GPU Accelerated Particle System

Project in The Course TNCG14 Advanced Computer Graphics

by August Ek and Oscar Johnson

ABSTRACT

The aim of this project was to create a particle system mainly accelerated by the GPU. The system would then be used to render different effects. The system was accelerated using instance rendering and transform feedback leading to a huge performance gain. The particle system created was used to render different visualizations such as smoke, fire and rigid objects colliding with planes.

1 INTRODUCTION

The graphics processing unit have a lot of potential which could be used in graphics calculations and even in arbitrary calculations surrounding these. One example well suited for this is a particle system. A particle system can be used in various applications. For example simulation of fluids, explosions and animation of hair.
The objective of this project is to extract the efficiency of the GPU. To demonstrate the power gained after GPU acceleration a particle system was implemented. Using particles to demonstrate gravitational physics and a simulation of smoke and fire with the use of perlin noise.
This project uses opengl 4.1 which is a constraint. No compute shader is available to make non graphical calculations on the GPU.

2 BACKGROUND AND RELATED WORK

GPU:s primary goal have been to render graphical elements, however with time the GPUs functionality has been extended. Even non graphical calculations is possible on the GPU, and should often times be made there. Which could increase the performance by a great factor. The liability is now more with the programmer to make use of the GPU and moving much of the logic from the CPU to the GPU. This to fully exploit the efficiency of the computer.
Even though compute shaders are not available in opengl 4.1 arbitrary calculations can be made on the GPU using a technique called transform feedback. This resolves the obstacle of not being able to do computations on the GPU. Another obstacle in GPU programming are the calls that has to be made from the CPU to GPU. These can create a bottleneck as well. Instancing is another technique which reduces the calls that has to be made. This basically creates a prototype of the particle instead of creating a new instance of each particle.
There are some related articles that we have used as material. These articles include information about smoke simulation using perlin noise and are referenced in the report.

3 RESULTS

The result was a simple particle engine with capability to spawn 200 times more particles than without GPU acceleration. The particle system implemented has these features:

- A perlin noise based fluid simulation
- Gravitational Physics
- Loading of PNG and OBJ files to particles

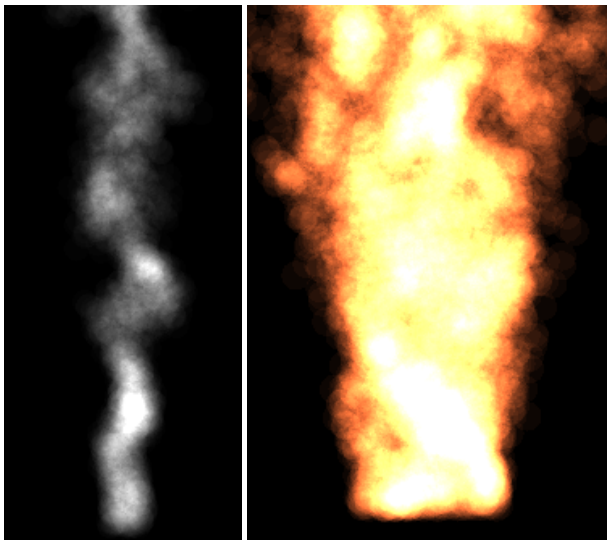

Figure 1: Smoke and fire rendered with a noise based velocity field.

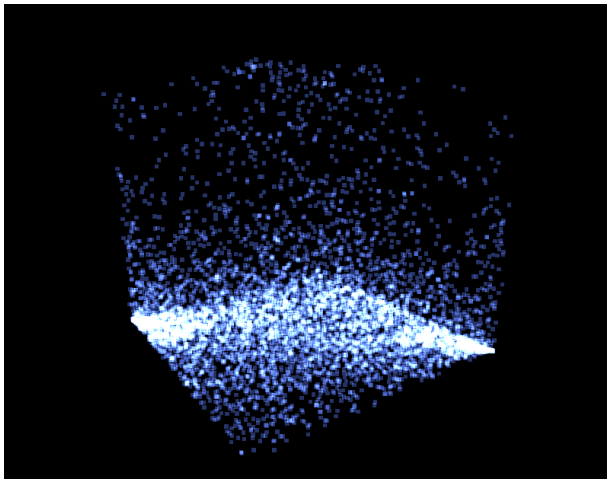

Figure 2: Particles colliding inside a bounding box.

	30 FPS	60 FPS
No Instancing, no Transform Feedback	10 000	5000
Instancing	300 000	100 000
Instancing and Transform Feedback	4 000 000	1 000 000

4 IMPLEMENTATION DETAILS

4.1 Particles

Each particle has two attributes: position and velocity. These two will need to be updated each frame before the particle is rendered to the screen. All of the particles are created at the start of the application and are then used to fill out the vertex buffer containing the positions and velocities a little by little.

4.2 Calculating positions

The new positions for the particles was calculated using transform feedback. This means that a shader was written that had the single purpose to calculate new positions for each particle. The resulting buffer from the shader was then used to draw the particles on the screen. Using this method gives a huge performance boost since all of the calculations could be moved from the CPU to the GPU, utilizing the parallel process power of the GPU and not relying on the few cores offered by the CPU. The figure below shows how the main particle loop works.

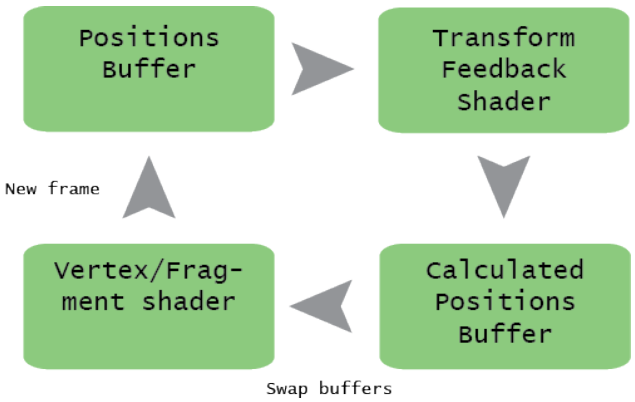

Figure 3: Diagram describing the transform feedback. Positions and velocities are used as input to the transform feedback shader which produces new positions. These positions are then used to render the particles on the screen.

4.3 Rendering particles

Instanced rendering was used to draw all of the particles with a single call to the GPU. This basically tells the GPU to render the same particle at different positions instead of render a different particle for each position. This offered a performance boost since the overhead related with making each API call would go from $O(n)$ to $O(1)$ per frame.

4.4 Placing particles

To make placing of particles easier and create more interesting visuals of the particles, an importer was made. This functionality receives either a binary png or an obj file. If an image is received particles are placed on a plane which represents the image. White pixel represents a particle while black pixel represents empty space. Receiving an obj file works similarly, a particle is instead represented by a vertex and placed in 3D space rather than in a 2D plane.

Figure 4: Particles placed at positions in the shape of a teapot.

4.5 Simulation

Two different simulations were made to test the capabilities of the particle system. The first one uses classical mechanics to demonstrate particles colliding with planes and the second uses a noise based velocity field to simulate smoke and fire.

4.5.1 Classical mechanics

The physics implementation includes basic gravitational force. For each frame a new velocity and position is calculated due to gravity. A bounding box is surrounding the particles allowing implementation of collision detection. This is done with a simple posteori collision detection. Meaning if the particle moves outside the boundings, move it within the boundings in and calculate the new velocity. The new velocity is then calculated with the reflection formula for vectors. If collision with the ground level would happen, the new velocity gets scaled with a number between 0.0 and 1.0. This is the coefficent of restitution, which represents the energy loss of the particle upon impact.
A calculation of the particles energy is done each frame. Using the particles mass, the gravity, distance to the bottom of the bounding box and velocity. This yields the kinetic and potential energy of the particle. If the energy value is less than a small given value, the particle is assumed still. To avoid unnecessary computation and vibrations at small energy values the velocity is set to zero and the position is assign previews value from then on.

4.5.2 Smoke

To make a smoke simulation correct from a physical point of view, there is the need to solve the equations of Navier-Stokes. However, to get a visually realistic result, procedural methods could be used to create the vector field which moves the particles. This is easier to implement, more memory efficient and gives a result that looks realistic enough for this implementation.
The velocity field used was generated by using classic perlin noise[1]. The noise potential is simply a scalar field when implementing in 2D giving:

$$\psi = \psi(x, y, z, time) \tag{1}$$

x, y, and z are the positions of the particle, time is the elapsed time since the simulation started and ψ is the noise potential. Using the noise directly as the velocity field results in a field that diverges at certain areas thus producing a field with properties not desired for this type of simulation. Robert Bridson[2] proposes a solution to this by using the curl of the velocity field since one classic calculus identity is that $\nabla \cdot (\nabla \times \mathbf{F}) = 0$. Calculating the curl is done in the way:

$$\mathbf{v} = \nabla \times \psi \tag{2}$$

Since only two dimensions were used, ψ were set to:

$$\psi = (0, 0, \psi) \tag{3}$$

giving:

$$\mathbf{v} = (\frac{d\psi}{dy}, -\frac{d\psi}{dx}) \tag{4}$$

Which is the velocity of the particle at the given position and time. The partial derivatives is then evaluated using simple finite differences with very small displacements.
The result is a divergence-free field which is essential to produce realistic velocity fields such as those of smoke or water. A comparison of the two noises can be seen below.

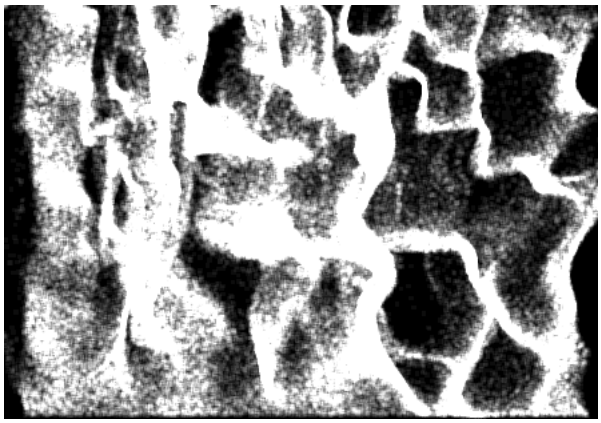

Figure 5: Classic noise used as velocity field, notice the gutters where the particles tend to move.

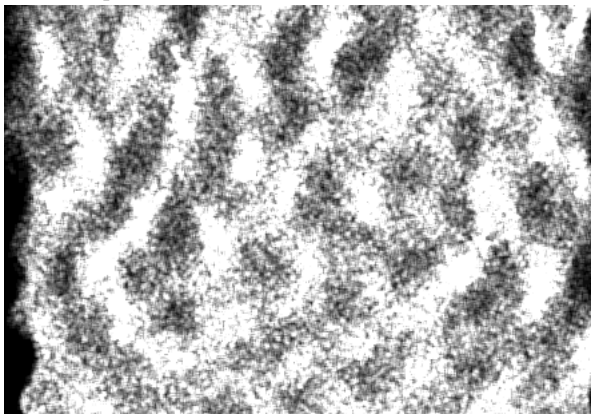

Figure 6: the curled noise used as vector field, this field is completely divergence - free.

Two computers with similar specifications were used in the implementation:

Macbook Air 2013	**Macbook Pro 2012**
1,3 GHz Intel Core i5	2,5 GHz Intel Core i5
8GB RAM	8GB RAM
Intel HD Graphics 5000	Intel HD Graphics 4000

None of these are specially powerful computers stating that there is no need for high-end hardware for this type of particle system.

5 Conclusion

The goal to implement a basic particle simulator was reached and by some extent exceeded. The simulator is using recent GPU techniques to boost the performance and using physics and vector calculus to simulate realistic scenarios. Transform feedback combined with instanced rendering removes the bottleneck of being limited to a low amount of particles. The system can easily handle hundreds of thousands of particles at smooth FPS rates even on low-end hardware. The particle system offers high modularity due to the fact that its very simple to replace the calculating vertex shader to give the particles a totally different pattern of motion thus simulating different phenomena.

6 Future work and Improvements

The smoke simulation could be improved in the way that the smoke respected boundaries giving the visual result that the smoke avoids some obstacles. This requires that the distance gradient of all obstacles is mixed with the noise-potential and we felt that we didnt have the time for this feature.
Endless tweaking on parameters for the vector field could be done to create other visuals such as water. There is no real limit on how this could be done since the only goal is to create something aesthetically pleasant. Most of the work of this project had something to do with the motion of the particles and thus the calculations done in the vertex- and transform shader. A lot of work could be done in the fragment shader to drastically improve the visuals.
We cant say that it wouldnt be interesting to implement a physically accurate simulation with regards on the equations of Navier-Stokes. That simulation could then be used as reference to compare with the current one in aspects of performance and visual realism.
A graphical user interface could be implemented to easy swap between different simulations and the user could also be able to change some of the parameters such as: number of particles in the system, size of each particle and the location where the particles should be spawned.

References

[1] McEwan, I., Sheets, D., Gustavson, S. and Richardson, M. 2011, *Efcient computational noise in GLSL.*

http://webstaff.itn.liu.se/~stegu/jgt2012/article.pdf

[2] Bridson, R., Hourihan, J. and Nordenstam, M. 2007, *Curl-Noise for Procedural Fluid Flow*. http://www.cs.ubc.ca/~rbridson/docs/bridson-siggraph2007-curlnoise.pdf