Another Image Debugger

FIRST SLAP EDITION

(250513)

TWO-PANEL VIEW

FLEXIBLE BIT PARSER

CROSS PLATFORM

MULTITHREAD DEBUGGING SUPPORT

BUILT-IN COMPARATOR AND OTHER TOOLS

1. Introduction

AID (Another Image Debugger) is a tool for visualization of binary data with a predefined pixel format. The data can be loaded from a file or can be transferred from a debugged platform via TCP/IP. The described tool allows for basic transformations, data recasting and images comparison. Intercepted images can be stored in one of the common graphics formats: TIFF, PNG, JPG, BMP or a dedicated one called RIC (Raw Image Container). Apart from the image source buffer, a RIC file stores additional information – particularly *notes* containing image statistics and/or comments.

AID allows for a relatively flexible pixel format specification, where each of the RGBA channels can have different bit count and data type (float, signed/unsigned integer). Data bits can be shared between RGB channels. A pixel column and a row stride can be also defined.

This tool implementation utilizes Qt(R) libraries delivered by the qt-project community on the GPL/LGPL license.

1.1. Basic architecture description

AID applies 3-level data processing to meet performance vs. functionality requirements. The raw data, sent from a debugged application or loaded from a file, are referenced as *source data* (1st level). These data are interpreted and transformed according to additional information included in a header and auxiliary structure sent/delivered along with the source data. The interpreted data are cached as *visual data* (2nd level). The visual data are stored with a device-independent 8-bit per channel format (as most display devices support up to 8bit per channel color depth). Additional fine-filtering applied by the user results in *render data* (3rd level). The render data, depending on the target platform, can be cached by an OS windows-server. The stage of transforming data between the 1st and the 2nd level is called *heavy filtering*, while the stage between the 2nd and the 3rd level can be described as *on-the-fly filtering*. The heavy filtering concerns time consuming routines like source data parsing. The on-the-fly filtering comprises: bit masking, image flip/scale and channel swapping. The main structure of data flow and processing is depicted in Fig. 1. Three built-in tools, whose full description can be found in Section "Tools" is briefly presented as follows:

- Reinterpret data – this tool allows for the source data reinterpretation. By example, source data sent by the debugged application as R5G6B5 can be reinterpreted as data with R8G8B8A8 pixel format with different image width and height. The reinterpret data tool produces new <u>visual data</u>.
- Recast data – this tool allows for creating <u>new source data</u> of the float 32 RGBA format from a given input source data. Such an operation can be useful for preparation of a reference image utilized later in a comparison process. This tool allows additionally for applying bias/gain alternations and image horizontal/vertical flip.
- Source comparator – compares two selected source data buffers. The comparison process is consisted of two main stages: recasting input data to float 32 RGBA format, and subtraction. For example – given two images: (1) 128x128 with

RGB8 pixel format and (2) 30x200 with 10-bit float RGB and 2 bit alpha channel, the comparison process consists of:

- determination of the images intersection. Assuming that the relative image shift is equal to zero (this parameter is configured by a user) the resulting image will have width of 30 and height of 128.
- data recasting. For (1) RGB8 → 32bit float RGBA;
  for (2) 10bit float RGB 2bit alpha → 32bit float RGBA conversions are performed.
- subtraction. This stage produces a new source data being a result of images' recast source data (1) minus (2) operation. The associated visual data represent an absolute value of this operation. Notice that the user has an access to both: the source and the visual data. Therefore, the visual data give the visual information about 'where' is a difference, while the source data give numerical information about a value of this difference.

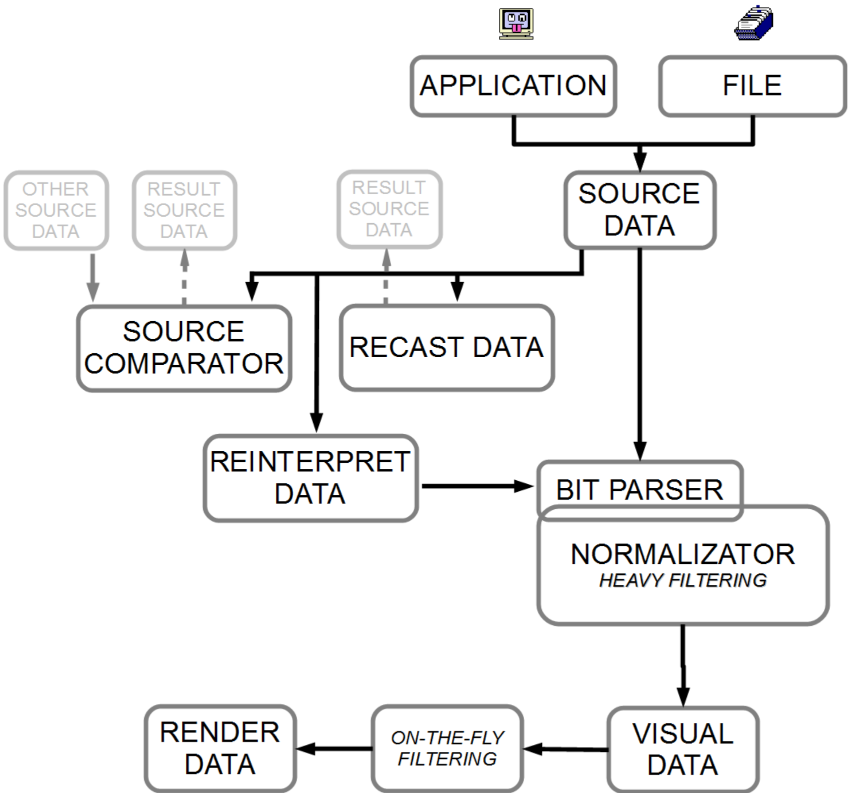


Fig.1. Data flow diagram.

2. Bit parser

The bit parser block is responsible for transforming the source data into the visual data. The parsing procedure is controlled by a pixel format string. This string can be composed by type tokens, channel tokens, numbers describing bits count and auxiliary modifiers.

Type tokens: **u** – unsigned integer; **i** – signed integer; **f** – float.

Channel tokens: **`R G B A`** – for red, blue, green and alpha channel, respectively; **`X`** – dummy bits for a bit stride definition; **`E`** – shared bits for the RGB channels.

Auxiliary modifiers: **`a`** – absolute value.

2.1 Pixel string format rules

- the pixel format string interpretation is case sensitive

- a valid bit-count per channel is within the range [0-32] for integers and 10, 11, 16 and 32 for the float types

- bit groups for a given channel can be scattered
  By example the following definition: `R2 G3 B4 R2 G2 B2 A1 X1 A2` is interpreted as a declaration of 4-bit red channel R[0-1;9-10], 5-bit green channel G[2-4;11-12], 6-bit blue channel B[13-16,21-22] and 3-bit alpha channel A[23, 25-26].

- a default pixel format is the unsigned integer
  If the pixel format string is defined without any format token (e.g. `R8 G8 B8`), RGB channels are treated as 8-bit unsigned integer values.

- the declared type token is valid for consecutive channels unless other declaration is detected in the pixel format string
  Let us consider the following pixel string: `R8 G8 iB8 A8`. The R and G channels are treated as unsigned integer 8-bit values. The B and A channels are interpreted as signed integer 8-bit values.
  For scattered bit groups the last declared type token is valid, therefore `uR2 G8 iR4` is interpreted as a signed integer 6-bit R channel and an 8-bit unsigned integer G.
  Type declaration has no effect for the shared bits: `iR5 uG5 E3` is interpreted as 8-bit signed integer value R, and 8-bit unsigned integer value G. A declaration of a channel type without committing a bit count is also valid. Such a construction can be applied when declaring a channel consisted of the shared bits only. `G2 iR E5` is interpreted as 7-bit unsigned integer G and 5-bit signed integer R with 5-bits shared.

- the pixel format string can contain spaces between tokens groups
  Therefore `uR4 iG4 B4 fA10` is equivalent to `uR4iG4B4fA10`.

Other examples:

R	G	B	A	Notes	Pixel format string
uint8	uint8	uint8	–		`uR8 G8 B8`
uint5	uint6	uint5	uint1		`uR5 G6 B5 A1`
uint4	int4	int4	float10		`uR4 iG4 B4 fA10`
float10	float11	int4	float16	with 4bits stride between G and B channels	`fR10 G11 X4 iB4 fA16`
int8	int8	int8	float16	with 4 MSBs shared for RGB channels and pixel column stride = 2 bits	`iR4 G4 B4 E4 fA16 X2`
uint7	uint8	uint7	–	with 4 LSBs shared for RGB channels	`uE4 R3 G4 B3`
uint8	uint8	uint8	uint2	with 4-4 bit groups interlaced	`uR4 G4 B4 R4 G4 B4 A2`

3. Image data transfer

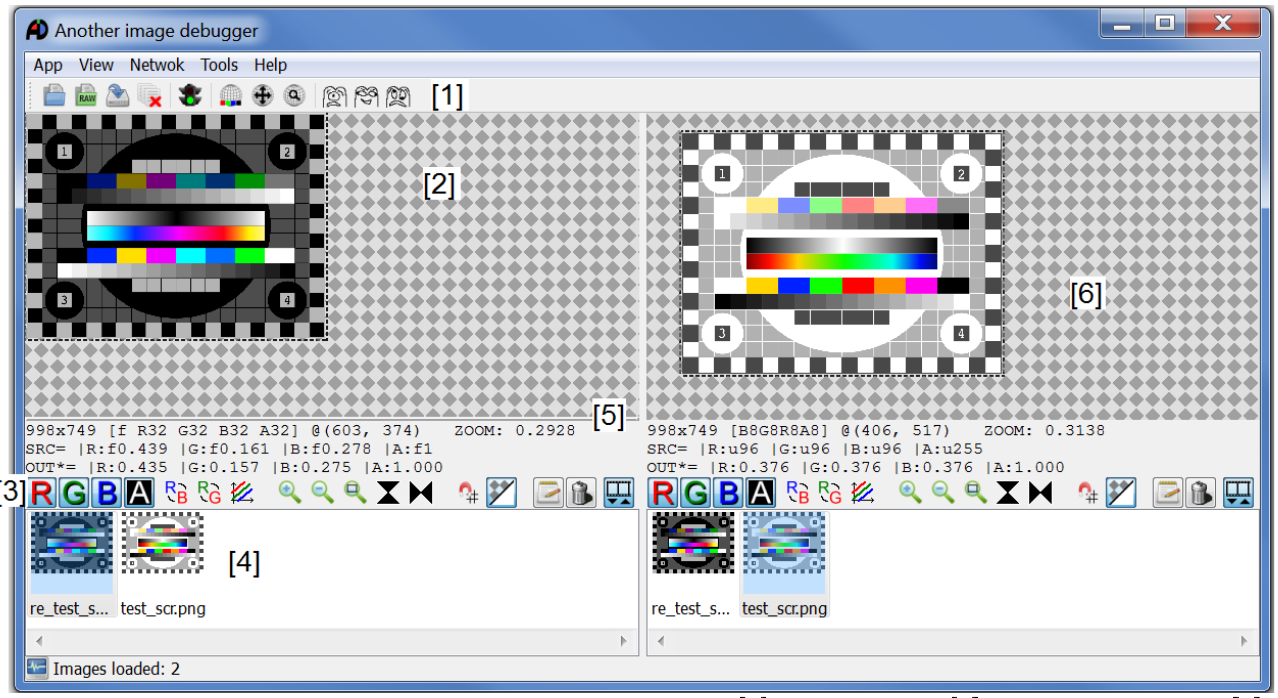
The communication between the debugged application(s) and AID can be realized by TCP/IP. The simple scenario of data transfer comprises three stages for a client application: (i) establishing a connection, (ii) waiting for a prompt char `'$'` from AID, (iii) sending data, (iv) disconnect event. Notice that the disconnect event triggers bit parsing and further data processing. The source data have to be preceded by a header, whose structure is presented below. A data stream in a typical transmission is defined as follows:

```
***HEADER****************************************************************
[4 bytes of magic chars] 'AID0'
[32bit unisgned integer] (iwidth)          image width  (<=2048)
[32bit unsigned integer] (iheight)         image height (<=2048)
[32bit unsigned integer] (pixFrmStrLen)    pixel format string length
[32bit unsigned integer] (srcDataSize)     image source data size
[32bit unsigned integer] (imgNameStrLen)   image name string length
[32bit unsigned integer] (imgNotesStrLen)  image notes string length
[32bit unsigned integer] (rowStrideInBits) bit count of the row stride
[32bit float           ]                   channel R gain
[32bit float           ]                   channel G gain
[32bit float           ]                   channel B gain
[32bit float           ]                   channel A gain
[32bit float           ]                   channel R bias
[32bit float           ]                   channel G bias
[32bit float           ]                   channel B bias
[32bit float           ]                   channel A bias
[32bit unsigned integer]                   auxiliary flags
***STRING SECTION********************************************************
[imgFrmStrLen   x 1 byte]                  pixel format string(<=128 bytes)
[imgNameStrLen  x 1 byte]                  image name string (<=128 bytes)
[imgNotesStrLen x 1 byte]                  image notes string(<=4KB)
***SOURCE DATA SECTION***************************************************
[srcDataSize x 1 byte]
```

A simple implementation of an AID client for Windows(R)/Linux operating systems can be found in attached examples. The above scheme describes also a structure of the dedicated RIC file format.

4. User interface

The main window can be composed of one or two view panels. The layout of the panels can be set to horizontal or vertical. Fig. 2 presents a selected view. The active panel bar marks the panel, whose content will be utilized by the 'Reinterpret data' and 'Recast data' tools as well as the 'Save panel content as' commands from the main menu. The view panel's toolbar triggers the 'on-the-fly' filters. Notice that the output of the 'on-the-fly' filters is utilized only to render the view panel content. The built-in tools and 'save to file' routines work on the source data only.



```
998x749 [f R32 G32 B32 A32] @(603, 374)   ZOOM: 0.2928
SRC= |R:f0.439 |G:f0.161 |B:f0.278 |A:f1
OUT*= |R:0.435 |G:0.157 |B:0.275 |A:1.000
```

```
998x749 [B8G8R8A8] @(406, 517)   ZOOM: 0.3138
SRC= |R:u96 |G:u96 |B:u96 |A:u255
OUT*= |R:0.376 |G:0.376 |B:0.376 |A:1.000
```

Fig. 2. AiD main window with two panels – vertical layout. [1] Main toolbar, [2] left/top view panel, [3] view panel toolbar, [4] pick-up list, [5] active panel bar mark, [6] right/bottom view panel.

There are special icons, presented in Fig.3, which can be assigned to images. The 'busy' icon is displayed, when the image is subjected to data processing like: bit parsing, recasting and comparison procedure. Additional progress notification can be displayed in the view panel. The 'non-renderable' icon is displayed when no visual data can be extracted from the source data processed against a given pixel format string.

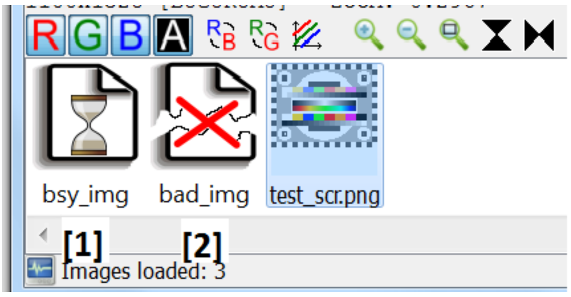
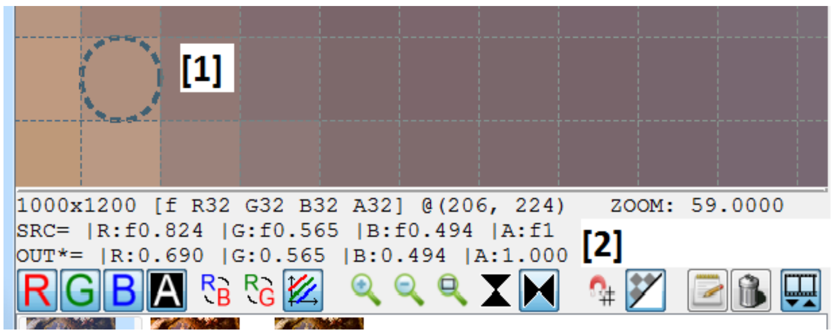


Fig. 3. Special icons: [1] busy icon, [2] non-renderable icon.

The basic operation like view scaling/move are performed by dragging a mouse and using a mouse roller, respectively. If the two panel mode is active and the CTRL key is hold, the mouse movement/roller is replicated on the other panel.

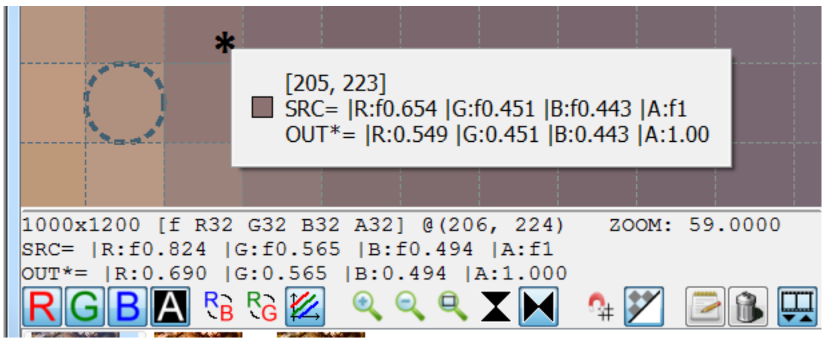
Left click on the image results in a pixel cursor positioning and the bottom info-bar pop-up. The bottom info-bar shows the coordinates of the selected pixel, source (SRC) and the render data (OUT) values. The pixel cursor is represented by a dotted circle (see Fig. 4.). The asterisk mark after "OUT" string notifies about gain/bias alternations applied during the visual data creation. Detailed gain/bias values can be found in the image's notes.



```
1000x1200 [f R32 G32 B32 A32] @(206, 224)   ZOOM: 59.0000
SRC= |R:f0.824 |G:f0.565 |B:f0.494 |A:f1
OUT*= |R:0.690 |G:0.565 |B:0.494 |A:1.000
```

Fig. 4. The pixel marker [1] and the bottom info-bar [2].

Right click on the image surface shows the pop-up pixel info-box (see Fig. 5.)



```
[205, 223]
SRC= |R:f0.654 |G:f0.451 |B:f0.443 |A:f1
OUT*= |R:0.549 |G:0.451 |B:0.443 |A:1.00
```

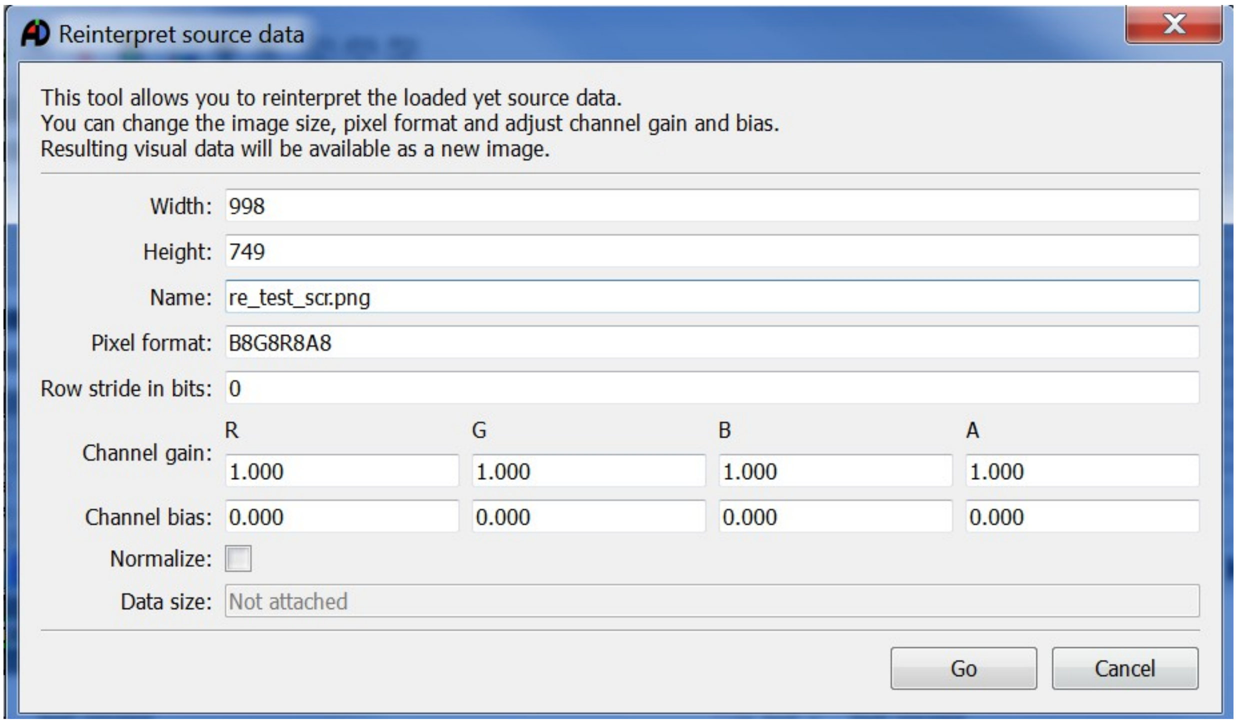
```
1000x1200 [f R32 G32 B32 A32] @(206, 224)   ZOOM: 59.0000
SRC= |R:f0.824 |G:f0.565 |B:f0.494 |A:f1
OUT*= |R:0.690 |G:0.565 |B:0.494 |A:1.000
```

Fig. 5. Pop-up pixel info-box.

5. Tools

AID provides three basic built-in tools.

The 'Reinterpret data' tool (see Fig. 6) allows for re-processing of the loaded yet source data according to the altered pixel format string definition. This tool allows also for changing the image size, name, flip, row stride and adjust image gain and bias factors utilized during the visual data extraction process. The data size box displays the actual size of the source data. The new image size, stride and pixel format must match the source data size or the newly reinterpreted image will be marked as non-renderable. The 'Reinterpret data' tool can be applied for 'non-renderable' images. By example when the client application sent an invalid data or header and the 'non-renderable' icon is visible, you can alter pixel image string and try to reinterpret the uploaded data again. For non-renderable images additional information about errors can be found in the image's notes.

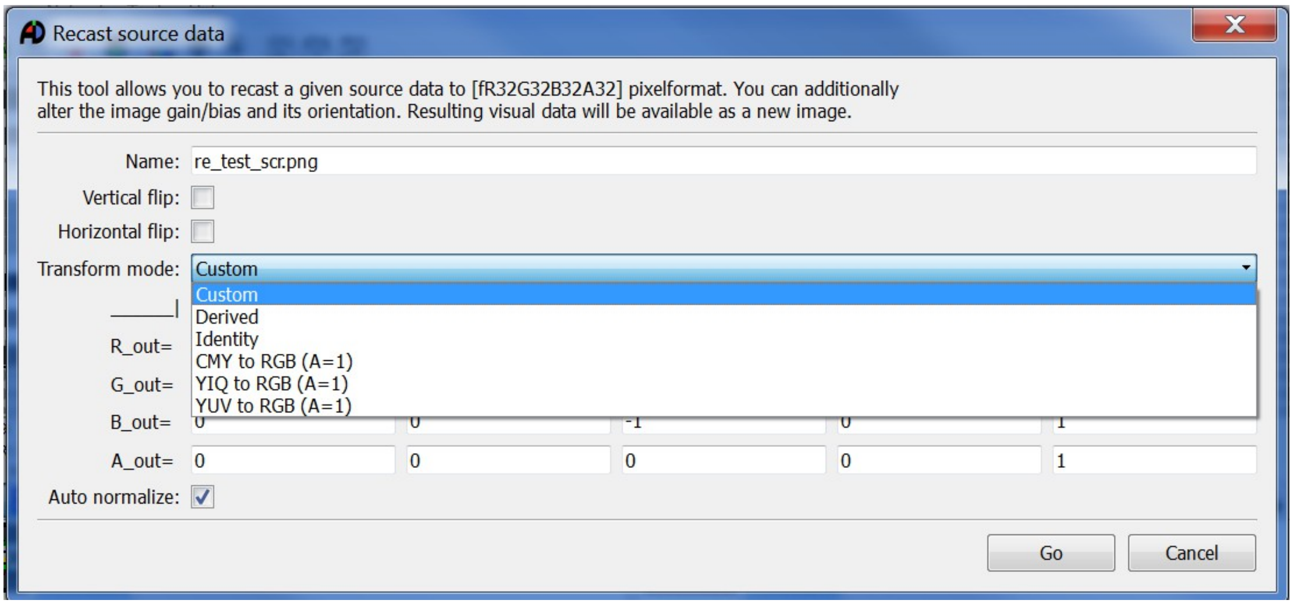

Reinterpret source data

This tool allows you to reinterpret the loaded yet source data.
You can change the image size, pixel format and adjust channel gain and bias.
Resulting visual data will be available as a new image.

Width:	998			
Height:	749			
Name:	re_test_scr.png			
Pixel format:	B8G8R8A8			
Row stride in bits:	0			
	R	G	B	A
Channel gain:	1.000	1.000	1.000	1.000
Channel bias:	0.000	0.000	0.000	0.000
Normalize:	☐			
Data size:	Not attached			

Fig. 6. Reinterpret source data tool dialog.

The 'Recast data' tool (see Fig. 7) performs a translation of the source data from its native format to the 32bit float RGBA pixel format. Additionally, an image flip and channel switching/multiplexing can be set. Alternations of the source data gain and bias can be also defined in the tool's dialog window. After the recast operation new source data are created and new visual data buffer is associated.

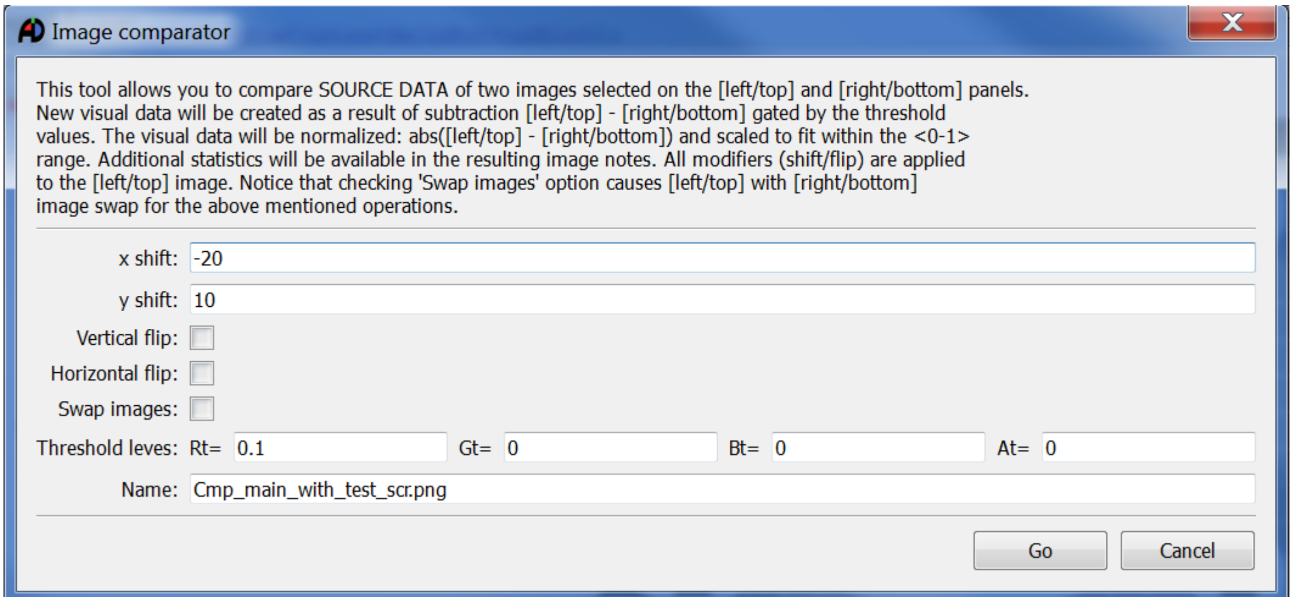

Recast source data

This tool allows you to recast a given source data to [fR32G32B32A32] pixelformat. You can additionally alter the image gain/bias and its orientation. Resulting visual data will be available as a new image.

Name:	re_test_scr.png				
Vertical flip:	☐				
Horizontal flip:	☐				
Transform mode:	Custom				
B_out=	0	0	-1	0	1
A_out=	0	0	0	0	1
Auto normalize:	✓				

Fig. 7. Recast data tool dialog.

The image comparator (see Fig. 8) outputs the visual interpretation along with numerical data of two given images subtraction. A dialog window for this tool allows for defining the images' relative offset and flip operation. For the intersecting area of the selected input images the recast operation is performed to obtain the 32bit float RGBA representation. Newly created source data are a result of subtraction of two recast intersecting areas. The quantization effect resulting from a comparison of two images with different color depth can be compensated by a further processing with the 'Recast data' tool. You can additionally set threshold levels for RGBA channels – below these values differences will not be reported.



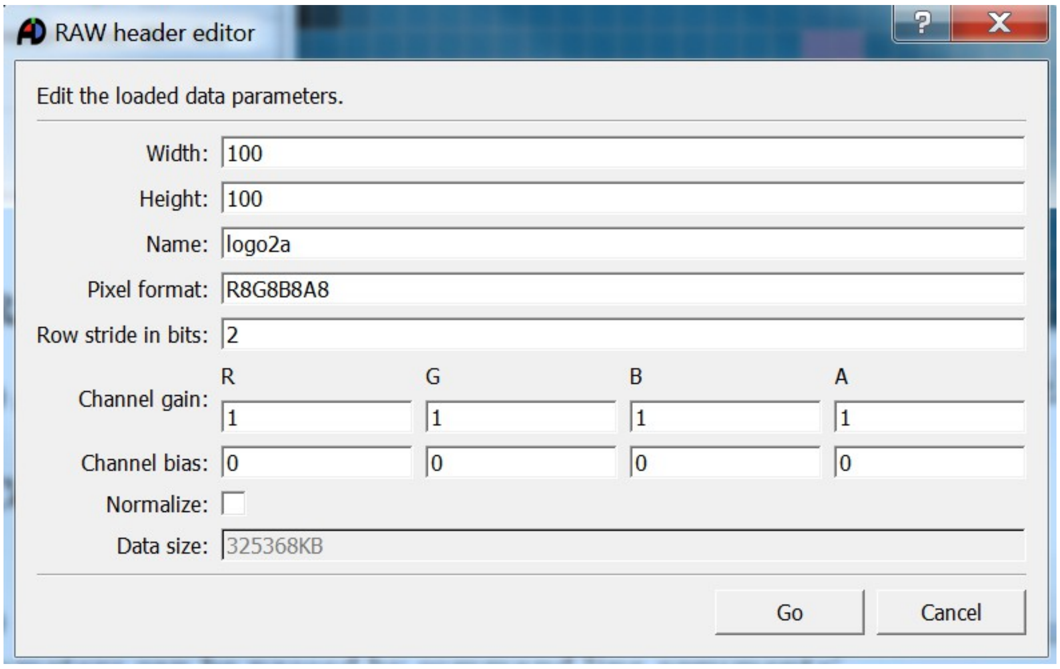
This tool allows you to compare SOURCE DATA of two images selected on the [left/top] and [right/bottom] panels. New visual data will be created as a result of subtraction [left/top] - [right/bottom] gated by the threshold values. The visual data will be normalized: abs([left/top] - [right/bottom]) and scaled to fit within the <0-1> range. Additional statistics will be available in the resulting image notes. All modifiers (shift/flip) are applied to the [left/top] image. Notice that checking 'Swap images' option causes [left/top] with [right/bottom] image swap for the above mentioned operations.

x shift: -20
y shift: 10
Vertical flip: ☐
Horizontal flip: ☐
Swap images: ☐
Threshold leves: Rt= 0.1 Gt= 0 Bt= 0 At= 0
Name: Cmp_main_with_test_scr.png

Fig. 8. Image comparator tool dialog.

6. RAW data files input

AID allows for visualization of data loaded from raw binary files. After opening selected RAW file, the RAW header editor window pops up (Fig. 9) allowing for manual header definition, needed for the visualization process.



Edit the loaded data parameters.

Width: 100
Height: 100
Name: logo2a
Pixel format: R8G8B8A8
Row stride in bits: 2

	R	G	B	A
Channel gain:	1	1	1	1
Channel bias:	0	0	0	0

Normalize: ☐
Data size: 325368KB

Fig. 9. RAW-data header editor.

7. Command line arguments

AID does not provide any configuration file functionality. Start-up parameters can be passed by command line arguments:

```
-winpos <xpos> <ypos>         window position
-winsize <width> <height>     window size
-port <port_number>           TCP/IP port number
-tout <time_out_in_secs>      TCP/IP socket timeout
-dhex                         hex values representation for integers
-maximgs <images_count_limit> loaded images limit
-gpos                         global position for images
-gzoom                        global zoom for images
-gflags                       global flags for images
-panelh                       two panels view with horizontal
                              layout
-panelv                       two panels view with vertical
                              layout
-fontscale <fscale>           additional font scaling factor
```

where the global position/zoom/flags cause the position, zoom and RGBA switch, linear filter parameters and flip configuration flags to be propagated among all of the loaded images instead of keeping this values assigned to each of the images individually.

8. To-do list

Future development will cover the following functionality:

- non-RGBA color space support (YUV, CMYK...)
- UART serial communication interface (support for an embedded system remote debug)
- a global header tool – the user will be able to define the header from the UI. In such case source data will no longer need to be delivered with the header. This can be useful when intercepting raw bit stream directly from third-party debuggers or UART interfaces.

9. License