Asian Journal of Information Technology 4 (9): 878-884, 2005

Simple Flight Simulator Model

Loay E.George and Suhad F.Sheehan
Informatics Instituites for Postgraduate Studies,University of Technology, Baghdad, Iraq

Abstract: Pilots are trained using computerized flight simulators. A flight simulator is a training system where pilots can acquire flying skills without need to practice on a real airplane. Simulators are used by professional pilots to practice flying strategies under emergency or hazardous conditions, or to train on new aircraft types. In this study a framework for flight simulation is presented and the layout of an implemented program is described. The calculations were based on simple theoretical approach. The implementation was based on utilizing some of utilities supported by ActiveX, DirectX and OpenGL written in Visual C++. The main design consideration is to build a simple flight simulation program can operate without need to high computer environment specifications.

Key words: OpenGL, activeX, directX, joystick, throttle, rudder, elevator, aileron, pitch, roll, yaw, longitude, latitude, heading, speed, altimeter, gauge

INTRODUCTION

This study is an attempt to establish a Flight Simulator Cockpit with almost its main gauges, control surfaces that are controlled by the trainee via a joystick and it responds like the real flight to the trainee instructions, there is an external view to make sense of the outside world, which animates according to the simulator actions and reactions. Several effective parameters are considered such as (wind, wind speed, crosswind speed, weight, minimum weight, delay time of the hardware, air speed, etc.) for giving reality to the system. A theoretical framework is developed as a mathematical model for simulation. All of its involved parameters are adopted in the designed model and they have been successfully implemented. Flight Simulator System software is based on the Object-Oriented concepts that simplifies the necessary synchronization and the coordination of the simulator system. The software was developed using a mixture of

1. DirectX for joystick as an input tool
2. ActiveX for drawing the 11 gauges
3. OpenGL for drawing the external view and all is written using Visual C++ language version 6.

Since the late 1950's the National Aeronautics and Space Administration (NASA) has found in-flight simulation to be an invaluable tool. In flight simulation has been used to anticipate problems and to avoid them and to solve problems once they appear. Before flying an experimental aircraft it is always desirable to consider the flying qualities of the vehicle. New aircraft of unusual configuration or flight envelope, however, require special handle[1]. Northrop Aircraft Group had been involved in aircraft simulation since the early 1960's. In the mid-1970's, the original analog simulation computers were replaced by digital computers[2].

THE SYSTEM LAYOUT

The Overall simulation model of the Flight Simulator is shown in a block diagram (Fig. 1). The Data Flow Diagram (DFD) is a perfect media that provide the work with a powerful and useful picture about the whole skeleton of the system from the beginning stages which help in deciding which way could be taken in order to synchronize and coordinate the overall systems' objects. The DFD of the implemented system is demonstrated in fig. 2. In the object oriented analysis the system could be modeled as a collection of objects and classes, the organization of these objects and classes is arranged by the class hierarchy (Fig. 3).

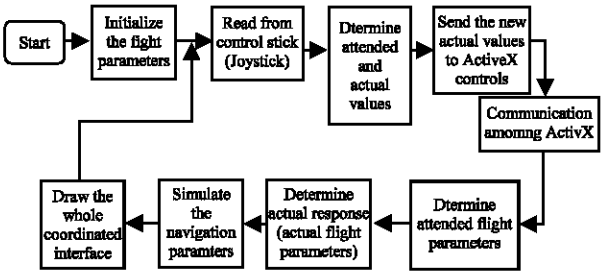

Fig. 1: The overall flight simulator block diagram

Corresponding Author: Dr. Loay E. George, Sheehan Informatics Instituites for Postgraduate Studies, University of Technology, Iraq Baghdad

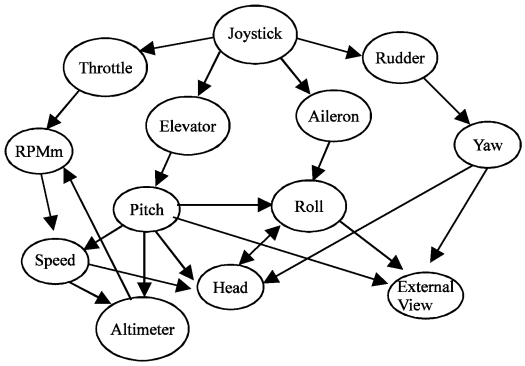

Fig. 2: Data flow diagram of the flight simulator system

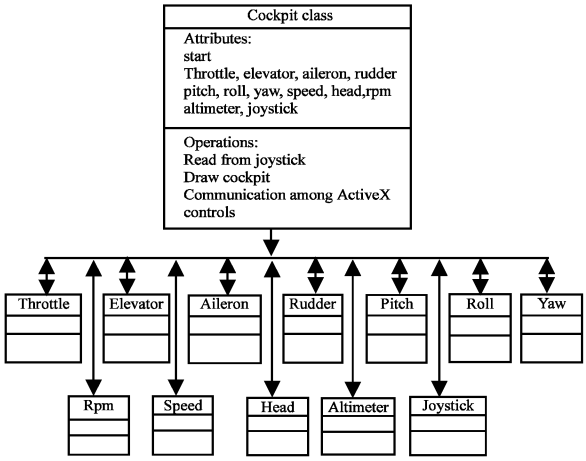

Fig. 3: The class hierarchy of the flight simulator system

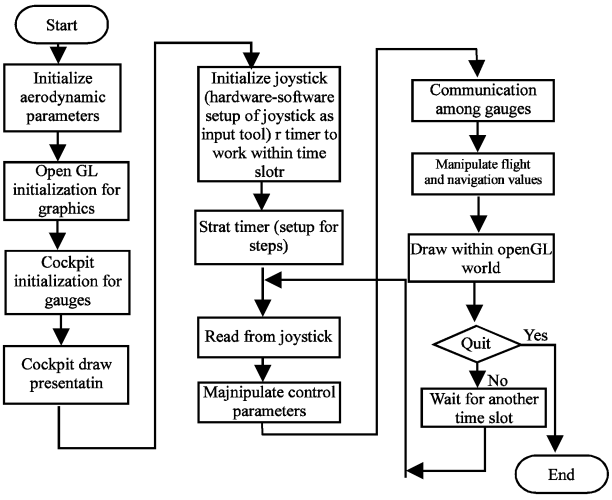

Fig. 4: Flowchart of the flight simulator system

The simulator system can simply be demonstrated as a closed loop, where the system takes attended values from Joystick and then after some processing operations (calculations), it will produce the output simulated values. The demonstration is extended to give more details about the system to make a sense about its data and operations flows, as the flowchart in the Fig. 4.

THE IMPLEMENTED FLIGHT SIMULATOR GAUGES

In this study the established Flight Simulator system has several gauges that could be classified into three main groups of gauges, each one of these gauges was implemented as alone ActiveX control, written as an independent programs. Eleven gauges were reconstructed to cover the constructed system, then there are eleven ActiveX programs, each ActiveX control program has three stages, these stages are:

- The drawing of the entire gauge bitmap (except the Horizon and Altimeter gauges, since they are different in their shapes, so they were drawn by using the graphical functions without bitmap).
- The calculations of the gauge parameters.
- The drawing of the gauge moving pointer (except the Horizon gauge).

The whole Flight Simulator gauges can be categorized into the following types of gauges:

CONTROL SURFACES GAUGES

The control surfaces are the electromechanical components that are responsible for driving the flight, they are represented by gauges controlled by the joystick, the control surfaces are as follows[4]:

Throttle: It's the element, which controls the RPM (Round Per Minute) of the flight engine, using the joystick (Fig. 5).

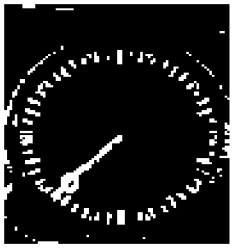

Fig. 5: Throttle gauge

6Elevator: It is the element, which controls the Pitch (climbs/ descents) operations by changing the elevator using the joystick of the flight either Pitch up (climbs) or Pitch down (descends), moving around X-axis (Fig. 6).

Fig. 6: Elevator gauge

Aileron: It is the control element, which is responsible for rolling the flight, it moves around z-axis, making the flight change its direction and causing the Roll activity (Fig. 7).

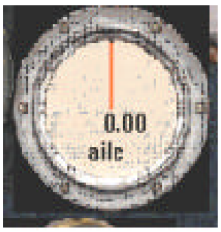

Fig. 7: Aileron gauge

Rudder: It is the control element, which is responsible for making the flight skewed around y-axis, causing the Yaw activity, by using the joystick (Fig. 8).

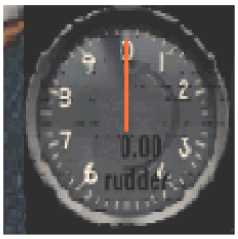

Fig. 8: Rudder gauge

The four control gauges are implemented in the following steps:

A. Drawing the bitmap, which is activated in the container program.

B. Redrawing the gauge pointer to show its movement according to the following programming steps:

```
a=(th2-th1)/(max Value -min Value)
b=th1-a*old Value
theta=a*oldValue +b
if ((oldValue >= minValue) and
(oldValue <= minValue))
Xnew=xcenter+0.83*r*sin(theta *DegreetoRadian)
Ynew=ycenter-0.83*r*cos(theta * DegreetoRadian)
Draw Line ( Xcenter, Ycenter),( Xnew, Ynew)
```

where:

th1 = -180 (It is the starting angle of the pointer (in degrees))

th2 = 180 (It is the ending angle of the pointer (in degrees))

r = 60 (It is the radius of the gauge circle)

(Xcenter=0, Ycenter=0) They are the coordinates of the central point of the gauge circle that the pointer rotating around it.

(Xnew, Ynew) They are the coordinates of the pointer's pointing end, they represent the new position of the pointer for each change in the Rudder guiding value.

C. Performing the required calculations. This gauges depends on some parameters (attended Value, ControlValue, maximum rate of ControlValue change, maximum ControlValue, minimum ControlValue, deltatime), this stage implies the following programming steps:

D = *Attended* ControlValue – ControlValue
Mx = *MaxRate* ControlValue * *deltatime*
If D > *Mx Then D* = *Mx* *Else*
If D < *-Mx Then D* = *-Mx*
End if
ControlValue = ControlValue + *D*
If ControlValue < in ControlValue *Then*
ControlValue = *Min* ControlValue
Else
If ControlValue > Max Controlvalue *Then*
ControlValue = *Max* ControlValue
End if

FLIGHT PARAMETERS

These gauges present the change in flight information due to the flight driving (through the control surfaces), the included flight parameters are as follows:

Pitch: The value of this gauge is represented by Pitch gauge the ascending pitch up or descending pitch down of the flight around x-axis, the change in its value depends on the control surface (Elevator) (Fig. 9).

Fig. 9: Pitch gauge

Its implementation is composed of the following stages:

A. Drawing the bitmap, which is activated by the container program.
B. Redrawing the gauge pointer, to present the new values of Pitch parameter, the redraw implementation implies the following programming steps:

```
a      =  ((th2-th1)/(maxValue -minValue)
b      =  th1-(a*oldValue)
theta  =  a*old Value +b
If (oldValue >= minValue) and (oldValue <= maxValue)
Xnew   =  xcenter+0.83*r*sin (theta * DgreetoRadian)
Ynew   =  ycenter- 0.83*r*cos (theta * DgreetoRadian)
DrawLine ( Xcenter, Ycenter),( Xnew, Ynew)
```

where,

th1=-180 (It is the starting angle of the pointer (in degrees))
th2=180 (It is the ending angle of the pointer (in degrees))
r=60 It is the radius of the gauge circle
(Xcenter=0; Ycenter=0 They are the coordinates of the central point of the gauge circle that the pointer rotates around.
(Xnew, Ynew) They are the coordinates of the pointer's end point, they represent the new position of the pointer after each change in the Pitch guiding value such that the sense of the pointer movement will match the change in Pitch value.

C. Performing the required calculations. This gauge depends on some parameters (Elevator, Elevator to Pitch, Height Speed Effect, Weight Effect, maximum rate of pitch change, deltatime), the stage implies the following programming steps:

```
D    =  Elevator * ElevatortoPitch * HeightSpeedEffect *
        WeightEffect
Mx   =  MaxRatePitch * deltatime
If D  >  Mx Then D = Mx   Else
If D <   -Mx Then
D    =   -Mx
Pitch = Pitch + D
If Pitch>90 Then Pitch = -Pitch +180
Heading=Heading+180
If Heading>=360 then Heading=Heading-360
Else if Pitch<-90 then Pitch=-Pitch-180
Heading=Heading+180
If Heading>=360 then Heading=Heading-360
```

Yaw: This flight angle is represented by Yaw gauge, yawing is responsible for turning the flight around y-axis, which makes the flight slightly skewed depending on the control surface (Rudder) (Fig. 10).

Fig. 10: Yaw gauge

Its implementation consists of the following steps:

A. Drawing the bitmap, which is activated by container program.
B. Redrawing the yaw gauge's pointer and its movements will reflect the changes, which occurred in Yaw, its implementation is the same as in the step B in above paragraph.
C. Performing the required calculations. This gauge depends on the parameters (Rudder, Rudder to Yaw, height speed effect, weight effect, maximum rate Yaw, deltatime); its implementation implies the following programming steps:

```
D  =  Rudder * RuddertoYaw * HeightSpeedEffect
      * WeightEffect
Mx =  MaxRateYaw * deltatime
If D > Mx Then D = Mx
Else If D < -Mx Then D = -Mx
Yaw = Yaw + D
```

RPM (Round Per Minute) Gauge: This gauge represents the increase or decrease in the power of the flight engine, which depends on throttle control that is changed by the joystick as the trainee desires (Fig. 11).

Fig. 11: RPM gauge

Its implementation implies the following steps:

A. Drawing the bitmap, which was initialized in the container program.
B. Redrawing the RPM gauge's pointer such that its movement should reflect the change, which occurred in RPM. its implementation is the same as in the step B in above paragraph.
C. Performing the required calculations. This gauge depends on some parameters (engine, maximum RPM,

throttle, throttletoRPM1, throttletoRPM2, oxygen effect, AltimetertoRPM, Altimeter). This stage is implemented by the following programming steps:

```
If Engine = 1 Then
AttendedRPM=MaxRPM*(Throttle*(ThrottletoRPM1+
ThrottletoRPM2* Throttle))
OxygenEffect = Exp (-AltimetertoRPM * Altimeter)
AttendedRPM = AttendedRPM * OxygenEffect
D  = AttendedRPM – RPM
Mx = MaxRateRPM * deltatime
If D > Mx Then D = Mx    Else   If D < -Mx Then D = -Mx
End if
RPM   = RPM + D
If RPM > MaxRPM  Then  RPM  =  MaxRPM  Else  RPM
          = 0
End if
```

Roll: The value of the flight parameter is represented by horizon gauge, it reflects the turning of the flight around z-axis, which making the flight changing its direction left or right, depending on
the control surface (Aileron) (Fig.12).

Fig. 12: Roll gauge

Its implementation consists of the following steps:

A. Drawing the Sky as a blue filled circle.
B. Drawing the Earth as a movable green chord according to the changing values of the roll. The movement of this chord depends on the following programming steps:

```
scl=r/maxpitch
d=newpitch*scl
sq=(r*r)-(d*d)
theta=atan(d/sqrt(sq))
p1.x=xcenter-r*cos(theta)
p1.y=ycenter+r*sin(theta)
p2.x=xc+r*cos(theta)
p2.y=yc+r*sin(theta)
rdroll=(oldroll*DegrretoRadian)
p11.x = (p1.x-xc)*cos(rdroll)+(p1.y-yc)*sin(rdroll)+xc
p11.y = (p1.x-xc)*sin(rdroll)+(p1.y-yc)*cos(rdroll)+yc
p22.x = (p2.x-xc)*cos(rdroll)+(p2.y-yc)*sin(rdroll)+xc
p22.y = (p2.x-xc)*sin(rdroll)+(p2.y-yc)*cos(rdroll)+yc
Draw Chord (p11and22)
```

where,

p1 It is the starting point of the chord
p1.x It is the x-coordinate of the point p1
p1.y It is the y-coordinate of the point p1
p2 It is the ending point of the chord
p2.x It is the x-coordinate of the point p2
p2.y It is the y-coordinate of the point p2
p11 It is the new starting point of the chord (after rolling)
p22 It is the new ending point of the chord (after rolling)
p11.x It is the x-coordinate of the point p11
p11.y It is the y-coordinate of the point p11
p22.x It is the x-coordinate of the point p22
p22.y It is the y-coordinate of the point p22
xc, yc The coordinates of the center point of the circle
r It is the radius of the circle

C. Performing the required calculations. This gauge depends on the parameters (Aileron, Aileron to Roll, height speed effect, cross wind effect, weight effect, maximum Rate Roll, delta time); this stage is implemented by the following programming steps:

```
D=  Aileron*Aileron2Roll*HeightSpeedEffect*
    CrossWindEffect*WeightEffect
Mx = MaxRateRoll * deltatime
If D > Mx  Then D = Mx    Else  If D < -Mx Then D = -Mx
           Roll = Roll + D
           If Roll > 180 Then Roll = Roll - 360
           Else if Roll < -180 Then Roll = Roll + 360
```

NAVIGATION PARAMETERS

These parameters are very important to pilot because they help him to make good assessments of the course. They provide information about the geographic position of the flight airplane and its motion direction
and speed. The navigation parameters include:

Heading: This gauge indicates the flight motion direction (in degrees) relative to the geographic north. Fig.13. Its implementation implies the following stages:

A. Drawing the bitmap, which is enabled by the container program.
B. Redrawing the pointer by applying the following programming steps:

```
Xnew=xcenter+0.83*r*sin
(old heading *degreetoRadian)
Ynew=ycenter-0.83*r*cos(oldHeading * DegreetoRadian)
DrawLine (Xcenter, Ycenter),( Xnew, Ynew)
```

Where:

r=60 It is the radius of the gauge circle
(Xcenter=0, Ycenter=0) They are the coordinates of the central point of the gauge circle that the pointer rotates around.
(Xnew, Ynew) They are the coordinates of the pointer's pointing end, they represent the new position of the pointing end after each change in the Heading value.

C. Performing the required calculations. This gauge depends on the parameters (Roll, Rolltohead, Yaw, Yawtohead, speedeffect, Deltatime) this stage implies the following programming steps:

deltaHead=(Roll*RolltoHead+Yaw*YawtoHead)*Speed Effect*deltatime
Head = Head + deltaHead
If Head < 0 Then Head = Head + 360 Else If Head >= 360 Then Head = Head – 360
End if

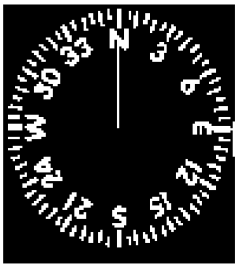

Fig. 13: Heading gauge

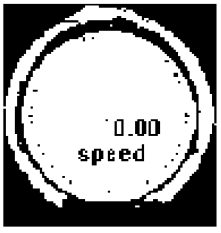

Fig. 14: Speed gauge

Speed: This gauge represents the actual speed of the flight. Fig.14. Its implementation consists of the following steps:

A. Drawing the bitmap which is enabled by the container program.
B. Redrawing the speed pointer such that its movement will reflect the variation in speed, this stage is implemented as follows:

a=((th2-th1)/max Speed
b=th1
theta=a* Speed +b
If ((oldSpeed >= minSpeed) and (oldSpeed <= maxSpeed))
Xnew=xcenter+ 0.83*r*sin (theta *DegreetoRadian)
Ynew=ycenter- 0.83*r*cos (theta * DegreetoRadian)
DrawLine (Xcenter, Ycenter),(Xnew, Ynew)
where,
th1=-135 It is the starting angle of the pointer (in degrees)
th2=135 It is the ending angle of the pointer (in degrees)
r=60 It is the radius of the gauge circle
(Xcenter=0; Ycenter=0) They are the coordinates of the center point of the gauge circle that the pointer rotates around.
(Xnew, Ynew) They are the coordinates of the pointer's pointing end, they represent the new position of the pointing end after each change in the speed value.

C. Performing the required calculations. It uses several effective parameters, such as (the RPM, maximum RPM, maximum speed of the flight, weight effect, dragcoefficient, airspeed, pitch, degreetoradian, decrementratespeed, incrementratespeed, deltatime), depending on the following programming steps:

R = RPM / MaxRPM
AttendedSpeed=MaxSpeed(R*(RPM2Spd1+RPMtoSpd2*R))* WeightEffect*
*AttendedSpeed = AttendedSpeed - DragCoeffecient * AirSpeed*
If Pitch > 0 Then
*AttendedSpeed=AttendedSpeed*Cos(Pitch*Degreeto Radian)*
Else If Pitch < 0 Then
*AttendedSpeed=AttendedSpeed*1+Cos(Pitch* DegreetoRadian)*
D = AttendedSpeed – Speed
*Mx = IncrementRateSpeed * deltatime*
*Mn = DecrementRateSpeed * deltatime*
If D > Mx Then D = Mx
Else If D < Mn Then D = Mn
Speed = Speed + D
If Speed > MaxSpeed Then Speed = MaxSpeed else If Speed<0 then Speed=0

Altimeter: This gauge is different from other gauges in its shape and it gives the height of the flight, it is composed of three vertical bar regions (red, orange, blue) and there is a movable blue indicator that moves up and down according to the Height value as shown in the following programming steps:

```
If (height >=0 and height <= 1500) then
scalefactor = rect1.Height / 1500.0
drwheight = rect1.Height - (height *scalefactor)
DrawLine (0,drw_height), (rect1.Width,drw_height)
```

where,

rect1.Height It is the height of the height of the rectangular gauge

rect1.Width It is the width of the height rectangular gauge

drw_height It is the value of the scaled (mapped) height according to the boundaries of the main rectangle which contains the three small bars (Fig. 15).

When the indicator points in the red region, it means the flying is in a dangerous height (between 0-300m), when the indicator points in the blue region it means the flying is in a less dangerous height (between 500-1500m),

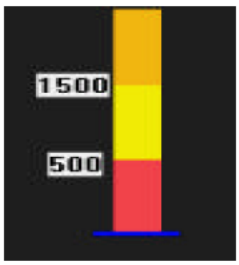

Fig. 15: Altimeter gauge

when the indicator points in the orange region it means the flight is in a safe height (from 1500 m and over). The operational program of the gauge object depends in its work on some parameters (Pitch, degreetoradain, deltatime, height) as shown in the following programming steps:

```
deltaheight = Speed* Sin(Pitch*DegreetoRadain)* deltatime
height = height + deltaheight
If height < 0 Then height = 0
```

Latitude and longitude: These two parameters considered as intermediate results indicate the geographic position of the flight in the world according to some parameters (speed, pitch, degree to radian, delta time) as in the following programming steps:

```
dHorz = Speed*Cos(Pitch*DegreetoRadian)*deltatime
LatScl = 6372* DegreetoRadian
LonScl=LatScl *Cos (Latitude*DegreetoRadian)
Longitude=Longitude+dHorz*Sin (Heading*Degreeto Radian)/LonScl
atitude=Latitude+dHorz*Cos (Heading *Degreeto Radian)/LatScl
```

FEATURES OF THE IMPLEMENTED FLIGHT SIMULATOR

The Fight Simulator system as shown in fig. 16 is implemented as a homogeneous collection composed of:

- The Cockpit as a panel shape (dashboard) which contains eleven gauges (each of them is an ActiveX program alone) with animated pointers as follows:

a Control surfaces set of gauges (Elevator, Aileron, Rudder, Throttle)

b Response set of gauges (Pitch, Horizon, Yaw, Rpm, Speed, Altimeter)

c Navigation Gauge: Head gauge

- The External view which is drawn using OpenGL, the simulated view of Earth surface (drawn as a plane) and Sky (drawn as a sphere), it gives the sense of fly when the simulator is running in any combination of axes (Up/ Skewed, Down/ Skewed, Skewed left or right).
- The start button for starting the Whole Flight Simulation task.
- The Joystick represents the input or controlling tool of the flight simulator, which is implemented using DirectX (DirectInput), which is work under OpenGL environment as an external class.

Fig. 16: Final flight simulator

REFERENCES:

1. Forsstrom, K. S., Array Processors in Real-Time Flight Simulation, Computer, p: 62.
2. Shafer, M. F., 1992. NASA Technical Memorandum 4396, In: Flight Simulation Studies At the NASA Dryden Flight Research Facility.