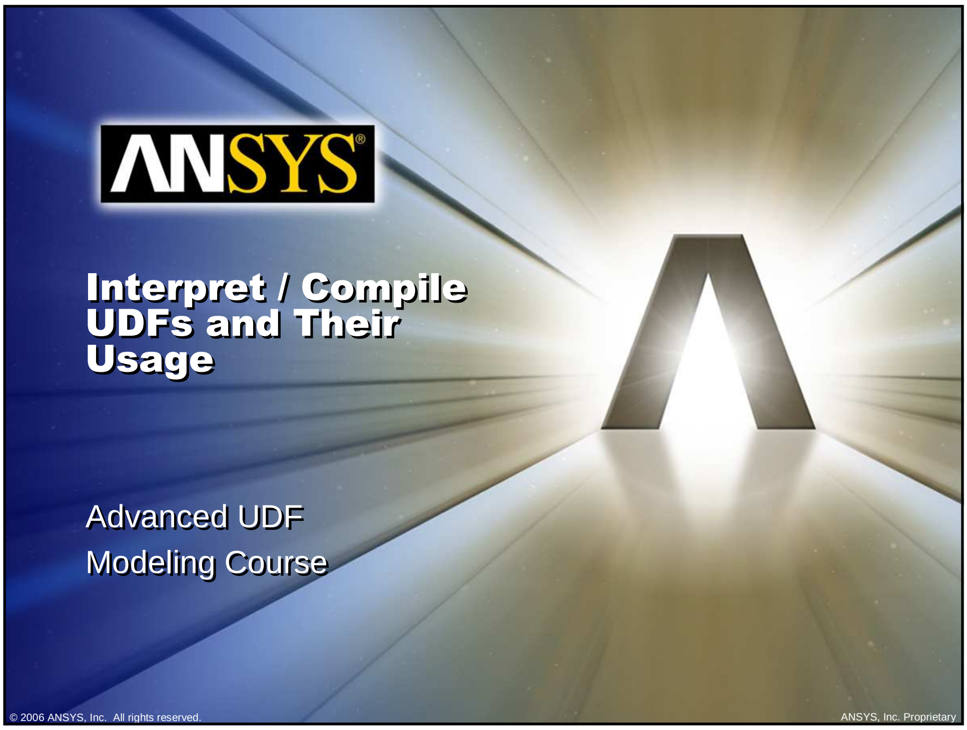


Interpret / Compile UDFs and Their Usage

Advanced UDF
Modeling Course

How to use the UDF

◆ First, we need to write and save the C-source file containing the appropriate **`DEFINE_MACRO`** routine(s).

◆ To use this file, the steps are:

1: Interpret / Compile the UDF
2: Start the solver (FLUENT) and read in your case/data files
3: Assign the UDFs in the BC and/or other panels for the appropriate zones
4: Set the UDF update frequency in the Iterate panel
5: Run the calculation as usual

◆ Note: Values obtained from and returned to the solver by UDFs must be in SI units

Interpreted Vs. Compiled Code

- UDFs can be ‘interpreted’ on-the-fly using the standard ‘GUI’
  - does not need a separate compiler and are architecture-independent
  - It translates the C-source to assembly language
  - Executes the code on line-by-line instantaneously
    - performs slower than compiled UDFs
  - The interpreter resides in the computer’s memory
    - involves extra memory usage
- UDFs can be precompiled before invoking in FLUENT
  - Needs a compiler
  - It translates the C-source to machine language (object modules)
  - Needs to follow a standard multi-step procedure (will be discussed later)
  - Creates ‘shared libraries’ linked with the rest of the solver

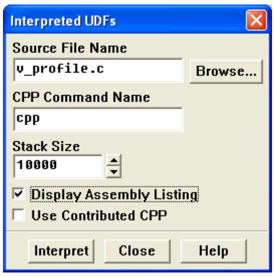
ALL INTERPRETED UDF-S CAN ALSO BE COMPILED
THOUGH THE CONVERSE IS NOT TRUE

Interpreted UDFs

- Interpreter limitations:
  - mixed mode arithmetic,
  - structure references etc.
  - cannot be linked to compiled system or user libraries
  - less powerful than compiled UDFs due to limitations in the C language supported by the interpreter
- In particular, interpreted UDFs cannot contain:
  - non ANSI-C prototypes for syntax
  - declarations of local structures, unions, pointers to functions, and arrays of functions
  - direct structure references
- Interpreted UDFs can indirectly access data stored in a FLUENT structure only via a set of macro-s

Interpreting the UDF (2)

- Define →User Defined Functions →Interpreted…



Interpreted UDFs

Source File Name: v_profile.c

CPP Command Name: cpp

Stack Size: 10000

☑ Display Assembly Listing

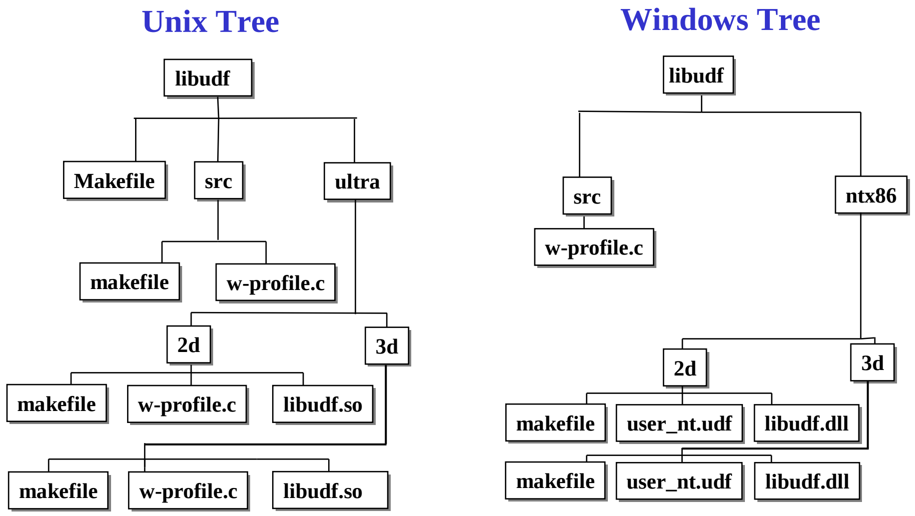
☐ Use Contributed CPP

- Click Interpret
- The assembly language code will scroll past window

Listing appearing on Fluent windows:

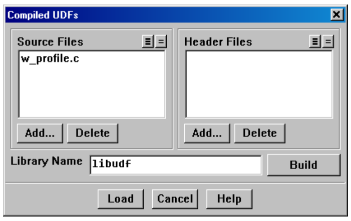
```
w_profile:
   .local.pointer thread (r0)
   .local.int position (r1)
0  .local.end
0   save
   .local.int f (r6)
8   push.int 0
10  save
   .local.int.
   .
   .  } Skipping display here
.L1:
132 restore
133 restore
134 ret.v
```

Compiled UDF Directory Structure



UDF Compilation in F6.2

- To compile UDFs from within Fluent, use:
  - Define→User_Defined→Functions→Compile…
- Placing source routines in your working directory would be sufficient and necessary
- This GUI creates the directory structure below your working directory where you have your case and data files
- This GUI identifies the architecture as well as the version of fluent running and compiles only for the appropriate UDF version (2d/2ddp/3d/3ddp/or any parallel version)



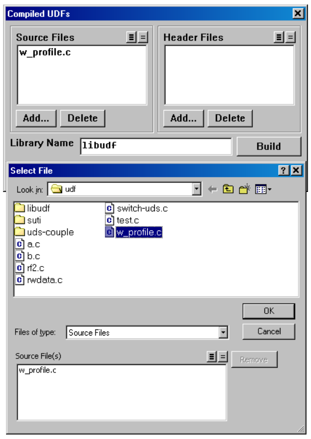
Source Files: w_profile.c

Library Name: libudf

UDF Compilation in F6.2

- Define→User_Defined Functions→Compile…
- Click on the “Add” button to browse and add source and header files
- Click on “Build” button to compile and then “Load” to load the library to a case file
- The compilation log appears on the Fluent console window and in a file named log
- To unload a compiled UDF, use

  Define→User_Defined Functions→Manage, select the library, then click Unload button



Source Files: w_profile.c

Library Name: libudf

Look in: udf

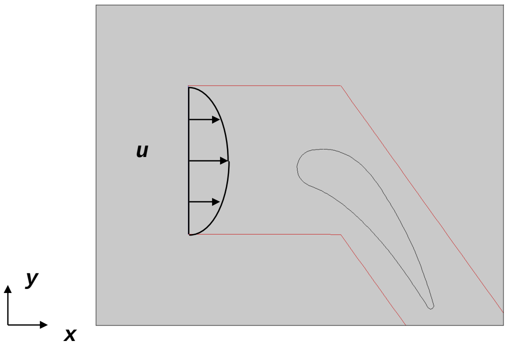
Files of type: Source Files

Source File(s): w_profile.c

Using UDFs - Example

- A non-uniform inlet velocity is to be imposed on the 2D turbine vane shown below. The x-velocity variation is to be specified as

$$u(y) = 20\ [\ 1 - (y/0.067)^2]$$



A Source Code Example

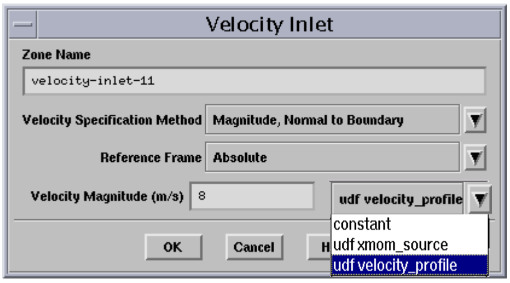
```
#include "udf.h"

DEFINE_PROFILE(velocity_profile, thread, position)
{
   real x[3]; /* this will hold the position vector*/
   real y;
   face_t f;

   begin_f_loop(f, thread)
    {
     F_CENTROID(x,f,thread);
     y = x[1];
     F_PROFILE(f, thread, position) = 20.*(1.- y*y /
                                          (.067*.067));
    }
   end_f_loop(f, thread)
 }
```

Activating the UDF

- Access the **`boundary condition`** panel
- Switch from **`constant`** to the **`UDF function`** in the **`Velocity Magnitude`** dropdown list



Velocity Inlet

Zone Name: velocity-inlet-11

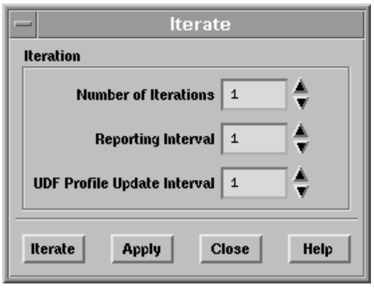
Velocity Specification Method: Magnitude, Normal to Boundary

Reference Frame: Absolute

Velocity Magnitude (m/s): 8 udf velocity_profile

Run the Calculation

- Run the calculation as usual
- You can change the **`UDF Profile Update Interval`** in the **`Iterate panel`** (here it is set to 1)



Iterate

Iteration

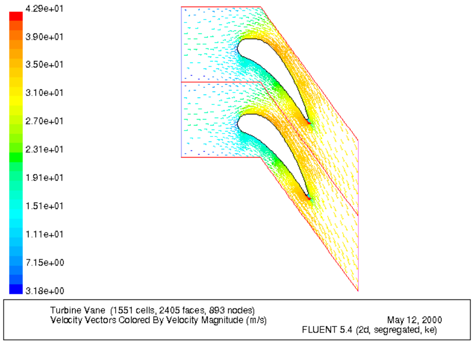
Number of Iterations: 1

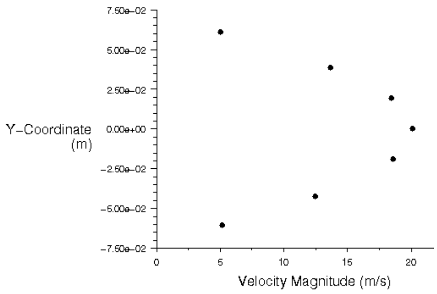
Reporting Interval: 1

UDF Profile Update Interval: 1

Solution of Example problem

- The figure at right shows velocity field throughout turbine blade passage
- The bottom figure shows the velocity plot at the inlet
- Notice the imposed parabolic profile


4.29e+01
3.90e+01
3.50e+01
3.10e+01
2.70e+01
2.31e+01
1.91e+01
1.51e+01
1.11e+01
7.15e+00
3.18e+00
Turbine Vane (1551 cells, 2405 faces, 893 nodes)
Velocity Vectors Colored By Velocity Magnitude (m/s)
May 12, 2000
FLUENT 5.4 (2d, segregated, ke)


7.50e−02
5.00e−02
2.50e−02
Y−Coordinate (m)
0.00e+00
−2.50e−02
−5.00e−02
−7.50e−02
Velocity Magnitude (m/s)