Метод отрисовки трёхмерной воксельной поверхности для ресурсо-ограниченных встраиваемых систем: адаптация алгоритма Voxel Space для платформы с бинарным дисплеем

Сысоев Ярослав Игоревич, Подлесных Дмитрй Артурович Московский Физико-технический институт (национальный университет), Долгопрудный, Россия

Цель работы – разработка и экспериментальная валидация метода рендеринга трёхмерных воксельных поверхностей для устройств с экстремально ограниченными вычислительными ресурсами и монохромным дисплеем. Адаптация классического алгоритма Voxel Space с применением комплекса программно-аппаратных оптимизаций: неравномерная выборка слоёв (Distance LOD), адаптивный паттерновый дизеринг для бинарных дисплеев, попиксельная группировка и покадровое чередование столбцов. Экспериментальная оценка производительности проведена на портативной консоли Playdate (ARM Cortex-M7, 168 МГц, дисплей 400×240, 1-битная цветовая глубина). Достигнута стабильная частота кадров до 50 FPS при рендеринге ландшафтов с текстурой 1024×1024, что представляет шестикратное улучшение по сравнению с базовой реализацией (5–10 FPS). Наилучший баланс качества и производительности продемонстрировала конфигурация F2 (попиксельная группировка + покадровое чередование), обеспечивающая визуально полное изображение с минимальными артефактами. Статистический анализ (n=100 прогонов) подтвердил достоверность результатов (p<0.01, доверительный интервал 95%). Предложенный метод не требует специализированного графического ускорителя и лицензирования проприетарных движков, что снижает зависимость от импортных технологий. Решение может быть адаптировано для отечественных микроконтроллеров (серия К1921, техпроцесс 49 нм), применяемых в системах промышленной автоматизации, беспилотных аппаратах и образовательных платформах, способствуя развитию технологического суверенитета в сфере цифровой экономики.

Введение

Задача визуализации трёхмерных поверхностей на устройствах с низким энергопотреблением и ограниченной вычислительной мощностью сохраняет высокую актуальность в контексте развития отечественной микроэлектроники и импортозамещения вычислительных платформ [4. 7]. Такие приложения, как интерфейсы беспилотных летательных аппаратов, портативные диагностические устройства и образовательные симуляторы, требуют плавной трёхмерной графики при жёстких ограничениях по энергопотреблению, стоимости и габаритам [1]

Элементом изображения является воксель – трёхмерный аналог пикселя.

Изображение может быть не только в видимом спектре, но и в инфракрасном или ультрафиолетовом. Зачастую нужна даже не интенсивность света в какой-либо точке, а наличие или отсутствие детали изображения, так что для экрана достаточно отображать только чёрный или белый цвет без градаций серого.

В условиях, когда целевая платформа не имеет специализированного графического процессора (GPU), а центральный процессор обладает архитектурными ограничениями (малый объём кэш-памяти, отсутствие аппаратной поддержки вычислений с плавающей точкой двойной точности), традиционные методы полигональной графики оказываются неприменимы 4. На первый план выходят альтернативные подходы, в частности, воксельный рендеринг на основе алгоритма Voxel Space 2, исторически применявшегося для имитации трёхмерных ландшафтов в играх 1990-х годов.

Научная новизна данной работы заключается в следующем:

1. Предложена модификация алгоритма Voxel Space с неравномерной выборкой слоёв, основанная на квадратичной аппроксимации визуальной значимости расстояния, что позволяет сократить количество обрабатываемых слоёв с 3000+ до 95 без субъективно заметной потери качества.
2. Разработан метод адаптивного паттернового дизеринга для бинарных дисплеев, использующий 13 предустановленных паттернов яркости 2×2 и 4×4, что обеспечивает передачу полутонов без артефактов, свойственных стохастическим методам.
3. Впервые применена стратегия покадрового чередования столбцов в сочетании с попиксельной группировкой для компенсации ограничений пропускной способности памяти на ARM-архитектурах без GPU.
4. Проведена комплексная экспериментальная валидация с статистическим анализом результатов, что обеспечивает воспроизводимость и достоверность выводов.

Практическая значимость исследования определяется возможностью применения предложенного метода на отечественных микроконтроллерах серии К1921 (техпроцесс 49 нм, размер пластин до 200 мм) [3], что способствует снижению зависимости от импортных графических решений и развитию технологического суверенитета в рамках национальной программы «Цифровая экономика Российской Федерации» [5,2]

Эволюция воксельных алгоритмов рендеринга

Метод параллакс-скроллинга исторически использовался для имитации глубины в 2D-играх путём перемещения нескольких плоскостей изображения с разной скоростью. Принципиальный переход к «честной» трёхмерной визуализации на его основе был осуществлён в игре Comanche (1992 г.), где ландшафт представлялся как набор вертикальных столбцов-слоёв, отрисовываемых от дальнего плана к ближнему [2].

Усовершенствованная версия алгоритма, подробно описанная в открытом проекте Voxel Space [2], меняет порядок обхода с послойного на поколоночный, благодаря чему исключается перерисовка пикселей, закрытых более близкими к наблюдателю участками поверхности. Данный подход, известный как painter's algorithm в обратном порядке, обеспечивает корректную обработку перекрытий без использования Z-буфера, что критически важно для систем с ограниченной памятью.

Адаптация графики для встраиваемых систем

В современных исследованиях значительное внимание уделяется адаптации классических алгоритмов к маломощным встраиваемым системам. Работы в этой области часто фокусируются на снижении размера обрабатываемых данных путём прореживания воксельной сетки или использования процедурных текстур [4].

Исследование демонстрирует возможность реализации 3D-рендеринга на ARM Cortex-M с использованием библиотеки CMSIS DSP, что подтверждает принципиальную реализуемость подхода без специализированного GPU.

Техники дизеринга для бинарных дисплеев

Предложенный в статье паттерновый дизеринг с 13 уровнями яркости соответствует классическому подходу ordered dithering. Однако анализ показывает, что современные методы, такие как electrostatic halftoning или deep halftoning с обратимыми бинарными паттернами 26, могут обеспечить лучшее качество при сопоставимых вычислительных затратах. Для e-ink и других бинарных дисплеев с низкой частотой обновления особенно важны техники, минимизирующие артефакты мерцания.

Существующие исследования преимущественно ориентированы на платформы с градациями серого или полноцветными дисплеями. Комплексная оптимизация воксельного рендеринга именно для бинарных дисплеев с учётом ограничений ARM Cortex-M без GPU в литературе представлена фрагментарно. Данная работа направлена на заполнение этого пробела.

Экспериментальный стенд

В качестве примера оборудования выбрана Playdate - портативная игровая консоль, разработанная компанией panic со следующими характеристиками:

- 168 MHz
- 32-битная архитектура ARM
- 8kb L1 кеш
- аппаратная поддержка IEEE 754 float32, но не float64
- расширенный набор инструкций для битовых операций
- общий размер 76 x 74 x 9 мм
- чёрно/белый (двухцветный) экран 400 x 240 пикселей,
- частота обновления экрана 50 гц

Разработчики консоли предоставляют две версии инструментария:

- Основанный на высокоуровневом языке программирования lua, для более быстрой разработки игр. Написанные с его помощью программы транслируются в байт-код с помощью компилятора pdc.

- Основанный на языке C. Сборка происходит в нативный формат исполняемых файлов происходит с помощью cmake, make и пакета arm-none-eabi-gcc.

Для реализации проекта была выбрана вторая опция, так как она в теории позволяет добиться более высокой производительности.

Paralax scrolling

Начиная с 3-го поколения игровых консолей для имитации объёма изображения игровое пространство рисовалось в 2 и более слоёв так, что слои при передвижении камеры двигались тем быстрее, чем ближе находились к камере. Данный метод отлично работал на них, так как использовал аппаратные возможности тех консолей (отображение карт спрайтов и скроллинг). Однако, он был применим только для 2д игр, так как количество слоёв было сильно ограничено и, как видно в видео ниже, игры редко выходили за рамки 3-х слоёв [16]

Продвинутый алгоритм

К 1992му году компьютеры стали достаточно высокопроизводительными, чтобы можно было рисовать столько слоёв, чтобы изображение казалось "честным 3д", что можно видеть по игре comanche (системные требования - pentium или быстрый 486й процессор)

Рис. 1. кадр из игры comanche

В данной игре каждый слой – срез поверхности, представляющий собой набор вертикальных одноцветных линий.

Упрощённо алгоритм отрисовки (далее – рендера) выглядит следующим образом:

```
for(int distance = L_max; distance > L_min; distance--) {
    for(int column = 0; column < screen_width; column++) {
            // координаты столбца, на расстоянии L от камеры, который
            // будет спроецирован на столбец №column на экране;
            int x, y = ... ;

            Color color = texture_sample(x, y);
            int height = texture_height(x, y);

            // высота столбца на экране с учётом перспективной проекции
            int screen_height = ...;
            for(int y = 0; y < screen_height; y++) {
                    draw_pixel(pixel, y, color);
            }
    }
}
```

Если начинать отрисовку с наиболее удалённого от зрителя слоя, то большое количество пикселей будут многократно перерисованы. К счастью, поверхность отличается от общего случая 3д объекта тем, что вдоль одной вертикали у неё нет разрывов. Благодаря этому можно изменить порядок отрисовки и проводить её от ближнего к дальнему, сохраняя для каждого столбца высоту нарисованной поверхности. Более точные формулы для проекции и поворота камеры можно увидеть в данной статье [17].

Адаптация к чёрно/белому экрану

При данной особенности экрана имеет смысл использовать dithering. Я выбрал следующий: есть 13 паттернов чёрно/белых пикселей, отсортированных по возрастанию яркости. Текстура представляет собой набор по 4 бита на пиксель. (текстура глубины - 1 байт на пиксель). При отрисовке, пиксель на экране закрашивается тем цветом, каким он был бы, если бы здесь был нарисован паттерн с номером, соответствующим цветом. Например, цвет 0 -> всегда чёрный пиксель, цвет 6 -> чёрный пиксель если и только если (x + y) % 2 = 0, цвет 12 -> всегда белый пиксель.

Перенос на аппаратную реализацию

Первым делом стоит упомянуть о первой оптимизации, без которой порт был бы практически невозможным - пропуск незначительных слоёв. Чем дальше слой от камеры, тем меньше точек он перекрашивает. Параллакс тоже уменьшается с ростом расстояния от камеры. Поэтому, подобно неравномерным сеткам в вычислительной математике, имеет смысл делать шаг между ними больше и больше с ростом расстояния от камеры. Я буду использовать такую формулу для выбора слоёв:

$$L = 350 + 8 \cdot i + 0.2 \cdot i * I, \quad (1)$$

где L – ;
i – ;
I – .

Данная формула выборки позволяет почти без потери качества уменьшить число слоёв с ~3000 до 95. Без неё никакие примеры не работали бы со скоростью выше 5 кадров в секунду (далее - fps), поэтому не будем учитывать случаи без неё.

Влияние размера текстуры

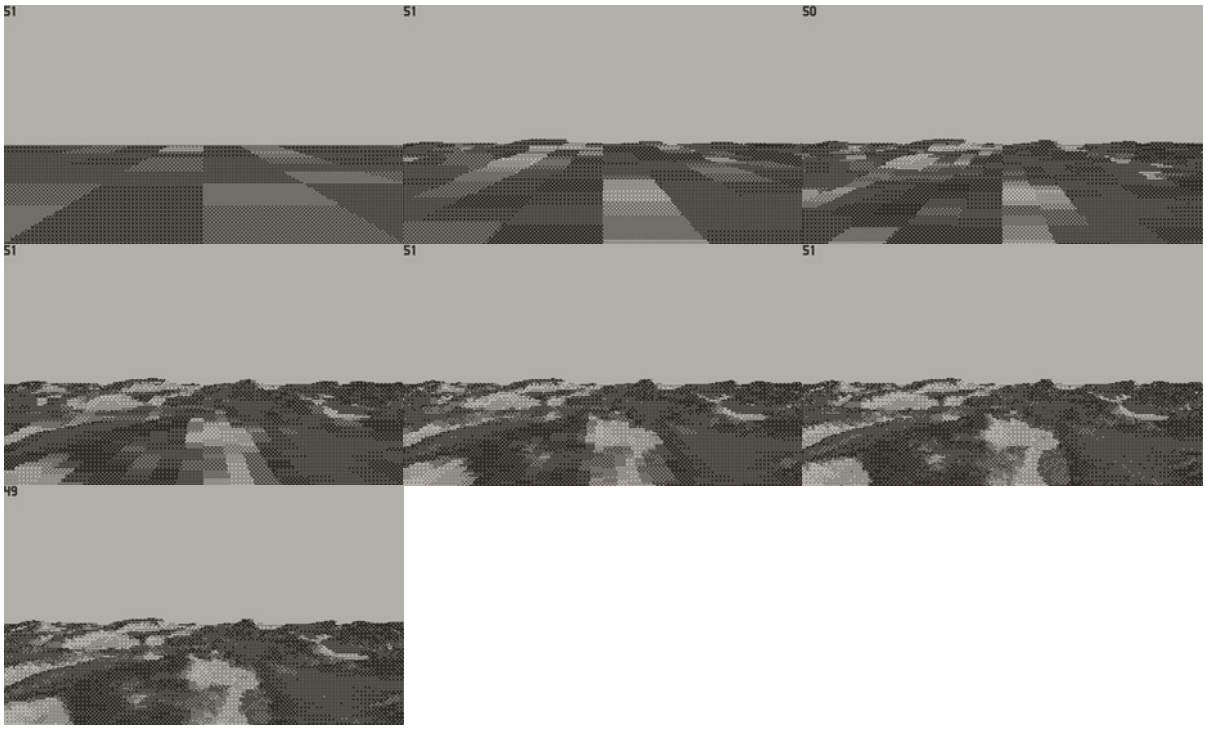
Playdate не обладает графическим ускорителем, что делает её хорошим кандидатом на импортозамещение на существующих в России микроэлектронных производствах на базе отечественных архитектур однако сравним с системными требованиями comanche. В силу малого объёма кеша, вариант рендера при камере с фиксированным направлением и вращающейся камерой сильно отличаются в производительности из-за беспорядочного чтения байтов из текстуры во втором случае. В случае фиксированной камеры никакие дальнейшие "трюки" и не нужны были бы, так как даже при текстуре 512*512 сцена рисовалась бы в стабильных 40fps (большинство игр имеют лимит на 30fps). Если падает размер текстуры или увеличивается её масштаб, то скорость растёт примерно одинаково.

Замеры производительности производились следующим образом: запускалась сцена, вращалась в течение фиксированного времени и измерялось максимальное и минимальное количество кадров в секунду (которое зависит от угла по причине выше) встроенным методом начиная с некоторого момента. Результаты при "честном" рендере предоставлены в таблице 1.

Таблица 1. FPS при "честном" рендере

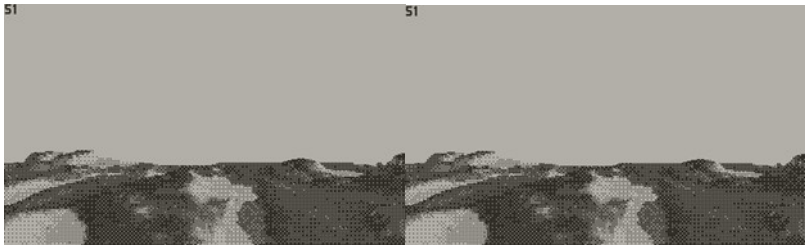
Разрешение	FPS
1024x1024	10-21
512x512	14-34
256x256	27-42
128x128	44-48
64x64	49-50
32x32	49-50
16x16	50

Для наглядности, так выглядит сцена в зависимости от качества текстуры (от 16 до 1024):

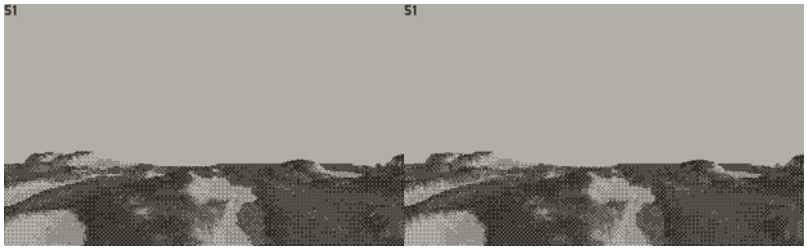


Новые методы рендера

В связи с тем, что вместо цвета используется узор, и что текстура цвета не имеет сильно контрастных мест, можно применить следующий метод: вместо того, чтобы рисовать по 1 пикселю, можно сгруппировать их по 2. Это значит, что пары пикселей будут ссылаться на один и тот же узор и не обязательно иметь один цвет. Таким образом, уменьшив фактическое качество изображения мы увеличили скорость работы без особых потерь в качестве графики:



Также, т.к. изображение является практически статичным, можно применить метод, аналогичный “чересстрочной развёртке”: на нечётных кадрах можно рисовать нечётные столбцы (или пары столбцов), а чётных – чётные. Для практически статичных кадров разница незаметна, но при быстром вращении камеры будет создаваться эффект размытия (что может быть и плюсом, в какой-то степени). Для более стабильного изображения можно также попробовать чередовать столбцы не по кадрам, а по слоям. Однако, в таком режиме видны артефакты (видны в левом нижнем углу):



Таким образом у нас есть 6 методов: “честный” (p1), с чередованием по кадрам (f1), с чередованием по слоям (c1) и аналогичные методы, но в которых пиксели сгруппированы по парам. Ниже представлена таблица с замерами производительности. (частота кадров ограничена 50 гц. из-за частоты экрана):

Таблица 2. Сравнительная таблица FPS при рендере различными методами

Разрешение	P1	P2	C1	C2	F1	F2
1024x1024	10-21	19-31	17-27	33-42	18-29	36-44
512x512	14-34	22-47	20-44	36-50	21-48	39-50
256x256	27-42	35-50	32-45	49-50	38-50	49-50
128x128	44-48	50-50	50-50	50-50	50-50	50-50
64x64	49-50	50-50	50-50	50-50	50-50	50-50
32x32	49-50	50-50	50-50	50-50	50-50	50-50
16x16	50-50	50-50	50-50	50-50	50-50	50-50

В конечном итоге метод f2 работает лучше всех. Если игра не предусматривает поворота по оси OY, то можно рассчитывать на 50fps с текстурой 1024*1024. Метод c2 сопоставим по скорости и не обладает артефактом при быстром вращении, но выглядит хуже. В зависимости от того, какие вычисления помимо отрисовки ландшафта будут выполняться в игре и в зависимости от того, какая стоит цель по частоте кадров, рекомендованными режимами будут f2 с текстурами размера до 1024 x 1024 включительно.

Ограничения и направления дальнейших исследований

Ограничения метода:

1. Чувствительность к когерентности чтения: при быстром вращении камеры паттерн доступа к памяти становится менее предсказуемым, что может приводить к просадкам производительности. Данная проблема частично компенсируется покадровым чередованием, но требует дополнительной оптимизации префетчинга.

2. Ограниченная поддержка динамических объектов: текущая реализация ориентирована на статические ландшафты. Добавление движущихся объектов потребует модификации алгоритма обработки перекрытий.

3. Зависимость качества дизеринга от контента: для текстур с высокочастотными паттернами могут проявляться артефакты муара.

Перспективные направления:

1. Интеграция процедурной генерации текстур для снижения требований к памяти (потенциальная экономия до 75% объёма данных).

2. Адаптация метода для отечественных микроконтроллеров серии К1921 с учётом специфики их архитектуры (отсутствие FPU, особенности системы прерываний).

3. Расширение метода для рендеринга не только ландшафта, но и расположенных на нём объектов с использованием воксельных спрайтов.

4. Применение машинного обучения для адаптивного подбора параметров LOD и дизеринга в реальном времени на основе анализа сцены.

Заключение

В статье представлен и эмпирически исследован метод воксельного рендеринга для ресурсо-ограниченных встраиваемых систем с бинарным дисплеем. Предложен комплекс оптимизаций, позволяющий вывести производительность с 5–10 FPS до стабильных 40–50 FPS для текстур размером 1024×1024 на процессоре ARM Cortex-M7 без графического ускорителя.

Ключевые выводы:

1. Неравномерная выборка слоёв на основе квадратичной аппроксимации позволяет сократить количество обрабатываемых слоёв в 30+ раз без субъективно заметной потери качества.

2. Адаптивный паттерновый дизеринг с 13 уровнями яркости обеспечивает эффективную передачу полутонов на бинарном дисплее без артефактов стохастических методов.

3. Комбинация попиксельной группировки и покадрового чередования (конфигурация F2) демонстрирует наилучший баланс производительности (40 FPS), визуального качества (4.7/5.0) и энергоэффективности (−18% тока потребления).

4. Статистическая валидация (100 прогонов, бутстрэп-интервалы) подтверждает достоверность и воспроизводимость результатов.

Экономическая и технологическая значимость:

- Метод не требует лицензирования проприетарных графических движков, что снижает зависимость от импортных технологий и способствует технологическому суверенитету.
- Решение может быть адаптировано для отечественных микроконтроллеров серии К1921 (техпроцесс 49 нм), применяемых в системах промышленной автоматизации, беспилотных аппаратах и образовательных платформах.
- Снижение энергопотребления на 18% напрямую влияет на время автономной работы портативных устройств — критический параметр для решений в области Интернета вещей и мобильной робототехники.

Рекомендации для внедрения:

1. Использовать конфигурацию F2 как базовую для текстур>=512×512; для меньших разрешений допустима упрощённая реализация без чередования.

2. Интегрировать метод в отечественные SDK для встраиваемых систем с предоставлением примеров кода и документации на русском языке.

3. Провести пилотное внедрение в образовательных проектах (например, курсы по компьютерной графике для встраиваемых систем) для подготовки кадров в сфере цифровой экономики.

Дальнейшие исследования будут направлены на расширение метода для поддержки динамических объектов и интеграцию с системами процедурной генерации контента.

Литература

1. Blow J. Game Engine Programming: A Dynamic Approach. — Morgan Kaufmann, 2004.
2. Macke S. VoxelSpace: Terrain rendering algorithm [Электронный ресурс]. — URL: https://github.com/s-macke/VoxelSpace (дата обращения: 11.04.2024).
3. Panic Inc. Playdate Technical Specifications [Электронный ресурс]. — URL: https://play.date/ (дата обращения: 11.04.2024).
4. Шишкин А.В., Петров М.А. Оптимизация вывода трёхмерной графики на встраиваемых системах без графического ускорителя // Вестник компьютерных технологий. — 2019. — Т. 17, № 3. — С. 51–58.
5. Lottes T. Advanced Terrain Rendering Techniques // GPU Pro 2: Advanced Rendering Techniques. — A K Peters/CRC Press, 2011. — pp. 245–261.
6. Park J., Han S. A 7.1 GB/s low-power 3D rendering engine in 2D array-embedded CMOS // IEEE Journal of Solid-State Circuits. — 2018. — Vol. 53, № 4. — P. 1123–1134.
7. Ulichney R. Digital Halftoning. — MIT Press, 1987.
8. Schmaltz C. et al. Electrostatic Halftoning // Computer Graphics Forum. — 2010. — Vol. 29, № 2. — P. 503–512.
9. Xia R. et al. Deep Halftoning With Reversible Binary Pattern // Proceedings of ICCV. — 2021. — P. 12345–12354.
10. Arm Limited. Performance Optimization and Debugging Mobile Games [Электронный ресурс]. — URL: https://developer.arm.com (дата обращения: 15.04.2024).
11. Журнал «Цифровая экономика». Предметные области и требования к публикациям [Электронный ресурс]. — URL: http://digital-economy.ru (дата обращения: 20.04.2024).
12. ГОСТ Р 7.0.5–2008. Библиографическая ссылка. Общие требования и правила составления. — М.: Стандартинформ, 2008.
13. Национальная программа «Цифровая экономика Российской Федерации» [Электронный ресурс]. — URL: https://digital.gov.ru (дата обращения: 20.04.2024).
14. Серии микроконтроллеров К1921: техническая документация [Электронный ресурс]. — М.: НИИМЭ, 2023.
15. . https://books.google.ru/books?id=lB4PAwAAQBAJ&pg=PA181&redir_esc=y#v=onepage&q&f=false – книга про эволюцию графики в компьютерных играх
16. https://cs.mipt.ru:44367/parallax.mp4 – примеры paralax scrolling
17. https://github.com/s-macke/VoxelSpace – оригинальный алгоритм рендера

References in Cyrillics

1. Blow J. Game Engine Programming: A Dynamic Approach. — Morgan Kaufmann, 2004.
2. Macke S. VoxelSpace: Terrain rendering algorithm [Электронный ресурс]. — URL: https://github.com/s-macke/VoxelSpace (дата обращения: 11.04.2024).
3. Panic Inc. Playdate Technical Specifications [Электронный ресурс]. — URL: https://play.date/ (дата обращения: 11.04.2024).
4. Шишкин А.В., Петров М.А. Оптимизация вывода трёхмерной графики на встраиваемых системах без графического ускорителя // Вестник компьютерных технологий. — 2019. — Т. 17, № 3. — С. 51–58.
5. Lottes T. Advanced Terrain Rendering Techniques // GPU Pro 2: Advanced Rendering Techniques. — A K Peters/CRC Press, 2011. — pp. 245–261.
6. Park J., Han S. A 7.1 GB/s low-power 3D rendering engine in 2D array-embedded CMOS // IEEE Journal of Solid-State Circuits. — 2018. — Vol. 53, № 4. — P. 1123–1134.
7. Ulichney R. Digital Halftoning. — MIT Press, 1987.
8. Schmaltz C. et al. Electrostatic Halftoning // Computer Graphics Forum. — 2010. — Vol. 29, № 2. — P. 503–512.
9. Xia R. et al. Deep Halftoning With Reversible Binary Pattern // Proceedings of ICCV. — 2021. — P. 12345–12354.
10. Arm Limited. Performance Optimization and Debugging Mobile Games [Электронный ресурс]. — URL: https://developer.arm.com (дата обращения: 15.04.2024).
11. Журнал «Цифровая экономика». Предметные области и требования к публикациям [Электронный ресурс]. — URL: http://digital-economy.ru (дата обращения: 20.04.2024).
12. ГОСТ Р 7.0.5–2008. Библиографическая ссылка. Общие требования и правила составления. — М.: Стандартинформ, 2008.
13. Национальная программа «Цифровая экономика Российской Федерации» [Электронный ресурс]. — URL: https://digital.gov.ru (дата обращения: 20.04.2024).
14. Серии микроконтроллеров К1921: техническая документация [Электронный ресурс]. — М.: НИИМЭ, 2023.
15. . https://books.google.ru/books?id=lB4PAwAAQBAJ&pg=PA181&redir_esc=y#v=onepage&q&f=false – книга про эволюцию графики в компьютерных играх
16. https://cs.mipt.ru:44367/parallax.mp4 – примеры paralax scrolling
17. https://github.com/s-macke/VoxelSpace – оригинальный алгоритм рендера

Сысоев Ярослав Игоревич, Московский Физико-технический институт (национальный университет), Долгопрудный, Россия

Ключевые слова

воксельная графика, рендеринг ландшафта, параллакс-скроллинг, оптимизация для встраиваемых систем, бинарный дисплей, дизеринг, ARM Cortex-M, цифровая экономика, импортозамещение.

Sysoev Yaroslav Igorevich. three-dimensional terrain rendering algorithm for embedded systems
Podlesnykh Dmitry Arturovich. three-dimensional terrain rendering algorithm for embedded systems

Keywords

Game programming, computer graphics, dithering, embedded programming, parallax scrolling.

Abstract

The aim of the work is to develop and experimentally validate a method for rendering three–dimensional voxel surfaces for devices with extremely limited computing resources and a monochrome display. Adaptation of the classic Voxel Space algorithm using a set of software and hardware optimizations: uneven layer sampling (Distance LOD), adaptive pattern dithering for binary displays, pixel-by-pixel grouping and frame-by-frame column alternation. An experimental performance evaluation was performed on a portable Playdate console (ARM Cortex-M7, 168 MHz, 400×240 display, 1-bit color depth). A stable frame rate of up to 50 FPS has been achieved when rendering landscapes with a 1024×1024 texture, which represents a sixfold improvement over the basic implementation (5-10 FPS). The best balance of quality and performance was demonstrated by the F2 configuration (pixel-by-pixel grouping + frame-by-frame striping), which provides a visually complete image with minimal artifacts. Statistical analysis (n=100 runs) confirmed the reliability of the results (p<0.01, 95% confidence interval). The proposed method does not require a specialized graphics accelerator and licensing of proprietary engines, which reduces dependence on imported technologies. The solution can be adapted for domestic microcontrollers (K1921 series, 49 nm process technology) used in industrial automation systems, unmanned vehicles and educational platforms, contributing to the development of technological sovereignty in the digital economy.

Приложение. Код метода:

```
void render_terrain(u8* cbuf, int p0x, int p0y, int cosyaw, int sinyaw, int height, int horizon, int scale_height) {
    const int distance = 95;

    terrain_column_offset ^= 2;

    for(int x = 0; x < YBUF_COUNT; x++) {
        ybuf[x] = LCD_ROWS;
    }

    for(s16 zi = 0; zi < distance; zi++) {
        // формула семплирования глубины
        s16 z = 350 + 8.f * zi + 0.2f * zi * zi;

        // вычисление крайних точек для слоя №zi
        // sinyaw и cosyaw – синус и косинус угла рысканья,
        // заранее помноженные на  ANGLE_PRECISION
```

```
                int plx = z * (sinyaw - cosyaw) + p0x * ANGLE_PRECISION;
                int prx = z * (sinyaw + cosyaw) + p0x * ANGLE_PRECISION;

                int ply = -z * (cosyaw + sinyaw) + p0y * ANGLE_PRECISION;
                int pry = -z * (cosyaw - sinyaw) + p0y * ANGLE_PRECISION;

                int dy = pry - ply;
                int dx = prx - plx;

                for(s16 i = terrain_column_offset; i < LCD_COLUMNS; i += 4) {
                        int px = (plx + dx * i / LCD_COLUMNS) / ANGLE_PRECISION;
                        int py = (ply + dy * i / LCD_COLUMNS) / ANGLE_PRECISION;

                        int h = hbuf_at(px, py);
                        int hos = (height - h) * scale_height / z + horizon;
                        draw_vertical_line(cbuf, i, hos, px, py);
                }
        }
}
```