ПРОГРАММИРОВАНИЕ, 2022, № 3, с. 3–10

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.92

ЭФФЕКТИВНАЯ ВИЗУАЛИЗАЦИЯ ДЛЯ СИСТЕМЫ ОТОБРАЖЕНИЯ В КАБИНЕ САМОЛЕТА ПО СТАНДАРТУ ARINC 661

© 2022 г. Б. Х. Барладян[a] (ORCID: 0000-0002-2391-2067), Л. З. Шапиро[a] (ORCID: 0000-0002-6350-851X), Н. Б. Дерябин[a] (ORCID: 0000-0003-1248-6047), Ю. А. Солоделов[b] (ORCID: 0000-0001-5891-7645), А. Г. Волобой[a,*] (ORCID: 0000-0003-1252-8294), В. А. Галактионов[a] (ORCID: 0000-0001-6460-7539)

[a] *Институт прикладной математики им. М.В. Келдыша РАН*
125047 Москва, Миусская пл., 4, Россия

[b] *Государственный научно-исследовательский институт авиационных систем*
125167 Москва, ул. Викторенко, 7, Россия

*****E-mail: voloboy@gin.keldysh.ru*

Поступила в редакцию 26.12.2021 г.
После доработки 10.01.2022 г.
Принята к публикации 17.01.2022 г.

Программное обеспечение, используемое в авионике, должно соответствовать строгим авиационным стандартам. Авиационный стандарт ARINC 661 определяет интерфейсы и ряд специфических требований к системе отображения информации в кабине пилота. В силу этой специфики возникла проблема достижения приемлемой скорости визуализации на перспективной платформе i.MX6 с пониженным энергопотреблением. В работе рассматривается разработка графической библиотеки OpenGL SC (Safety Critical), работающей в авиационной операционной системе реального времени JetOS и решающей данную проблему. Она позволяет отображать информацию, сформированную в соответствии со стандартом ARINC 661, с использованием аппаратной поддержки графического процессора Vivante. Предложен и реализован эффективный подход ускорения визуализации. В первую очередь были оптимизированы вызовы OpenGL в системе отображения информации, работающей в соответствии со стандартом. Однако такая модификация требует значительных затрат при сертификации всей системы отображения. Поэтому необходимая оптимизация была вынесена в отдельный промежуточный модуль. Предлагаемый подход позволяет достичь скорости визуализации, приемлемой для дисплея пилота в кабине экипажа.

DOI: 10.31857/S0132347422030025

1. ВВЕДЕНИЕ

Система отображения в кабине экипажа (CDS – Cockpit display system) предоставляет видимую и звуковую информацию о воздушном судне и окружающей среде и получает команды управления от экипажа. Таким образом, экипаж управляет самолетом посредством современной, так называемой “стеклянной кабины” и взаимодействует с бортовым радиоэлектронным оборудованием. Интерфейсы между CDS и пользовательскими приложениями (UA – User application) должны соответствовать авиационному стандарту ARINC 661 [1]. Основная цель интерфейса – минимизировать затраты при поддержке и развитии систем управления самолетом, добавлении новых функций отображения в кабине экипажа в течение срока службы самолета и модернизации устаревающих аппаратных средств в области быстро развивающихся технологий.

CDS предоставляет графические и интерактивные сервисы в кабине пилота для пользовательских приложений (UA). Программное ядро CDS в литературе часто называется “сервером ARINC 661”, подчеркивая важность соответствия стандарту. Приложение отправляет графическую информацию в CDS, где она визуализируется. Экипаж анализирует ее и контролирует поведение приложения с помощью команд, отправляемых от CDS к UA. Обмен данными между UA и CDS осуществляется через сеть.

Архитектура системы визуализации CDS должна быть отказоустойчивой, обладать достаточной надёжностью и поддерживать различные типы дисплеев, типичными примерами которых являются основной пилотажный дисплей, навигационный дисплей и дисплей на лобовом стекле. Очевидно, что скорость визуализации дисплея пилота должна быть близкой к отображению в реальном времени, чтобы обеспечить надежный и своевременный контроль над летательным аппаратом.

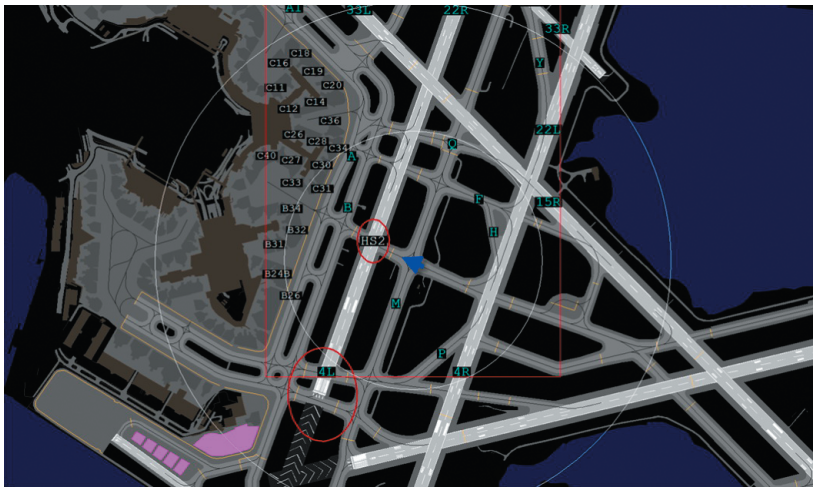

Рис. 1. Пример карты движения по аэродрому.

Одним из часто используемых приложений является карта движения по аэродрому (AMM – Airport Movement Map) [2], на которой отображается текущее местоположение самолета на рулежных дорожках аэропорта (рис. 1).

Одной из крупнейших компаний, разрабатывающей как собственный сервер ARINC 661, так и продукты для создания приложений, полностью соответствующих стандарту ARINC 661, является Ansys [3]. Многие разработчики авиационных приложений берут продукты компании Ansys за основу. Сервер ARINC 661, созданный этой компанией, для визуализации информации использует библиотеку OpenGL SC (Safety Critical), которая рекомендована для применения в авионике. Мы исследовали эффективность ее использования для приложения AMM под управлением авиационной операционной системы реального времени JetOS [4]. В исследованиях использовалась библиотека OpenGL SC 1.0.1 для перспективной авиационной платформы i.MX6 с поддержкой аппаратного ускорения на GPU Vivante [5]. Скорость визуализации для относительно простых тестов составила 2-3 кадра в секунду. Такая скорость визуализации недостаточна для отображения информации на дисплее пилота.

2. СУЩЕСТВУЮЩИЕ РАБОТЫ

Скорость визуализации пользовательских приложений на дисплее пилота критична. Большое количество работ посвящено проектированию, тестированию и ускорению CDS-интерфейсов и визуализации.

Несколько статей посвящены тестированию интерфейса CDS-UA. Интерфейс в первую очередь влияет на скорость визуализации дисплея и скорость реакции пилота на сообщаемую ситуацию. В статьях [6, 7] предлагается подход к автоматизации тестирования системы CDS. Предлагаемый подход проверяет CDS на двух уровнях: на системном уровне для проверки правильности работы CDS и на уровне системной интеграции CDS, когда они интегрированы с различными компонентами авионики. В рамках предложенного подхода авторы разрабатывают модели различных элементов CDS, которые затем используются для поддержки процесса автоматизированного тестирования. Этот подход к тестированию направлен на оценку того, правильно ли информация из пользовательских приложений отображается на CDS. Для этого авторы захватывают детали различных виджетов экранов дисплея и анализируют их на основе стандарта ARINC 661 для систем отображения информации в кабине экипажа. Таким способом оценивается ожидаемое поведение CDS, видимое на экранах самолета. В статье [8] предлагается эффективный подход к мониторингу и анализу сообщений ARINC 661 в реальном времени.

В другой статье представлена виртуальная платформа для эффективной и надежной разработки системы CDS [9]. С помощью этой платформы системы виртуального отображения для различных самолетов могут быть построены быстро в соответствии с требованиями пользователя, а соответствующие интерфейсы передачи данных могут создаваться динамически, что может решить проблему разработки CDS для одного конкретного типа самолета.

Работы [10] и [11] посвящены вопросам скорости визуализации. В этих работах предлагается использовать библиотеку OpenVG для визуализации вместо использования библиотеки OpenGL. OpenVG – это стандартный API двумерной векторной графики, созданный Khronos Group. Он

Рис. 2. AMM приложение. Треугольная сетка.

был разработан для рендеринга встроенного системного графического интерфейса пользователя, и его функции подходят для реализации большинства виджетов ARINC 661. Тем не менее, в нашем случае этот подход неприменим, поскольку сервер ARINC 661, разработанный компанией Ansys, предполагает использование именно библиотеки OpenGL, включая ее трехмерные возможности.

3. СПЕЦИФИКА ВИЗУАЛИЗАЦИИ ПО СТАНДАРТУ ARINC 661

Для анализа причин низкой скорости визуализации мы использовали три относительно простых примера геометрии. Все они были построены на основе приложения AMM (карта движения по аэродрому) аэропорта Шереметьево (Москва). Рулежные дорожки здесь представлены отрезками линий. Окружающая геометрия представлена набором треугольников.

Исследовалась скорость визуализации для трех вариантов приложения с соответствующей геометрией. Первый вариант приложения (рис. 2) содержит только геометрию, представленную треугольной сеткой.

Второй вариант приложения (рис. 3) содержит только рулежные дорожки, представленные линиями из отрезков. И третий вариант приложения (рис. 4) представляет собой карту рулежных дорожек и окружающей геометрии. Кнопки внизу экрана на этих рисунках позволяют управлять уровнем детализации визуализируемых объектов и их поворотом.

Эти три тестовых примера позволили нам понять основные проблемы визуализации приложения AMM сервером ARINC 661, созданном компанией Ansys, на платформе i.MX6 с аппаратным ускорением. Прямое использование библиотеки OpenGL SC для этих тестовых приложений приводит к слишком низкой скорости визуализации: 3 кадра в секунду для теста треугольной сетки, 2, 3 кадра в секунду для массива сегментов и 1,7 кадра в секунду для их комбинации.

Причиной низкой производительности, выявленной при трассировке кода, является неэффективное использование библиотеки OpenGL SC с аппаратной поддержкой. Эта неэффективность обусловлена двумя причинами:

1. Каждый отрезок линии и каждый треугольник треугольной сетки визуализируется в сервере отдельным набором команд библиотеки OGLX, как показано в таблице 1.

2. Сервер ARINC 661 в соответствии со стандартом использует гало (halo) для улучшения видимости графических примитивов. Гало – это полное оконтуривание виджета типа Gp (графические примитивы) линией контрастного цвета (обычно чёрной) для улучшения его видимости. В нашем случае использование гало приводит к тому, что в изображаемой геометрии каждый отрезок дополняется еще пятью, а каждый тре-

Табл. 1. Набор команд для визуализации отрезков и треугольников в сервере ARINC 661

Отрезок	Треугольник
sglBegin(SGL_LINES)	sglBegin(SGL_POLYGON)
sglVertex2f(x1, y1)	sglVertex2f(x1, y1);
sglVertex2f(x2, y2)	sglVertex2f(x2, y2);
sglEnd()	sglVertex2f(x3, y3);
	sglEnd()

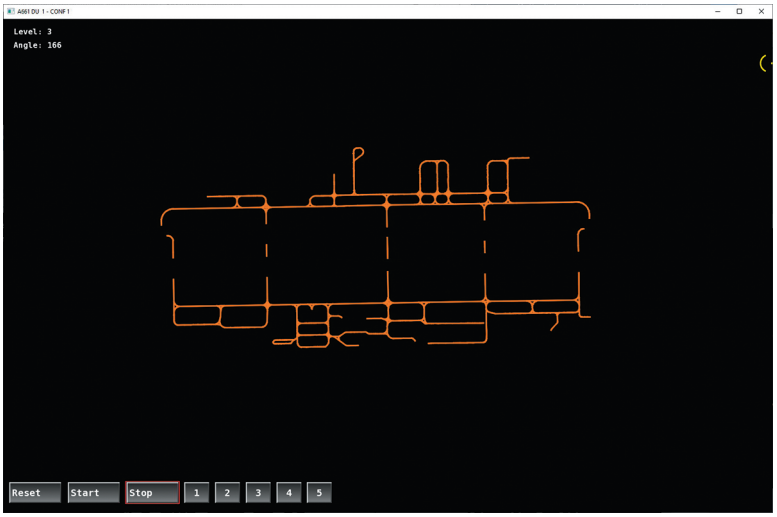

Рис. 3. AMM приложение. Линии рулежных дорожек.

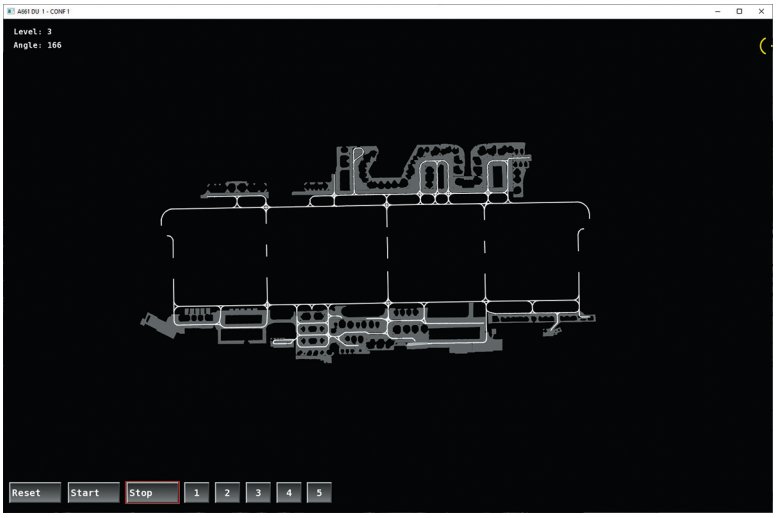

Рис. 4. AMM приложение. Треугольная сетка и линии рулежных дорожек.

угольник тремя специально расположенными отрезками. В результате для рассматриваемых тестовых примеров для визуализации линий (рис. 3) используется 5031 вызывов функции рисования *glDrawArrays()*, а для визуализации треугольников (рис. 2) – 7381 таких вызывов на кадр. Каждый вызов функции *glDrawArrays()* “ломает” конвейер OpenGL, данные переписываются из памяти CPU в память GPU, что и приводит к деградации производительности библиотеки OpenGL с поддержкой аппаратного ускорения. В этом случае практически сводится на нет преимущество аппаратного ускорения по сравнению с программной реализацией библиотеки, не использующей GPU.

Важность использования гало при визуализации геометрии в кабине пилотов видна из сравнения изображений на рисунках 5 и 6, где в первом случае гало использовалось, а во втором нет. Без использования гало часть геометрии, необходимой летчикам для правильной ориентации, пропадает. Это хорошо заметно на изображении взлетно-посадочной полосы в центре.

Таким образом, поддержка гало является важной и необходимой функциональностью в соответствии со стандартом ARINC 661.

4. ОПТИМИЗАЦИЯ ВЫЗОВОВ ФУНКЦИЙ OpenGL

Сервер ARINC 661, разработанный компанией Ansys, использует специальную промежуточную библиотеку OGLX для вызова необходимых функций библиотеки OpenGL. Первый вариант оптимизации был сделан непосредственно в биб-

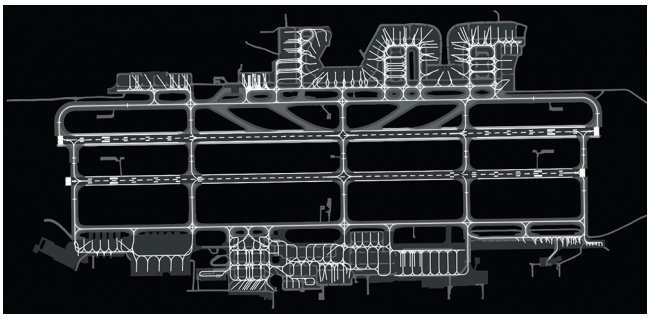

Рис. 5. Изображение аэропорта с использованием гало.

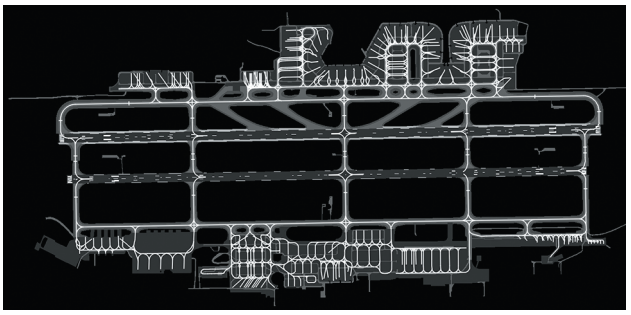

Рис. 6. Изображение аэропорта без использования гало.

лиотеке OGLX, которая визуализирует геометрии только посредством функции *glDrawArrays()*.

В функции *glDrawArrays()* используются следующие типы примитивов OpenGL: GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_TRIANGLES для рисования треугольников, и GL_LINE_STRIP, GL_LINE_LOOP и GL_LINES для рисования отрезков. Для уменьшения количество вызовов функции *glDrawArrays()*, мы используем в оптимизированной версии исключительно примитивы GL_TRIANGLES и GL_LINES. Другие типы примитивов конвертируются в них.

Кроме того, вместо установки текущего цвета с помощью функции *glColor4f()*, мы устанавливаем цвет в вершинах с помощью функции *glColorPointer()*. Для замены несколько вызовов *glDrawArrays()* на один, в мы объединяем массивы координат вершин и их атрибутов (цветов и координат текстур) для различных графических примитивов из нескольких последовательных вызовов функций рисования *glDrawArrays()*.

Объединение атрибутов вершин допустимо только в том случае, если последовательные вызовы функции рисования соответствуют одному и тому же состоянию OpenGL. Состояние определяется параметрами рисования OpenGL, которые были установлены при вызове функции *glDrawArrays()*. Среди параметров, определяющих состояние OpenGL в библиотеке OGLX, есть тип текущего примитива (треугольник или отрезок), ширина линий, идентификатор используемой текстуры, маски и функции трафаретов, и другие.

Алгоритм работы оптимизированной библиотеки OGLX можно теперь описать следующим образом:

1. Вызов таких функций OpenGL, как *glClearColor()*, *glStencilMask()*, *glClearStencil()*, *glClear()*, *glColor4f()*, *glEnable()*, *glDisable()*, *glEnableClientState()*, *glDisableClientState()*, *glLinewidth()*, *glStencilOp()*, *glStencilFunc()*, *glViewport()*, *glScissor()*, *glBindTexture()* заменяются соответствующими функциями с префиксом *Set*, то есть *SetClearColor()*, *SetStencilMask()*, *SetClearStencil()* и т.д. Эти функции вместо передачи параметров в OpenGL сохраняют их в специальной внутренней структуре. Эта структура, в частности, определяет текущее состояние OpenGL.

2. Вызов функции *glDrawArrays()* в библиотеке OGLX заменяется разработанной нами функцией *ProcessPrimitives()*. При ее вызове сравниваются параметры предыдущего и текущего состояний OpenGL. Если эти состояния совпадают (или это первый вызов функции *ProcessPrimitives()*), то данные рисования (координаты вершин и текстуры, цвета вершин) добавляются к уже сохраненным. Использование функции *glLoadMatrix()* требует отдельных вызовов функции *glDrawArrays()* для разных матриц. Чтобы избежать этих многократных вызовов координаты вершины перед добавлением умножаются на текущую матрицу. Примитивы треугольников типа GL_TRIANGLE_STRIP и GL_TRIANGLE_FAN преобразуются к типу GL_TRIANGLES. Примитивы сегментов типа GL_LINE_STRIP и GL_LINE_LOOP преобразуются к типу GL_LINES. Это преобразование примитивов необходимо для обеспечения возможности объединения их при рисовании функцией *glDrawArrays()*.

3. Если предыдущее и текущее состояния OpenGL не совпадают, то накопленные данные, объединенные для рисования с необходимыми настройками в OpenGL из предыдущего состояния, визуализируются одним вызовом функции *glDrawArrays()*.

4. В OpenGL устанавливаются только те параметры OpenGL, которые изменились по сравнению с предыдущим состоянием.

5. Массив координат и цветов вершин устанавливается в OpenGL с помощью функций *glEnableClientState()*, *glVertexPointer()* и *glColorPointer()*.

6. Если визуализируются текстурированные треугольники, то массив координат текстуры вершин задается с помощью функций *glEnable (GL_TEXTURE_2D)*, *glEnableClientState (GL_TEXTURE_COORD_ARRAY)* и *glTexCoordPointer()*. Если треугольники не текстурированы или рисуются сегменты, то перед вызовом *glDrawArrays()* вызываются функции

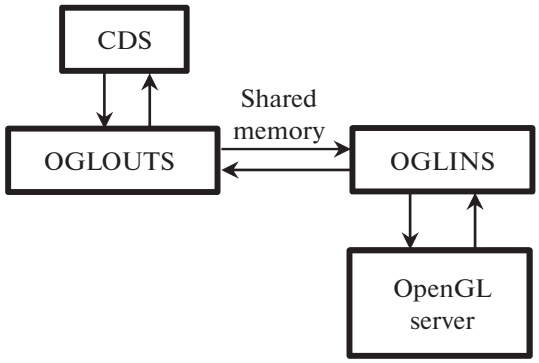

Рис. 7. Схема взаимодействия между CDS и сервером OpenGL.

glDisableClientState (GL_TEXTURE_COORD_ARRAY) и *glDisable (GL_TEXTURE_2D)*.

7. Функция *glDrawArrays()* вызывается соответственно с параметром GL_TRIANGLES или GL_LINES.

Следует отметить, что несколько функций OpenGL, таких как *glViewport()*, *glGenTextures()* и *glTexImage2D()* вызываются из библиотеки OGLX только один раз во время ее инициализации. Вызовы этих функций не изменяются.

В результате описанной оптимизации количество вызовов функции *glDrawArrays()* для приложения на рис. 2 (треугольная сетка) уменьшилось с 7398 до 46, для приложения с рулежными дорожками на рис. 3 – с 5049 до 46 и для приложения на рис. 4 (треугольники и отрезки) с 12436 до 47. Соответственно скорость была увеличена до 24.1 кадра в секунду для первого приложения, до 21.2 для второго и до 15.2 кадра в секунду для третьего приложения.

5. OPENGL СЕРВЕР

Решение, описанное в предыдущем разделе, обеспечивает приемлемую скорость визуализации для тестовых приложений с использованием сервера ARINC 661. Но у него есть существенный недостаток: оно требует заметных изменений в библиотеке OGLX, которая входит в состав сервера. Таким образом, это потребует новой сертификации сервера ARINC 661, изначально созданного компанией Ansys, что значительно увеличит сроки и затраты на сертификацию программного обеспечения авионики. Более рационально выделить реализацию разработанного метода оптимизации в отдельный модуль. Разумно поместить этот код непосредственно в библиотеку OpenGL или на промежуточном уровне между библиотекой и сервером ARINC 661. Базовая операционная система реального времени JetOS позволяет использовать многоядерность конкретного процессора. Поэтому для эффективности мы реализовали библиотеку OpenGL как модуль асимметричной многопроцессорной обработки (AMP) в терминах JetOS.

5.1. Реализация сервера OpenGL

Для многоядерной системы JetOS поддерживает возможность запуска нескольких модулей (или экземпляров JetOS) на одном устройстве. Эти модули работают независимо на разных ядрах процессора. Эта функция в JetOS называется асимметричной многопроцессорной обработкой (AMP). Это расширение стандарта ARINC 653 позволяет более эффективно использовать ресурсы процессора. Мы используем эту технологию для запуска библиотеки OpenGL на отдельном ядре процессора в качестве сервера OpenGL.

Для взаимодействия между системой отображения в кабине экипажа (CDS) и сервером OpenGL нами были введены две промежуточные библиотеки (рис. 7). Первая – это OGLOUTS, которая содержит аналоги вызовов OpenGL, но не обращается к ней, а только записывает последовательность вызовов и их параметры в память. Вторая – OGLINS, которая берет информацию из памяти и вызывает непосредственно функции сервера OpenGL. Взаимодействие реализовано через память, общую для CDS и сервера OpenGL.

В CDS все используемые функции OpenGL заменены на функции, реализованные в библиотеке OGLOUTS с тем же интерфейсом. Никаких изменений непосредственно в CDS делать не надо. Эти функции записывают их идентификаторы и все параметры в массивы в общей памяти, доступной для CDS и сервера OpenGL. Их фактическое выполнение начнется только после вызова функции *SwapBuffers()*, которая начинает рендеринг всего кадра.

Сервер OpenGL обрабатывает данные, подготовленные в общей памяти, с помощью специальной библиотеки OGLINS. Функции в этой библиотеке считывают необходимые данные из общей памяти, извлекают соответствующий идентификатор функции и ее параметры, записанные в общую память, и вызывают соответствующую функцию OpenGL.

Во время инициализации CDS использует несколько функций, которые должны выполняться немедленно. Это функции *glViewport()*, *glGenTextures()*, *glBindTexture()* и *glTexImage2D()*. В случае функции *glGenTexture()*, результат должен быть сразу возвращен в CDS для будущего использования функцией *glBindTexture()*. Это обеспечивается механизмом синхронизации. Возврат необходимых значений также реализуется через общую память.

Когда все данные, необходимые для визуализации одного кадра (параметры геометрии, различные атрибуты, параметры состояния OpenGL

Табл. 2. Скорость визуализации с использованием оригинальной библиотеки OGLX и двух предложенных подходов оптимизации. Скорость указана в кадрах в секунду

Тест/подход	Исходная OGLX	Оптимизированная OGLX	сервер OpenGL
Треугольная сетка (рис. 2)	3	24.1	20.2
Линии рулежных дорожек (рис. 3)	2.3	21.2	21.8
Линии рулежных дорожек и треугольная сетка (рис. 4)	1.7	15.2	15.2

и соответствующая последовательность вызовов функций OpenGL), подготовлены и вызывается функция *SwapBuffers()*, то механизм синхронизации запускает сервер OpenGL. Он обрабатывает все данные, подготовленные в общей памяти, и выводит построенное изображение данного кадра на экран дисплея.

5.2. Синхронизация между CDS и сервером OpenGL

Синхронизация работы CDS и сервера OpenGL выполняется с помощью специальных объектов, называемых событиями, которые реализуются через небольшие блоки общей памяти между модулями. В нашем случае используются два события:

событие StartOgl – устанавливается в сигнальное состояние системой CDS, когда данные, подготовленные в общей памяти, готовы к обработке сервером OpenGL;

событие EndOgl – устанавливается в сигнальное состояние сервером OpenGL, когда данные обработаны, кадр выведен на экран дисплея и сервер OpenGL готов обработать следующую порцию данных.

Первоначально событие StartOgl устанавливается в несигнальное состояние, а EndOgl устанавливается в сигнальное состояние.

Синхронизация на уровне CDS реализована в библиотеке OGLOUTS. Следует отметить, что синхронизация необходима только для функций, которые требуют немедленного выполнения на сервере OpenGL. Это функции, используемые на этапе инициализации и функция *SwapBuffers()*, которая вызывается для завершения рендеринга кадра. Остальные функции библиотеки OGLOUTS записывают результаты в общую память вместо использования внутренних массивов библиотеки OGLX.

Каждая функция из библиотеки OGLOUTS имеет имя и интерфейс такие же, как и у соответствующей функции из стандарта OpenGL. Идентификатор функции и все переданные параметры сохраняются в соответствующих массивах в общей памяти. Если необходимо вернуть данные (для функции *glGenTextures()*), то они извлекаются из общей памяти. Функция ждет, пока событие EndOgl перейдет в сигнальное состояние. Это означает, что предыдущая функция уже была выполнена сервером OpenGL. После чего событие EndOgl устанавливается в несигнальное состояние, а событие StartOgl устанавливается в сигнальное состояние. В случае функции *SwapBuffers()* весь набор вызовов функций OpenGL с соответствующими данными будет передан серверу OpenGL для выполнения рендеринга одного кадра. Массивы данных, устанавливаемые функциями *glVertexPointer()*, *glColorPointer()* и *glTexCoordPointer()*, также передаются в специальных массивах с использованием общей памяти.

Синхронизация на уровне сервера OpenGL относительно проста и представляется следующим алгоритмом:

1. Ждем, пока событие StartOgl перейдет в сигнальное состояние.

2. Устанавливаем событие StartOgl в не сигнальное состояние.

3. Вызываем функцию *process_all_ogl_commands()* для обработки подготовленных в общей памяти команд.

4. Устанавливаем событие EndOgl в сигнальное состояние и возвращаемся к пункту 1.

Функция *process_all_ogl_commands()* последовательно обрабатывает все функции OpenGL, подготовленные для выполнения в общей памяти, одну за другой. Соответствующий массив содержит идентификаторы функций и основные параметры. Окончание обработки в этом массиве обозначается специальным идентификатором функции. Следует отметить, что в случае нескольких функций, вызываемых во время инициализации и требующих немедленного выполнения, сервер OpenGL обработает их за один вызов. Во время обработки функции *SwapBuffers()* будет выполнен весь набор функций OpenGL, необходимых для визуализации данного кадра.

6. РЕЗУЛЬТАТЫ

Скорость визуализации с использованием оригинальной библиотеки OGLX и двух предложенных подходов представлена в таблице 2. Значения указаны в кадрах в секунду.

Из таблицы видно, что оба предложенных подхода к оптимизации системы отображения в кабине экипажа позволяют достичь результатов, приемлемых по скорости для использования в авиационных приложениях. Однако подход с использованием сервера OpenGL более предпочтителен как с точки зрения более простого процесса сертификации, так и дальнейшей поддержки, поскольку дополнительный код изолирован и от библиотеки OGLX, используемой сервером ARINC 661, и от библиотеки OpenGL.

Результаты исследования были также доложены на ежегодной международной конференции по компьютерной графике и зрению Графикон [12].

7. ЗАКЛЮЧЕНИЕ

При разработке любого приложения, предназначенного для использования в авионике, необходимо с одной стороны следовать достаточно строгим стандартам, принятым в авиации, а с другой – ориентироваться на вычислительные платформы с пониженным энергопотреблением и невысокой производительностью. Сочетание этих факторов определяет сложность такой работы.

Нами было разработано два подхода, позволяющих оптимизировать процесс визуализации при сохранении специфики стандарта ARINC 661, используемом в гражданской авиации. В результате удалось обеспечить приемлемую скорость рендеринга для приложений, использующих ресурсозатратный эффект гало, который обеспечивают лучшую читаемость виджетов на дисплее пилота. Разработка была сделана и протестирована на перспективной платформе i.MX6 с графическим ускорителем Vivante под операционной системой реального времени JetOS.

СПИСОК ЛИТЕРАТУРЫ

1. *E. Barboni, S. Conversy, D. Navarre, P. Palanque*, Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification. In: Doherty G., Blandford A. (eds) Interactive Systems. Design, Specification, and Verification. DSV-IS 2006. volume 4323 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2007. https://doi.org/10.1007/978-3-540-69554-7_3
2. GE Aviation. Airport Surface Moving Map, 2021. URL: https://www.geaviation.com/systems/avionics/navigation-guidance/airport-surface-moving-map
3. Ansys SCADE Solutions for ARINC 661 Compliant Systems, 2021. URL: https://www.ansys.com/products/embedded-software/solutions-for-arinc-661
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, vol. 28, issue 2, 2016, pp. 181–192. https://doi.org/10.15514/ISPRAS-2016-28(2)-12
5. *Barladian B.Kh., Deryabin N.B., Voloboy A.G., Galaktionov V.A., Shapiro L.Z.*, High speed visualization in the JetOS aviation operating system using hardware acceleration, Proceedings of the Graphicon 2020 conference, CEUR Workshop Proceedings, vol. 2744, 2020, pp. short3:1-short3:9, https://doi.org/10.51130/graphicon-2020-2-4-3
6. *Sartaj H., Iqbal M.Z., Khan M.U.*, Testing cockpit display systems of aircraft using a modelbased approach. Software and Systems Modeling, 20, 2021, pp. 1977–2002. https://doi.org/10.1007/s10270-020-00844-z
7. *Iqbal M.Z., Sartaj H., Khan M.U., Haq F.U., Qaisar I.*, A Model-Based Testing Approach for Cockpit Display Systems of Avionics, ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), 2019, pp. 67–77, https://doi.org/10.1109/MODELS.2019.00-14
8. *Yang W., Shen X., Qiu Q., Zhang J., Cais Y.*, An Efficient Approach for Monitoring and Analyzing Real-Time ARINC 661 Events, 2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC), 2018, pp. 1–5, https://doi.org/10.1109/DASC.2018.8569329
9. *Zheng Y. and Lei X.Y.*, Research and Implementation of Virtual Cockpit Panel Development Platform Based on ARINC 661. Proceedings of the IEEE Chinese Guidance, Navigation and Control Conference (CGNCC), 2014, pp. 1357–1361.
10. *Yoon J., Baek, N. and Lee, H.*, Graphics Rendering Based on OpenVG and Its Use Cases with Wireless Communications. Wireless Personal Communications, 94(2), 2017, 175–185. https://doi.org/10.1007/s11277-015-3163-y
11. *Yoon J., Baek, N. and Lee H.*, ARINC661 graphics rendering based on OpenVG, Proceedings of the 5th International Conference on IT Convergence and Security (ICITCS), Kuala Lumpur, Malaysia, Aug 2015.
12. *Barladian B., Shapiro L., Deryabin N., Solodelov Y., Voloboy A., Galaktionov V.* Optimizing ARINC 661 Rendering for OpenGL with Hardware Support in the JetOS Aviation Operating System, Proceedings of the Graphicon 2021 conference, CEUR Workshop Proceedings, vol. 3027, 2021, pp. 74–82, https://doi.org/10.20948/graphicon-2021-3027-74-82