Licensed under LGPL started in 2000, merged with ggg and resurrected in 2005
Contributors in order of appearance:
Calvin Williamson
Jean-Claude Wippler
Manish Singh
Jay Cox
Daniel Rogers
Sven Neumann
Michael Natterer
Øyvind Kolås
Philip Lafleur
Dominik Ernst
Richard Kralovic
Marco Ciampa
Victor Bogado
Martin Nordholts
Geert Jordaens
Garry R. Osgood
Thomas Manni
Michael Schumacher
Gabriel Burt
John Marshall

What is GEGL?

- Raster processing engine
  - Graph based
  - Plug-ins
  - HDR
  - Large images

The state of GEGL

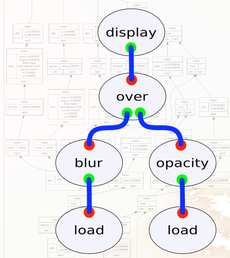
FOSDEM 2007

Øyvind Kolås, pippin@gimp.org

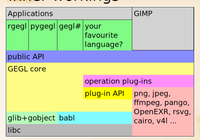
Outline

- What is GEGL
- The Public API
- Inner workings
- Future work

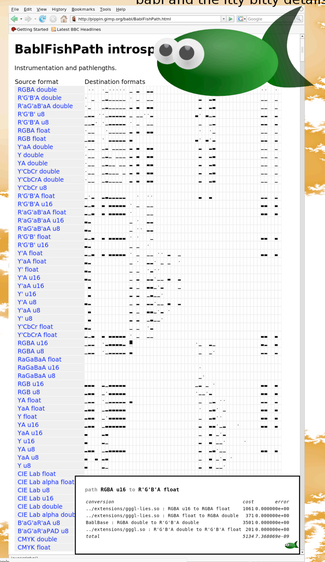
The GEGL API



Inner workings



babl and the itty bitty details



Future work:

http://www.gegl.org/

Core
- parallel and|or distributed processing
- More sophisticated caching
- GPU acceleration?
- Intermediate language and JIT to cpu/gpu/mmx etc?
- babl (pixelformats)
  - handle more than 3color + 1alpha components
- Abstractions on top of GEGL? (animated properties?)

Operations
- libopenraw CFA importer
- Advanced Bayer demosaicing
- noise reduction (graycstoration)
- HDR tonemapping
- Frequency Domain Processing
- Full SVG 1.1 Filter set
- Brushes for a paint core

Compositing

Non destructive editing

High Dynamic Range

Ruby

```
#!/usr/bin/env ruby
require 'gegl'

gegl = Gegl::Node.new

load1   = gegl_node_new_child :load,:path=>"images/romedalen.png"
load2   = gegl_node_new_child :load, :path=>"images/gegl.png"
blur    = gegl_node_new_child :gaussian_blur, :std_dev_x=>10.0, :std_dev_y=>10.0
opacity = gegl_node_new_child :opacity, :value=>0.5
over    = gegl_node_new_child :over
display = gegl_node_new_child :display

load1   >> blur >> over >> display
load2   >> opacity >> over[:aux]

display.process
sleep 10
```

Iterative chunkwise ROI procesing

GEGL uses information provided by the nodes to determine the smallest buffers needed at each stage of processing.

C

```
#include <gegl.h>

int main (int    argc, gchar **argv)
{
  GeglNode *gegl, *load1, *load2, *blur, *opacity, *over, *display;

  gegl_init (&argc, &argv);

  gegl = gegl_node_new ();

  load1   = gegl_node_new_child (gegl, "operation", "load",
                                       "path",      "images/romedalen.png",
                                       NULL);
  load2   = gegl_node_new_child (gegl, "operation", "load",
                                       "path",      "images/gegl.png",
                                       NULL);
  blur    = gegl_node_new_child (gegl, "operation", "gaussian-blur",
                                       "std-dev-x", 10.0,
                                       "std-dev-y", 10.0,
                                       NULL);

  opacity = gegl_node_new_child (gegl, "operation", "opacity",
                                       "value",     0.5,
                                       NULL);
  over    = gegl_node_new_child (gegl, "operation", "over", NULL);
  display = gegl_node_new_child (gegl, "operation", "display", NULL);

  gegl_node_link_many (load1, blur, over, display, NULL);
  gegl_node_link (load2, opacity);
  gegl_node_connect_to (opacity, "output", over, "aux");

  gegl_node_process (display);
  sleep (10);

  g_object_unref (gegl);

  gegl_exit ();
  return 0;
}
"c.c" 38L, 1333C written                                  38,1          All
```

Operations

brightness/contrast plug-in

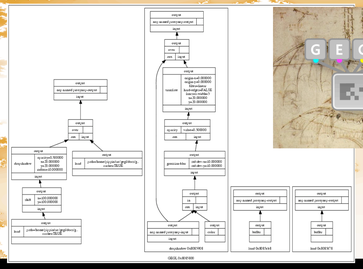
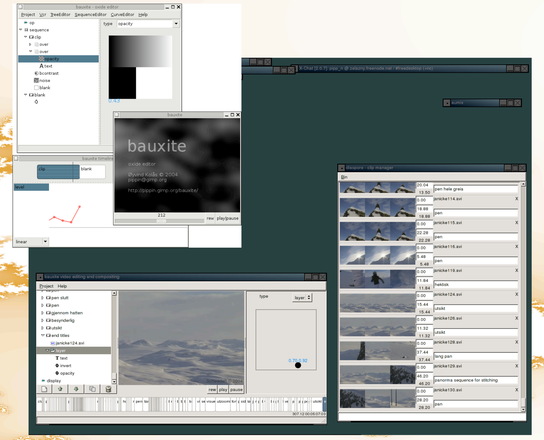
```
#!/ GEGL_CHANT_PROPERTIES
gegl_chant_double (contrast,   -5.0, 5.0, 2.0, "Range scale factor")
gegl_chant_double (brightness, -3.0, 3.0, 0.0, "Amount to increase brightness")
gegl
#define GEGL_CHANT_NAME          brightness_contrast
#define GEGL_CHANT_TITLE         "Brightness-contrast"
#define GEGL_CHANT_SELF_FILTER
#define GEGL_CHANT_DESCRIPTION   "Changes the light level and contrast."
#define GEGL_CHANT_CATEGORIES    "color"
#include "gegl-chant.h"

static gboolean
process (GeglOperation *operation,
         void          *in_buf,
         void          *out_buf,
         glong         n_pixels)
{
  GeglChantOperation *self = GEGL_CHANT_OPERATION (operation);
  gfloat             *pixel_in = in_buf;
  gfloat             *pixel_out = out_buf;
  gint               i;

  for (i=0; i<n_pixels; i++)
    {
      gint c;
      for (c=0; c<3; c++)
        pixel_out[c] = pixel_in[c]*(0.3) * self->contrast + self->brightness * (1.
        pixel_out[c]=pixel_in[c];
        pixel_in += 4;
        pixel_out += 4;
    }
  return TRUE;
}
#endif
                                                                          34,1          All
```

XML

```
#!/ gegl
<gegl>
    <over>
        <opacity value='0.5'/>
        <load path='images/gegl.png'/>
    </over>
    <gaussian-blur std-dev-x='10.0' std-dev-y='10.0'/>
    <load path='images/romedalen.png'/>
</gegl>
~
                                                                          1,1           All
```

Hit detection

Features:
8bit, 16bit integer and 32bit floating point, RGB, CIE Lab, and Y'CbCr output.
Non destructive editing
C, Ruby and Python interfaces.
Extendable through plug-ins.
XML model
Iterative processing.
Efficient subregion evaluation.
Supports processing and display of images larger than RAM
Bounding box based hit detection.
Rich core set of processing operations that internally computes with HDR capable 32bit floating point.

```
#!/usr/bin/env ruby

copyright =
/* !!!! AUTOGENERATED FILE generated by math.rb !!!!!
 *
 *  Copyright 2006 Øyvind Kolås <pippin@gimp.org>
 *
 * !!!! AUTOGENERATED FILE !!!!!
 */

n = [

    ['add',         'c = c + value'],
    ['subtract',    'c = c - value'],
    ['multiply',    'c = c * value'],
    ['divide',      'c = value==0.0?0.0:c/value'],
    ['gamma',       'c = pow (c, value)'],
#   ['threshold',   'c = c>=value?1.0:0.0'],
#   ['invert',      'c = 1.0-c']
    ]

a.each do
  |item|

    name     = item[0] + ''
    filename = name + '.c'

    puts 'generating #{filename}'
    file = File.open(filename, 'w')

    name     = item[0]
    capitalized = name.capitalize
    swapcased   = name.swapcase
    formula     = item[1]

    file.write copyright
    file.write "
#if GEGL_CHANT_PROPERTIES
gegl_chant_double (value, _G_MAXDOUBLE, G_MAXDOUBLE, 0.0, \"global value used if a
ox doesn't contain data\")
#else

#define GEGL_CHANT_POINT_COMPOSER
#define GEGL_CHANT_NAME          #{name}
#define GEGL_CHANT_DESCRIPTION   \"Math operation #{name} (#{formula})\"
#define GEGL_CHANT_SELF          \"#{filename}\"
#define GEGL_CHANT_CATEGORIES    \"compositors:math\"
#define GEGL_CHANT_PREPARE_INIT
#include \"gegl-chant.h\"

#include <math.h>

static void init (GeglChantOperation *self)
{
  GEGL_OPERATION_POINT_COMPOSER (self)->format = babl_format (\"RGBA float\");
  GEGL_OPERATION_POINT_COMPOSER (self)->aux_format = babl_format (\"RGB float\");
}

static gboolean
process (GeglOperation *op,
         void          *in_buf,
         void          *aux_buf,
         void          *out_buf,
         glong         n_pixels)
{
  gint i;
  gfloat *in = in_buf;
  gfloat *out = out_buf;
  gfloat *aux = aux_buf;

  if (aux == NULL)
    {
        gfloat value = GEGL_CHANT_OPERATION (op)->value;
        for (i=0; i<n_pixels; i++)
          {
            int _i;
            gfloat c;
            for (_i=0; _i<3; _i++)
              {
                c=in[_i];
                #{formula};
                out[_i]=c;
              }
            out[3]=in[3];
            in += 4;
            out+= 4;
          }
    }
  else
    {
        for (i=0; i<n_pixels; i++)
          {
            int _i;
            gfloat c;
            gfloat value;
            for (_i=0; _i<3; _i++)
              {
                c=in[_i];
                value=aux[_i];
                #{formula};
                out[_i]=c;
              }
            out[3]=in[3];
            in += 4;
            aux += 3;
            out+= 4;
          }
    }
  return TRUE;
}
#endif

  file.close
end
                                                                          114,1         Bot
```

Meta operation