Appendix A. Input Data Files

A.1 Introduction

The input data for the model surfaces are obtained from SDRC's I-DEAS Master Series 8.
The file "*fuselage*" contains the 3-D data points of the model surface: aircraft fuselage. The fuselage surface is a bicubic uniform B-spline surface composed of 10×10 patches.
The file "*f_upper_trimdata*" contains the 2-D data points of the open trimming curve from the conversion of the closed trimming curve, and it relimits the upper portion of the fuselage.
The file "*f_lower_trimdata*" contains the 2-D data points of the open trimming curve from the conversion of the closed trimming curve, and it relimits the lower portion of the fuselage.

The file "*wing_upper*" contains the 3-D data points of model surface: the upper portion of the aircraft wing after surface subdivision. The upper wing surface is a bicubic uniform B-spline surface composed of 9×9 patches.
The file "*w_upper_trimdata*" contains the 2-D data points of the open trimming curve for the upper half wing, and it relimits the upper portion of the wing.

The file "*wing_lower*" contains the 3-D data points of model surface: the lower portion of the aircraft wing after surface subdivision. The lower wing surface is a bicubic uniform B-spline surface composed of 9×9 patches.
The file "*w_lower_trimdata*" contains the 2-D data points of the open trimming curve for the lower half wing, and it relimits the lower portion of the wing.

A.2 "fuselage"

- ***Input Data File for the original fuselage***

```
11 11

-0.2628415      0.8474515      -0.013775
-0.09868316     0.7589718      -0.005172
0.03342164      0.6274693      0.001752
0.1227748       0.4636245      0.006434
0.1721154       0.2833392      0.00902
0.1908291       0.09719293     0.010001
0.1875508       -0.08994049    0.009829
0.167321        -0.27601       0.008769
0.1251663       -0.4582559     0.00656
0.05131095      -0.6298816     0.002689
-0.06184512     -0.7782089     -0.003241

-0.2706919      0.8474515      0.136019
-0.1065336      0.7589718      0.144623
0.02557124      0.6274693      0.151546
0.1149244       0.4636245      0.156229
0.164265        0.2833392      0.158815
0.1829787       0.09719293     0.159795
0.1797004       -0.08994049    0.159624
0.1594706       -0.27601       0.158563
0.1173159       -0.4582559     0.156354
0.04346055      -0.6298816     0.152484
-0.06969551     -0.7782089     0.146553

-0.2785423      0.8474515      0.285814
-0.1143839      0.7589718      0.294417
0.01772085      0.6274693      0.30134
0.107074        0.4636245      0.306023
0.1564146       0.2833392      0.308609
0.1751283       0.09719293     0.30959
0.17185         -0.08994049    0.309418
0.1516202       -0.27601       0.308358
0.1094655       -0.4582559     0.306149
0.03561016      -0.6298816     0.302278
-0.0775459      -0.7782089     0.296348

-0.2863926      0.8474515      0.435608
-0.1222343      0.7589718      0.444212
0.009870456     0.6274693      0.451135
0.09922362      0.4636245      0.455818
0.1485642       0.2833392      0.458403
0.1672779       0.09719293     0.459384
0.1639996       -0.08994049    0.459212
0.1437698       -0.27601       0.458152
0.1016151       -0.4582559     0.455943
0.02775977      -0.6298816     0.452072
-0.0853963      -0.7782089     0.446142

-0.294243       0.8474515      0.585403
-0.1300847      0.7589718      0.594006
0.002020062     0.6274693      0.600929
0.09137323      0.4636245      0.605612
0.1407138       0.2833392      0.608198
0.1594275       0.09719293     0.609179
0.1561492       -0.08994049    0.609007
0.1359194       -0.27601       0.607947
0.0937647       -0.4582559     0.605737
0.01990937      -0.6298816     0.601867
-0.09324669     -0.7782089     0.595937

-0.3020934      0.8474515      0.735197
-0.1379351      0.7589718      0.7438
```

```
-0.0058303     0.6274693      0.750724
0.08352283     0.4636245      0.755407
0.1328634      0.2833392      0.757992
0.1515771      0.09719293     0.758973
0.1482988      -0.08994049    0.758801
0.128069       -0.27601       0.757741
0.08591431     -0.4582559     0.755532
0.01205898     -0.6298816     0.751661
-0.1010971     -0.7782089     0.745731

-0.3099438     0.8474515      0.884992
-0.1457855     0.7589718      0.893595
-0.01368072    0.6274693      0.900518
0.07567244     0.4636245      0.905201
0.125013       0.2833392      0.907787
0.1437268      0.09719293     0.908768
0.1404484      -0.08994049    0.908596
0.1202186      -0.27601       0.907536
0.07806392     -0.4582559     0.905326
0.004208586    -0.6298816     0.901456
-0.1089475     -0.7782089     0.895525

-0.3177942     0.8474515      1.034786
-0.1536359     0.7589718      1.043389
-0.02153112    0.6274693      1.050313
0.06782205     0.4636245      1.054995
0.1171626      0.2833392      1.057581
0.1358764      0.09719293     1.058562
0.132598       -0.08994049    1.05839
0.1123682      -0.27601       1.05733
0.07021352     -0.4582559     1.055121
-0.00364187    -0.6298816     1.05125
-0.1167979     -0.7782089     1.04532

-0.3256446     0.8474515      1.184581
-0.1614863     0.7589718      1.193184
-0.02938151    0.6274693      1.200107
0.05997165     0.4636245      1.20479
0.1093122      0.2833392      1.207376
0.128026       0.09719293     1.208356
0.1247476      -0.08994049    1.208185
0.1045178      -0.27601       1.207124
0.06236313     -0.4582559     1.204915
-0.0114922     -0.6298816     1.201045
-0.1246483     -0.7782089     1.195114

-0.333495      0.8474515      1.334375
-0.1693367     0.7589718      1.342978
-0.03723191    0.6274693      1.349901
0.05212126     0.4636245      1.354584
0.1014618      0.2833392      1.35717
0.1201756      0.09719293     1.358151
0.1168972      -0.08994049    1.357979
0.09666742     -0.27601       1.356919
0.05451274     -0.4582559     1.35471
-0.01934259    -0.6298816     1.350839
-0.1324987     -0.7782089     1.344909

-0.3413454     0.8474515      1.484169
-0.1771871     0.7589718      1.492773
-0.0450823     0.6274693      1.499696
0.04427087     0.4636245      1.504379
0.09361142     0.2833392      1.506964
0.1123252      0.09719293     1.507945
0.1090469      -0.08994049    1.507773
0.08881703     -0.27601       1.506713
0.04666234     -0.4582559     1.504504
-0.02719299    -0.6298816     1.500633
-0.1403491     -0.7782089     1.494703
```

A.3 "f_upper_trimdata"

- ***Parametric values of the upper portion of the trimming curve***

21

5.198310289	0
5.198310289	0.61578637
5.198310289	1.22330432
5.198310289	1.83601342
5.198310289	2.44085469
4.80407117	2.71842525
4.50286454	3.17443372
4.27522422	3.68526908
4.09222701	4.22588379
3.95100074	4.78556821
3.84596372	5.35361243
3.77691841	5.93687699
3.76568771	6.52119198
3.85506397	7.0977747
4.04247876	7.63673474
4.34541639	8.07818596
4.74324467	8.37242107
5.198310289	8.43296074
5.198310289	8.95240918
5.198310289	9.47847904
5.198310289	10

A.4 "f_lower_trimdata"

Parametric values of the lower portion of the trimming curve

21

4.8	0
4.8	0.616
4.8	1.2233
4.8	1.836
4.8	2.441
4.553	2.785
4.481	3.276
4.43	3.769
4.38	4.255
4.344	4.753
4.318	5.246
4.308	5.749
4.307	6.244
4.3124	6.744
4.336	7.239
4.3074	7.731
4.487	8.205
4.8	8.433
4.8	8.952
4.8	9.48
4.8	10

A.5 "upper_wing"

- ***Input Data File for the upper portion of the original wing***

```
10 10
1.018304        0.06644559      0.3230724
0.9073450       0.06644559      0.3288875
0.7963862       0.06644559      0.3347026
0.6854273       0.06644559      0.3405178
0.5744685       0.06644559      0.3463329
0.4635097       0.06644559      0.3521480
0.3525508       0.06644559      0.3579631
0.2415920       0.06644559      0.3637782
0.1306332       0.06644559      0.3695933
0.01967432      0.06644559      0.3754084

1.021987        0.1687421       0.3933542
0.9110283       0.1687421       0.3991693
0.8000695       0.1687421       0.4049844
0.6891107       0.1687421       0.4107995
0.5781518       0.1687421       0.4166146
0.4671930       0.1687421       0.4224297
0.3562341       0.1687421       0.4282448
0.2452753       0.1687421       0.4340600
0.1343165       0.1687421       0.4398751
0.02335763      0.1687421       0.4456902

1.027564        0.2371926       0.4997699
0.9166053       0.2371926       0.5055851
0.8056465       0.2371926       0.5114002
0.6946877       0.2371926       0.5172153
0.5837288       0.2371926       0.5230304
0.4727700       0.2371926       0.5288455
0.3618112       0.2371926       0.5346606
0.2508523       0.2371926       0.5404757
0.1398935       0.2371926       0.5462908
0.02893464      0.2371926       0.5521059

1.033683        0.2858433       0.6165271
0.9227243       0.2858433       0.6223422
0.8117655       0.2858433       0.6281573
0.7008067       0.2858433       0.6339724
0.5898478       0.2858433       0.6397875
0.4788890       0.2858433       0.6456026
0.3679301       0.2858433       0.6514177
0.2569713       0.2858433       0.6572328
0.1460125       0.2858433       0.6630479
0.03505363      0.2858433       0.6688630

1.040120        0.3164891       0.7393514
0.9291613       0.3164891       0.7451666
0.8182024       0.3164891       0.7509817
0.7072436       0.3164891       0.7567968
0.5962848       0.3164891       0.7626119
0.4853259       0.3164891       0.7684270
0.3743671       0.3164891       0.7742421
0.2634083       0.3164891       0.7800572
0.1524494       0.3164891       0.7858723
0.04149058      0.3164891       0.7916874

1.046682        0.3348570       0.8645633
0.9357234       0.3348570       0.8703784
```

```
0.8247645      0.3348570      0.8761935
0.7138057      0.3348570      0.8820086
0.6028468      0.3348570      0.8878237
0.4918880      0.3348570      0.8936388
0.3809292      0.3348570      0.8994539
0.2699703      0.3348570      0.9052690
0.1590115      0.3348570      0.9110842
0.04805265     0.3348570      0.9168993

1.053282       0.3264946      0.9904958
0.9423232      0.3264946      0.9963109
0.8313644      0.3264946      1.002126
0.7204055      0.3264946      1.007941
0.6094467      0.3264946      1.013756
0.4984878      0.3264946      1.019571
0.3875290      0.3264946      1.025386
0.2765702      0.3264946      1.031202
0.1656113      0.3264946      1.037017
0.05465250     0.3264946      1.042832

1.059408       0.2793571      1.107379
0.9484488      0.2793571      1.113194
0.8374900      0.2793571      1.119010
0.7265311      0.2793571      1.124825
0.6155723      0.2793571      1.130640
0.5046135      0.2793571      1.136455
0.3936546      0.2793571      1.142270
0.2826958      0.2793571      1.148085
0.1717369      0.2793571      1.153900
0.06077811     0.2793571      1.159715

1.063964       0.1890429      1.194315
0.9530049      0.1890429      1.200130
0.8420461      0.1890429      1.205945
0.7310872      0.1890429      1.211761
0.6201284      0.1890429      1.217576
0.5091696      0.1890429      1.223391
0.3982107      0.1890429      1.229206
0.2872519      0.1890429      1.235021
0.1762931      0.1890429      1.240836
0.06533422     0.1890429      1.246651

1.065249       0.06644559     1.218835
0.9542900      0.06644559     1.224650
0.8433311      0.06644559     1.230466
0.7323723      0.06644559     1.236281
0.6214134      0.06644559     1.242096
0.5104546      0.06644559     1.247911
0.3994958      0.06644559     1.253726
0.2885369      0.06644559     1.259541
0.1775781      0.06644559     1.265356
0.06661926     0.06644559     1.271171
```

A.6 "lower_wing"

- ***Input Data File for the lower portion of the original wing***

```
10 10
1.018304      0.06644559    0.3230724
0.9073450     0.06644559    0.3288875
0.7963862     0.06644559    0.3347026
0.6854273     0.06644559    0.3405178
0.5744685     0.06644559    0.3463329
0.4635097     0.06644559    0.3521480
0.3525508     0.06644559    0.3579631
0.2415920     0.06644559    0.3637782
0.1306332     0.06644559    0.3695933
0.01967432    0.06644559    0.3754084

1.022619      0.01199421    0.4054021
0.9116597     0.01199421    0.4112172
0.8007009     0.01199421    0.4170323
0.6897421     0.01199421    0.4228474
0.5787832     0.01199421    0.4286625
0.4678244     0.01199421    0.4344776
0.3568655     0.01199421    0.4402927
0.2459067     0.01199421    0.4461078
0.1349479     0.01199421    0.4519229
0.02398903    0.01199421    0.4577380

1.028201     -0.004161355   0.5119140
0.9172418    -0.004161355   0.5177291
0.8062829    -0.004161355   0.5235442
0.6953241    -0.004161355   0.5293593
0.5843653    -0.004161355   0.5351744
0.4734064    -0.004161355   0.5409895
0.3624476    -0.004161355   0.5468046
0.2514888    -0.004161355   0.5526197
0.1405299    -0.004161355   0.5584348
0.02957109   -0.004161355   0.5642499

1.033816     -0.01537891    0.6190544
0.9228568    -0.01537891    0.6248695
0.8118979    -0.01537891    0.6306846
0.7009391    -0.01537891    0.6364998
0.5899803    -0.01537891    0.6423149
0.4790214    -0.01537891    0.6481300
0.3680626    -0.01537891    0.6539451
0.2571038    -0.01537891    0.6597602
0.1461449    -0.01537891    0.6655753
0.03518608   -0.01537891    0.6713904

1.039439     -0.02492892    0.7263633
0.9284806    -0.02492892    0.7321784
0.8175218    -0.02492892    0.7379935
0.7065629    -0.02492892    0.7438087
0.5956041    -0.02492892    0.7496238
0.4846452    -0.02492892    0.7554389
0.3736864    -0.02492892    0.7612540
0.2627276    -0.02492892    0.7670691
0.1517687    -0.02492892    0.7728842
0.04080990   -0.02492892    0.7786993
```

1.045083 -0.02767209 0.8340536
0.9341244 -0.02767209 0.8398687
0.8231656 -0.02767209 0.8456838
0.7122067 -0.02767209 0.8514989
0.6012479 -0.02767209 0.8573140
0.4902891 -0.02767209 0.8631291
0.3793302 -0.02767209 0.8689442
0.2683714 -0.02767209 0.8747593
0.1574125 -0.02767209 0.8805744
0.04645371 -0.02767209 0.8863895

1.050729 -0.02806410 0.9417855
0.9397704 -0.02806410 0.9476006
0.8288116 -0.02806410 0.9534157
0.7178527 -0.02806410 0.9592308
0.6068939 -0.02806410 0.9650460
0.4959350 -0.02806410 0.9708611
0.3849762 -0.02806410 0.9766762
0.2740174 -0.02806410 0.9824913
0.1630585 -0.02806410 0.9883064
0.05209970 -0.02806410 0.9941215

1.056370 -0.02335224 1.049416
0.9454111 -0.02335224 1.055232
0.8344523 -0.02335224 1.061047
0.7234934 -0.02335224 1.066862
0.6125346 -0.02335224 1.072677
0.5015757 -0.02335224 1.078492
0.3906169 -0.02335224 1.084307
0.2796581 -0.02335224 1.090122
0.1686992 -0.02335224 1.095937
0.05774040 -0.02335224 1.101752

1.061959 -0.01017789 1.156054
0.9509998 -0.01017789 1.161870
0.8400409 -0.01017789 1.167685
0.7290821 -0.01017789 1.173500
0.6181232 -0.01017789 1.179315
0.5071644 -0.01017789 1.185130
0.3962056 -0.01017789 1.190945
0.2852467 -0.01017789 1.196760
0.1742879 -0.01017789 1.202575
0.06332906 -0.01017789 1.208390

1.065249 0.06644559 1.218835
0.9542900 0.06644559 1.224650
0.8433311 0.06644559 1.230466
0.7323723 0.06644559 1.236281
0.6214134 0.06644559 1.242096
0.5104546 0.06644559 1.247911
0.3994958 0.06644559 1.253726
0.2885369 0.06644559 1.259541
0.1775781 0.06644559 1.265356
0.06661926 0.06644559 1.271171

A.7 "w_upper_trimdata"

- ***Parametric values of the upper portion of the trimming curve***

```
14
7.452615888    0
7.498244682    0.715849092
7.575342993    1.400416317
7.652508696    2.115873081
7.726936896    2.831422833
7.815337335    3.515642721
7.889604084    4.205885022
7.95976605     4.889419569
8.00999289     5.584876551
8.017186266    6.279005547
7.998340284    6.953275566
7.951603374    7.656998013
7.904047986    8.361114615
7.880193135    9
```

A.8 "w_lower_trimdata"

- ***Parametric values of the lower portion of the trimming curve***

```
14
7.452  0
7.4718 0.6975
7.5051 1.3995
7.5564 2.0844
7.5843 2.772
7.6194 3.4668
7.659  4.1715
7.6869 4.8591
7.7247 5.5413
7.7598 6.2334
7.758  6.9201
7.8237 7.6212
7.8561 8.3088
7.88   9
```

Appendix B. C++ Source Code Functional Description

B.1 Introduction

The Appendix is a list of C++ source code functions for this thesis. The files are listed in alphabetical order.

The description of each function includes function prototype, input, output arguments and function output.

B.2 basis.C

B.2.1 basis

Description:

Computes the active basis functions for a non-uniform cubic B-spline in a non-recursive procedure.

Function Prototype:

void basis (float, int, float [], float [])

Input Arguments:

float u; parametric value
int interval; knot interval
float knots[]; knot array

Output Arguments:

float N[4]; value of the four active basis functions

Function Output:

None

B.2.2 dbasis

Description:
Computes the derivatives of the active basis functions of a non-uniform cubic B-spline in a non-recursive procedure.

Function Prototype:
void dbasis (float, int, float [], float [])

Input Arguments:
float u; parametric value
int interval; knot interval
float knots[]; knot array

Output Arguments:
float dN[4]; the derivatives of the four active basis functions

Function Output:
None

B.3 constraints.C

B.3.1 constraintu

Description:
Computes the derivatives in u parametric direction at the trimming curve for the new surface.
Function Prototype:
void constraintu(int, int, float [], float [], float [], float [], float[], float [], float [], float [])

Input Arguments:
int num; trimmed surface's number of points in u parametric direction
int trimnum; trimmed surface's number of points in v parametric direction
float u_knots[]; array of knots in u parametric direction
float v_knots[]; array of knots in v parametric direction
float control_x[]; x coordinate component for the array of control points
float control_y[]; y coordinate component for the array of control points
float control_z[]; z coordinate component for the array of control points
float data[]; array of u, v parametric values for which the derivative is computed

Output Arguments:
float derivatives; array of derivative constraints in u direction

Function Output:
None

B.3.2 constraintv

Description:
Computes the derivatives in v parametric direction at the trimming curve for the new surface.

Function Prototype:
void constraintv(int, int, float [], float [], float [], float [], float [], float [], float [])

Input Arguments:

int num;	trimmed surface's number of points in u parametric direction
int trimnum;	trimmed surface's number of points in v parametric direction
float u_knots[];	array of knots in u parametric direction
float v_knots[];	array of knots in v parametric direction
float control_x[];	x coordinate component for the array of control points
float control_y[];	y coordinate component for the array of control points
float control_z[];	z coordinate component for the array of control points
float data[];	array of u, v parametric values for which the derivative is computed

Output Arguments:

float derivatives;	array of derivative constraints in v direction

Function Output:
None

B.4 create_surf.C

B.4.1 create_surf

Description:
Computes surface data for rendering B-spline surface by OpenGL.

Function Prototype:
void create_surf (int[], float[], float[], float[],float[], int[])

Input Arguments:

int numpts[];	array of the number of data points, numpts[0] = point number of a row numpts[1] = point number of a column
float datapts[];	array of data points to be used to create NUBS3 surface

Output Arguments:

float u_newknots[];	array of knots in u parametric direction
float v_newknots[];	array of knots in v parametric direction
float contrlpts[];	array of control points for rendering surface
int newnum[];	array of the number of new knots

Function Output:
None

B.5 curv_knot_constraint.C

B.5.1 curv_new_knot_num

Description:
Computes the number of new knots in the knot sequence based on the continuity constraints for a cubic B-spline curve.

Function Prototype:
int curv_new_knot_num (int, int[])

Input Arguments:
int numpts; number of data points
int continuities[]; array of continuities at domain knots

Output Arguments:
none

Function Output:
Number of new knots

B.5.2 curv_add_knot_const

Description:
Impose continuity constraints on the knot sequence

Function Prototype:
void curv_add_knot_const (int, float, int, int[], int, float[])

Input Arguments:
int numpts; number of data points
float knots[]; array of original knots sequence
int opcl; flag for open or closed trimming curve
int continuities[]; array of continuities at domain knots

Output Arguments:
int newnum; number of new knots

float newknots[]; new knot sequence;

Function Output:
None

B.6 display.C

B.6.1 display_g

Description:
Display the NURBS surface given the control point array, the knot vector and the order of the polynomial basis.

Function Prototype:
void display_g (float[], float[], float[], int[], Display*, Window, GLXContext)

Input Arguments:
float ut_newknots[]; array of knots in u parametric direction
float vt_newknots[]; array of knots in v parametric direction
float t_contrlpts[]; array of control points used to create surface
int t_newnum[]; array of the number of knots
Display *dpy; XtDisplay (glxarea)
Window win; XtWindow (glxarea)
GLXContext glxcontext; OpenGL rendering context

Output Arguments:
none

Function Output:
none

B.7 knot_constraints.C

B.7.1 new_knot_num

Description:
Computes the number of new knots in the knot sequence based on the continuity constraints.

Function Prototype:
void new_knot_num (int, float [])

Input Arguments:
int numpts; number of data points
float continuities []; array of continuities at domain knots

Output Arguments:
none
Function Output:
number of new knots

B.7.2 add_knot_const

Description:
Imposes continuity constraints on the knot sequence.

Function Prototype:
void add_knot_const (int, float [], int [], int, int, float [])

Input Arguments:
int numpts; number of data points
float knots[]; array of original knot sequence
int continuities[]; array of continuities at domain knots
int opcl; in this thesis it is always set as 0 for open trimming curve

Output Arguments:
float newknots []; new knot sequence

Function Output:
None

B.8 map_point.C

B.8.1 map_point

Description:
Computes a single point on NURBs, one coordinate at a time.

Function Prototype:
void map_point (int, float [], float [], float [], int, int, float [])

Input Arguments:
int num; surface's number of points in u parametric direction
float Nu [4]; surface's basis function in u parametric direction

float Nv [4];	surface's basis function in v parametric direction
float ctrlpt[];	surface's control points
int intU;	interval where the point is located in u parametric direction
int intV;	interval where the point is located in v parametric direction

Output Arguments:

float coordinate;	x, y or z component of the point on the surface

Function Output:
none

B.9 model_curve.C

B.9.1 create_NUBS3_curve

Description:
Creates the describing data for cubic non-uniform B-spline curve.

Function Prototype:
void create_NUBS3_curve (int, float [], int, int [], float [], float [], float [])

Input Arguments:

int numpts;	number of data points
float datpts [];	array of data points
int param;	type of parametrization
int continuities[];	array of continuities
float constraints[];	array of tangent vector constraints

Output Arguments:

float knots[];	array of knots
float ctrlpts[];	array of control points

Function Output:
none

B.10 model_surface.C

B.10.1 create_NUBS3_surface

Description:
Creates the describing data for a cubic non-uniform B-spline surface.

Function Prototype:
void create_NUBS3_surface (int, float [], int, int [], int [], float [], float [], float [], float [], float [])

Input Arguments:

int numpts [];	array of the number of points in each parametric direction (u, v)
float datpts [];	array of data points, one row in u parametric direction at a time
int param;	type of parametrization
int u_continuities[];	array of continuities at each u knot domain
int v_continuities[];	array of continuities at each v knot domain
float u_constraints[];	array of tangent vector in u parametric direction, one row in u parametric direction at a time
float v_constraints[];	array of tangent vector in v parametric direction, one row in v parametric direction at a time

Output Arguments:

float uknots[];	array of knots in u parametric direction
float vknots[];	array of knots in v parametric direction
float ctrlpts[];	array of control points

Function Output:
none

B.11 offset.C

B.11.1 calc_offset

Description:
Calculate the trim curve offset in parametric domain for the wing.

Function Prototype:
void calc_offset(float, float[], float[])

Input Arguments:

float offset;	the distance of trim curve need to be shifted

float trim_data[]; original parametric data of trimmming curve

Output Arguments:
float trim_offset[]; shifted parametric data of trim curve;

Function Output:
none

B.11.2 calc_offset_w

Description:
Calculate the trim curve offset in parametric domain for the fuselage.

Function Prototype:
void calc_offset_w (float, float[], float[], int)

Input Arguments:
float offset; the distance of trim curve need to be shifted
float trim_data[]; original parametric data of trimmming curve

Output Arguments:
float trim_offset[]; shifted parametric data of trim curve
int trimnum; number of points on the arc portion of trimming curve

Function Output:
none

B.12 parameterize.C

B.12.1 parametrize

Description:
Parametrizes a cubic B-spline curve using either uniform, chord-length, or centripedal parametrization.

Function Prototype:
void diff_vol (int, float [], int, int, float [])

Input Arguments:
int numpts; number of data points
float datpts [ ]; array of data points

int opcl; flag (0 = open trimming curve, 1 = closed trimming curve)
int param; type of parametrization

Output Arguments:
float knots []; knot sequence

Function Output:
None

B.12.2 pt_dist

Description:
Calculates the distance between two 3-D points.

Function Prototype:
float pt_distance (int, float [], int, float [])

Input Arguments:
int num1; the number of first points
float first []; the array of first data points
int num2; the number of last points
float last []; the array of last data points

Output Arguments:
none

Function Output:
Distance between the two points

B.13 point_and_derivative.C

B.13.1 point

Description:
Calculate an array of data points on the original surface to create the trimmed surface.

Function Prototype:
void point (int, int, float [], float [], float [], float [], float [], float [], float [])

Input Arguments:

int num;	surface's number of points in u parametric direction
int trimnum;	number of points defining the trimming curve
float u_knots [];	array of knots in u parametric direction
float v_knots [];	array of knots in v parametric direction
float control_x[];	array of control points (x coordinate component)
float control_y[];	array of control points (y coordinate component)
float control_z[];	array of control points (z coordinate component)
float trim_data[];	original surface's data points

Output Arguments:

float trim_points[];	array of points to be used as data points for creating trimmed surface

Function Output:

None

B.13.2 derivative

Description:

Calculates an array of derivatives in u parametric direction.

Function Prototype:

float derivative (int, int, float [], float [], float [], float [], float [], float [], float [])

Function Output:

Distance between the two points

Input Arguments:

int num;	surface's number of points in u parametric direction
int trimnum;	number of points defining the trimming curve
float u_knots [];	array of knots in u parametric direction
float v_knots [];	array of knots in v parametric direction
float control_x[];	array of control points (x coordinate component)
float control_y[];	array of control points (y coordinate component)
float control_z[];	array of control points (z coordinate component)
float trim_data[];	original surface's data points

Output Arguments:

float trim_derivative[]; array of derivative in u parametric direction for the new data

Function Output:

None

B.13.3 derivative2

Description:
Calculates an array of derivatives in v parametric direction.

Function Prototype:
float derivative2 (int, int, float [], float [], float [], float [], float [], float [], float [])

Function Output:
Distance between the two points

Input Arguments:

int num;	surface's number of points in u parametric direction
int trimnum;	number of points defining the trimming curve
float u_knots [];	array of knots in u parametric direction
float v_knots [];	array of knots in v parametric direction
float control_x[];	array of control points (x coordinate component)
float control_y[];	array of control points (y coordinate component)
float control_z[];	array of control points (z coordinate component)
float trim_data[];	original surface's data points

Output Arguments:
float trim_derivative[]; array of derivative in v parametric direction for the new data

Function Output:
None

B.14 refine.C

B.14.1 refine_pts

Description:
Adding three v iso-parametric curve near the trimming boundaries, it should be implemented by knot insertion algorithm

Function Prototype:
void refine_pts (int, int, int, float[],float[])

Input Arguments:

int rownum;	row number of data points
int incol;	column number before refinement
int outcol;	column number after refinement
float datain[];	array of data point before refinement

Output Arguments:
float dataout[]; array of data point after refinement

Function Output:
None

B.14.2 conic

Description:
Compute one v iso-parametric curve used to control the shape of fillet

Function Prototype:
void conic (float, float, float[], float[], float[], float[])

Input Arguments:
float t; parameter value for different point on a specific conic
float p; parameter value for a specific conic, it can be used to control the radius of the fillet
float one[]; array of data point, 1 vertex for defining a conic
float two[]; array of data point, 2 vertex for defining a conic
float three[]; array of data point, 3 vertex for defining a conic

Output Arguments:
float out[]; array of data point for intermidiate curve on the fillet

Function Output:
None

B.15 rendering.C

B.15.1 main

Description:
Open a user interactive platform to compute and display the objects.
This is the main program entry.

Function Prototype:
int main (int, char**)

Input Arguments:
int argv; number of command line arguments
int argv[]; array of text strings corresponding to the words in on

the command line; e.g., argv[0] == the program name

Output Arguments:
None

Function Output:
OpenGL and X-windows workstation are opened

B.15.2 init_surface

Description:
Process the geometric trimming for the model surfaces.

Function Prototype:
int init_surface ( )

Input Arguments:
none

Output Arguments:
none

Function Output:
Surface data

B.16 setup_system.C

B.16.1 setup_system

Description:
Set up basis function matrix for a given knot sequence.

Function Prototype:
void setup_system (int, float [], int, int [], int, float [])

Input Arguments:

int numpts;	number of data points
float knots [];	array of knots
int opcl;	flag: (0 = open trimming curve, 1 = closed trimming curve)
int continuities [];	array of continuities

int numctrl; number of control points

Output Arguments:
float Bmat[]; basis function matrix

Function Output:
None

B.17 solve.C

B.17.1 solve_lin_syst

Description:
Solves a linear system of equations given a matrix of size n × n + 1.

Function Prototype:
void solve_lin_syst (int, float [], float [], int)

Input Arguments:
int n; number of equations
float Mat []; matrix

Output Arguments:
float vect[]; solution vector
int error; error flag

Function Output:
None

B.18 solve_systems.C

B.18.1 setup_constraint_matrix

Description:
Assembles a B-spline surface constraints matrix

Function Prototype:
void setup_constraints_matrix (int, float [], int [], int [], int [], int [], float [], float [], float [])

Input Arguments:
int numpts [2]; number of data points in each direction
float datapts []; array of data points

int numctrl [2]; number of control points in each direction
int opcl [2]; open/closed flag
int u_continuities []; array of u continuities
int v_continuities []; array of v continuities
float u_constraints []; array of tangent constraints in u parametric direction
float v_constraints []; array of tangent constraints in v parametric direction

Output Arguments:
float constraint_matrix[]; matrix of data point and tangent constraints

Function Output:
None

B.18.2 u_tan_pt_constraints

Description:
Adds constraints to a particular row in u parametric direction of constraints matrix

Function Prototype:
void u_tan_pt_constraint (int, int, int, float [], int, int, int [], float [])

Input Arguments:
int row; row index
int crow; constraints row index
int numpts; number of data points in u parametric direction
int numctrl; number of control points in u parametric direction
int opcl; open/closed flag in u parametric direction
int u_continuities []; array of u continuities
float u_constraints []; array of tangent constraints in u parametric direction

Output Arguments:
float constraint_matrix[]; array of constraint matrix

Function Output:
None

B.18.3 v_tan_pt_constraints

Description:
Adds constraints to a particular row in v parametric direction of constraints matrix

Function Prototype:
void v_tan_pt_constraint (int, int, int, float [], int, int, int [], float [])

Input Arguments:

int row;	row index
int crow;	constraints row index
int numpts;	number of data points in v parametric direction
int numctrl;	number of control points in v parametric direction
int opcl;	open/closed flag in v parametric direction
int v_continuities [];	array of v continuities
float v_constraints [];	array of tangent constraints in v parametric direction

Output Arguments:
float constraint_matrix[]; array of constraint matrix

Function Output:
None

B.19 sur_knots.C

B.19.1 average_parametrization

Description:
driver to calculate u & v surface knots based on chord length parametrization

Function Prototype:
void average_parametrization (int[], float[], int[], int, float[], float[])

Input Arguments:

int numpts[];	array of number of data points in both parametric directions,
float datpts[];	array of data pts,
int opcl[];	array of open/closed flags (u,v)
int param;	type of parametrization

Output Arguments:

float u_knots[];	array of knots in u parametric direction
float v_knots[];	array of knots in v parametric direction

Function Output:
None

B.19.2 calc_u_knots

Description:
calculates knots in parametric u direction for cubic NUBS surface.

Function Prototype:
void calc_u_knots (int[], float[], int[], int, float[])

Input Arguments:
int numpts[]; array of number of data points in both parametric directions,
float datpts[]; array of data pts,
int opcl[]; array of open/closed flags (u,v)
int param; type of parametrization

Output Arguments:
float u_knots[]; array of knots in u parametric direction

Function Output:
None

B.19.3 calc_v_knots

Description:
Calculate knots in parametric v direction for cubic NUBS surface.

Function Prototype:
void calc_v_knots (int[], float[], int[], int, float[])

Input Arguments:
int numpts[]; array of number of data points in both parametric directions,
float datpts[]; array of data pts,
int opcl[]; array of open/closed flags (u,v)
int param; type of parametrization

Output Arguments:
float v_knots[]; array of knots in v parametric direction

Function Output:
None

B.19.4 average_u_distance

Description:
Compute the average distance between two u values

Function Prototype:
float average_u_distance (int[], float[], int, int)

Input Arguments:
int numpts[]; array of number of data points in both parametric directions
float datpts[]; array of data points
int n1; first point in u direction, point sequence number in u direction
int n2; second point in u direction

Output Arguments:
None

Function Output:
Average distance between data points in u direction

B.20 volume_grid.C

B.20.1 get_orig_points

Description:
Generate a grid of data points on the original surface to be used for error comparison later.

Function Prototype:
void get_orig_points (int, int, int, int, float [], float [], float [], float [], float [], float [])

Input Arguments:
int num; surface's number of points in u parametric direction
int trimnum; number of points representing the trimming curve
int grid_u; number of sampled data points in u direction
int grid_v; number of sampled data points in v direction
float u_knots []; array of knots in u parametric direction
float v_knots []; array of knots in v parametric direction
float control_x []; array of control points (x coordinate component)
float control_y []; array of control points (y coordinate component)
float control_z []; array of control points (z coordinate component)

Output Arguments:
float points []; array of data point from the original surface

Function Output:
None

B.20.2 get_trim_points

Description:
Generate a grid of data points on the trimmed surface to be used for error comparison later.

Function Prototype:
void get_trim_points (int, int, int, int, float [], float [], float [], float [], float [], float [])

Input Arguments:

int num;	surface's number of points in u parametric direction
int trimnum;	number of points representing the trimming curve
int grid_u;	number of sampled data points in u direction
int grid_v;	number of sampled data points in v direction
float u_knots [];	array of knots in u parametric direction
float v_knots [];	array of knots in v parametric direction
float control_x [];	array of control points (x coordinate component)
float control_y [];	array of control points (y coordinate component)
float control_z [];	array of control points (z coordinate component)

Output Arguments:

float points [];	array of data points from the trimmed surface

Function Output:
None

Appendix C. C++ Source Code

```
/*****************************************************************
* Function Name : basis.C
* Author:  Steven Flemming
*****************************************************************/
/* Function Prototypes */
void basis(float u, int interval, float knots[], float N[]);
void dbasis(float u, int interval, float knots[], float dN[]);
/* Function Definition */
void basis(float u, int interval, float knots[], float N[])
{
  int i,j;
  double dr[3],dl[3],n[4][4],temp,denom;
  n[0][0] = 1.0;
  /* run loop for cubic B-spline */
  for (i=0;i<=2;i++)
    {
      /* calculate intermediate values*/
      dr[i] = knots[interval+i+1] - u;
      dl[i] = u - knots [interval - i] ;
      n[0][i+1] = 0.0;

      /*run inerior loop*/
      for (j=0;j<=i;j++)
            {
              denom = dr[j] + dl[i-j];

              if (denom ==0.0)                    //check for division by zero
                temp = 0.0;
              else
                temp = n[j][i]/denom;
              n[j][i+1] = n[j][i+1] + dr[j]*temp;
              n[j+1][i+1] = dl[i-j]*temp;
            }
    }

  for (i=0;i<=3;i++)
    N[i] = n[i][3];
}

/* function definition: dbasis */

void dbasis(float u, int interval, float knots[], float dN[])
{
  int i,j;
  double dr[3],dl[3],dn[4][4],temp, denom;
  dn[0][0] = 1.0; /* initialize value */

  /* run loop for cubic B-spline*/
```

```
 for (i=0;i<=1;i++)
  {
    /* calculate intermediate values*/
    dr[i] = knots[interval+i+1] - u;
    dl[i] = u - knots[interval - i];
    dn[0][i+1] = 0.0;

    /* run interior loop*/
    for (j=0;j<=i;j++)
          {
            denom = dr[j]+ dl[i-j];
            if (denom == 0.0)                    /* check division by zero*/
              temp = 0.0;
            else
              temp = dn [j][i]/denom;
            dn[j][i+1] = dn[j][i+1] + dr[j]*temp;
            dn[j+1][i+1] = dl[i-j]* temp;
          }
  }
 i=2;
 dr[i]= knots[interval+i+1] - u;
 dl[i]= u - knots[interval-i];
 dn[0][i+1]= 0.0;
 for (j=0;j<=i;j++)
  {
    denom = dr[j] + dl[i-j];
    if (denom==0.0)
          temp = 0.0;
    else
          temp = (i+1)*dn[j][i]/denom;
    dn[j][i+1] = dn[j][i+1] - temp;    /* compute derivatives*/
    dn[j+1][i+1] = temp;
  }

 for (i=0;i<=3;i++)
   dN[i] = dn[i][3];
}
```

```
/*****************************************************************
Function name:  constraints.C
Author:  Roberto Rojas*/
*****************************************************************/
#include <math.h>
extern void basis(float u, int interval, float knots[], float N[]);
extern void dbasis(float u, int interval, float knots[], float dN[]);
extern void map_point(int num,float Nu[],float Nv[],float *ctrlpt,int intU,int intV,  float *coordinate);
void constraintu(int num, int trimnum, float *u_knots, float *v_knots, float *control_x, float *control_y, float
                    *control_z, float *data, float *derivative);
void constraintv(int num, int trimnum, float *u_knots, float *v_knots, float *control_x, float  *control_y, float
                    *control_z,  float *data, float *derivative);
```

```
void constraintu(int num, int trimnum, float *u_knots, float *v_knots, float *control_x, float *control_y, float
                        *control_z, float *data, float *derivative)
{
int j;
int intU,intV;
float u,v;
float x,y,z;
float dNu[4],Nv[4];
for(j=0;j<trimnum;j++)
{
  u = data[2*j];
  v = data[2*j+1];
  intU = floor(3+u);
  intV = floor(3+v);
  dbasis(u,intU,u_knots,dNu);
  basis(v,intV,v_knots,Nv);
  map_point(num+2,dNu,Nv,control_x,intU,intV,&x);
  map_point(num+2,dNu,Nv,control_y,intU,intV,&y);
  map_point(num+2,dNu,Nv,control_z,intU,intV,&z);
  derivative[3*j+0] = x;
  derivative[3*j+1] = y;
  derivative[3*j+2] = z;
}
}


/*****************************************************************************************/
void  constraintv(int num, int trimnum, float *u_knots, float *v_knots, float *control_x, float *control_y, float
*control_z, float *data, float *derivative)
{
  int j;
  int intU,intV;
  float u,v;
  float x,y,z;
  float Nu[4],dNv[4];
  for(j=0;j<trimnum;j++)
    {
      u = data[2*j];
      v = data[2*j+1];
      intU= floor(3+u);
      intV = floor(3+v);
      basis(u,intU,u_knots,Nu);
      dbasis(v,intV,v_knots,dNv);
      map_point(num+2,Nu,dNv,control_x,intU,intV,&x);
      map_point(num+2,Nu,dNv,control_y,intU,intV,&y);
      map_point(num+2,Nu,dNv,control_z,intU,intV,&z);
      derivative[3*j+0] = x;
      derivative[3*j+1] = y;
      derivative[3*j+2] = z;
    }
}
```

```
/*********************************************************************
* Function name: create_surf..C
* Author:  Xijun Wang
*********************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>                          /* To accommodate "strlen"  */
//-------------------------------------------------------------------------------------------
// Calculate the control vertices and knot sequence for model surfaces.
//-------------------------------------------------------------------------------------------
      void create_surf  (int numpts[], float datapts[], float u_newknots[], float v_newknots[],float  contrlpts[], int
      newnum[]);

void create_surf_w(float offset, float uw_newknots[], float vw_newknots[],float wcontrlpts[], int  wnewnum[],
                        float utw_newknots[], float vtw_newknots[],float tw_contrlpts[], int tw_newnum[], int
                        numpts_wing[], float trim_points_wing[]);
extern void calc_offset_w(float offset, float trim_data[], float trim_offset[], int trimnum);
extern void Create_NUBS3_Surface(int numpts[], float datpts[], int param, int u_continuities[],
                                  int v_continuities[], float u_constraints[], float v_constraints[], float u_newknots[],
                                  float v_newknots[], float contrlpts[], int newnum[]);
extern void point(int num, int trimnum, float u_newknots[], float v_newknots[],float control_x[], float
                     control_y[], float control_z[],  float trim_data[],  float trim_points[]);
extern void constraintu(int, int, float [], float [], float [], float [], float [],float [], float []);
extern void constraintv(int, int, float [], float [], float [], float [], float [],float [], float []);
void create_surf (int numpts_c[], float data_points_c[], float u_newknots_c[], float v_newknots_c[], float
                     contrlpts_c[], int newnum_c[])

{
  float   *u_constraints, *v_constraints;
  int      i, j, k;
  int      *u_continuities, *v_continuities;
  int       param = 2;
  u_continuities = new int[numpts_c[0]];
  v_continuities = new int[numpts_c[1]];
  for (i=0; i<numpts_c[0]; i++) u_continuities[i]=2;
  for (i=0; i<numpts_c[1]; i++) v_continuities[i]=2;
  u_constraints = new float[numpts_c[0]];
  for (i=0; i<numpts_c[0]; i++) u_constraints[i] = 0;
  v_constraints = new float[numpts_c[1]];
  for (i=0; i<numpts_c[1]; i++) v_constraints[i] = 0;

  Create_NUBS3_Surface(numpts_c,   data_points_c,   param,   u_continuities,   v_continuities,      u_constraints,

                                  v_constraints, u_newknots_c, v_newknots_c, contrlpts_c, newnum_c);


  delete []u_continuities;
  u_continuities = NULL;
  delete []v_continuities;
  v_continuities = NULL;
  delete []u_constraints;
  u_constraints = NULL;
  delete []v_constraints;
  v_constraints = NULL;
}
```

```
void create_surf_w(float offset, float uw_newknots[], float vw_newknots[],float wcontrlpts[], int  wnewnum[],
                   float utw_newknots[], float vtw_newknots[],float tw_contrlpts[], int tw_newnum[],
                    int numpts_wing[], float trim_points_wing[])
{
  FILE *fw, *fw1, *fw2, *fw3, *fw4, *fw5;

  float   *uw_constraints, *vw_constraints, *data_points, *trim_data,*trim_dataw, *trim_offset;
  float   *control_x, *control_y, *control_z, *trim_points;
  int      i,j,k;
  int     *uw_continuities, *vw_continuities;
  int     *vtw_continuities, *utw_continuities;
  float   *utw_constraints, *vtw_constraints;
  int      numpts[2];
  int      trimnum;
  int      param = 2;

  fw  = fopen("wing_upper", "r");                  // read file
  fscanf(fw, "%d", &numpts[0]);
  fscanf(fw, "%d", &numpts[1]);

  data_points = new float[3*numpts[0]*numpts[1]];
  for (i=0; i< numpts[0]*numpts[1]*3; i++)
    fscanf(fw, "%f", &data_points[i]);
  fclose(fw);

  uw_continuities = new int[numpts[0]];
  vw_continuities = new int[numpts[1]];

  for (i=0; i<numpts[0]; i++) uw_continuities[i]=2;
  for (i=0; i<numpts[1]; i++) vw_continuities[i]=2;

  uw_constraints = new float[numpts[0]];
  vw_constraints = new float[numpts[1]];
  for (i=0; i<numpts[0]; i++) uw_constraints[i]=0;
  for (i=0; i<numpts[1]; i++) vw_constraints[i]=0;

  Create_NUBS3_Surface(numpts, data_points, param, uw_continuities,  vw_continuities,uw_constraints,
                                  vw_constraints, uw_newknots, vw_newknots, wcontrlpts, wnewnum);


  fw1 = fopen("w_upper_trimdata", "r");    // read data file of the trimming points for upper wing
  fscanf(fw1,"%d", &trimnum);
  trim_dataw   = new float[2*trimnum];

  for (i=0; i<2*trimnum; i++)
    fscanf (fw1,"%f",&trim_dataw[i]);
  fclose(fw1);

  trim_offset = new float[2*(trimnum+1)];
  calc_offset_w(offset, trim_dataw, trim_offset, trimnum);
  trimnum = trimnum +1;
  trim_data = new float[2*trimnum];
  for (i=0; i<2*trimnum; i++)
```

```
trim_data[i] = trim_offset[i];

control_x=new float[2*(numpts[0]+2)*(trimnum+2)];
control_y=new float[2*(numpts[0]+2)*(trimnum+2)];
control_z=new float[2*(numpts[0]+2)*(trimnum+2)];
for (i=0; i < (numpts[0]+2)*(numpts[1]+2); i++)
  {
    control_x[i] = wcontrlpts[i*3+0];
    control_y[i] = wcontrlpts[i*3+1];
    control_z[i] = wcontrlpts[i*3+2];
  }

point(numpts[0], trimnum, uw_newknots, vw_newknots, control_x, control_y, control_z, trim_data,
       trim_points_wing);

fw2 = fopen("1_w_trim_points", "w");
for (i=0; i<numpts[0]*trimnum; i++)
fprintf(fw2, "%10f %10f %10f\n", trim_points_wing[3*i], trim_points_wing[3*i+1],
          trim_points_wing[3*i+2]);
fclose(fw2);

numpts_wing[0] = numpts[0];
numpts_wing[1] = trimnum;
vtw_continuities = new int[trimnum];
for (i=0; i<trimnum; i++)  vtw_continuities[i] = 2;
utw_continuities = new int[numpts[0]];
for (i=0; i<numpts[0]; i++)  utw_continuities[i] = 2;

utw_constraints = new float[trimnum*numpts[0]*4];
constraintu(numpts[0], trimnum, uw_newknots, vw_newknots, control_x, control_y, control_z,  trim_data,
                utw_constraints);

vtw_constraints = new float[trimnum*numpts[0]*4];
constraintv(numpts[0], trimnum, uw_newknots, vw_newknots, control_x, control_y, control_z, trim_data,
                vtw_constraints);

numpts[1]=trimnum;

Create_NUBS3_Surface(numpts, trim_points_wing, param, utw_continuities,  vtw_continuities,
                                utw_constraints, vtw_constraints, utw_newknots, vtw_newknots, tw_contrlpts,
                                tw_newnum);

fw3 = fopen("1_w_trim_uknots", "w");
fprintf(fw3, "%10d\n", tw_newnum[0]-6);
for (i=0; i<tw_newnum[0]-6; i++)
 fprintf(fw3, "%10.2f\n", utw_newknots[3+i]);
fclose(fw3);

delete [] control_x;
control_x = NULL;
delete [] control_y;
control_y = NULL;
delete [] control_z;
control_z = NULL;
delete [] trim_data;
trim_data = NULL;
```

```
 delete [] data_points;
 data_points = NULL;
 delete []uw_continuities;
 uw_continuities = NULL;
 delete []vw_continuities;
 vw_continuities = NULL;
 delete []uw_constraints;
 uw_constraints = NULL;
 delete []vw_constraints;
 vw_constraints = NULL;
 delete []utw_continuities;
 utw_continuities = NULL;
 delete []vtw_continuities;
 vtw_continuities = NULL;
 delete []utw_constraints;
 utw_constraints = NULL;
 delete []vtw_constraints;
 vtw_constraints = NULL;
 }



/*********************************************************************
* Function Name:  curv_knot_constraints.C
* Author:  Steven fleming
*********************************************************************/
// Contains routines to add continuity constraints to the knot vector
#include<math.h>
#define OPEN 0
#define CLOSED 1
int curv_new_knot_num(int numpts, int *continuities);
void curv_add_knot_const(int numpts,float knots[],int opcl,int continuities[],int newnum,float newknots[]);

// Function: new_knot_num by steven Fleming
// Calculates number of new knots in the knot sequence based on the continuity constraints
int curv_new_knot_num (int numpts, int *continuities)
{
 int newnum,i,j,continuity;
 j=4;
 for(i=1;i<numpts-1;i++)
 {
  continuity = continuities[i];
  switch(continuity)
  {
    case(0):
    j+=3;
    break;
    case(1):
    j+=2;
    break;
    default:
    j+=1;
  }
 }
 j+=4;
 newnum = j;
```

```
 return(newnum);
}

/* Function: add_knot_const
   Impose continuity constraints on the knot sequence
 output:
 newknots[]: new knot sequence;
 input:
 numpts: no. of data points;
 knots[]: array of original knots sequence;
 continuities[]: array of continuities at domain knots;
 opcl: flag ( open = 0; closed = 1 );  */

void curv_add_knot_const(int numpts,float knots[],int opcl,int continuities[],int newnum,
                                    float newknots[])
{
 int continuity,i,j,k,count;
 continuity = continuities[0];
 switch(continuity)
 {
  case(0):
  newknots[0] = knots[2];
  for(i=1;i<3;i++)
     newknots[i] = knots[3];
  break;
  case(1):
  newknots[0] = knots[1];
  newknots[1] = knots[2];
  for(i=2;i<=3;i++)
     newknots[i] = knots[3];
  break;
  default:
  newknots[0] = knots[0];
  for(i=1;i<=3;i++) newknots[i]=knots[i];
 }

 k=4;                                          // calculate the values after newknots [4]
 if(opcl==CLOSED)
 count = numpts-1;
 else
 count = numpts;
 for(i=1;i<count;i++)
 {
  continuity = continuities[i];
  switch(continuity)
  {
    case(0):
    for(j=1;j<=3;j++)
    {
      newknots[k] = knots[3+i];
      k++;
    }
    break;
    case(1):
    for(j=1;j<=2;j++)
    {
```

```
      newknots[k] = knots[3+i];
      k++;
    }
    break;
    default:
    newknots[k] = knots[3+i];
    k++;
  }
 }

 if(opcl==OPEN)
 {
  continuity = continuities[i-1];
  switch(continuity)
  {
    case(0):
    newknots[k]= knots[5+i];
    break;
    case(1):
    for(j=0;j<=1;j++)
    {
      newknots[k] = knots[4+j+i];
      k++;
    }
    break;
    default:
    for(j=0;j<=2;j++)
    {
      newknots[k] = knots[3+j+i];
      k++;
    }
   }
 }
 else
 {
  newknots[k] = knots[3+i];
  k++;
  for(j=3;j>=1;j--)
  {
    newknots[k] = newknots[k-1] + newknots[j] - newknots[j-1];
    k++;
  }
 }
}



/*********************************************************************
* Function name: display.C
* Author:  Xijun Wang
*********************************************************************/
//  Routine for displaying the Nurbs surface given the control point array, the
//  knot vector and the order of the polynomial basis

#include <stdio.h>
#include <stdlib.h>                    /* To accommodate "exit" */
```

```
#include <string.h>                    /* To accommodate "strlen" */
#include <iostream.h>
#include <Xm/Form.h>                   /* Motif Form widget */
#include <Xm/Frame.h>                  /* Motif Frame widget */
#include <X11/keysym.h>
#include <X11/Xutil.h>
#include <X11/Xatom.h>                 /* For XA_RGB_DEFAULT_MAP */
#include <X11/Xmu/StdCmap.h> /* For XmuLookupStandardColormap */
#include <X11/Xlib.h>                  /* To access fonts */
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glx.h>
#include <X11/GLw/GLwMDrawA.h>  /* Motif OpenGL drawing area */
//--------------------------------------------------------------
// subroutines defined in this file (in order)
//--------------------------------------------------------------
void display_g(float ut_newknots[], float vt_newknots[], float t_contrlpts[],
                int t_newnum[],Display *dpy, Window win, GLXContext glxcontext);
void display_r(float ut_newknots[], float vt_newknots[], float t_contrlpts[],
                int t_newnum[],Display *dpy, Window win, GLXContext glxcontext);
static void MakeRasterFont (Display *dpy, Window win, GLXContext glxcontext);
static void PrintString (char *);

//--------------------------------------------------------------
//  Global variables local to this file
//--------------------------------------------------------------
int showPoints_g = 0;
int showPoints_r = 0;
static GLuint FontOffset;
GLuint nub;
//=====================================================================
// This routine uses the Nurbs interface provided by GLU for rendering a Nubs surface
// Parameters
//   knots      -- knot sequence
//   ctlpoints  -- control point array
//   dpy        -- XtDisplay (glxarea)
//   win        -- XtWindow  (glxarea)
//   glxcontext -- OpenGL rendering context
// Returns
//   nothing
//=====================================================================

void display_g(float ut_newknots[], float vt_newknots[], float t_contrlpts[], int t_newnum[], Display *dpy,
                    Window win, GLXContext glxcontext)
{
  GLint num_ctrl[2], count, ustride, vstride;
  int i, j, k;
  num_ctrl[0] = t_newnum[0] - 4;
  num_ctrl[1] = t_newnum[1] - 4;
  int col = t_newnum[0] - 4;
  int row = t_newnum[1] - 4;

  float ***t_ctlpoints;
   t_ctlpoints = new float **[row];
    for( i = 0; i< row; i++)
     {
```

```
   t_ctlpoints[i] = new float *[col];
   for(j = 0; j<col; j++)
     t_ctlpoints[i][j] = new float[3];
   }

char *A[320] =     { "1",  "2",  "3",  "4",  "5",  "6",  "7",  "8",  "9", "10",   //pointer array containing string
                     "11", "12", "13", "14", "15", "16", "17", "18", "19", "20",    // used to display control points
                     "21", "22", "23", "24", "25", "26", "27", "28", "29", "30",
                    "31", "32", "33", "34", "35", "36", "37", "38", "39", "40",
                    "41", "42", "43", "44", "45", "46", "47", "48", "49", "50",
                     "51", "52", "53", "54", "55", "56", "57", "58", "59", "60",
                     "61", "62", "63", "64", "65", "66", "67", "68", "69", "70",
                     "71", "72", "73", "74", "75", "76", "77", "78", "79", "80",
                     "81", "82", "83", "84", "85", "86", "87", "88", "89", "90",
                     "91", "92", "93", "94", "95", "96", "97", "98", "99","100",
                    "101","102","103","104","105","106","107","108","109","110",
                    "111","112","113","114","115","116","117","118","119","120",
                    "121","122","123","124","125","126","127","128","129","130",
                    "131","132","133","134","135","136","137","138","139","140",
                    "141","142","143","144","145","146","147","148","149","150",
                    "151","152","153","154","155","156","157","158","159","160",
                    "161","162","163","164","165","166","167","168","169","170",
                    "171","172","173","174","175","176","177","178","179","180",
                    "181","182","183","184","185","186","187","188","189","190",
                    "191","192","193","194","195","196","197","198","199","200",
                    "201","202","203","204","205","206","207","208","209","210",
                    "211","212","213","214","215","216","217","218","219","220",
                    "221","222","223","224","225","226","227","228","229","230",
                    "231","232","233","234","235","236","237","238","239","240",
                    "241","242","243","244","245","246","247","248","249","250",
                    "251","252","253","254","255","256","257","258","259","260",
                    "261","262","263","264","265","266","267","268","269","270",
                    "271","272","273","274","275","276","277","278","279","280",
                    "281","282","283","284","285","286","287","288","289","290",
                    "291","292","293","294","295","296","297","298","299","300",
                    "301","302","303","304","305","306","307","308","309","310",
                    "311","312","313","314","315","316","317","318","319","320" };


ustride = num_ctrl[0]*3;
vstride = 3;

/* get the control points in the x, y, z format 3d array*/

count = 0;
for (i=0; i<num_ctrl[1]; i++)
  {
    for (j=0; j<num_ctrl[0]; j++)
          {
            for (k=0; k<3; k++)
              {
                t_ctlpoints[i][j][k] = t_contrlpts[count];
                count++;
              }
          }
  }
```

```
glDisable(GL_LIGHTING);                    // disable lighting
glColor3f(0.0,0.0,0.0);
GLfloat mat_ambient[]    = { 0.1, 0.1, 0.1, 1.0 };
GLfloat mat_shininess[]  = { 30.0 };
GLfloat light_position[] = {0.5, 5.5, 2.5, 1.0};
glEnable (GL_LIGHTING);
glEnable (GL_LIGHT0);
glLightfv (GL_LIGHT0, GL_AMBIENT, mat_ambient);
glLightfv (GL_LIGHT0, GL_POSITION,light_position);
glMaterialfv (GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);

/* set up the Nurb context */
GLUnurbsObj *thetrimNurb;
thetrimNurb = gluNewNurbsRenderer();
gluNurbsProperty(thetrimNurb, GLU_SAMPLING_TOLERANCE, 25.0);
gluNurbsProperty(thetrimNurb, GLU_DISPLAY_MODE, GLU_OUTLINE_PATCH);
gluNurbsProperty(thetrimNurb, GLU_CULLING, GL_TRUE);

glNewList (nub, GL_COMPILE);
gluBeginSurface(thetrimNurb);
gluNurbsSurface(thetrimNurb, t_newnum[1], vt_newknots, t_newnum[0], ut_newknots,
            ustride, vstride, t_contrlpts, 4, 4, GL_MAP2_TEXTURE_COORD_2);
gluNurbsSurface(thetrimNurb, t_newnum[1], vt_newknots, t_newnum[0], ut_newknots,
            ustride, vstride, t_contrlpts, 4, 4, GL_MAP2_NORMAL);
gluNurbsSurface(thetrimNurb, t_newnum[1], vt_newknots, t_newnum[0], ut_newknots,
            ustride, vstride, t_contrlpts, 4, 4, GL_MAP2_VERTEX_3);
gluEndSurface(thetrimNurb);

/* visualize the control point net */
if (showPoints_g)
  {
    glPointSize(4.0);                              // set point size to 4

    glColor3f(1.0, 1.0, 0.0);                      // set color to yellow
    glBegin(GL_POINTS);
    for (i = 0; i < num_ctrl[1]; i++)
          {
           for (j = 0; j < num_ctrl[0]; j++)
             {
              glVertex3f(t_ctlpoints[i][j][0],
                         t_ctlpoints[i][j][1], t_ctlpoints[i][j][2]);
             }
          }
    glEnd();

    count = 0;
    glColor3f (1.0, 1.0, 1.0);                     // set the text color to WHITE
    MakeRasterFont (dpy, win, glxcontext);         // load up the font (see above)
    for (i = 0; i < num_ctrl[1]; i++)
          {
           for (j = 0; j < num_ctrl[0]; j++)
             {
              glRasterPos3f (t_ctlpoints[i][j][0], t_ctlpoints[i][j][1],  t_ctlpoints[i][j][2]);    // position the bitmaps
              PrintString(A[count]);               // print the text string
              count++;
             }
```

```
            }
   }        // end-of-if
  glEnable (GL_LIGHTING);
  glEndList();

  for( i = 0; i< row; i++)
    {
      for(j = 0; j<col; j++)
      delete [] t_ctlpoints[i][j];                          // deallocate the memory
    delete [] t_ctlpoints[i];
    }
  delete [] t_ctlpoints;

}    // end-of-display_g

void display_r ( float ut_newknots[], float vt_newknots[], float t_contrlpts[],  int t_newnum[], Display *dpy,
                    Window win, GLXContext glxcontext)
{
  GLint num_ctrl[2], count, ustride, vstride;
  int i, j, k;
  num_ctrl[0] = t_newnum[0] - 4;
  num_ctrl[1] = t_newnum[1] - 4;
  int col = t_newnum[0] - 4;
  int row = t_newnum[1] - 4;

  float ***t_ctlpoints;
   t_ctlpoints = new float **[row];
    for( i = 0; i< row; i++)
     {
      t_ctlpoints[i] = new float *[col];
      for(j = 0; j<col; j++)
        t_ctlpoints[i][j] = new float[3];
     }

  char *A[320] =     { "1",  "2",  "3",  "4",  "5",  "6",  "7",  "8",  "9", "10",
                     "11", "12", "13", "14", "15", "16", "17", "18", "19", "20",
                     "21", "22", "23", "24", "25", "26", "27", "28", "29", "30",
                      “31", "32", "33", "34", "35", "36", "37", "38", "39", "40",
                      "41", "42", "43", "44", "45", "46", "47", "48", "49", "50",
                     "51", "52", "53", "54", "55", "56", "57", "58", "59", "60",
                     "61", "62", "63", "64", "65", "66", "67", "68", "69", "70",
                     "71", "72", "73", "74", "75", "76", "77", "78", "79", "80",
                     "81", "82", "83", "84", "85", "86", "87", "88", "89", "90",
                     "91", "92", "93", "94", "95", "96", "97", "98", "99","100",
                    "101","102","103","104","105","106","107","108","109","110",
                    "111","112","113","114","115","116","117","118","119","120",
                    "121","122","123","124","125","126","127","128","129","130",
                    "131","132","133","134","135","136","137","138","139","140",
                    "141","142","143","144","145","146","147","148","149","150",
                    "151","152","153","154","155","156","157","158","159","160",
                    "161","162","163","164","165","166","167","168","169","170",
                    "171","172","173","174","175","176","177","178","179","180",
                    "181","182","183","184","185","186","187","188","189","190",
                    "191","192","193","194","195","196","197","198","199","200",
                    "201","202","203","204","205","206","207","208","209","210",
                    "211","212","213","214","215","216","217","218","219","220",
```

```
                "221","222","223","224","225","226","227","228","229","230",
                "231","232","233","234","235","236","237","238","239","240",
                "241","242","243","244","245","246","247","248","249","250",
                "251","252","253","254","255","256","257","258","259","260",
                "261","262","263","264","265","266","267","268","269","270",
                "271","272","273","274","275","276","277","278","279","280",
                "281","282","283","284","285","286","287","288","289","290",
                "291","292","293","294","295","296","297","298","299","300",
                "301","302","303","304","305","306","307","308","309","310",
                "311","312","313","314","315","316","317","318","319","320" };

  ustride = num_ctrl[0]*3;
  vstride = 3;
  /* get the control points in the x, y, z format 3d array*/
  count = 0;
  for (i=0; i<num_ctrl[1]; i++)
    {
      for (j=0; j<num_ctrl[0]; j++)
            {
              for (k=0; k<3; k++)
                {
                  t_ctlpoints[i][j][k] = t_contrlpts[count];
                  count++;
                }
            }
    }
//-------------------------------------------------------------------------------------------------
  glDisable(GL_LIGHTING);                       // disable lighting
  glColor3f(1.0,0.5,0.0);                       // orange color
  GLfloat mat_ambient[]    = { 0.1, 0.1, 0.1, 1.0 };
  GLfloat mat_shininess[]  = { 50.0 };
  GLfloat light_position[] = {0.5, 5.5, 2.5, 1.0};
  glEnable (GL_LIGHTING);
  glEnable (GL_LIGHT0);
  glLightfv (GL_LIGHT0, GL_AMBIENT, mat_ambient);
  glLightfv (GL_LIGHT0, GL_POSITION,light_position);
  glMaterialfv (GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);

  /* set up the Nurb context */
  GLUnurbsObj *thetrimNurb;
  thetrimNurb = gluNewNurbsRenderer();
  gluNurbsProperty(thetrimNurb, GLU_SAMPLING_TOLERANCE, 25.0);
  gluNurbsProperty(thetrimNurb, GLU_DISPLAY_MODE, GLU_FILL);
  gluNurbsProperty(thetrimNurb, GLU_CULLING, GL_TRUE);

  glNewList (nub, GL_COMPILE);
  gluBeginSurface(thetrimNurb);
  gluNurbsSurface(thetrimNurb, t_newnum[1], vt_newknots, t_newnum[0], ut_newknots,
              ustride, vstride, t_contrlpts, 4, 4, GL_MAP2_TEXTURE_COORD_2);
  gluNurbsSurface(thetrimNurb, t_newnum[1], vt_newknots, t_newnum[0], ut_newknots,
              ustride, vstride, t_contrlpts, 4, 4, GL_MAP2_NORMAL);
  gluNurbsSurface(thetrimNurb, t_newnum[1], vt_newknots, t_newnum[0], ut_newknots,
              ustride, vstride, t_contrlpts, 4, 4, GL_MAP2_VERTEX_3);
  gluEndSurface(thetrimNurb);

  /* visualize the control point net */
```

```
 if (showPoints_r)
  {
    glPointSize(4.0);                    // set point size to 4

    glColor3f(1.0, 1.0, 0.0);            // set color to yellow
    glBegin(GL_POINTS);
    for (i = 0; i < num_ctrl[1]; i++)
          {
           for (j = 0; j < num_ctrl[0]; j++)
             {
              glVertex3f(t_ctlpoints[i][j][0],
                         t_ctlpoints[i][j][1], t_ctlpoints[i][j][2]);
             }
          }
    glEnd();

    count = 0;
    glColor3f (1.0, 1.0, 1.0);                    // set the text color to WHITE
    MakeRasterFont (dpy, win, glxcontext);        // load up the font (see above)
    for (i = 0; i < num_ctrl[1]; i++)
          {
           for (j = 0; j < num_ctrl[0]; j++)
             {
              glRasterPos3f (t_ctlpoints[i][j][0], t_ctlpoints[i][j][1], t_ctlpoints[i][j][2]);     // position the bitmaps
              PrintString(A[count]);              // print the text string
              count++;
             }
          }
  } // end-of-if
 glEndList();

 for( i = 0; i< row; i++)
  {
   for(j = 0; j<col; j++)
     delete [] t_ctlpoints[i][j]; // delete 3 of ....
   delete [] t_ctlpoints[i];
  }
 delete [] t_ctlpoints;

} // end-of-display

//=====================================================================
// MakeRasterFont --
// Sets up a font (helvetica) to be used to write text in the openGL drawing area
// Parameters
//   dpy       -- XtDisplay (glxarea)
//   win       -- XtWindow  (glxarea)
//   glxcontext -- OpenGL rendering context
// Returns
//   nothing
//=====================================================================
static void MakeRasterFont (Display *dpy, Window win, GLXContext glxcontext)
{
 XFontStruct *fontInfo;
 Font id;
 unsigned int first, last;
```

```
  fontInfo = XLoadQueryFont (dpy, "-adobe-helvetica-medium-r-normal--17-120-100-100-p-88-iso8859-1");
  if (fontInfo == NULL) { printf ("no font found\n"); exit (0); }
  id         = fontInfo->fid;
  first      = fontInfo->min_char_or_byte2;
  last       = fontInfo->max_char_or_byte2;
  FontOffset = glGenLists ((GLuint) last+1);
  if (FontOffset == 0) { printf ("out of display lists\n"); exit (0); }
  glXMakeCurrent (dpy, win, glxcontext);
  glXUseXFont (id, first, last-first+1, FontOffset+first);
}  // end-of-MakeRasterFont

static void PrintString (char *s)
{
  glPushAttrib (GL_LIST_BIT);
  glListBase (FontOffset);
  glCallLists (strlen(s), GL_UNSIGNED_BYTE, (GLubyte *) s);
  glPopAttrib ();
}



/*********************************************************************
Function Name:  knot_constraints.C
Author:   Steven fleming
*********************************************************************/
//Contains routines to add continuity constraints to the knot vector.

#include<math.h>
#define OPEN 0
#define CLOSED 1

int new_knot_num(int numpts, int *continuities);
void add_knot_const(int numpts,float knots[],int opcl,int continuities[], int newnum,float newknots[]);

/* Function: new_knot_num by steven Fleming
Calculates number of new knots */

int new_knot_num(int numpts, int *continuities)
{
  int newnum,i,j,continuity;
  j=4;
  for(i=1;i<numpts-1;i++)
  {
    continuity = continuities[i];
    switch(continuity)
    {
      case(0):
      j+=3;
      break;
      case(1):
      j+=2;
      break;
      default:
      j+=1;
    }
```

```
  }
  j+=4;
  newnum = j;
  return(newnum);
}

// Function: add_knot_const
// output: new knot vector
// input: no. of points,array of original knots,array of continuity
// constraint types,open/closed flag,new number of knots

void add_knot_const ( int numpts,float knots[],int opcl,int continuities[], int newnum,float newknots[] )
{
  int  continuity,i,j,k,count;
  continuity = continuities[0];
  switch(continuity)
  {
    case(0):
    newknots[0] = knots[2];
    for(i=1;i<3;i++)
       newknots[i] = knots[3];
    break;
    case(1):
    newknots[0] = knots[1];
    newknots[1] = knots[2];
    for(i=2;i<=3;i++)
       newknots[i] = knots[3];
    break;
    default:
    newknots[0] = knots[0];
    for(i=1;i<=3;i++) newknots[i]=knots[i];
  }

  k=4;
  if(opcl==CLOSED)
  count = numpts-1;
  else
  count = numpts;
  for(i=1;i<count;i++)
  {
    continuity = continuities[i];
    switch(continuity)
    {
      case(0):
      for(j=1;j<=3;j++)
      {
        newknots[k] = knots[3+i];
        k++;
      }
      break;
      case(1):
      for(j=1;j<=2;j++)
      {
        newknots[k] = knots[3+i];
        k++;
      }
```

```
    break;
    default:
    newknots[k] = knots[3+i];
    k++;
  }
 }

 if(opcl==OPEN)
 {
  continuity = continuities[i-1];
  switch(continuity)
  {
    case(0):
    newknots[k]= knots[5+i];
    break;
    case(1):
    for(j=0;j<=1;j++)
    {
      newknots[k] = knots[4+j+i];
      k++;
    }
    break;
    default:
    for(j=0;j<=2;j++)
    {
      newknots[k] = knots[3+j+i];
      k++;
    }
   }
 }
 else
 {
  newknots[k] = knots[3+i];
  k++;
  for(j=3;j>=1;j--)
  {
    newknots[k] = newknots[k-1] + newknots[j] - newknots[j-1];
    k++;
  }
 }
}



/*********************************************************************
Function Name: map_point.C
Author:  Robert Rojas
*********************************************************************/
// Computes a single point on a NUBS.One coordinate at a time.
// Input: No. of points in u- direction:numpts[0]
// B-spline basis functions: Nu & Nv;
// Control points in one coordinate:  contrl_x(or y or z)
// Interval on the parametric space:  IntU & Int V
// Output: the value of the coordinate x(or y or z) specified by the parametric mapping using variables u & v
/* Function Prototype */
void map_point(int num, float Nu[], float Nv[], float *ctrlpt, int intU, int intV, float *coordinate);
```

```
/* Function Definition */
void map_point(int num,float Nu[],float Nv[],float *ctrlpt,int intU,int intV, float *coordinate)
{
int i,j;
*coordinate=0.0;
for(i=0;i<4;i++)
for (j=0;j<4;j++)
*coordinate =*coordinate+ ctrlpt[num*(i+intV-3)+(j+intU-3)]*Nu[j]*Nv[i];
}


/*********************************************************************
Function name:  model_curve.C
Author:  Steven Flemming
*********************************************************************/
/*driver routine to create a cubic NUBS curve with specified constraints*/

#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <math.h>

#define OPEN 0
#define CLOSED 1

void Create_NUBS3_Curve(int, float[], int, int[], float [], float [], float[], int&);

extern void curv_parametrize(int, float[], int, int, float[]);
extern void curv_add_knot_const(int, float[], int, int[], int, float[]);
extern int curv_new_knot_num(int, int*);
extern void setup_system(int, float [], int, int [], int, float []);
extern void solve_system(int, float [], int, int, int [], float [], float [], float []);

//=========================================================
/*  Input:
    numpts:   Number of data points;
    datpts:   Array of data points;
    param:   Type of parametrization;
    continuities[]:   Array of continuities;
    constraints[]:   Array of tangent vector constraints;

    Output:
    curv_newknots[] : Array of knots;
    curv_contrlpts[]: Array of control points; */
//=========================================================
void Create_NUBS3_Curve(int numpts, float datpts[], int param, int continuities[],float constraints[],
                                float curv_newknots[], float curv_contrlpts[], int &curv_newnum)
{
  float tol = 0.01;
  float *curv_knots, *Nmat;
  int opcl,i,j,curv_numctrl;
  int order = 4;
  FILE *fp;
```

```
 /*determine if open or closed*/
 if ((fabs(datpts[0]-datpts[(numpts-1)*3] ) <=tol )&&          //by Comparing the distance of
                                                               //the first and last points
    (fabs(datpts[1]-datpts[(numpts-1)*3 + 1] ) <=tol) &&
    (fabs(datpts[2]-datpts[(numpts-1)*3 + 2] ) <=tol))
   opcl = CLOSED;
 else
   opcl = OPEN;
 if (opcl == CLOSED)
   curv_knots = new float[numpts + 16];
 else
   curv_knots = new float[numpts + 14];
 curv_parametrize(numpts, datpts, opcl,param,curv_knots);          // calculate knot sequence;
 curv_newnum = curv_new_knot_num(numpts, continuities);       // calculate number of new knots;
 curv_add_knot_const(numpts, curv_knots, opcl, continuities, curv_newnum, curv_newknots);

 // imposes continuity constraints on the knot sequence;
 // output new knot sequence;
 curv_numctrl = curv_newnum - 4;
 Nmat = new float[(curv_numctrl+1)*(curv_numctrl)];
 setup_system(numpts,curv_newknots,opcl,continuities,curv_numctrl,Nmat);

 // set up the basis function matrix;
 solve_system(numpts,datpts,curv_numctrl,opcl,continuities,constraints,Nmat,curv_contrlpts);

 // solve the system and get control points array;
}
```

```
/*****************************************************************
Function name:  model_Surface.C
Author:  Steven fleming
*****************************************************************/
//  Contains routines to run Model Surface Modelling routine to allow the user to easily
//  create a cubic NUBS surface on graPHIGS and proposed PHIGS + 3D NUBS surface function.
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

void  Create_NUBS3_Surface (int numpts[], float datpts[], int param, int u_continuities[],
                            int v_continuities[], float u_constraints[], float v_constraints[],
                            float u_newknots[], float v_newknots[], float contrlpts[], int newnum[]);
     extern  void  average_parametrization(int numpts[], float datpts[],int opcl[], int param,float   u_knots[],float
     v_knots[]);

extern  void  add_knot_const(int numpts,float knots[],int opcl,int continuities[],int newnum, float  newknots[]);
extern  int  new_knot_num(int numpts, int *continuities);
extern  void  setup_system(int numpts, float knots[], int opcl,int continuities[], int numctrl,float  Nmat[]);
extern  void  etup_constraint_matrix(int numpts[],int numctrl[], float datpts[], int opcl[],  int u_continuities[],
                          int v_continuities[], float u_constraints[], float v_constraints[], float
                        constraint_matrix[]);
extern void solve_systems(int numctrl[], float Amat[], float Bmat[], float constraint_matrix[], float  ctrlpts[]);
#define OPEN 0
```

```
#define CLOSED 1
//  This is the driver routine to create a cubic NUBS Surface.
void Create_NUBS3_Surface(int numpts[], float datpts[], int param, int u_continuities[],  int v_continuities[],
                                    float  u_constraints[], float v_constraints[], float u_newknots[], float
                                    v_newknots[], float contrlpts[], int newnum[])
{
  float tol = 0.001;
  float *u_knots, *v_knots;
  int opcl[2],i,j,k;
  int numctrl[2];
  float *Amat,*Bmat;
  float *constraint_matrix;

  /*determine if open or closed*/
  i=0;
  opcl[0] = CLOSED;
  opcl[1] = CLOSED;

  while((opcl[0] == CLOSED)&&(i<numpts[1]))
    {
     if  ((fabs(datpts[3*numpts[0]*i]   - datpts[3*numpts[0]*i+(numpts[0]-1)*3]  ) <= tol) &&
              (fabs(datpts[3*numpts[0]*i+1] - datpts[3*numpts[0]*i+(numpts[0]-1)*3+1]) <= tol) &&
              (fabs(datpts[3*numpts[0]*i+2] - datpts[3*numpts[0]*i+(numpts[0]-1)*3+2]) <=tol))
            {
             opcl[0] = CLOSED;
             i++;
            }
     else
            opcl[0] = OPEN;
    }

  i=0;
  while((opcl[1] == CLOSED)&&(i<numpts[0]))
    {
     if  ((fabs(datpts[3*i]   - datpts[3*numpts[0]*(numpts[1] -1)+3*i]   )<= tol) &&
              (fabs(datpts[3*i+1] - datpts[3*numpts[0]*(numpts[1] -1)+3*i+1] )<= tol) &&
              (fabs(datpts[3*i+2] - datpts[3*numpts[0]*(numpts[1] -1)+3*i+2] )<= tol))
            {
             opcl[1] = CLOSED;
             i++;
            }
     else
            opcl[1] = OPEN;
    }

  v_knots = new float[numpts[1]+6];
  u_knots = new float[numpts[0]+6];
  average_parametrization(numpts,datpts,opcl,param,u_knots,v_knots);
  newnum[0] = new_knot_num(numpts[0],u_continuities);
  newnum[1] = new_knot_num(numpts[1],v_continuities);
  add_knot_const(numpts[0],u_knots,opcl[0],u_continuities,newnum[0],u_newknots);
  add_knot_const(numpts[1],v_knots,opcl[1],v_continuities,newnum[1],v_newknots);
  numctrl[0] = newnum[0] - 4;
  numctrl[1] = newnum[1] - 4;
  Bmat = new float[(numctrl[0]+1)*(numctrl[0])];
  Amat = new float[(numctrl[1]+1)*(numctrl[1])];
```

```
  setup_system(numpts[0],u_newknots,opcl[0],u_continuities,numctrl[0],Bmat);
  setup_system(numpts[1],v_newknots,opcl[1],v_continuities,numctrl[1],Amat);
  constraint_matrix = new float[3*numctrl[0]*numctrl[1]];

  setup_constraint_matrix(numpts,numctrl,datpts,opcl,u_continuities,v_continuities,u_constraints,
                          v_constraints,constraint_matrix);

  solve_systems(numctrl,Amat,Bmat,constraint_matrix,contrlpts);

  delete []u_knots;
  u_knots = NULL;
  delete []v_knots;
  v_knots = NULL;
  delete []Bmat;
  Bmat = NULL;
  delete []Amat;
  Amat = NULL;
  delete []constraint_matrix;
  constraint_matrix = NULL;
}



/*********************************************************************
Function name:  offset.C
Author:  Xijun Wang
*********************************************************************/
// The function is to calculate the trim curve offset in parametric domain
// Input:
//      offset: the distance of trim curve need to be shifted;
//      trim_data[] : Original parametric data of trim curve;
//      trim_offset[]: shifted parametric data of trim curve;

#include <math.h>
#include <iostream.h>
void calc_offset_w(float offset, float trim_data[], float trim_offset[], int trimnum);
void calc_offset(float offset, float trim_data[], float trim_offset[]);
void calc_offset_w(float offset, float trim_data[], float trim_offset[], int trimnum)
{
  float *x, *y,*xm,*ym,*k, *kn, *kf, *km, *xn, *yn;
  int n = trimnum;              // number of points on the arc curve
  x = new float[n];
  y = new float[n];

  for(int i=0; i<trimnum; i++)
  {
    x[i] = trim_data[2*i];
    y[i] = trim_data[2*i+1];
  }

  xm = new float[n-1];
  ym = new float[n-1];
  k  = new float[n-1];
  kn = new float[n-1];
  kf = new float[n-1];
```

```
 km = new float[n-1];
 xn = new float[n-1];
 yn = new float[n-1];

 for( i=0; i<(trimnum-1); i++)
 {
  xm[i] = (x[i]+x[i+1])/2;              // Middle point of two neighbor points on the curve;
  ym[i] = (y[i]+y[i+1])/2;
  k[i] = (y[i+1]-y[i])/(x[i+1]-x[i]);   // Lean rate of two neighbor points on the curve;
  kn[i] = -1.0/k[i];                    // Lean rate of perpendicular line of two neighbor point line
  kf[i] = -1.0/sqrt(kn[i]*kn[i]+1);
  km[i] = offset*kf[i];
  xn[i] = xm[i] + km[i];                // New vector point from the midpoint of two neighbor points
  yn[i] = ym[i] + kn[i]*km[i];          // and shift away "offset", which represents the offset point
 }

 trim_offset[0] = trim_data[0]-offset;
 trim_offset[1] = trim_data[1];

 for(i = 1; i < trimnum ; i++)
  {
    trim_offset[2*i] = xn[i-1];
    trim_offset[2*i+1] = yn[i-1];
  }

 trim_offset[2*trimnum] = trim_data[2*(trimnum-1)]-offset;
 trim_offset[2*trimnum+1] = trim_data[2*(trimnum-1)+1];

  delete [] x;
  delete [] y;
  delete [] xm;
  delete [] ym;
  delete [] xn;
  delete [] yn;
  delete [] k;
  delete [] kn;
  delete [] kf;
  delete [] km;
  x = NULL;
  y = NULL;
  xm = NULL;
  ym = NULL;
  xn = NULL;
  yn = NULL;
  k = NULL;
  kn = NULL;
  kf = NULL;
  km = NULL;
}

void calc_offset(float offset, float trim_data[], float new_trim[])
{
 float *x, *y,*xm,*ym,*k, *kn, *kf, *km, *xn, *yn;

 int n = 14;              // number of points on the arc curve
 x = new float[n];
```

```
y = new float[n];

for(int i=0; i<14; i++)
{
  x[i] = trim_data[2*(i+4)];
  y[i] = trim_data[2*(i+4)+1];
}

xm = new float[n-1];
ym = new float[n-1];
k  = new float[n-1];
kn = new float[n-1];
kf = new float[n-1];
km = new float[n-1];
xn = new float[n-1];
yn = new float[n-1];

for( i=0; i<(n-1); i++)
{
 xm[i] = (x[i]+x[i+1])/2;              // Middle point of two neighbor points on the curve;
 ym[i] = (y[i]+y[i+1])/2;
 k[i] = (y[i+1]-y[i])/(x[i+1]-x[i]);   // Lean rate of two neighbor points on the curve;
 kn[i] = -1.0/k[i];                    // Lean rate of perpendicular line of two neighbor point line
 kf[i] = -1.0/sqrt(kn[i]*kn[i]+1);
 km[i] = offset*kf[i];
 xn[i] = xm[i] + km[i];                // New vector point from the midpoint of two neighbor points
 yn[i] = ym[i] + kn[i]*km[i];          // and shift away "offset", which represents the offset point
}

new_trim[2*4]   = trim_data[2*4];
new_trim[2*4+1] = trim_data[2*4+1]-offset;
float distance = new_trim[2*4+1]-trim_data[1];

 for(i = 0; i< 4; i++)
 {
   new_trim[2*i]   = trim_data[2*i];
   new_trim[2*i+1] = trim_data[1]+i*distance/4.0;
 }

 for(i = 5; i< n+4; i++)
 {
   new_trim[2*i] = xn[i-5];
   new_trim[2*i+1] = yn[i-5];
 }
 new_trim[2*(n+4)] = trim_data[2*(n+4-1)];
 new_trim[2*(n+4)+1] = trim_data[2*(n+4-1)+1]+offset;
 float distance1 = trim_data[2*20+1] - new_trim[2*(n+4)+1];

 for(i = n+5; i< 22; i++)
 {
   new_trim[2*i] = trim_data[2*(i-1)];
   new_trim[2*i+1] = new_trim[2*(n+4)+1]+(i-n-4)*distance1/3.0;
 }

 delete [] x;
 delete [] y;
```

```
  delete [] xm;
  delete [] ym;
  delete [] xn;
  delete [] yn;
  delete [] k;
  delete [] kn;
  delete [] kf;
  delete [] km;
  x = NULL;
  y = NULL;
  xm = NULL;
  ym = NULL;
  xn = NULL;
  yn = NULL;
  k = NULL;
  kn = NULL;
  kf = NULL;
  km = NULL;
}



/*************************************************************
Function name:  parametrize.C
Author: Steven fleming
Description: contains parametrization routines for Cubic  B-spline curves.
*************************************************************/
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
void parametrize(int numpts, float datpts[], int opcl, int param, float knots[]);
float pt_dist(int num1, float first[],int num2, float last[]);
#define OPEN 0
#define CLOSED 1
 /*=========================================================
  * Function: parametrize
  *=========================================================
  * Author: Steven Fleming
  * Description: To parametrize a cubic B-spline curve based on chord length
  *                  Method
  * Inputs: number of points, array of data points, open or closed,
  *           type of parametrization
  * Outputs: Knot vector
  *=========================================================*/
void parametrize(int numpts, float datpts[], int opcl, int param, float knots[])
{
 float ratio;         /*ratio of knot spacing to point dist*/
 float distance;
 float first[3], last[3];
 int i;
 float tol = 0.01;

 /* initialize first knot interval */
 knots[3] = 0.0;
 knots[4] = 1.0;
```

```
/* perform correct type of parametrization */
switch(param)
  {
    /* uniform parametrization */
case(1):
  for ( i = 5; i <= numpts+2; i++)
    knots[i] = knots[i-1] + 1;
  break;

  /*chord length parametrization*/
case(2):
  distance = pt_dist(1,datpts,2,datpts);
  ratio = 1/distance;
  /* calculate domain knots */
  for (i = 5; i <= numpts+2; i++)
    {
          distance = pt_dist(i-3, datpts, i-2, datpts);
          knots[i] = distance * ratio + knots[i-1];
    }
  break;

  /*centripetal parametrization*/
case(3):
  distance = pt_dist(1, datpts, 2, datpts);
  ratio = 1/sqrt(distance);
  /*calculate domain knots*/
  for (i = 5; i <= numpts + 2; i++)
    {
          distance = sqrt(pt_dist(i-3, datpts, i-2, datpts));
          knots[i] = distance*ratio + knots[i-1];
    }
  break;

default:
  for (i=5; i <= numpts+2; i++)
    knots[i] = knots[i-1] + 1;
  }

if (opcl == OPEN)
  {
    distance = knots[3] - knots[4];
    for (i = 3; i >= 1; i--)
          knots[i-1] = knots[i] + distance;
        distance = knots[numpts+2] - knots[numpts+1];
    for (i = 2; i <= 4; i++)
          knots[numpts+i+1] = knots[numpts+i] + distance;
  }
else
  {
    for (i=2; i >= 0; i--)
          knots[i] = knots[i+1] + (knots[numpts+i-1] - knots[numpts+i]);

    for (i=3; i <=5; i++)
          knots[numpts+i] = knots[numpts+i-1] + (knots[i+1] - knots[i]);
  }
```

```
}
 /*============================================================
  * module name: point_dist
  *============================================================
  * Description: calculates the actual distance between two 3-D points
  * Input: point arrays for first and last points, point numbers in array
  * Output: distance between points
  *============================================================*/
float pt_dist(int num1, float first[],int num2, float last[])
{
 float distance;    /* distance between points*/

 distance = sqrt ((last[3*(num2-1)] - first[3*(num1-1)])*
              (last[3*(num2-1)] - first[3*(num1-1)])+
                       (last[3*(num2-1)+1] - first[3*(num1-1)+1])*
              (last[3*(num2-1)+1] - first[3*(num1-1)+1])+
                       (last[3*(num2-1)+2] - first[3*(num1-1)+2])*
              (last[3*(num2-1)+2] - first[3*(num1-1)+2]));

 return (distance);
}



 /*************************************************************
  * Name: Point_and_derivative.C
  * Author: Roberto Rojas
  * Description: Computes new "data" points and derivatives on the trimmed patch
  * input:  num                          : number of u direction points,
  *         trimnum                       : number of points defining the trimming curve which are also  the
                                           number of "data points" in the v direction.
  *         u_ and v_ newknots            : so that the basis can be used to compute N.
  *         control_x and control_y:  To be used with N to map points on the original surface using map_point.
  *         trim_data                     : pointer to the array containing original surface data points
  * output:
  *         trim_points                : pointer to array containing data points pertaining to trimmed surface , to be
                                  used                                  when calling Model_surface.
  * ************************************************************/
#include<math.h>
void point(int num, int trimnum, float u_newknots[], float v_newknots[], float control_x[], float control_y[],
            float control_z[],  float trim_data[], float trim_points[]);
extern void basis(float, int, float*, float[]);
extern void map_point(int, float[], float[], float*, int, int, float*);
/*  function definition    */
void point (int num, int trimnum, float u_newknots[], float v_newknots[], float control_x[], float control_y[],
            float control_z[],  float trim_data[], float trim_points[] )
{
 int i,j,k,l,flag;
 int intU, intV;
 float u,v;
 float x,y,z;
 float Nu[4], Nv[4];

 for (j=0; j<trimnum; j++)
   for (i=0; i<num; i++)
```

```
    {
        u=i*trim_data[2*j]/(num-1);
        v=trim_data[2*j+1];

        flag = 0;
        for (k=0; k<num+2; k++)
           {
            if(u>=u_newknots[k] && u<u_newknots[k+1])
                {
                 intU = k;
                 flag = 1;
                }
            if(flag)
                break;
           }

        flag = 0;
        for (l=0; l<trimnum+2; l++)
          {
           if(v>=v_newknots[l] && v<v_newknots[l+1])
             {
                intV = l;
                flag = 1;
             }
           if(flag)
             break;
          }

        basis(u, intU, u_newknots, Nu);
        basis(v, intV, v_newknots, Nv);
        map_point(num+2, Nu, Nv, control_x, intU, intV, &x);
        map_point(num+2, Nu, Nv, control_y, intU, intV, &y);
        map_point(num+2, Nu, Nv, control_z, intU, intV, &z);

        trim_points[3*num*j + 3*i + 0] = x;
        trim_points[3*num*j + 3*i + 1] = y;
        trim_points[3*num*j + 3*i + 2] = z;
    }
}



/*****************************************************************
 * Function Name: point_and_derivative.C
 * Author: Robert Rojas
 * Date: February 16, 1994
 * Description: Computes new "data" points and derivatives on the trimmed patch
 * input:
 *          num                         : number of u direction points,
 *          trimnum                     : number of points defining the trimming curve which are also the
 number of      *                                   "data points" in the v direction.
 *          u_ and v_ newknots          : so that the basis can be used to compute N.
 *          control_x and control_y: To be used with N to map points on the original surface using map_point.
 *          trim_data                   : pointer to the array containing original surface data points
 * output:
 *          trim_points                 : pointer to array containing data points pertaining to trimmed
```

```
  *                                          surface , to be used when calling Model_surface.
  * ********************************************************************/
#include<math.h>

void point(int num, int trimnum, float u_newknots[], float v_newknots[], float control_x[], float control_y[],
            float control_z[],  float trim_data[], float trim_points[]);
extern void basis(float, int, float*, float[]);
extern void map_point(int, float[], float[], float*, int, int, float*);
/*  function definition    */
void point (int num, int trimnum, float u_newknots[], float v_newknots[],float control_x[], float control_y[],
                        float control_z[],  float trim_data[],  float trim_points[])
{
  int i,j,k,l,flag;
  int intU, intV;
  float u,v;
  float x,y,z;
  float Nu[4], Nv[4];

  for (j=0; j<trimnum; j++)
    for (i=0; i<num; i++)
      {
            u=i*trim_data[2*j]/(num-1);
            v=trim_data[2*j+1];

            flag = 0;
            for (k=0; k<num+2; k++)
               {
                 if(u>=u_newknots[k] && u<u_newknots[k+1])
                        {
                          intU = k;
                          flag = 1;
                        }
                 if(flag)
                        break;
               }

            flag = 0;
            for (l=0; l<trimnum+2; l++)
              {
                if(v>=v_newknots[l] && v<v_newknots[l+1])
                  {
                        intV = l;
                        flag = 1;
                  }
                if(flag)
                  break;
              }

            basis(u, intU, u_newknots, Nu);
            basis(v, intV, v_newknots, Nv);

            map_point(num+2, Nu, Nv, control_x, intU, intV, &x);
            map_point(num+2, Nu, Nv, control_y, intU, intV, &y);
            map_point(num+2, Nu, Nv, control_z, intU, intV, &z);

            trim_points[3*num*j + 3*i + 0] = x;
```

```
            trim_points[3*num*j + 3*i + 1] = y;
            trim_points[3*num*j + 3*i + 2] = z;
        }
    }



/*****************************************************************
 File Name: rendering.C
 Author:  Xijun Wang
*****************************************************************/
//  Routine for computing new blending surface of aircraft fuselage and wing
//  and displaying the Nurbs surface given the control point array, the
//  knot vector and the order of the polynomial basis

#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>                /* To accommodate "strlen"  */
#include <Xm/PushB.h>              // Motif PushButton widget
#include <Xm/Scale.h>              // Motif Scale (slider) widget
#include <Xm/Form.h>               /* Motif Form widget */
#include <Xm/Frame.h>              /*  Motif Frame widget */
#include <X11/keysym.h>
#include <X11/Xutil.h>
#include <X11/Xatom.h>             /*  For XA_RGB_DEFAULT_MAP */
#include <X11/Xmu/StdCmap.h>       /*  For XmuLookupStandardColormap */
#include <X11/Xlib.h>              /*  To access fonts */
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glx.h>
#include <GL/glut.h>
#include <X11/GLw/GLwMDrawA.h>  /*  Motif OpenGL drawing area */
#include <Xm/RowColumn.h>          // For Motif widgets
#include <Xm/MessageB.h>           // For Motif message dialog
#include <Xm/PushBG.h>             // For Motif Push button
#include <Xm/PushB.h>
#include <Xm/SeparatoG.h>          // For Motif sepeartor bar
#include <Xm/CascadeB.h>           // For Motif cascade button
#include <Xm/Xm.h>
#include <math.h>                  // For sin, cos, and M_pi function
#include <Xm/TextF.h>              // For Motif text field widgets
#include <Xm/LabelG.h>             // For Motif widgets
#include <Xm/CascadeB.h>           // Motif Cascade Button Widget
#include <Xm/Label.h>              // For Motif label
#include <Xm/PanedW.h>             // For paned window
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>           // For Motif widget to indicate action area
#include <Xm/FileSB.h>
#include <Xm/ArrowBG.h>
#include <Xm/ArrowB.h>

#ifndef CALLBACK
#define CALLBACK
#endif
```

```
//--------------------------------------------------------------------------
// global variables defined elsewhere
//--------------------------------------------------------------------------
extern Bool        doubleBuffer;
      float       offset = 0.4;            // distance of offset curve away from intersection curve
      int         identifier = 6;

      float    contrlpts_r[25000], u_newknots_r[500], v_newknots_r[500];     // right patch
      float    contrlpts_m[35000], u_newknots_m[500], v_newknots_m[500];     // middle blending patch
      float    contrlpts_l[25000], u_newknots_l[500], v_newknots_l[500];          // left patch
      float    contrlpts_n[25000], u_newknots_n[500], v_newknots_n[500];
      float    contrlpts[25000], u_newknots[500], v_newknots[500];                  // fuselage patch
      float     t_contrlpts[25000], ut_newknots[500], vt_newknots[500];             // trimmed fuselage patch
      float    w_contrlpts[25000], w_u_newknots[500], w_v_newknots[500];       // wing patch
      float    w_t_contrlpts[25000], w_ut_newknots[500], w_vt_newknots[500]; // trimmed wing patch
           int       newnum_r[2],newnum_m[2],newnum_l[2],newnum_n[2];

                     // number of control points,  [0] is u direction, [1] is v direction

      int      newnum[2], t_newnum[2], w_newnum[2],w_t_newnum[2];

     // global variables local to this file

      FILE       *fp, *fp1, *fp2, *fp3, *fp4, *fp5, *fp6, *fp7,*f, *f1, *f2, *f3,*f4;
      int     param = 2;
      int     i, j;
      int        grid_u, grid_v;

      int        *r1, *b1, *g1;
      int        sampling_rate;
      static GLXContext glxcontext;
      static GLuint     FontOffset;
      GLuint     nub;
      GLuint     trimnub;
      GLuint     nubcurve;
      GLuint     base;
      static Widget     area;
      float      phi   = 0.0;
      float      theta = 0.0;
      float      r     = 0.0;
      float      a     = 0.0;
      float      b     = 0.0;
      float      c     = 0.0;
      float      p, q;

//--------------------------------------------------------------------------
// subroutines defined elsewhere
//--------------------------------------------------------------------------
extern void refine_pts( int rownum, int incol, int outcol, float datain[],float dataout[]);
extern void conic ( float, float, float[], float[], float[], float[] );
extern void create_surf_w(float offset, float u_newknots[], float v_newknots[],float contrlpts[], int   newnum[],
                                 float ut_newknots[], float vt_newknots[],float t_contrlpts[], int t_newnum[], int
                                 numpts_wing[], float trim_points_wing[]);

extern void create_surf(int numpts[], float data_points[], float u_newknots[], float v_newknots[],
                              float contrlpts[], int newnum[]);
extern void calc_offset( float offset, float trim_data[], float trim_offset[] );
```

```
extern void Create_NUBS3_Surface(int numpts[], float datpts[], int param, int u_continuities[],
                                 int v_continuities[], float u_constraints[], float v_constraints[],
                                 float u_newknots[], float v_newknots[], float contrlpts[], int newnum[]);

extern void Create_NUBS3_Curve(int, float[], int, int[], float [], float [], float[], int&);
extern void bspline_diff_geo(int, float [], float [], int, float [], float [], float [], float [], float [],
                             float [], float [], int&, int []);
extern void point(int num, int trimnum, float u_newknots[], float v_newknots[],  float control_x[], float
                  control_y[],  float control_z[],  float trim_data[], float trim_points[]);
extern void get_orig_points (int num, int trimnum, int grid_u, int grid_v, float trim_data[],
                             float u_newknots[], float v_newknots[], float control_x[],
                             float control_y[], float control_z[], float orig_point_grid[]);
extern void get_trim_points(int num, int trimnum, int grid_u, int grid_v, float ut_newknots[], float
                      vt_newknots[],   float t_control_x[], float t_control_y[], float t_control_z[], float
                      trim_points_grid[]);
extern void constraintu(int, int, float [], float [],float [], float [], float [],float [], float []);
extern void constraintv(int, int, float [], float [],float [], float [], float [],float [], float []);
extern void display_r(float ut_newknots[], float vt_newknots[], float t_contrlpts[],
                             int t_newnum[], Display *dpy, Window win, GLXContext glxcontext);
extern void display_g(float ut_newknots[], float vt_newknots[], float t_contrlpts[],
                             int t_newnum[], Display *dpy, Window win, GLXContext glxcontext);
extern void initX (int, char**,
                   void(*)(Widget, XtPointer, XtPointer),
                   void(*)(Widget, XtPointer, XtPointer),
                   void(*)(Widget, XtPointer, XtPointer),
                   void(*)(Widget,XtPointer,XtPointer),
                   XtAppContext*,
                   Widget*,
                   Widget*,
                   Widget*);
//--------------------------------------------------------------------------
// subroutines local to this file
//--------------------------------------------------------------------------
int  main               (int, char**);
static void OpenGLinit     (Widget, XtPointer, XtPointer);
static void exposed        (Widget, XtPointer, XtPointer);
static void resized        (Widget, XtPointer, XtPointer);
static void input          (Widget, XtPointer, XtPointer);
static void intervalTimer   (XtPointer, XtIntervalId*);
static Boolean idleTime    (XtPointer);
static void draw           (Display*, Window, int);
static void initCustomized (Widget, Widget);
static void buttonCallback (Widget, XtPointer, XtPointer);
static void sliderCallback (Widget, XtPointer, XtPointer);
static void MakeRasterFont (Display*, Window, GLXContext);
static void isometric      (float, float, float);
static void PrintString (char*);
void   init_surface( );
void   init_new_surface( );

// ------------------------------------------------------------------------------------------
//   This is the main program entry.  Yes, it is short and you will likely
//   keep it this short for all your programs:  Make all your initialization
//   changes in the function "initX" and all OpenGL initialization changes
//   in the function "OpenGLinit" (which will be called once the event
//   processing starts up).
```

```
// Parameters
//   argc       -- number of command line arguments
//   argv[]     -- array of text strings corresponding to the words in on
//                 the command line; e.g., argv[0] == the program name
// Returns
//   0          -- UNIX error code  ( 0 = no error )
// ----------------------------------------------------------------------------------------
int main (int argc, char **argv)
{
  XtAppContext app;
  Widget toplevel, form, glxarea;

  initX (argc, argv,              /* Initialize X and Motif */
             OpenGLinit,
             exposed,
             resized,
             input,
             &app, &toplevel, &form, &glxarea);
  initCustomized (form, glxarea);
  XtRealizeWidget (toplevel);     /*  Realize the complete widget hierarchy */
  XtAppMainLoop (app);            /*  Begin event processing */
  return (0);                         /*  ANSI C requires main to return int */
}/*end-of-main*/

// OpenGLinit ------------------------------------------------------------------------------
//   This function is called by Motif as a "callback" during one of the
//   first "events", namely, when Motif initializes the OpenGL drawing area.
// Parameters
//   w          -- glxarea
//   clientData -- NULL  ( 4th parameter in XtAddCallback )
//   call       -- reason / event / width / height
// Returns
//   nothing
// ----------------------------------------------------------------------------------------
void OpenGLinit (Widget w, XtPointer clientData, XtPointer call)
{
  GLfloat mat_ambient[]    = { 1.0, 0.5, 0.5, 1.0 };
  GLfloat mat_diffuse[]    = { 1.0, 0.5, 0.0, 1.0 };
  GLfloat mat_specular[]   = { 1.0, 1.0, 1.0, 1.0 };
  GLfloat mat_shininess[]  = { 100.0 };
  GLfloat light_position_2[] = {0.0, 2.5, 0.5, 1.0};
#ifdef DEBUG                              /*  This is only activated if the */
  printf("*** OpenGLinit ***\n");    /*  parameter CFLAGS in the make file */
#endif                                    /*  also includes the word "-DDEBUG". */

  XVisualInfo *visinfo;                   /* This variable is local to this function only */
  /* Create OpenGL rendering context. */
  XtVaGetValues (w, GLwNvisualInfo, &visinfo, NULL);
  glxcontext = glXCreateContext (XtDisplay(w), visinfo, 0, True);

  /* Setup OpenGL state. */
  glXMakeCurrent (XtDisplay(w), XtWindow(w), glxcontext);
  r1 = new int[10];
  g1 = new int[10];
  b1 = new int[10];
```

```
  glGetIntegerv(GL_RED_BITS, r1);
  glGetIntegerv(GL_GREEN_BITS, g1);
  glGetIntegerv(GL_BLUE_BITS, b1);
  glClearColor (0.9, 0.9, 0.9, 0.0);
  glMaterialfv (GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
  glMaterialfv (GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);
  glLightfv (GL_LIGHT0, GL_AMBIENT, mat_ambient);
  glLightfv (GL_LIGHT0, GL_POSITION,light_position_2);
  glShadeModel (GL_SMOOTH);

  glEnable (GL_LIGHTING);
  glEnable (GL_LIGHT0);
  glEnable (GL_DEPTH_TEST);
  glEnable (GL_NORMALIZE);
  glEnable (GL_MAP2_VERTEX_3);
  init_surface();              // call function computing surface data
  init_new_surface( );
}/* end-of-OpenGLinit */
/*=========================================================================
// exposed -- GLwNexposeCallback
//   This is also a callback function.  It is called each time Motif sees
//   that the OpenGL drawing area needs to be redrawn, including when the
//   area is first created.  It therefore calls "draw".
// Parameters
//   w          -- glxarea
//   clientData -- NULL  ( 4th parameter in XtAddCallback )
//   call       -- reason / event / width / height
//
// Returns
//   nothing
//=========================================================================*/
void exposed (Widget w, XtPointer clientData, XtPointer call)
{
#ifdef DEBUG
  printf("*** exposed ***\n");
#endif
  draw (XtDisplay(area), XtWindow(area), int (identifier));
}/* end-of-exposed */
//============================================================
// resized -- GLwNresizeCallback
//   This callback function is activated each time the window is resized.
//   Note that a resize event will automatically generate a subsequent
//   expose event, which in turn will cause the window to be redrawn.
//   This function therefore does not have to call "draw" directly itself.
// Parameters
//   w          -- glxarea
//   clientData -- NULL  ( 4th parameter in XtAddCallback )
//   call       -- reason / event / width / height
// Returns
//   nothing
//============================================================
void resized (Widget w, XtPointer clientData, XtPointer call)
{
#ifdef DEBUG
  printf("*** resize ***\n");
#endif
```

```
  GLwDrawingAreaCallbackStruct *callData;
  callData = (GLwDrawingAreaCallbackStruct *) call;
  glXMakeCurrent (XtDisplay(w), XtWindow(w), glxcontext);
  /* SYNCHRONIZE:  Make sure the X resize event is complete before
     proceeding with the OpenGL resizing event... */
  glXWaitX ();

  /* ...proceed with the OpenGL process:
     This next statement will case the image to loose its aspect ratio. */
  glViewport (0, 0, callData->width, callData->height);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  if(callData->width <= callData->height)
    glOrtho (-5.0, 5.0, -5.0 * (GLfloat)callData->height/(GLfloat)callData->width,
                5.0 * (GLfloat)callData->height/(GLfloat)callData->width, 0.0, 20.0);
  else
    glOrtho (-5.0 * (GLfloat)callData->width/(GLfloat)callData->height,
                5.0 * (GLfloat)callData->width/(GLfloat)callData->height, -5.0, 5.0,
                0.0, 20.0);

}/* end-of-resized */
//=========================================================================
// input -- GLwNinputCallback
//   This callback function is activated each time the drawing area gets
//   a mouse of keyboard event.
// Parameters
//   w          -- glxarea
//   clientData -- NULL  ( 4th parameter in XtAddCallback )
//   call       -- reason / event / width / height
// Returns
//   nothing
//=========================================================================
static void input (Widget w, XtPointer clientData, XtPointer call)
{
  GLwDrawingAreaCallbackStruct *callData;
  callData = (GLwDrawingAreaCallbackStruct *) call;
  XEvent *event = callData->event;
}//end-of-input
//=========================================================================
// initCustomized --
//   This function creates additional widgets, registers their callbacks,
//   and located them relative to each other.
// Parameters
//   form       -- this is the main widget
//   glxarea    -- this widget if a child of the form widget
//
// Returns
//   nothing
//=========================================================================
static void initCustomized (Widget form, Widget glxarea)
{
  area = glxarea;
  // Create  menubar
  Widget menubar = XmVaCreateSimpleMenuBar (form, "menubar", NULL, 0);
  XtManageChild (menubar);
  // Create and locate the Exit button
```

```
 XmString exitButtonText = XmStringCreateLocalized ("EXIT");
 Widget exitButton = XtVaCreateManagedWidget ("exitButton",
   xmPushButtonWidgetClass, form,
   XmNlabelString,         exitButtonText,
   XmNrightAttachment,     XmATTACH_FORM,
   XmNtopAttachment,       XmATTACH_FORM,
   XmNheight,                50,
   XmNwidth,                 100,
   XmNtopOffset,             130,
   XmNleftOffset,            0,
   NULL);
 XmStringFree (exitButtonText);
 XtAddCallback (exitButton, XmNactivateCallback, buttonCallback, (XtPointer) 1);

  // Create the UP arrow

   Widget up = XtVaCreateManagedWidget ( "arrowUp",
                                         xmArrowButtonGadgetClass, form,
                                         XmNtopAttachment,       XmATTACH_FORM,
                                         XmNtopOffset,           35,
                                       XmNarrowDirection,      XmARROW_UP,
                                         XmNrightAttachment, XmATTACH_FORM,
                                         XmNrightOffset,             40,
                                         NULL );
  XtAddCallback ( up, XmNarmCallback, buttonCallback, ( XtPointer ) 2);

// Create the RIGHT arrow
   Widget right = XtVaCreateManagedWidget ( "arrowRight",
                                         xmArrowButtonGadgetClass, form,
                                         XmNtopAttachment,       XmATTACH_WIDGET,
                                         XmNtopWidget,           up,
                                         XmNleftAttachment,   XmATTACH_WIDGET,
                                         XmNleftWidget,              up,
                                         XmNarrowDirection,      XmARROW_RIGHT,
                                         NULL );
   XtAddCallback ( right, XmNarmCallback, buttonCallback, ( XtPointer ) 3);

// Create the LEFT arrow
   Widget left = XtVaCreateManagedWidget ( "arrowLeft",
                                         xmArrowButtonGadgetClass, form,
                                         XmNtopAttachment,       XmATTACH_WIDGET,
                                         XmNtopWidget,           up,
                                         XmNrightAttachment, XmATTACH_WIDGET,
                                         XmNrightWidget,             right,
                                         XmNrightOffset,         20,
                                         XmNarrowDirection,      XmARROW_LEFT,
                                         NULL );
   XtAddCallback ( left, XmNarmCallback, buttonCallback, ( XtPointer ) 4);

// Create the DOWN arrow
   Widget down = XtVaCreateManagedWidget ( "arrowDown",
                                         xmArrowButtonGadgetClass, form,
                                         XmNtopAttachment,       XmATTACH_WIDGET,
                                         XmNtopWidget,           left,
                                         XmNleftAttachment,   XmATTACH_WIDGET,
                                         XmNleftWidget,              left,
```

```
                                        XmNarrowDirection,      XmARROW_DOWN,
                                        NULL );
  XtAddCallback ( down, XmNarmCallback, buttonCallback, ( XtPointer ) 5);

// Create and locate the Zoom slider

  Widget zoomslider = XtVaCreateManagedWidget ("Zoom slider",
                                        xmScaleWidgetClass,     form,
                                        XtVaTypedArg,           XmNtitleString,
                                        XmRString,              " -   Zoom   +", 10,
                                        XmNmaximum,             20,
                        XmNminimum,             2,
                        XmNvalue,               9,
                        XmNshowValue,           False,
                        XmNdecimalPoints,       1,   //divide max by 10...move one decimal place right
                        XmNorientation,         XmHORIZONTAL,
                        XmNrightAttachment,     XmATTACH_WIDGET,
                        XmNrightWidget,         left,
                        XmNrightOffset,         50,
                        NULL);
  XtAddCallback (zoomslider, XmNdragCallback,sliderCallback, (XtPointer) 6);
  XtAddCallback (zoomslider, XmNvalueChangedCallback, sliderCallback, (XtPointer) 6);

  // Create and locate the Drawsurface button
  XmString drawsurfaceButtonText = XmStringCreateLocalized ("fuselage");
  Widget drawsurfaceButton = XtVaCreateManagedWidget ("drawsurfaceButton",
    xmPushButtonWidgetClass, form,
    XmNlabelString,         drawsurfaceButtonText,
    XmNrightAttachment,     XmATTACH_FORM,
    XmNbottomAttachment,    XmATTACH_FORM,
    XmNheight,              50,
    XmNwidth,               100,
    XmNbottomOffset,        20,
    XmNleftOffset,          0,
    NULL);
  XmStringFree (drawsurfaceButtonText);
  XtAddCallback (drawsurfaceButton, XmNactivateCallback, buttonCallback, (XtPointer) 7);


  // Create and locate the Drawsurface button
  XmString drawtrimsurfaceButtonText = XmStringCreateLocalized ("f_trimed");
  Widget drawtrimsurfaceButton = XtVaCreateManagedWidget ("drawtrimsurfaceButton",
    xmPushButtonWidgetClass, form,
    XmNlabelString,         drawtrimsurfaceButtonText,
    XmNrightAttachment,     XmATTACH_FORM,
    XmNbottomAttachment,    XmATTACH_WIDGET,
    XmNbottomWidget,        drawsurfaceButton,
    XmNheight,              50,
    XmNwidth,               100,
    XmNbottomOffset,        10,
    XmNleftOffset,          0,
    NULL);
  XmStringFree (drawtrimsurfaceButtonText);
  XtAddCallback (drawtrimsurfaceButton, XmNactivateCallback, buttonCallback, (XtPointer) 8);

  // Create and locate the Drawsurface button
```

```
  XmString overlapButtonText = XmStringCreateLocalized ("wing");
  Widget overlapButton = XtVaCreateManagedWidget ("overlapButton",
    xmPushButtonWidgetClass, form,
    XmNlabelString,         overlapButtonText,
    XmNrightAttachment,     XmATTACH_FORM,
    XmNbottomAttachment,    XmATTACH_WIDGET,
    XmNbottomWidget,        drawtrimsurfaceButton,
    XmNheight,              50,
    XmNwidth,               100,
    XmNbottomOffset,        10,
    XmNleftOffset,          0,
    NULL);
  XmStringFree (overlapButtonText);
  XtAddCallback (overlapButton, XmNactivateCallback, buttonCallback, (XtPointer) 9);

 // Create and locate the DrawWing button
  XmString wingButtonText = XmStringCreateLocalized ("wing_trimed");
  Widget wingButton = XtVaCreateManagedWidget ("wingButton",
    xmPushButtonWidgetClass, form,
    XmNlabelString,         wingButtonText,
    XmNrightAttachment,     XmATTACH_FORM,
    XmNbottomAttachment,    XmATTACH_WIDGET,
    XmNbottomWidget,        overlapButton,
    XmNheight,              50,
    XmNwidth,               100,
    XmNbottomOffset,        10,
    XmNleftOffset,          0,
    NULL);
  XmStringFree (wingButtonText);
  XtAddCallback (wingButton, XmNactivateCallback, buttonCallback, (XtPointer) 10);

 // Create and locate the DrawWingTrimming button
  XmString wingTrimButtonText = XmStringCreateLocalized ("fuselage&wing");
  Widget wingTrimButton = XtVaCreateManagedWidget ("wingTrimButton",
    xmPushButtonWidgetClass, form,
    XmNlabelString,         wingTrimButtonText,
    XmNrightAttachment,     XmATTACH_FORM,
    XmNbottomAttachment,    XmATTACH_WIDGET,
    XmNbottomWidget,        wingButton,
    XmNheight,              50,
    XmNwidth,               100,
    XmNbottomOffset,        10,
    XmNleftOffset,          0,
    NULL);
  XmStringFree (wingTrimButtonText);
  XtAddCallback (wingTrimButton, XmNactivateCallback, buttonCallback, (XtPointer) 11);

   // Create and locate the DrawWingTrimming overlap button
  XmString wingOverlapButtonText = XmStringCreateLocalized ("f_w_trimed");
  Widget wingOverlapButton = XtVaCreateManagedWidget ("wingOverlapButton",
    xmPushButtonWidgetClass, form,
    XmNlabelString,         wingOverlapButtonText,
    XmNrightAttachment,     XmATTACH_FORM,
    XmNbottomAttachment,    XmATTACH_WIDGET,
    XmNbottomWidget,        wingTrimButton,
    XmNheight,              50,
```

```
  XmNwidth,                 100,
  XmNbottomOffset,          10,
  XmNleftOffset,            0,
  NULL);
XmStringFree (wingOverlapButtonText);
XtAddCallback (wingOverlapButton, XmNactivateCallback, buttonCallback, (XtPointer) 12);

// Create and locate the Draw Wing button
XmString thirteenButtonText = XmStringCreateLocalized ("blending_s");
Widget thirteenButton = XtVaCreateManagedWidget ("thirteenButton",
  xmPushButtonWidgetClass, form,
  XmNlabelString,         thirteenButtonText,
  XmNrightAttachment,     XmATTACH_FORM,
  XmNbottomAttachment,    XmATTACH_WIDGET,
  XmNbottomWidget,        wingOverlapButton,
  XmNheight,              50,
  XmNwidth,               100,
  XmNbottomOffset,        10,
  XmNleftOffset,          0,
  NULL);
XmStringFree (thirteenButtonText);
XtAddCallback (thirteenButton, XmNactivateCallback, buttonCallback, (XtPointer) 13);

// Create and locate the Wing trimming button
XmString fourteenButtonText = XmStringCreateLocalized ("f&wtrimed_p");
Widget fourteenButton = XtVaCreateManagedWidget ("fourteenButton",
  xmPushButtonWidgetClass, form,
  XmNlabelString,         fourteenButtonText,
  XmNrightAttachment,     XmATTACH_FORM,
  XmNbottomAttachment,    XmATTACH_WIDGET,
  XmNbottomWidget,        thirteenButton,
  XmNheight,              50,
  XmNwidth,               100,
  XmNbottomOffset,        10,
  XmNleftOffset,          0,
  NULL);
XmStringFree (fourteenButtonText);
XtAddCallback (fourteenButton, XmNactivateCallback, buttonCallback, (XtPointer) 14);

 // Create and locate the   button
XmString fifteenButtonText = XmStringCreateLocalized ("f_middle ");
Widget fifteenButton = XtVaCreateManagedWidget ("fifteenButton",
  xmPushButtonWidgetClass, form,
  XmNlabelString,         fifteenButtonText,
  XmNrightAttachment,     XmATTACH_FORM,
  XmNbottomAttachment,    XmATTACH_WIDGET,
  XmNbottomWidget,        fourteenButton,
  XmNheight,              50,
  XmNwidth,               100,
  XmNbottomOffset,        10,
  XmNleftOffset,          0,
  NULL);
XmStringFree (fifteenButtonText);
XtAddCallback (fifteenButton, XmNactivateCallback, buttonCallback, (XtPointer) 15);

// Create and locate the    button
```

```
 XmString sixteenButtonText = XmStringCreateLocalized ("blending_p");
 Widget sixteenButton = XtVaCreateManagedWidget ("sixteenButton",
  xmPushButtonWidgetClass, form,
  XmNlabelString,          sixteenButtonText,
  XmNrightAttachment,      XmATTACH_FORM,
  XmNbottomAttachment,     XmATTACH_WIDGET,
  XmNbottomWidget,         fifteenButton,
  XmNheight,               50,
  XmNwidth,                100,
  XmNbottomOffset,         10,
  XmNleftOffset,           0,
  NULL);
 XmStringFree (sixteenButtonText);
 XtAddCallback (sixteenButton, XmNactivateCallback, buttonCallback, (XtPointer) 16);

// Update location of Motif / OpenGLEXIT drawing area
  XtVaSetValues (glxarea,
                 XmNtopAttachment,    XmATTACH_WIDGET,
                 XmNtopWidget,        menubar,
                 XmNbottomAttachment, XmATTACH_WIDGET,
                 XmNbottomWidget,     drawsurfaceButton,
                 NULL);

}//end-of-initCustomized

//========================================================================
// buttonCallback --
//   This callback function is activated each time a button is hit.
//
// Parameters
//   w          -- exitButton or redButton
//   clientData -- button ID  ( 4th parameter in XtAddCallback )
//   call       -- reason / event / click_count
//
// Returns
//   nothing
//========================================================================
void buttonCallback (Widget w, XtPointer clientData, XtPointer call)
{
 switch ((int)clientData)
 {
 case 1:
  // Process the "EXIT" button being hit.
  exit (0);
  break;

 case 2:
  // process the "UP_ARROW" button being hit
  phi = phi + 4.0;
  if( phi >=  88.0 ) phi =  88.0;
  if( phi <= -88.0 ) phi = -88.0;
  isometric (phi, theta, r);
  glScalef(4.0,4.0,4.0);
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;
```

```
case 3:
  // process the "RIGHT_ARROW" button being hit
  theta = theta - 4.0;
  isometric (phi, theta, r);
  glScalef(3.0,3.0,3.0);
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

case 4:
  // process the "LEFT_ARROW" button being hit
  theta = theta + 4.0;
  isometric (phi, theta, r);
  glScalef(3.0,3.0,3.0);
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

case 5:
  // process the "DOWN_ARROW" button being hit
  phi = phi - 4.0;
  if( phi >=  88.0 ) phi =  88.0;
  if( phi <= -88.0 ) phi = -88.0;
  isometric (phi, theta, r);
  glScalef(3.0,3.0,3.0);
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

case 7:
  // process the "fuselage" button being hit

  identifier = 1;
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

case 8:
  // process the "f_trim" button being hit

  identifier = 2;
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

case 9:
  // process the "overlap" button being hit

  identifier = 3;
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

case 10:
  // process the "11_curve" button being hit

  identifier = 4;
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

case 11:
  // process the "21_curve" button being hit
```

```
  identifier = 5;
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

 case 12:
  // process the "f_grid" button being hit

  identifier = 6;
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

 case 13:
  // process the "wing" button being hit

  identifier = 7;
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

 case 14:
  // process the "trim_w" button being hit

  identifier = 8;
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

 case 15:
  // process the "  " button being hit

  identifier = 9;
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

 case 16:
  // process the "  " button being hit

  identifier = 10;
  draw ( XtDisplay ( area ), XtWindow ( area ), int (identifier));
  break;

 }
 // This should never happen...
 cout << "WARNING : unknown callBack\n";
} // end-of-button-callBack
//=====================================================================
// sliderCallback --
//   This callback function is activated each time the slider is moved
// Parameters
//   dpy       -- XtDisplay (glxarea)
//   win       -- XtWindow (glxarea)
//   mode      -- Render mode in this case
//   index-- Counter that tells the program option selected by user
// Returns
//   nothing
//=====================================================================
static void sliderCallback( Widget w, XtPointer clientData, XtPointer call)
```

```
{
 XmScaleCallbackStruct *cbs = (XmScaleCallbackStruct*) call;

 if (cbs->reason == XmCR_DRAG)
 {
  cout << "SLIDER DRAGGED: " << cbs->value << "\n";
  //process the slider for zoom value being changed

  if(((int)clientData) == 6)
    {
          r = (cbs->value/1.0);
          draw(XtDisplay(area), XtWindow(area), int (identifier));
          return;
    }
 }
 if (cbs->reason == XmCR_VALUE_CHANGED)
  {
    cout << "SLIDER VALUE CHANGED: " << cbs->value << "\n";

 if (((int)clientData) == 6)                    // Return the value of radius
    r = (cbs->value/1.0);
    //glScalef(5.0,5.0,5.0 +5*r);
    draw(XtDisplay(area), XtWindow(area), int (identifier));
    return;
  }

}//end-of-sliderback

//=================================================================
// isometric --
//   function which defines the viewport settings
//   parameters
//     phi   -- angle to effect the up and down movement
//     theta -- angle to effect the left and right mevement
//     r     -- variable for storing the zoom value increments
//   returns
//     nothing
//=================================================================
void isometric(float phi, float theta, float r)
{
 int x = 1000;
 int y = 1000;
 //Define the viewport for isometric view
 glColor3f (1.0, 1.0, 1.0);
 //Define view direction and orientation for the isometric view
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 a = cos(phi/180*pi) * sin(theta/180.0*pi) * (3+r);
 b = sin(phi/180*pi) * (3+r);
 c = cos(phi/180*pi) * cos(theta/180.0*pi) * (3+r);
 gluLookAt(  a,   b,   c,  0.0, 2.0, 1.0, 0.0, 1.0, 0.0);

}// end-of-isometric

//=================================================================
//  Initializes the control points on the surfaces
```

```
//=====================================================================
void init_surface( )
{
// local data
float     *data_points, *trim_points, *trim_data, *trim_data_f, *trim_offset;
float     trim_points_wing[25000];
int       numpts[2], numpts_wing[2];
float     *data_points_m,*trim_points_m;
int           trimnum, num;
float         *u_constraints, *v_constraints;
float         *ut_constraints, *vt_constraints;
int       *u_continuities, *v_continuities;
int       *ut_continuities, *vt_continuities;
float     *control_x, *control_y, *control_z;
float     *t_control_x, *t_control_y, *t_control_z;
float     *trim_points_r,*trim_points_l,*trim_points_new;
float     *orig_point_grid, *trim_point_grid;

//---------------------------------------------------------------------
//read in the data for the surface and the trim curve
//---------------------------------------------------------------------
  fp  = fopen("fuselage", "r");                  // read file "fuselage" fuselage patch 11 row and 11 column data
  fp1 = fopen("f_upper_trimdata", "r");       // the parametric data of open trimming curve for fuselage

  fscanf(fp, "%d", &numpts[0]);
  fscanf(fp, "%d", &numpts[1]);
  fscanf(fp1,"%d", &trimnum);

  data_points = new float[3*numpts[0]*numpts[1]];       // store fuselage data points
  trim_data_f   = new float[2*trimnum];                 // store trim data

  for (i=0; i< numpts[0]*numpts[1]*3; i++)
    fscanf(fp, "%f", &data_points[i]);
  for (i=0; i< 2*trimnum; i++)
    fscanf (fp1,"%f",&trim_data_f[i]);
  fclose(fp);
  fclose(fp1);

  trim_offset = new float[(trimnum+1)];                  // offset curve data
//---------------------------------------------------------------------------
// calculate the offset curve parametric data
//---------------------------------------------------------------------------
  calc_offset(offset, trim_data_f, trim_offset);
  delete [] trim_data_f;
  trim_data_f = NULL;
  trimnum = trimnum +1;
  trim_data = new float[2*trimnum];
  for (i=0; i< 2*trimnum; i++)
  trim_data[i] = trim_offset[i];
//---------------------------------------------------------------------------
//Calculate the knot vector and the control points for defining original surface
//---------------------------------------------------------------------------
  create_surf(numpts, data_points, u_newknots, v_newknots, contrlpts, newnum);
  delete []data_points;
  data_points = NULL;
//---------------------------------------------------------------------------
```

```
// calculate the wing data in an individual function
//-------------------------------------------------------------------------
  create_surf_w (offset, w_u_newknots,  w_v_newknots, w_contrlpts, w_newnum,  w_ut_newknots,
                    w_vt_newknots, w_t_contrlpts, w_t_newnum, numpts_wing, trim_points_wing);
//-------------------------------------------------------------------------
// get the wing trimming end point as base points to be used later for computing control point of fillet
//-------------------------------------------------------------------------
   float basepts[45];
   for(i=0; i<15; i++)
   {
   basepts[3*i]   = trim_points_wing[3*(9+i*10)+0];
   basepts[3*i+1] = trim_points_wing[3*(9+i*10)+1];
   basepts[3*i+2] = trim_points_wing[3*(9+i*10)+2];
   }

  fp6 = fopen("3_conic", "w");
  for (i=0; i<45; i++)
    fprintf(fp6, "%13.9f %13.9f %13.9f\n", basepts[3*i], basepts[3*i+1], basepts[3*i+2]);
  fclose(fp6);
//-----------------------------------------------------------------
//Obtain points on the original surface in model space for the range  specified by the trim data
//-----------------------------------------------------------------
  control_x = new float[2*(numpts[0]+2)*(trimnum+2)];
  control_y = new float[2*(numpts[0]+2)*(trimnum+2)];
  control_z = new float[2*(numpts[0]+2)*(trimnum+2)];

  for (i=0; i < (numpts[0]+2)*(numpts[1]+2); i++)
    {
      control_x[i] = contrlpts[i*3+0];
      control_y[i] = contrlpts[i*3+1];
      control_z[i] = contrlpts[i*3+2];
    }
//-------------------------------------------------------------------------
// In order to perform an error analysis at a later stage, collect a grid of points that lie on the original surface for
the
// range specified by the trim data. This grid size will be maintained constant so that  corresponding and equal
// number of points can be obtained on the trimmed patch also.
//-------------------------------------------------------------------------
  grid_u = 50;
  grid_v = 50;
  orig_point_grid = new float [3*grid_u*grid_v];
  get_orig_points(numpts[0], trimnum, grid_u, grid_v, trim_data, u_newknots,
                          v_newknots, control_x, control_y, control_z, orig_point_grid);
//-------------------------------------------------------------------------
// compute the trim data based on parametric trim data and control points
//-------------------------------------------------------------------------
  trim_points = new float[3*numpts[0]*trimnum];
  point(numpts[0], trimnum, u_newknots, v_newknots, control_x, control_y, control_z, trim_data, trim_points);
  fp6 = fopen("1f_trim_points", "w");
  for (i=0; i<numpts[0]*trimnum; i++)
    fprintf(fp6, "%13.9f %13.9f %13.9f\n", trim_points[3*i], trim_points[3*i+1], trim_points[3*i+2]);
  fclose(fp6);
//-------------------------------------------------------------------------
// split the trimmed patch of fuselage into three part: right patch,
// middle patch,left patch.
//-------------------------------------------------------------------------
```

```
  trim_points_r = new float[3*5*11];
  trim_points_l = new float[3*4*11];
  for (i=0; i<3*5*11; i++)
    trim_points_r[i] = trim_points[i];       // for right patch
  for (i=0; i<3*4*11; i++)
    trim_points_l[i]= trim_points[i+3*18*11];   // for left patch

  f = fopen("1_5_11", "w");
  for (i=0; i<5*11; i++)
  fprintf(f, "%13.9f %13.9f %13.9f\n", trim_points_r[3*i], trim_points_r[3*i+1], trim_points_r[3*i+2]);
  fclose(f);

  f1 = fopen("1_4_11", "w");
  for (i=0; i<4*11; i++)
  {
  fprintf(f1, "%13.9f %13.9f %13.9f\n", trim_points_l[3*i], trim_points_l[3*i+1],
              trim_points_l[3*i+2]);
  }
  fclose(f1);

  trim_points_m = new float[3*15*11];           // for middle patch
  for (i=0; i<3*15*11; i++)
  trim_points_m[i]= trim_points[3*4*11+i];

  f = fopen("f_trim_m", "w");
  for (i=0; i<15*11; i++)
  fprintf(f, "%13.9f %13.9f %13.9f\n", trim_points_m[3*i], trim_points_m[3*i+1], trim_points_m[3*i+2]);
  fclose(f);
//-------------------------------------------------------------------------
// put fuselage middle patch and wing patch together in order to make blending
//-------------------------------------------------------------------------
  float p_inv[5000];          // tempory data for inverting trim_points_wing data order
                              // for example invert 1, 2, 3, 4, into 4, 3, 2, 1;
  for (j=0; j<15; j++)
  {
    for (i=10*j; i<10*(j+1); i++)
    {
      p_inv[3*i]   = trim_points_wing[3*(2*10*j+10-i)-3];
      p_inv[3*i+1] = trim_points_wing[3*(2*10*j+10-i)-2];
      p_inv[3*i+2] = trim_points_wing[3*(2*10*j+10-i)-1];
    }
  }
  float p15_21[5000];                                   // temporay data for storing blending patch points
  for (j=0; j<15; j++)
  {
    for (i = 3*(11+10)*j; i<3*(11*(j+1)+10*j); i++)
    p15_21[i] = trim_points_m[i-3*10*j];

    for (i = 3*(11*(j+1)+10*j);   i<3*(11+10)*(j+1); i++)
    p15_21[i] = p_inv[i-3*11*(j+1)];
  }

  f3 = fopen("1_15_21", "w");
  for (i=0; i<15*21; i++)
  {
  fprintf(f3, "%13.9f %13.9f %13.9f\n", p15_21[3*i], p15_21[3*i+1],
```

```
            p15_21[3*i+2]);
  }
  fclose(f3);
//-------------------------------------------------------------------------
//   add 3 iso-parametric curves near the trimming curve of wing
//-------------------------------------------------------------------------
float tempdata[20000];
// add three more curve away from offset trim curve on the wing
refine_pts( 15 /* in: row number */, 10/*in: original points number each row*/,
          13/* in: out points number per row*/, trim_points_wing/*in: data needed to refine*/,
          tempdata /* out: */ );


  for (j=0; j<15; j++)                // trim_points_wing need to be inversed;
  {
    for (i=13*j; i<13*(j+1); i++)
    {
      trim_points_wing[3*i]   = tempdata[3*(2*13*j+13-i)-3];
      trim_points_wing[3*i+1] = tempdata[3*(2*13*j+13-i)-2];
      trim_points_wing[3*i+2] = tempdata[3*(2*13*j+13-i)-1];
    }
  }
}
//-----------------------------------------------------------------------------
// put fuselage middle patch and refined wing patch together in order to make blending,
// and get 15 X 24 blending surface points
//-----------------------------------------------------------------------------
  float p15_24[5000];
  for (j=0; j<15; j++)
  {
    for (i = 3*(11+13)*j; i<3*(11*(j+1)+13*j); i++)
    p15_24[i] = trim_points_m[i-3*13*j];

    for (i = 3*(11*(j+1)+13*j);   i<3*(11+13)*(j+1); i++)
    p15_24[i] = trim_points_wing[i-3*11*(j+1)];
  }

//-----------------------------------------------------------------------------
// add 3 iso curves on the side of fuselage middle part away from trimming curve,
// after refinement, original 15*11 now 15*14
//-----------------------------------------------------------------------------
  refine_pts( 15, 11, 14, trim_points_m,tempdata );

  trim_points_new = new float[3*15*(13+14)];

  for (j=0; j<15; j++)
  {
    for (i = 3*(14+13)*j; i<3*(14*(j+1)+13*j); i++)
    trim_points_new[i] = tempdata[i-3*13*j];

    for (i = 3*(14*(j+1)+13*j);   i<3*(14+13)*(j+1); i++)
    trim_points_new[i] = trim_points_wing[i-3*14*(j+1)];
  }


//-----------------------------------------------------------------------------
```

```
// compute the fillet shape controling curve in order to control the fillet radius,
// at the same time, make the fillet blending patch in right shape
//-------------------------------------------------------------------------------
  float  blendpts[45];                        // curve on trimmed fuselage patch and wing patch;
  for (i=0; i<15; i++)
  {
    conic ( 0.5 , 0.5, trim_points_new[3*(13+27*i)+0], trim_points_new[3*(13+27*i+1)+0], basepts[3*i],
            blendpts[3*i]);                   // t = 0.5 and p = 0.5 for one point on the specified conic curve.
    conic ( 0.5 , 0.5, trim_points_new[3*(13+27*i)+1], trim_points_new[3*(13+27*i+1)+1], basepts[3*i+1],
            blendpts[3*i+1]);
    conic ( 0.5 , 0.5, trim_points_new[3*(13+27*i)+2], trim_points_new[3*(13+27*i+1)+2], basepts[3*i+2],
            blendpts[3*i+2]);
   }

//-------------------------------------------------------------------------------------
// add the above fillet controling curve in the blending surface
//-------------------------------------------------------------------------------------
float  *newpts;                                 // store point data of the final middle blending part
 newpts = new float[3*15*28];
 for(int j=0; j<15; j++)
 {
   for (i=15+14*2*(j-1); i<14*(2*j+1); i++)
   {
    newpts[3*i]   = trim_points_new[3*(i-j)+0];
    newpts[3*i+1] = trim_points_new[3*(i-j)+1];
    newpts[3*i+2] = trim_points_new[3*(i-j)+2];
   }
    i = 14*(2*j+1);
    newpts[3*i]   = blendpts[3*(i-14*(2*j+1)+j)+0];
    newpts[3*i+1] = blendpts[3*(i-14*(2*j+1)+j)+1];
    newpts[3*i+2] = blendpts[3*(i-14*(2*j+1)+j)+2];
 }
   for (i=407; i<420; i++)
  {
   newpts[3*i]   = trim_points_new[3*(i-407+392)+0];
   newpts[3*i+1] = trim_points_new[3*(i-407+392)+1];
   newpts[3*i+2] = trim_points_new[3*(i-407+392)+2];
  }

  f4 = fopen("1_15_28", "w");
  for (i=0; i<15*28; i++)
   fprintf(f4, "%13.9f %13.9f %13.9f\n", newpts[3*i], newpts[3*i+1],newpts[3*i+2]);
  fclose(f4);
//-------------------------------------------------------------------------------
//Sieve the original surface points information obtained from the previous operation
//to obtain the points that define "u" number of isoparametric curves.
//-------------------------------------------------------------------------------
  num = numpts[0];
  vt_continuities = new int[trimnum];
  for (i=0; i<trimnum; i++)vt_continuities[i] = 2;

  ut_continuities = new int[numpts[0]];
  for (i=0; i<num; i++)ut_continuities[i] = 2;
  ut_constraints = new float[trimnum*numpts[0]*4];
  constraintu(num, trimnum, u_newknots, v_newknots, control_x, control_y, control_z,
                trim_data, ut_constraints);
```

```
  vt_constraints = new float[trimnum*numpts[0]*4];
  constraintv(num, trimnum, u_newknots, v_newknots, control_x, control_y, control_z,
                trim_data, vt_constraints);
//-------------------------------------------------------------------
// Calculate the knot vector and the control points for defining trimmed surface. This trimmed surface
//  is a result of a random number of equally spaced trim data.
//-------------------------------------------------------------------
  numpts[1]=trimnum;
  Create_NUBS3_Surface(numpts, trim_points, param, ut_continuities, vt_continuities, ut_constraints,
                                    vt_constraints, ut_newknots, vt_newknots, t_contrlpts, t_newnum);
//-------------------------------------------------------------------------
//Collect a grid of points that lie on the "trimmed surface without a norm"
//-------------------------------------------------------------------------
  t_control_x = new float[2*(numpts[0]+2)*(trimnum+2)];
  t_control_y = new float[2*(numpts[0]+2)*(trimnum+2)];
  t_control_z = new float[2*(numpts[0]+2)*(trimnum+2)];
  for (i = 0; i<((numpts[0]+2)*(numpts[1]+2)); i++)
    {
      t_control_x[i] = t_contrlpts[i*3+0];
      t_control_y[i] = t_contrlpts[i*3+1];
      t_control_z[i] = t_contrlpts[i*3+2];
    }
  trim_point_grid = new float [3*grid_u*grid_v];
  get_trim_points(num, trimnum, grid_u, grid_v, ut_newknots, vt_newknots,
                          t_control_x, t_control_y, t_control_z, trim_point_grid);
  fp4 = fopen("trimsurfpts_11_22", "w");
  for (i=0; i<grid_u*grid_v; i++)
    fprintf(fp4, "%13.9f %13.9f %13.9f\n", trim_point_grid[3*i], trim_point_grid[3*i+1],
trim_point_grid[3*i+2]);
  fclose(fp4);

  delete []ut_continuities;
  ut_continuities = NULL;
  delete []vt_continuities;
  vt_continuities = NULL;
  delete []ut_constraints;
  ut_constraints = NULL;
  delete [] vt_constraints;
  vt_constraints = NULL;

  delete [] trim_points_m;
  delete [] trim_points_new;
  trim_points_m = NULL;
  trim_points_new = NULL;
  delete [] newpts;
  newpts = NULL;
  delete [] orig_point_grid;
  orig_point_grid = NULL;
  delete [] trim_point_grid;
  trim_point_grid = NULL;
  delete [] t_control_x;
  delete [] t_control_y;
  delete [] t_control_z;
  t_control_x = NULL;
  t_control_y = NULL;
  t_control_z = NULL;
```

```
}
//=====================================================================
// compute control points and knot vector of new blending surfaces which includes  three patches: right patch
//  is from the trimmed patch of fuselage right side, left patch is from the trimmed patch of fuselage left side,
//  and the middle patch is made of trimmed wing patch and middle portion of trimmed fuselage.
//=====================================================================
void init_new_surface( )
{
  FILE    *p, *p1, *p2, *p3;
  int          numpts1[2], numpts_r[2],numpts_l[2];
  float    *data_pts,  *trim_points_r, *trim_points_l;
  //-------------------------------------------------------------------
  //read in the data for the surface and the trim curve
  //-------------------------------------------------------------------
  p  = fopen("15_27", "r");                                    // read the data file for the middle patch of blending
  fscanf(p, "%d", &numpts1[0]);                                // surface.
  fscanf(p, "%d", &numpts1[1]);
  data_pts = new float[3*numpts1[0]*numpts1[1]];

  for(i=0; i<numpts1[0]*numpts1[1]*3; i++)
    fscanf(p, "%f", &data_pts[i]);

  fclose(p);

  create_surf(numpts1, data_pts, u_newknots_m, v_newknots_m, contrlpts_m, newnum_m);

// ====================================================================
  p1  = fopen("5_11", "r");                                    // read file for right patch
  fscanf(p1, "%d", &numpts_r[0]);
  fscanf(p1, "%d", &numpts_r[1]);
  trim_points_r = new float[3*numpts_r[0]*numpts_r[1]];
  for(i=0; i<numpts_r[0]*numpts_r[1]*3; i++)
  fscanf(p1, "%f", &trim_points_r[i]);
  fclose(p1);

  create_surf(numpts_r, trim_points_r, u_newknots_r, v_newknots_r, contrlpts_r, newnum_r);
// ====================================================================
  p2  = fopen("4_11", "r");                                    // read file for left patch
  fscanf(p2, "%d", &numpts_l[0]);
  fscanf(p2, "%d", &numpts_l[1]);
  trim_points_l = new float[3*numpts_l[0]*numpts_l[1]];
  for(i=0; i<numpts_l[0]*numpts_l[1]*3; i++)
  fscanf(p2, "%f", &trim_points_l[i]);
  fclose(p2);
  create_surf(numpts_l, trim_points_l, u_newknots_l, v_newknots_l, contrlpts_l, newnum_l);
//====================================================================
  delete []data_pts;
  data_pts = NULL;
  delete []trim_points_r;
  trim_points_r = NULL;
  delete []trim_points_l;
  trim_points_l = NULL;
  identifier = 1;
  draw(XtDisplay(area), XtWindow(area), int (identifier));
```

```
}
// ====================================================================
//   This function is called when the drawing area needs to be updated.
//   It will redraw the image in the OpenGL drawing area.
//   It has here been defined as a static function; it will therefore not
//   be called from function defined in other files.
//
// Parameters
//   dpy        -- XtDisplay (glxarea)
//   win        -- XtWindow (glxarea)
//
// Returns
//   nothing
//====================================================================
static void draw(Display *dpy, Window win, int identifier)
{
 #ifdef DEBUG
 printf("*** draw ***\n");
 #endif

 glXMakeCurrent (dpy, win, glxcontext);
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

 switch(identifier)
 {
 case 1:
  glPushMatrix();
  glTranslatef(0.5, 0.5, -0.5);
  glRotatef(-115,0,1,0);
  glRotatef(-45,0,0,1);
  glRotatef(-30,1,0,0);

  display_g(u_newknots, v_newknots, contrlpts, newnum, dpy, win, glxcontext);     // draw fuselage
                                                    // display_g draws grid image.
  glCallList(nub);
  glPopMatrix();
  break;

 case 2:
  glPushMatrix();
  glTranslatef(0.5, 0.5, -0.5);
  glRotatef(-115,0,1,0);
  glRotatef(-45,0,0,1);
  glRotatef(-30,1,0,0);

  display_g(u_newknots, v_newknots, contrlpts, newnum, dpy, win,glxcontext);         // draw fuselage
  display_r(ut_newknots, vt_newknots, t_contrlpts, t_newnum, dpy, win, glxcontext);
                                                    // display_r draw shaded image
  glCallList(nub);
  glPopMatrix();
  break;

 case 3:
  glPushMatrix();
  glTranslatef(0.5, 0.5, -0.5);
  glRotatef(-115,0,1,0);
```

```
  glRotatef(-45,0,0,1);
  glRotatef(-30,1,0,0);
  display_g(w_u_newknots, w_v_newknots, w_contrlpts, w_newnum, dpy, win,glxcontext);
                // draw wing surface

  glCallList(nub);
  glPopMatrix();
  break;

 case 4:
  glPushMatrix();
  glTranslatef(0.5, 0.5, -0.5);
  glRotatef(-115,0,1,0);
  glRotatef(-45,0,0,1);
  glRotatef(-30,1,0,0);

  display_r(w_ut_newknots, w_vt_newknots, w_t_contrlpts, w_t_newnum, dpy, win,  glxcontext);
                        // draw trimmed wing patch
  display_g(w_u_newknots, w_v_newknots, w_contrlpts, w_newnum, dpy, win,  glxcontext);
                         // draw wing surface

  glCallList(nub);
  glPopMatrix();
  break;
 case 5:
  glPushMatrix();
  glTranslatef(0.5, 0.5, -0.5);
  glRotatef(-115,0,1,0);
  glRotatef(-45,0,0,1);
  glRotatef(-30,1,0,0);
  display_g(u_newknots, v_newknots, contrlpts, newnum, dpy, win,glxcontext);      // draw fuselage
  display_r(w_ut_newknots, w_vt_newknots, w_t_contrlpts, w_t_newnum, dpy, win,glxcontext);
                // draw trimmed wing
  glCallList(nub);
  glPopMatrix();
  break;

  case 6:
  glPushMatrix();
  glTranslatef(0.2, 0.5, -0.5);
  glRotatef(-115,0,1,0);
  glRotatef(-45,0,0,1);
  glRotatef(100,1,0,0);
  display_r(w_ut_newknots, w_vt_newknots, w_t_contrlpts, w_t_newnum,dpy, win, glxcontext);
  display_r(ut_newknots, vt_newknots, t_contrlpts, t_newnum,dpy, win, glxcontext);
  display_r(w_ut_newknots, w_vt_newknots, w_t_contrlpts, w_t_newnum,dpy, win,   glxcontext);
  display_r(ut_newknots, vt_newknots, t_contrlpts, t_newnum,dpy, win, glxcontext);
  glCallList(nub);
  glPopMatrix();
  break;

 case 7:
  glPushMatrix();
  glTranslatef(0.3, 0.5, -0.5);
  glRotatef(-35,0,0,1);
  display_r(u_newknots_m, v_newknots_m, contrlpts_m, newnum_m, dpy, win, glxcontext);
```

```
 display_r(u_newknots_r, v_newknots_r, contrlpts_r, newnum_r, dpy, win, glxcontext);
 display_r(u_newknots_l, v_newknots_l, contrlpts_l, newnum_l, dpy, win, glxcontext);
 display_g(u_newknots, v_newknots, contrlpts, newnum, dpy, win, glxcontext);     // draw fuselage
 display_g(w_u_newknots, w_v_newknots, w_contrlpts, w_newnum, dpy, win,glxcontext);
 glCallList(nub);
 glPopMatrix();
 break;

case 8:
 glPushMatrix();
 glTranslatef(0.5, 0.5, -0.5);
 glRotatef(-115,0,1,0);
 glRotatef(-45,0,0,1);
 glRotatef(-30,1,0,0);

 display_g(w_ut_newknots, w_vt_newknots, w_t_contrlpts, w_t_newnum,dpy, winglxcontext);
 display_g(ut_newknots, vt_newknots, t_contrlpts, t_newnum,dpy, win, glxcontext);
 glCallList(nub);
 glPopMatrix();
 break;

case 9:
 glPushMatrix();
 glTranslatef(0.5, 0.5, -0.5);
 glRotatef(-115,0,1,0);
 glRotatef(-45,0,0,1);
 glRotatef(-30,1,0,0);
 display_g(u_newknots_m, v_newknots_m, contrlpts_m, newnum_m, dpy, win, glxcontext);
 glCallList(nub);
 glPopMatrix();
 break;

 case 10:
 glPushMatrix();
 glTranslatef(0.5, 0.5, -0.5);
 glRotatef(-115,0,1,0);
 glRotatef(-45,0,0,1);
 glRotatef(-30,1,0,0);

 display_g(u_newknots_m, v_newknots_m, contrlpts_m, newnum_m, dpy, win, glxcontext);
 display_g(u_newknots_r, v_newknots_r, contrlpts_r, newnum_r, dpy, win, glxcontext);
 display_g(u_newknots_l, v_newknots_l, contrlpts_l, newnum_l, dpy, win, glxcontext);
 glCallList(nub);
 glPopMatrix();
 break;

default:
 glPushMatrix();
 display_g(u_newknots_m, v_newknots_m, contrlpts_m, newnum_m, dpy, win, glxcontext);
 display_g(u_newknots_r, v_newknots_r, contrlpts_r, newnum_r, dpy, win, glxcontext);
 display_g(u_newknots_l, v_newknots_l, contrlpts_l, newnum_l, dpy, win, glxcontext);

 glCallList(nub);
 glPopMatrix();
 break;
}
```

```
  if (doubleBuffer) glXSwapBuffers (dpy, win);
  else              glFlush ();

  /* Avoid indirect rendering latency from queuing. */
  if (!glXIsDirect (dpy, glxcontext)) glXWaitGL ();

}



/*****************************************************************************
   * Filename: setup_system.C
   * Author: Steven Fleming
   * Description: contains routines to set up system of linear equations necessary to
   *                    solve for the control points of a cubic B-spline.
   *****************************************************************************/
#include <math.h>
#include <stdio.h>
void setup_system(int numpts, float knots[], int opcl,int continuities[], int numctrl,float  Nmat[]);
extern void basis(float u, int interval, float knots[], float N[]);
extern void dbasis(float u, int interval, float knots[], float dN[]);
#define OPEN 0
#define CLOSED 1
/*===========================================================================
   * Function: setup_system
   *===========================================================================
   * Author: Steven Fleming
   * Description: Basically sets up basis function matrix
   * input: number of data points, array of continuities, open/closed flag, new knot vector, number of  control
              points
   * output: basis function matrix (nxn part of nxn+1 matrix)
*===========================================================================*/
void setup_system(int numpts, float knots[], int opcl,int continuities[], int numctrl,float  Nmat[])
{
  int continuity, i, j, n;
  int interval = 0;
  float N[4], dN[4], count;
  n = numctrl;
  continuity = continuities[0];
  switch(continuity)
    {
  case(0):
  case(1):
    interval = 3;
    dbasis(knots[3], interval , knots, dN);
    Nmat[0] = dN[0];
    Nmat[1] = dN[1];
    break;
  default:
    if ( opcl == OPEN)
      {
            Nmat[0] = 1;
            Nmat[1] = -1;
      }
    else
```

```
    {
        Nmat[0] = 1;
        Nmat[n-3] = -1;
    }
  }
interval = 3;
basis(knots[3], interval , knots, N);
Nmat[n+1] = N[0];
Nmat[n+2] = N[1];
Nmat[n+3] = N[2];
j = 2;
count = numpts-1;
for (i=1; i<count; i++)
  {
    continuity = continuities[i];
    switch(continuity)
        {
    case(0):
      interval = j+1;
        dbasis(knots[j+2], interval, knots, dN);
        Nmat[(n+1)*j+j] = dN[2];
        Nmat[(n+1)*j+j+1] = dN[3];
        j++;
        Nmat[(n+1)*j+j] = 1;
        j++;
      interval = j+2;
        dbasis(knots[j+2], interval, knots, dN);
        Nmat[(n+1)*j+j-1] = dN[0];
        Nmat[(n+1)*j+j] = dN[1];
        j++;
        break;
    case(1):
      interval = j+1;
        dbasis(knots[j+2], interval, knots, dN);
        Nmat[(n+1)*j+j] = dN[2];
        Nmat[(n+1)*j+j+1] = dN[3];
        j++;
      interval = j;
        basis(knots[j+2], interval, knots, N);
        Nmat[(n+1)*j+j-1] = N[2];
        Nmat[(n+1)*j+j] = N[3];
        j++;
        break;
    default:
      interval = j+2;
        basis(knots[j+2], j+2, knots, N);
        Nmat[(n+1)*j+j-1] = N[0];
        Nmat[(n+1)*j+j] = N[1];
        Nmat[(n+1)*j+j+1] = N[2];
        j++;
        }
  }
if (opcl == OPEN)
  {
    continuity = continuities[i];
    switch(continuity)
```

```
            {
    case(0):
    case(1):
      interval = j+1;
            basis(knots[j+2], interval, knots, N);
            Nmat[(n+1)*j+j] = N[2];
            Nmat[(n+1)*j+j+1] = N[3];
            j++;
      interval = j;
            dbasis(knots[j+1], interval, knots, dN);
            Nmat[(n+1)*j+j-1] = dN[2];
            Nmat[(n+1)*j+j] = dN[3];
            break;
    default:
      interval = j+2;
            basis(knots[j+2], interval , knots, N);
            Nmat[(n+1)*j+j-1] = N[0];
            Nmat[(n+1)*j+j] = N[1];
            Nmat[(n+1)*j+j+1] = N[2];
            j++;
            Nmat[(n+1)*j+j-1] = 1;
            Nmat[(n+1)*j+j] = -1;
            }
  }
 else
  {
    continuity = continuities[0];
    switch(continuity)
            {
    case(0):
            Nmat[(n+1)*j+j+1] = 1;
            Nmat[(n+1)*j] = -1;
            j++;
      interval = j;
            dbasis(knots[j+1], interval, knots, dN);
            Nmat[(n+1)*j+j-1] = dN[2];
            Nmat[(n+1)*j+j] = dN[3];
            break;
    case(1):
            Nmat[(n+1)*j+j] = 1;
            Nmat[(n+1)*j] = -1;
            j++;
            Nmat[(n+1)*j+j] = 1;
            Nmat[(n+1)*j+1] = -1;
            break;
    default:
            Nmat[(n+1)*j+j] = 1;
            Nmat[(n+1)*j+1] = -1;
            j++;
            Nmat[(n+1)*j+j] = 1;
            Nmat[(n+1)*j+2] = -1;
            }
  }
}
```

```
/*****************************************************************
  * Function Name: Solve.C
  * Author: Steven Fleming
  * Description: Matrix utility functions
  *****************************************************************/
#include<math.h>
#include<stdio.h>
void solve_lin_syst(int n, float Mat[], int *error, float xx[]);
/*=======================================================*
  * Function Name: solve_lin_syst
  *=======================================================*
  * Description: Solves a linear system of equations given an augmented
  * matrix of size n x n + 1
 * Input: number of equations, augmented matrix (n x n + 1)
 * Output: error flag (checks for unique solution)
 * column vector of soultion values
 *=======================================================*/
void solve_lin_syst(int n, float Mat[], int *error, float xx[])
{
 int    i, *nrow,  j, k, p, flag = 1, ncopy;
 float  *scale, *A,   factor, nextfact, sum, absA;
// float A[10000], scale[10000];
 //int nrow[1000];
 A = new float[(n+1)*n];
 for (i=0; i<n*(n+1); i++)
   A[i] = Mat[i];
 nrow = new int[n];
 scale = new float[n];
 for(i=0; i<n; i++)
   {
    scale[i] = 0;
    for (j=0; j<n; j++)
          {
           absA = fabs(A[(n+1)*i+j]);
           if (absA>scale[i])           scale[i] = absA;
          }
    if (scale[i] == 0)
          {flag = 0; i= n-1;}
    nrow[i] = i;
   }

 if (flag)
   {
    for (i=0; i<n-1; i++)
          {
           factor = 0;
           for (j=i; j<n; j++)
             {
              nextfact = fabs(A[(n+1)*nrow[j]+i])/scale[nrow[j]];
              if (nextfact > factor)
                    {
                     p = j;
                     factor = nextfact;
                    }
             }
```

```
            if (A[(n+1)*nrow[p]+i] ==0)
              {flag = 0; i = n-2;}

            if (flag)
              {
                if (nrow[i] != nrow[p])
                      {
                        ncopy = nrow[i];
                        nrow[i] = nrow[p];
                        nrow[p] = ncopy;
                      }

                for (j = i+1; j<n; j++)
                      {
                        /* check if factor is 0 */
                        if (A[(n+1)*nrow[j] +i] != 0)
                          {
                            factor = A[(n+1)*nrow[j]+i]/A[(n+1)*nrow[i]+i];
                            for (k=0; k<=n; k++)
                                  {
                                    A[(n+1)*nrow[j]+k] -= factor*A[(n+1)*nrow[i] +k];
                                  }
                          }
                      }
              }
          }

    if (flag)
          {
            if (A[(n+1)*nrow[n-1]+n-1] == 0) flag = 0;
            if (flag)
              {
                xx[n-1] = A[(n+1)*nrow[n-1]+n]/A[(n+1)*nrow[n-1]+n-1];
                for (i=n-2; i>=0; i--)
                      {
                        sum = 0;
                        for (j = i+1; j<n; j++)
                          {
                            sum += A[(n+1)*nrow[i] + j]*xx[j];
                          }

                        xx[i] = (A[(n+1)*nrow[i]+n]-sum)/A[(n+1)*nrow[i]+i];

                      }
                *error = 1;
              }
          }
  }

  if (flag == 0) *error = 0;

delete [] nrow;
delete [] scale;
delete [] A;
nrow = NULL;
scale = NULL;
```

```
  A = NULL;

}



/*************************************************************
  *        File Name: solve_systems.C
  *        Author: Steven Fleming
  *        Description: Contains routines to assemble the surface constraint matrix
  *        and solve multiple systems for the B-spline control points
  *************************************************************/
#include<stdio.h>
#include<math.h>
void setup_constraint_matrix(int numpts[],int numctrl[], float datpts[], int opcl[], int u_continuities[], int
                                    v_continuities[], float u_constraints[], float v_constraints[], float
                                    constraint_matrix[]);
void u_tan_pt_constraints(int row, int crow, int numpts, float datpts[], int numctrl,int opcl,int  continuities[],
                                  float constraints[], float constraint_matrix[]);
void v_tan_pt_constraints(int row, int crow, int numpts, float datpts[], int numctrl, int opcl,int continuities[],
                                  float constraints[], float constraint_matrix[]);
void solve_systems(int numctrl[], float Amat[], float Bmat[], float constraint_matrix[], float ctrlpts[]);
extern void solve_lin_syst(int n, float Mat[], int *error, float x[]);
#define OPEN 0
#define CLOSED 1

 /*=====================================================*
  *        Function: setup_constraint_matrix
  *=====================================================*
  *        Author: Steven Fleming
  *        Description: Driver for assembling B-spline surface constraint matrix
  *        (CS = Amat*Ctrl*Bmat )
  *        Inputs: number of data points, array of data points, no of control points
  *        open/closed flag, (uv) continuities, (uv) constraints
  *        Outputs: constraint matrix
  *=====================================================*/
void setup_constraint_matrix(int numpts[],int numctrl[], float datpts[], int opcl[], int u_continuities[], int
                      v_continuities[], float u_constraints[], float v_constraints[], float constraint_matrix[])
{
 int i,row,crow,column,continuity;
 /* load constraints for each u row */
 row = 1;
 crow = 0;
 u_tan_pt_constraints(row,crow,numpts[0],datpts,numctrl[0],opcl[0],u_continuities,
                           u_constraints,constraint_matrix);
 crow++;
 for (i=2;i<numpts[1];i++ )
  {
    continuity = v_continuities[i-1];
    row++;
    switch (continuity)
          {
    case(0):
    row++;
```

```
u_tan_pt_constraints(row,crow,numpts[0],datpts,numctrl[0],opcl[0],u_continuities,u_constraints,constraint_mat
rix);
      row++;
      crow++;
      break;

      case(1):
      row++;

u_tan_pt_constraints(row,crow,numpts[0],datpts,numctrl[0],opcl[0],u_continuities,u_constraints,constraint_mat
rix);
      crow++;
      break;

      default:
     u_tan_pt_constraints(row,crow,numpts[0],datpts,numctrl[0],opcl[0],
                                u_continuities,u_constraints,constraint_matrix);
      crow++;
            }
    }

  continuity = v_continuities[i-1];
  i+1
  if(opcl[1] == OPEN)
    {
      row++;
      u_tan_pt_constraints(row,crow,numpts[0],datpts,numctrl[0],opcl[0],
                                 u_continuities,u_constraints,constraint_matrix);
    }

  /* load constraints for each v column */
  column =1;
  crow = 0;
  v_tan_pt_constraints(column,crow,numpts[1],datpts,numctrl[0],opcl[1],v_continuities,
                              v_constraints,constraint_matrix);
  crow++;
  for (i=2;i<numpts[0];i++ )
    {
      continuity = u_continuities[i-1];
      column++;
      switch (continuity)
            {
      case(0):
      column++;
      v_tan_pt_constraints(column,crow,numpts[1],datpts,numctrl[0],opcl[1],
                                    v_continuities,v_constraints,constraint_matrix);
      column++;
      crow++;
      break;

      case(1):
      column++;
      v_tan_pt_constraints(column,crow,numpts[1],datpts,numctrl[0],opcl[1],
                                    v_continuities,v_constraints,constraint_matrix);
      crow++;
```

```
    break;

    default:
    v_tan_pt_constraints(column,crow,numpts[1],datpts,numctrl[0],opcl[1],
                                    v_continuities,v_constraints,constraint_matrix);
    crow++;
          }
   }
  continuity = u_continuities[i-1];
  if (opcl[0] == OPEN)
   {
    column++;
    v_tan_pt_constraints(column,crow,numpts[1],datpts,numctrl[0],opcl[1],v_continuities,
                                    v_constraints,constraint_matrix);
   }

}

  /*=====================================================*
   *        Function Name: u_tan_pt_constraints
   *        Author: Steven Fleming
   *        Description: Adds constraints to a particular row (u direct) of the constraint matrix
   *
   *        Inputs: row index, constant row index, number of data pts in u direction, array of
   *        datapts, number of ctrlpts in u direct, open/closed flag in direction, continuities and
   *        constraints in u direction, constraint matrix
   *
   *        Outputs: updated constraint matrix
   *=====================================================*/
void u_tan_pt_constraints( int row, int crow, int numpts, float datpts[], int numctrl, int opcl,int continuities[],
                                    float constraints[], float constraint_matrix[])

  /* int row,crow,numpts,numctrl,opcl,continuities[];
   float datpts[],constraints[],constraint_matrix[];*/
{
  int continuity,i,j,k,l,m,n;
  int nconst,ndat;
  n=numctrl;
  ndat=numpts;
  nconst=0;

  for ( i=0; i<numpts;i++ )
   {
    switch(continuities[i])
          {
      case(0):
      nconst +=2;
      break;

      case(1):
      nconst +=1;
      break;
          }
   }

  for ( i=0;i<3;i++ )
```

```
{
 continuity = continuities[0];
 switch(continuity)
        {
 case(0):
 constraint_matrix[ 3*n*row + i] = constraints[3*nconst*crow+3+i];
 k=2;
 break;
 case(1):
 constraint_matrix[ 3*n*row + i] = constraints[3*nconst*crow+i];
 k=1;
 break;
 default:

 constraint_matrix[ 3*n*row + i] = 0.0;
 k=0;
        }
 constraint_matrix[ 3*n*row+3 + i] = datpts[3*ndat*crow+i];

 l=1;
 m=2;
 for (j=1;j<numpts-1;j++ )
        {
          continuity = continuities[j];
          switch (continuity)
            {
         case(0):
          constraint_matrix[ 3*n*row+3*m + i] = constraints[3*nconst*crow+3*k+i];
          m++;
          k++;
          constraint_matrix[ 3*n*row+3*m + i] = datpts[3*ndat*crow+3*l+i];
          m++;
          l++;
          constraint_matrix[ 3*n*row+3*m + i] = constraints[3*nconst*crow+3*k+i];
          m++;
          k++;
          break;
         case(1):
          constraint_matrix[ 3*n*row+3*m + i] = constraints[3*nconst*crow+3*k+i];
          m++;
          k++;
          constraint_matrix[ 3*n*row+3*m + i] = datpts[3*ndat*crow+3*l+i];
   m++;
        l++;
   break;

       default:
   constraint_matrix[ 3*n*row+3*m + i] = datpts[3*ndat*crow+3*l+i];
   m++;
        l++;
           }
        }

 if (opcl == OPEN )
        {
          continuity = continuities[j];
```

```
            switch (continuity)
              {
            case(0):
            case(1):
            constraint_matrix[ 3*n*row+3*m + i] = datpts[3*ndat*crow+3*l+i];
            m++;
            constraint_matrix[ 3*n*row+3*m + i] = constraints[3*nconst*crow+3*k+i];
            break;

            default:
            constraint_matrix[ 3*n*row+3*m + i] = datpts[3*ndat*crow+3*l+i];
            m++;
            constraint_matrix[ 3*n*row+3*m + i] = 0.0;
              }
           }
    else
           {
            continuity = continuities[0];
            switch (continuity)
              {
            case(0):
            constraint_matrix[ 3*n*row+3*m + i] = 0.0;
            m++;
            constraint_matrix[ 3*n*row+3*m + i] = constraints[3*nconst*crow+i];
            break;
            default:
            constraint_matrix[ 3*n*row+3*m + i] = 0.0;
            m++;
            constraint_matrix[ 3*n*row+3*m + i] = 0.0;
              }
           }
  }
}
 /*=====================================================*
  *        Function: v_tan_constraints
  *=====================================================*
  *        Description: Adds constraints to a particular column (v direction) of the constraint
  *        matrix
  *
  *        Inputs: column index, constant row index, number of data pts in v direction, array
  *        of datapts, number of ctrlpts in v directiom, open/closed flag in v direction,
  *        continuities and constraints in v direction, constraint matrix
  *
  *        Outputs: updated constraint matrix
  *=====================================================*/
void v_tan_pt_constraints(int col, int crow, int numpts, float datpts[], int numctrl,int opcl,int continuities[], float
                          constraints[], float constraint_matrix[])
{
 int continuity,i,j,k,l,m,n;
 int nconst,ndat;

 ndat=numpts;
 n=numctrl;
 nconst=0;

 for ( i=0; i<numpts;i++ )
```

```
 {
  switch(continuities[i])
        {
  case(0):
  nconst +=2;
  break;
  case(1):
  nconst +=1;
  break;
        }
 }

for ( i=0;i<3;i++ )
 {
  continuity = continuities[0];
  switch(continuity)
        {
  case(0):
  constraint_matrix[ 3*col + i] = constraints[3*nconst*crow +3 + i];
  k=2;
  break;
  case(1):
  constraint_matrix[ 3*col + i] = constraints[3*nconst*crow + i];
  k=1;
  break;

  default:
  constraint_matrix[ 3*col + i] = 0.0;
  k=0;
        }

  l=1;
  m=2;
  for (j=1;j<ndat-1;j++ )
            {
              continuity = continuities[j];
              switch (continuity)
                    {
              case(0):
                    constraint_matrix[ 3*n*m + 3*col + i] = constraints[3*nconst*crow +3*k +i];
              m++;
              k++;
              m++;
              l++;
              constraint_matrix[ 3*n*m + 3*col + i] = constraints[3*nconst*crow + 3*k +i];
              m++;
              k++;
              break;
              case(1):
              constraint_matrix[ 3*n*m + 3*col + i] = constraints[3*nconst*crow + 3*k+i];
              m++;
              k++;
              m++;
              l++;
              break;
              default:
```

```
                m++;
                l++;
                    }
                }

    if (opcl == OPEN )
        {
         continuity = continuities[j];
         switch (continuity)
          {
         case(0):
         case(1):
         m++;
         constraint_matrix[ 3*n*m + 3*col + i] = constraints[3*nconst*crow + 3*k+i];
         break;
         default:
          m++;
         constraint_matrix[ 3*n*m + 3*col + i] = 0.0;
          }
        }
    else
        {
         continuity = continuities[0];
         switch (continuity)
          {
         case(0):
          constraint_matrix[ 3*n*m + 3*col + i] = 0.0;
          m++;
          constraint_matrix[ 3*n*m + 3*col + i] = constraints[3*nconst*crow + i];
         break;
         default:
          constraint_matrix[ 3*n*m + 3*col + i] = 0.0;
          m++;
          constraint_matrix[ 3*n*m + 3*col + i] = 0.0;
          }
        }
   }
}

 /*=====================================================*
  *        Function solve_systems
  *=====================================================*
  *        Description: Solves the multiple linear systems for the B-spline surface control net
  *        Inputs: number of control points, Amat,Bmat, constraint matrix
  *        Outputs: control point array
  *=====================================================*/
void solve_systems(int numctrl[], float Amat[], float Bmat[],float constraint_matrix[], float ctrlpts[])
{
 int i,j,k,n,m,x,y,error,f,g;
 float *C_points;
 float *D_points;
 n = numctrl[0];
 m = numctrl[1];
 D_points = new float[n];
 C_points = new float[m];
```

```
 for (i=0;i<m;i++)
  {
   for ( j=0;j<3;j++ )
          {
           for ( k=0;k<n;k++ )
            Bmat[k*(n+1)+n] = constraint_matrix[ 3*n*i + 3*k + j];
           solve_lin_syst(n,Bmat,&error,D_points);
           for ( k=0; k<n; k++ )
            constraint_matrix[ 3*n*i + 3*k + j] = D_points[k];
          }
  }

 for ( i=0;i<n;i++ )
  {
   for ( j=0;j<3;j++ )
          {
           for ( k=0;k<m;k++)
            Amat[k*(m+1)+m] = constraint_matrix[ 3*n*k + 3*i + j];

           solve_lin_syst(m,Amat,&error,C_points);

           for ( k=0;k<m;k++)
            {
             ctrlpts[3*n*k + 3*i + j] = C_points[k];
            }
          }
  }
delete []C_points;
C_points = NULL;
delete []D_points;
D_points = NULL;
}



/************************************************************
 *          Function Name: sur_knots.C
 *          Author: Steven Fleming
 *          Description: Calculates the u & v knot vectors for a non-uniform B-spline surface
 *          Note: Adapted from code by Robert Jones
 ************************************************************/
#include <math.h>
#define OPEN 0
#define CLOSED 1
void average_parametrization(int numpts[], float datpts[],int opcl[],int param,float u_knots[],float v_knots[]);
void calc_u_knots(int numpts[],float datpts[],int opcl[],int param,float u_knots[]);
void calc_v_knots(int numpts[],float datpts[],int opcl[],int param,float v_knots[]);
float average_u_distance(int numpts[], float datpts[], int n1, int n2);
float average_v_distance(int numpts[], float datpts[], int n1, int n2);
float point_distance(float first[] ,float last[]);
 /*=====================================================*
  *          Function Name: average_parametrization
  *=====================================================*
  *          Description: driver to calculate u & v surface knots based on chord length parametrization
  *          Inputs:
  *         numpts[]: array of number of data points in both parametric directions,
```

```
   *         datpts[]: array of data pts,
   *        opcl[]: array of open/closed flags (u,v)
   *         Outputs: u_knots,v_knots
   *=====================================================*/
void average_parametrization(int numpts[], float datpts[],int opcl[],int param,float u_knots[],float v_knots[])
/* Calculate surface's knots in parametric u and v direction;

INPUT:
numpts[]: [0] and [1] number of data points in each direction;
datpts[]: array of data points;
opcl[]:  open or closed curve
param:  type of parameterization

OUTPUT:
u_knots[]: array of u knots;
v_knots[]: array of v knots;*/
{
  calc_u_knots(numpts, datpts, opcl, param, u_knots);
  calc_v_knots(numpts, datpts, opcl, param, v_knots);
}
  /*=====================================================*
   *         Module Name: calc_u_knots
   *=====================================================*
   *         Description: calculates knots in parametric u direction for cubic NUBS surface;
   *
   *         Inputs: same as above
   *
   *         Outputs:
        u_knots[]: array of knots in u direction;
   *
   *=====================================================*/
void calc_u_knots(int numpts[],float datpts[],int opcl[],int param,float u_knots[])
{
  int npts,i;
  float ratio;          /* ratio of knot spacing to pt distance */
  float distance;     /* knot distance */

  /* --- initialize knot spacing --- */
  u_knots[3] = 0.0;
  u_knots[4] = 1.0;

  /* perform correct type of parametrization */
  switch(param)
    {
      /* uniform parametrization */
  case(1):
    for ( i = 5; i<= numpts[0]+2;i++ )
      u_knots[i] = u_knots[i-1] + 1;
    break;

    /* chord length parametrization */
  case(2):
    ratio = average_u_distance(numpts,datpts,0,1);
    /* calculate interior knots */
    for ( i = 5; i <= numpts[0] + 2;i++ )
      {
```

```
            u_knots[i] = average_u_distance(numpts,datpts,i-4,i-3)/ratio + u_knots[i-1];
    }
break;

/* centripetal parametrization */
case 3:
  ratio = sqrt(average_u_distance(numpts,datpts,1,2));
  /* calculate interior knots */
  for (i=5; i<=numpts[0]+2; i++)
    {
            u_knots[i] = sqrt(average_u_distance(numpts,datpts,i-3,i-2))/ratio + u_knots[i-1];
    }
break;

default:
  for ( i=5; i<=numpts[0]+2;i++ )
      u_knots[i] = u_knots[i-1] + 1;
break;

}

/* --- Calculate End Knots based on open or closed curves --- */
 npts = numpts[0];
 if (opcl[0] == OPEN )                  /* open curve */
    {
            distance = u_knots[3] - u_knots[4];
            u_knots[2] = u_knots[3] + distance;  // the distance may be minus
            u_knots[1] = u_knots[2] + distance;
            u_knots[0] = u_knots[1] + distance;

            distance = u_knots[npts+2] - u_knots[npts+1];
            u_knots[npts+3] = u_knots[npts+2] + distance;
            u_knots[npts+4] = u_knots[npts+3] + distance;
            u_knots[npts+5] = u_knots[npts+4] + distance;

    }
 else                   /* closed curve */

    {
            u_knots[2] = u_knots[3] + ( u_knots[npts+1] - u_knots[npts+2]);
      u_knots[1] = u_knots[2] + ( u_knots[npts] - u_knots[npts+1]);
      u_knots[0] = u_knots[1] + ( u_knots[npts-1] - u_knots[npts]);

      u_knots[npts+3] = u_knots[npts+2] + ( u_knots[4] - u_knots[3]);
      u_knots[npts+4] = u_knots[npts+3] + ( u_knots[5] - u_knots[4]);
      u_knots[npts+5] = u_knots[npts+4] + ( u_knots[6] - u_knots[5]);
    }
}
/*=================================================*
 *        Module Name: calc_v_knots
 *=================================================*
 *
 *        Description: calculates the chord length v-knots for a length v-knots for a NURB surface
 *
 *        Inputs: array of number of points in each direction, array of data pts, array of
 *        open/closed flags, type of parametrization
```

```
   *
   *         Outputs: v_knots
   *=================================================*/
void calc_v_knots(int numpts[],float datpts[],int opcl[],int param,float v_knots[])
{
  int npts,i;  /* no of pts per c/s , for loop values */
  float ratio;          /* ratio of knot spacing to pt distance */
  float distance;    /* knot distance */

/* --- initialize knot spacing --- */
v_knots[3] = 0.0;
v_knots[4] = 1.0;

/* perform correct type of parametrization */
switch(param)
{
  /* uniform parametrization */
  case(1):
    for (i=5;i<=numpts[1]+2;i++)
      v_knots[i] = v_knots[i-1] + 1;
    break;

    /* chord length parametrization */
    case(2):
      ratio = average_v_distance(numpts,datpts,0,1);
      /* calculate interior knots */
      for (i=5;i<=numpts[1]+2;i++)
            {
              v_knots[i] = average_v_distance(numpts,datpts,i-4,i-3)/ratio + v_knots[i-1];
            }
      break;

      /* centripetal parametrization */
  case 3:
    ratio = sqrt(average_v_distance(numpts,datpts,1,2));
    /* calculate interior knots */
    for (i=5; i<=numpts[1]+2; i++)
      {
            v_knots[i] = sqrt(average_v_distance(numpts,datpts,i-3,i-2))/ratio + v_knots[i-1];
      }
  break;

default:
  for ( i=5; i<=numpts[1]+2;i++ )
     v_knots[i] = v_knots[i-1] + 1;
  break;

}

/* --- Calculate End Knots based on open or closed curves --- */
npts = numpts[1];
if (opcl[1] == OPEN )                    /* open curve */
  {
   distance = v_knots[3] - v_knots[4];
   v_knots[2] = v_knots[3] + distance;
   v_knots[1] = v_knots[2] + distance;
```

```
   v_knots[0] = v_knots[1] + distance;

   distance = v_knots[npts+2] - v_knots[npts+1];
   v_knots[npts+3] = v_knots[npts+2] + distance;
   v_knots[npts+4] = v_knots[npts+3] + distance;
   v_knots[npts+5] = v_knots[npts+4] + distance;

  }
 else                    /* closed curve */
 {
  v_knots[2] = v_knots[3] + ( v_knots[npts+1] - v_knots[npts+2]);
  v_knots[1] = v_knots[2] + ( v_knots[npts] - v_knots[npts+1]);
  v_knots[0] = v_knots[1] + ( v_knots[npts-1] - v_knots[npts]);

  v_knots[npts+3] = v_knots[npts+2] + ( v_knots[4] - v_knots[3]);
  v_knots[npts+4] = v_knots[npts+3] + ( v_knots[5] - v_knots[4]);
  v_knots[npts+5] = v_knots[npts+4] + ( v_knots[6] - v_knots[5]);
 }

 }


  /*=====================================================*
   *         Module Name: average_u_distance
   *=====================================================*
   *
   *         Description: computes the average distance between two u values
   *
   *         Inputs:
   *    numpts[]:  number of points in u and v direction;
   *    datpts[]:  array of data points;
   *    n1:        first point in u direction;
   *    n2:        second point in u direction;
   *         point sequence numbers in u direction)
   *         Outputs:
   *    Average distance between data points in u direction;
   *=====================================================*/
 float average_u_distance(int numpts[], float datpts[], int n1, int n2)
 {
  int i;
  float distance, sum, average;
  float first[3],last[3];
  int count;           /* number of distances */

  sum = count = 0;             /* initialize sum */
  for ( i = 0; i < numpts[1]; i++ )
    {
     first[0] = datpts[3*numpts[0]*i+3*n1];
     first[1] = datpts[3*numpts[0]*i+3*n1+1];
     first[2] = datpts[3*numpts[0]*i+3*n1+2];

     last[0] = datpts[3*numpts[0]*i+3*n2];
     last[1] = datpts[3*numpts[0]*i+3*n2+1];
     last[2] = datpts[3*numpts[0]*i+3*n2+2];

     distance = point_distance(first,last);        /* calculate distance */
```

```
    count++;
    sum = sum + distance;                    /* sum up distances */
   }

 average = sum/count;
 return(average);
}

/*=====================================================*
  *        Module Name: average_v_distance
  *=====================================================*
  *        Description: Computes the average distance between the two v values
  *
  *        Inputs: array of number of points in u and v, array of data points
  *        n1,n2 (data pt sequence numbers in v direction )
  *
  *        Outputs: Average distance between data pts in v direction
  *=====================================================*/
float average_v_distance(int numpts[], float datpts[], int n1, int n2)
{
 int i;
 float distance, sum, average;
 float first[3],last[3];
 int count;          /* number of distances */
 sum = count = 0;            /* initialize sum */
 for ( i = 0; i < numpts[0]; i++ )
   {
    first[0] = datpts[3*numpts[0]*n1+3*i];
    first[1] = datpts[3*numpts[0]*n1+3*i+1];
    first[2] = datpts[3*numpts[0]*n1+3*i+2];

    last[0] = datpts[3*numpts[0]*n2+3*i];
    last[1] = datpts[3*numpts[0]*n2+3*i+1];
    last[2] = datpts[3*numpts[0]*n2+3*i+2];

    distance = point_distance(first,last);       /* calculate distance */
    count++;
  sum = sum + distance;                     /* sum up distances */
   }
 average = sum/count;
 return(average);
}
/*=====================================================*
  *        Module Name: point_distance
  *=====================================================*
  *        Description: Calculates the actual distance between the two 3-d points
  *        Inputs: point arrays for first and last points
  *        Outputs: distance between points
  *=====================================================*/
float point_distance(float first[] ,float last[])
{
 float distance;
 distance = sqrt((last[0] - first[0]) * (last[0] - first[0]) + (last[1] - first[1]) * (last[1] - first[1]) + (last[2] - first[2])
                 *  (last[2] - first[2]));
 return(distance);
}
```

```
 /*****************************************************************
  *        Function Name:   volume_grid.C
  *        Author:  Roberto Rojas
  *        Description:  Creates a grid of datapoints to be later used to create differential volumes
  *        Note:     The higher the grid, the better the accuracy
  *****************************************************************
  *        Input:    num - u direction number of points
  *                  trimnum - number of points defining trimming curve, also v direction
  *                  number of "new data points"
  *                  u and v new_knots so that basis can be used to compute N ;  control x & y
  *                  to be used with N to map points on the original surface using map_point
  *                  Array trim_data, original surface data points
  *
  *        Output:  Array of trimmed surface points
  *****************************************************************/
#include<math.h>
extern void basis(float, int, float*, float[]);
extern void map_point(int, float[], float[], float*, int, int, float*);
void get_orig_points(int num, int trimnum, int grid_u, int grid_v,float *trim_data, float *u_newknots, float
                          *v_newknots, float *control_x, float *control_y, float *control_z, float *points);
void get_orig_points1(int num, int num1, int trimnum, int grid_u, int grid_v,float *trim_data,
                          float *u_newknots, float *v_newknots, float *control_x,
                          float *control_y, float *control_z, float *points);
void get_trim_points(int num, int trimnum, int grid_u, int grid_v, float *u_newknots,  float *v_newknots,
                           float  *control_x, float *control_y, float *control_z, float *points);
/* Function description
   Creates a grid of data points on the original surface to be later on used to create volume
   differentials between the two surfaces;
   num:      surface's number of points in the parametric u direction;
   trimnum:  number of points representing the trimming curve;
   int grid_u:  refinement in u direction;
   int grid_v:  refinement in v direction;
   u_newknots:  array of knots in parametric u direction;
   v_newknots:
   control_x, control_y, control_z: array of control points;
   trim_data: data value in parameter space,  representing the trimming curve.

  output:
  points: array of points from the original surface;
*/
void get_orig_points(int num, int trimnum, int grid_u, int grid_v, float *trim_data, float *u_newknots, float
                          *v_newknots, float *control_x, float *control_y, float *control_z, float *points)
{
  int i,j,k,l,intU,intV,variable1,variable2, flag;
  float u1,u2,u,v;
  float x,y,z;
  float Nu[4],Nv[4];

  for (j=0;j<grid_v;j++)
    for (i=0;i<grid_u;i++)
      {
            variable1=2*floor(j*(trimnum-1)/(grid_v-1));
```

```
        variable2=2*ceil(j*(trimnum-1)/(grid_v-1));

        u=trim_data[variable1]*i/(grid_u-1);
        v=trim_data[2*(trimnum-1)+1]*j/(grid_v-1);

        flag = 0;
        for (k=0; k<num+2; k++)
           {
            if(u>=u_newknots[k] && u<u_newknots[k+1])
                  {
                   intU = k;
                   flag = 1;
                  }
            if(flag)
                  break;
           }

        flag = 0;
        for (l=0; l<trimnum+2; l++)
         {
           if(v>=v_newknots[l] && v<v_newknots[l+1])
             {
                  intV = l;
                  flag = 1;
             }
           if(flag)
             break;
         }

        basis( u, intU, u_newknots, Nu);
        basis( v, intV, v_newknots, Nv);

        /* num + 2 to account for the double knots at the ends */
        map_point(num+2,Nu,Nv,control_x,intU,intV,&x);
        map_point(num+2,Nu,Nv,control_y,intU,intV,&y);
        map_point(num+2,Nu,Nv,control_z,intU,intV,&z);

        points[3*grid_u*j + 3*i + 0] = x;
        points[3*grid_u*j + 3*i + 1] = y;
        points[3*grid_u*j + 3*i + 2] = z;

    }

}


/* Function definition          */
void get_orig_points1(int num, int num1, int trimnum, int grid_u, int grid_v, float *trim_data,
                float *u_newknots, float *v_newknots, float *control_x,
                float *control_y, float *control_z, float *points)
{
  int i,j,k,l,intU,intV,variable1,variable2, flag;
  float u1,u2,u,v;
  float x,y,z;
  float Nu[4],Nv[4];
```

```
 for (j=0;j<grid_v;j++)
  for (i=0;i<grid_u;i++)
    {
          variable1=2*floor(j*(trimnum-1)/(grid_v-1));
          variable2=2*ceil(j*(trimnum-1)/(grid_v-1));

          u=trim_data[variable1]*i/(grid_u-1);
          v=trim_data[2*(trimnum-1)+1]*j/(grid_v-1);

          flag = 0;
          for (k=0; k<num+2; k++)
             {
              if(u>=u_newknots[k] && u<u_newknots[k+1])
                    {
                     intU = k;
                     flag = 1;
                    }
              if(flag)
                    break;
             }

          flag = 0;
          for (l=0; l<trimnum+2; l++)
           {
            if(v>=v_newknots[l] && v<v_newknots[l+1])
              {
                    intV = l;
                    flag = 1;
              }
            if(flag)
              break;
           }

          basis( u, intU, u_newknots, Nu);
          basis( v, intV, v_newknots, Nv);

          /* num + 2 to account for the double knots at the ends */
          map_point(num1+2,Nu,Nv,control_x,intU,intV,&x);
          map_point(num1+2,Nu,Nv,control_y,intU,intV,&y);
          map_point(num1+2,Nu,Nv,control_z,intU,intV,&z);

          points[3*grid_u*j + 3*i + 0] = x;
          points[3*grid_u*j + 3*i + 1] = y;
          points[3*grid_u*j + 3*i + 2] = z;
    }
}

/* create a grid of data points on the new surface to be later used to create
   volume differential between the two surfaces;

  OUTPUT: array of points from the trimmed surfaces;
*/
void get_trim_points(int num, int trimnum, int grid_u, int grid_v, float *u_newknots,  float *v_newknots, float
                          *control_x, float *control_y, float *control_z,  float *points)
{
  int i,j,k,l,flag;
```

```
  int intU, intV;
  float u,v;
  float x, y, z;
  float Nu[4], Nv[4];

  for(j=0;j<grid_v;j++)
    {
      for (i=0;i<grid_u;i++)
            {
              u=u_newknots[num + 2]*i/(grid_u-1);
              v=v_newknots[trimnum + 2]*j/(grid_v-1);

              flag = 0;
              for (k=0; k<num+2; k++)
                {
                  if(u>=u_newknots[k] && u<u_newknots[k+1])
                        {
                          intU = k;
                          flag = 1;
                        }
                  if(flag)
                        break;
                }

              flag = 0;
              for (l=0; l<trimnum+2; l++)
                {
                  if(v>=v_newknots[l] && v<v_newknots[l+1])
                        {
                          intV = l;
                          flag = 1;
                        }
                  if(flag)
                        break;
                }
              basis(u, intU, u_newknots, Nu);
              basis(v, intV, v_newknots, Nv);
              /*num +2 to account for double knots at the ends */
              map_point(num+2, Nu, Nv, control_x, intU, intV, &x);
              map_point(num+2, Nu, Nv, control_y, intU, intV, &y);
              map_point(num+2, Nu, Nv, control_z, intU, intV, &z);
              points[3*grid_u*j + 3*i + 0] = x;
              points[3*grid_u*j + 3*i + 1] = y;
              points[3*grid_u*j + 3*i + 2] = z;
            }
    }
}
```

Vita

Xijun Wang

The author was born on March 15, 1969, in China. In 1988, he began his undergraduate studies in Mechanical Engineering at Shanghai Jiao Tong University. After graduation, the author worked for Tsingtao Brewery Company for six years, then went to America to continue his graduate studies in Mechanical Engineering in the fall of 1998. After two terms in the University of Arkansas, he transferred to Virginia Tech, and finished his Master's degree in the spring of 2001.

Xijun plans to go to Richmond to join his family and start a career there.