

Eagle NURBS Guide

*"A resource guide
for development of Eagle NURBS functions
in Eagle V14"*

Contents

1 - Introduction	5
2 - How does Eagle integrate NURBS	7
3 - How does Eagle-NURBS differ?	8
4 - Start using Eagle-NURBS	9
5 - Working with curves and surfaces from a vessel	13
5.1 Compatibility Information about the Generated Model File	17
6 - Edit curve by changing points through which it passes	18
7 - Edit curve by changing its control points	24
8 - Edit tangent at curve ends	31
9 - Edit Ellipse	38
10 - Surface creation using Extrusion	42
11 - Solid and Shell creation using Extrusion	44
12 - Surface creation using Revolution	46
13 - Body and Shell creation using Revolution	48
14 - Surface creation using Sweep	50
15 - Shell and Body creation using Sweep	54
16 - Nurbs Configuration Settings - Changes to the INI file	56
17 - LibEagle Nurbs Extension LibOCC	58
17.1 OCC_init	58
17.2 OCC_libvers	58
17.3 OCC_open_model	58
17.4 OCC_importBREP	59
17.5 OCC_importIGES	59

17.6 OCC_addShape.....	60
17.7 OCC_getShape.....	60
17.8 OCC_addNurbsCurve.....	61
17.9 OCC_addBezierCurve.....	61
17.10 OCC_initProfile.....	61
17.11 OCC_addItemToProfile.....	62
17.12 PCC_terminateProfile.....	62
17.13 OCC_convert.....	63
17.14 OCC_extrusion.....	63
17.15 OCC_revolution.....	63
17.16 The OCC_sweep.....	64
17.17 OCC_pointInShape.....	64
17.18 OCC_CalcSurfaceNormal.....	65
18 - OCC Import and Export.....	66
18.1 Layers.....	68
18.2 Colors.....	69

1 - Introduction

The implementation of Eagle-NURBS is one of the major new features of version 14. The project is an evolving one, which has been defined in Phases detailed in supporting documentation "NURBS - Software Implementation Document" provided with each release. This document should be the first and last call for the latest information and development status.

The current status of implementation is Level 2.2. It includes loading of NURBS geometry into the Eagle workspace, visualization and inquiry of such geometries by means of all Eagle standard methods, creation of Curve entities, creation of Surface entities, creation of Shell and Body entities, intersections, plotting and storing of NURBS geometry in an Eagle model file.

In terms of individual features in Eagle, the above functionalities can be listed as:

1. Import
2. Display
3. Select
4. Edit
5. Set
6. Profile
7. System and Item Information
8. Calculations
9. Intersections
10. Curve creation
11. Interpolation
12. Projection
13. Convert
14. Extrusion
15. Revolution
16. Sweep
17. Print and Plot
18. Save
19. LibOCC

The standard functionalities of the Eagle Solids Modeling module co-exist with the new Eagle-NURBS capabilities, and so all the functionalities that have been seen in the "Solid Modeling Merlin" part of the help file can be used in conjunction with the present ones.

There are nevertheless differences between the two modules. The Eagle Solid Modeling module, based on the Merlin kernel, is a B-REP faceted Modeler. Merlin SM technology has proved powerful and precise enough for many applications however faceted representation has obvious limitations, particularly in terms of precision, storage usage and speed, especially when the shape required is rather complex. The market for NURBS curves and surfaces, together with

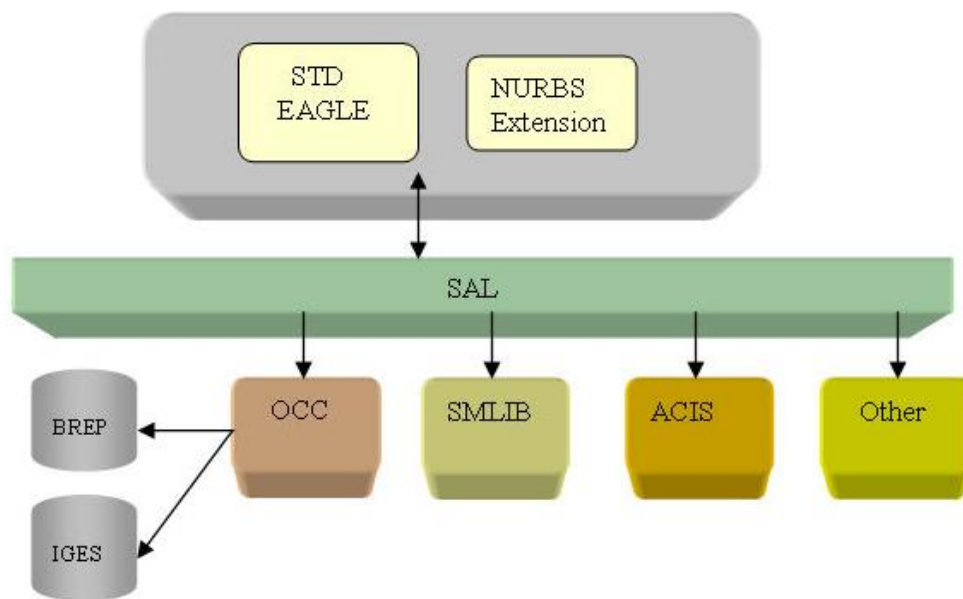
the advantage of working with standard and open model formats, like the IGES format, steered the integration of Eagle with OpenCascade one of the most interesting engines available for NURBS geometry. The resulting implementation is commonly known as Eagle-NURBS or the Exact Geometry Module.

In this section, we will give an overview of the key features related to working with the Eagle-NURBS.

2 - How does Eagle integrate NURBS

The NURBS geometric engine is completely embedded in the Eagle environment. NURBS entities like curves, profiles, surfaces, shell and bodies, co-exist with Eagle lines, pipes, faces, Ufos and all the other standard Eagle entities. The Eagle kernel determines which geometry it has to deal with depending on the command issued and the type of selected item, and on the basis of this switches to the appropriate geometric engine in a completely transparent manner to the application program. NURBS geometries stored in IGES files or in OCC-BREP files can be imported selectively and are embedded in the Eagle model file. The capability to handling IGES file is a very powerful feature, since IGES files are used natively by thousands of applications and millions of users opening the door for transparent integration of application data.

The implementation uses a Software Abstraction Layer (SAL). This has the purpose of providing an intermediate software layer to facilitate future implementation of NURBS libraries other than OCC.



3 - How does Eagle-NURBS differ?

Most of the standard Eagle commands maintain the same syntax and functionality in the Eagle-NURBS implementation. The Set and Select commands for example, while others such as the Curve, Interpolate and Projection commands are completely new.

The complete list of commands that either differ or are completely unique in the Eagle-NURBS implementation from the equivalent standard Eagle version is as follows:

- Unique Commands: Curve, Interpolate, Intersection, Pointinshape, Profile and Projection.
- Modified Commands: Area, Exclude, Identify, Import, Include, Mass, Meet, Nearest, Select, Set, Split and Volume.

The sections relating to Sys and Inf Function Information, as well as Selection Variables, include extra information about NURBS entities.

The Eagle-NURBS module includes additional primers to existing commands to obtain more control over specific NURBS entities in the Eagle workspace for the selection process. New primers allow the selection of exact geometry items: Curve, Profile, Surface, Shell or Body, but also allow extracting Curves, Profiles and Surfaces from such entities.

Viewing options give more control over the display processes of the object. It is possible to draw the Isoparametric control lines drawn in the U and V space of the object using the LINV and LINU parameters in conjunction with the Set CPAR option. This functionality is especially useful when surfaces are not planar.

In the Eagle-NURBS implementation the approximation of curves is only significant for the viewing process and does not have drawback on the calculation processes.

The full range of command changes and associated syntax are located in the Eagle Help file.

The Eagle-NURBS module requires that an "ex" feature line be present in the license file.

4 - Start using Eagle-NURBS

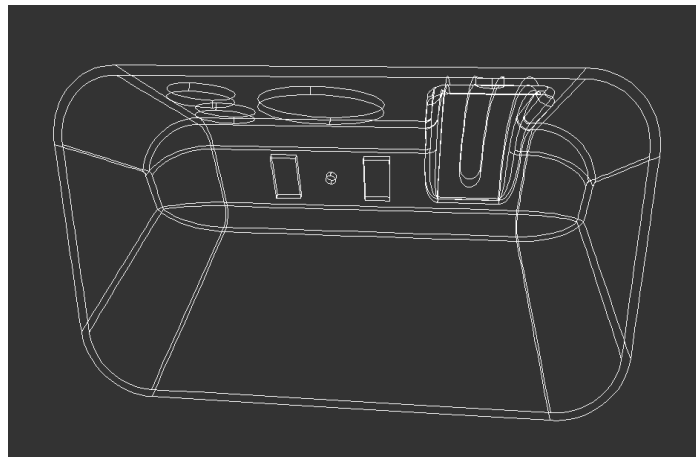
The target of this exercise is to provide a preliminary overview of the new functionalities. It starts by importing an IGES file into the Eagle workspace from an external program, and then selectively query the geometry, next see the advantages of using isoparametric lines view for better understanding of geometrical behavior, and finally show the usage of Eagle viewing in both wireframe and OpenGL modes.

The COVER.CMD command file contains the code that we use in this example.

Sample Code :

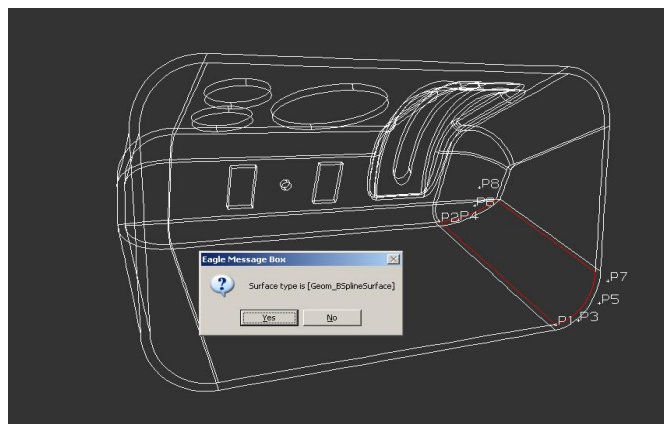
```
# COVER.CMD
# Declare the required set of variables
points p1\100
numerics v1\10
string surfType

# Define a proper background color
colour b,r=20,g=20,b=20
dynainput mode=1,w1=100
# Load the model using SUPPRESS to speed up the process
kill all
move p0
suppress
import cover.igs
unsuppress
select all
mod col=1
fit
polling;' Wireframe view of the model ';
```



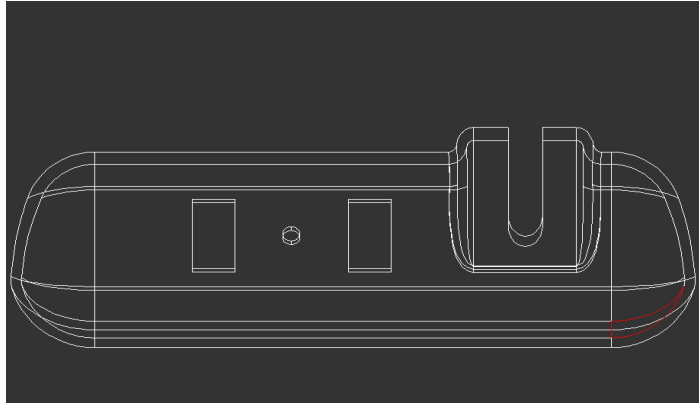
```
# Use selection to show geometrical information
```

```
polling;'Select a surface from the object ';
if vb=2 : goto termIt
nib 2
v1=inf(3)
p1=inf(1)
where p1\v1*v2
surfType=inf(8)
notify t=que;'Surface type is [^surfType]';
```



```
# Example of section views
```

```
box
planimetric
polling;'Indicate the first section plane';
if vb=2 : goto termIt
uncursor
p10=j
polling;'Indicate the second section plane';
if vb=2 : goto termIt
uncursor
p11=j
north p10,p11
```



Use Isoparametric lines to get a better idea of surface behavior

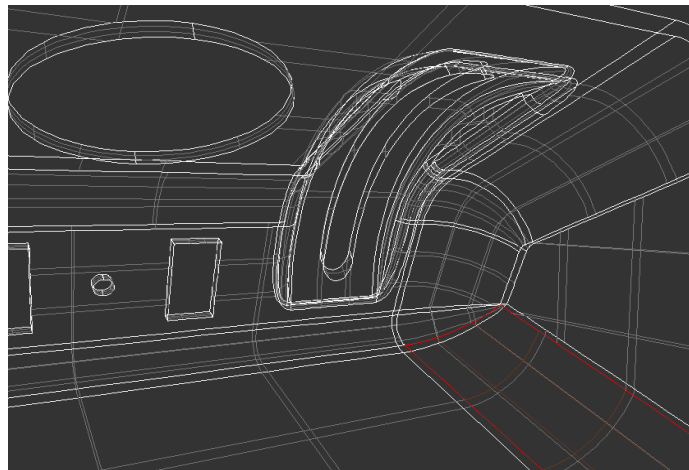
```
set cpar=on
```

```
set linu=2
```

```
set linv=2
```

```
show
```

```
polling;'Wireframe view with Isoparametric lines ';;
```



```
set cpar=off
```

Render the object in an OpenGL view

A proper usage of the next parameters is explained in details in the

INI file section part of this book

```
environment NURBS_FACETED=yes
```

```
set dtol=on
```

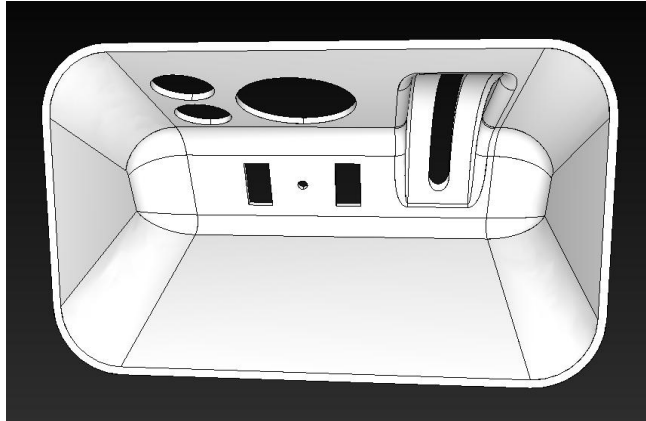
```
set ctol=0.1
```

```
opengl on
```

```
set glmode=edges
```

```
show
```

```
polling;'OpenGL view ';;
```



```
# Restore the standard wireframe view
```

```
opengl off
```

```
set glmode=std
```

```
set dtol=off
```

```
LABEL termIt
```

5 - Working with curves and surfaces from a vessel

The target of this exercise is to create a curve interpolating a series of points interactively provided by the user on a suitable view of a ship. After that, on the basis of this input, the program computes the corresponding series of curves that can be obtained by projecting the created one against the surfaces representing the hull of the vessel.

The VESSEL.CMD command file contains the code that we use in this example.

Sample Code :

```
# VESSEL.CMD
# Declare the required set of variables
points p1\100, pt1, pt2
numerics nPts, idCurve, nItemsPre, nItemsPost
string newCur, surfPre

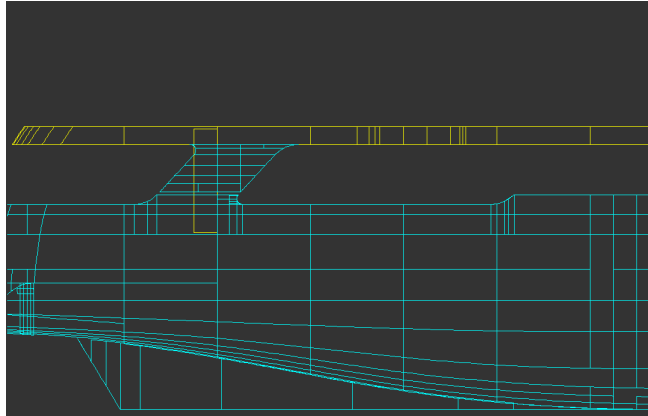
# Define a proper background color
polling b,r=20,g=20,b=20

# Load the model using SUPPRESS to speed up the display process
kill all
move p0
suppress
get $hull,i=0
unsuppress

# Ignore part of the hull to simplify calculations
part m=3
fragment 3
ignore 3

# Defines the box on a part of the model for working
box
pt1=b1&e2000
p3 =b2
pt2=midp(b1,b2)
pt2=pt2&u(p3)&n(p3) &w2000&d1000
box pt1,pt2
north

# The specified viewing brings the model in the following view
```



```

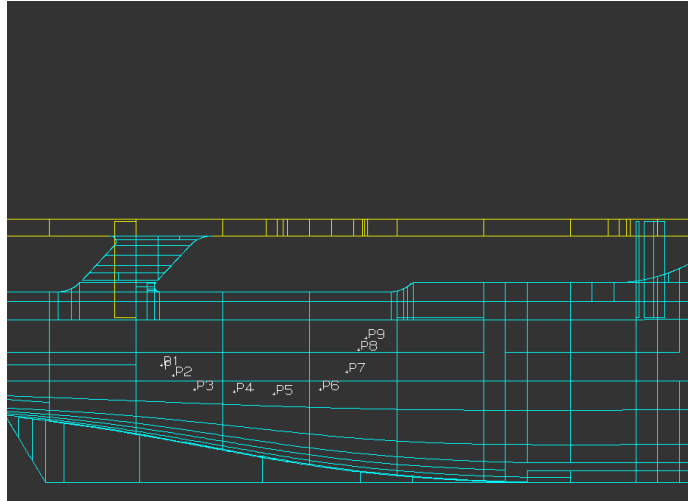
# Disable tolerance to allow fully interactive points specification
tolerance 0
approximation 32

# Use DYNAINPUT to provide a proper hint to the user
suppress m
dynainput mode=1,w1=100
switch on
nPts = 0

# Request the input points to be interpolated
LABEL anotherPoint
polling; 'Indicate a series of points ';;
if vb<>2 then
{
  # Set the point variables to be used for the interpolation
  uncursor
  nPts = nPts+1
  p[nPts]=j
  where p1\nPts
  goto anotherPoint
}

# Input points have been specified. Now show where they are placed
where p1\nPts
switch off

```

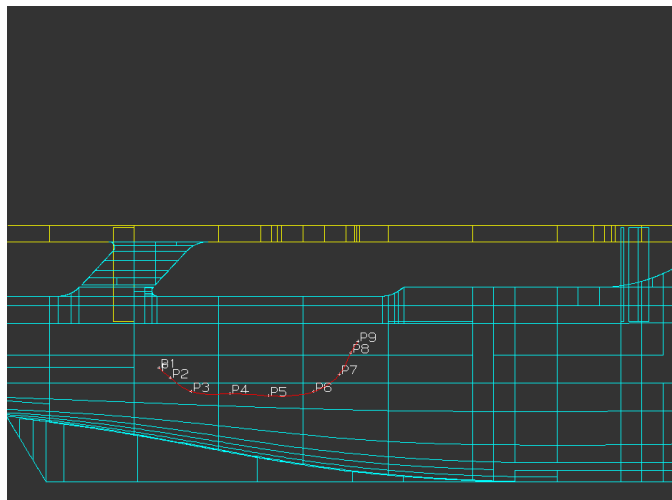


Create the curve using the interpolate command

pen 2

interpolate point=p1\nPts, type=1

pen 1



Finally project the curve against the hull.

The new curve is projected # against all relevant surfaces.

To do that, the surfaces are placed under the Partition bar before

invoking the projection command.

last

idCurve = inf(28)

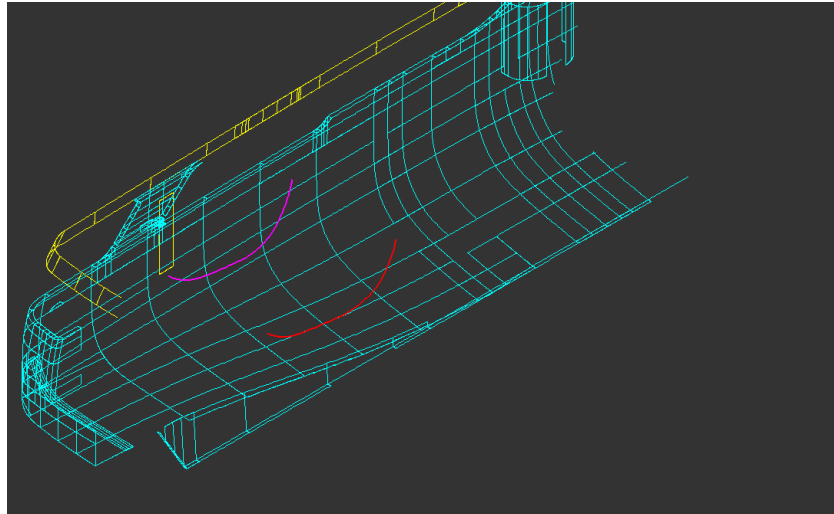
partition f=1

count nItemsPre,p

pen 7

Define a proper isometric view to being able to see new curves during

```
# creation
isometric
project id=idCurve, dir=(0,1,0)
pen 1
```



```
# Use a notify dialog to describe the obtained result
count nItemsPost,p
newCur =<nItemsPost-nItemsPre>
surfPre=<nItemsPre>
notify t=que; 'Pojection created N=[^newCur] curves '
        ' analyzing N=[^surfPre] surfaces';;
last nItemsPost-nItemsPre
highlight

# Restore original item visibility
see 3
unsuppress m

# Save the model to be used for next exercise
move p0
save exercise2,

# Save definition points to be used for next exercise
variable exercise2
#
```

5.1 Compatibility Information about the Generated Model File

Now, having a look at the file system and you will see that the SAVE command created only a single file. In fact this command has been enhanced to enable creation of a new model file that includes NURBS entities definition. This has been implemented defining a file format that consists of the OCC-BREP file, which represents all the NURBS entities, appended at the end of the standard model file which in turn represents all the entities in the workspace. In particular this also means that the definition of Curve, Profile, Surface, Shell and Body entities is split in two parts: their instance in the workspace, where each entity has its instance information (position and rotation matrix) and its attributes (color, pen, hatch and alphanumeric attributes), plus their geometrical definition, which is stored in the OCC-BREP section of the new Eagle model file, ready to be interpreted by OCC when the model is subsequently loaded.

The new Eagle version is capable of loading all previous Eagle model formats.

In the case of V14 NURBS model format, if the model file does not contain any new NURBS entities, then it can be read by previous versions of Eagle V.14 and V.12. However, if the model contains new NURBS entities then only V12 versions starting from V.12.6.0.B02 and later can direct read and load the model into the workspace but any new entities will be skipped. Any installations with older Eagle versions wishing to interact with models containing Nurbs will need and intermediate translation process.

6 - Edit curve by changing points through which it passes

The target of the next exercise is to edit the points through which the curve passes that was created in the previous exercise. This is carried out interactively by identifying new definition points. Note that there is a difference between this method and the one described in next exercise. The method in this exercise uses the Interpolate command to edit points through which the curve passes, while the method shown in the later exercise will edit curve control points iteratively using the Curve command. At the conclusion, on the basis of the new curve definition, this example invokes a further projection of the new curve against the surfaces representing the hull of the vessel to obtain a new series of curves.

The EDITPOINTS.CMD command file and its utility CHANGEPOINT.CMD contain the code that we use in this example.

Sample Code :

```
# EDITPOINTS.CMD
# Declare the required set of variables
points p1\100,pt1,pt2
numeric nPts, idCurve, nItemsPre, nItemsPost, result
string newCur, surfPre

# Define a proper background color
colour b,r=20,g=20,b=20

# Load the model created in previous exercise
kill all
move p0
suppress
get $exercise2,i=0
unsuppress

# Ignore part of the hull to simplify the calculations
part m=3
fragment 3
ignore 3

# Defines the box on a part of the model for working
box
pt1=b1&e2000
p50 =b2
pt2=midp(b1,b2)
pt2=pt2&u(p50)&n(p50) &w2000&d1000
box pt1,pt2
```

isometric

Disable tolerance to allow fully interactive points

tolerance 0

approximation 32

Use DYNAINPUT to provide a proper hint to the user

suppress m

dynainput mode=1,w1=100

Go through the edit process

This is done via a macro that is described at the end of the current one

do changePoint(@result)

if result = -1 : goto termIt

Project the new curve against all relevant surfaces of the hull.

To do that, such surfaces are placed under the Partition

last

idCurve = inf(28)

Display the old projected curves with a different colour

partition m=7

nib 6

Select the relevant surfaces

part f=1

count nItemsPre,p

Makes the new projection

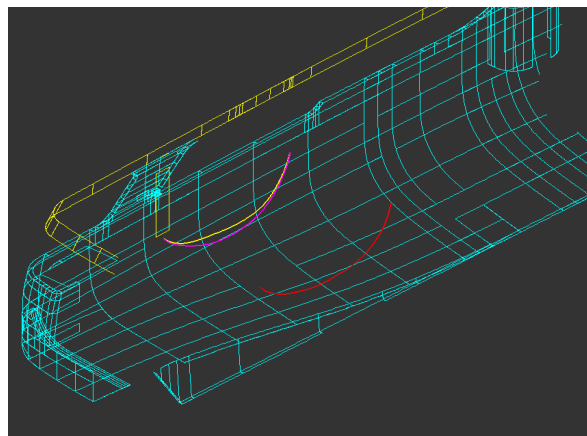
pen 7

Define a proper isometric view to being able to see new curves during creation

isometric

project id=idCurve, dir=(0,1,0)

pen 1



```

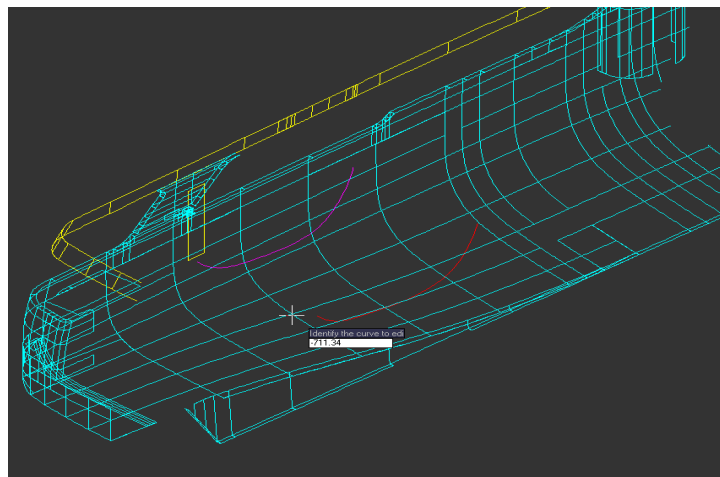
# Use a notify dialog to describe the obtained result
count nItemsPost,p
newCur=<nItemsPost-nItemsPre>
surfPre=<nItemsPre>
notify t=que;'Pojection created N=[^newCur] curves analyzing '
        'N=[^surfPre] surfaces';;
last nItemsPost-nItemsPre
high
unsuppres m
# Restore initial visibility
see 3

# CHANGEPOINT.CMD
#
# Utility to edit points on a curve
#
arguments numeric @result
{
  numerics narr[20], knots[100], mults[100], weights[100]
  numerics index, indRep, phase, ipanel, done
  points p1\100
  string value

  phase=1
  result=-1

  # Select the curve to edit
  polling; 'Identify the curve to edit ';;
  if vb=2 : goto termIt

```



```
uncursor
select xc
ifno notify t=que;'No curve has been selected';; goto termIt

# Retrieve definition points and number of points previously
# stored in variable file
retrieve exercise2

# Place the model in a proper view for working
north
phase=2

# Show curve parameters in a dedicated panel
narr[1] = inf(38)
# Setup info retrieved from original curve definition
narr[4] = nPts
where p1\narr[4]

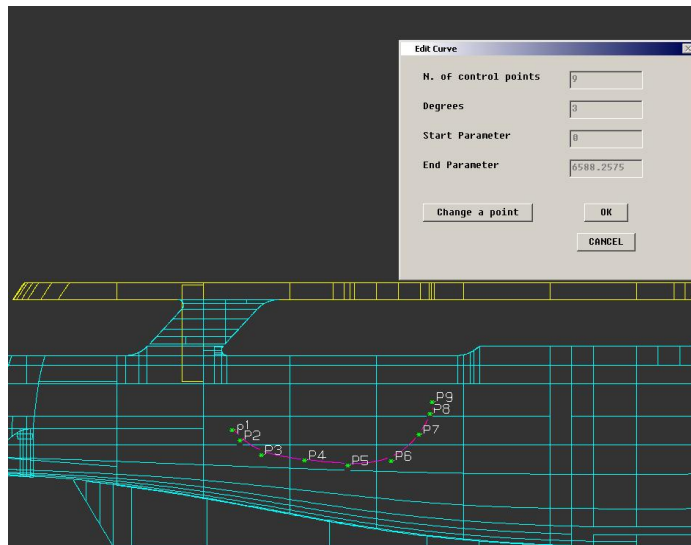
ipanel = 1
panel p=ipanel,off
panel p=ipanel,on,u=1,r=300,c=400,t='Edit Curve',
      pin=0,j=.7,.9,mod=1
option p=ipanel,f=editCurve.tab
# Fill the main curve info for this process
insert p=ipanel,b=2,t=narr[4]
insert p=ipanel,b=4,t=narr[3]
insert p=ipanel,b=6,t=narr[6]
insert p=ipanel,b=8,t=narr[7]
# Freeze fields so that they can only be viewed
freeze p=ipanel,b=2
freeze p=ipanel,b=4
freeze p=ipanel,b=6
freeze p=ipanel,b=8

done=0
while done=0
{
  polling; 'Make a choice ';;
}
LABEL morePoints
# Place the current curve on a separate layer for later cleaning it
```

```

fragment 10
# Creates gnode on points to facilitate selection
for index=1,narr[4]
  partition
  gnode v=p[index],m=3,f=9
  value = <index>
  modify a1=^value
next index

```



```

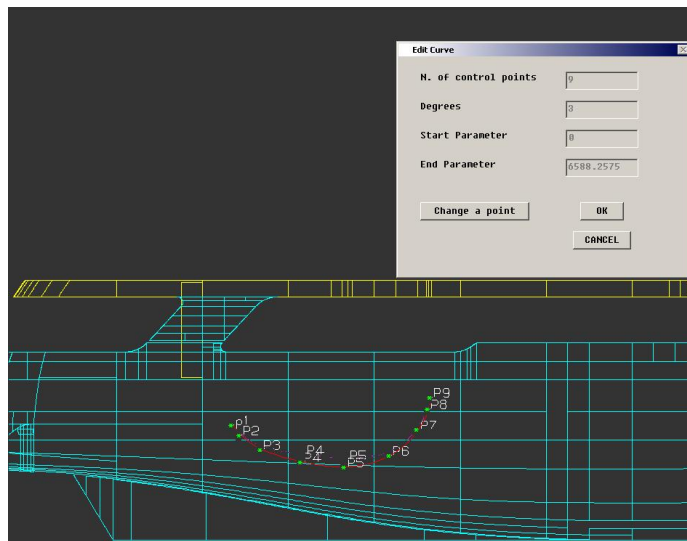
# Request the new point position
freeze p=ipanel, b=22
value = att(1)
indRep = ^value
polling; 'Indicate the new point position ';;
if vb=2 : goto termIt
uncursor
p[indRep]=j
where p[indRep]
# Remove the old curve and the intermediate Gnodes
erase f=9
erase f=10

# Create the new curve
pen 2
interpolate point=p1\narr[4],type=1
pen 1

unfreeze p=ipanel,b=22

```

result=0
phase=3



goto morePoints

LABEL termIt

erase f=9

if phase=2 then

{

partition f=10

fragment 0

}

Close the panel, when required

if phase>1 : panel p=ipanel,off

}

#

EDITCURVE.TAB

#

1, 30, 20,200,25,0,0,5,'N. of points': ;

2, 230, 20,100,25,0,0,6,"; ;

3, 30, 60,150,25,0,0,5,'Degrees': ;

4, 230, 60,100,25,0,0,6,"; ;

5, 30,100,150,25,0,0,5,'Start Parameter': ;

6, 230,100,100,25,0,0,6,"; ;

7, 30,140,150,25,0,0,5,'End Parameter': ;

8, 230,140,100,25,0,0,6,"; ;

20, 30,200,150,25,0,0,1,'Change a point': done=1; goto morePoints;

22,250,200, 60,25,0,0,1,'OK': done=1; goto termIt;

24,240,240, 80,25,0,0,1,'CANCEL': done=1; result=-1; goto termIt;

7 - Edit curve by changing its control points

The target of this exercise is to edit the curve control points. This is done by interactively identifying new control points. Finally, on the basis of the identification, we invoke a further projection of the new curve against the surfaces representing the hull of the vessel to obtain a new series of curves.

The EDITCTRLPTS.CMD command file and its utility CHANGECTRLPTS.CMD contain the code that we use in this example.

Sample Code :

```
# EDITCTRLPTS.CMD
# Declare the required set of variables
points p1\100,pt1,pt2
numeric nPts, idCurve, nItemsPre, nItemsPost, result
string newCur, surfPre

# Define a proper background color
colour b,r=20,g=20,b=20

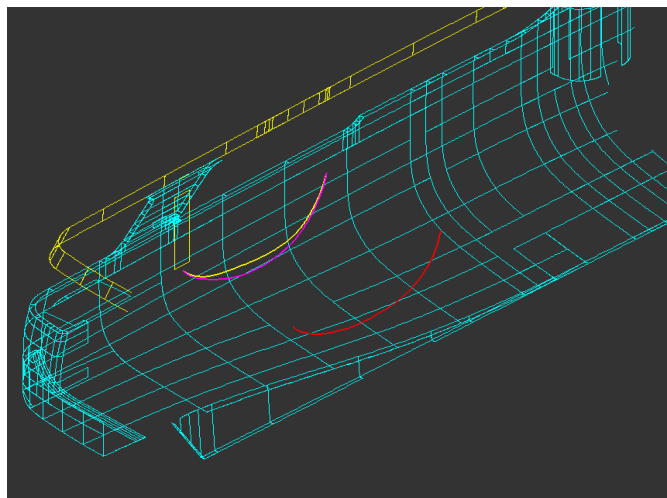
# Load the model created in previous exercise
kill all
move p0
suppress
get $exercise2,i=0
unsuppress

# Ignore part of the hull to simplify the calculations
part m=3
fragment 3
ignore 3

# Defines the box on a part of the model for working
box
pt1=b1&e2000
p50 =b2
pt2=midp(b1,b2)
pt2=pt2&u(p50)&n(p50) &w2000&d1000
box pt1,pt2
isometric
```

```
# Disable tolerance to allow fully interactive points
tolerance 0
approximation 32
# Use DYNAINPUT to provide a proper hint to the user
suppress m
dynainput mode=1,w1=100
# Go through the edit process
# This is done via a macro that is described at the end of the current one
do changectrlpts(@result)
if result = -1 : goto termIt
# Project the new curve against all relevant surfaces of the hull.
# To do that, such surfaces are placed under the Partition
last
idCurve = inf(28)
# Display the old projected curves with a different color
partition m=7
nib 6
# Select the relevant surfaces
part f=1
count nItemsPre,p

# Makes the new projection
pen 7
# Define a proper isometric view to being able to see new curves during
creation
isometric
project id=idCurve, dir=(0,1,0)
pen 1
```

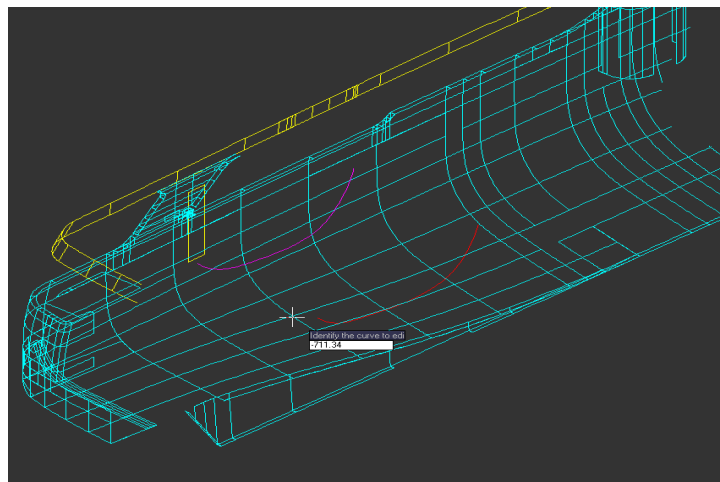


```

# Use a notify dialog to describe the obtained result
count nItemsPost,p
newCur=<nItemsPost-nItemsPre>
surfPre=<nItemsPre>
notify t=que;'Pojection created N=[^newCur] curves analyzing '
        'N=[^surfPre] surfaces';;
last nItemsPost-nItemsPre
high
unsuppres m
# Restore initial visibility
see 3

# CHANGECTRLPTS.CMD
#
# Utility to edit curve control points
arguments numeric @result
{
    numerics narr[20], knots[100], mults[100], weights[100]
    numerics index, indRep, phase, ipanel, done
    points p1\100
    string value
    phase=1
    result=-1
    # Select the curve to edit
    polling; 'Identify the curve to edit ';;
    if vb=2 : goto termIt

```



```
uncursor
select xc
ifno notify t=que;'No curve has been selected';; goto termIt

# Place the model in a proper view for working
north
phase=2

# Show curve parameters in a dedicated panel
narr[1] = inf(38)
knots[1] = inf(39)
mults[1] = inf(40)
weights[1] = inf(41)
p1= inf(1)
points[1] = inf(1)
where p1\narr[4]
#
ipanel = 1
panel p=ipanel,off
panel p=ipanel,on,u=1,r=300,c=400,t='Edit Control Points',
    pin=0,j=.7,.9,mod=1
option p=ipanel,f=editCurve.tab

# Fill the main curve info for this process
insert p=ipanel,b=2,t=narr[4]
insert p=ipanel,b=4,t=narr[3]
insert p=ipanel,b=6,t=narr[6]
insert p=ipanel,b=8,t=narr[7]
# Freeze fields so that they can only be viewed
freeze p=ipanel,b=2
freeze p=ipanel,b=4
freeze p=ipanel,b=6
freeze p=ipanel,b=8

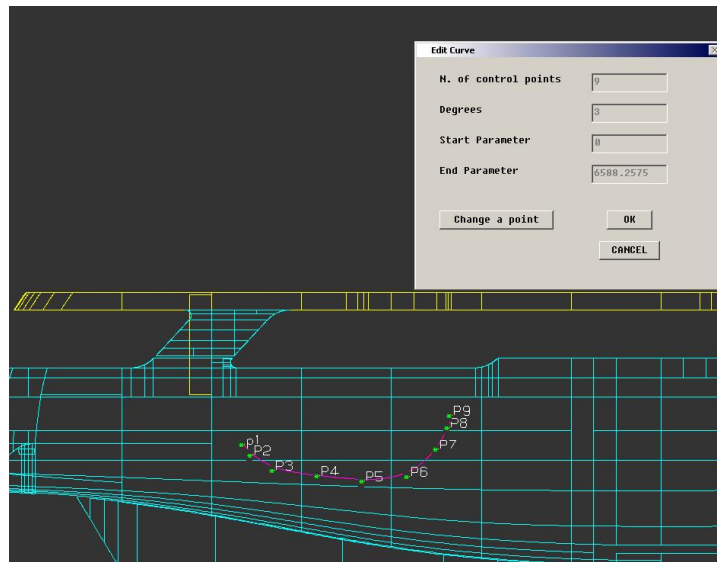
done=0
while done=0
{
    polling; 'Make a choice ';;
}
LABEL morePoints

# Place the current curve on a separate layer for later cleaning
fragment 10
# Creates gnode on points to facilitate selection
```

```

for index=1,narr[4]
  partition
  gnode v=p[index],m=3,f=9
  value = <index>
  modify a1=^value
next index

```



```

# Request the new point position
freeze p=ipanel,b=22
value = att(1)
indRep = ^value
polling; 'Indicate the new point position ';;
if vb=2 : goto termIt
uncursor
p[indRep]=j
points[indRep] = p[indRep]
where p[indRep]

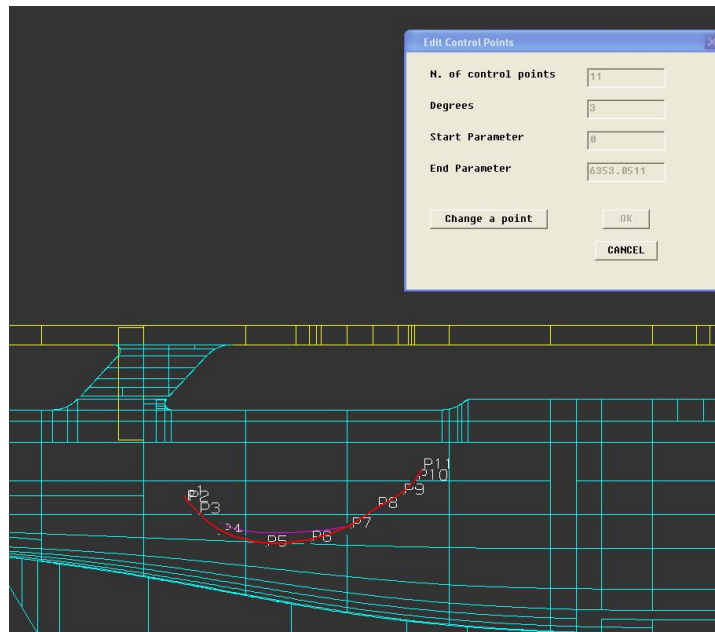
# Remove the old curve and the intermediate Gnodes
erase f=9
erase f=10

# Create the new curve using its original parameters
pen 2
Curve deg=narr[3], per=narr[1], ncpts=narr[4], cpts=points[1],
      nknot=narr[5], knot=knots[1], molts=mults[1], wei=weights[1]
pen 1

```

```

result=0
phase=3
  
```



```

unfreeze p=ipanel,b=22
goto morePoints
  
```

```
LABEL termIt
```

```
erase f=9
```

```
if phase=2 then
```

```
{
```

```
partition f=10
```

```
fragment 0
```

```
}
```

```
# Close the panel, when required
```

```
if phase>1 : panel p=ipanel,off
```

```
}
```

```
#
```

```
# EDITCURVE.TAB
```

```
#
```

```
1, 30, 20,200,25,0,0,5,'N. of control points': ;
```

```
2, 230, 20,100,25,0,0,6,": ;
```

```
3, 30, 60,150,25,0,0,5,'Degrees': ;
```

```
4, 230, 60,100,25,0,0,6,": ;
```

```
5, 30,100,150,25,0,0,5,'Start Parameter': ;  
6, 230,100,100,25,0,0,6,""; ;  
7, 30,140,150,25,0,0,5,'End Parameter': ;  
8, 230,140,100,25,0,0,6,""; ;  
20, 30,200,150,25,0,0,1,'Change a point': done=1; goto morePoints;  
22,250,200, 60,25,0,0,1,'OK': done=1; goto termIt;  
24,240,240, 80,25,0,0,1,'CANCEL': done=1; result=-1; goto termIt;
```

8 - Edit tangent at curve ends

The target of this exercise is to edit the curve that was created with exercise #2. This is done by interactively identifying new tangents at curve ends. Finally, on the basis of the previous actions, we invoke a further projection of the new curve against the surfaces, which represent the hull of the vessel, to obtain a new series of curves.

The EDITTANGENT.CMD command file and its utility CHANGETANGENT.CMD contain the code that we use in this example.

Sample Code :

```
# EDITTANGENT.CMD

# Declare the required set of variables
points p1\100,pt1,pt2
numeric nPts, idCurve, nItemsPre, nItemsPost, result
string newCur, surfPre

# Define a proper background color
colour b,r=20,g=20,b=20

# Load the model created in previous exercise
kill all
move p0
suppress
get $exercise2,i=0
unsuppress

# Ignore part of the hull to simplify the calculations
part m=3
fragment 3
ignore 3

# Defines the box on a part of the model for working
box
pt1=b1&e2000
p50 =b2
pt2=midp(b1,b2)
pt2=pt2&u(p50)&n(p50) &w2000&d1000
box pt1,pt2
isometric
```

```
# Disable tolerance to allow fully interactive points
tolerance 0
approximation 32

# Use DYNAINPUT to provide a proper hint to the user
suppress m
dynainput mode=1,w1=100

# Go through the edit process
# This is achieved via a macro that is described at the end of
# the current one
do changeTangent(@result)
if result = -1 : goto termIt

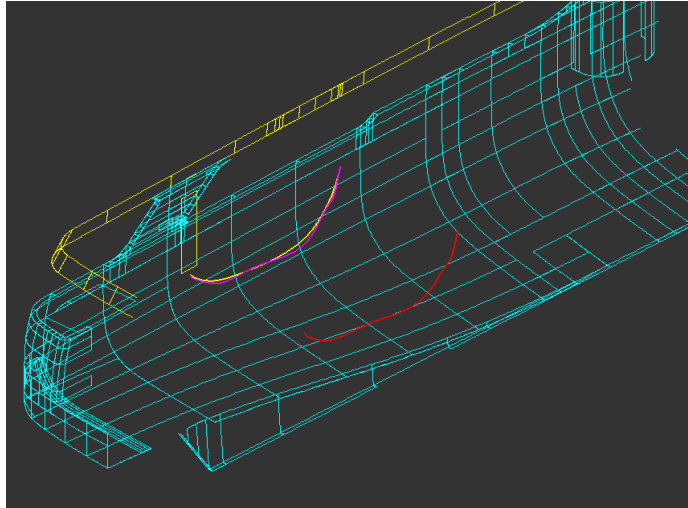
# Project the new curve against all relevant surfaces of the hull.
# To do that, such surfaces are placed under the Partition
last
idCurve = inf(28)

# Display the old projected curves with a different colour
partition m=7
nib 6

# Select the relevant surfaces
part f=1
count nItemsPre,p

# Makes the new projection
pen 7

# Define a proper isometric view to being able to see new curves during
# creation
isometric
project id=idCurve, dir=(0,1,0)
pen 1
```



```

# Use a notify dialog to describe the obtained result
count nItemsPost,p
newCur=<nItemsPost-nItemsPre>
surfPre=<nItemsPre>
notify t=que;'Pojection created N=[^newCur] curves analyzing'
        'N=[^surfPre] surfaces';;
last nItemsPost-nItemsPre
high
unsuppres m

# Restore initial visibility
see 3

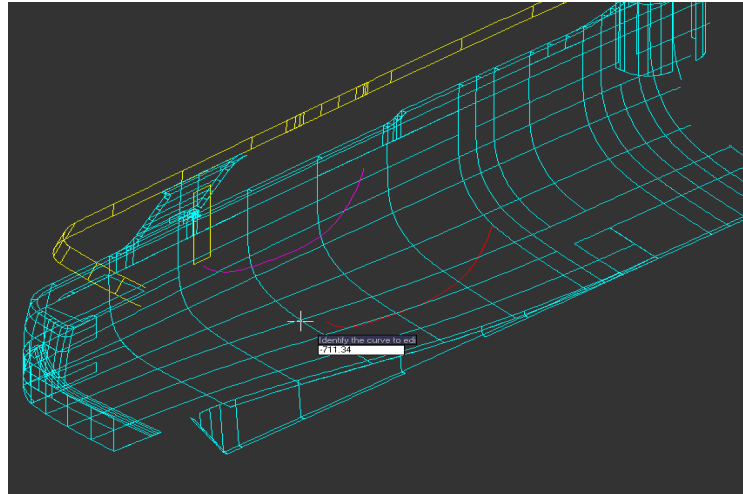
#
# CHANGETANGENT.CMD
#
# Utility to edit tangent at a curve ends
#

arguments numerics@result
{
  numerics narr[20], knots[100], mults[100], weights[100]
  numerics index, indRep, phase, ipanel, done
  points p1\100, dir1, dir2
  string value

  phase=1
  result=-1

```

```
# Select the curve to edit
polling; 'Identify the curve to edit ';;
if vb=2 : goto termIt
```



```
uncursor
select xc
ifno notify t=que;'No curve has been selected';; goto termIt

# Retrieve definition points previously stored in variable file
retrieve exercise2

# Place the model in a proper view for working
north
phase=2

# Show curve parameters in a dedicated panel
narr[1] = inf(38)
where p1\narr[4]

ipanel = 1
panel p=ipanel,off
panel p=ipanel,on,u=1,r=300,c=400,t='Edit curve',pin=0,j=.7,.9,mod=1
option p=ipanel,f=editTangent.tab

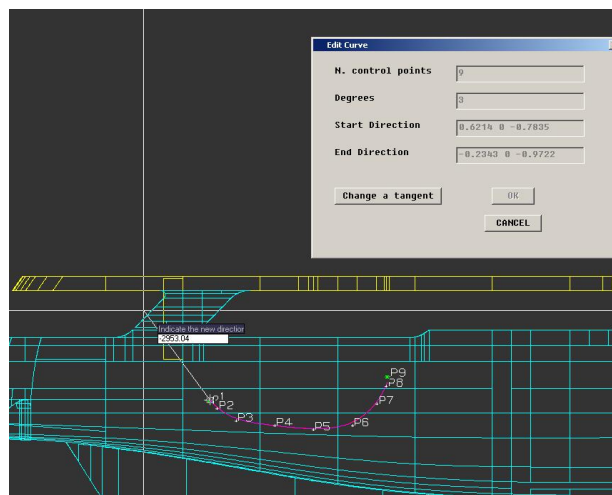
# Setup initial values for tangents
dir1 = dirc(p1,p2)
dir2 = dirc(p[narr[4]], p[narr[4]-1])
```

```

# Fill the relevant curve info for this process
insert p=ipanel,b=2,t=narr[4]
insert p=ipanel,b=4,t=narr[3]
insert p=ipanel,b=6,t=dir1
insert p=ipanel,b=8,t=dir2
# Freeze fields so that they can only be viewed
freeze p=ipanel,b=2
freeze p=ipanel,b=4
freeze p=ipanel,b=6
freeze p=ipanel,b=8

done=0
while done=0
{
  polling; 'Make a choice ';;
}

LABEL moreTangents
# Place the current curve on a separate layer for later cleaning it
fragment 10
# Creates gnode on both ends to facilitate selection
partition
gnode v=p[1],m=3,f=9
modify a1='1'
partition
gnode v=p[narr[4]],m=3,f=9
value = <narr[4]>
modify a1='^value'
polling; 'Indicate the extremity to be changed ';;
if vb=2 : goto termIt
  
```



```

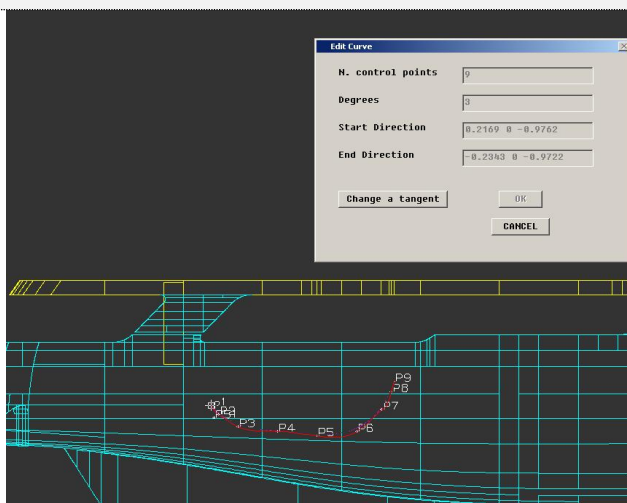
uncursor
select n
ifno notify t=que;'No point has been selected'; goto termIt

# Request the new point position
freeze p=ipanel,b=22
value = att(1)
p50=c
polling r=l; 'Indicate the new direction';
if vb=2 : goto termIt
uncursor
p51=j
where p51

# Update tangents
if value = 1 : dir1 = dirc( p50,p51)
else          dir2 = dirc( p50,p51)
# Remove the old curve and the intermediate gnodes
erase f=9
erase f=10
# Create the new curve
pen 2
interpolate point=p1\narr[4],type=1, start=dir1, end=dir2
pen 1

# Update info on the panel
insert p=ipanel,b=6,t=dir1
insert p=ipanel,b=8,t=dir2
unfreeze p=ipanel,b=22
result=0
phase=3

```



```
goto moreTangents

LABEL termIt
  erase f=9
  if phase=2 then
  {
    partition f=10
    fragment 0
  }
  # Close the panel, when required
  if phase>1 : panel p=ipanel,off
}

#
# EDITTANGENT.TAB
#
1, 30, 20,150,25,0,0,5,'N. control points': ;
2, 200, 20,180,25,0,0,6,": ;
3, 30, 60,150,25,0,0,5,'Degrees': ;
4, 200, 60,180,25,0,0,6,": ;
5, 30,100,150,25,0,0,5,'Start Direction': ;
6, 200,100,180,25,0,0,6,": ;
7, 30,140,150,25,0,0,5,'End Direction': ;
8, 200,140,180,25,0,0,6,": ;
20, 30,200,150,25,0,0,1,'Change a tangent': done=1; goto moreTangents;
22,250,200, 60,25,0,0,1,'OK': done=1; goto termIt;
24,240,240, 80,25,0,0,1,'CANCEL': done=1; result=-1; goto termIt;
```

9 – Edit Ellipse

The target of this exercise is to create a new curve representing an Ellipse and then change its geometrical definition. This is achieved by first retrieving curve information and then by interactively identifying the new subtended angles.

Sample Code :

```
# EDITELLIPSE.CMD
# Declare the required set of variables
points    p1\100,pt1,pt2
numerics  narr[20], ipanel, done, v1\10, x, y, alpha, minRad,
maxRad, ang1, ang2
string    value

# Define a proper background color
colour b,r=20,g=20,b=20

#
kill all
move p0
pen 1
line f=1; aw130, e260, aw130&s70, n140;;

move p0
pen 3
Ellipse center=(0,0,0), X=(1,0,0), Y=(0,1,0), maxrad=120, minrad=60,
angle1=45, angle2=-45, sense=1
pen 1
phase=2

# Show curve parameters in a dedicated panel
last
narr[1] = inf(38)
p1= inf( (1)
where P1,P3\4

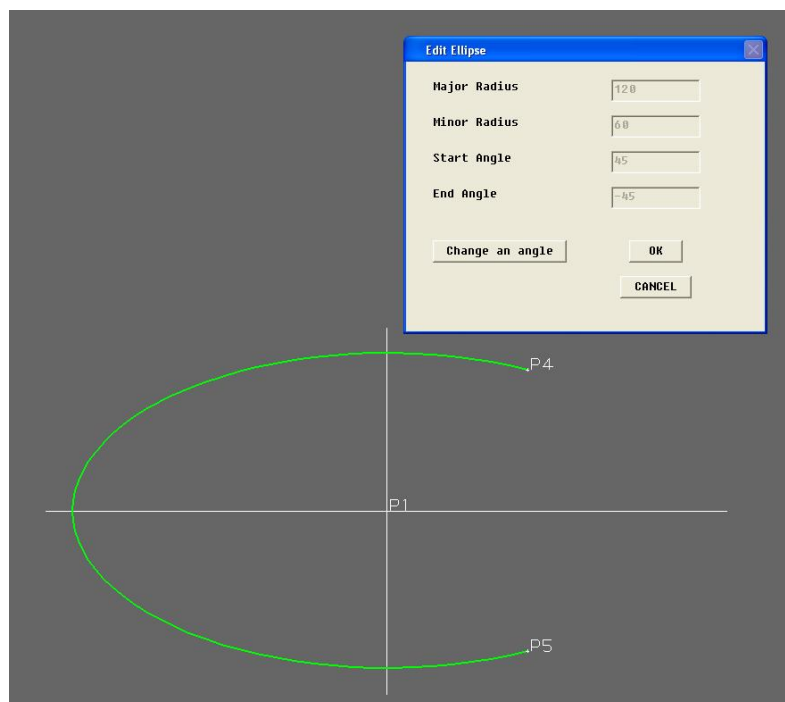
#
ipanel = 1
panel p=ipanel,off
panel p=ipanel,on,u=1,r=300,c=400,t='Edit Ellipse', mod=1
option p=ipanel,f=editEllipse.tab

# Fill the main curve info for this process
```

```

insert p=ipanel,b=2,t=narr[4]
insert p=ipanel,b=4,t=narr[5]
insert p=ipanel,b=6,t=narr[6]
insert p=ipanel,b=8,t=narr[7]

# Freeze them ...
freeze p=ipanel,b=2
freeze p=ipanel,b=4
freeze p=ipanel,b=6
freeze p=ipanel,b=8
  
```



```

#
done=0
while done=0
{
  polling; 'Make a choice ';;
}
LABEL again

# Place the current curve on a separate layer for later cleaning
fragment 10

# Creates gnodes on start and end angles to facilitate selection
partition
  
```

```

gnode v=p4,m=3,f=9
modify a1='1'
partition
gnode v=p5,m=3,f=9
modify a1='2'
see 1

#
polling ; 'Indicate the point to be changed ';;
if vb=2 : goto termlt
uncursor
select n
ifno notify T=QUE;'No point has been selected';; goto termlt

# Request the new point position
freeze p=ipanel,b=22
value = att(1)
polling ; 'Indicate the new point position ';;
if vb=2 : goto termlt

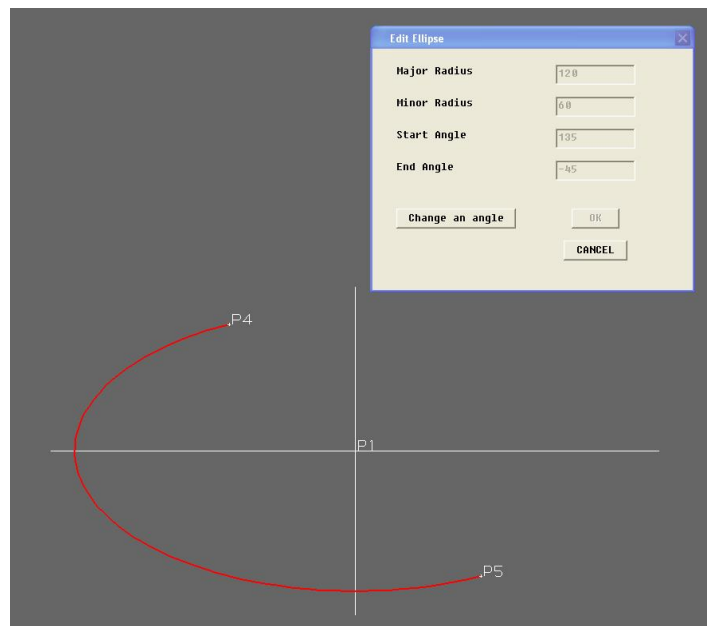
uncursor
p20=j
if value = 1 : ang1 = ang(p1,p20); ang2 = narr[7]
else      ang1 = narr[6]; ang2 = ang(p1,p20)

# Remove the old ellipse and the intermediate gnodes
erase f=10
erase f=9
Ellipse center=(0,0,0), X=(1,0,0), Y=(0,1,0), maxrad=narr[4],
      minrad=narr[5], angle1=ang1, angle2=ang2, sense=1

#
last
p1= inf(1)
narr[1]= inf(38)
nib 2
modify th=2

# Update info in panel
narr[6] = ang1
narr[7] = ang2
insert p=ipanel,b=6,t=narr[6]
insert p=ipanel,b=8,t=narr[7]

```



```

#
pen 1
unfreeze p=ipanel,b=22
goto again
LABEL termIt
erase f=9
#
panel p=ipanel,off
}

#
# EDITELLIPSE.TAB
#
1, 30, 20,200,25,0,0,5,'Major Radius': ;
2, 230, 20,100,25,0,0,6,": ;
3, 30, 60,150,25,0,0,5,'Minor Radius': ;
4, 230, 60,100,25,0,0,6,": ;
5, 30,100,150,25,0,0,5,'Start Angle': ;
6, 230,100,100,25,0,0,6,": ;
7, 30,140,150,25,0,0,5,'End Angle': ;
8, 230,140,100,25,0,0,6,": ;
20, 30,200,150,25,0,0,1,'Change an angle': done=1; goto again;
22,250,200, 60,25,0,0,1,'OK': done=1; goto termIt;
24,240,240, 80,25,0,0,1,'CANCEL': done=1; goto termIt;
  
```

10 – Surface creation using Extrusion

The target of this exercise is to create a surface by extruding an existing Curve between two points. The Disorder command is used to help clarify viewing and the iso-parametric lines are added to better understand surface behavior.

Sample Code :

```
# EXTRUDECURVE.CMD
# Declare the required set of variables
points    p1\100
numerics  CurveID

# Define a proper background color
colour b,r=20,g=20,b=20
dynainput mode=1,w1=100

# Load the model created in previous exercise
disorder on
disorder 1,2,3\256

#
kill all
move p0
get $extrusion

part m=3
modify f=2
CurveID= inf(28)
partition r
p1= inf(1)

# Creates the surface
pen 6
Extrusion entity=CurveID,P1\2
pen 1

last
modify f=1
modify col=6
```

```
# Use iso-parametric lines for better understanding of surface behavior
```

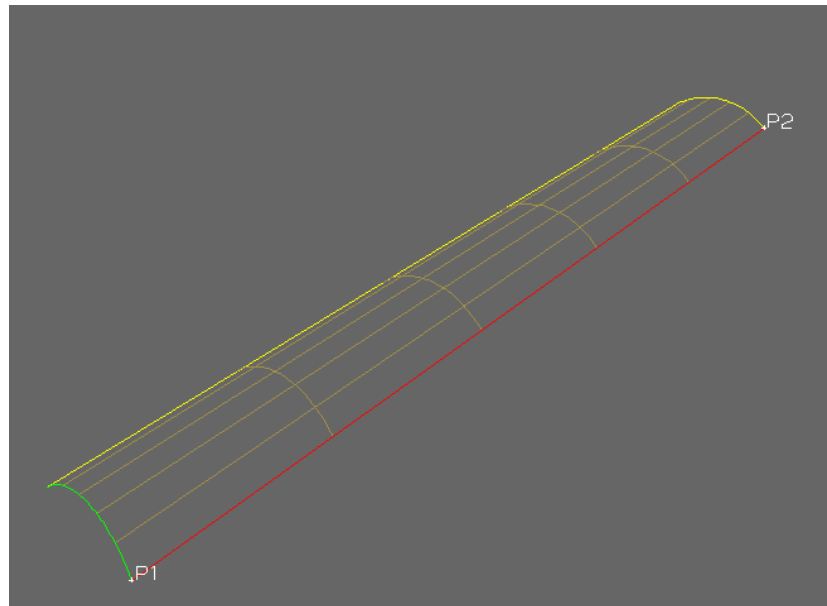
```
set cpar=on
```

```
set linu=4
```

```
set linv=4
```

```
fit
```

```
polling;'GREEN Curve has been extruded along RED spine ';;
```



11 – Solid and Shell creation using Extrusion

The target of this exercise is to create a Solid by extruding an existing Profile between two points. After this, using the same methodology, a Shell is created that differs from the Solid in that it consists of all the individual extruded Surfaces without the bottom and the top one, thus bringing an empty volume for the result.

Sample Code:

```
# EXTRUDEPROFILE.CMD
# Declare the required set of variables
points    p1\100
numerics  ProfileID

# Define a proper background color
colour b,r=40,g=40,b=40
dynainput mode=1,w1=100

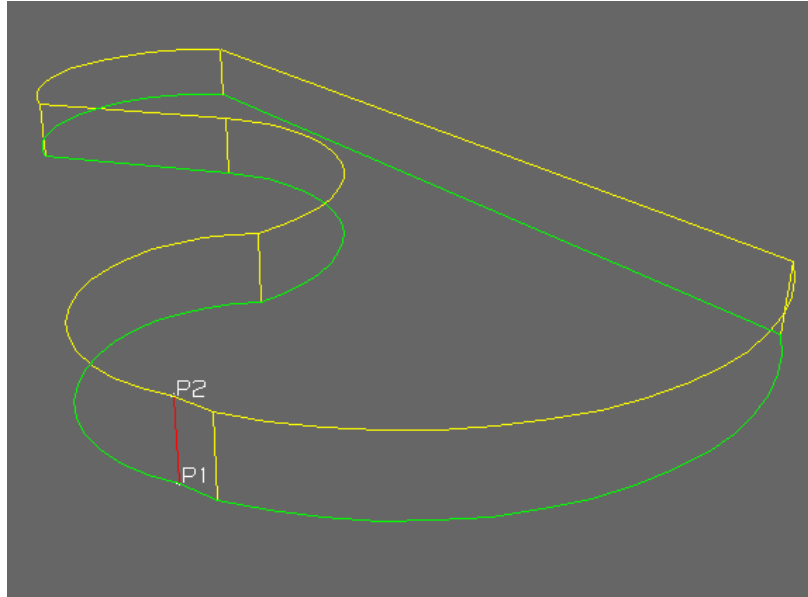
# Load the relevant model
kill all
move p0
get $extrprof

# Makes a profile from loaded vectors and curves
select all
profile e
last
nib 3
ProfileID= inf(28)

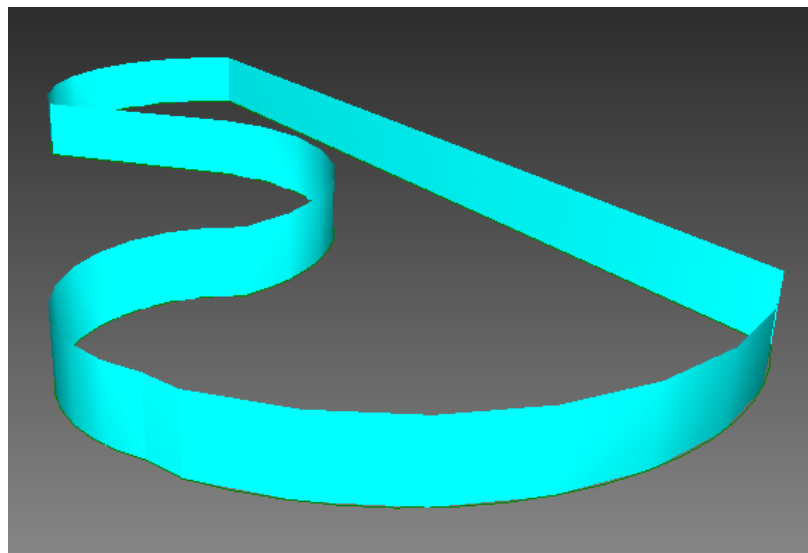
# Setup extrusion direction and length
p1=p0
p2=p0&u100
move p1
line f=2; p2;;
fit

# Creates the Body first
pen 6
Extrusion entity=ProfileID,P1\2,b=on
last
modify col=6
modify f=1
```

```
fit  
polling;'GREEN Profile has been extruded along RED vector ';;
```



```
# Now create a Shell  
pen 5  
Extrusion entity=profileID,P1\2  
last  
modify col=5
```



12 – Surface creation using Revolution

The target of this exercise is to create a surface by revolving an existing Curve around an axis by a given angle. The Disorder command is used to help clarify viewing and the iso-parametric lines are added to better understand surface behavior.

Sample Code :

```
# REVOLVECURVE.CMD
# Declare the required set of variables
points    p1\100
numerics  CurveID

# Define a proper background color
colour b,r=20,g=20,b=20
dynainput mode=1,w1=100

# Load the model created in previous exercise
disorder on
disorder 1,2,3\256

#
kill all
move p0
get $revol

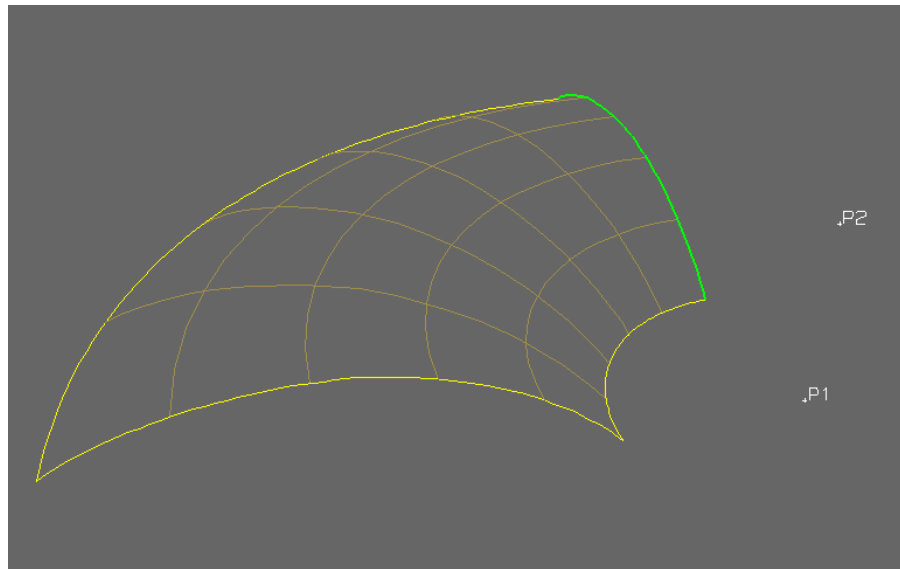
last
modify f=2
nib 3

#
shift n10
CurveID= inf(28)
p1=p0
p2=p0&u10
where P1\2

# Creates the surface
pen 6
Revolution entity=CurveID,P1,P2,90
pen 1
```

```
last
modify f=1
modify col=6

# Use iso-parametric lines for better understanding surface behavior
set cpar=on
set linu=4
set linv=4
fit
polling;'GREEN Curve has been revolved around P1-P2 of 90 degrees';;
```



13 – Body and Shell creation using Revolution

The target of this exercise is to create a Body by revolving an existing Profile around an axis by a given angle.

Sample Code :

```
# REVOLVEPROFILE.CMD
# Declare the required set of variables
points    p1\100
numerics  ProfileID

# Define a proper background color
colour b,r=20,g=20,b=20
dynainput mode=1,w1=100

# Load the model created in previous exercise
disorder on
disorder 1,2,3\256

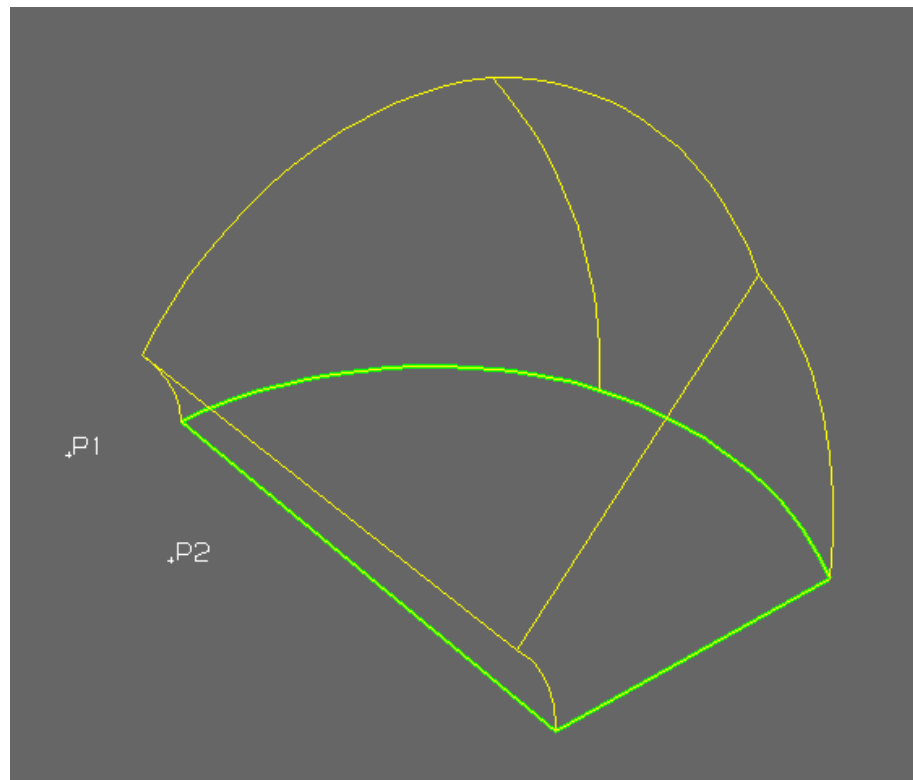
#
kill all
move p0
get $revprof

select all
profile e
last
ProfileID = inf(28)
modify f=2
nib 3
fit

p1=p0&e100
p2=p1&n10
where P1\2

# Creates the body
pen 6
Revolution entity=ProfileID,P1,P2,45,b=on
```

```
last  
modify f=1  
modify col=5  
  
#  
fit  
polling;'GREEN Profile has been revolved around P1-P2 of 45 degrees';;
```



14 – Surface creation using Sweep

The target of this exercise is to create surfaces by sweeping existing Curves along the path represented by other existing Curves, namely a Spine. It also demonstrates the use of the new Convert command that is used to create curve entities from Eagle circles or arcs. The Disorder command is used to help clarify the viewing and iso-parametric lines are added to better understand surface behavior.

Sample Code :

```
# SWEEPCURVE.CMD
# Declare the required set of variables
numerics SpineID, CurveID

# Define a proper background color
colour b,r=20,g=20,b=20
dynainput mode=1,w1=100

# Load the model created in previous exercise
disorder on
disorder 1,2,3\256

#
kill all
move p0
get $curves
fit

last
SpineID= inf(28)
nib 2
modify th=2
modify f=2

partition r
CurveID= inf(28)
nib 3
modify th=2
modify f=2

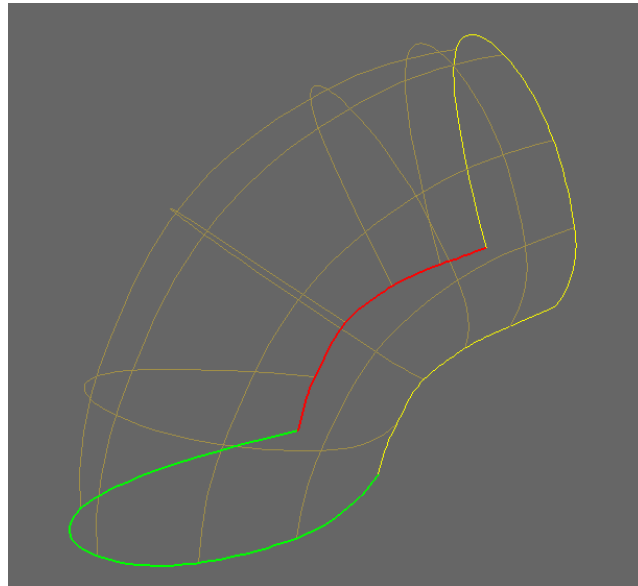
# Creates the first surface
pen 6
Sweep spine=SpineID,entity=CurveID
```

```

pen 1

last
modify f=1
modify col=6

# Use iso-parametric lines for better understanding surface behavior
set cpar=on
set linu=4
set linv=4
fit
polling;'GREEN Curve has been swept along the RED spine ';;
  
```



```

# Creates a surface sweeping a circle along an arc
kill all
move p0
cir r=50

# Generate a curve entity from the created circle
last
Convert
CurveID= inf(28)
nib 3
modify th=2
modify f=2
#
move p0
  
```

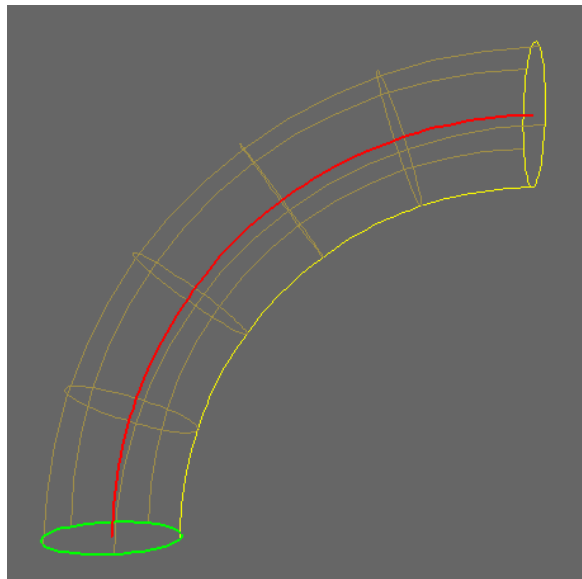
```

arc c=e300,a=90,d=n

# Generate a curve entity from the created arc
last
Convert
SpineID= inf(28)
nib 2
modify th=2
modify f=2

# Creates a second surface
pen 6
Sweep spine=SpineID,entity=CurveID
pen 1
last
modify f=1
modify col=6
fit

polling;'GREEN Curve has been swept along the RED spine ';;
  
```



```

# Creates a surface sweeping a circle along a line
kill all
move p0
cir r=50

# Generate a curve entity from the created circle
last
Convert
  
```

```

CurveID= inf(28)
nib 3
modify th=2
modify f=2
#
move p0
line; u200;;

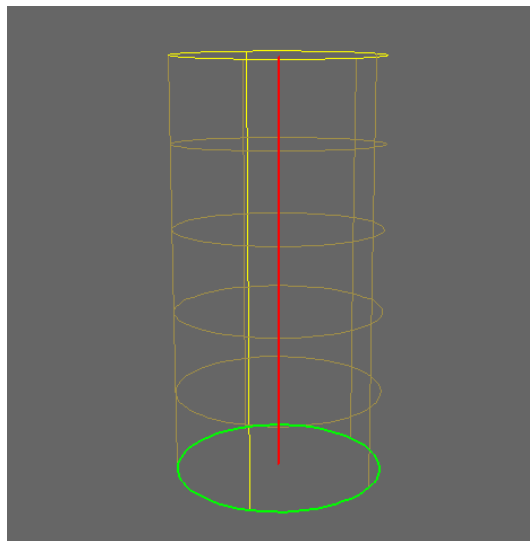
# Generate a curve entity from the created line
last
Convert
SpineID= inf(28)
nib 2
modify th=2
modify f=2

# Creates a second surface
pen 6
Sweep spine=SpineID,entity=CurveID
pen 1

last
modify f=1
modify col=6
fit
polling;'GREEN Curve has been swept along the RED spine ';;

#
disorder off

```



15 – Shell and Body creation using Sweep

The target of this exercise is to create a Body by sweeping existing profiles along the path represented by an existing Curve, namely a spine.

Sample Code :

```
# SWEEPPROFILE.CMD
# Declare the required set of variables
numerics SpineID, ProfileID

# Define a proper background color
colour b,r=40,g=40,b=40
dynainput mode=1,w1=100

#
kill all
move p0
get $prof
fit

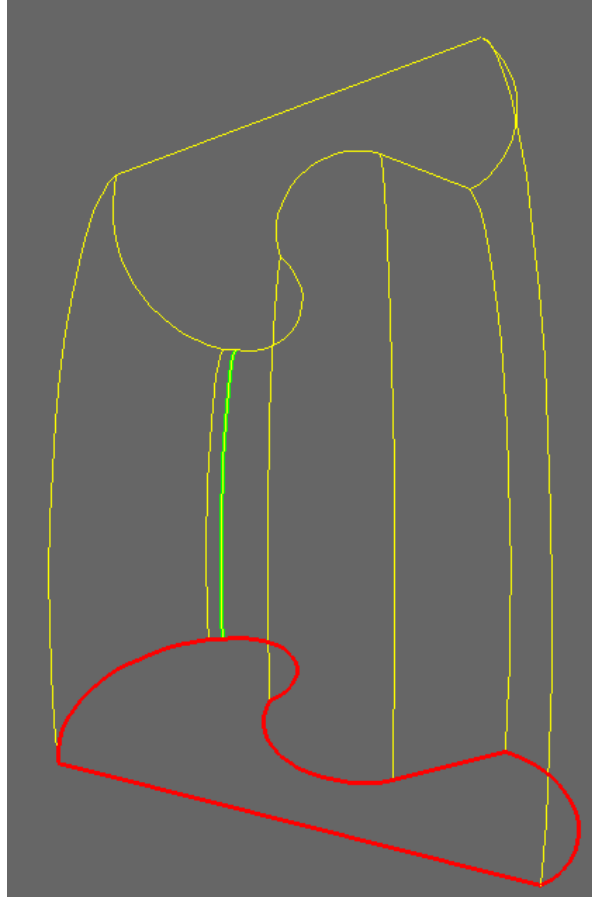
last
select all
profile e
ProfileID= inf(28)
nib 2
modify th=2
modify f=2

partition
move p0
arc c=w800,a=90,d=s
convert
SpineID= inf(28)
nib 3
modify th=2
modify f=2

# Creates the body
pen 6
Sweep entity=profileID,spine=spineID,b=on
pen 1
```

last
modify col=6

fit
polling; 'The RED Profile has been swept along the GREEN spine ';;



16 – Nurbs Configuration Settings - Changes to the INI file

The following new environment variables have been added to determine the behavior of specific commands in relation to NURBS entities.

16.1 *SELECT_RAYTRACE*

This setting enables the selection of an entity also to be computed internally to the surface boundary. This method can be slower than the standard one and therefore it should only be set by the application if the standard selection doesn't provide a valid solution. The variable is read at every time a selection is made.

Default :

`SELECT_RAYTRACE = no`

16.2 *NURBS_FACETED*

This setting indicates if NURBS are to be faceted for rendering in OpenGL or if they should be directly handled as NURBS.

NURBS in OpenGL can be slower than the corresponding faceted representation, but their representation is more precise and smooth. Consequently this option should only be set by the application if the normal faceting does not provide an acceptable result.

Faceting is performed using an approximation of the geometry that is driven by a parameter called "deviation". This can be described as the maximum distance from the triangulation to the real surface.

Its value is heuristically computed as "boxSize * 0.001 * 4", where "boxSize" is the size of the bounding box of the given surface. This value is used as default, that is for when SET DTOL=OFF. The user can change this behavior by specifying SET DTOL=ON and then provide the requested value by means of the SET CTOL=<value>.

There is no lower limit to CTOL, but the program gives a warning message when this value is lower than 0.001. The NURBS_FACETED variable is read at each rendering.

Default :

`NURBS_FACETED=yes`

16.3 NURBS_TRIMMED

This setting enables inclusion of Trimming Curves for NURBS when rendering in OpenGL, consequently this variable only has a meaning when NURBS_FACETED=no.

It is required when the geometry is not just limited by U-V natural bounds. The NURBS_TRIMMED variable is read at every rendering.

Default :

NURBS_TRIMMED=no

16.4 IGES_COLORS

This setting enables retrieval of color information from entities in an IGES file. If color information is assigned then the corresponding entity in the workspace will be assigned for both the color and the pen attribute. The IGES_COLORS variable is read at IMPORT

Default :

IGES_COLORS = no

16.5 IGES_LEVELS

This setting enables retrieval of level information from entities in an IGES file. If level information is assigned then the corresponding entity in the workspace will be assigned for the layer attribute, according to definitions in TABLAYER.DAT. The IGES_LEVELS variable is read at IMPORT.

Default :

IGES_LEVELS = no

17 - LibEagle Nurbs Extension LibOCC

A development API library has been created that extends the LibEagle functions to support NURBS entities.

The library extension adds the following functions:

17.1 *OCC_init*

The “OCC_init” function is implemented to initialise the library.

Sample

```
void OCC_init()
```

17.2 *OCC_libvers*

The “OCC_libvers” function implements display of information relating to the library version.

Sample

```
void OCC_libvers()
```

17.3 *OCC_open_model*

The “OCC_open_model” function is implemented to extend the currently available “EIF_open_model” function allowing reading and creating of a new type model file according to the changed specification described in the Save section. There are no signature changes with respect to the currently available EIF function.

Sample

```

void OCC_open_model(cname,length,read_write,firstit,error)
char *cname;      /* <i><di> Name of the file to be opened          */
                  /* <i><di> Name of the file to be open with the extension*/
int *length;      /* <i><di> Lenght of the file name          */
char *read_write; /* <i><di> Flag read/write          */
Item *firstit;    /* <i><di> Pointer to the first item          */
int *error;       /* <i><di> Error flag          */
  
```

17.4 OCC_importBREP

“OCC_importBREP” function is implemented to facilitate loading in the Eagle model the BREP file format described in import section

Sample

```

void OCC_ImportBREP(fname,append,firstItem, error)
char *fname;      /* <i><di> Name of the file to be imported          */
int append;       /* <i><di> append flag          */
Item *firstItem;  /* <i><di> Pointer to the first imported item          */
int *error;       /* <i><di> Error flag          */
  
```

17.5 OCC_importIGES

The “OCC_importIGES” function is implemented to facilitate loading in the Eagle model the IGES format described in the import section.

Sample

```

void OCC_ImportIGES(fname,append,firstItem, error)
char *fname;      /* <i><di> Name of the file to be imported          */
int append;       /* <i><di> append flag          */
Item *firstItem;  /* <i><di> Pointer to the first imported item          */
int *error;       /* <i><di> Error flag          */
  
```

17.6 OCC_addShape

The "OCC_addShape" function is implemented to facilitate loading in the Eagle model an individual TopoDS_Shape entity.

Sample

```
int OCC_addShape(ptr, frag, style, hatch, fill, id)
unsigned ptr;      /* <-> Pointer to the TopoDS_Shape */
int frag;          /* <-> The fragment of the item */
int style;         /* <-> Linestyle */
int hatch;        /* <-> Type of hatch */
int *fill;        /* <-> Fill colour */
int *id;          /* <-> Item identifier */
```

17.7 OCC_getShape

The OCC_getShape is implemented to retrieve a pointer to the TopoDS_Shape entity from the given OCC entity defined in the Eagle workspace.

Sample

```
void OCC_getShape(item_ptr, ptr, pos, rotmat, max, min, rotflag,
                 strfac, colfill, pen, hatch, layer)
Item item_ptr;    /* <-> pointer to the Shape item */
unsigned *ptr;    /* <-> Pointer to the TopoDS_Shape */
double *pos;     /* <-> the position of the Body */
double *rotmat;  /* <-> the post-mult rotation matrix */
int *rotflag;    /* <-> rotation flag */
double *strfac;  /* <-> the stretching factors */
int *colfill;    /* <-> colour fill */
int *pen;        /* <-> symbology information */
int *hatch;      /* <-> hatch type */
int *layer;      /* <-> layer */
```

17.8 OCC_addNurbsCurve

The OCC_addNurbsCurve is implemented to create a new Curve entity. This function returns a pointer to the created Curve entity. A NULL pointer indicates that the creation process returned an error.

Sample

```
Item OCC_addNurbsCurve( degree, periodic, numCpts, cPts, weights, numKnots, knots, mults)
int degree;          /* <-> degree */
int periodic;        /* <-> periodic flag */
int numCpts;         /* <-> number of control points */
double cPts[[3]];   /* <-> control points */
double weights[];    /* <-> weights on control points */
int numKnots;        /* <-> number of knots */
double knots[];      /* <-> knot values */
double mults[];      /* <-> multiplicity values */
```

17.9 OCC_addBezierCurve

The OCC_addBezierCurve is implemented to create a new Curve entity. This function returns a pointer to the created Curve entity. A NULL pointer indicates that the creation process returned an error.

Sample

```
Item OCC_addBezierCurve( rational, numCpts, cPts, weights)
int rational;         /* <-> rational flag */
int numCpts;         /* <-> number of control points */
double cPts[[3]];   /* <-> control points */
double weights[];    /* <-> weights on control points */
```

17.10 OCC_initProfile

The OCC_initProfile is implemented to initialise the creation of a new Profile entity. It is used in conjunction with the next OCC_addItemToProfile and OCC_terminateProfile.

Sample

```
void OCC_initProfile(void)
```

17.11 OCC_addItemToProfile

The OCC_addItemToProfile is implemented to allow adding a new item to the current Profile entity. The new item should be any of the basic Eagle items, among Lines, Arcs, Circles, Curves and also another Profile entity. The new item must be geometrically compatible with the existing Profile, which means that its start point must be coincident according to a specified tolerance to the end point of the existing Profile. Also, the individual components of the Profile must be planar. This function returns TRUE on success or FALSE on failure.

Sample

```
int OCC_addItemToProfile(itemPointer)
Item itemPointer; /* <-> Pointer to the existing item to be added */
```

17.12 PCC_terminateProfile

The OCC_terminateProfile is implemented to close the creation of a Profile entity. It is used in conjunction with previous OCC_initProfile and OCC_addItemToProfile. This function returns a pointer to the created Profile entity. A NULL pointer indicates that the creation process returned an error.

Sample

```
Item OCC_terminateProfile(erase)
int erase; /* <-> Flag to indicate if the added items must be
removed from the workspace */
```

17.13 OCC_convert

The OCC_convert is implemented to convert one single entity from/to Eagle items to/from OCC-curves, according to the behaviour of the Convert command. This function returns TRUE on success or FALSE on failure.

Sample

```
int OCC_convert(itemPointer)
Item itemPointer; /* <-> Pointer to the existing item to be converted */
```

17.14 OCC_extrusion

The OCC_extrusion is implemented to enable creation of a Surface entity representing an extrusion of a curve between two points. This function returns a pointer to the created Shape entity. A NULL pointer indicates that the creation process returned an error.

Sample

```
Item OCC_extrusion(pointer, p1, p2)
Item pointer; /* <-> Pointer to the Curve item */
double p1[3]; /* <-> First point defining the extrusion vector */
double p2[3]; /* <-> Second point defining the extrusion vector */
```

17.15 OCC_revolution

The OCC_revolution is implemented to enable creation of a Surface entity representing the rotational sweep of a Curve. This function returns a pointer to the created Shape entity. A NULL pointer indicates that the creation process returned an error.

Sample

```

ItemOCC_revolution(pointer, p1, p2, angle)
Item pointer;          /* <-> Pointer to the Curve item          */
double p1[3];         /* <-> First point defining the axis          */
double p2[3];         /* <-> Second point defining the axis         */
double angle;         /* <-> The angle of revolution, in degrees    */
  
```

17.16 The OCC_sweep

The OCC_sweep is implemented to enable creation of a Surface entity representing the sweep of a Curve along a Spine. This function returns a pointer to the created Shape entity. A NULL pointer indicates that the creation process returned an error.

Sample

```

ItemOCC_sweep(curvePointer, spinePointer)
Item curvePointer;    /* <-> Pointer to the Curve item          */
Item spinePointer;    /* <-> Pointer to the Spine item          */
  
```

17.17 OCC_pointInShape

The OCC_pointInShape is implemented that allows checking if a point belongs to an entity. If the specified item is a Body then returned values can be:

- 1 *if the point is inside the entity*
- 0 *if the point is on the entity*
- 1 *if the point is outside the entity*

If the specified item is a Curve or a Shell then returned values could be:

- 0 *if the point is on the entity*
- 1 *if the point is outside the entity*

If the specified item is a Surface then returned values can be:

- 0 *if the point is on the entity*
- 1 *if the point is in the same direction of the normal.*
- 1 *if the point is in the opposite direction of the normal.*

This function returns TRUE on success or FALSE on failure.

Sample

```

int OCC_pointInShape(itemPointer,pt,result)
Item itemPointer;    /* <i> Pointer to the item to be analysed    */
double *pt;         /* <i> Point on the item to be analysed    */
int *result;        /* <i> Result of the check: 0, 1 or -1    */
  
```

17.18 OCC_CalcSurfaceNormal

The OCC_CalcSurfaceNormal is implemented that allows calculating the normal of a Surface given a point on it. This function returns TRUE on success or FALSE on failure.

Sample

```

int OCC_calcSurfaceNormal(itemPointer,pt, normal)
Item itemPointer;    /* <i> Pointer to the item to be analysed    */
double *pt;         /* <i> Point on the item to be analysed    */
double *normal;     /* <i> Returned normal    */
  
```

18 - OCC Import and Export

The Import command is enhanced in V14 to enable loading of NURBS entities into the Eagle Stack (workspace). These NURBS entities are described in an external file. The Import command activates the appropriate callback activated according to the file extension. It also allows adding entities to the workspace, which means loading the required items without first cleaning the workspace.

Prototype:

```
IMPORT <filename>{a}
```

The optional "a" primer to the IMPORT command enables loading the entities into the workspace in append-mode, i.e. without cleaning the existing workspace.

The Import command is extended to support the following file formats:

- a. BREP.
Supported by OCC.
- b. IGES.
IGES files compliant to versions up to and including version 5.3 can be imported.

The following table maps IGES entities to BREP and Eagle entities:

IGES entity	BREP entity	Eagle entity
100: Circular Arc	Edge	Arc
102: Composite Curve	Wire	Curve
104: Conic Arc	Edge	Curve
110: Line	Edge	Line
112: Parametric Spline Curve	Edge / Wire	Curve
126: BSpline Curve	Edge / Wire	Curve
130: Offset Curve	Edge / Wire	Curve
141: Boundary	Wire	Curve
142: Curve On Surface	Wire	Curve
108: Plane	Face	Surface

114: Parametric Spline Surface	Face	Surface
118: Ruled Surface	Face / Shell	Surface / Shell
120: Surface Of Revolution	Face / Shell	Surface / Shell
122: Tabulated Cylinder	Face / Shell	Surface / Shell
128: BSpline Surface	Face	Surface
140: Offset Surface	Face	Surface
143: Bounded Surface	Face / Shell	Surface / Shell
144: Trimmed Surface	Face / Shell	Surface / Shell
190: Plane Surface	Face	Surface
186: ManifoldSolid	Solid	Body
514: Shell	Shell	Shell

The following table maps BREP entities to Eagle entities:

BREP entity	Eagle entity
BSpline Curve	Curve
Bezier Curve	Curve
Trimmed Curve	Curve
Circle	Arc
VectorWithMagnitude	Line
Ellipse	Curve
Offset Curve	Curve
BSpline Surface	Surface
Offset Surface	Surface
SurfaceOfLinearExtrusion	Surface
SurfaceOfRevolution	Surface
SweptSurface	Surface
RectangularTrimmedSurface	Surface

Meaning that the Eagle V14 Nurbs version model includes the following new item types:

Eagle Item Description	Eagle Item-type code
Body	23
Shell	24
Surface	25
Curve	26

The IMPORT command loads into the workspace from the data file all the entities that match the above entity types.

IGES and BREP entities that are not listed in the above table will be skipped and reported in a log file.

NURBS entities are defined in the workspace containing their geometrical instance (position and rotation matrix) and their attributes (color, pen, hatch and alphanumeric attributes), while the inside definition of the entity is handled by OCC, is retrieved by Eagle through the OCC library and saved in the model file using the OCC-BREP representation, as described SAVE command.

18.1 Layers

BREP files do not support non-geometrical information such as layer color and thickness. On the other hand, IGES files do contain these information and support for this is implemented as follows. Phase 1.1 of the implementation included the extension of the IMPORT command to support level (layer) and colour attributes from the imported files.

IMPORT of layer information requires the TABLAYER configuration file to facilitate layer mapping. This file is the same which is already used for importing DXF/DWG files. The format of the files is as follows:

TABLAYER

Eagle layer	Entity layer
<n>	<value 1>
<m>	<value 2>
...	...

This functionality is driven by a new environment variable:

INI Entry	
	IGES_LEVELS=yes no.

The default behaviour is not to import this information.

18.2 Colors

When importing Colour information, Eagle searches the nearest (R,G,B) definition available in the currently installed colour map and then associates the corresponding Eagle-colour index to the item, to both the Colour and Pen attributes. This means that the association is achieved automatically without the need for a configuration file.

Enabling this functionality is driven by an environment variable:

INI Entry	
	IGES_COLORS=yes no.

The default behavior is not to import this information.