User Defined Function: lidar_3D

M T Stickland, S Fabre, T Scanlon

January 2010

University of Strathclyde NORSEWInD Report UoSNW006

Department of Mechanical Engineering
University of Strathclyde
75 Montrose St, Glasgow
G1 1XJ

Fax +44 141 552 5105 Email: matt.stickland@strath.ac.uk
Tel: +44 141 548 2842

User Defined Function: lidar_3D

UoSNW006 revision 01

ABSTRACT

This report describes the User Defined Function Lidar_3D used to interrogate FLUENT data files to provide the relevant data for the MathCAD LiDAR simulation program. The UDF was written in the C programming language and compiled using Microsoft visual studio 2008. This report contains a listing of the program (version 1.03).

This report contains a description of the methodology required to compile the UDF so that it may be called by an "execute on demand" call from FLUENT.

The report also includes a description of the input and output data file formats.

User Defined Function: lidar_3D

UoSNW006 revision 01

User Defined Function: lidar_3D

UoSNW006 revision 01

1. INTRODUCTION

This report describes how the FLUENT data files were interrogated by the User Defined Function (UDF) "libudf2". The UDF was written in the C programming language and compiled using Microsoft visual studio 2008. A listing of the program version 1.03, which was the version in use at the time that this report was written, can be found in section # of this report.

This report contains a description of the method required to compile the UDF so that it may be called by an "execute on demand" call from FLUENT.

The report also explains how the UDF works though the comments made within the program listing in section 2

2. PROGRAM LISTING

```
/*This UDF provided relates to a support query. This UDF should be
treated as
a guideline outlining the application. This is not a consultancy
project
and has not therefore been checked under our consultancy procedures. It
is
your responsibility to check the validity of any simulation utilizing
this
UDF. Additionally, UDF support for current license
holders will be limited to guidance related to communication between a
UDF
and the FLUENT solver. Other aspects of the UDF development process
that
include conceptual function design, implementation (writing C code),
compilation and debugging of C source code, execution of the UDF, and
function design verification will remain the responsibility of the UDF
author.
Feedback, comments or queries are most welcome*/

#include "point_in_cell.h"
#include "math.h"

/*below is the macro that determines the location of the monitor points
and outputs required information. This is the only macro that requires
modification to
account for different points distribution or to change the output
method*/
/* note that phi is the cone angle */
/* note psi is the asimuth angle */
```

User Defined Function: lidar_3D

UoSNW006 revision 01

```
DEFINE_ON_DEMAND(lidar_3d)
{

int yorn, zone;
float xloc, yloc, zloc, height, dh, h, nh, ver;
float u, v, w, x, y, z, hmeas[50];
float phi_deg=30.4, phi_rad;
float npoints=50;
float pi=3.14159;
float psi_deg, psi_inc, psi_rad;
float ploc[50][3], Vlos[50][50], asangle[50];
int i,j,nj;
char dummy;

FILE *exp_file_ptr;
FILE *inp_file_ptr;

Domain *d;
Thread *ct;
cell_t c;
real point[3];

ver=1.03;

Message("lidar simulation program version %.2f started \n", ver);

/* open a file to write data */

exp_file_ptr=fopen("data.dat","w");


/* open a file to read data */


inp_file_ptr=fopen("instinfo.dat","r");



/* printf ("enter the measurement height of the lidar \n"); */
/* scanf ("%f", &height); */
/* printf ("enter the offset from the origin of the lidar x y and z
\n"); */
/* scanf ("%f %f %f", &xloc, &yloc, &zloc); */


      if (inp_file_ptr !=NULL)

            {
            fscanf(inp_file_ptr, "%s %s %s %s %s %s %s \n", &dummy,
&dummy, &dummy, &dummy, &dummy, &dummy, &dummy);
            fscanf(inp_file_ptr, "%d %d %f %f %f %f %f \n", &zone,
&yorn, &height, &nh, &xloc, &yloc, &zloc);
            }

      else
```

```
            {
            printf("no file name \n");
            }

fclose(inp_file_ptr);

/* inputs required for the fluent UDF get domain defines the entire
domain and lookup thread get the thread
number for the specified zone in the given domain */

d = Get_Domain(1);
ct = Lookup_Thread(d, zone);

Message("\n %d heights to be calculated \n", nh);

dh=height/nh;


phi_rad=phi_deg*pi/180;

psi_inc= 2*pi/50;

if (yorn == 1)
{
Message("verbose output version %.2f \n", ver);

for (j=1; j<=nh; j++)

{
      Message("measurement height %d being calculated \n", j);
      h=dh*j;
      fprintf(exp_file_ptr, "measurement height %.2f m \n",h);

      /* to find the velocity at the measurement height directly above
the lidar */

      /* define the point at the measurement height directly above */

      point[0]= xloc;
      point[1]= yloc;
      point[2]= zloc + h;

      /*pointer to the cell containing the xyz location*/

      c = cell_containing_point(point, ct);

      /* find the three components at that point */

      u=C_U(c,ct);
      v=C_V(c,ct);
      w=C_W(c,ct);

      fprintf(exp_file_ptr, " data at %6.1f m directly above lidar \n",
h);

      fprintf(exp_file_ptr, "   x     y      z       u     v       w
\n");
```

User Defined Function: lidar_3D

UoSNW006 revision 01

```
      fprintf(exp_file_ptr, "%6.1f %6.3f %6.3f %6.3f %6.3f %6.3f
\n",point[0],point[1],point[2], u, v, w);

      fprintf(exp_file_ptr, "Lidar simulation data \n");

      fprintf(exp_file_ptr, "   psi    x     y      z       u     v
w   Vlos \n");


      for (i=0;i<=49; i++)

            {

            /* calculate the x, y and z locations of the scan points
above an origin of 0, 0, 0 */


            psi_rad=i*psi_inc;
            psi_deg=psi_rad*180/pi;
            ploc[i][0]= h*sin(phi_rad)*sin(psi_rad)/cos(phi_rad) ;
            ploc[i][1]= h*sin(phi_rad)*cos(psi_rad)/cos(phi_rad) ;
            ploc[i][2]= h;

            }


      for (i=0;i<=49; i++)

            {

            /* apply an offset to the measurement locations due to the
lidar not being sited at the origin */
            /* and assign the location of the measurement point to the
array point which is the
            x y and z location that will be interrogated to find the u,
v and w at that point*/

            psi_rad=i*psi_inc;
            asangle[i]=i*psi_inc;
            psi_deg=psi_rad*180/pi;

            point[0]= ploc[i][0] + xloc;
            point[1]= ploc[i][1] + yloc;
            point[2]= ploc[i][2] + zloc;

            /* put the x y and z values into variables that are easily
recognised for calculating Vlos */

            x=ploc[i][0];
            y=ploc[i][1];
            z=ploc[i][2];

            /*pointer to the cell containing the xyz location*/

            c = cell_containing_point(point, ct);
```

```
            /*get the u, v and w velocity components in that cell */

            u=C_U(c,ct);
            v=C_V(c,ct);
            w=C_W(c,ct);

            /*calculate the V line of sight based on the x, y and z with
the Lidar at the origin */

            Vlos[j][i]=(u*x+v*y+w*z)/sqrt(x*x+y*y+z*z);

            /* put the offset so that the location of the measurememement
points are in the simulation coordinate system */

            x = ploc[i][0] + xloc;
            y = ploc[i][1] + yloc;
            z = ploc[i][2] + zloc;

            fprintf(exp_file_ptr, "%6.1f %6.3f %6.3f %6.3f %6.3f %6.3f
%6.3f %6.3f \n",psi_rad, x, y, z, u, v, w, Vlos[j][i]);


            }
}

}
else
{
Message("abreviated output version %.2f \n", ver);

fprintf(exp_file_ptr, "%.2f \n",nh);

for (j=1; j<=nh; j++)

{
      Message("measurement height %d being calculated \n", j);
      h=dh*j;
      fprintf(exp_file_ptr, "%.2f \n",h);

      point[0]= xloc;
      point[1]= yloc;
      point[2]= zloc + h;

      c = cell_containing_point(point, ct);

      u=C_U(c,ct);
      v=C_V(c,ct);
      w=C_W(c,ct);

      fprintf(exp_file_ptr, "%6.1f %6.3f %6.3f %6.3f %6.3f %6.3f
\n",point[0],point[1],point[2], u, v, w);

      for (i=0;i<=49; i++)

            {

            psi_rad=i*psi_inc;
```

```
            psi_deg=psi_rad*180/pi;
            ploc[i][0]= h*sin(phi_rad)*sin(psi_rad)/cos(phi_rad) ;
            ploc[i][1]= h*sin(phi_rad)*cos(psi_rad)/cos(phi_rad) ;
            ploc[i][2]= h;

            }


      for (i=0;i<=49; i++)

            {

            psi_rad=i*psi_inc;
            asangle[i]=i*psi_inc;
            psi_deg=psi_rad*180/pi;

            point[0]= ploc[i][0] + xloc;
            point[1]= ploc[i][1] + yloc;
            point[2]= ploc[i][2] + zloc;

            x=ploc[i][0];
            y=ploc[i][1];
            z=ploc[i][2];

            c = cell_containing_point(point, ct);

            u=C_U(c,ct);
            v=C_V(c,ct);
            w=C_W(c,ct);

            Vlos[j][i]=(u*x+v*y+w*z)/sqrt(x*x+y*y+z*z);

            x = ploc[i][0] + xloc;
            y = ploc[i][1] + yloc;
            z = ploc[i][2] + zloc;

            fprintf(exp_file_ptr, "%6.1f %6.3f %6.3f %6.3f %6.3f %6.3f
%6.3f %6.3f \n",psi_rad, x, y, z, u, v, w, Vlos[j][i]);
            }
}
}

fclose(exp_file_ptr);

      Message("lidar simulation ended \n");

}


cxboolean outward_face(face_t f, Thread *ft, cell_t c, Thread *ct)
{
  Thread *ct1;
  cell_t c1;
  ct1 = THREAD_T1(ft);
  if(NULLP(ct1))return TRUE; /* face on boundary */
  c1 = F_C1(f,ft);
```

```
  if ((c==c1)&&(ct==ct1))return FALSE; /* face's c1 is c so f inward to
c */
  return TRUE;
}
cxboolean point_in_cell(real point[ND_ND], cell_t c, Thread *ct)
{
  int i;
  real dist;
  real p_rel[ND_ND], f_cen[ND_ND], A[ND_ND];
  face_t f;
  Thread *ft;
  cxboolean inside = TRUE;
  c_face_loop(c, ct, i)
      {
        if(inside)
            {
              f=C_FACE(c, ct, i);
              ft=C_FACE_THREAD(c, ct, i);
              F_CENTROID(f_cen,f,ft);
              F_AREA(A,f,ft);
              NV_VV(p_rel,=,point,-,f_cen); /* point relative to f_cen
*/
              dist=NV_DOT(p_rel,A);
              if (outward_face(f,ft,c,ct)) /* Count as inside if dist ==
0.0 */
                  {if (dist > 0.0) inside = FALSE;}
              else
                  {if (dist < 0.0) inside = FALSE;}
            }
      }
  return inside;
}
cell_t cell_containing_point(real point[ND_ND], Thread *ct)
{
  cell_t c;
  cell_t c_in = NULL_CELL;
  begin_c_loop(c,ct)
      {
        if(c_in == NULL_CELL) /* if cell not found yet */
            {
              if (point_in_cell(point, c, ct)) c_in = c;
            }
      }
  end_c_loop(c,ct)
  return c_in;
}
```

3. INSTINFO FILE FORMAT

The file instinfo.dat, table 1, contains the instructions for the UDF about where to interrogate the FLUENT data set.

User Defined Function: lidar_3D

UoSNW006 revision 01

Zone	Vebose	Height	Nh	Xloc	Yloc	zloc
2	0	40	20	129	129	25.5

table 1; instinfo.dat file format

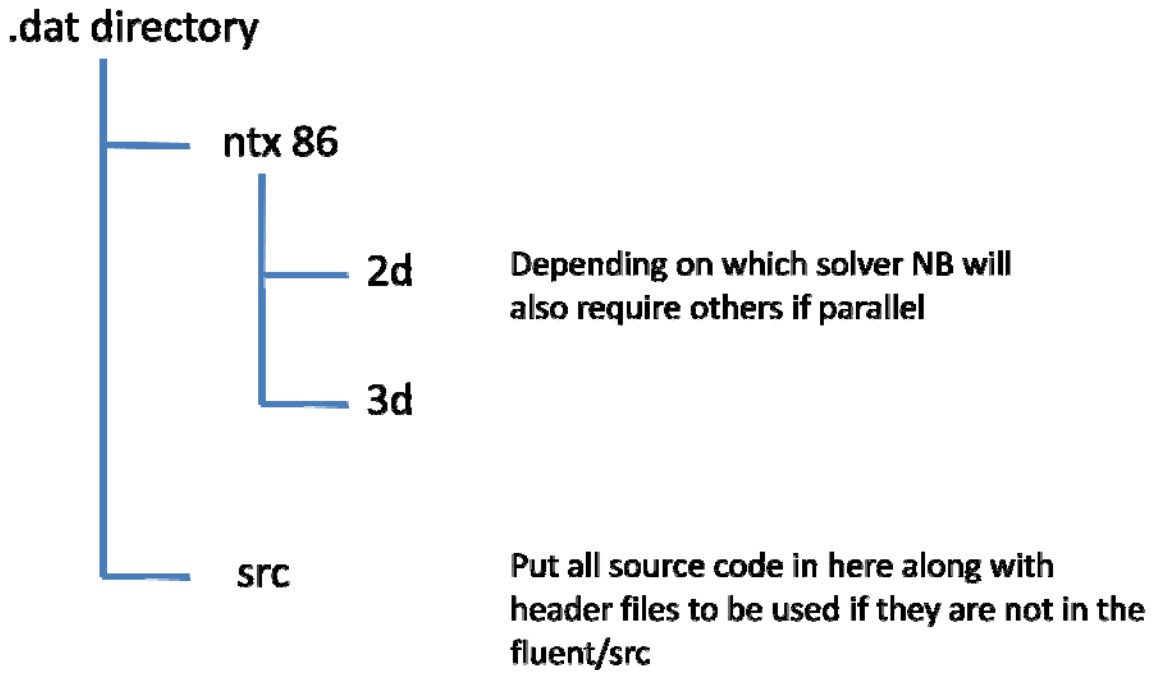
Zone	The zone is defined by Fluent and is the region in the domain that contains the fluid. It can be found by selecting Define/Boundary conditions/Zone from the main FLUENT menu. if in the Zone table the fluid surrounding the rig is selected then the number shown in the ID box at the bottom of the wind is the number to be entered. if this number is not correct then the following error message should appear Error: FLUENT received fatal signal (ACCESS_VIOLATION) 1. Note exact events leading to error. 2. Save case/data under new name. 3. Exit program and restart to continue. 4. Report error to your distributor. Error Object: ()
Verbose	Either 0 or 1 If 0 then all that is echoed to the FLUENT window is the sequential number of the height being interrogated If 1 then the velocity data from the interrogation is echoed to the FLUENT window.
Height	The height above the lidor that is to be interrogated
nh	The number of heights to be interrogated. The location of the measurement heights are then height/nh plus increments of height/nh op to height
Xloc	The location of the LiDAR in the x direction of the fluent coordinate system in the same scale as the fluent model
yloc	The location of the LiDAR in the y direction of the fluent coordinate system in the same scale as the fluent model
zloc	The location of the LiDAR in the y direction of the fluent coordinate system in the same scale as the fluent model

4. COMPILE PROCEDURE

To compile a UDF to interrogate the FLUENT data file a suitable C compiler and linker needs to be installed. The compiler and linker used in this case was Microsoft Visual Studio 2008 which is the one recommended by FLUNT. comprehensive details of creating UDF can be found in the relevant ANSYS FLUENT documentation.

Directory Structure

The library structure was created off of the directory containing the dat files



copy *user_nt.udf* and *makefile_nt.udf* from the main */fluent/src* directory into the new */2d* or */3d* directory.

copy the source file for the UDF into the */src* directory

Compiling The Udf

edit *user_nt.udf*

change SOURCES to include the new source file, VERSION to 2d or 3d depending on model, PARALLEL NODE to none if serial solver.

rename *makefile_nt.udf* to *makefile*

User Defined Function: lidar_3D

UoSNW006 revision 01

If the following error message appears during compilation

makefile [51]: fatal error U105 need to define the environment variable FLUENT_INC

the *makefile* needs to be edited to include

FLUENT_INC=c:\fluent.inc

It may be necessary from the console to run *vcvars32.bat* if editing the *makefile* does not work first time.

The path statement must include all directories that contain files that are going to be compiled or accessed during the compile process. To modify the path click on the start button, right click on computer, click on properties, click advanced settings, under the advanced tab click on environment variables, in the system variables edit the path statement.

Open the Visual Studio command prompt window

In the command prompt window change directory to the */3d* (assuming that the simulation is 3d) directory that contains the *makefile*. run *nmake* in this directory.

Running The UDF In FLUNET

When compilation has been successful the UDF needs to be loaded and run in FLUENT. From the main fluent window select *define/userdefined/functions/manage* menu load the compiled UDF

if successful a message similar to

```
Opening library "libudf2"...
Library "libudf2\ntx86\3d\libudf.dll" opened
        lidar_3d
Done.
```

will appear in the console where *lidar_3d* is the UDF defined in the compiled code.

after the simulation has completed or the data has been opened from an existing run the UDF is executed by selecting *define/userdefined/execute* one demand and then selecting the UDF from the drop down list.

NB if the lidar_3D UDF is run the zone in the *instinfo.dat* file must correspond to the fluid region.

UDF Output File

The lidar_3D UDf creates an output file *data.dat* The format of the file is shown in table 2.

<table>
<tr><td>Number of heights</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>height</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td rowspan="6">Repeated for n heights</td></tr>
<tr><td>point x</td><td>point y</td><td>point z</td><td>u</td><td>v</td><td>w</td><td></td><td></td><td></td></tr>
<tr><td>psi (rad)</td><td>x</td><td>y</td><td>z</td><td>u</td><td>v</td><td>w</td><td>Vlos</td><td rowspan="3">Repeated for 360°</td></tr>
<tr><td>psi (rad)</td><td>x</td><td>y</td><td>z</td><td>u</td><td>v</td><td>w</td><td>Vlos</td></tr>
<tr><td>psi (rad)</td><td>x</td><td>y</td><td>z</td><td>u</td><td>v</td><td>w</td><td>Vlos</td></tr>
</table>

table 2; lidar_3D UDF output file format

The output file from the UDF is read by the MathCAD LiDAR simulation program details of which can be found in the program description document [1].

5. REFERENCES

1. Stickland, M., Scanlon, T., Fabre, S., "MathCAD program; ZephIR_lidar_sim" report for EU NORSEWInD, UoSNW003, January 2010

User Defined Function: lidar_3D

UoSNW006 revision 01