PC-2023/24 Alpha-Compositor

Christian Mancini
E-mail address
christian.mancini1@edu.unifi.it

Abstract

In this report, we present the implementation of a parallel version of alpha compositing using C++ and OpenMP [2], achieving a speedup of nearly five times. We tested various combinations, including HD to Full HD, Full HD to 2K, and 2K to 4K, while varying the number of threads and images processed.

1. Introduction

Alpha compositing or alpha blending is the process of combining one image with a background to create the appearance of partial or full transparency. In a 2D image a color combination is stored for each picture element (pixel), often a combination of red, green and blue (RGB). When alpha compositing is in use, each pixel has an additional numeric value stored in its alpha channel (RGBA), with a value ranging from 0 to 1. A value of 0 means that the pixel is fully transparent and the color in the pixel beneath will show through. A value of 1 means that the pixel is fully opaque. With this definition in mind we can produce the effect of drawing the source pixels on top of the destination pixels (foreground over a background) using the following formula:

$$[RGBA]_d = [RGBA]_s + [RGBA]_d(1 - A_s) \quad (1)$$

For our scope the foreground must be completely opaque. Changing the weights leads to different results.

1.1. Compositing Algorithm

The compositing algorithm is straightforward, we just need to apply the 1 on every channel of the background image, as shown in listing 1.

Listing 1. Snippet of compose funcion

```
1
2  bool OpenMP_compose(const Image &foreground
       , Image &background) {
3      if (foreground.height > background.
           height | foreground.width >
           background.width) {
4          return false;
5      }
6      #pragma omp parallel for collapse(2)
           shared(foreground)
7      for (int y = 0; y < foreground.height;
           ++y) {
8          for (int x = 0; x < foreground.
               width; ++x) {
9
10             int backgroundIndex = (y *
                   background.width + x) *
                   STBI_rgb_alpha;
11             int foregroundIndex = (y *
                   foreground.width + x) *
                   STBI_rgb_alpha;
12
13             float alpha = static_cast<float
                   >(foreground.rgb_image[
                   foregroundIndex + 3]) /
                   255.0f;
14             float beta = 1.0f - alpha;
15             #pragma omp simd
16             for (int color = 0; color < 3;
                   ++color) {
17                 background.rgb_image[
                       backgroundIndex + color]
                        =
18                 static_cast<float>(
                       background.rgb_image[
                       backgroundIndex + color
                       ]) * beta
```

```
                    + static_cast<float>(
                        foreground.rgb_image[
                        foregroundIndex + color
                        ]) * alpha;

            }
        }
    }
    return true;
}
```

2. Results

Tables 1, 2, 3 shows compositing results and speed up, groupped by the background resolution.

Table 1. FullHD result

Threads	Composing Time	Background	Speedup
1	0.3958	FullHD	1.0
25	0.1198	FullHD	3.3038
50	0.1126	FullHD	3.5151
100	0.1195	FullHD	3.3121
200	0.1409	FullHD	2.8091
400	0.1965	FullHD	2.0142

Table 2. 2K result

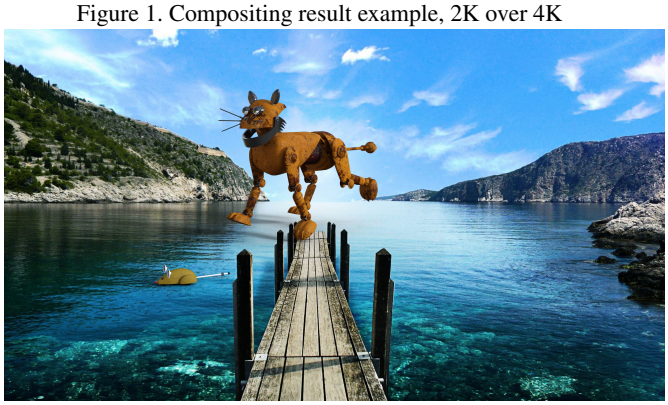
Threads	Composing Time	Background	Speedup
1	1.0272	2K	1.0
25	0.2661	2K	3.8602
50	0.2511	2K	4.0908
100	0.2250	2K	4.5653
200	0.2535	2K	4.0521
400	0.2866	2K	3.5841

Table 3. 4K result

Threads	Composing Time	Background	Speedup
1	2.0763	4K	1.0
25	0.4922	4K	4.2184
50	0.4668	4K	4.4479
100	0.4276	4K	4.8557
200	0.4343	4K	4.7808
400	0.4688	4K	4.4290

In Figure 1 we show the compositing result of a Background image in 4K and a Foreground in 2K.

Figure 1. Compositing result example, 2K over 4K



In Figuers 2, 3 and 4 we show the bar plots of the resoults

Figure 2. SpeedUp on FullHD Backgroung

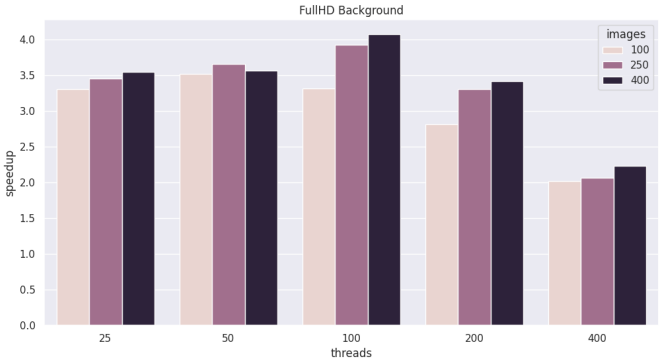


Figure 3. SpeedUp on 2K Backgroung

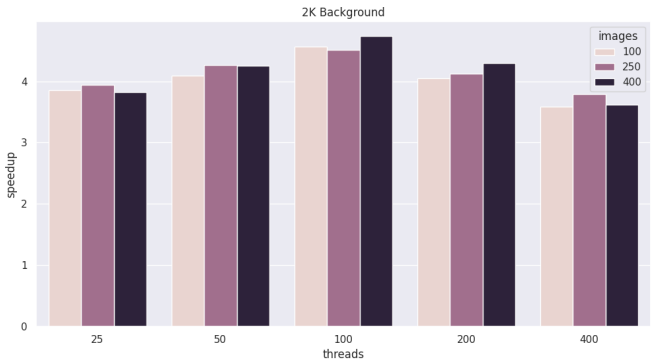
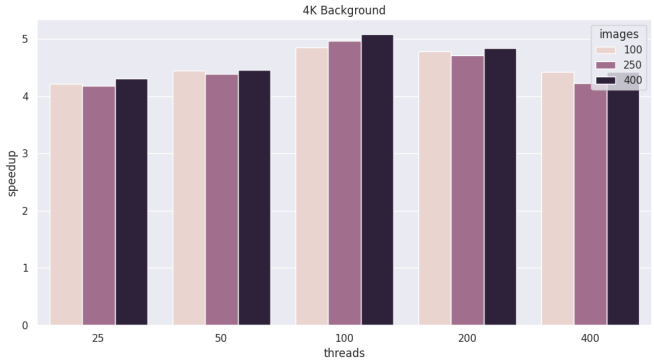


Figure 4. SpeedUp on 4K Backgroung



References

[1] J. Nichols and N. Nethercote. Valgrind.

[2] OpenMP Architecture Review Board. *OpenMP Application Program Interface*. OpenMP Architecture Review Board, 2019.