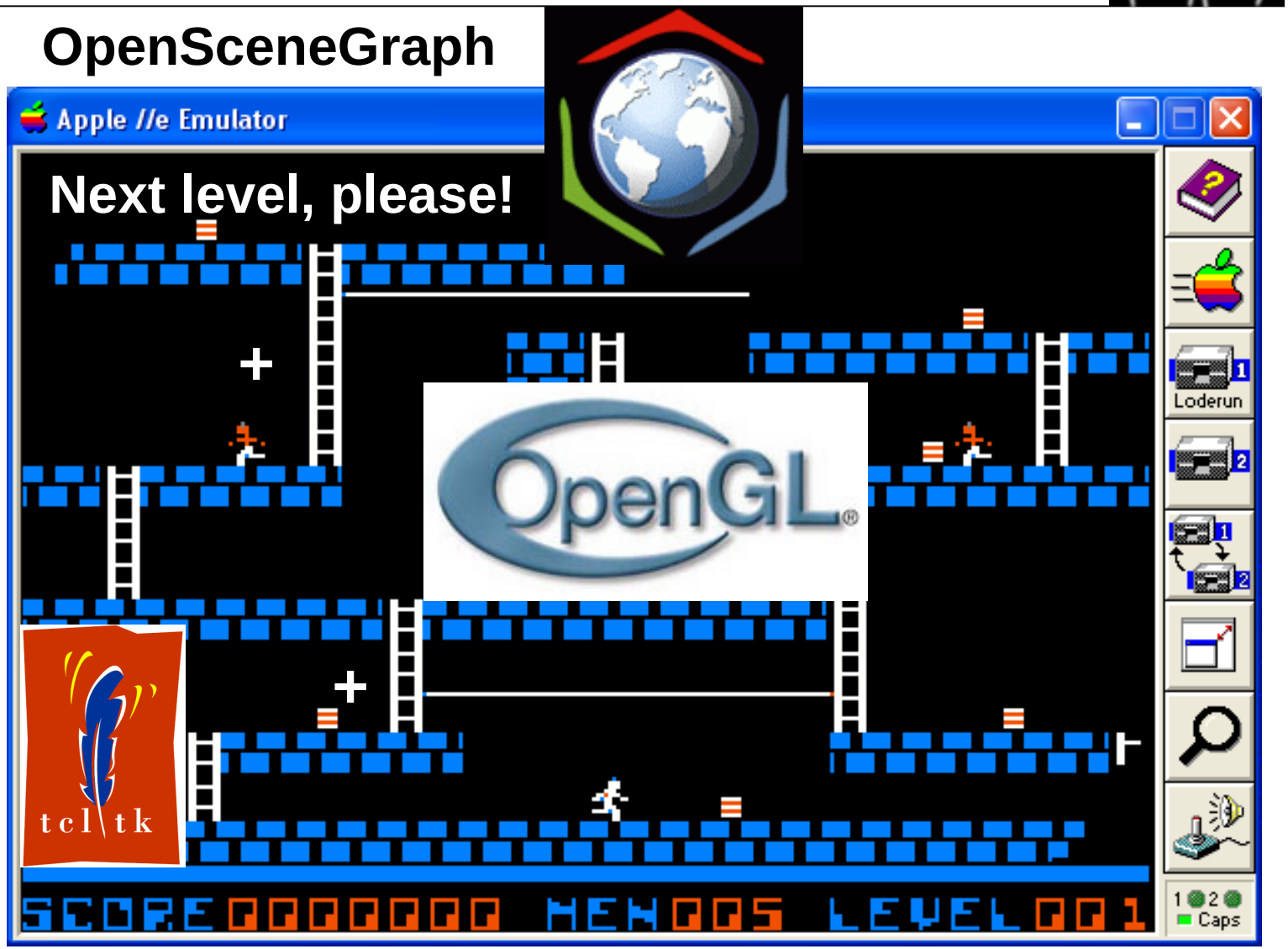
Tcl3D: Doing 3D with Tcl

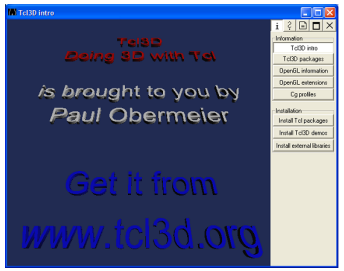
poSoft

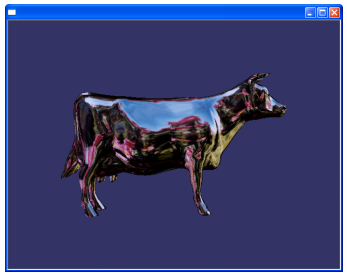
OpenSceneGraph

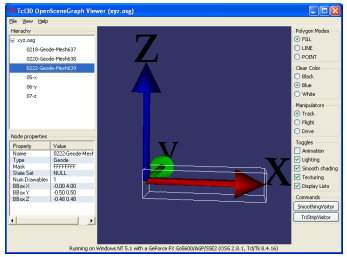
Tcl3D =



Presentation Overview





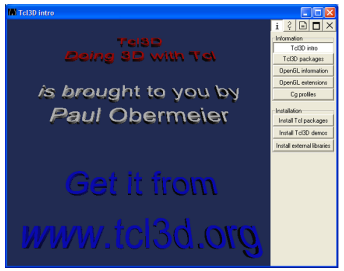


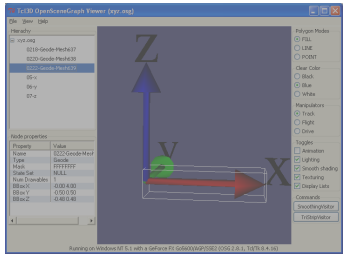
1. Tcl3D History

2. OpenSceneGraph Quick Tour

3. Tcl3D Module tcl3dOsg
   - Challenges
   - Wrapping Details
   - Examples
   - What's Next?

Tcl3D History





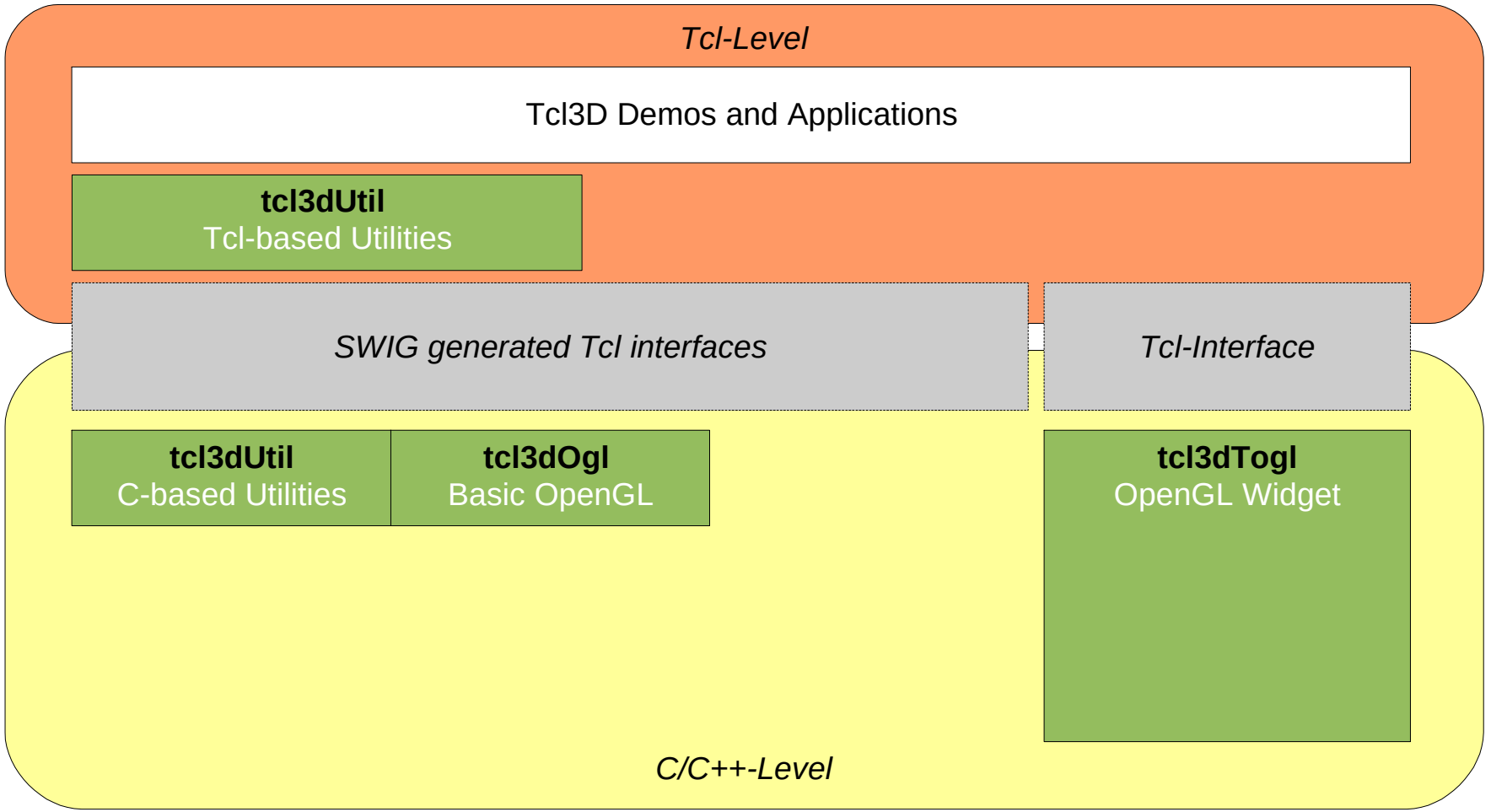
1. Tcl3D History

2. OpenSceneGraph Quick Tour

3. Tcl3D Module tcl3dOsg
   - Challenges
   - Wrapping Details
   - Examples
   - What's Next?

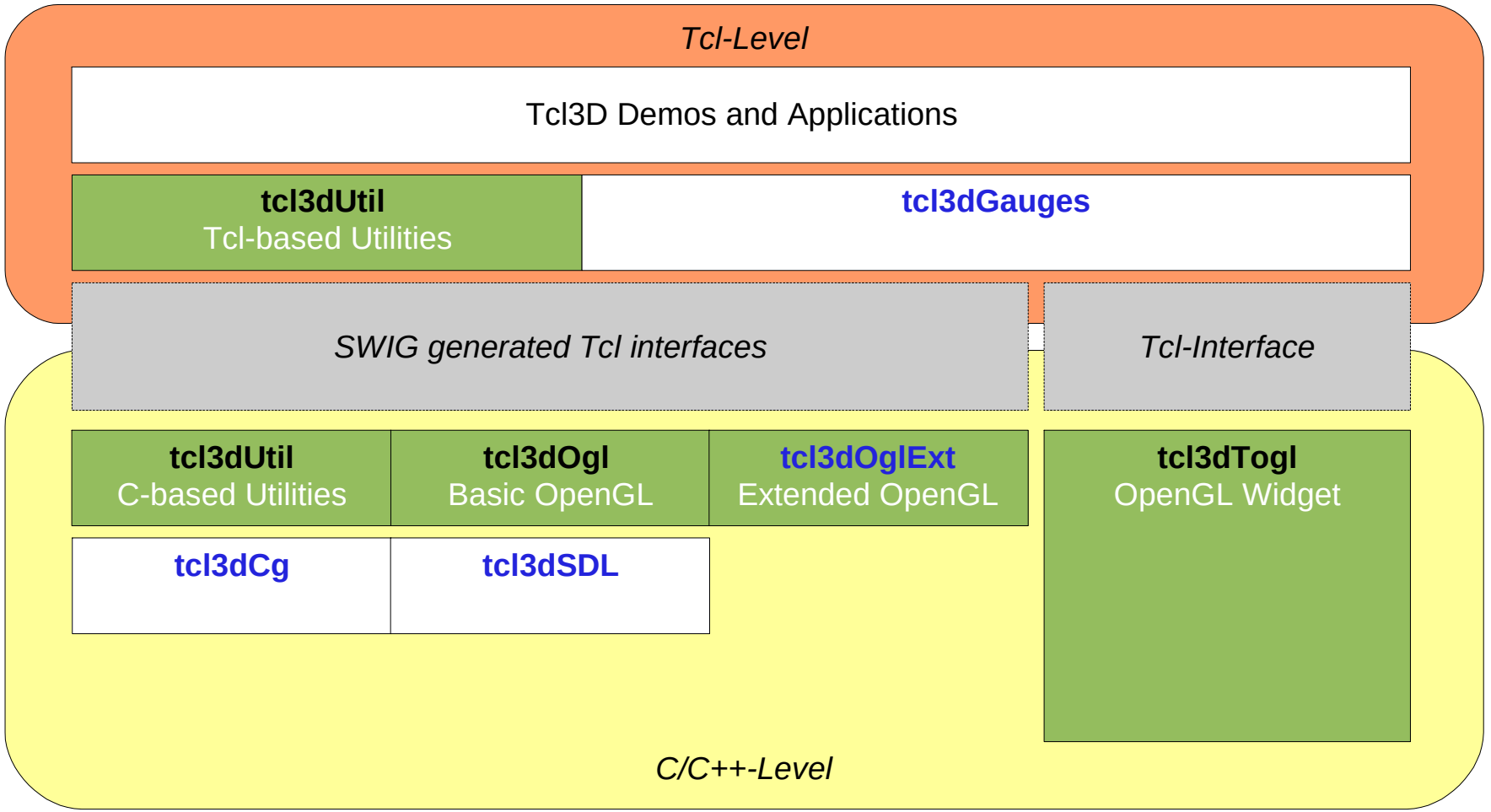
Tcl3D History: Version 0.1

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



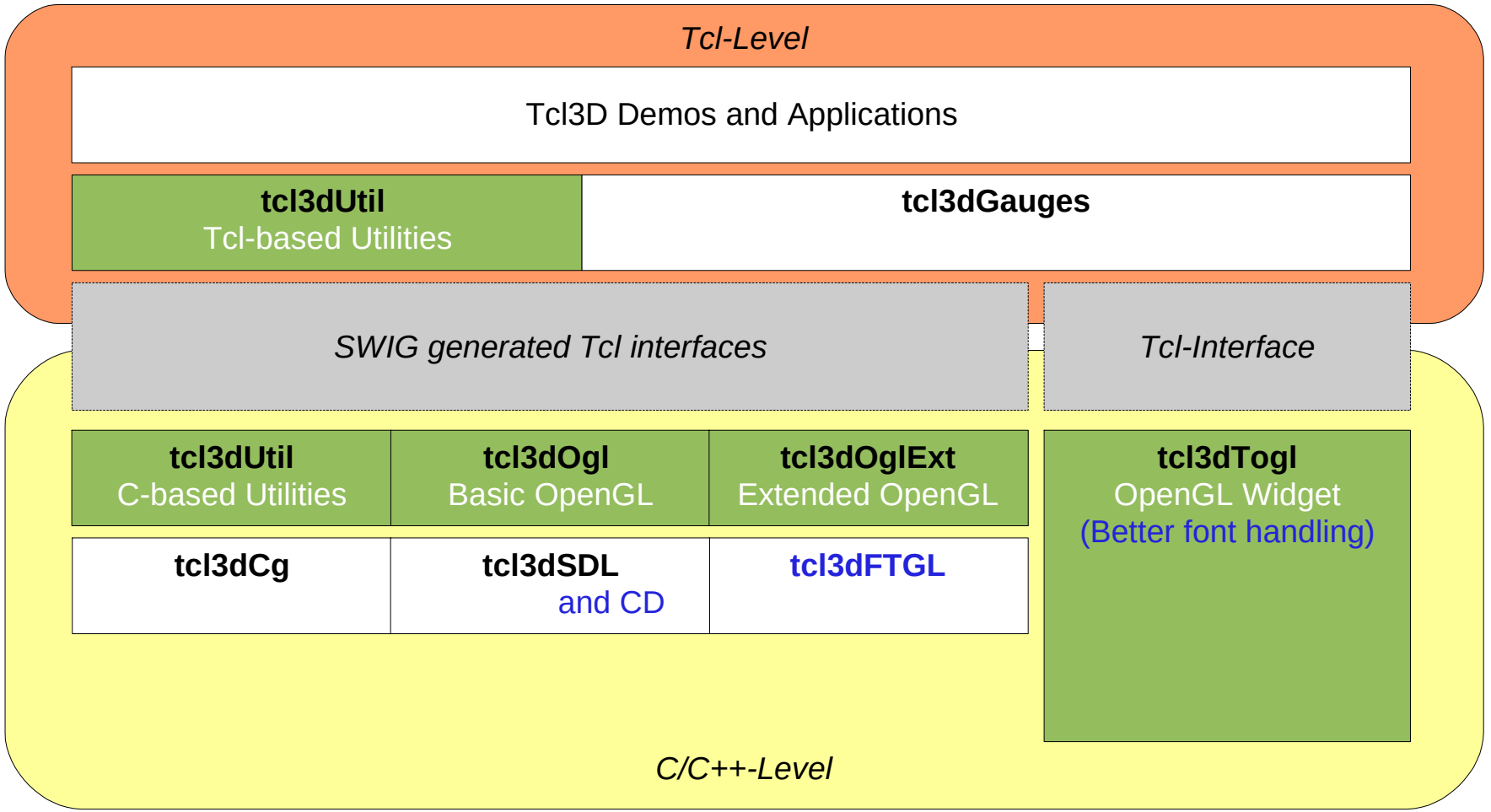
Tcl3D History: Version 0.2

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



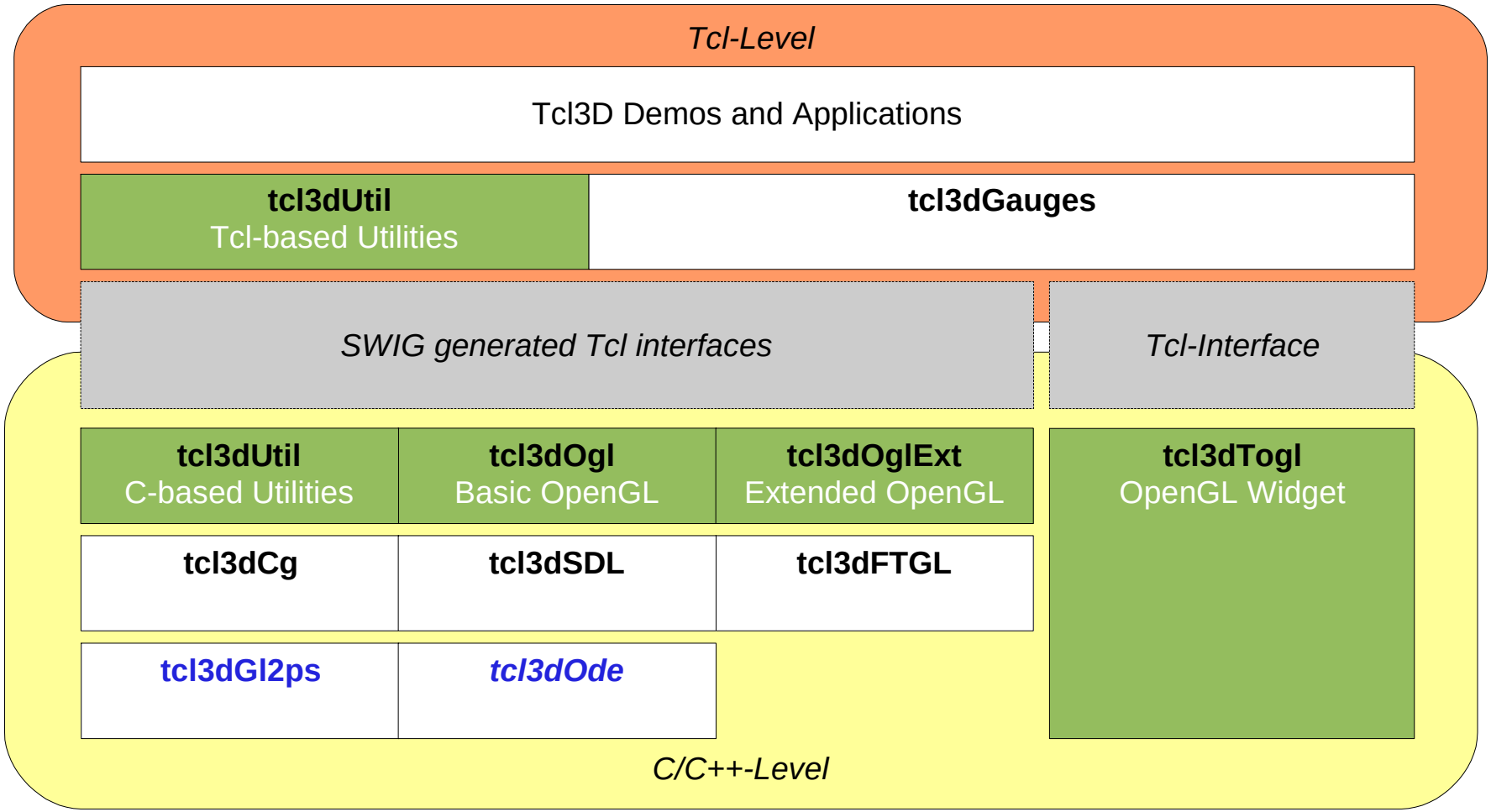
Tcl3D History: Version 0.3.0

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



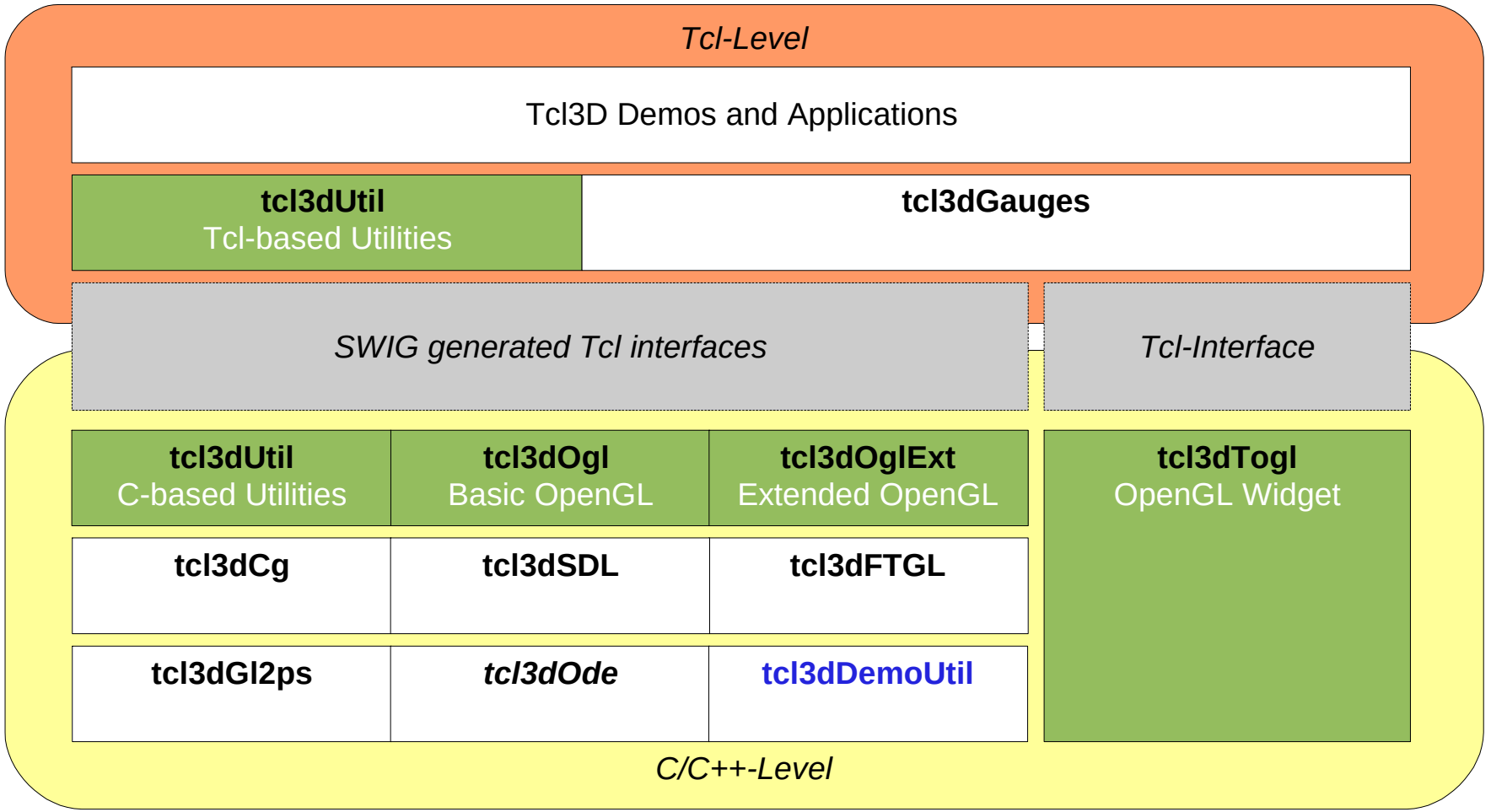
Tcl3D History: Version 0.3.1

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



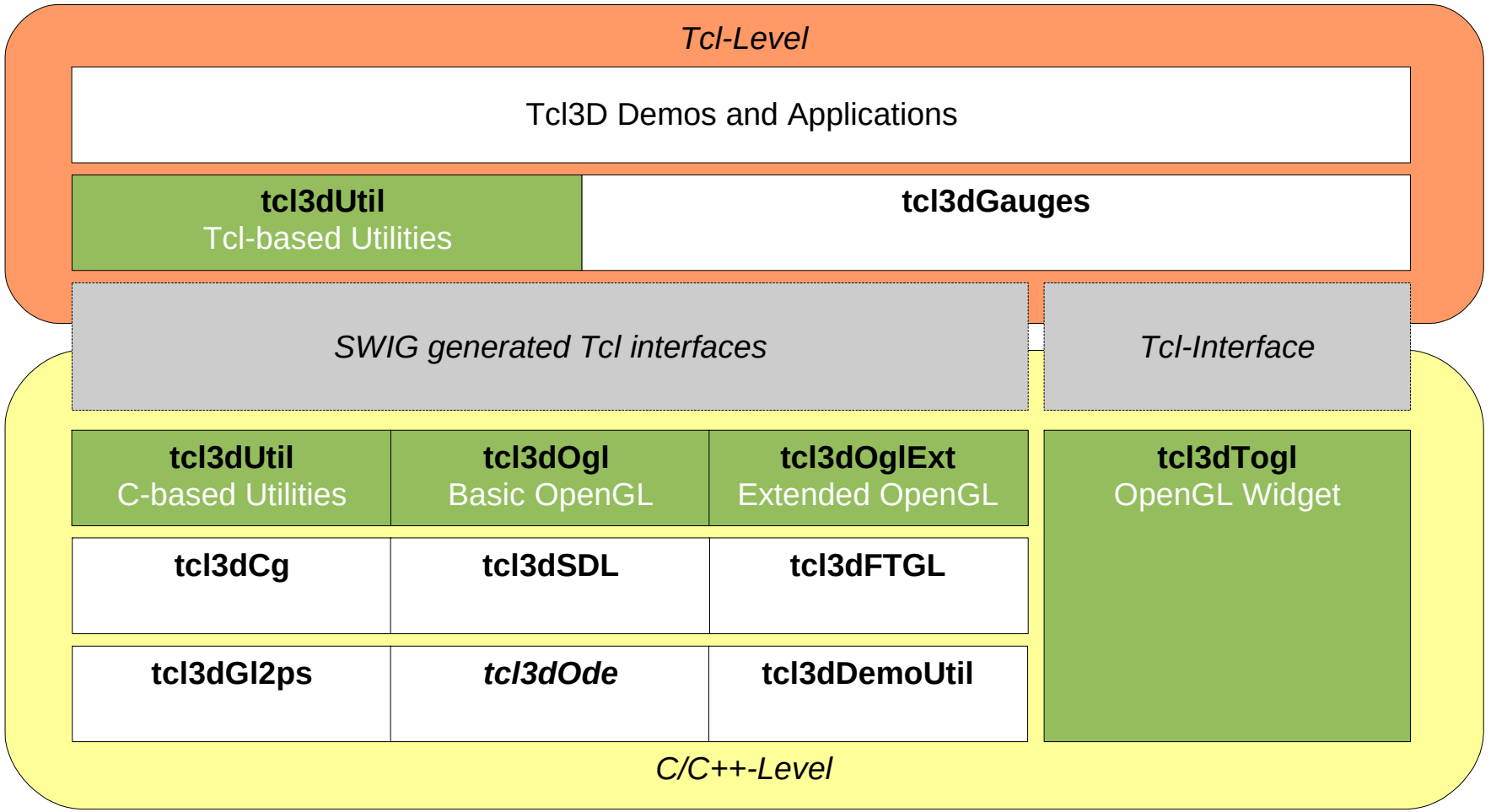
Tcl3D History: Version 0.3.2

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



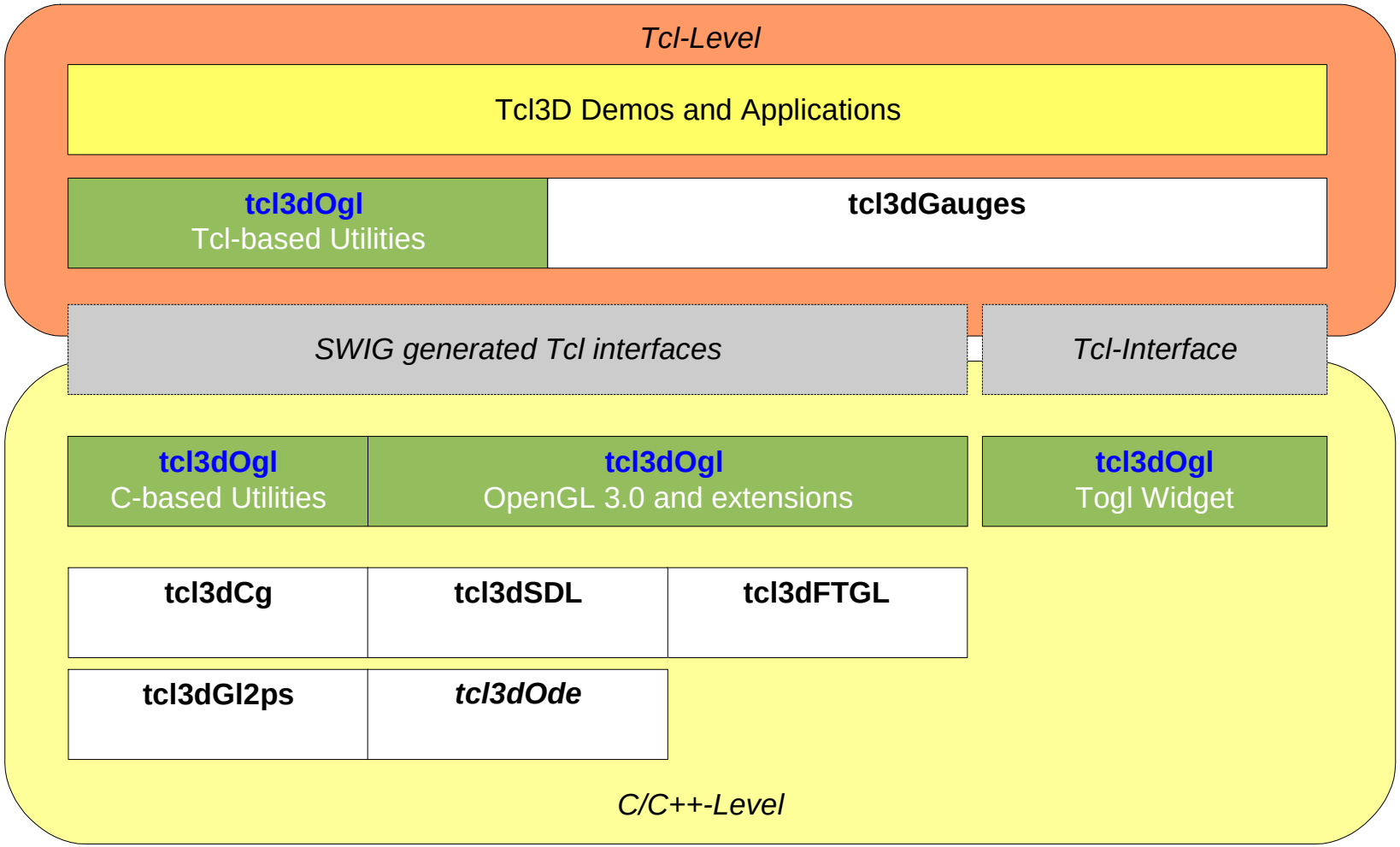
Tcl3D History: Version 0.3.3

Released 2008/09/14: Bug fixes, minor enhancements and several new demos.



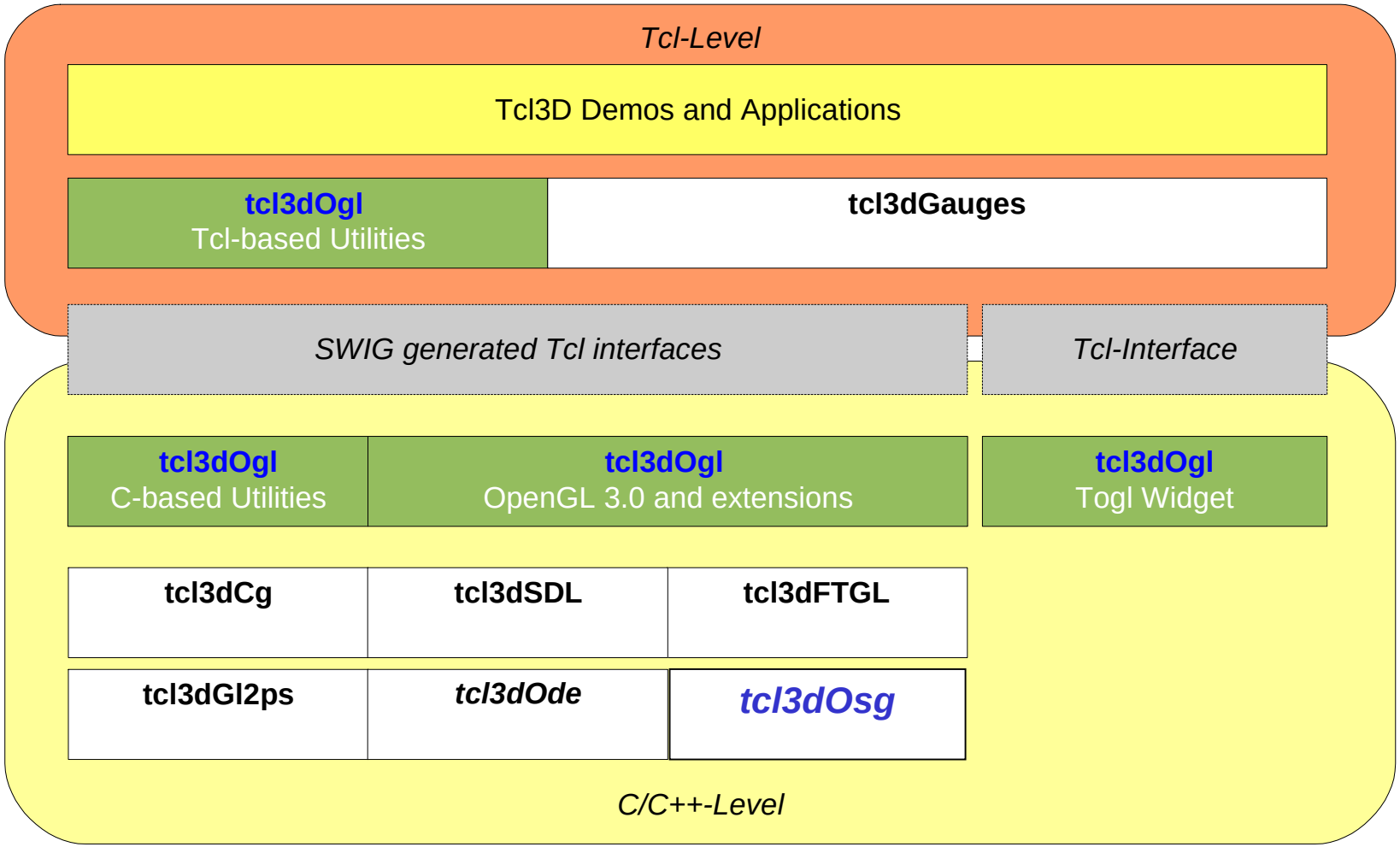
Tcl3D History: Version 0.4.0

Released 2008/12/30: OpenGL wrapping based on GLEW 1.5.1. Support of OpenGL 3.0. Reorganization of Tcl3D core module.

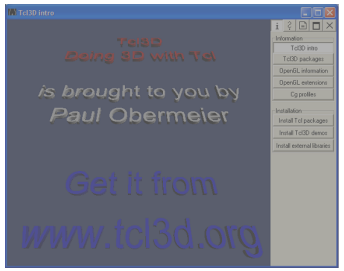


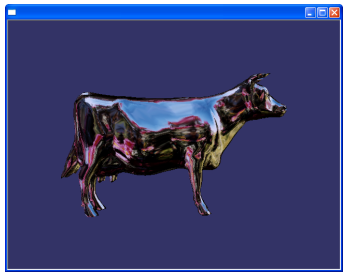
Tcl3D Future: Version 0.4.X or 0.5

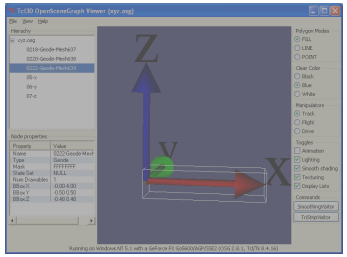
To be released in the near future: OpenSceneGraph module based on OSG version 2.8.1



OpenSceneGraph Quick Tour







1. Tcl3D History

2. OpenSceneGraph Quick Tour

3. Tcl3D Module tcl3dOsg
   - Challenges
   - Wrapping Details
   - Examples
   - What's Next?

OSG Quick Tour: Introduction

The OpenSceneGraph is an OpenSource, cross-platform graphics toolkit for the development of high-performance graphics applications such as flight simulators, games, virtual reality and scientific visualization.

It is based around the concept of a SceneGraph, providing an object-oriented framework on top of OpenGL. This frees the developer from implementing and optimizing low-level graphics calls and provides many additional utilities for rapid development of graphics applications.

Written entirely in Standard C++ and OpenGL it runs on all Windows platforms, OSX, GNU/Linux, IRIX, Solaris, HP-Ux, AIX and FreeBSD operating systems.

It makes full use of the STL and DesignPatterns.

Official homepage: http.//www.openscenegraph.org
Quick Start Guide: http://www.skew-matrix.com/OSGQSG/index.html

Most explanations in the following Quick Tour are taken from the Quick Start Guide written by Paul Martz.

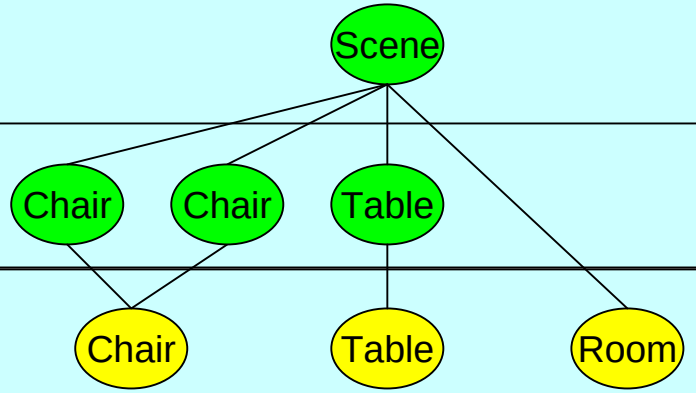
OSG Quick Tour: SceneGraph Structure

A scene graph is a hierarchical data structure that organizes spatial data for efficient rendering.

Top-level root node.

Group nodes organize geometry and the rendering state that controls their appearance.

Leaf nodes (Geodes) contain the actual geometry that make up the objects in the scene.



Advantage:
No triangle soup anymore, but structured 3D-objects.

OSG Quick Tour: Node Types

Examples of scene graph node types

Node

The Node class is the base class for all nodes in the scene graph. It contains methods to facilitate scene graph traversals, culling, application callbacks, and state management.

Group

The Group class is the base class for any node that can have children. It is a key class in the spatial organization of scene graphs.

Geode

The Geode (or *Geometry Node*) class corresponds to the leaf node in OSG. It has no children, but contains osg::Drawable objects that contain geometry for rendering.

LOD

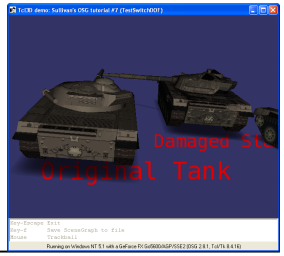
The LOD class displays its children based on their distance to the view point. This is commonly used to create varying levels of detail for objects in a scene.

MatrixTransform

The MatrixTransform class contains a matrix that transforms the geometry of its children, allowing scene objects to be rotated, translated, scaled, skewed, projected.

Switch

The Switch class contains a Boolean mask to enable or disable processing of its children.



OpenSceneGraph Architecture

OpenSceneGraph consists of 3 major components:

- Core OSG
- NodeKits
- Plugins

The **Core OSG** libraries provide essential scene graph and rendering functionality, as well as additional functionality that 3D graphics applications typically require.

NodeKits extend the functionality of core OSG scene graph node classes to provide higher-level node types and special effects.

OSG plugins are libraries that read and write 2D image and 3D model files.

Component: Core OSG

osg

This library contains the scene graph node classes that your application uses to build scene graphs. It also contains classes for vector and matrix math, geometry, and rendering state specification and management.

Other classes in osg provide additional functionality typically required by 3D applications, such as argument parsing, animation path management, and error and warning communication.

osgUtil

This utility library contains classes and functions for operating on a scene graph and its contents, gathering statistics and optimizing a scene graph, and creating the render graph.

There are also classes for geometric operations, such as Delaunay triangulation, triangle stripification, and texture coordinate generation.

osgDB

This library contains classes and functions for creating and rendering 3D databases. It contains a registry of OSG plugins for 2D and 3D file I/O, as well as a class for accessing those plugins. The osgDB database pager supports dynamic loading and unloading of large database segments.

osgViewer

This library contains classes that manage views into the scene. osgViewer integrates OSG with a wide variety of windowing systems.

Component: Nodekits

osgFX

This NodeKit provides additional scene graph nodes for rendering special effects, such as anisotropic lighting, bump mapping, and cartoon shading.

osgManipulator

This NodeKit contains several classes for manipulating selected objects in the scene graph.

osgParticle

This NodeKit provides particle-based rendering effects, such as explosions, fire, and smoke.

osgSim

This NodeKit supports the special rendering requirements of simulation systems and OpenFlight databases, such as terrain elevation query classes, light point nodes, and DOF transformation nodes.

osgText

This NodeKit is a powerful tool for adding text to your scene. It fully supports all TrueType fonts.

osgTerrain

This NodeKit provides support for rendering height field data.

osgShadow

This NodeKit provides a framework for supporting shadow rendering techniques.

Component: Plugins

Image plugins

Plugins for several 2D image file formats: .bmp, .dds, .gif, .jpeg, .pic, .png, .rgb, .tga, .tiff.

QuickTime plugin for loading movie files.

Plugin for loading font files using the FreeType library.

3D model plugins

Plugins for several 3D model file formats:

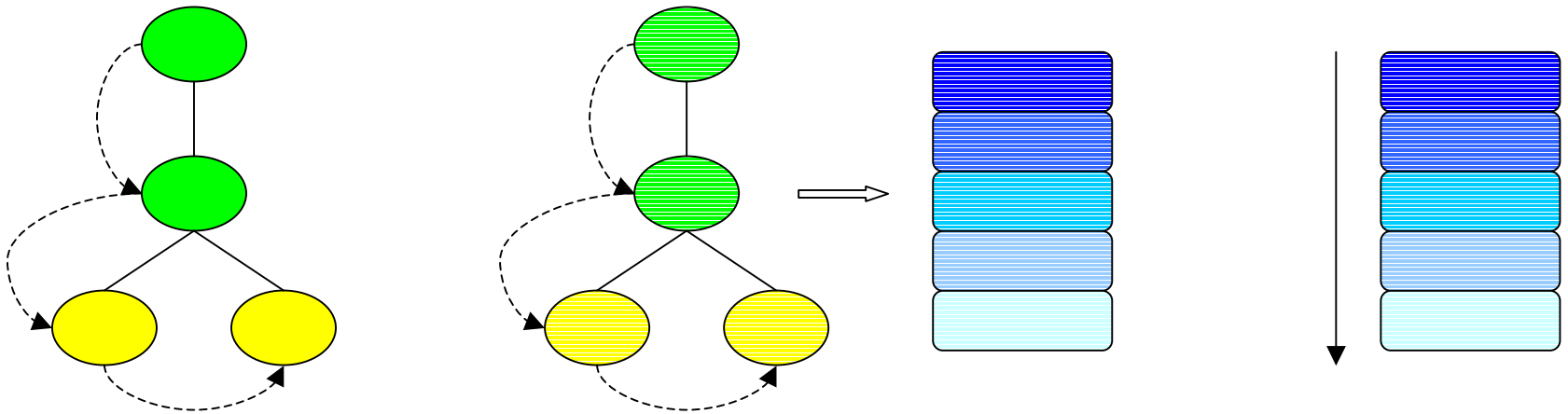
- 3D Studio Max (.3ds)
- Alias Wavefront (.obj)
- Carbon Graphics' Geo (.geo)
- COLLADA (.dae)
- ESRI Shapefile (.shp)
- OpenFlight (.flt)
- Quake (.md2)
- Terrex TerraPage (.txp)

OpenSceneGraph Update Traversal

The update traversal (sometimes referred to as the application traversal) allows the application to modify the scene graph, which enables dynamic scenes.

Updates are accomplished either directly by the application or with callback functions assigned to nodes within the scene graph.

Applications use the update traversal to modify the position of a flying aircraft in a flight simulation, for example, or to allow user interaction using input devices.



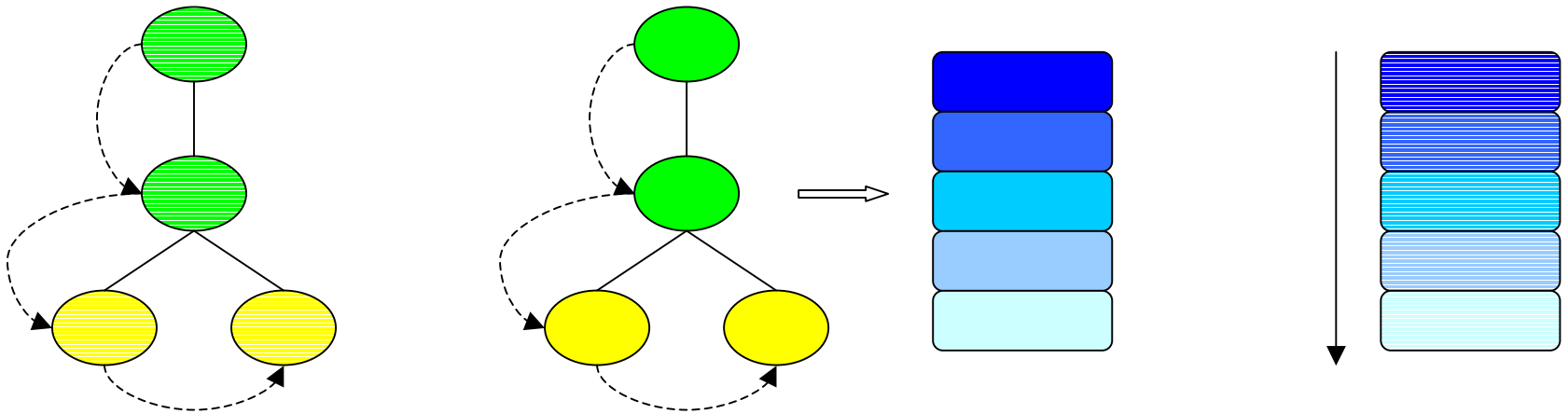
OSG Quick Tour: Render Stages

OpenSceneGraph Cull Traversal

During the cull traversal, the scene graph library tests the bounding volumes of all nodes for inclusion in the scene.

If a leaf node is within the view, the scene graph library adds leaf node geometry references to a final rendering list.

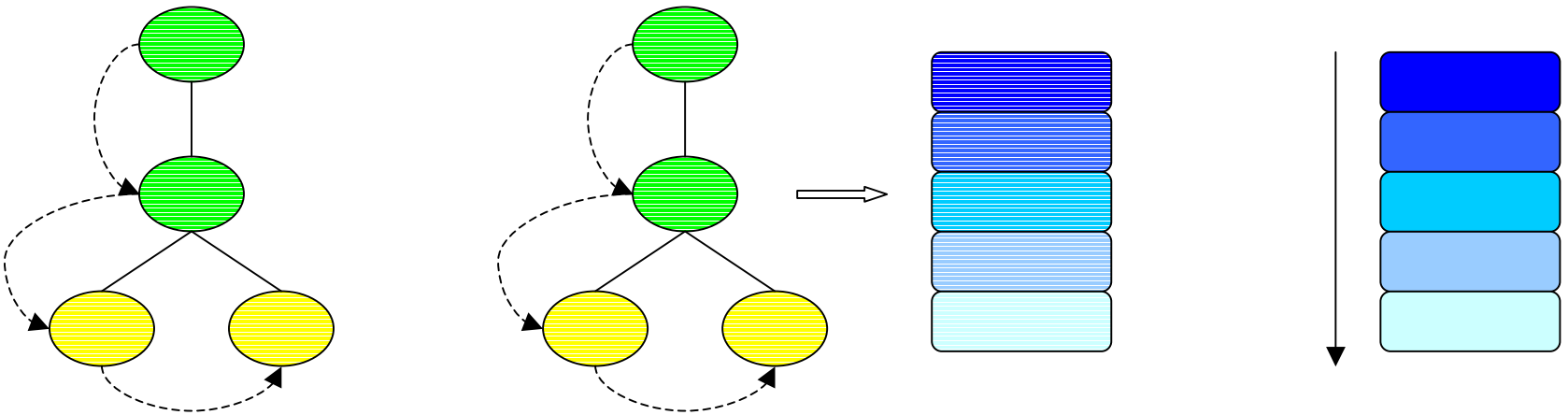
This list is sorted by opaque versus translucent, and translucent geometry is further sorted by depth.



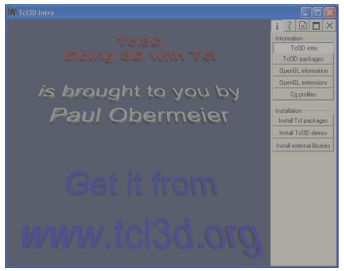
OSG Quick Tour: Render Stages

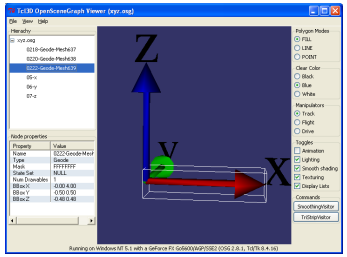
OpenSceneGraph Draw Traversal

In the draw traversal (sometimes referred to as the render traversal), the scene graph traverses the list of geometry created during the cull traversal and issues low-level graphics API calls to render that geometry.



Tcl3D Module tcl3dOsg





1. Tcl3D History

2. OpenSceneGraph Quick Tour

3. Tcl3D Module tcl3dOsg
   - Challenges
   - Wrapping Details
   - Examples
   - What's Next?

Tcl3D Module tcl3dOsg: Challenges

Complexity comparison of existing Tcl3D modules and new OSG wrapper.

Module name	Number of header files	LOC of wrapper file(s)
tcl3dOgl	< 10	130,000
tcl3dCg	< 10	23,000
tcl3dOsg	> 400	520,000

OpenSceneGraph makes heavy use of template and nested classes.

OpenSceneGraph makes heavy use of callback classes.

Tcl3D Module tcl3dOsg: Wrapping

Rename standard C++ operators and extend classes for Tcl style method names.

```
%define %OPERATOR_RENAME(NAME)
%extend NAME {
    %rename(eq)   operator ==;
    %rename(ne)   operator !=;
    %rename(less) operator <;

    %rename(mul) operator *;
    %rename(div) operator /;
    %rename(add) operator +;
    %rename(sub) operator -;

    %rename(mulSelf) operator *=;
    %rename(divSelf) operator /=;
    %rename(addSelf) operator +=;
    %rename(subSelf) operator -=;
}
%enddef
```

```
%define %VEC_EXTEND(NAME)
%extend osg::##NAME {

    // Assign a vector to another vector.
    NAME copy () const {
        return *self;
    }

    // Return element i of the vector.
    value_type get (int i) const {
        return (*self)[i];
    }

    %rename(cross) operator ^;
};
%enddef
```

Tcl3D Module tcl3dOsg: Viewer

Two possibilities for displaying 3D content: osg::Viewer or Togl widget.

```
# Create the viewer and set its scene data.
osgViewer::ViewerRef viewer [osgViewer::Viewer]
viewer setSceneData [CreateScene]
viewer setCameraManipulator [osgGA::TrackballManipulator]

# Use the standard OSG viewer window without any Tk widgets.
viewer setUpViewInWindow 50 50 500 400
viewer run
```


```
# Create the viewer and set its scene data.
osgViewer::ViewerRef viewer [osgViewer::Viewer]
viewer setSceneData [CreateScene]
viewer setCameraManipulator [osgGA::TrackballManipulator]

# Use the OSG viewer inside a Togl widget.
set osgwin [viewer setUpViewerAsEmbeddedInWindow 50 50 500 400]
viewer realize

# Propagate key and mouse events to embedded OSG window.
bind . <KeyPress>  "tcl3dOsgKeyPress $toglwin $osgwin %N"
bind $toglwin <B1> "tcl3dOsgButtonPress $toglwin $osgwin %x %y 1"
```

Tcl3D Module tcl3dOsg: NodeVisitor

osg::NodeVisitor is OSG's implementation of the Visitor design pattern [Gamma95].
In essence, **NodeVisitor** traverses a scene graph and calls a function for each visited node.
Specialized node visitors exist for miscellaneous puroposes.

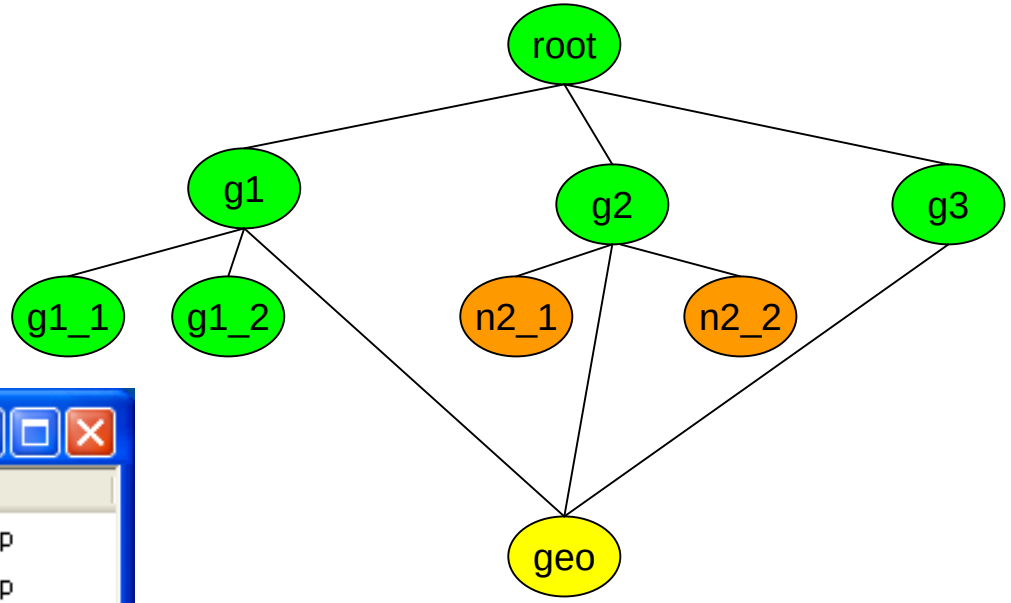
tcl3dOsgNodeVisitor is a specialized class inherited from base class osg::NodeVisitor to enable Tcl procedures to act as callback functions.

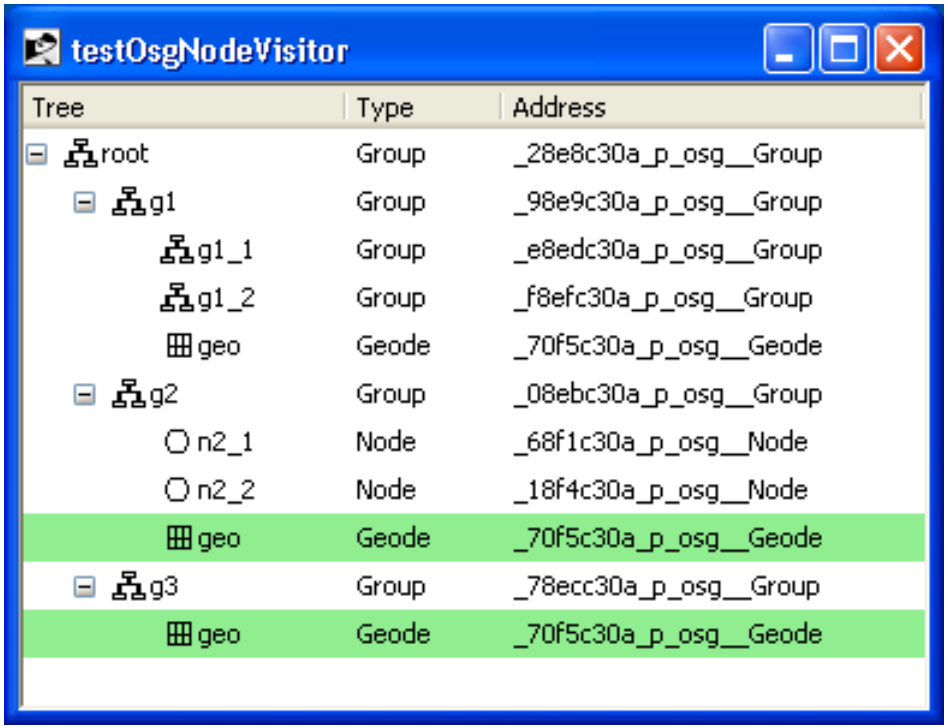
```
class tcl3dOsgNodeVisitor : public osg::NodeVisitor {
    public:

        tcl3dOsgNodeVisitor (Tcl_Interp *interp);
        tcl3dOsgNodeVisitor (Tcl_Interp *interp, TraversalMode tm);
        tcl3dOsgNodeVisitor (Tcl_Interp *interp, VisitorType type, TraversalMode tm);
        tcl3dOsgNodeVisitor (Tcl_Interp *interp, const std::string &tclProc,
                             const std::string &tclArgs="");
        virtual void apply (osg::Node &node);
        void setVisitorProc (const std::string &tclProc,
                             const std::string &tclArgs="");
        const std::string& getVisitorProc ();
        const std::string& getVisitorArgs ();
    ...
};
```

Tcl3D Module tcl3dOsg: NodeVisitor

tcl3dOsgNodeVisitor Example

Map the scene graph to a tree structure for display purposes.





Tcl3D Module tcl3dOsg: NodeVisitor

tcl3dOsgNodeVisitor Example

```
# Visitor callback procedure to display the scene graph hierarchy in a tree widget.
proc visit { node args } {
    set nodeName [osg::Object_getName $node]
    set nodeType [osg::Object_className $node]
    set n        [osg::Node_getNumParents $node]
    if { $n == 0 } {
        # This node does not have a parent. Thus it must be the root node.
    } elseif { $n == 1 } {
        # This node has exactly 1 parent. Thus we can insert it into the tree easily.
    } else {
        # We have a node with more than 1 parent, i.e. it is a graph and not a tree.
    }
}

# Declare a NodeVisitor and set its callback procedure to visit.
osg::tcl3dOsgNodeVisitor nv $::osg::NodeVisitor_TRAVERSE_ALL_CHILDREN
nv setVisitorProc visit

set root [osgDB::readNodeFile cow.osg]
$root accept nv
```

Tcl3D Module tcl3dOsg: NodeCallback

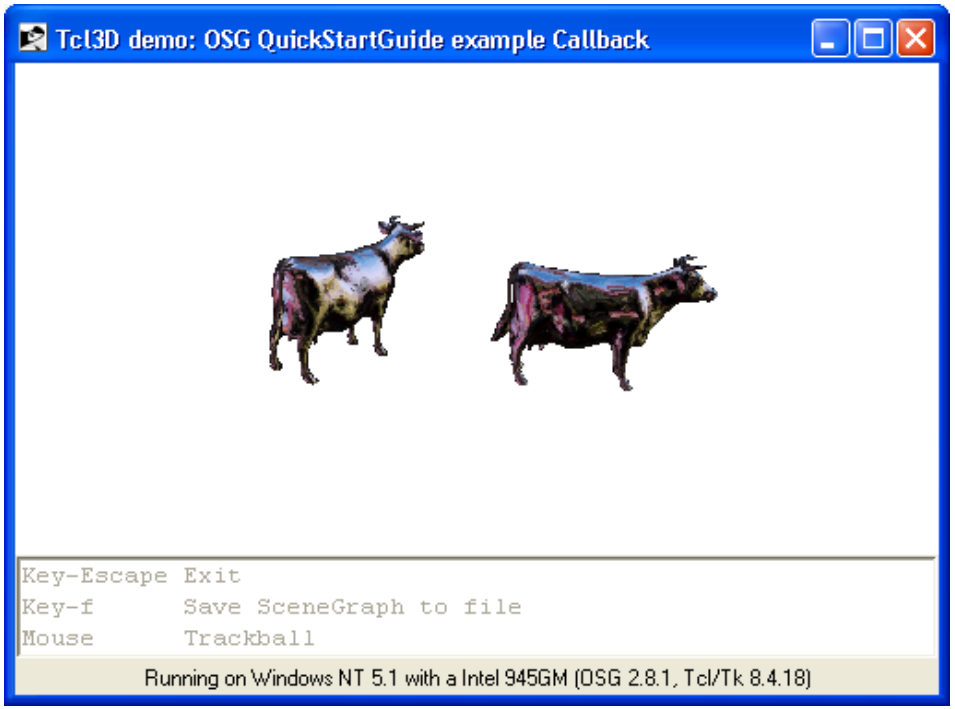
OSG's callback interface (**osg::NodeCallback**) is based on the Callback design pattern [Gamma95].
It allows you to assign callbacks to Node and Drawable objects.
OSG executes Node callbacks during the update and cull traversals, and executes Drawable callbacks during the cull and draw traversals.

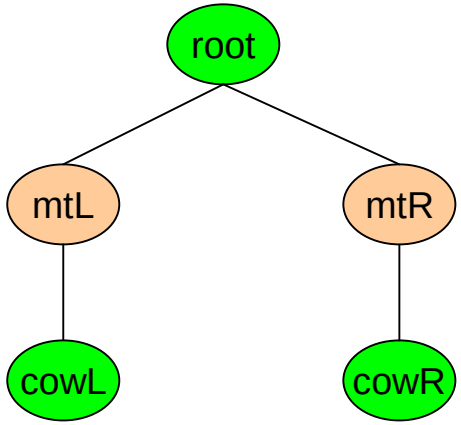
```
class tcl3dOsgNodeCallback : public osg::NodeCallback {
public:

    tcl3dOsgNodeCallback (Tcl_Interp *interp);
    tcl3dOsgNodeCallback (Tcl_Interp *interp, const std::string &tclProc,
                          const std::string &tclArgs="");
    virtual void operator() (osg::Node* node, osg::NodeVisitor* nv);
    void setCallbackProc (const std::string &tclProc, const std::string &tclArgs = "");
    const std::string& getCallbackProc ();
    const std::string& getCallbackArgs ();

private:
    std::string _tclProc;
    std::string _tclArgs;
    Tcl_Interp  *_interp;
};
```

Tcl3D Module tcl3dOsg: NodeCallback

tcl3dOsgNodeCallback Example

Auto-rotate parts of the scene graph by specifying a Nodecallback at a MatrixTransform node.





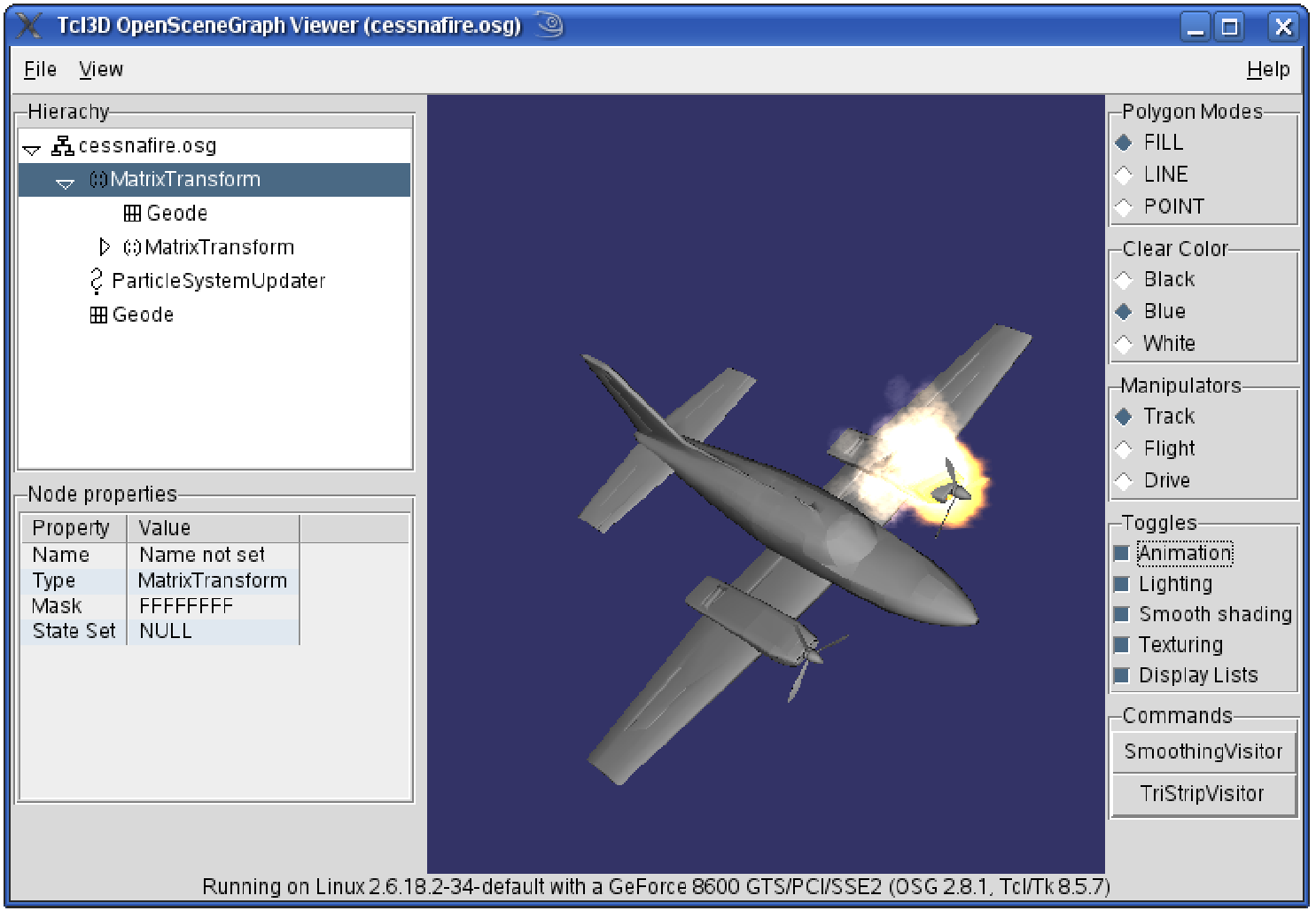
Tcl3D Module tcl3dOsg: NodeCallback

tcl3dOsgNodeCallback Example

```
# Use a NodeCallback to manipulate a MatrixTransform object's matrix.
proc RotateCB { node args } {
    global gAngle

    osg::Matrix mR
    osg::Matrix mT
    mT makeTranslate -6 0 0
    mR makeRotate $gAngle [osg::Vec3 rot 0 0 1]
    $node setMatrix [mR mul mT]

    # Increment the angle for the next frame.
    set gAngle [expr $gAngle + 0.01]
}

osg::MatrixTransform mtLeft

# Set the update callback.
osg::tcl3dOsgNodeCallback nc RotateCB
mtLeft setUpdateCallback nc
```

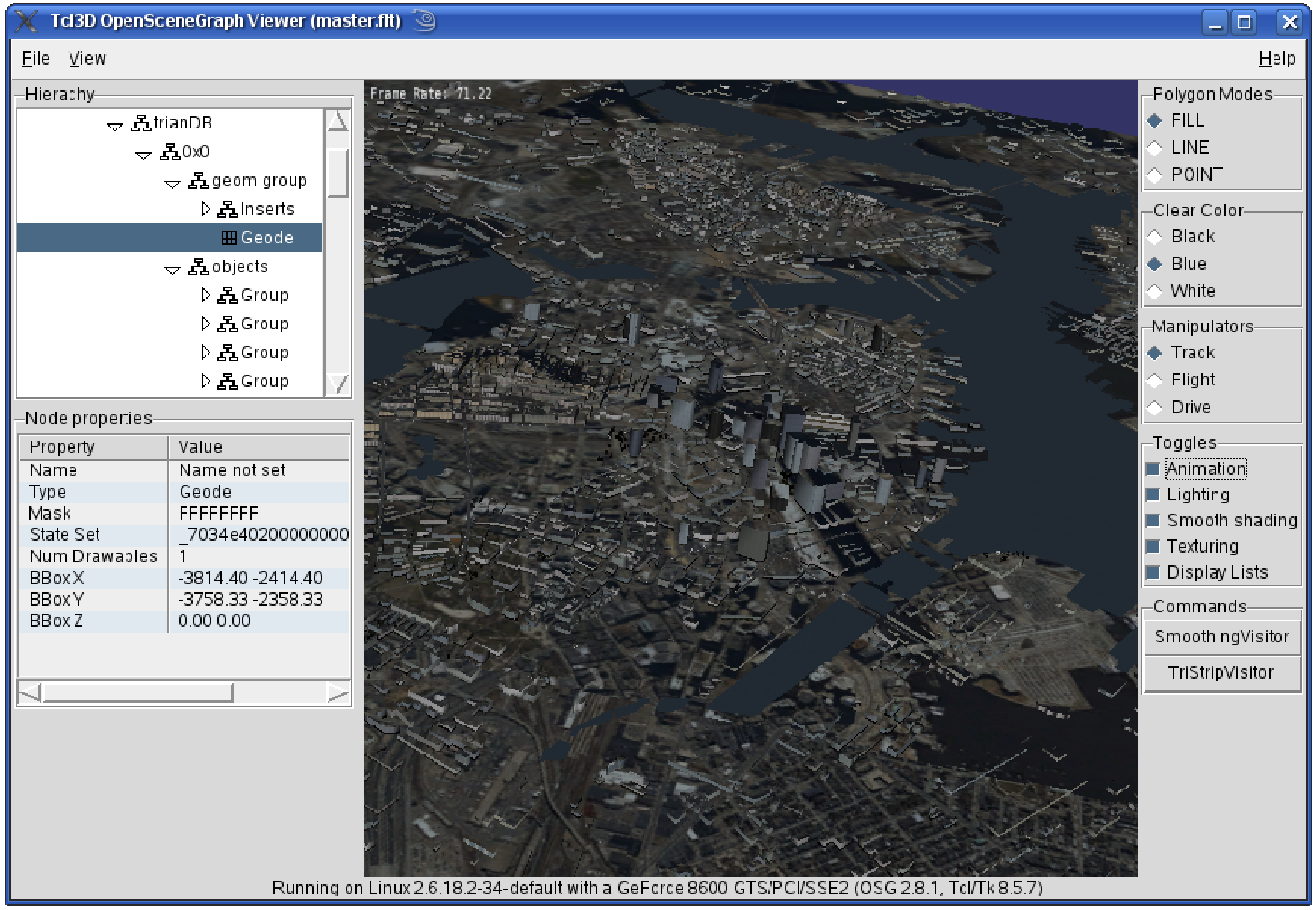
Tcl3D Module tcl3dOsg: Database Examples

Example: Particle system



Tcl3D Module tcl3dOsg: Database Examples

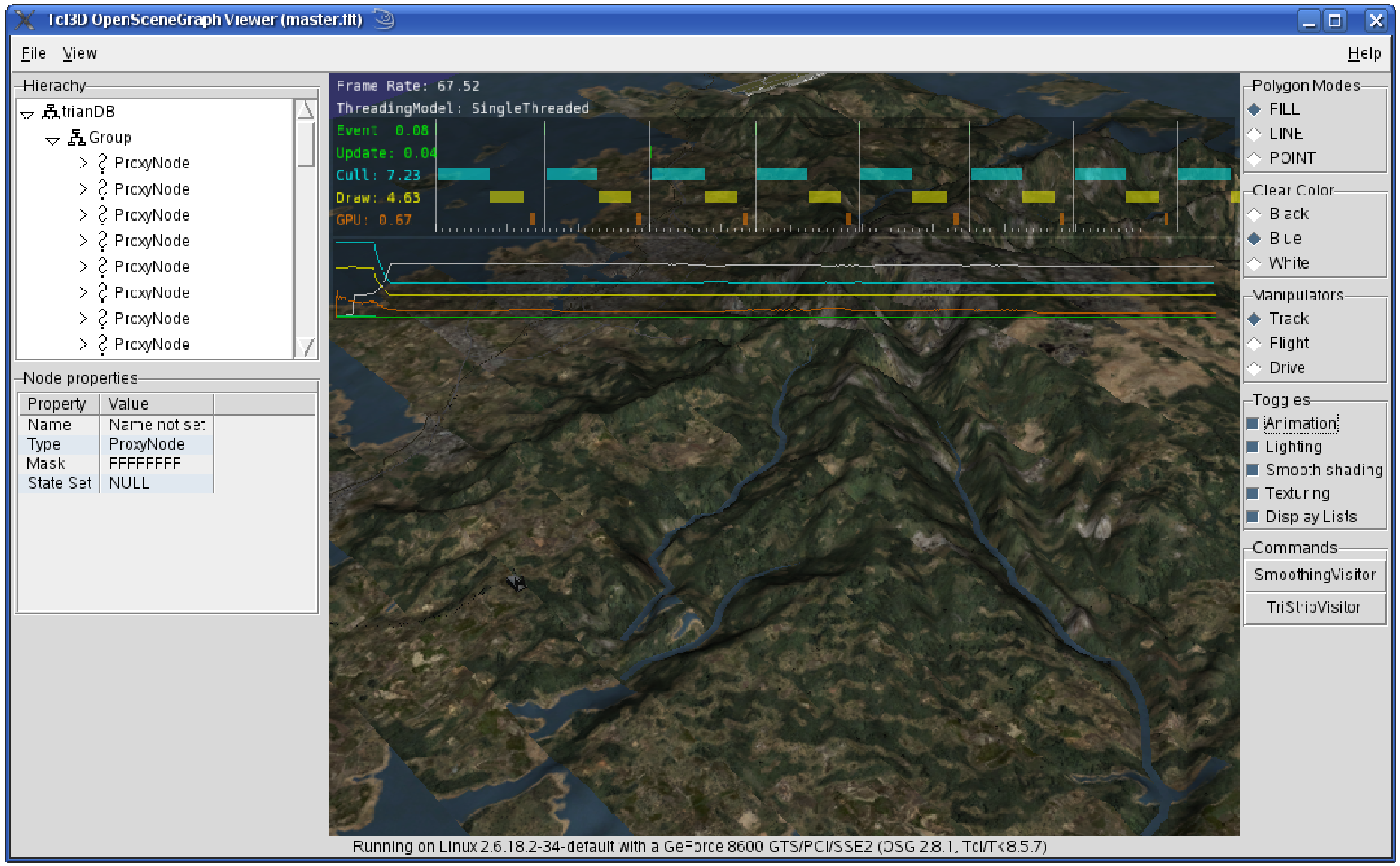
Example: Large Terrain Database (Simple statistics display)



Database from http://www.triangraphics.com

Tcl3D Module tcl3dOsg: Database Examples

Example: Large Terrain Database (Advanced statistics display)



Database from http://www.triangraphics.com

Tcl3D Module tcl3dOsg: Current State

tcl3dOsg uses latest official OpenSceneGraph version 2.8.1. (Released on 19th May 2009)

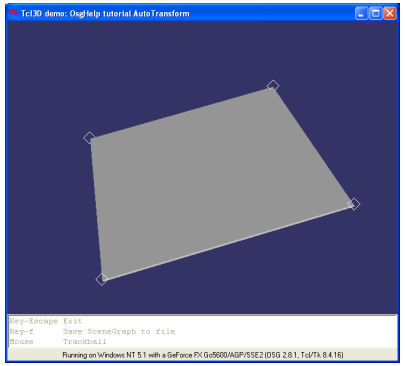
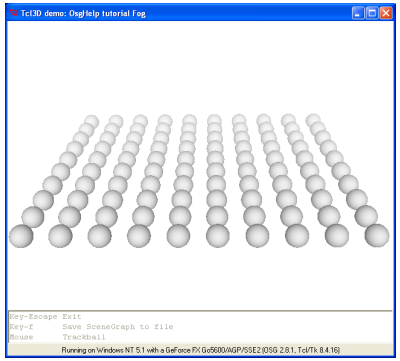
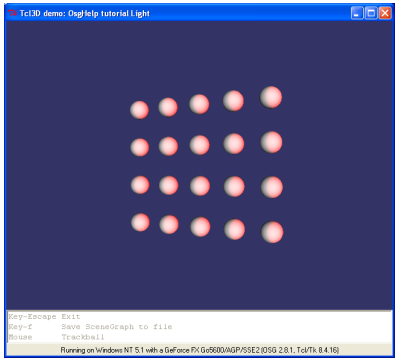
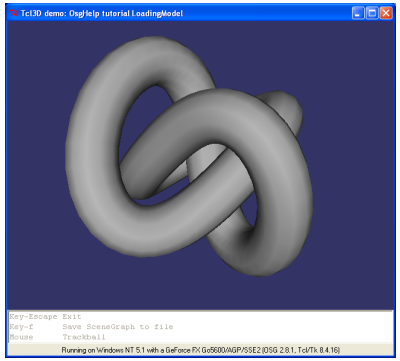
More than 30 OSG tutorials from different sources have been successfully ported.

A tile-based viewer application has been implemented.

Compiled and tested on Windows XP and Linux (32-bit and 64-bit).

Current state of wrapping:

```
Number of include files  : 462
Number of wrapped files  : 315
Number of unwrapped files: 147
```

Tcl3D Module tcl3dOsg: What's Next?

poSoft

Still lots of things to do:

Prepare first tcl3dOsg release.

Inclusion of demos in Tcl3D presentation framework.

Port more OSG programs for testing and demonstration.

Mac version has compile problems.

Improvements in wrapping.