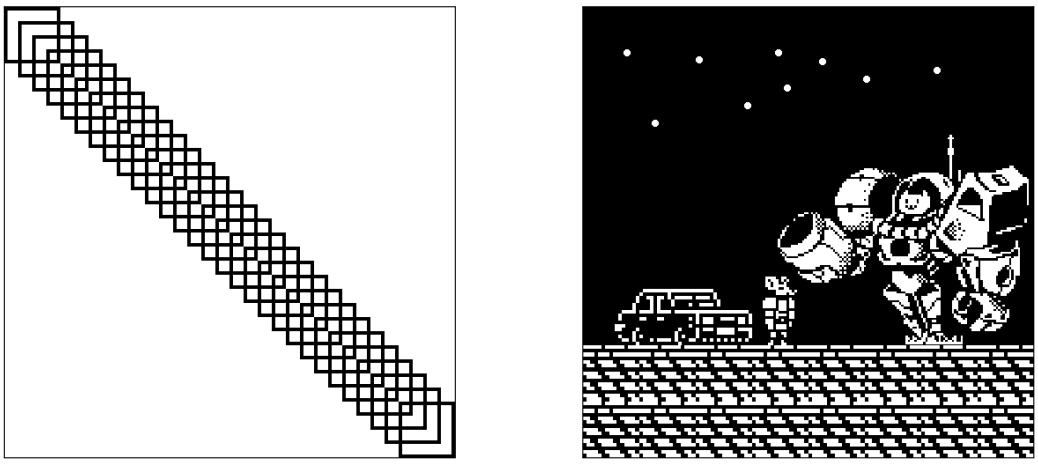
Содержание



(a) (b)

Рис. 1: Пример результатов работы бинарного рендерера.

1 Преабмула

Вы попали в прошлое на машине времени, в середину 20 века. Идёт эпоха зарождения ЭВМ. Деревья зелёные, лампы большие, частоты низкие.

На заседании КПСС от 10 октября 1952 года принято решение о разработке графического процессора ГП-1 (далее в коде **GP_ONE**), цель которого состоит в отображении карт, чертежей, текстовой информации и других изображений в 2D в реальном времени.

Вам нужно разработать прототип, который позволит в дальнейшем спроектировать и реализовать ГП-1. В услових имеющихся ограничений главный конструктор принимает следующие проектные решения:

- Используется строго чёрно-белый монитор, т.е. буфер кадра хранит строго 1 бит на 1 пиксел (см. Рис. 1).
- Буфер кадра хранится тайлами-полосками размера 1x16. Каждый тайл кодируется одним полу-словом типа `uint16_t`. Притом чем выше разряд бита в слове, тем более правый пиксель он кодирует (см. Рис. 2).
- Для того чтобы Ваш алгоритм можно было в последствии реализовать в аппаратуре необходимо ограничить используемые типы данных. Вы можете использовать только 2 типа данных: `int16_t` и `uint16_t`.

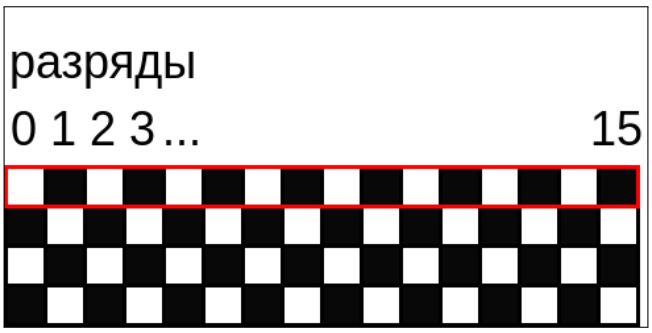


Рис. 2: Порядок битов в кодируемом тайле.

- К сожалению инженеры в вашей научной группе ещё не знают, будет ли реализована аппаратная поддержка операций умножения и деления. Скорее всего нет, поэтому **необходимо** обходиться без них. А вот **сдвиги**, сложения и вычитания точно будут.

- Можно выполнить вычисления с использованием умножений/делений во время компиляции (constexpr).

2 Задача

2.1 Основная часть

В буфер кадра растеризуются спрайты, которые состоят из 2 бит. 1 бит на цвет и 1 бит на альфа-канал отвечающий за прозрачность. Данные для спрайтов хранятся так же как и в буфере кадра – тайлами по 1x16.

Разрешение спрайтов 64x64, разрешение буфера кадра – 512x512. Однако позиция спрайта на изображении произвольная – и задаётся позицией левого верхнего угла спрайта (левого нижнего, если учитывать, что ось Y смотрит вниз).

Вам необходимо реализовать растеризацию спрайта с прозрачностью исходя из соображений максимальной экономии вычислений.

Предполагается что память у Вас относительно-быстрая. Поэтому вы должны постараться обрабатывать группы по 16 пикселей сразу 1 командой, оперирующей `int16_t` или `uint16_t`.

Известно, что разнообразие спрайтов не велико (не более 32х разных спрайтов). Но зато число отрисовываемых спрайтов может быть достатоно большим.

2.2 Детали реализации

Спрайт представлен в виде простой структуры в файле Sprite.h:

```
struct Sprite {
    uint16_t color[SPRITE_BUF_SIZE];
    uint16_t alpha[SPRITE_BUF_SIZE];
};
```

Для работы со спрайтами определены следующие константы:

1. `SPRITE_WIDTH` = 64 - ширина спрайта.
2. `SPRITE_HEIGHT` = 64 - высота спрайта.
3. `SPRITE_TILES_X` = 4 - количество тайлов в спрайте по оси X.
4. `SPRITE_TILES_Y` = `SPRITE_HEIGHT` - количество тайлов в спрайте по оси Y.
5. `SPRITE_BUF_SIZE` = `SPRITE_TILES_X*SPRITE_TILES_Y` - размер буфера, необходимый для хранения всех тайлов одного спрайта.

Буфер кадра представлен в виде похожей на спрайт структуры в файле FrameBuffer.h:

```
struct FrameBuffer {
    uint16_t color[FRAMEBUFFER_BUF_SIZE];
};
```

Для буфера кадра определены аналогичные спрайту константы.

Для эмуляции видеоускорителя ГП-1 был создан статический класс `GP_ONE`, описанный в файле `GP_ONE.h`:

```
class GP_ONE {
public:
    static Sprite   spriteMemory[MAX_SPRITE_COUNT];
    static uint16_t frameBuffer[FRAMEBUFFER_BUF_SIZE];

    static void loadSprites(const Sprite *sprites, uint16_t spriteCount);
```

```
    static void clearFrameBuffer(BackGroundColor bkgColor);
    static void drawSpriteInstances(const SpriteInstance *instances,
    uint16_t instanceCount);
    static void saveFrameBuffer(FrameBuffer &outFrameBuffer);
};
```

Массив **spriteMemory** эмулирует память ускорителя ГП-1, предназначенную для хранения спрайтов. А поле **frameBuffer** - эмулирует память ГП-1, предназначенную для хранения значений буфера кадра. Предполагается, что ГП-1 умеет рендерить только в свой внутренний буфер кадра, заданный этой переменной.

Вам необходимо реализовать 4 функции этого класса:

1. *loadSprites* – загружает массив спрайтов в память ГП-1.

2. *clearFrameBuffer* – очищает внутренний буфер кадра ГП-1 заданным цветом (чёрным, как на рис.1b или белым, как на рис.1a), задаваемым переменной типа BackGroundColor:

```
enum class BackGroundColor {
    BLACK, WHITE
};
```

3. *drawSpriteInstances* – единственная команда отрисовки. Принимает на вход массив инстансов загруженных в память ГП-1 спрайтов. Каждый инстанс спрайта задаётся структурой:

```
struct SpriteInstance {
    uint16_t x;
    uint16_t y;

    uint16_t ind;
};
```

   Где:

   - x, y – координаты левого верхнего угла спрайта.
   - ind – индекс спрайта в массиве спрайтов, загруженных в память ГП-1.

4. *saveFrameBuffer* – сохраняет значения внутреннего буфера кадра ГП-1 во внешний фреймбуфер, задаваемый аргументом outFrameBuffer.

Критерии оценки основной части:

- Проект шаблона из мейна автоматически запускает тесты для 5 сцен. Результаты работы вашей реализации он сохраняет в директории Render. Ваша задача добиться совпадения отрисовываемых вашим алгоритмом изображений с теми, что лежат в директории Scenes/Render. Т.е. необходимо получить уведомление об **успешном прохождении** всех 5 **тестов**. (**5 баллов**)

- Реализация без использования операций умножения/деления/взятия остатка от деления. (**3 балла**)

- Дополнительные оптимизации производительности кода. (**2 балла**)

- Реализуйте растеризацию спрайтов с использованием современных векторных расширений SSE, AVX/AVX2 или AVX512. Измерьте производительность векторной версии по отношению к скалярной на вашем процессоре и добавьте в отчёт. Для этой цели вы можете использовать “std::chrono::high_resolution_clock” (**3 балла**).

2.3 Сборка и требования к сдаче заданий

Сборка и запуск CMake проекта:

```
mkdir build
cd build
cmake ..
make
./BinaryRenderer
```

Рекомендуется всю реализацию базовой части задания поместить в файле `GP_ONE.cpp` без изменений в других файлах и структуре проекта.

В корень проекта необходимо приложить `readme.txt` файл с описанием выполненных пунктов основной и дополнительной частей задания.

Решение нужно прислать в виде zip архива с названием следующего формата:

`<номер группы>_<инициалы>_<фамилия>.zip`

В случае, если вы не с ВМК МГУ, в качестве номера группы напишите "000".

2.4 Дополнительная часть (анализ производительности)

Предполагаемая тактовая частота ГП-1 – 1 КГц ("Стрела" работала на 2 КГц). Требуемое разрешение изображения в буфере кадра – 512x512 пикслей (хотя здесь это не важно).

Сдвиги, сложения и вычитания занимают 1 такт. Любая операция работы с памятью также может быть принята за 1 такт (очень сильное предположение конечно, но в наших модельных условиях выполнимое).

Задача 1 (**2 балла**) – оценить, какое количество спрайтов в секунду можно будет отрисовывать если:

- Ваш алгоритм реализован на центральном процессоре, который умеет складывать, вычитать и сдвигать числа. Процесор скалярный, 1 команда за 1 такт.
- Ваш алгоритм реализован аппаратно (за 1 прохождение тактового сигнала через схему).
- Ваш алгоритм реализован аппаратно в 4 юнитах, позволяющих обрабатывать по 4 тайла 1x16 одновременно.
- Ваш алгоритм реализован аппаратно в 16 юнитах, позволяющих обрабатывать по 16 тайлов 1x16 одновременно.
- Ваш алгоритм реализован аппаратно в 64 юнитах, позволяющих обрабатывать по 64 тайла 1x16 одновременно.
- Ваш алгоритм реализован аппаратно в 256 юнитах, позволяющих обрабатывать весь спрайт одновременно.

Задача 2 (**2–5 баллов**) – оценить какое количество транзисторов потребует аппаратная реализация в 1, 4 и 16, 64 и 256 юнитах. Подсказка: самое точное решение вы получите если реализуете схему на VHDL, проведёте синтез в САПР (например Quartus) и возьмёте из очёта число логических вентилей, из которых уже можно оценить число транзисторов. В этом случае необходимо предоставить в отчёте схему. Но вы можете оценить количество требуемых транзисторов вручную.

Задача 3 (**1–3 балла**). Вы возвращаетесь в наше время. Ваш алгоритм реализован на 64-битном скалярном центральном процессоре и Вы имеете современный оптимизирующий компилятор. Теперь вы можете использовать типы данных int64_t и uint64_t. Оцените, какое количество спрайтов в секунду вы можете растеризовать на частоте 1 ГГц (Гигагерц). Изменится ли результат если вы имеете:

1. Супер-скалярный процессор с **очередным** выполнением команд, 2 команды за 1 такт.

2. Супер-скалярный процессор с **очередным** выполнением команд, 4 команды за 1 такт.

3. Супер-скалярный процессор с **вне-очередным** выполнением команд, 4 команды за 1 такт. Можно считать, что очередь команд достаточно большая, чтобы не рассматривать задержки, вызванные её заполнением.

Примечание: На наш взгляд на этот вопрос нет единственно-верного ответа. Нам хотелось бы увидеть творческий подход к решению проблемы и вменяемое обоснование для вашего ответа. Подумайте вот над чем. Если между вторым и третьим вариантом есть разница, тогда какие оптимизации компилятор мог бы применить применить для второго случая, чтобы сократить разрыв между 2 и 3 вариантами?

Подсказка: для анализа полученного ассемблера можно использовать удобный сервис Compiler Explorer (https://godbolt.org/)

Отчет о проделанной иследовательской работе необходимо положить в корневую директорию проекта.