# GDHE - Graphic Display for Hilare Experiments
# version 3.3

Matthieu Herrb
CNRS-LAAS

July 2003

# Contents

# Chapter 1

# Overview

This document introduces a new release of the GDHE software for 3D visualization of robotics applications [1].

This version is totally programmable by using of the Tcl/Tk [3, 4] scripting language. It uses the OpenGL [2] library to display the 3D primitives.

GDHE allows to build a 3D representation of the geometrical model of an environment and make it change with time. In order to achieve this, GDHE acts as a server that receives requests from a set of client processes. These requests describe the evolution of the model. Clients can be either modules that control a real system and that send data about the state of this system or simulation processes producing a simulated state of a virtual system.

GDHE accepts an unlimited number of clients, allowing to visualize simultaneously the state of multiple independent systems (for instance a multi-robots system).

Finally GDHE is able to record all the requests it receives from its clients to play them back later, without needing the clients.

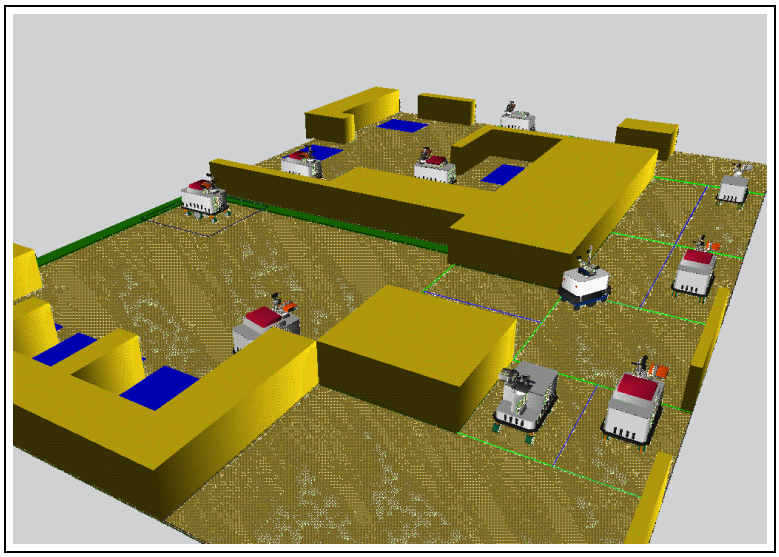

Figure 1.1: A sample multi-robot simulation displayed in GDHE

## 1.1 Starting GDHE

To start GDHE, the Unix environment variable GDHE must be defined and hold the path to the base directory of GDHE files. In the LAAS standard environment this is: `/usr/local/robots/share/gdhe`.

```
gdhe [-l log-file]
```

The `-l` option specifies the pathname of the log file in which all display requests are stored. The `reader` application then reads this log file back and sends them back for play back to GDHE.

**Warning!** GDHE overwrites the log file at each startup. In order to play back a log file, care should be taken to not overwrite it by using a different log file, or no log file at all.

## 1.2 The GDHE Tcl package

GDHE is also available as a Tcl/Tk extension, that can be loaded in the `wish` (or `rwish`) interpreter, using the Tcl `load` command:

```
load exec_pprefix/lib/gdhe.so
```

## 1.3 Init file

During startup GDHE reads the `.gdherc` file in the current directory, after loading the standard GDHE library (see chapter 2) and after creating the socket of the display server.

The `.gdherc` file can thus redefine some elements of the standard GDHE library.

To learn how to define an initial environment in the `.gdherc` file, please report to chapter 2.

## 1.4 Using the reader

The *reader* is a Unix application that can read log files created by GDHE (using the `-l` option) and play them back inside GDHE.

```
reader  [-c][-f file][-h host]
        [-n packets][-d millis][-p command][-l]
        [s factor]
```

| | |
|---|---|
| `-c` | compat mode: reads the old format log files. |
| `-f` *`file`* | reads the specified *file*. |
| `-h` *`host`* | connects to GDHE on specified `host`. |
| `-n` *`packets`* | reads the log file *packets* commands at a time. |
| `-d` *`millis`* | makes a pause of *millis* milliseconds between each command. |
| `-p` *`command`* | stops each time the specified `command` is found in the log file. |
| `-l` | lists the keywords recognized as `command` by the `-p` option. |
| `-s` *`factor`* | speeds the play back by the integer `factor`. |

# Chapter 2

# The standard GDHE application

GDHE includes a set of procedures developed at CNRS/LAAS for the visualization of experiments in the Robotics and Artificial Intelligence group.

## 2.1 Description of a standard environment

GDHE is highly programmable (see next chapters). However it is pre-configured with a set of functions that are well suited to represent the kind of environment in which the mobile robots of the LAAS. This default (or standard) configuration is described in this chapter.

### 2.1.1 The user interface

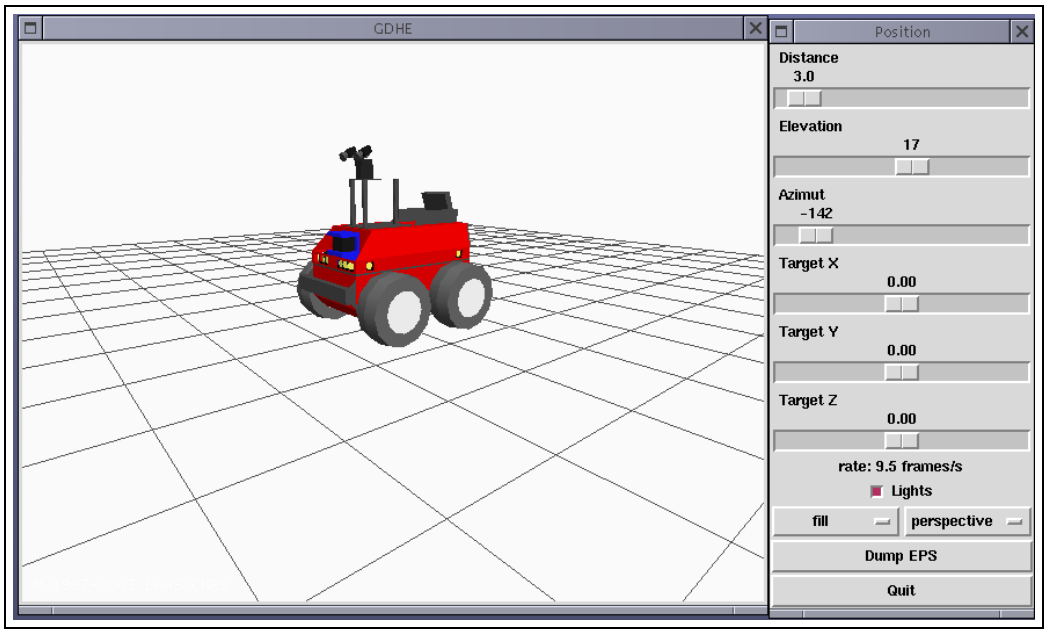


Figure 2.1: The default interface

The default GDHE user interface is made of 2 main windows: a window showing the scene

and a control window. The position of the observer can be modified by dragging with the left mouse button in the scene window.

The control panel offers the following settings:

**Distance:** the distance between the observer and the watched point in the scene.

**elevation:** the angle that the line of sight makes with the horizontal plane.

**azimut:** the direction of the line of sight measured around the vertical.

**target X:** $x$ coordinate of the point watched by the observer.

**target Y:** $y$ coordinate of the point watched by the observer.

**target Z:** $z$ coordinate of the point watched by the observer.

**Lights:** controls the computation of the lighting of the scene.

**Fill/Line:** selects rendering with filled facets or lines.

**Perspective/orthogonal:** computes the rendering of the scene using a perspective, resp. orthogonal projection.

**Dump EPS:** generates a dump in Encapsulated PostScript format of the visualization window.

**Quit:** quits the GDHE application.

### 2.1.2 Describing the scene

In the standard GDHE application, the scenes are split in two parts: a fixed environment and mobile objects. These two parts are described in Tcl variables. In order to make changes to a scene, the values of these variables can be modified.

#### Mobile objects

To represent mobile objects, GDHE is using several parallel Tcl arrays, indexed by object names, which can be arbitrary strings:

| | |
|---|---|
| pos | a Tcl array containing the position of each object. if pos has 3 elements, they are $xy\theta$, the position in the $z = 0$ plane of the object. If pos has six elements, they represent the Bryant angles and the $xyz$ position of the object in the 3D space. |
| robots | a Tcl array containing for each object the Tcl code to draw the object. For readability purposes, this code usually consists of a single procedure invocation, including arguments. The name of this array suggests that mobile objects are robots, but it can be anything. |
| platform | a Tcl array associating to each object some Tcl code to draw some instruments attached to the object. This array is particularly useful when drawing robots that can carry different type of instruments. platform does not need to be defined for each object. |
| walls | a Tcl array associating to each object a list of 2D segments in the local frame of the object. These segments (which typically correspond to obstacles perceived by a mobile robot) are drawn as walls (vertical rectangular facets) by GDHE. |
| rs_trajectory | a Tcl array describing a Reed and Shepp style (a succession of arcs and straight line segments) in the local frame of the object, drawn in the horizontal plane of the scene. |

The associative arrays of Tcl are used intensively here. An Object has a name, which is a string, and this name is used as an index in the various arrays described above to find out the various attributes of the object.

**Example:**

The xr4000 procedure draws a Nomadic XR4000 mobile robot. To place such a robot in the environment at coordinates $x = 2m$, $y = 1m$ and $\theta = 0$, just choose a name for it (for instance `r1` and evaluate the following Tcl code:

```
set robots(r1) xr4000
set pos(r1) {2.0 1.0 0.0}
```

The, to make this robot move, one just have to modify the value of pos(r1):

```
set pos(r1) {2.2 1.0 0.0}
```

This will redraw the XR4000 robot 20cm further down the $Ox$ axis.

### The environment

The standard GDHE application was designed to make use of environment models described in a format compatible with the format used by the European project MARTHA.

These environment models are represented by another set of Tcl variables. The initialization of these variables is generally done in a separate file, sourced during GDHE startup by `.gdherc`.

The various components of the environment are polygons, which are described using the formalism used for the MARTHA project. This formalism uses two arrays, one to hold the $xy$ coordinates of the vertices of the polygons and another one to hold a list of vertices indexes that build a polygon. The first array is called foo_vertices and the second one is foo_boundaries for a given element foo.

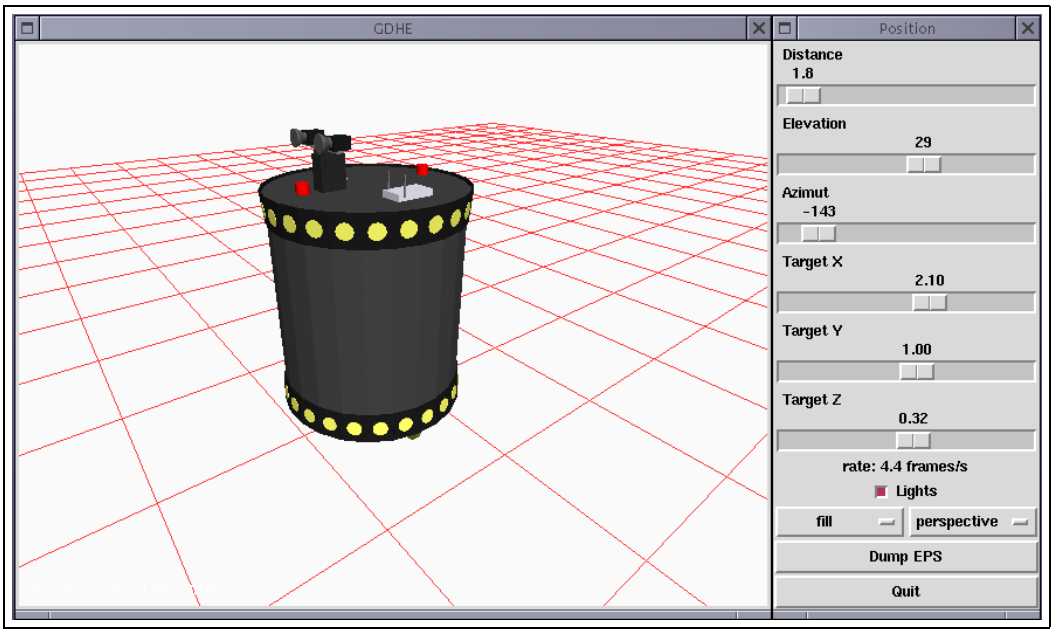


Figure 2.2: A sample result

Table 2.1: Structure of the environment model

| | |
|---|---|
| Cells | The environment in Martha is composed of *cells*, represented by the cell_vertices and cell_boundaries lists. |
| Stations | Parking areas for the robots are called *stations*. These are stored in the station_vertices and station_boundaries lists. |
| Obstacles | Static obstacles in the environment are described by the obstacle_vertices and obstacle_boundaries lists. |

## 2.2 Other Tcl procedures

The standard GDHE application provides a certain number of pre-defined objects (the LAAS robots and some accessories), but also some primitives that help to build new objects or to handle display windows.

### 2.2.1 object

object *name* { *definition* }

This function automates the definition, compilation and display of OpenGL display lists. During the first call to object an OpenGL display list is created and associated with *name* while *definition* is interpreted and displayed.

Further calls to the same object procedure only draws the recorded OpenGL display list, discarding the *definition*.

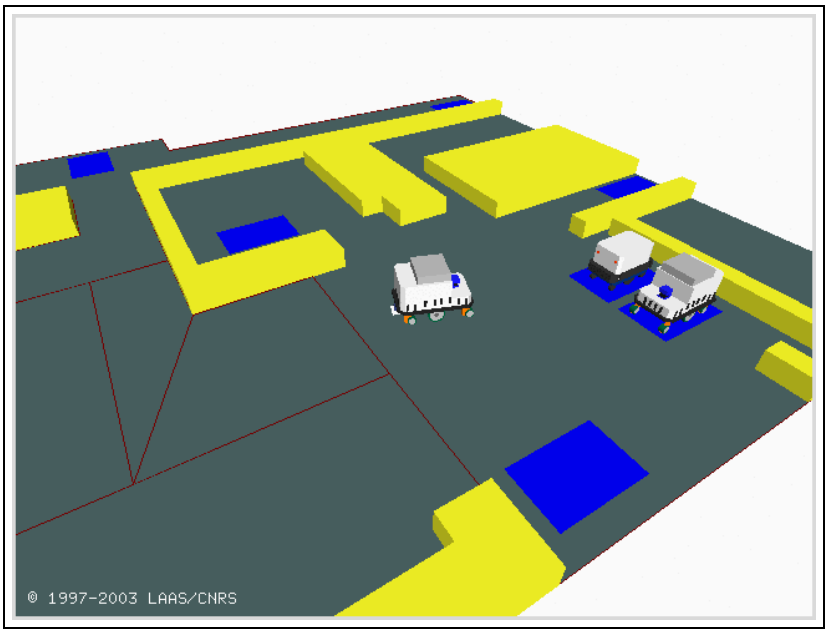


Figure 2.3: A sample Martha environment

**Remarks:**

- the *definition* part of an object should not contain a variable part. All variables will keep the value they had when the argument is first evaluated, no matter how the Tcl quoting is done.
- there are some limits in the current implementation on the redefinitions of an existing object.

**Example:**

The following code defines the desk procedure that draws a table. This procedure uses object to create an OpenGL display list called **Desk**.

```
proc desk {} {

  object Desk {
    pushMatrix
    translate 0.6 0.4 0
    color 200 200 100
    # Upper plane
    box 0 0 0.8 1.20 0.8 0.02
    # sides
    box -0.6 0 0 0.02 0.8 0.8
    box 0.6 0 0 0.02 0.8 0.8
    # bottom
    box 0 0.4 0.4 1.20 0.02 0.4
    popMatrix
```

```
  }
}
```

This procedure can be later referenced in the robots array to place two desks in the environment:

```
set robots(desk1) desk
set pos(desk1) { 0 0 -90 }
set robots(desk2) desk
set pos(desk2) { 0 1.5 -90 }
```

### 2.2.2 redrawAllWindows

redrawAllWindows

Triggers an immediate redisplay of all OpenGL windows displayed. This command is only useful l in a Tcl script to create an animation. When clients are sending requests to GDHE, redisplay is managed automatically when the global variable auto_redisplay is set to 1 (which is its default value).

## 2.3 Pre-defined models

This section describes the pre-defined objects in GDHE. These objects are all defined in the Models Tcl package. To add new objects to GDHE, the Tcl source file containing the definition of a Tcl procedure drawing this object at the origin should be placed in the `${GDHE}/tcl` Tcl package.

### 2.3.1 Mobile robots

#### Hilare 2

The h2 procedure displays the Hilare 2 robot from LAAS. The procedure h2_platform { *angle* } can be used as the value of the platform array to display the Hilare 2 laser scanner, oriented along *angle*.

#### Hilare 2bis

The h2bis procedure displays the Hilare 2bis robot from LAAS. The arm { *q1 q2 q3 q4 q5 q6* } can be used as the value of the platform array to display the manipulator arm of Hilare 2bis, with articular coordinates $q_1 \ldots q_6$.

#### Junior

The junior procedure displays the Junior robots from Midi-Robots. The junior_platform { *angle* } can be used as the value of the platform array to display the junior laser scanner, oriented along *angle*.

### Lama

The lama procedure display the Lama robot made from VNII-Transmach. This procedure has 5 parameters corresponding to the 5 internal degrees of freedom of the robot: $\alpha_1$, $\alpha_2$, $\beta_1$, $\beta_2$ et $\beta_3$. The lama_platine { *azi site* } procedure can be used as the value of the platform array to display the platform holding the stereo rig of lama, oriented among *azi* and *site* angles.

### XR4000

The xr4000 procedure displays a XR4000 robot from Nomadic Technologies.

### Scout

The scout procedure displays a Super-Scout robot from Nomadic Technologies.

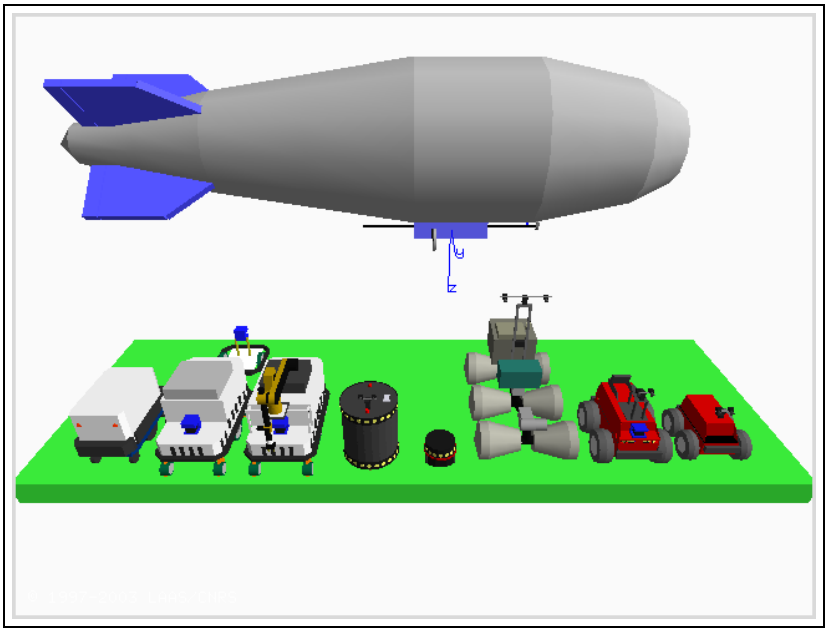

Figure 2.4: Know robots models

# Chapter 3

# Basic objects

To make it possible to create complex 3D models, GDHE provides a set of predefined primitive objects that can be used to build drawing procedures:

## 3.1 box

box $x_0$ $y_0$ $z_0$ *dx dy dz*

Draws a parallelepiped parallel to the axes. The ($x_0$ $y_0$ $z_0$) point is placed at the center of the lowest side.

## 3.2 cylinder

cylinder $x_0$ $y_0$ $z_0$ *axis* $d_1$ *[*$d_2$*] length [facets]*

Draws a cylinder or a cone parallel to the axes. ($x_0$ $y_0$ $z_0$) gives the center of the first facet. *axis* is **x**, **y** or **z** to indicate which is the main axis. $d_1$ and $d_2$ are the diameters at the two extremities. $d_2$ can be omitted, in which case a cylinder of diameter $d_1$ is produced. Finally *longueur* is the length of the cylinder.

If $d_1$ or $d_2$ has a negative value, the facet at the corresponding end of the cylinder is not drawn (and the corresponding diameter is $\|d_i\|$).

*facets* defines the number of facets used to approximate the cylinder. The default value is 12.

## 3.3 polygon

polygon $n$ $x_0$ $y_0$ ...

Draws a polygon in the $z = 0$ plane. The coordinates of the $n$ vertices are defined by ($x_0$ $y_0$ $x_1$ $y_1 \ldots$ $x_{n-1}$ $y_{n-1}$)

The normal of this polygon is oriented towards positive $z$.

If it has been declared beforehand (see concave), the polygon can be concave, but its edges should not cross themselves and there can be no double vertice (ie 2 vertices at exactly the same coordinates). Should this happen, the result is undefined.

## 3.4 polyline

polyline $n$ $x_0$ $y_0$ $z_0$ ...

Draws a sequence of 3D line segments. The coordinates of the $n$ vertices are given by $(x_0\ y_0\ x_1\ y_1 \ldots\ x_{n-1}\ y_{n-1})$.

## 3.5 prism

prism $n$ *dx dy dz x0 y0 z0* ...

Defines a prism which as the polygon with $n$ vertices $(x_0\ y_0\ z_0) \ldots$ as its basis and that extends towards the $(dx\ dy\ dz)$ vector.

If it has been declared beforehand (see concave), the polygon can be concave, but its edges should not cross themselves and there can be no double vertice (ie 2 vertices at exactly the same coordinates). Should this happen, the result is undefined.

## 3.6 sphere

sphere *x0 y0 z0 diam [facets]*

Draws a sphere centered on $(x_0\ y_0\ z_0)$ and with diameter *diam*.

*facets* defines the number of facets used to approximate the sphere. The default value is 12.

## 3.7 disk

disk *x0 y0 z0 axis diam [facets]*

Defines a disk centered on $(x_0\ y_0\ z_0)$, whose normal is oriented towards one of the axis as specified in *axis* (which ca be **x**, **y** or **z** and whose diameter is *diam*.

*facets* defines the number of facets used to approximate the disk. The default value is 12.

## 3.8 picture

picture *image* [ *width height* ]

Defines an image whose lower left corner will be in $(0, 0)$ in the $xOy$ plan from the Tk image called *image*. A *width* (length along the $x$ axis) and a *height* (length along the $y$ axis) can be optionnally defined.

This primitive should be used inside a display list for optimal performance. Otherwise it can be extremly unefficient.

## 3.9 drawString

drawString $[x\ y\ z]$ *string*

Draws the string given as argument starting at the specified coordinates or at the current origin if no coordinates are given.

The string is drawn using the default OpenGL font.

## 3.10 repere

repere $[length]$

Draws a frame[1] centered on the current origin. The *length* parameter defines the length of the drawn axis. The default value is 1.

[1] *repère* in french

# Chapter 4

# The GDHE protocol

GDHE accepts display requests from the network. These requests are defined by a specific protocol, inherited from a previous version of GDHE, that was dedicated to the STRADA application, in the frame of the Martha project (and even another, much older version).

In the current version of GDHE the compatibility with this older version have been maintained. A new request was added, that allows to evaluate an arbitrary Tcl expression. This new request is sufficient for all new applications that don't want to use the data types and representations from the Martha project.

The STRADA application is considering a multi-robots system in which robots are numbered from $r_0$ to $r_{n-1}$. There is a number of different robot types, also identified by numbers. Each robot is equipped with an orientable platform with one degree of freedom.

These robots are evolving in a static environment, known in advance, but they can discover and model new obstacles.

In this chapter, only the functions needed to used the new interface are described in details.

## 4.1 Requests

All requests and associated data structures are described in the `GDHE_packet.h` header file.

#### The `packet` structure

This structure holds an union of all data types accepted by the GDHE requests, plus two fields *which* and *command* that indicate respectively which objects a given command applies to and what is the command sent by this request.

#### Eval_expression

The eval_expression request is used by a client of GDHE to make it evaluate an arbitrary Tcl expression. It is a basic generic request that allows to extend the functionalities of GDHE, by providing a mean to modify arbitrary parts of the model of the scene that is displayed.

## 4.2 Client library

Prototypes of the client library of GDHE are given in the `GDHE_client_prot.h` file.

Table 4.1: List of the main GDHE requests

| Request | Parameter | Description |
|---|---|---|
| SELECT_ROBOT_TYPE | int | defines the kind of robot *which* |
| PLACE_ROBOT | position | places the *which* robot at the given position |
| ERASE_ROBOT | - | removes the *which* robot from the environment |
| PLACE_WALLS | v_segment | puts local obstacles around the *which* robot |
| ERASE_WALLS | - | removes local obstacles from the *which* robot |
| PLACE_RS_TRAJ | rs_trajectory | Draw an Reed&Sheep trajectory in front of the *which* robot |
| ERASE_RS_TRAJ | - | removes the trajectory for robot *which* |
| UPDATE_PER_OBST | polygon | Puts a global obstacle in the environment |
| ERASE_PER_OBST | - | removes a global obstacle |
| EVAL_EXPRESSION | char * | evaluates a Tcl expression |

All these functions return `TRUE` is everything went OK or `FALSE` in case of an error.

### 4.2.1 Initialization

A client should start by creating a connection to the GDHE server. For this, it needs to know the name of the machine on which GDHE is running.

```
extern int get_connection ( char *server_name );
```

### 4.2.2 Termination

When a client exits, it gets disconnected automatically from the GDHE server. It can be useful to disconnect it explicitly, using the disconnect function.

```
extern int disconnect(void);
```

### 4.2.3 Evaluating a Tcl expression

The main function of interest in the GDHE client is this one, that sends an expression to be evaluated by the server, expressed as a C string (null terminated):

```
extern int eval_expression ( char *expr );
```

The result of the evaluation is not available to the client, since the interface is designed to be asynchronous, for performance reasons.

# Chapter 5

# The Tcl-OpenGL interface

## 5.1 Introduction

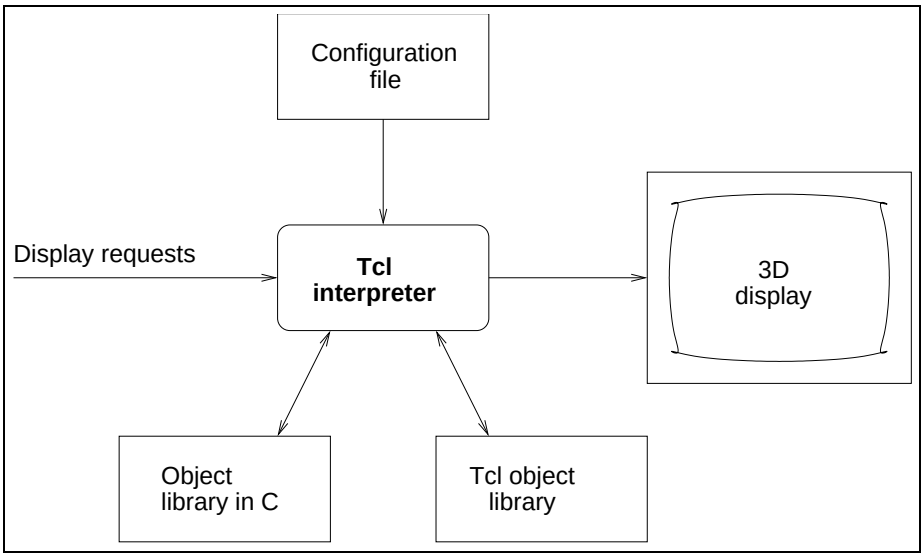


Figure 5.1: Architecture of GDHE

GDHE uses the Tcl/Tk scripting language to program several elements of it's user interface and to describe the scenes to be displayed.

The Tcl interpreter which is running asynchronously, is augmented by libraries of functions used to draw objects. These libraries are coded either in the C language or in Tcl itself. The interpreter is initialized by a configuration file that can be adapted for each application.

The Tcl interpreter acts as a server an can receive requests from its clients on a TCP/IP socket, or from the user who can send events through the graphical interface.

## 5.2 The display framework

GDHE offers a standard framework to display 3D data extracted either from a simulation or from real world data. By convention it uses a direct frame in which the $Oz$ axis represents

the vertical.

Several parameters available in OpenGL are fixed in GDHE in order to simplify the task of describing of the objects to display.

Specifically, in the *model view* transform from OpenGL, the part corresponding to the position of the observer relatively to the scene is represented by a polar transformation where the coordinates of the target point, the distance, the elevation and the azimuth of the observer are specified.

Also, the rendering model is kept simple. GDHE does not offer all the rendering parameters from OpenGL that could be used to obtain a more sophisticated (and more realistic) rendering.

## 5.3 OpenGL primitives

GDHE extends Tcl with a set of functions that provide an access to OpenGL primitives.

### 5.3.1 Handling of 3D display windows

GDHE uses the *Togl* widget, developed by Brian Paul to interface OpenGL and Tcl/Tk.

**togl**

togl *ident parameters...*

Creates an OpenGL widget (a new window) identified by *ident*. The Tcl options recognized by togl are:

**-width**

**-height** specify the size of the window.

**-ident** associate an identifier with this widget. *Warning !* all widgets that use the same *ident* share the same observer position. By default this identifier is the empty string "".

**-rgba** specifies if the widget will use the RGBA mode

**-double** specifies if the widget will use a double-buffer

**-depth** specifies if the widget will use a depth buffer

**-accum** specifies if the widget will use an accumulation buffer

**setObs**

| | | |
|---|---|---|
| *widget* setObs | -dist | *dist* |
| | -elev | *elev* |
| | -azi | *azi* |
| | -x | *x* |
| | -y | *y* |
| | -z | *z* |
| | -fovy | *fovy* |
| | -perspective\|-ortho | |
| | -projMat | *matrix* |
| | -viewMat | *matrix* |

Defines the position of the observer for the display window *widget*. The observer is looking at the $(xyz)$ target point and its position is given in polar coordinates by the distance *dist* and the two angles of elevation *elev* and azimuth *azi*. *fovy* defines the horizontal field of view of the observer.

**-perspective** specifies that a perspective projection will be used to render the scene. (This is the default).

**-ortho** specifies that an orthogonal projection will be used to render the scene.

**-projMat** allows to specify explicitly the projection matrix, rather than having OpenGL compute it from the individual parameters.

**-viewMat** allows to specify explicitly the OpenGL view matrix.

**redraw**

*widget* redraw *[force]*

Triggers a redraw of the specified Togl *widget*. If the *force* parameter is omitted, the redraw will be done *as soon as possible*, when the Tcl interpreter becomes idle. If the *force* parameter is present and has a `True` value, the redraw is immediate, interrupting any other activity in the Tcl interpreter.

For optimal performance of GDHE, it is strongly recommended to avoid setting the *force* parameter.

**dumpEps**

*widget* dumpEps *[-color] file*

Creates an encapsulated PostScript file with the contents of the OpenGL window *widget*. If the -color option is present, the resulting PostScript file will use color, otherwise it will be in grey shades.

This function overwrites silently *file* if it already exists.

The **Dump EPS** button of the standard user interface is using this function.

**dumpPpm**

*widget* dumpPpm *[-color] file*

Creates an image file in the *PPM* or *PGM* format from the contents of the OpenGL *widget*. If the -color option is present, the resulting file will be full color (PPM format), otherwise it will use shades of grey (PGM format).

This function overwrites silently *file* if it already exists.

### 5.3.2 Frames

GDHE offers a simple interface with the functions to manage the stack of frames of OpenGL.

**popMatrix**

popMatrix

Pops the current transform from the top of the stack.

**pushMatrix**

pushMatrix

Pushes the current transform on the top of the stack.

**rotate**

rotate *angle x y z*

Defines a rotation of value *angle* around the axis oriented along the $(xyz)$ directing vector.

**translate**

translate *dx dy dz*

Defines a translation.

**loadIdentity**

loadIdentity

Set the current transformation matrix to identity.

**loadMatrix**

loadMatrix *m11...*

Explicitley sets the current transformation matrix with the 16 values given as parameters.

**getMVMatrix**

getMVMatrix

**multMatrix**

multMatrix

### 5.3.3 Color

Currently, the only programmable attribute of surfaces displayed by GDHE is the color.

**color**

color *red green blue* [*alpha*]

Defines the current color. *Red*, *green* and *blue* are integers between 0 and 255. color defines both the ambient color and the diffuse reflexion color of the objects. The optional parameter *alpha* specifies the $\alpha$ channel value of the color. It is not really currently used by GDHE. (Some preliminary support for transparencies exists, but it's not really functional yet).

**clearColor**

clearColor *red green blue* [*alpha*]

Defines the background color of the image. *Red*, *green*, *blue* and *alpha* are integers between 0 and 255.

### 5.3.4 Other random procedures

**concave**

concave *option*

Indicates to prism- (prism §3.5) and polygon- (polygon §3.3) drawing primitives if the following objects are concave. If *option* is `True`, then objects can be concave and the tessellation function of the GLU library will be used to decompose the polygons into triangles before displaying them. If *option* is `False` the polygons are supposed to be convex and can thus be rendered directly.

If `concave false` was asserted and a concave object is evaluated, the displayed result is not predictable (and will generally look ugly).

**cullFace**

cullFace *option*

Defines if OpenGL should remove the back faces from the objects. If *option* is `True`, then back faces (oriented in the negative direction) are not displayed. If *option* is `False`, then back faces are displayed.

**setLights**

setLights

Re-reads the values of the Tcl variables that define the lighting of the scene. See § 5.4.2.

**sleep**

sleep *milliseconds*

Suspends the execution for the specified number of *milliseconds*. Don't use this command for too long pauses, because the application is totally unresponsive during this pause.

### 5.3.5 Display lists

The functions described here provide an interface with the primitives to manipulate *display lists* in OpenGL. They can be either used directly or through the more sophisticated object interface (see §2.2.1) to declare and display complicated shapes in one operation.

**newList**

newList $n$

Starts a new display list definition, numbered $n$. If there was already a display list with this number, it will be lost and replaced by the newly created one.

Please note that GDHE renders display lists while creating them (meaning that the OpenGL GL_COMPILE_AND_EXECUTE parameter is set).

**endList**

endList

Ends the definition of a display list.

**callList**

callList $n$

Displays the specified display list.

**genLists**

genLists $n$

Provides a mechanism to allocate unique display lists numbers. $n$ specifies how many indexes should be allocated. genLists returns the first allocated index. If $n$ indexes where requested, the $n-1$ other are following in sequence.

**deleteLists**

deleteLists *index [n]*

Frees the $n$ display lists starting at *index*. If $n$ is omitted, it defaults to 1.

## 5.4 Tcl procedures and variables

GDHE can be programmed with the Tcl language. Every GDHE application has to provide two specific Tcl procedures that are called by GDHE to perform its initialization and the drawings.

When GDHE starts, the `${GDHE}/tcl/setup.tcl` file is read by the Tcl interpreter. This file is part of the standard GDHE application and should generally not be modified in the normal use of the standard GDHE application.

In order to customize the standard GDHE application, the `.gdherc` file can be used. This file is read after `setup.tcl`, so that everything defined there can be overriden by `.gdherc`.

The `setup.tcl` file defines procedures and variables that are used aferwards by GDHE to manage the display. This section describes those procedures and variables.

### 5.4.1 Procedures called by Tcl

**setup**

setup

This Tcl procedule is called without parameters during GDHE startup, after reading the configuration file (`.gdherc`). The aim of this procedure is to set up the GDHE graphical user interface (create one or more OpenGL widgets and the associated control panels).

The standard GDHE application comes with a default setup procedure that opens one OpenGL window and the default control panel described at § 2.1.1. This behaviour can be altered by two global variables, gdheBase and gdheNoControlPanel. These variables can be set in `.gdherc`.

gdheBase is the name of a Tk widget that will be the father of the main OpenGL window (defaults to "."). In order to encapsulate GDHE in a pre-existing Tk application, just set gdheBase to the name of an existing Tk frame.

gdheNoControlPanel, if defined, prevents the creation of the standard control panel.

**draw_all**

draw_all *widget*

This Tcl procedure is called each time GDHE needs to redraw the contents of an OpenGL widget, either after the windowing system sent a redraw event or after an explicit call to the redraw procedure. It gets the name of the widget to redraw as an argument.

### 5.4.2 Variables used by GDHE

#### Automatic redraw

By default GDHE redraws the contents of the OpenGL window automatically after receiving and handling each request of a client. This automatic redrawing is controlled by the auto_redisplay variable which has the default value of 1.

If a client application needs to send several requests without triggering a redraw, it can set this variable to 0 and then explicitely call redraw for a specific window, or redrawAllWindows for all GDHE windows.

#### Lighting

GDHE uses Tcl variables to specify the lighting parameters of the scene. Since OpenGL can handle up to eight light sources, these variables are arrays in the Tcl namespace Gdhe::lights, whose indexes are **LIGHT0** to **LIGHT7**.

**Gdhe::lights::position** defines the position of the light sources. Each element of the array is a list of three or four elements that define the ($x$ $y$ $z$) coordinates or the direction of the source.

If the list is composed of three elements ou if the fourth element is equal to 0, then the light source is placed at the infinity and the three first elements define the direction of the source in the GDHE main frame. In this frame, (0 0 1) is a light source placed vertically above the scene.

If the fourth element is not 0, the source is placed at the point with coordinates ($x$ $y$ $z$).

**Gdhe::lights::ambient** defines the color and intensity of the ambient component of each light source.

**Gdhe::lights::diffuse** defines the color and intensity of the diffuse component (reflected by the objets) of each light source.

**Gdhe::lights::enabled** allows to enable or disable each of the eight sources individually. Each value is interpreted as a boolean.

# Chapter 6

# Extension modules

GDHE can be extended using Tcl modules that will provide new objects types or new procedures.

The procedure to load a GDHE extension is the same as for any Tcl module. Place:

```
package require module_name
```

in the GDHE init file `.gdherc` before using the procedures and variables it defines.

This chapter describes the existing extension modules.

## 6.1 Numerical Terrain Models

The terrain module allows to display numerical terrain models.

### 6.1.1 readTerrain

```
readTerrain name filename
```

Reads a terrain model in the file format used by LAAS EDEN experiments stored in the file specified by *filename* and creates a terrain with name *name* in GDHE's memory.

The color of each terrain vertice is obtained from the type associated in the data with this vertice, used as an index in the mntColor Tcl array.

### 6.1.2 readTerrainGeroms

```
readTerrainGeroms name filename
```

Same as readTerrain, execpt that the expected format of the terrain model is the one produced by the GEROMS tools.

### 6.1.3 terrain

```
terrain name
```

Draw the named terrain in the current scene. If no terrain with the given name exists, an error is generated.

### 6.1.4 deleteTerrain

deleteTerrain *name*

Destroys the terrain model named by *name* and frees the associated memory resources.

### 6.1.5 Terrains types

The mntColor array defines colors associated with each terrain type. Since the interpretation of the possible terrain types values is leaved free to the user, it's also the user's role to associate appropriate colors with the various possibles values for each vertice's type.

Each element of this array should be an a triple { *r g b* } where eqch component between 0 and 255 corresponds to the red, green and blue components respectively.

## 6.2 Planet

The planet module allows to draw a sphere representing a planet : a picture of the surface is projected as a texture on the outer surface of the sphere.

### 6.2.1 initPlanet

*widget* InitPlanet `[-interior]` `[-radius` *radius*`]` *[filename]*

Computes in the specified OpenGL widget an object representing a planet, using the picture stored in *filename*, or a default picture of the earth if *filename* is not specified. The **BUILTIN** string can be used to refer to this image.

GDHE provide some texture files in the `${GDHE}/images` directory:

| | |
|---|---|
| earth.xbm | simple monochrom model of the earth |
| earth.xpm | a colorful view of the earth from satellite images |
| earthcld.xpm | clouds from the high atmosphere of the earth |

The `-interior` parameter indicates to OpenGL that the inner face of the sphere is to used instead of the outer one. This is allows to create cloudy skys effects.

The `-radius` parameter specifies the radius of the sphere. The default is to create a sphere of 10m of radius.

Since this function only defines the display list for on OpenGL widget, it should normally never be called directly. Use the Tcl planet procedure instead.

### 6.2.2 drawPlanet

drawPlanet

Draws the current planet in the current OpenGL window at position (0 0 0).

This function should normally not be called directly. Use the planet procedure instead, or build you own procedure using planet as a model.

### 6.2.3 planet

planet *name filename*

This procedure draws a planet. It is suitable as a value in the robots array. *name* gives a name to the objet and *file* defints the name of the file containing the image to be used as a texture.

**Example:**

```
set robots(earth) { planet earth $env(GDHE)/images/earth.xpm }
set pos(earth) { 0 0 0 }
```

### 6.2.4 Bugs

- There is a problem when handling multiple windows
- This should be standardized with other similar GDHE modules

## 6.3 Mpeg

The Mpeg module helps in producing animations in the MPEG-1 format from GDHE.

Capturing an animation cannot be done in real time, given the computing time needed to compress the frames into MPEG streams. You need to be able to produce the images that will compose the animation off-line.

The folling steps are needed to produce an animation:

1. Create the OpenGL window with the desirated size of the animation
2. Initialize the capture
3. Generate each image and capture it one by one,
4. End the capture

*This module is currently more or less broken. The MPEG library used to compress the images is not good enough to produce good quality results. A replacement modules exists, but it is not finished and not documented either. You should bother the autors of GDHE to get it finished if you need it !*

# Bibliography

[1] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi robot cooperation in the martha project. *IEEE Robotics and Automation Magazine*, 1997.

[2] J. Nieder, T. Davis, and M. Woo. *OpenGL Programming Guide, The Official Guide to Learning OpenGL, Release 1.* Addison Wesley, 1993.

[3] J. K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1994.

[4] B. Welch. *Practical Programming in Tcl and Tk.* Prentice Hal, 1996.

# Index