GNUbik

GNUbik

This utility displays a 3 dimensional image of a magic cube — similar to the original introduced to the world by its Hungarian inventor, Erno Rubik. The user rotates sections of the cube, and attempts to solve the puzzle, by moving all the blocks to their correct positions and orientations, resulting in a cube with a single colour on each face.

1 Running the Program

Invoke GNUbik, by typing its name `gnubik` into a command line. Or you can set up a window manager menu, or file manager application, to run it. Whatever the chosen method of starting GNUbik, it must have access to a display with which it can render its images. Being an inherently graphic program, it won't do much otherwise.

GNUbik accepts all the standard X resource overrides such as `-geometry` *ww*x*hh*+*xx*+*yy* as well as all those specific to the program itself. See Section 3.1 [Options], page 4.

2 What it Does

GNUbik is graphic, in both its display, and its interface. User interactions are immediately made visible on the display.

3 Configuration

3.1 Options

The following options may be specified on the command line at invocation. You can access some of these parameters from a menu whilst the program is running.

`-z` *n*

`--size=`*n* Draws a cube of dimension n, where n is positive. The default value of n is 3.

`-a` *m*

`--animation=`*m*
When rotating parts of the cube, show m intermediate positions. The default for this value is 2. Setting to a higher value, will give smoother animations, but will slow down the whole program.

`-s`

`--solved` Starts with the cube already solved. By default, it starts in a random position.

`-h`

`--help` Display a help message and exit.

`-v`

`--version`
Display version number and then exit.

3.2 Resources

GNUbik uses the Athena widget set. Any widget's resource may be set to control menu labels, background colours etc. In addition, the following application resources set the colours of the cube faces:

```
! Colours of the cube faces

! resource    default
!
gnubik.color0: Red
gnubik.color1: Green
gnubik.color2: Blue
gnubik.color3: Cyan
gnubik.color4: Magenta
gnubik.color5: Yellow
```

4 Examining the Cube

You can change the viewing position of the cube using the mouse or the keyboard.

4.1 Using the keyboard to rotate the cube

You can re-orientate the cube using the following keys. The ⟨Up⟩ and ⟨Down⟩ keys rotate the cube about the horizontal axis. The ⟨Left⟩ and ⟨Right⟩ keys rotate the cube about the vertical axis. Holding down the ⟨Shift⟩ key whilst pressing the ⟨Left⟩ or ⟨Right⟩ arrow keys rotates the cube about the z axis.

4.2 Using the mouse to rotate the cube

Another way to rotate the whole cube is to use the mouse. Hold down button 1, and move the mouse anywhere on the window. This way, you can rotate the cube horizontally and vertically, If your mouse has a scroll wheel, it can be used to rotate the cube about the z axis.

5 Manipulating the Cube

To manipulate the cube, you need to select a block to move, and a direction to turn it. Place the mouse cursor on a block to select it, then move the cursor against the edge of the block towards the direction you want to turn. Click Button 1 when you're ready to turn make the move. At all times the mouse cursor will point in the direction of motion.

6 Running scripts

As well as manipulating the cube by hand as described in the previous chapter, you can also run scripts that perform automatic manipulations right in front of your eyes. Currently there are scripts available which randomize the cube, and which solve 3x3x3 cubes.

When GNUbik starts up, it reads files ending in a `.scm` suffix in the `.../share/gnubik/scripts` directory (system-wide files) and in the `.gnubik/scripts` directory under your home directory (personal files). These files are scheme scripts, and when they run they register themselves with the system so that items appear on the main GNUbik menu under the heading `Script-fu`. Making selections from this menu runs the appropriate script.

7 Writing scripts

8 Introduction to script-fu

GNUbik's script-fu is implemented with `GUILE`, GNU's ubiquitous extension language, which in turn implements `scheme`, a programming language descended from `lisp`.

In simple words, a script works as follows. When GNUbik starts up, it looks for scripts in standard places and executes each one it finds. In turn, the scripts define some procedures, and register some of their defined procedures with GNUbik, which associates the procedure with an item under the `Script-fu` menu.

When you select a menu item, GNUbik executes the procedure that was registered. The procedure will then request an object from GNUbik that reflects the current state of the cube. The procedure then works out a list of moves that it wants to impose on the cube, and sends this list to GNUbik. During the development of this list, the script may manipulate the cube reflection object so that it can keep track of the moves it makes. However, changes to the reflection object are never seen directly by the GNUbik program.

Once the script exits, GNUbik will cause all the movements which the script requested to take place on the on-screen cube, right in front of your eyes!

9 Initialization of the script-fu system

When GNUbik starts up, it scans the following directories in the order given for files with a `.scm` extension, and executes them in the order found (not necessarily alphabetical!) The directories searched are

- `.../share/gnubik/guile`
- `.../share/gnubik/scripts`
- `~/.gnubik/scripts`

Note that the exact location of the share directory is system dependent. The `share/gnubik/guile` directory is for scripts which the GNUbik program itself needs to function. They should not be changed once the GNUbik package is installed in a system. The `share/gnubik/scripts` directory is the place for system administrators to put scripts that all users can see on their GNUbik menus. Finally, the `.gnubik/scripts` directory is located under your home directory, and may contain any scripts that you have written, and which will only appear on the menu when you start the program up.

When the system finds a script, that script is executed and must use the following function to register itself with an item on GNUbik's menu bar.

gnubik-register-script *menu-path code* [Procedure]

menu-path should be a string indicating how the script should appear under the `Script-fu` menu, using `/` characters to indicate sub-menus and `_` characters in front of a letter to indicate that that letter should be used as the keyboard accelerator for the menu item. For example, the string `solvers/my _solver` will create an item under the `solvers` sub-menu, with `s` as the keyboard shortcut.

The second argument *code* should be a string specifying `scheme` code to run when the menu item is selected. It should usually be the name of a procedure which is the entry point for the script.

10 The basic (low-level) interface

This chapter describes the basic interface that GNUbik provides to the `scheme` world. This interface can be used with any cube geometry that GNUbik supports, but is not intuitive for humans and the API may change in the future.

If the API does change (to make it more consistent!) the version number of cube geometry in a cube object will also change, so any script which uses these functions directly should obtain a cube object from GNUbik and check the geometry version number.

10.1 The object given to the scheme world

In order to obtain a reflection of the current state of the cube, a script must call

gnubik-cube-state [Procedure]

This will return a `scheme` object, which is in fact a cons cell whose first component is a list describing the geometry of the cube, and the second component is a vector of six vectors of nxn items, where n is the size of the cube.

The geometry list consists of a version number, which will change if the definition of the cube object ever changes (scripts should check this to ensure compatibility or else things will surely break); the current version is 1 (one). The next item in the list is the dimensionality of the cube, which currently is always 3. The list then holds three more numbers describing the size of the cube in each direction. So a current 3x3x3 cube will have the geometry list `(1 3 3 3 3)`.

The second component of the cube cons cell is a vector of six vectors, one for each face. If you were to look at a cube with face index 4 towards you (remember vector indeces are zero based), then the top face would have index 0 (zero), the bottom 1 (one), the left 2, the right 3, and the back 5. Got it? Stay there. On faces 0 (zero), 3, and 4, the panels are numbered from left to right and top to bottom, so that panel 0 (zero) is at top-left and panel 5 is middle-right, for example. On the other faces, the panels are in the corresponding positions underneath their opposite faces, which means in practice that the numbering is from right to left, top to bottom. For example, the second panel (index 1) on face 2 is at the top. Finally, the 'top' edge of the top panel is against face 2. The elements of the vectors for each page represent the colour that the panels of that face are showing, numbered from 0 (zero) to 5 inclusive (the correspondence between these numbers and the actual colours seen on the cube on the screen is not provided by GNUbik to the scripts).

Whew.

If that was too sticky and you are using a 3x3x3 cube, then you should proceed to the next chapter which presents a more comfortable (and stable) interface.

10.2 Looking at the colours on the faces

Once you have a cube object, the following procedures are available to facilitate access to it.

get-colour *cube face index* [Procedure]

This procedure gets the colour of the *cube* at panel *index* on *face*, where the meaning of the latter two arguments is described above.

set-colour! *cube face index colour* [Procedure]

This procedure sets the colour of the given face to *colour*. Note that the change takes place only in the `scheme` *cube* object, not in the actual cube that the user is seeing on her screen.

10.3 Moving the object on the screen

When a script has determined a set of moves which are to be made on a cube, it calls the following procedure to get GNUbik to put the moves into its internal move buffer.

gnubik-rotate-animated *move-list* [Procedure]

The *move-list* is a list of lists of three items. For any one move (a quarter turn of a slice of the cube), the first item in the specification is the face which lies parallel to the desired slice, and may be 0 (zero), 1 (one) or two corresponding to the back, right and bottom faces. The second item in the specification is the slice number parallel to that plane; the slice next to the specified face has number (`n-1`) where `n` is the size of the cube, and the slices below are separated by (-2). For example, for a cube with three blocks per edge (a 3x3x3 cube, say), the slices are numbered 2, 0 and -2. The final component in a move specification may be 0 (zero) or 1 (one) depending on whether the slice should be moved anticlockwise or clockwise, respectively.

Note that the above function may be called as many times as desired during the execution of a script, but that the moves will not actually take place until the script finishes.

gnubik-rotate-fast *move-list* [Procedure]

This performs the same function as above, except that the moves take place instantly as soon as the script finishes, instead of being animated on the screen.

11 FLUBRD symbolic interface for 3x3x3 cubes

The following functions provide access to a cube object in a way which hides the gory details of the representation. This makes it easier for human beings to write code, and also guards against changes in the low-level API described in the previous chapter.

Unfortunately, it currently only works on 3x3x3 cubes.

The interface uses the common `FLUBRD` notation to refer to the various faces and panels on the cube. The letters correspond respectively to the words `front`, `left`, `up`, `back`, `right` and `down`, and where `up` and `down` can be read as `top` and `bottom`, respectively.

11.1 Looking at the colours on the faces

The following procedure returns the colour on the panel addressed symbolically by the letters of a given string.

get-colour-symbolic *cube symbol* [Procedure]

The *symbol* is a string consisting of one, two or three letters. If one letter is supplied, the colour of the panel at the centre of the face indicated by the letter is returned. If two letters are supplied, the colour returned is of the edge panel touching the face given by the second letter. If three letters are supplied, it is the colour of the corner panel which touches the edges indicated by the second two letters.

Note that the letters of the symbol must all be lower case. A corresponding function is supplied to make changes to the cube object.

set-colour-symbolic! *cube symbol colour* [Procedure]

This procedure sets the colour of the panel indicated by *symbol*, defined as above, to *colour*.

Note that changing the colour of a cube does not cause any other changes to take effect. In particular, no rotations of the on-screen cube are initiated.

11.2 Moving the faces

The following function allows scripts to request that faces of the cube are rotated by a quarter turn.

move-face *cube symbol* [Procedure]

Here, *symbol* is a string of one or two characters. The first character should always be a lower case letter which indicates which face to move. The default is to turn the face clockwise. This will also occur if the second character supplied is a space or a `+`. Only if a second character is supplied and is a `-` will the face be turned anti-clockwise.

Once a script has finished declaring moves in this way, GNUbik will send them to the display buffer for on-screen animation after the script calls

execute-move-buffer! [Procdure]

Cause all moves created with the `move-face` procedure to be applied to the on-screen cube.

Note that the on-screen cube will not actually display any change until the script exits.

The move buffer can be accessed directly via the symbol `move-buffer`. It is a list of moves that the low-level procedure `gnubik-rotate-animated` expects. Accessing and manipulating it directly before calling `execute-move-buffer!` may allow a script to perform some post-processing optimizations on the list of moves, for example. Alternatively, if the script decides to duck out or start again, the following procedure may be called to clear the buffer without sending any move notifications to GNUbik.

clear-move-buffer! [Procedure]

Empty the *move-buffer* without sending any move requests to the GNUbik program.

12 Hints and tips for script developers

12.1 'Live' menu entries

In order to eliminate the need to shutdown, startup, and then make selections from the menus every time a change is made to a script under development, you can create a script which reads another script every time it is invoked. This way any changes made to the latter script will manifest themselves immediately, speeding up the development cycle.

Suppose your development script is `/path/to/dev_script` (this can be anywhere at all in the filesystem). You can create a menu entry `Dev. test` to load and run this script by putting the following lines into a file called `~/.gnubik/scripts/dev-test.scm`

```
(define (run-dev-test)
   (load "/path/to/dev_script")
   (run-dev-script-proc))

(gnubik-register-script "Dev. test" "run-dev-test")
```

and then restarting GNUbik. Note that `run-dev-script-proc` is the name of the procedure you would normally have run directly from the menu (i.e. it is the one you would have passed to `gnubik-register-script`).