Escrevendo scripts no GIMP

Pintando com números

Muitos usuários se voltam para o GIMP quando precisam editar uma imagem. Nem todos, entretanto, sabem que é possível automatizar tarefas repetitivas por meio de scripts. Em adição ao dialeto Lisp nativo do GIMP, ainda temos uma alternativa bastante popular: a linguagem Python.

POR OLIVER FROMMEL

O GIMP – *GNU Image Manipulation Program* ou *Programa GNU para Manipulação de Imagens* – possui um belo cardápio de funções bastante úteis. Entretanto, navegar pela complicada estrutura de menus pode ser uma experiência desesperadora se for preciso aplicar a mesma seqüência de ferramentas a um grande número de imagens. Felizmente, o GIMP possui uma interface de programação integrada que permite aos usuários fazer scripts para facilitar o trabalho e automatizar tarefas. Mas vá tirando esse sorriso do rosto: a linguagem Scheme [1] não é o que o povo em geral consideraria uma diversão prazerosa; sua sintaxe recheada de parênteses é um verdadeiro pesadelo.

Não é surpresa alguma que os fãs de outras linguagens de programação tenham proposto e implementado alternativas. Há, por exemplo, uma interface para a linguagem Perl chamada de *GIMP-Perl*. Essa interface está bastante desenvolvida e possui um modo servidor, com o qual é possível interpretar scripts e modificar imagens sem que a interface gráfica do GIMP tenha que ser chamada.

O onipresente Python

Por algum motivo que nos escapa, o modo Perl não é incluído por padrão no GIMP presente na maioria das distribuições. Por outro lado, o modo Python já garantiu seu lugar como alternativa viável ao Scheme. O *GIMP-Python* está disponível em praticamente todas as distribuições Linux. Os usuários do SUSE, entretanto, estão sem sorte desta vez: por padrão não há Pyhton no GIMP instalado em seus sistemas.

Os usuários do Fedora Core só precisam sentar e relaxar; no Debian os usuários precisarão instalar o pacote `gimp-python`, que já vem instalado no Ubuntu.

Compilando o GIMP

O GIMP é um dos softwares mais comuns que existem. Mesmo assim, alguns ainda preferem compilá-lo a partir do código fonte. Se for esse o seu caso, procure pela versão mais atual do GIMP (atualmente, a 2.2) no site oficial [2]. Depois de descompactar o arquivo, rode o script de configuração com as opções de Python ativadas: `./configure --enable-python`. Para evitar conflitos, remova qualquer versão existente do GIMP antes de rodar o `make install`, que instala o GIMP em seu sistema.

Banco de dados de plugins

Os recursos necessários para lidar com plugins em Python estão no menu `Xtns | Python-Fu` (`Extras | Python-Fu`). O GIMP mostra no terminal de onde foi chamado qualquer erro que tenha ocorrido quando foi iniciado. Se você iniciar o programa por um menu ou ícone no seu desktop, será preciso abrir a janela de registro de erros em `File | Dialogs | Error console` (`Arquivo | Diálogos | Console de Erros`) para vê-los. Observe que o console não mostra todos os erros que possam ter ocorrido. Os programadores de scripts do GIMP provavelmente preferirão iniciar o programa numa janela de terminal para saber de toda a "fofoca".

A ferramenta mais importante para os camaradas programadores de scripts GIMP é o navegador para banco de dados de procedimentos – *Procedural Data Base* ou PDB (**figura 1**). O navegador PDB está localizado no menu principal em `Xtns | Python-Fu | PDB Browser` (`Extras | Python-Fu | PDB Browser`). Ele lista as funções disponíveis para programadores do GIMP. Por exemplo, `file_jpg_load`

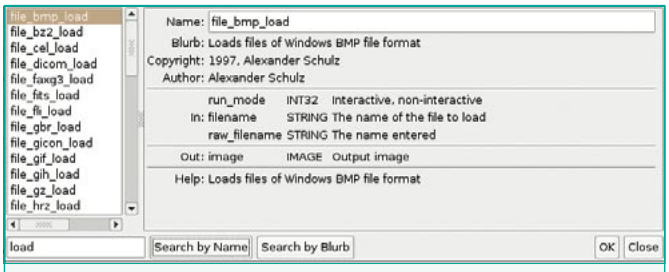

Figura 1: O navegador integrado de scripts do GIMP com o campo de busca no canto inferior esquerdo.

carrega uma imagem no programa. Pode-se rodar métodos como esse diretamente no console Python (`Xtns | Python-Fu | Console` ou `Extras | Python-Fu | Console`).

O navegador é a maior fonte de informações sobre as funções que sua versão de GIMP oferece. Muitos tutoriais e HOWTOs na Internet estão desatualizados, portanto o navegador é seu amigo. Se você procurasse por `layer`, por exemplo, o navegador mostraria as funções que contêm essa palavra. Você pode clicar em uma função no campo à esquerda para ver sua descrição à direita. Obviamente, os valores de entrada (`In`) e de retorno (`Out`) são tão importantes quanto as funções.

Registrando plugins

A estrutura básica de um plugin em Python é bastante simples: uma chamada ao método `register` para apresentar o GIMP ao novo plugin e uma ou mais fuções que completam o serviço. Nosso primeiro exemplo será um plugin simples que faz pipocar uma mensagem na tela. Procurando com o navegador PDB encontramos a função `gimp_message`, que possui apenas um parâmetro: o texto que queremos mostrar.

O método `register` precisa de não menos que onze parâmetros, nessa ordem: nome do plugin, descrição, texto de ajuda, autor, dados de copyright, data, caminho no menu, formatos de imagem permitidos, parâmetros do plugin, *buffer* de memória para o valor de retorno e função de trabalho. Depois de registrada, a primeira chamada deve ser para `main()`.

Nosso exemplo usa um plugin chamado `python_fu_simple`. Esse é o nome que nós (e outros plugins) vamos usar para chamar o plugin depois do registro. Os próximos quatro parâmetros são fáceis de preencher, pois são apenas blocos de texto com finalidade de informação. O caminho para o menu é mais importante, pois há dois deles: o menu principal do GIMP e o menu de contexto (o do botão direito do mouse) da imagem. O primeiro item do caminho para o menu principal deve ser `<Toolbox>`, e para o menu de contexto usa-se `<Image>`. O prefixo é seguido do caminho, separado por barras (/), para que o plugin tome seu lugar no menu apropriado: `<Image>/Python/Simple`.

O próximo parâmetro especifica os tipos de imagem (não os formatos) com que o script pode trabalhar, ou seja, imagens coloridas com (`RGBA`) ou sem o canal alfa (`RGB`) , ou imagens em tons de cinza (`GRAY`). Se o plugin nao se importar com isso, podemos usar um curinga (`*`). Finalmente, temos os parâmetros de entrada e saída e o nome da função de trabalho. A **listagem 1** mostra um exemplo simples mas funcional.

A função do plugin é chamada de `python_fu_simple` (linha 9) e a função de trabalho é `python_simple`. A função de trabalho sempre tem os parâmetros `img` e `drawable` (linha 5), que o GIMP automaticamente passa à função – a lista de parâmetros de entrada na linha 17 está vazia. A linha 3 mostra como carregar o módulo GIMP no Python.

Para continuar seguido nosso tutorial, salve a listagem como `simple.py` no diretório `~/.gimp/plug-ins`. Possivelmente o diretório do GIMP em seu `/home` pode estar com um nome diferente, como `.gimp-2.0` ou coisa parecida. Torne o script executável (`chmod +x`) e inicie o GIMP. Agora, crie uma nova imagem (`File | New`). Deve haver um item `Python | Simple` no menu de contexto.

Caixas de diálogo no GIMP

Obviamente, esse exemplo mínimo não faz nada de realmente útil. Vamos, então, passar a um exemplo menos trivial para demostrar alguns aspectos úteis do GIMP Python. O plugin adicionará agora a chamada *vinheta* (*vignette*) à imagem, ou seja, uma sombra que esconde as partes não essenciais da imagem e realça as mais importantes. O exemplo demonstra os parâmetros do plugin e as funções do GIMP que usam filtros selecionados.

A idéia básica é bastante simples. O script usa uma função para selecionar uma elipse do mesmo tamanho da imagem, depois inverte a seleção e a preenche com a ferramenta *preencher* (na caixa de ferramentas é o botão com o balde de tinta). Valores apropriados de transparência e gradientes fazem a borda entre a vinheta e a imagem ficar cuidadosamente difusas.

O registro do plugin é similar ao do nosso exemplo anterior, mas desta vez precisamos de novos parâmetros de entrada. O GIMP gerará automaticamente uma caixa de diálogo para eles. Para cada parâmetro dentro de chaves (`[]`) teremos quatro valores, encerrados entre parênteses. O primeiro parâmetro especifica o tipo, o segundo é uma variável de ambiente. Isso será usado pelo script, por exemplo, na função de trabalho. A seguir, temos uma descrição dos parâmetros e, finalmente, um valor padrão. A seção com os valores de entrada se parece com a mostrada na página ao lado:

Listagem 1: Um script simples para o GIMP

```
01 #!/usr/bin/python
02 from gimpfu import *
03
04 def python_simple(img, drawable):
05    gimp.message("Exemplo Simples")
06
07 register(
08    "python_fu_simple",
09    "Mostra uma caixa de diálogo.",
10     "Ajuda: Mostra uma caixa de diálogo simples.",
11    "Oliver Frommel",
12    "LinuxMagazine",
13    "2005",
14    "<Image>/Python/Simple",
15    "*",
16    [],
17    [],
18    python_simple)
19 main()
```

Figura 2: Uma imagem qualquer, antes e depois de passar pelo script de *vinheta*.

```
[
   (PF_INT, “feather_in”, §§
        “FeatheringRand”, 100),
   (PF_INT, “opacity_in”, §§
      “Opacity”, 50)
],
```

`PF_INT` refere-se a um inteiro. Os tipos de parâmetro são descritos com exemplos na documentação do GIMP Python em [3]. Infelizmente a documentação está levemente desatualizada – o plugin mostrado lá sequer roda na série 2.x do GIMP. Você talvez queira conferir o website do GIMP Perl em [4], que está, ao menos, atualizado. Obviamente, você terá que “quebrar a cabeça” para migrar as técnicas de Perl para Python mas, pelo lado positivo, os parâmetros são os mesmos, não importando a linguagem de script a ser usada.

O PDB e suas funções

A função de trabalho ainda não recebeu esse título – precisamos fazer algo quanto a isso. Uma procura no PDB por `select` revela um certo número de possíveis candidatos. O que que-

remos chama-se `gimp_ellipse_select`. O primeiro parâmetro é a imagem em si – que tem, obrigatoriamente, que estar em primeiro. Os quatro seguintes dizem o tamanho da seleção em valores de X e Y, a largura e a altura. O parâmetro `option` especifica se vamos adicionar a seleção (`CHANNEL_ADD_OP`) ou substituir uma existente (`CHANNEL_REPLACE_OP`). O navegador PDB usa nomes ligeiramente diferentes, da mesma forma que com outros tipos de parâmetros: por exemplo, `GIMP_CHANNEL_ADD_OP`. Nos scripts em Python, deixe de fora o prefixo `GIMP_`.

Ativando a opção `antialias` (valor `TRUE`), fazemos o GIMP suavizar a seleção. `feather` funciona de maneira similar para providenciar uma transição suave. O último parâmetro, `feather_radius`, especifica a largura da área de transição. O script não possui um valor fixo neste ponto, mas toma seu valor da variável de entrada definida pelo usuário chamada de `feather_in` (linha 4 na **listagem 2**).

Todos os métodos que usamos até agora pertencem ao objeto `pdb`, ou seja, são chamados por um `pdb.method()`. Isso dificilmente pode ser chamado de programação orientada a objetos, pois a POO é caracterizada por uma relação temática entre os métodos e os objetos. Entretanto, o GIMP Python permite a declaração de estruturas orientadas a objetos. Por exemplo, `img.add_layer(...)` funcionará no lugar de `pdb.gimp_add_layer(img, ...)`. Essa é uma chamada ao método `gimp_add_layer()` do objeto `img`, mas deixamos de lado a declaração explícita de `img`. O navegador PDB não documenta esse estilo de programação, você deve descobrir por si mesmo.

O método `gimp_selection_invert` inverte a seleção atual. Finalmente, o `gimp_edit_bucket_fill` preenche a área selecionada com a cor principal, especificada pelo segundo parâmetro de `FG_BUCKET_FILL`. Se usarmos `PATTERN_BUCKET_FILL`, o baldinho vai preencher a área selecionada com um padrão.

O próximo parâmetro especifica como aplicar a nova camada de cor ao fundo. O valor `NORMAL_MODE` simplesmente substitui a camada de cor existente. O valor `MULTIPLY_MODE` usado em nosso exemplo diz ao GIMP para misturar as cores existentes com as novas. O menu `Camadas` (`Layer`) fornece uma lista dos modos disponíveis e o navegador PDB informa as palavras-chave apropriadas (novamente, remova o prefixo `GIMP_`). Você pode querer experimentar com o `DISSOLVE_MODE`.

Desligando a função *desfazer*

O valor de entrada para o próximo parâmetro, `opacity_in`, especifica a opacidade do padrão de preenchimento. Um número grande cria uma borda escura, um número menor cria uma mais clara. Para completar as coisas, o script desfaz a seleção usando o método `gimp_selection_none`. Neste ponto, o usuário poderia desfazer todas as etapas de nosso script usando o botão de `Desfazer` (`Undo`) do GIMP, como se estivesse em modo interativo. Para evitar isso, desabilitamos a função desfazer chamando a função `gimp_image_undo_disable`.

Se rodarmos o script em uma imagem existente, o efeito será semelhante ao da **figura 2**. Nós exageramos o efeito na imagem para ficar mais evidente como exemplo, mas menos opacidade teria sido, em aplicações práticas, uma idéia melhor.

Dificuldades na depuração

Automatizar o GIMP com scripts em Python deveria ser uma coisa bastante simples. Infelizmente, a documentação vergonhosamente obsoleta é um obstáculo. Além disso, o ciclo de desenvolvimento torna tudo muito mais difícil. A cada vez que o script é alterado, você precisa sair do GIMP e iniciá-lo novamente. A possibilidade de carregar novos scripts com o GIMP em funcionamento seria algo maravilhoso.

A documentação precisa de atualização urgente. Na época em que escrevemos esta matéria, os programadores de Python no GIMP não tinham alternativa senão procurar pela Internet por várias fontes de informação (como as do site GIMP-Perl) e aplicar essas fontes em seu trabalho. Se isso não o incomodar, o Python dá aos programadores do GIMP uma arca cheinha de tesouros para sua satisfação. ■

GimpShop

Embora o *Gimp* tenha muitos dos recursos encontrados no *Photoshop*, usuários que migram do software da Adobe comumente reclamam que muitas das funções que costumam usar estão em lugares diferentes nos menus, tem outros nomes ou teclas de atalho. Isso "quebra" seu ritmo de trabalho, e impede que sejam tão produtivos quanto na solução proprietária, pelo menos até se adaptarem ao novo ambiente.

Scott Moschella decidiu resolver esse problema e criou o *GimpShop*: uma versão do *Gimp* modificada para se tornar mais parecida com o *Photoshop*. Os itens nos menus foram reordenados para ficarem na mesma ordem do software da Adobe, algumas ferramentas mudaram de nome e até os atalhos de teclado são os mesmos. Embora ainda seja "experimental", o software já está disponível para download em versões para Mac e Linux em:

http://plasticbugs.com/index.php?p=241

Informações

[1] Scheme: http://www.teach-scheme.org/Notes/scheme-faq.html

[2] GIMP: http://www.gimp.org

[3] Documentação sobre Python no GIMP: http://www.gimp.org/docs/python/structure-of-plugin.html

[4] Documentação sobre Perl no GIMP: http://imagic.weizmann.ac.il/~dov/gimp/perl-tut-2.0

[5] OGimp: http://ogimp.tk

[6] Gimp BR: http://www.gimp.com.br

Listagem 2: Função de trabalho

```
01 def python_vinheta(img, drawable, feather_in, opacity_in):
02     width = drawable.width
03     height = drawable.height
04     pdb.gimp_ellipse_select(img, 0, 0, width, height, CHANNEL_OP_REPLACE, TRUE, TRUE, feather_in)
05     pdb.gimp_selection_invert(img)
06     pdb.gimp_edit_bucket_fill(drawable, FG_BUCKET_FILL, MULTIPLY_MODE, opacity_in, 0, 0, 0, 0)
07     pdb.gimp_selection_none(img)
```