NASA Contract Report 181747

Pactruss Support Structure for Precision Segmented Reflectors

John M. Hedgepeth
Astro Aerospace Corporation
Carpinteria, California 93013

Prepared by Astro Aerospace Corporation
for NASA Langley Research Center
under Contract NAS1-17536, Task 9

June 1989

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

SECTION 1

INTRODUCTION

The construction and maintenance of the primary reflectors of future large space telescopes will demand major advances in the technology of space structures and materials. For example, the so-called Large Deployable Reflector (LDR), described in Reference 1, is planned to have an aperture of 20 meters and a wavefront accuracy of about five micrometers. The resulting ratio of allowable structural error to aperture of 10^{-7} is much smaller than present antenna systems and is surpassed only by non-deployable advanced optical instruments such as the Hubble Space Telescope.

Large reflectors such as the 20-meter LDR cannot, of course, be transported to space in their final configuration. In addition, because of the requirement that the surface of the reflector be mirror-like and very stable, the construction by purely deployable techniques is difficult. One such approach, described in Reference 2, and illustrated in Figure 1, utilizes a deployment canister containing interconnected modules of reflector panels and attached stowed truss segments. The canister walks itself around the perimeter of the reflector, deploying the modules sequentially and attaching them to the growing structure. Such an intelligent deployment mechanism would require extensive development.

The alternative to pure deployment is to use assembly in space. A possible approach would be to attach premanufactured reflector panels to a supporting stiff truss structure. While the truss structure could also be erected, the total on-orbit assembly work would be greatly reduced by deploying it. The recent invention of the Pactruss concept, described in Reference 3, with its strongly self-synchronized deployment, makes this combined approach particularly attractive.

The application of the Pactruss deployable structure to the Space Station primary structure is described in References 4 and 5. The concept is seen to provide a realistic alternative to assembly in enabling the construction of large trusses from small stowed volumes. In a similar way, the usefulness of

the concept to precision segmented reflectors can be shown. For high concentration-ratio solar collectors, for example, the Pactruss concept is shown in Reference 6 to provide an efficient precise backup to the reflector panels.

Because of its apparent ease of synchronous deployment, as well as its excellent deployed dimensional stability, the Pactruss concept has been identified as the primary candidate for the deployable truss for LDR (see Reference 1). Some primary issues, however, need to be investigated. One is whether the Pactruss design with its stringent geometric requirements can indeed be applied to a doubly-curved surface such as the parabolic reflector of LDR. Another is how to modify the design so that it can be stowed around, and attached to, a central part of the spacecraft for launch. In addition, quantitative evaluations of the weight and stiffness of the resulting deployed truss structure are needed. The investigations are described herein.

SECTION 2

GEOMETRY OF HYBRID PACTRUSS

Pactruss deployable structures stow in a very compact form. As shown schematically in Figure 2 and as a conceptual model in Figure 3, the structure is composed of verticals connected by surface members (longerons) and core members (diagonals). The verticals move up or down during stowage, the sense alternating in two of the three triangular directions over the planform. Alternating verticals are connected by non-folding longerons. In the third planform direction, adjacent verticals move in the same sense. The longerons in this direction must fold in order to allow the ends to move together. The verticals and the non-folding longerons form sets of parallelograms that become thin when stowed. In the stowed configuration, all members are vertical, with the folding longerons and diagonals occupying spaces provided by finite hinge offsets. The stowage is one of the type called "double fold" because the planform is packaged in both directions.

For a configuration such as the LDR, Figure 4, in which the deployed truss surrounds a center body, double-fold stowage presents a problem--the central opening occupied by the center body also packages during stowage. Thus, the stowed truss must be packaged separately from the center body and mounted to it after deployment. A better approach would be to have the truss stow around the center body. It could then be attached and any utility connections could be made and tested prior to launch.

The desirability of stowage around the center body motivated the creation of the hybrid Pactruss concept shown in Figure 5. The hybrid is made up of single-fold beams separating the planform into areas which are filled with double-fold Pactruss. The directions of the non-folding Pactruss longerons are taken to be parallel to the beams on the radial boundaries. On stowage, the hybrid truss shrinks to a "sleeve" around the center body as shown in Figure 6. The single-fold beams accordion-fold to flat packs and the Pactruss segments occupy the small triangular regions at the corners.

The width of the single-fold beams should be selected so that the central opening fits properly around the center body. If possible, the width also

should be selected to provide good locations for the attachment of reflector panels. A possible arrangement of panels is shown by the heavy lines in Figure 7. Here three corners of each panel are attached to the truss node lying under them. In the single-fold regions, the attachments are located differently than in the double-fold regions.

The truss configuration in Figures 5 through 7 is divided into six congruent sections. It is entitled "Sixpac" and is shown in perspective in Figure 8. Another possible arrangement would be the "Tripac" shown in Figure 9. Note that the beams are considerably wider in this particular version in order to interface with larger panels attached as shown by the heavy dots in the figure. Note that each panel has its own three attachment points instead of having to share them with adjacent panels.

Of course, other arrangements are possible. The two shown serve to demonstrate that pre-attachment of the truss to a center body is feasible.

SECTION 3

GEOMETRIC REQUIREMENTS OF THE PACTRUSS CONCEPT

The basic geometrical requirements for all trusses with slender members is that their centerlines must pass through a common point at the node to which they are attached.

The primary geometrical requirement pertaining to the Pactruss concept is that the distance between connected nodal pairs be maintained throughout deployment. Otherwise, the misfit would generate large axial loads in the members. For the folding longerons and diagonals, this requirement can be easily met by appropriately locating the knee hinge in each member. The non-folding members, on the other hand, must span the correct distance exactly.

The secondary geometrical requirement is that the hinges between members be located and oriented so that the amount of bending and torsion in each strut is small. Optimally, the entire deployment should be strain-free, except for those strains induced by hinge moments (either driving or frictional) or external loads.

Both of these requirements are complicated by the fact that the struts, although slender, have finite cross-sectional dimensions and require space in the package. Therefore, the hinges between the members must be offset from the center of the nodal points of the truss by as much as three member diameters. Preferably, the hinges at member ends should be located on their centerline so that no upsetting moments are generated by axial forces in the members of the deployed truss.

When the Pactruss forms a flat surface, with regularly spaced modules, the design of the Pactruss joints is straightforward. All the non-folding longerons are made the same length, with the same total hinge offset at the two ends, producing a deployed structure with uniform distances between the node points along the directions of the non-folding longerons. The hinge lines are oriented in the horizontal plane, perpendicular to the member centerline. During deployment, the projection of each node in the surface

plane moves directly and proportionally away from the projection of its neighbors. The deployment proceeds smoothly, with the non-folding longerons at a uniformly decreasing slope with respect to the horizontal plane.

For a doubly curved surface, on the other hand, the proper design is not so immediately evident. Careful examination, however, shows that at least one solution exists that satisfies the primary requirement of correct distances between nodes. Furthermore, the solution also yields hinge locations and orientations that satisfy the secondary requirement for the conditions of full deployment and full stowage. Whether the solution also allows strain-free deployment is by no means evident; the evaluation of deployment straining requires detailed analysis.

The design solution is an extension of that for the flat truss. First, the distances between adjacent deployed nodes along non-folding longerons must be a constant value. Second, the sum of the hinge offsets at the ends of each non-folding longeron must be the same. Third, the hinge lines must lie in the horizontal plane and be perpendicular to the longeron centerline. Note that these are the same as the rules used for the flat truss, except that the offset lengths can be different for the two directions in the flat truss--not so with the doubly curved truss.

That the solution yields a deployed and stowed strain-free Pactruss can be seen from the following reasoning:

Let the distance between nodes connected by non-folding longerons be ℓ and the total of the hinge offsets at the two ends be e. Let the upper-surface points of the deployed truss be located at (x_n, y_n, z_n) $n = 1,2,3...,N$ with

$$(x_i - x_j)^2 + (y_i - y_j)^2 = (z_i - z_j)^2 = \ell^2$$

for all i,j pairs connected by non-folding longerons. Assume (without loss of generality) that the origin of the coordinate system is attached to one of the nodes which does not move upward during stowage. Let the fixed node number be m. Then the coordinates of the n^{th} node when stowed can be seen to be

$$\xi_n = x_m + \frac{e}{\ell} (x_n - x_m)$$

$$\eta_n = y_m + \frac{e}{\ell} (y_n - y_m)$$

$$\zeta_n = z_m + \frac{e}{\ell} (z_n - z_m) + z_u$$

where $z_u = \ell - e$, for upwards stowing nodes

$= 0$, otherwise

One way to visualize this stowage is to assume that the non-folding longerons are removed and the hinge lines on either end are merged. The resulting nodal surface is then the same as the deployed one, except that it is shrunken by the factor e/ℓ. Inserting the longerons then raises the upward stowing nodes by the distance $\ell - e$.

Examination of the hinge lines shows that they have the same orientation whether deployed or stowed; the verticals have no rotation. The longerons thus do not bend or twist.

In order that the nodes lie on the desired paraboloidal surface and obey the internodal distance requirement, a computational algorithm must be developed as follows:

Let (x_1, y_1, z_1) and (x_2, y_2, z_2) be previously located points. The location of the next point (x, y, z) must satisfy the following equations:

$$(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = \ell^2$$

$$(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = \ell^2$$

$$z = z_o + \frac{x^2 + y^2}{4F}$$

where F is the focal length of the parabaloid.

The easiest way to solve for x, y, and z is to assume a value for z, solve the first two equations for x and y, and determine a new value for z. Repeating the process a finite number of times (say, 50) guarantees a very accurate answer.

Before determining the Pactruss nodes, the nodes for the beam lying along the x axis are found by using only one of the distance equations in addition to the paraboloid equation, with the assumption that y is the half-width of the beam. Thus for the beam of width b,

$$(x - x_1)^2 + (z - z_1)^2 = \ell^2$$

$$z = z_o + \frac{b^2}{16F} + \frac{x^2}{4F}$$

This is also solved most easily by iteration.

After the points on the upper surface are obtained, those for the lower surface are determined by subtracting the constant depth h from the z coordinates.

In order to determine the points, computer programs were written in the C language. The source code is included in the Appendix for GEN6PAC.C and GEN3PAC.C for generating the Sixpac and Tripac geometries, respectively. Inputs to these programs are:

N, the number of rings
ℓ, the longeron length
h, the truss depth
b, the beam width
F, the focal length

Output is a file, TRUSSNAME.DAT, listing the nodal coordinates.

Also included in the Appendix are source files GEN6STR.C and GEN3STR.C, which read the file TRUSSNAME.DAT and generate a file, TRUSSNAME.DTA, which contains

the nodal coordinates followed by a listing of node endpoints and strut type for each of the members.

An example of an output file for a two-ring Sixpac truss is included in Table 1. Note that there are 72 nodes (36 on each surface), 258 members, and 12 types of members. An additional point is included at the focal point. For this case, $\ell = 2.0$, $h = 2.0$, $b = \sqrt{3}$, $F = 5.0$.

SECTION 4

STRUCTURAL PERFORMANCE

In order to obtain quantitative estimates of the ability of the Pactruss structure to provide precise positioning and support to reflector panels, the five-ring Sixpac truss was analyzed using the MSC/pal 2* finite element analysis. The truss is shown in Figure 8 and has the following dimensions:

$F = 10.0$ m
$\ell = 2.0$ m
$h = 2.0$ m
$b = 1.732$ m

The necessary input file to the MSC/pal 2 program was generated by the computer program SEE2PAL.C included in the appendix, which reads the TRUSSNAME.DTA file previously prepared, and obtains structural property information from the data file TRUSSNAME.PRP. This latter file lists the strut area, modulus, and so forth, for each strut type.

4.1 STATIC PERFORMANCE

Static structural analysis was performed for two cases: One-g loading in the negative z-direction, and an angular acceleration of 0.001 radians/sec^2 around the y-axis. The first case was selected to determine the expected deflections during ground testing. The second case represents a severe case of loading due to steering control while in operational orbit. (Earlier estimates given in Reference 2 reveal that the maximum operational angular acceleration is of the order of 10^{-4} radians/sec^2.) In both cases, the truss is assumed to be mounted at the central opening.

Results of the static analyses, given in Figure 10, pertain to both the bare truss alone and to the truss loaded with reflector panels weighing 10 kg/m^2.

---

*MSC/pal 2 is a trademark of the MacNeal-Schwendler Corporation.

The mass of the structure itself included a joint weight equal to the weight of the tubular struts.

Examination of the deformation shapes showed that most of the deformation comes from straining of the inner rings. Accordingly, a tapered-stiffness design was developed which was much lighter than a uniform stiffness truss of equal overall stiffness. The tapered design was chosen for further study.

The tapered design deflects a maximum of 200 micrometers at the rim due to its own weight. The accuracy requirement that has been placed on the truss is that its attachment nodes should be precise to 100 micrometers without on-orbit adjustment. The fact that the gravity sag is only twice this much gives confidence that a combination of ground testing and analysis will be able to guarantee the required accuracy when the truss is deployed on-orbit.

The structural weight, including joints, of the tapered design is 819 kg. The truss supports panels weighing 3730 kg and is strong enough to carry them in a one-g field. Of more import is the fact that the maximum deflection at the rim caused by the angular acceleration of 1.0 milliradian/sec^2 is only 3.5 micrometers.

The distortion due to operational loading is better described in terms of the rms deviation from a best-fit paraboloidal surface. The source code of the program BESTFIT.C is included in the Appendix. This program reads the files output by MSC/pal 2 and determines the deviation from and the pointing error of the best-fit paraboloid. For the two-meter deep truss, the rms deviation is only 0.17 micrometer; the pointing error is 2×10^{-7} radians or 0.04 arc sec. These errors are well within the budgets of ~3 micrometers and 0.1 arc sec established for the LDR in Reference 1. The fact that the Pactruss is stiff enough to prevent unacceptable deflections due to operational loads means that the control system needed to correct the optical path can be of the low band pass type and interaction between the structure and the control system can be readily handled.

4.2 EFFECTS OF TRUSS DEPTH

Some influences of truss depth on structural performance are shown in Figure 11. The distortions increase rapidly for truss depths less than two meters.

Larger depths reduce the distortion at the cost of increased structural mass. Of more concern is the fact that the longer struts have a lower lateral vibration frequency, which is about 22 Hz for a two-meter depth. Earlier studies of sensitivity of surface accuracy to manufacturing imperfections (Reference 6) indicate that depths between one and three longeron lengths are best. All these results point to the desirability of a truss depth equal to or moderately greater than a longeron length.

4.3 VIBRATION PERFORMANCE

As an additional indication of the stiffness of the Pactruss, the dynamic analysis capability of MSC/pal 2 was used to obtain the natural vibration modes and frequencies for the tapered design. The structure includes the truss and the panels and is mounted at the 12 node points at the central opening. The results are shown in Figure 12 for the first 20 modes. The fundamental frequency is over 10 Hz, a high figure indeed for such a large structure. The modal density is high, with the first five modes having frequencies within 15 percent of each other.

SECTION 5

DEPLOYMENT ANALYSIS

In order to obtain some insight into the behavior of a Pactruss structure with a doubly curved surface, the sector shown darkened in Figure 13 was analyzed with an Astro Aerospace Corporation proprietary deployment program called ASTRAN. This computer analysis program was written during 1987 and has the following characteristics:

- High fidelity, efficient analysis of deformations and loads encountered by flexible truss structures during deployment.
- Large displacements and rotations, small distortions, quasi-static.
- Structural elements are clusters (hinge bodies) and struts connected by hinges.
- Can apply external loads, hinge moments, constraints, and prescribed hinge angles.
- Degrees of freedom are six for each cluster and one for each hinge.
- Written in C language; presently running on PC-family computers.
- ASTRAN is interactive. The analysis can be interrupted in order to examine aspects of the behavior and to change updating procedures in order to help convergence.
- The user develops a good "feel" of how the structure behaves. The similarity to conducting a test is uncanny.

The method has been applied to a two-bay segment of articulated Astromast consisting of 15 clusters, 33 hinges and 27 struts. The results obtained compared well with experimental results. It was also used to predict the stability of deployment of Z-Beam, which was being developed for the now-defunct COFS Mast Flight Experiment program. Results showed successful stowage even with a large imbalance in hinge friction moment.

The effect of the surrounding structure on the segment shown in Figure 13 was simulated by constraining the nodes along the beams to have a zero component

of displacement normal to the plane of the beams. The deployment is assumed to be driven at the two hinges joining the lower non-folding longerons to the inner cluster. The outer bays were allowed to deploy freely.

The analysis revealed a serious deployment defect in the Sixpac design. The inner bay deployed easily. The outer bays also deployed well for the initial stages, exhibiting no member straining, but as the inner bays reached full deployment, the outer bays "hung up" in the partially deployed condition shown in Figure 13. The difficulty arises from the fact that the folding longerons straighten prematurely and lock the outer bays, whose longerons should swing through the horizontal from the stowed downward sloping condition to the deployable upward sloping condition.

Numerous trials were made attempting to get the outer bays to pop through before the inner bay was fully deployed. Only by providing large asymmetrical drive moments to the outer bays could full deployment be achieved.

A model based on the Tripac design was similarly analyzed. Here the results were more promising. It is possible to achieve full deployment by driving the bays whose longerons must pass through the horizontal. Successful deployment, however, will probably require driving at many locations, thereby destroying the main advantage of the Pactruss concept.

One possible remedy is to modify the Pactruss concept somewhat in a manner shown in Figure 14. In this approach, the situation in which a bay's longerons must pass through the horizontal is avoided by requiring such bays to be horizontal when deployed. Extensions, with braces if necessary, provide the required mounting point at the paraboloidal surface. Bays for which the longerons move through less than 90 degrees are designed to be steeper, thereby returning to the paraboloidal surface.

The results for the deployment analysis of a sector of the modified truss shown darkened in Figure 14 and called XGTK are given in Figure 15. The deployment angle of the outer bays is shown as a function of the drive angle at the inner bay. The results for XGTK show smooth progress to full deployment even with no outer helping hinge moment, M_h. In comparison, the results for the unmodified design, identified as XGTEST, show smooth deployment until nearly full deployment, where the structure jumps to the undesirable result.

Clearly, fuller models of the deploying truss must be investigated in evaluating the deployment behavior. The effects of off nominal drive and hinge conditions and the effects of external forces need to be examined. These preliminary results indicate that the modified Pactruss with extensions will deploy successfully.

SECTION 6

CONCLUDING REMARKS

This investigation has shown that it is possible to design a hybrid Pactruss structure capable of supporting reflector panels to form a precise, doubly curved (paraboloidal) reflector surface. The truss can be stowed around a central body to which it is attached for launch and orbital deployment. The deployed truss is very stiff, stiff enough to resist operational loads with sub-micrometer rms distortions and to limit gravity sag to amounts small enough to enable ground testing and measurement of the surface.

Preliminary deployment analyses indicated that the Pactruss, modified with extensions to avoid lockup, can be deployed in a strain-free manner.

REFERENCES

1. Swanson, Paul N., A Lightweight Low Cost Large Deployable Reflector (LDR), JPL D-2283, NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, June 1985.

2. Hedgepeth, John M., Support Structures for Large Infrared Telescopes, NASA Contractor Report 3800, Astro Research Corporation, Carpinteria, California, 1984.

3. Von Roos, A., and J.M. Hedgepeth, Design, Model Fabrication, and Analysis for a Four-Longeron, Synchronously Deployable, Double-Fold Beam Concept, AAC-TN-1139, Astro Aerospace Corporation, Carpinteria, California, March 1985.

4. Hedgepeth, John M., Application of Pactruss to Space Station Structure, AAC-TN-1143, Astro Aerospace Corporation, Carpinteria, California, September 1985.

5. Hedgepeth, John M., Evaluation of Pactruss Design Characteristics Critical to Space Station Primary Structure, NASA Contractor Report 178171, Astro Aerospace Corporation, Carpinteria, California, February 1987.

6. Hedgepeth, John M., Influence of Fabrication Tolerances on the Surface Accuracy of Large Antenna Structures, AIAA Journal, Vol. 20, No. 5, pp. 680-686, May 1982.

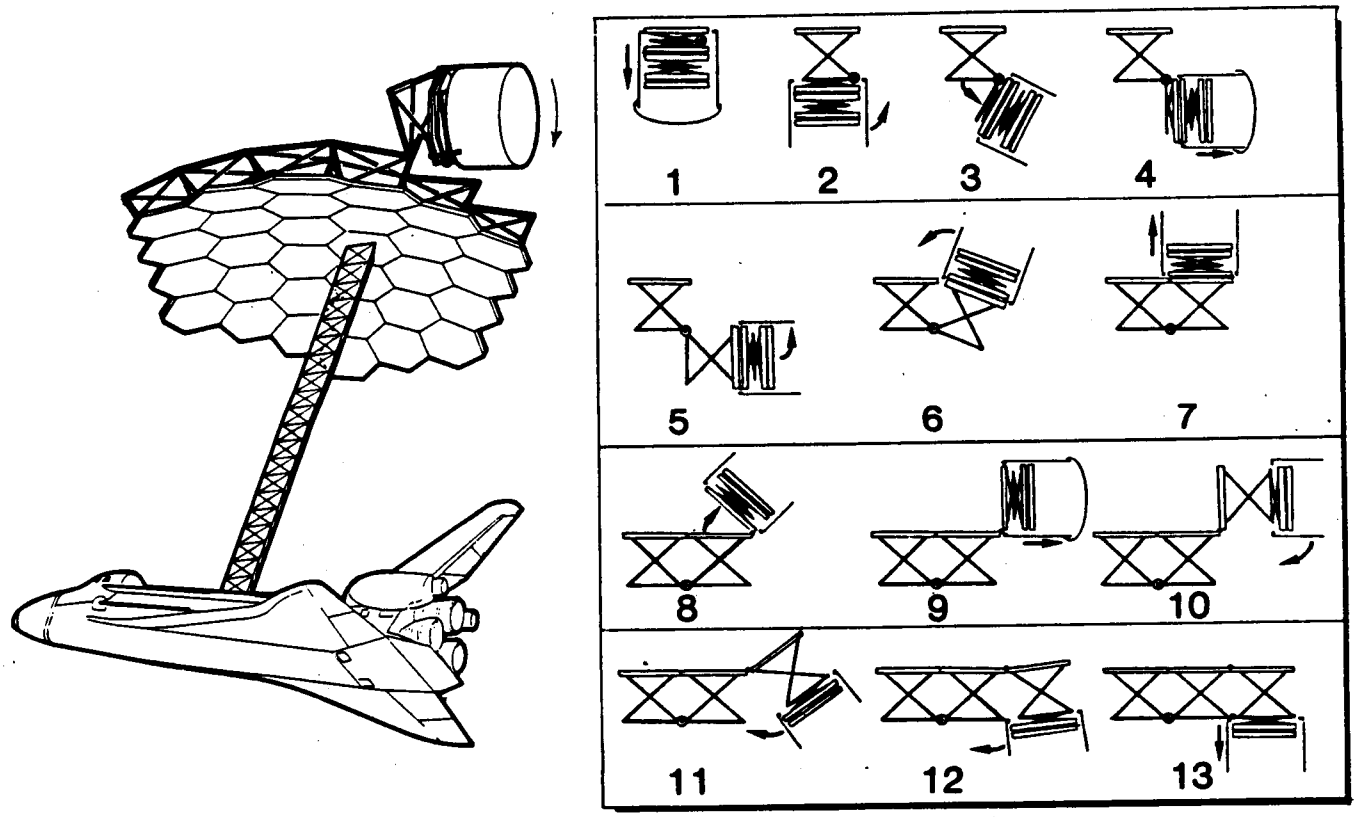


Figure 1. Sequentially deployable precision reflector.

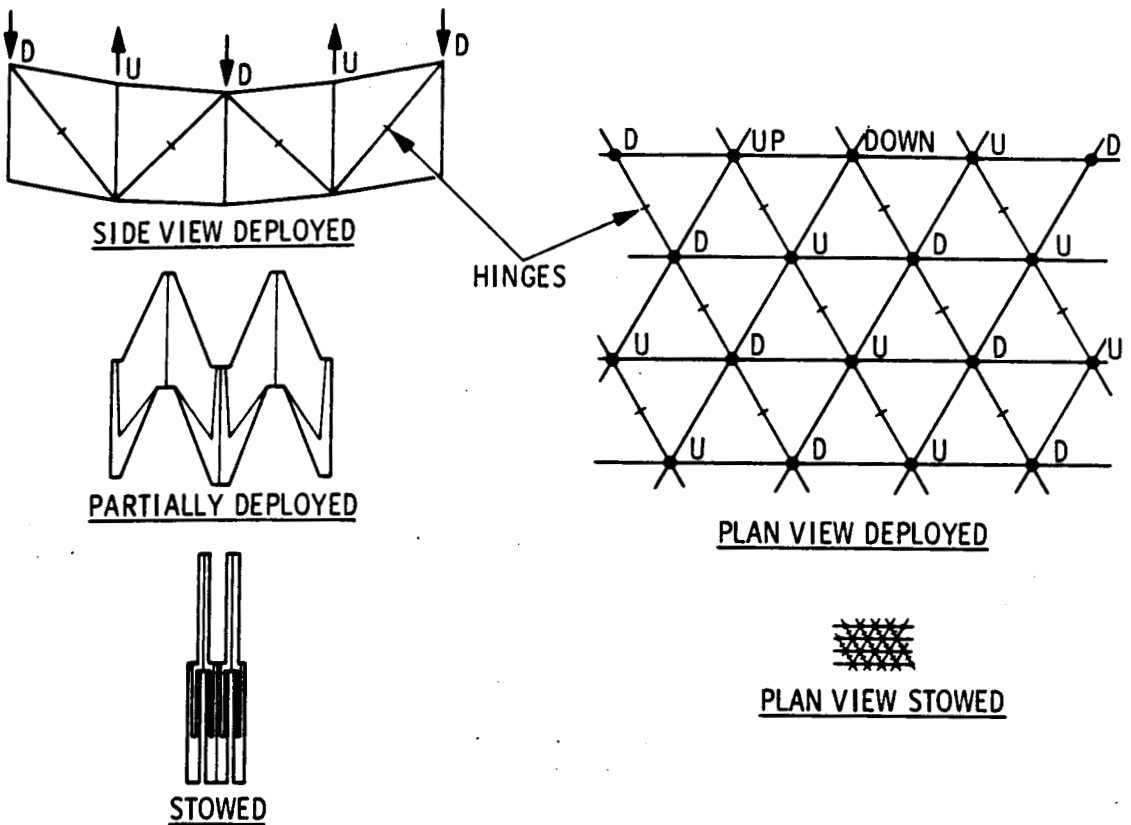


Figure 2. LDR Pactruss deployment scheme.

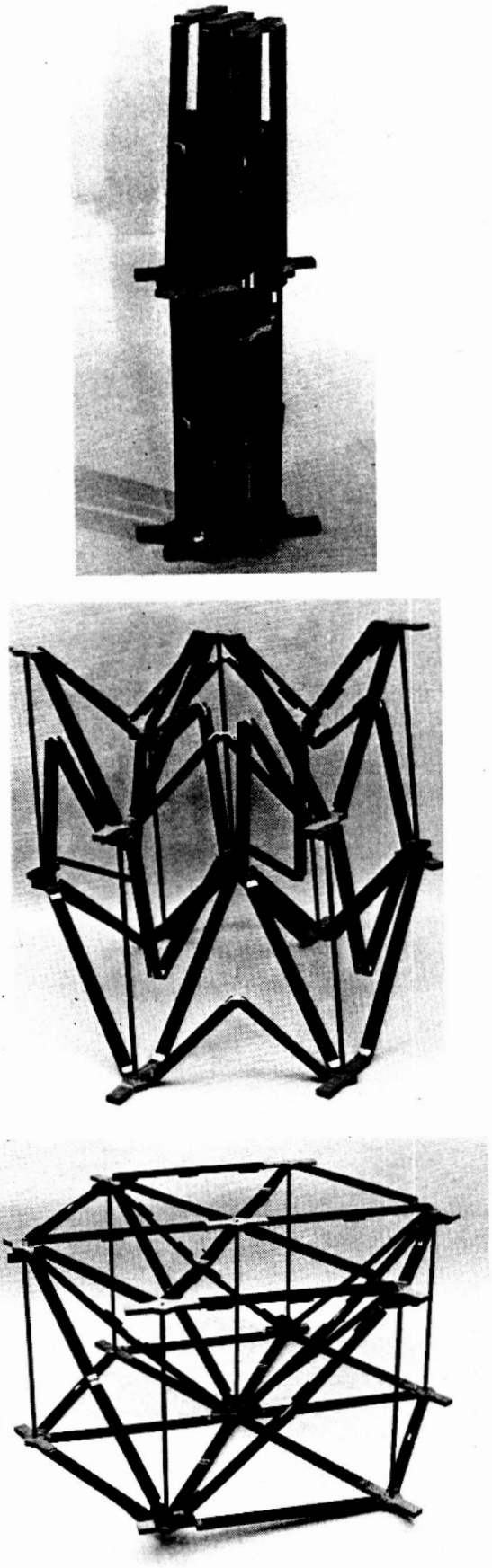
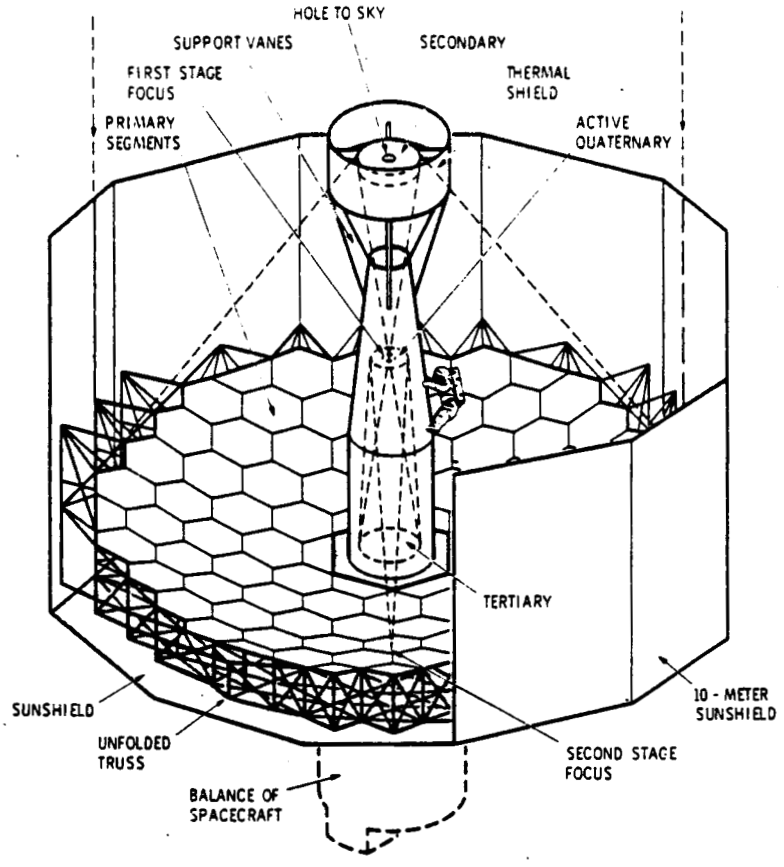
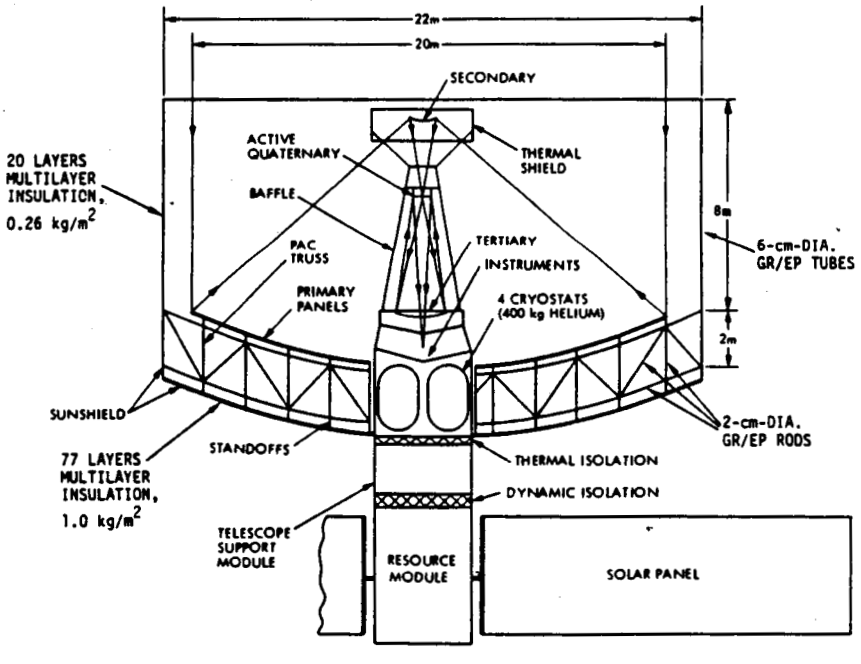


Figure 3. Deployment of conceptual model of triangular Pactruss.

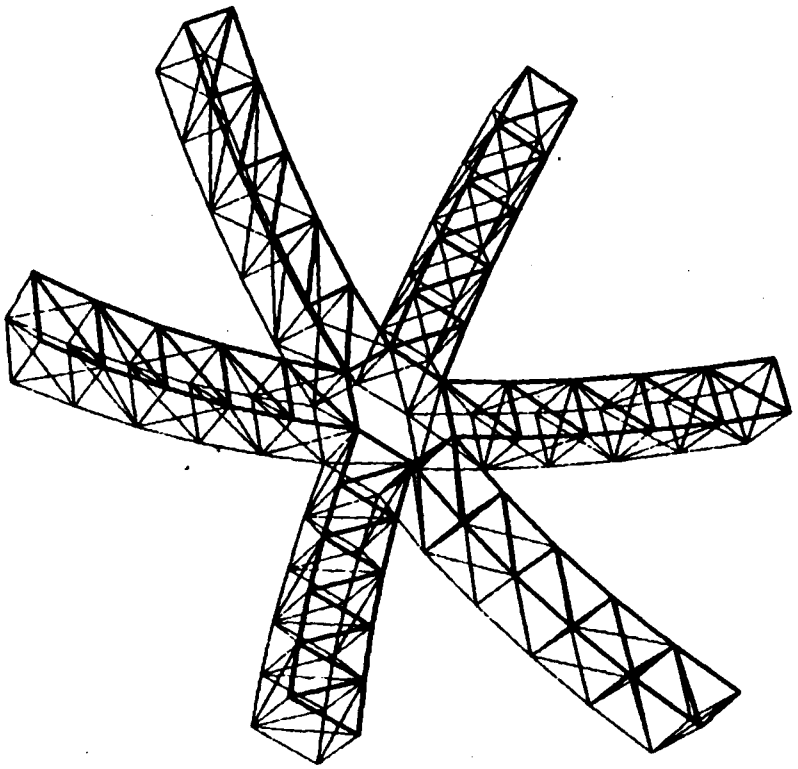


(a) Cutaway perspective

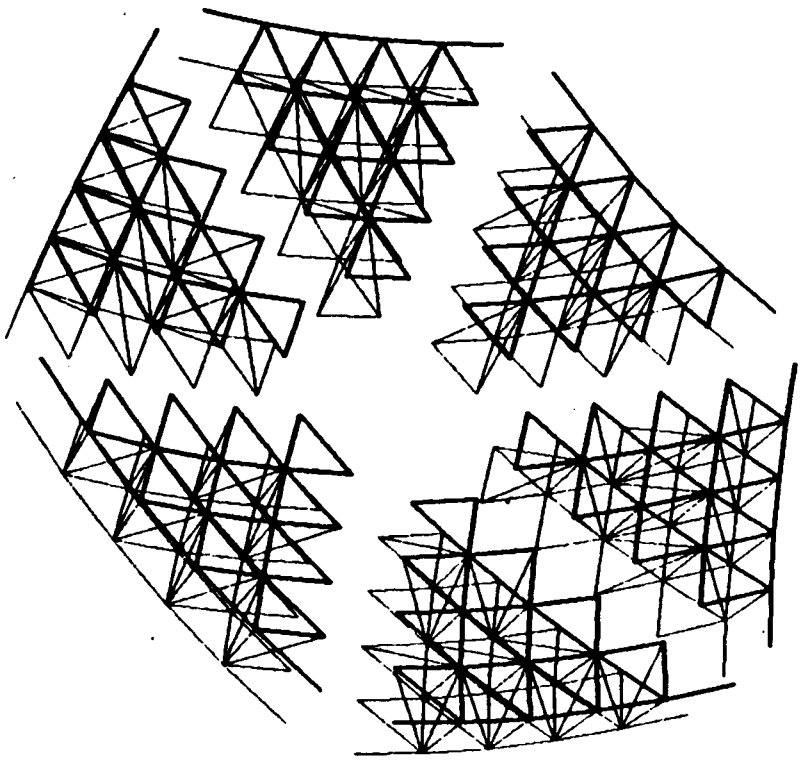


(b) Cross section

Figure 4. Application of Pactruss to the support structure for the primary mirror of the JPL low-cost LDR.

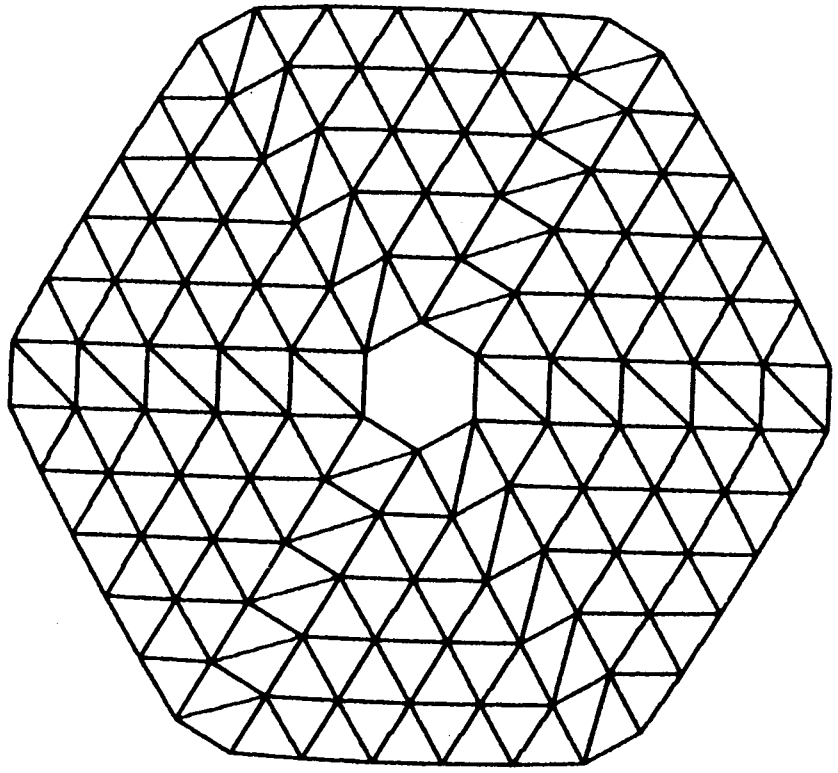


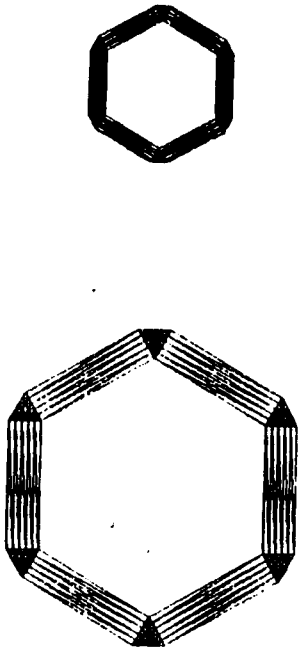
Single-fold beams



Pactruss

Figure 5. Hybrid Pactruss concept.





Enlarged stowed cross section

Figure 6. Stowed and deployed hybrid Pactruss.

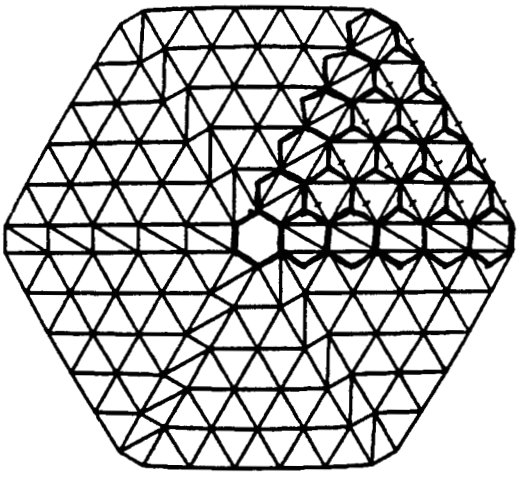


Figure 7. Hexagonal panels attached to hybrid Pactruss.

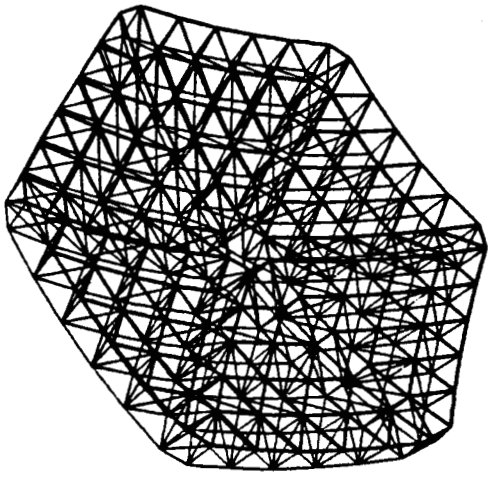


Figure 8. Sixpac support truss.

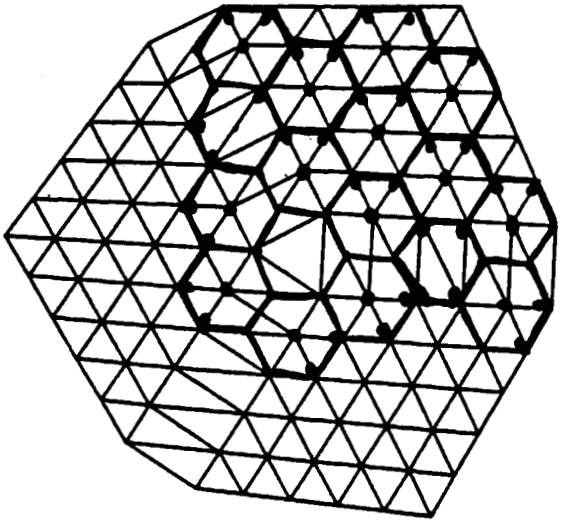


Figure 9. Tripac hybrid Pactruss support structure with panels.

CONFIGURATION 5-Ring, 2-Meter Sixpac	STRUCTURAL MASS (kg)	PANEL MASS (10 kg/m^2) (kg)	MAXIMUM DISPLACEMENT (μm) (1 g)	 1 x 10^{-3} rad/sec^2)
Lightweight: 2.5 cm x 1 mm wall	478	0	0.44	-
	478	3730	3.9*	7.5
Medium Weight: 2.5 cm x 3 mm wall	1214	0	0.40	-
	1214	3730	1.7*	3.3
Solid Struts:	2901	0	0.40	-
	1214	3730	0.9*	0.8
Tapered: Solid Strut, Ring 1	819	0	0.20	-
3 mm wall, Ring 2	819	3730	1.7	3.5
1 mm wall, Outer Rings				

*A few struts near center are overloaded (Euler column buckling)

Figure 10. Static performance for the hybrid Pactruss (E = 30.0 x 10^{10} N/m^2, K_{joint} = 2).

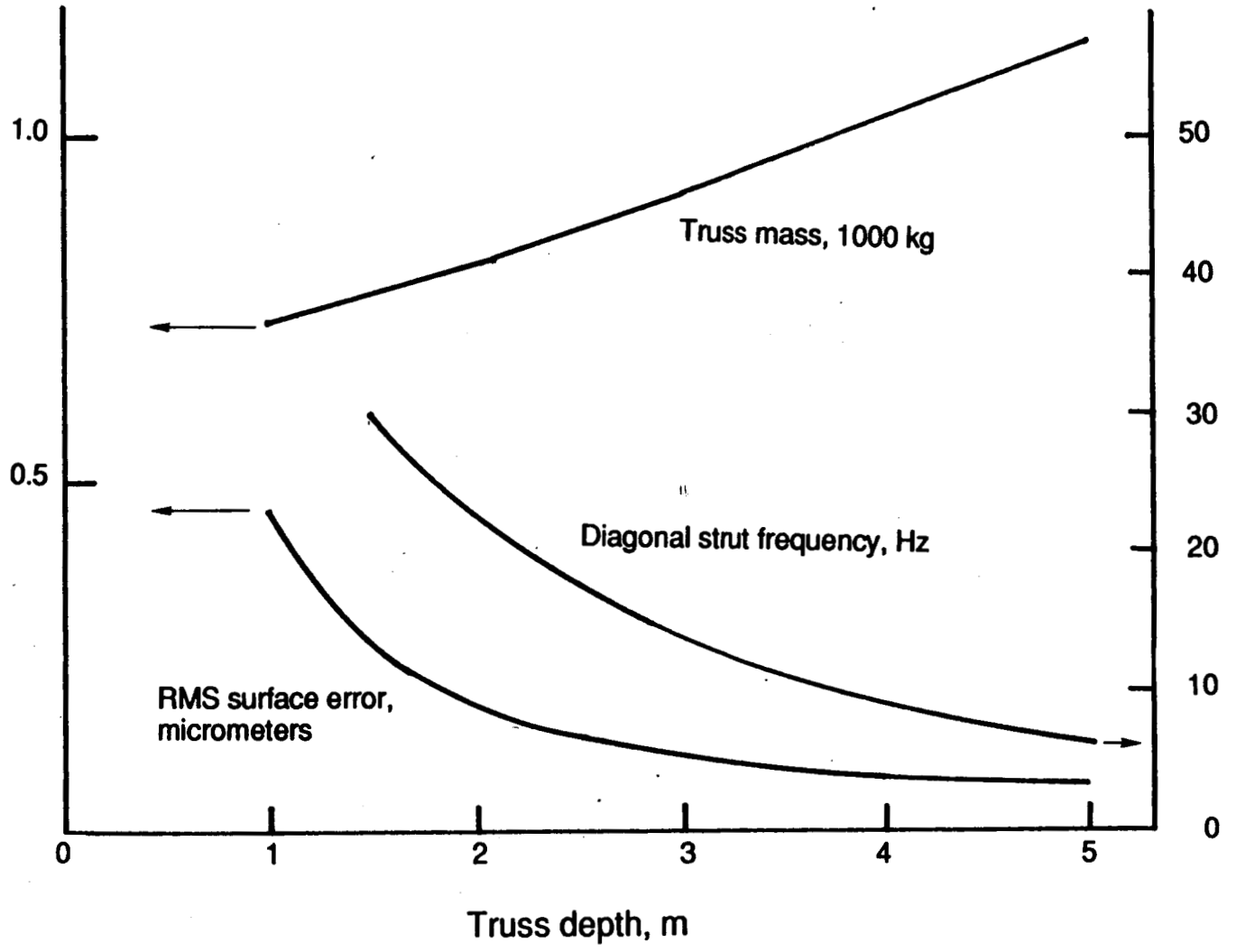


Figure 11. Effect of truss depth.

MODE	FREQUENCY (Hz)	MODE SHAPE
1	1.03718E+01	N = 1, ROCKING ABOUT XX
2	1.03718E+01	N = 1, ROCKING ABOUT YY
3	1.17236E+01	N = 0, ROTATION ABOUT ZZ
4	1.19297E+01	N = 2, NODE LINES AT -45,45 DEG
5	1.19297E+01	N = 2, NODE LINES AT 15,105 DEG
6	1.37824E+01	N = 0, AXIAL PUMPING
7	1.59909E+01	N = 3, NODE LINES AT -30,30,90 DEG
8	1.95641E+01	N = 3, NODE LINES AT 0,60,120 DEG
9	2.47072E+01	N = 4, NODE LINES AT 0,45,90,135 DEG
10	2.47072E+01	N = 4, MAXIMA AT 0,45,90,135 DEG
11	2.84482E+01	N = 5, MAXIMA AT -54,18,90,162,234 DEG
12	2.84482E+01	N = 5, NODE LINES AT -54,18, ETC. DEG
13	3.26987E+01	N = 5, MAXIMA AT -54,18, ETC. DEG
14	3.26987E+01	N = 5, NODE LINES AT -54,18, ETC. DEG
15	3.52508E+01	N = 6, MAXIMA AT 0,60, ETC. DEG
16	3.91162E+01	N = 6, NODE LINES AT 0,60, ETC. DEG
17	4.06708E+01	N = 3, SECOND RADIAL MODE; NODES AT 0,60,...
18	4.06708E+01	N = 3, ??
19	4.21089E+01	N = 4, ??
20	4.21089E+01	N = 2, SECOND RADIAL MODE; MAX AT 0,90,...

Figure 12. Vibration frequencies for center-mounted Sixpac truss with tapered truss stiffness and 10 kg/m² panels.

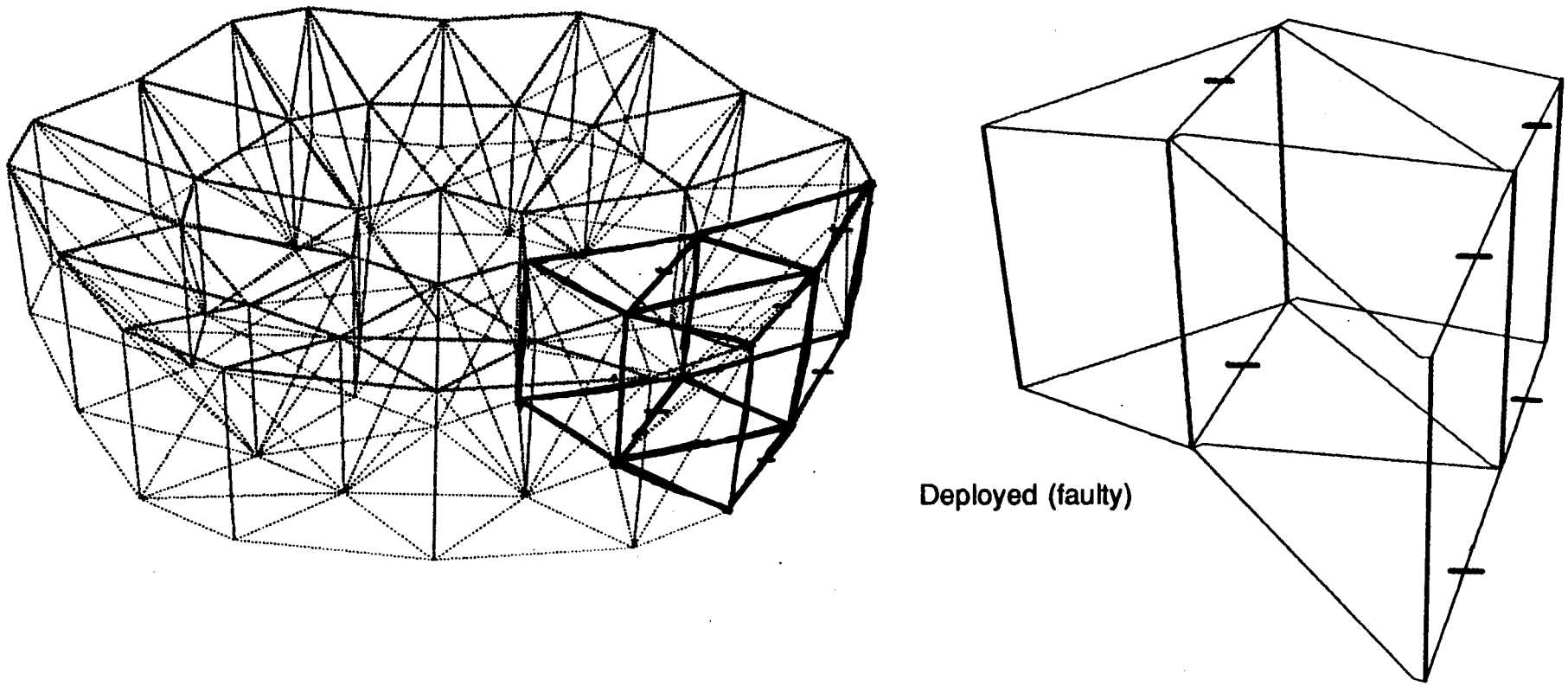


Figure 13. XGTEST, a model for investigating deployment of the sixpac configuration.

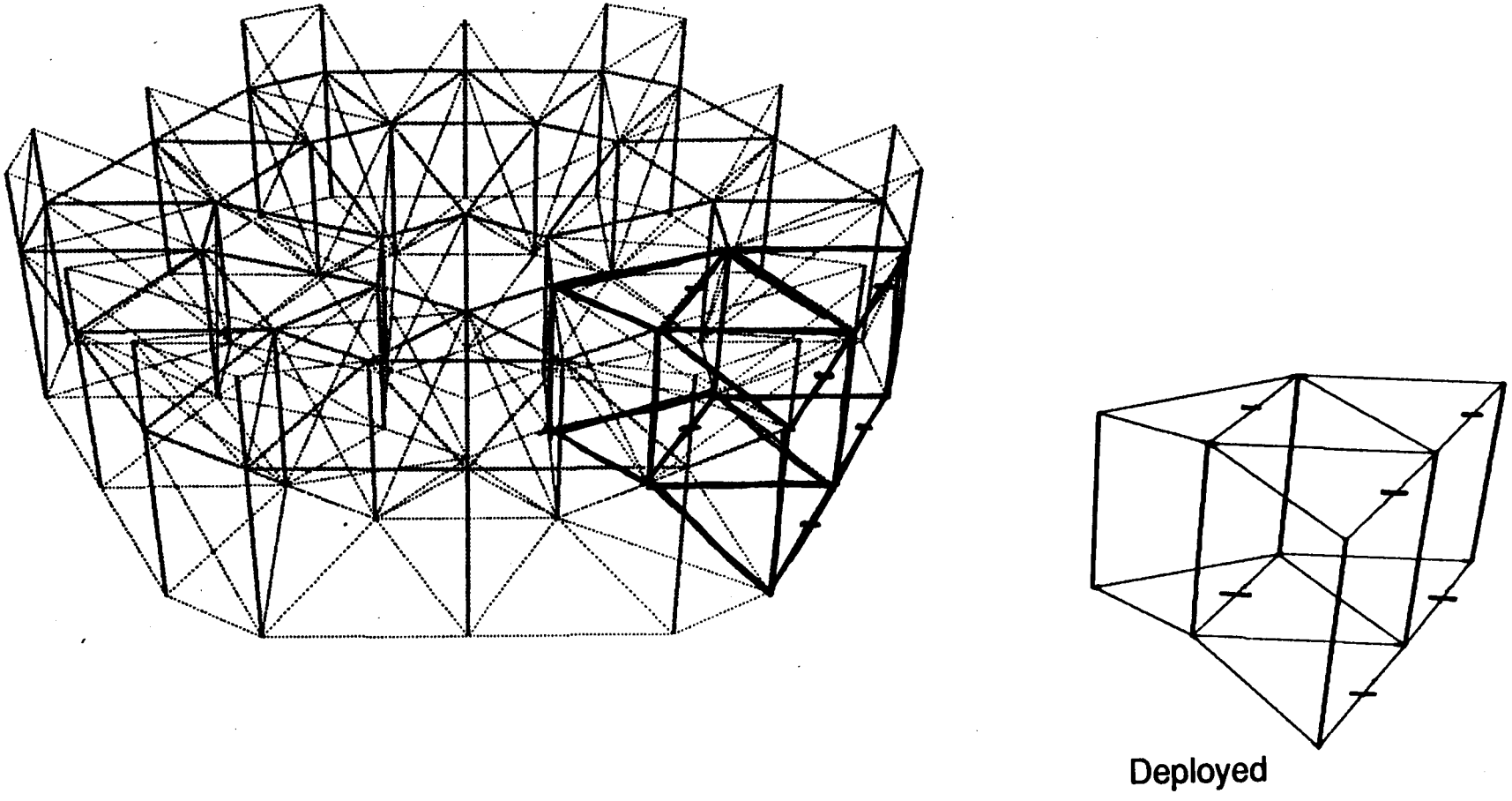


Figure 14. XGTK, a model for deployment investigation of Pactruss with extensions.

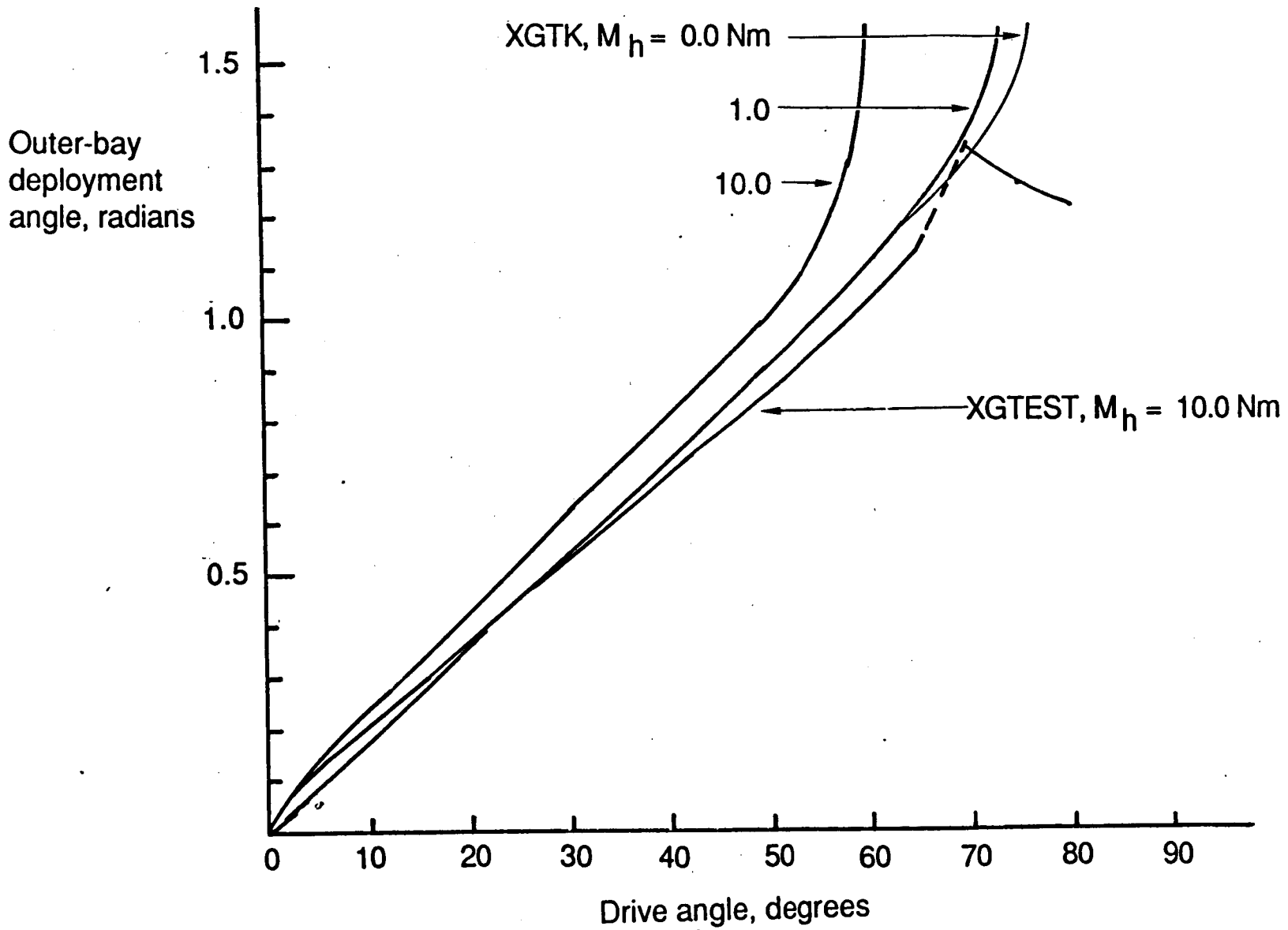


Figure 15. Results of outer-bay deployment for two models.

APPENDIX A

Example Files:

GEN6PAC.C
GEN3PAC.C
GEN6STR.C
GEN3STR.C
TWORING.DTA
SEE2PAL.C
BESTFIT.C

```
/* GEN6PAC.C - Program for generating the basic geometry of a hybrid Pactruss
 *          with six radial single-fold beams bounding the six sectors of
 *          double-fold Pactruss.  The truss nodes are required to lie on
 *          the paraboloidal surface
 *
 *                      z = r*r/(4*F)
 *
 *          where F is the focal length and r is the radius
 *
 *
 *                      r = sqrt(x*x + y*y)
 *
 *          The truss is composed of M rings.  The number of nodes in each
 *          sector is (M + 1)*(M + 2)/2.  The length of the non-folding
 *          surface struts is l, and the width of the single-fold beams is b.
 *          The truss height is h.
 *
 *          Although this analysis does not explicitly consider the effects
 *          of joint offsets, the geometry is exact for the case in which the
 *          length l is divided between offsets, totalling 3.5 tube diameters
 *          times the secant of the surface slope at the rim, and the strut
 *          itself.  The stowed geometry then can be obtained by uniformly
 *          shrinking the planform and appropriately adjusting the z-coordi-
 *          nate of each node.  Note that the deployment mechanics may
 *          necessarily involve some straining of the members as a result of
 *          the double curvature of the nodal surface, not as a result of
 *          finite offsets.
 *
 *          The nodes are numbered consecutively within each sector, starting
 *          at the first sector with 1, 2, 4, 7, 11, 16, etc. outward along
 *          the beam boundary with intermediate numbering counterclockwise
 *          along each ring.  Nodes in succeeding sectors are numbered in the
 *          same order consecutively.  The entire upper surface is numbered
 *          first, then the lower, and finally any accessory nodes are as-
 *          signed numbers.
 *
 *          The focal point is assigned the number 0.
 *
 *          The results are stored in ASCII form in the file XYZ.DAT where
 *          XYZ is entered in the command tail.  The default is SIXPAC.
 *          The first entry is the parameter values.  The second line contains
 *          the total number of nodes in the truss.  The following lines are
 *          the coordinates of each node in x,y,z form listed in order.
 *
 *          The input parameters M, F, l, b, and h are obtained from the file
 *          XYZ.PRM, which is a one-line ASCII file with the values, in order,
 *          separated by commas.  The default values for SIXPAC are 6, 10.0,
 *          2.0, 1.154701, and 2.0.
 *
 *                                          John M. Hedgepeth   8/26/87
 *
 *      Altered to allow F = infinity by inputting F < 0        9/16/87
 *      Updated to MS C 5.0                                     12/29/87
 *
 */

#include <scitech.h>

int M = 5;
double F = 10.;
```

```
double l =  2.;
double b = 1.154701;
double h = 2.;

void initbeam(),newpoint(),fillsec();

main(argc,argv)
int argc;
char *argv[];
{
    int i,j,n,chr,N;
    static double x,y,z,coord[100][3],cosv,sinv;
    static char inname[50],outname[50];
    FILE *infile,*outfile;

    strcpy(outname,"SIXPAC.DAT");

    if(argc > 1) {
        strcpy(outname,argv[1]);
        strcat(outname,".DAT");
        strcpy(inname,argv[1]);
        strcat(inname,".PRM");
        strupr(inname);

        if((infile = fopen(inname,"r")) == NULL) {
            printf("\nCannot open %s.  Aborted.\n",inname);
            exit(1);
        }
        if(fscanf(infile," %d %lf %lf %lf %lf", &M, &F, &l, &b, &h) != 5) {
            printf("\nCannot read input parameters.  Aborted.\n");
            exit(1);
        }
        fclose(infile);

        if(F == 0)
            F = -10000.;
    }
    N = ((M + 1)*(M + 2))/2;

    if((outfile = fopen(outname,"r")) != NULL) {
        fflush(stdin);
        printf("\nFile %s exists.  Do you want to write over it? (Y/N) <N>",
                                                                    outname);
        if(toupper(getch()) != 'Y')
            exit(1);

        fclose(outfile);
    }
    outfile = fopen(outname,"wt");

    fprintf(outfile," %d, %lf, %lf, %lf, %lf\n",M, F, l, b, h);
    fprintf(outfile," %d\n",12*N + 1);
    x = 0;
    fprintf(outfile," %.8le, %.8le, %.8le\n",x,x,F);

    initbeam(coord);

    fillsec(coord);

    for(i=0; i<6; i++) {
```

```
        cosv = cos((double)i*PI/3);
        sinv = sin((double)i*PI/3);

        for(n=1; n<= N; n++) {
            x = coord[n][0]*cosv - coord[n][1]*sinv;
            y = coord[n][0]*sinv + coord[n][1]*cosv;
            z = coord[n][2];
            fprintf(outfile," %.8le, %.8le, %.8le\n",x,y,z);
        }
    }

    for(i=0; i<6; i++) {
        cosv = cos((double)i*PI/3);
        sinv = sin((double)i*PI/3);

        for(n=1; n<= N; n++) {
            x = coord[n][0]*cosv - coord[n][1]*sinv;
            y = coord[n][0]*sinv + coord[n][1]*cosv;
            z = coord[n][2] - h;
            fprintf(outfile," %.8le, %.8le, %.8le\n",x,y,z);
        }
    }
    fclose(outfile);

    printf("\nFile %s successfully written.\n",outname);
}


void initbeam(coord)
double coord[100][3];
{
    int i,n,j;
    double x,y,z,xold,ksi,sqrt3,eps;

    sqrt3 = 3.;
    sqrt3 = sqrt(sqrt3);

    x = sqrt3*b/2.;
    y = b/2.;
    coord[1][0] = x;
    coord[1][1] = y;
    coord[1][2] = b*b/(4.*F);
    if(F < 0)
        coord[1][2] = 0;

    for(i=1; i<=M; i++) {
        n = (i*(i + 1))/2 + 1;
        if(F < 0) {
            x += 1;
            coord[n][2] = 0;
        }
        else {
            xold = x;
            eps = 1/2.;
            for(j=0; j<50; j++,eps/=2.) {
                ksi = x + eps - xold;
                z = ksi*(ksi + 2.*xold)/(4.*F);
                if(ksi*ksi + z*z <= 1*1)
                    x += eps;
            }
```

```
            coord[n][2] = (x*x + y*y)/(4.*F);
        }
        coord[n][0] = x;
        coord[n][1] = y;
    }
}


void fillsec(coord)
double coord[100][3];
{
    int i,n,p,q,r;
    double x,y,z,temp,cosv,sinv;

    temp = PI/3.;
    cosv = cos(temp);
    sinv = sin(temp);

/* First fill in the other radial boundary
 */
    for(i=1; i<=M; i++) {
        n = ((i + 1)*(i + 2))/2;
        x = coord[n - i][0] - b*sinv;

        coord[n][0] = x*cosv + b*sinv;
        coord[n][1] = x*sinv + b*cosv;
        coord[n][2] = coord[n - i][2];
    }

/* Then fill horizontal rows
 */
    for(i=1; i<=M-1; i++)
        for(n=i+1; n<=M; n++) {
            p = ((n - 1)*n)/2 + i;
            q = p + 1;
            r = (n*(n + 1))/2 + i + 1;

            newpoint(p,q,r,coord);
        }
}


void newpoint(p,q,r,coord)
int p,q,r;
double coord[100][3];
{
    int i,n;
    double x,y,z,zold,Rbar[3],delR[3],rhosq,d,temp,eps,mult;

    for(i=0,rhosq=0; i<3; i++) {
        Rbar[i] = (coord[p][i] + coord[q][i])/2;
        delR[i] = coord[p][i] - coord[q][i];
        rhosq += delR[i]*delR[i];
    }

    d = rhosq - delR[2]*delR[2];
    zold = (Rbar[0]*Rbar[0] + Rbar[1]*Rbar[1])/(4.*F);
    eps = zold/16;

    for(i=0,mult=1.; i<50; i++,eps*=mult) {
```

```
            z = zold + eps - Rbar[2];
            if(F < 0) {
                z = zold = eps = 0;
                i = 49;
            }
            temp = l*l - rhosq*(0.25 + z*z/d);
            temp /= d;
            temp = sqrt(temp);

            x = Rbar[0] - z*delR[0]*delR[2]/d - delR[1]*temp;
            y = Rbar[1] - z*delR[1]*delR[2]/d + delR[0]*temp;

            if(x*x + y*y >= 4.*F*(zold + eps))
                zold += eps;
            else mult = 0.5;

        }
        coord[r][0] = x;
        coord[r][1] = y;
        coord[r][2] = zold;
    }
```

```
/* GEN3PAC.C - Program for generating the basic geometry of a hybrid Pactruss
 *          with three radial single-fold beams bounding the three sectors of
 *          double-fold Pactruss.  The truss nodes are required to lie on
 *          the paraboloidal surface
 *
 *                      z = r*r/(4*F)
 *
 *          where F is the focal length and r is the radius
 *
 *
 *                      r = sqrt(x*x + y*y)
 *
 *          The truss is composed of M rings.  The number of nodes in each
 *          sector is M*(M + 3) + 1.  The length of the non-folding
 *          surface struts is l, and the width of the single-fold beams is b.
 *          The truss height is h.
 *
 *          Although this analysis does not explicitly consider the effects
 *          of joint offsets, the geometry is exact for the case in which the
 *          length l is divided between offsets, totalling 3.5 tube diameters
 *          times the secant of the surface slope at the rim, and the strut
 *          itself.  The stowed geometry then can be obtained by uniformly
 *          shrinking the planform and appropriately adjusting the z-coordi-
 *          nate of each node.  Note that the deployment mechanics will
 *          necessarily involve some straining of the members as a result of
 *          the double curvature of the nodal surface, not as a result of
 *          finite offsets.
 *
 *          The nodes are numbered consecutively within each sector, starting
 *          at the first sector with 1, 3, 7, 13, 21, etc. outward along
 *          the boundary of the beam running out along the x axis.  Along the
 *          other beam the points are numbered 1, 4, 7, 14, 22, etc.  The
 *          other points in the first sextant is numbered with odd numbers
 *          increasing toward the center.  The even numbers are assigned to
 *          the symmetrically located points in the second sextant.  Points are
 *          assigned to the midpanel points of the beam for possible use; they
 *          are even numbered 2, 6, 12, 20, 30, etc.  Nodes in succeeding
 *          sectors are numbered in the same order consecutively.  The entire
 *          upper surface is numbered first, then the lower, and finally any
 *          accessory nodes are assigned numbers.
 *
 *          The focal point is assigned the number 0.
 *
 *          The results are stored in ASCII form in the file XYZ.DAT where
 *          XYZ is entered in the command tail.  The default is SIXPACK.
 *          The first entry is the parameter values.  The second line contains
 *          the total number of nodes in the truss.  The following lines are
 *          the coordinates of each node in x,y,z form listed in order.
 *
 *          The input parameters M, F, l, b, and h are obtained from the file
 *          XYZ.PRM.  The default values for TRIPAC are 5, 10.0, 2.0, 3.4641,
 *          and 2.0.
 *
 *                                          John M. Hedgepeth   9/6/87
 *
 *      Updated to MS C 5.0                                     12/29/87
 *
 */

#include <scitech.h>
```

```
int M = 5;
double F = 10.;
double l =  2.;
double b = 3.4641;
double h = 2.;

void initbeam(),sympoint(),newpoint(),fillsec();

main(argc,argv)
int argc;
char *argv[];
{
    int i,j,n,chr,N;
    double x,y,z,coord[100][3],cosv,sinv;
    char inname[50],outname[50];
    FILE *infile,*outfile;

    strcpy(outname,"TRIPAC.DAT");

    if(argc > 1) {
        strcpy(outname,argv[1]);
        strcat(outname,".DAT");
        strcpy(inname,argv[1]);
        strcat(inname,".PRM");
        strupr(inname);

        if((infile = fopen(inname,"r")) == NULL) {
            printf("\nCannot open %s.  Aborted.\n",inname);
            exit(1);
        }
        if(fscanf(infile," %d %lf %lf %lf %lf", &M, &F, &l, &b, &h) != 5) {
            printf("\nCannot read input parameters.  Aborted.\n");
            exit(1);
        }
        fclose(infile);
    }
    N = M*(M + 3) + 1;

    if((outfile = fopen(outname,"r")) != NULL) {
        fflush(stdin);
        printf("\nFile %s exists.  Do you want to write over it? (Y/N) <N>",
                                                                    outname);
        if(toupper(getch()) != 'Y')
            exit(1);

        fclose(outfile);
    }
    outfile = fopen(outname,"wt");

    fprintf(outfile," %d, %lf, %lf, %lf, %lf\n",M, F, l, b, h);
    fprintf(outfile," %d\n",6*N + 1);
    x = 0;
    fprintf(outfile," %.8le, %.8le, %.8le\n",x,x,F);

    initbeam(coord);

    fillsec(coord);

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

```
        cosv = cos((double)i*PI*2./3.);
        sinv = sin((double)i*PI*2./3.);

        for(n=1; n<= N; n++) {
            x = coord[n][0]*cosv - coord[n][1]*sinv;
            y = coord[n][0]*sinv + coord[n][1]*cosv;
            z = coord[n][2];
            fprintf(outfile," %.8le, %.8le, %.8le\n",x,y,z);
        }
    }

    for(i=0; i<3; i++) {
        cosv = cos((double)i*PI*2./3.);
        sinv = sin((double)i*PI*2./3.);

        for(n=1; n<= N; n++) {
            x = coord[n][0]*cosv - coord[n][1]*sinv;
            y = coord[n][0]*sinv + coord[n][1]*cosv;
            z = coord[n][2] - h;
            fprintf(outfile," %.8le, %.8le, %.8le\n",x,y,z);
        }
    }
    fclose(outfile);

    printf("\nFile %s successfully written.\n",outname);
}


void initbeam(coord)
double coord[100][3];
{
    int i,n,j;
    double x,y,z,xold,ksi,sqrt3,eps;

    sqrt3 = 3.;
    sqrt3 = sqrt(sqrt3);

    x = b/(2.*sqrt3);
    y = b/2.;
    coord[1][0] = x;
    coord[1][1] = y;
    coord[1][2] = (x*x + y*y)/(4.*F);

    for(i=1; i<=M; i++) {
        n = i*(i + 1) + 1;
        xold = x;
        eps = 1/2.;
        for(j=0; j<50; j++,eps/=2.) {
            ksi = x + eps - xold;
            z = ksi*(ksi + 2.*xold)/(4.*F);
            if(ksi*ksi + z*z <= 1*1)
                x += eps;
        }
        coord[n][0] = x;
        coord[n][1] = y;
        coord[n][2] = (x*x + y*y)/(4.*F);

        coord[n - 1][0] = (xold + x)/2.;
        coord[n - 1][1] = 0;
        coord[n - 1][2] = x*x/(4.*F);
```

```
    }
}


void fillsec(coord)
double coord[100][3];
{
    int i,n,p,q,r;
    double x,y,z;

/* First fill in the other radial boundary
 */
    for(i=1; i<=M; i++) {
        n = i*(i + 1) + 1;
        sympoint(n,coord);
    }

/* Then fill slanted rows and symmetrical points
 */
    for(i=1; i<=M; i++) {
        for(n=1; n<i; n++) {
            r = i*(i + 1) + 1 + 2*n;
            p = r - 2;
            q = r - 2*i;

            newpoint(p,q,r,coord);
            sympoint(r,coord);
        }
        r = i*(i + 3) + 1;
        p = r - 2;
        q = r - 1;
        newpoint(p,q,r,coord);
    }
}


void sympoint(p,coord)
int p;
double coord[100][3];
{
    double x,y,temp,cosv,sinv;

    temp = 2.*PI/3.;
    cosv = cos(temp);
    sinv = sin(temp);

    x = coord[p][0];
    y = coord[p][1];

    coord[p + 1][0] = x*cosv + y*sinv;
    coord[p + 1][1] = x*sinv - y*cosv;
    coord[p + 1][2] = coord[p][2];
}


void newpoint(p,q,r,coord)
int p,q,r;
double coord[100][3];
{
    int i,n;
```

```
    double x,y,z,zold,Rbar[3],delR[3],rhosq,d,temp,eps,mult;

    for(i=0,rhosq=0; i<3; i++) {
        Rbar[i] = (coord[p][i] + coord[q][i])/2;
        delR[i] = coord[p][i] - coord[q][i];
        rhosq += delR[i]*delR[i];
    }

    d = rhosq - delR[2]*delR[2];
    zold = (Rbar[0]*Rbar[0] + Rbar[1]*Rbar[1])/(4.*F);
    eps = zold/16;

    for(i=0,mult=1.; i<50; i++,eps*=mult) {
        z = zold + eps - Rbar[2];
        temp = 1*1 - rhosq*(0.25 + z*z/d);
        temp /= d;
        temp = sqrt(temp);

        x = Rbar[0] - z*delR[0]*delR[2]/d - delR[1]*temp;
        y = Rbar[1] - z*delR[1]*delR[2]/d + delR[0]*temp;

        if(x*x + y*y >= 4.*F*(zold + eps))
            zold += eps;
        else mult = 0.5;

    }
    coord[r][0] = x;
    coord[r][1] = y;
    coord[r][2] = zold;
}
```

```
/* GEN6STR.C - Generates the connectivity array for the six-spoke hybrid
 *          Pactruss.  Input data is read from XYZ.DAT.
 *
 *          The command is
 *                  gen6str xyz
 *          If xyz is omitted, the default is SIXPAC.
 *
 *          The connectivity array is created by the routines, is appended to
 *          the input nodal array and the combination is output to XYZ.DTA
 *          which is an ASCII file in the format needed for input to
 *          SEETRUSS.CMD.
 *
 *          The types of struts are:
 *
 *              Numbering
 *          External Internal
 *
 *             1        0   Non-folding Pactruss upper-surface struts
 *             2        1   Upper-surface Pactruss diagonals
 *             3        2   Upper-surface beam longerons and battens
 *             4        3   Upper-surface beam diagonals
 *             5        4   Non-folding Pactruss lower-surface struts
 *             6        5   Lower-surface Pactruss diagonals
 *             7        6   Lower-surface beam longerons and battens
 *             8        7   Lower-surface beam diagonals
 *             9        8   Pactruss verticals
 *             :        9   Beam verticals
 *             ;       10   Core Pactruss diagonals
 *             <       11   Beam core diagonals
 *
 *                                          John M. Hedgepeth       8/27/87
 *
 *      Updated to MS C 5.0                                         12/29/87
 *
 */

#include <scitech.h>
int M;
int Ntypes = 12;

int pacsurf(),paccore(),beamsurf(),beamcore();

main(argc,argv)
int argc;
char *argv[];
{
    int i,j,n,N,Nnodes,Nstruts,n1,n2,type;
    int Npacsurf,Npaccore,Nbeamsurf,Nbeamcore;
    int con1[100][3],con2[100][3],con3[100][3],con4[100][3];
    double x,y,z;
    char inname[50],outname[50];
    FILE *infile,*outfile;

    strcpy(inname,"SIXPAC.DAT");
    strcpy(outname,"SIXPAC.DTA");

    if(argc > 1) {
        strcpy(inname,argv[1]);
        strcpy(outname,argv[1]);
        strcat(inname,".DAT");
```

```
    strcat(outname,".DTA");
}

strupr(inname);
strupr(outname);

if((infile = fopen(inname,"rt")) == NULL) {
    printf("\nCannot open %s.  Aborted.\n",inname);
    exit(1);
}
if(fscanf(infile," %d, %lf, %lf, %lf, %lf %d",
                                        &M,&x,&x,&x,&x,&Nnodes) != 6) {
    printf("\nCannot read input parameters.  Aborted.\n");
    exit(1);
}

if((outfile = fopen(outname,"r")) != NULL) {
    fflush(stdin);
    printf("\nFile %s exists.  Do you want to write over it? (Y/N) <N>",
                                                              outname);
    if(toupper(getch()) != 'Y')
        exit(1);

    fclose(outfile);
}
outfile = fopen(outname,"wt");

N = ((M + 1)*(M + 2))/2;
Npacsurf = pacsurf(con1);
Npaccore = paccore(con2);
Nbeamsurf = beamsurf(con3);
Nbeamcore = beamcore(con4);
Nstruts = 6*(2*Npacsurf + Npaccore + 2*Nbeamsurf + Nbeamcore);

fprintf(outfile," %d, %d, %d\n",Nnodes,Nstruts,Ntypes);

for(i=0; i<Nnodes; i++) {
    if(fscanf(infile," %lf, %lf, %lf",&x,&y,&z) != 3) {
        printf("Corrupt data for node %d.  Aborted.\n",i);
        exit(1);
    }
    fprintf(outfile," %.8le, %.8le, %.8le\n",x,y,z);
}

fclose(infile);

for(j=0; j<Npacsurf; j++) {
    for(i=0; i<6; i++) {
        n1 = con1[j][0] + i*N;
        n2 = con1[j][1] + i*N;
        fprintf(outfile," %d, %d, %d\n",n1,n2,con1[j][2]);
        n1 += 6*N;
        n2 += 6*N;
        fprintf(outfile," %d, %d, %d\n",n1,n2,con1[j][2] + 4);
    }
}

for(j=0; j<Npaccore; j++) {
    for(i=0; i<6; i++) {
        n1 = con2[j][0] + i*N;
```

```
            n2 = con2[j][1] + i*N;
            fprintf(outfile," %d, %d, %d\n",n1,n2,con2[j][2]);
        }
    }

    for(j=0; j<Nbeamsurf; j++) {
        for(i=0; i<6; i++) {
            n1 = con3[j][0] + i*N;
            n2 = con3[j][1] + i*N;
            if(n1 > 6*N)
                n1 -= 6*N;
            if(n2 > 6*N)
                n2 -= 6*N;
            fprintf(outfile," %d, %d, %d\n",n1,n2,con3[j][2]);

            n1 += 6*N;
            n2 += 6*N;
            fprintf(outfile," %d, %d, %d\n",n1,n2,con3[j][2] + 4);
        }
    }

    for(j=0; j<Nbeamcore; j++) {
        for(i=0; i<6; i++) {
            n1 = con4[j][0] + i*N;
            n2 = con4[j][1] + i*N;
            if(n1 > 6*N)
                n1 -= 6*N;
            if(n2 > 12*N)
                n2 -= 6*N;
            fprintf(outfile," %d, %d, %d\n",n1,n2,con4[j][2]);
        }
    }

    fclose(outfile);

    printf("\n\nData file %s successfully written.\n",outname);
}


int pacsurf(buf)
int buf[100][3];
{
    int i,n,p,n1,n2;

    for(n=1,p=0; n <= M; n++) {
        for(i=1; i<=n; i++) {
            n1 = (n*(n + 1))/2 + i + 1;
            n2 = ((n - 1)*n)/2 + i;

            buf[p][0] = n1;
            buf[p][1] = n1 - 1;
            buf[p][2] = 1;
            p++;

            if(i == n)
                break;

            buf[p][0] = n1;
            buf[p][1] = n2;
            buf[p][2] = 0;
```

```
            p++;

            buf[p][0] = nl;
            buf[p][1] = n2 + 1;
            buf[p][2] = 0;
            p++;
        }
    }
    return p;
}


int paccore(buf)
int buf[100][3];
{
    int i,n,p,N,nl,n2;

    N = 6*((M + 1)*(M + 2))/2;

    for(n=2,p=0; n<=M; n++) {
        for(i=0; i <= n; i++) {
            nl = (n*(n + 1))/2 + i + 1;

            if(i>0 && i<n) {
                buf[p][0] = nl;
                buf[p][1] = nl + N;
                buf[p][2] = 8;
                p++;
            }

            if((n & 1) == 0) {
                n2 = ((n - 1)*n)/2 + i;

                if(i>0 && i<n) {
                    buf[p][0] = nl;
                    buf[p][1] = n2 + N;
                    buf[p][2] = 10;
                    p++;

                    buf[p][0] = nl;
                    buf[p][1] = n2 + N + 1;
                    buf[p][2] = 10;
                    p++;
                }

                n2 = ((n + 1)*(n + 2))/2 + i + 1;

                if(i > 0 && n < M) {
                    buf[p][0] = nl;
                    buf[p][1] = n2 + N;
                    buf[p][2] = 10;
                    p++;
                }

                if(i < n && n < M) {
                    buf[p][0] = nl;
                    buf[p][1] = n2 + N + 1;
                    buf[p][2] = 10;
                    p++;
                }
```

```
            }
        }
    }
    return p;
}


int beamsurf(buf)
int buf[100][3];
{
    int i,nl,n2,p,N;

    N = 5*((M + 1)*(M + 2))/2;

    for(i=0,p=0; i<=M; i++) {
        nl = (i*(i + 1))/2 + 1;
        n2 = ((i + 1)*(i + 2))/2 + 1;

/* Beam battens
 */
        buf[p][0] = nl;
        buf[p][1] = nl + N + i;
        buf[p][2] = 2;
        p++;

        if(i == M)
            break;

/* Beam longerons
 */
        buf[p][0] = nl;
        buf[p][1] = n2;
        buf[p][2] = 2;
        p++;

        buf[p][0] = nl + N + i;
        buf[p][1] = n2 + N + i + 1;
        buf[p][2] = 2;
        p++;

/* Beam diagonals
 */
        buf[p][0] = nl;
        buf[p][1] = n2 + N + i + 1;
        buf[p][2] = 3;
        p++;
    }
    return p;
}



int beamcore(buf)
int buf[100][3];
{
    int i,nl,n2,p,N5,Nsurf;

    N5 = 5*((M + 1)*(M + 2))/2;
    Nsurf = 6*N5/5;
```

```
    for(i=0,p=0; i<=M; i++) {
        n1 = (i*(i + 1))/2 + 1;

        buf[p][0] = n1;
        buf[p][1] = n1 + Nsurf;
        buf[p][2] = 9;
        p++;

        if(i > 0) {
            buf[p][0] = n1 + N5 + i;
            buf[p][1] = n1 + N5 + i + Nsurf;
            buf[p][2] = 9;
            p++;
        }

        buf[p][0] = n1;
        buf[p][1] = n1 + N5 + i + Nsurf;
        buf[p][2] = 11;
        p++;

        if((i & 1) == 0) {
            n2 = ((i - 1)*i)/2 + 1;

            if(i > 0) {
                buf[p][0] = n1;
                buf[p][1] = n2 + Nsurf;
                buf[p][2] = 11;
                p++;

                buf[p][0] = n1 + N5 + i;
                buf[p][1] = n2 + N5 + i - 1 + Nsurf;
                buf[p][2] = 11;
                p++;
            }

            if(i < M) {
                n2 = ((i + 1)*(i + 2))/2 + 1;

                buf[p][0] = n1;
                buf[p][1] = n2 + Nsurf;
                buf[p][2] = 11;
                p++;

                buf[p][0] = n1 + N5 + i;
                buf[p][1] = n2 + N5 + i + 1 + Nsurf;
                buf[p][2] = 11;
                p++;
            }
        }
    }
    return p;
}
```

```
/* GEN3STR.C - Generates the connectivity array for the three-spoke hybrid
 *          Pactruss.  Input data is read from XYZ.DAT.  The connectivity array
 *          is appended to the input nodal array and the combination is output
 *          to XYZ.DTA which is an ASCII file in the format needed for input to
 *          SEETRUSS.CMD.
 *
 *          The types of struts are:
 *
 *              Numbering
 *          External Internal
 *
 *             1        0   Non-folding Pactruss upper-surface struts
 *             2        1   Upper-surface Pactruss diagonals
 *             3        2   Upper-surface beam longerons and battens
 *             4        3   Upper-surface beam diagonals
 *             5        4   Non-folding Pactruss lower-surface struts
 *             6        5   Lower-surface Pactruss diagonals
 *             7        6   Lower-surface beam longerons and battens
 *             8        7   Lower-surface beam diagonals
 *             9        8   Pactruss verticals
 *             :        9   Beam verticals
 *             ;       10   Core Pactruss diagonals
 *             <       11   Beam core diagonals
 *
 *                                          John M. Hedgepeth       9/6/87
 *
 *      Updated to MS C 5.0                                         12/29/87
 *
 */

#include <scitech.h>
int M;
int Ntypes = 12;

int pacsurf(),paccore(),beamsurf(),beamcore();

main(argc,argv)
int argc;
char *argv[];
{
    int i,j,n,N,Nnodes,Nstruts,n1,n2,type;
    int Npacsurf,Npaccore,Nbeamsurf,Nbeamcore;
    int con1[100][3],con2[100][3],con3[100][3],con4[100][3];
    double x,y,z;
    char inname[50],outname[50];
    FILE *infile,*outfile;

    strcpy(inname,"TRIPAC.DAT");
    strcpy(outname,"TRIPAC.DTA");

    if(argc > 1) {
        strcpy(inname,argv[1]);
        strcpy(outname,argv[1]);
        strcat(inname,".DAT");
        strcat(outname,".DTA");
    }

    strupr(inname);
    strupr(outname);
```

```
if((infile = fopen(inname,"rt")) == NULL) {
    printf("\nCannot open %s.  Aborted.\n",inname);
    exit(1);
}
if(fscanf(infile," %d, %lf, %lf, %lf, %lf %d",
                                        &M,&x,&x,&x,&x,&Nnodes) != 6) {
    printf("\nCannot read input parameters.  Aborted.\n");
    exit(1);
}

if((outfile = fopen(outname,"r")) != NULL) {
    fflush(stdin);
    printf("\nFile %s exists.  Do you want to write over it? (Y/N) <N>",
                                                              outname);
    if(toupper(getch()) != 'Y')
        exit(1);

    fclose(outfile);
}
outfile = fopen(outname,"wt");

N = M *(M + 3) + 1;
Npacsurf = pacsurf(con1);
Npaccore = paccore(con2);
Nbeamsurf = beamsurf(con3);
Nbeamcore = beamcore(con4);
Nstruts = 3*(2*Npacsurf + Npaccore + 2*Nbeamsurf + Nbeamcore);

fprintf(outfile," %d, %d, %d\n",Nnodes,Nstruts,Ntypes);

for(i=0; i<Nnodes; i++) {
    if(fscanf(infile," %lf, %lf, %lf",&x,&y,&z) != 3) {
        printf("Corrupt data for node %d.  Aborted.\n",i);
        exit(1);
    }
    fprintf(outfile," %.8le, %.8le, %.8le\n",x,y,z);
}

fclose(infile);

for(j=0; j<Npacsurf; j++) {
    for(i=0; i<3; i++) {
        n1 = con1[j][0] + i*N;
        n2 = con1[j][1] + i*N;
        fprintf(outfile," %d, %d, %d\n",n1,n2,con1[j][2]);
        n1 += 3*N;
        n2 += 3*N;
        fprintf(outfile," %d, %d, %d\n",n1,n2,con1[j][2] + 4);
    }
}

for(j=0; j<Npaccore; j++) {
    for(i=0; i<3; i++) {
        n1 = con2[j][0] + i*N;
        n2 = con2[j][1] + i*N;
        fprintf(outfile," %d, %d, %d\n",n1,n2,con2[j][2]);
    }
}

for(j=0; j<Nbeamsurf; j++) {
```

```
        for(i=0; i<3; i++) {
            n1 = con3[j][0] + i*N;
            n2 = con3[j][1] + i*N;
            if(n1 > 3*N)
                n1 -= 3*N;
            if(n2 > 3*N)
                n2 -= 3*N;
            fprintf(outfile," %d, %d, %d\n",n1,n2,con3[j][2]);

            n1 += 3*N;
            n2 += 3*N;
            fprintf(outfile," %d, %d, %d\n",n1,n2,con3[j][2] + 4);
        }
    }

    for(j=0; j<Nbeamcore; j++) {
        for(i=0; i<3; i++) {
            n1 = con4[j][0] + i*N;
            n2 = con4[j][1] + i*N;
            if(n1 > 3*N)
                n1 -= 3*N;
            if(n2 > 6*N)
                n2 -= 3*N;
            fprintf(outfile," %d, %d, %d\n",n1,n2,con4[j][2]);
        }
    }

    fclose(outfile);

    printf("\n\nData file %s successfully written.\n",outname);
}


int pacsurf(buf)
int buf[100][3];
{
    int i,n,p,n1,n2;

    for(n=1,p=0; n <= M; n++) {
        for(i=1; i<=n; i++) {
            n1 = n*(n + 1) + 2*i + 1;
            n2 = (n - 1)*n + 2*i - 1;

            if(i < n) {
                buf[p][0] = n1;
                buf[p][1] = n2;
                buf[p][2] = 1;
                p++;
            }

            buf[p][0] = n1;
            buf[p][1] = n1 - 2;
            buf[p][2] = 0;
            p++;

            if(i == n) {
                buf[p][0] = n1;
                buf[p][1] = n1 - 1;
                buf[p][2] = 0;
                p++;
```

```
                buf[p][0] = nl;
                buf[p][1] = nl - 2*(n + 1);
                buf[p][2] = 1;
                p++;

                break;
            }

            buf[p][0] = nl;
            buf[p][1] = n2 + 2;
            buf[p][2] = 0;
            p++;

            buf[p][0] = nl + 1;
            buf[p][1] = nl - 1;
            buf[p][2] = 0;
            p++;

            buf[p][0] = nl + 1;
            if(n == i + 1)
                buf[p][1] = n2 + 2;
            else
                buf[p][1] = n2 + 3;
            buf[p][2] = 0;
            p++;

            buf[p][0] = nl + 1;
            buf[p][1] = n2 + 1;
            buf[p][2] = 1;
            p++;
        }
    }
    return p;
}


int paccore(buf)
int buf[100][3];
{
    int i,n,p,N,nl,n2;

    N = 3*(M*(M + 3) + 1);

    for(n=1,p=0; n<=M; n++) {
        for(i=0; i <= n; i++) {
            nl = n*(n + 1) + 2*i + 1;

            if(i>0) {
                buf[p][0] = nl;
                buf[p][1] = nl + N;
                buf[p][2] = 8;
                p++;

                if(i != n) {
                    buf[p][0] = nl + 1;
                    buf[p][1] = nl + 1 + N;
                    buf[p][2] = 8;
                    p++;
                }
```

```
}

if(((i + n) & 1) == 0) {
    if(i != n) {
        buf[p][0] = nl;
        buf[p][1] = nl + 2 + N;
        buf[p][2] = 10;
        p++;

        buf[p][0] = nl + 1;
        buf[p][1] = nl + 3 + N;
        buf[p][2] = 10;
        p++;

        if(i != 0) {
            buf[p][0] = nl;
            buf[p][1] = nl - 2 + N;
            buf[p][2] = 10;
            p++;

            buf[p][0] = nl + 1;
            buf[p][1] = nl - 1 + N;
            buf[p][2] = 10;
            p++;

            buf[p][0] = nl;
            buf[p][1] = nl - 2*n + N;
            buf[p][2] = 10;
            p++;

            buf[p][0] = nl + 1;
            buf[p][1] = nl + 1 - 2*n + N;
            buf[p][2] = 10;
            p++;

            if(n < M) {
                buf[p][0] = nl;
                buf[p][1] = nl + 2*(n + 1) + N;
                buf[p][2] = 10;
                p++;

                buf[p][0] = nl + 1;
                buf[p][1] = nl + 1 + 2*(n + 1) + N;
                buf[p][2] = 10;
                p++;
            }
        }
    }
    if(i == n) {
        buf[p][0] = nl;
        buf[p][1] = nl - 1 + N;
        buf[p][2] = 10;
        p++;

        buf[p][0] = nl;
        buf[p][1] = nl - 2 + N;
        buf[p][2] = 10;
        p++;

        if(n < M) {
```

```
                        buf[p][0] = nl;
                        buf[p][1] = nl + 2*(n + 1) + 1 + N;
                        buf[p][2] = 10;
                        p++;

                        buf[p][0] = nl;
                        buf[p][1] = nl + 2*(n + 1) + N;
                        buf[p][2] = 10;
                        p++;
                    }
                }
            }
        }
    }
    return p;
}


int beamsurf(buf)
int buf[100][3];
{
    int i,nl,n2,p,N;

    N = M*(M + 3) + 1;

    for(i=0,p=0; i<=M; i++) {
        nl = i*(i + 1) + 1;
        n2 = (i + 1)*(i + 2) + 1;

/* Beam battens
 */
        buf[p][0] = nl;
        if(i > 0)
            buf[p][1] = nl + 2*N + 1;
        else
            buf[p][1] = nl + 2*N;
        buf[p][2] = 2;
        p++;

        if(i == M)
            break;

/* Beam longerons
 */
        buf[p][0] = nl;
        buf[p][1] = n2;
        buf[p][2] = 2;
        p++;

        if(i > 0)
            buf[p][0] = nl + 2*N + 1;
        else
            buf[p][0] = nl + 2*N;
        buf[p][1] = n2 + 2*N + 1;
        buf[p][2] = 2;
        p++;

/* Beam surface diagonals
 */
        buf[p][0] = nl;
```

```
        buf[p][1] = n2 + 2*N + 1;
        buf[p][2] = 3;
        p++;

        if(i > 0)
            buf[p][0] = n1 + 2*N + 1;
        else
            buf[p][0] = n1 + 2*N;
        buf[p][1] = n2;
        buf[p][2] = 3;
        p++;
    }
    return p;
}



int beamcore(buf)
int buf[100][3];
{
    int i,n1,n2,p,N2,Nsurf;

    N2 = 2*(M*(M + 3) + 1);
    Nsurf = 3*N2/2;

    for(i=0,p=0; i<=M; i++) {
        n1 = i*(i + 1) + 1;

        buf[p][0] = n1;
        buf[p][1] = n1 + Nsurf;
        buf[p][2] = 9;
        p++;

        if(i > 0) {
            buf[p][0] = n1 + N2 + 1;
            buf[p][1] = n1 + N2 + 1 + Nsurf;
            buf[p][2] = 9;
            p++;
        }

        buf[p][0] = n1;
        if(i > 0)
            buf[p][1] = n1 + N2 + 1 + Nsurf;
        else
            buf[p][1] = n1 + N2 + Nsurf;
        buf[p][2] = 11;
        p++;

        if((i & 1) == 0) {
            n2 = (i - 1)*i + 1;

            if(i > 0) {
                buf[p][0] = n1;
                buf[p][1] = n2 + Nsurf;
                buf[p][2] = 11;
                p++;

                buf[p][0] = n1 + N2 + 1;
                buf[p][1] = n2 + N2 + 1 + Nsurf;
                buf[p][2] = 11;
```

```
                p++;
            }

            n2 = (i + 1)*(i + 2) + 1;

            if(i < M) {
                buf[p][0] = n1;
                buf[p][1] = n2 + Nsurf;
                buf[p][2] = l1;
                p++;

                if(i > 0)
                    buf[p][0] = n1 + N2 + 1;
                else
                    buf[p][0] = n1 + N2;
                buf[p][1] = n2 + N2 + 1 + Nsurf;
                buf[p][2] = l1;
                p++;
            }
        }
    }
    return p;
}
```

```
73, 258, 12
0.00000000e+000, 0.00000000e+000, 4.00000000e+000
9.99999534e-001, 5.77350000e-001, 8.33332556e-002
2.94193152e+000, 5.77350000e-001, 5.61768380e-001
1.97096553e+000, 2.25911243e+000, 5.61768380e-001
4.74468663e+000, 5.77350000e-001, 1.42783651e+000
3.85313753e+000, 2.22460999e+000, 1.23722240e+000
2.87234308e+000, 3.82034415e+000, 1.42783651e+000
1.12309793e-015, 1.15470000e+000, 8.33332556e-002
9.70965992e-001, 2.83646243e+000, 5.61768380e-001
-9.70965992e-001, 2.83646243e+000, 5.61768380e-001
1.87234355e+000, 4.39769415e+000, 1.42783651e+000
6.11750234e-015, 4.44921998e+000, 1.23722240e+000
-1.87234355e+000, 4.39769415e+000, 1.42783651e+000
-9.99999534e-001, 5.77350000e-001, 8.33332556e-002
-1.97096553e+000, 2.25911243e+000, 5.61768380e-001
-2.94193152e+000, 5.77350000e-001, 5.61768380e-001
-2.87234308e+000, 3.82034415e+000, 1.42783651e+000
-3.85313753e+000, 2.22460999e+000, 1.23722240e+000
-4.74468663e+000, 5.77350000e-001, 1.42783651e+000
-9.99999534e-001, -5.77350000e-001, 8.33332556e-002
-2.94193152e+000, -5.77350000e-001, 5.61768380e-001
-1.97096553e+000, -2.25911243e+000, 5.61768380e-001
-4.74468663e+000, -5.77350000e-001, 1.42783651e+000
-3.85313753e+000, -2.22460999e+000, 1.23722240e+000
-2.87234308e+000, -3.82034415e+000, 1.42783651e+000
-4.57397792e-015, -1.15470000e+000, 8.33332556e-002
-9.70965992e-001, -2.83646243e+000, 5.61768380e-001
9.70965992e-001, -2.83646243e+000, 5.61768380e-001
-1.87234355e+000, -4.39769415e+000, 1.42783651e+000
-1.94142662e-014, -4.44921998e+000, 1.23722240e+000
1.87234355e+000, -4.39769415e+000, 1.42783651e+000
9.99999534e-001, -5.77350000e-001, 8.33332556e-002
1.97096553e+000, -2.25911243e+000, 5.61768380e-001
2.94193152e+000, -5.77350000e-001, 5.61768380e-001
2.87234308e+000, -3.82034415e+000, 1.42783651e+000
3.85313753e+000, -2.22460999e+000, 1.23722240e+000
4.74468663e+000, -5.77350000e-001, 1.42783651e+000
9.99999534e-001, 5.77350000e-001, -1.91666674e+000
2.94193152e+000, 5.77350000e-001, -1.43823162e+000
1.97096553e+000, 2.25911243e+000, -1.43823162e+000
4.74468663e+000, 5.77350000e-001, -5.72163486e-001
3.85313753e+000, 2.22460999e+000, -7.62777599e-001
2.87234308e+000, 3.82034415e+000, -5.72163486e-001
1.12309793e-015, 1.15470000e+000, -1.91666674e+000
9.70965992e-001, 2.83646243e+000, -1.43823162e+000
-9.70965992e-001, 2.83646243e+000, -1.43823162e+000
1.87234355e+000, 4.39769415e+000, -5.72163486e-001
6.11750234e-015, 4.44921998e+000, -7.62777599e-001
-1.87234355e+000, 4.39769415e+000, -5.72163486e-001
-9.99999534e-001, 5.77350000e-001, -1.91666674e+000
-1.97096553e+000, 2.25911243e+000, -1.43823162e+000
-2.94193152e+000, 5.77350000e-001, -1.43823162e+000
-2.87234308e+000, 3.82034415e+000, -5.72163486e-001
-3.85313753e+000, 2.22460999e+000, -7.62777599e-001
-4.74468663e+000, 5.77350000e-001, -5.72163486e-001
-9.99999534e-001, -5.77350000e-001, -1.91666674e+000
-2.94193152e+000, -5.77350000e-001, -1.43823162e+000
```

```
-1.97096553e+000, -2.25911243e+000, -1.43823162e+000
-4.74468663e+000, -5.77350000e-001, -5.72163486e-001
-3.85313753e+000, -2.22460999e+000, -7.62777599e-001
-2.87234308e+000, -3.82034415e+000, -5.72163486e-001
-4.57397792e-015, -1.15470000e+000, -1.91666674e+000
-9.70965992e-001, -2.83646243e+000, -1.43823162e+000
9.70965992e-001, -2.83646243e+000, -1.43823162e+000
-1.87234355e+000, -4.39769415e+000, -5.72163486e-001
-1.94142662e-014, -4.44921998e+000, -7.62777599e-001
1.87234355e+000, -4.39769415e+000, -5.72163486e-001
9.99999534e-001, -5.77350000e-001, -1.91666674e+000
1.97096553e+000, -2.25911243e+000, -1.43823162e+000
2.94193152e+000, -5.77350000e-001, -1.43823162e+000
2.87234308e+000, -3.82034415e+000, -5.72163486e-001
3.85313753e+000, -2.22460999e+000, -7.62777599e-001
4.74468663e+000, -5.77350000e-001, -5.72163486e-001
3, 2, 1
39, 38, 5
9, 8, 1
45, 44, 5
15, 14, 1
51, 50, 5
21, 20, 1
57, 56, 5
27, 26, 1
63, 62, 5
33, 32, 1
69, 68, 5
5, 4, 1
41, 40, 5
11, 10, 1
47, 46, 5
17, 16, 1
53, 52, 5
23, 22, 1
59, 58, 5
29, 28, 1
65, 64, 5
35, 34, 1
71, 70, 5
5, 2, 0
41, 38, 4
11, 8, 0
47, 44, 4
17, 14, 0
53, 50, 4
23, 20, 0
59, 56, 4
29, 26, 0
65, 62, 4
35, 32, 0
71, 68, 4
5, 3, 0
41, 39, 4
11, 9, 0
47, 45, 4
17, 15, 0
53, 51, 4
23, 21, 0
59, 57, 4
```

29, 27, 0
65, 63, 4
35, 33, 0
71, 69, 4
6, 5, 1
42, 41, 5
12, 11, 1
48, 47, 5
18, 17, 1
54, 53, 5
24, 23, 1
60, 59, 5
30, 29, 1
66, 65, 5
36, 35, 1
72, 71, 5
5, 41, 8
11, 47, 8
17, 53, 8
23, 59, 8
29, 65, 8
35, 71, 8
5, 38, 10
11, 44, 10
17, 50, 10
23, 56, 10
29, 62, 10
35, 68, 10
5, 39, 10
11, 45, 10
17, 51, 10
23, 57, 10
29, 63, 10
35, 69, 10
1, 31, 2
37, 67, 6
7, 1, 2
43, 37, 6
13, 7, 2
49, 43, 6
19, 13, 2
55, 49, 6
25, 19, 2
61, 55, 6
31, 25, 2
67, 61, 6
1, 2, 2
37, 38, 6
7, 8, 2
43, 44, 6
13, 14, 2
49, 50, 6
19, 20, 2
55, 56, 6
25, 26, 2
61, 62, 6
31, 32, 2
67, 68, 6
31, 33, 2
67, 69, 6

```
1, 3, 2
37, 39, 6
7, 9, 2
43, 45, 6
13, 15, 2
49, 51, 6
19, 21, 2
55, 57, 6
25, 27, 2
61, 63, 6
1, 33, 3
37, 69, 7
7, 3, 3
43, 39, 7
13, 9, 3
49, 45, 7
19, 15, 3
55, 51, 7
25, 21, 3
61, 57, 7
31, 27, 3
67, 63, 7
2, 33, 2
38, 69, 6
8, 3, 2
44, 39, 6
14, 9, 2
50, 45, 6
20, 15, 2
56, 51, 6
26, 21, 2
62, 57, 6
32, 27, 2
68, 63, 6
2, 4, 2
38, 40, 6
8, 10, 2
44, 46, 6
14, 16, 2
50, 52, 6
20, 22, 2
56, 58, 6
26, 28, 2
62, 64, 6
32, 34, 2
68, 70, 6
33, 36, 2
69, 72, 6
3, 6, 2
39, 42, 6
9, 12, 2
45, 48, 6
15, 18, 2
51, 54, 6
21, 24, 2
57, 60, 6
27, 30, 2
63, 66, 6
2, 36, 3
38, 72, 7
```

8, 6, 3
44, 42, 7
14, 12, 3
50, 48, 7
20, 18, 3
56, 54, 7
26, 24, 3
62, 60, 7
32, 30, 3
68, 66, 7
4, 36, 2
40, 72, 6
10, 6, 2
46, 42, 6
16, 12, 2
52, 48, 6
22, 18, 2
58, 54, 6
28, 24, 2
64, 60, 6
34, 30, 2
70, 66, 6
1, 37, 9
7, 43, 9
13, 49, 9
19, 55, 9
25, 61, 9
31, 67, 9
1, 67, 11
7, 37, 11
13, 43, 11
19, 49, 11
25, 55, 11
31, 61, 11
1, 38, 11
7, 44, 11
13, 50, 11
19, 56, 11
25, 62, 11
31, 68, 11
31, 69, 11
1, 39, 11
7, 45, 11
13, 51, 11
19, 57, 11
25, 63, 11
2, 38, 9
8, 44, 9
14, 50, 9
20, 56, 9
26, 62, 9
32, 68, 9
33, 69, 9
3, 39, 9
9, 45, 9
15, 51, 9
21, 57, 9
27, 63, 9
2, 69, 11
8, 39, 11

14, 45, 11
20, 51, 11
26, 57, 11
32, 63, 11
4, 40, 9
10, 46, 9
16, 52, 9
22, 58, 9
28, 64, 9
34, 70, 9
36, 72, 9
6, 42, 9
12, 48, 9
18, 54, 9
24, 60, 9
30, 66, 9
4, 72, 11
10, 42, 11
16, 48, 11
22, 54, 11
28, 60, 11
34, 66, 11
4, 38, 11
10, 44, 11
16, 50, 11
22, 56, 11
28, 62, 11
34, 68, 11
36, 69, 11
6, 39, 11
12, 45, 11
18, 51, 11
24, 57, 11
30, 63, 11

```
  /* SEE2PAL.C - Reads geometry from XYZ.DTA (SEETRUSS format), strut cross
   *          section and material properties for each type of strut from file
   *          XYZ.PRP.  Produces file XYZ.TXT which is suitable for input into
   *          the MSC PAL2 finite element analysis program.
   *
   *          The file XYZ.PRP has a row, for each type, of the following quanti-
   *          ties separated by white space for each type:  type number (starting
   *          at 1), strut area A, Young's modulus E, material density rho, and
   *          the strut Euler buckling stress S_euler.  The material density is
   *          adjusted to include joint and payload masses.
   *
   *                                          John M. Hedgepeth       8/30/87
   *
   *              Updated to MS C 5.0                                 12/30/87
   *              Added thermal properties to material specs          5/11/88
   *
   *
   */

#include <scitech.h>

char *getdataline(FILE *);

main(argc,argv)
int argc;
char *argv[];
{
    int i,n,N,Ntypes,Nnodes,Nstruts,nl,n2,type,*p_strut,*ptr;
    double x,y,z,A,E,rho,S_euler;
    char dtaname[30],prpname[30],outname[30],name[30],*line;
    FILE *dtafile,*prpfile,*outfile;

    strcpy(dtaname,"SIXPAC.DTA");
    strcpy(prpname,"SIXPAC.PRP");
    strcpy(outname,"SIXPAC.TXT");
    strcpy(name,"SIXPAC");

    if(argc > 1) {
        strcpy(dtaname,argv[1]);
        strcpy(prpname,argv[1]);
        strcpy(outname,argv[1]);
        strcpy(name,argv[1]);

        strcat(dtaname,".DTA");
        strcat(prpname,".PRP");
        strcat(outname,".TXT");
    }

    strupr(dtaname);
    strupr(prpname);
    strupr(outname);
    strupr(name);

    if((dtafile = fopen(dtaname,"rt")) == NULL) {
        printf("\nCannot open %s.  Aborted.\n",dtaname);
        exit(1);
    }

    if((prpfile = fopen(prpname,"rt")) == NULL) {
        printf("\nCannot open %s.  Aborted.\n",prpname);
```

```
        exit(1);
    }

    line = getdataline(dtafile);
    if(sscanf(line," %d, %d, %d",&Nnodes,&Nstruts,&Ntypes) != 3) {
        printf("\nCannot read input parameters from %s.  Aborted.\n",dtaname);
        exit(1);
    }

/* Allocate space for strut connection array
 */
    if((p_strut = (int *)calloc(3*Nstruts,sizeof(int))) == NULL) {
        printf("\nCannot allocate space for strut array.  Aborted.\n");
        exit(1);
    }

    if((outfile = fopen(outname,"r")) != NULL) {
        fflush(stdin);
        printf("\nFile %s exists.  Do you want to write over it? (Y/N) <N>",
                                                                    outname);
        if(toupper(getch()) != 'Y')
            exit(1);

        fclose(outfile);
    }

    if((outfile = fopen(outname,"wt")) == NULL) {
        printf("\Cannot open %s.  Aborted.\n",outname);
        exit(1);
}

/* Start with truss name
 */
    fprintf(outfile,"TITLE %s TRUSS\n\n",name);

/* Discard first (zeroeth) node
 */
    line = getdataline(dtafile);
    if(sscanf(line," %lf, %lf, %lf",&x,&y,&z) != 3) {
        printf("\nCannot scan node 0.  Aborted.\n");
        exit(1);
    }


/* Transfer node coordinates
 */
    fprintf(outfile,"NODAL POINT LOCATIONS 1\n");

    for(n=1; n<Nnodes; n++) {
        line = getdataline(dtafile);
        if(sscanf(line," %lf, %lf, %lf",&x,&y,&z) != 3) {
            printf("\nCannot scan node %d.  Aborted.\n",n);
            exit(1);
        }

        fprintf(outfile,"%-3d  %12.6lf, %12.6lf, %12.6lf\n",n,x,y,z);
    }
    fprintf(outfile,"\n");

/* Discard rotation degrees of freedom
```

```
 */
    fprintf(outfile,"ZERO 1\nRA OF ALL\n\n");

/* Select type of mass lumping
 */
    fprintf(outfile,"LUMPED MASS TYPE 1\n");

/* Fill the strut connection array
 */
    ptr = p_strut;
    for(i=0; i<Nstruts; i++) {
        line = getdataline(dtafile);
        if(sscanf(line," %d, %d, %d",&n1,&n2,&type) != 3) {
            printf("\nCannot read data for strut %d.  Abort.\n",i);
            exit(1);
        }
        *ptr++ = type;
        *ptr++ = n1;
        *ptr++ = n2;
    }
    fclose(dtafile);

/* Transfer properties, connectivity, and ANALYZE for each type
 */
    fprintf(outfile,"C\nC  ********** Struts ***************\nC\n");
    fprintf(outfile,"\nANALYZE 0\n");

    N = Nnodes/12;

    for(n=0; n<Ntypes; n++) {
        line = getdataline(prpfile);
        if(sscanf(line," %d %lf %lf %lf %lf",&type,&A,&E,&rho,&S_euler)
                                                        != 5) {
            printf("\nCannot read properties for type %d.  Aborted.",n+1);
            exit(1)
            ;
        }

        fprintf(outfile,"\nC  ** Type %d **\n\n",type);
        fprintf(outfile,"MATERIAL PROPERTIES %lE, 0.0, %lf, 0.3, %lE, "
                                            "1.0, 0.0\n", E,rho,S_euler);
        fprintf(outfile,"BEAM TYPE 1,  %lE\n",A);

        ptr = p_strut;
        for(i=0; i<Nstruts; i++) {
            if(*ptr++ == n) {
                n1 = *ptr++;
                n2 = *ptr++;
                fprintf(outfile,"CONNECT %d TO %d\n",n1,n2);
            }
            else {
                ptr++;
                ptr++;
            }
        }
    }

    fprintf(outfile,"\END DEFINITION\n");

    fclose(prpfile);
```

```
    fclose(outfile);

    printf("\nPal2 input file %s successfully written. \n",outname);
}

/*************************************************************
 */
char *getdataline(FILE *infile)
{
    int chr,old;
    char *ptr;
    static char line[81];

    while(fgets(line,80,infile) != NULL) {
        ptr = line;
        while(isspace((chr = (int)*ptr)))
            ptr++;
        if(isdigit(chr) || chr == '.' || chr == '+' || chr == '-')
            return ptr;
    }
    if(feof(infile)) {
        printf("\007\nEnd of file reached.\n");
        return line;
    }
    else {
        printf("\007\nError in reading pal2 file.  Abort.\n");
        exit(1);
    }
}

/*************************************************************
 */
```

```
/* BESTFIT.C - Determines the location x0,y0,z0 of the vertex and the
 *          orientation angles phix,phiy of the best-fit paraboloid for
 *          a deformed surface.  The focal length of the paraboloid is held
 *          constant.  The parameters of the reference paraboloid and the
 *          coordinates of nodes on the undeformed surface are read from the
 *          file XYZ.DAT which was formed by the command
 *
 *                          genNpac.exe xyz     (N = 6 or 3)
 *
 *          The values of the deformations at each point are read from the
 *          file ABC.DEF which is obtained from the output of MSC Pal2 STAT2
 *          by using a text editor to extract the title and the list of dis-
 *          placements.
 *
 *          The method used is to find the values of x0,y0,z0,phix,phiy which
 *          minimize the sum of (Un cosalfa)*(Un cosalfa)over all the nodes,
 *          where
 *
 *              Un cosalfa = {uz - z0 + x*phiy -y*phix
 *                              - (ux - x0 - z*phiy)*x/(2*F)
 *                              - (uy - y0 + z*phix)*y/(2*F)}/{1 + z/(2*F)}
 *
 *          is one-half of the error in the reflected path length for the
 *          displaced paraboloid.   A weighting factor of cosalfa is used.
 *          The displacements x0 and y0 are constrained to be related to the
 *          rotations so that the focal point is stationary.  Thus
 *
 *              x0 = -F*phiy
 *              y0 =  F*phix
 *
 *          Results are reported to the screen as values of the paraboloid
 *          displacement and rotation as well as the rms value of the residual
 *          half-path-length error.
 *
 *                                                  j.m.hedgepeth   9/14/87
 *
 *      Updated for MS C 5.0                                        12/30/87
 */

#include <scitech.h>

int M;
double disp[5],F;

main()
{
    int i,j,m,flag,lpa[5];
    double x,y,z,*p_crd,*p_def,*p_real,c[6],d[6],Mat[3][4],w,W;
    FILE *datfile,*deffile;
    char datname[50],defname[50],linebuf[100],*p_chr,*n;
    char title[80];

    printf("\n\nBESTFIT reads input from xyz.DAT and abc.DEF.\n");
    printf("Enter pathname of .DAT file?  ");
    gets(datname);
    putchar('\n');

    if((n = strchr(datname,'.')) == NULL)
        strcat(datname,".DAT");
```

```
    if((datfile = fopen(datname,"rt")) == NULL) {
        printf("\nCannot open %s for reading.  Aborted.\n",datname);
        exit(1);
    }
    printf("Enter pathname of .DEF file?  ");
    gets(defname);
    putchar('\n');

    if((n = strchr(defname,'.')) == NULL)
        strcat(defname,".DEF");

    if((deffile = fopen(defname,"rt")) == NULL) {
        printf("\nCannot open %s for reading.  Aborted.\n",defname);
        exit(1);
    }

    if(fscanf(datfile," %d, %lf, %lf, %lf, %lf",&i,&F,&x,&y,&z) != 5) {
        printf("\nCannot read parameters from .DAT file.  Abort.\n");
        exit(1);
    }
    if(fscanf(datfile," %d",&M) != 1) {
        printf("\nCannot read number of nodal points.  Abort.\n");
        exit(1);
    }

/* Ignore first node (focal point) and divide M - 1 by 2 to use only upper
 * surface points.
 */
    M = (M - 1)/2;
    if(fscanf(datfile," %lf, %lf, %lf",&x,&y,&z) != 3) {
        printf("\nCannot scan coordinates of node 0.  Abort.\n");
        exit(1);
    }

/* Allocate space for arrays
 */
    if((p_crd = (double *)calloc(3*M,sizeof(double))) == NULL) {
        printf("\nCannot allocate memory for coordinate array.  Abort.\n");
        exit(1);
    }

    if((p_def = (double *)calloc(3*M,sizeof(double))) == NULL) {
        printf("\nCannot allocate memory for deflection array.  Abort.\n");
        exit(1);
    }

/* Fill coordinate array
 */
    p_real = p_crd;
    for(i=0; i<M; i++) {
        if(fscanf(datfile," %lf, %lf, %lf",&x,&y,&z) != 3) {
            printf("\nCannot scan coordinates for node %d.  Abort.\n",i+1);
            exit(1);
        }
        *p_real++ = x;
        *p_real++ = y;
        *p_real++ = z;
    }
    fclose(datfile);
```

```
/* Examine first lines of .DEF file until we reach deflection data.  Pick up
 * title on the way.
 */
/* Search each line for the title or for start of deflections
 */

    flag = 1;
    while(flag) {
        fgets(linebuf,100,deffile);
        if(sscanf(linebuf," %d %lf %lf %lf",&j,&x,&y,&z) == 4)
            break;

        p_chr = linebuf;
        while(*p_chr == ' ' || *p_chr == '\t' || *p_chr == '\n')
            p_chr++;
        if(*p_chr) {
            strcpy(title,p_chr);
            flag = 0;
        }
    }

    if(flag)
        strcpy(title,"  TRUSS");
    else {
        while(fgets(linebuf,100,deffile) != NULL) {
            if(sscanf(linebuf," %d %lf %lf %lf",&j,&x,&y,&z) == 4) {
                flag = 1;
                break;
            }
        }
    }
    if(flag == 0) {
        printf("\nError reading start of .DEF file.  Abort.\n");
        exit(1);
    }

/* Fill deflection array
 */

    p_real = p_def;

    for(i=0;i<j-1; i++) {
        *p_real++ = 0;
        *p_real++ = 0;
        *p_real++ = 0;
    }

    *p_real++ = x;
    *p_real++ = y;
    *p_real++ = z;
    i++;

    for(; i<M; i++) {
        if(fgets(linebuf,100,deffile) == NULL) {
            printf("\nCannot read .DEF file for node %d.  Abort.\n",i + 1);
            exit(1);
        }
        if(sscanf(linebuf," %d %lf %lf %lf",&j,&x,&y,&z) != 4) {
            printf("\nCannot scan deflections of node %d.  Abort.\n",i + 1);
            exit(1);
```

```
        }
        for(;i<j-1; i++) {
            *p_real++ = 0;
            *p_real++ = 0;
            *p_real++ = 0;
        }

        *p_real++ = x;
        *p_real++ = y;
        *p_real++ = z;
    }

    fclose(deffile);

/* Prepare to fill matrix
 */

    for(i=0; i<4; i++) {
        for(j=0; j<3; j++)
            Mat[j][i] = 0;
        d[i] = 0;
    }
    W = 0;

    for(m=0; m<M; m++) {
        x = *p_crd++;
        y = *p_crd++;
        z = *p_crd++;

        x /= 2.*F;
        y /= 2.*F;
        z /= 2.*F;

        w = 1./(1. + 2.*z);

        c[0] = x*w;
        c[1] = y*w;
        c[2] = -w;
        c[3] = -y;
        c[4] = x;

        w = sqrt(w);

        x = *p_def++;
        y = *p_def++;
        z = *p_def++;

        c[5] = c[0]*x +c[1]*y + c[2]*z;

        for(i=0; i<4; i++) {
            for(j=0; j<3; j++)
                Mat[j][i] += c[i + 2]*c[j + 2]*w;
            d[i] += c[i + 2]*c[5]*w;
        }
        W += w;
    }

    for(i=0; i<4; i++)
        d[i] /= W;
```

```
    w = .000001;

    dcrout(4,3,1,Mat,w,&W,&i,lpa);

    W = d[3];
    for(i=0; i<3; i++) {
        disp[i] = Mat[i][3];
        W -= d[i]*disp[i];
    }

    for(i=0; i<25; i++)
        putchar('\n');

    printf("           %s\n\n",title);
    printf("\tRms distortion from best-fit paraboloid   = %le\n\n",sqrt(W));
    printf("\tVertical displacement of paraboloid:   z0 = %le\n\n",disp[0]);
    printf("\tRotation of paraboloid about:     x axis  = %le\n",disp[1]/2./F);
    printf("\t                                  y axis  = %le\n",disp[2]/2./F);

    for(i=0; i<10; i++)
        putchar('\n');

    fflush(stdin);
    getch();
}
```

NASA National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-181747		

4. Title and Subtitle	5. Report Date
Pactruss Support Structure for Precision Segmented Reflectors	June 1989
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
John M. Hedgepeth	AAC-TN-1153
	10. Work Unit No.
	585-02-31-01

9. Performing Organization Name and Address	11. Contract or Grant No.
Astro Aerospace Corporation 6384 Via Real Carpinteria, CA 93013-2993	NAS1-17536, Task 9
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Contractor Report
NASA Langley Research Center Hampton, VA 23665-5225	14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Wilbur B. Fichter

16. Abstract

This report deals with the application of the Pactruss deployable structure to the support of large paraboloidal reflectors of very high precision. The Pactruss concept, originally conceived for the Space Station truss, is shown to be suitable for use in a triangular arrangement to support a reflector surface composed of hexagonal reflector panels. A hybrid of Pactruss structural and deployable single-fold beams is shown to accommodate a center body. A minor alteration in the geometry is shown to be necessary in order to avoid lock-up during deployment.

In order to assess the capability of the hybrid Pactruss structure, an example truss supporting a full-scale (20-meter-diameter) infra-red telescope is analyzed for static and dynamic performance. A truss structure weighing 800 kilograms is shown to give adequate support to a reflector surface weighing 3,000 kilograms.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Reflectors Antennas Synchronous deployment Large deployable reflector Truss Vibration	Unclassified - Unlimited Subject Category 39

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	75	A04