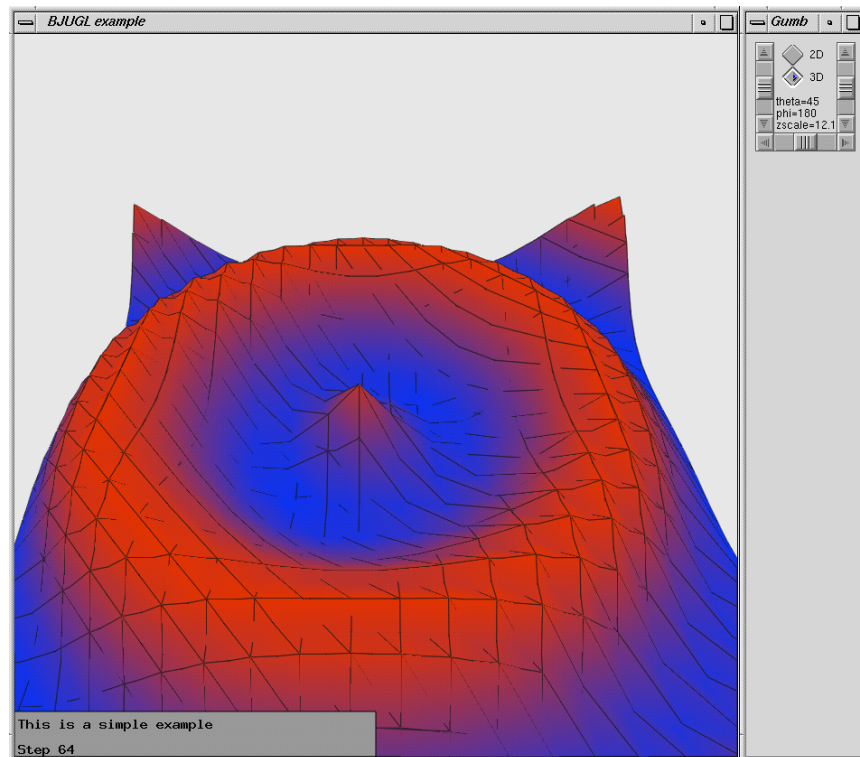


BJUGL and BUI

Bertram Lückehe and Jörn Behrens

April 5, 2000



Contents

1	What is BJUGL and what does BUI?	2
2	The Name	2
3	Concept of BJUGL	3
4	Display modes of BJUGL	3
5	How to use BJUGL	3
6	Function reference of BJUGL	4
7	Function reference of BUI	5
8	Customising BUI.SetParams_	5
9	Reporting bugs	6
A	Copyright Notice	6
B	License	6
C	Warranty	6

1 What is BJUGL and what does BUI?

BJUGL is a graphics library designed for drawing two or three dimensional graphics of a scalar function of two variables and a two dimensional vector field on a variable grid. It can draw graphics on the screen using OpenGL, save these graphics in a tiff (Tagged Image File Format) file format or produce printable output in color PostScript. It also offers you a simple graphical user interface made with GLUT and mui called BUI.

BJUGL has been designed for real time visualization, i.e. graphics output takes place, while the simulation is running. In technical terms, a second process is spawned from the main simulation process, that takes care for the user interaction an the rendering. BJUGL takes advantage of accelerated graphics hardware, as it builds on top of OpenGL.

BUI provides only one function, that opens a simple configurable user interface for inserting parameters to a simulation program interactively.

2 The Name

BJUGL stands for **B**ertram's and **J**örn's **U**ltrasimple **G**raphics **L**ibrary. BUI is the acronym for **B**ertram's **U**ser **I**nterface.

3 Concept of BJUGL

To introduce the functionality of BJUGL, let us describe some abstract concepts behind its design. We assume to have given some functions:

$$x : I \rightarrow \mathbb{R}, \quad (1)$$

$$y : I \rightarrow \mathbb{R}, \quad (2)$$

$$u : I \rightarrow \mathbb{R}, \quad (3)$$

$$v : I \rightarrow \mathbb{R}, \quad (4)$$

$$f : I \rightarrow \mathbb{R}, \quad (5)$$

$$n : J \rightarrow I^L, \quad (6)$$

where the pair $(x(i), y(i))$ denotes the (2-dimensional) coordinates of a grid node, $(u(i), v(i))$ a vector-valued function value at each node (eg. a wind field), and $f(i)$ is a scalar function on the grid nodes. I is a set of indices $I = \{i = 1, 2, 3, \dots, N\}$ denoting the grid nodes, J is an other set of indices denoting the cells of the grid (triangles, rectangles), $J = \{j = 1, 2, 3, \dots, M\}$. The function n maps the cell's local node indices $\{1, 2, \dots, L\}$ (L the number of cell vertices, $L = 3$ for triangles, $L = 4$ for rectangles, etc.) to the global node indices, thus $n(j) = (i_1, i_2, i_3)$ (where $L = 3$) means that cell no. j is formed by the three nodes with indices i_1 , i_2 , and i_3 . Given these functions, we can visualize very easily the following:

- The grid, formed by the list of cells J in 2D and 3D (z -axis corresponds to f -value at each node).
- The scalar function f in 2D (f -value represented by color) and 3D (f -value represented by color and z -value), with or without the grid outlines.
- The vectorfield (using flags at each gridpoint) in 2D, with or without the grid outlines.

4 Display modes of BJUGL

Several display modes can be chosen. In order to use BJUGL as a real time visualization tool, one can choose the OpenGL based display to the screen. This mode is also required in order to save tagged image file format (TIFF) graphics files to disk. TIFF files are just bitmapped hardcopies of the OpenGL output. Finally one can choose PostScript output format. This can be used even without the OpenGL library, but PostScript output only supports 2D graphics.

5 How to use BJUGL

The library was written using C. You can directly implent it into your own C or fortran programs. There is an include file called BJUGL.h that contains the function definitions needed for C programs. When linking your program you have to link this library to it and the following others:

- libGLU.a and libGL.a from OpenGL
- libglut.a and libmui.a from GLUT
- libtiff.a - the library for the tiff output
- libXmu.a, libXi.a, libXext.a and libX11.a from the X-Windows system

First you have to call `BJG_GraphicsInit_`. This starts a child process and opens a graphics window. The graphics is drawn by calling `BJG_GraphicsDraw_`. This function does not do the drawing itself, but puts your graphics data into some shared memory and tells the graphics process launched by `BJG_GraphicsInit_` to perform the drawing. `BJG_GraphicsQuit_` at least tells the graphics process to close the window and terminate.

For using `BUI_SetParams_` you don't need to call `BJG_GraphicsInit_` before.

Using `BJUGL` from your Fortran90 Programs is equally simple. The File `BJUGL.f90` implements a Fortran90 Module, providing access to some constants a data structure, and external function declarations. The function's names are similar to the C-interface's names, only the underscore at the end is omitted: `BJG_GraphicsInit`, `BJG_GraphicsDraw`, `BJG_GraphicsQuit`, and `BUI_SetParams`.

6 Function reference of BJUGL

I have written this in C function style. All parameters are passed via pointers, because Fortran90 does it this way and so the library can be used from C and Fortran90 programs. Wether a pointer points to a single value or to an arrays should be extracted from the type of data expected in the parameter list.

int BJG_GraphicsInit_(int *nn, float *xx, float *yy, int *wtl, char *wtext, int *mtl, char *mskfile, int *drwtyp, *initial_drawmode, *initial_dimension)

Function : Initialise graphics output and open output window (only if `DRAWMODE_GL` is set)

Parameters :

***nn** number of boundary nodes

***xx** x-coordinates of boundary

***yy** y-coordinates of boundary

***wtl** number of characters in window title string

***wtext** window description text (title string)

***mtl** number of characters in land data file name

***mskfile** name of land data file

***drwtyp** set initial drawing type. This is a combination of flags defined in `BJUGL.h`. They can be combined using logical OR. The scalar and vector field cannot be drawn at the same time, you should not combine these flags. Valid flags are:

FLAGGRID draw grid

FLAGVECTOR draw vector field

FLAGSCALAR draw scalar field

FLAGCONTINENT draw continent lines

***initial_drawmode** a combination of flags that define which type of graphics output is generated. The flags may be combined with logical OR. They are defined in `BJUGL.h`

DRAWMODE_GL draw OpenGL (screen)

DRAWMODE_TIFF write tiff images to files (only possible, if also OpenGL is activated)

DRAWMODE_PS write PostScript images to files

***initial_dimension** 2 or 3 for 2 or 3D graphics.

Return value : Returns 0 on succes, -1 if there was an error in allocating memory or starting the graphics child process.

int BJG_GraphicsDraw_(int *nnd, int *ntr, int *nnv, float *xl, float *yl, float *xx, float *yy, float *val, float *xval, float *yval, int *ftl, char *framtext, int *tnl, char *tn, struct des-text *description)

Function : Draw image with new graphics data

Parameter :

- ***nd** number of nodes per grid polygon, i.E. 3 for a triangular grid. Values other than 3 have not been tested yet...
- ***ntr** number of polygons of the grid
- ***nv** number of vectors of vector field
- ***xl** x-coordinates of grid nodes
- ***yl** y-coordinates of grid nodes
- ***xx** x-coordinates of vectors
- ***yy** y-coordinates of vectors
- ***val** values of the scalar function at the grid nodes coordinates
- ***xval** x values of vectors
- ***yval** y values of vectors
- ***ftl** and ***framtext** frame text - uh, this is not used any more - we should kick it out!!
- ***tnl** length of output file name
- ***tn** output file name. A suffix .tiff is added for tiff images and .ps for PostScript graphics
- ***description** description text of graphics, hmm I still have to write about the format of the description text structure...

Return value : The function returns 0 on succes, 1 if the graphics process has quitted or was killed and -1 if an error occured in allocating memory for the graphics data.

void BJJ_GraphicsQuit_()

Function : terminate graphics process and close graphics windows

Parameters : none

7 Function reference of BUI

void BUI_SetParams_(struct control_struct *control)

Function: Open a window where the user can change various parameters of a structure. For how to customise this function see section 8. `BUI_SetParams_` is **not** included in the BJUGL library, we have put it into an own lib called BUI.

Parameters :

- ***control** structure to be modified

8 Customising BUI_SetParams_

The function `BUI_SetParams_` opens a window and offers the user a graphical user interface to change some parameters that can control your program. You can define the number and type of widgets and the data types the user can enter there. This is done in the following way:

You have to make some adjustments in the source code of the library and you have to recompile it. All parameters to be edited by the user have to be contained in a structure that is called `control_struct`. It is defined in the file `control.h`. Here you have to put the definition of your control structure. You must use the name `control_struct`. Next you edit the file `buttons.h`. Here are the most important parameters `SP_NUMBUTTONS` the number of buttons you want to have and `SP_SAVENAME`. This tells `BUI_SetParams_` where in your control structure it can find a filename

for writing the parameters if the users presses the save button. `SP_BUTSPERCOL` define how many buttons should be put into one column and `SP_WINSIZEX` and `SP_WINSIZEY` the size of the window. If the buttons don't fit all into the window you won't see the last ones, there will be no error message.

Then come the definitions of the buttons. They are stored in the array `sp_buttons[]`. The elements are structures of the type `sp_but`. Take a look at `buttons.h`. Important is the last variable `var` it tells BUI how many bytes from the beginning of `control_struct` the variable to be modified by this button can be found. I calculate them with the little program `mkcindex.c` that writes `control_index.h`

9 Reporting bugs

If you find any bugs please send them with a detailed description of how you got to the bug to Jörn (behrens@ma.tum.de) or Bertram (bertram.lueckehe@gmx.de). The Software will be provided by Munich University of Technology, Center for Mathematical Sciences, Chair of Numerical Analysis and Scientific Computing. It can be used free of charge for scientific purposes only. Please visit the BJUGL Homepage for the most up to date release:

<http://www-m3.ma.tum.de/m3/software/bjugl/>.

A Copyright Notice

This software is provided for non-commercial use only. See the license conditions in file `LICENCE` and the warranty conditions in file `WARRANTY`.

Copyright ©2000 Jörn Behrens, Bertram Lückehe

B License

The use of BJUGL is hereby granted free of charge for an unlimited time, provided the following rules are accepted and applied:

1. You may use or modify this code for your own non commercial purposes.
2. The code may not be re-distributed without the consent of the authors.
3. The copyright notice and statement of authorship must appear in all copies.
4. You accept the warranty conditions (see file `WARRANTY`).
5. In case you intend to use the code commercially, we oblige you to sign an according licence agreement with the authors.

C Warranty

This code has been tested up to a certain level. Defects and weaknesses, which may be included in the code, do not establish any warranties by the authors.

The authors do not make any warranty, express or implied, or assume any liability or responsibility for the use, acquisition or application of this software.