The Overture Hyperbolic Grid Generator
User Guide, Version 1.0

William D. Henshaw [1]
Centre for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, 94551
henshaw@llnl.gov
http://www.llnl.gov/casc/people/henshaw
http://www.llnl.gov/casc/Overture

November 20, 2012

UCRL-MA-134240

Abstract

This document describes the `HyperbolicMapping` class for generating surface and volume grids using a marching algorithm. Surface grids can be grown over any other Mapping that defines a surface in three-dimensions including a `CompositeSurface` which represents a surface as a collection of multiple sub-surfaces. Volume grids can be generated in two or three space dimensions. A variety of boundary conditions are available.

Contents

1 HyperbolicMapping **3**
1.1 Hyperbolic Marching Equations 3
1.2 Algorithm 7

2 Steger-Chan Hyperbolic Marching **12**

3 The Osher-Sethian Level Set (Hamilton-Jacobi) Marching Equations **14**

4 Distributing points by equidistribution of a weight function **14**

5 Boundary conditions **15**
5.1 Boundaries, Ghost Points and the BoundaryOffset 16
5.2 Normal Blending 16
5.3 Projection of boundary points on surfaces 16
5.4 Heuristic Comments on Hyperbolic Parameters 17
5.5 Hints to making a grid 17

6 Creating a surface grid on another Mapping or CompositeSurface **17**
6.1 Use of the 'boundary offset' option when generating grids on CAD. 18

[1] This work was performed under the auspices of the U.S. Department of Energy (DOE) by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and by DOE contracts from the ASCR Applied Math Program.

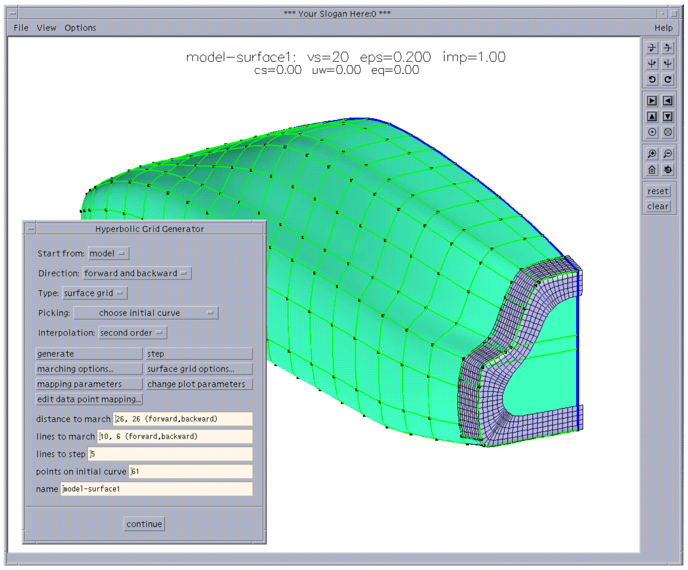
7 Examples **19**
7.1 Bump . . . 20
7.2 Flat Plate . . . 21
7.3 Mast for a sail . . . 22
7.4 Airfoil grids . . . 23
7.5 Surface grid generation on a CompositeSurface for a soup can . . . 24
7.6 Surface and Volume Grid Generation on a CAD model for an Automobile. . . . 25
7.7 Grid Generation for a Truck. . . . 27
7.7.1 Front . . . 27
7.7.2 Wheel . . . 29
7.7.3 Cab tender . . . 30

8 Class member functions **31**
8.1 Constructor . . . 31
8.2 Constructor . . . 31
8.3 Constructor . . . 31
8.4 isDefined . . . 31
8.5 printStatistics . . . 31
8.6 setBoundaryConditionMapping . . . 31
8.7 setSurface . . . 32
8.8 setIsSurfaceGrid . . . 32
8.9 setStartingCurve . . . 32
8.10 saveReferenceSurfaceWhenPut . . . 32
8.11 setup . . . 33
8.12 setParameters . . . 33
8.13 setPlotOption . . . 34
8.14 smooth . . . 34
8.15 inspectInitialSurface . . . 34
8.16 generate . . . 34

1 HyperbolicMapping

The HyperbolicMapping can be used to generate surface and volume grids by marching along or from a given reference curve or surface.

See the Mapping monster manual [3] for a information on many other Mappings as well as a description of Mappings in general.



model−surface1: vs=20 eps=0.200 imp=1.00
cs=0.00 uw=0.00 eq=0.00

Start from: model
Direction: forward and backward
Type: surface grid
Picking: choose initial curve
Interpolation: second order
distance to march 26, 26 (forward,backward)
lines to march 10, 6 (forward,backward)
lines to step 5
points on initial curve 61
name model-surface1

Figure 1: Snapshot of the hyperbolic grid generator. A surface grid is grown on a CAD model for an automobile. A starting curve is chosen and the grid is grown in both directions over the surface.

1.1 Hyperbolic Marching Equations

Let (r, s, t) denote the parameter space (computational) coordinates. Instead of taking parameter space to be the unit cube we instead take the grid spacing in parameter space to be 1, $\Delta r = \Delta s = \Delta t = 1$.

Given a surface $\mathbf{x}_0(r, s) = \mathbf{x}(r, s, t = 0)$ we wish to generate a volume grid, $\mathbf{x}(r, s, t)$, so that the grid lines in the t-direction are nearly orthogonal to the grid lines in the two other directions. We call $\mathbf{x}(r, s, t = 0)$ the initial front and think of the variable t as a time like variable. If we have generated the grid to "time" $t = t_0$ we call $\mathbf{x}(r, s, t_0)$ the current front.

The basic marching equations to determine $\mathbf{x}(r, s, t)$ given $\mathbf{x}(r, s, 0)$ are defined by the hyperbolic PDE

$$\begin{aligned} \mathbf{x}_t &= S(r,s,t)\ \mathbf{n}(r,s,t) && \\ \mathbf{x}(r,s,0) &= \mathbf{x}_0(r,s) && \text{initial conditions} \\ B(\mathbf{x}(r,s,t)) &= 0 && \text{boundary conditions} \end{aligned}$$

where

$$\begin{aligned} \mathbf{n}(r,s,t) &= \frac{\mathbf{x}_r \times \mathbf{x}_s}{\|\mathbf{x}_r \times \mathbf{x}_s\|} && \text{normal to the front} \\ S(r,s,t) & && \text{scalar speed function} \end{aligned}$$

and the norm $\|\cdot\|$ is defined by

$$\|\mathbf{f}\|^2 \equiv \mathbf{f} \cdot \mathbf{f}.$$

These equations march the grid in the direction locally orthogonal to the current front. The speed function $S(r,s,t)$ determines how fast the front propagates; it can depend on local properties of the front. Smoothing is also added to the equations so we actually solve a parabolic equation of the form

$$\mathbf{x}_t = S(r,s,t)\mathbf{n} + \epsilon(r,s,t)(\mathbf{x}_{rr} + \mathbf{x}_{ss})$$

To ensure that the front always propgates in the forward direction we require $\mathbf{n}\cdot\mathbf{x}_t > 0$ or equivalently

$$\mathbf{n}\cdot\Big(S(r,s,t)\mathbf{n} + \epsilon(\mathbf{x}_{rr} + \mathbf{x}_{ss})\Big) > 0$$

In addition to smoothing the grid in the (r,s) directions, the the parabolic smoothing term will tend to slow the front where the curvature is negative (i.e. $\mathbf{n}\cdot(\mathbf{x}_{rr}+\mathbf{x}_{ss}) < 0$) and speed up the front where the curvature is positive. Note that choosing too large a value for ϵ could cause the front to propogate in the wrong direction resulting in negative cell-volumes. The speed function $S(r,s,t)$ and dissipation coefficient ϵ should be specified so that we get a "nice grid". A nice grid should not have any grid lines that cross, it should be reasonably orthogonal and reasonably smooth.

The marching equations can be solved with an implicit time marching algorithm. To do this we first linearize the equations about the current front, $\mathbf{x}(r,s,t^n)$, to obtain an equation of the form

$$\mathbf{x}_t = A(r,s,t)\mathbf{x}_r + B(r,s,t)\mathbf{x}_s + \epsilon(\mathbf{x}_{rr} + \mathbf{x}_{ss}) + \mathbf{f}(r,s,t)$$

This equation can be solved using a $\theta-$scheme for $\mathbf{x}(r,s,t^n) \approx \mathbf{x}^n$,

$$\begin{aligned}
\frac{\mathbf{x}^{n+1}-\mathbf{x}^n}{\Delta t} &= \theta\left[A(r,s,t^{n+1})\mathbf{x}_r^{n+1} + B(r,s,t^{n+1})\mathbf{x}_s^{n+1} + \epsilon(\mathbf{x}_{rr}^{n+1} + \mathbf{x}_{ss}^{n+1})\right] \\
&\quad + (1-\theta)\left[A(r,s,t^n)\mathbf{x}_r^n + B(r,s,t^n)\mathbf{x}_s^n + \epsilon(\mathbf{x}_{rr}^n + \mathbf{x}_{ss}^n)\right] + \mathbf{f}^n \\
\mathbf{f}^n &= S^n\mathbf{n}??
\end{aligned}$$

where $\theta = 1$ corresponds to backward-Euler. For efficiency we use an approximate factorization to reduce the implicit matrix solve to a sequence of block-tridiagonal solves.

We now consider choices for the speed function, $S(r,s,t)$. Following the approach of Steger-Chan we define the speed function based on the local cell-areas of the front,

$$\begin{aligned}
S_A(r,s,t) &= d_0(t)\ \Delta t\ \overline{\Delta a}/\Delta a \\
d_0(t)\Delta t &= \text{distance to march in a time step } \Delta t, \text{ (approximate average value)} \\
\Delta a(r,s) &= \|\mathbf{x}_r \times \mathbf{x}_s\| \qquad \text{proportional to the local area of the front} \\
\overline{\Delta a}(r,s) &= \text{Locally averaged value of } \Delta a(r,s)
\end{aligned}$$

The speed function is proportional to the local cell area divided by a locally average cell area. The averaged cell area, $\overline{\Delta a}(r,s)$, is computed by smoothing the cell area $\Delta a(r,s)$ using a simple Jacobi type interation. As a result of using this speed function the grid will tend to grow faster where the area of cells on the front are smaller and slower where the grid cells are larger compared to the local average. Asymptotically a front will tend toward a curve where the surface areas are equal. For example, a front may tend to a sphere or a plane in 3D, depending on the boundary conditions for the front. Steger and Chan also use a sophisticated dissipation term as described in section 2.

Following the approach of Sethian we could also choose the speed function proportional the the local curvature,

$$\begin{aligned}
S_c(r,s,t) &= (1 - \epsilon_c\kappa(r,s,t)) \\
\kappa(r,s,t) &= \text{local curvature}
\end{aligned}$$

If $\epsilon_c > 0$ then we are guaranteed that grid lines will not locally cross, although the front could propogate in the wrong direction id S_c becomes negative. Here the curvature κ causes the grid to move faster where the curvature is negative and slower where it is positive. The hyperbolic grid generator allows one to use a combination of the area based speed function and the curvature based spped function. The comnbined speed function is taken as the product of S_A and S_c,

$$S(r,s,t) = d_0(t)\ \Delta t\ \overline{\Delta a}/\Delta a\ \ (1-\epsilon_c\kappa(r,s,t))$$

For 2D volume grids or 3D surface grids there is also an option to blend the solution obtained from the above equation with a distribution of points based on equidistributing a weight function based on the arclength and curvature. If we equidistrubte the arclength, for example, we will obtain a distribution of points, $\mathbf{x}^E$, that are equally spaced in arclength. A new front is defined by averaging the equidistributed points with the points determined by using the speed function.

$$\tilde{\mathbf{x}}(r,s,t) = (1-\omega^E)\mathbf{x}(r,s,t) + \omega^E\mathbf{x}^E(\mathbf{x}(r,s,t))$$

The equidistributed points are determined by a weight function

$$w(r) = \alpha^A\|\mathbf{x}_r\|/\|\mathbf{x}_r\|_\infty + \alpha^C\|\mathbf{x}_{rr}\|/\|\mathbf{x}_{rr}\|_\infty$$

where $\|\mathbf{f}\|_\infty = \max_r \|\mathbf{f}(r)\|$. The weight function is equidsitributed over the unit interval to determine positions $r_i^E \in [0,1]$, $i=1,2,\ldots,N$, $r_{i+1}^E > r_i^E$, such that

$$\int_{r_i^E}^{r_{i+1}^E} w dr = \frac{1}{N}\int_0^1 w dr$$

This last equation expresses the condition that the weight function is equidistributed. The new grid points positions are computed by evaluating the curve, $\mathbf{c}(r)$, defining the current front at the new parameter positions r_i^E,

$$\mathbf{x}^E := \mathbf{c}(\mathbf{r}^E)\ :\ \text{re-evaluate the curve at the new positions}$$

Following Steger, the hyperbolic marching equations can be cast in an alternative form

$$\mathbf{x}_r\cdot\mathbf{x}_t = 0 \tag{1}$$
$$\mathbf{x}_s\cdot\mathbf{x}_t = 0 \tag{2}$$
$$\mathbf{x}_r\times\mathbf{x}_s\cdot\mathbf{x}_t = \Delta V(r,s,t) \tag{3}$$

The first two equations specify the orthogonality conditions while the last equation specifies the local volume of the cell, ΔV. We can solve these equations for $\mathbf{x}_t$ and we see that the solution is defined by locally marching along rays that move in the normal direction:

$$\begin{aligned}
\mathbf{x}_t(r,s,t) &= \frac{\mathbf{x}_r(r,s,t)\times\mathbf{x}_s(r,s,t)}{\|\mathbf{x}_r(r,s,t)\times\mathbf{x}_s(r,s,t)\|^2}\Delta V \\
&= \frac{\Delta V}{\|\mathbf{x}_r(r,s,t)\times\mathbf{x}_s(r,s,t)\|}\ \ \mathbf{n}(r,s,t) \\
\mathbf{n}(r,s,t) &= \frac{\mathbf{x}_r(r,s,t)\times\mathbf{x}_s(r,s,t)}{\|\mathbf{x}_r(r,s,t)\times\mathbf{x}_s(r,s,t)\|}
\end{aligned}$$

and thus we can identify the speed function

$$S(r,s,t) = \frac{\Delta V}{\|\mathbf{x}_r(r,s,t)\times\mathbf{x}_s(r,s,t)\|}$$

If we choose $\Delta V(r,s,t) = c\|\mathbf{x}_r(r,s,t)\times\mathbf{x}_s(r,s,t)\|$, for a constant c, then the grid lines in the marching direction will just be straight lines parallel to the normal of the original surface. Of course the grid generated by this system may develop singularities, if any part of the original surface is concave. To avoid this problem extra smoothing is added.

If we choose $\Delta V(r,s,t) = c$ then the grid spacing in the normal direction will be inversely proportional to the local surface cell area. Thus the grid will grow fastest where the cells are small.

The basic marching distance depends on the type of stretching, the total distance to march D, and the number of steps to march, N:

$$\mathbf{d}_{\mathbf{i}}^n = \frac{D}{N} \qquad \text{constant spacing}$$
$$\mathbf{d}_{\mathbf{i}}^n = D\ \alpha^n \frac{\alpha - 1}{\alpha^{N+1} - 1} \qquad \text{geometric stretching}$$

The volume element appearing in the marching step is a product of the marching distance times the ratio of the averaged area element $\overline{\Delta a_{\mathbf{i}}}$ to the area element $\Delta a_{\mathbf{i}}$

$$\Delta V_{\mathbf{i}} = \mathbf{d}_{\mathbf{i}}^n \frac{\Delta a_{\mathbf{i}}}{\overline{\Delta a_{\mathbf{i}}}} \qquad : \text{volume element}$$

Parameters appearing in the code are

number of volume smooths : number of times we smooth $\Delta a_{\mathbf{i}}$ to obtain $\overline{\Delta a_{\mathbf{i}}}$.

uniform dissipation coefficient : ϵ, coefficient of the parabolic terms.

implicit coefficient : θ coefficient of implicit time stepping.

equidistribution : weight factor for the equidistributed approach.

arclength weight : α_A weight for arclength in equidistribution weight function

curvature weight : α_C weight for curvature in equidistribution weight function

curvature speed : ϵ_c weight factor for the curvature dependent speed function.

1.2 Algorithm

Here is a summary of the algorithm

Notation:

n_d : domain dimension, $n_d \equiv 2$ for 2D volume grids or 3D surface grids, $n_d \equiv 3$ for 3D volume grids
$\mathbf{C}$: starting surface (or starting curve)
$\mathbf{R}$: reference surface for surface grid generation
$\Delta a_{\mathbf{i}}$: local surface area (arclength in 2D)
$\overline{\Delta a_{\mathbf{i}}}$: smoothed surface area (smoothed arclength in 2D)
$\mathbf{n_i}$: normal
D : marching distance
N : number of steps to march
$\mathbf{i}$: multi-index $\mathbf{i} = (i_1, i_2)$ for 3D volume grids, or $\mathbf{i} = (i_1)$ for 2D grids or 3D surface grids.

Algorithm 1 Hyperbolic grid generator: Generate a volume grid in 2D or 3D or a surface grid in 3D

```
1:  function generate( )
2:      $\mathbf{x}_{\mathbf{i}}^0 := \mathbf{C}(\mathbf{r_i})$                ▷ evaluate the starting surface (starting curve if $n_d \equiv 2$)
3:      if this is a surface-grid then
4:          projectInitialCurveOntoReferenceSurface($\mathbf{x}^0, \mathbf{n_i}, \mathbf{xt}; \mathbf{R}$)
5:      end if
6:
7:      hyperbolic marching steps:
8:      for $n = 0, 1, ..., N$ do
9:          getNormalAndSurfaceArea($\mathbf{x}^n, \mathbf{n}, \Delta a, \overline{\Delta a}, \mathbf{xr}, \mathbf{xs}$)
10:         getDistanceToStep($\mathbf{d_i}$)  : get marching distance
11:         getCurvatureDependentSpeed($\mathbf{d_i}$)  : adjust marching distance for curvature
12:         if $n \equiv 0$ and this is not a surface grid then
13:             $\mathbf{xt_i} := \mathbf{d_i}\ (\overline{\Delta a_{\mathbf{i}}}/\Delta a_{\mathbf{i}})\ \mathbf{n_i}$          ▷ linearize about this value of $\mathbf{x}_t$
14:         end if
15:         Form the right-hand-side:
16:         $\mathbf{r_i} := \mathbf{d_i}\ (\overline{\Delta a_{\mathbf{i}}}/\Delta a_{\mathbf{i}})\ \mathbf{n_i} + \epsilon_e \Delta_{+r}\Delta_{-r}\mathbf{x}_{\mathbf{i}}^n + \epsilon_e \Delta_{+s}\Delta_{-s}\mathbf{x}_{\mathbf{i}}^n$
17:         $A := A(\mathbf{xr}, \mathbf{xs}, \mathbf{xt})$          ▷ linearized coefficient matrix for implicit time stepping
18:         $B := B(\mathbf{xr}, \mathbf{xs}, \mathbf{xt})$          ▷ form implicit time stepping matrices:
19:         $M_1 = I + A\Delta_{0r} - \epsilon_i \Delta_{+r}\Delta_{-r}$
20:         $M_2 = I + B\Delta_{0s} - \epsilon_i \Delta_{+s}\Delta_{-s}$          ▷ $M_2 = I$ for $n_d \equiv 2$
21:         $\mathbf{v} := M_2^{-1} M_1^{-1} \mathbf{r}$          ▷ solve for the correction
22:         $\mathbf{x}_{\mathbf{i}}^{n+1} := \mathbf{x}_{\mathbf{i}}^n + \mathbf{v_i}$
23:
24:         Next apply BC's and optionally adjust for equidistribution.
25:         For surface grids project $\mathbf{x}^{n+1}$ onto the reference surface:
26:         applyBoundaryConditions($\mathbf{x}^{n+1}$)
27:
28:         $\mathbf{xt_i} := \mathbf{x}_{\mathbf{i}}^{n+1} - \mathbf{x}_{\mathbf{i}}^n$          ▷ linearize about this value of $\mathbf{x}_t$
29:     end for
30: end function
```

Algorithm 2 Project the initial curve onto the reference surface and determine the initial normal

```
1: function projectInitialCurveOntoReferenceSurface($\mathbf{x}^0, \mathbf{n_i}, \mathbf{xt}; \mathbf{R}$)
2:     Project initial curve onto the reference surface, compute normal
3:     project($\mathbf{x}^0, \mathbf{n_i}; \mathbf{R}$)
4:     In case the initial curve lies on a edge in the reference surface where
5:     the normal is ill-defined, take a small initial step and then recompute the normal.
6:     getNormalAndSurfaceArea($\mathbf{x}^0, \mathbf{n}, \Delta a, \overline{\Delta a}$)
7:     getDistanceToStep($\mathbf{d_i}$)                                  ▷ get marching distance
8:     $\delta = .1$ : take this fraction of a step
9:     $\mathbf{x}^1_\mathbf{i} := \mathbf{x}^0_\mathbf{i} + \delta\ \mathbf{d_i}\ (\overline{\Delta a}_\mathbf{i}/\overline{\Delta a}_\mathbf{i})\ \mathbf{n}^0_\mathbf{i}$                ▷ take a small step
10:    applyBoundaryConditions($\mathbf{x}^1, \mathbf{n}$)        ▷ this will also project onto the reference surface
11:    $\mathbf{xt_i} := (\mathbf{x}^1_\mathbf{i} - \mathbf{x}^0_\mathbf{i})/\delta$                  ▷ linearize about this value of $\mathbf{x}_t$
12: end function
```

Algorithm 3 Determine the normal and surface area

```
function getNormalAndSurfaceArea($\mathbf{x}^n, \mathbf{n}, \Delta a, \overline{\Delta a}, \mathbf{xr}, \mathbf{xs}$)
    Param: $\mathbf{x}^n$ : position of the front
    Param: $\Delta a_{\mathbf{i}}$ : (output) vertex centred area element
    Param: $\overline{\Delta a}_{\mathbf{i}}$ : (output) vertex centred averaged area element
    Param: $\mathbf{xr}$ : (output)
    Param: $\mathbf{xs}$ : (input/output) : for a surface grid $\mathbf{xs}$ defined on input as the normal to the surface.

    $\hat{\mathbf{n}}_{\mathbf{i}+\frac{1}{2}} := (\mathbf{x}_{i+1} - \mathbf{x}_i) \times (\mathbf{x}_{j+1} - \mathbf{x}_j)$    ▷ unnormalized face centred normal
    $\mathbf{n}_{\mathbf{i}+\frac{1}{2}} := \hat{\mathbf{n}}_{\mathbf{i}+\frac{1}{2}} / \|\hat{\mathbf{n}}_{\mathbf{i}+\frac{1}{2}}\|$    ▷ face centred normal
    $\hat{\mathbf{n}}_{\mathbf{i}} := \frac{1}{4}(\mathbf{n}_{i_1-\frac{1}{2},i_2-\frac{1}{2}} + \mathbf{n}_{i_1+\frac{1}{2},i_2-\frac{1}{2}} + \mathbf{n}_{i_1-\frac{1}{2},i_2+\frac{1}{2}} + \mathbf{n}_{i_1+\frac{1}{2},i_2+\frac{1}{2}})$
    $\mathbf{n}_{\mathbf{i}} := \hat{\mathbf{n}}_{\mathbf{i}} / \|\hat{\mathbf{n}}_{\mathbf{i}}\|$    ▷ vertex centred normal
    $\Delta a_{\mathbf{i}+\frac{1}{2}} := \|\hat{\mathbf{n}}_{\mathbf{i}+\frac{1}{2}}\|$    ▷ cell centred area element
    vertex centred area element:
    $\Delta a_{ij} := \frac{1}{4}(\Delta a_{i-\frac{1}{2},i_2-\frac{1}{2}} + \Delta a_{i+\frac{1}{2},i_2-\frac{1}{2}} + \Delta a_{i-\frac{1}{2},i_2+\frac{1}{2}} + \Delta a_{i+\frac{1}{2},i_2+\frac{1}{2}})$

    apply special boundary conditions to normals
    if trailing edge boundary condition then
        set normal to the trailing edge direction
    else if boundary matches to an adjacent surface then
        project the normal at the boundary to be tangent to the boundary condition surface
        $\mathbf{n}_{\mathbf{i}}^B$ : normal to the boundary condition surface
        $\mathbf{n}_{\mathbf{i}} := \mathbf{n}_{\mathbf{i}} - (\mathbf{n}_{\mathbf{i}} \cdot \mathbf{n}_{\mathbf{i}}^B)\mathbf{n}_{\mathbf{i}}^B$    ▷ for boundary points
        $\mathbf{n}_{\mathbf{i}} := \mathbf{n}_{\mathbf{i}} / \|\mathbf{n}_{\mathbf{i}}\|$
    else if boundaryCondition=fixXfloatYZ or boundaryCondition=fixYfloatXZ etc. then
        adjust normal to be consistent with the boundary condition
    end if

    blend nearby normals with the boundary normal
    for $m = 1, 2, \ldots,$ numberOfLinesToBlend do
        $\omega = m/(\text{numberOfLinesToBlend} + 1)$
        $\mathbf{n}_{\mathbf{i}+m} = \omega \mathbf{n}_{\mathbf{i}+m} + (1-\omega)\mathbf{n}_{\mathbf{i}}$
        $\mathbf{n}_{\mathbf{i}+m} = \mathbf{n}_{\mathbf{i}+m} / \|\mathbf{n}_{\mathbf{i}+m}\|$
    end for

    Compute smoothed area elements:
    $\omega := .1625$    ▷ under-relaxation parameter
    $\overline{\Delta a}_{\mathbf{i}} := \Delta a_{\mathbf{i}}$
    for $m = 1, 2, \ldots,$ numberOfVolumeSmoothingIterations do
        $das_{\mathbf{i}} := (1-\omega)\overline{\Delta a}_{\mathbf{i}} + \omega/4(\overline{\Delta a}_{i_1+1} + \overline{\Delta a}_{i_1-1} + \overline{\Delta a}_{i_2+1} + \overline{\Delta a}_{i_2-1})$
    end for
    $\mathbf{xr}_{\mathbf{i}} := \frac{1}{2}(\mathbf{x}_{i_1+1} - \mathbf{x}_{i_1-1})$
    if $n_d \equiv 2$ then
        $\mathbf{xs}_{\mathbf{i}} := \frac{1}{2}(\mathbf{x}_{i_2+1} - \mathbf{x}_{i_2-1})$
    end if
end function
```

Algorithm 4 Apply boundary conditions.

```
function applyBoundaryConditions($\mathbf{x}^n, \mathbf{n}$)
    Purpose : Apply boundary conditions to the current front. Optionally equidsitribute lines.
    For surface grids, project the front onto the reference surface
    Param: ig : Denotes the index for ghost points
    Param: ib : Denotes the index for boundary points

    if boundaryCondition == freeFloating then
        $\mathbf{x_{ig}} = 2\mathbf{x_{ib}} - \mathbf{x_{ib+1}}$                    ▷ extrapolate ghost line
    else if boundaryCondition == outwardSplay then
    else if boundaryCondition == fixXfloatYZ then
    else if boundaryCondition == periodic then
    else if boundaryCondition == matchToMapping then
        Project the boundary points onto the boundary mapping
        $\mathbf{B}$ := mapping defining the boundary that we should match to
        $\mathbf{B}$.project($\mathbf{x_{ib}}$)
    end if

    equidistributeGridLines( $\mathbf{x}^{n+1}$ )

    if this is a surface-grid then
        project($\mathbf{x}^0, \mathbf{n_i}; \mathbf{R}$)
    end if

    apply periodic boundary conditions
end function
```

Algorithm 5 Get the distance to step.

```
function getDistanceToStep($\mathbf{d}$)
    Purpose : Return the current suggested distance to step
    Param: n : current step number
    Param: N : number of lines to march
    Param: D : distance to march
    Param: $\alpha$ : geometric stretching factor

    if constant spacing then
        $\mathbf{d} = D/N$
    else if geometric spacing then
        $\mathbf{d} = D\alpha^n \frac{\alpha-1}{\alpha^{N+1}-1}$
    end if
end function
```

Algorithm 6 Equidistribute the grid lines.

```
1: function equidistributeGridLines($\mathbf{x}^n$)
2:     Purpose : Adjust points based on the arclength and curvature
3:     Param: $\mathbf{x}^n$ : current grid point positions
4:     Param: $\mathbf{c}$ : A curve that interpolates the points $\mathbf{x}^n$
5:     Param: $\omega^E$ : equidistribution weight, $0 \le \omega^E \le 1$
6:     Param: $\alpha^A$ : equidistribution arclength weight
7:     Param: $\alpha^C$ : equidistribution curvature weight
8:
9:     if $n_d \equiv 2$ : only used for domain dimension equal to 2 then
10:        : compute a weight function based on arclength and curvature
11:        $\mathbf{ds}_{i_1+\frac{1}{2}} := \|\mathbf{x}_{i_1+1} - \mathbf{x}_{i_1}\|$          ▷ chord length
12:        $\mathbf{dss}_{i_1} := \|\mathbf{x}_{i_1+1} - 2\mathbf{x}_{i_1} + \mathbf{x}_{i_1-1}\|$
13:        $w_{i_1+\frac{1}{2}} := \alpha^A \mathbf{ds}_{i_1+\frac{1}{2}}/\|\mathbf{ds}\|_\infty + \alpha^C \mathbf{dss}_{i_1+\frac{1}{2}}/\|\mathbf{dss}\|_\infty$
14:        equidistribute the weight function: determine positions $\mathbf{r}_i^E \in [0,1]$ such that:
15:        $\int_{\mathbf{r}_i^E}^{\mathbf{r}_{i+1}^E} w dr = \frac{1}{N}\int_0^1 w dr$
16:        $\mathbf{x}^E := \mathbf{c}(\mathbf{r}^E)$          ▷ re-evaluate the curve at the new positions
17:        : weighted average of current positions and equidistributed positions
18:        $\mathbf{x}^n := (1-\omega^E)\mathbf{x}^n + \omega^E \mathbf{x}^E$
19:    end if
20: end function
```

2 Steger-Chan Hyperbolic Marching

The approach discussed here follows *Enhancements of a Three-Dimensional Hyperbolic Grid Generation Scheme* by Chan and Steger[2] and *A Hyperbolic Surface Grid Generation Scheme and Its Applications* by Chan and Buning[1].

Notation: Unit square coodinates (r,s,t) with marching direction along t.

Given a surface $\mathbf{x}(r,s,t=0)$ we wish to generate a volume grid, $\mathbf{x}(r,s,t)$, that extends in a direction that is nearly normal to the surface. To do this we choose $\mathbf{x}_t$ to satisfy

$$\mathbf{x}_r \cdot \mathbf{x}_t = 0 \tag{4}$$
$$\mathbf{x}_s \cdot \mathbf{x}_t = 0 \tag{5}$$
$$\mathbf{x}_r \times \mathbf{x}_s \cdot \mathbf{x}_t = \Delta V(r,s,t) \tag{6}$$

where we have added the additional condition specifying the local volume of the cell.

To avoid a small time step in advancing the front we linearize and use an implicit time stepping method. We can linearize about the state $\mathbf{x}^0$ (which we will later take to be the current time step). It is easier if we linearize the equations in their original form of equation 6,

$$\begin{aligned}\mathbf{x}_t^0 \cdot \mathbf{x}_r + \mathbf{x}_r^0 \cdot \mathbf{x}_t &= 0\\ \mathbf{x}_t^0 \cdot \mathbf{x}_s + \mathbf{x}_s^0 \cdot \mathbf{x}_t &= 0\\ (\mathbf{x}_s^0 \times \mathbf{x}_t^0)\cdot \mathbf{x}_r + (\mathbf{x}_t^0 \times \mathbf{x}_r^0)\cdot \mathbf{x}_s + (\mathbf{x}_r^0 \times \mathbf{x}_s^0)\cdot \mathbf{x}_t &= \Delta V(r,s,t) + 2\Delta V^0\end{aligned}$$

or in matrix form

$$A_0\mathbf{x}_r + B_0\mathbf{x}_s + C_0\mathbf{x}_t = \mathbf{f}$$

or

$$\begin{bmatrix}(\mathbf{x}_t^0)^T\\ 0\\ (\mathbf{x}_s^0\times\mathbf{x}_t^0)^T\end{bmatrix}\mathbf{x}_r + \begin{bmatrix}0\\ (\mathbf{x}_t^0)^T\\ (\mathbf{x}_t^0\times\mathbf{x}_r^0)^T\end{bmatrix}\mathbf{x}_s + \begin{bmatrix}(\mathbf{x}_r^0)^T\\ (\mathbf{x}_s^0)^T\\ (\mathbf{x}_r^0\times\mathbf{x}_s^0)^T\end{bmatrix}\mathbf{x}_t = \begin{bmatrix}0\\ 0\\ V(r,s,t)+2\Delta V^0\end{bmatrix}$$

or

$$\mathbf{x}_t = -C_0^{-1}A_0\mathbf{x}_r - C_0^{-1}B_0\mathbf{x}_s + C_0^{-1}\mathbf{f}$$

Writing this in incremental form

$$A_0(\mathbf{x}_r - \mathbf{x}_r^0) + B_0(\mathbf{x}_s - \mathbf{x}_s^0) + C_0\mathbf{x}_t = \mathbf{g} = \begin{bmatrix}0\\ 0\\ V(r,s,t)\end{bmatrix}$$

If $\delta\mathbf{x} = \mathbf{x}^{n+1} - \mathbf{x}^n$ then using the approximation $\mathbf{x}_t \approx \mathbf{x}^{n+1} - \mathbf{x}^n$ $(\Delta t = 1)$

$$\delta\mathbf{x} = -C_0^{-1}A_0\ \delta\mathbf{x}_r - C_0^{-1}B_0\ \delta\mathbf{x}_s + C_0^{-1}\mathbf{g}$$

Discretizing with backward Euler

$$[I + C_0^{-1}A_0\Delta_{0r} + C_0^{-1}B_0\Delta_{0s}]\ \delta\mathbf{x} = C_0^{-1}\mathbf{g}$$

approximate factorization

$$[I + C_0^{-1}A_0\Delta_{0r}][I + C_0^{-1}B_0\Delta_{0s}]\ \delta\mathbf{x} = C_0^{-1}\mathbf{g}$$

Smoothing is added to this equation

$$\begin{aligned}[I + C_0^{-1}A_0\Delta_{0r} - \epsilon_i\Delta_{+r}\Delta_{-r}][I + C_0^{-1}B_0\Delta_{0s} - \epsilon_i\Delta_{+s}\Delta_{-s}]\ \delta\mathbf{x} = C_0^{-1}\mathbf{g}&\\ + \epsilon_e\Delta_{+r}\Delta_{-r}\mathbf{x}^n + \epsilon_e\Delta_{+s}\Delta_{-s}\mathbf{x}^n&\\ + D_r\mathbf{x}^n + D_s\mathbf{x}^n&\end{aligned}$$

Note that the smoothing terms have components in the normal and tangential directions. The smoothing will increase the step size in concave regions $\mathbf{n}\cdot(\mathbf{x}_{rr}+\mathbf{x}_{ss}) > 0$ and decrease the step size in convex regions.

The cell volume can be computed to be the local area of the element times a user specified step length,

$$\Delta V = \Delta L(r,s,t)\Delta A(r,s,t)$$

where the step length may be chosen to stretch the grids lines in any desired way. The area $\Delta A(r,s,t)$ is usually smoothed using a few Jacobi iterations.

The variable dissipation coefficients are defined by

$$\begin{aligned} D_r &= \epsilon_{er}(r,s,t)\Delta_{+r}\Delta_{-r} \\ \epsilon_{er}(r,s,t) &= \epsilon_e R_r N_r \\ N_r &= \|\mathbf{x}_t\|/\|\mathbf{x}_r\| \\ R_r &= K^n \overline{d}_{\mathbf{i}}^r a_{\mathbf{i}}^r \end{aligned}$$

Scaling function, K^n,

$$K^n = \begin{cases} \sqrt{(n-1)/(n_{\max}-1)} & \text{if } 2 \le n \le n_{\text{trans}} \\ \sqrt{(n_{\text{trans}}-1)/(n_{\max}-1)} & \text{if } n_{\text{trans}}+1 \le n \le n_{\max} \end{cases}$$

Grid point distribution sensor,

$$\begin{aligned} \overline{d}_{\mathbf{i}}^r &= \max((d_{\mathbf{i}}^r)^{2/K^n}, 0.1) \\ d_{\mathbf{i}}^r &= \frac{\|\Delta_{+r}\mathbf{x}^{n-1}\| + \|\Delta_{-r}\mathbf{x}^{n-1}\|}{\|\Delta_{+r}\mathbf{x}^{n}\| + \|\Delta_{-r}\mathbf{x}^{n}\|} \end{aligned}$$

$$n_{\text{trans}} = \max((3/4)n_{\max}, \text{minimum n where} \max_{\mathbf{i}} d_{\mathbf{i}}^r(n) - \max_{\mathbf{i}} d_{\mathbf{i}}^r(n-1) < 0 \text{ or } \max_{\mathbf{i}} d_{\mathbf{i}}^s(n) - \max_{\mathbf{i}} d_{\mathbf{i}}^s(n-1) < 0)$$

Grid angle distribution sensor

$$a_{\mathbf{i}}^r = \begin{cases} (1-\cos^2\alpha_{\mathbf{i}})^{-1} & \text{if } 0 \le \alpha_{\mathbf{i}} \le \pi/2 \\ 1 & \text{if } \pi/2 < \alpha_{\mathbf{i}} \le \pi \end{cases}$$

$$\begin{aligned} \cos\alpha_{\mathbf{i}} &= \hat{\mathbf{n}}_{\mathbf{i}}\cdot\mathbf{t}_+^r = \hat{\mathbf{n}}_{\mathbf{i}}\cdot\mathbf{t}_-^r && \text{angle between normal and tangent} \\ \mathbf{n} &= (\mathbf{t}_+^r - \mathbf{t}_-^r)\times(\mathbf{t}_+^s - \mathbf{t}_-^s) \\ \widehat{\mathbf{n}} &= \frac{\mathbf{n}}{\|\mathbf{n}\|} && \text{normal to surface} \\ \mathbf{t}_+^r &= \frac{\Delta_{+r}\mathbf{x}}{\|\Delta_{+r}\mathbf{x}\|} && \text{unit tangent to the right of node } \mathbf{i} \\ \mathbf{t}_-^r &= \frac{\Delta_{-r}\mathbf{x}}{\|\Delta_{-r}\mathbf{x}\|} && \text{unit tangent to the left of node } \mathbf{i} \end{aligned}$$

Note that

$$\begin{aligned} C_0 &= \begin{bmatrix} (\mathbf{x}_r^0)^T \\ (\mathbf{x}_s^0)^T \\ \mathbf{N}_0^T \end{bmatrix} \\ \mathbf{N}_0 &= \mathbf{x}_r^0 \times \mathbf{x}_s^0 = \|\mathbf{x}_r^0 \times \mathbf{x}_s^0\|\mathbf{n}_0 \\ \det(C_0) &= \mathbf{x}_r^0 \times \mathbf{x}_s^0 \cdot \mathbf{N}_0 = \mathbf{N}_0\cdot\mathbf{N}_0 = \|\mathbf{N}_0\|^2 \end{aligned}$$

and C_0^{-1} is given explicitly by

$$C_0^{-1} = \left[(\mathbf{x}_s \times \mathbf{N}_0)/\|\mathbf{N}_0\|^2 \qquad (-\mathbf{x}_r \times \mathbf{N}_0)/\|\mathbf{N}_0\|^2 \qquad \mathbf{N}_0/\|\mathbf{N}_0\|^2 \right]$$

In particular

$$C_0^{-1}\mathbf{g} = V(r,s,t)\frac{\mathbf{x}_r^0 \times \mathbf{x}_s^0}{\|\mathbf{x}_r^0 \times \mathbf{x}_s^0\|^2}$$

3 The Osher-Sethian Level Set (Hamilton-Jacobi) Marching Equations

Reference *Level Set Methods* by J. Sethian[6].

Another way to generate a hyperbolic grid, suggested by Sethian as an application of level-set methods is to solve the equations

$$\begin{aligned} \mathbf{x}_t &= (1-\epsilon\kappa)\mathbf{n}(\mathbf{x}) = V(\mathbf{x})\mathbf{n} \\ \kappa &= \text{ curvature} \end{aligned}$$

If $\epsilon > 0$ then we are guaranteed that grid lines will not locally cross. Here the curvature κ causes the grid to move faster where the curvature is negative and slower where it is positive.

For grid generation we do not want to march backwards so we must not let the speed function V become negative. Sethian also adds smoothing in the tangential direction.

The curvature of a curve $x(r)$ is $\mathbf{k} = \mathbf{x}_{ss}$ where s is the arclength or in terms of a general parameterization:

$$\mathbf{k} = \frac{\mathbf{x}_r \times \mathbf{x}_{rr}}{\|\mathbf{x}_r\|^3} = \mathbf{x}_{ss}$$

The curvature has dimensions of one over a length.

I prefer to use a non-dimensional form for the curvature

$$k_r(\mathbf{x}) = \frac{\mathbf{n}\cdot\mathbf{x}_{rr}}{\|\mathbf{x}_r\|}$$

with the speed function

$$V(\mathbf{x}) = max(V_{\min}, 1 + \epsilon\max(k_r, k_s))$$

4 Distributing points by equidistribution of a weight function

For 2D grids or 3D surfaces (i.e. domainDimension==2) the grid lines in the tangential direction (i.e. not the marching direction) can be distributed to place more points where the curvature or arclength is large. This option can be combined, in a weighted fashion, with the other marching methods. Here is how this is done:

1. Take a step with the hyperbolic generator to give positions $\mathbf{x}$.
2. Equidistribute the points $\mathbf{x}$ using a weighted combination of arclength and curvature,

   $$\mathbf{x}^E = \text{Equidistribute}(\mathbf{x})$$

   This equidistribution is performed by the `ReparameterizationTransform`, described elsewhere.
3. Choose the new positions to be a weighted average of the original positions and the equidistributed points

   $$\begin{aligned} \mathbf{x}^{n+1} &= (1-\alpha)\mathbf{x} + \alpha\mathbf{x}^E \\ \alpha &= \text{equidistributionWeight} \end{aligned}$$

Notes:

- weighting by arclength is quite useful in many situations. It can be used to build a nice surface grid.
- weighting by curvature doesn't work very well; this needs some work to make the correct defintion of the curvature.

5 Boundary conditions

The enum `BoundaryCondition` defines the available boundary conditions,

freeFloating boundary values obtained by extrapolation. $\mathbf{u}_{-1} = 2\mathbf{u}_0 - \mathbf{u}_1$.

outwardSplay This boundary condition causes the boundary of the grid to splay outwards or inwards in proportion to the distance marched. Choose a value of

splayFactor=0. : no splay

splayFactor=.1 : small amount or splay.

splayFactor=1. : a large splay (generates a nearly circular boundary ??).

splayFactor=-.2 : negative for inward splay (doesn't woork too well)

The splay is computed as

$$\begin{aligned}
d &= \|\mathbf{x}_0^n - \mathbf{x}_0^{n-1}\| \qquad \text{(marching distance)} \\
\mathbf{v} &= \mathbf{x}_0 - \mathbf{x}_1 \qquad \text{(vector along ouwtard tangent)} \\
\mathbf{x}_{-1} &= 2\mathbf{x}_0 - \mathbf{x}_1 + \lambda d \frac{\mathbf{v}}{\|\mathbf{v}\|} \\
\mathbf{x}_0 &= .5\mathbf{x}_0 + .25(\mathbf{x}_{-1} + \mathbf{x}_1) \\
\lambda &= \text{splayFactor}
\end{aligned}$$

fixXfloatYZ : the x values of the boundary points are kept constant.

fixYfloatXZ : the y values of the boundary points are kept constant.

fixZfloatXY : the z values of the boundary points are kept constant.

floatXfixYZ : the y, z values of the boundary points are kept constant.

floatYfixXZ : the x, z values of the boundary points are kept constant.

floatZfixXY : the x, y values of the boundary points are kept constant.

floatCollapsed ??

periodic

xSymmetryPlane

ySymmetryPlane

zSymmetryPlane

singularAxis

matchToMapping : project the boundary values to lie on a given Mapping (or CompositeSurface). The projection is done so that the grid lines hitting the boundary are nearly orthogonal. This projection is defined by taking the predicted positions $\mathbf{x}_i$ and changing the boundary value $\mathbf{x}_0$ and the ghost value by

$$\begin{aligned}\mathbf{x}_0 &\leftarrow \mathbf{P}(\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_0) \qquad \text{(project onto the BC mapping)}\\ \mathbf{x}_{-1} &\leftarrow 2\mathbf{x}_0 - \mathbf{x}_1\\ \mathbf{x}_{-1} &\leftarrow \mathbf{x}_{-1} + (\mathbf{n}_0\cdot(\mathbf{x}_1 - \mathbf{x}_{-1}))\mathbf{n}\end{aligned}$$

With $\theta = 1$ the boundary value would be the projection of $\mathbf{x}_1$ onto the boundary.

matchToPlane : like matchToMapping except that you will be prompted to define an arbitrary plane to use as the mapping to match to.

5.1 Boundaries, Ghost Points and the BoundaryOffset

The `HyperbolicMapping` adds an extra line of points outside the grid; these are called **ghost points**. Ghost points are used to make it easier to apply boundary conditions and will likely be used when the grid is used in a PDE solver.

When a grid is generated with the hyperbolic grid generator one has a choice of which line to use as the ghost line. Let's say we are building a grid starting from a curve and that we put $N+1$ points on the curve, $\mathbf{x}_i$, $i = 0,\ldots,N$. Normally the points $i = 0$ and $i = N$ will be the boundary points and the points $i = -1$ and $i = N+1$ will be the ghost points. The `boundaryOffset(side,axis)` array can be used to change the position of the boundary. By setting `boundaryOffset(0,0)=1` the point $i = 1$ will become the boundary point and the point $i = 0$ will be the ghost point. See Section 6.1 for an example.

It may be important to choose a `boundaryOffset(0,0)=1` when growing a surface grid since one may want to be able to precisely place the last grid line (next to a crease in the surface, for example). (Appears in the asmo example).

The last line generated by the marching algorithm is always treated as a ghost point, since we do not want to create an extra line by extrapolation say. Thus `boundaryOffset(1,domainDimension)`≥ 1 where domainDimension equals 2 for a a grid in two dimensions or a surface grid in three dimensions, and domainDimension equals 3 for a 3d volume grid.

5.2 Normal Blending

When a boundary condition is specified so that the grid must match to some specified Mapping at the boundary then the normals near the boundary are blended with the direction taken by the boundary. This is necessary when the direction of the boundary is not normal to the starting surface.

The blending is done with a simple linear function for points

$$\begin{aligned}\omega_i &= \frac{i-b}{N-b}\\ \mathbf{n}_i &= \omega_i\mathbf{n}_i + (1-\omega_i)\mathbf{n}_b \qquad i = 0,1,\ldots,N\end{aligned}$$

The number of points to be blended can be specified.

5.3 Projection of boundary points on surfaces

For surface grids we project all ghost point values onto the reference surface. This always includes points on the ghost lines in the non-marching direction but also the ghost lines in the marching direction if the boundary condition in that direction is set to 0 (i.e. interpolation). In the latter case the ghost points are obtained first by extrapolation and then these extrapolated points are projected.

To prevent the projection of boundary use the **project ghost points** menu option to turn off the projection of ghost points on specified sides.

5.4 Heuristic Comments on Hyperbolic Parameters

There are many parameters to the hyperbolic grid generator. Here are some heuristics that you can use to help you choose the right values.

uniform dissipation coefficient : This term wants to make the front flat. This is the coefficient of the smoothing term $\Delta_r \mathbf{u}$. In concave corners it will cause the front to move faster since this is what happens when the front in straightened out. At convex corners the front will move slower and could move in the wrong direction if it is flattened out too much.

volume smoothing iterations : This term wants to make the grid spacing along the front become uniform. It will tend to make the outer surface become a spherical shape. As the number of these smoothing iterations is increased the speed of the front will become inversely proportional to the cell area. Small cells will move faster than large cells. This term will never cause the front to move backward.

5.5 Hints to making a grid

If you are having trouble making a grid

take a few small steps : first try to make a grid very close to the starting surface.

increase the number of steps : for a fixed marching distance. This will allow the grid more time to deal with difficult situations. After building a grid will lots of points you can change the resolution at the very end by using the 'lines' option. This will cause the fine resolution grid to be interpolated on a coarser grid using the interpolation defined in the `DataPointMapping`.

6 Creating a surface grid on another Mapping or CompositeSurface

A surface grid can be grown on any Mapping defining a surface or on a CompositeSurface which consists of a set up sub-surfaces.

To grow a new hyperbolic surface grid on another surface:

1. Define an initial curve to start from:

   User defined : before entering the `HyperbolicMapping` menu you may define an initial curve using any available Mapping.

   curve from edges : Create an initial curve as the union of edges from the reference surface. You can interactively choose edges of surfaces or sub-surfaces.

   curve from a coordinate line : choose a coordinate line from the reference surface.

   project a line : define a line segement in 3D which is projected onto the reference surface.

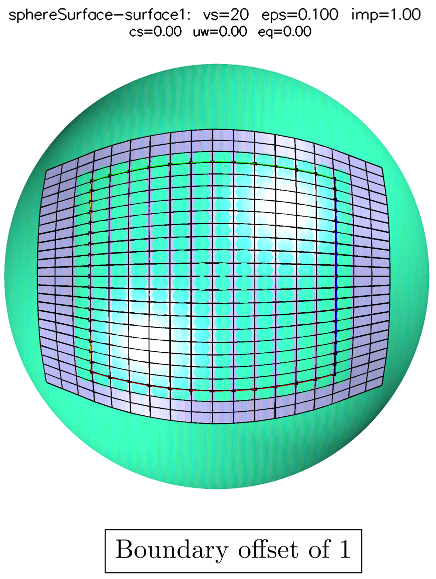
   project a spline : define a spline in 3D which is projected onto the reference surface.

2. Create the hyperbolic surface patch by growing the grid from the initial curve in either direction or in both directions.

6.1 Use of the 'boundary offset' option when generating grids on CAD.

The 'boundary offset' option can be used to force ghost points of surface grid to lie precisely on the CAD surface (see also the discussion in Section 5.1). This is especially useful when building grids to be used with high-order discretizations that require multiple ghost points (e.g. for interpolation). By default, ghost points are computed by extrapolation from the surface grid points that were computed by marching. Rather than project the ghost points on to the surface (which can be problematic – but maybe should be added as an option), the boundary of the grid is shifted inward by a specified number of grid points so that the ghost points are taken from those surface grid points that were generated by marching. This is illustrated in Figure 2 for a surface grid generated on a sphere. By increasing the boundary offset, more and more ghost points can be placed precisely on the surface. In general the 'boundary offset' can be specified as an different integer for each side of the grid.

Note: in practice one should not set the 'boundary offset' when making the surface grid, but rather when the volume grid is generated from the surface grid (this is due to the current implementation). The Ogen command file `Overture/sampleGrids/hypeCyl.cmd` demonstrates the use of the 'boundary offset' to generate a grid for a high-order accurate discretization.



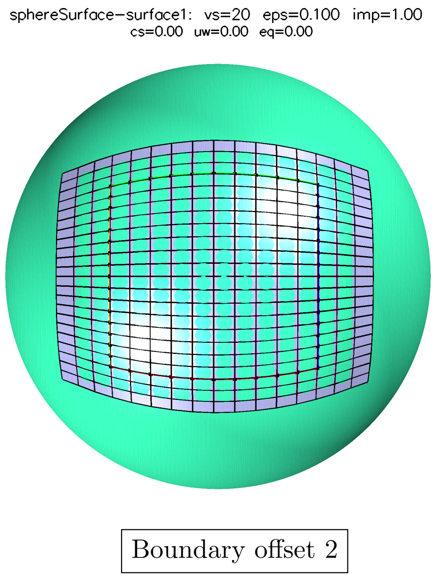


Figure 2: The 'boundary offset' option can be used to enure that ghost points lie on the CAD surface. This may be needed for high-order accurate approximations. The surface grids are shown with 3 ghost lines. The boundary of the surface grid is shown with the red, green, blue and yellow lines. This boundary moves as the boundary offset is changed. Extrapolated ghost points that do not lie precisely on the surface appear shaded blue since they lie slightly above the actual surface.

7 Examples

Parameters appearing in the figure titles

vs : number of volume smooths

eps : coefficient of the dissipation term

imp : coefficient of the implicit time stepping. $imp = 1.$ is fully implicit, $imp = 0.$ is explicit.

cs : curvature speed coefficient.

uw : coefficient of the upwind method.

eq : coefficient of the equidistribution.

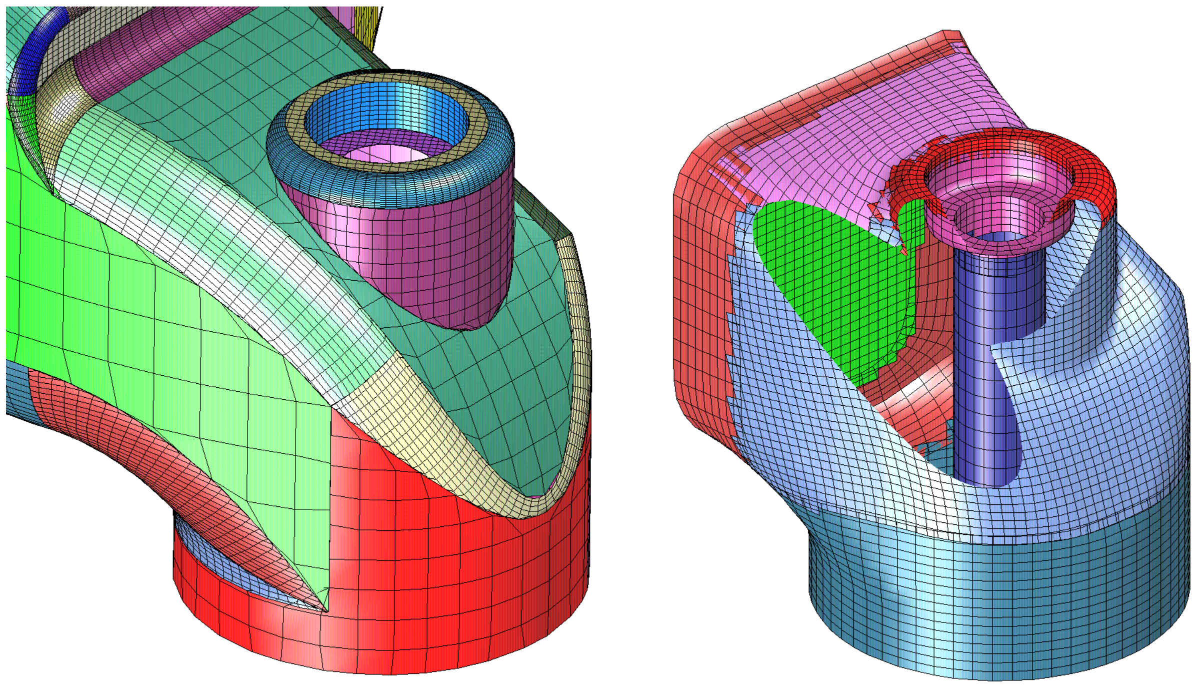


Figure 3: An overlapping grid (bottom) generated on a portion of the CAD surface (top). Most of the component grids that make up the overlapping grid were generated with the hyperbolic grid generator.

7.1 Bump

These figures show a hyperbolic grid generated in both directions from a smooth spline. The effect of changing various parameters is demonstrated. See the command file `Overture/sampleMappings/hypeBump.cmd`

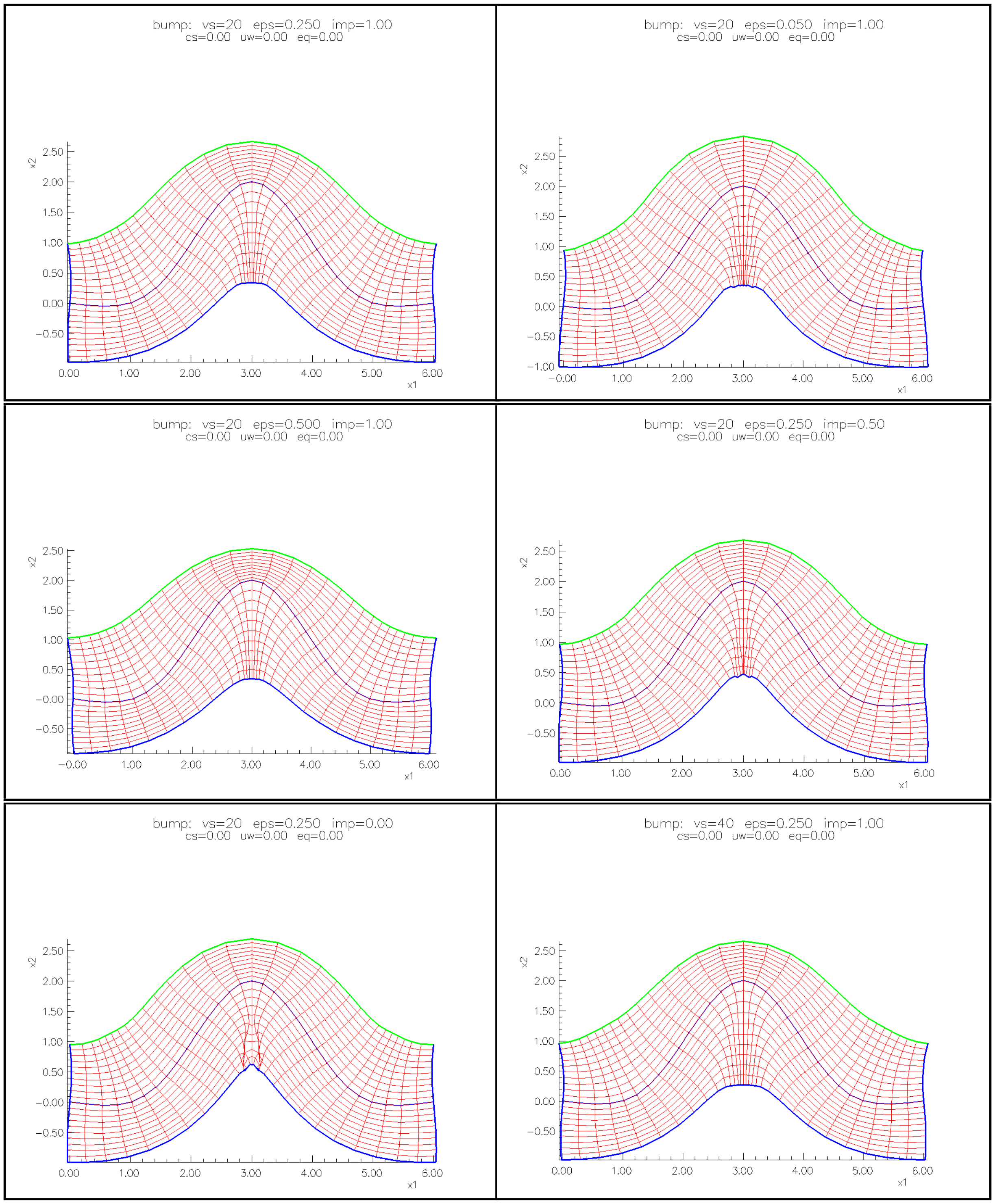


Figure 4: Hyperbolic grids generated using different values of the papers (noted in the titles)

7.2 Flat Plate

A spline is built to define a 'flat plate' with rounded edges. The shape preserving option is used with the spline which allows only a few knots to define the spline. A hypebolic grid is grown starting from the spline. The figures show the resulting grids as various parameters are changed. See the command file `Overture/sampleMappings/hypeLine.cmd`

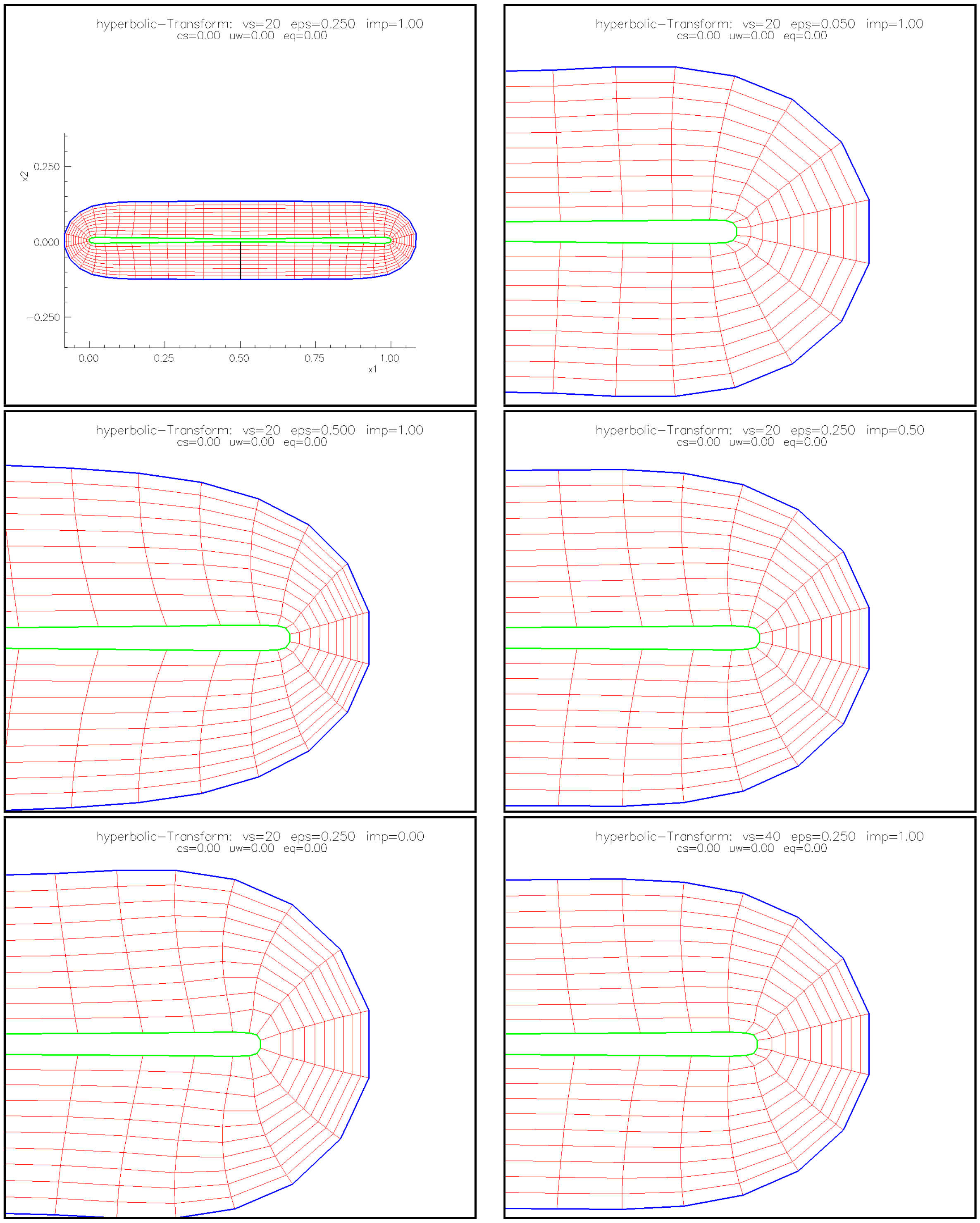
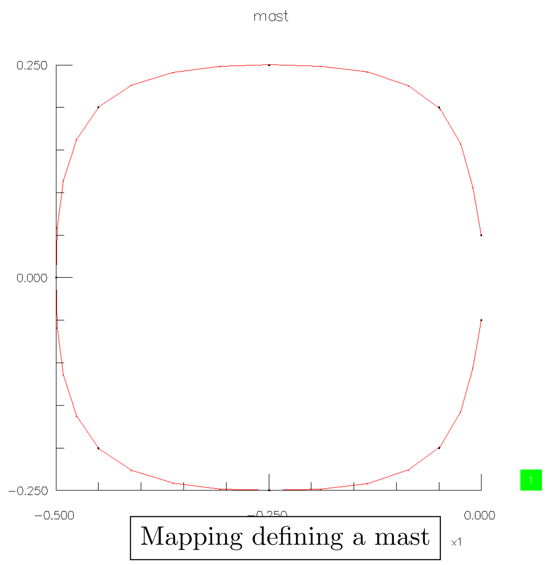

hyperbolic−Transform: vs=20 eps=0.250 imp=1.00
cs=0.00 uw=0.00 eq=0.00
x2
0.250
0.000
−0.250
0.00
0.25
0.50
0.75
1.00
x1
hyperbolic−Transform: vs=20 eps=0.050 imp=1.00
cs=0.00 uw=0.00 eq=0.00
hyperbolic−Transform: vs=20 eps=0.500 imp=1.00
cs=0.00 uw=0.00 eq=0.00
hyperbolic−Transform: vs=20 eps=0.250 imp=0.50
cs=0.00 uw=0.00 eq=0.00
hyperbolic−Transform: vs=20 eps=0.250 imp=0.00
cs=0.00 uw=0.00 eq=0.00
hyperbolic−Transform: vs=40 eps=0.250 imp=1.00
cs=0.00 uw=0.00 eq=0.00

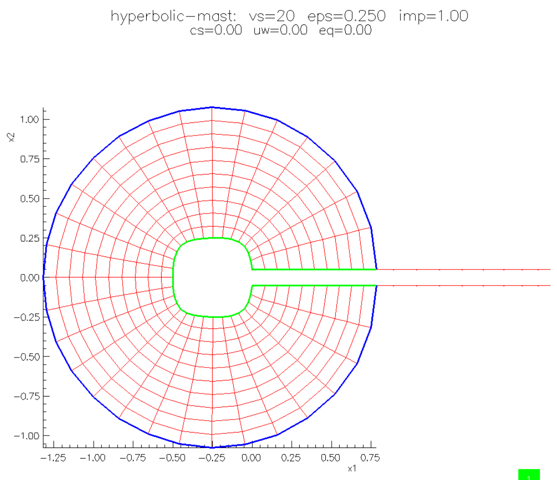
Figure 5: Hyperbolic grids generated using different values of the papers (noted in the titles)

7.3 Mast for a sail

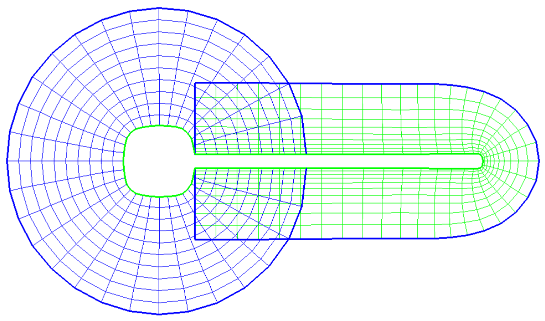
This example shows the use of the 'match to a mapping' boundary condition. In this case the boundary condition for the hyperbolic marching is that the boundary points should lie on some other specified Mapping. See the command file `Overture/sampleMappings/mastSail2d.cmd`



Mapping defining a mast



A hyperbolic grid is marched starting from the mast and matching to the sail



Hyperbolic grids created for a mast attached to a sail

Figure 6: Hyperbolic grids generated for a mast and sail.

7.4 Airfoil grids

The `AirfoilMapping` can be used to generate various types of airfoil shapes. These shapes can be used as starting curves for the hypebolic grid generator. Some care must be taken at the trailing edge since the curvature is so large. The boundary condition '`trailing edge`' is specified so the grid generator can choose a good marching direction at the trailing edge.

See the command file `Overture/sampleMappings/hypeNaca.cmd`

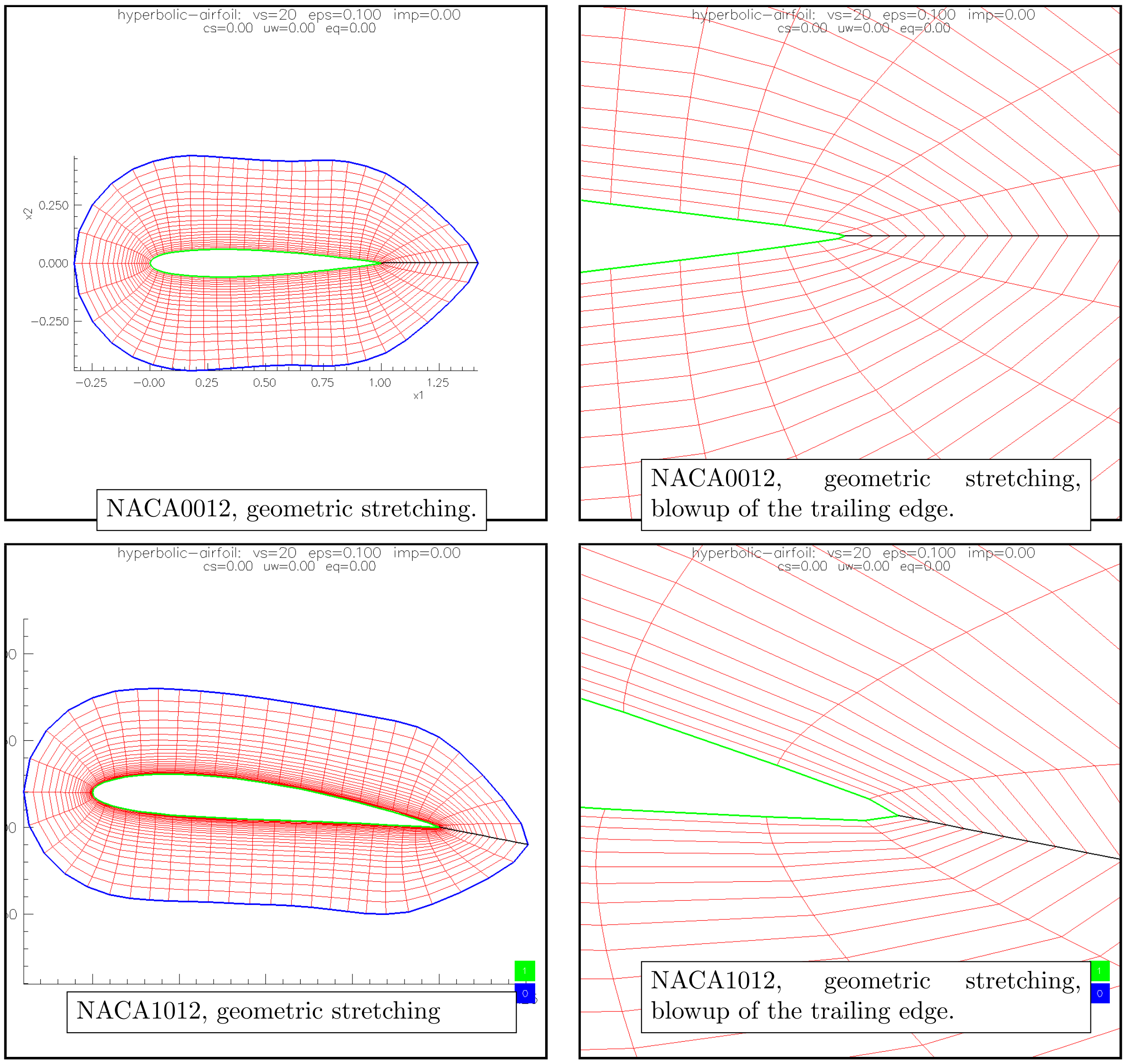
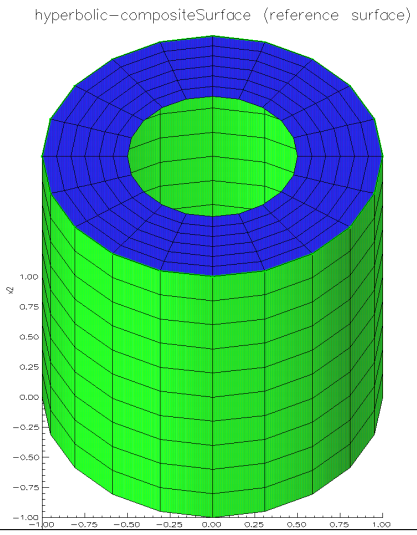

hyperbolic−airfoil: vs=20 eps=0.100 imp=0.00
cs=0.00 uw=0.00 eq=0.00
x2
0.250
0.000
−0.250
−0.25
−0.00
0.25
0.50
0.75
1.00
1.25
x1
NACA0012, geometric stretching.
NACA0012, geometric stretching, blowup of the trailing edge.
NACA1012, geometric stretching
NACA1012, geometric stretching, blowup of the trailing edge.

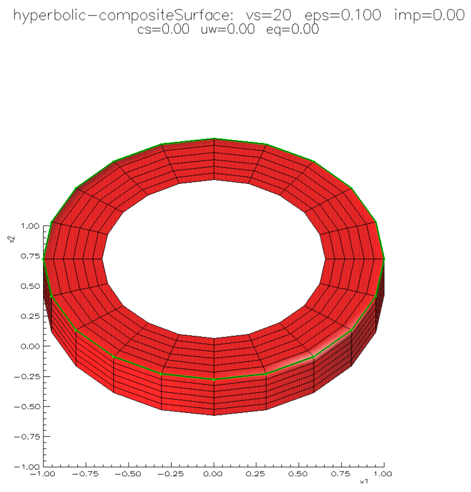
Figure 7: Hyperbolic grids generated for NACA airfoils.

7.5 Surface grid generation on a CompositeSurface for a soup can

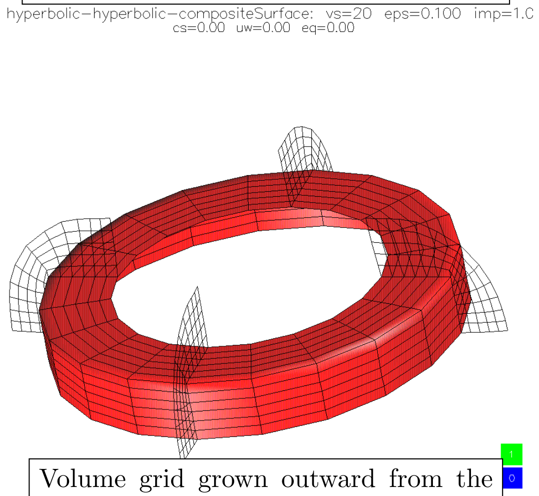
In this example we first build a CompositeSurface for a soup can consisting of two subsurfaces. A surface grid is then generated around the edge. A volume grid is grown outward from the surface grid. See the command file `Overture/sampleMappings/hypeCan.cmd`



Reference surface is a CompositeSurface



Surface grid grown in both directions from the corner.


hyperbolic–hyperbolic–compositeSurface: vs=20 eps=0.100 imp=1.0
cs=0.00 uw=0.00 eq=0.00
1
0

Volume grid grown outward from the surface grid.

Figure 8: Hyperbolic grids generated for a *soup can*.

7.6 Surface and Volume Grid Generation on a CAD model for an Automobile.

Figure (**??**) show an overlapping grid for the *ASMO* prototype automobile. The geometry of the asmo is defined by a CAD model and saved in an IGES file.

Creating an overlapping grid for this geometry requires some experience in using the various tools - **rap** for CAD fixup, **mbuilder** for building mappings and hyperbolic grids and **ogen**, the overlapping grid generator.

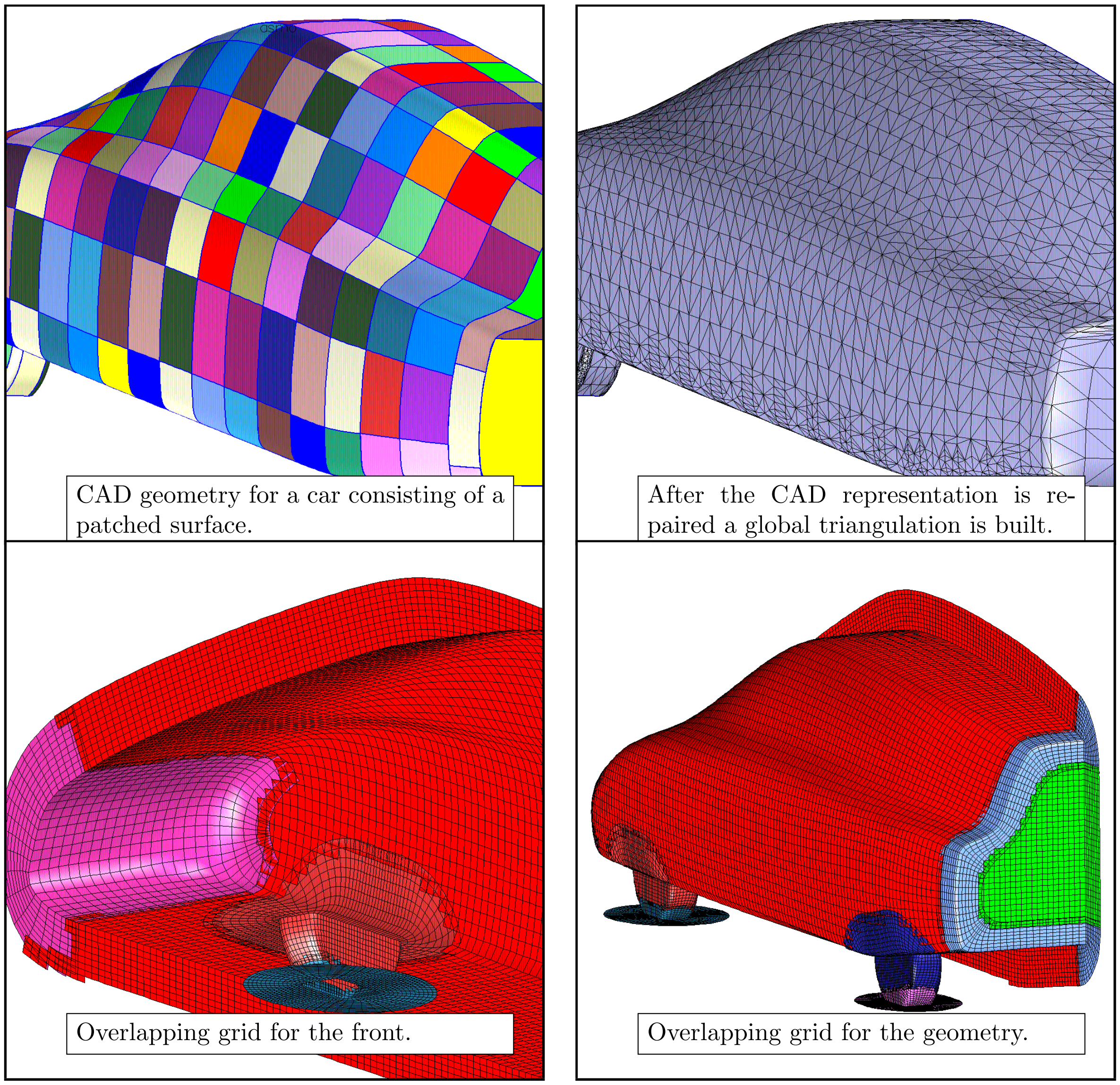
Here are the steps taken to build the grid for the asmo. The steps will use the command files `asmoNoWheels.cmd`, `asmoBody.cmd`, `asmoFrontWheel.cmd`, `asmoBackWheel.cmd` and `asmo.cmd` found in the `Overture/sampleGrids` directory. They will also use the **rap**, **mbuilder** and **ogen** programs found in the `Overture/bin` directory. The IGES file defining the asmo CAD geometry is found in `Overture/-sampleMappings/asmo.igs`.

Step 1. CAD cleanup with rap: The **rap** program is used to build a version of the asmo without any wheels by running "rap asmoNoWheels.cmd". This program will pause at various stages so you can see what it does. It will create the file `asmoNoWheels.hdf`. The CAD model has duplicate surfaces which are deleted. After deleting the wheels the holes in the body are filled in by deleting trimming curves. After cleanup the connectivity is determined and a global triangulation is built. Refer to publications[4, 5] for further details of the CAD fixup and connectivity algorithms. These are available from the Overture web page, under publications.

Step 2. Grids for the body: Running "mbuilder asmoBody.cmd" will generate grids around the body of the asmo. The file `asmoNoWheels.hdf` built in step 1. will be read in. The file `asmoBody.hdf` will be created. The `mbuilder` program will use the **MappingBuilder** class to coordinate the construction of grids on the CAD surface. Body fitted grids are built by choosing a starting curve on the surface, growing a surface grid from this start curve and then generating a volume grid from the surface grid. The aim was to build a few number of high quality grids. We also build a large cartesian box to place the car in.

Step 3. Grids for the wheels: Running "mbuilder asmoFrontWheel.cmd" and "mbuilder asmoBack-Wheel.cmd" will generate grids for the front wheel and back wheel and create the files `asmoFrontWheel.hdf` and `asmoBackWheel.hdf`. These command files will directly read the asmo IGES file `asmo.igs` and select a subset of the surfaces to work on since it is faster to work with a smaller geometry. The trimmed surfaces near the rear wheel do not match very well and the surface has to be repaired.

Step 4. Overlapping grid: Running "ogen asmo.cmd" will build the overlapping grid for the asmo. It will read the component grids generated by the previous steps. When the asmo grid was made for the first time, the wheels were left off in order to simplify the grid generation. The wheels were then added, one at a time. This is in general a good approach to use: slowly build up the grid for a complicated geometry starting from a simplified version.


CAD geometry for a car consisting of a patched surface.

After the CAD representation is repaired a global triangulation is built.

Overlapping grid for the front.

Overlapping grid for the geometry.

Figure 9: Generating a grid on a CAD model for the ASMO car.

7.7 Grid Generation for a Truck.

In this case study we illustrate the use of the hyperbolic grid generator and mappingBuilder to construct grids on a truck. This 'truck' is actually a wind-tunnel model.

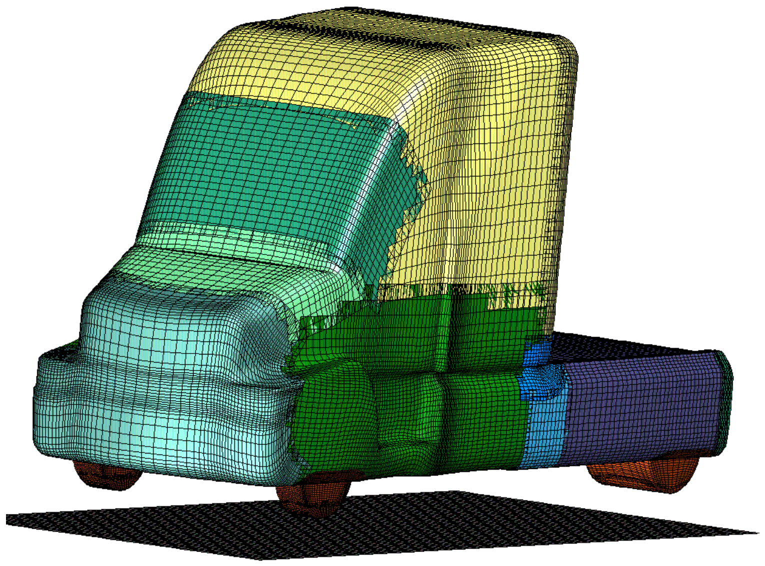


Figure 10: Overlapping grid for the cab of a truck.

7.7.1 Front

Figure (11) shows the surface grid for the front of the truck. The starting curve for the grid was generating by cutting the CAD model with a plane. The surface grid was grown, stretched and smoothed.

Remarks:

- The equidistribution weight was turned on to generate the surface grid. This allowed the grid to march more cleanly over the surface.
- The CAD surface grid bends sharply in a small region on top of the bumper. The grid was smoothed in this region without projecting onto the CAD surface so as to smooth this indentation out a little bit.

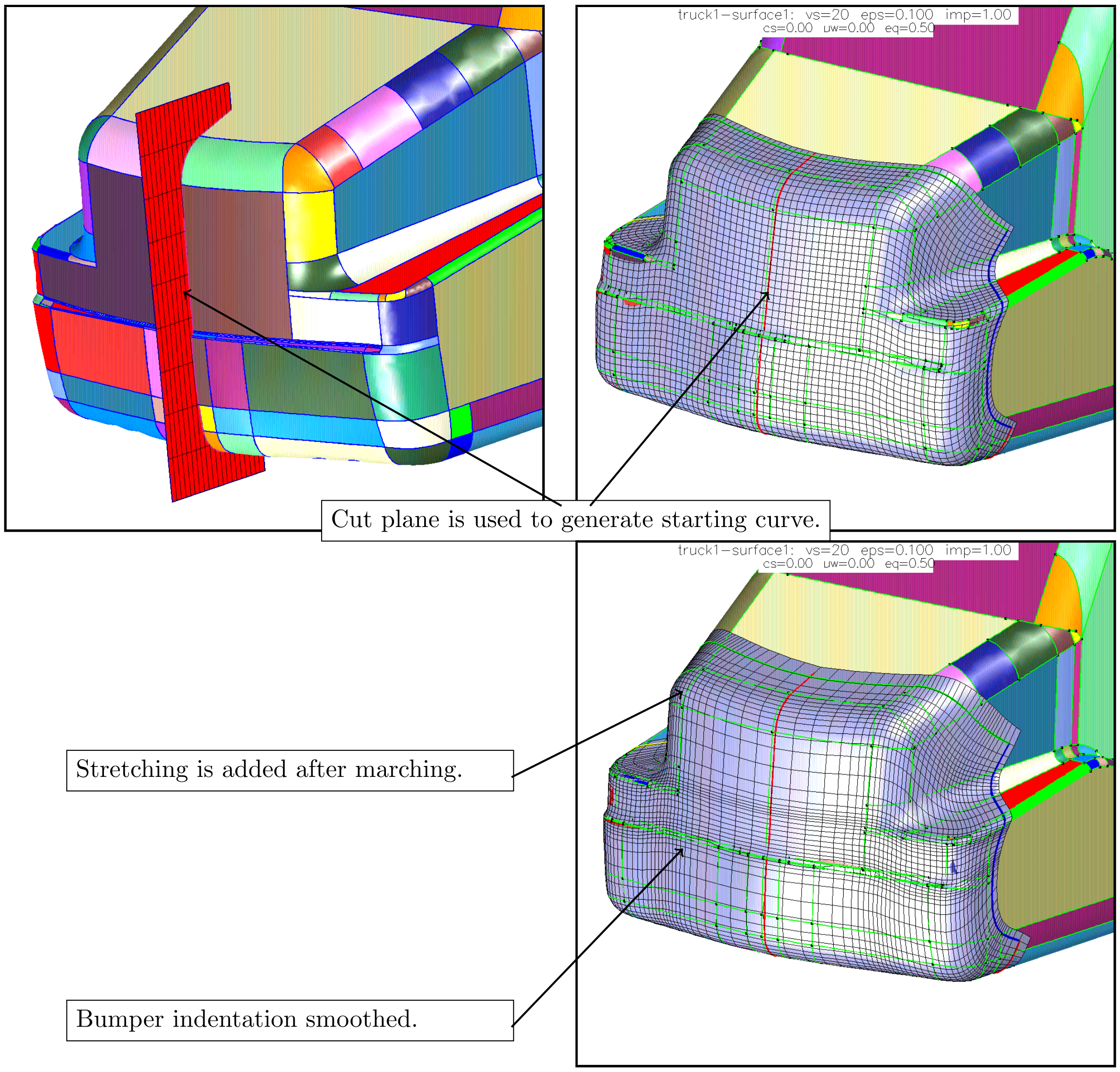

truck1−surface1: vs=20 eps=0.100 imp=1.00
cs=0.00 uw=0.00 eq=0.50
Cut plane is used to generate starting curve.
truck1−surface1: vs=20 eps=0.100 imp=1.00
cs=0.00 uw=0.00 eq=0.50
Stretching is added after marching.
Bumper indentation smoothed.

Figure 11: Grids for the front of the cab. Top left: starting curve is generated by intersecting a plane with the surface. Top right: surface grid is grown. Bottom: surface grid is stretched and smoothed. The identation over the bumper is smoothed slightly by locally smoothing without projecting onto the CAD surface.

7.7.2 Wheel

Each wheel is covered with two grids as shown in figure (12). The tricky part here is to have a grid line follow all the corners.

Remarks:

- The surface grid for the wheel-body join was generated by matching to 'interior matching curves'. This causes grid lines to follow the edges of the wheel.

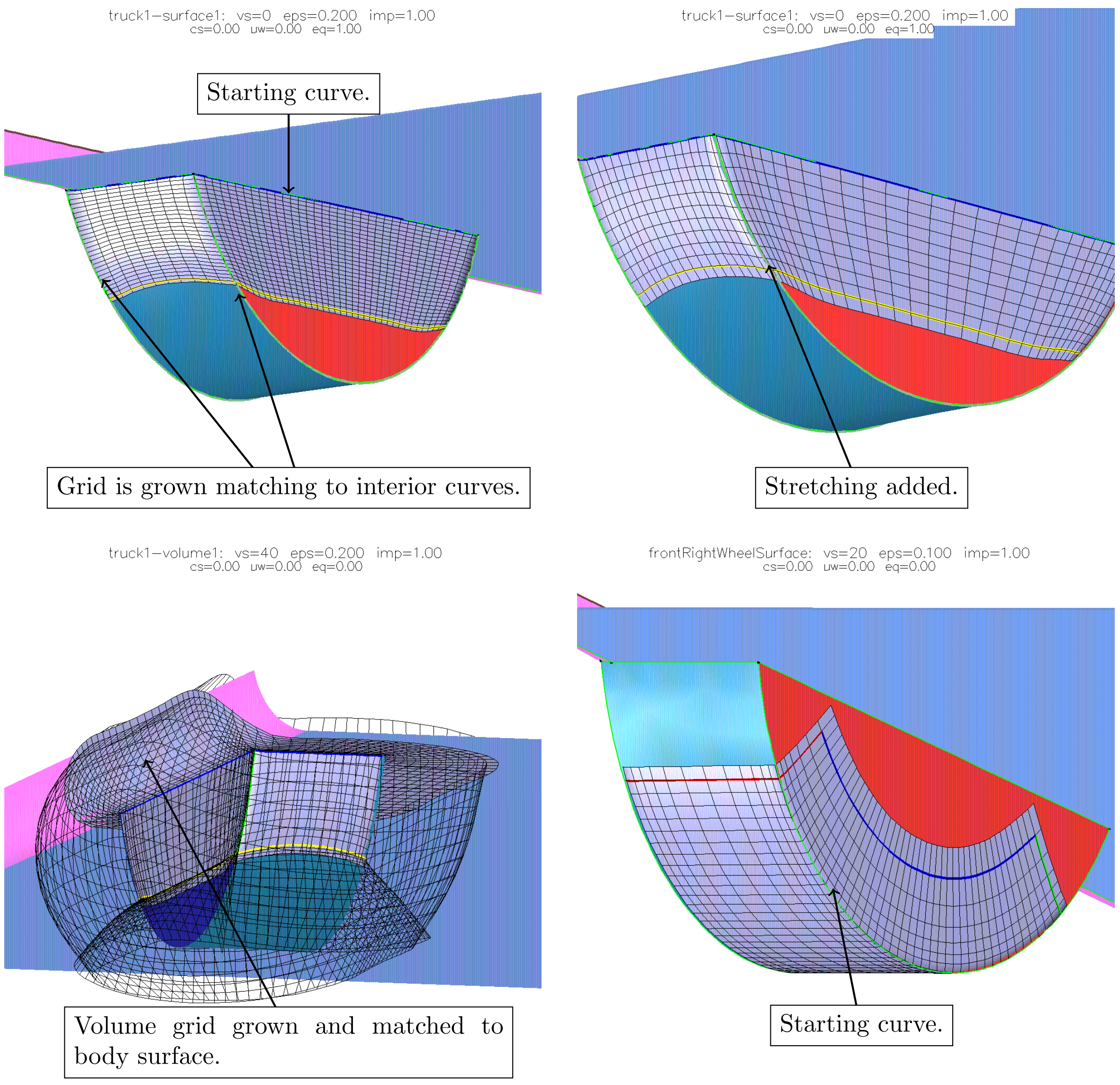

truck1−surface1: vs=0 eps=0.200 imp=1.00
cs=0.00 uw=0.00 eq=1.00
Starting curve.
Grid is grown matching to interior curves.
truck1−surface1: vs=0 eps=0.200 imp=1.00
cs=0.00 uw=0.00 eq=1.00
Stretching added.
truck1−volume1: vs=40 eps=0.200 imp=1.00
cs=0.00 uw=0.00 eq=0.00
Volume grid grown and matched to body surface.
frontRightWheelSurface: vs=20 eps=0.100 imp=1.00
cs=0.00 uw=0.00 eq=0.00
Starting curve.

Figure 12: Wheel Grids. Top left: Surface grid is generated by matching to 'interior matching curves'. Top right: surface grid is stretched and smoothed. Bottom left: volume grid for wheel-body-join is grown by matching to body surface. Bottom right: Surface grid for wheel.

7.7.3 Cab tender

The grid for the cab tender was built by projecting a transfinite interpolation (TFI) mapping onto the CAD surface. This option is available from the **MappingBuilder**. This gave a nicer grid than the hyperbolic grid generator. Two curves were defined for the TFI mapping by intersecting the CAD surface with planes.

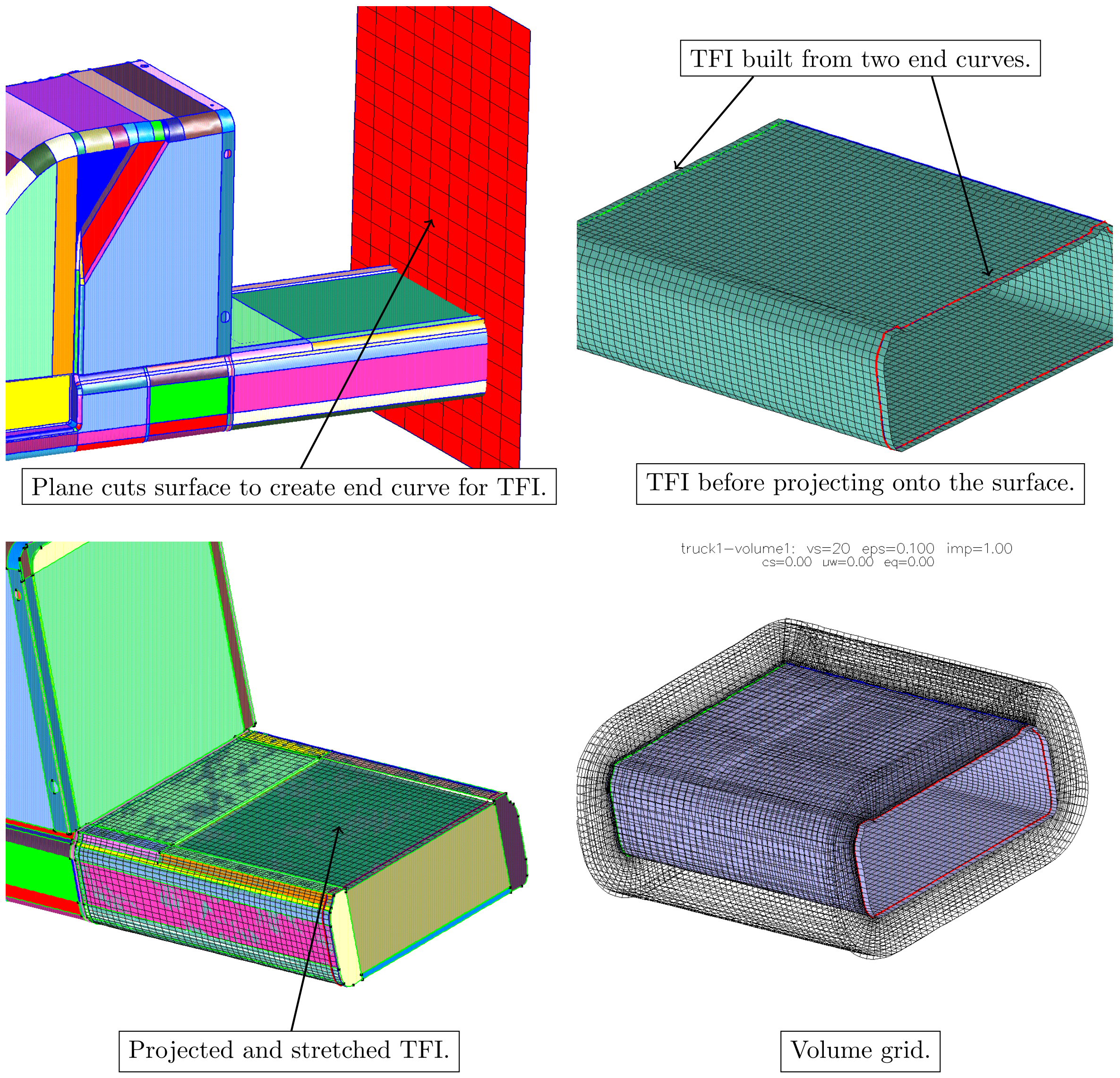

TFI built from two end curves.
Plane cuts surface to create end curve for TFI.
TFI before projecting onto the surface.
truck1–volume1: vs=20 eps=0.100 imp=1.00
cs=0.00 uw=0.00 eq=0.00
Projected and stretched TFI.
Volume grid.

Figure 13: Cab Tender Grids.

8 Class member functions

8.1 Constructor

HyperbolicMapping()

Purpose: Create a mapping that can be used to generate a hyperbolic volume grid.

8.2 Constructor

HyperbolicMapping(Mapping & surface_)

Purpose: Create a mapping that can be used to generate a hyperbolic volume grid.

surface_ (input): Generate the grid starting from this curve (2D) or surface (3D)

8.3 Constructor

HyperbolicMapping(Mapping & surface_, Mapping & startingCurve)

Purpose: Create a hyperbolic surface grid.

surface_ (input): Generate the grid on this surface in 3D.

startingCurve :

8.4 isDefined

bool
isDefined() const

Description: return true if the Mapping has been defined.

8.5 printStatistics

int
printStatistics(FILE *file =stdout)

Description: Print timing statistics.

8.6 setBoundaryConditionMapping

```
//================================================================
int
setBoundaryConditionMapping(const int & side,
                            const int & axis,
                            Mapping & map,
                            const int & mapSide =-1,
                            const int & mapAxis =-1)
```

Purpose: Supply a mapping to match a boundary condition to.

side,axis (input) : match to this boundary of the hyperbolic grid.

map (input): Match the boundary values of the grid to lie on this surface or match the boundary values to lie on the face of this Mapping defined by (mapSide,mapAxis).

mapSide,mapAxis (input) : use this face of the Mapping 'map'. Supply these values if the hyperbolic grid is to be matched to a face of 'map', rather than map itself.

8.7 setSurface

int
setSurface(Mapping & surface_, bool isSurfaceGrid =true */, bool init /* = true)

Purpose: Supply the curve/surface from which the grid will be generated.

surface_ (input): Generate the grid starting from this curve (2D) or surface (3D)

isSurfaceGrid (input) : set to true if a surface grid should be built, set to false if a volume grid should be created.

init (input) : if true, initialize hyperbolic parameters such as the distance to march etc. If false, keep parameters as they are.

8.8 setIsSurfaceGrid

void
setIsSurfaceGrid(bool trueOrFalse)

Purpose: Indicate whether a surface grid or volume grid should be built.

trueOrFalse (input) : set to true if a surface grid should be built, set to false if a volume grid should be created.

8.9 setStartingCurve

int
setStartingCurve(Mapping & startingCurve, bool init = true)

Purpose: Supply a starting curve for a surface grid.

startingCurve (input):

init (input) : if true, initialize hyperbolic parameters such as the distance to march etc. If false, keep parameters as they are.

8.10 saveReferenceSurfaceWhenPut

int
saveReferenceSurfaceWhenPut(bool trueOrFalse = true)

Purpose: Save the reference surface and starting curve when 'put' is called.

init (input) : if true, initialize hyperbolic parameters such as the distance to march etc.

8.11 setup

int
setup()

Access: protected.

Purpose: Define properties of this mapping

8.12 setParameters

int
setParameters(const HyperbolicParameter & par,
const IntegerArray & ipar = Overture::nullIntArray(),
const RealArray & rpar = Overture::nullRealDistributedArray(),
const Direction & direction = bothDirections)

Purpose: Define a parameter for the hyperbolic grid generator.

par (input): The possible value come from the enum `HyperbolicParameter`:

- **growInBothDirections** : grow the grid in both directions.
- **growInTheReverseDirection** : grow the grid in the reverse direction (this will result in a left handed coordinate system.
- **numberOfRegionsInTheNormalDirection**
- **stretchingInTheNormalDirection**
- **linesInTheNormalDirection** : specify the number of lines to use in the normal direction.
- **distanceToMarch** : ipar(0) = region number, rpar(0) = distance
- **spacing** : ipar(0) = region number, rpar(0) = dz0, rpar(1)=dz1
- **boundaryConditions**
- **dissipation**
- **volumeParameters**
- **barthImplicitness**
- **axisParameters**
- **THEtargetGridSpacing** : rpar(0) gives the target grid spacing when choosing the number of grid points in the tangential direction (i.e. for the start curve and for marching on surfaces). A negative value means use a best guess.
- **THEinitialGridSpacing** : rpar(0) gives the target grid spacing when choosing the number of grid points for marching volume grids (e.g. the spacing of the first grid line for volume grids). A negative value means use a best guess.
- **THEspacingType** : a value from SpacingType enum
- **THEspacingOption** : a value from SpacinOptionEnum
- **THEgeometricFactor** : the geometric spacing factor

value (input):

direction (input) : The hyperbolic surface can be grown in two possible directions (or both directions). `direction` indicates which direction the new parameter values should apply to: (enum Direction)

- **direction=bothDirections** : parameters apply to both the forward and reverse directions.
- **direction=forwardDirection** : parameters apply to the forward direction.
- **direction=reverseDirection** : parameters apply to the reverse direction.

8.13 setPlotOption

int
setPlotOption(PlotOptionEnum option, int value)

Description: set a plot option.

choosePlotBoundsFromGlobalBounds: if true use global bounds for plotting, allows calling program to set the view

8.14 smooth

int
smooth(GenericGraphicsInterface & gi, GraphicsParameters & parameters)

Access: protected

Description: Smooth the hyperbolic grid using the elliptic grid generator.

8.15 inspectInitialSurface

int
inspectInitialSurface(realArray & xSurface, realArray & normal)

Purpose: Inspect the initial surface for corners etc.

8.16 generate

int
generateOld()

Purpose: Generate the hyperbolic grid. *** OLD VERSION ***

Return value: 0 on success, 1=hypgen not available

References

[1] W. M. Chan and P. G. Buning, *A hyperbolic surface grid generation scheme and its applications*, paper 94-2208, AIAA, 1994.

[2] W. M. Chan and J. L. Steger, *Enhancements of a three-dimensional hyperbolic grid generation scheme*, Appl. Math. Comput., 51 (1992), pp. 181–205.

[3] W. D. Henshaw, *Mappings for Overture, a description of the Mapping class and documentation for many useful Mappings*, Research Report UCRL-MA-132239, Lawrence Livermore National Laboratory, 1998.

[4] ——, *An algorithm for projecting points onto a patched CAD model*, Research Report UCRL-JC-144016, Lawrence Livermore National Laboratory, 2001. To appear in Engineering with Computers.

[5] N. A. Petersson and K. K. Chand, *Detecting translation errors in CAD surfaces and preparing geometries for mesh generation*, in Proceeding of the 10th International Meshing Rountable, 2001.

[6] J. A. Sethian, *Level Set Methods*, Cambridge University Press, 1996.

Index

algorithm, 7

equidistribution, 14