Metaball water simulation in Unity

Project specification, DH2323 Computer Graphics and Interaction

2020-05-05
Anders Steen (astee@kth.se)
Grade ambition: A

1 Background

Fluid simulation is a computationally difficult task, and there are multiple different approaches to creating such simulations [1]. Different contexts call for different requirements that are put on the results of the fluid simulation, some contexts require high physical accuracy, while others require plausibility as well as real time performance [1]. A context that has the latter requirements are video games, since they are required to be interactive [2]. There exists multiple methods for simulating fluids in video games, with different advantages and disadvantages [1, 2].

Water occurring in video games is not uncommon. Water occurs in games like the original Half-Life game [3] as well as the modern game Sea of Thieves [4]. While the water in these games look different and act different from each other, what these games have in common is that the water is not interacted with in any further extent than player characters swimming through it. Developing games in which players interact with water in more complex ways can be something desirable and somewhat novel.

There is a computer graphics concept called metaballs [5]. Metaballs are used to model surfaces (among other things) within computer software that can easily take on a distinctly organic look when rendered [5]. Particle systems where the particles are metaballs have been used to create water simulations, specifically realistic real time simulations of streams of water across terrain [6]. In that simulation, there is support for splashing water, and the metaballs are rendered using a billboard rendering technique. Other rendering methods are also suggested, making the simulation adaptable to the needs of the context.

When creating games, it is very common to make use of a so called game engine. The game engine is a development environment for creating video games, aiding the development by including essentials like a rendering engine, sound engine, physics engine, among other things [7]. A world leading game engine is one called Unity, which is free to use assuming you make very limited revenue from using it [8]. The use of Unity is widespread, with a large and active community of game makers of varying levels of experience [8].

2 Problem

An idea for a video game is to have a river flow down some terrain and let forests, societies and other phenomena grow organically around the river. Players should be able to interact with the terrain, changing elevation and other properties to alter the course of the river, and in that way affect the life around the river. One problem to solve in order to realize the game

idea is how the river should be constructed so that it looks plausible and is interactive in real time gameplay.

This project proposes to construct the river using a particle system of metaballs similar to [6], but with less photorealism, and using 3D metaballs with mesh rendering. The goal is to implement the simulation and render it in a rudimentary way inside Unity. This tests the utility of the proposed method of real time interactive water simulation in a real and popular video game development environment. This also evaluates whether the proposed method can be use to realize the aforementioned game idea.

3 Implementation

There are multiple different milestones that can be identified in this project. As stated in the section above, the least acceptable goal is to make the simulation in Unity, or to prove that it is not possible to simulate in this way in real time (at least in Unity). At the point where the least goal has been reached, the possibility of using the technique for constructing the river in the development of the game idea in Unity will have been determined.

3.1 Features

- Terrain rendered from heightmap texture in Unity.
- Metaball particle system:
    - Multiple settings:
        - Strength of forces, e.g. gravity and inter-particle forces.
        - Particle spawn point and spawn rate.
        - Number of particles.
        - Metaball radius.
    - Physical simulation with gravity, cohesive forces between metaballs and damping of metaball velocity.
    - Collision detection between metaballs and terrain triangle mesh:
        - Ray-triangle intersection tests.
        - Tree of axis-aligned bounding boxes for sped up collision detection.
- Movable camera using keyboard and/or mouse.
- Metaball rendering:
    - Defining bounds of metaball scalar field from metaball positions.
    - Calculating metaball scalar field from metaball positions and falloff function.
    - Polygonizing metaball scalar field into triangle mesh using marching cubes algorithm.
    - Rendering triangle mesh using the Unity rendering engine.
- Water and terrain shading using materials created in Unity.
- Extra version using Unity physics for comparison.
    - Native unity sphere colliders and rigidbodies for particles.
    - Native unity terrain collider for terrain mesh.

4 Final system sketch

The final system could look something like the concept pictures in figures 1 and 2.

Figure 1 shows a concept image created in the 3D software Blender, where a river runs down a plane. The final system will have the terrain be defined by a heightmap, rather than a triangle mesh as in figure 1, which will give less of the low-polygon-count look and more of a smooth terrain look. The colors and 3D look in figure 1 are accurate to the final system.

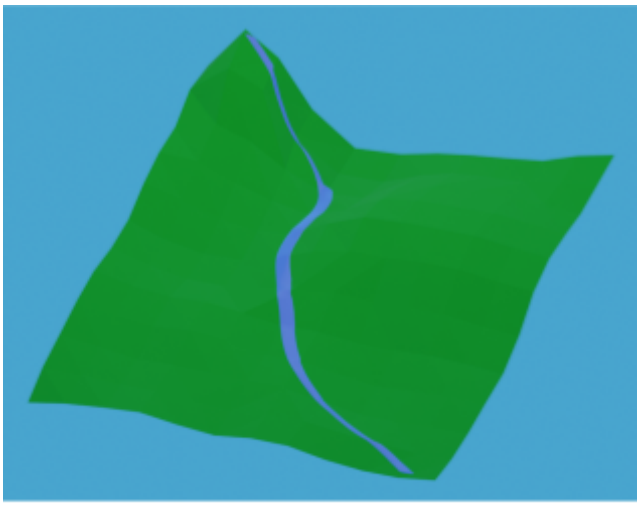

Figure 1: Low-poly concept river.

Figure 2 shows a concept image drawn in 2D. The image is meant to showcase the fact that the river should be able to fork and form lakes naturally given the correct terrain conditions. The colors and lighting of figure 2 might be inaccurate to the final system.

Looking at figures 1 and 2 should give a sufficient impression of what the final system will look like mid-simulation. The goal is to have the particle system start emitting particles at the top of the terrain, meaning the water will flow downwards during the course of the simulation. The particles will not have to despawn, they can be allowed to loop back to the top when reaching the bottom. After the particles have reached the end, the simulation should continue, producing results similar to those seen in figures 1 and 2.

Figure 2: River forking and lake formation.

The final system will also feature a version that uses the native Unity physics system to simulate the physics of the simulation. This addition is a mere curiosity, but can be used for comparison between the different physics systems used.

5 Risks and challenges

Below is a table describing potential risks and challenges specific to this project.

Description	Contingency plan
Unity has poor support for heightmaps	Some other method could be used to represent the terrain, e.g. a triangle mesh.
Visual plausibility of the simulation turns out worse than expected	Shifting the implementation to one more similar to the one proposed by [6].
Model will turn out less sophisticated than desired grade dictates	Extending the project by making the model more complex, using models described in [2] and its previous parts.
Collision detection turns out to be too difficult to implement	Using Unity native collision detection to achieve a visually satisfying result would be a good alternative.

6 Perceptual study

Important to video games is so-called suspension of disbelief, which is where a person faced by something surreal intentionally avoids thinking critically about what is being viewed [11, 12]. The importance of suspension of disbelief in video games lies in players accepting the game world despite in-game behaviors and phenomena are not realistic to the real world [12]. Simulating water in the way suggested in this project specification is not physically accurate, meaning its behavior in a video game would not be identical to real world water. Examining if players will suspend their disbelief when faced by the present water simulation is an important step of evaluating the utility and video game applicability of the water simulation method. A perceptual study related to this project could therefore be to evaluate to what degree players will suspend their disbelief when faced by the water simulation.

7 Project blog

The project progress will be updated continuously in a project blog, found at the following URL: https://metawater.blogspot.com/.

8 References

- [1] Mauricio Vines, Won-Sook Lee, and Catherine Mavriplis. “Computer animation challenges for computational fluid dynamics”. In: International Journal of Computational Fluid Dynamics26 (July 2012), pp. 407–434. doi:10.1080/10618562.2012.721541.
- [2] Michael J Gourlay. “Fluid Simulation for VideoGames Part 21: Conclusion”. In: (2016).
- [3] Wikipedia, 2020. *Half-Life (video game)*. https://en.wikipedia.org/wiki/Half-Life_(video_game) (Accessed 2020-04-03)
- [4] Wikipedia, 2020. *Sea of Thieves*. https://en.wikipedia.org/wiki/Sea_of_Thieves (Accessed 2020-04-03).
- [5] Wikipedia, 2020. *Metaballs*. https://en.wikipedia.org/wiki/Metaballs (Accessed 2020-04-03).
- [6] J.-W. Chang et al. “Real-time rendering of splashing stream water”. In: Proceedings - 3rd International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIHMSP 2007.Vol. 1. 2007, pp. 337–340. isbn: 0769529941.
- [7] Wikipedia, 2020. *Game engine*. https://en.wikipedia.org/wiki/Game_engine (Accessed 2020-04-03).
- [8] Unity, 2019. *Core Platform*. https://unity.com/products/core-platform (Accessed 2020-04-03).
- [9] Wikipedia, 2020. *Phong reflection model*. https://en.wikipedia.org/wiki/Phong_reflection_model (Accessed 2020-04-03).
- [10] Wikipedia, 2020. *Phong shading*. https://en.wikipedia.org/wiki/Phong_shading (Accessed 2020-04-03).
- [11] Wikipedia, 2020. *Suspension of disbelief*. https://en.wikipedia.org/wiki/Suspension_of_disbelief (Accessed 2020-04-04).
- [12] Scholarly Gamers, 2017. *The Suspension of Disbelief Required by Video Gaming*. https://www.scholarlygamers.com/feature/2017/05/11/suspension-disbelief-required-video-gaming/ (Accessed 2020-04-04).