Project
An ant simulation

January 10, 2022

1 Introduction

The project needs to be sent four days before the final exam. Only the source code and the report are required (no binary code). Please send it to robinfaurypro@gmail.com. In the rapport, you must discuss any technical choices you did. Why you prefer to create a buffer or a surface, why you recompute this value, etc... The project is an ant simulation. Ants must leave the anthill to find some resources. The only ability they have is to put pheromone on the floor. Thus pheromones can store information like the distance to the anthill or the food.
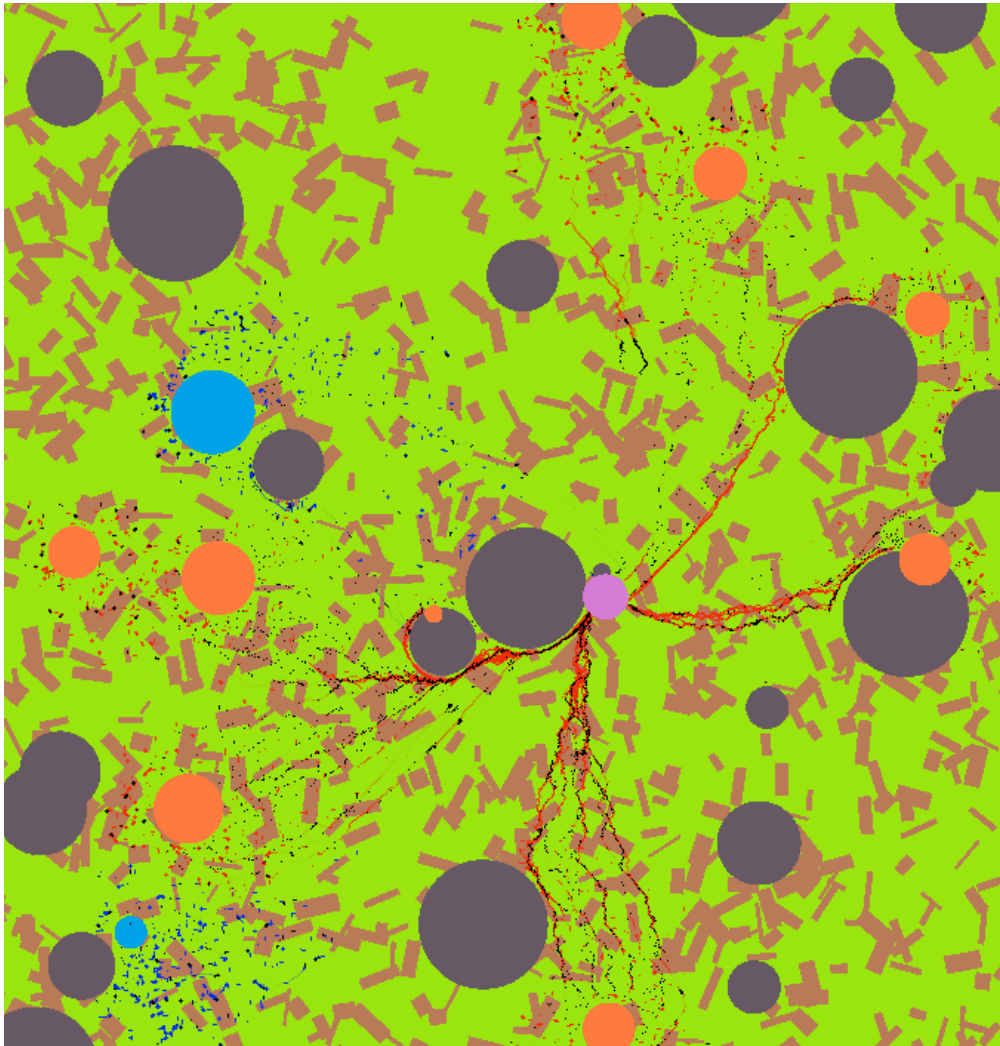

Figure 1: The ant simulation

The initial code of the project can be find in this git repository:

$$https://github.com/robinfaurypro/GPGPU\_ISIMA\_2021-2022.git$$

For this simulation, we will use a real time application. The CUDA part will compute the image to show as fast as possible and OpenGL will print this image on the screen. Working with image is more complex than working with buffer. For the moment we use only buffer to keep simple. However, to print on the screen a buffer, we need first to convert it as a texture. A better option is to work directly with texture. GPU texture can be seen in the same way than buffer, but they come with a collection of tool (interpolation, repeat, ...).

For this project, we ask OpenGL to generate the texture to show and we use the interop feature to convert it into a surface. Surfaces can be send to the kernel in the same way than buffers. You can create a float4 on the kernel to fill the surface with your color. You can also see how to create a surface CUDA in the exemple (Creation of a surface cuda).

```
__global__ void kernel_draw_map(cudaSurfaceObject_t surface) {
        int32_t x = blockIdx.x * blockDim.x + threadIdx.x;
        int32_t y = blockIdx.y * blockDim.y + threadIdx.y;
        float4 color = make_float4(0.6f, 0.9f, 0.05f, 1.0f);
        surf2Dwrite(color, surface, x * sizeof(float4), y);
}
```

2 Drawing the map

The CUDA languages support the struct keyword. You can create a struct for the rock or the stick or the water and use thus structures in your kernel.

```
struct Rock {
        float u;
        float v;
        float radius;
};

__global__
void kernel_draw_map(cudaSurfaceObject_t surface, Rock* rocks, int32_t nb_rocks)
```

Generate a random map by using the std::uniform_real_distribution tool. You can create a buffer for each type of object you want in your scene or one big buffer with all of them. In this exemple I choose:

- 42 rocks with the color 0.4f, 0.35f, 0.39f, 1.0f

- 3 water source with the color 0.725f, 0.478f, 0.341f, 1.0f

- 15 food spot with the color 1.f, 0.478f, 0.241f, 1.0f

- 2000 sticks with the color 0.725f, 0.478f, 0.341f, 1.0f

- 1 anthill with the color 0.839f, 0.486f, 0.839f, 1.0f

2.1 Draw a circle

To draw a circle you can use the hypotf function from CUDA. This function compute the distance between two point.

```
if (hypotf(food[i].u - uv.x, food[i].v - uv.y) < food[i].radius) {
        color = FOOD_COLOR;
}
```

For your information, operator can be overridden to create useful function. In this case we can create this one:

```
__device__ float2 operator-(float2 a, float2 b) {
        return make_float2(a.x - b.x, a.y - b.y);
};
```

You can notice the __device__ keyword to specify the function to the device. This function is accessible only by other device functions or kernel functions.

2.2   Draw a oriented box

Oriented box is more difficult to generate. Fortunately, drawing shape is well define right now and you can find useful formulas in this website (https://www.iquilezles.org/www/articles/distfunctions2d/distfunctions2d.htm). Thus content is very often use by people working on shadertoy (https://www.shadertoy.com). If you want to learn more about graphic computing, I recommend you to take a look in detail in the code of some shaders.

```
float OrientedBox( in vec2 p, in vec2 r, float ang ) {
    vec2 w = vec2(cos(ang),sin(ang));

    vec4  q = w.xyyx * p.xxyy;
    vec4  s = w.xyyx * r.xxyy;

    return max(
        max( abs(q.x+q.z)-r.x, abs(q.w-q.y)-r.y ) /
        max( abs(w.x-w.y), abs(w.x+w.y)),
        max( abs(p.x)-max(abs(s.x-s.z),abs(s.x+s.z)),
        abs(p.y)-max(abs(s.y+s.w),abs(s.y-s.w))));
}
```

Be careful, this code is in GLSL. You need to convert it into CUDA code if you want to include some stick in your project. The r.xxyy command is equivalent to make_float4(x, x, y, y).
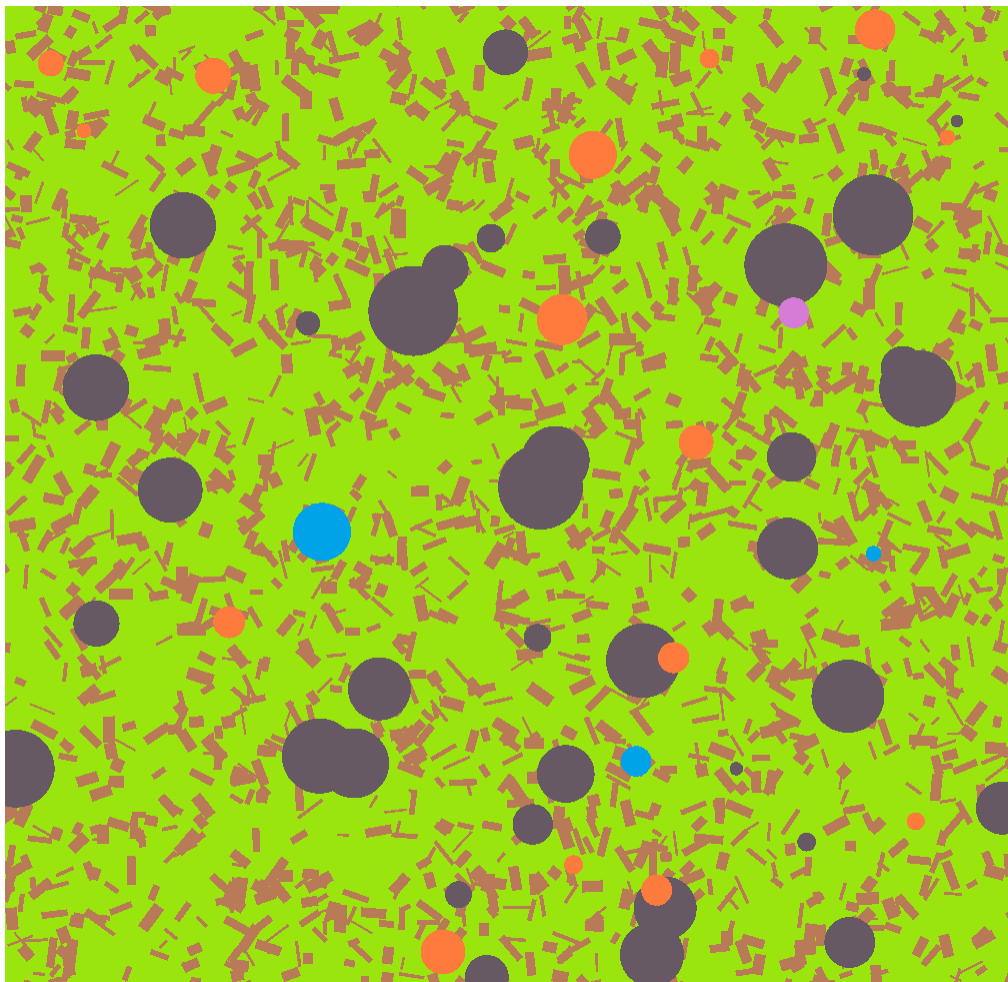

Figure 2: The map

3 FPS

If you make the choice to draw the same amount of object in your scene, you need to check the performance of your algorithm. To do so, you need to profile your kernel. A good option is to use the nvprof tool from the CUDA SDK or a custom computation of the Frame Per Second. Most of the time, the FPS are computed simply because it a better metric when you work on a software. Be careful, Drawing the FPS value a lot of time per second will slow down your simulation. A good option is to print it every second in the terminal or in the title of the window.

4 Ants

The aim of the ants is to move from the anthill and go to the food or the water and come back when they find some resources. They can go through stick but you cannot walk on rock. Ant can have for example a forward direction and a radius of detection. Moving ants randomly may be the first step to implement. To do so, we can rotate the direction of an angle between $\frac{-\pi}{4}$ and $\frac{\pi}{4}$ and add the new direction of the ant to its position. The maximum random angle can be part of the ant struct or hard codded. Having an angle smaller than π will force the ant to explore the environment instead of randomly walking around the anthill. Unfortunately, random in GPGPU is not easy. You can use cuRand to create random number or you can using this formula:

```
__device__ float random(float x, float y) {
        float t = 12.9898f*x + 78.233f*y;
        return abs(fracf(t * sin(t)));
}
```

We simply generate a sinus function with a very high frequency and we grab only the absolute value of the fractional part of the result. This give us a number between 0 and 1. The function need a seed generator, the value of u and v are good candidates.

4.1 Pheromones

Ants use pheromones to communicate with other ants. A good way to implement it is to create a new surface to store the value of the pheromone. Each ant will send a pheromone with a value of 0.01 and will try to find the strongest pheromone around. For each frame, each value of pheromone will be reduce by 0.0001 for the pheromone of the anthill and 0.03 for the pheromones of the food and the water for example. Thus values can be adapted if you choose a large among of ants. Basically, less evaporated pheromones will lead to more sub optimal path. A strong evaporation will lost ants.
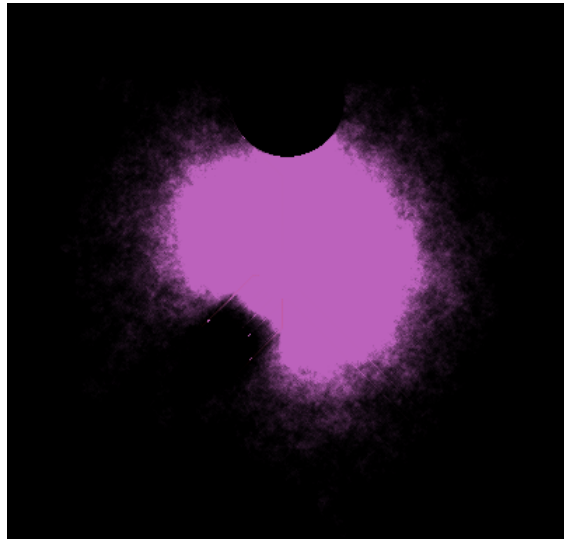

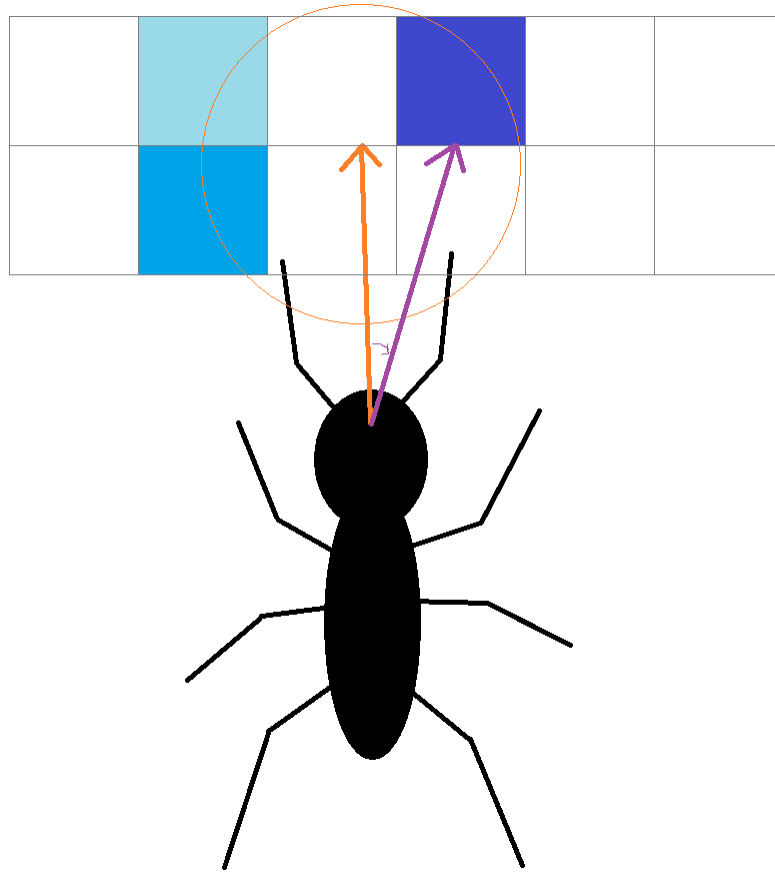Figure 3: The known map

4.2 Read the pheromone

Figure 4: The known map

Because ants can only perceive the environment in front of them we must only consider front pixel. In the example, the ant can read the six pixel in front of its. Reading the pheromone is done in a loop over the pixel depending of the size of the perception radius. A good perception radius is more than 5 pixels. The dark blue square means a lot of ants find some water and put some water pheromone in this case. The more ant stacks pheromones the more the pixel become blue. We can now ask to the ant to rotate in the direction of the best candidate.

4.3 Write the pheromone

Just before moving, the ant need to store a pheromone in its position.A good idea is to write the value in the same pheromone surface you use decide where you want to go. However, if one ant finish its move and write its pheromone will another ant is currently reading the same pixel, you will have a read/write conflict. You don't want to read a value will another thread to currently writing a new value in the same pixel. To avoid this situation you can tell to threads to only read the pixel, synchronize all the threads and then write the new value. The CUDA function to do it is : _syncthreads(); You can use it inside the kernel between the read and write.