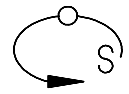
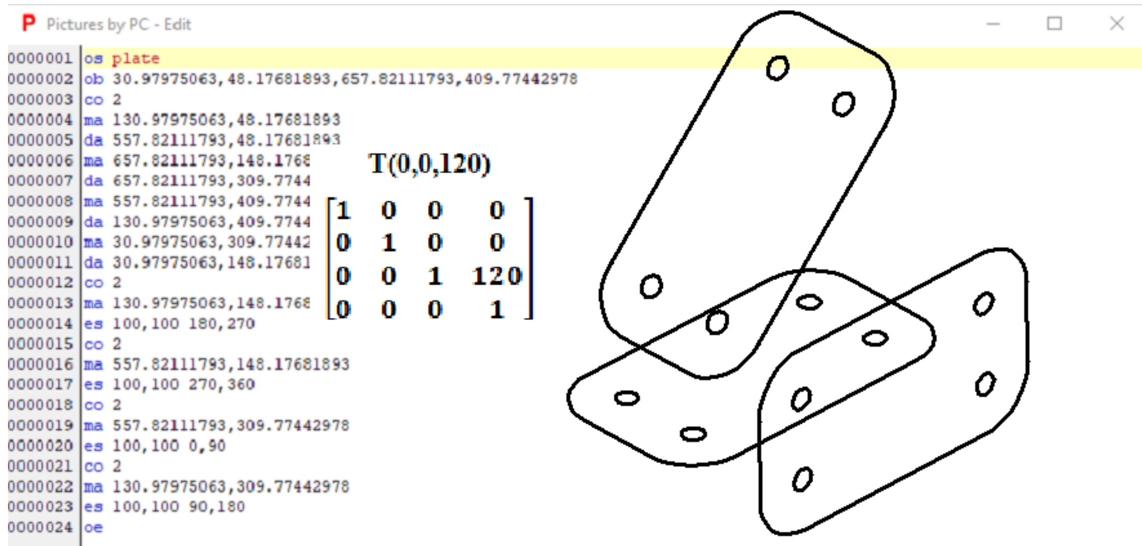


Zugriff auf die Puffer-internen Vektordaten



Eine interessante Besonderheit von **Pictures by PC** ist seine offene Vektor-Geometrie. Zeichnungsdaten werden im Klartext-Format gespeichert, sind einfach zu interpretieren und lassen sich auch nach der interaktiven Zeichnungserstellung noch direkt im Speicher manipulieren. Allerdings sollte Letzteres nur von geübten Bedienern vorgenommen werden, da dadurch Zeichnungsdaten unwiederbringlich zerstört werden könnten. Kritische Daten müssen also unbedingt vor einer solchen Manipulation gesichert werden.

Warum wurde dieses offene-Vektor-Klartext-Konzept für **Pictures** gewählt? Ganz einfach deshalb, dass "Zeichnungen" von jedem externen, normalen Texteditor geschrieben, editiert und manipuliert werden könnten. Transparenter und einfacher können Geometriedaten nicht verwaltet werden. Zudem erleichtert es einen einfachen, bidirektionaler Geometrie-Datenaustausch.

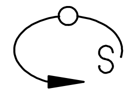
An dieser Stelle wollen wir uns aber nicht mit dem CAD-Datenaustausch befassen (dafür dient normalerweise das Exchange-Package), sondern wir wollen die Vorteile des offenen Vektor-Formats für die Programmierung und Automatisierung nutzen. Dafür gibt es unter **Pictures** einige hilfreiche Werkzeuge und Methoden, die wir im Folgenden näher erläutern wollen.

Wenn wir mit dem Kommando

```
poly * -p0,0..1000,800
```

eine einfache zweidimensionale Gerade erzeugen, kann man mit dem Editor durch

```
edit *
```



den Pufferinhalt (Speicher) mit seinen Vektordaten (Zeichnungsbefehlen) sichtbar machen. Man erkennt folgende Zeilen

Pictures by PC - Edit

```
0000001 os obj1
0000002 ob 0,0,1000,800
0000003 co 2
0000004 ma 0,0
0000005 da 1000,800
0000006 oe
```

Das sind die speicherinternen Vektordaten. Die zwei, blauen Buchstaben am Zeilenanfang stellen die Zeichnungsbefehle dar, die in der Pictures-Hilfe (s.u.) genau beschrieben werden. Wir wollen aber an dieser Stelle schon vorab die oben dargestellten 6 Zeilen einmal kurz erläutern.

In Zeile 1 beginnend wird ein neues Zeichnungsobjekt definiert. "os" steht für "object start" und der Objektname lautet "obj1". Ein solches Objekt muss immer mit "oe" für "object end" abgeschlossen werden (Zeile 6). Zwischen "os" und "oe" wird die Geometrie der "Zeichnung" mit Vektorbefehlen und Koordinaten (bzw. Parametern) definiert (beschrieben). Für unseren Fall heißt das im Einzelnen.

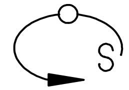
Zeile 2 beschreibt eine Objektbox ("ob"). Das ist ein Parameter für die "Abmessung" des Objekts. Genau genommen beschreibt "ob" einen das Objekt umschreibenden Fensterbereich mit der linken, unteren Koordinate (0,0) (bzw. x=0, y=0) und der rechten, oberen Fensterecke (1000, 800) (bzw. x=1000, y=800). Sinnvollerweise wird die Objekt-Box im allgemeinen vom Programm automatisch z.B. per box-Kommando (s.u.) berechnet. Bei der "manuellen" Erzeugung kann "ob" also getrost weggelassen werden.

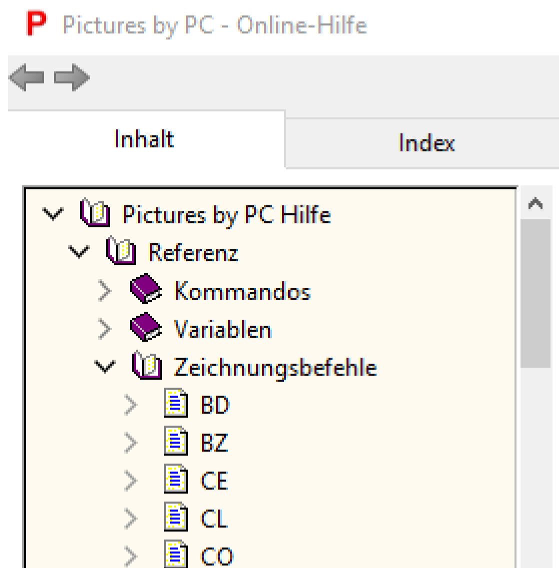
Bemerkung: Wozu ist dann "ob" überhaupt sinnvoll? Z.B. ist es für ein schnelleres "Zooming" hilfreich, nur die Objekt-Box zu analysieren, um zu prüfen, ob ein Objekt innerhalb des Fensters liegt und angezeigt werden soll oder nicht.

Betrachten wir die Editorzeile 3. Diese besagt, dass mit dem Farbbefehl "co" (color) im Folgenden die Farbe 2 (also Grün) genutzt wird.

"ma 0,0" in Zeile 4 ("ma"=move absolute) besagt, dass der "Vektorgenerator" mit "gehobenem, virtuellen Stift" auf die Koordinate (0,0) positioniert wird. "da 1000,800" ("da"=draw absolute) bedeutet dann, dass jetzt mit "abgesenktem Stift" ein Strich bis zur Koordinate x=1000, y=800 erzeugt wird.

Die "Zeichenmaschine" von Pictures besteht also aus einer "Plotter-ähnlichen-Vektor-Befehls-Sprache". Details dazu findet man unter der <F1>-Hilfe in alphabetischer Auflistung.





Schließen wir jetzt den Puffer-Editor zunächst wieder mit der <ESC>-Taste und sichern die "Zeichnung" mit

```
saveas "test.vec" -p
```

In dieser Datei liegen die Daten genauso im Klartext-Format vor, wie oben (allerdings ergänzt durch einige, für uns momentan nicht relevante "oi"-Preview-Daten s.u.).

test.vec

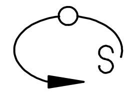
Datei Bearbeiten Ansicht Extras ?

test.vec

```
0001 oi preview.data.8=2120,iJ8udVuchrBgSF+
0002 oi preview.data.7=1772,6DF9j0RsCWeWFFv
0003 oi preview.data.6=1424,CjEmwStG0dXf7Ls
0004 oi preview.data.5=1076,wAo3MxJOAABk9AK
0005 oi preview.data.4=728,Pa6VZS3kkcZmmZBh
0006 oi preview.data.3=380,6Onq8fLz9PX29/j5
0007 oi preview.data.2=32,AwMDBAMDBAUIBQUEB
0008 oi preview.data.1=0,/9j/4AAQSkZJRgABAQ
0009 os obj1
0010 ob 0,0,1000,800
0011 co 2
0012 ma 0,0
0013 da 1000,800
0014 oe
```

Wenden wir uns jetzt der eigentlichen Idee dieser Lektion zu. Unsere derzeit im Puffer befindlichen Daten unserer "fertigen Zeichnung" können wir auch nachträglich noch manipulieren (Vorsicht: leider auch zerstören). Dazu gibt es **drei Methoden**.

Die erste besteht darin die Daten mit dem Puffer-Editor anzuzeigen (edit) und die entsprechenden Zeichnungsbefehle per Tastatur manuell zu ändern.



Die **zweite Methode** besteht in der Manipulation der Daten mit dem text-Kommando, einem per Kommando steuerbaren "Texteditor" (der auch schon einmal zu einer "reichlich exotischen", einzeiligen "Programmiersprache" ausarten kann). Die Handhabung der text-Kommandos werden wir im Folgenden als erstes aufgreifen.

Die **dritte Methode** nutzt den direkten Puffer-Zugriff mit Hilfe der BIX-Programmierung. Die komplette Kenntnis dazu lässt sich allerdings an dieser Stelle nicht vermitteln, sondern ist Inhalt eines "echten" Programmierseminars. Trotzdem wollen wir in dieser Lektion einen kleinen Einblick mit einige einfachen Sequenzen vermitteln.

Wir wollen aber zuerst die zweite Methode benutzen. So sollen also per text-Kommando (<F1>-Hilfe, Kommandos, "Text") die Änderungen Kommando-orientiert vorgenommen werden. Wie schon gesagt, ist "Text" ein per Kommando steuerbarer Editor, mit dem sich Manipulationen vornehmen lassen, als würde man sie per Tastatur manuell mit einem Editor eingegeben haben. Lassen sie uns das schrittweise austesten

Machen wir dazu eine kleine Vorübung. Geben Sie folgendes Kommando in die Pictures-Kommandozeile ein

```
text g1:1 fco vline ; echo $textpos ; echo $line
```

so erhalten als Ergebnis z.B.

```
1:11
co 2
```

Dieses muss erklärt werden. Prinzipiell greifen wir mit dem zeilenorientierten text-Kommando (einem per Kommando steuerbaren Editor) auf die internen, im Puffer vorhandenen Daten (also unsere oben erzeugte Linie) zu. Im text-Kommando selbst gibt es auch wiederum Kommandos, die ihrerseits mit einem Kennbuchstaben eingeleitet werden. Mehrere solcher Kommandos werden in einer Zeile durch Leerzeichen (Space) getrennt.
Werfen wir also einen Blick auf das obige text-Kommando. Dann besagt das. Gehe ("g" für "go") mit dem (unsichtbaren) Cursor in den 1. Puffer in die 1. Zeile, suche ("f" für "find") den Zeichnungsbefehl "co" ("co" für "color") und speichere den Zeileninhalt in eine Variable "line" ("v" für "Variable", "line" als (willkürlichen) Variablennamen). Danach werden mit dem "echo"-Kommando der Inhalt der Standard-Variablen TEXTPOS bzw. unserer selbstdefinierten Variablen "line" ausgegeben. Dabei enthält die Variable TEXTPOS die Puffer-Nummer (hier: 1:) und die aktuelle Zeilennummer (in der der co-Befehl gefunden wurde (hier: 11)), während in der Variablen "line" der Inhalt der kompletten Editor-Zeile gespeichert wurde (hier: "co 2", also Farbe 2=Grün).

Nach dieser kleinen Vorübung wollen wir jetzt den im Puffer vorhandenen Strich (obj1, unserer "Zeichnung") auch nachträglich noch ändern. Es soll z.B. die Volllinie auf den Linientyp "255,25" und die Farbe Grün ("co 2") auf Rot ("co 3") umgefärbt, aber zusätzlich auch noch eine Linienstärke von 3 mm hinzugefügt werden ("co 3 3"). Natürlich könnte man das mit den Kommandos "chglt obj1 255,25" bzw. "color obj1 3,3" auch auf "klassische Weise" erreichen. Wir wollen aber die Manipulation direkt im Puffer vornehmen, als würden wir mit "edit " den Puffer öffnen und manuell per Tastatur die Zeile "co 2" auf "co 3 3" ändern und davor die Zeile "lt 255,25" einfügen. Das entspräche der manuellen Methode 1. Nach dem Schließen des Editors (<ESC>)

und nach einem Bildneuaufbau würde der Strich rot, aber auch noch mit gestrichelten Linientyp und Linienstärke, angezeigt

Das wollen wir jetzt mit folgendem Kommando erreichen.

```
text g1:1 fco "ilt 255,25" fco d "ico 3 3"
```

In Worten heißt das. Gehe ("g" für "go") mit dem (unsichtbaren) Cursor in den 1. Puffer in die 1. Zeile, suche ("f" für "find") den Zeichnungsbefehl "co" ("co" für "color"), füge davor den Linientyp "lt 255,25" ein, finde abermals "co", lösche ("d" für "delete") die aktuelle Zeile und füge ("i" für "insert") an dieser Stelle den Zeichnungsbefehl "co 3 3" ein. Dabei sind übrigens die Insert-Befehle einschließlich des "i" von Ausrufezeichen ("") umklammert, weil die Ausdrücke "lt 255,25" bzw. "co 3 3" Leerzeichen enthalten.

Mit obigem text-Kommando wurden so die internen Daten unseres Strichs wie folgt geändert.

```
os obj1
ob 0,0,1000,800
lt 255,25
co 3 3
ma 0,0
da 1000,800
oe
```

Nachdem Sie das text-Kommando oben eingegeben haben, wurde es zwar bereits ausgeführt, zeigt aber scheinbar noch keine Veränderung. Diese wird erst nach einem Bildneuaufbau (z.B. redraw) oder besser noch nach Eingabe von z.B.

```
bix ToggleShowLineWidth
```

wirksam. Mit Letzterem werden zusätzlich noch die Linienstärken auf dem Bildschirm dargestellt. Diese Anzeige kann übrigens wahlweise ein- und ausgeschaltet werden (Toggle-Modus)

Das oben genutzte Kommando funktionierte ja bereits ganz ordentlich, ist so aber nicht verallgemeinerbar. Erzeugten wir also nach unserem Strich noch ein weiteres, grünes Objekt, so würde dieses anstelle des Strichs umgefärbt. Außerdem funktionierte unser Kommando nur im Puffer 1. Deutlich präziser anstelle des obigen wäre also folgendes Kommando (ggf. vorher undo oder "open test")

```
text g$[modbuf]1 fco "ilt 255,25" "r\co 2\co 3 3\" -o obj1
```

und anschließendem Bildneuaufbau.

Oder sofern "obj1" das aktive Objekt wäre, könnte auch diese Variante genutzt werden

```
text g$[modbuf]1 fco "ilt 255,25" "r\co 2\co 3 3\" -o *
```

Bei den letzten beiden Kommandos wurde anstelle des Puffers 1: die Standard-Variable MODBUF für den aktuellen Puffer genutzt. Die Kommandos "d" und "i" wurden durch ein "r" (für "replace") mit einem Separator "\" ersetzt. Und mit der Option "-o" wurde die Ersetzung nur gezielt auf das Objekt angewandt.

Sie ahnen sicherlich schon, dass man mit dem Text-Kommando erstaunliche Manipulationen vornehmen kann. Man könnte es vielleicht sogar als die "kompakteste, einzeilige Programmiersprache" bezeichnen.

Die oben vorgenommenen Manipulationen zur Änderungen von Linientyp, Farbe und Linienstärke hätten, wie ja oben schon erwähnt, auch per "normaler" Kommandoeingabe erzielt werden können. Deshalb wollen wir ergänzend jetzt einmal ein völlig neues Objekt (hier eine Ellipse) direkt im Puffer erzeugen. Wenn Sie folgendes Kommando eingeben

```
text g$[modbuf]1 t4 fos "ios obj2" "ico 3" "ima 500,400" "iel 100,75" "ioe"
```

und anschließend

```
box all ; redraw
```

eingeben. So haben Sie gerade eine rote ("co 3") Ellipse ("el 100,75") mit einem x-Radius von 100 bzw. einem y-Radius von 75 an der Position 500,400 ("ma 500,400") als Objekt "obj2" erzeugt. Die beiden Kommandos "t4" und "fos" "überspringen" wir an dieser Stelle zunächst einfach einmal (Erklärung später). Sie dienen nur dazu, das neue Objekt an der richtigen Position im Puffer einzufügen (hinter ggf. schon vorhandene oi-Preview-Daten)

```
0000007 oi preview.data.2=32,AwMDBA
0000008 oi preview.data.1=0,/9j/4AA
0000009 os obj2
0000010 ob 400,325,600,475
0000011 co 3
0000012 ma 500,400
0000013 el 100,75
```

Mit dem text-Kommando lassen sich also erstaunliche Manipulationen vornehmen. Um das zu nutzen, bedarf es lediglich der Kenntnis der entsprechenden Zeichnungsbefehle (s.o. Abb.-Hilfe).

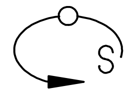
Nur am Rande sollte an dieser Stelle auch die 3. Methode der Manipulation der Zeichnungsbefehle mittels des in Pictures integrierten Basics (BIX) erwähnt werden, ohne dass wir näher darauf eingehen wollen. Eine entsprechende Basic-Anweisung, die obige Ellipse ebenfalls erzeugen würde, lautete dann z.B. (ggf. zuvor wieder "undo" oder "open test")

```
bix -l vars ; bix set g1=New BufferNewObject :
g1.Insert "co 3" : g1.Insert "ma 500,400": g1.Insert
"el 100,75" : g1.CreateObject "*", opts:="AB"
```

Die darin enthaltenen Zeichnungsbefehle kennen Sie ja bereits. Hierbei wird der Objektname hochzählend automatisch erzeugt (obj...), das Objekt aktiviert und die Objektbox berechnet (Optionen "AB")

Die gleiche Wirkung hätte man jedoch auch mit dem klassischen Kommando

```
arc * -tc500,400 -r100,75
```



erzielen können. Daher wollen wir einmal einen einfachen Fall herausgreifen, bei dem der direkte Zugriff auf die Vektor-Daten gegenüber einem "normalen" Kommando deutliche Vorteile bietet. Wenn wir z.B. mit der Kommandosequenz

```
set fillet=100 ; recta plate -f
```

interaktiv ein großes, ausgerundetes Rechteck mit einem Eckenradius von 100 im Pictures-Anzeigefenster manuell durch die Selektion zweier diagonaler "Eckpunkte" erzeugen, ist das sicherlich nicht sonderlich beeindruckend. Will man aber aus diesem Objekt "plate" möglichst schnell ein Schild mit Befestigungslöchern machen, ist das aufgrund der unbekannten Koordinaten und Abmessungen interaktiv nicht mehr so einfach machbar.

Ganz einfach ist das jedoch mit dem Text-Kommando, sofern man die Zeichenbefehle etwas besser kennt. Sehen wir uns das gerade erzeugte Objekt mit

```
edit *
```

zuerst einmal im Editor an. Dann sehen die Daten etwa so aus

```
P Pictures by PC - Edit
0000020 oe
0000021 os plate
0000022 ob -1.95653317,101.42891445,1005.5989785,711.97747824
0000023 co 2
0000024 ma 98.04346683,101.42891445
0000025 da 905.5989785,101.42891445
0000026 ma 1005.5989785,201.42891445
0000027 da 1005.5989785,611.97747824
0000028 ma 905.5989785,711.97747824
0000029 da 98.04346683,711.97747824
0000030 ma -1.95653317,611.97747824
0000031 da -1.95653317,201.42891445
0000032 co 2
0000033 ma 98.04346683,201.42891445
0000034 es 100,100 180,270
0000035 co 2
0000036 ma 905.5989785,201.42891445
0000037 es 100,100 270,360
0000038 co 2
0000039 ma 905.5989785,611.97747824
0000040 es 100,100 0,90
0000041 co 2
0000042 ma 98.04346683,611.97747824
0000043 es 100,100 90,180
0000044 oe
```

Uns interessieren nur die Befehle für die Teilkreise ("es" für Ellipsen-Segmente). Z.B. bedeutet "es 100,100 180,270" dass es sich um einen linksdrehenden Viertelkreis mit einem Radius von 100 handelt, dessen Anfangswinkel bei 180° beginnt und mit einem Endwinkel von 270° endet. Die zugehörige Mittelpunktkoordinate findet man in der "ma"-Zeile davor. Wenn wir also mit dem text-Kommando die Mittelpunktkoordinaten duplizieren und mit einem "el 20,20"-Befehl ergänzen, erhalten wir die gewünschten Befestigungslöcher mit einem Radius von 20.

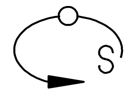
Geben Sie also folgendes Kommando ein

```
text g$[modbuf]1 t99 fes -1 pparam +2 "ima $$param" "iel 20,20" l3 -s -o * ; redraw
```

so erzielen Sie den gewünschten Effekt. Die geänderten Daten im Bereich der eingefügten Kreise erkennen Sie in der Abbildung unten.

```
0000031 da -1.95653317,201.42891445
0000032 co 2
0000033 ma 98.04346683,201.42891445
0000034 es 100,100 180,270
0000035 ma 98.04346683,201.42891445
0000036 el 20,20
0000037 co 2
0000038 ma 905.5989785,201.42891445
0000039 es 100,100 270,360
0000040 ma 905.5989785,201.42891445
0000041 el 20,20
0000042 co 2
0000043 ma 905.5989785,611.97747824
0000044 es 100,100 0,90
0000045 ma 905.5989785,611.97747824
0000046 el 20,20
0000047 co 2
0000048 ma 98.04346683,611.97747824
0000049 es 100,100 90,180
0000050 ma 98.04346683,611.97747824
0000051 el 20,20
0000052 oe
```

Das text-Kommando oben ist aufgrund der Wiederholungen fast schon ein kleines, einzeiliges Programm. Dazu hier eine kurze Erklärung. Zunächst setzten wir den "unsichtbaren Cursor" in die 1. Zeile im aktuellen Puffer ("g$[modbuf]1"). Mit "t99" ("t" für "trap") stoßen wir auf einen bislang unbekannten Zeichnungsbefehl. Dieser dient u.a. der Unterdrückung von Fehlermeldungen im text-Kommando. Im Klartext lautet der Befehl in etwa so. Sofern das text-Kommando z.B. durch eine Mehrfachsuche (oder Fehlfunktion) das Ende des Puffers erreicht, soll es mit der 99. Anweisung in der Text-Kommandozeile (die es in dieser Zeile allerdings nicht gibt) fortfahren. In der Praxis heißt das, dass eine eventuelle Fehlermeldung unterdrückt würde. Die Anweisung "fes" besagt, dass nach dem "es"-Befehl gesucht wird. Mit "-1" (minus 1 ; früher '-1) wird die vorherigen Zeile angesprochen und deren Parameterwerte werden mit "pparam" ("p" für Parameter und "param" für den Namen einer Variablen, die die Zentrumskoordinaten des Kreis-(Ellipsen-)Segments enthält) gespeichert. Danach springt man mit "+2" zwei Zeilen weiter (vor das "es") und fügt dort mit "ima $$param" das Duplikat der Zentrumskoordinate aus der Variablen param und ebenfalls noch mit "iel 20,20" den Kreisradius ein. Durch "l3" ("l" für "loop") wird erreicht, dass auf die dritte Anweisung in der Text-Kommando-Zeile, also "fes", zurückgesprungen wird, um in einer Schleife auch noch die übrigen Kreise zu erzeugen. Die Option -s (für "substitute") bewirkt, das der Inhalt der Variablen ($param) in der text-Anweisung



selbst noch einmal substituiert wird (daher das doppelte $-Zeichen). Mit der Option "-o *" werden, wie Sie ja wissen, die Manipulationen nur auf das aktive Objekt angewandt.

Die Kommandozeile wirkt im ersten Moment vielleicht ein wenig komplex, folgt aber einer einfachen Logik und kann ja Schritt für Schritt ausgetestet werden.

Zweifellos ist aber in diesem Fall die Methode, die Kreise mit dem Text-Kommando zu erzeugen, der Konstruktion mittels interaktiver Kommandos ganz deutlich überlegen. Stellen Sie sich beispielsweise eine komplexe Bohrplatte mit Hunderten von Bohrlöchern vor, deren Durchmesser Sie ändern müssten. Mit dem Text-Kommando wäre das blitzartig gelöst. Apropos, wollten wir die Radien der soeben erzeugten Kreise auch nachträglich ändern, muss das text-Kommando, da man ja keine neuen Kreise erzeugen will, natürlich anders lauten, z.B.

```
rad=50 ; text g$[modbuf]1 t99 fel d "iel $rad,$rad" l3 -o * ; redraw
```

Hierbei wurde übrigens die Option -s (für "substitute") nicht benötigt, da die Variable "rad" schon außerhalb des text-Kommandos definiert wurde.

Wie Sie sehen, lohnt es sich also, die wichtigsten Text-Kommandos und Zeichenbefehle näher kennenzulernen.

Lassen Sie uns noch zum Abschluss das Text-Kommando dazu nutzen, unser Objekt "plate", das ja zweidimensional ist, in die 2.5D-Welt zu überführen, allerdings auch mit dem text-Kommando und nicht wie sonst üblich interaktiv. Aufgabenstellung sei es, unsere Platte auf eine Z-parallele 2.5D-Ebene der Höhe Z=120 zu verschieben.

Wenn wir das direkt in den Vektor-Daten bewerkstelligen wollen, müssen wir uns zuvor ein wenig mit Matrix-Operationen beschäftigen. Das ist insofern auch wichtig, als man bei der Programmierung geometrischer Manipulationen praktisch nicht ohne diese auskommt.

Grundsätzlich lassen sich die üblichen Manipulationen wie Verschieben, Skalieren, Rotieren, Spiegeln oder auch Scheren im zwei- oder dreidimensionalen Raum mit Matrix-Operationen mit 4 x 4 Matrizen beschreiben. Beginnen wir mit unserer einfachen Aufgabenstellung, der Verschiebung der Z-Ebene auf die (willkürlich) geforderte Höhe von 120.

Eine allgemeine Verschiebungs-Matrix ("T" für "Translation") sieht wie folgt aus

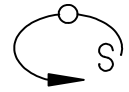
T(tx,ty,tz)

$$\begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

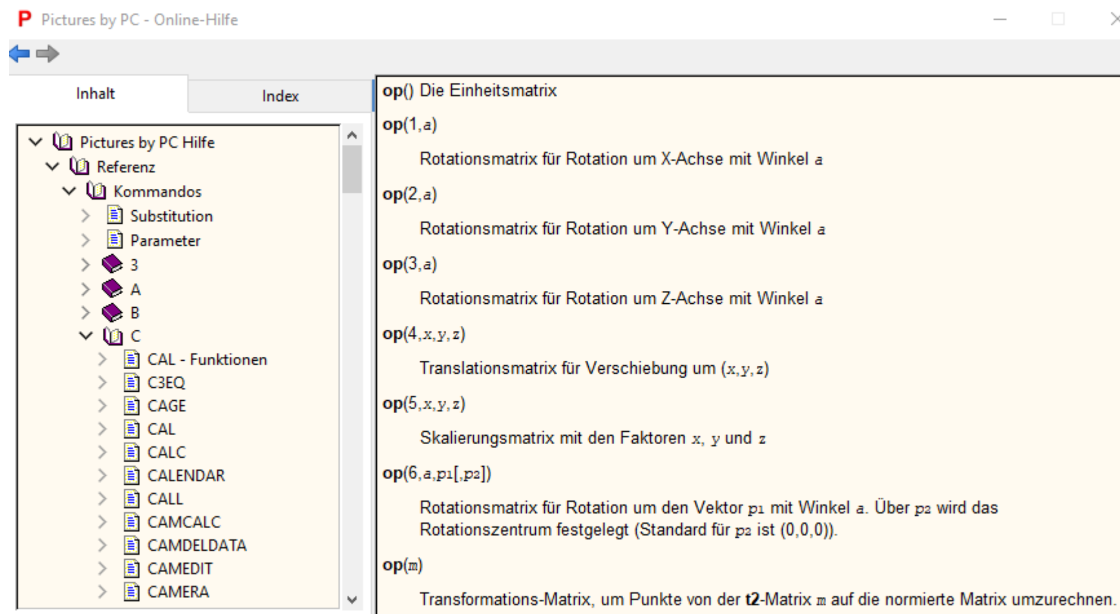
In unserem Fall wollen wir nur auf z=120 verschieben, also lautet die Matrix

T(0,0,120)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 120 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Innerhalb von **Pictures by PC** können solche Matrizen leicht mit der cal-Funktion **op** berechnete werden.



Uns interessiert also im Moment nur die Funktion "op(4,0,0,120)", die wir mit dem Kommando "val" berechnen können.

```
val op(4,0,0,120)
```

Als Ergebnis erhalten wir

```
op(4,0,0,120) = (1,0,0,0,0,1,0,0,0,0,1,120,0,0,0,1)
```

Die 16 Zahlen (4 x 4) auf der rechten Seite der Gleichung (ohne Klammern) stellen jetzt die Transformationsmatrix für eine Verschiebung auf z=120 dar. (Denken Sie daran, dass Sie die Zahlen nicht eintippen, sondern in der Infozeile mit <Strg>+<m> und den Pfeiltasten markieren und ins Clipboard übernehmen sollten)

Der Vollständigkeit halber sollten wir natürlich auch noch darauf hinweisen, dass auch im Basic alle Matrix-Operationen ebenfalls zur Verfügung stehen, allerdings einer anderen Syntax folgen. In BIX würde die Verschiebung auf Z=120 berechnet mit

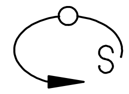
```
bix print op("T",0,0,120)
```

Wenn wir diesen 16 Zahlen jetzt einen "t2"-Zeichnungsbefehl voranstellen und diese Zeile in die Pufferdaten in das Objekt "plate" hinter die "ob"-Zeile (also vor "co") einfügen, haben wir unser 2D-Objekt "plate" in die 2.5D-Welt überführt. Das Text-Kommando, dass das bewirkt, lautet z.B.

```
text g$[modbuf]l "sos plate" fco "it2 1,0,0,0,0,1,0,0,0,0,1,120,0,0,0,1" ; box plate
```

Das bedeutet, das plane Objekt "plate" liegt jetzt im dreidimensionalen Raum auf der Höhe Z=120. Das box-Kommando im Ausdruck oben berechnet die Objekt-Box (interner "ob"-Befehl) neu. Das ist notwendig.

Sichtbar wird die Transformation erst, wenn z.B. die isometrische Sicht eingeschaltet wird



```
setpp -i
```

oder die Ansicht dynamisch gedreht wird.

Wie Sie sehen, kann es nicht schaden, sich auch mit den Matrix-Operationen etwas näher zu beschäftigen. Wir hatten ja bereits in den vorhergehenden Lektionen das "scale3"-Kommando mehrfach angewandt, aber dessen Komplexität sicherlich noch nicht wirklich erkannt. So verfügt "scale3" u.a. auch über eine Option -t, die es erlaubt eine Skalierung mit 4 x 4 Matrizen durchzuführen.

Zeigen wir das an einem nicht ganz trivialen Beispiel. Wir wollen unser Objekt "plane" z.B. um 90° um seine x-parallele Achse drehen. Natürlich würden Sie das normalerweise mit dem "rot3"-Kommando machen. Wir wollen hier nur zeigen, dass das erstaunlicherweise auch mit dem variantenreichen scale3-Kommando mittels Matrix-Operation funktioniert. In bekannter Weise erfolgt die Berechnung der Rotations-Matrix mit

```
val op(1,PI/2)
```

als Ergebnis erhält man

```
op(1,PI/2) = (1,0,0,0,0,0,-1,0,0,1,0,0,0,0,0,1)
```

Sodass man natürlich

```
scale3 plate -t1,0,0,0,0,0,-1,0,0,1,0,0,0,0,0,1
```

eingeben könnte. Das wäre die "manuelle" Methode. Wir wollen aber automatisieren. Also wäre die Skalierung mit der direkt berechneten Formel deutlich praktischer, also

```
scale3 plate -t$[?op(1,pi/2)]
```

Damit haben wir den gewünschten Effekt der x-90°-Drehung in einem Schritt erzielt. Die Wirkung ist schon recht verblüffend, wenn man bedenkt, dass ein scale3-Kommando anstelle eines "rot3"-Kommandos genutzt wurde.

Man kann ja einmal der Phantasie freien Raum lassen, um sich auszumalen, was allein mit Matrix-Operationen alles möglich ist.

Die oben genutzten 16 Zahlen sind für die Lesbarkeit der Matrix sicherlich etwas unhandlich. Deshalb nutzten Sie ggf. folgenden Trick. Vergrößern Sie versuchsweise einmal das Kommandofenster (s.u.) und geben Sie dann das Kommando

```
val op(1,PI/2)
```

erneut ein. Dann können Sie zusätzlich auch die 4 x 4 Matrix-Darstellung erkennen. Das ist deutlich besser lesbar und kann in jedem relevanten Geometriebuch nachgeschlagen werden.

```
:
:val op(1,PI/2)
op(1,PI/2) =
        1.00000000      0.00000000      0.00000000      0.00000000
        0.00000000      0.00000000     -1.00000000      0.00000000
        0.00000000      1.00000000      0.00000000      0.00000000
        0.00000000      0.00000000      0.00000000      1.00000000
:
```

Eine andere, alternative Methode, die Matrix anzuzeigen, bestünde auch darin, sie sich im Sheet anzusehen. Mit dem Kommando

```
cal a1=op(1,pi/2)
```

wird die Matrix in die ersten Sheet-Zellen eingetragen. Das können Sie selbstständig durch Eingabe von

```
sheet
```

überprüfen.

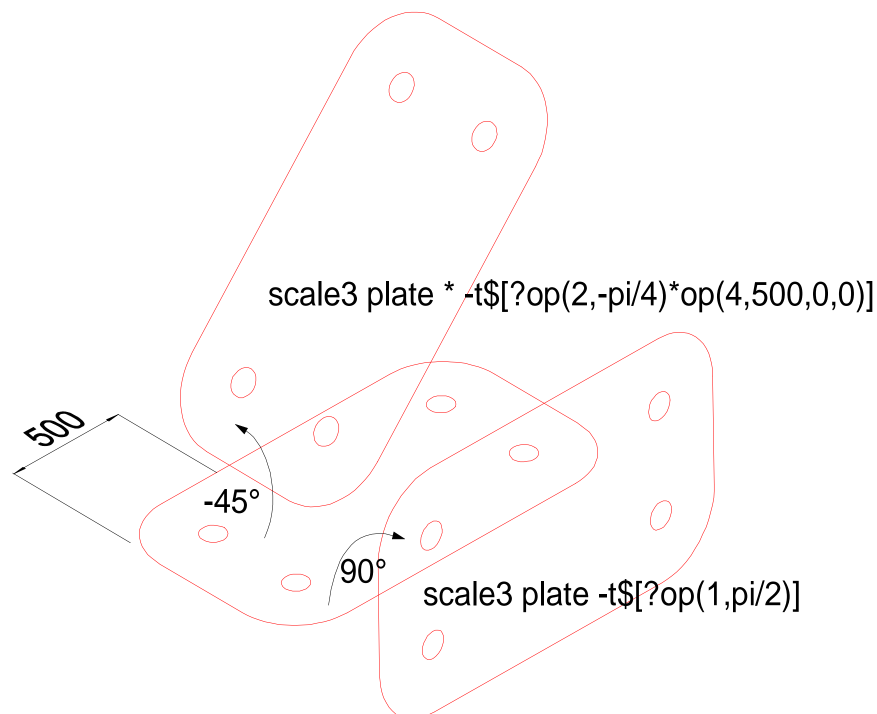
Zurück zu den Matrix-Operationen. Nun sollte natürlich auch noch erwähnt werden, dass sich Matrizen-Operationen natürlich auch kombinieren lassen (Matrix-Multiplikation). Wenn wir also unser Objekt "plate" z.B. in X-Richtung um 500 verschieben und anschließend um -45° um die y-Achse drehen wollen, lautet das Kommando für die Berechnung der Matrix

```
val op(2,-PI/4)*op(4,500,0,0)
```

Bitte beachten Sie dabei die Reihenfolge der Matix-Multiplikation (von rechts nach links). Es wird also erst die Verschiebung und anschließend die Drehung ausgeführt.

Somit lautet das scale3-Kommando (hier in der Formelschreibweise und mit Duplizierung (u.a. auch wegen der vielen Nachkommastellen))

```
scale3 plate * -t$[?op(2,-pi/4)*op(4,500,0,0)]
```



Damit wollen wir diesen kurzen Abstecher zu den Matrix-Operationen abschließen und wenden uns noch einmal kurz mit dem text-Kommando zu.

Einen ganz speziellen Zeichenbefehl, nämlich "oi" (für "object info"), sollten wir jedoch in jedem Fall noch erwähnen. Mit ihm können beliebige, textuelle Informationen in den Pfuffer eingetragen und an ein Objekt gekoppelt werden. Das wird u.a. für Stücklisten, Datenbank- oder CAM-Informationen und Vieles mehr genutzt. In älteren Pictures-Versionen wurden auch mit " ' " oder "xk" bzw. "xl" andere Befehle genutzt, um Informationen in die Geometrie-Daten einzufügen. Diese sind auch heute noch kompatibel. Für neuere Anwendungen sollte man aber besser "oi" benutzen.

Will man z.B. in die "internen Daten" des Objekts "plate" eine Information "Mein Text" einfügen, könnte das z.B. mit

```
object info plate myinfo "Mein Text"
```

geschehen. Wenn Sie jetzt in den Editor schauen, erkennen Sie dort die neue Zeile

```
oi info=Mein Text
```

Wollte man im aktuellen Puffer aus einer solchen Zeile die Text-Daten auch wieder auslesen, lautete für Basic das Kommando z.B.

```
bix print app.object("plate").info("myinfo")
```

aber auch mit dem Text-Kommando ließe sich das lösen

```
text g$[modbuf]1 "soi myinfo" vvar ; subst var "oi myinfo=" "" ; echo $var
```

Dabei wird in der einfachsten Form mit "s" für "search" nach "oi myinfo" direkt gesucht, die Puffer-Zeile mit "v" für "variable" in der Variable var gespeichert, der "überflüssige" Text "oi myinfo=" mit dem subst-Kommando durch einen leeren Text "" ersetzt und mit dem echo-Kommando ausgegeben.

Sucht man zuvor auch noch das Objekt, kann das Kommando folgendermaßen lauten

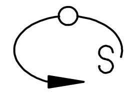
```
text g$[modbuf]1 "sos plate" foi vvar ; subst var "oi myinfo=" "" ; echo $var
```

Da die Textinformationen mit "oi" mit dem Objekte verknüpft sind, werden sie natürlich bei einer Duplizierung eines Objekts auch auf das Duplikat übertragen.

Nutzen Sie beispielsweise unter **Pictures** die integrierten Stücklisten-Tools, so basieren deren Attribute auf dem "oi"-Befehl. Natürlich werden für komplexe Auswertungen von Objekt-Informationen im allgemeinen, individuelle BIX-Routinen erstellt.

Fazit:

Wie Sie sehen, bietet die Transparenz der textbasierten Zeichnungsbefehle sehr viele Möglichkeiten zur Manipulation und zur Individualisierung. Verlieren Sie sie bei Ihren Projekten also nicht aus den Augen und vertiefen Sie Ihre Kenntnisse im Selbststudium.



Zugriff auf Geometrie-Elemente

Vorbemerkung:

Lassen Sie uns kurz bevor wir fortfahren noch einmal den Unterschied zwischen einem "Objekt" und einem "Element" unter **Pictures by PC** klarstellen. Ein "Objekt" ist ein geometrisches Gebilde, auf das per Objektnamen zugegriffen werden kann und dass gewöhnlich mit einem oder mehreren Kommandos generiert wurde. Dabei besteht ein Objekt hinwiederum aus (einem oder) mehreren "Elementen". Diese stellen die kleinste, unter **Pictures** verwaltete Geometrie-Einheiten dar. Elemente haben keinen Namen, mit dem sie gezielt angesprochen werden könnten.

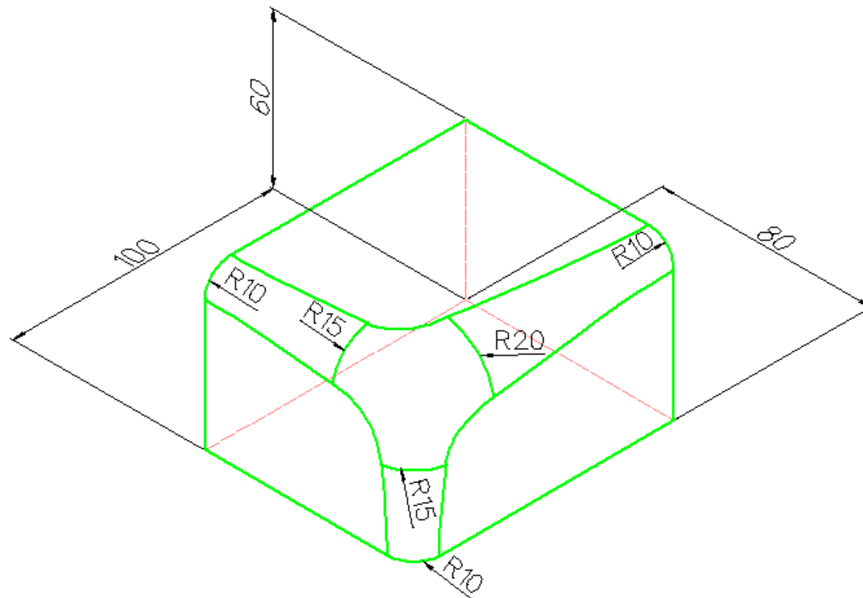
Verdeutlichen wir das z.B. an unserem, einfachen Objekt "plate" (s.o.). Das Objekt wurde mit dem Kommando "recta" erzeugt und ist über den Objektnamen ansprech- und manipulierbar. Geometrisch setzt sich "plate" aus einigen "Elementen" (hier: aus Geraden und Kreissegmenten) zusammen. Ein Einzelnes davon kann man im allgemeinen mit Kommandos nicht direkt ansprechen.

Um gerade diese vermeintliche "Einschränkung" zu überwinden, wurde diese Lektion ja bewusst verfasst,. Durch den direkten Puffer-Zugriff und die Manipulation der Zeichenbefehle (s.o.) haben wir das ja oben schon ansatzweise gezeigt.

Lassen Sie uns damit also fortfahren.

In den vorangegangenen Lektionen hatten wir ja schon gezeigt, dass man mit **Pictures** geometrische Gebilde per Programm-Sequenzen automatisch erzeugen kann. Dennoch erfordern viele Geometrie-Modifikation (insbesondere an 3D-Elementen) einen manuellen Eingriff des Benutzers. Das steht aber gerade im Widerspruch zum Wunsch, unsere Prozesse automatisieren zu wollen. Im Folgenden soll daher das Prinzip gezeigt werden, wie dieser "Engpass" zu überwinden ist.

Zur Veranschaulichung sei unsere Aufgabenstellung, drei angrenzende Kanten eines 3D-Quaders mit einer Kofferecke mit variablen Radien (s.u.) zu versehen (vgl. auch kofferecke2.prc).



Den Quader mit den Abmessungen 100 x 80 x 60 am Nullpunkt beginnend, erzeugen wir mit dem uns bekannten Kommando

```
block3d * -p0,0,0..100,80,60
```

Es empfiehlt sich eine isometrische Sicht und Shading einzuschalten

```
setpp -i ; shade -d
```

Als Vorübung wollen wir jedoch erst einmal eine einfache, direkte Kantenabrundung mit einem Radius von 20 an dem Quader vornehmen. Wollte man das interaktiv machen, so nutzte man das Kommando

```
blnd3d -r20
```

und müsste die entsprechende Kante per Fadenkreuz selektieren.

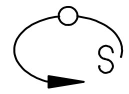
Wenn wir die Kantenrundung allerdings automatisch erzeugen wollen, fehlen uns dazu bislang geeignete Mittel.

Denn leider gibt es bei dem Kommando "blnd3d" keine Option, mit denen Elemente wie z.B. Punkte, Kanten oder Flächen mit numerischen Koordinaten anzuwählen sind. Dieses "Problem" lässt sich in älteren Pictures-Versionen nur "sehr trickreich" und alles andere als ideal lösen (s. kofferecke.prc)

Deshalb wurde in **Pictures Rev. 3.8** diese Problematik mit dem Kommando "biac" für "**B**lock **I**nter**A**ction of **C**ommand" überwunden. Allerdings ist das Kommando etwas erklärungsbedürftig. Zunächst geben wir aber das Kommando

```
biac -e g=ed/p=90,0,60 blnd3d -r20
```

ein und erhalten eine automatisch Abrundung der oberen, x-parallelen Kante mit einem Radius von 20. Also genau das, was wir haben wollten, Nun müssen wir das Kommando nur noch verstehen.



Beim Kommando-Aufruf von biac (oben) gaben wir mit der Option -e (elements) dezidierte Elemente-Informationen an, damit das darauf folgende blnd3d-Kommando ausgeführt werden konnte.

Im obigen Beispiel wurde in der Option -e Folgendes angegeben:

Die Rastgruppe "g=ed/p=90,0,60" bezieht sich auf die 3D-Kante (edges, "ed") an der Punktposition p=90,0,60 (also an der oberen Kante auf der rechten Seite). An diesem Punkt soll nun mit dem im Ausdruck oben folgenden Kommando "blnd3d -r20" die Kante selektiert werden.

Natürlich muss dabei obige Syntax eingehalten werden. Die Logik ist aber leicht zu verstehen.

Damit ist das Prinzip grundsätzlich bereits erklärt und wir können uns der Erzeugung der Kofferecke zuwenden. Vorher müssen wir die jetzige Ausrundung mit "undo" allerdings noch zurücknehmen.

Der Unterschied bei Kantenabrundungen zur Erzeugung einer Kofferecke besteht darin, dass die drei angrenzenden Kantenradien und das "Set-Back" am Eckpunkt erst vordefiniert werden müssen (-s für "set"), bevor die Kofferecke dann endgültig erzeugt werden kann.

Interaktiv würden wir das mit den Kommandos

```
blnd3d -s (Kanten-Radien vordefinieren)
blnd3d -sv (Setback an Ecke setzen)
blnd3d -fb (Alle vordefinierten Radien erzeugen)
```

lösen.

Wenn wir interaktiv mit dem Kommando

```
blnd3d -cb -r20 -q10 -s
```

die rechte Seite der oberen, in x-Richtung verlaufenden Kante des Quaders anklicken würden, definierten wir an dieser Kante einen Übergangsradius von 20 auf 10 vor.

Das wollen wir aber jetzt ohne manuellen Eingriff und ohne Benutzerdialog direkt mit "biac" vollziehen. Das Kommando lautet

```
biac -e g=ed/p=90,0,60 blnd3d -cb -r20 -q10 -s
```

Im obigen Beispiel wird in der Option -e Folgendes angegeben:

Die Rastgruppe "g=ed/p=90,0,60" bezieht sich auf die 3D-Kante (edges) an der Punktposition p=90,0,60 (also an der oberen Kante auf der rechten Seite). An diesem Punkt soll nun mit dem im Ausdruck oben folgenden Kommando "blnd3d -cb -r20 -q10 -s" die Kante selektiert und der variable Radius (20 auf 10) vordefiniert werden.

Jetzt haben wir also die obere Kante an unserem Quader mit dem variablen Radius von 20 auf 10 zur Ausrundung vordefiniert.

Nehmen wir uns jetzt die y-parallele, linke, obere Kante an einem vorderen Punkt (y=10) vor und definieren einen variablen Radius von 10 auf 15, dann lautet das Kommando

```
biac -e g=ed/p=0,10,60 blnd3d -cb -r10 -q15 -s
```

Bleibt noch die vordere Kante unten mit einem Radius von 15 auf 10 übrig

```
biac -e g=ed/p=0,0,10 blnd3d -cb -r15 -q10 -s
```

Damit sind alle drei Kanten vordefiniert und es muss noch der Set-Back-Punkt am Eckpunkt gewählt werden. Das geschieht z.B. mit

```
biac -e g=vt/p=0,0,60 blnd3d -v -s1
```

Beachten Sie, dass dabei die Rastgruppe in der Option -e mit "vt" für "vertex" (Ecke) angegeben wurde. Damit ist auch das Set-Back vordefiniert (ggf. besser sichtbar mit nicht schattierter Sicht).

Als Letztes gilt es jetzt natürlich noch die vordefinierten Abrundungen auch wirklich auszuführen. Das geschieht durch Anklicken des Volumenkörpers "body" mit

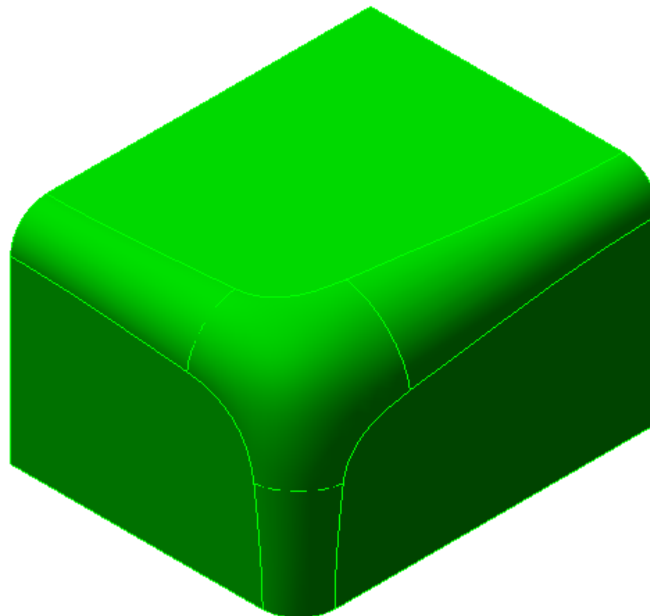
```
biac -e g=body/p=0,0,0 blnd3d -fb
```

alternativ hätte z.B. aber auch mit.

```
biac -e g=ed/p=0,0,0 blnd3d -fb
```

funktioniert.

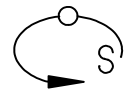
Als Ergebnis erhalten wir unsere "schöne, automatisch" erzeugte Kofferecke.



Natürlich ist "biac" nicht nur auf so einfache Ausrundungen beschränkt, sondern kann mit jedem interaktiven Kommando, das Elemente-Selektion zulässt, kombiniert werden.

Lassen Sie uns nur zur Übung einfach mit der lokalen Operation "dfaces" zum Löschen von Flächen die "Set-Back"-Fläche, also die soeben erzeugt Kofferecke, wieder entfernen, allerdings unter Erhaltung der variablen Radien.

Da in diesem Fall ein exakter Selektionspunkt auf der Eckfläche relativ schwierig zu ermitteln wäre, nutzen wir im Kommando unten nur "ungefähre, ganzzahlige" Punkt-Koordinaten und einen Toleranzwert "t=10" (maximal zulässige, absolute Abweichung von 10), dann funktioniert das z.B. mit




```
biac -e g=fc/t=10/p=10,10,50 dfaces3d -c
```

Natürlich sollte der angegebene Punkt (hier: 10,10,50) der Eckfläche "so gut als möglich" nahekommen, aber dennoch einfach abschätzbar sein (möglichst "glatte" Koordinaten). Mit der Angabe eines angemessenen Toleranzwertes (hier: t=10) können wir mit einer "gewissen" Punktabweichung leben, um die "richtige" Fläche zu selektieren.

Fazit:

Wie Sie sehen, bietet das "biac"-Kommando für die Automatisierung vielseitig Möglichkeiten, bei denen bislang ein Benutzereingriff durch manuelles "Anklicken" unvermeidlich war. Das eröffnet das hochinteressante Feld der parametrischen Konstruktion und Automatisierung, ohne die interaktiven Manipulationen einzuschränken, also genau das, was wir uns zum Ziel gesetzt hatten.

Machen Sie Gebrauch von diesen enormen Möglichkeiten.