

GDR Programmer Documentation

(last modification: 19 March 1992)

This document describes how to write an animated program for GDR. A program implementing a graph algorithm is written in plain C with macros and functions to access and animate the graph created by the user. The declarations needed for interfacing with GDR are provided by an `#include "gr.h"` statement. The code that is to be executed by the user is written as a parameterless procedure `animat()`. Examples of previously written animations can be found in files `bicon.c`, `dfa.c`, `dfs_d.c`, `nfa.c`, and `prim.c`.

The first step is to code the graph algorithm in C using graph traversal macros where appropriate. GDR provides the following traversal macros (the definitions for these are found in `def.h`):

- `for_adjacent(v,e,w)` creates a loop whose body is executed once for each outgoing edge e from vertex v — the vertex w is the other endpoint of e (for undirected graphs, all edges incident to v are traversed).
- `for_incoming_edges(v,e,u)` traverses all edges $e = (u,v)$ leading into v (in the case of undirected graphs, this is identical to `for_adjacent`).
- `for_all_vertices(v)` accesses all valid vertices v
- `for_all_edges(e)` accesses all edges e

In GDR a graph is a set of edges — multiple edges between the same two vertices are allowed. The above macros may not work correctly if the graph is modified in the loop body, e.g. if edges are deleted. More careful implementation using the functions `first_out_edge` and `next_out_edge` may be required. Tables 1 and 2 list all the GDR functions for accessing and changing graph structure.

Animations are created by changing the logical structure, geometry, or display characteristics of a graph in response to actions taken by the algorithm. Simple animations can be created by highlighting vertices and edges, by exposing or hiding their labels, or by changing the text of the labels. The functions to do this are described in Table 3 (the functions for changing labels are described in Table 2).

Table 4 shows other useful functions for interacting with the display of the graph. Vertices and edges can be blinked (the delay of a blink is controlled by a compile-time parameter `BLINK_DELAY` in `gr.h`). Windows to display text can be created and manipulated. Pauses for user response can be added using `suspend_animation` (a general pause mechanism), `get_xy_from_mouse` (to query user for a position), or `select_vertex` (to have user point to a vertex).

GDR also gives the programmer access to the geometry of the displayed graph — for details, see Table 5.

Suppose an animated program for directed graphs has been written in file `alg.c`. It can be compiled and linked with GDR using the following commands

```
cc -c -I/usr/include/X11 alg.c
cc -c -I/usr/include/X11 -DICON="Title Bar" -DDIRECTED gr.c
cc -o alg gr.o gr_tool.o pane.o list.o alg.o -lX11 -lm
```

This assumes that `gr_tool.c`, `pane.c`, and `list.c` have already been compiled (using an incantation like that of the first line above). The `-I/usr/include/X11` indicates the directory where X11 header files are to be found, `-DICON="Title Bar"` causes the given text to appear in the icon and the title bar of the GDR window (if this is omitted, the title bar just has the name GDR), and `-DDIRECTED` is omitted if the program deals with undirected graphs. The program `alg` then becomes a version of GDR customized to run the animation supplied by the programmer in `alg.c`.

Another alternative is to add an entry for `alg` in the `Makefile`, and then invoke the command `make alg`.

Further customizations can be done by editing the file `gr.h` (be sure to recompile everything when you do this).

[make a list of the sorts of things that can be customized] [explain blink delays and blink ratio]

Table 1: Access to Logical Graph Attributes in GDR

<code>max_vertex()</code>	returns the maximum vertex id possible (current implementation does not guarantee that it's valid)
<code>is_valid_vertex(vertex:v)</code>	returns TRUE if <code>v</code> is a valid vertex, FALSE otherwise (the vertex whose id is <code>v</code> may have been deleted)
<code>get_a_vertex()</code>	returns the first valid vertex in the graph
<code>are_edges_equal(edge:e1,e2)</code>	returns TRUE if the edges are the same, FALSE otherwise
<code>head(edge:e)</code>	returns the vertex to which <code>e</code> is directed (the second vertex to be added to <code>e</code> in case of an undirected edge)
<code>tail(edge:e)</code>	returns the vertex from which <code>e</code> is directed (the first vertex to be added to <code>e</code> in case of an undirected edge)
<code>other_vertex(vertex:v,edge:e)</code>	returns the vertex that along with <code>v</code> defines the edge <code>e</code>
<code>first_out_edge(vertex:v)</code>	returns the first outgoing edge from vertex <code>v</code> (first edge on <code>v</code> 's adjacency list if undirected graph)
<code>next_out_edge(vertex:v,edge:e)</code>	returns the next outgoing edge after <code>e</code> from vertex <code>v</code> (to be used in conjunction with <code>first_out_edge</code> for sequential access of adjacency lists)
<code>first_in_edge(vertex:v)</code>	returns the first edge into vertex <code>v</code> (for directed graphs; first edge on <code>v</code> 's adjacency list if undirected)
<code>next_in_edge(vertex:v,edge:e)</code>	returns the next edge after <code>e</code> going into vertex <code>v</code> (analogous to <code>next_out_edge</code>)
<code>first_edge()</code>	returns the first edge from the list of all edges in the graph
<code>next_edge(edge:e)</code>	returns the next edge after <code>e</code> in the list of all edges
<code>vertex_label(vertex:v)</code>	returns the label of vertex <code>v</code> in an allocated string
<code>edge_label(edge:e)</code>	returns the label of edge <code>e</code> in an allocated string

Table 2: Changing Logical Graph Attributes in GDR

<code>add_vertex(int:x,y;string:label)</code>	adds a new vertex and returns its id; the vertex is drawn at position <code>(x,y)</code> and is given <code>label</code> as its label
<code>add_edge(vertex:v1,v2;string:label)</code>	adds a new edge from <code>v1</code> to <code>v2</code> and returns its id; label of the new edge is <code>label</code>
<code>delete_vertex(vertex:v)</code>	deletes vertex <code>v</code> and all incident edges from the graph
<code>delete_edge(edge:e)</code>	deletes edge <code>e</code> from the graph
<code>change_vertex_label(vertex:v;string:label)</code>	changes the label of <code>v</code> to <code>label</code>
<code>change_edge_label(edge:e;string:label)</code>	changes the label of <code>e</code> to <code>label</code>

Table 3: Access to Display Attributes in GDR

<code>is_highlighted_vertex(vertex:v)</code>	returns TRUE if the vertex <code>v</code> is highlighted
<code>is_highlighted_edge(edge:e)</code>	returns TRUE if the edge <code>e</code> is highlighted
<code>is_exposed_vertex_label(vertex:v)</code>	returns TRUE if the label of vertex <code>v</code> is exposed
<code>is_exposed_edge_label(edge:e)</code>	returns TRUE if the label of edge <code>e</code> is exposed
<code>highlight_vertex(vertex:v)</code>	highlights vertex <code>v</code> , i.e. makes its background white
<code>un_highlight_vertex(vertex:v)</code>	removes highlighting from vertex <code>v</code> , i.e. makes its background black
<code>highlight_edge(edge:e)</code>	highlights edge <code>e</code> , i.e. fattens all of the line segments representing <code>e</code>
<code>un_highlight_edge(edge:e)</code>	removes highlighting from edge <code>e</code> , i.e. represents the edge as line segments of thickness 1
<code>expose_vertex_label(vertex:v)</code>	causes the vertex label of <code>v</code> to appear on the display
<code>expose_edge_label(edge:e)</code>	causes the edge label of <code>e</code> to appear on the display
<code>hide_vertex_label(vertex:v)</code>	causes the vertex label of <code>v</code> to be erased from the display
<code>hide_edge_label(edge:e)</code>	causes the edge label of <code>e</code> to be erased from the display

Table 4: Miscellaneous Programmer Functions in GDR

<code>blink_vertex(vertex:v,int:count)</code>	causes vertex <code>v</code> to blink (change from unhighlighted to highlighted and back) <code>count</code> times
<code>blink_edge(edge:e,int:delay,count)</code>	causes edge <code>e</code> to blink (details same as vertex blinking)
<code>print_graph_data(string:name;flag:append)</code>	outputs the current graph, in GDR format, to file <code>name</code> ; if <code>append</code> is <code>TRUE</code> , the existing file gets appended, else it is overwritten (can be used for automatic generation of stills from animation)
<code>suspend_animation()</code>	gives control back to edit mode; program execution is resumed when user clicks left in the <code>RESUME ?</code> window or presses <code>q</code> or <code>Q</code>
<code>create_text_window(int:x,y;string:message)</code>	creates a window with upper left corner at position <code>(x,y)</code> and <code>message</code> written into it (<code>message</code> can have multiple lines, delimited by <code>\n</code>), and returns a window identifier (type <code>Window</code>)
<code>write_text_window(Window:w,string:message)</code>	changes the content of window <code>w</code> to <code>message</code> and causes window <code>w</code> to appear (if hidden)
<code>hide_window(Window:w)</code>	makes window <code>w</code> disappear from view
<code>kill_window(Window:w)</code>	destroys the window <code>w</code> and unmaps it in the process; warning: any further references to the window identifier <code>w</code> will result in an X Window Protocol error.
<code>get_xy_from_mouse(int reference:x,y)</code>	prompts user to click the mouse and the <code>x</code> and <code>y</code> coordinates are passed back by reference (returns <code>FALSE</code> if user hits <code>[Control-C]</code> to abort, else returns <code>TRUE</code>)
<code>select_vertex()</code>	returns the id of a vertex selected by left mouse click or <code>NULL_VERTEX</code> if user typed <code>q</code> before clicking the mouse
<code>query(int: x,y; string:msg,ans)</code>	prints <code>msg</code> in a box at position <code>(x,y)</code> and returns what the user typed through <code>ans</code> ; the actual value returned is a flag indicating a response other than <code>C</code>

Table 5: Access to Geometric Attributes in GDR

<code>add_vertex(int:x,y;string:label)</code>	adds a new vertex and returns its id; the vertex is drawn at position (x,y) and is given label as its label
<code>vertex_x(vertex:v)</code>	returns the x-coordinate of the vertex v
<code>vertex_y(vertex:v)</code>	returns the y-coordinate of the vertex v
<code>move_vertex_relative(vertex:v;int:d_x,d_y)</code>	moves the vertex v to the right d_x units and down d_y units (left and/or up if negative numbers are used); also moves the associated edges; returns FALSE on error, else returns TRUE
<code>edge_label_x(edge:e)</code>	returns the x-coordinate of the label of edge e
<code>edge_label_y(edge:e)</code>	returns the y-coordinate of the label of edge e
<code>move_edge_label(edge:e;int:x,y)</code>	moves the edge label of e so that it is centered at point (x,y)
<code>window_width()</code>	returns the width of the graph display area
<code>window_height()</code>	returns the height of the graph display area