To appear in *Optimization Methods & Software*
Vol. 00, No. 00, Month 20XX, 1–17

# Automatic Differentiation of the Open CASCADE Technology CAD System and its coupling with an Adjoint CFD Solver

Mladen Banovic[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*)

Automatic Differentiation (AD) is applied to the open-source CAD system *Open CASCADE Technology* using the AD software tool *ADOL-C* (*Automatic Differentiation by OverLoading in C++*). The differentiated CAD system 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:** Automatic Differentiation, CAD system, Adjoint CFD method, Gradient-based optimisation.

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

## 1. Introduction

Gradient-based optimisation methods are recognised for their computational efficiency, especially when considering cases with many control variables. However, industrial application of gradient-based methods for shape optimisation is not widespread, even though typical stochastic optimisation methods are limited to very coarse design spaces. In the past this was mainly due to the absence of robust and effective computational methods to compute the gradients for problems such as Computational Fluid Dynamics (CFD). Over the past decades adjoint methods [6, 10, 13] have emerged as the most effective methods to compute CFD gradients since they can compute gradients with respect to an arbitrary number of design variables at near-constant computational cost similar to the *primal* flow computation.

Today, industrial application of gradient-based optimisation is primarily limited by the immaturity of the parametrisation tools that define the design space. Furthermore, in gradient-based optimisation we not only need to evaluate the shape for a given set of design variables, but also their derivatives. So far, the derivative computation for a complete design chain, ranging from the parametrisation to the desired final target value, has only been demonstrated either with inaccurate finite differences [14] or with exact differentiation only for bespoke chains with limited versatility [5]. To obtain the gradient for the complete design chain, the calculation of shape sensitivities w.r.t. its parametrisation is required. In this paper we close this gap by demonstrating the differentiation of a generic CAD system, in this case the most widely used open-source CAD system Open CASCADE Technology. The results presented in this paper demonstrate the application of the resulting generic, complete design chain to an internal cooling channel of a turbomachinery blade. However, consisting of generic components, the chain can be used in a wide variety of applications.

### 1.1 *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 noisy shapes, regularisation of the surface is required, often chosen as implicit [11] or explicit [12]. Another popular approach is to define a global perturbation field using radial basis functions [4] or with 'Free Form Deformation' which interpolates the deformation of the domain within the control lattice of a volume spline [9, 15]. All of these approaches offer simple derivative computation, but only produce the optimal shape as a deformed mesh rather than a CAD surface.

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. [20, 21] use NURBS control points in the CAD-native boundary representation (BRep) as a design space. Their 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 of the standardised STEP format, therefore ignoring any design parametrisation defined in the CAD system. For computing the derivatives, the NSPCC geometry kernel has been differentiated using the source-transformation AD tool Tapenade [8].

Robinson et al. [14] use the design parametrisation defined in a closed-source CAD

system. Since the source code is not available, algorithmic differentiation can not be applied and they obtain the derivatives using finite differences. Step-widths must be carefully chosen to limit truncation error. To tackle possible changes in topology, the shape differences need to be computed between triangulations (STL) of the CAD surfaces. The significant advantage of this approach is that it allows to define the design parametrisation including geometric constraints through the CAD system.

Dannenhoffer and Haimes [3] use the open-source CAD kernel Open CASCADE Technology (OCCT). 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 geometry. They have also considered the automatic differentiation of the OCCT code, but did not implement it due to a large complexity of the sources.

As demonstrated in this paper, the automatic differentiation of OCCT is indeed feasible. For this purpose, we differentiated the OCCT kernel (*v7.0*) using a forward *traceless* mode and a reverse trace-based mode of the AD software tool ADOL-C (*Automatic Differentiation by OverLoading in C++*). For the first time, as this study explains, a fully developed CAD system has been differentiated. Considering other CAD-based methods, this one offers a number of significant advantages. As opposed to the finite difference approaches [14], the geometric derivatives are not affected by truncation errors but are exact. Furthermore, the computational efficiency of the method in most cases 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.

The paper is organised as follows. Section 2 explains the automatic differentiation of the OCCT CAD system. Section 3 briefly describes the U-bend parametrisation, which is our current test-case, and the verification of the differentiated OCCT, along with performance tests. Section 4 presents the governing equations for a flow problem and its adjoint, together with an assembly of the relevant derivatives. Section 5 shows an application of the complete differentiated design chain to gradient-based optimisation of the U-bend duct.

## 2. Automatic Differentiation of OCCT

### 2.1 *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 doesn't generate intermediate source codes [18]. 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. On the other hand, i.e., using the *traceless* option, the derivative computation is propagated 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* one allows so

far only the computation of first order derivatives. An additional facility of the *traceless* option is that it offers 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.

### 2.2 *Differentiating a Small OCCT kernel*

The ADOL-C library is integrated into a certain code 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. Otherwise, i.e., if the *adouble* chain is broken in some part of the code, the derivative values will be incorrect at the end. While facing a complicated *object-oriented* code like OCCT, the process of replacing the declaration types is not simple. The goal is to find a possible way (or ways) of *adouble* injection.

Before attempting to differentiate the OCCT kernel, a reduced kernel related to geometry was generated. The smaller the kernel is, the less amount of modification has to be introduced upon ADOL-C integration. Therefore, the minimal geometry kernel was a good starting point.

In order to make a proof of concept for the automatic differentiation of such a minimal kernel, we considered a geometrical entity class that corresponds to a two-dimensional B-spline curve - *Geom2d_BSplineCurve*. The idea was to differentiate the shape of a curve with respect to its properties/parameters:

- poles - collection of two-dimensional points (class: *gp_Pnt2d*),
- weights - collection of *double* values.

The first step was to create new classes:

- *gp_Pnt2d_AD* - the two-dimensional point class that has coordinates as *adoubles*,
- *Geom2d_BSplineCurve_AD* - the two-dimensional B-spline curve class that is almost an exact copy of the original one, with extra *AD* properties of poles and weights. Poles are collections of *gp_Pnt2d_AD* objects, while weights are collections of *adouble* objects.

Moreover, additional methods were developed. The shape of a curve is calculated in the *D0* method. Therefore, a new method was introduced inside the *Geom2d_BSplineCurve_AD* class - *D0_AD*. As the name implies, this method works with *AD* poles and weights. Starting from it, in the same manner, *adouble* injection had to be propagated to all dependant routines continuously - without breaking the chain rule. It is important to note here that these routines are not just in the B-spline class, but there are also external methods which are parts of B-spline evaluation algorithms.

This proof of concept was implemented successfully, but not used in the rest of the minimal geometry kernel due to large amount of code duplication. Another approaches of ADOL-C integration were considered.

### 2.3 *Approaches of Differentiating OCCT*

After the successful automatic differentiation of a small example, i.e., a two-dimensional B-spline curve, by code-duplication, other possibilities were investigated in order to decide on how to proceed in general. So far, there are several possible approaches with

respect to the code modification:

(1) Introduce *AD*-specialised properties and methods in original entity classes.
(2) Use an **inheritance** model to implement child classes of the original entities, defining extra *AD* properties and methods.
(3) Create completely isolated entry points designed for *AD* only - **controller** approach.
(4) Modify original entities in a way of **C++ templates**.
(5) **Typedef** approach.

One problem of the first two approaches is that they impose duplicate collections in the same object, which can be error-prone. For example, in the case of the B-spline curve object, it will have the original collections of poles and weights as well as the *AD* collections of poles and weights. Just replacing the original collections with the *AD* collections is not feasible because it will lead to compilation errors. The original signatures should remain untouched, otherwise one has to modify a lot of source files. Another issue that comes in mind is data consistency: if the original collections are modified, one has to make sure that at the same time the *AD* collections are updated as well. Without implementing some kind of synchronisation between these collections, any of the first two approaches is not a good choice because one has to track the original collections all the time. Although the **inheritance** model would look like a nice idea, it is not a good choice because it is not the real inheritance model:

- A child object will contain duplicate collections referring to the same thing.
- A child object will contain duplicate methods (e.g. the *D0* and *D0_AD* methods).
- Majority of methods from a parent class will not be used (e.g. in a case of B-spline curve, only the *D0* method is interesting for differentiation).

The **controller** approach introduces new lightweight classes - *controllers*, which are isolated entry points for *AD* only. Considering a two dimensional B-spline curve class, a new controller class *Geom2d_BSplineCurveADController* will hold:

- A pointer to the original curve (entity).
- The *AD* collections as properties, together with corresponding methods for getting and setting them (also known as *getters* and *setters*).
- Certain methods, like the *D0* (almost exact copy of the original one), which are defined for dealing with *adoubles*.

An additional task that has to be done for all three approaches explained so far is implementing new external methods, i.e., the ones that are used in the recursion tree of the *D0* method (members of *B-spline Curve Library*), which can deal with *adoubles*. Therefore, code duplication risk comes here as well. It could be reduced for some cases by reorganising and modifying original OCCT sources in a generic (*template*) way. Before doing so, one does not know how many methods except *D0* would have to be differentiated at the end and how deep one must dig into the sources to implement all extra *AD* methods in order to propagate the *adouble* injection wherever necessary. It would be very hard and time consuming to maintain such a code.

In order to avoid the code duplication as much as possible, one could consider modifying the OCCT sources in a way of **C++ templates**. This approach is ideal from the maintenance point of view. There would be no additional *AD* classes introduced and the switching would be done in original classes just by specifying a template type, i.e.:

- *Geom2d_BSplineCurve<double>*,
- *Geom2d_BSplineCurve<adouble>*.

Although this idea looks perfect, the work behind it is enormous - the full OCCT source code has to be changed (*templated*). The code duplication risk is minimal, but still exists, because *adouble* objects can not just fit everywhere. There are a lot of places in the code, some of which will be mentioned later, where one must interfere and add/modify some code lines in order to make everything work. For such cases, one has to use *template specialisation*, which is somehow similar to function overloading and allows the user to write specific method implementations that will be used for *adoubles*. In most of the cases, these specific implementations will not be that much different from the original ones, which is why they impose a certain code duplication. Furthermore, it is important to note that these methods have to be maintained together with the templated code.

The **typedef** approach is the most intrusive way of integrating ADOL-C into OCCT and it was chosen as the way to proceed with differentiating the OCCT sources. The idea is to replace all *doubles* by *adoubles*, by using an existing *typedef* which is named *Standard_Real* in OCCT. The main advantage of this approach is that code modification should be as minimal as possible, while the drawback is about sacrificing memory and efficiency to some extend because all *double* variables, even the ones that are not needed for the automatic differentiation, will become *adouble* objects.

Although the idea about the *typedef* approach looks simple, it is not as straightforward as one would expect. The differentiation 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. 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 $2\times$*Standard_Integer* (which is the *typedef* for *int* type). Once the *typedef Standard_Real* corresponds to *adouble*, the static assertion fails, therefore giving a compile-time error. The reason of having such an assertion lies in a definition of the *union* of two members {*Standard_Real*, $2\times$*Standard_Integer*} (*union* is a special class type in C/C++). Therefore, we had to leave *double* there because *adouble* couldn't fit. To overcome compile-time issues, this involved a certian modification of other sources that used the *union* in order to create a sort of bridge between *doubles* and *adoubles*. Fortunatelly, the *union* is used only in the low level of OCCT that is not related to the modeling algorithms.
- Hundreds of places in the OCCT code involve explicit/implicit conversion of *Standard_Real* to *int* type. In the terms of automatic differentiation, this shouldn't be allowed because there is a risk of breaking the chain rule and obtainining wrong derivatives at the end. Being aware of the risk, we allowed it simply by using the *getValue* method on the *adouble* object (which extracts the *double* value) wherever necessary. Otherwise, a detailed investigation has to be performed in order to understand why the conversions are necessary and whether they affect the chain rule.
- 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. Again, this means that there is a risk of breaking the chain rule and that is why the derivatives have to be carefully verified.
- Functions for printing to an output, like *sprintf*, can not accept *adouble* as an argument. Therefore, we used the *getValue* method here as well. The *sprintf* function is used in a lot of places in OCCT, mostly referred to printing standard CAD output formats like STEP and IGES.

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

- The left and right shift operators are overloaded in the *adouble* class. Since they are also used in the OCCT output system, the files were corrupted by *adoubles*. The solution was again to use the *getValue* method wherever necessary.
- C dynamic memory allocation is used in the OCCT kernel. This causes errors once the *adoubles* are present. In order to fix it, 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 couldn't differentiate, as described in Sec. 3.
- In many places of the OCCT code, we had to use the explicit conversion from *real* numbers to *adoubles*, if that numbers are passed as arguments to the specific overloaded methods. For example, consider the *SetCoord* method which is overloaded such that there are two possibilities: *SetCoord(Standard_Integer, Standard_Real)* and *SetCoord(Standard_Real, Standard_Real)*. A case in the code where developer uses the second method with *real* numbers, e.g., *SetCoord(6., 8.)*, does not work in the differentiated sources because the compiler calls the method with primitive types as arguments, which is the first method (at least it has one primitive type - *Standard_Integer*). Without any interaction this might work, but certainly it is wrong. The correct way is to add an explicit conversion wherever necessary, 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 reverse mode of AD, and currently, its integration in the basic version is completed. 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 *trace-based* reverse mode 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.

## 3. Verification of Differentiated OCCT with U-bend Test-case

### 3.1 *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 Figure 1.

The parametrisation is done on the U-part; it is based on an approach of achieving the three-dimensional volume by lofting through cross-sections: the final B-Spline surfaces are constructed by taking as inputs the $n$ curves/slices generated along a guiding path-line. Each 2-D slice lies on a plane which is orthogonal to the pathline. The pathline is described as a B-Spline curve. The slice consists of 4 Bezier curves with 4 control points each, forming a closed 'wire' with a total of 12 control points. Each control point position within the 2-D slice has its own law of evolution along the pathline. Therefore, parameters of the laws of evolution are actual design parameters considered in the optimisation.

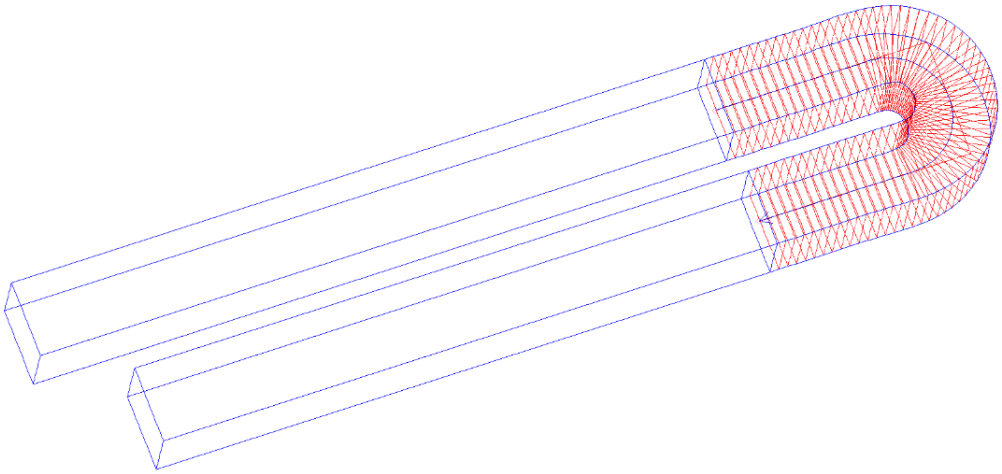


Figure 1. U-bend geometry

Table 1. OCCT *Automated Testing System* final results

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

### 3.2 *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 Section 2, a large number of run-time errors had to be resolved during the testing phase. Only a very small number of issues could not be resolved, see Table 1 for details. The majority of the tests marked *failed* relates 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 occur. The only solution here is to rewrite the package such that it follows the C++ standards for memory management.

In the next step, we verify the correctness of the computed derivatives. As a representative example, we compare here the sensitivities with respect to one design parameter calculated by the *traceless* forward mode of AD and Finite Differences, see Figure 2. The overall magnitude plots illustrate that these results coincide to a very high extent, the quantitative comparison between AD and Finite Differences for the same design parameter is shown in Table 2. 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 is how to not just automatically produce the derivative code, but also to derive relevant derivative regression tests from existing primal tests.

In order to validate the reverse mode differentiation of OCCT against the forward mode of AD, we have developed an optimisation example that is related only to the

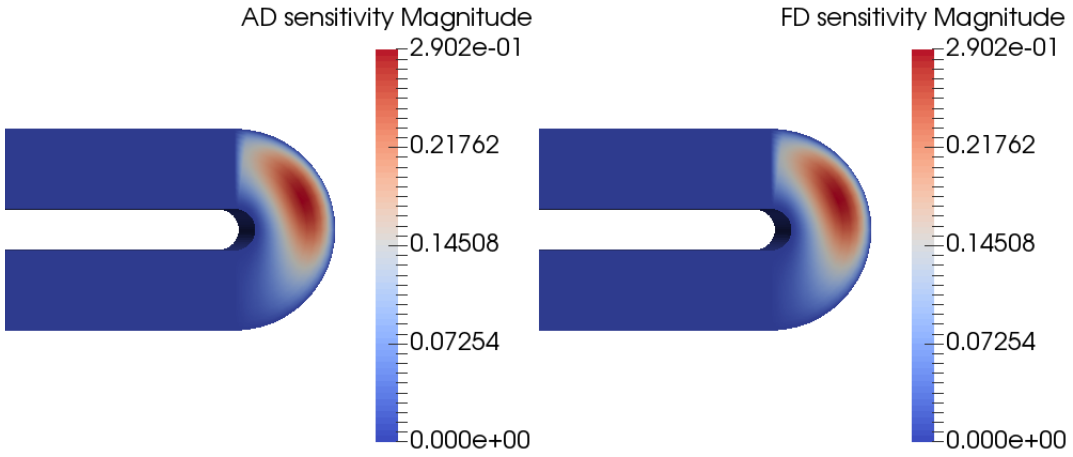



Figure 2. U-part sensitivities evaluated by AD (left) and Finite Differences (right)

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

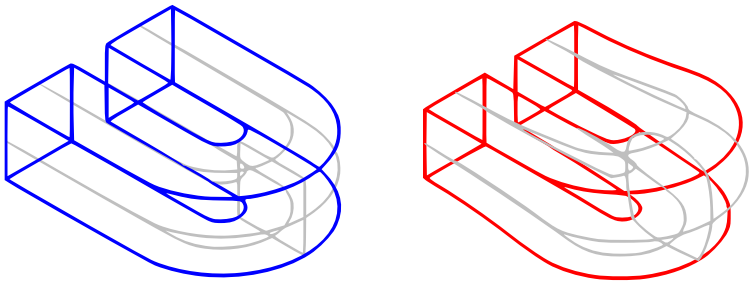
| AD value | FD value | Abs. difference |
|---|---|---|
| 1.823811578e-06 | 1.823818874e-06 | 7.295247578e-12 |
| 2.086595608e-05 | 2.086587847e-05 | 7.760991162e-11 |
| 0.009633046803 | 0.009633047311 | 5.084421235e-10 |
| 0.1895715499 | 0.1895715552 | 5.274124776e-09 |
| ⋮ | ⋮ | ⋮ |
| 0.2550856811 | 0.2550857767 | 9.561067155e-08* |

*Maximal difference in the current design parameter.

CAD system. The optimisation example is organised as follows:

(1) Construct two U-bends: original and perturbed one, see Figure 3.
(2) Sample final B-Spline surfaces with 12K pairs of $(u, v)$ parametric coordinates. These parametric coordinates are later used in B-Spline algorithms to evaluate the corresponding three-dimensional points $(x, y, z)$.
(3) Define an objective function as the sum of squared distances of all $(x, y, z)$ point pairs.
(4) Declare the original design parameters as independent variables of the system.
(5) Minimise the objective function by using the limited-memory BFGS optimisation algorithm with boundary constraints (L-BFGS-B) [22].
(6) Run the whole optimisation twice to compare two versions of differentiated OCCT kernel - one with *traceless* and the other one with *trace-based* ADOL-C headers.

This example of matching two U-bends has validated the derivative computation for both modes of AD, the L-BFGS-B optimiser converged in both cases at the same point. It is important to note here that during the optimisation, we observed small differences between the gradients, visible in Figure 4, while the quantitative comparison is shown in Table 3. The reason for such a behaviour lies in a fact that some overloaded operators of ADOL-C differ between the *trace-based* and *traceless* options. It is not just for the derivative calculation, also the primal evaluation is affected in the same order of magnitude. Considering that, along with numerical noise and machine precision, we can accept these differences. Furthermore, they did not affect the L-BFGS-B algorithm even if it



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

Figure 3. CAD optimisation with two U-bends

Table 3. Gradients comparison for initial U-bend optimisation iteration

| Traceless forward mode of AD | Reverse mode of AD | Abs. difference |
|---|---|---|
| 0.536097611674074 | 0.536097611674079 | 5e-15 |
| 0.021153242018414 | 0.021153242018392 | 2.2e-14 |
| 3.540316673464497 | 3.540316673464540 | 4.3e-14 |
| 4.570581137688175 | 4.570581137688300 | 1.25e-13 |
| ⋮ | ⋮ | ⋮ |
| 6.892393505252683 | 6.892393505251692 | 9.91e-13* |

*Maximal difference.

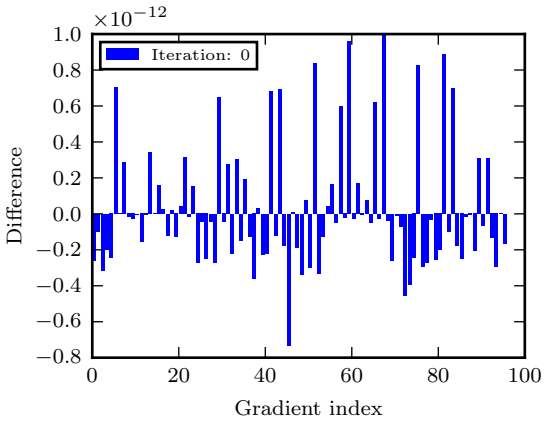



Figure 4. Gradient differences between two modes of AD for initial U-bend optimisation iteration

took a large step in the beginning. The optimiser converged using the gradients provided with both modes of AD.

### 3.3 *Performance tests*

The performance of the differentiated OCCT sources has been analysed using the U-bend optimisation example. The time required for a single geometry optimisation iteration (the objective function value and the corresponding derivatives) has been measured.

Table 4. U-bend single optimisation iteration timings with original and differentiated (AD) sources

| | Original sources | Forward mode - 1 dir. | Forward mode - 96 dir. | Reverse mode |
|---|---|---|---|---|
| Avg. time (second) | 0.421 | 2.053 | 12.839 | 5.613 |
| Run-time ratio | | 4.88 | 30.5 | 13.33 |

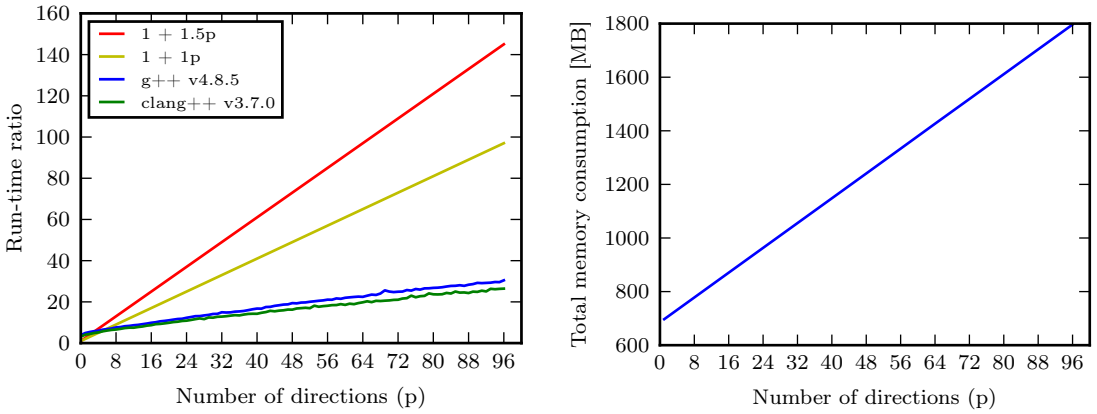
*The results are based on 10 measurements.

Regarding differentiated OCCT sources in the *traceless* forward mode, the derivatives have been computed in one direction (*scalar* mode) as well as in 96 directions (*vector* mode), which is the total number of design parameters. The average timings and the run-time ratios between the differentiated and the original sources are shown in Table 4.

By the theory [7], the run-time ratio between the derivative computation and the function (primal) evaluation should be in range $[1 + p, 1 + 1.5p]$, where $p$ is the number of directions. Comparing this with the results in Table 4, one can state the following: for the *traceless* forward *scalar* mode (1 direction) the run-time ratio is approximately equal to 5, which is around two times higher than the theoretical expectations; whereas in the case of 96 directions (*traceless* forward *vector* mode), the run-time ratio is approximately equal to 31, which is below the theoretical lower range boundary $(1 + p) = 97$. Then the run-time ratio of the reverse mode of AD is even better, where we have a 56% improved efficiency.

Further investigations were done to figure out why the run-time ratio of the *traceless* forward *scalar* mode is relatively high. The first assumption was related to the way of how overloaded operators are written: for each operator, there is a *for loop* that iterates $p$ times in order to compute the derivative in each direction. Since in the *scalar* mode $p$ equals 1, the *for loops* are actually not necessary. Therefore, the *for loops* have been removed from each overloaded operator of *traceless* ADOL-C and the run-times have been measured again. The results show that this modification of ADOL-C code does not make any significant difference, the run-time ratio for the *scalar* mode has been improved only a few percents. Hence, the code modification was discarded. Another interesting aspect is the time required just for the overloaded operators in the primal code, without any derivative computation. For such a purpose, ADOL-C provides a flag in the *traceless* headers named *zero-order scalar*. Having this flag set, a single iteration of the U-bend optimisation example has been executed with the total average run-time of 1.624s. Considering the data shown in Table 4, the *zero-order scalar* run-time actually takes 79% of the total *scalar* mode run-time, which means that just calling the overloaded operators is very expensive in terms of run-time yielding already a run-time ration of 4. This price then pays off if one uses the *traceless* forward *vector* mode in many directions.

An overview of run-time ratios evaluated in the range of 1 to 96 directions is shown in Figure 5(a). The red and yellow lines represent theoretical expectations, while the blue and green lines represent actual measurements with the sources compiled using different compilers. The sources compiled with *clang++* have performed faster. Furthermore, the same application for the U-bend construction has been profiled in the same range of number of directions with the profiling tool *Massif* (which is a part of the *Valgrind* tools for debugging and profiling). The memory requirements with respect to the number of directions are shown in Figure 5(b). As expected, the memory consumption is linearly dependent on the number of directions.




(a) Overview of run-time ratios

(b) Memory profiling summary

Figure 5. Traceless forward mode: summary of run-time ratios and memory requirements (U-bend)

## 4. Application of CAD-sensitivities in Aerodynamic Shape Optimisation

The main subject of this paper is the automatic differentiation of the CAD-system, hence only a brief overview of the gradient-based shape optimisation loop will be presented here, more details can be found in [20, 21]. The aerodynamic performance of a given CAD-geometry usually could be described with a scalar objective function $J$ (drag, lift, total pressure loss, etc.). In order to optimise a given shape, we consider an the optimisation problem of the CAD parameters $\alpha$:

$$\min_{\alpha} \; J(U(X(\alpha)), X(\alpha), \alpha) \tag{1}$$

$$\text{such that } R(U(X(\alpha)), X(\alpha)) = 0\,. \tag{2}$$

The constraint of the Reynolds-Averaged Navier-Stokes equations (2) $R = 0$, with the state variables $U$ and computational mesh coordinates $X$, depends on design parameters $\alpha$. 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. (1)-(2) yields:

$$\frac{dJ}{d\alpha} = \left[\frac{dJ}{dX} + \nu^T f\right]\frac{\partial X}{\partial \alpha}\,, \tag{3}$$

where

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

and $\nu$ represents the solution of the adjoint equation

$$\left(\frac{\partial R}{\partial U}\right)^T \nu = \frac{\partial J}{\partial U}\,. \tag{4}$$

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

$$\frac{dJ}{d\alpha} = \frac{dJ}{dX_S}\frac{dX_S}{d\alpha} \,. \tag{5}$$

The first term in Eq. (5), 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) Integrate the reverse mode of AD into OCCT, thus having a full reverse mode differentiation of the complete differentiated design chain to compute the desired gradient.

The idea 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:

$$\alpha^{(n+1)} = A\big(\alpha^{(n)}, \frac{dJ}{d\alpha}(\alpha^{(n)})\big) \,, \tag{6}$$

with $A$ as an iterative optimisation algorithm.

## 5. U-bend Optimisation in a Complete Differentiated Design Chain

The optimisation methodology described in the previous section is applied to the above mentioned U-bend CAD-model. The complete case description can be found in [16] with the corresponding related research [2, 17]. In this work, we investigate the single objective of decreasing total pressure losses between the inlet and the outlet pipe.

The CFD computations are performed by the in-house discrete adjoint solver STAMPS (previously *mgOpt*) [1] developed at Queen Mary University of London. The U-bend is a rather challenging case for the compressible CFD solver, due to the long inlet and outlet tails which are needed to establish the fully developed flow, but convergence is affected by the wave reflection with long travel times between inlet and outlet. 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 [19].

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 move-

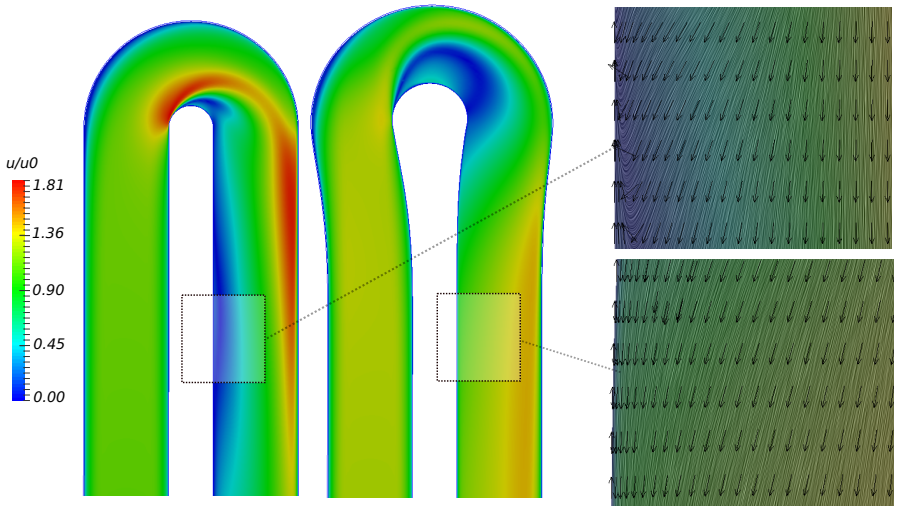



Figure 6. Left: Baseline and Optimised Mid-span Velocity Magnitude; Right: Flow Streamlines in the outlet leg

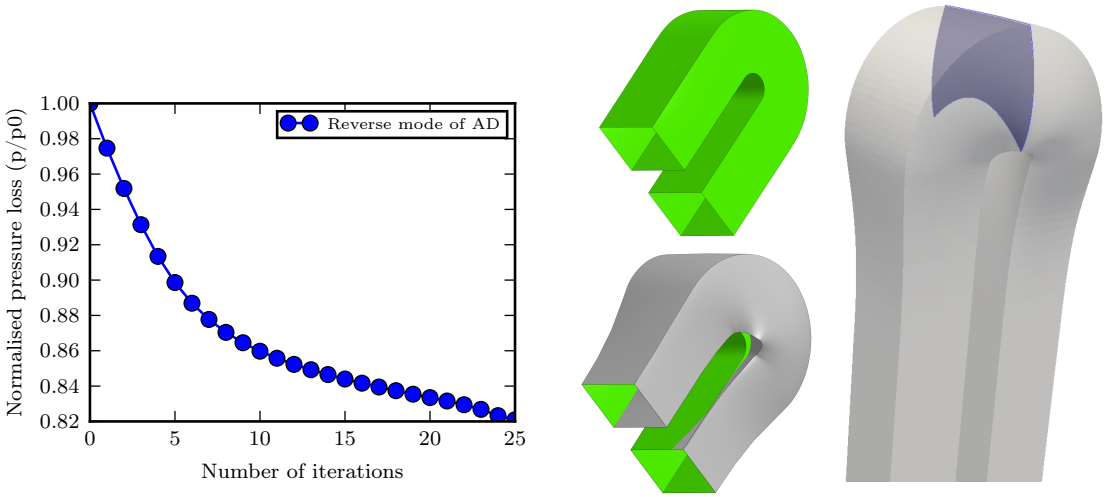
ment more robust and hence was finally used to drive the optimisation. On the other hand, potentially faster L-BFGS-B was taking too large steps in the initial iteration that generated a U-bend shape not suitable for the volume mesh movement. One solution, at least without major re-development of the standard optimisation libraries, is to limit the design space such that lower and upper boundary constraints have values very close to the original design parameters. Although a suitable U-bend shape is generated for the volume mesh movement, this solution has a major drawback: final result is not better than the one evaluated using the steepest descent due to this limitation on the design space.

A very particular feature of the flow in the baseline geometry is the separation after the U-part and downstream in the outlet leg, which is negatively influencing the optimisation objective. As shown in Figure 6, the CAD-driven optimisation managed to eliminate this, as well as the corresponding reverse flow motion present in the initial design. As discovered by the other investigations, the major increase in the U-bend inner radius and the diameter (width) of the U-part helps reducing adverse pressure gradients after the U-turn, consequently reattaching flow much faster.

The optimisation history, yielding 18% improvement, is shown in Figure 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 over 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.

## 6. Conclusion

The automatic differentiation of the Open CASCADE Technology CAD system involved a significant amount of code modification. Although ADOL-C has been successfully integrated into OCCT, we have verified the correctness of the computed derivatives only for the U-bend test-case. Therefore, the derivative validation of the rest of differentiated




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

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

OCCT sources still 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. One possibility is to build a test suite that would be based on the OCCT primal *Automated Testing System* (mentioned in Sec. 3).

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 mode differentiation, we benefit in improved efficiency by 56%. 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 system has been coupled with a discrete adjoint flow solver also produced with automatic 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 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 4 Bezier curves in a cross-sectional plane defined by 12 control points. The three-dimensional shape is 'lofted' along a pathline over the slices. The values of pathline and control parameters are defined along B-spline curves around the U-Bend. In total, there are 96 design variables. Application of gradient-based optimisation to this geometry results in a 18% reduction of total pressure loss in the duct.

To give an overview about timings, the computational time of CAD geometry construc-

tion and 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. 5 takes a whole day, the total required computational time for the CAD part can be expressed in minutes.

The paper demonstrates that differentiation of entire CAD system is entirely feasible and can be applied to industrial cases. The inclusion of the CAD system 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 system rather than finite differences not only produces exact shape derivatives, but also improves 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 Mr. Sergey Slyadnev (Open CASCADE) for his support related to the AD part and the CAD itself, and Dr. 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] F. Christakopoulos, D. Jones, and J.D. Müller, *Pseudo-timestepping and validation for automatic differentiation derived code*, Computers and Fluids 46 (2011), pp. 174–179.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

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

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

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

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

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

[12] 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.

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

[14] 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.

[15] T.W. Sederberg and S.R. Parry, *Free-form deformation of solid geometric models*, ACM SIGGRAPH computer graphics 20 (1986), pp. 151–160.

[16] T. Verstraete and J.D. Müller, *Aboutflow testcase: VKI U-bend*, www.aboutflow.sems.qmul.ac.uk/events/munich2016/benchmark/.

[17] 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.

[18] 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.

[19] J. Witteveen and H. Bijl, *Explicit mesh deformation using inverse distance weighting interpolation*, 19th AIAA Computational Fluid Dynamics Conference (2009), AIAA 2015-3996.

[20] 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.

[21] 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).

[22] 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.