



МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ М. В. ЛОМОНОСОВА
Факультет вычислительной математики и кибернетики
Кафедра системного анализа

Отчёт и документация по заданию

«Компьютерная графика: 2D-графика на языке C++»

Студент 415 группы
А. Т. Айтеев

Москва, 2021

Содержание

1	Предварительная подготовка для написания кода	3
1.1	Скачать и установить шаблон	3
1.2	Установка необходимых пакетов для работы	3
1.3	Компиляция	4
2	Критерии оценки	6
2.1	Основная часть	6
2.2	Дополнительная часть(анализ производительности)	6
3	Структура готового приложения	8
3.1	Папка 0 16 BIT	9
3.2	Папка 1 MMX,SSE,AVX	10
4	Как компилировать и запускать готовое задание	10
5	Подробности реализации приложений	11
5.1	Общая структура GP_ONE	11
5.2	Папка 0 16BIT	13
5.2.1	Программа binary_rendering_1_EASY	13
5.2.2	Программа binary_rendering_2_NORMAL	14
5.2.3	Программа binary_rendering_3_HARD	15
5.3	Папка 1 MMX,SSE,AVX/0 Four Tiles Only	16
5.3.1	Программа 0_binary_rendering_2_NORMAL	16
5.3.2	Программа 1_binary_rendering_2_NORMAL_MMX	17
5.3.3	Программа 2_binary_rendering_2_NORMAL_SSE	18
5.3.4	Программа 3_binary_rendering_2_NORMAL_AVX	20
5.3.5	Программа MOD TO ANALYSE	21
5.4	Папка 1 MMX,SSE,AVX/1 All Possible Tiles	22
5.4.1	Программа 0_binary_rendering_2_NORMAL	23
5.4.2	Программа 1_binary_rendering_2_NORMAL_MMX	24
5.4.3	Программа 2_binary_rendering_2_NORMAL_SSE	25
5.4.4	Программа 3_binary_rendering_2_NORMAL_AVX	27
6	Анализ скорости выполнения программ	30
6.1	Особенность реализаций	30
6.2	16 BIT	30
6.3	0 Four Tiles Only	31
6.4	1 All Possible Tiles	33
7	Дополнительная часть (анализ производительности)	35
7.1	16 BIT	35
7.2	64 BIT	35

1 Предварительная подготовка для написания кода

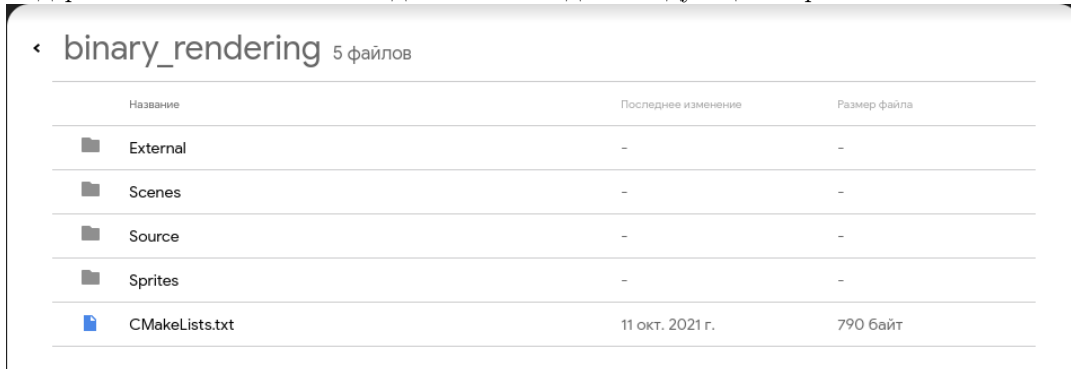
В данной главе мы рассмотрим как подготовиться к написанию проекта, а так же пройдем подготовительный этап для написания кода.

1.1 Скачать и установить шаблон

Прежде всего необходимо скачать архив "binary_rendering.zip" по следующей ссылке: [ссылка на архив](#).

Содержимое папки binary_rendering"разархивировать в удобное для вас место (в нашем случае это папка "CG TASK1").

Содержимое папки CG TASK1 должно выглядеть следующим образом:



◀ binary_rendering 5 файлов		
Название	Последнее изменение	Размер файла
External	-	-
Scenes	-	-
Source	-	-
Sprites	-	-
CMakeLists.txt	11 окт. 2021 г.	790 байт

1.2 Установка необходимых пакетов для работы

Далее необходимо ввести следующие команды в терминал:

```
sudo apt-get update  
sudo apt install build-essential
```

Они установят основные компоненты GCC и базовые пакеты для работы с языками «C» и «C++» (Подробнее смотрите по ссылке [Установка GCC в Ubuntu](#))

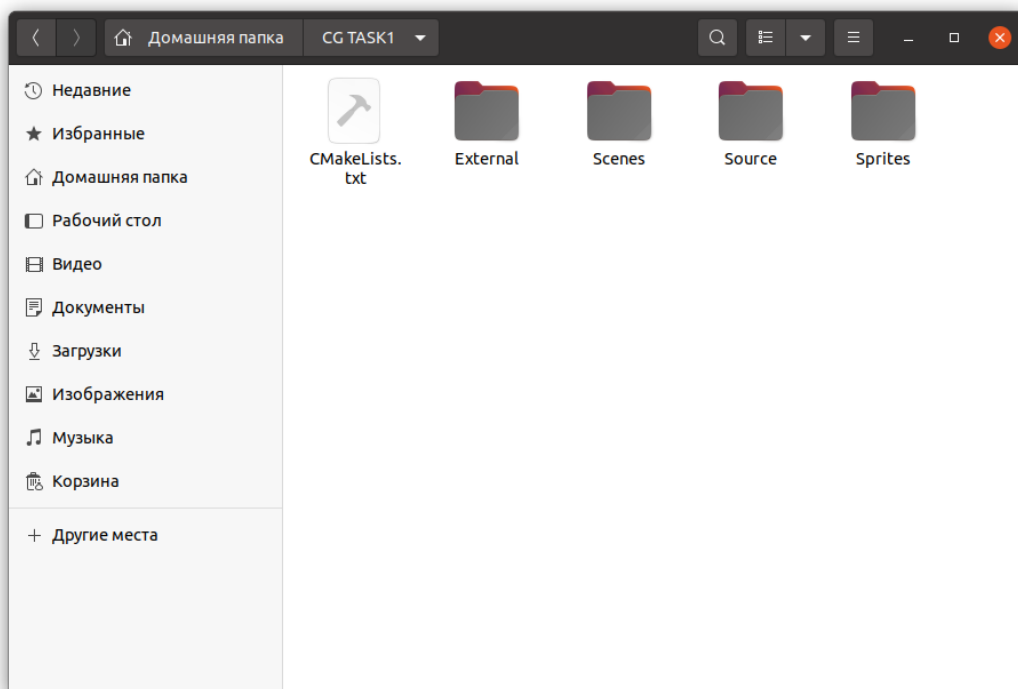
Так же установим пакеты для компиляции make файлов:

```
sudo apt-get install cmake  
sudo apt-get install libglfw3-dev
```

Теперь мы полностью готовы к написанию программы.

1.3 Компиляция

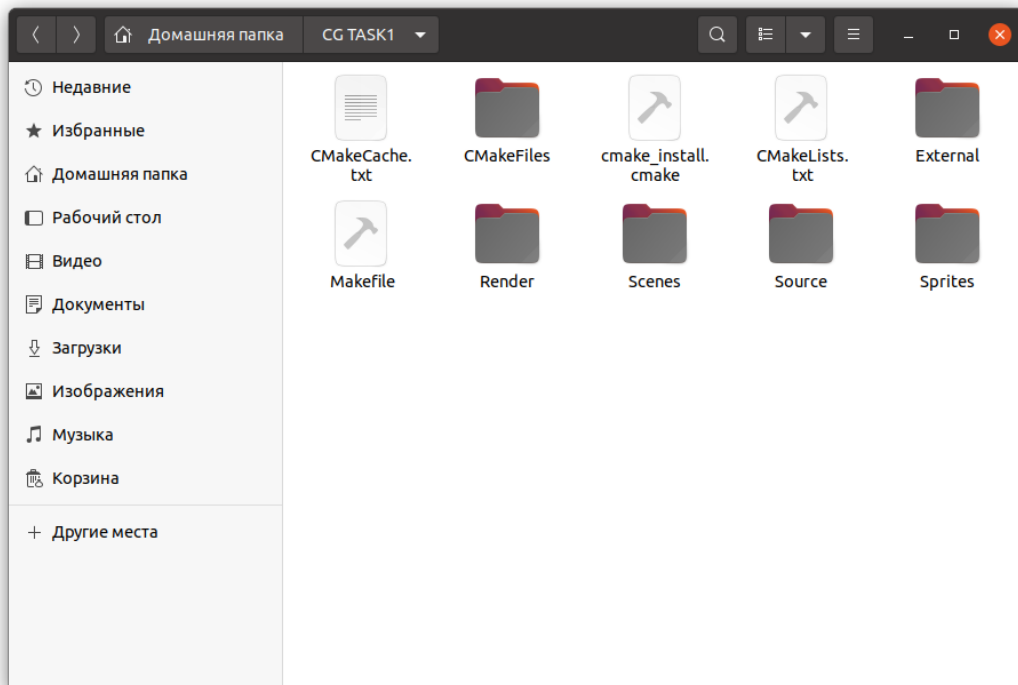
Откройте в терминале корень проекта (в папке CG TASK1)



Введите следующую команду

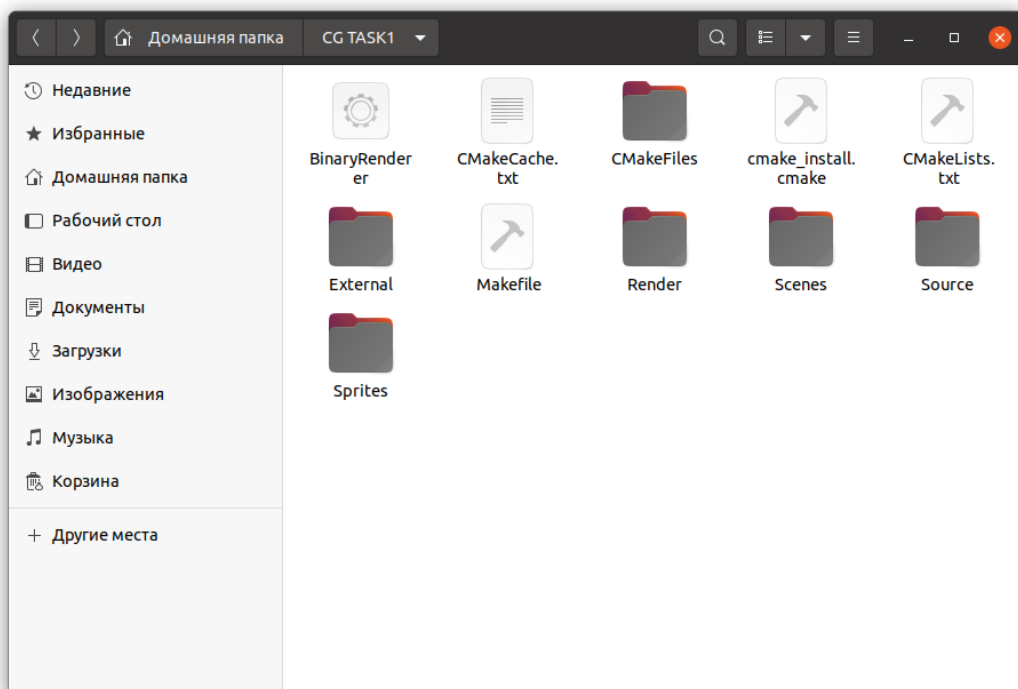
```
cmake CMakeLists.txt
```

Содержимое папки изменится следующим образом:



Для компиляции проекта будем использовать следующую команду:

`make`



Теперь в корне появился исполняемый файл `BinaryRender er`

Который через эту папку без труда можно запустить через команду

`./BinaryRender er`

Теперь мы полностью готовы к написанию кода

2 Критерии оценки

2.1 Основная часть

- 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).

2.2 Дополнительная часть(анализ производительности)

Предполагаемая тактовая частота ГП-1 – 1 КГц (“Стрела” работала на 2 КГц). Требуемое разрешение изображения в буфере кадра – 512x512 пикселей (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. Процессор 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.

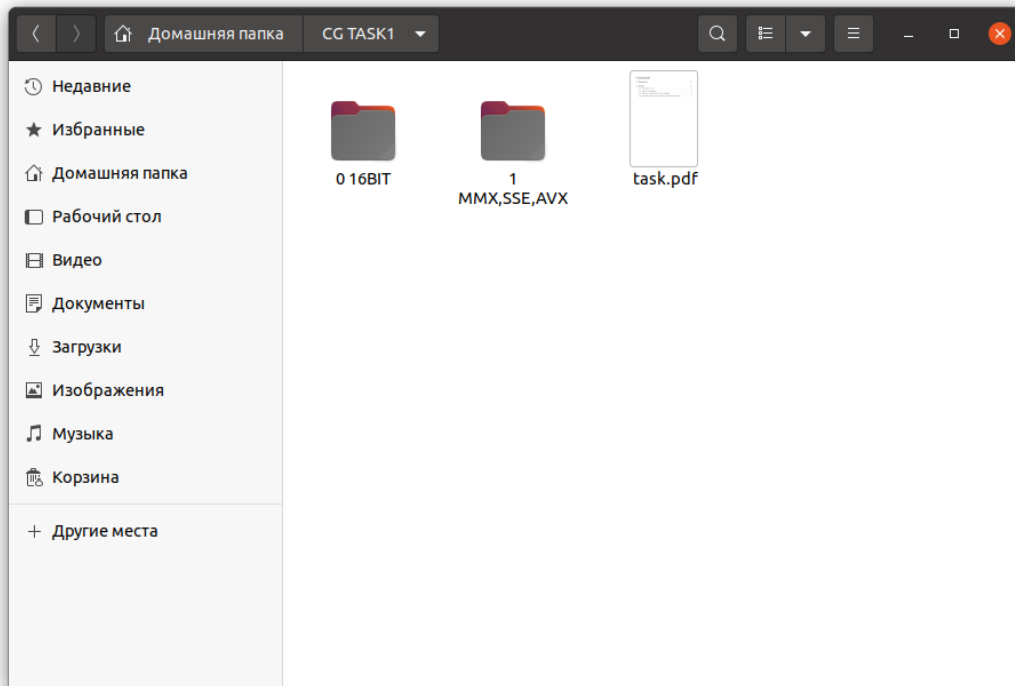
Задача 2 (2–5 баллов) – оценить какое количество транзисторов по- требует аппаратная реализация в 1, 4 и 16, 64 и 256 юнитах. Подсказка: самое точное решение вы получите если реализуете схему на VHDL, проведёте синтез в САПР (например Quartus) и возьмёте из отчёта число логических вентилях, из которых 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.

3 Структура готового приложения

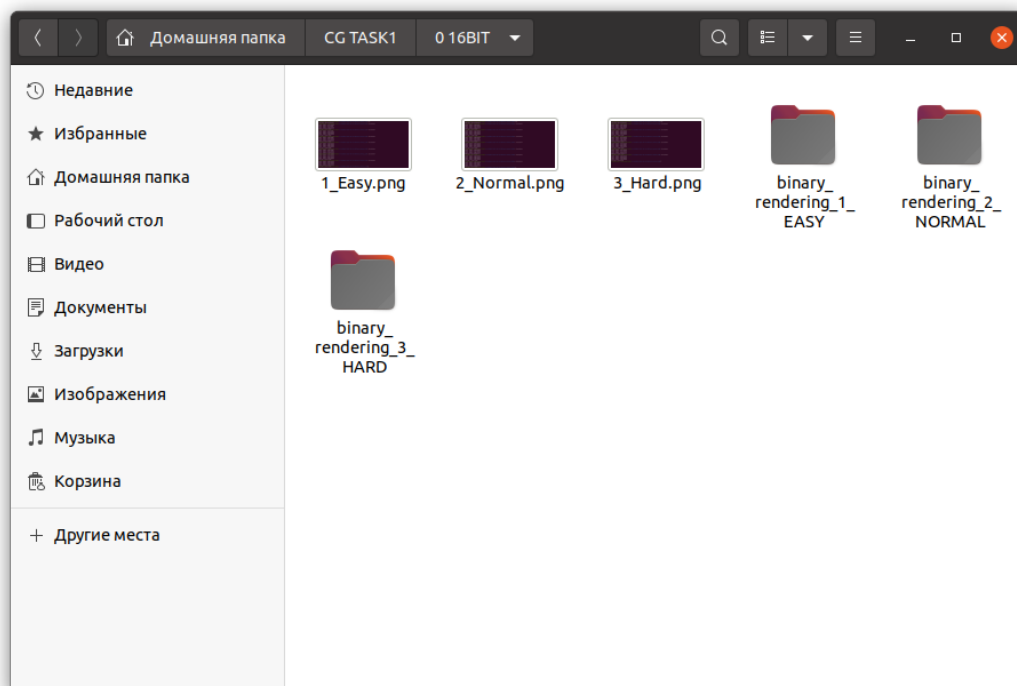
В нашем случае готовое задание было создано и добавлено в архив под названием 415_AT_Aiteyev
В корне содержатся следующие папки:



Папка "0 16BIT" содержит основную часть программы, а именно первые три пункта
Папка "1 MMX,SSE,AVX" содержит выполненную часть последнего пункта основной части
Файл "task.pdf" прилагается к шаблону задания.

3.1 Папка 0 16 BIT

В данной папке содержатся следующие файлы и папки:



Первые три файла - скриншоты выполнения программ.

Папки "EASY", "NORMAL", "HARD" - различные версии программы.

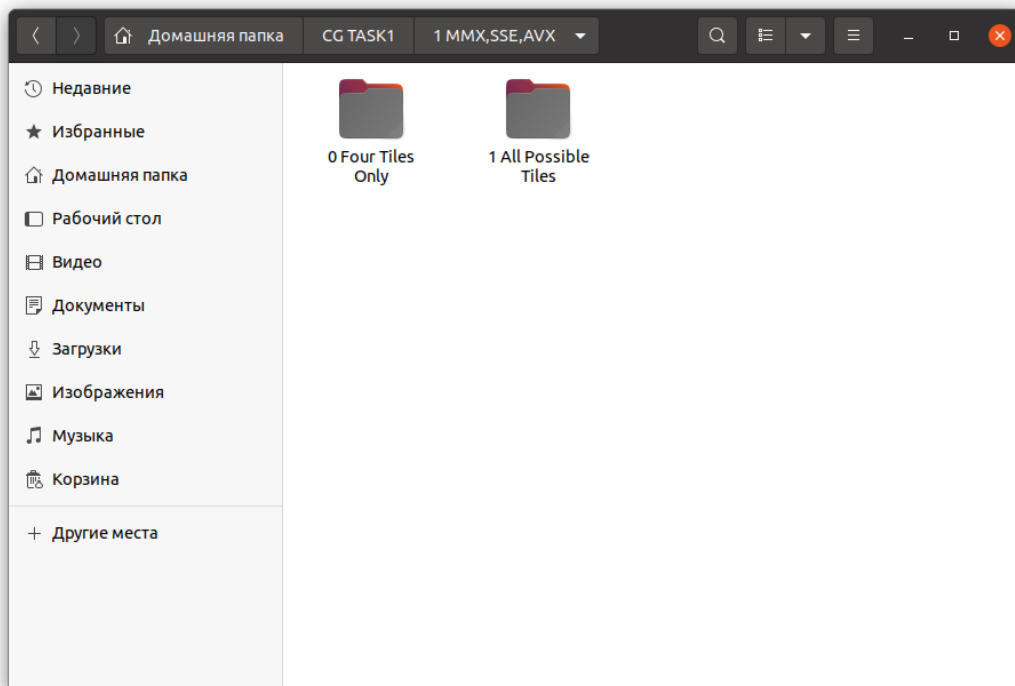
Папка "EASY" содержит самую неоптимизированную версию программы. (так как была написана первой)

Папка "NORMAL" содержит более оптимизированную и интуитивно понятную программу.

Папка "HARD" содержит самую оптимизированную версию программы. (Насколько это возможно без применения ассемблельных вставок, распаралеливания и оптимизации со стороны компилятора)

3.2 Папка 1 MMX,SSE,AVX

В данной папке содержатся следующие папки:



Папка "0 Four Tiles Only" содержит программы, реализованные на MMX, SSE, AVX инструкциях. Особенность реализации в том, что растеризация спрайтов происходит построчно. (4 тайла по оси X, причем, заполняется только 64 бита регистра)

Папка "1 All Possible Tiles" содержит программы, реализованные на MMX, SSE, AVX инструкциях. Особенность реализации в том, что растеризация спрайтов происходит с заполнением всего регистра, в отличие от "0 Four Tiles Only". (В случае MMX заполняется 64-я часть, SSE 32-я часть, AVX 16-я часть спрайта соответственно)

4 Как компилировать и запускать готовое задание

В нашем случае готовое задание было создано и добавлено в архив под названием "415_AT_Aiteyev.zip"

- 1) разархивируем в любое удобное место
- 2) Открываем нужную нам версию программы (например "binary_rendering_1_EASY")
- 3) выполняем набор команд в нужной нам программе

```
cmake CMakeLists.txt
make
```

Для запуска выполним команду

```
./BinaryRenderer
```

в корне программы

5 Подробности реализации приложений

В данной главе мы рассмотрим код каждого приложения. Сразу отметим, что каждое приложение - это модифицированный шаблон, в котором изменены только файлы "GP_ONE", "main.cpp", "CMakeLists.txt".

Сразу заметим, что изменения в "CMakeLists.txt" были произведены только в программах, содержащих инструкции AVX и в приложении "MOD TO ANALYSE".

5.1 Общая структура GP_ONE

Перед просмотром кода каждого приложения, важно понять как работает растеризация спрайта. Для примера возьмем версию приложения "binary_rendering_2_NORMAL"

```
1 | #include "GP_ONE.h"
2 |
3 | Sprite GP_ONE::spriteMemory[MAX_SPRITE_COUNT];
4 | uint16_t GP_ONE::frameBuffer[FRAMEBUFFER_BUF_SIZE];
5 |
6 | void GP_ONE::loadSprites(const Sprite *sprites, uint16_t spriteCount) {
7 |
8 |     // loading sprites into the memory buffer
9 |     for (uint16_t i = 0; i < spriteCount; i++)
10 |         spriteMemory[i] = sprites[i];
11 | }
12 |
13 | void GP_ONE::clearFrameBuffer(BackColorColor bkgColor) {
14 |
15 |     // Clear buffer
16 |     uint16_t BGcolor = (uint16_t) bkgColor ? (uint16_t)(0b1111'1111'1111'1111)
17 |                                     : (uint16_t)(0b0000'0000'0000'0000);
18 |     for (uint16_t i = 0; i < FRAMEBUFFER_BUF_SIZE; i++)
19 |         frameBuffer[i] = BGcolor;
20 | }
21 |
22 | void GP_ONE::saveFrameBuffer(FrameBuffer &outFrameBuffer) {
23 |
24 |     // Put GPU buffer in out buffer
25 |     for (uint16_t i = 0; i < FRAMEBUFFER_BUF_SIZE; i++)
26 |         outFrameBuffer.color[i] = frameBuffer[i];
27 | }
28 |
29 | void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
30 |
31 |     // frameBuffer = (Color & Alpha) | (frameBuffer & ~Alpha)
32 |
33 |     uint16_t Sind ;
34 |     uint16_t Sx;
35 |     uint16_t Sy ;
36 |
37 |     uint16_t shift_tile_X;
38 |     uint16_t shift_coor_X;
39 |
40 |     uint16_t global_And_Sprite_Shift;
41 |     uint16_t Sprite_Shift;
42 |
43 |     uint16_t color;
44 |     uint16_t alpha;
45 |
46 |
47 |
48 |
49 |
50 |
51 |
52 |
53 |
54 |
55 |
56 |
57 |
58 |
59 |
60 |
61 |
62 |
```

```

63 |     for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
64 |
65 |         Sind = instances[currSprite].ind;
66 |         Sx   = instances[currSprite].x;
67 |         Sy   = instances[currSprite].y;
68 |
69 |         shift_tile_X = Sx/uint16_t(16);
70 |         shift_coor_X = Sx%uint16_t(16);
71 |
72 |
73 |         for (uint16_t i = 0; i < SPRITE_TILES_Y; ++i) {
74 |             for (uint16_t j = 0; j < SPRITE_TILES_X; ++j) {
75 |
76 |                 global_And_Sprite_Shift = (i+Sy)*FRAMEBUFFER_TILES_X + (j + shift_tile_X);
77 |                 Sprite_Shift = i*SPRITE_TILES_X + j;
78 |
79 |                 color = spriteMemory[Sind].color[Sprite_Shift] >> shift_coor_X;
80 |                 alpha = spriteMemory[Sind].alpha[Sprite_Shift] >> shift_coor_X;
81 |
82 |                 framebuffer[ global_And_Sprite_Shift ] =
83 |                 color&alpha | framebuffer[ global_And_Sprite_Shift ]&~alpha;
84 |
85 |                 if (shift_coor_X != uint16_t(0)){
86 |                     color =
87 |                     spriteMemory[Sind].color[Sprite_Shift] << (uint16_t(16)-shift_coor_X);
88 |                     alpha =
89 |                     spriteMemory[Sind].alpha[Sprite_Shift] << (uint16_t(16)-shift_coor_X);
90 |
91 |                     framebuffer[global_And_Sprite_Shift + uint16_t(1)] =
92 |                     color&alpha | framebuffer[ global_And_Sprite_Shift + uint16_t(1) ]&~alpha;
93 |                 }
94 |             }
95 |         }
96 |     }
97 | }

```

Листинг 1: Общая структура GP_ONE

Обратим внимание, что абсолютно во всех приложениях изменению подвергся только метод "drawSpriteInstance". Далее будем рассматривать только эту функцию.

В общем случае, функция "drawSpriteInstance" работает следующим образом: в первом цикле мы просматриваем каждый спрайт, запоминаем вспомогательные переменные Sind, Sx, Sy, shift_tile_X, shift_coor_X. shift_tile_X смещает указатель на буфер по оси X. Делим на 16, потому что размер тайла 16. shift_coor_X смещает побитово спрайт между двумя элементами буфера. Она необходима, если координата спрайта не кратна 16-ти.

Двойной цикл нужен чтобы пройти по каждому тайлу спрайта; вычисляем global_And_Sprite_Shift - сдвиг указателя на буфер, Sprite_Shift - сдвиг указателя на спрайт. переменные color и alpha содержат в себе цвета тайла текущего спрайта; затем записываем в буфер по формуле

$$frameBuffer = (Color \& Alpha) | (frameBuffer \& \sim Alpha).$$

Если Sx не кратен 16, то выполняется условие shift_coor_X != uint16_t(0), следовательно нужно заполнить соседний тайл буфера оставшейся частью тайла спрайта (именно поэтому мы и используем побитовые сдвиги).

5.2 Папка 0 16BIT

Для точного вычисления времени в каждом приложении, в файле "main.cpp" переменная TEST_ITERATIONS равна 10000. Это единственное изменение в файле "main.cpp" во всех трех программах.

5.2.1 Программа binary_rendering_1_EASY

Данная программа была написана первой, поэтому её алгоритм не выглядит столь эргономично как в binary_rendering_2_NORMAL:

```
1  uint16_t res(uint16_t buf, uint16_t color, uint16_t alpha){
2
3      return buf = color & alpha | buf & ~alpha;
4  }
5
6  void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
7
8      // frameBuffer = (Color & Alpha) | (frameBuffer & ~Alpha)
9      for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
10
11          uint16_t Sind = instances[currSprite].ind;
12          uint16_t Sx   = instances[currSprite].x;
13          uint16_t Sy   = instances[currSprite].y;
14
15          for (uint16_t i = 0; i < SPRITE_TILES_Y; ++i) {
16              for (uint16_t j = 0; j < SPRITE_TILES_X; ++j) {
17
18                  if (Sx%16 != 0){
19                      uint16_t xx1 = frameBuffer[(i+Sy)*FRAMEBUFFER_TILES_X + j + Sx/16 + 0];
20                      uint16_t xx2 = frameBuffer[(i+Sy)*FRAMEBUFFER_TILES_X + j + Sx/16 + 1];
21
22                      uint16_t yy1 = spriteMemory[Sind].color[i*SPRITE_TILES_X + j] >> Sx%16;
23                      uint16_t yy2 = spriteMemory[Sind].color[i*SPRITE_TILES_X + j] << (16-Sx%16);
24
25                      uint16_t zz1 = spriteMemory[Sind].alpha[i*SPRITE_TILES_X + j] >> Sx%16;
26                      uint16_t zz2 = spriteMemory[Sind].alpha[i*SPRITE_TILES_X + j] << (16-Sx%16);
27
28                      frameBuffer[(i+Sy)*FRAMEBUFFER_TILES_X + j + Sx/16 + 0] = res(xx1,yy1,zz1);
29                      frameBuffer[(i+Sy)*FRAMEBUFFER_TILES_X + j + Sx/16 + 1] = res(xx2,yy2,zz2);
30                  }
31
32                  if (Sx%16 == 0){
33                      uint16_t xx = frameBuffer[(i+Sy)*FRAMEBUFFER_TILES_X + j + Sx/16];
34                      uint16_t yy = spriteMemory[Sind].color[i*SPRITE_TILES_X + j];
35                      uint16_t zz = spriteMemory[Sind].alpha[i*SPRITE_TILES_X + j];
36
37                      frameBuffer[(i+Sy)*FRAMEBUFFER_TILES_X + j + Sx/16] = res(xx, yy, zz);
38                  }
39              }
40          }
41      }
42  }
43 }
```

5.2.2 Программа binary_rendering_2_NORMAL

Данная программа приобрела более лаконичный вид, благодаря тому что мы убрали сложные структуры и ввели больше переменных, чтобы значительно сократить количество вычислений.

```
1 void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
2
3     // framebuffer = (Color & Alpha) | (framebuffer & ~Alpha)
4
5     uint16_t Sind ;
6     uint16_t Sx;
7     uint16_t Sy ;
8
9     uint16_t shift_tile_X = Sx/16;
10    uint16_t shift_coor_X = Sx%16;
11
12    uint16_t global_And_Sprite_Shift;
13    uint16_t Sprite_Shift;
14
15    uint16_t color;
16    uint16_t alpha;
17
18    for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
19
20        Sind = instances[currSprite].ind;
21        Sx   = instances[currSprite].x;
22        Sy   = instances[currSprite].y;
23
24        shift_tile_X = Sx/16;
25        shift_coor_X = Sx%16;
26
27
28        for (uint16_t i = 0; i < SPRITE_TILES_Y; ++i) {
29            for (uint16_t j = 0; j < SPRITE_TILES_X; ++j) {
30
31                global_And_Sprite_Shift = (i+Sy)*FRAMEBUFFER_TILES_X + (j + shift_tile_X);
32                Sprite_Shift = i*SPRITE_TILES_X + j;
33
34                color = spriteMemory[Sind].color[Sprite_Shift] >> shift_coor_X;
35                alpha = spriteMemory[Sind].alpha[Sprite_Shift] >> shift_coor_X;
36
37                framebuffer[ global_And_Sprite_Shift ] =
38                color&alpha | framebuffer[ global_And_Sprite_Shift ]&~alpha;
39
40                if (shift_coor_X != uint16_t(0)){
41                    color =
42                    spriteMemory[Sind].color[Sprite_Shift] << (uint16_t(16)-shift_coor_X);
43                    alpha =
44                    spriteMemory[Sind].alpha[Sprite_Shift] << (uint16_t(16)-shift_coor_X);
45
46                    framebuffer[global_And_Sprite_Shift + 1] =
47                    color&alpha | framebuffer[ global_And_Sprite_Shift + 1 ]&~alpha;
48                }
49            }
50        }
51    }
52 }
53 }
```

5.2.3 Программа binary_rendering_3_HARD

Данная программа является модификацией предыдущей. Мы удалили несколько переменных, так как операция присваивания выполняется дольше, чем арифметическая операция, и несоразмерно создавать переменные, которые будут применяться лишь единожды, или время создание переменной будет дольше чем выполнение аналогичной операции, заменили умножение и деление на работу со сдвигами. Так же заметим, что создание переменных заполняет стек и, очевидно, что обращение к нему выполняется быстрее чем обращение к оперативной памяти.

```
1 void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
2
3     // frameBuffer = (Color & Alpha) | (frameBuffer & ~Alpha)
4     uint16_t Sind ;
5
6     uint16_t shift_tile_X;
7     uint16_t shift_coor_X;
8     uint16_t rev_Shift_coor_X;
9
10    uint16_t global_And_Sprite_Shift;
11    uint16_t Sprite_Shift;
12
13    uint16_t color;
14    uint16_t alpha;
15
16    for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
17
18        Sind = instances[currSprite].ind;
19
20        shift_tile_X = instances[currSprite].x >> uint16_t(4);
21        shift_coor_X = instances[currSprite].x & uint16_t(15);
22
23        if (shift_coor_X != uint16_t(0)){
24            rev_Shift_coor_X = uint16_t(16)-shift_coor_X;
25        }
26
27        for (uint16_t i = 0; i < SPRITE_TILES_Y; ++i) {
28
29            global_And_Sprite_Shift = ((i+instances[currSprite].y)<<5) + (shift_tile_X);
30            Sprite_Shift = i<<2 ;
31
32            for (uint16_t j = 0; j < SPRITE_TILES_X; ++j) {
33
34                color = spriteMemory[Sind].color[Sprite_Shift+j] >> shift_coor_X;
35                alpha = spriteMemory[Sind].alpha[Sprite_Shift+j] >> shift_coor_X;
36
37                framebuffer[ global_And_Sprite_Shift +j] =
38                color&alpha | framebuffer[ global_And_Sprite_Shift +j]&~alpha;
39
40                if (shift_coor_X != uint16_t(0)){
41                    color =
42                    spriteMemory[Sind].color[Sprite_Shift+j] << rev_Shift_coor_X;
43                    alpha =
44                    spriteMemory[Sind].alpha[Sprite_Shift+j] << rev_Shift_coor_X;
45
46                    framebuffer[global_And_Sprite_Shift+1+j] =
47                    color&alpha | framebuffer[global_And_Sprite_Shift+1+j]&~alpha;
48                }
49            }
50        }
51    }
52 }
53 }
```

5.3 Папка 1 MMX,SSE,AVX/0 Four Tiles Only

За основу всех программ была взята версия NORMAL, так как она более интуитивно понятная.

Кроме того мы изменили переменную TEST_ITERATIONS с 1000 на 100 в каждой программе в папке. Это нам пригодится впредь при анализе производительности обычной версии и инструкций процессора.

Обратим особое внимание на эту часть кода:

```
1 | for (uint16_t k = 0; k < 100; ++k)
2 |     result_m64 = _m_por(_mm_andnot_si64(alpha_m64, buffer_m64),
                        _mm_and_si64(alpha_m64, color_m64));
```

В некоторых программах она может вызывать другие функции, в зависимости от применяемых инструкций, однако основная суть сохраняется. Данный цикл будет необходим при изучении эффективности алгоритма по отношению к обычной, 16-ти битной версии.

5.3.1 Программа 0_binary_rendering_2_NORMAL

```
1 | void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
2 |     // frameBuffer = (Color & Alpha) | (frameBuffer & ~Alpha)
3 |     uint16_t Sind ;
4 |     uint16_t Sx;
5 |     uint16_t Sy ;
6 |
7 |     uint16_t shift_tile_X = Sx/16;
8 |     uint16_t shift_coor_X = Sx%16;
9 |
10 |    uint16_t global_And_Sprite_Shift;
11 |    uint16_t Sprite_Shift;
12 |
13 |    uint16_t color;
14 |    uint16_t alpha;
15 |
16 |    for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
17 |
18 |        Sind = instances[currSprite].ind;
19 |        Sx   = instances[currSprite].x;
20 |        Sy   = instances[currSprite].y;
21 |
22 |        shift_tile_X = Sx/16;
23 |        shift_coor_X = Sx%16;
24 |
25 |        for (uint16_t i = 0; i < SPRITE_TILES_Y; ++i) {
26 |            for (uint16_t j = 0; j < SPRITE_TILES_X; ++j) {
27 |
28 |                global_And_Sprite_Shift = (i+Sy)*FRAMEBUFFER_TILES_X + (j + shift_tile_X);
29 |                Sprite_Shift = i*SPRITE_TILES_X + j;
30 |
31 |                color = spriteMemory[Sind].color[Sprite_Shift] >> shift_coor_X;
32 |                alpha = spriteMemory[Sind].alpha[Sprite_Shift] >> shift_coor_X;
33 |
34 |                for (uint16_t k = 0; k < 1; ++k)
35 |                    frameBuffer[ global_And_Sprite_Shift ] =
36 |                        color&alpha | frameBuffer[ global_And_Sprite_Shift ]&~alpha;
37 |
38 |                if (shift_coor_X != uint16_t(0)){
39 |                    color =
40 |                        spriteMemory[Sind].color[Sprite_Shift] << (uint16_t(16)-shift_coor_X);
41 |                    alpha =
42 |                        spriteMemory[Sind].alpha[Sprite_Shift] << (uint16_t(16)-shift_coor_X);
43 |
44 |                    for (uint16_t k = 0; k < 1; ++k)
45 |                        frameBuffer[global_And_Sprite_Shift + 1] =
46 |                            color&alpha | frameBuffer[ global_And_Sprite_Shift + 1 ]&~alpha;
47 |                }
48 |            }
49 |        }
50 |    }
51 | }
52 | }
```


5.3.2 Программа 1_binary_rendering_2_NORMAL_MMX

```
1 void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
2
3     // framebuffer = (Color & Alpha) / (framebuffer & ~Alpha)
4
5     uint16_t Sind ;
6     uint16_t Sx   ;
7     uint16_t Sy   ;
8
9     uint16_t shift_tile_X;
10    uint16_t shift_coor_X ;
11
12    uint16_t global_And_Sprite_Shift;
13    uint16_t Sprite_Shift;
14
15    __m64 color_m64;
16    __m64 alpha_m64 ;
17    __m64 buffer_m64 ;
18    __m64 result_m64 ;
19
20    uint16_t* result = (uint16_t*)&result_m64;
21
22
23    for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
24
25        Sind = instances[currSprite].ind;
26        Sx   = instances[currSprite].x;
27        Sy   = instances[currSprite].y;
28
29        shift_tile_X = Sx/uint16_t(16);
30        shift_coor_X = Sx%uint16_t(16);
31
32
33        for (uint16_t i = 0; i < SPRITE_TILES_Y; ++i) {
34
35            //SPRITE_TILES_X==4
36            global_And_Sprite_Shift = (i+Sy)*FRAMEBUFFER_TILES_X + (shift_tile_X);
37            Sprite_Shift = i*SPRITE_TILES_X;
38
39            color_m64 = _mm_set_pi16(
40                spriteMemory[Sind].color[Sprite_Shift+3] >> shift_coor_X,
41                spriteMemory[Sind].color[Sprite_Shift+2] >> shift_coor_X,
42                spriteMemory[Sind].color[Sprite_Shift+1] >> shift_coor_X,
43                spriteMemory[Sind].color[Sprite_Shift  ] >> shift_coor_X);
44
45            alpha_m64 = _mm_set_pi16(
46                spriteMemory[Sind].alpha[Sprite_Shift+3] >> shift_coor_X,
47                spriteMemory[Sind].alpha[Sprite_Shift+2] >> shift_coor_X,
48                spriteMemory[Sind].alpha[Sprite_Shift+1] >> shift_coor_X,
49                spriteMemory[Sind].alpha[Sprite_Shift  ] >> shift_coor_X);
50
51            buffer_m64 = _mm_set_pi16(
52                framebuffer[global_And_Sprite_Shift+3],
53                framebuffer[global_And_Sprite_Shift+2],
54                framebuffer[global_And_Sprite_Shift+1],
55                framebuffer[global_And_Sprite_Shift  ]);
56
57            for (uint16_t k = 0; k < 100; ++k)
58                result_m64 = _m_por(_mm_andnot_si64(alpha_m64, buffer_m64),
59                                    _mm_and_si64(alpha_m64, color_m64));
60
61            for (uint16_t j = 0; j < SPRITE_TILES_X; ++j)
62                framebuffer[global_And_Sprite_Shift+j] = result[j];
63
64
65            if (shift_coor_X != uint16_t(0)){
66
67                color_m64 = _mm_set_pi16(
68                    spriteMemory[Sind].color[Sprite_Shift+3] << (uint16_t(16)-shift_coor_X),
69                    spriteMemory[Sind].color[Sprite_Shift+2] << (uint16_t(16)-shift_coor_X),
70                    spriteMemory[Sind].color[Sprite_Shift+1] << (uint16_t(16)-shift_coor_X),
71                    spriteMemory[Sind].color[Sprite_Shift  ] << (uint16_t(16)-shift_coor_X));
72
73            }
```

```

74         alpha_m64 = _mm_set_pi16(
75             spriteMemory[Sind].alpha[Sprite_Shift+3] << (uint16_t(16)-shift_coor_X),
76             spriteMemory[Sind].alpha[Sprite_Shift+2] << (uint16_t(16)-shift_coor_X),
77             spriteMemory[Sind].alpha[Sprite_Shift+1] << (uint16_t(16)-shift_coor_X),
78             spriteMemory[Sind].alpha[Sprite_Shift] << (uint16_t(16)-shift_coor_X));
79
80         buffer_m64 = _mm_set_pi16(
81             framebuffer[global_And_Sprite_Shift+4],
82             framebuffer[global_And_Sprite_Shift+3],
83             framebuffer[global_And_Sprite_Shift+2],
84             framebuffer[global_And_Sprite_Shift+1]);
85
86         for (uint16_t k = 0; k < 100; ++k)
87             result_m64 = _m_por(_mm_andnot_si64(alpha_m64, buffer_m64),
88                                 _mm_and_si64(alpha_m64, color_m64));
89
90         for (uint16_t j = 0; j < SPRITE_TILES_X; ++j)
91             framebuffer[global_And_Sprite_Shift+j+uint16_t(1)] = result[j];
92     }
93 }
94 }
95 }
96
97 }

```

5.3.3 Программа 2_binary_rendering_2_NORMAL_SSE

```

1  void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
2
3      // framebuffer = (Color & Alpha) / (framebuffer & ~Alpha)
4      uint16_t Sind ;
5      uint16_t Sx ;
6      uint16_t Sy ;
7
8      uint16_t shift_tile_X;
9      uint16_t shift_coor_X ;
10
11     uint16_t global_And_Sprite_Shift;
12     uint16_t Sprite_Shift;
13
14     __m128i color_m64;
15     __m128i alpha_m64 ;
16     __m128i buffer_m64 ;
17     __m128i result_m64 ;
18
19     uint16_t* result = (uint16_t*)&result_m64;
20
21     for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
22
23         Sind = instances[currSprite].ind;
24         Sx = instances[currSprite].x;
25         Sy = instances[currSprite].y;
26
27         shift_tile_X = Sx/uint16_t(16);
28         shift_coor_X = Sx%uint16_t(16);
29
30         for (uint16_t i = 0; i < SPRITE_TILES_Y; ++i) {
31
32             global_And_Sprite_Shift = (i+Sy)*FRAMEBUFFER_TILES_X + (shift_tile_X);
33             Sprite_Shift = i*SPRITE_TILES_X;
34
35             color_m64 = _mm_set_epi16( 0,0,0,0,
36                 spriteMemory[Sind].color[Sprite_Shift+3] >> shift_coor_X,
37                 spriteMemory[Sind].color[Sprite_Shift+2] >> shift_coor_X,
38                 spriteMemory[Sind].color[Sprite_Shift+1] >> shift_coor_X,
39                 spriteMemory[Sind].color[Sprite_Shift] >> shift_coor_X);
40
41
42
43
44

```

```

45     alpha_m64 = _mm_set_epi16( 0,0,0,0,
46     spriteMemory[Sind].alpha[Sprite_Shift+3] >> shift_coor_X,
47     spriteMemory[Sind].alpha[Sprite_Shift+2] >> shift_coor_X,
48     spriteMemory[Sind].alpha[Sprite_Shift+1] >> shift_coor_X,
49     spriteMemory[Sind].alpha[Sprite_Shift  ] >> shift_coor_X);
50
51     buffer_m64 = _mm_set_epi16(0,0,0,0,
52     framebuffer[global_And_Sprite_Shift+3],
53     framebuffer[global_And_Sprite_Shift+2],
54     framebuffer[global_And_Sprite_Shift+1],
55     framebuffer[global_And_Sprite_Shift  ]);
56
57     for (uint16_t k = 0; k < 100; ++k)
58     result_m64 = _mm_or_si128( _mm_andnot_si128(alpha_m64, buffer_m64),
59                               _mm_and_si128(alpha_m64, color_m64));
60
61     for (uint16_t j = 0; j < SPRITE_TILES_X; ++j)
62         framebuffer[global_And_Sprite_Shift+j] = result[j];
63
64
65
66     if (shift_coor_X != uint16_t(0)){
67
68         color_m64 = _mm_set_epi16( 0,0,0,0,
69         spriteMemory[Sind].color[Sprite_Shift+3] << (uint16_t(16)-shift_coor_X),
70         spriteMemory[Sind].color[Sprite_Shift+2] << (uint16_t(16)-shift_coor_X),
71         spriteMemory[Sind].color[Sprite_Shift+1] << (uint16_t(16)-shift_coor_X),
72         spriteMemory[Sind].color[Sprite_Shift  ] << (uint16_t(16)-shift_coor_X));
73
74         alpha_m64 = _mm_set_epi16( 0,0,0,0,
75         spriteMemory[Sind].alpha[Sprite_Shift+3] << (uint16_t(16)-shift_coor_X),
76         spriteMemory[Sind].alpha[Sprite_Shift+2] << (uint16_t(16)-shift_coor_X),
77         spriteMemory[Sind].alpha[Sprite_Shift+1] << (uint16_t(16)-shift_coor_X),
78         spriteMemory[Sind].alpha[Sprite_Shift  ] << (uint16_t(16)-shift_coor_X));
79
80         buffer_m64 = _mm_set_epi16( 0,0,0,0,
81         framebuffer[global_And_Sprite_Shift+uint16_t(4)],
82         framebuffer[global_And_Sprite_Shift+uint16_t(3)],
83         framebuffer[global_And_Sprite_Shift+uint16_t(2)],
84         framebuffer[global_And_Sprite_Shift+uint16_t(1)]);
85
86         for (uint16_t k = 0; k < 100; ++k)
87         result_m64 = _mm_or_si128( _mm_andnot_si128(alpha_m64, buffer_m64),
88                                   _mm_and_si128(alpha_m64, color_m64));
89
90
91         for (uint16_t j = 0; j < SPRITE_TILES_X; ++j)
92             framebuffer[global_And_Sprite_Shift+j+uint16_t(1)] = result[j];
93
94     }
95 }
96 }
97 }

```

5.3.4 Программа 3_binary_rendering_2_NORMAL_AVX

[illegible]

```
74     buffer_m64 = _mm256_set_epi16(0,0,0,0,0,0,0,0,0,0,0,0,  
75     frameBuffer[global_And_Sprite_Shift+uint16_t(4)],  
76     frameBuffer[global_And_Sprite_Shift+uint16_t(3)],  
77     frameBuffer[global_And_Sprite_Shift+uint16_t(2)],  
78     frameBuffer[global_And_Sprite_Shift+uint16_t(1)]);  
79  
80     for (uint16_t k = 0; k < 100; ++k)  
81         result_m64 = _mm256_or_si256(_mm256_andnot_si256(alpha_m64, buffer_m64),  
82                                     _mm256_and_si256(alpha_m64, color_m64));  
83  
84  
85     for (uint16_t j = 0; j < SPRITE_TILES_X; ++j)  
86         frameBuffer[global_And_Sprite_Shift+j+uint16_t(1)] = result[j];  
87  
88 }  
89 }  
90 }  
91 }
```

5.3.5 Программа MOD TO ANALYSE

Данная программа отличается существенно от четырех предыдущих, так как содержит в себе код четырех программ одновременно. Заголовочные файлы GP_ONE.h и GP_ONE.cpp переименованы в "GP_ONE_0" , "GP_ONE_1" , "GP_ONE_2" , "GP_ONE_3" в соответствии с отсутствием инструкций, MMX, SSE и AVX соответственно.

В CMakeLists.txt введены следующие изменения:

1) Добавлена строка для компиляции инструкций AVX

```
1 || SET(CMAKE_CXX_FLAGS "-mavx2")
```

2) Добавлены хедеры и файлы предыдущих программ для компилирования

```

1 | add_executable(BinaryRenderer
2 |     Source/main.cpp
3 |
4 |     Source/GP_ONE_0.h
5 |     Source/GP_ONE_0.cpp
6 |     Source/GP_ONE_1.h
7 |     Source/GP_ONE_1.cpp
8 |     Source/GP_ONE_2.h
9 |     Source/GP_ONE_2.cpp
10 |    Source/GP_ONE_3.h
11 |    Source/GP_ONE_3.cpp
12 |
13 |    Source/Sprite.h
14 |    Source/Sprite.cpp
15 |    Source/FrameBuffer.h
16 |    Source/FrameBuffer.cpp
17 |    Source/SpriteInstance.h
18 |    Source/Scene.h
19 |
20 |    External/Clock.h
21 |    External/Clock.cpp
22 |    External/ImageManager.h
23 |    External/ImageManager.cpp
24 |    External/SceneLoader.h
25 |    External/SceneLoader.cpp
26 | )

```

Сильные изменения притерпел файл `main.cpp`, так как нужно было включить сразу несколько новых хедеров и, кроме того, записывать время выполнения каждого GPU в отдельные файлы "Result0" , "Result1" , "Result2" , "Result3".

Для анализа введены дополнительные глобальные переменные:

```
1 | const int TEST_ITERATIONS = 100;  
2 | const int COMMAND_SIZE = 100;  
3 | const int GPUselect = 3;
```

COMMAND_SIZE отвечает сколько раз выполнится команда

$$frameBuffer = (Color \& Alpha) | (frameBuffer \& Alpha)$$

К примеру, в примере ниже COMMAND_SIZE равен 100.

```
1 | for (uint16_t k = 0; k < 100; ++k)  
2 |     result_m64 = _m_por(_mm_andnot_si64(alpha_m64, buffer_m64),  
                           _mm_and_si64(alpha_m64, color_m64));
```

GPUselect переменная, от значения которой зависит выбор нужного GPU для анализа

Так же написан небольшой скрипт plot.py, которое визуализирует полученные данные. Подробнее будет описано в следующей главе.

5.4 Папка 1 MMX,SSE,AVX/1 All Possible Tiles

За основу всех програм была взята версия NORMAL, так как она более интуитивно понятная. Кроме того мы изменили переменную TEST_ITERATIONS с 1000 на 100 в каждой программе в папке. Это нам пригодится впредь при анализе производительности обычной версии и инструкций процессора.

5.4.1 Программа 0_binary_rendering_2_NORMAL

```
1 void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
2
3     // framebuffer = (Color & Alpha) | (framebuffer & ~Alpha)
4
5     uint16_t Sind ;
6     uint16_t Sx   ;
7     uint16_t Sy   ;
8
9     uint16_t shift_tile_X;
10    uint16_t shift_coor_X ;
11
12    uint16_t global_And_Sprite_Shift;
13    uint16_t Sprite_Shift;
14
15    __m64 color_m64;
16    __m64 alpha_m64 ;
17    __m64 buffer_m64 ;
18    __m64 result_m64 ;
19
20    uint16_t* result = (uint16_t*)&result_m64;
21
22
23    for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
24
25        Sind = instances[currSprite].ind;
26        Sx   = instances[currSprite].x;
27        Sy   = instances[currSprite].y;
28
29        shift_tile_X = Sx/uint16_t(16);
30        shift_coor_X = Sx%uint16_t(16);
31
32
33        for (uint16_t i = 0; i < SPRITE_TILES_Y; ++i) {
34
35            //SPRITE_TILES_X==4
36            global_And_Sprite_Shift = (i+Sy)*FRAMEBUFFER_TILES_X + (shift_tile_X);
37            Sprite_Shift = i*SPRITE_TILES_X;
38
39            color_m64 = _mm_set_pi16(
40                spriteMemory[Sind].color[Sprite_Shift+3] >> shift_coor_X,
41                spriteMemory[Sind].color[Sprite_Shift+2] >> shift_coor_X,
42                spriteMemory[Sind].color[Sprite_Shift+1] >> shift_coor_X,
43                spriteMemory[Sind].color[Sprite_Shift  ] >> shift_coor_X);
44
45            alpha_m64 = _mm_set_pi16(
46                spriteMemory[Sind].alpha[Sprite_Shift+3] >> shift_coor_X,
47                spriteMemory[Sind].alpha[Sprite_Shift+2] >> shift_coor_X,
48                spriteMemory[Sind].alpha[Sprite_Shift+1] >> shift_coor_X,
49                spriteMemory[Sind].alpha[Sprite_Shift  ] >> shift_coor_X);
50
51            buffer_m64 = _mm_set_pi16(
52                framebuffer[global_And_Sprite_Shift+3],
53                framebuffer[global_And_Sprite_Shift+2],
54                framebuffer[global_And_Sprite_Shift+1],
55                framebuffer[global_And_Sprite_Shift  ]);
56
57            for (uint16_t k = 0; k < 100; ++k)
58                result_m64 = _m_por(_mm_andnot_si64(alpha_m64, buffer_m64),
59                                    _mm_and_si64(alpha_m64, color_m64));
60
61            for (uint16_t j = 0; j < SPRITE_TILES_X; ++j)
62                framebuffer[global_And_Sprite_Shift+j] = result[j];
63
64
65            if (shift_coor_X != uint16_t(0)){
66
67                color_m64 = _mm_set_pi16(
68                    spriteMemory[Sind].color[Sprite_Shift+3] << (uint16_t(16)-shift_coor_X),
69                    spriteMemory[Sind].color[Sprite_Shift+2] << (uint16_t(16)-shift_coor_X),
70                    spriteMemory[Sind].color[Sprite_Shift+1] << (uint16_t(16)-shift_coor_X),
71                    spriteMemory[Sind].color[Sprite_Shift  ] << (uint16_t(16)-shift_coor_X));
72
73            }
```

```

74         alpha_m64 = _mm_set_pi16(
75             spriteMemory[Sind].alpha[Sprite_Shift+3] << (uint16_t(16)-shift_coor_X),
76             spriteMemory[Sind].alpha[Sprite_Shift+2] << (uint16_t(16)-shift_coor_X),
77             spriteMemory[Sind].alpha[Sprite_Shift+1] << (uint16_t(16)-shift_coor_X),
78             spriteMemory[Sind].alpha[Sprite_Shift] << (uint16_t(16)-shift_coor_X));
79
80         buffer_m64 = _mm_set_pi16(
81             framebuffer[global_And_Sprite_Shift+4],
82             framebuffer[global_And_Sprite_Shift+3],
83             framebuffer[global_And_Sprite_Shift+2],
84             framebuffer[global_And_Sprite_Shift+1]);
85
86         for (uint16_t k = 0; k < 100; ++k)
87             result_m64 = _m_por(_mm_andnot_si64(alpha_m64, buffer_m64),
88                                 _mm_and_si64(alpha_m64, color_m64));
89
90         for (uint16_t j = 0; j < SPRITE_TILES_X; ++j)
91             framebuffer[global_And_Sprite_Shift+j+uint16_t(1)] = result[j];
92     }
93 }
94 }
95 }
96 }
97 }

```

5.4.2 Программа 1_binary_rendering_2_NORMAL_MMX

```

1 void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
2
3     // framebuffer = (Color & Alpha) | (framebuffer & ~Alpha)
4
5     uint16_t Sind ;
6     uint16_t Sx ;
7     uint16_t Sy ;
8
9     uint16_t shift_tile_X;
10    uint16_t shift_coor_X ;
11
12    uint16_t global_And_Sprite_Shift;
13    uint16_t Sprite_Shift;
14
15    __m64 color_m64;
16    __m64 alpha_m64 ;
17    __m64 buffer_m64 ;
18    __m64 result_m64 ;
19
20    uint16_t* result = (uint16_t*)&result_m64;
21
22    for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
23
24        Sind = instances[currSprite].ind;
25        Sx = instances[currSprite].x;
26        Sy = instances[currSprite].y;
27
28        shift_tile_X = Sx/uint16_t(16);
29        shift_coor_X = Sx%uint16_t(16);
30
31        for (uint16_t i = 0; i < SPRITE_TILES_Y; ++i) {
32
33            global_And_Sprite_Shift = (i+Sy)*FRAMEBUFFER_TILES_X + (shift_tile_X);
34            Sprite_Shift = i*SPRITE_TILES_X;
35
36            color_m64 = _mm_set_pi16(
37                spriteMemory[Sind].color[Sprite_Shift+3] >> shift_coor_X,
38                spriteMemory[Sind].color[Sprite_Shift+2] >> shift_coor_X,
39                spriteMemory[Sind].color[Sprite_Shift+1] >> shift_coor_X,
40                spriteMemory[Sind].color[Sprite_Shift] >> shift_coor_X);
41
42
43
44

```



```

45     alpha_m64 = _mm_set_pi16(
46         spriteMemory[Sind].alpha[Sprite_Shift+3] >> shift_coor_X,
47         spriteMemory[Sind].alpha[Sprite_Shift+2] >> shift_coor_X,
48         spriteMemory[Sind].alpha[Sprite_Shift+1] >> shift_coor_X,
49         spriteMemory[Sind].alpha[Sprite_Shift] >> shift_coor_X);
50
51     buffer_m64 = _mm_set_pi16(
52         framebuffer[global_And_Sprite_Shift+3],
53         framebuffer[global_And_Sprite_Shift+2],
54         framebuffer[global_And_Sprite_Shift+1],
55         framebuffer[global_And_Sprite_Shift]);
56
57     for (uint16_t k = 0; k < 1; ++k)
58         result_m64 = _m_por(_mm_andnot_si64(alpha_m64, buffer_m64),
59                             _mm_and_si64(alpha_m64, color_m64));
60
61     for (uint16_t j = 0; j < SPRITE_TILES_X; ++j)
62         framebuffer[global_And_Sprite_Shift+j] = result[j];
63
64     if (shift_coor_X != uint16_t(0)){
65
66         color_m64 = _mm_set_pi16(
67             spriteMemory[Sind].color[Sprite_Shift+3] << (uint16_t(16)-shift_coor_X),
68             spriteMemory[Sind].color[Sprite_Shift+2] << (uint16_t(16)-shift_coor_X),
69             spriteMemory[Sind].color[Sprite_Shift+1] << (uint16_t(16)-shift_coor_X),
70             spriteMemory[Sind].color[Sprite_Shift] << (uint16_t(16)-shift_coor_X));
71
72         alpha_m64 = _mm_set_pi16(
73             spriteMemory[Sind].alpha[Sprite_Shift+3] << (uint16_t(16)-shift_coor_X),
74             spriteMemory[Sind].alpha[Sprite_Shift+2] << (uint16_t(16)-shift_coor_X),
75             spriteMemory[Sind].alpha[Sprite_Shift+1] << (uint16_t(16)-shift_coor_X),
76             spriteMemory[Sind].alpha[Sprite_Shift] << (uint16_t(16)-shift_coor_X));
77
78         buffer_m64 = _mm_set_pi16(
79             framebuffer[global_And_Sprite_Shift+4],
80             framebuffer[global_And_Sprite_Shift+3],
81             framebuffer[global_And_Sprite_Shift+2],
82             framebuffer[global_And_Sprite_Shift+1]);
83
84         for (uint16_t k = 0; k < 1; ++k)
85             result_m64 = _m_por(_mm_andnot_si64(alpha_m64, buffer_m64),
86                                 _mm_and_si64(alpha_m64, color_m64));
87
88         for (uint16_t j = 0; j < SPRITE_TILES_X; ++j)
89             framebuffer[global_And_Sprite_Shift+j+uint16_t(1)] = result[j];
90
91     }
92 }
93 }
94
95 }

```

5.4.3 Программа 2_binary_rendering_2_NORMAL_SSE

```

1  void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
2
3      // framebuffer = (Color & Alpha) | (frameBuffer & ~Alpha)
4
5      uint16_t Sind ;
6      uint16_t Sx ;
7      uint16_t Sy ;
8
9      uint16_t shift_tile_X;
10     uint16_t shift_coor_X ;
11
12     uint16_t global_And_Sprite_Shift;
13     uint16_t Sprite_Shift;
14
15
16
17

```

```

18  __m128i color_m64;
19  __m128i alpha_m64 ;
20  __m128i buffer_m64 ;
21  __m128i result_m64 ;
22
23  uint16_t* result = (uint16_t*)&result_m64;
24
25  for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
26
27      Sind = instances[currSprite].ind;
28      Sx   = instances[currSprite].x;
29      Sy   = instances[currSprite].y;
30
31      shift_tile_X = Sx/uint16_t(16);
32      shift_coor_X = Sx%uint16_t(16);
33
34      for (uint16_t i = 0; i < SPRITE_TILES_Y>>uint16_t(1); ++i) {
35
36          //SPRITE_TILES_X==4
37          //global_And_Sprite_Shift = (2*i+Sy)*FRAMEBUFFER_TILES_X + (shift_tile_X);
38          global_And_Sprite_Shift = ((i<<uint16_t(1))+Sy)*FRAMEBUFFER_TILES_X + shift_tile_X;
39          Sprite_Shift = (i<<uint16_t(1))*SPRITE_TILES_X;
40
41          color_m64 = _mm_set_epi16(
42              spriteMemory[Sind].color[Sprite_Shift+uint16_t(7)] >> shift_coor_X,
43              spriteMemory[Sind].color[Sprite_Shift+uint16_t(6)] >> shift_coor_X,
44              spriteMemory[Sind].color[Sprite_Shift+uint16_t(5)] >> shift_coor_X,
45              spriteMemory[Sind].color[Sprite_Shift+uint16_t(4)] >> shift_coor_X,
46              spriteMemory[Sind].color[Sprite_Shift+uint16_t(3)] >> shift_coor_X,
47              spriteMemory[Sind].color[Sprite_Shift+uint16_t(2)] >> shift_coor_X,
48              spriteMemory[Sind].color[Sprite_Shift+uint16_t(1)] >> shift_coor_X,
49              spriteMemory[Sind].color[Sprite_Shift
50                  ] >> shift_coor_X);
51
52          alpha_m64 = _mm_set_epi16(
53              spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(7)] >> shift_coor_X,
54              spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(6)] >> shift_coor_X,
55              spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(5)] >> shift_coor_X,
56              spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(4)] >> shift_coor_X,
57              spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(3)] >> shift_coor_X,
58              spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(2)] >> shift_coor_X,
59              spriteMemory[Sind].alpha[Sprite_Shift
60                  ] >> shift_coor_X);
61
62          buffer_m64 = _mm_set_epi16(
63              framebuffer[global_And_Sprite_Shift+uint16_t(3)+FRAMEBUFFER_TILES_X],
64              framebuffer[global_And_Sprite_Shift+uint16_t(2)+FRAMEBUFFER_TILES_X],
65              framebuffer[global_And_Sprite_Shift+uint16_t(1)+FRAMEBUFFER_TILES_X],
66              framebuffer[global_And_Sprite_Shift+
67                  FRAMEBUFFER_TILES_X],
68              framebuffer[global_And_Sprite_Shift+uint16_t(3)],
69              framebuffer[global_And_Sprite_Shift+uint16_t(2)],
70              framebuffer[global_And_Sprite_Shift+uint16_t(1)],
71              framebuffer[global_And_Sprite_Shift
72                  ]);
73
74          for (uint16_t k = 0; k < uint16_t(1); ++k)
75              result_m64 = _mm_or_si128( _mm_andnot_si128(alpha_m64, buffer_m64),
76                  _mm_and_si128(alpha_m64, color_m64));
77
78          for (uint16_t j = 0; j < SPRITE_TILES_X; ++j){
79              framebuffer[global_And_Sprite_Shift+j] = result[j];
80              framebuffer[global_And_Sprite_Shift+j+FRAMEBUFFER_TILES_X] =
81                  result[j+SPRITE_TILES_X];
82          }
83
84          if (shift_coor_X != uint16_t(0)){
85
86              color_m64 = _mm_set_epi16(
87                  spriteMemory[Sind].color[Sprite_Shift+uint16_t(7)]<<(uint16_t(16)-shift_coor_X),
88                  spriteMemory[Sind].color[Sprite_Shift+uint16_t(6)]<<(uint16_t(16)-shift_coor_X),
89                  spriteMemory[Sind].color[Sprite_Shift+uint16_t(5)]<<(uint16_t(16)-shift_coor_X),
90                  spriteMemory[Sind].color[Sprite_Shift+uint16_t(4)]<<(uint16_t(16)-shift_coor_X),
91                  spriteMemory[Sind].color[Sprite_Shift+uint16_t(3)]<<(uint16_t(16)-shift_coor_X),
92                  spriteMemory[Sind].color[Sprite_Shift+uint16_t(2)]<<(uint16_t(16)-shift_coor_X),
93                  spriteMemory[Sind].color[Sprite_Shift+uint16_t(1)]<<(uint16_t(16)-shift_coor_X),
94                  spriteMemory[Sind].color[Sprite_Shift
95                      ]<<(uint16_t(16)-shift_coor_X))
96          }
97      }
98  }
99
100 ;

```

```

93         alpha_m64 = _mm_set_epi16(
94             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(7)]<<(uint16_t(16)-shift_coor_X),
95             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(6)]<<(uint16_t(16)-shift_coor_X),
96             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(5)]<<(uint16_t(16)-shift_coor_X),
97             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(4)]<<(uint16_t(16)-shift_coor_X),
98             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(3)]<<(uint16_t(16)-shift_coor_X),
99             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(2)]<<(uint16_t(16)-shift_coor_X),
100            spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(1)]<<(uint16_t(16)-shift_coor_X),
101            spriteMemory[Sind].alpha[Sprite_Shift
102            ]<<(uint16_t(16)-shift_coor_X));
103
104        buffer_m64 = _mm_set_epi16(
105            framebuffer[global_And_Sprite_Shift+uint16_t(4)+FRAMEBUFFER_TILES_X],
106            framebuffer[global_And_Sprite_Shift+uint16_t(3)+FRAMEBUFFER_TILES_X],
107            framebuffer[global_And_Sprite_Shift+uint16_t(2)+FRAMEBUFFER_TILES_X],
108            framebuffer[global_And_Sprite_Shift+uint16_t(1)+FRAMEBUFFER_TILES_X],
109            framebuffer[global_And_Sprite_Shift+uint16_t(4)],
110            framebuffer[global_And_Sprite_Shift+uint16_t(3)],
111            framebuffer[global_And_Sprite_Shift+uint16_t(2)],
112            framebuffer[global_And_Sprite_Shift+uint16_t(1)]);
113
114        for (uint16_t k = 0; k < uint16_t(1); ++k)
115            result_m64 = _mm_or_si128( _mm_andnot_si128(alpha_m64, buffer_m64),
116                                     _mm_and_si128(alpha_m64, color_m64));
117
118        for (uint16_t j = 0; j < SPRITE_TILES_X; ++j){
119            framebuffer[global_And_Sprite_Shift+j+uint16_t(1)] = result[j];
120            framebuffer[global_And_Sprite_Shift+j+FRAMEBUFFER_TILES_X+uint16_t(1)]
121                = result[j+SPRITE_TILES_X];
122        }
123    }
124 }
125 }
126 }

```

5.4.4 Программа 3_binary_rendering_2_NORMAL_AVX

```

1
2 void GP_ONE::drawSpriteInstances(const SpriteInstance *instances, uint16_t instanceCount) {
3
4     // framebuffer = (Color & Alpha) | (framebuffer & ~Alpha)
5
6     uint16_t Sind ;
7     uint16_t Sx   ;
8     uint16_t Sy   ;
9
10    uint16_t shift_tile_X;
11    uint16_t shift_coor_X ;
12
13    uint16_t global_And_Sprite_Shift;
14    uint16_t Sprite_Shift;
15
16    __m256i color_m64;
17    __m256i alpha_m64 ;
18    __m256i buffer_m64 ;
19    __m256i result_m64 ;
20
21    uint16_t* result = (uint16_t*)&result_m64;
22
23
24    for (uint16_t currSprite = 0; currSprite < instanceCount; currSprite++){
25
26        Sind = instances[currSprite].ind;
27        Sx   = instances[currSprite].x;
28        Sy   = instances[currSprite].y;
29
30        shift_tile_X = Sx/uint16_t(16);
31        shift_coor_X = Sx%uint16_t(16);
32
33

```

```

34     for (uint16_t i = 0; i < SPRITE_TILES_Y>>uint16_t(2); ++i) {
35
36         //SPRITE_TILES_X==4
37         global_And_Sprite_Shift = ((i<<uint16_t(2))+Sy)*FRAMEBUFFER_TILES_X + (shift_tile_X)
;
38         Sprite_Shift = (i<<uint16_t(2))*SPRITE_TILES_X;
39
40         color_m64 = _mm256_set_epi16(
41             spriteMemory[Sind].color[Sprite_Shift+uint16_t(15)] >> shift_coor_X,
42             spriteMemory[Sind].color[Sprite_Shift+uint16_t(14)] >> shift_coor_X,
43             spriteMemory[Sind].color[Sprite_Shift+uint16_t(13)] >> shift_coor_X,
44             spriteMemory[Sind].color[Sprite_Shift+uint16_t(12)] >> shift_coor_X,
45             spriteMemory[Sind].color[Sprite_Shift+uint16_t(11)] >> shift_coor_X,
46             spriteMemory[Sind].color[Sprite_Shift+uint16_t(10)] >> shift_coor_X,
47             spriteMemory[Sind].color[Sprite_Shift+uint16_t(9)] >> shift_coor_X,
48             spriteMemory[Sind].color[Sprite_Shift+uint16_t(8)] >> shift_coor_X,
49             spriteMemory[Sind].color[Sprite_Shift+uint16_t(7)] >> shift_coor_X,
50             spriteMemory[Sind].color[Sprite_Shift+uint16_t(6)] >> shift_coor_X,
51             spriteMemory[Sind].color[Sprite_Shift+uint16_t(5)] >> shift_coor_X,
52             spriteMemory[Sind].color[Sprite_Shift+uint16_t(4)] >> shift_coor_X,
53             spriteMemory[Sind].color[Sprite_Shift+uint16_t(3)] >> shift_coor_X,
54             spriteMemory[Sind].color[Sprite_Shift+uint16_t(2)] >> shift_coor_X,
55             spriteMemory[Sind].color[Sprite_Shift+uint16_t(1)] >> shift_coor_X,
56             spriteMemory[Sind].color[Sprite_Shift] >> shift_coor_X);
57
58         alpha_m64 = _mm256_set_epi16(
59             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(15)] >> shift_coor_X,
60             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(14)] >> shift_coor_X,
61             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(13)] >> shift_coor_X,
62             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(12)] >> shift_coor_X,
63             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(11)] >> shift_coor_X,
64             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(10)] >> shift_coor_X,
65             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(9)] >> shift_coor_X,
66             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(8)] >> shift_coor_X,
67             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(7)] >> shift_coor_X,
68             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(6)] >> shift_coor_X,
69             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(5)] >> shift_coor_X,
70             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(4)] >> shift_coor_X,
71             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(3)] >> shift_coor_X,
72             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(2)] >> shift_coor_X,
73             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(1)] >> shift_coor_X,
74             spriteMemory[Sind].alpha[Sprite_Shift] >> shift_coor_X);
75
76         buffer_m64 = _mm256_set_epi16(
77             framebuffer[global_And_Sprite_Shift+uint16_t(3)+3*FRAMEBUFFER_TILES_X],
78             framebuffer[global_And_Sprite_Shift+uint16_t(2)+3*FRAMEBUFFER_TILES_X],
79             framebuffer[global_And_Sprite_Shift+uint16_t(1)+3*FRAMEBUFFER_TILES_X],
80             framebuffer[global_And_Sprite_Shift+
81                 3*FRAMEBUFFER_TILES_X],
82             framebuffer[global_And_Sprite_Shift+uint16_t(3)+2*FRAMEBUFFER_TILES_X],
83             framebuffer[global_And_Sprite_Shift+uint16_t(2)+2*FRAMEBUFFER_TILES_X],
84             framebuffer[global_And_Sprite_Shift+uint16_t(1)+2*FRAMEBUFFER_TILES_X],
85             framebuffer[global_And_Sprite_Shift+
86                 2*FRAMEBUFFER_TILES_X],
87             framebuffer[global_And_Sprite_Shift+uint16_t(3)+
88                 FRAMEBUFFER_TILES_X],
89             framebuffer[global_And_Sprite_Shift+uint16_t(2)+
90                 FRAMEBUFFER_TILES_X],
91             framebuffer[global_And_Sprite_Shift+uint16_t(1)+
92                 FRAMEBUFFER_TILES_X],
93             framebuffer[global_And_Sprite_Shift+
94                 FRAMEBUFFER_TILES_X],
95             framebuffer[global_And_Sprite_Shift+uint16_t(3)],
96             framebuffer[global_And_Sprite_Shift+uint16_t(2)],
97             framebuffer[global_And_Sprite_Shift+uint16_t(1)],
98             framebuffer[global_And_Sprite_Shift]
99             ]);
100
101         for (uint16_t k = 0; k < 100; ++k)
102             result_m64 = _mm256_or_si256(
103                 _mm256_andnot_si256(alpha_m64, buffer_m64),
104                 _mm256_and_si256(alpha_m64, color_m64));
105
106         for (uint16_t j = 0; j < SPRITE_TILES_X; ++j){
107             framebuffer[global_And_Sprite_Shift+j] = result[j];
108             framebuffer[global_And_Sprite_Shift+j+FRAMEBUFFER_TILES_X] =
109                 result[j+SPRITE_TILES_X];
110             framebuffer[global_And_Sprite_Shift+j+2*FRAMEBUFFER_TILES_X] =
111                 result[j+2*SPRITE_TILES_X];
112             framebuffer[global_And_Sprite_Shift+j+3*FRAMEBUFFER_TILES_X] =
113                 result[j+3*SPRITE_TILES_X];
114         }
115     }

```

```

109     if (shift_coor_X != uint16_t(0)){
110
111         color_m64 = _mm256_set_epi16(
112             spriteMemory[Sind].color[Sprite_Shift+uint16_t(15)] << (uint16_t(16)-shift_coor_X),
113             spriteMemory[Sind].color[Sprite_Shift+uint16_t(14)] << (uint16_t(16)-shift_coor_X),
114             spriteMemory[Sind].color[Sprite_Shift+uint16_t(13)] << (uint16_t(16)-shift_coor_X),
115             spriteMemory[Sind].color[Sprite_Shift+uint16_t(12)] << (uint16_t(16)-shift_coor_X),
116             spriteMemory[Sind].color[Sprite_Shift+uint16_t(11)] << (uint16_t(16)-shift_coor_X),
117             spriteMemory[Sind].color[Sprite_Shift+uint16_t(10)] << (uint16_t(16)-shift_coor_X),
118             spriteMemory[Sind].color[Sprite_Shift+uint16_t(9)] << (uint16_t(16)-shift_coor_X),
119             spriteMemory[Sind].color[Sprite_Shift+uint16_t(8)] << (uint16_t(16)-shift_coor_X),
120             spriteMemory[Sind].color[Sprite_Shift+uint16_t(7)] << (uint16_t(16)-shift_coor_X),
121             spriteMemory[Sind].color[Sprite_Shift+uint16_t(6)] << (uint16_t(16)-shift_coor_X),
122             spriteMemory[Sind].color[Sprite_Shift+uint16_t(5)] << (uint16_t(16)-shift_coor_X),
123             spriteMemory[Sind].color[Sprite_Shift+uint16_t(4)] << (uint16_t(16)-shift_coor_X),
124             spriteMemory[Sind].color[Sprite_Shift+uint16_t(3)] << (uint16_t(16)-shift_coor_X),
125             spriteMemory[Sind].color[Sprite_Shift+uint16_t(2)] << (uint16_t(16)-shift_coor_X),
126             spriteMemory[Sind].color[Sprite_Shift+uint16_t(1)] << (uint16_t(16)-shift_coor_X),
127             spriteMemory[Sind].color[Sprite_Shift] << (uint16_t(16)-shift_coor_X));
128
129         alpha_m64 = _mm256_set_epi16(
130             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(15)] << (uint16_t(16)-shift_coor_X),
131             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(14)] << (uint16_t(16)-shift_coor_X),
132             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(13)] << (uint16_t(16)-shift_coor_X),
133             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(12)] << (uint16_t(16)-shift_coor_X),
134             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(11)] << (uint16_t(16)-shift_coor_X),
135             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(10)] << (uint16_t(16)-shift_coor_X),
136             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(9)] << (uint16_t(16)-shift_coor_X),
137             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(8)] << (uint16_t(16)-shift_coor_X),
138             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(7)] << (uint16_t(16)-shift_coor_X),
139             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(6)] << (uint16_t(16)-shift_coor_X),
140             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(5)] << (uint16_t(16)-shift_coor_X),
141             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(4)] << (uint16_t(16)-shift_coor_X),
142             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(3)] << (uint16_t(16)-shift_coor_X),
143             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(2)] << (uint16_t(16)-shift_coor_X),
144             spriteMemory[Sind].alpha[Sprite_Shift+uint16_t(1)] << (uint16_t(16)-shift_coor_X),
145             spriteMemory[Sind].alpha[Sprite_Shift] << (uint16_t(16)-shift_coor_X));
146
147         buffer_m64 = _mm256_set_epi16(
148             frameBuffer[global_And_Sprite_Shift+uint16_t(4)+3*FRAMEBUFFER_TILES_X],
149             frameBuffer[global_And_Sprite_Shift+uint16_t(3)+3*FRAMEBUFFER_TILES_X],
150             frameBuffer[global_And_Sprite_Shift+uint16_t(2)+3*FRAMEBUFFER_TILES_X],
151             frameBuffer[global_And_Sprite_Shift+uint16_t(1)+3*FRAMEBUFFER_TILES_X],
152             frameBuffer[global_And_Sprite_Shift+uint16_t(4)+2*FRAMEBUFFER_TILES_X],
153             frameBuffer[global_And_Sprite_Shift+uint16_t(3)+2*FRAMEBUFFER_TILES_X],
154             frameBuffer[global_And_Sprite_Shift+uint16_t(2)+2*FRAMEBUFFER_TILES_X],
155             frameBuffer[global_And_Sprite_Shift+uint16_t(1)+2*FRAMEBUFFER_TILES_X],
156             frameBuffer[global_And_Sprite_Shift+uint16_t(4)+ FRAMEBUFFER_TILES_X],
157             frameBuffer[global_And_Sprite_Shift+uint16_t(3)+ FRAMEBUFFER_TILES_X],
158             frameBuffer[global_And_Sprite_Shift+uint16_t(2)+ FRAMEBUFFER_TILES_X],
159             frameBuffer[global_And_Sprite_Shift+uint16_t(1)+ FRAMEBUFFER_TILES_X],
160             frameBuffer[global_And_Sprite_Shift+uint16_t(4)],
161             frameBuffer[global_And_Sprite_Shift+uint16_t(3)],
162             frameBuffer[global_And_Sprite_Shift+uint16_t(2)],
163             frameBuffer[global_And_Sprite_Shift+uint16_t(1)]);
164
165         for (uint16_t k = 0; k < 1; ++k)
166             result_m64 = _mm256_or_si256(_mm256_andnot_si256(alpha_m64, buffer_m64),
167                                     _mm256_and_si256(alpha_m64, color_m64));
168
169         for (uint16_t j = 0; j < SPRITE_TILES_X; ++j){
170             frameBuffer[global_And_Sprite_Shift+j+uint16_t(1)] = result[j];
171             frameBuffer[global_And_Sprite_Shift+j+uint16_t(1)+FRAMEBUFFER_TILES_X] =
172                 result[j+SPRITE_TILES_X];
173             frameBuffer[global_And_Sprite_Shift+j+uint16_t(1)+2*FRAMEBUFFER_TILES_X] =
174                 result[j+2*SPRITE_TILES_X];
175             frameBuffer[global_And_Sprite_Shift+j+uint16_t(1)+3*FRAMEBUFFER_TILES_X] =
176                 result[j+3*SPRITE_TILES_X];
177         }
178     }
179 }
180 }

```


6 Анализ скорости выполнения программ

Данная глава содержит в себе несколько рассуждений по поводу оптимизации и уместности использовать MMX, SSE, AVX инструкции для ускорения работы программы.

6.1 Особенность реализаций

Стоит отметить, что при выполнении программы выполняется одна и та же задача, а значит и при множественном запуске программы скорость выполнения должна быть одинаковой, что не является действительным. Так как мы запускаем программу в операционной системе, то часть компьютерных ресурсов неравномерно распределяются на другие задачи, что замедляет работу нашей программы. Учитывая все это уместно считать эффективным временем работы программы - её наименьшее время работы при множественном запуске.

6.2 16 BIT

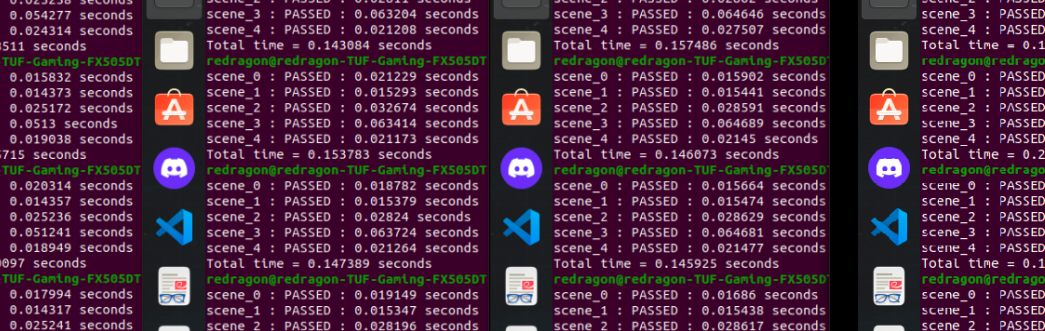
Запустим приложения EASY, NORMAL, HARD, 6 раз с интервалом в 10 секунд, чтобы время выполнения программы было меньше.

Получим, следующее эффективное время работы программ:

	EASY:	NORMAL:	HARD:
1	1.5724	1.39961	1.34166
2	1.5659	1.39685	1.34038
3	2.93157	2.38966	2.2113
4	6.69882	4.78407	4.39724
5	2.19121	1.84275	1.72608
6	Total time: 14.9599	Total time: 11.81294	total time: 11.01666

6.3 0 Four Tiles Only

Запустим все приложения в папке 6 раз с интервалом в 10 секунд, чтобы время выполнения программы было меньше. Напомним, что в данном случае $\text{TEST_ITERATIONS} = 100$, $\text{COMMAND_SIZE} = 1$:



```

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.020247 seconds
scene_1 : PASSED : 0.014415 seconds
scene_2 : PASSED : 0.025258 seconds
scene_3 : PASSED : 0.054277 seconds
scene_4 : PASSED : 0.024314 seconds
Total time = 0.138511 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.015832 seconds
scene_1 : PASSED : 0.014375 seconds
scene_2 : PASSED : 0.025172 seconds
scene_3 : PASSED : 0.0513 seconds
scene_4 : PASSED : 0.019038 seconds
Total time = 0.125715 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.020314 seconds
scene_1 : PASSED : 0.014357 seconds
scene_2 : PASSED : 0.025236 seconds
scene_3 : PASSED : 0.051241 seconds
scene_4 : PASSED : 0.018949 seconds
Total time = 0.130097 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.017994 seconds
scene_1 : PASSED : 0.014317 seconds
scene_2 : PASSED : 0.025241 seconds
scene_3 : PASSED : 0.051073 seconds
scene_4 : PASSED : 0.019051 seconds
Total time = 0.127676 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.014797 seconds
scene_1 : PASSED : 0.014415 seconds
scene_2 : PASSED : 0.02534 seconds
scene_3 : PASSED : 0.051264 seconds
scene_4 : PASSED : 0.019129 seconds
Total time = 0.124945 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.018219 seconds
scene_1 : PASSED : 0.014465 seconds
scene_2 : PASSED : 0.025343 seconds
scene_3 : PASSED : 0.051658 seconds
scene_4 : PASSED : 0.019159 seconds
Total time = 0.128844 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.015308 seconds
scene_1 : PASSED : 0.015254 seconds
scene_2 : PASSED : 0.02811 seconds
scene_3 : PASSED : 0.063204 seconds
scene_4 : PASSED : 0.021208 seconds
Total time = 0.143084 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.021229 seconds
scene_1 : PASSED : 0.015293 seconds
scene_2 : PASSED : 0.032674 seconds
scene_3 : PASSED : 0.063414 seconds
scene_4 : PASSED : 0.021173 seconds
Total time = 0.153783 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.018782 seconds
scene_1 : PASSED : 0.015379 seconds
scene_2 : PASSED : 0.02824 seconds
scene_3 : PASSED : 0.063724 seconds
scene_4 : PASSED : 0.021264 seconds
Total time = 0.147389 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.019149 seconds
scene_1 : PASSED : 0.015347 seconds
scene_2 : PASSED : 0.028196 seconds
scene_3 : PASSED : 0.063101 seconds
scene_4 : PASSED : 0.021231 seconds
Total time = 0.147024 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.020375 seconds
scene_1 : PASSED : 0.015313 seconds
scene_2 : PASSED : 0.028274 seconds
scene_3 : PASSED : 0.063781 seconds
scene_4 : PASSED : 0.021224 seconds
Total time = 0.148967 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.025527 seconds
scene_1 : PASSED : 0.015325 seconds
scene_2 : PASSED : 0.028171 seconds
scene_3 : PASSED : 0.063255 seconds
scene_4 : PASSED : 0.021225 seconds
Total time = 0.153528 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.021231 seconds
scene_1 : PASSED : 0.015482 seconds
scene_2 : PASSED : 0.02862 seconds
scene_3 : PASSED : 0.064446 seconds
scene_4 : PASSED : 0.027507 seconds
Total time = 0.157486 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.015902 seconds
scene_1 : PASSED : 0.015441 seconds
scene_2 : PASSED : 0.028591 seconds
scene_3 : PASSED : 0.064689 seconds
scene_4 : PASSED : 0.02145 seconds
Total time = 0.146073 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.015664 seconds
scene_1 : PASSED : 0.015474 seconds
scene_2 : PASSED : 0.028629 seconds
scene_3 : PASSED : 0.064681 seconds
scene_4 : PASSED : 0.021477 seconds
Total time = 0.145925 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.01686 seconds
scene_1 : PASSED : 0.015438 seconds
scene_2 : PASSED : 0.028617 seconds
scene_3 : PASSED : 0.064463 seconds
scene_4 : PASSED : 0.021485 seconds
Total time = 0.147063 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.015703 seconds
scene_1 : PASSED : 0.015545 seconds
scene_2 : PASSED : 0.028699 seconds
scene_3 : PASSED : 0.064716 seconds
scene_4 : PASSED : 0.021627 seconds
Total time = 0.146229 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.018698 seconds
scene_1 : PASSED : 0.01548 seconds
scene_2 : PASSED : 0.028669 seconds
scene_3 : PASSED : 0.064924 seconds
scene_4 : PASSED : 0.021573 seconds
Total time = 0.149344 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.020288 seconds
scene_1 : PASSED : 0.014415 seconds
scene_2 : PASSED : 0.037226 seconds
scene_3 : PASSED : 0.092399 seconds
scene_4 : PASSED : 0.027188 seconds
Total time = 0.195203 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.023859 seconds
scene_1 : PASSED : 0.0229 seconds
scene_2 : PASSED : 0.037709 seconds
scene_3 : PASSED : 0.093662 seconds
scene_4 : PASSED : 0.027303 seconds
Total time = 0.265633 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.023287 seconds
scene_1 : PASSED : 0.018157 seconds
scene_2 : PASSED : 0.037408 seconds
scene_3 : PASSED : 0.092995 seconds
scene_4 : PASSED : 0.027222 seconds
Total time = 0.198969 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.029132 seconds
scene_1 : PASSED : 0.018234 seconds
scene_2 : PASSED : 0.037309 seconds
scene_3 : PASSED : 0.097617 seconds
scene_4 : PASSED : 0.027088 seconds
Total time = 0.195375 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.018318 seconds
scene_1 : PASSED : 0.018802 seconds
scene_2 : PASSED : 0.03736 seconds
scene_3 : PASSED : 0.10055 seconds
scene_4 : PASSED : 0.027705 seconds
Total time = 0.262235 seconds

redragonredragon-TUF-Gaming-FX5850T
scene_0 : PASSED : 0.020075 seconds
scene_1 : PASSED : 0.018135 seconds
scene_2 : PASSED : 0.037729 seconds
scene_3 : PASSED : 0.09307 seconds
scene_4 : PASSED : 0.027419 seconds
Total time = 0.195989 seconds

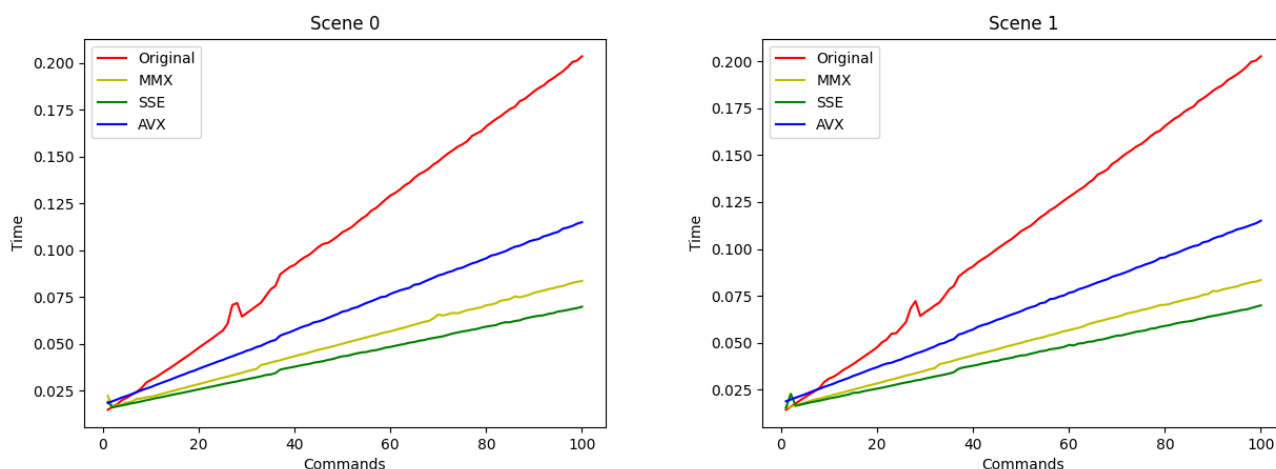
```

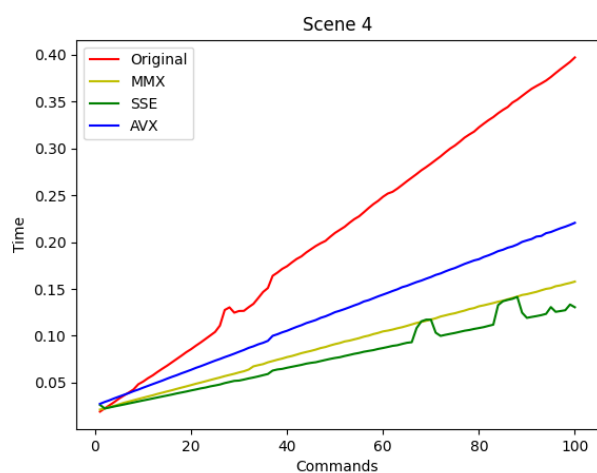
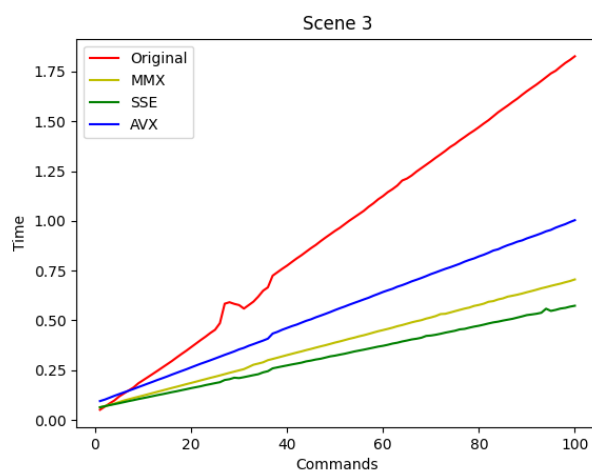
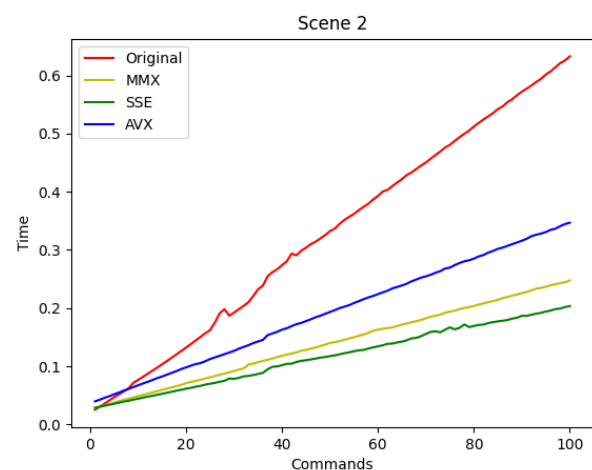
Получим, следующее эффективное время работы программ:

	ORIGINAL :	MMX :	SSE :	AVX :
1	0.014797	0.015308	0.015664	0.018318
2	0.014317	0.015254	0.015438	0.018157
3	0.025172	0.02811	0.028591	0.037226
4	0.051073	0.063101	0.064646	0.092399
5	0.018949	0.021173	0.021477	0.027088
6				
7	Total time: 0.124308	Total time: 0.142946	Total time: 0.145816	Total time: 0.193188

Заметим, что при увеличении размера регистра увеличивается время выполнения программы. Это напрямую связано с тем, что операция присваивания выполняется значительно дольше чем арифметическая операция над регистром.

Для наглядности увеличим количество арифметических операций в 100 раз. Для этого присвоим переменной `COMMAND_SIZE` значение 100 в программе "MOD TO ANALYSE" и рассмотрим следующие графики:





Наглядно видно, что при большей работе с регистрами путем арифметических операций и других операций по стоимости ниже операции присваивания, применение регистров обретает свой смысл. Неровности линий графиков обусловлены невозможностью операционной системы уделить все ресурсы на выполнение нашей программы, что очевидно приводит к снижению производительности.

Так же мы можем наблюдать, что не всегда использование регистров большего размера ускоряет работу приложения, что является следствием того что процессор тратит много времени на заполнение регистра.

Отсюда получаем вывод, что для нашей программы нет необходимости в применении инструкций процессора, так как две операции конъюнкции, одна операция дизъюнкции и отрицания занимают значительно меньше времени чем операция присваивания.

6.4 1 All Possible Tiles

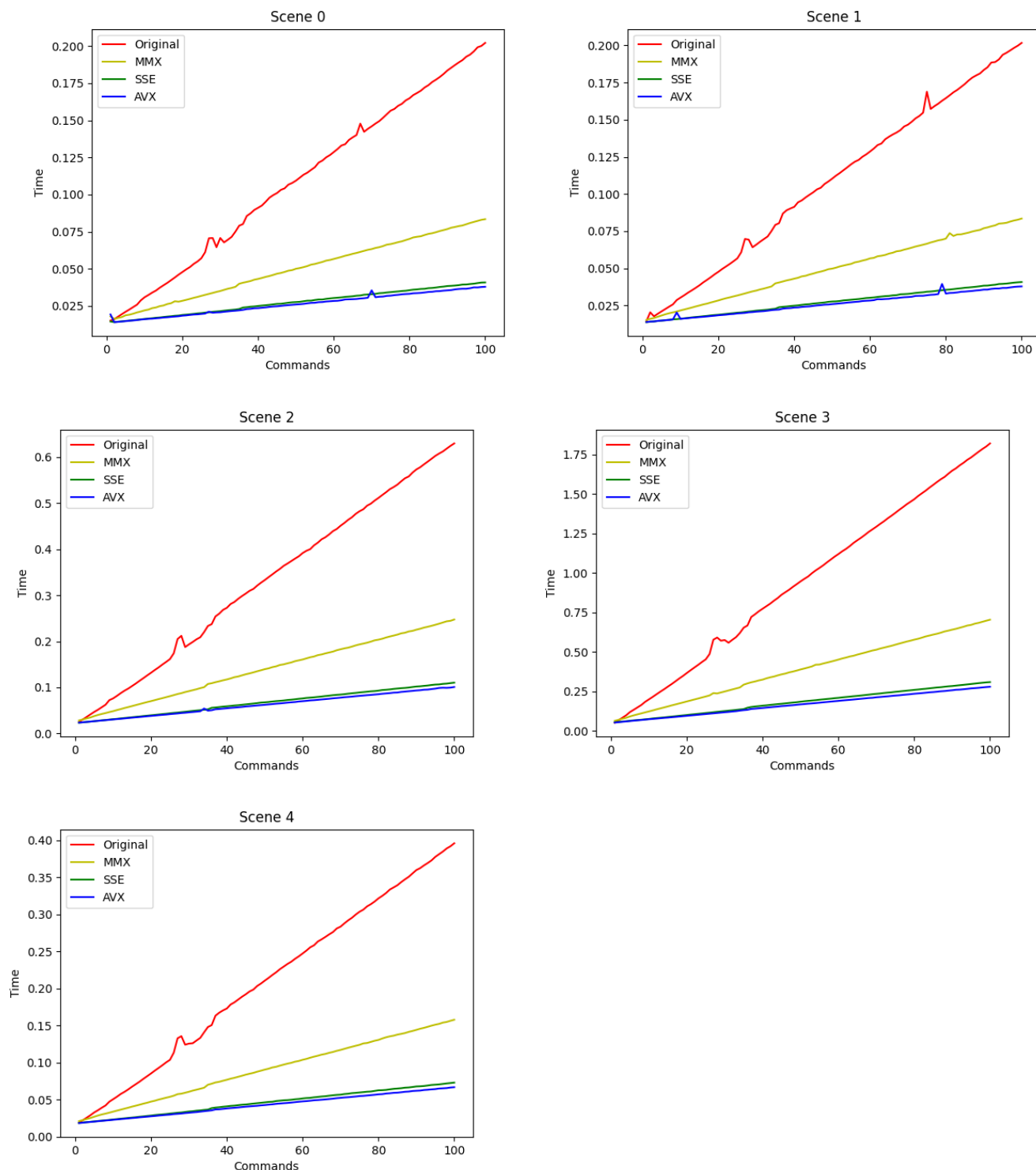
Запустим все приложения в папке 7 раз с интервалом в 10 секунд, чтобы время выполнения программы было меньше. Напомним, что в данном случае $TEST_ITERATIONS = 100$, $COMMAND_SIZE = 1$;

The image shows 20 terminal windows arranged in a 5x4 grid. Each window displays the output of a benchmark program. The first column shows the 'ORIGINAL' baseline. The next three columns show results for MMX, SSE, and AVX instructions. Each window lists five scenes with their individual execution times and a 'Total time' at the bottom. The windows are titled 'redragon@redragon-TUF-Gaming-FX505DT [100%] Built target BinaryRenderer'.

Получим, следующее эффективное время работы программ:

	ORIGINAL:	MMX:	SSE:	AVX:
1				
2	0.014648	0.015342	0.014446	0.014299
3	0.014329	0.01524	0.013802	0.014023
4	0.025238	0.028037	0.023453	0.024066
5	0.051162	0.063195	0.053486	0.052783
6	0.019051	0.02113	0.018549	0.018803
7	Total time: 0.124308	Total time: 0.110211	Total time: 0.123736	Total time: 0.123974

Заметно, что при заполнении всего регистра скорость выполнения программы стала значительно выше, так как используются не только первые 64 бит регистра. Здесь, в отличие от примера с "0 Four Tiles Only" эффективность программы с регистрами AVX является самой оптимальной, если количество арифметических операций достаточно много, что можно заметить по графикам ниже:



Однако преимущество использовать AVX по отношению к SSE появляется только если количество арифметических операций будет очень много. Это наглядно видно по углу наклона прямой.

Можем сделать вывод, что для нашей программы лучше всего использовать MMX или SSE, так как минимальное время этих програм немного ниже оригинала.

7 Дополнительная часть (анализ производительности)

7.1 16 BIT

Мы имеем при себе процессор с частотой 1КГц

Для анализа производительности будем использовать программу "binary_rendering_3_HARD"

Для условности будем учитывать что операции сдвига, сложения, приравнивания, сравнения, отрицания, побитового "и" и побитового "или" занимают один такт при выполнении.

Следовательно в нашем алгоритме содержится следующее количество операций:

1) Если координата спрайта по x кратна 16 будет выполнено следующее количество тактов

```
1 || instanceCount( 1+2+2+1+1 + 64(4+2+1 + 4(3+3+7+1+1) ) ) = 4295*instanceCount;
```

2) Если координата спрайта по x не кратна 16 будет выполнено следующее количество тактов

```
1 || instanceCount( 1+2+2+1+2+1 + 64(4+2+1 + 4(3+3+7 + (1+3+3+9) + 1)) ) = 8137*instanceCount;
```

Следовательно при использовании такого центрального процессора один спрайт будет растеризироваться 4.295 секунды в случае когда координата спрайта по x кратна 16, и 8.137 секунд иначе.

7.2 64 BIT

Если мы будем использовать uint_64 и процессор с частотой 1ГГц, то скорость выполнения алгоритма изменится. Для примера используем для анализа программу "1_binary_rendering_2_NORMAL_MMX"

В нашем алгоритме содержится следующее количество операций:

1) Если координата спрайта по x кратна 64 будет выполнено следующее количество тактов

```
1 || instanceCount( 1 + 1 + 4 + 64( 4+2 + 2+2+5+ 1 ) ) = 1030*instanceCount;
```

2) Если координата спрайта по x не кратна 64 будет выполнено следующее количество тактов

```
1 || instanceCount( 1 + 1 + 4 + 64( 4+2 + 2+2+5+ 1 + 1 + 2 + 2 + 1 + 5 + 2 ) ) = 1862*instanceCount
```

Следовательно один спрайт будет растеризироваться $1.03 * 10^{-6}$ и $1.862 * 10^{-6}$ секунд соответственно