УДК 004.454 Научная статья

https://doi.org/10.32603/2071-8985-2025-18-8-68-74

Методы повышения производительности работы с графическими данными во встраиваемых системах на базе Linux

К. В. Пугин✉, А. М. Гиацинтов, К. А. Мамросенко

Научно-исследовательский институт системных исследований Национального исследовательского центра «Курчатовский институт», Москва, Россия

✉ rilian@niisi.ras.ru

Аннотация. Рассматриваются вопросы работы подсистемы вывода на экран и подсистемы графического ускорения ОС Linux. Описаны методы для повышения пропускной способности памяти на встраиваемых системах. Авторами предложен новый метод минимизации требуемой пропускной способности памяти за счет использования метаданных об изображении и сжатия при передаче данных между ОЗУ, контроллером вывода на экран и GPU. В целях тестирования производительности в части работы с графическими данными модифицирован графический стек Linux с использованием разработанного метода. Измерения проводились на встраиваемой системе на примере 3D-приложения для нескольких случаев – как с использованием X-сервера, так и без такового. Полученные результаты позволят повысить производительность на встраиваемых системах в части работы с графическими данными в условиях ограниченной пропускной способности канала между графическими компонентами и ОЗУ.

Ключевые слова: встраиваемые системы, производительность, графический ускоритель, драйверы графического стека

Для цитирования: Пугин К. В., Гиацинтов А. М., Мамросенко К. А. Методы повышения производительности работы с графическими данными во встраиваемых системах на базе Linux // Изв. СПбГЭТУ «ЛЭТИ». 2025. Т. 18, № 8. С. 68–74. doi: 10.32603/2071-8985-2025-18-8-68-74.

Конфликт интересов. Авторы заявляют об отсутствии конфликта интересов.

Финансирование: Публикация выполнена в рамках НИР НИЦ «Курчатовский институт» – НИИСИ по теме № FNEF-2024-0003 «Методы разработки аппаратно-программных платформ на основе защищенных и устойчивых к сбоям систем на кристалле и сопроцессоров искусственного интеллекта и обработки сигналов».

Original article

Methods for Improving Performance when Working with Graphic Data in Embedded Linux-Based Systems

K. V. Pugin✉, A. M. Giatsintov, K. A. Mamrosenko

Scientific Research Institute for System Analysis of the National Research Center «Kurchatov Insitute», Moscow, Russia

✉ rilian@niisi.ras.ru

Abstract. This article is concerned with the operation of the Linux display and graphics acceleration subsystems. It also discusses the methods for increasing memory bandwidth on embedded systems. The authors propose a new method for minimizing the required memory bandwidth by using image metadata and compression when transferring data between RAM, the display controller, and the GPU. To test the performance in terms of working with graphic data, the Linux graphic stack was modifed using the developed method. Measurements were made for several cases on the embedded system using a 3D application both with and without the X server. The obtained results will allow increasing the performance on embedded systems when working with graphic data under conditions of limited channel bandwidth between graphics components and RAM.

Keywords: embedded systems, performance, graphics accelerator, graphics stack drivers

For citation: Pugin K. V., Giatsintov A. M., Mamrosenko K. A. Methods for Improving Performance when Working with Graphic Data in Embedded Linux-Based Systems // LETI Transactions on Electrical Engineering & Computer Science. 2025. Vol. 18, no. 8. P. 68–74. doi: 10.32603/2071-8985-2025-18-8-68-74.

Conflict of interest. The authors declare no conflicts of interest.

Financing: Publication was carried out as part of the research for NRC «Kurchatov Institute» – SRISA under the topic no. FNEF-2024-0003 «Methods for Developing Hardware and Software Platforms Based on Secure and Fault-Tolerant Systems-on-Chip and Artificial Intelligence and Signal Processing Co-Processors».

Введение. Графические данные в современных условиях требуют достаточно большого для встраиваемых систем объема оперативной памяти (десятков мегабайт) и скорости обработки (для обеспечения частоты обновления не менее 30 Гц). Во встраиваемых системах в большинстве случаев имеются ограничения по размеру, скорости и пропускной способности памяти. Поэтому исследования, результатом которых станет сокращение размеров хранимых данных либо снижение объема передаваемых данных за единицу времени, остаются актуальными. В рамках данной статьи авторы пытаются ответить на вопрос, каким образом можно снизить объем передаваемых данных между контроллером вывода на экран и графическим ускорителем в наиболее вероятных во встраиваемых системах сценариях работы. Авторами предложен новый метод минимизации требуемой пропускной способности памяти за счет использования метаданных об изображении и сжатия при передаче данных между ОЗУ, контроллером вывода на экран и GPU (Graphics Processing Unit – графическим ускорителем). В статье описаны уже имеющиеся работы по схожим вопросам, проведен обзор того, как выполняется работа с графическими данными в Linux, а также приведено общее описание примененного метода и приведены экспериментальные результаты его применения.

1. Анализ предшествующих работ. Одними из первых сжатие фреймбуфера предложили авторы в [1], их алгоритм позволил увеличить пропускную способность памяти на 31 % на видео. В [2] исследователи выдвигают идею, что за уменьшение производительности графических систем может быть ответственна низкая скорость памяти, а также низкая пропускная способность каналов доступа к ней как по их числу, так и по скорости передачи данных. В дальнейшем другие исследователи в [3] рассматривают влияние параллелизации на изменение производительности при условии низких скоростей доступа к памяти с помощью построения нескольких моделей.

Применительно к мультимедиа данную задачу исследовали в [4]. В приведенной статье описан один из алгоритмов компрессии, который может быть использован для передачи буферов между подсистемой вывода на экран и GPU, что позволяет сократить расход памяти и уменьшить занимаемую пропускную способность шины. Для эффективной работы предложенного алгоритма в части графики и вывода на экран необходимо применение техник, описанных в данной статье.

Как одна из техник оптимизации, возможность применения сжатия в GPU для уменьшения использования пропускной способности памяти упоминается в [5], на базе которой были разработаны наша методика и похожая на нее методика AFBC (Arm FrameBuffer Compression), которая применяется в графических контроллерах Mali [6]. При этом, в отличие от методов, применяемых в данной статье, в методике Mali GPU сначала проводит декомпрессию, а затем обрабатывает фреймы, тогда как данная методика предлагает распространять фреймы во внутреннем сжатом формате GPU. В отличие от предлагаемой оптимизации, AFBC хранит метаданные совместно с данными, поэтому пересылка чисто метаданных по шине без обновления основных данных затруднена.

2. Методы рендеринга и обработки графических данных в GPU под управлением ОС Linux. Под графическими данными будем понимать графические объекты, которые созданы при помощи API OpenGL и Vulkan – буферы, текстуры.

Методы рендеринга в целом делятся на Immediate mode и Tile mode [7]. GPU на персональных компьютерах чаще всего используют Immediate mode, в котором рендеринг происходит последовательно – полностью обрабатывается вся картинка, и только потом происходит переход на следующий шаг. С другой стороны, большинство мобильных GPU (Arm, PowerVR, Imagination) используют Tile mode. В отличие от предыдущего режима, в этом режиме GPU разделяет картинку

на небольшие фрагменты, называемые тайлами. Рендер в тайловых GPU происходит в два шага: на первом выполняются все этапы конвейера прорисовки до растеризации для каждого примитива, и одновременно с этим составляется список тайлов, которые необходимо растеризовать, т. е. тех, на которых присутствуют примитивы. На втором шаге фрагментный шейдер выполняется только для тех тайлов, на которых находятся примитивы, запрошенные к прорисовке. Некоторую часть информации о тайлах иногда GPU хранят в метаданных для ускорения рендеринга. Тайловый способ рендеринга позволяет уменьшить требования к пропускной способности памяти в ходе выполнения фрагментного шейдера.

Данные в графических ускорителях хранятся в различных форматах. Формат, который может быть применен во всех графических ускорителях, – это линейный формат изображения, в котором пиксели хранятся последовательно и покоординатно (левый верхний угол изображения в данной системе имеет координаты [0, 0], затем в памяти хранится вся первая линия изображения, затем – вся вторая, и т. д.). Данный формат удобен для чтения и прорисовки при помощи CPU. Для использования в GPU он, зачастую, малоприменим вследствие того, что GPU оптимизирован для работы с более эффективными внутренними форматами, которые различаются от производителя к производителю. Некоторые внутренние форматы в GPU, например сжатые внутренним алгоритмом, могут требовать выделения дополнительной памяти для хранения метаданных независимо от области данных. При этом использование метаданных может очень сильно ускорять обработку данных в GPU за счет быстрой реконструкции обновляемых областей в ходе тайлового рендеринга (подробнее см. в п. 3). В большинстве встраиваемых систем графический ускоритель и контроллер вывода на экран – отдельные устройства, которые могут быть разработаны разными производителями, что отличает их от настольных компьютеров, где оба данных компонента в большинстве случаев находятся в составе видеокарты и управляются совместно. Взаимодействие контроллера вывода на экран и графического ускорителя происходит через стандартную системную шину, и данные зачастую передаются от устройства к устройству при помощи обычного DMA (Direct Memory Access – технология прямого доступа к оперативной памяти, минуя центральный процессор). На встраиваемых системах зачастую нет отдельной видеопамяти либо она используется только ускорителем. В связи с этим становится критичной скорость записи и чтения данных из ОЗУ для графического ускорителя и контроллера вывода на экран, а также изменение формата данных для оптимизации работы с ОЗУ.

В подавляющем большинстве дистрибутивов Linux для работы с графическими ускорителями при использовании графических интерфейсов типа «рабочий стол» применяется программный интерфейс API (Application Programming Interface) OpenGL (Open Graphics Library), за исключением некоторых композиторов на платформе wlroots, использующих Vulkan. Для унификации API и возможности работы с закрытыми библиотеками поддержки GPU, предоставляемых производителями, используется libglvnd, через которую происходит передача всех графических вызовов в нижележащие библиотеки драйверов. В библиотеках драйверов графические вызовы и графические данные преобразуются в объекты памяти, которые затем передаются при помощи Graphics Execution Manager (GEM) API в ядро Linux. В пространстве ядра данные объекты передаются драйвером устройства ускорителю для обработки. В ряде случаев результат обработки возвращается в пользовательское пространство. Если несколько пользовательских приложений используют одни и те же графические данные (например, приложение создает окно при помощи OpenGL, а затем X Server переносит его на экран), при запросе данных возможен их перенос между процессами без копирования. Данные в памяти могут храниться в различных форматах, которые зачастую не соответствуют друг другу. Direct Rendering Manager (DRM) Linux API предлагает определять форматы данных как идентификатор формата, описывающий используемые цветовые компоненты, и модификаторы, определяющие расположение цветовых компонентов в памяти. Некоторые идентификаторы форматов обязательны для реализации всеми производителями (например, формат ARGB8888), в то время как поддерживаемые модификаторы могут значительно различаться у разных производителей. Некоторые производители поддерживают несколько модификаторов форматов одновременно, например хранение данных в виде тайлов и сжатие данных (например, ускорители фирмы Arm [8]). Для этого в DRM преду-

смотрен механизм отдельной передачи областей памяти для метаданных, если такое использование модификаторов необходимо для работы с GPU.

Несмотря на то, что существуют различные способы вывода информации на экран, такие, как представленный в [9], большинство контроллеров вывода на экран используют механизм scanout (согласно [10]). Для scanout необходимо создание специальных буферов, которые будут передаваться в формате, требуемом контроллером вывода на экран, и окончательная их обработка (в частности, наложение аппаратного курсора) также будет проводиться вне GPU. Следовательно, зачастую необходимы специальные техники работы с такими изображениями как со стороны API графических ускорителей (например, для Vulkan существуют специальное расширение VK_KHR_display [11] для взаимодействия с контроллерами вывода на экран), так и со стороны контроллеров вывода на экран. Одно из известных аппаратных расширений – AFBC – описано в [6] совместно с применяемой им методикой оптимизации.

3. Методы оптимизации графической информации. С помощью модификаторов форматов и использования API DRM определенным образом можно применить несколько оптимизаций при разработке драйверов GPU и DC (Direct Connect) во встраиваемых системах. Первая оптимизация – межпроцессная передача данных GPU в том формате, в котором он сгенерирован. Это позволяет пользовательским приложениям напрямую передавать буферы, минуя все стадии промежуточной конвертации между форматами. Вторая оптимизация – использование для scanout того же модификатора хранения данных, что и GPU в ходе рендера. Это позволяет избежать конвертирования данных из формата, применяемого GPU, в используемый DC формат с возможным промежуточным шагом в виде CPU-формата. Третья оптимизация – возможность использова-

Результаты экспериментального исследования
Experimental results

Наименование теста	DRM						X Server			
	N	S	T	ST	TC	STC	N	S	T	ST
build	85	86	124	124	163	233	46	73	83	151
build (vbo)	98	99	158	157	208	607	50	78	100	153
texture (nearest)	92	93	134	134	170	395	47	72	88	136
texture (linear)	92	93	133	134	169	391	47	72	88	136
texture (mipmap)	94	94	137	136	173	426	47	73	89	137
shading (gouraud)	65	66	116	117	152	249	45	67	84	121
shading (blinn-phong-inf)	63	63	107	107	128	220	43	64	76	105
shading (phong)	63	63	81	81	91	135	38	53	61	79
shading (cel)	61	61	66	66	73	98	34	46	52	65
bump (high-poly)	71	70	89	89	102	159	40	57	66	87
bump (normals)	108	108	128	128	156	345	46	70	85	123
bump (height)	98	98	106	106	125	223	43	63	74	103
effect2d (0,1,0)	38	38	45	45	49	59	28	35	39	45
effect2d (1,1,1,1,1)	16	16	16	16	17	18	13	15	15	17
pulsar	61	60	104	104	130	250	42	61	75	109
desktop (blur)	18	18	23	23	24	28	18	20	21	24
desktop (shadow)	32	32	52	52	58	80	30	40	44	57
buffer (map)	20	20	11	11	12	12	32	33	37	39
buffer (subdata)	19	19	11	11	11	11	30	31	35	37
buffer (map,interleave)	24	24	25	25	26	26	33	42	51	61
ideas	35	35	42	42	44	46	34	40	43	46
jellyfish	16	16	42	42	46	58	26	33	36	45
terrain	3	3	3	3	3	3	3	3	3	3
shadow	53	53	75	75	89	132	37	51	60	80
refract	7	6	7	7	9	9	7	7	9	9
conditionals (0,0)	48	48	112	112	147	366	44	67	82	128
conditionals (5,0)	46	46	56	56	61	82	31	41	46	57
conditionals (0,5)	48	48	112	112	146	358	44	67	82	126
function (low)	47	47	76	76	83	136	37	52	59	79
function (medium)	45	45	49	49	53	68	29	38	41	50
loop (no-fragment-loop)	47	47	75	75	85	131	37	51	58	78
loop (no-fragment-uniform)	47	47	75	75	85	131	37	51	58	78
loop (fragment-uniform)	38	38	39	39	42	51	25	31	34	40
glmark2 Score:	51	51	73	73	88	167	34	48	56	78

ния метаданных GPU контроллером вывода на экран. Если такая возможность существует (например, контроллер вывода на экран и GPU от одного производителя), то пересылка метаданных совместно с данными может также дать прирост производительности.

Было проведено экспериментальное исследование изменений производительности графического стека Linux при условии применения оптимизаций, представленных в п. 2, как по отдельности, так и совместно. Для сравнения производительности использовалась исследовательская плата на архитектуре Little-Endian MIPS с 2D- и 3D-ядрами, работающая на CPU 600 МГц. Использовался тест glmark2 версии 21.12 для OpenGL ES на платформе X11 и на платформе DRM (без X-сервера). Графический стек проверялся с помощью Xorg 21.1.8. Все тестовые запуски проводились при разрешении экрана и приложения 1920 × 1080. Наименования тестовых запусков в таблице строятся следующим образом: индекс может состоять из нескольких букв, каждая из которых обозначает свою оптимизацию:

• N – оптимизации отсутствуют, драйверы идентичны применяемым в [12];

• S – применена возможность обмена ядерными данными между процессами графических приложений напрямую через драйвер GPU, минуя копирование;

• T – применена возможность обмена тайловыми данными между GPU и DC напрямую при помощи использования идентичной тайловой структуры на GPU и DC;

• C – применена возможность обмена сжатыми данными между GPU и DC напрямую при помощи идентичных алгоритмов сжатия на GPU и DC. На многих GPU невозможна без применения опции T, поскольку не все ускорители способны к сжатию линейных данных.

4. Обсуждение результатов. Как можно видеть, X Server при применении очень сильно влияет межпроцессная оптимизация – результаты в целом возрастают на 40 % только за счет данной техники. При этом при использовании моноприложения glmark2 эта оптимизация не дает прироста совсем – общий результат практически идентичен, что объясняется отсутствием необходимости межпроцессного взаимодействия в тесте такого рода. Влияние оптимизации формата хранения дает прирост около 64 % с использованием X Server, и на 43 % на моноприложении glmark2. Применение общих метаданных без первой оптимизации дает небольшой прирост на моноприложении glmark2 – 20 % относительно применения только оптимизации общего формата. При этом применение всех трех оптимизаций на моноприложении дает прирост производительности в 227 % по сравнению с применением общих метаданных. Исходя из данных результатов можно сделать вывод, что в определенных условиях (использование моноприложений) предложенные нами оптимизации позволяют в 2 раза повысить производительность графического стека. Также можно сделать вывод, что данные оптимизации также могут повысить производительность на 40–63 % для ряда операций с использованием X Server. При помощи данных подходов были снижены объемы передачи данных между процессами графического сервера и 3D-приложения. Для формирования итогового изображения графическим сервером, другим процессам достаточно передать графические метаданные. Поскольку графические метаданные имеют в несколько раз меньший объем, то операции копирования, занимающие значительную часть в общем количестве операций, выполняются значительно быстрее.

Выводы и заключение. Результаты данной статьи могут быть применены во встраиваемых системах с сходными параметрами аппаратного обеспечения (см. п. 3) для значительного повышения производительности в части работы с графическими данными. Поскольку во многих встраиваемых системах (например, в мониторах остановок или поездов метро) необходимо применение моноприложений, использование разработанного метода позволяет выбирать аппаратные компоненты с более низкой пиковой производительностью. Это становится возможным вследствие более эффективного использования пропускной способности памяти. В последующих исследованиях возможен более полный анализ применимости данных методов в различных встраиваемых системах для отображения видеоинформации – цифровых рекламных панелей или уличных мониторов слежения за трафиком. Возможно исследование применимости данного метода для прямого взаимодействия между блоками видеоинформации и контроллером вывода на экран или графическим ускорителем без использования ОЗУ. Также в дальнейшем необходимо определение граничных условий применимости метода на аппаратных платформах с различающимися частотами CPU и памяти, и влияние данных частот на изменение производительности.

Список литературы

1. Ma Z., Segall A. Frame buffer compression for low-power video coding // 18th IEEE Intern. Conf. on Image Proc. Brussels, Belgium: IEEE, 2011. P. 757–760. doi: 10.1109/ICIP.2011.6116665.

2. Sun X.-He, Liu Yu-H. Utilizing concurrency: A new theory for memory wall // Languages and Compilers for Parallel Comp. Conf. paper. Lecture Notes in Comp. Sci. (LNCS 10136). Cham, Switzerland: Springer, 2016. P. 18–23. doi: 10.1007/978-3-319-52709-3_2. URL: https://link.springer.com/book/10.1007/978-3-319-52709-3 (дата обращения: 24.12.2024).

3. When parallel speedups hit the memory wall / A. F. A. Furtunato, K. Georgiou, K. Eder, S. Xavier-de-Souza // IEEE Access. 2010. Vol. 8. P. 79225–79238. doi: 10.1109/ACCESS.2020.2990418.

4. Optimized lossless embedded compression for mobile multimedia applications / S. Yoon, S. Jun, Y. Cho, K. Lee, H. Jang, T. H. Han // J. Electronics. 2020. Vol. 9, no. 5. P. 868. doi: 10.3390/electronics9050868.

5. Mittal S., Vetter J. S. A survey of architectural approaches for data compression in cache and main memory systems // IEEE Transactions on Parallel and Distributing Systems. 2016. Vol. 27, no. 5. P. 1524–1536. doi: 10.1109/TPDS.2015.2435788.

6. Bratt I. The ARM® Mali-T880 Mobile GPU // 2015 IEEE Hot Chips 27 Symp. (HCS). Cupertino, CA, USA: IEEE, 2015. P. 1–27. doi: 10.1109/HOTCHIPS.2015.7477462.

7. Optimization of false-overlap detection of tile assembly in tilebased rendering / B. Yang, M. Fan, M. Han, Y, Geng // 2020 Asia-Pacific Signal and Information Proc. Association Annual Summit and Conf. (APSIPA ASC). Auckland, New Zealand: IEEE, 2020. P. 126–134.

8. Harris P. Arm Mali GPUs. Best practices developer guide. Recommendations for efficient API usage. Cambridge, ARM, 2019. 57 p.

9. Nishida S. Non-scanning display method // Proc. Intern. Conf. on Information Technol.: Coding and Comp. Las Vegas, NV, USA: IEEE, 2002. P. 1000417. doi: 10.1109/ITCC.2002.1000417.

10. LaValle S. M. Virtual reality. Cambridge University Press, 2023.

11. Orbach A. Swapchains unchained! (What you need to know about Vulkan WSI) // 2016 Vulkan DevDay UK. Cambridge, UK: Khronos, 2016. P. 193–204.

12. Пугин К. В., Мамросенко К. А., Гиацинтов А. М. Архитектура программного обеспечения для задач взаимодействия контроллера вывода на экран и операционной системы // Радиоэлектроника. Наносистемы. Информационные технологии (РЭНСИТ). 2021. Т. 13, № 1. С. 87–94. doi: 10.17725/rensit.2021.13.087.

Информация об авторах

Пугин Константин Витальевич – младший научный сотрудник НИИСИ, НИЦ «Курчатовский институт», Нахимовский пр., д. 36, корп. 1, Москва, 117218, Россия.
E-mail: rilian@niisi.ras.ru
https://orcid.org/0000-0002-0053-6230

Гиацинтов Александр Михайлович – канд. техн. наук, старший научный сотрудник НИИСИ, НИЦ «Курчатовский институт», Нахимовский пр., д. 36, корп. 1, Москва, 117218, Россия.
E-mail: giatsintov@niisi.ras.ru
https://orcid.org/0000-0002-1304-4016

Мамросенко Кирилл Анатольевич – канд. техн. наук, руководитель НИИСИ, НИЦ «Курчатовский институт», Нахимовский пр., д. 36, корп. 1, Москва, 117218, Россия.
E-mail: mamrosenko_k@niisi.ras.ru
https://orcid.org/0000-0002-9889-0560

References

1. Ma Z., Segall A. Frame buffer compression for low-power video coding // 18th IEEE Intern. Conf. on Image Proc. Brussels, Belgium: IEEE, 2011. P. 757–760. doi: 10.1109/ICIP.2011.6116665.

2. Sun X.-He, Liu Yu-H. Utilizing concurrency: A new theory for memory wall // Languages and Compilers for Parallel Comp. Conf. paper. Lecture Notes in Comp. Sci. (LNCS 10136). Cham, Switzerland: Springer, 2016. P. 18–23. doi: 10.1007/978-3-319-52709-3_2. URL: https://link.springer.com/book/10.1007/978-3-319-52709-3 (data obrashhenija: 24.12.2024).

3. When parallel speedups hit the memory wall / A. F. A. Furtunato, K. Georgiou, K. Eder, S. Xavier-de-Souza // IEEE Access. 2010. Vol. 8. P. 79225–79238. doi: 10.1109/ACCESS.2020.2990418.

4. Optimized lossless embedded compression for mobile multimedia applications / S. Yoon, S. Jun, Y. Cho, K. Lee, H. Jang, T. H. Han // J. Electronics. 2020. Vol. 9, no. 5. P. 868. doi: 10.3390/electronics9050868.

5. Mittal S., Vetter J. S. A survey of architectural approaches for data compression in cache and main memory systems // IEEE Transactions on Parallel and Distributing Systems. 2016. Vol. 27, no. 5. P. 1524–1536. doi: 10.1109/TPDS.2015.2435788.

6. Bratt I. The ARM® Mali-T880 Mobile GPU // 2015 IEEE Hot Chips 27 Symp. (HCS). Cupertino, CA, USA: IEEE, 2015. P. 1–27. doi: 10.1109/HOTCHIPS.2015.7477462.

7. Optimization of false-overlap detection of tile assembly in tilebased rendering / B. Yang, M. Fan, M. Han, Y, Geng // 2020 Asia-Pacific Signal and Information Proc. Association Annual Summit and Conf. (APSIPA ASC). Auckland, New Zealand: IEEE, 2020. P. 126–134.

8. Harris P. Arm Mali GPUs. Best practices developer guide. Recommendations for efficient API usage. Cambridge, ARM, 2019. 57 p.

9. Nishida S. Non-scanning display method // Proc. Intern. Conf. on Information Technol.: Coding and Comp. Las Vegas, NV, USA: IEEE, 2002. P. 1000417. doi: 10.1109/ITCC.2002.1000417.

10. LaValle S. M. Virtual reality. Cambridge University Press, 2023.

11. Orbach A. Swapchains unchained! (What you need to know about Vulkan WSI) // 2016 Vulkan DevDay UK. Cambridge, UK: Khronos, 2016. P. 193–204.

12. Pugin K. V., Mamrosenko K. A., Giacintov A. M. Arhitektura programmnogo obespechenija dlja zadach vzaimodejstvija kontrollera vyvoda na jekran i operacionnoj sistemy // Radiojelektronika. Nanosistemy. Informacionnye tehnologii (RJeNSIT). 2021. T. 13, № 1. S. 87–94. doi: 10.17725/rensit.2021.13.087. (In Russ.).

Information about the authors

Konstantin V. Pugin – Junior Researcher at the SRISA, National Research Center «Kurchatov Institute», Nakhimovsky pr., 36, build. 1, Moscow, 117218, Russia.
E-mail: rilian@niisi.ras.ru
https://orcid.org/0000-0002-0053-6230

Alexander M. Giatsintov – Cand. Sci. (Eng.), Senior Researcher at the SRISA, National Research Center «Kurchatov Institute», Nakhimovsky pr., 36, build. 1, Moscow, 117218, Russia.
E-mail: giatsintov@niisi.ras.ru
https://orcid.org/0000-0002-1304-4016

Kirill A. Mamrosenko – Cand. Sci. (Eng.), Head at the SRISA, National Research Center «Kurchatov Institute», Nakhimovsky pr., 36, build. 1, Moscow, 117218, Russia.
E-mail: mamrosenko_k@niisi.ras.ru
https://orcid.org/0000-0002-9889-0560

Статья поступила в редакцию 10.06.2025; принята к публикации после рецензирования 11.07.2025; опубликована онлайн 30.10.2025.

Submitted 10.06.2025; accepted 11.07.2025; published online 30.10.2025.