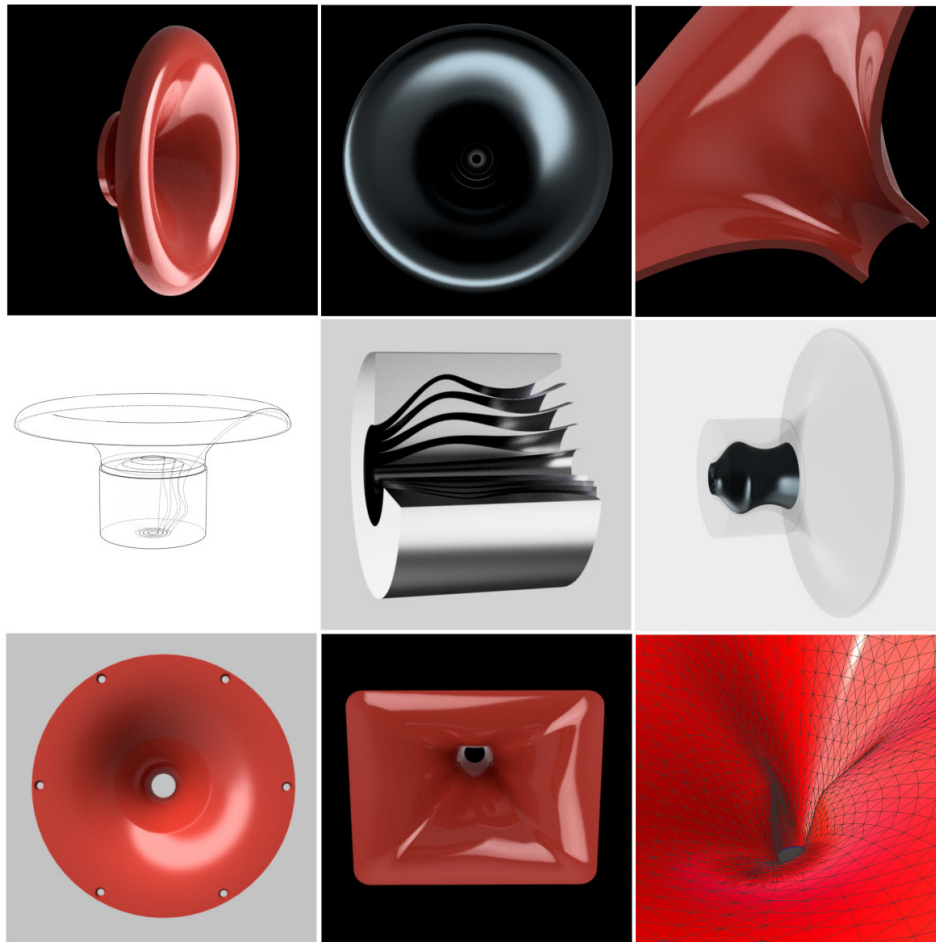


Ath 4



User Guide

Release 4.8.2

ATH - Advanced Transition Horns

Copyright © 2019-2022 Marcel Batík

<https://at-horns.eu>

Release date: 03/2022

Contents

1 Introduction.....	5
1.1 Installation.....	5
1.1.1 Prerequisites.....	5
1.1.2 Program deployment.....	6
1.1.3 Adapting the configuration file.....	6
1.1.4 Running the program.....	7
1.2 Workflow overview.....	9
2 Designing a waveguide.....	10
2.1 Overview on the geometry.....	10
2.1.1 Profile types.....	10
2.1.2 Explicit shape definition.....	10
2.1.3 Implicit coverage definition (guiding curve).....	11
2.1.4 Shape of mouth outline.....	11
3 Using BEM analysis.....	12
3.1 The Mesh.....	12
3.1.1 Subdomains and interfaces.....	13
3.2 Source definition.....	14
3.3 Geometry grid - terminology.....	15
3.3.1 Meshing the slices.....	15
3.4 Circular symmetry mode.....	16
4 Horn definition file.....	17
4.1.1 Horn Geometry.....	18
4.1.2 Morph feature.....	19
4.1.3 Mouth rollback.....	19
4.1.4 Mesh generator.....	20
4.1.5 ABEC/BEM Project Settings.....	22
4.1.6 Program Output.....	24
5 Exporting to CAD.....	26
5.1 Export.....	26
5.2 Import into Autodesk Fusion 360.....	26
6 Tutorial.....	27
6.1 The basics.....	27
6.2 Running the program.....	29
6.3 Running BEM analysis.....	31
6.4 Refining the profile.....	35
6.4.1 Explicit definition.....	35
6.4.2 Using guiding curves.....	37
6.4.3 Using the morph feature.....	39
6.5 Modelling a complex source.....	40
6.5.1 External Shaping Plug.....	41
6.6 Using circular arc profiles.....	42
6.7 Setting subdomain interfaces.....	43
6.8 Free standing horns.....	46
6.8.1 Mouth rollback.....	47
6.9 Simulating a source without a waveguide.....	48
6.10 CircSym mode.....	50
6.11 CircSym mode (free standing case).....	53
6.12 Adding enclosures.....	55
6.12.1 Pre-defined enclosure.....	55

6.12.2 User-defined enclosure plan.....	57
6.13 Adding lumped element models.....	59
6.14 Reporting Results.....	60
7 Disclaimer.....	62
8 References.....	62
Appendix A.....	63
Appendix B.....	64
Appendix C.....	70

1 Introduction

The program 'Ath' originated as a generator of geometries for acoustic horns and waveguides. It can also generate complete ready-to-use projects for the R&D Team's software ABEC/AKABAK, a 3D acoustic boundary element method (BEM) solver.

Custom scripts for Autodesk Fusion 360 are available to import the generated shapes as smooth surfaces for a subsequent high-quality CAD modeling and/or 3D printing.

Ath is available for download at <https://at-horns.eu> as a freeware for personal non-commercial use.

This document serves both as a reference and a tutorial:

Chapters 2 and 3 give an overview of the basic principles.

Chapter 4 is the complete reference for the *horn definition file*.

Chapter 6 gives a tutorial in a form of a guided tour through the most of the program features.

1.1 Installation

1.1.1 Prerequisites

The program runs on Microsoft Windows operating system (tested on Windows 7, 10).

The following software is not strictly necessary to run Ath but makes a vital part of what the program was designed to cooperate with. All of it is available also as freeware for personal use (with various limitations):

- Gmsh - mesh generator (available at <https://gmsh.info>)
- ABEC/AKABAK + VACS for BEM analysis of the horn acoustics
Available at <http://www.randteam.de>
- Gnuplot - <http://www.gnuplot.info> (optional for data visualization)
- STL viewer (e.g. Autodesk Meshmixer - <https://www.meshmixer.com>)
- CAD (e.g. Autodesk Fusion 360 - <https://www.autodesk.com/products/fusion-360>)

Note: The external mesh generator is not needed for axisymmetric devices and in that case it can be left out completely, if there's no need to generate STL files.

1.1.2 Program deployment

The software is deployed on a target PC manually, there's no automatic installer. All that is needed is to extract the files from the distribution package anywhere on a disk drive and manually adapt the main configuration file (1.1.3):

1. Download the zip package from ATH website¹ and extract its contents to an arbitrary destination directory on your hard drive.
2. Create a root directory for your projects. The program will store all its output under this directory in automatically created subdirectories, one per project.
3. Adapt the global configuration file *ath.cfg* to your environment (see 1.1.3).

Although Ath is an application for Windows OS, the program has no graphical user interface (i.e. no own window) and must be run from a command line window (*cmd.exe*; normally located in *C:\Windows\System32*) by typing commands manually. It is out of scope of this document to describe the general work with the command line. See 1.1.4 for more details about how to run the program.

1.1.3 Adapting the configuration file

Before using the program a global configuration file *ath.cfg* must be adapted to the local environment. The file is located in the same directory as the main executable (*ath.exe*). It is read each time the program is executed. It's an ordinary ASCII text file and must be edited in a plain text editor. The syntax is the same as for the horn definition file described in Chapter 4.

Output root directory

Item **OutputRootDir** sets a target directory where the output of the program will be stored. For each project a subdirectory with the same name as the project definition file will be automatically created. The user is responsible for providing the root directory (e.g.):

```
OutputRootDir = "D:\Horns"
```

Connecting mesh generator

The program doesn't contain its own meshing routines and relies on a user-provided external application that does the actual meshing. One of such high performance and readily available generators is Gmsh². For this reason Ath utilizes Gmsh geometry script format (*.geo) as its default output. To connect Ath to a meshing program, path to the mesh generator executable must be set as **MeshCmd**. In the case of Gmsh, this would be (e.g.):

```
MeshCmd = "D:\gmsh-4.6.0-windows64\gmsh.exe %f -"
```

Note the placeholder "%f" - it will be automatically replaced by the actual geo filename.

¹ <https://www.at-horns.eu/download.html>

² <https://gmsh.info>

Connecting gnuplot

To make use of a graphing output incorporated in Ath, path to gnuplot³ executable must be set:

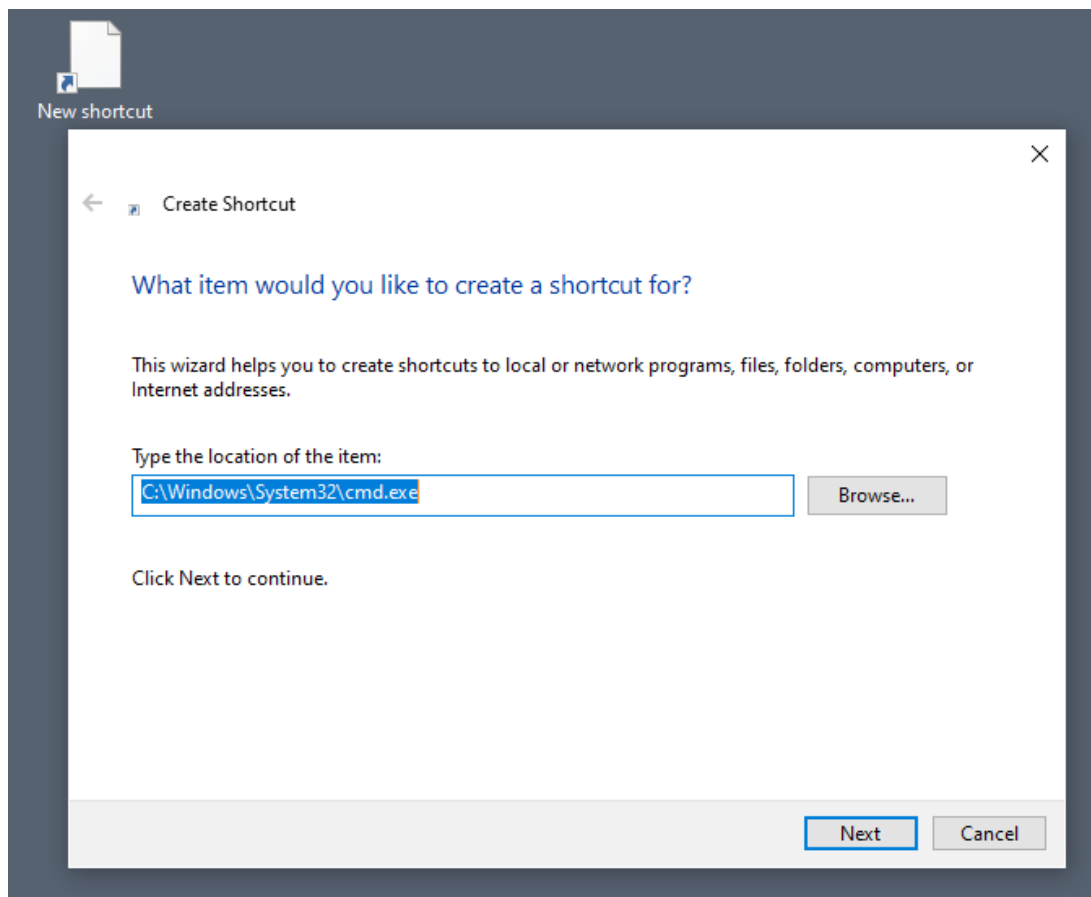
GnuplotPath = "C:\Program Files\gnuplot\bin\gnuplot"

1.1.4 Running the program

Perhaps the most convenient way of working with Ath is to create a Windows shortcut to *cmd.exe* and set the "Start in" property of the shortcut to the Ath program directory.

Prepare the environment - Step 1)

In Windows, either on Desktop or in File Explorer, click right mouse button and select "New -> Shortcut". In the "Create Shortcut" window click a "Browse..." button and select the path to the *cmd.exe* file, typically located in the directory C:\Windows\System32:

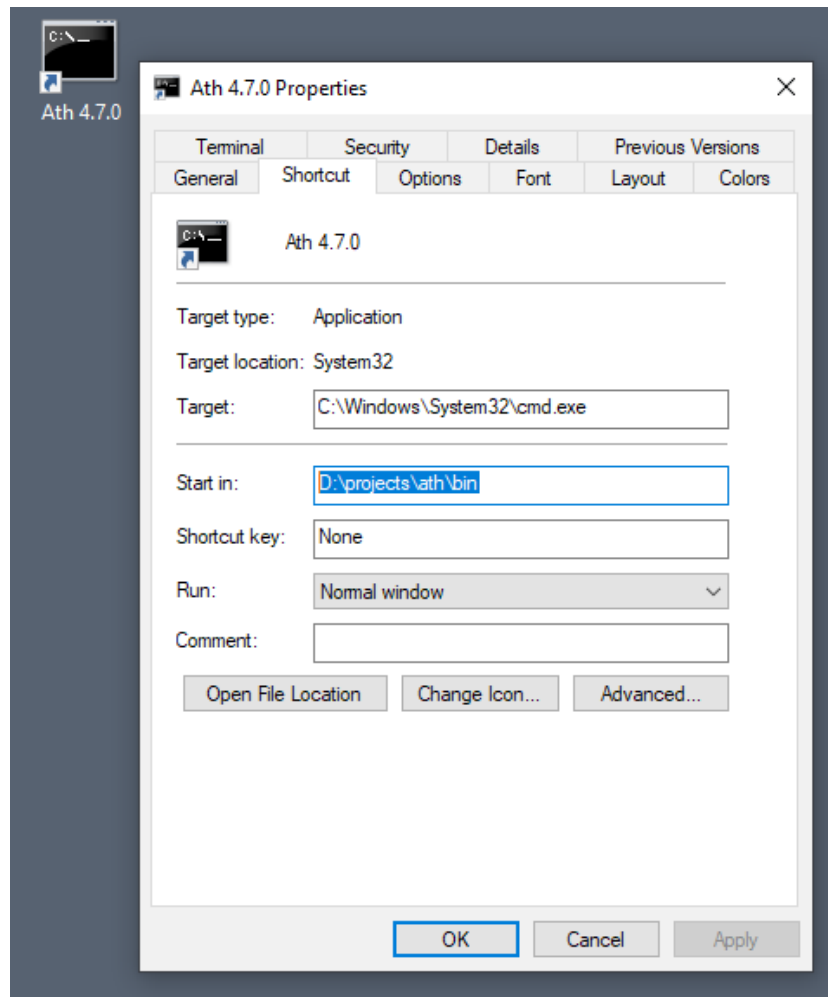


Click next and type a name of your choice for the new shortcut (e.g. "Ath 4.8.2"). Then click "Finish".

³ <http://www.gnuplot.info>

Step 2)

Right-click on the newly created shortcut, select "Properties" and type the path to the Ath executable, i.e. where you extracted the Ath package (*ath.exe*), to the "Start in" field:



You are now ready to run Ath by double-clicking on the shortcut icon (still note that this will lead you to the command line only where commands must be typed manually).

1.2 Workflow overview

This is how a typical work with the tool is organized. A tutorial is given in Chapter 6.

1. To define a waveguide geometry, so called *definition file* must be prepared first. Most easily an existing (demo) file can be adapted. This file stores the complete description of the geometry and all the related project parameters - more on this in Chapter 4.
2. The program **ath.exe** is executed with a horn definition filename as its parameter. If connected to a mesh generator (1.1.3), the generator will be executed automatically. Output files are stored in the dedicated directory (1.1.3): STL/MSH file, ABEC project, ASCII coordinate files, etc.
3. The generated mesh/STL files are visually inspected in a favourite 3D viewer (Meshmixer, etc.)
4. The generated BEM project is opened in ABEC and a numerical analysis performed. Steps 1 - 4 are repeated until fully satisfied with the results.
5. The final shape can be imported into CAD as a smooth surface for a subsequent processing or the generated STL file used directly to prepare the manufacturing of the actual device.

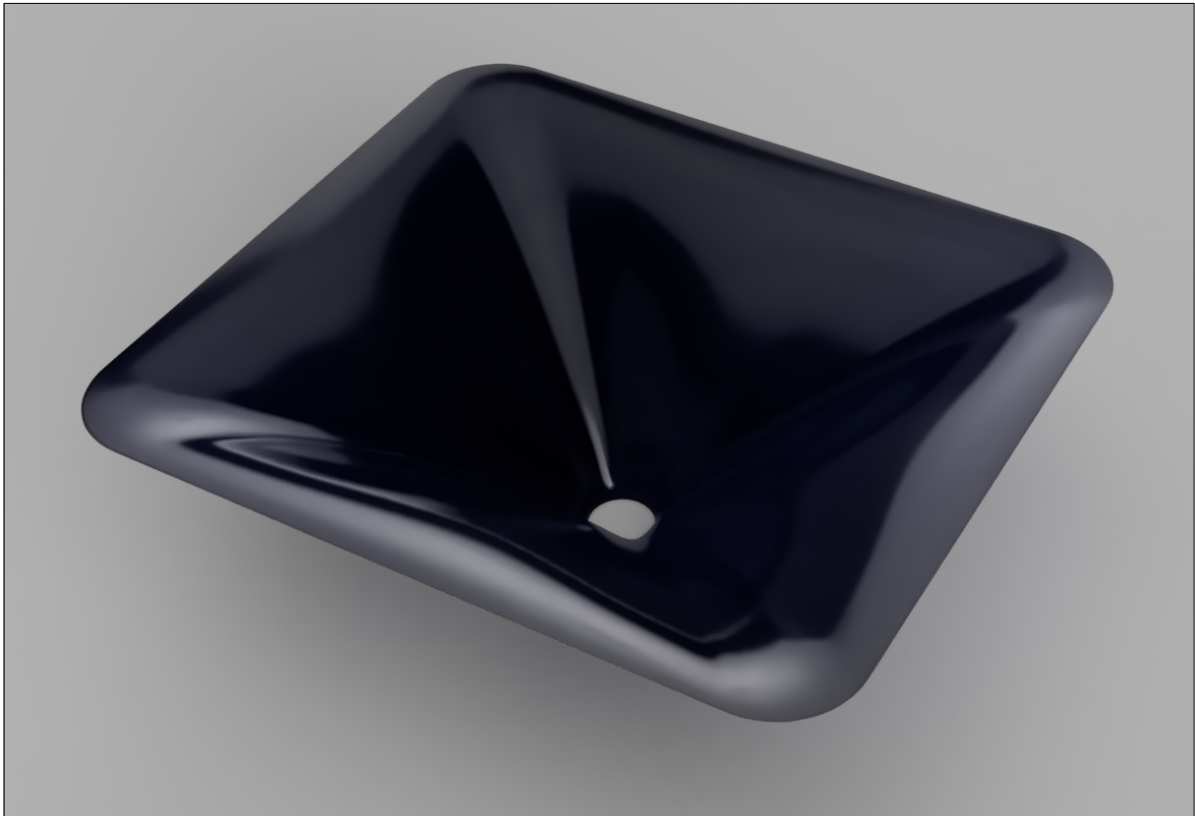


Fig 1: High-performance waveguide designed with Ath (Fusion 360 visualization)

2 Designing a waveguide

2.1 Overview on the geometry

This chapter gives a short overview of the basic principles of describing a waveguide shape. At this point the reader is highly encouraged to go through the reference [1] to get familiar with the OS-SE profile formula used - much of what can be achieved is done through manipulation of the profile formula parameters. Even though the tool makes it possible to design a shape without any mathematics, some of the flexibility is sacrificed if used that way.

Horn⁴ profile can be defined either a) explicitly by writing mathematical expressions of the profile formula parameters, or b) implicitly, and maybe more intuitively, by using a concept of *guiding curve*.

2.1.1 Profile types

The following profile options are currently available⁵:

- | | |
|------------------------|--------------------------------------------------------------------------------------------------------|
| a) <i>OS-SE</i> | Composed profile: hyperbola + superelliptical termination (see [1])
<code>Throat.Profile = 1</code> |
| b) <i>Circular arc</i> | A pure circular arc from throat to mouth.
<code>Throat.Profile = 3</code> |

2.1.2 Explicit shape definition

Explicit definition gives the most flexibility. Virtually every profile parameter can be expressed freely as a mathematical formula, using a parameter **p** which represents the angle around the horn (i.e. 0 - 360°).

For example the following expression can be used for a variable coverage angle:

```
Coverage.Angle = 40 + 10*cos(p)^2 ; [deg]
```

For complete list of available standard functions see Appendix A.

Explicit definition is used whenever there's no definition of a *guiding curve* in the definition file (see the next section).

⁴ Note that the words 'horn' and 'waveguide' are used interchangeably in this document.

⁵ There was a separate conical profile in the previous versions of the tool but with the introduction of the 'k' parameter into the OS-SE formula, this became superfluous - conical profile can be set with k = 0.

2.1.3 Implicit coverage definition (guiding curve)

Guiding curve is a virtual closed loop of wire that the horn surface goes through at some defined distance from the throat (see Fig.1).

In the current implementation the coverage angle around the waveguide is automatically calculated so that each profile goes exactly through the guiding curve at a defined distance from the throat. Shape of the guiding curve thus intuitively defines the overall shape of the horn.

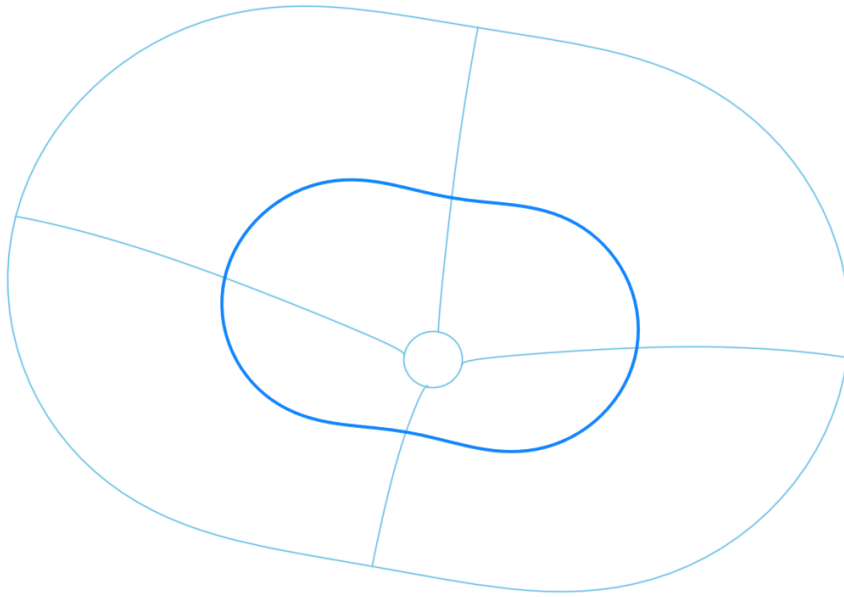


Fig 2: Horn with a guiding curve (the bold line)

Although the shape of a guiding curve could be completely arbitrary, two basic curves are implemented at the moment: *superellipse*⁶ and *superformula*⁷, the later alone giving a huge number of shapes possible.

2.1.4 Shape of mouth outline

For the mouth outline shape, two possibilities are implemented:

- a) fully determined by the profile parameters (either via a guiding curve or an explicit definition),
- b) forced to be a rectangle or a circle by morphing the shape resulting from option a).

See the `Morph.*` parameters.

⁶ <https://en.wikipedia.org/wiki/Superellipse>

⁷ <https://en.wikipedia.org/wiki/Superformula>

3 Using BEM analysis

3.1 The Mesh

In this context a mesh is a discretized surface description for the purpose of numerical analysis. BEM solver takes a boundary mesh of the analyzed object as its input. Ath doesn't generate meshes directly but relies on external meshing applications, like Gmsh. The Ath program produces a mesh description in a form of a Gmsg script (*.geo) file. This file is then processed by the meshing generator that produces the actual mesh files, either in STL or MSH (Gmsh) format. However is this processing hidden from the user for the most part, there are still important parameters that should be set properly and the user should be familiar with them to make the best of the analysis. This chapter provides an overview.

The most important parameter is the maximum size of the individual mesh elements. First of all, this sets a limit on the highest frequency of the analysis – the elements must be definitely smaller than half-wavelength at the highest frequency of interest, and typically a considerably smaller size is needed to get reliable results: to reach 10 kHz mesh resolution between 5 - 8 mm is already good enough in many practical situations. The elements should also be small enough to capture the finer details of more complex shapes. It has been found in practice that the element size can increase further from the throat without sacrificing the precision too much and Ath offers this possibility. The finer the mesh, the longer the calculation takes, so it's always a balancing act.

There are several parameters that define mesh resolution, i.e. the maximum size of a mesh element. See items `Mesh.*Resolution` for the details. The mesh resolution doesn't have to be constant across the whole surface and there are several parameters for this resolution in different significant areas of the horn. If set differently, mesh resolution changes gradually in between.

The generated ABEC project takes a mesh file in MSH format version 2.2. By default this is the instruction in the *.geo script file and the meshing program should output the mesh in this format. If Gmsh is used as the mesing engine, this all goes automatically. All boundary elements, i.e. horn wall surfaces, driving elements and interfaces are written in one single file, logically grouped ("tagged") into named groups.

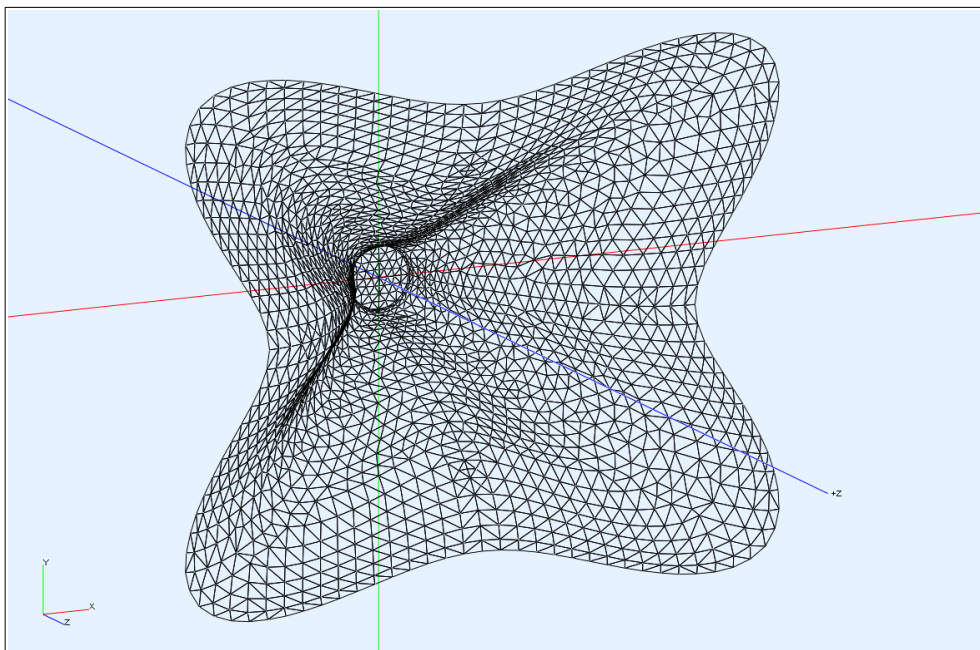


Fig 3: BEM mesh example (elements undifferentiated; ABEC 3)

3.1.1 Subdomains and interfaces

Terms subdomains and interfaces come from ABEC implementation and terminology: subdomains are basically (disjunctive) volumes of air; interfaces are the virtual boundaries separating them (and connecting them at the same time).

There are two types of subdomains: interior (closed) and exterior. By default Ath models the interior of a horn as one or more interior subdomains and the exterior as one exterior subdomain, solved separately in ABEC. This is also the only way to solve an infinite baffle radiation in ABEC, as everything analyzed must be in front of the baffle. By enclosing the interior of the horn (i.e. including the elements behind the baffle) by a subdomain interface boundary that lies in front of the baffle, this requirement is satisfied. For free standing horns this subdomain division is not strictly necessary as all elements can be defined in one exterior subdomain. Nevertheless, it can be used anyway.

For setting the number of subdomains and their interfaces use the items `Mesh.SubdomainSlices`, `Mesh.InterfaceOffset` and `Mesh.InterfaceDraw` – for more detail see also example in 6.7.

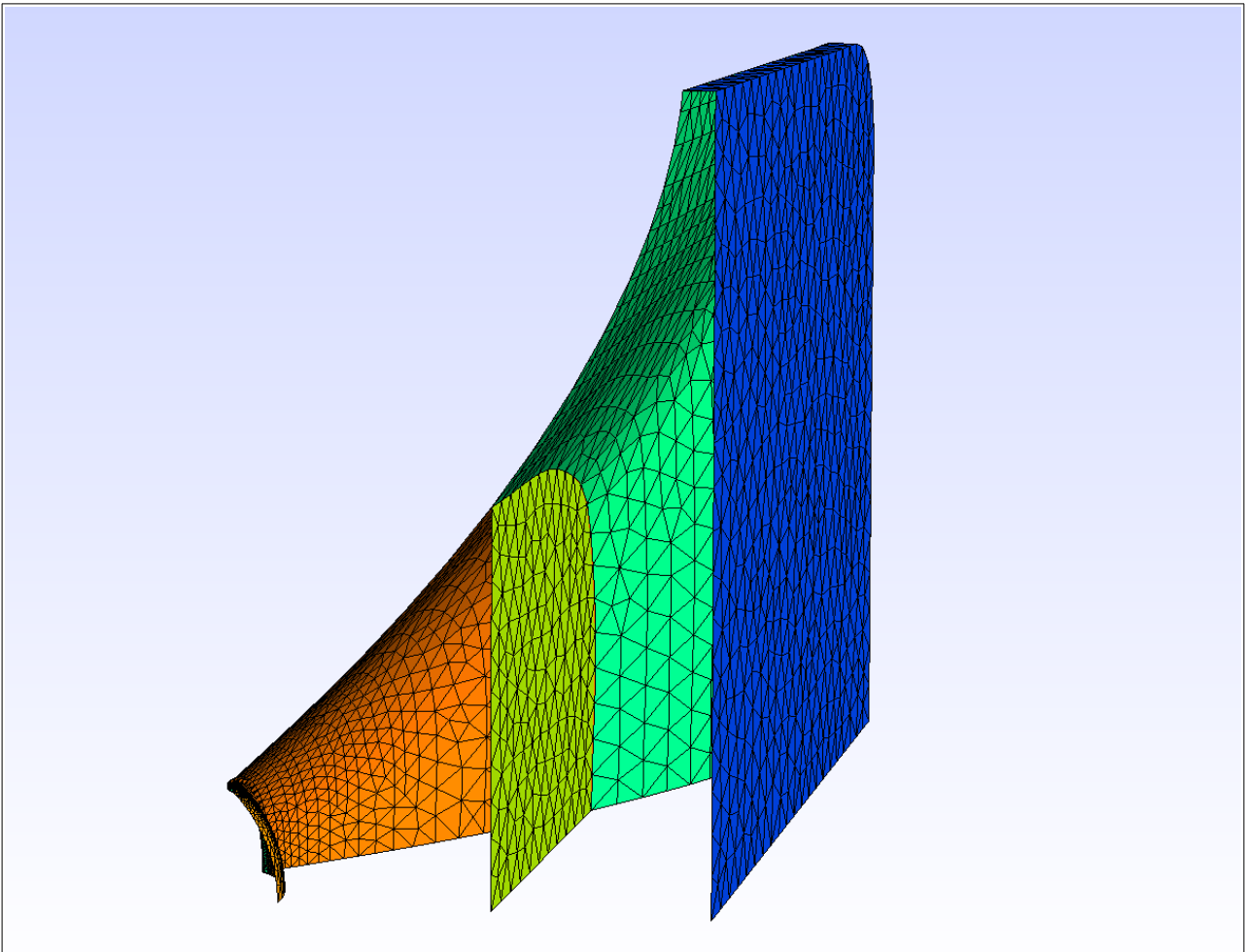


Fig 4: MSH file with source, two interior subdomains and their interfaces (opened in Gmsh)

3.2 Source definition

BEM analysis requires an acoustic source to be defined besides the horn itself - something that actually emits sound, i.e. drives the horn. These are called *driving elements* - the elements of the boundary mesh that are assigned a motion, its direction and intensity (either as a value of constant acceleration, velocity or excursion). Each element can be set to vibrate either in the direction of its normal vector (i.e. perpendicular to its face), or in any other defined direction. Elements forming a spherical surface can thus simulate either a pulsating source creating a spherical wavefront or an axially vibrating dome, depending on the assigned velocity direction.

There are two options for the source definition:

a) Define a simple spherical cap (or a flat piston, as a special case of thereof), vibrating either radially (pulsating source) or axially (moving back and forth like a piston). This is a common way of modeling ideal output of compression drivers. See the `Source.*` items.

b) Define arbitrary axi-symmetric geometry of the source, along with any optional non-moving parts, by a script file. This way a source can be defined in a great level of detail and complexity. In this cases use the `Source.Contours` item. See Appendix C for the syntax of the source definition script.

Notes:

- For a simulation of the source only, without a horn, set `Length=0` and `Mesh.LengthSegments=1`.
- Item `Source.Velocity` sets the direction of motion of the vibrating elements.

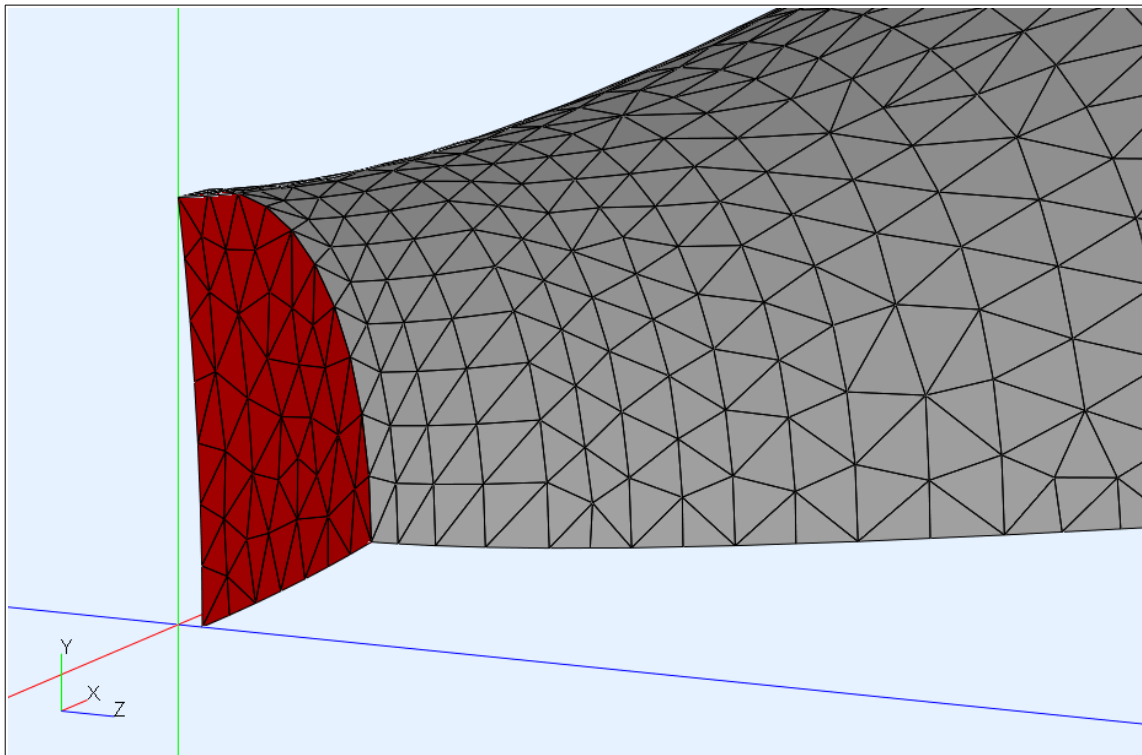


Fig 5: Ideal spherical wavefront source (ABEC display)

In the above picture a spherical wavefront source, matched to the throat opening angle (7°) is shown. If the opening angle was zero this would become a flat wavefront.

3.3 Geometry grid - terminology

Internally the whole geometry is defined at discrete points in a regular grid and subsequently processed as **profiles** and **slices**. These curves then define the whole surface. In the following triangular wireframe visualization an example of waveguide geometry with 8 slices and 32 profiles is shown.

Slices are numbered by integers starting at 0 at the throat. Profiles are also numbered from 0, the first being the one at 3 o'clock (when looking at the device from the front), increasing in anti-clockwise direction.

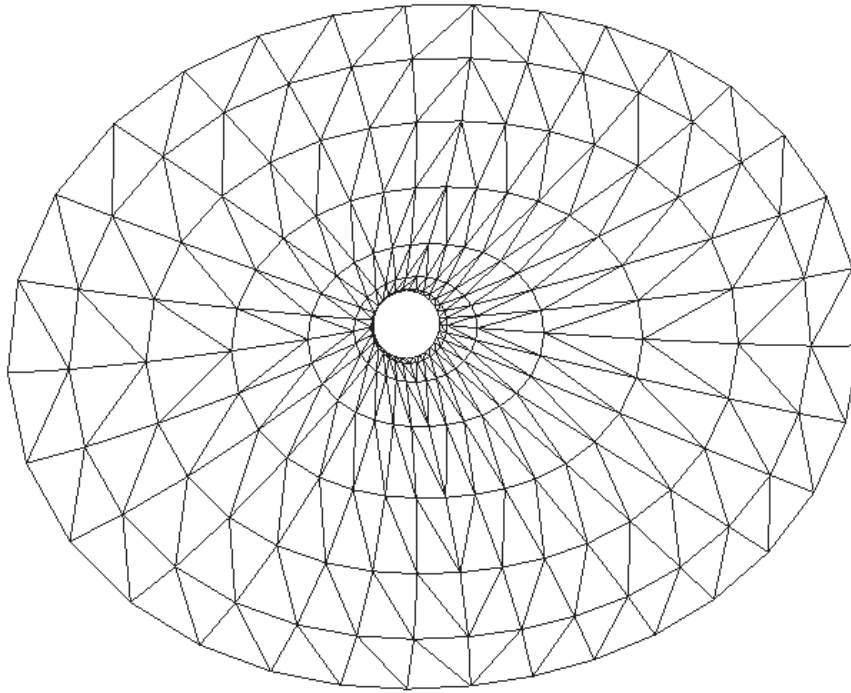


Fig 6: 32 profiles (radial curves) and 8 slices (closed ellipses)

3.3.1 Meshing the slices

There is an important distinction between the above described grid and the mesh. The main difference is that the densities of the two are independent to a large degree. Whereas the grid sets how densely the actual geometry is sampled, affecting the overall resolution of the surface, the density of the final mesh is set independently from the grid and can be higher.

This is how the meshing works:

- for each slice a smooth spline curve is created (i.e. controlled by the points of the grid),
- surface stripes between each adjacent pair of slices are created,
- each stripe is meshed independently, with a mesh resolution defined by the `Mesh.*Resolution` items.

3.4 Circular symmetry mode

For a waveguide exhibiting circular symmetry (i.e. being axisymmetric), the numerical analysis can be simplified considerably and the calculation time reduced dramatically. Instead of using a full 3D surface mesh, only a much simplified description of a single 2D profile curve is needed. For this purpose ABEC/AKABAK comes with a "CircSym" mode and Ath allows to generate such project.

To generate a BEM project in this mode, all that is necessary is to select a profile for the analysis (for the profile numbering see 3.3):

```
ABEC.SimProfile = 0
```

When the above configuration item is defined, the ABEC project that Ath generates will be modified for the "CircSym" analysis, employing the circular symmetry mode for the selected profile. Typically the definition file will also describe an axisymmetric device as well, but in principle this is not necessary or connected in any way - any profile of any device can be selected for the CircSym analysis.

In the circular symmetry mode Ath generates complete ABEC projects directly, without a need of calling an external mesh generator, i.e. for using Ath in this mode exclusively, you don't need a separate mesh generator at all (the item **MeshCmd** of the configuration file won't be used).

See section 6.10 for an example.

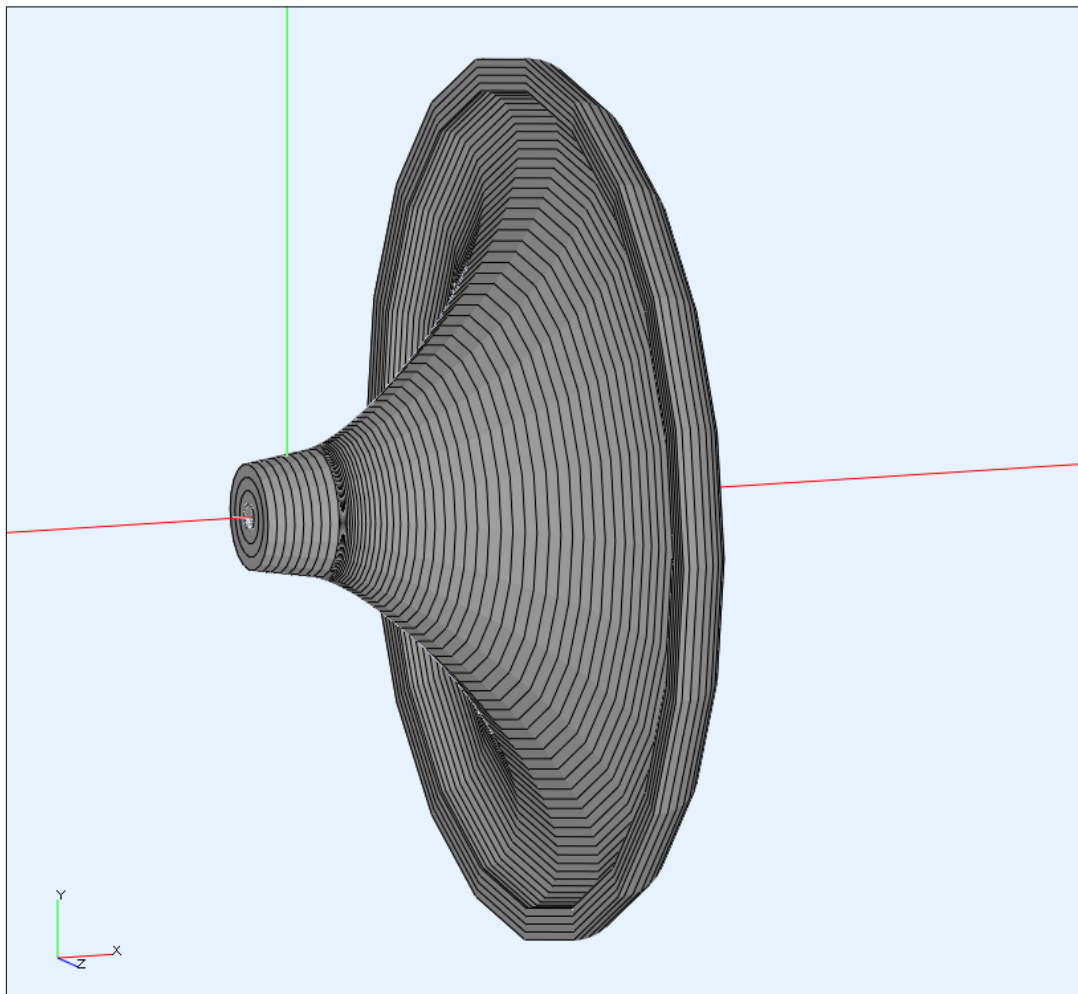


Fig 7: Simplified mesh of an axisymmetric free standing waveguide

4 Horn definition file

Ath takes its input in a form of a text file, called *definiton file*. It describes the complete geometry of one particular device and sets all the additional parameters controlling the behaviour of the tool and the output requested. It is an ordinary ASCII file to be created and edited with a plain text editor, such as Notepad or Notepad++⁸.

The structure of the file is simple and the only syntactical entity is an *item*. Items are specified in an arbitrary order in the following format: ***ItemKey[:tag] = Value***

ItemKey is always one of the predefined keywords. In some cases several items of the same *ItemKey* can be defined - in that case each item must be marked with a unique *tag*.

Value can be one of several recognized types:

Value Type	Description
f	Floating point (real) number. Always use dot (".") as the decimal separator.
f []	List of real numbers. Values are separated by commas (",").
i	Integer number (i.e. a whole number, without a decimal part).
i []	List of integer numbers. Values separated by commas (",").
b	Logical type (true/false); can be one of these constants: 1, yes, true, 0, no, false
ex	General mathematical expression. See Appendix B for a summary.
s	Text string (to the end of line or the nearest comment).
c	Script code. Can be either a path to an external script file, or the script code itself included in the definition file, enclosed in curly brackets ("...") - see 6.5.
{ }	Composed item - container for nested items, enclosed in curly brackets ("...").

Same basic rules:

- All *ItemKeys* are case sensitive (i.e. "Length" is different from "length").
- Comments starting with semicolon (";") can be used anywhere in the file - the rest of the line will be ignored. Empty lines are also ignored (there is one exception to this rule - see the Appendix B).
- The order of items in the definition file can be arbitrary.
- Some of the items are mandatory, i.e. without a default value, and must be always set. The non-mandatory items can be omitted – in that case their default values will be used.
- The program doesn't do any syntax checking - if a keyword is not recognized, it is ignored.
- If the same item is defined multiple times, the later definiton overwrites the previous one.

While this chapter gives a complete reference of the available functionalities, a more accessible and illustrative overview is given in Chapter 6. Probably the easiest way to get yourself familiar with the sript syntax is to go through the examples provided in the Ath delivery package.

Also, some of the parameters apply only when some other parameters are defined, so they work together as a group. Please see the following reference for the details.

⁸ Highly recommened, available as freeware at <https://notepad-plus-plus.org/>

4.1.1 Horn geometry

ItemKey (=default value)	Value	
Throat.Diameter	f	Throat diameter [mm] ($=2r_0$, see [1])
Throat.Profile = 1	i	Profile geometry: 1 = OS-SE (see also Term.*) 3 = circular arc (see CircArc.*)
Throat.Angle = 0	ex	Throat opening angle, where applicable. This value is half the included angle [deg].
Throat.Ext.Angle = 0	f	Angle of an optional conical throat extension. This value is half the included angle [deg].
Throat.Ext.Length = 0	f	Axial length of an optional conical throat extension [mm].
Slot.Length = 0	ex	Axial length of an initial straight waveguide segment [mm].
Length	ex	Nominal axial length (depth) of the device [mm], mandatory item.
Term.s = 0.7	ex	The parameter 's' of the OS-SE formula [1].
Term.q = 0.995	ex	The parameter 'q' of the OS-SE formula [1].
Term.n = 4.0	ex	The parameter 'n' of the OS-SE formula [1].
Coverage.Angle	ex	Explicit expression for the coverage angle [deg].
OS.k = 1	ex	The parameter 'k' of the OS-SE formula [1].
Rot = 0	ex	Final rotation of the computed profile around the point $[0, r_0]$ [deg].
GCurve.Type	i	1 = superellipse (GCurve.SE.* items used) 2 = superformula (GCurve.SF.* items used) Sets the type of a guiding curve used (for Geometry.Definition=2) Default: not defined (=using explicit shape definition)
GCurve.Dist	ex	Distance of the guiding curve from the throat (xy) plane: If set in $<0, 1>$, it is taken as the fractional part of the total horn length. If > 1 , it is taken as absolute distance [mm].
GCurve.Width	f	Sets the absolute size of a guiding curve (the size along the x axis) [mm].
GCurve.AspectRatio = 1	f	Sets the height / width ratio for the guiding curve. This is used universally - for a superellipse it sets the semi-axes ratio, for a superformula it allows to scale the whole shape in one direction.
GCurve.SE.n = 3	f	Exponent of a guiding superellipse (≥ 2)
GCurve.SF	f[]	Superformula definition - array of six numbers corresponding to the superformulla parameters: a, b, m, n1, n2, n3. In this definition $m = m1 = m2$. The absolute size of the curve is set with GCurve.Width and GCurve.AspectRatio .
GCurve.SF.a GCurve.SF.b GCurve.SF.m1 GCurve.SF.m2 GCurve.SF.n1 GCurve.SF.n2 GCurve.SF.n3	f	Alternate definition of a superformulla. In this definition the parameters are set individually by their names.
GCurve.Rot = 0	f	Rotation of the whole guiding curve in anti-clockwise direction [deg].
CircArc.TermAngle = 1	ex	Only for Throat.Profile=3 . Sets the mouth terminal angle [deg].
CircArc.Radius	ex	Explicit radius for Throat.Profile=3 . [mm]

4.1.2 Morph feature

The morph feature transforms the initial ("raw") shape into a new one with a defined mouth outline. For more details see [1].

Morph.TargetShape = 0	i	0 = keep original shape, do not morph 1 = morph to rectangle 2 = morph to circle
Morph.TargetWidth = 0	f	The requested final width of the mouth outline [mm]. 0 = keep raw width
Morph.TargetHeight = 0	f	The requested final height of the mouth outline [mm]. 0 = keep raw height
Morph.CornerRadius = 35	f	Corner radius for a rectangular outline [mm].
Morph.FixedPart = 0	ex	Portion of the horn length from the throat that will be kept fixed during the final morph transformation. Set as a number between 0 - 1. 0 = start at the throat, i.e. at z=0
Morph.Rate = 3	ex	The rate of the morph transformation. It controls how gradual or rapid the transition is, i.e. how quickly is the profile changed towards the mouth. Lower values give more rapid transition. Minimum is 1. For more details see [1].
Morph.AllowShrinkage = 0	b	Sets whether shrinkage of the original shape is allowed as the result of the feature. If set to 0 the target size will be automatically enlarged if needed to ensure that no shrinkage happens anywhere.

4.1.3 Mouth rollback

Rollback = 0	b	Activates mouth rollback feature. Vital for free standing horns.
Rollback.StartAt = 0.5	ex	Fraction of the length of the profile where the rollback starts. Note: The "length" here is not the axial length but the actual length along the profile curve.
Rollback.Angle = 180	ex	Desired terminating wall angle, relative to the horn axis. [deg] (Value of 180 means that the curve makes a "half circle" in total.)

4.1.4 Mesh generator

Basic Geometry

ItemKey (=default value)	Value	
Mesh.Quadrants = 1	i	Sets the portion of the 3D mesh to be used for BEM analysis: 1 = quadrant 1 only ($x \geq 0$ and $y \geq 0$) 12 = quadrant 1 + quadrant 2 ($y \geq 0$) 14 = quadrant 1 + quadrant 4 ($x \geq 0$) 1234 = full mesh
Mesh.AngularSegments	i	The grid density: total number of calculated profiles around the waveguide. Suitable value generally depends on the complexity of the shape. It must be multiple of 4 (this is checked and adjusted automatically). Typical value: 100
Mesh.LengthSegments	i	The grid density: total number of slices along the length. Distances between individual slices are set by Mesh.ZMapPoints .
Mesh.CornerSegments	i	Number of profiles reserved for the corner of the rounded rectagle if Morph.TargetShape=1 .
Mesh.ThroatSegments	i	Number of slices reserved for the throat extension, if any (see Throat.Ext.*).
Mesh.ThroatResolution = 5	f	Nominal BEM mesh resolution at $z = 0$ [mm].
Mesh.MouthResolution = 8	f	Nominal BEM mesh resolution at $z = \mathbf{Length}$ [mm]. Sizes of mesh elements for z between 0 and Length are smoothly interpolated between *.ThroatResolution and *.MouthResolution. (The old item Mesh.InterfaceResolution is now obsolete and will be removed in the future.)
Mesh.SubdomainSlices	i[]	Indices of grid slices for subdomain interfaces. Default: not defined, i.e. Mesh.LengthSegments-1 (the last slice) Use Mesh.SubdomainSlices= (i.e. empty list) to disable the feature, i.e. to use only one single exterior subdomain.
Mesh.InterfaceOffset	f[]	Forward protrusions of the interfaces [mm]. This array should have the same length as Mesh.SubdomainSlices to which it corresponds. Default: not defined (all zero)
Mesh.InterfaceDraw	f[]	Forward-draw depths of the interfaces [mm]. This array should have the same length as Mesh.SubdomainSlices to which it corresponds. Default: not defined (all zero)
Mesh.RearShape = 1	i	1 = full model (realistic) 2 = flat disc (mathematical surface) Shape of the rear side of the free standing horn model.
Mesh.WallThickness = 5	f	Wall thickness for a free standing horn [mm].
Mesh.RearResolution = 10	f	Rear wall mesh resolution for a free standing horn [mm].

Enclosure definiton (optional)

ItemKey	Value
<pre> ; Stock enclosure: Mesh.Enclosure = { Spacing = Depth = EdgeRadius = EdgeType = 1 2 FrontResolution = BackResolution = } ; User-defined plan my_plan = { ... ; see Appendix C } Mesh.Enclosure { Plan = my_plan Spacing = FrontResolution = BackResolution = } </pre>	<pre> {} </pre> <p>If defined, specifies an enclosure to be used. Either a pre-defined shape or a user-defined ground plan can be used. This is given by the presence or absence of the subitem Plan (if not set, a pre-defined shape is used).</p> <p>Spacing = <left, top, right, bottom> For a pre-defined shape, specifies the distances between the waveguide outline and the front baffle outer edges [mm].</p> <p>Depth = <depth> Depth of a pre-defined enclosure [mm].</p> <p>EdgeRadius = <radius> Radius (size) of the edge treatment [mm].</p> <p>EdgeType = 1 2 1 = rounded (default) 2 = chamfered</p> <p>Plan = <plan_item> Specifies a user-defined ground plan, where <plan_item> is the name of the plan item. For the complete syntax of the plan definition see Appendix C.</p> <p>FrontResolution = <f1,f2,f3,f4> Front baffle mesh resolution for corners in quadrants 1 - 4, [mm].</p> <p>BackResolution = <b1,b2,b3,b4> Back baffle mesh resolution for corners in quadrants 1 - 4, [mm].</p>

4.1.5 ABEC/BEM project settings

General Simulation Parameters

ItemKey (=default value)	Value	
ABEC.SimType = 1	i	Boundary conditions: 1 = infinite baffle 2 = free standing horn
ABEC.SimProfile = -1	i	Selects a profile for the Circular Symmetry mode and activates this mode. -1 = not used
ABEC.f1	f	Low frequency limit [Hz]
ABEC.f2	f	High frequency limit [Hz]
ABEC.NumFrequencies	i	Number of frequency points for the BEM analysis.
ABEC.Abscissa = 1	i	Spacing of frequency points: 1 = logarithmic 2 = linear
ABEC.MeshFrequency = 1000	f	The <i>mesh frequency</i> as documented in ABEC manual. The best practice is to make the mesh resolution high enough by setting the Mesh.*Resolution so that ABEC won't need (and won't do) further re-meshing. If this value is increased, ABEC may perform some additional meshing by subdividing the existing one. Note: For the Circular Symmetry mode this value should be increased considerably, at least to 30000 Hz.

Acoustic Source definition

Source.Shape = 1	i	1 = spherical cap 2 = flat disc Sets the driving wavefront shape at the throat. Note that the shape itself doesn't define a velocity direction of the vibrating elements - see also Source.Velocity .
Source.Radius = -1	f	Radius of the driving spherical cap [mm]. If set to -1, the radius will be calculated automatically to match the (average) throat opening angle.
Source.Curv = 0	i	Curvature of a spherical cap (Source.Shape = 1): 1 = convex -1 = concave 0 = automatic (matching the throat angle)
Source.Velocity = 1	i	Velocity direction of the driving elements: 1 = normal to element surface 2 = axial (piston motion in z-axis)
Source.Contours	c	Source definition script. See 6.5. When present, it overrides all the other Source.* parameters except Source.Velocity .

Transducer LE Model (optional)

LE = <LE_script_file>	s	Includes a lumped-element model. The models are stored as separate files in a subdirectory "bin\lib\drivers". For more information about LE models see 6.13.
LE.System	s	The internal tag of the 'System' section of the LE script. This is "S1" by default. [2]
LE.Driver	s	The internal tag of the 'Driver' section of the LE script. This is "D1" by default. [2]
LE.Voltage	f	Input voltage (Vrms) for the LE analysis. The default value is 2.83.

Observations

ItemKey	Value
<pre>ABEC.Polars:<polar_tag> = { MapAngleRange = NormAngle = Distance = Offset = Inclination = Curves = } ; Examples: ; Default for free standing ABEC.Polars:SPL = { MapAngleRange = 0,180,37 } ; Infinite baffle ABEC.Polars:SPL = { MapAngleRange = 0,90,19 Offset = 125 }</pre>	<p>{} Polar map data calculation:</p> <p>MapAngleRange = <a0, a1, N> The angle range a0 .. a1 [deg] and the number of angular points N. Angular resolution = (a1 - a0) / (N - 1).</p> <p>NormAngle = <angle> Normalizes the data to the specified angle [deg].</p> <p>Distance = <mic_distance> Distance of the observation points from the origin [m]. If not specified, farfield calculation with level and phase adjusted for 1 m distance is used.</p> <p>Offset = <center_offset> Moves the center of rotation along the z axis [mm]. This is important especially for an infinite baffle, where it must be in front of the baffle. By default the center of rotation is at z=0.</p> <p>Inclination = <angle> Rotates the plane of the polar observation arc [deg]. In a full-space analysis 0 = horizontal, 90 = vertical plane.</p> <p>Curves = <angle_list> Calculation of individual frequency responses at selected angles, in addition to the polar map defined with MapAngleRange. "angle_list" is a comma-separated list of angles [deg].</p>

4.1.6 Program output

These items control what files the program generates and how.

ItemKey (=default value)	Value	
Output.SubDir Example: Output.SubDir = "demos"	s	Optional - a subdirectory of the OutputRootDir for the output of the program. This way it's possible to organize the projects. If not defined the output directory will be created directly under the OutputRootDir path.
Output.DestDir	s	Optional - the target directory for the current project. This item overwrites the setting of OutputRootDir in the global configuration file.
Output.STL = 1	b	Generates STL file (instructs mesh generator to do so).
Output.MSH = 0	b	Generates MSH file (instructs mesh generator to do so). Note that this is not the mesh for BEM analysis and contains only the (whole) waveguide surface.
Output.ABECProject = 0	b	Generates ABEC project.

Grid coordinates export

ItemKey	Value	
GridExport:<tag> = { ProfileRange = <from>,<to> SliceRange = <from>,<to> ExportProfiles = 0 1 ExportSlices = 0 1 Scale = 1.0 Delimiter = ";" FileExtension = "csv" SeparateFiles = 0 1 }	{ }	Exports grid points as raw X,Y,Z coordinates to ASCII file. Creates two types of files/curves: profiles and slices (depending on the setting of ExportProfiles and ExportSlices), numbered as explained in 3.3. These files can be directly imported by the included Fusion 360 Add-Ins as spline curves.

Reporting

ItemKey	Value
<pre>Report = { Title = <title> PolarData = <polar_tag> NormAngle = <deg> MaxAngle = <deg> SPL_Range = <dB SPL> DrvImp_range = <Ohm> ExcRefSPL = <dB SPL> MaxRadius = <mm> Width = <pixels> Height = <pixels> GnuplotCode = <gpl_file> } ; Example: Report = { Title = "Demo" NormAngle = 10 Width = 1024 Height = 768 }</pre>	<p>{} For more information about using reports, see 6.14.</p> <p>The report will use SPL data from the polar map specified by its tag (see ABEC.Polars). The default tag is "SPL".</p> <p>NormAngle specifies the normalizing angle, if any [deg].</p> <p>MaxAngle sets the chart angle range [deg]. Default = 90.</p> <p>SPL_Range sets the range for absolute SPL plots [dB]. Maximum value is calculated automatically to fit the data and rounded to the nearest multiple of 5 dB.</p> <p>DrvImp_range sets the range of the plot with electrical impedance of the driver, if applicable [Ohm].</p> <p>ExcRefSPL sets the value of SPL for the reference excursion calculation [dB], if applicable.</p> <p>MaxRadius sets the horn radius range for a profile sketch [mm], where applicable.</p> <p>Width and Height set the size of the image in pixels.</p> <p>GnuplotCode allows user customized reports - Gnuplot is executed so that it uses this source code. Source code files are stored in the subdirectory "bin\lib\scripts". By default the source code is determined automatically, depending on whether a driver LE model is used.</p>

5 Exporting to CAD

Among the output files of the program can be a STL file of the horn surface (`Output.STL=1`). Although useful in many situations, like e.g. a quick visual inspection in a favourite STL viewer, there is a better way to transfer the data for a subsequent CAD processing. The geometry should be imported as a smooth surface suitable for a subsequent high-quality 3D modeling.

Customized import scripts for Autodesk Fusion 360 are delivered in the Ath package.

5.1 Export

To export the coordinates use the item `GridExport` (keep the values default for a subsequent Fusion 360 import to work properly).

Two types of files are created:

- slices file(s) (*_slices.csv)
- profiles file(s) (*_profiles.csv).

5.2 Import into Autodesk Fusion 360

Two import scripts can be found in the Ath package:

- **CurvesImport**
Imports individual splines from either a slices or a profiles file.
- **SurfaceImport**
Creates smooth (lofted) horn surface from the slices file.

In order to use these scripts they must be first imported as user Scripts into Fusion 360.

6 Tutorial

This chapter gives a guided tour through the functionalities of the program. We start by designing a simple OS-SE waveguide and analyzing it with BEM. Then we show how to use some more elaborate features of the tool. All the actual scripts used in this chapter are available in the demo folder of the distribution package.

6.1 The basics

Let's say we want to design and analyze an OS-SE waveguide of the following parameters:

Mouth diameter:	10" (254 mm)
Nominal coverage angle:	90°
Throat specification:	1" (25.4 mm) diameter; 14° opening angle

We start by creating a definition file (see full reference in Chapter 4). First, the obvious and mandatory:

```
Throat.Profile = 1      ; 1 = OS-SE waveguide
Throat.Diameter = 25.4  ; [mm]
Throat.Angle = 7       ; half the included angle [deg]
Coverage.Angle = 45    ; half the included angle [deg]
```

We would like the mouth diameter to be set to 254 mm explicitly but unfortunately that's not how the tool works, at least not in the current implementation. The basic parameter instead is always the *length* of the device - based on the length and the profile(s) defined, the outer dimensions just result from that (also note the the length doesn't have to be the same in each profile). It would be much more difficult doing it the other way around. For now we just estimate how deep the waveguide needs to be to have the desired outer diameter and eventually adjust something later (not necessarily the length if that's what we want to be fixed for some reason):

```
Length = 100           ; [mm]
```

Now we have the basic part of the profile defined. What we do next is setting the mouth terminating flare of the OS-SE formula (see [1]). Let's try some (already proven, in fact) starting values:

```
Term.s = 0.5
Term.n = 4.0
Term.q = 0.996
```

At this point we have defined the complete geometry of the waveguide itself. There are still parameters that must be set properly, however.

At the moment we want the raw shape without any additional morphing:

```
Morph.TargetShape = 0      ; 0 = no morphing (the default)
```

We also need to setup the mesh and its resolution (this affects the BEM performance immensely):

```
Mesh.AngularSegments = 64
Mesh.LengthSegments = 20
Mesh.ThroatResolution = 4.0 ; [mm]
Mesh.InterfaceResolution = 8.0 ; [mm]
Mesh.InterfaceOffset = 5.0 ; [mm]
```

As the last step we setup the required output. At this point we want only a STL file so we can visually inspect the resulting shape (we don't setup an ABEC/BEM project yet):

```
Output.STL = 1  
Output.ABECProject = 0
```

That's all! We have written the definition file. Let's save this file as "demo1.cfg" and continue with the next step - running the Ath program.

Note: You may have noticed that we omitted all the `Source.*` items. That's because we are satisfied with the default values at the moment. This means that an ideal (pulsating) spherical wavefront matching the throat opening angle will be used, which is just fine - we have nothing better at the moment anyway.

6.2 Running the program

Run the command line window (*cmd.exe*) and make sure you stand in the directory where you put the Ath program - in this example it is the directory `D:\projects\ath` (how to do this see 1.1.4). Try to dry run the tool just by typing `ath` and press Enter - the program should print the following text on the screen and exit (so you can continue using the command line), which means it is working and ready⁹:

```
D:\projects\ath>ath
Ath 4.8.0
-----
Freeware version for personal non-commercial use
-----
Usage: ath <horn_definition_file>
```

Note: In the command line window you can return back to the previously typed command by pressing the "up" arrow key and move through the recent command history.

To process a prepared definition file we need to pass this file as a command line paramater to the Ath program. This is done by typing the (either relative or absolute) path to the definition file after the `ath` command. In our example the definition file `demo1.cfg` is located in a subdirectory called "demos" (if you are not sure, you can't do wrong by typing the full path, e.g. "`D:\projects\ath\demos\demo1.cfg`").

```
D:\projects\ath>ath demos\demo1.cfg
Ath 4.8.0
-destination directory: D:\Horns\Demos\demo1
-initializing
-fixed length: 100 mm
-calculating profiles
-final mesh average throat angle: 7.000 deg
-running 'D:\gmsk-4.6.0-windows64\gmsk.exe mesh.geo -'
[...]
Done.

Final width x height = 269.4 x 269.4 mm (10.607 x 10.607")
Final length =      100.0 mm (3.937")
```

What happened is that the program succesfully processed the input script and generated the requested output files in the destination directory. Note that a subdirectory *demo1* was automatically created under the target directory, i.e. each project gets its own subdirectory with the same name as the script to store the data.

The last two lines of the output of the program show the final dimensions of the device as calculated by the program:

```
Final width x height = 269.4 x 269.4 mm (10.607 x 10.607")
Final length =      100.0 mm (3.937")
```

⁹ In these examples Ath is configured to be using Gmsg as the external meshing engine (see 1.1.3).

Note the reported outer diameter of 269.4 mm. Because we wanted the outer diameter to be 254 mm (10"), let's go back to the script file and adjust the shape to better match our goal. There are many parameters or their combinations that could do it. In our example we decided (after a few tries) simply to adjust the length from the initial 100 mm to 94 mm. This gives us the overall diameter of 253.6 mm, which is just fine:

```
D:\projects\ath>bin\ath cfg\demo1.cfg
Ath 4.8.0
-destination directory: D:\Horns\Demos\demo1
-initializing
-fixed length: 94 mm
-calculating profiles
-final mesh average throat angle: 7.000 deg
-matched wavefront radius: 104.21 mm
-running 'D:\gms4-4.6.0-windows64\gms4.exe mesh.geo -'
[...]
Done.

Final width x height = 253.6 x 253.6 mm (9.986 x 9.986")
Final length = 94.0 mm (3.701")
```

Now we are ready to run the BEM analysis. Before doing so we can have a quick look at our waveguide by opening the generated STL file¹⁰:

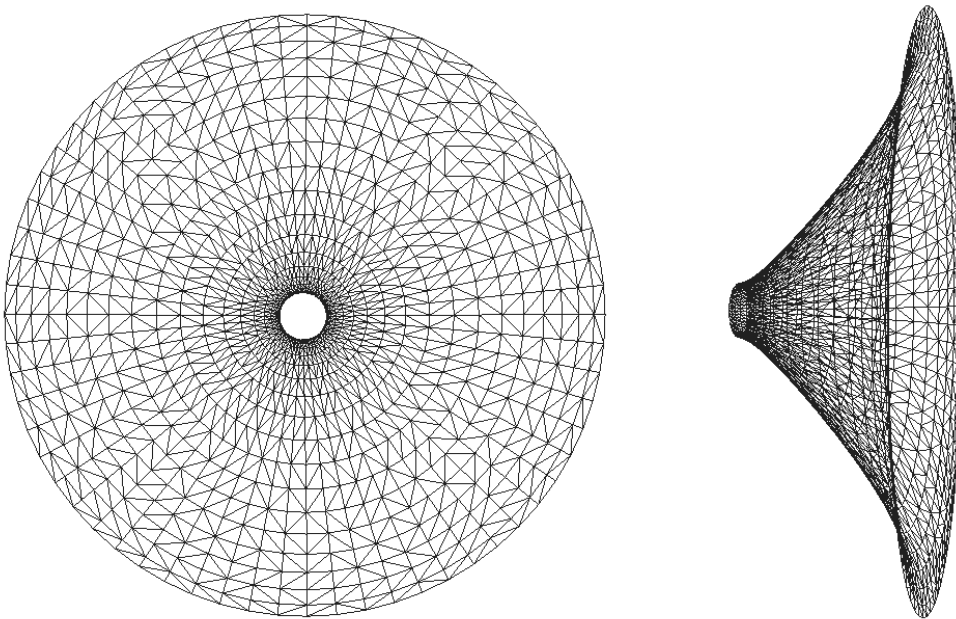


Fig 8: STL file preview

The geometry grid, i.e. all the profiles and slices, are clearly visible in this wireframe visualization.

¹⁰ For example using the online viewer at <https://www.viewstl.com>

6.3 Running BEM analysis

Ath can generate a complete ABEC project for the BEM analysis. First of all, we must activate this in our script:

```
Output.ABECProject = 1
```

Now we continue by setting the parameters of the analysis (we have already set the basic mesh resolution in the previous section). Most of the time we won't change these values very often. Some of the parameters can also be changed subsequently in the generated ABEC scripts manually, without a need to re-run Ath.

Let's place our waveguide in an infinite baffle and setup the frequency range for analysis:

```
ABEC.SimType = 1                ; 1 = infinite baffle

ABEC.f1 = 1000                  ; [Hz]
ABEC.f2 = 10000                 ; [Hz]
ABEC.NumFrequencies = 20
ABEC.MeshFrequency = 1000       ; [Hz]
```

We also need to set the "observations", i.e. all the data we want ABEC to calculate and present, like polar maps, etc. So let's setup a polar map, tagged simply as "SPL", by the following definition:

```
ABEC.Polars:SPL = {
  MapAngleRange = 0,90,19      ; first angle, last angle, number of points
  NormAngle = 20                ; normalization angle [deg]
  Distance = 3                  ; [m]
  Offset = 95                   ; [mm]
}
```

Note the sub-item "Offset = 95" - that's for taking into account the length of the waveguide, which is 94 mm (see the previous section). If we omitted this item (Offset = 0) some of the larger off-axis points would end up being behind the infinite baffle and the corresponding polars wouldn't get calculated. When setting polar maps, always remember to set the offset properly, especially for infinite baffles.

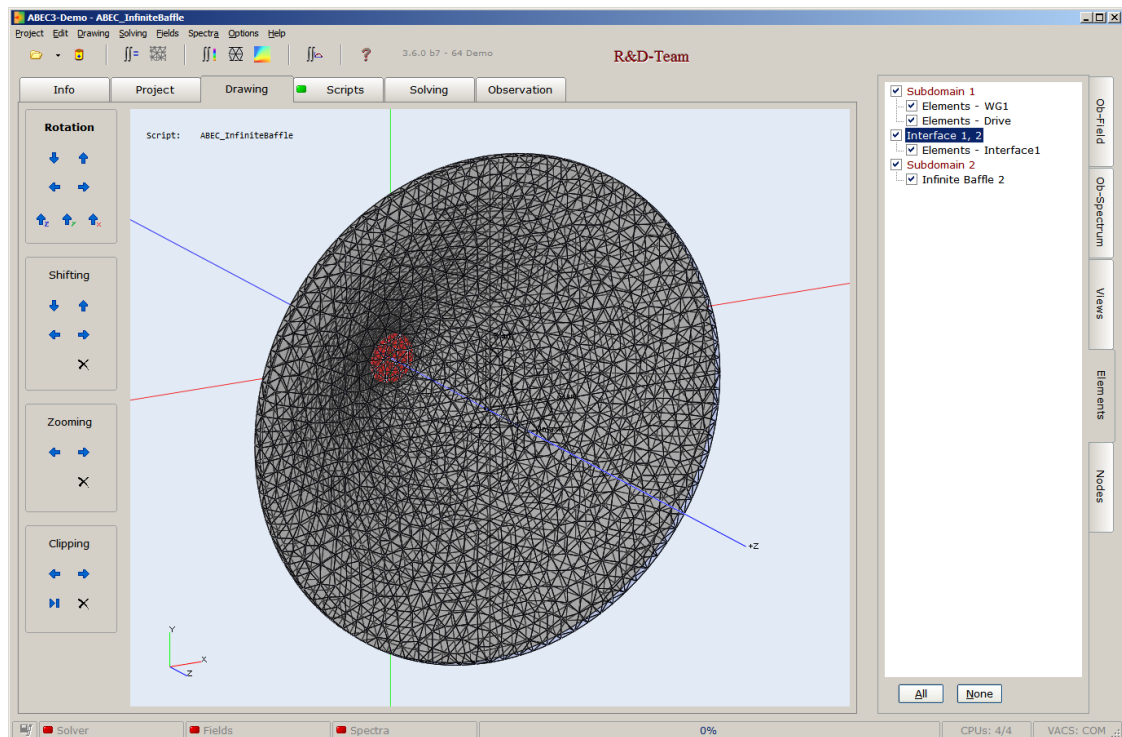
Note: For an axisymmetric shape it doesn't have any sense but in general we may want to observe the polars also in a different plane than the horizontal (i.e. the default one). In that case we would simply define another **ABEC.Polars** item(s) with a different tag and set the **Inclination** parameter accordingly:

```
ABEC.Polars:SPL_D = {          ; diagonal polars
  MapAngleRange = 0,90,19      ; first angle, last angle, number of points
  NormAngle = 20                ; normalization angle [deg]
  Distance = 3                  ; [m]
  Offset = 95                   ; [mm]
  Inclination = 45              ; [deg]
}
```

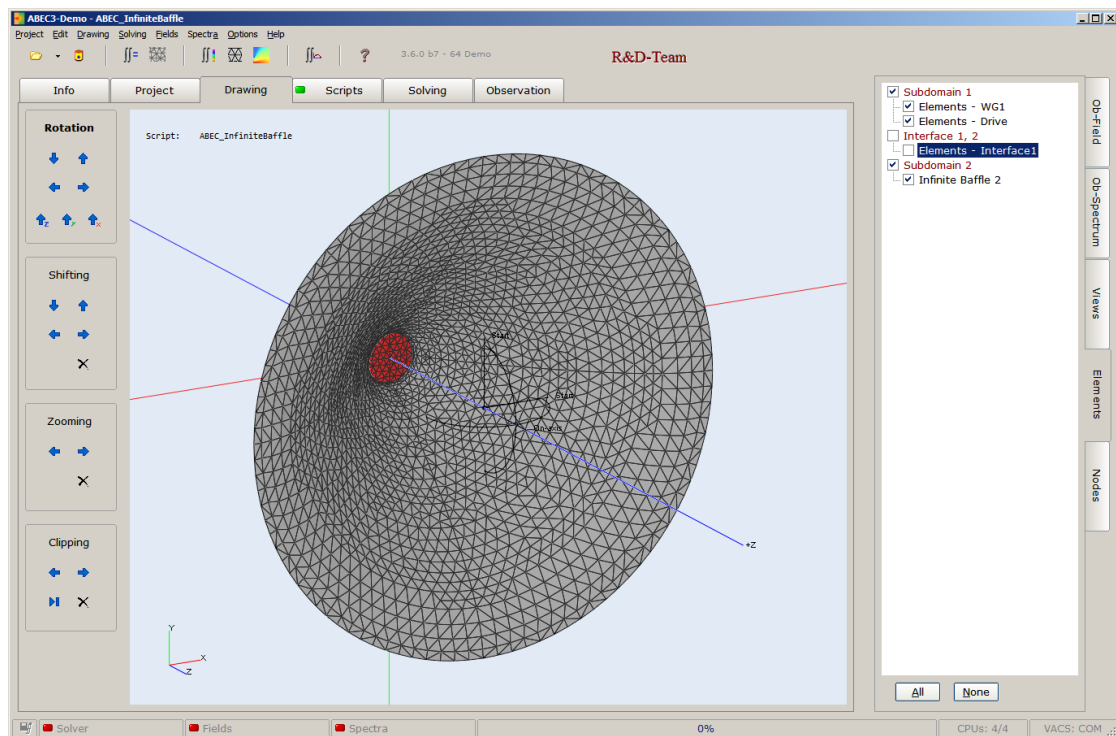
Now we have the script finished and there's nothing easier than to re-run the program, open the generated project in ABEC and run the numerical analysis.

Start ABEC and open the generated ***.abec** file, or simply double-click this file in Windows Explorer (this works only if the extension *.abec is correctly associated to the ABEC application).

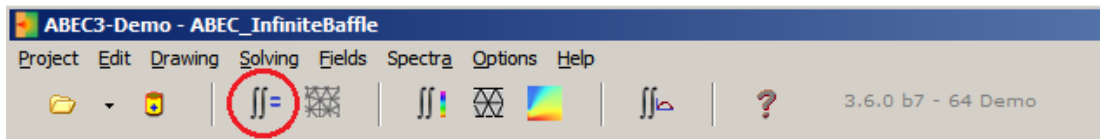
After opening the project file, ABEC shows the loaded mesh in the the "Drawing" tab:



In the "Elements" tab on the right you can see all the logical parts the mesh is composed of. By default all elements are displayed. To see the device better we can switch-off the interface mesh, otherwise covering the mouth:

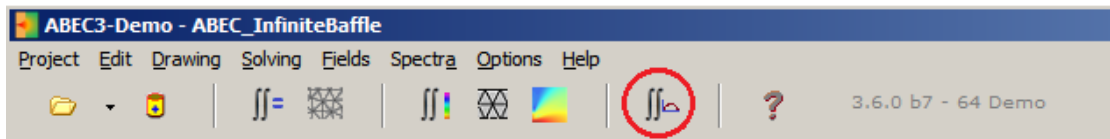


To run the calculation, press F5 or select "Solving - Start Solving" from the menu, or just click the marked button:



ABEC starts solving and a progress indicator is shown on the window status bar. The project **demo1** is quite small and it takes no more than about 5 minutes on an average contemporary home PC.

The next step is to calculate the spectra. Do this by pressing F7 or selecting "Spectra - Start calculation" from the menu or clicking the marked button:



Again, ABEC starts the calculations but this time it is generally much faster than the previous step.

After the calculation, by default VACS viewer is started automatically and the data are transferred from ABEC to VACS for a presentation:

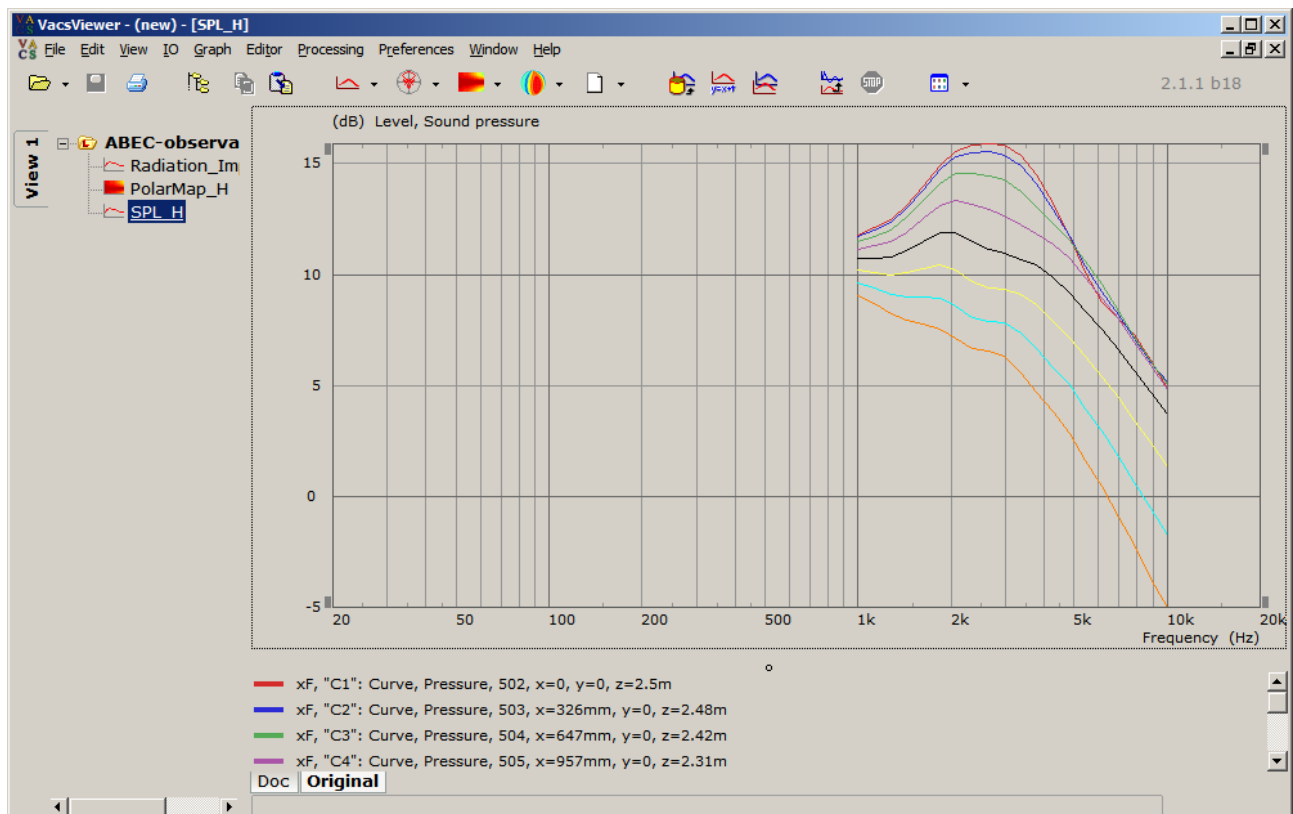


Fig 9: VACS viewer

For this project we calculated the horizontal polars (the waveguide is axisymmetric) - both as a Bode plot and in a form of a polar map, and the throat radiation impedance:

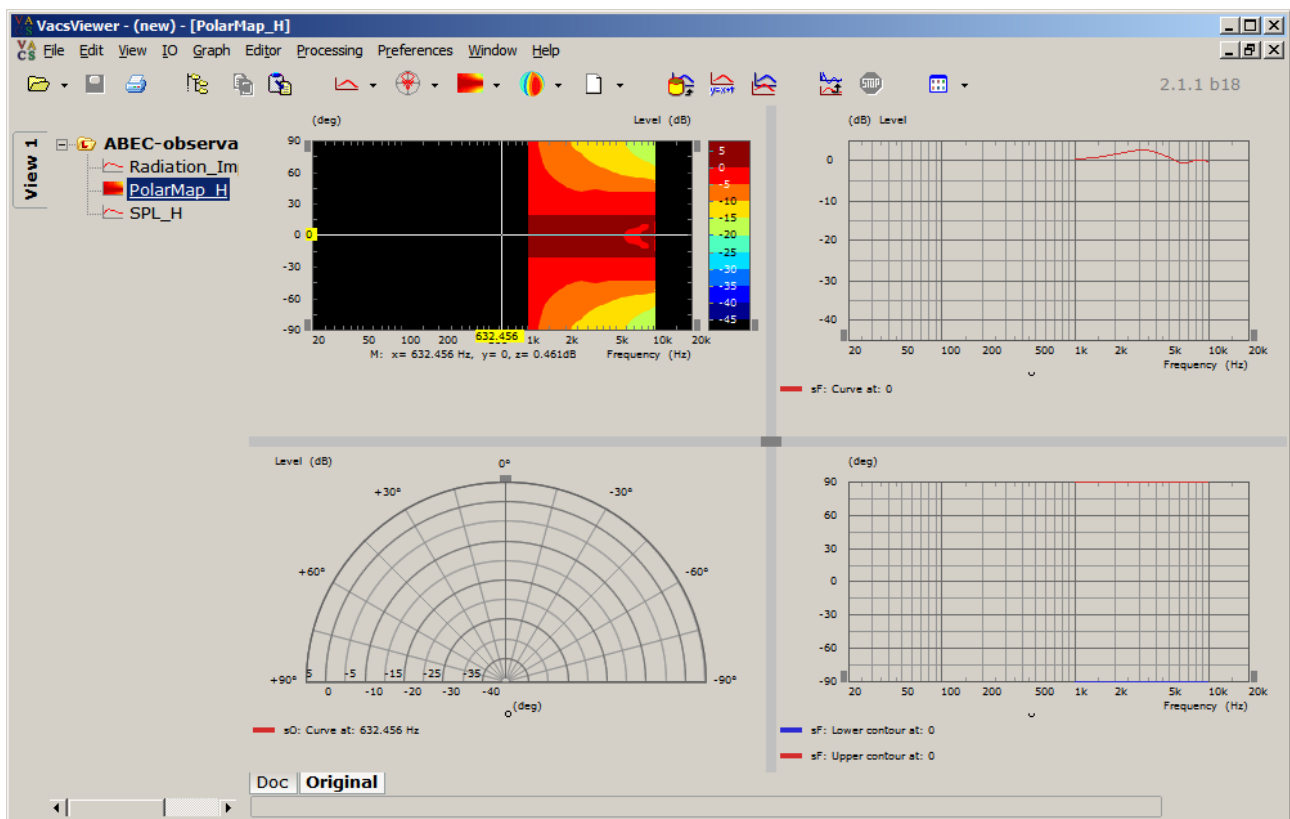


Fig 10: VACS - polar map

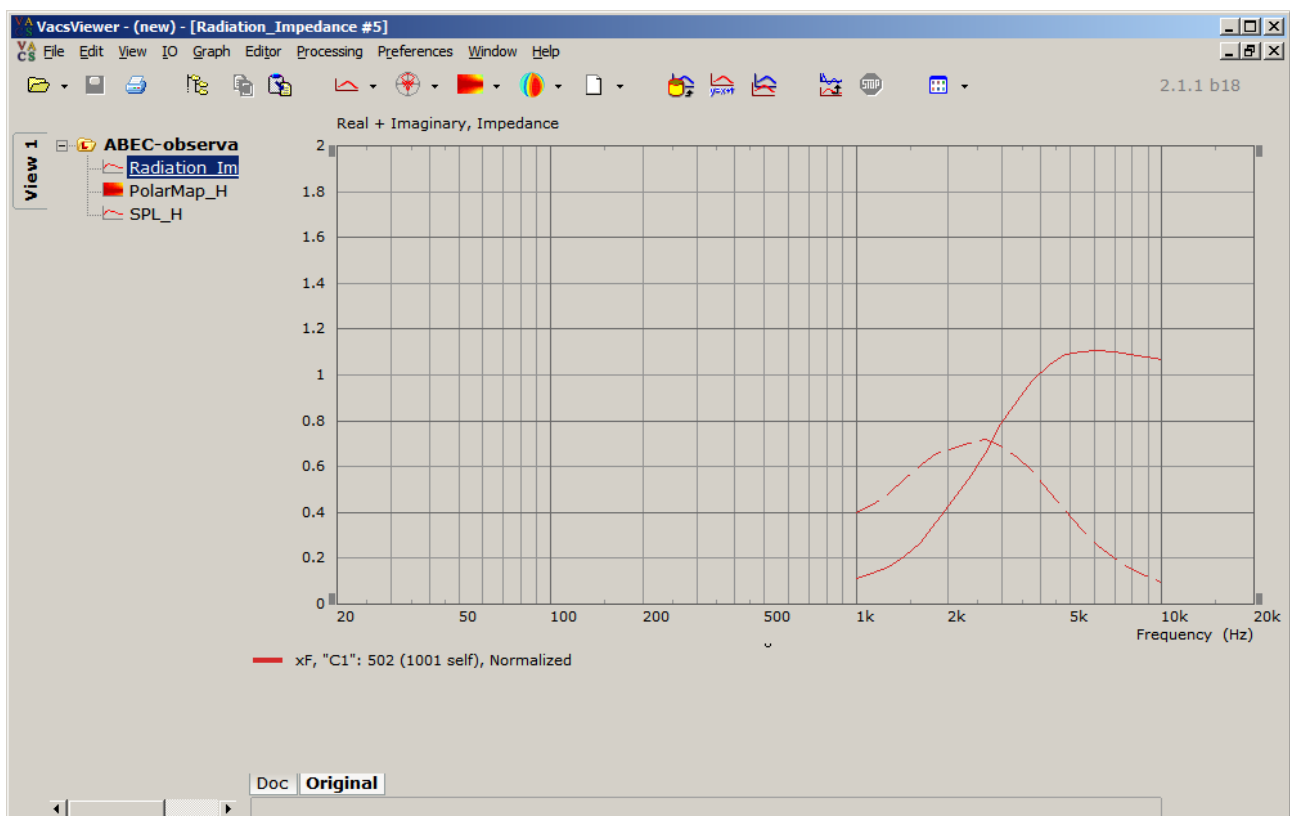


Fig 11: VACS - radiation impedance

6.4 Refining the profile

So far we defined a simple axisymmetric waveguide. We can do much more than that. As the next step let's add some non-symmetry. First let's continue with our explicit formula approach, then we will show how the concept of a guiding curve works.

6.4.1 Explicit definition

In our example we specified the coverage angle merely as a constant:

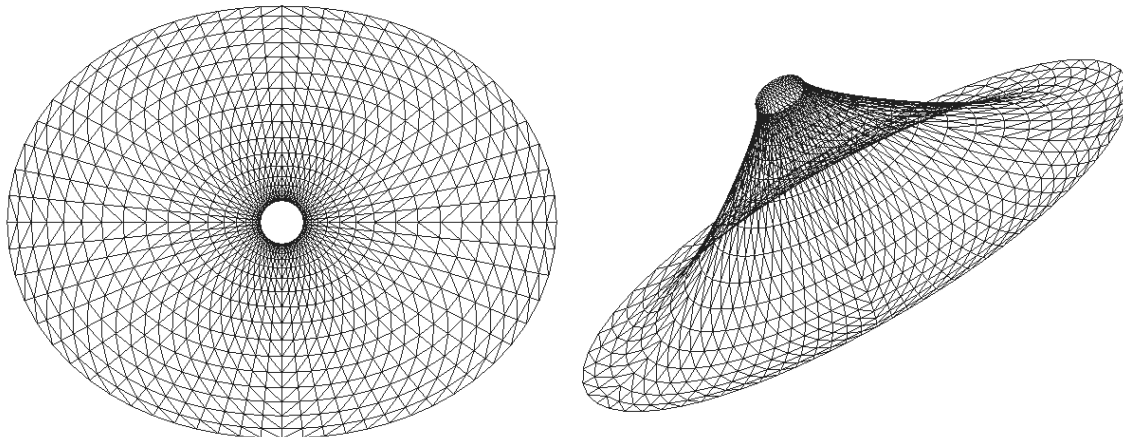
```
Coverage.Angle = 45 ; demo1.cfg
```

What we can do instead is to define the coverage as a function of the angle around the horn:

```
Coverage.Angle = 45 - 10*sin(p)^2 ; demo2.cfg
```

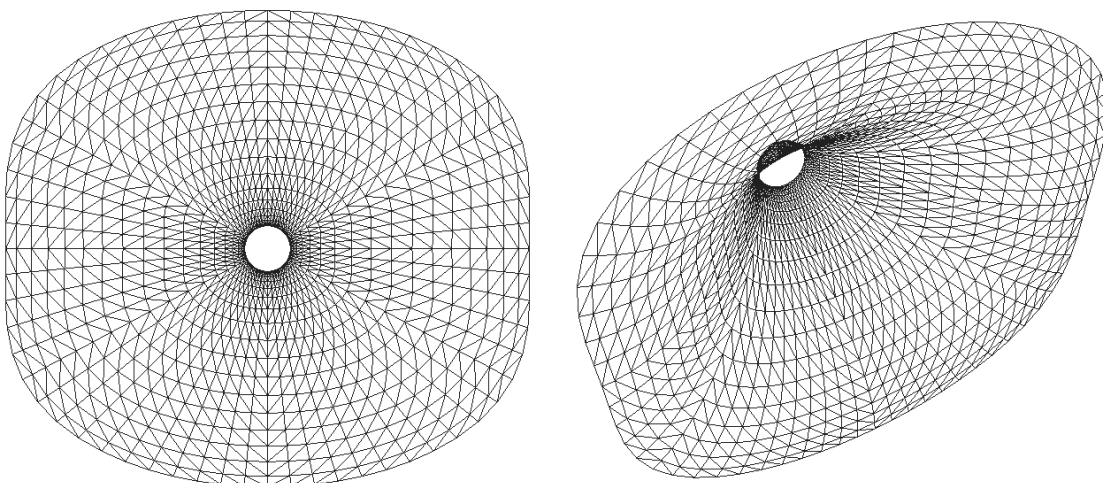
Note that the (implicitly defined) parameter 'p' (for *phi*) means just the angle around the horn.

What happened? We made the mouth elliptical by reducing the vertical coverage angle:



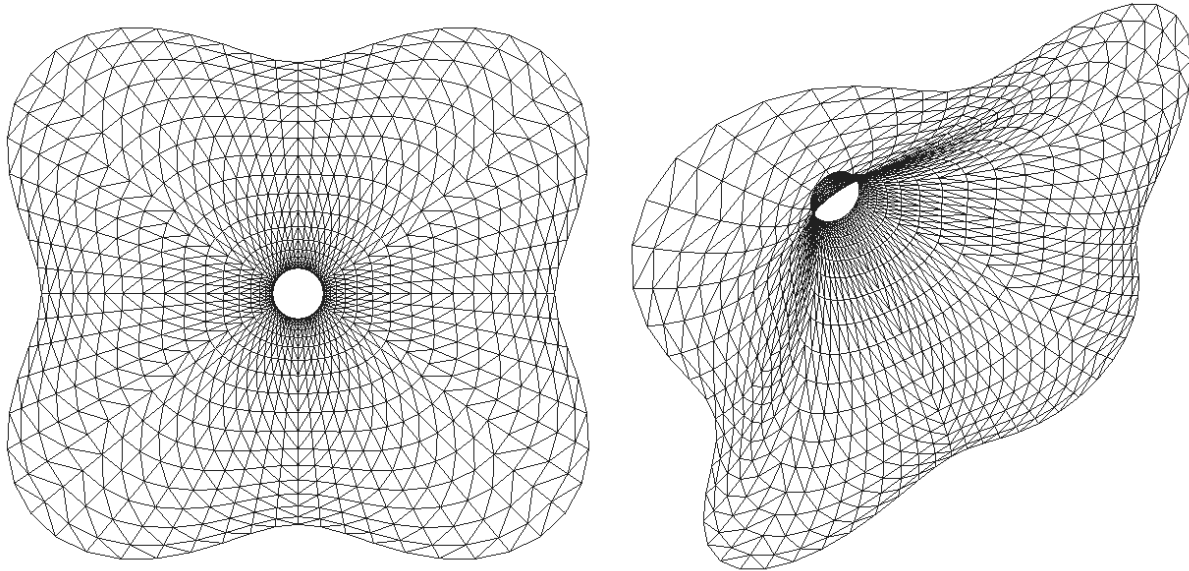
Let's do something more and adjust the coverage angle again:

```
Coverage.Angle = 45 - 10*sin(p)^2 - 6*cos(p)^12 ; demo3.cfg
```



In fact we don't have to limit ourselves only to the coverage angle - actually any definition of an item of the type "ex" (as defined in Chapter 4) can be set this way. In the following example we make e.g. the mouth flare variable, larger on the diagonals:

```
Term.s = 0.3 + 0.5*sin(2*p)^4 ; demo4.cfg
```



As you may have noticed, the most useful functions here are the sines and cosines. This is not a difficult stuff and after some training these expressions can be used quite intuitively. What definitely helps is to be able to visualize these functions, for example with the excellent Desmos calculator¹¹.

For the complete list of available mathematical functions, see the Appendix A.

Note: In all the previous examples the length of the horn is set constant. This means that no matter what the profiles are, the horn can still be mounted in a flat baffle. We could also set the length variable - then the horn would be suitable only as free standing (including the analysis).

¹¹ <http://www.desmos.com/>

6.4.2 Using guiding curves

Another approach to define a shape is by using a *guiding curve*. Either a superellipse or a superformula can be used for this purpose. The horn wall will go through this curve at the specified distance(s) from the throat.

So let's choose some nice superformula. Again, the Desmos calculator can be of great help here¹²:

The curve in the following picture is defined as $\{\mathbf{a}, \mathbf{b}, \mathbf{m}, \mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3\} = \{0.95, 0.82, 4, 0.8, 7, 1.9\}$. The bold green line in the picture is the "width" of the curve, referenced further.

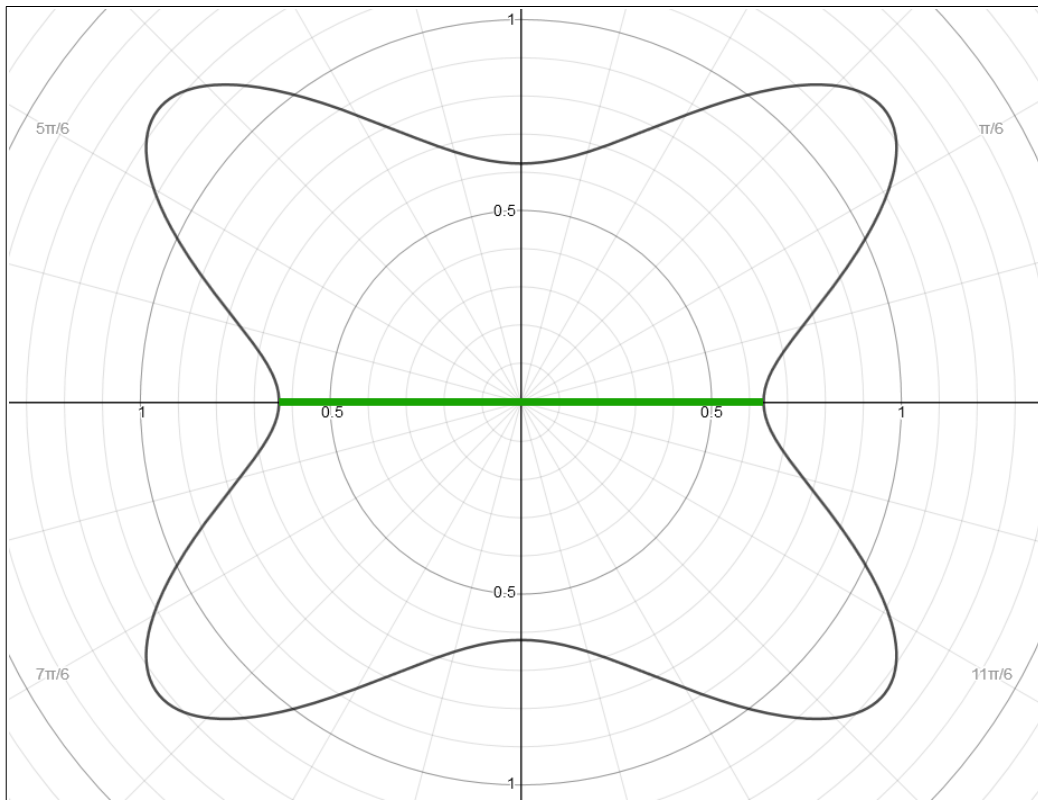


Fig 12: Superformula example (Desmos calculator)

In the definition file this curve is defined as follows:

```
GCurve.Type = 2 ; superformula
GCurve.SF = 0.95,0.82,4,0.8,7,1.9 ; values of a,b,m,n1,n2,n3
```

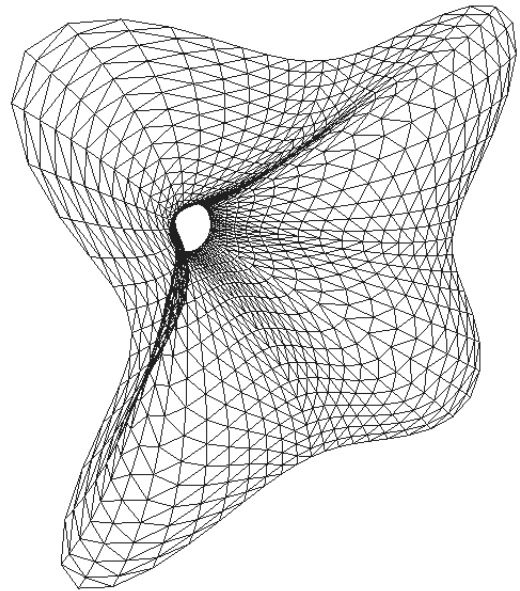
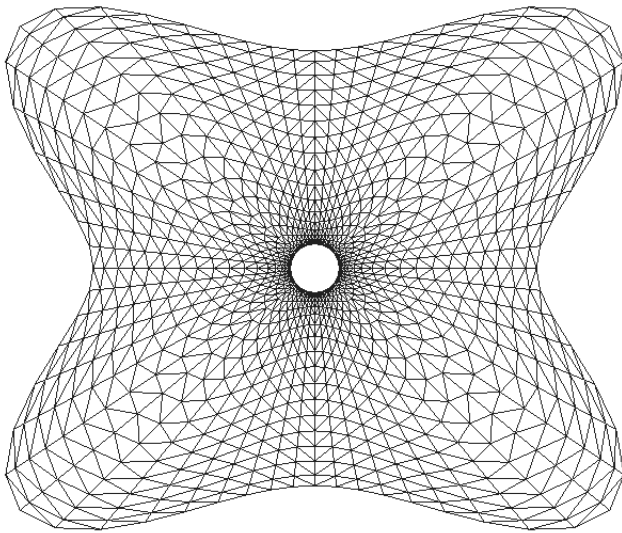
So far this was only for the shape itself. We also have to set the absolute size (length of the green line):

```
GCurve.Width = 68 ; [mm]
```

Finally we need to set the distance of the curve from the throat.

```
GCurve.Dist = 0.5 ; half the length
```

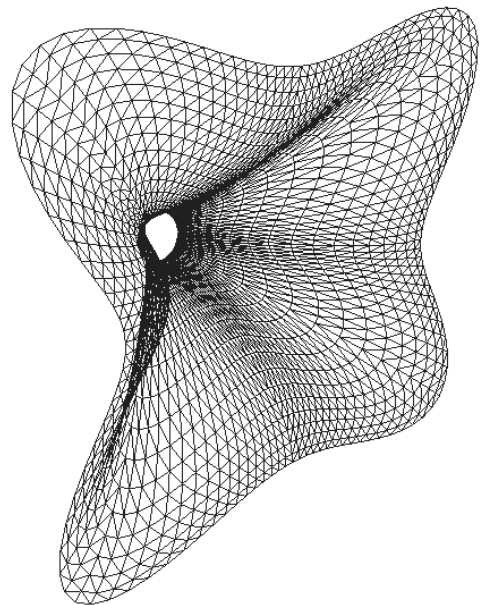
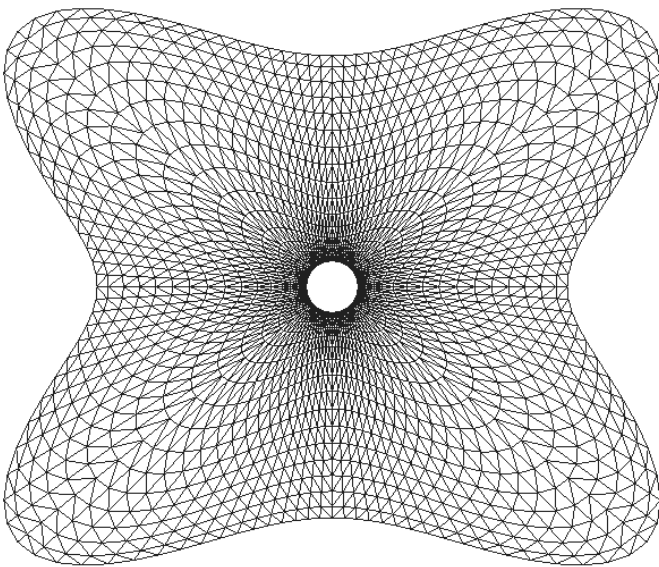
¹² <https://www.desmos.com/calculator/pw8t05wbav>



The grid is getting a bit coarse. This can be improved simply by increasing the number of profiles:

```
Mesh.AngularSegments = 120
```

```
; demo5.cfg
```



6.4.3 Using the morph feature

So far in our examples we let the overall shape to be determined by the basic geometry specified. By using the morph feature we can transform this "raw" shape so that the mouth outline will be a different curve of our choice (see [1]).

Let's take our last example (demo5.cfg) and set the mouth outline to rectangular:

```
                                ; demo6.cfg
Morph.TargetShape = 1           ; morph to a rounded rectangle
Morph.FixedPart = 0.0           ; start at throat
Morph.Rate = 3                  ; how fast is the transition (1=fastest/abrupt)
Morph.CornerRadius = 12         ; [mm]

Mesh.CornerSegments = 4         ; number of profiles reserved for corners
```

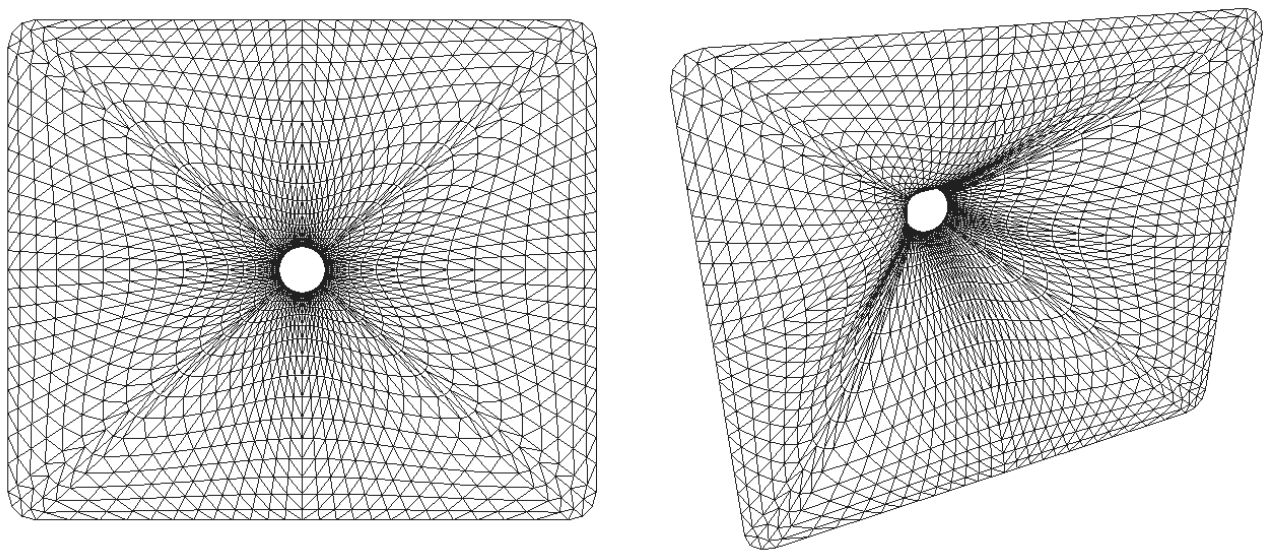
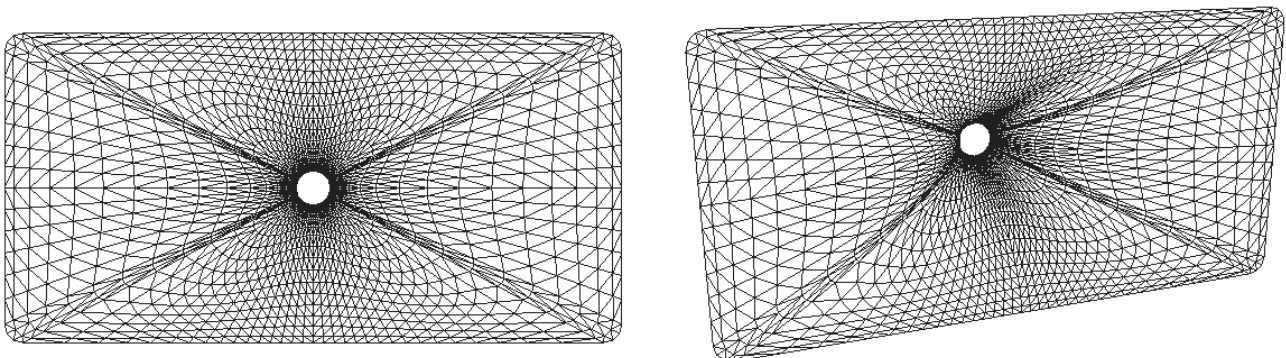


Fig 13: demo6.cfg

If no new dimensions are specified, a circumscribed rectangle is used as a new outline. In this case its dimensions were 254 x 216 mm. We could however force the dimensions to anything we want:

```
Morph.TargetWidth = 400         ; [mm]
Morph.TargetHeight = 200        ; [mm]
Morph.AllowShrinkage = 1        ; force the smaller height
```



6.5 Modelling a complex source

In the previous examples we didn't define specific acoustic sources (via the `Source.*` items). If we don't specify a source explicitly, an output of an ideal compression driver is used for the BEM analysis, i.e. a purely spherical (or flat, depending on the throat opening angle) wavefront is assumed.

Let's now suppose we want to model a more complex source, for example a dome tweeter. First, we must define the shapes of all the parts that will be placed inside the throat of the horn and tell the program what are the driving elements. For this we use a *source definition script*. The Appendix B gives a complete reference for the syntax, along with an example. So let's take the example given in the appendix and show how it is used in a practice.

We have two options how to specify the source geometry in the definition file:

a) Save the source definition script into a separate file and make a link to this file:

```
Source.Contours = D:\projects\ath\cfg\dome.src ; demo7.cfg
```

b) Make the source definition script code part of the definition file itself:

```
Source.Contours = { ; demo7.cfg
    ; dome tweeter example
    zoff -2

    point p1 4.68 0 2
    point p2 0 14 0.5
    point p3 1 15 0.5
    point p4 0 16 0.5
    point p5 0 17 1
    cpoint c1 -18.59 0
    cpoint c2 0 15

    arc p1 c1 p2 1.0
    arc p2 c2 p3 0.75
    arc p3 c2 p4 0.25
    line p4 p5 0
    line p5 WG0 0
}
```

Let's take our first example (demo1.cfg) and modify it for the dome tweeter defined above. Besides setting the item `Source.Contours` as shown above we have to set the throat diameter to match the geometry of the tweeter:

```
Throat.Diameter = 34 ; [mm]
```

Finally, there's another important parameter that must be set when modelling axially vibrating sources:

```
Source.Velocity = 2 ; move in axial direction
```

We want the driving elements to move axially now. If we didn't set this item we would be modelling basically a pulsating spherical cap instead of a vibrating dome. The item `Source.Velocity` is also the only one from the `Source.*` items that has any relevance at this moment – all the other are ignored if `Source.Contour` is present.

Also note that we don't have to change anything else in the horn definition. The whole geometry stays the same by definition, now only for a different throat size (as a consequence of this the resulting outer size may change).

If we wanted to visually inspect the source we just defined we would have to look at either the MSH file or open the ABEC project. The acoustic source is not part of the STL file.

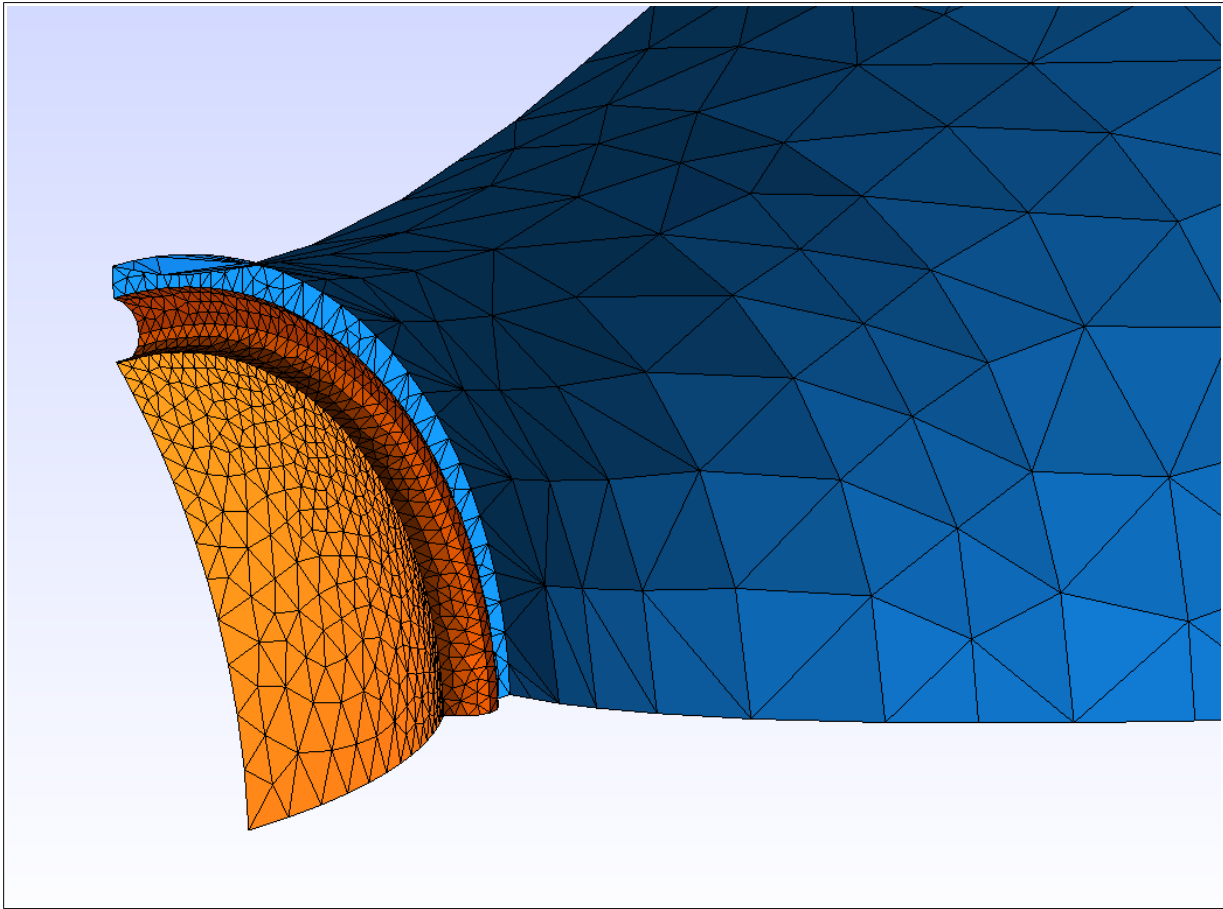


Fig 14: Dome tweeter in a waveguide (Gmsh visualization)

By employing the source definition script virtually any axisymmetric source can be defined, theoretically including a compression driver with its phase plug. This was never actually tried however. Such a task will probably require some more elaborate and convenient modelling techniques. This may be implemented in some future version of the tool.

6.5.1 External Shaping Plug

External Shaping Plug (ESP) is a device proposed for converting a flat wavefront of an acoustic source into a spherical wave before the sound is let to propagate freely in an attached horn.

To learn how to use an ESP in your projects please consult the Application Note 1 [3], available at the ATH website¹³.

¹³ <https://at-horns.eu>

6.6 Using circular arc profiles

Using a circular arc as the horn profile (`Throat.Profile = 3`) is the most simple option in a sense that the only actual parameter is the radius of the arc that simply connects the throat with the mouth outline (which can be set either by a guiding curve or an explicit nominal coverage angle). No additional mouth termination is used.

There are two (mutually exclusive) ways to define an arc, i.e. its radius:

- Explicit radius definition:

```
CircArc.Radius = 100 ; [mm]
```

- Radius determined from the required mouth termination angle:

```
CircArc.TermAngle = 1 ; [deg] ; demo11.cfg
```

Notes:

- Both the above definitions can use a mathematical expression as their value.
- The profile can be rolled-back in case of a free standing horn as any other profile.

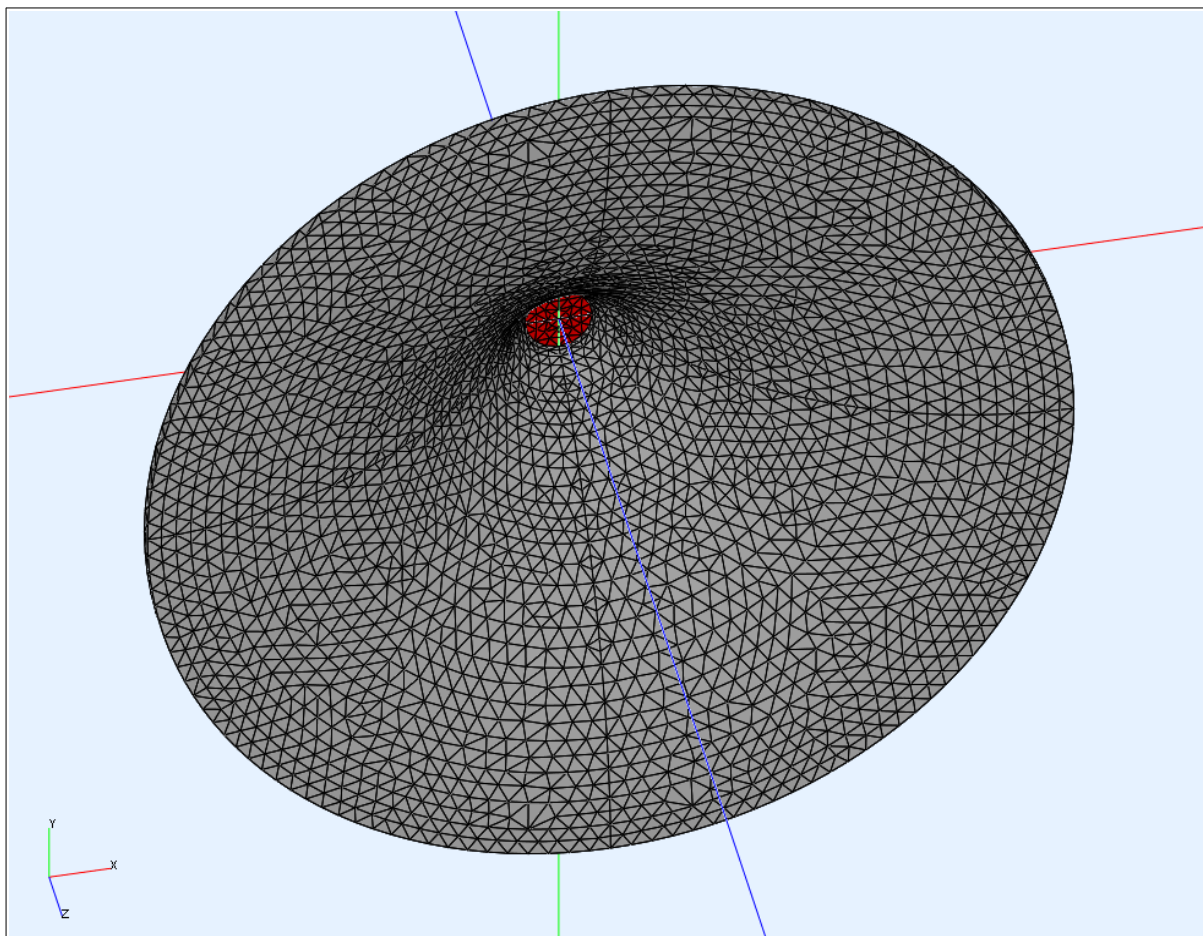


Fig 15: Circular arc profile (demo11)

6.7 Setting subdomain interfaces

So far we didn't care about the BEM subdomain interfaces as described in 3.1.1. By default one subdomain interface is automatically placed at the end of the profile, separating the interior of the horn and the exterior subdomain. In a case of an infinite baffle simulation there must always be an interface enclosing the elements "behind" the baffle, i.e. there must be at least one interface for this purpose. For free standing horns this interface can be manually placed anywhere along the length or even left out completely (then the only subdomain left is the exterior). In both cases we can add as many interior subdomains as we like.

We setup the interfaces with the `Mesh.SubdomainSlices` item by listing the slices where we want the interfaces to be placed at. Let's take the example `demo7.cfg` and proceed with that. The grid in this example has 20 slices (`Mesh.LengthSegments = 20`). If no explicit interface definition was set it would be automatically set by default as if it was like this:

```
Mesh.SubdomainSlices = 19      ; interface at slice 19 [0,...,Mesh.LengthSegments-1]
Mesh.InterfaceOffset = 0       ; [mm]
Mesh.InterfaceDraw = 0         ; [mm]
```

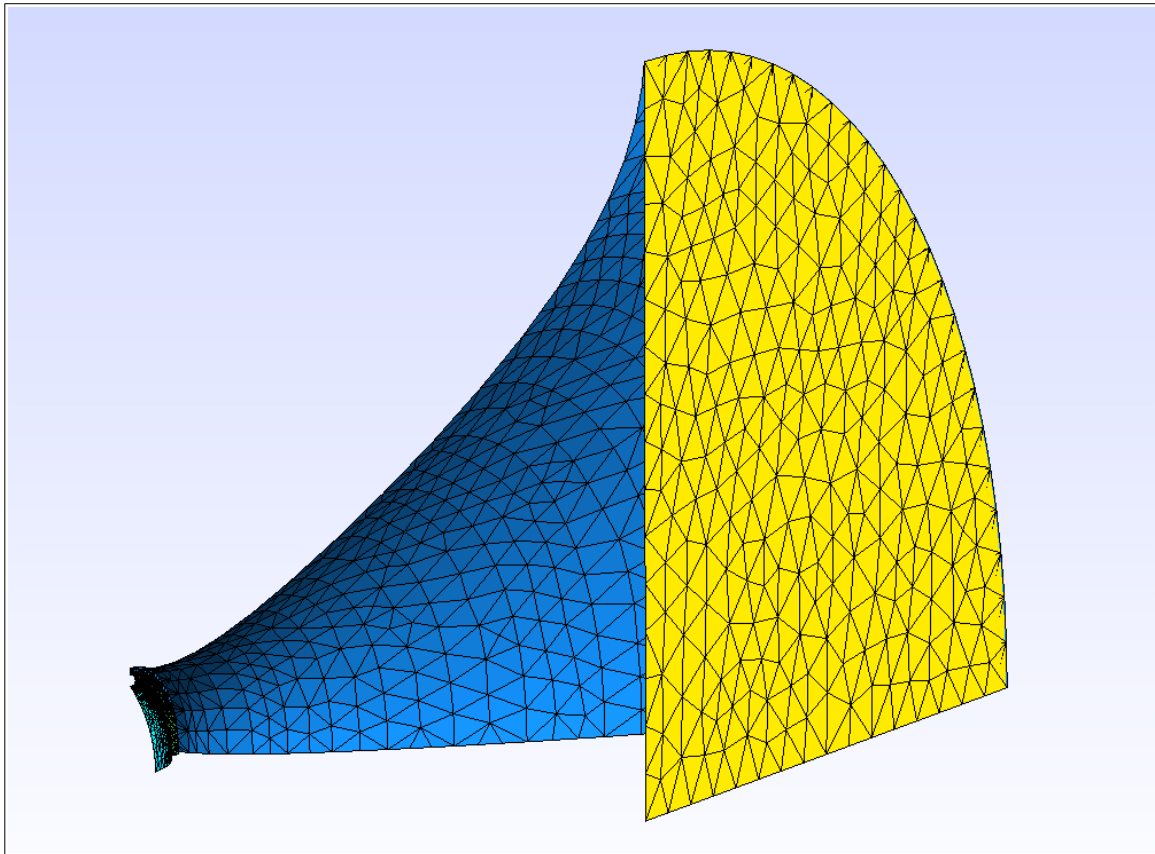


Fig 16: The default subdomain interface

Note: For free standing horns we can remove even the default interface and leave everything in the exterior subdomain simply by setting the `Mesh.SubdomainSlices` to empty value:

```
Mesh.SubdomainSlices =
```

There are situations where protruding the interface joint may give more stable BEM results:

```
Mesh.InterfaceOffset = 5      ; [mm]
```

Note that the sharp angle at the interface-horn joint has been removed, at the cost of adding some more boundary elements:

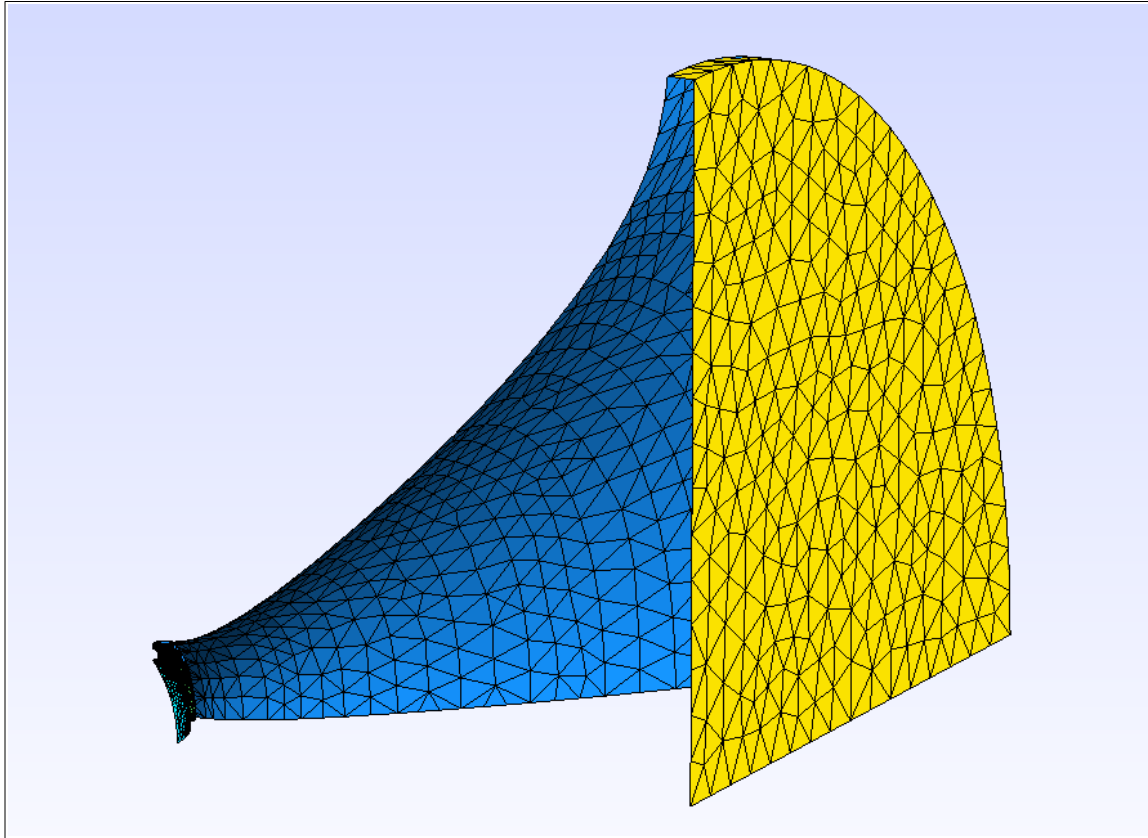


Fig 17: Interface offset

Let's now add one more interior subdomain. The `Mesh.*` items mentioned above are in fact arrays of numbers - so far we defined and used only their first elements. We can add an interior subdomain easily by placing another interface, e.g. at slice index 8, before the existing interface at slice 19 (which remains exactly the same):

```
Mesh.SubdomainSlices = 8,19      ; slice indices for interfaces  
Mesh.InterfaceOffset = 0,5       ; interface offsets [mm]  
Mesh.InterfaceDraw = 0,0        ; interface draws [mm]
```

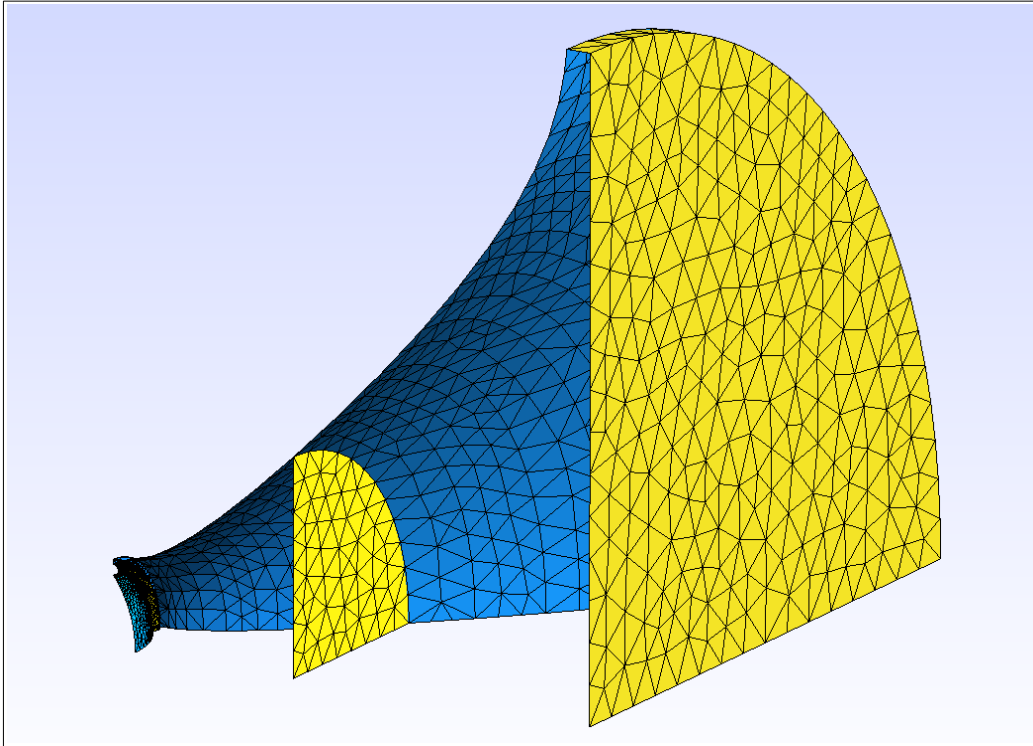


Fig 18: Second interior subdomain added

One more paramter can be set independently for each interface - its draw:

```
Mesh.InterfaceDraw = 12,0 ; [mm]
```

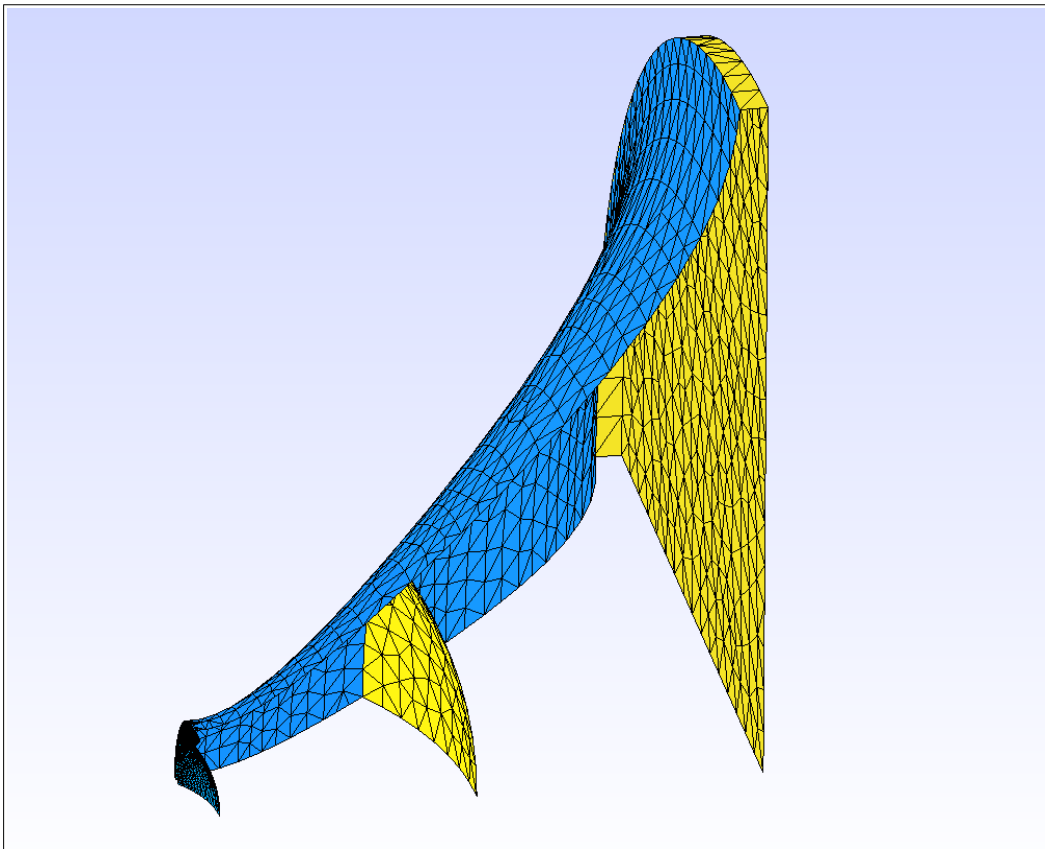


Fig 19: Interface draw

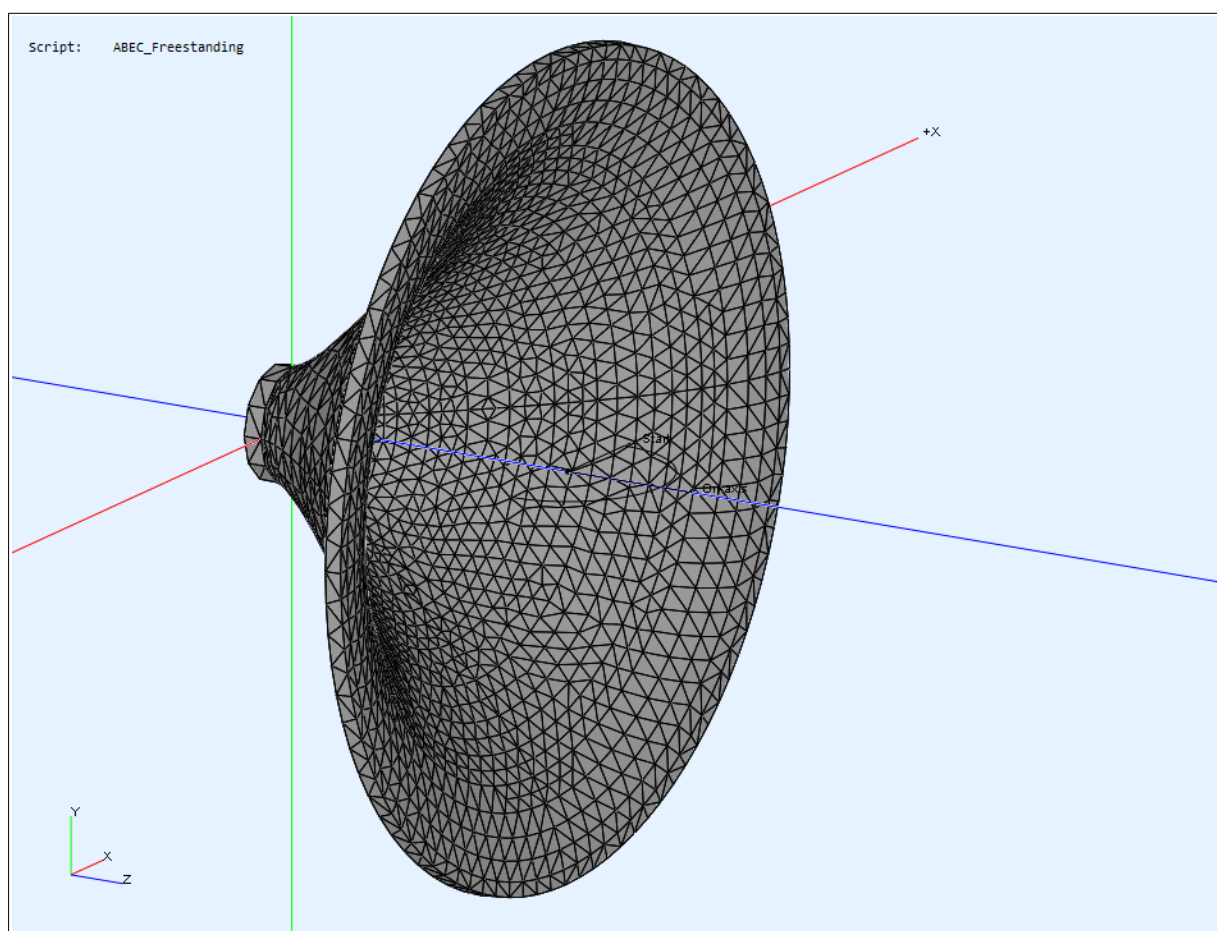
6.8 Free standing horns

In the previous examples an infinite baffle was used for the BEM analysis, i.e. the mouth of the horn was placed in a infinite plane. That also means that the horns all had a constant length, i.e. a flat mouth outline, otherwise the program would refuse to do it.

There's another option – free standing horn, i.e. a horn placed in a free field. The basic difference is that in this case we must also model the back side of the horn, making it a solid object with a volume.

Let's take our first example (demo1.cfg) and make a free standing horn from that:

```
ABEC.SimType = 2                                ; free field analysis          ; demo8.cfg
Mesh.RearShape = 1                               ; fully modeled
Mesh.RearResolution = 10.0                       ; [mm]
Mesh.WallThickness = 5.0                         ; [mm]
```



Now there's no infinite baffle and the horn is fully modeled with its rear side. The thickness of the wall is set with `Mesh.WallThickness`. The item `Mesh.RearResolution` sets the mesh resolution for the rear surface of the horn.

6.8.1 Mouth rollback

Mouth rollback is a vital ingredient for a good performance of a free standing waveguide, eliminating the diffraction and reflection at the termination to a great degree.

The rollback is always performed as the last step of the calculation and is applied on a shape the waveguide would have without the rollback feature used - the waveguide is basically "bended". For that reason the final dimensions of the device can change, including the prescribed length.

Let's take our previous example (demo8.cfg) and add a mouth rollback:

```
Rollback = 1 ; do the rollback
Rollback.StartAt = 0.5 ; start at 50% of the profile length
Rollback.Angle = 150 ; how much to actually roll it back [deg]
```

Note that it may be necessary to adjust the subdomain interface parameters, if used (see 6.7 for details), otherwise it could intersect the horn surface.

```
Mesh.SubdomainSlices = 12 ; slice index
Mesh.InterfaceDraw = 20 ; [mm]
```

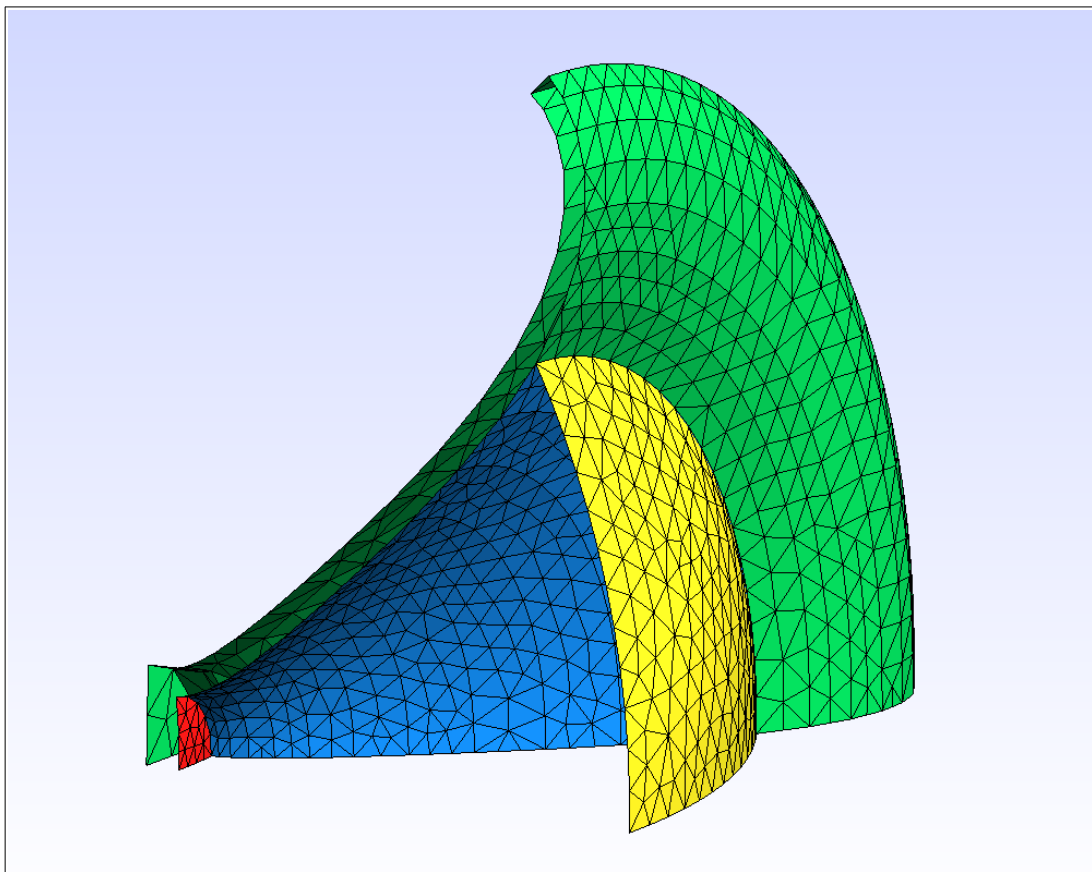


Fig 20: Rolled-back mouth (interior=blue; exterior=green)

Note: In a case of free standing horn, the subdomain interface can be removed completely and leave only the exterior domain - see 6.7.

6.9 Simulating a source without a waveguide

Ath offers a possibility to simulate an acoustic source without any waveguide. In such case, typically a user-defined source will be specified via the Source Definition Script (see 6.5). This way any direct-radiating transducer can be simulated. To show how it's done let's take the example `demo7.cfg` and modify it accordingly, simulating a direct-radiating tweeter in a flat baffle:

```
Length = 0 ; eliminating the horn ; demo10.cfg
Mesh.LengthSegments = 1 ; must be set to 1
```

Because the whole mesh is so small in this case, we can increase the mesh resolution considerably without worrying too much about computation time (remember that the resolution of the source mesh is set independently in the source definition script):

```
Mesh.ThroatResolution = 2.0 ; [mm]
Mesh.InterfaceResolution = 2.0 ; [mm]
```

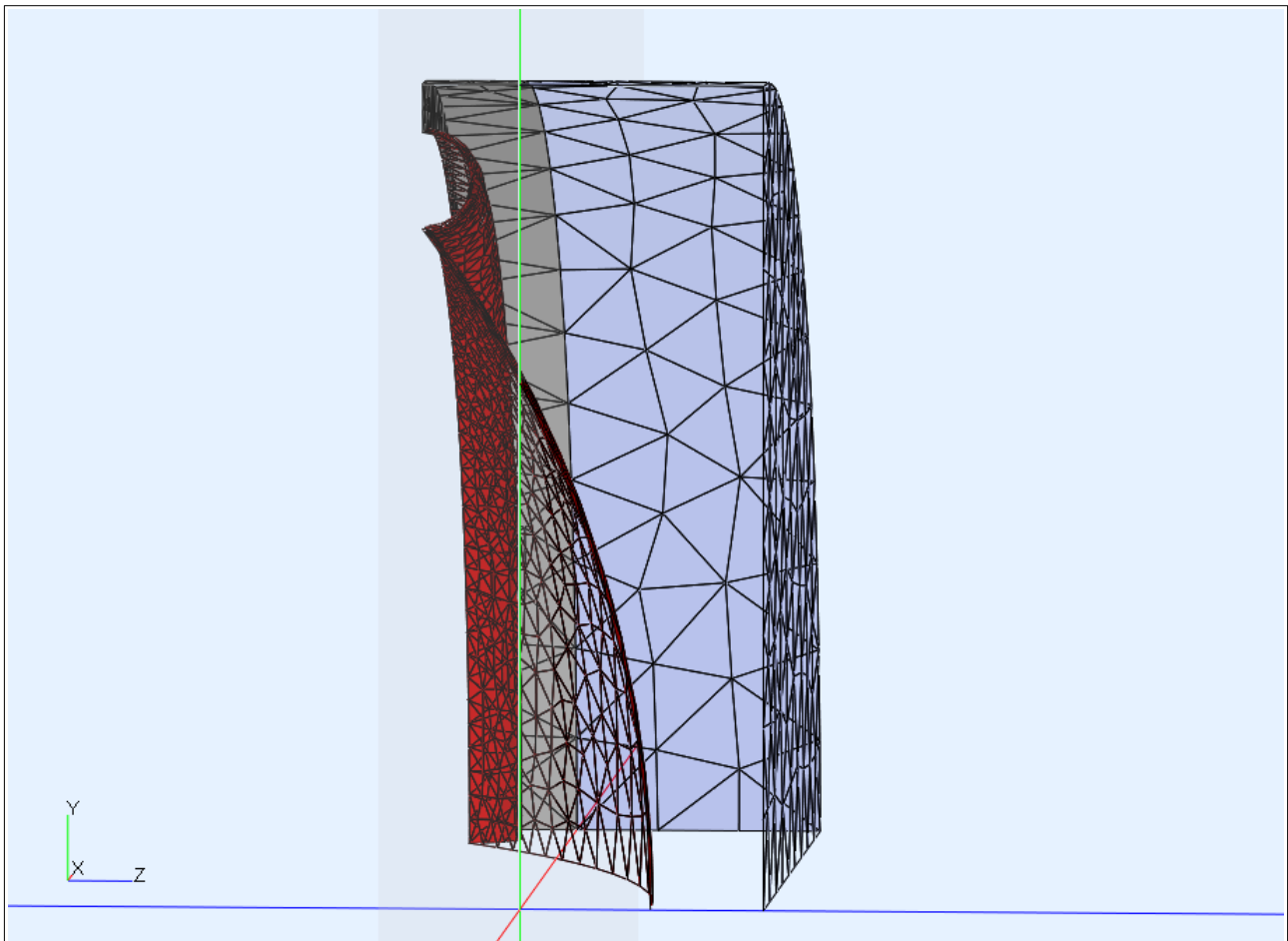


Fig 21: Dome tweeter in an infinite baffle (ABEC display)

Please note, as this is an IB simulation, that in the above picture the interface reaches further in front of the dome apex, i.e. the whole dome is covered by the interface boundary. This must be the case, otherwise the surfaces would cross. Eventually the parameter `Mesh.InterfaceOffset` can be easily increased to ensure this doesn't happen.

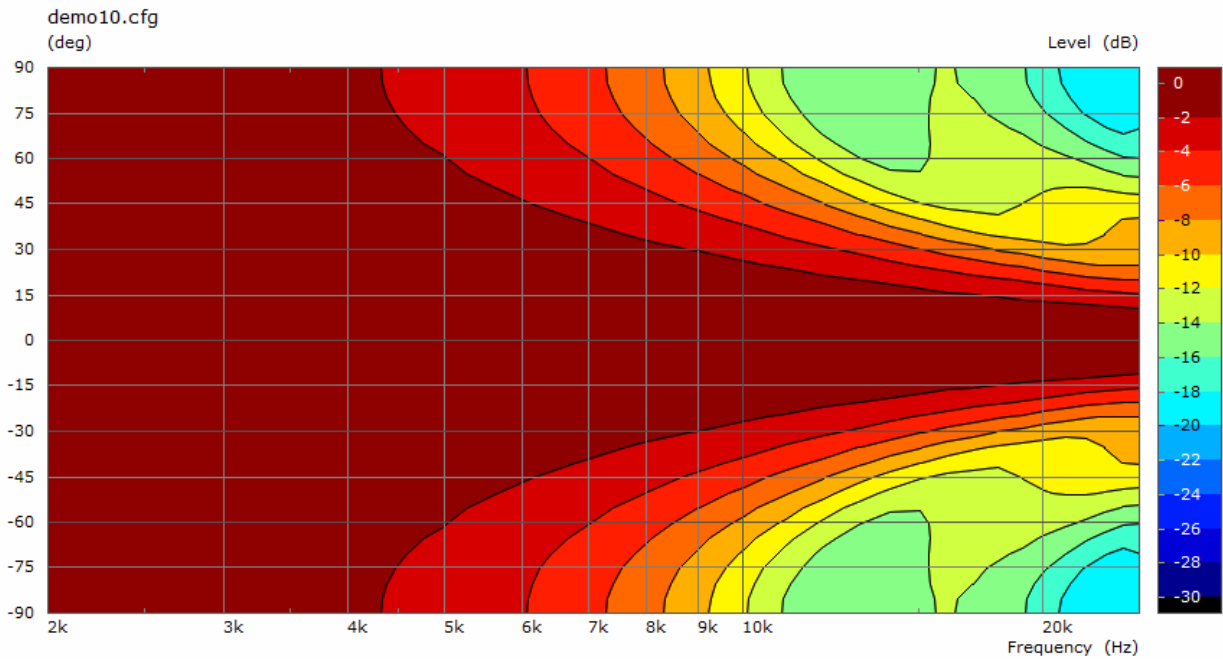


Fig 22: demo10.cfg / dome tweeter in an infinite baffle - BEM analysis results (polar map)

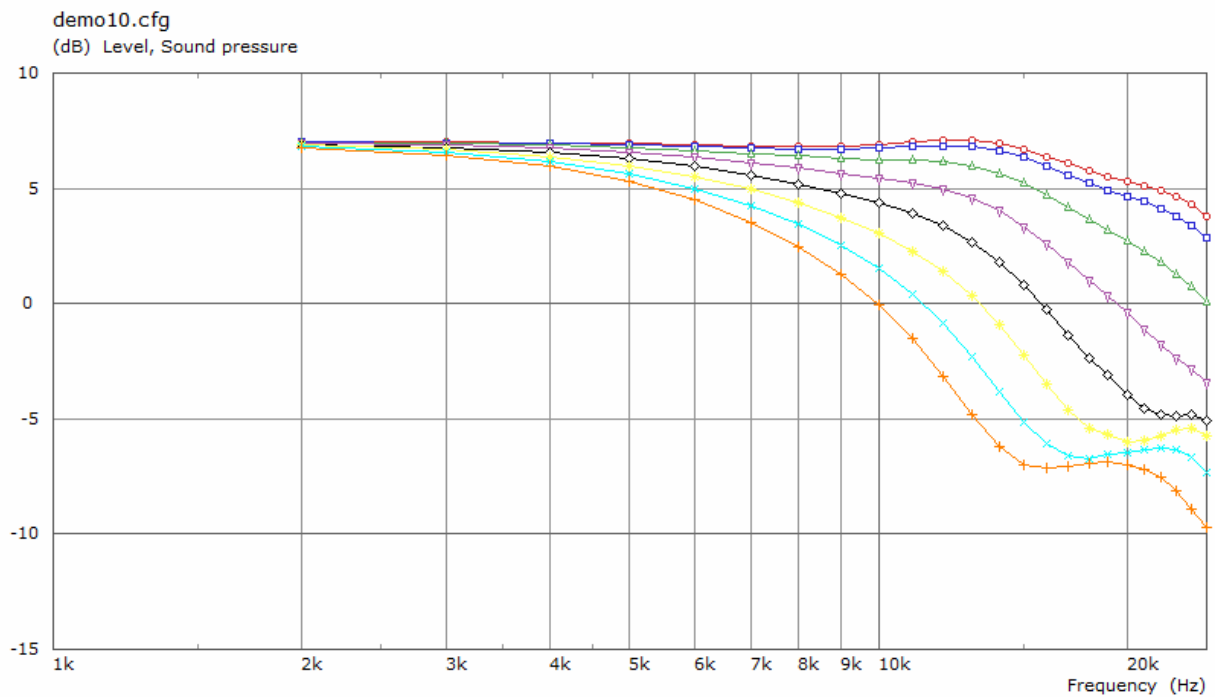


Fig 23: demo10.cfg / dome tweeter in an infinite baffle - BEM analysis results

Note that in the above example we set a linear step of the frequency points for the analysis to capture the higher frequencies better:

```
ABEC.Abcissa = 2 ; linear frequency step
```

6.10 CircSym mode

In the "CircSym" mode, only strictly axisymmetric devices are analyzed, without the need to create their full-3D mesh - all it takes for axisymmetric devices is a single profile. This way the simulation complexity is greatly reduced, as are the computation times. Typically a much higher resolution can be achieved in much shorter time than with a general 3D simulation. Let's take our very first example (demo1.cfg) and convert it into axisymmetric mode. It doesn't matter whether the device described in the script itself is axisymmetric or not - we force the CircSym mode by selecting a single chosen profile for the analysis:

```
ABEC.SimProfile = 0 ; 0 = profile number ; demo1C.cfg
```

The item definition above alone does the job. However, there are several more things we want to change from the original script. First, the mesh resolution needs to be increased considerably, otherwise we would likely get very poor results:

```
ABEC.MeshFrequency = 40000 ; [Hz]
```

Also the frequency resolution can be increased without worrying too much about computation time:

```
ABEC.f1 = 200 ; [Hz]
ABEC.f2 = 20000 ; [Hz]
ABEC.NumFrequencies = 100
```

In the CircSym mode the items defining the mesh resolution are ignored. We could leave them without any effect, delete or just comment them out:

```
; Mesh.ThroatResolution = 4.0 ; [mm]
; Mesh.InterfaceResolution = 8.0 ; [mm]
```

In this example let's also remove the interface offset as it seems to do nothing useful in this case (interface is now simply the line P23 - P20 in the Fig. 24):

```
; Mesh.InterfaceOffset = 5.0 ; [mm]
```

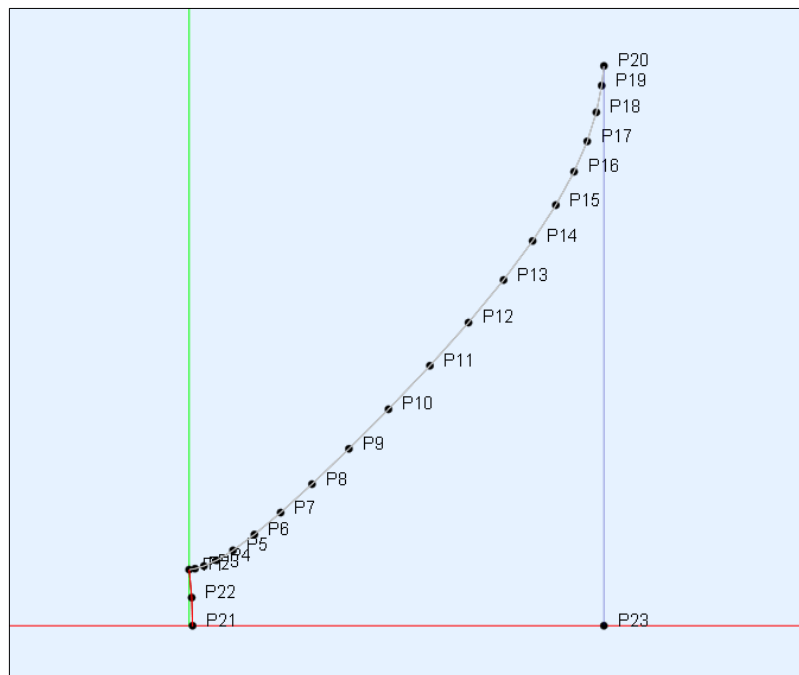


Fig 24: CircSym profile definition (ABEC visualization)

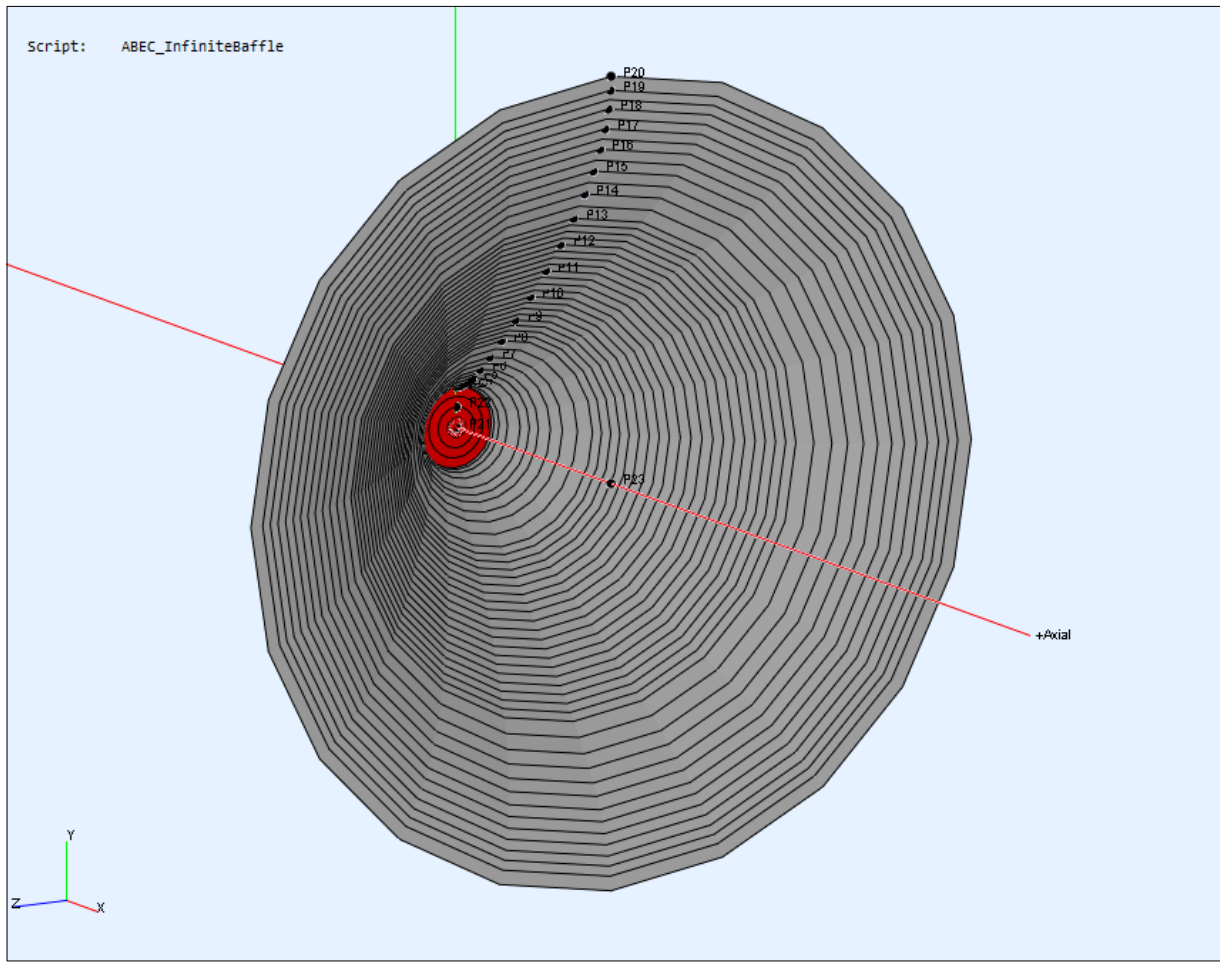


Fig 25: CircSym mesh definition (ABEC visualization)

Figures 24 and 25 show the resulting circular mesh as displayed in ABEC. The number of profile points (P1 - P20 in Fig. 24) is still controlled by the item Mesh.LengthSegments:

```
Mesh.LengthSegments = 20
```

In this particular example it seems high enough but generally the number of points (i.e. number of line segments approximating the profile) can be increased greatly - we want a high mesh resolution anyway.

Let's now run the BEM analysis in ABEC and see the results.

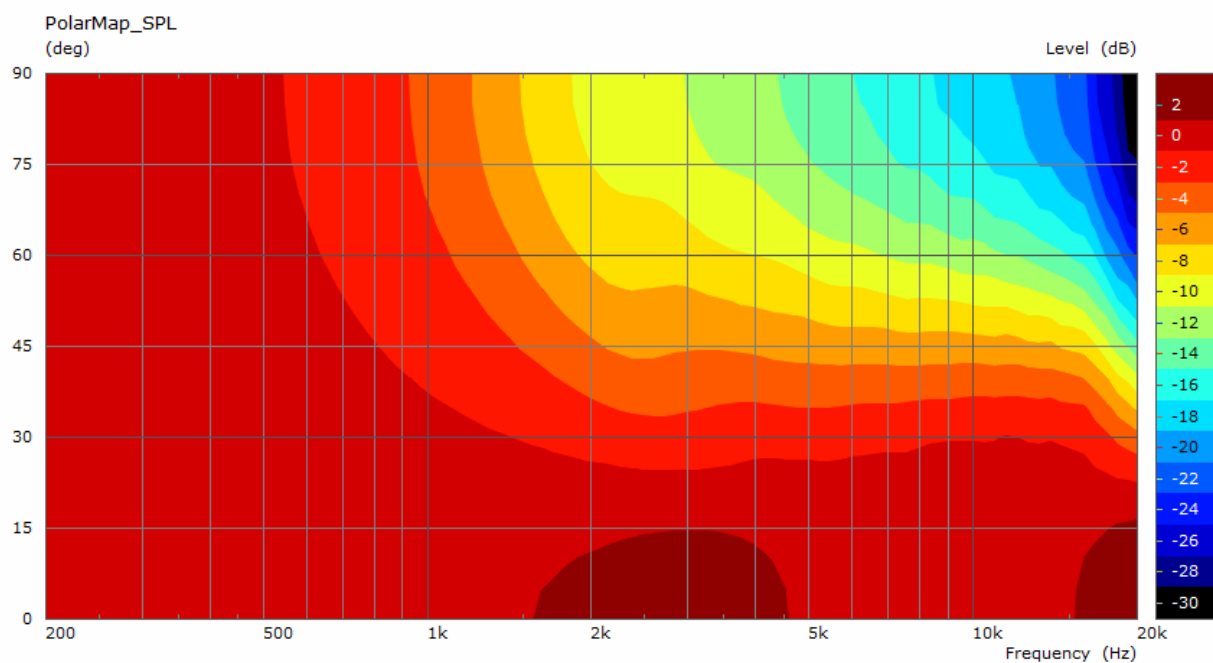


Fig 26: demo1C.cfg - Resulting polar map (VACS visualization)

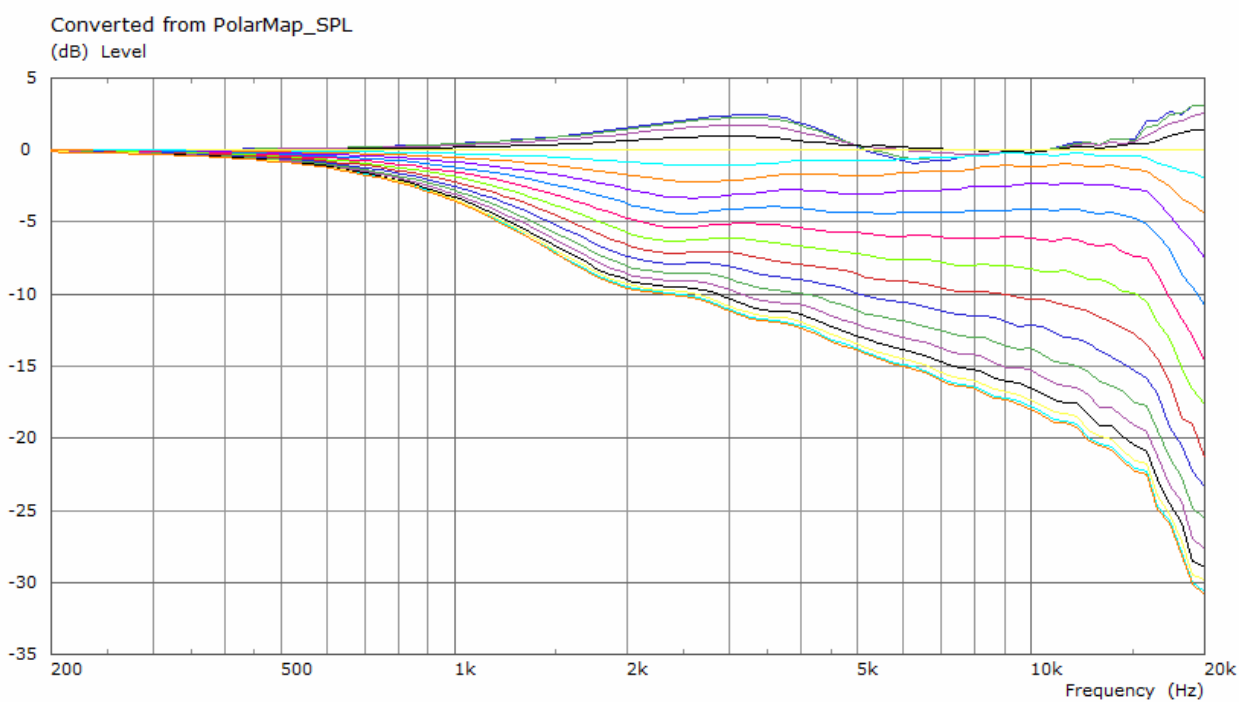


Fig 27: demo1C.cfg - individual polars (converted from polar map in VACS)

6.11 CircSym mode (free standing case)

In this section we show an example of a free standing waveguide analyzed in the CircSym mode. Let's take the demo8.cfg (see 6.8) and modify it accordingly.

Before anything else, however, let's add some mouth rollback:

```
Rollback = 1 ; activates the rollback ; demo8C.cfg
Rollback.StartAt = 0.6 ; start at 60% of profile length
Rollback.Angle = 180 ; termination angle [deg]
```

In the free standing case we can disable the subdomain handling and use only one (exterior) subdomain instead:

```
Mesh.SubdomainSlices = ; intentional assignment of an empty value
```

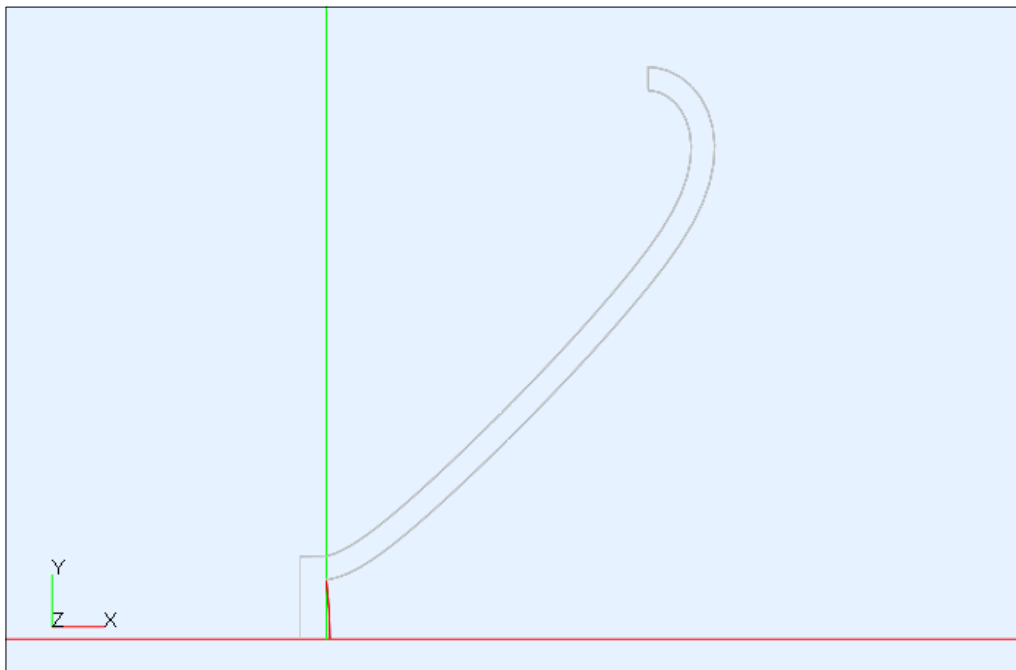


Fig 28: demo8C.cfg - free standing CircSym profile (no interfaces)

Finally, we set the ABEC/BEM parameters:

```
ABEC.SimType = 2 ; 2 = free-air
ABEC.SimProfile = 0
ABEC.f1 = 200 ; [Hz]
ABEC.f2 = 20000 ; [Hz]
ABEC.NumFrequencies = 100
ABEC.MeshFrequency = 30000 ; [Hz]

ABEC.Polars:SPL = {
  MapAngleRange = 0,180,37 ; 0 - 180 deg with 5 deg step
  NormAngle = 20 ; [deg]
  Distance = 3 ; [m]
}
```

Note that we set the polar map to include all the angles from 0 to 180 degrees off-axis, i.e. covering the full sphere. Tag "SPL" is the default tag that is used to generate reports (6.14).

Let's show the results of the BEM analysis:

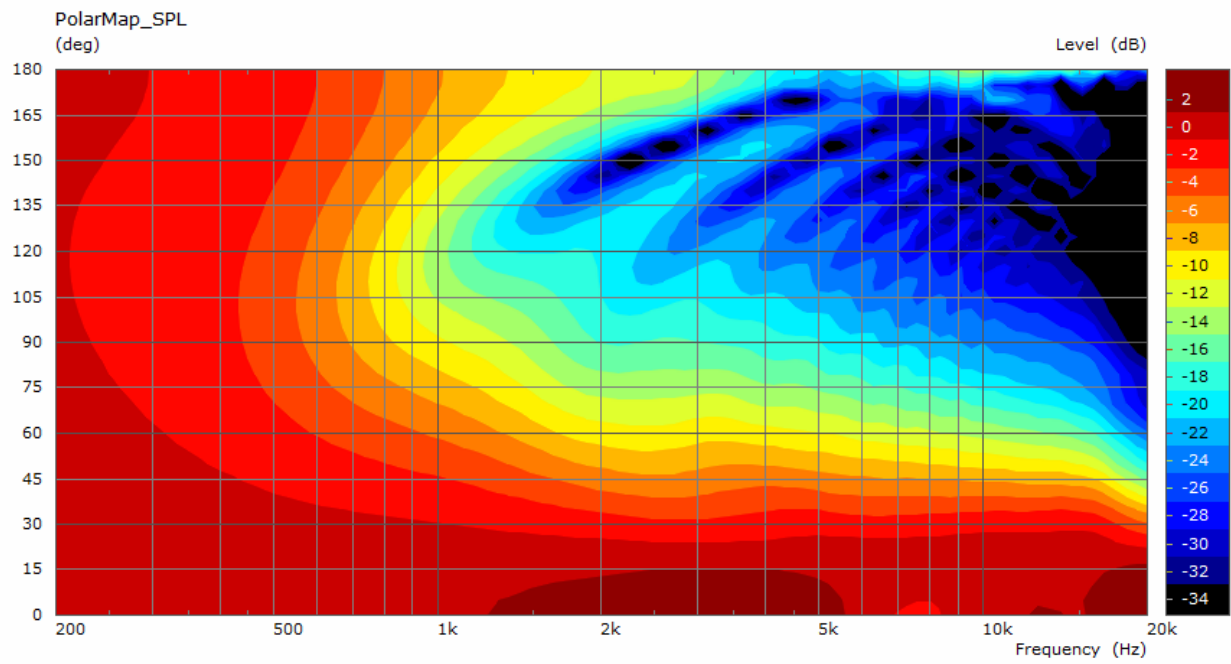


Fig 29: demo8C.cfg - Polar map (0 - 180 deg)

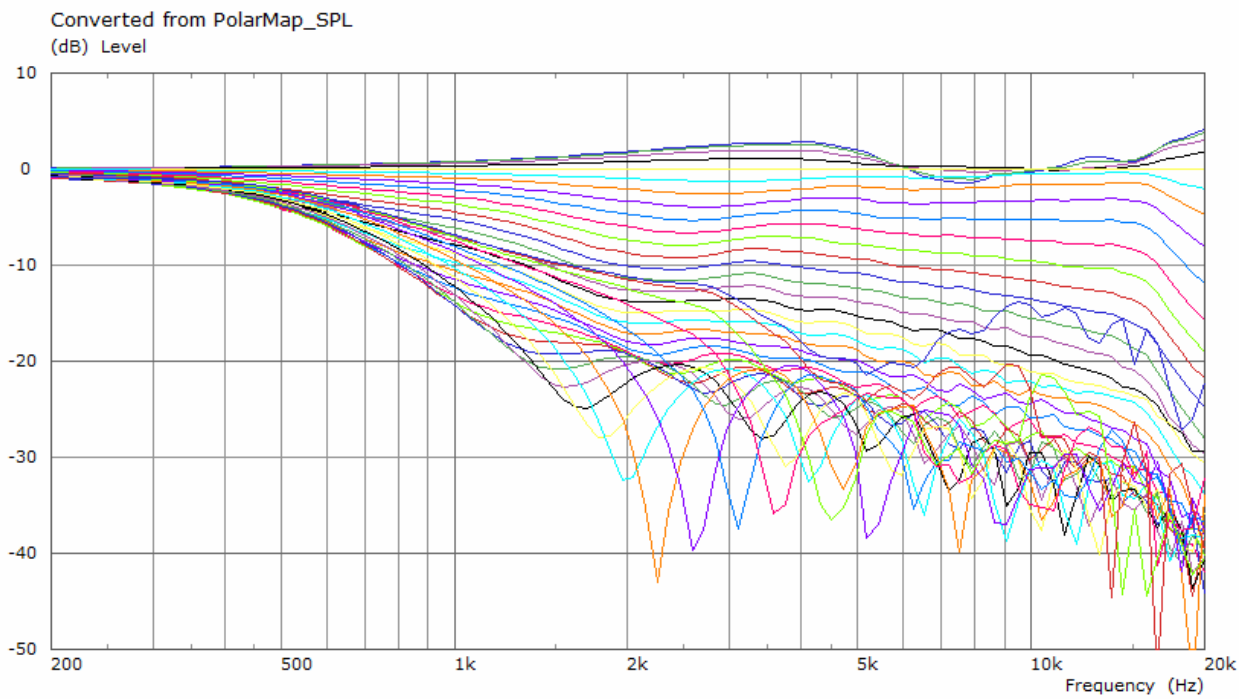


Fig 30: demo8C.cfg - Individual polars (0 - 180 deg)

We will use these results to illustrate the reporting feature implemented in Ath in the following example.

6.12 Adding enclosures

Since Ath 4.8.2 there's a possibility to add an enclosure to the project, i.e. to build the device into a cabinet of defined shape and then simulate the whole in a free field.

At the moment there are two ways to define an enclosure:

- 1) use a quick pre-defined shape (a rectangular box, optionally with rounded or chamfered edges),
- 2) specify a user-defined ground plan for an "extruded" cabinet.

In all cases the enclosure is specified by the item **Mesh.Enclosure**. Two examples are presented in the following paragraphs, illustrating the features available.

6.12.1 Pre-defined enclosure

The first example is a waveguide in a simple rectangular box with rounded edges. Such cabinet is already pre-defined in Ath and can be used right away.

Suppose we already have a waveguide geometry defined (only the most relevant items are shown):

```
Throat.Diameter = 25.4
Throat.Angle = 10
Coverage.Angle = 45
Length = 60
Term.s = 1
Term.n = 4
OS.k = 1
Term.q = 0.98

Mesh.LengthSegments = 24
Mesh.AngularSegments = 64
Mesh.ThroatResolution = 4
Mesh.MouthResolution = 8           ; mesh element size for the horn outline
Mesh.SubdomainSlices =             ; intentionally empty value (no subdomains)
```

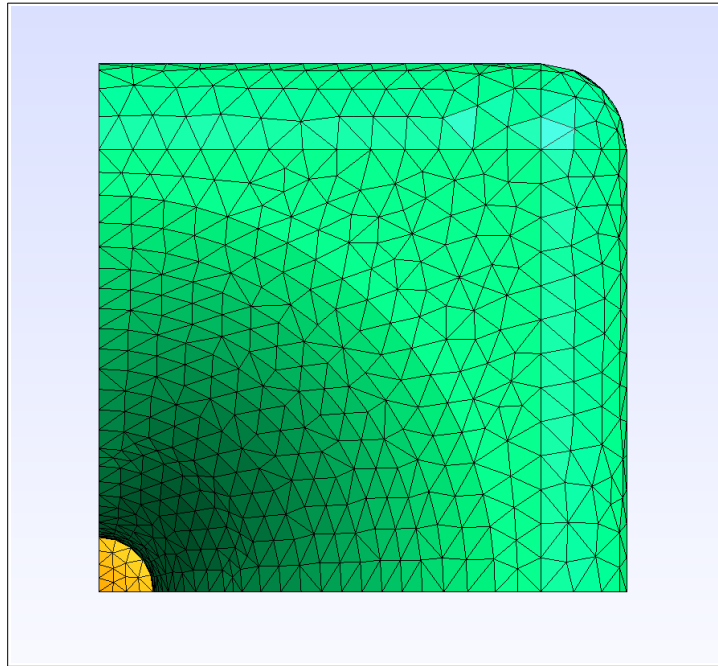
To put the waveguide into a cabinet we add the following lines. The outer cabinet dimensions are given by the **Spacing** item (distances from the waveguide outline to the outer edges of the enclosure):

```
Mesh.Enclosure = {
  Spacing = 30,30,30,200           ; edge distances (left,top,right,bottom) [mm]
  Depth = 200                     ; enclosure depth [mm]
  EdgeRadius = 20                  ; radius of the edge treatment [mm]
  EdgeType = 1                    ; 1=rounded, 2=chamfered
  FrontResolution = 8,8,16,16      ; front side mesh element size (q1,q2,q3,q4) [mm]
  BackResolution = 20,20,20,20     ; back side mesh element size (q1,q2,q3,q4) [mm]
}
```

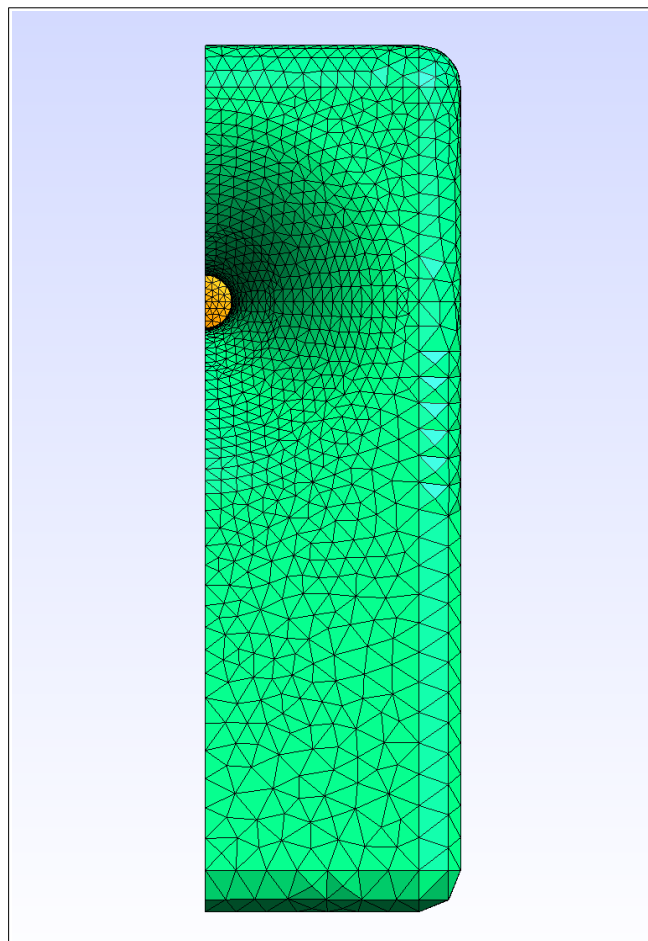
In fact, only the **Depth** item is mandatory in this case, all other items would be assigned their default values if omitted. What's important with enclosures in general is to set the nominal mesh element sizes properly - too low and the BEM results won't be reliable, too high and the number of elements quickly exceeds what's manageable. Typically, it's possible to start with smaller elements at and near the waveguide and progressively increase the element size towards back of the cabinet. This is specified with the items **FrontResolution** and **BackResolution** - each consisting of four numbers, corresponding to the corners of the four quadrants (see **Mesh.Quadrants**).

Note: If only a lower number of values are specified, e.g. "FrontResolution = 8", the remaining values are automatically set to the last value present.

By default only the first quadrant is meshed to be employed in quarter-space symmetry:

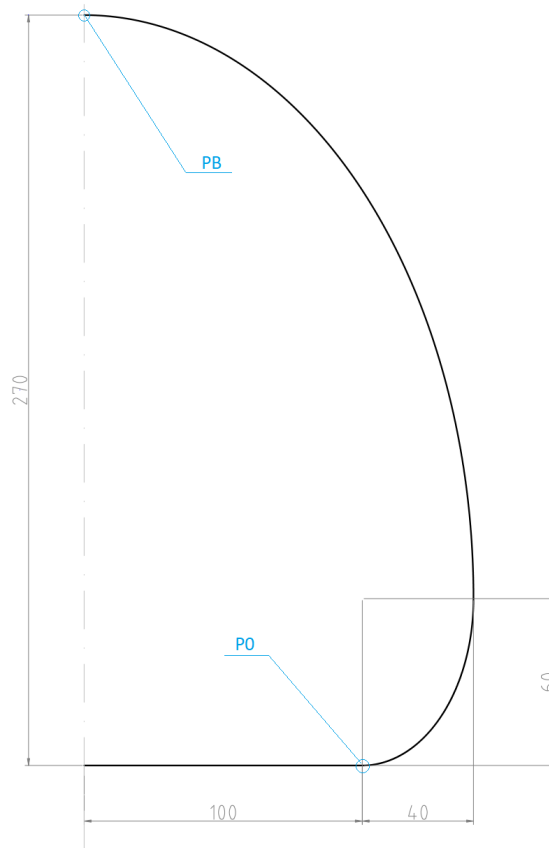


To use more quadrants we simply set the item **Mesh.Quadrants** to the desired value (see 4.1.4). For example, to simulate an enclosure in half-space symmetry, with a space left for a low frequency transducer under the waveguide, we need to use quadrants 1 and 4, i.e. set **Mesh.Quadrants = 14**:



6.12.2 User-defined enclosure plan

The second option for an enclosure definition is to use a custom ground plan. Suppose we want to model an enclosure with the ground plan shown on the following picture (front baffle being the horizontal line at the bottom).



Ground plan is simply a consecutive list of 2D entities (lines, arc, curves) connecting the points $P0=[*,0]$ and $PB=[0,*]$ (see the drawing), in this order. This particular example is composed of just two elliptical arcs. Syntax of a ground plan definition is very much similar to the *Source Definition Script* (SDS) - for a complete reference please see the Appendix C.

In this case the ground plan definition (labeled "my_plan") has the following form:

```
my_plan = {
  point P0 100 0 8
  point P1 140 60 12
  point PB 0 270 20
  cpoint C1 100 60
  cpoint C2 0 60
  ellipse P0 C1 P1 C1
  ellipse P1 C2 PB PB
}
```

To use it, simply specify this ground plan in the **Mesh.Enclosure** section as the **Plan** item:

```
Mesh.Enclosure = {
  Plan = my_plan
  Spacing = 0,30,0,200
  FrontResolution = 8,8,16,16
  BackResolution = 20,20,20,20
}
```

The points **P0** and **PB** are mandatory and must be always explicitly defined. The rest is up to the user.

The resulting mesh (half-space symmetry, see **Mesh.Quadrants**) is shown on the following picture. Note that the nominal mesh element sizes can be freely specified along the curve from the front to the back (i.e. in each control point).

A short note on the **Spacing** item: In this case only the "top" and "bottom" values are used, the other ignored. Also note that the front baffle is always assumed flat and must be big enough to fit the waveguide defined.

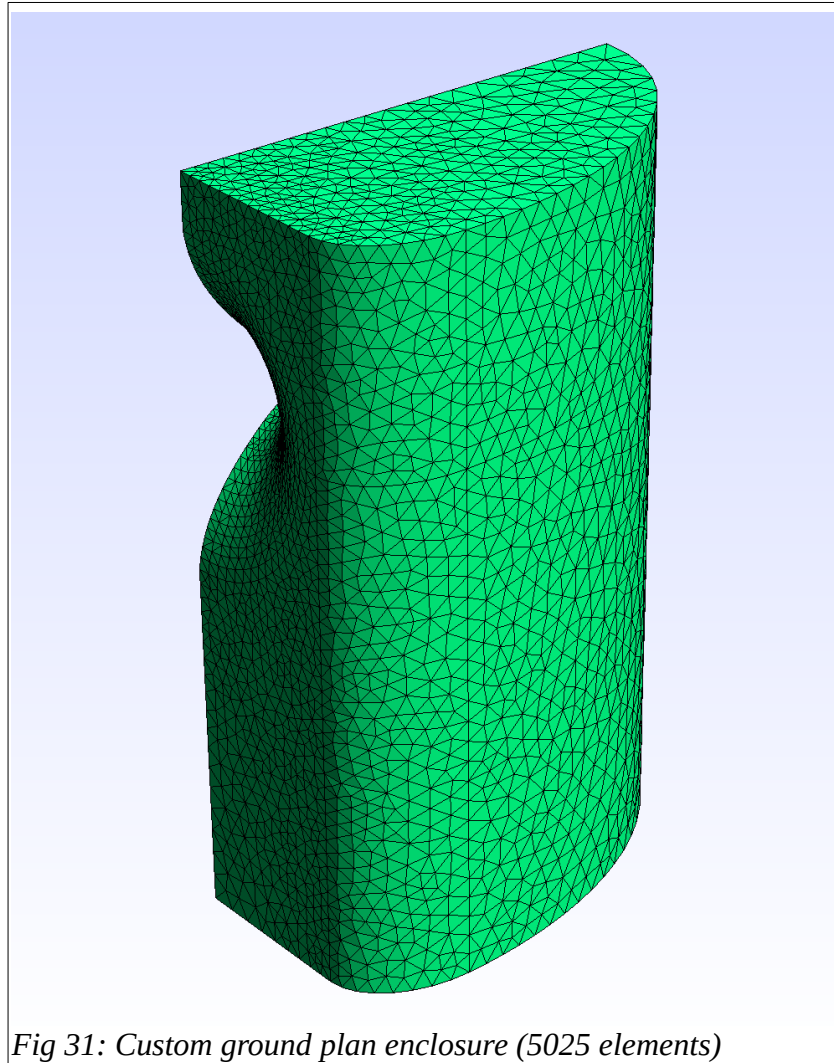


Fig 31: Custom ground plan enclosure (5025 elements)

6.13 Adding lumped element models

The Lumped Element (LE) method allows a simplified model of an electro-acoustical transducer (e.g. a compression driver) to be used together with BEM simulation of the sound waves propagation. These two parts are coupled together in ABEC to simulate the whole system. The LE modeling framework utilized is that of ABEC and the reader is referred to its documentation for a further information [2].

Adding a ready LE model to a project is extremely easy - just specify a LE script with the 'LE' item:

```
LE = generic25
```

This instructs Ath to copy the file 'generic25.txt' from the subdirectory 'bin\lib\drivers' into the target ABEC project directory and to configure the project so that it uses this script.

There are no other requirements on the LE model, except the following rules:

- The LE/BEM coupling node is tagged as **DrvGroup=1001**. This is the coupling element between the LE part and the BEM part of the model.
- The script doesn't explicitly define the input voltage, i.e. there is no 'Def_Driving' section. The input voltage is instead specified in the Ath script as 'LE.Voltage'. The section 'Def_Driving' is inserted into the LE script by Ath automatically.

Note that it may be necessary to explicitly specify the 'System' and the 'Driver' element tags as defined in the LE script, in case that these are not the default values, i.e. '**S1**' and '**D1**' respectively. This is done via the items 'LE.System' and 'LE.Driver'. These tags are then used in an observation script for the excursion and electrical impedance calculation.

Also note that once the LE script file is copied into the ABEC project directory, we are free to modify this file manually (the file will be overwritten only by a subsequent Ath execution).

For an example of a LE model script file see the included file 'generic25.txt', which corresponds to a typical 1" compression driver.

6.14 Reporting Results

The results from ABEC calculations can be presented either directly in VACS (as in Fig. 29, 30) or in a form of the following composed plot:

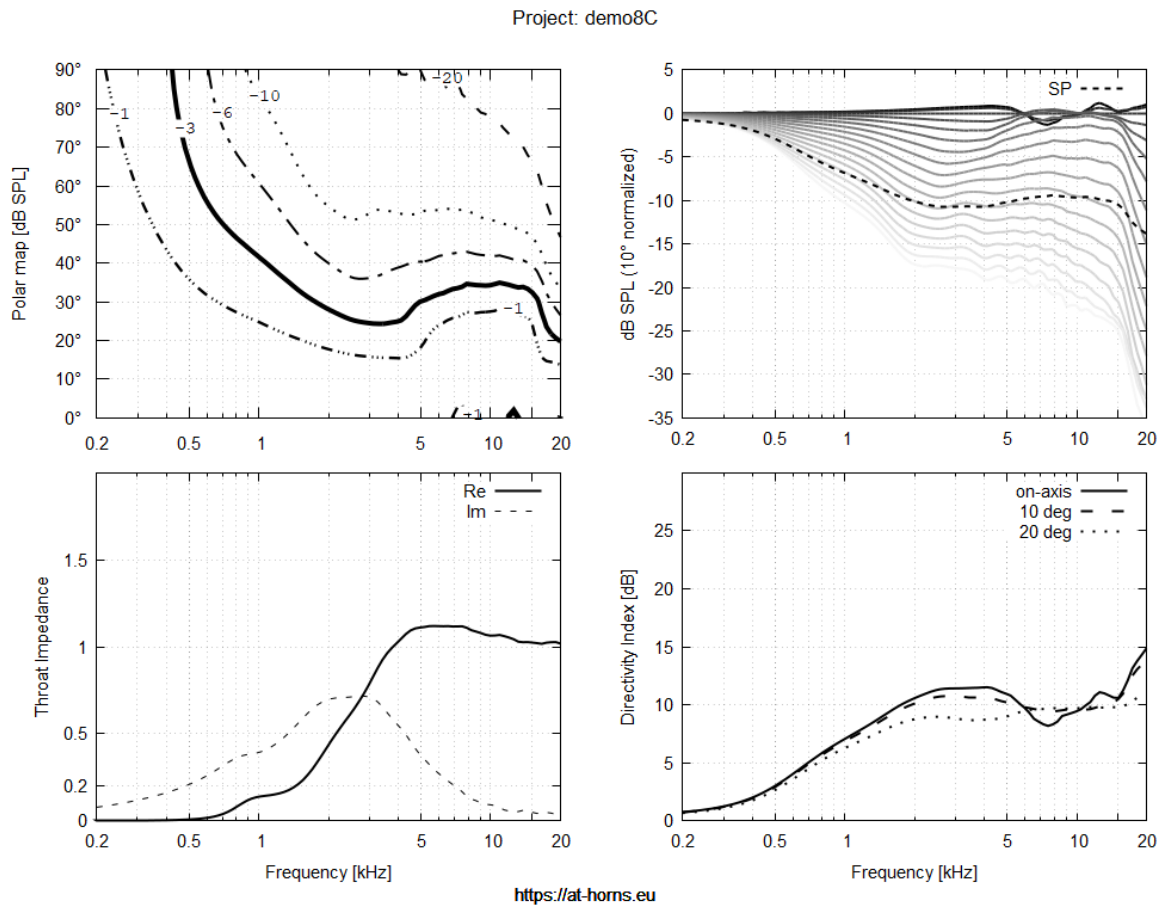


Fig 32: demo8C.cfg - Ath standard report (using gnuplot)

To make this functionality work, you need to have *gnuplot*¹⁴ installed on your PC. Gnuplot is a powerful source code driven graphing utility, available also as binaries for Windows OS. Once you have gnuplot installed, set the path to the gnuplot executable in the Ath main configuration file (see 1.1.3).

To create such report (as PNG image), follow this procedure:

- 1) Configure the 'Report' item in the script (see 4.1.6)
- 2) In ABEC, after the Spectra calculation, save the results via "Spectra - Export data as text" (Ctrl+F7).
- 3) Run Ath as before, but add a parameter "-r" at the end of the command line, e.g.:

```
D:\projects\ath>ath demos\demo8C.cfg -r
```

- 4) The resulting PNG image is stored in the ABEC project directory, in a subdirectory called "Results".

¹⁴ <http://gnuplot.info>

Note that when a LE model is used, the report uses a different/extended format by default:

ST260

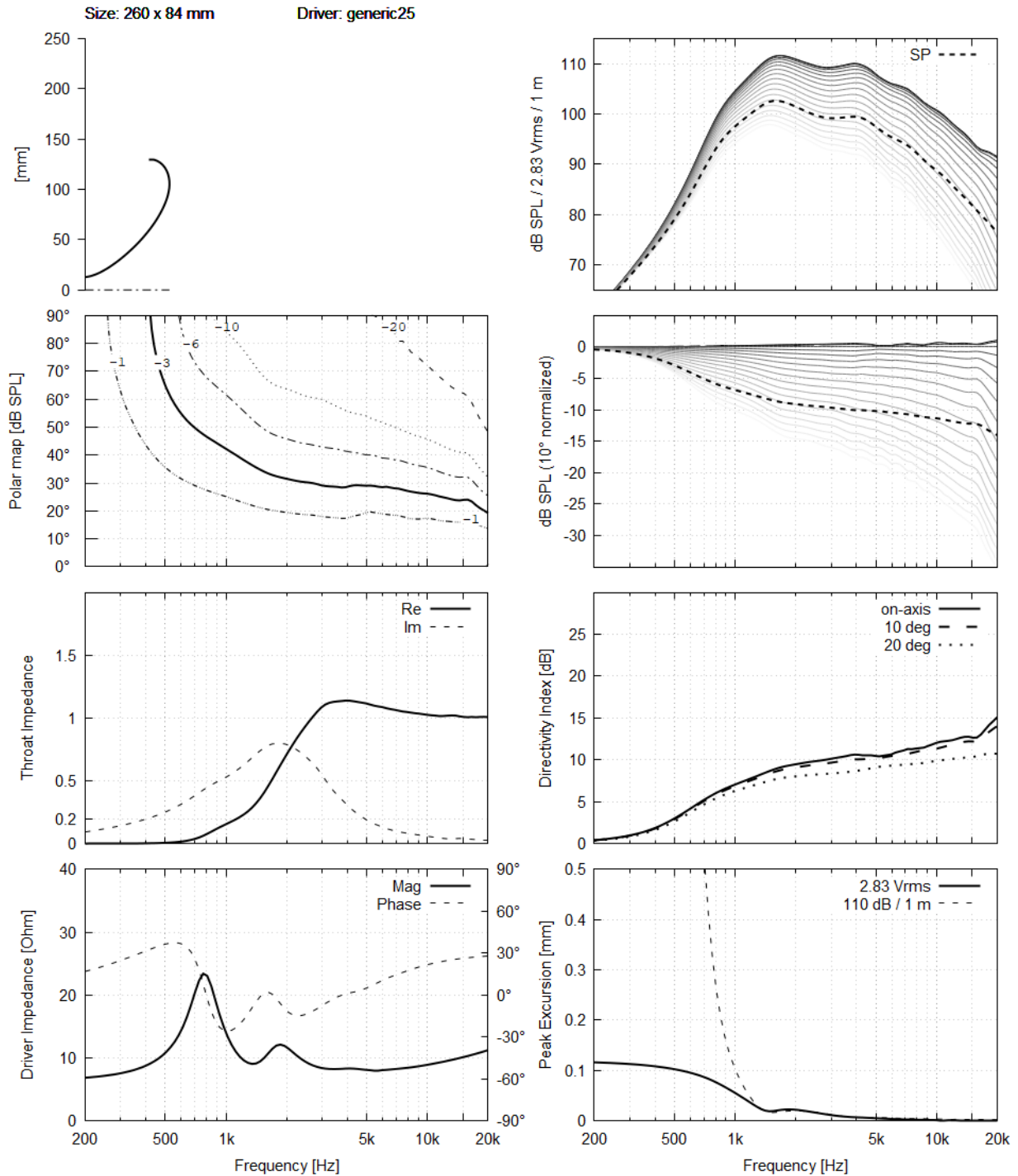


Fig 33: Report example using a transducer LE model (ST260 waveguide)

User can also provide a custom Gnuplot script and setup the Report functionality so that it uses that code. Simply create your own Gnuplot code and save the file into the 'bin\lib\scripts' subdirectory.

7 Disclaimer

The autor of this document takes no responsibility of a potential hearing loss or damage caused by any of the devices built based on instructions presented in this document or elsewhere. These waveguides can be so good that the playback levels can easily become dangerously loud. Take care and enjoy! :-)

8 References

- [1] Batík, 2021, OS-SE Acoustic Waveguide (available at [ATH website](#))
- [2] R&D Team - ABEC help file
- [3] Batík, 2021, ATH Application Note 1 - Spherical Wave Forming

Third-party libraries used

Lua scripting language <https://www.lua.org>
 <http://webserver2.tecgraf.puc-rio.br/~lhf/>

Many thanks go to the authors of the above software that made this tool possible.

Special thanks to Dr. Earl Geddes for his unrelenting support for the DIY community.

Appendix A - Math Expressions

In the definition file, for all the items of "ex" value type a free-form mathematical expression can be specified as their value. The following parameters and functions can be used in the expressions.

Available parameters:

Parameter	Description
p	The angle φ around the waveguide axis in the interval $<0, 2\pi$).

Available functions (as used in C99 standard):

acos	cos	floor	log	rad
acosh	cosh	fma	log10	remainder
asin	deg	fmax	log1p	round
asinh	erf	fmin	log2	scalbn
atan	erfc	fmod	logb	sin
atan2	exp	frexp	modf	sinh
atanh	exp2	gamma	nearbyint	sqrt
cbrt	expm1	hypot	nextafter	tan
ceil	fabs	ldexp	pi	tanh
copysign	fdim	lgamma	pow	trunc

Appendix B – Source Definition Script

Introduced in Ath 4.6, an arbitrary source shape can be defined for the BEM simulation. This is mainly useful when modeling direct radiating tweeters but it can also be used for a simulation of a non-flat compression driver output wavefront (either in shape or amplitude), or even of a complete compression driver phase plug. The only limitation is that the whole additional structure must be axi-symmetric, to match the circular throat of the waveguide.

Geometric entites

Currently, three basic geometrical entites are available: points, lines and circular arcs. All points must be defined first, before the higher elements which are then defined by means of these points. Each point is assigned an identifier by which it is referenced further in the script. There is also a special object type called *control point*, which is a virtual helper point to define geometry of other elements (e.g. centers of arcs).

Disjunctive parts

The whole geometric structure can be composed of several disjunctive parts. Each part is defined by a sorted list of joined elements. Each part except the first one must form a closed-loop curve, i.e. with the last point identical to the first one. The individual disjunctive parts must be separated by an empty line in the script.

Connecting source to the waveguide

The first (and maybe the only) defined part always defines the surface enclosing the throat. The order of its elements is important: the contour must start at a point on the axis and always end at a special point **WG0** where the contour meets the first point of the waveguide profile (see below).

Syntax Summary

```
point <id> <z> <r> <mesh_size>
```

<id>	Unique alpha-numerical identifier of the point.
<z>	z-coordinate, i.e the distance from the throat [mm].
<r>	r-coordinate, i.e. the distance from the horn's axis [mm].
<mesh_size>	Constraint on the mesh element size [mm].

```
cpoint <id> <z> <r>
```

Control point. The meaning of the parameters is the same as for a point. Note that the parameter <mesh_size> is missing - that's because control point doesn't define any physical boundary.

```
line <point_1> <point_2> <weight>
```

<point_1>	id of the starting point.
<point_2>	id of the ending point.
<weight>	Weight of the surface's acoustic radiation (1 = full, 0 = not moving).


```
arc    <point_1> <center> <point_2> <weight>
```

<point_1> id of the starting point.
 <center> id of the center point OR the radius value [mm]
 <point> id of the ending point.

IMPORTANT: The arc angle must be strictly less than π (180°).

Special points

WG0

Point identified as WG0 is automatically defined at coordinates `[0, Throat.Diameter/2]` and always represents the first point of the actual waveguide profile. This point has a special internal meaning and **must be used** to connect the source contour to the waveguide - it is not equivalent to a user-defined point at the same coordinates.

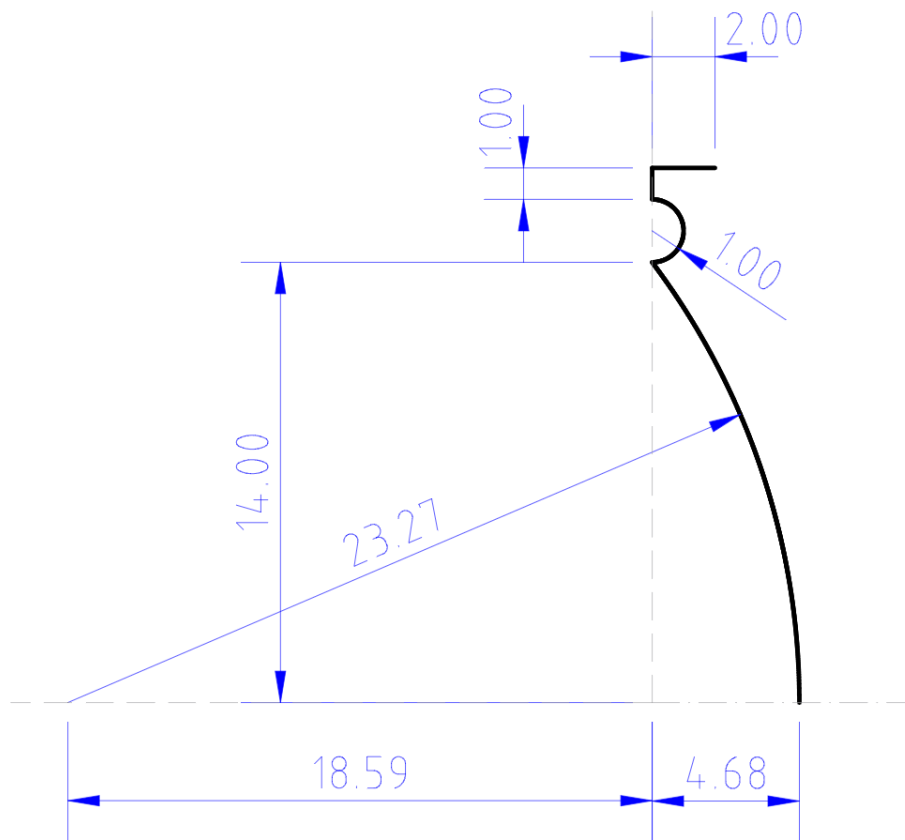
Special keywords

`zoff` <z_offset>

When used, this parameter shifts the whole geometry in the z-axis direction [mm], i.e. the value *z_offset* is added to all z-coordinates of the points. To take effect, it must be set before the points it should affect are defined (it is possible to change this value any number of times in the script if necessary). Note that the point WG0, i.e. the point where the contour meets the waveguide, is always at $z=0$. It is possible to use a different coordinates for the contour description but then the drawing must be shifted to match WG0. This is what the command `zoff` is for.

Example

Let's say we want to define a dome tweeter as depicted on the following drawing.

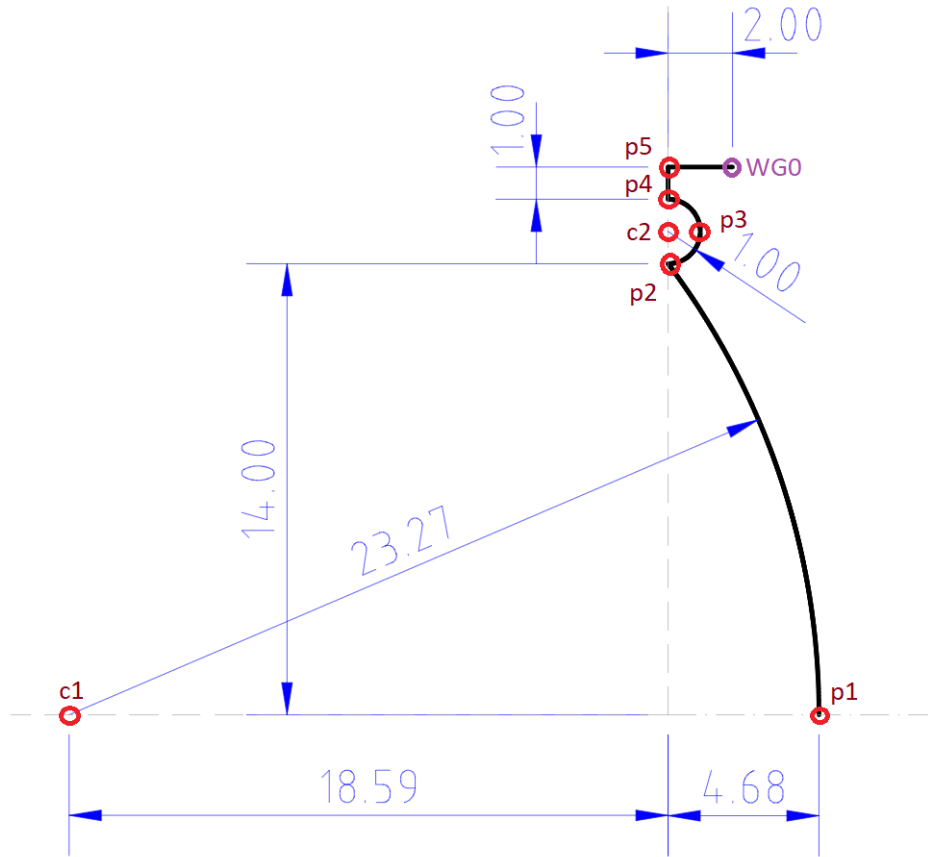


There are two circular arcs as moving parts (the diaphragm and the surround) and two line segments as parts of a mounting plate.

Note that the axial (z) coordinate is arbitrary, with $z = 0$ at the edge of the diaphragm. This is fine, we only have to remember to finally shift the whole geometry with `zoff` (more on that later) to properly connect to the rest of a waveguide profile.

The radial coordinate axis (r) must always match the axis of the device.

First, we need to identify all the significant points that will allow us to construct the whole contour - see the drawing below. Remember that the arc angles must be strictly less than π (180°) - that's one of the reasons why the surround is divided into two 90° arcs at an additional point p3. The other reason is that we will want to set the acoustic radiation weights for the two arcs differently (as the surround contributes less towards its outer edge):



Based on the above drawing we now define physical points **p1** – **p5** and two control points **c1**, **c2** for the centers of the arcs:

```
point p1 4.68 0 2
point p2 0 14 0.5
point p3 1 15 0.5
point p4 0 16 0.5
point p5 0 17 1
cpoint c1 -18.59 0
cpoint c2 0 15
```

Note that we also set the mesh size constraints at the points to capture the shape of the modeled object in just enough detail. This is important for the subsequent meshing of the 3D surface. In axisymmetric mode these values are ignored.

After we have defined the points, we can continue with a list of higher-order elements. In our example this will be the first and the only segment of our whole contour, as there are no other disjunctive parts. Note that the order of the list is important here - the segments must start on axis and terminate at the point WG0. Make sure that there are no empty lines that would separate the contour into separate parts:

```
arc    p1 c1 p2 1.0
arc    p2 c2 p3 0.75    ; arc    p2 1.0 p3 0.75 ← alternative definition without c2
arc    p3 c2 p4 0.25    ; arc    p3 1.0 p4 0.25 ← alternative definition without c2
line   p4 p5 0
line   p5 WG0 0
```

Several things to notice here:

- The last element of the contour is the line **p5 – WG0**. The contour must always be connected to the horn by using the point WG0, which is automatically defined as the first point of the actual horn profile (in this example the throat diameter is set to 34 mm, so this line is parallel to z axis as the result). This way it is ensured that there won't be any holes in the meshed surface and that the mesh elements will form one cohesive structure.
- The last number in each arc and line definition is the weight of its acoustic radiation. By setting this value to zero we say that this is a non-moving part (the same as the horn wall).

The last important thing to take care about is the absolute coordinates of the points. That's because the horn always starts at $z = 0$. So far we have used the coordinates directly from the drawing – that would be wrong! If you look carefully at the drawing you'll see that we need to shift the whole source contour by -2 mm in the z-direction. We can do this easily by using the `zoff` command.

Now we can summarize the complete script file:

```
zoff   -2

point  p1 4.68 0 2
point  p2 0 14 0.5
point  p3 1 15 0.5
point  p4 0 16 0.5
point  p5 0 17 1

cpoint c1 -18.59 0
cpoint c2 0 15

arc    p1 c1 p2 1.0
arc    p2 c2 p3 0.75
arc    p3 c2 p4 0.25
line   p4 p5 0
line   p5 WG0 0
```

This script can now be either saved in a separate text file or included directly in the definition file (see the example in 6.5).

The finished mesh based on the previous source definition (driving elements in red):

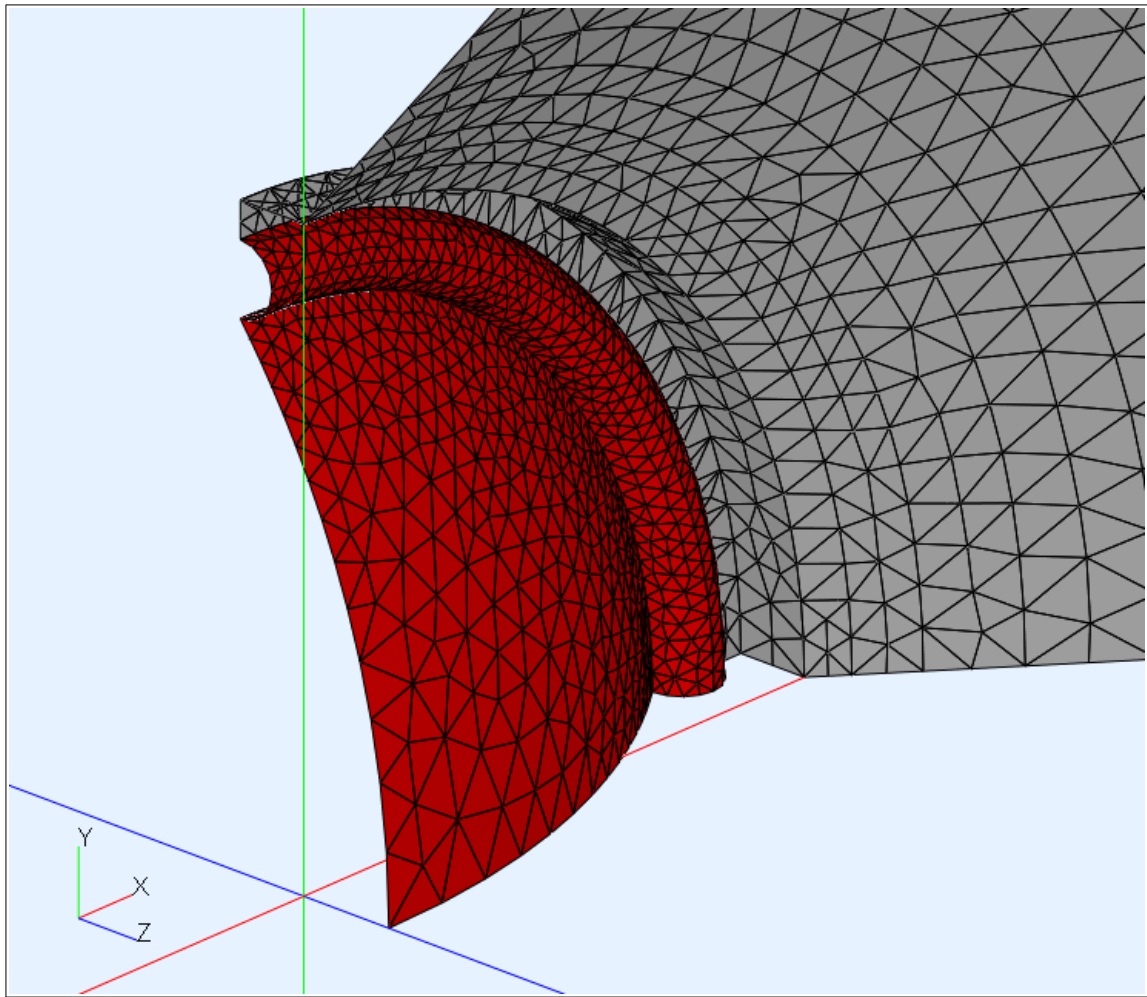


Fig 34: Dome tweeter source (ABEC display)

Appendix C – User-defined enclosure plan

Enclosure ground plan is a named set of graphical entities (points, lines, etc.) forming a description of the shape of an enclosure wall. Prepared ground plan can be then referenced in the **Mesh.Enclosure** section.

Only enclosures with flat front baffle are possible at the moment - the front baffle must also be large enough to accomodate the whole waveguide defined.

Constructing the ground plan

- The plan is defined in a local 2D coordinate system [x, y], with [0, 0] being the center of the front baffle.
- First, a set of points must be defined, two of those having a special meaning: **P0** and **PB**

The point **P0** ("P-zero") must be located somewhere on the x axis ($y = 0$), the point **PB** somewhere on the y axis ($x = 0$). The wall shape then consist of an consecutive list of 2D entites (lines, arcs, etc.) connecting the point **P0** with the point **PB** (in this order) by means of any number of optional points and curves in between. For an example of a complete plan see paragraph 6.12.2.

Syntax of graphical elements

`point <id> <x> <y> <mesh_size>`

Creates a physical point to be used further in the script. The value `mesh_size` affects the mesh density at this point.

`cpoint <id> <x> <y>`

Creates a virtual helper point to be used further in the script (e.g. for an arc center, etc.).

`line <id1> <id2>`

Creates a straight line connecting the points <id1> and <id2>.

`arc <id1> <id2> <id3>`

Creates a circular arc. Points <id1>...<id3> are the starting point, the center point and the end point. The arc angle must be strictly less then π (180°).

`ellipse <id1> <id2> <id3> <id4>`

Creates an elliptical arc. Points <id1>...<id4> are the stariting point, the center point, a point anywhere on the major axis and the end point.

`bezier <id1> ... <idN>`

Creates a Bezier curve with the given control points.