Automatic Differentiation of a CAD System applied in an Adjoint CFD Method

Mladen Banovic, Orest Mykhaskiv, Salvatore Auriemma
Andrea Walther, Jens-Dominik Müller, Herve Legrand

mladen.banovic@math.uni-paderborn.de, o.mykhaskiv@qmul.ac.uk,
salvatore.auriemma@opencascade.com

15th September, 2016
AD2016, Christ Church Oxford, UK
CAD-based shape optimisation

Given a CAD-model with design parameters $\alpha$

Gradient-based optimisation

$$\min_{\alpha} J(U(X_s(\alpha)), X_s(\alpha), \alpha)$$

(1)\[R(U(X_s(\alpha)), X_s(\alpha)) = 0\]

(2)\[\frac{dJ}{d\alpha} = \frac{dJ}{dX_s} \frac{dX_s}{d\alpha}\]

(3)\[\begin{align*}
\text{CFD sensitivity } \frac{dJ}{dX_s} : \text{ Adjoint method (efficiency)} \\
\text{CAD sensitivity } \frac{dX_s}{d\alpha} : \text{ Forward Automatic Differentiation}
\end{align*}\]
OpenCASCADe Technology

OpenCASCADe Technology (OCCT) is an open source C++ library, consisting of thousands of classes and providing solutions in the areas of:

- Surface and solid modelling: to model any type of object,
- 3D and 2D visualization: to display and animate objects,
- Data exchange (import and export standard CAD formats) and tree-like data model.
Testcase under investigation: U-bend pipe

Parametrisation - done in the U-part

- Based on a cross-sectional design approach, **lofting**, which takes N-slices as inputs in order to construct a final surface.
Testcase under investigation: U-bend pipe

Parametrisation - done in the U-part

- Based on a cross-sectional design approach, lofting, which takes N-slices as inputs in order to construct a final surface.
- The generic slice consists of 12 control points and each control point is characterized by a law of evolution along the pathline.
Testcase under investigation: U-bend pipe

Parametrisation - done in the U-part

- Based on a cross-sectional design approach, **lofting**, which takes N-slices as inputs in order to construct a final surface.
- The generic slice consists of 12 control points and each control point is characterized by a law of evolution along the pathline.
- In particular, every law is B-spline curve whose control points coordinates are the **design parameters** (96) of the simulation.
Calculating the CAD sensitivities (I)

In order to calculate the shape derivatives w.r.t. design parameters, OCCT has been differentiated by integrating the AD tool ADOL-C.
Calculating the CAD sensitivities (I)

In order to calculate the shape derivatives w.r.t. design parameters, OCCT has been differentiated by integrating the AD tool ADOL-C.

Automatic Differentiation by OverLoading in C++

ADOL-C uses operator overloading concept to compute first and higher derivatives of vector functions that are written in C or C++.

<table>
<thead>
<tr>
<th>Options</th>
<th>Differentiation modes</th>
<th>Integrated to OCCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace-based</td>
<td>forward, reverse</td>
<td>X</td>
</tr>
<tr>
<td>traceless</td>
<td>forward</td>
<td>✓</td>
</tr>
</tbody>
</table>

Traceless forward differentiation

Computes first order derivatives. The derivative computation is propagated directly on the function evaluation.
Traceless forward vector mode

Operator overloading example

class myadouble{
    double value;
    double *ADvalue = new double[NUMBER_OF_DIRECTIONS];
    // multiplication ...
    inline myadouble operator * (const myadouble& a) const {
        myadouble tmp;
        tmp.value = value * a.value;
        for(size_t i = 0; i < NUMBER_OF_DIRECTIONS; ++i)
            tmp.ADvalue[i] = ADvalue[i] * a.value + value * a.ADvalue[i];
        return tmp;
    }
};

Description

- **NUMBER_OF_DIRECTIONS** = number of design parameters.
- Derivatives w.r.t. all design parameters are evaluated with just one code run.
OCCT Differentiation (I)

All variables that may be considered as differentiable quantities must be of an active type, which is named `adouble` in ADOL-C.

Possible ways of code modification

1. Introduce AD-specialized properties and methods in original entity class.
2. Use the inheritance model to create a child class from every original entity class, defining extra AD properties and methods.
3. Create a class which holds a pointer to the original entity class + extra AD properties and methods - controller approach.
4. Modify original entities in a way of C++ templates.
5. Typedef approach - succeeded!
OCCT Differentiation (I)

All variables that may be considered as differentiable quantities must be of an active type, which is named adouble in ADOL-C.

Possible ways of code modification

1. Introduce AD-specialized properties and methods in original entity class.
OCCT Differentiation (I)

All variables that may be considered as differentiable quantities must be of an active type, which is named adouble in ADOL-C.

Possible ways of code modification

1. Introduce AD-specialized properties and methods in original entity class.

2. Use the inheritance model to create a child class from every original entity class, defining extra AD properties and methods.
OCCT Differentiation (I)

All variables that may be considered as differentiable quantities must be of an active type, which is named adouble in ADOL-C.

Original code ─ adouble injection ─ Differentiated code

Possible ways of code modification

1. Introduce AD-specialized properties and methods in original entity class.
2. Use the inheritance model to create a child class from every original entity class, defining extra AD properties and methods.
3. Create a class which holds a pointer to the original entity class + extra AD properties and methods - controller approach.
OCCT Differentiation (I)

All variables that may be considered as differentiable quantities must be of an active type, which is named adouble in ADOL-C.

Possible ways of code modification

1. Introduce AD-specialized properties and methods in original entity class.
2. Use the inheritance model to create a child class from every original entity class, defining extra AD properties and methods.
3. Create a class which holds a pointer to the original entity class + extra AD properties and methods - controller approach.
4. Modify original entities in a way of C++ templates.
OCCT Differentiation (I)

All variables that may be considered as differentiable quantities must be of an active type, which is named adouble in ADOL-C.

Possible ways of code modification

1. Introduce AD-specialized properties and methods in original entity class.
2. Use the inheritance model to create a child class from every original entity class, defining extra AD properties and methods.
3. Create a class which holds a pointer to the original entity class + extra AD properties and methods - controller approach.
4. Modify original entities in a way of C++ templates.
5. Typedef approach - succeeded!
Coding issues

There was a lot of code modification introduced because the adouble objects can not just fit everywhere, i.e.:

- Converting a real value to integer.
- Arguments for calling the external functions that are not part of OCCT must remain as doubles.
- Input/Output system: adouble objects can corrupt the files.
- Using C functions for memory management (malloc, free, memcpy, memset) will cause the run-time exceptions.
Coding issues

There was a lot of code modification introduced because the adouble objects can not just fit everywhere, i.e.:

- Converting a real value to integer.
- Arguments for calling the external functions that are not part of OCCT must remain as doubles.
- Input/Output system: adouble objects can corrupt the files.
- Using C functions for memory management (malloc, free, memcpy, memset) will cause the run-time exceptions.

Testing original (primal) OCCT functionality

Automated Testing System of OCCT has been executed after successful ADOL-C integration, with the final results:

<table>
<thead>
<tr>
<th>Tests marked:</th>
<th>OK</th>
<th>Tests marked failed:</th>
<th>FAILED</th>
<th>Success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>11,306</td>
<td></td>
<td>374</td>
<td></td>
<td>97%</td>
</tr>
</tbody>
</table>
CAD sensitivities

Example of derivatives calculated using the vector mode
Performance tests (I)

The average values of total computational time required for the U-bend construction.

\textbf{g++ compiler v4.8.5}

\begin{tabular}{l|ccc}
 & Original sources & AD sources - 1 dir. & AD sources - 96 dir. \\
\hline
Avg. time (sec) & 0.037 & 0.340 & 2.127 \\
Run-time ratio & 9.13 & 57.07 & \\
\end{tabular}

Primal with operator overloading: 0.269 sec; run-time ratio: 7.21.
## Performance tests (I)

The average values of total computational time required for the U-bend construction.

### g++ compiler v4.8.5

<table>
<thead>
<tr>
<th></th>
<th>Original sources</th>
<th>AD sources - 1 dir.</th>
<th>AD sources - 96 dir.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. time (sec)</td>
<td>0.037</td>
<td>0.340</td>
<td>2.127</td>
</tr>
<tr>
<td>Run-time ratio</td>
<td></td>
<td>9.13</td>
<td>57.07</td>
</tr>
</tbody>
</table>

Primal with operator overloading: 0.269 sec; run-time ratio: 7.21.

### clang++ compiler v3.7.0

<table>
<thead>
<tr>
<th></th>
<th>Original sources</th>
<th>AD sources - 1 dir.</th>
<th>AD sources - 96 dir.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. time (sec)</td>
<td>0.035</td>
<td>0.298</td>
<td>1.833</td>
</tr>
<tr>
<td>Run-time ratio</td>
<td></td>
<td>8.59</td>
<td>52.75</td>
</tr>
</tbody>
</table>

Primal with operator overloading: 0.216 sec; run-time ratio: 6.20.
Run-time ratios

- 1 + 1.5p
- 1 + 1p

- g++ v4.8.5
- clang++ v3.7.0
Memory requirements

Number of directions vs. Total memory consumption [MB]

- Memory requirements increase linearly with the number of directions.
- The graph shows a direct correlation between the number of directions and memory consumption.
CAD optimisation loop

- CAD, set $\alpha$
- Mesh on CAD
- Run primal CFD
- Run adjoint CFD
- CFD Sensitivity
- CAD sensitivity

Objective Function Sensitivity:

$$\frac{dJ}{d\alpha} = \frac{dJ}{dX_S} \frac{dX_S}{d\alpha}$$

**STAMPS solver**

- CFD Solver from QMUL
- Discrete Adjoint AD
- Spring/Elasticity Analogy
- Mesh Perturbation

**OCCT AD**

- Build Shapes ($\alpha$)
- Find Mesh on CAD
- Parametric Mesh ($u, v$)
- Sensitivities in ($u, v$)

**Optimiser**

- Steepest Descent
CAD optimisation loop

Gradient verifications
✓ Sensitivities in STAMPS and OCCT were verified with FD

Flow conditions
Incompressible flow, $U_0 = 8.8$, $Re=43830$, $\rho = 1.204 kg/m^3$, $\mu = 1.813 \times 10^{-5} kg/(sm)$, $P = 101300 Pa$ and $T = 293.15 K$
Optimisation results

Cost function:
Total pressure losses between inlet/outlet

Design improvement:
16%
Optimisation results (II)

Total pressure

Velocity change
Conclusions

**Current state**
✓ Differentiated full CAD-system
✓ Coupled with adjoint CFD
✓ CAD available at each iteration step

**Future steps**
✗ Validate derivative calculation of differentiated OCCT by building new test cases.
✗ Integrate the ADOL-C trace functionality into OCCT in order to use the reverse mode of AD at appropriate places.
Research is conducted within IODA\textsuperscript{1} project

*Industrial Optimal Design using Adjoint CFD*

\textsuperscript{1}http://ioda.sems.qmul.ac.uk/ Grant Agreement No. 642959.