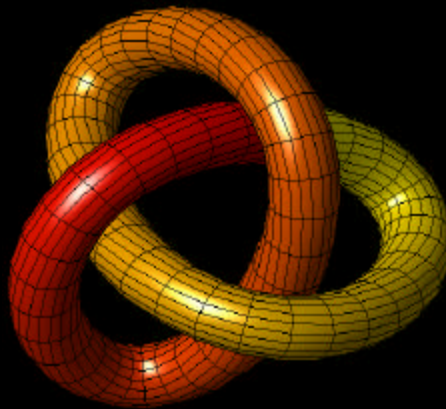
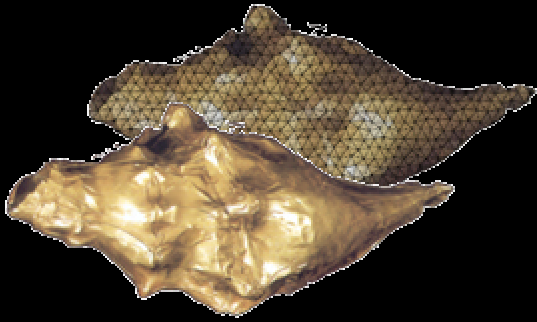

UNDERLYING TOPICS ON CAD / CAM / CG



Dr. Oscar E. Ruiz S.
CAD/CAM/CG Laboratory - EAFIT University - Medellín, Colombia
Copyright A.S.M.E. (American Society of Mechanical Engineers)

TABLE OF CONTENTS

INTRODUCTION	2
ACKNOWLEDGMENTS.	2
1. BASIC CONCEPTS OF PROGRAMMING	4
1.1 ALGORITHMS	4
1.2 PSEUDO LANGUAGE	4
1.2.1 ASSERTION (PRE, POST, INV)	5
1.2.2 PRECONDITION AND POSTCONDITION	5
1.3 EXECUTABLE INSTRUCTIONS	5
1.3.1 ASSIGNMENT, VERIFICATION OR COMPARISON	5
1.3.2 SELECTION AND DECISION MAKING	6
1.4 ITERATIVE COMMANDS OR CYCLES	7
1.5 PROGRAMMING CRITERIA	8
1.5.1 STRUCTURE	8
1.5.2 NAME OF FUNCTIONS AND VARIABLES	9
1.5.3 VALIDATION OF DATA	10
1.5.4 ERRORS	10
1.5.5 ASSOCIATION OF OPERATIONS	10
1.5.6 INDENTATION	11
1.6 EXERCISES - BASIC CONCEPTS OF PROGRAMMING.	12
1.6.1 OBTAINING INFORMATION FROM MATLAB.	12
1.6.2 SCALARS, VECTORS AND MATRICES OPERATIONS.	13
1.6.3 USER INTERFACE COMMANDS INTRODUCTION.	14
1.6.4 2D PLOTTING.	15
1.6.5 3D PLOTTING.	16
1.6.6 DATA INPUT.	17
1.6.7 GRAPHIC OPTIONS.	18
1.6.8 MENU HANDLING.	19
1.6.9 SUMMATION OF VALUES.	20
1.6.10 AVERAGE VALUES.	21
1.6.11 SORTING.	22
1.6.12 MIN MAX.	23
1.6.13 FUNCTION PROGRAMMING	24
2. BASIC CONCEPTS ON LINEAR ALGEBRA	25
2.1 VECTORS AND POINTS	25
2.1.1 OPERATIONS WITH VECTORS	25
2.1.1.1 Addition and Subtraction	25
2.1.1.2 Product	26
2.1.1.2.1 Scalar, inner or Dot Product (\cdot)	26
2.1.1.2.2 Cross Product (\times)	27

2.2	MATRICES	28
2.2.1	PROPERTIES OF MATRICES	28
2.2.2	OPERATIONS WITH MATRICES	28
2.2.2.1	Addition and subtraction	28
2.2.2.2	Neutral element	29
2.2.2.3	Scalar product	29
2.2.2.4	Multiplication of matrices	29
2.2.2.5	Identity matrix	29
2.2.2.6	Transposed matrix	30
2.2.2.6.1	Symmetric matrix	30
2.3	HOMOGENEOUS COORDINATES	30
2.3.1	POINT	31
2.3.2	VECTORS	31
2.3.3	MATRICES	31
2.4	EIGENVALUES AND EIGENVECTORS	31
2.5	NORMS OF VECTORS AND MATRICES	32
3.	GEOMETRIC TRANSFORMATIONS	33
3.1	DEFINITIONS OF MATHEMATICAL ENTITIES	33
3.1.1	CANONICAL RIGHT HANDED OR DEXTEROUS COORDINATE FRAME	33
3.1.2	GEOMETRIC TRANSFORMATIONS AND MATRIX NOTATION	34
3.2	RIGID TRANSFORMATIONS ABOUT WORLD COORDINATE AXES	34
3.2.1	TRANSLATION	34
3.2.2	ROTATIONS ABOUT MAIN AXES.	35
3.2.3	ROTATIONS PARALLEL TO WORLD AXES.	36
3.2.4	RIGIDITY OF TRANSFORMATION VS CANONICAL RIGHT HANDED SYSTEMS.	37
3.2.5	THE NON-COMMUTATIVE GROUP OF GEOMETRIC TRANSFORMATIONS	38
3.2.6	EXAMPLE. RIGID TRANSFORMATIONS.	39
3.3	RIGID TRANSFORMATIONS ABOUT ARBITRARY DIRECTIONS. QUATERNION METHOD	41
3.3.1	ROTATIONS ABOUT ARBITRARY AXES PIVOTED IN THE ORIGIN.	41
3.3.2	IDENTIFICATION OF AXIS AND ANGLE OF ORIGIN-BASED ROTATIONS. EIGENVALUE METHOD.	42
3.3.3	ROTATIONS ABOUT ARBITRARY AXES PIVOTED OUTSIDE THE ORIGIN.	42
3.3.4	IDENTIFICATION OF AXIS AND ANGLE OF ARBITRARY ROTATIONS. EIGENVALUE METHOD.	43
3.3.5	EXAMPLE. GENERAL RIGID TRANSFORMATIONS.	44
3.4	NON RIGID TRANSFORMATIONS	44
3.4.1	MIRROR OR REFLECTION	44
3.4.1.1	Mirror about a point	45
3.4.1.1.1	Mirror about the origin	45
3.4.1.1.2	Mirror about a point different from the origin	45
3.4.1.2	Mirror about a plane.	46
3.4.1.2.1	Householder transformation	46
3.4.1.2.2	Example. Mirror on the YZ plane.	46
3.4.1.2.3	Mirror through an arbitrary plane.	47
3.4.2	SCALING	48
3.4.2.1	Scaling with respect to the origin.	48
3.4.2.2	Scaling with respect to an arbitrary point.	49
3.4.3	SHEAR	49

3.4.3.1	Shear with respect to an arbitrary point.	50
3.5	EXERCISES – GEOMETRIC TRANSFORMATIONS –	51
3.5.1	ROTATIONS AROUND THE MAIN AXES.	51
3.5.2	ITERATIVE ROTATIONS I.	52
3.5.3	ITERATIVE ROTATIONS II.	53
3.5.4	MIRROR ABOUT THE ORIGIN.	54
3.5.5	MIRROR ABOUT XY PLANE.	54
3.5.6	MIRROR ABOUT X=Y PLANE.	54
3.5.7	MIRROR ABOUT Z AXIS.	54
3.5.8	SCALING.	54
3.5.9	SHEARING.	54
3.5.10	ITERATIVE ROTATIONS III.	55
3.5.11	ITERATIVE ROTATIONS IV.	56
3.5.12	SOLID GENERATION.	58
3.5.13	TRANSFORMATION OF A SOLID BODY I.	59
3.5.14	TRANSFORMATIONS OF A SOLID BODY II.	60
3.5.15	MIRROR ABOUT XZ PLANE.	61
3.5.16	GENERAL TRANSFORMATIONS I.	62
3.5.17	GENERAL TRANSFORMATIONS II.	63
3.5.18	SWEEP OF A CROSS SECTION ALONG AN ARBITRARY PATH.	64
3.5.19	SIMULTANEOUS SWEEP AND TWIST OF A CROSS SECTION.	65
3.5.20	PARALLEL PROJECTION.	67
3.5.21	PERSPECTIVE PROJECTION.	68
4.	CURVES AND SURFACES	70
4.1	RANDOM SAMPLES	70
4.1.1	POLYNOMIAL INTERPOLATION	70
4.1.2	STATISTIC INTERPOLATION	71
4.2	ORDERED SAMPLES. PARAMETRIC EQUATIONS	72
4.2.1	PARAMETRIC CURVES	74
4.2.1.1	Bezier Curve	76
4.2.1.1.1	Scalar form of the Bezier coefficients.	76
4.2.1.1.2	Matrix form of the Bezier coefficients.	76
4.2.1.2	Uniform B-Spline Curve	78
4.2.1.2.1	Continuity analysis for Spline curves	80
4.2.1.2.2	Closed Spline curves	80
4.2.2	SURFACES	82
4.2.2.1	Control point sets	82
4.2.2.1.1	Example of control Polyhedron generation (The Möbius Band)	84
4.2.2.2	Parametric Surfaces	86
4.2.2.3	Bezier Surface	86
4.2.2.4	Spline Surface	87
4.3	EXERCICES - CURVES AND SURFACES -	90
4.3.1	CONTROL POINT GENERATION OF A TOROIDAL SPIRAL.	90
4.3.2	BEZIER INTERPOLATION FUNCTION.	91
4.3.3	SELF-DEFINED INTERPOLATION FUNCTION.	93
4.3.4	GENERAL CURVE INTERPOLATION FUNCTION.	94
4.3.5	CONTROL POINT GENERATION OF A TOROIDAL SURFACE.	95

4.3.6	CONTROL POINT GENERATION OF A CONICAL SPIRAL BAND.	96
4.3.7	BEZIER SURFACE I.	97
4.3.8	BEZIER SURFACE II.	99
4.3.9	GENERIC SURFACE INTERPOLATION FUNCTION.	101
4.3.10	BEZIER AND SPLINE SURFACE INTERPOLATION.	102
5.	GEOMETRIC MODELING	103
5.1	CONSTRUCTIVE SOLID GEOMETRY (CSG)	104
5.2	BOUNDARY REPRESENTATION (B-REP)	106
5.2.1	EXAMPLE. RELATIONSHIP BETWEEN TOPOLOGY AND GEOMETRY	111
5.3	EXAMPLE 1. DEFINITION OF BOUNDARY REPRESENTATION	112
5.3.1	BOUNDARY REPRESENTATION	112
5.3.2	CSGREPRESENTATION	113
5.4	EXAMPLE 2. DEFINITION OF BOUNDARY REPRESENTATION	114
5.4.1	NOMENCLATURE OF ENTITIES	115
5.4.2	BOUNDARY REPRESENTATION	115
5.4.3	CSGREPRESENTATION	117
5.5	EXAMPLE 3. GEOMETRIC MODELING	118
5.5.1	PROGRAMMING IN MATLAB	122
5.6	CELL DECOMPOSITION AND SPATIAL OCCUPANCY ENUMERATION	124
5.6.1	QUADTREES	124
5.6.1.1	Observations	125
5.7	EXERCISES – GEOMETRIC MODELING-	127
5.7.1	CSGAND B-REP I.	127
5.7.2	CSGAND B-REP II.	129
5.7.3	CSGAND B-REP II.	131
5.7.4	2D SPATIAL DECOMPOSITION.	134
6.	FINAL EXERCISES	135
6.1	EXERCISE I	135
6.2	EXERCISE II.	137
6.3	EXERCISE III.	140
7.	APPENDIX	144
7.1	CHAPTER 3 -EXAMPLES / PROOFS FOR GEOMETRIC TRANSFORMATIONS -	144
7.1.1	EXAMPLE. RIGID TRANSFORMATIONS (SECTION 3.2.6) - <i>MATLAB CODE</i>	144
7.1.1.1	Auxiliary functions	146
7.1.2	RIGID TRANSFORMATIONS - DEMONSTRATIONS AND PROOFS	148
7.1.3	ADDITIONAL CASES OF MIRROR TRANSFORMATIONS	152
7.1.3.1	Mirror about XZ plane	152
7.1.3.2	Mirror about a plane $X = Y$	153
7.1.3.2.1	Example. Application of the Householder transformation. to perform a mirror about the plane $X = Y$	154
7.2	CHAPTER 4 -EXAMPLES OF CONTROL POINT GENERATION (MATLAB CODE)	155

7.2.1	EXAMPLE. (SPHERE)	155
7.2.2	EXAMPLE. (CONE)	156
7.2.3	EXAMPLE. (CYLINDER)	157
7.2.4	EXAMPLE. (THE MÖBIUS BAND)	158
7.3	CHAPTER 5 – EXAMPLE 3 – IMPLEMENTATION IN MATLAB –	159
7.3.1	AUXILIARY FUNCTIONS	160
8.	BIBLIOGRAPHY	164

INDEX OF FIGURES.

FIGURE 1. PSEUDOCODESTRUCTURE.....	4
FIGURE 2. REPRESENTATION OF AN ASSERTION.....	5
FIGURE 3. PRECONDITION AND POSTCONDITION.....	5
FIGURE 4. ITERATIVE STRUCTURE FOR	8
FIGURE 5. ITERATIVE STRUCTURE WHILE	8
FIGURE 6. GRAPH OF FUNCTION CALLS.....	9
FIGURE 7. STORE THE FINAL RESULTS.....	23
FIGURE 8. SCALAR PRODUCT (PROJECTION OF A VECTOR OVER ANOTHER).....	26
FIGURE 9. CROSS PRODUCT OF TWO VECTORS.....	27
FIGURE 10. PROPERTIES WITH MATRICES.....	28
FIGURE 11. EXAMPLE OF MULTIPLYING A MATRIX BY A SCALAR.....	29
FIGURE 12. EXAMPLE OF MULTIPLICATION OF MATRICES.....	29
FIGURE 13. EXAMPLE OF A MATRIX TRANSPOSITION.....	30
FIGURE 14. DEXTEROUS COORDINATE SYSTEM.....	33
FIGURE 15. SEQUENCE OF GEOMETRIC TRANSFORMATIONS.....	33
FIGURE 16. TRANSLATION TRANSFORMATION.....	35
FIGURE 17. ROTATION TRANSFORMATION ABOUT MAIN AXIS ZW.....	35
FIGURE 18. EXAMPLE OF RIGID TRANSFORMATIONS.....	39
FIGURE 19. ROTATION BY A TANGLE ABOUT ARBITRARILY INCLINED AXIS PASSING THROUGH THE ORIGIN.....	41
FIGURE 20. ROTATION ABOUT AN ARBITRARY AXIS $L=[E,PV]$ BY A TANGLE.....	41
FIGURE 21. ROTATION ABOUT AN ARBITRARY AXIS $L=[E,PV]$ BY A TANGLE.....	43
FIGURE 22. MIRROR ABOUT THE ORIGIN.....	45
FIGURE 23. MIRROR ABOUT A POINT REMOVED FROM THE ORIGIN.....	45
FIGURE 24. MIRROR OR REFLECTION ABOUT A PLANE THAT CROSSES THE ORIGIN.....	46
FIGURE 25. PARALLEL PROJECTION ON A PLANE THAT CROSSES THE ORIGIN.....	46
FIGURE 26. APPLICATION OF HOUSEHOLDER REFLECTOR TO GET A REFLECTION THROUGH PLANE YZ.....	47
FIGURE 27. SCALING TRANSFORMATION WITH THE ORIGIN AS THE FIXED POINT.....	48
FIGURE 28. SCALING TRANSFORMATION WITH FIXED POINT (B) AWAY FROM THE ORIGIN.....	48
FIGURE 29. SHEAR EFFECT WITH ORIGIN AS FIXED POINT.....	49
FIGURE 30. SHEAR EFFECT WITH ARBITRARY FIXED POINT.....	49
FIGURE 31. EXERCISE 3.1.....	51
FIGURE 32. EXERCISE 3.2.....	52
FIGURE 33. EXERCISE 3.3.....	53
FIGURE 34. EXERCISE 3.10.....	55
FIGURE 35. EXERCISE 3.11.....	56
FIGURE 36. EXERCISE 3.12.....	58
FIGURE 37. EXERCISE 3.13.....	59
FIGURE 38. EXERCISE 3.14.....	60
FIGURE 39. EXERCISE 3.15.....	61
FIGURE 40. EXERCISE 3.16.....	62
FIGURE 41. EXERCISE 3.17.....	63
FIGURE 42. GENERATED EXTRUSION OF CIRCULAR CROSS SECTION.....	64
FIGURE 43. SWEEPING AND TWEAKING OF A CROSS SECTION.....	65
FIGURE 44. RIBBON OF RADIUS =40, A=3, B=6, T=360° AND N=120.....	66
FIGURE 45. PARALLEL PROJECTION.....	67
FIGURE 46. PERSPECTIVE PROJECTION.....	69
FIGURE 47. PARAMETRIC MAPPINGS FOR CURVES AND SURFACES.....	72
FIGURE 48. SET CONVEXITY.....	73
FIGURE 49. CONVEX HULL OF A 2D SET S.....	73
FIGURE 50. CONTAINMENT OF A PARAMETRIC FORM.....	73
FIGURE 51. INVARIANCE UNDER INVERSION OF SAMPLE ORDER.....	73
FIGURE 52. SET OF BEZIER WEIGHT COEFFICIENTS (NUM POINS=4).....	74

Eliminado: ¶

FIGURE 53. SET OF UNIFORM B- SPLINE WEIGHT COEFFICIENTS (NUM. POINTS =4).	74
FIGURE 54. EXAMPLE OF BEZIER CURVE	74
FIGURE 55. EXAMPLE OF SPLINE CURVE	74
FIGURE 56. CLOSED BEZIER CURVE (CASE. K=7 CONTROL POINTS (P0=P6).	77
FIGURE 57. BEZIER COEFFICIENTS FOR K=7 CONTROL POINTS	77
FIGURE 58. MATHEMATICALLY OBTAINED CONTROL POINTS OF A TOROIDAL SPIRAL	82
FIGURE 59. CURVE FIT ON THE CONTROL POINTS OF A TOROIDAL SPIRAL	82
FIGURE 60. MATHEMATICALLY OBTAINED CONTROL POINTS OF A SPHERE	82
FIGURE 61. SURFACE FIT ON THE CONTROL POINTS OF A SPHERE	82
FIGURE 62. MATHEMATICALLY OBTAINED CONTROL POINTS OF A CONE	82
FIGURE 63. SURFACE FIT ON THE CONTROL POINTS OF A CONE	82
FIGURE 64. MATHEMATICALLY OBTAINED CONTROL POINTS OF A CYLINDER	82
FIGURE 65. SURFACE FIT ON THE CONTROL POINTS OF A CYLINDER	82
FIGURE 66. DIGITIZED CONTROL POINTS OF A BACKBONE	83
FIGURE 67. RECONSTRUCTED SURFACE OF A BACKBONE	83
FIGURE 68. FINAL SET OF CONTROL POINTS OF A MÖBIUS BAND	84
FIGURE 69. MESH OF A MÖBIUS BAND	84
FIGURE 70. PARAMETERIZATION OF THE MÖBIUS BAND	84
FIGURE 71. BAND WITH PARAMETER $\gamma = 0.250$	85
FIGURE 72. BAND WITH PARAMETER $\gamma = 10$	85
FIGURE 73. BAND WITH PARAMETER $g = 40$	85
FIGURE 74. DISPOSITION OF THE CONTROL POLYHEDRON	86
FIGURE 75. CONTROL POLYHEDRON WITH 21 ² 21 POINTS	87
FIGURE 76. BEZIER SURFACE PATCH FOR A 21 ² 21 CONTROL POLYHEDRON	87
FIGURE 77. VARIABLE MAP FOR SPLINE PATCH, ACCORDING TO CONTROL POLYHEDRON	88
FIGURE 78. A QUARTER OF THE SPLINE STAGES FOR 21 ² 21 CONTROL POLYHEDRON	89
FIGURE 79. SPLINE PATCH FOR 21 ² 21 CONTROL POLYHEDRON (STAGES WITH KXL=3x4 CONTROL POINTS)	89
FIGURE 80. CONTROL POLYHEDRON FOR TORUS	89
FIGURE 81. SPLINE PATCH FOR TORUS CONTROL POINTS. OPENED IN U PARAMETER, CLOSED IN W PARAMETER.	89
FIGURE 82. CONTROL POINT SET OF A SPIRAL IN A TORUS	90
FIGURE 83. MATHEMATICALLY OBTAINED CONTROL POINTS OF A TORUS	95
FIGURE 84. SURFACE OF A TORUS	95
FIGURE 85. CONTROL POINT SET OF A TORUS WITH AN AXIAL SWEEP OF 1.5PI	95
FIGURE 86. SURFACE OF A TORUS WITH SWEEP OF 1.5PI IN BOTH ANGULAR PARAMETERS	95
FIGURE 87. MATHEMATICALLY OBTAINED CONTROL POINTS OF A BAND ON A CONE	96
FIGURE 88. MESH OF A BAND ON A CONE	96
FIGURE 89. THREE MATRICES WHICH STORE THE CALCULATED SURFACE	98
FIGURE 90. BEZIER SURFACE OF A CONICAL BAND	102
FIGURE 91. SPLINE SURFACE OF A CONICAL BAND	102
FIGURE 92. PRIMITIVE ENTITIES	104
FIGURE 93. EXAMPLE OF CSG TREE	105
FIGURE 94. BOOLEAN OPERATIONS	105
FIGURE 95. DEFINITION OF 2 MANIFOLD IN E^3	106
FIGURE 96. DEFINITION OF 2 MANIFOLD WITH BOUNDARY, IN E^3	106
FIGURE 97. BOUNDARY REPRESENTATION	106
FIGURE 98. BODY WITH TWO LUMPS	110
FIGURE 99. LUMP BOUNDED BY TWO SHELLS	110
FIGURE 100. SHELL (MANIFOLD) WITHOUT BORDER	110
FIGURE 101. SHELLS (MANIFOLDS) WITH BORDER	110
FIGURE 102. FACE WITH HOLE, ITS SURFACE (LOOPS FORMED BY EDGES) AND ITS CARRIER SURFACE (SURFACE)	110
FIGURE 103. EDGE, ITS BOUNDARY (VERTICES) AND ITS CARRIER CURVE (CURVE)	110
FIGURE 104. EXAMPLE OF A PARAMETRIC SURFACE (SURFACE)	110

FIGURE 105. EXAMPLE OF A SURFACE (SURFACE) AND PARAMETRIC CURVES (CURVE)	110
FIGURE 106. TOPOLOGIC EQUIVALENCY UNDER GEOMETRIC DIFFERENCE	111
FIGURE 107. EXAMPLE 1 NOMENCLATURE OF ENTITIES FOR THE B-REP TOPOLOGIC ANALYSIS OF A BODY....	112
FIGURE 108. EXAMPLE 1. CSG REPRESENTATION	113
FIGURE 109. EXAMPLE 2. RIGID BODY (NON MANIFOLD).....	114
FIGURE 110. EXAMPLE 2 NOMENCLATURE OF ENTITIES FOR THE B-REP TOPOLOGIC ANALYSIS OF A BODY....	115
FIGURE 111. EXAMPLE 2. CSG REPRESENTATION	117
FIGURE 112. EXAMPLE 3. NOMENCLATURE OF ENTITIES FOR THE B-REP TOPOLOGIC ANALYSIS OF A BODY... 118	
FIGURE 113. CILINDER.	121
FIGURE 114. EXAMPLE 3. CSG TREE	121
FIGURE 115. LOCAL SPACE OF COORDINATES FOR DEFINITION OF CURVED SURFACE.....	122
FIGURE 116. QUADTREE SPATIAL REPRESENTATION.....	125
FIGURE 117. TREE STRUCTURE REPRESENTATION FOR THE QUADTREE STRUCTURE.	125
FIGURE 118. EXERCISE 5.7.1.....	127
FIGURE 119. COORDINATE PLANE.....	128
FIGURE 120. LOCAL COORDINATE FRAMES FOR WEDGE, BLOCK AND PYRAMID.....	128
FIGURE 121. EXERCISE 5.7.2.....	129
FIGURE 122. COORDINATE PLANE.....	130
FIGURE 123. WEDGE.....	130
FIGURE 124. EXERCISE 5.7.3.....	131
FIGURE 125. COORDINATE PLANE.....	132
FIGURE 126. CONTROL SURFACE	133
FIGURE 127. EXERCISE 4.4.....	134
FIGURE 128. FINAL EXERCISE #1	135
FIGURE 129. ILLUSTRATION OF THE ARGUMENTS FOR DRAW_SOLID(BODY, DIMS)	136
FIGURE 130. FINAL EXERCISE #2	137
FIGURE 131. COORDINATED SPACE FOR DEFINING A MESH.....	138
FIGURE 132. WEDGE(DX, DY, DZ)	138
FIGURE 133. MAIN PARAMETERS OF THE INITIAL POSITION OF THE TORUS.....	140
FIGURE 134. MESH OF THE CONTROL POLYHEDRON OF A TORUS.....	140
FIGURE 135. PRINCIPAL AXES OF TORUS IN INITIAL AND FINAL POSITION	142
FIGURE 136. SPLINE SURFACE OF AN ORIGINAL HORIZONTAL TORUS AND A TRANSFORMED VERTICAL TORUS. DIFFERENT CLOSURE DIRECTIONS ARE SHOWN	143
FIGURE 137. RIGID TRANSFORMATIONS.	148
FIGURE 138. SHOWING THAT TRANSLATIONS DO NOT AFFECT ORIENTATION OF THE OBJECT. ONLY ROTATIONS DO SO.....	151
FIGURE 139. MIRROR ABOUT THE XZ PLANE.	152
FIGURE 140. MIRROR ABOUT A PLANE $X = Y$	153
FIGURE 141. PROCEDURE BASED ON THE HOUSEHOLDER REFLECTOR TO OBTAIN THE TRANSFORMATION MATRIX FOR A MIRROR ABOUT THE PLANE $X = Y$	154

INDEX OF EQUATIONS.

EQUATION 1. DEFINITION OF A VECTOR AS THE DIFFERENCE BETWEEN TWO COORDINATE POINTS.....	25
EQUATION 2. MAGNITUDE OF A VECTOR	25
EQUATION 3. DIRECTION COSINES OF A VECTOR	25
EQUATION 4. ADDITION OR SUBTRACTION OF VECTORS.....	25
EQUATION 5. SCALAR PRODUCT OF TWO VECTORS.....	26
EQUATION 6. DISTRIBUTIVE LAW OF MULTIPLICATION FOR VECTORS.....	26
EQUATION 7. RELATIONS BETWEEN UNITARY AXES VECTORS.....	26
EQUATION 8. CROSS PRODUCT OF TWO VECTORS IN E^3	28
EQUATION 9. REPRESENTATION OF A MATRIX $M \times N$	28
EQUATION 10. SCALAR PRODUCT OF A MATRIX WITH A REAL NUMBER.....	29
EQUATION 11. IDENTITY MATRIX	30
EQUATION 12. HOMOGENEOUS COORDINATE FORM OF A POINT.	31
EQUATION 13. HOMOGENEOUS COORDINATE FORM OF A VECTOR	31
EQUATION 14. HOMOGENEOUS COORDINATE FORM OF A MATRIX.....	31
EQUATION 15. CHARACTERISTIC POLYNOMIAL OF A MATRIX A.	32
EQUATION 16. TRANSLATION TRANSFORMATION IN CARTESIAN COORDINATES.....	35
EQUATION 17. TRANSLATION TRANSFORMATION IN HOMOGENEOUS COORDINATES	35
EQUATION 18. MATRIX FORM OF ROTATION AROUND THE ZW AXIS.....	35
EQUATION 19. ROTATION MATRIX AROUND Z AXIS.	36
EQUATION 20. ROTATION TRANSFORMATION AROUND THE XW AXIS.	36
EQUATION 21. ROTATION MATRIX AROUND XW	36
EQUATION 22. ROTATION TRANSFORMATION AROUND AXIS YW	36
EQUATION 23. ROTATION MATRIX AROUND YW.....	36
EQUATION 24. DECOMPOSITION OF A ROTATION ABOUT ARBITRARY AXIS IN E^3	37
EQUATION 25. A "HOMOGENEOUS COORDINATE FORM" TRANSFORMATION: ROTATION FOLLOWED BY TRANSLATION.....	37
EQUATION 26. CONDITIONS FOR A RIGID TRANSFORMATION.	37
EQUATION 27. TRANSFORMATION OF COORDINATE SYSTEM BY RIGID TRANSFORMATION.....	38
EQUATION 28. INVERSE OF A RIGID TRANSFORMATION	38
EQUATION 29. ROTATION OF A POINT ABOUT AN ORIGIN-PIVOTED AXIS CALCULATED BY THE QUATERNION METHOD.	41
EQUATION 30. EIGENVALUE AND EIGENVECTOR MATRICES.	42
EQUATION 31. GENERAL ROTATION BY T ABOUT AN ARBITRARY LINE $L = (E, P, V)$ IN E^3	42
EQUATION 32. INSTANTANEOUS CENTER OF ROTATION FOR ARBITRARY MOVEMENT IN E^3	43
EQUATION 33. MIRROR TRANSFORMATION ABOUT THE ORIGIN.....	45
EQUATION 34. TRANSFORMATION CHAIN FOR MIRROR ABOUT AN ARBITRARY POINT.....	45
EQUATION 35. HOUSEHOLDER REFLECTOR, USED TO MIRROR A POINT ABOUT AN ORIGIN-PIVOTED PLANE.	46
EQUATION 36. MIRROR TRANSFORMATION ABOUT A PLANE NOT CROSSING THE ORIGIN.....	47
EQUATION 37. SCALING TRANSFORMATION WITH ORIGIN AS FIXED POINT.	48
EQUATION 38. SCALING TRANSFORMATION WITH RESPECT TO AN ARBITRARY FIXED POINT IN E^3	49
EQUATION 39. MATRIX FOR SHEAR WITH RESPECT TO THE ORIGIN.....	49
EQUATION 40. SHEAR TRANSFORMATION ABOUT AN ARBITRARY POINT PV IN E^3	50
EQUATION 41. CALCULATION OF A PERSPECTIVE PROJECTION OF A SINGLE POINT INTO A PLANE.	68
EQUATION 42. BERNSTEIN FORM OF THE BEZIER COEFFICIENTS	76
EQUATION 43. MATRIX FORMULATION OF PARAMETRIC CURVES.....	76
EQUATION 44. FORMULATION OF SURFACE PATCH WITH $(M+1) \times (N+1)$ CONTROL POINTS.....	86
EQUATION 45. EXAMPLE OF BEZIER PATCH FORMULATION FOR A 3×4 CONTROL POLYHEDRON.	87
EQUATION 46. MATRIX FORMULATION FOR STAGE OF A SPLINE SURFACE PATCH.....	87
EQUATION 47. MIRROR TRANSFORMATION ABOUT THE XZ PLANE.....	152
EQUATION 48. MIRROR TRANSFORMATION MATRIX ABOUT THE XZ PLANE	152
EQUATION 49. MIRROR TRANSFORMATION ABOUT THE PLANE $X = Y$	153
EQUATION 50. MIRROR TRANSFORMATION MATRIX ABOUT THE $X = Y$ PLANE	153

INDEX OF TABLES.

TABLE 1. GRAPHIC REPRESENTATION OF AN ASSIGNMENT.....	5
TABLE 2. COMPARISON SYMBOLS	6
TABLE 3. ILLUSTRATION OF CONDITIONALS.....	6
TABLE 4. STRUCTURE OF AN ITERATIVE COMMAND.....	7
TABLE 5. USAGE OF SYMBOLIC NAMES.....	9
TABLE 6. ASSOCIATION OF CONDITIONS	10
TABLE 7. COMPARISON BETWEEN INDENTED AND NON-INDENTED CODE	11
TABLE 8. PROPERTIES OF THE CROSS PRODUCT.	27
TABLE 9. NORMS OF VECTORS AND MATRICES	32
TABLE 10. ILLUSTRATIVE STEPS OF A TRANSFORMATION SEQUENCE OF AN OBJECT.	40
TABLE 11. MATRIX FORMULATION FOR BEZIER CURVES.	77
TABLE 12. MATRIX FORMULATION FOR UNIFORM B-SPLINE CURVES.	78
TABLE 13. DISPOSITION OF CONTROL POINTS IN STAGES FOR OPEN SPLINE CURVES.....	79
TABLE 14. DISPOSITION OF CONTROL POINTS IN STAGES FOR CLOSED SPLINE CURVES	81
TABLE 15. EXAMPLE OF TOPOLOGIC EQUIVALENCE AND NON EQUIVALENCE	104
TABLE 16. RELATIONS AND HIERARCHY OF TOPOLOGIC AND GEOMETRIC ELEMENTS IN B-REP.....	107
TABLE 17. SPECIFICATION OF TOPOLOGIES AND GEOMETRIES IN A BODY	111
TABLE 18. EXAMPLE 1. TABLE OF BOUNDARY REPRESENTATION OF A RIGID BODY	112
TABLE 19. EXAMPLE 2. TABLES OF BOUNDARY REPRESENTATION OF A RIGID BODY.....	115
TABLE 20. EXAMPLE 3. TABLE OF BOUNDARY REPRESENTATION OF A RIGID BODY	120

INTRODUCTION

The purpose of this notes is to introduce the reader in basic but important aspects of CAD, CAM and Computer Graphics. The pre-requisites for understanding it are the normal mathematical tools that are part of any undergraduate engineering curricula, and a reasonable interest in scientific programming.

Although the material seems aimed to practitioners and programmers in CAD / CAM / CG, the underlying concepts directly apply to kinematics, robotics, machine tool characterization and, more recently, to medical applications in the boundary with engineering disciplines. This is particularly true for the section on geometric transformations, which is the basis for understanding and working with kinematics and dynamics.

A shallow and pragmatic review of topology, as applied by engineers to CAD, allows the reader to understand many particularities and limitations of the CAD packages. It allows to make informed decisions on the data format used to transmit geometric information for design and manufacturing operations.

The material on parametric curves and surfaces is aimed to inform the reader of the basic technical challenges met by the early researcher on this topic (Bezier and De Casteljau). In this a way, that procedures and formulae proposed by those researchers become natural consequences when attempting to solve the problem of interpolating sequences of points in E^3 . The reader will find in here surface types that are not in the common literature (and whose engineering application may not be widespread), but which oblige a complete understanding of the basic concepts.

ACKNOWLEDGMENTS.

Although it is a hard task to count everyone, my trial starts for my parents, brothers and sisters, who taught me love for learning, reading, math, and respect for fine thinking. My advisors at University of Illinois (Urbana-Champaign) profs. Placid Ferreira, Shiv Kapoor and Richard DeVor struggled to teach me sound scientific mental processes, as well as concrete technical tools and knowledge. I am deeply in debt with them for all their help and the understanding of academic excellence that they tried to pass on to me. Los Andes University (Bogota, COLOMBIA), gave me one of the best undergraduate education in Colombia. In particular, I appreciate the large efforts devoted there to form our minds into entities able of scientific thinking, through tough courses in math, physics, fluid mechanics, thermal sciences, dynamics, etc. (in the Mechanical Eng. Dept), and computer graphics, algorithmics, computer graphics, compilers, computer architecture, etc (in the Computer Science Dept.).

Generations of my students (since 1996) at EAFIT University, Medellin - COLOMBIA, have debugged the imperfections and limitations of the present material. As research and teaching assistants, they have also collected the many exercises given in my exams, polished them, purge their errors, etc. The list includes Carlos Henao, Germán Corredor, Juan D. Vanegas, Elizabeth Rendón, Carlos A. Toro, Sonia Insingares, Hector Villegas, Jairo Saldarriaga, Luis M. Ruiz, Carlos Gonzalez, Johanna Acosta, Sebastián Peña, Alejandro Hoyos, Juan Santiago Mejia, Carlos Roldan, etc. My research assistants Sebastian Schrader, Eliana Vásquez, Natalia Alvarez, Miguel Granados, Jorge Posada helped through their application of the material included here into real CAD / CAM / CG applications. A special mention is owe to my research assistant Carlos Emilio Roldán, who has done a tremendous effort to bring the basic scientific content into a printable, esthetic, and appealing form, at the same time pointing out the unavoidable technical mistakes. The support of Yazmit Eliana Castaño in giving format to the different language versions of the document is also deeply thanked.

People and research institutes which have also contributed to this work in various ways are: Prof. Dr. J. L. Encarnacao, Dr. Joachim Rix, Dr. Uwe Jasnoch and Dr. Georgios Sakas from Fraunhofer Institute for Computer Graphics (Darmstadt, Germany); Dr. Charles Wu from Ford Motor Co. (Dearborn, USA); Dr. Xoan Leiceaga Baltar from University of Vigo, (Galicia-Spain), and many others.

Finally, I would like to thank EAFIT University, in particular the Dean of Engineering, Dr. Alberto Rodriguez and the Dean of Research, Dr. Félix Londoño, for their continuous support to the CAD / CAM / CG Laboratory, through research and teaching resources. Those resources were and are capital in developing these notes.

The EAFIT University and myself wish to thank the Gold Museum (Museo del Oro) and Banco de la Republica de Colombia, for the images of gold pieces, belonging to its collection, which appear in the cover of the book.

1. BASIC CONCEPTS OF PROGRAMMING

1.1 Algorithms

The solution of a problem can be developed by means of a mathematical model. This model can be represented as a set of instructions that are realized by an algorithm. Each instruction has a precise meaning and is executed a finite number of times.

1.2 Pseudo language

The pseudo language or pseudo code is a way to describe sequentially the logical procedure that an algorithm must realize. It is also the previous step to the codification or implementation of the algorithm in any programming language. Its use allows the planning of the program, that is to say, the programmer is only interested in the logic and the control structures, not in the syntactic rules of a specific language. This facilitates the correction of possible logical errors that the algorithm may contain.

```
(1) function Find maximum value
(2)
(3)     {Let M = array of real numbers }
(4)     maximum = value of the first position in the array M;
(5)
(6)     for (each position i in array M)
(7)     {
(8)         if (value i in array M is greater than value stored in maximum )
(9)         {
(10)             update the value of maximum ;
(11)         }
(12)     }
(13)
(14)     {maximum = maximum value in M}
```

Figure 1. Pseudo code structure

In Figure 1 some important characteristics of a pseudo code are observed. The first of them is that small letters in bold are used for the key words of programming languages. The conditions of flow control (**if**, **for**, **while**) are used in the propositions of the pseudo language. The conditional expressions as the one expressed in Figure 1 line (8), may be informal propositions, instead of conditional expressions of a programming language. Notice that the assignment in the Figure 1 line (4) uses an informal expression to the right, and that the cycle **for** of Figure 1 line (6) closes the repetition of the set of instructions under its domain. The domain is a group of instructions that opens and closes the pair of characters "{" and "}" respectively.

1.2.1 Assertion (Pre, Post, Inv)

The status of a program can be described by using propositions located in any place of interest. This type of proposition is called an **assertion**. In the design and interpretation of the different parts of the pseudo code, the assertions are between brackets "{ }". The assertion describes the status of the variables of the program when the execution passes by the point where it is located. Therefore the assertion is **NOT** an executable instruction, but only a description of a status of the program. In conclusion an assertion doesn't have any effect on the execution of the program.

By definition, when describing the state of the execution of the program (with all the possible cases) an assertion is never false. For example, the assertion in Figure 2 indicates that when the execution of the program passes by it, the variable *Interruptor 1* is True and *Interruptor 2* is False.

$$\{ (Interruptor1 == True) \wedge (Interruptor2 == False) \}$$

Figure 2. Representation of an assertion

1.2.2 Precondition and postcondition

When one wants to specify the operation of a program, its initial and final status should be described so that its correct execution can be verified. To describe the initial status of the program, an assertion called **precondition** is used and to describe the final status of the program another assertion called **postcondition** is used as well. Retaking the pseudo code from Figure 1, it is established that the precondition and the postcondition are defined as they are shown in Figure 3.

$$\begin{aligned} \{ \text{pre: } M &= \text{array of real numbers} \} \\ \{ \text{post: } Maximum &= \text{maximum value in } M \} \end{aligned}$$

Figure 3. Precondition and postcondition

It is recommended to specify the precondition at the beginning and the postcondition at the end of the code. They should be fulfilled every time that the program is executed

1.3 Executable instructions

1.3.1 Assignment, verification or comparison

The use of variables allows the simulation of a process, task or problem in such a way that the information stored in them can be manipulated according to the user's necessity. An assignment example is observed in Table 1.

Table 1. Graphic Representation of an assignment

$a = 1;$	Arithmetic assignation
$color = 'rojo';$	Character string assignation

The computer executes each assignment sentence in two stages. In the first one, the value of the expression that is written to the right side of the assignment operator is calculated. In the second stage, the value is stored in the variable whose name is written to the left of the assignment operator.

In the assertions that evaluate the content of variables in a program the relationship or comparison operators shown in Table 2 are used

Table 2. Comparison symbols

MATLAB	C/C++	MEANING
>	>	Greater than
<	<	Lower than
==	==	Equal to
~=	!=	Different of
<=	<=	Lower or equal to
>=	>=	Greater or equal to

1.3.2 Selection and decision making

It is necessary to incorporate decision structures, so that an algorithm can follow different execution routes. A decision instruction evaluates a condition and in function of the result obtained, the execution branches. The most used structure in programming languages is **if** and **else**. Table 3 illustrates the general form of this instruction and an example of its usage. Generic illustration of conditional instructions. Program that calculates the absolute value of a number by means of decision structures

Table 3. Illustration of Conditionals

<pre> if (condition1) { group of commands 1; } else if (condition2) { group of commands 2; } else if (condition3) { group of commands 3; } else { group of commands 4; } </pre> <p><i>Generic illustration of conditional instructions</i></p>	<pre> {pre: N is a real number} if(N > 0) { ABS = N } else if (N < 0) { ABS = - N } else { ABS = 0 } {post : ABS = N } </pre> <p><i>Program that calculates the absolute value of a number by means of decision structures</i></p>
--	--

The conditions are verified one by one. If the condition is satisfied, the command block in its interior is executed. If it is not satisfied, the other conditions will be verified. An **else** should always be placed at the end to include the pathological or not foreseen cases.

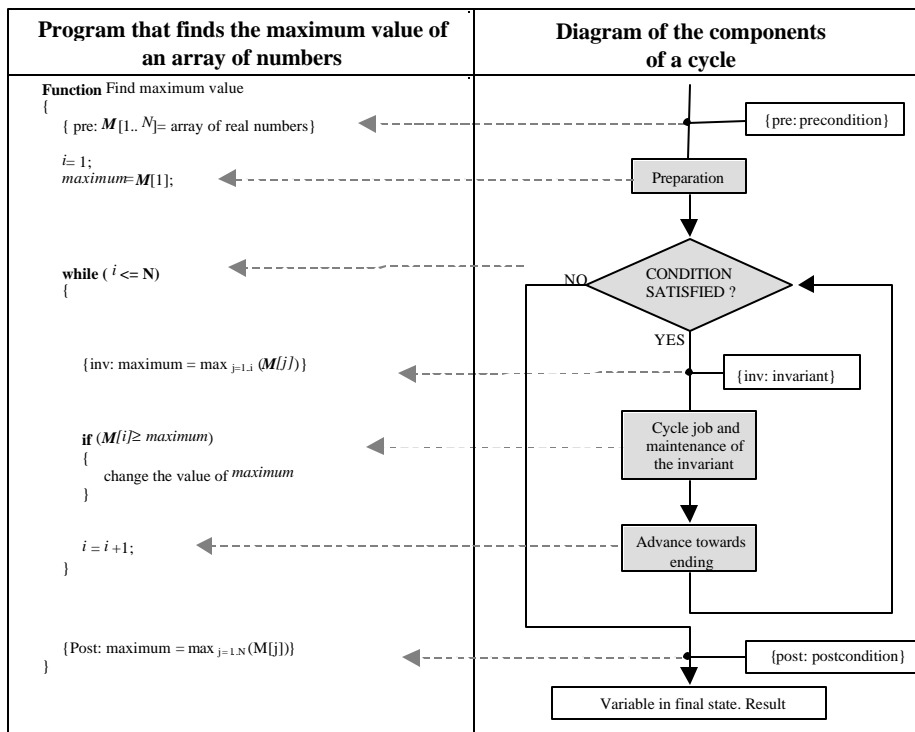
1.4 Iterative commands or cycles

The iterative commands allow the repeated execution of one or several commands, whenever a condition initially established is satisfied. The iterative instructions are also known as **cycles**. Every cycle should be characterized by containing:

1. Preparatory instructions.
2. Condition of authorization for execution of the cycle body.
3. Invariant: an assertion that is true at the beginning of each iteration of the cycle.
4. Cycle body.
 - 4.1 Job of the cycle and maintenance of the invariant.
 - 4.2 Advancing towards cycle ending.

The execution of an iterative command is authorized by a condition that must be evaluated as true. If the condition is true, the body of the cycle is executed. The execution of the iterative command ends at the moment in which the condition becomes false.

Table 4. Structure of an iterative command



In order to specify the behavior of an iterative command, an assertion or statement (it is not an executable instruction) is identified. This assertion is called **invariant** and represents the status of the execution exactly after the Boolean condition controlling the iterations, and holds invariant and true each time the cycle body is going to be executed (see Table 4). Therefore, the invariant determines (a) the preparation for the loop, (b) the *hard-work* part of the loop, and (c) the advancement towards termination. In addition, the logical equation:

“invariant AND not (Boolean condition for the loop)” must coincide with the Post-condition. Because its massive influence in all other instructions of an iterative instruction, the invariant is the single-most important logical predicate to be identified when writing such instructions

One of the most common iterative commands is **for** (Figure 4). Its operation results from the previous knowledge of the number of iterations (N) required to execute the block of commands. The iterative cycle stops when the execution reaches this predetermined number of iterations. Inside the area of commands the variable “i”, which carries the value of the number of iterations, should **NOT** be manipulated, because the cycle **for** automatically updates it in each cycle.

```
for ( $i \leq N$ )  
{  
  {inv: (invariant)}  
  commands  
}
```

Figure 4. Iterative structure **for**

While, is another iterative command (Figure 5) whose authorization to be executed is controlled explicitly in its preparation and within the commands in its domain. In these commands, when the execution has reached its objective, it so happens that the condition is not fulfilled anymore and therefore the cycle **while** ends. Then the following instruction to the cycle **while** is executed.

```
while (condition)  
{  
  {inv: (invariant)}  
  commands  
   $variable = variable \pm increment$   
}
```

Figure 5. Iterative structure **while**

The use of **while** allows the termination of the cycle according to the user's necessities.

1.5 Programming criteria

For the interpretation of a program the use of certain programming norms that allow the follow up of the program become necessary, as well as the detection and correction of errors.

1.5.1 Structure

A program or task can be realized in two ways. In the first, commands should be grouped by specific **functions** and then call them from a main code. By doing so, a short, simple and easy to understand code is achieved. The second alternative is to make a long and complex sequence of commands, therefore it is not advisable.

A function is a short sequence of commands that solves a specific problem. For complex tasks this problem will be divided in conceptually consistent sub problems. Each one will be responsibility of a function. The decomposition of sub-problems in yet other sub-problems follows the same philosophy, until achieving that the biggest problem is solved with the collaboration of several sub routines and small functions.

Figure 6 shows a group of calls and dependencies inside the program or function F_i . An arrow of F_i toward F_j indicates that the function F_i uses the function F_j .

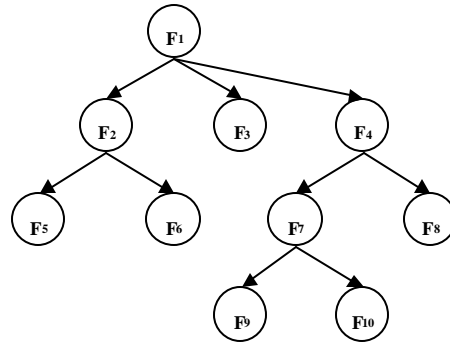


Figure 6. Graph of function calls

It is always suggested to begin to write the functions from the base of the graph of calls (F_5 , F_6 , F_9 , F_{10} , F_8). It allows to write and to prove each function as they end, thus saving time and avoiding errors.

1.5.2 Name of functions and variables

In the case of the functions, the name should describe its purpose clearly. The name of the variables has to be of mnemonic type so that it describes as much as possible their content. In order to increase the information provided by the names, it is necessary to implement the use of comments inside the program that give bigger description of use and type of data.

It is not advisable that numeric literals (constants) or others (1.05, 2, -1, 'a', etc) appear in the program. It is necessary to assign the literal to a variable and to use only that variable for that objective during the whole program. In the event of being required the change of such literal, it is easier to change the value in the initial assignment of the variable than to look for that literal in each instruction where it was used and replace the value. See Table 5.

Table 5. Usage of Symbolic Names

NOT RECOMENDED	OPTIMAL
<pre> if (N == 2) { i = 2; ... k = k/2; } </pre>	<pre> size = 2; if (N == size) { i = 2; k = k/size; } </pre>

Notice that the use of constant 2 in the substitution process on the left side of Table 5 is not convenient. One reason is that its process is risky because **NOT** all constants 2 should be changed (See Table 5 right side).

1.5.3 Validation of data

It must be verified that the data input to every function are within the established range and that they have consistent values. There may exist erroneous data which enter the function, if this happens the problem should be solved. If it is not possible the program must be aborted and a report should be made informing the user about the existing error.

1.5.4 Errors

There exist two types of errors:

Of Syntax: they are found in the compilation phase or interpretation phase of the program, they occur due to characteristic causes of the language syntax. They are easy to correct.

Of Logic: they occur during the execution of a program. They are difficult to detect, they may or may not stop the execution of the program, thus producing erroneous results. The following are typical errors:

1. Inconsistent handling of matrix dimensions, operating matrices whose dimension is **NOT** compatible.
2. Omission of the “advancing towards ending” in a **while** cycle.
3. Incorrect calls of functions or disorder in the input parameters.

1.5.5 Association of operations

When using expressions, which involve two or more operators, it is important to apply the priority rules, which govern the order and the precedence of the operations. When a combination of conditions must be fulfilled, it is necessary to use grouping signs in order to ensure that they are executed in the desired sense. (Table 6).

Table 6. Association of conditions

NOT RECOMENDED	IMPROVED
<pre>size = 3; degrees = 2; if ~ N < size ^ M > degrees v R = 1 { ... }</pre>	<pre>size = 3; degrees = 2; if (~ ((N < size) ^ (M > degrees))) v (R = 1) { ... }</pre>

1.5.6 Indentation

Indentation is a tabulation that is controlled by the user, in which executions or specific tasks are visually grouped. A good indentation improves the design; it facilitates the debugging and modification of a program as well.

The structure of a program that calculates the factorial of a non-negative number is taken as an example (Table 7). It is obvious that the right side code is clearer, since it can be easily established which tasks are executed in the iterative command **while**.

Table 7. Comparison between indented and non-indented code.

NON-INDENTED CODE	INDENTED CODE
<pre> {pre: $N \geq 0$ } $i = 0$; $fact = 1$; while ($i \neq N$) { inv: $fact = (i-1)!$ } $fact = fact * i$; $i = i + 1$; } {post: $fact = N!$ }</pre>	<pre> {pre: $N \geq 0$ } $i = 0$; $fact = 1$; while ($i \neq N$) { {inv: $fact = (i-1)!$ } $fact = fact * i$; $i = i + 1$; } {post: $fact = N!$ }</pre>

1.6 EXERCISES - BASIC CONCEPTS OF PROGRAMMING.

A brief explanation about the operating of some MATLAB commands will be found on this chapter, creation of new functions and how to execute them from a main program. The commands are recognized because their characters are in **bold** letters.

1.6.1 The Variable Working Space in MATLAB.

OBJECTIVE:

To learn some basic commands.

PROCEDURE:

To obtain information. Type the command **help** and then the name of the topic or command to be consulted.
help

1. Eliminate one or all the variables from the workspace. **clear**
2. Clean the command window. **clc**
3. List the variables of the workspace. **who**
4. See the latest computed result. **ans**

The commands used to work on MATLAB may be typed directly on the command window. They can also be previously written on a text file with extension “.m”. When typing the file name at the MATLAB prompt the commands are read directly from the file and executed sequentially. In order to create this type of files (also called *scripts*), the “New M-file” key is selected on the MATLAB menu (File / New / M-file), allowing the invocation the MATLAB text editor for their creation and debugging. For the correct operation of the *scripts* it is necessary that the directory where the file to be executed is located be included on the MATLAB work path directories (see the **path** command).

1.6.2 Scalars, Vector and Matrix Operations in MATLAB.

OBJECTIVE:

Realize the main arithmetic operation among scalars, vector and matrices. To store the history of the session in a log file.

PROCEDURE:

1. Read the help about the **diary** command. Open a diary or log for the work that follows, with the name that you wish. You will need this file at the end of this exercise.
2. Eliminate one or all the variables from the workspace.
3. Create a scalar whose value is 12. Store the value of the variable at *esc1*.
4. Create a scalar whose value is 4. Store the value of the variable at *esc2*.
5. Add the values of the variable *esc1* and *esc2*. Store the result in the variable *sum_esc*.
6. Subtract the values of the variable *esc1* and *esc2*. Store the result in the variable *subst_esc*.
7. Multiply the values of the variable *esc1* and *esc2*. Store the result in the variable *mult_esc*.
8. Divide the value of the variable *esc1* by the values of the variable *esc2*. Store the result in the variable *div_esc*.
9. Raise the value of the variable *esc1* to the value of variable *div_esc*. Store the result in the variable *exp_esc*. **operator ^.**
10. Calculate the square root of the variable *exp_esc*. Store the result in the variable *rz_esc*. **sqrt**
11. Create a 4x4 matrix ***M1*** with all its elements equal to one. **ones**
12. Create a 4x4 matrix ***M2*** with all its elements being random numbers. **rand**
13. Create a 3x4 matrix ***M3*** with all its elements equal to zero. **zeros**
14. Create a 4x4 identity matrix, the matrix name must be ***identity***. **eye**
15. Multiply the value of the variable *esc1* by the matrix ***identity***. Store the result in the variable ***mult1***.
16. Multiply the matrix ***mult1*** by the matrix ***M2***. Store the result in the variable ***mult2***. To calculate $M3 = M1 * M2$ verify that $M1$ is $m \times n$ and $M2$ is $n \times p$.
17. Multiply the value of the variable *esc2* by the matrix ***M1***. Store the result in the variable ***produc1***.
18. Multiply the elements of the matrix ***produc1*** one by one by the elements of the matrix ***M2***. Store the result in the variable ***produc2***.
19. Raise the element of the matrix ***M2*** to the square root. Store the result in the variable ***elev***.
20. Create the vector (1x3) ***vect1*** = [1 2 3].
21. Save the first three values of an ***M2*** column in the variable ***vect2***.
22. Create a vector ***vect2***, wich values are in a range from 1 to 50 with intervals of 2. Operator :
23. Query the size of ***vect2*** and store the result on the variable ***v_size***. **size**
24. Transpose the vector ***vect1*** and store the result on the same variable.
25. Multiply the vector ***vect2*** by vector ***vect1*** and store the result on the variable ***prod***.
26. Close and save the log file.
27. Put the commands of this exercise in the file or script "*exercise_CP1_002.m*". Such commands can be found in the log file. Execute the file "*exercise_CP1_002.m*" from the prompt MATLAB. **script**.

1.6.3 Introduction to User Commands for Session Control in MATLAB.

OBJECTIVE:

Create applications to work with users interface commands. Enter the input data with the keyboard or the mouse. This exercise must be executed from a *script* or a MATLAB file commands (*.m).

PROCEDURE:

1. Eliminate all the variables from the workspace
2. Clean the command window.
3. Create a menu window (for example to choose a day from the week) and store the result on the variable *day*. **menu**.
4. Make a pause and then continue with the execution. **pause**
5. Display a message on the MATLAB command line. It should request the user to press any key to continue with the execution program. **disp, pause**
6. Display a message on MATLAB command line. It should request the user to type a number. Store the result on the variable *number*. **Input**

1.6.4 2D Plotting in MATLAB.

OBJECTIVE:

Plot the expression " $y = \sin(x)$ ". The x values are in a range $[0, 2\pi]$ with intervals of $\pi/12$. This exercise must be executed from a *script* or a MATLAB file commands (*.m).

PROCEDURE:

1. Eliminate all the variables from the workspace
2. Request a MATLAB graphic window. **figure**
3. Clean the graphic window. **clf**
4. Create a row vector x with values in $[0, 2\pi]$ with intervals of $\pi/12$.
5. Calculate y vector as " $y = \sin(x)$ ". **sin()**
6. Plot x vs y . **plot**
7. Name the axis "X" and "Y" on the graphic. **xlabel, ylabel**
8. Title the graphic. **title**
9. Turn on the graphic grid. **grid**
10. Make a pause during the execution program.
11. Change the axis limits of the graphic. Left inferior point = (-1, -2), right superior point = $(3\pi, 2)$. **axis**
12. Make a pause during the execution.
13. Turn off the graphic grid.
14. Display in the graphic window the text '(1.0, 1.1)' on the window position (1.0, 1.1). **text**
15. Put on the graphic window labels for the left inferior point and right superior point.
16. Display a message on MATLAB command line. It should request the user to pick a point with the mouse, by using the graphic window. **ginput**
17. Display a text on the graphic window, use the mouse to pick its position. **gtext**

1.6.5 3D Plotting in MATLAB.

OBJECTIVE:

Calculate and plot the expression " $z = \cos(x) + \sin(y)$ " for values of x and y in a range $[0, 10]$ with intervals of 0.05. This exercise must be executed from a *script* or a MATLAB file commands (*.m).

PROCEDURE:

1. Eliminate all the variables from the workspace.
2. Request a MATLAB graphic window.
3. Clean the graphic window.
4. Create a row vector x with values are in $[0, 10]$ with intervals of 0.05.
5. Create a row vector y with values are in $[0, 10]$ with intervals of 0.05.
6. Calculate z vector as function of " $z = \cos(x) + \sin(y)$ ". *cos()*, *sin()*.
7. Plot x, y, z *plot3*
8. Place axes labels "X", "Y" and "Z" on the graphic window.
9. Title the graph.
10. Turn on the graphic grid.

1.6.6 Plotting of 2D Polygonal Regions in MATLAB.

OBJECTIVE:

Use the mouse to capture points from the screen. Data input with keyboard or the mouse. This exercise must be executed from a *script* or a MATLAB file commands (*.m).

PROCEDURE:

1. Eliminate one or all the variables from the workspace
2. Display a message on MATLAB command line. It should request the user to type the number of points to be captured from the screen. Store the result on the variable *N*
3. Request a MATLAB graphic window.
4. Clean the graphic window.
5. Change the axis limits of the graphic. Left inferior point (0,1.5), right superior point (3,3.5).
6. Label the axis X" and "Y" on the graphic window.
7. Create the vector *coordX* = [1,2,2,1].
8. Create the vect or *coordY* = [2,2,3,3].
9. Plot the vector *coordY* vs *coordX*
10. Make a pause during the execution program.
11. Request a new MATLAB graphic window.
12. Extend the vector *coordX* and *coordY* in such a way that the command to plot *coordY* vs *coordX* generates a closed rectangle. Plot again *coordY* vs *coordX*.
13. Freeze the graphic. **hold**
14. Change the axis limits of the graphic. Left inferior point (0,1.5), right superior point (3,3.5).
15. Fill the closed polygon. **fill**
16. Display a message on MATLAB command line to request the user to catch the *N* points from the screen.
17. Use the mouse to catch *N* points from the screen. Store the result on the array *xy* (*N*x2) .
18. Plot the second column of *xy* vs. the first one.

1.6.7 Generation and Plotting of 3D Meshes in MATLAB.

OBJECTIVE:

Plot a surface with different colors, by using the *colormap*. This exercise must be executed from a *script* or a MATLAB command file (*.m).

PROCEDURE:

1. Eliminate the variables from the work space.
2. Request a MATLAB graphic window.
3. Clean the graphic window.
4. Create a row vector *x* with values are in [-3, 3] with intervals of 0.5.
5. Create a *y* vector equal to
6. Calculate *X* and *Y* matrices that define a grid, based on *x* and *y*. *meshgrid*
7. Calculate the *Z* matrix in function of *X* and *Y*. *peaks*
8. Plot *Z*. *surf*
9. Define a gray color map. *colormap*
11. Title the graphic.
12. Label the axis "X", "Y" and "Z" on the graphic.
10. Request a new MATLAB graphic window.
11. Plot *Z*.
12. Define a different color map.
13. Title the graphic.
14. Name the axis "X", "Y" and "Z" on the graphic.

1.6.8 Menu Management in MATLAB.

OBJECTIVE:

To create an application by using the graph command **figure** and the interface command **menu**. This exercise must be executed from a *script* or a MATLAB command file (*.m).

PROCEDURE:

1. Eliminate all the variables from the work space.
2. Clean the command window
3. Create a window menu with the following options: yellow, blue, red, green and exit.
4. Iteratively, place a text (see table) in the graphic window, according to the user selection. To do that, you must request and clean a MATLAB graphic window inside of the iterative cycle. *text, if, while*

Selection	Text	Position in graphic window
'yellow'	'yellow'	Lower right corner
'blue'	'blue'	Upper right corner
'red'	'red'	Upper left corner
'green'	'green'	Lower left corner
'exit'	--	--

5. The program must end when the user selects the option *exit while*

1.6.9 Invariants and Iterative Cycles (summation of array contents) in MATLAB.

OBJECTIVE:

To calculate the summation of the values of a column vector \mathbf{M} . This exercise must be executed from a *script* or a MATLAB command file (*.m).

PROCEDURE:

1. Eliminate all the variables from the workspace.
2. Prompt the user for the number of rows of vector \mathbf{M} . Store that value in the variable N .
3. Generate the vector \mathbf{M} of $N \times 1$. This vector must be filled with random values.
4. Calculate the summation of entries in \mathbf{M} by using an iterative cycle **while** with a control variable i . A variable *sum* serves as a partial accumulator along the cycle execution.
5. Store the final result of *sum* in a new variable called *Sumator*.
6. Display on the command line the value of *Sumator*.

1.6.10 Invariants and Iterative Cycles (average values of array contents) in MATLAB .

OBJECTIVE:

To calculate the average of the values of a column vector \mathbf{M} . This exercise must be executed from a *script* or a MATLAB command file (*.m).

PROCEDURE:

1. Eliminate all the variables from the work space.
2. Prompt the user for the number of rows of vector \mathbf{M} . Store that value on the variable N .
3. Generate the vector \mathbf{M} of $N \times 1$. This vector must be filled with random values.
4. Calculate the summation of entries in \mathbf{M} by using an iterative cycle **while** with a control variable i . A variable sum serves as a partial accumulator along the cycle execution.
5. Calculate the average of the \mathbf{M} values. This result must be store in the variable $mean$.

1.6.11 Invariants and Iterative Cycles (sorting or array contents) in MATLAB.

OBJECTIVE:

To sort the values of a row vector **a** in decreasing order. This exercise must be executed from a *script* or a MATLAB command file (*.m).

PROCEDURE:

1. Write a function **[b] = swap(a, i₁, i₂, i₃, i₄)**. This function swaps two values in a matrix. The inputs are: matrix **a** and the locations of the two values, (i₁, i₂) and (i₃, i₄). The output parameter is the matrix **b**, equal to **a**, except for the swapped values.
2. Write a function **[b] = my_sort(a)**. The input parameter to this function is the vector (1xN or Nx1) **a** in arbitrary order. The output parameter **b** is the sorted copy of **a**, with its values in decreasing order. This function should contain:
 - 2.1. A counter *cont1*, started at one.
 - 2.2. An iterative cycle **while** which covers all the positions of vector **a**. Use the counter *cont1* and the variable *N* to control the iterative cycle operation. Create inside this cycle another iterative cycle **while**. It must be controlled by another counter called *cont2* and by the variable *N*. This last cycle ensures the organizing of the sub-array **a**(1 : *cont1*) by calling function **swap**.

1.6.12 Invariants and Iterative Cycles (minima and maxima of array contents) in MATLAB

OBJECTIVE:

To identify the maximum, minimum and the locations of those values within a vector \mathbf{M} ($1 \times N$ or $N \times 1$). This exercise must be executed from a *script* or a MATLAB command file (*.m).

PROCEDURE:

1. Eliminate all the variables from the work space.
2. Prompt the user for the number of rows of vector \mathbf{M} . Store, that value in the variable N .
3. Generate the vector \mathbf{M} of $N \times 1$. This vector must be filled with random values.
4. Initialize and use the variables:

 $posmax$: index where the maximum value is stored.

 $posmin$: index where the minimum value is stored.

 Max : maximum value of the vector.

 Min : minimum value of the vector.
5. Execute the iterative cycle **for** controlled by i and N to calculate the maximum and minimum values and its positions.
6. When the iterative cycle is completed store the final result is as it appears on Figure 7, in a matrix called **Minmax**

$$\mathbf{Minmax} = \begin{bmatrix} Min & posmin \\ Max & posmax \end{bmatrix}$$

Figure 7. Store the Final Results.

1.6.13 Functions and Sub-Routines in MATLAB.

OBJECTIVE:

To build a MATLAB program which calculates the average, addition, min y max, of the values of a matrix **A**. This exercise must be executed from a *script* or a MATLAB command file (*.m).

PROCEDURE:

1. Prompt the user for the number of rows and columns of matrix **A**. Store these values in the variable **M** and **N** respectively.
2. Generate a matrix **A** of $M \times N$. This matrix must be filled with random values.
3. Write a function $[sum] = sumt(A)$ that calculates the total addition (*sum*) of the elements of matrix **A**.
4. Write a function $[avg] = average(A)$ that calculates the average value (*avg*) of the elements of matrix **A**.
5. Write a function $[max_v, posmax] = maximum(A)$ that finds the maximum value (*max_v*) of the matrix **A** and stores its location in vector (1x2) *posmax*.
6. Write a function $[min_v, posmin] = minimum(A)$ that finds the minimum value (*min_v*) of the matrix **A** and stores its location in vector (1x2) *posmin*.
7. Write a main program called *exercise_1_13*, where each one of the developed functions is used.

2. BASIC CONCEPTS ON LINEAR ALGEBRA

2.1 Vectors and points

A *point*, represents a specific position in E^3 . Seen from a reference frame, it is described by coordinate values specific to such a coordinate system. A *vector* is defined as the difference between the positions of two points. A three-dimensional vector is defined as follows: (Equation 1):

$$\mathbf{v} = (\mathbf{p}_2 - \mathbf{p}_1) = (x_2 - x_1, y_2 - y_1, z_2 - z_1) = (v_x, v_y, v_z)$$

Equation 1. Definition of a vector as the difference between two coordinate points.

Here the cartesian components v_x, v_y, v_z are the components of \mathbf{v} on the X, Y and Z axes respectively.

A vector can be described as a directed line segment, between two points, with a magnitude and a direction. For any three-dimensional vector, the magnitude is found by using Pythagoras' theorem, Equation 2.

$$|\mathbf{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

Equation 2. Magnitude of a Vector

The **magnitude** of a vector is independent of its coordinate representation. The vector's **direction** is given by the *direction angles* α, β, γ that the vector forms with each one of the positive coordinate axes (Equation 3).

$$\cos(\alpha) = \frac{v_x}{|\mathbf{v}|} \quad \cos(\beta) = \frac{v_y}{|\mathbf{v}|} \quad \cos(\gamma) = \frac{v_z}{|\mathbf{v}|}$$

Equation 3. Direction Cosines of a vector

It is only necessary to specify two of the direction cosines of the vector, since:

$$\cos^2(\alpha) + \cos^2(\beta) + \cos^2(\gamma) = 1$$

The length of a vector \mathbf{v} , written as $|\mathbf{v}|$, is called magnitude or *norm*. For an n -dimensional vector the length is calculated as follows:

$$\text{if } \mathbf{v} = [v_1, v_2, v_3, \dots, v_n] \quad \text{then} \quad |\mathbf{v}| = \sqrt{v_1^2 + v_2^2 + v_3^2 + \dots + v_n^2}$$

2.1.1 Operations with vectors

2.1.1.1 Addition and Subtraction

Functionality: $[+]: \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$

This operation is limited only to vectors of the same dimension, Equation 4.

$$\text{Let } \mathbf{v} = [v_x, v_y, v_z] \quad , \quad \mathbf{w} = [w_x, w_y, w_z] \\ \mathbf{v} \pm \mathbf{w} = [v_x \pm w_x, v_y \pm w_y, v_z \pm w_z]$$

Equation 4. Addition or subtraction of Vectors

2.1.1.2 Product

2.1.1.2.1 Scalar, inner or Dot Product (•)

Functionality: $[\bullet]: \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}$

The scalar product of two vectors A and B is defined as a scalar quantity that is equal to the multiplication of the vectors magnitudes and the cosine of the angle a between their directions. This operation is written mathematically as in Equation 5

$$A \bullet B \equiv |A| \cdot |B| \cos(a) \quad 0 \leq a \leq \pi$$

Equation 5. Scalar product of two vectors

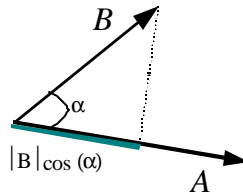


Figure 8. Scalar product (Projection of a vector over another)

Where a is the angle between A and B . In Figure 8 the magnitudes of vectors A and B are shown. Observe that it is not necessary for A and B to have the same length. In this figure can also be seen that $|B| \cos(a)$ stands for the projection of B over A , therefore, the definition of $(A \bullet B)$ can be considered as the (signed) magnitude of the projection of B over A (or vice versa, since $(A \bullet B) = (B \bullet A)$).

The dot product of two vectors is zero if the two vectors are orthogonal and obeys the distributive law of multiplication with respect to addition (Equation 6).

$$A \bullet (B + C) = (A \bullet B) + (A \bullet C)$$

Equation 6. Distributive law of multiplication for vectors

If A is perpendicular to B ($\alpha = 90^\circ$), then $(A \bullet B) = 0$. Also, $(A \bullet B) = 0$ in the most trivial case in which A or B are zero. If A or B point to the same direction ($\alpha = 0^\circ$), then $(A \bullet B) = (|A| |B|)$. If A and B point to opposite directions ($\alpha = 180^\circ$), then $(A \bullet B) = -(|A| |B|)$. The scalar product is negative when $90^\circ < \alpha < 180^\circ$.

The unitary vectors, i, j and k , that define positive direction of the axes X, Y and Z of a right-handed reference frame satisfy:

$$i \cdot i = j \cdot j = k \cdot k = 1, \quad i \cdot j = j \cdot i = 0, \quad j \cdot k = k \cdot j = 0, \quad k \cdot i = i \cdot k = 0$$

Equation 7. Relations between unitary axes vectors

The vectors A and B can be expressed in component form like:

$$A = (a_x)i + (a_y)j + (a_z)k \quad B = (b_x)i + (b_y)j + (b_z)k$$

So, the scalar product of A and B given by the preceding equations can be reduced to:

$$(A \bullet B) = a_x b_x + a_y b_y + a_z b_z$$

In the special case in which ($A = B$), it is seen that: $(A \cdot A) = a_x^2 + a_y^2 + a_z^2 = |A|^2$

2.1.1.2.2 Cross Product (\times)

Functionality: $[\times]: \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$

Given two vectors A and B , the cross product ($A \times B$) is defined as a third vector C . This resulting vector is then defined as in Equation 8 and written as:

$$C = A \times B$$

The magnitude of this vector is

$$|C| = |A| |B| |\sin(\alpha)|$$

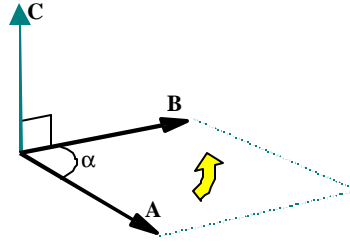


Figure 9. Cross product of two vectors

Observe that the quantity $|A| |B| |\sin(\alpha)|$, is equal to the area of the parallelogram formed by A and B , as shown in Figure 9. The direction of ($A \times B$) is perpendicular to the plane holding A and B . The sense of this new vector is determined based on the sense of advance of a right threaded screw. It may be more convenient to use the *right hand* rule to memorize this concept, recalling that the fingers close in the same sense as is vector A swapped onto vector B . The thumb will then point in the direction of C .

Some properties of the cross product that follow from its definition are the following:

Table 8. Properties of the cross product.

1. If A is parallel to B ($\alpha = 0^\circ$ or 180°), then $(A \times B) = 0$; hence, $(A \times A) = 0$
2. If A is perpendicular to B , then $|A \times B| = |A| |B|$
3. The cross product obeys the distributive law with respect to the sum.
 $A \times (B + C) = A \times B + A \times C$

Unlike the scalar product, for the cross product it is important the order in which the operation is done,

$(A \times B) = -(B \times A)$. The *right hand* rule is a way to verify this.

In the case of three-dimensional vectors, the cross product is evaluated as in Equation 8.

$$C = A \times B = \begin{vmatrix} i & j & k \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix} = i(A_y B_z - A_z B_y) + j(A_z B_x - A_x B_z) + k(A_x B_y - A_y B_x)$$

$$C = A \times B = ((A_y B_z - A_z B_y), (A_z B_x - A_x B_z), (A_x B_y - A_y B_x))$$

Equation 8. Cross product of two vectors in E^3

2.2 Matrices

A *matrix* is a rectangular array of m rows and n columns. The elements that conform a matrix are commonly complex numbers. In the most part of this material, they will have null imaginary parts, that is, they will be real numbers. Also a matrix may store non-real or complex numbers. This type of matrices will not be considered in this book.

If there are m rows and n columns, we say that the size of the matrix is $m \times n$, and we refer to it as an “ $m \times n$ matrix”, or just as a “rectangular matrix”. A $n \times n$ matrix is called a “square matrix” and it is said to have size n . The entry or element in the i^{th} row and j^{th} column of a matrix A of size $m \times n$, is denoted as a_{ij} (Equation 9).

NOTATION: Matrices will be written with capital letters A, B, C ...etc.

The matrix A of size $m \times n$ is frequently abbreviated as $A = (a_{ij})_{m \times n}$.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{1j} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{2j} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{i1} & a_{i2} & a_{ij} & \dots & a_{in} \\ a_{m1} & a_{m2} & a_{mj} & \dots & a_{mn} \end{bmatrix}$$

Equation 9. Representation of an Matrix $m \times n$

2.2.1 Properties of matrices

$$\begin{aligned} M &= I \cdot M = M \cdot I \\ 0 &= M \cdot 0 = 0 \cdot M \\ M_3 &= M_2 \cdot M_1 \neq M_1 \cdot M_2 \\ M &= M_3 \cdot (M_2 \cdot M_1) = (M_3 \cdot M_2) \cdot M_1 \\ M_3 \cdot (M_2 \cdot M_1) &= M_3 \cdot M_1 \cdot M_2 \end{aligned}$$

Figure 10. Properties with matrices

2.2.2 Operations with Matrices

2.2.2.1 Addition and subtraction

Only matrices with the same size can be added or subtracted. If A and B are both $m \times n$ matrices, their addition or subtraction is a matrix resulting $m \times n$ of adding or subtracting the corresponding entries in each matrix.

If $A = (a_{ij})_{m \times n}$ and $B = (b_{ij})_{m \times n}$ their addition or subtraction is:

$$\begin{aligned} (A + B) &= (a_{ij} + b_{ij})_{m \times n} \\ (A + B) &= (B + A) \\ (A - B) &= -(B - A) \end{aligned}$$

2.2.2.2 Neutral element

The *zero* matrix $m \times n$ denoted by O , is a matrix of size $m \times n$ with each entry equal to zero. In such a case

$$(A+O) = A = (O+A)$$

For every matrix A of size $m \times n$, the zero matrix is the neutral element of the matrix addition operation for the set of $m \times n$ matrices.

2.2.2.3 Scalar product

The scalar product of a matrix A with a real number k is defined as a new matrix. Each entry of kA is equal to the product of the real number with the corresponding entry in the original matrix A Figure 11.

If $A = (a_{ij})_{m \times n}$ and k is any real number, then the scalar product of A and k is (Equation 10).

$$kA = (ka_{ij})_{m \times n}$$

Equation 10. Scalar product of a matrix with a real number

$$\text{if } k = 3 \quad \text{and} \quad A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \Rightarrow kA = \begin{bmatrix} 3 & 6 \\ 9 & 12 \end{bmatrix}$$

Figure 11. Example of Multiplying a matrix by a scalar

2.2.2.4 Multiplication of matrices

In order to evaluate the product AB of two matrices A and B , it is required that the number of columns of A be equal to the number of rows in B .

If $A = (a_{ij})_{m \times n}$ is a matrix of size $m \times n$ and $B = (b_{ij})_{n \times p}$ is a matrix of size $n \times p$, then c_{ij} is the *dot product* of the i^{th} row vector of A , a_{ij} with the j^{th} column vector of B , b_{ij} .

Therefore, $c_{ij} = a_{i1} b_{1j} + a_{i2} b_{2j} + \dots + a_{in} b_{nj} = a_i \bullet b_j$

The order of the product $C = AB$ is: $A_{m \times n} B_{n \times p} = C_{m \times p}$

$$A = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 138 \end{bmatrix}, \quad B = \begin{bmatrix} 6 & 7 & 8 \\ 9 & 10 & 11 \\ 12 & 13 & 14 \end{bmatrix}, \quad A.B = \begin{bmatrix} 33 & 36 & 39 \\ 1710 & 1855 & 2000 \end{bmatrix}$$

Figure 12. Example of multiplication of matrices

2.2.2.5 Identity matrix

To every square matrix of size n corresponds a multiplicative neutral element. This is, there exists a unique matrix I_n of size $n \times n$ such that

$$(AI_n) = (I_nA) = A$$

For any matrix A of size $n \times n$, it is said that I_n is the identity matrix of size n , or simply the *Identity matrix* (Equation 11).

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Equation 11. Identity Matrix

In general, the identity will be simply noted as I , with its dimensions implicit from the context.

2.2.2.6 Transposed matrix

Let matrix $A = (a_{ij})_{m \times n}$, then the transposed matrix of A , denoted by $B = A^T$ is defined as $B = (b_{ij})_{n \times m}$ where $b_{ij} = a_{ji}$ for every ij .

$$A = \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 1 \end{bmatrix} \Rightarrow A^T = B \Rightarrow B = \begin{bmatrix} 2 & 5 \\ 3 & 6 \\ 4 & 1 \end{bmatrix}$$

Figure 13. Example of a matrix transposition

2.2.2.6.1 Symmetric matrix

The matrix $A = (a_{ij})_{n \times n}$ is called symmetric if $A = A^T$. This is, $a_{ij} = b_{ji}$ for every ij .

A *symmetric matrix* is symmetric across its main diagonal. Therefore, the main diagonal of A and A^T are the same. The main diagonal of a square of order n is formed by the elements $a_{11}, a_{22}, a_{33}, \dots, a_{nn}$.

2.3 Homogeneous coordinates

A cartesian point $[x, y, z]$ in E^3 can be represented by a quadruple $[x, h, y, h, z, h, h]$ for $h \in \mathbf{R}$. The quadruple $[x, h, y, h, z, h, h]$ is said to be in *homogeneous coordinates*. The following remarks apply:

- From a point $[x_h, y_h, z_h, w]$ in homogeneous coordinates the corresponding cartesian point in E^3 can be retrieved via $[x_h/h, y_h/w, z_h/w]$ for $w \neq 0$.
- A point $p_h = [x_p, y_p, z_p, w]$ in homogeneous coordinates admits any $w \in \mathbf{R}, w \neq 0$. In this material, it is chosen $w = 1$.
- If $w = 1$, the point $p_h = [x, y, z, 1]$ in homogeneous coordinates corresponds to $p = [x, y, z]$ in E^3 cartesian coordinates.
- A vector v_h in homogeneous coordinates can be seen as the subtraction between two homogeneous points; $[x_2, y_2, z_2, 1] - [x_1, y_1, z_1, 1]$. Therefore, it is written as $v_h = [v_x, v_y, v_z, 0]$, and it represents the cartesian vector $v = [v_x, v_y, v_z]$ in E^3 .

The formulation of transformations in homogeneous coordinates allows a unified mathematical treatment of all of them (rotations, translations, projections, etc.). Such a unique representation is not possible with cartesian coordinates, as seen later in Chapter 3 "Geometric Transformations".

2.3.1 Point

A point (x, y, z) in homogeneous coordinate form is illustrated in Equation 12.

$$P_H = \begin{bmatrix} X \\ Y \\ Z \\ \mathbf{a} \end{bmatrix}$$

Equation 12. Homogeneous coordinate form of a point. For effects of CAD/CAM/CG applications $\mathbf{a} = 1$ is used.

2.3.2 Vectors

A vector $[v_x, v_y, v_z]$ in homogeneous coordinate form is illustrated in Equation 13.

$$V_H = \begin{bmatrix} V_x \\ V_y \\ V_z \\ 0 \end{bmatrix}$$

Equation 13. Homogeneous coordinate form of a vector

2.3.3 Matrices

If M and P are non-homogeneous matrix and point respectively, and if $P_2 = M \cdot P$, the same operation in homogeneous coordinates would be

$$\begin{bmatrix} P_2 \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} M & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \cdot \begin{bmatrix} P \\ 1 \end{bmatrix}$$

Therefore M_H represents the homogeneous form of M (Equation 14).

$$M_H = \left[\begin{array}{ccc|c} M & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

Equation 14. Homogeneous coordinate form of a matrix

NOTE: For the purpose of this book, points in homogeneous coordinate form will always have a 1 at the end. Vectors in homogeneous coordinate form will always have a 0 at the end.

2.4 Eigenvalues and Eigenvectors

Given a square matrix $A = [a_{jk}]$. The eigenvalues λ and eigenvectors x satisfy the equation::

$$Ax = \lambda x, \quad x \neq 0$$

Where I is a specific scalar (real or complex number), and x is a specific vector. A scalar I , like the one in the previous equation for a vector $x^{-1} \theta$, is called an *eigenvalue* of A , and the vector x is called an *eigen vector* of A corresponding to the *eigenvalue* I .

The previous equation can also be written as:

$$(A - I)x = 0$$

These are n unknown, algebraic, linear equations x_1, \dots, x_n (the components of x). For these equations to have solutions $x^{-1} \theta$, their matricial coefficient $(A - I)$ must be singular. For example, for $n = 2$, the matrix would be:

$$\begin{bmatrix} a_{11} - I & a_{12} \\ a_{21} & a_{22} - I \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

In components we have:

$$(a_{11} - I)x_1 + a_{12}x_2 = 0 \quad \text{and} \quad a_{21}x_1 + (a_{22} - I)x_2 = 0$$

$A - I$ is singular if and only if the determinant $\det(A - I)$, known as the characteristic polynomial of A , is zero.

$$\det(A - I) = \begin{vmatrix} a_{11} - I & a_{12} \\ a_{21} & a_{22} - I \end{vmatrix} = I^2 - (a_{11} + a_{22})I + a_{11}a_{22} - a_{12}a_{21} = 0$$

Equation 15. Characteristic Polynomial of a matrix A .

This equation is used to find the values of I . Since it is a polynomial with real coefficients, it may have complex solutions. In such a case, each complex solution $I = a + ib$ ($a, b \in R$) guarantees that $I^* = a - ib$ is also a solution. The same holds for the eigenvectors: $x = c + id$, $c - id$ are solutions of $Ax = Ix$.

2.5 Norms of vectors and matrices

In general the “size” of a vector has been measured with the traditional Pythagorean distance $|v| = \sqrt{v^T \cdot v}$. However, there are many estimators for the size of a vector or matrix, suited for the particular application at hand (see Table 9). In control theory, is common the use of $\| \cdot \|_Y$ for such an estimation. In CAGD (Computer Aided Geometric Design) the norm of a vector (or point) will remain as $|v| = \sqrt{v^T \cdot v}$ while the norm of a matrix will, in general be $\|A\| = \det(A)$.

Table 9. Norms of vectors and matrices

NORMS OF A VECTOR	NORMS OF A MATRIX
$\ X\ _n = \left(\sum X_i^n \right)^{1/n}$ $\ X\ _2 = \left(\sum X_i^2 \right)^{1/2}$ $\ X\ _\infty = \lim_{X \rightarrow \infty} \left(\sum X_i^x \right)^{1/x}$	$\ A\ = \sum \lambda_i$ $\ A\ = \sum \text{Diag.}(A)$ $\ A\ = \det(A)$

3. GEOMETRIC TRANSFORMATIONS

Objects in Computer Aided Geometric Design are expressed by describing two aspects: geometry and topology. Geometry is responsible for giving dimensions and position to the particular object at hand. Therefore, geometric transformations change the dimensions and/or positions of such objects.

Geometric transformations that only change position of the object are called rigid, while those that change dimensions, proportions or angles of the object components are non-rigid. In order to evaluate the effects of transformations on the geometries of objects, formalism is required to express those geometries. This formalism is the one of coordinate frames (or systems). It is used in two ways: (i) To locate an object in the E^3 space, a basic, omnipresent coordinate frame is defined, called the World Coordinate System (WCS), usually noted $[X_w, Y_w, Z_w, O_w]$ and discussed below. (ii) An instantaneous coordinate system, usually called $S_i = [X_i, Y_i, Z_i, O_i]$ is anchored to an object, in arbitrary position within it. A geometric transformation of the object is totally determined if one knows the effects that it performs on that coordinate system S_i , independently of how complex or large is the object.

3.1 Definitions of mathematical entities

Definition. Coordinate Frame or System

A coordinate frame $S_i = [X_i, Y_i, Z_i, O_i]$ is formed by three vectors X_i, Y_i, Z_i applied in point O_i (Figure 14), with X_i, Y_i, Z_i linearly independent.

3.1.1 Canonical right handed or dexterous coordinate frame

A coordinate frame $S_i = [X_i, Y_i, Z_i, O_i]$ is formed by three vectors X_i, Y_i, Z_i applied in point O_i (Figure 14). S_i is right handed (or dexterous) system if it holds (a), (b) and (c) below:

- a) $\|X_i\| = \|Y_i\| = \|Z_i\| = 1$
- b) $X_i \cdot Y_i = Y_i \cdot Z_i = X_i \cdot Z_i = 0$
- c) $X_i \times Y_i = Z_i$

The previous characteristics are equivalent to say that $\det([X_i, Y_i, Z_i]) = +1$.

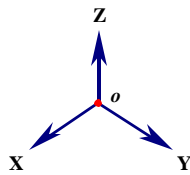


Figure 14. Dexterous Coordinate System.

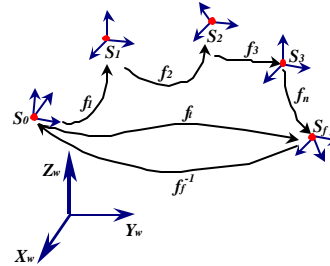


Figure 15. Sequence of geometric transformations.

In domains of mechanical engineering, computer graphics, CAD, etc., a canonical, right handed coordinate system is attached to an object undergoing a transformation $f(\cdot)$. For example, the system $[x_0, y_0, z_0, O_0]$ in Figure 17. This system, chosen to be the canonical, right handed one only for the sake of simplicity, replaces

the whole object for the purposes of characterizing and testing the transformations proposed. Therefore, the coordinate system suffers the changes caused by the transformation $f()$ (it moves, rotates, or is deformed). Moreover, defining the effects of the transformation on the coordinate system $[x_0, y_0, z_0, O_0]$ guarantees that such effects will be suffered by the object as a whole. Conversely, if the transformation $f()$ were unknown, it could be determined from the initial and final coordinate systems $[x_0, y_0, z_0, O_0]$ and $[x_f, y_f, z_f, O_f]$ respectively. Therefore:

$$S_f = [X_f \ Y_f \ Z_f \ O_f] = f[X_0 \ Y_0 \ Z_0 \ O_0] = f \cdot S_0$$

3.1.2 Geometric transformations and matrix notation

Geometric transformations are functions $f()$ that transform coordinate systems (see Figure 15). Therefore $f(S_i) = S_{i+1}$ implies that S_i was transformed by $f()$ to yield S_{i+1} . It turns out that $f()$ is a linear function, and its application can be expressed as $f \cdot S_i = S_{i+1}$, where f is a matrix, whose dimensions are such that matrix pre-multiplication with S_i is possible.

As geometric transformations are expressed in matrix notation, the following observations are relevant:

- Dimensions of S_i and S_{i+1} are the same, and thus, f must be a square matrix.
- Notation $f(g(S_i)) = f \cdot g \cdot S_i$ means that $g()$ is applied on S_i first, and then $f()$ is applied on the result $g(S_i)$.
- Transformations are not commutative: $f(g(S_i)) = f \cdot g \cdot S_i \neq g \cdot f \cdot S_i = g(f(S_i))$.
- Transformations are associative: $(f_1 \cdot f_2 \cdot f_3) \cdot S_i = (f_1 \cdot (f_2 \cdot f_3)) \cdot S_i = ((f_1 \cdot f_2) \cdot f_3) \cdot S_i = f_1 \cdot (f_2 \cdot (f_3 \cdot S_i))$.
- $f \cdot S_i = S_{i+1}$ means that f transforms S_i into S_{i+1} . If a transformation g exists which transforms S_{i+1} back to S_i , one says that g inverts the effect of f . Mathematically,
 $g = f^{-1}$ (g inverts the effect of f)
 $f = g^{-1}$ (f inverts the effect of g)
 $f \cdot g = g \cdot f = I$ and thus $f \cdot g \cdot S_i = g \cdot f \cdot S_i = S_i$ (f and g applied consecutively have no effect: they leave the object unchanged).
- Any sequence of transformations $f_1, f_2, f_3, \dots, f_n$ applied to an object S_0 in sequence f_1, f_2 , etc., can be clustered into one equivalent transformation $f_n \cdot f_{n-1} \dots f_2 \cdot f_1 = f_i$ which condenses the total information on the transformation chain. Therefore, $(f_n \cdot f_{n-1} \dots f_2 \cdot f_1) \cdot S_0 = f_i \cdot S_0 = S_f$.
- A transformation chain $f_1, f_2, f_3, \dots, f_n$ applied on S_0 to produce S_f , $(f_n \cdot f_{n-1} \dots f_2 \cdot f_1) \cdot S_0 = f_i \cdot S_0 = S_f$ implies that f_i is applied on S_{i-1} to produce an intermediate result S_i ($f_i \cdot S_{i-1} = S_i$). The last result of this sequence is S_f .

3.2 Rigid Transformations about World Coordinate Axes

Rigid transformations are widely used in kinematics and robotics, since most of the mechanical devices can be modeled as non-deformable under operating conditions for analyzing kinematic variables (position, velocity and acceleration).

3.2.1 Translation

In a translation the objects undergo a displacement, defined by a vector $[\Delta_x, \Delta_y, \Delta_z]$ as seen in the World Coordinate System $[X_w, Y_w, Z_w, O_w]$ (Figure 16). Observe the movement of the coordinate system $[X_0, Y_0, Z_0, O_0]$ towards $[X_f, Y_f, Z_f, O_f]$ (Z_0 and Z_f not shown for the sake of clarity).

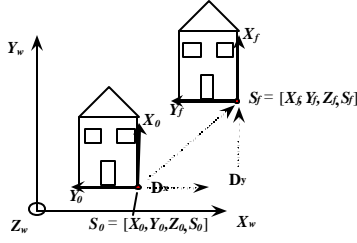


Figure 16. Translation transformation

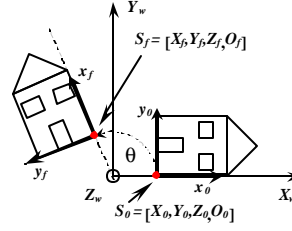


Figure 17. Rotation transformation about main axis Zw.

The coordinates of the object change as:

$$x_2 = x_1 + D_x \quad y_2 = y_1 + D_y \quad z_2 = z_1 + D_z$$

which can be written in Cartesian coordinates form as shown in Equation 16. However, to express the translation as a multiplication rather than as an addition, homogeneous coordinates are used, as in Equation 17.

$$\begin{bmatrix} \hat{e}x_2 \\ \hat{e}y_2 \\ \hat{e}z_2 \\ \hat{e}1 \end{bmatrix} = \begin{bmatrix} \hat{e}x_1 \\ \hat{e}y_1 \\ \hat{e}z_1 \\ \hat{e}1 \end{bmatrix} + \begin{bmatrix} D_x \\ D_y \\ D_z \\ 0 \end{bmatrix}$$

Equation 16. Translation transformation in Cartesian Coordinates.

$$\begin{bmatrix} \hat{e}x_2 \\ \hat{e}y_2 \\ \hat{e}z_2 \\ \hat{e}1 \end{bmatrix} = \begin{bmatrix} \hat{e}1 & 0 & 0 \\ \hat{e}0 & 1 & 0 \\ \hat{e}0 & 0 & 1 \\ \hat{e}1 & 0 & 0 \end{bmatrix} \begin{bmatrix} D_x \\ D_y \\ D_z \\ 0 \end{bmatrix} + \begin{bmatrix} \hat{e}x_1 \\ \hat{e}y_1 \\ \hat{e}z_1 \\ \hat{e}1 \end{bmatrix} \quad f = \text{trans}(D_x, D_y, D_z) = \begin{bmatrix} \hat{e}1 & 0 & 0 \\ \hat{e}0 & 1 & 0 \\ \hat{e}0 & 0 & 1 \\ \hat{e}1 & 0 & 0 \end{bmatrix}$$

Equation 17. Translation transformation in Homogeneous Coordinates

3.2.2 Rotations about main axes.

In this section it is assumed that the axis of rotation is one of X_w , Y_w , or Z_w , and therefore the pivot point is the origin O_w (see Figure 17). The angles have plus or minus sign, according to the right hand rule applied using the corresponding rotation axis. Therefore, a rotation by q around axis X_w is not the same as a rotation of q around axis $-X_w$. It is customary to always let the axis be one of X_w , Y_w , or Z_w , and to assign the sense of the rotation to q . As an example of rotation, if point (x_1, y_1, z_1) is rotated around Z_w by an angle q , the rotated point (x_2, y_2, z_2) will be:

$$\begin{aligned} x_2 &= R \cos(a+q) = R \cdot \cos(a) \cdot \cos(q) - R \cdot \sin(a) \cdot \sin(q) & x_2 &= x_1 \cdot \cos(q) - y_1 \cdot \sin(q) \\ y_2 &= R \sin(a+q) = R \cdot \cos(a) \cdot \sin(q) + R \cdot \sin(a) \cdot \cos(q) & y_2 &= x_1 \cdot \sin(q) + y_1 \cdot \cos(q) \\ z_2 &= z_1 \end{aligned}$$

In matrix form (Equation 18):

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(q) & -\sin(q) & 0 & 0 \\ \sin(q) & \cos(q) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \text{Rot}(Z_w, q) \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}$$

Equation 18. Matrix form of Rotation around the Zw axis.

$$Rot(Z_w, ?) = \begin{bmatrix} \cos(?) & -\sin(?) & 0 & 0 \\ \sin(?) & \cos(?) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 19. Rotation Matrix around Zaxis.

in similar way, a rotation around X_w is (Equation 20):

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(?) & -\sin(?) & 0 \\ 0 & \sin(?) & \cos(?) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = Rot(X_w, \mathbf{q}) \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}$$

Equation 20. Rotation transformation around the X_w axis.

$$Rot(X_w, \mathbf{q}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(?) & -\sin(?) & 0 \\ 0 & \sin(?) & \cos(?) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 21. Rotation Matrix around X_w .

in similar way, a rotation around Y_w is (Equation 22):

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(?) & 0 & \sin(?) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(?) & 0 & \cos(?) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = Rot(Y_w, \mathbf{q}) \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}$$

Equation 22. Rotation transformation around axis Y_w

$$Rot(Y_w, \mathbf{q}) = \begin{bmatrix} \cos(?) & 0 & \sin(?) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(?) & 0 & \cos(?) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 23. Rotation matrix around Y_w

3.2.3 Rotations parallel to world axes.

Until now, rotations were performed around the coordinate axes X_w , Y_w or Z_w of the World Coordinate System. This implies in particular, axes pivoted in the origin $(0,0,0)$. However, it is common to require rotations around arbitrary axes, parallel to X_w , Y_w or Z_w , anchored in an arbitrary point $\mathbf{pv} = [p_x, p_y, p_z]$ in E^3 . In such a case, the procedure is:

- Translate the pivot point \mathbf{pv} to the origin. This means, translate S_θ by $\mathbf{M}_I = \mathbf{trans}(p_x, p_y, p_z)$. Or in other words: $S_I = \mathbf{M}_I * S_\theta$.

- b) Apply the prescribed rotation as in Equation 18 to Equation 22. The transformation would be $M_2 = Rot(axis, q)$. Now, $S_2 = M_2 * S_1$.
- c) Translate the pivot point back to its original position. This means, translate the object by $M_3 = trans(p_x, p_y, p_z)$. The final position would be $S_3 = M_3 * S_2$.

Therefore, the whole transformation would be: $M_t = M_3 * M_2 * M_1$, and $S_3 = M_3 * M_2 * M_1 * S_0 = M_t * S_0 = S_f$.

The sequence to transform S_0 into S_f can be also written as:

$$S_f = (T^{-1} \cdot R_{axis}(?) \cdot T) \cdot S_0 \quad \text{with} \quad M_t(?) = T^{-1} \cdot R_{axis}(?) \cdot T$$

Equation 24. Decomposition of a rotation about arbitrary axis in E^3

This equation happens to be applicable to a general rotation in E^3 .

3.2.4 Rigidity of Transformation vs Canonical Right Handed Systems.

- i) In cartesian coordinates, a rotation R applied on a point (X, Y, Z) , followed by a translation T on the intermediate result, is written as:

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \left(R \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \right) + T \quad \text{with } R(3 \times 3), T(3 \times 1) \text{ matrices.}$$

This effect can be packed into one (matrix multiplication) operation in homogeneous coordinates (for explanation of “homogeneous coordinates” refer to Chapter 2 “Linear Algebra”):

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & & & T \\ R & & & \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = M \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}$$

Equation 25. A “homogeneous coordinate form” transformation: Rotation followed by translation

M represents a rotation followed by a translation, in this order. From previous discussion, it is known that its effect is different to a translation followed by a rotation.

- ii) If M is a pure rotation, then $T = 0$ ($T = [0,0,0]^T$ (3x1) vector). If M is a pure translation, then $R = I_{3 \times 3}$, (the (3x3) identity matrix).
- iii) A rigid transformation (Equation 25) is identified because its submatrix $R = [U_1, U_2, U_3]$, with each U_i of dimension (3×1) , satisfies:

$$\left. \begin{array}{l} a) \quad |U_1| = |U_2| = |U_3| = 1 \\ b) \quad U_1 \wedge U_3, U_1 \wedge U_2, U_2 \wedge U_3 \\ c) \quad U_1 \wedge U_2 = U_3 \end{array} \right\}$$

Equation 26. Conditions for a rigid transformation.

Where condition (i) suggests that the matrix does not amplify the dimensions of objects. Condition (ii) relates to maintaining angular relations unchanged. And (iii) says that the matrix keeps the right handedness of coordinate systems. The reader may observe the similarity between the conditions for transformation rigidity and the specifications of canonical right handed systems. Although this

similarity is not an accident, to establish the connection is beyond the scope of this material. It must be kept in mind that the rigidity of a M transformation must be tested with the conditions (i), (ii), (iii) in Equation 26.

- iv) If M is rigid, its application does not deform the object. It only changes its position in E^3 .
- v) The rigidity of M is stored in its R submatrix and not in the translation part T .
- vi) The transformation of system S_i into S_{i+1} by a transformation M is written as:

$$S_{i+1} = M \cdot S_i = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \cdot S_i$$

Equation 27. Transformation of coordinate system by Rigid Transformation.

It is important to keep in mind that the rigidity is a characteristic of the transformation M , while canonical dexterity is a characteristic of the coordinate system S_i or S_{i+1} , that is, the data transformed by the function M (Equation 27). If the S_i system is canonical right handed, and M is rigid, then S_{i+1} would be canonical right handed. If the S_i system is canonical right handed, and S_{i+1} is canonical right handed, the transformation that changes S_i into S_{i+1} is rigid. The reader is invited to think how to diagnose the rigidity of M given only the information of S_i and S_{i+1} , (assume they are non-canonical right handed systems).

3.2.5 The non-commutative group of geometric transformations

Consider M the set of rigid geometric transformations, a, b, c, d elements of M and \oplus the composition operation on transformations of M . The following properties make of M a (non-commutative) group because:

1. $\forall a, b, \in M \quad a \oplus b = c \in M$ composition of transformations is a transformation.
2. $\forall a \in S, \exists N \in M \quad a \oplus N = N \oplus a = a$ there exist a null transformation in M .
3. $\forall a, b, c \in M \quad (a \oplus b) \oplus c = a \oplus (b \oplus c)$ composition is associative.
4. $\forall a \in S, \exists a^{-1} \in M \quad a \oplus a^{-1} = a^{-1} \oplus a = N$ every rigid transformations has an inverse in M .

By applying these concepts to geometric transformations we have that:

- a) The null transformation is:

$$N = \begin{bmatrix} I_{3 \times 3} & 0_{3 \times 1} \\ 0_{1 \times 3} & I_{1 \times 1} \end{bmatrix}$$

Since it is formed by a null rotation ($I_{3 \times 3}$) and the null translation $0_{3 \times 1}$.

- b) The inverse transformation of a given M (rigid) satisfies:

$$M = \begin{bmatrix} \hat{e} R & T \hat{u} \\ \hat{e} 0_{1 \times 3} & I_{1 \times 1} \hat{u} \end{bmatrix} \quad \text{P} \quad M^{-1} = \begin{bmatrix} \hat{e} R^{-1} & -R^{-1} \cdot T \hat{u} \\ \hat{e} 0_{1 \times 3} & I_{1 \times 1} \hat{u} \end{bmatrix} \quad \text{since } R^{-1} \text{ exists}$$

Equation 28. Inverse of a rigid transformation.

It is evident from Equation 28 that the inverse of a pure translation and a pure rotation are respectively:

$$T_{inv} = \begin{bmatrix} \hat{e} I_{3 \times 3} & -T \hat{u} \\ \hat{e} 0_{1 \times 3} & I_{1 \times 1} \hat{u} \end{bmatrix}, \quad R_{inv} = \begin{bmatrix} \hat{e} R^{-1} & 0_{3 \times 1} \hat{u} \\ \hat{e} 0_{1 \times 3} & I_{1 \times 1} \hat{u} \end{bmatrix}$$

From now on, the notation $I_{3 \times 3}$ will be replaced simply by I , with the dimensions of the matrices implicit in the equation.

3.2.6 Example. Rigid transformations.

1. Determine the chain of rigid transformations (translations and rotations parallel to the World Axes) required to bring the object **BODY₀** from the initial to the final position (**BODY_f**).
2. Program the example in MATLAB.

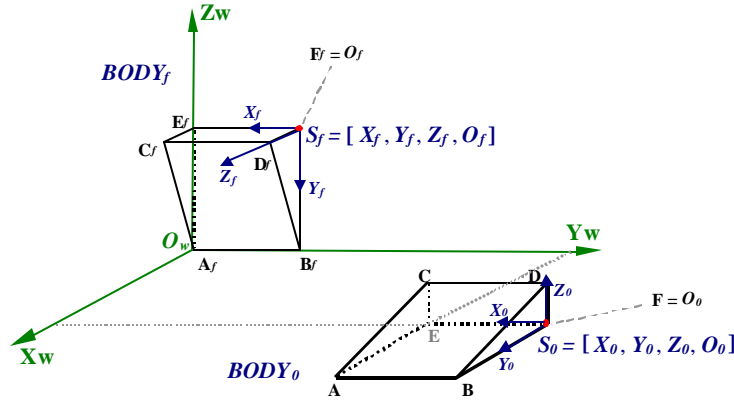


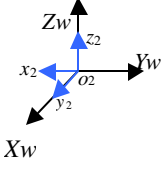
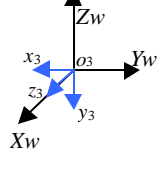
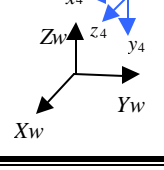
Figure 18. Example of rigid transformations.

The method of solution is the following:

- a. Assign coordinates $[x, y, z, 1]$ to each one of the vertices of the object, in initial and final positions. Ensure that initial (**BODY₀**) and final objects (**BODY_f**) are consistent.
- b. Attach a canonical coordinate frame S_0 to the object in the initial position. Find the numerical values for $S_0 = [X_0, Y_0, Z_0, O_0]$. Verify that S_0 is a canonical dexterous frame.
- c. Draw and find numerical values for the frame corresponding to the transformed S_0 in the final configuration. Verify that S_f is a canonical dexterous frame.
- d. Perform the elementary transformations to bring S_0 to intermediate positions until S_f is reached. An initial translation to O_w is suggested, to have better control upon subsequent rotations. Fill Table 10 with the relevant data.
- e. Numerically, check that the transformation M_i effectively transforms S_{i-1} into S_i .
- f. Numerically, check that the accumulated transformation $M = M_n \cdot M_{n-1} \cdot \dots \cdot M_1$ indeed transforms S_0 into S_f .
- g. Transform object **BODY₀** into **BODY_f** by application of M .
- h. Program the procedure above, by defining the functions *translation_matrix(dx,dy,dz)* and *rotation_matrix(axis, angle)*. Define also a function *plt_axes(S,d)* which draws a coordinate system S , scaled by a factor d ($d = 1$ implies no scaling).

Solution

Table 10. Illustrative steps of a transformation sequence of an object.

#M	Transformation	Matrix	Graphical Frame S_i	Numerical Frame S_i	Rigid Y/N
M_1	$\text{trans}(-F_x, -F_y, -F_z)$	$\begin{bmatrix} 1 & 0 & 0 & -F_x \\ 0 & 1 & 0 & -F_y \\ 0 & 0 & 1 & -F_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$		$\begin{matrix} x_2 & y_2 & z_2 & O_2 \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix}$	Y
M_2	$\text{rot}(Y_w, 90^\circ)$	$\begin{bmatrix} \cos(90) & 0 & \sin(90) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(90) & 0 & \cos(90) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$		$\begin{matrix} x_3 & y_3 & z_3 & O_3 \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix}$	Y
M_3	$\text{trans}(F_{xf}, F_{yf}, F_{zf})$	$\begin{bmatrix} 1 & 0 & 0 & F_{xf} \\ 0 & 1 & 0 & F_{yf} \\ 0 & 0 & 1 & F_{zf} \\ 0 & 0 & 0 & 1 \end{bmatrix}$		$\begin{matrix} x_4 & y_4 & z_4 & O_4 \\ \begin{bmatrix} 0 & 0 & 1 & F_{xf} \\ -1 & 0 & 0 & F_{yf} \\ 0 & -1 & 0 & F_{zf} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix}$	Y

The development of this example and evolution of coordinate systems in MATLAB are illustrated in Appendix, section 7.1.1.

3.3 Rigid Transformations about arbitrary directions. Quaternion Method

Until now, rotations were performed around the coordinate axes X_w , Y_w or Z_w (World Coordinate System). However, it is common to require rotations around arbitrary axis $L = [v, pv]$ (with direction v and pivot point pv) in E^3 . As a first approach, in this section the method is presented to rotate an object around an axis in space with arbitrary orientation but passing through the origin (see Figure 19). Then, the method will be extended to rotated objects around axes with arbitrary direction, and anchored in an arbitrary pivot point in E^3 .

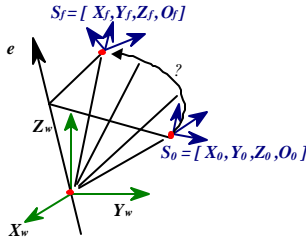


Figure 19. Rotation by a ? angle about arbitrarily inclined axis passing through the origin.

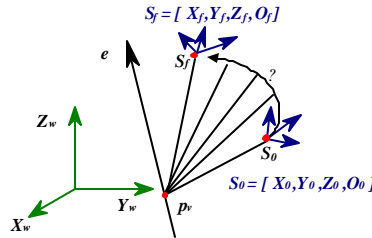


Figure 20. Rotation about an arbitrary axis $L=[e,pv]$ by a ? angle.

3.3.1 Rotations about arbitrary axes pivoted in the origin.

Assume that $L = [e = (e_x, e_y, e_z), O = (0,0,0)]$ is an infinite line in space E^3 , with direction $e = (e_x, e_y, e_z)$ (unit vector) and pivot point $O = (0,0,0)$. A rotation of a system S_0 by ? radians around L (positive ? means counterclockwise sense with respect to e) is possible by applying the quaternion formula to rotation of a point $p_0 = (x, y, z)$, Equation 29 ([KOR.85]):

$$p_f = p_o + (2F(?)) \cdot (e_i \times p_o) + 2 \cdot (e_i \times (e_i \times p_o))$$

Equation 29. Rotation of a point about an origin-pivoted axis calculated by the quaternion method.

Where:

?	Rotation angle (in radians, positive with respect to e).
$e_i = \sin(?/2) \cdot e$	Orientation of the rotation axis.
$e = (e_x, e_y, e_z)$	Orientation of the rotation axis. Unit vector.
p_0	Initial position
p_f	Final position
$F(?)$	$\cos(?/2)$

Observe that Equation 29 rotates points. The reader is invited to

- Prove that the same formula applies in rotating vectors
- From (a), deduce the procedure to rotate coordinate frames S_0 about L by ? radians to obtain S_f .
- From (a) and (b), deduct the procedure to compute the rotation about L by an angle ? (which are given) in matrix form, $M?$. Observe that Equation 29 is not a transformation in matrix form as are Equation 16 to Equation 22. Suggestion: remember that $S_f = M? \cdot S_0$.

3.3.2 Identification of axis and angle of origin-based rotations. Eigenvalue Method.

Assume that M_θ is a rotation by an angle θ about a unitary axis e anchored in the origin. From the quaternion method, it is possible to deduce M_θ from θ and e . A converse question is: given a known M_θ how to find θ and e . Without proof, the following procedure is given below. The interested reader may refer to [BOT.79]:

1. Arrange M_θ in form $M_\theta = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$. In this case, $T = 0$.
2. Extract from R its eigenvalues λ_i and eigenvectors v_i . $\lambda_i = [\lambda_1, \lambda_2, \lambda_3]$, $V = [v_1, v_2, v_3]$. Notice that $R.v_i = \lambda_i v_i$.
3. Given R , a rigid transformation, it will have one eigenvalue, suppose λ_1 , equal to 1. v_1 , its corresponding eigenvector, is the (non unitary) vector of the rotation axis. Make $e = v_1/|v_1|$. See Equation 30.

$$\theta = [1, \cos(\theta) + i.\sin(\theta), \cos(\theta) - i.\sin(\theta)] \quad , \quad V = [v_1 \quad v_2 \quad v_3] = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} \begin{bmatrix} a + i.b \\ c + i.d \\ e + i.f \end{bmatrix} \begin{bmatrix} a - i.b \\ c - i.d \\ e - i.f \end{bmatrix}$$

Equation 30. Eigenvalue and eigenvector matrices.

4. The other two eigenvalues, say λ_2 and λ_3 , will have the following form: $\lambda_2, \lambda_3 = \cos(\theta) \pm i.\sin(\theta)$. Calculate θ .
5. Because e and $-e$ are the same eigenvector of R , test that the angle θ found is the one that corresponds to the right handed rotation using the e found. Correct the sign of θ if necessary.
6. $[e, \theta]$ are the direction of the rotation axis and the rotation angle respectively. The pivot point of the rotation axis is (0,0,0), according to the hypothesis.

3.3.3 Rotations about arbitrary axes pivoted outside the origin.

Assume that $L = [e = (e_x, e_y, e_z), pv = (pv_x, pv_y, pv_z)]$ is an infinite line in space E^3 , with direction $e = (e_x, e_y, e_z)$ (unit vector) and pivot point $pv = (pv_x, pv_y, pv_z)$, as in Figure 20. A rotation of a coordinate system S_0 about L by θ radians to obtain S_θ is performed with the following sequence:

1. Bring pv to the origin by applying $M_1 = trans(-pv_x, -pv_y, -pv_z)$. Therefore $S_1 = M_1 S_0$.
2. Apply the quaternion method to rotate S_1 around an axis with direction e , passing through the origin. Therefore $S_2 = q(e, \theta, S_1)$, where $q(\cdot)$ is the transformation in Equation 29.
3. Take pv back to its original place by applying $M_3 = trans(pv_x, pv_y, pv_z)$. Therefore $S_3 = M_3 S_2$.
4. The total transformation of S_0 is written as :

$$S_3 = M_3.q(e, \theta, S_1).M_1.S_0 = trans(+pv).q(e, \theta, trans(-pv).S_0)$$

Equation 31. General rotation by θ about an arbitrary line $L=(e,pv)$ in E^3

3.3.4 Identification of axis and angle of arbitrary rotations. Eigenvalue Method.

Given arbitrary initial and final positions of an object, determined by coordinate frames $S_0 = [X_0, Y_0, Z_0, O_0]$ and $S_f = [X_f, Y_f, Z_f, O_f]$ respectively, one needs to find $L = [e, pv]$, the instantaneous rotation axis in E^3 that takes S_0 onto S_f . e is the unit vector direction of the rotation axis, pv is the instantaneous center of rotation, and θ is the rotation angle (Figure 21). Notice that pv is a special point on L , in contrast with the situation of Figure 20. Without proof, the following procedure is given:

1. Find M , recalling that: $S_f = M \cdot S_0 \cdot P$ $M = S_f \cdot S_0^{-1}$, since S_0^{-1} exists.
2. Place M in form $M = \begin{bmatrix} \hat{e}R & T\hat{u} \\ \hat{e}0 & I\hat{u} \end{bmatrix}$. In this case, $T \neq 0$.
3. Extract e (unit vector) and θ from R via the eigenvalue – eigenvector method.
4. Find pv by using the Equation 32.

$$p_v = O_0 + \frac{\theta_o}{2} - \frac{(\theta_o \times e)}{2 \cdot \tan(\theta/2)} \quad \text{with} \quad \theta_o = O_f - O_0$$

Equation 32. Instantaneous center of rotation for arbitrary movement in E^3 .

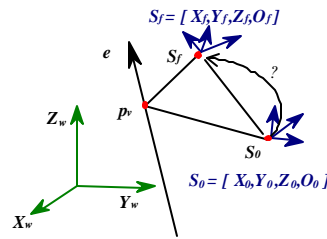


Figure 21. Rotation about an arbitrary axis $L=[e,pv]$ by a θ angle.

The reader is invited to proof:

1. That in spite of $T \neq 0$, the direction of the instantaneous rotation axes, e , and the corresponding angle θ can be determined from the eigenvalue - eigenvector method in exactly the same manner as when $T = 0$.
2. Equation 32.
3. That $S_f = M \cdot S_0$ has a solution for M , given that S_0 is a coordinate system.
4. That if $M_q = \begin{bmatrix} \hat{e}R & T\hat{u} \\ \hat{e}0 & I\hat{u} \end{bmatrix}$ performs a rotation by an angle θ about a unitary arbitrary axis e anchored in the origin, then $T = 0$. Suggestion: transform the origin.
5. Let M_t represent a chain of transformations $M_t = M_n \cdot M_{n-1} \cdot M_{n-2} \cdot \dots \cdot M_1$ with pure rotations and pure translations mixed in arbitrary order. Prove, by induction on n , that when M_t is applied on a homogeneous vector $V_n = [V \ 0]^T$, only the rotations affects V_n , while the translations “pass” through V_n .

without having any effect. The reader should find the relation between this fact and proofs (1) and (4) above. (see Appendix, section 0).

3.3.5 Example. General rigid transformations.

For the example in Figure 18:

1. Determine the equivalent instantaneous rotation axis $L = [e, pv]$ and angle θ that replicate the whole displacement. Use the eigenvalue – eigenvector method. Use Equation 29 to Equation 32.
2. Test the procedures of eigenvalue – eigenvector methods by programming the example in MATLAB. Starting from $BODY_0$ and S_0 you should obtain $BODY_f$ and S_f applying Equation 31.
3. Obtain in MATLAB a graph similar to Figure 21, including $L = [e, pv]$ and θ , as a consequence of the previous steps.

3.4 Non rigid transformations

Non-rigid transformations are widely used in visualization activities. Although many applications lay in the domain of entertainment, very important ones relate to realistic display of technical systems and scientific data from physical phenomena. If the transformation is:

$$M = \begin{bmatrix} \dot{e}R & T\ddot{u} \\ \dot{e}0 & I\ddot{u} \end{bmatrix} \quad \text{with} \quad R = [U_1 \ U_2 \ U_3], \quad U_i (3 \times 1)$$

Non-rigid transformations have a formulation that violates the conditions in Equation 26, repeated here to illustrate the discussion.

$$\begin{aligned} a) \quad & |U_1| = |U_2| = |U_3| = 1 \\ b) \quad & U_1 \wedge U_3, U_1 \wedge U_2, U_2 \wedge U_3 \\ c) \quad & U_1 \wedge U_2 = U_3 \end{aligned}$$

Although there are interactions between effects, in general the violation of condition (a) produces a scaling of the object. Violation of condition (b) produces a shear effect, by which the object becomes slanted; and violation of condition (c) changes the right-handedness of the systems associated with the object, therefore producing mirror effects. One may realize, however, that violation of (b) will also produce dimensional distortions, not only angle distortions. Therefore, failure to comply with one condition usually produces violations in the others.

3.4.1 Mirror or Reflection

Mirror transformations turn right handed systems into left handed systems and viceversa. If a right handed coordinate system is mirrored, its right-handedness property is lost, and therefore $U_1 \wedge U_2 \neq U_3$. Conditions (a) and (b) in Equation 26 remain unchanged. Mirrors can be produced about a focal point or a reflection plane. Unexpectedly, mirrors about an axis happen to be rigid transformations, being equivalent to a rotation by 180 degrees around such axis.

3.4.1.1 Mirror about a point

3.4.1.1.1 Mirror about the origin

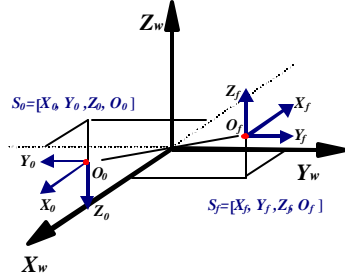


Figure 22. Mirror about the Origin

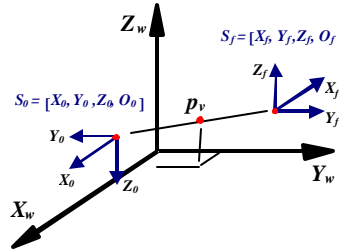


Figure 23. Mirror about a point removed from the origin.

Given a point (Figure 22) with coordinates (x_1, y_1, z_1) , its mirror image about the origin, (x_2, y_2, z_2) is:

$$x_2 = -x_1 \quad y_2 = -y_1 \quad z_2 = -z_1$$

In homogeneous matrix coordinates one has Equation 33.

$$\begin{bmatrix} \hat{e}X_2\hat{u} \\ \hat{e}Y_2\hat{u} \\ \hat{e}Z_2\hat{u} \\ \hat{e}1\hat{u} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{e}X_1\hat{u} \\ \hat{e}Y_1\hat{u} \\ \hat{e}Z_1\hat{u} \\ \hat{e}1\hat{u} \end{bmatrix} = \mathbf{Mirror}(\text{point}(0,0,0)) \times \begin{bmatrix} \hat{e}X_1\hat{u} \\ \hat{e}Y_1\hat{u} \\ \hat{e}Z_1\hat{u} \\ \hat{e}1\hat{u} \end{bmatrix} \quad \text{with} \quad \mathbf{Mirror}(\text{point}(0,0,0)) = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 33. Mirror transformation about the origin

The reader is invited to confirm that in $\mathbf{Mirror}(\text{point}(0,0,0))$, $U_1 \times U_2 = -U_3$. Therefore, it is not a rigid transformation.

3.4.1.1.2 Mirror about a point different from the origin

Figure 23 shows the mirror transformation about a point pv , different from the origin. The procedure required is:

1. Use M_1 , which brings pv to the origin, to affect S_0 : $S_1 = M_1 \cdot S_0 = \text{trans}(-pv) \cdot S_0$.
2. Apply $M_2 = \text{mirror}(\text{point}(0,0,0))$ on S_1 to obtain S_2 : $S_2 = M_2 \cdot S_1 = \text{mirror}(\text{point}(0,0,0)) \cdot S_1$.
3. Use the inverse of M_1 , to take the pivot back to pv . $S_3 = M_3 \cdot S_2 = \text{trans}(+pv) \cdot S_2$.

Trivially, the transformation required is analogous to Equation 24 or Equation 31:

$$\boxed{\mathbf{S}_f = \text{mirror}(\text{point}(p_v)) \cdot \mathbf{S}_0 = \mathbf{M}_m \cdot \mathbf{S}_0 = \text{trans}(+p_v) \cdot \text{mirror}(\text{point}(0,0,0)) \cdot \text{trans}(-p_v) \cdot \mathbf{S}_0 \Rightarrow \text{mirror}(\text{point}(p_v)) = \text{trans}(+p_v) \cdot \text{mirror}(\text{point}(0,0,0)) \cdot \text{trans}(-p_v)}$$

Equation 34. Transformation chain for mirror about an arbitrary point.

In similar way to the process carried out in the example from Figure 18, one may fill the table corresponding to the evolution of the coordinate system from Figure 23, from S_0 to S_f .

3.4.1.2 Mirror about a plane.

In literature (see for example [FOL.90, MOR.99]) mirrors or reflections around planes are usually built by writing implicit equations. In contrast, this material employs a technique derived from the numerical analysis, called Householder Reflectors, used to calculate QR factorization of matrices. It reflects rows or columns of a matrix about a particular plane ([GOL.96,TRE.97]), normal to other vectors of the matrix.

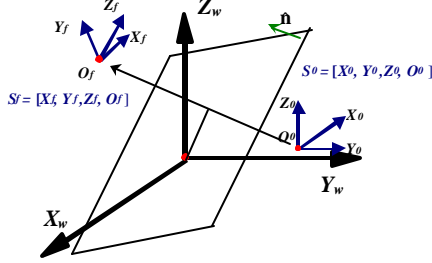


Figure 24. Mirror or reflection about a plane that crosses the origin.

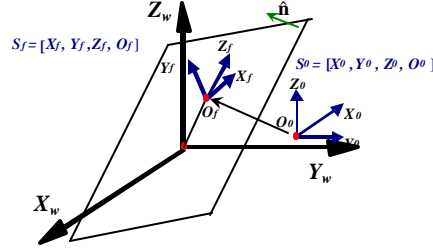


Figure 25. Parallel projection on a plane that crosses the origin

3.4.1.2.1 Householder transformation

The version of the Householder transformation given here differs in formula to the one given in the references. Given a plane $P = [n, (0,0,0)]$ passing through the origin $(0,0,0)$, with normal unit vector n , the reflection of a point p_0 across such a plane is defined as in Figure 24:

$$\begin{aligned} a) \quad & \mathbf{H}_n = \text{mirror}(\text{plane}(\hat{n}, [0,0,0])) = \mathbf{I} - 2 \cdot \hat{n} \cdot \hat{n}^T \Rightarrow \mathbf{p}_f = \mathbf{H}_n \cdot \mathbf{p}_0 \\ b) \quad & \mathbf{p}_f = \mathbf{H}_n \cdot \mathbf{p}_0 = \mathbf{p}_0 - 2 \cdot \hat{n} \cdot (\hat{n} \bullet \mathbf{p}_0) = \mathbf{p}_0 - 2 \hat{n} \cdot (\hat{n}^T \cdot \mathbf{p}_0) \end{aligned}$$

Equation 35. Householder Reflector, used to mirror a point about an origin-pivoted plane.

Equation 35 can be used either with Cartesian or homogeneous coordinates. $\mathbf{p}_0, \mathbf{p}_f$ and \mathbf{n} are column vectors, \mathbf{I} is the identity matrix and \hat{n} is unitary. Part (a) is the matrix form, while part (b) is the equivalent vector form of the equation in (a), with the dot product clearly marked (\bullet).

3.4.1.2.2 Example. Mirror on the YZ plane.

A mirror about plane **YZ** inverts the x coordinate, leaving y and z unchanged. The transformation equations are:

$$x_2 = -x_1 \quad y_2 = y_1 \quad z_2 = z_1$$

Therefore, the transformation in matrix form would be:

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} = \text{mirror}(\text{plane YZ}) \cdot \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} \quad \text{with} \quad \text{mirror}(\text{plane YZ}) = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We would like to obtain the same result by applying the Householder Reflection. The procedure and result are illustrated in Figure 26.

1. Define the unit vector normal to the plane **YZ**: $\mathbf{n} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$
2. Apply the Householder formula (Equation 35):

$$\mathbf{H}_n = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - 2 \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}_n = \text{mirror}(\text{planeYZ})$$

Figure 26. Application of Householder Reflector to get a reflection through plane YZ

The result is the same as the one obtained by formulation of the transformation equations on each coordinate. The advantage of the Householder method is that allows a systematic procedure. Figure 26 shows the application by using Cartesian vectors. The same result is obtained if homogeneous coordinates are used.

The reader may find easy solutions with simple variations of the Householder Reflector for two problems:

1. To find the parallel projection of an object onto a plane (Figure 25).
2. To find the perspective projection of an object onto a plane.

Other illustrated cases of mirror or about planes are considered in Appendix, section 0.

3.4.1.2.3 Mirror through an arbitrary plane.

The Householder transformation reflects the object through a plane passing by the origin, $(0,0,0)$. If the coordinate system S_0 must be reflected through a general plane $P = [\mathbf{n}, \mathbf{pv}]$ with normal \mathbf{n} and pivot point \mathbf{pv} , to get S_f , the procedure is analogous to the applied in mirror or rotations outside the origin:

1. Use \mathbf{M}_1 , which takes \mathbf{pv} to the origin, to affect S_0 : $S_1 = \mathbf{M}_1 \cdot S_0 = \text{trans}(-\mathbf{pv}) \cdot S_0$.
2. Apply $\mathbf{M}_2 = \text{mirror}(\text{plane}[\mathbf{n}, (0,0,0)])$ on S_1 to obtain S_2 :
 $S_2 = \mathbf{M}_2 \cdot S_1 = \mathbf{H}_n \cdot S_1 = \text{mirror}(\text{plane}[\mathbf{n}, (0,0,0)]) \cdot S_1$.
3. Use the inverse of \mathbf{M}_1 , to take the pivot back to \mathbf{pv} . $S_3 = \mathbf{M}_3 \cdot S_2 = \text{trans}(+\mathbf{pv}) \cdot S_2$.

$$\begin{aligned} S_f = \text{mirror}(\text{plane}(\mathbf{n}, \mathbf{p}_v)) \cdot S_0 &= \mathbf{M}_m \cdot S_0 = \text{trans}(+\mathbf{p}_v) \cdot \mathbf{H}_n \cdot \text{trans}(-\mathbf{p}_v) \cdot S_0 \quad \Rightarrow \\ \text{mirror}(\text{plane}(\mathbf{n}, \mathbf{p}_v)) &= \text{trans}(+\mathbf{p}_v) \cdot \mathbf{H}_n \cdot \text{trans}(-\mathbf{p}_v) \end{aligned}$$

Equation 36. Mirror transformation about a plane not crossing the origin.

3.4.2 Scaling

The scaling transformation selectively enlarges or contracts coordinates of the object S_0 , yielding a scaled object S_f . For example, S_f may have its X coordinates equal to twice the X coordinate of S_0 , while at the same time having Y equal to half of the Y coordinate in S_0 , etc.

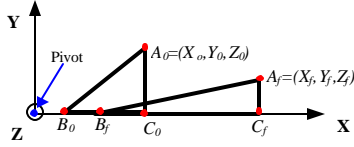


Figure 27. Scaling transformation with the origin as the fixed point.

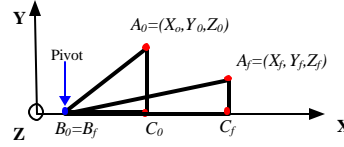


Figure 28. Scaling transformation with fixed point (B) away from the origin.

There may be different scaling factors S_x , S_y and S_z , for the three main axes (anisotropic scaling). For strictly scaling transformations one has $S_i \in [0, \infty]$, with the negative case being a hybrid between mirror and scaling. For the example in Figure 27, $S_x=2.0$, $S_y=0.5$ and $S_z=1.0$.

3.4.2.1 Scaling with respect to the origin.

Figure 27 shows the case in which the object is scaled with respect to the origin as fixed point. This means that the origin is considered part of the object. The scaling is performed with respect to the origin, and therefore points such as B, C, or D move away from (or towards) it.

Equation 37 presents the general matrix form of scaling with respect to the origin.

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} = \text{scale}(S_x, S_y, S_z) \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} \Rightarrow \text{scale}(S_x, S_y, S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 37. Scaling Transformation with origin as fixed point.

According to the values of the S_i parameter, the following effects are produced:

1. If $S_i > 1$, there is an *expansion* in the i^{th} direction.
2. If $S_i < 1$, there is a *contraction* in the i^{th} direction.
3. If $S_i = 1$, there is *no* deformation in the i^{th} direction.
4. If $S_i < 0$, there is a *reflection* and *scaling* in the i^{th} direction.

The reader is invited to find which conditions in Equation 26 are violated and which ones are preserved by a scale transformation (Equation 37).

3.4.2.2 Scaling with respect to an arbitrary point.

Up to now, Equation 37 scales an object assuming that it is pivoted in the origin, $(0,0,0)$. Figure 27 shows that scaling will in this case translate the object. Every point of the object moves. However, if it is wished that a point \mathbf{pv} of the object S_0 be fixed (for example point B in Figure 28), with final object S_f , the procedure is analogous to the applied in mirror or rotations outside the origin:

1. Use M_1 , which takes \mathbf{pv} to the origin, to affect S_0 : $S_1 = M_1 \cdot S_0 = \text{trans}(-\mathbf{pv}) \cdot S_0$.
2. Apply $M_2 = \text{scale}(S_x, S_y, S_z)$ on S_1 to obtain S_2 : $S_2 = M_2 \cdot S_1 = \text{scale}(S_x, S_y, S_z) \cdot S_1$.
3. Use the inverse of M_1 , to take the pivot back to \mathbf{pv} . $S_3 = M_3 \cdot S_2 = \text{trans}(+\mathbf{pv}) \cdot S_2$.

$$\begin{aligned} S_f = \text{scale}(\mathbf{p}_v, S_x, S_y, S_z) \cdot S_0 = M_s \cdot S_0 = \text{trans}(+\mathbf{p}_v) \cdot \text{scale}(S_x, S_y, S_z) \cdot \text{trans}(-\mathbf{p}_v) \cdot S_0 \Rightarrow \\ \text{scale}(\mathbf{p}_v, S_x, S_y, S_z) = \text{trans}(+\mathbf{p}_v) \cdot \text{scale}(S_x, S_y, S_z) \cdot \text{trans}(-\mathbf{p}_v) \end{aligned}$$

Equation 38. Scaling transformation with respect to an arbitrary fixed point in E^3

3.4.3 Shear

In the shear effect, points of the object change their position, in an amount proportional to other coordinates. Therefore, the x coordinate may change as influenced by the y coordinate via a Dx/Dy value, and similarly happens with every coordinate (see Equation 39). In this example, for point (x,y,z) , x changes in proportion to coordinate y , measured from the origin. This transformation is obviously non-rigid since it preserves neither lengths nor angles.

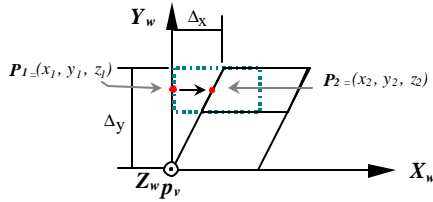


Figure 29. Shear Effect with origin as fixed point.

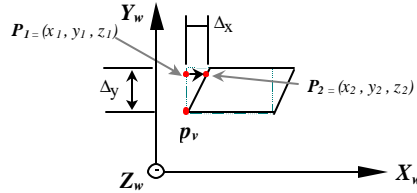


Figure 30. Shear Effect with arbitrary fixed point

For the example displayed in Figure 29 the deformation equations would be:

$$x_2 = x_1 + Dx/Dy \cdot y_1 \quad y_2 = y_1 \quad z_2 = z_1$$

It is easy to realize that the general form for shear with the origin as pivot point is the one shown in Equation 39:

$$\begin{bmatrix} \hat{e}_1 & \frac{Dx}{Dy} & \frac{Dx}{Dz} & 0 \\ \hat{e}_2 & 1 & \frac{Dy}{Dz} & 0 \\ \hat{e}_3 & \frac{Dz}{Dy} & \frac{Dz}{Dz} & 1 \\ \hat{e}_4 & 0 & 0 & 1 \end{bmatrix} = \text{shear} \left(\frac{Dx}{Dy}, \frac{Dx}{Dz}, \frac{Dy}{Dz}, \frac{Dy}{Dx}, \frac{Dz}{Dx}, \frac{Dz}{Dy} \right)$$

Equation 39. Matrix for shear with respect to the origin.

Notice that interactions of any coordinate upon others are possible. This transformation, in general, satisfies none of the conditions for rigidity (Equation 26).

3.4.3.1 Shear with respect to an arbitrary point.

Equation 39 shears an object assuming that it is pivoted in the origin. Figure 29 shows that shearing also translates the object. If point pv of the object S_0 is to be fixed (Figure 30), with final object S_f the procedure is analogous to Equation 38 (see Equation 40).

1. Use M_I , which takes pv to the origin, to affect S_0 : $S_I = M_I \cdot S_0 = trans(-pv) \cdot S_0$.
2. Apply $M_2 = shear(D_{xy}, D_{xz}, D_{yx}, D_{yz}, D_{zx}, D_{zy})$ on S_I to obtain S_2 :
 $S_2 = M_2 \cdot S_I = shear(D_{xy}, D_{xz}, D_{yx}, D_{yz}, D_{zx}, D_{zy}) \cdot S_I$.
3. Use the inverse of M_I , to take the pivot back to pv . $S_3 = M_3 \cdot S_2 = trans(+pv) \cdot S_2$.

$$\begin{aligned}
 S_f &= shear(p_v, \Delta_{xy}, \Delta_{xz}, \Delta_{yx}, \Delta_{yz}, \Delta_{zx}, \Delta_{zy}) \cdot S_0 = M_s \cdot S_0 = \\
 &= trans(+p_v) \cdot shear(\Delta_{xy}, \Delta_{xz}, \Delta_{yx}, \Delta_{yz}, \Delta_{zx}, \Delta_{zy}) \cdot trans(-p_v) \cdot S_0 \Rightarrow \\
 shear(p_v, \Delta_{xy}, \Delta_{xz}, \Delta_{yx}, \Delta_{yz}, \Delta_{zx}, \Delta_{zy}) &= trans(+p_v) \cdot shear(\Delta_{xy}, \Delta_{xz}, \Delta_{yx}, \Delta_{yz}, \Delta_{zx}, \Delta_{zy}) \cdot trans(-p_v)
 \end{aligned}$$

Equation 40. Shear transformation about an arbitrary point pv in E^3

3.5 EXERCISES – GEOMETRIC TRANSFORMATIONS –

3.5.1 Rotations around the Main Axes.

ROTATION MATRIX

OBJECTIVE: To experiment with rigid transformations applied on a 2-dimensional figure.

PROCEDURE:

1. Create matrix that stores the vertices A, B, C and D of the plane that is shown in Figure 31 and name it *face1*.

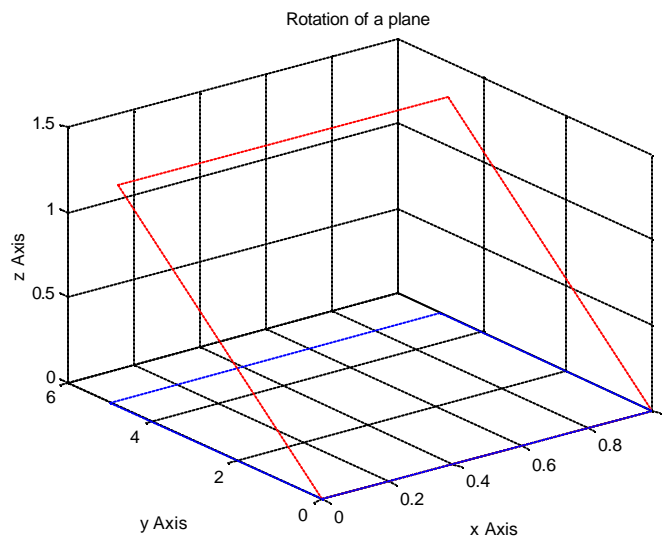


Figure 31. Exercise 3.1

2. Prompt the user for angle of rotation (θ) and the coordinated rotation axis (ax) to be used to rotate the figure.
3. Create a function named $[M] = \text{rotation_matrix}(ax, \theta)$, in which the rotation matrix is calculated. Remember that the variable ax takes values 'X', 'Y' or 'Z'.
4. Make the transformation of *face1* using M and assign the result to a variable named *face2*.
5. Plot both figures in the same MATLAB figure window, the original figure with blue, and with red the transformed one.
6. Name each one of the axes and title the graph.

3.5.2 Iterative Rotations I.

ROTATION MATRIX

OBJECTIVE: To experiment with iterative rigid transformations applied on a 2-dimensional figure.

PROCEDURE:

1. Create matrix that stores the vertices A, B ,C and D of the horizontal rectangle shown in Figure 32. Assign it the name *face_1*.

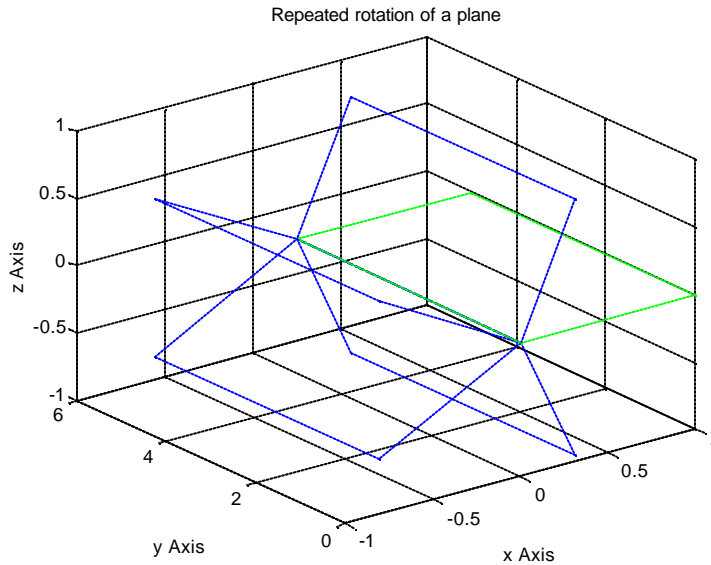


Figure 32. Exercise 3.2

2. Prompt for the number of rotations to execute on *face_1*.
3. Use 'Y' as the rotation axis. Calculate the rotation angle in order to obtain intermediate symmetrical positions.
4. Create a function named $[M] = \text{rotation_matrix} (ax, th)$, in which the rotation matrix is calculated. Remember that the variable *ax* takes values 'X', 'Y' or 'Z'.
5. Make the transformation of *face_i* using *M* and assign the result to a variable named *face_{i+1}*.
6. Make an iteration structure such that the sequence of *face_i* is plotted in the same MATLAB figure window.
7. Name each one of the axes and title the graph.

3.5.3 Iterative Rotations II.

ROTATION MATRIX

OBJECTIVE: To experiment with iterative rigid transformations on a planar polygon. The polygon does not pass through the origin.

PROCEDURE:

The procedure is the same as for the exercise of Figure 32, but in this case *face1* is defined by:

$$\begin{array}{rcl} A & = & [1 \quad 1 \quad 0 \quad 1]' \\ B & = & [3 \quad 1 \quad 0 \quad 1]' \\ C & = & [3 \quad 3 \quad 0 \quad 1]' \\ D & = & [1 \quad 3 \quad 0 \quad 1]' \end{array}$$

Use 'Z' as the rotation axis. The rotation angle for each iteration is 45 degrees.

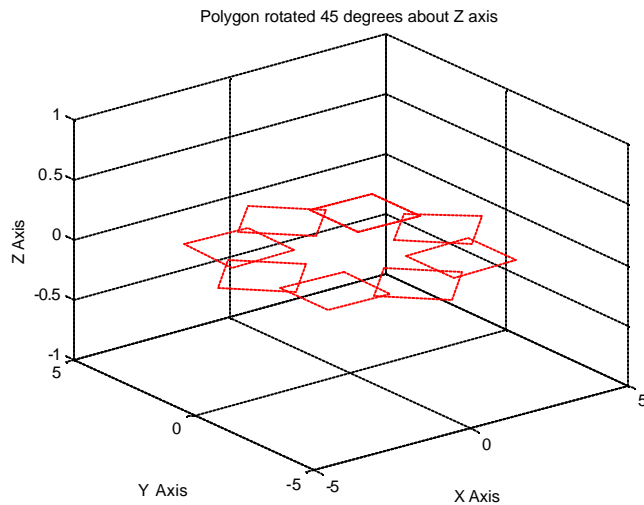


Figure 33. Exercise 3.3

3.5.4 Mirror about the origin.

Create a polygon and mirror it about the origin.

3.5.5 Mirror about XY plane.

Create a polygon and mirror it about the plane XY .

3.5.6 Mirror about $X=Y$ plane.

Create a polygon and mirror it about the plane $X = Y$.

3.5.7 Mirror About Z axis.

Create a polygon and mirror it about the Z axis (Is this a non-rigid transformation?).

3.5.8 Scaling.

Create a polygon and scale it about the origin with scaling factors 1/2 , 2 y 0.3 in the directions X, Y and Z respectively.

3.5.9 Shearing.

Create a polygon and transform it with a shear defined by $D_{xy}=2$. All the remaining interactions are null.

3.5.10 Iterative Rotations III.

RIGID TRANSFORMATIONS

OBJECTIVE:

To make iteratively two independent rotations around the principal axes.

PROCEDURE:

Given the body ABCD, with the edge AD having vertex A at the origin and D along the X axis (Figure 34), you must make a program that: (a) Rotates the body N times about the Z axis (rotation angle $q=\pi/N$), and (b) in each iteration rotates it an angle a about the instantaneous position of the AD edge ($a = q$).

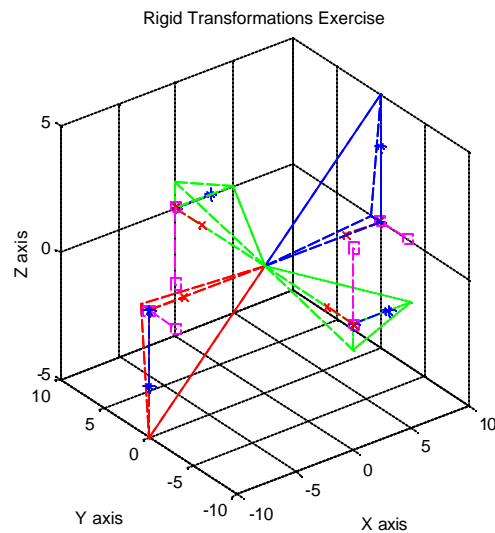


Figure 34. Exercise 3.10

1. Write a function `[body] = create_body()` that will be in charged of generating and delivering the information of the vertices and the faces of the body in an array named **body**. Also, this function must plot the body in its initial position with the color blue.
2. Attach to the solid in every position an auxiliary coordinated system anchored to vertex D (use a specific function for this task, such as `plt_axes()`).
3. Write the function `bodyi+1 = my_trans(bodyi, theta, alpha)`, that rotates the object **body_i** an angle q around the Z axis and subsequently around the AD edge an angle a , to then deliver the result **body_{i+1}**. (Remember that this edge is parallel to the X axis in the initial position of the body).
4. Create a function that plots the evolution of the body. The odd iterations will use green, while the even iterations will use red.

3.5.11 Iterative Rotations IV.

RIGID TRANSFORMATIONS

OBJECTIVE:

To use the Quaternion method to verify the effect of a transformation made by prescribed rotation matrices used to rotate about the principal axes.

PROCEDURE:

1. Write a function $[P_x, P_y, P_z] = \text{my_mesh}()$ that fills the necessary matrices for declaring a mesh and outputs it as P_x, P_y, P_z . The base of the mesh, of size 10x10, will be on the plane XY.

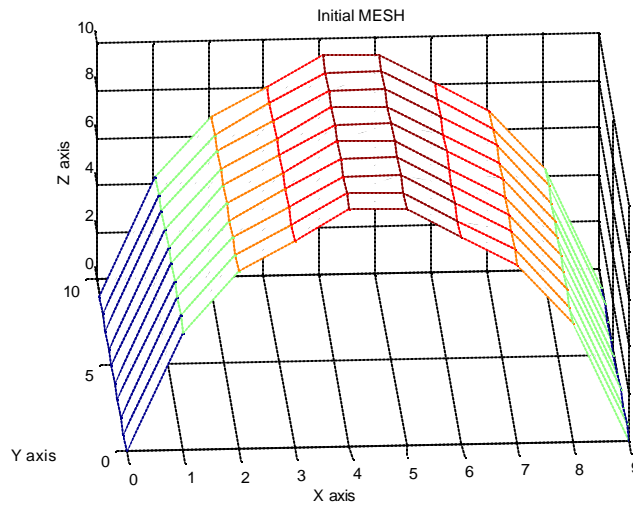


Figure 35. Exercise 3.11

2. Write a function $[P_f] = \text{rotate_by_quaternion}(e, th, P_0)$, that applies the quaternion formula. This function rotates a point P_0 around the vector e and outputs the rotated point P_f . The angle th is in degrees. Assume for this function that e is not necessary unitary.
3. Write a function $[Q_x, Q_y, Q_z] = \text{rotate_mesh}(P_x, P_y, P_z, e, th)$. This function rotates a mesh given by P_x, P_y, P_z around the vector e (not unitary) an angle th (in degrees). Outputs the result (rotated mesh) in the matrices Q_x, Q_y, Q_z .
4. Write a function $[Mr] = \text{rotation_matrix}(th, ax)$, that outputs the homogeneous coordinate matrix for a rotation of th degrees around any axis 'X', 'Y' or 'Z'.
5. Write a function $[Q_x, Q_y, Q_z] = \text{rotate_mesh_2}(P_x, P_y, P_z, ax, th)$. This function rotates with an angle th (degrees) a mesh given by P_x, P_y, P_z around an axis ax (axis 'X', 'Y' or 'Z'). Outputs the result (rotated mesh) in the matrices Q_x, Q_y, Q_z . Use the function $\text{rotation_matrix}()$.

6. Create a program named “*main*”, where the following step are included:
- a) Calls the function $[P_x, P_y, P_z] = \text{my_mesh}()$ that creates the entry data (the mesh).
 - b) In a MATLAB figure window plots the mesh formed by P_x, P_y, P_z .
 - c) Applies the function $[Q_x, Q_y, Q_z] = \text{rotate_mesh}(P_x, P_y, P_z, e, th)$ on P_x, P_y, P_z . The axis e must be one of the principal axes X, Y, or Z, in vector form (3x1). The angle th must be given in degrees. Here the function $[P_f] = \text{rotate_by_quaternion}(e, th, P_0)$ must be used.
 - d) In a new MATLAB figure window shows the original position of the mesh as well as the one obtained by the quaternion method. The title should be “*Transformation by the Quaternion Method*”.
 - e) Calls the function $[Qm_x, Qm_y, Qm_z] = \text{rotate_mesh_2}(P_x, P_y, P_z, ax, ang)$, on P_x, P_y, P_z , where the axis ax (as 'X', 'Y' or 'Z') and the rotation angle are to be consistent with (c). In $\text{rotate_mesh_2}()$ the function $[Mr] = \text{rotation_matrix}(ang, ax)$ must be used.
 - f) In a third MATLAB figure window shows the original mesh as well as the mesh obtained by this method (matrix). The title should be “*Transformation by Matrix Method*”. Compare the plots of (d) and (f).
 - g) Include the principal axes names in every graph.

3.5.12 Solid Generation.

FUNCTIONS WITH SOLIDS

OBJECTIVE:

To create a function that simplifies the display of an arbitrary solid of planar faces.

PROCEDURE:

- Describe the solid of Figure 36 as:

body = [a,d,c,b,a,	a,e,d,a,	a,b,f,e,a,	b,c,f,b,	c,d,e,f,c];
	Loop1	Loop2	Loop3	Loop4	Loop5

 loop_dim = [4,3,4,3,4];
- Create a function **draw_solid(body, loop_dim)**, where **body** is a matrix $4 \times M$ or $3 \times M$ that contains each one of the loops of edges of the body. **loop_dim** is a $1 \times L$ vector that contains the number of vertices that compose each loop that compose the body. L = number of loops of straight edges that conform the body. The vertices in **body** may be input as homogeneous coordinates or cartesian coordinates.
- Make a plot of the faces by means of the above procedure.

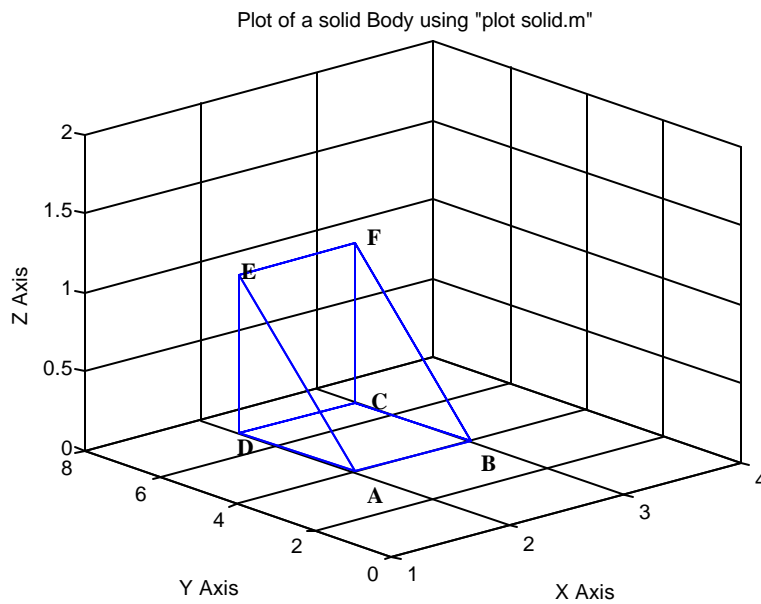


Figure 36. Exercise 3.12

3.5.13 Transformation of a Solid Body I.

OBJECTIVE:

To create and plot objects in E^3 , to make transformations on them and to plot the transformed objects.

PROCEDURE:

1. Create points A, B, C, D, E and F for the vertices of the body (wedge).
2. Create the loops corresponding to the five faces of the body.
3. Execute the necessary transformations to go from the initial to the final position.
4. Plot the body in intermediate positions drawing each position in a different color.

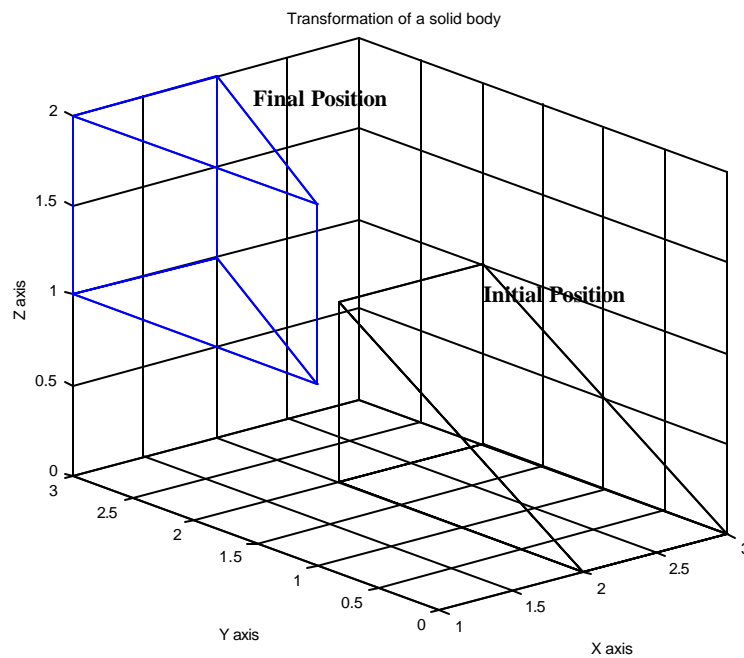


Figure 37. Exercise 3.13

3.5.14 Transformations of a Solid Body II.

OBJECTIVE:

To make transformations on objects in E^3 , To calculate the necessary transformations using the evolution of the auxiliary axes instead of the evolution of the body.

PROCEDURE:

1. Create points A, B, C, D, E and F for the vertices of the body (wedge).
2. Create the loops corresponding to the five faces of the body.
3. Define the initial auxiliary coordinate system. $S_0 = [X_0, Y_0, Z_0, O_0]$.
4. Define the final auxiliary coordinate system $S_f = [X_f, Y_f, Z_f, O_f]$.
5. Transform the auxiliary coordinate system from the initial to the final position using intermediate translations and rotations on this object.
6. Plot the evolution of the auxiliary coordinate system from S_0 to S_f .
7. Collect the transformation matrices in a single matrix M .
8. Apply M to the initial body in order to obtain the final body.

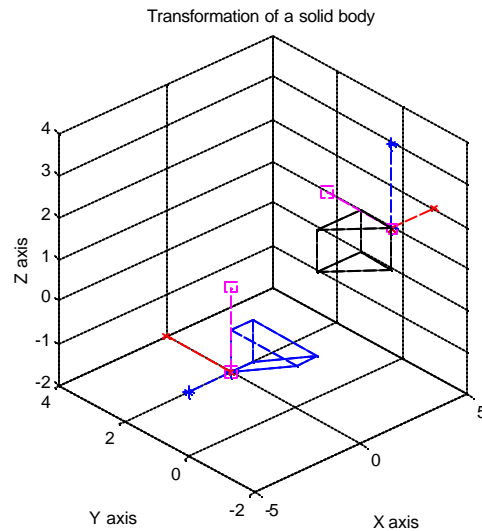


Figure 38. Exercise 3.14

3.5.15 Mirror About XZ plane.

NON-RIGID TRANSFORMATIONS

OBJECTIVE: To study the non-rigid transformations.

PROCEDURE:

1. Define the vertices of the body in Figure 39 in the variable **body**. Create the variable **loop_dim** that would contain the dimensions of each loop of the body. These variables will be used to call the function **draw_solid(body, loop_dim)** defined previously.
2. Create a function **[bodyf , Sf] = mirror_XZ(body , S)** to apply a reflection about the plane **XZ**. The input variables are **body** and **S = [X Y, Z, O]**. Where **S = [X Y, Z, O]** is the auxiliary coordinated system attached to the body. **bodyf** and **Sf** are respectively the transformed body and coordinated frame.

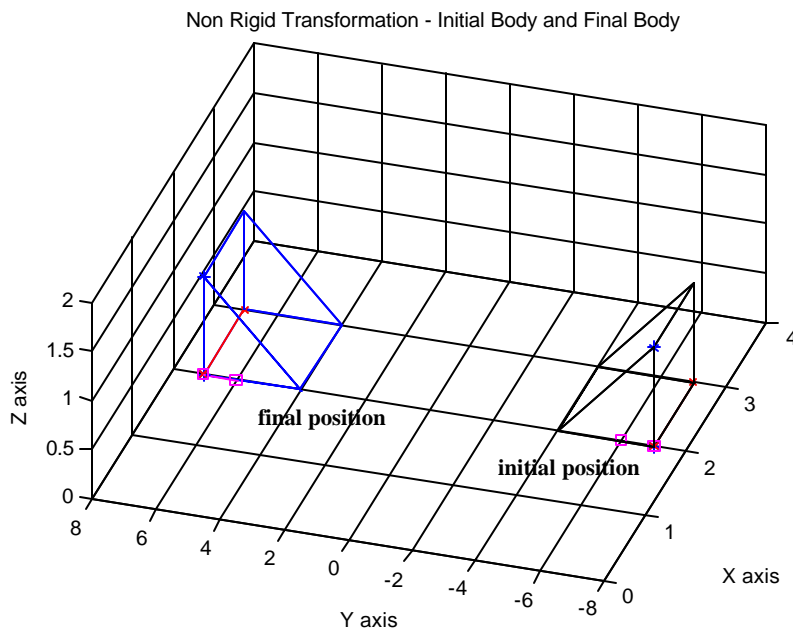


Figure 39. Exercise 3.15

3.5.16 General Transformations I.

GENERAL TRANSFORMATIONS

OBJECTIVE: To combine rigid and non-rigid transformations.

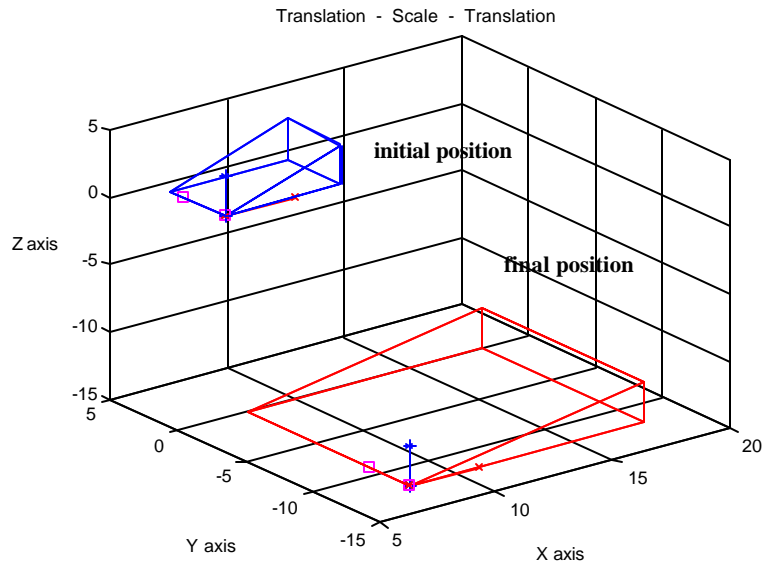


Figure 40. Exercise 3.16

PROCEDURE:

1. Generate the body `[body, loop_dim]` corresponding to the object shown in the initial position of Figure 40.
2. Attach to the body, the auxiliary coordinated system as shown.
3. Make the necessary transformations in order to obtain the body in the final position.

3.5.17 General Transformations II.

GENERAL TRANSFORMATIONS

OBJECTIVE: To combine rigid and non-rigid transformations.

PROCEDURE:

1. Create the points *A*, *B*, *C*, *D*, *E* and *F* for the vertices of the body in the initial position. (Figure 41).
2. Generate the body by means of *[body, loop_dim]* explained in previous exercises.
3. Transform the body and its corresponding auxiliary coordinate system from the initial to the final position, using translation, rotation, mirror, scale and/or shear where necessary.

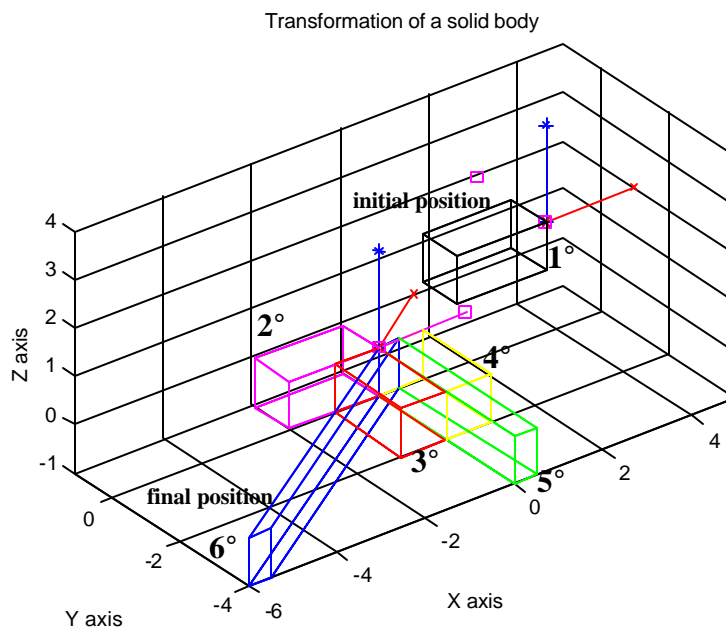


Figure 41. Exercise 3.17

3.5.18 Sweep of a Cross Section along an arbitrary Path.

SWEEP OF A CROSS SECTION ALONG AN ARBITRARY PATH

GIVEN:

1. A Piecewise Linear 1-manifold (a straight -segment polyline) **path** in \mathbb{R}^3 .
2. A closed non self- intersecting polygon **Pol** contained in plane XY, containing the origin (0,0,0), and attached to its local coordinate system $S_L=[X_L, Y_L, Z_L, O_L]$. The system S_L is coincident with the WCS.

GOAL:

1. Calculate the transformations M_i ($i=1,2,\dots,n$) which move the coordinate system S_L attached to **Pol** to : (a) make the origin O_L coincident with the point **path** _{i} , and (b) make the Z_L axis coincident with the axis **Zi** from point **path** _{i} to point **path** _{$i+1$} .
2. Apply the sequence of transformations M_i to **Pol**, to get **Pol** _{i} ($i=1,2,\dots,n$).
3. Use the sequence of extruded cross sections **Pol** _{i} ($i=1,2,\dots,n$), to generate the skin of the extruded object by constructing a (MATLAB) mesh, **snake**. (Figure 42).

NOTES :

(i) For point (i) you do not have sufficient information. Answer what is the cause of the insufficiency. Avoid this insufficiency by using quaternion to transform Z_L into Z_i . Use the quaternion to define an M_i homogenous transformation from S_L to S_i

(ii) Your **path** should be a smoothly evolving curve. Otherwise you may get a self-intersecting skin.

RESULTS

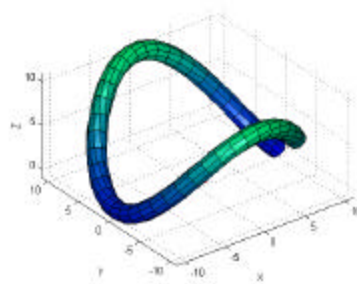


Figure 42. Generated extrusion of circular cross section.

3.5.19 Simultaneous Sweep and Twist of a Cross Section.

SIMULTANEOUS SWEEP AND TWIST OF A CROSS SECTION

GIVEN

1. A right-handed WCS.
2. An ellipsoidal cross section *pol*, whose contour follows the equation

$$x(?) = a \cdot \cos (?)$$

$$y(?) = b \cdot \sin (?)$$

$$z = 0$$

with $0 \leq ? = 360^\circ$, and a and b are the x - and y -direction semi -axes respectively. The polyline *pol* is defined in a local frame $S_L = [X_L, Y_L, Z_L, O_L]$, initially coincident with the WCS.

3. A circular path *path* (defined in the WCS) with radius R , lying on the X_w - Y_w plane, defined by the dF (angular increment of the arc)

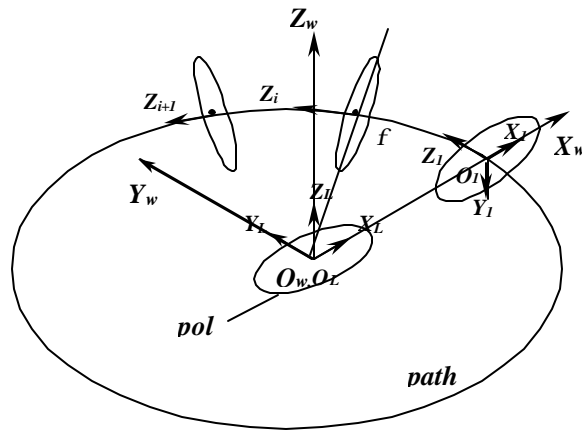


Figure 43. Sweeping and Tweaking of a Cross Section

GOAL:

1. To generate a sequence *pol*(1), *pol*(2), ..., *pol*(i),... of rigid transformations of *pol* with the following conditions for *pol*(i): (a) The geometric center O_L of *pol*(i) must be placed onto the point *path*(i)= O_i , (b) The Z_L axis of *pol* must be placed in the direction *path*(i+1)- *path*(i), which defines the axis Z_i (c) in each iteration i the instantaneous S_i system rotates a dF around the instantaneous Z_i axis. (Figure 43).
2. To thread corresponding points between stages *pol*(i) and *pol*(i+1) to generate a mesh called *ribbon*.

REMARKS:

4. The program must define a proper $d\theta$ to accomplish the desired number of spins of the cross section around its local Z_i axis.
5. The program must define a proper dF , taking into account that the ribbon must be defined by n radial partitions.
6. The program must draw each intermediate position $pol(i)$ ($i=1,2,3...n$) that the cross section pol adopts through the generation of the ribbon.
7. The program must accumulate and thread all $pol(i)$ cross sections into an ordered mesh format.

PROCEDURE:

1. Generate the ellipsoid pol in the local coordinate frame S_L .
2. Generate the circular $path$ with an R radius.
3. Reposition the cross section pol in position S_L .
4. For each S_i position calculate a transformation matrix Mt_i that transforms S_L onto S_i . This matrix has two components; (i) rotation about some axis in the WCS. (ii) rotation about an instantaneous Z_i axis. (Figure 44).

RESULTS:

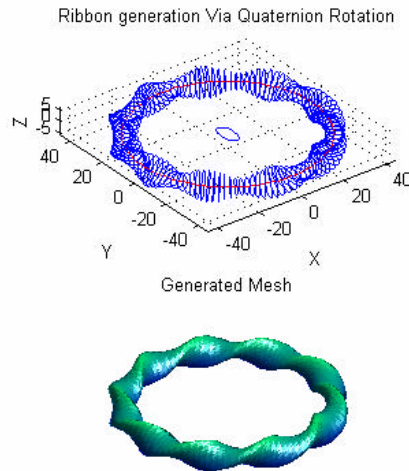


Figure 44. Ribbon of Radius =40, $a=3$, $b=6$, $\theta=360^\circ$ and $n=120$

3.5.20 Parallel Projection.

PARALLEL PROJECTION

GIVEN:

1. A right-handed WCS pivoted on O_o .
2. A Solid Body B_o with an attached coordinate system S_o .
3. A plane $\pi = [P_v, n]$ in R^3 , pivoted into any P_v point and defined by a normal vector n . In general, P_v is not the origin.

GOAL:

To write down a program that calculates and applies (to a body B_o) the transformation matrix M for parallel projection onto a given plane $\pi = [P_v, n]$.

PROCEDURE:

1. Generate the body B_o and its attached coordinate system S_o .
2. Write down a generic function $M = \text{project}(pv, n)$ that calculates the homogenous non rigid transformation M for parallel projection onto a plane $\pi = [pv, n]$.
3. Apply the transformation M to the coordinate frame S_o .
4. Apply the transformation M to the body B_o .
5. Plot the body B_o and its projection B_{op} . (Figure 45)

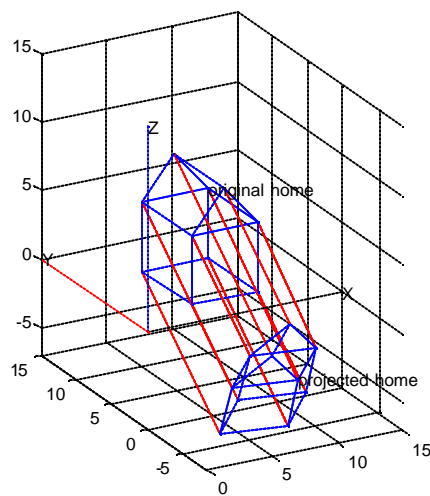


Figure 45. Parallel Projection

3.5.21 Perspective Projection.

PERSPECTIVE PROJECTION

GIVEN

1. A right-handed WCS pivoted on O_o .
2. A Solid Body B_o with an attached coordinate system S_o .
3. A plane $\pi = [P_v, n]$ in R^3 , pivoted on a point P_v with normal (non unitary) vector n . In general, P_v is not the origin.
4. A point of observation P_{obs} .

TASK

To write down a program that calculates and draws the perspective projection (calculated from P_{obs}) of a body B_o projected over a given plane $\pi = [P_v, n]$.

PROCEDURE

1. Generate the body B_o and its attached coordinate system S_o .
2. Plot the projection lines from P_{obs} to each vertex in B_o .
3. Write down a generic function $B_p = \text{persp_project}(B_o, P_v, n, P_{obs})$ which calculates B_p , the perspective projection of B_o onto plane $\pi = [P_v, n]$, with observer position P_{obs} , using the following formula for each vertex P_o that will be projected:

$$P_p = P_{obs} + (P_o - P_{obs}) \cdot \frac{1 - (P_o - P_v) \cdot n}{(P_o - P_{obs}) \cdot n}$$

Equation 41. Calculation of a Perspective projection of a single point into a plane.

4. Plot the body B_o and its projection B_p . (Figure 46)

Remarks

The reader must visually check if each projection of each point p_i of B_o onto the π plane is the intersection between the ray $P_{obs} - p_i$ against π .

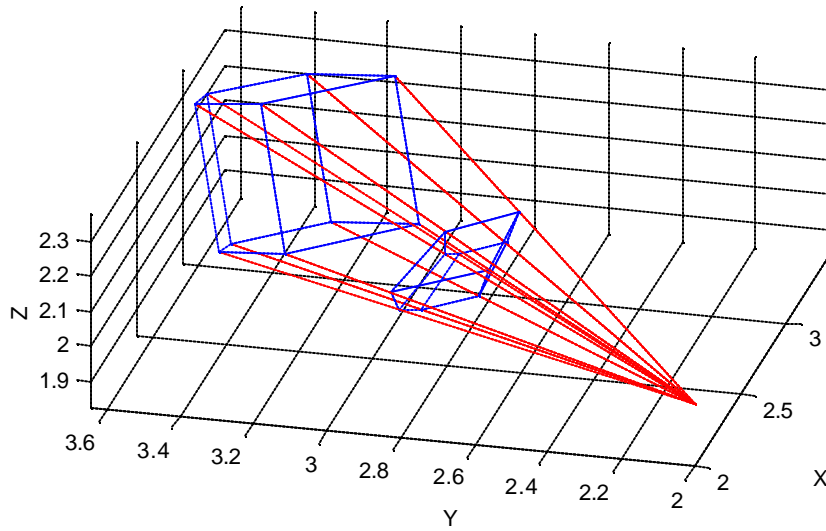


Figure 46. Perspective Projection

4. CURVES AND SURFACES

In Computer Aided Geometric Design (CAGD) analytic curves and surfaces are required as carriers for the topological entities of a geometric model. For example, FACES are connected, compact subsets of an analytic surface (a 2-manifold with boundary) in E^3 , while EDGES are connected, compact subsets of an analytic curve (a 1-manifold with boundary) in E^3 . FACES and EDGES are only subsets of the surface or curve (respectively) in which they are mounted. The mathematical definition of carrier geometries (curves or surfaces) immediately affects how subsets of them can be expressed and bounded. The definition of curves and surfaces is generally expressed in terms of parameters, giving origin to what is called *parametric* equations.

The mathematical problems faced in CAGD applications are the following:

- (i) Given a finite set of points sampled on a curve or surface, how to find a mathematical expression of such a set. The set of points might be randomly (without order) sampled, requiring a different treatment, in contrast with cases in which points are sampled following a systematic and ordered pattern. When the order of the point set is not meaningful, there are two techniques widely used: (a) polynomial and (b) statistical equation fitting. When the order of the point sample must be respected, parametric forms are applied.
- (ii) Once the mathematical expression has been found, estimations of its likeness may be required.
- (iii) Evaluation of the formula in locations away from the initial point set is required.
- (iv) In addition to simply compute the locus of the points (item (iii)), other estimations are of interest: tangent planes, derivatives, gradients, Frenet frames, curvatures, etc. These estimations are important in Computer Aided Manufacturing applications (CAM), for example CNC machining.
- (v) Given a mathematical expression of a curve or surface, it is of interest to convert it to other representation or equation, with a different set of parameters or variables.

The purpose of this chapter is to introduce the reader into this domain, and to provide concepts to respond some of the issues stated.

4.1 Random Samples

4.1.1 Polynomial Interpolation

In this case, an equation of the form $z = f(x,y)$ or $g(x,y,z) = c$ is forced to pass exactly by each one of the point samples $p_i = (x_i, y_i, z_i)$ ($i = 1..n$). The natural candidate to propose is a polynomial equation of the form

$$Z = a_0 X^0 + a_1 X^1 + a_2 X^2 + \dots + a_n X^n + b_0 Y^0 + b_1 Y^1 + \dots + b_n Y^n + b_{0,0} X^0 Y^0 + b_{1,1} X^1 Y^1 + \dots + b_{n,n} X^n Y^n + \dots$$

in which the degree is controlled by the number of points collected. For example, if a sample of five (x,y) pairs is available, the equation to fit would have the form:

$$\begin{aligned} Y_1 &= a_0 + a_1 X_1 + a_2 X_1^2 + a_3 X_1^3 + a_4 X_1^4 \\ Y_2 &= a_0 + a_1 X_2 + a_2 X_2^2 + a_3 X_2^3 + a_4 X_2^4 \\ &\dots\dots\dots \\ Y_5 &= a_0 + a_1 X_5 + a_2 X_5^2 + a_3 X_5^3 + a_4 X_5^4 \end{aligned}$$

With unknowns a_0, a_1, a_2, a_3, a_4 . The matrix equation to solve for the unknowns takes the form:

$$\begin{bmatrix} 1 & X_1 & X_1^2 & X_1^3 & X_1^4 \\ 1 & X_2 & X_2^2 & X_2^3 & X_2^4 \\ 1 & X_3 & X_3^2 & X_3^3 & X_3^4 \\ 1 & X_4 & X_4^2 & X_4^3 & X_4^4 \\ 1 & X_5 & X_5^2 & X_5^3 & X_5^4 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \end{bmatrix}$$

which can be abbreviated as:

$$X.A = Y$$

The following observations are relevant:

- (i) X is called a Vandermonde Matrix ([GOLUB.96]), which is known to be ill-conditioned for inversion, given the radical variation in the absolute value of its entries.
- (ii) The form $g(x,y,z)=c$ so defined effectively passes through each sampled point. However, its oscillation between samples is uncontrolled, particularly as the value of n grows.
- (iii) Due to the previous considerations, only low degree (2 or 3) solutions are attempted in very specific CAGD problems. Therefore the applicability of this method in CAGD is limited.

4.1.2 Statistic Interpolation

In this case, the designer *estimates* that the data set adheres to a particular equation form. Therefore the data set is used to give the best estimation (in statistical sense) of the equation parameters. Again, the equation form is $g(x,y,z) = c$. However, in contrast with polynomial interpolation, the number of samples is not limited by the number of parameters to estimate (the limitation in taking large samples lies on economical reasons). In the equations below, the proposed form is a quadratic equation $y = a_2.x^2 + a_1.x^1 + a_0$. The number of samples is not limited to 3, so in this case 20 samples are collected. The equations are stated as follows:

$$\begin{aligned} Y_0 &= a_0 + a_1 X_1 + a_2 X_1^2 + e_0 \\ Y_1 &= a_0 + a_1 X_2 + a_2 X_2^2 + e_1 \\ &\vdots \\ Y_{20} &= a_0 + a_1 X_{20} + a_2 X_{20}^2 + e_{20} \end{aligned}$$

Where e corresponds to both the limitations of the equation to estimate the data, and to sampling (experimental) errors.

The following observations are relevant:

- (i) By changing variables (for example naming X^2 as x_2 , etc.), a linear regression analysis can be applied to solve for a_0, a_1, a_2 . A thorough explanation is beyond the purpose of this chapter, since the statistical – mathematical procedures vary with the form of each estimated shape. However, the usual goal is to express the problem in terms of linear regressions, and to convert the results back to the original parameters.
- (ii) In CAGD and CAM the usual forms are cylinders, planes, circles, cones, etc., since these are the most common primitives and finishing features used in mechanical design (bevels, chamfers, fillets, etc.).

Therefore the equations to fit will acquire the corresponding form. For instance in a circle $(x-x_0)^2 + (y-y_0)^2 = R^2$ the unknowns are x_0 , y_0 and R ; a minimum of three (non-collinear) samples would be then required.

- (iii) An important field of application of this approach is in Geometric Tolerances, where verifications on proposed features determine whether the part is within tolerances and quality limits or not. In this case, position of axes, perpendicularity, coaxiality, flatness, etc., are estimated as expressed above.

4.2 Ordered Samples. Parametric Equations

In contrast with polynomial and statistical fits, a parametric curve $f(u)$ (or surface $f(u,v)$) when fit to a sequence of points $S = \{p_0, p_1, \dots, p_n\}$ (S is called *control polygon*) changes with the order (sense) of the points in S . A parametric curve (Figure 47) is a function $f(u): E \rightarrow E^3$, with $f(u) = (x(u), y(u), z(u))$, where the parameter u is one of many possible ways to parameterize the curve. For example, another parameterization could be $g(v) = f(u/2) = (x(u/2), y(u/2), z(u/2))$, with $v = u/2$. On the curve, parameter v would vary twice as fast as u . However, given a consistent definition of the respective domains of u and v , $f(u)$ and $g(v)$ describe exactly the same set of points in E^3 . A special parameterization $h(s)$ of a curve is the one in which the parameter s measures the length of the curve. This case is called *unit-speed parameterization*, and presents special conditions in engineering and physics. However, not all the curves allow an analytic closed form with unit speed parameterization ([ONE.66]).

Likewise, a parametric surface is a function $f(u,v): E^2 \rightarrow E^3$, with $f(u,v) = (x(u,v), y(u,v), z(u,v))$. The same considerations expressed for parametric curves are valid, including the fact that unit-speed parameterizations for surfaces imply that a square unit in parameter space determines a square unit on the surface. Again, such parameterizations are rarely available in analytic closed form. It is common practice to define the parameters in such a way that the relevant part of the curve or surface corresponds to $u \in [0,1]$, $v \in [0,1]$.

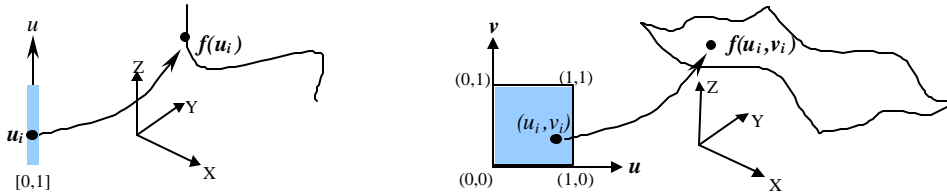


Figure 47. Parametric mappings for curves and surfaces

Natural questions arise about parametric curves and surfaces, such as: (i) which are their mathematical characteristics, (ii) how to deform, translate or rotate them, (iii) how to relate their mathematical properties to consequences in the domains of manufacturing and design.

In order to present a coherent discussion, several concepts must be introduced, as follows:

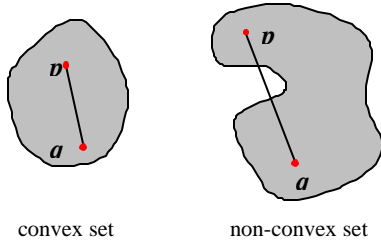


Figure 48. Set convexity

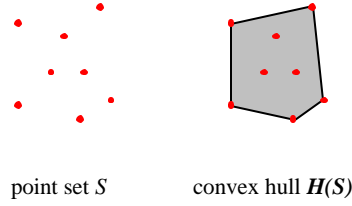


Figure 49. Convex Hull of a 2D set S

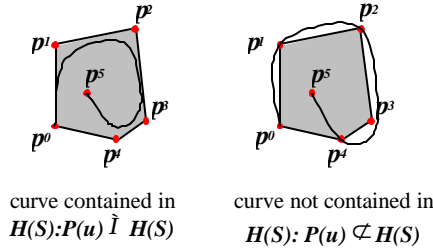


Figure 50. Containment of a parametric form

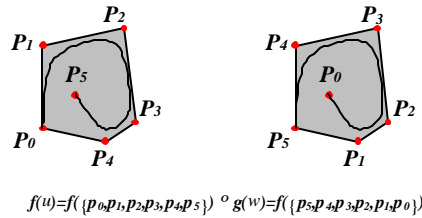


Figure 51. Invariance under inversion of sample order.

Definition 1. Convex Set.

A set C contained in a universe U is convex if

$$\forall p, q \in C, \overline{pq} \subseteq C$$

Where \overline{pq} represents the segment with end elements p and q , defined by $\overline{pq} = \{1.p + (1-1).q \mid 1 \in [0,1]\}$, with the addition and multiplication operators defined in the universe U . If $U = E^3$, with real addition and multiplication, a segment has the usual meaning, of a connected subset of a straight line (Figure 48).

Definition 2. Convex Hull.

Let $S = \{p_0, p_1, \dots, p_n\}$ be a set of points $p_i \in E^3$. The convex hull of S , $H(S) \in E^3$ is the minimal convex set such that $S \subseteq H(S)$ (Figure 49).

Property 1. Containment for a parametric form.

In industrial applications it is required that parametric curves $f(u)$ (or surfaces $f(u,v)$) fit to a control polygon (or polyhedron) $S = \{p_0, p_1, \dots, p_n\}$ and to hold that $f(u) \subseteq H(S)$ ($f(u,v) \subseteq H(S)$). This property establishes that the approximation to the control polygon cannot have erratical behavior (Figure 50).

Property 2. Invariance of a parametric form under inversion of the originating set.

Let a parametric curve $f(u)$ (or surface $f(u,v)$) be fit to a control polygon $S = \{p_0, p_1, \dots, p_n\}$, and a parametric curve $g(w)$ (or surface $g(w,s)$) be fit to a control polygon $S = \{p_n, p_{n-1}, \dots, p_0\}$. The sets $g(w) \in E^3$ and $f(u) \in E^3$ satisfy $f(u) \equiv g(w)$. This property (Figure 51) establishes that the parametric forms (used in industrial

applications) must be invariant under inversion of the sequence of their originating point set. This property is essential in applications in which the actual shape produced is independent of the *direction* of sampling of the point set.

4.2.1 Parametric Curves

The following definitions sustain previous considerations and properties:

Definition 3. Parametric Curve as a Weighted Sum.

Given a sequence of n points $S = \{ p_0, p_1, \dots, p_{n-1} \}$ a parametric shape proposed by Bezier (BEZIER), De Casteljeau (DeCasteljeau) and others is:

$$\mathbf{p}(u) = \sum_{i=0}^{n-1} B_i(u) \cdot \mathbf{P}_i \quad \text{with} \quad 0 \leq u \leq 1, \quad \sum_{i=0}^{n-1} B_i = 1$$

It can be easily seen that this formulation satisfies Property 1 (containment for a parametric form). The $B_i(u)$ are called weights, blending functions or coefficients. In Bezier the amount of blending functions is dependent on the number of control points).

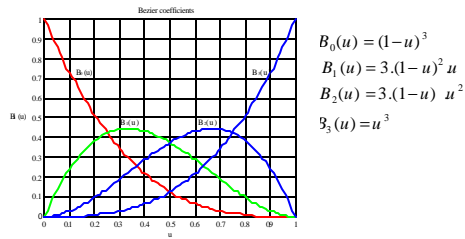


Figure 52. Set of Bezier weight coefficients (num. points = 4).

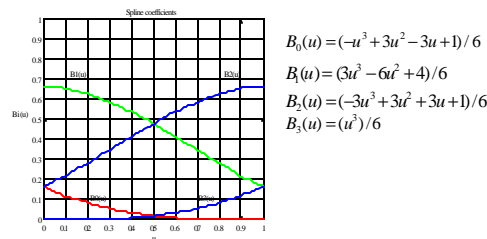


Figure 53. Set of Uniform B-spline weight coefficients (num. points = 4).

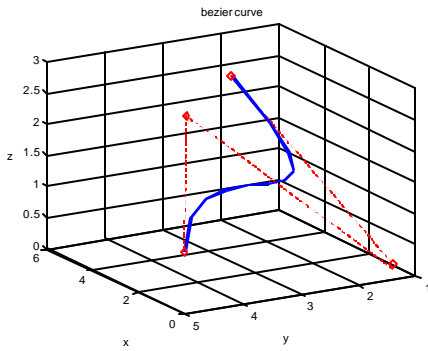


Figure 54. Example of Bezier Curve

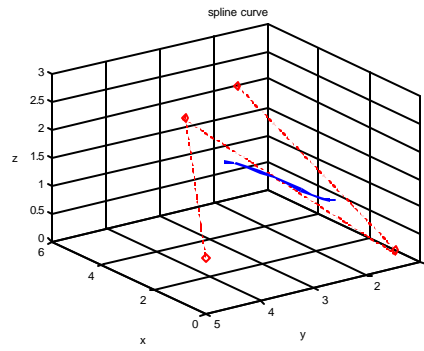


Figure 55. Example of Spline Curve

Property 3. Symmetry of the Weight Set.

In order to satisfy Property 2 (invariance under inversion of point sequence), the $B_i(u)$ coefficients of the equation

$$p(u) = \sum_{i=0}^{n-1} B_i(u) \cdot P_i$$

must be symmetrical about $u = 0.5$ (Figure 52 and Figure 53). Therefore

$$B_0(u) = B_{n-1}(1-u), \quad B_1(u) = B_{n-2}(1-u), \quad \dots \quad B_j(u) = B_{n-1-j}(1-u)$$

Property 4. Adherence of curve to control polygon at endpoints .

A special case of Property 1 (Containment for a parametric form), is the one in which the $B_i(u)$ coefficients of the equation

$$p(u) = \sum_{i=0}^{n-1} B_i(u) \cdot P_i$$

hold that (condition 1):

$$\begin{array}{llll} B_0(0) = 1, & B_1(0) = 0, & \dots & B_{n-1}(0) = 0 \\ B_0(1) = 0, & B_1(1) = 0, & \dots & B_{n-1}(1) = 1 \end{array}$$

In this case the calculated point equals to the control point,

$$p(u=0) = P_0, \quad p(u=1) = P_{n-1}$$

Which makes the curve to adhere to the initial and final points of the control polygon (Figure 54). If, in addition one has (condition 2) the derivative evaluated in a specific value of the parameter:

$$p'(u=0) = k_0(P_1 - P_0) \quad \wedge \quad p'(u=1) = k_f(P_{n-1} - P_{n-2}) \quad \text{for some scalars } k_0, k_f,$$

the tangent of the curve adheres to the initial and final segments of the control polygon. There exist curve formulations that do not obey these two conditions, for example Spline curves (Figure 55), while Bezier ones do adjust to them.

Definition 4. A Polynomial formulation for the blending functions.

A common case of Property 1 (containment for a parametric form), is the one in which the $B_i(u)$ blending functions of the equation

$$p(u) = \sum_{i=0}^{n-1} B_i(u) \cdot P_i$$

are polynomials.

Figure 52 and Figure 53 show sets of $B_i(u)$ polynomial blending functions. Notice that (a) $u \in [0,1]$, (b) the evaluated weight or coefficient $B_i(u) \in [0,1]$, (c) the Bezier set satisfies Property 4 (adherence of curve to control polygon at endpoints) while the Spline set does not. Instead, Spline curves are strongly attracted by the intermediate control points (in this case P_1 and P_2) as the large relative size of the B_1 and B_2 coefficients suggest. (d) The Property 3 (symmetry of the weight set around $u=0.5$) is evident, since $B_0(u) = B_3(1-u)$ and $B_1(u) = B_2(1-u)$. The degree of each polynomial blending function in both the Bezier and Spline cases depends on the number of points to interpolate (*degree = n = num_points - 1 in C programming language. n = num_points in MATLAB programming language*).

4.2.1.1 Bezier Curve

4.2.1.1.1 Scalar form of the Bezier coefficients.

The Bezier curve approaching an n-point control polygon is given by:

$$p(u) = \sum_{i=0}^{n-1} B_i(u) \cdot P_i \quad , \quad u \in [0,1]$$

with $B_i(u)$ (also known as *Bernstein* polynomial) being the i^{th} ($i=0..n$) weight coefficient for the i -th control point:

$$B_{i,n}(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

with

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} = C(i, n)$$

Equation 42. Bernstein form of the Bezier coefficients

Figure 57 shows a set of Bezier coefficients for the case $k = 7$ control points. Observe the symmetry property, as well as the fact that the $B_i(u)$ coefficients are independent of the *location* of the control points. Thus, any other Bezier curve with $k=7$ points will have the same $B_i(u)$ coefficients. It is also evident (Figure 56) that the Bezier formulation forces the curve to adhere, in position and tangency, to the extremes of the control polyhedron.

4.2.1.1.2 Matrix form of the Bezier coefficients.

From the example discussed above ($n = 3$, control points = 4) and Equation 42, the Bezier formulation for the curve is:

$$\mathbf{p}(u) = \begin{bmatrix} (1-3u+3u^2-u^3) \\ (3u-6u^2+3u^3) \\ (3u^2-3u^3) \\ (u^3) \end{bmatrix}^T \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Which can be expressed in matrix form as:

$$\mathbf{p}(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

$$\boxed{\mathbf{p}(u) = \mathbf{U}_k \cdot \mathbf{M}_k \cdot \mathbf{Q}_k}$$

Equation 43. Matrix formulation of parametric curves.

Where the terms \mathbf{U}_k , \mathbf{M}_k and \mathbf{Q}_k in Equation 43 are characteristic for k control points. They appear in

Table 11.

Table 11. Matrix formulation for Bezier Curves.

$k = n+1$	U_k	M_k	Q_k
2	$U_2 = [u \ 1]$	$M_2 = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$	$Q_2 = \begin{bmatrix} \hat{e} P_0 \hat{u} \\ \hat{e} P_1 \hat{u} \end{bmatrix}$
3	$U_3 = [u^2 \ u \ 1]$	$M_3 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	$Q_3 = \begin{bmatrix} \hat{e} P_0 \hat{u} \\ \hat{e} P_1 \hat{u} \\ \hat{e} P_2 \hat{u} \end{bmatrix}$
4	$U_4 = [u^3 \ u^2 \ u \ 1]$	$M_4 = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$	$Q_4 = \begin{bmatrix} \hat{e} P_0 \hat{u} \\ \hat{e} P_1 \hat{u} \\ \hat{e} P_2 \hat{u} \\ \hat{e} P_3 \hat{u} \end{bmatrix}$
5	$U_5 = [u^4 \ u^3 \ u^2 \ u \ 1]$	$M_5 = \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ -4 & 12 & -12 & 4 & 0 \\ 6 & -12 & 6 & 0 & 0 \\ -4 & 4 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$	$Q_5 = \begin{bmatrix} \hat{e} P_0 \hat{u} \\ \hat{e} P_1 \hat{u} \\ \hat{e} P_2 \hat{u} \\ \hat{e} P_3 \hat{u} \\ \hat{e} P_4 \hat{u} \end{bmatrix}$

Figure 54 and Figure 55 show the results of using the two different types of interpolation coefficients (Bezier and Spline) in fitting a curve to the same 4-point control polygon.

Observe the effect of Property 4 (adherence of curve to control polygon) of the Bezier curve, in contrast with the Spline one. Figure 56 shows that, given the adherence property for Bezier curves, achieving a C^1 (tangent - continuous) closed Bezier interpolation is an over-specified problem and therefore non-solvable with arbitrary p_1 and p_{n-2} points. It must be handled by controlling p_1 and p_{n-2} to achieve the desired tangent. If in addition C^2 (curvature-continuous) interpolations are required, points p_2 and p_{n-3} must be manipulated.

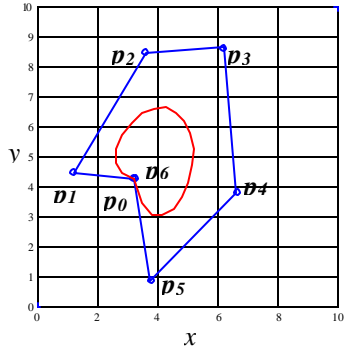


Figure 56. Closed Bezier Curve
(case. $k=7$ control points ($p_0=p_6$)).

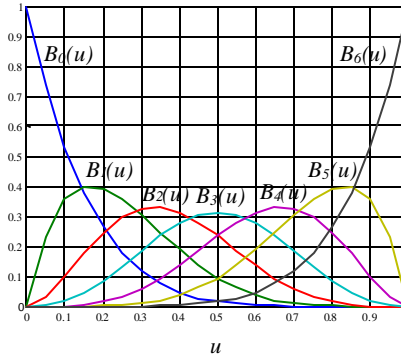


Figure 57. Bezier coefficients for $k=7$ control points.

Remarks:

- i. Bezier formulation uses $Bi(u)$ polynomials whose degree grows with the number of control points (degree = n). Therefore, with large sets of control points, the orders of such polynomials become extremely large.
- ii. Modification of a control point in Bezier formulation implies the recalculation of the whole curve, since each point $P(u)$ of the curve is affected by all control points.
- iii. As Figure 55 suggests, Spline curves were designed to be used in subsets (stages) of a larger control polygon. Their weakness in tracking the endpoints of the control polygon becomes strength in accommodating tangency and curvature conditions of the previous and next stage. This quality also facilitates the design of closed smooth shapes. Because of their design in stages, Spline ease the disadvantages (i) and (ii) of Bezier curves.

4.2.1.2 Uniform B-Spline Curve

As mentioned before, the properties of parametric curves are concentrated in their coefficient set $Bi(u)$. The matrix form for Bezier curves has a counterpart for Spline ones. Table 12 shows the results of the derivations for U_k , M_k , and Q_k , found in the literature. The equation for Spline Curves is identical to the Bezier case, namely Equation 43, [except](#) for the content of the M_k matrix.

The interested read may refer to MOR.85 and FAR.90 and many other authors in search for the derivations leading to Table 12.

Table 12. Matrix formulation for Uniform B-Spline Curves.

k	U_k	M_k	Q_k
2	$U_2 = [u \ 1]$	$M_2 = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$	$Q_2 = \begin{bmatrix} P_{i-1} \\ P_i \end{bmatrix}$
3	$U_3 = [u^2 \ u \ 1]$	$M_3 = \frac{1}{2} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix}$	$Q_3 = \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \end{bmatrix}$
4	$U_4 = [u^3 \ u^2 \ u \ 1]$	$M_4 = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$	$Q_4 = \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$

In order to use Spline curves, the overall control polygon $S = \{p_0, p_n\}$ must be processed in stages. Every stage has the same number k of points (usually is $k = 2, 3, 4, 5$).

Table 13 presents the composition of the control polygon for each stage. The reader is invited to calculate the number of stages (f) as function of the n (number of points -1) and k (points per stage) values.

Table 13. Disposition of control points in stages for Open Spline Curves.

	p ₀	p ₁	p ₂	p ₃	p ₄						p _{n-3}	p _{n-2}	p _{n-1}
Open Spline, k = 4															
Stage 1	p ₀	p ₁	p ₂	p ₃											
Stage 2		p ₁	p ₂	p ₃	p ₄										
Stage 3			p ₂	p ₃	p ₄	p ₅									
...												
...										p _{n-6}	p _{n-5}	p _{n-4}	p _{n-3}		
...										p _{n-5}	p _{n-4}	p _{n-3}	p _{n-2}		
Stage f											p _{n-4}	p _{n-3}	p _{n-2}	p _{n-1}	
Open Spline, k = 3															
Stage 1	p ₀	p ₁	p ₂												
Stage 2		p ₁	p ₂	p ₃											
Stage 3			p ₂	p ₃	p ₄										
...						...									
...							p _{n-5}	p _{n-4}	p _{n-3}		
...											p _{n-4}	p _{n-3}	p _{n-2}		
Stage f												p _{n-3}	p _{n-2}	p _{n-1}	

The stage $i (i=1, 2, 3...)$ in a Spline curve has the following form:

$$P_i(u) = U_k \cdot M_k \cdot \begin{bmatrix} p_{i-1} \\ p_i \\ \dots \\ p_{i+k-2} \end{bmatrix}$$

with dimensions of $U_k (1 \times k)$, of $M_k (k \times k)$ and the local control polygon $(k \times 3)$.

4.2.1.2.1 Continuity analysis for Spline curves

For the Spline curves, the overlapping control points in consecutive stages allows the fulfilling of their continuity requirements (see

Table 13). This overlap is determined by the degree of the weight coefficients $B_i(u)$ which is dictated by the number of points in the stage. For example, for stages with $k = 4$ points, the degree of the $B_i(u)$ coefficients is 3 and the overlap between stages is also 3 points. In this case, one obtains C2 (curvature), and obviously C0 (simple) and C1 (tangent) continuities between consecutive stages.

Points per stage k	Degree of the Stage n	Continuity			Comment
		C^0	C^1	C^2	
1	0 (isolated points)				$stage_i(u) = P_i \cdot u \cdot \hat{I} [0,1]$ Spline is the control points.
2	1 (linear)	✓			Spline is the control polygon. Tangency is not kept.
3	2 (parabolic)	✓	✓		Tangency is kept.
4	3 (cubic)	✓	✓	✓	Curvature is kept.

4.2.1.2.2 Closed Spline curves

In order to fit closed Spline curves to a control polygon $S = \{p_0 \dots p_n\}$ additional stages are appended by recycling the initial points p_0, p_1, p_2 , etc. and stopping before the stage $[p_0 \dots p_{k-1}]$ is repeated. Table 14 shows the composition of the stages for closed Spline curves. Again, the reader is invited to calculate the number of stages (f).

The stage $i (i=1, 2, 3 \dots n)$ in a Closed Spline curve has the following form:

$$p_1(u) = U_k \cdot M_k \cdot \begin{bmatrix} p_0 \\ p_1 \\ \dots \\ p_{k-1} \end{bmatrix} \dots p_i(u) = U_k \cdot M_k \cdot \begin{bmatrix} p_{i-1} \\ p_i \\ \dots \\ p_{i+k-2} \end{bmatrix} \dots p_n(u) = U_k \cdot M_k \cdot \begin{bmatrix} p_{n-1} \\ p_0 \\ \dots \\ p_{k-2} \end{bmatrix}$$

Table 14. Disposition of control points in stages for Closed Spline Curves

	p_0	p_1	p_2	p_3	p_4			p_{n-3}	p_{n-2}	p_{n-1}	p_0	p_1	p_2
Closed Spline, $k = 4$															
Stage 1	p_0	p_1	p_2	p_3											
Stage 2		p_1	p_2	p_3	p_4										
Stage 3			p_2	p_3	p_4	p_5									
...													
									p_{n-4}	p_{n-3}	p_{n-2}	p_{n-1}			
										p_{n-3}	p_{n-2}	p_{n-1}	p_0		
											p_{n-2}	p_{n-1}	p_0	p_1	
Stage f												p_{n-1}	p_0	p_1	p_2
Closed Spline, $k = 3$															
Stage 1	p_0	p_1	p_2												
Stage 2		p_1	p_2	p_3											
Stage 3			p_2	p_3	p_4										
...												
										p_{n-3}	p_{n-2}	p_{n-1}			
											p_{n-2}	p_{n-1}	p_0		
Stage f												p_{n-1}	p_0	p_1	

4.2.2 Surfaces

4.2.2.1 Control point sets

The control polyhedron (polygon) or the set of control points plays a very important role in parametric surface generation (as in parametric curves), this is because this object is the physical support for the creation of curves and surfaces that are used to model real world objects.

It can be obtained specific control [polyhedrae](#) (and control polygons) in many ways. 1) To define them with mathematical expressions such as parametric and geometric definitions, for instance a plane torus, a cone, a sphere, a Möbius band, etc.

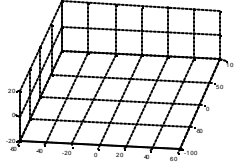


Figure 58. Mathematically obtained control points of a toroidal spiral.

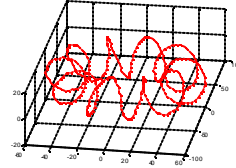


Figure 59. Curve fit on the control points of a toroidal spiral.

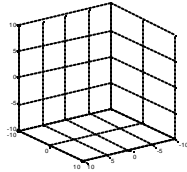


Figure 60. Mathematically obtained control points of a sphere.

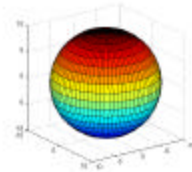


Figure 61. Surface fit on the control points of a sphere.

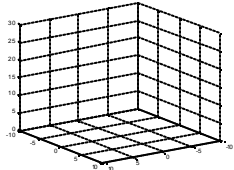


Figure 62. Mathematically obtained control points of a cone.

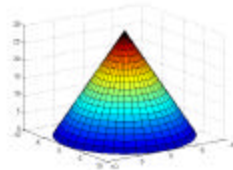


Figure 63. Surface fit on the control points of a cone.

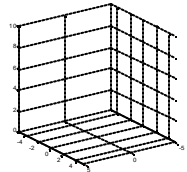


Figure 64. Mathematically obtained control points of a cylinder.

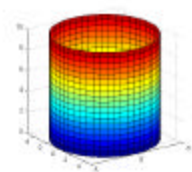


Figure 65. Surface fit on the control points of a cylinder.

Another technique, 2), is employed when such a surface or curve is not easily expressed in mathematical terms (Figure 66), it consists of obtaining or sampling the control points directly [from](#) the surface of an existing object. There are also many developed techniques in this class of control point set creation (digitalization). To mention just a few most common we have (in descending order of accuracy): Laser 3D scan, Range Imaging and tactile point capture. With the aid of these techniques together with complex surface reconstruction algorithms, intricate shapes like the backbone in Figure 67 can be accurately reconstructed for many scientific and engineering purposes.

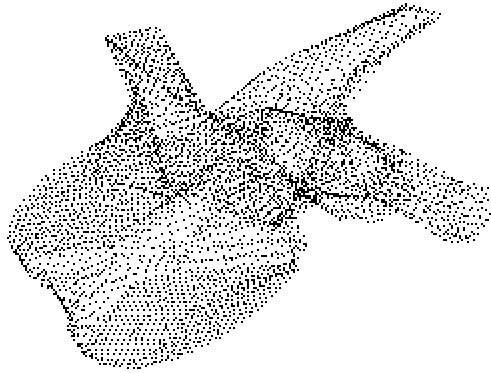


Figure 66. Digitized control points of a backbone



Figure 67. Reconstructed Surface of a backbone

4.2.2.1.1 Example of control Polyhedron generation (The Möbius Band)

When defining a control point set by mathematical means, it is advisable to design a parametric representation in which one or several geometric parameters can be controlled in a way to generate the required points in space.

In the following example a parametric technique of point generation is illustrated for creating the surface of a Möbius Band.

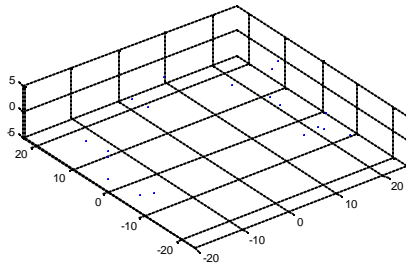


Figure 68. Final set of control points of a Möbius Band

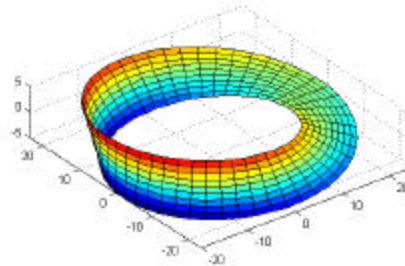


Figure 69. Mesh of a Möbius Band

It is important to exercise judgement of main parameters involved in a certain shape and recognize which are more suitable for describing the needed shape. In this particular case we conceive the control point set of the Möbius Band as an “army” of points on a generative line in space which rotates about an axis in space while it also rotates about its own midpoint (Figure 70). For a Möbius Band the number of rotations of the generative line about the midpoint is related with the revolution this line does about the axis ($\gamma = \frac{1}{2} \theta$). The sweep about the axis is 2π , and the rotation of the line about its midpoint will then be π . The revolution of a generative line of length L about an axis ($q = 2\pi$) and about the midpoint ($g = \frac{1}{2}$) are the main parameters that will be taken into account to make a characteristic Möbius Band.

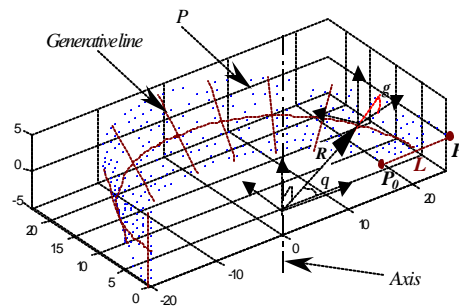


Figure 70. Parameterization of the Möbius Band.

The parameterization consists of controlling a point generative line (L) that rotates about an axis and at the same time about its midpoint. As for the points in the line, the coordinates of the extreme points P_0 and P_1 are defined as a function of the parameters mentioned above. Then a percentage or proportion expression produces the midway points between these extremes, and hence, an adequate set of control points is obtained.

The $[x, y, z]$ coordinates for the extreme points

$$P_0 = \left[\left(R - \frac{L}{2} \cos g \right) \cdot (\cos q), \left(R - \frac{L}{2} \cos g \right) \cdot (\sin q), \left(-\frac{L}{2} \sin g \right) \right]$$

$$P_1 = \left[\left(R + \frac{L}{2} \cos g \right) \cdot (\cos q), \left(R + \frac{L}{2} \cos g \right) \cdot (\sin q), \left(\frac{L}{2} \sin g \right) \right]$$

are related in a *proportion* expression for producing the rest of the points

$$P(a) = aP_1 + (1-a)P_0$$

where a is the longitudinal parameter and P is any point in the set.

An interesting variation of this example can be obtained by modifying the value of the angle (γ) that the line rotates about its midpoint. If, for example, this value is $\gamma = 1q$ then a shape like in Figure 71 would be obtained, and so on. The reader is invited to explore changing the values of the parameters for this figure in order to obtain other forms.

NOTE: It is advisable to use an independent cycle or loop to control each main parameter involved in the generation of the shape.

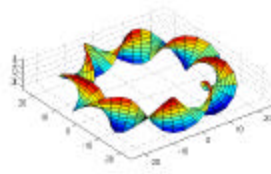
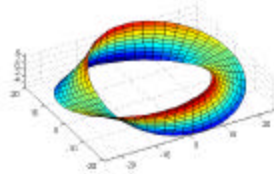
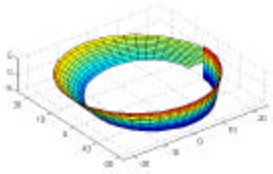


Figure 71. Band with parameter $g = 0.25q$

Figure 72. Band with parameter $g = 1q$

Figure 73. Band with parameter $\gamma = 4q$

See Appendix 7.2 for the MATLAB code developed for this example as well as for some of the control sets generated for Figure 58 to

Figure 64.

4.2.2.2 Parametric Surfaces

Parametric surfaces are generally defined by a vector function $\mathbf{p}():[0,1] \times [0,1] \rightarrow \mathbf{E}^3$:

$$\mathbf{p}(u,w) = (x(u,w), y(u,w), z(u,w))$$

The underlying structure is based on the definition of parametric curves. In other words, there exists no special formulation for surfaces. Instead, the formulation for curves is replicated into two dimensions. The general form for Bezier and Spline surfaces is therefore:

$$\mathbf{p}(u,w) = (x_1(u) \cdot x_2(w), y_1(u) \cdot y_2(w), z_1(u) \cdot z_2(w))$$

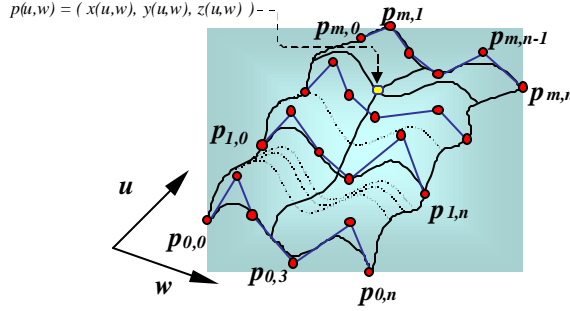


Figure 74. Disposition of the Control Polyhedron

Figure 74 shows the disposition of a control polyhedron, formed by $(m+1) \times (n+1)$ control points, to which a parametric surface $\mathbf{p}(u,w)$ is to be fit. A surface $\mathbf{p}(u,w)$ is a sequence of curves, each one corresponding to a value of u with internal parameter w . The formulation is exactly as in the curve case, the only difference being that the control points are now function of u , $\mathbf{Q}_n(u)$, as follows:

$$\mathbf{p}_u(w) = \mathbf{W}_n \cdot \mathbf{M}_n \cdot \mathbf{Q}_n(u)$$

The reasoning can be inverted to interpret the surface as a sequence of curves $\mathbf{P}_w(u)$, where each curve adjusts to control points determined by w , $\mathbf{Q}_m(w)$, and has internal parameter u , as follows:

$$\mathbf{p}_w(u) = \mathbf{U}_m \cdot \mathbf{M}_m \cdot \mathbf{Q}_m(w)$$

This symmetry leads to the formulation in Equation 44:

$$\boxed{\mathbf{P}(u,w) = \mathbf{U}_m \cdot \mathbf{M}_m \cdot \mathbf{Q}_{m,n} \cdot \mathbf{M}_n^T \cdot \mathbf{W}_n^T}$$

Equation 44. Formulation of surface patch with $(m+1) \times (n+1)$ control points.

4.2.2.3 Bezier Surface

As in the case of Bezier curves, the whole set of control points is used in one Bezier patch. As a consequence, the degree of the interpolating polynomials $B_{ui}(u)$ and $B_{wj}(w)$ is controlled by the number of rows and columns of the control polyhedron. Therefore, Bezier surfaces inherit characteristics of the Bezier curves: (a) adherence in position and tangency to the boundaries of the control polyhedron, (b) confinement within the convex hull $H(S)$ ($H(S)$ is a polyhedron in \mathbf{E}^3) of the control polyhedron S , (c) weakness in following events in the center of the stage (Figure 75 and Figure 76), given the low weight of the central coefficients (Figure 57).

As illustration, for a control polyhedron of size 3×4 points, the equation $Q(u,w) = UMuPMwTWT$, applied using the matrices M specific for Bezier case (

Table 11), becomes (the procedure for the “x” coordinate is shown, the same is done for the “y” and “z” components):

$$Q_x(u, w) = \begin{bmatrix} u^2 & u & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_{x1,1} & p_{x1,2} & p_{x1,3} & p_{x1,4} \\ p_{x2,1} & p_{x2,2} & p_{x2,3} & p_{x2,4} \\ p_{x3,1} & p_{x3,2} & p_{x3,3} & p_{x3,4} \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}^T \cdot \begin{bmatrix} w^3 & w^2 & w & 1 \end{bmatrix}^T$$

Equation 45. Example of Bezier Patch formulation for a 3×4 control polyhedron.

The reader is invited to verify the sizes of the matrices in Equation 45 to realize that it actually represents three matrix equations, each one with scalar result: $Q_x(u,w)$, $Q_y(u,w)$ and $Q_z(u,w)$. The individual components are calculated as $Q_x(u,w) = U \cdot M_u \cdot P_x \cdot M_w^T \cdot W^T$, $Q_y(u,w) = U \cdot M_u \cdot P_y \cdot M_w^T \cdot W^T$, $Q_z(u,w) = U \cdot M_u \cdot P_z \cdot M_w^T \cdot W^T$. Each matrix P_x , P_y , P_z must be the component x , y and z respectively of the matrix P in Equation 45, and therefore it must have the same layout (rows, columns and sorting).

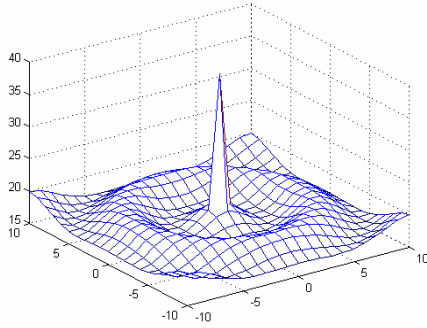


Figure 75. Control Polyhedron with 21×21 points.

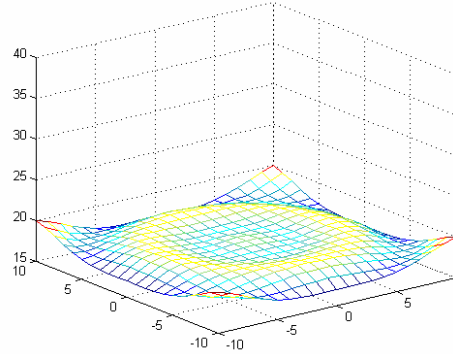


Figure 76. Bezier surface patch for a 21×21 control polyhedron.

4.2.2.4 Spline Surface

In analogous way to curve treatment, in which each stage of size k of a control polygon is extracted to fit a local Spline curve, a local Spline surface is fit to a subset of $k \times l$ points of the control polyhedron (see Figure 77). The problem is stated as to fit a parametric uniform B-Spline patch on a rectangular polyhedron of $(m+1) \times (n+1)$ points, using stages of size k out of the $(m+1)$ points and stages of size l out of the $(n+1)$ ones. The governing equation is:

$$Q_{s,t}(u, w) = U_k \cdot M_k \cdot P_{s,t} \cdot M_l^T \cdot W_l^T$$

Equation 46. Matrix formulation for stage of a Spline surface patch.

The notation used in Equation 46 is:

$(m+1)$	Number of point rows of the control polyhedron.
$(n+1)$	Number of point columns of the control polyhedron.
k	Points/stage and continuity control in the row direction of the control polyhedron.
l	Points/stage and continuity control in the column direction of the control polyhedron.
s	Index of current stage in the row direction. $s \in [1: m + 2 - k]$
t	Index of current stage in the column direction. $t \in [1: n + 2 - l]$
u	Parameter of stages in the row direction. $u \in [0, 1]$
w	Parameter of stages in the column direction. $w \in [0, 1]$
$P_{s,t}$	Subset of the control polyhedron, of size $(k \times l)$, located at coordinates (t, s) .
U_k	$U_k = [u^{k-1} \ u^{k-2} \ \dots \ u \ 1]$
W_l	$W_l = [w^{l-1} \ w^{l-2} \ \dots \ w \ 1]$

Figure 77 also suggests that the way to achieve a closed patch in both directions (m and n) is to recycle or reuse the control points in an analog manner to the one applied in closing Spline curves. Observe that patches, in general, allow independent formulations in the m and n directions. This obviously affects the order of the curves in each direction, but, more interestingly, applies to the fact that the curves in the m and n directions could have completely different characteristics (Spline, Bezier, NURBs, etc). The reader is invited to visit exercise 5.3.3 and 5.3.6 to analyze this aspect.

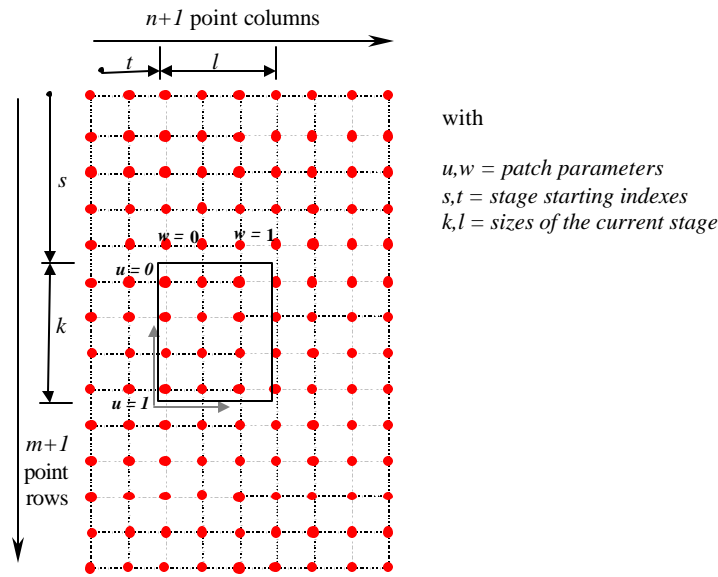


Figure 77. Variable map for Spline patch, according to control polyhedron.

Figure 78 and Figure 79 present the Spline version of the patch fit to the 21×21 control polyhedron in Figure 75. A portion of the total set of stages is presented in Figure 78, while the total patch is shown in Figure 79. Observe that the Spline formulation follows more faithfully the central spike.

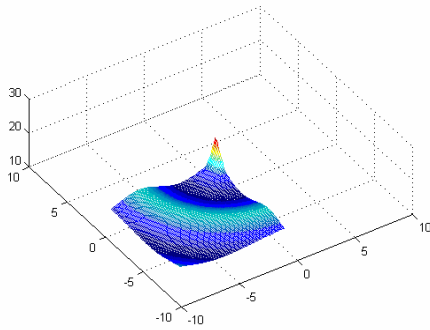


Figure 78. A quarter of the Spline stages for 21x21 control polyhedron

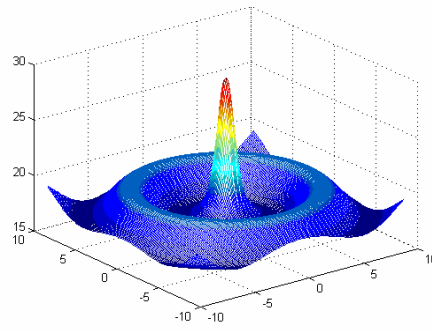


Figure 79. Spline patch for 21x21 control polyhedron (stages with $k \times l = 3 \times 4$ control points).

Figure 80 shows the control polyhedron for a torus, while Figure 81 displays the Spline patch, fitted with stages of $(k \times l = 3 \times 4)$ control points. Independent from the control polyhedron, either direction can be declared as “open” or “closed”, thus producing the effect shown in Figure 81.

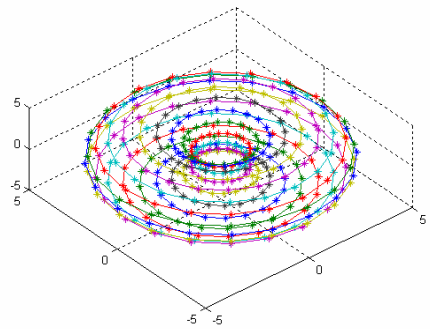


Figure 80. Control polyhedron for torus.

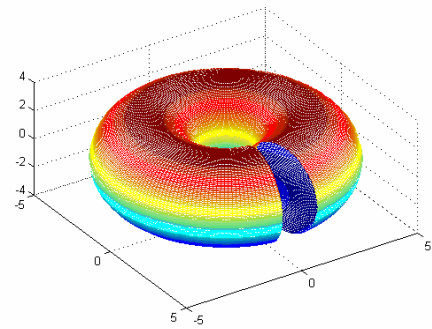


Figure 81. Spline patch for torus control points. Opened in u parameter, closed in w parameter.

4.3 EXERCICES - CURVES AND SURFACES -

4.3.1 Control point Generation of a Toroidal Spiral.

CONTROL POINT GENERATION (TOROIDAL SPIRAL)

OBJECTIVE

To exercise judgement of main parameters involved for mathematically describing a certain shape. Computationally.

PROCEDURE

You will generate the control point set of the spiral shown in Figure 82.

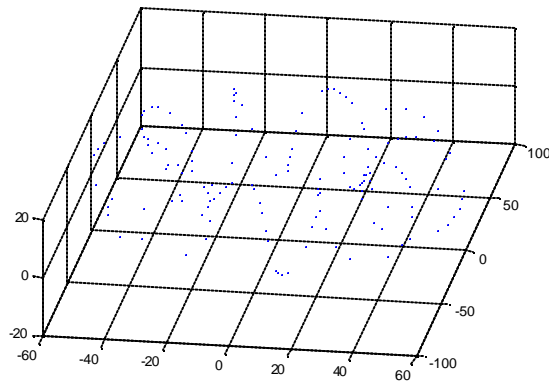


Figure 82. Control point set of a spiral in a torus

1. Write a function `[pts] = toroidal_spiral()`. This function should define or have as input the following arguments: R the radius of the underlying torus, r the radius of the spiral loops, a factor k that will control the number of loops the spiral will perform when swept around the torus axis.
2. Recognize the main parameters involved in the generation of this shape.
3. Define appropriate increasing steps for the parameters. For instance, $\delta\alpha$ as an increment step for an angular parameter.
4. Develop and Implement underlying parametric equations that control the generation of the coordinates of each control point.
5. The function should return an array of the coordinates of the control points as `pts`.

4.3.2 Bezier Interpolation Function.

CURVE INTERPOLATION

OBJECTIVE

To generate an introductory environment for parametric curves.

PROCEDURE

Given 4 points in E^3 , P_0, P_1, P_2, P_3 calculate their coefficients in a parametric curve, as a function of a u parameter.

A curve is a sequence of points, calculated with the formula:

$$f(u) = b_0(u)P_0 + b_1(u)P_1 + b_2(u)P_2 + b_3(u)P_3$$

Where: $b_0(u), b_1(u), b_2(u), b_3(u)$ are scalar functions (the set of weights for the points, evaluated for each value of parameter u). P_0, P_1, P_2, P_3 , are points of dimension (3×1)

The sequence is produced as the parameter u takes values $0, 0.1, 0.2, 0.3, 0.4, \dots, 1.0$

The functions $b_0(u), b_1(u), b_2(u), b_3(u)$ are:

$$b_0(u) = (1-u)^3$$

$$b_1(u) = 3u(1-u)^2$$

$$b_2(u) = 3u^2(1-u)$$

$$b_3(u) = u^3$$

ACTIVITIES

1. Write the 4 functions, $b_0(u), b_1(u), b_2(u), b_3(u)$ which calculate $b_i(u)$ for a given value of u .
2. Calculate the history of the coefficients $b_0=b_0(u), b_1=b_1(u), b_2=b_2(u), b_3=b_3(u)$ for u varying from 0 to 1.0 with increments of $du=0.1$. Store the history of u in U (upper case) and the $b_i(u)$ histories in B_i (upper case).
3. Plot the functions $b_0=b_0(u), b_1=b_1(u), b_2=b_2(u), b_3=b_3(u)$ against u (in the same window). Should you use a 2D or 3D plot? How many and which values does the independent variable u take?
4. Complete the code written in (2) as follows: for each element in the sequence of parameter u , $u = [0.1, 0.2, 0.3, 0.4 \dots 1.0]$ calculate $f(u)$ (see formula above) and extract its components fx, fy, fz ($f(u) = [fx, fy, fz]^T$). You should progressively fill up the row vectors X, Y, Z with $fx(u), fy(u), fz(u)$ respectively. Use the code written in (2).

$$X = [fx(0.0), fx(0.1), fx(0.2), fx(0.3), \dots, fx(1.0)]$$

$$Y = [fy(0.0), fy(0.1), fy(0.2), fy(0.3), \dots, fy(1.0)]$$

$$Z = [fz(0.0), fz(0.1), fz(0.2), fz(0.3), \dots, fz(1.0)]$$

5. From your work from (1) to (4), write a function $[X, Y, Z] = my_bzip(P_0, P_1, P_2, P_3)$ which interpolates the 4 points using the formula given above ($f(u)$). The arrays X, Y, Z are described in the previous numeral.

6. Write a *main* routine which:

- i. Initialize the required variables (du , number of interpolated points, etc.).
- ii. Initialize the points P_0, P_b, P_2, P_3 as you wish.
- iii. Call the function *my_bzr* () to calculate the interpolation X, Y, Z
- iv. Open a window and draw the control polygon $P_0 P_b P_2 P_3$ in E^3 .
- v. Draw the interpolation of the points P_0, P_b, P_2, P_3 , contained in X, Y, Z . Do not change window nor allow the previous one to be erased.

QUESTIONS:

1. Which are the dimensions of $bi(u)$? _____
2. Which are the dimensions of X, Y , or Z as function of du ? _____
3. Which are the dimensions of U as function of du ? _____
4. Which MATLAB function is used to draw a scalar function of one variable (i.e. $y = h(x)$)?

You have written the MATLAB code to draw a Bezier curve in E^3 .

4.3.3 Self-Defined Interpolation Function.

CONCEPTUAL EXERCISE. PARAMETRIC CURVES.

OBJECTIVE

To apply the concepts that govern parametric curves by defining an interpolation defined by the student.

PROCEDURE

1. Develop a set of interpolation functions B_0, B_1, B_2, B_3 , different from Bezier or Spline ones, for points P_0, P_1, P_2, P_3 , with the following conditions:

- 1.1. $0 \leq B_i(u) \leq 1$ ($i=0, \dots, 3$) for each $u \in [0, 1]$

- 1.2. $\sum_{i=0}^3 B_i(u) = 1$ for each $u \in [0, 1]$.

- 1.3. $B_i(u) = B_{3-i}(1-u)$, $i = 0..3$ (B_0 vs. B_3 , B_1 vs. B_2 should be symmetrical about $u=0.5$).

2. Use the coefficients developed, B_i ($i = 0..3$), to interpolate the group of points $P_0, P_1, P_2, P_3 \in E^3$ defined arbitrarily.

3. Plot:

- 3.1. $B_i(u)$ for $i=0..3$ and $u \in [0, 1]$. (the history of coefficients $B_i(u)$ as function of u).

- 3.2. $\sum_{i=0}^3 B_i(u) \cdot P_i = f(u)$ for $u \in [0, 1]$. (your own interpolation).

4.3.4 General Curve Interpolation Function.

EXERCICES CURVES SPLINE / BEZIER

OBJECTIVE

To develop a generic code for the interpolation of Spline or Bezier curves.

PROCEDURE

Given:

1. The type of curve: Spline ('s') vs. Bezier ('b').
2. The control polygon as an array of n points (size $3 \times n$).
3. The increment du : 0.1, 0.05 or 0.2.
4. The number of points per stage k (only for Spline curves). Possible values: 2, 3, 4.

Goal

To calculate the curve Spline or Bezier according to the parameters given by the user.

Observations:

1. Spline curves have several stages.
2. For Bezier curves, you may choose between (a) to use the whole set of control points in one stage, (b) to define several stages, with overlaps of exactly one control point between consecutive stages.
3. Draw the control polygon in the same window as the interpolated curve.
4. Your program must work for different combinations of the input variables (type of curve, control polygon, increment du , number of points per stage).

4.3.5 Control Point Generation of a Toroidal Surface.

CONTROL POINT GENERATION (TORUS)

OBJECTIVE

To exercise judgement of main parameters involved for mathematically describing a certain shape. Computationally, a cycle or loop should always be used to control each main parameter involved in the generation of the shape.

PROCEDURE

You will calculate a set of rectangular arrays of the coordinates of the control points (x,y,z) in E^3 , stored in variables P_x , P_y , P_z that describe the shape of a torus. The revolution axis will be the world coordinate Z axis.

1. Write a function $[P_x, P_y, P_z] = \text{torus_z}()$. This function should define or have as input the following arguments: R the *big* radius and r the *small* radius of the underlying torus, and other parameters you may consider necessary for the following tasks.
2. Define appropriate increasing steps for the parameters. For instance, $\delta\alpha$ as an increment step for an angular parameter.
3. Develop and Implement underlying parametric equations that control the generation of the coordinates of each control point.

This function must be able to control the amount of revolution in either senses or angular parameters. This is to say, an open torus may be generated with a slight modification in the parameters of this function, as shown in Figure 85 and in Figure 86.

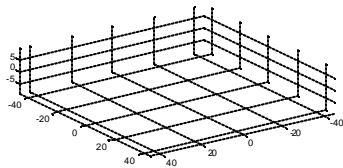


Figure 83. Mathematically obtained control points of a torus

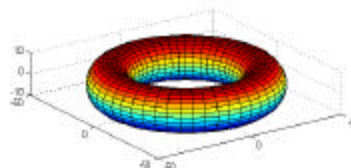


Figure 84. Surface of a torus

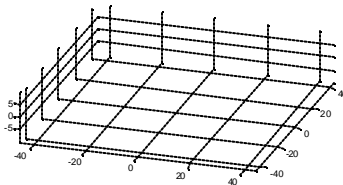


Figure 85. Control point set of a torus with an axial sweep of 1.5PI

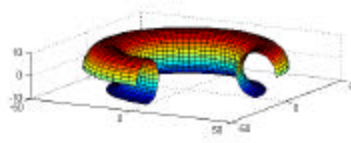


Figure 86. Surface of a torus with sweep of 1.5PI in both angular parameters

Con formato: Sin viñetas ni numeración

Eliminado:

Eliminado: ¶
 <#>Figure 85.Control point set of a torus with an axial sweep of 1.5PI¶
 ¶

Eliminado: ¶
 Figure 86.

4.3.6 Control Point Generation of a Conical Spiral Band.

CONTROL POINT GENERATION (CONICAL SPIRAL BAND)

OBJECTIVE

To exercise judgement of main parameters involved for mathematically describing a certain shape. Computationally, a cycle or loop should always be used to control each main parameter involved in the generation of the shape.

PROCEDURE

You will calculate a set of rectangular arrays of control points (x,y,z) in E^3 , stored in variables P_x , P_y , P_z that describe the shape of a band spread over a cone Figure 87. The revolution axis will be the world coordinate Z-axis.

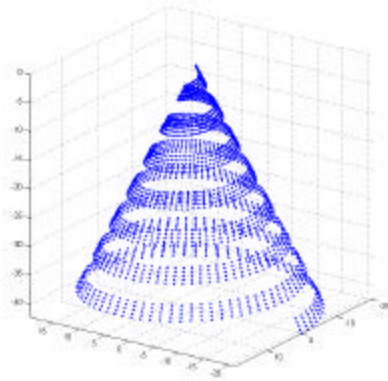


Figure 87. Mathematically obtained control points of a band on a cone

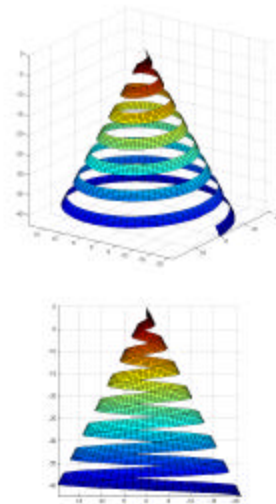


Figure 88. Mesh of a band on a cone

1. Write a function $[P_x, P_y, P_z] = \text{cone_band_z}(H, \alpha, L, \text{spins})$. This function should define or have as input the following arguments: H the height of the underlying cone, α the apex angle, L the size of the width of the band (length of generative line) and spins as the number of spins the band must accomplish from the base of the cone to its apex.
2. Recognize the main parameters involved in the generation of this shape.
3. Define appropriate increasing steps for the parameters. For instance, $\delta\alpha$ as an increment step for an angular parameter.
4. Develop and Implement underlying parametric equations that control the generation of the coordinates of each control point.

- An important expression that can be used for this problem is a length proportion expression as used to create the Möbius Band. See the solved example for the generation of the control points of the cone (Appendix 7.2).

4.3.7 Bezier Surface I.

BEZIER SURFACE I

OBJECTIVE

To use Bezier surfaces to prepare the student for future work with Spline surfaces.

PROCEDURE

You will calculate a Bezier surface for a rectangular array of control points (x,y,z) in E^3 , stored in variables *cpt_x*, *cpt_y*, *cpt_z* described below. The Bezier surface that you will produce must be stored in 3 rectangular matrices *BZR_X*, *BXR_Y*, *BZR_Z*. Therefore, your goal is to correctly fill *BZR_X*, *BXR_Y*, *BZR_Z* and to plot the resulting surface. See Figure 89.

Each point of the Bezier surface is calculated as a function of two parameters: u, v ($u = 0, du, 2du, 3du, \dots, 1.0$; $v = 0, dv, 2dv, 3dv, \dots, 1.0$). Note that du y dv are indicators of the mesh refinement. Typical values for du or dv are $0.05, 0.1, 0.2$. The algorithm should prompt the user or explicitly set values for du and dv .

- Write a function *contrl_pt*() which produce 3 matrices containing the control points for the surface.

$$cpt_x = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \end{bmatrix} \quad cpt_y = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{bmatrix} \quad cpt_z = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1.5 & 1.5 & 0 \\ 0 & 1.5 & 1.5 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

- Calculate, as function of du , the number of values of u that must be considered (N_cells_u).

$N_cells_u =$ _____

- Calculate, as function of dv , the number of values of v that must be considered (N_cells_v).

$N_cells_v =$ _____

$$[BZR_X(u,v), BZR_Y(u,v), BZR_Z(u,v)] = BZR(u, v, cpt_x, cpt_y, cpt_z)$$

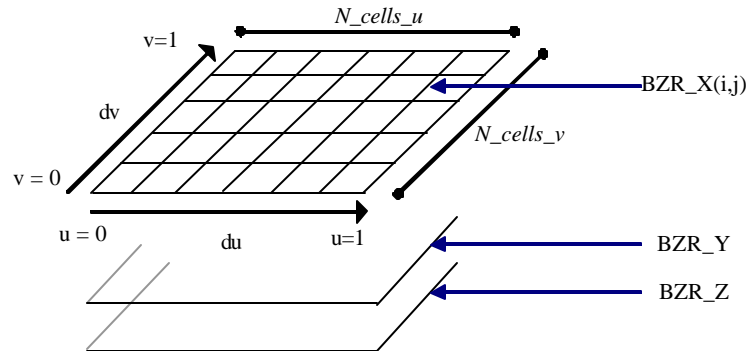


Figure 89. Three matrices which store the calculated surface.

4. Write the instructions (in MATLAB code) which allocate the exact amount of memory for matrices **BZR_X**, **BZR_Y**, **BZR_Z** and initialize it with zeros.
5. Specify the range of values that *i* and *j* should take the variable *i* is an index for elements in the *u* direction and the variable *j* is an index for elements in the *v* direction.
6. Write the instructions (in MATLAB code) which calculate the surface points (**BZR_X**(*u,v*), **BZR_Y**(*u,v*), **BZR_Z**(*u,v*)) for a given combination of (*u*, *v*) and a control polyhedral *cpt_x*, *cpt_y*, *cpt_z*.
7. Write the instructions (in MATLAB code) which draw the mesh of the control points *cpt_x*, *cpt_y*, *cpt_z*.
8. Write the instructions (in MATLAB code) which draw the Bezier surface **BZR_X**, **BZR_Y**, **BZR_Z**.
9. Write the instructions (in MATLAB code) in a script *main.m* to (a) fill the necessary data structures, (b) call the function to calculate the Bezier patch, and (c) plot both, the control polyhedral and its Bezier surface.

4.3.8 Bezier Surface II.

BEZIER SURFACE II

OBJECTIVE

To prepare the concept of local patch within a larger control polyhedron. To calculate a Bezier patch, for a local control polyhedron of size $l \times k$, extracted from position (b,a) of a larger control polyhedron, generated with an analytic form. The local Bezier patch will have maximal size 4×4 .

PROCEDURE

1. Create the control polyhedron:

P_x, P_y are square matrices of integer contents $[0,10] \times [0,10]$ (*meshgrid*).

P_z is a matrix generated with the following formula:

$$P_z = \left(\frac{\cos(x-5) \cdot P_x}{5} \right) \cdot \left(\frac{\cos(y-5) \cdot P_y}{5} \right)$$

Those matrices are to be created within a function $[P_x, P_y, P_z] = \text{contrl_p}()$.

2. Prompt the user for, or initialize in the code, the following variables:

du : value of increment for parameter u .
 dv : value of increment for parameter v .
 k : number of control points in direction u .
 l : number of control points in direction v .
 a : column origin of the local control polyhedron within the global control polyhedron.
 b : row origin of the local control polyhedron within the global control polyhedron.

3. Create a function $M = \text{calc_M}(k)$, which returns the Bezier coefficient matrix for k control points. $k = 2, 3$ or 4 .
4. Extract the local control polyhedron from the global one, starting in column a , row b , with size $k \times l$ respectively, by using the following instructions

$$\begin{aligned} pt_x &= P_x(b:b+l-1, a:a+k-1) \\ pt_y &= P_y(b:b+l-1, a:a+k-1) \\ pt_z &= P_z(b:b+l-1, a:a+k-1) \end{aligned}$$

5. Create a function $U = \text{calc_uv}(k, u)$, such that $U = [u^{k-1} u^{k-2} \dots u^1]$, for $k \in \mathbb{N}$, $u \in [1,0]$.
6. Create a function $[bezX, bezY, bezZ] = \text{Bezier}(U, V, Mu, Mv, pt_x, pt_y, pt_z)$, to calculate a point of the local Bezier patch. $U = [u^{k-1} u^{k-2} \dots u^1]$ and $V = [v^{l-1} v^{l-2} \dots v^1]$. Mu is the matrix of Bezier coefficients of size $k \times k$. Similarly, Mv is a matrix $l \times l$. pt_x, pt_y, pt_z is the local control polyhedron, extracted at position (b,a) from the global polyhedron. $bezX, bezY, bezZ$ are the coordinates of the calculated Bezier point.

7. To create the control loops (*for*) required to calculate and store all points $[bezX, bezY, bezZ]$ from the surface. Each Bezier point is calculated for a combination of parameters (u, v) . The sizes of the variables $bezX, bezY, bezZ$, required to store the local Bezier patch are dictated by du and dv .
8. To create a function which plots the calculated local Bezier surface $(bezX, bezY, bezZ)$. Title the resulting figure and name its axes. In another window, plot the global control polyhedral.

4.3.9 Generic Surface Interpolation Function.

HYBRID SPLINE / BEZIER SURFACES

OBJECTIVE

To develop a generic program, able to generate Spline, Bezier or Spline – Bezier_surfaces (different interpolations in rows and columns).

Given:

- surface type: Spline, Bezier, Spline-Bezier ('s', 'b', 'sb', 'bs'). 'sb' means interpolation Spline in rows and Bezier in the columns of the control polyhedral (similarly with 'bs').
- X, Y, Z : control polyhedral .Grid in X, Y, Z randomly generated.
- du : increment in value of parameter u ($du=0.1, 0.05, 0.2$).
- dv : increment in value of parameter v ($dv = 0.1, 0.05, 0.2$).
- k : number of control points in the u direction (columns) ($k=2,3,4$) to be taken per stage.
- l : number of control points in the v direction (rows) ($l=2,3,4$) to be taken per stage.

Observations:

1. Two windows must be used. The first one displays control polyhedron. The second shows the interpolated surface.
2. The program must work correctly with all possible combinations of the input data.

4.3.10 Bezier and Spline Surface Interpolation.

BEZIER AND SPLINE SURFACES

OBJECTIVE

To fit a parametric Spline surface on a generated control polyhedron. (*CONICAL SPIRAL BAND*)

PROCEDURE

You will develop a set of functions in order to define essential parameters and variables needed to create the Bezier and Spline surfaces of a conical band according to the given specifications.

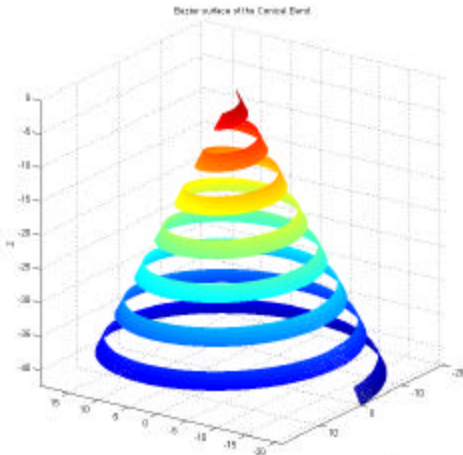


Figure 90. Bezier surface of a conical band

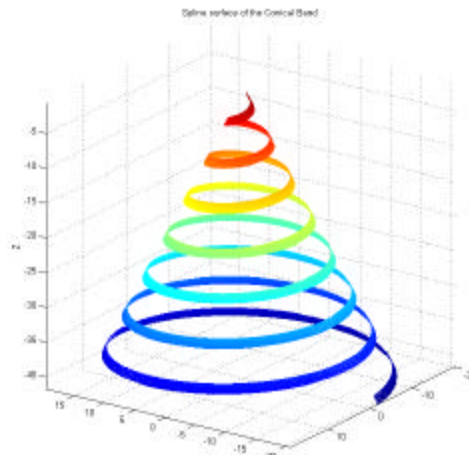


Figure 91. Spline surface of a conical band

NOTE: Unlike the Bezier surface interpolation, the Spline does not cover the whole range of points in width and length of the band. Notice that the Spline surface is thinner than the Bezier surface as well as shorter.

1. Write a MATLAB function $[Px, Py, Pz] = \text{gen_conical_band}(H, R, L)$ which generates the control points of a conical band in *MESH* format, given the height H , a radius R , and a width for the band L . The variables Px, Py, Pz obtained in 4.3.6 can be used here.
2. Develop a set of generic surface creation functions, both Bezier and Spline. The advantage of these functions is that they are useful for quickly building practically any surface given the control polyhedron in *MESH* format. Examine the scheme of Figure 77 in order to use the notation specified there in your code.

$[Bx, By, Bz] = \text{surface_bezier}(Px, Py, Pz, K, L, du, dv)$	$[PXs, PYs, PZs] = \text{surface_spline}(Px, Py, Pz, K, L, du, dv)$
$U = u_bezier(k, du)$	$U = u_spline(k, du)$
$M = m_bezier(k)$	$M = m_spline(k)$

3. Write a script MAIN.m in which the polyhedron generation routine is called and the surface functions are called and drawn in separate figure windows.

5. GEOMETRIC MODELING

The geometric modeling aided by computer implies the usage of mathematical representations and data to approximate virtual models to real objects, according to certain characteristics of the latter. The formalisms to represent objects are denominated representation *schemes (or simply schemes)*. An instance of data and relations affixed to a scheme (obviously with the purpose of representing a real object) are denominated a (*computational*) *model*. Typical schemes are *Constructive Solid Geometry (CSG)* and *Boundary Representation (B-Rep)*.

Some relevant issues to these representation schemes are the following ones:

- a) Given a real object R and a representation scheme f (for example *B-rep*, *CSG* or other) it is asked if R can be represented by more than a computational model f_1, f_2, \dots following this scheme. The answer is usually positive. For example, an object R can have different *CSG* images that represent it.
- b) Given two representation schemes, f_1 and g_1 , and the computational models of an object R in such schemes, f_1 and g_1 , it is asked if f_1 can be translated to g_1 and vice versa. The answer is usually negative. For example once a model has been translated from scheme *CSG* to *B-Rep*, the inverse procedure is not possible.
- c) Given a computational model f_1 in a given representation scheme f_1 , it is asked if this model represents one and only one object of the real world. The answer is usually positive for representations which allow the manufacturing of the object, while in those which imply only the graphic display the answer can be negative. For example in the “*wireframe*” scheme a single model may correspond to several real objects which are completely different. Of course, in engineering applications it is expected that the model f_1 does not imply a non existent object (Escher type).


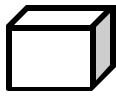


This chapter examines briefly some representation schemes and applies notions such as Geometric Transformations, Curves and Surfaces and Topology to define consistent and non ambiguous representations.

In *Computer Aided Geometric Design (CAGD)* the term **geometry** refers to the position and shape of an entity in space, while the term **topology** refers to the relations of vicinity, collective relations, number of holes, etc, which are in such entity. Therefore, points, lines, planes, curves, surfaces, coordinate systems are elements with geometric information, while segments, faces, facets, loops, shells, bodies etc. are topologic objects. Basically topology is a branch of mathematics in charge of studying the continuity and connectivity as well as preserving them when the figures are deformed.

In topology, properties like areas, lengths, volumes or angles are not considered. Topology is not interested in the metric properties of the geometric entities. Instead, it is interested in the properties that remain unchanged among transformations that scale, twist or compress the figure without breaking, perforating or creating self-intersections.

Two figures can be *geometrically* different and *topologically* equivalent. It is said that two geometric entities are *topologically* equivalent when one of them can be transformed into the other one by means of a continuous mapping, one-to-one and onto (Table 15).

Table 15. Example of topologic equivalence and non equivalence

OBJECT 1	OBJECT 2	RELATION
		Topologically Equivalent
		Topologically non-equivalent

The following schemes will be examined in this chapter: a) Decompositions, b) Constructive Solid Geometry and c) Boundary Representation. Although many other variations exist, it is considered that those mentioned cover a conceptual range enough to understand any other.

5.1 Constructive Solid Geometry (CSG)

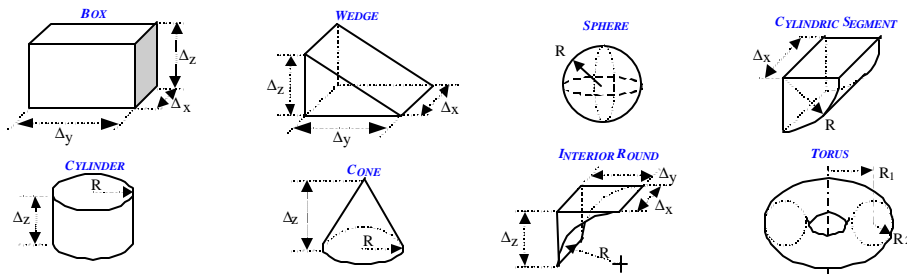


Figure 92. Primitive Entities

The Constructive Solid Geometry uses boolean operations to build a model of an object with a complex shape. The information obtained from this method which describes the model is stored in a structure called *tree*, as in Figure 93 in which the *leaf Nodes* (those with no descendants) represent predefined lumps of the space E^3 , called primitives (Figure 92). Some basic primitives are $Box(Dx, Dy, Dz)$, $Wedge(Dx, Dy, Dz)$, $Sphere(R)$, $Cylinder(Dz, R)$, $Cone(Dz, R_0, R_f)$, $Torus(R_1, R_2)$, etc.

Each node that is not a leaf represents an operation that is executed on its descendants forming a new body. This result is a compact subset of E^3 which generally is no longer expressible as a primitive. The so-called boolean operations (Union, subtraction, Intersection) (Figure 94) are binary since they require two arguments, while the geometric transformations are unitary operations since they operate on a single argument.

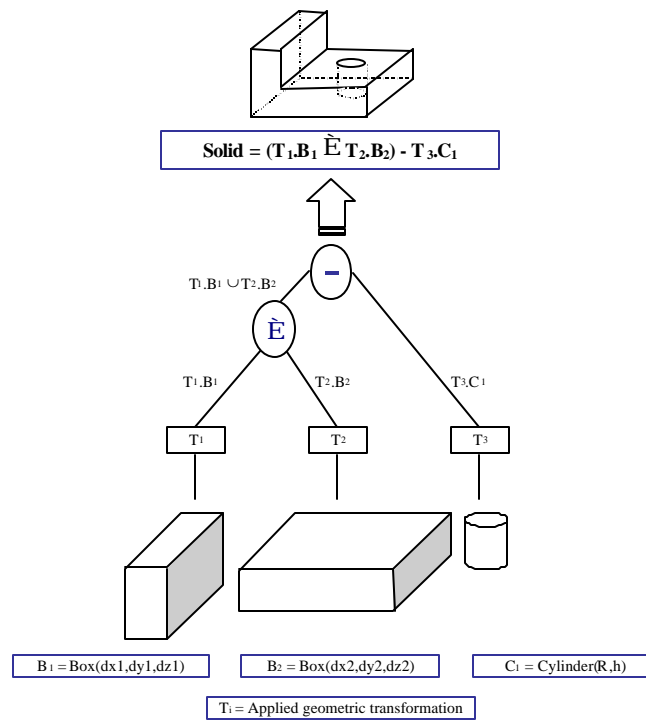


Figure 93. Example of CSG tree

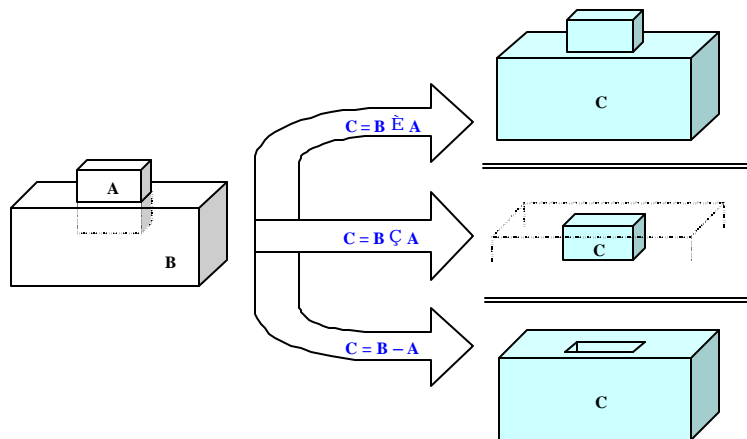


Figure 94. Boolean Operations

5.2 Boundary Representation (B-Rep)

A “surface” (Figure 95) is mathematically defined as a 2-manifold M embedded in E^3 . Informally, this means that every ball $B(p,r)$ of radius r , centered in p (p being a point of set M) intersects M in a set D which is isomorphic with a planar disc.

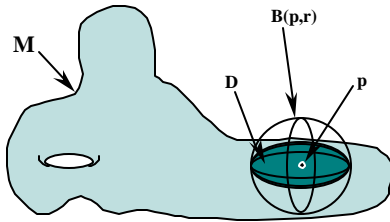


Figure 95. Definition of 2 Manifold in E^3

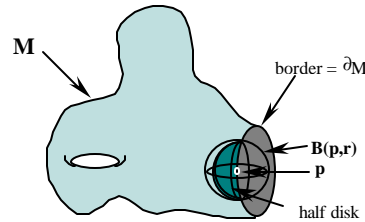


Figure 96. Definition of 2 Manifold with boundary, in E^3

When M is not completely closed, there are points (called *boundary* or *border* points) at the “edge” of M for which the disc D mentioned above is transformed into a semi-disc. In such case it is said that M is a 2-manifold with border, embedded in E^3 .

The boundary representation (B-rep) uses the convention that a body is uniquely expressed by its boundary M , which is a 2-manifold in E^3 (Figure 97). For that, it is necessary to specify what is the “interior” of M , with the help of basic concepts of vector calculus. Notice that if M has borders, it is impossible to define interior vs. exterior. The 2-manifolds with boundary are essential to define non closed “shell s”, of vital importance in applications of machining CNC, stereolithography, visualization, etc. Additionally, Finite Element Analysis software (FEA) usually requires “shell” data rather than a solid object (defined further on).

In some applications (for example the *Marching Cubes* algorithm) the decision about if a point in E^3 is *on*, *inside* or *outside* M , is executed if there is a scalar function $f(x,y,z)$. If $f(x,y,z) = 0$ the point is *on* M , if $f(x,y,z) > 0$ the point is *inside* M and if $f(x,y,z) < 0$ the point is *outside* M . Such a function $f(\cdot)$ is not always available for the designer, but the B-rep or boundary of the solid plays such a role.

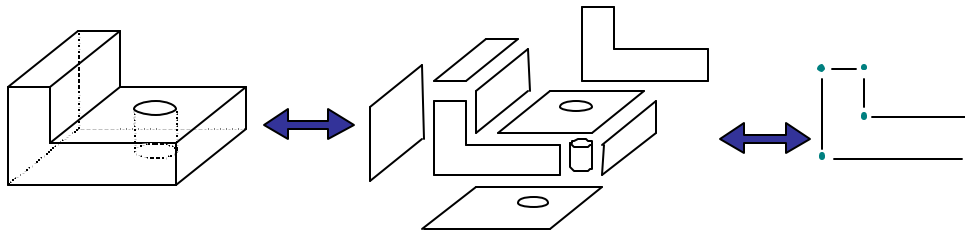


Figure 97. Boundary Representation

The B-rep schemes require a strict hierarchy of geometric and topologic entities. Although every geometric modeler (ACIS, ParaSolids, IDEAS, CATIA, etc.) uses different names, a typical hierarchy is shown in Table 16.

Table 16. Relations and hierarchy of topologic and geometric elements in B-rep

TOPOLOGIES		GEOMETRIES	
BODY	Set of possibly disconnected solid regions or LUMPs		
LUMP	A solid connected region, bounded by a SHELL		
SHELL	The boundary of a LUMP region. A 2-manifold in general without border.		
FACE	A connected subset of points belonging to one SURFACE. The subset is bounded by closed contours (<i>loop</i>) formed by EDGES (<i>edge</i>) contained in the SURFACE.	SURFACE	Analytic surface, in parametric form $[X(u,v), Y(u,v), Z(u,v)]$ or implicit form $f(x,y,z)=c$.
LOOP	Closed non-autointersecting path, formed by EDGES and fully contained in a SURFACE carrier.		
EDGE	A connected subset of points belonging to a CURVE. Two VERTEX, contained in the CURVE bound the subset.	CURVE	Analytic curve, in parametric form $[X(u), Y(u), Z(u)]$ or implicit form $f(x,y,z)=c$.
VERTEX	A connected subset of points belonging to a POINT. Obviously there is only one POINT in such subset.	POINT	(x,y,z) in E^3

Notes:

1. A BODY is composed by several LUMPs or disconnected solid portions.
2. A LUMP is a compact, connected, bounded set of points p in E^3 . A LUMP is bounded by SHELLs. If the LUMP has inner cavities it is bounded by more than one SHELL.
3. A SHELL is a connected portion of a LUMP's boundary. It is formed by points p of the LUMP such that a ball of radius $r>0$ centered in p has points inside and outside the LUMP.
4. A SHELL is a connected subset of a LUMP boundary. A SHELL is composed by several FACES.
5. A FACE is a connected portion of a SHELL. A FACE is bounded by LOOPS. If the FACE has holes, it is bounded by several LOOPS. Otherwise, it is bounded by one LOOP.
6. A LOOP is composed by a set of EDGES.
7. An EDGE is bounded by 2 VERTEX.

8. Figure 95 An EDGE limits exactly two FACES. Additionally, an EDGE $e = (v_0, v_f)$ in FACE f is traveled as $e^* = (v_f, v_0)$ in FACE f^* (the neighbor to f by e). The EDGE e^* is denominated (in some texts) as the partner of e , and vice versa. If e is part of the border of M (that is to say M is a manifold with border Figure 96), it appears only in one FACE.
9. The EDGEs of the most external LOOP of a FACE are traversed in counterclockwise direction (CCW), in accordance with the external normal of the FACE. The edges of internal LOOPS, corresponding to holes of the FACE are traversed in the clockwise direction (CW).

Explanatory notes about Topology

In the following definitions and observations the universal set is $U (U \hat{=} E^3)$.

1. BALL ($B(r,p)$) is a sphere centered in p of radius r to the set

$$B(r,p) = \{ q \in U \mid d(p,q) \leq r \}$$

All the points belonging to the space U that are inside a sphere of radius r centered in p .

2. Boundary (∂A) of a set A in U .

$$\partial A = \{ p \in U \mid \forall r > 0 (B(p,r) \cap A \neq \emptyset \wedge B(p,r) \cap A^c \neq \emptyset) \}$$

Those points on which every Ball centered falls partially outside and partially inside A .

3. Closed set A in U .

$A \hat{=} U$ is closed if $\partial A \hat{=} A$. If $\partial A \hat{=} A^c$ (the complement of A) then A is *open*. Some sets are not strictly open (or closed) since they contain only part of their boundary.

4. Bounded set in U

$A \hat{=} U$ is bounded if there exist a $r > 0$ and a point $p \in U$ such that $A \hat{=} B(p,r)$.

In other words, some ball centered in some point of U completely contains A .

5. Disconnected sets A, B in U .

$A \hat{=} U$ and $B \hat{=} U$ are disconnected if " $a \in A, b \in B, ab \cap (A \cup B)^c \neq \emptyset$ ".

All the segments that join points of A with points of B have a portion outside A y and B .

6. Compact set.

$A \hat{=} U$ is compact if it is closed and bounded.

7. BODY.

A BODY B is a compact (possibly disconnected) set in U .

8. LUMP.

A LUMP L is a compact connected set in U .

9. SHELL.

A SHELL SH is the boundary of a LUMP ($SH = \partial L$)

10. FACE.

A FACE F is a connected subset of a SHELL $S (F \hat{=} SH)$. Therefore a SHELL is the union of FACES. A FACE must be a connected set, however it may have holes. In this case the boundary of the FACE, ∂F , will have several disconnected elements (called LOOPS). To calculate the boundary of a FACE F it is necessary to understand that F is embedded in a geometry $S(u,v)$ (that is called "the

carrier surface" of F). It is established that $U = S(u, v)$ and $F' = S(u, v) - F$. In other words, $S(u, v)$ is the universal set where the complement of F is calculated from. Notice that $F \cap S(u, v)$.

11. LOOP.

A LOOP L is a connected component of the boundary of a FACE. The boundary of a FACE is formed with several closed LOOPS L_0, L_1, \dots, L_n , where $L_0 \cap L_1 \cap \dots \cap L_n = \emptyset$. The LOOP L_0 will be the external boundary and L_1, L_2 , etc will be the internal boundaries (i.e. the holes) of F .

12. EDGE.

An EDGE E is a connected subset of a LOOP L . Therefore a LOOP can be found as the union of EDGES. A LOOP must be connected and cannot have gaps (interruptions of the LOOP). To calculate the boundary of an EDGE E it is necessary to understand that E is embedded in a geometry $C(u)$ (that is denominated "the carrier curve" of E). Therefore $U = C(u)$ and $E' = C(u) - E$. In other words, $C(u)$ is the universal set where the complement of E is calculated from. Notice that $E \cap C(u)$.

13. VERTEX.

A VERTEX V is part of the boundary of an EDGE. Each EDGE E has exactly two VERTICES such that $\{V_0\} \cap \{V_1\} = \emptyset$. Notice that $\{V_0, V_1\} \cap C(v)$. The carrier geometry of a VERTEX V is a POINT $(x, y, z) \in E^3$. Notice that two VERTICES can be carried by the same POINT (x, y, z) . An example of that would happen in an EDGE E that starts and ends at the same POINT (x, y, z) of E .

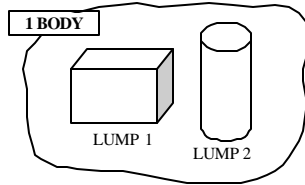


Figure 98. BODY with two LUMPS

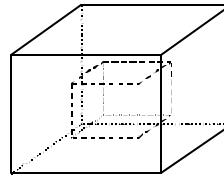


Figure 99. LUMP bounded by two SHELLs

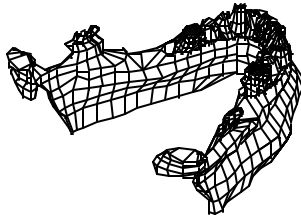


Figure 100. SHELL (manifold) without border

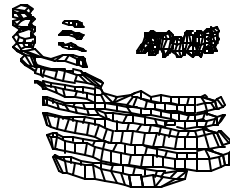


Figure 101. SHELLs (Manifolds) with border

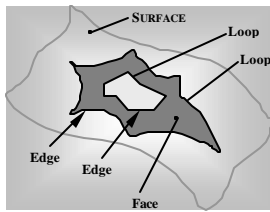


Figure 102. FACE with hole, its surface (LOOPS formed by EDGES) and its carrier surface (SURFACE).

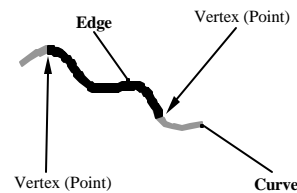


Figure 103. EDGE, its boundary (VERTICES) and its carrier curve (CURVE)

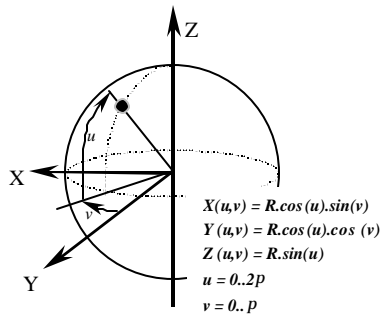


Figure 104. Example of a parametric surface (SURFACE)

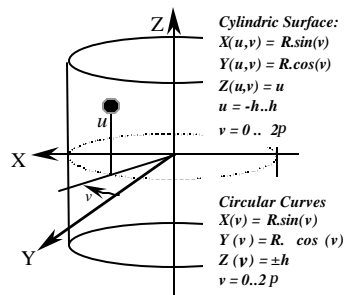


Figure 105. Example of a surface (SURFACE) and parametric curves (CURVE)

5.2.1 Example. Relationship between Topology and Geometry

The following example shows two bodies with same topology and different geometry

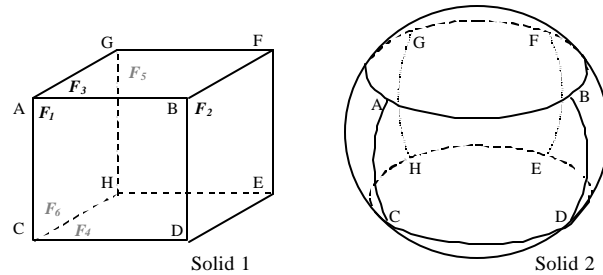


Figure 106. Topologic Equivalency under Geometric difference

Observation: The EDGE AC appears exactly in two FACES ($F1$ and $F6$). In $F1$ it is traveled as AC while in $F6$ it is traveled as CA . The same thing is certain for each EDGE of a manifold M without border.

Table 17. Specification of topologies and geometries in a body

	TOPOLOGY						GEOMETRY	
							Solid 1	Solid 2
Body:	B1							
Lump:	L1							
Face:	F1	F2	F3	F4	F5	F6	Planes	Sphere
Loop:	L1	L2	L3	L4	L5	L6	Lines	Curves
Edge:	AC CD DB BA	EF FB BD DE	GA AB BF FG	DC CH HE DE	EH HG GF FE	AG GH HC CA		
Vertex:	A	E	G	D	E	A		
	C	F	A	C	H	G		
	D	B	B	H	G	H		
	B	D	F	E	F	C	Points	

5.3 Example 1. Definition of Boundary Representation

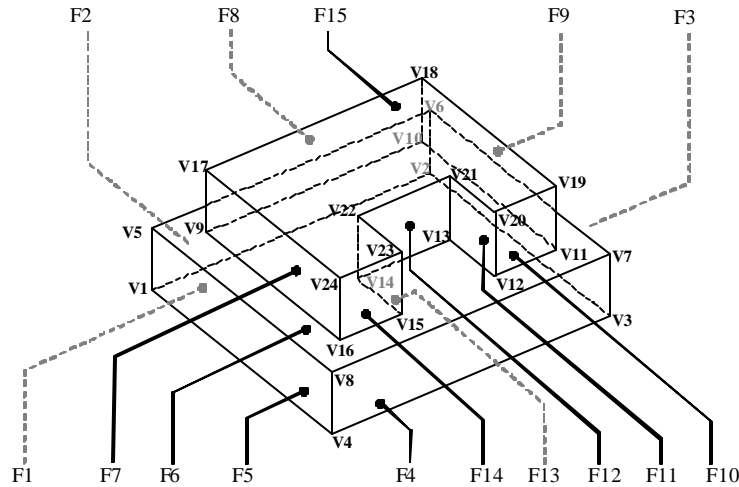


Figure 107. Example 1 Nomenclature of entities for the B-Rep topologic analysis of a body

Next, the boundary representation is developed (Table 18) as well as the development of the construction through CSG (Figure 108), for the body shown in Figure 107.

5.3.1 Boundary Representation

In order to understand the development of the example; follow the notation used in Figure 107. A FACE (F) is a closed contour or LOOP (L) conformed by points VERTEX (V).

Table 18. Example 1. Table of Boundary Representation of a rigid body

Body	B1														
Lump	L1														
Shell	S1														
Face	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
Loop	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15
Vertex	V1	V1	V6	V8	V1	V6	V10	V17	V17	V18	V19	V20	V21	V23	V23
	V2	V5	V7	V4	V4	V5	V11	V9	V18	V19	V20	V21	V22	V15	V24
	V3	V6	V3	V3	V8	V8	V12	V16	V10	V11	V12	V13	V14	V16	V17
	V4	V2	V2	V7	V5	V7	V13	V24	V9	V10	V11	V12	V13	V15	V24
							V14								V23
							V15								V22
							V16								V21
							V9								V20

For this example, the geometric carriers of the FACES are planes formed by straight lines.

5.3.2 CSG Representation

It was implemented, in the CSG construction of the body in Figure 107, only one type of primitive: *Box*. In order to achieve a correct construction of the solid (*S*), it is necessary to apply certain transformations (*T_i*) on every primitive for proper placing of each with respect to existing entities.

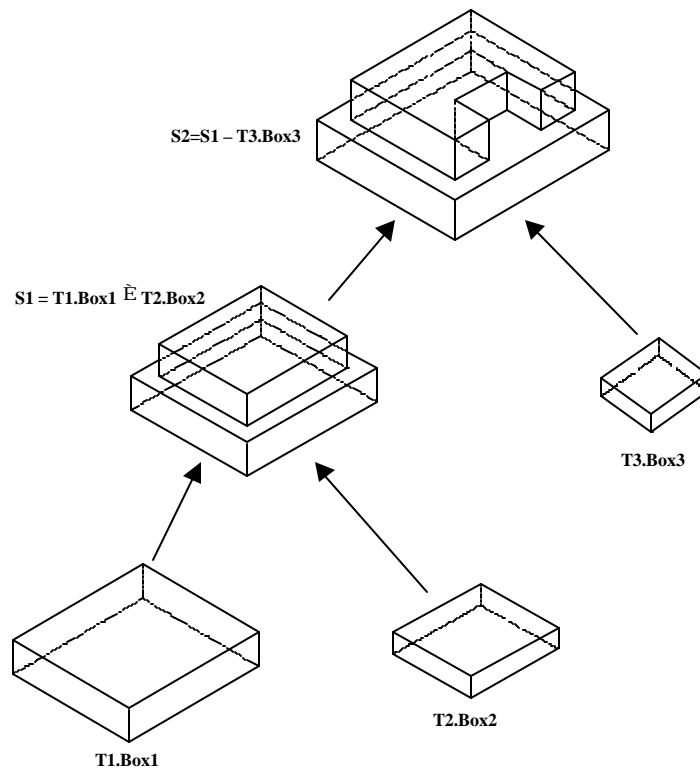


Figure 108. Example 1. CSG Representation

5.4 Example 2. Definition of Boundary Representation

The following is an example of a non - manifold. It is a non - manifold because there is an edge (V3-V4) that is shared by more than two faces (namely four). The final representation is based on two LUMPs belonging to the same BODY.

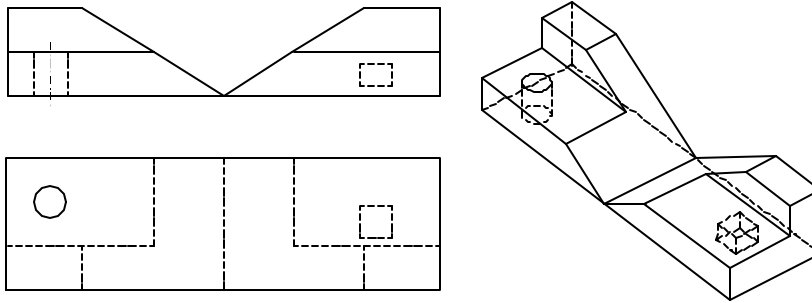


Figure 109. Example 2. Rigid Body (Non manifold)

Next the Boundary Representation is developed (Table 19) and the development of the construction through CSG (Figure 111), for the body shown in Figure 109.

5.4.1 Nomenclature of entities

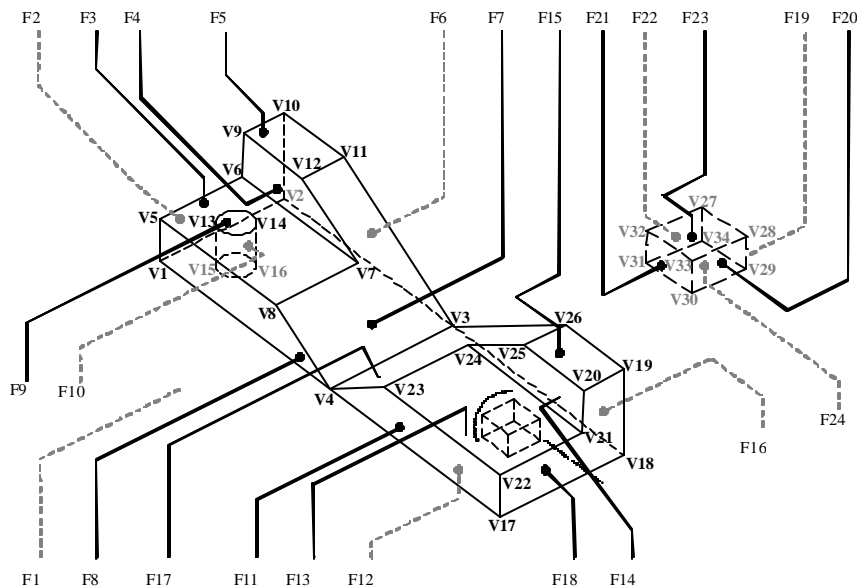


Figure 110. Example 2 Nomenclature of entities for the B-Rep topologic analysis of a body.

5.4.2 Boundary Representation

As in the previous example, in order to understand the development of this one; follow the notation used Figure 110. A FACE (F) is a closed contour or LOOP (L) conformed by points VERTEX (V).

Table 19. Example 2. Tables of Boundary Representation of a rigid body

Body	B1											
Lump	L1											
Shell	S1											
Face	F1		F2	F3		F4	F5	F6	F7	F8	F9	F10
Loop	L1	L1-1	L2	L3	L3-1	L4	L5	L6	L7	L8	L9	L10
Vertex	V1	V15 V16	V5	V5	V13	V9	V10	V11	V11	V8	V13	V13
	V2		V6	V8	V14	V6	V9	V3	V12	V5	V15	V14
	V3		V9	V7	V7	V12	V2	V7	V1	V16	V16	
	V4		V10	V6	V12	V11	V10	V8	V4	V14	V15	
			V2					V4				
			V1						V3			

Body	B1													
Lump	L2													
Shell	S1								S2					
Face	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24
Loop	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22	L23	L24
Vertex	V4	V3	V24	V20	V19	V26	V26	V19	V29	V29	V30	V27	V28	V29
	V17	V18	V23	V25	V26	V19	V3	V20	V28	V30	V31	V32	V33	V34
	V22	V17	V22	V24	V25	V18	V4	V21	V27	V33	V32	V31	V32	V31
	V23	V4	V21	V21	V20	V3	V23 V24 V25	V22 V17 V18	V34	V28	V33	V34	V27	V30

In this example, the geometric carriers of the FACES are planes formed by straight lines except for the cases of F9 and F10, where the EDGES V13-V14 and V15- V16 carry curved lines. Hence, the faces bounded by these edges are non-planar surfaces.

5.4.3 CSG Representation

The CSG construction of this body is shown in Figure 111; The following primitives were used: *Box*, *Cylinder* and *Wedge*. In order to achieve a correct construction of the solid (S), it is necessary to apply certain transformations (Ti) on every primitive for proper placing of each with respect to existing entities.

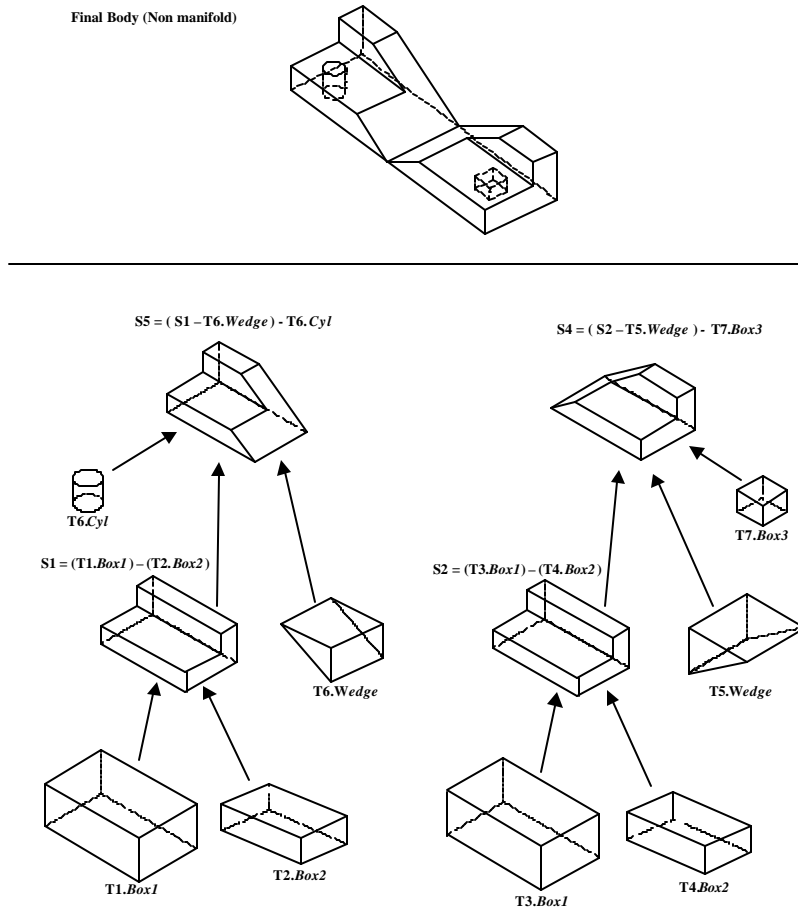


Figure 111. Example 2. CSG Representation

Notice that since this is a non manifold, a final union is not performed and so, for the sake of this example the tree is left as shown treating the object as two different LUMPs placed together. The fact of being a non manifold also indicates that this object cannot be achieved in reality exactly as is being represented here.

1.2 Assign values to the geometries of Figure 112. You must specify the POINTs and analytic forms for all curved geometry present. The curves and/or surfaces must be written in the form $F(x,y,z) = c$, with the ranges for x,y and z clearly defined. Give names to all curved geometries.

Solution:

V1 = [A, 0, C, 1]	V12 = [J+R,E,C, 1]	G15: Cylinder: $(x-H)^2 + (y-I)^2 = R^2$ $G \leq z \leq F$
V2 = [0, 0, C, 1]	V13 = [J+R,E,D, 1]	G16: Cylinder: $(x-H)^2 + (y-I)^2 = R^2$ $G \leq z \leq F$
V3 = [A, 0, 0, 1]	V14 = [J-R,E,D, 1]	G17: Cylinder: $(x-J)^2 + (y-E)^2 = R^2$ $D \leq z \leq C$
V4 = [0, E, C, 1]	V15 = [A, E, C, 1]	G18 -v19: Circumference: $(x-H)^2 + (y-I)^2 = R^2$ $z = F$
V5 = [0, E, 0, 1]	V16 = [0, 0, 0, 1]	G17 -v20: Circumference: $(x-H)^2 + (y-I)^2 = R^2$ $z = G$
V6 = [A, E, 0, 1]	V17 = [H+R, I, G, 1]	G18 -v19: Circumference: $(x-H)^2 + (y-I)^2 = R^2$ $z = F$
V7 = [0, B, 0, 1]	V18 = [H+R, I, F, 1]	G17 -v20: Circumference: $(x-H)^2 + (y-I)^2 = R^2$ $z = G$
V8 = [A, B, 0, 1]	V19 = [H-R, I, F, 1]	G11 -v12: Circumference: $(x-J)^2 + (y-E)^2 = R^2$ $z = C$
V9 = [A, B, K, 1]	V20 = [H-R, I, G, 1]	G13 -v14: Circumference: $(x-J)^2 + (y-E)^2 = R^2$ $z = D$
V10 = [0, B, K, 1]		
V11 = [J-R,E,C, 1]		

Write the Topological structure for the solid of Figure 112.

1.3 For topologies carried by curved geometries (surfaces and / or lines) indicate the name that you assigned in the previous point to such geometries.

Solution:

Curved carrier geometries appear in bold in the cells (Table 20) corresponding to the initial or final vertex of the curved EDGE, or in the curved FACE, that they carry. Two LUMPS are defined in order to have manifold topology.

Table 20. Example 3. Table of Boundary Representation of a rigid body

BODY												
LUMP	L1											
SHELL	Sh1								Sh2			
FACE	F1	F2	F5	F8	F9	F11	F12	F17 GF17	F13	F14	F15 GF15	F16 GF16
LOOP	L1	L2	L5	L8	L9	L11	L12	L17	L13	L14	L15	L16
VERTEX	v1	v1	v15	v2	v2	v3	v14	v12	v18 G18-19	v17 G17-20	v18 G18-19	v17 G17-20
	v15	v3	v6	v4	v16	v16	v13 G13-14	v13	v19 G18-19	v20 G17-20	v17 G17-20	v18 G18-19
	v12	v6	v5	v5	v3	v5		v14			v20 G17-20	v19 G18-19
	v11	v15	v4	v16	v1	v6		v11 G11-12			v19	v20 G17-20
	v4		v11									
	v2		v14									
			v13									
			v12									
LUMP	L2											
SHELL	Sh3											
FACE	F3	F4	F6	F7	F10							
LOOP	L3	L4	L6	L7	L10							
VERTEX	v6	v7	v6	v9	v5							
	v8	v5	v9	v8	v7							
	v9	v10	v10	v7	v8							
			v5	v10	v6							

1.4 Write the CSG tree of the solid in Figure 112. You must give the necessary dimensions for the primitives *BLOCK(DX,DY,DZ)* and *CYLINDER(R, DZ)* (created with center of gravity in the origin (0,0,0). Figure 113). and *WEDGE(DX,DY,DZ)*. Likewise, you must specify the geometric transformations required to position such primitives in the right places in order to participate in the boolean operations.

1.4.1 List of primitives and names.

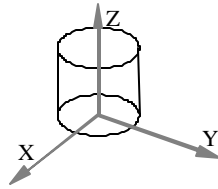


Figure 113. Cylinder.

Solution:

```

B1 = BLOCK(A, E, C )           // large block
B2 = WEDGE(A, B-E, K )         // wedge
C1 = CYLINDER(R, (C-D) )       // cylinder for the seat. Any height larger than (C-D) works fine,
                                // but it will affect the geometric transformation required to position
                                // the primitive!!!.
C2 = CYLINDER(R, (F-G) )       // internal cylinder
    
```

1.4.2 List of Geometric transformations and their names.

Solution:

```

M1 = trans(A/2 , E/2, C/2 )    // apply M1 to B1
M2 = trans(A/2, B, 0)*rot( Z,180) // apply M2 to B2
M3 = trans( A/2, E, [D+(C-D)/2] ) // apply M3 to C1
M4 = trans( H, I, [G+( F-G )/2] ) // apply M4 to C2
    
```

1.4.3 CSG Tree

Solution:

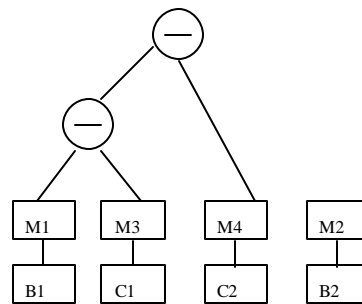


Figure 114.Example 3. CSG Tree

- 1.5 Define with equations the curved surface that you consider, is the basis to represent the curve surfaces of the model. Define the basic surface in the local space of size $1 \times 1 \times 1$ units³ of Figure 115.

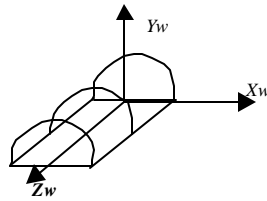


Figure 115. Local Space of coordinates for definition of curved surface.

Your equations must have the form:

$$\begin{aligned} X &= X(I_1, I_2) \\ Y &= Y(I_1, I_2) \\ Z &= Z(I_1, I_2) \end{aligned}$$

Choose, explain and draw the parameters λ_1, λ_2 , in similar way as the used to define the Mobius Band, the Torus and the surface $\sin(R)/R$.

Solution:

$$\begin{aligned} x(I_1, I_2) &= R \cdot \cos(I_1), \\ y(I_1, I_2) &= R \cdot \sin(I_1), \\ z(I_1, I_2) &= I_2, \\ 0 \leq I_1 \leq 2\pi, \quad 0 \leq I_2 \leq 1 \end{aligned}$$

or (since the cylinder is displaced in X direction),

$$\begin{aligned} x(I_1, I_2) &= R \cdot \cos(I_1) - 1, \\ y(I_1, I_2) &= R \cdot \sin(I_1), \\ z(I_1, I_2) &= I_2, \\ 0 \leq I_1 \leq 2\pi, \quad 0 \leq I_2 \leq 1 \end{aligned}$$

- 1.6 Define the chain of required transformations to place the primitive surface in the correct places, positions and with correct dimensions within the Brep described previously. Write your answers in this way:

$$M = \text{rot}(\dots) * \text{trans}(\dots) * \dots * \text{scale}(\dots) \quad \text{etc.}$$

Solution:

$$\begin{aligned} M_face_17 &= \text{trans}(A/2, E, (C-D)) * \text{rot}(Z, 180) * \text{scale}(1, 1, (C-D)) \\ M_face_15 &= \text{trans}(H, I, G) * \text{rot}(Z, 180) * \text{scale}(1, 1, (F-G)) \\ M_face_16 &= \text{trans}(H, I, G) * \text{scale}(1, 1, (F-G)) \end{aligned}$$

5.5.1 Programming in MATLAB

(See Appendix, section 7.3 to find the MATLAB code exposed here)

Write the necessary code to computationally produce the B-rep of the object in point (1).

Functions to write (if you feel that you may need additional functions, propose and implement them):

main.m

Initializes the variables, calls the sub-ordinate routines, creates the curved surfaces, positions the surfaces correctly, draws the solid with the curved surfaces in the right places, etc. The curved surfaces may be plotted with the instruction *mesh()*. YOU DO NOT HAVE TO DRAW PARAMETRIC SURFACES.

[solid, dims_of_loops] = gen_solid()

Creates a solid whose loops are packed in **solid** and whose dimensions are contained in **dims_of_loops**. Ignore here curved FACES or EDGES. For the purposes of this point, curves entities may be assumed as straight lines or plane surfaces.

[PX, PY, PZ]=gen_surf()

Generate the basic surface in unit space 1x1x1.

[P1X, P1Y, P1Z]=transf_1(PX, PY, PZ)

Transform the basic surface defined in 1x1x1 to the required size and position. You will call it as *configuration 1*. Use the transformations that you proposed in point 1.6.

[P2X, P2Y, P2Z]=transf_2(PX, PY, PZ)

Transform the basic surface defined in 1x1x1 to the required size and position. You will call it as *configuration 2*. Use the transformations that you proposed in point 1.6.

[P3X, P3Y, P3Z]=transf_3(PX, PY, PZ)

Transform the basic surface defined in 1x1x1 to the required size and position. You will call it as *configuration 3*. Use the transformations that you proposed in point 1.6.

If you consider it necessary, define as many functions **transf_i (PX, PY, PZ)** as you wish.

draw_solid(solid, dims_of_loops)

Draws the solid whose loops are packed in **solid**, and whose loop dimensions are contained in **dims_of_loops**.

5.6 Cell decomposition and spatial occupancy enumeration

The spatial enumeration is based on the decomposition of the form in simple constituent elements. Such basic elements can be denominated in general *cells*, which are parallelepipeds in 2D and 3D. The capacity of such scheme is based on the clustering of these cells for representation of complex objects, such clustering being usually intensive in its amount of data. Given that an abstraction of the figure is made by representing only up to certain level of detail, the representations improves in quality as the level of detail to represent is smaller. Likewise the quantity of required data grows, being this growth of polynomial order ($O((1/d)^2)$ or $O((1/d)^3)$, where d is the level of detail represented). When the constituent cells are identical in size and semantic capacity, there are exhaustive enumeration schemes, with Pixels, and Voxels (volumetric pixels). When the shape and semantic capacity of the cells agree with the locality of the shape in which the cell is placed, certain schemes are used, such as *Quadrees* / *Octrees*. Given the low level of complexity of the exhaustive schemes, this section will focus on the Quadrees. The Octrees are a natural extension in 3D of this principle.

5.6.1 Quadrees

The quadtree representation Q (Figure 116) of a Lump B in 2D (possibly disconnected) is executed through the following steps:

- 1 A dimension D is set, it establishes a measure of the universe that will be possible to represent. It will be representable everything that, inside the plane $R \times R$ remains inside the *minmax* $[(-D,-D),(D,D)]$. Every part of B that falls outside this square, will disappear from the representation.
- 2 For the universe, $[(-D,-D),(D,D)]$ is evaluated if, B occupies it completely, is totally disjointed from it or occupies it partially. If B occupies the universe totally, Q is marked as "full" or "black". If B is outside the universe Q is marked as "empty" or "white". In these two cases the representation is concluded. But, if B occupies the universe partially ("partial" or "gray"), the following steps are to be taken.
- 3 The universe is divided in 4 quadrants $Q_0 = [(-D,-D),(0,0)]$, $Q_1 = [(0,-D),(D,0)]$, $Q_2 = [(0,0),(D,D)]$, $Q_3 = [(-D,0),(0,D)]$. They are enumerated in CCW direction.
- 4 For each Q_i (Q_0, Q_1, Q_2, Q_3) their dimensions are evaluated. In case that the level of detail that a cell represents is smaller than the minimum resolution of the scheme, it is approximated by "full" or "empty". If the size is still inside the resolution of the scheme, then step (2) is redone for each Q_i .

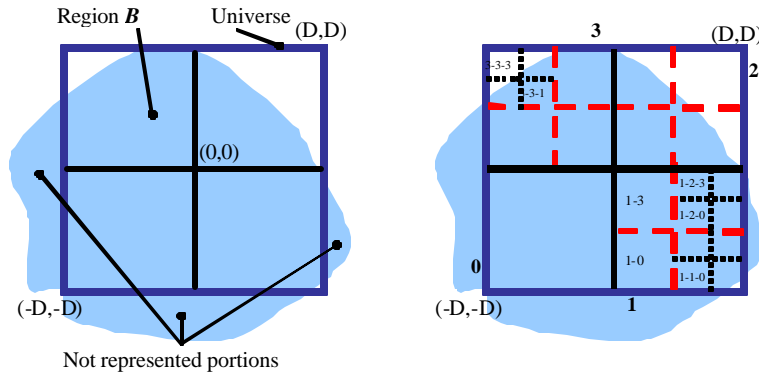


Figure 116. Quadtree Spatial Representation

The conclusions of steps 1 ~ 4 are consigned in a “tree”, data structure, as seen in (Figure 117). The quadtree Q has as many levels as wanted. However, its depth n ($n = 0,1,\dots$) is constrained by setting a maximum resolution d . Given that $d \gg D/2^n$ then $n \gg \log_2(D/d)$.

The analysis is made by levels in the following way:

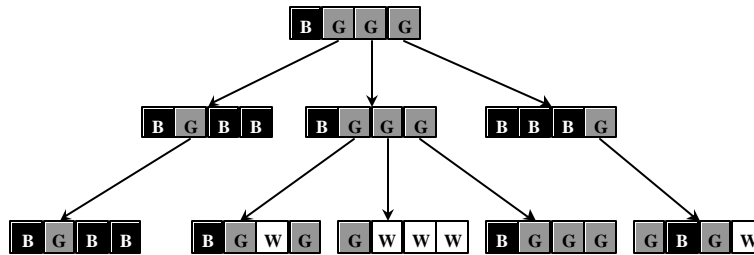


Figure 117. Tree structure representation for the quadtree structure.

5.6.1.1 Observations

- Each cell of the Quadtree can be named according to its position inside the quadrants and subquadrants (Figure 116), the cell 1-2-3 is the one that is in a third subdivision level. In the first division it was classified in the quadrant 1, in the second division it was in the quadrant 2 and in the third division in the quadrant 3.
- The length of the “word” that denotes the position of a cell also indicates its size. In the case of cell “3-1-1” (word length = 3) its size will be $D/4 = D/(2^{3-1})$. Cell “1-0” (length = 2) will have size $D/2 = D/(2^{2-1})$, etc.
- A sequence of geometric transformations M and M^{-1} applied on Q will not result in Q . That is $Q^{-1} M^{-1} \cdot M \cdot Q$ (the reader is hereby encouraged to determine the cause).

- iv. The Quadtree / Octree representation does not show changes below certain resolution. Therefore, translations of arbitrary distances are not realizable in this scheme. The recognizable distance d of any translation applied to Q must be approximated this way:

$$d \approx D + \frac{D}{2} + \frac{D}{4} + \frac{D}{8} + \dots + \frac{D}{2^n} + \dots$$

- v. For a 3-dimensional figure the same previous analysis is made, but dividing the universe in octants.
vi. When the desired level is reached, the programmer must convert gray boxes into white or black.

5.7 EXERCISES – GEOMETRIC MODELING -

5.7.1 CSG and B-REP I

CSG AND BOUNDARY REPRESENTATION. TRANSFORMATIONS

OBJECTIVE

To practice boundary representation and CGS (Constructive Solid Geometry) concepts .

PROCEDURE

For the body in Figure 118, write the topologic structure using boundary representation and CGS construction. Follow the figure notation.

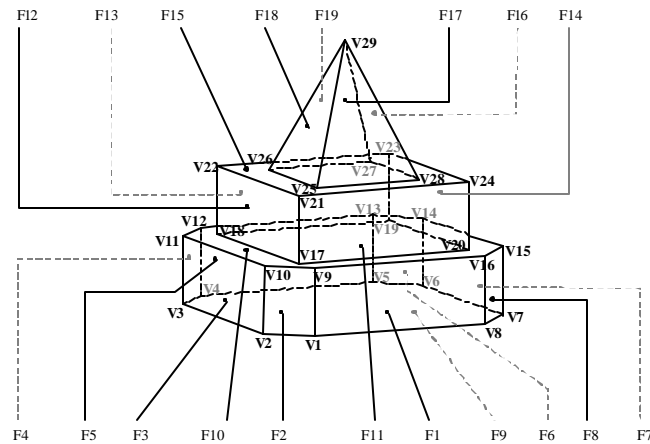


Figure 118. Exercise 5.7.1

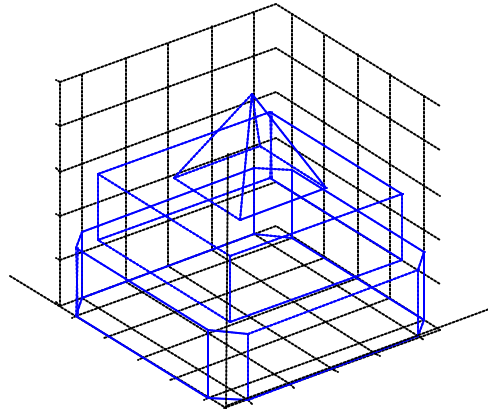


Figure 119. Coordinate plane

1. Assign numerical values to all the vertices of Figure 118 using the coordinate system of Figure 119 .
2. Write the topology structure using Boundary Representation. Follow the figure notation of Figure 118.
3. Draw the CSG tree for the body of of Figure 118. Use the following primitives: *block*(*DX*, *DY*, *DZ*), *pyramid* (*DX*, *DY*, *DZ*) and *wedge*(*DX*, *DY*, *DZ*) whose local coordinate frames are as per Figure 120.

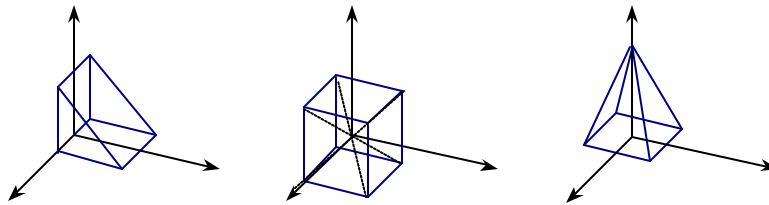


Figure 120. Local Coordinate Frames forWedge, Block and Pyramid.

4. Make a program to build the Brep the solid model of Figure 118. Required functions are:
 - 4.1. **exercise_5_7_1.m**. Main program, which initializes variables and calls sub-functions to build the solid.
 - 4.2. **[solid, dims_of_loops]=gen_solid_001()**. This function creates the data for the solid. The output variables are **solid** (an array containing the loops for all faces) and **dims_of_loops** (an array containing the number of vertices in each loop).
 - 4.3. **draw_solid(solid, dims_of_loops)**. This function draws the solid whose loops are packed in **solid**, and whose loop dimensions are contained in **dims_of_loops** .

5.7.2 CSG and B-REP II.

CSG AND BOUNDARY REPRESENTATION

OBJECTIVE

To practice boundary representation and CGS (Constructive Solid Geometry) concepts .

PROCEDURE

For the body in Figure 121, write the topologic structure using boundary representation and CGS construction. Follow the figure notation.

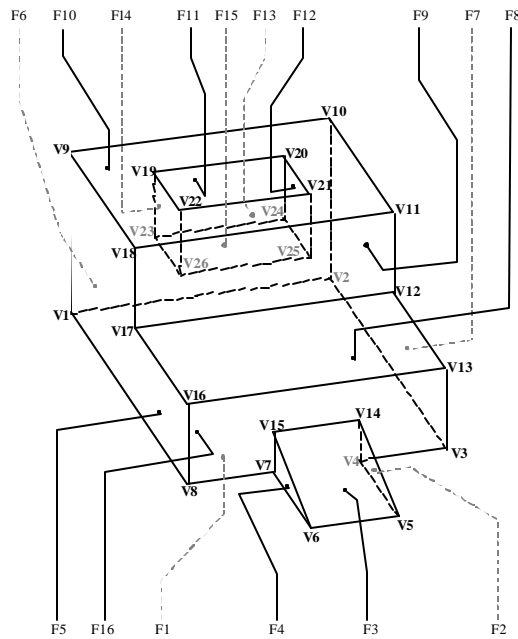


Figure 121. Exercise 5.7.2.

1. Assign numerical values to all the vertices of Figure 121, take the coordinate plane of Figure 122 as reference.

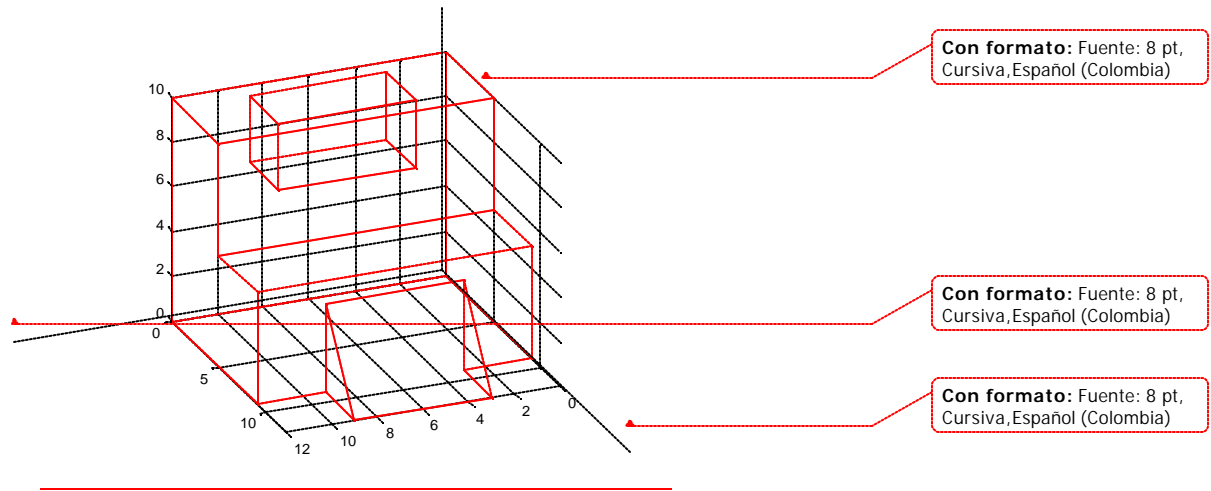


Figure 122. Coordinate plane

2. Write the topologic structure using boundary representation. Follow the figure notation Figure 121.
3. Implement the CSG construction for the body Figure 121. Use the following primitives: *BLOCK*(*DX*, *DY*, *DZ*) (created with mass center on the origin (0,0,0)) and *WEDGE*(*DX*, *DY*, *DZ*) (Figure 123). In order to achieve a correct construction of the solid, it is necessary to use certain transformations on every primitive to realize the Boolean operations.

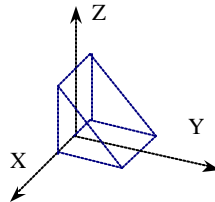


Figure 123. Wedge

4. Make a program to build the solid model of Figure 121; The required functions are:
 - 4.1. **Exercise_5_7_2.m**. Main program, initialize variables and call sub-functions to build the solid.
 - 4.2. **[solid, loop_dims]=gen_solid_003()**. This function creates the solid matrix. The output variables are **solid** that is an array containing the solid and **loop_dims** that is an array containing the number of vertices on each face.
 - 4.3. **draw_solid(solid, loop_dims)**. This function draws the solid whose loops are packed in **solid**, and whose loop dimensions are contained in **loop_dims**.

5.7.3 CSG and B-REP II.

CSG AND BOUNDARY REPRESENTATION, TRANSFORMATIONS

OBJECTIVE

To practice boundary representation and CGS (Constructive Solid Geometry) concepts .

PROCEDURE

For the body in Figure 124, write the topologic structure using boundary representation and CGS construction. Follow the figure notation.

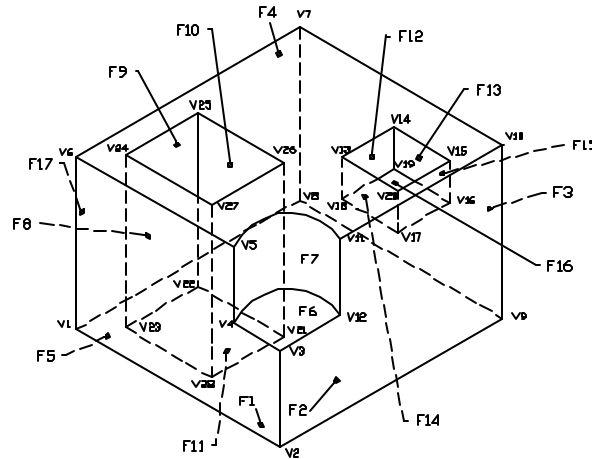


Figure 124. Exercise 5.7.3

1. Assign numerical values to all the vertices of Figure 124, take the coordinate plane of Figure 125 as reference.

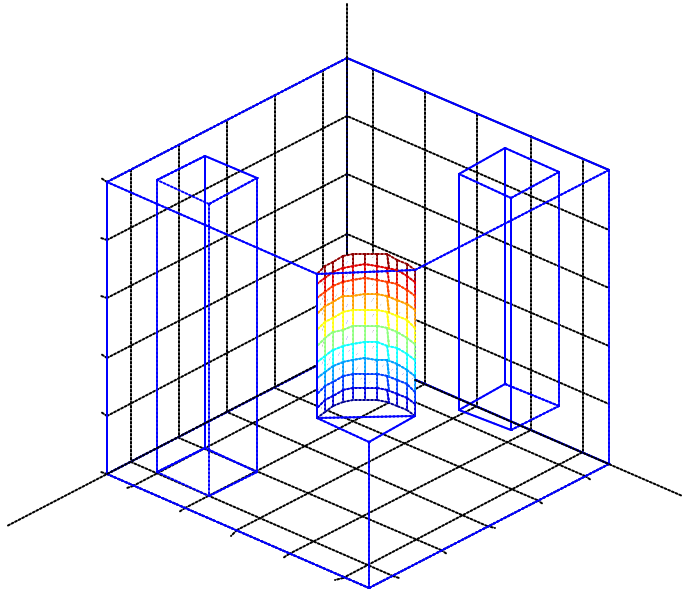


Figure 125. Coordinate plane

2. Write the topologic structure using boundary representation. Follow the figure notation Figure 124.
3. Implement the CSG construction for the body of Figure 124. Use the following primitives: *BLOCK*(*DX*, *DY*, *DZ*) and *CYLINDER*(*R*, *h*) (created with mass center on the origin (0,0,0)). In order to achieve a correct construction of the solid, it is necessary to use certain transformations on every primitive to realize the Boolean operations.
4. Make a program to build the solid model Figure 124. Required functions are:
 - 4.1. **Exercise_5_7_3.m**. Main program, initialize variables and call sub-functions to build the solid.
 - 4.2. **[solid, loop_dims]=gen_solid_002()**. This function creates the solid matrix. The input variables are **solid** that is an array containing the solid and **loop_dims** that is an array containing the number of vertices in each face.
 - 4.3. **draw_solid(solid, loop_dims)**. This function draws the solid whose loops are packed in **solid** and whose loop dimensions are contained in **loop_dims**.
 - 4.4. **[BZRX, BZRY, BZRZ]=gen_mesh()**. This function generates a bezier surface to show the curved part of the solid. The original surface has as base the XY plane in a space of 1 x 1 x 0.5 (Figure 126).
 - 4.5. Realize the all functions to make the indispensable transformations in order to place the surface on the final position. (Rotation, translation and scale).

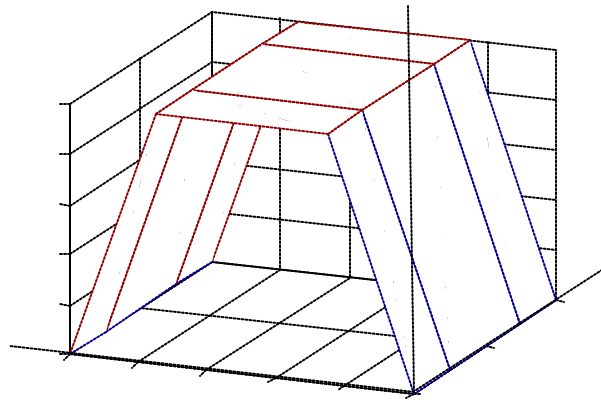


Figure 126. Control Surface

5.7.4 2D Spatial Decomposition.

SPATIAL DECOMPOSITION

OBJECTIVE

To practice spatial decomposition concepts.

PROCEDURE

Make the spatial decomposition of the following figures (Figure 127) up to the fourth level, taking into account that the first quadrant is defined by the *minmax* of the figure:

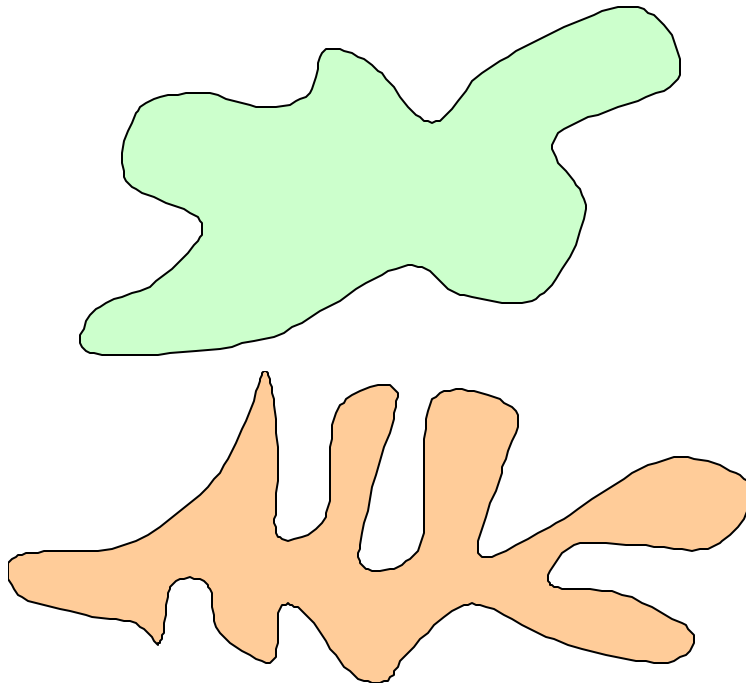


Figure 127. Exercise 4.4

6. FINAL EXERCISES

6.1 EXERCISE I

OBJECTIVE:

To practice with the combination of rigid and non-rigid transformations in order to gain agility with the manipulation of matrices and objects.

PROCEDURE

The figure shows a five-sided object, all of its faces are planar except for the one that involves vertices {E, F, C, D}. This face is **not** planar.

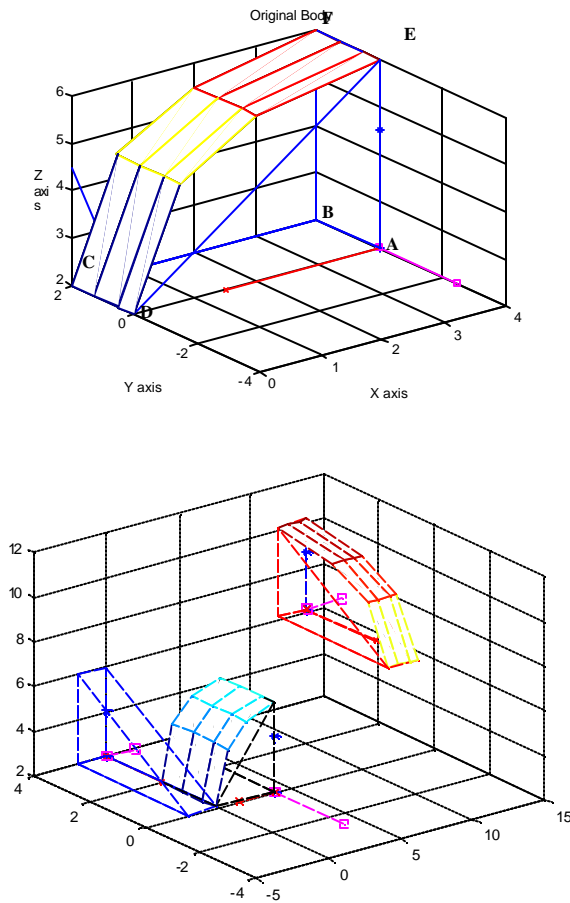
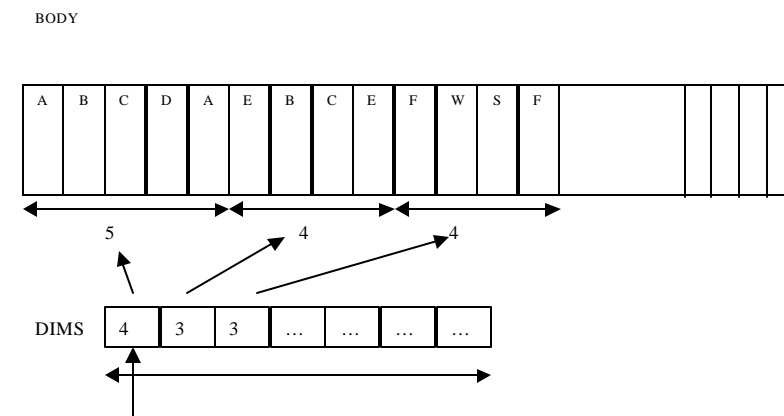


Figure 128.Final Exercise #1

1. Assign coordinates to the vertices A, B, C, D and E.
2. Assign coordinates to the face of vertices { E, F, C, D } in order to be drawn with the instruction *mesh*. The mesh should contain a minimum of 4x4 vertices. Name the variables you use as **MESHX**, **MESHY**, **MESHZ**.
3. Write the routine or code named **draw_solid(body, dims)** that draws all the *loops* or faces of the body. The arguments for this function should be:
 - **BODY**: A matrix of size $(4 \times M)$ that contains the vertices of the body packed by faces (i.e. ABCDA, EFBAE, etc.)
 - **DIMS**: A matrix of size $(1 \times N)$ that says how many *real* vertices has each face. See Figure 129.



Con formato: Inglés (Estados Unidos)

Figure 129. Illustration of the arguments for **draw_solid(body, dims)**

Where the elements of the array DIMS do not take into account the initial vertex repeated at the end of the vertex list of a face, only real vertices per face are considered.

4. Formulate yourself a problem in which the solid given in (1) is to be transformed to another position with at least one rotation and one translation. Draw a complete specification of the problem that you have planned.
5. Write a detailed chart with the necessary transformations in order to achieve the proposed objective. Apply the transformations only to an auxiliary coordinate system attached to the body in the initial position. Draw the evolution of this reference object.
6. Write a function named: **[NEW_X, NEW_Y, NEW_Z] = transform_mesh(MESHX, MESHY, MESHZ, M)** with the necessary instructions to apply any transformation **M** to a mesh contained in **MESHX**, **MESHY**, **MESHZ** and that will produce as result another mesh contained in the variables **NEW_X**, **NEW_Y**, **NEW_Z**.
7. The initial and the final body including their respective non-planar faces and auxiliary coordinate systems must appear in a single fully labeled MATLAB figure window.

6.2 EXERCISE II.

OBJECTIVE:

To practice with the combination of rigid and non-rigid transformations in order to gain agility with the manipulation of matrices and objects.

To review the topologic description methods learned for analytically describing and building computational models of objects.

PROCEDURE

The body shown in Figure 130 has the following characteristics.

- A hollow space that involves the vertices {v5,v6,v7,v8,v9,v10,v11,v12}.
- A ramp that involves the vertices {v13,v14,v15,v16,v17,v18}.
- A rounded edge resembling a circular prism.

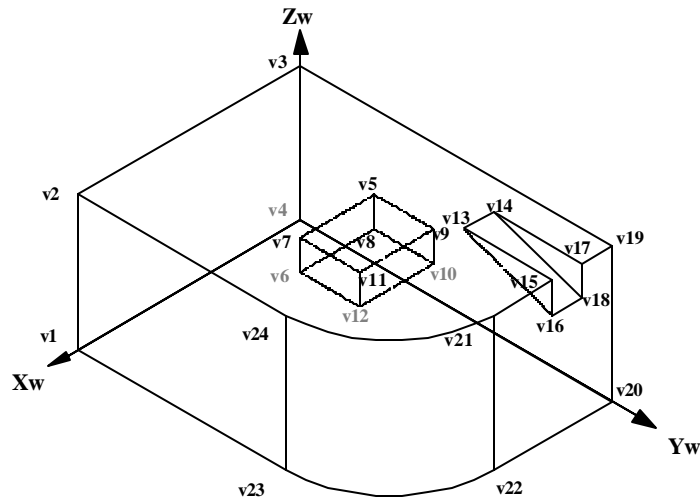


Figure 130. Final Exercise #2

- Complete the nomenclature for the body according to your requirements for making a boundary representation (B-Rep). Label in Figure 130 the faces, the vertices, etc, as needed.
- Give coordinates to the vertices. (the arcs v23-v22 y v21-v24 are a quarter of a circle).
- Label in Figure 130 the **SHELLs**, **LUMP**s, etc. Write the B_rep of the body, and in this representation, clearly mark those *topologies* which their *geometry carriers* are not straight or planar. Say which geometry carriers should they possess (there is no need to give the mathematical equations for these geometries).

4. In the coordinated space shown in Figure 131, define an approximated surface as you consider would fit to the rounded edge. You must define the mesh contained in a coordinated space given by (1.0 , 1.0, 1.0), and positioned as shown in the figure.

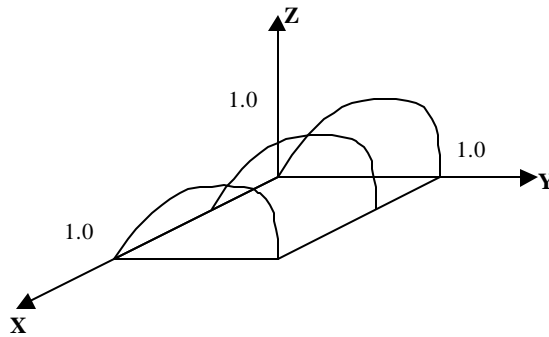


Figure 131. Coordinated space for defining a mesh

MESHX =

MESHY =

MESHZ =

5. Define the transformation sequence that places the mesh in the position and size needed to fit the body in Figure 130.

The coordinates in (2) should be such that the identification of the transformations (you propose) are as simple as possible. You must make suppositions when creating the transformations, explain them.

6. Attach an auxiliary reference frame X_f, Y_f, Z_f, O_f to the mesh in its final position. Draw the evolution of this reference object produced by the transformation sequence you developed in (5), from its initial position to X_f, Y_f, Z_f, O_f . Is this a rigid transformation sequence?. Prove it and give a proper argument.

7. Draw a CSG tree that produces the body shown in Figure 130. You may use the following primitives:

- Box(D_x, D_y, D_z), centered at the origin
- Wedge(D_x, D_y, D_z), placed on the planes XY, XZ y YZ
- Cylinder (Radius , Height) centered at the origin

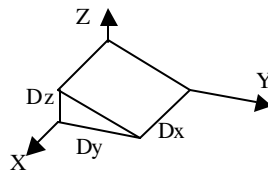


Figure 132. Wedge(D_x, D_y, D_z)

Boolean operations

- Intersection
- Union
- Difference

Transformations: The ones you may need.

Use dimensions consistent with the given coordinates in (2) to define the primitives. You do not need to exactly define the transformations, but you do need to clearly its position in the CSG tree, its type and function.

8. Draw the complete body in a MATLAB figure window using the functions *draw_solid(body , dims)*, *transform_mesh(MESHX, MESHY, MESHZ , M)*, *plt_axes()*, etc. Develop before as well as any other routine that you may need. Additionally draw the in another figure window, the evolution of the mesh *MESHX, MESHY, MESHZ* with its respective auxiliary reference frame.

6.3 EXERCISE III.

In this exercise, it is required a surface interpolation of a torus in two different positions and with closure of the surfaces in different directions.

OBJECTIVES

- Create the control point set or control polyhedron of a torus based on the parameters shown in Figure 133 (Control points generated in exercise 4.3.5 can be used).
- Create a Spline surface interpolation from the previous control set.
- Perform a geometrical transformation on the set of control points.
- Create a new interpolated Spline surface from the transformed point set of (3).

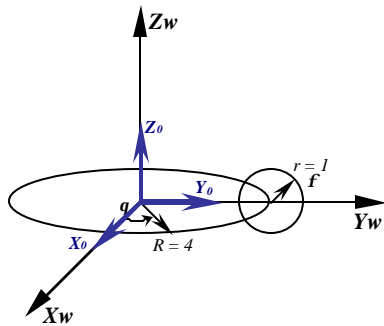


Figure 133. Main parameters of the initial position of the torus

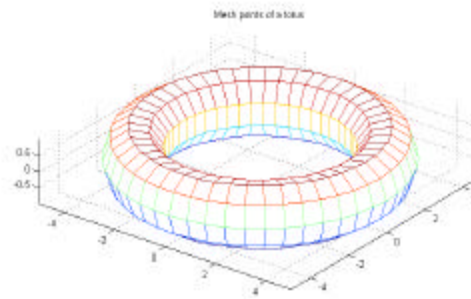


Figure 134. MESH of the control polyhedron of a torus

PROCEDURE

- Write a MATLAB function `[PX, PY, PZ] = gen_torus(R , r)` which generates the control points of a torus in *MESH* format, given a primary *R* and secondary *r* radius.
- Write in MATLAB the necessary functions to generate, from a set *MxN* of control points, a Spline surface with the following input arguments:

`draw_spline(PX, PY, PZ, k, l, close_row, open_col)`

Where:

PX, PY, PZ: Are *MESH* format matrices of the coordinates of the control points.
N: Columns of control points ($N = n+1$).
M: Rows of control points ($M = m+1$).
k: Size of the stage in the *row* direction.
l: Size of the stage in the *column* direction
close_row (0/1): Flag for closure in *row* direction of the control points
close_col (0/1): Flag for closure in *column* direction of the control points.

Examine the scheme of Figure 77 in order to use the notation specified there in your code.

Other functions as specified below must be implemented in order to achieve the objectives.

2.1. **[CPX,CPY,CPZ] = complete_control(PX, PY, PZ, k, l, close_row, open_col)**

```
{pre:   PX, PY, PZ:   matrices MxN of real numbers.
        N:           Columns of control points (N = n+1).
        M:           Rows of control points (M = m+1).
        k:           Size of the stage in the row direction.
        l:           Size of the stage in the column direction
        close_row (0/1): Flag for closure in row direction of the control points
        close_col (0/1): Flag for closure in column direction of the control points.
}
{post:
    (close_row ^ (CPX,CPY,CPZ are PX, PY, PZ extended in the row direction)) OR
    (close_col ^ (CPX,CPY,CPZ are PX, PY, PZ extended in the column direction))
}
```

This routine completes *PX, PY, PZ* in the direction of *rows* and/or *columns* in order to satisfy the closure of the surface in either direction

2.2. **M = spline_matrix(k)**

```
{pre:   k = 2, 3, 4}
{post:   M = Coefficient matrix for Spline interpolation Mk (k = 2,3,4)}
```

2.3. **U = calc_U(u , k)**

```
{pre:   u ∈ [0,1], k ∈ {2,3,4}
{post:   U = [uk-1, uk-2, ..., u1, 1 ] }
```

2.4. **[QX, QY, QZ] = local_patch(PX,PY,PZ,du,dw,Mu,Mw)**

```
{pre:   QX,QY,QZ are k x l matrices of real numbers.
        du ∈ [0,1]
        dw ∈ [0,1]
        Mu = Mk: Coefficient matrix of the patch. (k x k).
        Mw = Ml: Coefficient matrix of the patch (l x l)
}
{post:   QX,QY,QZ contain
        QX(u,w) = Uk · Mk · PX · MlT · WlT ,
        QY(u,w) = Uk · Mk · PY · MlT · WlT ,
        QZ(u,w) = Uk · Mk · PZ · MlT · WlT
        for (u=0, du, 2du, 3du,...,1.0) x (w=0, dw, 2dw, 3dw,...,1.0)
}
```

3. In a second figure window, generate the Spline interpolation for the points of the torus. The surface must be opened in q direction and closed in f direction. Take two points per stage in q sense and four points per stage in f sense.
4. In Figure 135, associate and draw an auxiliar reference frame to the torus $S_0 = [X_0, Y_0, Z_0, O_0]$. Draw in the figure the auxiliar reference frame $S_f = [X_f, Y_f, Z_f, O_f]$. Write here the two reference frames in numeric form:

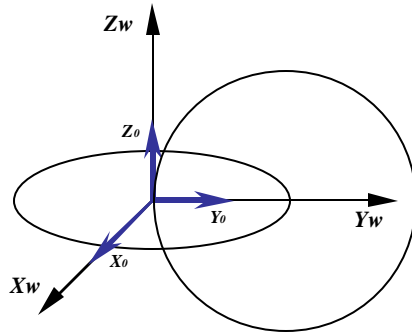
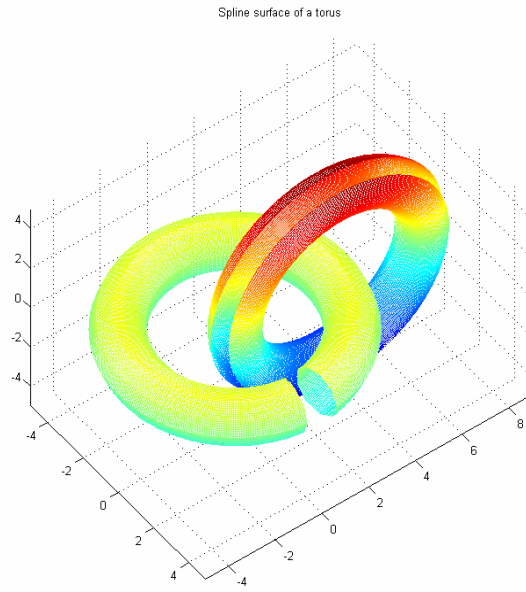


Figure 135. Principal axes of torus in initial and final position

$$S_0 =$$

$$S_f =$$

5. Calculate in numeric form the matrix Mt that transforms S_0 into S_f . Write clearly which equations were used and the result. A table of intermediate transformation steps is NOT needed.
6. Write in MATLAB the following function:
 $[QX, QY, QZ] = transform_mesh(PX, PY, PZ, M)$
 {pre: PX, PY, PZ are matrices of real numbers that form points in E^3 .
 M : Transformation matrix in homogeneous coordinates.
 }
 {post: QX, QY, QZ are new points, previously PX, PY, PZ transformed by M .
 }
7. Write in a script MAIN.m the instructions and the necessary function calls in order to obtain the Spline surface interpolations of the two positions of the torus in the second figure window. The second position must correspond to a torus closed in the direction of q and opened in the direction of f . For the second torus take two points per stage in the f direction and four points per stage in the q . Your work done must permit you obtain this second torus just by changing the input arguments to the function `draw_spline(PX, PY, PZ, k, l, close_row, open_col)`.



*Figure 136. Spline Surface of an original horizontal torus and a transformed vertical torus.
Different closure directions are shown.*

7. APPENDIX

7.1 CHAPTER 3 –Solutions for Proposed Exercises on Geometric Transformations -

7.1.1 Example. Rigid transformations (section 3.2.6 . Example. Rigid transformations.) - MATLAB code

The following are the MATLAB code blocks for the example in section 3.2.6 (Chapter 3 – Geometric Transformations).

```
%===== MATLAB MAIN SCRIPT =====
%   Example section 3.2.6
%
%   OBJECTIVE: To create a boundary representation of a solid
%               and perform rigid transformations on it in order to
%               take from an initial position and orientation to a final
%               sate.
%
%   FUNCTIONS USED:   << rotation_matrix.m, translation_matrix.m, p_plot.m,
%                   transformation.m, plt_axes.m >>
%
%=====
%Screen and memory cleanup
clear
close all
clc
%-----%
%Creation of the solid body by defining vertices and loops (faces)
%Vertices
pa = [8 5 0 1]';
pb = [8 8 0 1]';
pc = [5 5 2 1]';
pd = [5 8 2 1]';
pe = [5 5 0 1]';
pf = [5 8 0 1]';
%Loops
L1 = [pa pe pf pb pa];
L2 = [pb pd pf pb];
L3 = [pa pc pe pa];
L4 = [pc pd pf pe pc];
L5 = [pa pc pd pb pa];
%-----%
%Building the auxiliary axes at the
%initial position
x =[0 -1 0 0]';
y =[1 0 0 0]';
z =[0 0 1 0]';
o =[5 8 0 1]';
So = [x,y,z,o];
d = 2;
plt_axes(So,d);
%-----%
```

```
% Plotting the initial figure
hold on
grid on
axis([0 10 0 10 0 10])
view(132,28)
p_plot(L1,L2,L3,L4,L5,'k')
%Graph legends definition
title('Rigid transformation of a Body');
xlabel('X axis')
ylabel('Y axis')
zlabel('Z axis')

%-----%
%-----%
%Building the Transformation matrices
[trans1] = translation_matrix(-5,-8,0);
[rot] = rotation_matrix('y',90);
[trans2] = translation_matrix(0,3,3);
%-----%
%Call of the function responsible of transforming the solid's loops
% and that gives the resulting rotation parameters after all transformation have been
% executed (axis and angle).

[nL1,nL2,nL3,nL4,nL5,V,D] = transformation(trans1,rot,trans2,L1,L2,L3,L4,L5);
%-----%
%Transforming the auxiliary axes
S1 = trans1*So;
S2 = rot*S1;
S3 = trans2*S2;
%-----%
%-----%

%Plotting the transformed body
p_plot(nL1,nL2,nL3,nL4,nL5,'b')
%-----%
%Plotting the transformed auxiliary axes
% in each transformation stage.
plt_axes(S1,d);
plt_axes(S2,d);
plt_axes(S3,d);
%-----%
%The program will inform the resultant angle and axis as the parameters for a possible
% rotation by quaternion. These parameters are obtained in "transform.m" by evaluating the
% eigen value and eigen vector of the resultant transformation matrix.
disp('The resulting angle of rotation for the transformations made is: ')
a = real(D(1,1));
angulo = acos(a)*180/pi
disp('The resulting rotation axis for the transformations made is: ')
rotation_axis = V(1:3,3)
```

7.1.1.1 Auxiliary functions

```
function [ ] = p_plot(L1,L2,L3,L4,L5,d)
%Function that plots the specific body used in this exercise
% the loop (FACES storing the vertices) are given as input
%-----
plot3(L1(1,:),L1(2,:),L1(3,:),d)
plot3(L2(1,:),L2(2,:),L2(3,:),d)
plot3(L3(1,:),L3(2,:),L3(3,:),d)
plot3(L4(1,:),L4(2,:),L4(3,:),d)
plot3(L5(1,:),L5(2,:),L5(3,:),d)
%-----

function [trans] = translation_matrix(dx,dy,dz)
%Function that generates a translation matrix (trans) based on input displacements relating
% to each coordinate dx,dy,dz
%-----
A = eye(4,4);
A(1,4)= dx;
A(2,4)= dy;
A(3,4)= dz;
trans = A;

function [mr] = rotation_matrix(ax,ang)
% This function calculates a matrix of rotation (mr) to be used in a
% geometric transformation procedure, based on
% an angle and an axis of rotation that are input.
%-----
% Input: "ang" Is the rotation angle in degrees.
% "ax" Axis of rotation.
% Output: "Mr" is the rotation matrix 4x4
%-----

ang = ang*pi/180;

if (ax == 'x')
    mr=[1 0 0 0;
        0 cos(ang) -sin(ang) 0;
        0 sin(ang) cos(ang) 0;
        0 0 0 1];

elseif (ax == 'y')
    mr=[cos(ang) 0 sin(ang) 0;
        0 1 0 0;
        [-sin(ang) 0 cos(ang) 0;
        0 0 0 1];

elseif (ax == 'z')
    mr=[cos(ang) -sin(ang) 0 0;
        sin(ang) cos(ang) 0 0;
        0 0 1 0;
        0 0 0 1];

end

function [nL1,nL2,nL3,nL4,nL5,V,D] = transformation(trans1,rot,trans2,L1,L2,L3,L4,L5)
%Function that applies a resulting transformation matrix to each loop (face) of the solid
%The function also returns the eigen value and eigen vector of the resultant transformation
%matrix.
%-----
m = trans2*rot*trans1;
nL1 = m*L1;
nL2 = m*L2;
nL3 = m*L3;
nL4 = m*L4;
nL5 = m*L5;
[V,D]=eig(m);
%-----
```

```
function plt_axes(S,d)
% This function plots an auxiliary coordinate system (X,Y,Z). It is useful to have all the
% information concerning the auxiliary axes packed in one matrix S, in order to perform
% transformations on it.
% Input: S = [x,y,z,o];
%         "x","y","z" are column vectors for the axes
%         (i.e. X = [1 0 0 0]', Y = [0 1 0 0]', Z = [0 0 1 0]' )
%         "o" the homogeneous-coordinate-point origin from
%         which the three axes are going
%         to start. (column-wise)
%         "d" is an optional input, it represents a scaling factor
%         for the axes when it is
%         dimensionally necessary.
%-----
if (nargin == 1)
    dim = 1;
else
    dim = d;
end

x = S(1:3,1)';
y = S(1:3,2)';
z = S(1:3,3)';
o = S(1:3,4)';
x = ( x/norm(x) )*dim;
y = ( y/norm(y) )*dim;
z = ( z/norm(z) )*dim;
ox = [o',(o+x)'];
oy = [o',(o+y)'];
oz = [o',(o+z)'];

plot3(ox(1,:),ox(2,:),ox(3,:), 'r x --')
hold on
plot3(oy(1,:),oy(2,:),oy(3,:), 'm s --')
hold on
plot3(oz(1,:),oz(2,:),oz(3,:), 'b * --')
hold on
```

7.1.2 Rigid Transformations - Demonstrations and proofs

- Given a set of rigid transformations (arbitrarily mixed rotations and translations):

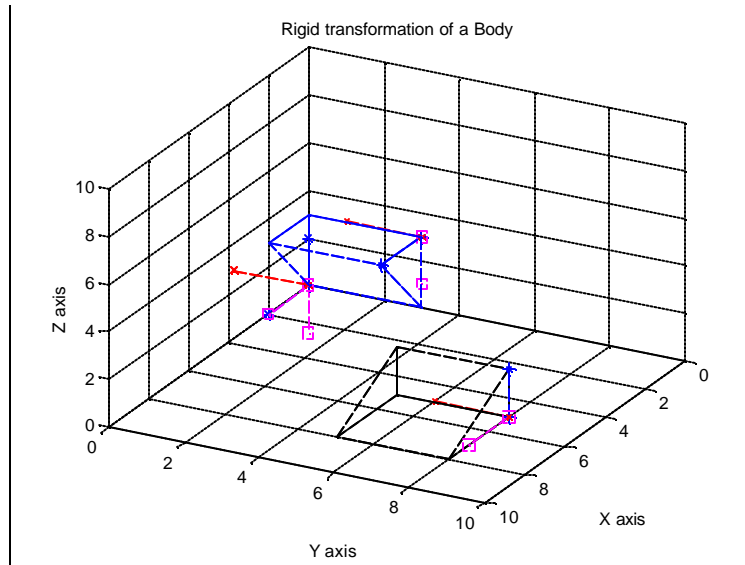


Figure 137. Rigid Transformations.

- Show that the orientation of a body transformed by M_T depends only on the rotations $R_1, R_2, R_3, \dots, R_n$. (translations may be ignored for purposes of orientation).
- Show that $R_5, R_4, R_2, R_1 = R$ is the rotational part of M_T $M_T = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$
- Develop an example implemented in MATLAB, with two translations and two rotations, and verify what you proved in (a) and (b).

Solution

- Vectors, unlike points, store the information regarding to direction. When applying a rotation R and a translation T to a vector v , it can be seen that the orientation of the vector is immune to the translation. It is not important where the vector is in space.

$$M_n \cdot M_R \cdot v_H = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ 0 \end{bmatrix} = \begin{bmatrix} I \cdot (R \cdot v + T \cdot (0)) \\ 0 \end{bmatrix} = \begin{bmatrix} R \cdot v \\ 0 \end{bmatrix}$$

Hence, when a vector is transformed by a sequence of rotations and translations, it will only be affected by the rotations. The translations may be neglected.

- Assume that a chain of transformations is applied:

$$v_F = M_T \cdot v_0 = M_6 \cdot M_5 \cdot M_4 \cdot M_3 \cdot M_2 \cdot M_1 \cdot v_0 =$$

$$= \begin{bmatrix} I & T_6 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_5 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_4 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} I & T_3 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_2 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ 0 \end{bmatrix} = \left[\begin{array}{c|c} R_5 \cdot R_4 \cdot R_2 \cdot R_1 & f(T_6, T_3, R_1, R_2, R_4, R_5) \\ \hline 0 & 1 \end{array} \right] \cdot \begin{bmatrix} v \\ 0 \end{bmatrix}$$

From the first proof above, only the rotational part prevails, and can be decomposed as:

$$\left[\begin{array}{c|c} R_5 \cdot R_4 \cdot R_2 \cdot R_1 & 0 \\ \hline 0 & 1 \end{array} \right] \cdot \begin{bmatrix} v \\ 0 \end{bmatrix} = \begin{bmatrix} R_5 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_4 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_2 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ 0 \end{bmatrix} = M_5 \cdot M_4 \cdot M_2 \cdot M_1 \cdot v_0 = M_r \cdot v_0$$

This confirms that a vector is only affected by rotations and not by translations. If we apply M_T (the total) and M_r (the reduced, rotational) transformations to an object we will get two different final positions but the same orientation.

b) Development in MATLAB

```
%===== MATLAB MAIN SCRIPT =====  
%  
% OBJECTIVE: To proof the invariability of orientation among translations.  
%           To proof de dependence of orientation among rotations.  
%  
% FUNCTIONS USED: << rotation_matrix.m, translation_matrix.m, plt_axes.m >>  
%               See appendix section 7.1.1.1 for details of these functions.  
%=====  
%Screen and memory cleanup  
clear  
close all  
clc  
%-----%  
%Building the auxiliary axes at the  
%initial position (origin)  
x = [1 0 0 0]';  
y = [0 1 0 0]';  
z = [0 0 1 0]';  
o = [0 0 0 1]';  
So = [x y z o];  
subplot(1,2,1)  
plt_axes(So,3)  
%-----%  
%Graph legends definition  
title('Initial coordinate system')  
xlabel('X axis')  
ylabel('Y axis')  
zlabel('Z axis')  
grid on  
axis([-3 3 -3 3 -3 3]);  
axis equal  
%-----%  
subplot(1,2,2)  
hold on  
grid on  
view(120,30)  
%-----%
```



```
%Building the transformation matrices
[rot1] = rotation_matrix('x',30);
[trans1] = translation_matrix(2,3,5);
[rot2] = rotation_matrix('y',60);
[trans2] = translation_matrix(3,1,-4);
%Building the resulting transformation matrices
MT = trans2 * rot2 * trans1 * rot1;
MTR = rot2 * rot1;
%-----%

%Application of translations and rotations
S1t = MT * So;
plt_axes(S1t,3)

%Application of the rotations only
S2r = MTR * So;
plt_axes(S2r,3)
%-----%
%Graph legends definition
xlabel('X axis')
ylabel('Y axis')
zlabel('Z axis')
n = sprintf('The orientation of the vectors hold among translations, but it varies among rotations')
title(n)
axis equal
```

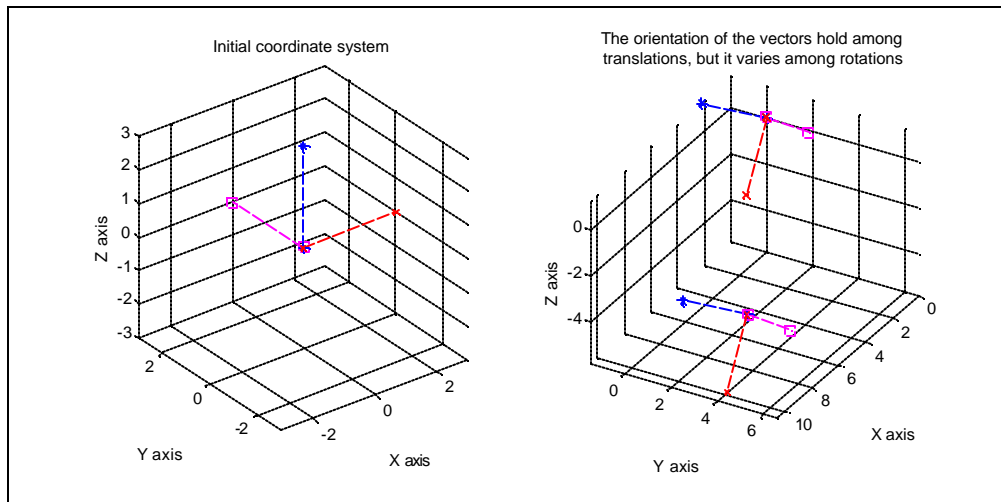


Figure 138. Showing that translations do not affect orientation of the object. Only rotations do so

7.1.3 Additional cases of mirror transformations

7.1.3.1 Mirror about XZ plane

(Inversion of Y)

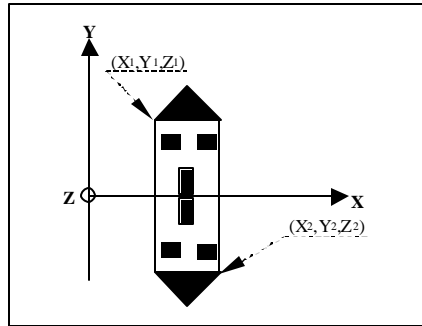


Figure 139. Mirror about the XZ plane.

Let a point with coordinates $P_f(X_1, Y_1, Z_1)$. To take it to another position in space of coordinates (X_2, Y_2, Z_2) such as

$$\begin{pmatrix} X_2 = X_1 \\ Y_2 = -Y_1 \\ Z_2 = Z_1 \end{pmatrix}$$

its mirror image about the XZ plane is calculated. This procedure is illustrated in matrix form in Equation 47. The transformation matrix is shown in Equation 48.

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} = \mathbf{mirror}(\text{plane XZ}) \cdot \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix}$$

Equation 47. Mirror transformation about the XZ plane

$$\mathbf{mirror}(\text{plane XZ}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 48. Mirror transformation matrix about the XZ plane.

7.1.3.2 Mirror about a plane $X = Y$

(Interchanges X with Y)

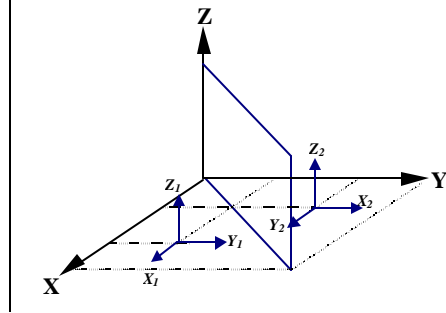


Figure 140. Mirror about a plane $X = Y$

Let a point with coordinates $P_1(X_1, Y_1, Z_1)$. To take it to another position in space of coordinates (X_2, Y_2, Z_2) such as

$$\begin{pmatrix} X_2 = Y_1 \\ Y_2 = X_1 \\ Z_2 = Z_1 \end{pmatrix}$$

its mirror image about the $X = Y$ plane is calculated. This procedure is illustrated in matrix form in Equation 49. The transformation matrix used is shown in Equation 48.

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} = \text{mirror}(\text{plane } X = Y) \cdot \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix}$$

Equation 49. Mirror transformation about the plane $X = Y$.

$$\text{mirror}(\text{plane } X = Y) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 50. Mirror transformation matrix about the $X = Y$ plane

Notice that **mirror**(plane $X=Y$) is not a rigid transformation since it doesn't satisfy $U_1 \times U_2 = U_3$.

7.1.3.2.1 Example. Application of the Householder transformation. to perform a mirror about the plane $X = Y$

For a mirror about the plane $X = Y$, the corresponding normal (unitary) vector and the Householder transformation matrix are illustrated in Figure 141.

$$\begin{aligned}
 n &= \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \end{bmatrix} \\
 H_n &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - 2 \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \end{bmatrix} \\
 H_n &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - 2 \cdot \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
 H_n &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
 H_n &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$

Figure 141. Procedure based on the Householder reflector to obtain the transformation matrix for a mirror about the plane $X = Y$.

As shown, the result obtained is the same reached before for the mirror reflection through plane $X = Y$.

7.2 CHAPTER 4 - Examples of control point generation (MATLAB code)

7.2.1 Example. (sphere)

```
function [Px,Py,Pz] = sphere()
% This function generates the control points for a sphere
%-----
% EAFIT University
% Dr. Prof. Oscar E. Ruiz S.
% Colombia - Medellin
%=====

close all

R      = 10;           % Radius of sphere
n_arcs = 80;           % number of generative arcs
n_pts  = 20;           % Number of points per arc
d_alf  = pi/n_pts;     % Incremental step of parameter
d_th   = 2*pi/n_arcs;  % Incremental step of parameter

Px      = [];
Py      = [];
Pz      = [];

for (th = 0 : d_th : 2*pi)
    i = 1;
    x      = [];
    y      = [];
    z      = [];
    for (alf = 0 : d_alf : pi)
        x(i,1) = R*sin(alf)*cos(th);
        y(i,1) = R*sin(alf)*sin(th);
        z(i,1) = R*cos(alf);
        i = i+1;
    end
    figure(1);hold on;grid on
    plot3(x,y,z,'.')
    view(145,20)
    axis equal

    Px = [Px; x'];
    Py = [Py; y'];
    Pz = [Pz; z'];
end
figure(2)
surf(Px,Py,Pz)
grid on
view(145,20)
axis equal
```

7.2.2 Example. (Cone)

```
function [Px,Py,Pz] = cone_z()
% This function generates the control points for a cone aligned with the z-axis
% Uses a longitudinal proportion expression for generating points on a revolving line
%-----
% EAFIT University
% Dr. Prof. Oscar E. Ruiz S.
% Colombia - Medellin
%=====
close all
clc

H      = 30;           % Height of the cone
R      = 10;           % Radius of the cone
npts   = 20;           % Number of points on the generative line
n_slices = 40;         % Number of generative lines
d_alf  = 1/(npts-1);   % Increment step of longitudinal parameter
Pint   = zeros(npts+1,3); % initialization of array of points in the generative line
rev     = (2*pi);      % amount of revolution
d_th   = rev/(n_slices-1); % Increment step of angular parameter

Px      = [];
Py      = [];
Pz      = [];

for( th = 0:d_th:rev )
    P1 = [R*cos(th),R*sin(th),0];
    Po = [0,0,H];
    i = 1;
    for( alf = 0:d_alf:1 )
        Pint(i,:) = (1-alf)*Po + P1*(alf); % longitudinal proportion expression
        i = i+1;
    end
    figure(1);hold on;grid on
    plot3(Pint(:,1),Pint(:,2),Pint(:,3),'.')
    view(145,20)

    Px = [Px; Pint(:,1)'];
    Py = [Py; Pint(:,2)'];
    Pz = [Pz; Pint(:,3)'];
    Pint = zeros(npts+1,3);
end
```

7.2.3 Example. (Cylinder)

```
function [Px,Py,Pz] = cylinder_z()
% This function generates the control points for a cylinder aligned with the z-axis
%-----
% EAFIT University
% Dr. Prof. Oscar E. Ruiz S.
% Colombia - Medellin
%=====

close all

H      = 10;          % Height of the cylinder
R      = 5;           % Radius of the cylinder
npts   = 20;          % number of points per generative vertical line
nslices = 50;         % number of generative vertical lines
revs   = 2*pi;

dth    = revs/(nslices);
dh     = H/(npts-1);
Px     = [];
Py     = [];
Pz     = [];

for (th = 0:dth:revs)
    i = 1;
    x = [];
    y = [];
    z = [];
    for (h = 0:dh:H)
        x(i) = R*cos(th);
        y(i) = R*sin(th);
        z(i) = h;
        i = i+1;
    end
    figure(1);hold on;grid on, view(145,20),axis equal
    plot3(x,y,z,'.')
    Px = [Px; x];
    Py = [Py; y];
    Pz = [Pz; z];
end
```

7.2.4 Example. (The Möbius Band)

```
function [Px,Py,Pz] = mobius_band()

% This function generates the control points for a mobius band
% Uses a longitudinal proportion expression for generating points on a revolving line
%-----
% EAFIT University
% Dr. Prof. Oscar E. Ruiz S.
% Colombia - Medellín
%=====

close all

R          = 20;
n_slices   = 50;                % INT. Number of generative lines to make along theta.
rev        = (2*pi);            % Amount of revolution to perform about theta.
dth        = rev/n_slices;      % Increment step for the sweep angle theta.
n_pts      = 10;                % Amount of points in the generative line.
dalf       = 1/(n_pts-1);       % Increment step for the point coordinate generaion on
                                % the line.
L          = 10;                % Length of the generative line.

Px         = [];
Py         = [];
Pz         = [];
Pint       = [];
for (th = 0:dth:rev)
    gama    = 1/2*th;            % Control of the number of loops the band will make
    P1      = [(R-L/2*cos(gama))*cos(th),...
               (R-L/2*cos(gama))*sin(th),...
               (-L/2*sin(gama))];
    P2      = [(R+L/2*cos(gama))*cos(th),...
               (R+L/2*cos(gama))*sin(th),...
               (L/2*sin(gama))];

    for (alf = 0:dalf:1)
        Pint = [Pint; (alf)*P2 + (1-alf)*P1 ];
    end

    figure(1)
    plot3(Pint(:,1),Pint(:,2),Pint(:,3),'.')
    hold on; grid on; axis equal
    Px      = [Px; Pint(:,1)'];
    Py      = [Py; Pint(:,2)'];
    Pz      = [Pz; Pint(:,3)'];
    Pint    = [];
end

figure(2)
surf(Px,Py,Pz)
grid on
axis equal
```


7.3 CHAPTER 5 – Example 3 – Implementation in MATLAB–

```
%===== MATLAB MAIN SCRIPT =====
% Appendix Chapter 5 -GEOMETRIC MODELING-
% exercise_CP3_01.m
%
% OBJECTIVE:      To integrate the concepts of transformations,
%                  boundary representations, and parametric curves.
%
% FUNCTIONS USED:  << rotation_matrix.m >>
%                  << translate_matrix.m >>
%                  << draw_solid.m >>
%                  << gen_solid.m >>
%                  << gen_surf.m >>
%                  << scale.m >>
%                  << transf_1.m >>
%                  << transf_2.m >>
%                  << transf_3.m >>
%                  << transmsh.m >>
%-----
% EAFIT University
% Dr. Prof. Oscar E. Ruiz S.
% Colombia - Medellin
%=====
clear all
clc
clf

% Solid
[solid,dims_of_loops]= gen_solid;
draw_solid(solid,dims_of_loops);

%primitive
[PX,PY,PZ] = gen_surf;

% Transformation 1: face 15
[P1X,P1Y,P1Z] = transf_1(PX,PY,PZ);
mesh(P1X,P1Y,P1Z);

% Transformation 2: face 16
[P2X,P2Y,P2Z] = transf_2(PX,PY,PZ);
mesh(P2X,P2Y,P2Z);

% Transformation 3: face 17
[P3X,P3Y,P3Z] = transf_3(PX,PY,PZ);
mesh(P3X,P3Y,P3Z);
```

7.3.1 Auxiliary functions

```
function [ ] = draw_solid(obj,loop_dim)
% This function draws the solid whose loops are packed in obj,
% and whose loop dimensions are contained in loop_dim.
%-----
% Input: obj:    packed loops of the body. Each loop has format
%              v0, v1,...,vn, v0. Each vertex Vi is a (3x1)or (4x1) vector.
%              loop_dim: a (1xM) vector of integer numbers (M=number of loops
%              in "obj". loop_dim(i) is the number of edges of loop i-th.
% Output: none. The routine draws the solid.
%-----
N=size(obj,2);
i=1;
st=0;

while (i<=N)
    if (st==0)
        draw=obj(:,i);
        st=1;
    else if (st==1)
        draw=[draw obj(:,i)];
        if (draw(:,1)==obj(:,i))
            plot3(draw(1,:),draw(2,:),draw(3,:))
            st=0;
            hold on
            grid on
            axis equal
        end
    end
    i=i+1;
end
xlabel('EJE X')
ylabel('EJE Y')
zlabel('EJE Z')
title('SOLID BODY')
```

```
function [solid,dims_of_loops] = gen_solid

% Coordinates of the vertices
v1= [10  0  10 1]';
v2= [0   0  10 1]';
v3= [10  0  0  1]';
v4= [0  10  10 1]';
v5= [0  10  0  1]';
v6= [10 10  0  1]';
v7= [0  20  0  1]';
v8= [10 20  0  1]';
v9= [10 20  3  1]';
v10=[0  20  3  1]';
v11=[3  10  10 1]';
v12=[7  10  10 1]';
v13=[7  10  6  1]';
v14=[3  10  6  1]';
v15=[10 10  10 1]';
v16=[0  0  0  1]';
v17=[8  4  5  1]';
v18=[8  4  8  1]';
v19=[6  4  8  1]';
v20=[6  4  5  1]';

%Packing of loops
%lump shell
```

```
L1=[v1 v15 v12 v11 v4 v2 v1];
L2=[v1 v3 v6 v15 v1];
L5=[v15 v6 v5 v4 v11 v14 v13 v12 v15];
L8=[v2 v4 v5 v16 v2];
L9=[v2 v16 v3 v1 v2];
L11=[v3 v16 v5 v6 v3];
L12=[v14 v13 v14];
L17=[v12 v13 v14 v11 v12];
B1=[L1 L2 L5 L8 L9 L11 L12 L17]; %Big box
%lump2 shell3
L3=[v6 v8 v9 v6];
L4=[v7 v5 v10 v7];
L6=[v6 v9 v10 v5 v6];
L7=[v9 v8 v7 v10 v9];
L10=[v5 v7 v8 v6 v5];
B2=[L3 L4 L6 L7 L10]; %wedge

% Solid
solid=[B1 B2];
dims_of_loops=[6 4 8 4 4 2 4 3 3 4 4 4];
```

```
function [PX,PY,PZ] = gen_surf
% This function generates a half cylinder surface.
% The axis of the cylinder is the Z axis. Its radius is R=1.
%-----
% Input: none
% Output: PX, PY, PZ: the mesh for the half cylinder
%-----
R=.5;
dH=0.1;
dth=15;
PX=[];
PY=[];
PZ=[];
i=1;
for ang=0:dth:180
    ang=ang*pi/180;
    x=R*cos(ang);
    y=R*sin(ang);
    j=1;
    for H=0:dH:1
        z=H;
        PX(i,j)=x;
        PY(i,j)=y;
        PZ(i,j)=z;
        j=j+1;
    end
    i=i+1;
end
```

```
function [M] = scale(Sx,Sy,Sz)

s=[Sx 0 0 0
    0 Sy 0 0
    0 0 Sz 0
    0 0 0 1];
M=s;
```

```

function [M] = rotation_matrix(ang,ax)
% This function produces the rotation matrix
% in homogeneous coordinates. Rotation is about one
% of the World Coordinate System: X, Y or Z.
%-----
% Input: ang= angle of rotation (in degrees)
%       ax = axis of rotation ('X', 'Y', 'Z')
% Output: M: the 4x4 homogeneous rotation matrix.
%-----

ang=ang*pi/180;

if (ax=='X')
    m=[[1 0 0 0]
        [0 (cos(ang)) (-sin(ang)) 0]
        [0 (sin(ang)) (cos(ang)) 0]
        [0 0 0 1]];
elseif (ax=='Y')
    m=[[cos(ang) 0 (sin(ang)) 0]
        [0 1 0 0]
        [(-sin(ang)) 0 (cos(ang)) 0]
        [0 0 0 1]];
elseif (ax=='Z')
    m=[[cos(ang) (-sin(ang)) 0 0]
        [(sin(ang)) (cos(ang)) 0 0]
        [0 0 1 0]
        [0 0 0 1]];
else
    'error: rotation_matrix( ): wrong axis'
    keyboard
end
M=m;

```

```

function [P1X,P1Y,P1Z] = transf_1(PX,PY,PZ)
% This function transforms a mesh PX, Py, PZ to a new
% position. The transformation chain is:
% (1) rot(z,_), (2) scale(), (3) trans()
%-----
% Input: PX, PY, PZ: the input mesh
% Output: P1X, P1Y, P1Z: the transformed mesh
%
%-----
[M1] = rotation_matrix(180,'Z');
[M2] = scale(2,2,3);
[M3] = translate_matrix(7,4,5);
Mt1 = M3*M2*M1;
[P1X,P1Y,P1Z] = transmesh(PX,PY,PZ,Mt1);

```

```

function [P2X,P2Y,P2Z] = transf_2(PX,PY,PZ)
% This function transforms a mesh PX, Py, PZ to a new
% position. The transformation chain is:
% (1) scale(), (2) trans()
%-----
% Input: PX, PY, PZ: the input mesh
% Output: P2X, P2Y, P2Z: the transformed mesh
%
%-----
[M1] = scale(2,2,3);
[M2] = translate_matrix(7,4,5);
Mt2 = M2*M1;
[P2X,P2Y,P2Z] = transmesh(PX,PY,PZ,Mt2);

```

```
function [P3X,P3Y,P3Z] = transf_3(PX,PY,PZ)
% This function transforms a mesh PX, Py, PZ to a new
% position. The transformation chain is:
% (1) rot(z,_), (2) scale(), (3) trans()
%-----
% Input: PX, PY, PZ: the input mesh
% Output: P3X, P3Y, P3Z: the transformed mesh
%-----

[M1]= rotation_matrix(180,'Z');
[M2]= scale(4,4,4);
[M3]= translate_matrix(5,10,6);
Mt3 = M3*M2*M1;
[P3X,P3Y,P3Z]= transmesh(PX,PY,PZ,Mt3);

function [M] = translate_matrix(dx,dy,dz)
% This function produces the translation matrix
% in homogeneous coordinates, for displacement dx, dy, dz.
%-----
% Input: dx,dy,dz= displacement vector
% Output: M: the 4x4 homogeneous translation matrix.
%-----

A = eye(4,4);
A(1,4)= dx;
A(2,4)= dy;
A(3,4)= dz;
M=A;

function [Px,Py,Pz] = transmesh(X,Y,Z,M)
% This function transforms a mesh X, Y, Z by a (4x4).
% transformation M.
%-----
% Input: X, Y, Z: the mesh to transform
% M : the homogeneous transformation matrix
% Output: Px,Py,Pz: The transformed mesh.
%-----

[r,c]=size(X);

Px=[];
Py=[];
Pz=[];

for i=1:r
    for j=1:c
        Pto_i=[X(i,j) Y(i,j) Z(i,j) 1]'; %Initial point
        Pto_f=M*Pto_i; %Final point
        Px(i,j)=Pto_f(1,1);
        Py(i,j)=Pto_f(2,1);
        Pz(i,j)=Pto_f(3,1);
    end
end
```

8. BIBLIOGRAPHY

- [AHO.73] Aho, A., Hopcroft, J., Ullman, J. (1973) *Data Structures and Algorithms*. Addison Wesley Publishing Co.
- [BOT.79] Bottema, O., Roth, B. (1979) *Theoretical Kinematics* North Holland Press, New York.
- [ANS.93] ANSI Y14.5.1M-Draft, 1993, "Mathematical Definition of Dimensioning and Tolerancing Principles", *American Society of Mechanical Engineers*. New York
- [BOL.91] Bolle, M, Rudd, C., Vemuri, 1991, "On Three Dimensional Surface Reconstruction Methods". *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol3,No1, pp. 1-13.
- [BOR.98] Borhese, N., Ferrigno, G., Baroni, G., Pedotti, A., Ferrari, S., Savare, E., 1998, "AutoScan: A Flexible and Portable 3D Scanner". *IEEE Computer Graphics and Applications, Computer Graphics I/O Devices* May/June, pp. 39-41.
- [CAR.95] Carr, K., 1995, " Modeling and verification methods for the inspection of geometric tolerances using point data ", Ph.D. Dissertation, University of Illinois at Urbana - Champaign.
- [CAT.78] Catmull, E., Clark, J., 1978, "Recursively Generated B-Spline Surfaces on Arbitrary Topological meshes". *Computer Aided Design*, Vol 10, No 6, Nov, pp 350-355.
- [DOO.78] Doo, D., Sabin, M., 1978, "Behavior of Recursive Division of Surfaces near Extraordinary Points ". *Computer Aided Design*, Vol 10, No 6, Nov, pp 356-360.
- [DRE.88] Drebin R., Carpenter, L. and Hanrahan, P, 1988 "Volume Rendering", *ACM Computer Graphics*, Vol. 22, Number 4, August.
- [EDE.94] Edelsbrunner, H., 1994, "Three Dimensional Alpha Shapes". *ACM Transactions on Graphics*, Vol 13, No 1, Jan, pp 43-72.
- [EDE.87] Edelsbrunner, H., 1987, "Algorithms in Combinatorial Geometry", Springer Verlag New York.
- [FAR.90] Farin, G. (1990) *Curves and Surfaces for Computer Aided Geometric Design. A Practical Guide* Academic Press. Boston.
- [FOL.90] Foley, J., van Dam, A., Feiner, S., Hughes, J. 1990, *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Co. ISBN 0-201-12110-7.
- [GOL.96] Golub, G., VanLoan, C. (1996) *Matrix Computations*. Johns Hopkins University Press. 1996.
- [GRI.95] Grimm, C., Hughes, J., 1995, "Modeling Surfaces of Arbitrary Topology using Manifolds". *Proceedings, SIGGRAPH - 95 Annual Conference Series*, Los Angeles, August 6-11, n. 29, pp. 359-368.
- [GUO.97] Guo, B., 1997, "Surface Reconstruction: From Points to Splines". *Computer Aided Design*, Vol 29, No 4, pp 269-277.
- [HOP.92] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W., 1992, "Surface Reconstruction from Unorganized Points". *Proceedings, SIGGRAPH -92 Annual Conference Series*, Chicago, July 26-31, n. 26, pp. 71-78.
- [KHA.95] Khan, M., Vance, J., 1995, "A Mesh Reduction Approach to Parametric Surface Polygonization". *Proceedings, ASME Design Engineering Technical Conferences. - Advances in Design Automation*, pp. 41-48.
- [KOL.99] Kolman, Bernard, 1999. *ALGEBRA LINEAL CON APLICACIONES Y MATLAB*. Prentice Hall.
- [JOY.96] Joyanes A., Luis, 1996. *-Fundamentos de Programación, Algoritmos y estructuras de datos*. McGraw Hill
- [KRE.93] Kreyszig, E., 1993, *Advanced Engineering Mathematics*. John Wiley & Sons.
- [KOR.85] Yoram, K. 1985, *Robotics for Engineers*, McGraw-Hill Book Co. ISBN 0-07-035399-9
- [LOR.87] Lorensen, W., Cline, H., 1987, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", *ACM Computer Graphics*, Vol. 21, No. 24, July, pp. 163-169.

- [NEU.97] Neugebauer P., 1997, "Geometrical Cloning of 3D Objects via Simultaneous Registration of Multiple Range Images". *Proceedings, International Conference on Shape Modeling and Applications*, Aizu-Wakamatsu, Japan, March 3-6.
- [MOR.85] Mortenson, M., 1985, *Geometric Modeling*, John Wiley and Sons. New York
- [MOR.99] Mortenson, M., 1999, *Mathematics for Computer Graphics Applications*, Industrial Press, Inc. New York. Second edition.
- [ONE.66] O'Neill, Barnet, 1966, *Elementary Differential Geometry* Academic Press. San Diego
- [PET.98] Petrov, M., Talapov, A., Robertson, T., Lebedev, A., Zhilyaev A., Polonsky, L., 1998, "Optical 3D Digitizers: Bringing Life to the Virtual World". *IEEE Computer Graphics and Applications, Computer Graphics I/O Devices* May/June 1998, pp. 28-37.
- [PRE.85] Preparata, F. and Shamos, M. I., 1985, "Computational Geometry. An Introduction", Springer Verlag. New York.
- [RAN.94] Ranyak, P., 1994, "Application Interface Specification (AIS), Version 2.1". *Consortium for Advanced Manufacturing International (CAM-I)*. Integrity Systems, USA.
- [RUI.95] Ruiz, O., 1995, "Geometric Reasoning in Computer Aided Design, Manufacturing and Process Planning", Ph.D. Dissertation, University of Illinois at Urbana-Champaign.
- [RUI.94] Ruiz, O., Marin, R. and Ferreira, P. (1994) *A Geometric Reasoning Server with Applications to Geometric Constraint Satisfaction and Reconfigurable Feature Extraction*. *Proceedings, 3rd Luso-German Workshop on Graphics and Modeling in Science and Technology*, Coimbra, Portugal.
- [RUI.97b] Ruiz, O., Isaza, D., 1997, "Fitting and Edition of Surfaces to Digitizations. An Application AIS Compatible". *Proceedings, EGRAF-97. I Iberoamerican Symposium on Graphic Expression*, Oct 13-15, 1997 Camaguey, Cuba
- [RUI.98a] Ruiz, O., Henao, C., 1998, "Modelaje Geométrico De Estructura Ósea". *X Congreso Internacional de Ingeniería Gráfica*, June 3-5 1998, Málaga, Spain.
- [RUI.98c] Ruiz, O., Posada, J., 1998, "Computational Geometry In The Preprocessing Of Point Clouds For Surface Modeling". *Proceedings, IDMME-98 (Integrated Design and Manufacturing in Mechanical Engineering)*, May 27-29, Compiègne, France. pp. 613-620
- [RUI.00] Ruiz, O., 2000, "DigitLAB, an Environment and Language for Manipulation of 3D Digitizations ". *Proceedings, IDMME-2000 (Integrated Design and Manufacturing in Mechanical Engineering)*, May 16-19, Montreal, Canada.
- [SAM.95] Samet, H., 1995, "Application of Spatial Data Structures: Computer Graphics, Image Processing and GIS", Addison Wesley Publishing. Co.
- [SZE.93] Szeliski, R., Tonnesen, D., Terzopoulos, D., 1993, "Modeling Surfaces of Arbitrary Topology with Dynamic Particles". *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, pp. 82-87.
- [TRE.97] Trefethen, L., Bau, D., 1997, *Numerical Linear Algebra*, Symposium for Industry and Applied Mathematics, SIAM. ISBN 0-089871-361-7.
- [VAN.92] Vanzandt W., 1992, "Scientific Visualization: One Step in lab Analysis Workflow", *Advanced Imaging*, Feb, pp. 20.
- [YAU.83] Yau, M., Srihari, S.N., 1983, "A hierarchical Data Structure for Multidimensional Digital Images", *Communications of the ACM*, 66, 7(July 1983), pp. 504-515.