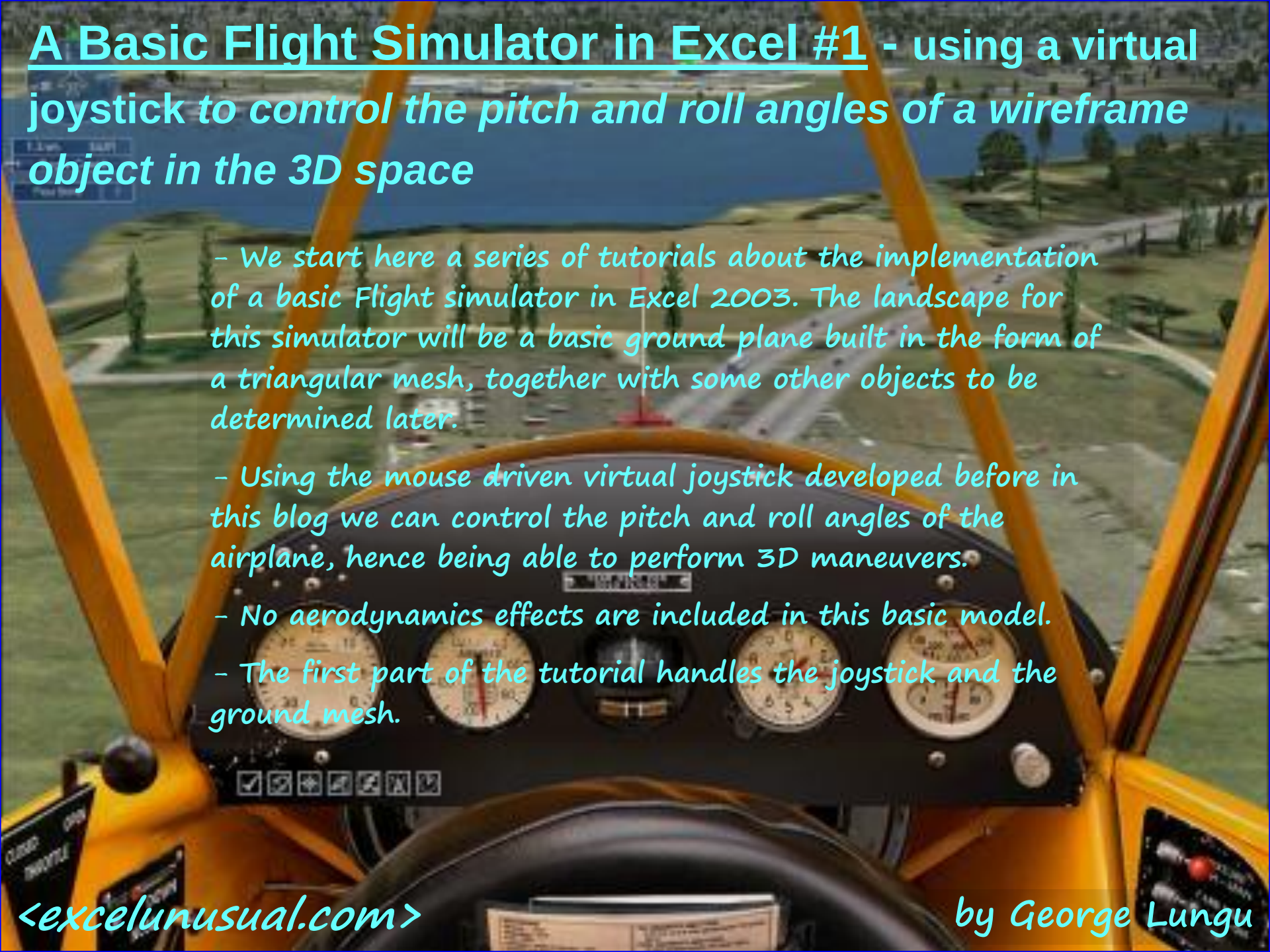
A Basic Flight Simulator in Excel #1 - using a virtual joystick *to control the pitch and roll angles of a wireframe object in the 3D space*



- We start here a series of tutorials about the implementation of a basic Flight simulator in Excel 2003. The landscape for this simulator will be a basic ground plane built in the form of a triangular mesh, together with some other objects to be determined later.

- Using the mouse driven virtual joystick developed before in this blog we can control the pitch and roll angles of the airplane, hence being able to perform 3D maneuvers.

- No aerodynamics effects are included in this basic model.

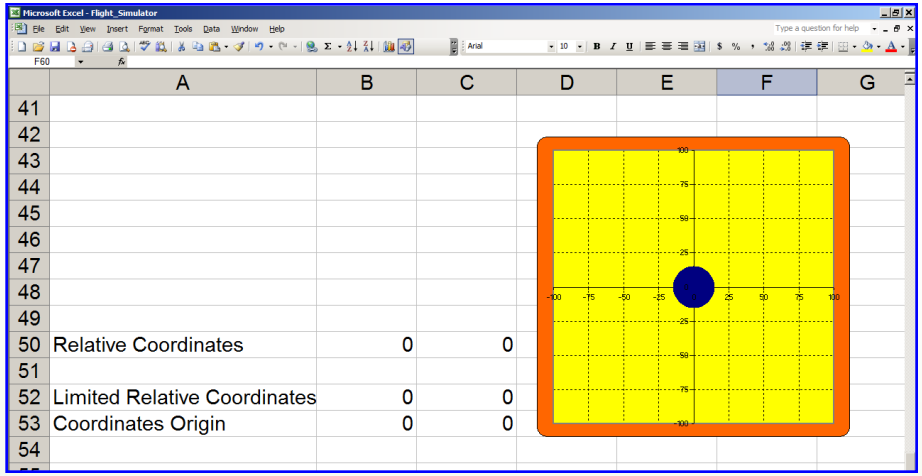
- The first part of the tutorial handles the joystick and the ground mesh.

<excelunusual.com>

by George Lungu

The virtual joystick:

- We will use elements of a previously created virtual joystick model.
- Open a new worksheet and name it “Flight_Simulator”.
- Rename the first worksheet “Tutorial_1”.
- The joystick will be implemented as a 2D scatter chart.
- Clicking the chart will start the “JoyStick” macro which continuously updates (in range B50:C50) the relative mouse coordinates relative to the initial click point on the screen (when the macro was triggered).
- Cells A50, A52 and A53 contain labels.
- Range B50:C50 contain the relative x-y coordinates of the mouse relative to the initial click point.
- Range B52:C52 contains the previously mentioned x-y coordinates (from range B50:C50) but limited to +/- 100.
- Range B53:C53 is the origin of the joystick on the chart and this range is filled with two zeros.
- The 2D scatter chart has both axes scaled to [-100, 100] and the data plotted is from the range B52:C53.
- The “JoyStick()” macro seen to the right, was previously explained and a few more details are given in the next page. For additional insight, the reader is advised to read the Joystick tutorial from January 2011.




```
Public Declare Function GetCursorPos _
       Lib "user32" (Some_String As POINTAPI) As Long

Type POINTAPI
   X  As Long
   Y  As Long
End Type

Dim RunPause As Boolean
---------------------------------------------------------------------
Sub JoyStick()
Dim Pt0 As POINTAPI
Dim Pt1 As POINTAPI
RunPause = Not RunPause
GetCursorPos Pt0
  Do While RunPause = True
  DoEvents
   GetCursorPos Pt1
   [B50] = Pt1.X - Pt0.X
   [C50] = -Pt1.Y + Pt0.Y
   Loop
End Sub
```

A few explanations about the "JoyStick()" macro:

The following are declarations:

```
Public Declare Function GetCursorPos _
      Lib "user32" (Some_String As POINTAPI) As Long
```

Declaration of a special API (Application Programming Interface) function which retrieves the cursor position

```
Type POINTAPI
   X  As Long
   Y  As Long
End Type
```

Declaration of a special structure (Point API) used as the output type of the previous API function. It is essentially the pair of coordinates (as long integers) of the cursor on the screen started to be measured from the upper left corner of the screen.

```
Dim RunPause As Boolean
```

Boolean variable declaration which has the role of a "switch", keeping track if the macro is running or is stopped. This will allow the macro to be started or stopped using the same button

```
Sub JoyStick()
```
→ Declaration of the macro

```
Dim Pt0 As POINTAPI
Dim Pt1 As POINTAPI
```
→ Declaration two Point API type structures, one as initial click coordinates and the second as the current (dynamic) cursor coordinates

```
RunPause = Not RunPause
```
→ Boolean "flip", if the macro is stopped this will start it and vice versa

```
GetCursorPos Pt0
```
→ Assigns variable "Pt0" the initial click coordinates

```
  Do While RunPause = True
```
→ Conditional "Do" loop declaration (start)

```
  DoEvents
```
→ Always add this statement if you ever need to stop the loop manually or update a chart while the loop is running

```
   GetCursorPos Pt1
```
→ Every loop cycle assigns "Pt1" the cursor coordinates

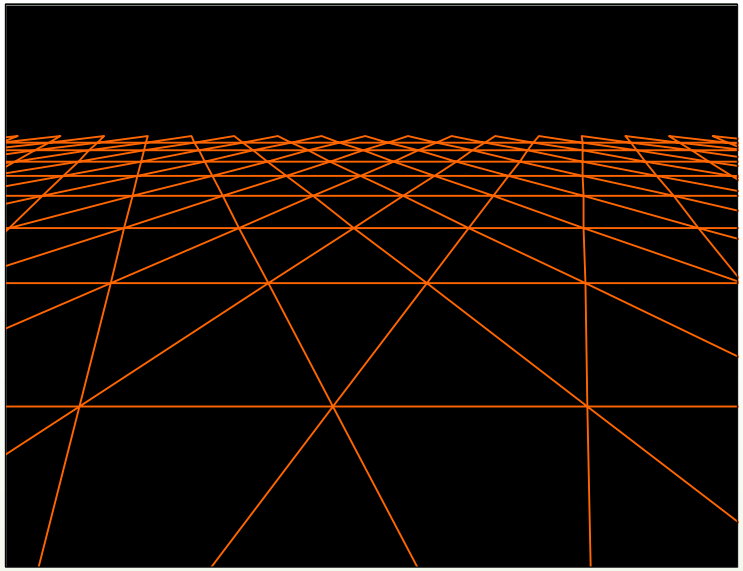
```
   [B50] = Pt1.X - Pt0.X
   [C50] = -Pt1.Y + Pt0.Y
```
→ Every loop cycle calculate the relative coordinates and display them in the range "B50:C50" (figure out why I wrote those formulas the way I wrote them)

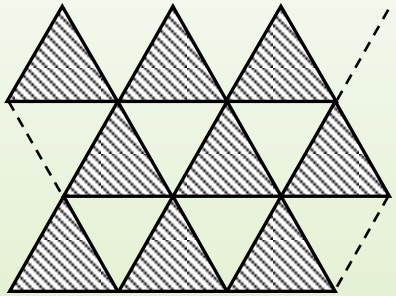
```
   Loop
```
→ End of "Do" loop

```
End Sub
```
→ End of macro declaration

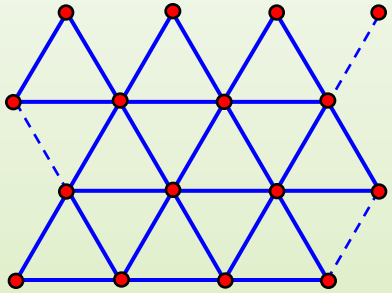
The ground plane:

- We will use here the type of ground plane used before in the *roller coaster* model. To the right you can see a 3D snapshot of that ground plane.
- We will generate the ground as a triangle mesh having 40x40 triangles.
- In order to get that type of mesh, we need to create 40 x 40 triangles and place them staggered just like in the picture to below (which shows a small 3x3 grid demo created using nine triangles).





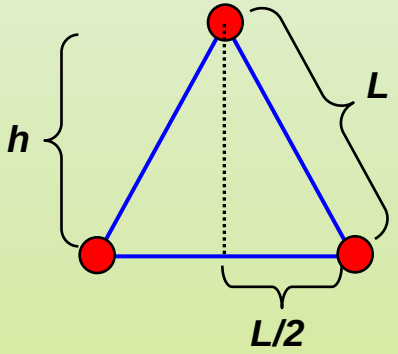
Generating the vertices:



- In order to illustrate the principle we will use a small 3x3 grid.
- For this 3x3 mesh of triangles we have a 4x4 array of vertices (in red). In general for an n x n array of triangles we need an (n+1) x (n+1) array of vertices

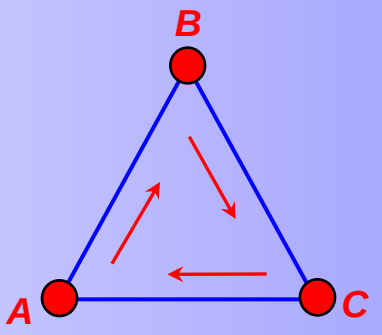
- Using the notation L for the side of the equilateral triangle we can write the following relationship:

$$h = \frac{\sqrt{3}}{2} \cdot L$$



The x, y and z vertex matrices:

- We will chart the ground plane as a single curve on a 2D scatter chart.
- Since there are 40 x 40 = 1600 triangles and each triangle is charted as a four point group **(A=>B=>C=>A)** on the chart, a (4 x 1600) = 6400 point curve will be charted to represent the ground plane.
- We need to be able to make invisible any triangle which has at least one vertex behind the observer, otherwise annoying artifacts will be noticed in the image (I observed this effect by chance in previous models).



- To be able to withdraw triangles from the visible area of the chart we need to create a table with 6400 x-y data points and 1599 additional blank rows, one blank row between each 4 data rows corresponding to each triangle. This way if a certain triangle falls behind the observer we can replace its vertices with very large numbers (9999 for instance) and the triangle will disappear from view.
- Due to the blank row in the data table between each triangle and its neighbors, each triangle will be disjoint (unconnected) from its neighbors. This way, by moving a triangle out of view will not leave “strings” attached to it’s visible neighbors. Again I observed this “attached strings” effect in previous models and the only way I was able to get rid of it was by inserting blank rows in the data table.
- We will define a 123 x 41 x-y-z Vertex array having the x-y-z data for 41 x 41 vertices:
- We will also define a “Landscape” array whose elements will be added to the z-coordinates in the “Ground” array.
- We could have just used a single 3-column table for the x-y-z vertex data but having a 2-dimensional individual table makes it more easy to insert non-flat data (hills and mountains for instance) and chart this data easier. The picture to the right shows how the data is packed in the first two rows of this table.

first vertex

x0	x1	x2	x3	.....	x39	x40
y0	y1	y2	y3	.....	y39	y40
z0	z1	z2	z3	.....	z39	z40
x41	x42	x43	x44	.....	x80	x81
y41	y42	y43	y44	.....	y80	y81
z41	z42	z43	z44	.....	z80	z81

Creating the vertex matrices for the ground plane:

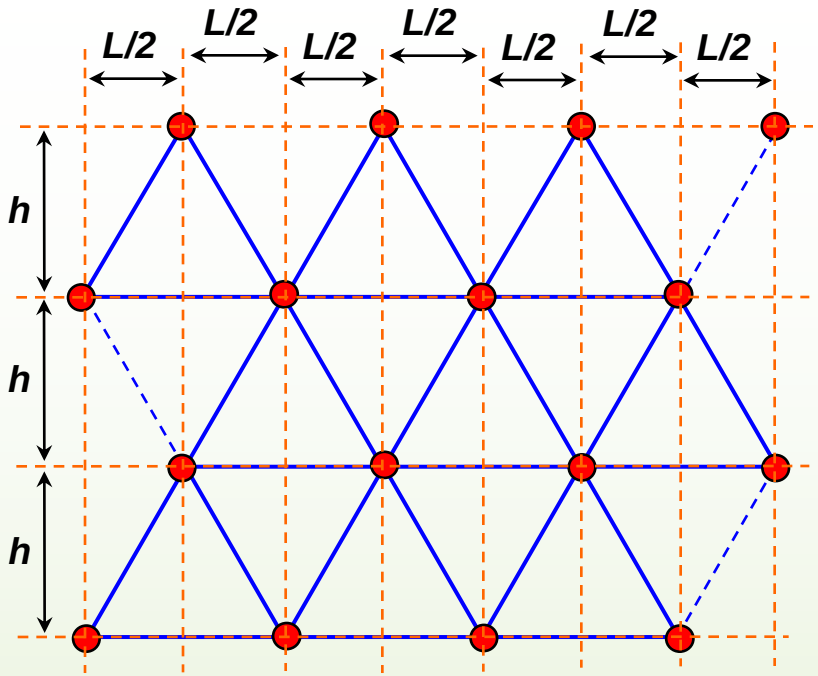
- Create a general ground plane parameter area:
- The ranges U73:U74, U76:U78 U80, U130 and U180 are labels.
- V73:V74 contain the geometrical parameters of the triangles:

V73: “=100”

V74: “=sqrt(3)*V73/2

	U	V	W
70			
71			
72			
73	L	100	
74	h	86.60254	
75			
76	Offset_x	0	
77	Offset_y	0	
78	Offset_z	0	
79			

-Fill V76:V78 with zeroes for now. We can plug formulas in the offset area later if we ever need to do so.



Create a “Landscape” array: For now fill all the range V211:BJ251 with zeroes. This could later be filled with various landscape effects (valleys and hills).

Create the “x-y-z Vertex” array:

V81: “=V$76”, W81: “=V81+$V$73”

V82: “=V$77 then copy this one cell to the right (cell W82)

V83: “=$V$78+V211” then copy this one cell to the right (cell W83)

Copy range V81:W83 to V84:W86 and change the following V84: “=V$76+$V$73/2”

Change V85: “=V82+$V$74” and copy this cell to the right (to cell W85)

Copy range V84:W86 to V87:W89 and change V87: “=V$76”

Copy range V84:W89 to range V90:W203

Copy range W81:W203 to range BJ81:BJ203 and you finished the vertex array.

to be continued...