Un método de elementos finitos adaptativo para problemas de optimización de forma

Pedro Morin

Instituto de Matemática Aplicada
del Litoral
Universidad Nacional del Litoral
Santa Fe, Argentina

En colaboración con

R.H. Nochetto (Maryland)
M.S. Pauletti (Maryland-Texas)
M. Verani (Milan)

IMAL — 29 de octubre de 2010

Layout

Layout

Shape optimization problems

- Cost functional $J = J(\Omega, u(\Omega))$
- $\Omega \subset \mathbb{R}^d$
- $u = u(\Omega)$ solution to a PDE $\mathcal{A}u(\Omega) = f$ in Ω
- Minimization problem:

$$\text{find } \Omega^* \in \mathcal{U}_{\text{ad}} : \qquad J(\Omega^*, u(\Omega^*)) = \min_{\Omega \in \mathcal{U}_{\text{ad}}} J(\Omega, u(\Omega))$$

$\mathcal{U}_{\text{ad}}$: set of admissible domains

$$\begin{aligned} \text{Minimize} \quad & J = J(\Omega, u(\Omega)) \\ \text{s.t.} \quad & \Omega \in \mathcal{U}_{\text{ad}} \qquad \mathcal{A}u(\Omega) = f \end{aligned}$$

Shape optimization problems

- Cost functional $J = J(\Omega, u(\Omega))$
- $\Omega \subset \mathbb{R}^d$
- $u = u(\Omega)$ solution to a PDE $\mathcal{A}u(\Omega) = f$ in Ω
- Minimization problem:

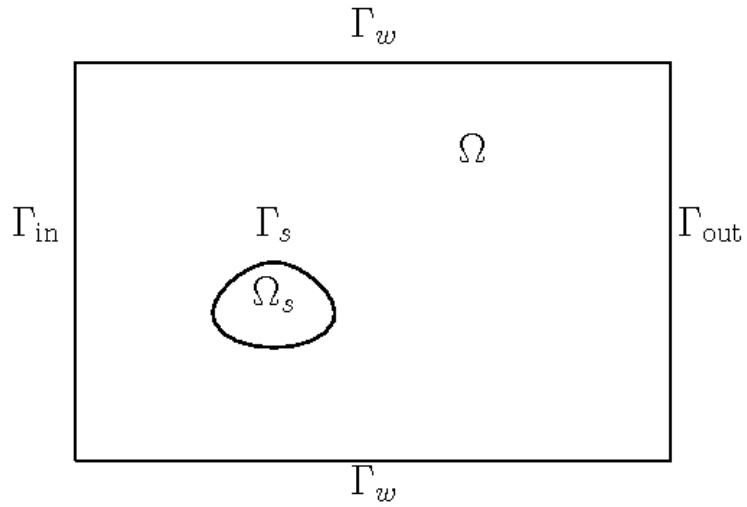
$$\text{find } \Omega^* \in \mathcal{U}_{\text{ad}} : \qquad J(\Omega^*, u(\Omega^*)) = \min_{\Omega \in \mathcal{U}_{\text{ad}}} J(\Omega, u(\Omega))$$

$\mathcal{U}_{\text{ad}}$: set of admissible domains

$$\begin{aligned} &\text{Minimize} && J = J(\Omega, u(\Omega)) \\ \text{s.t.} \quad &\Omega \in \mathcal{U}_{\text{ad}} && \mathcal{A}u(\Omega) = f \end{aligned}$$

Example: Minimization of an obstacle drag

$$
\begin{aligned}
-\nabla \cdot \boldsymbol{T}(\boldsymbol{u},p) &= 0 && \text{in } \Omega \\
\nabla \cdot \boldsymbol{u} &= 0 && \text{in } \Omega \\
\boldsymbol{u} &= \boldsymbol{u}_d && \text{on } \Gamma_{\text{in}} \cup \Gamma_s \cup \Gamma_w \\
\boldsymbol{T}(\boldsymbol{u},p)\cdot \boldsymbol{n} &= 0 && \text{on } \Gamma_{\text{out}}
\end{aligned}
$$



$$
\boldsymbol{T}(\boldsymbol{u},p) = 2\nu\varepsilon(\boldsymbol{u}) - p\boldsymbol{I}, \quad \varepsilon(\boldsymbol{u}) = \frac{\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T}{2}, \quad \boldsymbol{u}_d = \begin{cases} V_\infty \boldsymbol{i} & \text{on } \Gamma_{\text{in}} \\ \boldsymbol{0} & \text{on } \Gamma_w \cup \Gamma_s \end{cases}
$$

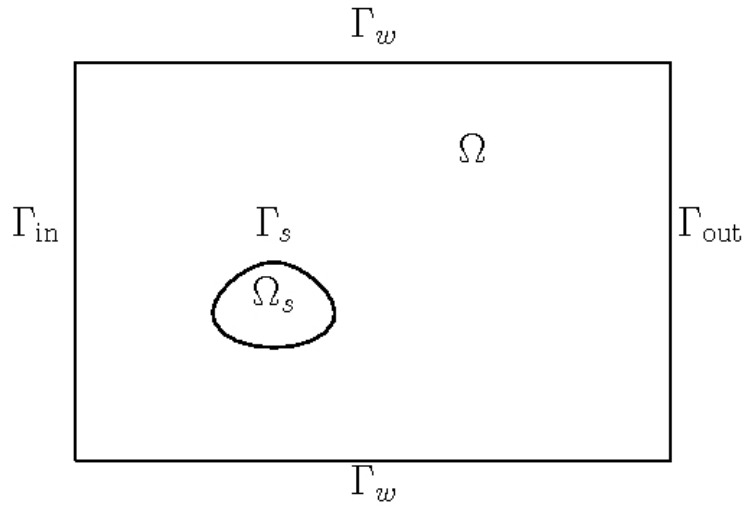
Drag Functional: $J(\Omega,[\boldsymbol{u},p]) = -\int_{\Gamma_s} \left(\boldsymbol{T}(\boldsymbol{u},p)\,\boldsymbol{n}\right)\cdot \boldsymbol{i}\; d\Gamma$

$$
\Omega^* \longleftarrow \min_{\Omega \in \mathcal{U}_{\text{ad}}} J(\Omega, u(\Omega)),
$$

$$
\mathcal{U}_{\text{ad}} = \{\Omega : |\Omega| = V, \text{ with } \Gamma_{\text{in}}, \Gamma_{\text{out}}, \Gamma_w \text{ fixed}\}
$$

Example: Minimization of an obstacle drag

$$
\begin{aligned}
-\nabla \cdot \boldsymbol{T}(\boldsymbol{u},p) &= 0 && \text{in } \Omega \\
\nabla \cdot \boldsymbol{u} &= 0 && \text{in } \Omega \\
\boldsymbol{u} &= \boldsymbol{u}_d && \text{on } \Gamma_{\text{in}} \cup \Gamma_s \cup \Gamma_w \\
\boldsymbol{T}(\boldsymbol{u},p)\cdot \boldsymbol{n} &= 0 && \text{on } \Gamma_{\text{out}}
\end{aligned}
$$



$$
\boldsymbol{T}(\boldsymbol{u},p) = 2\nu\varepsilon(\boldsymbol{u}) - p\boldsymbol{I}, \quad \varepsilon(\boldsymbol{u}) = \frac{\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T}{2}, \quad \boldsymbol{u}_d = \begin{cases} V_\infty \boldsymbol{i} & \text{on } \Gamma_{\text{in}} \\ \boldsymbol{0} & \text{on } \Gamma_w \cup \Gamma_s \end{cases}
$$

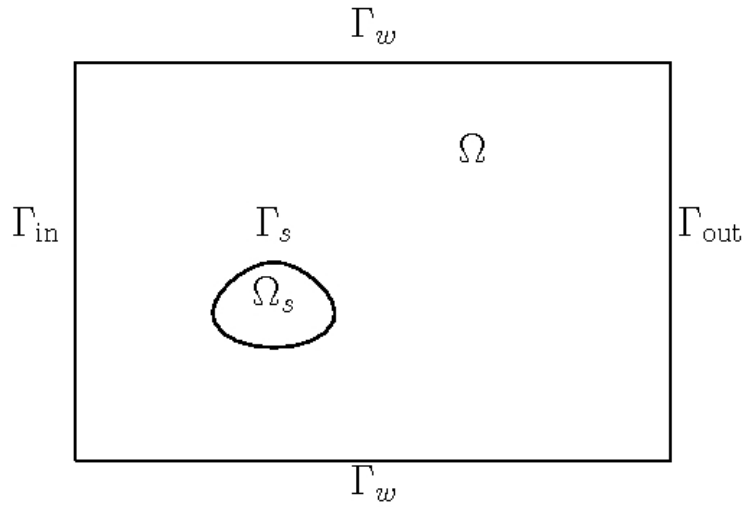
Drag Functional: $$J(\Omega,[\boldsymbol{u},p]) = -\int_{\Gamma_s} \big(\boldsymbol{T}(\boldsymbol{u},p)\,\boldsymbol{n}\big)\cdot \boldsymbol{i}\; d\Gamma$$

$$\Omega^* \longleftarrow \min_{\Omega\in\mathcal{U}_{\text{ad}}} J(\Omega, u(\Omega)),$$

$$\mathcal{U}_{\text{ad}} = \{\Omega : |\Omega| = V, \text{ with } \Gamma_{\text{in}}, \Gamma_{\text{out}}, \Gamma_w \text{ fixed}\}$$

Example: Minimization of an obstacle drag

$$
\begin{aligned}
-\nabla \cdot \boldsymbol{T}(\boldsymbol{u},p) &= 0 && \text{in } \Omega \\
\nabla \cdot \boldsymbol{u} &= 0 && \text{in } \Omega \\
\boldsymbol{u} &= \boldsymbol{u}_d && \text{on } \Gamma_{\text{in}} \cup \Gamma_s \cup \Gamma_w \\
\boldsymbol{T}(\boldsymbol{u},p)\cdot \boldsymbol{n} &= 0 && \text{on } \Gamma_{\text{out}}
\end{aligned}
$$



$$
\boldsymbol{T}(\boldsymbol{u},p) = 2\nu\varepsilon(\boldsymbol{u}) - p\boldsymbol{I}, \quad \varepsilon(\boldsymbol{u}) = \frac{\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T}{2}, \quad \boldsymbol{u}_d = \begin{cases} V_\infty \boldsymbol{i} & \text{on } \Gamma_{\text{in}} \\ \boldsymbol{0} & \text{on } \Gamma_w \cup \Gamma_s \end{cases}
$$

Drag Functional: $$J(\Omega,[\boldsymbol{u},p]) = -\int_{\Gamma_s} \big(\boldsymbol{T}(\boldsymbol{u},p)\,\boldsymbol{n}\big)\cdot \boldsymbol{i}\; d\Gamma$$

$$
\Omega^* \longleftarrow \min_{\Omega \in \mathcal{U}_{\text{ad}}} J(\Omega, u(\Omega)),
$$

$$
\mathcal{U}_{\text{ad}} = \{\Omega : |\Omega| = V, \text{ with } \Gamma_{\text{in}}, \Gamma_{\text{out}}, \Gamma_w \text{ fixed}\}
$$

Layout

Shape Calculus

We can define the shape derivative of a cost functional $J(\Omega)$ at Ω, in the direction of a vector field $\boldsymbol{v}$: $dJ(\Omega;\boldsymbol{v})$.

The velocity method: Let $\boldsymbol{v}$ be a smooth vector field. Define

$$\frac{d}{dt}X(t;x) = \boldsymbol{v}(X(t;x)) \quad t>0 \qquad\qquad x\in\mathbb{R}^d.$$
$$X(0,x) = x$$

Let Ω_t be the image under $X(t;\cdot)$ of Ω

$$\Omega_t := \Big\{ X(t;x) : x\in\Omega \Big\}$$

We then define the shape derivative of $J(\Omega)$ in the direction $\boldsymbol{v}$ as

$$dJ(\Omega;\boldsymbol{v}) = \lim_{t\to 0}\frac{J(\Omega_t)-J(\Omega)}{t}.$$

Shape Calculus

We can define the shape derivative of a cost functional $J(\Omega)$ at Ω, in the direction of a vector field $\boldsymbol{v}$: $dJ(\Omega;\boldsymbol{v})$.

The velocity method: Let $\boldsymbol{v}$ be a smooth vector field. Define

$$\begin{aligned} \frac{d}{dt}X(t;x) &= \boldsymbol{v}(X(t;x)) \quad t>0 \\ X(0,x) &= x \end{aligned} \qquad x \in \mathbb{R}^d.$$

Let Ω_t be the image under $X(t;\cdot)$ of Ω

$$\Omega_t := \Big\{ X(t;x) : x \in \Omega \Big\}$$

We then define the shape derivative of $J(\Omega)$ in the direction $\boldsymbol{v}$ as

$$dJ(\Omega;\boldsymbol{v}) = \lim_{t\to 0} \frac{J(\Omega_t)-J(\Omega)}{t}.$$

Shape Calculus

We can define the shape derivative of a cost functional $J(\Omega)$ at Ω, in the direction of a vector field $\boldsymbol{v}$: $dJ(\Omega;\boldsymbol{v})$.

The velocity method: Let $\boldsymbol{v}$ be a smooth vector field. Define

$$\frac{d}{dt}X(t;x) = \boldsymbol{v}(X(t;x)) \quad t>0 \qquad\qquad x\in\mathbb{R}^d.$$
$$X(0,x) = x$$

Let Ω_t be the image under $X(t;\cdot)$ of Ω

$$\Omega_t := \left\{ X(t;x) : x\in\Omega \right\}$$

We then define the shape derivative of $J(\Omega)$ in the direction $\boldsymbol{v}$ as

$$dJ(\Omega;\boldsymbol{v}) = \lim_{t\to 0}\frac{J(\Omega_t)-J(\Omega)}{t}.$$

Shape Calculus

We can define the shape derivative of a cost functional $J(\Omega)$ at Ω, in the direction of a vector field $\boldsymbol{v}$: $dJ(\Omega;\boldsymbol{v})$.

The velocity method: Let $\boldsymbol{v}$ be a smooth vector field. Define

$$\begin{aligned} \frac{d}{dt}X(t;x) &= \boldsymbol{v}(X(t;x)) \quad t>0 \\ X(0,x) &= x \end{aligned} \qquad x \in \mathbb{R}^d.$$

Let Ω_t be the image under $X(t;\cdot)$ of Ω

$$\Omega_t := \left\{ X(t;x) : x \in \Omega \right\}$$

We then define the shape derivative of $J(\Omega)$ in the direction $\boldsymbol{v}$ as

$$dJ(\Omega;\boldsymbol{v}) = \lim_{t\to 0} \frac{J(\Omega_t)-J(\Omega)}{t}.$$

Shape Calculus

Hadamard-Zolesio Theorem: There exists a function $g(\Omega)$ defined on $\partial\Omega$

$$dJ(\Omega; \boldsymbol{v}) = \int_{\partial\Omega} g(\Omega)(\boldsymbol{v} \cdot \boldsymbol{n})\, d\Gamma$$

The shape derivative depends only on the normal component of v on $\partial\Omega$.

Shape Calculus

Hadamard-Zolesio Theorem: There exists a function $g(\Omega)$ defined on $\partial\Omega$

$$dJ(\Omega;\boldsymbol{v}) = \int_{\partial\Omega} g(\Omega)(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

The shape derivative depends only on the normal component of $\boldsymbol{v}$ on $\partial\Omega$.

Shape Calculus

Hadamard-Zolesio Theorem: There exists a function $g(\Omega)$ defined on $\partial\Omega$

$$dJ(\Omega;\boldsymbol{v}) = \int_{\partial\Omega} g(\Omega)(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

The shape derivative depends only on the normal component of $\boldsymbol{v}$ on $\partial\Omega$.

Volume

$$J(\Omega) = \int_\Omega dx$$

$$dJ(\Omega;\boldsymbol{v}) = \int_\Omega \nabla\cdot\boldsymbol{v}\,dx = \int_\Gamma 1(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

$$J(\Omega) = \int_\Omega \phi\,dx \qquad (\phi:\mathbb{R}^d\to\mathbb{R})$$

$$dJ(\Omega;\boldsymbol{v}) = \int_\Omega \nabla\cdot(\phi\boldsymbol{v})\,dx = \int_\Gamma \phi(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

Shape Calculus

Hadamard-Zolesio Theorem: There exists a function $g(\Omega)$ defined on $\partial\Omega$

$$dJ(\Omega;\boldsymbol{v}) = \int_{\partial\Omega} g(\Omega)(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

The shape derivative depends only on the normal component of $\boldsymbol{v}$ on $\partial\Omega$.

Volume

$$J(\Omega) = \int_\Omega dx$$

$$dJ(\Omega;\boldsymbol{v}) = \int_\Omega \nabla\cdot\boldsymbol{v}\,dx = \int_\Gamma 1(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

$$J(\Omega) = \int_\Omega \phi\,dx \qquad (\phi:\mathbb{R}^d\to\mathbb{R})$$

$$dJ(\Omega;\boldsymbol{v}) = \int_\Omega \nabla\cdot(\phi\boldsymbol{v})\,dx = \int_\Gamma \phi(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

Shape Calculus

Hadamard-Zolesio Theorem: There exists a function $g(\Omega)$ defined on $\partial\Omega$

$$dJ(\Omega;\boldsymbol{v}) = \int_{\partial\Omega} g(\Omega)(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

The shape derivative depends only on the normal component of $\boldsymbol{v}$ on $\partial\Omega$.

Area/Perimeter

$$J(\Omega) = \int_{\partial\Omega} d\Gamma$$

$$dJ(\Omega;\boldsymbol{v}) = \int_{\partial\Omega} \kappa(\boldsymbol{v}\cdot\boldsymbol{n})d\Gamma$$

$$J(\Omega) = \int_{\partial\Omega} \phi\,d\Gamma \qquad (\phi:\mathbb{R}^d\to\mathbb{R})$$

$$dJ(\Omega;\boldsymbol{v}) = \int_{\partial\Omega} \left(\frac{\partial\phi}{\partial\boldsymbol{n}} + \phi\kappa\right)(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

Shape Calculus

Hadamard-Zolesio Theorem: There exists a function $g(\Omega)$ defined on $\partial\Omega$

$$dJ(\Omega;\boldsymbol{v}) = \int_{\partial\Omega} g(\Omega)(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

The shape derivative depends only on the normal component of $\boldsymbol{v}$ on $\partial\Omega$.

Area/Perimeter

$$J(\Omega) = \int_{\partial\Omega} d\Gamma$$

$$dJ(\Omega;\boldsymbol{v}) = \int_{\partial\Omega} \kappa(\boldsymbol{v}\cdot\boldsymbol{n})d\Gamma$$

$$J(\Omega) = \int_{\partial\Omega} \phi\,d\Gamma \qquad (\phi : \mathbb{R}^d \to \mathbb{R})$$

$$dJ(\Omega;\boldsymbol{v}) = \int_{\partial\Omega} \left(\frac{\partial\phi}{\partial\boldsymbol{n}} + \phi\kappa\right)(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

Shape Calculus

Hadamard-Zolesio Theorem: There exists a function $g(\Omega)$ defined on $\partial\Omega$

$$dJ(\Omega;\boldsymbol{v}) = \int_{\partial\Omega} g(\Omega)(\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

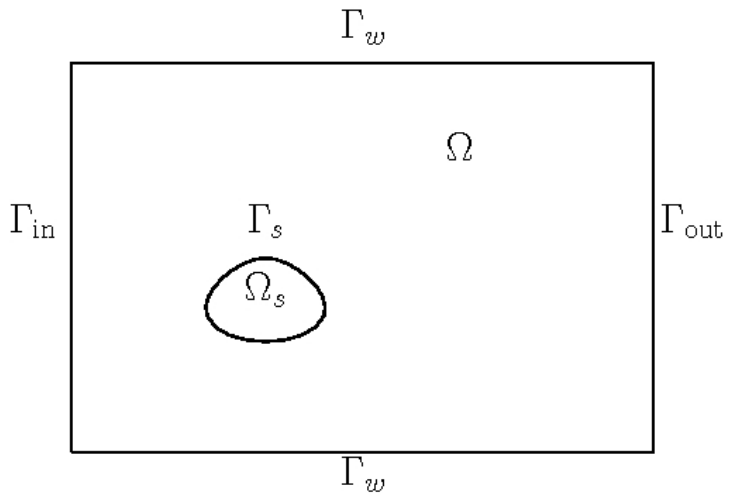
The shape derivative depends only on the normal component of $\boldsymbol{v}$ on $\partial\Omega$.

Obstacle Drag

$$dJ(\Omega;\boldsymbol{v}) = -2\nu \int_{\Gamma_s} [\varepsilon(\boldsymbol{u}) : \varepsilon(\boldsymbol{z})](\boldsymbol{v}\cdot\boldsymbol{n})\,d\Gamma$$

With z solution to the adjoint problem

$$\begin{aligned} -\nabla\cdot\boldsymbol{T}(\boldsymbol{z},q) &= 0 && \text{in } \Omega \\ \nabla\cdot\boldsymbol{z} &= 0 && \text{in } \Omega \\ \boldsymbol{z} &= -\boldsymbol{i} && \text{on } \Gamma_s \\ \boldsymbol{z} &= 0 && \text{on } \Gamma_w \cup \Gamma_{\text{in}} \\ \boldsymbol{T}(\boldsymbol{z},q)\cdot\boldsymbol{n} &= 0 && \text{on } \Gamma_{\text{out}} \end{aligned}$$



Layout

Shape Optimization through Adaptive SQP

Goal: Design an algorithm to adaptively build $\{\Omega_k\}_{k\geq 0}$ converging to a local minimizer Ω^* of $J(\Omega, u(\Omega))$. Use low resolution at initial stages and improve accuracy upon convergence.

First Step: Design ∞-SQP. An *ideal* Sequential Quadratic Programming algorithm assuming the PDE's can be solved exactly.

Second Step: Design a discrete counterpart of ∞-SQP, using adaptivity to improve approximation upon convergence

$\rightsquigarrow$ Adaptive Sequential Quadratic Programming (ASQP)

Shape Optimization through Adaptive SQP

Goal: Design an algorithm to adaptively build $\{\Omega_k\}_{k\geq 0}$ converging to a local minimizer Ω^* of $J(\Omega, u(\Omega))$. Use low resolution at initial stages and improve accuracy upon convergence.

First Step: Design ∞-SQP. An *ideal* Sequential Quadratic Programming algorithm assuming the PDE's can be solved exactly.

Second Step: Design a discrete counterpart of ∞-SQP, using adaptivity to improve approximation upon convergence

$\rightsquigarrow$ Adaptive Sequential Quadratic Programming (ASQP)

Shape Optimization through Adaptive SQP

Goal: Design an algorithm to adaptively build $\{\Omega_k\}_{k\geq 0}$ converging to a local minimizer Ω^* of $J(\Omega, u(\Omega))$. Use low resolution at initial stages and improve accuracy upon convergence.

First Step: Design ∞-SQP. An *ideal* Sequential Quadratic Programming algorithm assuming the PDE's can be solved exactly.

Second Step: Design a discrete counterpart of ∞-SQP, using adaptivity to improve approximation upon convergence

$$\rightsquigarrow \text{Adaptive Sequential Quadratic Programming (ASQP)}$$

Towards ∞-SQP

- $\mathbb{V}(\Gamma_k)$: Hilbert space defined on $\Gamma_k = \partial\Omega_k$ ($L^2(\Gamma_k)$, $H^1(\Gamma_k)$, etc)
- $b_k(\cdot,\cdot) : \mathbb{V}(\Gamma_k) \times \mathbb{V}(\Gamma_k) \to \mathbb{R}$ continuous and coercive bilinear form.
- Quadratic model of J at $\Omega_k \quad \rightsquigarrow \quad Q_k : \mathbb{V}(\Gamma_k) \to \mathbb{R}$

$$Q_k(w) = J(\Omega_k) + dJ(\Omega_k, w) + \frac{1}{2} b_k(w,w)$$

- Search direction $v_k :=$ minimizer of $Q_k(w)$. Hence

$$v_k \in \mathbb{V}(\Gamma_k) : \; b_k(v_k, w) = -\langle g_k, w \rangle_k , \quad \forall w \in \mathbb{V}(\Gamma_k)$$

- v_k is a descent direction

$$0 \leq b_k(v_k, v_k) = -\langle g_k, v_k \rangle_k = -dJ(\Omega_k, v_k)$$

The choice of b_k may have a *regularizing effect*

Towards ∞-SQP

- $\mathbb{V}(\Gamma_k)$: Hilbert space defined on $\Gamma_k = \partial\Omega_k$ ($L^2(\Gamma_k)$, $H^1(\Gamma_k)$, etc)
- $b_k(\cdot,\cdot) : \mathbb{V}(\Gamma_k) \times \mathbb{V}(\Gamma_k) \to \mathbb{R}$ continuous and coercive bilinear form.
- Quadratic model of J at $\Omega_k \quad \rightsquigarrow \quad Q_k : \mathbb{V}(\Gamma_k) \to \mathbb{R}$

$$Q_k(w) = J(\Omega_k) + \langle g_k, w\rangle_k + \frac{1}{2} b_k(w,w)$$

- Search direction $v_k :=$ minimizer of $Q_k(w)$. Hence

$$v_k \in \mathbb{V}(\Gamma_k) : \; b_k(v_k, w) = -\langle g_k, w\rangle_k , \quad \forall w \in \mathbb{V}(\Gamma_k)$$

- v_k is a descent direction

$$0 \leq b_k(v_k, v_k) = -\langle g_k, v_k\rangle_k = -dJ(\Omega_k, v_k)$$

The choice of b_k may have a *regularizing effect*

Towards ∞-SQP

- $\mathbb{V}(\Gamma_k)$: Hilbert space defined on $\Gamma_k = \partial\Omega_k$ ($L^2(\Gamma_k)$, $H^1(\Gamma_k)$, etc)
- $b_k(\cdot,\cdot) : \mathbb{V}(\Gamma_k) \times \mathbb{V}(\Gamma_k) \to \mathbb{R}$ continuous and coercive bilinear form.
- Quadratic model of J at $\Omega_k \quad \rightsquigarrow \quad Q_k : \mathbb{V}(\Gamma_k) \to \mathbb{R}$

$$Q_k(w) = J(\Omega_k) + \langle g_k, w\rangle_k + \frac{1}{2} b_k(w,w)$$

- Search direction $v_k :=$ minimizer of $Q_k(w)$. Hence

$$v_k \in \mathbb{V}(\Gamma_k) : \; b_k(v_k, w) = -\langle g_k, w\rangle_k , \quad \forall w \in \mathbb{V}(\Gamma_k)$$

- v_k is a descent direction

$$0 \leq b_k(v_k, v_k) = -\langle g_k, v_k\rangle_k = -dJ(\Omega_k, v_k)$$

The choice of b_k may have a *regularizing effect*

Towards ∞-SQP

- $\mathbb{V}(\Gamma_k)$: Hilbert space defined on $\Gamma_k = \partial\Omega_k$ ($L^2(\Gamma_k)$, $H^1(\Gamma_k)$, etc)
- $b_k(\cdot,\cdot) : \mathbb{V}(\Gamma_k) \times \mathbb{V}(\Gamma_k) \to \mathbb{R}$ continuous and coercive bilinear form.
- Quadratic model of J at $\Omega_k \quad \rightsquigarrow \quad Q_k : \mathbb{V}(\Gamma_k) \to \mathbb{R}$

$$Q_k(w) = J(\Omega_k) + \langle g_k, w\rangle_k + \frac{1}{2} b_k(w,w)$$

- Search direction $v_k :=$ minimizer of $Q_k(w)$. Hence

$$v_k \in \mathbb{V}(\Gamma_k) : \; b_k(v_k, w) = -\langle g_k, w\rangle_k , \quad \forall w \in \mathbb{V}(\Gamma_k) \qquad \left(\mathcal{B}_k v_k = -g_k\right)$$

- v_k is a descent direction

$$0 \leq b_k(v_k, v_k) = -\langle g_k, v_k\rangle_k = -dJ(\Omega_k, v_k)$$

The choice of b_k may have a *regularizing effect*

Towards ∞-SQP

- $\mathbb{V}(\Gamma_k)$: Hilbert space defined on $\Gamma_k = \partial\Omega_k$ ($L^2(\Gamma_k)$, $H^1(\Gamma_k)$, etc)
- $b_k(\cdot,\cdot) : \mathbb{V}(\Gamma_k) \times \mathbb{V}(\Gamma_k) \to \mathbb{R}$ continuous and coercive bilinear form.
- Quadratic model of J at $\Omega_k \quad \rightsquigarrow \quad Q_k : \mathbb{V}(\Gamma_k) \to \mathbb{R}$

$$Q_k(w) = J(\Omega_k) + \langle g_k, w\rangle_k + \frac{1}{2} b_k(w,w)$$

- Search direction $v_k :=$ minimizer of $Q_k(w)$. Hence

$$v_k \in \mathbb{V}(\Gamma_k) : \; b_k(v_k, w) = -\langle g_k, w\rangle_k , \quad \forall w \in \mathbb{V}(\Gamma_k) \qquad \left(\mathcal{B}_k v_k = -g_k\right)$$

- v_k is a descent direction

$$0 \le b_k(v_k, v_k) = -\langle g_k, v_k\rangle_k = -dJ(\Omega_k, v_k)$$

The choice of b_k may have a *regularizing effect*

∞-SQP Algorithm

Given an initial domain Ω_0, set $k = 0$ and iterate:

(a) Compute $u_k = u(\Omega_k)$ by solving $\mathcal{A}u_k = f$

(b) Compute the Riesz representation $g_k = g(\Omega_k)$ of the shape derivative $dJ(\Omega_k, \cdot)$ (solve dual problem)

(c) Compute the search direction v_k by solving

$$v_k \in \mathbb{V}_k : \qquad b_k(v_k, w) = -\langle g_k, w \rangle, \quad \forall w \in \mathbb{V}_k$$

(d) Determine an admissible stepsize μ_k satisfying

$$J(\Omega_k + \mu_k \boldsymbol{v}_k) << J(\Omega_k) \qquad \text{(line search)}$$

(e) Update $\Omega_{k+1} \leftarrow \Omega_k + \mu_k \boldsymbol{v}_k \qquad (\boldsymbol{v}_k = v_k \boldsymbol{n})$; $\qquad k \leftarrow k + 1$.

NOT REALISTIC ∞-dimensional problems

∞-SQP Algorithm

Given an initial domain Ω_0, set $k = 0$ and iterate:

(a) Compute $u_k = u(\Omega_k)$ by solving $\mathcal{A}u_k = f$

(b) Compute the Riesz representation $g_k = g(\Omega_k)$ of the shape derivative $dJ(\Omega_k, \cdot)$ (solve dual problem)

(c) Compute the search direction v_k by solving

$$v_k \in \mathbb{V}_k : \qquad b_k(v_k, w) = -\langle g_k, w \rangle, \quad \forall w \in \mathbb{V}_k$$

(d) Determine an admissible stepsize μ_k satisfying

$$J(\Omega_k + \mu_k \boldsymbol{v}_k) << J(\Omega_k) \qquad \text{(line search)}$$

(e) Update $\Omega_{k+1} \leftarrow \Omega_k + \mu_k \boldsymbol{v}_k \qquad (\boldsymbol{v}_k = v_k \boldsymbol{n})$; $\qquad k \leftarrow k + 1$.

NOT REALISTIC ∞-dimensional problems

Towards a realistic adaptive algorithm

Replace non-computable operations by adaptive finite approximations.

$\rightsquigarrow$ two main sources of error:

Adaptive approximation of PDE (PDE Error) $\rightsquigarrow$ u_k, J, dJ (g_k)

Adaptive approximation of domain (Geometric Error) $\rightsquigarrow$ v_k

Goals:

- distribute the computational effort between the two sources of error and adjust the accuracies along the iteration
  $\rightsquigarrow$ it is wasteful to impose PDE Error finer than Geometric Error
- sort out whether geometric singularities are genuine to the problem or due to lack of resolution
  $\rightsquigarrow$ correct the effects of early (no longer necessary) mesh refinements

Towards a realistic adaptive algorithm

Replace non-computable operations by adaptive finite approximations.

$\rightsquigarrow$ two main sources of error:

Adaptive approximation of PDE (PDE Error) $\rightsquigarrow$ u_k, J, dJ (g_k)

Adaptive approximation of domain (Geometric Error) $\rightsquigarrow$ v_k

Goals:

- distribute the computational effort between the two sources of error and adjust the accuracies along the iteration
  $\rightsquigarrow$ it is wasteful to impose PDE Error finer than Geometric Error
- sort out whether geometric singularities are genuine to the problem or due to lack of resolution
  $\rightsquigarrow$ correct the effects of early (no longer necessary) mesh refinements

Towards a realistic adaptive algorithm

Replace non-computable operations by adaptive finite approximations.

$\rightsquigarrow$ two main sources of error:

Adaptive approximation of PDE (PDE Error) $\rightsquigarrow$ u_k, J, dJ (g_k)

Adaptive approximation of domain (Geometric Error) $\rightsquigarrow$ v_k

Goals:

- distribute the computational effort between the two sources of error and adjust the accuracies along the iteration
  $\rightsquigarrow$ it is wasteful to impose PDE Error finer than Geometric Error
- sort out whether geometric singularities are genuine to the problem or due to lack of resolution
  $\rightsquigarrow$ correct the effects of early (no longer necessary) mesh refinements

ASQP algorithm

- $\mathcal{T}_k$: triangulation of Ω_k
- $\mathbb{S}_k = \mathbb{S}_k(\Omega_k)$: finite element space defined on Ω_k
- $\mathbb{V}_k = \mathbb{V}_k(\Gamma_k)$: finite element space defined on Γ_k
- $\mathcal{E}_k = (\Omega_k, \mathbb{S}_k, \mathbb{V}_k)$

The ASQP algorithm is an iteration of the form:

$$\mathcal{E}_k \rightarrow \texttt{APPROXJ} \rightarrow \texttt{DIRECTION} \rightarrow \texttt{LINESEARCH} \rightarrow \texttt{UPDATE} \rightarrow \mathcal{E}_{k+1}$$

Adaptivity is carried out in APPROXJ and DIRECTION

- APPROXJ acts on PDE Error
- DIRECTION acts on Geometric Error
- LINESEARCH enforces a substantial reduction of cost functional
- UPDATE $\Omega_{k+1} = \Omega_k + t d_k$

ASQP algorithm

- $\mathcal{T}_k$: triangulation of Ω_k
- $\mathbb{S}_k = \mathbb{S}_k(\Omega_k)$: finite element space defined on Ω_k
- $\mathbb{V}_k = \mathbb{V}_k(\Gamma_k)$: finite element space defined on Γ_k
- $\mathcal{E}_k = (\Omega_k, \mathbb{S}_k, \mathbb{V}_k)$

The ASQP algorithm is an iteration of the form:

$$\mathcal{E}_k \rightarrow \texttt{APPROXJ} \rightarrow \texttt{DIRECTION} \rightarrow \texttt{LINESEARCH} \rightarrow \texttt{UPDATE} \rightarrow \mathcal{E}_{k+1}$$

Adaptivity is carried out in APPROXJ and DIRECTION

- APPROXJ acts on PDE Error
- DIRECTION acts on Geometric Error
- LINESEARCH enforces a substantial reduction of cost functional
- UPDATE $\Omega_{k+1} = \Omega_k + t d_k$

ASQP algorithm

- $\mathcal{T}_k$: triangulation of Ω_k
- $\mathbb{S}_k = \mathbb{S}_k(\Omega_k)$: finite element space defined on Ω_k
- $\mathbb{V}_k = \mathbb{V}_k(\Gamma_k)$: finite element space defined on Γ_k
- $\mathcal{E}_k = (\Omega_k, \mathbb{S}_k, \mathbb{V}_k)$

The ASQP algorithm is an iteration of the form:

$$\mathcal{E}_k \rightarrow \texttt{APPROXJ} \rightarrow \texttt{DIRECTION} \rightarrow \texttt{LINESEARCH} \rightarrow \texttt{UPDATE} \rightarrow \mathcal{E}_{k+1}$$

Adaptivity is carried out in APPROXJ and DIRECTION

- APPROXJ acts on PDE Error
- DIRECTION acts on Geometric Error
- LINESEARCH enforces a substantial reduction of cost functional.
- UPDATE $\Omega_{k+1} \leftarrow \Omega_k + \mu_k V_k$

ASQP algorithm

- $\mathcal{T}_k$: triangulation of Ω_k
- $\mathbb{S}_k = \mathbb{S}_k(\Omega_k)$: finite element space defined on Ω_k
- $\mathbb{V}_k = \mathbb{V}_k(\Gamma_k)$: finite element space defined on Γ_k
- $\mathcal{E}_k = (\Omega_k, \mathbb{S}_k, \mathbb{V}_k)$

The ASQP algorithm is an iteration of the form:

$$\mathcal{E}_k \rightarrow \texttt{APPROXJ} \rightarrow \texttt{DIRECTION} \rightarrow \texttt{LINESEARCH} \rightarrow \texttt{UPDATE} \rightarrow \mathcal{E}_{k+1}$$

Adaptivity is carried out in APPROXJ and DIRECTION

- APPROXJ acts on PDE Error
- DIRECTION acts on Geometric Error
- LINESEARCH enforces a substantial reduction of cost functional.
- UPDATE $\Omega_{k+1} \leftarrow \Omega_k + \mu_k V_k$

ASQP algorithm

- $\mathcal{T}_k$: triangulation of Ω_k
- $\mathbb{S}_k = \mathbb{S}_k(\Omega_k)$: finite element space defined on Ω_k
- $\mathbb{V}_k = \mathbb{V}_k(\Gamma_k)$: finite element space defined on Γ_k
- $\mathcal{E}_k = (\Omega_k, \mathbb{S}_k, \mathbb{V}_k)$

The ASQP algorithm is an iteration of the form:

$$\mathcal{E}_k \rightarrow \texttt{APPROXJ} \rightarrow \texttt{DIRECTION} \rightarrow \texttt{LINESEARCH} \rightarrow \texttt{UPDATE} \rightarrow \mathcal{E}_{k+1}$$

Adaptivity is carried out in APPROXJ and DIRECTION

- APPROXJ acts on PDE Error
- DIRECTION acts on Geometric Error
- LINESEARCH enforces a substantial reduction of cost functional.
- UPDATE $\Omega_{k+1} \leftarrow \Omega_k + \mu_k V_k$

ASQP algorithm

- $\mathcal{T}_k$: triangulation of Ω_k
- $\mathbb{S}_k = \mathbb{S}_k(\Omega_k)$: finite element space defined on Ω_k
- $\mathbb{V}_k = \mathbb{V}_k(\Gamma_k)$: finite element space defined on Γ_k
- $\mathcal{E}_k = (\Omega_k, \mathbb{S}_k, \mathbb{V}_k)$

The ASQP algorithm is an iteration of the form:

$$\mathcal{E}_k \rightarrow \texttt{APPROXJ} \rightarrow \texttt{DIRECTION} \rightarrow \texttt{LINESEARCH} \rightarrow \texttt{UPDATE} \rightarrow \mathcal{E}_{k+1}$$

Adaptivity is carried out in APPROXJ and DIRECTION

- APPROXJ acts on PDE Error
- DIRECTION acts on Geometric Error
- LINESEARCH enforces a substantial reduction of cost functional.
- UPDATE $\Omega_{k+1} \leftarrow \Omega_k + \boldsymbol{\mu}_k V_k$

ASQP (Module APPROXJ)

APPROXJ enriches/coarsens PDE space $\mathbb{S}_k$ to control the error in the approximate functional value $J_k(\Omega_k + \mu_k V_k)$ to the prescribed tolerance ε (Goal oriented adaptivity / DWR method)

We choose $\varepsilon := \gamma \mu_k \|V_k\|^2_{\Gamma_k}$, $\gamma > 0$

$$\texttt{APPROXJ} \quad \Rightarrow \quad \left| J(\Omega_k + \mu_k \boldsymbol{V}_k) - J_k(\Omega_k + \mu_k \boldsymbol{V}_k)) \right| \leq \gamma \mu_k \|V_k\|^2_{\Gamma_k}$$

- The value of ε is not very demanding $\rightsquigarrow$ coarse meshes at the beginning and a combination of refinement and coarsening later on.
- DWR detects geometric singularities, such as corners, and refines/coarsen what is necessary for computing J accurately

Dual Weighted Residual method (DWR) [Becker, Rannacher '96 & '01]

ASQP (Module APPROXJ)

APPROXJ enriches/coarsens PDE space $\mathbb{S}_k$ to control the error in the approximate functional value $J_k(\Omega_k + \mu_k V_k)$ to the prescribed tolerance ε (Goal oriented adaptivity / DWR method)

We choose $\varepsilon := \gamma \mu_k \|V_k\|^2_{\Gamma_k}$, $\gamma > 0$

$$\texttt{APPROXJ} \quad \Rightarrow \quad \left| J(\Omega_k + \mu_k \boldsymbol{V}_k) - J_k(\Omega_k + \mu_k \boldsymbol{V}_k)) \right| \leq \gamma \mu_k \|V_k\|^2_{\Gamma_k}$$

- The value of ε is not very demanding $\rightsquigarrow$ coarse meshes at the beginning and a combination of refinement and coarsening later on.
- DWR detects geometric singularities, such as corners, and refines/coarsen what is necessary for computing J accurately

ASQP (Module APPROXJ)

APPROXJ enriches/coarsens PDE space $\mathbb{S}_k$ to control the error in the approximate functional value $J_k(\Omega_k + \mu_k V_k)$ to the prescribed tolerance ε (Goal oriented adaptivity / DWR method)

We choose $\varepsilon := \gamma \mu_k \|V_k\|^2_{\Gamma_k}$, $\gamma > 0$

$$\texttt{APPROXJ} \quad \Rightarrow \quad \left| J(\Omega_k + \mu_k \boldsymbol{V}_k) - J_k(\Omega_k + \mu_k \boldsymbol{V}_k) \right| \leq \gamma \mu_k \|V_k\|^2_{\Gamma_k}$$

- The value of ε is not very demanding $\rightsquigarrow$ coarse meshes at the beginning and a combination of refinement and coarsening later on.
- DWR detects geometric singularities, such as corners, and refines/coarsen what is necessary for computing J accurately

ASQP (Module APPROXJ)

APPROXJ enriches/coarsens PDE space $\mathbb{S}_k$ to control the error in the approximate functional value $J_k(\Omega_k + \mu_k V_k)$ to the prescribed tolerance ε (Goal oriented adaptivity / DWR method)

We choose $\varepsilon := \gamma \mu_k \|V_k\|^2_{\Gamma_k}$, $\gamma > 0$

$$\texttt{APPROXJ} \quad \Rightarrow \quad \left| J(\Omega_k + \mu_k \boldsymbol{V}_k) - J_k(\Omega_k + \mu_k \boldsymbol{V}_k) \right| \leq \gamma \mu_k \|V_k\|^2_{\Gamma_k}$$

- The value of ε is not very demanding $\rightsquigarrow$ coarse meshes at the beginning and a combination of refinement and coarsening later on.
- DWR detects geometric singularities, such as corners, and refines/coarsen what is necessary for computing J accurately

ASQP (Module DIRECTION)

- G_k approximation to shape derivative $g(\Omega_k)$
- $v_k \in \mathbb{V}(\Gamma_k)$ exact solution of $\mathcal{B}_k v_k = -G_k$
- $V_k \in \mathbb{V}(\Gamma_k)$ finite element approximation to v_k.

DIRECTION adapts Geometric space $\mathbb{V}_k$
(refine/coarsen approx. to Γ_k) s.t.

$$\|V_k - v_k\|_{\Gamma_k} \leq \theta \|V_k\|_{\Gamma_k} \qquad (\theta \leq 1/2)$$

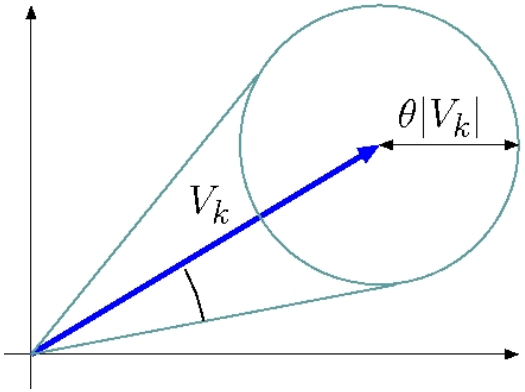
Example: $\mathcal{B}_k = -\Delta_{\Gamma_k} \Rightarrow$ Adaptivity for Laplace-Beltrami
[Demlow, Dziuk '07], [Mekchay, M., Nochetto '09]

$$\Downarrow$$

$$\begin{aligned} \left|J(\Omega_k + \mu_k \boldsymbol{V}_k) - J(\Omega_k + \mu_k \boldsymbol{v}_k)\right| &\simeq \mu_k \left|dJ(\Omega_k; \boldsymbol{V}_k - \boldsymbol{v}_k)\right| \\ &= \mu_k \left|b_k(v_k, V_k - v_k)\right| \\ &\leq \delta \mu_k \|V_k\|^2_{\Gamma_k} \qquad (\delta = M\,\theta\,(1+\theta)) \end{aligned}$$

ASQP (Module DIRECTION)

- G_k approximation to shape derivative $g(\Omega_k)$
- $v_k \in \mathbb{V}(\Gamma_k)$ exact solution of $\mathcal{B}_k v_k = -G_k$
- $V_k \in \mathbb{V}(\Gamma_k)$ finite element approximation to v_k.



DIRECTION adapts Geometric space $\mathbb{V}_k$ (refine/coarsen approx. to Γ_k) s.t.

$$\|V_k - v_k\|_{\Gamma_k} \leq \theta \|V_k\|_{\Gamma_k} \qquad (\theta \leq 1/2)$$

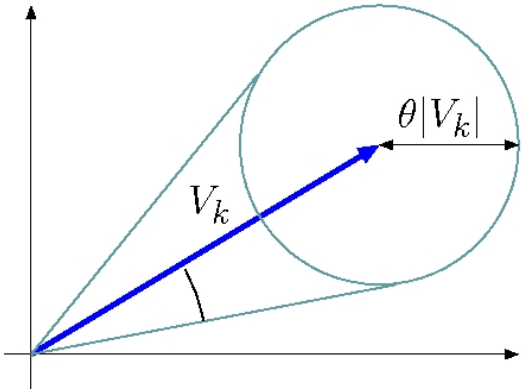
Example: $\mathcal{B}_k = -\Delta_{\Gamma_k} \Rightarrow$ Adaptivity for Laplace-Beltrami [Demlow, Dziuk '07], [Mekchay, M., Nochetto '09]

$$\Downarrow$$

$$\begin{aligned} \left|J(\Omega_k + \mu_k \boldsymbol{V}_k) - J(\Omega_k + \mu_k \boldsymbol{v}_k)\right| &\simeq \mu_k \left|dJ(\Omega_k; \boldsymbol{V}_k - \boldsymbol{v}_k)\right| \\ &= \mu_k \left|b_k(v_k, V_k - v_k)\right| \\ &\leq \delta \mu_k \|V_k\|^2_{\Gamma_k} \qquad (\delta = M\,\theta\,(1+\theta)) \end{aligned}$$

ASQP (Module DIRECTION)

- G_k approximation to shape derivative $g(\Omega_k)$
- $v_k \in \mathbb{V}(\Gamma_k)$ exact solution of $\mathcal{B}_k v_k = -G_k$
- $V_k \in \mathbb{V}(\Gamma_k)$ finite element approximation to v_k.



DIRECTION adapts Geometric space $\mathbb{V}_k$ (refine/coarsen approx. to Γ_k) s.t.

$$\|V_k - v_k\|_{\Gamma_k} \leq \theta \|V_k\|_{\Gamma_k} \qquad (\theta \leq 1/2)$$

Example: $\mathcal{B}_k = -\Delta_{\Gamma_k} \Rightarrow$ Adaptivity for Laplace-Beltrami [Demlow, Dziuk '07], [Mekchay, M., Nochetto '09]

$$\Downarrow$$

$$\begin{aligned} \left|J(\Omega_k + \mu_k \boldsymbol{V}_k) - J(\Omega_k + \mu_k \boldsymbol{v}_k)\right| &\simeq \mu_k \left|dJ(\Omega_k; \boldsymbol{V}_k - \boldsymbol{v}_k)\right| \\ &= \mu_k \left|b_k(v_k, V_k - v_k)\right| \\ &\leq \delta \mu_k \|V_k\|^2_{\Gamma_k} \qquad (\delta = M\,\theta\,(1+\theta)) \end{aligned}$$

ASQP

- APPROXJ $\Rightarrow \quad \left|J(\Omega_k + \mu_k \boldsymbol{V}_k) - J_k(\Omega_k + \mu_k \boldsymbol{V}_k))\right| \leq \gamma \mu_k \|V_k\|^2_{\Gamma_k}$
- DIRECTION $\Rightarrow \left|J(\Omega_k + \mu_k \boldsymbol{V}_k) - J(\Omega_k + \mu_k \boldsymbol{v}_k)\right| \leq \delta \mu_k \|V_k\|^2_{\Gamma_k}$
- APPROXJ + DIRECTION $\Rightarrow$ bound on local error at iteration k

$$\left|J_k(\Omega_k + \mu_k \boldsymbol{V}_k) - J(\Omega_k + \mu_k \boldsymbol{v}_k)\right| \leq \big(\gamma + \delta\big) \mu_k \|V_k\|^2_{\Gamma_k}$$

(choose $\gamma \simeq \delta \;\rightarrow$ balance computational loads)

Consistency check of ASQP

If ASQP converges to a stationary point ($\mu_k \|V_k\|^2_{\Gamma_k} \to 0$)

- DIRECTION and APPROXJ approximate the descent direction $\boldsymbol{v}_k$ and functional $J(\Omega_k)$ increasingly better
- PDE and Geometric Error progressively decrease as $k \to \infty$

ASQP

- APPROXJ $\Rightarrow \quad \left|J(\Omega_k + \mu_k \boldsymbol{V}_k) - J_k(\Omega_k + \mu_k \boldsymbol{V}_k))\right| \leq \gamma \mu_k \|V_k\|^2_{\Gamma_k}$
- DIRECTION $\Rightarrow \quad \left|J(\Omega_k + \mu_k \boldsymbol{V}_k) - J(\Omega_k + \mu_k \boldsymbol{v}_k)\right| \leq \delta \mu_k \|V_k\|^2_{\Gamma_k}$
- APPROXJ + DIRECTION $\Rightarrow$ bound on local error at iteration k

$$\left|J_k(\Omega_k + \mu_k \boldsymbol{V}_k) - J(\Omega_k + \mu_k \boldsymbol{v}_k)\right| \leq \big(\gamma + \delta\big) \mu_k \|V_k\|^2_{\Gamma_k}$$

(choose $\gamma \simeq \delta \;\rightarrow\;$ balance computational loads)

Consistency check of ASQP

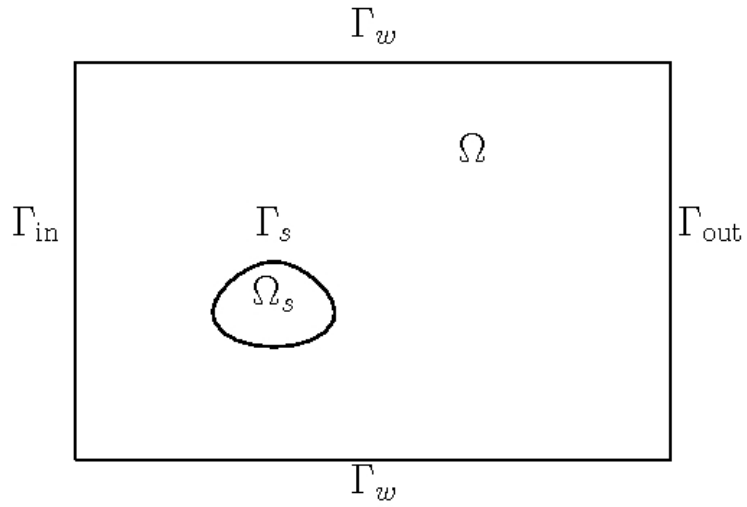
If ASQP converges to a stationary point ($\mu_k \|V_k\|^2_{\Gamma_k} \to 0$)

- DIRECTION and APPROXJ approximate the descent direction $\boldsymbol{v}_k$ and functional $J(\Omega_k)$ increasingly better
- PDE and Geometric Error progressively decrease as $k \to \infty$

Layout

Minimization of an obstacle drag

$$
\begin{aligned}
-\nabla \cdot \boldsymbol{T}(\boldsymbol{u},p) &= 0 && \text{in } \Omega \\
\nabla \cdot \boldsymbol{u} &= 0 && \text{in } \Omega \\
\boldsymbol{u} &= \boldsymbol{u}_d && \text{on } \Gamma_{\text{in}} \cup \Gamma_s \cup \Gamma_w \\
\boldsymbol{T}(\boldsymbol{u},p)\cdot \boldsymbol{n} &= 0 && \text{on } \Gamma_{\text{out}}
\end{aligned}
$$



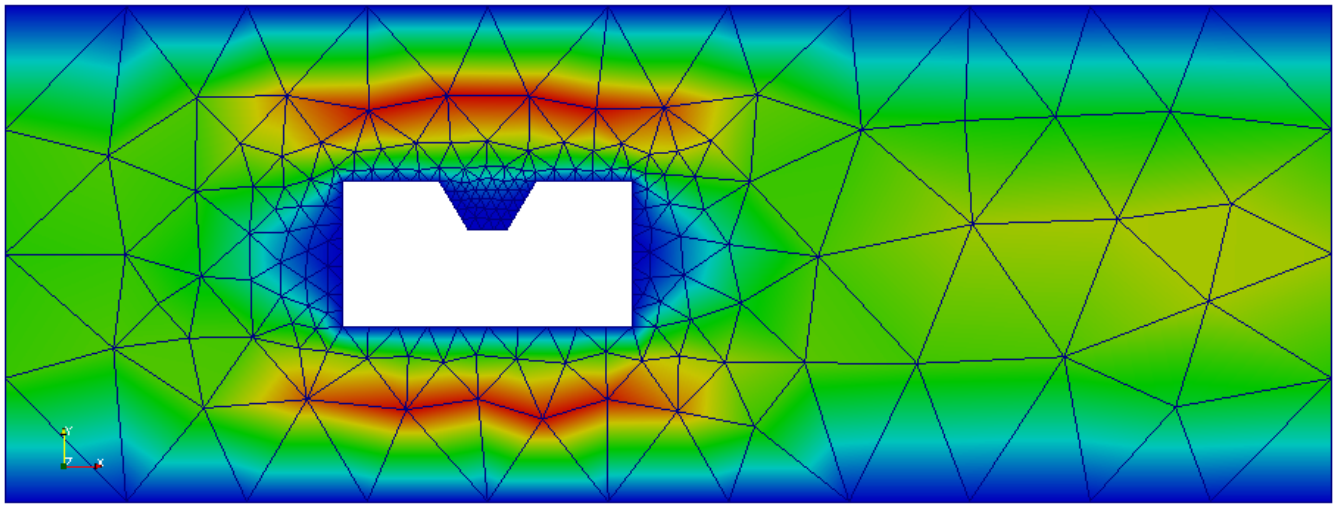
$$
\boldsymbol{T}(\boldsymbol{u},p) = 2\nu\varepsilon(\boldsymbol{u}) - p\boldsymbol{I}, \quad \varepsilon(\boldsymbol{u}) = \frac{\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T}{2}, \quad \boldsymbol{u}_d = \begin{cases} V_\infty \boldsymbol{i} & \text{on } \Gamma_{\text{in}} \\ \boldsymbol{0} & \text{on } \Gamma_w \cup \Gamma_s \end{cases}
$$

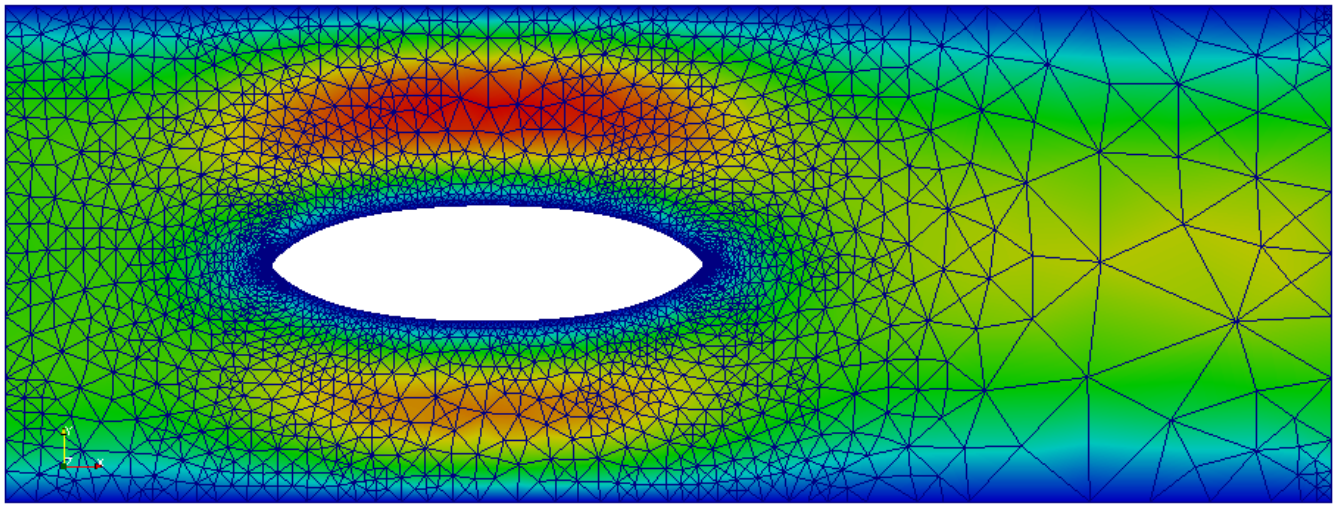
Drag Functional: $$J(\Omega,[\boldsymbol{u},p]) = -\int_{\Gamma_s} \left(\boldsymbol{T}(\boldsymbol{u},p)\,\boldsymbol{n}\right)\cdot \boldsymbol{i}\, d\Gamma$$

$$
\Omega^* \longleftarrow \min_{\Omega \in \mathcal{U}_{\text{ad}}} J(\Omega, u(\Omega)),
$$

$$
\mathcal{U}_{\text{ad}} = \{\Omega : |\Omega| = V, \text{ with } \Gamma_{\text{in}}, \Gamma_{\text{out}}, \Gamma_w \text{ fixed}\}
$$

Initial and Final Shape (DRAG minimization)





Numerical simulations $\rightsquigarrow$ toolbox ALBERTA `www.alberta-fem.de`

Further Ingredients

- Mesh quality: is maintained through an optimization routine that works locally. We try to avoid *remeshing*, but sometimes it is necessary.
- Timestep: is controlled to ensure proper reduction and to avoid *node crossing*. Remeshing helps to allow bigger timesteps.
- Volume or perimeter constraints: are handled with Lagrange multipliers and maintained up to machine precision.
- Refinement on the moving boundary: is done in a *geometrically consistent* way, to avoid overrefinement around fake corners and bad approximation of curvature.

Conclusions

- We developed an adaptive finite element method for shape optimization, based on an adaptive finite dimensional version of an ∞-dimensional SQP algorithm $\rightsquigarrow$ ASQP
- The adaptive refinement acts on two different sources of error:
  - PDE error due to discretization
  - Geometric Error due to boundary approximation
- A dynamically varying accuracy balances the computational effort between the two sources of error
- The algorithm is able to sort out whether geometric singularities are genuine to the problem or due to lack of numerical resolution, and to correct the effects of early (and no longer necessary) mesh refinements

The Module APPROXJ

$[\mathcal{T}_*, U_*, Z_*, J_*, G_*] = \texttt{APPROXJ}(\Omega, \mathcal{T}, \varepsilon)$

$\Omega_* = \Omega$

do

$[U_*, Z_*] = \texttt{SOLVE}(\Omega, \mathcal{T}_*)$

$\{\eta_*(T)\}_{T\in\mathcal{T}_*} = \texttt{ESTIMATE}(U_*, Z_*, \mathbb{S}_*)$

$[\mathcal{R}_*, \mathcal{C}_*] = \texttt{MARK}(\mathcal{T}_*, \{\eta_*(T)\}_{T\in\mathcal{T}_*})$

if $(\eta_*(\mathcal{T}_*) > \varepsilon)$

$[\mathcal{T}_*, \mathcal{C}_*] = \texttt{REFINE}(\mathcal{T}_*, \mathcal{R}_*)$

elseif $(\eta_*(\mathcal{C}_*) < \delta\varepsilon)$

$\mathcal{T}_* = \texttt{COARSEN}(\mathcal{T}_*, \mathcal{C}_*)$

endif

while $(\eta_*(\mathcal{T}_*) > \varepsilon)$

$J_* = \texttt{EVALJ}(\Omega, \mathcal{T}_*, U_*)$

$G_* = \texttt{RIESZ}(\Omega, \mathcal{T}_*, U_*, Z_*)$

SOLVE: solve the discrete system for the primal and dual variables U_*, Z_*.

The Module APPROXJ

$[\mathcal{T}_*, U_*, Z_*, J_*, G_*] = \texttt{APPROXJ}(\Omega, \mathcal{T}, \varepsilon)$

$\Omega_* = \Omega$

do

$[U_*, Z_*] = \texttt{SOLVE}(\Omega, \mathcal{T}_*)$

$\{\eta_*(T)\}_{T\in\mathcal{T}_*} = \texttt{ESTIMATE}(U_*, Z_*, \mathbb{S}_*)$

$[\mathcal{R}_*, \mathcal{C}_*] = \texttt{MARK}(\mathcal{T}_*, \{\eta_*(T)\}_{T\in\mathcal{T}_*})$

if $(\eta_*(\mathcal{T}_*) > \varepsilon)$

$[\mathcal{T}_*, \mathcal{C}_*] = \texttt{REFINE}(\mathcal{T}_*, \mathcal{R}_*)$

elseif $(\eta_*(\mathcal{C}_*) < \delta\varepsilon)$

$\mathcal{T}_* = \texttt{COARSEN}(\mathcal{T}_*, \mathcal{C}_*)$

endif

while $(\eta_*(\mathcal{T}_*) > \varepsilon)$

$J_* = \texttt{EVALJ}(\Omega, \mathcal{T}_*, U_*)$

$G_* = \texttt{RIESZ}(\Omega, \mathcal{T}_*, U_*, Z_*)$

ESTIMATE: use DWR method to compute a posteriori error estimators taylored to approximate the functional J.

The Module APPROXJ

$[\mathcal{T}_*, U_*, Z_*, J_*, G_*] = \texttt{APPROXJ}(\Omega, \mathcal{T}, \varepsilon)$

$\Omega_* = \Omega$

do

$[U_*, Z_*] = \texttt{SOLVE}(\Omega, \mathcal{T}_*)$

$\{\eta_*(T)\}_{T\in\mathcal{T}_*} = \texttt{ESTIMATE}(U_*, Z_*, \mathbb{S}_*)$

$[\mathcal{R}_*, \mathcal{C}_*] = \texttt{MARK}(\mathcal{T}_*, \{\eta_*(T)\}_{T\in\mathcal{T}_*})$

if $(\eta_*(\mathcal{T}_*) > \varepsilon)$

$[\mathcal{T}_*, \mathcal{C}_*] = \texttt{REFINE}(\mathcal{T}_*, \mathcal{R}_*)$

elseif $(\eta_*(\mathcal{C}_*) < \delta\varepsilon)$

$\mathcal{T}_* = \texttt{COARSEN}(\mathcal{T}_*, \mathcal{C}_*)$

endif

while $(\eta_*(\mathcal{T}_*) > \varepsilon)$

$J_* = \texttt{EVALJ}(\Omega, \mathcal{T}_*, U_*)$

$G_* = \texttt{RIESZ}(\Omega, \mathcal{T}_*, U_*, Z_*)$

MARK: use *maximum strategy*. Given $0 < \delta^- \ll \delta^+ < 1$, let $\eta_* = \max_{T\in\mathcal{T}_*} \eta_*(T)$ and

$$\eta_*(T) > \delta^+\eta_* \quad \Rightarrow \quad T \in \mathcal{R}_*, \qquad\qquad \eta_*(T) < \delta^-\eta_* \quad \Rightarrow \quad T \in \mathcal{C}_*.$$

The Module APPROXJ

$[\mathcal{T}_*, U_*, Z_*, J_*, G_*] = \texttt{APPROXJ}(\Omega, \mathcal{T}, \varepsilon)$

$\Omega_* = \Omega$

do

$[U_*, Z_*] = \texttt{SOLVE}(\Omega, \mathcal{T}_*)$

$\{\eta_*(T)\}_{T\in\mathcal{T}_*} = \texttt{ESTIMATE}(U_*, Z_*, \mathbb{S}_*)$

$[\mathcal{R}_*, \mathcal{C}_*] = \texttt{MARK}(\mathcal{T}_*, \{\eta_*(T)\}_{T\in\mathcal{T}_*})$

if $(\eta_*(\mathcal{T}_*) > \varepsilon)$

$[\mathcal{T}_*, \mathcal{C}_*] = \texttt{REFINE}(\mathcal{T}_*, \mathcal{R}_*)$

elseif $(\eta_*(\mathcal{C}_*) < \delta\varepsilon)$

$\mathcal{T}_* = \texttt{COARSEN}(\mathcal{T}_*, \mathcal{C}_*)$

endif

while $(\eta_*(\mathcal{T}_*) > \varepsilon)$

$J_* = \texttt{EVALJ}(\Omega, \mathcal{T}_*, U_*)$

$G_* = \texttt{RIESZ}(\Omega, \mathcal{T}_*, U_*, Z_*)$

REFINE/COARSEN: Refine the marked elements (in $\mathcal{R}_*$) using a Geometrically Consistent Refinement algorithm on the boundary; coarsen the elements in $\mathcal{C}_*$.

DWR for Drag functional

The following DWR error estimate holds

$$|J(\boldsymbol{u},p)-J(\boldsymbol{U},P)| \leq \sum_T \left\{ \|r(\boldsymbol{U},P)\|_T \|\boldsymbol{z}-\boldsymbol{Z}\|_T + \|j(\boldsymbol{U},P)\|_{\partial T} \|\boldsymbol{z}-\boldsymbol{Z}\|_{\partial T} \right\} + \sum_T \|\mathcal{R}(\boldsymbol{U})\|_T \|q-Q\|_T,$$

with

$$r(\boldsymbol{U})|_T := -\nabla\cdot(2\nu\varepsilon(\boldsymbol{U})-P\mathbf{I}) \qquad \mathcal{R}|_T := \nabla\cdot\boldsymbol{U},$$

$$j(\boldsymbol{U})|_e = \begin{cases} \frac{1}{2}[2\nu\varepsilon(\boldsymbol{U})\cdot\boldsymbol{n}-P\boldsymbol{n}] & e\cap\partial\Omega=\emptyset \\ 2\nu\varepsilon(\boldsymbol{U})\cdot\boldsymbol{n}-P\boldsymbol{n} & e\subset\Gamma_{out} \\ 0 & \text{otherwise,} \end{cases}$$

SQP with constraint

- Quadratic model $Q(w) := J(\Omega) + \langle g, w\rangle + \frac{1}{2} b_\Gamma(w,w)$
- Volume $C(\Omega) = \int_\Omega dx \quad \rightsquigarrow \quad dC(\Omega; w) = \langle 1, w\rangle$
- Lagrangian

$$\mathcal{L}(w,\lambda) = Q(w) + \lambda(C(\Omega + w) - VOL) \simeq Q(w) + \lambda(dC(\Omega, w))$$

- Find (v, λ) such that

$$\begin{aligned} \nabla_w Q(v) + \lambda &= 0 \\ C(\Omega + v\boldsymbol{n}) - VOL &= 0 \end{aligned}$$

- $v = v_0 + \lambda v_1$, with $\mathcal{B} v_0 = -g$ and $\mathcal{B} v_1 = -1$
- Apply Newton's method to $c(\lambda) := \int_{\Omega + \mu(v_0 \boldsymbol{n} + \lambda v_1)} dx - VOL = 0$

The Module LINESEARCH

$[\Omega_*, \mathcal{T}_*, \mu] = \texttt{LINESEARCH}(\Omega, \mathcal{T}, \boldsymbol{V}, J, G, m)$

$\mu = 2m\,, \;\; \varphi = J\,, \;\; \psi = \langle G, V \rangle_\Gamma$

do

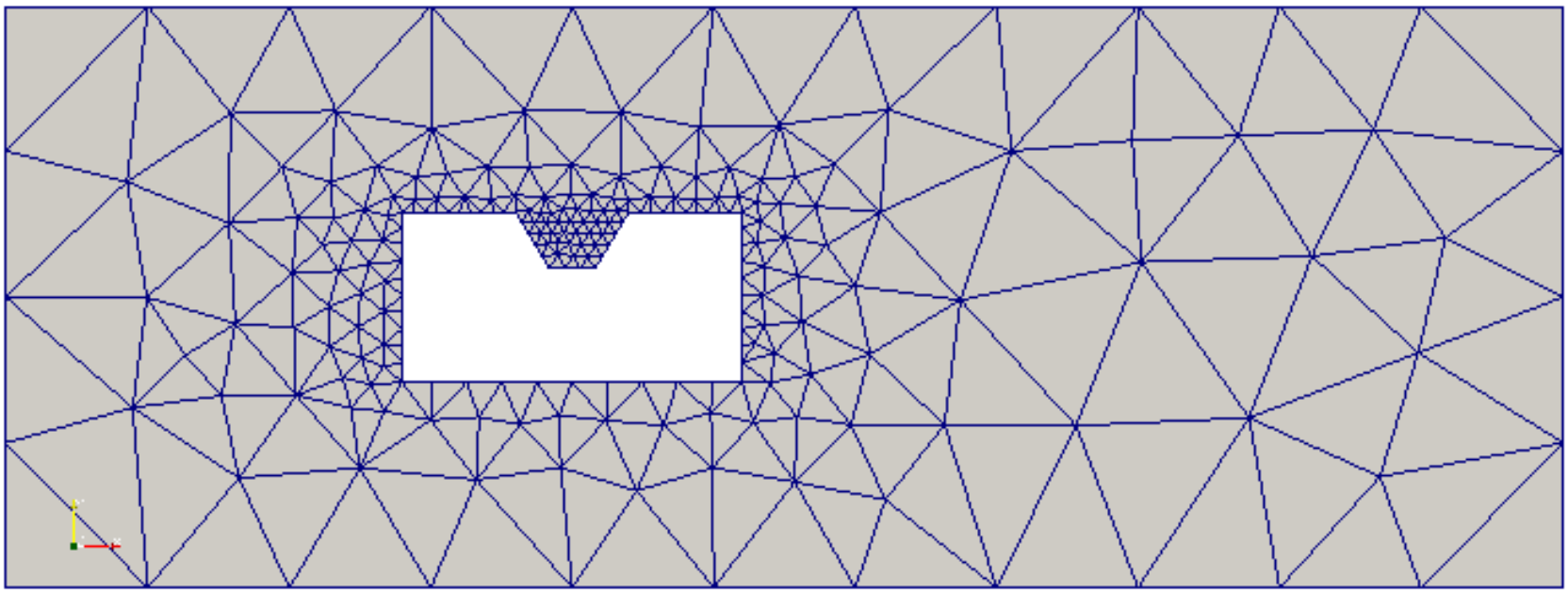
$\mu \leftarrow \mu/2;$
$[\Omega_*, \mathcal{T}_*] = \texttt{UPDATE}(\Omega, \mathcal{T}, \boldsymbol{V}, \mu)$
$[U_*, Z_*] = \texttt{SOLVE}(\Omega_*, \mathcal{T}_*)$
$\varphi_* = \texttt{EVALJ}(\Omega_*, \mathcal{T}_*, U_*)$

while $(\varphi_* > \varphi + \alpha\mu\psi)$

Evolution of meshes (DRAG minimization)

go back

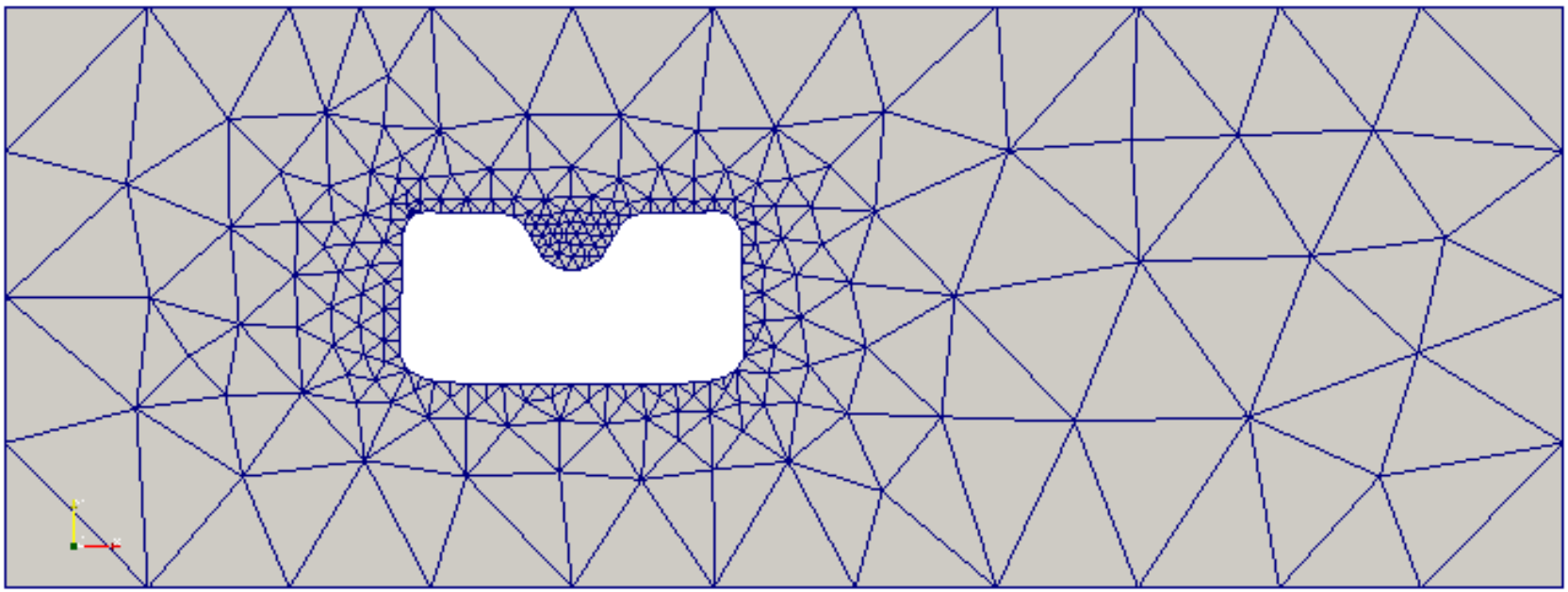
Initial Mesh



Evolution of meshes (DRAG minimization)

go back

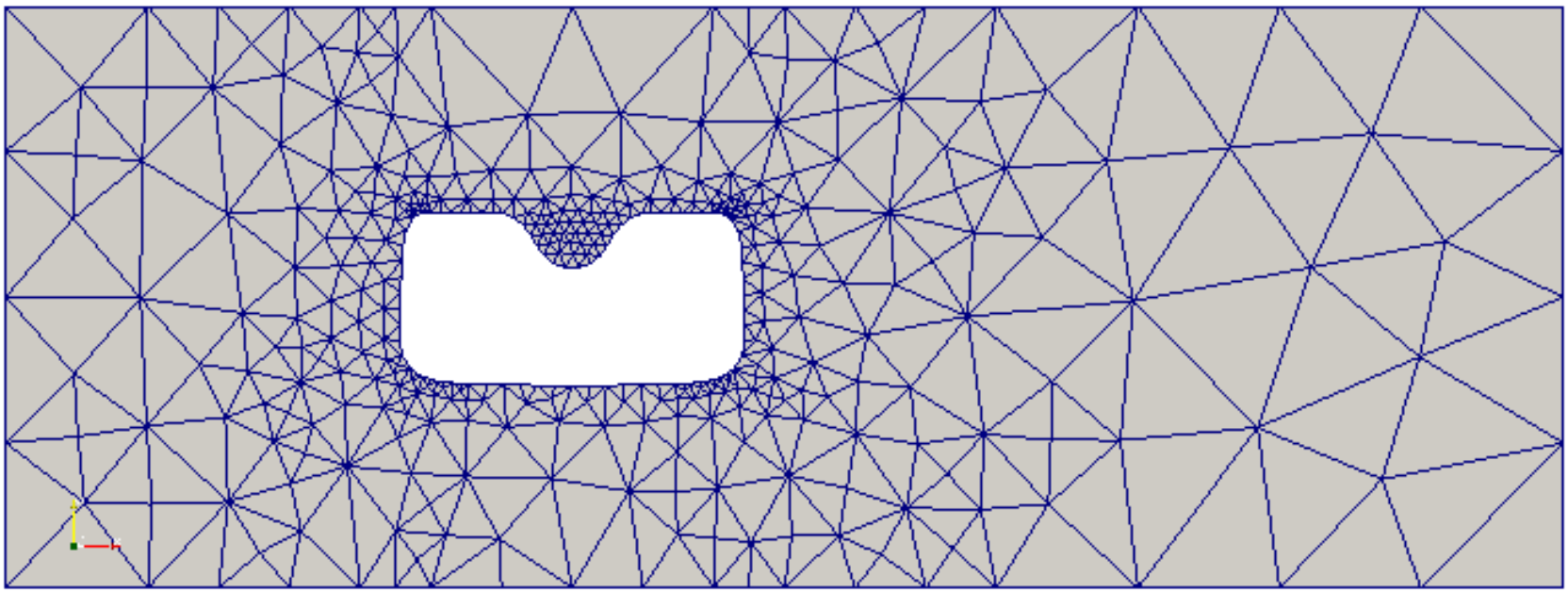
Mesh after 1 iteration



Evolution of meshes (DRAG minimization)

go back

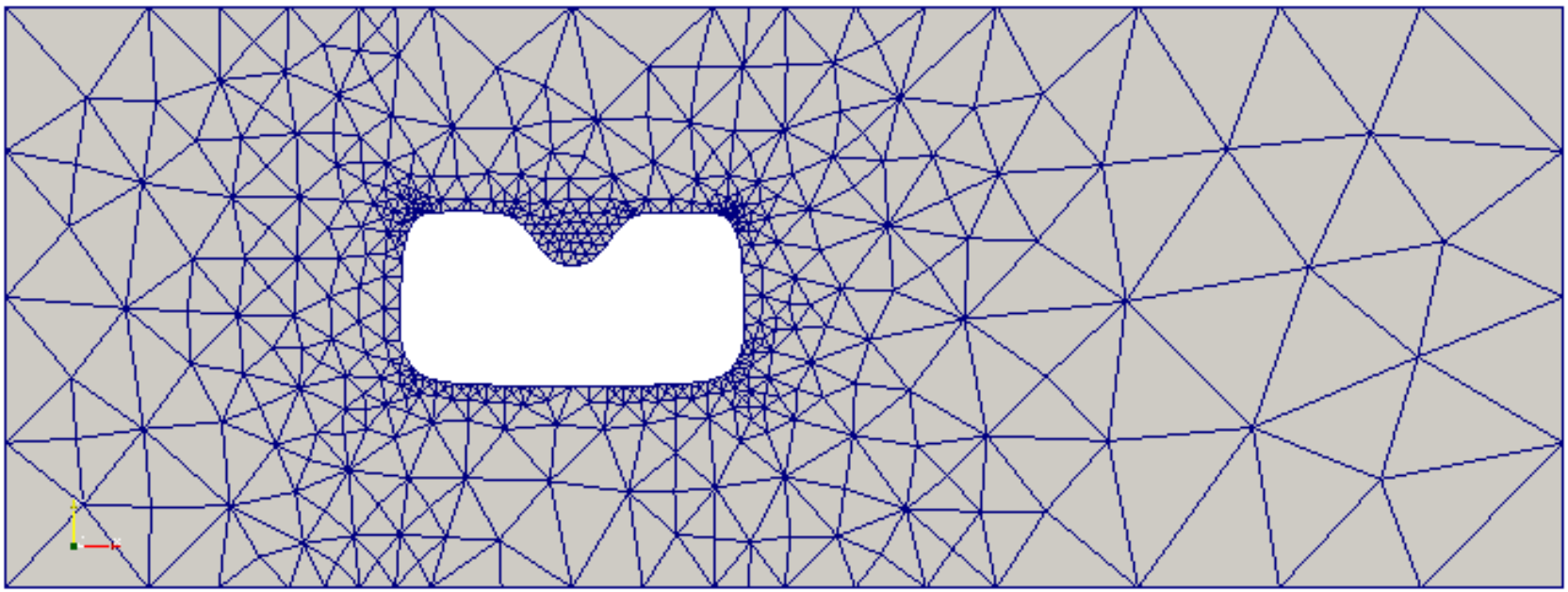
Mesh after 2 iterations



Evolution of meshes (DRAG minimization)

go back

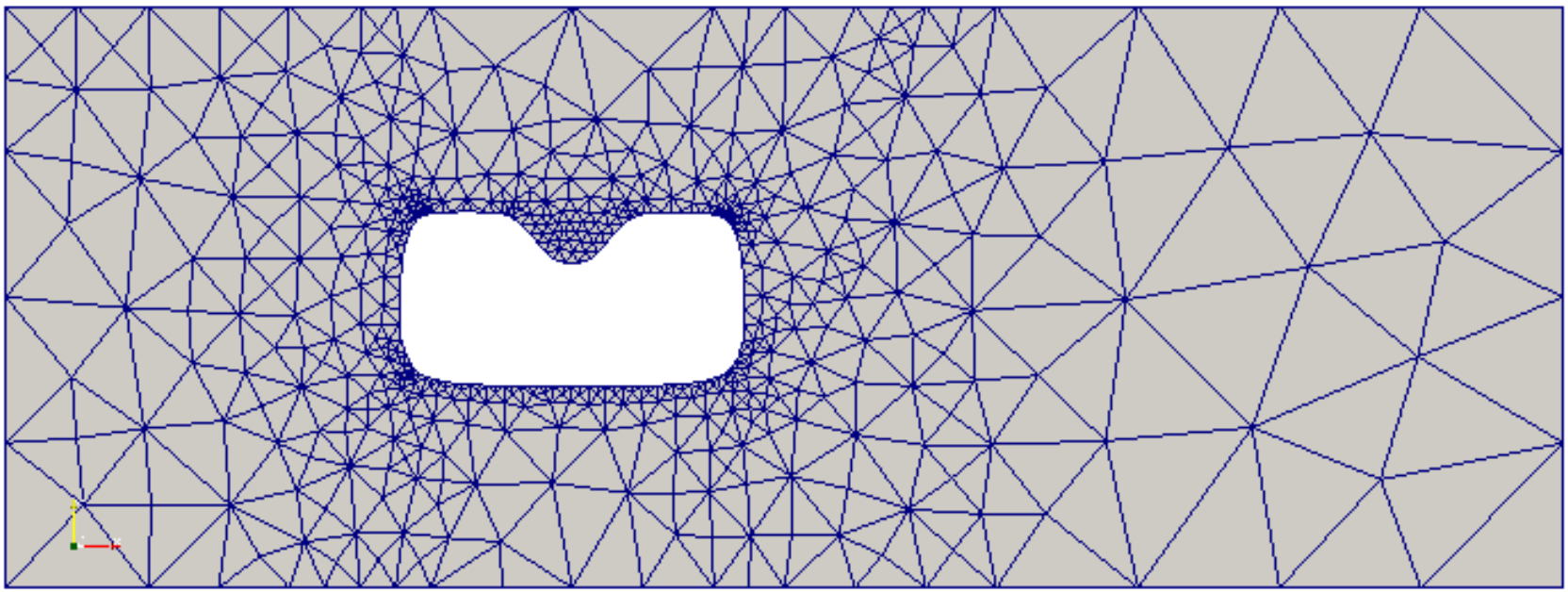
Mesh after 3 iterations



Evolution of meshes (DRAG minimization)

go back

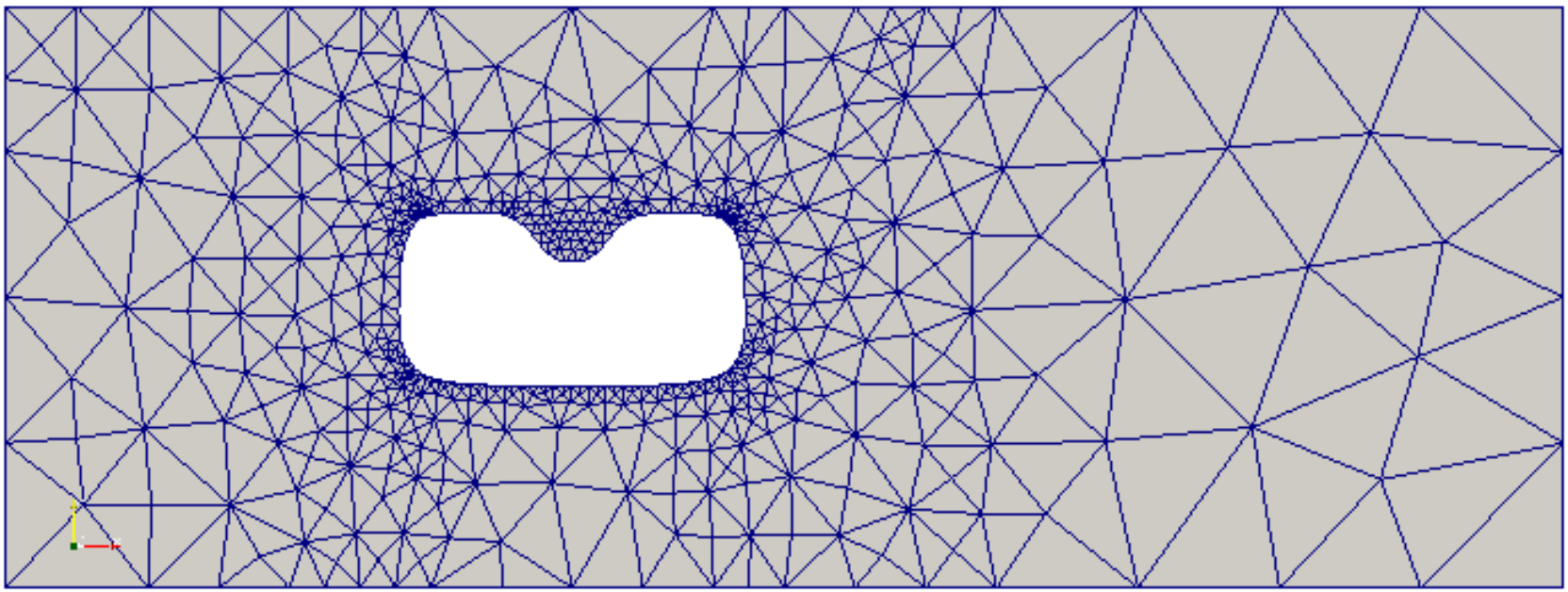
Mesh after 4 iterations



Evolution of meshes (DRAG minimization)

go back

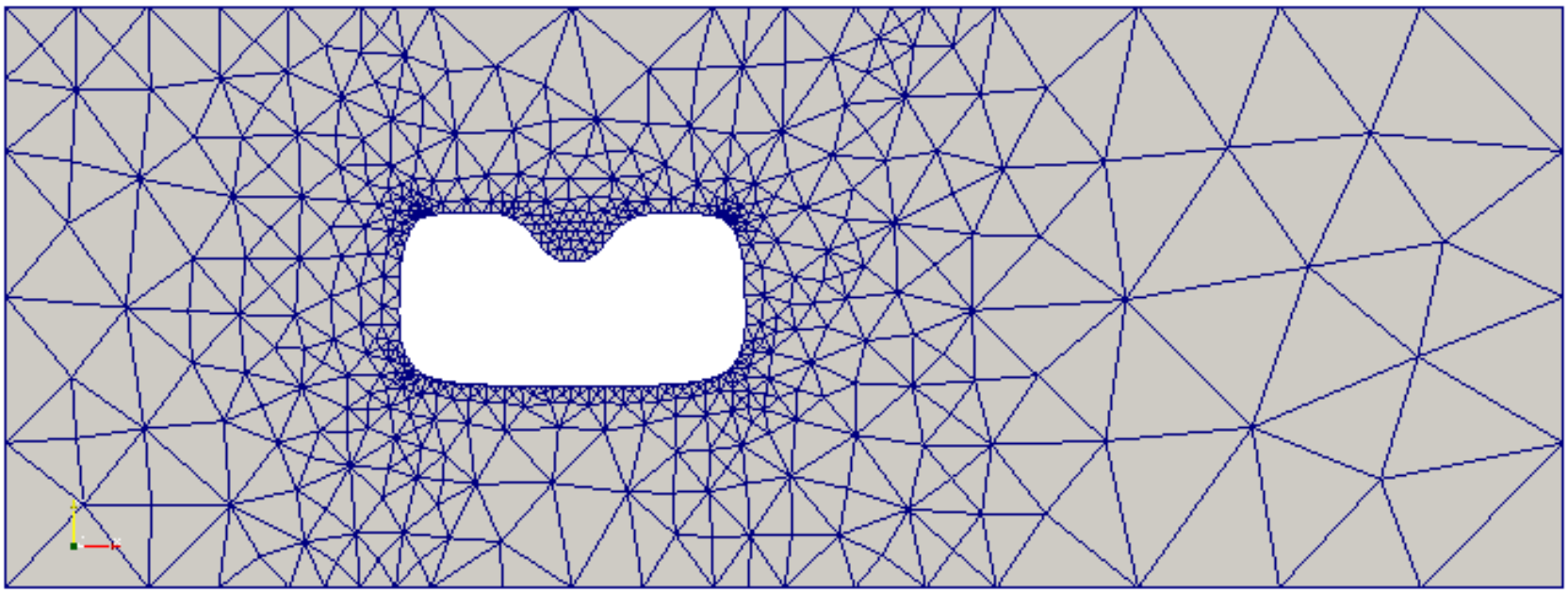
Mesh after 5 iterations



Evolution of meshes (DRAG minimization)

go back

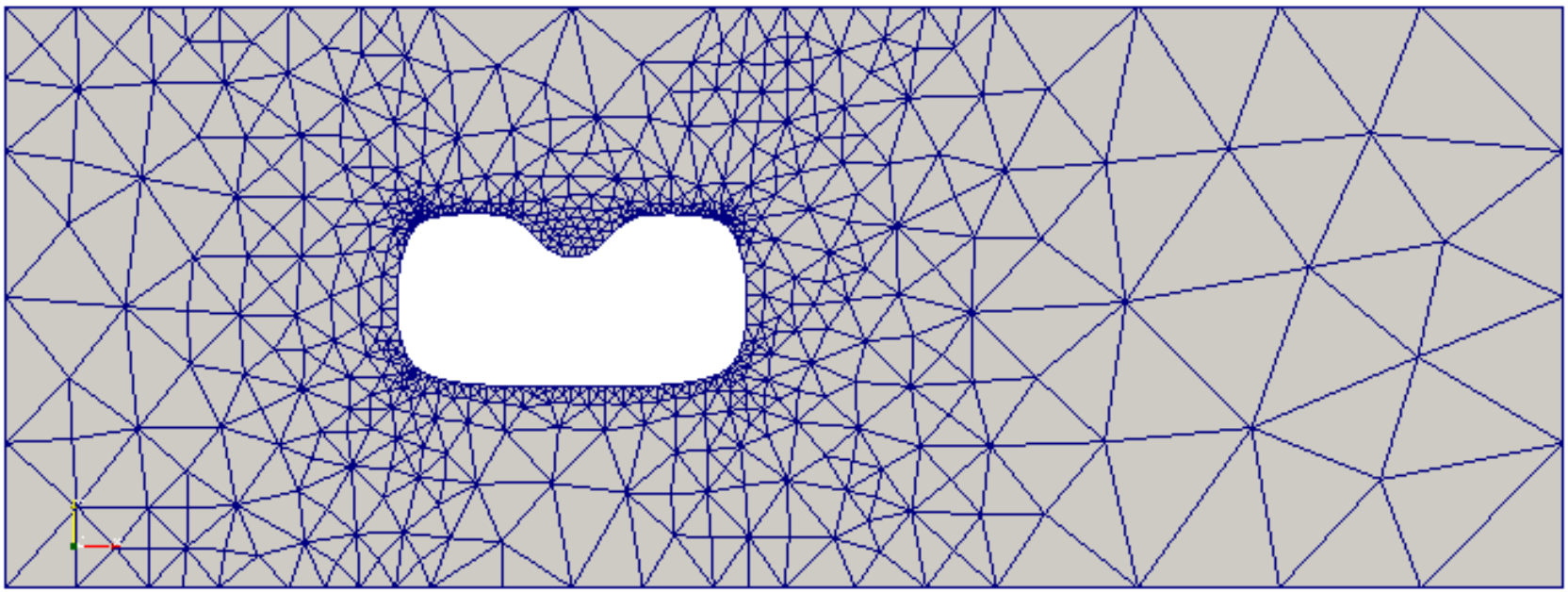
Mesh after 6 iterations



Evolution of meshes (DRAG minimization)

go back

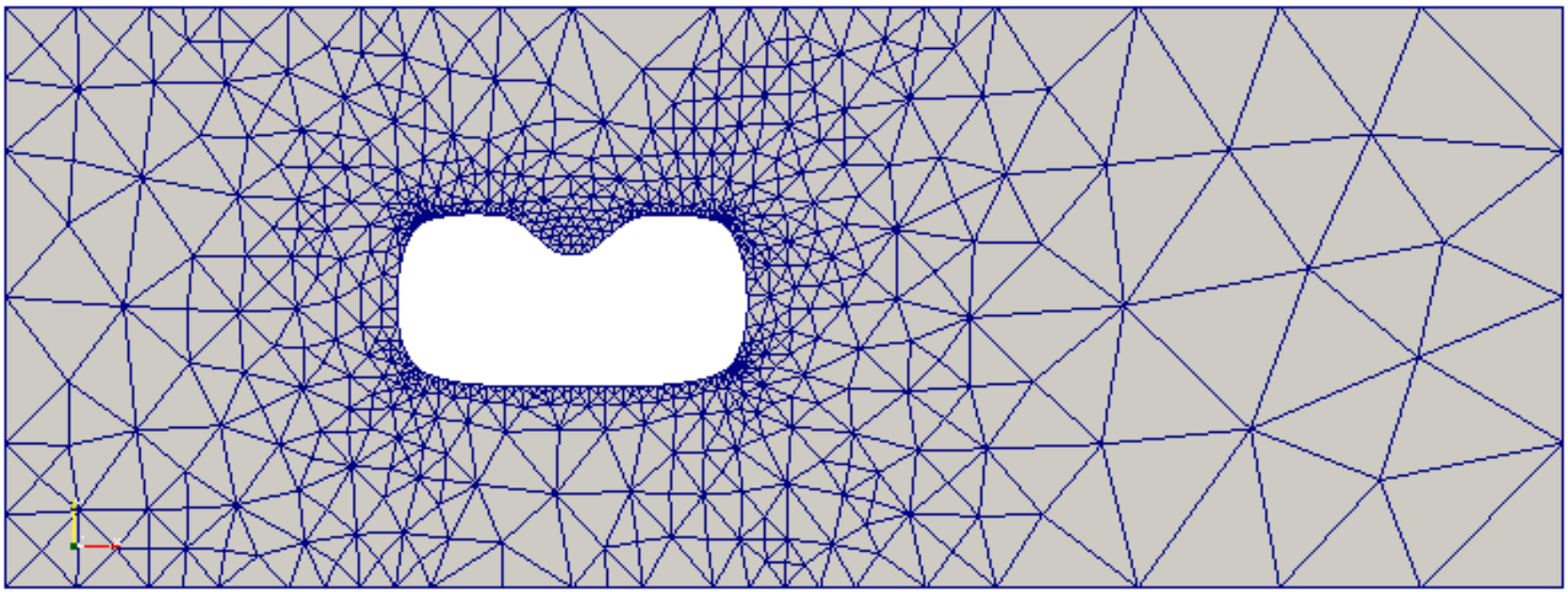
Mesh after 9 iterations



Evolution of meshes (DRAG minimization)

go back

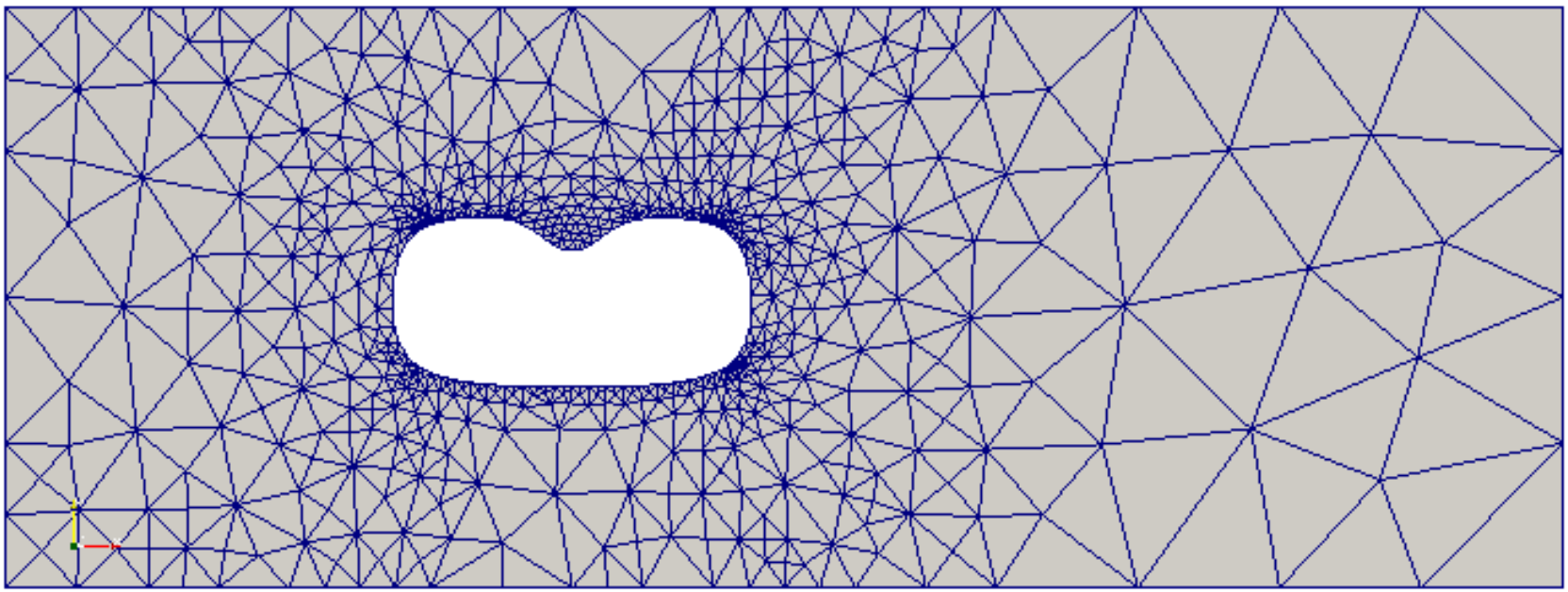
Mesh after 10 iterations



Evolution of meshes (DRAG minimization)

go back

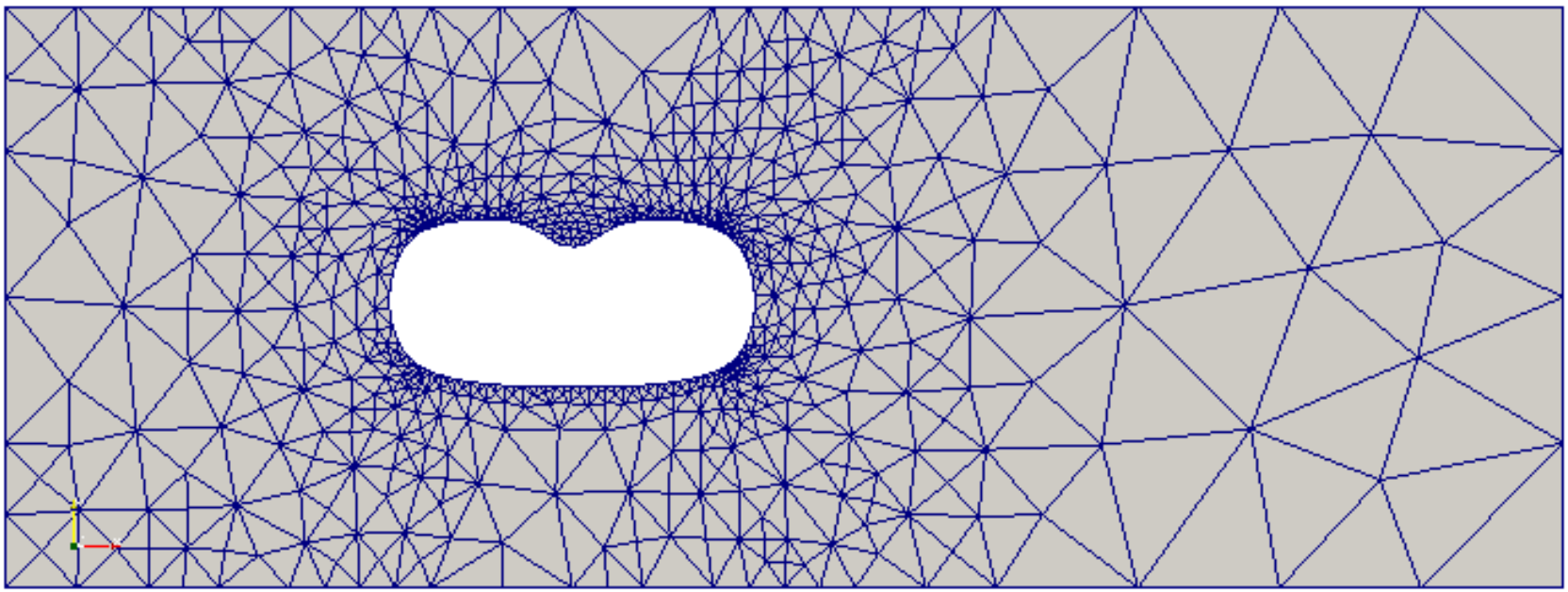
Mesh after 15 iterations



Evolution of meshes (DRAG minimization)

go back

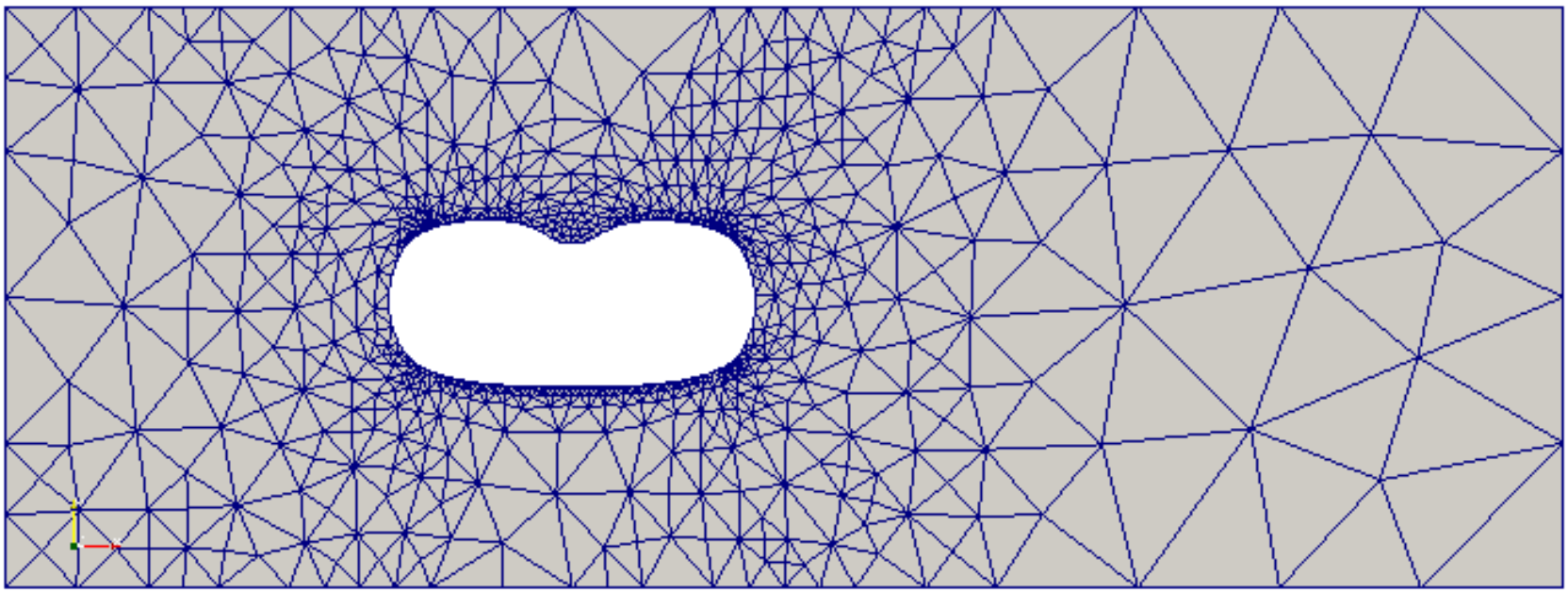
Mesh after 16 iterations



Evolution of meshes (DRAG minimization)

go back

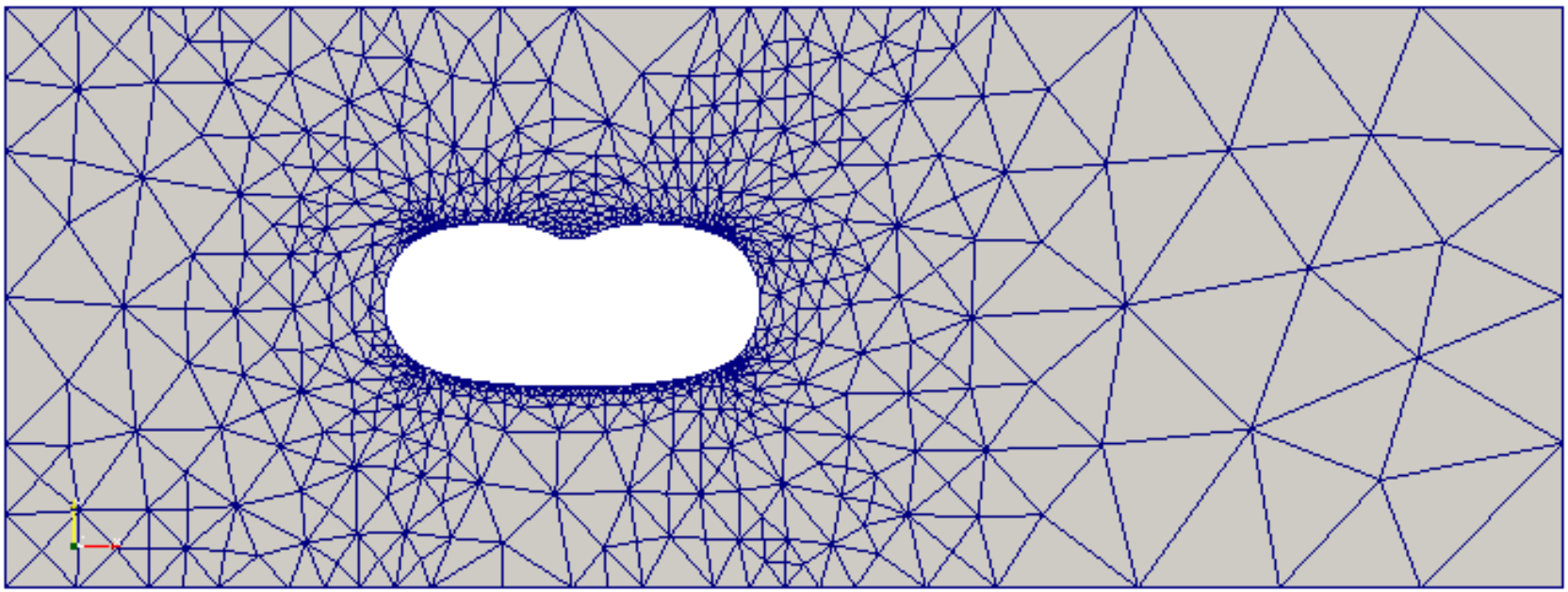
Mesh after 17 iterations



Evolution of meshes (DRAG minimization)

go back

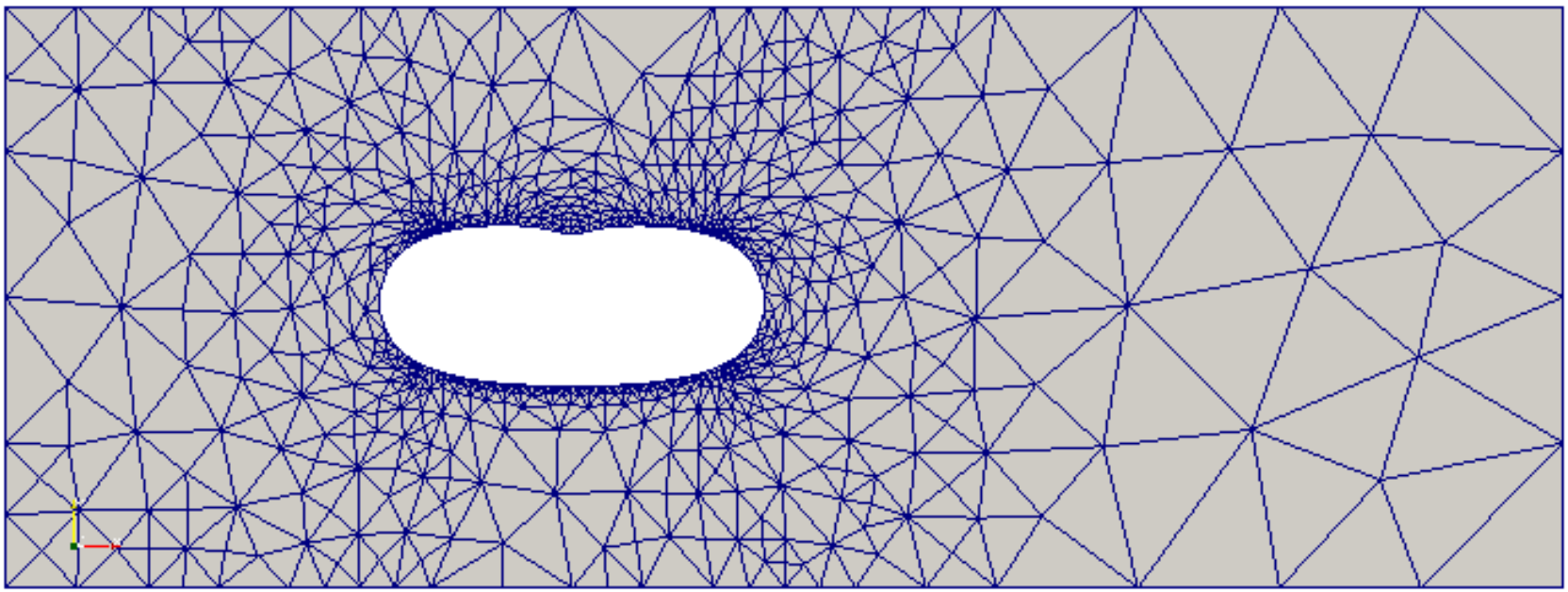
Mesh after 20 iterations



Evolution of meshes (DRAG minimization)

go back

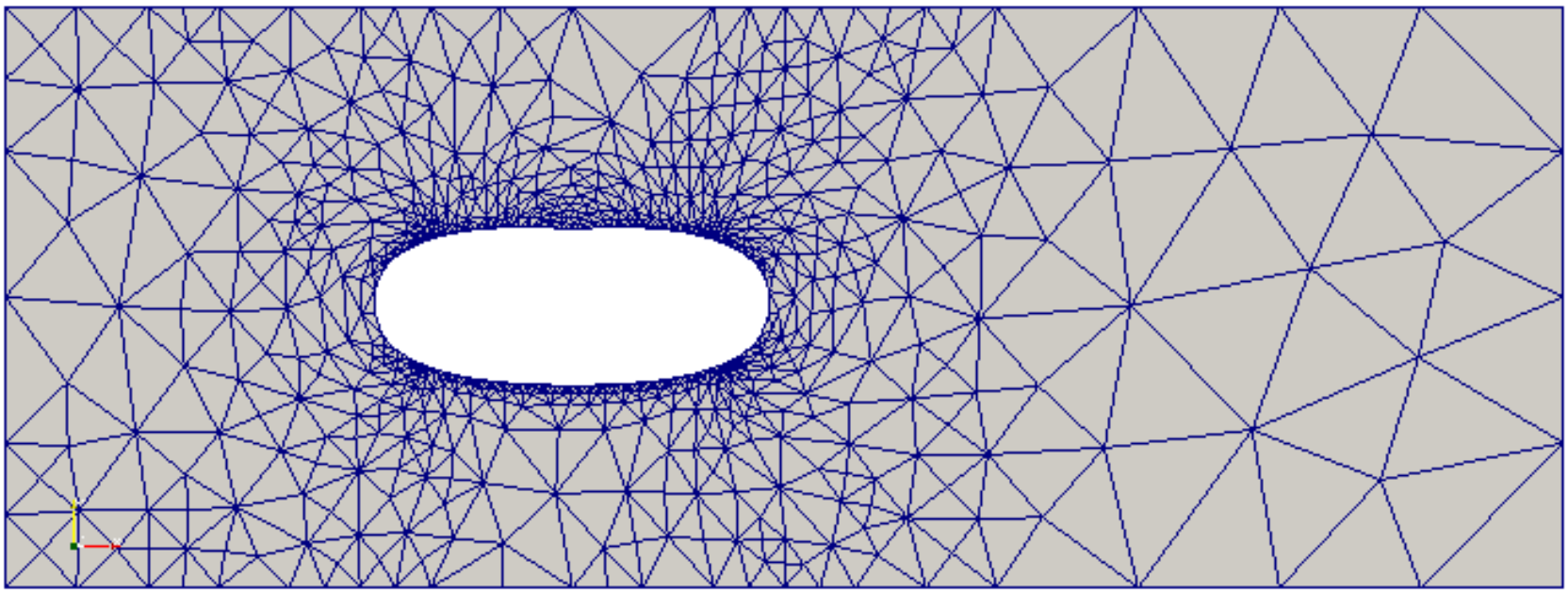
Mesh after 22 iterations



Evolution of meshes (DRAG minimization)

go back

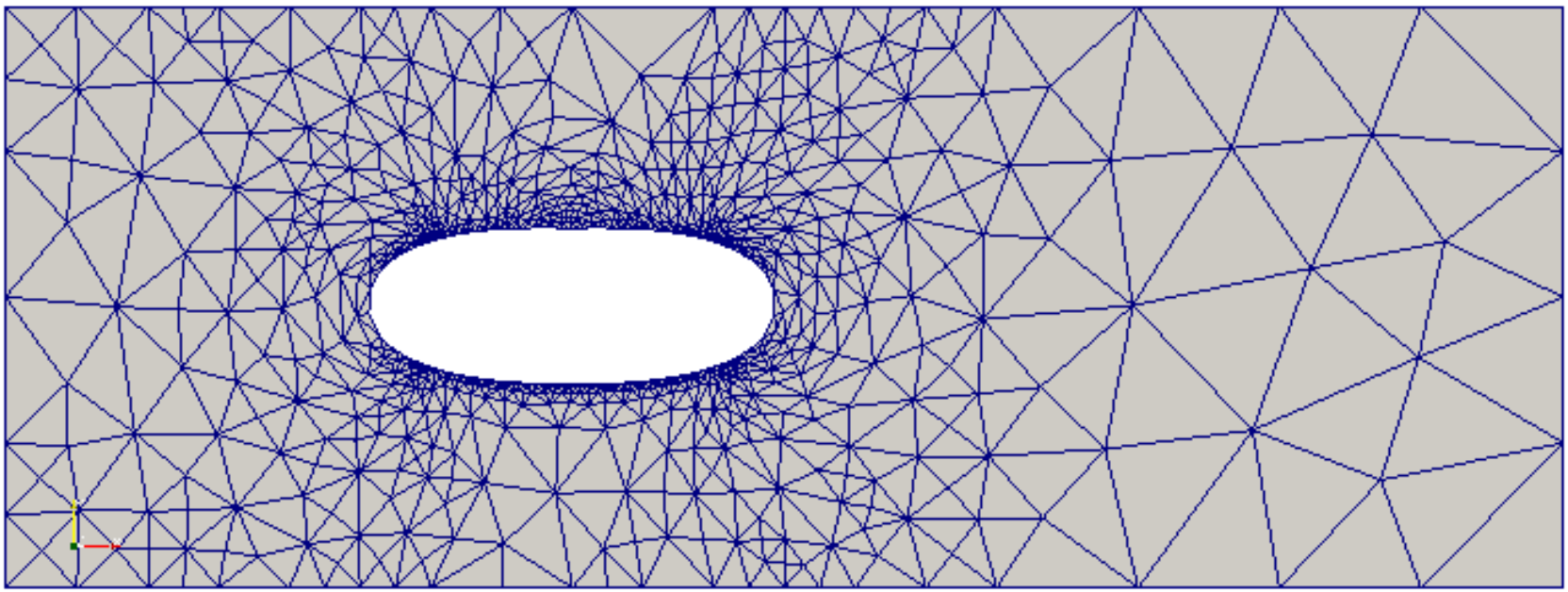
Mesh after 25 iterations



Evolution of meshes (DRAG minimization)

go back

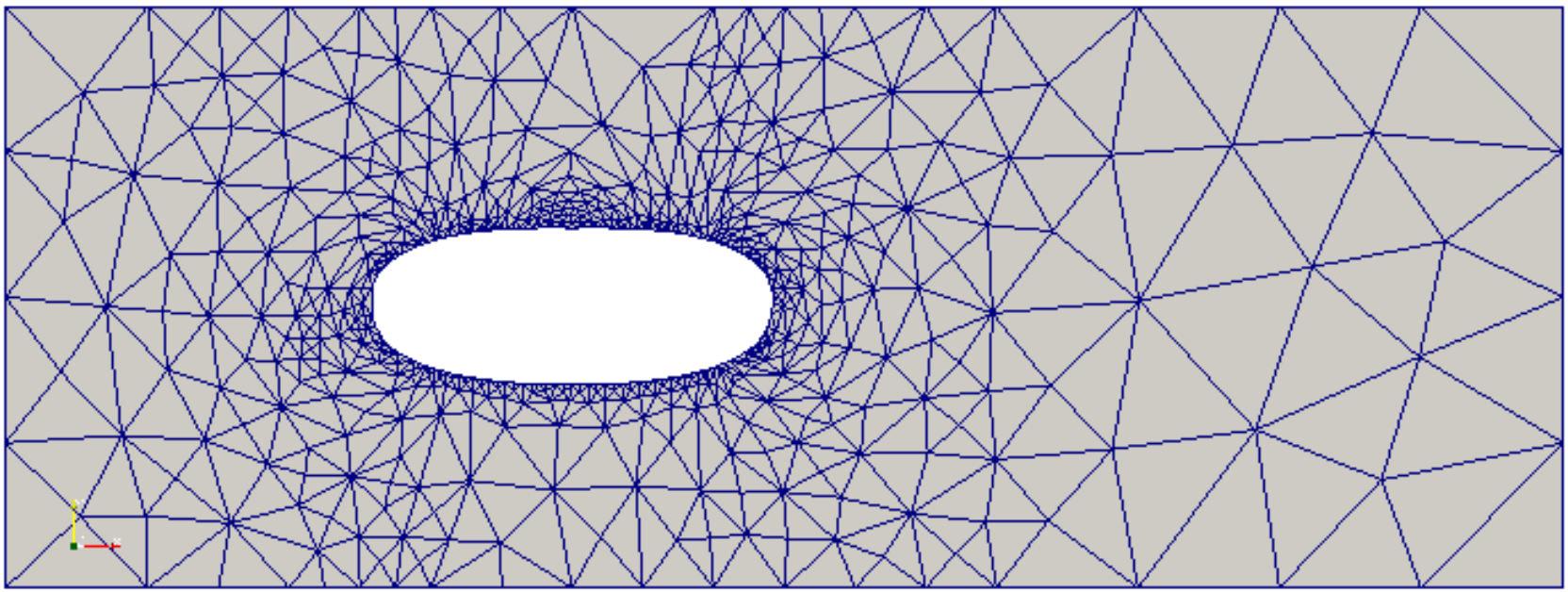
Mesh after 27 iterations



Evolution of meshes (DRAG minimization)

go back

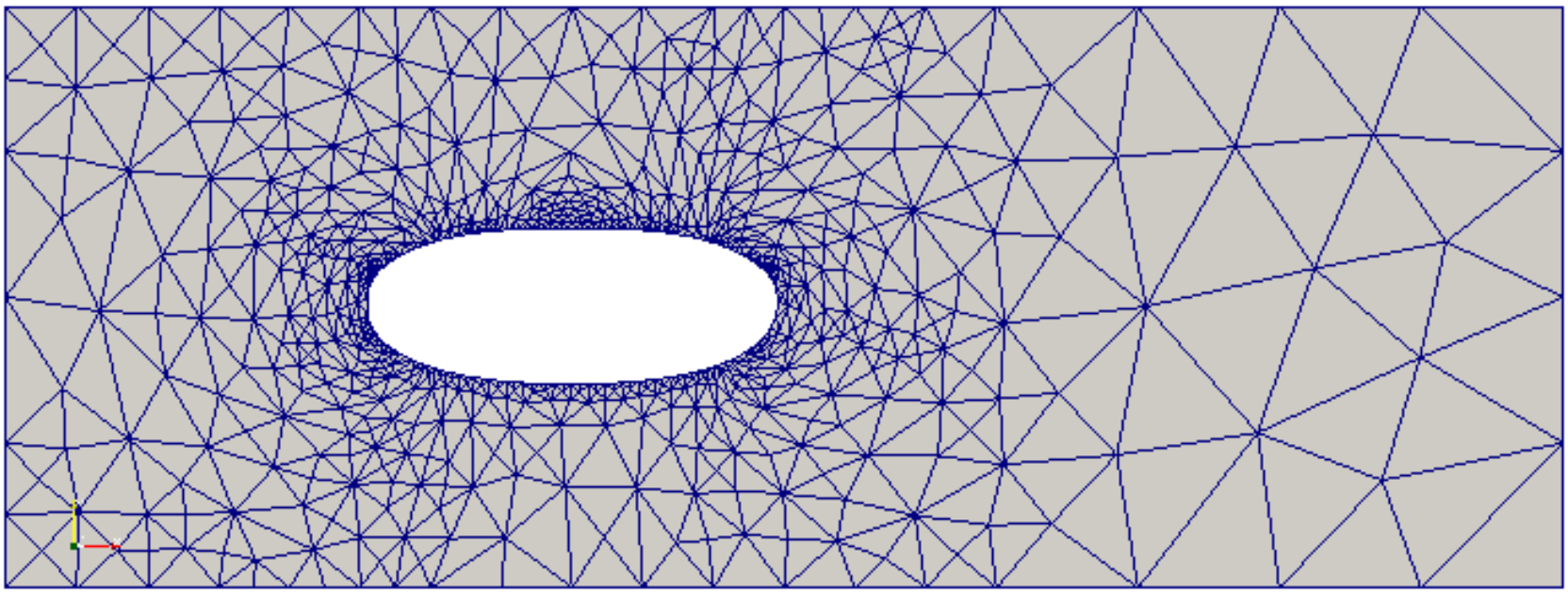
Mesh after 28 iterations



Evolution of meshes (DRAG minimization)

go back

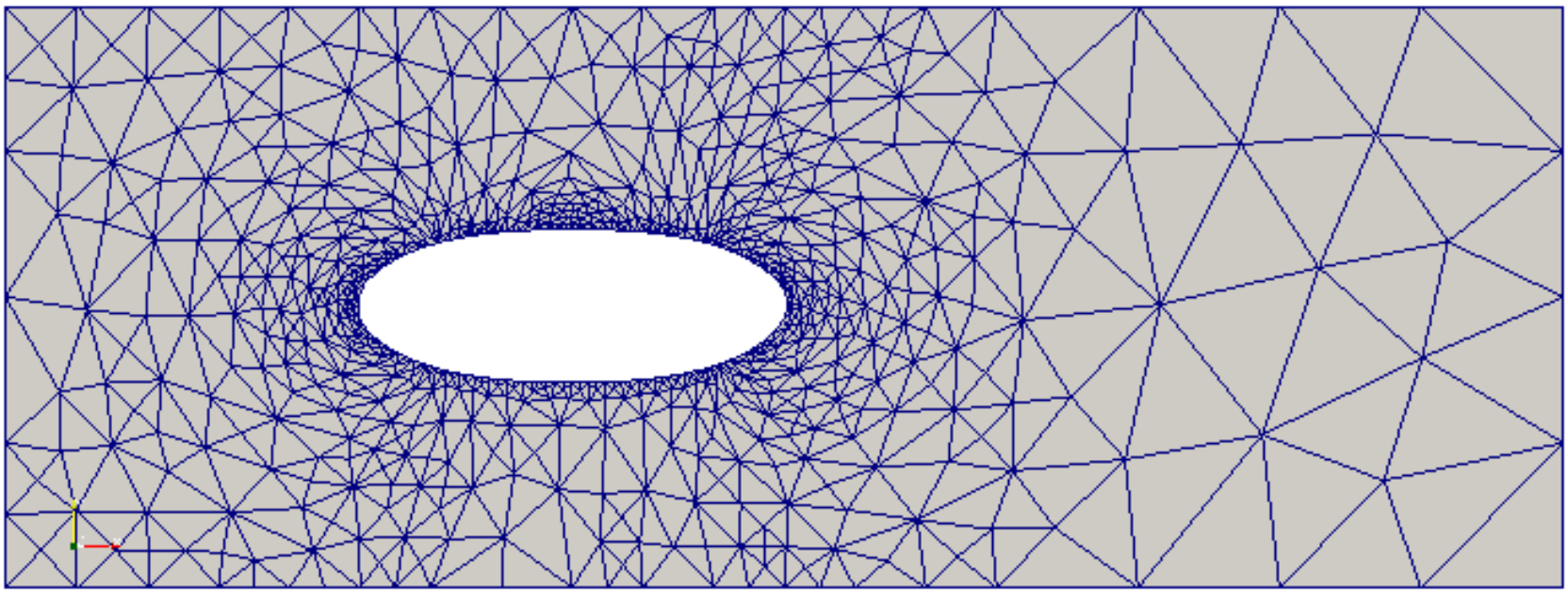
Mesh after 30 iterations



Evolution of meshes (DRAG minimization)

go back

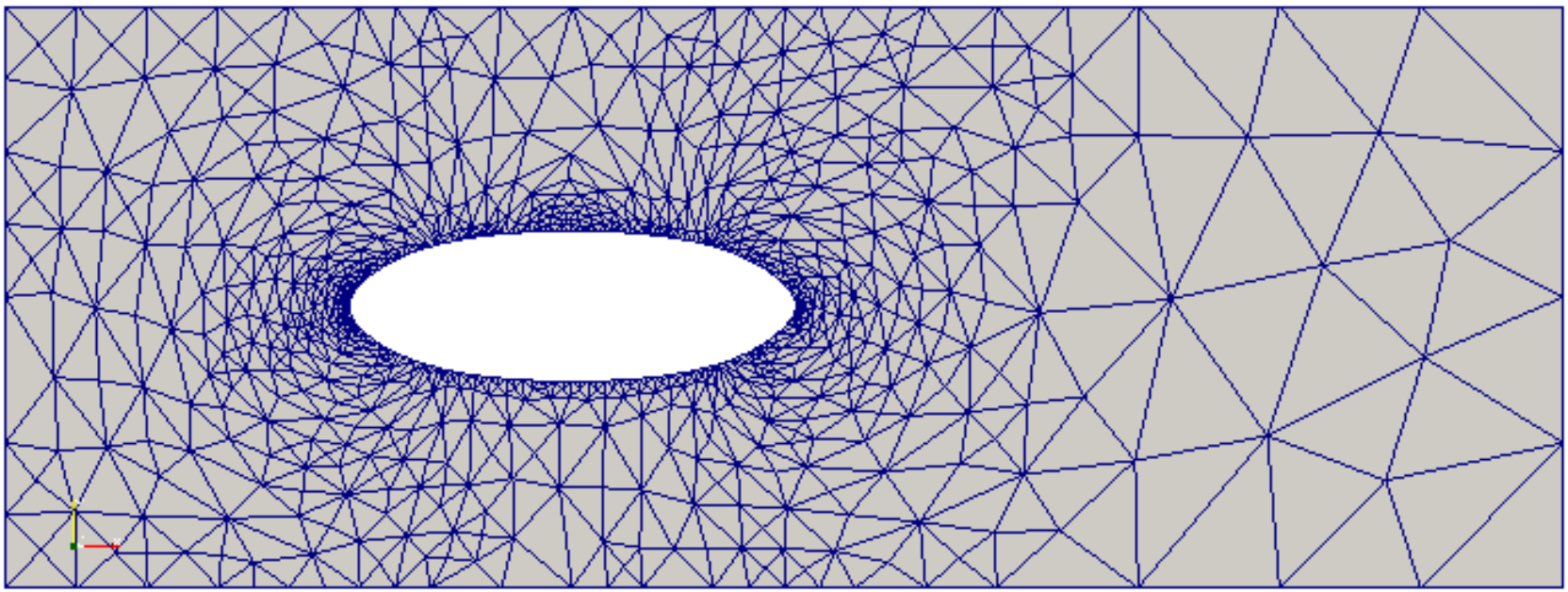
Mesh after 32 iterations



Evolution of meshes (DRAG minimization)

go back

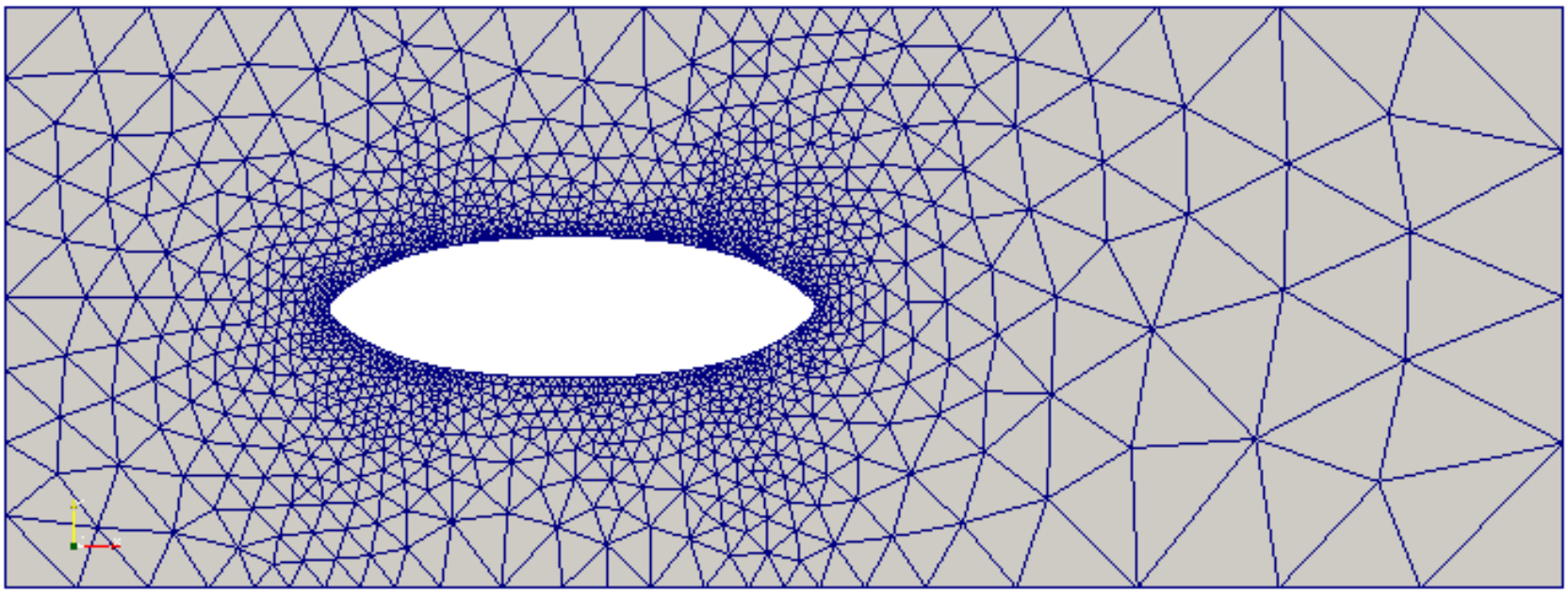
Mesh after 34 iterations



Evolution of meshes (DRAG minimization)

go back

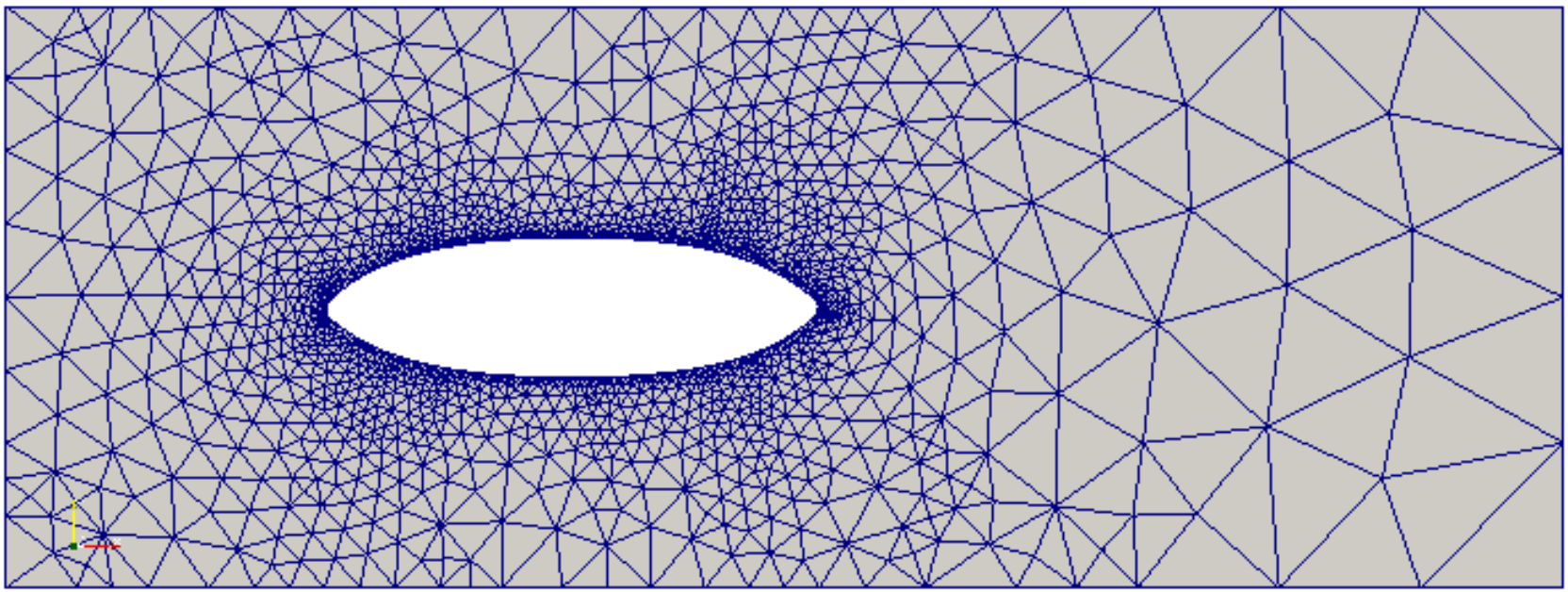
Mesh after 40 iterations



Evolution of meshes (DRAG minimization)

go back

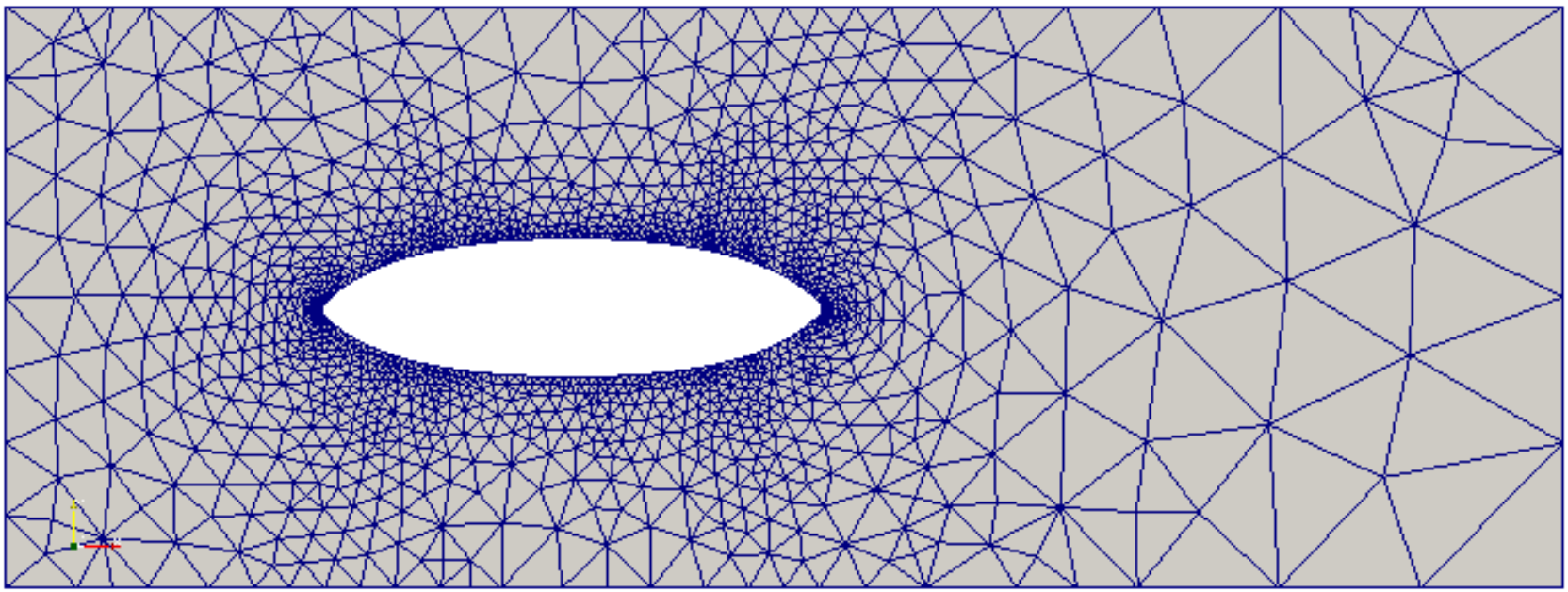
Mesh after 42 iterations



Evolution of meshes (DRAG minimization)

go back

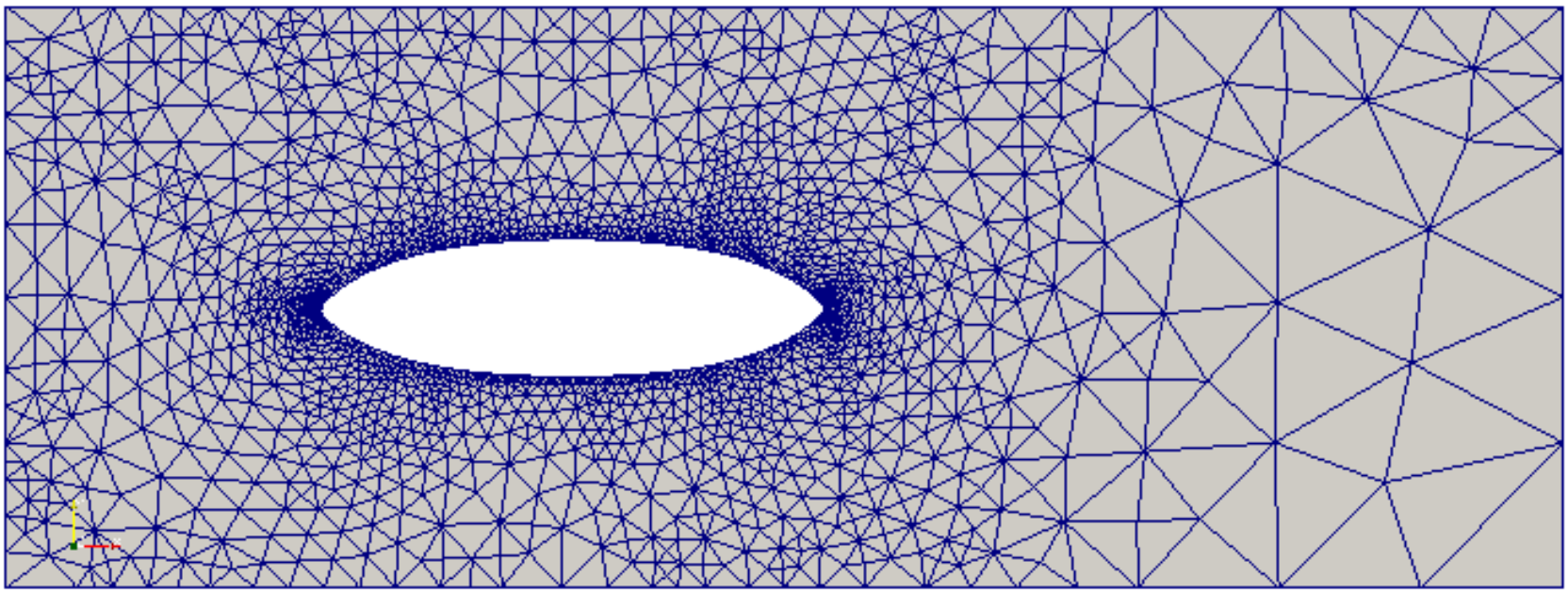
Mesh after 45 iterations



Evolution of meshes (DRAG minimization)

go back

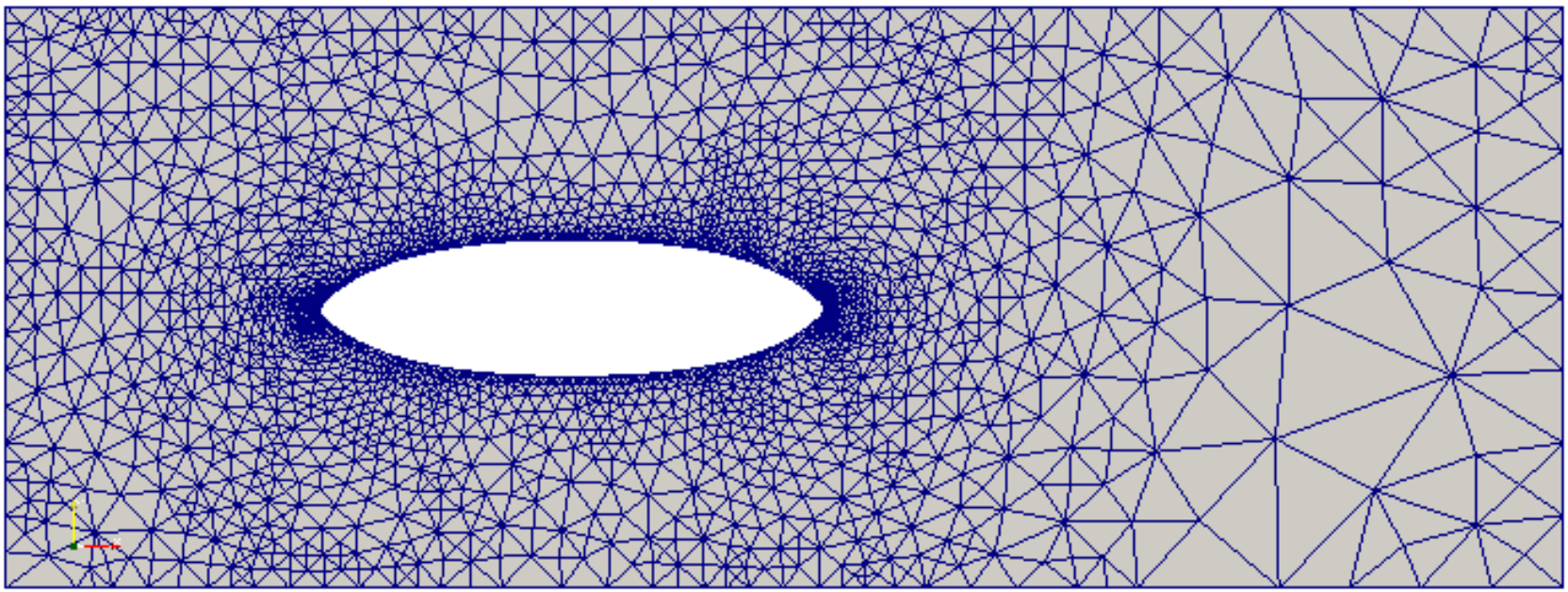
Mesh after 47 iterations



Evolution of meshes (DRAG minimization)

go back

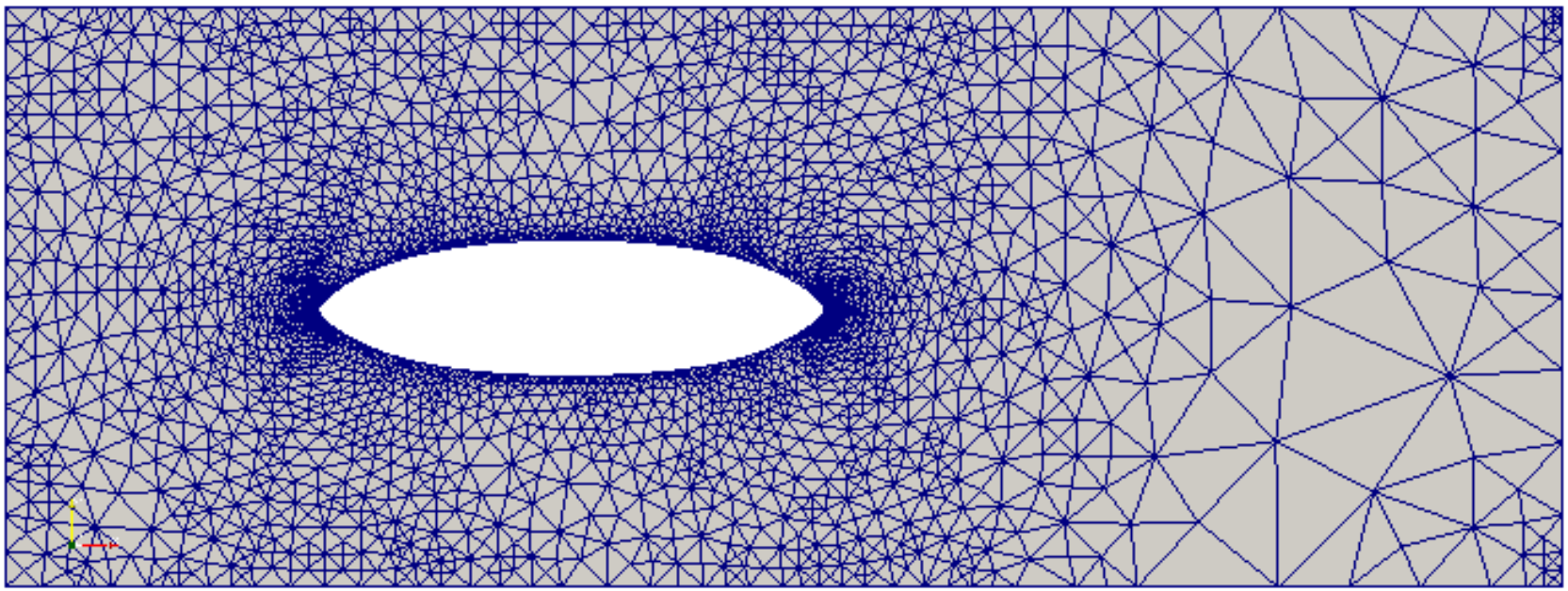
Mesh after 49 iterations



Evolution of meshes (DRAG minimization)

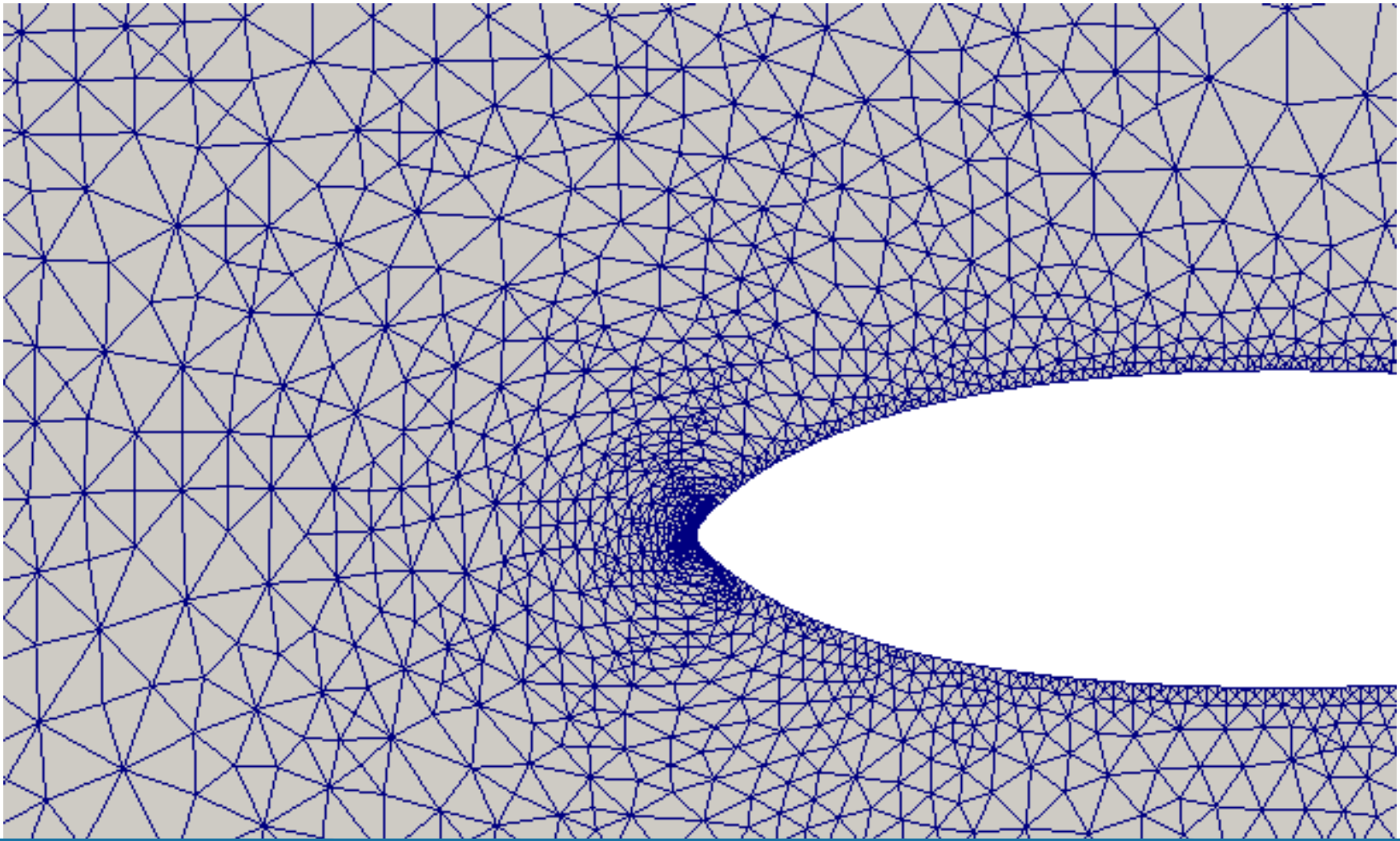
go back

Mesh after 52 iterations



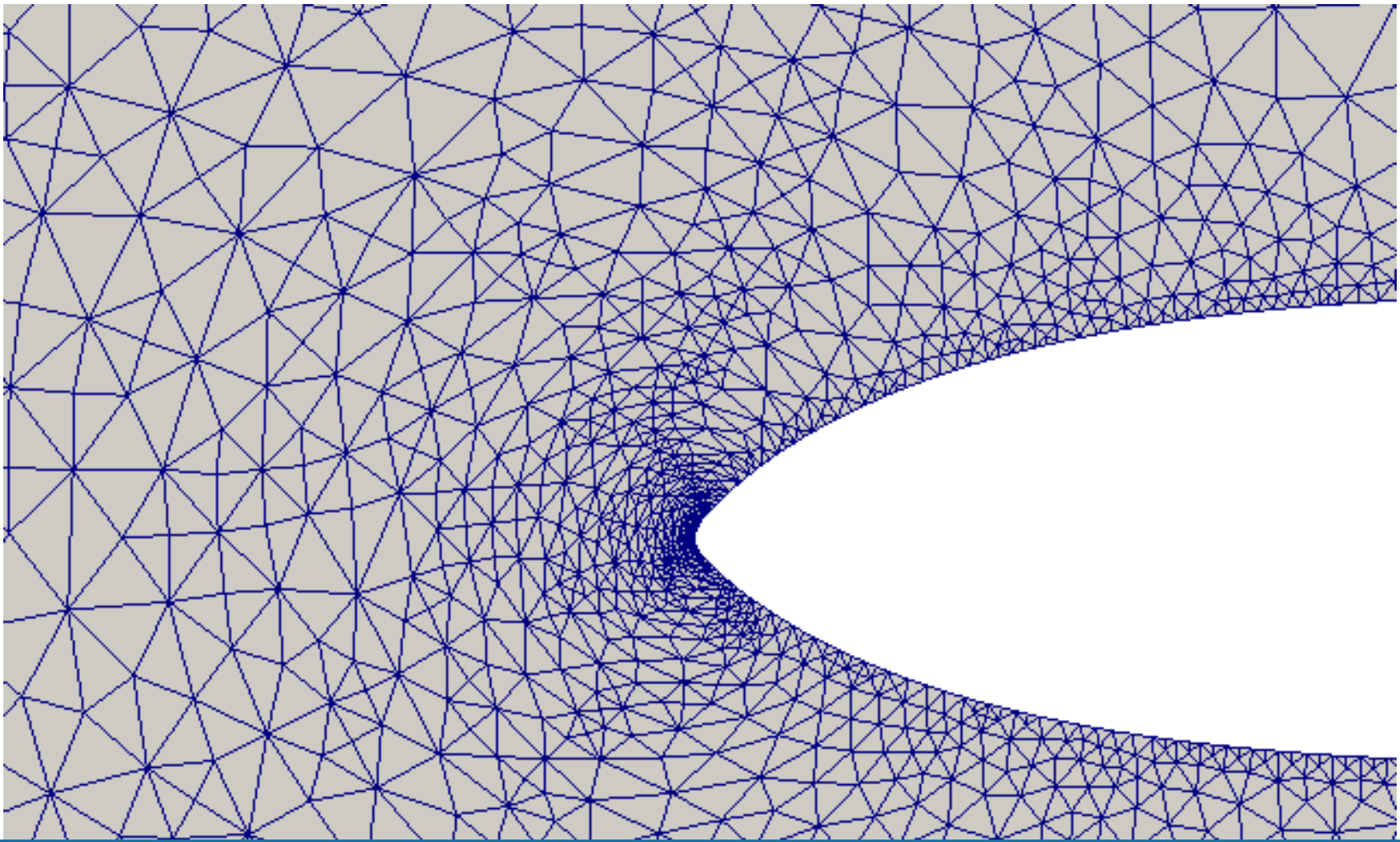
Zoom of final mesh (DRAG minimization)

go back



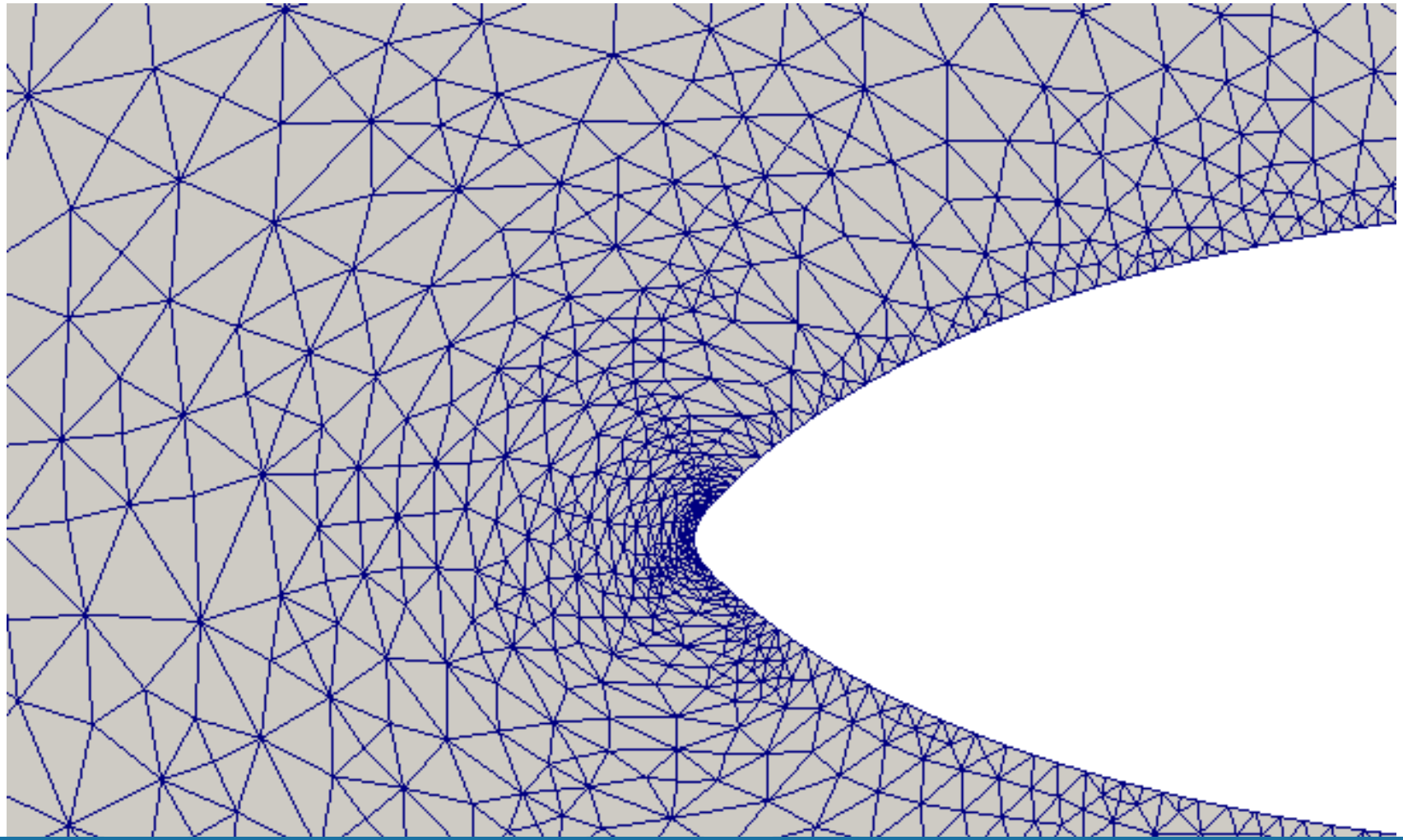
Zoom of final mesh (DRAG minimization)

go back



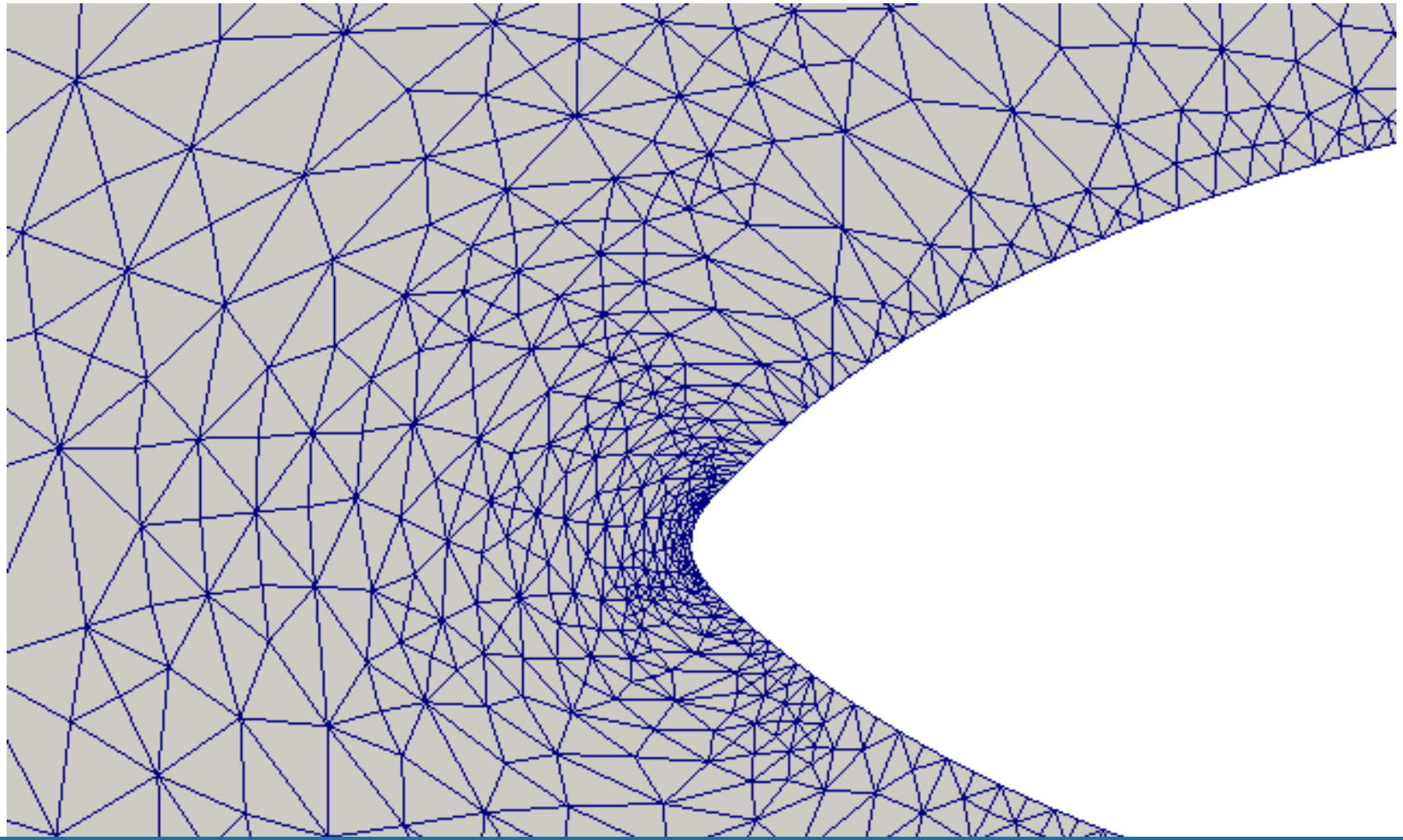
Zoom of final mesh (DRAG minimization)

go back



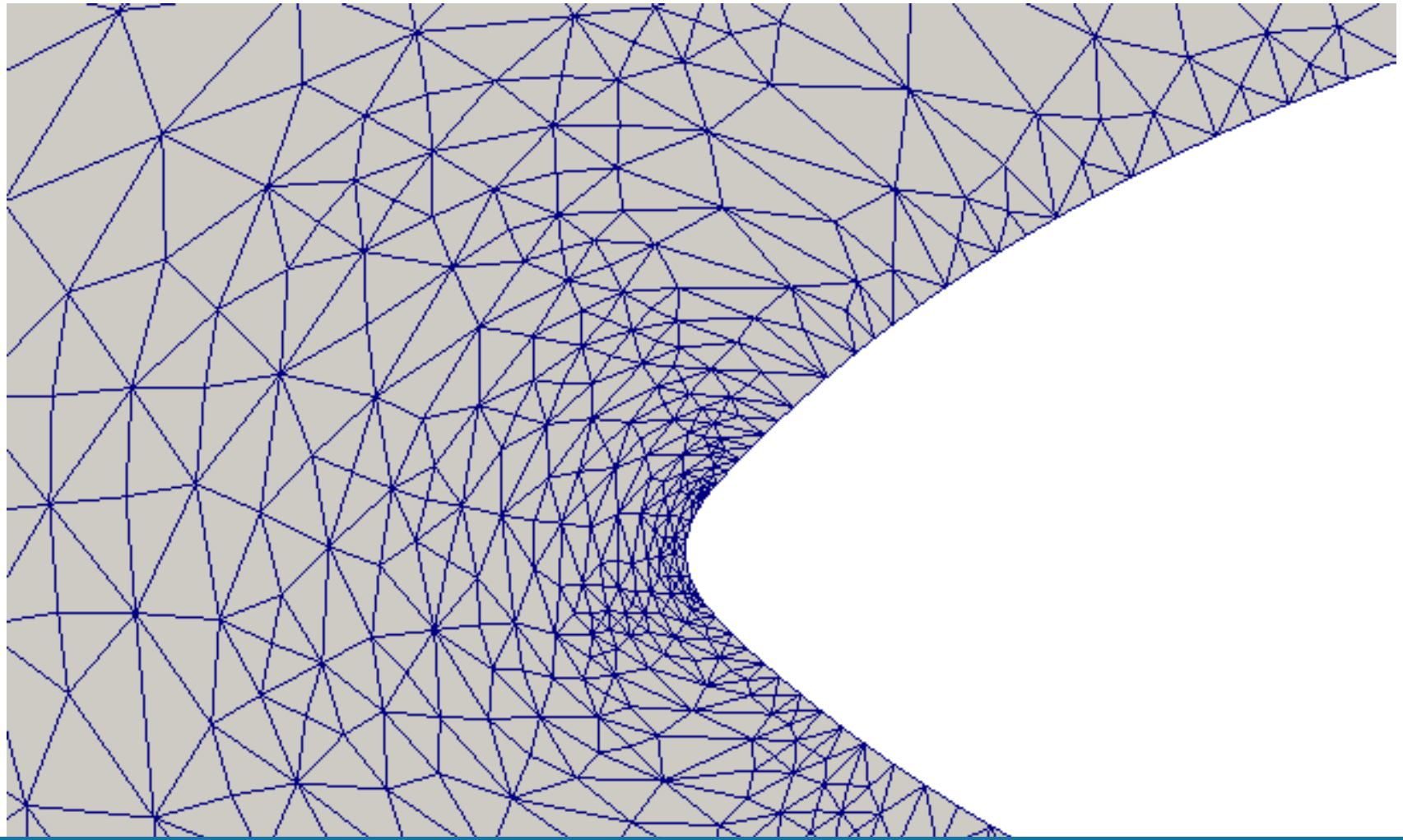
Zoom of final mesh (DRAG minimization)

go back



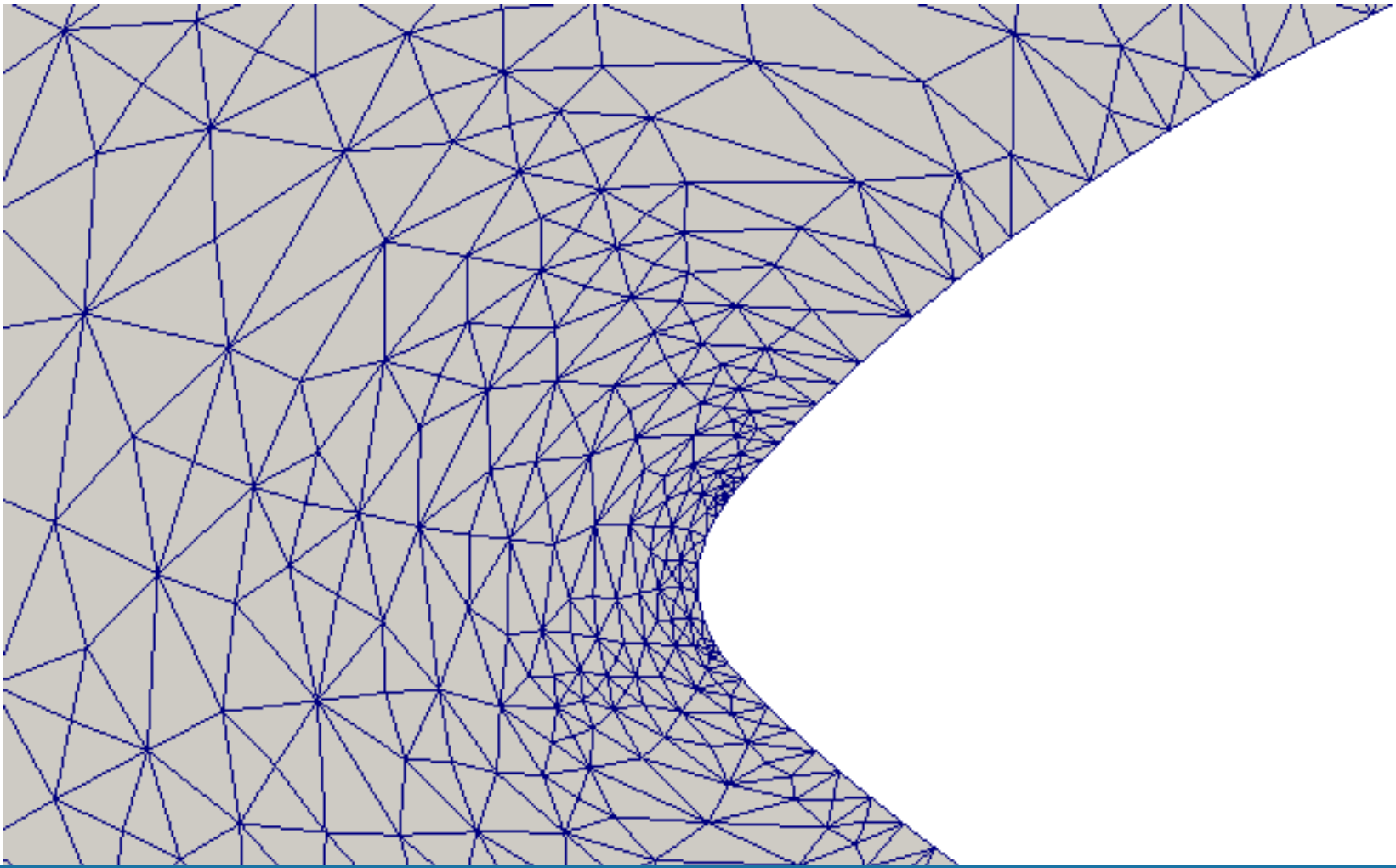
Zoom of final mesh (DRAG minimization)

go back



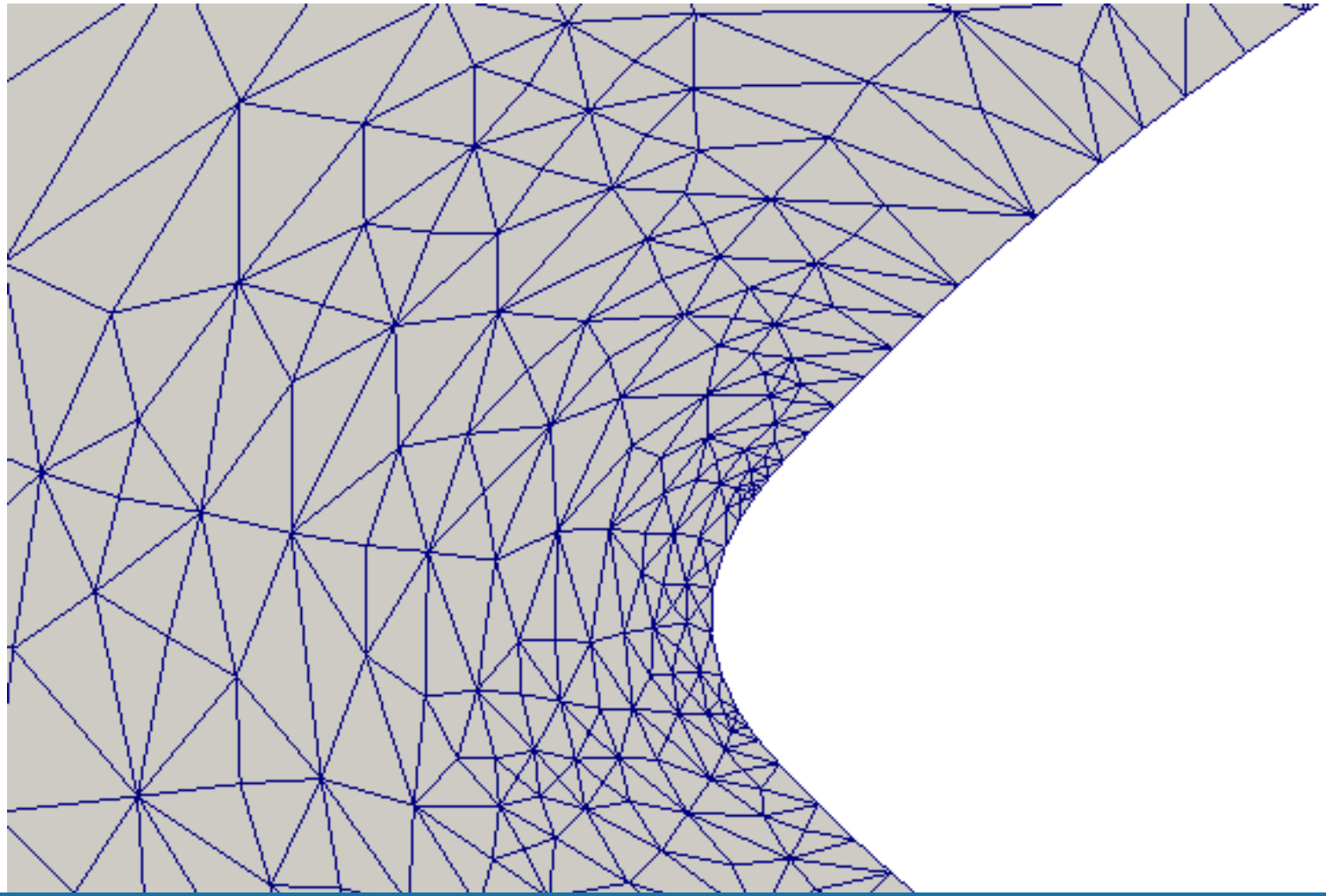
Zoom of final mesh (DRAG minimization)

go back



Zoom of final mesh (DRAG minimization)

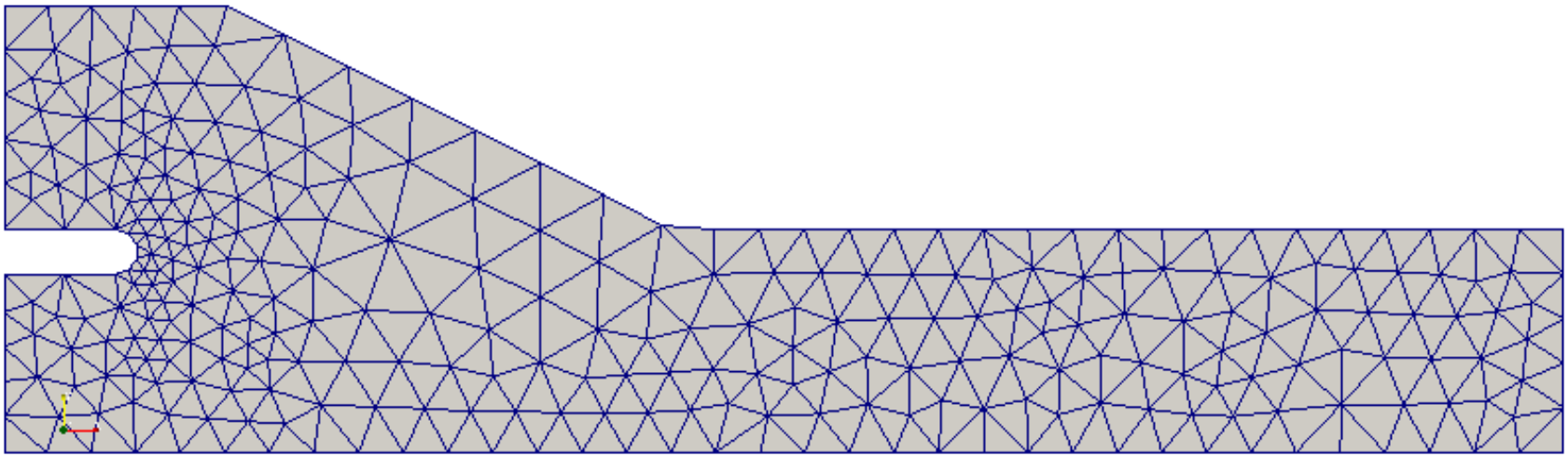
go back



Evolution of meshes (BYPASS: energy minimization)

go back

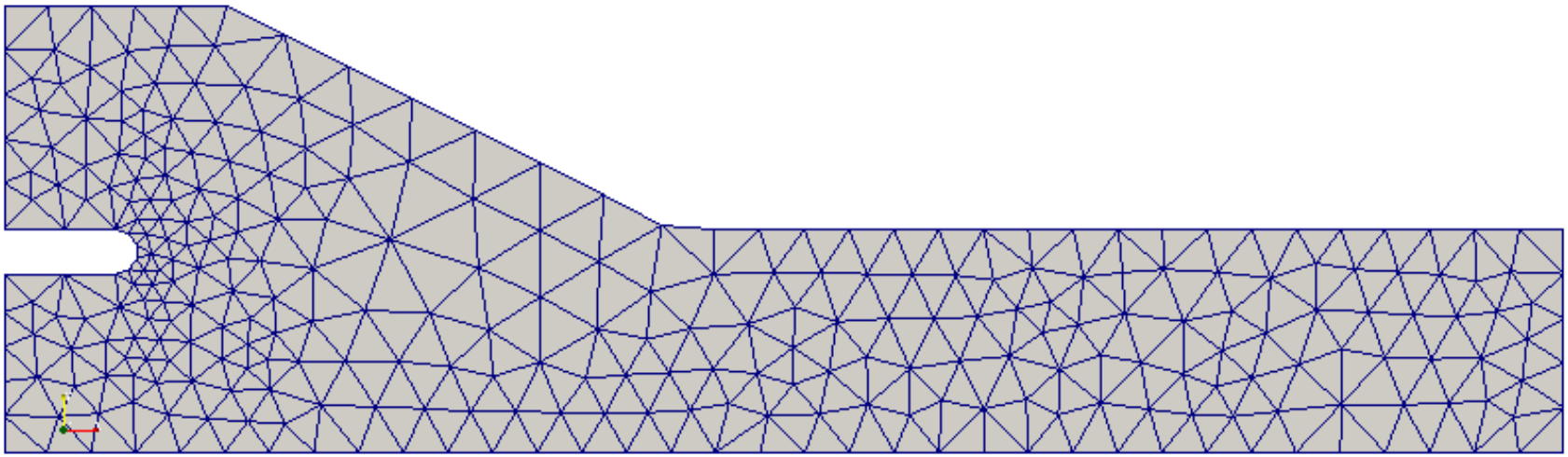
Initial Mesh



Evolution of meshes (BYPASS: energy minimization)

go back

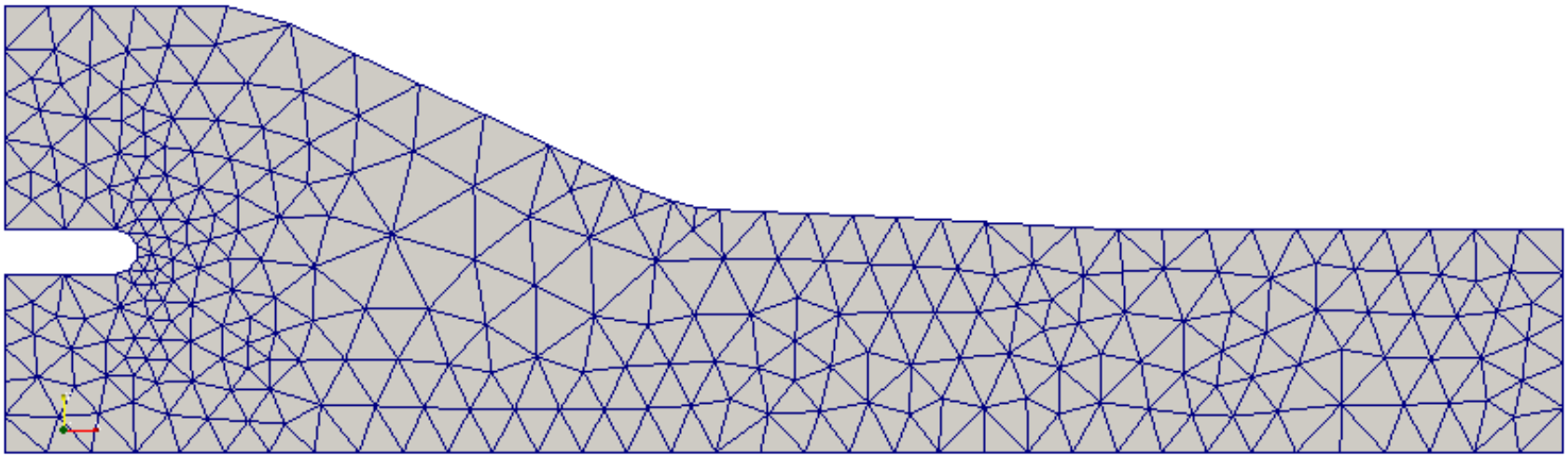
Mesh after 1 iteration



Evolution of meshes (BYPASS: energy minimization)

go back

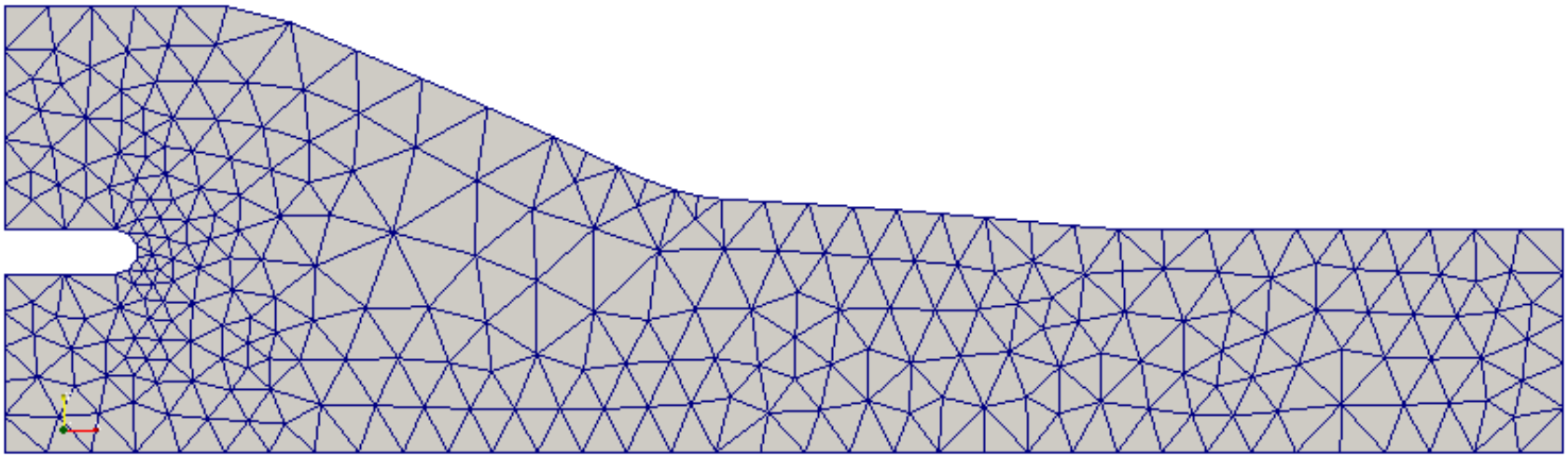
Mesh after 2 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

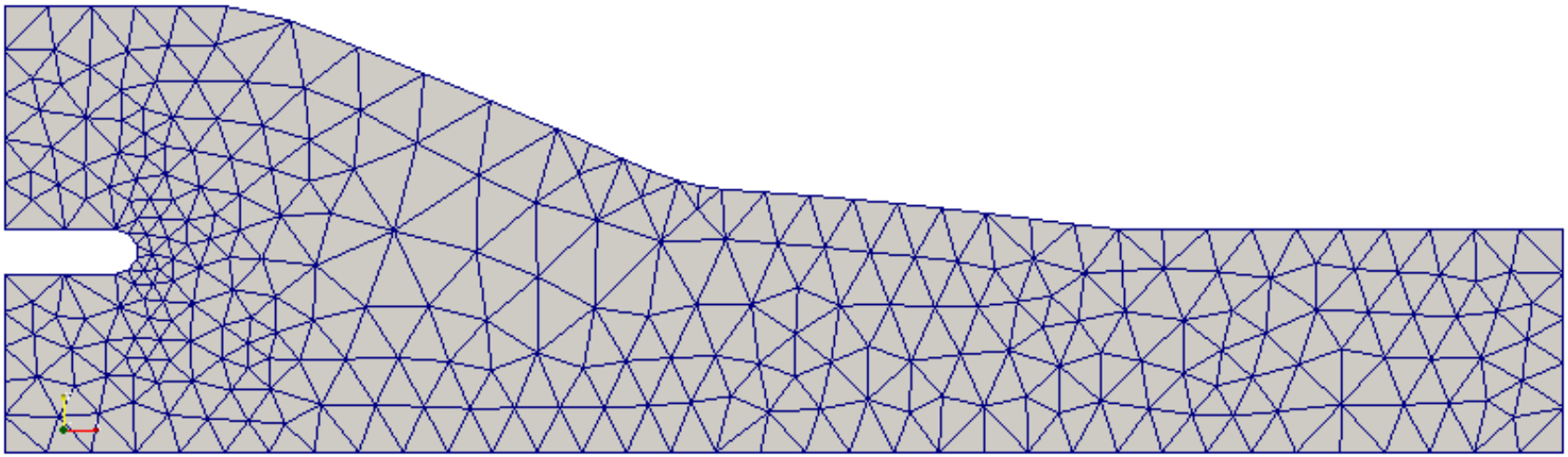
Mesh after 3 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

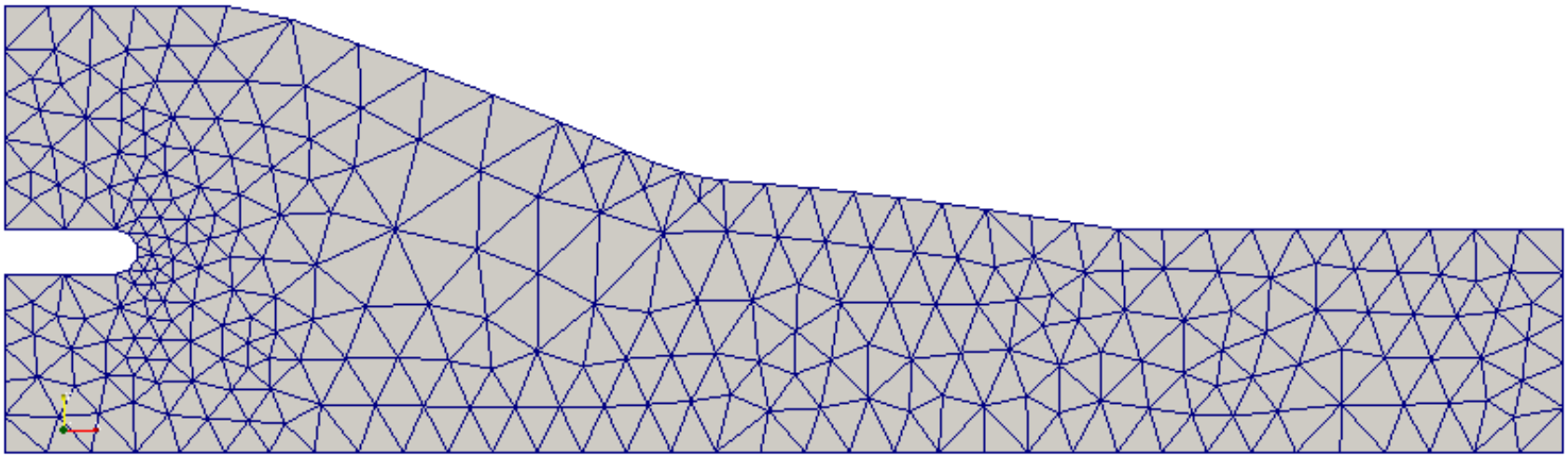
Mesh after 4 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

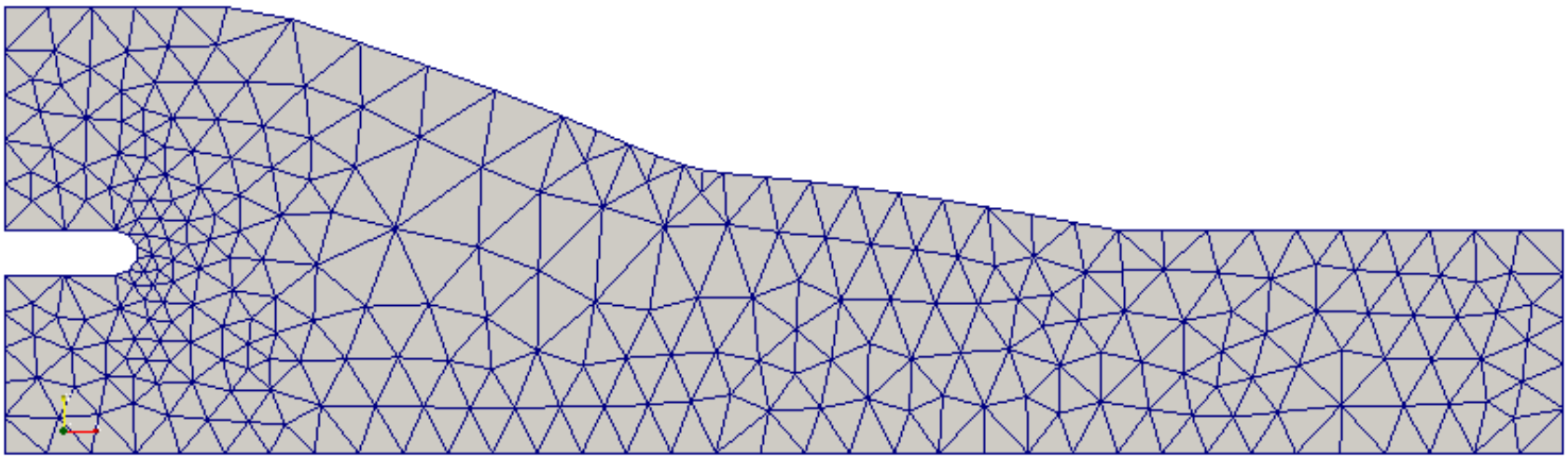
Mesh after 5 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

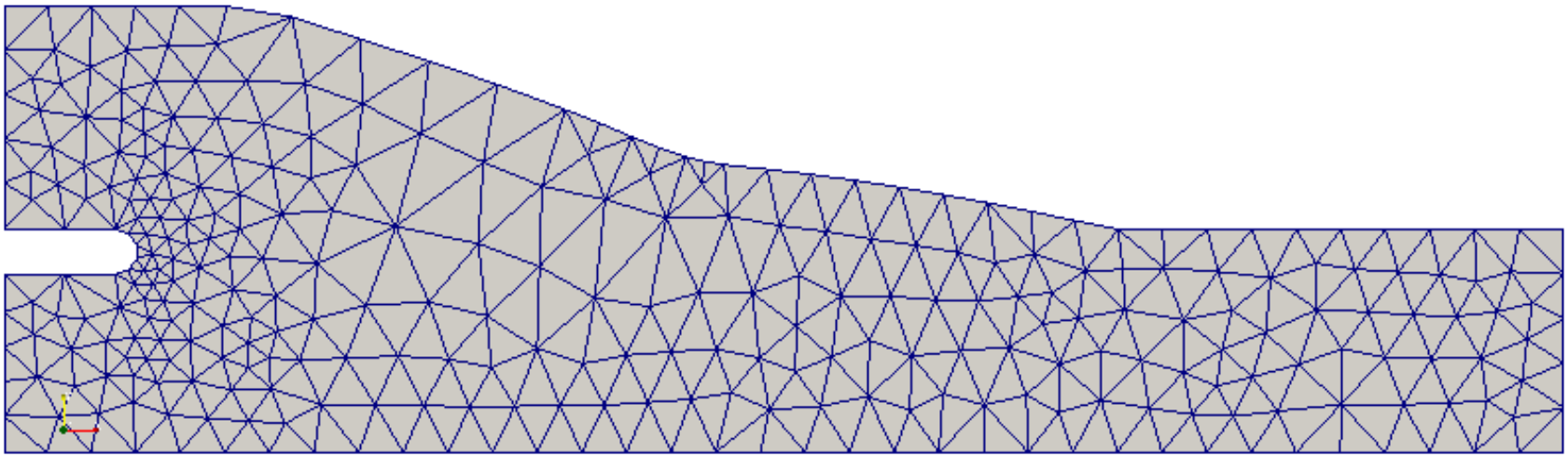
Mesh after 6 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

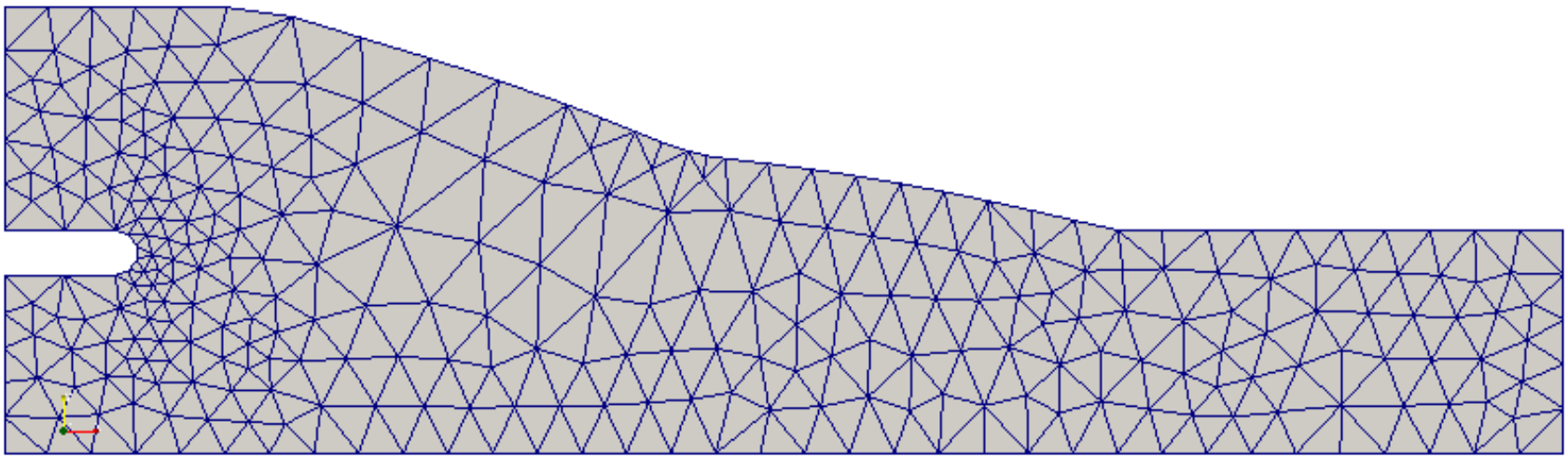
Mesh after 7 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

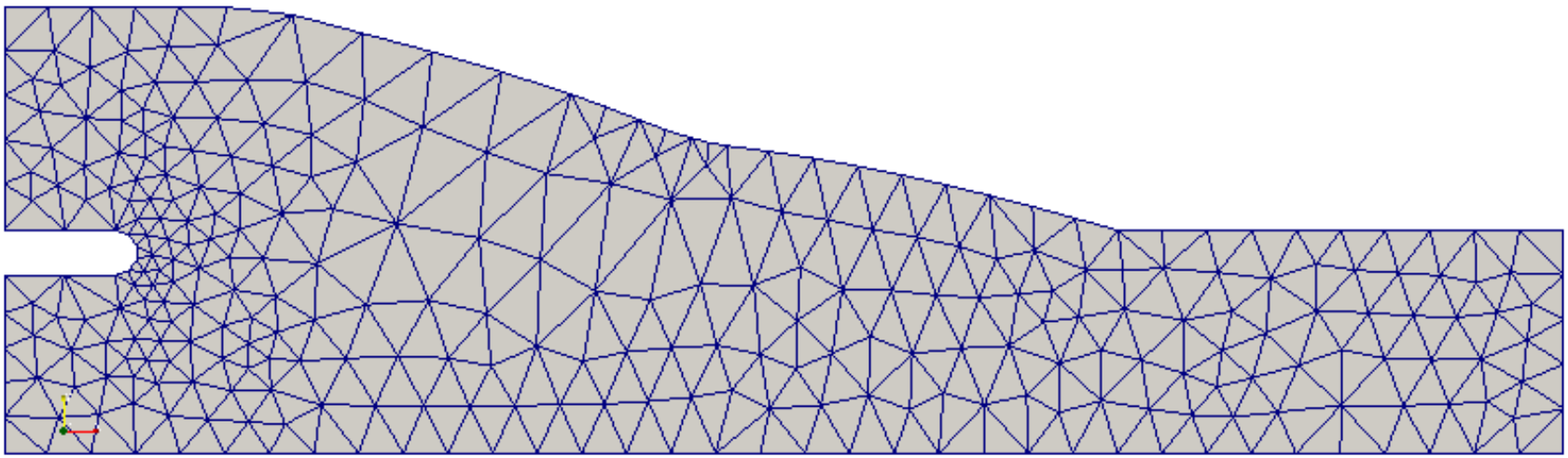
Mesh after 8 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

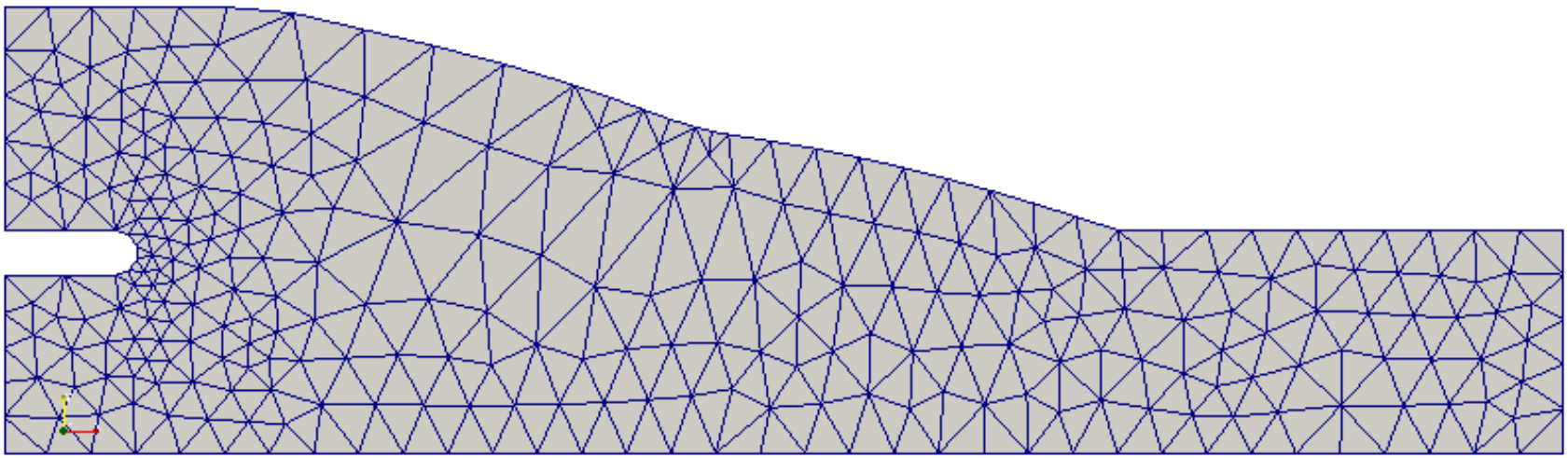
Mesh after 9 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

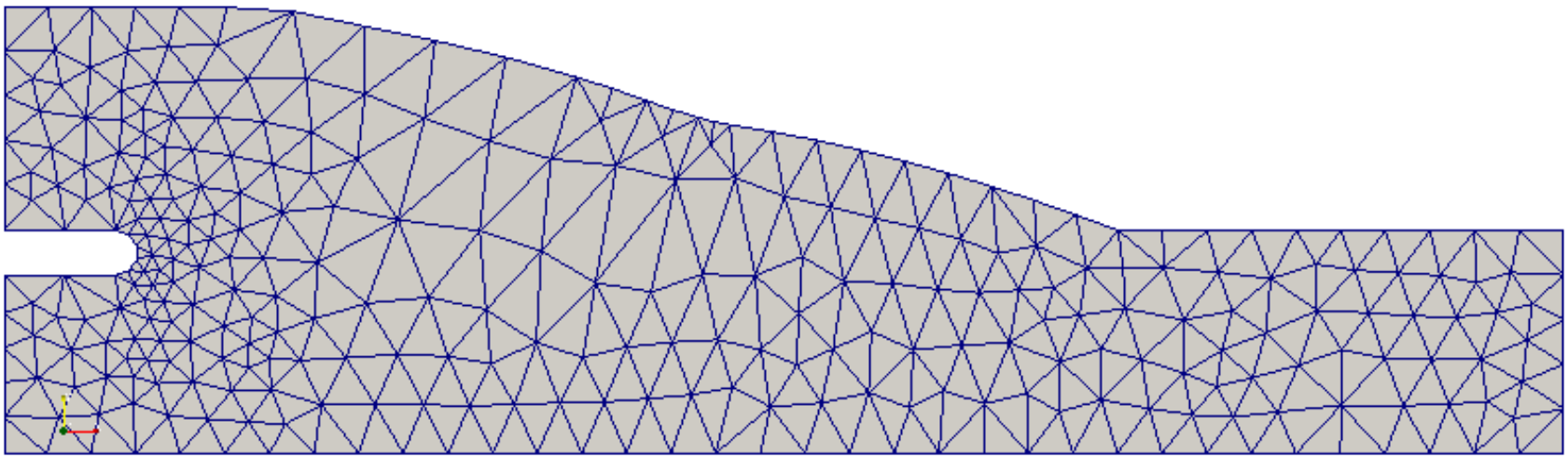
Mesh after 10 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

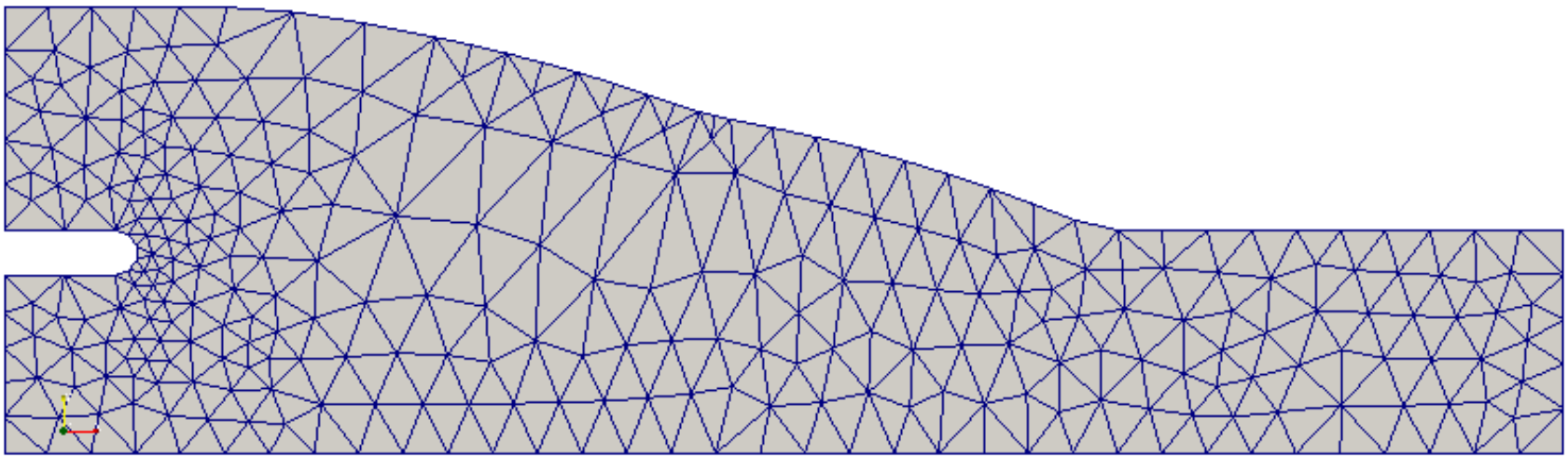
Mesh after 11 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

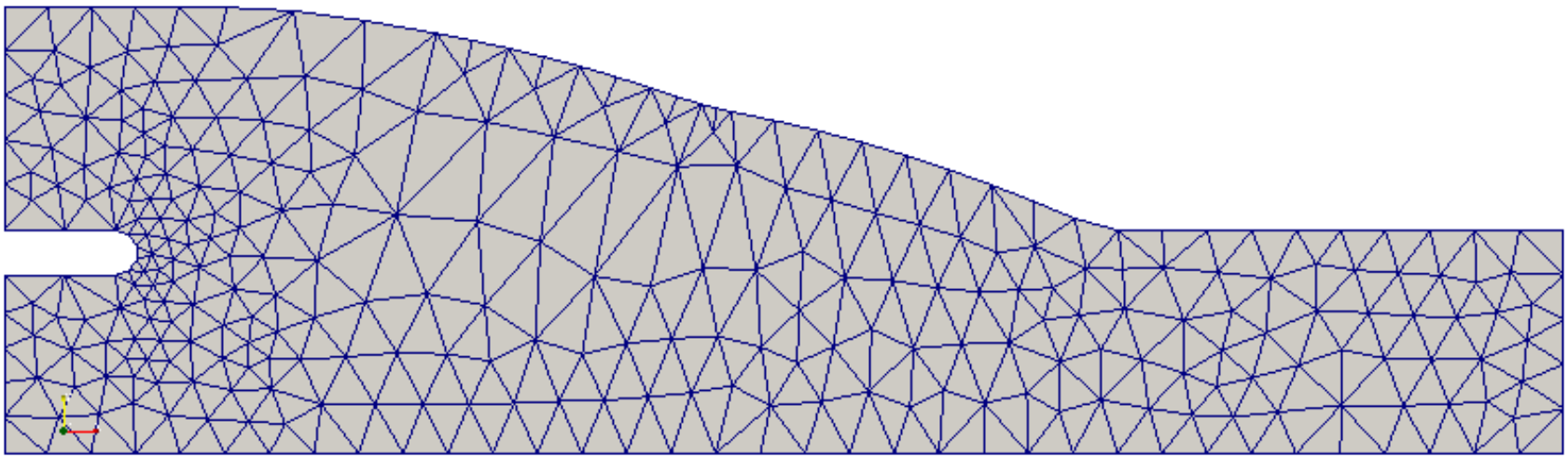
Mesh after 12 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

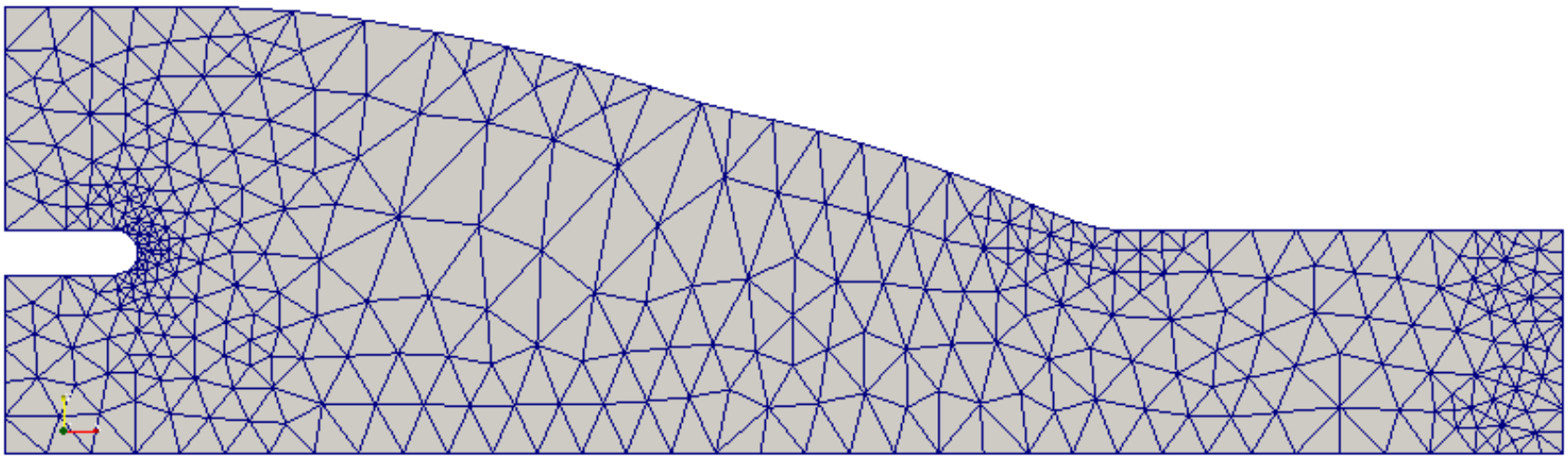
Mesh after 13 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

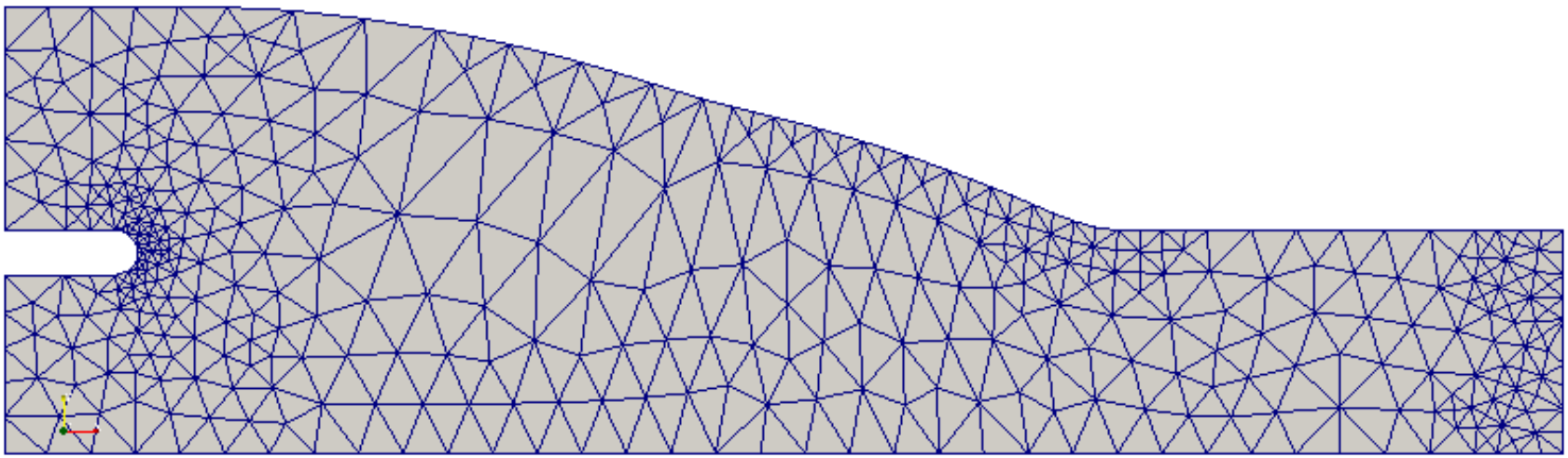
Mesh after 14 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

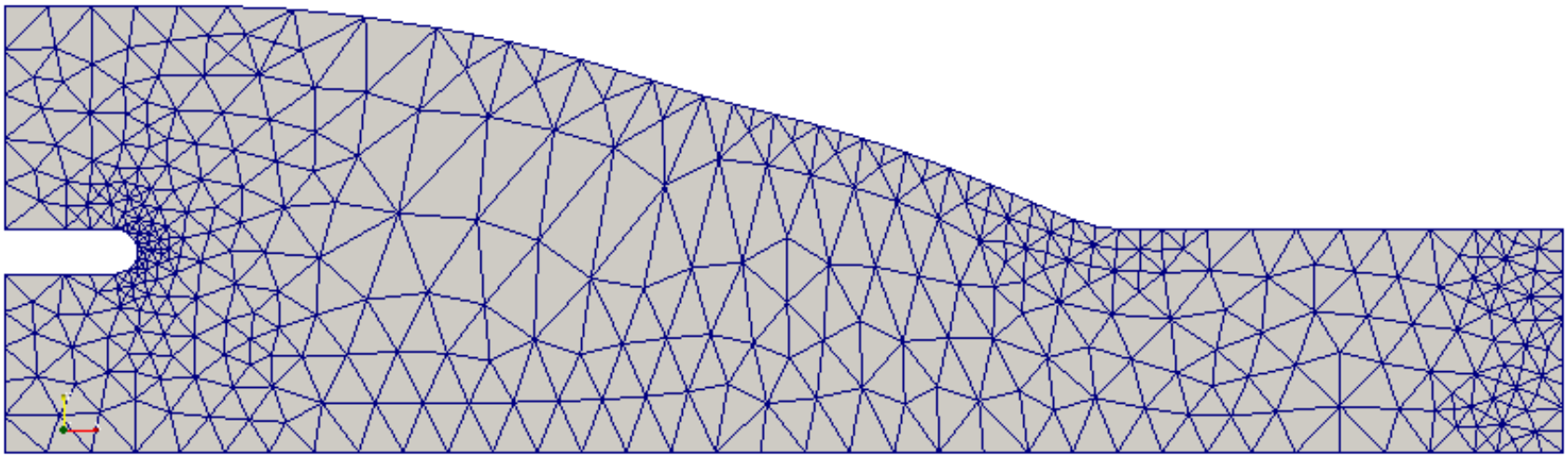
Mesh after 15 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

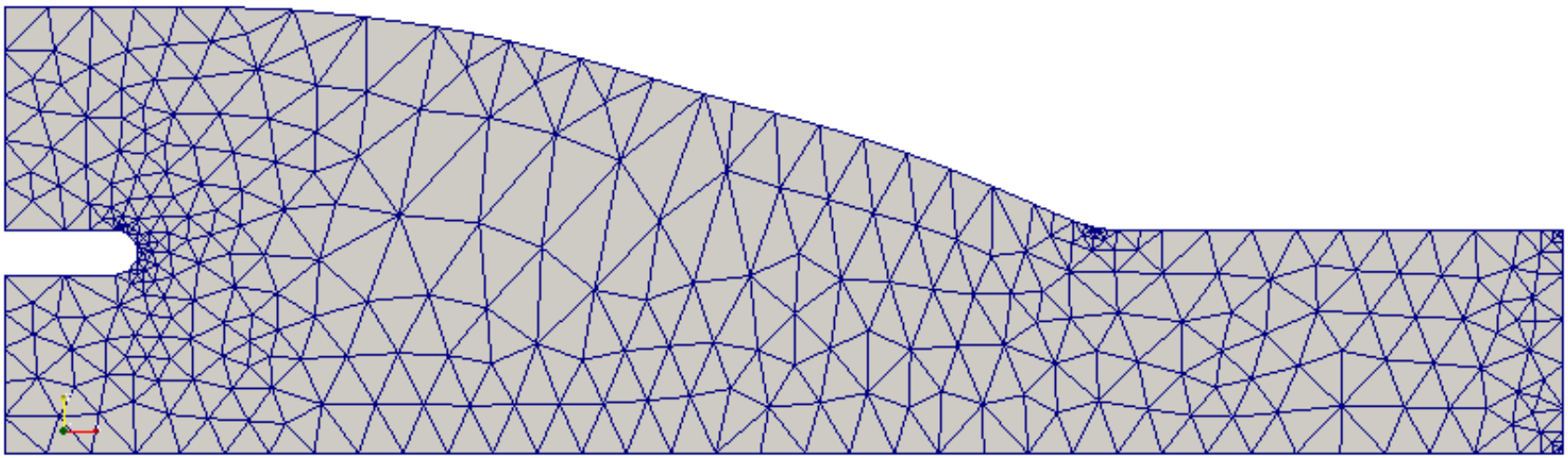
Mesh after 16 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

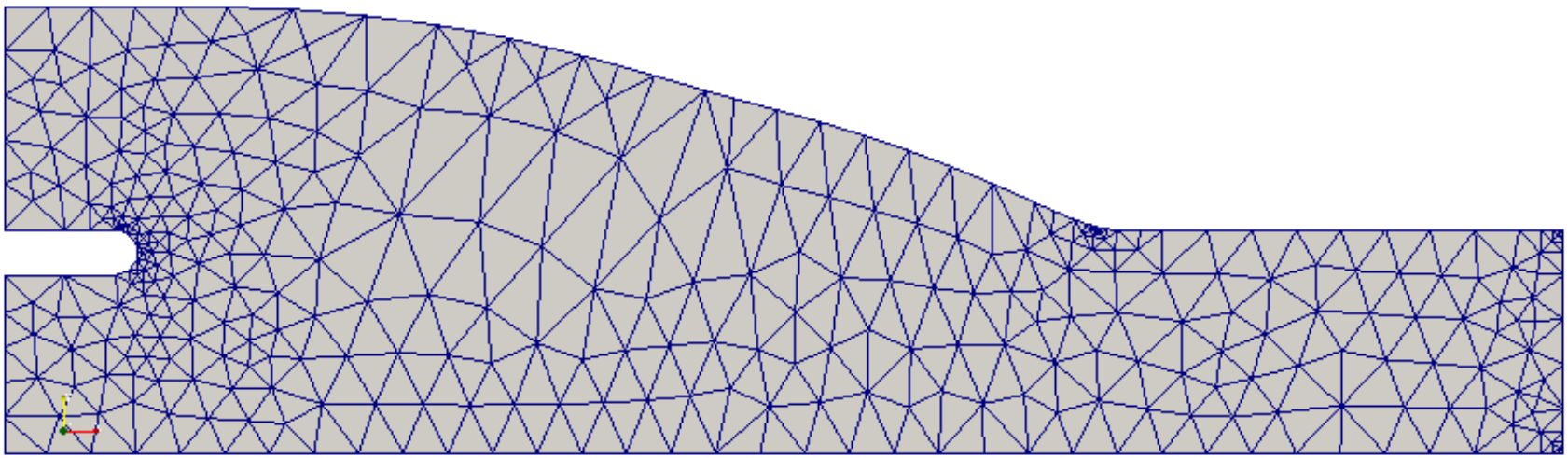
Mesh after 17 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

Mesh after 18 iterations



Evolution of meshes (BYPASS: energy minimization)

go back

Mesh after 19 iterations