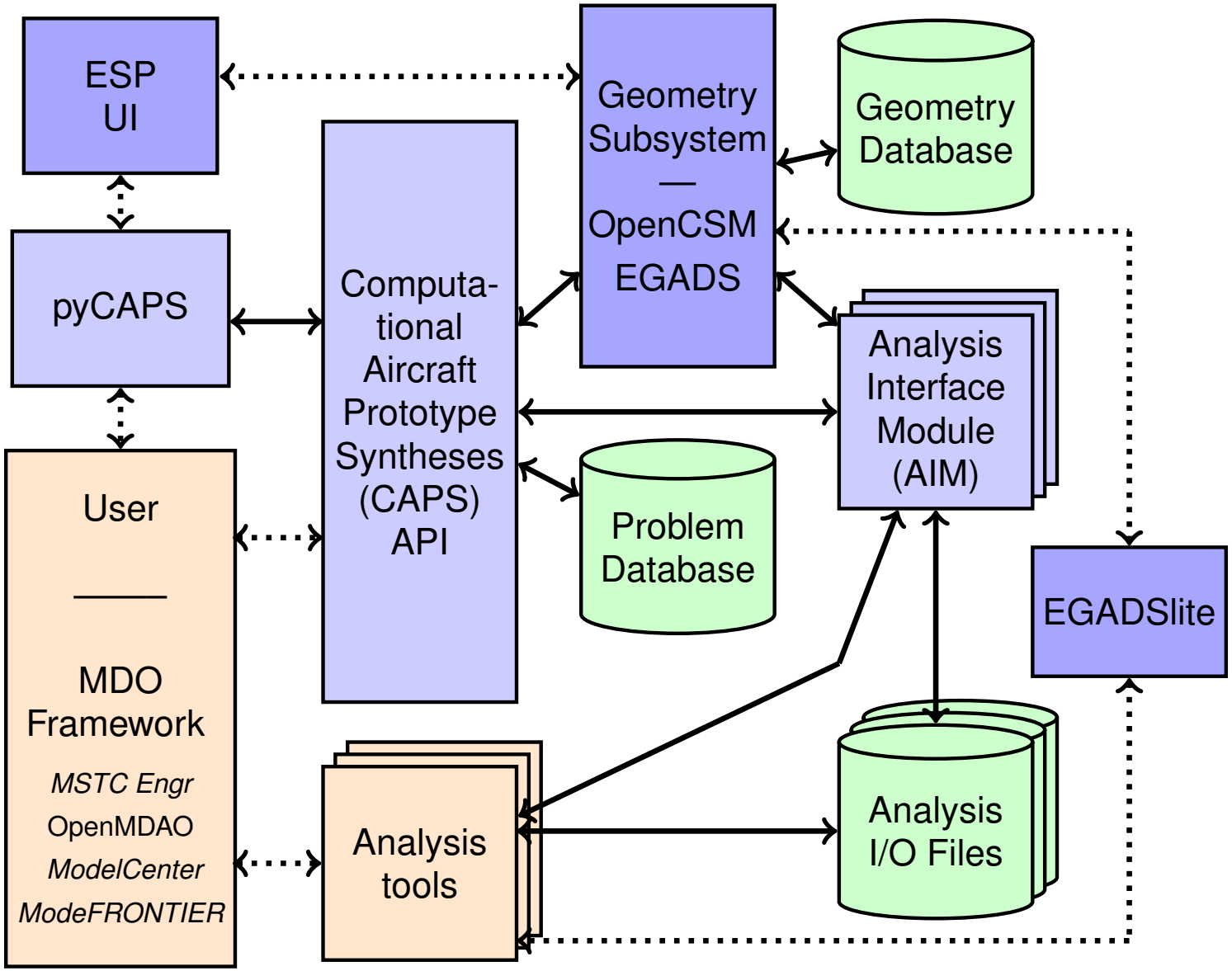
Concepts and Terminology used in the Engineering Sketch Pad at Revision 1.21

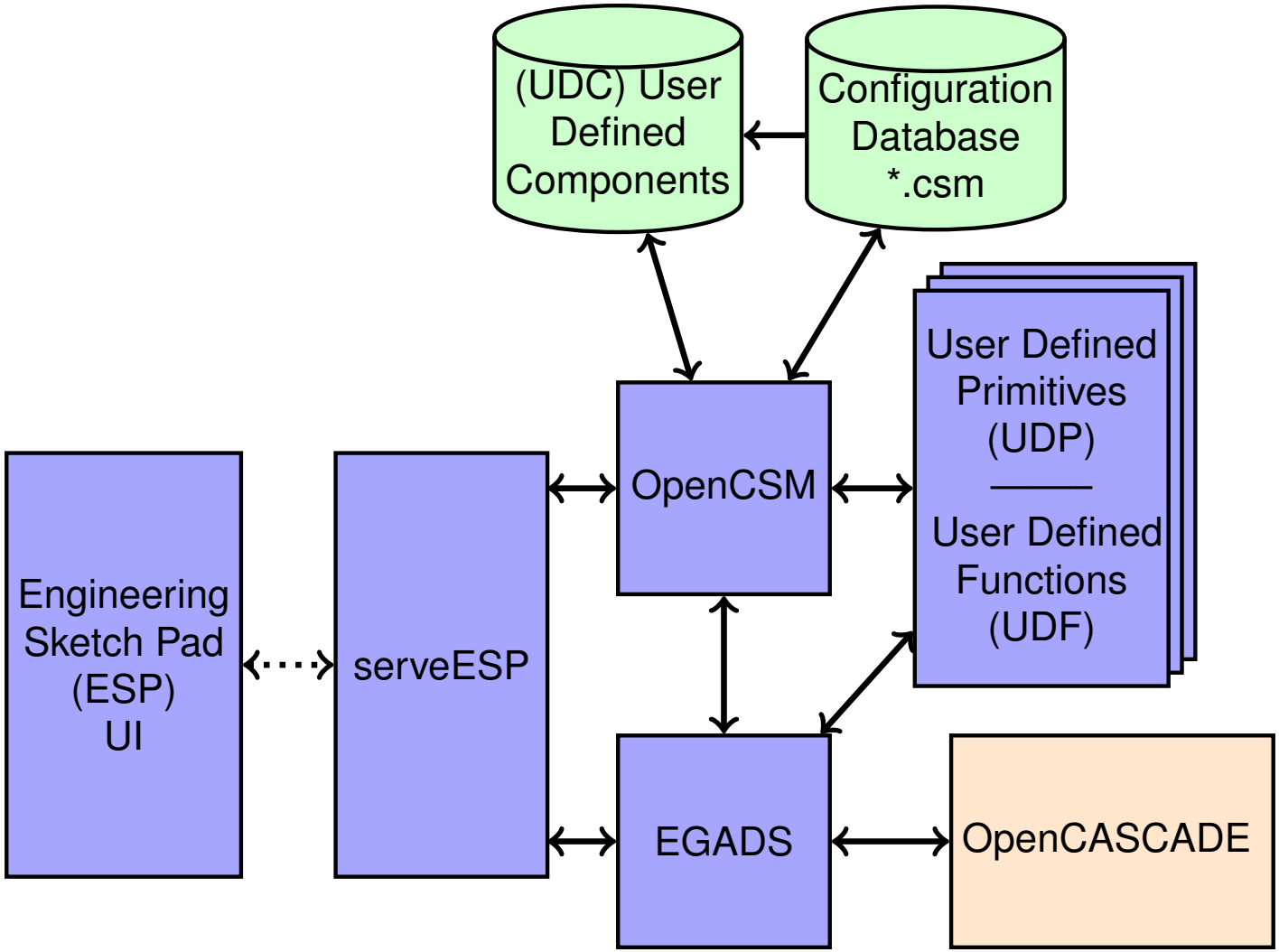
Bob Haimes
haimes@mit.edu
Aerospace Computational Design Lab
Department of Aeronautics & Astronautics
Massachusetts Institute of Technology

Part of the Engineering Sketch Pad (ESP)



ESP Table of Contents

ESP's Geometry Subsystem Architecture



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 $[u, v] = g(t)$

curve

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

EGADS Topology

Boundary Representation – BRep

Top Down ↓ ↑ *Bottom Up*

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**	

- 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 a point – $[x, y, z]$
- 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 pcurve

EGADS Topology Objects – Loops

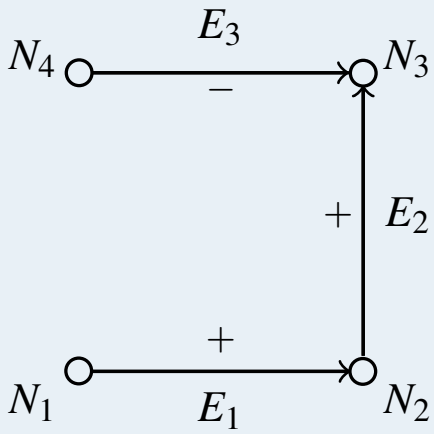
Loop – without a reference surface

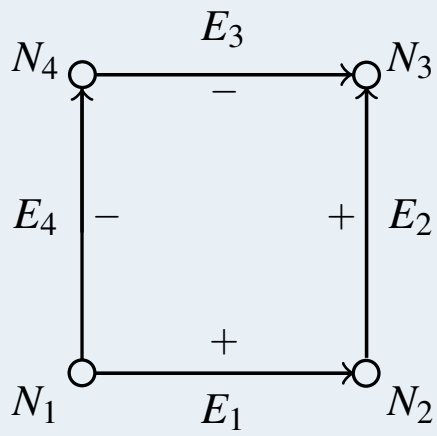
1. Free standing connected Edges that can be used in a non-manifold setting (for example in WireBodies)
2. A list of connected Edges associated with a Plane (which does not require pcurves)

- An ordered collection of Edge objects with associated senses that define the connected *Wire/Contour/Loop*
- Segregates space by maintaining material to the left of the running Loop (or traversed right-handed pointing out of the intended volume)
- No Edges should be Degenerate
- Types: **OPEN** or **CLOSED** (comes back on itself)

EGADS Topology Objects – Loops

Loop – without a reference surface





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

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

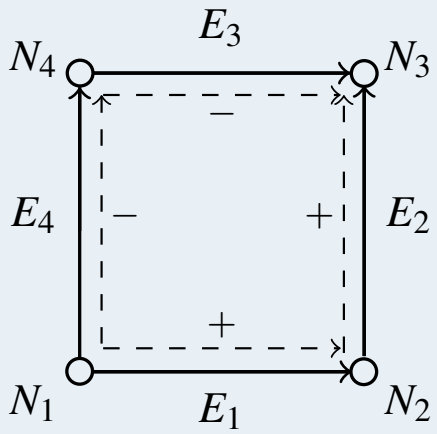
EGADS Topology Objects – Loops

Loop – with a reference surface

1. Collections of Edges (like without a surface) followed by a corresponding collection of pcurves that define the $[u, v]$ trimming on the surface
   - 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)
   - An Edge may be found in a Loop twice (with opposite senses) and with different pcurves. For example a closed cylindrical surface at the seam – one pcurve would represent the beginning of the period where the other is the end of the periodic range.
   - Types: **OPEN** or **CLOSED** (comes back on itself)

EGADS Topology Objects – Loops

Loop – with a reference surface (**CLOSED**)



dotted lines indicate associated pcurves

EGADS Topology Objects – Face

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

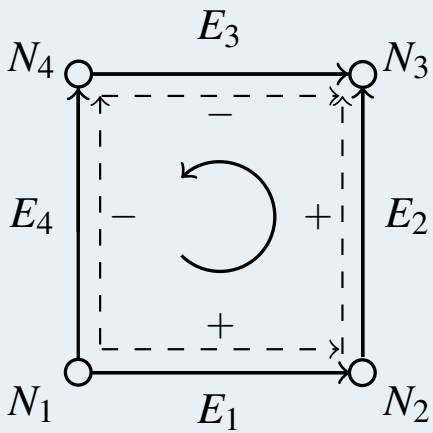
Note that this is coupled with the Loop's orientation (i.e. an outer Loop traverses the Face in a right-handed manner defining the outward direction)

EGADS Topology Objects – Face

Face

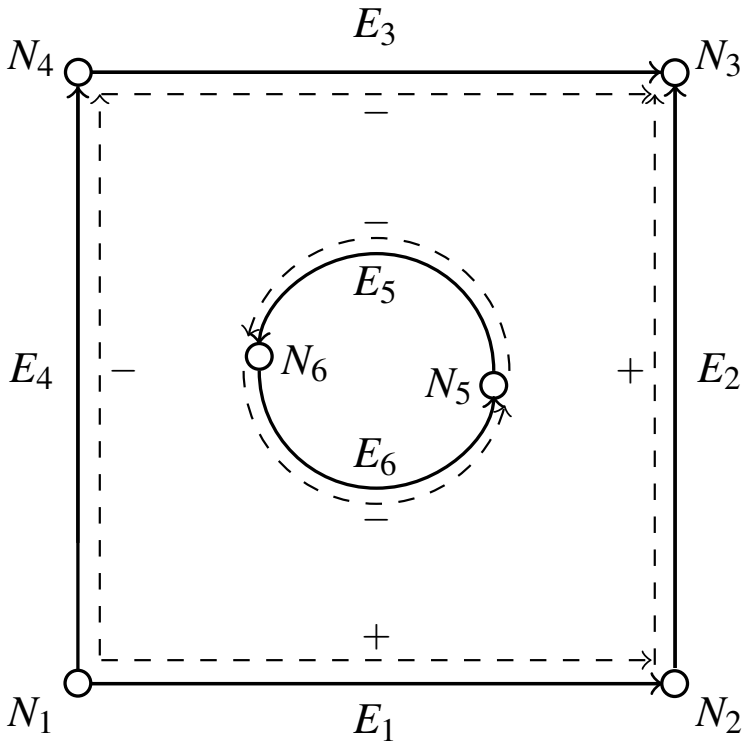
- An outer Loop traverses the Face in a right-handed manner
- Inner Loops trim the Face in a left-handed manner
- *Material* is to the left of the Edges going around the Loops

surface normal
is out of the page



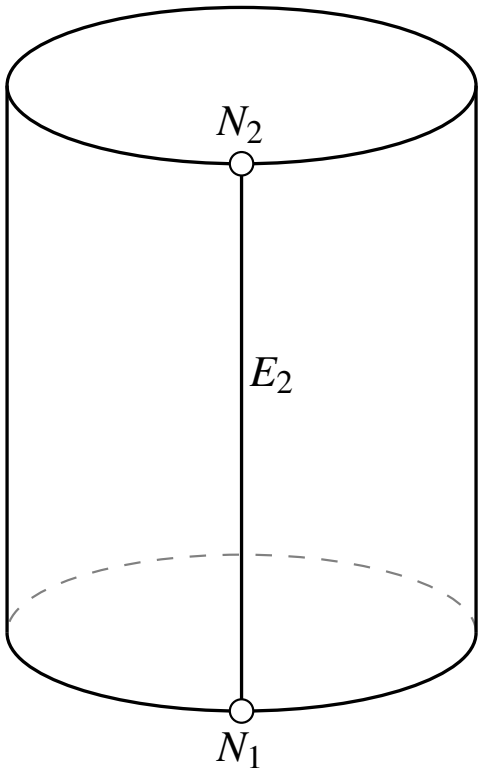
Single Outer Loop – right handed/counterclockwise: $+E_1$ $+E_2$ $-E_3$ $-E_4$

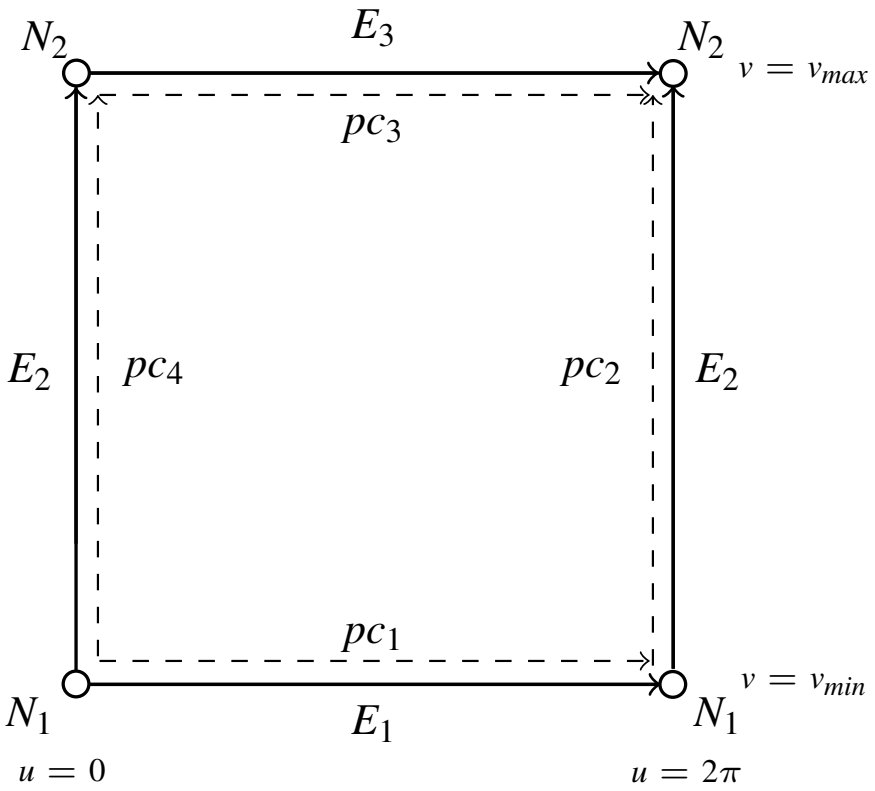
EGADS Topology Objects – Face



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

EGADS Topology Objects – Face



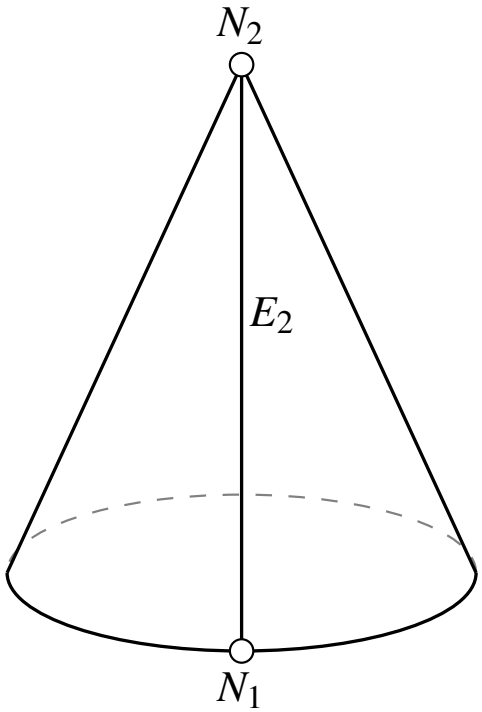


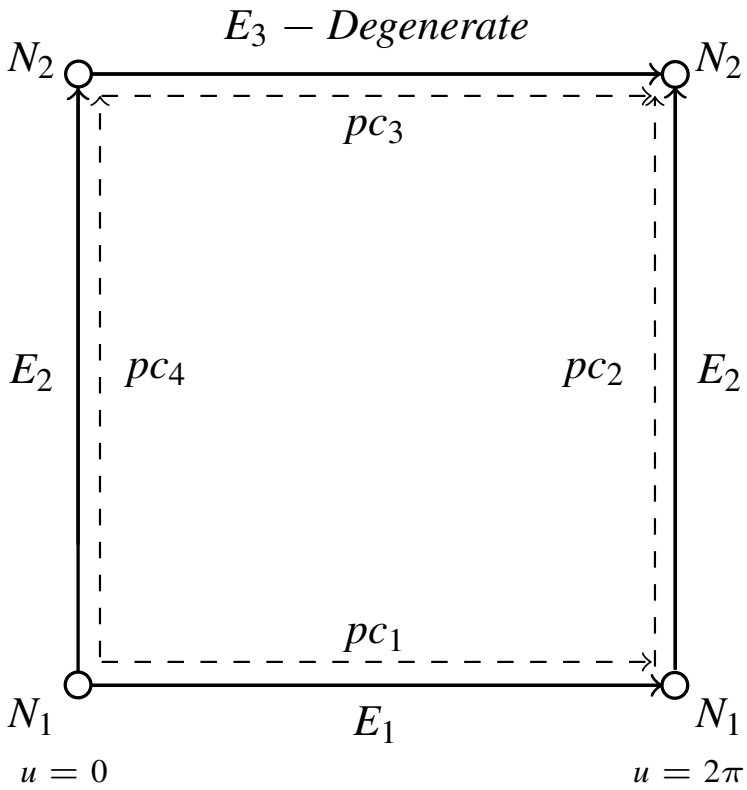
Unrolled periodic cylinder Face

Single Outer Loop – right handed/counterclockwise:

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

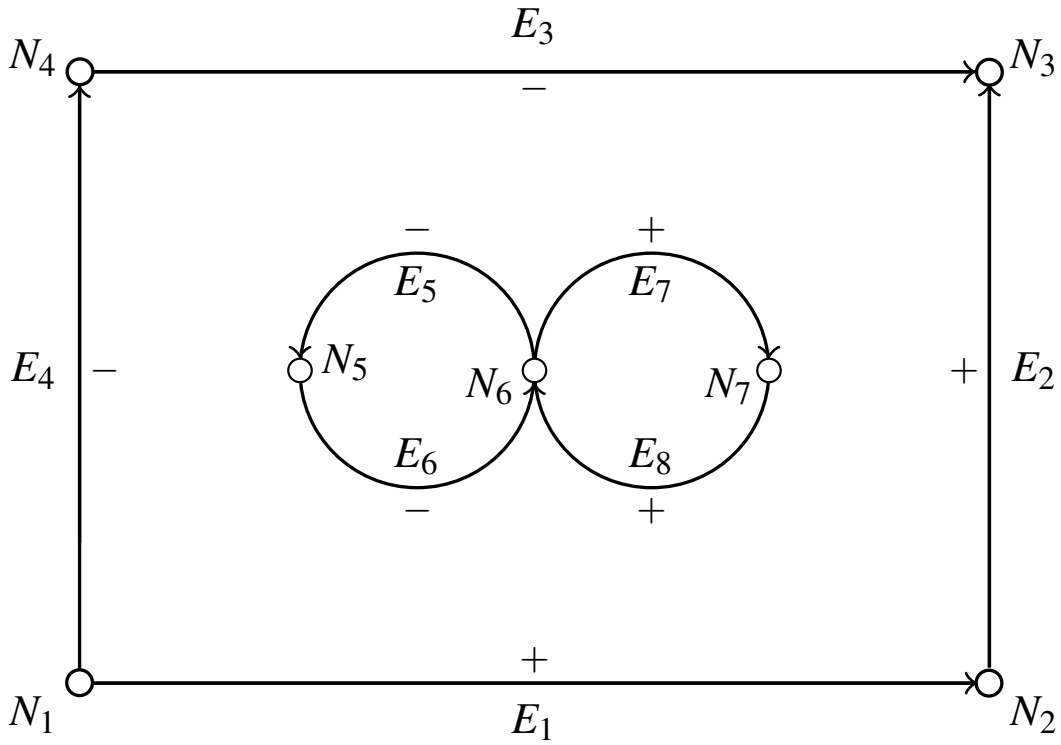
EGADS Topology Objects – Face





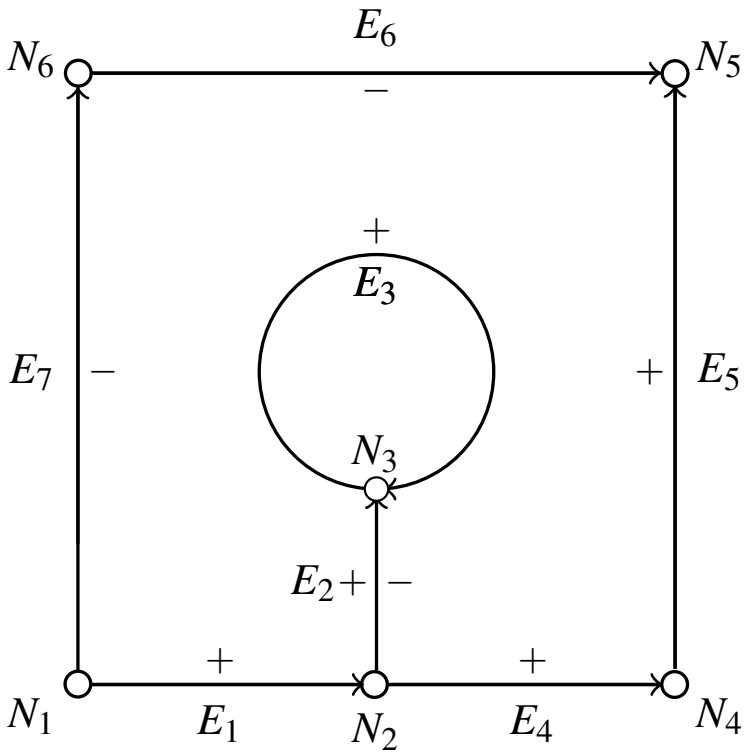
Unrolled Cone

EGADS Topology Objects – Face



- 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: $+E_7$ $+E_8$

EGADS Topology Objects – Face



Single Outer Loop – right handed/counterclockwise:

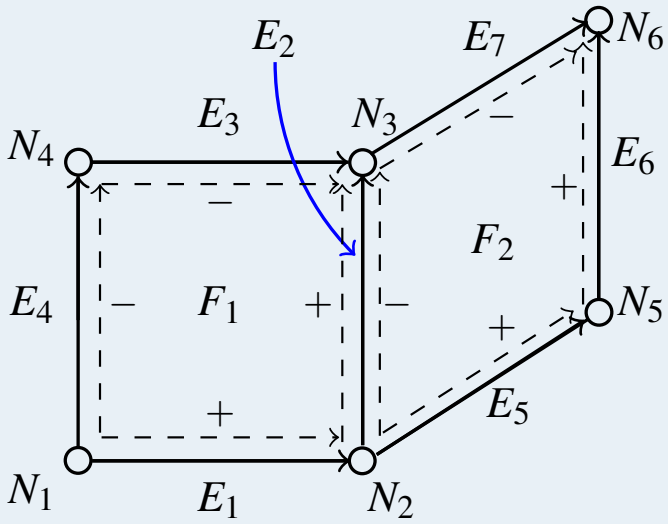
$+E_1$ $+E_2$ $+E_3$ $-E_2$ $+E_4$ $+E_5$ $-E_6$ $-E_7$

Note: pcurve is the same for both sides of E_2

EGADS Topology Objects

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

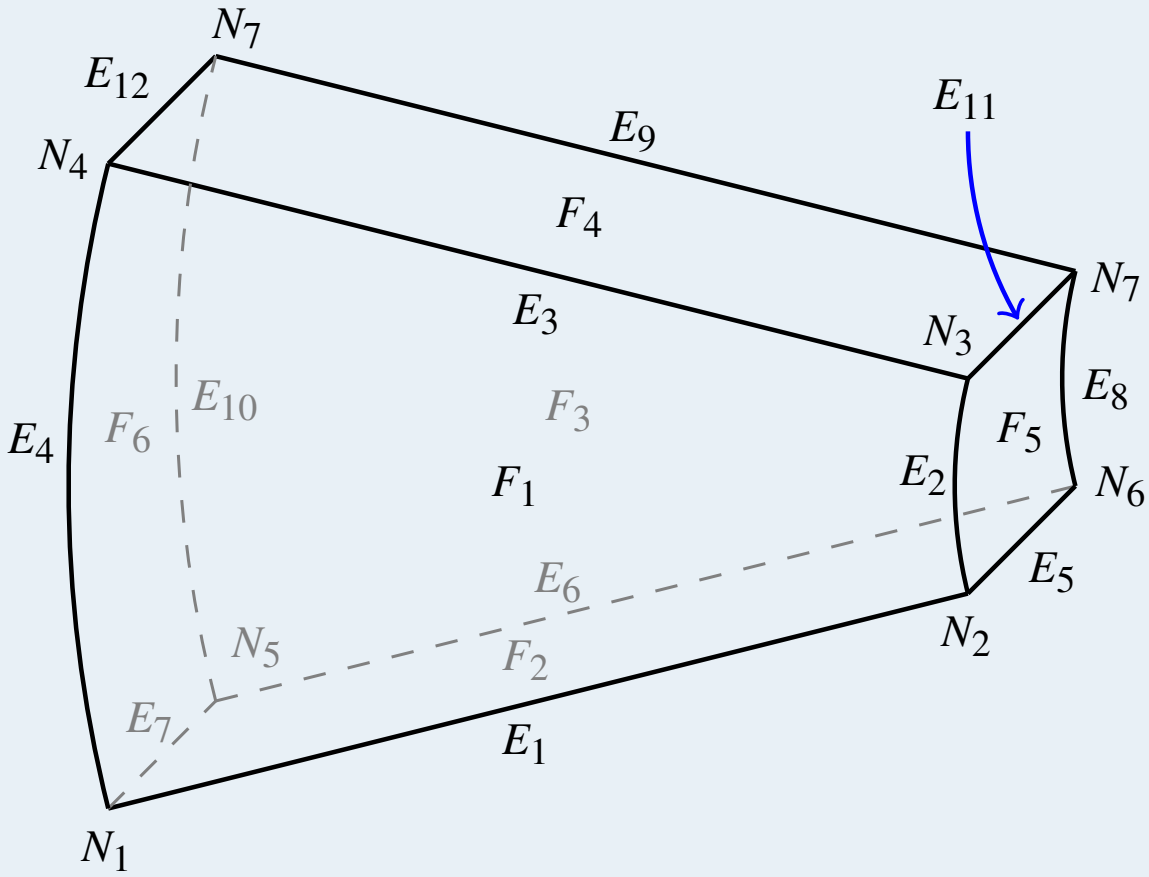
Face #2 Loop: $+E_5$ $+E_6$ $-E_7$ $-E_2$

EGADS Topology Objects

Body

- Container used to aggregate Topology
- Connected to support non-manifold collections at the Model level
- *Owns* all the Objects contained within
  - A **WIREBODY** type contains a single Loop
  - A **FACEBODY** contains a single Face – IGES import
  - A **SHEETBODY** contains one or more Shell(s) which can be either non-manifold or manifold (though usually a manifold Body of this type is promoted to a **SOLIDBODY**)
  - **SOLIDBODY**:
    - A manifold collection of one or more **CLOSED** Shells with associated senses
    - There may be only one outer Shell (sense = 1) and any number of inner Shells (sense = -1)
    - Edges (except **DEGENERATE**) found exactly twice (sense = ± 1)

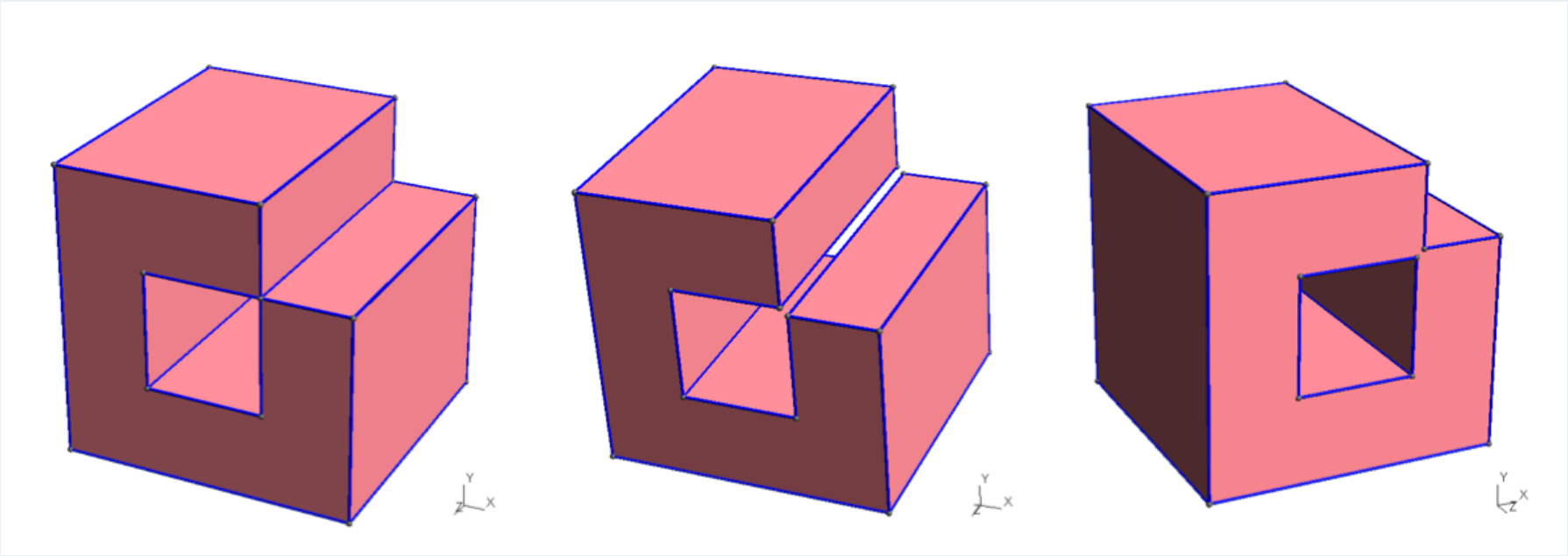
Simple **SOLIDBODY** example



8 Nodes, 12 Edges, 6 Loops and 6 Faces

EGADS Topology Objects

Manifold (**SOLID**) vs. Non-manifold (**SHEET**) Bodies



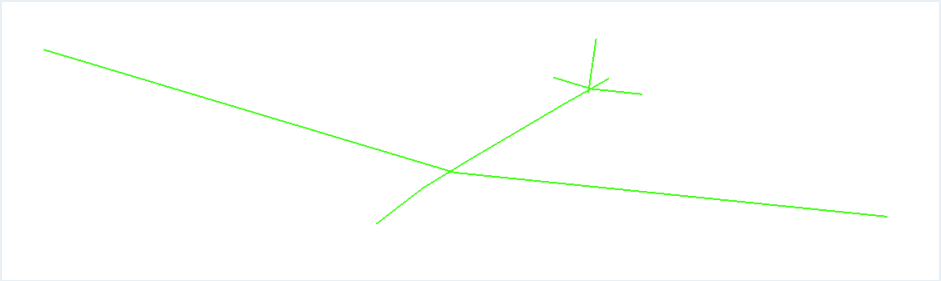
non-manifold manifold manifold

Model

- A collection of Body, EBody and Tessellation Objects
  becomes the *Owner* of contained Objects
- Returned by SBO & Sew Functions (only Body Objects)
- Read and Written by EGADS

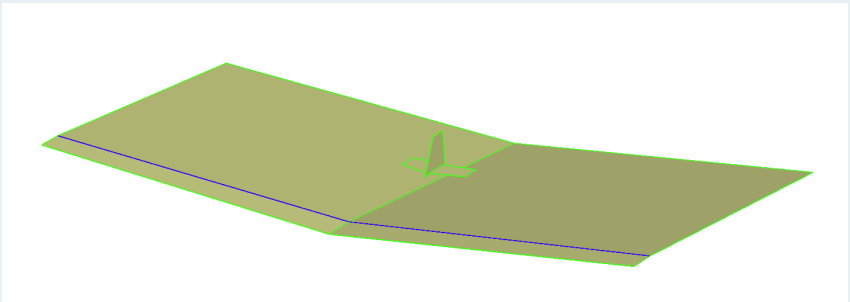
EGADS Topology Objects

Body Examples

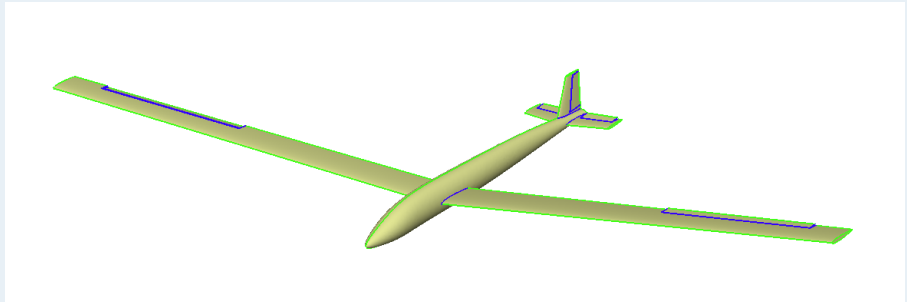


Wire Bodies

Face (Sheet) Bodies *



Sheet Bodies



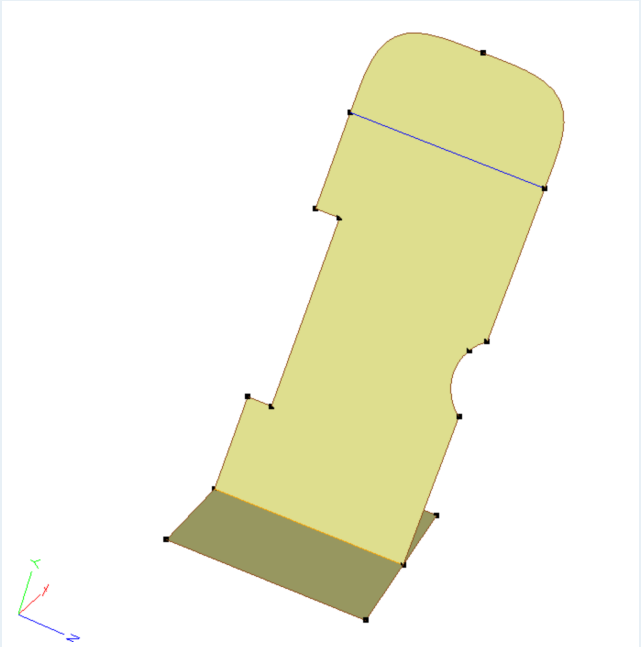
Solid Body

* OpenCSM treats all **FACEBODY**s as **SHEETBODY**s

EGADS Objects – Effective Topology

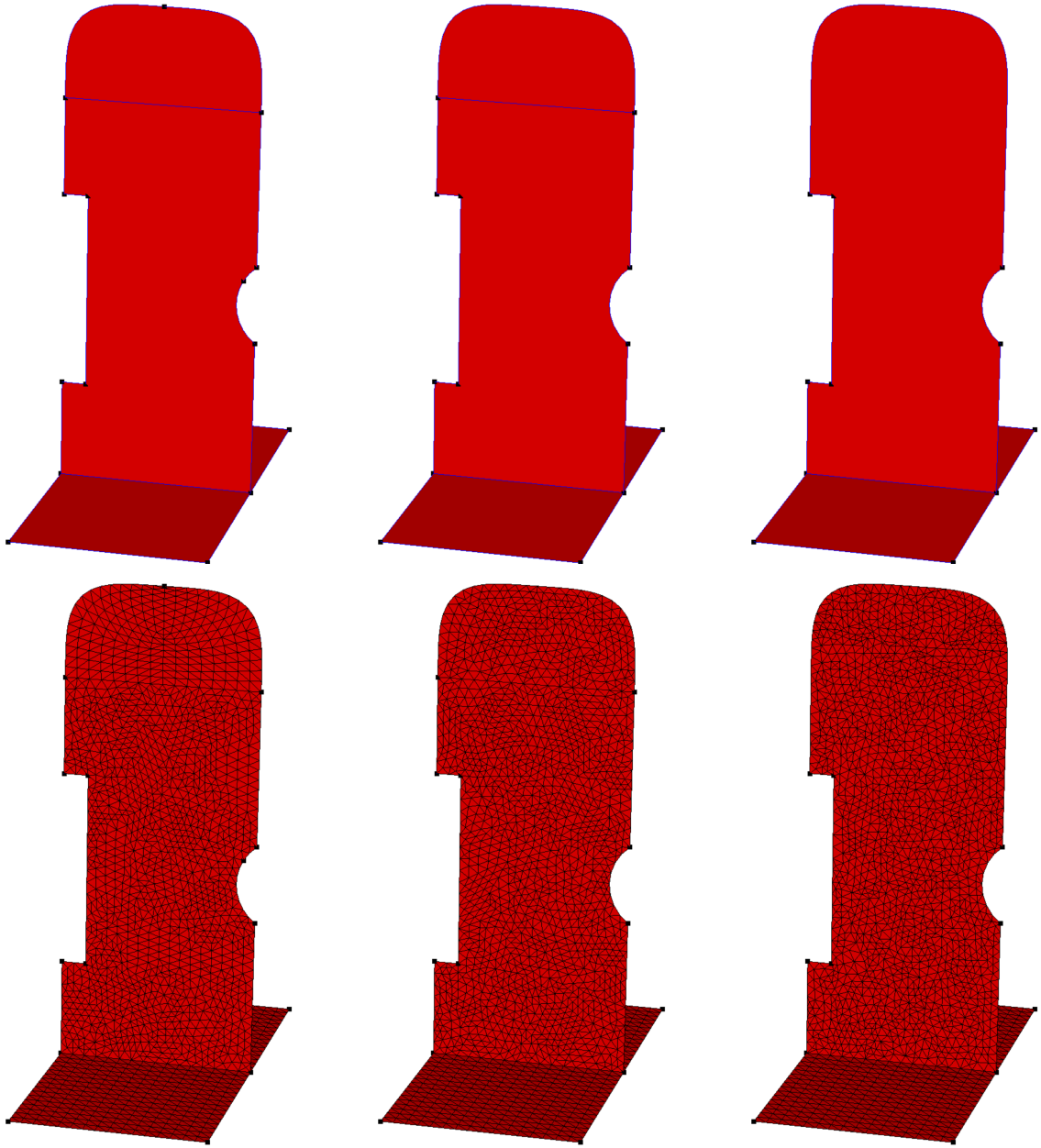
Virtual Topology & BRep closure

- BRep Topology inhibits *quality* meshes
  - Spurious Nodes
  - Small Edges that could be coalesced
  - Sliver Faces
- EGADS' *Effective Topology*
  - Automatically removes spurious Nodes
  - Automatically coalesces Edges
  - Collects Faces explicitly or by attribute uses *EGADS* tesellation & global $[u, v]$
  - Adjusts *Effective Face & Edge* evaluations based on closure



- EGADS has been updated to generate fewer spurious Nodes during the Boolean Operators

EGADS Objects – Effective Topology



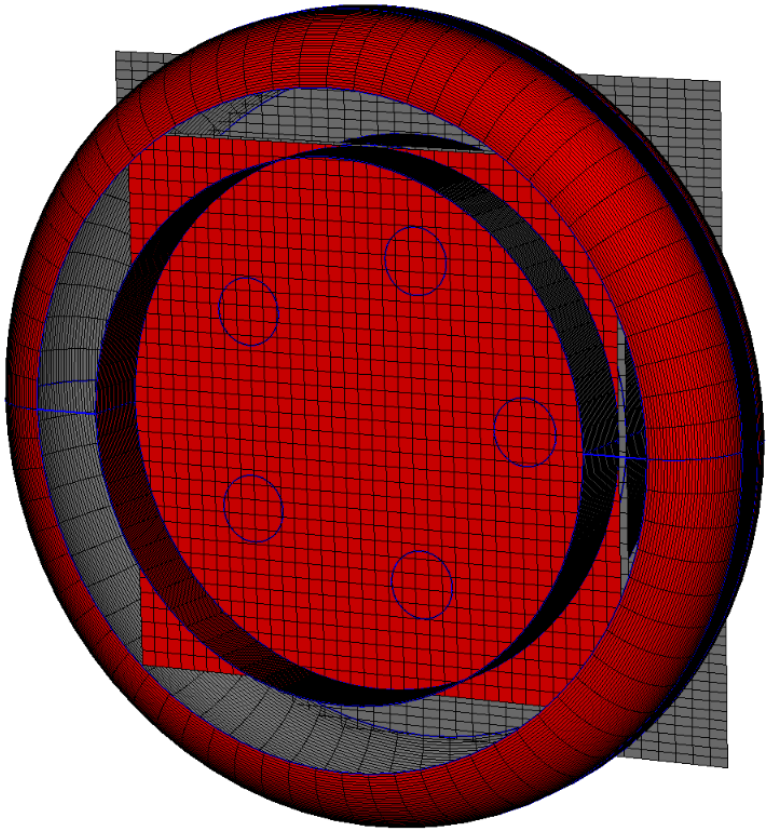
EGADS Objects – Effective Topology

Topology	BRep define	Effective	Note
Model			Container for Bodies, EBodies & Tessellations
Body	BODY	EBODY	An EBody is a modification of a Body w/ "E" entities
Shell	SHELL	ESHELL	1 to 1 mapping
Face	FACE	EFACE	EFace consists of 1 or more Faces
Loop	LOOP	ELOOPX	
Edge	EDGE	EEDGE	EEdge consists of 1 or more ordered Edges
Node	NODE	n/a	no ENodes, but not all Nodes in the Body are found in the EBody

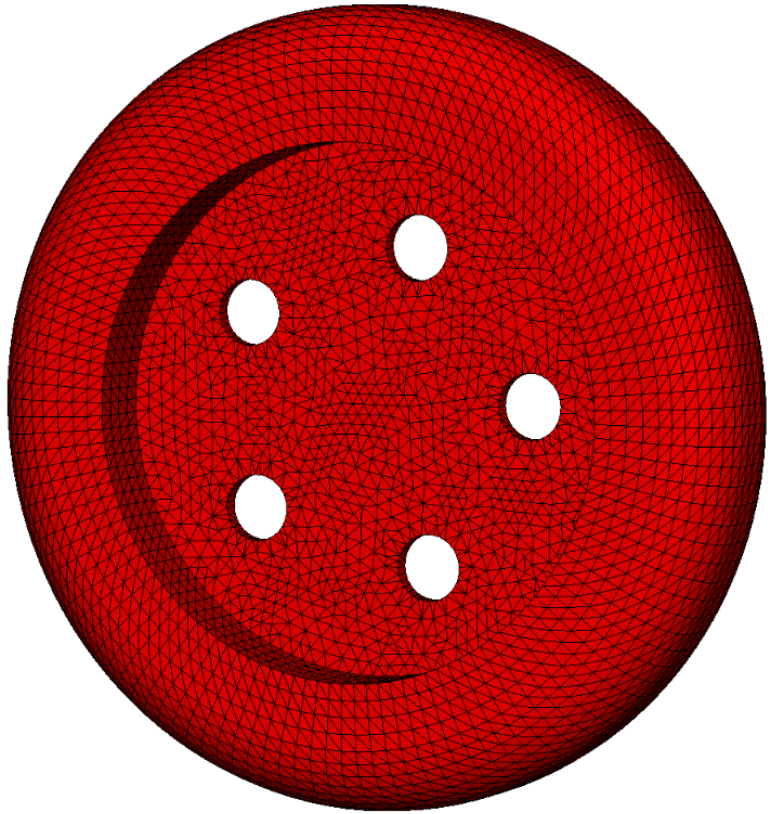
EGADS Objects – Tessellation

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

EGADS Objects – Tessellation



Geometry Tessellation



Body/EBody Tessellation

EGADS Objects – Attribution

- Attributes – metadata consisting of name/value pairs
  - Unique name – no spaces
  - A single type: Integer *, Real, String, CSys (Coordinate Systems)
  - A length (not for strings)
- Objects
  - Any EGADS Object can have multiple Attributes (each with a unique name)
  - 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 are lost
- CSys Attributes are modified through Transformations

* OpenCSM supports only Real numeric attributes (integer values are converted)

OpenCSM Terminology

OpenCSM (Constructive Solid Modeling) is an ESP component that:

- allows for the build of Parametric Models
- uses a *stack*-like language specifically targeted for:
  - building geometries that exactly *fit* the analysis/meshing at-hand
  - building multidisciplinary geometries, which share parameters and geometric entities
  - consistently *tagging* resultant geometry with attributes
- stack contains EGADS Body objects and/or Nodes
- provides "design velocities" (Parametric Sensitivities)
- can access custom CAD-like "features" (operations) UDPs, UDFs, and UDCs

OpenCSM Terminology

Solid Modeling

- Construction process guarantees that built models can be realizable **SOLID**s
  - watertight representation needed for 3D grid generators
  - **WIREBODY**s and **SHEETBODY**s are supported where needed
- Parametric models are defined in terms of:
  - Feature Tree
    - "recipe" for the construction of geometry
    - each "branch" specifies a *stack* operation
  - Design Parameters
    - "values" (dimension/sizing) that together describe a particular instance of the resultant build
    - can be scalar, vector or arrays
    - can have an associated "velocity"
  - *Internal* (driven) variables – in the form of mathematical expressions that depend on Design Parameters

OpenCSM – Customized *Features*

User Defined Primitives

- UDP geometry construction can be written either *top-down*, *bottom-up* or both
- UDPs are EGADS applets
  - create and return EGADS Body or Node Objects
  - has access to the entire suite of methods provided by EGADS
  - written in C, C++, or FORTRAN, are compiled and built into Shared Objects/DLLs
- UDPs are coupled into ESP dynamically at run time

User Defined Functions

UDFs are like UDPs except:

- can pull items off of the stack
- are not required to return EGADS Body or Node Objects

OpenCSM – Customized *Features*

User Defined Components

- UDCs can be thought of as "macros"
  and are found as separate files (from the *.csm* file)
- UDCs create zero or more stack entries
- UDCs are written as CSM-like scripts
  like routines, UDCs have *interface syntax* and specific *internal* variable scoping

OpenCSM – Definitions (1)

@-parameter A local variable that is set by the system every time a new Body is created or a SELECT statement is executed. The local variables, which cannot be set by the user, contain information such as the identity of various entities or mass properties.

argument An expression that is input to an CSM statement. Arguments are positional (that is, their meanings are specified by their order). Optional arguments are listed last, and their default values are listed in the command's description.

associative A concept that means that an entity in one Body is another representation of some other entity in some other Body.

autosave.csm A file that contains a snapshot of the state of ESP before any command is executed.

OpenCSM – Definitions (2)

Attribute A user-defined name/value pair that is associated with a Branch, Body, Face, Edge, or Node. Names that begin with an underscore _) have special meaning to CSM and those that begin with a period (.) have special meaning to EGADS. The values associated an attribute can either be a string value (prepended by a dollar-sign ($)) or a semicolon-separated list of expressions.

activity An characteristic of a Branch which tells if the Branch should be executed the next time the Model is re-built. ESP supports 'active' and 'suppressed' activities.

OpenCSM – Definitions (3)

Body An object that is created by ESP to represent some physical artifact. OpenCSM supports SolidBodys, SheetBodys (which consist of a collection of connected Faces that may or may not be manifold), WireBodys (which consist of a collection of connected Edges, where each Edge shares a bounding Node with at most one other Edge), and NodeBodys (which consist of a single point in space).

Boolean operation An operation that combines two Bodys (on the stack). The UNION operation returns the fusion of two Bodys, the INTERSECT operation returns the common part of two Bodys, and the SUBTRACT operation returns the portion of Body1 that is not in Body2.

OpenCSM – Definitions (4)

Branch An entity in the Model's Feature Tree that corresponds to either a primitive solid, transformation, Boolean operator, sketch entity, or other item used in the construction of a Model.

BRep A boundary representation is a collections of Nodes, Edges, and Faces that describe the boundary of a Body.

browser A computer program with which a user interacts with ESP. ESP currently runs in FireFox , Google Chrome and many others.

client A program, typically a web browser, with which a user interacts. The client handles some of ESP's operations directly (such as image manipulation), but sends messages to the server to perform the majority of ESP's operations.

OpenCSM – Definitions (5)

collapse The process of 'closing up' a node in a tree so that its children are not displayed. This is accomplished by pressing the − to the left of an (expanded) tree node.

command Synonym for statement.

command line The statement typed into a terminal window to start `serveESP`.

Configuration Parameter A value that can be set by the user, either programmatically or via the `ESP` user interface, that is used to generate a specific instance of a model. Sensitivities cannot can be found with respect to a Configuration Parameter.

OpenCSM – Definitions (6)

constructive solid modeler A process by which complex Bodys are created through the combination of simpler (primitive) Bodys.

curve A path through space, where the locations of points along the curve are given as $[x, y, z] = f(t)$, where t is called the parametric coordinate. Examples of curves include lines, conics, and NURBS curves.

degree of freedom A variable in a sketch whose value must be computed by satisfying one or more constraints. Each line in a sketch adds 2 degrees of freedom, each circular arc adds 3 degrees of freedom, ...

OpenCSM – Definitions (7)

dot-suffix A mechanism through which some property of a (multi-values) Parameter or Variable is returns (rather than the Parameter's value). For example, `x.nrow` returns the number of rows of `x`.

drag An operation in which a user presses a mouse button and holds it down while moving it to another location on the screen.

Design Parameter A value that can be set by the user, either programmatically or via the `ESP` user interface, that is used to generate a specific instance of a model. Sensitivities of the geometry or tessellation can be found with respect to any Design Parameter.

OpenCSM – Definitions (8)

Design Velocity A change in an input parameter from which changes in the local surface normals will be computed.

Edge The part of a BRep that is associated with a curve. Each Edge has an underlying curve, the parametric coordinate (`tbeg`) at the beginning of the Edge, the parametric coordinate (`tend`) at the end of the Edge, and the Nodes at tbeg and tend. If all the Edges in a Body support exactly two Faces, the Body is said to be manifold.

EGADS The Electronic Geometry Aircraft Design System, is an open-source geometry interface to OpenCASCADE, in which the functionality in OpenCASCADE that is needed for construction of typical applications is incorporated into about 100 C-functions.

OpenCSM – Definitions (9)

ESP The Engineering Sketch Pad is a browser-based software system that allows users create, modify, (re-)build, and save constructive solid models built via OpenCSM.

expand he process of 'opening up' a node in a tree to see its children nodes. This is accomplished by pressing the + to the left of a (collapsed) tree node.

expression An algebraic combination of variables and constants that produce a single number. Expressions can use any of OpenCSM's built-in functions and/or dot-suffixes. Expressions are used as argument to OpenCSM's commands.

OpenCSM – Definitions (10)

Face The part of a BRep that is associated with a surface. Faces are bounded by trimming curves in the form of Loops. Each Face has only one outer Loop and zero or more inner Loops (which represent holes). The trimming curves, which corresponds to the Face's bounding Edges, are described as a series of pcurves.

OpenCSM – Definitions (11)

Feature Tree A build prescription that is made up of a series of statements (or commands). The statements in the Feature Tree are executed sequentially (with loops being represented by patterns and logic represented by IFTHEN blocks). During the execution of the Feature Tree, a stack of Bodys are maintained. Each statement that generates a Body puts it onto the stack; statements that modify or combine Bodys get their inputs by popping Bodys off the stack (with the most recently created being popped off first). When CSM completes, the Bodys that remain on the stack are available as output of CSM.

OpenCSM – Definitions (12)

flying mode A way of panning, zooming, and rotating a display in which the motion of the image in the Graphics Window changes as long as the user holds the mouse button. Use the ! key in the Graphics Window to toggle flying mode on and off.

function An atomic operation that transforms its inputs into a single value. Example include trigonometric operations and single in-line logical constructs.

global Attribute An Attribute that is specified before any other CSM command or associated with a SET command before any Bodys are created. Global Attributes are added to any Body created by CSM.

OpenCSM – Definitions (13)

Graphics window The window on the top-right of the ESP screen that contains a graphical representation of the current configuration.

hostname The name of the computer that is running the server (typically serveESP). If using a single computer for both the browser and server, use 'Localhost' as the hostname.

journal A file that is written (on the server) that keeps track of the commands that user executed while running ESP. A user (who has access to the server) can copy the journal file to another name and use it to automatically replay the session that was journalled during a future invocation of serveESP.

OpenCSM – Definitions (14)

Key window The window on the bottom left of the ESP screen that contains a spectrum to indicate sensitivity values. If no sensitivity is active, this window in blank.

Local Variable Either an array of numeric values (which can contain only one value, in which case it is called a scalar) or a string of characters. Local variables get their values via SET and GETATTR statements. Local variables are not accessible outside CSM, but only within CSM while the Feature Tree is being executed.

OpenCSM – Definitions (15)

Loop A collection of Edges, arranged end to end, where each Edge has exactly two neighboring Edges. Loops, when applied to a surface, tells the part of the surface that is inside the Face.

manifold solid A manifold solid is represented by a BRep, whose Edges all support two Faces.

Messages window The window on the bottom right of the ESP screen that contains status information and other messages to the user.

OpenCSM – Definitions (16)

Model A container that contains the Parameters and (Feature Tree) Branches.

Node The topological entity associated with a single location in space. Nodes can be free-standing, such as in a NodeBody, but usually are at the ends of Edges.

OpenCASCADE An open-source geometry system on which EGADS is built.

OpenCSM – Definitions (17)

OpenCSM The open-source constructive solid modeler that is a feature-based, associative, and parametric and which build Bodys that are either manifold solids (the typical output) or non-manifold sheets and wires (such as may be needed for representing wake sheets and antennae).

Parameter A two-dimensional array of floating-point numbers that is used during the build process to generate a specific instance of a Model.

pattern A looping construct, originally used to generate a series of features on a Body (such as a regularly-space series of holes).

OpenCSM – Definitions (18)

point A location in space either at a Node, along an Edge (or curve), or on a Face (or surface).

port The port number on which the server (typically serveESP) is listening for requests by the browser. serveESP uses 7681 as its default port.

primitive A CSM statement that generates either a BOX, SPHERE, CYLINDER, CONE, or TORUS, or a user-defined primitive.

OpenCSM – Definitions (19)

semicolon-separated list A list of expressions (that evaluate to numeric values) that are written with semicolons (;) between entries. A semicolon-separated list may optionally be terminated with a semicolon.

sensitivity The derivative of the location on a Body with respect to one or more of the Design Parameters.

server A computer program in which OpenCSM runs and which 'serves' Models and Boundary Representations to ESP. The program 'serveESP' is the initial server for ESP.

OpenCSM – Definitions (20)

sketch A 2-D drawing composed of lines, circular arcs, a splines, that is used to define a SheetBody (with a single Face) or WireBody. Sketches are typically used as the basis of grown solids such as EXTRUDE, REVOLVE, RULE, and BLEND. (The latter two of these actually use a series of sketches.

sketch constraint A rule for specifying the relationships between sketch variables.

sketch variable A degree of freedom within a sketch. There are two sketch variables associated with the point between each pair of sketch segments and one additional sketch variable associated with each circular arc segment.

OpenCSM – Definitions (21)

stack A construct used with the build process to establish parent-child relationships between various features in the Feature Tree. Primitive statements, which create Bodys, push them onto the top of the stack. Transformation statements pop the top Body (or group) from the stack, transform it/them, and then pushes the transformed result back onto the stack. Boolean operation pop two (or more) Bodys from the top of the stack and push the resultant Body back onto the stack.

statement A line of CSM code that corresponds to one of the steps in the build process in the Feature Tree.

OpenCSM – Definitions (22)

suppressed A possible state for a Branch; Branches that are suppressed are not executed when the Feature Tree is executed. Suppression is typically used to temporarily remove a feature during a build.

surface A sheet in space, where the locations of points on the surface are given as $[x, y, z] = f(u, v)$, where u and v are called the parametric coordinates. Examples of surfaces include planes, cylindrical surface, and tensor-product NURBS surfaces.

transformation A type of CSM statement that pops a Body (or group) from the top of the stack, modifies it, and then pushes the modified Body (or group) back onto the stack. Examples of transformations include TRANSLATE, ROTATE*, and SCALE.

OpenCSM – Definitions (23)

Tree window The window on the top-left of the ESP screen that contains command buttons, a tree-like view of the current Parameters, a tree-like view of the current Branches (of the Feature Tree), and a tree-like view of the display settings.

UDC User-defined component. This is essentially a macro that is stored in a .udc file. It is execute with a UDPRIM statement, where the primtype either starts with / or $/

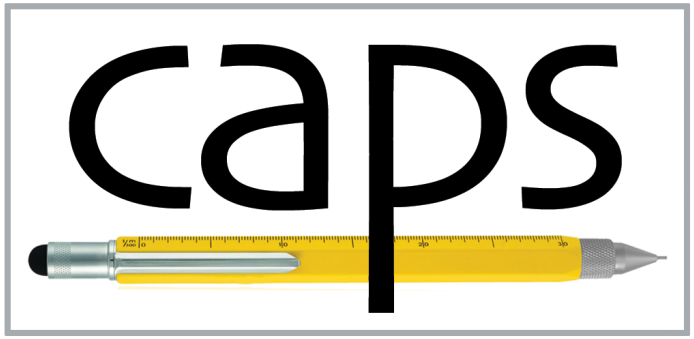
UDF User-defined function. The difference between a UDF and a UDP is that a UDP does not get any of its inputs from the stack, whereas a UDF consumes one or more Bodys from the stack.

OpenCSM – Definitions (24)

UDP User-defined primitive. This is a user-supplied compiled file (from C or FORTRAN) that creates a non-standard primitive. It is executed with a UDPRIM statement, where the primtype starts with a letter

WebViewer A piece of software, built upon the standard WebGL, that allows for the real-time view angle changes in a browser.

Computational Aircraft Prototype Synthesis



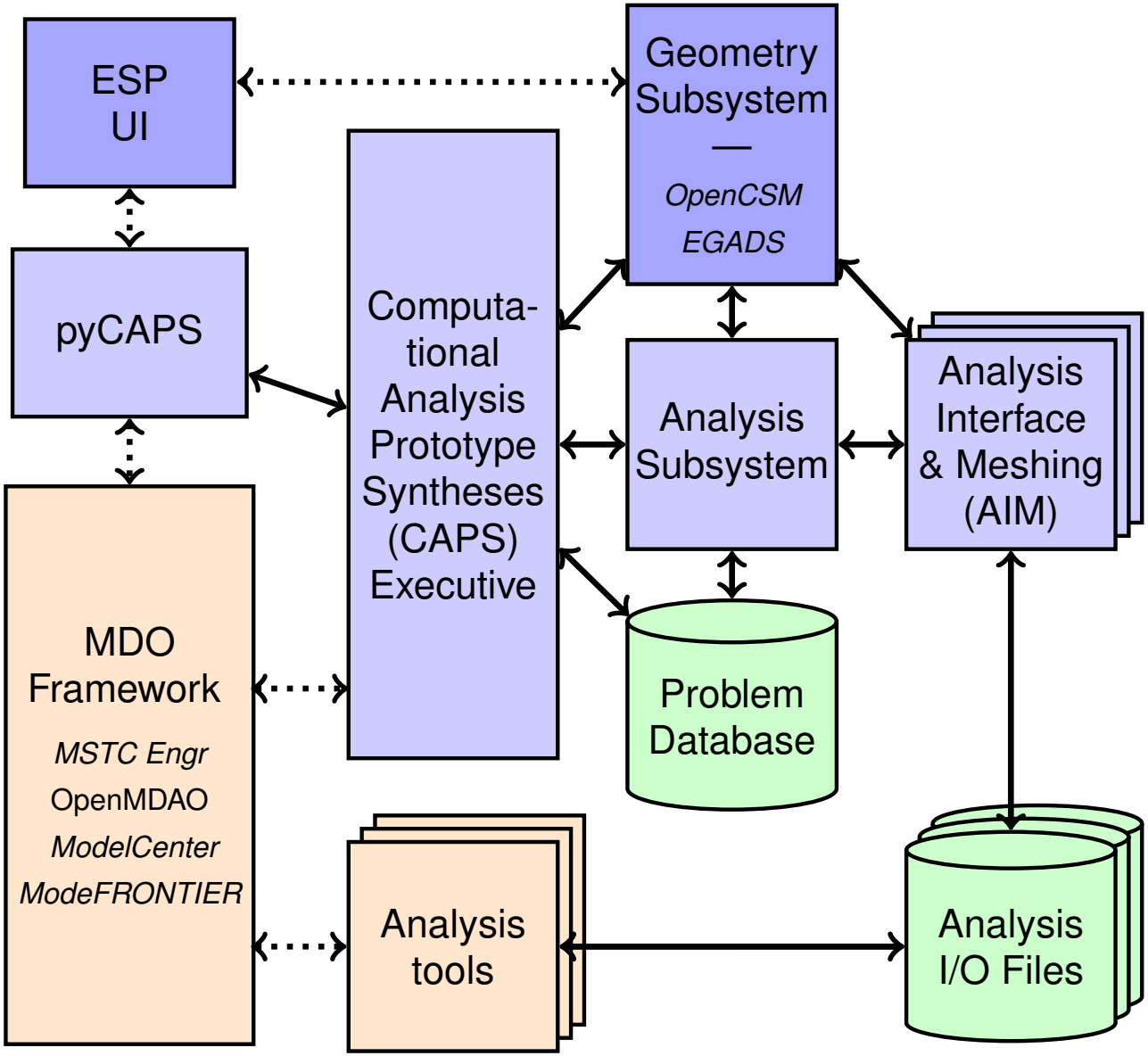
CAPS Goals

- Provide a Conceptual/Preliminary Integrated Design Environment
- Provide the tools & techniques for generalizing analysis coupling
  - multidisciplinary coupling: aeroelastic, FSI
  - multi-fidelity coupling: conceptual and preliminary design
- Provide the tools & techniques that directly deal with geometry

CAPS Access

- The entry point into the CAPS system is the C/C++ API –or–
- pyCAPS: A Python interface

ESP's Analysis Subsystem – CAPS



CAPS Definitions

Problem Object

The Problem is the top-level *container* for a single mission. It maintains a single set of interrelated geometric models, analyses to be executed, connectivity and data associated with the run(s), which can be both multi-fidelity and multidisciplinary. There can be multiple Problems in a single execution of CAPS and each Problem is designed to be *thread safe* allowing for multi-threading of CAPS at the highest level.

Value Object

A Value Object is the fundamental data container that is used within CAPS. It can represent *inputs* to the Analysis and Geometry subsystems and *outputs* from both. Also Value Objects can refer to *mission* parameters that are stored at the top-level of the CAPS database. The values contained in any *input* Value Object can be bypassed by the *linkage* connection to another Value (or *DataSet*) Object of the same *shape*.

Analysis Object

The Analysis Object refers to an instance of running an analysis code. It holds the *input* and *output* Value Objects for the instance and a directory path in which to execute the code (though no explicit execution is initiated). Multiple various analyses can be utilized and multiple instances of the same analysis can be handled under the same Problem.

Bound Object

A Bound is a logical grouping of BRep Objects that all represent the same entity in an engineering sense (such as the "outer surface of the wing"). A Bound may include BRep entities from multiple Body Objects; this enables the passing of information from one Body (for example, the aero OML) to another (the structures Body).

Dimensionally:

- 1D – Collection of Edges
- 2D – Collection of Faces

CAPS Definitions

VertexSet Object

A VertexSet is a *connected* or *unconnected* group of locations at which discrete information is defined. Each *connected* VertexSet is associated with one Bound and a single *Analysis*. A VertexSet can contain more than one DataSet. A *connected* VertexSet can refer to 2 differing sets of locations. This occurs when the solver stores it's data at different locations than the vertices that define the discrete geometry (i.e. cell centered or non-isoparametric FEM discretizations). In these cases the solution data is provided in a different manner than the geometric.

DataSet Object

A DataSet is a set of engineering data associated with a VertexSet. The rank of a DataSet is the (user/pre)-defined number of dependent values associated with each vertex; for example, scalar data (such as *pressure*) will have rank of one and vector data (such as *displacement*) will have a rank of three. Values in the DataSet can either be deposited there by an application or can be computed (via evaluations, data transfers or sensitivity calculations).

CAPS Object Hierarchy

Object Internals

All Objects can have:

- a SubType
- children in the form of CAPS Objects

Note: Body Objects are EGADS Objects (egos)

Problem Object – SubTypes: Parametric or Static (no Value Objects)

Children Objects	**SubTypes**
capsValue	GeometryIn, GeometryOut, Parameter, User
capsAnalysis	
capsBound	

CAPS Object Hierarchy

Analysis Object

Children Objects	SubTypes
capsValue	AnalysisIn, AnalysisOut, AnalysisDynO

Bound Object

Children Objects	SubTypes
capsVertexSet	Connected, Unconnected

VertexSet Object

Children Objects	SubTypes
capsDataSet	FieldIn, FieldOut, User, GeomSens, TessSens, Builtin

CAPS Body Filtering

CSM AIM targeting: "capsAIM"

The `CSM` script generates Bodies which are designed to be used by specific AIMs. The AIMs that the Body is designed for is communicated to the CAPS framework via the "capsAIM" string attribute. This is a semicolon-separated string with the list of AIM names. Thus, the `CSM` author can give a clear indication to which AIMs should use the Body. For example, a body designed for a CFD calculation could have:

ATTRIBUTE capsAIM $su2AIM;fun3dAIM;cart3dAIM

CAPS AIM Instantiation: "capsIntent"

The "capsIntent" Body attribute is used to disambiguate which AIM instance should receive a given Body targeted for the AIM. An argument to `caps_load` accepts a semicolon-separated list of keywords when an AIM is instantiated in CAPS/pyCAPS. Bodies from the "capsAIM" selection with a matching string attribute "capsIntent" are passed to the AIM instance. The attribute "capsIntent" is a semicolon-separated list of keywords. If the string to `caps_load` is NULL, all Bodies with a "capsAIM" attribute that matches the AIM name are given to the AIM instance.

Other Reserved CAPS Attribute names

capsLength

This string Attribute must be applied to an EGADS Body to indicate the length units used in the geometric construction.

capsBound

This string Attribute must be applied to EGADS BRep Objects to indicate which CAPS Bound(s) are associated with the geometry. A entity can be assigned to multiple Bounds by having the Bound names separated by a semicolon. Face examples could be “Wing”, “Wing;Flap”, “Fuselage”, and etc.

Note: Bound names should not cross dimensional lines.

capsGroup

This string Attribute can be applied to EGADS BRep Objects to assist in grouping geometry into logical sets. A geometric entity can be assigned to multiple groups in the same manner as the capsBound attribute.

Note: CAPS does not internally use this, but is suggested of classifying geometry.

Analysis Interface & Meshing (AIM) 1/3

CAPS AIMs are dynamically loaded EGADS applets, which are similar in concept to OpenCSM's UDPs and UDFs

- Analysis identification – at AIM registration
  - number of inputs expected & number of possible outputs
  - geometric *intention(s)* expected
- Analysis internal state handling – *updateState*
  - always called before *Pre*, *Post* or *AIM discretization*
  - handling the complexities of the execution calling sequences: restarting, auto-execution and FSI like situations
  - plugin deals with populating the discrete BRep data from the mesh (the CAPS bound object)

Deals with the idiosyncrasies and peculiarities of each Analysis

Analysis Interface & Meshing (AIM) 2/3

- Analysis input generation – *Pre*
  - supplies Analysis Subsystem with information required to generate the input for the analysis (and optionally meshing)
    - format for the input file
    - possibly attribute BRep with geometric-based information
    - preparing the BRep data to be used for grid generation
- Analysis Execution (optional)
  - if exists allows for auto-execution of the AIM
  - must execute the analysis either:
    - directly (linked into the AIM)
    - by a *system* call
    - should not be something that takes much CPU time

Analysis Interface & Meshing (AIM) 3/3

- Analysis output parsing – *Post*
  - plugin deals with populating *bound*-based scalar, vector and/or state vector data from the solver run
  - reads or calculates integrated (performance) measures that can be used as objective functions for optimization
- Multidisciplinary coupling – when required
  - plugin provides functions to use the discrete data to Interpolate and/or Integrate (consistent with solver)
  - plugin provides *reverse* differentiated Interpolate and Integrate functions to facilitate conservative transfer optimization
  - automatically initiated in a *lazy* manner when the data transfer is requested

CAPS Phases 1/4

Phases can be thought of as branches of the design-decision "tree"

In most cases:

- The first *phase* builds the objects
- The subsequent *phases* discover the existing objects

At all times the current object state is mirrored on disk:

- Uses directory structure to mimic the object hierarchy
- Writes objects when updated in binary (which are usually small)

In addition, CAPS API function's output are *journaled* to disk

End result:

- Execution can be paused, interrupted or error and continued later
- AIM updateState/postAnalysis re-executed during continuation when last invocation is reached to reestablish any internal storage

CAPS Phases 2/4

CAPS Execution Modes

CAPShas 4 basic modes for starting a session:

- Scratch – This is for development (and not production). It will remove any existing data in the *Scratch* directory of the Problem's path
- Initial – This *phase* is started by a call to caps_open that points to a nonexistent *phase* subdirectory. The initialization can either be from a CSM, geometry file, an OpenCSM or EGADS Model.
- Continuation* – This occurs when CAPS has not finished a *phase* either do to an interruption or not reaching caps_close (that can mark the *phase* complete). In this case the CAPS application or pyCAPS script can be run from the beginning, but reading results from the *journal* is used to quickly get to the position where the *phase* terminated.
- Starting from a completed *phase* optionally with a new CSM file

* works best when most of the computation is controlled by CAPS
must be same ESP rev & architecture (due to the chaotic nature of meshing)

CAPS Directory Structure

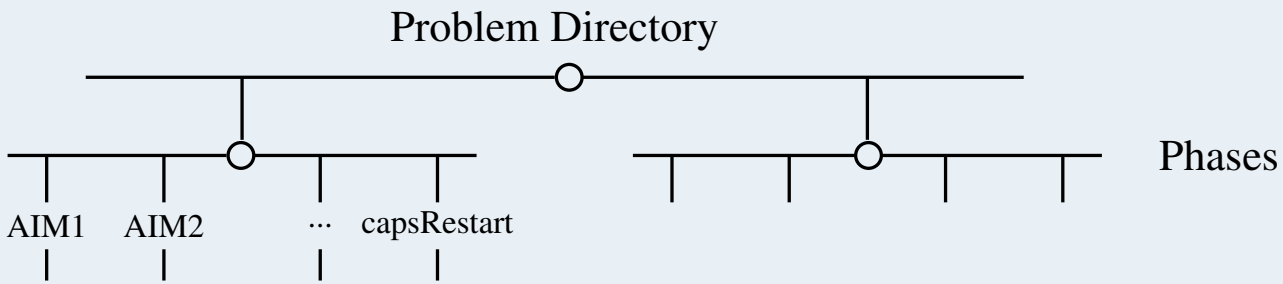
At the top specified directory level you will find *phase* subdirectories

In each *phase* subdirectory you may see:

- capsRestart.cpc – a CSM saved state file – or –
  capsRestart.egads – an EGADS file (for nonparametric runs)
- capsRestart – subdirectory that contains the CAPS restart data
- capsClosed – the *phase* has been closed (caps_close has been called marking completion)
- capsLock – an indication that another application is executing in this subdirectory
- AIMnames – subdirectories each related to an AIM instance in the running CAPS Problem/Phase

Notes:

- *Scratch* phase (no name specified) is not as protected as the others
- CAPS Problem directory or individual *phase* subdirectories can be managed by phaseUtil



CAPS Phases 4/4

Managing Problem/Phase directories

Using the command-line to copy portions of the CAPS Problem/Phase directory structure may produce odd results. The *best practice* is to use the CAPS application `phaseUtil` instead:

```
Usage:  phaseUtil prPath                          list Phases
        phaseUtil prPath -l phName                show lock owner
        phaseUtil prPath -r phName                remove lock
        phaseUtil prPath -c phName newPath        copy Phase out
        phaseUtil prPath -p phName newName        copy Phase in Problem
        phaseUtil prPath -d phName                delete Phase
        phaseUtil prPath -f newPath               full copy of Problem
```