# Introduction

## The state of CFD

## The problems of grid generation

## Structured vs. Unstructured grids

## Why UCS?

UCS provides several advantages over traditional CFD, such as better slip line resolution and streamline-oriented grids, but our interest in the system arises primarily from its automatic generation of a computational grid. Design optimization codes will produce many different design options (thousands), and there is a need for a low-level CFD program that is reasonably fast, reliably accurate, and requires little interaction on the part of the user to validate these designs. The automatically generated, streamline-aligned coordinate system UCS provides has the potential to satisfy these requirements, if it can be shown to work reliably and in an intuitive way for a wide range of simulations.

# The Unified Coordinates

## The history of UCS

### Hui 2-, 3-D

The unified coordinates were first used to solve time-dependent flows in 19991, where they were applied to the two-dimensional, inviscid, Euler equations for an ideal gas. In that paper, a Godunov scheme with a flux- limited MUSCL extension to second order was used to solve a variety of problems, including a steady Riemann problem, flow through a transonic duct, Mach reflection of a traveling shock wave, and an implosion/explosion problem. In 20012, a similar scheme was applied in three dimensions to the steady Riemann problem and to supersonic corner flow. Two-dimensional, inviscid, external flows around both steady and oscillating airfoils were presented later3. During its original development by Hui *et al*, it was found that uniform Cartesian grids could be generated at the upstream simulation boundary, and allowed to “flow” through the simulation region, automatically conforming to simulation boundaries and fitting itself to body surfaces, as shown in Fig. %%. Unsteady boundary conditions were also easily modeled with UCS, as in Fig. %%.

### Hui Viscous

UCS has been applied to more than just the Euler equations. An application to the full Navier-Stokes equations4 showed not only that UCS was capable of resolving viscous flows, but also that the system was robust enough to handle complex phenomena like shock-induced boundary-layer separation with recirculation, as shown in Fig. %%, thus answering one of the major concerns about its Lagrangian methodology. Additional models to which UCS has been applied include multi-material flows5, magneto-hydrodynamics6, and gas-kinetic BGK simulations of a freely falling plate7,8.

#### Space-marching Euler

### Avalanche modeling

## The UCS transformation

### Grid velocity

#### Hui’s methods

##### Lagrangian

##### Grid-angle-preserving

##### Jacobian-preserving

##### Skewness-preserving

#### Limitations to Hui’s work

##### Do not follow boundaries exactly

##### Does not pin grid points to singular points

##### Elastic “pressure-based” grid possibilities?

### Compatibility conditions

#### Derivation

#### Typical solution methods

##### Finite difference?

##### Boundary value methods?

##### Impacts on solution stability

#### The full UCS equations

##### Strong conservation form for flow variables and metric components

We here present the full transformation of the Euler equations to Cartesian coordinates, while maintaining Cartesian velocity components. In all of the following, implied sums over greek indices are to be taken over four-dimensional space-time, and sums over latin indices are to be taken over three-dimensional space. In three dimensions, the Euler equations are given by:



We wish to express in the unified coordinate system given by the transformation:



may be written more succinctly as , which naturally leads to  , which can be further written as:





where we have defined the following:



Geometric compatibility conditions are required in order for to be well-behaved. In particular, we require that partial derivatives commute:

This can be expanded to read (see also Hui9):



We now transform using :



If we multiply this by the Jacobian and apply the differential product rule, we find:



It can be shown that  is identically equal to zero10, eliminating one term. It is likewise possible to show that the final non-conservative term is zero. The general idea is to use the compatibility conditions to show that the term is identically zero. The details are as follows:

It is possible to write the inverse of a 3x3 matrix in terms of vector cross products:



If we rewrite:



It can easily be seen that  must be identically zero, by exploiting the anti-symmetry of the Levi-Civita tensor. We are therefore left with the conservative, fully transformed Euler equations in unified coordinates:



describes the behavior of the physical flow quantities in the unified coordinates. We know from , however, that time evolution equations also exist for the grid metric components. This is best handled by appending the time-dependent compatibility conditions to to yield an expanded equation set.

We use to define the total derivative of transformed coordinates:



We are now in a position to write the total transformed Euler equations in the unified coordinate system:



##### Source terms for physical coordinates

##### Free specification of grid velocity, as long as transformation remains well-behaved (J>0?)

### Assumptions

#### Eta, Zeta are material coordinates (Lagrangian-esque)

#### … at least mostly. Perhaps some kind of balance between material-ness of Eta, Zeta and distance from boundaries?

#### Or maybe better to just pin grid point motion at boundaries.

#### Xi is determined such that a well-behaved grid is obtained. In 2-D, grid-angle preserving is the preferred method.

### Solution algorithm

#### Time-step-Eulerian

#### Dimension Splitting vs. Finite-Volume methods

#### 1st-order Godunov solver

#### 2nd-order MUSCL update

#### Godunov dimensional splitting

##### Describe Godunov algorithm

#### Finite-Volume

##### Describe algorithm and rationale, as well as drawbacks.

### Specification of grid motion

#### Grid-angle-preserving

##### In two-dimensional flow, it is possible to force the grid to move in such a way that the angle of intersection between lines of constant %xi and lines of constant %eta is preserved. That is:



We take for two-dimensional flow, and becomes:



If we define we can write which lead to We rewrite :



Which leads to:



At this point, we apply the UCS compatibility conditions:



We must also here choose whether to solve the equation for grid velocity components directly, or to solve for grid velocity magnitude. We follow the magnitude approach here, and define:



and similarly for %eta. This allows us to write as



can then be solved using a number of techniques.

#### Jacobian-preserving

##### For three-dimensional flows, grid-angle-preserving is no longer as useful. Preserving the jacobian of the grid transformation is different technique that can be used for 3-D flows. We first define some useful variables:



We also require that %eta and %zeta be material coordinates:



We finally rewrite the compatibility conditions:

.



The equation for preservation of grid jacobian can then be written as:



Applying the compatibility conditions yields



We can differentiate to find:



These can be combined to yield:



#### Elastic boundary-conforming?

# Verification of Codes – Almost copy/paste from SciTECH

## Order of convergence verification

### Measure rate of convergence of code to complex, exact, solution

### Compare convergence rate with that expected based on algorithm.

### Highly sensitive to even subtle errors

## Method of Manufactured Solutions

### Exact solutions are rarely complex enough to yield useful verification tests.

### Boundary conditions for exact solutions are typically simplistic.

### MMS begins with a manufactured solution and an analytically computed source term

#### Manufactured solution can be as complex as desired, but must be differentiable.

#### Boundary conditions are arbitrary, based on the value of the solution at the boundaries.

#### Source term is computed using computer-aided algebra (CAS) systems and code generation tools.

## Integrative Method of Manufactured Solutions

### Numerical integration accuracy

### IMMS performance (Roy’s method)

### Multidimensional integration with discontinuities

## Verification of BACL-Streamer & UCS

### Choice of exact & manufactured solutions

### Verification of Euler solver

### Verification of UCS solver under various forms of grid motion

### Verification of BCs?

# Challenges of Moving Grids

## The equations are solved in unsteady, computational coordinates.

## The boundaries are defined in physical coordinates.

## How do you efficiently apply unsteady BC’s in this way?

### Convert physical BCs to computational coordinates

### Determine computational coordinates from grid indices (d-xi = 1)

### Requires conversion of BCs to computational coordinates, but no loop over boundary points.

## How do you accurately resolve singular points in the flow?

## How do you ensure accurate tracing of boundary surfaces?

# Code Iterations

## v.1.0

### Basic Fortran

### Variable-size grid handled using dynamic memory allocation and array copying.

### Code outline

## v.2.0

### Advanced Fortran

### Linked-list implementation of variable grid

#### Each node points to each other node

### Individual nodes represented as derived types

### Code outline

## BACL-Streamer

### Code design

#### Space & time looping

#### Mixed-Language Programming

#### Modularity

### Unit tests

#### Direct testing of expected outcome

#### Grid convergence studies

#### “Does routine \_\_X\_\_ do what I want it to do?”

### Algorithmic implementation

#### Dimensional splitting

#### U = h u

##### Lagrangian-esque grid motion

##### Grid-angle-preserving grid motion

### Code outline

# Conclusion