Alpha Compositing

Parallel Computing 2023-2024

Christian Mancini

Florence, August 2024

Alpha Compositing

What is alpha compositing

Alpha compositing is the process of combining one image with a background to create the appearance of partial or full transparency.

When alpha compositing is in use, each pixel has an additional numeric value stored in its alpha channel, 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.

Foreground HD (left) and background Full HD (right)

Compositing Result

Alpha Compositing Formula

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

- The formula is very simple, we just have to apply it on each pixel.

Implementation

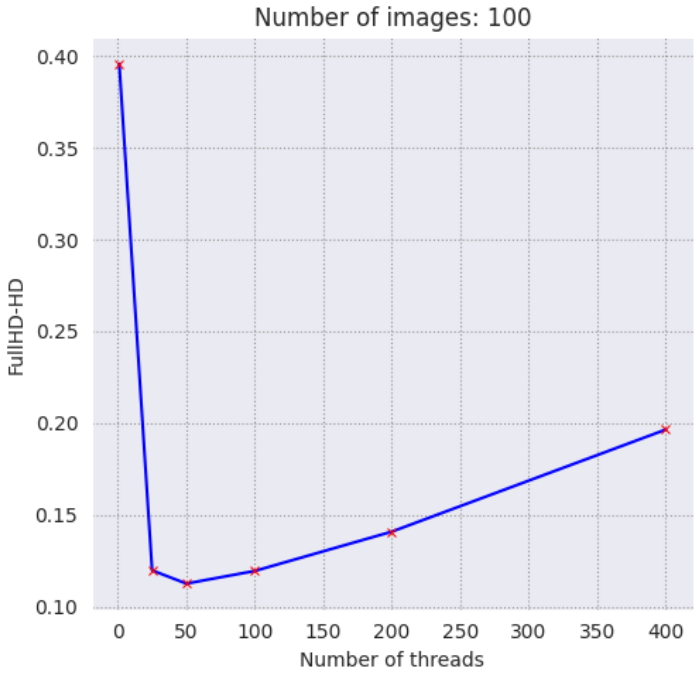
```
bool OpenMP_compose(const Image &foreground, Image &background) {
    if (foreground.height > background.height | foreground.width > background.width) {
        return false;
    }
#pragma omp parallel for collapse(2) shared(foreground)
    for (int y = 0; y < foreground.height; ++y) {
        for (int x = 0; x < foreground.width; ++x) {

            int backgroundIndex = (y * background.width + x) * STBI_rgb_alpha;
            int foregroundIndex = (y * foreground.width + x) * STBI_rgb_alpha;

            float alpha = static_cast<float>(foreground.rgb_image[foregroundIndex + 3]) / 255.0f;
            float beta = 1.0f - alpha;
#pragma omp simd
            for (int color = 0; color < 3; ++color) {
                background.rgb_image[backgroundIndex + color] =
                        static_cast<float>(background.rgb_image[backgroundIndex + color]) * beta
                        + static_cast<float>(foreground.rgb_image[foregroundIndex + color]) * alpha;

            }
        }
    }
    return true;
}
```

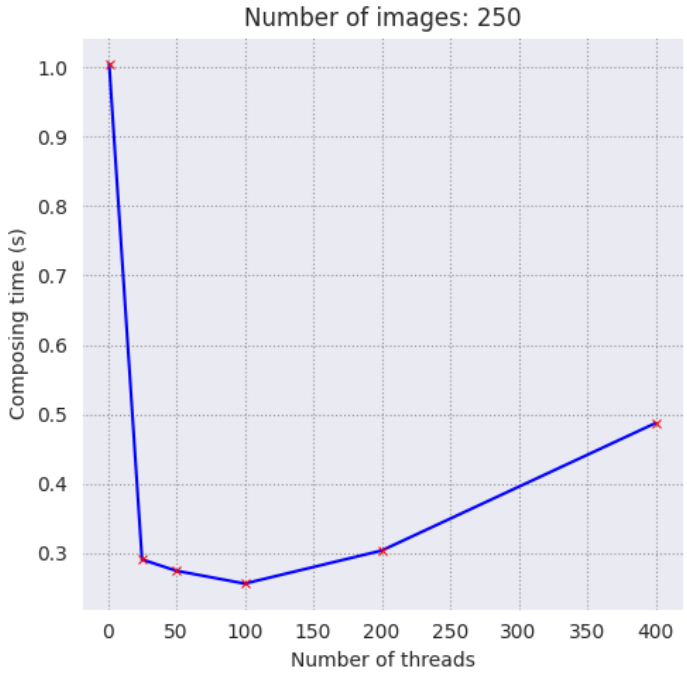
Implementation

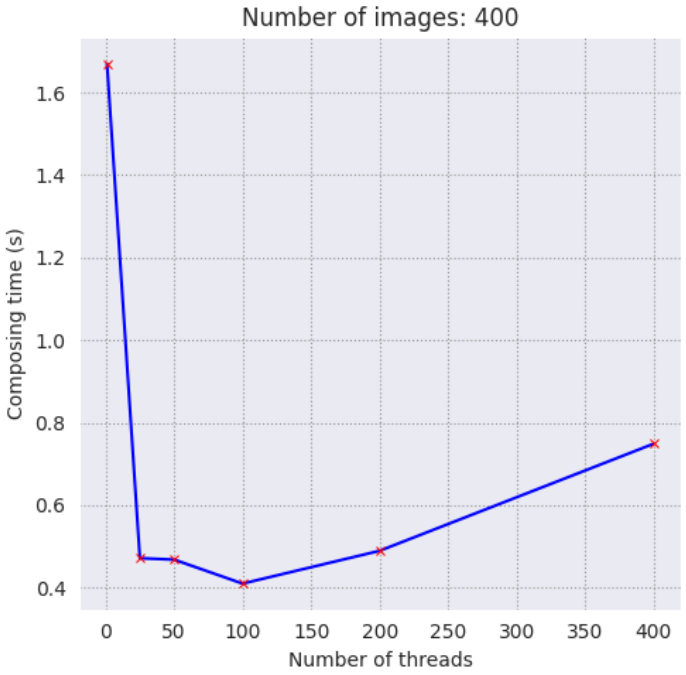
```
bool OpenMP_compose(const Image &foreground, Image &background) {
    if (foreground.height > background.height | foreground.width > background.width) {
        return false;
    }
#pragma omp parallel for collapse(2) shared(foreground)
    for (int y = 0; y < foreground.height; ++y) {
        for (int x = 0; x < foreground.width; ++x) {

            int backgroundIndex = (y * background.width + x) * STBI_rgb_alpha;
            int foregroundIndex = (y * foreground.width + x) * STBI_rgb_alpha;

            float alpha = static_cast<float>(foreground.rgb_image[foregroundIndex + 3]) / 255.0f;
            float beta = 1.0f - alpha;
#pragma omp simd
            for (int color = 0; color < 3; ++color) {
                background.rgb_image[backgroundIndex + color] =
                        static_cast<float>(background.rgb_image[backgroundIndex + color]) * beta
                        + static_cast<float>(foreground.rgb_image[foregroundIndex + color]) * alpha;

            }
        }
    }
    return true;
}
```

Here we parallelise the 2 loops that are perfectlly nested.
Infact we can use collapse and, since in our implementation the foreground is always the same, we set it a shared.

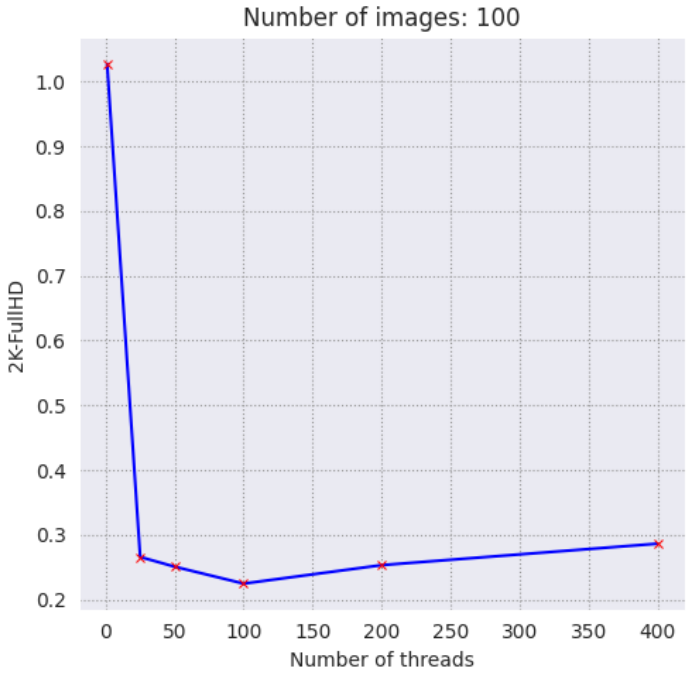
Parallel work: HD over FullHD

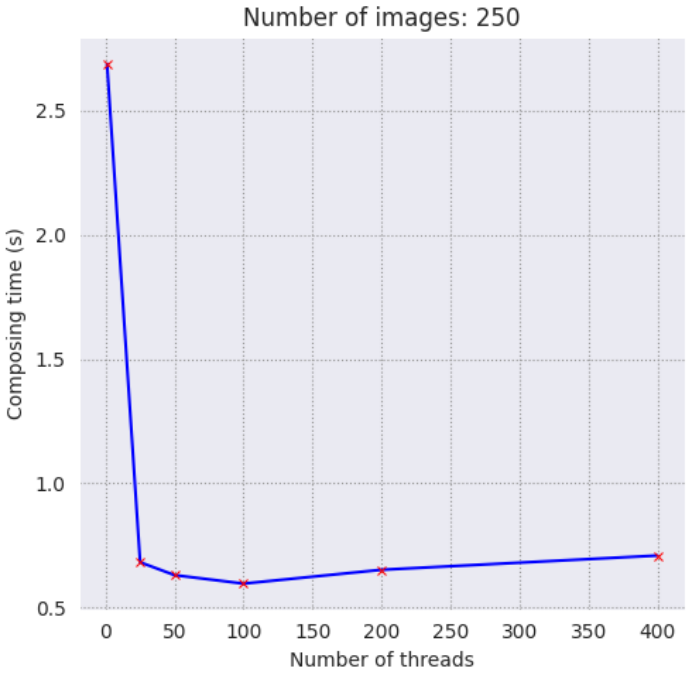


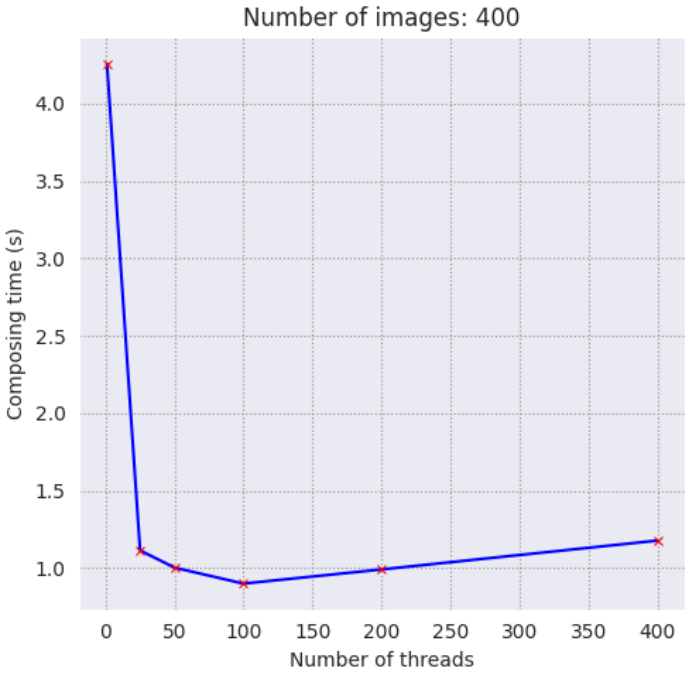




Parallel work: FullHD over 2K







Parallel work: 2K over 4K