iceWing
User and Programming Guide

Program: Frank Lömker
Documentation: Frank Lömker, Andreas Hüwel

19th January 2005

Contents

1 Short overview

What's all about?

iceWing, an **I**ntegrated **C**ommunication **E**nvironment **W**hich **I**s **N**ot **G**esten (This is a reference to an older program, the predecessor of iceWing.) is a graphical plugin shell. It can be used for single image analysis, but is especially useful for realtime video-stream processing. Predefined or self-written plugins operate hierarchically on the pre-given data and can also generate new data-streams. iceWing provides mutual communication of this plugins. There is e.g. the predefined plugin for reading images from grabber-hardware with the help of the AVlib and also from external, network wide processes via DACS streams. Even writing of own e.g. audio-stream processing plugins is realizable.

Not being only a batch-plugin-shell iceWing is also a highly customizable GUI platform for the plugins: It has a list of given GUI-elements and allows the plugins to simply make use of them. So plugins can show the user their current status and can let the user change parameters on the fly. Moreover methods of easy visualization of plugin results are available. The plugins can open any number of windows and display in these windows any data in a graphical form.

Where to get iceWing?

floemker@techfak.uni-bielefeld.de

Who did iceWing?

Program: Frank Lömker
floemker@techfak.uni-bielefeld.de
This documentation: Frank Lömker, floemker@techfak.uni-bielefeld.de
Andreas Hüwel, andreas.huewel@gmx.de

License

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

2 Installation

2.1 Requirements

Libraries needed iceWing is targeted at Unix-like operating systems and was actually tested on Linux and Alpha/True64. For compiling and using it some programs and libraries must be installed:

- Basic commands like bunzip2, tar, make, makedepend, c-compiler (probably gcc) - nothing that a normal Unix installation does not provide.
- X11
- gtk with header files, tested for version >=1.2.5, <2.0
- gdk-pixbuf with headers – optional, needed to support loading additional image-formats, tested for version >= 0.8
- libjpeg – optional, needed to support jpeg- and mjpeg-saving
- libpng – optional, needed to support png-saving and loading of images with a depth of 16 bit
- libz – optional, used by the above optional png saving package.

We will show for RedHat packages (rpm) how to verify this. Debian packages are quite similar. You can easily check with the following shell command

```
> rpm -qa |grep gtk
```

which packages are installed and which version that libraries have. iceWing needs the headers, too, and the given developer packets provide them. If you compile your own libraries from sources, you have to add the headers into the default-include path, so iceWing will find them. After verifying all this, you only need to get the iceWing tarball "icewing-version.tar.bz2".

Further stuff You can use DACS for your own iceWing plugins to communicate with other external net wide processes. It works quite similar to Corba. You do *not* need DACS for any iceWing internal communication (iceWing - plugin, plugin - plugin) or to e.g. access files of the system. Additionally, iceWing has integrated support for reading/publishing images to/from other programs and for remote control of plugin sliders via DACS. If DACS is not available, these features can be disabled during compiling.

More information about DACS can be found in the Web under this address:

http://www.techfak.uni-bielefeld.de/ags/ni/projects/dacs/

Many details of DACS are also in the dissertation of Niels Jungclaus [Jun98].

2.2 Installation

Lets assume you have the tarball in "./". Then simply

```
> bunzip2 -c icewing-version.tar.bz2 | tar -xv
> cd icewing-version
```

where version is the particular iceWing version number you are using.

Adopt the Makefile to your system Now you have to edit the Makefile to adopt it to your system and in- or exclude the support for additional packages. Mostly it will be uncommenting some few lines that you probably will not need on your system.

PREFIX: It must be set to your installation place, for example "/usr/local"

FLAGS: Eventually you must adopt the compiler flags to your hardware (E.g.: You might not have a Pentium/Athlon processor?)

WITH_AV: If included, iceWing will have support for grabbing images from a camera. On Alpha/True64 systems an external library – the AVlib – is needed. This library is written by the AG-AI and supports grabbing of images via the Multi Media Extension MME. Composit and SVideo cameras are supported. The value given to WITH_AV specifies the location of the AVlib.

On Linux support for image grabbing is directly integrated into iceWing. This gets activated if WITH_AV is defined. Composite and SVideo cameras are supported with the help of the "Video for Linux Two" interface (see "http://linux.bytesex.org/v4l2/"). If additionally WITH_FIRE is defined FireWire (IEEE1394) cameras are supported, too. Here, the external libraries libraw1394.a and libdc1394_control.a are needed (see "http://www.linux1394.org/" and "http://sourceforge.net/projects/libdc1394/"). WITH_FIRE gives the location of these two libraries.

WITH_TRAJECT: Adds a library, that is used by the plugin "tracking". It enhances the plugins capabilities: Tracking then is able to send a special datatype via DACS. You probably want to disable this.

WITH_DACS: If included, iceWing will have support for DACS, which is like Corba for network wide interprocess communication. This additionally adds libsfb to iceWing, which is used to encode different data structures.

WITH_GPB: Enables loading of images of other formats than "pnm". For this, the gdk-pixbuf library is used.

WITH_JPG, WITH_PNG, WITH_ZLIB, WITH_AVI: Enables further image saving formats. When all are commented, iceWing can only save the various "pnm" formats. Moreover, iceWing will not be able to load 16 bit png images.

"make" it all After adopting the Makefile, you can build the installation files from the sources:

```
> cd {wherever your sources are}
> make depend
> make
```

You must now login as admin/root, if $(PREFIX) is not writable for the current user. The Makefile expects the directory that is named in $(PREFIX) to be existing - if not:

```
> mkdir $(PREFIX)
```

The installation is now fully prepared. Now the time for installation has come!

```
> cd {wherever your sources are}
> make install
```

To all the cautious admins: You eventually want to check the groups and rights of the new dirs now.

That's it! For further details about what was done, just have insight into the installation log file, which is located at "$(PREFIX)/share/log/iceWing.log".

If the new "icewing"-executable is in the execute-path, you can right away start "icewing", the executable (see section 3.1 Quicktour). If you are interested in what is where in this installation, have a look at section 6.1 Filesystem.

Troubleshooting Check the output for errors, also the installation-logfile. You are sure, that you installed the needed libraries properly, but

```
> make
```

produced errors like:

/bin/sh: gdk-pixbuf-config: command not found

Let's again assume you used the RedHat package named "gdk-pixbuf-devel" (well, Debian packages are treated similar). Then check via

```
> rpm -ql gdk-pixbuf-devel |grep gdk-pixbuf-config
```

or with *full* search

```
> find / -name gdk-pixbuf-config
```

where the needed packed-files got installed. Maybe they are simply not in default-execute path!?

Assume you find "gdk-pixbuf-config" in "/opt/gnome/bin/gdk-pixbuf-config". With the bash-shell you can replace the compiling

```
> make
```

command by:

```
> PATH=$PATH:/opt/gnome/bin make
```

Or you change the Makefile entry "GDK_PIXBUF = gdk-pixbuf-config" to "GDK_PIXBUF = /opt/gnome/bin/gdk-pixbuf-config".

User guide

3 Introduction

3.1 Quick "on the fly" tour

In this Quicktour you will get a short overview over iceWing and experiment with it's GUI. For this you will start a session with a small video as input and then play a bit with a demo plugin. With the sources of iceWing you got a small video, which you can use during this tour. It can be found in the docs directory under the name "quicktour.avi". Alternatively, you can choose any other image or video (of a format, that iceWing supports).

Compiling the plugin During installation of iceWing the demo plugin was not compiled and installed. So let's do that now.

```
> cd {wherever your iceWing-sources are}
> cd plugins/demo
```

Now make sure that you have $(PREFIX)/bin in the default-execute path or, alternatively, change the Makefile in the current directory. The Makefile needs to find "icewing-config", which was installed in $(PREFIX)/bin. Then simply

```
> make depend
> make
> make install
```

This compiles the demo plugin and installs it under the name libdemo.so in the directory $(PREFIX)/lib/iceWing, where iceWing can find it automatically.

The quick tour You tell by command line parameters from where iceWing will take it's images. There are several kinds of sources:

- file-loading from disk (like you do for this session)
- grabbing from camera
  (only available, if iceWing was compiled with AVlib support)
- DACS streams
  (only available, if iceWing was compiled with DACS support)

So let's jump in and start iceWing with the example plugin "demo": You launch iceWing in the shell-command line with (as one single line, after changing to the right directory)

```
> cd {wherever your iceWing-sources are}
> icewing -sp docs/quicktour.avi -l demo
```

On the starting console iceWing and the plugin write constantly their status.

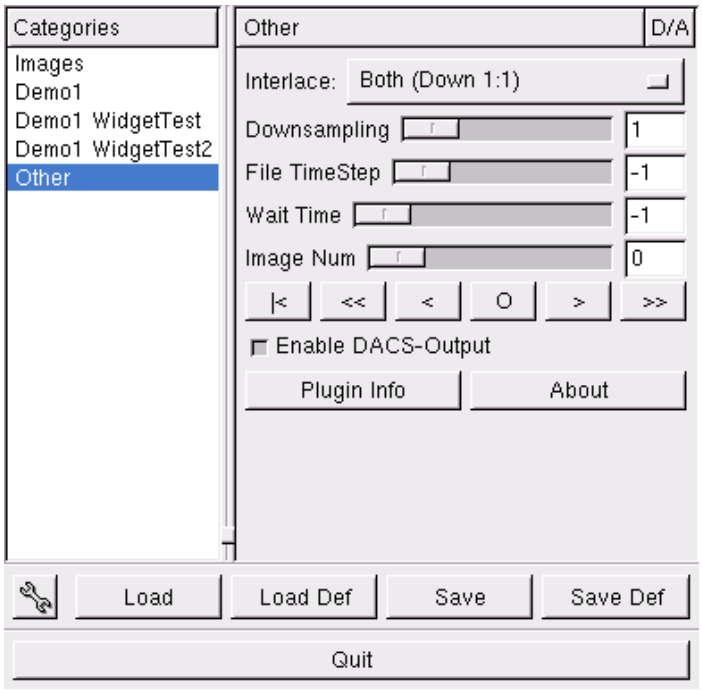
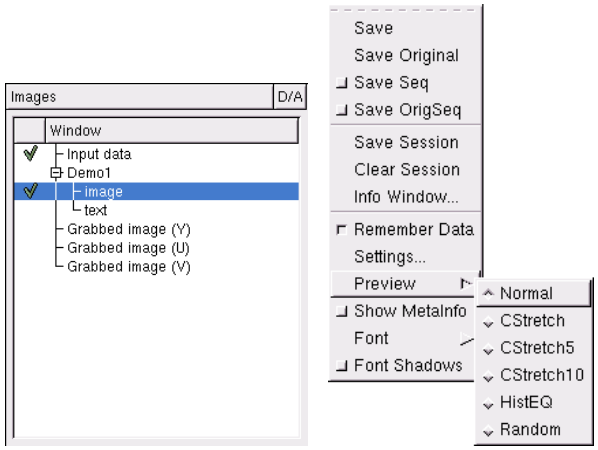


Figure 3.1: The main window of iceWing.

In figure 3.1 you see the iceWing main GUI, so lets explore some things. On the left side you see the "Categories" area, which has now "Images" and "Other" and some plugin pages.

The category "Other" Click on "Other" to activate this special page. You can see several sliders on the right side. If you have during a session as data source not a video stream but one single image, "Image Num" will always be set to 0. None the less iceWing has still the option to (re)acquire that single image on a timely basis: The slider "Wait Time" sets the delay (in ms), after that this (for video-streams: next) image shall be acquired automatically. If set to -1, iceWing waits until you click manually the read buttons below. So now set the wait time to 200ms, and you see the mass of console output greatly reduced.

The plugin pages You also launched with the command line parameter "-l" the plugin "demo", which is inside libdemo.so. Every plugin can generate none or more page-entries on the left "Category" side. By selecting the pages you can view plugin status or change the plugin parameters on the right side. Page "Demo1" has some options, that directly affect the plugin. "Demo1 WidgetTest" and "Demo1 WidgetTest2" not *do* anything useful. But they show you all the available GUI-elements of iceWing, that can also be easily used by your own plugins.



(a) (b)

Figure 3.2: (a) The page "Images". (b) The context menu of window "Demo1 image".

How to display plugin results? With click on category "Images" you see on the page on the right side several "Window" entries coming up (see figure 3.2(a)). They show you all windows that the plugins wish to display. Double-click (de)activates the window of that entry and you can see what the plugins are doing.

You see one window "Input data" for the current input image, one separate window for each YUV channel of the source image and two windows for the output of the plugin "Demo1".

Lets now activate the first demo window called "Demo1.image". Inside this new window press the right mouse button and verify that "Remember Data" is active (see figure 3.2(b)). This flag makes all coming zoom commands operate on the currently loaded image data, not only on *new* acquired/grabbed images. If you deselect this option, and e.g. try to zoom around, the zoom commands will still be remembered - but the window content will still display (unchanged until next image reading) the last acquired image without any of the zoom commands visible. When you set "Wait Time" = -1, you see the "Remember Data" flag's functionality very clear: When

"Remember Data" is inactive, only reloading the image source will refresh the window content and show your until then done zooms.

Zooming the window content Inside the window shift/crtl/alt-keys plus middle mouse button zoom in/out/reset the image. When you just hold the middle button and move the mouse around, you can pan inside the image, if you have zoomed into it. If you have problems with the zooming, look at the section 5.2.6 "Middle mouse button functions".

Now play a bit with the "Remember Data" flag and "Wait Time" and zooming to work it out.

Manipulating image colors In the context menu (right mouse button) you can see the "Preview->Random" option. This tool randomizes the color assignments. This can be useful to verify the output of a color separating plugin, because very similar/neighbored colors now get very distinguishable.

Saving the zoomed part into a separate file After you played around a bit, you can safe the frame content into a file with context menu entry "Save". The picture will be the window content and it's name be "Gsnap_0.ppm". If you wish different name/format, you can change this (and more) by clicking the wrench icon of the main Window (see figure 3.1) to get the preferences window. There you can change the image name to e.g. "quicktour_zoomed_%d.ppm" (the %d is a placeholder for the image number).

More about this plugin via "plugin info" On the page "Other" in the main window (figure 3.1) click "Plugin Info". Here you see how the plugin "Demo1" is integrated into the iceWing system, You see the sheet and the only two active plugins are "Demo1" and "grab". The plugin "grab" is needed to acquire the image from hard disk (command line parameter "-sp"). All other plugins ("record" etc.) are inbuilt ones, but they are not registered into the current session of iceWing and thus are not active.

More of this all later in the section 5.2.4 Plugin Info...

3.2 The special plugin "grab"

The plugin "grab" is a very basic and thus inbuilt plugin, that allows to acquire images or video streams from grabber hardware, disk or via DACS from external processes. This plugin will be used by nearly all other image processing plugins. But iceWing not *needs* this plugin! But then you need your own plugin as "data-provider" for e.g. audio streams or likewise.

Some comments on up-/downsampling iceWing provides the images via the plugin "grab" (or other data-provider plugins). And the plugin "grab" provides two separate queues of image histories: downsampled and upsampled (i.e. full/original sized) images. With a downsample factor of 1 the two queues have the same images, while a factor of 3 makes the downsampled images consist of every 3rd pixel of the original sized image. Both queues have a history size - how many images will be stored for plugin access to older images. Excellent - but what for?

Well, this feature is useful when both reduced and detailed image size versions are needed for the global task. E.g. a continously operating plugin "trajectory tracking" uses the faster downsampled images to swiftly keep track of the hand-position. Meanwhile, but much less frequently, another very time-intensive plugin "object detection" separates an object in that hand. For this job, it may register for the more detailed queue of upsampled images.

4 The command line interface

4.1 The command line parameters in detail

```
> icewing -h
```

gives you the list of command line parameters. Now they will be explained more detailed, arranged to subject.

General options

-n <**name**> When you use DACS, the launched instances of iceWing must be somehow addressable. This option specifies the process name of this instance of iceWing - the default name is "icewing". If there are several instances of iceWing (network wide), you must care: Give at least those iceWing processes unambiguous names, that are used with DACS.

@<**file**> This allows to store arguments in files. Option "@" replaces it's argument <file> with the content of <file>. Any lines in <file> starting with '#' are ignored, the remaining lines are treated as further options.

-fr <**ms**> Sets frame rate in [ms per frame]. This option is only for the AVlib version for Alpha machines, default: 40ms. On Linux machines this option is ignored.

-p <**width**>**x**<**height**> Sets the size of preview windows, default: 378x278.

-rc <**config-file**> Load the config file <config-file> *additionally* to the standard file "$(HOME)/.icewing_value" (which is read first).

-ses <**session-file**> Load the session file session-file *instead* of "$(HOME)/.icewing_session" and use this file for any session related operations.

-time <**cnt**> Specifies after how many main loop iterations time measurements are given out. If cnt<=0 all time measurements are disabled. The default is 50.

-iconic Start the main iceWing window iconified.

-t <**talklevel**> iceWing outputs debug messages only if their level is below <talklevel>, default: 5, used range of levels: 0..4.

Input options

Remember: All this options, that are related to image input (-sg, -sp, -sp1, -sd, -c, -f, -r, -crop, -rot, -fr, and -oi) are passed to the special plugin "grab". If you use another plugin as data-source, that plugin will have it's own input options (passed via "-a"). Neither "grab" knows of the other plugins options, nor does the other plugin see this input options.

So *this* options could also be thought as "input options for plugin grab". You can use multiple instances of the plugin grab and thus multiple images at the same time. If you want to do that, you have to pass these options via the iceWing option "-a" to the additional grab instances. Thus every instance of grab can get it's very own options. The multiple instances are created by loading the plugin via the option "-l" multiple times.

-sg <inputDrv> <drvOptions> *S*ource of *G*rabber: If you use a grabber camera for your images, you must include the AVlib in installation. The AVlib supports several camera systems, and here you select which you use. iceWing simply passes the given parameters on.

<inputDrv> can be one of PAL_COMPOSITE, PAL_S_VIDEO or FIREWIRE (or abbreviated "C", "S" or "V", and "F").

<drvOptions> can be "0", "1" or a driver specific option string. MME (used on Alpha machines) can handle two grabber cameras, and 0/1 chooses which of them to use. For Linux systems this 0/1 option gets ignored. Instead the Linux drivers have a number of different options further described in section 4.2.

And at least the Linux part of AVlib provides help for it's driver options: If you are not sure, which options your selected driver, e.g. "F", has, try

```
> icewing -sg F help
```

But remember: This help does not work, if you have in your configuration a wait time of -1 (see 5.2.3)! Why? The grabber driver needs to get called before it can output it's help. Oh – but a wait time of -1 makes iceWing wait for manual image grabbing, so the driver gets not activated anyhow. In such case the GUI appears and you must once manually click the "Grab Image" button. Alternatively you can change the configuration of your wait time *before* you gather help about your grabber options.

If you only give -sg without anything special, the default setting is PAL_S_VIDEO with no options.

-sp <fileset> *S*ource of *P*ictures: What you specify as <fileset> will be the picture(s) data-source for the plugin "grab". Reaching the last picture iceWing loops, beginning again with the first picture.

iceWing natively supports the *pnm* image format. Depending on your version (or if installed at all) of the gdk-pixbuf library, the range of supported image formats is greatly enhanced. Version 0.16 provides e.g. this formats: *bmp*, *gif*, *ico*, *jpeg*, *png*, *ras*, *tiff*, and *xbm*. *pnm* images are read in bit depths from 8 to 32 and in special variants float and double images can be read, too. *png* images are read in 8 bit and 16 bit depths. All other formats are 8 bit only.

<fileset> specifies the list of file names. It has the following format:

fileset = fileset | 'y' | 'r' | 'e' | 'E' | 'file'

Single Pictures You can simply give single pictures as files

```
> icewing -sp image.ppm image2.gif
```

'y', 'r' → YUV or RGB The pictures can be stored as color model YUV (which is the default) or RGB. With a "y" or "r" in front of the name you can specify the color model of the coming files. So this example names one YUV, two RGB and another YUV picture as data source sequence:

```
> icewing -sp imageYUV1.ppm r imageRGB1.ppm imageRGB2.ppm
y imageYUV2.ppm
```

'file' with %d or any int based printf() conversion specifier As further option you can name a whole series of pictures: with e.g. %d the plugin "grab" replaces %d by integer numbers beginning from 0 and tries to open the file. As another example "%04d" matches all numbers, starting with 4 leading zeros. As soon as iceWing cannot match the current number, it moves on to the next fileset. E.g.

```
> icewing -sp image%03d.ppm picture%d.ppm
```

makes the plugin "grab" increasingly scan for (and if found: load) files named with "image000.ppm", "image001.ppm"... If no further file is found, it scans for pictures named "picture0.ppm", "picture1.ppm"...

'file' with at least %t or %T, or a combination of %t, %T, and %d An alternative method to specify a series of pictures, one example would be '/tmp/image%T_%t.png'. If %t or %T is inside the file name part of one 'file' to the -sp option, iceWing scans the complete directory, in the example '/tmp', for files matching the file name part with any numbers replacing %d, %T, and %t. It loads the found files sorted by the numbers replacing %d, %T, and %t in that direction. I.e. the coarsest order is given by %d and the finest by %t.

In this case no printf() style format specifiers are allowed, as files with any number format are used at the same time.

'e', 'E' → Open files/Check file extension iceWing must know the number of images you specified on the command line. To verify if a file is a movie file and then get its frame count, iceWing opens every file during startup. With an 'E' in front of the file names iceWing opens only these files which have a known movie extension (e.g. '.avi'). This speeds up the program start. With an 'e' in front of the file names you switch back to opening all files. The default is to open all files.

-sp1 <fileset> This is just the same as -sp. But after the last picture is reached and every registered plugin has finished it's work on that picture, the iceWing process ends instead of looping back to the first picture.

-sd <stream> [synclev] Use a DACS stream as input of images (for plugin "grab").

An external process creates that stream of images somewhere in the network. It has published it via DACS, and this iceWing process can order that stream as input of images.

The stream can have some synchronize-tokens of several hierarchical levels integrated. This SYNC-tokens of increasing level create substructures of the incoming data (e.g. letters, words, sentences...). You can register the stream at a given synclevel. Level 0 means *every* single image will be delivered to this iceWing instance. Higher levels lead to fewer images, depending strongly on the SYNC-level philosophy of the stream creating process. You need to get this information about the stream creating process to choose the appropriate SYNC-level, with that you register the DACS stream. If the stream delivering process separates the images with SYNC-level 2 and you order this stream at this level, you will get always the latest image. Any older images get dropped off the stream, as soon as the new image arrives the stream. So with this strategy iceWing gets always up to date images - simply by ordering at the appropriate level.

Caution: SYNC-level of 0 is *special* and means, that this instance of iceWing gets *every* single image, that is put into the stream (no loss of any image). You may need this, e.g. because plugins sometimes need access to older, but still unreceived images. But the DACS process must store all of the undelivered images. If iceWing consumes the images at a slower rate than the images are put into the stream (in average), this will definitely lead sooner or later to a *huge* size of the DACS process – and finally (when the storage limit is exceeded) the process gets killed by the system.

If you wish write your own image-creation process to send images to iceWing via DACS: There is already the SFB-360 internal data type "struct Bild_t" (which is declared in the file "sfb.h"). It encodes the image, and you must use it as

the type to be passed to DACS. Then iceWing can receive images from your own process.

For further details about DACS see the dissertation of Nils Jungclaus [Jun98].

-crop x y width height Crop a rectangle starting at position (x,y) of size width x height from the input image. If width or height are smaller than zero or zero, the values are measured from the right or bottom side. E.g. "-crop 5 10 -5 -10" would crop a border of 5 pixel from the left and right sides and a border of 10 pixels from the top and bottom sides of the input image.

-rot {0, 90, 180, 270} Rotate the input image by 0°, 90°, 180°, or 270°. The default is 0°.

Options for up-/downsampling behavior

-c <cnt> iceWing internally manages a queue of downsampled images. With this option you can specify the length of this queue.

Default value is 2.

-f [cnt] If you do not specify this option, iceWing has only the downsampled queue of images. With "-f" iceWing activates the *F*ull sized (i.e. the upsampled) queue of images.

The optional [cnt] sets the queue size, the default is 1.

-r <factor> *R*emember downsample factor of input images.

If you use already downsampled images as input, unfortunately iceWing does not know this without further notifying. <factor> tells the interested plugins, what factor the source images got downsampled.

Remember: With downsampled image sources, plugin grab will *additionally* downsample the input into the downsample queue. So when the input images already have downsample factor of 2, and the current iceWing instance has downsample factor of 3, the images in the downsample queue will have a *true* downsample factor of 6 (compared to the original image), while the images inside the upsampled queue have downsample factor of 2. The only way to solve this and e.g. simulate the original size of the image is to use this option "-r": You tell iceWing, what downsample factor the input images already have. And the plugins can (but not need to) make use of this additional information.

And also remember: Even the commando "Save Original" saves the original (=unrendered) image, but *including* the given downsample factor (set in figure 3.1 page "other" or in paragraph Downsampling 5.2.3).

Output/remote control options

The "-o" options have several different purposes regarding the communication to other programs and with the sub options you specify, what to output and what interfaces to enable.

-of With option "-of", this instance of iceWing can be fully remote controlled via DACS including nearly every single GUI-widget element.

This uses the capability of iceWing to save and load all it's current status into the config file (while session file stores the window properties). Remote control via DACS works quite similar: With this option "-of" iceWing publishes DACS wide the function void <icewing>_control(char[]). <icewing> is the name of this instance of iceWing (that you specified with option -n). The "char[]" means, that you send a normal c-string as parameter to that function. The content of that string can be any lines of the iceWing configuration file and iceWing accepts the new settings.

Additionally this option publishes to DACS the function struct Bild_t <icewing>_getImg(imgspec). With this function, external processes can receive an image from this instance of iceWing via DACS. The images are send encoded in the SFB-360 "struct Bild_t" data type. There are further options, to allow the external process to select precisely which image it receives, and in which downsample format.

The format of "imgspec":

```
['PLUG' <plugnum>] ('NUM' <imgnum>|'TIME' <sec> <usec>|
'FTIME' <sec> <usec>) down
```

PLUG If multiple instances of the plugin grab are running, multiple images are available at the same time. With PLUG you can select the image from the instance <plugnum>. The default is 1, i.e. the image from the first grab instance.

NUM Every single image in iceWing has a continuous number, starting with 0. iceWing returns the image with the number <imgnum>. If <imgnum><0, return a full size image, see option "-f". If <imgnum>==0, return the current full size image.

So the sign determines, from which queue iceWing takes the image from: the upsampled or the downsampled queue (See also options "-c" and "-f").

If the upsampled queue is not existing, you will always get the downsampled version of the image.

TIME iceWing returns that image with a grabbing time most similar to (<sec> <usec>). The image is taken from the downsampled queue.

If TIME is in the future, the nearest image is taken - and that will always be the most recent image.

FTIME iceWing returns a full size image with a grabbing time most similar to (<sec> <usec>).

Again, if the upsampled queue is not existing, you will always get the downsampled version of the image.

<**down**> iceWing will downsample the returned image by factor <down>. This factor is applied *additionally* to the iceWing downsample factor (adjustable in page "Other", see figure 3.1)!

As example: iceWing has set a downsample factor of 2 and this option <down> is e.g. set to 3. Now the image will be delivered to DACS with a true downsample factor of 6 (well, with FTIME it is 3).

-oi [interval] Output images on DACS stream <icewing>_images and provide a function void <icewing>_setCrop("x1 y1 x2 y2") to crop the streamed images. <icewing> stands for the DACS name of this iceWing process (see option "-n"). With the optional [interval] iceWing will send only every nth image to the stream. If the upsampled queue exists, you will get an upsampled image, otherwise a downsampled image. The images are sent in the "struct Bild_t" format.

With the function <icewing>_setCrop("x1 y1 x2 y2") a freely defined rectangle of the image can be dumped to the stream. The parameter string defines the rectangle. The four coordinates refer to the full size, i.e. not downsampled image. iceWing adapts them internally to the real image size.

-os Output some (currently very few) status information on DACS stream <icewing>_status.

The function "output_out_status (char *msg)" declared in output.h sends on this stream.

Plugin options

-l <plugin libraries> Each plugin lives inside a library. This option loads the plugin into iceWing. If you wish to have several instances of a plugin, repeat the name of the relevant library. But be cautious: not all plugins can operate as several instances (e.g. plugin "min")!

Search order: First the library is searched as specified with this option. If the library was not found, iceWing searches again in $(PREFIX)/lib/iceWing/ for the library. If still not found, the name is expanded by "lib[...].so" and again searched in $(PREFIX)/lib/iceWing/.

-lg *Not* for Alpha machines! Similar to "-l", but with a *very* impacting difference: while dlopen()'ing the library, the flag RTLD_GLOBAL is set (makes all not-static objects of the library global for the whole iceWing process)! So this is more a linker option, than an iceWing feature!

Use only, when you *really* know what you are doing (e.g. *great* danger of name-clashes...)!

-h Additionally to the help page iceWing writes the names of all plugin instances, that are created by parameter "-l" or "-lg" and then terminates. You need the precise names of the plugin instances if you wish to send options to specific plugin instances with option "-a".

-a <plugin instance> <option> Send arguments to a plugin instance.

Every plugin should provide help on it's options. To find those help messages, use parameter '-a plugin "-h" '.

-d <plugins> Disables the given plugin instances.

Normally all loaded plugins get activated. You can toggle plugin activation also via GUI, but sometimes you may wish to start an iceWing session with an initially disabled plugin instance.

E.g. '-d "backpro imgclass" '

4.2 Parameters of the grabber driver

Under Linux, if you use a grabber camera as a source for your images, you can pass different options to the grabber drivers. If you pass "help" as an option, you get an overview of all available options, e.g. for the firewire driver:

```
> icewing -sg F help
```

The help message is given on the first invocation of the grabber. So if you have in your configuration a wait time of -1 (see 5.2.3), you need to manually grab an image to get the help output. The different options are separated by ":". Parameters of an option are separated by a "=" from the option name, e.g. "camera=2:bayer=hue" would be an allowed option string. In detail the different options are:

V4L2 driver for Composite or S_Video devices

help Shows the help page of the driver.

device=name Specifies the device, the driver will use to grab images from. If this option is not specified, "/dev/video" is used.

buffer=cnt V4L2 can use several intermediate image buffers to compensate for an intermediate slowness of the program before grabbing the next frame. This option sets the number, the default is 4.

Firewire driver for DV-cameras connected via firewire

help Shows the help page of the driver.

device=name Specifies the device, the driver will use to grab images from. If this option is not specified, "/dev/video1394" is used.

camera=val Multiple cameras can be connected to the computer via one device. Here you can select with a number starting with 1 which of these cameras should be used. The default is 1, the first one.

fps=val DV-cameras normally support different speeds in which they can deliver the images. This option sets the frame rate the camera should operate in. Supported frame rates are: 1.875, 3.75, 7.5, 15, 30, and 60. The default is 15.

mode=yuvXXX|rgbXXX|monoXXX|16monoXXX DV-cameras can support different color spaces and different image sizes for the images. With this option you can select which mode the camera should operate in if an image without any downsampling should be grabbed. If the desired mode is not supported by the camera, the driver falls back to a supported mode. The "XXX" specifies the desired width, e.g. "mono1024" would be a possible mode specifier. The default is yuv640x480.

bayer=down|neighbor|hue Some cameras support color image grabbing, but deliver the image not decomposed but as one gray image plane with an embedded bayer pattern. In a bayer pattern a square of size 2 by 2 pixels holds information about all three RGB channels. To decompose this information, different interpolation methods exist. If this option is given, a gray image of depth 8 or 16 bit with an embedded bayer pattern is expected and decomposed with the specified method. The supported interpolation methods:

- **down** The 2x2 bayer square is used to only get one color pixel, the destination image gets downsampled by a factor of 2.
- **neighbor** Nearest neighbor interpolation, where each interpolated output pixel gets the value of the nearest pixel in the input image, is used.
- **hue** Smooth hue transition interpolation. Here the green channel is gained by bilinear interpolation. For the blue and the red channel a "hue value" gets defined as B/G or R/G. The neighboring hue values are then used to estimate a color pixel. E.g. if a blue pixel is located on the left and on the

right side of a pixel, the blue pixel in the middle gets estimated by: $B_M = \frac{G_M}{2} * \left(\frac{B_L}{G_L} + \frac{B_R}{G_R}\right)$

pattern=RGGB|BGGR|GRBG|GBRG The information in a 2 by 2 bayer square can be ordered in different ways. This option specifies how it is ordered. The default is RGGB, red in the first pixel, green in the second pixel and the first pixel on the second row, and blue in the second column on the second row.

stereo If given, an image of type YUV422 is expected. However, this memory is not interpreted as a normal color image, but as two interlaced images which get decoded and saved one after the other. For example the Videre stereo camera saves its two images in this way.

debug If given, different debugging information about the camera, it's capabilities, and the current driver status is printed to the console.

4.3 Configuration files

iceWing uses two configuration files, which get loaded and stored during runtime:

.icewing_session stores the window properties of the current session. Default wise it's "$(HOME)/.icewing_session", but you can use alternative files via command line option "-ses". In the preferences window or with the context menu you can save your current session into an alternative file.

The content is simple - for each active window there is an entry like

"Name of the window"= win-x win-y width height zoom pan-x pan-y

where win-x and win-y specify the window position, zoom specifies the zoom factor of the window (0 means fit-to-window) and pan-x and pan-y specifies the panning position for the content of the window. The zoom and panning values are only stored if "Save pan/zoom values" in the preferences window is active, see section 5.2.2. Lines starting with "#" are treated as comments and get ignored.

.icewing_value stores all settings of every single GUI value of the plugins and the iceWing system. Additionally, the hotkeys for the context menu of the image windows get stored in these files. Default wise it's "$(HOME)/.icewing_value", but you can use *additional* files via command line option "-rc" or in the main window with the Load/Save buttons.

If you wish to remote control the iceWing process via DACS, you must know the structure of the content of this config file. The best will be to save the current settings and have a look at the file.

Every line holds for one iceWing widget it's setting. Each widget is unambiguously addressed (it's path) via it's window name or it's category name and it's widget name. It's entries look like this: "windowname.widgetname" = value

Each widget type has it's own kind of values (e.g. for booleans: true=1, false=0), the most complex widget surely is "list". More details about the different widgets can be found in the Programming Guide in section 7.3.1.

5 The Graphical User Interface

5.1 The iceWing render chain

Why should you know this details? Well, you will need to understand the basics of the underlying render mechanism to fully understand how the GUI works and what the many GUI-commands are used for! In iceWing rendering data to image windows is done in several steps. Figure 5.1 shows an overview of the complete rendering process. By using the different GUI commands you can manipulate and inspect the data at different positions of the render chain.

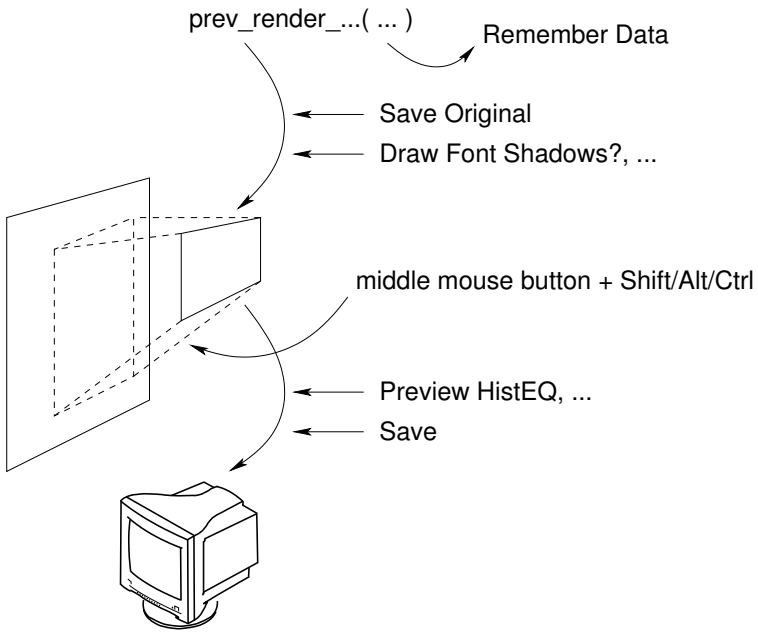


Figure 5.1: The render chain of iceWing.

If a plugin wants to display any data, it calls one of the different `prev_render_xxx()` functions (see section 7.3.2 for an in deep description of these functions). An example is the plugin "grab", which displays the loaded or grabbed image in the window "Input data".

The first thing these render functions do is to check the "Remember Data" flag, which is associated to every image window. If the flag is set, the complete data which

is passed to the render functions is copied for later use. This allows to re-render the complete image without the help of the plugin. If any of the remaining parameter of the render chain are changed, e.g. the displayed part of the data or the zoom level, the data can be immediately redisplayed. This means, that you can instantly see the effect on your current image. Otherwise, if the flag "Remember Data" is not set, the effect of the changed parameter gets only visible after the next call of the plugin to the `prev_render_xxx()` functions. For the "grab" plugin for example this means that you see the effect not until the next image is loaded or grabbed.

If an image should be displayed, the data of the complete image can now be saved with the help of the "Save Original" GUI function. *Attention:* This saves only the first image, not any text, which may be rendered on the image, nor any lines, circles or anything else besides the first image.

The next step in the render chain is the rendering of the data into an internal buffer. During this the data can be modified. E.g. you can add a drop shadow to all displayed text via the context menu of the image windows. Moreover the displayed region and the size/zoom factor can be changed interactively with the mouse. Thus the coordinate system for the data as specified by the plugin (the "world coordinates") and the coordinate system for the rendered image as displayed on the screen (the "screen coordinates") are not identical.

The last step in the render chain is the display of the buffer on the screen. During this some color changes, e.g. a histogram equalization, can be applied to the buffer. Besides displaying the result on screen the result can be saved to a file with the help of the "Save" GUI function.

5.2 The GUI commands

5.2.1 iceWing main window

The main window is divided into two main parts: In the upper area you see "Categories" and to it's right the page content of the selected category. Most plugins will create at least one entry (called "page") into the categories-list. It allows you to check/change that plugins parameters. Please see the respective plugin's documentation for any plugin specific information. In this documentation only the somewhat special categories "Images" and "Other" will be described in more detail (see sections 5.2.5 and 5.2.3).

Besides these categories you see some global buttons in the iceWing main window:

Detach/attach symbol (D/A) Clicking this symbol at the top right corner will detach this page into a separate window. Clicking again will put it back into the iceWing main window.

You will use this feature, if you repeatedly wish to flip quickly from one page to another. Or you change the slider on plugin page 1 and want to see the effects on plugin page 2. If you e.g. work on a plugin sheet and wish to single step (wait-delay=-1) through the next images and watch the plugins doing, you surely wish to detach the "other" page.

Preferences button The wrench icon opens the preference window, where you can configure different settings for the iceWing main program. See section 5.2.2 for a complete description.

Load/Save buttons Loads/saves all made settings from all widgets inside iceWing from/into the file "$(HOME)/.icewing_value", if you selected the "Def" (meaning "default") variant of the buttons. The Load/Save buttons allow to select the file name. But be aware: this is different to the command "load/save session", which stores the window size and position of all open windows, not the widget settings.

5.2.2 Preferences button

The wrench icon opens the preference window, where you can configure different settings for the iceWing main program. "Image Saving" specifies settings for the save functions in the context menu of image windows. In detail these are

Image format Specifies the file format used for saving images. If "By extension" is selected, the format gets selected based on the extension of the file name.

Image name The file name, under which the images will be saved. You can embed different information in the name by using the following modifiers:

%d: The consecutive image saving counter, starting at 0.
%t: The milliseconds part of the time the image gets saved.
%T: The seconds part of the time the image gets saved.
%b: Name of the current user.
%h: The system's host name.
%w: The name of the window, from which the image gets saved.

Any of the above modifiers can be changed by printf() style format specifiers. E.g. "image%03d.ppm" would result in "image000.ppm", "image001.ppm" and so on as file names.

AVI framerate If avi files get saved, this value will be entered in the file for the frame rate. That is this value will not really change the saved data, only this one setting in the header of the saved avi file is changed.

Quality The quality and thus the compression value used when saving jpeg images or avi files. 100 means a small compression and thus a good quality.

Show SaveMessage If activated a dialog is shown after every image saving which confirms the successful saving.

Save full window If activated and "Save" or "Save Seq" is used as the saving command, an image of the size of the complete image window will be saved. If deactivated a black border around the image will not be saved.

Reset FilenameCounter By using %d in the file name, a consecutive image saving counter is embedded in the image file name. This button resets this value to zero.

"Other" has settings for the session handling and the GUI. In detail these are

Save As / Save / Clear These buttons are similar to the menu entries in the context menu of the image windows. "Save As" saves the current session in a file and allows at the same time to change the current session file name, "Save" saves the current session in the current session file, and "Clear" physically deletes the current session file and thus clears this session. The default session file is "$(HOME)/.icewing_session".

A session file stores a list of windows and their configuration, their position, their size, and optionally their zoom and panning values. On the next start of iceWing a session file can be loaded, which then will restore the windows and the window configuration.

Auto-save at exit If activated, at program exit the iceWing window configuration will be saved in the current session file.

Save pan/zoom values If activated the current panning position and the current zoom value will be saved additionally in session files. Otherwise only the position and size of all open windows will be saved.

Use tree for image windows If activated the window list in the category "Images" will use a tree widget. Otherwise a list widget will be used. If you change this setting, you must save the current settings (by using the "Save" or "Save Def" buttons in the main window) and restart iceWing.

Auto add render widgets The rendering in image windows can be modified by using commands in the context menu of the image windows. By default these commands are only visible if the plugin which performs the rendering has added the commands. If this button is activated, all commands which belong to the kind of rendering used in an image window are added automatically by iceWing.

5.2.3 Commands in category "Other"

The category "Other" has widgets, which specify settings for the plugin "grab", and some widgets for the iceWing main program:

Interlace Different grabber camera systems have different methods of sending the video-stream. Here you can select what of the grabbed data should be used. "Both" selects the complete image, "Even" selects only the even field of an interlaced image. "Even + Aspect" grabs only the even field and afterwards halves the image in the horizontal direction to get square pixels again. "Down 2:1/RegUp 2:2" adjusts for grabbed halve field images by halving the image in the horizontal direction and afterwards telling other plugins that the image was downsampled in the vertical direction, too (see also command line parameter "-r").

Downsampling A ratio of 1 will grab and deliver the image 1:1. But if you e.g. set it to 3, then only every 3rd pixel in both horizontal and vertical direction of the full sized image will be delivered to other plugins - it will become scaled down by a factor of 3. There can be two queues, where the images are stored – one for the original size image and another queue for the downsampled images (see command line parameter "-c" and "-f").

File TimeStep Every image the plugin "grab" provides to other plugins is marked with a time stamp. E.g. if you use the grabber, the time stamp marks the time the image was grabbed. With this slider you can select the behavior if files from disk should be loaded. -1 sets the time stamp to the time the image was loaded. Other values specify an increase of the time stamp in ms. E.g. if set to 40 the first image gets a time stamp of 0ms, the second of 40ms, than 80ms ....

Frame correct Wait Time If this button is not selected, iceWing waits exactly the time slice specified with the next slider before the next image gets acquired. If this button is selected, iceWing adapts this time slice. For example if the processing of the last image took 50ms and the 'Wait Time' slider is set to 200ms, iceWing will only wait 150ms.

Wait Time and positioning buttons Sets the delay (in milliseconds), until the next image shall be acquired from disk/grabber. If you set it to -1, iceWing waits until you manually change the image by pressing one of the positioning/acquiring buttons. This -1 works like a "pause-mode".

Image Num If you work with a video stream on disk, which you have specified with the command line parameter "-sp", this slider will appear. It shows the current position inside the video stream and allows to seek to an other position.

Enable DACS Output If iceWing outputs any data via DACS, e.g. if you have used the command line parameter "-oi", toggling this button disables/enables this outputting.

Plugin Info Opens the "Plugin Info" window, which shows different information about all loaded plugins. This window is described in more detail in the section 5.2.4.

About Opens a window showing some information about iceWing, e.g. the version number and the copyright.

5.2.4 The "Plugin Info" window

The "Plugin Info" window, which can be opened with a button in the category "Other", shows different information about plugins iceWing knows about and about the communication between them. Figure 5.2.4 shows all the different pages of this window.

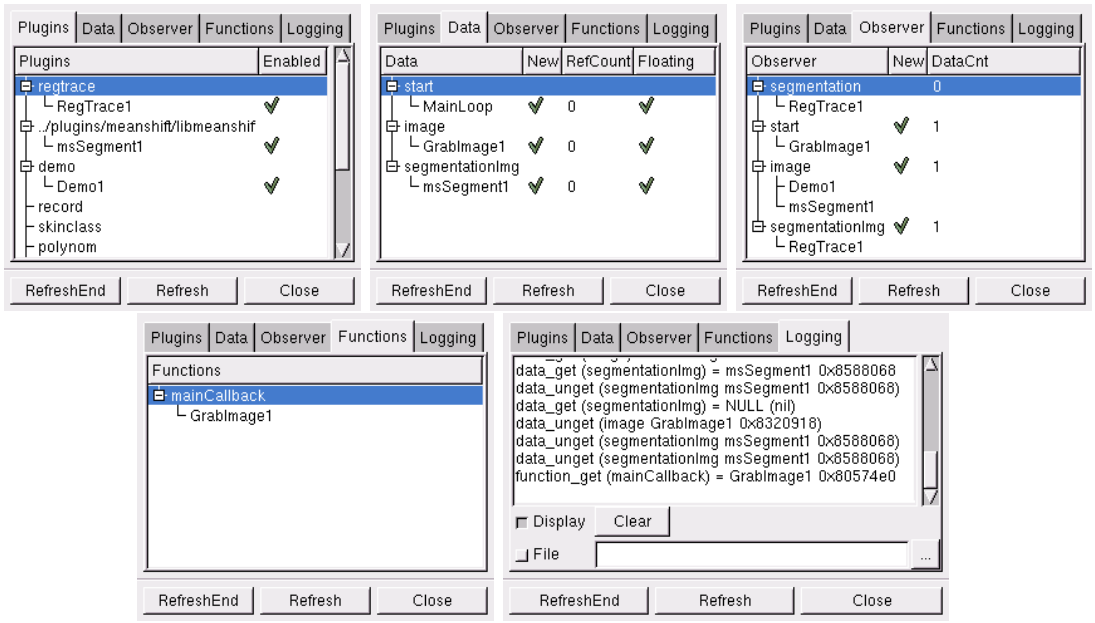


Figure 5.2: All pages of the window "Plugin Info", which shows information about loaded plugins.

Inside a main loop iceWing calls the registered plugin instances repeatedly. The order in which the instances get called is defined by the plugins. For this and for the communication between plugins iceWing offers different functionality. Plugin instances can interchange data, they can observe the provision of data, and they can call functions of other plugin instances. Details about these communication possibilities is displayed on the remaining pages. The displayed information is not continously updated. By pressing "Refresh" the current internal state about the communication information is displayed immediately. "RefreshEnd" defers the display shortly before the end of the main loop, directly before any floating data with a reference count of zero gets deleted (see below). So by pressing "RefreshEnd" you will get in a lot of cases information about all data, which was provided during the last main loop iteration.

Plugins On the first page *"Plugins"* a list of all plugins iceWing knows about and all instances of these plugins is displayed. A double-click on one of the plugin instances (de)activates the instance. If an instance is deactivated no functions of this plugin instance are called. This is similar in effect to the command line parameter "-d". Additionally there is a context menu with two entries. The first entry allows to (de)activate an instance of a plugin. With the second a new instance of a plugin gets created. This is similar to specifying "-l pluginName" on the command line.

Data The *"Data"* page shows information about all data the plugins have created. In iceWing an identifier, a string, is always associated with all data. This identifier allows the plugins to access the data. In the first column the identifier and the instance name of the plugin, which has made the data available, is displayed. All data inside iceWing is reference counted and gets automatically deleted if the reference count drops to zero. If the data is marked as "floating" the deletion is deferred until the end of one main loop iteration. Data is marked as "New" as long one main loop iteration is not finished since the data was provided by a plugin instance.

Observer The *"Observer"* page shows information about plugin instances, which observe the provision of data. If new data with an identifier, which is observed, gets provided by any plugins the observer of this data are called with the new data as an argument. Thus the order in which instances get called is defined by the provision of data and by the observation of these data elements. There is one special data element. The data "start" is not provided by a plugin, but by iceWing at the start of every main loop. Plugin instances can observe this data and thus get called at the start of the main loop. The "Observer" page shows the different registered observer and information about currently known data elements with a corresponding identifier. New data is available, if the data

was provided during the current main loop iteration. Additionally the amount of available data elements with the observed identifier is displayed.

Functions The *"Functions"* page shows information about all functions, which plugin instances have published. The function identifier and the plugin instance names, which have provided the function, are displayed.

Logging The *"Logging"* page allows to get more detailed real time information about the communication between the plugins. By activating the "Display" toggle button calls to the different iceWing functions, which deal with the communication between plugins, are shown in the text widget above. For a description of the different functions please see section 7.2. The "Clear" button clears the text widget. Deactivating the toggle button stops the logging. Activating the "File" toggle button records this information in the file, whose name is given in the string widget next to the toggle button. The file is closed if the toggle button is deactivated.

5.2.5 Category "Images" and image windows

The category "Images" shows a list of all image windows that the plugins wish to display. Double-clicking an entry of the list opens or closes the window of that entry.

Every of these image windows iceWing or any plugin creates will have different standard menu entries in a context menu. You can access this menu by clicking with the right mouse button in an open image window. Depending on the things a plugin renders in the image, there might be additional plugin specific entries.

Here is a list of entries you will find in all or, partly, a lot of image windows:

Save Saves the actual visible window content, including all active rendering manipulations. In the preference window of iceWing you can specify different parameters for the saving process, e.g. the file name and the file format.

Save Original Saves the underlying original "world coordinate" image, without any active rendering manipulations, in the original color space (for example YUV). But if the downsample factor (page "other") is >1, you will still save downsampled images. This downsampling happens inside the plugin "grab" *before* the original images are anyhow rendered in a image window or passed to further iceWing plugins. Moreover this function does not save any texts, lines, regions, or anything else besides images the plugin may display in the image. Only the first image is saved.

Save Seq Once activated, every new acquired image (the visible window content, including all active rendering manipulations) is saved continuously until deactivated. Normally, if the active image format is for single images, a series of image files gets

stored. Otherwise, if the AVI file format is selected, a single video-stream file is stored. You can change this format in the preferences window, see section 5.2.2 for further details.

Caution! If you have selected the AVI format in the preference window, you must remember: To prevent corrupt AVI files, you must end this "Save Seq" by (de)selecting this command in the context menu again and continue with at least one new image to close the saved AVI file. Alternatively while recording you can close the whole image window, and iceWing sends the close file command itself. Changing the Image format in the preference window will do the same. Otherwise the AVI will miss some finalizing code and thus will be corrupt. So be warned: Just closing iceWing itself is a good way to corrupt your currently running AVI recording.

Save OrigSeq Same as "Save Seq", but the original "world coordinate" images are used. For the data, which gets saved by this function the same as already stated under "Save Original" applies.

Save Session Saves the current iceWing window configuration under the currently active session name. On the next start of iceWing every currently open window will be remembered as it is. Without any special parameters the next time iceWing is launched, the default configuration-file in "$(HOME)/.icewing_session" will be used to restore any windows. Alternatively, the command line parameter "-ses <session-file>" can be used to switch to non-default session files.

The Save/Clear Session menu entries are the same commands as the ones in the preferences-window.

Clear Session This physically deletes the current session file and thus clears the session. At the next launch the window layout of iceWing will look like the inbuilt default.

Info Window Opens a small window, which displays the coordinates and color (in several color spaces) of the pixel at the mouse position, if the mouse is inside any image window. Additionally, this window contains two toggle button for two special functions. If "Grab Values" is pressed, the info window waits on a press with the left mouse button inside one of the image windows. The values at the time the button was pressed are then additionally displayed in the info window. Thus two positions can be easily compared. If "Measure" is pressed, distances and angles in the image windows can be measured. If the left mouse button is pressed inside an image window and then moved to a second position, the distance of these two positions and the angle between a horizontal line and the marked line are displayed in the info window. Afterwards, to change the initially selected line, the end points of the marked line can be dragged around.

Remember Data (De-)activates the "Remember Data" mode, that was already discussed in detail in section 5.1.

Settings Opens a window, where you can change some further image window related options. As one point different options for the "Show Meta info" feature and the image rendering can be specified. Moreover, all plugins can add any widgets to this window. For a description of these options please refer to the special plugin documentation. The other options are:

Histogram "Show Meta info" displays histograms of all rendered images. Here you can change the kind of this histograms. "Use lines" changes the visualization form of the histograms (filled or single lines). "Include min/max" changes the method used for scaling the histogram in the vertical axis. If activated, the biggest value of the histogram is used to scale the histogram. If deactivated, the first and the last histogram entries are not considered during determining the biggest value of the histogram. Useful, if a small object is located on an otherwise black or white image.

scaleMin/Max Images in iceWing can have any data type ranging from unsigned chars to doubles. For displaying these images must be converted to a range of 0 to 255. With these sliders you can influence the conversion process. With "scaleMin = -2" the image values are simply shifted in the range 0..255 without locking further at the image values, e.g. for unsigned short images 65535 is displayed with a value of 255. "scaleMin = -1" clamps the image values to a range of 0..255.

All other values for the sliders consider the minimal and maximal values of the to be displayed image. The Min slider specifies the amount, the minimal value is shifted towards the maximal value, in percent of the difference of the minimal value and the maximal value. The Max slider shifts the maximal value towards the minimal value. All pixels darker then the shifted minimal value are displayed as black, all pixels lighter then the maximum are displayed as white, and everything in between is stretched linearly.

Entries of the submenu Preview

Normal Shows the image in the window with the original colors as they were specified during rendering.

CStretch The color histogram of the image is stretched: The minimum (and maximum) of all used colors is set to 0 (255) and all colors are stretched linearly to the full range 0 - 255 again.

CStretch5 5%, beginning with the nearest to black (white) pixels are set to black (white). All other colors are stretched linearly.

CStretch10 Same as CStretch5, but with 10% of all pixels to black/white.

HistEQ Equalize the histogram of the image.

Equalization is used to repair images that have too much contrast or are too light or dark. Equalization attempts to flatten the histogram of the image.

Random This randomly shuffles the color map.

So colors of the original image, that look very similar, change to very distinguishable colors. This is very useful to e.g. verify some color-separation processes.

Show MetaInfo This menu entry appears only if images are displayed in the window. "Show MetaInfo" (de-)activates the display of some additional information of the original image as passed to the render functions, for example the color-space of the input image, the size of the image in pixels and the color histogram of the image.

Font This menu entry appears only if text may be displayed in the window. Selects the font and thus the size of any text, which get rendered inside the image window (for example the text shown if the meta info is activated).

Font Shadows This menu entry appears only if text may be displayed in the window. Shows the rendered text with shadowed fonts. Helps sometimes to make the text more visible.

5.2.6 Panning/Zooming the image windows

If a new image window is opened the rendered image is displayed in such a zoom level that it is always completely visible. By pressing the shift key and the middle mouse button inside the window you can zoom into the image, by pressing the ctrl key and the middle mouse button you can zoom out. Alt plus the middle mouse button resets to the fit to window displaying mode. If fit to window is *not* active you can hold the middle mouse button and move around inside the window to move the clipping frame for the "world coordinate" picture, i.e. to pan inside the image.

Attention:
If your window manager is already using one or more of that combinations for itself, then that signal can not reach iceWing! Then you have to change your window manager setting: Disable that mouse signal there, so it will no longer get caught by your window manager and can reach iceWing.

5.3 The GUI widgets

(TODO: FULL list, or only short summary?) Widget “List” can have a context menu. And it can be reordered via drag’n drop...

in Goptions.h opts_xxx_create()

Programming guide

6 iceWing Files

6.1 Filesystem hierarchy

Let's see, what the installation consists of. Below the installation prefix, for example "/usr/local", you have this structure-content:

bin icewing (the executable itself)

icewing-config

A shell script similar to e.g. gtk-config which makes compiling of own plugins easy. It generates compiler-flags, extracts system-paths and more.

include - All the headers for your own plugin programming

iceWing - headers from the iceWing system

iwPlugins - plugin headers

If your plugin pubishes new structures to be used by other plugins, you probably want to put them here.

lib/iceWing Here is the default place for iceWing to search for plugin libraries. Normally you give iceWing by command line parameter "-l" library names to load. If iceWing is not able to load them, it automatically tries to load them from this directory.

man - The manpage of iceWing

share iceWing - Place where plugins can store additional data files.

E.g. the plugin "Face" has here it's config-file placed, and the polynom-classificator plugin stores it's data here, too.

log/icewing.log - the installation log file

6.2 Headerfiles overview

TODO

7 iceWing – Ein CASE Tool

Currently only available in German.

Originally taken from [Löm04, Anhang B]. Since then updated according to changes in iceWing

Bei der Entwicklung von Software im wissenschaftlichen Bereich gibt es einige sehr aufwendige Aufgaben, die jedes mal wieder anfallen, aber nicht direkt zu der eigentlichen Aufgabe gehören. Sehr häufig benötigt man eine Möglichkeit, möglichst einfach und schnell Parameter eines Algorithmus zu beeinflussen. Ähnlich wichtig ist auch die Möglichkeit der einfachen visuellen Darstellung von beliebigen Daten. Die Daten müssen jederzeit leicht inspizierbar und abspeicherbar sein. Bei größeren integrierten Systemen ist es von Vorteil, wenn Einzelkomponenten getrennt voneinander entwickelt werden können und anschließend mit möglichst wenig Geschwindigkeitsverlust untereinander flexibel kommunizieren können.

Eine ergonomisch ausgefeilte Oberfläche, die auch für mit einem Programm nicht vertraute Person leicht bedienbar ist, wird im wissenschaftlichen Bereich dagegen meist nicht benötigt. Wichtig ist meist, daß ein kleines mit der zu lösenden Aufgabe gut vertrautes Team einfach ihren speziellen Algorithmus entwickeln und optimieren kann. Dieses Team von Spezialisten muß die Oberfläche leicht bedienen können. Dabei soll meist kein fertiges Programm für einen Endanwender entstehen. Verschiedene Systeme bieten auf diesen Gebieten schon Funktionalität an. Das kommerzielle MATLAB hat beispielsweise umfangreiche Möglichkeiten zur Visualisierung und zur Oberflächengenerierung, die durch die MATLAB eigene Scriptsprache einfach benutzt werden können [The03]. Die freie Scriptsprache Tcl mit ihrem graphischen Toolkit Tk bietet ebenfalls sehr einfache Möglichkeiten, eine Oberfläche zu generieren [Ous94].

Bestehende Systeme sind allerdings meist nicht auf die speziellen Belange bei der Entwicklung von wissenschaftlicher Software optimiert. Daher wurde iceWing, *the Integrated Communication Environment Which Is Not Gesten*[1], neu entwickelt. iceWing ist eine graphische Shell, die für zur Laufzeit nachladbare Plugins die oben angeführten Funktionalitäten auf einfache Weise zur Verfügung stellt. In den nächsten Abschnitten wird iceWing nun näher vorgestellt. Dazu wird zuerst ein Überblick über die Erstellung von Plugins gegeben. Die nachfolgenden Kapitel gehen dann auf verschiedene Details exemplarisch näher ein. Somit ist die folgende Beschreibung kein komplettes Referenzhandbuch mit einer Darstellung aller Funktionen und Typen von iceWing.

[1] Dies ist ein Hinweis auf ein älteres Programm, den Vorgänger von iceWing.

Weitere hier nicht erwähnte Punkte finden sich in den Headern und den Beispielplugins von iceWing.

7.1 Überblick

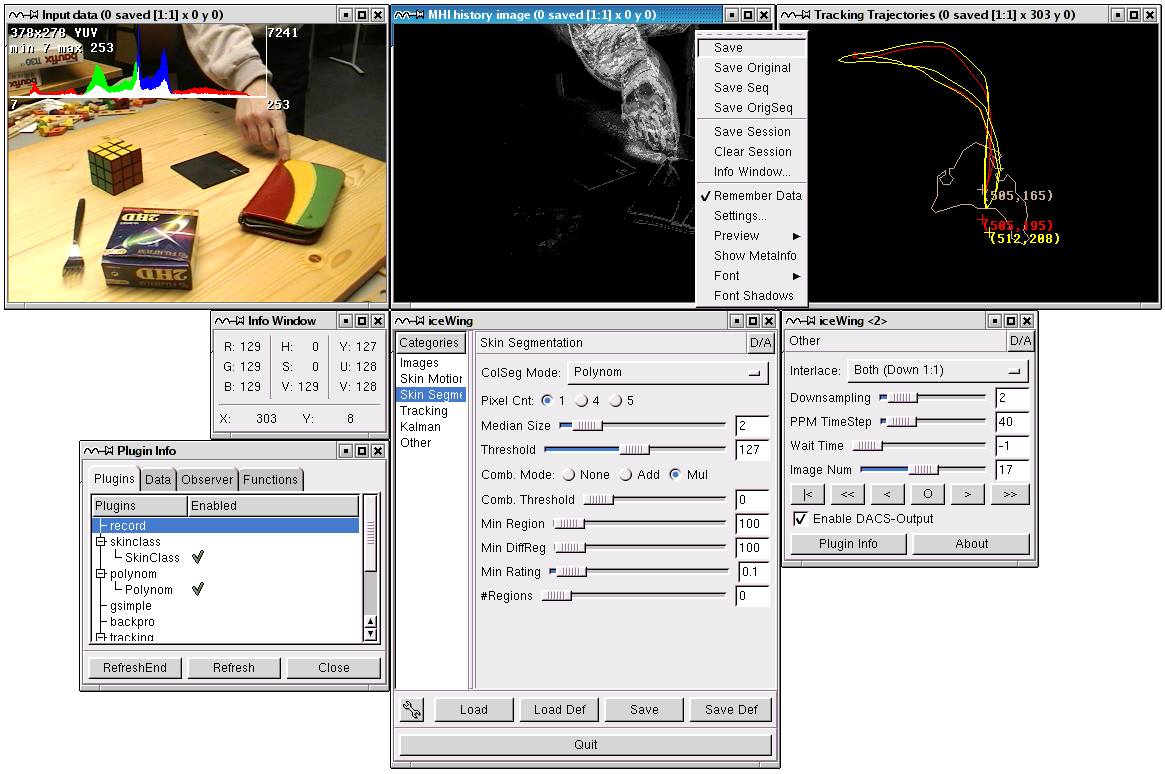


Abbildung 7.1: Eine typische Sitzung mit iceWing. Verschiedene Zwischenergebnisse sind visualisiert und können begutachtet werden. Auf verschiedene Parameter kann interaktiv Einfluß genommen werden.

iceWing ist ein in C geschriebenes Programm, das als shared Libraries realisierte Plugins zur Laufzeit nachladen kann. Getestet wurde es unter i386 Linux mit dem GCC Version 2.95 bis Version 3.4[2] und auf Alphas unter OSF 4.0f mit den Compilern DEC C Version 5.9 und GCC Version 3.1. Für graphische Ausgaben und die Oberflächengenerierung wird zwingend das GTK Toolkit[3] in der Version 1.2 benötigt. Durch andere externe Libraries kann der Funktionsumfang beispielsweise bezüglich unterstützter Graphikformate gesteigert werden. Abbildung 7.1 zeigt eine typische

[2]siehe http://gcc.gnu.org/
[3]siehe http://www.gtk.org/

Sitzung mit iceWing, bei der Hände in einer Bildsequenz segmentiert und verfolgt werden.

Plugins in iceWing

Das Hauptprogramm iceWing stellt nur eine initiale Oberfläche und verschiedene Hilfsroutinen zur Verfügung. Die eigentliche Funktionalität wird durch Plugins realisiert. Plugins für iceWing müssen das in Abbildung 7.2 dargestellte Interface implementieren. An Hand eines minimalen Plugins ist dies in Abbildung 7.3 dargestellt. Wird ein Plugin in Form einer shared Library geladen, wird zuerst eine Funktion mit Namen `plug_get_info()` aufgerufen. Dies ist zugleich der einzig fest vorgegebene Einsprungpunkt in die Library. Die Funktion dient der Instantiierung einer neuen Instanz eines Plugins, sie hat damit die Aufgabe einer Factory-Funktion. Die Funktion gibt einen Zeiger auf eine gefüllte Struktur vom Typ `plugDefinition` zurück.

```
typedef struct plugDefinition {
    char *name;
    plugType type;
    void (*init) (struct plugDefinition *plug,
                  grabParameter *para, int argc, char **argv);
    int  (*init_options) (struct plugDefinition *plug);
    void (*cleanup) (struct plugDefinition *plug);
    BOOL (*process) (struct plugDefinition *plug,
                     char *id, struct plugData *data);
} plugDefinition;
```

Abbildung 7.2: Die Struktur plugDefinition, die Plugins implementieren müssen.

Die Struktur `plugDefinition` enthält alle Informationen, die iceWing über ein neues Plugin haben muß. Die Funktionszeiger `init()`, `init_options()`, `cleanup()` und `process()` definieren Einsprungpunkte für die Instanz des Plugins. `init()` dient der allgemeinen Initialisierung einer Instanz. Hier werden beispielsweise auch Kommandozeilenargumente verarbeitet. In `init_options()` wird das graphische Benutzerinterface initialisiert. Genaueres hierzu findet sich in Abschnitt 7.3. `cleanup()` wird am Programmende aufgerufen, um die Freigabe von Ressourcen zu ermöglichen. `process()` wird schließlich aufgerufen, wenn die eigentliche Funktionalität des Plugins ausgeführt werden soll. Wann dies der Fall ist, kann mit Hilfe der Funktionen zur Kommunikation zwischen Plugins definiert werden. Näheres hierzu findet sich in Abschnitt 7.2. Die Variable `name` gibt den Namen der Instanz an. Da der Name zur Identifikation der Instanz dient, muß er eindeutig sein. `type` sollte aus Kompatibilitätsgründen immer auf `PLUG_IMAGE` gesetzt werden.

Da das Plugin aus Abbildung 7.3 immer einen festen Namen zurückgibt, läßt es sich nur einmal instantiieren. Um dies zu ändern, muß die Struktur vom Typ

```
#include "main/plugin.h"

static void min_init (plugDefinition *plug,
                      grabParameter *para, int argc, char **argv)
{
    ...
}

...

static plugDefinition plug_min = {
    "Min",
    PLUG_IMAGE,
    min_init,
    min_init_options,
    min_cleanup,
    min_process
};

plugDefinition *plug_get_info (int cnt, BOOL *append)
{
    *append = TRUE;
    return &plug_min;
}
```

Abbildung 7.3: Das Grundgerüst eines minimalen Plugins.

`plugDefinition` dynamisch alloziert werden und der dort enthaltene Name muß eindeutig gemacht werden. Dies kann durch die Integration der Instanznummer `cnt` in den Namen geschehen. `cnt` gibt an, wie häufig das Plugin schon instantiiert werden sollte. `plug_get_info()` ändert sich damit zu:

```
*append = TRUE;
plugDefinition *def = calloc (1, sizeof(plugDefinition));
*def = plug_min;
def->name = g_strdup_printf ("Min%d", cnt);
return def;
```

Damit kann das Plugin nun beliebig häufig instantiiert werden.

Sollen Plugins in C++ geschrieben werden, kann dies entsprechend dem bisher erläuterten Vorgehen geschehen. Alternativ steht aber auch die in Abbildung 7.4 dargestellte C++-Klasse zur Verfügung. Durch Ableiten von dieser Klasse ist ebenfalls die Erzeugung einer Plugin-Instanz möglich. Die Factory-Funktion `plug_get_info()` wird in diesem Fall zu

```
*append = TRUE;
ICEWING::Plugin* newPlugin =
```

```
            new ICEWING::MinPlugin (g_strdup_printf("C++Min%d", cnt));
    return newPlugin;
```

wobei `ICEWING::MinPlugin` eine von `ICEWING::Plugin` abgeleitete Klasse ist.

```
namespace ICEWING {
    class Plugin : public plugDefinition {
    public:
        Plugin (char *name);
        virtual ~Plugin() {};

        virtual void Init (grabParameter *para, int argc, char **argv) = 0;
        virtual int  InitOptions () = 0;
        virtual bool Process (char *ident, plugData *data) = 0;
    };
}
```

Abbildung 7.4: Die Klasse Plugin, die eine der Sprache C++ gerechte Erstellung von Plugins in C++ erlaubt.

7.2 Kommunikation zwischen Plugins

Innerhalb einer Hauptschleife ruft iceWing wiederholt die verschiedenen Plugins auf. Die Reihenfolge des Aufrufs können dabei die Plugins bestimmen. Hierfür und für die weitergehende Kommunikation zwischen Plugins stehen verschiedene Möglichkeiten zur Verfügung. Plugins können Daten untereinander austauschen. Sie können das Ablegen von Daten beobachten und sie können Funktionen zur Verfügung stellen und zur Verfügung gestellte Funktionen aufrufen. Details zu diesen Kommunikationsmöglichkeiten werden nun vorgestellt.

Austausch von Daten

Daten in iceWing bestehen immer aus einem String, der als Identifier dient, einem Referenzzähler und den Daten in Form eines Zeigers. Abgelegt und für andere Plugins zur Verfügung gestellt werden sie mit der Funktion

```
typedef void (*plugDataDestroyFunc) (void *data);

void plug_data_set (plugDefinition *plug, char *ident, void *data,
                    plugDataDestroyFunc destroy);
```

Die Funktion `destroy()` wird aufgerufen, wenn der Referenzzähler der Daten am Ende eines Hauptschleifendurchlaufs auf Null gesunken ist. Für den Zugriff auf abgelegte Daten gibt es die Funktion

```
typedef struct plugData {
    plugDefinition *plug;   /* Plugin, welches die Daten abgelegt hat */
    char *ident;            /* ident, mit dem die Daten abgelegt wurden */
    void *data;             /* Die abgelegten Daten */
} plugData;

plugData* plug_data_get (char *ident, plugData *data);
```

und Varianten für den vereinfachten Aufruf. Mit `plug_data_set()` lassen sich unter einem Identifier verschiedene Daten ablegen. Mit Hilfe des Parameters `data` bei `plug_data_get()` ist es möglich, auf diese nacheinander zuzugreifen. Ist dieser `NULL`, werden die ersten unter dem spezifizierten Identifier abgelegten Daten zurückgegeben. Bei nachfolgenden Aufrufen mit dem jeweils zurückgegebenen Datenzeiger werden die jeweils nächsten Daten zurückgegeben. Bei jedem Aufruf von `plug_data_get()` erhöht sich jeweils der Referenzzähler der Daten. Diese Referenzen können durch die Funktion

```
void plug_data_unget (plugData *data);
```

wieder freigegeben werden. Für jeden erfolgreichen Aufruf von `plug_data_get()` ist damit ein Aufruf von `plug_data_unget()` nötig, damit einmal abgelegte Daten durch den Aufruf der Funktion `destroy()` wieder freigegeben werden können.

Beobachten von Daten

Mit der bisher beschriebenen Funktionalität von iceWing ist noch nicht klar, wann die Funktion `process()` der Plugins aufgerufen wird. Dies wird durch das Beobachten von Daten festgelegt. Mit der Funktion

```
void plug_observ_data (plugDefinition *plug, char *ident);
```

kann ein Plugin das Ablegen von Daten mit dem Identifier `ident` beobachten. Werden *neue* Daten unter diesem Identifier abgelegt, wird die Funktion `process()` des Plugins aufgerufen. Neu sind diejenigen Daten, die seit dem Start des aktuellen Hauptschleifendurchlaufs abgelegt wurden. Am Anfang eines Hauptschleifendurchlaufs legt iceWing Pseudodaten unter dem Identifier `"start"` ab. Andere Plugins können diese Daten beobachten und werden damit jeweils am Anfang der Hauptschleife aufgerufen. Diese Plugins können nun ihrerseits Daten mit anderen Identifiern ablegen und so den Aufruf von weiteren Plugins initiieren. Sind keine Plugins mehr vorhanden, die sich für neu abgelegte Daten angemeldet haben, beginnt die Hautschleife durch Ablegen von Pseudodaten unter dem Identifier `"start"` von neuem. Die Plugins werden dabei sequentiell aufgerufen. Erst nachdem die Funktion `process()` eines Plugins beendet wurde, wird die `process()`-Funktion des nächsten Plugins aufgerufen. Wurden mehrere Daten unter dem gleichen Identifier abgelegt, werden die Plugins trotzdem nur einmal aufgerufen. Sollen die Plugins alle Daten verarbeiten, müssen sie diese sequentiell mit Hilfe von `plug_data_get()` abholen. Durch den Aufruf von

```
void plug_observ_data_remove (plugDefinition *plug, char *ident);
```

kann ein Plugin das Beobachten von Daten schließlich wieder beenden.

Austausch von Funktionen

Das im letzten Abschnitt beschriebene Kommunikationsverfahren ist rein Datengetrieben. Zusätzlich gibt es aber auch noch eine Möglichkeit, Funktionen eines Plugins für andere Plugins zur Verfügung zu stellen. Dies geschieht mit Hilfe der Funktion

```
typedef void (*plugFunc) ();

void plug_function_register (plugDefinition *plug,
                             char *ident, plugFunc func);
```

Durch den Identifier `ident` kann auf die registrierte Funktion wieder zugegriffen werden. Dies geschieht mit Hilfe der Funktion

```
typedef struct plugDataFunc {
    plugDefinition *plug;   /* Plugin, welches die Funktion abgelegt hat */
    char *ident;            /* ident, mit dem die Funktion abgelegt wurde */
    plugFunc func;          /* Die registrierte Funktion */
} plugDataFunc;

plugDataFunc* plug_function_get (char *ident, plugDataFunc *func);
```

Ähnlich wie bei dem Ablegen von Daten können auch mehrere Funktionen unter einem Identifier abgelegt werden. Durch Setzen von `func` auf `NULL` kann auf die erste unter einem Identifier abgelegte Funktion zugegriffen werden. Bei nachfolgenden Aufrufen mit dem jeweils zurückgegebenen Funktionszeiger werden die jeweils nächsten Funktionen zurückgegeben. Durch die Funktion

```
void plug_function_unregister (plugDefinition *plug,
                               char *ident, plugFunc func);
```

kann eine zur Verfügung gestellte Funktion wieder zurückgezogen werden.

7.3 Graphische Funktionalitäten

Die graphischen Funktionalitäten von iceWing lassen sich in drei Gruppen einteilen. Zu der ersten Gruppe zählen Funktionen zur Erstellung eines Benutzerinterfaces bestehend aus Widgets. Die zweite Gruppe von Funktionen beschäftigt sich mit der Darstellung von verschiedenen Daten. Zusätzlich gibt es noch verschiedene Funktionen, die sich diesen Gruppen nicht direkt zuordnen lassen. Die Erstellung des graphischen Interfaces findet hauptsächlich in der Funktion `init_options()` der verschiedenen Plugins statt. Alle in diesem Kapitel aufgeführten Funktionen können aber auch zu jedem späteren Zeitpunkt aufgerufen werden. In den folgenden Abschnitten werden die verschiedenen Funktionalitäten nun näher erläutert.

7.3.1 Erstellung eines Benutzerinterfaces

Alle Widgets, die sich mit Funktionen von iceWing erstellen lassen, folgen der gleichen Philosophie. Bei der Erstellung eines Widgets wird jeweils die Adresse einer Variablen angegeben. Diese Variable wird im Hintergrund geändert, ohne daß das Plugin eingreifen muß, dies anderseits aber auch nicht kann. Daneben ist das Layout, in dem die Widgets erstellt werden, weitestgehend vorgegeben. Durch diese Einschränkung der Funktionalität ergeben sich zwei Vorteile:

- Die Erstellung und Verwaltung von Widgets wird sehr einfach.
- Es gibt die Möglichkeit des automatischen Ladens, Speichern und extern über DACS gesteuerten Setzens der Werte von Widgets. Diese Funktionalität ist vollständig unabhängig von jedweder Unterstützung seitens der Plugins.

Abbildung 7.5 zeigt eine Übersicht aller Widgets, die sich mit iceWing erstellen lassen.

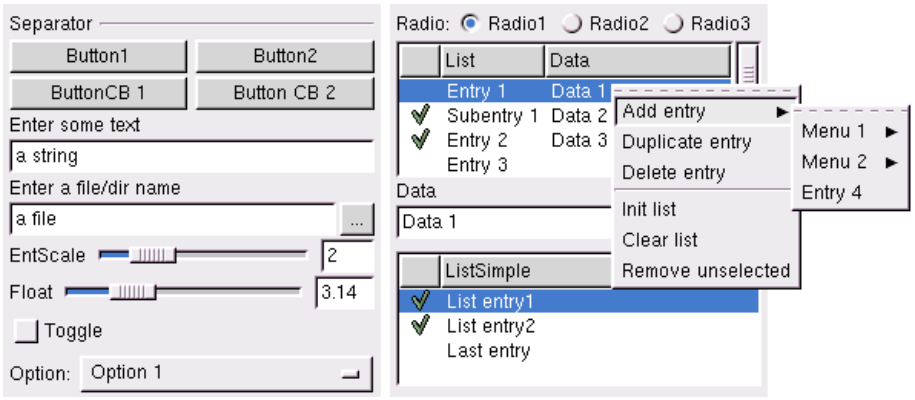


Abbildung 7.5: Alle Widgets, die iceWing für die Oberflächengenerierung zur Verfügung stellt.

Graphisches Benutzerinterface

Widgets lassen sich in iceWing auf beliebig vielen Seiten im iceWing-Hauptfenster, im Kontextmenü von Anzeigefenstern und im "Settings"-Fenster von Anzeigefenstern erstellen. In der Abbildung 7.1 sind diese verschiedenen Stellen zu sehen. Eine neue Seite im iceWing-Hauptfenster kann mittels der Funktion

```
int opts_page_append (char *title);
```

erstellt werden. `title` gibt den Namen an, der in der Liste "Categories" angezeigt wird. Zurück gegeben wird ein Index, der bei der Erstellung von Widgets angegeben werden muß. Den Index der "Settings"-Fenster von Anzeigefenstern, der das Erstellen von Widgets in diesen Fenstern erlaubt, erhält man durch die Funktion

```
int prev_get_page (prevBuffer *b);
```

Widgets lassen sich mit Funktionen erstellen, die jeweils nach dem Schema `opts_<widgetname>_create()` benannt sind. Für vier verschiedene Widgettypen sind dies beispielsweise

```
void opts_separator_create (long page, char *title);
void opts_button_create (long page, char *title, char *ttip, gint *value);
void opts_entscale_create (long page, char *title, char *ttip,
                           gint *value, gint left, gint right);
void opts_float_create (long page, char *title, char *ttip,
                        gfloat *value, gfloat left, gfloat right);
```

Die Parameterliste der Funktionen ist jeweils einheitlich aufgebaut. `page` gibt an, auf welcher Seite das Widget erscheinen soll. Neue Widgets werden jeweils unten auf der angegebenen Seite angefügt. `title` gibt den Namen des Widgets an. In Abbildung 7.5 war er beispielsweise `"Enter some text"` für das String Widget und `"EntScale"` für den Integer-Slider. Dieser Name muß in Verbindung mit dem Seitennamen programmweit eindeutig sein, da er auch als Identifier für das Widget eingesetzt wird. Der Identifier ist dabei `"pagetitle.widgettitle"`. Durch diesen Identifier läßt sich das Widget auch nachträglich ansprechen. `ttip` spezifiziert den Tooltip des Widgets. Durch `value` wird die Adresse einer Variablen angegeben. Die Variable wird von iceWing in einem eigenen Thread im Hintergrund modifiziert, ohne daß das Plugin eingreifen muß. Die restlichen Parameter spezifizieren jeweils die erlaubten Werte der Variablen.

Zusätzlich ist es möglich, den Wert eines Widgets nachträglich zu setzen und ein Widget wieder zu löschen. Dafür dienen folgende zwei Funktionen:

```
long opts_value_set (char *title, void *value);
gboolean opts_widget_remove (char *title);
```

`title` ist dabei der Identifier eines Widgets. In `value` wird, im Falle eines Integers, der zu setzende Wert übergeben. Ansonsten ist es ein Zeiger auf den zu setzenden Wert. Soll beispielsweise das Widget `"EntScale"` aus Abbildung 7.5 auf der Seite `"demo"` auf den Wert 3 gesetzt werden, kann dies durch

```
opts_value_set ("demo.EntScale", GINT_TO_POINTER(3));
```

geschehen. Beim Setzen des Widget `"Float"` geschieht dies hingegen durch

```
float newval = 3.0;
opts_value_set ("demo.Float", &newval);
```

Nichtgraphisches Interface

In diese automatische Behandlung von Variablen zum Laden und Speichern lassen sich auch Variablen einfügen, denen kein Widget zugeordnet ist. Dies wird durch die Funktionen

```
typedef enum {
    OPTS_BOOL, OPTS_INT, OPTS_LONG, OPTS_FLOAT, OPTS_DOUBLE, OPTS_STRING
} optsType;

typedef void (*optsSetFunc) (void *value, void *new_value, void *data);

void opts_variable_add (char *title, optsSetFunc setval, void *data,
                        optsType type, void *value);
void opts_varstring_add (char *title, optsSetFunc setval, void *data,
                         void *value, int length);
```

ermöglicht. `title` entspricht der `titel`-Variablen der Widget-Funktionen. `value` gibt die Adresse der zu ladenden/speichernden Variablen an. Durch `type` wird deren Typ bekanntgegeben. Ist `func` auf `NULL` gesetzt, wird die Variable nicht nur automatisch im Hintergrund gespeichert, sondern auch geladen und gesetzt. Ansonsten wird, sobald die Variable modifiziert werden soll, die Funktion `setval` mit `data` als einem zusätzlichen Argument aufgerufen. Diese Funktion ist dann für das Modifizieren der Variablen verantwortlich. Durch die Benutzung von `opts_varstring_add()` kann für Strings zusätzlich eine Maximallänge des Strings angegeben werden, die beim Setzen des Strings nicht überschritten wird.

Plugin-Support

Gewisse durch Widgets steuerbare standardisierte Funktionen sind für viele Plugins interessant. Zusätzlich benötigen die meisten Plugins mindestens eine Seite, um dort Widgets zu plazieren. Um dies zu vereinfachen, gibt es folgende Funktion, die eine neue Seite mit zwei speziellen Widgets anlegt:

```
typedef enum {
    PLUG_PAGE_NOPLUG        = 1 << 0
} plugPageFlags;

int plug_add_default_page (plugDefinition *plugDef, char *suffix,
                           plugPageFlags flags);
```

Die Funktion legt eine neue Seite im iceWing-Hauptfenster unter dem Namen '`plugDef->name" "suffix`' an und gibt deren Index zurück. Zusätzlich werden bis zu zwei Widgets angelegt, deren Funktionalität vollständig von iceWing realisiert wird. Das erste Widget ermöglicht das Aktivieren/Deaktivieren des Aufrufs der `process()` Funktion des Plugins.

Das zweite Widget wird angelegt, falls nicht `PLUG_PAGE_NOPLUG` in `flags` angegeben wurde. Über dieses zweite Widget kann gesteuert werden, welche Daten von welchen anderen Plugins an das Plugin `plugDef` weitergereicht werden sollen. Normalerweise wird die Funktion `process()` des Plugins `plugDef` aufgerufen, sobald Daten eines Identifiers von einem beliebigen Plugin vorhanden sind, die von dem Plugin `plugDef`

beobachtet werden. Stehen zu diesem Zeitpunkt mehrere Daten des Identifiers zur Verfügung, wird das Plugin trotzdem nur einmal aufgerufen. Das Plugin kann die zusätzlichen Daten durch `plug_data_get()` abrufen.

Durch den Aufruf von `plug_add_default_page()` wird dieses Verhalten geändert. Ist in diesem Fall im zweiten Widget nichts eingetragen, wird das Plugin für alle Daten von allen Plugins, für das sich das Plugin angemeldet hat, getrennt aufgerufen. Werden im Widget Namen von Plugins eingetragen, wird das Plugin nur für diese Plugins aufgerufen. Stehen Daten von anderen Plugins zur Verfügung, wird das Plugin nicht aufgerufen.

Da iceWing mit diesem Widget zusätzlich weitere Informationen über die Abhängigkeiten der Plugins hat, wird hier gegebenenfalls eine Umsortierung der Aufrufreihenfolge durchgeführt. Beobachten mehrere Plugins den gleichen Identifier `"ident"`, ist normalerweise deren Aufrufreihenfolge nicht spezifiziert. Wird in diesem zweiten Widget aber nun spezifiziert, daß ein Plugin `"ident"` von einem dieser anderen Plugins haben möchte, wird sichergestellt, daß erst dieses andere Plugin aufgerufen wird.

7.3.2 Graphische Anzeige von Daten

iceWing enthält verschiedene Möglichkeiten, Daten zu visualisieren. Plugins können beliebig viele Fenster zur Darstellung von Daten anlegen. Der Benutzer kann diese jederzeit öffnen oder wieder schließen, in ihnen zoomen und scrollen, ihren Inhalt abspeichern oder sich Metainformationen über den Inhalt anzeigen lassen. Bei all diesen Aktionen sind die Plugins nicht involviert. Um die nötigen Zoom- und Redrawaktionen kümmert sich vollständig iceWing. Vektorobjekte wie Linien und Ellipsen werden dabei je nach Zoomstufe neu gezeichnet.

Verwaltung von Fenstern

Neue Fenster können mit der Funktion

```
typedef struct prevBuffer {
    ...
    guchar *buffer;         /* Buffer fuer das zu erzeugende Bild */
    int width, height;      /* Breite, Hoehe des Buffers */
    GtkWidget *window;      /* Fenster fuer den Buffer */
    ...
} prevBuffer;

prevBuffer *prev_new_window (char *title, int width, int height,
                             gboolean gray, gboolean show);
```

angelegt werden. `title` ist einerseits der Name des Fensters und andererseits ein Identifier, der zusammen mit den Namen der mit `opts_page_append()` angelegten Seiten im iceWing-Hauptfenster programmweit eindeutig sein muß. Enthält der Name

einen Punkt (`"."`), werden die Fenster in der "Images"-Liste im Hauptfenster in einer Baumstruktur angezeigt. `width` und `height` geben die initiale Größe des Fensters an. Der Benutzer kann die Fenstergröße nachträglich jederzeit ändern. Durch Angabe von `-1` wird die Standardbreite beziehungsweise Höhe genommen. `gray` gibt an, ob alles in Graustufen oder in Farbe dargestellt werden soll. Durch `show = TRUE` wird das Fenster immer nach dem Anlegen sofort geöffnet. Ansonsten passiert dies erst, wenn der Benutzer das Fenster per Doppelklick in der "Images"-Liste öffnet.

Nennenswert Speicher wird durch ein Fenster erst nach dem ersten Öffnen verbraucht. Alle iceWing-Funktionen zum Zeichnen in einem Fenster testen zuerst, ob es offen ist und kehren sofort zurück, falls dies nicht der Fall ist. Damit ist es aus der Sicht des Ressourcenverbrauchs unproblematisch, viele Fenster anzulegen und Ausgaben in diesen ohne weitere Tests vorzunehmen. Sind Berechnungen für die Ausgabe von Daten nötig, kann ein Test, ob ein Fenster offen ist, aber sinnvoll sein. Dies kann durch `(buffer->window != NULL)` geschehen. Durch

```
void prev_free_window (prevBuffer *b);
```

kann ein per `prev_new_window()` angelegtes Fenster schließlich wieder entfernt werden.

Der Benutzer kann in den Fenstern jederzeit zoomen und scrollen. Zusätzlich können diese Aktionen auch programmgesteuert durch die Funktion

```
void prev_pan_zoom (prevBuffer *b, int x, int y, float zoom);
```

durchgeführt werden. Der Inhalt des Fenster `b` wird an der Stelle `(x,y)` mit einem Zoom-Wert von `zoom` dargestellt. Sind einzelne Werte kleiner Null, werden die entsprechenden alten Werte nicht verändert.

Zum Teil ist es nötig, daß ein Plugin Informationen über Mausaktionen und Tastendrücke in einem Fenster bekommt. Dies wird durch die Funktionen

```
typedef enum {
    PREV_BUTTON_PRESS               = 1 << 0,
    PREV_BUTTON_RELEASE             = 1 << 1,
    PREV_BUTTON_MOTION              = 1 << 2
    PREV_KEY_PRESS                  = 1 << 3,
    PREV_KEY_RELEASE                = 1 << 4
} prevEvent;
typedef void (*prevButtonFunc) (prevBuffer *b, prevEvent signal,
                                int x, int y, void *data);
typedef void (*prevSignalFunc) (prevBuffer *b, prevEventData *event,
                                void *data);

void prev_signal_connect (prevBuffer *b, prevEvent sigset,
                          prevButtonFunc cback, void *data);
void prev_signal_connect2 (prevBuffer *b, prevEvent sigset,
                           prevSignalFunc cback, void *data);
```

ermöglicht. Tritt eins der mittels `sigset` angegebenen Ereignisse ausgelöst mit der linken Maustaste oder einer Taste im Fenster `b` auf, wird die Funktion `cback()` mit `data` als zusätzliches Argument aufgerufen. `cback()` wird zusätzlich mit dem aufgetretenem Signal und weiteren Informationen über das Ereignis aufgerufen. `prev_signal_connect()` erlaubt dabei nur die Verarbeitung von Mausereignissen, `prev_signal_connect2()` sowohl Maus- als auch Tastaturereignisse. Bei allen Mauskoordinaten sind jeweils Modifikationen durch Zoomen und Scrollen herausgerechnet.

Darstellung von graphischen Objekten

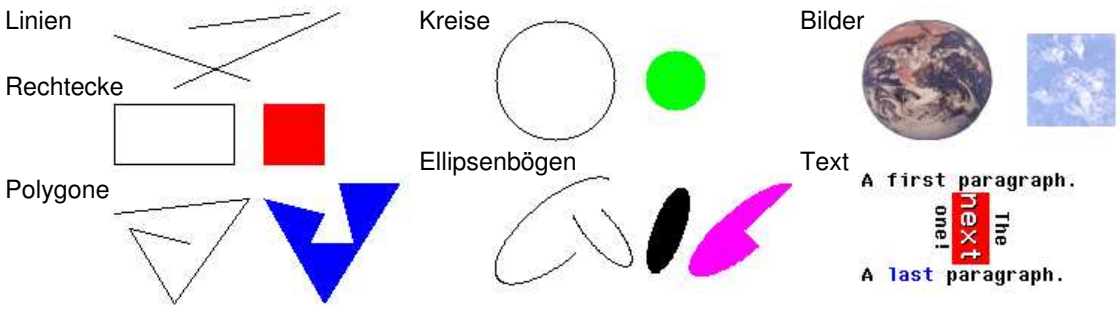


Abbildung 7.6: Alle graphischen Elemente, die iceWing darstellen kann.

iceWing bietet die Möglichkeit, verschiedene Graphikprimitiva in den Fenstern darzustellen. Abbildung 7.6 zeigt eine Übersicht aller in iceWing enthaltener Objekte. Die Darstellung der Objekte ist mehrstufig. In einem ersten optionalen Schritt wird eine Kopie aller Originaldaten angelegt, die für die Darstellung eines Objektes benötigt werden. Damit erhält iceWing die Möglichkeit, das Bild komplett ohne Interaktion mit dem Plugin neu aufzubauen – Scrollen und Zoomen ohne Pluginhilfe wird möglich. Anschließend werden die Objekte entsprechend den aktuellen Scroll- und Zoomwerten in einem Offscreenbuffer dargestellt. Dieser Buffer zeigt exakt den Bildausschnitt, der auch im Fenster zu sehen sein wird, und kann damit für ein Redraw des Fensters eingesetzt werden – wiederum ohne Interaktion mit dem Plugin. In einem finalen Schritt wird der Buffer schließlich in das Fenster übertragen.

Die verschiedenen Objekte können über ein einheitliches Interface verschiedener Funktionen dargestellt werden. Zwei von ihnen sind

```
typedef enum {
    PREV_IMAGE,
    PREV_TEXT,
    PREV_LINE,
    ...
    PREV_NEW = 30
} prevType;
```

```
#define RENDER_THICK    (1<<30)  /* beachte die thickness-Einstellung */
#define RENDER_CLEAR    (1<<31)  /* loesche den Buffer */

void prev_render_data (prevBuffer *b, prevType type, void *data,
                       int disp_mode, int width, int height);
void prev_render_list (prevBuffer *b, prevType type, void *data,
                       int size, int cnt,
                       int disp_mode, int width, int height);
```

Bei allen Renderfunktionen gibt es verschiedene Standardargumente. In dem Fenster `b` wird das Objekt dargestellt. `width` und `height` geben die Gesamtgröße aller Objekte an, die in dem Fenster noch dargestellt werden. Diese beiden Werte werden benötigt, um gegebenenfalls den Zoomfaktor für die Darstellung des Objektes korrekt zu berechnen, falls ein "Fit to window"-Darstellungsmodus für das Fenster gewählt ist. Über `disp_mode` kann die Darstellungsart modifiziert werden. Das Flag `RENDER_CLEAR` löscht vor dem Zeichnen sowohl den Inhalt des kompletten Fensters als auch die internen Kopien aller Parameter aller vorher durchgeführten Zeichenaktionen. `RENDER_THICK` erlaubt die dickere Darstellung von Linien entsprechend einer gesondert gewählten Einstellung. Die restlichen Argumente spezifizieren jeweils das darzustellende Objekt. Bei `prev_render_data()` wird ein beliebiges Objekt dargestellt, bei `prev_render_list()` ein Feld gleicher Objekte. `type` gibt den Typ der Objekte an. `data` ist entweder ein Zeiger auf die Daten eines Objektes oder ein Zeiger auf ein Feld von Objektdaten. Im zweiten Fall wird über `cnt` die Länge des Feldes und über `size` die Größe eines Feldelementes angegeben.

Werte für darzustellende Objekte werden über Strukturen angegeben. Für Linien ist dies beispielsweise

```
typedef struct {
    imgColtab ctab;
    int r, g, b;
    int x1, y1, x2, y2;
} prevDataLine;
```

Die Strukturen der restlichen vorhandenen Objekte sind vergleichbar aufgebaut. Über `ctab` wird angegeben, wie `r`, `g` und `b` interpretiert werden sollen. Durch das Setzen von `ctab` auf `IMG_RGB` werden sie beispielsweise als Punkt im RGB-Farbraum interpretiert, durch `IMG_YUV` als Punkt im YUV-Farbraum und durch einen Zeiger auf eine Farbtabelle wird `r` als Index in diese Farbtabelle interpretiert. Die Angabe von `-1` für `r`, `g` oder `b` hat eine Sonderbedeutung. In diesem Fall wird der entsprechende Farbkanal beim Darstellen des Objektes nicht geändert. `x1`, `y1`, `x2` und `y2` sind schließlich die Koordinaten der Endpunkte der Linie. Um ein subpixelgenaues Darstellen von Objekten zu ermöglichen, gibt es von den Strukturen für Vektorobjekte jeweils Varianten mit `float`-Datentypen, beispielsweise `prevDataLineF` für Linien. Zur Darstellung gibt es entsprechende `prevType`-Werte, für Linien beispielsweise `PREV_LINE_F`.

Um den Aufruf zu vereinfachen gibt es für `prev_render_list()` je einen Wrapper pro Objekttyp. Felder von Linien oder Bildern lassen sich beispielsweise auch durch

```
void prev_render_lines (prevBuffer *b, prevDataLine *lines, int cnt,
                        int disp_mode, int width, int height);
void prev_render_imgs (prevBuffer *b, prevDataImage *imgs, int cnt,
                       int disp_mode, int width, int height);
```

darstellen. Bei der Bildverarbeitung muß sehr häufig ein Bild dargestellt werden. Für den häufigen Fall von 8 Bit Bildern gibt es dafür

```
void prev_render (prevBuffer *b, guchar **planes,
                  int width, int height, imgColtab ctab);
```

womit in diesem Fall keine Struktur gefüllt werden muß. Diese Funktion löscht zusätzlich durch Setzen von `RENDER_CLEAR` vor der Darstellung des Bildes das komplette Fenster. Sollen andere Bildtypen dargestellt werden, iceWing unterstützt auch Bildtiefen von 16 Bit, 32 Bit, Float und Double, oder wird anderweitig mehr Flexibilität benötigt, müssen entweder `prev_render_imgs()` oder die allgemeinen Renderfunktionen benutzt werden.

Für die vereinfachte Textausgabe gibt es zusätzlich die Funktion

```
void prev_render_text (prevBuffer *b, int disp_mode, int width, int height,
                       int x, int y, char *format, ...);
```

Diese Funktion führt intern ein `sprintf()` aus und gibt den erhaltenen String dann aus. Im String können zusätzlich Formatierungsanweisungen enthalten sein, um Farbe, Art und Ausrichtung zu modifizieren. Durch ein eingebettetes `<fg="255 0 0" bg="0 0 0" font=big>` kann beispielsweise auf einen großen roten Font auf schwarzem Grund umgestellt werden. Weitere Details zu den möglichen Formatierungsanweisungen finden sich im Header "Grender.h".

Die Darstellung von Objekten läßt sich durch zwei weitere Funktionen beeinflussen:

```
void prev_set_bg_color (prevBuffer *buf, uchar r, uchar g, uchar b);
void prev_set_thickness (prevBuffer *buf, int thickness);
```

`prev_set_bg_color()` gibt an, mit welcher Farbe das Fenster bei Auftreten von `RENDER_CLEAR` gelöscht werden soll. Durch `prev_set_thickness()` werden Linien in der angegebenen Dicke gezeichnet, falls `RENDER_THICK` bei der Ausgabe von Objekten mit angegeben war.

Alle bisher beschriebenen Funktionen zum Darstellen von Objekten zeichnen die Objekte inkrementell in einen einem Fenster zugeordneten Offscreenbuffer. Dieser läßt sich in einem finalen Schritt mittels

```
void prev_draw_buffer (prevBuffer *b);
```

in dem Fenster darstellen. Zusammenfassend sind somit folgende Schritte nötig, um mittels iceWing etwas in einem Fenster darzustellen:

Anlegen eines neuen Fensters `b` mittels `prev_new_window()`. Dies ist initial einmal nötig. Eine gute Stelle hierfür ist die Funktion `init_options()` des Plugins.
LOOP
Darstellen beliebiger Objekte in dem Offscreenbuffer des Fensters `b` durch mehrfaches Aufrufen von `prev_render_xxx()`-Funktionen. Bei dem ersten Aufruf sollte `RENDER_CLEAR` gesetzt sein, um das Fenster zu löschen.
Darstellen des Offscreenbuffers im Fenster `b` durch Aufruf der Funktion `prev_draw_buffer()`.

Neben diesem hier beschriebenen Interface mit den `prev_render_xxx()`-Funktionen zur Darstellung von Objekten gibt es auch noch ein Interface bestehend aus `prev_drawXxx()`-Funktionen. Dies ist ein Interface niedriger Abstraktionsrate, welches beispielsweise die Scrollposition oder die Zoomeinstellung ignoriert. Weitere Details hierzu finden sich in den entsprechenden Headern.

7.3.3 Weitere graphische Funktionalitäten

Muß die Programmausführung beendet werden, sollte dies niemals mit der normalen ANSI-Funktion `exit()` geschehen. Für das sofortige Beenden des Programms sollte immer die Funktion

```
void gui_exit (int status);
```

benutzt werden. Der direkte Aufruf von `exit()` kann sowohl zu einer "Segmentation violation" als auch zum Stehenbleiben des Programms führen. Da verschiedene graphische Funktionalitäten mit Hilfe eines eigenen Threads realisiert sind, muß beim Beenden eine korrekte Synchronisierung mit diesem Thread stattfinden. `gui_exit()` stellt dies sicher.

Bilder werden in iceWing durch die Struktur

```
typedef enum {
    IMG_8U, IMG_16U, IMG_32S, IMG_FLOAT, IMG_DOUBLE
} imgType;

typedef struct imgImage {
    guchar *data[3];    /* Die eigentlichen Bilddaten */
    int planes;         /* Anzahl der Farbebenen von data */
    imgType type;       /* Typ von data */
    int width, height;  /* Breite, Hoehe des Bildes in Pixel */
    int rowstride;      /* Abstand zweier Linien in Bytes */
                        /*   >0: Farbbilder sind interleaved in data[0] */
    imgColtab ctab;     /* Farbraum von data */
} imgImage;
```

verwaltet. Die Bilddaten können hier in verschiedenen Typen und in verschiedenen Anordnungen abgelegt sein. Durch `imgType` wird der Typ von `data` spezifiziert – von 8 Bit unsigned bis double. Die Anordnung der Daten kann sowohl *planed* als auch *interleaved* sein. Bei einer planed Anordnung sind die einzelnen Farbebenen des Bildes getrennt voneinander in `data[0]`, `data[1]` und `data[2]` angegeben. Bei interleaved enthält `data[0]` die einzelnen Farbwerte für jedes Pixel direkt hintereinander. Für `planes=3` im RGB-Farbraum kommt beispielsweise zuerst der Rotwert von Pixel 1, dann der Grünwert und der Blauwert. Anschließend kommen diese drei Werte für Pixel 2 bis schließlich die drei Werte für Pixel `width*heigth` in `data[0]` abgelegt sind. Bilder aller dieser Typen und Anordnungen können sowohl angelegt, freigegeben, geladen, gespeichert als auch angezeigt werden.

Für die Verwaltung von Bildern gibt es verschiedene Funktionen. Einige wichtige sind

```
imgImage* img_new (void);
imgImage* img_new_alloc (int width, int height, int planes, imgType type);
void      img_free (imgImage *img, imgFree what);
imgImage* img_load (char *fname, imgStatus *status);
imgStatus img_save_format (imgImage *img, imgFormat format,
                           char *fname, imgFileData *data);
```

Weitere Details zu diesen Funktionen sowie verschiedene andere Funktionen zur Verwaltung von Bildern finden sich im Header “Gimage.h”.

7.4 Weitere Funktionalitäten

Neben den bisher beschriebenen Funktionalitäten stehen weitere sowohl im graphischen Bereich als in anderen Bereichen zur Verfügung. In diesem Abschnitt wird eine Auswahl von ihnen näher vorgestellt. Daneben gibt es andere, die nicht genauer vorgestellt werden. Im Header “output.h” finden sich beispielsweise Funktionen zur Vereinfachung der Ausgabe von Daten über DACS. Es stehen Funktionen zur Ausgabe von Bildern, zur Ausgabe von Statusmeldungen und zur Ausgabe von allgemeinen Daten über Streams zur Verfügung. Weiterhin können Funktionen über DACS zur Verfügung gestellt werden. Die Anmeldung bei DACS, die Fehlerbehandlung und die Generierung eindeutiger Stream- und Funktionsnamen werden jeweils von diesen Funktionen übernommen.

Weitere Beispiele für weitere Funktionen sind das “session management” oder Funktionen für das Registrieren eines neuen Graphikprimitives, um diesen mit den allgemeinen Renderfunktionen darzustellen. Weitere Details zu diesen wie auch zu den bisher vorgestellten Funktionen finden sich in den verschiedenen Headern von iceWing.

Das Plugin “grab”

Ein zentrales Plugin ist das in iceWing eingebaute Plugin "grab". Es ermöglicht das Einlesen und Weiterreichen einer Folge von Bildern. Die Bilder können von einem Grabber (angesteuert durch Video4Linux 2 oder FireWire unter Linux und MME auf Alphas), von DACS im `Bild_t`-Format oder aus Dateien verschiedener Pixelformate eingelesen werden. Sobald das Plugin von iceWing aufgerufen wird, liest es das nächste Bild ein und stellt es mit Hilfe von `plug_data_set()` unter dem Ident `"image"` für andere Plugins zur Verfügung. Als Format wird dabei das folgende `grabImageData` benutzt:

```
typedef struct grabImageData {
    imgImage *img;        /* Das eigentliche Bild */
    struct timeval time;  /* Zeitpunkt des Grabbens */
    int img_number;       /* Fortlaufende Nummer */
    char *fname;          /* Bild aus einer Datei? -> Name der Datei */
    int downw, downh;     /* Faktor, um den das Bild verkleinert wurde */
} grabImageData;
```

Zusätzlich stellt es die eingelesenen Bilder wahlweise auch über einen DACS-Stream im `Bild_t`-Format zur Verfügung. Über eine DACS-Funktion können das aktuelle und wahlweise auch ältere Bilder ebenfalls im `Bild_t`-Format von anderen Programmen abgeholt werden. Existiert ein Observer für den Ident `"imageRGB"`, wird das aktuelle Bild zusätzlich mit Hilfe von `plug_data_set()` unter diesem Ident im RGB-Farbraum zur Verfügung gestellt. Dazu wird wiederum der Typ `grabImageData` benutzt.

Hilfsfunktionen

"tools.h" stellt verschiedene kleinere Hilfsroutinen zur Verfügung, die immer wieder benötigt werden. Dazu gehören Funktionen zur Ausgabe von Fehlern, von Warnungen, von Debugmeldungen und Funktionen zum Testen von Assertions:

```
void debug_x (int level, char *str, ...);
void debug_1 (int level, char *str);
void debug_2 (int level, char *str, ARG1);
...
void warning_x (char *str, ...);
void warning_1 (char *str);
void warning_2 (char *str, ARG1);
...
void error_x (char *str, ...);
void error_1 (char *str);
void error_2 (char *str, ARG1);
...
void assert_x (scalar expression, char *str, ...);
void assert_1 (scalar expression, char *str);
void assert_2 (scalar expression, char *str, ARG1);
...
```

Alle diese Funktionen führen intern ein `sprintf()` aus. Die Funktionen `debug_xxx()` und `assert_xxx()` erzeugen nur dann Code, wenn das Macro `DEBUG` beim Kompilieren gesetzt ist. `debug_xxx()` gibt nur dann etwas aus, wenn `level` kleiner als der über die iceWing-Kommandozeile gewählte `talklevel` ist. `warning_xxx()` gibt die Meldung immer aus, `error_xxx()` bricht zusätzlich die Programmausführung ab. Die `_x()`-Varianten der Funktionen erlauben eine beliebige Anzahl von Argumenten. Sie benötigen allerdings für die vollständige Funktion entweder den GCC oder einen ANSI-C99 kompatiblen Compiler. Ansonsten sind diese Varianten nicht als Macro, sondern als Funktionen realisiert, die weniger Informationen als die Macro-Varianten zur Verfügung stellen.

Auch für die Messung der Ausführungszeit von Programmteilen stehen verschiedene Funktionen zur Verfügung. Unter anderem sind dies

```
int  time_add (char *name);
void time_start (int nr);
long time_stop (int nr, BOOL show);

#define time_add_static(number,name) ...
#define time_add_static2(number,name,number2,name2) ...
```

`time_add()` legt einen neuen Zeitmesser an und gibt einen Index zum Ansprechen des Messers zurück. Per `time_start()` kann er gestartet und per `time_stop()` wieder gestoppt werden. Durch `show = TRUE` wird die vergangene Zeit direkt nach `stdout` ausgegeben. Ansonsten geschieht dies nach einer gewissen Anzahl Durchläufe der Hauptschleife. Über `time_add_static()` kann die Initialisierung vereinfacht werden. Hier wird eine neue static Variable mit Namen `number` definiert und per einmaligem Aufruf von `time_add()` initialisiert. Die Benutzung kann wie folgt aussehen:

```
/* Definition anderer Variablen */
...
time_add_static (time_demo, "Demo Messung");

time_start (time_demo);
/* Ausfuehrung des zu messenden Programmteils */
...
time_stop (time_demo, FALSE);
```

Die Auswertung von Kommandozeilenargumenten wird durch die Funktion

```
char parse_args (int argc, char **argv, int *nr, void **arg, char *pattern);
```

erleichtert. Die Funktion testet, ob `argv[*nr]` in `pattern` vorhanden ist, wobei die Groß-/Kleinschreibung von `argv[*nr]` ignoriert wird. Anschließend wird `*nr` passend erhöht, so daß beim nächsten Aufruf von `parse_args()` das nächste Argument untersucht werden kann. Das Format von `pattern` in EBNF ist `{ "-" token ":" ch ["r"|"ro"|"i"|"io"|"f"|"fo"|"c"] " " }`, wobei `token` ein beliebiger String ohne `" "` und `":"` ist und `ch` ein beliebiges Zeichen ist. `ch`

wird zurückgegeben, falls `-token` gefunden wurde. Die restlichen optionalen Zeichen in `pattern` sind Modifikatoren. `"r"` gibt an, daß zusätzlich ein Argument benötigt wird, welches in der Variablen `arg` zurückgegeben wird. Bei `"i"` wird ein zusätzliches Integer-Argument benötigt, bei `"f"` ein Float-Argument. `"c"` bedeutet, daß `token` beliebig fortgesetzt werden kann. Diese Fortsetzung wird in der Variablen `arg` zurückgegeben. Durch `"o"` kann schließlich angegeben werden, daß das String- oder Int-Argument optional ist. Die Anwendung der Funktion `parse_args()` zeigt am einfachsten ein Beispiel:

```
void *arg;
char ch, *str_arg;
int nr = 0, int_arg;

str_arg = NULL;
int_arg = 0;
while (nr < argc) {
    ch = parse_args (argc, argv, &nr, &arg,
                     "-I:Ii -S:Sr -H:H -HELP:H --HELP:H");
    switch (ch) {
        case 'I':
            int_arg = (int)(long)arg;
            break;
        case 'S':
            str_arg = (char*)arg;
            break;
        case 'H':
        case '\0':
            help();
        default:
            fprintf (stderr, "Unknown character %c!\n", ch);
            help();
    }
}
```

Bibliography

[Jun98] Nils Jungclaus. *Integration verteilter Systeme zur Mensch-Maschine-Kommunikation*. Dissertation, Universität Bielefeld, Technische Fakultät, 1998.

[Löm04] Frank Lömker. *Lernen von Objektbenennungen mit visuellen Prozessen*. Dissertation, Universität Bielefeld, Technische Fakultät, 2004. http://bieson.ub.uni-bielefeld.de/volltexte/2004/549/.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, March 1994.

[The03] The Mathworks. Matlab, 2003. http://www.mathworks.com.

Index