To appear in *Optimization Methods & Software*
Vol. 00, No. 00, Month 20XX, 1–18

Algorithmic Differentiation of the Open CASCADE Technology CAD Kernel and its coupling with an Adjoint CFD Solver

Mladen Banović[a*], Orest Mykhaskiv[b], Salvatore Auriemma[c], Andrea Walther[a], Herve Legrand[c] and Jens-Dominik Müller[b]

[a] *Universität Paderborn, Warburger Str. 100, 33098 Paderborn, Germany*
[b] *Queen Mary University of London, Mile End Road, London, E14NS, UK*
[c] *OpenCascade, 1 Place des Frères Montgolfier, 78280 Guyancourt, France*

(*v1.0 released February 2017*)

Computer Aided Design (CAD) tools are extensively used to design industrial components, however contrary to e.g. Computational Fluid Dynamics (CFD) solvers, shape sensitivities for gradient-based optimisation of CAD-parametrised geometries have only been available with inaccurate and non-robust finite differences. Here Algorithmic Differentiation (AD) is applied to the open-source CAD kernel *Open CASCADE Technology* using the AD software tool *ADOL-C* (*Automatic Differentiation by OverLoading in C++*). The differentiated CAD kernel is coupled with a discrete adjoint CFD solver, thus providing the first example of a complete differentiated design chain built from generic, multi-purpose tools. The design chain is demonstrated on the gradient-based optimisation of a squared U-bend turbo-machinery cooling duct to minimise the total pressure loss.

Keywords: Algorithmic Differentiation, CAD kernel, Adjoint CFD method, Gradient-based optimisation.

*Corresponding author. Email: mladen.banovic@math.uni-paderborn.de

1. Introduction

Computer Aided Design (CAD) systems carry out an essential role in engineering design workflows. In a multi-disciplinary product development cycle, the so-called CAD/-CAE/CAM workflow, CAD systems are used extensively to construct and manipulate the geometric representation of the design. In Computer Aided Engineering (CAE), this geometric model is currently used to define the object surface when meshing the computational domain. CAE packages such as Computational Fluid Dynamics (CFD) or Computational Structural Mechanics (CSM) then use these meshes when solving the governing partial differential equations (PDEs) to assess the design performance. After a number of design cycles that are typically controlled manually, the selected design can be manufactured using the geometric CAD model (Computer Aided Manufacturing – CAM).

The next advance in engineering design with CAE is to employ optimisation algorithms which systematically explore the design space and determine the design with optimal performance.

To find the optimal design, one can choose between gradient-free and gradient-based optimisation methods. Gradient-free approaches such as Genetic or Evolutionary Algorithms, require only the standard output data from each element in the chain, *the primal*, such as grid coordinates on the shape, lift and drag coefficients or maximal stresses. Hence it is straightforward to build complete CAD/CAE/CAM optimisation workflows, the only element that is required additionally is the definition of the design space through a set of design variables and their allowable ranges. However, CFD analysis has very long compute times and often needs very rich design spaces, which makes the gradient-free approach prohibitively expensive for industrial application.

We consider here gradient-based methods because they are recognised for their computational efficiency, especially when optimising cases with many control variables. Gradients then need to be computed for all elements in the design chain.

Over the past decades adjoint methods [7, 12, 17] have emerged in CFD as they can compute gradients with respect to an arbitrary number of design variables at near-constant computational cost similar to the *primal* flow computation. Similarly, adjoint methods for linear elasticity are widely used in topology optimisation of structures [1]. Today, industrial application of gradient-based optimisation is primarily limited by the immaturity of the parametrisation tools that define the design space. While CAD-based parametrisations are used with gradient-free methods, but restricted to a very small number of design variables, computing CAD gradients is an unresolved problem.

So far, the derivative computation for a complete design chain, ranging from the shape parametrisation to the computation of the objective function to optimise, has not been demonstrated with exact gradient computation. Robinson et al. [18] used inaccurate finite differences to evaluate gradients. A typical workaround is to use bespoke, non-CAD parametrisations [6], which can be differentiated exactly but offer only limited versatility. Most importantly, such approaches break the overall workflow as the optimal shape exists only as a deformed mesh, but not in CAD, which is a severe shortcoming for routine industrial application.

In this paper we close this gap by demonstrating the differentiation of a generic CAD kernel, in this case the most widely used open-source CAD kernel Open CASCADE Technology (OCCT) [16]. The coupling is shown between the differentiated OCCT kernel and a discrete adjoint CFD solver which is also developed using algorithmic differentiation. The coupled differentiated chain is applied to the optimisation of an internal cooling channel of a turbo-machinery blade. The results demonstrate a significant reduction in

the objective function value. Since the considered design chain consists of generic components, it can be used in a wide variety of applications as discussed later.

The paper is organised as follows. Sec. 2 reviews current gradient-based parametrisation approaches and highlights their shortcomings. Sec. 3 explains the algorithmic differentiation of the OCCT CAD kernel. Sec. 4 briefly describes the U-bend parametrisation, which is our current test-case, and the verification of the differentiated OCCT, along with performance tests. Sec. 5 presents the governing equations for a flow problem and its adjoint, together with an assembly of the relevant derivatives. Sec. 6 shows an application of the complete differentiated design chain to gradient-based optimisation of the U-bend duct. Finally, Sec. 7 offers conclusions.

2. CAD-based Parametrisation Approaches

Typically, gradient-based shape optimisation chains use simple shape parametrisations such as node-based approaches where the displacement of every surface node of the grid is a control variable. To avoid highly oscillatory shapes, regularisation of the gradients or the surface is required, often chosen as implicit [13] or explicit [14] smoothing. Another popular approach is to define a global perturbation field using radial basis functions [5] or with 'Free Form Deformation' which interpolates the deformation of the domain within the control lattice of a volume spline [11, 20]. All of these approaches offer simple derivative computation, but only produce the optimal shape as a deformed mesh rather than a CAD surface.

A completely different approach tackles the shape itself as an own object to be modified, instead of considering every surface node as a control variable. This involves so-called shape derivatives which are rather complicated to derive, see e.g. [19, 21]. However, once this mathematical description is available, adjoint methods can be applied again to compute these shape derivatives.

On the other hand, CAD-based methods work directly with the CAD description in the optimisation loop and can use CAD model parameters as design variables. As a benefit of using these methods, one starts and retrieves the optimal shape in a CAD geometry, but computing the derivatives is more challenging.

Xu et al. [25, 26] use NURBS control points in the CAD-native boundary representation (BRep) to define the design space. Their 'NURBS-based Parametrisation with Complex Constraints' (NSPCC) approach is vendor-neutral (i.e. the parametrisation is not tied to the internal representation of a specific CAD system) and considers only the BRep as given, e.g. in the standardised STEP format, therefore obviating the need to establish any design parametrisation in the CAD system. For computing the derivatives, the NSPCC geometry kernel has been differentiated using the source-transformation AD tool Tapenade [10].

Robinson et al. [18] use the design parametrisation defined in a closed-source CAD system and compute the derivatives using finite differences. To tackle possible changes in topology, the shape differences need to be computed between triangulations (STL) of the CAD surfaces. The use of STL avoids robustness issues with patch re-numbering, but introduces issues of surface-surface projection when computing distances between two STL surfaces. As inherent of finite differences, step-widths must be carefully chosen to limit truncation error. On the other hand, using a parametrisation that is explicitly defined in CAD allows to build geometric constraints into the design space.

Dannenhoffer and Haimes used the open-source CAD kernel Open CASCADE Technology (OCCT) to develop a fully-parametric, feature-based solid-modeling system with

web-based user interface [9]. To obtain the derivatives they applied analytic differentiation to simple geometrical shapes (such as circles or cylinders), while finite differences are used for the more complex geometries [4]. They also considered algorithmic differentiation of the OCCT code, but did not demonstrate an implementation due to the high complexity of the source code.

As is shown in this paper, the algorithmic differentiation of OCCT is indeed feasible. For this purpose, we differentiated the OCCT kernel (*v7.0*) using a *traceless* forward mode as well as *trace-based* forward and reverse modes of the AD software tool ADOL-C (*Automatic Differentiation by OverLoading in C++*) [23]. This work is the first time a fully developed CAD kernel has been differentiated. Section 4 presents the advances this approach offers for gradient accuracy and the opportunity to use the efficient reverse mode of algorithmic differentiation.

Considering other CAD-based approaches, the application of AD to a complete CAD kernel offers a number of significant advantages. As opposed to the finite difference approaches [18], the geometric derivatives are not affected by projection and truncation errors but are exact (up to floating point roundoff). Furthermore, the computational efficiency of the method is superior compared to the finite differences approach. The temporal complexity of the derivative computation can be dramatically reduced even further, once the reverse mode integration is adapted for further structure exploitation.

3. Algorithmic Differentiation of OCCT

3.1 *Introduction to ADOL-C*

ADOL-C is a software tool that facilitates the computation of first and higher derivatives of vector functions that are defined by computer programs written in C or C++. As noted in the acronym, this software tool is based on the *operator overloading* concept (instead of source transformation) and therefore it does not generate intermediate source code. ADOL-C features the following options and differentiation modes:

- trace-based (differentiation modes: *forward* or *reverse*),
- traceless (differentiation mode: *forward*).

These options impose different ways of derivative computation. Concerning the *trace-based* variants, operator overloading generates an internal representation of the function to be differentiated. Later on, this internal function representation is used by ADOL-C driver functions to evaluate the derivatives. Using the *traceless* option, the derivative computation is executed directly together with the primal/function evaluation. Another difference between these options is that the *trace-based* approaches feature the computation of higher order derivatives, while the *traceless* approach allows so far only the computation of first order derivatives. Both options offer the derivative computation in one (*scalar* mode) or many directions (*vector* mode). Although increasing the number of directions requires more computational time and memory, it enables the derivative computation w.r.t. many design parameters with a single code execution.

The ADOL-C library is integrated into the code that is to be differentiated by injection of its specific *adouble* type instead of the native *real* type used by OCCT. That is, one has to use the specific ADOL-C type as a declaration type to all relevant *real* variables, i.e. variables that depend on the design variables and influence the output variables. Otherwise, i.e. if the *adouble* chain is broken in some part of the code, the derivative values will be incorrect. When applying this to a complicated *object-oriented* code like

OCCT, the process of replacing the declaration types is not simple. The differentiation of the OCCT sources as well as the issues encountered during the process are explained in the following section.

3.2 *The Typedef Approach of Differentiating OCCT*

The **typedef** specifier is a reserved keyword in C/C++ used to declare an alias for another data type. OCCT defines the alias *Standard_Real* for *double* data type and we used it as an entry point for algorithmic differentiation, changing *Standard_Real* to become an alias for *adouble* type of ADOL-C. In the so-called typedef approach, almost all declarations of *doubles* variables are replaced by the declaration of *adouble* variables. The main advantage of this approach is that code modification should be minimal with the drawback of sacrificing memory and efficiency to some extent because almost all *double* variables, even the ones that are not needed for the algorithmic differentiation, become *adouble* objects. With the minimal code modification, it should be also straightforward to maintain the differentiated code alongside the original code development. Therefore, it is a very appropriate solution for differentiating large and complex source codes like OCCT.

Although the idea about the *typedef* approach looks simple, it is not as straightforward as one would expect. The differentiation of OCCT version 7.0 involved a significant amount of code modification and even after successful compilation, a large number of run-time errors had to be resolved during the testing phase.

3.2.1 Compile and run-time issues of OCCT differentiation

Here we describe the difficulties faced during the *typedef* implementation and the corresponding solutions. Some of the compile-time issues were related to:

- On a very low level of the OCCT kernel, static assertion is used to check whether the size of *Standard_Real* is equal to the size of 2×*Standard_Integer* (which is the *typedef* for *int* type). Once the *typedef Standard_Real* corresponds to *adouble*, the static assertion fails, therefore yielding a compile-time error. The reason of having such an assertion is due to a definition of the *union* with two members of {*Standard_Real*, 2×*Standard_Integer*}. *Union* is a user-defined type in C/C++ in which all its members occupy the same physical space in memory. Until C++11 standard, *unions* were allowed to store only primitive data types. Without further investigation how to integrate a non-primitive type with C++11 standard, we kept the *double* data type here because the *union* appears only in the low level of OCCT that is not related to the modeling algorithms. This affects the class *FSD_BinaryFile*, where the *union* is used to inverse bytes of a *real* number. To overcome compile-time issues, keeping the *double* variables results in a certain modification of other sources that used the *union* in order to create a sort of bridge between *doubles* and *adoubles*.
- Hundreds of places in the OCCT code involve explicit/implicit conversion of *Standard_Real* to *int* type. In the terms of algorithmic differentiation, this could cause a problem because an integer object does not carry along the derivative information. By doing such a type change, the computational graph of the function to be differentiated is disconnected in that part. Hence, the chain rule is broken and wrong derivatives may be computed as a result. This is the reason why such a cast is not automatically supported in the *adouble* class. Being aware of the risk, we allowed it simply by using the *getValue* method on the *adouble* object which extracts the primal (*double*) value

of it. A small example where this could cause a problem is related to a method of the *BndLib_Box2dCurve* class, as shown in Listing 1. Let us assume that the input *adouble* variables `aT` and `aPeriod` carry the derivative information. Such a derivative information will not be propagated in the line where `k` is computed because the expression `(-aT/aPeriod)` is converted to *int*. On the following line, the result `aTRet` is computed using the previously described variable `k` with the truncated derivative information. On the other hand, there is no problem if the inputs `aT` and `aPeriod` are not 'active', i.e. do not carry derivative information. Since this is application-dependent, the derivatives have to be carefully verified or the original code structure has to be modified to avoid such problems.

- *Standard_Real* is not the only *typedef* for *double* in OCCT. Although it is broadly used in OCCT, there are many other *typedefs*: *Quantity_Acceleration*, *Quantity_Area*, *Quantity_Coefficient*, *doublereal*, *GLdouble* etc. Most of them are replaced by *Standard_Real* to keep consistency across the OCCT kernel, but there are also exceptions where native *double* type is required. These exceptions mostly relate to packages that belong to the visualisation module of OCCT, e.g. the *OpenGL* package which uses both *doubles* and *floats*. This means that the *adouble* presence is reduced in such packages as much as possible (which is reasonable), but not entirely. The issue lies in the fact that visualisation packages use geometrical entities like point, vector and axis, that already contain *adouble* objects because they use *Standard_Real typedef*. To overcome this problem is not simple, but it is successfully resolved by introducing intermediate variables and breaking the chain rule wherever required.
- Functions that are a part of external libraries can not work with *adoubles*. Therefore, the compiler reports type mismatch in such places. Depending on the function arguments, whether they are pointers or not, we had to substitute *adoubles* with *doubles* or call the *getValue* method on the *adouble* objects. This includes also functions like `modf` (decomposes a number into integer and fractional parts) and `fmod` (calculates remainder of the floating point division operation). They are C-functions (defined in the header *cmath*) and the differentiation rule for them is simply not defined. Therefore, similar to the type cast case described above, the chain rule is broken when using these functions.
- Functions for printing to an output, like *sprintf*, can not accept *adouble* as an argument. However, the *sprintf* function is used in a lot of places in OCCT, mostly referred to printing standard CAD output formats like STEP and IGES. Since *sprintf* is a C-function and can not be overloaded in ADOL-C, we used the *getValue* method here as well.

Listing 1 *BndLib_Box2dCurve* class method (simplified)

```
Standard_Real BndLib_Box2dCurve::AdjustToPeriod(const Standard_Real aT,
                                                const Standard_Real aPeriod)
{
  Standard_Integer k;
  Standard_Real aTRet;
  aTRet=aT;
  if (aT<0.) {
    k=1+(Standard_Integer)(-aT/aPeriod).getValue(); //possible loss of derivative
    aTRet=aT+k*aPeriod;
  }
  //...
  return aTRet;
}
```

Furthermore, some of the run-time issues were related to:

- The left and right shift operators (<< and >>) are overloaded in the *adouble* class. Since they are also used in the OCCT output system, the files were corrupted by *adoubles*, because the derivative information is also printed to the output. There is no need that files like STEP contain such an additional information. Hence, the solution was just to extract the *primal* values of *adoubles* using the *getValue* method wherever necessary.
- C dynamic memory allocation is used in the OCCT kernel. This causes errors once the *adoubles* are present. An *adouble* object has to be initialised using its constructor, such that the correct memory amount is allocated. The C-function *malloc* does not achieve that, triggering 'segmentation fault' errors upon program execution. For this reason, we replaced the functions *malloc/free* with the C++ operators *new/delete*. Moreover, the C-functions *memset* and *memcpy* were replaced by *for* loops in order to manually assign or copy the values from one pointer to another. Otherwise, the memory exceptions would occur. Even more complex low-level memory management is used in one specific package of OCCT that we could not differentiate, as described in Sec. 4.
- In many places of the OCCT code, we had to use explicit conversion from *real* numbers to *adoubles*, if these numbers are passed as arguments to the specific overloaded methods. For example, consider the *SetCoord* method which is overloaded in OCCT in two different ways: *SetCoord(Standard_Integer, Standard_Real)*, which sets a coordinates value by its index, and *SetCoord(Standard_Real, Standard_Real)*, which specifies each coordinate value independently. A case in the code where developer uses the second method with *real* numbers, e.g. *SetCoord(6., 8.)*, is not correctly identified in the differentiated sources because the compiler calls the method with primitive types as arguments, which is the first method. Without any interaction this may not produce a runtime error, but certainly the values are wrong. The correct way in such cases is to add an explicit conversion, i.e. *SetCoord((Standard_Real) 6., (Standard_Real) 8.)*.

After integrating the *traceless* forward mode into OCCT by using the *typedef* approach, we have also considered the *trace-based* options of ADOL-C. The first step is to compile OCCT sources with the ADOL-C *trace* headers, which has been completed successfully. The following step is to use ADOL-C driver functions in the specific parts of the code in order to evaluate the derivatives. Both *traceless* forward mode and the *trace-based* modes of AD have been validated in a specific test-case that is described in the following section. Further work will be dedicated to structure exploitation when using the reverse mode for efficiency improvement.

4. Verification of Differentiated OCCT with U-bend Test-case

4.1 *Testing the Differentiated OCCT Kernel*

After successful compilation of the differentiated OCCT kernel, we first verify the original (primal) functionality. For such a purpose, OCCT provides its own *Automated Testing System* which consists of more than 10,000 tests related to all OCCT modules. As mentioned in Sec. 3, a large number of run-time errors had to be resolved during the testing phase. A very small number of issues could not be resolved, see Table 1 for details. The majority of the tests marked *failed* are related to the package *AdvApp2Var* which deals with approximation of functions based on Legendre polynomials. In the beginning, this approximation code was written in Fortran and later automatically translated to C code. The package involves low-level memory management routines which make *adouble* handling quite difficult. Therefore, whenever it is used, a run-time memory exception will

Table 1. OCCT *Automated Testing System* final results

Tests marked: OK	Tests marked failed: FAILED	**Success rate**
11,306	374	**97%**

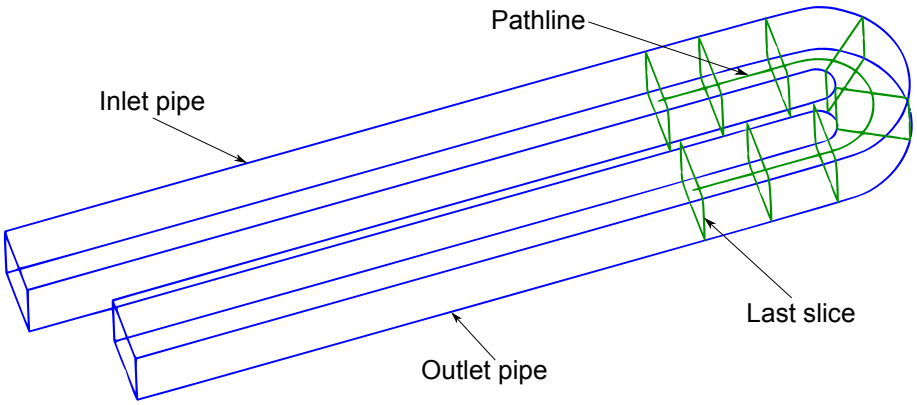


Figure 1. U-bend geometry

occur. The only solution here is to rewrite the package such that it follows the C++ standards for memory management.

In Sec. 4.3, we verify the correctness of the computed derivatives using the geometric model of the U-Bend of Sec. 4.2.

4.2 *U-bend Parametrisation*

The U-bend under investigation is a typical internal cooling channel used in a turbine blade application. The baseline geometry consists of a circular U-part with attached inlet and outlet legs which are not modified during the optimisation. The baseline geometry is shown in Fig. 1.

The parametrisation is done on the U-part: the three-dimensional shape is constructed by continuous evolution of a rectangular 2-D cross-section along a guiding circular path-line (the sweeping function in CAD). The cross-section is defined by 4 Bézier curves with 4 control points each, forming a closed 'wire' with a total of 12 control points. Each 2-D slice lies on a plane orthogonal to a pathline, which itself is described as a B-spline curve (in green in Fig. 1). The parametric coordinates of each control point within the 2-D cross-section have their own law of evolution along the pathline, also defined by a B-spline curve. The final B-spline surfaces of the U-Bend are fitted to the swept cross-sections. Therefore, the actual design parameters considered in the optimisation are the parameters of the laws of evolution, in this example 96 degrees of freedom.

4.3 *Gradient Verification using U-bend Parametrisation*

The correctness of the computed gradients is verified using the U-bend parametric model. As a representative example, we compare here the surface sensitivities with respect to one design parameter calculated by the *traceless* forward mode of AD and finite differences (Fig. 2). The overall magnitude plots illustrate that these results coincide to a very high extent. Also the quantitative comparison between AD and finite differences for the same

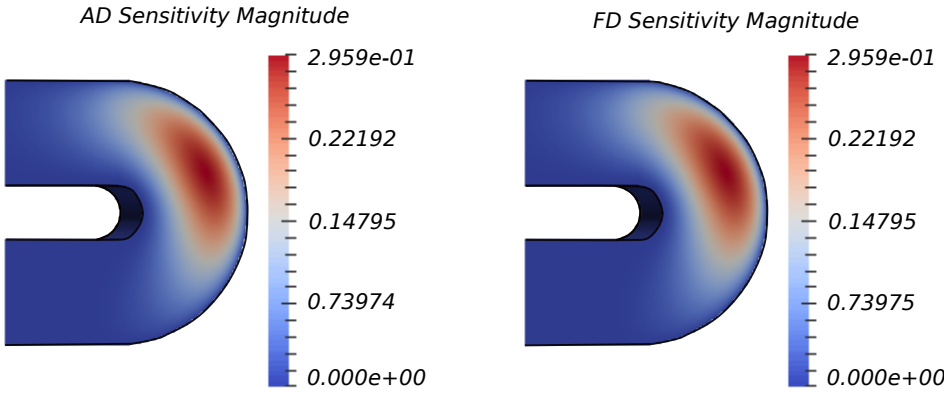


Figure 2. U-part sensitivities evaluated by AD (left) and finite differences (right)

Table 2. AD and FD values comparison for several U-bend point coordinates

AD value	FD value	Abs. difference
0.00038436	0.00038426	1.02e-07
0.06339189	0.06339125	6.41e-07
0.15615249	0.15614906	3.43e-06
0.12874039	0.12873815	2.24e-06
⋮	⋮	⋮
0.27459387	0.27458089	1.30e-05*

*Maximal difference

design parameter shows mutual agreement (Table 2). Furthermore, the same surface sensitivities are also verified with a Taylor test:

$$f(x+h) - f(x) - h\frac{\partial f}{\partial x}(x) = \mathcal{O}(h^2). \tag{1}$$

The Taylor test was performed on a number of arbitrary surface points with a range of step sizes $h \in [10^{-1}, 10^{-10}]$. The error plots (the left-hand side of Eq. (1)) in eight surface point coordinates are presented in Fig. 3. Here we observe even better convergence than the theoretical convergence rate of h^2. This behaviour continues until $h = 10^{-4}$, where the errors reach machine precision.

We also validate the derivatives computed with the *trace-based* variants against the *traceless* forward mode. The results presented in Table 3 and Table 4 show some small disparity (close to machine precision) between the gradients. This is due to the differences between some overloaded operators of ADOL-C in the *trace-based* and *traceless* options. Not only the derivative calculation but also the primal evaluation is affected in the same order of magnitude. The differences between the *trace-based* and the *traceless* variants are therefore small enough not to yield radically different CAD models, and hence both can be used equally.

This verification ensures the correctness of the computed derivatives only for the computational path exercised by the U-bend geometry. Although this test-case does use a lot of methods from the OCCT kernel, it represents a very small part of the complete OCCT capability. Clearly, adding definitions for all possible input and output variables to all regression tests is not a feasible task. An unanswered challenge to the AD community

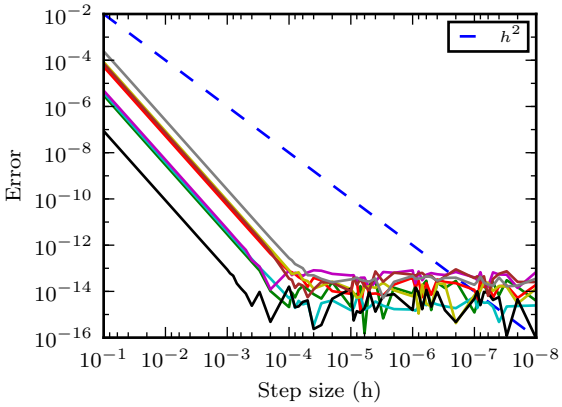


Figure 3. Taylor test overview for eight U-bend surface point coordinates

Table 3. AD Traceless-forward and AD Trace-based forward gradient comparison for several U-bend point coordinates

AD Traceless-forward gradient	AD Trace-based forward gradient	Abs. difference
1.03293863915149e-01	1.03293863915283e-01	1.34e-13
2.36707086987917e-01	2.36707086987913e-01	4.00e-15
1.28682761975210e-01	1.28682761975225e-01	1.50e-14
1.25652442670046e-02	1.25652442669980e-02	6.60e-15
⋮	⋮	⋮
2.24699593280905e-01	2.24699593281250e-01	3.45e-13*

*Maximal difference

Table 4. AD Traceless-forward and AD Trace-based reverse gradient comparison for several U-bend point coordinates

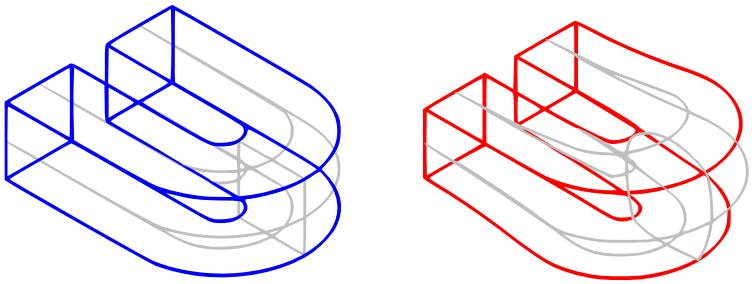
AD Traceless-forward gradient	AD Trace-based reverse gradient	Abs. difference
1.03293863915149e-01	1.03293863915222e-01	7.30e-14
2.36707086987917e-01	2.36707086987912e-01	5.00e-15
1.28682761975210e-01	1.28682761975216e-01	6.00e-15
1.25652442670046e-02	1.25652442670053e-02	6.99e-16
⋮	⋮	⋮
2.24699593280905e-01	2.24699593281155e-01	2.50e-13*

*Maximal difference

is how to not just automatically produce the derivative code, but also to derive relevant derivative regression tests from existing primal tests.

4.4 *Performance Tests*

To measure performance by computing a large number of derivatives, we have developed an optimisation example that is a typical, often executed task in CAD, so-called 'surface fitting'. It is used to find a set of design parameters in parametrisation P to match a



(a) Original U-bend shape (b) Perturbed U-bend shape

Figure 4. CAD optimisation with two U-bends

certain geometry T, with the following optimisation problem:

$$\min_{x\in\mathbb{R}^{96}} f(x) = \sum_{i=1}^{12000} \|P_i(x) - T_i\|^2 \quad \text{s.t.} \quad lb_j \leq x_j \leq ub_j, \quad j = 1, \ldots, 96. \tag{2}$$

where j is the number of the design variables, i is the index of one of 12000 sampling points distributed uniformly over the surface, $P_i(x)$ and T_i are the points on the original and target (perturbed) surfaces respectively, lb_j and ub_j are lower and upper box limits for the j-th design parameter. The verification proceeds as follows:

(1) Construct two U-bends: original and perturbed, see Fig. 4.
(2) Sample both final B-spline surfaces with 12K pairs of (u_i, v_i) parametric coordinates. These parametric coordinates are later used in B-spline algorithms to evaluate the corresponding three-dimensional points $P_i(x)$ and T_i.
(3) Define an objective function as in Eq. (2).
(4) Declare the original design parameters (x) as independent variables of the system.
(5) Minimise the objective function by using the limited-memory BFGS optimisation algorithm with box constraints (L-BFGS-B) [27].

Before analysing the performance, the optimisation example was executed with the *traceless* forward mode and the *trace-based* reverse mode. The L-BFGS-B optimiser converged at the same point using the gradients provided with both modes of AD. This served as an additional step to validate the derivatives.

The performance of the differentiated OCCT sources has been analysed and compared to the original sources. The time required for a single geometry optimisation iteration (the objective function value and the corresponding derivatives) has been measured. All three differentiation modes of ADOL-C (the *traceless* forward mode and the *trace-based* forward and reverse modes) have been evaluated in the tests. It is important to note that the measurements related to the *trace-based* modes include the time for generating the trace (*tracing*), because we need to perform *tracing* on every iteration. The reason is that the OCCT geometry computation is a highly non-linear algorithm and a different computational path may be traversed at each design iteration. This requires a re-evaluation of the trace. The results, i.e. quantitative comparisons of the average timings and run-time ratios, where the derivatives have been computed in one direction (*scalar* mode) as well as in 96 directions (*vector* mode), are shown in Table 5 and Table 6,

Table 5. U-bend single optimisation iteration timings with original and differentiated (AD) sources with number of directions $p = 1$ (*scalar* mode)

	Original sources	Traceless forward	Trace-based forward	Trace-based reverse
Avg. time (second)	0.421	1.26	4.726	4.66
Run-time ratio		2.99	11.23	11.07

*The results are based on 10 measurements. Tracing time: 3.9s – run-time ratio: 9.25.

Table 6. U-bend single optimisation iteration timings with original and differentiated (AD) sources with number of directions $p = 96$ (*vector* mode)

	Original sources	Traceless forward	Trace-based forward	Trace-based reverse
Avg. time (second)	0.421	12.597	18.132	4.654
Run-time ratio		29.92	43.07	11.05

*The results are based on 10 measurements. Tracing time: 3.9s – run-time ratio: 9.25.

respectively.

According to the theory [8], the run-time ratio between the derivative computation in the forward mode of AD and the function (primal) evaluation should be in range $[1 + p, 1 + 1.5p]$, where p is the number of directions. Furthermore, the run-time ratio between the derivative computation in the reverse mode of AD and the function (primal) evaluation should be in range $[1 + 2q, 1.5 + 2.5q]$, where q is the number of adjoints. In our test example $q = 1$, therefore the expected range is [3, 4].

Comparing this with the results in Table 5 and Table 6, one can state that the differentiated OCCT sources yield run-time ratios that are even below the theoretical lower range boundaries. One reason for this is that the derivation of the theoretical bounds assumes a rather pessimistic runtime ratio for nonlinear univariate operations. Therefore, much better run-time ratios achieved with the *traceless* forward mode might be connected to the limited use of these costly operations by OCCT. Alternatively, one might also assume that compiler optimisation could be a reason for this good run-time ratio. However, a similar effect, i.e. a better run-time ratio than predicted by the theory, is observable also for the *trace-based* forward mode where compiler optimisation is not available in a comprehensive fashion due to the used overloading approach. Finally, the reverse mode of AD obtains a 63% improved efficiency in contrast to the *traceless* forward mode of AD.

An overview of all run-time ratios evaluated in the range of 1 to 96 directions (with included timings for *tracing*), together with the memory requirements w.r.t. the maximal number of directions, is shown in Fig. 5. Additionally to the AD run-time ratios, the run-time ratios for finite differences are also shown. To evaluate the memory requirements (with $p = 96$), the same U-bend application has been profiled with the profiling tool *Massif* – which is a part of the *Valgrind* tools for debugging and profiling [15]. The sources have been compiled with two compilers: *g++ v4.8.5* and *clang++ v3.7.0*, showing similar results. To summarise, AD requires more memory than the finite difference approach, but it performs significantly faster, especially when using many directions.

5. Application of CAD-sensitivities in Aerodynamic Shape Optimisation

The main subject of this paper is the algorithmic differentiation of the CAD-kernel, hence only a brief overview of the gradient-based shape optimisation loop will be presented here,

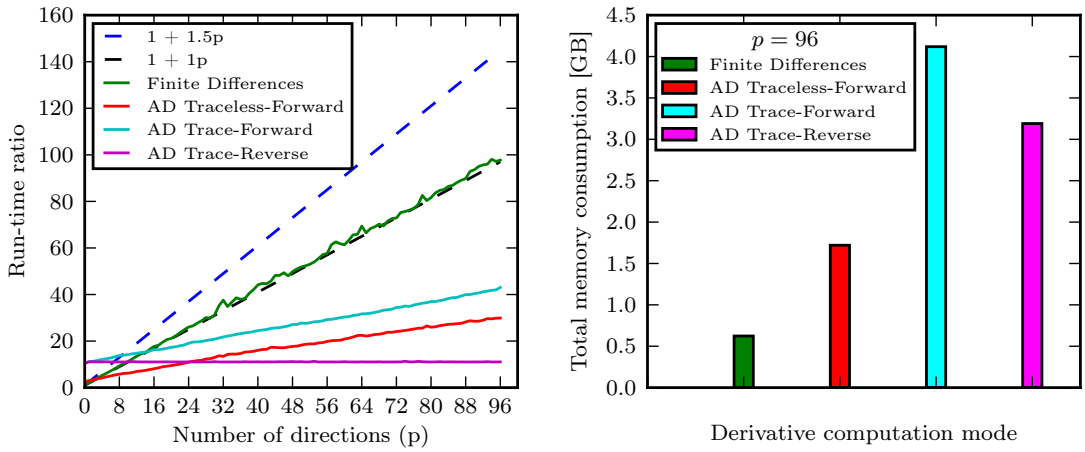


Figure 5. Summary of run-time ratios (left) and total memory requirements (right) for U-bend example

more details can be found in [25, 26]. The aerodynamic performance of a given CAD-geometry is usually described with a scalar objective function J such as drag, lift, or total pressure loss. We consider a minimisation problem for the aerodynamic objective with the CAD parameters α as design variables:

$$\min_{\alpha} \; J(U(X(\alpha)), X(\alpha), \alpha) \tag{3}$$

$$\text{subject to } R(U(X(\alpha)), X(\alpha)) = 0\,. \tag{4}$$

The constraint of the Reynolds-Averaged Navier-Stokes equations (4) $R = 0$, is a function of the state variables U and computational mesh coordinates X, which in turn depend on design parameters α. For a large number of design parameters, which is usually the case in industrial applications, the adjoint method proves to be computationally efficient and will be applied here. The application of the chain rule to the system Eq. (3)-(4) yields:

$$\frac{dJ}{d\alpha} = \left[\frac{dJ}{dX} + \nu^T f\right]\frac{\partial X}{\partial \alpha}\,, \tag{5}$$

where

$$f = -\frac{\partial R}{\partial X}$$

and ν represents the solution of the adjoint equation

$$\left(\frac{\partial R}{\partial U}\right)^T \nu = \frac{\partial J}{\partial U}\,. \tag{6}$$

After computing the solution of the primal and adjoint equations one can map the obtained sensitivity of the coordinates of the volume mesh X onto the mesh coordinates of nodes on the design surfaces X_S. For instance, using a spring-based analogy as the volume-surface deformation algorithm, the sensitivity in terms of perturbations of the

surface nodes becomes

$$\frac{dJ}{d\alpha} = \frac{dJ}{dX_S}\frac{dX_S}{d\alpha} \,. \tag{7}$$

The first term in Eq. (7), the *CFD sensitivity*, corresponds to the sensitivity of the objective function w.r.t. displacements of the surface grid points X_S, the second term, the *CAD sensitivity*, represents the geometric derivative of the surface grid points X_S w.r.t. design parameters. While the CFD sensitivity is evaluated using the computationally efficient adjoint method, we propose two possibilities of computing the total gradient $dJ/d\alpha$:

(1) Compute the CAD sensitivity independently by using the OCCT kernel differentiated in the *traceless* forward mode and couple both sensitivities at the end.
(2) Use the differentiated OCCT kernel in the *trace-based* reverse mode, thus having a full reverse mode differentiation of the complete differentiated design chain to compute the desired gradient.

The procedure of the second approach is that the CFD sensitivity has to be evaluated first and then propagated as a derivative seed vector using the reverse differentiated OCCT. Once this derivative seed vector is set, the ADOL-C driver function will use it to evaluate the total gradient. After obtaining the total gradient, one can use it in gradient-based optimisation loops:

$$\alpha^{(n+1)} = A\big(\alpha^{(n)}, \frac{dJ}{d\alpha}(\alpha^{(n)})\big) \,, \tag{8}$$

with A as an iterative optimisation algorithm.

6. U-bend Optimisation in a Complete Differentiated Design Chain

The optimisation methodology described in the previous section is applied to the U-bend CAD-model of Sec. 4.2. The complete case description can be found in [3, 22]. In this work, we investigate the single objective of decreasing total pressure loss between the inlet and the outlet pipe.

The CFD computations are performed by the in-house discrete adjoint solver STAMPS [2] developed at Queen Mary University of London. The U-bend is a rather challenging case for the compressible CFD solver, due to the very low flow speed exacerbated by the long inlet and outlet tails which are needed to establish fully developed flow. The mesh has approximately 170,000 cells.

At each optimisation step, the surface mesh is recalculated on the updated CAD geometry given by the OCCT kernel and then the surface displacements are propagated into the interior domain. The deformation of the volume mesh is computed using inverse distance weighting [24].

Two optimisation methods were applied: steepest descent (S-D) and L-BFGS-B. Although the first method is considered to be inferior in performance, its explicit control of the design steps made parametrisation updates and corresponding volume mesh movement more robust and hence was finally used to drive the optimisation.

The dominant flow phenomenon in the baseline geometry is the separation after the U-part and downstream in the outlet leg, which significantly adds to the total pressure loss. As shown in Fig. 6, the CAD-driven optimisation managed to reduce its size, resulting

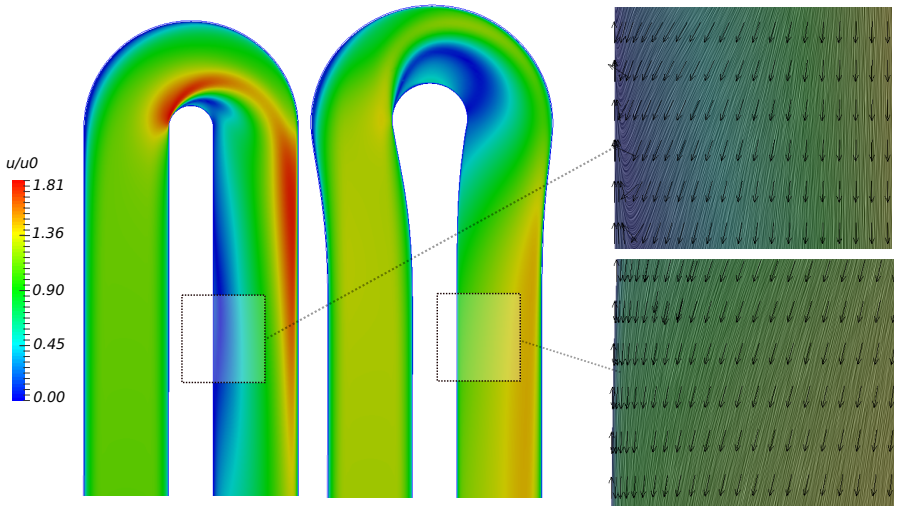
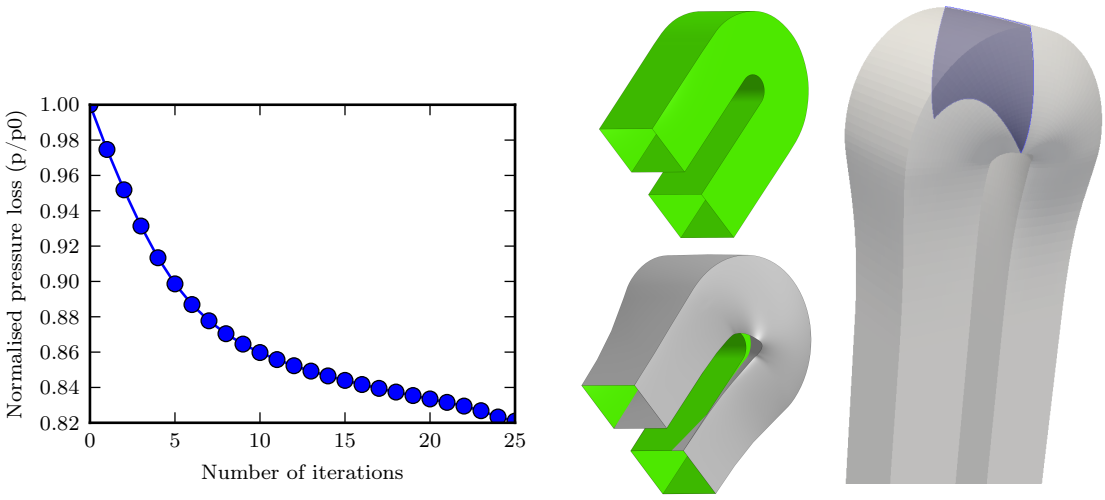


Figure 6. Left: Baseline and Optimised Mid-span Velocity Magnitude; Right: Flow Streamlines in the outlet leg



(a) Objective function (b) Initial (green) and optimised (gray) CAD model

Figure 7. U-bend optimisation results in a complete differentiated design chain

in much lower loss of total pressure. This is mainly due to the increase in inner radius of the bend and the cross-sectional area of the U-part.

The optimisation history, yielding 18% improvement, is shown in Fig. 7. The design iterations broke down at this stage because the mesh quality of the deformed mesh became too poor for the flow solver to converge. As we provide an exact CAD description of the geometry, we could remesh and run further optimisation steps. However, remeshing is currently a manual step and not included in the automated design algorithm. Improved methods for mesh deformation with better mesh quality are an active area of research in the field.

7. Conclusion

Algorithmic differentiation of the Open CASCADE Technology CAD kernel has been performed successfully, however requiring a significant amount of code modification. ADOL-C has been successfully integrated into OCCT, the correctness of the computed derivatives have been verified for the U-bend test-case, but the derivative validation of the rest of differentiated OCCT sources remains challenging. Certainly, some parts will be validated as we move on to the next test-cases. However, a more generic approach of testing the derivatives has to be investigated. There is a need for an effective and automated framework, an *Automated Testing System* which derives gradient tests from the regression tests available for the OCCT kernel.

The first results of using the reverse mode differentiation of OCCT show a significant reduction in the temporal complexity of the derivative computation. Compared to the *traceless* forward vector mode differentiation, we benefit in improved efficiency by 63%. There are some ways of code modification to make this even more efficient. This requires a complete understanding of the complex OCCT source code, at least the part used for the U-bend. Since ADOL-C is integrated into OCCT by the *typedef* approach, the first step is to reduce the total number of *adoubles* used in the optimisation process as much as possible. All variables that do not participate in the differentiation chain should have a *passive* declaration type. Next thing is to find linear solvers in the OCCT code and take them out of the differentiation process. Although this looks very straightforward, it is not as simple as one would expect, especially in the complex object-oriented source code like OCCT. This will be a part of our future research.

The differentiated CAD kernel has been coupled with a discrete adjoint flow solver also produced with algorithmic differentiation. The work demonstrates for the first time the differentiation of a complete design chain with generic, multi-purpose components which can be applied to a very wide variety of shape parametrisations expressed in CAD.

The effective application of the design chain is demonstrated by application to a U-bend turbo-machinery cooling channel. A typical, but complex CAD parametrisation for this type of geometry is used which starts from a two-dimensional curve formed by a set of Bézier curves in a cross-sectional plane. Then the three-dimensional shape is 'swept' along a pathline over the slices. The values of pathline and control parameters are defined along B-spline curves around the U-bend. Application of gradient-based optimisation to this geometry results in a 18% reduction of total pressure loss in the duct.

The computational time to construct the CAD geometry and to evaluate the corresponding sensitivities is negligible compared to the CFD part (primal and adjoint evaluation). On an average desktop computer configuration, where the U-bend optimisation presented in Sec. 6 takes a whole day, the total required computational time for the CAD part takes only several minutes.

The paper demonstrates that differentiation of complete CAD kernel is entirely feasible and can be applied to industrial cases. The inclusion of the CAD kernel in the design chain avoids the effort and errors associated with mesh-based methods of manual re-transcription of optimised shapes back into CAD. Application of AD to the CAD kernel rather than finite differences not only produces exact shape derivatives, but also strengthens the robustness of the method and reduces its computational cost. These improvements enable a step change in industrial shape optimisation with gradient-based methods.

Acknowledgements

The authors are very thankful to Sergey Slyadnev (Open CASCADE) for his support related to the AD part and the CAD itself, and Prof. Tom Verstraete (Queen Mary University of London) for his support related to the U-bend parametrisation.

This research is part of the *IODA* project - *Industrial Optimal Design using Adjoint CFD. IODA* is *Marie Sklodowska-Curie Innovative Training Network* funded by European Commission under Grant Agreement No. 642959.

References

[1] M.P. Bendsøe and O. Sigmund, *Topology optimization by distribution of isotropic material*, Springer, 2004.

[2] F. Christakopoulos, D. Jones, and J.D. Müller, *Pseudo-timestepping and verification for automatic differentiation derived CFD codes*, Computers & Fluids 46 (2011), pp. 174–179.

[3] F. Coletti, T. Verstraete, J. Bulle, T. Van der Wielen, N. Van den Berge, and T. Arts, *Optimization of a U-Bend for Minimal Pressure Loss in Internal Cooling Channels - Part II: Experimental Validation*, Journal of Turbomachinery 135 (2013), p. 051016.

[4] J. Dannenhoffer and R. Haimes, *Design Sensitivity Calculations Directly on CAD-based Geometry*, 53rd AIAA Aerospace Sciences Meeting, AIAA SciTech Forum (2015), AIAA 2015-1370.

[5] A. De Boer, M.S. Van der Schoot, and H. Bijl, *New method for mesh moving based on radial basis function interpolation*, in *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics, Egmond aan Zee, The Netherlands, September 5-8, 2006*, ECCOMAS, 2006.

[6] N.R. Gauger, A. Walther, C. Moldenhauer, and M. Widhalm, *Automatic differentiation of an entire design chain for aerodynamic shape optimization*, in *New Results in Numerical and Experimental Fluid Mechanics VI*, Springer, 2007, pp. 454–461.

[7] M.B. Giles, M.C. Duta, J.D. Müller, and N.A. Pierce, *Algorithm developments for discrete adjoint methods*, AIAA journal 41 (2003), pp. 198–205.

[8] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd ed., Society for Industrial Mathematics, 2008.

[9] R. Haimes and J. Dannenhoffer, *The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry*, 21st AIAA Computational Fluid Dynamics Conference, Fluid Dynamics and Co-located Conferences (2013), AIAA 2013-3073.

[10] L. Hascoët and V. Pascual, *The Tapenade Automatic Differentiation tool: Principles, Model, and Specification*, ACM Transactions On Mathematical Software 39 (2013).

[11] S. Jakobsson and O. Amoignon, *Mesh deformation using radial basis functions for gradient-based aerodynamic shape optimization*, Computers & Fluids 36 (2007), pp. 1119–1136.

[12] A. Jameson, *Aerodynamic design via control theory*, in *Recent advances in computational fluid dynamics*, Springer, 1989, pp. 377–401.

[13] A. Jameson and A. Vassberg, *Studies of alternative numerical optimization methods applied to the brachistochrone problem*, Computational Fluid Dynamics Journal 9 (2000).

[14] A. Jaworski and J.D. Müller, *Toward modular multigrid design optimisation*, in *Lecture Notes in Computational Science and Engineering*, Vol. 64, Springer, "New York, NY, USA", 2008, pp. 281–291.

[15] N. Nethercote and J. Seward, *Valgrind: a framework for heavyweight dynamic binary instrumentation*, in *ACM Sigplan notices*, ACM, 2007, pp. 89–100.

[16] *OpenCascade*, https://www.opencascade.com/, accessed 17 July 2017.

[17] O. Pironneau, *On optimum design in fluid mechanics*, Journal of Fluid Mechanics 64 (1974), pp. 97–110.

[18] T.T. Robinson, C.G. Armstrong, H.S. Chua, C. Othmer, and T. Grahs, *Optimizing parameterized CAD geometries using sensitivities based on adjoint functions*, Computer-Aided Design and Applications 9 (2012), pp. 253–268.

[19] S. Schmidt, E. Wadbro, and M. Berggren, *Large-scale three-dimensional acoustic horn optimization*, SIAM Journal on Scientific Computing 38 (2016), pp. B917–B940.

[20] T.W. Sederberg and S.R. Parry, *Free-form deformation of solid geometric models*, ACM SIGGRAPH computer graphics 20 (1986), pp. 151–160.
[21] M. Sonntag, S. Schmidt, and N. Gauger, *Shape derivatives for the compressible navierstokes equations in variational form*, Journal of Computational and Applied Mathematics 296 (2016), pp. 334–351.
[22] T. Verstraete, F. Coletti, J. Bulle, T. Vanderwielen, and T. Arts, *Optimization of a U-Bend for Minimal Pressure Loss in Internal Cooling Channels - Part I: Numerical Method*, Journal of Turbomachinery 135 (2013), p. 051015.
[23] A. Walther and A. Griewank, *Getting Started with ADOL-C*, in *Combinatorial scientific computing*, Chapman & Hall/CRC Computational Science, Dagstuhl Seminar Proceedings 09061, 2012, pp. 181–202.
[24] J. Witteveen and H. Bijl, *Explicit mesh deformation using inverse distance weighting interpolation*, 19th AIAA Computational Fluid Dynamics Conference (2009), AIAA 2015-3996.
[25] S. Xu, W. Jahn, and J.D. Müller, *CAD-based shape optimisation with CFD using a discrete adjoint*, Int. J. Numer. Meth. Fluids 74 (2013), pp. 153–68.
[26] S. Xu, D. Radford, M. Meyer, and J.D. Müller, *CAD-Based Adjoint Shape Optimisation of a One-Stage Turbine with Geometric Constraints*, ASME Turbo Expo 2015 2C: Turbomachinery (2015).
[27] C. Zhu, R.H. Byrd, P. Lu, and J. Nocedal, *Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization*, ACM Transactions on Mathematical Software (TOMS) 23 (1997), pp. 550–560.