Alpha Compositing

Parallel Computing 2023-2024

Christian Mancini

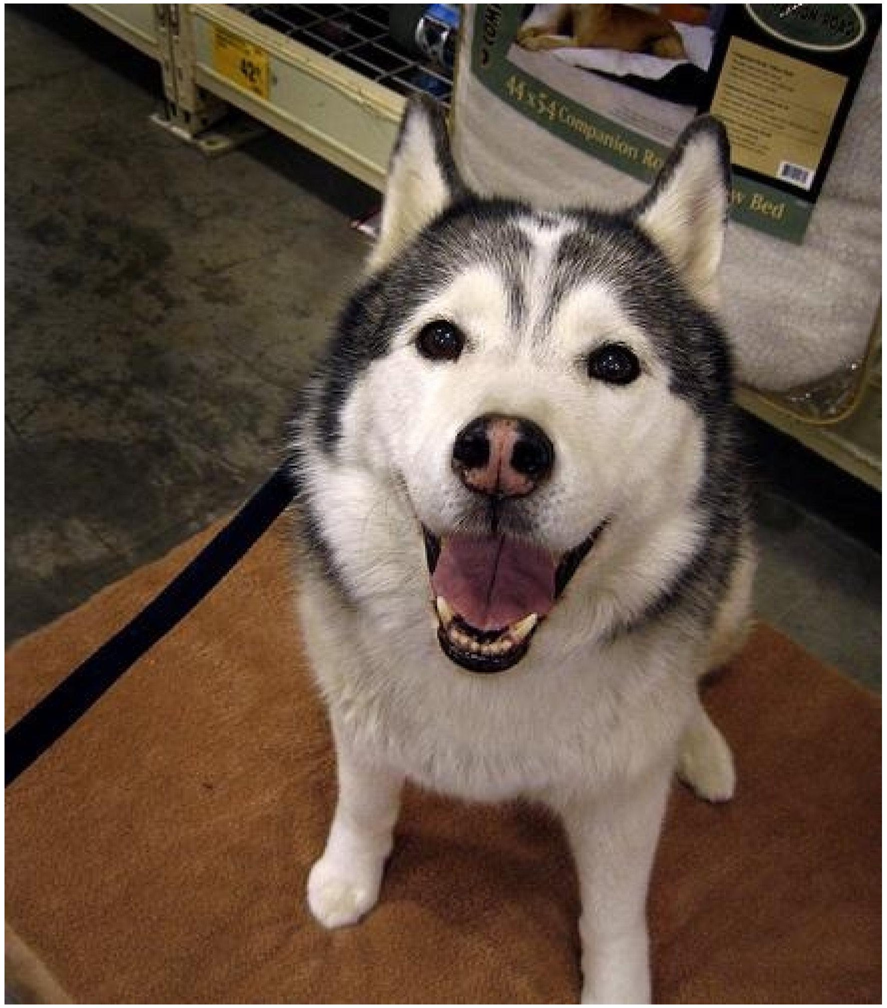
Florence, 12 March 2024

Alpha Compositing

What is alpha compositing

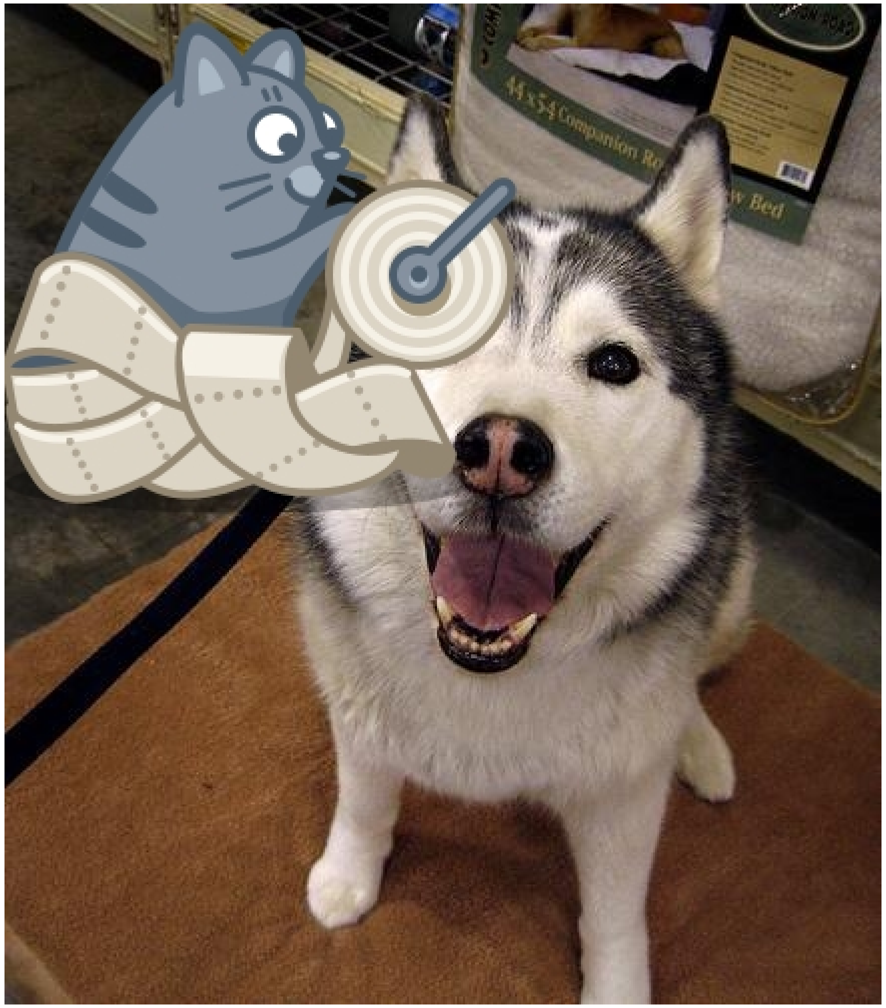
Alpha compositing is the process of combining one image with a background to create the appearance of partial or full transparency.

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 (left) and background (right)

Compositing Result

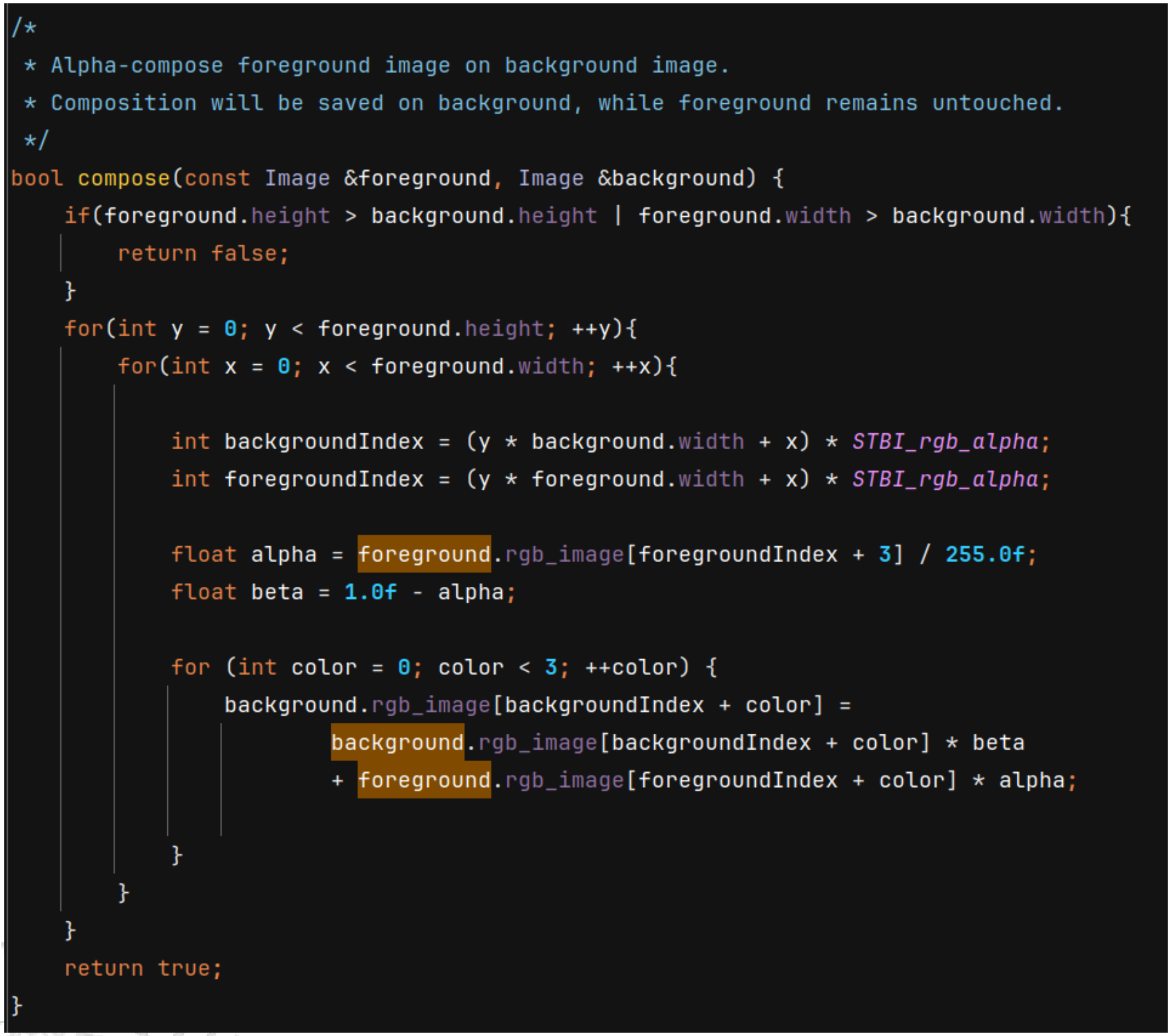


Alpha Compositing Formula

$$[RGBA]_d = [RGBA]_{\square} + [RGBA]_d (1 - A_s)$$

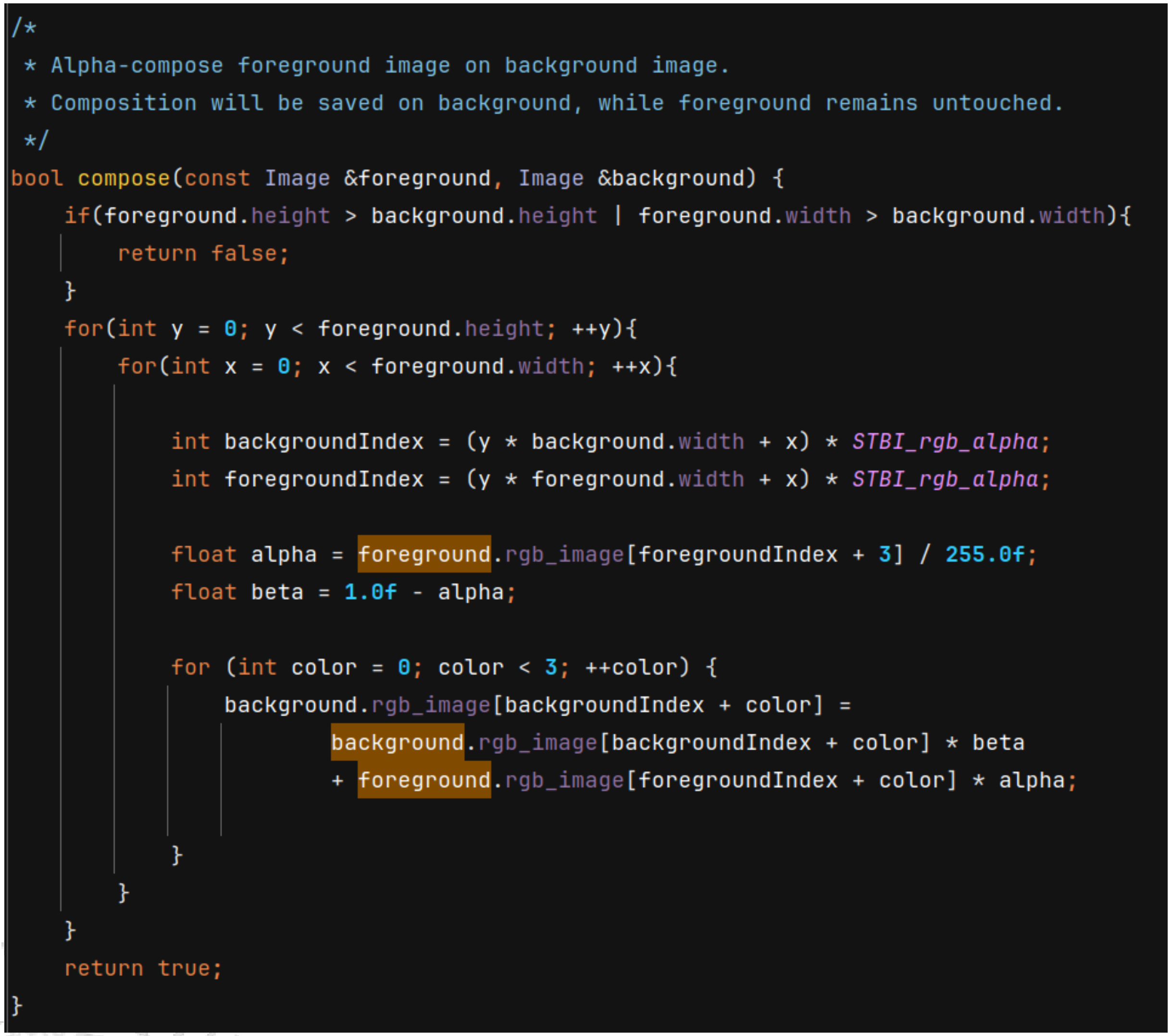
The formula is very simple, we just have to apply it on each pixel.

Implementation



```
/*
 * Alpha-compose foreground image on background image.
 * Composition will be saved on background, while foreground remains untouched.
 */
bool compose(const Image &foreground, Image &background) {
    if(foreground.height > background.height | foreground.width > background.width){
        return false;
    }
    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 = foreground.rgb_image[foregroundIndex + 3] / 255.0f;
            float beta = 1.0f - alpha;

            for (int color = 0; color < 3; ++color) {
                background.rgb_image[backgroundIndex + color] =
                        background.rgb_image[backgroundIndex + color] * beta
                        + foreground.rgb_image[foregroundIndex + color] * alpha;

            }
        }
    }
    return true;
}
```

Implementation

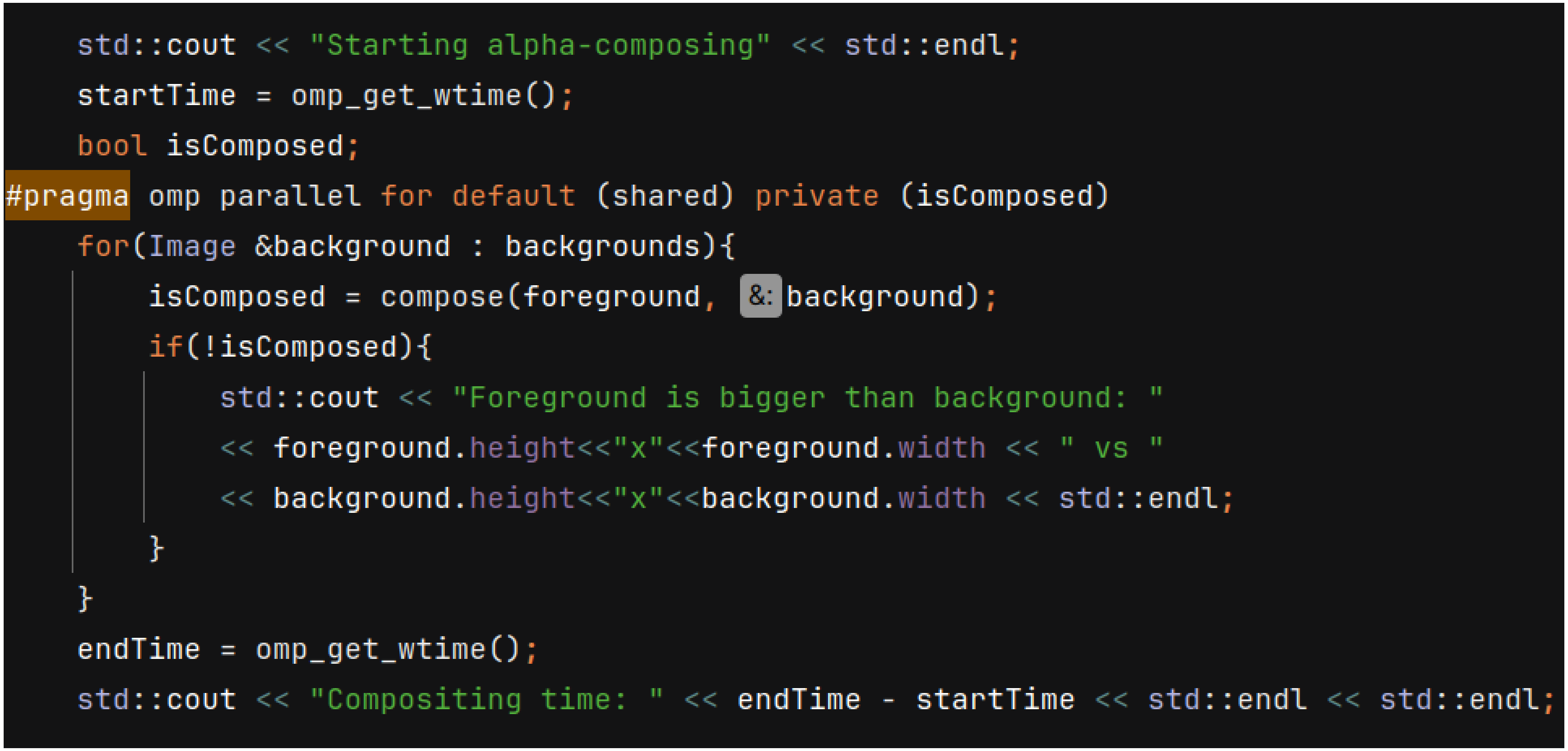


```
/*
 * Alpha-compose foreground image on background image.
 * Composition will be saved on background, while foreground remains untouched.
 */
bool compose(const Image &foreground, Image &background) {
    if(foreground.height > background.height | foreground.width > background.width){
        return false;
    }
    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 = foreground.rgb_image[foregroundIndex + 3] / 255.0f;
            float beta = 1.0f - alpha;

            for (int color = 0; color < 3; ++color) {
                background.rgb_image[backgroundIndex + color] =
                        background.rgb_image[backgroundIndex + color] * beta
                        + foreground.rgb_image[foregroundIndex + color] * alpha;

            }
        }
    }
    return true;
}
```

This implementation works with the serial and the parallel version since we decided to let a thread do a composition.

We will use OpenMP during the function call.

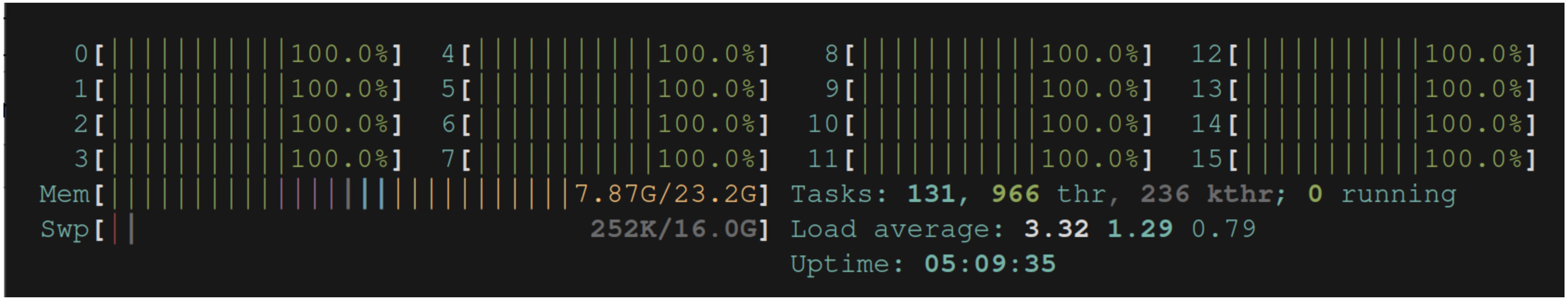
Parallel Call



```
    std::cout << "Starting alpha-composing" << std::endl;
    startTime = omp_get_wtime();
    bool isComposed;
#pragma omp parallel for default (shared) private (isComposed)
    for(Image &background : backgrounds){
        isComposed = compose(foreground, &: background);
        if(!isComposed){
            std::cout << "Foreground is bigger than background: "
            << foreground.height<<"x"<<foreground.width << " vs "
            << background.height<<"x"<<background.width << std::endl;
        }
    }
    endTime = omp_get_wtime();
    std::cout << "Compositing time: " << endTime - startTime << std::endl << std::endl;
```

We just need a private copy the boolean variable isComposed

Parallel work



```
   0[|||||||||||100.0%]   4[|||||||||||100.0%]    8[|||||||||||100.0%]   12[|||||||||||100.0%]
   1[|||||||||||100.0%]   5[|||||||||||100.0%]    9[|||||||||||100.0%]   13[|||||||||||100.0%]
   2[|||||||||||100.0%]   6[|||||||||||100.0%]   10[|||||||||||100.0%]   14[|||||||||||100.0%]
   3[|||||||||||100.0%]   7[|||||||||||100.0%]   11[|||||||||||100.0%]   15[|||||||||||100.0%]
 Mem[||||||||||||||||||||||||||||7.87G/23.2G] Tasks: 131, 966 thr, 236 kthr; 0 running
 Swp[||                           252K/16.0G] Load average: 3.32 1.29 0.79
                                              Uptime: 05:09:35
```

Every core is working during the parallel execution