JavaVis
A computer vision library in Java

http://javavis.sourceforge.net

Contents

- Features of JavaVis
- Three applications:
  - JavaVis2D
  - JavaVis3D
  - Desktop

Features of JavaVis

- Written in Java
- Computer vision/image processing library
- Free software. Open code
- It has more than 60 computer vision algorithms (i.e. Canny, Nitzberg, morphological operators, etc.)
- Teaching oriented, but can be used in research
- *Traditional* image processing, 3D processing and desktop

Frameworks

- JavaVis incoporates three frameworks:
  - JavaVis2d is the classic framework, for image processing
  - JavaVis3d allows to manage 3D images, define by 3D points, with or without color information
  - JavaVisDesktop is an application which allows to visualize partial results, oriented for teaching tasks

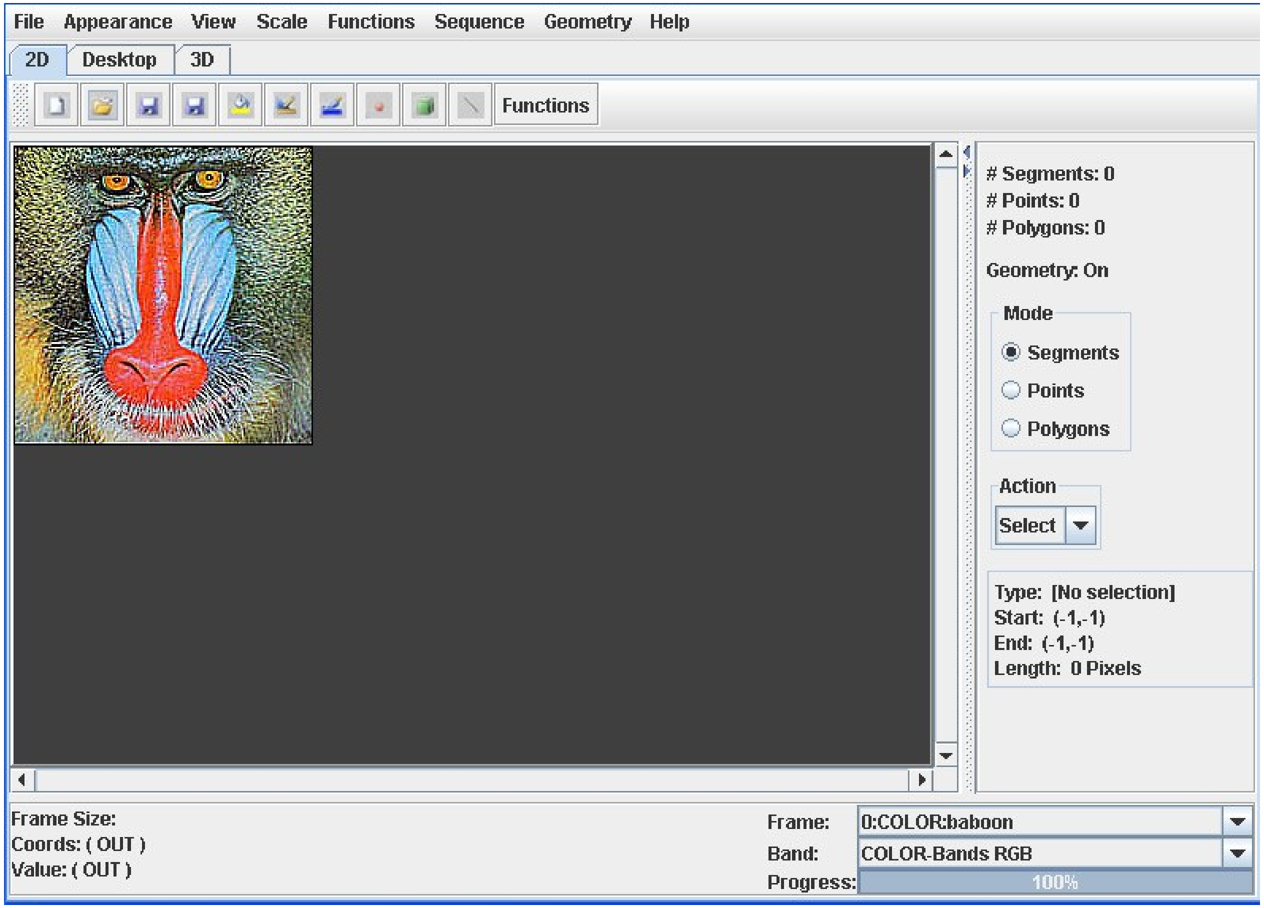
Installation

- JavaVis needs Java Sun JDK 1.5 or greater. Incorporates: autoboxing, enums, and all the new features in JDK1.5
- Download JavaVis from: http://sourceforge.net/projects/javavis http://javavis.sourceforge.net
- Decompress it
- It is prepared to work with Eclipse. Just import the project
- There exists an Ant task for execution, but we can compile and execute just including all the libraries in the *lib* folder

Directory

- images: contains images
- lib: addittional libraries for JavaVis
- bin: binary (.class) files
- javavis: source
  - base: basic classes
  - desktop: desktop framework
  - jip2d: 2d framework
  - jip3d: 3d framework

JavaVis 2D



Segments: 0
Points: 0
Polygons: 0

Geometry: On

Mode

● Segments
○ Points
○ Polygons

Action

Select

Type: [No selection]
Start: (-1,-1)
End: (-1,-1)
Length: 0 Pixels

Frame Size:
Coords: (OUT)
Value: (OUT)

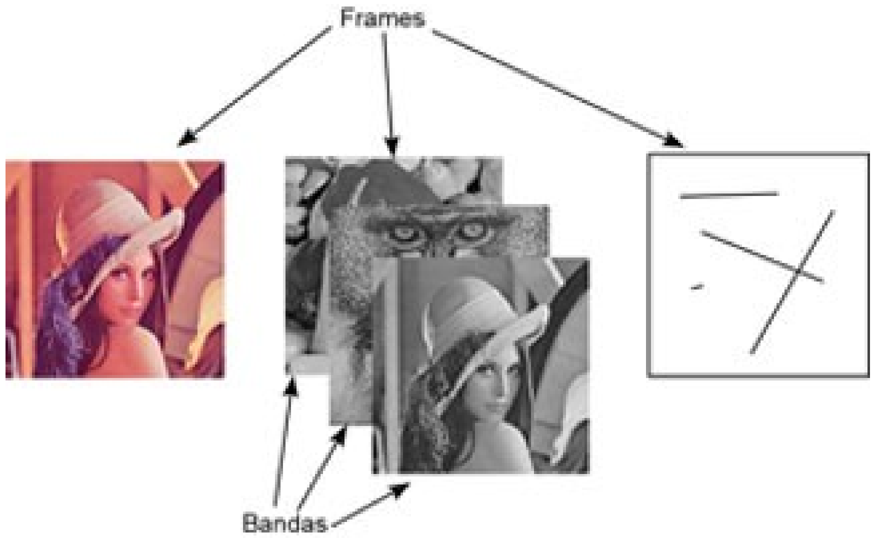
Frame: 0:COLOR:baboon
Band: COLOR-Bands RGB
Progress: 100%

Features

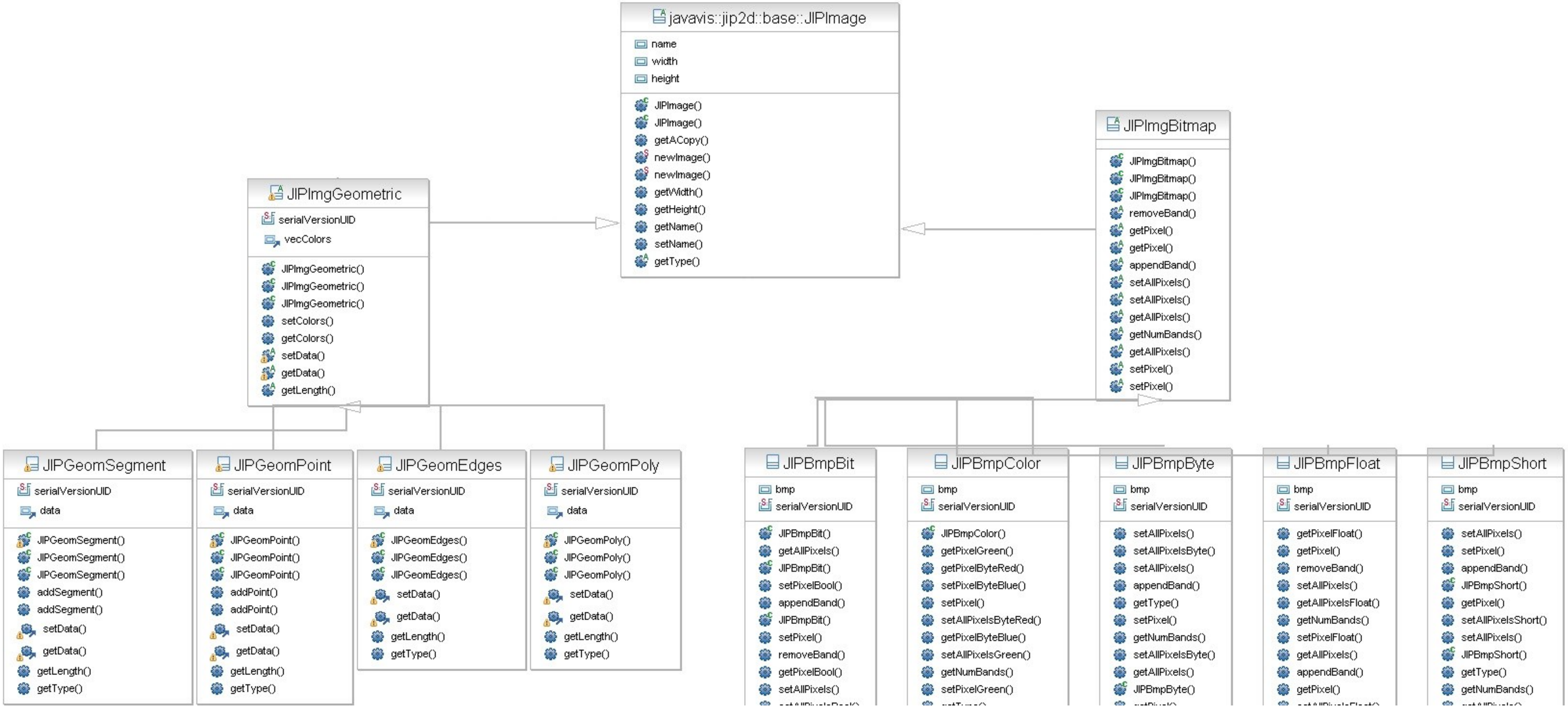
- *Traditional* image processing framework
- Goal: implement once, use elsewhere.
- An algorithm is implemented and the library is in charge of input and output parameter checking, showing images in the GUI, and so on
- Three ways to execute an algorithm:
  - From the GUI: in order to allow visual inspection of the results
  - From command line: fast processing of the images
  - From another algorithm: allows to reuse algorithms

Image format

- JavaVis manages image sequences
- An element in the sequence is an image (or frame)
- The image basic type is JIPimage



Class hierarchy



javavis::jip2d::base::JIPImage
- name
- width
- height
- JIPImage()
- JIPImage()
- getACopy()
- newImage()
- newImage()
- getWidth()
- getHeight()
- getName()
- setName()
- getType()

JIPImgGeometric
- serialVersionUID
- vecColors
- JIPImgGeometric()
- JIPImgGeometric()
- JIPImgGeometric()
- setColors()
- getColors()
- setData()
- getData()
- getLength()

JIPImgBitmap
- JIPImgBitmap()
- JIPImgBitmap()
- JIPImgBitmap()
- removeBand()
- getPixel()
- getPixel()
- appendBand()
- setAllPixels()
- setAllPixels()
- getAllPixels()
- getNumBands()
- getAllPixels()
- setPixel()
- setPixel()

JIPGeomSegment
- serialVersionUID
- data
- JIPGeomSegment()
- JIPGeomSegment()
- JIPGeomSegment()
- addSegment()
- addSegment()
- setData()
- getData()
- getLength()
- getType()

JIPGeomPoint
- serialVersionUID
- data
- JIPGeomPoint()
- JIPGeomPoint()
- JIPGeomPoint()
- addPoint()
- addPoint()
- setData()
- getData()
- getLength()
- getType()

JIPGeomEdges
- serialVersionUID
- data
- JIPGeomEdges()
- JIPGeomEdges()
- JIPGeomEdges()
- setData()
- getData()
- getLength()
- getType()

JIPGeomPoly
- serialVersionUID
- data
- JIPGeomPoly()
- JIPGeomPoly()
- JIPGeomPoly()
- setData()
- getData()
- getLength()
- getType()

JIPBmpBit
- bmp
- serialVersionUID
- JIPBmpBit()
- getAllPixels()
- JIPBmpBit()
- setPixelBool()
- appendBand()
- JIPBmpBit()
- setPixel()
- removeBand()
- getPixelBool()
- setAllPixels()

JIPBmpColor
- bmp
- serialVersionUID
- JIPBmpColor()
- getPixelGreen()
- getPixelByteRed()
- setPixelByteBlue()
- setPixel()
- setAllPixelsByteRed()
- getPixelByteBlue()
- setAllPixelsGreen()
- getNumBands()
- setPixelGreen()

JIPBmpByte
- bmp
- serialVersionUID
- setAllPixels()
- setAllPixelsByte()
- setAllPixels()
- appendBand()
- getType()
- setPixel()
- getNumBands()
- setAllPixelsByte()
- getAllPixels()
- JIPBmpByte()

JIPBmpFloat
- bmp
- serialVersionUID
- getPixelFloat()
- getPixel()
- removeBand()
- setAllPixels()
- getAllPixelsFloat()
- getNumBands()
- setPixelFloat()
- getAllPixels()
- appendBand()
- getPixel()

JIPBmpShort
- bmp
- serialVersionUID
- setAllPixels()
- setPixel()
- appendBand()
- JIPBmpShort()
- getPixel()
- setAllPixelsShort()
- setAllPixels()
- JIPBmpShort()
- getType()
- getNumBands()

Image format: bitmap

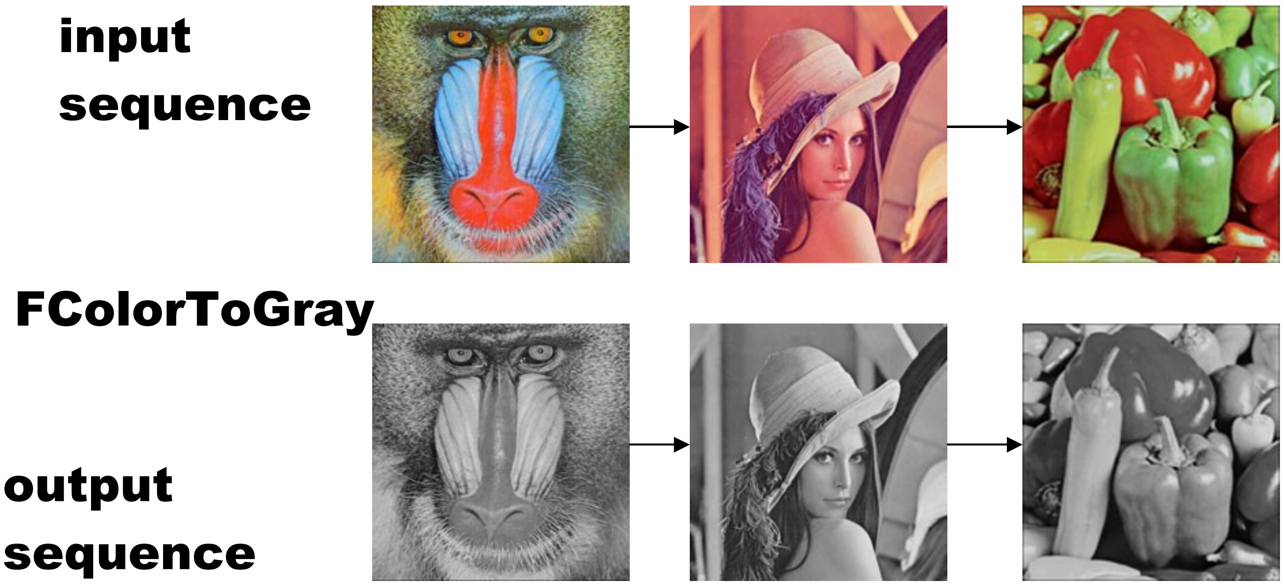
- A bitmap is a matrix
- It can be one of these types:
  - BIT (0,1), BYTE (0..255), SHORT (0..65525), FLOAT (0..1), COLOR (three bands (RGB) of BYTE type)
- Internally, each class stores an onedimensional array of data: in case of bit *boolean*[], case of byte *byte*[] and so on.

Image format: geometric

- A geometrical type contains geometrical elements
- This type only stores the coordinates of the points in the element
- Geometrical types: POINT, SEGMENT (two points), EDGES (a list of adjancent points and can be not closed), POLY (list of point not necessarily adjacents and always closed)
- A frame can contain more than one element (i.e. if it is POINT type, it can contain more than one point)

File format

- A file contains a sequence
- When a file is processed using an algorithm, a new file is obtained with the same number of frames than the original file, where each frame has been processed with the algorithm



input sequence

FColorToGray

output sequence

Algorithms

- To implement a new algorithm in JavaVis, we have to implement a function
- A function is a Java class which inherits from the abstract class JIPFunction
- JavaVis allows to implement once and use it in different ways
- When defining an algorithm, we just need to implement the algorithm and input and output parameters
- Parameter checking, in the GUI, input/output, etc. are done by JavaVis

Defining a new function

- Function must be included in the javavis.jip2d.functions package
- The class name must begin with F
- The new class must be placed in the javavis//jip2d//functions directory
- To show the function in the functions bar, a icon of 17x17 in JPEG or GIF format must be placed in the icons directory

Implementing a new algorithm

- **We just need to implement the constructor and the *processImg* method**
- **At the constructor, we define information and input/output parameters:**

```
public FBrightness() {
        super();
        name = "FBrightness";
        description = "Adjusts the brightness of the image.";
        groupFunc = FunctionGroup.Adjustment;
        JIPParamInt p1 = new JIPParamInt("perc", false, true);
        p1.setDefault(100);
        p1.setDescription("Percentage (when 100% the image is not modified)");
        addParam(p1);
}
```

Implementing a new algorithm

- **Our algorithm is placed in the only method we have to implement: *processImg***
- **This method always has a parameter JIPImage and returns a JIPImage object**

```
public JIPImage processImg(JIPImage img) throws
    JIPException
```

- **This is the code called by the GUI, Launch or another function**

Programming with JavaVis

- Sequence management (*JIPSequence* class):
  - getNumFrames(), getName()[setName(String)], getFrame(n)[setFrame(img,n)], insertFrame(img,n), addFrame(img), removeFrame(n), appendSequence(seq)
- JIPFunction class incorporates a method called processSeq(seq)
- The GUI (and Launch) calls this method for the complete sequence
- If our algorithm needs to process the complete sequence, we must redefine this method

Managing an image

- **Creating a new image: we have several ways to create an image:**
  - ***static JIPImage.newImage(b, w, h, ImageType)***
  - ***JIPImage img.clone();***
  - ***new JIPBmpByte(w, h);***
- **Methods from JIPImage**
  - **getWidth, getHeight, setName, getName, getType**
- **Methods from JIPImgBitmap**
  - **getPixel(x,y), getPixel(b,x,y), getAllPixels(), getAllPixels(b), (and its corresponding *set*) Every subclass of JIPImgBitmap implements these methods. They manage double values, i.e., the internal values are converted to double**
  - **getNumBands, removeBand, appendBand**

Managing an image

- **Each image type class (JIPBmpBit, JIPBmpByte, ...) has additional methods to access pixel values**
- **For example, JIPBmpBit, has a method called getPixelBool(x,y) which returns the boolean value at that pixel. So, we can access with boolean getPixelBool(x,y) or with double getPixel(x,y).**
- ***Caution*: internal representation (e.g. byte) can return a negative value. We recommend to use double methods, unless you have a clear idea of what you are doing!**

Geometrical images

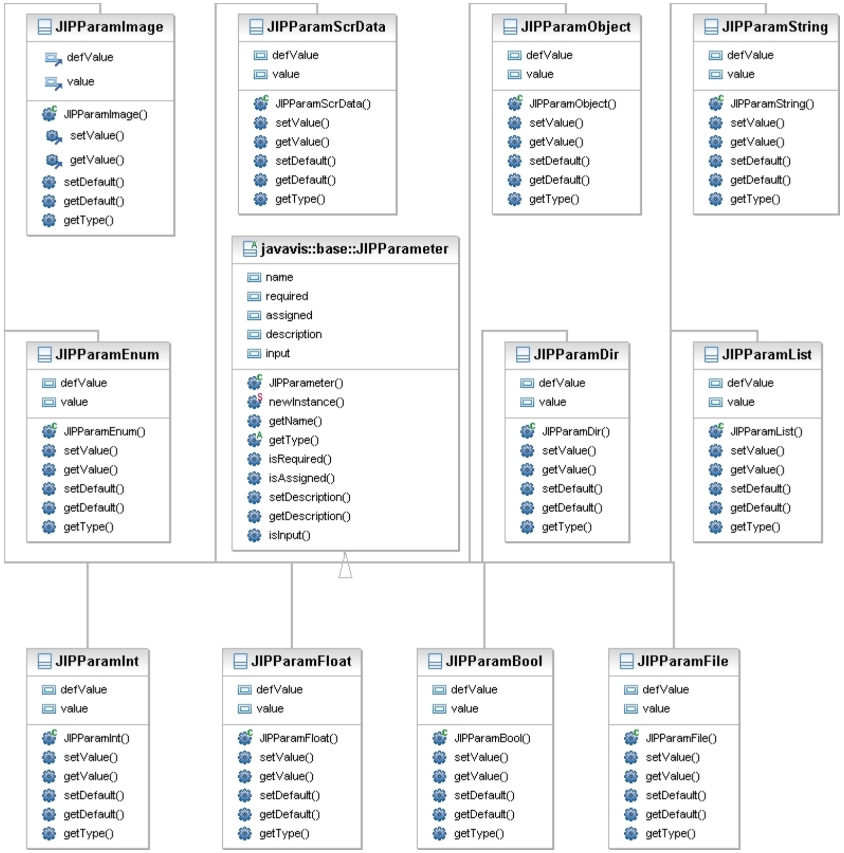
- As indicated, geometrical images contain a list of coordinates.
- getData, setData: use an ArrayList. Depending of the geometrical type it can be an ArrayList<Integer> (in case of POINT, SEGMENT) or ArrayList<ArrayList<Integer>> (EDGES, POLY)
- Geometrical data also have a color associated with each element, i.e., we can have 10 points each of them with a different color (beta)

Managing images inside a functions

- JIPImage processImg (JIPImage img)
- This method always receives a JIPImage object
- Imagine that our algorithm can be only applied to JIPBmpByte, we can do the following:
  - if (img.getType()!=ImageType.BIT) Exception //*Checking*
  - JIPBmpBit imgBmp = (JIPBmpBit)img; // *Cast*
- Imagine now that can be only applied to anyone of the the JIPImgBitmap subclasses
  - JIPImgBitmap imgBmp = (JIPImgBitmap)img; imgBmp.getPixel(x,y) // *This can be useful as any subclass of JIPImgBitmap must implement this method*

Parameters of a function

- We have an abstract class JIPParameter
- There are several subclasses of JIPParameter, indicating different kind of data



Parameters of a function

- **In order to define a parameter, we create, at the constructor, a JIPParameter object:**
  - ***JIPParamXXX p1=new JIPParamXXX(name, required, input);***

  where *name* is the name identifying the parameter; *required* indicates if the parameter is required or not, and *input* indicates if it is input or output parameter

  e.g. JIPParamFloat p1=new JIPParamFloat("sigma",true,false);
- **Default value of the parameter:**
  - p1.setDefault(1.0f);
- **Description of the parameter:**
  - p1.setDescripion("Level of Gaussian smoothed");
- **The parameter is added to the list of parameters (*params* is already defined in JIPFunction)*:***
  - addParameter (p1);

Parameter checking

- **The GUI and the *Launch* class do the parameter checking**
- **We can assume that when the *processImg* method is executed, parameters have its value assigned**
- **To get the parameter value:** getParamValueFloat("sigma");
- **If our function has output parameters, these must be defined and stored in *results***
- **To give value to an output parameter:** setParamValue("nombre",valor);
- **If we execute a function with output parameters, we get them (once *processImg* is executed) with** funcion.getParamValue("nombre");

Managing an error

- The main way to managing an error is using the JIPException exception: throw a new JIPException when something wrong happens.
- The library catches those exceptions and manages it
- In your code, you can catch these exceptions and process it

debugging

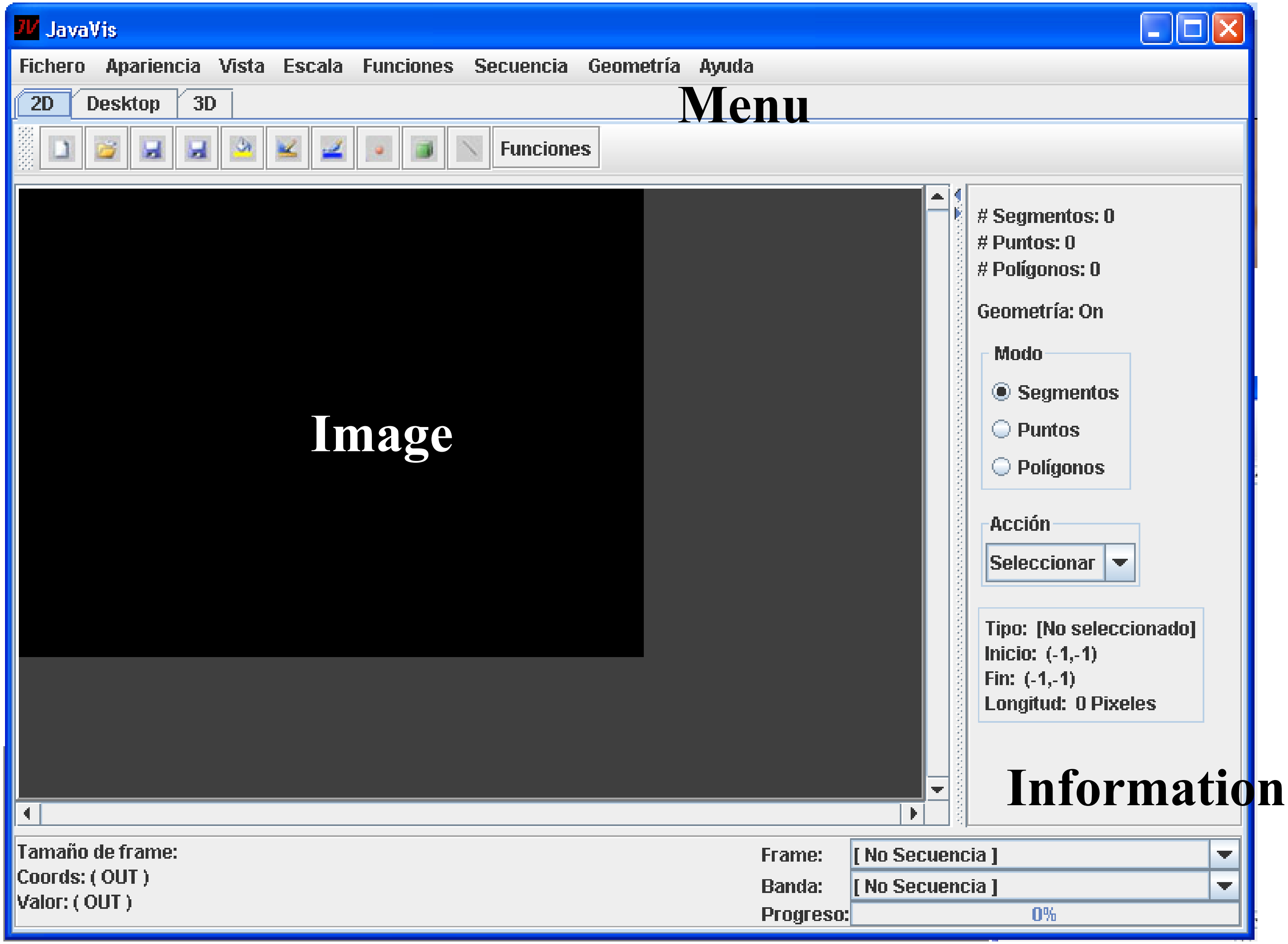
- To debug an algorithm, there is an easy way to create a log. It uses the log4j API and sends all the errors to a file (log/javavis.log)
- There is a file (resources/log4j.properties) in which we can adjust the level of log to write.
- Log4j creates a hierarchy (info(less level), debug, warning, error(highest level)). The default level is debug, but change it in the previous file, eliminates all the messages in lower levels.
-

Complete code: FBinarize

```
package javavis.jip2d.functions;
//Imports …..
//A function must inherits from JIPFunction
public class FBinarize extends JIPFunction {
    private static final long serialVersionUID = -7262973524107183332L;
    public FBinarize() {// Constructor
        super();
        name = "FBinarize"; // Name of the function
        //Description
       description = "Transforms a BYTE image to binary";
       //GUI group
       groupFunc = FunctionGroup.Transform;
        // First parameter
        JIPParamInt p1 = new JIPParamInt("u1", false, true);
         p1.setDefault(128);
         p1.setDescription("Lower bound of the range to consider as 1");
         //Second parameter
         JIPParamInt p2 = new JIPParamInt("u2", false, true);
         p2.setDefault(255);
         p2.setDescription("Upper bound of the range to consider as 1");

         addParam(p1);
         addParam(p2);     }
     // Here we can define our algorithm
     public JIPImage processImg(JIPImage img) throws JIPException {
           JIPBmpBit res = null;
           //Get the parameter values
           int p1 = getParamValueInt("u1");
           int p2 = getParamValueInt("u2");
```

```
// This function is only defined for BYTE images
     if (img.getType() == ImageType.BYTE) {
         int w = img.getWidth();
         int h = img.getHeight();
         int b = ((JIPBmpByte)img).getNumBands();
         // Output image
         res = new JIPBmpBit(b, w, h);
           long percTotal = totalPix * b;
         // For each band
         for (int nb = 0; nb < b; nb++) {
              // Get all the pixel at once, because we do not need
              // neighbor relations
              double[] bmp = ((JIPBmpByte)img).getAllPixels(nb);

              boolean[] bin = new boolean[w * h];
              for (int i = 0; i < w * h; i++) {
                    bin[i] = (bmp[i] >= p1 && bmp[i] <= p2);
                    percProgress = (int)((100*((nb+1)*totalPix + i))/percTotal);
               }
               // Once the band is processed, it is assigned to the
               // output image
              res.setAllPixelsBool(nb, bin);
          }
     }
     else
              throw new JIPException("Binarize only defined for BYTE
         images");
     return res;
   }
}
```

2D GUI



JavaVis
Fichero Apariencia Vista Escala Funciones Secuencia Geometría Ayuda
2D Desktop 3D
Menu
Funciones
Image
Information

Segmentos: 0
Puntos: 0
Polígonos: 0

Geometría: On

Modo
● Segmentos
○ Puntos
○ Polígonos

Acción
Seleccionar

Tipo: [No seleccionado]
Inicio: (-1,-1)
Fin: (-1,-1)
Longitud: 0 Pixeles

Tamaño de frame:
Coords: (OUT)
Valor: (OUT)

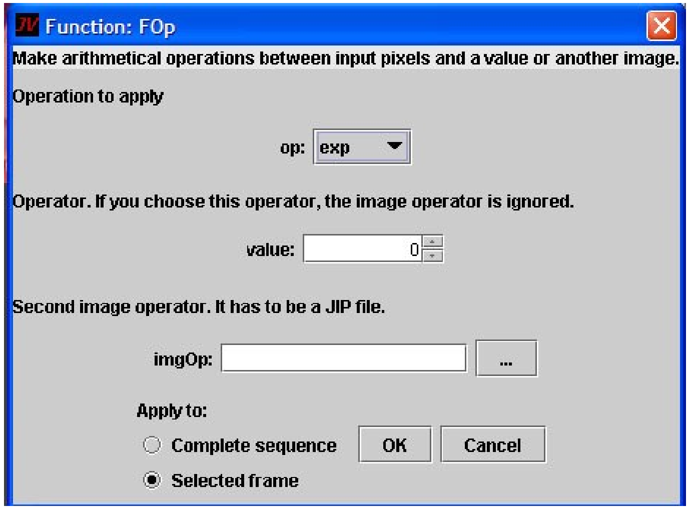
Frame: [No Secuencia]
Banda: [No Secuencia]
Progreso: 0%

GUI

- The information area shows the pixel value where the pointer is placed
- The pointer can be moved by the *q, a, s, z* keys, up, left, right and down, respectively
- When we change from a bitmap frame to another bitmap, the image shown is changed. From a bitmap to geometric or geometric to geometric, the geometrical data is superposed
- *Select* mode allows to select geometrical elements and *Add* adds new elements
- Two bars: functions (a new icon must be included for each new function) and tools
- Several menus

Function execution

- There is a function menu
- It is necessary to have an image in the environment to execute a functions
- When a function is executed, a parameter input window is shown, where we can enter the parameter values


Function: FOp

Make arithmetical operations between input pixels and a value or another image.

Operation to apply

op: exp

Operator. If you choose this operator, the image operator is ignored.

value: 0

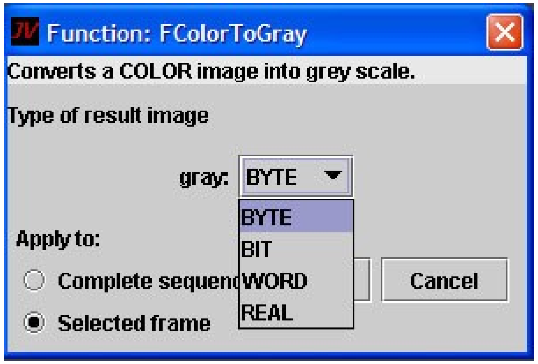
Second image operator. It has to be a JIP file.

imgOp:

Apply to:

○ Complete sequence

● Selected frame


Function: FColorToGray

Converts a COLOR image into grey scale.

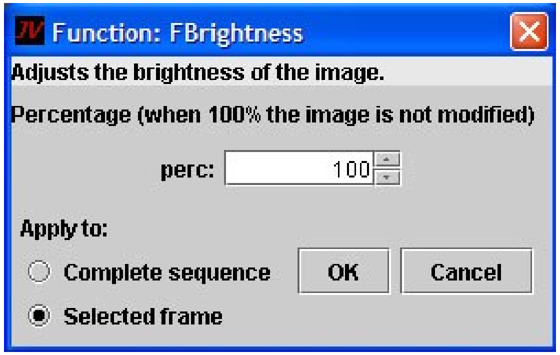
Type of result image

gray: BYTE

Apply to:

○ Complete sequen

● Selected frame


Function: FBrightness

Adjusts the brightness of the image.

Percentage (when 100% the image is not modified)

perc: 100

Apply to:

○ Complete sequence

● Selected frame

Function execution from command line

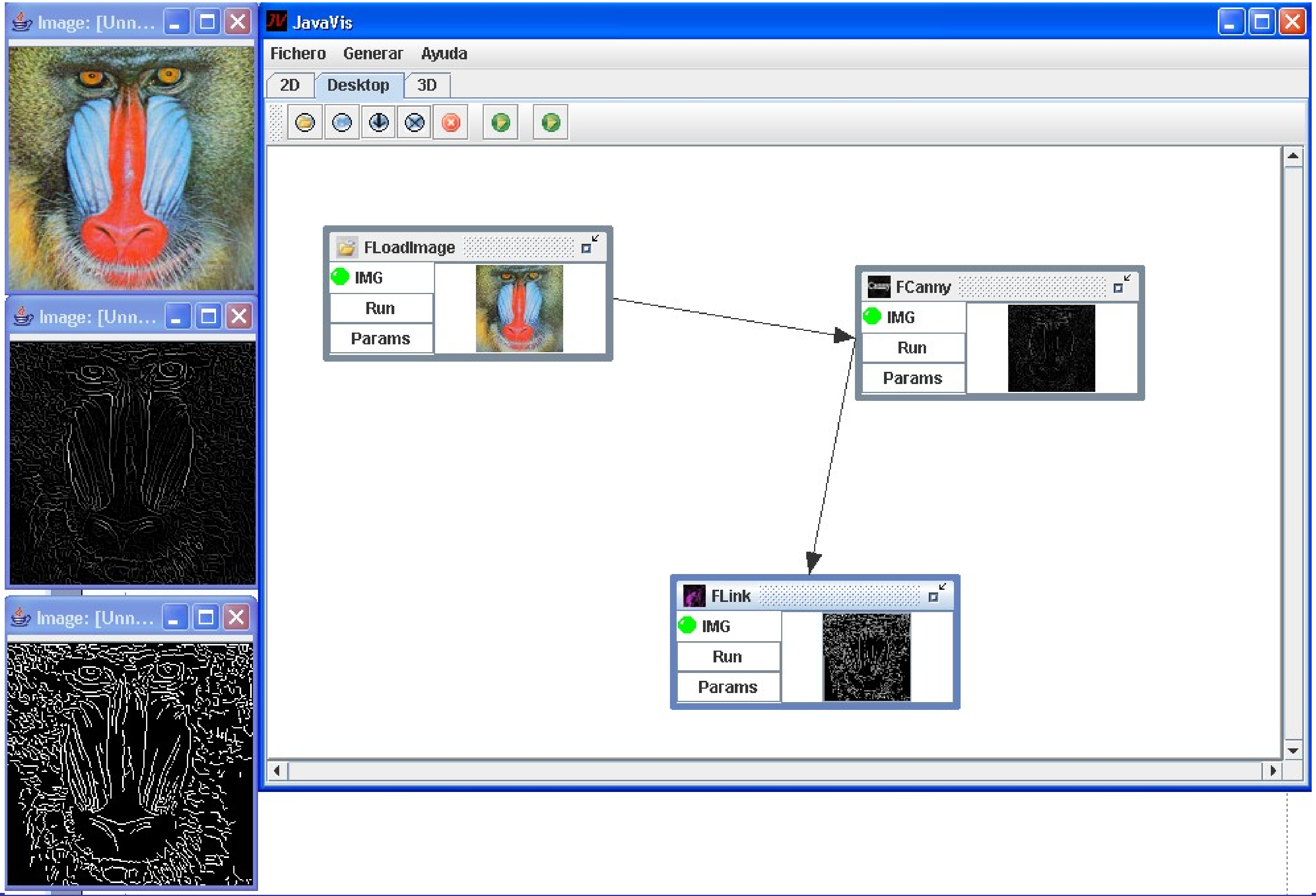
- **Use the *Launch* class (parameter order is not relevant)**
  - java jip.Launch FCanny -sigma 1.5 fich.jip salida.jip
- **Information about the class**

```
C:\eclipse\workspace\JavaVis>java jip.Launch FCanny -help
FUNCTION:
Detects edge using the Canny’s method

Instructions for use: java Canny <parameters> <infile> [<outfile>]
   <infile>: Source file to process [REQUIRED]
   <outfile>: Destination file [Default: out.jip]
Parameters:
   -help
      Shows method of use
   -sigma <real> [Default: 1.0]
      Level of gaussian smoothed
   -brightness <integer> [Default: 100]
      Brightness adjustment
```

Use from another function

- **We can call other function**
- **Object creation:**
  - FCanny fc = new FCanny();
- **Parameter assignment (if neccessary):**
  - fc.setParamValue("sigma",1.5f);
- **Function execution:**
  - JIPImage salida = fc.processImg(img);
- **Error checking, catching the exception (*JIPException*)**

JavaVis Desktop

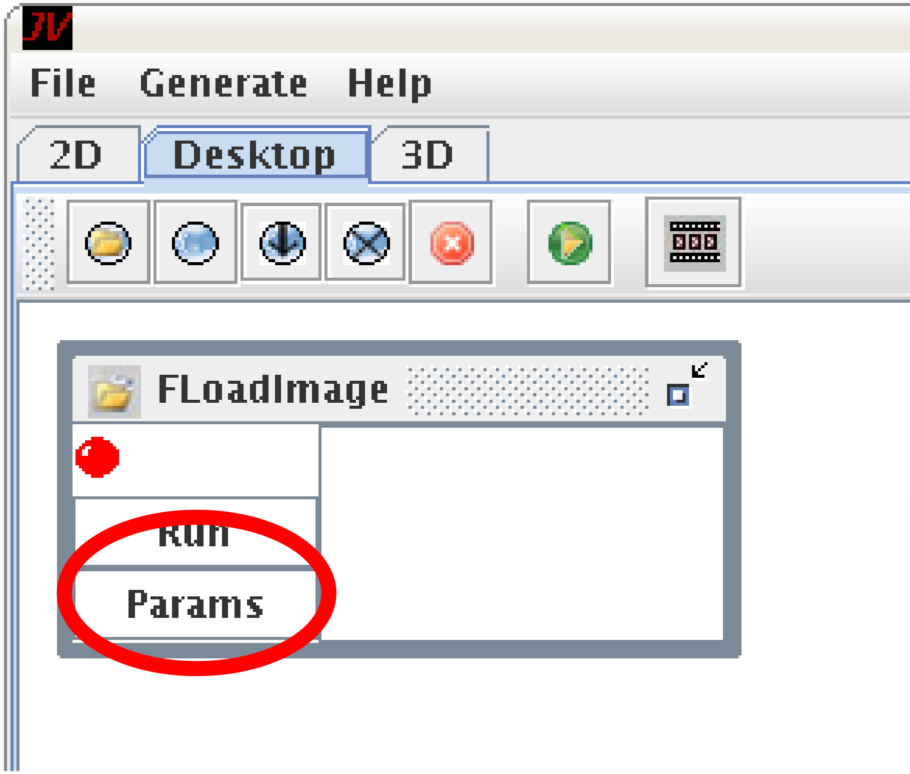


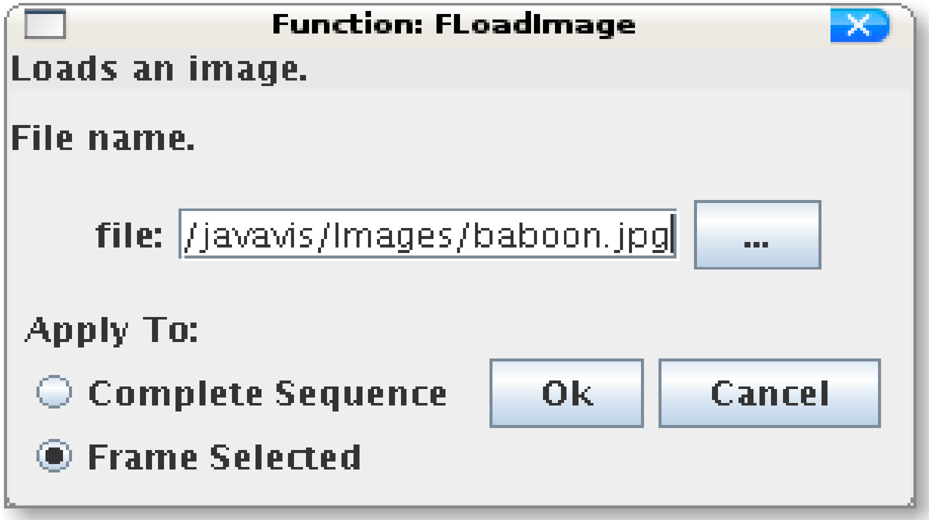
JavaVis Desktop

- Visual tool for batch processing
- Intermediate results preview
- JIPFunction code generation

JavaVis Desktop

- First load an image




Function: FLoadImage

Loads an image.

File name.

file: /javavis/Images/baboon.jpg

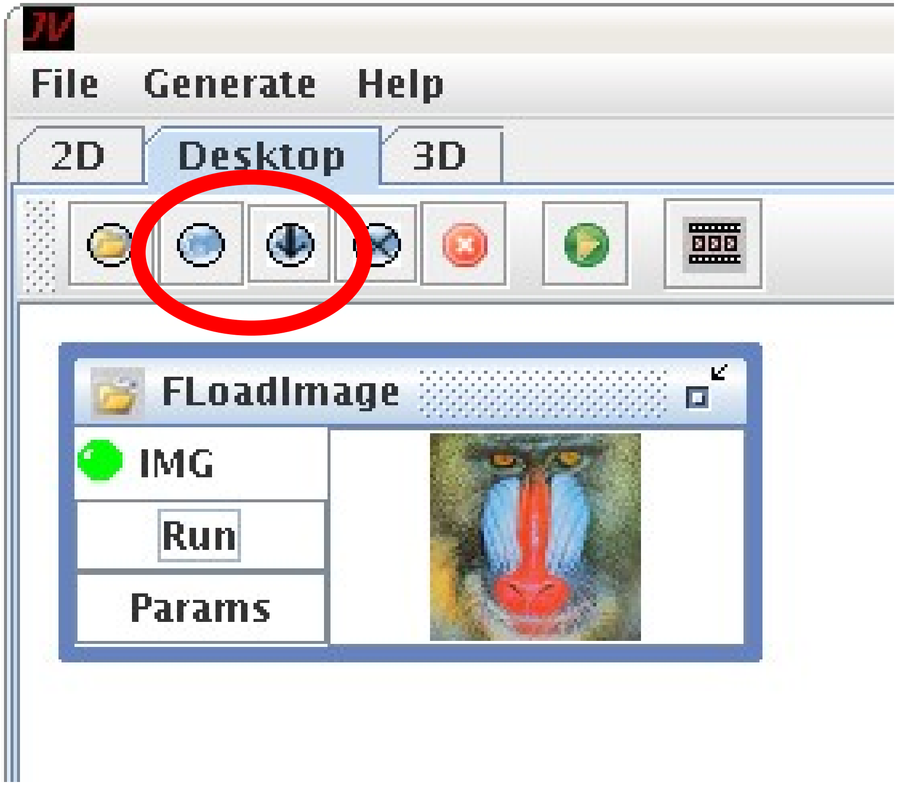
Apply To:

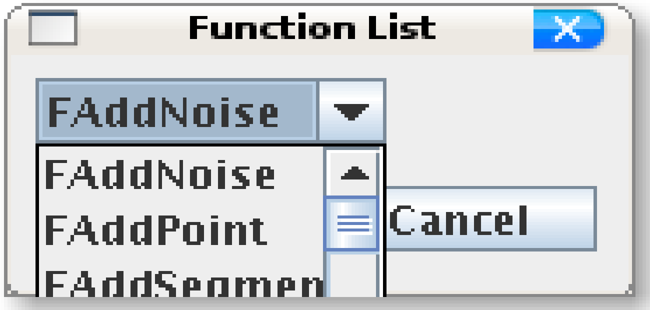
○ Complete Sequence

● Frame Selected

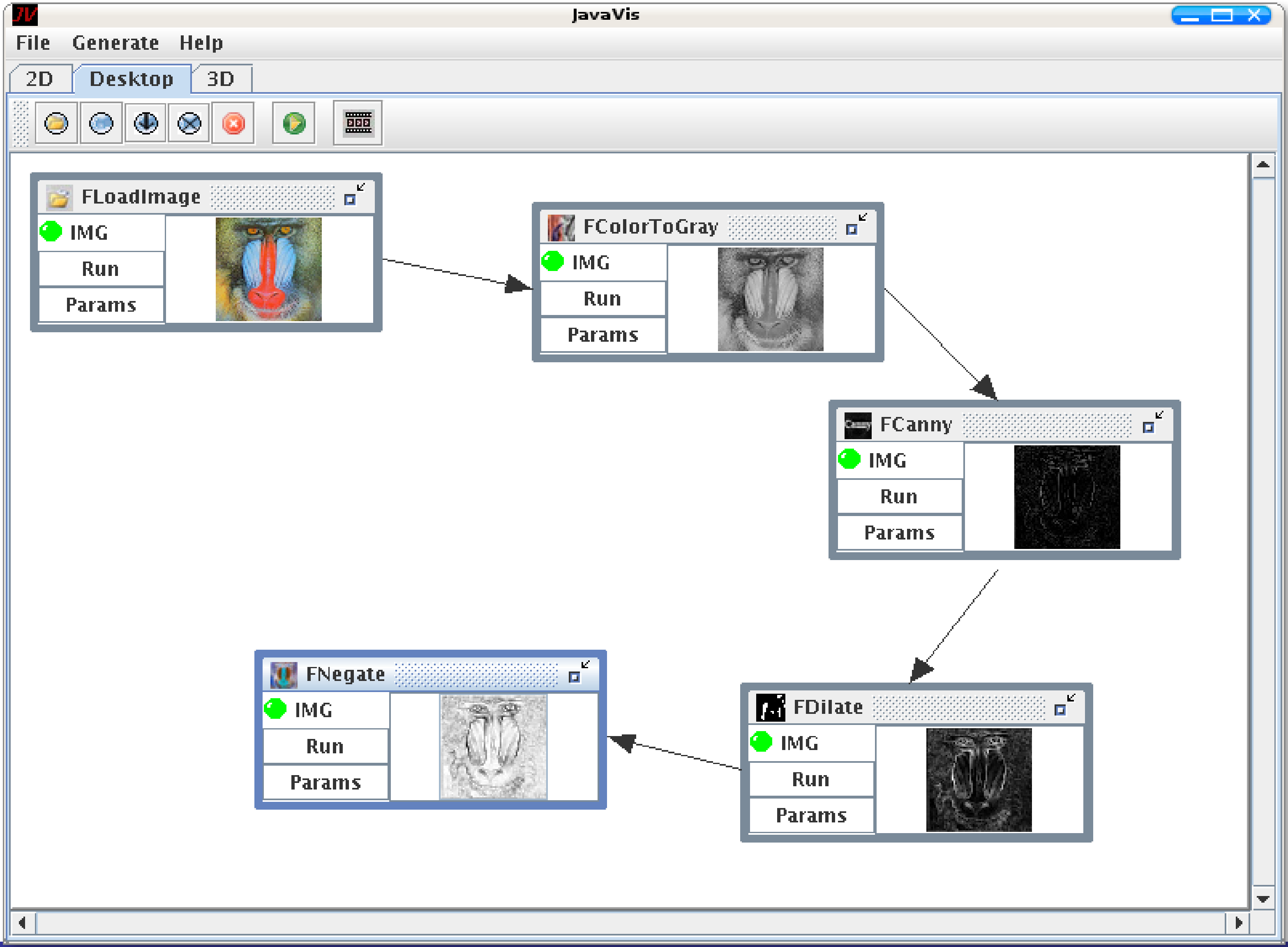
JavaVis Desktop

- Add the next function



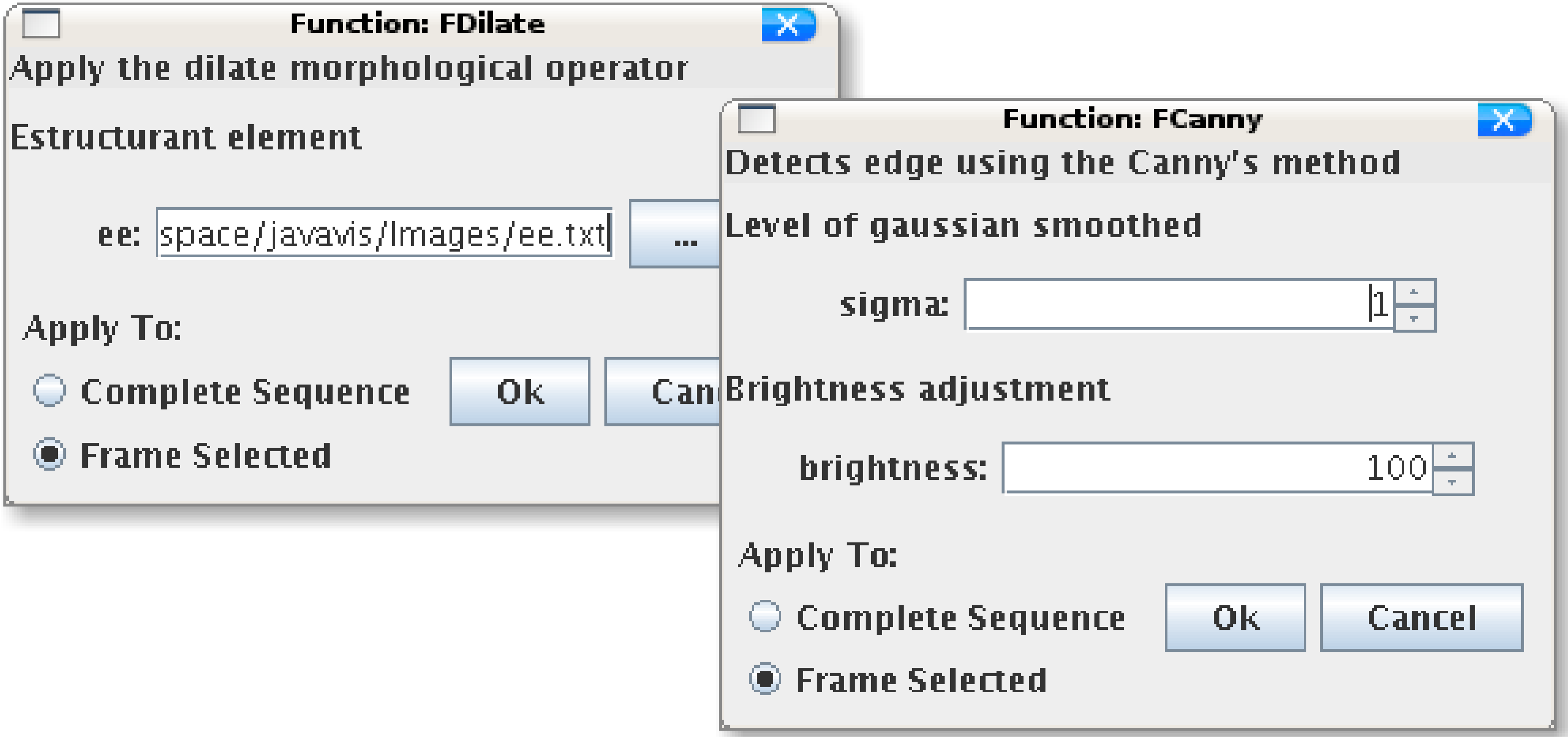


JavaVis Desktop



JavaVis Desktop

- Set parameters before pressing “Run”
- Parameters depend on functions



Function: FDilate

Apply the dilate morphological operator

Estructurant element

ee: space/javavis/Images/ee.txt

Apply To:

○ Complete Sequence

● Frame Selected

Function: FCanny

Detects edge using the Canny's method

Level of gaussian smoothed

sigma: 1

Brightness adjustment

brightness: 100

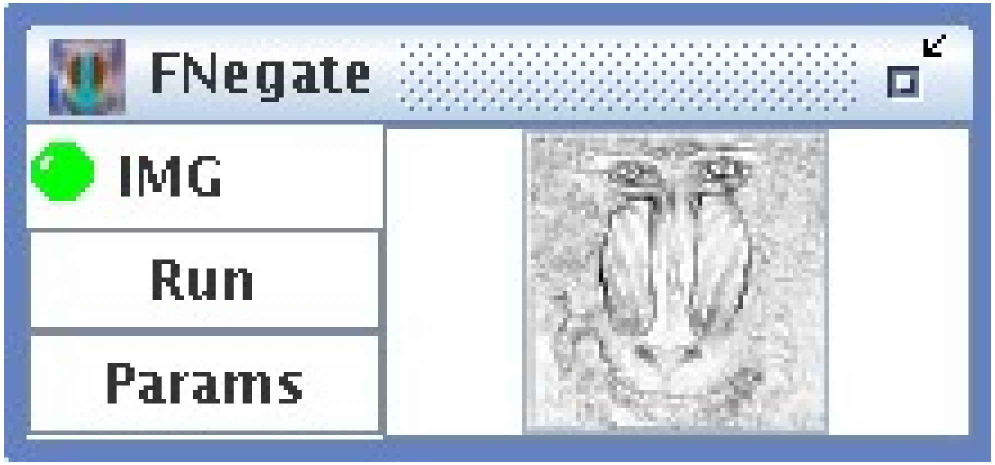
Apply To:

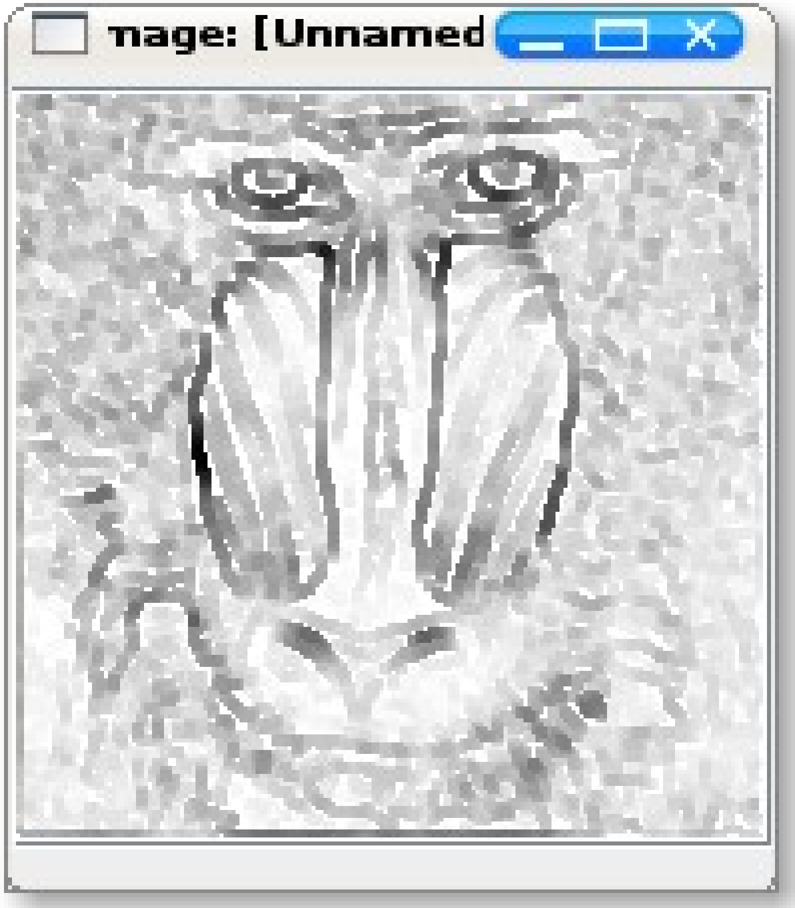
○ Complete Sequence

● Frame Selected

JavaVis Desktop

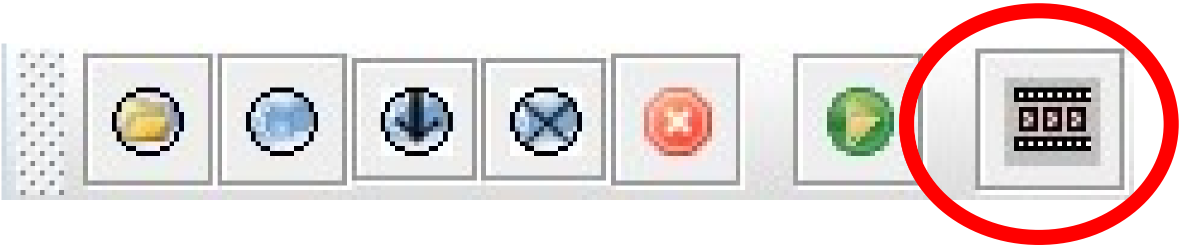
- Previews and results

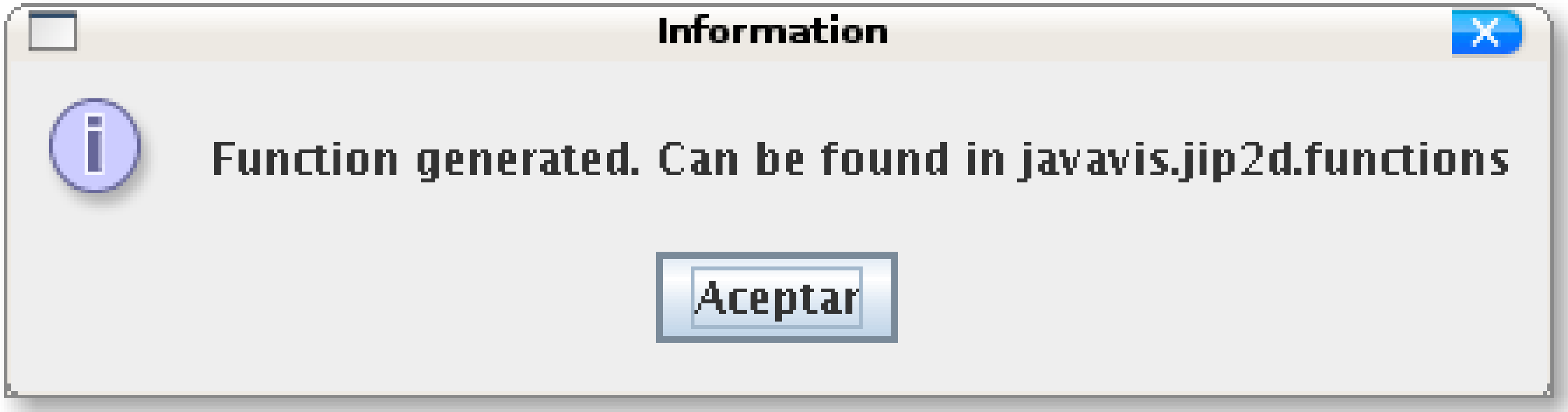




JavaVis Desktop

- **JIPFunction code generation**

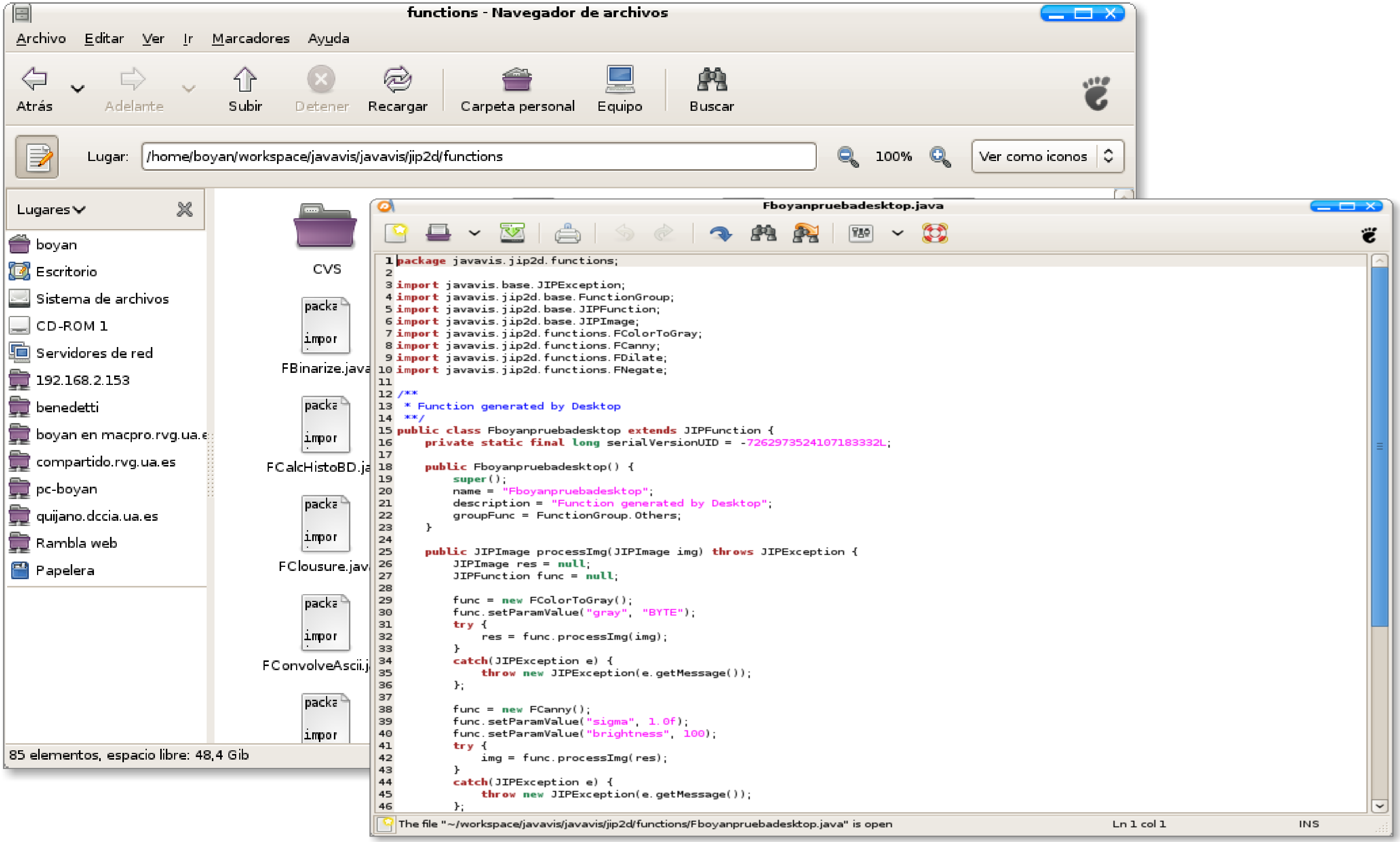




Information

Function generated. Can be found in javavis.jip2d.functions

JavaVis Desktop



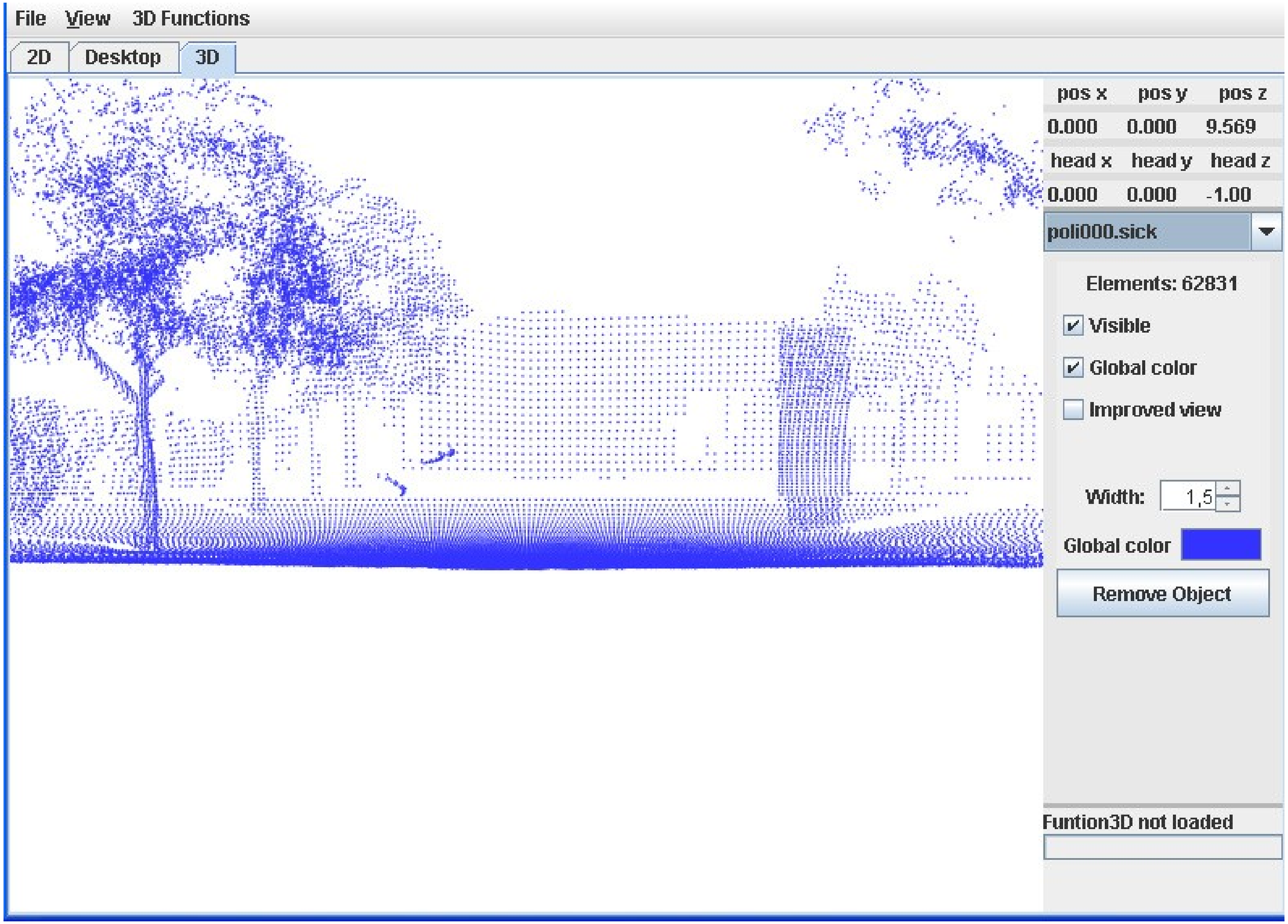
Lugar: /home/boyan/workspace/javavis/javavis/jip2d/functions

85 elementos, espacio libre: 48,4 Gib

```java
package javavis.jip2d.functions;

import javavis.base.JIPException;
import javavis.jip2d.base.FunctionGroup;
import javavis.jip2d.base.JIPFunction;
import javavis.jip2d.base.JIPImage;
import javavis.jip2d.functions.FColorToGray;
import javavis.jip2d.functions.FCanny;
import javavis.jip2d.functions.FDilate;
import javavis.jip2d.functions.FNegate;

/**
 * Function generated by Desktop
 **/
public class Fboyanpruebadesktop extends JIPFunction {
    private static final long serialVersionUID = -7262973524107183332L;

    public Fboyanpruebadesktop() {
        super();
        name = "Fboyanpruebadesktop";
        description = "Function generated by Desktop";
        groupFunc = FunctionGroup.Others;
    }

    public JIPImage processImg(JIPImage img) throws JIPException {
        JIPImage res = null;
        JIPFunction func = null;

        func = new FColorToGray();
        func.setParamValue("gray", "BYTE");
        try {
            res = func.processImg(img);
        }
        catch(JIPException e) {
            throw new JIPException(e.getMessage());
        };

        func = new FCanny();
        func.setParamValue("sigma", 1.0f);
        func.setParamValue("brightness", 100);
        try {
            img = func.processImg(res);
        }
        catch(JIPException e) {
            throw new JIPException(e.getMessage());
        };
```

The file "~/workspace/javavis/javavis/jip2d/functions/Fboyanpruebadesktop.java" is open

JavaVis3D



pos x	pos y	pos z
0.000	0.000	9.569

head x	head y	head z
0.000	0.000	-1.00

poli000.sick

Elements: 62831

☑ Visible

☑ Global color

□ Improved view

Width: 1,5

Global color 🟦

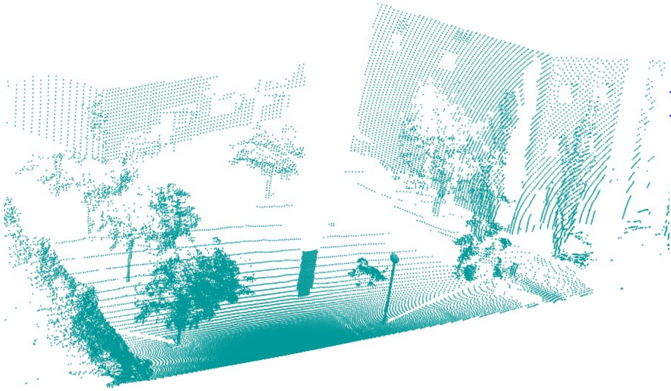
Funtion3D not loaded

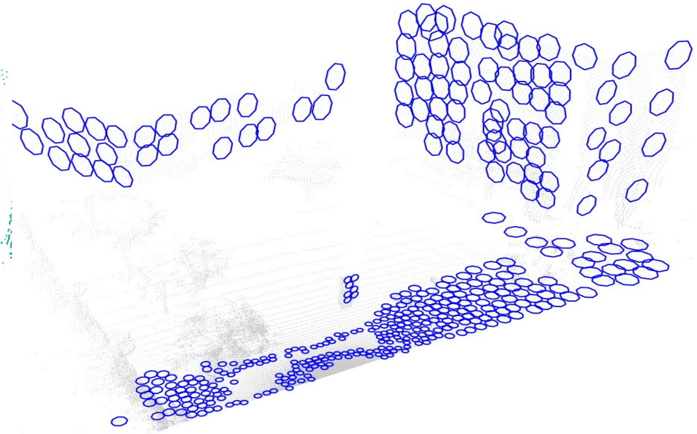
JavaVis3D: Features

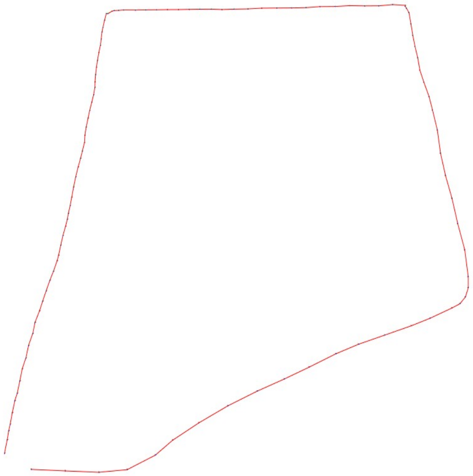
- *3D* image processing framework
- It follows the same philosophy than JavaVis 2D
- Java3D for displaying 3D graphics
- 3D geometric library included
- Goal: implement once, use elsewhere.
- An algorithm is implemented and the library is in charge of input and output parameter checking, showing 3D images in the GUI3D, and so on

Image 3D Format

- Types: Point3D, Plane3D, Normal3D, Segment3D...
- Special type: Trajectory3D. The same idea that sequences for JavaVis
- ScreenData: Collection of geometric entities that can be draw on the screen







Algorithms 3D

- To implement a new algorithm 3D, we have to implement a function 3D
- A function 3D is a Java class which inherits from the abstract class JIPFunction3D
- JavaVis allows to implement once and use it in different ways
- When defining an algorithm 3D, we just need to implement the algorithm and input/output parameters
- Parameter checking, drawing in the GUI3D, input/output, etc. are done by JavaVis

Defining a new function 3D

- New 3D functions are defined like original 2D JavaVis functions
- Function 3D must be included in the javavis.jip3d.functions package
- The class name must begin with F
- The new class must be placed in the javavis//jip3d//functions directory
- We just need to implement the constructor and the *processData* method

Implementing a new algorithm 3D

- **At the constructor, we define information and input/output parameters:**

```
public FPointFilter()
{
        super();
        this.allowed_input = ScreenOptions.tPOINTSET3D;
        this.group = Function3DGroup.Mapping;

        // resolution param. Cube side length for grouping points
        FunctionParam p1 = new FunctionParam("Resolution",
            FunctionParamType.tREAL);
        p1.setValue(0.10);

        this.addParam(p1);
  }
```

Implementing a new algorithm 3D

- Our algorithm is placed in the only method we have to implement: *processData*
- This method always has a parameter ScreenData

```
public void proccessData(ScreenData scr_data) throws
    JIPException;
```

- This is the code called by the GUI, Launch or another function 3D

Complete 3D Function code example:

```
package functions3D;
//imports
import geom3D.Octree;...
//Class FPointFilter. This class is used to reduce the number ...

public class FPointFilter extends Function3D {
  public FPointFilter() {
    super();
    this.allowed_input = ScreenOptions.tPOINTSET3D;
    this.group = Function3DGroup.Mapping;
    // resolution param. Cube side length for grouping points
    FunctionParam p1 = new FunctionParam("Resolution",
         FunctionParamType.tREAL);
    p1.setValue(0.10);
    this.addParam(p1);
  }

  public void proccessData(ScreenData scr_data) throws
         JIPException {
    result_list = new ArrayList<ScreenData>();
    Point3D bound_sup;
    Point3D bound_inf;
    float resolution = (float)this.paramValueReal("Resolution");
    Object []elements;
    Point3D element;
    int cont;
    ArrayList<Point3D> complete_list;
    PointSet3D ret;
    double prog_inc;

    bound_sup = new Point3D(200, 200, 200);
    bound_inf = new Point3D(-200, -200, -200);
    total_data = new Octree(bound_inf, bound_sup, resolution);

    elements = scr_data.elements();
    prog_inc = 50.0/elements.length;

    for(cont=0;cont<elements.length;cont++){
      element = (Point3D) elements[cont];
      total_data.insert(element);
      progress += prog_inc;
    }
    complete_list = total_data.getAll();

    ret = new PointSet3D(new ScreenOptions());
    ret.name = "ReducedPointSet";
    prog_inc = 50.0 / complete_list.size();
    for(cont=0;cont<complete_list.size();cont++){
      element = complete_list.get(cont);
      ret.insert(element);
      progress += prog_inc;
    }

    result_list.add(ret);
  }
}
```