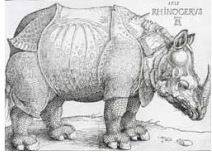
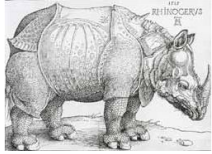
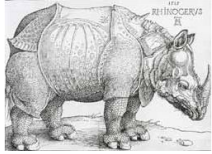


Scheme ACIS Interface Driver Extension (Scheme AIDE)



Для чего используется?

It provides a means of exercising ACIS functionality without writing or compiling a stand-alone C++ application. This helps developers learn and prototype functionality.

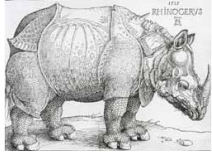


Starting Scheme AIDE

Windows Platform

Topic: *Scheme AIDE Application

To start the application on a Windows platform, locate the executable file in your installed directory and double click on the filename. The executable filename is $A3DT\bin\$ARCH\acis3dt.exe, where $A3DT is an environment variable that is defined as the name of your root installation directory and $ARCH is an environment variable that is defined as the name of your platform directory (NT).



```
; Sample input file test.scm
(define b1 (solid:block (position -20 -20 0) (position 20 20 5)))
;; b1
(define s1 (solid:sphere (position 0 0 0) 15))
;; s1
(define new (bool:intersect b1 s1))
;; new
```

Example 6-1. Sample Command Input File

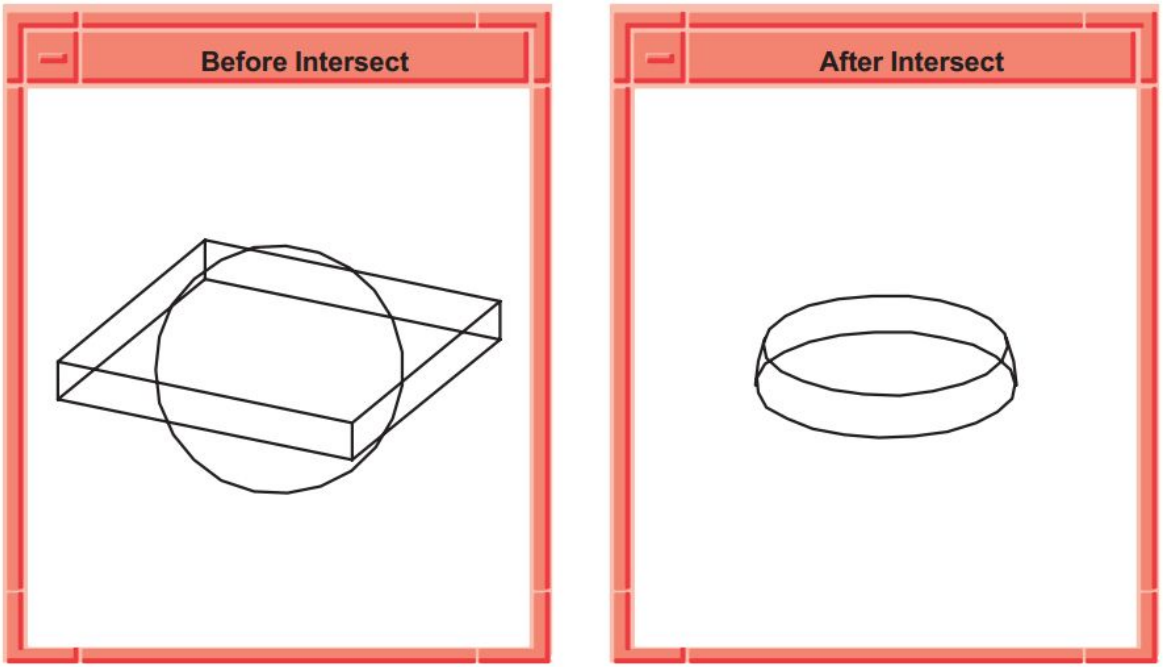
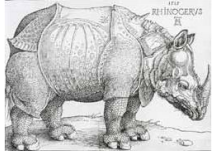
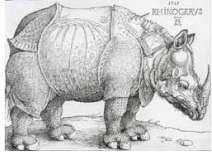


Figure 6-1. Command Input File



```
; Delete all entities from the view
(part:clear)
;; #t
; Load the Scheme input file test.scm
(load "test.scm")
;; ()
```

Example 6-2. Executing the Command Input File



Unite a Block and a Cylinder

Topic: Scheme AIDE Application

Example 6-3 creates a block and a cylinder, then unites them. Figure 6-2 shows the model before and after the entities are united.

```
; Create a solid block
(define b1 (solid:block (position -20 -20 -20)
    (position 20 20 20)))
;; b1
; Create a cylinder
(define c1 (solid:cylinder (position 20 0 -20)
    (position 20 0 20) 20)))
;; c1
; Unite the two bodies into a new body
(define u1 (bool:unite b1 c1))
;; u1
```

Example 6-3. Unite Block and Cylinder

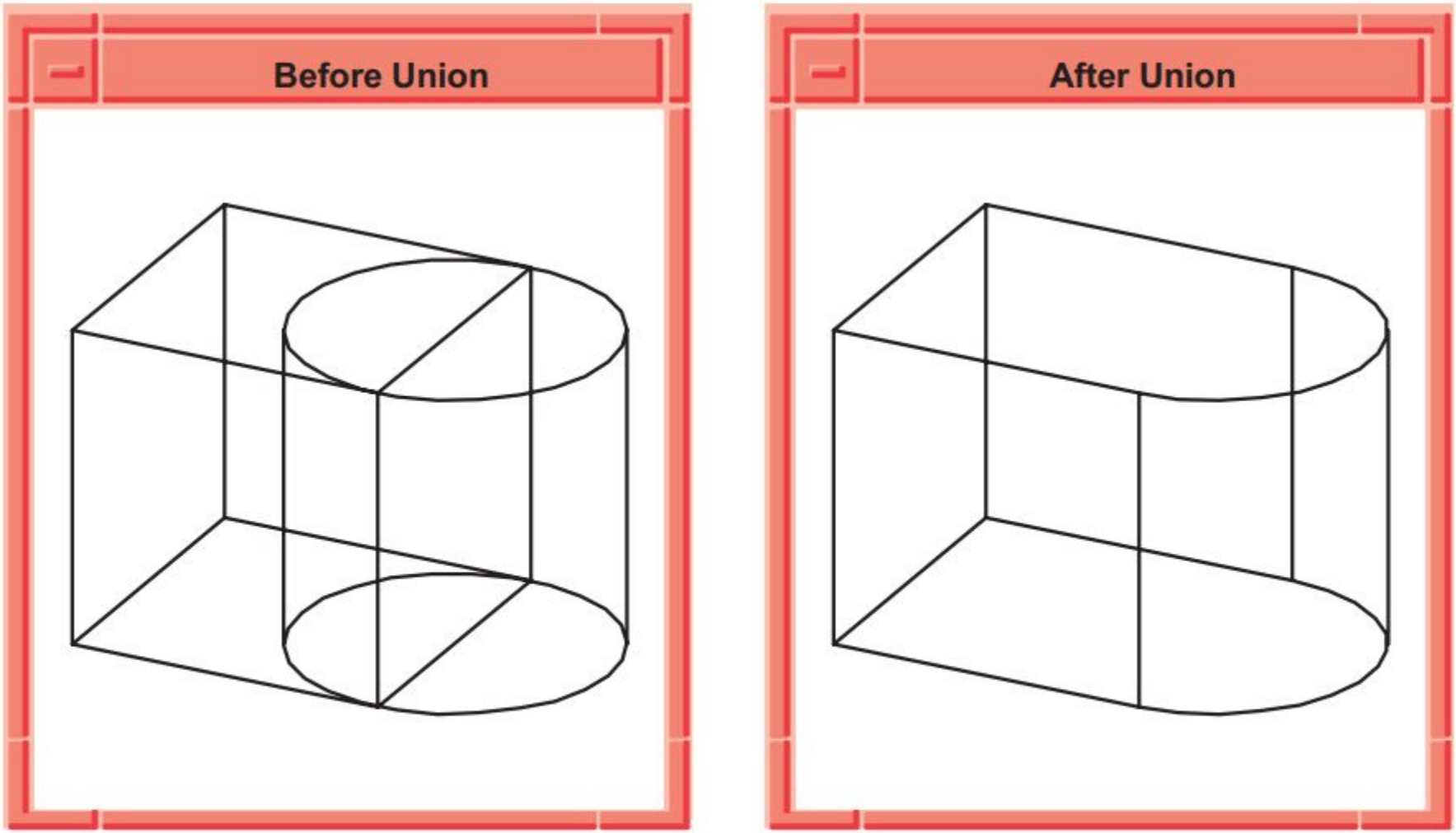
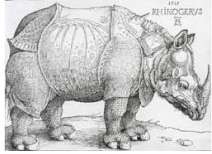


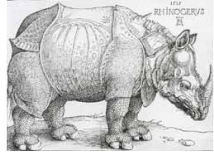
Figure 6-2. Unite Block and Cylinder



Rotate Body

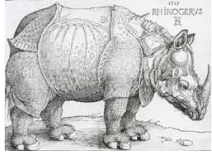
Topic: Scheme AIDE Application

Example 6-4 rotates the body u1 (after the union) from Example 6-3 so you can view it from a different angle. A transform is first defined, then applied to the body. Figure 6-3 shows the rotated body.

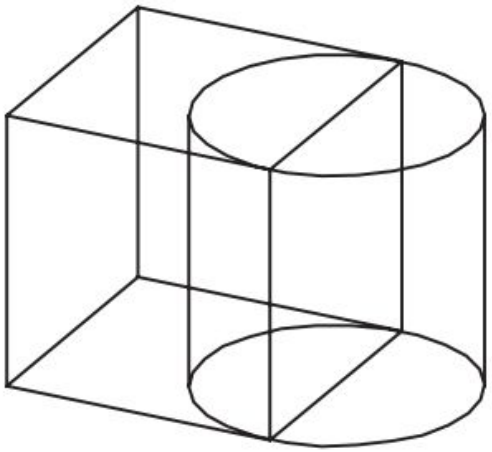


```
; (part:clear)
; Create a solid block
(define b1 (solid:block (position -20 -20 -20)
    (position 20 20 20)))
;; b1
; Create a cylinder
(define c1 (solid:cylinder (position 20 0 -20)
    (position 20 0 20) 20)))
;; c1
; OUTPUT Before Union
; Unite the two bodies into a new body
(define u1 (bool:unite b1 c1))
;; u1
; Define a transform, t1, to rotate an object about the z-axis by
; 90 degrees
(define t1 (transform:rotation (position 0 0 0 )
    (gvector 0 0 1) 90))
;; t1
; OUTPUT After Union
; Apply the transform t1 to the body u1
(entity:transform u1 t1)
;; #[entity 2 1]
; OUTPUT Rotate/Transform
```

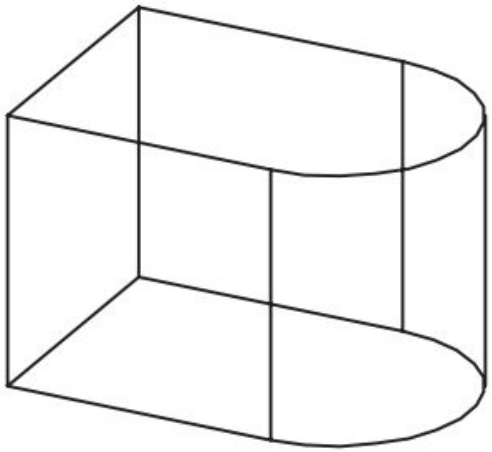
Example 6-4. Rotate Body



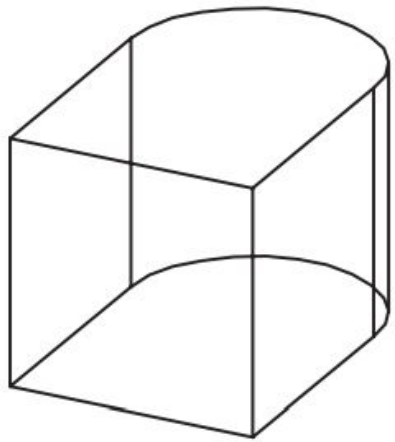
Before Union

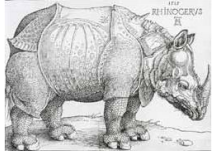


After Union



Rotate/Transform





Save and Load

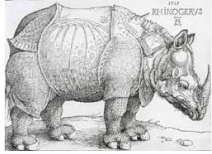
Topic: Scheme AIDE Application

ACIS models can be saved to a part save file for later retrieval. They can be saved in text mode (file extension .sat) or binary mode (file extension .sab). Refer to the *Kernel Component Manual* for information about save files.

Example 6-5 saves the body u1 created in Example 6-4 into a part save file in text format and then retrieves it from that file as a body with a new name.

```
; Save the model in text mode to file test.sat
(part:save "test" #t)
;; #t
; Delete all the entities
(part:clear)
;; #t
; Nothing in view window now
; Retrieve, or load, the model from the text mode file test.sat
(part:load "test")
;; (#[entity 4 1])
```

Example 6-5. Save and Load

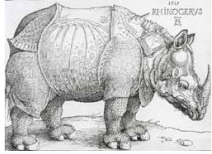


Expression

acis>

prefix notation

```
acis>(* 45 68)
3060
acis>(* 45 68 77)
235620
acis>(* (+ 40 5) (* 4 17))
3060

acis>(solid:block  (position 0 0 0) (position 20 20 20))
#[entity 1 0]
```

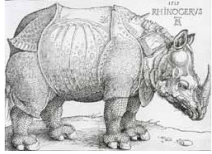


External Representation

#[type_of_object <arguments>]

;creates a position object
acis>(position 20 20 20)
#[position 20 20 20]

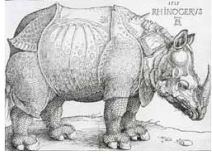
;creates a solid block
acis>(solid:block (position 0 0 0) (position 20 20 20))
#[entity 1 0]



Defining Variables

(define <label> <expression>)

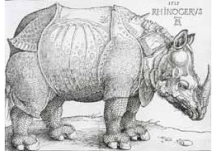
```
acis>(define prod (* 45 68))
prod
acis>prod
3060

acis>(define p1 (position 10 20 30))
p1
```



Defining Functions

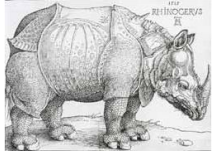
;Ddefine the procedure “square”

```
acis>(define (square x) (* x x))
square
acis>(square 5)
25
```



Conditional Statements

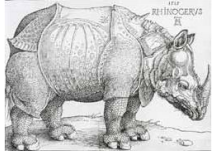
```
(cond (<condition_1> <consequence_1> )
        (<condition_2> <consequence_2> )
        (<condition_3> <consequence_3> )
        (else < consequence_3 >)
)

;Procedure for printing out an edge's type
(define  (tell_my_edge_type edge)
      (cond
      ((edge:circular? edge) (print "Circular_edge"))
      ((edge:elliptical? edge) (print "Elliptical_edge"))
            )
      (else (print "Is this an edge?"))
)
```



Conditional Statements

and or not

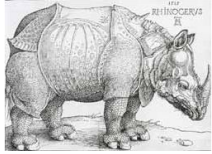
```
;Procedure for printing out an edge's type
(define  (tell_my_edge_type edge)
        (cond
        ((and (edge? edge) (edge:circular? edge)) (print "Circular_edge"))
        ((and (edge? edge) (edge:elliptical? edge)) (print "Elliptical_edge"))
                )
        (else (print "Is this an edge?"))
)
```



Conditional Statements

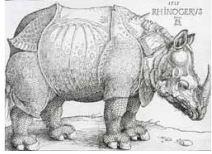
(if (<condition> <consequence> <alternative>)

```
(define cube (solid:block (position -30 -30 -30) (position 0 0 0)))
(define ball (solid:sphere (position 100 100 100) 25))
(define intersection (bool:intersect cube ball))

(if (solid? Intersection)
(print “They overlap”)
(print “They don’t overlap”)
)
```



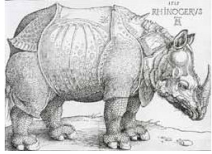
Recursion and Lists

```
; Create a cube and a sphere a unite them to form a body called union
(define cube (solid:block (position -30 -30 -30) (position 0 0 0)))
(define ball (solid:sphere (position 5 5 5) 25))
(define union (bool:unite cube ball))

; Create a list of the edges in union, find its lenght, start the recursion
; To run type (et union)
(define (et body)
      (define eelist (entity:edges body))
       (define list-length (length eelist))
       (work-through eelist list-length) )
```



Recursion and Lists

```
; Definition of work-through
(define ( work-through alist index)
        (define edge (list-ref alist (- index 1)))
         (tell-me-edge-type edge)
         (if (<= 0 (- index 1))
                ( work-through alist (- index 1))
                (print "No more edges")
        )
 )

(et union) ;Run the program
```



Recursion and Lists

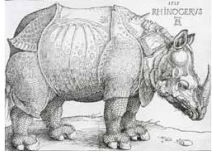
Многие функции ACIS Scheme возвращают тип **list** (список)

(define eelist (entity:edges body)

Здесь функция **entity:edges** возвращает **list**, который мы назвали **eelist**

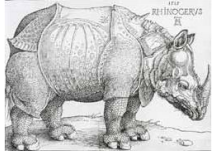
С помощью функции **length** мы можем узнать длину списка, а с помощью процедуры **list-ref** получить нумерованный элемент списка.

ACIS Scheme содержит еще две полезные функции для работы со списком: **car** и **cdr**. Первая, возвращает головной элемент списка, а вторая – остаток списка(без первого элемента.
Функции позволяют обрабатывать список без выяснения его длинны.



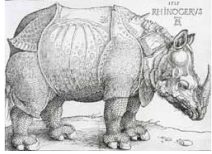
Recursion and Lists

```
acis>(define e-list '(e1 e2 e3  e4 ())
e-list
acis>(print (car e-list))
e1
acis>(print (cdr e-list))
(e2 e3 e4 ())
```



Recursion and Lists

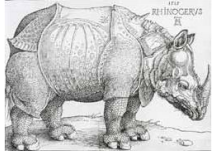
```
(define ( work-through alist)
      (define edge (car alist))
       (tell-me-edge-type edge)
       (if (null?(cdr alist))
            (print "No more edges")
            ( work-through (cdr alist))
      )
)
```



For-each

(**for-each** procedure list1)

```
(define (edge-types body)
  (define elist (entity:edges body))
    (for-each tell-me-edge-type elist))
```



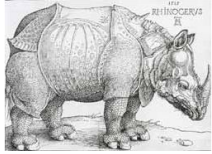
Set!

Значение переменной меняется с помощью оператора **set!**

(**define** val 67)
;val=67
(**set!** val 77)
;val=77

Оператор также используется для расширения списка

(**set!** load-path (**const** “C:\tmp” load-path))

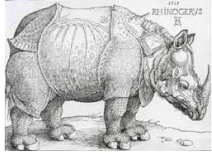


Define Local Variables

Функция **let** позволяет использовать переменные локально.

```
(let ((variable-name-1)(expression-1)
      (variable-name-2)(expression-2)
      (variable-name-3)(expression-3))
      body-expression)
```

```
(define ( work-through alist)
  (let ((edge (car alist))
        (tale (cdr alist)))
  (tell-me-edge-type edge)
  (if (null? tail)
    (print "No more edges")
    ( work-through (tail))
        )
 )
```

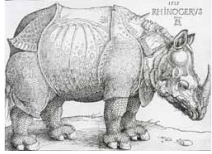


Lambda

Ключевое слово позволяет использовать не именованную(локальную) процедуру.

(**lambda** (function -arguments) (function-body))

```
(define  c-face
(lambda (body)
  (let* (
         (face-list (entity:faces body))
         (number (length face-list)))
    (display "Nunber of faces=")
    (display number)
    (newline))
       )
 )
```

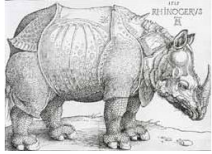


Do

Ключевое слово позволяет использовать не именованную(локальную) процедуру.

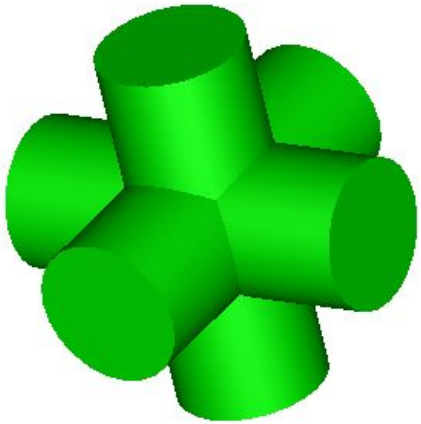
```
(do (variable init-expression update-expression)
      (test-expression exit-expression) continue-expression)

; Print out 10 different position round the base of a cone
; To run type (basepos)
(define basepos
 (lambda ()
  (let*
   ((cone (solid:cone (position 0 0 0)
                      (position 0 0 30) 15 0))
    (edges (entity:edges cone))
    (base (car edges)))
 (do ((param 0 (+ param 0.1)))
     ((> param 1) 'finish)
   (display (curve:eval-pos (curve:from-edge base) param))
   (newline)))))
```

view



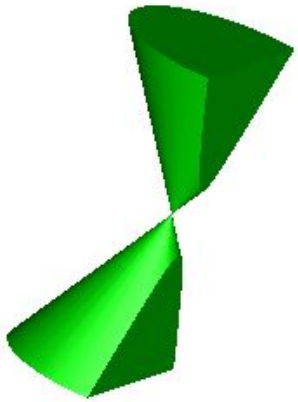
```
acis>(view:set
        (position 200 -400 200)
        (position 0 0 0)
        (gvector 0 0 1))
```

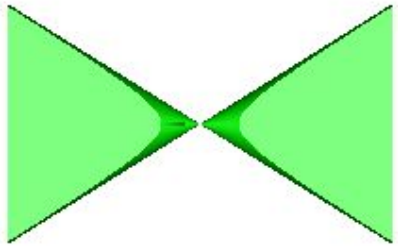
CSG



```
(define c1 (solid:cylinder (position 0 0 -50)(position 0 0 50) 20))
(define c2 (solid:cylinder (position  0 0 -50)(position 0 0 50) 20))
(define c3 (solid:cylinder (position  0 0 -50)(position 0 0 50) 20))
(define t1 (transform:rotation (position 0 0 0) (gvector 1 0 0) 90))
(define t2 (transform:rotation (position 0 0 0) (gvector 0 1 0) 90))
(entity:transform c1 t1)  ;Rotate about the x-axis
(entity:transform c2 t2)  ;Rotate about the y-axis
(define cross (solid:unite c1 c2 c3))   ;Unite c1 with c2 and c3
```

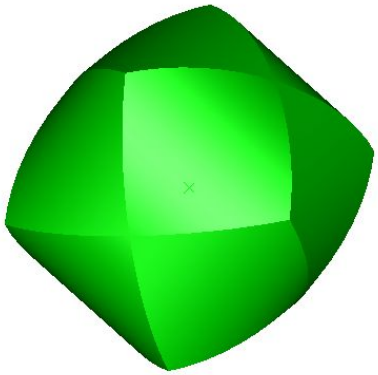
CSG

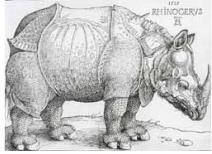




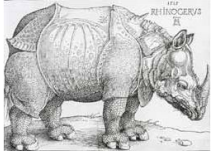
```
(define cone1 (solid:cone (position 40 0 0) (position 0 0 0) 25 0))
(define cone2 (solid:cone (position -40 0 0) (position 0 0 0) 25 0))
(define cone3 (solid:unite cone1 cone2))
(define plane  (face:plane (position -100 100 5) 200 200 (gvector 0 0 -1)))
(define cut (sheet:face plane))
(bool:subtract cone3 cut )
```

Mass Propeties





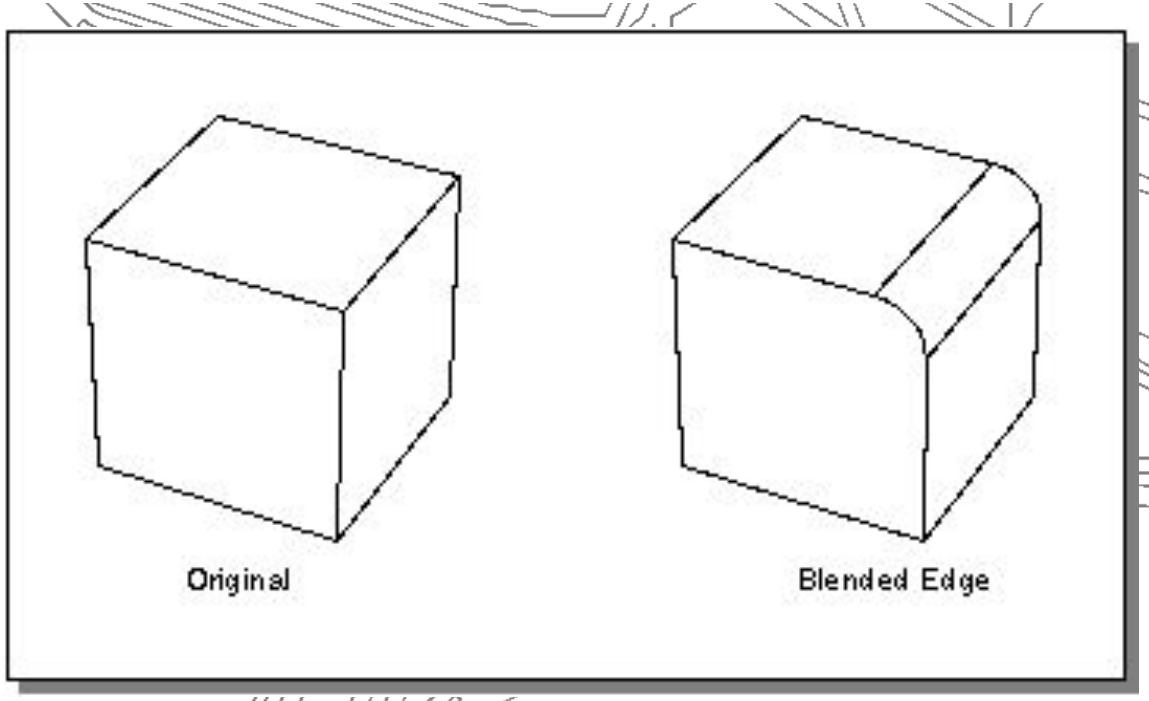
```
; Make a cylinder called cyl1
( define cyl1 ( solid:cylinder ( position 0 0 -50) ( position 0 0 50) 20))
; Make a cylinder called cyl2
( define cyl2 ( solid:cylinder ( position 0 -50 0) ( position 0 50 0) 20))
; Make a cylinder called cyl3
( define cyl3 ( solid:cylinder ( position -50 0 0) ( position 50 0 0) 20))
; Intersect cyl1 and cyl2 and cyl3
( solid:intersect cyl1 cyl2 cyl3)
; Find out its mass
 ( solid:massprop cyl1)
; (("volume" . 37490.3513788031) ("accuracy achieved" . 1.5977076644866 5e-005))
```

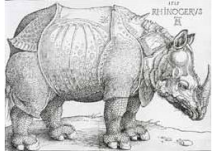


Model Modification in ACIS

Blending

The sharp edges and vertices in models must often be replaced by faces in order to improve the model. This operation is called blending. Blending is used to soften sharp edges and corners and to create smooth transitions from one surface to another. These changes may be needed to make a model more photorealistic, more aesthetically pleasing, safer, stronger (fewer stress points), or physically realizable (easier--or even possible--to manufacture).

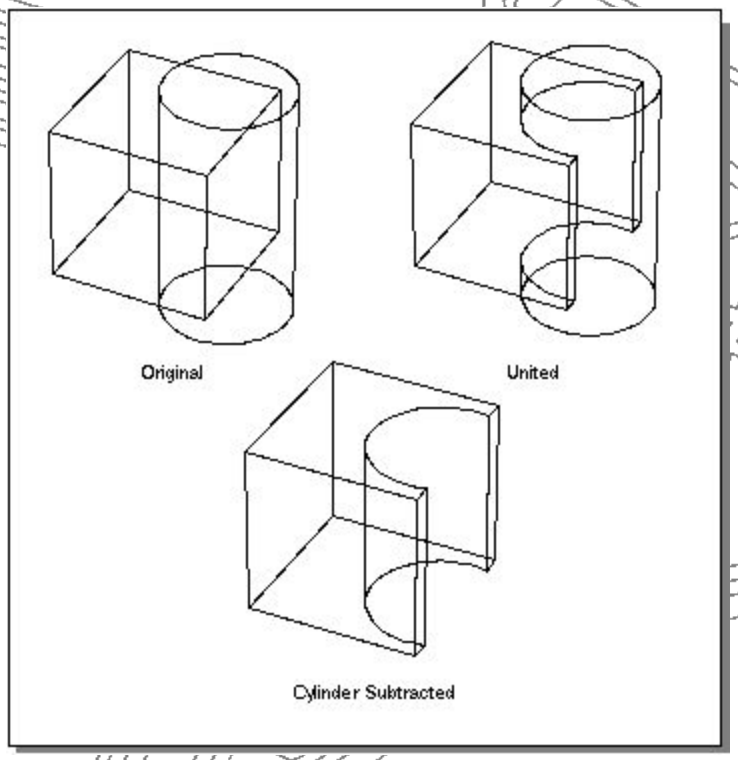


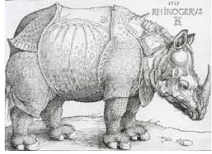


Model Modification in ACIS

Booleans

Boolean operations (Booleans) perform the set operations unite, intersect, and subtract on bodies. Booleans operate on model topology. Booleans use intersectors to find intersections between bodies and then decide which pieces to group together and which to discard. A body may be composed of solid, sheet or wire components.

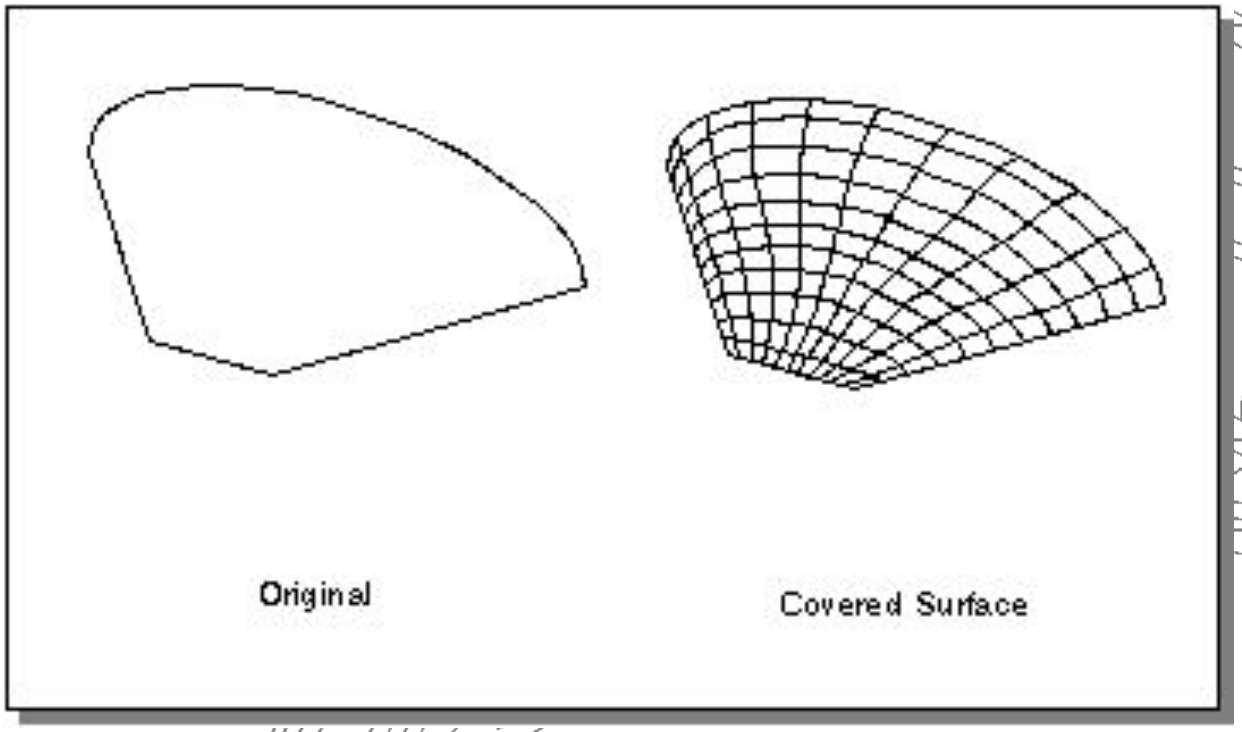


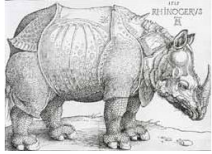


Model Modification in ACIS

Covering

Covering fits a surface over a closed loop of curves (wires); i.e., all the boundaries must be specified. For each wire in the wire body, an attempt is made to calculate a surface which contains all of the edges of the wire. A face is created and the coedges of the wire are made into loops in the face. If a surface can be calculated, it is used for the geometry of the face.

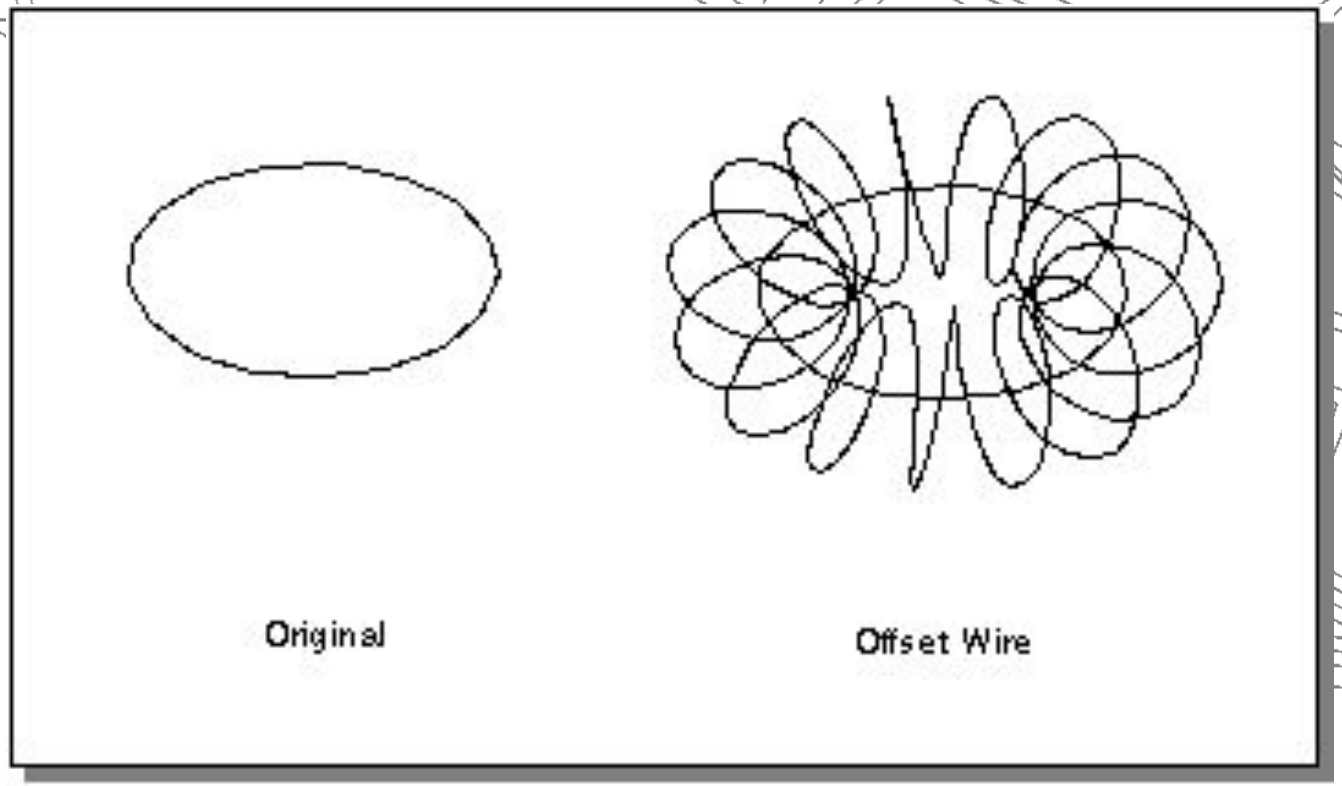


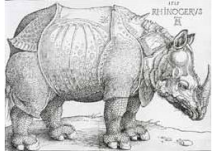


Model Modification in ACIS

Offsetting

New wires or faces can be created by offsetting from a reference wire body or face. Laws may be used for offsetting.

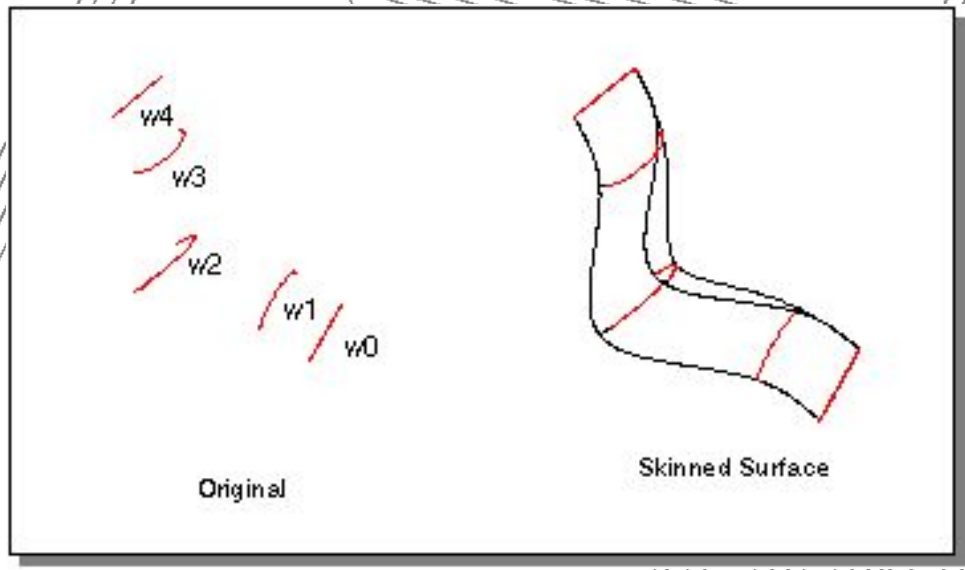


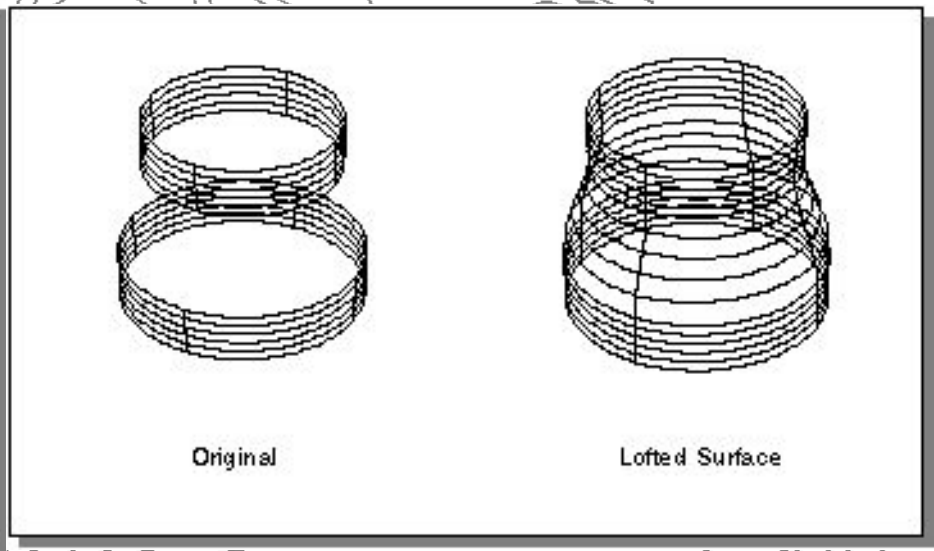


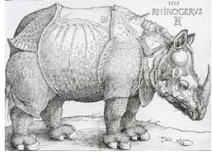
Model Modification in ACIS

Skinning and Lofting

Skinning fits a surface through a series of curves (wire bodies). Lofting starts with a surface and fits another surface through a coedge of the original surface and a series of curves (coedges). Lofting takes into consideration the tangents from the original surface at the first coedge and last curve. Laws may be used for lofting.



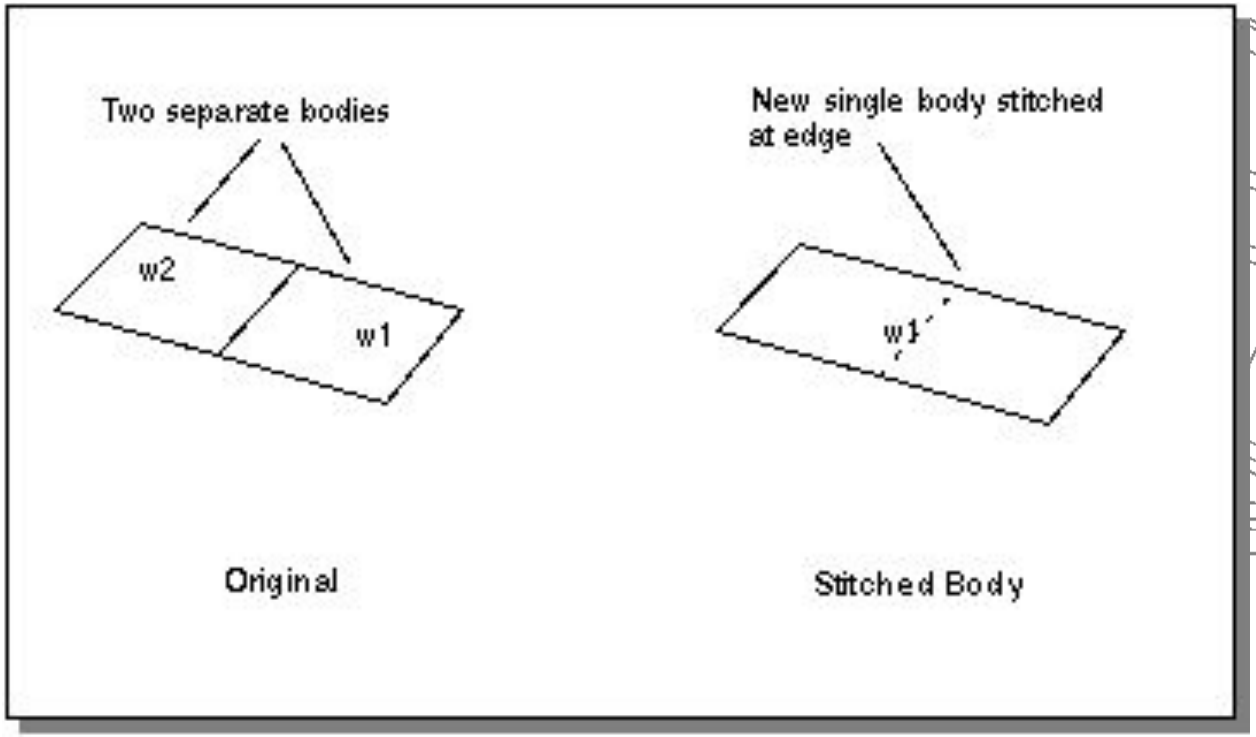


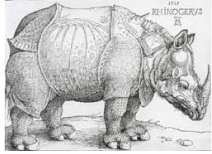


Model Modification in ACIS

Stitching

Stitching joins two bodies along edges or vertices that are identical. A stitch is simpler than a Boolean operation because stitching avoids face-face intersections and the evaluation of lump and shell containments. Most of the overhead in a stitching operation is associated with comparing edges to determine if they are entirely identical (coincident) or share some coincident subregion.

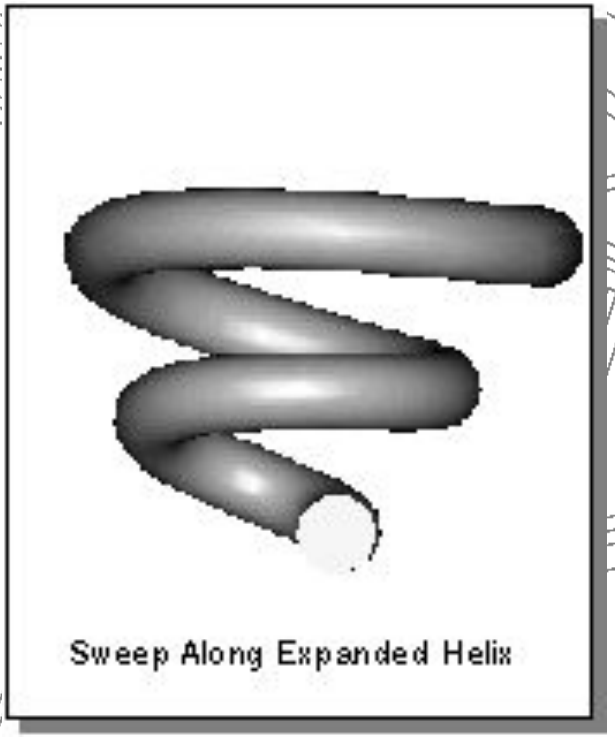




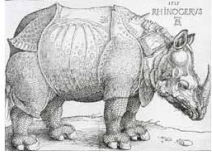
Model Modification in ACIS

Sweeping

Sweeping creates either a solid body or a sheet body by sweeping the profile, or shape, along a path or along an axis. The profile can be a face, a wire body, a closed group of edges, or an open group of edges. Whereas an open group of edges always results in a sheet body when swept, all other profile instances can result in either a solid or a sheet body. Laws may be used for sweeping.



Model Modification in ACIS



Sweeping

```
;; create an equilateral triangular prism
(define wedge
  (solid:sweep-wire
    (wire-body
      (list
        (edge:linear (position 0 0 -20) (position 30 0 -20))
        (edge:linear (position 30 0 -20) (position 30 60 -20 "polar"))
        (edge:linear (position 30 60 -20 "polar") (position 0 0 -20))))
    40))
```