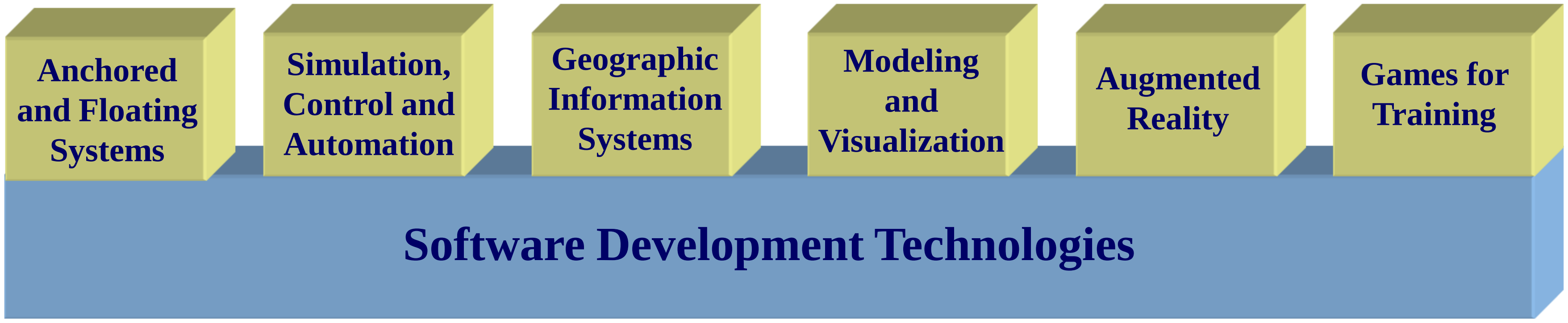
Tecgraf Development Tools: IUP, CD and IM

Antonio Scuri

scuri@tecgraf.puc-rio.br

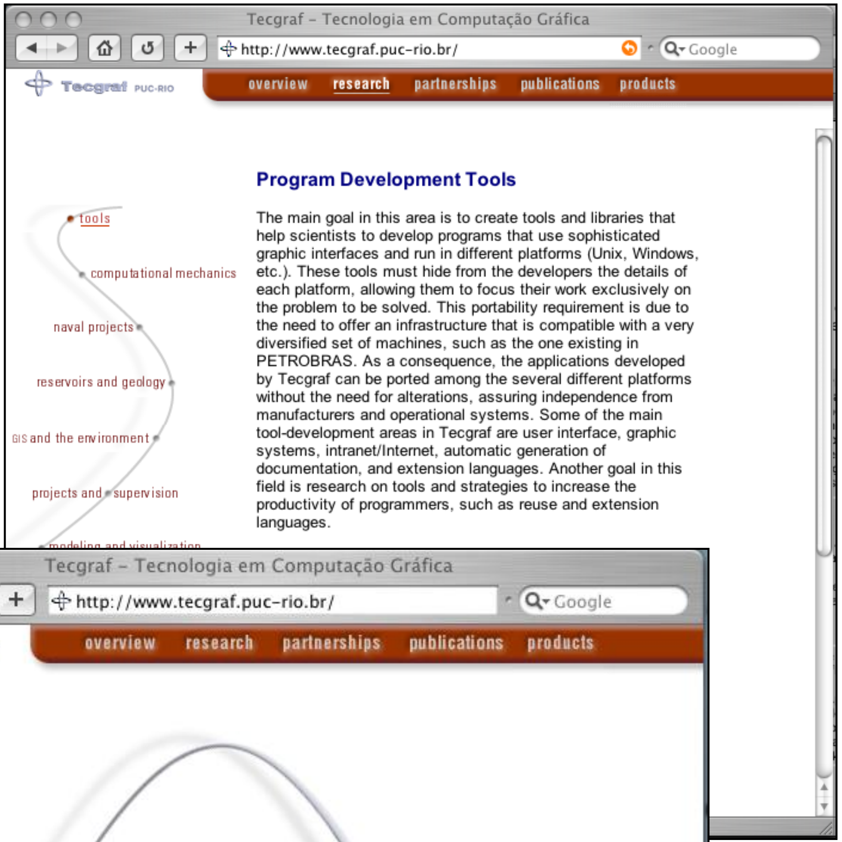
The Laboratory

Tecgraf
PUC-RIO


Anchored and Floating Systems
Simulation, Control and Automation
Geographic Information Systems
Modeling and Visualization
Augmented Reality
Games for Training
Software Development Technologies

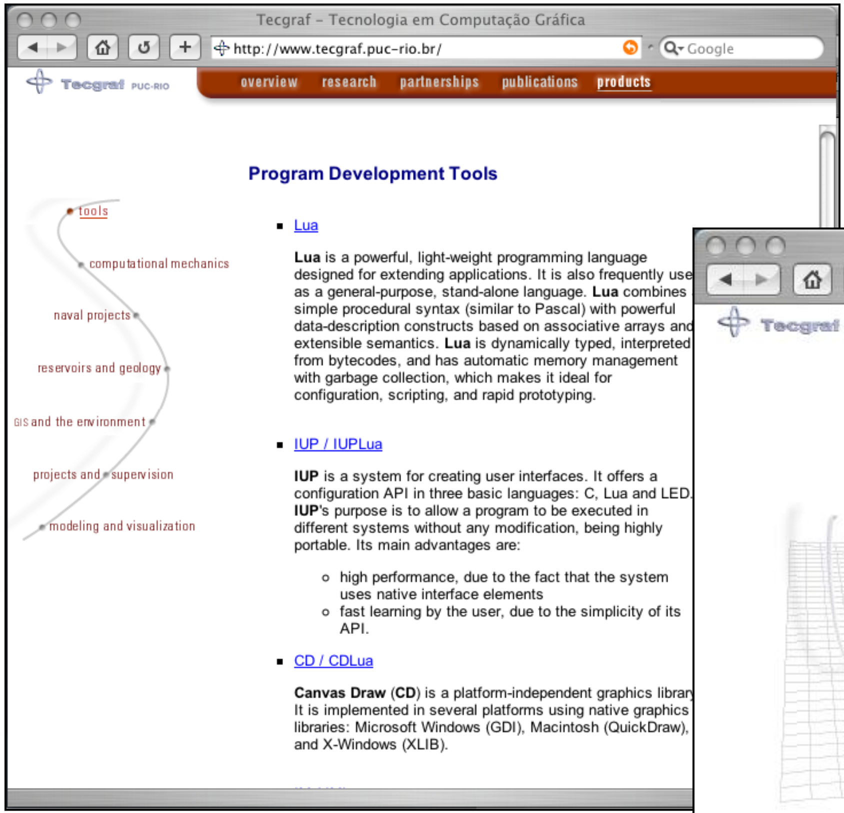
For more Information

http://www.tecgraf.puc-rio.br/



Program Development Tools

The main goal in this area is to create tools and libraries that help scientists to develop programs that use sophisticated graphic interfaces and run in different platforms (Unix, Windows, etc.). These tools must hide from the developers the details of each platform, allowing them to focus their work exclusively on the problem to be solved. This portability requirement is due to the need to offer an infrastructure that is compatible with a very diversified set of machines, such as the one existing in PETROBRAS. As a consequence, the applications developed by Tecgraf can be ported among the several different platforms without the need for alterations, assuring independence from manufacturers and operational systems. Some of the main tool-development areas in Tecgraf are user interface, graphic systems, intranet/Internet, automatic generation of documentation, and extension languages. Another goal in this field is research on tools and strategies to increase the productivity of programmers, such as reuse and extension languages.



Program Development Tools

- Lua

  Lua is a powerful, light-weight programming language designed for extending applications. It is also frequently use as a general-purpose, stand-alone language. **Lua** combines simple procedural syntax (similar to Pascal) with powerful data-description constructs based on associative arrays and extensible semantics. **Lua** is dynamically typed, interpreted from bytecodes, and has automatic memory management with garbage collection, which makes it ideal for configuration, scripting, and rapid prototyping.

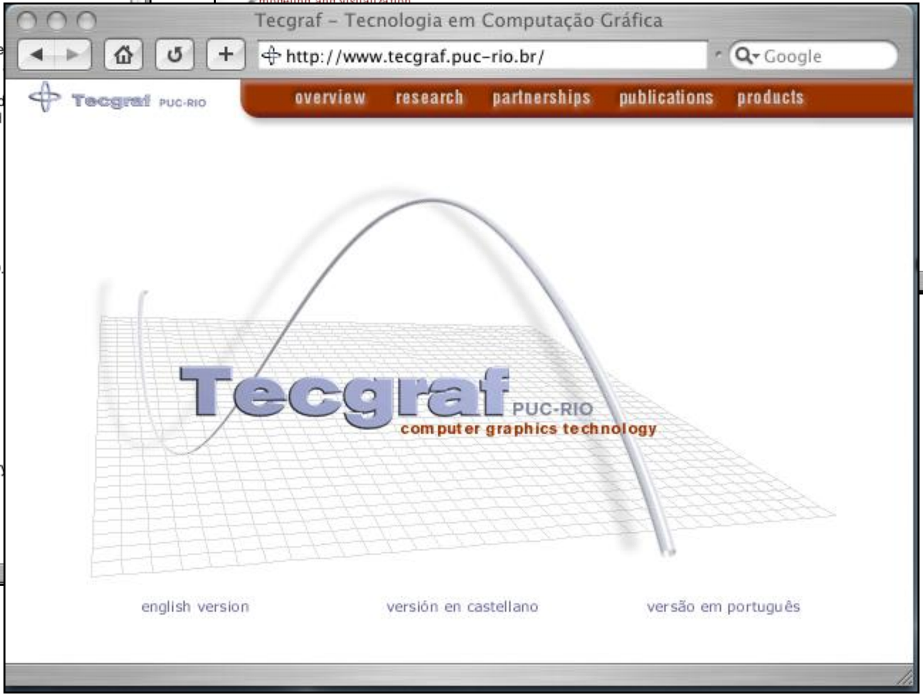
- IUP / IUPLua

  IUP is a system for creating user interfaces. It offers a configuration API in three basic languages: C, Lua and LED. **IUP**'s purpose is to allow a program to be executed in different systems without any modification, being highly portable. Its main advantages are:

  - high performance, due to the fact that the system uses native interface elements
  - fast learning by the user, due to the simplicity of its API.

- CD / CDLua

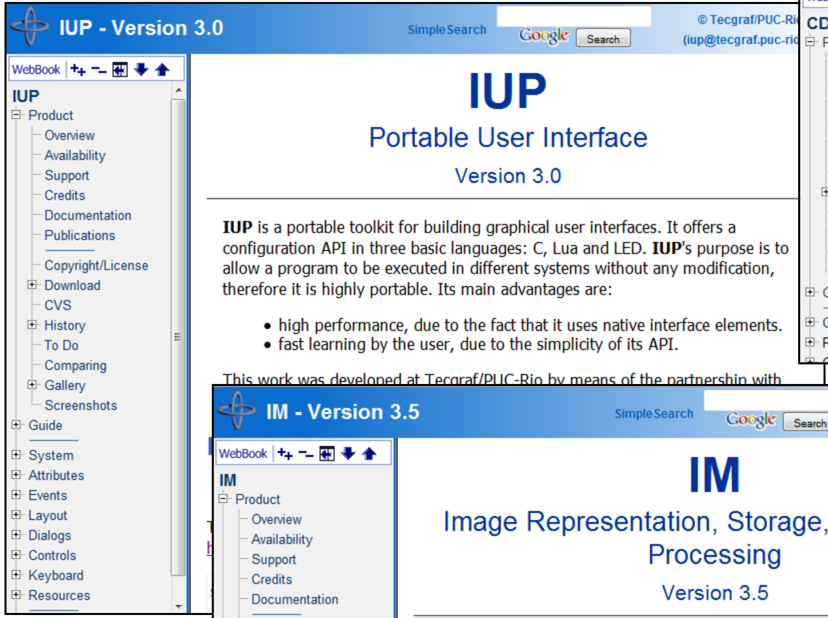
  Canvas Draw (**CD**) is a platform-independent graphics librar It is implemented in several platforms using native graphics libraries: Microsoft Windows (GDI), Macintosh (QuickDraw), and X-Windows (XLIB).



The Tools

http://www.tecgraf.puc-rio.br/cd/

http://www.tecgraf.puc-rio.br/iup/



IUP

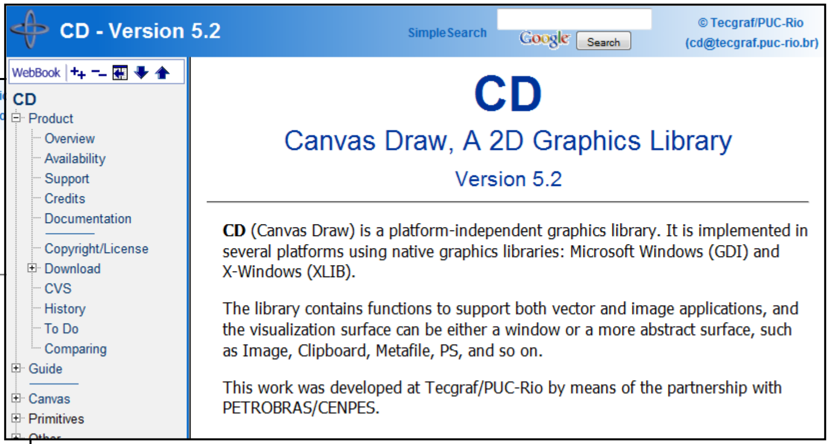
Portable User Interface

Version 3.0

IUP is a portable toolkit for building graphical user interfaces. It offers a configuration API in three basic languages: C, Lua and LED. **IUP**'s purpose is to allow a program to be executed in different systems without any modification, therefore it is highly portable. Its main advantages are:

- high performance, due to the fact that it uses native interface elements.
- fast learning by the user, due to the simplicity of its API.

This work was developed at Tecgraf/PUC-Rio by means of the partnership with



CD

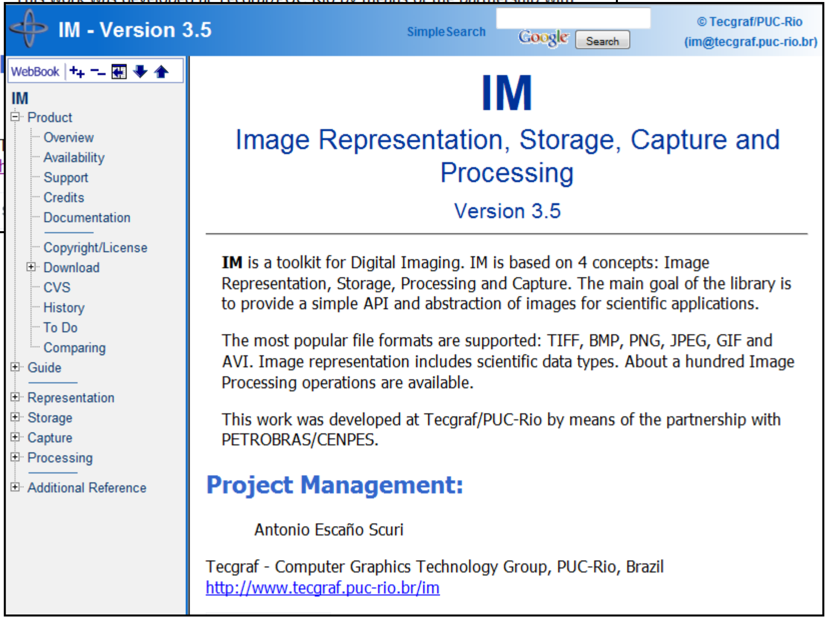
Canvas Draw, A 2D Graphics Library

Version 5.2

CD (Canvas Draw) is a platform-independent graphics library. It is implemented in several platforms using native graphics libraries: Microsoft Windows (GDI) and X-Windows (XLIB).

The library contains functions to support both vector and image applications, and the visualization surface can be either a window or a more abstract surface, such as Image, Clipboard, Metafile, PS, and so on.

This work was developed at Tecgraf/PUC-Rio by means of the partnership with PETROBRAS/CENPES.



IM

Image Representation, Storage, Capture and Processing

Version 3.5

IM is a toolkit for Digital Imaging. IM is based on 4 concepts: Image Representation, Storage, Processing and Capture. The main goal of the library is to provide a simple API and abstraction of images for scientific applications.

The most popular file formats are supported: TIFF, BMP, PNG, JPEG, GIF and AVI. Image representation includes scientific data types. About a hundred Image Processing operations are available.

This work was developed at Tecgraf/PUC-Rio by means of the partnership with PETROBRAS/CENPES.

Project Management:

Antonio Escaño Scuri

Tecgraf - Computer Graphics Technology Group, PUC-Rio, Brazil
http://www.tecgraf.puc-rio.br/im

http://www.tecgraf.puc-rio.br/im/

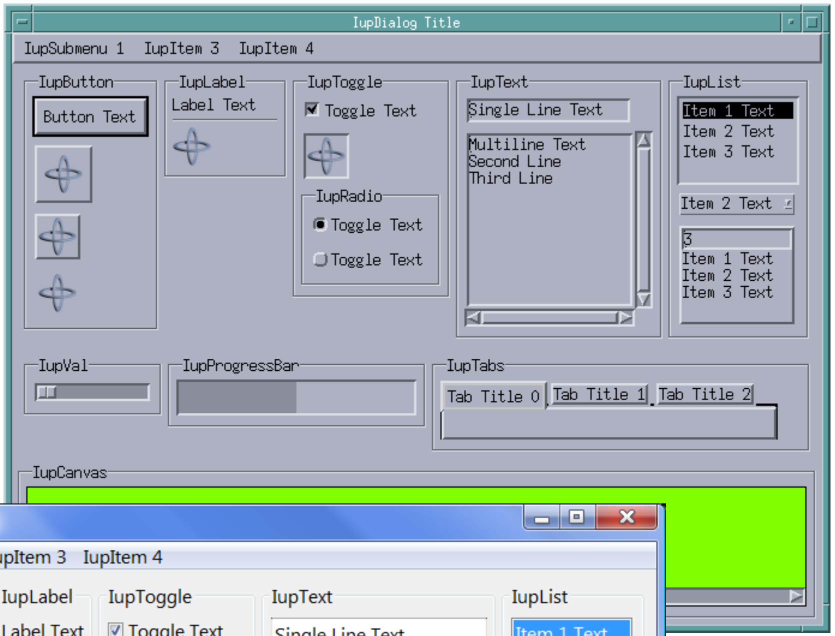
Webbook
Tecmake
LuaBinaries

General Characteristics

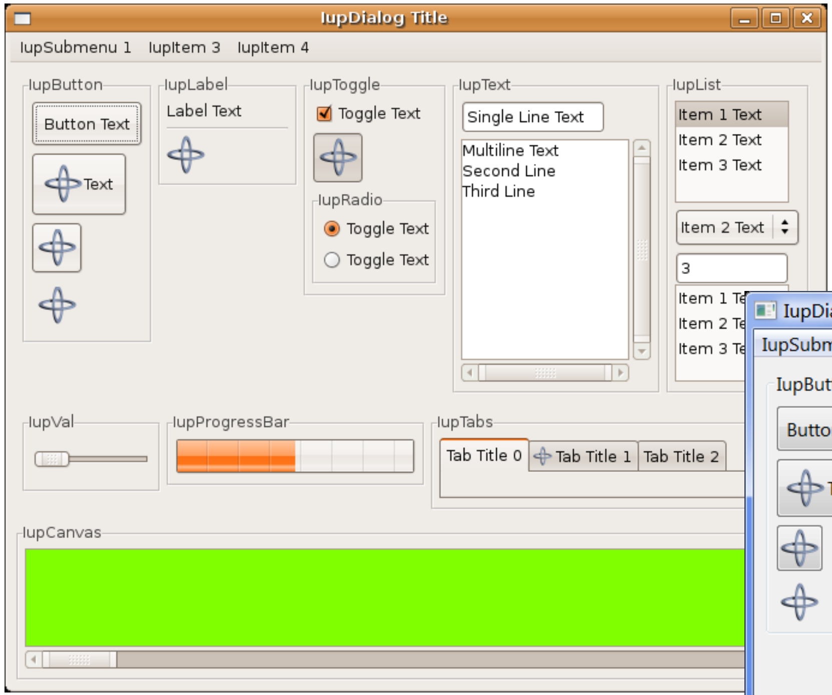
- Abstraction Model
- Portability (Desktop)
- Performance (native system resources)
- Simple but Powerful
- Free and Open Source (same license of Lua)
- Binaries for many platforms ready to Download
- API in C and in Lua
- Extensive Documentation with Examples
- Backward Compatibility Policy
- Best Learning Curve (*)

IUP - Portable User Interface

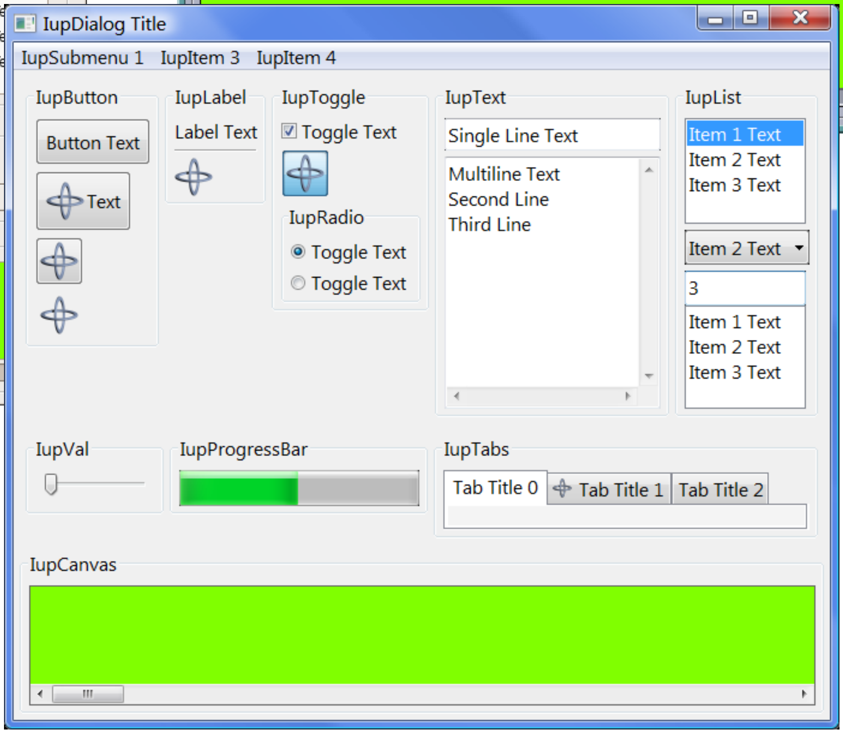
Motif in MWM



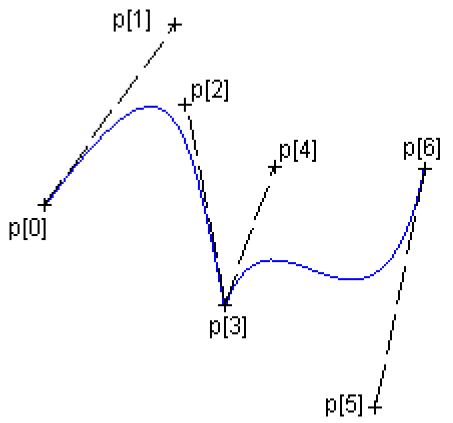
GTK in Gnome

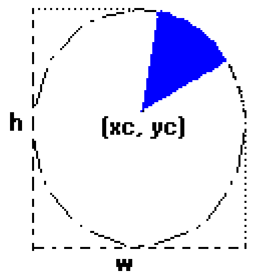


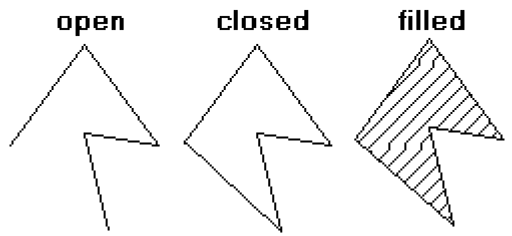
Windows Vista

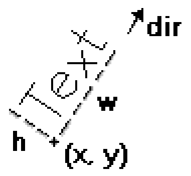


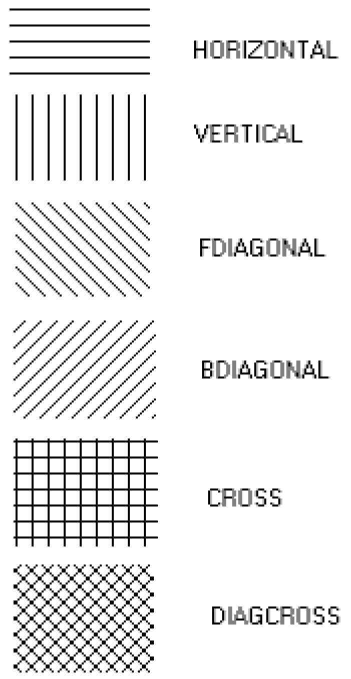
CD – A 2D Graphics Library

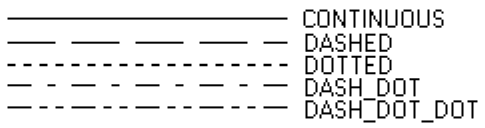


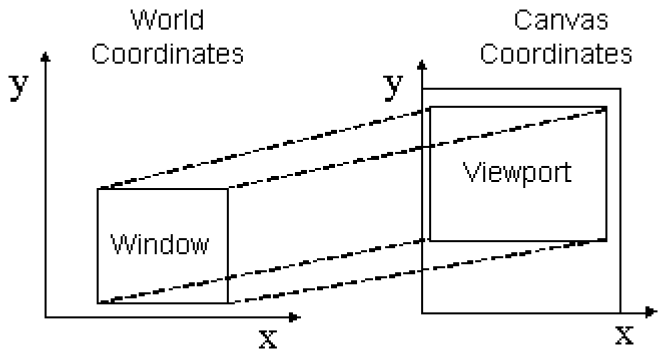


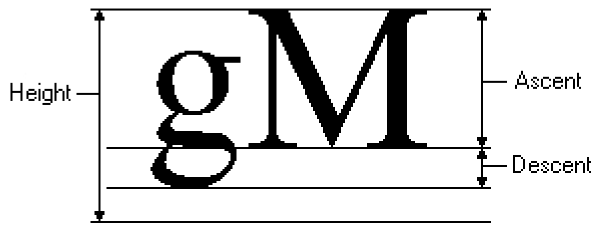




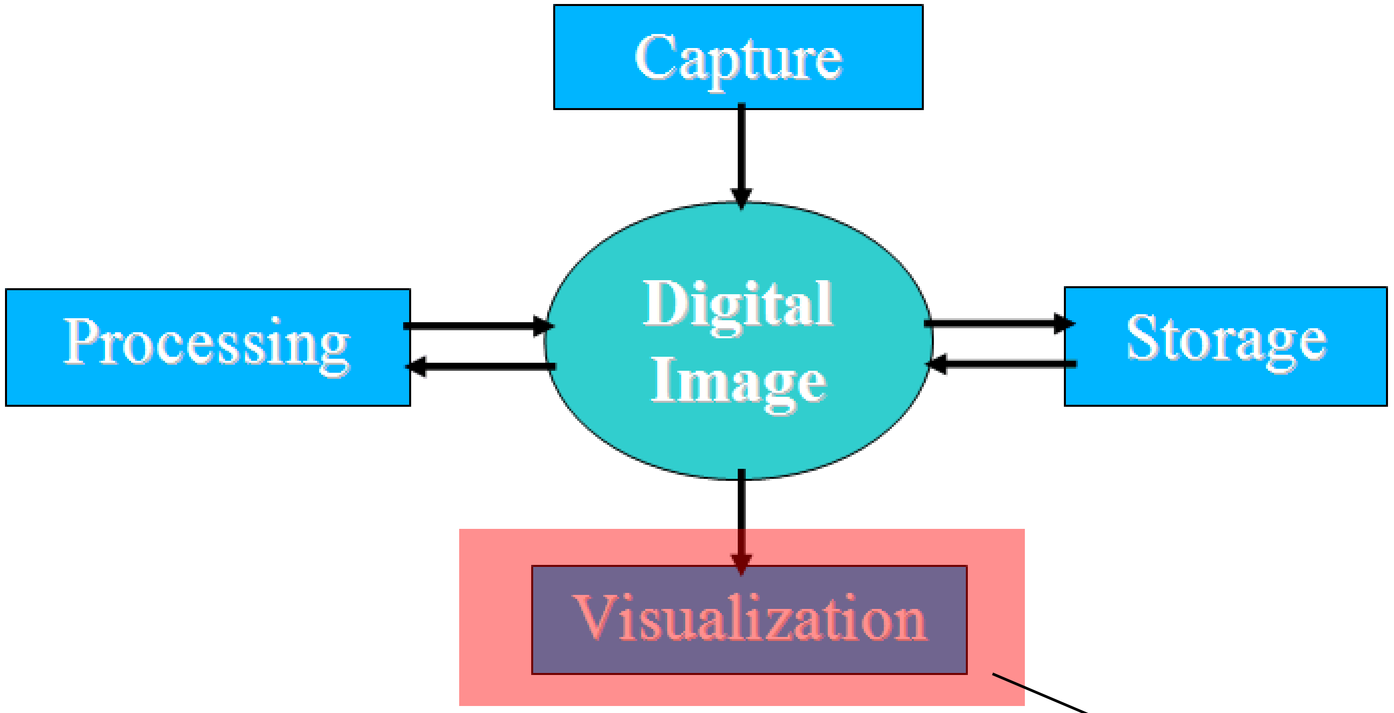






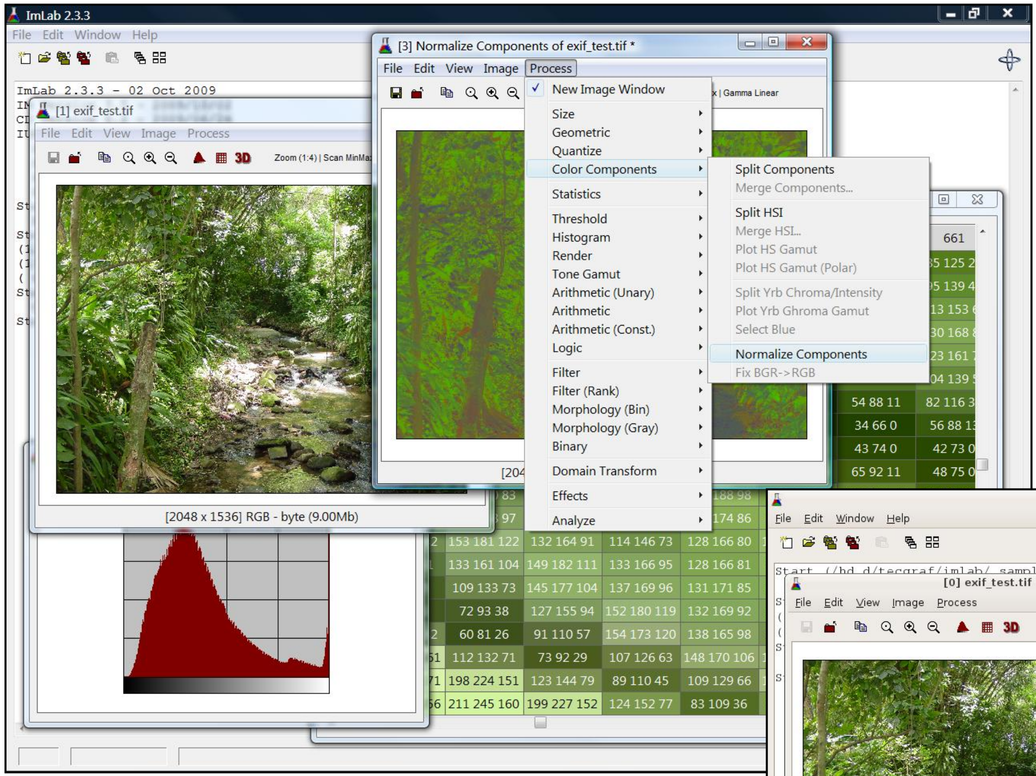


IM – Image Representation, Storage, Capture and Processing

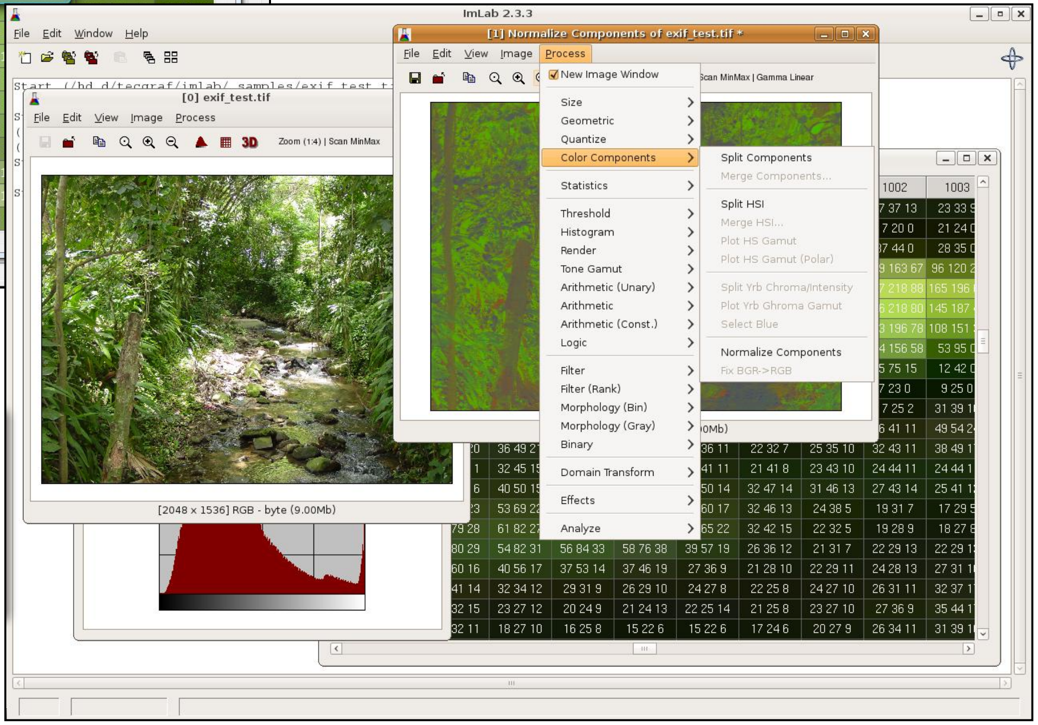


CD

Example of a Complete Application



Windows Vista



GTK in Gnome

Differential Characteristics

- **IUP**: abstract layout. Dialog construction without manually position of controls.
- **CD**: same API for several types of Canvases (Window, Printer, Clipboard, Image, Postscript)
- **IM**: focus in scientific applications but flexible to serve any kind of application.

Binding Lua

- Initialization

  `require"imlua"` (LUA_CPATH)

  `require"cdlua"`

  `require"iuplua"`

- General Rules
  - Functions: `imXxx => im.Xxx`
  - Definitions: `IM_XXX => im.XXX`
  - Methods: `imFileXXX(ifile,... => ifile:XXX(...`

- Special Constructors for IUP

Binding Lua – IUP Controls

- Metatable:

  -
    ```
    IupSetAttribute(label, "TITLE", "test")    =>
            label.title = "test"
    ```

  -
    ```
    title = IupGetAttribute(label, "TITLE")    =>
            title = label.title
    ```

  -
    ```
    IupSetCallback(button, "ACTION", button_action_cb);   =>
            function button:action() ... end
    ```
- Constructors:

  -
    ```
    IupButton("test")    =>
            iup.button{title = "test", alignment="acenter"}
    ```

  -
    ```
    IupHbox(bt1, bt2, NULL)   =>
            iup.hbox{bt1, bt2, margin="10x10"}
    ```

Examples IM - Indexing

```
require"imlua"

local filename = "lena.jpg"
local image = im.FileImageLoad(filename)

local r = image[0]
local g = image[1]
local b = image[2]

for row = 0, image:Height() - 1, 10 do
   for column = 0, image:Width() - 1, 10 do
        r[row][column] = 0
        g[row][column] = 0
        b[row][column] = 0
   end
end

image:Save("lena_indexing.bmp", "BMP")
```

Examples IM - Processing

```
image, err = im.FileImageLoad(file_name);

if (err and err ~= im.ERR_NONE) then
  error(err_msg[err+1])
end

local new_image = im.ProcessCropNew(image, x1, x2, y1, y2)
image:Destroy()

new_image:Save(file_name, "JPEG");
new_image:Destroy()
```

Examples IM - Analyze

```
local image = im.FileImageLoad(filename)

local gray = im.ImageCreate(image:Width(), image:Height(), im.GRAY,
   image:DataType())
im.ConvertColorSpace(image, gray)
gray:Save("lena_gray.jpg", "JPEG")

local binary = im.ImageCreate(image:Width(), image:Height(), im.BINARY,
   image:DataType())
im.ProcessSliceThreshold(gray, binary, 0, 128)
binary:Save("lena_binary.jpg", "JPEG")

local region = im.ImageCreate(image:Width(), image:Height(), im.GRAY,
   im.USHORT)
local count = im.AnalyzeFindRegions(binary, region, 4, 1)
print("regions: ", count)
```

Examples CD - Postscript

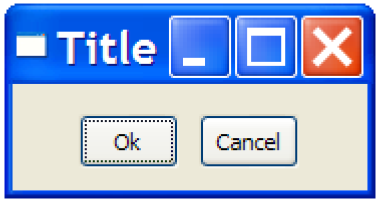
```
local canvas = cd.CreateCanvas(cd.PS, "cdtest.ps")

canvas:MarkSize(40)
canvas:Mark(x, y)

canvas:Font("Courier", cd.PLAIN, 12)
canvas:TextAlignment(cd.CENTER)
canvas:Text(x, y, text)

local xmin, xmax, ymin, ymax = canvas:GetTextBox(x, y, text)
canvas:Rect(xmin, xmax, ymin, ymax)

canvas:Kill()
```

Examples CD with IM

```
require"imlua"
require"cdlua"
require"cdluaim"

local image = im.ImageCreate(500, 500, im.RGB, im.BYTE)
local canvas = image:cdCreateCanvas()  -- Creates a CD_IMAGERGB canvas

canvas:Activate()
canvas:Background(cd.EncodeColor(255, 255, 255))
canvas:Clear()
fgcolor = cd.EncodeColor(255, 0, 0) -- red
fgcolor = cd.EncodeAlpha(fgcolor, 50) -- semi transparent
canvas:Foreground(fgcolor)
canvas:Font("Times", cd.BOLD, 24)
canvas:Text(100, 100, "Test")
canvas:Line(0,0,100,100)
canvas:Kill()

image:Save("new.bmp", "BMP")
```

Examples IUP – Abstract Layout

```
dlg = iup.dialog
{
  iup.hbox
  {
    iup.fill{},
    iup.button{title="Ok",size="40"},
    iup.button{title="Cancel",size="40"},
    iup.fill{}
   ;margin="10x10", gap="10"
  }
 ;title="Test"
}

dlg:show()
```



Callbacks in Lua

```
btok = dlg[1][2]
btcancel = dlg[1][3]

btok.bgcolor = "255 0 0"

function btok:action()
  local aux = self.fgcolor
  self.fgcolor = self.bgcolor
  self.bgcolor = aux
End

function btcancel:action()
  return iup.CLOSE
end
```

Example with IUP, CD and IM

```
image = im.FileImageLoad("flower.jpg")
cnv = iup.canvas{rastersize = image:Width().."x"..image:Height(), border
   = "NO"}
cnv.image = image -- store the new image in the IUP canvas as an
   attribute

function cnv:map_cb()
  -- the CD canvas can only be created when the IUP canvas is mapped
  self.canvas = cd.CreateCanvas(cd.IUP, self)
end

function cnv:action()
  -- called everytime the IUP canvas needs to be repainted
  self.canvas:Activate()
  self.canvas:Clear()
  self.image:cdCanvasPutImageRect(self.canvas, 0, 0, 0, 0, 0, 0, 0, 0)
end

dlg = iup.dialog{cnv}
dlg:show()
```

Antonio Scuri

scuri@tecgraf.puc-rio.br