International Journal of Control and Automation
Vol.9, No.1 (2016), pp.347-358
http://dx.doi.org/10.14257/ijca.2016.9.1.30

Development of Key Functions for Flight Simulator

ChungJae Lee[1], SeokYoon Kang[1], Seog Geun Kang[2], Kyong Hoon Kim[1] and Ki-Il Kim[1*]

[1]Department of Informatics, Research Center for Aerospace Parts Technology
[2]Department of Semiconductor Engineering
Gyeongsang National University
501 Jinju-Dearo, Korea, 52828
**kikim@gnu.ac.kr*

Abstract

A flight simulator is usually developed to verify the operations of aircraft and train the pilot through it. For this goal, it is necessary to include a variety of functions that work in real-time way. However, due to a huge volume of terrain data, graphic rendering speed becomes slow whenever additional computing processing is demanded. To address this problem, in this paper, we propose two techniques to speed up rendering speed for new flight simulator under our software architecture and module. Both hybrid cloud effect and adaptive culling scheme are proposed and implemented by OpenSceneGraph(OSG) library on Windows. Through diverse experimental results to verify their suitability, we demonstrate that acceptable rendering speed is maintained or enhanced without regard to additional computing overhead.

Keywords: *flight simulator, OpenScenGraph, Open Source*

1. Introduction

Recently, many simulation programs have been developed to verify new system and make users understood it in advance. Furthermore, these systems are expanded to cover entertainment system such as game as well as training system in order to prevent the risk or accident virtually. Among them, flight simulator is used to train pilot by providing the diverse situations including flying environment according to type of aircrafts. Without this system, it is very hard to make specific situations such as encountering enemy fighters as well as schedule many chances for actual flight due to high cost of operation. By this reason, several simulators have been developed for general or specific purpose and widely used in real world. In the point of operations, since the flight simulator is greatly different from other simulators such as automobile and ship in that their operations are usually accomplished at all possible spaces such as atmosphere and ground. In addition, since flight simulator should support high and wide point of view in three dimensional space, its computation complexity become higher than others at usual scenarios. Furthermore, realistic scenarios are in accordance with high dynamics of vehicle to increase immersion in flight simulator as indicated in [1].

Based on above needs, two types of flight simulator, commercial and open source, have been developed and released to public users. Microsoft Flight Simulator X (FSX) is good example of commercial one. But, even though they provide sufficient SDKs to develop new function, source code is not open to public users. On the other hand, Flight Gear, open source flight simulator, provides full source code for development. However, it is hardly used to verify flight control because they do not provide external interface. So,

[1] This paper is a revised and expanded version of a paper entitled "Software Architecture for Open Source Based Flight Simulator" presented at MAS, Jeju, Nov. 2015.

its main usage is limited to game or reference model to develop new one. In summary, two types of simulator have advantage or disadvantage in terms of extensibility which is to verify the operations and provide software platform.

Moreover, mentioned analysis for current flight simulator implies two major requirements for new flight simulator. One is open source and the other is compatibility with other systems. However, as described before, current existing simulators cannot meet above requirements together yet. For this goal, in this paper, we develop a new flight simulator based on OSG library as well as software architecture in which rendering and flight control software are separately implemented. By using OSG[2][3], we are able to recognize the possibility of open source library to generate graphical scene for flight simulator. In addition, separate architecture that is compatible with other software such as Matlab helps us to verify correctness of the several flight control algorithms.

In addition to addressed functional demands, rendering speed is the most important measurable performance metric because flight simulator work in real-time way without regardless of processing various functions and a huge volume of terrain data together. Assume that the rendering speed is below than threshold, the users will not choose this simulator because they are very sensitive to uncomfortable scene transition. To meet above demands, in this paper, we propose two adaptive schemes to speed up rendering speed without regards to terrain data as well as additional computing needs in it. They include cloud effect and dynamic culling for weather effect. Since existing typical solutions are not sufficient for fast rendering, we propose new schemes to increase the rendering speed in software approach. And then, we present the experimental results for each scheme in several scenarios. Through this results, we prove the suitability of the proposed scheme in terms of the rendering speeds by providing measured values and comparing it with comparative simulator.

This paper is organized as follows. In section 2, we briefly review the current flight simulators and their outstanding properties. And then, our new flight simulator is explained in section 3. New schemes to improve rendering speed and experimental results are presented in section 4 and 5, respectively. Finally, conclusion and further works are given in section 5.

2. Review of Flight Simulator

Generally, the main objective of the flight simulator is to train the pilots by accumulating their flight experience. In the initial stage, flight simulator was used for only military purpose. But, as the airliner has been gradually included in it, it is widely used for various purposes.

The good example of flight simulator is Microsoft Flight Simulator X (FSX), Lockheed Martin Prepar3D, Laminar Research X-Plane, Lead Pursuit Falcon 4.0 and Flight-Gear [4] Among them, FSX[5] is the most popular and well referred one. While starting as game, it provides the most realistic graphics and abundant dynamic models for aircrafts. In addition, joystick and rudder are available to fly a plane. Furthermore, formation flight can be accomplished by multi-plays. Whereas, one of outstanding features of Prepar3D is education scenarios directly created by user. Another flight simulator worthwhile mentioning is Flight-Gear. It is developed as open source so its usage is free. Flight-Gear is continually upgraded by the users worldwide who contribute to new functions or enhancement of the existing ones. By this simulator, many researchers and developers get the important information and shares their experience. To accelerate this collaboration, Flight-Gear is designed to include software framework for further extension and improvement.

However, even though many available flight simulators have been developed, its core algorithms and development techniques are hidden from developers. By this reason, it is very hard to use the existing scheme to verify the operations of new control algorithm employed in dynamic model. Thus, when it comes to develop new type of aircraft, new flight simulator is usually developed. However, due to expensive development cost, it is desirable to take new software platform for flight simulator into account. This is motivation of open source-based new platform for flight simulator which can be used to verify the operation with the external interface for dynamic model of aircraft.

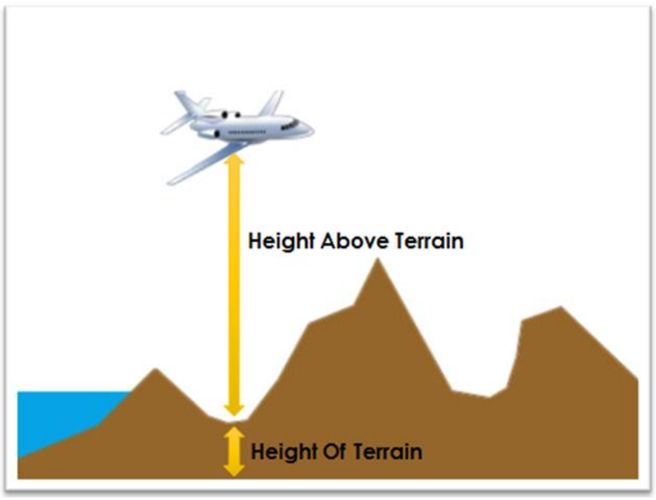


Figure 1. Definition of HAT and HOT

3. A New Flight Simulator

In this section, we describe proposed flight simulator in the point of software architecture and implemented major functions. The proposed flight simulation is to provide the realistic scenarios as well as main functions to help training. Also, it includes the special effects which are required to increase sense of reality.

3.1. Two Key Components for Flight Simulator

A rendering software for flight simulation is largely affected by the computation technique which is used to handle aircraft information, external weather environment and objects such as other aircrafts and missiles. Thus, it is necessary to define how to compute the essential information and meet requirement for rendering. Specially, prior to implementation, core information such as Height Of Terrain(HOT), Height Above Terrain(HAT), Line Of Sight(LOS) should be defined and computed in flight simulator as defined in Figure 1. Also, it is demanded how to increase rendering speed without regard to additional objects for external environments. In this section, we describe our approaches for these two areas.

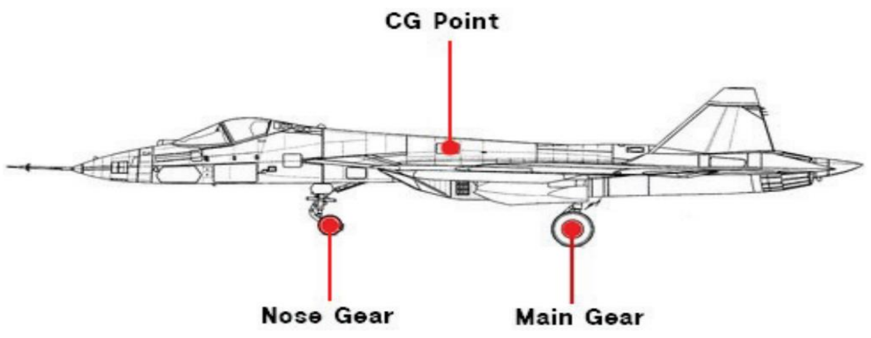


Figure 2. Core points

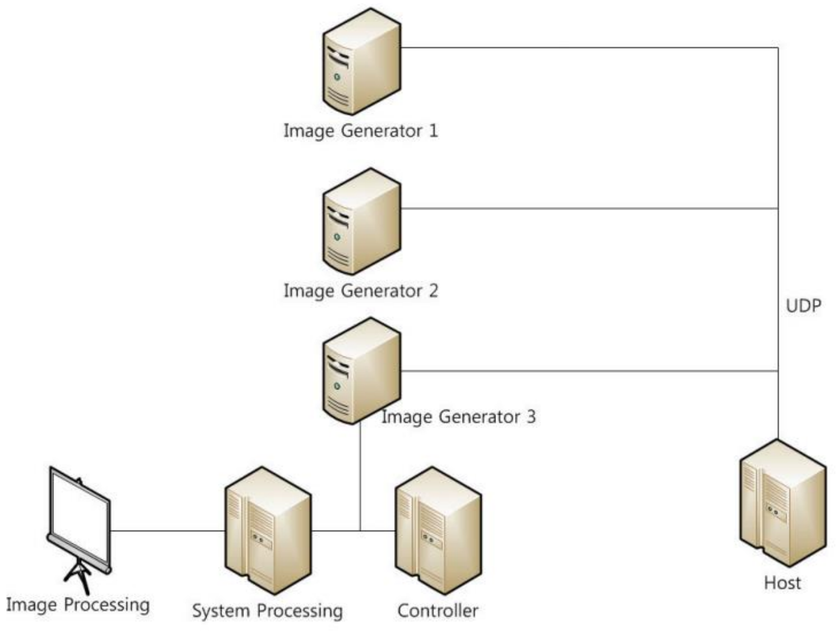


Figure 3. System Architecture

First, the mentioned three parameters are related to terrain data. Usually, Digital Terrain Elevation Data(DTED) [6] is generally used to create terrain data by connecting altitude information measured by satellite and aircraft. And then, DTED is mapped with satellite picture or ground texture. However, there are some problems to use this ground on simulator in case of landing and takeoff. This problem is caused by the altitude difference on DTED and created ground. Due to this difference, despite the aircraft still flies in the air, it seems that the position of aircraft is on the ground or below it. In order to solve this problem and provide the appropriate altitude information in the user's view point, it is required to compute HOT, HAT and LOS. So, these values are essential information to be obtained according to current type of aircrafts. To compute these values, we propose new technique to get LOS, HAT, and HOT together. Among them, both HAT and HOT can be computed by Center of Gravity Point (CG), Nose Gear (NG) and two Main Gears (MG) which is shown in Figure 2. The reason to consider nose gear and two main gears is that these equipment are actual touching parts on the ground. Following three steps are taken to compute HOT and HAT. 1) Collect position information including latitude, longitude, pitch, roll and yaw of aircraft. 2) Compute the position information for three gears based on CG of the aircraft. Table 1 shows the relative position information from the CG where each value represents the distance from ground in meter. 3) Based on this information, we can first compute the HAT value which is represented by AltitudeHAT in Equation 1. And then, HOT, named AltitudeHOT, is finally determined.

Table 1. Position of points

Point	Coordinate(X,Y,Z)
CG	(0,0,0)
Nose	(0.021, 4.698, -1.962)
Main1	(-0.973, -0.488, -1.909)
Main2	(0.973, -0.488, -1.909)

$$AltitudeHOT = AltitudeOrigin - AltitudeHAT \qquad (1)$$

Second, it is related to rendering technique for external environments surrounding the aircraft such as weather. However, since their reality is mostly dependent on the number of composing elements under the assumption that more computing resource is required for more realistic scene than normal one, it is required to maintain acceptable rendering speed despite additional processing. Among many objects, particle system for cloud, rain, snow are one of great source for overhead. Even though particle system has advantage in reality due to accurate description of movement and appearance with large number of small elements, the overhead become larger and larger as the number of elements increases. To prevent this problem, two main algorithms such as Level of Detail (LOD) and culling [7][8] have been already proposed. However, current static approach is not suitable for flight simulator where various objectives form such a dynamic environment. So, dynamic and adjustable approach is highly demanded.

3.2. Software Architecture

In this section, we describe our software architecture and modules to implement two mentioned components. The system architecture for flight simulator consists of host and image generator as shown in Figure 3. Host is responsible for configuration of simulators including dynamics model of aircraft. This means that environment parameters, point of view as well as configuration of the related parameters for each scenario is requested by the host. Image generator is to render images in three dimensional space in order to provide realistic flight situation. In case of image generator, it is required to visualize the terrain data according to Field of View (FOV) through different channel on respective system. In our system, we introduce three channels for FOV with 160 degree width and 40 degree height. The three separate channels are for system processing, image processing and system control part. Image processing part is responsible for visualizing the computed data in three dimensional space. For example, this part includes the processing for snow, rain, special values such as HOT, HAT, LOS. Also, special effect for explosion is also implemented in this one. In addition to these functions, most important component in it is to maintain acceptable Frames Per Second (FPS), 60Hz in our work, by employing new overhead processing module.

The host and image generator use UDP socket for bi-directional communications. The transferred information includes the status of aircraft, simulation environments, information such as HAT/HOT/LOS. In this architecture, software modules are shown in Figure 4. Each module and its function are explained in next.

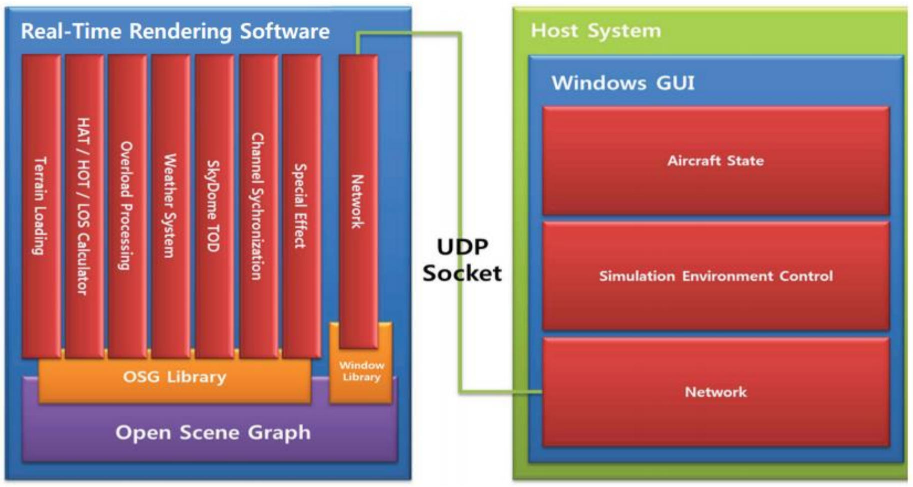


Figure 4. Software Modules

- Terrain Loading: This module is to process terrain data which is created by combining DTED and satellite image. In this paper, we use the terrain data describing area around one of airports in Korea. It consists of 100km DTED satellite and 10m resolution image. And, TerraVista software is used to create terrain data.
- HAT/HOT/LOS Calculator: For the simulator, it is very important to maintain the distance between aircraft and ground in case of landing, takeoff and bomb drop. Thus, it is necessary to keep its accuracy and computation during flight. For the continuous confirmation of this value, we check it through text box.
- Weather System : This module consists of three major weather effects, that is, snow, rain and cloud. Specially, three different levels are employed to represent the strength or density of each component according to the user input.
- Sky Dome and Time of Day : For the better reality, sky dome and date information are required. The date and time is controlled by the host and different scenario is implemented according to them. To achieve this, host sends the time information at every second. Depending on this time, sunlight and location of sun are determined.
- Special Effect : In the case of military aircraft, the special effects such as launching missile and explosion is demanded to provide user more realistic scene. They are implemented by particle system of OSG.
- Channel Synchronization : In our system, the proposed simulation system consists of three channels. Thus, each channel displays the identical scene or aircraft status with different FOV in three channels. In our scheme, host sends data in 60Hz frequency by multicast communication so image generator system adjusts the rendering speed as 60Hz in order to accomplish synchronization.
- Overload Processing : Usually, flight simulation has the required minimum rendering speed. Thus, it is required to keep higher rendering speed than requirement. But, due to additional processing overhead for weather system and computing HAT, it is very hard to maintain minimum rendering speed without new functions. They are included in overload processing module and executed when the minimum rendering speed is not preserved.
- Network : The data communication between image generator and host is implemented by windsock as UDP packet. Host sends aircraft status including

location as well as event data to trigger configuration changes in image generator software.

3.3. Enhancing Rendering Speed and Implementation

In this section, we present the proposed schemes to increase and maintain rendering speed and their implementation through dynamic culling and cloud effect.

Cloud Effect for the realistic flight environments, it is essential to provide a variety of cloud effects in the simulator. Also, according to the fact that cloud effect is affected by the volume and wind in its visuality, it is commonly known that there are many common properties between cloud and fog. For cloud modeling, we make use of particle system in a way of applying physical movement of vapor. But, this particle system incurs additional overhead for rendering. Based on this analysis, in this paper, we propose new scheme which is to dynamically adapt the fog effect and adjust compositional parameters according to situation. The main task is to employ hierarchical cloud modeling in order to maintain the acceptable rendering speed. More detailed, two approaches are taken to increase the rendering speed for the cloud effect. One is to introduce texture mapping and fog effect instead of particle system. The other is apply the texture mapping and particle according to the situation. Since texture mapping has problem of degradation in realistic visuality in near distance, we introduce two cloud layers and use fog effect to improve visuality according to position of aircraft. Through two schemes, we can maintain the rendering speed as well as enhance reality for cloud at the same time. To implement the cloud, the following variables are defined in Table 2 where they are transferred from the image generators.

Table 2. Variables for Cloud Effect

Variable name	Meaning	Variable Type
wZ	Altitude of Aircraft	Double
Density	Density of fog effect	Float
Bot	Bottom position of cloud layer	int
top	Top position of cloud layer	int
Cloud_outside	Height of cloud entry layer	int

The values in Table 2 are used to present fog effect according to the altitude. Furthermore, we define new function, CloudsDensity, as shown below. to set density of fog effect according to the altitude of aircraft. If current position of aircraft is between two cloud layers, it implies that aircraft travels in the cloud. So, we increase density of fog in order to implement foggy effect. Otherwise, when aircraft locates outside of cloud layer, we differentiate the density value by referring to entry point of cloud and current location. For example, if an aircraft is approaching cloud layer slowly, density of fog gradually increases to display realistic scene.

```
function CloudsDensity (Density, bot, top, wZ, cloud_outside)

  if ( wZ >= bot and wZ <= top)
          Density = 0.0005f
  else if (wZ >= (bot-cloud_outside) and wZ < bot)
          Density = (wZ - (bot - cloud_outside) * 0.000001f
  else if (wZ < (top + cloud_outside) and wZ > top)
          Density = ((top+cloud_outside) - wZ) * 0.000001f
```

In addition to above adaptive control for density of fog, we adapt texture mapping and particle system adaptively according to current locations. This is to increase reality by replacing fog effect by the particle system. Since particle system provides more realistic effect than texture mapping, we apply particle system partly in case when aircraft travels in the cloud. To implement this function, we create clouds around the field of view to produce natural effect. The flow chart for this approach is shown in Figure 5.

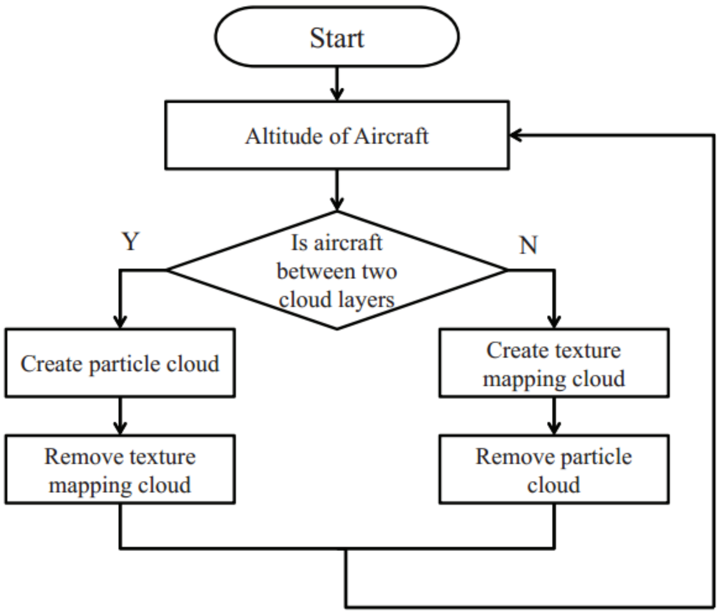

Start
Altitude of Aircraft
Is aircraft between two cloud layers
Y
N
Create particle cloud
Remove texture mapping cloud
Create texture mapping cloud
Remove particle cloud

Figure 5. Flowchart of Cloud Effect

As illustrated in Figure 5, it is necessary to measure the current altitude of the aircraft. Depending on current location, two different actions are taken. First, if the aircraft locates in the cloud, we improve reality by creating the cloud by the particle system and removing the texture mapping. Otherwise, we enhance rendering speed by removing particle cloud and creating texture mapping. Furthermore, for the former case, we repeatedly create clouds in a predetermined distance from the current aircraft. By this way, if an aircraft approaches the nearest cloud, a new cloud is automatically ahead position within the range. By this way, we prevent drastic view transition in the display.

Dynamic Culling in addition to cloud effect, it is required to implement the external weather effect. For this effect, both customized and predefined module are the general approach [9][10]. First, in customized approach, it is possible to set different size, color, location of particle for the effect. But, this configurable property causes additional overhead as the number of particle increases so rendering speed is greatly affected by this overhead. Furthermore, weather effect disappears in the view due to no more available computing resource. So, in order to allow weather effect as well as improve rendering speed, we propose dynamic culling scheme depending on change in weather. In the proposed scheme, following four steps are taken as an extension of our previous work[11].

- Measuring rendering speed
- Establishing line of sight
- Determining maximum culling coverage
- Determining new culling ratio

Mentioned four steps are to maintain rendering speed by adjusting culling ratio without regard to overhead of weather effect. First, if the new additional computing overhead such as weather effect is caused, we measure how much rendering speed are reduced. Since the rendering speed is affected by type of weather effect and density of it, we periodically measure the rendering speed. Second step is to define temporary distance of the visuality considering reality depending on effect type because it is dynamically changed. Next, culling coverage is newly configured to complement the reduced rendering speed. At this step, distance of the visuality usually becomes longer than it in second step so culling starting point is fixed to the distance of the visuality. This culling starting point can be represented as Equation 2.

$$Near\_Far\_Ratio = \frac{Default\_NearPlane\_Distance}{Customized\_FarPlane\_Distance} \tag{2}$$

In Equation 2, Default_NearPlane_Distance represents the closest distance from viewpoint in the camera while Customized_FarPlane_Distance does the configured distance of the visuality. In the last step, if new culling ratio computed in third step is greater than reduced rendering speed, it is required to maintain and improve the frame rate. To achieve this, we increase Near_Far_Ratio in Equation 2 until the difference between improved rendering speed and reduced one is bounded within the predetermined threshold.

4. Implementation and Experimental Results

4.1. Development System

For actual implementation of flight simulator, we employ OpenSceneGraph (OSG) which is open source graphic middleware as well as widely used in education, game, medical and other visualization software. The simulator system consists of hardware described in Table 3. For the comparison, we run FSX for several similar scenarios.

Table 3. Development System

Hardware	Specification
CPU	Intel i7-3770 3.40GHz
Memory	8GB
VGA	NVDIA GeForce GTX 680
Terrain Data	3.17 GB

4.2. Experimental Results

First experimental results are shown in Figure 6 in the aspect of cloud effect. In Figure 6, texture mapping technique is used to render cloud while particle system is introduced when an aircraft is entering into the cloud in Figure 7. In addition to their visual effect, another parameter, FPS, are compared in the below Table 4.

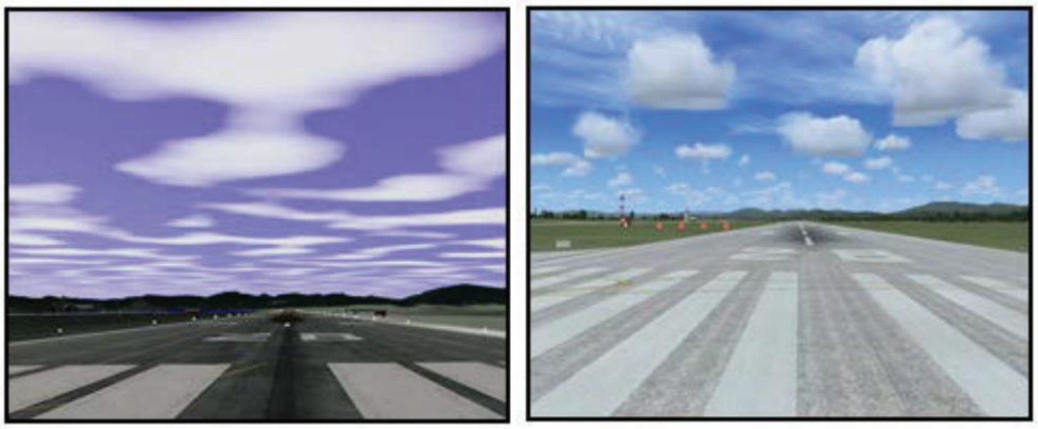


Figure 6. Cloud Rendering Before Entering Cloud, Left : Ours, Right : FSX

Table 4. Comparison of FPS

Test Number	Particle System	Texture Mapping	Proposed
1	23.36	59.84	56.48
2	32.82	58.82	57.76
3	30.45	59.46	54.37

In Table 4, we run ten times experiments and get the average FPS for them. As illustrated in Table 4, the worst FPS is observed in particle system while texture mapping reveals the best one. The proposed scheme shows the acceptable FPS in that there is slight difference between texture mapping and proposed scheme. But, in the point of reality, there is great gap between two schemes. This result is brought from our core technology that applies cloud effect according to current rendering speed.

Another experiment result is for dynamic culling. For this experiment, we use high resolution terrain data and then rain effect additionally. We run the several scenarios according to number of vertex as well as density of weather effect as shown in Table 5 Example scene for ran effect is shown in Figure 8. The proposed scheme shows the similar effect to the FSX. Moreover, the proposed scheme improves the reality by implementing the situation that the rain is approach toward the aircraft.

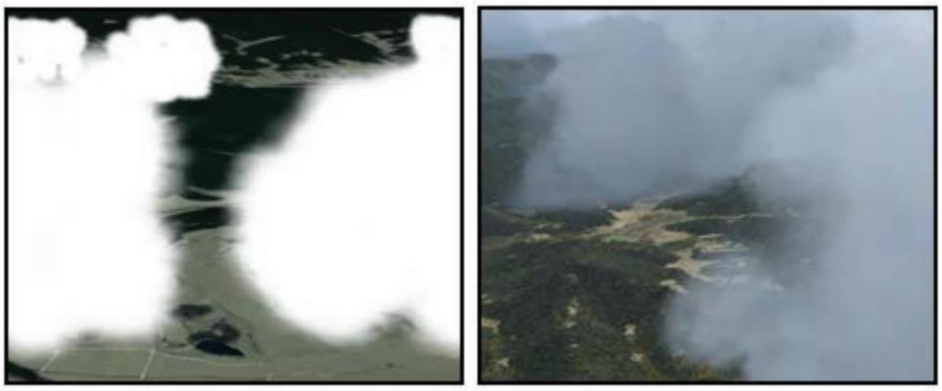


Figure 7. Cloud Rendering After Entering Cloud, Left: Ours, Right: FSX

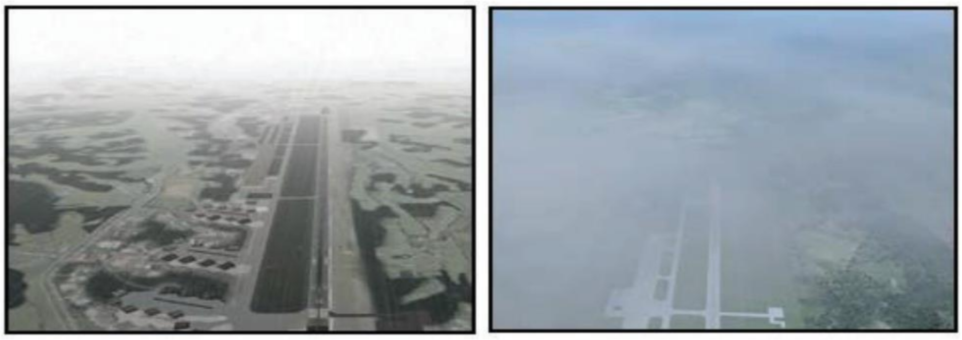


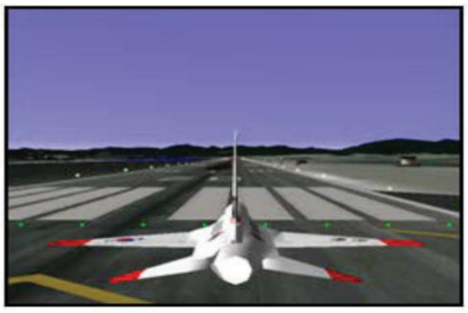
Figure 8. Rain Effect, Left : Ours, Right : FSX

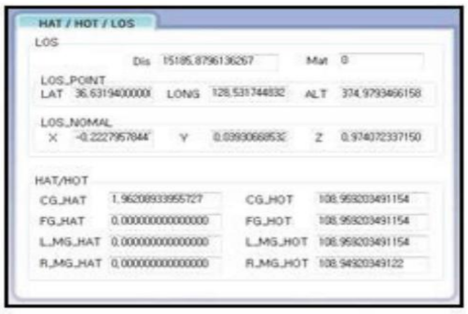
Table 5. Comparison of FPS for Weather Effect

Vertex	Density of Weather Effect	Fixed Rate Culling	Proposed
100,000	Default	60	60
	Light	49	60
	Medium	45	60
	Heavy	43	60
120,000	Default	47	60
	Light	42	60
	Medium	38	60
	Heavy	35	60
140,000	Default	44	60
	Light	37	60
	Medium	36	60
	Heavy	34	60

Through Table 5, we can identify that the proposed scheme shows the better FPS than fixed rated culling. This is because the culling ratio in the proposed scheme is adjusted dynamically depending on the vertex and density while fixed rate culling fails to work in varying condition. By the help of dynamic culling, weather effect is clearly displayed without any problem. Also, its reality seems acceptable as compared to FSX. One point worthwhile mentioning is that proposed scheme maintains 60 Hz for all cases. This is because we fixed the maximum FPS as 60 Hz in our hardware. Thus, even though the higher FPS is observed, the minimum value between observed one and 60 Hz is reported.

The last experiment is to test accuracy of our method to compute HOT, HAT, and LOS. Figure 9 shows the measured value in the rendering software. Through this experiment, we can see the identical values between measured and given value.





HAT / HOT / LOS

LOS

Dis 15185.8796136267 Mat 0

LOS_POINT

LAT 36.6319400000(LONG 128.531744832 ALT 374.9793466158

LOS_NOMAL

X -0.2227957844 Y 0.0393066853 Z 0.974072337150

HAT/HOT

CG_HAT	1.96208933955727	CG_HOT	108.959203491154
FG_HAT	0.00000000000000	FG_HOT	108.959203491154
L_MG_HAT	0.00000000000000	L_MG_HOT	108.959203491154
R_MG_HAT	0.00000000000000	R_MG_HOT	108.94920349122

Figure 9. Identifying HOT, HAT, and LOS, Left : Aircraft Position, Right : Measured Values

5. Conclusion

In this paper, we proposed how to develop flight simulator through open source middleware, OSG and prove its possibility to make use of OSG for flight simulator. The software architecture and their components are described and explained. During development, we focused on rendering speed in flight simulator by including cloud effect and dynamic culling. Each function is dynamically adjusted according to reality and current situation. Finally, we presented the experimental results depending on a variety of scenarios. Through them, we proved that the proposed simulator can meet general requirements as well as improve rendering speed at the acceptable level.

Related to this work, we continue our work in two ways. First, we are going to extend current simulator to include other functions such as lightening. Also, comparison with other simulators will be accomplished. Second, in order to speed up the rendering speed, other drawbacks will be identified and improved. This procedure will include how to use current measured values to implement essential components.

Acknowledgments

This work was supported by research fund, Gyeongsang National University, 2015 and the BK21 Plus Program (Research Team for Software Platform on Unmanned Aerial Vehicle, 21A20131600012) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education.

References

[1] X. Hu, S. Bo, Z. Huiqin, X. Bing, W. Hao and H. Ge, “Visual Simulation System for Flight Simulation based on OSG”, Proceedings of IEEE International Conference on Audio Language and Image Processing, Shanghai, China, **(2010)** November 23-25.

[2] R. Wang, Editor, “OpenSceneGraph 3 Beginner’s Guide”, Packt Publishing Ltd., Birmingham, **(2012)**.

[3] R. Wang Editor, “OpenSceneGraph 3 Cookbook”, Packt Publishing Ltd., Birmingham, **(2010)**.

[4] I. Strachan, Editor, “Jane’s Simulation and Training Systems”, Janes Information Group, Alexandria, **(2010)**.

[5] Microsoft Flight Simulator, “http://www.microsoft.com/products/games/FSXInsider/fsxlauncher.aspx”

[6] G. Miliaresis and C. Paraschou, “Vertical Accuracy of the SRTM DTED Level 1 of Create”, Journal of Applied Earth Observation and Geoinformation, vol. 7, no. 1, **(2005)**, pp. 49-59.

[7] C. Wang, H. Xu, H. Zhang and D. Han, “A Fast 2D Frustum Culling Approach”, Proceedings of IEEE International Conference on Computer Engineering and Technology, Chengdu, China, **(2010)** April 16-18.

[8] R. Zhang, “Real-Time Optimization Technology and Its Application in Terrain Rendering”, Proceedings of IEEE International Conference on Image and Signal Processing, Shanghai, China, **(2011)** October 15-17.

[9] L. Li., W. Wan, X. Li and Z. Wang, “Weather Phenomenon Simulations in 3D Virtual Scenes based on OSG Particle System”, International Communication Conference on Wireless Mobile and Computing, Shanghai, China, **(2011)** November 14-16.

[10] H. Xining, L. Ning, Z. Dinghai and X. Renjie, “Design and Application of General Data Structure for Particle System”, International Conference on Computer Science and Electronics Engineering, Hangzhou, China, **(2012)** March 23-25.

[11] C. Lee, S. Kang, S. Kang and K. Kim, “Software Architecture for Open Source Based Flight Simulator”, 4th International Conference on Modeling and Simulation, Jeju, **(2015)** November 25-28.