

2.2 Bilder in Gamera (1)

Allgemeines zu Gamera

- interaktive Nutzung über GUI
 - Start mit Befehl `gamera_gui &`
 - Befehlseingabe in Python-Shell
 - Bildmethoden auch über Kontextmenü in Objektliste
- automatisierte Nutzung über Python-Scripts
 - Script muss zuerst folgenden Code enthalten:

```
from gamera.core import *
init_gamera()
```
- Bildmethoden heißen *Plugins*
 - unterschiedliche Methoden je nach Bildtyp (RGB, Onebit, ...)
 - nach Kategorien dokumentiert in Gamera HTML-Doku

1

2.2 Bilder in Gamera (3)

Wichtige Bildeigenschaften

- Größe des Bildes:
 - `ncols` = Anzahl Spalten (x-Koordinate)
 - `nrows` = Anzahl Zeilen (y-Koordinate)
 - `x` läuft also von 0 bis `ncols - 1` und `y` von 0 bis `nrows - 1`
 - Datentyp der Bildpunkte
 - `data.pixel_type` kann sein RGB, GREYSCALE oder ONEBIT
- Diese Eigenschaften sind Properties, werden also ohne Funktionsklammern aufgerufen, z.B.

```
if img.data.pixel_type != ONEBIT:
```

3

2.2 Bilder in Gamera (2)

Bildkonstruktor

```
Image(Point ul, Point lr, pixeltype)
```

- `ul` meint "upper left" und `lr` "lower right"
- Koordinaten werden ab links oben gemessen
- `pixeltype` kann sein: RGB, GREYSCALE oder ONEBIT (default)
- Speicher wird allokiert und alle Pixel Weiß gesetzt

Beispiel:

```
# Anlegen 11x11 Farbbild
img = Image(Point(0,0), Point(10,10), RGB)
```

- es gibt weitere Konstruktoren, z.B.

```
Image(otherimage)
```

zum Anlegen eines Bilds derselben Größe und Typ

2

2.2 Bilder in Gamera (4)

Pixel-Zugriff

- erfolgt mit den Methoden
 - `get(Point(x,y))`
 - `set(Point(x,y), pixelvalue)`
wobei `pixelvalue` je nach Bildtyp gegeben wird durch
 - ▷ 0 oder 1 für Onebit-Bilder
 - ▷ 0 bis 255 für Greyscale-Bilder
 - ▷ `RGBPixel(r,g,b)` mit $0 \leq r,g,b \leq 255$ für RGB-Bilder

Beispiel:

```
# zähle Anzahl schwarzer Pixel in Onebit-Bild
n = 0
for x in range(img.ncols):
    for y in range(img.nrows):
        n += img.get(Point(x,y))
```

4

2.2 Bilder in Gamera (5)

Bilddatei-Zugriff

- einlesen geht mit der Funktion

- `load_image(filename)`
 - gibt Image zurück
 - kann TIFF und PNG lesen

- schreiben geht mit der Image-Methode

- `save_PNG(filename)`

Beispiel:

```
# schreibe 11x11 Bild mit rotem Punkt in Mitte
img = Image(Point(0,0), Point(10,10), RGB)
img.set(Point(5,5), RGBPixel(255,0,0))
img.save_PNG("out.png")
```

5

2.2 Bilder in Gamera (7)

Views (1)

- Gamera benutzt "shared data" Modell

- dieselben Daten über verschiedene *Views* zugreifbar
- Datentyp *Image* ist tatsächlich ein View
- Vorteile:
 - Bilder sind "lightweight" Objekte (auch call by value schnell)
 - Daten können verschieden dargestellt werden (z.B. als CC oder Onebit-Bild)
 - Bildteile können ohne Neuallokation wie Bilder behandelt werden

- Erzeugung neuen Views mit

```
SubImage(Image img, Point ul, Point lr)
```

- *ul* meint "upper left" und *lr* "lower right"
- der so erzeugte View verwendet dieselben Daten wie *img*

7

2.2 Bilder in Gamera (6)

Zeichenroutinen

- Plugins in Kategorie "Draw", z.B.

- `draw_line(p1, p2, pixelvalue, thickness)`
- `flood_fill(point, pixelvalue)`

Konvertierung zwischen Bildtypen

- mit den Image-Methoden `to_onebit()`, `to_rgb()`

und `to_greyscale()`

- Algorithmus Greyscale → Onebit später

Beispiel:

```
# mache Bild, wenn nötig, zu SW-Bild
if img.data.pixel_type != ONEBIT:
    img = img.to_onebit()
```

6

2.2 Bilder in Gamera (8)

Views (2)

- wichtige Eigenschaften von Image-Views:

- *data* = zugrundeliegende Bilddaten
- *offset_x* = Verschiebung x-Koordinaten gegenüber Data
- *offset_y* = Verschiebung y-Koordinaten gegenüber Data

Beispiel in Python-Shell von *gamera_gui*:

```
# erzeuge zwei Views auf denselben Daten
>>> img1 = Image(Point(0,0), Point(50,50))
>>> img2 = SubImage(img1, Point(5,5), Point(10,10))
>>> img2.offset_x
5
>>> img2.ncols
6
>>> img2.data.ncols
51
```

8

2.2 Bilder in Gamera (9)

Speichermanagement

- auf Python-Seite *Garbage Collection*
 - *Reference Counter* zählen wieviele Views ein Bild verwenden
 - Speicher für das Bild wird erst freigegeben, wenn es von keinem View mehr verwendet wird
 - Beispiel:

```
img1 = Image(Point(0,0),Point(50,50))
img2 = SubImage(img1,Point(5,5),Point(10,10))
del img1 # löscht view img1, aber nicht img1.data
del img2 # jetzt erst wird img1.data gelöscht
          # weil keine Referenz darauf mehr da ist
```

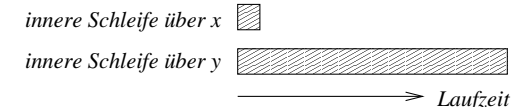
- auf C++-Seite (eigene Plugins):
 - *ImageView* und *ImageData* müssen explizit deleted werden
 - Ausnahme: *ImageView* als return-Wert an Python gegeben

9

2.2 Bilder in Gamera (11)

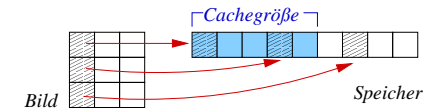
Laufzeitvergleich der Zugriffs-Reihenfolge

- Testergebnisse laut Gamera-Doku:



- Ursache:

- Speicherhierarchie des Rechners
 - ▷ benachbarte Daten werden blockweise in Cache geladen
 - ▷ bei Iteration über eine Spalte wird auf nicht benachbarten Speicher zugegriffen
 - ⇒ Daten werden ständig zwischen Cache und Hauptspeicher ausgetauscht

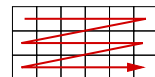


11

2.2 Bilder in Gamera (10)

Interne Datenspeicherung

- zwei interne Formate:
 - *DENSE* = Speicherung aller Pixel
 - ▷ Pixel in *Row-Column Order* im Speicher abgelegt:
 - ⇒ Sequentieller Zugriff in dieser Reihenfolge
 - schneller, d.h. innere Schleife über *x* wählen (s.u.)
 - *RLE* = Run Length Encoding
 - ▷ nur bei Onebit-Bildern möglich; kann evtl. weniger Speicher brauchen
 - ▷ manche Operationen (z.B. Verschieben) können effizienter sein
 - ▷ meist ist der Pixelzugriff aber langsamer ⇒ besser normales Format verwenden
- Wie wählt man das Format?
 - im Konstruktor oder bei *image_copy()* als optionales Argument
 - kann abgefragt werden über *data.storage_format*
 - mögliche Werte: *DENSE* oder *RLE*



10