ImagePy 开源图像处理可视化框架

飞鸟，流沙

2017 年 4 月 29 日

目录

1 介绍 Introduction **3**
1.1 参考 . . . 3
1.2 安装 . . . 3
1.3 About . . . 3
1.4 ImagePy 交流 QQ 群 . . . 3

2 开发文档 **3**
2.1 系统架构 . . . 3
2.2 插件加载 PluginLoader . . . 5
2.3 管理器 Manager . . . 6
2.4 常用功能汇总 IPy . . . 8
2.5 图像类 ImagePlus . . . 8
2.6 参数交互 ParaDialog . . . 10
2.7 滤波器 Filter . . . 12
2.8 整体操作 Simple . . . 15
2.9 自由插件 Free . . . 16
2.10 宏引擎 Macros . . . 18
2.11 交互工具 Tool . . . 18

3 操作手册 **20**
3.1 界面构成 . . . 20
3.2 相关概念 . . . 21
3.3 打开图像 . . . 23
3.4 常用工具 . . . 23
3.5 直方图调整 . . . 27
3.6 数学运算 . . . 27
3.7 滤波器 . . . 28
3.8 形态学操作 . . . 30
3.9 几何变换 . . . 30
3.10 选区运算 . . . 31
3.11 标记和像素统计 . . . 32
3.12 录制宏 . . . 34
3.13 开发 ImagePy 功能拓展 . . . 36

1 介绍 Introduction

1.1 参考

结构化你的工程——Python 最佳实践指南
ImagePy 官网主页| Github 下载| 知乎专栏

1.2 安装

ImagePy 初始版本以 Python2 为基础，后续拓展到 Python3。可以提供全平台支持。其中依赖 wxPython、libwebkitgtk-dev、six 等。

1. 安装 Python 运行环境。
2. 安装 ImagePy 依赖库：
   (a) Windows 安装 pip(3) install six wx numpy scipy
   (b) Mac 安装 sudo pip(3) install six wx numpy scipy
   (c) Linux 安装 sudo apt install libwebkitgtk-dev sudo pip(3) install six wx numpy scipy
3. 下载 ImagePy 并使用：在 Github 上下载 ImagePy 包，解压后找到 main.py 文件，双击运行即可。

1.3 About

作为 ImageJ 的开发成员，希望采纳优秀的面向插件的设计思想，同时充分发挥 Python 强大而丰富的第三方库。以此为初衷，开发了 ImagePy 开源图像图像处理框架。该框架在保留了 ImageJ 的插件设计和友好交互的同时，借鉴 OpenCV 的 Python 版，采用了 Numpy 作为基础图像数据结构，极大提高开发效率，降低开发门槛。

1.4 ImagePy 交流 QQ 群

ImagePy 的 QQ 交流群 (596310256)，此群是 ImageP 软件开发者和使用者交流群。如果遇到安装问题、Bug 报告、特殊需求定制，或者希望参与到项目开发，都可以加入。本群同时也可以用来交流软件使用技巧、图像处理基础支持、Python 图像编程、科学计算、Numpy、SciPy、Scikit-Learn、OpenCV4Python、ITK、WXPython、ImageJ 等的使用经验和技巧。

2 开发文档

2.1 系统架构

概述：ImagePy 是一款基于 Python 的可拓展的图像处理框架，可谓是 Python 版的 ImageJ。依托于 Python 简洁的语法和丰富的三方库，Image 设计的精简而可拓展。可以轻松介入任何基于 Numpy 的图像处理库，如 OpenCV、SciPy、Scikit-Learn 等。

设计思想：ImagePy 采用了插件式设计思想，根据功能的形式，制定了五种引擎模板。具体功能都是他们的子类，按照一定的方式组织起来，然后由相应的加载器和管理器维护。某种意义上，菜单即目录，功能即文件。

软件环境：Python2/3: 基础开发语言，可以使用 C/C++ 等语言进行拓展。Numpy: 科学计算库，底层主要由 C/C++ 编写，并依赖 MKL 等数学运算库来编译，为 Python 提供了对科学

计算的支持，提供了基础的数组存储和操作，是 ImagePy 的图像数据基础。SciPy: 基于 NumPy 数组，提供高级的数学函数。WxPython：ImagePy 的界面框架，核心由 C++ 编写而成，提供了 Python 版本的接口。

设计架构：

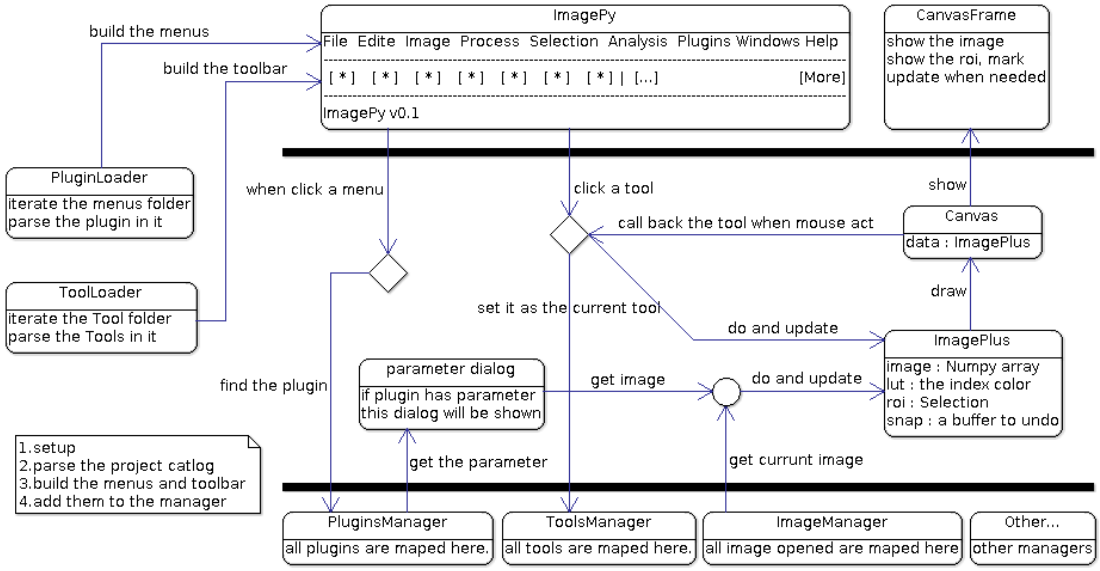

图 1: ImagePy Architecture

采用了 MVC 的设计思想，由三大模块组成：

1. View 展示层：ImagePy 展示层由主界面、菜单栏、工具条和图像窗口等视图构成。
2. Model 逻辑层：当用户执行操作命令时，软件会对当前图像做相应功能的处理。
3. Control 控制层：ImagePy 框架底层由若干个管理器，组织所有的插件和工具以及打开的图像。项目启动时，加载器会扫描项目文件夹，并按照其组织结构，映射成对应的菜单栏和工具栏。

ImagePy 项目结构：

1. ImagePlus: 图像封装类，包含图像数据 Image、图像的 Snap 镜像、ROI、色彩索引表、Mark 表层标记等。
2. IPy：对一些和常用功能的引用类，可以通过 IPy 访问 ImagePy 的许多常用功能，如打开图像、输出日志等。
3. ui
   - ui.Canvas: 负责绘制 ImagePlus，包括绘制图像数据、绘制选取、标记 Mark 等。
   - ui.CanvasFrame: Canvas 的封装类，使之能以一个窗口的形式展示。
   - ui.ParaDialog: 参数交互类。图像处理的诸多函数，有各种参数。采用数据驱动视图的方式，可以自动生成各类交互对话框。
4. core

- loader PluginLoader：插件加载类，负责解析 Menus 目录，并加载其中的插件。ToolsLoader: 工具加载类，负责解析 Tools 目录，并加载其中的工具。
- engines Filter：滤波器基类，用于图像处理类的插件，帮助用户进行一系列工作，比如处理多通道和图像栈，支持选区 ROI 等操作。Simple：简单插件基类，与 Filter 不同的是，它处理的对象不是图像，而是 ImagePlus 整体（比如处理图象栈、ROI 等）。Free：自由插件基类。该类运行时不依赖图像，可以在任何情况下执行（如打开图像功能）。Macros：宏插件。它可以由一组命令字符串组成，并依次执行。Tool：工具基类。定义了一组鼠标事件的处理函数，可以在 Canvas 的鼠标事件中被回调。
- roi Point：点选区 Line：线选区 Polygon：多边形选区...
- managers PluginsManager：用于管理所有的插件 ToolsManager：管理所用的工具 WindowsManager: 管理全部打开的图像窗口 ClipboardManager：管理剪贴板 ColorManager：颜色管理器
- menus 这里存放全部的插件，可以按照任意目录进行解析，常见的有：File Edit ...
- tools 这里用于存放所有的工具，注意工具只能够存放在 tools 的以及子目录下。Standud Transform ...

2.2 插件加载 PluginLoader

ImagePy 运行时，插件由 PluginsLoader 负责加载，由 PluginManager 负责管理，并被映射成菜单，在用户点击的时候执行相应的功能。

插件目录：rootdir/imagepy/menus

加载规则：

1. 位于 menus 目录及其子目录下
2. 如果文件名以 _plg.py 结尾，则文件内的 Plugin 类会被加载。
3. 如果文件名以 _plgs.py 结尾，则文件内的 Plugin 列表 plgs 将被依此加载。
4. 如果文件名以.mc 结尾，则会被解析成宏命令进行加载。

插件和菜单：ImagePy 的菜单全部是插件，插件在程序启动的时候被加载和解析为主程序的菜单。映射方式是：

- 目录下的文件夹结构被映射为主程序的相应菜单结构。
- 菜单的标题有插件类的 Title 属性决定。
- 对于宏映射成的菜单，标题就是文件名（不包括拓展名）。

调用顺序：由于菜单是由文件目录映射而成的，因此默认是按照字母顺序排序而成的。我们可以人工指定其顺序。

1. plgs 的内部顺序，有序列决定，可以在其中加入'-'，这样会被映射为菜单分割线。

   例如 menus>Process>Filter>binary_plgs.py:

   plgs = [Dilation, Erosion, '-', Closing, Opening, '-', Outline, FillHoles, EDT]

2. 目录中的插件可以在包的 ___init___.py 文件中加入 catlog=[...] 来指定。

   填写文件夹或者文件名（不包括拓展名）。比如 menus> ___init___.py

   catlog = ['File','Edite','Image','Process','Selection','Analysis','Plugins','Window','Help']

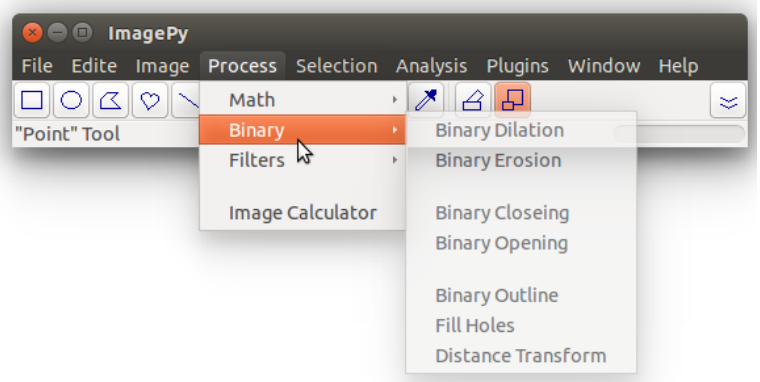

图 2: ImagePy Plugin Loader

工具加载 ToolsLoader
工具目录：imagepy>tools
加载规则：

1. 位于 tools 目录及其目录下
2. 如果文件名以 _tol.py 结尾，则文件内的 Plugin 类会被加载。
3. 如果文件名以 _tols.py 结尾，则文件内的 Plugin 列表 plgs 将依此被加载。
4. 如果文件以.mc 结尾，则会被解析成宏命令进行加载。
5. 总结：工具架子啊规则与插件加载规则基本一致，需要注意的是工具的 plgs 列表是一个二元组列表 [(Plugin,"XXX.gif")]，后者将会被当做图标加载到工具栏上。

工具与工具栏

1. 位于 tools > standard 目录下的工具称之为标准工具，始终出现在工具栏左侧。
2. 其他目录下的工具称为专业工具，共用右侧空间，可以点击最左侧下拉按钮进行切换。

调用顺序：与插件的调用有顺序一致。
工具栏图标
准备一个与工具文件名相同（不包括后缀 _tol.py）的 16×16 的 gif 小图标，放在相同目录下。
当程序被加载时，将作为该工具的图标。一个 Measure 工具集的例子，以下是 ___init___.py 文件中的内容：catlog = ['coordinate_tol', 'distance_tol', 'angle_tol', 'area_tol']

2.3 管理器 Manager

Manager: imagepy > core > managers
管理器是 ImagePy 中的一个重要的概念，作为插件系统，部件高内聚低耦合，而管理器是全局协调各个部件的。

1. WindowsManager：最重要的管理器，负责管理和调度窗口。
   - add(cls,win): 将窗口添加到管理器中（窗口打开时自动添加）
   - remove(cls,win): 将窗口从管理器中移除（窗口关闭时自动移除）

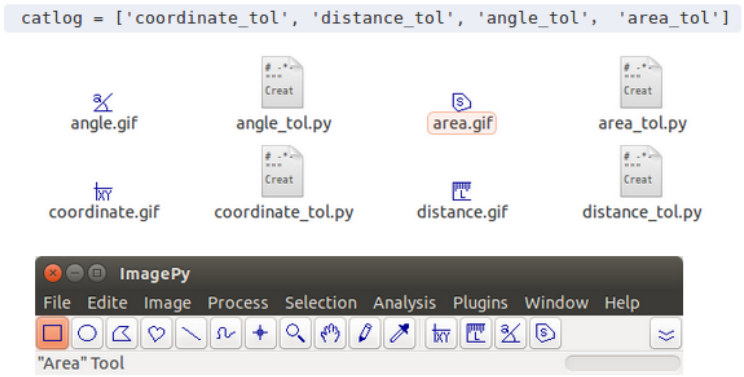

图 3: ImagePy Tool Loader

- get(cls,title=None): 获取制定的窗口，如果没有指定，则返回最前端的一个。
- get_titles(cls): 获取当前所有窗口的标题序列。
- name(cls,name): 获得命名，如果没有重复则返回本身，如果有重复则添加数字后缀。
- close(cls,name)：关闭窗口。

2. TextLogManager: 日志管理器，与 WindowsManager 用法一致。
3. TableLogManager: 表格窗口管理器，与 WindowsManager 用法一致。
4. PluginsManager: 插件管理器，插件被 PluginsLoader 解析后，会以键值对的形式保存在这里。
   - add(cls, plg): 将插件添加到管理器。
   - get(cls, name): 根据名称获取插件实例。
5. ToolsManager: 工具管理器，工具被 ToolsLoader 解析后，会以键值对的形式保存在这里。
   - add(cls, plg): 将工具添加到管理器。
   - get(cls, name): 根据名称获取工具实例。
   - set(cls, tool): 设置当前工具，一般有工具栏的点击事件触发。
6. RoiManager: 用于存储和选取，方便在需要的时候加载。
   - add(cls,name,roi)：将选区添加到管理器。
   - get(cls, name): 根据名称获取选区实例。
7. ColorManager：颜色管理器。
   - get_color(cls): 弹出颜色对话框，交互式获取颜色。
   - set_front(cls,color): 设置前景色。
   - set_back(cls,color): 设置背景色。
   - get_front(cls,one=False): 获取前景色。
   - get_back(cls,one): 获取背景色。
   - get_lut(cls,name=”grays”): 获取索引表，默认是灰度色阶。

8. ClipBoardManager: 剪切板管理器
    - roi: 剪切的选区
    - img: 剪切的图像

2.4 常用功能汇总 IPy

IPy：imagepy > IPy

IPy 是一些常见功能的汇总，如获取当前窗口、显示图片等功能，这些功能本来是分布在各个功能组件中，IPy 只是重新引用和整理，并且放置在了包的根目录，这样可以方便地引用，提高开发效率。

- currapp=None 当前程序主窗口对象
- get_window() 获取当前窗口
- get_ips() 获取当前的 ImagePlus
- show_img(imgs,title) 展示图片，标题设置为 title
- alert(info, title="image-py") 弹出警告框，默认标题为"image-py"
- yes_no(info, title="image-py") 弹出是非对话框，默认标题是"image-py"
- get_path(title,filt,para=None) 弹出选取文件对话框，结果填写在 para 上
- get_dir(title,filt,para=None) 弹出获取目录对话框，结果填写在 para
- get_para(title,view,para) 调用 ParaDialog，解析 view、para
- table(title, data, cols=None, rows=None) 填写表格、标题行、数据块、列标、行标
- write(cont,title="ImagePy") 写日志，默认标题是"ImagePy"
- set_process(i) 设置进度条
- set_info(i) 设置状态栏信息
- run_macros(cmds) 执行宏

2.5 图像类 ImagePlus

ImagePy 采用 numpy.ndarray 多维数组作为图像数据结构。虽然 Numpy 很强大，但是如果希望它能够支持图像处理的全部特性（索引色支持、撤销、选区支持、多通道支持、图像栈支持等）。还需要一些其他的数据结构予以必要的辅助，因此，定义了 ImagePlus 类。

ImagePlus 类图

如图4所示，可以看出，ImagePlus 是一个 Numpy 的容器，撞在了若干张图片，一张缓冲图，一个选区，一个掩模以及一个标记。

一些概念：

- 图像序列与当前图像: ImagePlus 能够存储一系列图片，然而我们只能够展示其中一张，这样就有了游标和当前图像的概念。可以获取图像列表，也可以获取当前图像。
- LUT(look up table): 索引色存放在 ImagePlus 上，如果是单通道图像，那么在图像被绘制在 Canvas 上时，会自动套用索引。

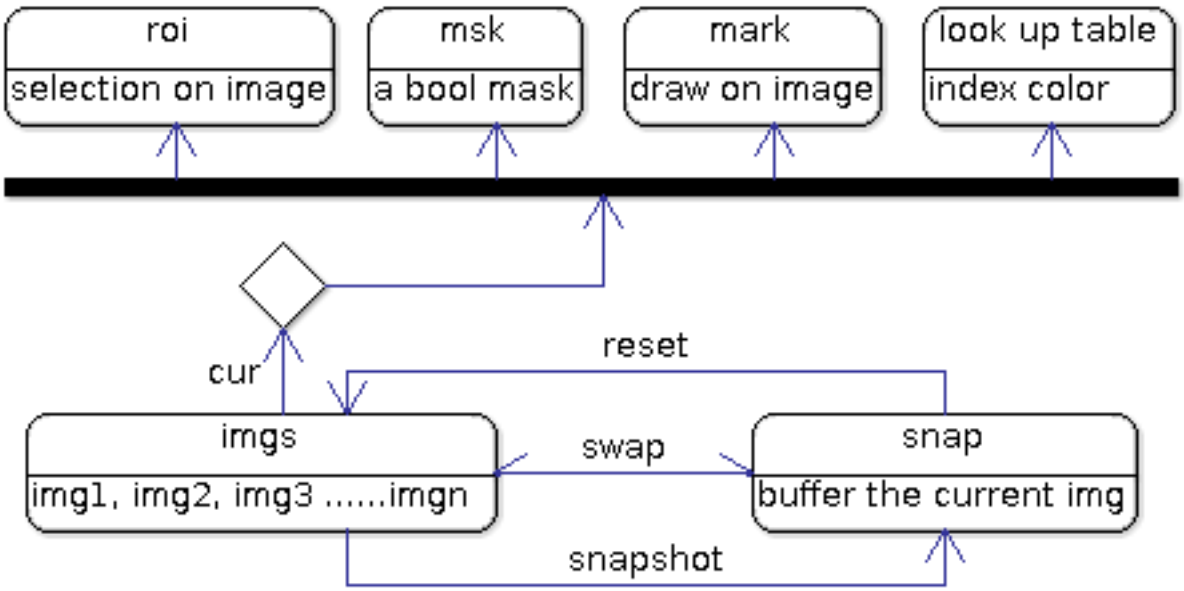

图 4: ImagePy ImagePlus

- snapshot: ImagePlus 为我们准备了一个缓冲图像，最直观的看，可以实现一步撤销功能。更重要的是，一些滤波运算中必须有一张缓冲副本才能够实施算法。有了 snap 的存在，可以节省不必要的内存开支，高效地完成特定任务。
- reset: 利用快照 snap 恢复当前图此案够，如果存在掩模，则只恢复掩模内的像素。
- swap: 交换当前图像和快照缓冲图像，也就是实现 ImagePy 的撤销功能。
- roi 和 mask: roi 是通过交互选择的选区，而 mask 则是一张二值标记图像。通常，ImagePy 自动使用 roi 绘制 mask。
- mark: 和显示相关，在图像上展示必要的提示信息。这些额外的信息可以给我们一些帮助。如 roi>mask+snap+reset 选区外的内容，这样就实现了只对选区内的像素进行处理的功能。本质上是整幅图像都已经处理过后，随后利用镜像恢复选区以外的内容，这样看起来就像是处理了选区内的像素。

类总览

- 成员
  - imgs: 所持有的图像序列（单幅图也作为序列）
  - imgtype: 图像类型，8-bit, 16-bit, rgb, float
  - title: 图像的标题
  - snap：图像快照
  - roi: 与之相关的选区
  - mark: 与之相关的标记
  - mask: 与之相关的掩模，多是由 roi 自动生成
  - lut: 索引表

- 方法
  - 设置属性 set_imgs: 设置图像序列 set_cur: 设置当前帧 set_title: 设置图像标题
  - 获取属性 get_img: 获取当前图像 get_imgtype: 获取当前图像类型 ge

图像和图像栈：ImagePlus 构造或设定图像时，需要传图图像序列 imgs。本质上是一个图像容器，可以装载单张或者多张图像，区别如下：

- 单幅图像传入单个图像，也需要构造一个序列：[img]
- 多图图像对于多幅图像，有两种形式，即离散列表和连续栈。
  - 离散列表 [img1,img2,...]
  - 连续栈 3d/4d numpy.ndarray

  其中，离散列表内存不连续，因此可以随时进行插入或者删除的切片操作。而为了计算方便，连续栈在内存中占据了连续的空间，因此不能够进行动态地添加或者删除切片等改变图像栈大小的操作。
- 混淆如果有一张三通道的彩色图像，注意两种构造方式的区别：
  -
  $$rgb$$
  表示一个离散列表，装有一幅彩色图像
  - rgb 表示一个连续栈，装有 3 张图像。

2.6 参数交互 ParaDialog

对于做科学计算的程序员来说，制作界面编程总是一件令人不愉快的事情。为了降低界面制作的难度，定义了 ParaDiagram 类，采用了数据驱动试图的设计思想，会根据数据自动生成交互环境，同时在数据发生变化的时候可以进行回调，实时预览。

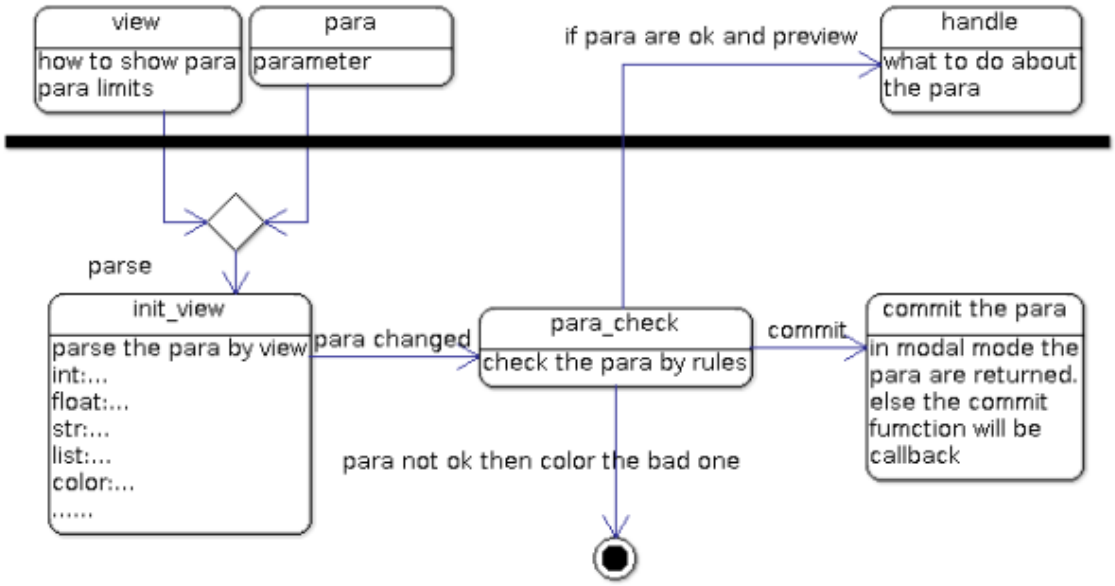

图 5: ImagePy Parameter Dialog

类结构

```
class ParaDialog (wx.Dialog):
    def __init__( self, parent, title):
      ...
    def init_view(self, items, para, preview=False, modal = True):
        # init_view: 根据items、para 圣城街面， preview 是窗口是否需要添加预览复选框，model指明窗口是否是模态对话框。
        # ...
    def para_check(self, para, key):
        # para_check: 参数改变时调用，key是改变的参数键，检查参数是否合法，如果全部合法则返回True。
        # ...
    def para_changed(self, key):
        # para_changed: 参数改变时调用，并在预览状态下自动调用handle。
        # ...
    def set_handle(self, handle):
        # set_handle: 设定处理函数，将会在参数改变并打开预览的情况下调用。
        # ...
```

ParaDialog 支持 int、float、slide、bool、str、list、color、lab、img、tab、ctrl 等。

- int: (int,(0,10),0,"title","key","unit"), (整数，0-10，无小数，标题，键，单位)
- float: (float,(0,10),2,"title","key","unit"), (浮点，0-1，两位小数，标题，键，单位)
- slide: (slide,(0,10),"title","key","unit"), 与 int 类似，只是用滑块进行交互。
- bool: (bool,"title","key"), (bool，标题，键)
- str: (str,"title","key","unit"), (字符串标题，键，单位)
- list: (list,"type","title","key","unit"), (列表，类型，标题，键，单位)
- color: ("color","title","key","unit"), (颜色，标题，键，单位)
- lab: ("lab","cont")，展示一段提示语。
- img: ("img","title","key","unit"), 从框架的 Manager 中获取一幅图像。
- tab: ("tab","title","key","unit"), 从框架的 Manager 中获取一张表。
- ctrl: ("ctrl",key,ctrl), 添加任意的控件。

一个简单的例子：

```
para = {'m':1}
view = [(float, (0,150), 1, 'weight', 'm', 'kg')]
IPy.get_para('Test', view, para)
```

解释：有一个参数 m, 默认值为 1，(int, (0,150), 1，‘weight’，‘m’，‘kg’) 的意思是说，该参数之一个整数，取值范围在 0-150 之间，有一位小数精度，交互提示为 weight，对应与参数 m，单位是 kg。通过图 6，我们清楚地看到了，当输入操作超出给定的范围或精度时，背景变成了黄色。

一个较完整的例子：

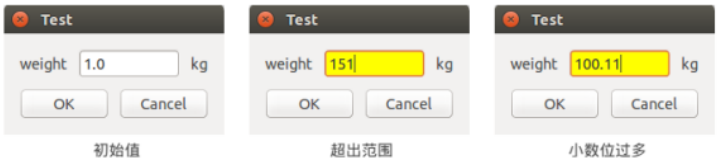

图 6: ImagePy Parameter Dialog Ex1

```
para = {'i':10, 'f':1.0, 's':50, 'b':True, 'str':'abc', 'l':'2','c':(255,0,0)}
view = [('lab', 'This is a parameter test'),
        (int, (0,150), 1, 'int', 'i', 'u'),
        (float, (0,1), 1, 'float', 'f', 'u'),
        ('slide', (0,150), 'slide', 's', 'u'),
        (bool, 'this is a bool test', 'b'),
        (str, 'str', 'str', 'u'),
        (list, ['1','2','3'], int, 'list', 'l', 'u'),
        ('color', 'color','c','u')]
IPy.get_para('Test', view, para)
```

效果如图 7

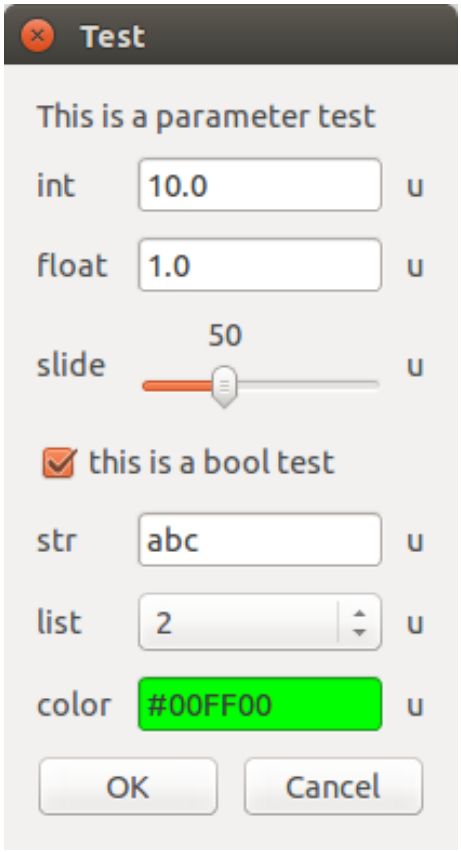

图 7: ImagePy Parameter Dialog Ex2

2.7 滤波器 Filter

Filter 是图像处理插件的基类，用于处理处理图形，继承该类并进行简单的配置，可以自动进行很多预处理和后处理工作。

Filter 类图如图 8，主要功能是：

- 进行必要的参数类型检查。
- 帮助生成参数交互（对话框），并支持实时预览。
- 提供快照、撤销、选区、多通道、图像栈等功能。

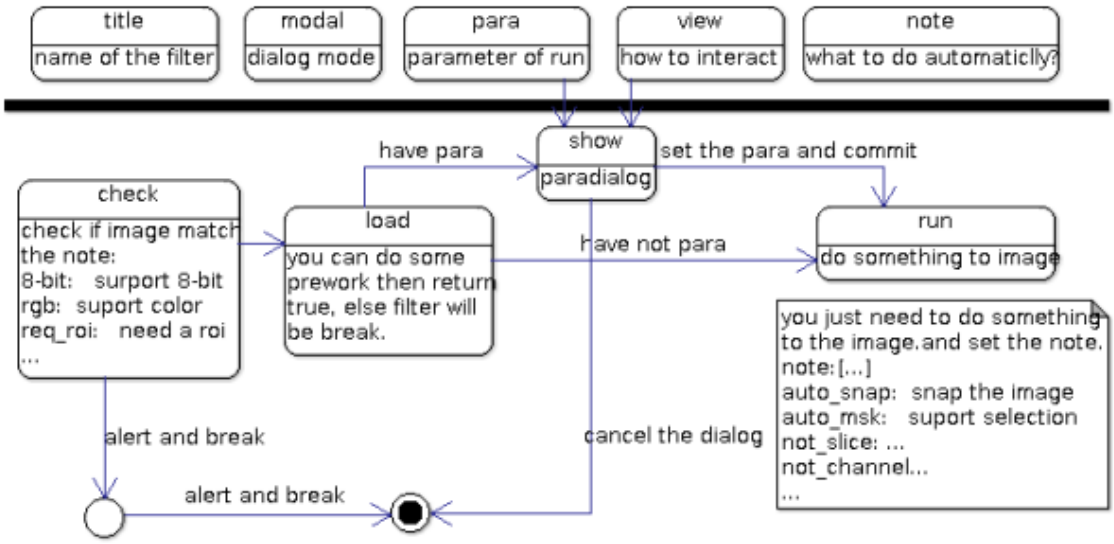

图 8: ImagePy Filter

```
1  class Filter:
2      # title: 插件的标题，默认为"Filter"，作为菜单栏的显示、交互对话框的标题、以
                                                    及插件管理器中的主键。
3      title = 'Filter'
4      # modal: 是否是模态对话框。默认模态为True即是模态对话框。
5      modal = True
6      # para, view: 核心函数要用到的参数以及交互方式，默认为None即不需要交互。
7      para, view = None, None
8      # note: 插件标签，决定了Filter的运行属性。
9      note = []
10
11     def __init__(self, ips=None):
12         #...
13         #...
14     def show(self):
15         # 弹出交互对话框，一般只需要设置para、view，Filter会自动调用ParaDialog完
                                                    成交互对话框，必要时可以覆盖
                                                    show方法。
16         #...
17     def run(self, ips, snap, img, para = None):
18         #...
19         #...
20     def check(self, ips):
21         # 根据note标识，对当前图像进行检查，不符合要求则弹框提示退出。
22         #...
23     def preview(self, para):
24         # 根据参数进行预览
25         #...
```

```
26      def load(self, ips):
27          # 加载插件，可以进行预处理（如获取图像直方图）并进行检查（如是否为二值图像）。
28          # 返回为True则继续执行后续流程，返回为False则终止流程。
29          #...
30      def start(self, para=None):
31          # 启动函数。当Filter被执行时调用，参数para为None则直接进入交互模式；否则执行run。
32          # 菜单点击传入None，运行宏可以传入para参数。
33          #...
```

关键部分再释义：

- note，插件标签，决定了 Filter 的运行特性，参数解释：
  - 检查类（作用于 check）
    - all: 滤镜能处理所有类型
    - 8_bit: 滤镜能处理 8 位灰度图像
    - 16_bit: 滤镜能处理 16 位灰度图像
    - rgb: 滤镜能处理 rgb 彩色图像
    - float: 滤镜能处理浮点图像
    - req_roi: 滤镜需要选区
  - 行为类（作用于 run）
    - preview: 是否需要提供预览功能
    - auto_snap: 是否需要执行前自动快照
    - auto_msk: 是否要支持选区
    - not_channel: 是否在处理彩色图像时自动处理每个通道（如不填写为是）
    - not_slice: 是否在处理图像栈的时候询问，从而处理每个层（如不填写为是）
    - 2int: 如果精度低于 16 位整数，是否在处理之前把图像转为 16 位整数（一些运算会产生负数或溢出）
    - 2float: 如果精度低于 32 位浮点，是否在处理之前把图像转为 32 位浮点（一些运算需要在浮点上做才能保证质量）
- run(self, ips, snap, img, para = None)，核心函数，参数含义分别是：
  - ips: 当前处理的 ImagePlus。
  - snap: 当前 ImagePlus 的快照（只有 note 设置了 auto_snap 标签后才会自动生成）。
  - img : 处理完当前 ImagePlus 后的返回值。
  - para: 当前的处理函数的参数。

一个高斯模糊的例子

```
1   import scipy.ndimage as nimg
2   from core.engines import Filter
3   class Gaussian(Filter):
4       title = 'Gaussian'
5       note = ['all', 'auto_msk', 'auto_snap','preview']
6       #parameter
7       para = {'sigma':2}
```

```
8         view = [(float, (0,30), 1,  'sigma', 'sigma', 'pix')]
9         #process
10        def run(self, ips, snap, img, para = None):
11            nimg.gaussian_filter(snap, para['sigma'], output=img)
```

效果如图 9所示：我们并没有做任何关于彩色图像的处理，以及如何处理选区，仅仅 9 行代码，我们就得到了一个界面友好的交互式滤镜，并且它支持任意图像类型，支持选区，能够撤销！

图 9: ImagePy Filter Ex1

2.8 整体操作 Simple

Simple 是另一个重要的图像处理引擎，结构与 Filter 大体一致，可以类比学习。注意 Simple 不是针对单张图像，而是针对整个图像序列，或者 ImagePlus 本身进行操作，如撤销操作、选区扩张操作、或者 3D 滤波器操作等。由于 Simple 是针对整体进行处理，因而相比 Filter 不需要自动便利通道和图像栈，因而也没有行为参数和行为函数。

```
1   class Simple:
2       # title: 插件的标题，默认值是"Simple"，用作菜单栏的显示、交互式对话框的标
                                                    题、以及插件管理器中的主键。
3       title = 'Simple'
4       # para,view: 核心函数需要用到的参数，以及他们的交互方式，默认为 None，代表不
                                                    需要交互。
5       para, view = None, None
6       note = []
7
8       def __init__(self, ips=None):
9           #...
10          #...
11      def show(self):
12          # 弹出交互对话框，一般只需要设置para、view，Filter会自动调用ParaDialog完
                                                    成交互对话框，必要时可以覆盖
                                                    show方法。
13          #...
14      def run(self, ips, imgs, para = None):
15          # 核心函数，我们将得到如下参数：ips, imgs, para = None，解释如下：
16          # ips:当前处理的 ImagePlus
17          # imgs:当前处理的 ImagePlus 的图像序列
18          # para:当前的处理函数参数
19          #...
```

```
    def check(self, ips):
        # 根据 note 标识，对当前图像进行检查，如果不符合要求则弹出提示，退出。
        #...
    def load(self, ips):
        # 加载插件，可以进行预处理（如获取图像直方图）并进行检查（如是否为二值图像）。
        # 返回为True则继续执行后续流程，返回为False则终止流程。
        #...
    def start(self, para=None):
        # 启动函数。当Filter被执行时调用，参数para为None则直接进入交互模式；否则执行run。
        # 菜单点击传入None，运行宏可以传入para参数。
        #...
```

关于插件标签，他们指明 Simple 能处理哪些类型, note 插件标签的说明：

- all: 滤镜能处理所有类型
- 8_bit: 滤镜能处理 8 位灰度图像
- 16_bit: 滤镜能处理 16 位灰度图像
- rgb: 滤镜能处理 rgb 彩色图像
- float: 滤镜能处理浮点图像
- req_roi: 滤镜需要选区
- req_stack: 滤镜需要图像栈
- req_stack2d: 滤镜需要图像列表
- req_stack3d: 滤镜需要连续图像栈

一个使用 ImagePy 进行选取扩张的例子：

```
from core.engines import Simple
class Inflate(Simple):
    title = 'Inflate'
    note = ['all', 'req_roi']
    para = {'r':5}
    view = [(int, (1,100),0, 'radius', 'r','pix')]
    def run(self, ips, imgs, para = None):
        ips.roi = ips.roi.buffer(para['r'])
```

注意：Inflate 既然是要扩张选区，那么自然要求图像必须有选区，我们在 note 里加上”req_roi”，这样我们可以免于在执行之前判断是否存在选区，以及如果不存在如何提示用户，只需专心做核心工作。效果如图 10

2.9 自由插件 Free

Free 是与系统相关最弱的一种引擎，仅仅是完成一个任务。之所以称之为 Free，是因为它可以不依赖于图像而运行，不像 Filter、Simple 等的处理对象都是 ImagePlus（如果没有图像，则经过 check 函数检查后会提示并终止运行）。Free 可以用来处理一些与图像处理无关的任务，如打开、新建图像，打开主题帮助等。把这些部分以插件形式提供，降低系统耦合度，便于拓展和维护。

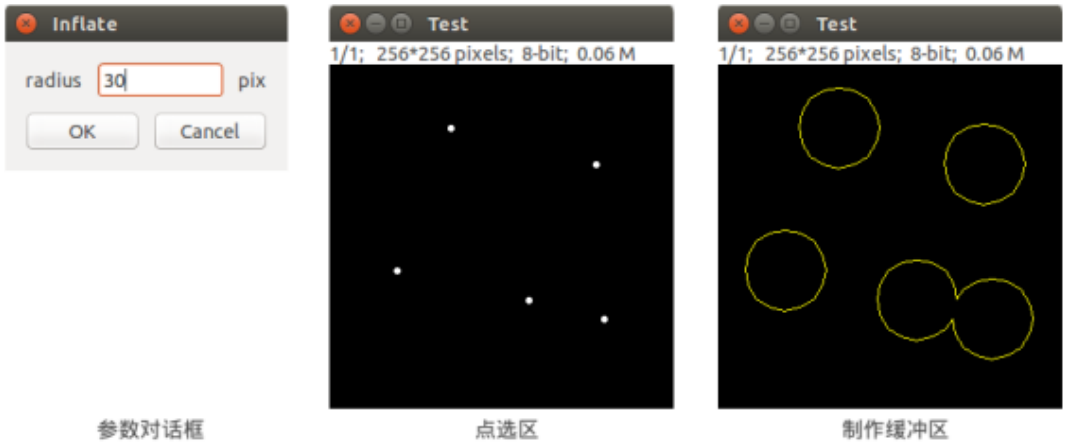

图 10: ImagePy Simple Ex1

```
class Free:
    # title:插件标题，默认值是"Free",作为菜单栏显示、交互对话框的标题、以及插件管理器中的主键。
    title = 'Free'
    # 核心函数用到的参数及交互方式，默认None不交互。
    para, view = None, None

    def __init__(self):
        #...
    def show(self):
        # 弹出交互对话框，需要设定para和view，Filter会自动调用ParaDialog生成交互对话框。必要时可以覆盖show方法。
        #...
    def run(self, para = None):
        # 核心函数，para将作为参数传入。
        #...
    def start(self, para=None):
        # 启动函数，放Free被执行时调用，参数para如果是None，则进入交互模式。如果有值，则直接运行run。
        # 菜单点击的方式下都是传入的 None，而运行宏的时候，可以传入 para 参数。
        #...
```

一个例子

```
from core.engines import Free
class OpenFile(Free):
    title = 'Open'
    para = {'path':''}

    def show(self):
        filt = 'BMP files (*.bmp)|*.bmp|PNG files (*.png)|*.png|JPG files (*.jpg)|*.jpg|GIF files (*.gif)|*.gif'
        return IPy.getpath('Open..', filt, self.para)
    def run(self, para = None):
        path = para['path']
        fp, fn = os.path.split(path)
        fn, fe = os.path.splitext(fn)
```

```
13          img = imread(path)
14          IPy.show_img([img], fn)
```

注意：在 ImagePy 里，一些与图像处理不相关的工作都是继承于 Free 的，可以是打开图像，查看版本号，自动更新，甚至是退出程序。

2.10 宏引擎 Macros

Macros 类是一个宏执行器引擎，负责将一串 ImagePy 命令依次执行。Macros 是为了在 ImagePy 中实现宏功能，统一为一种引擎接口而设计的辅助类。

```
1  class Macros:
2      # title: 插件的标题，将作为菜单栏的显示，交互对话框的标题，以及插件管理器中
                                                    的主键。
3      title = 'Macros'
4      # 宏命令列表
5      cmds = [....]
6      def run(self):
7          # 依次执行cmds中的每一条命令
8          # ...
9
10     def start(self, para=None):
11         # 代理函数，执行run
12         # ...
```

实现机制：宏录制功能应该是 ImagePy 里非常炫的一个功能了，得益于整体的高弹性设计，宏录制功能实际实现代码不操作 20 行！

以高斯模糊为例，演示实现机制：

- 录制演示：Plugins > macros > Recoder
- 录制机制：不论是 Filter、Simple、Free，都是引擎类，有共同的接口 start(self,para)。当某个插件被执行后，title>para 将被录制器记录。Gaussian>”sigma”:2.0。
- 执行机制：所有的插件都被 PluginsManager 所管理，PluginsManager 你恶不实际维护了一个以插件的 title 为主键的键值对（类似于 MFC 中的消息映射宏）。所以，接下来进行一下步骤：
  1. 解析宏命令。使用 > 进行字符串分割。
  2. 使用分割后的 title 作为主键在 PluginsManager 中查找，得到滤波器实例。
  3. 调用 eval 函数，把 para 重新解析为 Python 对象（充分发挥了脚本语言的优势）。
  4. 执行获取的 Python 对象（这里是滤波器）的 start 方法，并把 para 当做参数输入。如果 para 为 None 进入交互模式，否则执行 run。

2.11 交互工具 Tool

Tool 交互工具。Filter、Simple 都可以处理图像，但是有时候需要用鼠标对图像进行交互操作，比如选区操作、绘图操作等。ImagePlus 被绘制在一个 Canvas 上，Canvas 是 wxpython 的 Panel 子类，可以对其添加鼠标事件，但我们并不推荐这样做，原因之一是这样做比较繁琐，其次，多工具同时注册事件，会引起管理混乱和事件冲突。

Tool 的结构图如图11所示。

事件的调用规则：

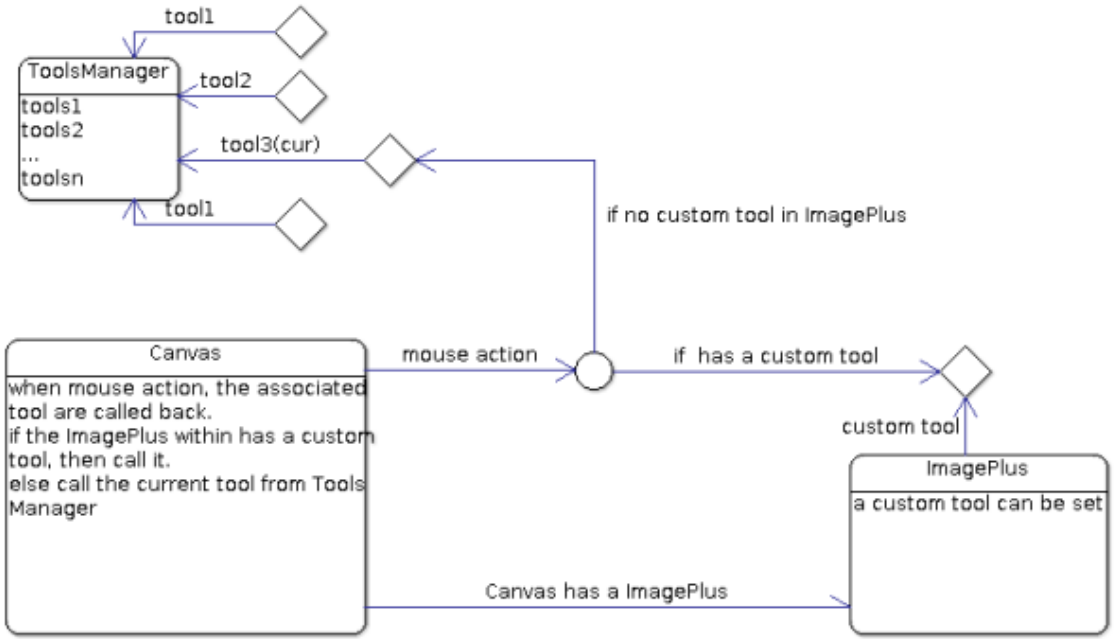

图 11: ImagePy Tool

1. 事件最初由 Canvas 类触发。

2. 如果 Canvas 持有的 ImagePlus 有一个 CustomTool，则交给它来处理。

3. 如果没有，则交给 ToolsManager 的当前工具来处理。

注意：一般来说，比如选区，画笔等工具，是全局有效的，但有时我们需要于单一图像做特定交互，比如当某个图像栈进入了三视图观察状态时，这是 CustomTool 就变得很有用。

类结构

```
class Tool:
    title = 'Tool'
    view, para = None, None

    def show(self):
        # 双击工具按钮时，弹出交互对话框。
        # 一般时候我们只需要设定 para， view，Tool 会自动调用 ParaDialog 生成交互对话框。
        # 必要时，可以覆盖 show 方法。
        #...

    # 当交互对话框确认时将会被调用，用于和ColorManager 通讯生效我们的设置。
    def config(self):pass
    # 工具被选中时调用。
    def load(self):pass
    # 由当前工具切换到另一个工具时调用（可以处理绘制了一半还没有闭合的多边形选区等）
    def switch(self):pass
    def mouse_down(self, ips, x, y, btn, **key): pass
    def mouse_up(self, ips, x, y, btn, **key): pass
    def mouse_move(self, ips, x, y, btn, **key): pass
    def mouse_wheel(self, ips, x, y, d, **key): pass
    def start(self):ToolsManager.set(self)
```

关于 mouse_xxx 的参数:

- ips: 与事件相关的 ImagePlus
- x: 数据坐标系下的 x
- y: 数据坐标系下的 y
- btn: (up/down/move) 表示按下键（左键 1，中键 2，右键 3）
- d: (wheel) 代表滚动量
- **key 其他参数
  - ”shift”:shift 是否被按下
  - ”ctrl”:ctrl 是否被按下
  - ”alt”:alt 是否被按下
  - ”canvas”: 获取与事件关联的 canvas

一个画笔的例子

```
from core.draw import paint
from core.engines import Tool
import wx
class Plugin(Tool):
    title = 'Pencil'
    view = [(int, (0,30), 0,  'width', 'width', 'pix')]
    para = {'width':1}

    def __init__(self):
        self.sta = 0
        self.paint = paint.Paint()
        self.cursor = wx.CURSOR_CROSS

    def mouse_down(self, ips, x, y, btn, **key):
        self.sta = 1
        self.paint.set_curpt(x,y)
        ips.snapshot()

    def mouse_up(self, ips, x, y, btn, **key):
        self.sta = 0

    def mouse_move(self, ips, x, y, btn, **key):
        if self.sta==0:return
        self.paint.lineto(ips.get_img(),x,y, self.para['width'])
        ips.update = True

    def mouse_wheel(self, ips, x, y, d, **key):pass
```

效果如图 12

3 操作手册

3.1 界面构成

1. 主界面如图13所示。

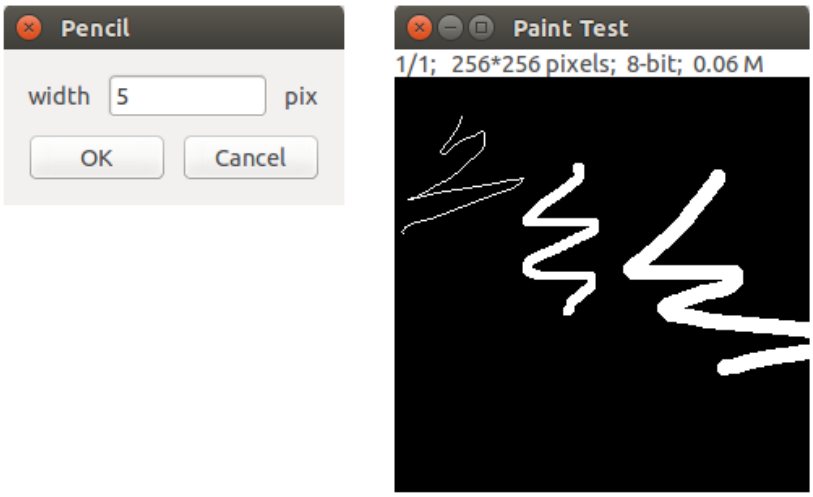

图 12: ImagePy Tool Example

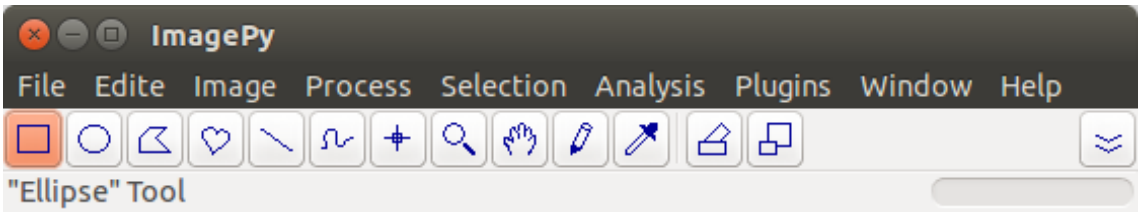

图 13: ImagePy Main Frame

2. 菜单栏

   - File：打开，保存图像，批量导入，保存图像栈
   - Edite：剪切，复制，裁剪，描边，填充
   - Image：类型转换，亮度，对比度，色彩转换，索引颜色，几何变换，图像栈操作，画布大小
   - Process：数学运算，滤波器，形态学，两附图之间的数学运算
   - Selection：选区的操作，矢量交，并，补运算，缓冲区生成，选区管理
   - Analysis：区域标记，像素统计
   - Plugins：宏录制，插件浏览，插件模糊查询，扩展插件
   - Window：窗口管理
   - Help：主题帮助

3. 工具栏工具通过鼠标交互，作用在图像上，包括图像查看，放大，缩小，在图像上指定选区，绘画，取色等，以及量距等。

4. 状态栏展示一些与当前操作相关的信息，当图像进行一些耗时工作时，会启用进度条。

3.2 相关概念

图像处理的一些基本概念。

- 图像概念: 数字图像处理中所指的图像，确切地说叫光栅图，是点阵构成的，所以也称之为点阵图或位图（bit map），它的大小又深度和尺寸决定。
- 尺寸: 构成图像的点阵的行数和列数，这决定了图像对空间细节的表现能力，所以也称之为分辨率。
- 格式，深度: 每个点阵的数据类型，如每个点是一个 [0,255] 的整数，是一个小数…每种数据类型在计算机中占特定的位，我们称单个像素所占的位称为图像的深度，这决定了单个点的信息描述能力，因此有些时候深度也称之为色彩分辨率。
- 通道: 对于一些图像（比如彩色图像），包含了多个尺寸相同的点阵，我们称每个单一层为一个通道，这些通道各自描述了图像的某个维度的信息，只有合成之后，才是完整的。
- 存储格式: 位图是一个一个的点数据，因此占据大量的存储空间，因而存储时可以对其进行压缩，由不同的原理得到的各种算法，就形成了存储格式，常见的有 Bmp(未压缩), jpg, png, tif 等，不同与位图，他们的存储大小除了尺寸，还取决与像素的内容，不过无论采用什么压缩格式，在进行数字图像处理时，往往要先解压缩，在内存中形成位图。注意，与像素数据格式是两个概念。

ImagePy 用到的一些概念

1. 图像栈: 一些有相同尺寸，相同格式的图像构成的图像序列，称之为图像栈。ImagePy 中图像栈分为连续栈和列表栈，其中列表栈支持随时动态添加、删除图像，而连续栈则不可以。一些三维滤波器则必须在连续栈上才能运行。二者可以转换。
2. 索引色: 对于灰度图像，有时会通过特定的映射关系，把它映射成彩色。这的确增强了图像的肉眼可判读性，但并没有增加其包含的信息量，因而有时也称之为伪彩色。示例如图 7。

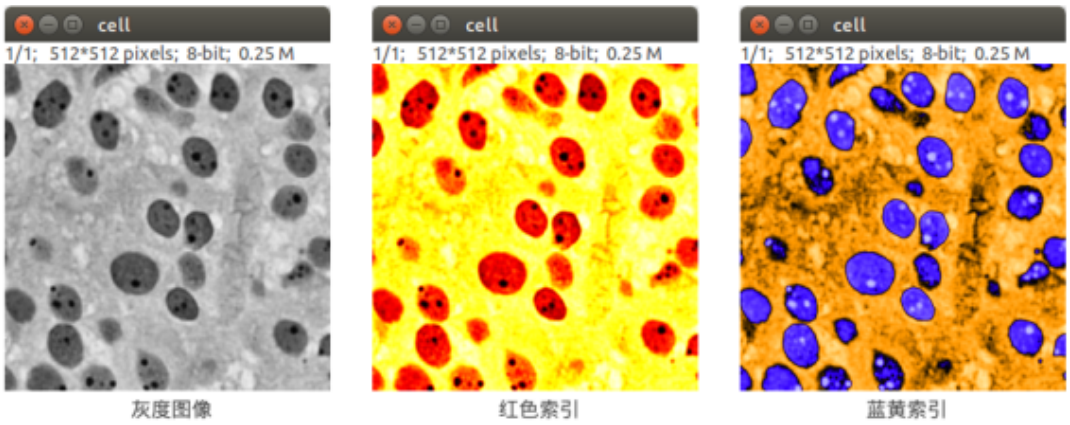

图 14: ImagePy Colormap/LukUpTable

3. 选区: 顾名思义，它只是用于指定图像的一块区域，自身并没有什么作用，只是为其他的操作指明目标。在 ImagePy 里，选区是指一个叠加在图像上的矢量图形，可以是点，线，面，镂空多边形。
4. 掩膜: 一个与图像大小一样，全部像素由 0-1 构成的一张位图。类似与选区，但掩膜更直接地指明某个像素是否处于被选中状态，这有助于一些相关操作。(掩膜可以由选区填充而得到)
5. 标记: 是一个叠加在图像上的可绘制的一个层，可以是适量图形，位图，文字等，标记不对图像产生任何作用和影响，仅仅用于提示人。

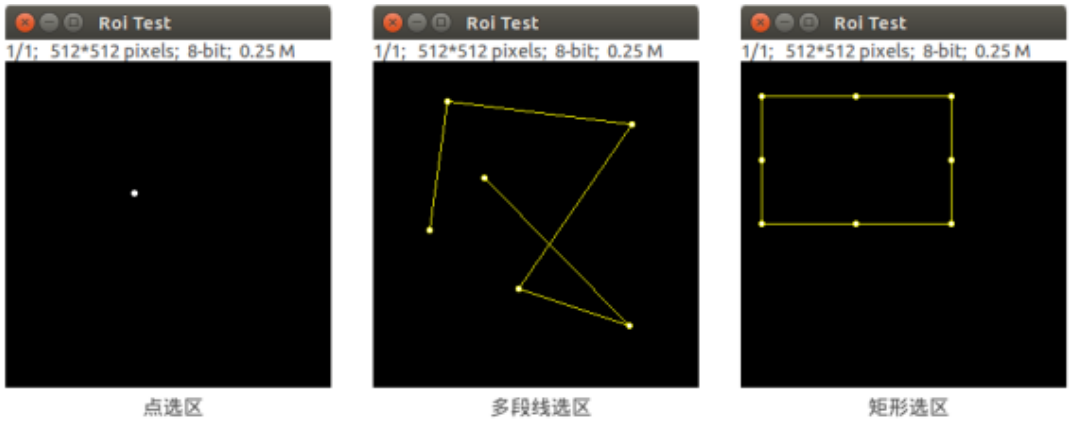

图 15: ImagePy Select Region

3.3 打开图像

- Menus > File > New
- Menus > File > Open
- Menus > File > Open Url
- Menus > Import > Sequence
- Menus > Import > Save
- Menus > Import > Save As > Save Sequence

3.4 常用工具

1. 选取类: 这些工具能够通过鼠标操作，在画布上绘制一个选取，ImagePy 支持点，线，面，复杂多边形。

    - Rectangle: 指定矩形选区，鼠标位于矩形内部，可以拖拽移动选区，鼠标位于四角或边缘时，可以调整选取的大小。
    - Ellipse: 指定椭圆选区，鼠标位于椭圆内部，可以拖拽移动选区，鼠标位于四角或边缘时，可以拖拽调整选取的大小。
    - Polygon: 多边形选区，鼠标依次点击，最后一个点右键闭合。当鼠标位于多边形内部，可以拖拽移动选区，鼠标位于节点时，可以拖拽调整节点位置。
    - FreeArea: 自由区域选区，按下鼠标，绘制区域，抬起时自动闭合。当鼠标位于多边形内部，可以拖拽移动选区，鼠标位于节点时，可以拖拽调整节点位置。
    - Line: 多段线选区，鼠标依次点击，点右键结束。当鼠标位于节点时，可以拖拽调整节点位置。
    - FreeLine：自由轨迹选区，按下鼠标，绘制区域，抬起时自动闭合。当鼠标位于节点时，可以拖拽调整节点位置。
    - Point：鼠标点击添加一个点选区，当鼠标位于某个点附近，可以拖拽移动点的位置。
    - 关于叠加：所有的选区都可以按住 Shift 键进行叠加操作，另外面类型选区，还支持按住 Ctrl 键进行裁剪和镂空操作。

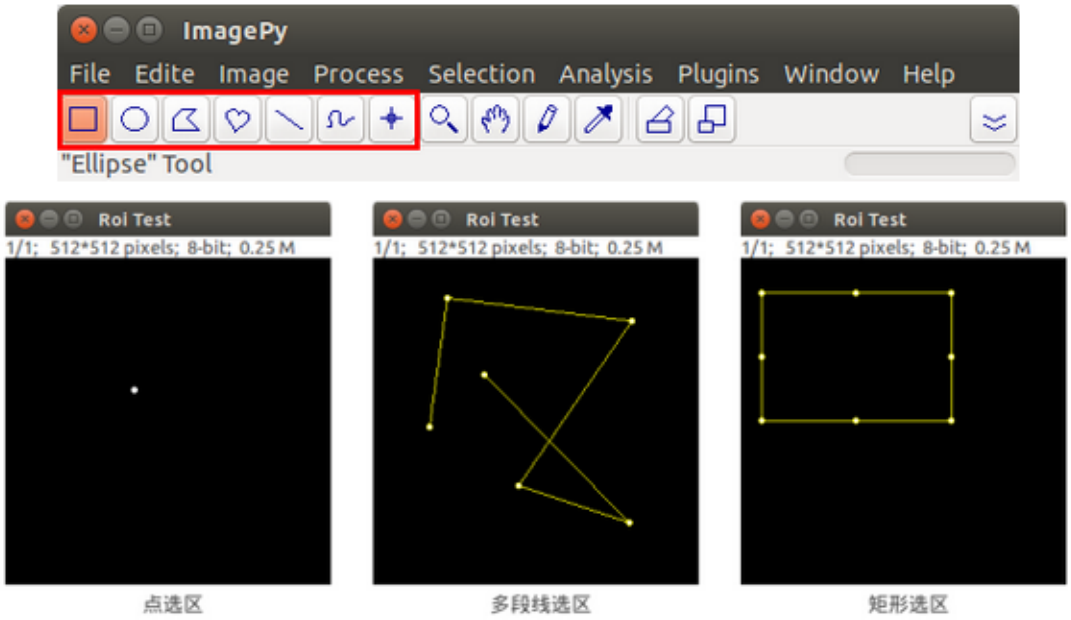

图 16: ImagePy Selection

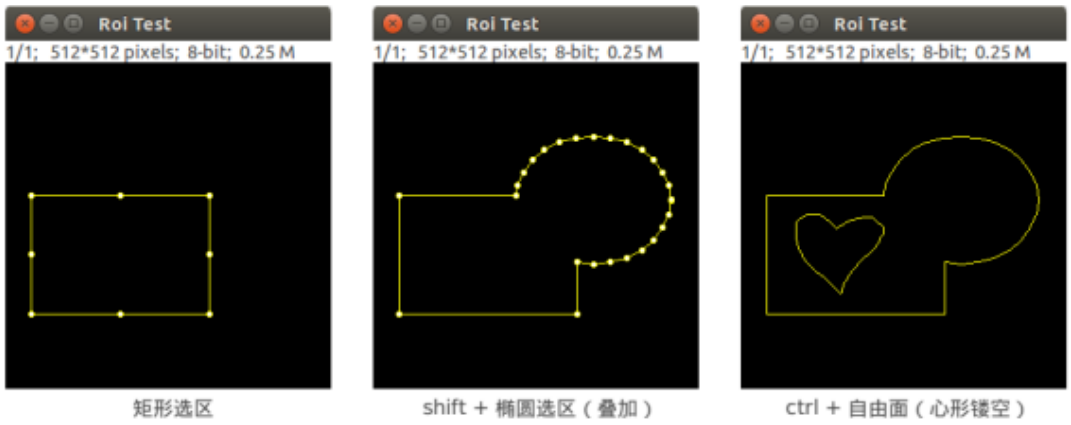

图 17: ImagePy MultiOps

2. 查看类：用于浏览图像、缩放、平移等操作。
   - Scale: 单击左键放大，右键缩小，按住滚轮拖拽可进行移动。
   - Move: 拖拽进行移动，滚轮上下翻滚，可改变比例。
3. 绘图类：简单的绘图工具，可以在画布上选取颜色，通过调色板设定颜色，在图像上绘制笔记。
4. Painter: 画笔，在画布上绘制，双击图标可以弹出对话框设置笔尖半径。
5. ColorPicker: 点击画布上的某一点摄取颜色，左键设为前景色，右键则是设置背景色。双击图标可以弹出颜色对话框。
6. 变换类: 几何变换工具，支持旋转，缩放，当有选区时只作用在选区内。
   - Rotate: 交互式进行图像旋转变换，可以用鼠标拖拽圆心到指定位置，图像将围绕该点旋转，同时可以在对话框输入各参数。

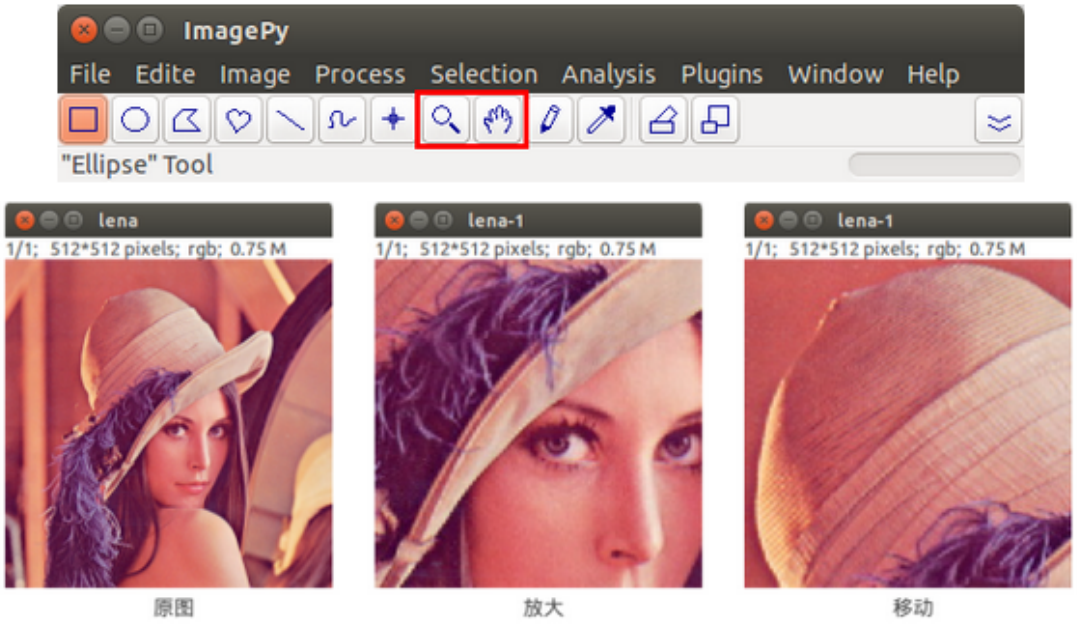

图 18: ImagePy View Image

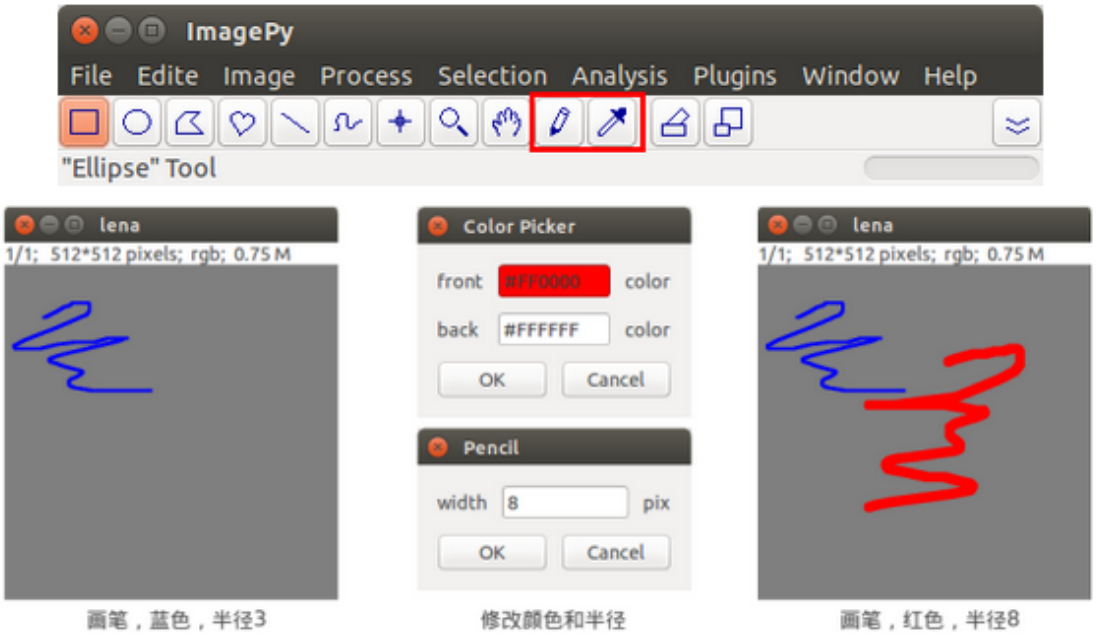

图 19: ImagePy Draw

- Scale: 交互式进行图像缩放变换，可以用鼠标拖拽选区四角和四边进行缩放，也可以在选区内拖拽，进行图像平疑，同时可以在对话框输入各参数。

7. 测量类

- Coordinate: 鼠标点击添加一个点测点，测点旁边会显示其坐标。
- Distance: 鼠标依次点击，点右键结束，形成的各点之间会用线段相连，在线段中间部位，显示线段长度。

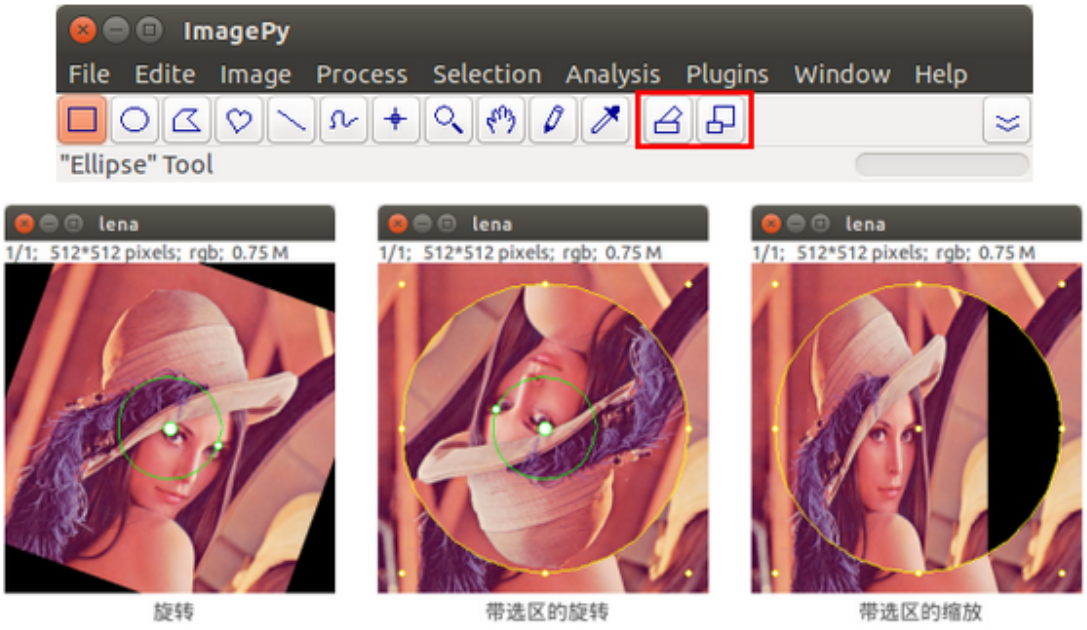

图 20: ImagePy Transform

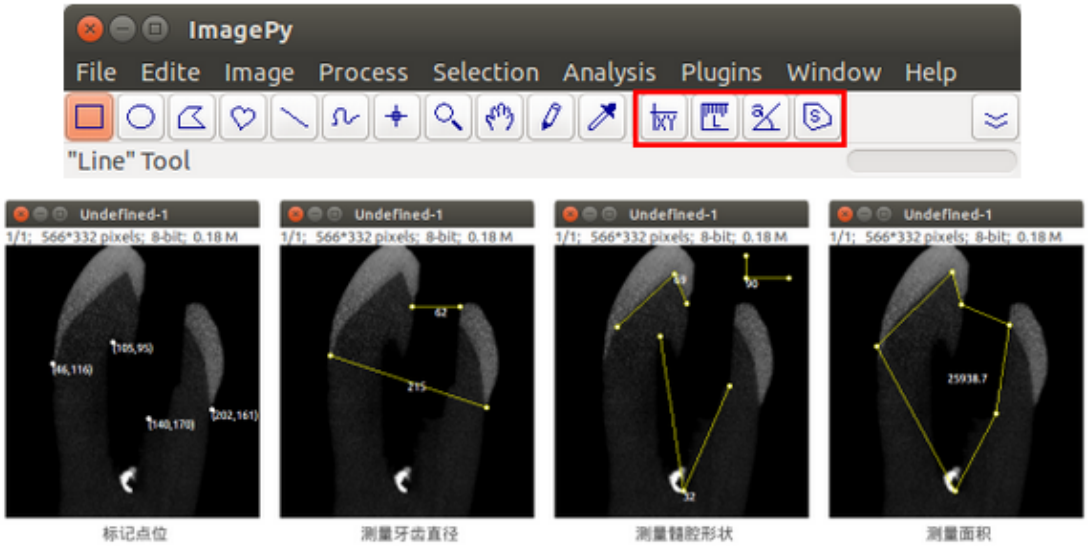

图 21: ImagePy Measurement

- Angle: 鼠标依次点击，点右键结束，形成的各点之间会用线段相连，在角点处会显示该点所链接两线段的夹角。
- Area: 鼠标依次点击，最后一个点右键闭合。多边形中间显示面积。当鼠标位于多边形内部，可以拖拽移动选区，鼠标位于节点时，可以拖拽调整节点位置。

8. 图像栈操作

- Add Slice: 在当前位置添加一张图像
- Delete Slice: 删除当前图像
- Previous Slice: 跳转到前一张

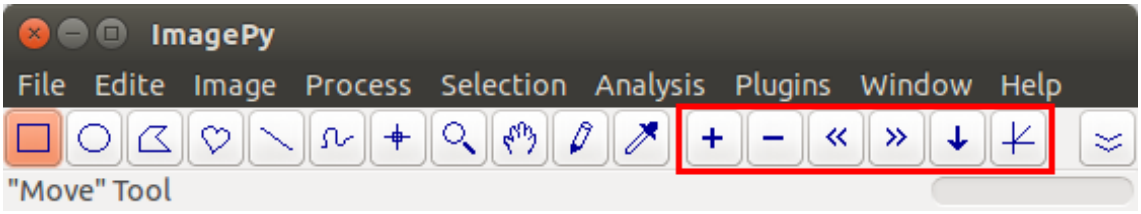

图 22: ImagePy Images Stack

- Next Slice: 跳转到后一张
- Set Slice: 跳转到特定的位置
- Orthogonal: 三视图观察

3.5 直方图调整

直方图是对图像内的像素进行频率统计而绘制出来的，他体现了图像的像素值的统计分布，属于全局特征，也称之为色阶。

ImagePy 中所有的直方图运算都支持图像预览和撤销操作，如果有选区在图像上，则自动只处理选取内的像素，在处理图像栈的时候，会询问是否进行批量处理。

所有的直方图调整，实质相当与对原图的像素值做一个 $y = kx + b$ 的运算，通过一个线性变换，映射到一个新值的过程。

1. Image > Grey Stairs：色阶调整
2. Image > Bright Constract：亮度对比度
3. Image > Threshold：阈值调整，一种非此即彼的映射，等价于对比度的极限调整。
4. Image > Color Balance：色彩平衡（只对彩色图像有效，可以理解成依次调整各通道的亮度，对比度）
5. Image > Color Stairs：彩色色阶（只对彩色图像有效，可以理解成依次调整各通道的色阶）

3.6 数学运算

直方图运算的实质是对像素值做一个线性运算，而数学运算则更广义的针对每个值做一个预定义的数学运算。

ImagePy 中所有的数学运算都支持图像预览和撤销操作，如果滤波器有参数输入会提供一个友好的交互对话框。如果有选区在图像上，则自动只处理选取内的像素，在处理图像栈的时候，会询问是否进行批量处理。

- Process > Math > Add: 加法运算（等价于亮度调整，因为加法是线性运算）
- Process > Math > Multiply: 乘法运算（等价于对比度调整，因为加法是线性运算）
- Process > Math > Max: 最大值运算，图像上各个像素与一个定值求较大值
- Process > Math > Min: 最小值运算，图像上各个像素与一个定值求较小值
- Process > Math > Squre Root: 平方根运算，对图像上各个像素求平方根

- Process > Math > Garmma: Garmma 曲线矫正，对图像进行 Garmma 修正，这在一些硬件设备的增益矫正中非常常见。
- Process > Image Calculator: 进行多图之间的运算，支持两附图之间对应像素进行加法，减法，最大，最小，差异运算。

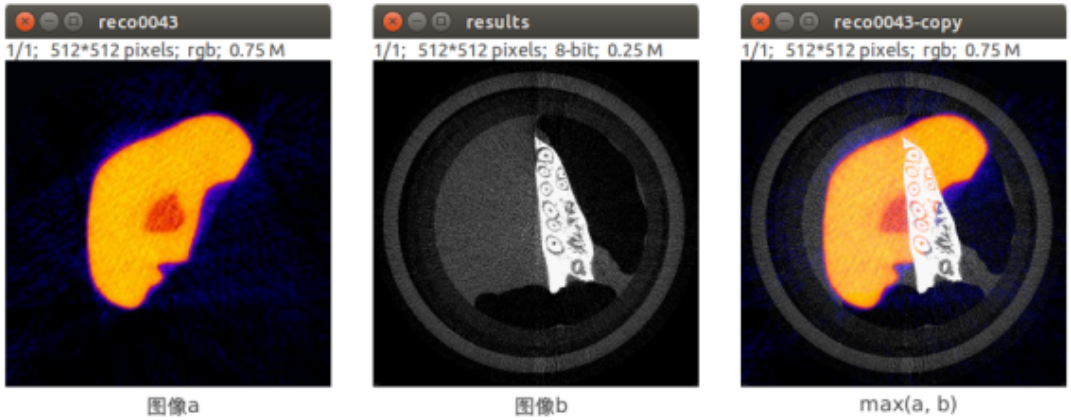

图 23: ImagePy Calculator

3.7 滤波器

前面两节的直方图调整和数学运算有个共同点，就是新图像的像素由原图相同位置的像素经过一个数学运算得到，某种意义上，直方图调整可以看作是数学运算的一个子集，而这里我们讨论的滤波器运算，新图像的像素不仅由原图相同位置的像素决定，而且和与之邻近的一些像素有关。

ImagePy 中绝大多数的滤波器（除了一些三维的）都支持图像预览和撤销操作，如果滤波器有参数输入会提供一个友好的交互对话框。如果有选区在图像上，则自动只处理选取内的像素，在处理图像栈的时候，会询问是否进行批量处理。

1. 经典高通-低通滤波器

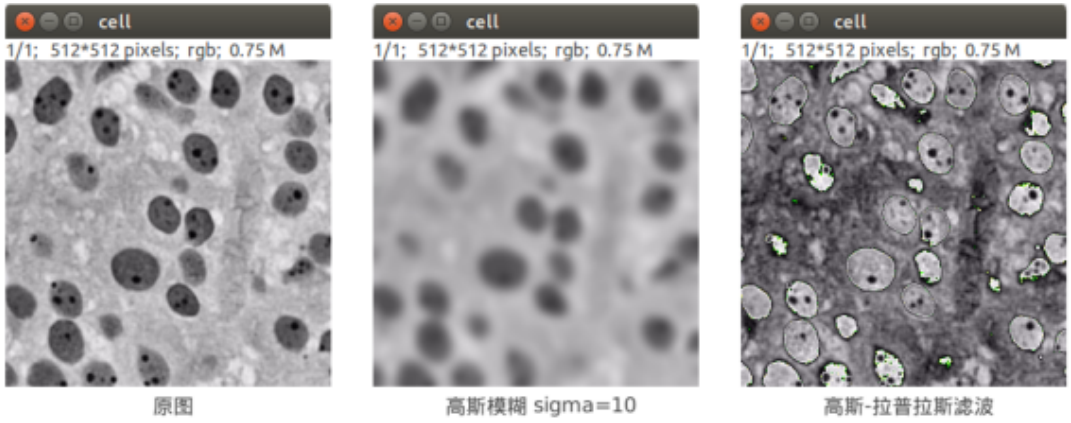

图 24: ImagePy Filter High-Low-Pass

- Process > Filter > Gaussian: 高斯模糊，由一个二维正太分布做滤波核。

- Process > Filter > Gaussian Laplace: 高斯-拉普拉斯，由一个墨西哥帽子形的函数做滤波核

2. 排序滤波器

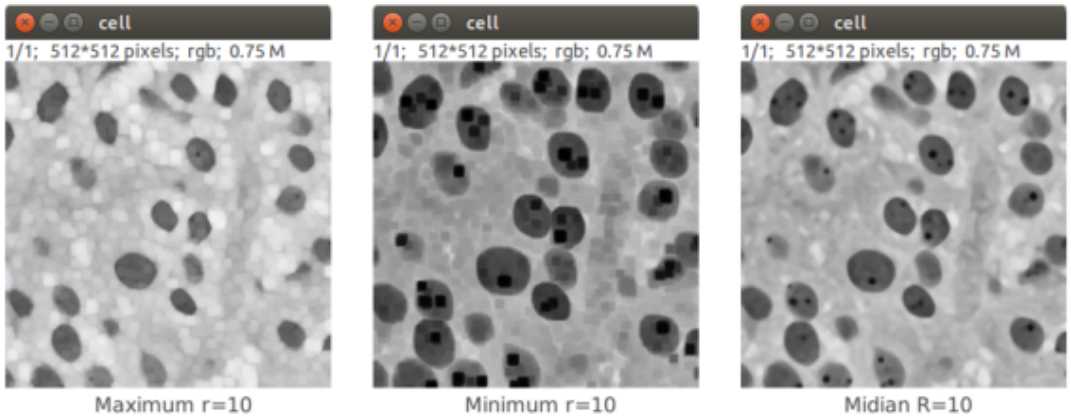

图 25: ImagePy Filter Sort

- Process > Filter > Maximum: 窗口内最大值
- Process > Filter > Minimum: 窗口内最小值
- Process > Filter > Midian: 中值滤波器

3. 基础梯度算子

- Process > Filter > Prewitt: 正交边缘算子
- Process > Filter > Sobel: 对角边缘算子

4. 其他

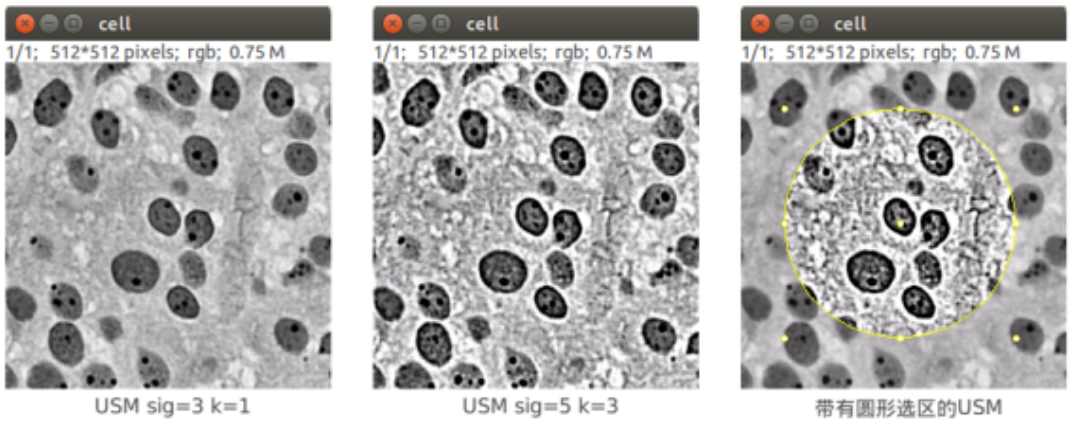

图 26: ImagePy Filter Unsharp Mask

- Process > Filter > Unsharp Mask：非锐化掩膜（原图 + k * （原图-原图的高斯模糊)），它增强了图像的细节，但同时也对噪声敏感，为了降低噪声的影响，可以在之前使用高斯滤波器。
- Process > Filter > Gaussian3D：三维滤波器，需要图像栈，在三维空间进行高斯模糊

3.8 形态学操作

前几节介绍的都是线性滤波器，这里我们介绍形态学运算，形态学多应用在二值图像中。ImagePy 中所有的形态学滤波器都支持图像预览和撤销操作，如果滤波器有参数输入会提供一个友好的交互对话框。如果有选区在图像上，则自动只处理选取内的像素，在处理图像栈的时候，会询问是否进行批量处理。

1. 基础运算: 形态学基础运算好比是一个筛子，能够透过特定的集合图形，而新图像上的点，由能否透过决定。

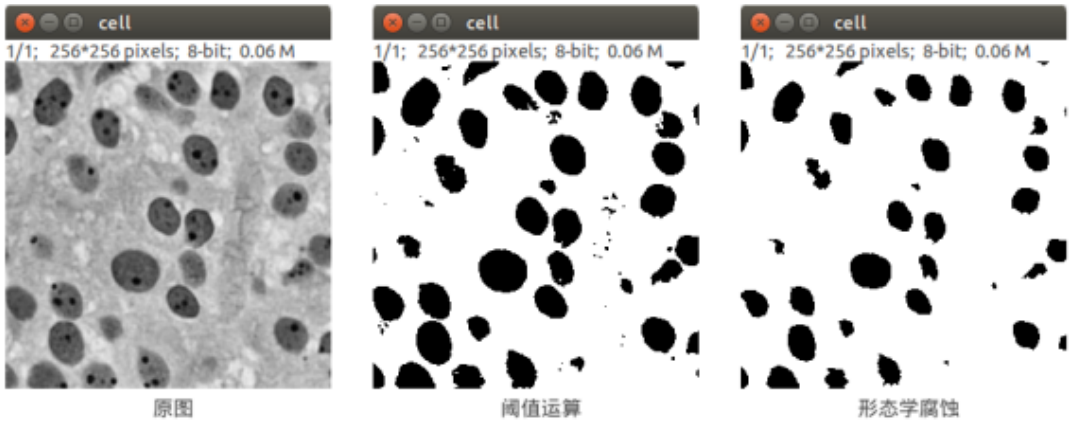

图 27: ImagePy Morphology Basic

- Process > Binary > Dilation: 膨胀运算，前景区域按照滤波核扩张
- Process > Binary > Erosion: 腐蚀运算，前景区域按照滤波核收缩
- Process > Binary > Closing: 闭运算（腐蚀 + 膨胀），这种运算能够去除前景中细微的连接
- Process > Binary > Opening: 开运算（膨胀 + 腐蚀），这种运算能够使得前景中狭小的缝隙联通

2. 高级运算: 基于二值图像的其他一些运算

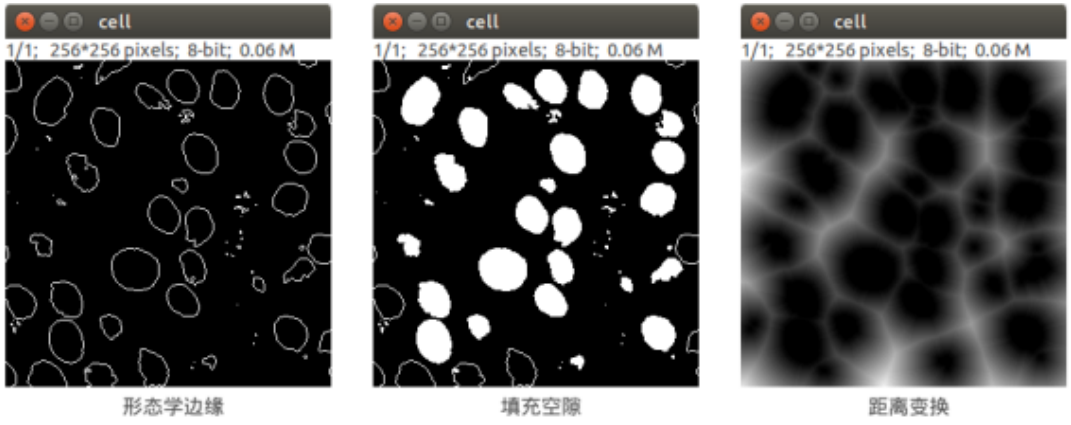

图 28: ImagePy Morphology Advanced

- Process > Binary > Outline: 轮廓计算，本质是进行扩张之后与原图相减

- Process > Binary > Fill Holes: 填充镂空，前景区域上封闭的空洞
- Process > Binary > Distance Transform: 距离变换，计算每一个背景像素到与之最近的前景像素的距离

3.9 几何变换

直方图，数学运算，滤波器，形态学概括来说，新的值都是由原图对应位置或区域的内容决定的，而本节我们讨论的几何变换，是一个位置关系之间的映射。ImagePy 中所有的几何变换都支持图像预览和撤销操作，如果有选区在图像上，则自动只处理选取内的像素，在处理图像栈的时候，会询问是否进行批量处理。此外由于这种变换操作性较强，因而专门设计了两款工具。

图 29: ImagePy Linear Transform

- Image > Transform > Rotate: 对图像进行一定角度的旋转
- Image > Transform > Scale: 对图像进行一定尺度的缩放

3.10 选区运算

ImagePy 选区操作暂时不支持撤销

这里演示几个点选区，然后进行膨胀运算，变成了圆形面选区，再做凸包运算，成为一个凸多边形选区。这里用自由多段线工具绘制 IPY 字样，然后进行膨胀，成为空心，再用 Shift + 矩形工具，叠加一个下划线。

- Selection > All: 选取全部
- Selection > None: 取消选区
- Selection > Inflate: 选区膨胀（Buffer 运算）
- Selection > Shrink: 选区收缩
- Selection > Convex Hull: 凸包运算
- Selection > Bound Box: 最小外接矩形
- Selection > Clip: 去除图像以外的选区
- Selection > Invert: 选区取反

图 30: ImagePy Section Ex1

图 31: ImagePy Section Ex2

将刚才得到的 IPY 添加到选区管理器，然后重新切换到第一幅图，从管理器以 Difference 方式导入 IPY，并填充。

- Selection > Add To Manager: 将选区加入到选区管理器
- Selection > Load From Manager: 从选区管理器加载到当前图像
- Selection > Union: 从选区管理器加载，并叠加到当前选区
- Selection > Differance: 从选取管理器加载，并镂空在当前选区

3.11 标记和像素统计

本节的操作多数与统计相关，结果往往是一张表格，ImagePy 中的表格以独立窗口形式体现，可以另存为 Excel。

- 区域标记 Analysis > Label Image:

  针对二值图像，像素形成联通的区域，所谓区域标记，就是给每个区域标记成一个唯一的颜色。

- 像素统计 Analysis > Histogram:

  对图像中的点进行频率统计并制成表格展示，对于一张经过区域标记的图像来说，其统计结果也意味着每个区域的面积。

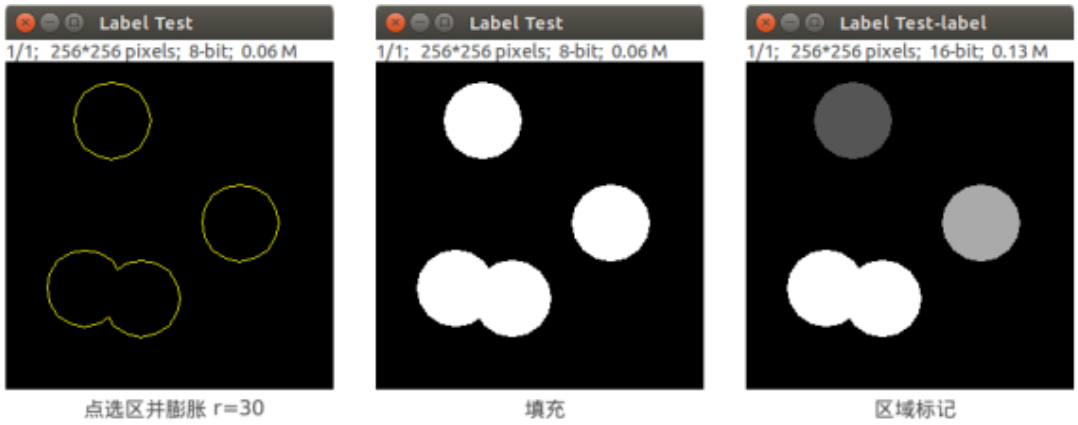

图 32: ImagePy Label Count Ex1

图 33: ImagePy Label Count Ex2

图 34: ImagePy Label Count Ex3

- 碎片分析
  - Analysis > Segment > Statistic: 统计每个碎片内部的像素最大值，最小值，方差等信息
  - Analysis > Segment > Position: 统计每个碎片的质心，外接矩形等位置信息
  - Analysis > Segment > Mark Point: 计算并绘制每个碎片的质心
  - Analysis > Tables > Save As CSV: 将表格另存为 Excel 兼容的 CSV 文本格式

– Analysis > Tables > Save As Tab: 将表格另存为 Excel 兼容的 Tab 制表符格式

3.12 录制宏

宏: 至此我们结合 ImagePy 讨论了图像处理方面的绝大多数基础内容，涵盖了图像的基础概念，直方图，数学运算，滤波器，形态学，几何变换，以及像素统计分析。接下来，我们结合一个具体案例，来看看 ImagePy 能做哪些实际工作，同时介绍宏的概念。

一个细胞计数的例子

细胞技术是医学和生物学研究中经常面对的问题，我们需要计算细胞不同时刻的数量，来估算其分裂速度。当我们在显微镜下获取了细胞的影像。传统的人工计数工作繁琐，且容易出错，本节我们将带大家用 ImagePy 来做细胞计数，这其中用到的一些方法，都是前面讨论过的。

- 图像增强: 图像中细胞比背景暗，但是如果直接进行阈值分割，无法得到纯净的细胞，因此要先对图像做一些预处理。首先用高斯模糊抑制噪声，然后用大尺度 USM 滤波器增强对比。

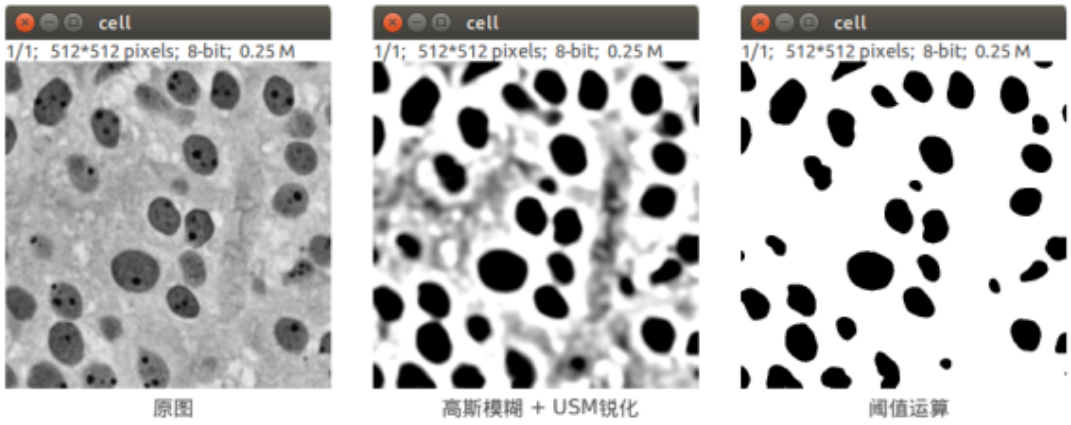

图 35: ImagePy Macro Enhance

- 阈值分割, 标记，统计: 经过预处理，图像可以用阈值分割得到较为理想的二值图像。不过背景是白色，前景是黑色，因此我们要对图像进行一次求反。进而进行区域标记，并统计像素分布。

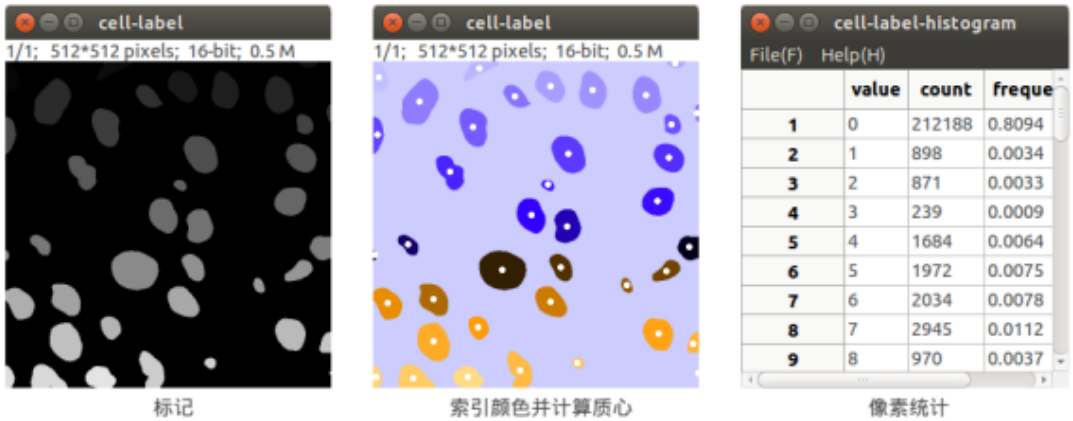

图 36: ImagePy Label and Count

经过阈值分割，每个碎片代表一个细胞，而区域标记后，每个细胞则由唯一颜色构成，此

时像素统计的条目，就是细胞总数（去除背景），而求和，就是细胞的面积。那么问题来了，如果我有多个类似的图片需要处理，是否需要反复进行以上操作呢？这的确很繁琐，好在有宏可以帮助我们。

录制宏: 所谓宏就是 ImagePy 预定义好的功能，我们可以经过简单的操作，重组这些功能，并且在需要的时候依次执行。

- Plugins > Macros > Recoder

  打开宏录制器，我们重新操作细胞计数，我们会发现，每进行一项操作，录制器里面就会多出一行记录，比如我们进行了一个 sigma=5 的高斯模糊，则录制器里出现了 Gaussian > sigma:5.0，不难看出，所谓宏命令，其实就是记录了所执行的命令名称以及其参数，在必要的时候，再次执行而已。执行与保存宏。

  当一切执行完毕，我们得到了如下命令记录。我们可以在录制器中按 F5 执行全部命令，也可以用鼠标选中其中若干行，按 F6 执行选中的行。当然，如果仅仅是这样，我们下次重启时，就由丢失了，所以我们可以保存他，这里我们保存在项目下 Mens > Plugins 目录下，存储为 Cell$_{C}ount.mc$

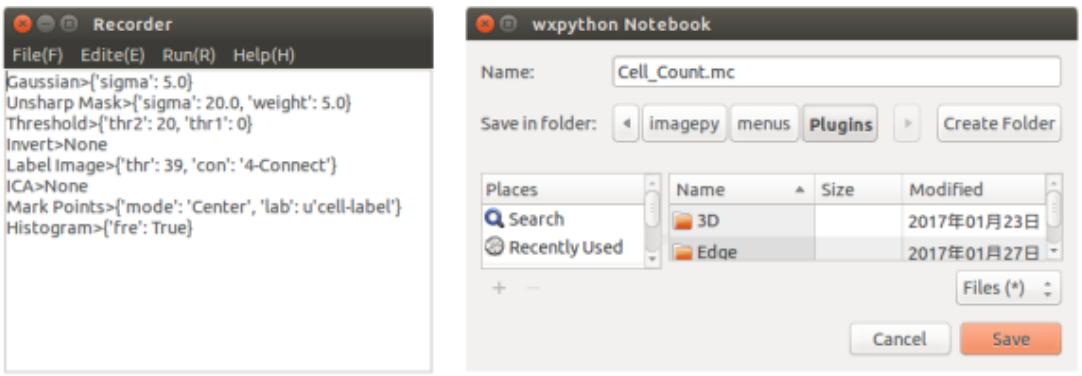

图 37: ImagePy Macros Record and Save

- Plugins > Cell$_{C}ount$

我们重新启动 ImagePy，看看发生了什么。我们刚才保存的宏文件被映射成了一个菜单项！打开细胞图片，运行该菜单项，直接得到了标记和分析结果！是不是很激动。

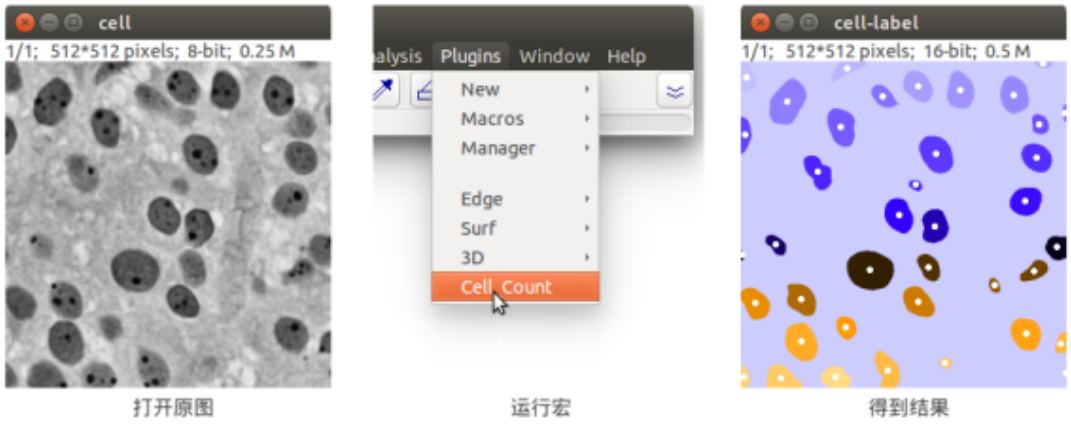

图 38: ImagePy Macros Run

- 能否放到工具栏上?

  我们把 Cell_Count.mc 拷贝到项目下 Tools > Transform 目录下，并准备一个 16x16 的名字为 Cell_Count.mc.gif 的图标，重启，就完成了。看看效果吧！(点击之后效果相同，并且这种方法也可以给任何一个菜单想做快捷按钮)。

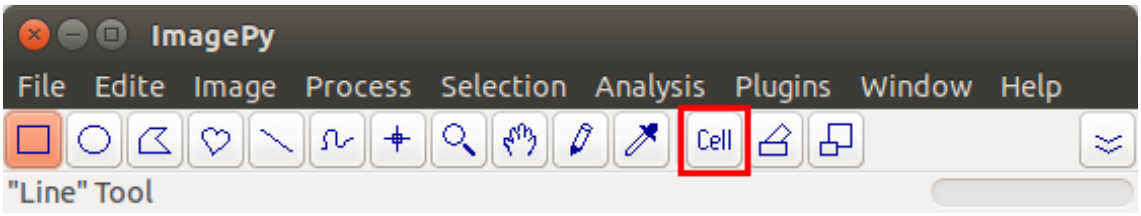

图 39: ImagePy MacrosToolbar

宏为我们提供了一种高效的批处理方式，事实上我们可以把宏放在 Menus 目录下或其任意子目录下，重启时将会映射到对应的菜单项。如果需要调整菜单顺序，可以修改对应文件夹下的 init.py 文件，文件内定义了菜单目录顺序。

3.13 开发 ImagePy 功能拓展

- ImagePy 是什么

  ImagePy 不只是一款图像处理软件，更是一个超轻量级，高扩展性的插件式图像处理框架。框架基于 Python 实现，图像数据基于 Numpy，任何基于 Numpy 的算法库，都可以轻松接入，同时 ImagePy 为其提供有好交互环境，包括对话框，图像预览，撤销支持，选区支持，多通道支持和图像栈支持等…拿来主义可以说是 ImagePy 的设计精髓，scikit-image，opencv 等鼎鼎大名的开源图像处理库，尽为我所用，并且是轻松无缝结合。同样你也可以轻松的扩展一个工具，由于更深入的内容与操作手册的主旨无关，关于扩展开发，我将在开发文档中展开介绍，这里仅仅是一个初步认识，下面就以 scikit-image 的一个 Canny 算子为例，施展吸星大法。

- 为什么选择 Python

  1. 语法简洁，易学，开发环境搭建简单。
  2. 语言灵活性高，编程更方便
  3. 有大量的第三方类库支持，图像方面有 scipy, scikit-image, opencv 等。
  4. 大多数科学计算库都是基于 Numpy 的，这让设计统一框架变得容易。

- 接入 Canny 算子的例子步骤:

  1. 引入核心类库，引入 Filter: Filter 是滤波器积累，核心类库是准备接入的核心函数。
  2. 继承 Filter: 所有滤波器的基类，集成之后自动获得了很多交互功能。
  3. 设定 Title: 这将是菜单栏上展示的内容。
  4. 设定 Note: 这告诉 ImagePy 需要帮你做哪些前期和后期工作，比如对选区的支持，对图像栈的支持等。。
  5. 设定 Para：核心函数需要用到的参数。
  6. 设定 view：插件执行时的交互方式（与参数对应，只需要指明参数类型，取值范围等信息即可）。

7. 核心函数，run(): 这一步仅仅是调用核心函数，snap 是缓存图像，img 是前景图，para 是通过对话框交互得到的参数。你要做的仅仅是把 snap 的值经过作用，赋予 img。

代码：

```
    # -*- coding: utf-8 -*
from skimage import feature
from core.engines import Filter
class Plugin(Filter):
    title = 'Canny'
    note = ['all', 'auto_msk', 'auto_snap', 'preview']

    para = {'sigma':1.0, 'low_threshold':10, 'high_threshold':20}

    #parameter
    view = [(float, (0,10), 1,  'sigma', 'sigma', 'pix'),
            ('slide',(0,30), 'low_threshold', 'low_threshold',''),
            ('slide',(0,30), 'high_threshold', 'high_threshold','')]
    #process
    def run(self, ips, snap, img, para = None):
        return feature.canny(snap, sigma=para['sigma'],
                     low_threshold=para['low_threshold'],
                     high_threshold=para['high_threshold'],
                     mask=ips.get_msk())*255
```

效果图：

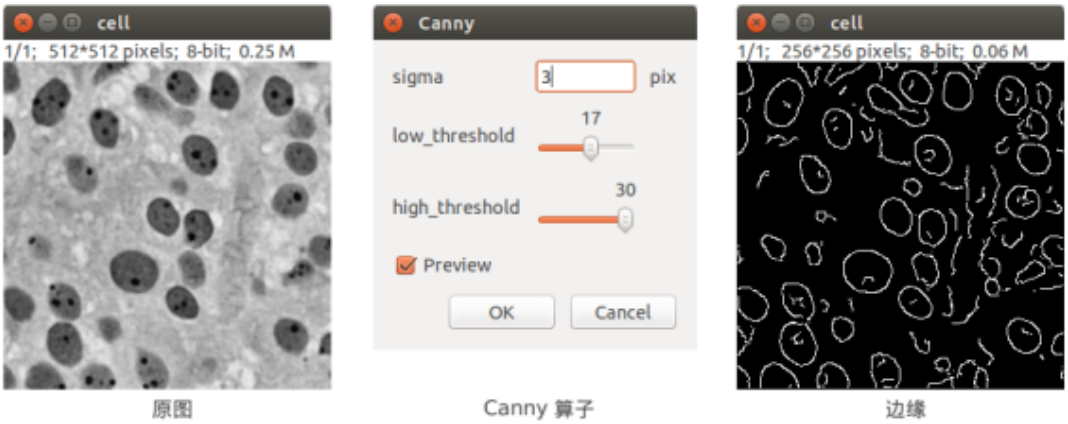

图 40: ImagePy Plugin