Parasolid 3

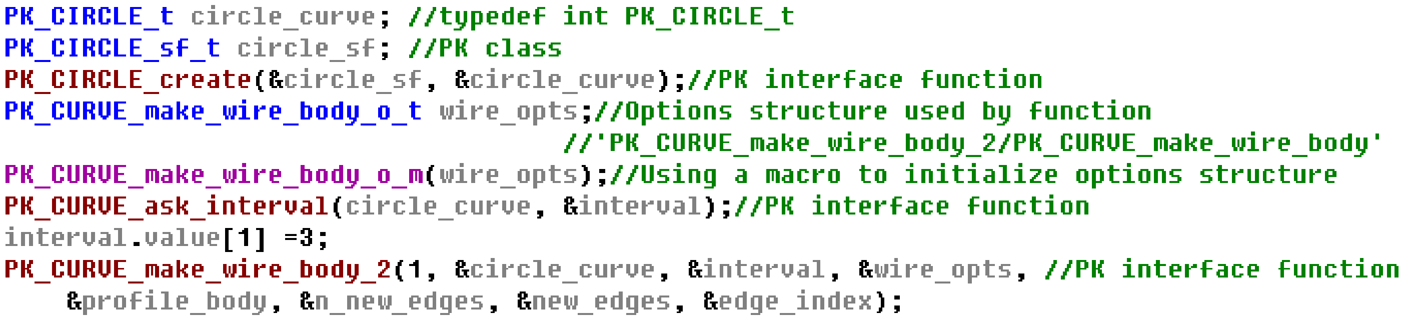
Human Centered CAD Laboratory

Contents

- Preview Forms of PK Interfaces
- Creating Bodies
- Boolean Operations
- Profiling
- Blending

Preview Forms of PK interfaces

- PK classes
  - Usually, names are of the form, ‘PK_XXXX_XXXX_t’.
- PK functions
  - PK_ <OBJECT>_ <text> (received arguments,…, returned arguments)
- PK option structures
  - Option structure : “_o_t”
  - Initialize of Option structure : “_o_t” ⟶“_o_m”



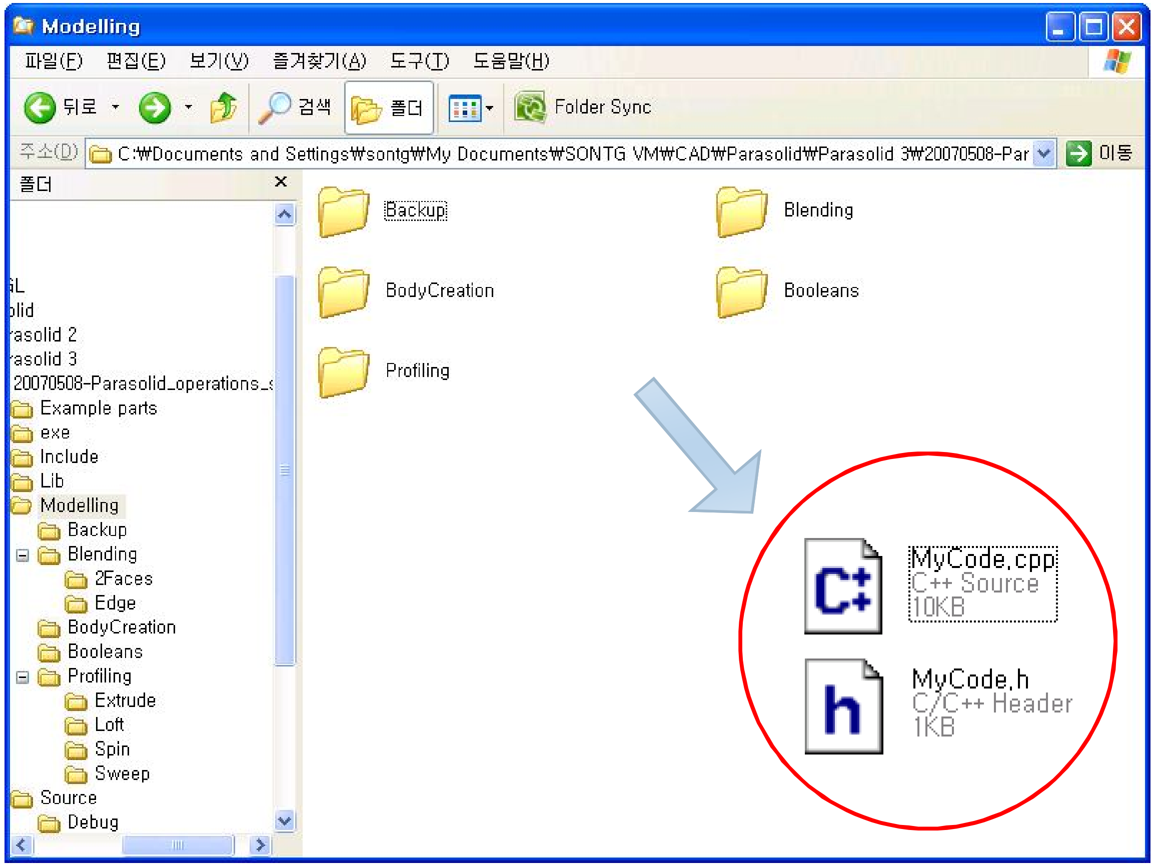
```
PK_CIRCLE_t circle_curve; //typedef int PK_CIRCLE_t
PK_CIRCLE_sf_t circle_sf; //PK class
PK_CIRCLE_create(&circle_sf, &circle_curve);//PK interface function
PK_CURVE_make_wire_body_o_t wire_opts;//Options structure used by function
                                     //'PK_CURVE_make_wire_body_2/PK_CURVE_make_wire_body'
PK_CURVE_make_wire_body_o_m(wire_opts);//Using a macro to initialize options structure
PK_CURVE_ask_interval(circle_curve, &interval);//PK interface function
interval.value[1] =3;
PK_CURVE_make_wire_body_2(1, &circle_curve, &interval, &wire_opts, //PK interface function
    &profile_body, &n_new_edges, &new_edges, &edge_index);
```

Preview Forms of PK interfaces

- Freeing memory used by return structures
  - Some return structures have code supplied to free the space pointed to by the structure. For a return structure whose name is of the form:
    - PK_<something>_r_t
  - then the freeing code is:
    - PK_<something>_r_f

How To ?

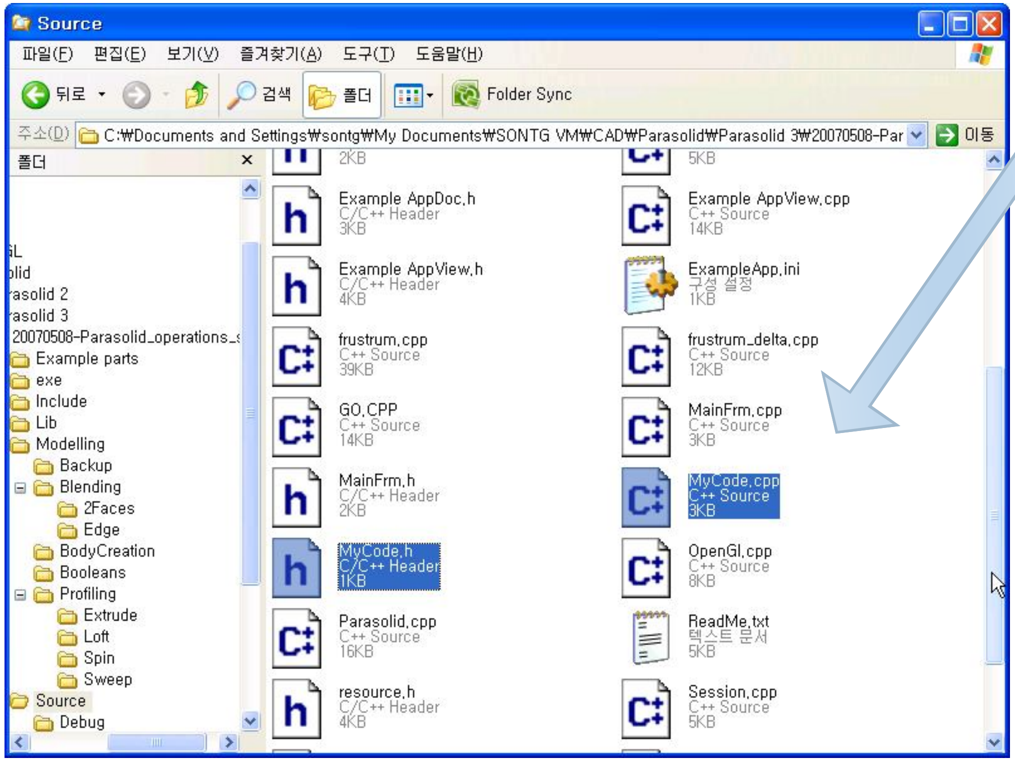
- Source Files

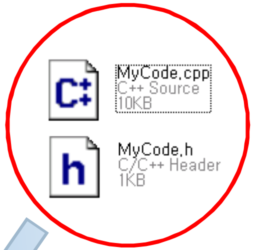


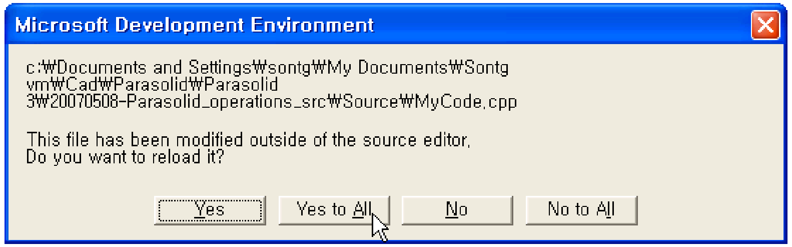
  - Each directory has two files for each practice

How To ?

- Replace

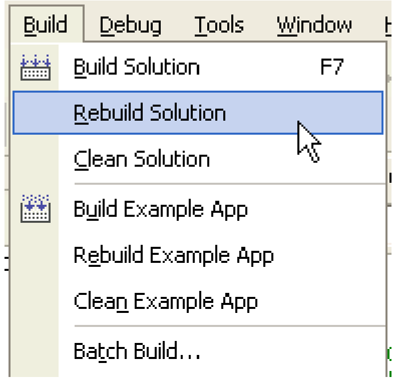

Source
파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)
뒤로
검색
폴더
Folder Sync
주소(D) C:₩Documents and Settings₩songt₩My Documents₩SONTG VM₩CAD₩Parasolid₩Parasolid 3₩20070508-Par
이동
폴더
Example parts
exe
Include
Lib
Modelling
Backup
Blending
2Faces
Edge
BodyCreation
Booleans
Profiling
Extrude
Loft
Spin
Sweep
Source
Debug
Example AppDoc.h C/C++ Header 3KB
Example AppView.cpp C++ Source 14KB
Example AppView.h C/C++ Header 4KB
ExampleApp.ini 구성 설정 1KB
frustrum.cpp C++ Source 39KB
frustrum_delta.cpp C++ Source 12KB
GO.CPP C++ Source 14KB
MainFrm.cpp C++ Source 3KB
MainFrm.h C/C++ Header 2KB
MyCode.cpp C++ Source 3KB
MyCode.h C/C++ Header 1KB
OpenGl.cpp C++ Source 8KB
Parasolid.cpp C++ Source 16KB
ReadMe.txt 텍스트 문서 5KB
resource.h C/C++ Header 4KB
Session.cpp C++ Source 5KB




Microsoft Development Environment

c:₩Documents and Settings₩songt₩My Documents₩Songt vm₩Cad₩Parasolid₩Parasolid 3₩20070508-Parasolid_operations_src₩Source₩MyCode.cpp

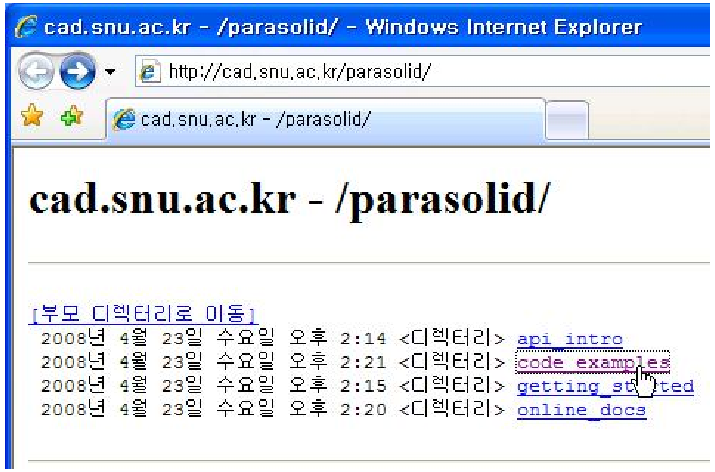
This file has been modified outside of the source editor.
Do you want to reload it?



- Replace the files in source folder with those in each example folder and “Rebuild Solution”

Reference

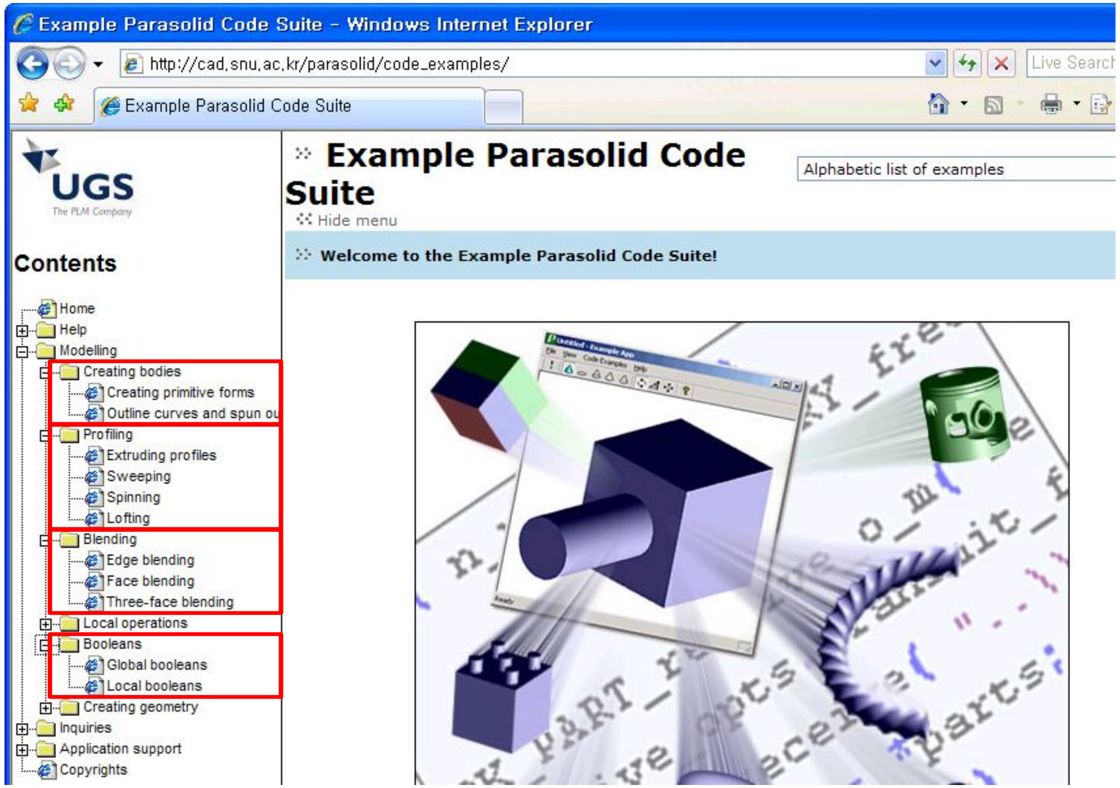
- Documentation on the Web
  - http://cad.snu.ac.kr/parasolid
  - code_example directory



cad.snu.ac.kr - /parasolid/ - Windows Internet Explorer
http://cad.snu.ac.kr/parasolid/
cad.snu.ac.kr - /parasolid/

cad.snu.ac.kr - /parasolid/

```
[부모 디렉터리로 이동]
 2008년 4월 23일 수요일 오후 2:14 <디렉터리> api_intro
 2008년 4월 23일 수요일 오후 2:21 <디렉터리> code_examples
 2008년 4월 23일 수요일 오후 2:15 <디렉터리> getting_st  ted
 2008년 4월 23일 수요일 오후 2:20 <디렉터리> online_docs
```



Example Parasolid Code Suite - Windows Internet Explorer
http://cad.snu.ac.kr/parasolid/code_examples/
Live Search
Example Parasolid Code Suite
UGS
The PLM Company
Alphabetic list of examples
Hide menu

Example Parasolid Code Suite

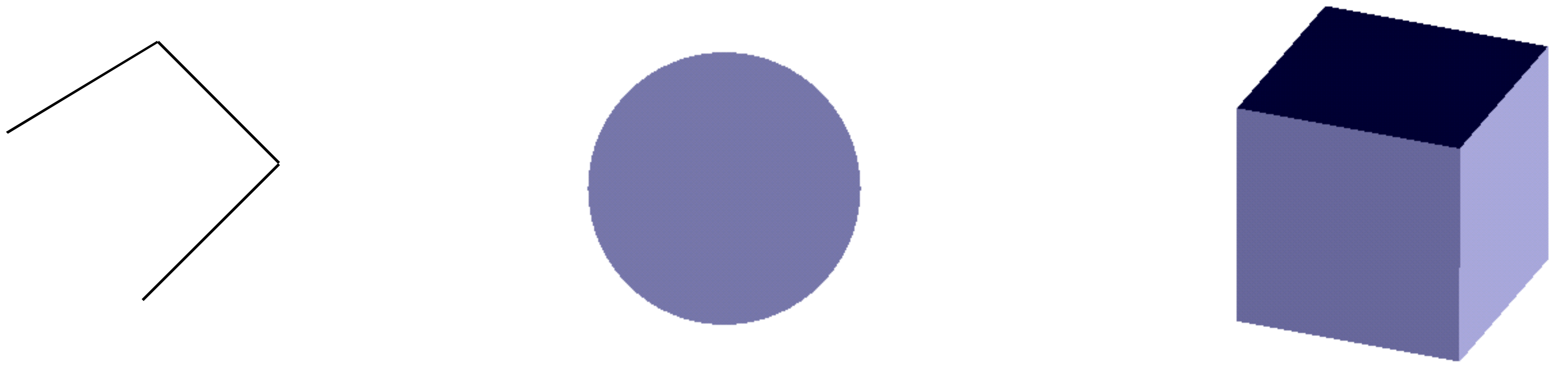
Welcome to the Example Parasolid Code Suite!

Contents

- Home
- Help
- Modelling
  - Creating bodies
    - Creating primitive forms
    - Outline curves and spun ou
  - Profiling
    - Extruding profiles
    - Sweeping
    - Spinning
    - Lofting
  - Blending
    - Edge blending
    - Face blending
    - Three-face blending
  - Local operations
  - Booleans
    - Global booleans
    - Local booleans
  - Creating geometry
- Inquiries
- Application support
- Copyrights

Primitives – Creating Bodies

- Wire body
- Sheet body
- Solid body

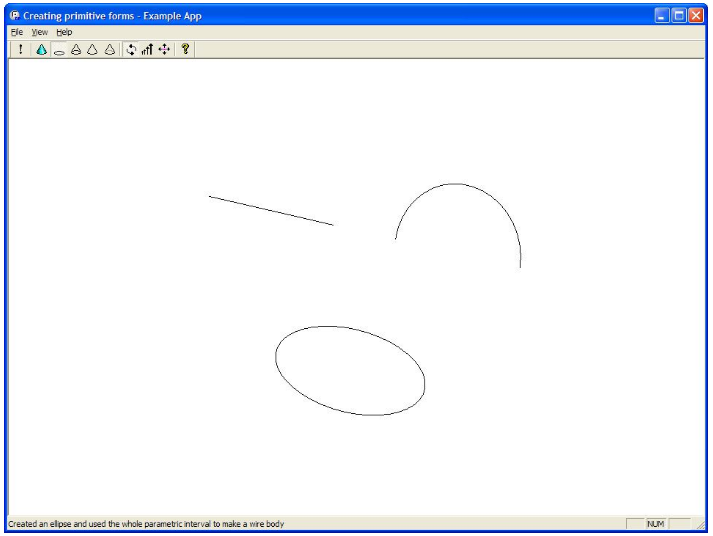


Primitives – Creating Wire body

- Step 1~3

1. Create a line and use it to make a wire body
2. Create a circle and minimum body and imprint the circle on the minimum body
3. Create a ellipse and use it to make a wire body

```
PK_CURVE_make_wire_body_2 (
 --- received arguments ---
int n_curves, --- number of curves (ie, length arrays)
const PK_CURVE_t curves[], --- curves to create a wire body
const PK_INTERVAL_t bounds[], --- bounds of each curve
const PK_CURVE_make_wire_body_o_t *options, --- options structure
--- returned arguments ---
PK_BODY_t *const body, --- the created wire body
int *const n_new_edges, --- number of new edges
PK_EDGE_t **const new_edges, --- new edges
int **const edge_index --- pos in original array
 )
```



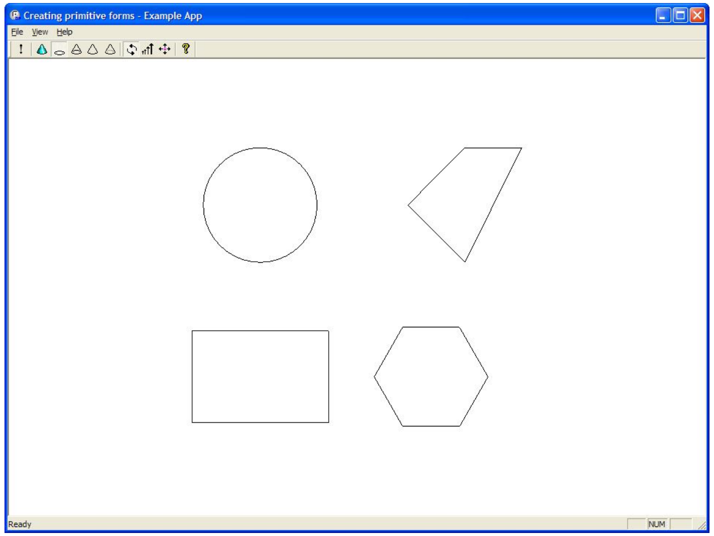
Created an ellipse and used the whole parametric interval to make a wire body

Primitives – Creating Sheet body

- Step4~7

4. Create a circle
5. Create a plane
6. Create a rectangle
7. Create a polygon

```
PK_BODY_create_sheet_circle (
 --- received arguments ---
double radius, --- radius of circle (>0)
const PK_AXIS2_sf_t *basis_set, --- position and orientation
(may be NULL)
--- returned arguments ---
PK_BODY_t *const body --- sheet body returned
)


PK_BODY_create_sheet_planar
PK_BODY_create_sheet_rectangle
PK_BODY_create_sheet_polygon
```

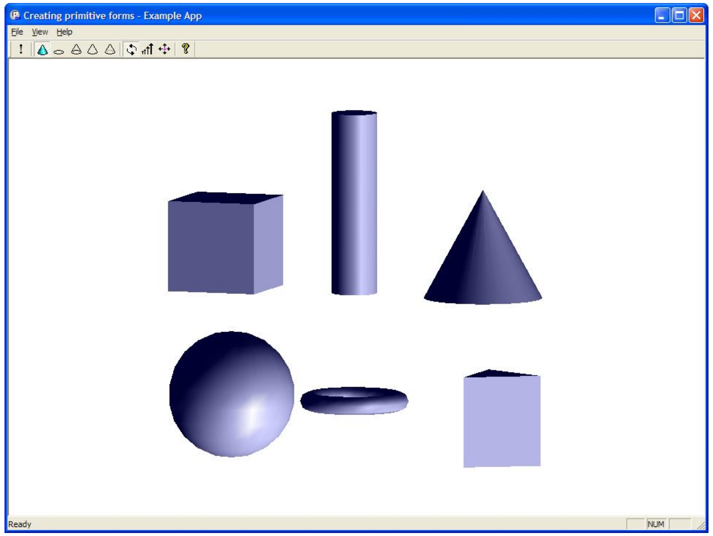


Primitives – Solid body

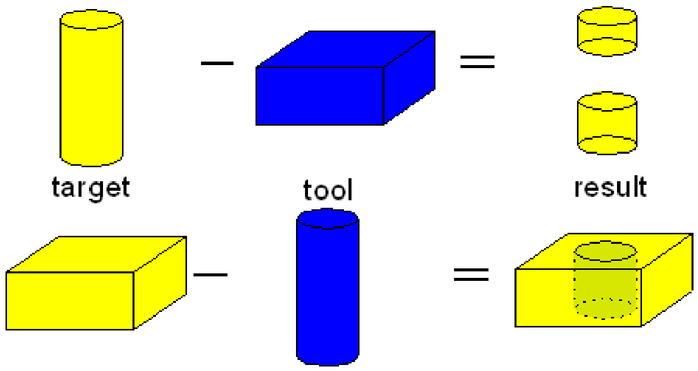
- Step 8~13

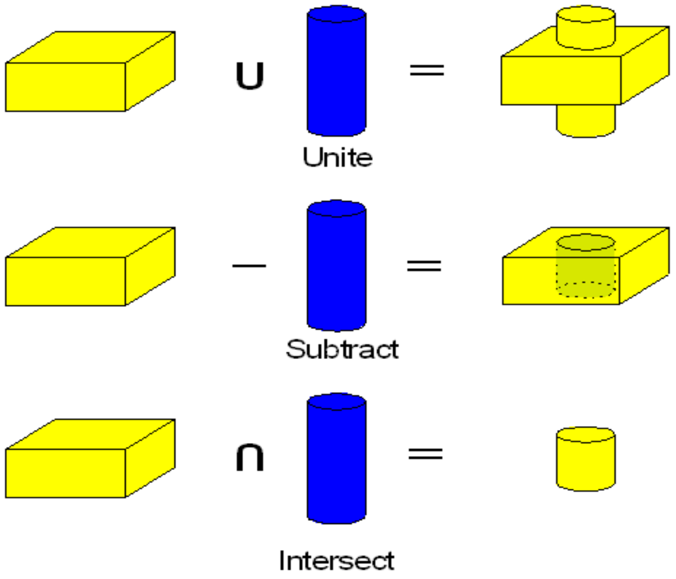
8. Create a block
9. Create a cylinder
10. Create a cone
11. Create a sphere
12. Create a torus
13. Create a prism

```
PK_BODY_create_solid_block (
--- received arguments ---
double x, --- block extent in local x direction (>0)
double y, --- block extent in local y direction (>0)
double z, --- block extent in local z direction (>0)
const PK_AXIS2_sf_t *basis_set, --- position and
orientation (may be NULL)
 --- returned arguments ---
PK_BODY_t *const body --- solid body returned
)

PK_BODY_create_solid_sphere
PK_BODY_create_solid_torus
PK_BODY_create_solid_cyl
PK_BODY_create_solid_cone
PK_BODY_create_solid_prism
```



Boolean





Target & Tool

The target is modified by the tool, and the tool is deleted at the end of the operation.

Global Booleans (PK_BODY_boolean_2)

Comparison of all face pairs from the target and tool bodies

The functions without options

PK_BODY_unite_bodies

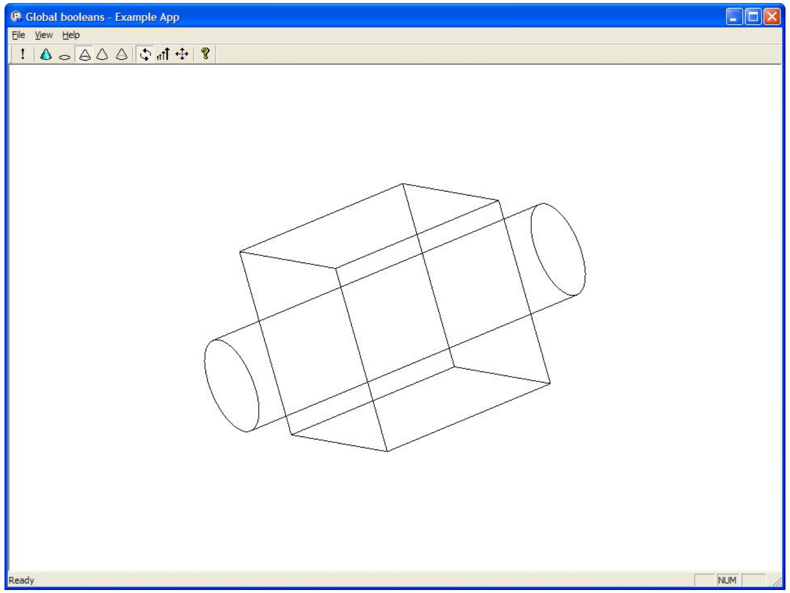
PK_BODY_subtract_bodies

PK_BODY_intersect_bodies

Boolean – Boolean operations

- Step 1 - Create a block and a cylinder

```
PK_SESSION_ask_curr_partition (
 --- returned arguments ---
PK_PARTITION_t *const partition --- current partition
)
```



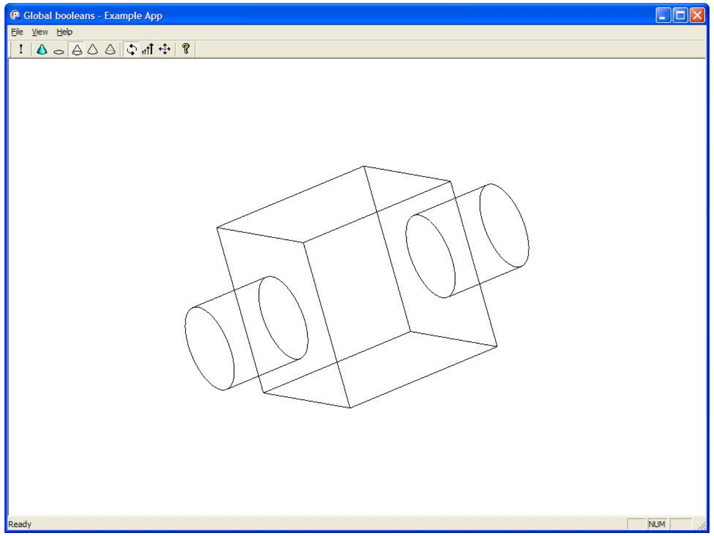
```
PK_PARTITION_make_pmark (
 --- received arguments ---
PK_PARTITION_t partition, --- partition
--- returned arguments ---
PK_PMARK_t *const pmark --- partition mark
)
```

```
PK_PMARK_goto (
 --- received arguments ---
PK_PMARK_t pmark, --- pmark to go to
 --- returned arguments ---
......
)
```

Boolean – Boolean operations

Step 2 - Unite the two bodies

```
PK_BODY_boolean_2 (
 --- received arguments ---
PK_BODY_t target, --- body to receive message
int n_tools, --- number of tool bodies
const PK_BODY_t tools[], --- tool bodies
const PK_BODY_boolean_o_t *options, --- boolean options
 --- returned arguments ---
PK_TOPOL_track_r_t *const tracking, --- tracking information
PK_boolean_r_t *const results --- boolean results
)
```



```
PK_BODY_boolean_o_m( opts );
PK_BODY_boolean_2( block, 1, &cylinder, &opts, &tracking, &results );
PK_TOPOL_track_r_f(&tracking);
PK_boolean_r_f(&results );
```

Boolean – Boolean operations

Step 3 - Rollback

PK_PMARK_goto (

 --- received arguments ---

PK_PMARK_t pmark, --- pmark to go to

 --- returned arguments ---

int *const n_new,

PK_ENTITY_t **const new_entities, --- entities created by roll operation

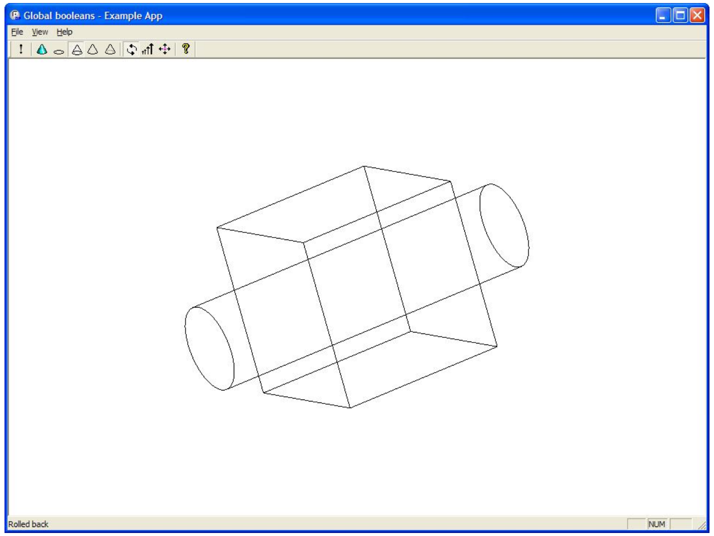
int *const n_mod,

PK_ENTITY_t **const mod_entities, --- entities modified by roll operation

int *const n_del,

int **const del_entities --- entities deleted by roll operation

)

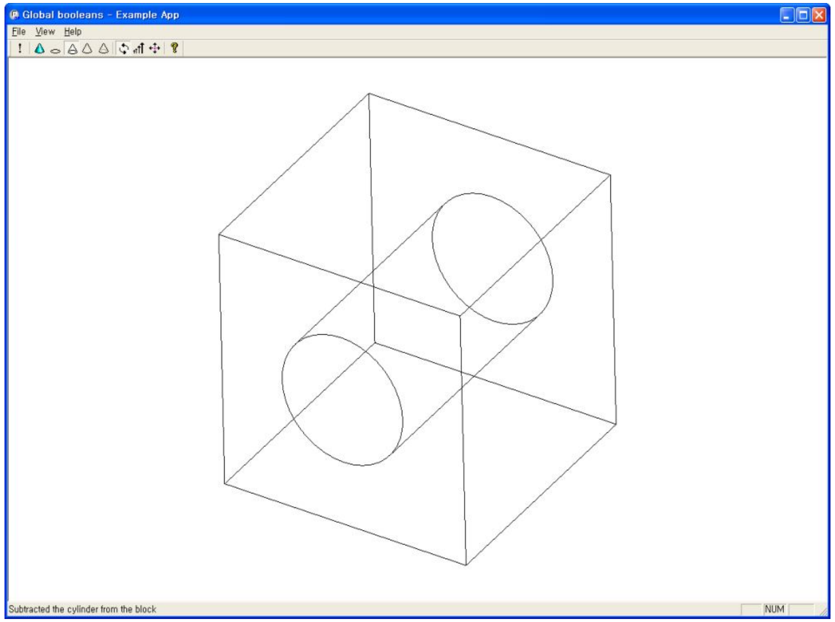


Rolled back

Boolean – Boolean operations

Step 4 - Subtract the bodies

```
PK_BODY_boolean_o_m( opts );

opts.function = PK_boolean_subtract_c;

PK_BODY_boolean_2( block, 1, &cylinder, &opts, &tracking, &results );
```

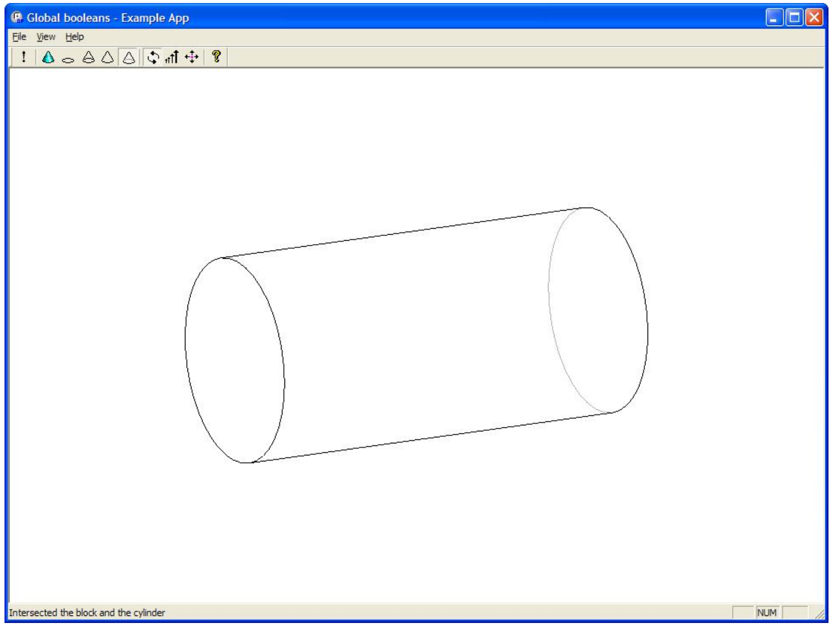


Subtracted the cylinder from the block

Boolean – Disjoint target

Step 5, 6 - Rollback and intersect the bodies

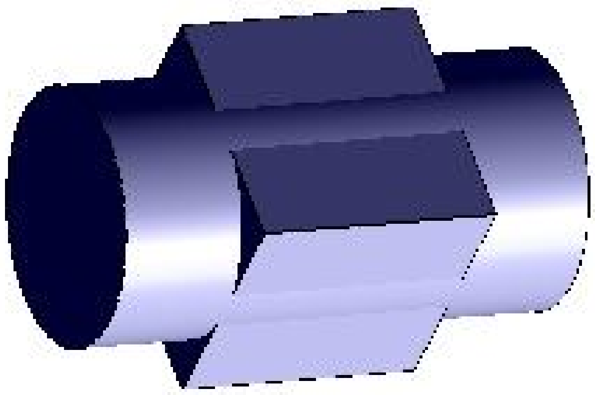
```
PK_BODY_boolean_o_m( opts );

opts.function = PK_boolean_intersect_c;

PK_BODY_boolean_2( block, 1, &cylinder, &opts, &tracking, &results );
```



Intersected the block and the cylinder

Boolean – Disjoint target

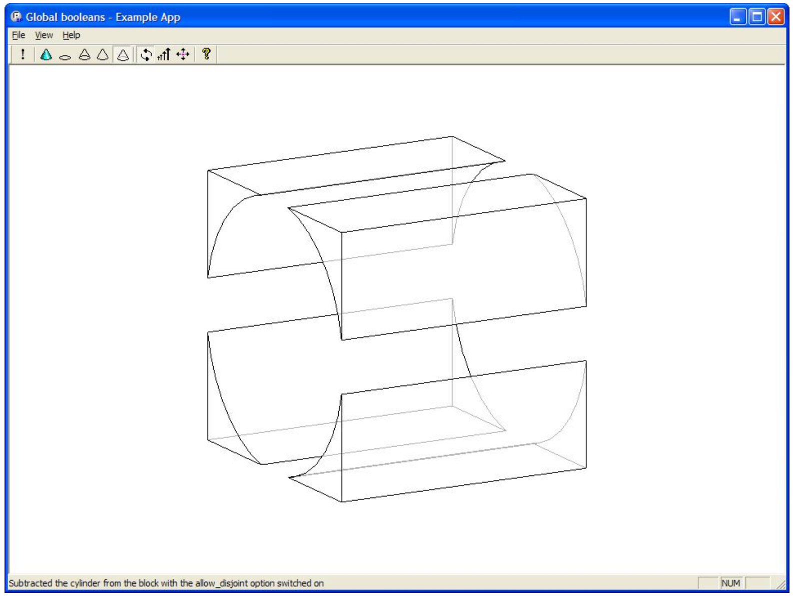
- Step 7 - Delete the cylinder and create a bigger cylinder



Boolean – Disjoint target

Step 8 - Subtract the bodies, creating a disjoint result

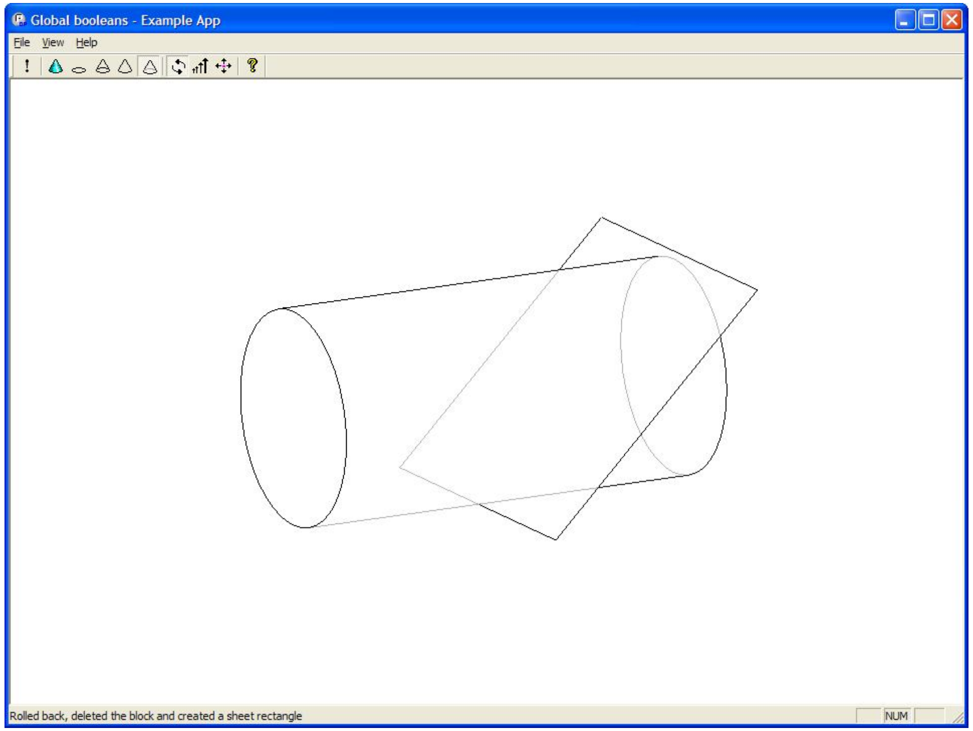
```
PK_BODY_boolean_o_m( opts );

opts.function = PK_boolean_subtract_c;

opts.allow_disjoint = PK_LOGICAL_true;

PK_BODY_boolean_2( block, 1, &cylinder, &opts, &tracking, &results );
```



Subtracted the cylinder from the block with the allow_disjoint option switched on

Boolean – Fence options and sheet punching

- Step 9 - Delete the block and create a sheet

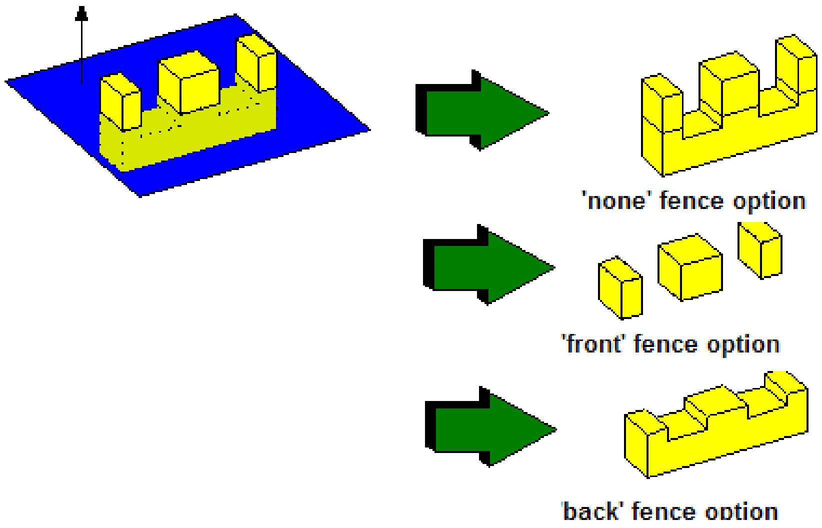


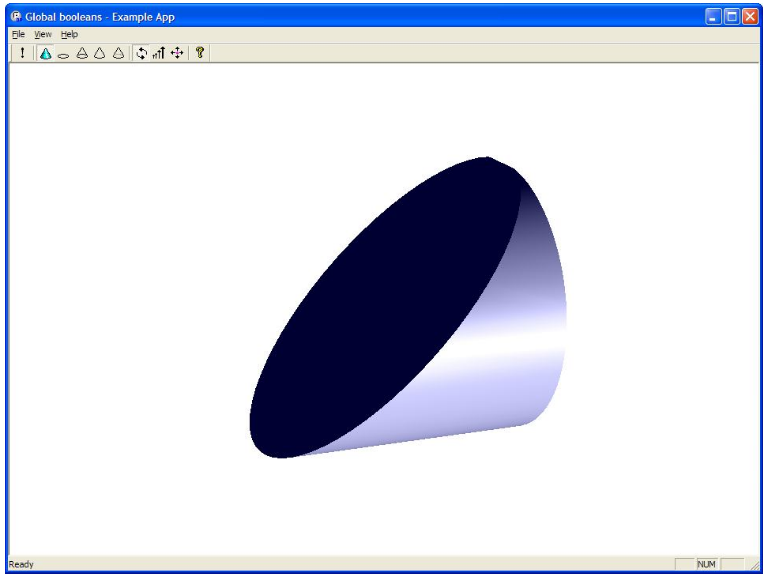
Rolled back, deleted the block and created a sheet rectangle

Boolean – Fence options and sheet punching

Step 10 - Subtract the sheet using "back fence" option

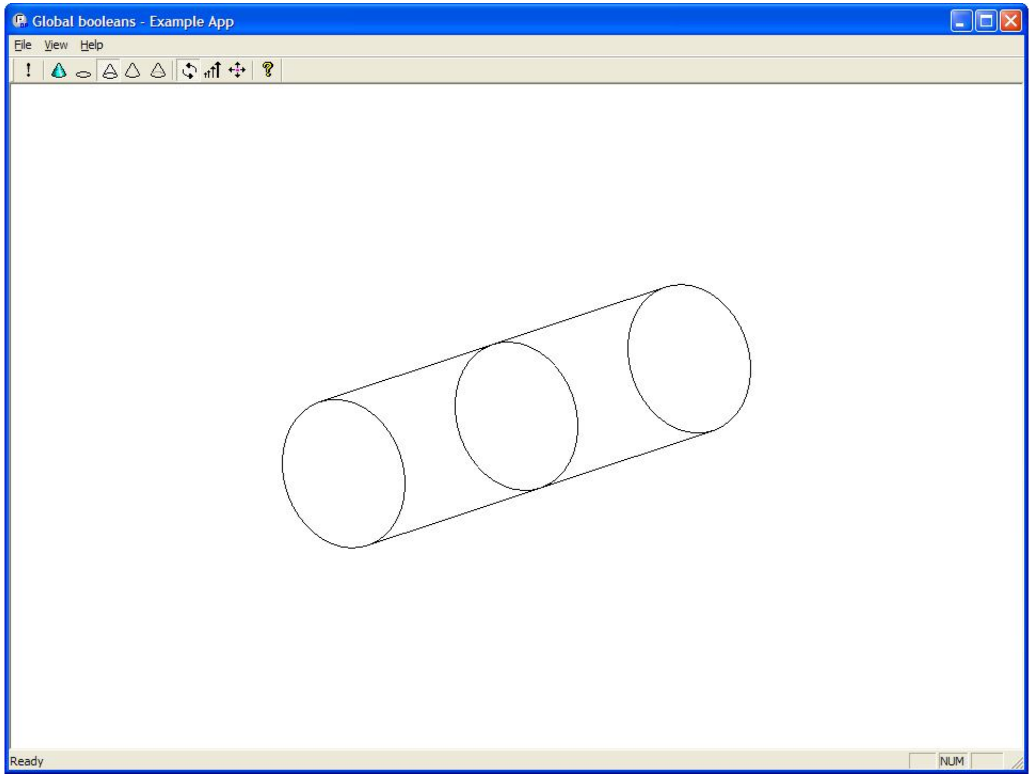
```
PK_BODY_boolean_o_m( opts );
opts.function = PK_boolean_subtract_c;
opts.fence = PK_boolean_fence_back_c;

PK_BODY_boolean_2( cylinder, 1, &rectangle, &opts, &tracking, &results );
```





Boolean - Merge imprinted

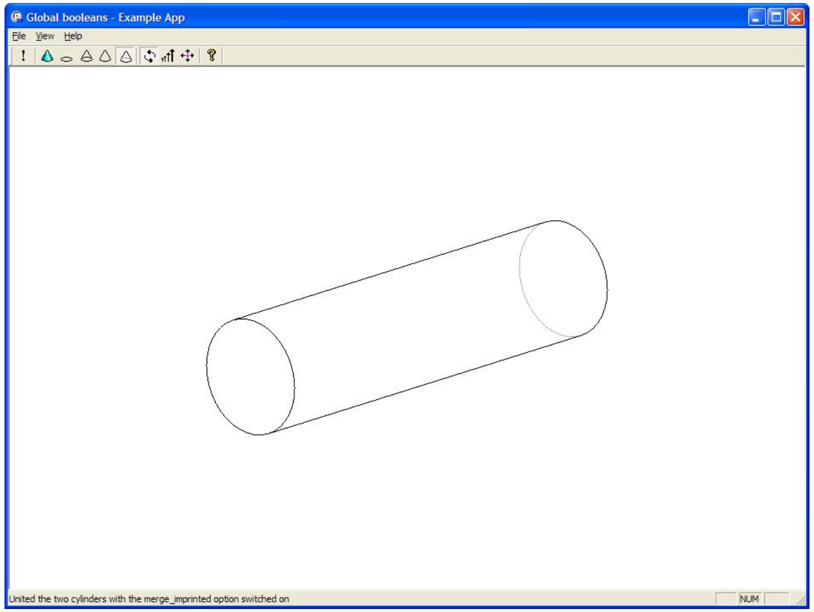
- Step 11 - Delete sheet and create a second cylinder



Boolean - Merge imprinted

Step 12 - Unite, merging imprinted edges

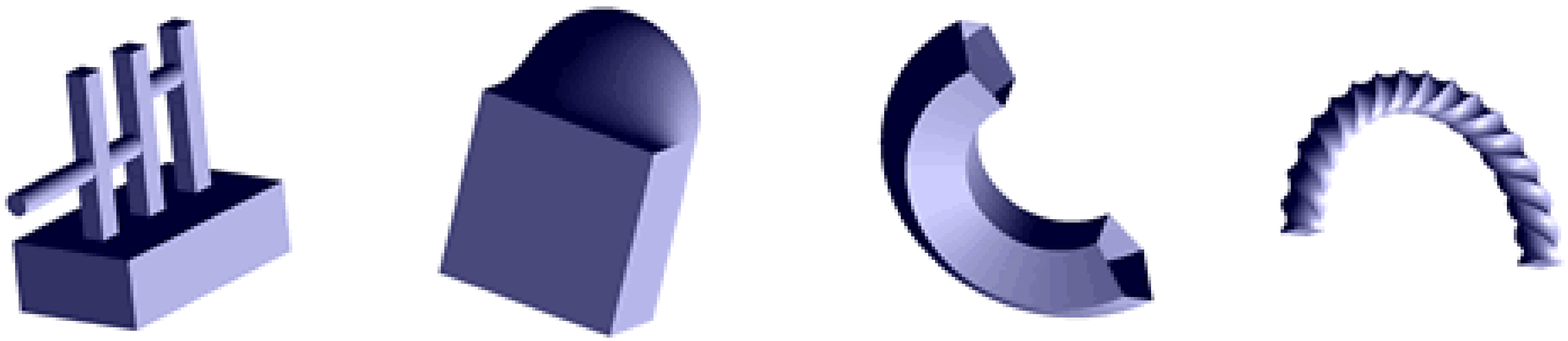
```
PK_BODY_boolean_o_m( opts );

opts.merge_imprinted = PK_LOGICAL_true;

PK_BODY_boolean_2( cylinder, 1, &cylinder_2, &opts, &tracking, &results );
```



United the two cylinders with the merge_imprinted option switched on

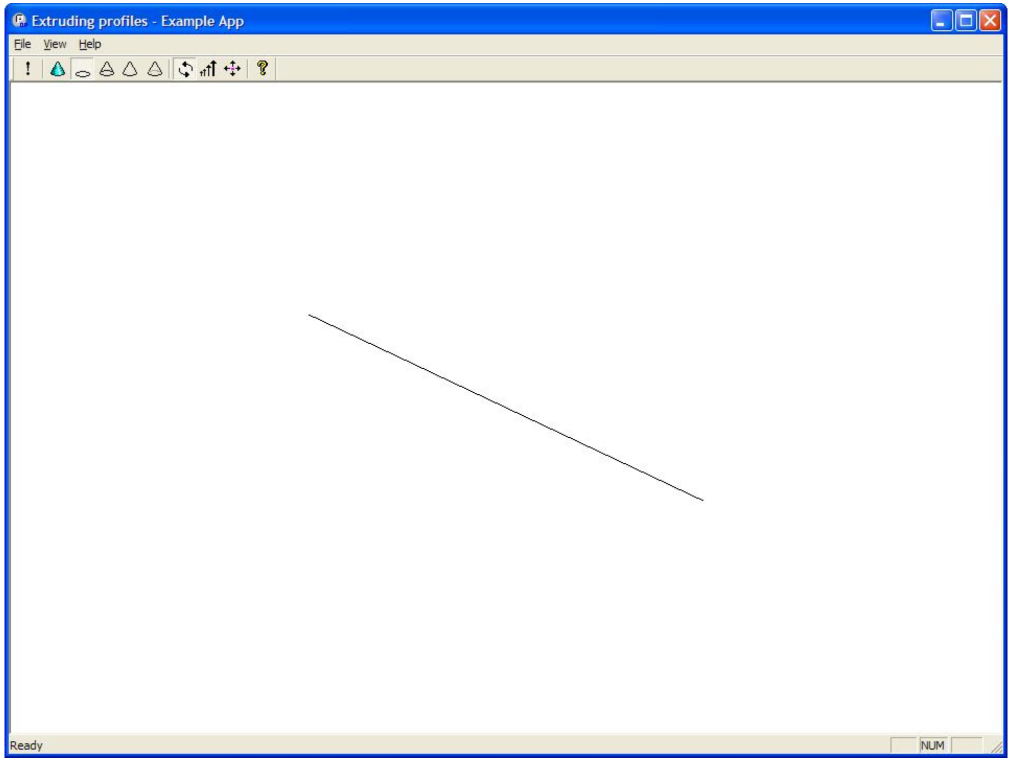
Profiling

- Extrude
- Loft
- Spin
- Sweep



Profiling – Extrude

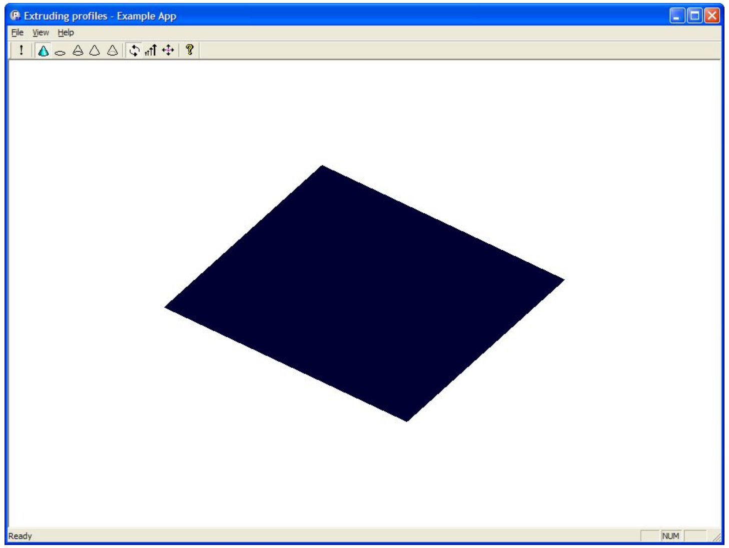
- Step 1 - Create a wire body



Profiling – Extrude

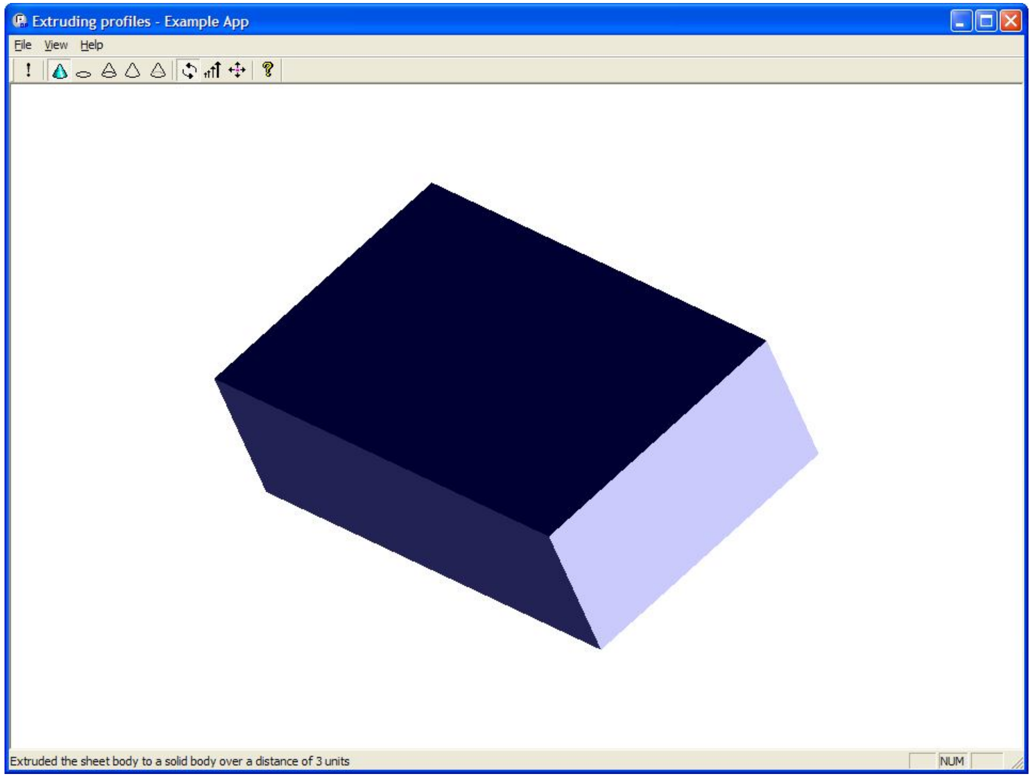
Step 2 - Extrude the wire to a sheet

```
PK_BODY_extrude (
 --- received arguments ---
PK_BODY_t profile, --- minimum, wire or sheet profile --- to extrude
PK_VECTOR1_t path, --- direction of linear extrusion
const PK_BODY_extrude_o_t *options, --- options structure
 --- returned arguments ---
PK_BODY_t *const body, --- resulting extruded body
PK_TOPOL_track_r_t *const tracking, --- tracking information
PK_TOPOL_local_r_t *const results --- status information
)
```



Profiling – Extrude

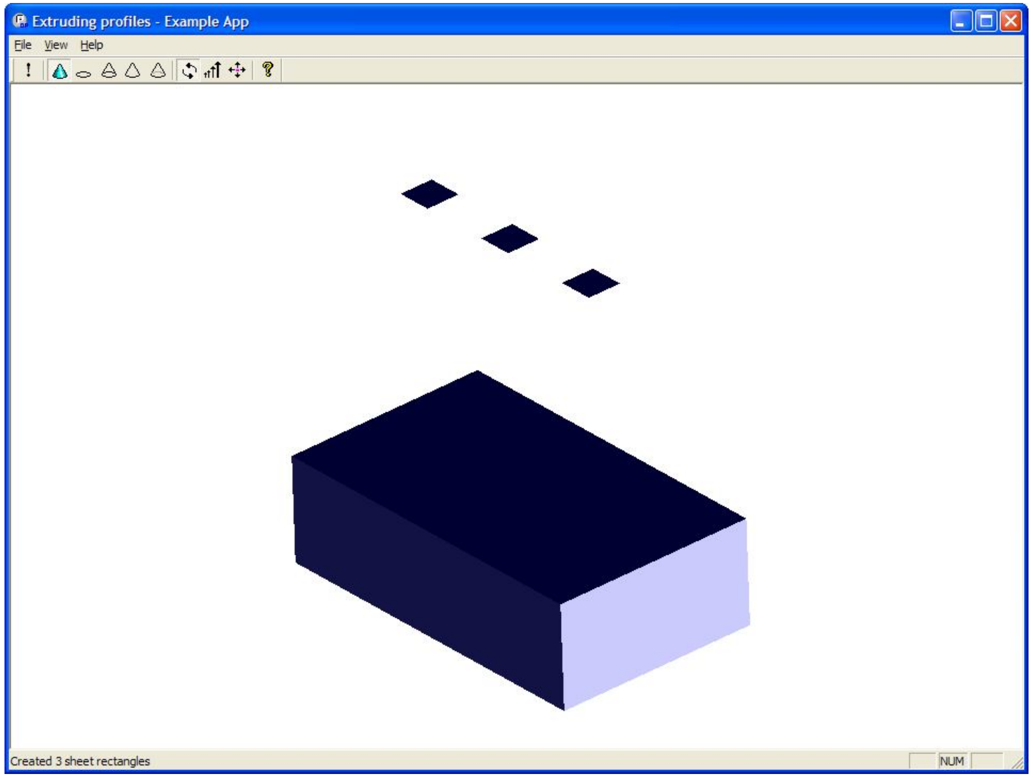
- Step 3 - Extrude the sheet to a block



Extruded the sheet body to a solid body over a distance of 3 units

Profiling – Extrude

- Step 4 - Create three sheet rectangles

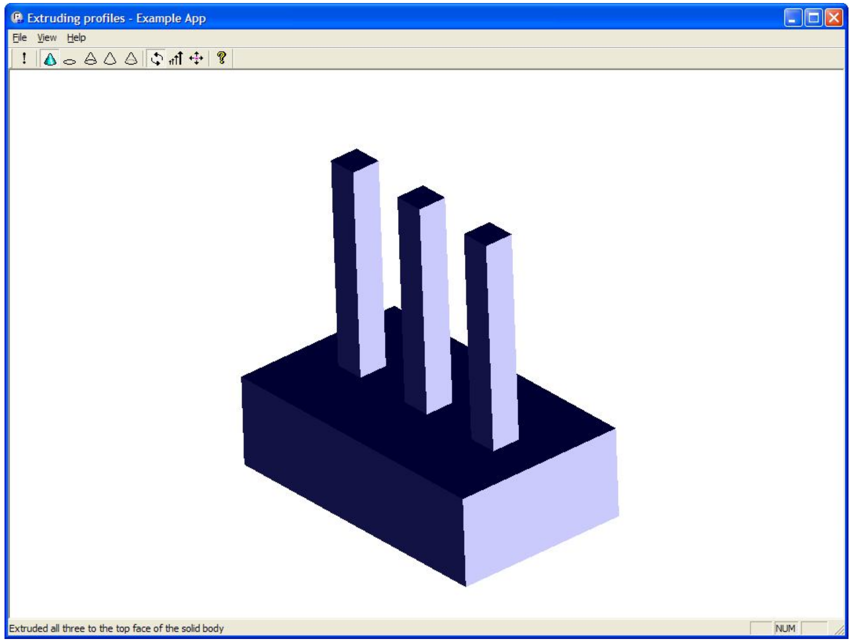


Created 3 sheet rectangles

Profiling – Extrude

- Step 5 - Extrude each rect. to the top face of the solid

```
PK_BODY_extrude_o_m( extrude_opts );
extrude_opts.end_bound.bound = PK_bound_face_c ;
extrude_opts.end_bound.entity = face;

PK_BODY_extrude(sheet_rectangle[0], path, &extrude_opts, &extruded_body, &tracking, &results);
```

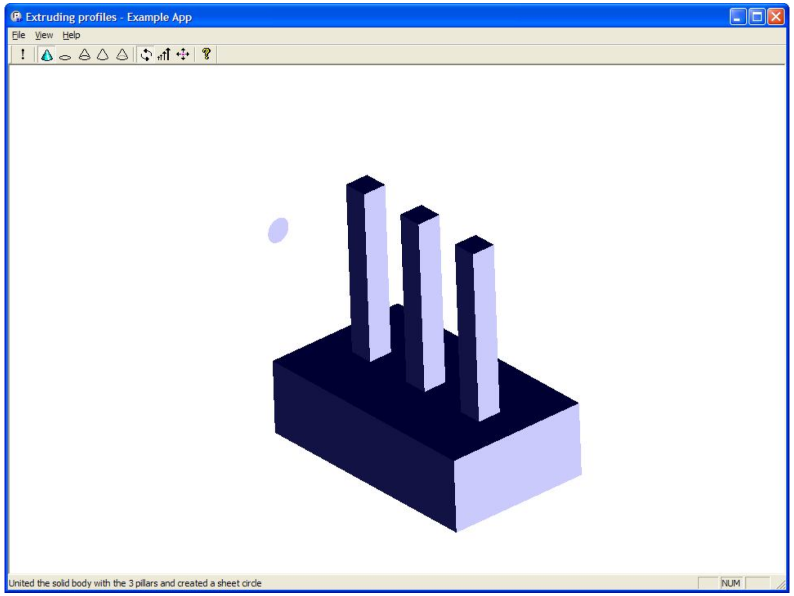


Extruded all three to the top face of the solid body

Profiling – Extrude

Step 6 - Create a sheet circle

```
PK_BODY_unite_bodies (
 --- received arguments ---
PK_BODY_t target, --- Body to receive message
int n_tools, --- Number of tool bodies
const PK_BODY_t tools[], --- Tool bodies
 --- returned arguments ---
int *const n_bodies, --- Number of resultant bodies
PK_BODY_t **const bodies --- Resultant bodies
)
```

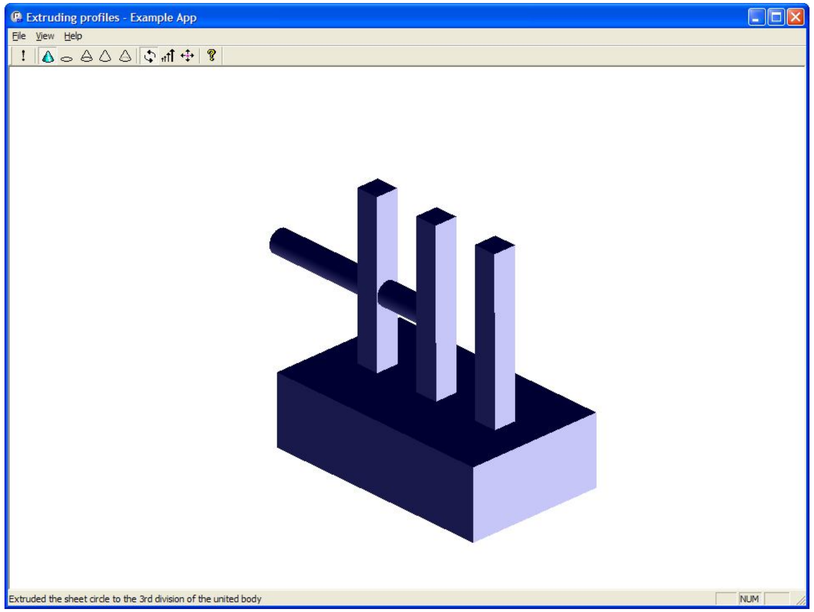


United the solid body with the 3 pillars and created a sheet circle

Profiling – Extrude

- Step 7 - Extrude circle to 3rd division of body

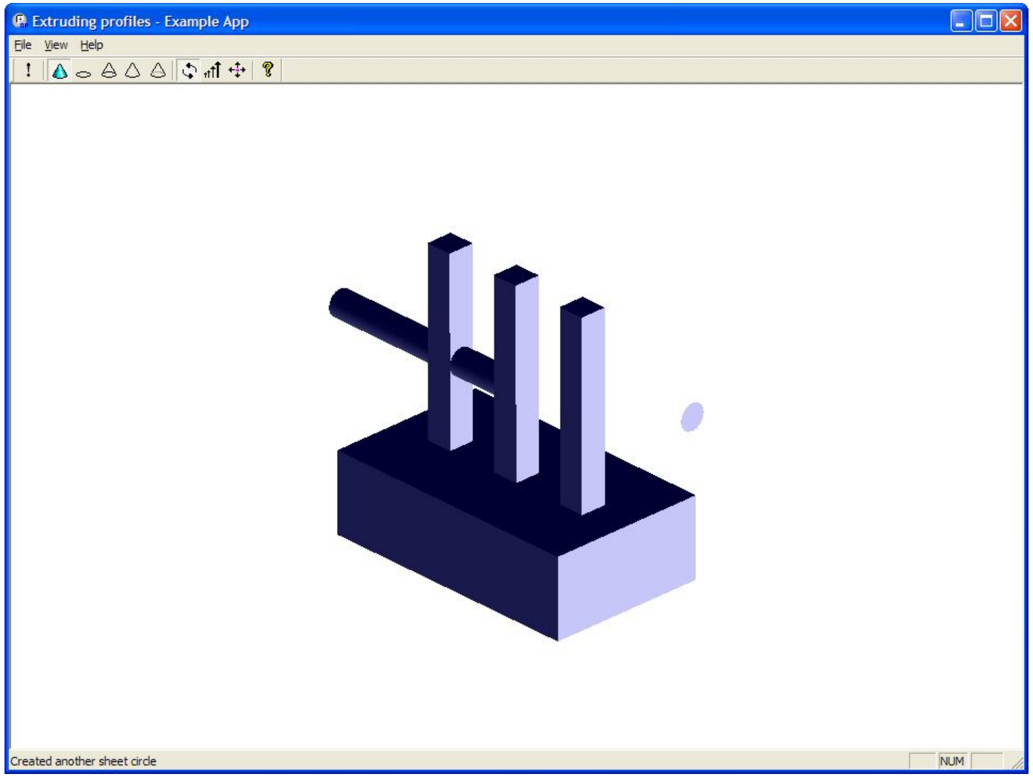
```
PK_BODY_extrude_o_m( extrude_opts );
extrude_opts.end_bound.bound = PK_bound_body_c ;
extrude_opts.end_bound.entity = united_body[0];
extrude_opts.end_bound.nth_division = 3;

PK_BODY_extrude(circle, path, &extrude_opts, &extruded_body, &tracking, &results);
```



Extruded the sheet circle to the 3rd division of the united body

Profiling – Extrude

- Step 8 - Create a second circle

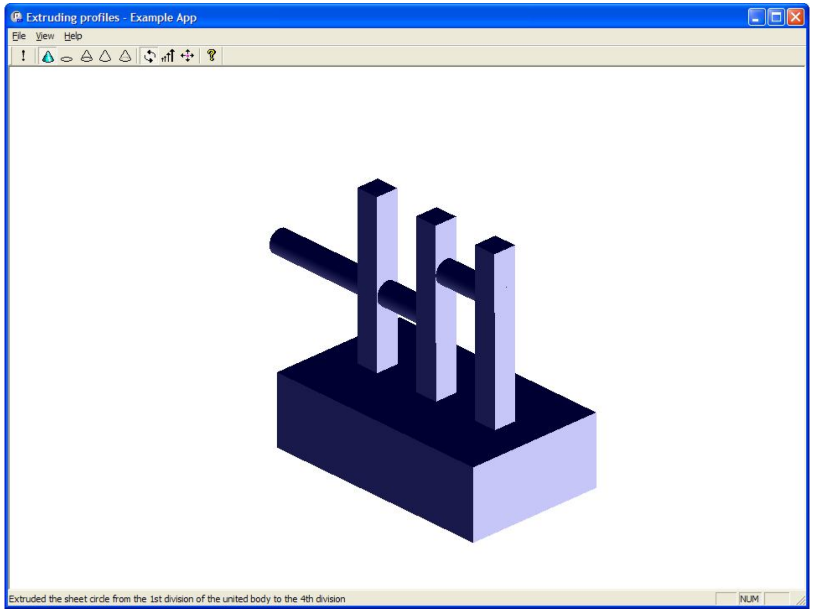


Created another sheet circle

Profiling – Extrude

Step 9 - Extrude circle between 1st and 4th divisions

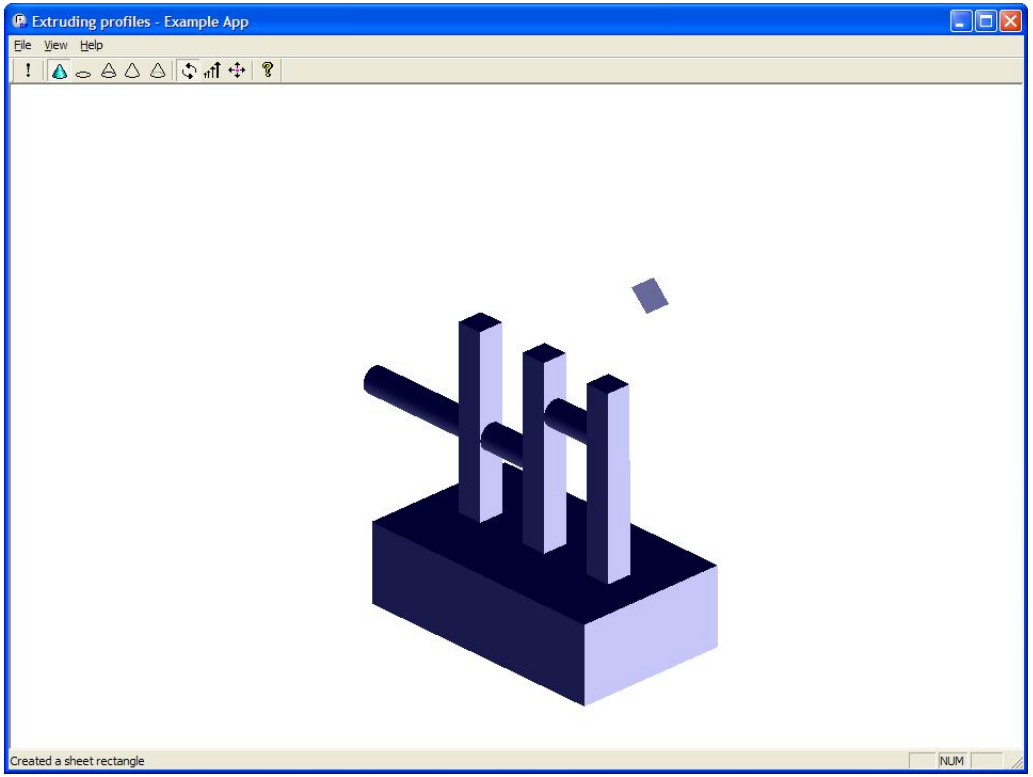
```
PK_BODY_extrude_o_m( extrude_opts );

extrude_opts.start_bound.bound = PK_bound_body_c ;
extrude_opts.start_bound.entity = united_body[0];
extrude_opts.start_bound.nth_division = 1;
extrude_opts.end_bound.bound = PK_bound_body_c ;
extrude_opts.end_bound.entity = united_body[0];
extrude_opts.end_bound.nth_division = 3;

PK_BODY_extrude(circle, path, &extrude_opts,
&extruded_body, &tracking, &results);
```



Extruded the sheet circle from the 1st division of the united body to the 4th division

Profiling – Extrude

- Step 10 - Create a rectangle

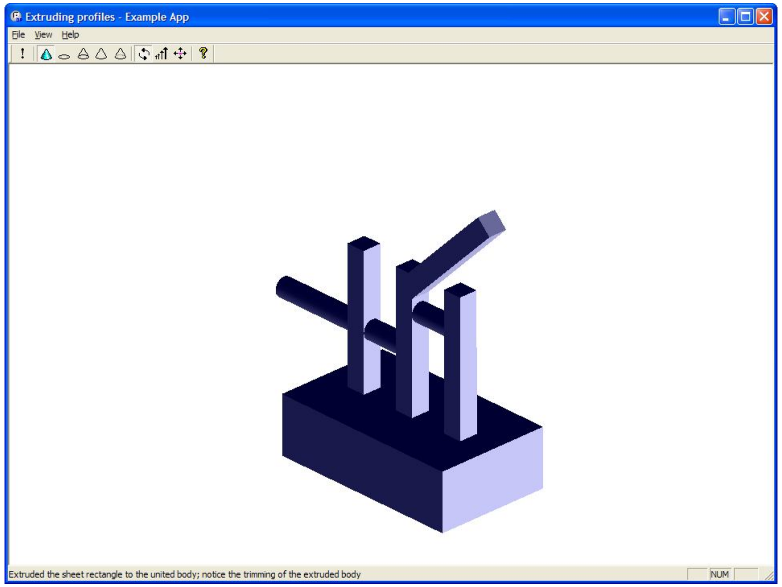


Created a sheet rectangle

Profiling – Extrude

Step 11 - Extrude rectangle to body

```
path.coord[0] = -0.70710678118654752440084436210485;
path.coord[1] = -0.70710678118654752440084436210485;
path.coord[2] = 0.;

PK_BODY_extrude_o_m( extrude_opts );
extrude_opts.end_bound.bound = PK_bound_body_c ;
extrude_opts.end_bound.entity = united_body[0];

PK_BODY_extrude(rectangle, path, &extrude_opts,
&extruded_body, &tracking, &results);
```

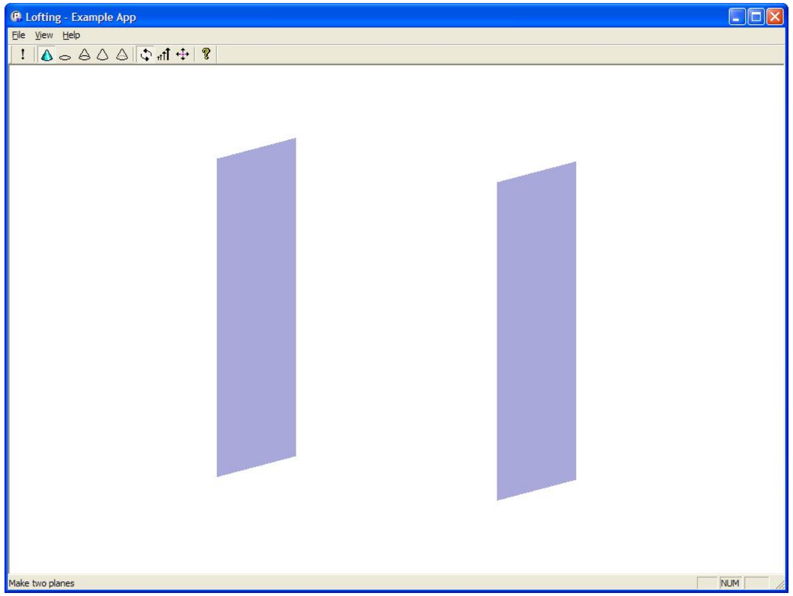


Extruded the sheet rectangle to the united body; notice the trimming of the extruded body

Profiling – Loft ; Simple Loft

- Step 1 - Receive two planar sheets

```
PK_PART_find_entity_by_ident (
 --- received arguments ---
PK_PART_t part, --- part in which to search for entity
PK_CLASS_t class, --- class of entity
int identifier, --- identifier of entity
 --- returned arguments ---
PK_ENTITY_t *const entity --- entity (may be NULL)
)
```

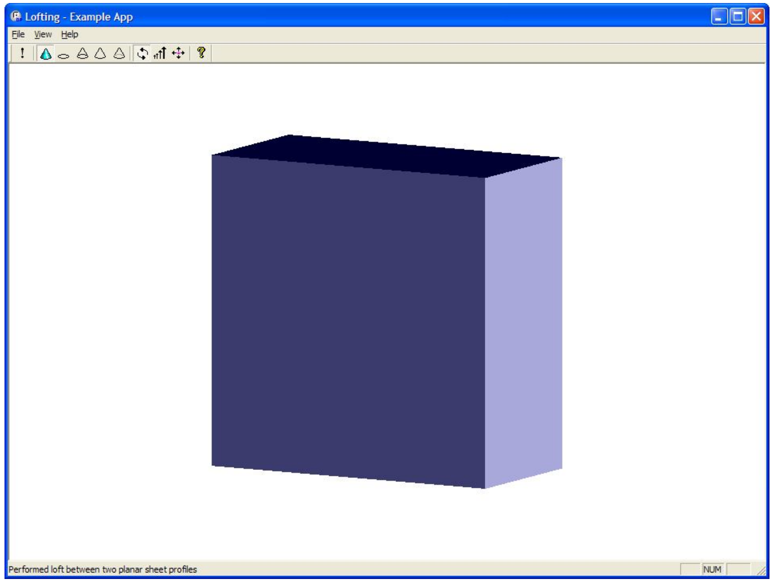


Make two planes

Profiling – Loft ; Simple Loft

- Step 2 - Create solid block by lofting between them

```
PK_BODY_make_lofted_body (
 --- received arguments ---
int n_profiles, --- number of profiles
const PK_BODY_t profiles[], --- profiles to loft
const PK_VERTEX_t start_vertices[],--- start vertices
const PK_BODY_make_lofted_body_o_t *options, ---
options on lofting
 --- returned arguments ---
 PK_BODY_tracked_loft_r_t *const lofted_body --- result
lofted body
)
```

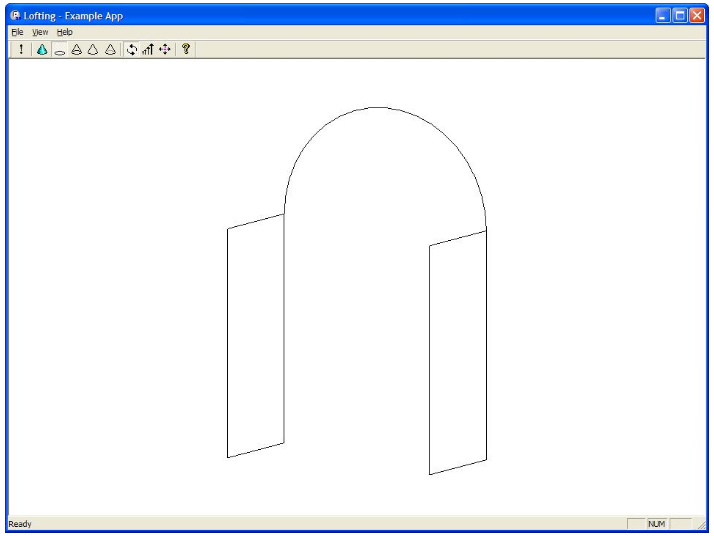


Performed loft between two planar sheet profiles

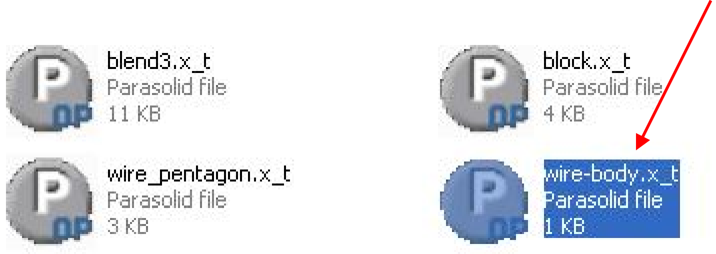
Profiling – Loft ; with Guide Wire

Step 3 - Roll back and read in guide wire

```
PK_PART_receive (
 --- received arguments ---
const char *key, --- key string
const PK_PART_receive_o_t *options, --- receive options
 --- returned arguments ---
int *const n_parts, --- number of parts received
PK_PART_t **const parts --- parts received
)
```



```
PK_PART_receive_o_m( receive_opts );
receive_opts.transmit_format = PK_transmit_format_text_c;
PK_PART_receive( "..\\Example Parts\\wire-body", &receive_opts, &n_parts, &parts );
```



blend3.x_t
Parasolid file
11 KB

block.x_t
Parasolid file
4 KB

wire_pentagon.x_t
Parasolid file
3 KB

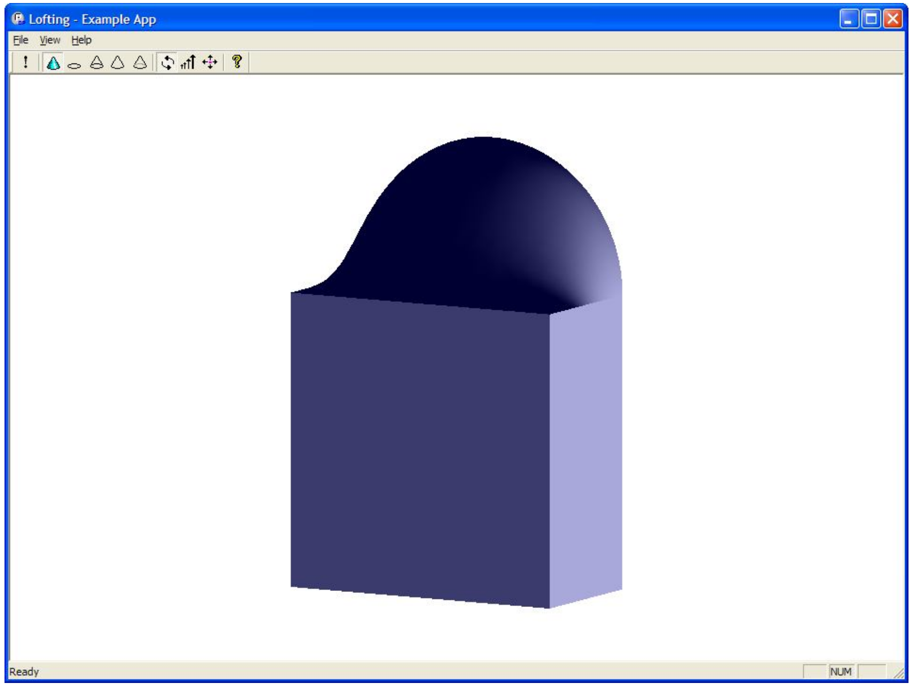
wire-body.x_t
Parasolid file
1 KB

Read and load existing file

Profiling – Loft ; with Guide Wire

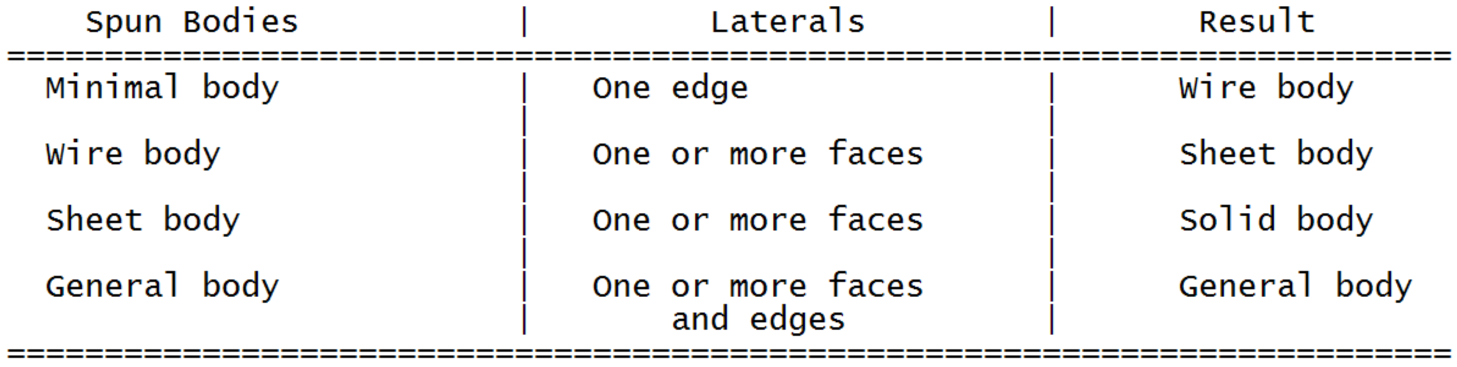
- Step 4 - Perform loft using guide wire to constrain shape

```
loft_opts.n_guide_wires = 1;
loft_opts.guide_wires = &guide_body;

PK_BODY_make_lofted_body(2, profiles, vertices,
&loft_opts, &loft_track);
```

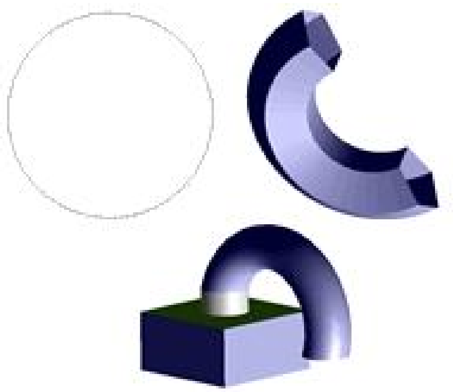


Profiling – Spin

- Step 1



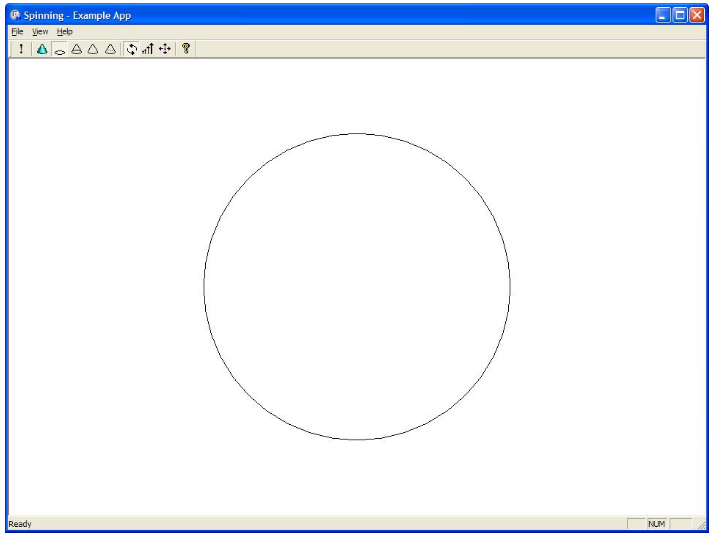
Spun Bodies	Laterals	Result
Minimal body	One edge	Wire body
Wire body	One or more faces	Sheet body
Sheet body	One or more faces	Solid body
General body	One or more faces and edges	General body



Profiling – Spin

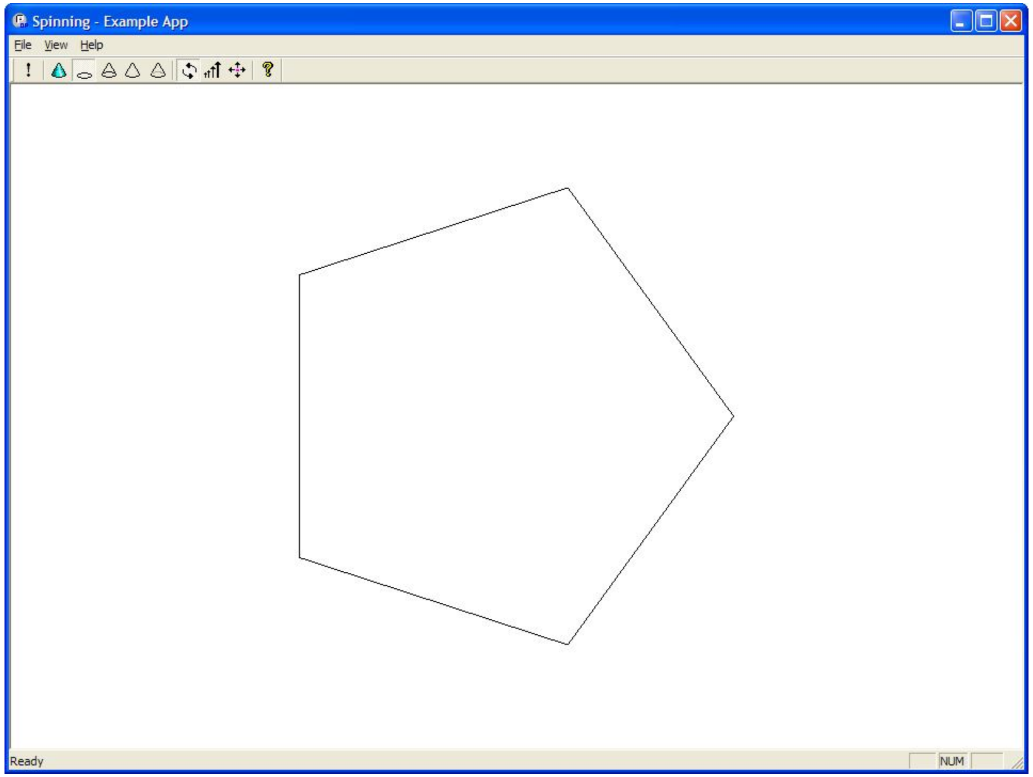
- Step 1 - Create a point and a minimum body
  - Spin minimum body to create wire circle

```
PK_BODY_spin (
--- received arguments ---
PK_BODY_t body, --- minimum, wire or sheet body
const PK_AXIS1_sf_t *axis, --- spin axis
double angle, --- spin angle
PK_LOGICAL_t local_check, --- whether local checking will be done
 --- returned arguments ---
int *const n_laterals, --- number of laterals
PK_TOPOL_t **const laterals, --- new edges of faces
PK_TOPOL_t **const bases, --- entities swept into laterals
PK_local_check_t *const check_result ---result of local check
)
```



Profiling – Spin

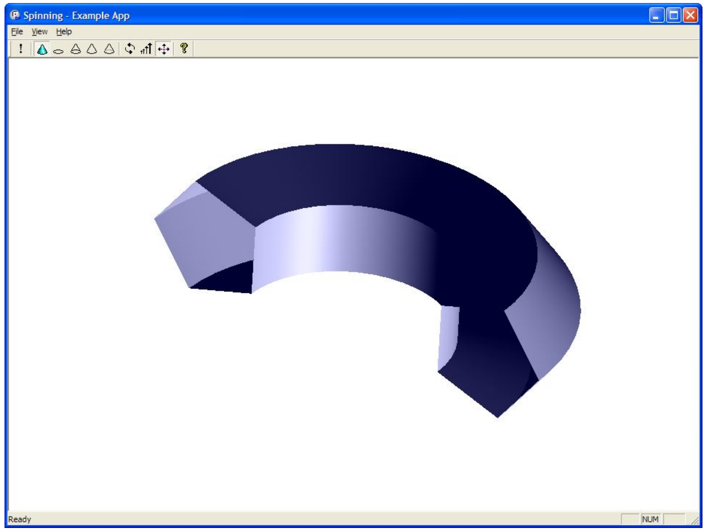
- Step 2 - Receive a wire body



Profiling – Spin

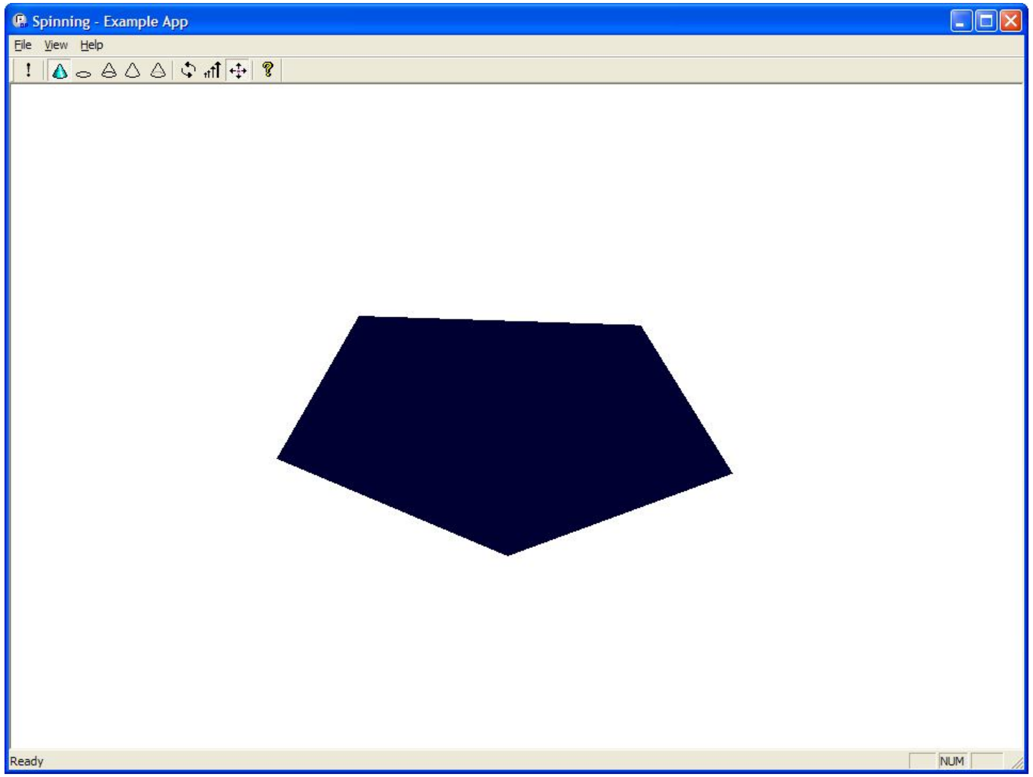
- Step 3 -Spin wire body to create sheet body

```
axis2.location.coord[0] = -10;
axis2.location.coord[1] = 0;
axis2.location.coord[2] = 0;

axis2.axis.coord[0] = 0;
axis2.axis.coord[1] = -1;
axis2.axis.coord[2] = 0;

PK_BODY_spin(wire_body,&axis2,3.14,PK_LOGICAL_true,&n_laterals,&laterals,&bases,&check_result);
```



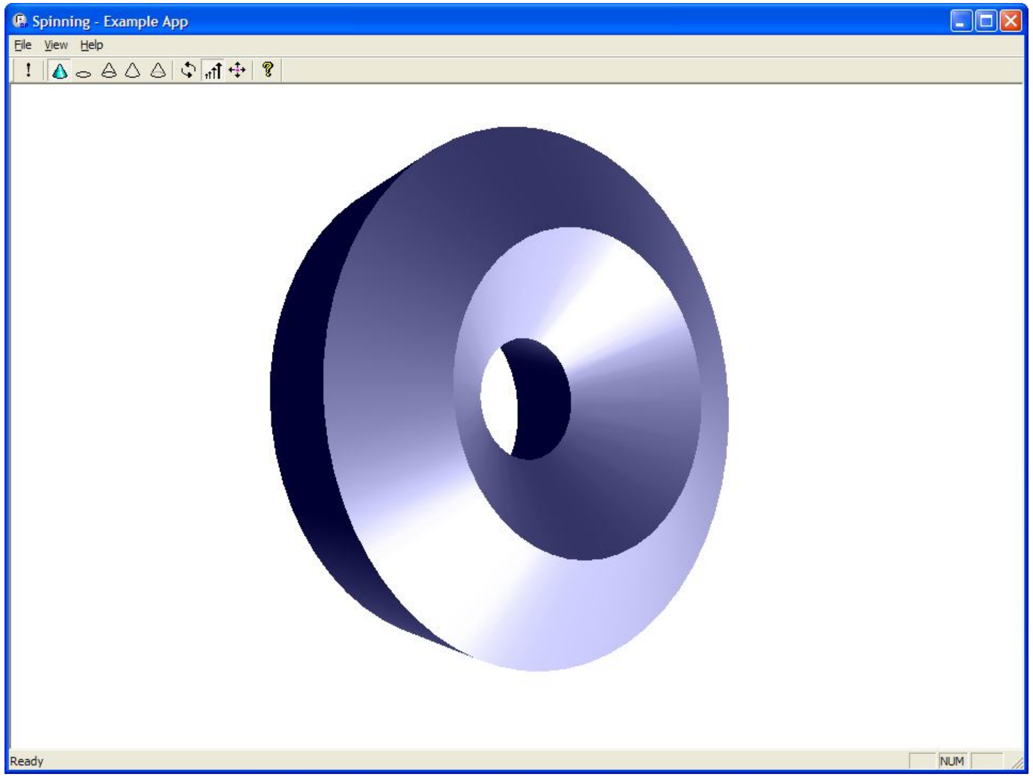
Profiling – Spin

- Step 4 - Create a sheet body



Profiling – Spin

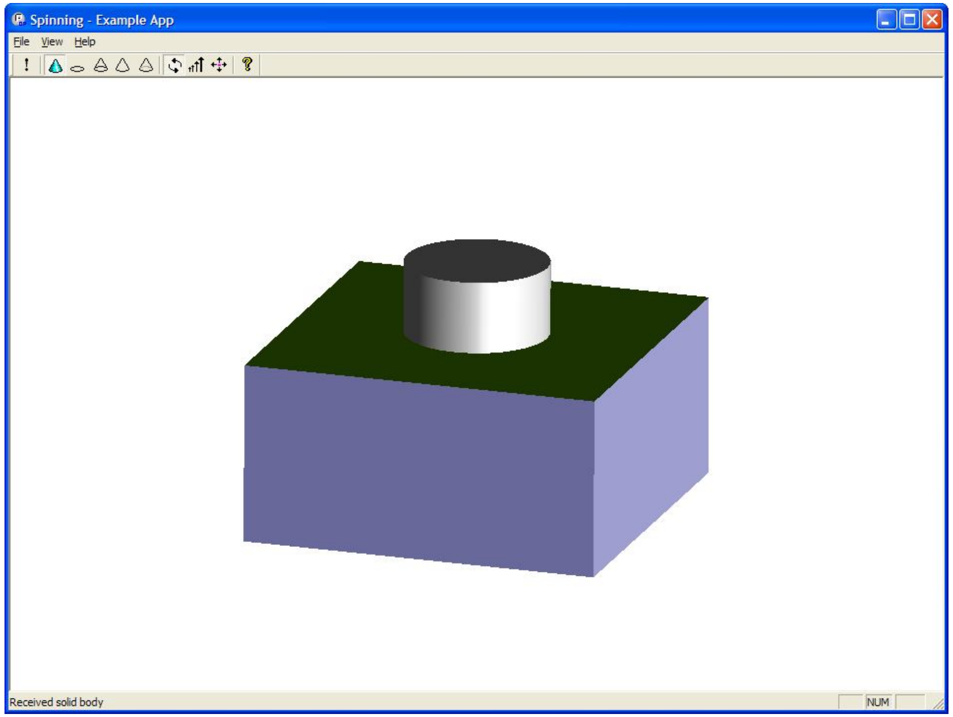
- Step 5 - Spin sheet body to create solid body



Profiling – Spin

- Step 6 - Receive a solid body and identify a face on solid body

```
PK_PART_find_entity_by_ident (*parts, PK_CLASS_face, 89, &face_to_spin);
```

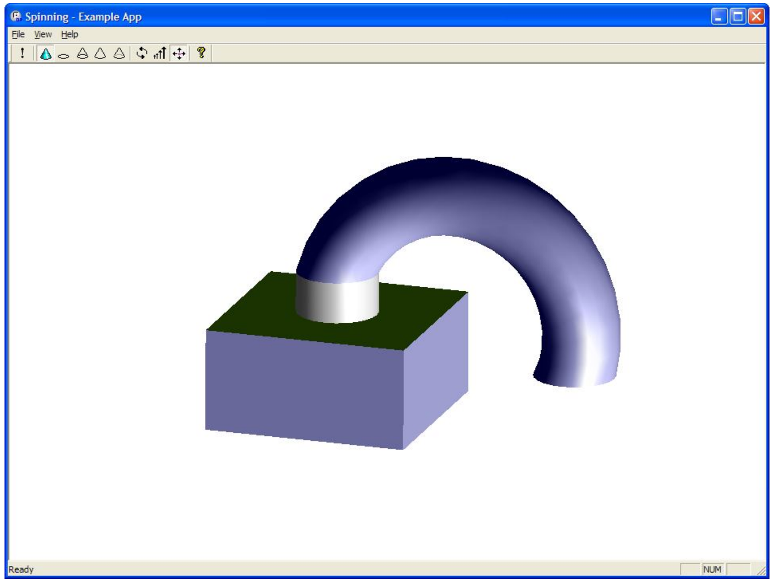


Received solid body

Profiling – Spin

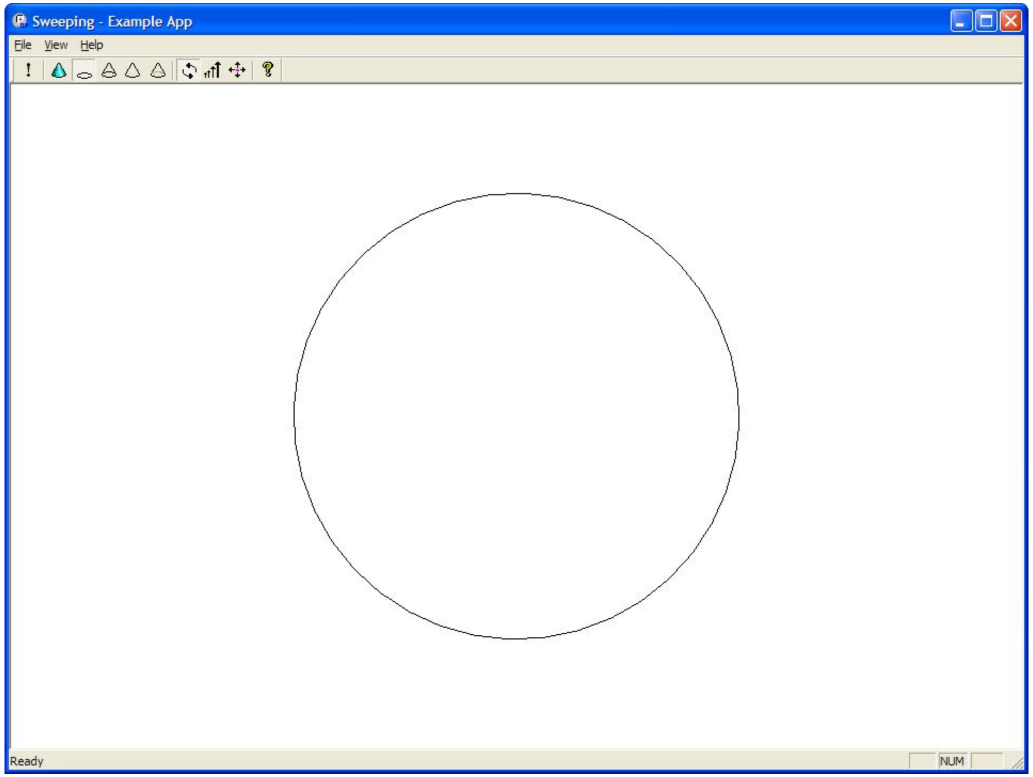
- Step 7 - Spin face on solid body

```
PK_FACE_spin (
 --- received arguments ---
int n_faces, --- number of faces
const PK_FACE_t faces[], --- faces
const PK_AXIS1_sf_t *axis, --- spin axis
double angle, --- spin angle
PK_LOGICAL_t local_check, --- whether local checking will
be done
--- returned arguments ---
int *const n_laterals, --- number of laterals
PK_FACE_t **const laterals, --- new faces (may be NULL)
PK_EDGE_t **const bases, --- edges swept into laterals
(may be NULL)
PK_local_check_t *const check_result --- result of local check
)
```



Profiling – Sweep

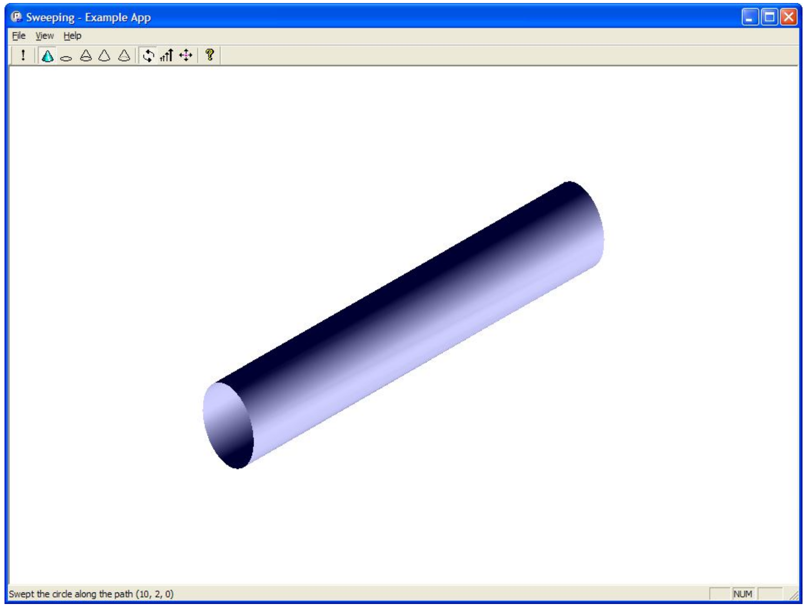
- Step 1 - Create a wire profile



Profiling – Sweep

- Step 2 - Sweep profile along a path

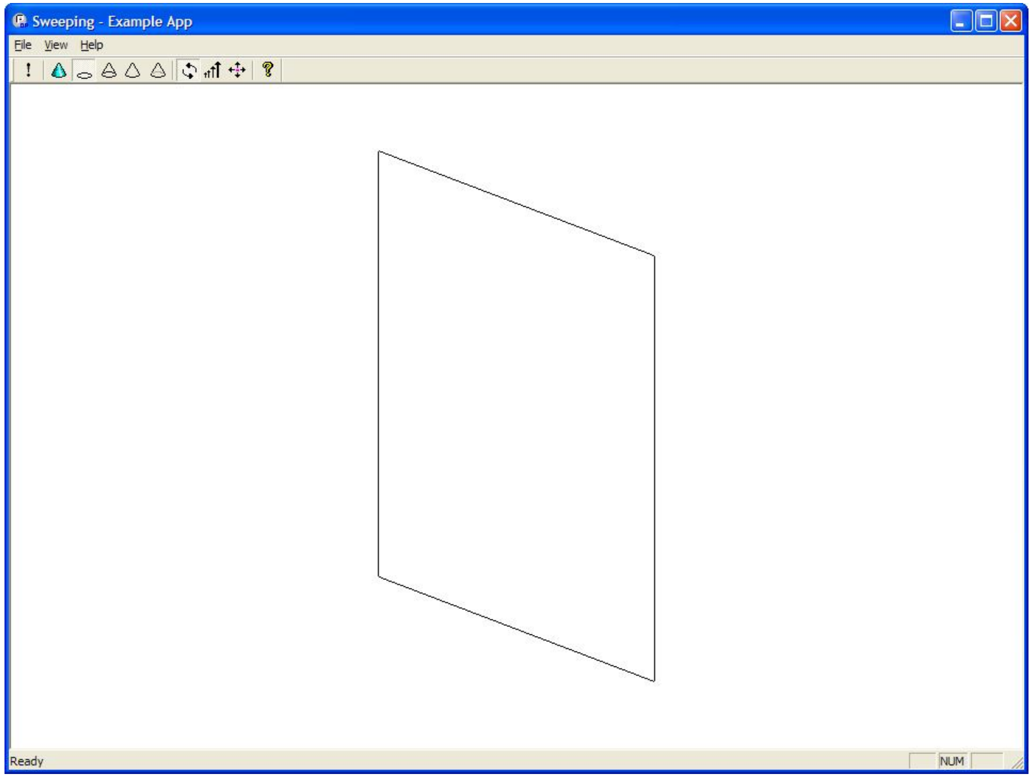
```
PK_BODY_sweep (
 --- received arguments ---
PK_BODY_t body, --- minimum, wire or sheet body
PK_VECTOR_t path, --- translation vector
PK_LOGICAL_t local_check, --- whether local checking
will be done
 --- returned arguments ---
int *const n_laterals, --- number of laterals
PK_TOPOL_t **const laterals, --- new edges or faces
(may be NULL)
PK_TOPOL_t **const bases, --- entities swept into laterals
(may be NULL)
PK_local_check_t *const check_result --- result of local
check
)
```



Swept the circle along the path (10, 2, 0)

Profiling – Sweep

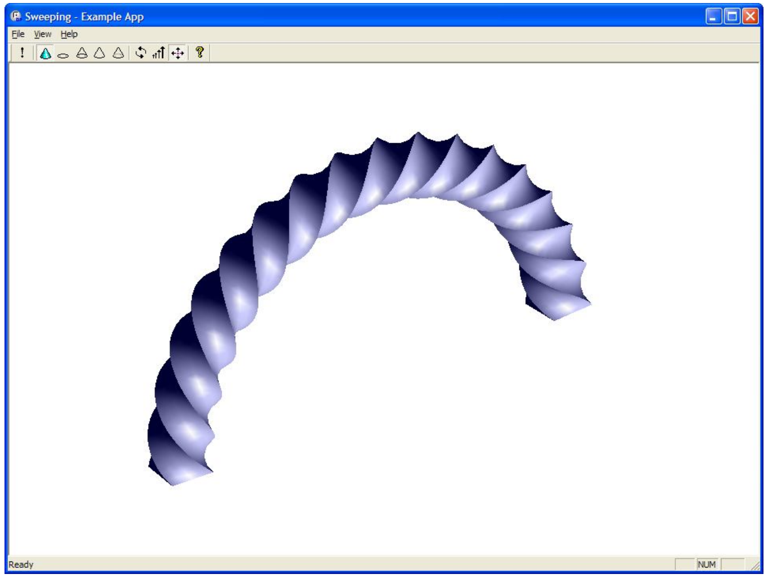
- Step 3 - Create a sheet rectangle profile



Profiling – Sweep

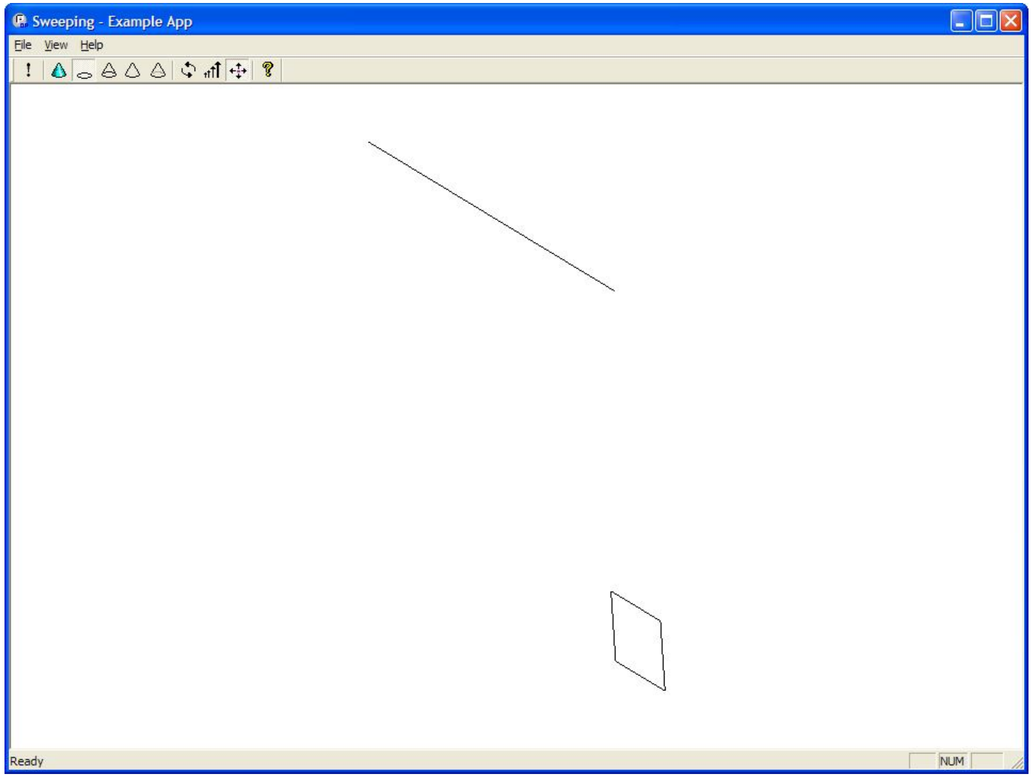
- Step 4 - Sweep profile, applying a twist

```
PK_BODY_sweep_law_t twist;

values[0] = 0.;
values[1] = 25;

PK_BODY_ask_vertices(path_body, &n_vertices, &vertices);
twist.law_type = PK_BODY_sweep_law_discrete_c;
twist.law_set.n_vertices = 2;
twist.law_set.vertices = vertices;
twist.law_set.values = values;

PK_BODY_make_swept_body_o_m( sweep_opts );
sweep_opts.twist = twist;

PK_BODY_make_swept_body(profile_body, path_body,
PK_ENTITY_null, &sweep_opts, &swept_res);
```



Profiling – Sweep

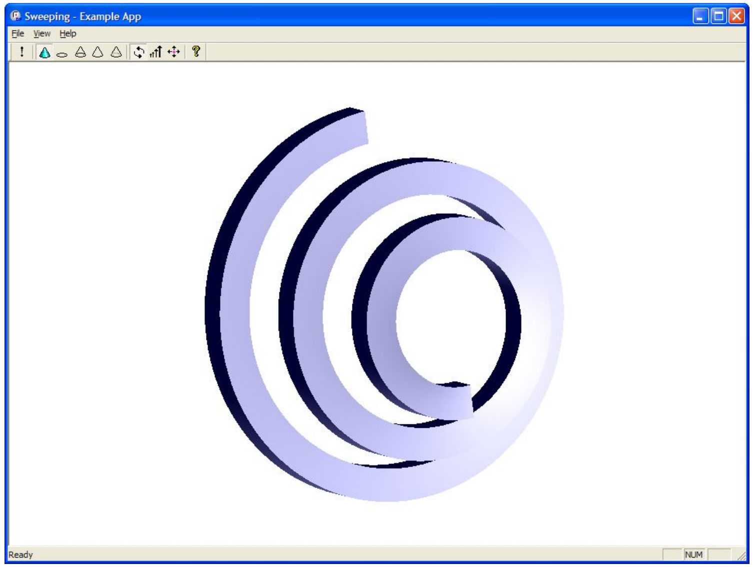
- Step 5 - Create a wire path and a rectangular sheet profile



Profiling – Sweep

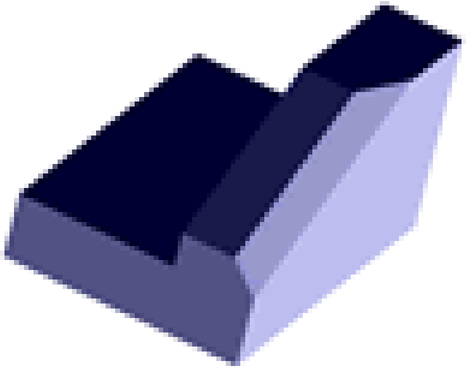
- Step 6 - Sweep profile along path, applying both scale and twist

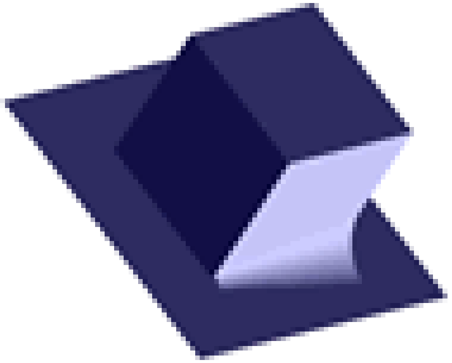
```
PK_BODY_make_swept_body_o_m( sweep_opts );
sweep_opts.twist.law_type = PK_BODY_sweep_law_discrete_c;
sweep_opts.scale.law_type = PK_BODY_sweep_law_discrete_c;
sweep_opts.scale_type  = PK_BODY_sweep_scale_posn_c;
```



Blend

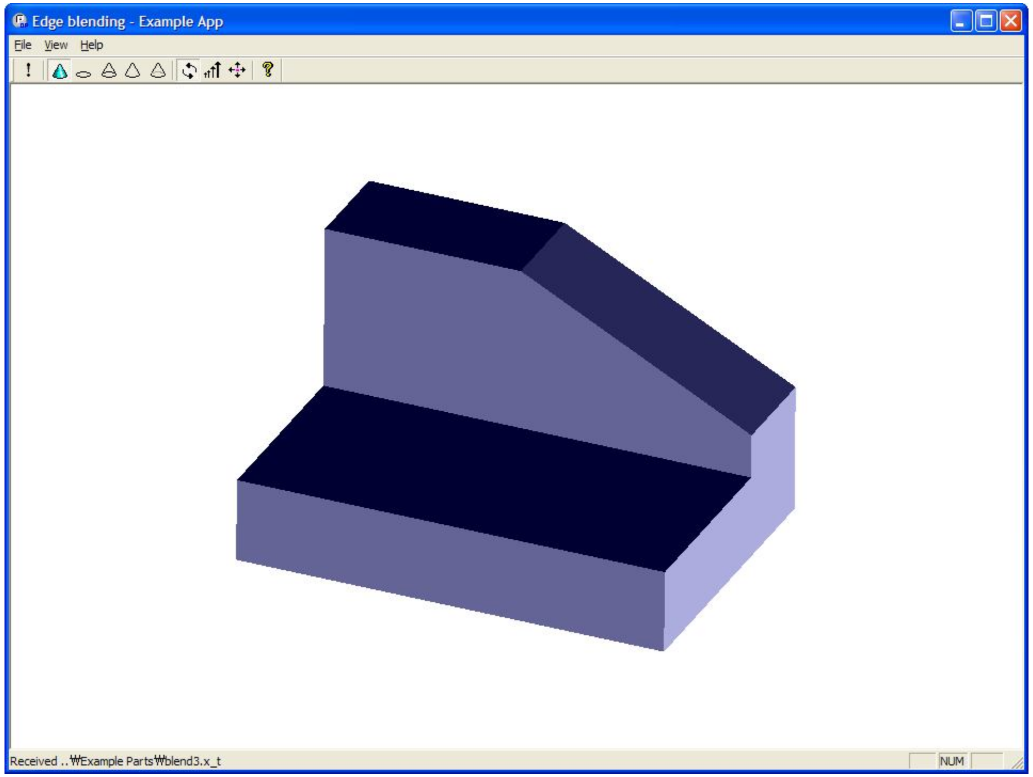
- Edge Blending
- Two-Face Blending





Blend – Edge Blending

- Step 1 - Receive body with notch

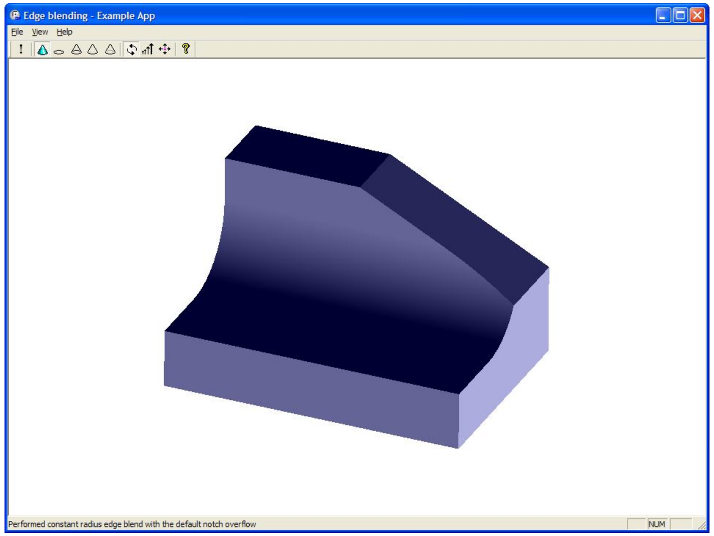


Received ..₩Example Parts₩blend3.x_t

Blend – Edge Blending

Step2 -Create blend with default notch overflow

```
PK_EDGE_set_blend_constant (
 --- received arguments ---
int n_edges, --- no. of edges to blend
const PK_EDGE_t edges[], --- edges to have blends set
double radius, --- blend radius
const PK_EDGE_set_blend_constant_o_t *options,
 --- returned arguments ---
int *const n_blend_edges, --- no. of edges with blends set
PK_EDGE_t **const blend_edges --- edges with blends set
)
```



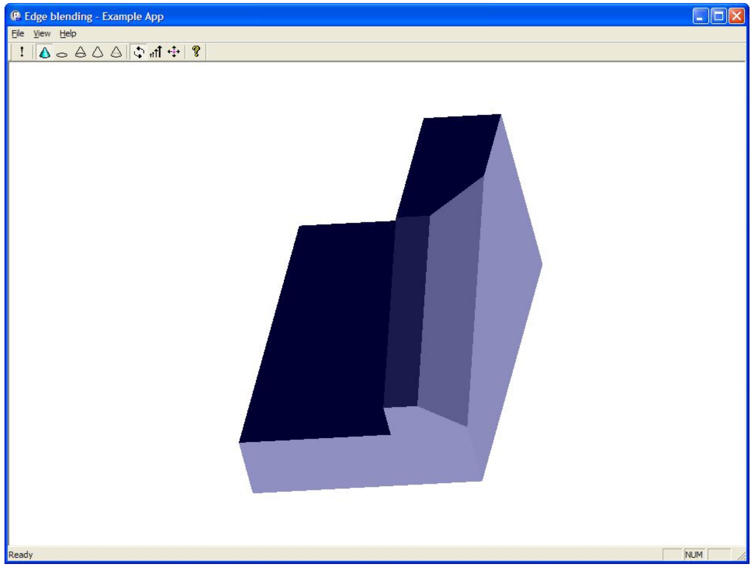
Performed constant radius edge blend with the default notch overflow

Step3 - Rollback

Blend – Edge Blending

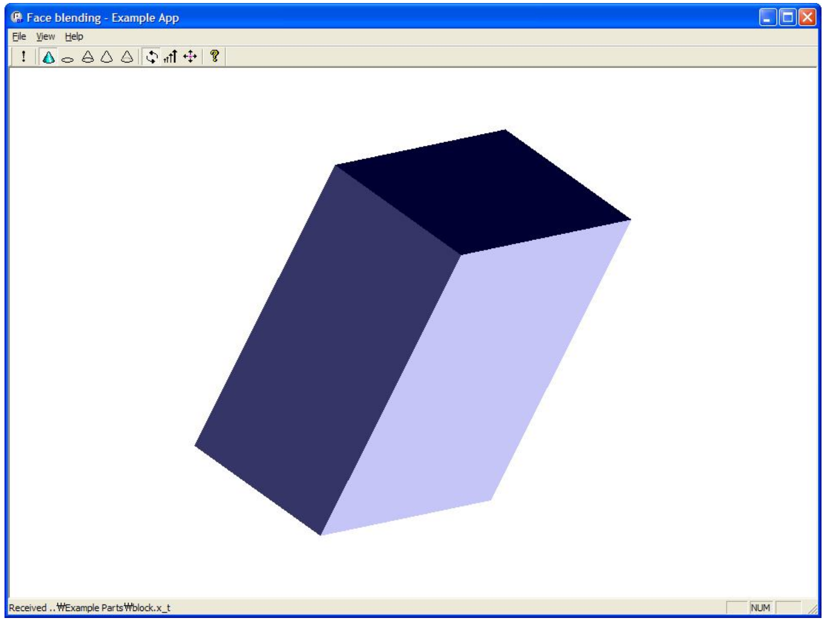
- Step 4 - Create blend with cliff overflow

```
PK_EDGE_set_blend_chamfer (
 --- received arguments ---
int n_edges, --- no. of edges to blend
const PK_EDGE_t edges[], --- edges to have blends set
double range_1, --- range on first face
double range_2, --- range on other face
const PK_FACE_t faces[], --- faces of first range  (optional)
const PK_EDGE_set_blend_chamfer_o_t *options, ---
options structure
 --- returned arguments ---
int *const n_blend_edges, --- no. of edges with blends set
 PK_EDGE_t **const blend_edges --- edges with blends set
)
```

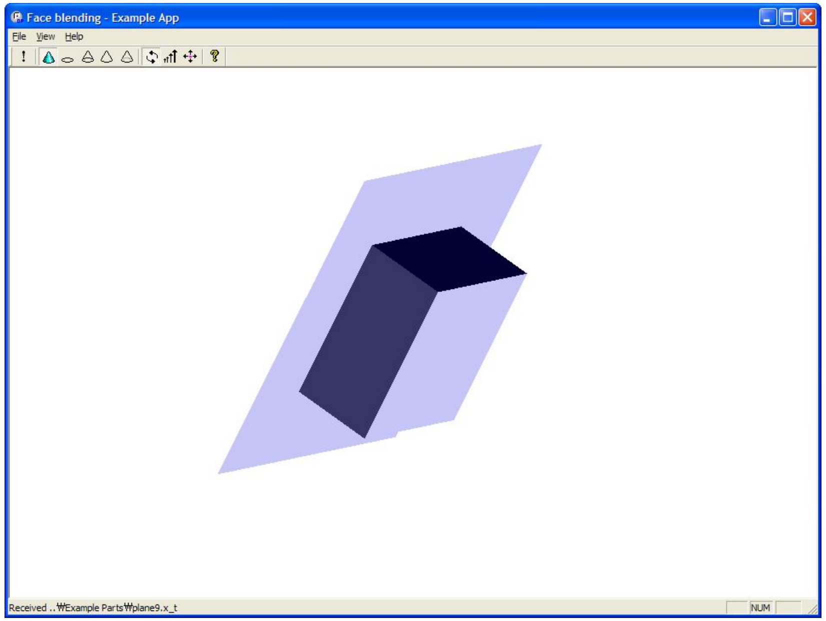


Blend – Two Face Blending

- Step 1, 2 - Load a block and a plane



Received ..₩Example Parts₩block.x_t

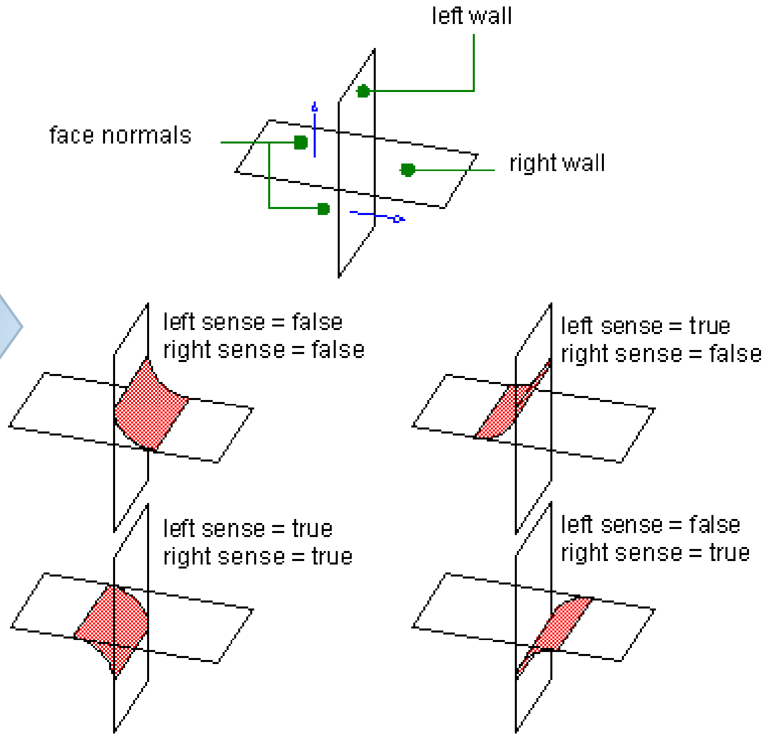


Received ..₩Example Parts₩plane9.x_t

Blend – Two Face Blending

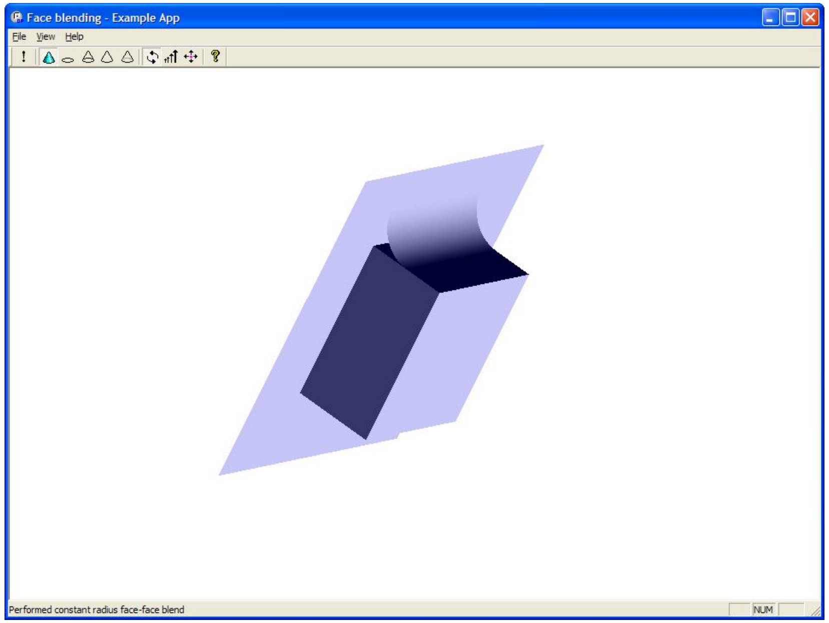
- Step 3 - Create a constant radius rolling ball blend

```
PK_FACE_make_blend (
 --- received arguments ---
int n_left_wall_faces, --- number of faces in left wall
const PK_FACE_t left_wall_faces[], --- faces in left wall
int n_right_wall_faces, --- number of faces in right wall
const PK_FACE_t right_wall_faces[], --- faces in right wall
PK_LOGICAL_t left_sense, --- blend direction from left wall
PK_LOGICAL_t right_sense, --- blend direction --- from right wall
const PK_FACE_make_blend_o_t *options, --- options structure
 --- returned arguments ---
int *const n_sheet_bodies, --- number of sheet bodies created
PK_BODY_t **const sheet_bodies, --- sheet bodies
int *const n_blend_faces, --- number of blend faces created
PK_FACE_t **const blend_faces, --- blend faces
PK_TOPOL_array_t **const unders, --- underlying topology of each face
PK_blend_rib_r_t *const ribs, --- ribs returned (if any)
PK_fxf_error_t *const fault --- fault found (if any)
)
```

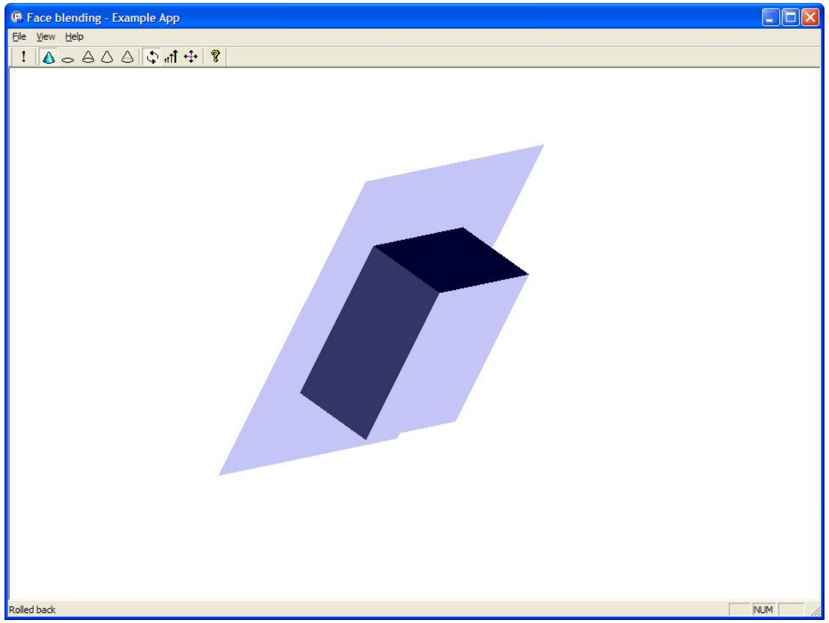


Blend – Two Face Blending

- Step 4 - Rollback



Performed constant radius face-face blend

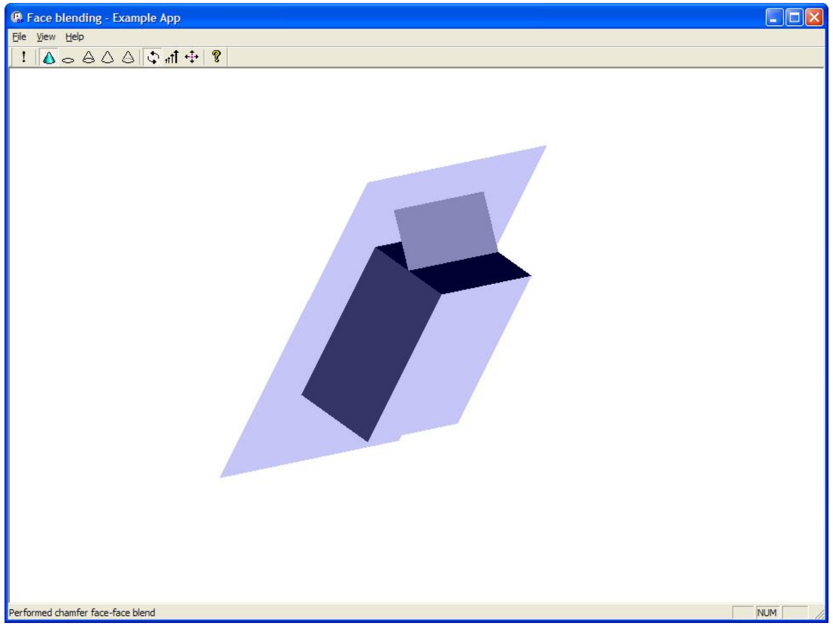


Rolled back

Blend – Two Face Blending

Step 5 - Create a chamfer blend

```
PK_FACE_make_blend_o_m( options );
options.shape.parameter = line;
options.shape.radius = 2.5;
options.shape.xs_shape = PK_blend_xs_shape_chamfer_c;
options.walls = PK_blend_walls_trim_no_c;
```



Performed chamfer face-face blend

The End