High Speed Visualization in the JetOS Aviation Operating System Using Hardware Acceleration

Boris Barladian[0000-0002-2391-2067], Nikolay Deryabin[0000-0003-1248-6047], Alexey Voloboy[0000-0003-1252-8294], Vladimir Galaktionov[0000-0001-6460-7539], and Lev Shapiro[0000-0002-6350-851X]

The Keldysh Institute of the Applied Mathematics of RAS, Moscow, Russia
bbarladian@gmail.com,{voloboy, vlgal, pls}@gin.keldysh.ru

Abstract. The paper discusses details of the pilot display visualization that uses the hardware acceleration capabilities of the Vivante graphics processor in the JetOS aviation operating system. Previously the OpenGL Safety Critical library was implemented without hardware acceleration. This was done in such a way because software library is easier to certify in accordance with the avionics requirements. But usage of the software OpenGL does not provide acceptable visualization speed for modern Flight Display and 3D relief applications. So more complex visualization approach utilized the GPU acceleration capabilities was elaborated. Although the OpenGL library was implemented for a specific GPU and took into account its specificity, the described approach to adapt the MESA open source library can be used for other GPUs. An effective algorithm for multi-window visualization using the implemented library with hardware acceleration is present. The described approach allows you to achieve the visualization speed acceptable for the pilot display of the aircraft.

Keywords: Pilot Display, Embedded Systems, Real-time Operating System, OpenGL Safety Critical, Multi-windowing.

1 Introduction

In [1] requirements were formulated for a real-time operating system (RTOS) designed to work with integrated modular avionics. In particular, the RTOS should comply with the ARINC 653 standard [2]. The software used in aircraft must be certified in accordance to strict rules. The certification requires the correct software development processes in accordance with DO-178C [3], as well as full access to source codes. The real-time operating system JetOS [4] under development, which meets these requirements, is currently being prepared for certification. An important component of the on-board software is the pilot display visualization system which must meet the OpenGL SC (Safety Critical) standard to be used in aviation software.

In [5, 6] a software implementation of the OpenGL SC graphics library running under the JetOS RTOS was presented. Software implementation of OpenGL is easier to certify in accordance with the requirements described above. However, the use of hardware acceleration provides a higher visualization speed. The achieved visualization speed reported in [6] was not satisfactory for typical aviation applications. Moreover satisfactory speed for most of the analyzed applications was achieved using four processor cores only. However the use of four cores is not always acceptable because there are other consumers of computing resources in the on-board system. It also creates certain problems for system certification.

The goal of our research is to find a solution which provides visualization of a pilot display in multiwindows mode with acceptable speed. And in the same time the solution should possible to certificate for usage in civil aircrafts. The OpenGL software solution mentioned above is close to this goal but not fully acceptable due to low speed reason. This is why it was decided to investigate possibility to use the GPU hardware acceleration. Because of certification requirement the MESA package [7] with open sources was the only choice.

MESA (also called Mesa3D and the Mesa 3D Graphics Library) is an open source software implementation of OpenGL, Vulkan, and other graphics API. MESA translates the API specifications into manufacturer specific graphics hardware drivers. It is focused on providing high performance when working with 3D-graphics through the hardware acceleration provided by GPUs. For a particular GPU an appropriate specific driver is required. Some processor manufacturers, such as AMD or Intel, develop drivers for the open-source MESA package for their processors by themselves. Other manufacturers, such as Nvidia or Vivante, completely replace the entire MESA, providing their own proprietary implementation of the graphics API (OpenGL library). For such equipment the MESA development community creates alternative drivers (for example, Nouveau for Nvidia or Etnaviv for Vivante). The implementation of various versions of OpenGL (including OpenGL SC) based on commercial drivers is considered in [8, 9]. However using binary drivers in JetOS RTOS is not possible due to certification requirements.

Therefore an open source driver is needed to use the hardware acceleration of a graphics processor in JetOS RTOS. Unfortunately a simple way – to use existent drivers from MESA – does not work because MESA is elaborated mainly for Linux and Windows. But the JetOS RTOS has a lot of own specifics. So in our case, when using the i.MX6 platform with the Vivante GPU, the only way is to investigate possibility to adapt MESA and Etnaviv driver codes. It should be noted that the system requires two libraries for working with graphics: the first one provides the OpenGL API and the second one directly displays the image built by the OpenGL library on the monitor screen (the frame buffer library). In this paper we consider only the implementation of the OpenGL.

So the implementation of the GPU hardware support for visualization in the JetOS operating system can be divided into two stages:

1. Adaptation of the MESA package to work under control of the JetOS RTOS;
2. Direct implementation of the OpenGL SC standard based on the adapted package.

2 Adaptation of the MESA package to work under control of the JetOS real time OS

The MESA package is currently adapted only for use under operating systems such as Linux, Android and Windows [7]. For these operating systems, the compilation and integration technology for most of the currently used GPUs is available in the MESA package. But the development of applications, drivers and libraries for running under JetOS is significantly different from the technology of software development for Linux or Windows. To adapt the MESA package to work under the JetOS that meets to the ARINC 653 standard, the following main problems must be solved:

1. Only appropriate JetOS special functions can be used to allocate memory.
2. Memory allocation is allowed only at the partition initialization stage.
3. Dynamic memory allocation, freeing memory is prohibited. This means that the corresponding MESA calls must be excluded or rewritten using own memory manager.
4. Any multithreading in accordance with the ARINC 653 specification cannot be used. Therefore the corresponding MESA calls using mutex objects should be excluded or rewritten.
5. The MESA package contains a large amount of code intended for use in various operating systems for working with various graphic processors. All such redundant code must be excluded in accordance with the DO-178C. The size of the source code of the MESA package is ~125 megabytes and contains ~6000 files. Most of them are not needed for OpenGL hardware acceleration of i.MX6 platform under JetOS. But the package has complex structure with internal dependences. So elicitation of redundant code is a challenging task.

We succeeded to solve to some extent all the problems indicated in points 1–5 and to implement the hardware OpenGL (HWGL) library based on the MESA package. All aviation applications from [5, 6] were successfully executed under JetOS using this library. The size of the library was reduced to ~40 megabytes and ~1600 files. Of course the work is not completed yet. Further optimization of the code is necessary in order to remove its unused parts (so-called “dead code” in term of DO-178C). For this purpose the appropriate JetOS tools can be used that capture the use of various functions during application running. Unused functions or code branches should be excluded.

After MESA code adaptation the OpenGL Safety Critical should be implemented to make our library acceptable for avionics software. This work should be carried out separately for standard OpenGL SC versions 1.0.1 and 2.0.1. The OpenGL SC 1.0.1 standard [10] was developed on the basis of the OpenGL 1.3 standard as a subset of it with additional restrictions. They are mainly related to the prohibition of use of double precision variables and certain restrictions on the permissible function parameters. Thus the implementation of the OpenGL SC 1.0.1 standard is not so difficult. Implementation of the library according to the OpenGL SC 2.0.1 standard [10] requires a lot of efforts. The standard is based of the OpenGL ES 2.0 but it contains also func-

tions defined in the later OpenGL ES standards. In addition, the standard does not define a function for shaders compilation. The standard defines only the glProgramBinary function which provides the loading of a precompiled binary object. What exactly compiler should be used for shaders is left, apparently, at the discretion of the library developers. It looks reasonable to move the shader compiler to a separate utility that will be used at the stage of application preparation. This should somewhat simplify the certification of the system as a whole.

3 Visualization speed using hardware acceleration

The hardware OpenGL SC 1.0.1 library implemented on the basis of the MESA package was tested on the same aviation applications that were used for elaboration of the software library in [6]. These are S_PFD (SSJ-100 Primary Flight Display), M_PFD (MC-21 Primary Flight Display), Counters, GlassCockpit, FlightDisplay and SVS (relief visualization). Table 1 shows the visualization speed (in frames per second) for the software OpenGL (SWGL) from [6] and the hardware OpenGL (HWGL) elaborated by us basing on MESA. Column SWGL 1 presents speed for use of 1 core, while SWGL 4 – 4 cores was used.

Table 1. Visualization speed (in frames per second) of the software and hardware OpenGL SC 1.0.1 libraries elaborated by us for the aviation JetOS RTOS

	SWGL 1	SWGL 4	HWGL
S_PFD	5.9	13.8	10.8
M_PFD	6.3	15.6	20.0
Counters	23.9	35.4	60
GlassCockpit	10.5	28.7	29.7
FlightDisplay	9.7	26.4	60
SVS	7.7	20.9	60

It should be noted that the speed of 60 frames per second indicated in the table for some applications is restricted by the display vertical synchronization frequency. The speed of HWGL for these applications is higher. Thus, the use of our HWGL library in the JetOS operating system allows to achieve significant visualization acceleration due to the use of hardware acceleration of the Vivante GPU. The software implementation even using four cores is faster for one specific application only.

4 Multi-window display visualization

An important requirement for the aircraft's on-board system software is multi-window visualization capabilities when various information, generated by numerous independent flight control systems, is simultaneously displayed on one multi-function display. For the software implementation of OpenGL, a compositor was developed in

work [6]. It receives images generated by various applications and displays them in the specified screen windows using the frame buffer library. Two approaches were considered in that work. In the first one all applications and the compositor work in the own partitions, but on one processor core according ARINC 653 standard. In the second one the so-called AMP technology was used when a separate instance of the JetOS operating system was running on each processor core and so each application and compositor works on own core. The acceptable visualization speed for aviation systems was achieved with the second approach only.

Unfortunately both compositor approaches cannot be used for OpenGL with hardware acceleration. A typical aviation platform has only one GPU. We cannot use separate OpenGL instance in each partition or instance of JetOS. So to solve this problem we developed a new approach when OpenGL commands are executed in one special partition of the JetOS operating system.

The application using visualization via the OpenGL library for can be divided into two parts:

1. Preparation of data necessary for OpenGL operation: geometry parameters, various attributes, camera parameters, etc.
2. Direct execution of OpenGL functions with prepared data.

In our implementation these components are run in different partitions of the JetOS operating system. This allows us to separate data preparation and OpenGL execution which has to use the only GPU in case of hardware acceleration. Each application stores the indices of the OpenGL functions needed for visualization together with the corresponding parameters in a special array. It is located in the memory shared with the partition that performs real calls of the OpenGL functions (we will further call it the OpenGL partition). When all the calls necessary for generating the one frame image are prepared then the OpenGL partition is started. Only this partition uses the HWGL library and load GPU. Fig. 1 illustrates the implemented approach.

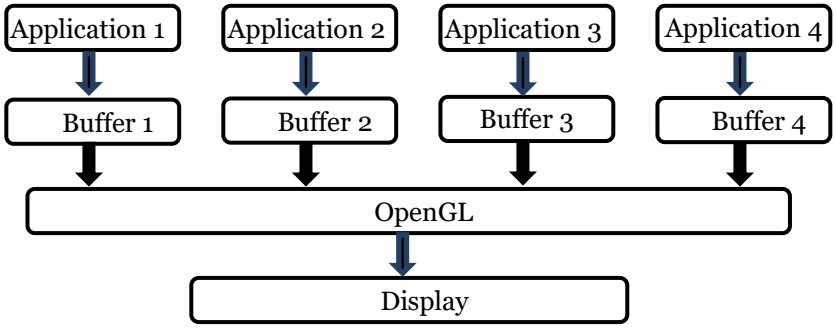

Fig. 1. Scheme of multi-window visualization in case of hardware acceleration.

The code for each application is used almost unchanged. Minimal changes are necessary to provide necessary viewport position on the screen and to synchronize the operation of applications and the OpenGL partition. Encoding of calls to the OpenGL

functions and writing the necessary data to the corresponding buffer is performed by the special OGLOUT library. This library replaces the real calls to OpenGL functions by writing the necessary data and indices of the corresponding OpenGL functions.

The encoding itself is relatively simple. If the function passes a fixed number of parameters, then the index of the called function is written to the current position of the array, followed by the values of the parameters. This is applicable to the most of OpenGL functions, such as glViewport(), glClearColor(), glClearDepthf(), glClear-Stencil(), etc. In more complex cases when number of parameters is not defined at the time of the call are passed (for example, glVertexPointer()) the encoding technology is somewhat complicated. During the call of such functions, instead of the pointer, the index of the reserved position in the output array is written. Then from this position values are written by this pointer in functions such as glDrawArrays() or glDrawEle-ments().

Decoding the information recorded by applications is performed in the OpenGL partition using the OGLIN library sequentially for each array generated by the appli-cations. The OGLIN library consists of one function process_ogl_input_array(). It sequentially reads data from the array written by the application, then according to the index of the OpenGL function it goes to the piece of code written for this function and extracts the data transferred from subsequent elements of the array including pointer values. After that it calls the specified OpenGL function with extracted parameters.

Disadvantage of this approach is that it is hard to implement the OpenGL functions that return parameter values (such as, for example, glGenLists(), glGetError(), glGetString(), glIsEnabled(), glGetPointerv(), glGetTexParameteriv()). However an analysis of the currently used practical applications in onboard equipment has shown that these functions are not used in them. Calls to these functions are also absent in the OGLX library [11] which is currently used in the development of most onboard applications with output to the display screen. Since according to DO-178C there should not be a non-executable code (Dead code) in the on-board equipment software we do not need to solve this problem

Synchronization of the applications and the OpenGL partition is done using special objects called events [6]. Interaction between applications and the OpenGL partition is implemented in the way close to the synchronization between client and servers described in [6] and we omit these details here to simplicity.

5 The multi-windowing algorithm performance

The developed algorithm was tested using multi-window visualization of two pairs of practical applications:

1. PFD (Primary Flight Display) and DOORS (state of aircraft doors), demonstrated on Fig. 2;
2. PFD (Primary Flight Display) and ND (Navigation Display), demonstrated on Fig. 3.

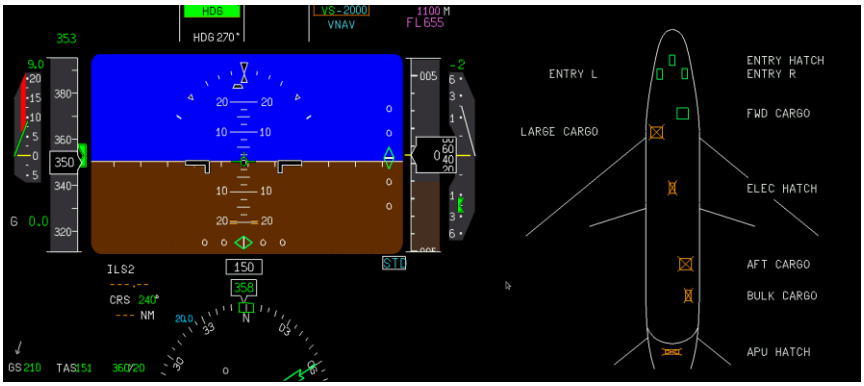

Fig. 2. Multi-window visualization of two applications – PFD and DOORS.

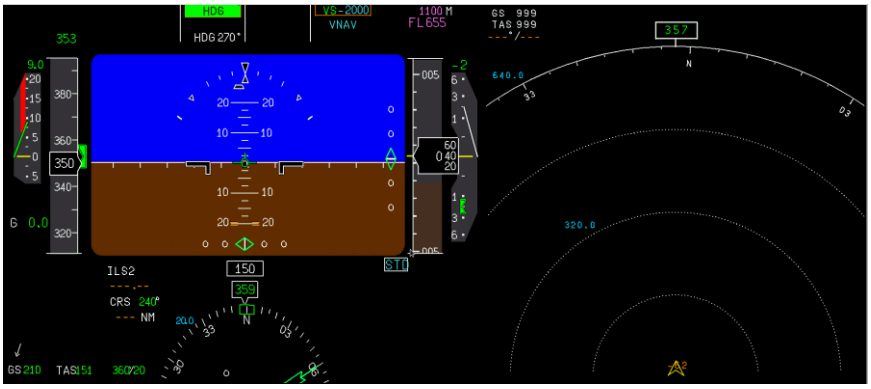

Fig. 3. Multi-window visualization of two applications – PFD and ND.

Table 2 shows multi-window visualization speed (in frames per second) for the software OpenGL (SWGL) and the hardware OpenGL (HWGL). Visualization via the SWGL library has the possibility to run each application with own instance of OpenGL and compositor on own processor core. Visualization via the HWGL uses the algorithm described in previous chapter. Column SWGL 1 presents speed for use of 1 core, while SWGL 3 – 3 cores were used.

Table 2. Multi-window visualization speed (in frames per second) for the software and hardware OpenGL SC libraries.

	SWGL 1	SWGL 3	HWGL (our solution)
PFD + DOORS	4.5	6.2	14.9
PFD + ND	4.6	6.2	16.5

It should be noted that the visualization approach presented in this paper (HWGL + new multi-window algorithm) finally provides the speed which 2-3 times higher than visualization speed in case of software libraries using. Taking into account that the pilot display intended for visualization of information about flight and state of aircraft systems (but not for a movie demonstration) the speed in 15-20 frames per second can be considered as acceptable.

6 Conclusion

Studies have shown the possibility to use the GPU hardware acceleration in avionics. All study components – the JetOS real-time operating system, the OpenGL SC standard, the Vivante GPU – impose own limitations and particularities. Additionally, we always need to think about certification of on-board software according to the DO-178C avionics standard. All these complicated the project much. But finally we succeeded to elaborate the approach and the algorithms which, on the one hand, provide acceptable visualization speed for the aircraft pilot display and, on the other hand, can be certified to be used in a real aviation operating system.

The technology of the hardware OpenGL SC library creation basing on the adaptation of the open-source MESA package can be applicable not only for the Vivante GPU but for other graphics hardware as well.

References

1. Fedosov, E.A., Koverninskii, I.V., Kan, A.V., Solodelov, Yu.A.: Application of real-time operating systems in integrated modular avionics, https://osday.ru/2015/solodelov.html. Last accessed 5 Jul 2020
2. Ananda, C.M., Sabitha, N., Mainak, G.H.: ARINC 653 API and its application – An insight into Avionics System Case Study. Defence Science Journal 63(2), 223-229 (2013)
3. DO-178C Software Considerations in Airborne Systems and Equipment Certification, https://en.wikipedia.org/wiki/DO-178C. Last accessed 5 Jul 2020
4. Mallachiev, K.M., Pakulin, N.V., Khoroshilov, A.V.: Design and architecture of real-time operating system. Proceedings of the Institute for System Programming 28(2), 181-192 (2016)
5. Barladian, B.Kh., Voloboy, A.G., Galaktionov, V.A., Knyaz, V.V., Koverninskii, I.V., Solodelov, Yu.A., Frolov, V.A., Shapiro, L.Z.: Efficient Implementation of OpenGL SC for Avionics Embedded Systems. Programming and Computer Software 44(4), 207–212 (2018)
6. Barladian, B.Kh., Shapiro, L.Z., Mallachiev, K.A., Khoroshilov, A.V., Solodelov, Yu.A., Voloboy, A.G., Galaktionov, V.A., Koverninskii, I.V.: Visualization Component for the Aircraft Real-Time Operating System JetOS. Programming and Computer Software 46(3), 167–175 (2020)
7. The Mesa 3D Graphics Library, https://www.mesa3d.org. Last accessed 5 Jul 2020
8. Baek, N., Lee, H.: OpenGL ES 1.1 Implementation Based on OpenGL. Multimedia Tools and Applications 57(3), 669–685 (2012)
9. Lee, H., Baek, N.: OpenGL SC Emulation Based on OpenGL and OpenGL ES. In: Cozzi, P., Riccio, Ch. (eds.) OpenGL Insights, pp. 121-131 (2012)

10. OpenGL Safity Critical specifications, https://www.khronos.org/registry/OpenGL/specs/sc. Last accessed 5 Jul 2020
11. Ansys SCADE Display Capabilities. https://www.ansys.com/products/embedded-software/ansys-scade-display/scade-display-capabilities. Last accessed 5 Jul 2020