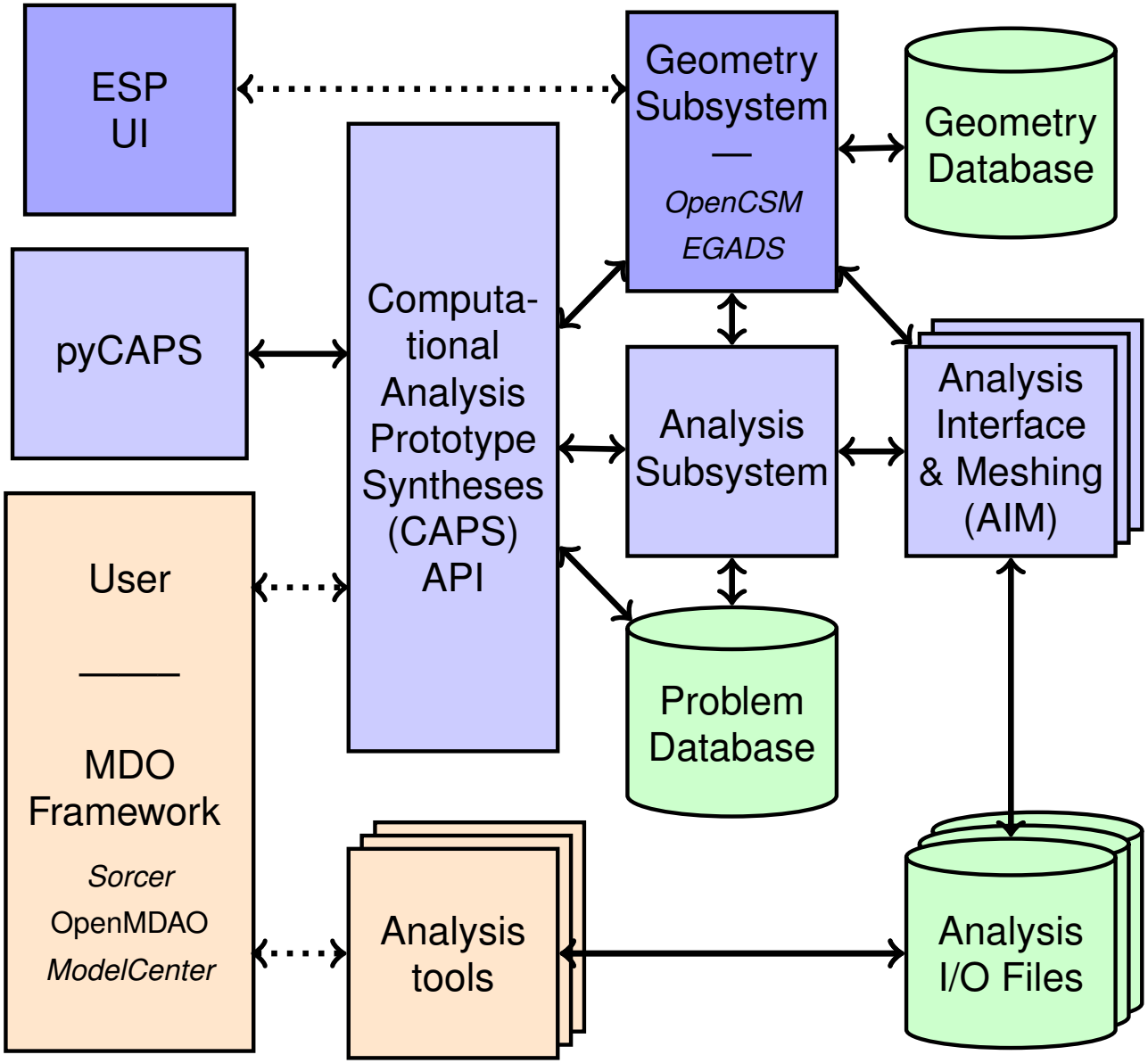
Concepts and Terminology used in the Engineering Sketch Pad at Revision 1.14

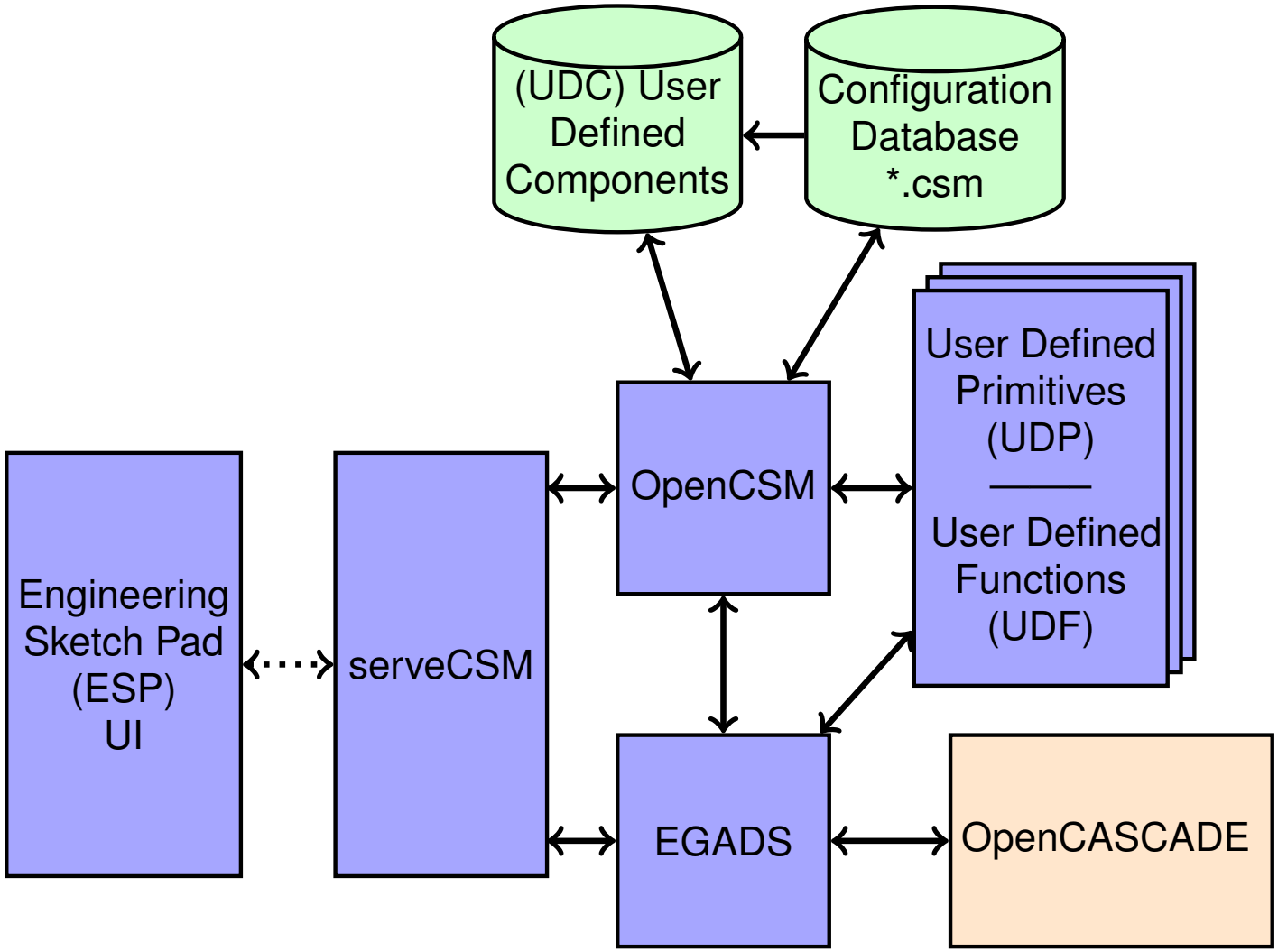
Bob Haimes
haimes@mit.edu
Aerospace Computational Design Lab
Department of Aeronautics & Astronautics
Massachusetts Institute of Technology

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

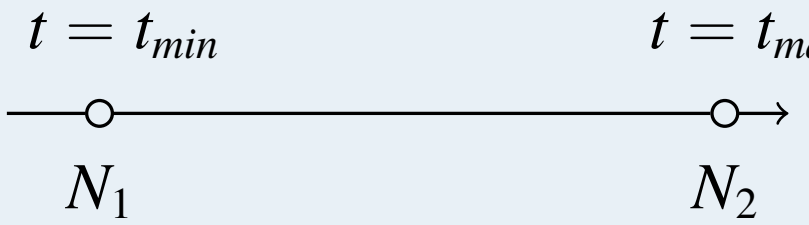
EGADS Topology Objects

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

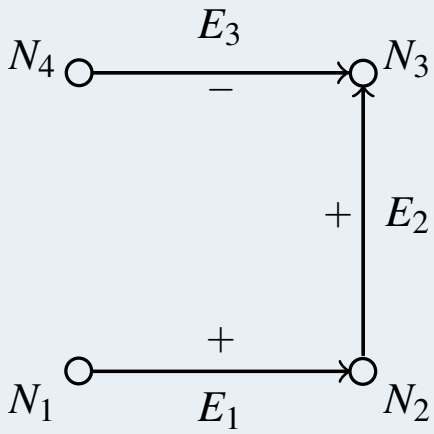
Loop – without a reference surface

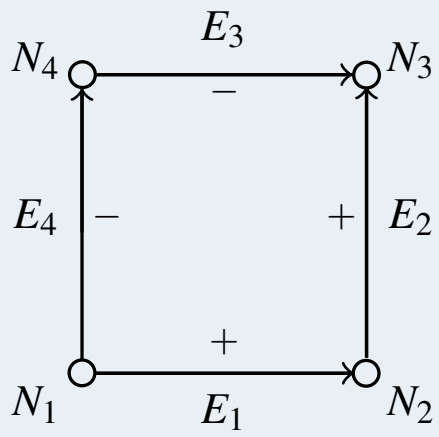
1. Free standing connected Edges that can be used in a non-manifold setting (for example in WireBodies)
2. A list of connected Edges associated with a Plane (which does not require pcurves)

- 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

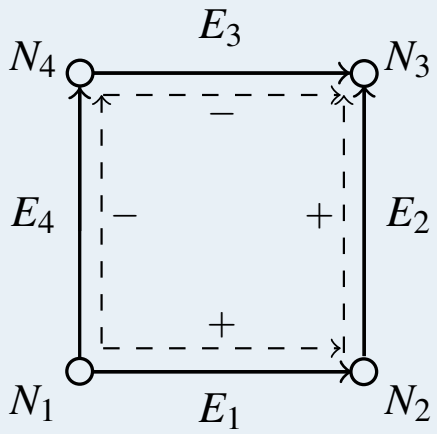
Loop – with a reference surface

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

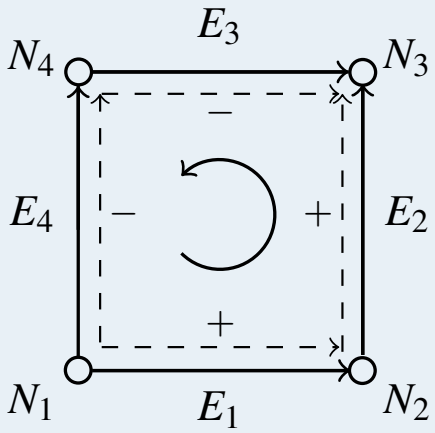
Note that this is coupled with the Loop's orientation (i.e. an outer Loop traverses the Face in a right-handed manner defining the outward direction)

EGADS Topology Objects – Face

Face

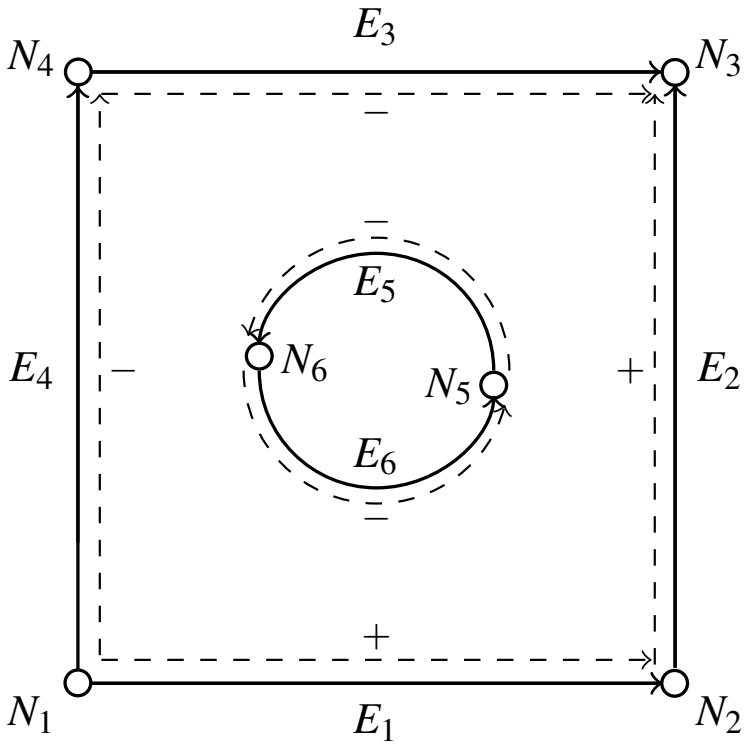
- An outer Loop traverses the Face in a right-handed manner
- Inner Loops trim the Face in a left-handed manner
- *Material* is to the left of the Edges going around the Loops

surface normal
is out of the page



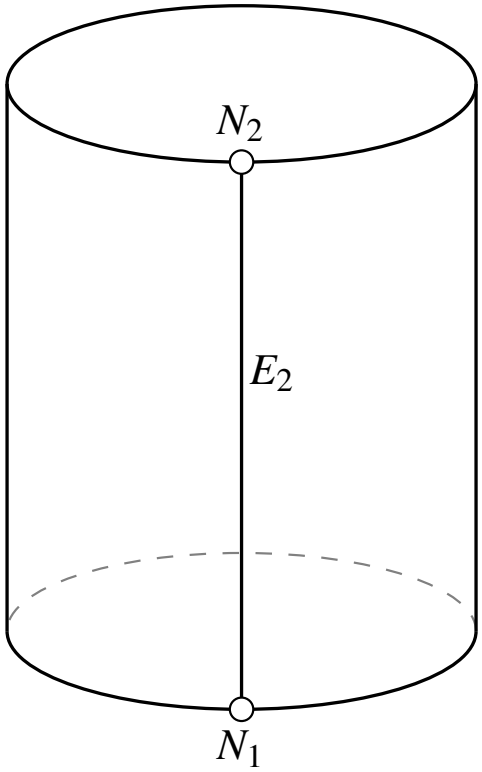
Single Outer Loop – right handed/counterclockwise: $+E_1$ $+E_2$ $-E_3$ $-E_4$

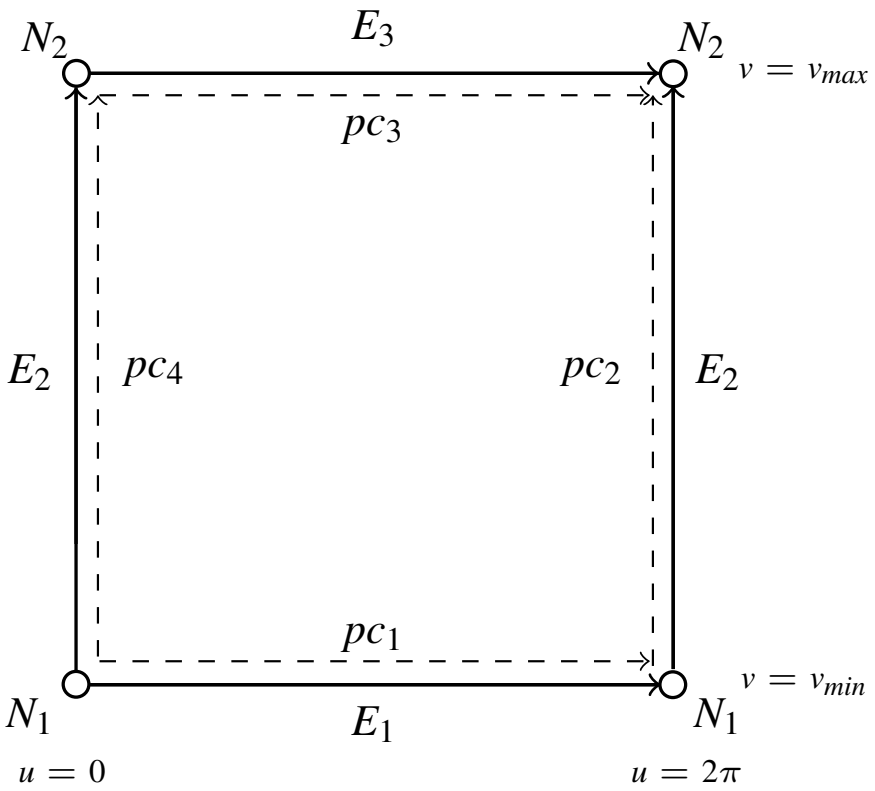
EGADS Topology Objects – Face



- 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



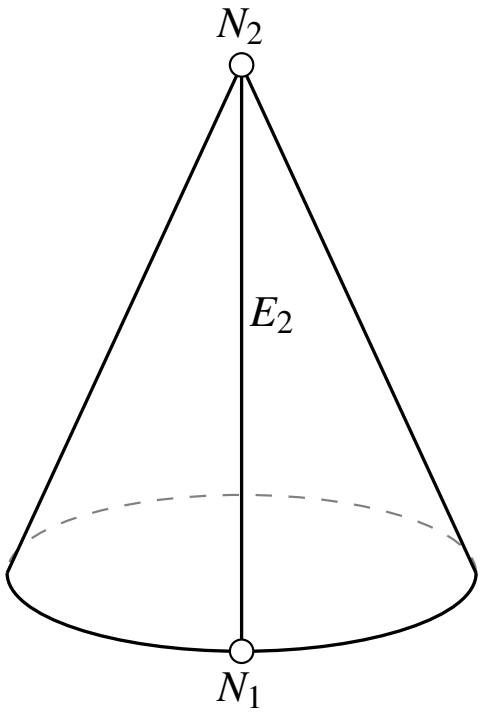


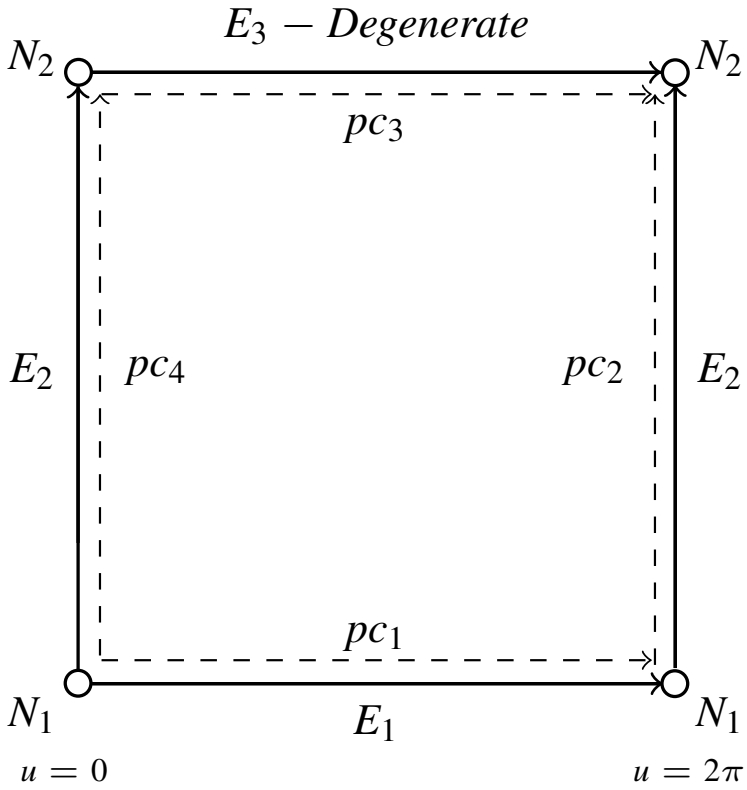
Unrolled periodic cylinder Face

Single Outer Loop – right handed/counterclockwise:

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

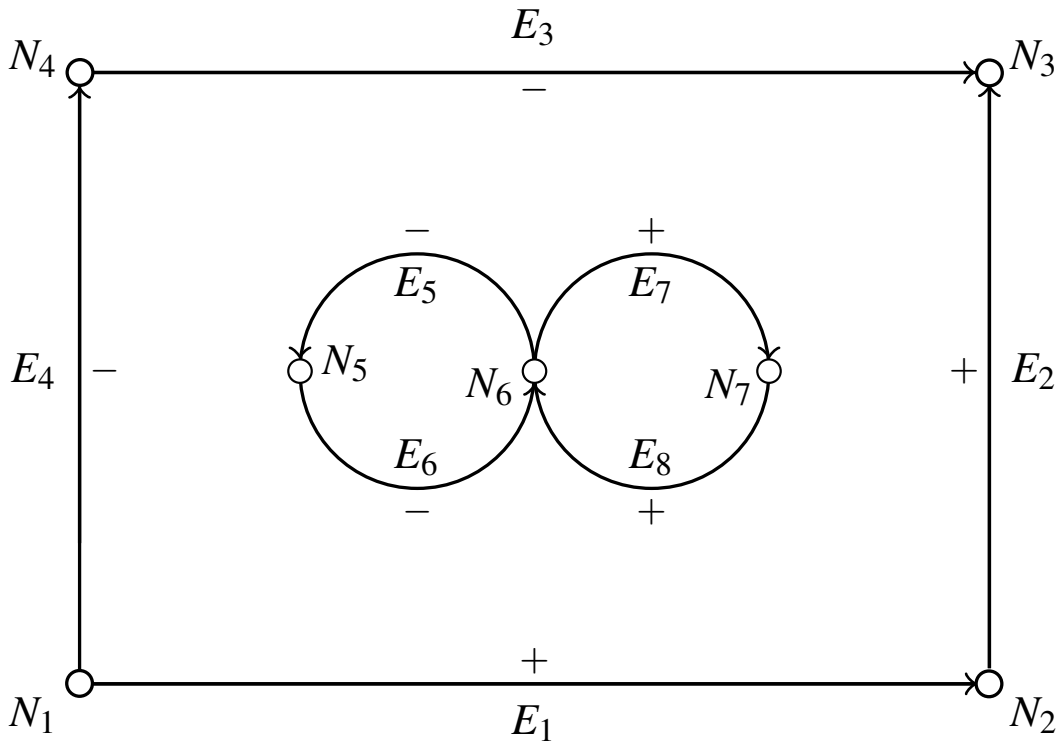
EGADS Topology Objects – Face





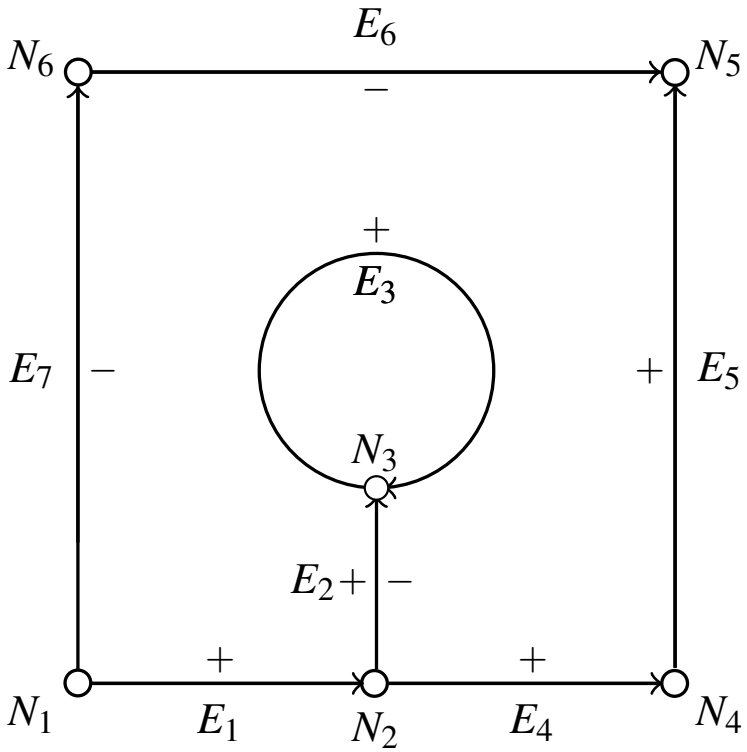
Unrolled Cone

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:

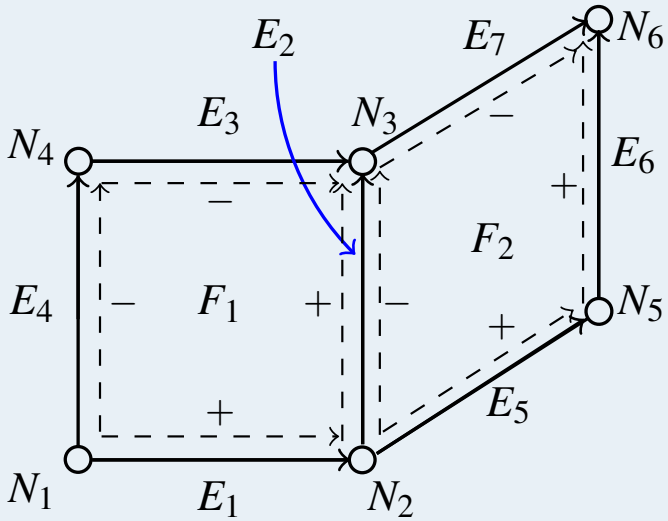
$+E_1$ $+E_2$ $+E_3$ $-E_2$ $+E_4$ $+E_5$ $-E_6$ $-E_7$

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

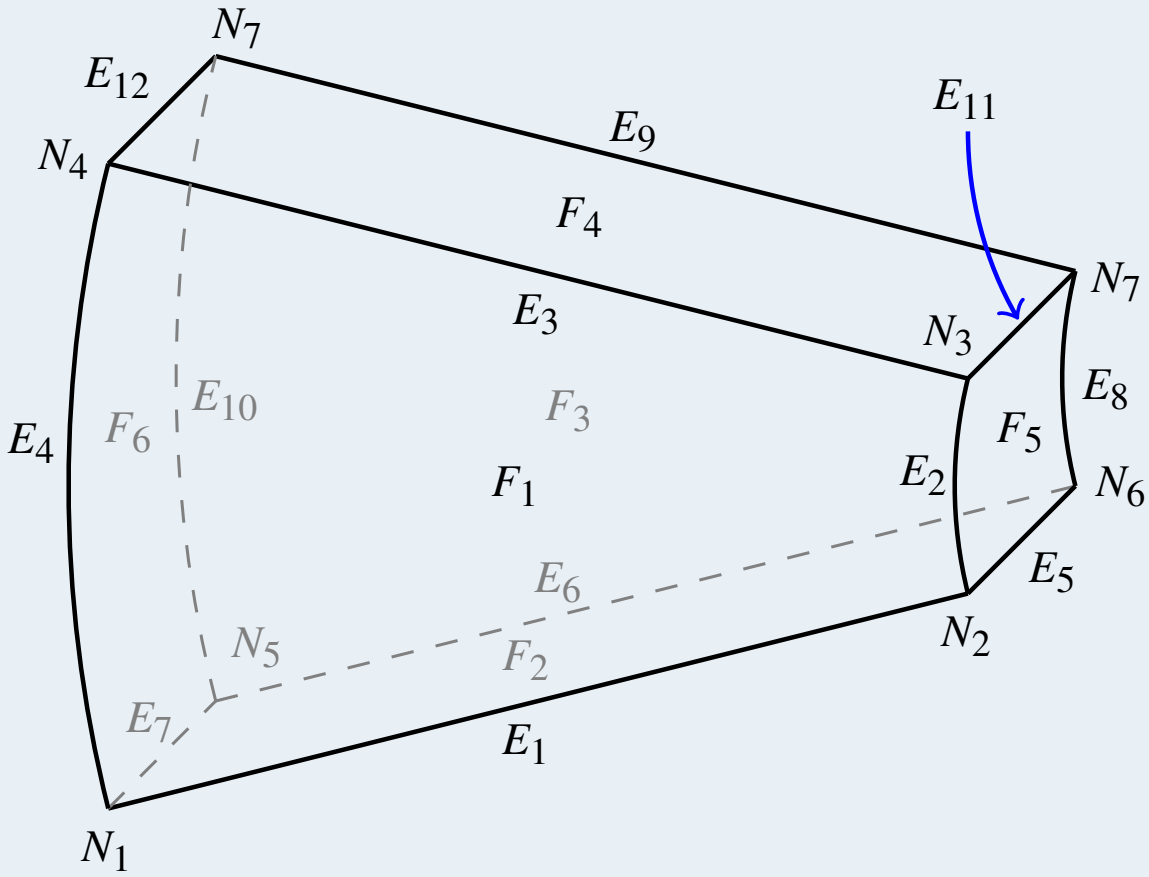
Face #2 Loop: $+E_5$ $+E_6$ $-E_7$ $-E_2$

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)

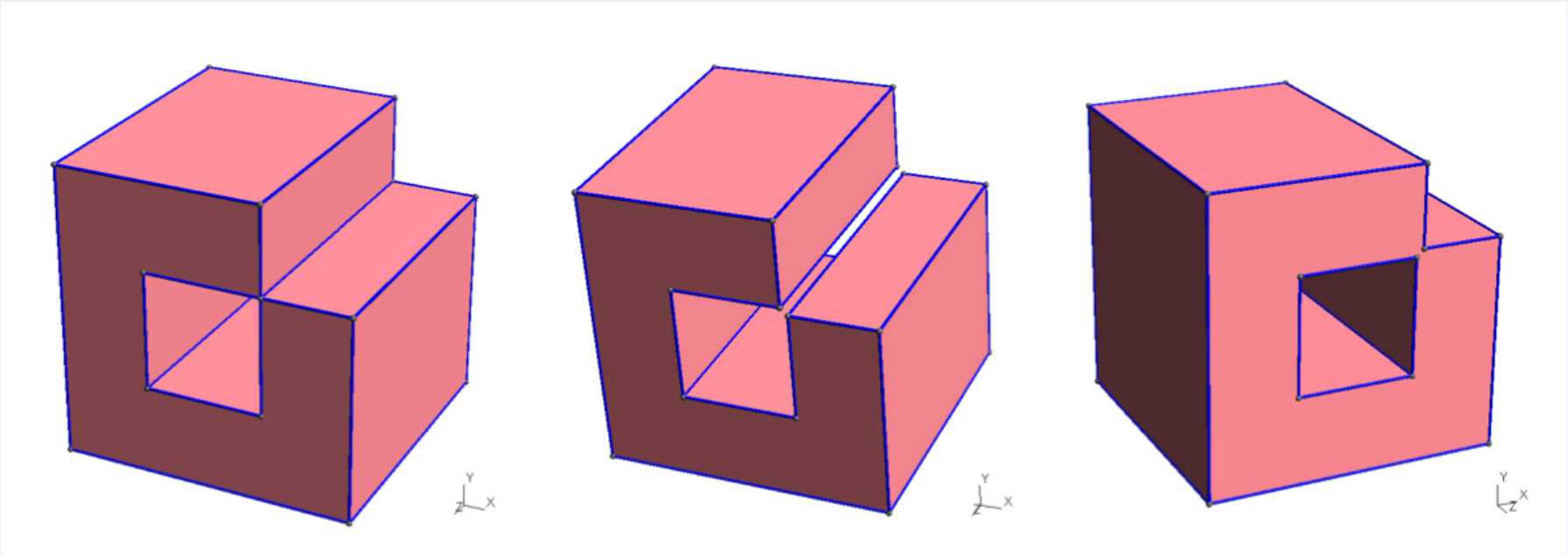
Simple **SOLIDBODY** example



8 Nodes, 12 Edges, 6 Loops and 6 Faces

EGADS Topology Objects

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



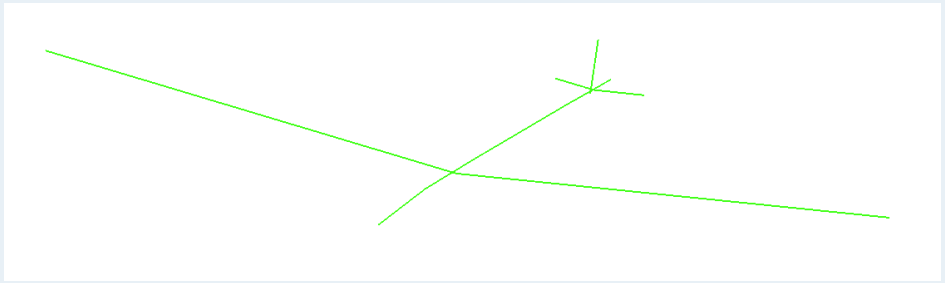
non-manifold manifold manifold

Model

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

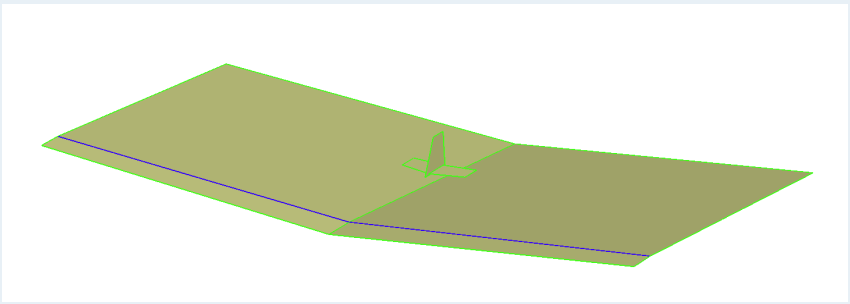
EGADS Topology Objects

Body Examples

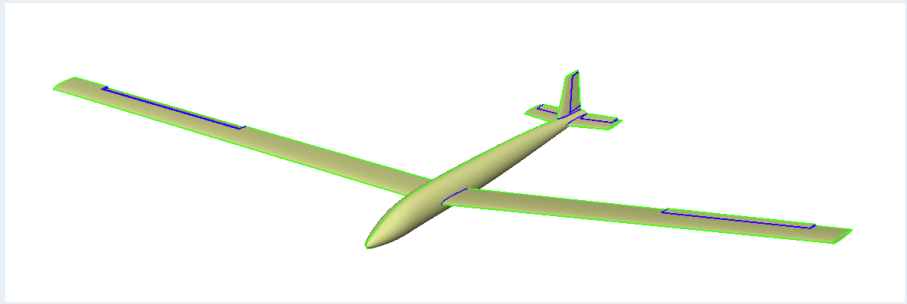


Wire Bodies

Face (Sheet) Bodies *



Sheet Bodies



Solid Body

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

EGADS Objects – 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*

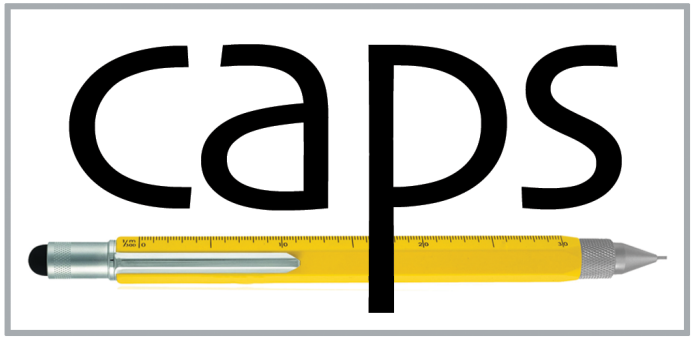
User Defined Components

- 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

...to be supplied by John

Computational Aircraft Prototype Synthesis



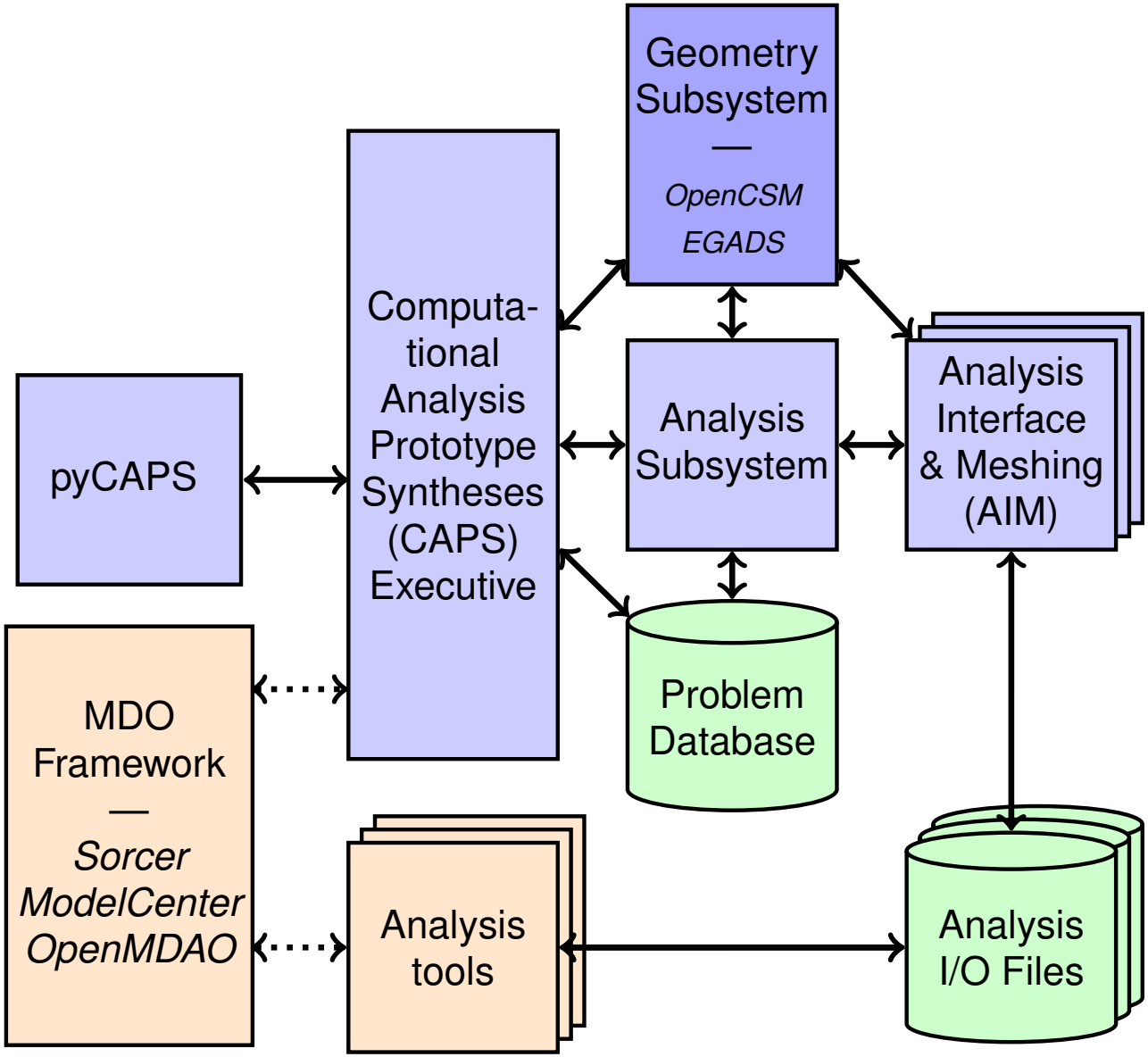
CAPS Goals

- Augment/fix MDO frameworks
- 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 main entry point into the CAPS system is the C/C++ API
- pyCAPS: Python interface for testing, demos and training

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 Bodies; 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, Branch, Parameter, User
capsAnalysis	
capsBound	

CAPS Object Hierarchy

Analysis Object

Children Objects	SubTypes
capsValue	AnalysisIn, AnalysisOut

Bound Object

Children Objects	SubTypes
capsVertexSet	Connected, Unconnected

VertexSet Object

Children Objects	SubTypes
capsDataSet	User, Analysis, Interpolate, Conserve, Builtin, Sensitivity

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)

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 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
  - 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)

- 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