DOI: 10.15514/ISPRAS-2020-32(1)-3

Система визуализации для авиационной ОС реального времени JetOS

[1] Б.Х. Барладян, ORCID: 0000-0002-2391-2067 <bbarladian@gmail.com>
[1] Л.З. Шапиро, ORCID: 0000-0002-6350-851X <pls@gin.keldysh.ru>
[2] К.А. Маллачиев, ORCID: 0000-0002-4112-5403 <mallachiev@ispras.ru>
[2,3,4,5] А.В.Хорошилов, ORCID: 0000-0002-6512-4632 <khoroshilov@ispras.ru>
[6] Ю.А. Солоделов, ORCID: 0000-0001-5891-7645 <yasolodelov@2100.gosniias.ru>
[1] А.Г.Волобой, ORCID: 0000-0003-1252-8294 <voloboy@gin.keldysh.ru>
[1] В.А. Галактионов, ORCID: 0000-0001-6460-7539 <vlgal@gin.keldysh.ru>
[6] И.В. Ковернинский, ORCID: 0000-0002-8571-324X <ivkoverninsk@2100.gosniias.ru>
[1] Институт прикладной математики им. М.В. Келдыша РАН,
125047, Россия, Москва, Миусская пл., д. 4
[2] Институт системного программирования имени В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[3] Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1
[4] НИУ Высшая школа экономики,
101978, Россия, г. Москва, ул. Мясницкая, д. 20
[5] Московский физико-технический институт,
141701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9
[6] Государственный научно-исследовательский институт авиационных систем,
125319, Россия, Москва, ул. Викторенко, 7

Аннотация. В работе рассматриваются вопросы создания систем визуализации для бортовых комплексов гражданской авиации. Все программное обеспечение, используемое на борту судна, должно соответствовать международно-принятым стандартам безопасности. Это накладывает дополнительные требования и к используемому оборудованию, и к процессу разработки системы. Данная работа посвящена специфике использования многоядерных процессоров в авиационных встраиваемых системах для повышения производительности программной реализации библиотеки OpenGL SC. Возможность использования многоядерных процессоров в критических для безопасности системах обеспечивается в перспективной российской операционной системе реального времени (ОСРВ) JetOS. Рассматриваются также реализация многооконной визуализации с использованием библиотеки OpenGL SC.

Ключевые слова: OpenGL SC; встроенные системы; операционная система реального времени; многоядерные вычисления; ускорение рендеринга; многооконность; компоновщик

Для цитирования: Барладян Б.Х., Шапиро Л.З., Маллачиев К.А., Хорошилов А.В., Солоделов Ю.А., Волобой А.Г., Галактионов В.А., Ковернинский И.В. Система визуализации для авиационной ОС реального времени JetOS. Труды ИСП РАН, том 32, вып. 1, 2020 г., стр. 57-70. DOI: 10.15514/ISPRAS–2020–32(1)–3

Rendering System for the Aircraft Real-Time OS JetOS

[1] B.Kh. Barladian, ORCID: 0000-0002-2391-2067 <bbarladian@gmail.com>
[1] L.Z. Shapiro, ORCID: 0000-0002-6350-851X <pls@gin.keldysh.ru>
[2] K.A. Mallachiev, ORCID: 0000-0002-4112-5403 <mallachiev@ispras.ru>
[2,3,4,5] A.V. Khoroshilov, ORCID: 0000-0002-6512-4632 <khoroshilov@ispras.ru>
[6] Y.A. Solodelov, ORCID: 0000-0001-5891-7645 <yasolodelov@2100.gosniias.ru>
[1] A.G. Voloboy, ORCID: 0000-0003-1252-8294 <voloboy@gin.keldysh.ru>
[1] V.A. Galaktionov, ORCID: 0000-0001-6460-7539 <vlgal@gin.keldysh.ru>
[6] I.V. Koverninskiy, ORCID: 0000-0002-8571-324X <ivkoverninsk@2100.gosniias.ru>
[1] Keldysh Institute of Applied Mathematics Russian Academy of Science,
4, Miusskaya sq., Moscow, 125047, Russia
[2] Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[3] Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[4] National Research University, Higher School of Economics
20, Myasnitskaya Ulitsa, Moscow, 101978, Russia
[5] Moscow Institute of Physics and Technology (State University),
9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russian Federation
[6] State Research Institute of Aviation Systems
7, Viktorenko street, Moscow, 125319, Russia

Abstract. The paper discusses the creation of rendering systems for airborne civil aviation systems. All software used on board must comply with internationally accepted safety standards. This imposes additional requirements on both the hardware used and the system development process. This work is devoted to the specifics of using multi-core processors in aviation embedded systems to improve the performance of software implementation of the OpenGL SC library. The possibility of using multi-core processors in safety-critical systems is provided by the Russian real-time operating system JetOS. Implementation of multi-window rendering using the software OpenGL SC library is also considered.

Keywords: OpenGL SC; embedded systems; real-time operating system; multi-core calculations; rendering acceleration; multi-windowing; compositor

For citation: Barladian B.Kh., Shapiro L.Z., Mallachiev K.A., Khoroshilov A.V., Solodelov Y.A., Voloboy A.G., Galaktionov V.A., Koverninskiy I.V. Rendering System for the Aircraft Real-Time OS JetOS. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 1, 2020. pp. 57-70 (in Russian). DOI: 10.15514/ISPRAS–2020–32(1)–3

1. Введение

Современные комплексы бортового оборудования самолетов проектируются на основе концепции интегрированной модульной авионики [1, 2], в основе которой лежит объединение приборов и бортовых процессоров в единую сеть, управляемую операционной системой реального времени (ОСРВ). Использование этой концепции и современного оборудования (например, компьютерных дисплеев вместо механических приборов) позволяет снизить количество кабелей и устройств на борту и, тем самым, уменьшить взлетный вес лайнера. Таким образом, возникает задача создания программного обеспечения дисплеев в кабине пилота, которое должно обеспечивать надежное отображение информации с интерактивной скоростью, используя ресурсы процессора с пониженным энергопотреблением, который устанавливается на борту.

Процесс создания авиационной техники должен следовать международным стандартам для авиационной промышленности, включая и соответствующие стандарты на разработку программного обеспечения. Без соблюдения этих стандартов и получения

соответствующих сертификатов допуск к полетам и экспорт построенных в России самолетов невозможен. Международный стандарт ARINC 653 [3] описывает требования к ОСРВ и программный интерфейс между прикладным авиационным ПО и операционной системой. Также отображение информации на пилотном дисплее должно удовлетворять стандарту OpenGL SC (Safety Critical) [4]. Важным требованием является также возможность сертификации в соответствии с требованиями авиационных стандартов DO-178C [5]. Так как сертификация графического ускорителя без участия производителя невозможна, то мы не рассматриваем вопрос создания системы визуализации с их использованием.

В работе [6] рассматривалась программная реализация графической библиотеки OpenGL SC, предназначенная для работы под управлением перспективной российской бортовой операционной системы реального времени JetOS [7]. Хотя нам удалось значительно ускорить стандартную программную реализацию OpenGL, однако достичь скорости реального времени на одном ядре типичного авиационного процессора не представляется возможным. При проведении исследований операционная система JetOS еще не была полностью реализована, в частности, не поддерживались многоядерные процессоры. В соответствии со стандартом ARINC 653 в авиационных ОСРВ нельзя использовать многопоточность. Поэтому часть исследований по возможному распараллеливанию алгоритмов была сделана под операционной системой Linux. В настоящее время JetOS поддерживает специальное расширение стандарта ARINC 653, позволяющее использовать многоядерные процессоры. Расширение называется Asymmetric Multi-Processing (AMP). Это позволило перейти к задаче ускорения визуализации, используя многоядерные процессоры, не отклоняясь от разрешенных стандартов.

2. Технология AMP

Следует отметить существенную разницу между технологией AMP и многопоточностью, используемой в Linux для ускорения OpenGL SC на многоядерных компьютерах. В то время как многопоточность обеспечивает эффективную конкурентную работу нескольких потоков в одном адресном пространстве, технология AMP в JetOS поддерживает возможность запуска нескольких модулей (т.е. экземпляров JetOS), каждое из которых работает на своем ядре процессора независимо от других модулей. Этот подход обеспечивает безопасность и надежность, необходимые для бортового программного обеспечения, но является менее эффективным в сравнении с производительностью, достигаемой при использовании технологии многопоточности. Конфигурация проекта, использующего AMP технологию, называется AMP проектом. Таким образом, AMP проект позволяет создавать приложения с параллельными вычислениями, отдельные части которых выполняются на различных ядрах процессора под управлением различных экземпляров JetOS.

Разработанное нами ускорение визуализации для многоядерного компьютера базируется на подходе распараллеливания библиотеки SC OpenGL, предложенном в [6]. В его основе лежит параллельная генерация нескольких последовательных кадров. Основными проблемами при реализации такого подхода являетются обмен информацией между различными частями приложения (различными модулями) и синхронизация их работы.

AMP проект поддерживает именованные разделяемые блоки памяти, доступ к которым возможен из различных модулей. Использование таких блоков памяти, обеспечивает обмен информацией между модулями. Для синхронизации работы модулей мы будем использовать специальные объекты, называемые **событиями**.

Для синхронизации процессов, выполняемых модулями A и B, мы используем объект событие (AMP_EVENT), доступный одновременно в обоих модулях. Для создания этого объекта мы использовали небольшие блоки памяти, доступ к которым возможен из этих

модулей. Состояние объекта **событие** определяется значением целочисленной переменной, хранящейся в этом блоке. Мы будем рассматривать только два состояния этого объекта: ***Событие взведено*** **(AMP_UP)** – значение переменной 1 и ***Событие сброшено*** (**AMP_DOWN)** – значение 0. Невозможность одновременного изменения состояния объекта **события** из двух модулей обеспечивается с помощью использования атомарных операций для доступа к объекту **событие**.

Для удобства работы с событиями нами был разработан набор функций, обладающих мнемоническими именами (см. листинг 1).

```
#define AMP_UP         1
#define AMP_DOWN   0
typedef int*  AMP_EVENT;
/// Get event state.
int AMP_GetEventState(AMP_EVENT ev)
{
  return ev[0];
}
/// Set the event in the  state "up".
void AMP_SetEvent(AMP_EVENT ev)
{
  atomic_int *atomic = (atomic_int *)ev;
  atomic_store(atomic, AMP_UP);
}
/// Set the event in the  state "down".
void AMP_ResetEvent(AMP_EVENT ev)
{
  atomic_int *atomic = (atomic_int *)ev;
  atomic_store(atomic, AMP_DOWN);
}
/// Infinitely wait while event is in state "down".
void AMP_WaitEvent(AMP_EVENT ev)
{
  RETURN_CODE_TYPE ret;
  while (ev[0] == AMP_DOWN)
  {
    TIMED_WAIT(MILLISECOND, &ret);
  }
}
```

Листинг 1. Функции для работы с событиями
Listing 1. Functions to deal with events

Функция *TIMED_WAIT()*, выполняющая необходимое ожидание, обеспечивается операционной системой JetOS. Отметим еще, что указатели, определяющие события в разных модулях, могут отличаться друг от друга в силу разных адресных пространств в них, но переменные по этим указателям имеют одинаковые значения.

3. Повышение производительности библиотеки OpenGL

Рассмотрим использование AMP технологии для повышения производительности программной реализации библиотеки OpenGL SC. Схема работы библиотеки OpenGL на самом верхнем уровне при визуализации одного кадра приложения включает в себя три основных этапа, представленных на рис. 1.

Первый и третий этапы должны выполняться строго последовательно. На первом этапе каждый кадр обрабатывается последовательно по мере поступления новых инструкций из приложения. На третьем этапе изображение копируется из внутреннего буфера в буфер экрана. Этот этап выполняется драйвером кадрового буфера. Копирование должно выполняться последовательно кадр за кадром в порядке их создания приложением. Таким

образом, выполнение первого и третьего этапов не может быть распараллелено, но они могут выполняться одновременно со вторым этапом для разных кадров. К счастью, время выполнения первого этапа для типичных приложений авионики обычно относительно мало.

Рис. 1. Схема работы OpenGL для генерации одного кадра
Fig. 1. OpenGL working scheme for generating one frame

Для распараллеливания вывода на экран с использованием технологии JetOS AMP на разных ядрах (разных экземплярах JetOS) выполняется параллельно генерация нескольких последовательных кадров. Количество их будет на единицу меньше чем количество экземпляров JetOS. В нашей схеме эти экземпляры JetOS будут выплнять роль серверов. Один экземпляр JetOS будет работать как клиент. В нем также реализуются первый и третий этапы, представленные на рис. 1.

В отличие от многопоточной реализации, AMP технология не позволяет запустить произвольное количество экземпляров JetOS. Количество экземпляров JetOS не может превышать количество ядер для данного процессора. Типичный авиационный процессор PowerPC (P3041) [8], используемый в наших исследованиях, имеет четыре ядра. Потенциально в авиационном оборудовании может использоваться процессор PowerPC (P4080) с восемью ядрами. В нашем случае возможно параллельно запустить четыре экземпляра JetOS. Они использовались следующим образом:

a. клиент – обрабатывает команды OpenGL и выводит подготовленное изображение на экран;
b. три сервера, которые выполняют генерацию кадров параллельно.

Теперь работа OpenGL с параллельной генерацией кадров может быть представлена схемой, показанной на рис. 2.

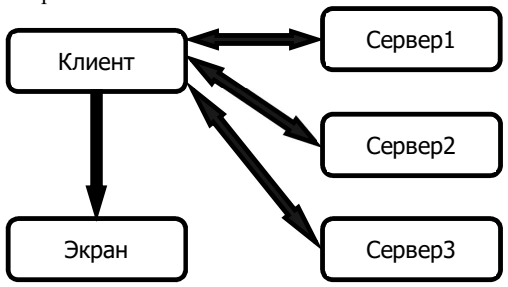

Рис. 2. Схема работы OpenGL с использованием четырех экземпляров JetOS
Fig. 2. OpenGL operational pattern using four JetOS instances

Обмен данными между клиентом и сервером осуществляется через блок общей памяти. Этот блок содержит контекст OpenGL, в котором хранятся инструкции OpenGL для генерируемого данным сервером кадра, обработанные на первом этапе (см. рис. 1). В этом контексте также выделены все соответствующие буферы, необходимые для работы OpenGL. В частности, здесь размещается буфер изображения, в котором будет создано конечное изображение данного кадра.

Для синхронизации работы клиентского модуля с несколькими серверами используются два события для каждого сервера:

a. **Start_render** – взводится клиентом, когда данные, необходимые для генерации данного кадра, готовы и сервер может ее начать;
b. **End_render –** взводится сервером, когда он завершил генерацию кадра. Тогда клиент может передать подготовленное изображение на дисплей с помощью библиотеки кадрового буфера, и продолжить обработку инструкций OpenGL для следующего кадра.

Для реализации этой пары событий введен доступный клиенту и серверу 16-байтовый блок памяти. Первая половина этого блока используется для события **Start_render**, а вторая половина для события **End_render**. Псевдокод алгоритма работы клиента показан на листинге 2.

```
1. indx=0;
2.  Обработка команд OpenGL для текущего кадра;
3.  AMP_SetEvent(Start_render [indx]);
4.  indx++
5.   While(indx < S_NUM) go to p.2
6.  indx = 0;
7. AMP_WaitEvent(End_render[indx]);
8. AMP_ResetEvent(End_render [indx]);
9. Вывод изображения, полученного от сервера, на экран;
10.   Обработка команд OpenGL для следующего кадра;
11.   AMP_SetEvent(Start_render [indx]);
12.   indx++;  indx = indx % S_NUM
13.   Go to p.7
```

Листинг 2. Псевдокод работы клиента
Listing 2. Pseudo-code of client operation

Константа S_NUM равна количеству серверов (т.е. 3 в нашем случае).

Псевдокод алгоритма работы *i*-того сервера, каждый из которых работает со своими 16-битовыми блоками, можно записать так, как показано на листинге 3.

```
1.  AMP_WaitEvent(Start_render [i]);
2.  AMP_ResetEvent(Start_render [i]);
3.  Генерация кадра;
4.  AMP_SetEvent(End_Render[i]);
5.  Go to p.1;
```

Листинг 3. Псевдокод работы i-го сервера
Listing 3. Pseudo-code of client operation

Во время инициализации все события **Start_render** устанавливаются в AMP_DOWN, а события **End_render** в AMP_UP. После запуска клиент сначала обрабатывает инструкции OpenGL для первых трех кадров и передает заполненные контексты на серверы. И только затем, после взведения события **End_render**[0], начинает передачу сгенерированных изображений на дисплей.

Также необходимо было решить проблему с указателями на контекст OpenGL, которые устанавливаются в одном модуле (на клиенте), а используются в другом (на сервере). Проблема заключается в том, что модули имеют разные адресные пространства. Данные, совместно используемые общими блоками памяти, имеют одинаковые значения, но в

адресных пространствах разных модулей указатели принимают разные значения. Таким образом, их следует корректировать в зависимости от модуля, поскольку адреса начала блока памяти в разных модулях разные. Чтобы решить эту проблему, в контексте OpenGL сохраняется значение указателя на него в адресном пространстве клиентского модуля. Используя разницу между текущим значением указателя на контекст в модуле сервера и сохраненным, корректируются указатели на сервере перед их использованием для генерации кадра, а затем они восстанавливаются перед передачей обработки очередного кадра клиенту.

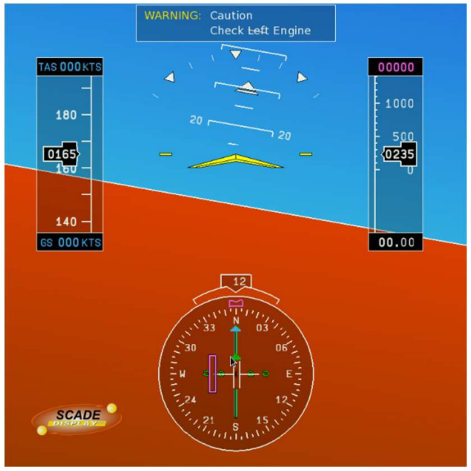

Рис. 3. Приложение GlassCockpit
Fig. 3. GlassCockpit application

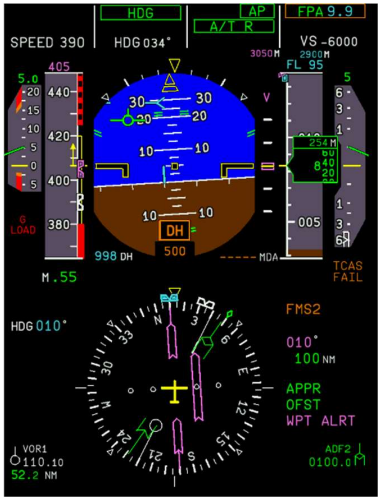

Рис. 4. Основной дисплей полета (PFD)
Fig. 4. Primary Flight Display (PFD)

Результаты тестов для приложений GlassCockpit и PFD (рис. 3 и 4), а также приложения визуализации рельефа местности SVS представлены на рис. 5. Использование трех серверов (четырех процессорных ядер) ускоряет визуализацию примерно в 2,9 раза для

приложения PFD, в 2,8 раза для приложения ClassCockpit и в 3,1 раза для приложения SVS.

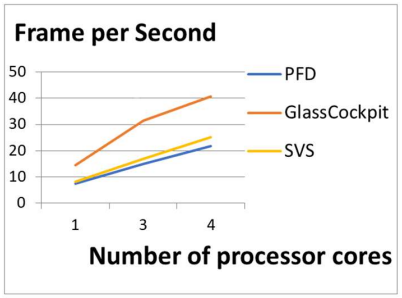

Рис. 5. Зависимость скорости визуализации от количества используемых процессорных ядер
Fig. 5. Dependence of visualization speed on the number of processor cores used

4. Многооконная визуализация

При разработке кабин современных самолетов существует тенденция использовать большие дисплеи, чтобы объединить в себе информацию о полетной навигации и состоянии оборудования самолета. Рис. 6 показывает кабину самолета MS-21, иллюстрирующую данную тенденцию.

Рис. 6. Кабина самолета MS-21
Fig. 6. Cabin of the aircraft MS-21

Количество дисплеев в кабине самолета MS-21 уменьшено (например, по сравнению с кабиной самолета Airbus A320), но дисплеи стали намного шире и позволяют отображать больше информации.

В итоге информация о полете и работе оборудования, генерируемая многочисленными системами управления полетом, должна отображаться на широкоэкранных, многофункциональных дисплеях. Она должна отображаться одновременно и в удобной для восприятия форме. В частности, этой информацией является скорость полета, указатель положения, высотомер, указатель поворота и скольжения, указатель вертикальной скорости и т. д. При этом также должны отображаться такие технические характеристики, как частота вращения двигателя, давление масла и количество топлива. Кроме того, необходимо визуализировать карту местности, различные пневматические, гидравлические и электрические цепи, данные метеорологических радаров, различные виды предупреждений и многое другое. Эта информация обычно генерируется независимыми системами, и они не должны мешать друг другу в соответствие с требованиями стандарта ARINC 653.

Для изображения информации от нескольких систем на одном экране современные операционные системы реализуют многооконный интерфейс, когда содержимое каждого приложения отображается в собственном окне. Упрощенный подход состоит в том, чтобы

позволить каждому приложению открывать неперекрывающееся с другими окно на дисплее. Такой подход позволяет ускорить визуализацию, в то же время его реализация для систем, критичных для безопасности, требует значительных усилий. Эта задача решается с помощью разработки компоновщика (compositor), который обеспечивает поддержку эффективной многооконной визуализации.

Различные подходы к реализации компоновщика для систем, критичных для безопасности, рассмотрены в [9]. Одной из реализаций компоновщика является расширение EGL_EXT_compositor для CoreAVI, который обеспечивает многооконную визуализацию для OpenGL SC 1.0.1 и OpenGL SC 2.0 [10].

Разрабатываемая нами библиотека OpenGL SC [6] предназначена для работы под операционной системой JetOS. Это определяет специфику разработки и предъявляет существенные требования к разрабатываемому коду и алгоритмам. В частности, для сертификации системы требуется полный доступ к исходным кодам библиотеки OpenGL SC и компоновщика. С другой стороны, при разработке компоновщика мы можем воспользоваться AMP технологией ОСРВ JetOS.

В работе [9] рассматриваются два основных типа графической компоновки: компоновка на аппаратном уровне и компоновка в кадровый буфер. Несмотря на то, что преимущества компоновки уровня оборудования включают хорошую производительность, энергосбережение и эффективность при работе с большим количеством обновлений, этот подход требует дополнительной полосы пропускания для отображения всех окон. Это также требует специальной поддержки драйвера кадрового буфера, который недоступен для нас на используемом в настоящее время оборудовании.

Компоновка в кадровый буфер объединяет элементы из нескольких приложений и внеэкранных буферов в один кадровый буфер. Затем кадровый буфер выводится на экран. Такая компоновка требует только одного слоя для отображения всех буферов. Фактически это единственный доступный подход в нашем случае. Схема визуализации данных при компоновке в кадровый буфер показана на рис. 7.

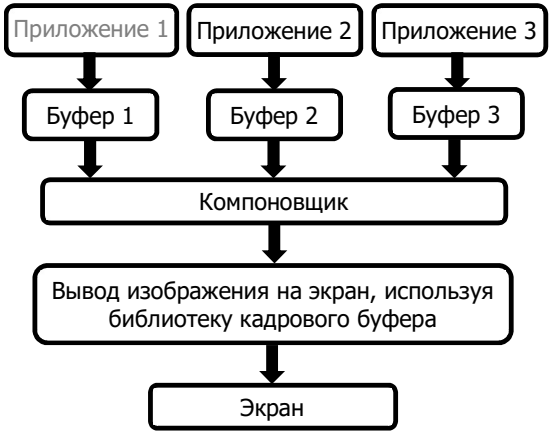

Рис. 7. Схема визуализации данных для компоновщика кадрового буфера
Fig. 7. Data visualization scheme for the framebuffer composer

Каждое приложение отображает данные с использованием библиотеки OpenGL SC в собственном буфере. Эти буферы затем передаются в компоновщик. Он формирует из них единый слой кадрового буфера и визуализирует его на дисплее с помощью библиотеки кадрового буфера. Основной проблемой здесь является эффективная синхронизация независимо работающих приложений и компоновщика.

Как было подробно описано в докладе [11], были исследованы два варианта реализации многооконной визуализации. Первый использует стандартные средства ARINC 653 – разделы, а второй использует AMP технологию JetOS.

5. Использование разделов для многооконной визуализации

В соответствии с требованиями ARINC 653 JetOS обеспечивает разделение памяти и времени между разделами. Диспетчеризация работы разделов осуществляется на постоянной, циклической основе. Каждому разделу отводится определенная фиксированная часть от полного периода работы системы. Таким образом, обеспечивается детерминистическое поведение системы. Каждое приложение и компоновщик работают в своем разделе операционной системы. Синтезированные изображения передаются из приложения в компоновщик с помощью специальных блоков общей памяти. Каждое приложение использует собственный блок памяти для синтеза изображений. Компоновщик имеет доступ к этому блоку памяти только для чтения.

Синхронизация между приложениями и компоновщиком обеспечивается сообщениями, передаваемыми между разделами по специальным каналам связи стандарта ARINC 653. Два канала используются между каждым приложением и компоновщиком. Первый канал используется приложением для информирования компоновщика о готовности изображения к визуализации. Второй канал используется компоновщиком для информирования приложения о том, что изображение было визуализировано, и приложение может снова использовать буфер для генерации изображения следующего кадра.

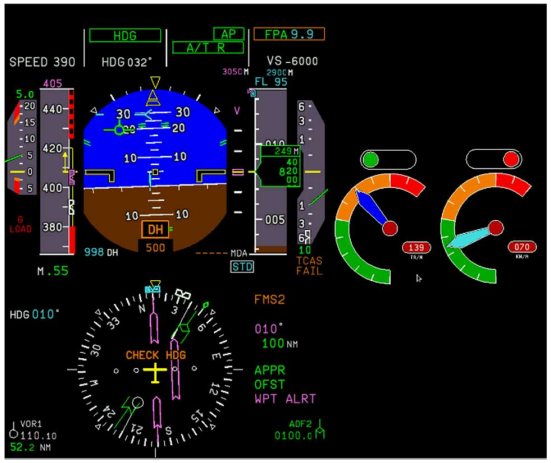

Рис. 8. Компоновка двух приложений
Fig. 8. A composition of two applications

Пример результата работы такого компоновщика приведен на рис. 8. Два приложения: PFD (приведенное на рис. 4) слева и простое приложение Counter справа работают одновременно (фактически – по очереди в соответствии с требованиями ARINC 653), а изображения, создаваемые приложениями, визуализируются компоновщиком.

Предложенный подход работает правильно, но скорость визуализации в данном примере недостаточна для авиационных приложений. Оба приложения работают со скоростью ~ 5 кадров в секунду. Есть несколько причин для такого поведения. Во-первых, типичный авиационный процессор PowerPC [8] обладает относительно низкой производительностью. Вторая причина заключается в том, что все разделы работают на

одном ядре процессора и имеют заранее определенное время его использования. Мы можем лишь попытаться оптимизировать это разделение времени с учетом реальных потребностей приложений. В данном примере время кадра состояло из 45 мс для приложения PFD, 15 мс для приложения Counter и 16 мс для компоновщика. Этот распределение обеспечивает относительно сбалансированный доступ к процессору для данных приложений. Очевидно, что для других приложений соотношения времен будет другим. А дальнейшее ускорение возможно только за счет использования всех процессорных ядер.

6. Использование технологии AMP для многооконной визуализации

В этом случае каждое приложение и компоновщик могут выполняться на своем экземпляре операционной системы. Для передачи изображения из приложения в компоновщик каждое приложение использует собственный блок памяти, имеющий общий доступ с компоновщиком. В настоящее время AMP технология не обеспечивает пересылку сообщений между модулями, работающими на разных ядрах процессора. Поэтому для синхронизации используется механизм событий, описанный в разд. 2. Для синхронизации взаимодействия каждого приложения с компоновщиком необходимо два события:

– взводится приложением, когда сгенерированное им изображение готово для визуализации компоновщиком;
– взводится компоновщиком, когда изображение уже было визуализировано и соответствующий блок памяти может снова использоваться приложением для генерации следующего кадра.

Тогда алгоритм работы приложения можно записать следующим образом (листинг 4):

```
While (true)
{
  AMP_WaitEvent(End_copy);
  AMP_ResetEvent(End_copy);
  Генерация изображения;
  AMP_SetEvent(Start_copy);
}
```

Листинг 4. Алгоритм работы приложения
Listing 4. Application operation algorithm

Обработка изображений компоновщиком происходит по алгоритму, показанному на листинге 5.

```
for (int i = 0; i < application number; i++)
{
  If (AMP_GetEventState(Start_copy[i]) == AMP_UP)
  {
     AMP_ResetEvent(Start_copy[i]);
     Вывод изображения на экран;
     AMP_SetEvent(End_copy[i]);
  }
}
```

Листинг 5. Обработка изображений компоновщиком
Listing 5. Image processing by the composer

На стадии инициализации все события **End_copy** взведены в состояние AMP_UP, а **Start_copy** сброшены в состояние AMP_DOWN.

Примеры изображений, полученных с помощью предложенного многооконного подхода, реализованного с использованием AMP технологии, показаны на рис. 9-11.

Рис. 9. Многооконный дисплей трех приложений: GlassCockpit, Counter, карта
Fig. 9. Multi-window display of three applications: GlassCockpit, Counter, map

Рис. 10. Многооконный дисплей двух приложений: GlassCockpit и рельеф местности
Fig. 10. Multi-window display of three applications: GlassCockpit and topographical relief

Рис. 11. Многооконный дисплей двух приложений: PFD и состояние дверей
Fig. 11. Multi-window display of two applications: PFD and door status

Наша реализация многооконной визуализации показала следующие скорости на процессоре PowerPC e500mc (4 ядра, 1 ГГц).

Рис. 9: все три приложения показывались со скоростью 16 кадров в секунду. Рис. 10: приложение GlassCockpit показывалось со скоростью 16 кадров в секунду, а визуализация рельефа местности – со скоростью 9.2 кадра в секунду. Рис. 11: приложение PFD (основной дисплей полета) обновлялось со скоростью 10.3 кадра в секунду, а более простое приложение, показывающее состояние дверей, – 21.7 кадра в секунду.

7. Заключение

Разработка системы визуализации данных о полете и отображения информации о состоянии самолета обладает своей спецификой, связанной с критически важными вопросами безопасности. Эта специфика часто не позволяет применять готовые, известные решения для той или другой функциональности. При разработке программного обеспечения для бортового оборудования необходимо также учитывать то, что на борт ставятся энергосберегающие и относительно низкоэффективные процессоры. В то же время система должна обеспечивать интерактивную скорость визуализации.

Проведенные исследования показали возможность повышения скорости визуализации программной реализации библиотеки OpenGL SC на типичном авиационном компьютере

на базе относительно слабого процессора PowerPC (1 ГГц) за счет использовании нескольких ядер процессора. С помощью AMP технологии, разрешенной для авиационных систем, было достигнуто ускорение как для работы одиночного приложения, так и для программного компоновщика, который обеспечивает компоновку в один кадровый буфер изображений, сгенерированных несколькими отдельными приложениями. При визуализации многооконного экрана достигнута интерактивная скорость ~15-20 кадров в секунду.

Список литературы / References

[1]. Федосов Е.А. Проект создания нового поколения интегрированной модульной авионики с открытой архитектурой. Полет, №8, 2008 г., стр. 15-22 / Fedosov E.A. Project On New-Generation Open Architecture Integrated Modular Avionics Development. Flight, №8, 2008, pp. 15-22 (in Russian).

[2]. Федосов Е.А., Косьянчук В.В., Сельвесюк Н.И. Интегрированная модульная авионика. Радиоэлектронные технологии, №1, 2015 г., стр. 66-71 / Fedosov E.A., Kosyanchuk V.V., Selvesyuk N.I. Integrated Modular Avionics. Radioelectronic Technologies, №1, 2015, pp. 66-71.

[3]. ARINC Standards Store. Available at https://www.aviation-ia.com/product-categories, accessed 15.12.2019.

[4]. Safety Critical Working Group. Available at https://www.khronos.org/openglsc, accessed 15.12.2019.

[5]. DO-178C Software Considerations in Airborne Systems and Equipment Certification. Available at https://my.rtca.org/NC__Product?id=a1B36000001IcmqEAC, accessed 15.12.2019.

[6]. Б.Х. Барладян, А.Г. Волобой, В.А. Галактионов, В.В. Князь, И.В. Ковернинский, Ю.А. Солоделов, В.А. Фролов, Л.З. Шапиро. Эффективная реализация OPENGL SC для авиационных встраиваемых систем. Программирование, том 44, № 4, 2018 г., стр. 3-10 / B.Kh. Barladian, A.G. Voloboy, V.A. Galaktionov, V.V. Knyaz', I.V. Koverninskii, Yu.A. Solodelov, V.A. Frolov, L.Z. Shapiro. Efficient Implementation of OpenGL SC for Avionics Embedded Systems. Programming and Computer Software vol. 44, № 4, 2018, pp. 207–212

[7]. Солоделов Ю.А., Горелиц Н.К. Сертифицируемая бортовая операционнаяс истема реального времени JetOS для российских проектов воздушныхсудов. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 171-178 / . Solodelov Yu.A., Gorelits N.K. Certifiable onboard real-time operation system JetOS for Russian aircrafts design. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 3, 2017. pp. 171-178 (in Russian). DOI: 10.15514/ISPRAS-2017-29(3)-10.

[8]. Central Processing Module (CPM/ P3041-VPX 3U). Available at http://www.nkbvs.ru/en/products/elektronnie-modyli/vpx-3u/moduli-universalnogo-protsessora-dannix-mypd-p3041/, accessed 15.12.2019.

[9]. A Safety Critical Compositor for OpenGL SC. Available at http://www.coreavi.com/sites/default/files/compositor_whitepaper_final.pdf, accessed 15.12.2019.

[10]. EGL_EXT_compositor. FACE-aligned Safety Critical Compositor. Available at https://coreavi.com/wp-content/uploads/2018/08/coreavi_product_brief_-_egl_ext_compositor.pdf, accessed 15.12.2019.

[11]. B.Kh. Barladian, L.Z. Shapiro, K.M. Mallachiev, A.V. Khoroshilov, Y.A. Solodelov, A.G. Voloboy, V.A. Galaktionov, I.V. Koverninskiy. Multi-windows rendering using software OpenGL in avionics embedded systems. In Proc. of the 29th International Conference on Computer Graphics and Vision. CEUR Workshop Proceedings, vol. 2485, 2019, paper 7.

Информация об авторах / Information about authors

Борис Хаимович БАРЛАДЯН, старший научный сотрудник, кандидат технических наук, доцент. Научные интересы: компьютерная графика, компьютерное моделирование.

Boris Haimovich BARLADYAN, Senior Researcher, Candidate of Technical Sciences, Associate Professor. Research interests: computer graphics, computer modeling.

Лев Залманович ШАПИРО, старший научный сотрудник, кандидат технических наук, доцент. Научные интересы: компьютерная графика, компьютерное моделирование, вычислительная оптика.

Lev Zalmanovich SHAPIRO, Senior Researcher, Candidate of Technical Sciences, Associate Professor. Research interests: computer graphics, computer modeling, computational optics.

Курбанмагомед Абдурагимович МАЛЛАЧИЕВ, младший научный сотрудник. В 2018 окончил аспирантуру ВМК МГУ им. М.В. Ломоносова. Научные интересы: верификация программного обеспечения, операционные системы реального времени.

Kurbanmagomed Abdurahimovich MALLACHIEV, Junior Researcher. In 2018 completed his postgraduate education at the CMC faculty of Lomonosov Moscow State University. Research interests: software verification, real-time operating systems.

Алексей Владимирович ХОРОШИЛОВ, ведущий научный сотрудник, кандидат физико-математических наук, директор Центра верификации ОС Linux в ИСП РАН, доцент кафедр системного программирования МГУ и ВШЭ. Основные научные интересы: методы проектирования и разработки ответственных систем, формальные методы программной инженерии, методы верификации и валидации, тестирование на основе моделей, методы анализа требований, операционная система Linux.

Alexey Vladimirovich KHOROSHILOV, Leading Researcher, Ph.D. in Physics and Mathematics, Director of the Linux OS Verification Center at ISP RAS, Associate Professor of System Programming Departments at Moscow State University and the Higher School of Economics. Main research interests: design and development methods for critical systems, formal methods of software engineering, verification and validation methods, model-based testing, requirements analysis methods, Linux operating system.

Юрий Алексеевич СОЛОДЕЛОВ, начальник сектора, окончил Московский авиационный институт в 2009 году. Научные интересы: операционные системы реального времени, разработка сертифицируемого бортового ПО.

Yuri Alekseevich SOLODELOV, Head of Sector, graduated from Moscow Aviation Institute in 2009. Research interests: real-time operating systems, development of certified on-board software.

Алексей Геннадьевич ВОЛОБОЙ, ведущий научный сотрудник, доктор физико-математических наук, доцент. Научные интересы: компьютерная графика, оптика, оптическое моделирование.

Alexey Gennadievich VOLOBOY, Leading Researcher, Doctor of Physical and Mathematical Sciences, Associate Professor. Research interests: computer graphics, optics, optical modeling.

Владимир Александрович ГАЛАКТИОНОВ, главный научный сотрудник, доктор физико-математических наук, профессор. Научные интересы: компьютерная графика, вычислительная оптика, компьютерная лингвистика, научная визуализация.

Vladimir Alexandrovich GALAKTIONOV, Chief Researcher, Doctor of Physical and Mathematical Sciences, Professor. Research interests: computer graphics, computational optics, computer linguistics, scientific visualization.

Игорь Викторович КОВЕРНИНСКИЙ, заместитель начальника отделения, окончил МФТИ в 1972 г. Научные интересы: интегрированная модульная авионика.

Igor Viktorovich KOVERNINSKY, deputy head of department, graduated from Moscow Institute of Physics and Technology in 1972. Research interests: integrated modular avionics.