esiea
INGÉNIEUR·E·S D'UN NUMÉRIQUE UTILE

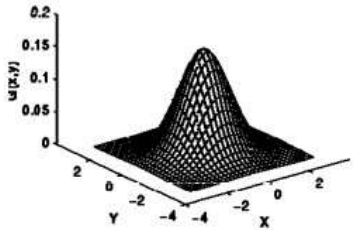
Image Processing Using CUDA

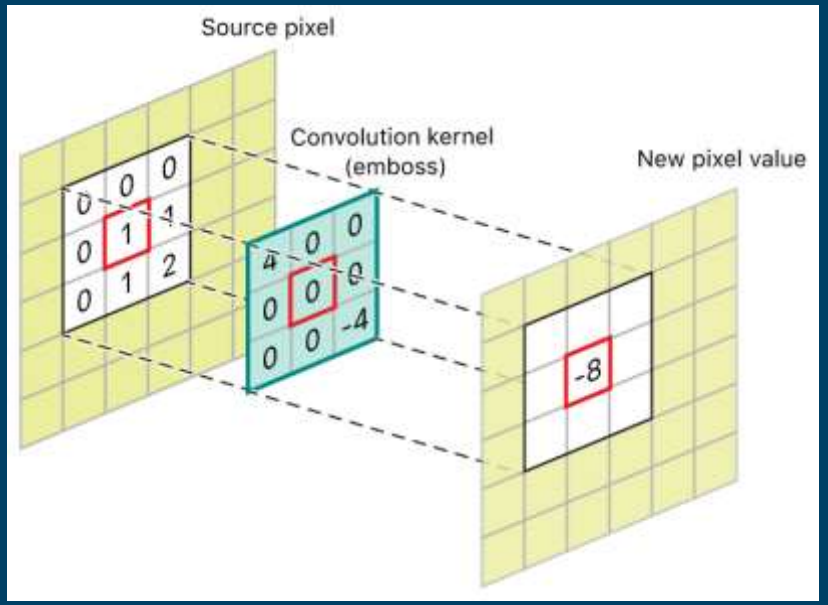
A Presentation by Leslie TIENTCHEU

Introduction

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

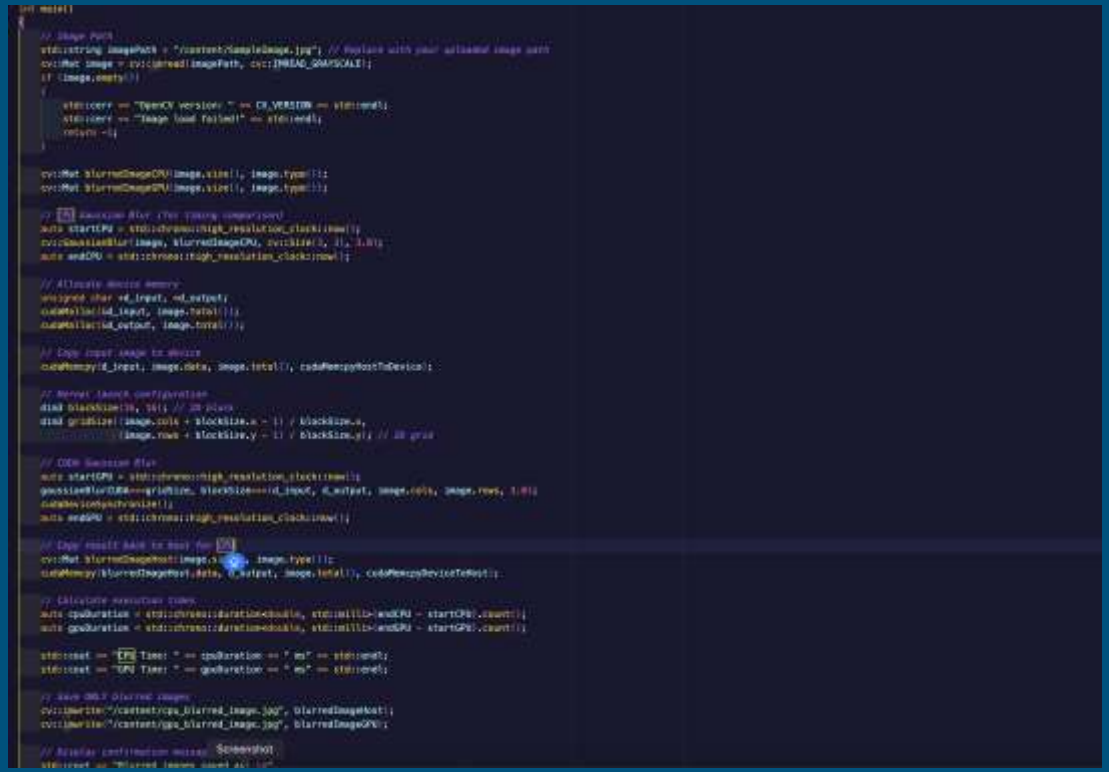
A graphical representation of the 2D Gaussian distribution with mean(0,0) and σ = 1 is shown to the right.





GaussianBlur

Base Sequential Code



Snippet of base sequential

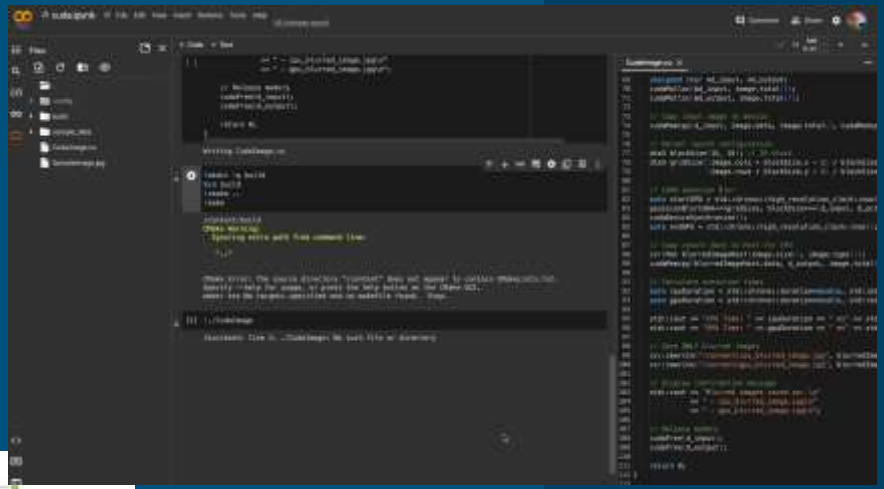
```
// CPU Gaussian Blur (for timing comparison)
auto startCPU = std::chrono::high_resolution_clock::now();
cv::GaussianBlur(image, blurredImageGPU, cv::Size(3, 3), 3.0);
auto endCPU = std::chrono::high_resolution_clock::now();
```

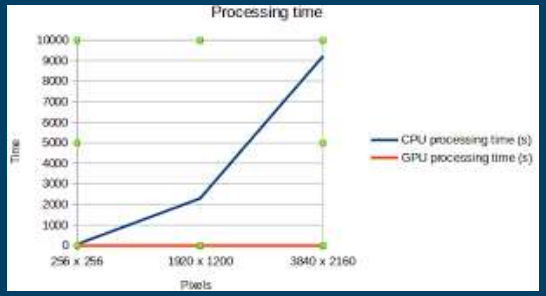
CUDA Kernel - gaussian Blur CUDA

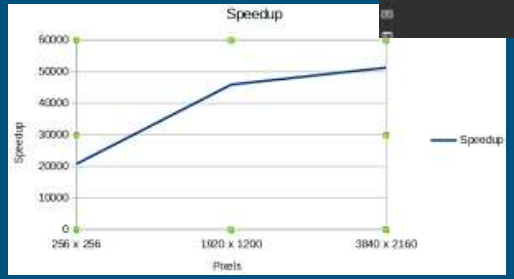
Snippet of CUDA Kernel

```cpp
// CUDA kernel for Gaussian blur
__global__ void gaussianBlurCUDA(const unsigned char* input, unsigned char* output,
                                 int width, int height, float sigma) {
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x < width && y < height) {
        float sum = 0.0f;
        float totalWeight = 0.0f;

        // Sample 3x3 neighborhood for simplicity... you can increase the kernel size
        for (int dy = -1; dy <= 1; dy++) {
            for (int dx = -1; dx <= 1; dx++) {
                int pixelX = x + dx;
                int pixelY = y + dy;

                if (pixelX >= 0 && pixelX < width && pixelY >= 0 && pixelY < height) {
                    float weight = gaussianWeight(dx, dy, sigma);
                    sum += input[pixelY * width + pixelX] * weight;
                    totalWeight += weight;
                }
            }
        }
        // Normalize and cast to unsigned char before assigning to output
        output[y * width + x] = (unsigned char)((sum / totalWeight) + 0.5f);  // Add 0.5 for rounding
    }
}
```

Performance Comparison & Execution Times







Results - Images

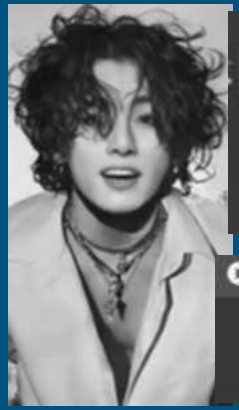
Real-image

BlurredImage

Real-image

```
!./CudaImage

CPU Time: 25.2502 ms
GPU Time: 1.7666 ms
Blurred images saved as:
 - cpu_blurred_image.png
 - gpu_blurred_image.png
```

Real-image



```
!./CudaImage

CPU Time: 29.9277 ms
Blurred image saved as:
 - cpu_blurred_image.jpeg
```

```
!./CudaImage

GPU Time: 0.485882 ms
Blurred image saved as:
 - gpu_blurred_image.jpeg
```

BlurredImage

```
!./CudaImage

CPU Time: 5.02373 ms
Blurred image saved as:
 - cpu_blurred_image.jpg
```

```
!./CudaImage

GPU Time: 1.49642 ms
Blurred image saved as:
 - gpu_blurred_image.jpg
```

BlurredImage

Enhancements for Better Performance

As we work on making our GPU-accelerated blur even better, let's focus on five main areas that can bring significant improvements:

Make Shared Memory Work Smarter:

- Improve overall speed by optimizing how we use shared memory. Smart caching of frequently accessed data in shared memory can speed things up by reducing delays in memory access.

Get the Kernel Just Right:

- Boost performance by finding the best setup for threads and blocks in the Gaussian blur CUDA kernel. A well-optimized kernel can really make the GPU work more efficiently.

Adapt to Image Details:

- Adjust the blur kernel size dynamically based on different parts of the image. This adaptive approach ensures we blur things just right, taking into account various details in the image.

Smooth Data Movement:

- Make the most of the GPU by using asynchronous data transfer between the CPU and GPU. This helps us do multiple things at once, making better use of the GPU's abilities.

Use Profiling Tools Wisely:

- Take advantage of CUDA profiling tools to understand where things can be faster. Profiling helps us find specific areas that need improvement, ensuring we focus on what really matters.

Thank You!!