

# Sketch: Flexibel Zeichnen mit Python

**Bernhard Herzog**  
Intevation GmbH

Sketch ist ein Vektorzeichenprogramm für GNU/Linux. Dieser Vortrag gibt einen Überblick über die Fähigkeiten von Sketch und eine Einführung in die Erweiterungsmöglichkeiten durch Skripte.

## 1. Einführung

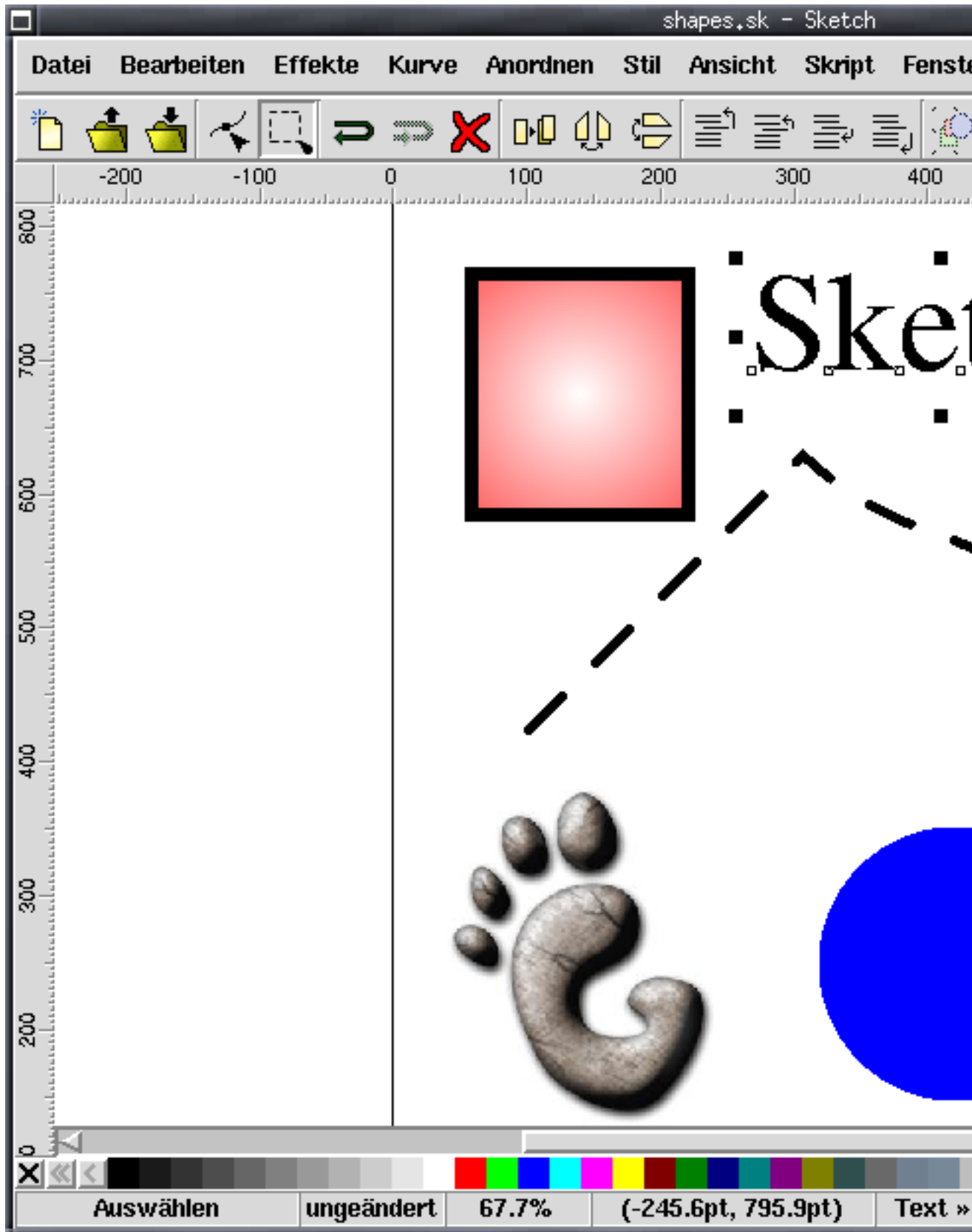
Anders als bei Bildbearbeitungsprogrammen wie etwa Gimp (<http://www.gimp.org>), besteht eine Grafik in einem Vektorzeichenprogramm wie Sketch (<http://sketch.sourceforge.net/>) nicht aus Pixeln, sondern aus Objekten, die im wesentlichen als mathematische Beschreibung vorliegen. Diese Darstellung ermöglicht zum einen, die Grafiken beliebig zu Skalieren. Außerdem lassen sie sich leichter ändern, da man, um etwa die Farbe eines Objektes zu ändern, das Objekt nicht komplett neu zeichnen muß, sondern einfach eine neue Farbe zuweisen kann.

Die aktuelle stabile Version ist 0.6.13. Dieser Artikel bezieht sich hauptsächlich auf diese Version, Ende gibt es jedoch noch einen Ausblick auf die Entwicklerversionen (0.7.x)

## 2. Arbeiten mit Sketch

### 2.1. Hauptfenster und Werkzeuge

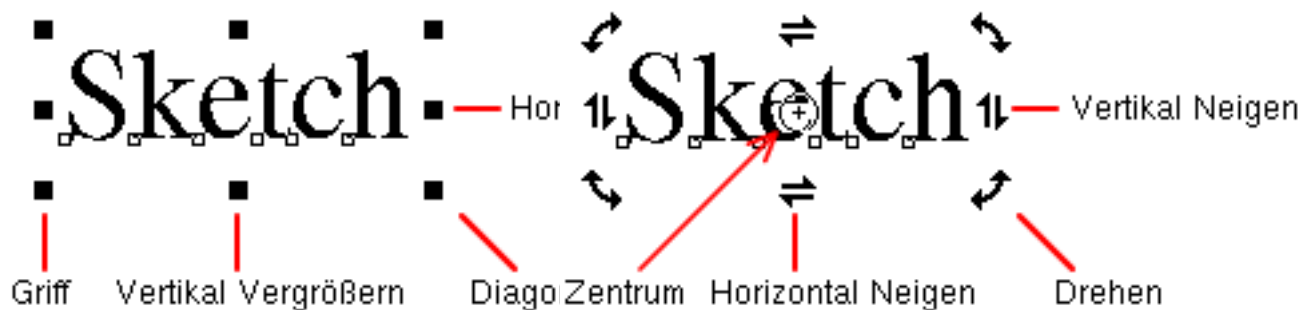
Der folgende Screenshot zeigt Das Hauptfenster von Sketch mit einigen einfachen Grafikobjekten.



Das Hauptfenster gliedert sich grob, von oben nach unten, in Menü, Werkzeugleiste, Zeichenfläche, Farbpalette und Statuszeile.

Die meisten Arbeiten an einer Grafik finden auf der Zeichenfläche statt, wo man mit der Maus Objekte zeichnet und bearbeitet. Was man auf der Zeichenfläche gerade tun kann, hängt vom gerade eingestellten Modus ab, der in der Statuszeile ganz links angezeigt wird.

Im obigen Bildschirmfoto befindet sich Sketch gerade im "Bearbeiten"-Modus, in dem Objekte ausgewählt und verschoben oder gedreht werden können. Das Textobjekt "Sketch" ist gerade ausgewählt und mit den kleinen Quadraten um das Objekt herum kann man das Objekt durch einfaches Klicken und Ziehen vergrößern. Ein einfacher Klick auf das ausgewählte Objekt ändert die Quadrate zu Pfeilen, mit denen man das Objekt drehen und neigen kann:



Wenn man anfängt, mit Sketch zu arbeiten, will man typischerweise entweder an einer bereits vorhandenen Zeichnung weiter arbeiten oder eine Neue anfangen.

Bestehende Zeichnungen lädt man über das Datei-Menü oder über den Entsprechenden Knopf in der Werkzeugleiste. Der normale Datei-Öffnen Befehl kann auch einige fremde Formate laden. Die unterstützen Formate werden automatisch erkannt. Mehr Informationen dazu finden sich im Abschnitt über Datenaustausch.

Neue Objekte zeichnet man mit den Zeichenwerkzeugen, die man am einfachsten über die Werkzeugleiste erreicht:



Von links nach rechts sind dies: Rechteck, Kreis/Ellipse, Bézierkurve, Linienzug, Text und Rasterbilder/EPS.

Rechtecke und Ellipsen zeichnet man einfach durch Klicken und Ziehen mit der Maus. Bei gleichzeitig gedrückter Strg-Taste werden Quadrate beziehungsweise Kreise gezeichnet. Mit der Umschalten-Taste gibt der erste Punkt den Mittelpunkt statt einer Ecke an.

Bei Linienzügen und Bézierkurven sind mehrere Mausklicks erforderlich, wobei ein Klick bei Linienzügen einen Eckpunkt festlegt. Bei Bézierkurven bestimmt ein Klick und nachfolgendes Ziehen einen Punkt und die Richtung der Kurve an dem Punkt. Die mittlere Maustaste beendet den Zeichenvorgang.

Beim Text-Werkzeug klicken Sie einfach an die Stelle, wo der Text erscheinen soll und tippen ihn ein.

Das Raster/EPS-Werkzeug öffnet einen Dateidialog mit dem Sie die Datei auswählen können, die eingebettet wird. Sketch unterstützt eine Reihe gängiger Rasterformate wie etwa PNG und Jpeg, sowie Encapsulated PostScript (EPS). Nachdem die Datei geladen wurde, können Sie sie mit der Maus auf der Zeichenfläche positionieren. Für EPS-Grafiken ruft Sketch automatisch ghostscript auf, um eine Vorschaugrafik zu erzeugen. Beim Drucken wird die EPS-Datei dann aber so wie sie ist in die PostScript-Ausgabe geschrieben.

Nachdem einige Objekte gezeichnet sind, möchte man sie oft weiter verändern und zum Beispiel einen Kreis etwas vergrößern oder einen Eckpunkt eines Linienzuges verschieben. Dazu gibt es zwei Werkzeuge, "Auswählen" und "Bearbeiten":



Grob gesagt, kann man mit dem Auswahlwerkzeug (links) Objekte Maus auswählen und als ganzes verändern und sie z.B. verschieben oder drehen. Mit dem "Bearbeiten"-Werkzeug (rechts) kann man ein ausgewähltes Objekt im Kleinen ändern und zum Beispiel einem Rechteck abgerundete Ecken verpassen oder einen Knotenpunkt einer Bézierkurve verschieben.

Im Auswahlwerkzeug wählt man mit der Maus Objekte aus, entweder durch einfaches Daraufrufen oder mit dem "Gummiband", indem man klickt und ein Rechteck aufzieht wodurch alle Objekte ausgewählt werden, die in dem Rechteck liegen. Bei gedrückter Strg-Taste werden die so bestimmten Objekte aus der Auswahl entfernt, bei gedrückter Umschalten-Taste zur Auswahl hinzugefügt.

Das Bearbeiten-Werkzeug funktioniert je nach Typ des ausgewählten Objekts anders, es ist aber immer so, daß, wenn das Objekt überhaupt bearbeitet werden kann, entsprechende Griffe erscheinen. Bei Rechtecken kann man dann die Ecken abrunden, Kreise und Ellipsen in Tortenstücke umwandeln und bei Polygonen und Bezierkurven die Eck- und Kontrollpunkte verschieben.

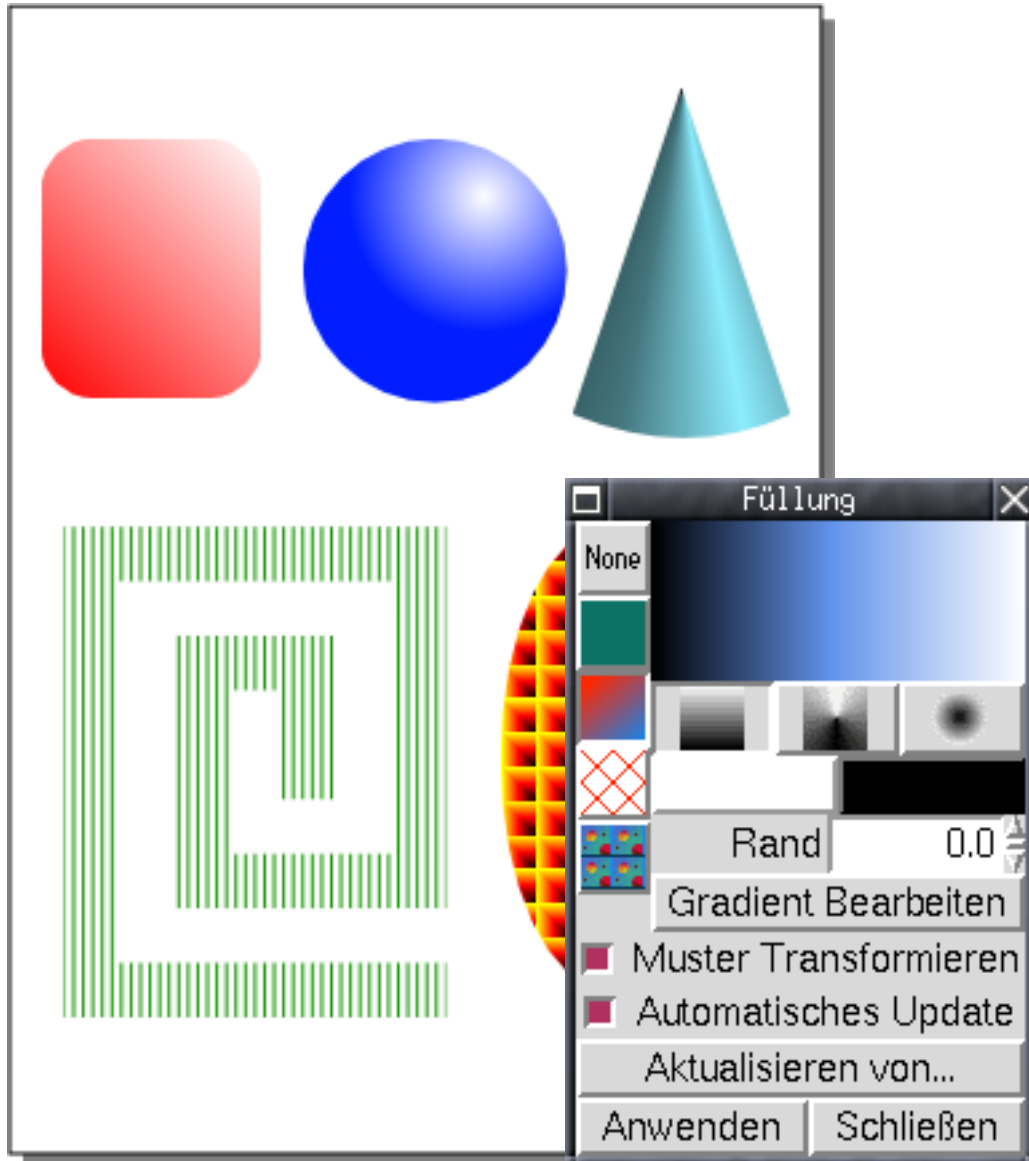
## **2.2. Füllungen und Umrisse**


Die meisten der oben erwähnten Grundobjekte können mit Füllungen und Umrissen versehen werden. Die einzige Ausnahme sind Raster und EPS-Grafiken, sowie zu einem gewissen Grad Text, welcher nur eine Füllung haben kann.

### **2.2.1. Füllungen**

Als Füllungen bietet Sketch neben einfachen Farben auch Farbgradienten, Strichelungen und Muster an, letztere aber bisher nur mit Rasterbildern. Natürlich kann ein Objekt auch einfach gar keine Füllung haben, wodurch das Innere des Objekts gar nicht bemalt wird.

Die folgende Abbildung zeigt Objekte mit allen gängigen Füllungen. Daneben ist ein Screenshot des Dialogs, mit dem sich die Füllungen einstellen lassen.



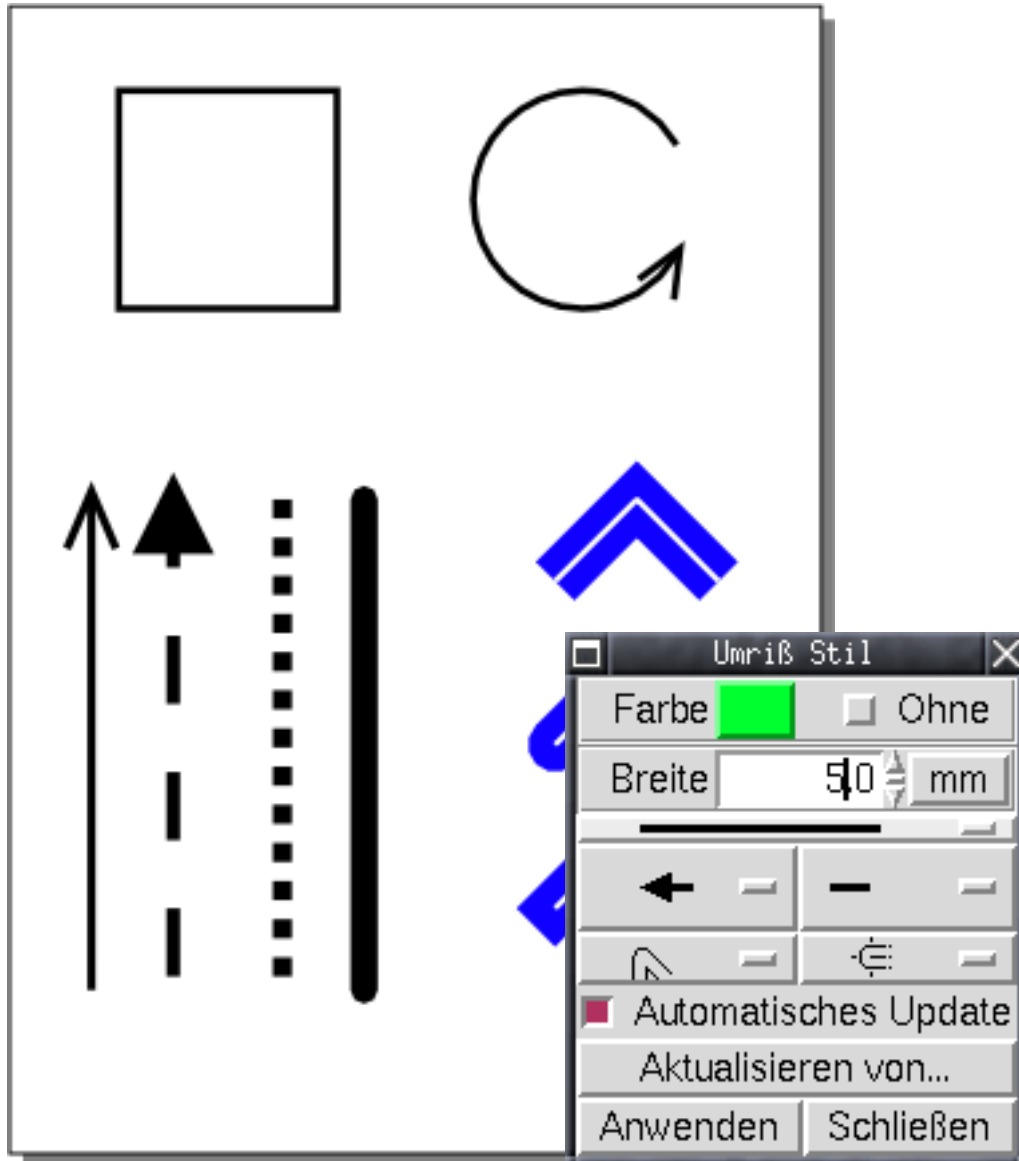
Wenn man nur einfache Farben zuweisen möchte, geht es häufig einfacher und schneller mit einem Klick in die Palette. Ein Klick auf das Kreuz am linken Ende der Palette () entfernt die Füllung komplett.

### 2.2.2. Umriß

Bei den Umrissen kann man Farbe, Liniendicke, Strichelung, Pfeilspitzen und Ecken- und Endenstile definieren. Wie die Füllung ist der Umriß optional. Ein Objekt, für das keinen Umriß definiert ist wird nur gefüllt, sofern eine Füllung zugewiesen ist. Objekte ohne Füllung und Umriß sind unsichtbar, können aber dennoch nützlich sein, etwa für an einer Kurve ausgerichteten Text, wo man die Kurve

verschwinden lassen möchte.

Die folgenden beiden Screenshots zeigt die diversen Möglichkeiten sowie den Liniendialog. Die drei blauen Linien mit dem weißen Kern zeigen die Ecken- und Endenstile. Um nur mal schnell eine neue Linienfarbe zuzuweisen, reicht auch ein Klick mit der mittleren Maustaste in die Palette, wobei wiederum ein Klick auf das Kreuz in der Palette den Umriß ganz entfernt.



## 2.3. Einfacher Text

Für Text bietet Sketch zunächst einmal nur ein sehr einfaches Textobjekt an, das nur einzeiligen Text mit einer einzigen Schriftart darstellen kann. Immerhin kann man es aber mit beliebigen Füllungen versehen, an Pfaden ausrichten und den Text sogar in Kurven umwandeln, die man dann weiter bearbeiten kann, was bei einem der Beispiele zu Überblendungen verwendet wird.

Mit Pluginobjekten erlaubt Sketch mittlerweile auch mehrzeiligen Text.

## 2.4. Gruppen und Ebenen

Zusammengehörende Objekte können zu einer Gruppe zusammengefasst werden. Dadurch kann man sie leichter als ganzes bearbeiten. Natürlich kann eine Gruppe auch wieder in ihre Bestandteile auflösen. Beides, Gruppieren und Gruppierungen Aufheben, geht am einfachsten über die entsprechenden Knöpfe in der Werkzeugleiste:



Hin und wieder ist es nützlich ein Objekt auszuwählen, das Bestandteil einer Gruppe ist um es in irgendeiner Weise zu ändern ohne gleich alle anderen Objekte in der Gruppe mit zu ändern oder die Gruppe erst aufzuheben, das Objekt zu ändern und dann mühsam die Gruppe wieder zusammenzustellen. Dazu wählt man zuerst die Gruppe aus und klickt dann mit gleichzeitig gedrückten Strg- und Umschalttasten auf das Objekt.

Eine weitere Möglichkeit, Objekte zusammenzufassen, sind Ebenen. Ebenen sind sehr viel loser als Gruppen, da automatisch die Objekte in den Ebenen ausgewählt werden statt der Ebenen selbst. Bei Gruppen wird normalerweise die Gruppe an sich ausgewählt. Im Gegensatz zu Gruppen bieten Ebenen aber einige weitere Möglichkeiten, die über den Ebenen-Dialog zugänglich sind:





Wie man sieht, hat jede Ebene einen Namen und es gibt für jede Ebene noch fünf Knöpfe, die, von links nach rechts, festlegen, ob eine Ebene auf der Zeichenfläche dargestellt wird, ob sie gedruckt wird, ob sie editierbar ist, ob sie normal oder nur im Umriß angezeigt wird und in welche Farbe die Umrisse ihrer Objekte dargestellt werden. Die Umrißdarstellung bezieht sich nur auf die Bildschirmdarstellung und hat keinerlei Auswirkungen darauf, wie die Zeichnung gedruckt wird.

## 2.5. Gitter und Hilfslinien

Im obigen Ebenendialog sieht man neben der normalen Ebene, "Ebene 1", auch noch zwei spezielle Ebenen, "Gitter" und "Hilfslinien".

Die Gitterebene enthält das Gitter, das über das "Ausrichten" Menü eingerichtet wird: Der Menüpunkt "Am Gitter Fangen" stellt das Gitter an und aus (ob es sichtbar ist, wird jedoch über den Ebenendialog eingestellt). Wenn dieser Menüpunkt aktiv ist, wirken die Gitterpunkte "magnetisch" und erlauben so, Objekte gezielt in regelmäßigen Abständen anzuordnen. Die Gitterweite, also die Abstände zwischen den Gitterpunkten kann mit dem Gitterdialog festgelegt werden der mit "Gitter. . ." geöffnet wird.

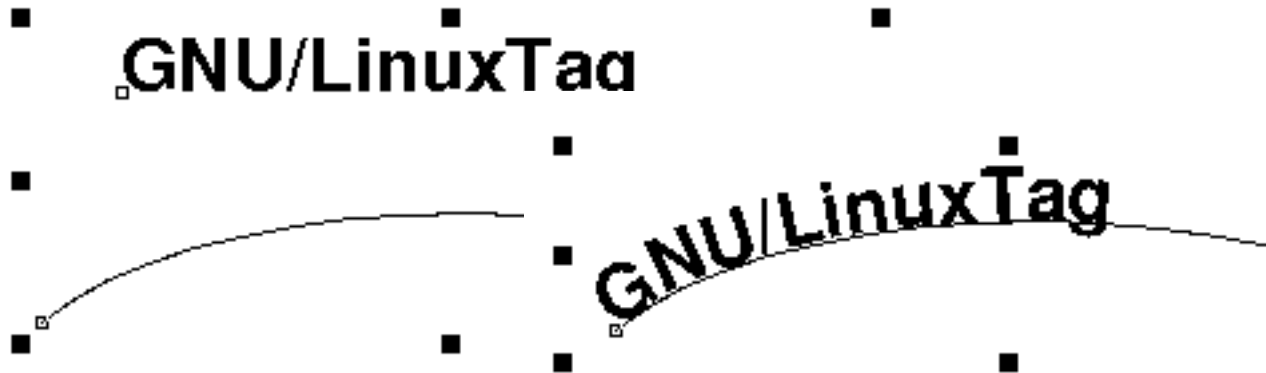
Die Hilfslinienebene enthält Hilfslinien, die, wenn der "An Hilfslinien fangen" aktiv ist ebenfalls magnetisch wirken. Bei einer neuen Zeichnung ist die Hilfslinienebene leer. Neue Hilfslinien zeichnet man mit den beiden "Hilfslinie zeichnen" Menüeinträgen oder man legt sie im Hilfsliniendialog ("Hilfslinien. . .") numerisch fest.

Die so erzeugten Hilfslinien sind waagerechte oder senkrechte Geraden. Das ist das, was man am häufigsten braucht. Es ist aber auch oft praktisch, wenn auch andere Objekte magnetisch sein können. Sketch bietet dazu einerseits den Menüpunkt "An Objekten fangen", durch den die Eckpunkte von Polygonen und andere Spezielle Punkte von Objekten anziehend wirken. darüber hinaus kann man aber auch einfach beliebige Objekte auf die Hilfslinienebene verschiebt.

## 2.6. Spezialeffekte

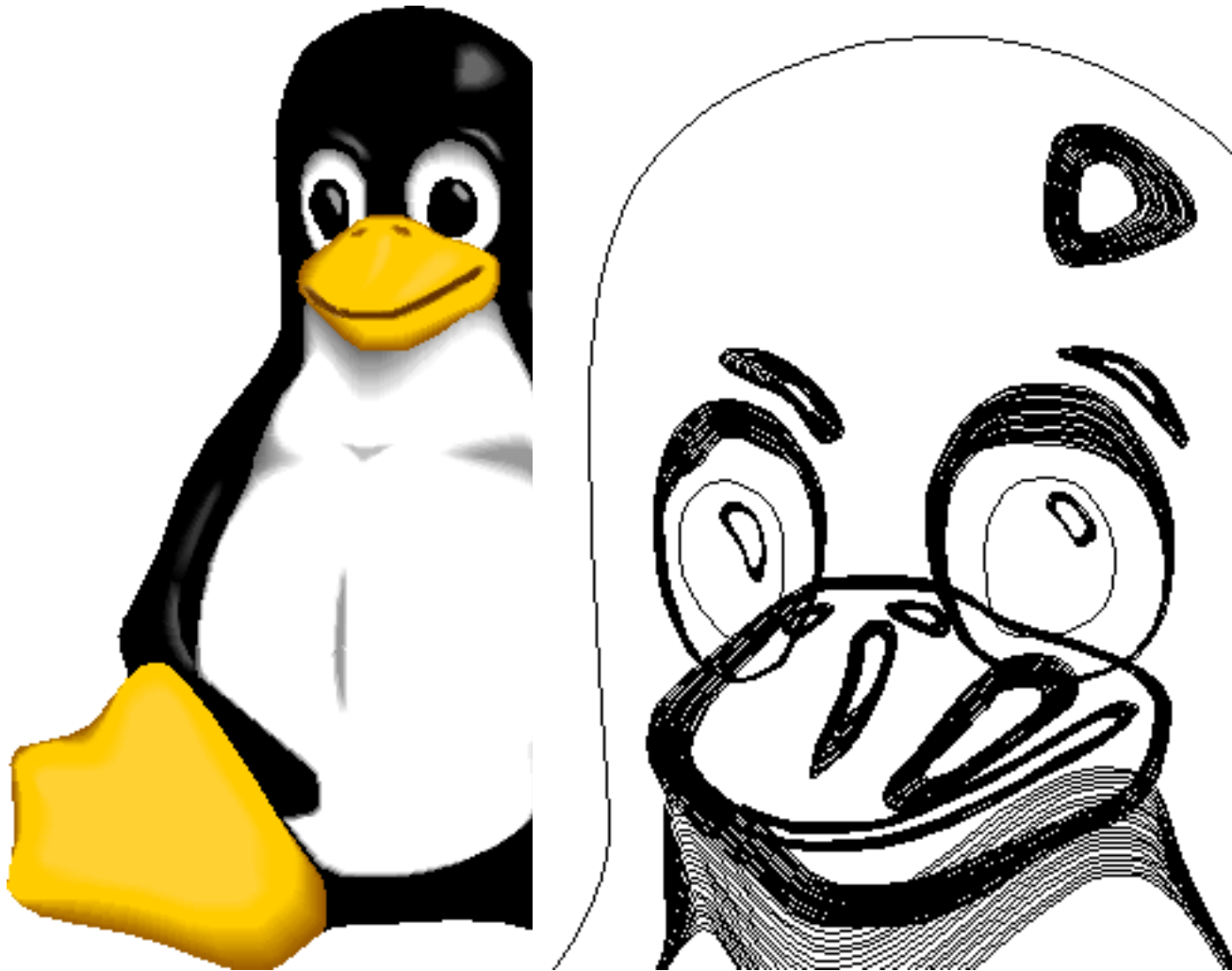
### 2.6.1. Pfadtext

Sketch kann Text an Kurven ausrichten, so daß der Text an der Kurve entlang gesetzt ist. Dazu wählt man das Textobjekt und die Kurve aus (linker Screenshot) und kombiniert sie zum Pfadtextobjekt mit dem Menüpunkt "Pfadtext Erzeugen" im "Effekte" Menü (rechter Screenshot):



### 2.6.2. Blendgruppen

Gradienten bieten schon sehr nützliche Möglichkeiten, um Schattierungen zu erzeugen, manchmal braucht man aber Farbverläufe, die sich mit den vordefinierten Gradienten nicht erzeugen lassen. In solchen Fällen kann man sich mit Überblendungen helfen, wie etwa bei dieser Tux-Version, die ursprünglich von Simon Budig mit einem proprietären Zeichenprogramm vektorisiert wurde:



In der Umriß-Darstellung des Kopfes rechts, sieht man, daß die Farbverläufe aus vielen sehr ähnlichen Objekten bestehen.

Um eine Überblendungen zu erzeugen, wählt man einfach die beiden zu überblendenden Objekte aus wendet den Menüpunkt "Überblenden" aus dem "Effekte" Menü an. Sketch bemüht sich, aus jeder Kombination von Objekten etwas sinnvolles zu machen, sinnvoll einsetzen lassen sich Überblendungen aber an sich nur, wenn beide Objekte eine Ähnliche Struktur haben. Überblendungen werden übrigens automatisch aktualisiert, sobald Anfangs- oder Endobjekt verändert werden.

Als zweites Beispiel für Überblendungen ein Neon-Schriftzug:

a) GNU

b) GNU

c) GNU

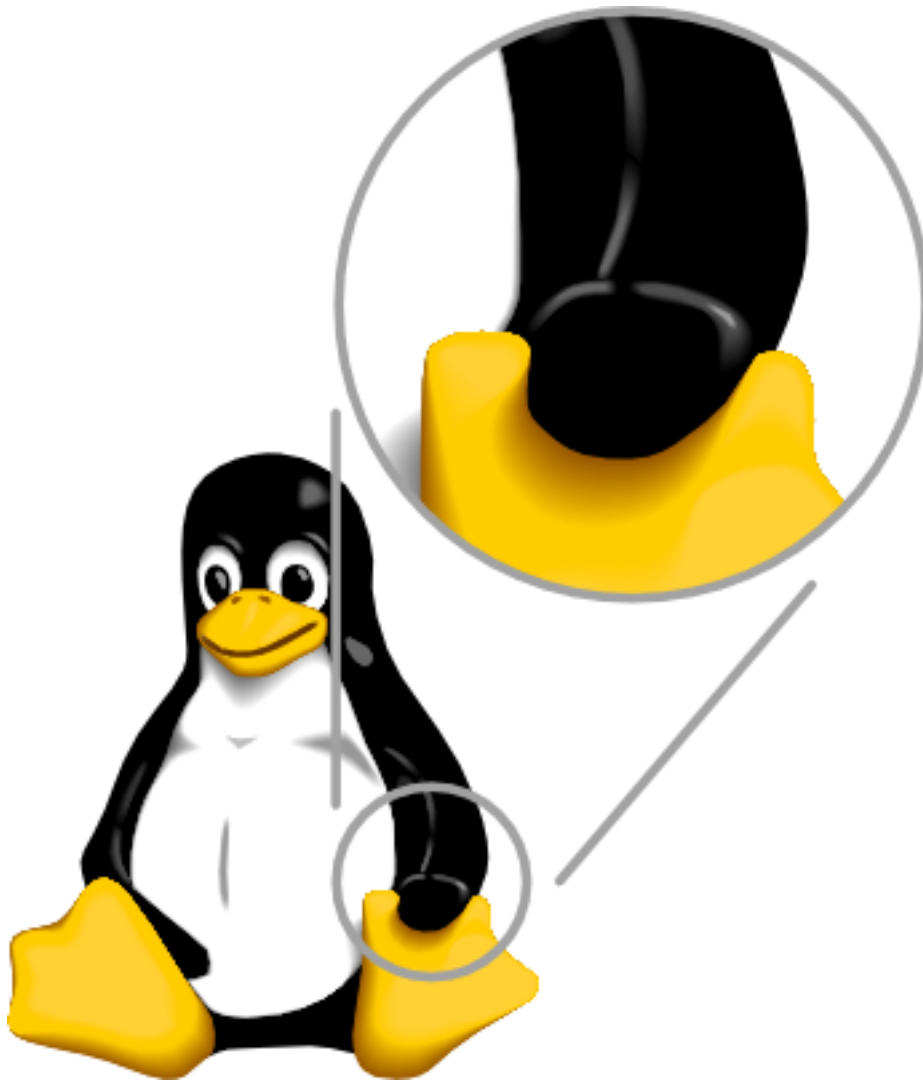
d) GNU

Als Ausgangspunkt in a) ein einfaches Textobjekt. In b) wurde das Textobjekt mit "In Kurven umwandeln" aus dem "Kurve" Menü in eine Gruppe von Bézierobjekten umgewandelt und mit einem schwarzen Umriß versehen. Die Füllung wurde entfernt. In c) wurde der Umriß stark verstärkt und dann eine Kopie mit dünnem weißen Umriß darübergelegt. In d) schließlich sind beide Objekte mit 10 Schritten überblendet. Der schwarze Hintergrund sorgt für den "richtigen" Neon-Effekt.

### 2.6.3. Maskierungen

Maskierungen sind spezielle Gruppen, bei denen ein Objekt eine Region definiert, auf die das Zeichnen der in der Gruppe enthaltenen Objekte beschränkt wird. Im folgenden Beispiel ist der große Kreis eine

Maskierung in der der Kreis als Maske für eine vergrößerte Kopie des Pinguins fungiert:



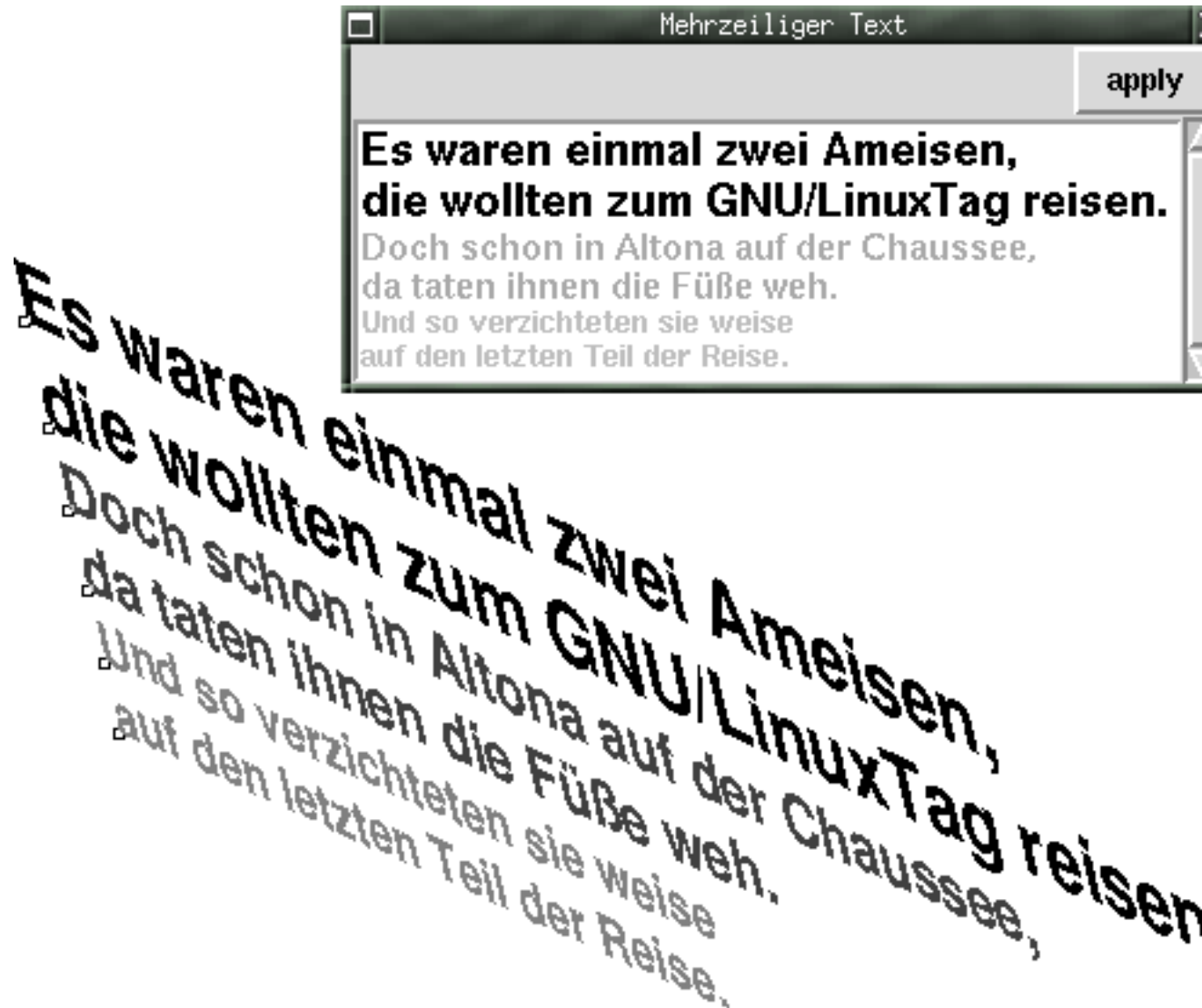
Um eine Maskierung zu erzeugen, wählt man die Maske und die zu maskierenden Objekte aus, wobei man darauf achten muß, daß die Maske oben liegt, und ruft den Menüpunkt "Maskierungsgruppe erzeugen" auf.

## 2.7. Pluginobjekte und andere Erweiterungen

Die eingebauten Objekttypen sind zwar schon sehr mächtig, bieten aber nicht immer genau das, was man braucht. Daher können in Sketch mit Plugins neue Objekttypen definiert werden.

### 2.7.1. Multi-Line Text

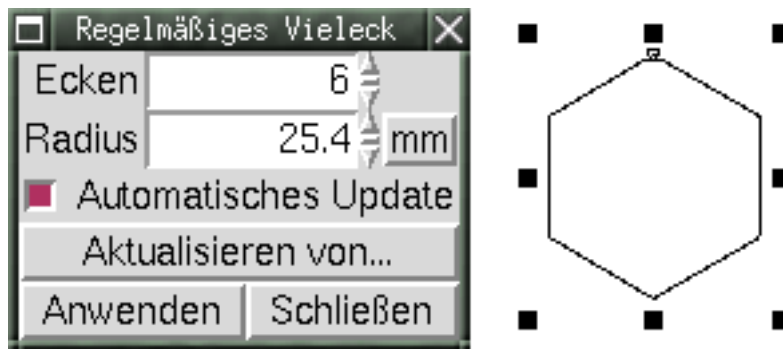
Weiter oben wurde bereits das Plugin für mehrzeiligen Text erwähnt. Es wurde von Christof Ecker geschrieben und ist seit Version 0.6.13 ein Standardbestandteil des Sketch Pakets. Hier als Beispiel das zum Plugin gehörende Fenster, in dem der Text bearbeitet wird und die Zeichenfläche auf der der Text mit dem Auswahlwerkzeug gedreht und geneigt wurde:



### 2.7.2. Regelmäßige Polygone

Das erste Pluginobjekt, das für Sketch geschrieben wurde, erzeugt reguläre Vielecke, und diente

hauptsächlich dazu zu testen, wie Pluginobjekte funktionieren können. Es ist an sich wenig spektakulär:

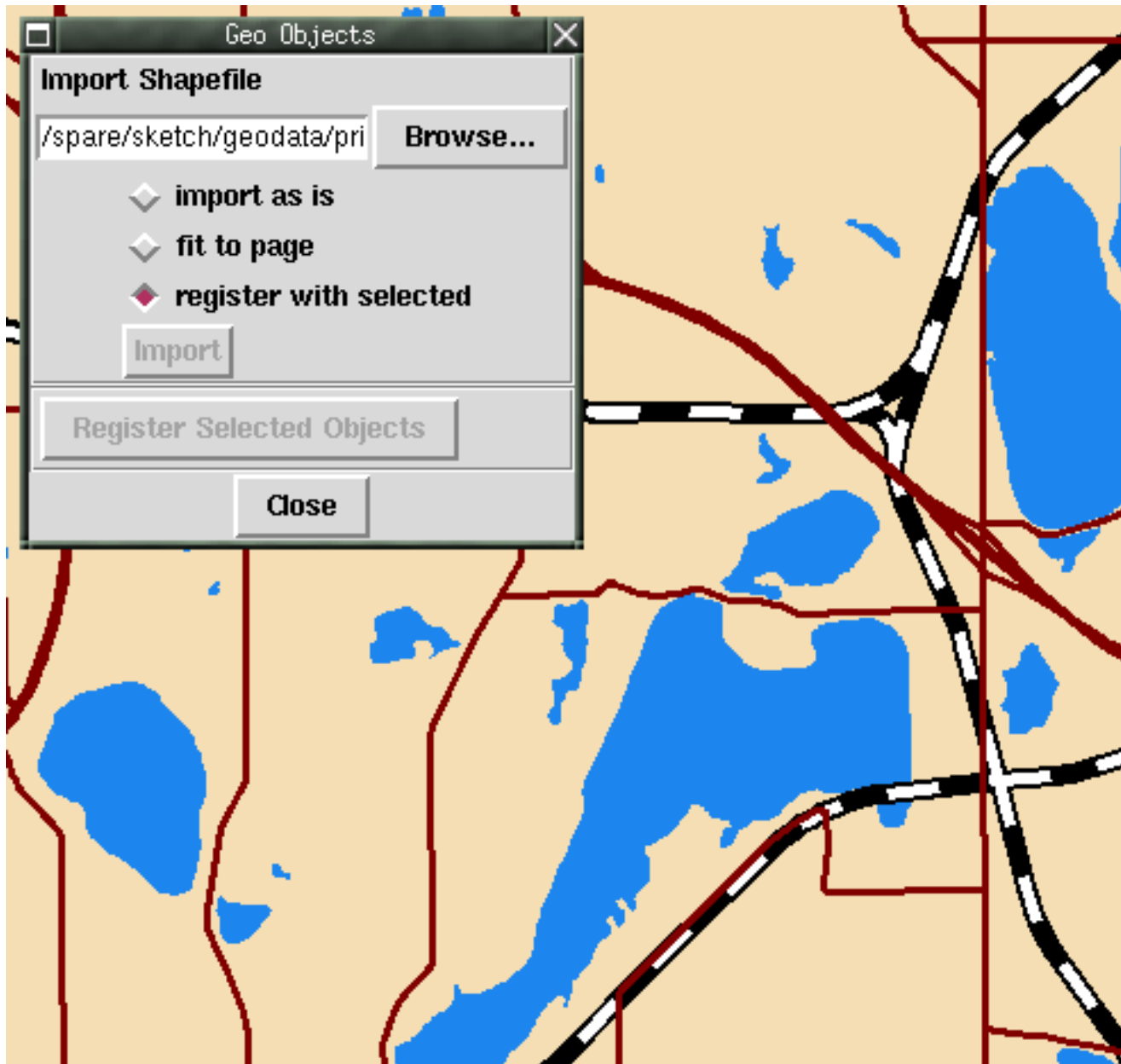


Wer sich dafür interessiert, selber Pluginobjekte zu schreiben, sollte am besten damit anfangen, sich den Quelltext zu diesem Plugin anzuschauen.

### 2.7.3. GeoObject

Wer mit geographischen Vektordaten und Sketch eine Karte produzieren möchten kann dazu das GeoObject Plugin verwenden. Es liest Shapefiles als Gruppen von Polygonen in Sketch ein. Das Ergebnis mit passenden Füllungen und Umrissen sieht dann etwa so aus (Screenshot mit dem

GeoObjects dialog; mit Demodaten vom MapServer (<http://mapserver.gis.umn.edu/index.html>):



## 2.8. Datenaustausch

Anwendungsprogramme existieren nicht in einem Vakuum, sondern müssen meist auch Daten mit anderen Programmen austauschen. Sketch unterstützt eine Reihe fremder Dateiformate, die einfach mit



den normalen Datei Öffnen und Speichern befehlen gelesen und geschrieben werden können.

Für diejenigen Anwender, die bereits seit Längerem mit Unix-ähnlichen Systemen arbeiten, ist besonders interessant, daß Sketch XFig-Dateien lesen kann. Daten aus Windows- oder Mac-Programmen konvertiert man am besten in Illustrators AI Format, Corel's CMX oder auch Windows WMF Format um sie dann in Sketch zu laden. Des weiteren unterstützt Sketch auch SVG.

Leider werden noch nicht alle Features dieser Formate unterstützt, so daß die eingelesene Grafik nicht immer so aussieht, wie in dem Originalprogramm.

Fremdformate, die Sketch schreiben kann, sind z. B. PDF (mit Hilfe von reportlab), SVG, Illustrator AI und natürlich EPS.

Daten die noch nicht in einem Format vorliegen, das Sketch lesen kann, lassen sich unter Umständen mit pstoedit oder AutoTrace konvertieren.

Pstoedit konvertiert PostScript-Dateien mit Hilfe von ghostscript in editierbare Vektorformate, darunter auch Sketch's Eigenes. AutoTrace dagegen vektorisiert Rastergrafiken und kann sie in verschiedenen Vektorformaten speichern. Es beherrscht auch Sketch's Dateiformat. AutoTrace wurde verwendet, um eine Vektorversion von Gnomes Fußabdruck für das "Fußspuren"-Skript zu erzeugen.

## 3. Skripte

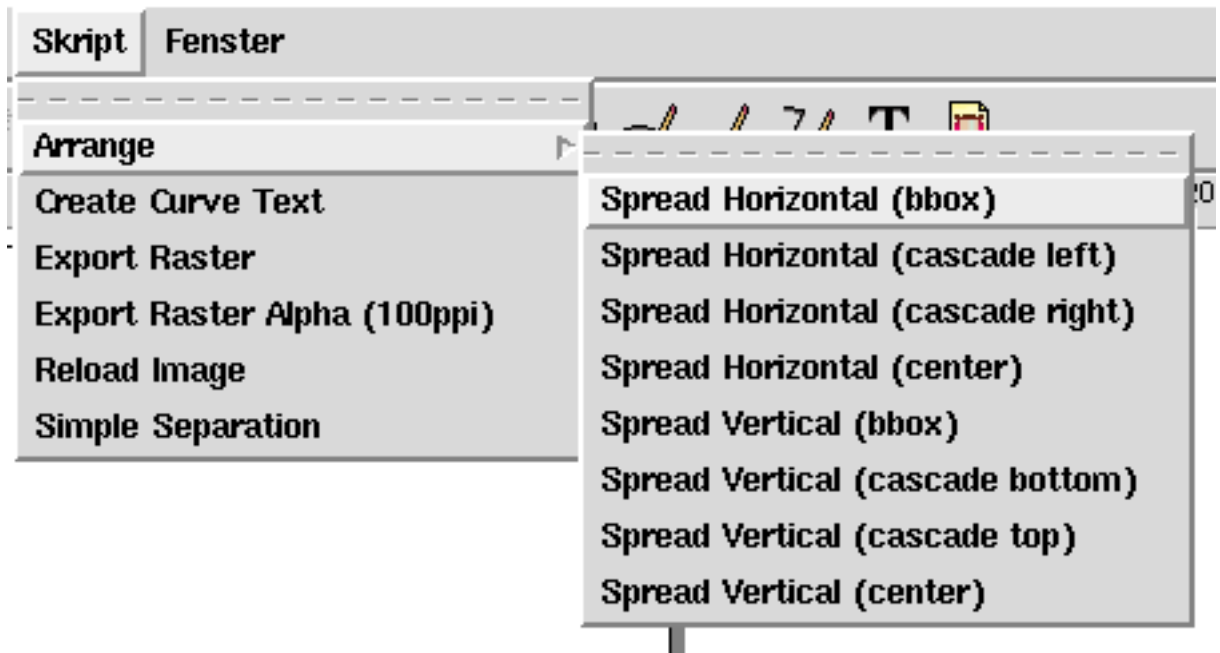
Sketch ist zum größten Teil in der Programmiersprache Python geschrieben. Python ist eine leicht zu erlernende, sehr dynamische Interpretersprache, die sich auch gut für Benutzerskripte eignet. Da Sketch selbst in Python geschrieben ist, haben Benutzerskripte fast automatisch Zugriff auf alle Sketch Interna.

In diesem Abschnitt werden wir uns anschauen, wie man Sketch durch Pythonskripte erweitert. Eine Einführung in Python würde den Rahmen dieses Artikels sprengen. Grundkenntnisse in Python werden daher vorausgesetzt.

### 3.1. Skripte Einbinden

Als erstes schauen wir uns an, wie ein Skript in Sketch eingebunden wird. Auf den Bildschirmfotos haben Sie vielleicht schon das "Skript" Menü bemerkt. Alle Benutzerdefinierten Skripte sind in diesem Menü aufgeführt.

Seit Sketch 0.6.13 enthält das "Skript" Menü bereits einige Skripte, die mit Sketch ausgeliefert werden (Beschriftung in Englisch, da diese Skripte noch nicht in die Übersetzung mit einbezogen sind):



Um ein selbstgeschriebenes Skript in das "Skript"-Menü zu bekommen nehmen wir erst einmal ein Skript das praktisch nichts tut, außer "Hallo von Sketch" in einer MessageBox auszugeben:

```
import Sketch.Scripting

def hallo(context):
    context.application.MessageBox(title = "Hallo",
                                  message = "Hallo von Sketch",
                                  buttons = ("OK",))

Sketch.Scripting.AddFunction('hallo', 'Hallo', hallo)
```

Das Modul `Sketch.Scripting` enthält den Teil von Sketch, der die Benutzerskripte verwaltet. Die Funktion `Sketch.Scripting.AddFunction` fügt die Funktion `hallo` und dem internen Namen `'hallo'` mit dem Titel `'Hallo'` in die Liste aller Benutzerskripte ein. Der Titel erscheint dann im "Skript"-Menü.

Die Funktion `hallo` selbst hat einen Parameter `context`. Wenn die Funktion von Sketch aufgerufen wird, ist `context` ein Objekt, das Referenzen auf einige andere Objekte enthält. So ist z.B. `context.application` das Applikationsobjekt, das einige globale Dinge verwaltet und hier verwendet wird, weil es über die `MessageBox`-Methode eine `MessageBox` angezeigt werden kann. Das wichtigste Objekt ist aber `context.document`, das Dokument, das heißt die Zeichnung, die gerade bearbeitet wird. Dieses werden wir später noch näher kennen lernen.

Speichern Sie diese Datei als `~/ .sketch/hallo.py`. Es ist an sich relativ egal, wo die Datei gespeichert wird, solange sie in Python's Pfad (`sys.path`) ist. Sketch fügt automatisch einige Verzeichnisse zum Pfad hinzu, unter anderem `~/ .sketch/`, so daß es ein sehr praktischer Platz für Benutzerskripte ist.

Jetzt müssen wir nur noch dafür zu sorgen, daß diese Module auch importiert wird. Das geschieht in `~/ .sketch/userhooks.py`. Diese Datei wird automatisch von Sketch beim Start importiert. Falls diese Datei noch nicht existiert, legen Sie sie einfach mit folgendem Inhalt an

```
import hallo
```

(Fügen Sie diese Zeile hinzu, falls `userhooks.py` bereits existiert).

Wenn Sie nun Sketch neu starten, sollte der Menüpunkt "Hallo" im "Skript"-Menü auftauchen. Wenn Sie ihn aufrufen, sollte in etwa folgender Dialog erscheinen:



## 3.2. Objekte Verschieben

Nun zu einem sinnvolleren Skript, das sogar etwas tut, was Sketch selbst noch nicht kann. Der "Ausrichten"-Dialog erlaubt es, alle gerade ausgewählten Objekte aneinander oder an der Seite auszurichten. Dabei werden jedoch alle Objekte einzeln verschoben, so daß die relativen Positionen der Objekte untereinander nicht erhalten bleiben. Das ist zwar meistens das, was man braucht, es ist aber auch häufig wünschenswert, alle gewählten Objekte gemeinsam zu Zentrieren, so daß die relative Anordnung erhalten bleibt. Dies soll unser Skript erreichen.

Dieses Skript muss im wesentlichen zwei Dinge kennen, um seine Arbeit erledigen zu können: den Mittelpunkt der Seite und den Mittelpunkt aller ausgewählten Objekte.

Das Dokument-Objekt, das man wie oben erwähnt als `context.document` ansprechen kann, enthält auch ein Seitenlayout, was zur Zeit allerdings nur das Seitenformat, also im Grunde die Breite und Höhe der Seite, umfaßt. Die Ausmaße der Seite bekommt man am einfachsten über die `pageSize` Methode des Dokument-Objekts. Da der Ursprung des Koordinatensystems, das Sketch verwendet in der unteren Linken Ecke ist, erhält man die Koordinaten des Seitenmittelpunkts durch halbieren der Breite bzw. Höhe:

```
def ausrichten(context):
    doc = context.document

    # Mittelpunkt der Seite
    breite, hoehe = doc.PageSize()
    mitte_x = breite / 2
    mitte_y = hoehe / 2
```

Dieses Code-Fragment ist der Anfang des Skriptes, das wir hier schreiben möchten. Da man das Dokument recht häufig braucht, ist es nützlich, es am Anfang in der lokalen Variablen `doc` abzulegen.

Der nächste Schritt ist der Mittelpunkt der Auswahl, der auch sehr einfach aus dem Dokument bestimmt werden kann. In Sketch 0.6 wird die Auswahl vom Dokument-Objekt verwaltet (anders als in 0.7, wo es ein separates Editor Objekt gibt) und es gibt eine Methode, die nur Ausmaße der Ausgewählten Objekte als "rect" zurückgibt. Ein "rect" definiert ein rechteckiges Gebiet auf der Zeichenfläche. Es hat unter anderem eine `center` Methode die seinen Mittelpunkt zurückgibt. Damit bekommen wir den Mittelpunkt der Auswahl:

```
# Mittelpunkt der Auswahl.
auswahl_x, auswahl_y = doc.SelectionBoundingRect().center()
```

Bleibt noch, die Objekte tatsächlich zu verschieben. Auch dazu hat das Dokument-Objekt bereits eine passende Methode, `TranslateSelected`, die alle ausgewählten Objekte um den angegebenen Vektor verschiebt:

```
doc.TranslateSelected(Point(mitte_x - auswahl_x,
                           mitte_y - auswahl_y))
```

Der Vektor wird durch ein `Point` Objekt gegeben. `Point` Objekte werden in Sketch an vielen Stellen verwendet, um Punkte oder Vektoren darzustellen. Sie überladen einige Operatoren, so daß man z.B. Vektoren addieren kann, oder per Indexoperator (`[ ]`) auf seine Elemente zugreifen kann. Letzteres sorgt unter anderem dafür, daß man `Point` Objekte ähnlich wie Tupel auspacken kann, was im obigen Code-Fragment auch bereits benutzt wurde, da der Rückgabewert der `center` Methode ein `Point` ist.

Der Übersicht halber das komplette Skript:

```
import Sketch.Scripting
from Sketch import Point

def ausrichten(context):
    doc = context.document

    # Mittelpunkt der Seite
    breite, hoehe = doc.PageSize()
    mitte_x = breite / 2
    mitte_y = hoehe / 2

    # Mittelpunkt der Auswahl
    auswahl_x, auswahl_y = doc.SelectionBoundingRect().center()

    # Alle ausgewählten Objekte verschieben, so daß die Mitte der Auswahl
    # auf der Seitenmitte zu liegen kommt.
```

```
doc.TranslateSelected(Point(mitte_x - auswahl_x,  
                           mitte_y - auswahl_y))  
  
Sketch.Scripting.AddFunction('ausrichten', 'Gemeinsam Ausrichten',  
                             ausrichten,  
                             script_type = Sketch.Scripting.AdvancedScript)
```

Eine wichtige Änderung gegenüber dem ersten Skript ist der vierte Parameter von `AddFunction`. Der Schlüsselwertparameter `script_type` legt den Typ des Skripts fest. Der Standardwert ist `Sketch.Scripting.SafeScript`. Ein `SafeScript` wird von Sketch mit besonderen Wrapperobjekten ausgeführt, die dem Programmierer einige Arbeit abnehmen. So muß man sich bei einem `SafeScript` zum Beispiel nicht um Undo-Informationen kümmern. Der Nachteil bei `SafeScripts` ist jedoch, daß nicht alle Methoden der Objekte zugänglich sind, was allerdings hauptsächlich daran liegt, daß nicht für alle definiert ist, ob und in welcher Weise sie in einem `SafeScript` verwendet werden können.

Der hier verwendete Skripttyp, `AdvancedScript` erlaubt dagegen Zugriff auf alle Methoden und Attribute, da keinerlei Wrapper verwendet werden. Das macht `AdvancedScripts` sehr flexibel, da sie wirklich alles in Sketch verwenden können, andererseits aber auch "gefährlicher", da man besser aufpassen muß, was man tut. So muß ein `AdvancedScript` zum Beispiel Undo-Informationen richtig handhaben. Obiges Skript braucht dazu aber nichts spezielles zu tun, da die einzige Änderung, die es am Dokument vornimmt, über die `TranslateSelected` Methode läuft, die bereits alles nötige selbst macht.

### 3.3. Fußspuren

Als drittes Skript zeichnen wir Fußspuren entlang eines Pfades. Das Skript wird als Eingabe zwei Objekte erwarten, einen Fußabdruck und den Pfad, an dem die Spuren entlang laufen sollen. Das Resultat wird dann etwa so aussehen:



Auf den ersten Blick sieht das vielleicht nach viel Rechnerei aus, da man ja aus der mathematischen Beschreibung der Kurve bestimmen muß, wie man die einzelnen Abrücke positionieren muß. Wenn man sich aber daran erinnert, daß Sketch ja auch Text an einer Kurve ausrichten kann, wird klar, daß passender Code bereits irgendwo in Sketch existieren muß und da Sketch in Python implementiert ist, kann unser Skript auch darauf zugreifen, auch dann, wenn dies von den Programmierern noch gar nicht vorgesehen war. Wir müssen nur herausfinden, wo. Leider sind die Interna von Sketch nicht sehr gut dokumentiert, so daß man nicht umhinkommt, sich etwas in den Quelltext einzulesen.

Der für uns hier wichtige Teil von Sketch, die Implementierung des Pfadtext Objektes, ist im `Sketch.Graphics.text`. Wir brauchen insbesondere die Funktion `coord_sys_at` und die Konstante `PATHTEXT_ROTATE`. Der Anfang unseres Scripts mit allen imports sieht daher zunächst einmal so aus:

```
import Sketch.Scripting
from Sketch.Graphics.text import coord_sys_at, PATHTEXT_ROTATE
from Sketch import Group, Translation, Scale
```

Eine weitere Sache, die wir entscheiden müssen, ist, wie der Benutzer definiert, welches Objekt an welchem ausgerichtet wird, also welches Objekt der Fußabdruck ist. Leider läßt sich in Sketch nicht feststellen, in welcher Reihenfolge Objekte ausgewählt wurden. Als Konvention wollen wir daher annehmen, daß genau zwei Objekte ausgewählt werden und daß das obere der beiden die Kurve ist, an der die Fußspuren entlanglaufen.

Also holen wir uns am Anfang unsere Funktion die aktuell ausgewählten Objekte mit `SelectedObjects` und prüfen, ob es genau zwei sind, und ob das Obere eine Kurve ist:

```
def foot_prints_along_path(context):
    doc = context.document
    objects = doc.SelectedObjects()

    if len(objects) == 2 and objects[1].is_curve:
```

Als erstes holen wir uns den Fußabdruck. Wir erzeugen eine Kopie des Abdrucks um das Objekt, das ja noch Teil des Dokuments ist nicht zu verändern:

```
    foot_print = objects[0].Duplicate()
```

Der Rest des Skripts ist leichter, wenn der Abdruck, der als Vorlage aller anderen Abdrücke dient am Ursprung liegt. Wir verschieben ihn also so, daß seine linke untere Ecke im Ursprung liegt. Die `Transform`-Methode verändert das Objekt. Daher war es gerade wichtig, es vorher zu kopieren:

```
    r = foot_print.coord_rect
    foot_length = r.right - r.left
    foot_print.Transform(Translation(-r.right + foot_length/2,
                                     -r.bottom))
```

Nun zum Pfad. Die `Paths` Methode eines Objekts gibt ein Tupel von Pfadobjekten zurück, sofern das Objekt dies unterstützt, was immer dann der Fall ist, wenn sein `is_curve` Attribut wahr ist:

```
    path = objects[1].Paths()[0]
```

Die Methode `arc_lengths` (Bogenlängen) liefert eine Liste mit (Länge, Punkt) paaren, wobei Punkt ein Punkt auf der Kurve ist und Länge die Länge der Kurve vom Anfangspunkt bis zu dem jeweiligen Punkt. Die Punkte sind so positioniert, daß man die Kurve zwischen ihnen als Strecke ansehen kann:

```
arc_lengths = path.arc_lengths()
```

Die eigentliche Arbeit des Scripts wird in Schleife erledigt, in der wir die Fußabdrücke nacheinander und abwechselnd links und rechts positionieren bis wir die Entfernung vom Anfang zum Ende zurückgelegt haben. Dazu brauchen wir noch ein paar Variablen, nämlich für die Gesamtlänge des Pfades (`total_length`) damit wir wissen wie weit wir gehen, dann die bereits zurückgelegte Entfernung (`distance`), einen Zähler, um zu wissen welche Fußabdrücke links und welche rechts sind, sowie eine Liste in der wir alle Fußabdrücke sammeln (`foot_prints`):

```
total_length = arc_lengths[-1][0]
distance = 0
count = 0
foot_prints = []
```

Die Schleife läuft solange wie wir mehr als eine Fußlänge vom Ende entfernt sind:

```
while total_length - distance > foot_length:
```

Bei jedem Schleifendurchlauf bestimmen wir die Abbildung, die die Vorlage `foot_print` and die richtige Position bringt, nämlich `distance` vom Anfang entfernt:

```
trafo = coord_sys_at(arc_lengths, distance, PATHTEXT_ROTATE)
```

Wenn es ein rechter Fuß sein soll (wir nehmen an, daß die Vorlage zu einem linken Fuß gehört), müssen wir erst noch an der X-Achse spiegeln:

```
if count % 2:
    trafo = trafo(Scale(1, -1))
```

Kopiere die Vorlage und transformiere sie in die richtige Stellung. Wie oben erläutert verändert `Transform` das Objekt, daher die Kopie:

```
foot = foot_print.Duplicate()
foot.Transform(trafo)
foot_prints.append(foot)
```

Am Ende jedes Durchlaufs noch die zurückgelegte Entfernung und den Zähler aktualisieren:

```
distance += foot_length
count += 1
```

Wenn die Schleife fertig ist, haben wir alle Fußabdrücke in der Liste. Wir müssen diese Objekte jetzt nur noch als Gruppe in das Dokument einfügen. Bei sehr kurzen Kurve, oder langen Füßen, kann es passieren, daß gar keine Objekte Positioniert wurden, daher Erzeugen wir nur dann eine Gruppe, wenn überhaupt Objekte da sind:

```
if foot_prints:
    doc.Insert(Group(foot_prints))
```

Nun noch einmal das komplette Script. Hier sind auch noch einige fehlende imports enthalten:

```
import Sketch.Scripting
from Sketch.Graphics.text import coord_sys_at, PATHTEXT_ROTATE
from Sketch import Group, Translation, Scale

def foot_prints_along_path(context):
    doc = context.document
    objects = doc.SelectedObjects()

    if len(objects) == 2 and objects[1].is_curve:
        foot_print = objects[0].Duplicate()

        r = foot_print.coord_rect
        foot_length = r.right - r.left
        foot_print.Transform(Translation(-r.right + foot_length/2,
                                         -r.bottom))

        path = objects[1].Paths()[0]
        arc_lengths = path.arc_lengths()

        # In der folgenden Schleife positionieren wir nacheinander die
        # Fußabdrücke abwechselnd links und rechts bis wir die
        # Entfernung vom Anfang zum Ende zurückgelegt haben.

        total_length = arc_lengths[-1][0]
        distance = 0
        count = 0
```



```
foot_prints = []
while total_length - distance > foot_length:
    trafo = coord_sys_at(arc_lengths, distance, PATHTEXT_ROTATE)
    if count % 2:
        trafo = trafo(Scale(1, -1))
    foot = foot_print.Duplicate()
    foot.Transform(trafo)
    foot_prints.append(foot)

    distance += foot_length
    count += 1

# Zum Schluß fügen wir die Fußabdrücke als Gruppe in das
# Dokument ein. Die Insert-Methode kümmert sich automatisch um
# Undo-Informationen.
if foot_prints:
    doc.Insert(Group(foot_prints))

Sketch.Scripting.AddFunction('foot_prints_along_path', 'Foot Prints',
                             foot_prints_along_path,
                             script_type = Sketch.Scripting.AdvancedScript)
```

Wenn alles klappt, müsste man jetzt z.B. mit diesen beiden Objekten



zu diesem Ergebnis kommen



### 3.3.1. Erweiterungsmöglichkeiten

Dieses Skript, das ja eigentlich gar nicht auf Fußabdrücke beschränkt ist, ließe sich noch viele Erweiterungen zu. So könnte man z.B. die Größen oder Positionen der Objekte zufallsgesteuert ein wenig variieren.

Man könnte das ganze auch noch zu einem Plugin-Objekt erweitern und damit dafür sorgen, daß die Fußspuren automatisch aktualisiert werden, sobald man die Kurve bearbeitet.

## **4. Ausblick auf Sketch 0.7**

Der augenfälligste Unterschied zwischen den stabilen 0.6er Versionen und den 0.7er Entwicklerversionen ist, daß Sketch 0.7 auf GTK statt auf Tk als GUI-Toolkit aufsetzt und ein Gimp ähnliches Benutzerinterface hat mit einem Werkzeugkasten (im Bild oben links) und einem Hauptfenster

für jede geöffnete Zeichnung:



Weitere Neuigkeiten sind:

- Kantenglättung mit Libart (optional). Einige der Beispielgrafiken in diesem Artikel sind Screenshots der Entwicklerversion.
- Freihandwerkzeug. Die so gemalten Linien werden durch Bezierkurven approximiert
- Metadaten. Mit jedem Objekt können Metadaten in Form von Schlüsselwort/Wert-Paaren verknüpft werden.

## A. Weiterführende Informationen

### Webseiten

Sketch Heimatseite

Die Heimatseite (<http://sketch.sourceforge.net/>) von Sketch ist der Hauptanlaufpunkt für Informationen über Sketch und zum Herunterladen des Programms selbst.

Sketch Webseiten bei LinuxGraphic.org (französisch)

Auf der französischen Webseite über Sketch (<http://www.linuxgraphic.org/section2d/sketch/index.html>) gibt es ein recht komplettes Benutzerhandbuch zu Sketch, sowie eine sehr ansehnliche Clipart-Sammlung (<http://www.linuxgraphic.org/section2d/sketch/cliparts/index.html>). Die Cliparts sind auch für Leute interessant, die des Französischen nicht mächtig sind.

Artikel über Sketch

- LinuxFocus, November 1999 (<http://www.linuxfocus.org/Deutsch/November1999/article120.html>). Yves Ceccone beschreibt das Arbeiten mit Sketch anhand der Zeichnung einer Diskette und demonstriert einige Spezialeffekte.
- LinuxUser, September 2001 (<http://www.linux-user.de/ausgabe/2001/09/040-vektorgrafik/vektorzeichenprogramme-3.html>). Frank Wieduwilt vergleicht einige Vektorzeichenprogramme für GNU/Linux. Fazit:

Sketch ist dem Ziel, ein universelles Zeichenprogramm für Linux zu werden, schon am nächsten, sodass man der weiteren Entwicklung gespannt harren darf.

Python

Die Programmiersprache Python (<http://www.python.org/>) ist eine sehr flexible, interpretierte Sprache, die sich zudem noch sehr leicht durch in C-geschriebene Module erweitern läßt.

## ReportLab

ReportLab (<http://reportlab.com/>) ist eine Firma, deren Hauptprodukt eine Python-Bibliothek gleichen Namens ist, die PDF erzeugen kann. ReportLab ist freie Software.

## pstoedit

pstoedit (<http://www.pstoedit.net/>) ist ein Programm das mit Hilfe von ghostscript PostScript-Dateien in eine reihe von Vektorformaten konvertieren kann, darunter auch Sketch's eigenes .sk-Format.

## AutoTrace

Das Programm AutoTrace (<http://autotrace.sourceforge.net/>) vektorisiert Rastergrafiken und kann sie in verschiedenen Vektorformaten speichern. Es beherrscht auch Sketch's Dateiformat. AutoTrace wurde verwendet, um eine Vektorversion von Gnomes Fußabdruck für das "Fußspuren"-Skript zu erzeugen.