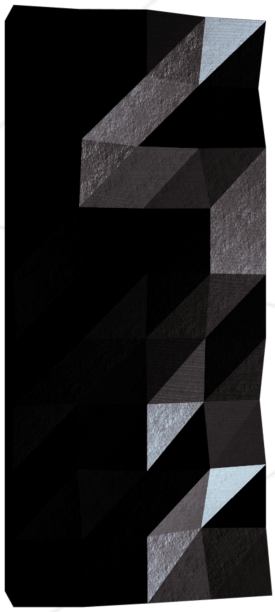


Particle System

OpenCL introduction

Summary: This project is an introduction to parallel computing on GPUs using libraries such as OpenCL, OpenGL, CUDA, Metal, or Vulkan.

Version: 4

Contents

I	Forewords	2
II	Introduction	3
III	Goals	4
IV	General instructions	5
V	Mandatory part	6
VI	Bonus part	7
VII	Submission and peer-evaluation	8
VIII	Illustrations	9

Chapter I

Forewords

Compute shaders operate differently from other shader stages. All of the other shader stages have a well-defined set of input values, some built-in and some user-defined. The frequency at which a shader stage executes is specified by the nature of that stage; vertex shaders execute once per input vertex, for example (though some executions can be skipped via caching). Fragment shader execution is defined by the fragments generated from the rasterization process.

Compute shaders work very differently. The "space" that a compute shader operates on is largely abstract; it is up to each compute shader to decide what the space means. The number of compute shader executions is defined by the function used to execute the compute operation. Most important of all, compute shaders have no user-defined inputs and no outputs at all. The built-in inputs only define where in the "space" of execution a particular compute shader invocation is.

Therefore, if a compute shader wants to take some values as input, it is up to the shader itself to fetch that data, via texture access, arbitrary image load, shader storage blocks, or other forms of interface. Similarly, if a compute shader is to actually compute anything, it must explicitly write to an image or shader storage block.

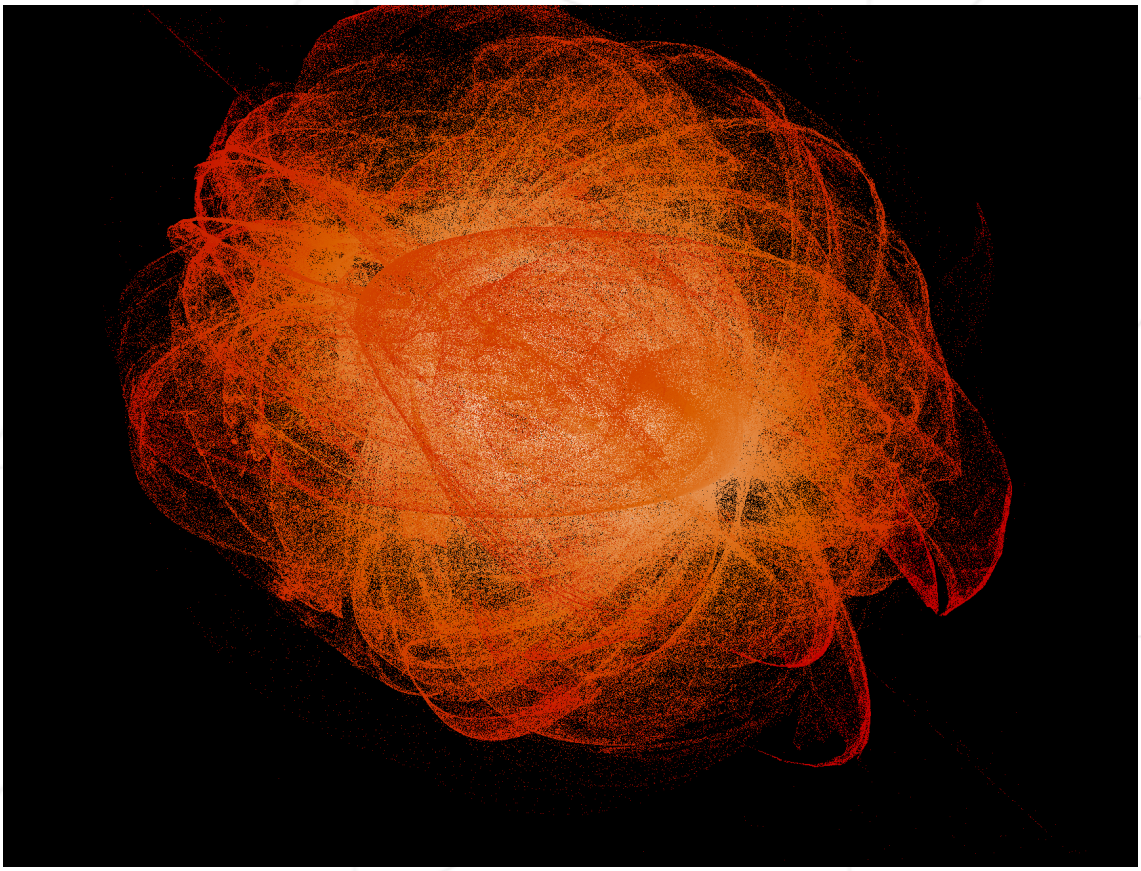
- Core in version 4.5 (2014)
- Core since version 4.3 (2012)

Compute shaders are awesome, they're easy to use and efficient when you need to mix rendering and computation. They are in OpenGL since version 4.3 (2012). In 2015,

Chapter II

Introduction

In this project, you will implement a particle system with a high number of particles (the more, the better). To achieve this, the GPU is your friend, as it allows you to massively parallelize your computations.



This is what the project could look like.

Chapter III

Goals

You will learn to use GPU programming libraries such as OpenCL, CUDA, Vulkan, Metal, or OpenGL — along with their associated kernel or shader languages — to bring your particle system to life.

You will also work on rendering, strengthening your knowledge of graphical APIs like OpenGL, Metal or Vulkan.

This project is a great introduction to performance optimization. Don't hesitate to research, experiment, and test various approaches to achieve the best results. In fact... you'll have to. So get to it — and it'll be much easier if you enjoy the process.

Chapter IV

General instructions

- You must use recent GPU libraries, with at least the latest stable major version (e.g., OpenGL 4.x, OpenCL 1.2+, Vulkan, CUDA, or Metal).
- The particles shall never be allocated on the RAM. Never. Even on initialization. Never. If you ever ask yourselves "Could I, now ?", the answer will always be "Never. Never, ever.". Everything will be allocated on the video memory.
- Performance wise, it'll be expected of you to run at least one million particles at 60 fps and three million (minimum) at 20 FPS. For testing purposes, the number of particles must be easily modifiable.
- You must use interoperability and, for cleanliness you have to synchronize the shared memory. Memory acquire and release functions are your friends.
- You are free to use whatever language you want.
- You can use the graphic library of your choice (GLFW, SDL2, Glut, MLX ...).
- A Makefile or something similar is required. Only what contains your repository will be evaluated.

Chapter V

Mandatory part

You have to implement a particle system. These particles will be initialized in a sphere or a cube shape and it will be possible to switch between the shapes with inputs. They must be attracted to a gravity center which can be turned on or off with an input. This gravity center can be static (initialized under the cursor when the key is pressed) or follow the cursor depending on the input.

It's necessary for them to be allocated on the GPU memory (VRAM). It will be expected of you to consider some performance constraints : one million particles at 60 fps and three millions at 20.

You must print the FPS counter on the window (in the title for example), this will make perfomances check easier.

For viewing pleasure, you will put some colors. They will depend at least on the distance between the particles and the cursor.

Remember that cleanliness for interoperability is really, really important. If you don't know what we're saying, read again the general instructions.

`The particles don't require a mesh.`

Chapter VI

Bonus part

When you are sure that everything works fine and your performances meet all constraints, here are some bonuses that could be fun to add :

- A camera which travel in this beautiful particles world, controlable with WASD and the mouse. Because that's freakin' cool.
- Emitters generating particles with life span.

There will be some points dedicated to these bonuses and some more for your creativity.

Keep in mind that performances constraints won't apply here.. that's not a reason to go with 10.000 particles at 20 FPS.

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter VII

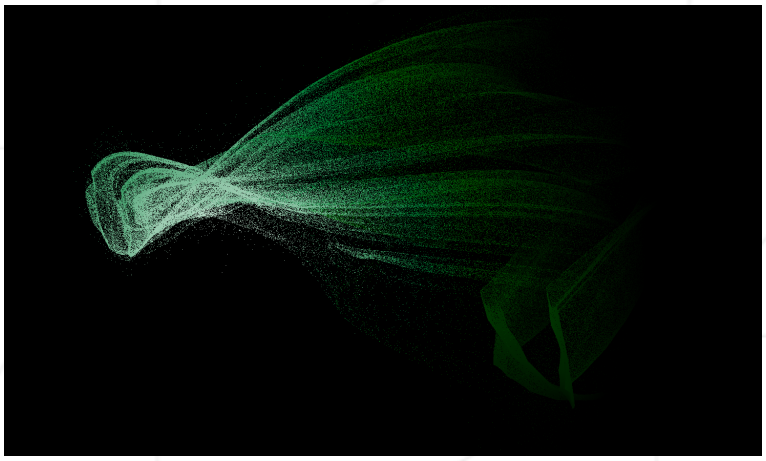
Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

Chapter VIII

Illustrations

Particles following the cursor:



Rotating around a gravity center: