→ EXCERPT TRAINING COURSE DATA STRUCTURES IN HALCON

ANDREAS HEINDL

CONTACT US

For quotations on HALCON trainings contact us at

Andreas Heindl
andreas@heindl-solutions.com

http://www.heindl-solutions.com/halcon-training.en.html

EXCERPT OF TRAINING COURSE - DATA STRUCTURES IN HALCON

HEINDL SOLUTIONS

andreas@heindl-solutions.com
http://www.heindl-solutions.com

Heindl Solutions is your **MVTec Certified Integration Partner**. We create **HALCON machine vision solutions** and **complete GUI applications**.

HEINDL SOLUTIONS

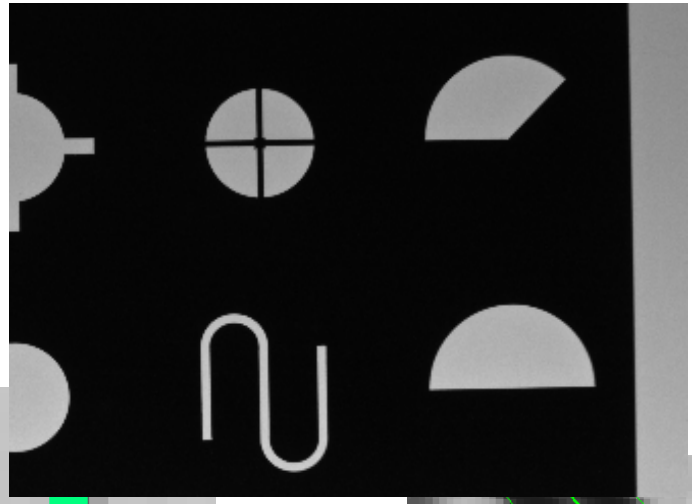
ICONIC AND CONTROL PARAMETERS

Basically two types of parameters:

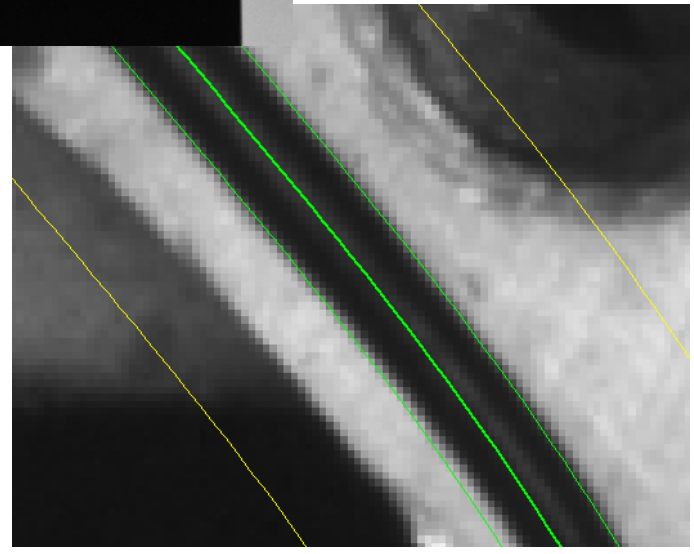
- Iconic parameters / **Objects**
- Control parameters / **Tuples**
- Objects and tuples can be combined in **vectors**

ICONIC PARAMETERS

Bilder (images)



Konturen (contours, XLDs)



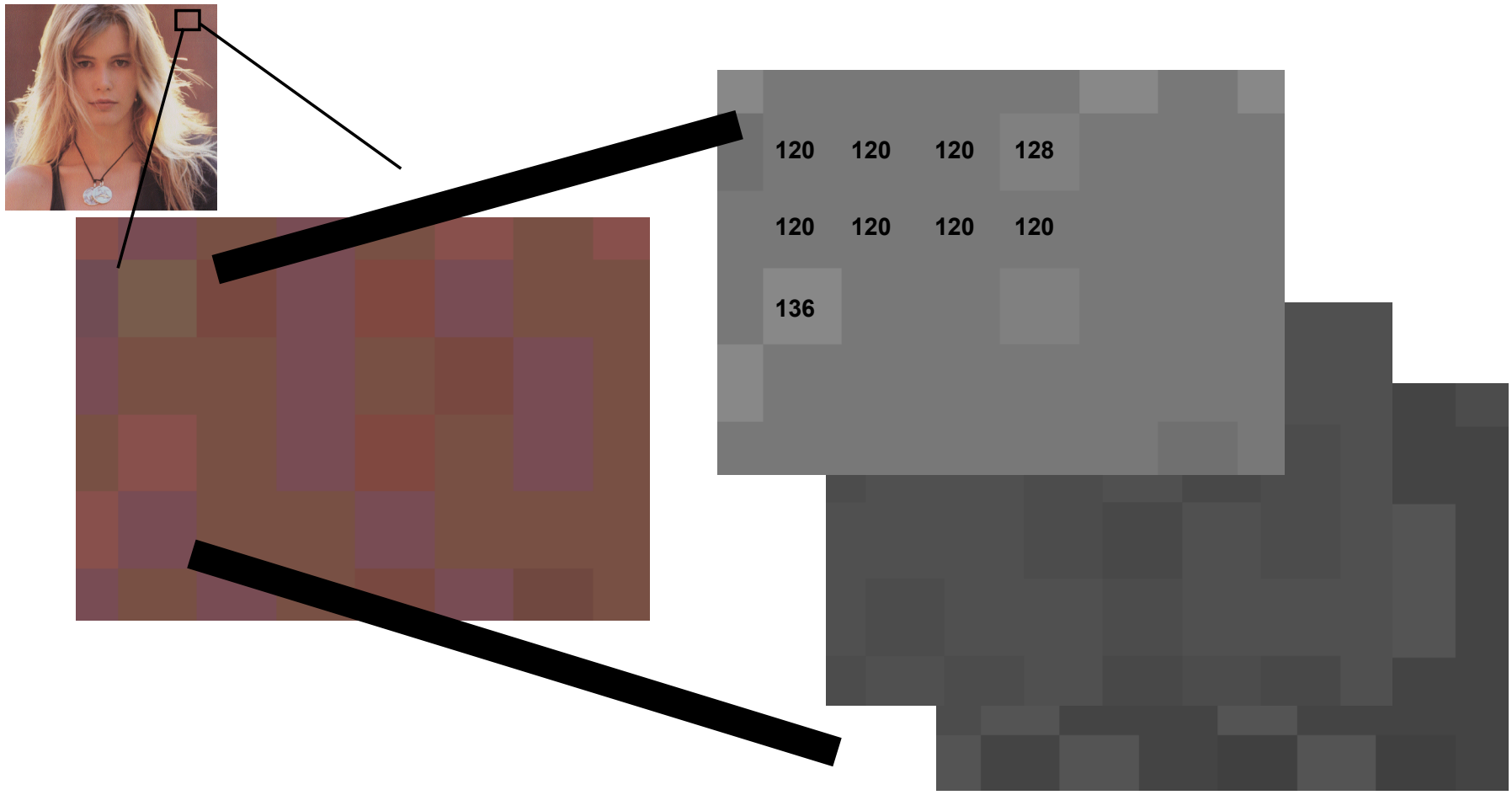
Regionen (regions)

IMAGES

An image consists of:

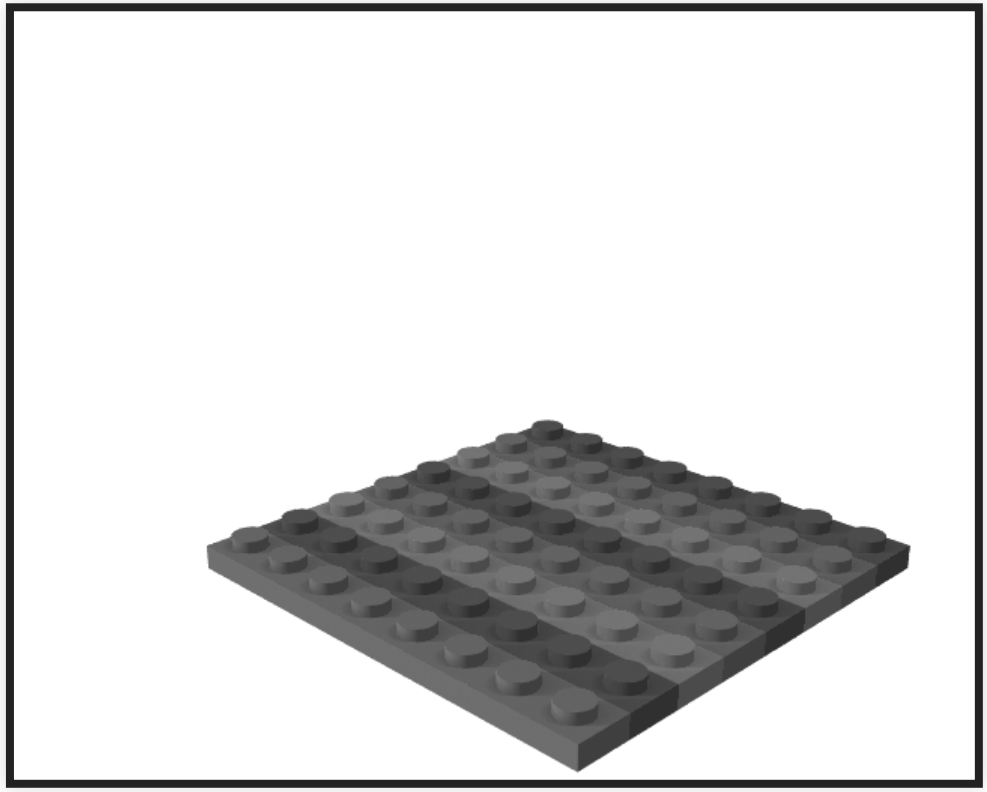
- **Channels** (1..N)
  - **Gray values** (1 .. w × h, `byte`, `real`, …)
- **Domain** (1)
- **Height**
- **Width**

IMAGES



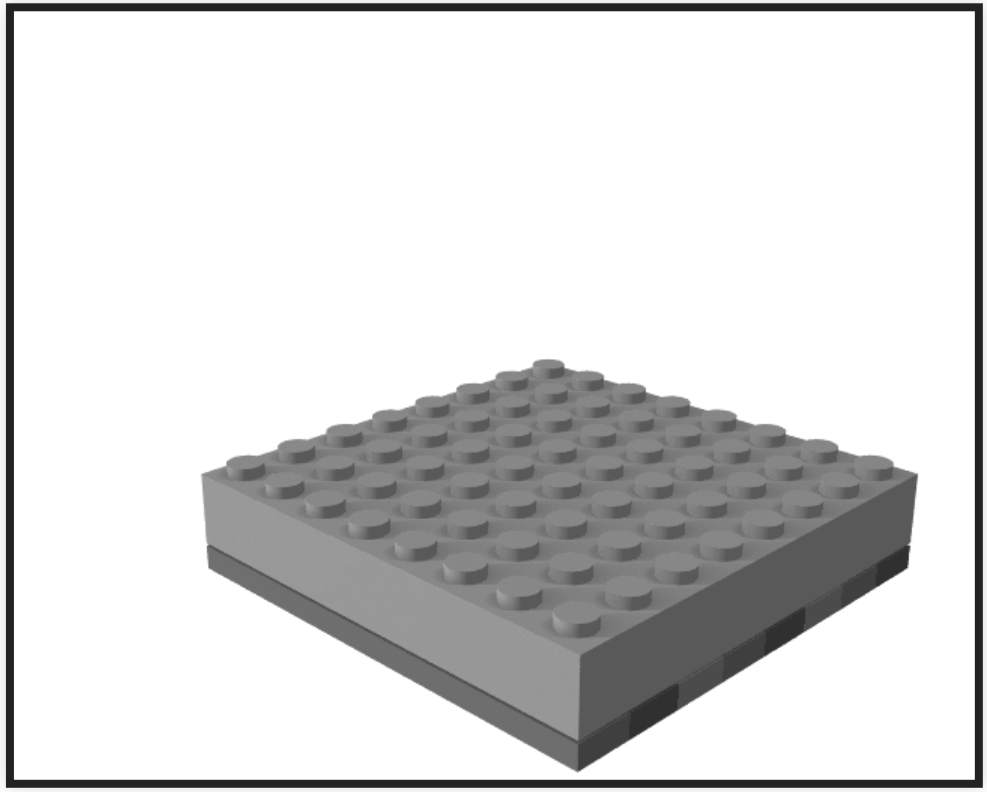
IMAGES

- **Domain region**
  default: full domain



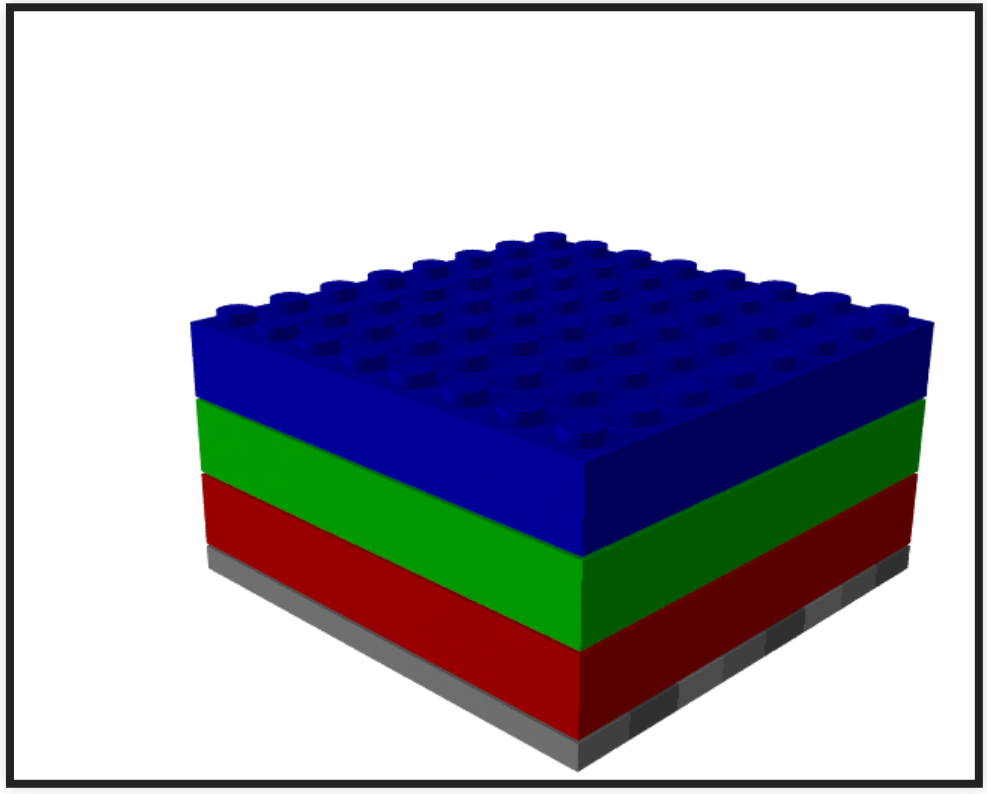
IMAGES

- Domain region
- **Single channel**



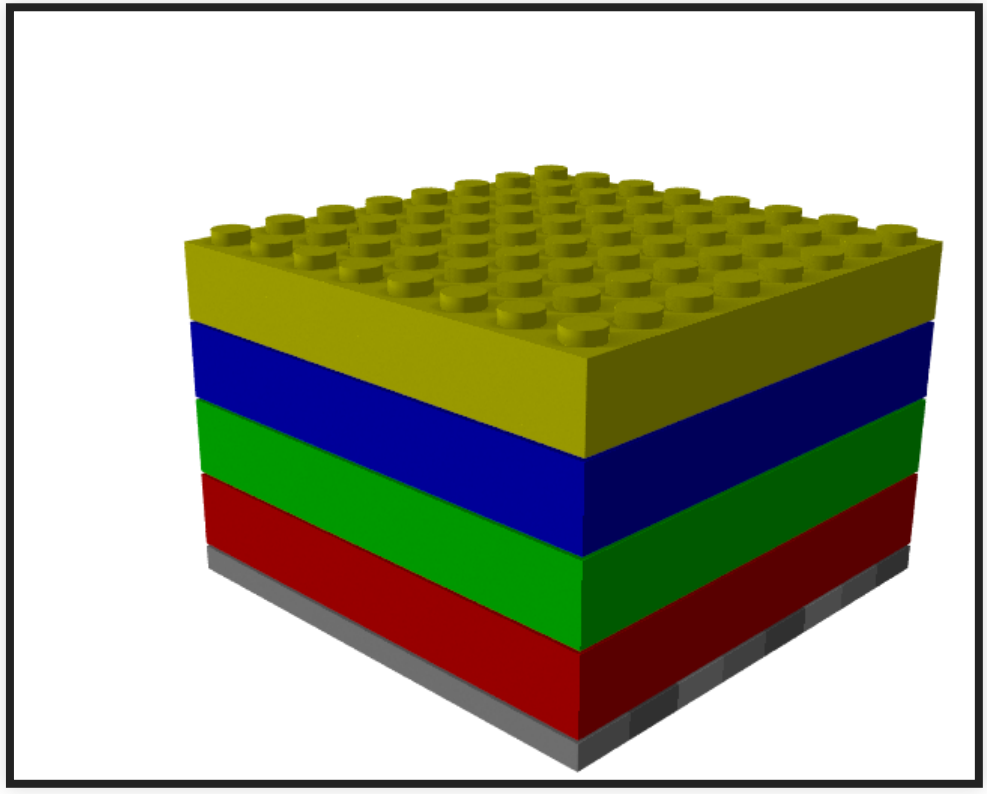
IMAGES

- Domain region
- Channel 1
- Channel 2
- Channel 3



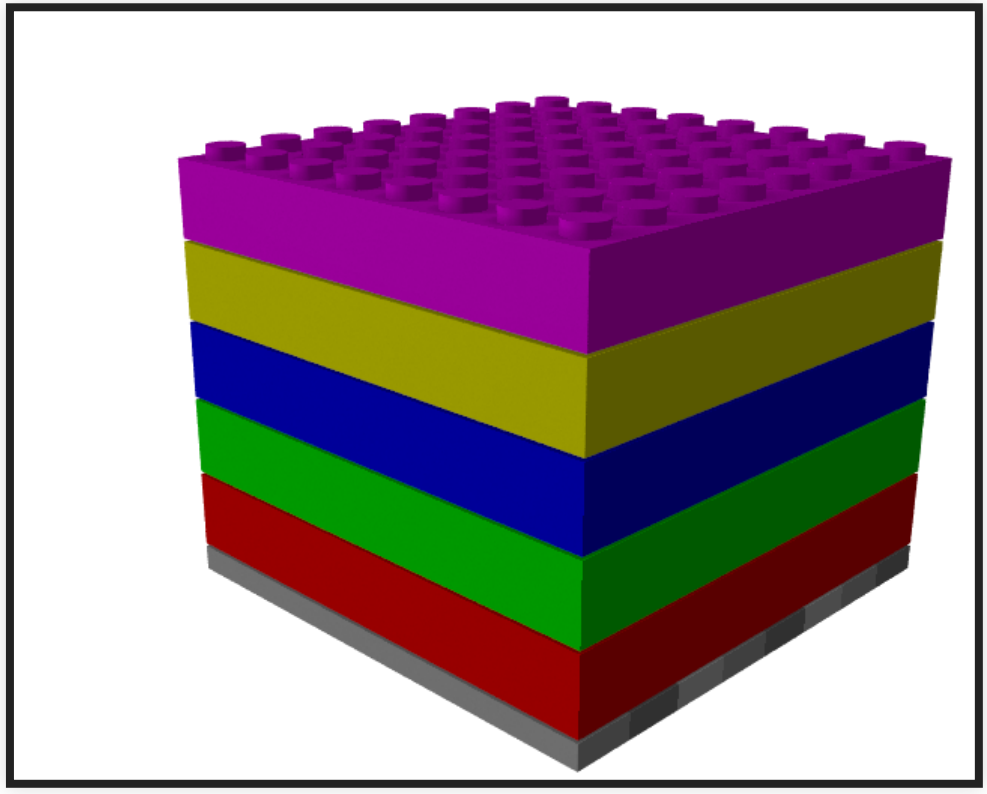
IMAGES

- Domain region
- Channel 1
- Channel 2
- Channel 3
- **Channel 4**



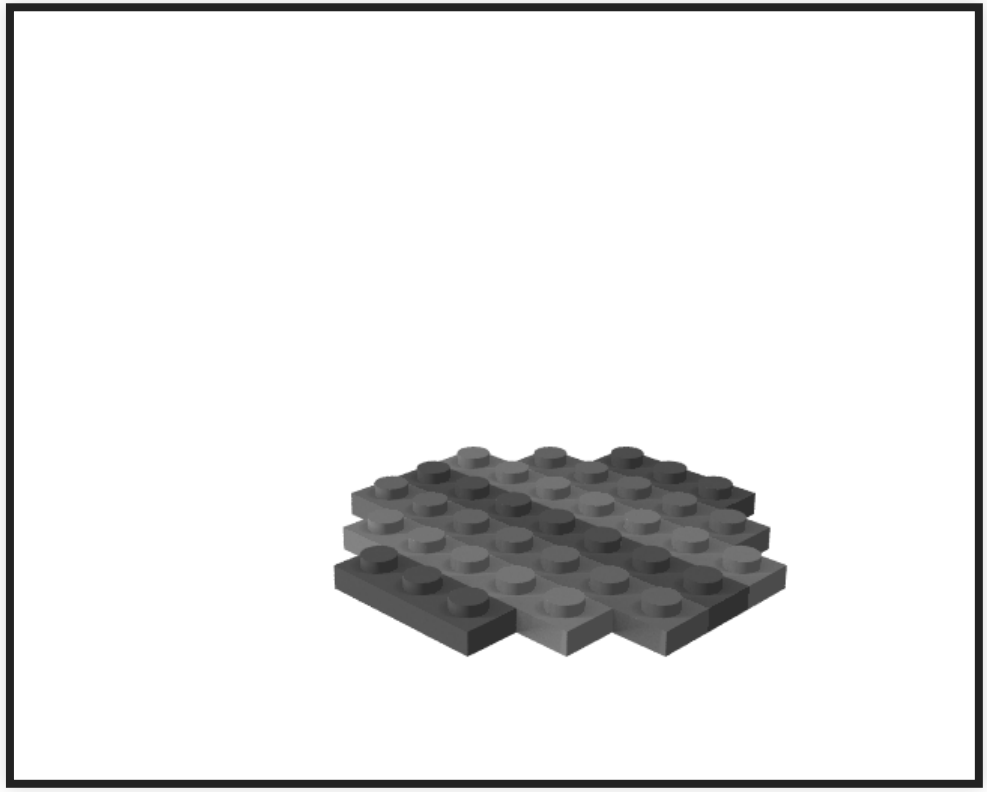
IMAGES

- Domain region
- Channel 1
- Channel 2
- Channel 3
- Channel 4
- ⋮
- Channel N



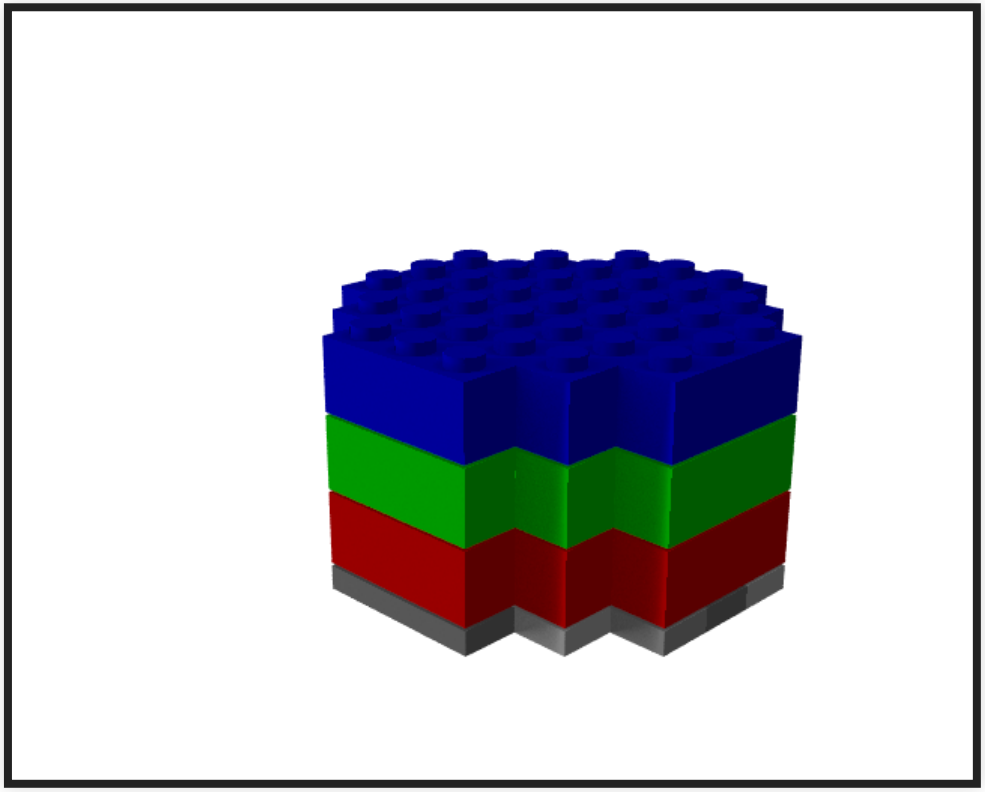
IMAGES

- Domain region can be **reduced**



IMAGES

- Domain region
  can be **reduced**
- Most operators process only gray values within domain
  - **not all operators respect the image domain**



READ/WRITE IMAGES

-

`$HALCONIMAGES/patras.png`

```
read_image (Image, 'patras')

write_image (Image, 'png', 0,'C:/TEMP/MyImage')
:
```

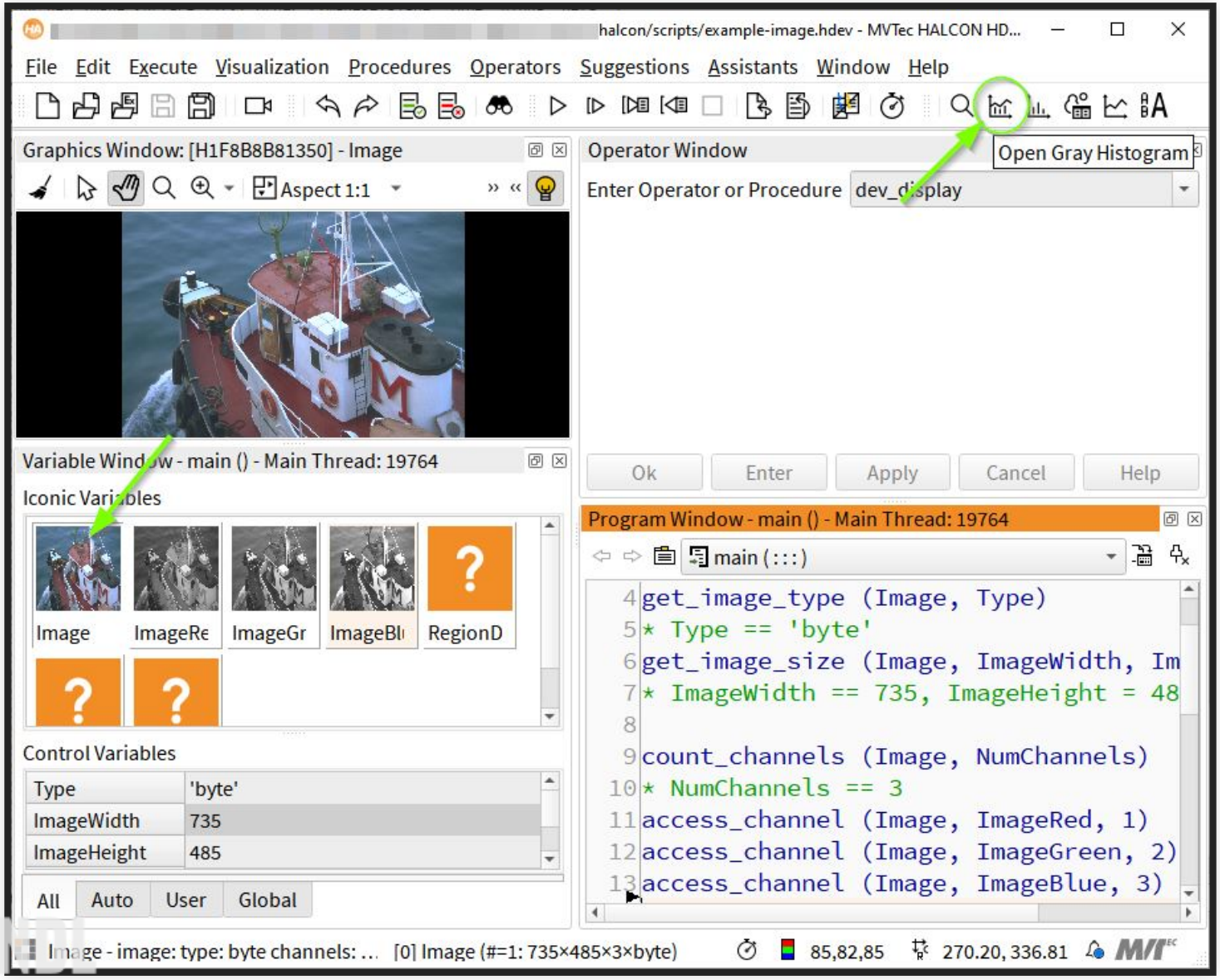
☞ GET PROPERTIES OF IMAGES

```
:
get_image_type (Image, Type)
* Type == 'byte'
get_image_size (Image, ImageWidth, ImageHeight)
* ImageWidth == 735, ImageHeight = 485

:
```

☞ ACCESS TO (COLOR) CHANNELS

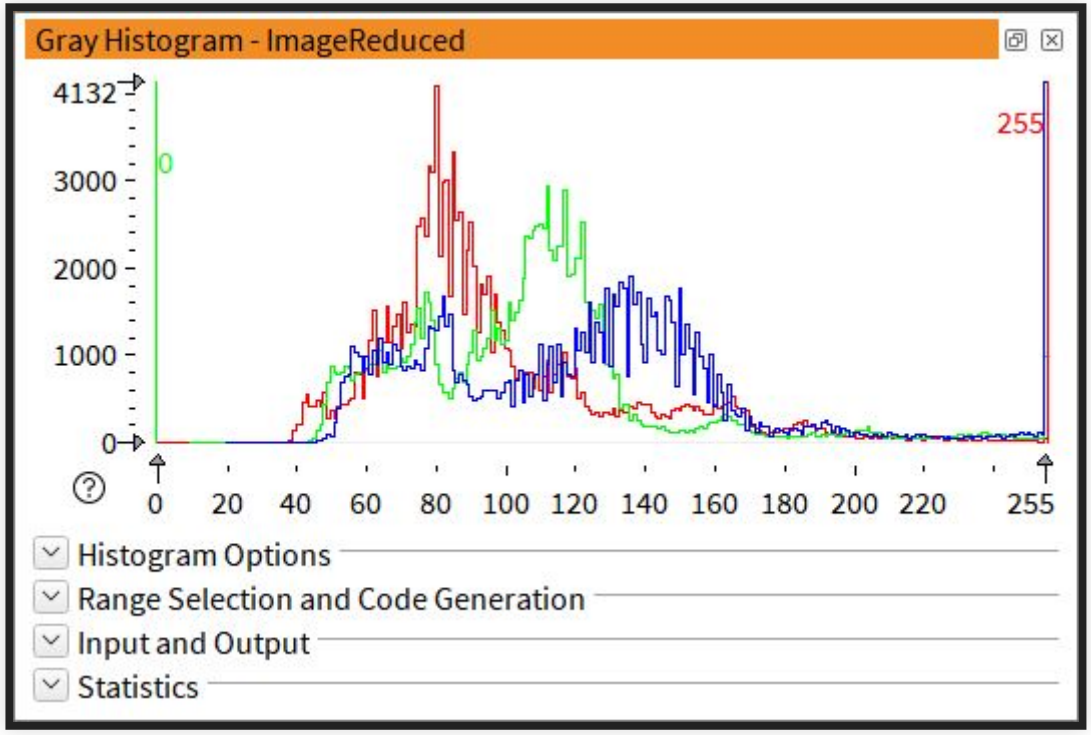
```
⋮
count_channels (Image, NumChannels)
* NumChannels == 3
access_channel (Image, ImageRed, 1)
access_channel (Image, ImageGreen, 2)
access_channel (Image, ImageBlue, 3)
⋮
```

☞ VIEW HISTOGRAM



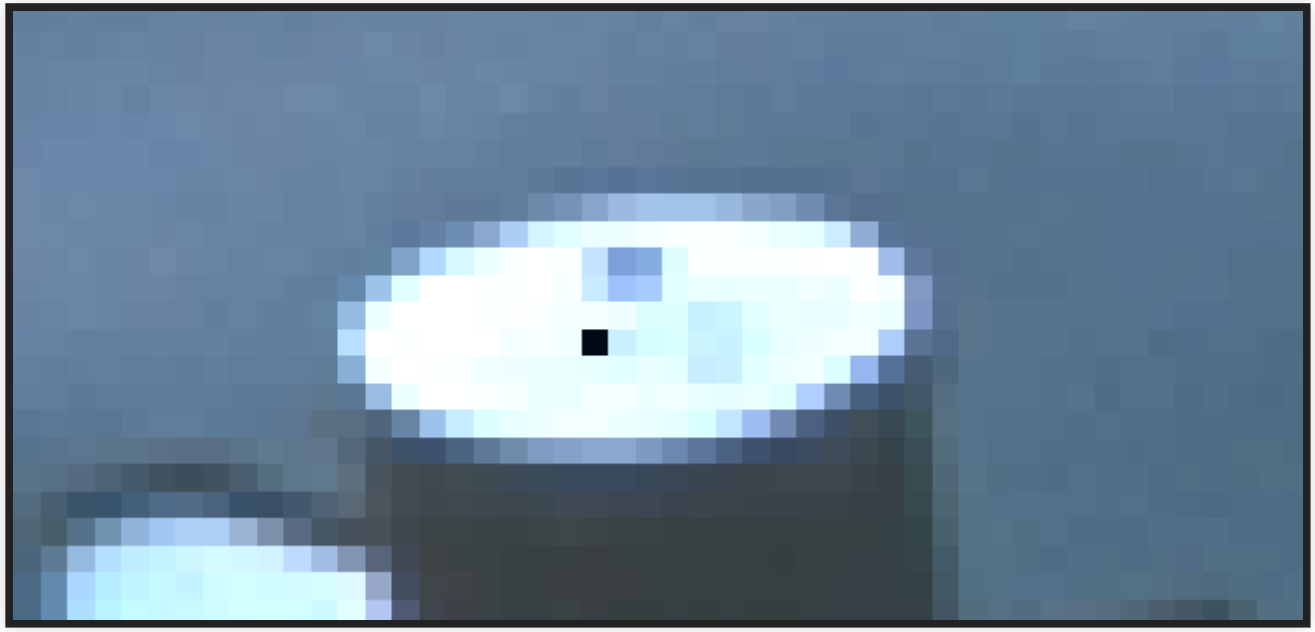

HEINDL
SOLUTIONS

👉 VIEW HISTOGRAM



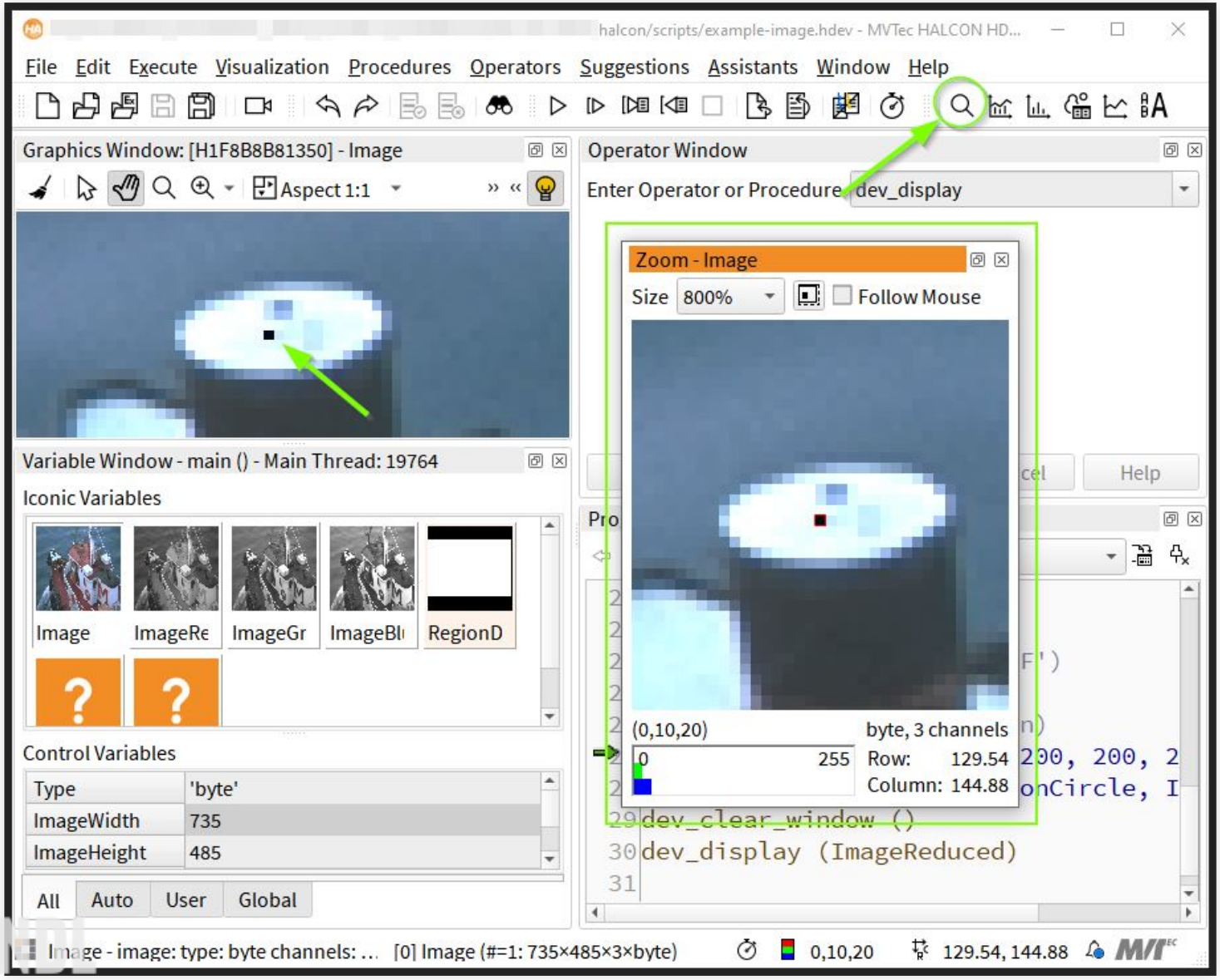
HEINDL SOLUTIONS

☞ ACCESS TO PIXEL GRAY VALUES



```
:
get_grayval (Image, 130, 145, Grayval)
* Grayval == [214, 252, 255]
set_grayval (Image, 130, 145, [0,10,20])
:
```

☞ ZOOM TO IMAGE PART




HEINDL
SOLUTIONS

☞ GET IMAGE DOMAIN

```
⋮
get_domain (Image, RegionDomain)
```

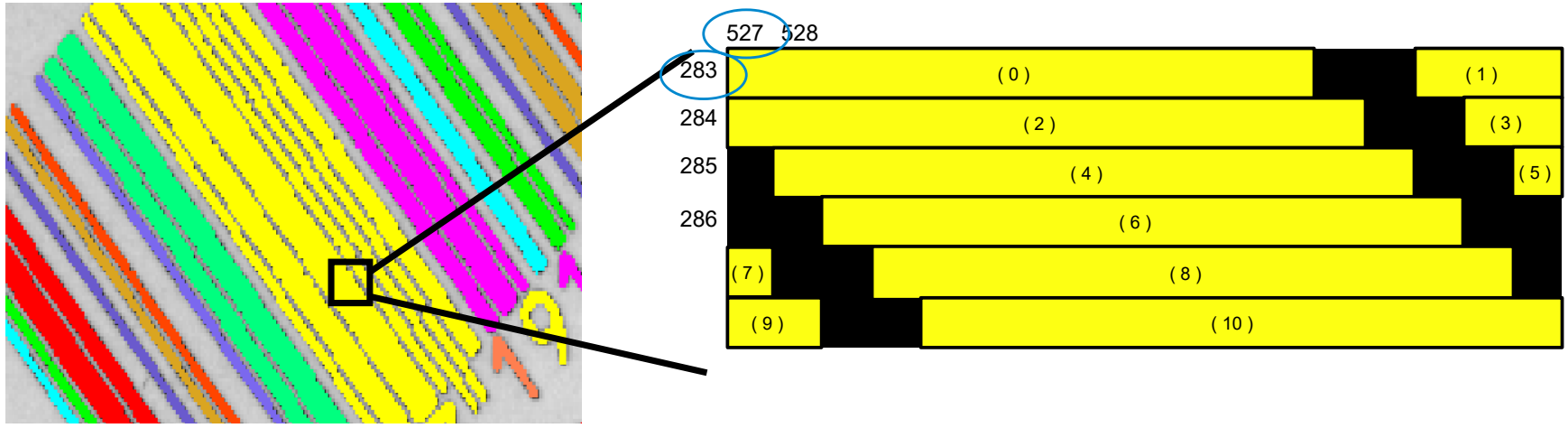
MODIFY IMAGE DOMAIN

```
* Create arbitrary region (for demo)
gen_circle (RegionCircle, 200, 200, 200.5)

reduce_domain (Image, RegionCircle, ImageReduced)

dev_clear_window ()
dev_display (ImageReduced)
```

REGIONS

- arbitrary shape (not restricted to rectangles, circles, …)
- pixel precise
- run-length encoded

REGIONS

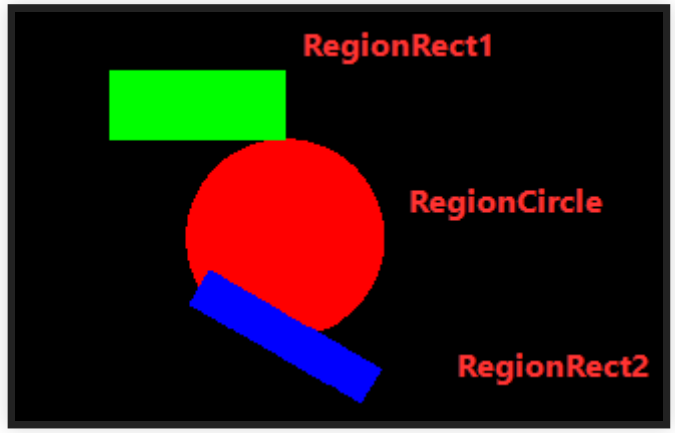


(Index)	0,	1,	2,	3,	4,	5,...
Row	[283,	283,	284,	284,	285,	285,
ColumnBegin	[527,	541,	527,	542,	528,	543,
ColumnEnd	[538,	543,	539,	543,	540,	543,

USAGE OF REGIONS

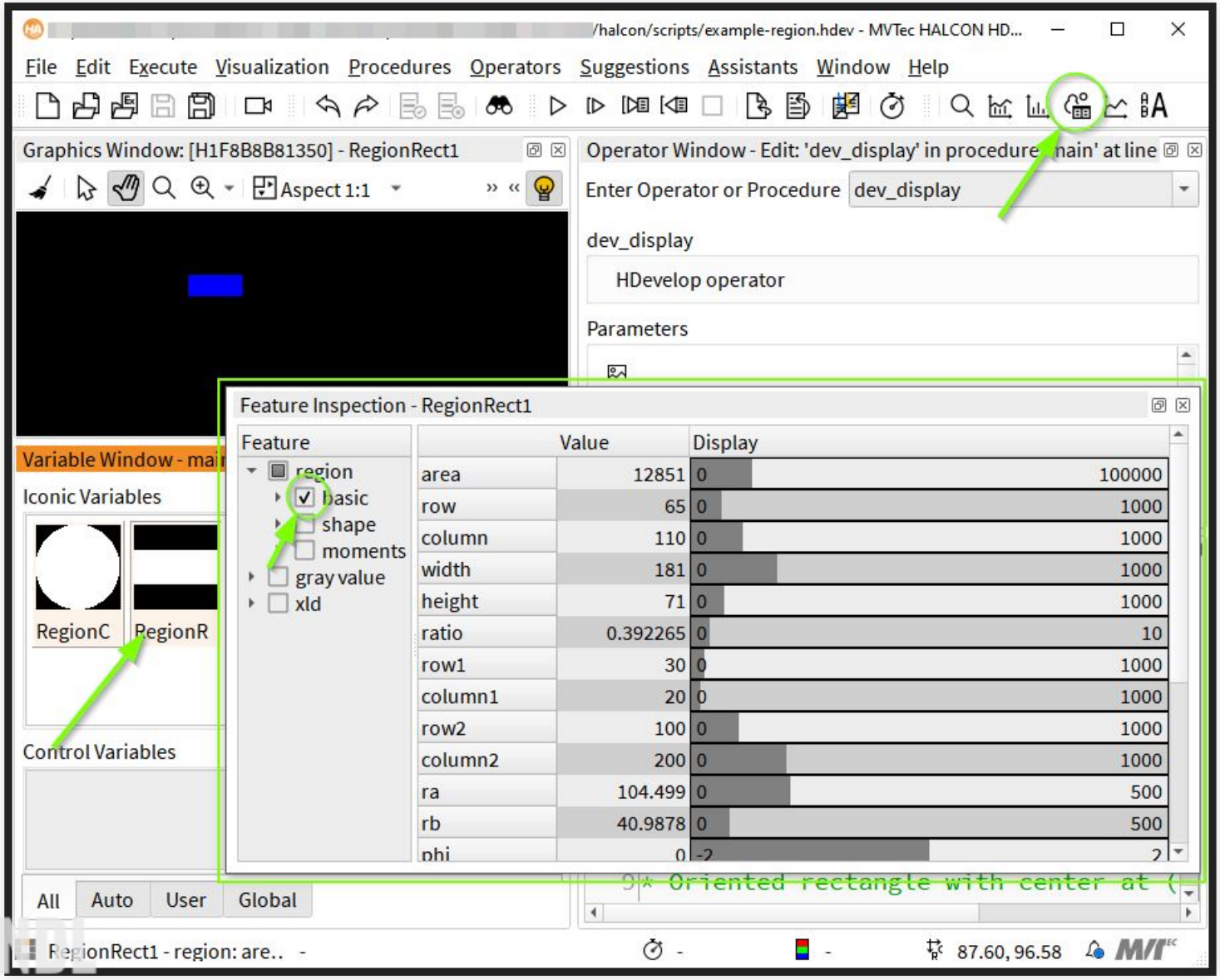
- Segmentation (see blob analysis)
- Image domain
  - Processing in restricted area of image
  - Speedup
- Find features (Feature extraction)

☞ GENERATE REGIONS



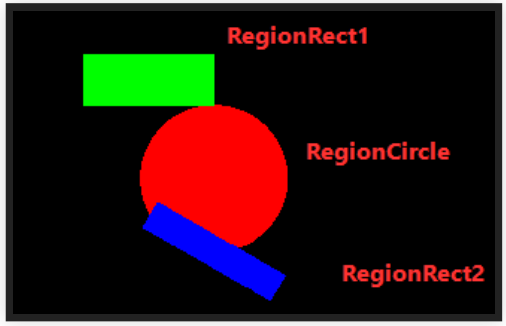

```
* Circle at (200, 200) with R=100.5
gen_circle (RegionCircle, 200, 200, 100.5)

* Axis-parallel rectangle from R1C1 to R2C2
gen_rectangle1 (RegionRect1, 30, 20, 100, 200)

* Oriented rectangle with center at (300,200)
* angle 30deg and half(!) axis lengths 100, 20
gen_rectangle2 (RegionRect2, 300, 200, rad(-30), 100, 20)
```

REGION FEATURES




HEINDL
SOLUTIONS

☞ ACCESS TO REGIONS

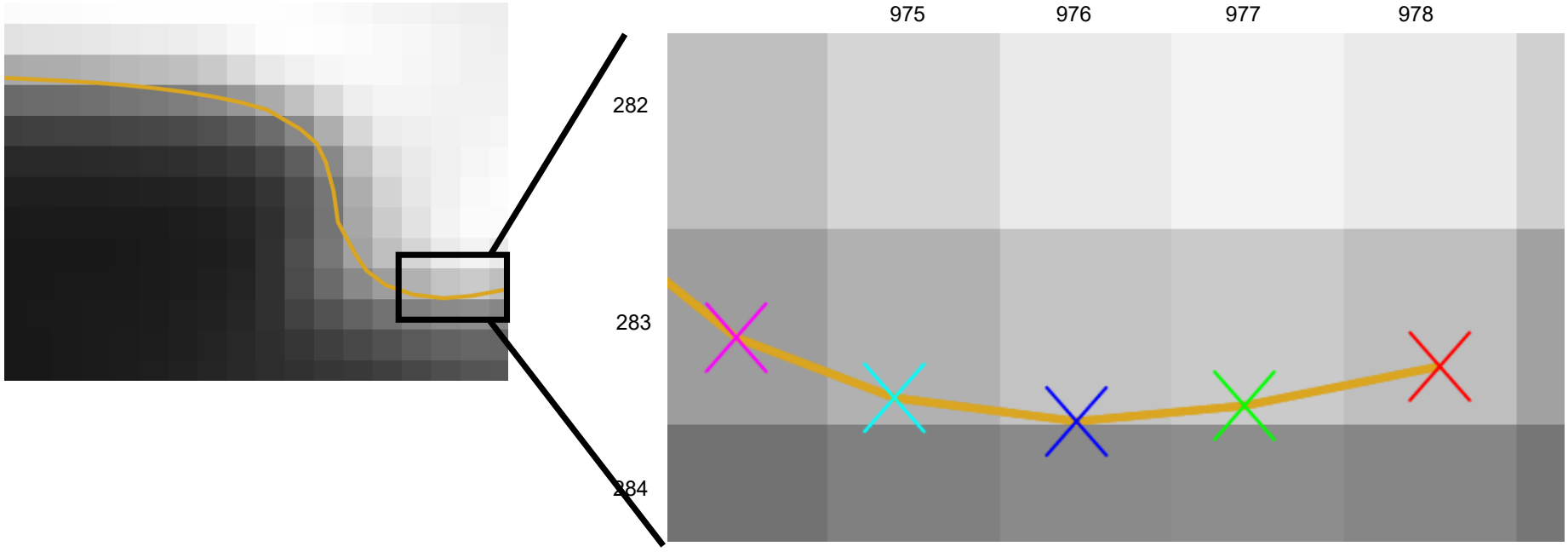



```
:
get_region_points (RegionRect1, RowsRect1, ColumnsRect1)
* RowsRect1 == [30,30,30,...]
* ColumnsRect1 == [20,21,22,...]

get_region_runs (RegionRect1, RunsRow, RunsColumnBegin, RunsColumnEnd
* RunsRow == [30,31,32,...]
* RunsColumnBegin == [20,20,20,...]
* RunsColumnEnd == [200,200,200,...]

area_center (RegionRect1, AreaRect1, __, __)
* AreaRect1 == 12851
```

CONTOURS / XLDS

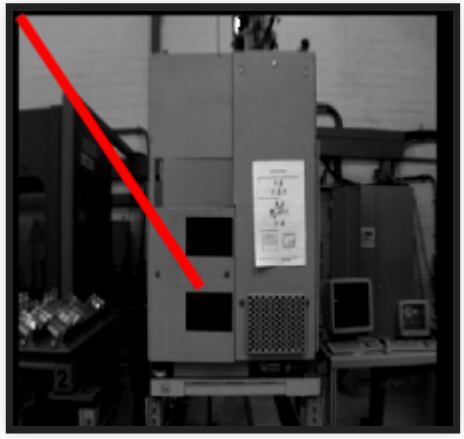
- XLD: eXtended Line Descriptor
- subpixel-precise values
- connected points, can be closed
- 0/0 in the center of a pixel

CONTOURS / XLDS



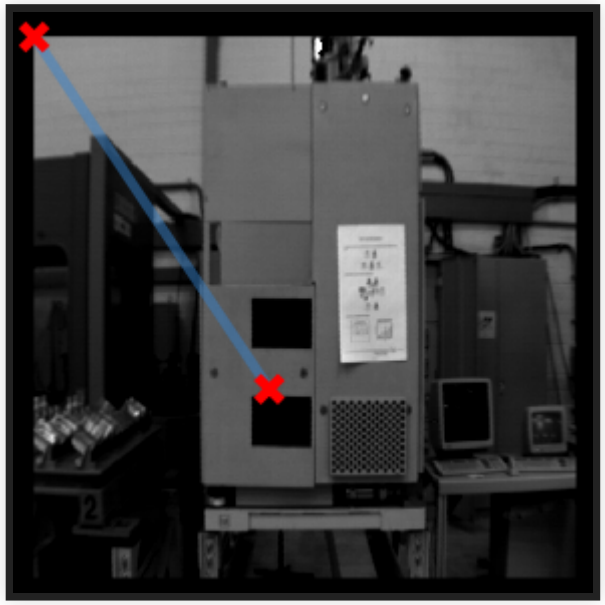

```
get_contour_xld (Contours, Row, Col)
    Row     [283.202, 283.403, 283.483, 283.363, ...
    Column  [978.060, 976.926, 975.950, 974.892, ...
```

GENERATE CONTOURS



```
* Read an example image
* just to have some background
read_image (Image, 'fabrik')

* A simple line contour from (0,0) to (333,222)
gen_contour_polygon_xld (Contour, [0,333], [0,222])
```

GET CONTOUR POINTS



```
:
* Get contour points
get_contour_xld (Contour, Rows, Columns)

* "Special" contour generating operator: Often
* used to visualize coordinates
gen_cross_contour_xld (Cross, Rows, Columns, 32, rad(45))
```

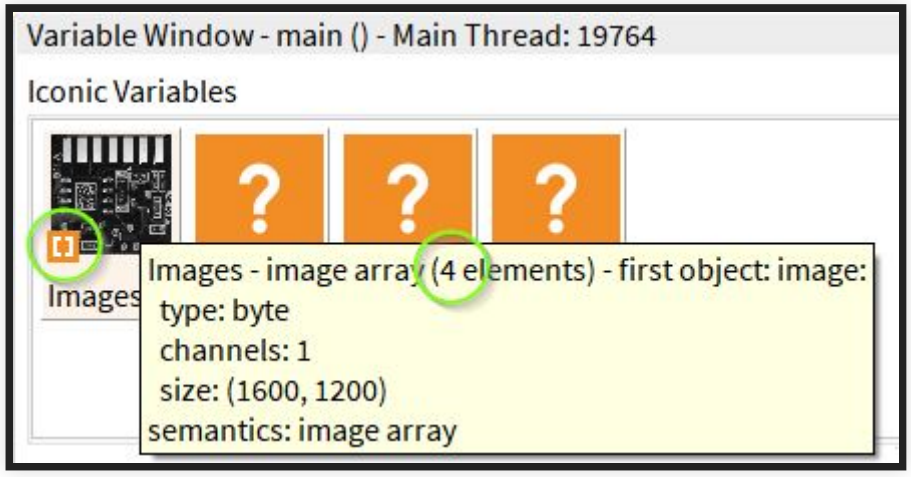
OBJECTS

- A object can contain multiple "sub-objects"

```
gen_empty_obj (EmptyObject)
concat_obj (EmptyObject, Image, MyObjects)
concat_obj (MyObjects, Region, MyObjects)
concat_obj (MyObjects, XldEdges, MyObjects)
```

```
4 sub-objects
image
    type: byte
    domain: full
    channels: 3
region
    area:111060
xld_cont
    num points:121
xld_cont
    num points:206
```

HANDLING OF OBJECTS




```
* Read the following 4 images into an object array:
* printer_chip/printer_chip_01.png
* printer_chip/printer_chip_02.png
* printer_chip/printer_chip_03.png
* printer_chip/printer_chip_04.png
read_image (Images, 'printer_chip/printer_chip_0' + [1,2,3,4])

* Count number of objects in the object array
count_obj (Images, NumberImages)
* NumberImages == 4
```

👉 HANDLING OF OBJECTS

```
:
* Access the first object (1-based index)
select_obj (Images, Image1, 1)
* Access the third object (1-based index)
select_obj (Images, Image3, 3)

* Concatenate Image1 and Image3
concat_obj (Image1, Image3, Imaged1And3)

* Concatenate Image1And3 and Images
concat_obj (Imaged1And3, Images, Imaged1And3AndImages)
```

CONTROL PARAMETERS

- **Integer(s)** (Handles, Pointer, Booleans)
- **Real(s)** (Floating point numbers, Double)
- **String(s)** (characters)
- **Handle(s)** (to windows, models, dictionaries, …)

CONTROL PARAMETERS

- Integer(s) (Handles, Pointer, Booleans)
  - `42`
  - `1585967204848`
  - `0`
  - `-1`
  - `true` / `false`

CONTROL PARAMETERS

- Real(s) (Floating point numbers, Double)
  - `3.141592`
  - `-5.3`
  - `1.#INF`

CONTROL PARAMETERS

- String(s) (characters)
  - Enclosed by single quotation marks
    `'string'`
  - typical escape seq.
    `'Line1\nLine2'`
  - Paths always with slashes
    `'C:/HALCON/images'`

CONTROL PARAMETERS

- String(s) (characters)
  - Prior to HALCON 18.11
    - Sequence of bytes
    - Interpretation depends on Locale settings
      - often Latin-1
  - since HALCON 18.05 / 18.11
    - Sequence of bytes interpreted as UTF-8

CONTROL PARAMETERS

- String(s) (characters)
  - since HALCON 18.05 / 18.11
    - Behavior of string operators can be adjusted with `set_system ('tuple_string_operator_mode', …)`
    - Behavior of "UTF-8 at language interfaces" can be adjusted with `SetHcInterfaceStringEncodingIsUtf8`

CONTROL PARAMETERS

- String(s) (characters)
  - Prior to HALCON 18.11

```
HeartArrow := '♥➞'
Length     := strlen(HeartArrow)
* Length == 6
Arrow := HeartArrow{1}
* Arrow == '\x9D'
```

CONTROL PARAMETERS

- String(s) (characters)
  - since HALCON 18.05 / 18.11

```
HeartArrow := '❤➞'
Length     := strlen(HeartArrow)
* Length == 2
Arrow := HeartArrow{1}
* Arrow == '➞'
```

CONTROL PARAMETERS

- **Handle(s)**
  - **Prior to** HALCON 18.05:
    - Data type: Control tuple containing an integer
    - but **semantic interpretation**: Pointer to complex data structure (model)

CONTROL PARAMETERS

- since HALCON 18.05:
  - `H205DB005090`
  - Reference to complex data structure (Model, Window, …)
  - **Ref-Counted**
    Automated `clear` if no longer used
  - Keeps semantic type in tuple operations ➞ better inspection of models

CONTROL PARAMETERS

- Can contain **multiple** and **different** elements
  `A := [1, 2, 'string', 3.14]`

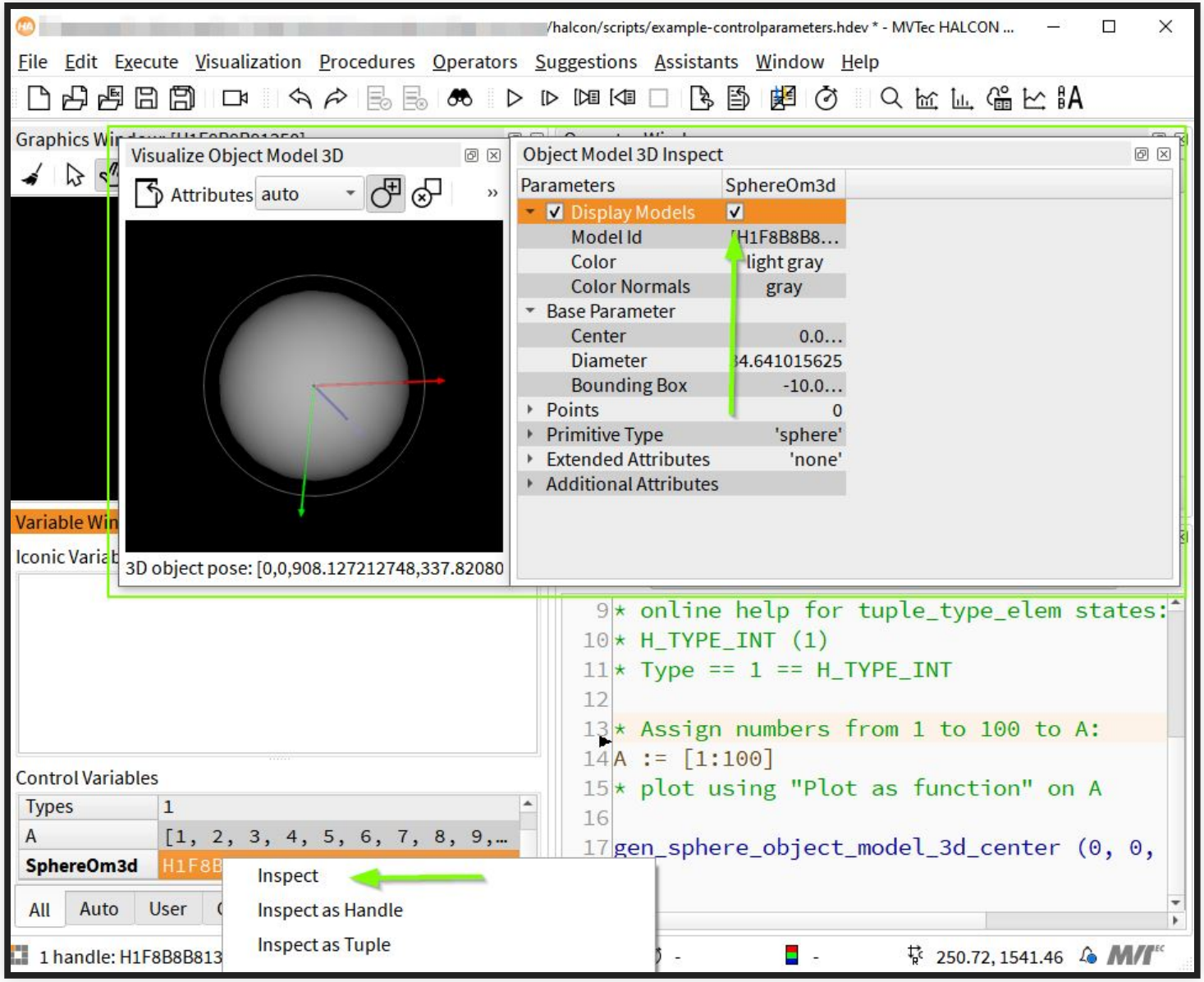
👉 CONTROL PARAMETERS: TYPES

```
* Assign an integer to variable MyTuple
MyTuple := 42

* get type of elements in MyTuple
Type := type_elem(MyTuple)
* Type == 1
* same:
tuple_type_elem (Type, Types)
* online help for tuple_type_elem states:
* H_TYPE_INT (1)
* Type == 1 == H_TYPE_INT
```

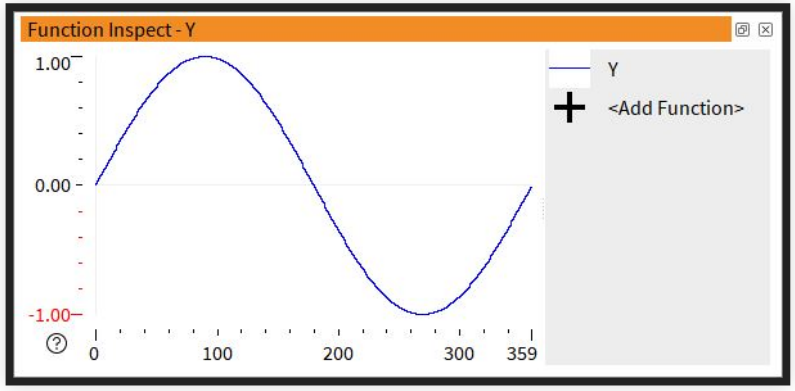
👉 CONTROL PARAMETERS: INSPECTION

```
* Generate a (3d) sphere at (0,0,0)
gen_sphere_object_model_3d_center (0, 0, 0, 10, SphereOm3d)

⋮
```

CONTROL PARAMETERS: INSPECTION



☞ CONTROL PARAMETERS: INSPECTION




```
* Assign numbers from 0 to 359 to X
X := [0:359]
* plot using "Plot as function" on X

Y := sin(rad(X))
* plot Sine using "Plot as function" on Y
```

SIGNATURES OF OPERATORS

C:/projects/barcode.hdev

Graphics Window

Operator Window

find_bar_code

Image

SymbolRegions

BarCodeHandle

CodeType

DecodedDataStrings

Variable Window

```
BarCodeHandle        H205DB005060
DecodedDataStrings   '9783893191727'
```

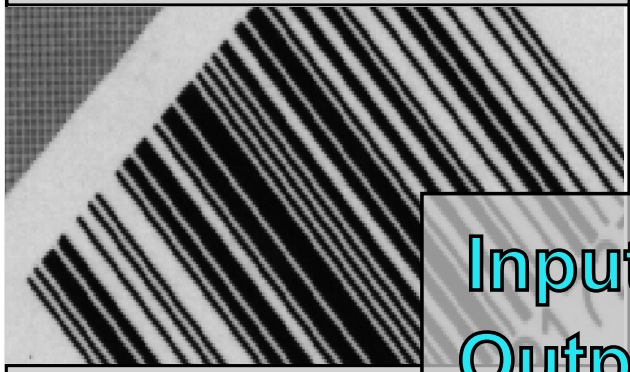
Program Window

```
read_image (Image, 'barcode/ean13/ean1311.png')
dev_set_draw ('margin')
dev_set_line_width (3)
create_bar_code_model ([], [], BarCodeHandle)
find_bar_code (Image, SymbolRegions,BarCodeHandle, 'auto', DecodedDataStrings)
clear_bar_code_model (BarCodeHandle)
```

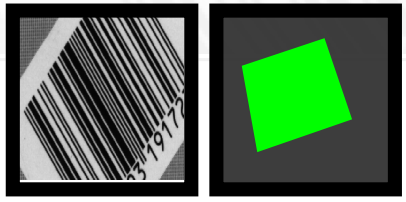
SIGNATURES OF OPERATORS

C:/projects/barcode.hdev

Graphics Window



Variable Window



BarCodeHandle	H205DB005060
DecodedDataStrings	'9783893191727'

Input objects
Output objects
Input controls
Output controls

Operator Window

find_bar_code

Image

SymbolRegions

BarCodeHandle

CodeType

DecodedDataStrings

```
read_image (Image, 'barcode/ean13/ean1311.png')
create_bar_code_model ([], [], BarCodeHandle)
find_bar_code (Image, SymbolRegions,BarCodeHandle, 'auto', DecodedDataStrings)
clear_bar_code_model (BarCodeHandle)
```

HEINDL SOLUTIONS

COORDINATE SYSTEMS

- Coordinates **X** / **Column**
  left ➞ right 0 .. (Width - 1)
- Coordinates **Y** / **Row**
  top ➞ bottom 0 .. (Height - 1)
- **Angles**
  Counterclockwise direction
  Degrees **and** Radians

VECTORS

- Combination of iconic objects **or** control tuples
  ```
  Vi := { Img1, Img2, Region, XLD }
  Vc := { [1, 2, 'string'], 3.14}
  ```
- Can be **multidimensional**, i.e. vectors can contain vectors
- Dimensions are deduced implicitly
- Can be confusing: `Vc.at(0)[0]`

VECTORS

- Vectors themselves cannot be used for operator calls
- Conversion from tuple with `convert_tuple_to_vector_1d`
- Conversion to tuple with `convert_vector_to_tuple`

MODELS IN HALCON

- Create models using `create_…` resp. `gen_…` resp. `open_…`
- Usage, for example
  - `find_bar_code (…)`
  - `get_bar_code_object (…)`
- Clean up models using `clear_…` resp. `close_…`

GENERIC PARAMETERS

```
GenParamNames  := []
GenParamValues := []
GenParamNames  := [GenParamNames, 'num_scanlines']
GenParamValues := [GenParamValues, 10]
GenParamNames  := [GenParamNames, 'orientation']
GenParamValues := [GenParamValues, 0.0]
create_bar_code_model (GenParamNames, GenParamValues, BarCode
```

DICTIONARIES

- key/value store
- arbitrary tuple and/or iconic object data within a single container
- increasingly also used for operator calls

DICTIONARIES

- Organize data

```
list_image_files ('barcode/25industrial', 'default', [], ImageFiles)
InputFilesDict := []
for IndexImageFile0 := 0 to |ImageFiles|-1 by 1
   ImageFile := ImageFiles[IndexImageFile0]
   read_image (Image, ImageFile)
   create_dict (InputFileDict1)
   set_dict_tuple (InputFileDict1, 'image_file', ImageFile)
   set_dict_object (Image, InputFileDict1, 'image')
   Exposure := number(regexp_match(ImageFile,'exposure([0-9]+)'))
   set_dict_tuple (InputFileDict1, 'exposure', Exposure)
   InputFilesDict := [InputFilesDict, InputFileDict1]
endfor
* ...
for Index0 := 0 to |InputFilesDict|-1 by 1
    InputFileDict1 := InputFilesDict[Index0]
    get_dict_object (Image, InputFileDict1, 'image')
    dev_display (Image)
    get_dict_tuple (InputFileDict1, 'image_file', ImageFile)
    dev_disp_text (ImageFile, 'window', 'top', 'left', 'black', [], [
    stop ()
```

```
endfor
```

DICTIONARIES

- Usage in some modern operator calls

```
create_dict (Params)
set_dict_tuple (Params, 'remove_outer_edges', 'true')
set_dict_tuple (Params, 'max_gap', 150)
:
find_box_3d (…, …, …, …, Params, …, …, BoxInformation)
get_dict_tuple (BoxInformation, 'sampled_scene', OM3DSampledScene)
get_dict_tuple (BoxInformation, 'sampled_edges', OM3DSampledEdges)
```

ARRAYS

	Iconic	Control	Vector
Start index	1	0	0
Creation	`gen_empty_obj` `concat_obj`	`A:=` `[1,2,3]`	`Vect:={1,2,3}`
Size	`count_obj`	`|A|`	`Vect.length()`
Access	`select_obj`	`A[0]`	`Vect.at(0)`

INTERACTIVE DATATYPE EXPLORER

- http://www.heindl-solutions.com/interactive-halcon-datatype-explorer.en.html

USE HALCON DATA STRUCTURES IN AN EXAMPLE

USE HALCON DATA STRUCTURES IN AN EXAMPLE

- get list of images in directory `barcode/mixed`
  hint: **`list_image_files`**
- iterate over images: show images and their filenames
  hint: **`for`**, **`read_image`** , **`dev_display`** , **`dev_disp_text`**
- **`stop()`** after each image

USE HALCON DATA STRUCTURES IN AN EXAMPLE

- …
- decode bar codes in `'auto'` mode
  hint:
  - `create_bar_code_model`
  - `find_bar_code` (N times)
  - `clear_bar_code_model`
- `stop()` and `dev_display()` only in Debug mode
  hint: introduce variable Debug

USE HALCON DATA STRUCTURES IN AN EXAMPLE

- …
- perform runtime measurements:
  - `dev_update_off`
  - separate `read_image` from decoding
    - read all images in one call to `read_image`
  - measure runtime
    hint: `count_seconds (Then)`, …, `count_seconds (Now)`

EXCERPT OF TRAINING COURSE - DATA STRUCTURES IN HALCON

HEINDL SOLUTIONS

andreas@heindl-solutions.com
http://www.heindl-solutions.com

Heindl Solutions is your **MVTec Certified Integration Partner**. We create **HALCON machine vision solutions** and **complete GUI applications**.

HEINDL SOLUTIONS