NAME

VCG tool – visualization of compiler graphs

SYNOPSIS

vcg [options] [filename]
xvcg [options] [filename]

SYNOPSIS FOR SEQUENCES OF GRAPH SPECIFICATIONS

vcg –multi [options] [filename] [options] [filename] ...
xvcg –multi [options] [filename] [options] [filename] ...

DESCRIPTION

The *VCG tool* reads a VCG specification and visualizes the graph. If not all positions of nodes are fixed, the tool lays the graph out using several heuristics as reducing the number of crossings, minimizing the size of edges, centering of nodes. The specification language of the *VCG tool* is nearly compatible to GRL, the language of the *Edge tool,* but contains many extensions. The *VCG tool* allows folding of dynamically or statically specified regions of the graph. It uses colors and runs on *X11.* (An older version runs on *Sunview.)*

GENERAL OPTIONS

– The input is *stdin* instead of a file.

–h | –? Print a help message about the usage of the tool.

–v | –version Print a version and copyright message.

–silent Be silent during the layout. No messages or warnings are produced.

–nocolors | –blackwhite
Do not use colors on a color screen. On a monochrom screen the graph is drawn in black & white even if it is specified with colors. On a color screen, this mode is simulated, if this option is selected. The option is useful, if the *VCG tool* conflicts with other programs that need colors.

–e <num> | –error <num>
Stop after <*num*> errors during parsing of the specification (default: 16).

–a <num> | –animation <num>
Start the tool as animation handler. This implies that the tool is controlled by signals *(USRSIG1, USRSIG2)* from an external program. The signal *USRSIG1* causes the *VCG tool* to open the display window and reload its input file. The signal *USRSIG2* causes the *VCG tool* to close the display window. The tool processes the input and indicates the completion of the processing to the controlling program. If <*num*> is greater 0, this indication is done by sleeping <*num*> seconds and touching the input file afterwards such that its time stamp is refreshed. If <*num*> is less than 0, this indication

is done by sleeping – <*num*> seconds and sending signal *USRSIG1* to the caller process afterwards.

–m | **–multi**
Read multiple files one after another to display a sequence of graphs. Instead of an exit of the tool, the next file is read. The option **–multi** must be specified before the first input file.

–bary
Use bary centering to reduce the number of edge crossings (default).

–median
Use median centering instead of bary centering to reduce the number of crossings. On graphs with large average degree of edges, bary centering might be faster.

–barymedian
Use a hybrid method to reduce the number of edge crossings. Bary centering is used for all nodes with different bary values. For nodes with same bary values, the crossing reduction heuristics normally has a random effect. With this hybrid method, the crossing reduction of nodes with same bary values is done by median centering.

–medianbary
Use a hybrid method to reduce the number of edge crossings. Median centering is used for all nodes with different median values. For nodes with same median values, the crossing reduction heuristics normally has a random effect. With this hybrid method, the crossing reduction of nodes with same median values is done by bary centering.

–notune | **–nofinetune**
Switch the fine tuning layout phase off. Fine tuning is a postprocessing phase after partitioning of nodes into layers. Fine tuning changes the layers of nodes to minimize the length of edges. If this phase is switched off, it yield a less compact distribution of nodes in the levels.

–manhattan | **–orthogonal**
Switch the orthogonal manhattan layout on. This forces edges of gradient 0 or 90 degree. The result of this layout might be more aesthetical, if additionally the priority layout phase with straight line tuning is used. Thus, these phases are switched on, too, unless priority layout and straight line tuning is switched explicitly off.

–nomanhattan | **–noorthogonal**
Switch the orthogonal manhattan layout explicitly off.

–smanhattan
Use the orthogonal manhattan layout without separation of horizontal line segments. Horizontal line segments are shared between different edges. This looks nice for trees but might be confusing in general, because the location of an edge might be ambiguously.

–nosmanhattan
Switch the orthogonal manhattan layout without separation explicitly off.

–prio
Switch the priority layout on. This forces straight long edges with gradient 90 degree during the node placement phase. The straight line tuning phase can be used to improve the priority layout. Thus, this phase is switched on,

too, unless straight line tuning is switched explicitly off.

–noprio Switch the priority layout explicitly off.

–straight Switch the straight line tuning phase on. This phase forces straight long edges with gradient 90 degree. It can be used to improve the priority layout.

–nostraight Switch the straight line tuning phase explicitly off.

–nonearedge Do not allow near edges in the layout.

–l | –latelabels Create the labels after the partitioning of edges. This has only an effect if labels are shown in a nondirty way. If labels are created after the partitioning of edges, the layout will be a little bit wider and may have less crossings. But note that sometimes this layout may also be worser than the normal layout.

–hidesingles | –ignoresingles
Ignore single nodes (nodes without visible edges) that are not connected with the remaining graph. These nodes are sometimes not of interest, but would spread the layout if they were visible.

–s | –summarize Switch edge summary layout on. Multiple edges between the same source and target node are summarized if they have the same visible appearance. This reduces the number of visible edges.

OPTIONS FOR THE LAYOUT SPEED

The speed of the layout process can be influenced by selecting time limits and iteration bounds of the different layout phases. Optional layout phases can completely be skipped. But note that we need a minimal time for each graph, in order to initialize the internal data structures. Furthermore, a fast layout is probably also an ugly layout.

–t <num> | –timelimit <num>
Set the time limit to <*num*> seconds of real time. If the time limit is exceeded, the fastest possible layout mode is switched on. This may yield a very ugly layout. Of course, a time limit does not mean that the layout really needs so much time: The layout may be faster, because the graph structure is very simple, or it may be slower, because even the fastest possible layout already exceeds the time limit. Further note, that the layout depends on the real time, i.e. on the load of the computer. Thus, given a time limit, two identical trys need not to give identical results.

–f | –fast Switch the fast and dirty and ugly mode on. The layout phase will be very fast, but the layout will be ugly. This is helpful on very large graphs where aesthetic visibility is of minor importance. The option **–fast** implies **–bmax 2 –cmax 2 –pmax 2**

–b <num> | –bmax <num> | –bending <num>
Set the maximal number of iterations used for the reduction of edge bendings to <*num*>. Edge bendings are used to avoid that edges are drawn across

nodes. Reducing the number of iterations results in a faster but ugly layout, i.e. to much bendings occur. The default value is 100.

–c <num> | –cmax <num> | –crossing <num>
Set the maximal number of iterations used for the reduction of edge crossings to <*num*>. The edge crossing reduction method is called bary centering or median centering. These may be very time consuming on large graphs. Reducing the number of iterations results in a faster but ugly layout. As default, the method is iterated as long as improvements are possible.

–cmin <num> Set the minimal number of iterations used for the reduction of edge crossings to <*num*>. The crossing reduction method tries to detect improvements of the layout. If an iteration of that method does not yield an improvment, the method normally stops. But this situation might be a local minimum of the quality of the layout, i.e. further iterations might find a better layout. Thus, the minimal number of iterations can be specified. The default value is 0.

–p <num> | –pmax <num> | –pendulum <num>
Set the maximal number of iterations used for the balancing by the pendulum method to <*num*>. The pendulum method calculates the x co-ordinates of nodes such that the layout is medium dense and balanced. It tries to avoid extreme gradients of edges. It works like a pendulum where the nodes are the balls, the edges are the strings and uppermost nodes are fixed on the ceiling. Reducing the number of iterations results in a faster but ugly layout. The default value is 100.

–pmin <num> Set the minimal number of iterations used for the balancing by the pendulum method to <*num*>. If an iteration of that method does not yield an improvment, the method normally stops. But this situation might be a local minimum of the quality of the layout, i.e. further iterations might find a better layout. Thus, the minimal number of iterations can be specified. The default value is 0.

–r <num> | –rmax <num> | –rubberband <num>
Set the maximal number of iterations used for the balancing by the rubberband method to <*num*>. The rubberband method calculates the x co-ordinates of nodes such that the nodes are centered relatively to their incoming and outgoing edges. It works like a network where the edges pull on the nodes like rubberbands. Reducing the number of iterations results in a faster but ugly layout. The default value is 100.

–rmin <num> Set the minimal number of iterations used for the balancing by the rubberband method to <*num*>. If an iteration of that method does not yield an improvment, the method normally stops. But this situation might be a local minimum of the quality of the layout, i.e. further iterations might find a better layout. Thus, the minimal number of iterations can be specified. The default value is 0.

–smax <num> Set the maximal number of iterations used for the straight line tuning phase to <*num*>. This phase forces straight long edges with gradient 90 degree. It can be used to improve the priority layout or the manhattan layout.

–nocopt | –nocopt2
Skip the optimization phase 2 during the crossing reduction. This phase takes very long time on very large graphs. Before reducing the number of iterations of the crossing reduction phase (see option **–cmax <num>**) you should first try to skip this optimization phase 2.

–nocoptl | –nocoptlocal
Switch a local crossing optimization off. This phase additionally examines pairs of edge polygons and tries to unwind them. It slows down if the degree of the nodes is large.

OPTIONS FOR THE DISTRIBUTION OF NODES

The quality of the layout is mainly influenced by the distribution of the nodes into the hierarchy levels. Nodes of the same level will appear in the same row. Since it depends on the application which hierarchy is the best, there are different algorithms for this phase.

–d normal Normal distribution of nodes into the levels (default). This algorithm is based on the calculation of the strongly connected components.

–d dfs Depth first search distribution of nodes into the levels. This is the fastest method.

–d 0 | –d minbackward
Reduce the number of backward edges heuristically. If the graph is acyclic, no backward edges will occur, i.e. all edges point into the same direction.

–d + | –d maxdepth
Maximize the depth of the layout heuristically. It should be used if the layout is too wide in x direction. This algorithm is very fast.

–d – | –d mindepth
Minimize the depth of the layout heuristically. It should be used if the layout is too wide in y direction. This algorithm is very fast.

–d ++ | –d maxdepthslow
Maximize the depth of the layout heuristically. See above. This algorithm is very slow, but may give better results.

–d — | –d mindepthslow
Minimize the depth of the layout heuristically. See above. This algorithm is very slow, but may give better results.

–d minindeg | –d minindegree
Prepare the nodes by sorting them according increasing indegree (number of incoming edges). The nodes with the minimal indegree come first. This may have various effects on the layout.

–d maxindeg | –d maxindegree
Prepare the nodes by sorting them according decreasing indegree. The nodes with the maximal indegree come first. This may have various effects on the layout.

–d minoutdeg | –d minoutdegree
Prepare the nodes by sorting them according increasing outdegree (number of outgoing edges). The nodes with the minimal outdegree come first. This may have various effects on the layout.

–d maxoutdeg | –d maxoutdegree
Prepare the nodes by sorting them according decreasing outdegree. The nodes with the maximal outdegree come first. This may have various effects on the layout.

–d mindeg | –d mindegree
Prepare the nodes by sorting them according increasing degree (number of incoming and outgoing edges). The nodes with the minimal degree come first. This may have various effects on the layout.

–d maxdeg | –D maxdeg
Prepare the nodes by sorting them according decreasing degree. The nodes with the maximal degree come first. This may have various effects on the layout.

–d tree Use a specialized layout for trees. It does not work with non-trees.

OPTIONS FOR THE VIEW

The view of a graph is the method of the visual appearance of the graph in the window after the layout. Changing the view does not require a relayout of the graph. Views are transformations that are done during the drawing.

–view normal Normal view onto the graph. No distortion.

–view cfish Cartesian fisheye view. The graph is distorted such that the whole graph is visible. Horizontal and vertical lines don't change their direction.

–view fcfish Cartesian fisheye view with fixed size focus. The graph is distorted, but not necessarily the whole graph is visible.

–view pfish Polar fisheye view. The graph is distorted such that the whole graph is visible. Even horizontal and vertical lines might be distorted.

–view fpfish Polar fisheye view with fixed size focus. The graph is distorted, but not necessarily the whole graph is visible.

–spline Use splines instead of polygons to draw edges. This is mainly useful if you want to export the graph into a high quality PostScript picture. WARNING: Drawing splines is very slow.

–nonodes Suppress drawing of nodes.

–noedges Suppress drawing of edges.

–xpos <num> Set the x-coordinate of the initial point of the graph that appears at the window origin or of the initial focus point to <*num*>.

–ypos <num> Set the y-coordinate of the initial point of the graph that appears at the window origin or of the initial focus point to <*num*>.

PRINTER DRIVER INTERFACE

The printer driver interface allows to produce an output file of the visualized graph without the need of interaction. The VCG tool acts as a kind of converter program in this case: it converts a VCG file into a PostScript or bitmap file. It is recommended to use the option **–silent** in combination to one of the options **–vcgoutput, –psoutput, –pbmoutput,** or **–ppmoutput.** Example:

```
xvcg –silent –color –scale 75 –psoutput test.ps test.vcg
```

converts the VCG file test.vcg into a PostScript file that contains a color picture of the graph scaled by 75 %. In this case the interactive display is suppressed. If the graph does not fit on the page, the output might be nonsense.

PRINTER DRIVER OPTIONS

–vcgoutput <filename>
Produce a VCG file named <*filename*> that contains the graph laid out, i.e. including information about the co-ordinates of the visible nodes. The most of the following format options have no effect for the VCG file format.

–psoutput <filename>
Produce a PostScript file named <*filename*> that contains the graph.

–pbmoutput <filename>
Produce a bitmap file named <*filename*> in PBM format that contains the graph in black and white.

–ppmoutput <filename>
Produce a bitmap file named <*filename*> in PPM format that contains the graph in colors.

–paper <papertype>
Select the paper type. Default is a4. <*papertype*> is one of:

a4	din A4 (21 x 30 cm)
A4	din A4 (21 x 30 cm)
b5	din B5 (18.5 x 27 cm)
B5	din B5 (18.5 x 27 cm)
a5	din A5 (15 x 21 cm)
A5	din A5 (15 x 21 cm)

11x17	tabloid (11 x 17 in)
tabloid	tabloid (11 x 17 in)
8x11	letter (8.5 x 11 in)
letter	letter (8.5 x 11 in)
8x14	legal (8.5 x 14 in)
legal	legal (8.5 x 14 in)

–noBoundingBox
Suppress the calculation of a BoundingBox (PostScript format).

–color Select a color file output. This option works only with the PostScript format.

–grey Select a greyscaled file output. This option works only with the PostScript format.

–blackwhite Select a monochromatic file output. This is the default color. This option works only with the PostScript format.

–portrait Select the paper orientation ‘Portrait’ (default).

–landscape Select the paper orientation ‘Landscape’.

–split <num> Split the graph into <*num*> pages. This option works only with the PostScript format. The number of pages must be one of 1, 4, 9, 16, or 25.

–xdpi <num> Set the horizontal resolution for the PPM and PBM format. This allows to adapt the bitmap formats to the resolutions of the printer.

–ydpi <num> Set the vertical resolution for the PPM and PBM format. This allows to adapt the bitmap formats to the resolutions of the printer.

–scale <num> Scale the graph to <*num*> percent for the file output. The default scaling fits the graph with maximal aspect ratio to the paper size.

–width <float> <units>
Fit the graph such that its width is smaller than <*float*> <*units*>. This works only if no scaling is specified. <*units*> are:

in	Inches
inch	Inches
ft	Foot
foot	Foot
feet	Foot
mm	Millimeter
cm	Centimeter
dm	Decimeter
m	Meter

–height <float> <units>
Fit the graph such that its height is smaller than <*float*> <*units*>. This works only if no scaling is specified.

–lm <float> <units>
Set the left margin of the output to <*float*> <*units*>. This works only if no right margin is specified. The default position is centered on the page. In some cases the *BoundingBox* of the PostScript output meight be wrong. If a *BoundingBox* is needed then we recommend the options **–lm 0 cm –bm 0 cm.**

–rm <float> <units>
Set the right margin of the output to <*float*> <*units*>. This works only if no left margin is specified. The default position is centered on the page.

–tm <float> <units>
Set the top margin of the output to <*float*> <*units*>. This works only if no bottom margin is specified. The default position is centered on the page.

–bm <float> <units>
Set the bottom margin of the output to <*float*> <*units*>. This works only if no top margin is specified. The default position is centered on the page.

X11 OPTIONS

–display <host:dpy>
Set the remote X11 server to *host:dpy*. This is analogous to *xterm(1l).*

–geometry <geom>
Specify the hint of size and location of the X11 window. This is analogous to *xterm(1l).*

–bw <num> Set the border width of the X11 window to <*num*> pixels. This is analogous to *xterm(1l).*

–font <xfont> Set the font used for messages and menu items in the X11 window. This is analogous to *xterm(1l).*

–grabinputfocus Switch setting of InputFocus on or off (depending on the default). Cause the VCG tool to execute a XSetInputFocus, or to avoid to execute a XSetInputFocus on initialization.

GRAMMAR

The grammar of the specification language is the following:

graph
```
::=  "graph:" '{' graph_entry_list '}'
;
```

graph_entry_list
```
::=  graph_entry_list graph_entry
|  graph_entry
;
```

graph_entry
```
::=  graph_attribute
| node_defaults
| edge_defaults
| foldnode_defaults
| foldedge_defaults
| graph
| node
| edge
| nearedge
| bentnearedge
| backedge
;
```

graph_attribute
```
::=  "title" ':' string
| "label" ':' string
| "info1" ':' string
| "info2" ':' string
| "info3" ':' string
| "color" ':' enum_color
| "textcolor" ':'enum_color
| "bordercolor" ':'enum_color
| "width" ':' integer
| "height" ':' integer
| "borderwidth" ':' integer
| 'x' ':' integer
| 'y' ':' integer
| "loc:" '{' 'x' ':' integer 'y' ':' integer '}'
| "folding" ':' integer
| "scaling" ':' float
| "shrink" ':' integer
| "stretch" ':' integer
| "textmode" ':' enum_textmode
| "shape" ':' enum_shape
| "level" ':' integer
| "vertical_order" ':' integer
| "horizontal_order" ':' integer
| "status" ':' enum_status
| "xmax" ':' integer
| "ymax" ':' integer
| "xbase" ':' integer
| "ybase" ':' integer
| "xspace" ':' integer
```

```
| "xlspace" ':' integer
| "yspace" ':' integer
| "xraster" ':' integer
| "xlraster" ':' integer
| "yraster" ':' integer
| "invisible" ':' integer
| "hidden" ':' integer
| "classname" integer ':' string
| "infoname" integer ':' string
| "colorentry" integer ':' integer integer integer
| "layoutalgorithm" ':' enum_layoutalgorithm
| "layout_downfactor" ':' integer
| "layout_upfactor" ':' integer
| "layout_nearfactor" ':' integer
| "splinefactor" ':' integer
| "late_edge_labels" ':' enum_yes_no
| "display_edge_labels" ':' enum_yes_no
| "dirty_edge_labels" ':' enum_yes_no
| "finetuning" ':' enum_yes_no
| "ignoresingles"  ':' enum_yes_no
| "straight_phase" ':'  enum_yes_no
| "priority_phase" ':'  enum_yes_no
| "manhattan_edges" ':'  enum_yes_no
| "smanhattan_edges" ':'  enum_yes_no
| "nearedges" ':' enum_yes_no
| "orientation" ':' enum_orientation
| "node_alignment" ':' enum_node_align
| "port_sharing" ':' enum_yes_no
| "arrowmode" ':' enum_arrow_mode
| "spreadlevel" ':' integer
| "treefactor" ':' float
| "crossingphase2" ':' enum_yes_no
| "crossingoptimization" ':' enum_yes_no
| "crossingweight" ':' enum_cross_weight
| "view" ':'  enum_view
| "edges" ':' enum_yes_no
| "nodes" ':' enum_yes_no
| "splines" ':' enum_yes_no
| "bmax" ':' integer
| "cmax" ':' integer
| "cmin" ':' integer
| "pmax" ':' integer
| "pmin" ':' integer
| "rmax" ':' integer
```

```
        | "rmin" ':' integer
        | "smax" ':' integer
        ;
enum_color
        ::=  "aquamarine"
        | "black"
        | "blue"
        | "cyan"
        | "darkblue"
        | "darkcyan"
        | "darkgreen"
        | "darkgrey"
        | "darkmagenta"
        | "darkred"
        | "darkyellow"
        | "gold"
        | "green"
        | "khaki"
        | "lightblue"
        | "lightcyan"
        | "lightgreen"
        | "lightgrey"
        | "lightmagenta"
        | "lightred"
        | "lightyellow"
        | "lilac"
        | "magenta"
        | "orange"
        | "orchid"
        | "pink"
        | "purple"
        | "red"
        | "turquoise"
        | "white"
        | "yellow"
        | "yellowgreen"
        | integer
        ;
enum_orientation
        ::=  "top_to_bottom"
        | "bottom_to_top"
        | "left_to_right"
        | "right_to_left"
```

```
        ;
enum_layoutalgorithm
        ::=
        | "tree"
        | "maxdepth"
        | "mindepth"
        | "maxdepthslow"
        | "mindepthslow"
        | "maxdegree"
        | "mindegree"
        | "maxindegree"
        | "minindegree"
        | "maxoutdegree"
        | "minoutdegree"
        | "minbackward"
        | "dfs"
        ;
enum_status
        ::=   "black"
        | "grey"
        | "white"
        ;
enum_yes_no
        ::=   "yes"
        | "no"
        ;
enum_cross_weight
        ::=   "bary"
        | "median"
        | "barymedian"
        | "medianbary"
        ;
enum_view
        ::=   "cfish"
        | "fcfish"
        | "pfish"
        | "fpfish"
        ;
enum_arrow_mode
        ::= "fixed"
        | "free"
```

```
        ;
foldnode_defaults
        ::=  "foldnode." node_attribute
        ;
foldedge_defaults
        ::=  "foldedge." edge_attribute
        ;
node_defaults
        ::=  "node." node_attribute
        ;
edge_defaults
        ::=  "edge." edge_attribute
        ;
node    ::=  "node:" '{' node_attribute_list '}'
        ;
node_attribute_list
        ::=  node_attribute_list node_attribute
        | node_attribute
        ;
edge    ::=  "edge:" '{' edge_attribute_list '}'
        ;
nearedge
        ::=  "nearedge:" '{' edge_attribute_list '}'
        ;
bentnearedge
        ::=  "bentnearedge:" '{' edge_attribute_list '}'
        ;
backedge
        ::=  "backedge:" '{' edge_attribute_list '}'
        ;
edge_attribute_list
        ::=  edge_attribute_list edge_attribute
        | edge_attribute
        ;
node_attribute
        ::=  "title" ':' string
        | "label" ':' string
        | "info1" ':' string
        | "info2" ':' string
```

```
          | "info3" ':' string
          | "color" ':' enum_color
          | "textcolor" ':'enum_color
          | "bordercolor" ':'enum_color
          | "width" ':' integer
          | "height" ':' integer
          | "borderwidth" ':' integer
          | "loc:" '{' 'x' ':' integer 'y' ':' integer '}'
          | "folding" ':' integer
          | "scaling" ':' float
          | "shrink" ':' integer
          | "stretch" ':' integer
          | "textmode" ':' enum_textmode
          | "shape" ':' enum_shape
          | "level" ':' integer
          | "vertical_order" ':' integer
          | "horizontal_order" ':' integer
          ;
```

enum_textmode

```
          ::=  "center"
          | "left_justify"
          | "right_justify"
          ;
```

enum_shape

```
          ::=  "box"
          | "rhomb"
          | "ellipse"
          | "triangle"
          ;
```

enum_node_align

```
          ::=  "bottom"
          | "top"
          | "center"
          ;
```

edge_attribute

```
          ::=  "sourcename" ':' string
          | "targetname" ':' string
          | "label" ':' string
          | "textcolor" ':'enum_color
          | "color" ':' enum_color
          | "thickness" ':' integer
          | "class" ':' integer
```

```
        | "priority" ':' integer
        | "arrowcolor" ':' enum_color
        | "backarrowcolor" ':' enum_color
        | "arrowsize" ':' integer
        | "backarrowsize" ':' integer
        | "arrowstyle" ':' enum_arrowstyle
        | "backarrowstyle" ':' enum_arrowstyle
        | "linestyle" ':' enum_linestyle
        | "anchor" ':' integer
        | "horizontal_order" ':' integer
        ;
```

enum_linestyle

```
        ::=  "continuous"
        | "solid"
        | "dotted"
        | "dashed"
        | "invisible"
        ;
```

enum_arrowstyle

```
        ::=  none
        | line
        | solid
        ;
```

WARNINGS

The *VCG tool* needs about 200 bytes per edge and node. Depending on the layout, it will produce a lot of additional dummy nodes and dummy edges, such that it may run out of memory. The layout algorithm needs exponentially time in the worst case.

ACKNOWLEDGEMENTS

The *Edge tool* was developed at the University of Karlsruhe. GRL was described by S. Manke and F.N. Paulisch.
Our colleagues M. Alt and C. Ferdinand found the most bugs and gave many proposals for improvements.
The Institute of Compiler Construction at the University of Saarland, Germany, and the COMPARE Consortium were the first and most important users of the *VCG tool* and gave us the motivation, the time and many hints during the development of the tool.

SEE ALSO

Sunview(1) X11(1l) edge(l)
vcgdemomaker(l) pbmshift(l) pbmrot90(l) pbm2hp(l)
VCG – Visualization of Compiler Graphs, Design Report and User Documentation, Ref.

Compare, USAAR-1049-visual, January 1994, updated January 1995

BUGS

The X11 version has the ‘InputFocus’ problem. This problem is solved for 99 % of all cases, but may still occur.
If a graph is written to a file and reload from this file, the layout may be different and may be ugly.
The attribute horizontal_order does only works for connected graphs, but not for unconnected graphs.
For some parameters, negative values are inappropriate even if the tool does not crashes in such situations. However the result will be very ugly. For instance, the values of *xbase* and *ybase* should always be greater than zero.
The spline routine is still too slow and has some bugs when exporting to a multi page PostScript file.
Currently, no further bugs are known.

AUTHORS

Georg Sander, University of Saarland, Germany.
Iris Lemke, University of Saarland, Germany.