The Use of Geometry from within the Engineering Sketch Pad – Rev 1.26
The EGADS API

Bob Haimes
haimes@mit.edu
Aerospace Computational Design Lab
Department of Aeronautics & Astronautics
Massachusetts Institute of Technology

Overview

- Background & ESP
- Objects
  - Geometry
  - Topology
  - Tessellation
- Meshing
- Programming Examples
- Closing Remarks

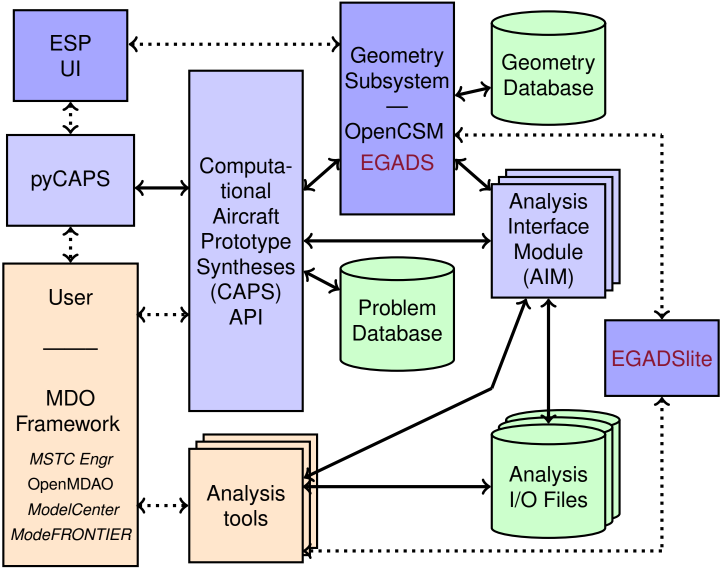
ESP **is:**

- a parametric geometry creation and manipulation system designed to **fully support** the analysis and design of aerospace vehicles (**aCAD**)
- a stand-alone system for the development of geometric models
- can be embedded into other software systems to support their geometric and process needs

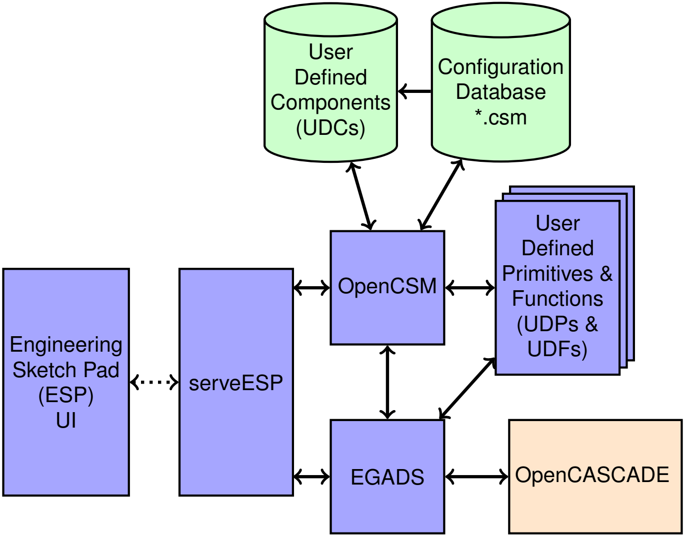
ESP **is not:**

- a full-featured mechanical computer-aided design (**mCAD**) system
- a system to be used for creating “drawings”
- an MDO Framework (but can support them)

Engineering Sketch Pad (ESP)



ESP's Geometry Subsystem Architecture



EGADS Overview

The Engineering Geometry Aircraft Design System (EGADS) is an open-source geometry interface to OpenCASCADE

- reduces OpenCASCADE's 17,000 methods to about ~100 calls
    - Supports C, C++ & FORTRAN
- provides *bottom-up* and/or *top-down* construction
- geometric primitives
    - curve: line, circle, ellipse, parabola, hyperbola, offset, bezier, BSpline (including NURBS)
    - surface: plane, spherical, conical, cylindrical, toroidal, revolution, extrusion, offset, bezier, BSpline (including NURBS)
- solid creation and Boolean operations (*top-down*)
- provides persistent user-defined attributes on topological entities
- adjustable tessellator (*vs* a surface mesher) with support for finite-differencing in the calculation of parametric sensitivities

EGADS Overview

System Support

- Mac OSX with **clang**, **ifort** and/or **gfortran**
- LINUX with **gcc**, **ifort** and/or **gfortran**
- Windows with Microsoft Visual Studio C++ and **ifort**
- No globals (but not entirely thread-safe due to OpenCASCADE)
- Various levels of output (0-none, through 3-debug)
- Written in C and C++
- pyEGADS only requires a current version of Python

EGADS Objects (**egos**)

- Pointer to a C structure – allows for an *Object-based* API
- Treated as “blind” pointers (i.e., not meant to be dereferenced)
- **ego**s are INTEGER*8 variables in FORTRAN

EGADSlite – for HPC Environments

- No construction supported
- Same API and Object model as EGADS
    - Can use EGADS to prototype/build EGADSlite code
- Suitable for an MPI setup:
    - Data export from EGADS via a *stream*
    - Data import to EGADSlite from the *stream*
    - *Stream* setup to Broadcast (or write to disk)
- ANSI C – No OpenCASCADE
- Tiny memory footprint
- Thread safe and scalable
    - EGADS' OpenCASCADE evaluation functions replaced with those written for EGADSlite
- See $ESP_ROOT/externApps/Pagoda/EGADSserver for an MPI example

EGADS Objects – egos

- Context – Holds the *globals*
- Transform
- Tessellation
- Nil (allocated but not assigned) – internal
- Empty – internal
- Reference – internal
- Geometry
  - pcurve, curve, surface
- Topology
  - Node, Edge, Loop, Face, Shell, Body, Model
- *Effective Topology*
  - EEdge, ELoop, EFace, EShell, EBody

See `$ESP_ROOT/include/egadsTypes.h` for a list of `define`s

EGADS Objects – Attribution

- Attributes – metadata consisting of name/value pairs
    - Unique name – no spaces
    - A single type: Integer, Real, String, CSys, Pointer (not persistent)
    - A length (for Integers & Reals)
- Objects
    - Any (non-internal) Object can have multiple Attributes
    - Only Attributes on Topological Objects are copied and are persistent (saved)
- SBO & Intersection Functions
    - Unmodified Topological Objects maintain their Attributes
    - Face Attributes are carried through to the resultant fragments
    - All other Attributes may be lost
- CSys Attributes are modified through Transformations

EGADS Geometry Objects

surface

- 3D surfaces of 2 parameters $[u, v]$
- Types: Plane, Spherical, Cylindrical, Revolution, Toriodal, Trimmed, Bezier, BSpline, Offset, Conical, Extrusion
- All types abstracted to $[x, y, z] = f(u, v)$

pcurve – Parameter Space Curves

- 2D curves in the Parametric space $[u, v]$ of a surface
- Types: Line, Circle, Ellipse, Parabola, Hyperbola, Trimmed, Bezier, BSpline, Offset
- All types abstracted to $[u, v] = h(t)$

curve

- 3D curve – single running parameter (t)
- Same types as pcurve but abstracted to $[x, y, z] = g(t)$

Boundary Representation – BRep

Topological Entity	Geometric Entity	Function
Model		
Body	Solid, Sheet, Wire	
Shell		
Face	**surface**	$(x, y, z) = \mathbf{f}(u, v)$
Loop		
Edge	**curve**	$(x, y, z) = \mathbf{g}(t)$
Node	**point**	

Top Down ↓

↑ *Bottom Up*

- Nodes that bound Edges may not be on underlying curves
- Edges in the Loops that trim the Face may not sit on the surface hence the use of pcurves

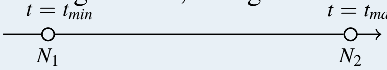
EGADS Topology Objects

Node

- Contains $[x, y, z]$
- Types: none

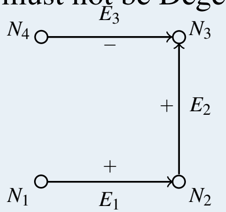
Edge

- Has a 3D curve (if not Degenerate)
- Has a t range (t_{min} to t_{max}, where $t_{min} < t_{max}$)
  Note: The positive orientation is going from t_{min} to t_{max}
- Has a Node for t_{min} and for t_{max} – can be the same Node
- Types:
  - OneNode – periodic
  - TwoNode – normal
  - Degenerate – single Node, t range used for the associated pcurve

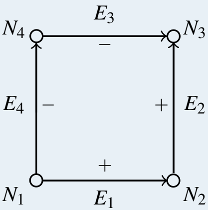


Loop – without a reference surface

1. Free standing connected Edges that can be used in a non-manifold setting (for example in WireBodies)
2. A list of connected Edges associated with a Plane (which does not require pcurves)
   - An ordered collection of Edge objects with associated senses
   - No Edges must not be Degenerate
   - Types:



Open: $+E_1$ $+E_2$ $-E_3$



Closed: $+E_1$ $+E_2$ $-E_3$ $-E_4$

Loop – with a reference surface

1. Collections of Edges followed by a corresponding collection of pcurves that define the $[u, v]$ trimming on the surface
   - An ordered collection of Edge objects with associated senses
   - Degenerate Edges are required when the $[u, v]$ mapping collapses like at the apex of a cone (note that the pcurve is needed to be fully defined using the Edge's t range)
   - Trims the surface by maintaining material to the left of the running Loop
   - An Edge may be found in a Loop twice (with opposite senses) and with different pcurves.
   - Types: Open or Closed (comes back on itself)

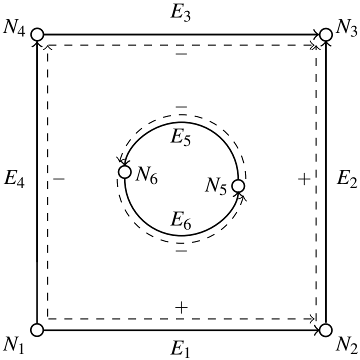
EGADS Topology Objects

Face

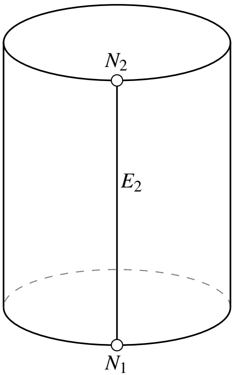
- A surface bounded by one or more Loops with associated senses
- Only one outer Loop (sense = 1) and any number of inner Loops (sense = -1). Note that under very rare conditions a Loop may be found in more than 1 Face – in this case the one marked with sense = +/- 2 must be used in a reverse manner.
- All Loops must be Closed
- Loop(s) must not contain reference geometry for Planar surfaces
- If the surface is not a Plane then the Loop's reference Object must match that of the Face
- Type is the orientation of the Face based on surface's $U \otimes V$:
    - SFORWARD or SREVERSE when the orientations are opposed

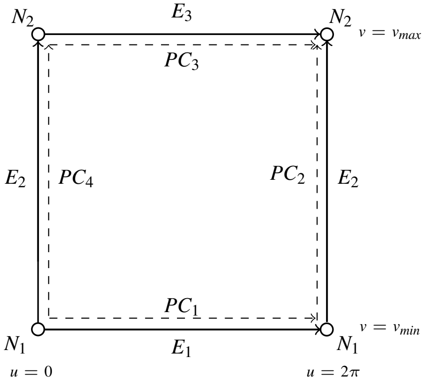
Note that this is coupled with the Loop's orientation (i.e. an outer Loop traverses the Face in a right-handed manner defining the outward direction)

EGADS Topology Objects – Face Examples



- Outer Loop – right handed/counterclockwise: $+E_1$ $+E_2$ $-E_3$ $-E_4$
- Inner Loop – left handed/clockwise: $-E_5$ $-E_6$

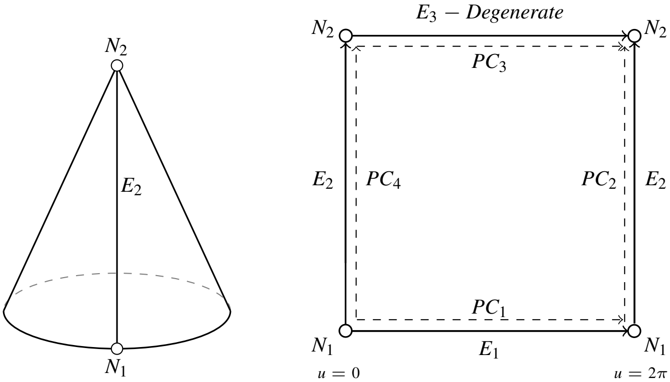




Unrolled periodic cylinder Face

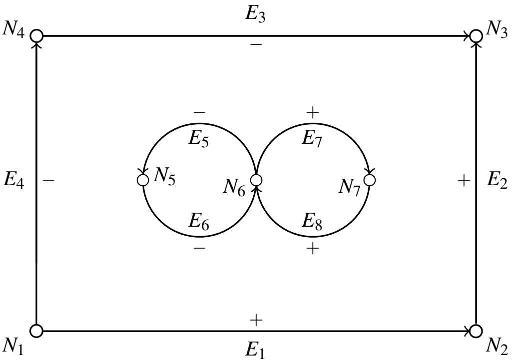
Single Outer Loop – right handed/counterclockwise:

$+E_1$ $+E_2$ $-E_3$ $-E_2$

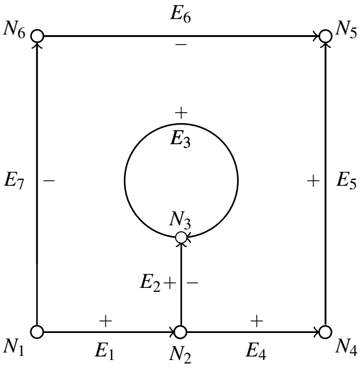


Unrolled Cone

EGADS Topology Objects – Face Examples



- Outer Loop – right handed/counterclockwise: $+E_1$ $+E_2$ $-E_3$ $-E_4$
- Inner Loop #1 – left handed/clockwise: $-E_5$ $-E_6$
- Inner Loop #2 – left handed/clockwise: $+E_7$ $+E_8$



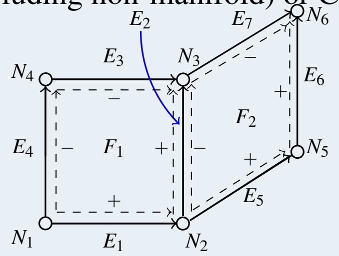
Single Outer Loop – right handed/counterclockwise:

$+E_1$ $+E_2$ $+E_3$ $-E_2$ $+E_4$ $+E_5$ $-E_6$ $-E_7$

Note: PCurve the same for both sides of E_2

Shell

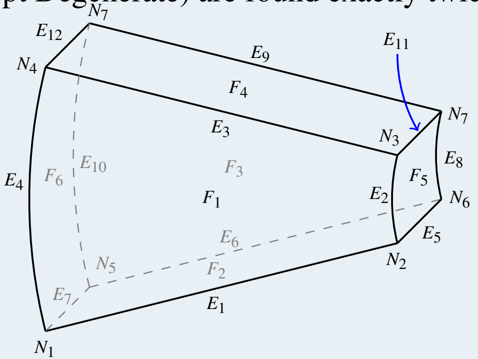
- A collection of one or more connected Faces that if Closed segregates regions of 3-Space
- All Faces must be properly oriented
- Non-manifold Shells can have more than 2 Faces sharing an Edge
- Types: Open (including non-manifold) or Closed



Face #1 Loop: $+E_1$ $+E_2$ $-E_3$ $-E_4$
Face #2 Loop: $+E_5$ $+E_6$ $-E_7$ $-E_2$

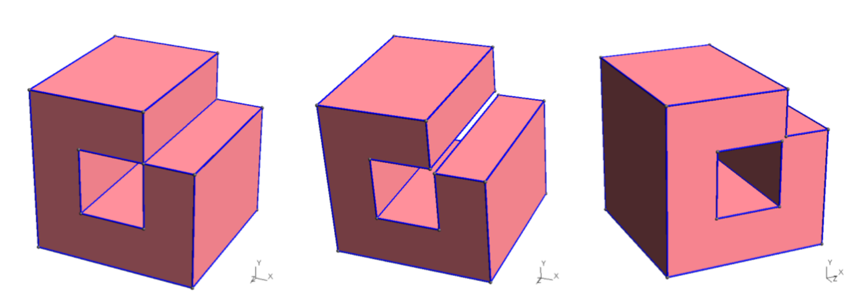
SolidBody

- Manifold collection of one or more Closed Shells
- One outer Shell (sense = 1); any number of inner (sense = -1)
- Edges (except Degenerate) are found exactly twice (sense = ± 1)



Simple SolidBody: 8 Nodes, 12 Edges, 6 Loops and 6 Faces

Manifold vs. Nonmanifold



nonmanifold manifold manifold

figure stolen from “An introduction to Geometrical Modelling and Mesh Generation: The Gmsh Companion” by Christophe Geuzaine, Emilie Marchandise & Jean-François Remacle – used without permission!

Can the geometry be manufactured?

EGADS Topology Objects

Body – including SolidBody

- Container used to aggregate Topology
- Connected to support non-manifold collections at the Model level
- *Owns* all the Objects contained within
- Types:
  - A WireBody contains a single Loop
  - A FaceBody contains a single Face – IGES import
  - A SheetBody contains a single Shell which can be either non-manifold or manifold (though usually a manifold Body of this type is promoted to a SolidBody)

Model

- A collection of Bodies – becomes the *Owner* of contained Objects
- Returned by SBO & Sew Functions
- Read and Written by EGADS

Helper Functions

- makeLoop
  - Connects unrelated (via Nodes) Edges from a list
  - Uses input tolerance to match entities
  - Result may be multiple Loops
- makeFace
  - From Closed Planar Loop
  - From surface with limits
- sewFaces
  - Connects Faces with unrelated Topology
  - Uses input tolerance to match entities
  - Returns a Model – may have multiple Bodies
  - Can connect in a nonmanifold manner

EGADS Tessellation Objects

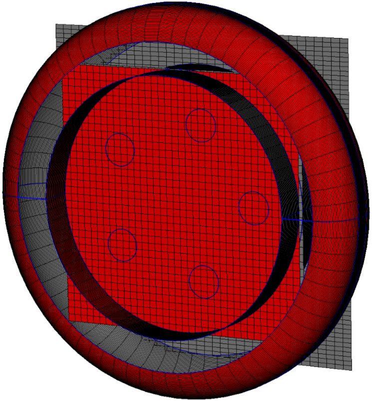
Geometry

- Unconnected discretization of a range of the Object
  - Polyline for curves at constant t increments
  - Regular grid for surfaces at constant increments (isoclines)

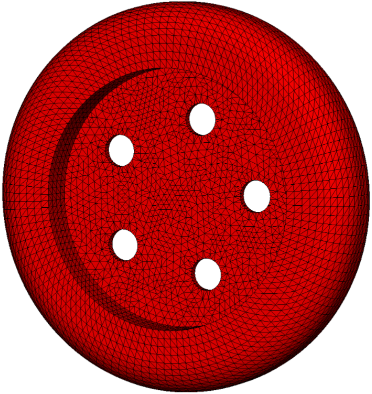
Body Topology

- Connected and trimmed tessellation including:
  - Polyline for Edges
  - Triangulation for Faces
  - Optional Quadrilateral Patching for Faces
- Ownership and Geometric Parameters for Vertices
- Adjustable parameters for side length and curvature (x2)
- Watertight
- Exposed per Face/Edge or Global indexing

EGADS Tessellation Objects



from $ESP_ROOT/bin/vGeom



from $ESP_ROOT/bin/vTess

The EGADS/EGADSlite API

- Function names begin with "EG_" – or –
  "IG_" for the FORTRAN bindings
- Functions almost always return an integer *error code*
- *Object-based* – procedural, usually with the first argument an **ego**
- Signatures usually have the inputs first, then output argument(s)
- Some outputs may be pointers to lists of *things*
  `EG_free` needs to be used when marked as "freeable"
- **ego**s have:
  - *Owner*: Context, Body, or Model
  - Reference Objects (objects they depend upon)
- When a Body is made, all included Objects are copied – not referenced

See `$ESP_ROOT/doc/EGADS/egads.pdf` for a detailed description of all of the functions.
See `$ESP_ROOT/include/egads.h` for a complete listing of the functions.
See `$ESP_ROOT/include/egadsErrors.h` for a list of the return code `define`s.

Deleting Objects

- Use the function `EG_deleteObject` (or `ig_deleteobject`)
  `EGADSlite` has a limited ability to delete Objects
- The Object must be reference *free* – i.e. not used by another
  - Delete in the opposite order of creation
  - If in a Body, delete the Body (unless the Body is in a Model)
- `EG_deleteObject` on a Context does not delete the Context
  - Deletes all Objects in the Context that are not in a Body
  - Use `EG_close` to delete all objects in a Context (and the Context)

Another Rule

- A Body can only be in one Model
  - Copy the Body of interest, then include the copy in the new Model

Meshing Considerations

Point Queries – Evaluations

- Derivatives – `EGADS/EGADSlite` provides 1^{st} and 2^{nd}
  - Curves: at $t \Rightarrow X$, $\frac{dX}{dt}$ and $\frac{d^2X}{dt^2}$
    tangency & curvature*
  - Surfaces: at $[u, v] \Rightarrow X$, $\frac{dX}{du}$, $\frac{dX}{dv}$, $\frac{d^2X}{du^2}$, $\frac{d^2X}{dv^2}$ and $\frac{d^2X}{dudv}$
    normal $(\frac{dX}{du} \otimes \frac{dX}{dv})^\dagger$ & curvature*
- Continuity – Derivative unavailable for less than C^2
  - Degenerate points (Poles of a sphere, Apex of a cone)
  - BSpline/NURBS with *multiplicity of knots*
  - Appropriate derivatives are returned as 0.0

* Note: Returns Radius of Curvature(s) and the associated direction(s)
† Note: The Face normal may be opposite that of the surface

Meshing Considerations

Point Queries – Inverse Evaluations

pcurve: $t = \mathbf{h}^{-1}(u, v)$
curve/Edge: $t = \mathbf{g}^{-1}(x, y, z)$
surface/Face: $[u, v] = \mathbf{f}^{-1}(x, y, z)$

Accomplished with 1^{st} and 2^{nd} derivatives from Evaluation

- Optimization (Newton-Raphson)
  - Minimize distance to requested position – needs start location not projection in a particular direction
  - stopping criteria – only as accurate as this ϵ
  - needs clear line-of-site to target (wing near TE on wrong side)
  - can get stuck in local minima
  - periodicity – result may need to be adjusted by the period
- Costly when robust (requires many seed locations)
- When invoked with Edge/Face will limit to result to the bounds

Meshing Considerations

Point Queries – Contained within Predicates

Answers the question: Is the specified location in this entity?

Useful meshing queries:

- Is this $[u, v]$ in the Face (`EG_inFace`)?
  - Is this $[u, v]$ in the Face's valid parametric box?
  - Is this $[u, v]$ trimmed away or in a hole?
- Is this X contained within the *Solid* (`EG_inTopology`)?
  - Performed by ray-casting and counting crossings
  - Can be expensive
- Ambiguous (within the tolerance and) near bounds

It may be better to answer these queries in a discrete setting

Meshing Considerations

Surface Degeneracies

If the surface meshing algorithm has smoothness assumptions, then knowledge of degenerate locations is critical!

- Degenerate points (Poles of a sphere, Apex of a cone)
  - You can depend on these points being Nodes
  - Degenerate Edges mark these locations
  - Zero (or close) derivatives are returned for an Evaluation
- BSpline/NURBS with *multiplicity of knots* — C^1 or C^0
  - Much harder to deal with!
  - Will probably also have Edge curves with *kinks*
  - `EG_loadModel` has a flag that splits Faces/Edges at C^1 and/or C^0 locations (does not change the geometry) and places *kinks* at topological bounds
  - If you are constructing the geometry – **DON'T DO THIS!**

Watertight BRep Meshing

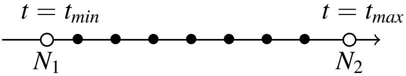
Dimensional Strategy

First mesh all of the Edges and then the Faces:

- Set the boundary vertices from lower in the topological hierarchy
- Mesh the interior
- Assume that C^0 locations are at bounds

Discretizing Edges

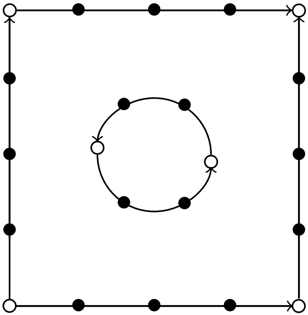
- Set the start and end Node positions (can be the same)
- Evaluate at t_{min} & t_{max} and compare to Node positions gets local (endpoint) tolerances
- Use a scheme to distribute vertices along the Edge (t or other) do not add vertices at the ends within the Node's local tolerance



Watertight BRep Meshing

Discretizing Faces

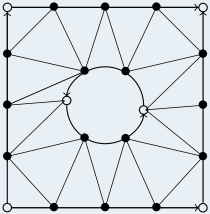
- Foreach Loop that bounds the Face:
  - Collect the Edge discretization *tail to head-1* or *head to tail-1* based on the Edge's sense in the Loop
  - Use the Edge vertex t value `EG_getEdgeUV` to get $[u, v]$
  - Makes a 2D closed set of line segments & 3D bounds for the Face



Watertight BRep Meshing

Discretizing Faces – continued

- Any closed 2D region can be triangulated
  No additional vertices are needed



- Use a scheme to enhance the initial triangulation
  - Query in X and insert in $[u, v]$
  - When a triangle side in on an Edge/Node care should be taken:
    Compare X with the surface/Face evaluation at $[u, v]$
    Do not add vertices within that local tolerance

Programming Example – Cart3D Tri File (1)

```
#include "egads.h"
#include <math.h>
#include <string.h>

#ifdef WIN32
#define snprintf _snprintf
#endif

int main(int argc, char *argv[])
{
  int          i, j, k, status, oclass, mtype, nbody, nvert, ntriang, nface;
  int          plen, tlen, pty, pin, tin[3], *senses;
  const int    *ptype, *pindex, *tris, *tric;
  char         filename[20];
  const char   *OCCrev;
  float        arg;
  double       params[3], box[6], size, verts[3];
  const double *points, *uv;
  FILE         *fp;
  ego          context, model, geom, solid, *bodies, tess, *dum, *faces;

  if ((argc != 2) && (argc != 5)) {
    printf(" Usage: egads2tri Model [angle relSide relSag]\n\n");
    return 1;
  }

  /* look at EGADS revision */
  EG_revision(&i, &j, &OCCrev);
  printf("\n Using EGADS %2d.%02d with %s\n\n", i, j, OCCrev);
```

Programming Example – Cart3D Tri File (2)

```
/* initialize */
status = EG_open(&context);
if (status != EGADS_SUCCESS) {
  printf(" EG_open = %d!\n\n", status);
  return 1;
}
status = EG_loadModel(context, 0, argv[1], &model);
if (status != EGADS_SUCCESS) {
  printf(" EG_loadModel = %d\n\n", status);
  return 1;
}
status = EG_getBoundingBox(model, box);
if (status != EGADS_SUCCESS) {
  printf(" EG_getBoundingBox = %d\n\n", status);
  return 1;
}
size = sqrt((box[0]-box[3])*(box[0]-box[3]) + (box[1]-box[4])*(box[1]-box[4]) +
            (box[2]-box[5])*(box[2]-box[5]));

/* get all bodies */
status = EG_getTopology(model, &geom, &oclass, &mtype, NULL, &nbody,
                        &bodies, &senses);
if (status != EGADS_SUCCESS) {
  printf(" EG_getTopology = %d\n\n", status);
  return 1;
}
```

Programming Example – Cart3D Tri File (3)

```
params[0] =  0.025*size;
params[1] =  0.001*size;
params[2] = 15.0;
if (argc == 5) {
  sscanf(argv[2], "%f", &arg);
  params[2] = arg;
  sscanf(argv[3], "%f", &arg);
  params[0] = arg;
  sscanf(argv[4], "%f", &arg);
  params[1] = arg;
  printf(" Using angle = %lf, relSide = %lf, relSag = %lf\n", params[2], params[0], params[1]);
  params[0] *= size;
  params[1] *= size;
}
 printf(" Number of Bodies = %d\n\n", nbody);

/* write out each body as a different Cart3D ASCII tri file */
for (i = 0; i < nbody; i++) {
  snprintf(filename, 20, "egads.%3.3d.a.tri", i+1);
  solid = bodies[i];
  mtype = 0;
  EG_getTopology(bodies[i], &geom, &oclass, &mtype, NULL, &j, &dum, &senses);
  if (mtype == SHEETBODY) {
    status = EG_makeTopology(context, NULL, BODY, SOLIDBODY, NULL, j, dum, NULL, &solid);
    if (status == EGADS_SUCCESS) {
      printf(" SheetBody %d promoted to SolidBody\n", i);
      mtype = SOLIDBODY;
    } else {
      printf(" SheetBody %d cannot be promoted to SolidBody\n", i);
    }
  }
  if (mtype != SOLIDBODY) continue;   /* only Solid Bodies! */
```

Programming Example – Cart3D Tri File (4)

```
status = EG_makeTessBody(solid, params, &tess);
if (status != EGADS_SUCCESS) {
  printf(" EG_makeTessBody %d = %d\n", i, status);
  if (solid != bodies[i]) EG_deleteObject(solid);
  continue;
}
status = EG_getBodyTopos(solid, NULL, FACE, &nface, &faces);
if (status != EGADS_SUCCESS) {
  printf(" EG_getBodyTopos %d = %d\n", i, status);
  if (solid != bodies[i]) EG_deleteObject(solid);
  EG_deleteObject(tess);
  continue;
}
EG_free(faces);

/* get counts */
status = EG_statusTessBody(tess, &geom, &j, &nvert);
printf(" statusTessBody = %d %d  npts = %d\n", status, j, nvert);
if (status != EGADS_SUCCESS) continue;
ntriang = 0;
for (j = 0; j < nface; j++) {
  status = EG_getTessFace(tess, j+1, &plen, &points, &uv, &ptype, &pindex,
                          &tlen, &tris, &tric);
  if (status != EGADS_SUCCESS) {
    printf(" Error: EG_getTessFace %d/%d = %d\n", j+1, nvert, status);
    continue;
  }
  ntriang += tlen;
}
```

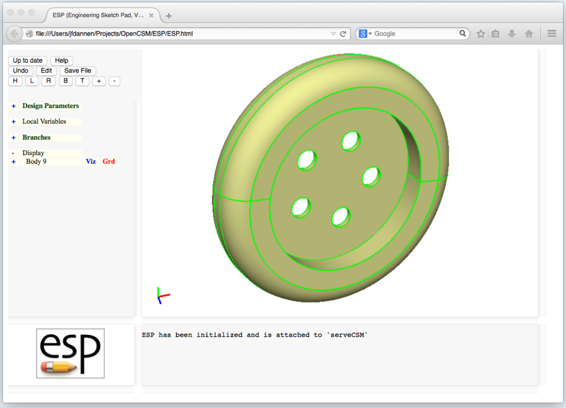
Programming Example – Cart3D Tri File (5)

```
/* write it out */
fp = fopen(filename, "w");
if (fp == NULL) {
  printf(" Can not Open file %s! NO FILE WRITTEN\n", filename);
  if (solid != bodies[i]) EG_deleteObject(solid);
  continue;
}
printf("\nWriting Cart3D component tri file %s\n", filename);
/* header */
fprintf(fp, "%d  %d\n", nvert, ntriang);
/* ...vertList     */
for (j = 0; j < nvert; j++) {
  status = EG_getGlobal(tess, j+1, &pty, &pin, verts);
  if (status != EGADS_SUCCESS)
    printf(" Error: EG_getGlobal %d/%d = %d\n", j+1, nvert, status);
  fprintf(fp, " %20.13le %20.13le %20.13le\n", verts[0], verts[1], verts[2]);
}
/* ...Connectivity */
for (j = 0; j < nface; j++) {
  status = EG_getTessFace(tess, j+1, &plen, &points, &uv, &ptype, &pindex,
                          &tlen, &tris, &tric);
  if (status != EGADS_SUCCESS) continue;
  for (k = 0; k < tlen; k++) {
    status = EG_localToGlobal(tess, j+1, tris[3*k  ], &tin[0]);
    if (status != EGADS_SUCCESS)
      printf(" Error: EG_localToGlobal %d/%d = %d\n", j+1, tris[3*k  ], status);
    status = EG_localToGlobal(tess, j+1, tris[3*k+1], &tin[1]);
    if (status != EGADS_SUCCESS)
      printf(" Error: EG_localToGlobal %d/%d = %d\n", j+1, tris[3*k+1], status);
```

```
        status = EG_localToGlobal(tess, j+1, tris[3*k+2], &tin[2]);

        if (status != EGADS_SUCCESS)
          printf(" Error: EG_localToGlobal %d/%d = %d\n", j+1, tris[3*k+2], status);
        fprintf(fp, "%6d %6d %6d\n",tin[0], tin[1], tin[2]);
      }
    }
    /* ...Component list*/
    for (j = 0; j < ntriang; j++) fprintf(fp, "%6d\n", 1);
    fclose(fp);

    if (solid != bodies[i]) EG_deleteObject(solid);

  }

  /* cleanup and close */

  status = EG_deleteObject(tess);

  if (status != EGADS_SUCCESS) printf(" EG_deleteObject tess  = %d\n", status);

  status = EG_deleteObject(model);

  if (status != EGADS_SUCCESS) printf(" EG_deleteObject model = %d\n", status);

  EG_close(context);


  return 0;
}
```

Programming Example – Parametric Tire (1)

Example of both *Bottom Up* and *Top Down* Construction

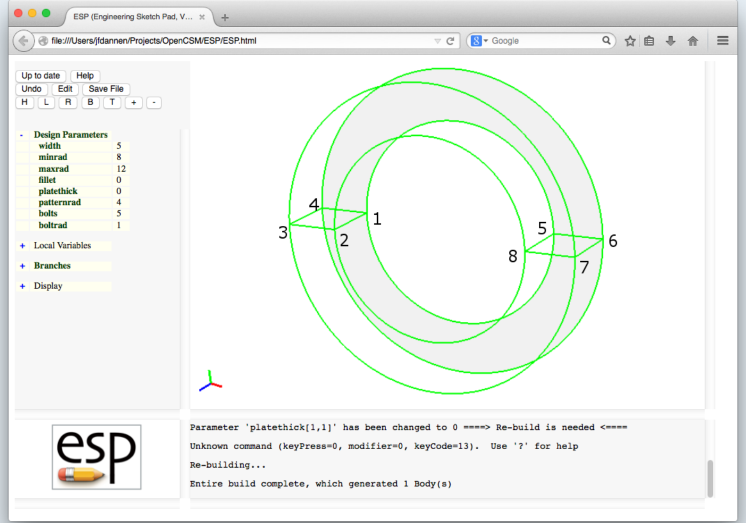


Name	Description
`width`	width
`minrad`	minimum radius
`maxrad`	maximum radius
`fillet`	fillet radius at outside
`thick`	wheel thickness
`bolts`	number of bolt holes
`crad`	radius of bolt circle
`brad`	radius of bolt hole

Programming Example – Parametric Tire (2)

```
#define TWOPI    6.2831853071795862319959269

  /* declarations */
  int     status = EGADS_SUCCESS;
  int     sense[20], oclass, mtype, nchild, *senses, i;
  int     bolts;
  double  width, minrad, maxrad, fillet, thick, crad, brad;
  double  node1[3], node2[3], node3[3], node4[3], node5[3], node6[3], node7[3], node8[3];
  double  cent1[3], cent2[3], axis1[3], axis2[3], axis3[3];
  double  data[18], trange[2];
  ego     context, enodes[8], ecurve[16], eedges[16], eloop, efaces[8], eshell;
  ego     esurface[4], epcurve[4], ebody1, ebody2, ebody3, ebody4;
  ego     elist[20], emodel, *echilds2, source, *echilds, eref, ebody;

 /* set the parameter values */
 width  =  5.0;
 minrad =  8.0;
 maxrad = 12.0;
 fillet =  2.0;
 thick  =  0.5;
 bolts  =  5;
 crad   =  5.0;
 brad   =  1.0;
```

Locate and Make the Nodes

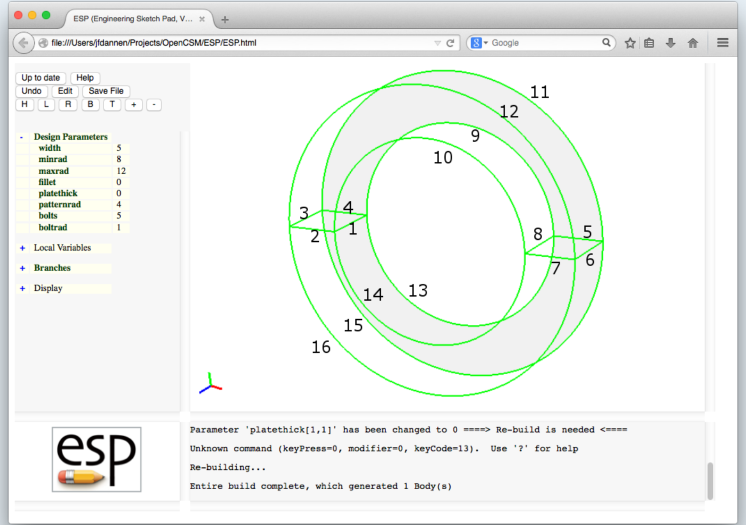


Programming Example – Parametric Tire (4)

```
/* define a context */
status = EG_open(&context);
printf("EG_open -> status=%d\n", status);
if (status < EGADS_SUCCESS) exit(1);

/* Node locations */
node1[0] = -minrad;  node1[1] = 0.0;  node1[2] = -width / 2.0;
node2[0] = -minrad;  node2[1] = 0.0;  node2[2] =  width / 2.0;
node3[0] = -maxrad;  node3[1] = 0.0;  node3[2] =  width / 2.0;
node4[0] = -maxrad;  node4[1] = 0.0;  node4[2] = -width / 2.0;
node5[0] =  minrad;  node5[1] = 0.0;  node5[2] = -width / 2.0;
node6[0] =  maxrad;  node6[1] = 0.0;  node6[2] = -width / 2.0;
node7[0] =  maxrad;  node7[1] = 0.0;  node7[2] =  width / 2.0;
node8[0] =  minrad;  node8[1] = 0.0;  node8[2] =  width / 2.0;

/* make the Nodes */
status = EG_makeTopology(context, NULL, NODE, 0, node1, 0, NULL, NULL, &enodes[0]);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_makeTopology(context, NULL, NODE, 0, node2, 0, NULL, NULL, &enodes[1]);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_makeTopology(context, NULL, NODE, 0, node3, 0, NULL, NULL, &enodes[2]);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_makeTopology(context, NULL, NODE, 0, node4, 0, NULL, NULL, &enodes[3]);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_makeTopology(context, NULL, NODE, 0, node5, 0, NULL, NULL, &enodes[4]);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_makeTopology(context, NULL, NODE, 0, node6, 0, NULL, NULL, &enodes[5]);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_makeTopology(context, NULL, NODE, 0, node7, 0, NULL, NULL, &enodes[6]);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_makeTopology(context, NULL, NODE, 0, node8, 0, NULL, NULL, &enodes[7]);
if (status != EGADS\_SUCCESS) goto cleanup;
```

Programming Example – Parametric Tire (5)

Locate and Make the Edges



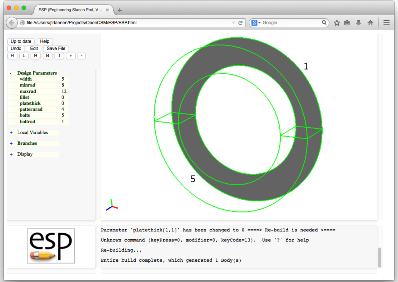
Programming Example – Parametric Tire (6)

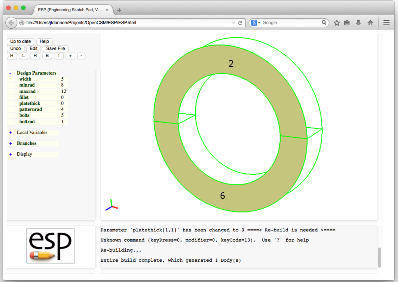
```
/* make (linear) Edge 1 */
data[0] = node1[0];
data[1] = node1[1];
data[2] = node1[2];
data[3] = node2[0] - node1[0];
data[4] = node2[1] - node1[1];
data[5] = node2[2] - node1[2];

status  = EG_makeGeometry(context, CURVE, LINE, NULL, NULL, data, &ecurve[0]);

if (status != EGADS_SUCCESS) goto cleanup;


status = EG_invEvaluate(ecurve[0], node1, &trange[0], data);

if (status != EGADS_SUCCESS) goto cleanup;

status = EG_invEvaluate(ecurve[0], node2, &trange[1], data);

if (status != EGADS_SUCCESS) goto cleanup;

elist[0] = enodes[0];
elist[1] = enodes[1];

status   = EG_makeTopology(context, ecurve[0], EDGE, TWONODE, trange, 2, elist, NULL,
                           &eedges[0]);

if (status != EGADS_SUCCESS) goto cleanup;
```

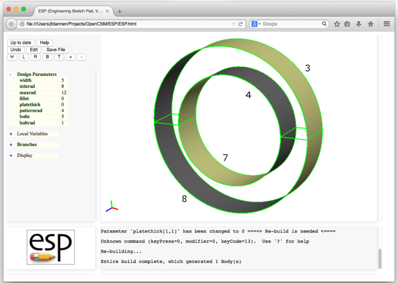
Programming Example – Parametric Tire (7)

```
/* data used in creating the arcs */
axis1[0] = 1;   axis1[1] = 0;   axis1[2] = 0;
axis2[0] = 0;   axis2[1] = 1;   axis2[2] = 0;
axis3[0] = 0;   axis3[1] = 0;   axis3[2] = 1;
cent1[0] = 0;   cent1[1] = 0;   cent1[2] = -width / 2;
cent2[0] = 0;   cent2[1] = 0;   cent2[2] =  width / 2;

/* make (circular) Edge 9 */
data[0] = cent1[0];   data[1] = cent1[1];   data[2] = cent1[2];
data[3] = axis1[0];   data[4] = axis1[1];   data[5] = axis1[2];
data[6] = axis2[0];   data[7] = axis2[1];   data[8] = axis2[2];   data[9] = minrad;

status = EG_makeGeometry(context, CURVE, CIRCLE, NULL, NULL, data, &ecurve[8]);
if (status != EGADS_SUCCESS) goto cleanup;


status = EG_invEvaluate(ecurve[8], node5, &trange[0], data);
if (status != EGADS_SUCCESS) goto cleanup;

status = EG_invEvaluate(ecurve[8], node1, &trange[1], data);
if (status != EGADS_SUCCESS) goto cleanup;
if (trange[0] > trange[1]) trange[1] += TWOPI;   /* ensure trange[1] > trange[0] */

elist[0] = enodes[4];
elist[1] = enodes[0];
status   = EG_makeTopology(context, ecurve[8], EDGE, TWONODE, trange, 2, elist, NULL,
                           &eedges[8]);

if (status != EGADS_SUCCESS) goto cleanup;
```

Programming Example – Parametric Tire (8)







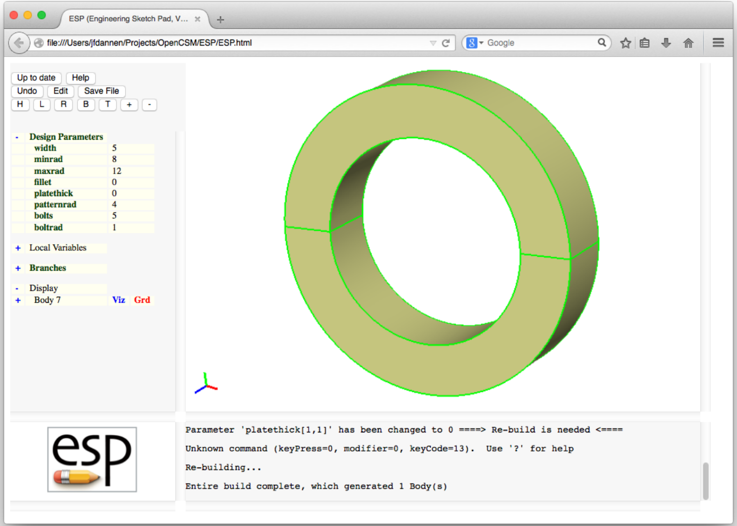
Programming Example – Parametric Tire (9)

```
/* make the outer cylindrical surface */
data[0] = cent1[0];   data[1]  = cent1[1];   data[2]  = cent1[2];
data[3] = axis1[0];   data[4]  = axis1[1];   data[5]  = axis1[2];
data[6] = axis2[0];   data[7]  = axis2[1];   data[8]  = axis2[2];
data[9] = axis3[0];   data[10] = axis3[1];   data[11] = axis3[2];   data[12] = maxrad;
status  = EG_makeGeometry(context, SURFACE, CYLINDRICAL, NULL, NULL, data, &esurface[0]);
if (status != EGADS_SUCCESS) goto cleanup;

/* make the inner cylindrical surface */
data[0] = cent1[0];   data[1]  = cent1[1];   data[2]  = cent1[2];
data[3] = axis1[0];   data[4]  = axis1[1];   data[5]  = axis1[2];
data[6] = axis2[0];   data[7]  = axis2[1];   data[8]  = axis2[2];
data[9] = axis3[0];   data[10] = axis3[1];   data[11] = axis3[2];   data[12] = minrad;
status  = EG_makeGeometry(context, SURFACE, CYLINDRICAL, NULL, NULL, data, &esurface[1]);
if (status != EGADS_SUCCESS) goto cleanup;

/* make (planar) Face 1 */
sense[0] = SFORWARD;   sense[1] = SREVERSE;   sense[2] = SFORWARD;   sense[3] = SFORWARD;
elist[0] = eedges[3];  elist[1] = eedges[8];  elist[2] = eedges[4];  elist[3] = eedges[10];
status   = EG_makeTopology(context, NULL, LOOP, CLOSED, NULL, 4, elist, sense, &eloop);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_makeFace(eloop, SFORWARD, NULL, &efaces[0]);
if (status != EGADS_SUCCESS) goto cleanup;
```

Programming Example – Parametric Tire (10)

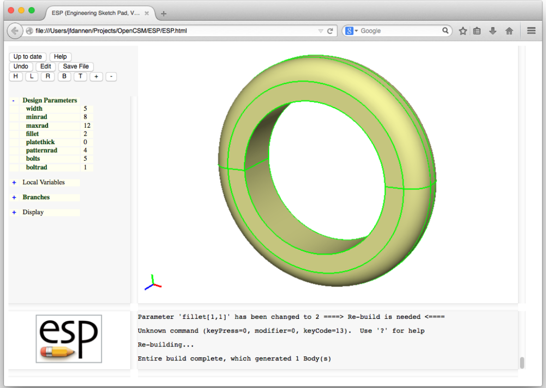
```
/* make (cylindrical) Face 3 */
status = EG_otherCurve(esurface[0], ecurve[2], 0, &epcurve[0]);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_otherCurve(esurface[0], ecurve[10], 0, &epcurve[1]);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_otherCurve(esurface[0], ecurve[5], 0, &epcurve[2]);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_otherCurve(esurface[0], ecurve[11], 0, &epcurve[3]);
if (status != EGADS_SUCCESS) goto cleanup;

sense[0] = SFORWARD;   sense[1] = SREVERSE;   sense[2] = SFORWARD;   sense[3] = SFORWARD;
elist[0] = eedges[2];  elist[1] = eedges[10]; elist[2] = eedges[5];  elist[3] = eedges[11];
elist[4] = epcurve[0]; elist[5] = epcurve[1]; elist[6] = epcurve[2]; elist[7] = epcurve[3];
status = EG_makeTopology(context, esurface[0], LOOP, CLOSED, NULL, 4, elist, sense, &eloop);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_makeTopology(context, esurface[0], FACE, SREVERSE, NULL, 1, &eloop, sense,
                         &efaces[2]);
if (status != EGADS_SUCCESS) goto cleanup;
:
:

 /* make the shell and initial Body */
status = EG_makeTopology(context, NULL, SHELL, CLOSED, NULL, 8, efaces, NULL, &eshell);
if (status != EGADS_SUCCESS) goto cleanup;
status = EG_makeTopology(context, NULL, BODY, SOLIDBODY, NULL, 1, &eshell, NULL, &ebody1);
if (status != EGADS_SUCCESS) goto cleanup;
```

Complete *Bottom UP* build



Programming Example – Parametric Tire (12)

```
/* add fillets if desired (result is ebody2) */
if (fillet > 0.0) {
  elist[0] = eedges[10]; elist[1] = eedges[11]; elist[2] = eedges[14]; elist[3] = eedges[15];
  status = EG_filletBody(ebody1, 4, elist, fillet, &ebody2, NULL);
  if (status != EGADS_SUCCESS) goto cleanup;
  status = EG_deleteObject(ebody1);
  if (status != EGADS_SUCCESS) goto cleanup;
} else {
  ebody2 = ebody1;
}
```



Programming Example – Parametric Tire (13)

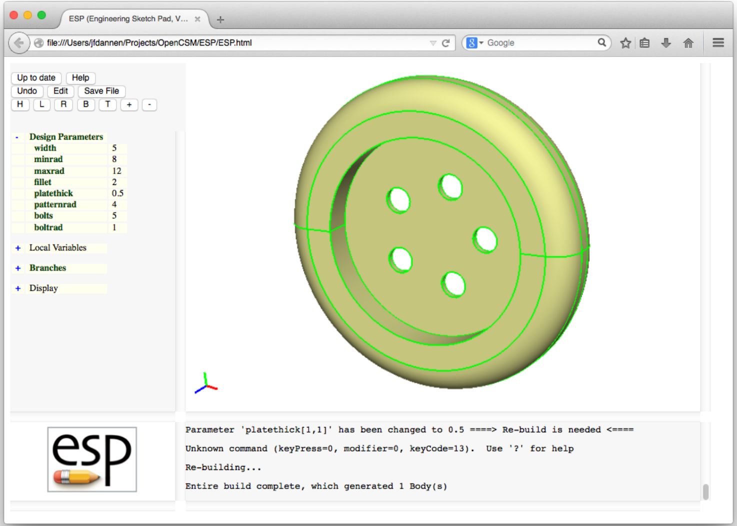
```
if (thick > 0.0) {
  data[0] = 0;   data[1] = 0;   data[2] =  thick / 2;
  data[3] = 0;   data[4] = 0;   data[5] = -thick / 2;
  data[6] = (minrad + maxrad) / 2;
  status = EG_makeSolidBody(context, CYLINDER, data, &ebody3);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_generalBoolean(ebody2, ebody3, FUSION, 0.0, &emodel);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_deleteObject(ebody2);
  if (status != EGADS_SUCCESS) goto cleanup;
  status = EG_deleteObject(ebody3);
  if (status != EGADS_SUCCESS) goto cleanup;
  status = EG_getTopology(emodel, &eref, &oclass, &mtype, data, &nchild, &echilds, &senses);
  if (status != EGADS_SUCCESS) goto cleanup;

  if (oclass != MODEL || nchild != 1) {
    printf("No model or are returning more than one body ochild = %d, nchild = %d/n",
           oclass, nchild);
    status = -999;
    goto cleanup;
  }

  status = EG_copyObject(echilds[0], NULL, &source);
  if (status != EGADS_SUCCESS) goto cleanup;
  status = EG_deleteObject(emodel);
  if (status != EGADS_SUCCESS) goto cleanup;
```

```
/* add bolt holes */
for (i = 0; i < bolts; i++) {
  data[0] = crad * cos(i * (TWOPI / bolts));
  data[1] = crad * sin(i * (TWOPI / bolts));
  data[2] =  thick / 2.0;
  data[3] = crad * cos(i * (TWOPI / bolts));
  data[4] = crad * sin(i * (TWOPI / bolts));
  data[5] = -thick / 2.0;
  data[6] = brad;

  status = EG_makeSolidBody(context, CYLINDER, data, &ebody4);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_generalBoolean(source, ebody4, SUBTRACTION, 0.0, &emodel);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_deleteObject(source);
  if (status != EGADS_SUCCESS) goto cleanup;
  status = EG_deleteObject(ebody4);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_getTopology(emodel, &eref, &oclass, &mtype, data, &nchild, &echilds2,
                          &senses);
  if (status != EGADS_SUCCESS) goto cleanup;

  if (oclass != MODEL || nchild != 1) {
     printf("Not a model or are returning more than one body ochild = %d, nchild = %d/n",
            oclass, nchild);
     status = -999;
     goto cleanup;
  }
```

Programming Example – Parametric Tire (15)

```
        status = EG_copyObject(echilds2[0], NULL, &source);
        if (status != EGADS_SUCCESS) goto cleanup;

        status = EG_deleteObject(emodel);
        if (status != EGADS_SUCCESS) goto cleanup;
      }
      ebody = source;
  } else {
      ebody = ebody2;
  }

  /* make and dump the model */
  status = EG_makeTopology(context, NULL, MODEL, 0, NULL, 1, &ebody, NULL, &emodel);
  printf("EG_makeTopology -> status=%d\n", status);
  if (status != EGADS_SUCCESS) goto cleanup;

  status = EG_saveModel(emodel, "tire.egads");
  printf("EG_saveModel -> status=%d\n", status);

  /* cleanup */
  status = EG_deleteObject(emodel);
  printf("EG_close -> status=%d\n", status);

cleanup:
  status = EG_close(context);
  printf("EG_close -> status=%d\n", status);
  return 0;
}
```

Programming Example – Parametric Tire (16)



Closing Remarks

- The source to these programming examples can be found at:
  `$ESP_ROOT/doc/EGADS/Tutorial`
- Other EGADS examples can be found in the ESP Distribution:
  `$ESP_ROOT/src/EGADS/examples`
- OpenCSM
  - The parametric *engine* on top of EGADS (like SolidWorks is to Parasolid)
  - Basically *Top Down* but can support *Bottom Up* builds by UDPs
  - Tire UDP – `$ESP_ROOT/doc/UDP_UDF/data/udpTire.c`
- A note on OpenCASCADE:
  Though EGADS was originally a *thin* layer over OpenCASCADE many of the methods that did not work well have been replaced.