# 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 osition 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

##### Transformation to grid components

Godunov’s method for solving partial differential equations requires the solution of various one-dimensional Riemann problems, where the initial conditions and discontinuity are given by the interface between two adjoining cells. Therefore, in order to apply the Godunov method to solve, it is necessary to express the velocity vector in terms of components that are normal and tangential to the cell interface. If we define the vectors  as an orthonormal basis where  is normal to the cell interface, then we may write



Under this transformation, the tangential derivatives vanish at cell interfaces, and the remaining unsteady, one-dimensional equations become11:



For the sake of brevity, we examine only the first dimensional case, though the rest are similar. The non-conservative source terms are a result of the non-inertial velocity components, and are analogous to the centrifugal and coriolis force terms encountered in the physics of rotating coordinate systems. These source terms are non-zero only at the cell boundaries.

There is no general solution to unless  is constant across the cell boundary. It is therefore necessary to choose some suitable average to be applied at the cell boundary:



Both flow and grid velocity components are transformed using . Once transformed, the normal grid velocity component must also be averaged as before: . This averaged value is used both in the solution of the Riemann problem and in the computation of intercellular fluxes.

##### Eigensystem of governing equations

In order to compute the solution to the one-dimensional Riemann problem defined by and the initial conditions:



The first step is to compute the derivatives:





The eigenvalues can be computed as the solutions of the equation:



The corresponding eigenvectors are:



From these we may compute both the generalized Riemann invariants and the Rankine-Hugoniot relations.

##### Riemann invariants and rarefaction wave relations

The generalized Riemann invariants are relations that hold true across smooth waves, and are given by the relations12:



We begin with density:



Then normal velocity:



The tangential velocities are simple:



The rarefaction head and tail speeds are given by the eigenvalues :



It is finally necessary to account for pressure changes for points within the rarefaction wave. The slope of a characteristic for a rarefaction wave is:



can be combined with to solve for :



##### The Rankine-Hugoniot conditions and shock wave relations

The generalized Riemann invariants do not hold across discontinuous waves such as shocks. For such waves, the Rankine-Hugoniot conditions must be used12:



The derivation is somewhat tedious, but straightforward. The general idea is to use a Galilean velocity transformation to a frame where the shock speed is zero, and then solve the left hand side to find the relations between flow variables and the shock speed. This yields the relations:



##### Slip lines

The third type of wave is the linearly degenerate slip line. This discontinuity moves at the normal speed of the fluid, and pressure and normal velocity remain constant across the wave while density and tangential velocity may jump discontinuously.

##### The one-dimensional Riemann problem in the unified coordinates

The boundary between two adjacent cells can be represented as a one-dimensional Riemann problem, as in Figure %%. The Riemann problem consists of 3 waves: a central, linearly degenerate, slip line, across which pressure and normal velocity are constant while density and tangential velocity may jump discontinuously, and two nonlinear waves which may be either rarefaction waves or shocks, and across which

Using - and , it is possible to define a function of pressure representing the jump in normal velocity across the central slip line:

 r

 represents the normal velocity computed across the  wave given a central pressure  . This yields a nonlinear equation that can be solved for the pressure between the two nonlinear waves. From this, the rest of the flow variables can be computed directly. The solution of this nonlinear equation for pressure can be computed by iteration, and is the most computationally expensive step in the traditional Godunov method. The present work uses a Newton-Rhapson solver for this purpose. Approximate Riemann solvers, which do not depend on iterative solution schemes, offer substantial performance gains, but they are not discussed here.

#### Spatial accuracy and boundary interpolation 2nd-order MUSCL update

The Godunov method is inherently first-order accurate, but it is possible to boost the order of spatial accuracy using MUSCL interpolation to reconstruct the left and right boundary states used in the Riemann problem. In particular, for the boundary between the cells *i* and *i*+1, and for flow variable *w*, we have:



#### Godunov dimensional splitting

##### Hui1 uses a dimensional splitting technique to solve the multidimensional Euler equations. The algorithm is as follows:

* For each coordinate direction *n*:
  + For each cell *i*, *j*, *k*:
    - For each interface *+*, *-*:
      * Apply MUSCL reconstruction using .
      * Transform the flow and grid velocity to normal and tangential components using: 
      * Solve the Riemann problems as described in section %% to find the flow variables at each interface: 
      * Transform interface velocity back to Cartesian components using: 
    - Update coordinate-appropriate grid metric components: 
    - Compute interface fluxes using interface flow variables and central metric variables, e.g.:
    - Compute new conserved quantities  using and updated metric components
    - Update conserved variables using (e.g.):
    - Convert updated conserved variables to updated primitive variables.

#### Finite-Volume

The dimensionally split algorithm above suffers from two major drawbacks. First, it is difficult to choose adaptive time steps accurately. In the Godunov method, the temporal stability condition is dependent on the maximum wave speed present in the problem. This is known only after the solution of all the Riemann problems at all cell interfaces, so it is impossible to compute directly for dimensional splitting algorithms.

Second, and more importantly, the manner in which fluxes are computed using cell-specific metric components makes it impossible to enforce strong conservation in the algorithm. A better approach would be to rather implement a finite-volume algorithm, as follows. Unfortunately, the FV approach is computationally unstable for two- and three-dimensional problems under the Godunov method without special treatment.

* Compute all cell interface fluxes:
  + For each interface:
    - Perform MUSCL interpolation if applicable
    - Transform velocity vectors to normal & tangential components
    - Solve Riemann problem to find interface variables
    - Transform interface velocity back to Cartesian components
    - Compute interface flux vector using Riemann interface variables, average metric components, and average grid velocity
* Use maximum Riemann wave speed to determine maximum time step as:
* Compute conserved variables
* Update conserved variables using computed flux vectors
* Compute updated primitive variables

### 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