Разработка системы визуализации для гражданских воздушных судов

Б.Х. Барладян [1], Н.Б. Дерябин [1], А.Г. Волобой [1], В.А. Галактионов [1], Л.З. Шапиро [1], И.В. Валиев [1], Ю.А. Солоделов [2]

[1] Институт прикладной математики им. М.В. Келдыша РАН, Москва, Миусская пл., д. 4, 125047, Россия
[2] Государственный научно-исследовательский институт авиационных систем, Москва, ул. Викторенко, д.5., 7125319, Россия

Аннотация

Приборные панели современных самолетов создаются по концепции «стеклянной кабины». Эта новая идеология интерфейса позволяет улучшить восприятие важной полетной информации за счет объединения ее в один многофункциональный дисплей, обеспечивающий целостное, легко воспринимаемое изображение полетной информации. В работе рассматриваются проблемы, возникающие при разработке сертифицируемой системы визуализации дисплея пилота, предназначенной для работы на гражданских воздушных судах под управлением российской операционной системы реального времени JetOS. Показано, что нами были успешно решены задачи многооконной визуализации множества авиационных приложений с приемлемой скоростью на перспективной авиационной вычислительной платформе. В статье перечислено несколько алгоритмических решений, позволяющих добиться требуемой скорости визуализации, а также намечены пути дальнейших работ.

Ключевые слова

Дисплей кабины пилота, бортовая система визуализации, OpenGL SC, ОСРВ, авионика, драйвер GPU.

Development of a Visualization System for Civil Aircraft

B.Kh. Barladian [1], N.B. Deryabin [1], A.G. Voloboy [1], V.A. Galaktionov [1], L.Z. Shapiro [1], I.V. Valiev [1], Y.A. Solodelov [2]

[1] The Keldysh Institute of Applied Mathematics Russian Academy of SciencesMoscow, Miusskaya pl., 4, 125047, Russia
[2] State Research Institute of Aviation Systems Moscow, Viktorenko str., 5., 7125319, Russia

Abstract

The instrument panels of modern aircraft are created according to the concept of a "glass cockpit". This new interface concept enhances the perception of critical flight information by integrating it into one multi-functional display that provides a cohesive, easy-to-read view of flight information. The paper deals with the problems that arise in the development of a certified visualization system for the pilot's display, designed to work on civil aircraft under the Russian real-time operating system JetOS. It is shown that we have successfully solved the problems of multi-window visualization of many aviation applications with an acceptable speed on a promising aviation computing platform. The article lists several algorithmic solutions that allow achieving the required rendering speed, as well as outlines ways for further work.

ГрафиКон 2023: 33-я Международная конференция по компьютерной графике и машинному зрению, 19-21 сентября 2023 г., Институт проблем управления им. В.А. Трапезникова Российской академии наук, г. Москва, Россия
EMAIL: bbarladian@mail.com (Б.Х. Барладян); dek@keldysh.ru (Н.Б. Дерябин); voloboy@gin.keldysh.ru (А.Г. Волобой); vlgal@gin.keldysh.ru (В.А. Галактионов); pls@gin.keldysh.ru (Л.З. Шапиро); piv@gin.keldysh.ru (И.В. Валиев); yasolodelov@2100.gosniias.ru (Ю.А. Солоделов)
ORCID: 0000-0002-2391-2067 (Б.Х. Барладян); 0000-0003-1248-6047 (Н.Б. Дерябин); 0000-0003-1252-8294 (А.Г. Волобой); 0000-0001-6460-7539 (В.А. Галактионов); 0000-0002-6350-851X (Л.З. Шапиро); 0000-0003-2937-8480 (И.В. Валиев); 0000-0001-5891-7645 (Ю.А. Солоделов)

Keywords
Cockpit display, onboard visualization system, OpenGL SC, RTOS, avionics, GPU driver.

1. Введение

Современные комплексы бортового оборудования (КБО) проектируются в соответствии с идеологией, известной как «Интегрированная модульная авионика» (ИМА) [1-4]. Эта идеология используется в лайнерах Airbus A320, Boeing 787, Sukhoi Superjet 100 и МС-21. Принципиальным вопросом при разработке бортовой авионики является то, что она относится к системам, критическим с точки зрения безопасности (safety critical). Эта специфика не позволяет применять готовые, известные решения для той или другой функциональности. В силу этой специфики КБО требуют специальной сертификации для своего использования. Сертификация, в частности, определяет организацию процесса разработки, который должен проводиться в соответствии с требованиями стандарта DO-178C [5]. Ключевой особенностью ИМА является возможность исполнения нескольких функциональных приложений, реализующих программную часть той или иной самолетной системы, на одном вычислителе. Необходимым условием при этом является разделение приложений по времени исполнения и доступным ресурсам, т.е. ограничение вытесняющей многозадачности и контроль доступа к памяти для нескольких приложений. Такой режим работы приложений обеспечивается операционной системой реального времени (ОСРВ), что делает ОСРВ неотъемлемой и важнейшей частью любого современного вычислительного модуля и комплекса бортового оборудования в целом. Программный интерфейс ОСРВ, предназначенный для использования на борту воздушного судна, должен соответствовать стандарту ARINC 653 [6].

На протяжении длительного времени как в отечественной программе ИМА, так и на разрабатываемых воздушных судах применялись зарубежные ОСРВ (такие как VxWorks 653 или Thales MACS2). Однако в последнее время использование зарубежных программных продуктов сталкивается с серьезными проблемами, а в настоящее время стало невозможным. В связи с этим, была разработана отечественная бортовая операционная система реального времени (ОСРВ) JetOS [4]. Так как бортовые системы для различных самолетов строятся на разных вычислительных платформах (процессорах и соответствующих GPU), то операционная система должна быть адаптирована и сертифицирована под каждую платформу.

При разработке кабин современных самолетов существует тенденция использовать большие дисплеи и визуализировать на них одновременно различную информацию о полетной навигации и состоянии оборудования самолета. На рисунке 1 показана приборная панель МС-21, иллюстрирующая данную тенденцию.

Рисунок 1 – Приборная панель самолета МС-21

Таким образом, разработка системы визуализации кабины пилота воздушного судна является неотъемлемой частью разработки бортового программного обеспечения. Процесс разработки должен соответствовать авиационным стандартам DO-178C и ARINC 653. При этом задача осложняется невысокой производительностью процессоров (на борт обычно ставятся

энергосберегающие вычислители, имеющие историю надежного использования) и высокими требованиями к надежности и скорости визуализации.

Статья организована следующим образом. Раздел 2 представляет архитектуру бортовой системы визуализации и особенности двух стандартов OpenGL SC, предписанных для использования в авиации. Разделы 3-5 рассказывают о создании системы визуализации дисплея пилота, в процессе решения возникающих проблемы было разработано множество алгоритмов, детали реализации которых были опубликованы также и в отдельных статьях. Раздел 6 предлагает новый алгоритм, а также направления дальнейших исследований. Раздел 7 содержит заключение.

2. Бортовая система визуализации

Одним из важных требований к системному бортовому ПО является использование специальной версии графической библиотеки OpenGL SC (Safety Critical), предназначенной для использования в системах, критических с точки зрения безопасности. В настоящее время библиотека OpenGL SC представлена двумя версиями стандарта - 1.0.1 и 2.0. Стандарт OpenGL SC 1.0.1 является подмножеством стандарта OpenGL 1.3. Стандарт OpenGL SC 2.0 основан на стандарте OpenGL ES 2.0, но не является его подмножеством. Он разработан для критически важного для безопасности графического оборудования, работающего на встроенных устройствах. В нем удалены аспекты OpenGL ES 2.0, которые не соответствуют детерминированной безопасности программных приложений. В частности, удален встроенный компилятор шейдеров. Предполагается, что вершинный (vertex) и фрагментный (fragment) шейдеры автономно компилируются и связываются с двоичным объектом программы, который генерирует исполняемый код из скомпилированных шейдеров. При этом способ создания (интерфейс) бинарного объекта программы в стандарте не оговорен. Стандартизован только интерфейс функции ProgramBinary(), которая позволяет загрузить предварительно скомпилированный двоичный объект.

Стандарты OpenGL SC 2.0 и OpenGL SC 1.0.1 существенно отличаются. Некоторые функции версии 1.0.1 удалены в версии 2.0 и могут быть заменены шейдерными программами. Другими словами, в OpenGL SC 2.0 упор делается на программируемый конвейер трехмерной графики, а в OpenGL SC 1.0.1 – на фиксированную функциональность.

В общем виде систему визуализации в кабине самолета можно представить в виде схемы на рисунке 2.

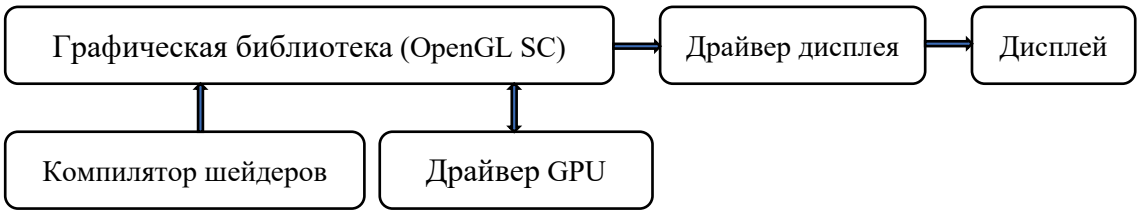

Рисунок 2 – Схема визуализации в кабине самолета

Графическая библиотека для программы, написанной по одному из стандартов OpenGL SC, генерирует изображения для каждого кадра и выводит их на экран, используя драйвер дисплея. При разработке программной реализации графической библиотеки только драйвер дисплея учитывает специфику конкретной вычислительной платформы. Программная версия графической библиотеки разрабатывается на языке C. Ее перенос на другую платформу не должен вызывать существенных проблем.

Для использования аппаратной поддержки графического процессора для каждой вычислительной платформы необходимо разрабатывать свой драйвер GPU, который обеспечит доступ графической библиотеки к необходимым ресурсам графического ускорителя.

Для работы с приложениями, написанными по стандарту OpenGL SC 2.0, необходим внешний компилятор шейдеров, поскольку стандарт OpenGL SC 2.0 предполагает загрузку шейдеров в виде уже скомпилированного двоичного объекта.

3. Программная реализация библиотеки OpenGL SC

Как уже было сказано, для программной реализации графической библиотеки только драйвер дисплея должен учитывать специфику конкретной вычислительной платформы. Сама программная реализация OpenGL является платформо-независимой. В силу этих причин программная реализация OpenGL была первым шагом в наших работах.

Разумеется, для программной реализации OpenGL, даже существенно оптимизированной, нельзя достичь производительности библиотеки, использующей аппаратную поддержку. Поэтому для повышения производительности мы распараллелили работу библиотеки на многоядерных процессорах. Для этого было использовано специальное расширение стандарта ARINC 653, реализованное в JetOS. Расширение называется Asymmetric Multi-Processing. Это позволило ускорить визуализацию в 2.5 – 3 раза для типичных авиационных приложений, используя многоядерные процессоры, не отклоняясь от разрешенных стандартов.

Реализация алгоритмически оптимизированной программной версии OpenGL SC 1.0.1 подробно описана в работах [7, 8]. Подготовка сертификационного пакета для программной реализации библиотеки не вызывает существенных проблем, поскольку весь код доступен и написан на языке C. Но работа программной реализации OpenGL заканчивается генерацией изображения кадра в памяти процессора, а для вывода на экран дисплея еще необходим драйвер дисплея. Без этого библиотека просто не может быть использована в реальном КБО. И, если сама программная реализация OpenGL является платформо-независимой, то драйвер дисплея зависит, естественно, от используемого оборудования. Для платформы i.MX6 нам удалось реализовать такой драйвер с минимальным использованием кода, специфичного для GPU.

При использовании алгоритмически оптимизированной программной версии OpenGL SC 1.0.1 удалось получить приемлемые скорости визуализации для таких типовых приложений, как Primary flight display (PFD) – рисунки 3 (для Sukhoi Superjet 100) и 4 (для МС–21), индикатор состояния дверей Doors – рисунок 5 и Navigation display (ND) – рисунок 6. Скорость программной визуализации без аппаратной поддержки, полученная для этих приложений при использовании одного и четырех ядер процессора на платформе i.MX6, приведена в таблице 1 (в кадрах в секунду).

Рисунок 3 – PFD SS100

Рисунок 4 – PFD МС–21

Таблица 1 – Скорость визуализации (кадры в секунду) программной реализации Open GL SC

Приложение	1 ядро процессора	4 ядра процессора
PFD Sukhoi Superjet	5.9	13.8
PFD МС–21	5.3	15.6
Doors	12.0	34.7
Navigation display	22.5	40.3

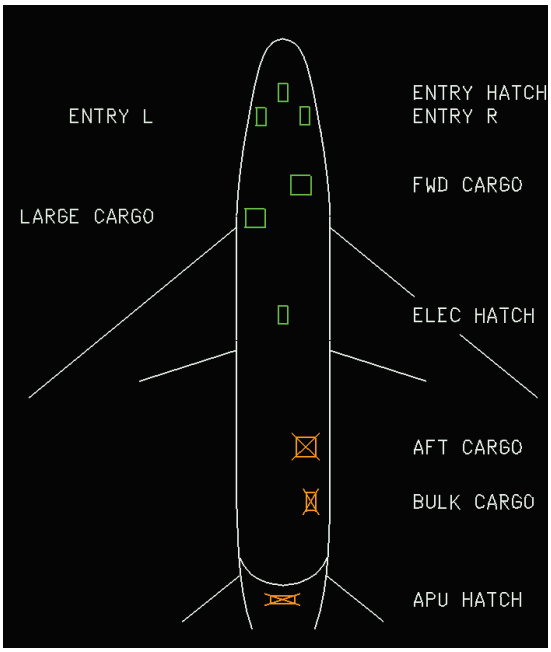

Рисунок 5 – Doors

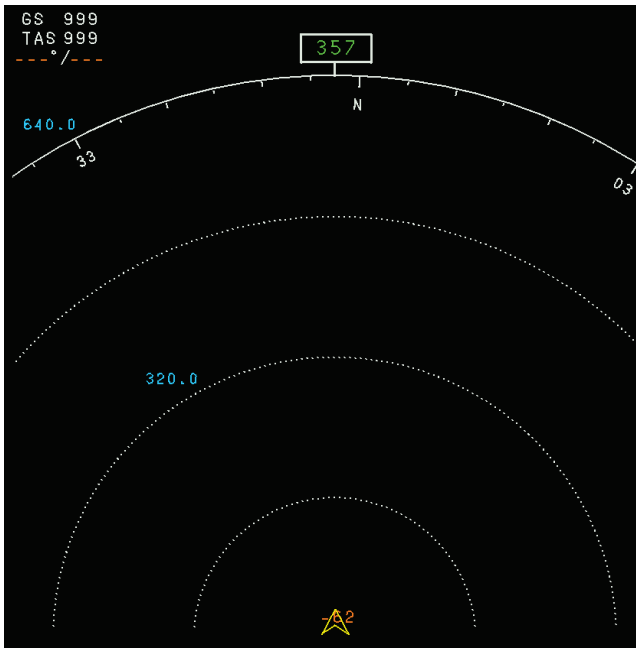

Рисунок 6 – Navigation display

Использование многоядерности процессора для программной реализации графической библиотеки позволило также относительно эффективно реализовать многооконную визуализацию. Реализация использует Asymmetric Multi-Processing – специальное расширение стандарта ARINC 653, реализованное в JetOS [8]. Для обоих примеров многооконной визуализации, представленных на рисунках 7 и 8, была получена скорость визуализации 4.5 – 6.2 кадров в секунду для чисто программной реализации OpenGL SC, не использующей аппаратное ускорение.

Рисунок 7 – Многооконная визуализация ND и PFD МС–21

В итоге, разработанная нами программная реализация библиотеки OpenGL SC 1.0.1 при использовании многоядерности процессора оказалась достаточно эффективной для многих практических авиационных приложений. Однако, в ряде случаев в КБО может возникать необходимость использования ядер процессора не только для визуализации. В связи с этим необходимо было реализовать аппаратную поддержку для библиотеки OpenGL SC, что и явилось следующим шагом в создании системы визуализации бортового ПО.

Рисунок 8 – Многооконная визуализация Doors и PFD МС–21

4. Реализация библиотеки OpenGL SC с аппаратной поддержкой

Необходимость разработки сертификационного пакета для библиотеки в соответствии с требованиями КТ-178С требует открытого доступа к всем ее кодам. Однако в большинстве случаев драйверы, предоставляемые производителями GPU, не имеют открытого программного кода и, соответственно, не могут быть сертифицированы для использования в авиации. Также драйвер необходимо адаптировать для работы под управлением ОСРВ JetOS. Реализация различных версий OpenGL SC на базе коммерческих драйверов рассматривается в работах [9-11], однако сертификация этих двоичных драйверов невозможна.

Для решения этой проблемы наша разработка базировалась на базе пакета Mesa [12]. Mesa представляет собой программную реализацию OpenGL, Vulkan и других спецификаций графического API с открытым исходным кодом. Mesa переводит эти спецификации в драйверы конкретного графического оборудования. Некоторые производители процессоров, такие как, например, AMD или Intel, сами разрабатывают драйверы с открытым кодом для производимых ими процессоров. Другие производители, такие как Nvidia или Vivante, полностью заменяют Mesa, обеспечивая собственную реализацию библиотеки OpenGL в двоичном виде. Для такого случая сообщество разработчиков Mesa создает альтернативные открытые драйверы, такие как Nouveau для Nvidia или Etnaviv для Vivante. В нашем случае, при использовании платформы i.MX6 с графическим процессором Vivante, единственной возможностью разработки сертифицируемой библиотеки OpenGL SC с аппаратной поддержкой является использование в качестве основы кодов драйвера Etnaviv.

Детально процесс реализации графической библиотеки, поддерживающей стандарты OpenGL SC 1.0.1 и OpenGL SC 2.0.1 с аппаратной поддержкой на базе пакета Mesa, описан в работах [13, 14]. Отметим здесь, что реализация многооконного режима потребовала принципиально другого алгоритма по сравнению с программной реализацией, поскольку при использовании аппаратной поддержки может быть использован только один экземпляр OpenGL. Аппаратная поддержка GPU позволила в большинстве случаев увеличить скорость визуализации как для стандартных приложений (таблица 2), так и для многооконной визуализации (таблица 3)..

Таблица 2 – Скорость визуализации (кадры в секунду) программной реализации Open GL SC и при использовании аппаратной поддержки

Приложение	1 ядро процессора	4 ядра процессора	поддержка GPU
PFD Sukhoi Superjet	5.9	13.8	10.8
PFD МС–21	5.3	15.6	20.0
Doors	12.0	34.7	60
Navigation display	22.5	40.3	60

Таблица 3 – Скорость многооконной визуализации (кадры в секунду) программной реализации Open GL SC и при использовании аппаратной поддержки

Приложение	1 ядро процессора	3 ядра процессора	поддержка GPU
Navigation display + PFD (рис. 7)	4.5	6.2	14.9
Doors + PFD (рис. 8)	4.6	6.2	16.5

При распараллеливании системы визуализации в многооконном режиме использовалось только 3 ядра: 2 ядра для двух экземпляров OpenGL SC для каждой составляющей картинки и один для компоновщика всего изображения [8].

Графическая библиотека OpenGL SC стандарта 2.0 в настоящее время является наиболее используемой в авиационных приложениях. Ее программная реализация не имеет практического смысла, поскольку шейдеры в настоящее время эффективно поддерживаются в современных GPU, а их программная реализация будет не эффективна. Для работы с приложениями, написанными под этот стандарт, необходима реализация внешнего компилятора шейдеров, поскольку стандарт предполагает загрузку шейдеров в виде уже скомпилированного двоичного объекта. Здесь еще следует отметить проблему сертификации графической библиотеки OpenGL с аппаратной поддержкой. Реализация графической библиотеки с аппаратной поддержкой на базе пакета Mesa содержит большое количество кода на языке C++, который невозможно сертифицировать по авиационным стандартам. Этот код, в основном, используется в компиляторе шейдеров. Поэтому его трудно исключить для стандарта OpenGL SC 1.0.1, но относительно просто для стандарта OpenGL SC 2.0, поскольку компилятор шейдеров вынесен во внешнюю программу, которая используется только на этапе подготовки приложений.

5. Оптимизация использования сервера ARINC 661

Система отображения в кабине экипажа (Cockpit display system, на английском - CDS) предоставляет видимую и звуковую информацию о воздушном судне и окружающей среде и получает команды управления от экипажа. Таким образом экипаж управляет самолетом и взаимодействует с бортовым радиоэлектронным оборудованием. Интерфейсы между CDS и пользовательскими приложениями (User application – UA) должны удовлетворять авиационному стандарту ARINC 661 [15]. CDS предоставляет графические и интерактивные услуги для пользовательских приложений (UA) в кабине. Приложение отправляет графическую информацию в CDS. Экипаж контролирует поведение UA с помощью команд, отправляемых от CDS к UA. Обмен данными между UA и CDS осуществляется через сеть в соответствии со стандартом ARINC 661.

Приложения, представляющие карту движения по аэродрому с взлетно-посадочной полосой и рулежными дорожками (рисунок 9), создавались сервером ARINC 661, разработанным компанией Ansys Inc. [16].

Код приложения предполагает использование библиотеки OpenGL, включая ее трехмерные возможности. Реализация этого сервера оказалась не эффективной с точки зрения использования OpenGL SC с аппаратным ускорением. Каждый отрезок линии и каждый треугольник в нем рисовался отдельным набором команд библиотеки OpenGL, в то время как эффективный протокол работы с GPU подразумевает выполнение как можно большого набора команд за одно обращение. В работе [17] мы оптимизировали работу приложения путем объединения этих команд в группы рисования однородных примитивов – отрезков или треугольников. В результате этой оптимизации визуализация рулежных дорожек и окружающей геометрии на рисунке 9 была ускорена с 1.7 до 15.2 кадров в секунду.

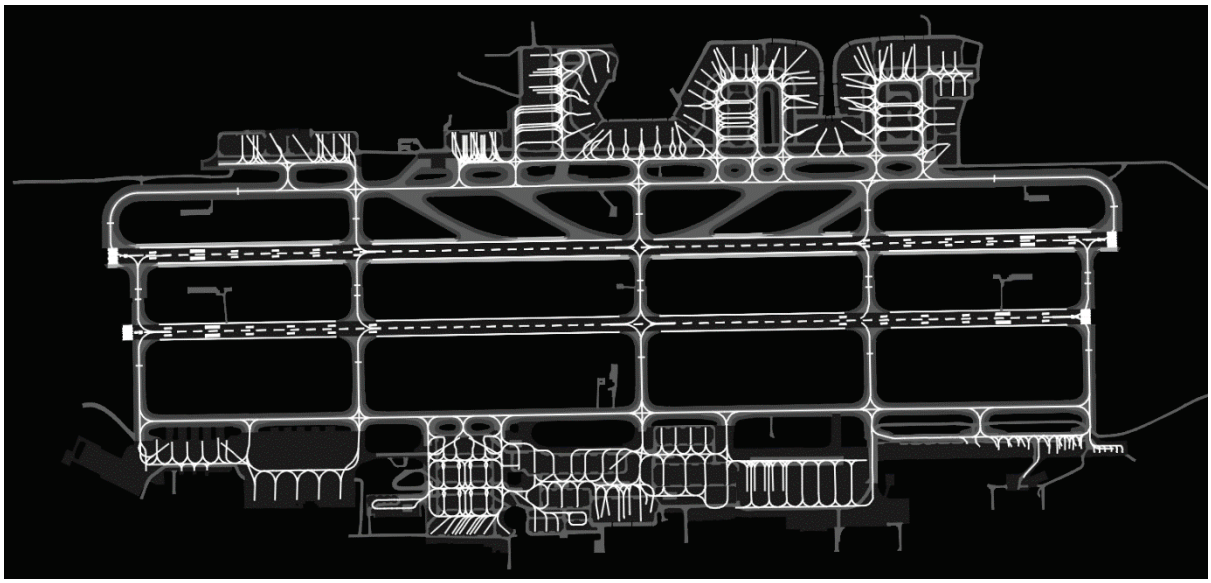

Рисунок 9 – Приложение «карта движения по аэродрому»

6. Направление дальнейших работ

Несмотря на достигнутую приемлемую скорость визуализации известных приложений (порядка 15 кадров в секунду и выше) крайне желательна дальнейшая оптимизация использования графической библиотеки при работе в операционной системе реального времени. Это становится необходимым, если более сложные приложения используются в КБО или характеристики вычислительной платформы оказываются слабее, чем у использованных в исследовании процессоров.

В ОСРВ существуют достаточно жесткие требования безопасности, которые не позволяют полностью использовать все ресурсы процессора только для графической компоненты. Для обеспечения реального времени, система должна иметь возможность переключать достаточно часто использование процессора для выполнения разных задач. В соответствии со стандартом ARINC 653 каждое приложение выполняется в отдельном разделе (partition) ОСРВ. Диспетчеризация работы разделов является строго детерминированной по времени. Алгоритм диспетчеризации разделов определяется заранее в зависимости от конфигурации разделов в модуле, общих требований к ресурсам, наличия ресурсов и требований отдельных разделов.

Для того, чтобы обеспечить заданное время реакции системы необходимо, чтобы активация соответствующего раздела происходила не реже чем через заданный интервал. Как правило время реакции системы не должно превышать десятков, максимум сотни милисекунд. Кроме графического приложения, которое отображает поступающую информацию, в системе должно работать как минимум еще одно приложение, которое будет обрабатывать поступающую информацию и передавать ее графичскому приложению. Таким образом длительность окна раздела, в котором выполняется графическое приложение, не может превышать нескольких десятков милисекунд (желательно 10-20 милисекунд). Следует отметить, что переключение разделов довольно затратная процедура. Требования по безопасности регламентируют, что при каждом переключении должны быть очищены кэши и регистры, сохранено их содержимое и затем восстановлено при продолжении работы данного раздела. Таким образом, при работе дополнительных разделов скорость визуализации дополнительно падает на 30-50%.

Для решения этой проблемы предполагается выделить работу библиотеки OpenGL SC в отдельный раздел ОСРВ и выполнять ее на отдельном ядре процессора с использованием GPU, с учетом всех ограничений, налагаемых стандартом. Это будет дальнейшее развитие подходов, реализованных в работах [14, 17]. В работе [14] драйвер OpenGL, использующий GPU, работал в отдельном разделе и принимал команды из других разделов, но все разделы для различных окон работали на одном ядре процессора. В работе [17] драйвер OpenGL работал в отдельном разделе на отдельном ядре, но при этом использовалось «сжатие» команд, специфичное для

сервера ARINC 661, которое будет не эффективно в общем случае. Теперь же предполагается разработка драйвера OpenGL общего вида, который будет работать на отдельном ядре процессора и обрабатывать команды OpenGL, поступающие из графических приложений, работающих на других ядрах процессора. Этот подход должен повысить скорость визуализации поскольку происходит распараллеливание работы графического приложения. Часть работы (подготовка данных и команд) происходит на одном ядре процессора, а другая часть (драйвер OpenGL) – на другом ядре процессора и на GPU. Пока драйвер визуализирует данный кадр на GPU, раздел на другом ядре подготавливает данные и команды для следующего кадра.

Первые тесты показали увеличение производительности порядка 10-15%, но этот вопрос требует дальнейшего исследования, поскольку есть дополнительные издержки, связанные с синхронизацией и передачей данных между ядрами. Мы надеемся их минимизировать.

7. Заключение

Создание системы визуализации пилотных дисплеев для гражданских воздушных судов является сложной и многофакторной задачей. Главным требованием, как и при создании любого гражданского бортового ПО, остается безопасность полетов. Это накладывает принципиальные ограничения, значительно усложняющие создание графической системы. Первое ограничение – это надежная, энергосберегающая вычислительная платформа. Например, на борт воздушного судна нельзя поставить известные и достаточно мощные графические процессоры Nvidia или AMD из-за их высоких энергетических затрат, т.к. проектируется, что графическая система должна работать и в нештатной ситуации (например, отключение двигателей). Надежность процессоров проверяется со временем, поэтому на борт ставятся не самые новые процессоры, производительность их невысокая по современным меркам. Вторым фактором разработки системы визуализации является ее использование под операционной системой реального времени, которая во многом отличается от общепринятых ОС Linux или Windows. Сложность создания графической системы также связана с жесткими стандартами гражданской авиации. Они определяют требования к процессу создания бортового программного обеспечения. Поэтому, например, нельзя просто взять готовые существующие реализации соответствующих библиотек и драйверов. При этом графическая система дисплея пилота должна гарантировать приемлемую, интерактивную скорость визуализации авиационных приложений.

Нами были успешно решены задачи многооконной визуализации множества авиационных приложений с приемлемой скоростью на перспективной авиационной вычислительной платформе. Однако эта работа не имеет четко определенной «финальной точки». На разные серии и модели самолетов ставят разные вычислительные платформы, и некоторые из решений, успешно работающих для одной платформы, могут не иметь эффекта для другой. Также, авиационные приложения становятся все более сложными, чтобы предоставлять пилотам дополнительную информацию, позволяющую повысить безопасность управления самолетом. А это означает, что для новой платформы или нового приложения может понадобиться разработка новых методов и алгоритмов. В то же время выпуск конкретной модели самолета можно рассматривать как значительный этап этой работы.

8. Список источников

[1] Федосов Е.А. Проект создания нового поколения интегрированной модульной авионики с открытой архитектурой // Полет. 2008. № 8. С. 15-22.

[2] Применение операционных систем реального времени в интегрированной модульной авионике / Федосов Е.А., Ковернинский И.В., Кан А.В., Солоделов Ю.А. // OS DAY. 2015. URL: https://osday.ru/2015/solodelov.html (дата обращения 01.07.2023).

[3] Федосов Е.А., Косьянчук В.В., Сельвесюк Н.И. Интегрированная модульная авионика // Радиоэлектронные технологии. 2015. №1. С. 66-71.

[4] Солоделов Ю.А., Горелиц Н.К. Сертифицируемая бортовая операционная система реального времени JetOS для российских проектов воздушных судов // Труды ИСП РАН. 2017. Том 29 № 3. С. 171-178. DOI: 10.15514/ISPRAS-2017-29(3)-10.

[5] DO-178C Software Considerations in Airborne Systems and Equipment Certification // 2011. URL: https://www.rtca.org/products/do-178c-electronic (дата обращения 01.02.2022).
[6] Avionics application software standard interface (ARINC 653) // 2015. SAE-ITC.
[7] Barladian B.K., Voloboy A.G., Galaktionov V.A. et al. Efficient Implementation of OpenGL SC for Avionics Embedded Systems // Programming and Computer Software, 2018. 44. Pp. 207–212. DOI: 10.1134/S0361768818040059.
[8] Система визуализации для авиационной ОС реального времени JetOS / Барладян Б.Х., Шапиро Л.З., Маллачиев К.А., Хорошилов А.В., Солоделов Ю.А., Волобой А.Г., Галактионов В.А., Ковернинский И.В. // Труды Института системного программирования РАН. 2020. Том 32 № 1. С. 57-70. DOI: 10.15514/ISPRAS-2020-32(1)-3.
[9] Baek N., Lee H. “OpenGL ES 1.1 Implementation Based on OpenGL // Multimedia Tools and Applications. 2012. V. 57 № 3. Pp. 669–685.
[10] Lee H., Baek N. OpenGL SC Emulation Based on on OpenGL and OpenGL ES // OpenGL Insights by Patrick Cozzi and Christophe Riccio. 2012. Pp. 121-131.
[11] Baek N., Kim K.J. Design and implementation of OpenGL SC 2.0 rendering pipeline // Cluster Computing. 2019. 22. Pp. 931–936. DOI: 10.1007/s10586-017-1111-1.
[12] The Mesa 3D Graphics Library // URL: http://www.mesa3d.org (дата обращения 01.07.2023).
[13] High speed visualization in the JetOS aviation operating system using hardware acceleration / Barladian B.Kh., Deryabin N.B., Voloboy A.G., Galaktionov V.A., Shapiro L.Z. // CEUR Workshop Proceedings. 2020. V. 2744. pp. 107:1-107:9. DOI: 10.51130/graphicon-2020-2-4-3
[14] Multiwindow Rendering on a Cockpit Display Using Hardware Acceleration / Barladian B.K., Deryabin N.B.,Shapiro L.Z., Solodelov Yu.A., Voloboy A.G. and Galaktionov V.A. // Programming and Computer Software. 2021. V. 47 № 6. pp. 457–465. DOI: 10.1134/S0361768821060025.
[15] ARINC 661P1-8 Cockpit Display System Interfaces to User Systems, Part 1, Avionics Interfaces, Basic Symbology, and Behavior // ARINC. 2020, URL: https://www.aviation-ia.com/products/661p1-8-cockpit-display-system-interfaces-user-systems-part-1-avionics-interfaces-basic (дата обращения 01.07.2023).
[16] Ansys SCADE Solution for ARINC 661 Compliant Systems // URL: https://www.ansys.com/products/embedded-software/solutions-for-arinc-661 (дата обращения 01.07.2023).
[17] Efficient Rendering for the Cockpit Display System Designed in Compliance with the ARINC 661 Standard / Barladian B.K., Shapiro L.Z., Deryabin N.B.,Solodelov Yu.A., Voloboy A.G. Galaktionov V.A. // Programming and Computer Software. 2022. V. 48 № 3, pp.147-154. DOI: 10.1134/S0361768822030021.