УДК 378:004
DOI: 10.25686/978-5-8158-2474-4-2025-298-304

Многослойная визуализация дисплея пилота гражданского воздушного судна

Б. Х. Барладян, Н. Б. Дерябин, А. Г. Волобой, Л. З. Шапиро, Е. Ю. Денисов, В. А. Галактионов
Институт прикладной математики им. М.В. Келдыша РАН, Москва, Россия

Аннотация. Создание системы визуализации дисплея пилота является непростой задачей. С одной стороны, она должна работать под операционной системой реального времени со множеством присущих ей ограничений. С другой стороны, она должна обеспечивать приемлемое время визуализации информации, подготовленной различными разделами бортовой авионики. В данной работе рассматриваются вопросы обеспечения эффективной и безопасной генерации изображений из нескольких разделов операционной системы реального времени (ОСРВ) на один экран. Предполагается использование библиотеки стандарта OpenGL SC 2.0. Нам удалось предложить и реализовать эффективное решение для задачи одновременного вывода на экран информации из нескольких разделов. При этом мы смогли обеспечить приемлемую скорость визуализации 25 кадров в секунду.

***Ключевые слова*:** дисплей кабины пилота, бортовая система визуализации, многооконный дисплей, OpenGL SC, ОСРВ, авионика.

Layered Visualization of a Civil Aircraft Pilot Display

B. Kh. Barladian, N. B. Deryabin, A. G. Voloboy, L. Z. Shapiro, E. Yu. Denisov, V. A. Galaktionov
Keldysh Institute of Applied Mathematics RAS, Moscow, Russia

Abstract. Creating a pilot display visualization system is a challenging task. On the one hand, it must operate under a real-time operating system with its many inherent limitations. On the other hand, it must provide an acceptable time for rendering of information prepared by various sections of the onboard avionics. The paper considers issues of ensuring efficient and safe generation of images from several sections of the real-time operating system (RTOS) onto a single screen. It is assumed that the OpenGL SC 2.0 standard library is used. We propose and implement an efficient solution for the task of simultaneously displaying information from several sections on the screen. At the same time, we are able to provide an acceptable visualization speed of 25 frames per second.

***Keywords*:** cockpit display, on-board visualization system, multi-window display, OpenGL SC, RTOS, avionics.

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, 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 иной функциональности. В силу этой специфики КБО должны быть сертифицированы для использования на гражданских авиалайнерах. Сертификация, в частности, требует организации процесса разработки в соответствии с требованиями стандарта DO-178C (в русскоязычной редакции – КТ-178С) [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 комплекса бортового оборудования. Программный интерфейс ОСРВ, предназначенный для использования на борту воздушного судна, должен соответствовать стандарту ARINC 653 [6].

Для обеспечения безопасности каждое из функциональных приложений должно выполняться в отдельном разделе ОСРВ так, чтобы возможные ошибки в одном разделе имели бы минимальное влияние на работу приложений, работающих в других разделах. Большая часть этих требований обеспечивается ОСРВ, соответствующей стандарту ARINC 653. Однако у графических приложений есть дополнительная специфика, когда на один многофункциональный дисплей одновременно выводится различная информация, генерируемая многочисленными независимыми системами управления полетом.

В общем случае выводимая информация может выводиться как в различные окна на дисплее, так и в общие для различных приложений окна. Для удовлетворения требований по безопасности авиационные приложения используют специальные версии графических библиотек стандартов OpenGL SC. Различные подходы к разработке таких библиотек можно найти в работах [7-9].

В наших исследованиях использовалась ОСРВ JetOS [4] и собственная реализация графической библиотеки стандарта OpenGL SC 2.0 с использованием аппаратного ускорения на базе пакета MESA [10].

Постановка задачи

Задача визуализации из нескольких разделов в различные окна одного дисплея, так называемой многооконной визуализации, рассматривалась в работах [11-13]. В этих работах изображения генерировались независимо в каждом разделе, а затем компоновщик, который работал в отдельном разделе, компоновал их в заданных окнах экрана, используя библиотеку кадрового буфера. Однако такой подход не может быть осуществлен при использовании OpenGL с аппаратным ускорением, поскольку в типичной авиационной платформе имеется только один графический процессор и мы не можем использовать отдельные экземпляры OpenGL в каждом разделе. В силу требований безопасности и специфики JetOS графический драйвер, реализующий OpenGL SC, может работать только в одном разделе ОСРВ.

Следует также отметить, что подход с использованием компоновщика позволяет реализовать только так называемый «многооконный» режим, когда каждое приложение может выводить информацию только в свое окно. На практике, однако, бывает необходимость в некоторых случаях выводить информацию из нескольких приложений в общее окно на дисплее. Поэтому для реализации возможности синтеза графики из нескольких разделов с использованием аппаратного ускорения в OpenGL мы предложили специальный подход, когда все команды OpenGL выполняются в одном специальном разделе операционной системы JetOS. Первые шаги по реализации нашего подхода были сделаны в работе [10] для реализации библиотеки OpenGL SC 1.01 на платформе i.MX6 с GPU Vivante.

Предлагаемое решение

Работу приложения, использующего библиотеку OpenGL для визуализации, можно разделить на две компоненты:

1) подготовка различных данных, необходимых для работы OpenGL: геометрия и ее параметры, различные атрибуты и т.д.;

2) непосредственный вызов функций OpenGL с подготовленными параметрами.

Для использования аппаратного ускорения в OpenGL в нашей реализации эти компоненты выполняются в разных разделах операционной системы JetOS. Каждое приложение, подготавливающее данные, выполняется в своем разделе (рисующие разделы). Оно не осуществляет реальные вызовы функций OpenGL, а записывает индексы необходимых для визуализации функций OpenGL с соответствующими параметрами в специальный буфер (массив). Этот массив находится в общей памяти с разделом, который и вызывает функции библиотеки OpenGL (растеризующий раздел). Когда вся информация, необходимая для генерации изображения одного кадра, подготовлена рисующими разделами, то начинает работать растеризующий раздел.

Для работы системы нам в первую очередь необходимо передать данные и команды OpenGL, сформированные в рисующих разделах в растеризующий раздел, где эти команды будут реально выполняться. Для этих целей мы разработали две специальные библиотеки для рисующих и растеризующего разделов: liboglout2 и liboglin2 соответственно. Суффикс 2 в именах этих библиотек означает, что они предназначены для работы с командами OpenGL стандарта OpenGL SC 2.0. В работе [10] рассматриваемый подход первоначально разрабатывался для стандарта OpenGL SC 1.1 и там библиотеки назывались соответственно liboglout1 и liboglin1. С точки зрения предложенной нами технологии разница между стандартами OpenGL SC 1.1 и 2.0 в первую очередь определяется разными наборами функций OpenGL. Кроме того, для стандарта OpenGL SC 2.0 необходимо поддерживать возможность использования нескольких шейдерных программ и шейдерных объектов типа UNIFORM даже в одном разделе ОСРВ.

Библиотека liboglout2 содержит набор функций, эмулирующих соответствующие функции стандарта OpenGL SC 2.0, только вместо их выполнения она записывает их идентификаторы в участок общей памяти. В эту же общую память записываются также все параметры и данные, необходимые для выполнения этих функций.

Эти эмулирующие функции относительно просты и имеют ту же нотификацию, что и соответствующие им функции OpenGL. Если такая функция имеет фиксированное число параметров, то в текущую позицию массива записывается индекс вызываемой функции, а за ним подряд значения параметров. Это относится к большинству функций OpenGL: таких, как, например, glClearColor(), glClearDepthf(), glClearStencil(), glUniform1i() и т.д. Этот же способ используется тогда, когда передаются массивы известной длины. В более сложных случаях, когда передаются указатели на массивы с неопределенной в момент вызова длиной (например, при вызове glVertexAttribPointer()), технология кодирования вызова несколько усложняется. В момент вызова таких функций указатель запоминается, а данные по этому указателю используются позже при вызове функций glDrawArrays() или glDrawRangeElements(), когда необходимые размерности становятся известными. При этом массивы с запомненными указателями (как правило, координаты вершин, или их текстурные координаты) копируются в зарезервированную область общей памяти. Использоваться они будут в библиотеке liboglin2 при выполнении реальных функций OpenGL. Библиотека liboglout2 связывается (линкуется) с рисующим разделом вместо реальной OpenGL SC 2.0. Сам код раздела при этом практически не изменяется. Необходимы только определенные изменения в конфигурационных файлах и некоторый код в main(), который пишется по общему шаблону.

Декодирование информации, записанной приложениями с помощью библиотеки liboglout2, в вызовы функций OpenGL осуществляется в одном растеризующем разделе с помощью библиотеки liboglin2. Оно выполняется последовательно для каждого пакета данных, сгенерированного соответствующими приложениями. Библиотека liboglin2 состоит из одной функции ProcessOGL_Input(), на вход которой подается массив, записанный в рисующем разделе. Она последовательно читает данные из массива, по прочитанному индексу функции OpenGL переходит к участку кода, написанного для данной функции, извлекает из последующих элементов массива параметры функции, включая значения указателей, и вызывает с этими параметрами заданную функцию OpenGL. Библиотека liboglin2 связывается (линкуется) с растеризующим разделом. Этот раздел также связывается (линкуется) с библиотекой OpenGL SC 2.0. На рисунке 1 представлена схема синтеза графики из нескольких разделов с учетом использования библиотек liboglout2 и liboglin2.

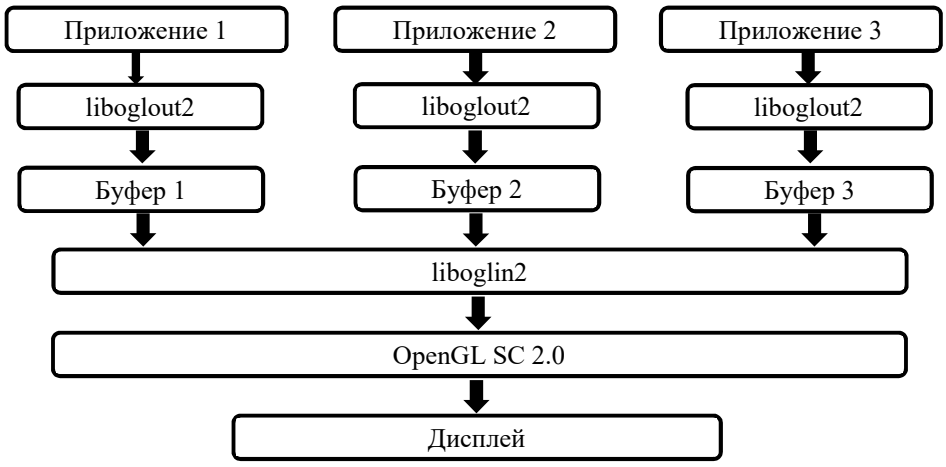

Рисунок 1. Схема работы приложений с использованием библиотек liboglout2 и liboglin2

Проблемы в этой схеме возникают при использовании нескольких шейдерных программ, текстур и щейдерных объектов, создаваемых в разных рисующих разделах. Реальное выполнение команд OpenGL происходит в растеризующем разделе, где и назначаются (либо генерируются) идентификаторы таких объектов. Рисующие разделы не имеют обратной связи с растеризующим разделом и не могут узнать текущий идентификатор объекта. Идентификаторы объектов одного типа

должны быть уникальными и различаться, если они созданы в разных разделах. Кроме того, следует учитывать, что в общем случае код может быть сгенерирован разными версиями пакета SCADE (с помощью которого типично генерируются программы для авионики) или написан вручную. Мы должны учитывать эту проблему при использовании идентификаторов следующих объектов:

- идентификаторы шейдерных программ, которые определяются вызовом функции glCreateProgram();
- идентификаторы щейдерных объектов типа UNIFORM, которые определяются вызовом функции glGetUniformLocation();
- идентификаторы щейдерных объектов типа VERTEX ATTRIBUTE, которые определяются вызовом функции glGetAttribLocation();
- идентификаторы текстур, которые определяются вызовом функции glGenTextures().

Для решения этой проблемы в библиотеке liboglout2 используется внутреннее собственное определение этих идентификаторов. Когда в liboglin2 необходимо использовать реальное значение идентификатора при вызове функции OpenGL, то оно определяется с помощью соответствующей таблицы по внутреннему значению идентификатора библиотеки liboglout2. Экземпляры библиотек liboglout2 в каждом разделе используют собственное адресное пространство, поэтому для последовательной нумерации идентификаторов одного типа в разных разделах используются общие переменные – указатели идентификаторов каждого типа, хранящиеся в общей памяти рисующих разделов.

```
static UINT32* prog_ident_ptr;
static UINT32* tex_ident_ptr;
static UINT32* uniform_ident_ptr;
static UINT32* attrib_ident_ptr;
```

Для связи этих переменных с библиотекой liboglout2, прилинкованной к данному конкретному рисующему разделу, указатели устанавливаются в библиотеку liboglout2 с помощью соответствующих функций:

```
VOID SetProgramIdentPtr(GLuint* identPtr)
        {
         sProgIdentPtr = identPtr;
        }
VOID SetTextureIdentPtr(GLuint* identPtr)
        {
         sTexIdentPtr = identPtr;
        }
VOID SetUniformIdentPtr(GLuint* identPtr)
        {
         sUniformIdentPtr = identPtr;
        }
VOID SetAttribIdentPtr(UINT32* identPtr)
        {
         sAttribIdentPtr = identPtr;
        }
```

Эта установка должна быть сделана до первого обращения раздела к функциям OpenGL. При вызове в рисующем разделе функции OpenGL для получения идентификатора объекта библиотека liboglout2 возвращает значение по нужному из указанных указателей и увеличивает это значение на 1. Такой подход гарантирует, что значения идентификаторов объектов одного типа не будут повторяться при их получении из разных рисующих приложений. В библиотеке liboglin2 для определения идентификатора объекта данного типа имеется массив, для которого идентификатор из liboglout2 рассматривается как индекс, значение массива по которому равно идентификатору, назначенному библиотекой OpenGL.

В каждом разделе теперь потенциально может использоваться собственный программный объект, один или несколько. Эти программные объекты могут создаваться явно с помощью вызова функции

glCreateProgram() или неявно внутри библиотеки OGLX пакета SCADE. Предложенный подход в некотором смысле автоматически поддерживает оба варианта создания программных объектов. Перед вызовом функций OpenGL необходимо установить используемый программный объект функцией glUseProgram(sProgIdent). Переменная sProgIdent объявлена в библиотеке liboglout2. Она содержит идентификатор программного объекта, используемого разделом. Если использовался пакет SCADE, то это единственный способ получить этот идентификатор. В случае явного вызова функции glCreateProgram() идентификатор программного объекта будет возвращен ею. Идентификаторы текстур, объектов типа UNIFORM и типа VERTEX ATTRIBUTE используются обычным образом при прямом написании кода раздела, использующего функции OpenGL. В случае использования пакета SCADE в библиотеке OGLX для идентификаторов будут автоматически использоваться внутренние переменные библиотеки liboglout2.

Результаты тестов

На рисунке 2 приведен результат генерации изображений из пяти разделов, одновременно передающих информацию для вывода на экран. Каждый раздел использует один или два собственных программных объекта. В левой верхней части экрана приведено изображение, сгенерированное приложением, подготовленным с помощью пакета SCADE R16 с кодогенератором KCG 6.4.3. В правой верхней части экрана отображается изображение, сгенерированное приложением «Карта перемещения по аэропорту» (AMM Airport Moving Map), написанным по стандарту OpenGL SC 2.0 без использования пакета SCADE. Слева внизу – приложение, использующее программный объект из пакета SCADE R19 с кодогенератором KCG 6.7.5, но не использующее пакет для генерации кода. В центре внизу расположено аналогичное приложение, использующее программный объект из пакета SCADE R16 с другими текстурами. Справа внизу изображение, сгенерированное приложением, которое использует два программных объекта – один из пакета SCADE R16, а второй из пакета SCADE R19. Скорость визуализации изображений из пяти разделов – 25 кадров в секунду.

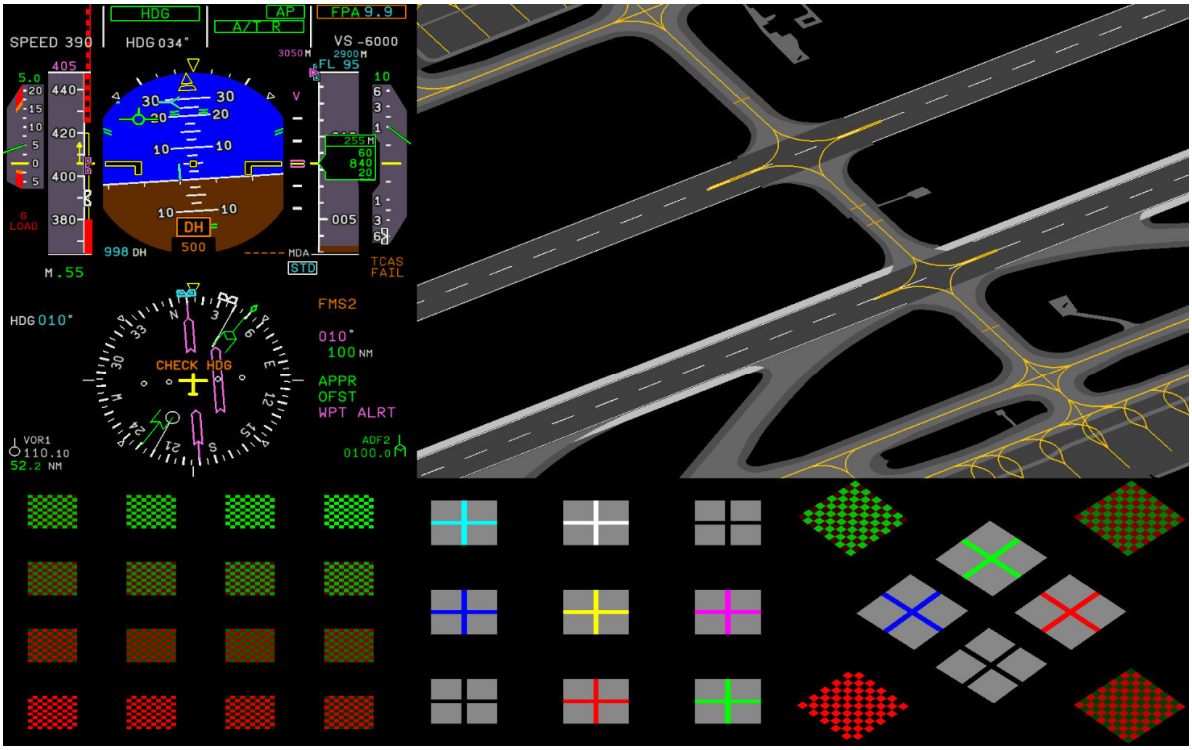

Рисунок 2. Генерация изображений из пяти разделов с использованием разных программных и текстурных объектов

Верхние изображения на рисунке 2 имеют также и смысловое значение для пилота: слева показан основной дисплей полета (Primary flight display), а слева – взлетно-посадочные полосы и рулежные дорожки аэропорта. Однако для тестирования нашего подхода нам было важно, что все пять разделов

генерировали изображения различными средствами, используемыми при разработке приложений для бортового программного обеспечения.

На рисунке 3 приведен результат генерации изображений из трех разделов.

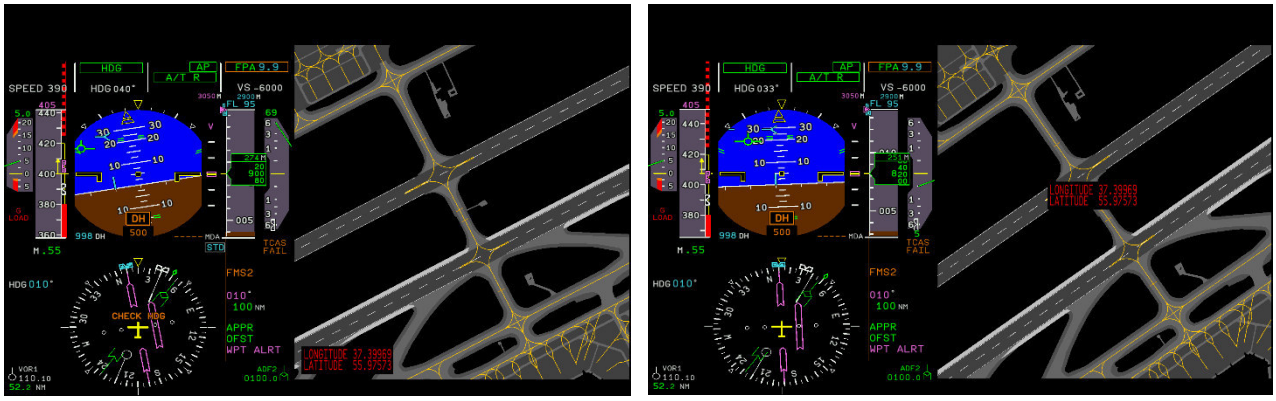

Рисунок 3. Генерация изображений из трех разделов

Левая и правая части рисунка 3 нам уже знакомы – это основной дисплей полета и карта перемещения по аэропорту. Они генерируются в первых двух разделах. Третий раздел добавляет надпись поверх изображения, генерируемого вторым разделом. Положение надписи может варьироваться в зависимости от состояния оборудования самолета или его положения в пространстве, что и показано на двух частях рисунка, соответствующих разным кадрам.

Заключение

Создание системы визуализации дисплея пилота является непростой задачей. С одной стороны, она должна работать под операционной системой реального времени со множеством присущих ей ограничений. Также она должна быть создана таким образом, чтобы ее можно было сертифицировать (иначе она не может быть использована в бортовом оборудовании). С другой стороны, она должна обеспечивать приемлемое время визуализации информации, подготовленной различными разделами бортовой авионики.

Нам удалось предложить и реализовать эффективное решение для задачи одновременного вывода на экран информации из нескольких разделов. При этом программы приложений, работающих в этих разделах, могут быть написаны с помощью различных средств: как с использованием разных версий пакета SCADE, так и без него. Мы смогли обеспечить при этом приемлемую скорость визуализации 25 кадров в секунду.

Список литературы

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. OSDAY 2015. URL: http://osday.ru/solodelov.html.

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. Т. 29, № 3. С. 171-178. DOI: 10.15514/ISPRAS-2017-29(3)-10.

5. DO-178C Software Considerations in Airborne Systems and Equipment Certification. URL: http://www.rtca.org/store_product.asp?prodid=803.

6. Avionics application software standard interface (ARINC 653). SAE-ITC, 2015.

7. Baek N., Kim K.J. Design and implementation of OpenGL SC 2.0 rendering pipeline // Cluster Comput. 2019. Vol. 22. P. 931–936. URL: https://doi.org/10.1007/s10586-017-1111-1.

8. Baek N., Lee H., OpenGL ES 1.1 Implementation Based on OpenGL // Multimedia Tools and Applications. 2012. Vol. 57, iss. 3. P. 669–685.

9. Yoon J., Baek N., Hwanyong L. Graphics Rendering Based on OpenVG and Its Use Cases with Wireless Communications. Wireless Personal Communications. 2017. Vol. 94, iss. 2. P. 175–185. doi: 0.1007/s11277-015-3163-y.

10. Specifics of the Development of an On-Board Visualization System for Civil Aircrafts / B.Kh. Barladian, N.B. Deryabin, A.G. Voloboy, et al. // Programming and Computer Software. 2024. Vol. 50, iss. 3. P. 215-223. DOI: 10.1134/S0361768824700014.

11. A Safety Critical Compositor for OpenGL SC 1.0.1 and OpenGL SC 2.0. URL: http://www.coreavi.com/sites/default/files/compositor_whitepaper_final.pdf.

12. EGL_EXT_compositor. URL: http://www.coreavi.com/sites/default/files/coreavi_product_brief_egl_ext_compositor.pdf.

13. High speed visualization in the JetOS aviation operating system using hardware acceleration / B.Kh. Barladian, N.B. Deryabin, A.G. Voloboy, V.A. Galaktionov, L.Z. Shapiro // CEUR Workshop Proceedings. 2020. Vol. 2744. P. 107:1-107:9. DOI: 10.51130/graphicon-2020-2-4-3.