Gimp - Werkstatt - Newsletter
Let's read about Gimp

Inhalt:

Neues aus der Gimp-Werkstatt	von Eleanora	1
Kleine Welt	von rainerstollwetter	4
RGB Farbräume	von terrikay	8
Scripten mit Python	von Hans	10

von Eleanora

Neues aus der Werkstatt

Sommer in der GIMP-Werkstatt

Aktivitäten im Sommer

Trotz des schönen Wetters gab es auch in diesem Sommer in der Werkstatt eine Menge zu lesen und zu lernen.

Einige Teilnehmer haben das schöne Wetter zum Anlass genommen kleine Treffen zu verabreden. „Beschnuppert" haben sich Hans und Avarra bei Regen im Tierpark Eeholt im Norden Deutschlands.

Im Westen trafen sich Rainer, Yellow und Sterni zum fotografieren im Affen und Vogelpark Eckenhagen.

Urlaub im Westen machten terrikay und Dirk auf der schönen schottischen Insel Arran, wo sie unseren Arran trafen.

Viele wunderschöne Fotos von Landschaft und Tierwelt sind bei diesen Treffen entstanden, die im Forum veröffentlicht wurden.

Aber auch Urlaubsgeschichten aus dem sonnigen Süden gibt es, die nicht ganz so positiv verlaufen sind.

Wettbewerbe im Sommer

Der Sommer in der Gimp-Werkstatt stand wieder ganz im Zeichen vieler Wettbewerbe. Und im Juni war das Wetter so schön, dass wir uns alle nur noch Abkühlung wünschten. Secretwz gewann im Juni den Werkstattbild-Wettbewerb zum Thema „In den Tiefen der Meere". Den Foto-Wettbewerb gewann Hans mit seinem Bild zum Thema „Fließendes". Den U-100 Wettbewerb gewann im Juni Selma mit ihrer Interpretation zu Thema „Wasser".

Im Juli waren die Wettbewerbe nicht weniger spannend. Den Fotowettbewerb gewann Fruchtzwergbiene mit ihren „sommerlichen" Johannisbeeren. Zum Thema „Sommerliches". Levan gewann den Werkstattbildwettbewerb mit seiner „Inneren Ruhe" und hal9000 überzeugte im U-100 Wettbewerb mit seiner Sicht des „Weltalls".

Im August war terrikay mit Ihrer Kamera ganz „Nah dran" und gewann damit dein Foto-Wettbewerb. Selma begab sich auf Safari in den „Tropischen Regenwald" für unseren Werkstattbild-Wettbewerb und heipo traute sich in die „Unterwelt" um ein Bild für den U-100 Wettbewerb zu basteln.

Auch einen Animationswettbewerb gibt es in der Gimp-Werkstatt, nur leider kann man diese Bilder nicht ausdrucken, weshalb sie hier leider nur erwähnt werden. Den Animationswettbewerb zum Thema „Tiere in Bewegung" gewann CD mit ihrer Kuh, die schreiben konnte. Den Wettbewerb zum Thema „Zauberei" gewann Selma mit ihrem Dschinn der aus der Flasche kam und seinen Meister in ein Schweinchen verwandelte. Zur Zeit ist können Animationen zum Thema „in der Schule" gebastelt werden. Der Wettbewerb läuft noch bis zum 18.10.2009

Weitere Wettbewerbe

Neben unseren klassischen Wettbewerben haben sich unsere Mitglieder noch ganz viele andere Wettbewerbe ausgedacht.

Für die Fotografen unter uns findet regelmäßig ein Themenwettbewerb statt. Der Wettbewerb läuft über 7 Tage oder bis 7 Fotos eingereicht wurden. Für die Teilnehmer gibt es aber keine klassische Abstimmung, sondern eine detaillierte Beurteilung des Fotos durch den Themenersteller. Der Gewinner nennt das neue Thema und startet den Folgewettbewerb. Es kommen immer sehr schöne Fotos zustande.

Für Tutorienbastler/innen gibt es den Tut-Wettbewerb. Es finden sich immer zwei Gegner zusammen, die über ein vorgegebenes Tutorial ein Bild erstellen. Nachdem beide Gegner ihre Umsetzungen veröffentlicht haben, gibt es eine demokratische Abstimmung, an der jedes Forenmitglied teilnehmen kann.

Ein weiterer Wettbewerb hat sich etabliert, in dem Mitgliedsausweise erstellt werden. In diesem Monat ist es ein Ausweis für die Mitgliedschaft im Grundlagenkurs. Einen Mitgliedsausweis für die Mitgliedschaft in der Gimp-Werkstatt kann bereits bei eibauoma beantragt werden.

Mit dem Ausweis kannst du deine Homepage schmücken oder als bekennender Gimper den Ausweis in deine Signatur einbinden.

Im September finden auch wieder Wettbewerbe statt und ich bin schon ganz gespannt, welche Bilder bis Ende September noch eingereicht werden.

Im Fotowettbewerb solltet ihr euch auf den Weg machen um eine „Endstation" zu finden.

Im Werkstattbild-Wettbewerb könnt ihr diesen Monat Werbung machen für euch oder ein Fantasie-Produkt.

Der U-100 Wettbewerb wurde nach langer Diskussion zum U-200 Wettbewerb umgetauft. Das heißt alle Mitglieder, die bei Veröffentlichung ihres Bildes noch keine 200 Beiträge geschrieben haben, können an diesem Wettbewerb teilnehmen.

Ich wünsche allen Teilnehmern und allen Lesern viel Spaß bei unseren Wettbewerben.

Eleanora

von terrikay

Farbmanagement

RGB Farbräume

Wer eine Spiegelreflexkamera oder hochwertige Bridgekamera hat, dem ist sicherlich schon aufgefallen, dass man einstellen kann, ob die jpg-Bilder im sRGB oder im AdobeRGB-Farbraum gespeichert werden sollen.

Aber was bedeuten diese Farbräume eigentlich und wer sollte welchen verwenden? Diesen Fragen will ich hier einmal nachgehen, da teilweise abenteuerliche Erklärungen existieren.

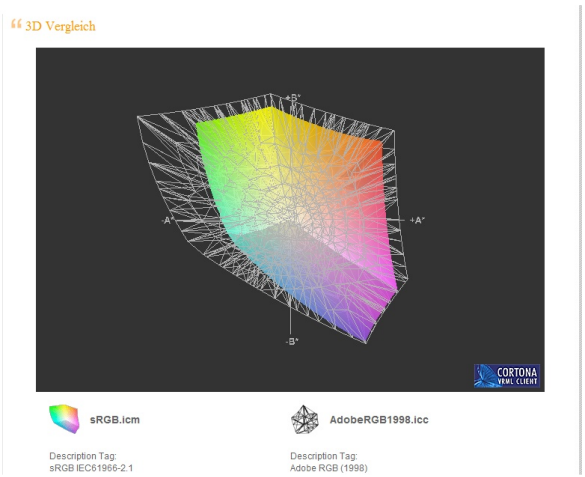
RGB ist zunächst einmal die Bezeichnung eines Farbmodells, dass heißt alle Farben werden aus rot, grün und blau zusammengemischt. Nach diesem Farbmodell arbeiten alle Monitore, insofern ist es sinnvoll, dass auch die Bilddateien, die am Monitor dargestellt werden sollen, nach diesem Farbmodell beschrieben werden.

Bei jpgs sind das pro Farbkanal maximal 256 Abstufungen. Jeder der drei Farbkanäle kann Werte zwischen 0 und 255 einnehmen - das macht insgesamt 2563 - also ca. 16 Millionen mögliche Farbabstufungen.

Trotzdem ist damit noch nicht definiert, welcher Farbraum dargestellt werden kann - wo also die Grenzen der definierten Farben liegen.

Der größtmögliche Farbraum wären alle vom Menschen wahrnehmbare Farben - das ist das gesamte Spektrum zwischen Infrarot und Ultraviolett.

Allerdings ist kein Monitor in der Lage, alle diese Farben anzuzeigen. Also hat man sich 1996 geeinigt, einen Farbraum zu definieren, der von guten Monitoren der damaligen Zeit auch angezeigt werden konnte - nämlich StandardRGB oder auch einfach sRGB.



Das ist seither der übiche RGB-Farbraum, auch im Web sollte dieser Farbraum definitionsgemäß benutzt werden.

Früher (bis vor wenigen Jahren) waren

die meisten Monitore auch nur in der Lage, so einigermaßen den sRGB-Farbraum anzuzeigen.

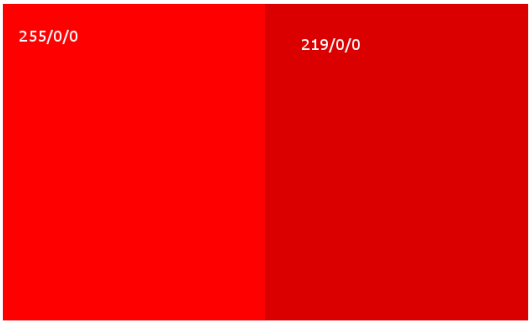
1998 tanzte die Firma Adobe ein wenig aus der Reihe. Sie definierte einen größeren Farbraum, insbesondere, um mehr Grün-, Gelb- und Rottöne darstellen zu können. Der Farbraum AdobeRGB 1998 war geboren. Damals konnten allerdings nur wenige, sehr teure Grafikmonitore diese Farben auch tatsächlich darstellen.

Die Grafik zeigt einen Vergleich zwischen den Farbräumen. Der bunt dargestellte Farbraum ist der sRGB-Farbraum, das weiße Gitter stellt den AdobeRGB-Farbraum dar. Das Bild ist ein Screenshot von der Seite http://iccview.de - eine sehr interessante Seite um Farbräume sichtbar zu machen. Funktioniert leider nur unter Windows mit dem Internet-Explorer, der dazu auch noch ein Plugin braucht.

So - der AdobeRGB-Farbraum ist nun also deutlich größer aber ist er auch besser?

Wenn der eigene Monitor die Farben nicht anzeigen kann, ist der größere Farbraum wenig sinnvoll - man sieht eh nicht mehr Farben. Außerdem muss man zwingend Farbmanagement betreiben, sonst ist die Darstellung falsch.

Ohne Farbmanagement werden die Farben einfach im Monitorfarbraum dargestellt - die Farbe 255/0/0 bedeutet dann einfach das stärkste Rot, dass der Monitor anzeigen kann. Im Falle eines optimalen sRGB-Monitors entspricht das dann also genau einem kräftigen Rot an der äußerstenn Spitze des sRGB-Farbraumes. Im größteren Adobe-RGB-Farbraum entspricht 219/0/0 dieser Farbe.



Ein sRGB-Monitor ohne Farbmanagement zeigt jetzt also ein ganz anderes rot an. Mit Farbmanagement würde das Ganze wieder angepaast und umgerechnet, allerdings hat man eben nichts vom größten Farbraum, weil man die Unterschiede sowieso nicht sehen kann.

Erst mit einem Monitor, der einen größeren Farbraum darstellen kann (ein sogenannter Wide-Gamut-Monitor), ist der AdobeRGB-Farbraum evtl. sinnvoll. Seit einigen Jahren sind diese Monitore bezahlbar geworden, allerdings sollten sie unbedingt kalibriert und profiliert werden damit man die vielen Farben auch "zähmen" kann (s. letzter Newsletter und Artikel von Dirk in diesem Newsletter).

Theoretisch gibt es Vorteile im Druck, da einige Farben (vor allem im Grün- und Gelbbereich) gedruckt werden können, die im sRGB-Farbraum nicht vorkommen. Diese Farben werden vom AdobeRGB-Farbraum weitgehend abgedeckt.

Allerdings erfordert auch das zwingend Farbmanagement. Die meisten Ausbelichter gehen davon aus, dass sowieso nur Bilder im sRGB-Farbraum angeliefert werden, und dann wird das Endergebnis wieder falsch.

Wenn man also die Vorteile von AdobeRGB nutzen möchte, dann muss man darauf achten, dass der gewählte Ausbelichter auch Farbmanagement einsetzt.

Wer im RAW-Format fotografiert, für den ist es übrigens egal, was an der Kamera eingestellt ist, da der Farbraum erst später bei der Entwicklung festgelegt wird - das RAW-Format speichert nur die Sensordaten und hält noch alle Möglichkeiten offen.

Noch mal kurz zusammengefasst:

- wer kein Farbmanagement einsetzt, einen älteren Monitor ohne Wide-Gamut hat oder ein Notebook als Hauptrechner verwendet, sollte sRGB benutzen.
- wer seine Bilder beim Drogeriemarkt um die Ecke ausdrucken lässt, sollte sRGB benutzen.
- wer Bilder ins Internet stellt, sollte sRGB benutzen.

AdobeRGB kann benutzen

- wer Lust hat, sich in die ganze Sache ernsthaft einzuarbeiten, Farbmanagement betreibt, einen Wide-Gamut-Monitor sein Eigen nennt, der den erweiterten Farbraum auch anzeigen kann und beim Ausdrucken darauf achtet, dass beim Ausbelichter auch Farbmanagement eingesetzt wird - dann bleiben allerdings nicht mehr viele Anbieter.

Bilder, die ins Web gestellt werden sollen, müssenn dann allerdings vorher ins sRGB-Format konvertiert werden.

Wer sich fürr den größeren Farbraum entscheidet hat also in jedem Fall mehr Arbeit. Inzwischen gibt es eine Reihe noch größerer Farbräumeräume, die aber noch wenig verbreitet sind und auf die ich hier deshalb nicht eingehe.

terrikay

von Hans

Scripten mit Python

Bilderrahmen erstellen in Python

An Hand dieses kleinen Beispiels wollen wir erklären, wie einfach man die Funktionalität von GIMP mit Pythonskripts erweitern kann. Python selbst ist eine relativ einfach zu erlernende Skriptsprache, welche etliche Anbindungen zu anderen Programmen hat. Dazu gehören z. B. GIMP und QT.

Zum besseren Verständnis sind die Zeilen nummeriert worden. Das komplette Listing findet ihr unter snippet002.html.

In diesem Artikel bezieht sich eine vierstellige Nummer in eckigen Klammern auf die dort angegebene Zeilennummerierung. Die Zeilennummer ist nicht Bestandteil des Codes.

Die Aufgabe

Das Programm soll um ein vorhandenes Bild einen Rahmen zeichnen. Der Rahmen besteht aus einem äußeren Rahmen dessen Breite durch einen Schieberegler (Slider) und dessen Farbe durch eine Farbauswahl eingestellt werden soll. Auf dem äußeren Rahmen soll eine dünne Linie gezeichnet werden, hier als innerer Rahmen bezeichnet. Farbe und zu verwendender Pinsel sollen einstellbar sein.

Das Skript besteht aus mehreren Teilen

- Initialisierung (globale Voreinstellung)
- Hauptfunktion
- Nebenfunktionen
- Implementierung in GIMP (Registrierung)
- Startfunktion

Wir werden die Funktionen in der Reihenfolge durchgehen, wie sie eigentlich abgearbeitet werden.

Startfunktion

Besteht lediglich aus einer Zeile Die Funktion main() muss in jedem Skript für GIMP vorkommen und steht normalerweise immer am Ende des Programms (0186)

Implementierung in GIMP

Dieses ist eine der kompliziertesten Funktionen. Wenn man den Aufbau dieser Funktion verstanden hat, dann ist sie jedoch nicht mehr so kompliziert.

```
0137 register( "hans_snip002",
0138   "Snippet 002",
0139   "pyGimp Kurs",
0140   "Hans-G. Normann",
0141   "(©) 2009 Hans-G. Normann",
0142   "2009-05-19",
0143   "<Image>/Python-Fu/Snippet/Nr002/Bilderrahmen",
0144   None,
0145   [ (PF_SLIDER, 'v_asize',     "Breite äußerer Rand",       10,[1,300,1]),
0146     (PF_SLIDER, 'v_isize',     "Breite innerer Rand",       10,[1,300,1]),
0147     (PF_COLOR,  'v_framecolor',"Farbe des Randes",          (0,0,0)),
0148     (PF_BRUSH,  'v_brush',     "Pinsel",                    'Circle (01)'),
0149     (PF_COLOR,  'v_brushcolor',"Linienfarbe",               (255,0,0))
0150   ],
0151   '',
0152   snip002)
```

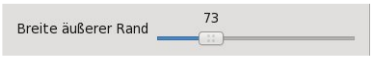
In Zeile (0137) wird angegeben, unter welchem internen Namen diese Funktion in GIMP registriert werden soll. Dieser Name ist für den Anwender nicht sichtbar. Zeile (0143) gibt an, wo unsere Funktion in GIMP zu finden ist. Wir finden sie entweder im Menü unter Python-FU / Snippet / Nr 002 / Bilderrahmen oder im Kontextmenü zu einem Bild (rechte Maustaste auf das Bild).

Laut Dokumentation folgt nun ein Parameter für die Einstellungen. Das ist für den Anfänger schwer verständlich aber aus Sicht des Programmierers ist das so. Dieser eine Parameter kann sich aus diversen Elementen zusammensetzen. Ist mehr als ein Element vorhanden, dann muss dieser Parameter in Eckige Klammern gesetzt werden. Im Beispiel geht der Parameter von Zeile (0145) bis (0150).

Wir haben hier fünf Elemente vorgesehen. Die Maße für die Rahmen werden durch einen Schieberegler (PF_SLIDER) eingestellt. Beim Ersten Aufruf hat der Regler den Wert 10 und kann einen Wert von 1 bis 300 aufnehmen. Beim Verschieben erhöht sich der wert um 1. Der gewählte Wert wird in der Variablen v_asize (0145) oder v_iszie (0146) zurückgegeben. Es ist natürlich auch der Text anzugeben, welcher vor dem Schieberegler stehen soll.

Die Farbe für inneren und äußeren Rand werden durch eine Farbauswahl (PF_COLOR) (0147), (0149) festgelegt. Das gemeine an den Farbangaben ist, das diese in RGB anzugeben sind. (255,0,0) ist reines rot, (0,0,0) ist schwarz, (255,255,255) ist weiß.

Hier die verwendeten Elemente in der grafischen Ausgabe



PF_SLIDER

PF_COLOR

PF_BRUSH

Zusammen sieht das dann so aus:

PARAMETER

Werden bei der Entwicklung in der Registerfunktion Änderungen vorgenommen, dann muss GIMP neu gestartet werden. Deshalb empfiehlt es sich vorher einen Plan zu machen, was benötigt wird. Eine Datei kann mehr als eine Registerfunktion enthalten.

Es sind hier noch zwei Funktionen zur Anzeige der Lizenzbedingungen und des Haftungsausschlusses in deutsch und englisch hinzugefügt worden. Diese sind für das eigentliche Programm nicht notwendig, sollten jedoch in keinem Programm fehlen. Ob

man dieses in dieser Form macht, überlasse ich einmal den Beteiligten selbst.

Initialisierung

```
0001 #!/usr/bin/env python
0002 # -*- coding: utf-8 -*-
```

Der Übersicht halber sollte man in einem Programm einen kleinen Block haben, wo man bereits die wichtigsten Variablen vorbelegt (0001) bis(0033). (0001) ist Linux - spezifisch und gibt an, wodurch diese Anweisungen interpretiert werden sollen. (0002) ist ebenso für Linux. Darin wird bekannt gegeben, welcher Zeichensatz zu verwenden ist.

```
0004 from gimpfu import *
```

(0004) diese Angabe ist für GIMP Skripte obligatorisch. In der Pythondatei gimpfu sind sämtliche Zugriffe von Python auf GIMP geregelt sowie die Funktionen definiert.

Es empfiehlt sich eine Variable für das Debuggen einzurichten. Ist diese Variable auf True (ja) gesetzt, dann werden z.B. mittels print Befehl zusätzliche Ausgaben auf das Terminal geschrieben. Dazu muss GIMP aber aus einem Terminal (Konsole, Xterminal, DOS Box, etc.) gestartet werden.

Hauptschleife

Die Hauptschleife beginnt mit Zeile (0038). Die beiden Parameter für image und drawable müssen immer als erstes angegeben werden. Die namen für die beiden Parameter önnen frei gewählt erden. Diesen folgen die Parameter aus der Registerfunktion. Reihenfolge und Anzahl der Parameter müssen exakt mit der Registerfunktion übereinstimmen.

```
0039   debug = False
0040
0041   if debug:
0042     print "debug is on"
0043
0044   # Undo Funktion ausschalten, wenn debug nicht läuft
0045   if not debug:
0046     pdb.gimp_image_undo_group_start(img)
0047     pdb.gimp_image_undo_disable(img)
0048     print 'stop undo function'
0049
0050
0051   pdb.gimp_selection_clear(img)
0052
0053   #Rand prüfen
0054   if o_isize > o_osize:
0055     o_isize = o_osize
0056
0057   pdb.gimp_context_set_foreground(o_framecolor)
```

Als erstes beginnen wir mit den notwendigen Voreinstellungen für die Funktion (0039) bis (0057). Dazu gehören z. B. Einstellen von Vorder- und Hintergrundfarbe, Auswahl aufheben, etc. Wenn der innere Rand größer als der äußere Rand ist, würden wir in

das Bild zeichnen. Deshalb verhindern wir dieses in Zeile (0054) und (0055).

```
0065  LayerDefaults = {
0066    'Width':img.width,
0067    'Height':img.height,
0068    'Type':RGBA_IMAGE,
0069    'Name':'New Layer',
0070    'Opacity':100,
0071    'Mode':NORMAL_MODE
0072  }
0073
0074  n = len(img.layers)
0075  o_layer = f_layer_new(img, LayerDefaults)
0076  pdb.gimp_image_add_layer(img, o_layer, n+1)
```

Zunächst vergrößern wir die Leinwand um die in o_asize angegebenen Pixel je Seite, also insgesamt um den doppelten Betrag. Weil diese Funktion so elend viele Parameter hat, kann man wegen der Verständlichkeit diese in ein sogenanntes assoziatives Array packen (0065) bis (0072). Bei einem Assoziativem Array ordnet man jedem Element einen Namen zu mit welchem es später dann wieder aufgerufen werden kann. Der Code wird dadurch lesbarer.

Zunächst ermitteln wir, wie viele Ebenen das Bild hat (0074) bevor wir in Zeile (0075) und (0076) eine neue transparente Ebene erzeugen und fügen sie dem Bild als unterste Ebene hinzu. Anschließend füllen wir sie mit der Vordergrundfarbe, welche schon vorher gesetzt wurde (0057).

```
0088  n = len(img.layers)
0089  LayerDefaults['Name'] = 'New Frame'
0090  o_layer1 = f_layer_new(img, LayerDefaults)
0091  pdb.gimp_image_add_layer(img, o_layer1, n-1)
0092  pdb.gimp_image_set_active_layer(img, o_layer1)
```

Es wird eine weitere transparente Ebene erzeugt, welche über die des Hintergrundes gelegt wird. (0088) bis (0092). Die neue Ebene wird zur aktiven Ebene gemacht (0092).

```
0099  pdb.gimp_selection_all(img)
0100  pdb.gimp_selection_shrink(img, o_isize)
0101  pdb.gimp_edit_stroke(o_layer1)
```

Auf dieser Ebene erzeugen wir eine Auswahl (alles) (0099), verkleinern dieses um den wert o_isize (0100) und zeichnen sie mit dem aktuellen Pinsel nach (0101)

Funktionen

```
0122 def f_layer_new(img,p):

0123  """creates new layer by using parameters LayerDefaults\nreturns parameters"""
0124  return pdb.gimp_layer_new(img, p['Width'], p['Height'], p['Type'],p['Name'],
p['Opacity'],p['Mode'])
```

Wir haben hier nur eine Zusatzfunktion definiert (f_layer_new) (0122). Es ist lediglich eine Redefinition der bereits durch gimpfu bereitgestellten Funktion pdb.gimp_layer_new. Aus Gründen der Lesbarkeit wurde dieser Weg gewählt.

Ergebnis

Das war eigentlich schon alles. Man kann diese Funktion natürlich erweitern, z. B. durch

- Automatisches Hinzufügen eines Copyrights
- Zeichnen von mehreren kleiner werdenden inneren Rahmen
- Zeichnen von mehreren um wenige Pixel versetzte Rahmen
- Inneren Rahmen mit Bumpmap aufpeppen
- Verwendung von Farbverläufen

Der Kreativität sind keine Grenzen gesetzt. Viel Spaß in einen Einstieg in Python wünscht euch

Hans

An diesem Newsletter haben mitgewirkt:

Autoren:

rainerstollwetter, terrikay, hans, eleanora

Korrekturlesen:

Cheeky-devil

Zusammenstellung:

Eleanora

Alle Mitwirkenden freuen sich auf dein Feedback. Hast du Vorschläge für einen weiteren Artikel oder möchtest du selber auch mitwirken, dann melde dich in unserem Gimp-Werkstatt-Forum.