since OpenGL version $depr : $f"
    }
```

```
}

Output:
Deprecated since OpenGL version 3.1 : glAccum
Deprecated since OpenGL version 3.1 : glAlphaFunc
Deprecated since OpenGL version 3.1: glAreTexturesResident
Deprecated since OpenGL version 3.1: glArrayElement
Deprecated since OpenGL version 3.1: glBegin
...
```

The next table contains an overview of deprecated functions sorted into categories.
See the *OpenGL 4.6 API Reference Guide* (https://www.khronos.org/files/opengl46-quick-reference-card.pdf) for further details.

Category	Examples
Vertex Specification	`glBegin, glEnd, glVertex, glNormal, glColor, glTexCoord`
Vertex Arrays	`glVertexPointer, glEnableClientState, glArrayElement`
Display Lists	`glNewList, glCallList, glGenLists, glListBase`
Lighting	`glMaterial, glLight, glLightModel, glShadeModel`
Texturing	`glTexEnv, glGetTexEnv, glBitmap`
Matrix operations	`glLoadMatrix, glMultMatrix, glTranslate, glRotate`
Raster operations	`glDrawPixels, glCopyPixels, glPixelTransfer, glRasterPos`
State	`glPushAttrib, glPushClientAttrib, glPosAttrib`
Fog and Clipping	`glFog, glClipPlane`
Evaluation	`glMap, glMapGrid, glEvalCoord, glEvalMesh`
Selection and Feedback	`glInitNames, glLoadName, glFeedbackBuffer, glPassThrough`
Convolution filters	`glConvolutionFilter2D, glConvolutionParameter`
ColorTables	`glColorTable, glCopyColorTable, glGetColorTable`

Table 41: Deprecation categories

The deprecated functions are still supported by the OpenGL drivers. Typically, these operate in the so-called Compatibility Profile, where all functionality of the fixed-function pipeline is still available. You have to explicitly switch to the OpenGL core profile by using the extension `WGL/GLX_ARB_create_context`.

Note:
This extension functionality is available via the Togl command line `-coreprofile`.

For the deprecated matrix functions, replacement functions have been integrated into the OpenGL module of Tcl3D.

All functions have the same signature, with the exception, that the Tcl3D functions have an additional parameter at the end for returning the calculated matrix:

```
void gluPerspective   (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)
void tcl3dPerspective (double   fovy, double   aspect, double   zNear, double   zFar, float *res)
```

Deprecated function	tcl3dOgl function
Viewing related functions	
glFrustum	tcl3dFrustum
glOrtho	tcl3dOrtho
gluPerspective	tcl3dPerspective
gluLookAt	tcl3dLookAt
Modelling related functions	
glRotate[f|d]	tcl3dRotate[f|d]
glTranslate[f|d]	tcl3dTranslate[f|d]
glScale[f|d]	tcl3dScale[f|d]
glLoadIdentity	tcl3dMatfIdentity, tcl3dMatdIdentity
glMultMatrix[f|d]	tcl3dMatfMult[f|d]
glLoadMatrix[f|d]	No replacement, use glUniformMatrix* functions instead
glMatrixMode	No replacement.
glPushMatrix	No replacement.
glPopMatrix	No replacement.

Table 42: Tcl3D replacement functions

5 Demos and Applications

This chapter describes the Tcl3D demonstration programs and applications.

5.1 Demonstration programs

More than 200 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 https://www.tcl3d.org/demos.html.

The demo applications can be executed by starting the `presentation.tcl` script at the root of the demo hierarchy.

The Tcl3D presentation is divided into 3 menus:
- Information
- 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 help and documentation menu gives you some online information about how to use the Tcl3D presentation framework.

The Tcl3D demos and tutorials menu is divided into the following categories:
- Category **Tutorials and books** contains scripts, which have been converted from C/C++ to Tcl3D, coming from the following sources:
  - OpenGL Red Book
  - NeHe tutorials
  - Kevin Harris CodeSampler web site (not existent anymore)
  - Vahid Kazemi's GameProgrammer page (not existent anymore)
  - Nopper's OpenGL core profile demos
- 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.
- Category **OpenSceneGraph** contains scripts demonstrating the OpenSceneGraph functionality of Tcl3D.

5.2 OpenGL Information Center

The OpenGL Information Center supplies lots of information an OpenGL developer (both C/C++ and Tcl) might need.

Information about installed OpenGL driver

The *GL information* category shows general information about the installed OpenGL and Tcl3D versions.

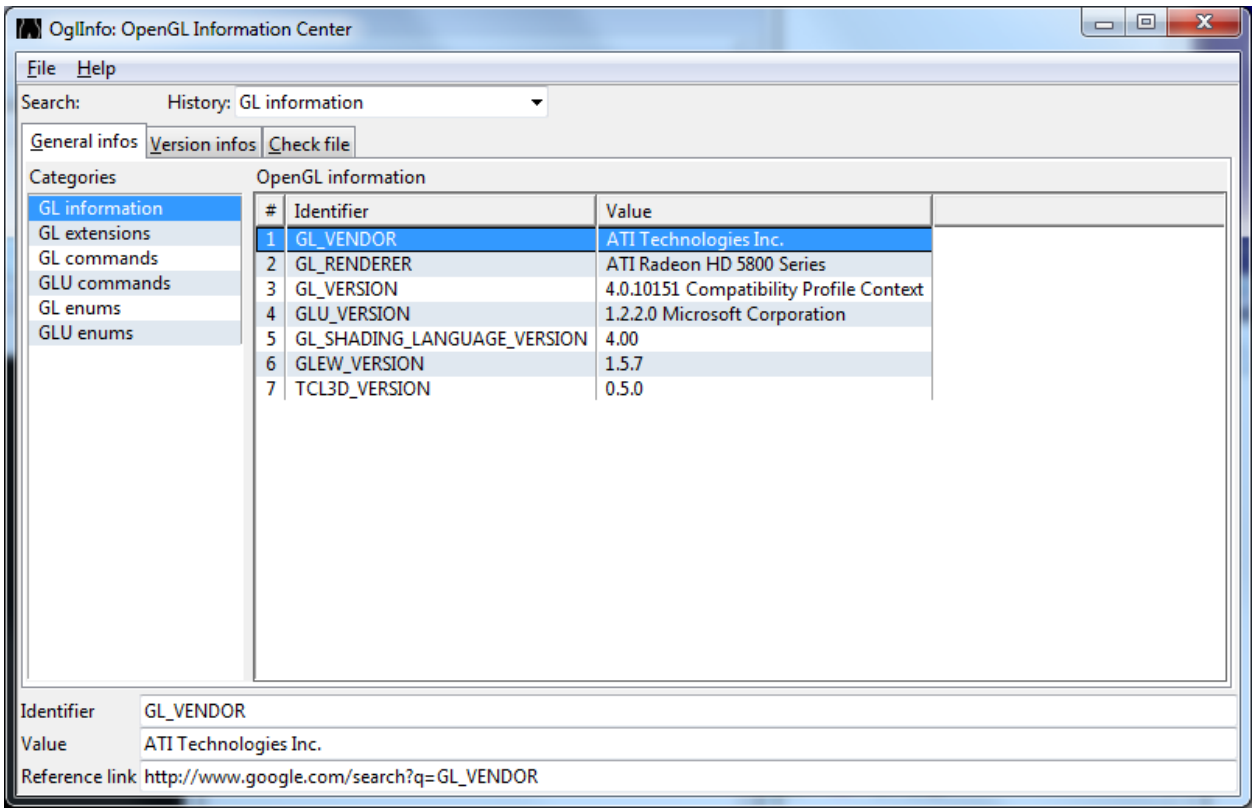


Figure 4: OglInfo category: GL information

Information about OpenGL extensions

The *GL extensions* category shows the following information about each OpenGL extension:

- Information about availability on underlying hardware and driver.
- Information about availability with used GLEW version.

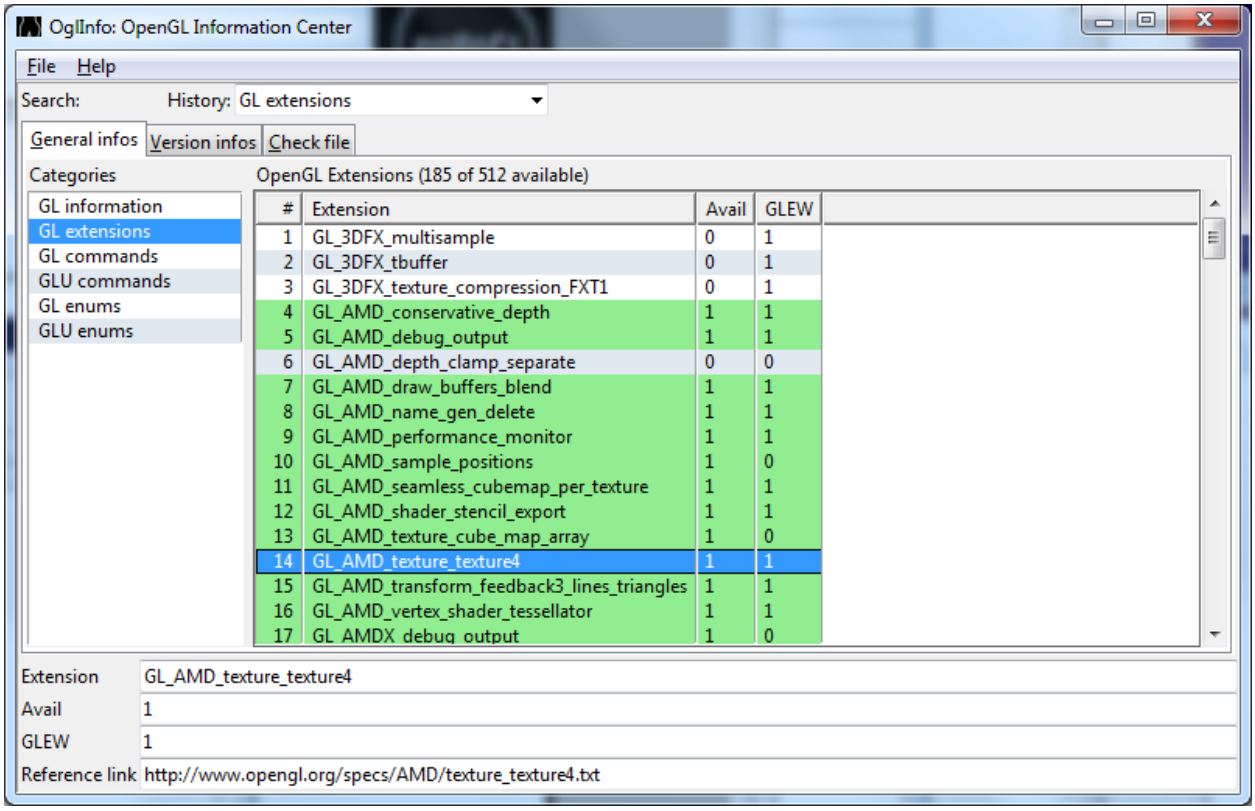


Figure 5: OglInfo category: GL extensions

Information about OpenGL core and extension functions

The *GL commands* category shows the following information about each OpenGL function:

- Availability on underlying hardware and driver.
- Corresponding OpenGL version or extension.
- Deprecated since OpenGL version (0.0 means not deprecated).
- Signature of function (both in C and Tcl).
- URL of official documentation page.

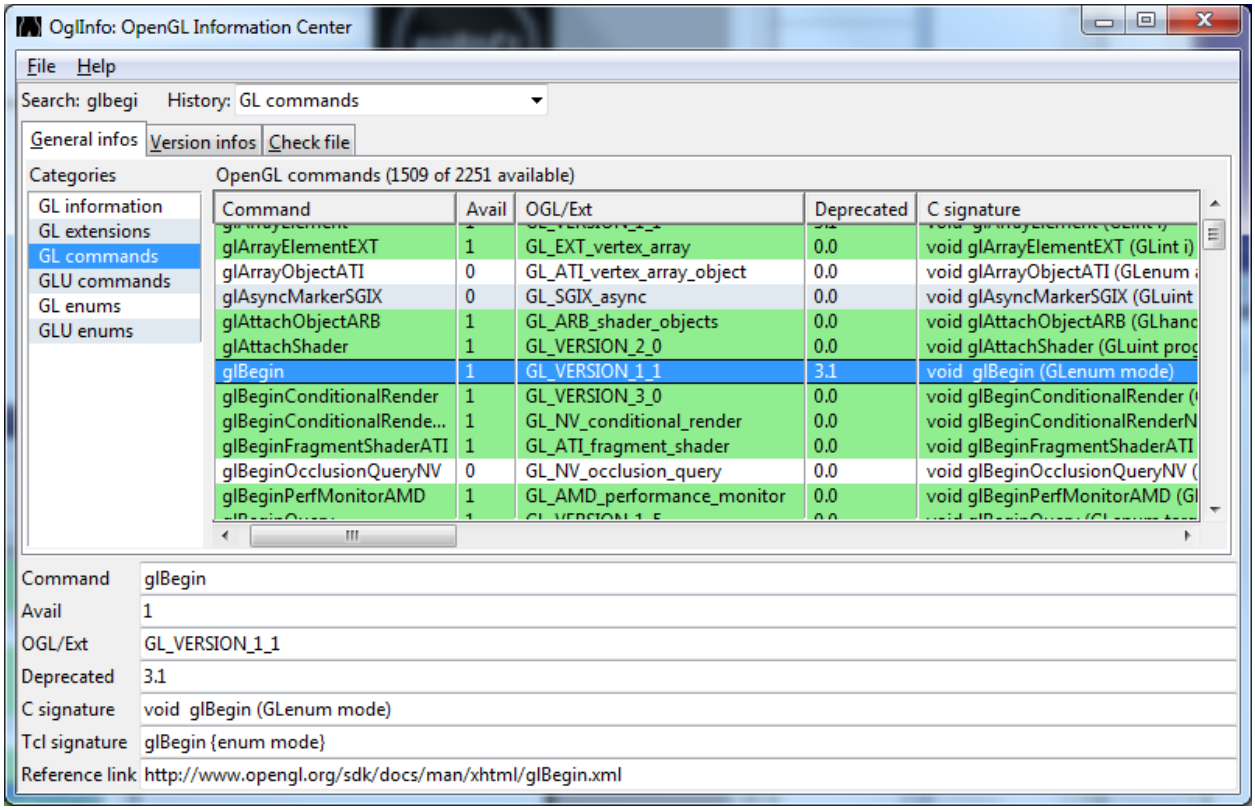


Figure 6: OglInfo category: GL commands

Information about OpenGL core and extension enumerations

The *GL enums* category shows the following information about each OpenGL enumeration:

- Decimal value.
- Hexadecimal value.
- Corresponding OpenGL version or extension.

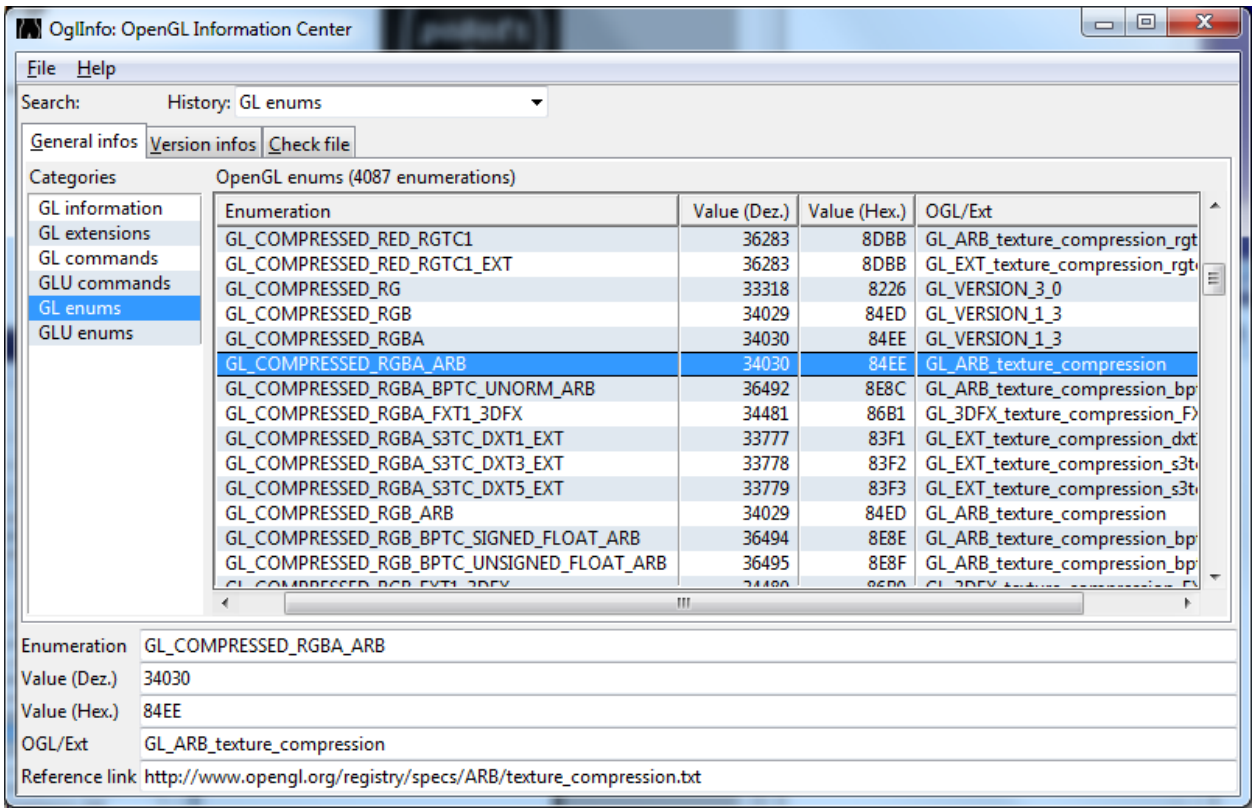


Figure 7: OglInfo category: GL enums

Information about OpenGL versions and extensions

The *Version infos* tab shows the following information about each OpenGL version or extension:

- List of corresponding functions.
- List of corresponding enumerations.

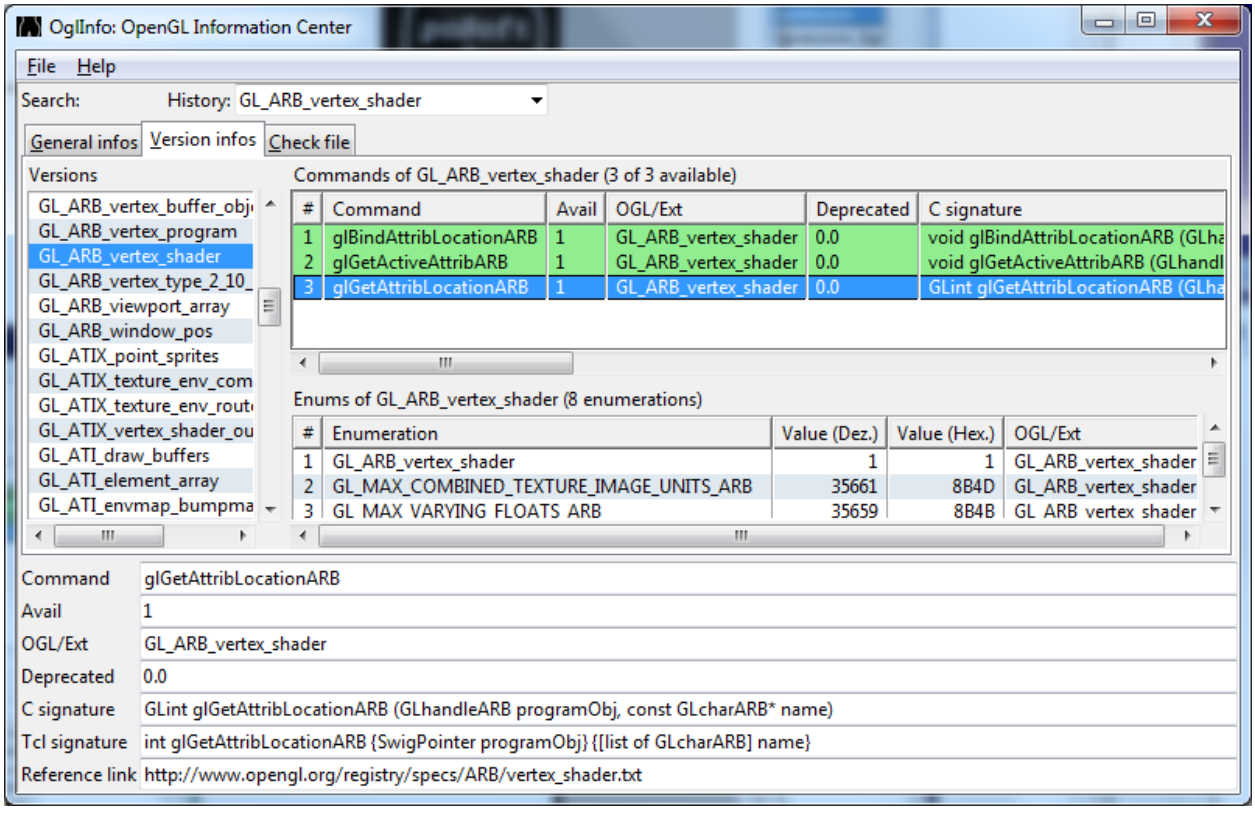


Figure 8: OglInfo category: Version infos

File check

The *Check File* tab allows to load OpenGL source files (C or Tcl) into a text widget.
The following information about the file is provided:

- List of used OpenGL functions.
- List of used OpenGL enumerations.

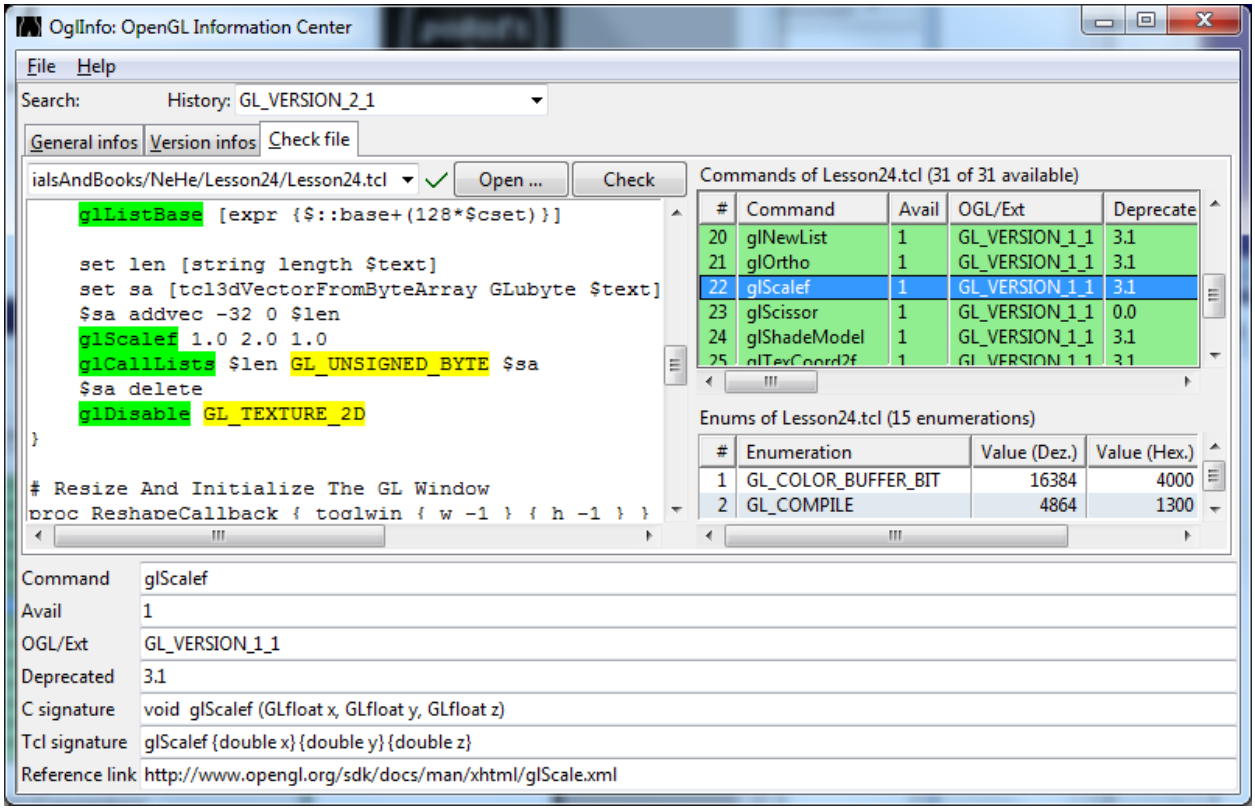


Figure 9: OglInfo category: Check file

History

All visited tables are held in a history for easy re-access.

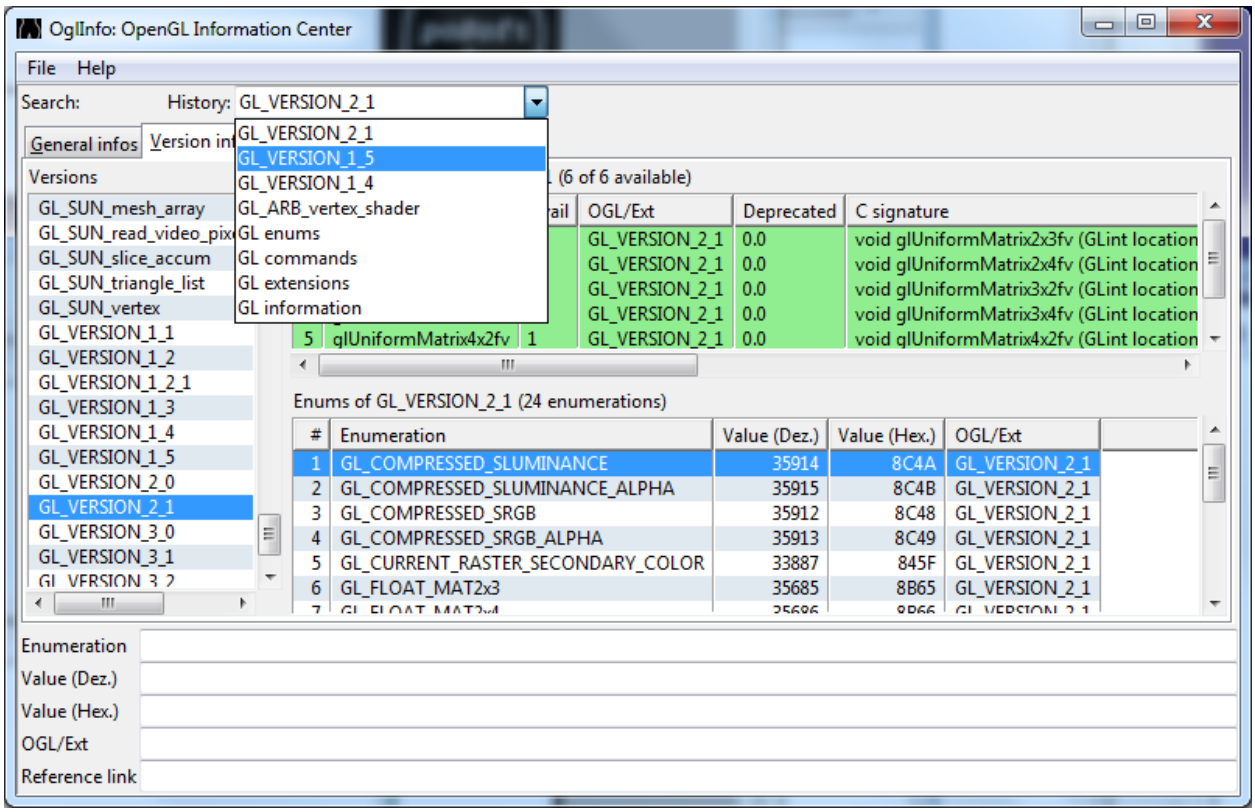


Figure 10: OglInfo history

Export functionality

All tables can be exported as CSV files for further inspection with a spreadsheet program.

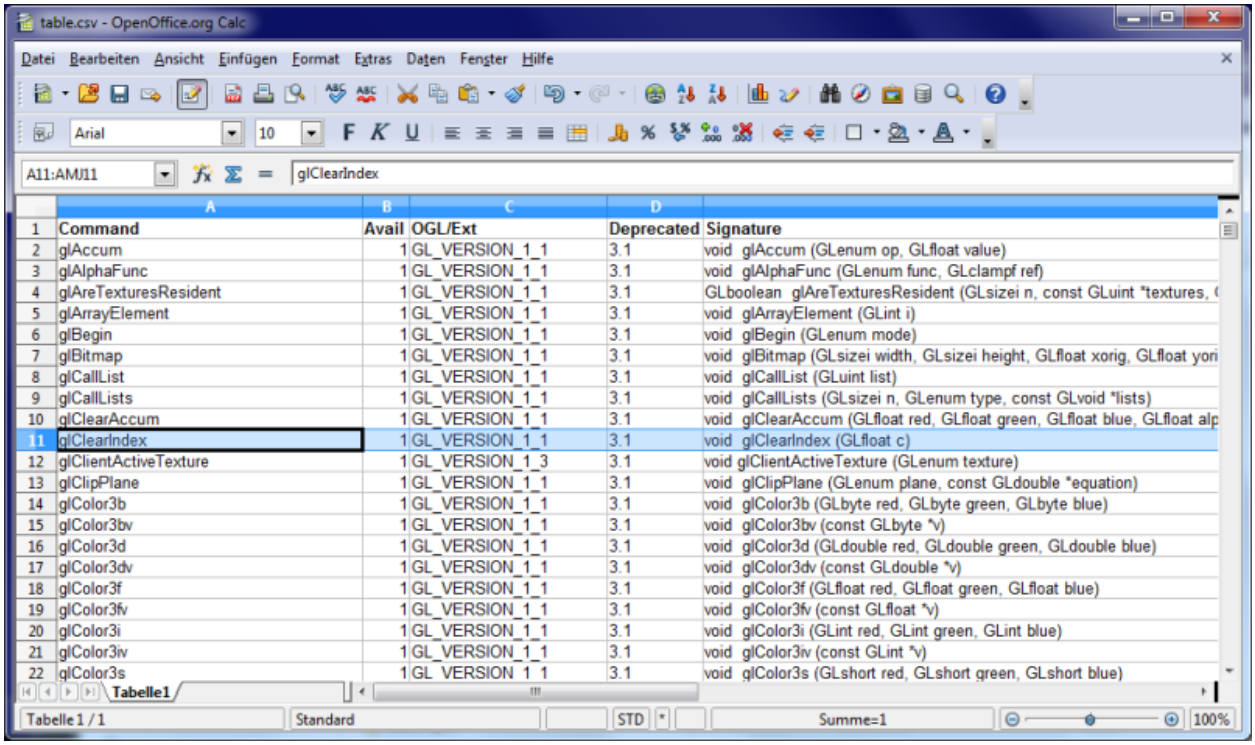


Figure 11: OglInfo export

6 Release Notes

The detailed release history can be found at https://www.tcl3d.org/history.html

The following table shows the Tcl3D releases in relation to official OpenGL releases.

OpenGL version	Release date	Tcl3D version	Wrapped OpenGL version	Wrapped file
OpenGL 1.0	01/1992			
OpenGL 1.1	01/1997			
OpenGL 1.2	03/1998			
OpenGL 1.2.1	10/1998			
OpenGL 1.3	08/2001			
OpenGL 1.4	07/2002			
OpenGL 1.5	07/2003			
OpenGL 2.0	09/2004			
	05/2005	Tcl3D 0.1.0	OpenGL 1.1	gl.h
	01/2006	Tcl3D 0.2.0	OpenGL 2.0	OglExt
OpenGL 2.1	07/2006			
OpenGL 3.0	07/2008			
	12/2008	Tcl3D 0.4.0	OpenGL 3.0	GLEW 1.5.1
OpenGL 3.1	03/2009			
OpenGL 3.2	08/2009			
	03/2010	Tcl3D 0.4.2	OpenGL 3.2	GLEW 1.5.3
OpenGL 3.3	03/2010			
OpenGL 4.0	03/2010			
	07/2010	Tcl3D 0.4.3	OpenGL 4.0	GLEW 1.5.4
OpenGL 4.1	07/2010			
	12/2010	Tcl3D 0.5.0	OpenGL 4.1	GLEW 1.5.7
OpenGL 4.2	08/2011			
OpenGL 4.3	08/2012			
OpenGL 4.4	07/2013			
OpenGL 4.5	08/2014			
OpenGL 4.6	07/2017			
	08/2018	Tcl3D 0.9.3	OpenGL 4.6	GLEW 2.1.0
	04/2024	Tcl3D 0.9.5	OpenGL 4.6	GLEW 2.2.0
	12/2024	Tcl3D 1.0.0	OpenGL 4.6	GLEW 2.2.0

Table 43: Tcl3D and OpenGL timeline

7 References and Abbreviations

Tcl3D specific references:

[1] Tcl3D homepage: https://www.tcl3d.org/

[2] Tcl3D page on the Tcler's Wiki: https://wiki.tcl-lang.org/page/Tcl3D

[3] Tcl3D discussion page: https://wiki.tcl-lang.org/page/Tcl3D+Discussion

[4] Tcl3D demo page: https://wiki.tcl-lang.org/page/Tcl3D+Demo+of+the+Month

[5] Tcl3D Reference Manual: https://www.tcl3d.org/docTcl3D.html

Libraries wrapped with Tcl3D:

[6] Togl: https://sourceforge.net/projects/togl/

[7] SDL: https://www.libsdl.org/

[8] FTGL: https://sourceforge.net/projects/ftgl/

[9] Freetype: https://freetype.org/

[10] GL2PS: https://geuz.org/gl2ps/

[11] OSG: https://www.openscenegraph.org/

[12] GLEW: https://github.com/nigels-com/glew

Demos used in Tcl3D:

[13] NeHe tutorials: https://nehe.gamedev.net/

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

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

[16] Nate Robins OpenGL tutorials: https://user.xmission.com/~nate/tutors.html

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

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

[19] Nopper's OpenGL core profile demos: https://github.com/McNopper/OpenGL

Tools needed for Tcl3D development:

[20] CMake Software Build System: https://cmake.org/

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

[22] Tcl (Batteries included distribution): https://www.tcl3d.org/bawt/

[23] Starpack Wiki page: https://wiki.tcl-lang.org/page/Starpack

Documentation:

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

[25] OpenGL Wiki page: https://wiki.tcl-lang.org/page/OpenGL

[26] OpenGL Registry: https://registry.khronos.org/OpenGL/index_gl.php

[27] OpenGL 4.6 API Reference Guide: https://www.khronos.org/files/opengl46-quick-reference-card.pdf

Abbreviation	Meaning
API	Application Program Interface
BAWT	Build Automation With Tcl
BSD	Berkeley Software Distribution
GUI	Graphical User Interface
PDF	Portable Document Format
RGB	Red-Green-Blue
Tcl	Tool Command Language
Tk	Toolkit
URL	Uniform Resource Locator