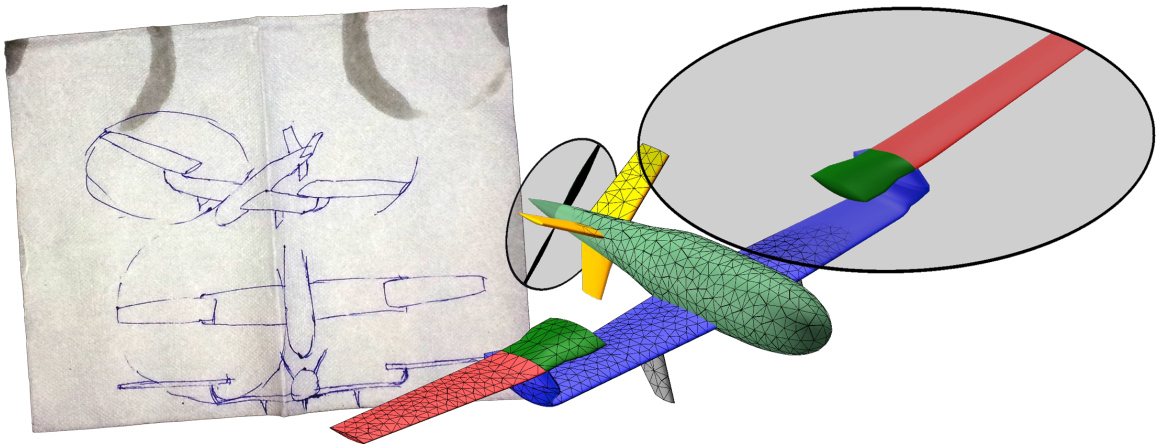


OpenVSP
Under the Hood

Rob McDonald

Not a “How To”

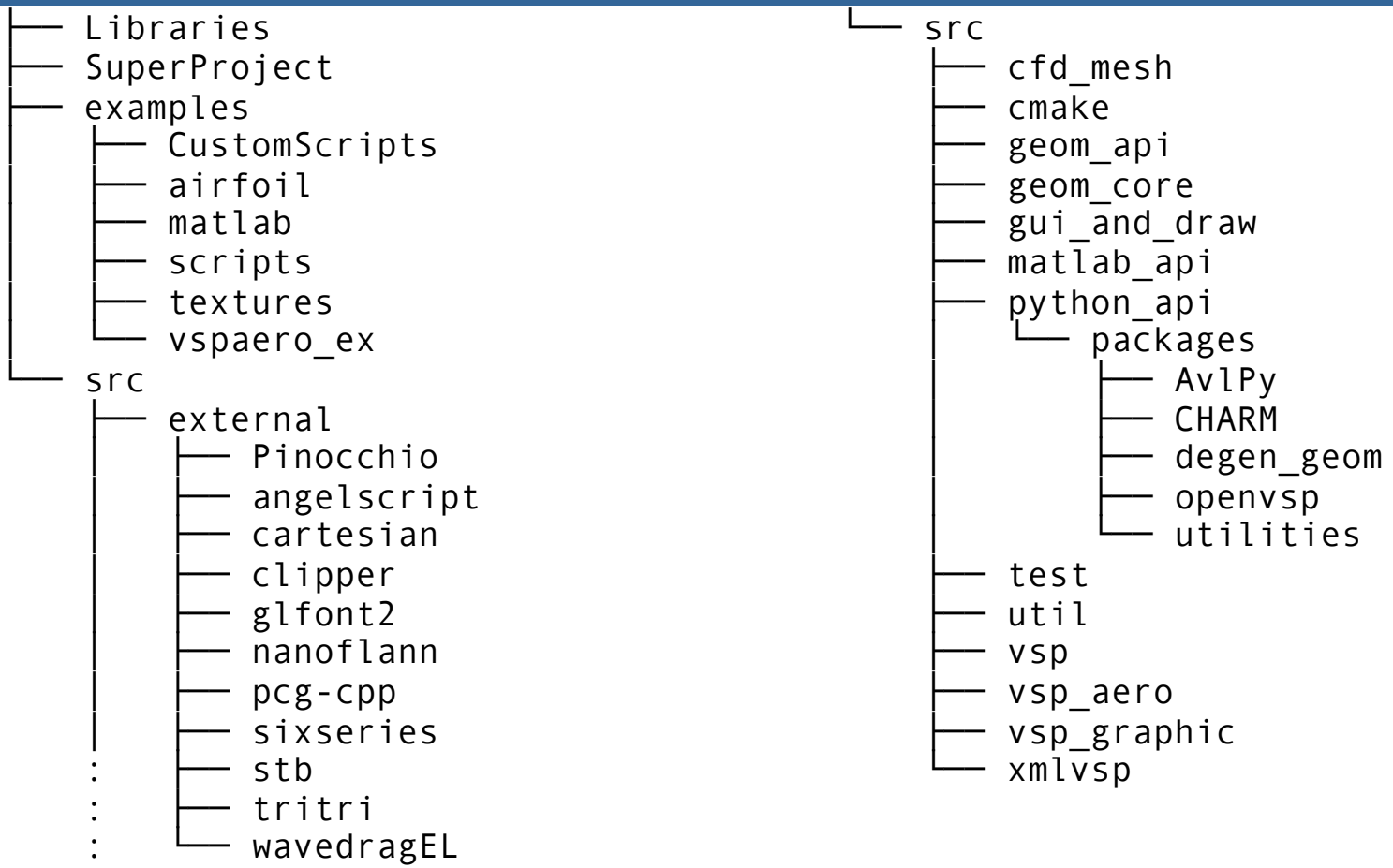
This is *not* a tutorial on how to compile OpenVSP.

Also *not* a tutorial on Git, CMake, SWIG, or anything else.

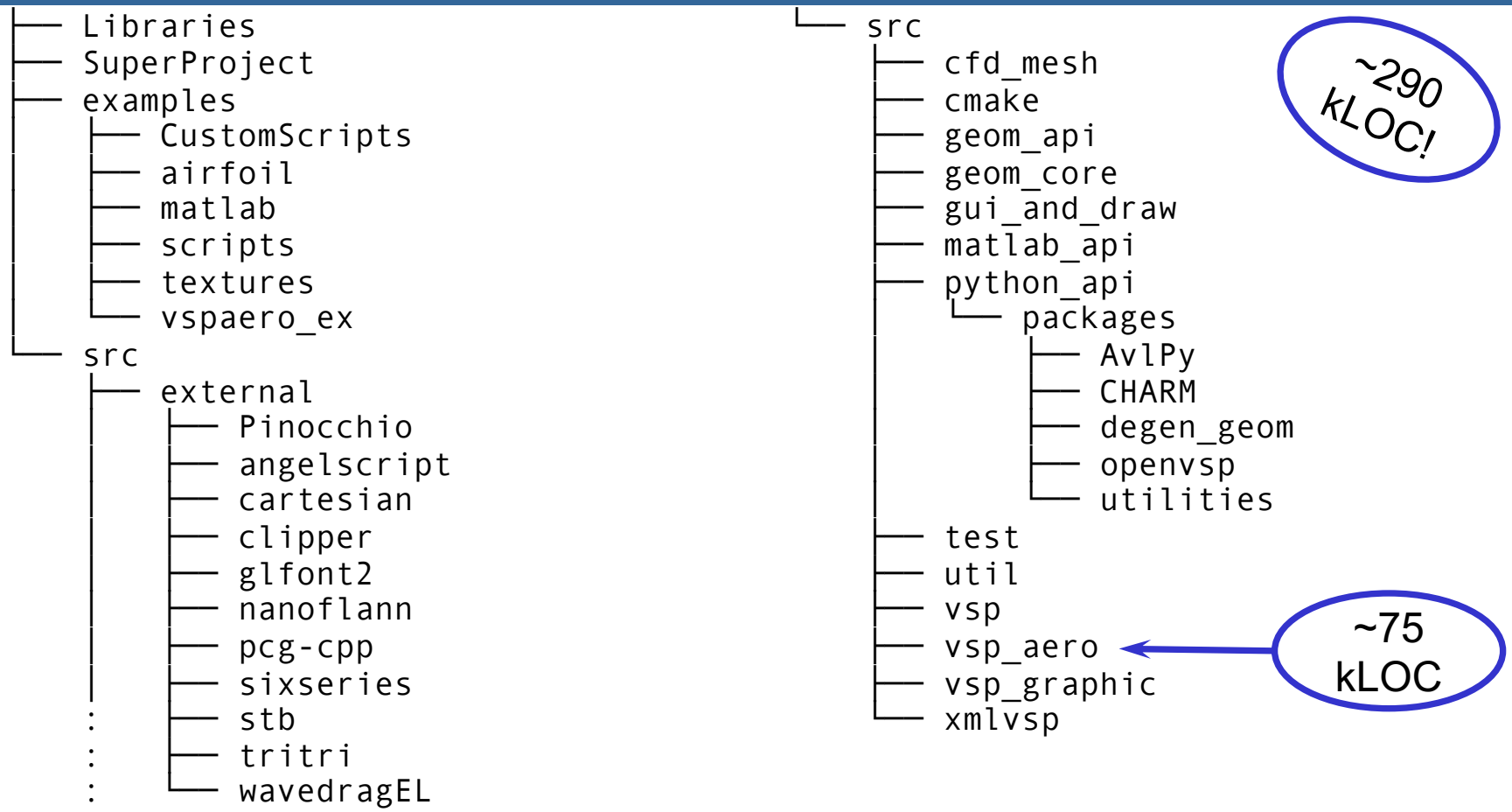
Justin Gravett did a great job of that at the 2020 Workshop (video slides).

This is a guided stroll through the source to help you find your way if you ever find yourself here again.

Repository Overview



Repository Overview



Toolchain

- CMake
  - Cross-platform meta build tool. Used to build the build files.
- git
  - Version control system
- C++ Compiler
  - Should work on any C++11 compiler.
  - Regularly built with: gcc, msvc, clang, xcode.

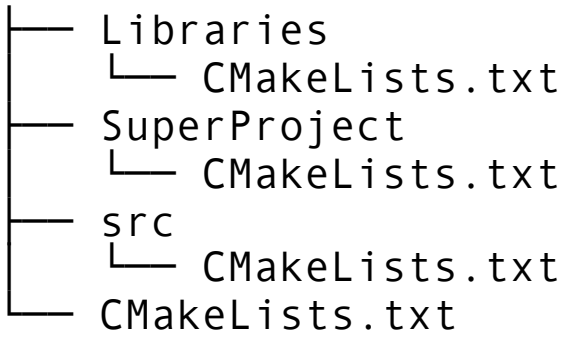
Optional

- SWIG – Simplified Wrapper Interface Generator
  - Used to make API accessible from almost any scripting language.
- Python and/or Matlab
- Doxygen
  - Prepare HTML documentation.
  - `vsp -doc`
    - APIReadme.md
    - APITestCode.vspscript
    - openvsp_as.h

Build Process Options

CMake follows instructions found in CMakeLists.txt files. You will find one of these files in almost every directory of a CMake project.

OpenVSP's CMake has four possible entry points.



External libraries only

Libraries and VSP in one SuperProject

OpenVSP (and VSPAERO etc.)

Top-level redirect to OpenVSP

If you are only compiling (not developing) OpenVSP, use SuperProject.
(SuperProject is not frequently tested. If it fails, use the other door.)

Developers should build Libraries and VSP as separate projects.

Top-level entry point will allow IDEs to 'see' Libraries, examples, etc.

“External” Libraries

Black Box	• GLEW	- OpenGL Extension manager
	• GLM	- OpenGL Math
	• Eigen3	- Linear algebra
	• CppTest	- Unit testing framework
	• CMinpack	- Levenberg-Marquardt optimizer (FitModel)
	• LibXML2	- XML
	• ~~Triangle~~	- ~~2D Delaunay mesh generator~~ (Being Replaced)
Grey Box	• LibIGES	- IGES File I/O
	• STEPCODE	- STEP File I/O
	• FLTK	- Cross platform GUI
	• Adept2	- Automatic differentiation (VSPAERO only)
	• Delabella	- 2D Delaunay mesh generator
No Box	• exprparse	- Math expression parser
	• Code-Eli	- Curve and surface library

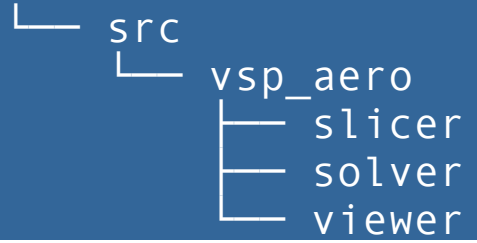
~73 kLOC!

“Internal” Libraries

└── src
    └── external

Black Box	clipper	- 2D Polygon clipping (Projected Area)
	glfont2	- OpenGL font
	Pinocchio	- Skeleton rigging (Human movement)
	stb	- Image file I/O
	pcg-cpp	- Random number generator
Grey Box	nanoflann	- Fast nearest neighbor search
	tritri	- Geometric primitives
	angelscript	- Embedded scripting language
	cartesian	- 2D Plotting for FLTK
No Box	sixseries	- NACA 6-Series airfoils
	wavedragEL	- Eminton Lord wave drag calculation

VSPAERO & Friends



VSPAERO source is entirely independent of OpenVSP.

Dave Kinney uses hand-written makefiles on Linux only.

OpenVSP team integrates VSPAERO with CMake build system.

OpenMDAO team maintains separate VSPAERO build system.

`vspviewer` uses STB image file I/O libraries and FLTK / fluid from CMake system.

`vspaero` needs compiler support for OpenMP to build multi-threaded version.

- On Windows, CMake helps us bundle required system libraries in `*.zip`.
- On Linux, OpenMP libraries are installed by default or `gcc` can statically link.
- On MacOS, OpenMP libraries are not installed and `clang/xcode`'s OpenMP does not support static linking. Consequently, we use a secondary compiler to build vspaero (solver only) on MacOS. Specified to CMake via: `-DCXX_OMP_COMPILER=/path/to/gcc`

API Bindings

```
└── src
    ├── matlab_api
    └── python_api
```

If SWIG is installed, you can build bindings for one or more scripting language.
- Bindings drawn from C++ API.
- Angelscript API maintained separately.
  - Human error can (and does) occur.

Detailed instructions by Justin Gravett in 2020 workshop (video slides).

Results in a binary file (*.dll, *.so, etc.) that provides OpenVSP functionality to scripting language via API calls.

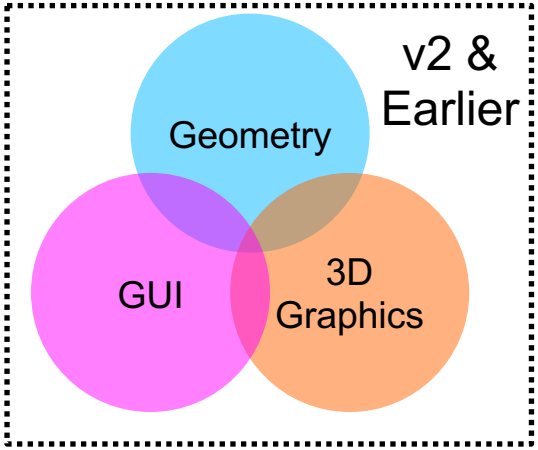
Binary file must ‘match’ the version of script interpreter.
- Some flexibility possible (e.g. Python 3.6.1 to 3.6.2).
- Not always easy to know what will break interoperation.
- Target system-installed Python 3 on Ubuntu.
- OpenVSP currently targets Python 3.6 & 3.9 on Windows & Mac.
  - Should we update?

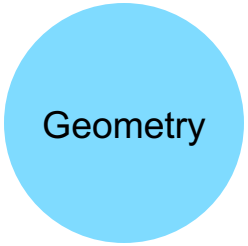
Architecture

```
└── src
    ├── cfd_mesh
    ├── geom_api
    ├── geom_core
    ├── gui_and_draw
    ├── util
    ├── vsp
    ├── vsp_graphic
    └── xmlvsp
```

Provides C++ API

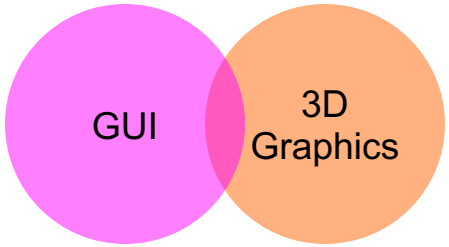
Provides main()






```
vspbatch        main()
apitest         main()
geom_api        no-op StartGui()
```

HPC &
Non-graphics
Applications




```
vsp             main()
apitest_g       main()
geom_api_g      real StartGui()
```

“Built-In” Libraries

```
└── src
    ├── util
    └── xmlvsp
```

- xmlvsp
  - Simple wrappers to make libXML2 easier to use.
  - Note, there is no XML ‘Schema’.
    - There is no defined/documented ‘file format’.
      - Files highly dynamic (e.g. custom components).
    - Reading / writing the XML file is strongly discouraged.
      - Use API instead.
- util
  - Collection of many ‘small’ classes and routines.

`vec3d (vec2d)`	Point/vector
`Matrix4d`	Transformation matrix
`VspCurve (Vsp1DCurve)`	Piecewise Bezier curve
`VspSurf`	Piecewise Bezier surface
`MessageMgr`	Message passing
`DrawObj`	3D graphics data package
`(many more)`	

FooMgr

Our convention for the singleton pattern. Not unlike a big global variable.

`VehicleMgr, ParmMgr, AdvLinkMgr, AnalysisMgr, FitModelMgr`

And many others…

```
class FooMgrSingleton{
private:
    FooMgrSingleton();
    FooMgrSingleton( FooMgrSingleton const& copy ) = 0;
    FooMgrSingleton& operator=( FooMgrSingleton const& copy ) = 0;
public:
    static FooMgrSingleton& getInstance(){
        static FooMgrSingleton instance;
        return instance;
    }
};
#define FooMgr FooMgrSingleton::getInstance()
```

gui_and_draw

└── src
    └── gui_and_draw

View & Controller of MVC.

- Model is provided by `geom_core`.
- Often via singletons.
- `Update()` is slash and burn.

Programmatically generated GUI's.

- OpenVSP does not use `fluid` or *.fl files (`vspviewer` does).

`GuiDevice`	Enhanced widgets
`GroupLayout`	GUI layout and design helpers
`ScreenBase`	Screen base classes (GeomScreen)
`ScreenMgr`	'Owns' all screens, manages update
`MainVSPScreen`	Main window with menus
`ManageGeomScreen`	Geom browser GUI
`MainGLWindow`	3D Graphics portal
`fooScreen`	'foo' Screen implementation

vsp_graphic

Interface to graphics hardware via OpenGL.

Will need to be re-written as OpenGL is deprecated (Direct3D, Metal, Vulkan).

`void UpdateDrawObj();`

Called by `Update()`. Manipulate large data. Sets `m_GeomChanged=true;`

`void VspGlWindow::update();`

Calls `LoadDrawObj()`. Calls `_update()`. Sets `m_GeomChanged=false;`

`void LoadDrawObj( vector< DrawObj* > & draw_obj_vec );`

Transfers `DrawObj`. Can make small adjustments – color or shade vs. wire.

`void VspGlWindow::_update( std::vector<DrawObj *> objects );`

Creates Entity corresponding to DrawObj.

`string DrawObj::m_GeomID;`

Unique identifier of `DrawObj` (not Geom). Used to prevent un-needed transfers.

`bool DrawObj::m_GeomChanged;`

Flag to trigger update of bulk data.

`util/DrawObj`	Transports graphics data from geom_core
`Entity`	Responsible for drawing
`Scene`	Manages drawing entities in scene

cfd_mesh

Implements ‘CFD Mesh…’, ‘FEA Mesh…’, and ‘Trimmed Surfaces…’.

Surface-surface intersection.

Unstructured isotropic surface mesh generation.

Once a part of `geom_core`, but separated to de-clutter.

- Expect more such separation in the future.
- Should be developed as a stand-alone library to facilitate unit testing.

geom_core

Everything else.

- Much of this should be moved to various subdirectories.
- Should also be refactored into libraries with API's for better unit testing.

Infrastructure

`ParmMgr, Parm, ParmContainer, Vehicle, AnalysisMgr, ResultsMgr`

Geometry

`Geom, PodGeom, HingeGeom, PropGeom, WingGeom`
`XSecSurf, XSec, XSecCurve, Airfoil`

Modeling

`DesignVarMgr, VarPresetMgr, AdvLinkMgr, SnapTo, FitModelMgr`

Analysis

`MeshGeom & TMesh` (CompGeom, Mass Prop, Area Slice, Wave Drag Slicing)
`WaveDragMgr, ParasiteDragMgr, VSPAEROMgr, ProjectionMgr`

Other

`Light, Measure, Texture, Material`
`CustomGeom, ScriptMgr` (Angelscript API)

Infrastructure

Infrastructure

`Parm`	Parameter to be varied, has unique `m_ID`
`ParmContainer`	Thing that has `Parm's`, also has unique `m_ID`
`ParmMgr`	Keeps track of all `Parms`
`Vehicle`	Overall model, is a `ParmContainer`

When a `Parm` is changed, the model does not `Update()` immediately. Instead, `ParmContainer::ParmChanged()` is called to notify the `ParmContainer` which will likely set `m_LateUpdateFlag = true;`.

Later, an `Update()` cycle will traverse everything and will set `m_LateUpdateFlag = false;`.

Infrastructure

└── src
    └── geom_core

"Long ago", analysis tools were added to OpenVSP in an ad-hoc basis. Each one extended the API and command line syntax in unique and incompatible ways. Each one added a new output file. A unified approach was required. Inputs and outputs are handled through a name/data/doc triplet approach

`NameValData` New! Name, Val, Doc
`NameValCollection`

Outputs are placed in a `Results` class managed by `ResultsMgr`. Results are handled through a similar name/data/doc triplet

`Results`
`ResultsMgr`

Analyses only need to implement two methods. `AnalysisMgr` handles their inputs and execution as well as the analyses themselves.

```
Analysis::SetDefaults();
Analysis::Execute();
AnalysisMgr::RegisterBuiltins();
```

Geom::Update

`ParmContainer`
`GeomBase`
`GeomXForm`
`Geom` — Main geometry base class
`Geom::Update();` — Handles processing of `Parms` into a geometry

`Update()` Uses a fine-grained approach with caching of intermediate results. More granularity may be implemented in the future.

`GeomBase::SetDirtyFlags();` — Routine to classify `ParmChanged`

```
bool m_SurfDirty;
bool m_XFormDirty;
bool m_TessDirty;
bool m_HighlightDirty;
bool m_FeaDirty;
```

Shape & everything else
Position, rotation, symmetry
Tessellation, clustering
Active XSec
Structure parts

Geom::UpdateSurf()

`Geom`	Main geometry base class
`Geom::Update();`	Handles processing of `Parms` into a geometry
`FooGeom::UpdateSurf();`	Routine to populate `m_MainSurfVec`

`m_MainSurfVec` Contains a single (no symmetry) un-transformed copy of the surfaces constituting a geometry.

```
vector <VspSurf> m_MainSurfVec;

piecewise_surface_type VspSurf::m_Surface;

typedef eli::geom::surface::piecewise<eli::geom::surface::bezier, double, 3> piecewise_surface_type;
```

`Geom::ApplySym(v1, v2)` Populates `v2` with symmetrical copies of `v1` transformed to their final location.

```
vector <VspSurf> m_SurfVec;
ApplySym( m_MainSurfVec, m_Surfvec );
```

To add a Geom

```
└── src
    ├── geom_core
    └── gui_and_draw
```

`Geom`	Main geometry base class

To implement built-in `fooGeom`, you inherit new classes from `Geom` and `GeomScreen`:

```
class FooGeom : public Geom;
class FooScreen : public GeomScreen;
```

And then implement these methods:

`FooGeom::FooGeom();`	Constructor, sets up `Parms`
`FooGeom::UpdateSurf();`	Routine to populate `m_MainSurfVec`
`FooGeom::ComputeCenter();`	Routine to calculate `m_Center`
`FooGeom::Scale();`	Routine to scale dimensional `Parms`
`FooGeom::AddDefaultSources();`	Routine to add CFD Mesh default sources
`FooGeom::OffsetXSecs();`	Routine to calculate `Parms` for `ConformalGeom`
`FooScreen::FooScreen();`	Constructor, sets up `GuiDevices` in `Screen`
`FooScreen::Update();`	Update screen to latest `Parms`
`Misc boiler plate`	Stuff to plug `Geom` into everything

Recommend you start by copying (and studying) `PodGeom`.

GeomXSec

Some geometries need to work with a collection of cross sections.

```
class GeomXSec : public Geom;
    FuseGeom, StackGeom, WingGeom, PropGeom
```

`GeomXSec` has a `m_XSecSurf` that is a container for `XSecs`. It handles add, copy, paste, delete, etc. of `XSecs`.

```
class XSecSurf : public ParmContainer;
```

The `XSec` class handles positioning (rotate, translate) and also has a `m_XSecCurve`.

```
class XSec : public ParmContainer;
    FuseXSec, StackXSec, WingSect, PropXSec
```

GeomXSec (cont.)

The XSec class handles positioning (rotate, translate) and also has a m_XSecCurve.

```
class XSec : public ParmContainer;
    FuseXSec, StackXSec, WingSect, PropXSec
```

The XSecCurve is analogous to a 2D Geom. It translates Parm values into a VspCurve, m_Curve.

```
class XSecCurve : public ParmContainer;
    PointXSec, EllipseXSec, RoundedRectXSec, EditCurveXSec, etc.
```

Airfoils are XSecCurves with concepts of chord & t/c instead of width & height.

```
class Airfoil : public XSecCurve;
    FourDigMod, OneSixSeries, SixSeries, FileAirfoil, CSTAirfoil, etc.
```

Questions?