Table a39 **SUBROUTINE LAME, written by Charles Rankin.**
This user-written STAGS subroutine is for shell types that are not included in the STAGS "standard shell surfaces" list. **On the M-1 record if ISHELL = 1 (shell type no. 1) the shell surface is defined by SUBROUTINE LAME.** This particular version of SUBROUTINE LAME generates the "soccerball" spherical cap. See the first six shell units listed in Table a37 for a list of input data for a 180-degree "soccerball" spherical cap. This spherical cap has the same shape as Shell unit no. 1 of the 360-degree "eqellipse" model, but the finite element mesh differs. The singularity at the pole of the spherical cap in the 360-degree "eqellipse" model, in which polar coordinates are used, is avoided by the "soccerball" finite element grid. **As of this writing the thickness of the 180-degree "soccerball" spherical cap (Shell units 1 - 6) MUST BE UNIFORM.** See Figs. 169 and a2 and a3 for the finite element “soccerball” configuration. See Fig. a1 for the 360-degree polar coordinate configuration.

```
=====================================================================
#include "keydefs.h"
      subroutine LAME ( IUNIT, PROP, XYs, ISLAM )

*     ------------------------------------------------------------------
*     Given shell unit IUNIT & surface coordinates (Xs,Ys),  compute
*     branch coordinates (F,G,H) of the point.
*     GENERATES SOCCER BALL MESH using 3 Units
*
*     Compute the  First Fundamental Form,  or the derivative of the
*     position vector (F,G,H), as a function of Xs (Fx,Gx,Hx)  or of
*     Ys (Fy,Gy,Hy). Setting ISLAM = 1 aligns the local x axis along
*     the vector (Fx,Gx,Hx).   The z axis is always perpendicular to
*     both base vectors (Fx,Gx,Hx) and (Fy,Gy,Hy)
*
*     The PROP vector contains the following parameters, here:
*
*     PROP(1) = Radial coordinate (minimum) MUST BE ZERO!!
*     PROP(2) = Radial coordinate (maximum) 0<PROP(2)<=90. (degrees)
*               This coordinate applies to all THREE units: maximum
*               for the assemblage.
*     PROP(3) = Hoop coordinate (minimum--degrees)
*     PROP(4) = Hoop coordinate (maximum--degrees)
*               Note: Either Prop(3) = Prop(6) OR
*                            Prop(4) = Prop(7) -- BUT NOT BOTH
*                     Either Prop(3) = (Prop(6)+Prop(7))/2 or
*                            Prop(4) = (Prop(6)+Prop(7))/2
*     PROP(5) = Radius
*     PROP(6) = Hoop coordinate for COMBINED 3 Units,
*               (minimum--degrees)
*     PROP(7) = Hoop coordinate for COMBINED 3 Units,
```

```
*                (maximum--degrees)
*
*
*     -----------------------------------------------------------------

      _implicit_none_
#include "pie.h"

      Integer IUNIT
      Real    PROP(*)
      Real    XYs(2)
      Integer ISLAM
      _float_ xx,dx,yy,dy
      _float_ a,b,d1,xp,yp,zp
      _float_ z0p

      _float_ sn,cs,ar
      _float_ xpx,ypx,xpp,ypp,zx,zy,zpx,zpp
      _float_ rm,rh,tn
      _float_ one,two,four,ninety,ft5
      _float_ xus(4),yus(4),shp(4)
      _float_ csdy, sndy, shp1,shp2,shp3,shp4

      shp1(xx,yy)=(1.-xx)*(1.-yy)
      shp2(xx,yy)=(1.-xx)*yy
      shp3(xx,yy)=xx*yy
      shp4(xx,yy)=xx*(1.-yy)

#include "lamex.h"

      a=Prop(5)    !  RADIUS
      b=a*sin(dtr*Prop(2))  ! Radius of opening at base

      z0p= a**2-b**2  !  Maximum "Z" offset
      ar=b*.4   !  Ratio of soccerball "square" to total meridional span
      ft5=(Prop(7)-Prop(6))*.5  ! Half the included angle

*     Rescale X coordinate to lie between 0 and 1:

      dx=prop(2)
      xx=xys(1)/dx

*     Rescale Y coordinate to lie between 0 and 1:

      dy=Prop(4)-Prop(3)

      sndy=sin(dtr*(Prop(7)-Prop(6)))
      csdy=cos(dtr*(Prop(7)-Prop(6)))
```

```
      yy=(XYs(2)-Prop(3))/dy

*     Compute Global Coordinates

      cs = cos(dtr*xys(2))
      sn = sin(dtr*xys(2))
      islam=2
      if (Prop(3).eq.Prop(6) .and. Prop(4).eq.Prop(7)) then

*     Define 4 local points:

         xus(1)=0.
         yus(1)=0.
         xus(2)=ar*csdy
         yus(2)=ar*sndy
         xus(3)=ar*(1.+csdy)
         yus(3)=ar*sndy
         xus(4)=ar
         yus(4)=0.

         xp=shp1(xx,yy)*xus(1)+shp2(xx,yy)*xus(2) +
     &        shp3(xx,yy)*xus(3)+shp4(xx,yy)*xus(4)
         yp=shp1(xx,yy)*yus(1)+shp2(xx,yy)*yus(2) +
     &        shp3(xx,yy)*yus(3)+shp4(xx,yy)*yus(4)
         xpx =
     &      (1.-yy)*(xus(4)-xus(1)) + yy*(xus(3)-xus(2))
         ypx =
     &      (1.-yy)*(yus(4)-yus(1)) + yy*(yus(3)-yus(2))
         xpp =
     &      (1.-xx)*(xus(2)-xus(1)) + xx*(xus(3)-xus(4))
         ypp =
     &      (1.-xx)*(yus(2)-yus(1)) + xx*(yus(3)-yus(4))
         if(xx.le.1.e-5) then
            islam=2
         else
            islam=1
         endif


      else if (Prop(3).lt.ft5 ) then

         xus(1)=ar
         yus(1)=0
         xus(2)=ar*(1.+csdy)
         yus(2)=ar*sndy
         xus(3)=(1.-yy)*xus(1)+yy*xus(2)
         yus(3)=(1.-yy)*yus(1)+yy*yus(2)
         xp = b*xx*cs + (1.-xx)*xus(3)
```

```fortran
          yp = b*xx*sn + (1.-xx)*yus(3)


*debug      write (6,*) 'xp,yp,zp = ',xp,yp,zp

*     Compute derivatives (for First Fundamental Form)
*     ===============================================

          xpx = b*cs-xus(3)
          ypx = b*sn-yus(3)
          xpp = -b*sn*xys(1)*dtr
     &        +(1.-xx)*(xus(2)-xus(1))/dy
          ypp = b*xys(1)*cs*dtr
     &        +(1.-xx)*(yus(2)-yus(1))/dy
          islam=1


      elseif (Prop(4).gt.ft5) then


          xus(1)=ar*(1.+csdy)
          yus(1)=ar*sndy
          xus(2)=ar*csdy
          yus(2)=ar*sndy
          xus(3)=(1.-yy)*xus(1)+yy*xus(2)
          yus(3)=(1.-yy)*yus(1)+yy*yus(2)
          xp = b*xx*cs + (1.-xx)*xus(3)
          yp = b*xx*sn + (1.-xx)*yus(3)

*debug      write (6,*) 'xp,yp,zp = ',xp,yp,zp

*     Compute derivatives (for First Fundamental Form)
*     ===============================================

          xpx = b*cs-xus(3)
          ypx = b*sn-yus(3)
          xpp = -b*sn*xx*dtr
     &        +(1.-xx)*(xus(2)-xus(1))/dy
          ypp = b*xx*cs*dtr
     &        +(1.-xx)*(yus(2)-yus(1))/dy
          islam=1
      endif


      zp = -z0p+sqrt(abs(a**2-xp**2-yp**2))


*     Set STAGS System Coordinates
```

```
*     ++++++++++++++++++++++++++++

      ff = xp
      gg = yp
      hh = zp


      d1 = zp+z0p
      zx  = -xp/d1
      zy  = -yp/d1

      zpx = zx*xpx+zy*ypx
      zpp = zx*xpp+zy*ypp

*     Set STAGS system variables with derivative information
*     ==========================================================

      fx = xpx
      gx = ypx
      hx = zpx

      fy = xpp
      gy = ypp
      hy = zpp

      return

      end
=================================================================
```