A GPU-Based Level of Detail System for the Real-Time Simulation and Rendering of Large-Scale Granular Terrain

Craig Leach
University of Cape Town
craigleach007@gmail.com

Patrick Marais
University of Cape Town
patrick@cs.uct.ac.za

ABSTRACT

We describe a system that is able to efficiently render large-scale particle-based granular terrains in real-time. This is achieved by integrating a particle-based granular terrain simulation with a heightfield-based terrain system, effectively creating a level of detail system. By quickly converting areas of terrain from the heightfield-based representation to the particle-based representation around dynamic objects which collide with the terrain, we are able to create the appearance of a large-scale particle-based granular terrain, whilst maintaining real-time frame rates. The system presented is a proof of concept, to show that such a system may be viable for use in real-time applications in the future, but initial results are encouraging.

Keywords

Level of Detail, Terrain, Rendering, Simulation

1 INTRODUCTION

Real-time computer games and simulations often contain large virtual terrain environments. This terrain may consist of various granular materials, such as sand, rubble and rocks. Granular terrain displays many complex interactions, both between the constituent granules, and with objects. Previous approaches to rendering such terrains rely on simple textured geometry, with little to no support for dynamic interactions.

Recently, particle-based granular simulations, such as that of Bell et al.[Bel05] have emerged as an alternative method for simulating volumes of granular materials. These systems simulate granular materials by using particles to represent the individual granules, and exhibit realistic, physically correct interactions with dynamic objects.

Longmore et al.[Lon13] extended the work of Bell et al. with a GPU-based implementation, in order to improve the simulation performance. However, the resulting system remains computationally expensive, and only small volumes of granular material cane be simulated in real time.

In order to overcome this limitation, we extend Longmore's system, by integrating it with a heightfield-based terrain system to create a level of detail system for simulating large-scale granular terrain. The particle-based terrain system is used to represent areas of terrain around dynamic objects, whereas the heightfield-based terrain is used elsewhere. This allows large-scale granular terrain to be simulated in real-time, with physically correct dynamic interactions. This is made possible by a novel system, which allows for terrain to be converted from one representation to the other in real-time, while maintaining changes made to the particle-based system in the heightfield-based system. We use a GPU geometry clip maps implementation for our heightfield-based terrain system, which is very efficient. This frees up GPU resources for the particle-based simulation.

The system presented is an initial attempt to create a system capable of simulating and rendering large-scale particle-based granular terrains. While many issues still remain, the initial results are promising. We show that the system is capable of simulating and rendering multiple particle-based simulations across a large-scale terrain, whilst maintaining real-time performance. In one scenario, 10 high-fidelity simulations are run at the same time, whilst maintaining 30 frames per second. However, the number of active simulations is limited by the computational resources of the GPU. Stuttering during terrain representation conversions has been reduced, but unfortunately, still remains. Additionally, the particle sizes don't allow for sand to be realistically simulated on current GPUs. However, other granular materials may still be simulated.

We make the following contributions: a technique for scaling particle-based terrain simulations to achieve finer grained interactions, and a system which is able to convert between particle-based and heightfield-based terrain representations in real-time.

The remainder of the paper is organised as follows: We begin by exploring related work in the field. We then introduce our system, giving a broad overview of how it works. Each individual contribution is then analysed, before we present our results. Finally we draw conclusions, and present possible future research in this field.

2 RELATED WORK

Terrain forms an important part of virtual environments, and the efficient creation and rendering of terrain has thus been an active area of research for many years. Heightfield based terrains have become popular in modern games and visual effects simulations. However, these terrains display little to no dynamic interactions with dynamic objects. Recently, particle-based granular terrain simulations have emerged as an alternative to these techniques. These systems display realistic interactions, but are computationally expensive.

2.1 Heightfield-Based Terrain

Heightfields represent a terrain as a grid of regularly sampled points, with each point in the grid storing the elevation at the corresponding point on the terrain. Heightfields are often stored as greyscale images, although other formats also exist. This makes them compact and highly portable. Dynamic terrain is easily supported, as one only need alter the values stored in the grid. However, as there is only one height associated with any point on the terrain, complex structures such as overhangs cannot be represented. This may be overcome by adding static geometry at points on the terrain where these structures are required.

In order to render a heightfield, a mesh is created that links each node to its adjacent nodes using triangles. However, for large terrain such a naive approach produces too many triangles. Level of detail (LOD) schemes can be used to overcome this limitation. Such schemes use different representations of an object, with a simplified geometric structure, in order to manage rendering efficiency [Lue02].

GPU geometry clipmaps[Asi05] is a level of detail technique for rendering heightfield-based terrains. The technique represents a terrain using a set of concentric regular grids, or "clipmap levels", of increasing sizes, centred about the viewer. The terrain is then rendered by translating and scaling this grid structure in the vertex shader, displacing the vertices in Y-dimension to correspond with the clipmap heightfield. The technique is capable of rendering large-scale terrains in real-time, with minimal overhead.

The original GPU geometry clipmaps implementation did not provide a method for texturing the resulting terrain. Torchelsen et al.[Tor08] introduced a texturing technique which assigns a texture coordinate to each vertex within the grid. A value is obtained, which defines how many times the texture is repeated between vertices in the grid. This then allows the correct texture coordinate to be inferred for each fragment, allowing texturing of the GPU geometry clipmap based terrain.

2.2 Particle-Based Granular Terrain

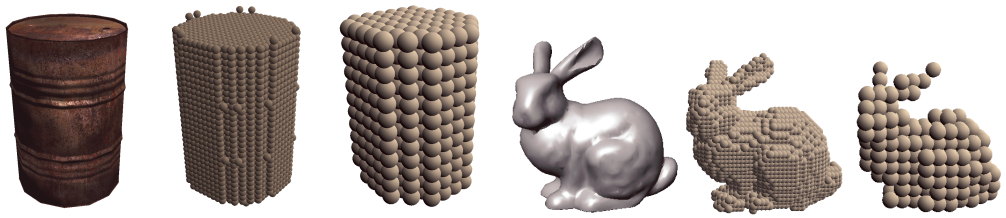
Bell et al.[Bel05] created a system to simulate a volume of granular material. The system uses particles to represent the volume. The particles collectively form a volume, in the same way grains of sand form a pile of sand. Granular materials behave differently to fluids and require a unique set of algorithms to model their characteristics. Granular materials may flow down a slope, like fluid, or form a static volume, like a solid. The system thus uses specialised granular equations to model the particle interactions.

The sand created using this system is very high fidelity, and allows for realistic, physically correct interactions to occur with dynamic objects. The system simulates the natural interactions of sand using rigid bodies, which are made up of groups of four particles, in a tetrahedral structure. Shear, normal and frictional forces are modelled for both collisions between the various particles in the system, and the collisions of particles with dynamic objects. Each individual particle need only check for collisions within its local neighbourhood, and thus exhibits $O(n)$ complexity. Unfortunately, since a large number of particles are required to represent even a small volume of sand, it is infeasible to use such a system to represent a large area of terrain.

Longmore[Lon13] extends this approach to leverage the parallel processing capabilities of modern GPUs. Grids are used to represent the sand particles within the system; one grid stores the positions of the particles, whilst another grid stores their momentums. These grids are used to create a 3D texture, which is passed to the GPU fragment shader, which performs the paricle simulation. The system first calculates the forces applied to each particle, then applies these forces to update the rigid body attributes. The particles are then updated to match the rigid bodies. Finally, the particles are rendered using a splat-based rendering technique.

While more efficient than a conventional CPU-based implementation, the system may still only be used for smaller-scale granular volumes, as the simulation remains computationally expensive. Additionally, due to the size of the 3D texture, memory usage is a major concern, and limits the volume of sand that can be simulated.

In order for the particle system to interact with a model, the model must be converted to a particle-based representation. This is achieved by creating a signed distance field[Sig03] for each model. Computing a signed distance field is expensive, so this conversion is performed



(a) Barrel Model (b) Density of 0.25 (c) Density of 0.5 (d) Stanford Bunny (e) Density of 0.25 (f) Density of 0.5

Figure 1: Two models are converted to a particle-based representation. (a) barrel model. (d) Stanford bunny. The particle-based representation for two different particle scales is presented, corresponding to two differently scaled particle systems.

as a pre-processing step. Once the signed distance field has been generated, particles are created at any negative value in the distance field, which lies adjacent to a positive value (i.e. at each point on the border of the object). The particles together form a single rigid body within the system, in the same way that multiple particles constitute a granule. This creates a surface layer of particles which represent the object. This is sufficient, as even if we added particles throughout the body of the model, collisions with external particles would first occur with these surface particles, therefore preventing the internal particles from colliding with external particles. In fact, by using a single outer layer of particles to represent the objects, both memory and computational resources are saved. Whilst the particle-based representation should interact with the terrain, the model should still be rendered in its original form.

O'Brien et al.[Obr01] introduce a system which allows for a simplified motion model to be used for particle simulation, which effectively creates an LOD system for particle based simulations (or "SLOD", using the papers nomenclature). Under their system, the particle based simulation is subdivided into groups of particles. Each group of particles is treated as a single granule, and the result of the interaction of this granule is applied to each of its constituent particles. However, the particle systems for which it has been implemented are rather simple, and it has not been shown that this system can be extended for use with granular terrain. Furthermore, the speed up from this form of LOD would not be sufficient to allow simulation of a complete terrain. Solenthaler and Gross[Sol11] use two discreet particle resolutions to perform fluid simulations. The coarser resolution simulates the fluid as a whole, whilst the finer resolution is only used in areas where complex interactions occur. Their system produces high quality results, while simultaneously reducing simulation complexity. However, the increase in performance is proportional to the reduction in particle count, and thus, such a system could not be adapted to simulate entire terrains, as even smaller terrains would still require far too many particles to simulate in real-time.

3 SYSTEM OVERVIEW

Our system is composed of three major components: a GPU particle-based granular terrain system, a heightfield-based terrain system, and the terrain manager (Figure 2). The terrain manager lies at the heart of our LOD system. This component is responsible for converting between the two terrain representations, in both directions, and converting models to a particle-based representation, so that they may interact with the terrain. It also holds preinitialised particle-based simulations, so they may be quickly inserted, without worrying about the cost of initialisation.

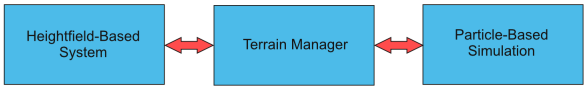


Figure 2: The basic system architecture.

The idea behind our LOD technique is fairly simple; we convert areas of terrain around collisions to the particle-based representation, so that the object may interact with the terrain in a realistic and believable fashion. This creates the illusion of an entire particle-based terrain. Once the object has come to rest, updates for that particular particle-based simulation are disabled, until another collision occurs with it, which helps to limit the overall number of active particle systems. It is still rendered as the particle-based representation though, which creates the illusion that there are more active particle systems than there actually are. Finally, if no further collisions occur with the simulation after a period of two minutes, the area of terrain represented by the particle system is converted back to the heightfield-based representation, preserving any changes made to the terrain in the particle-based simulation. The particle simulation is then returned to the pool of inactive simulations, so that it may be reused for further interactions.

The system is currently not very robust. For instance, if an object is about to leave the current area covered by the current particle simulation, a new particle system must be added for the object to move into. Also, the resolution of the particle system is locked at initialisation.

This means that if the camera moves closer towards a particle system, the particle system will not be refined to adjust for this change. We have chosen to focus on the conversion between the two terrain representations, and have left these issues for future work.

Our heightfield-based terrain system is based directly on the work by Asirvatham and Hoppe[Asi05]. Below, we analyse the other two components of the system: the particle-based terrain simulation, and the conversion process.

4 PARTICLE-BASED GRANULAR TERRAIN

The particle-based terrain system developed by Longmore et al.[Lon13] forms the basis of our particle-based terrain level of detail. In this section we provide a brief description of the system. Additional details may be found in the original paper.

The system simulates the natural interactions of sand using rigid bodies, which are made up of groups of four particles, in a tetrahedral structure. The particles then interact with particles from other rigid bodies, which applies a force to the rigid body. The system produces realistic particle-based terrain, which exhibits physically correct interactions. The particle system leverages the computational power of modern GPUs, while remaining hardware agnostic.

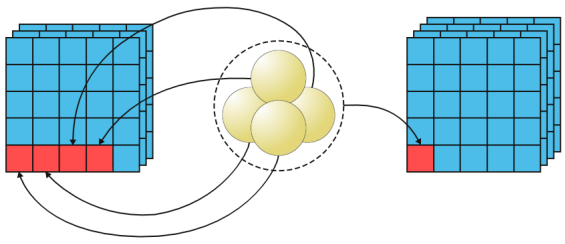


Figure 3: Granules are made up of four particles. The corresponding particle and rigid body attributes are stored in textures[Lon13].

The data required for the particles and rigid bodies is stored in a collection of textures. Textures may be used as a data source, or a data target, and thus present an attractive method to store this information. Additionally, texture caching allows extremely fast access to data which is accessed in spatially local area. Each texture is a four channel floating point texture, and represents a property of the group of particles or rigid bodies. These properties are position, orientation, momentum, and angular momentum for the rigid bodies, and position, momentum, force and offset for the particles.

The particles are linked to the rigid bodies using unique identifiers. Each particle is allocated an identifier, based on its position within the texture. This identifier is stored in the alpha channel of the position texture. The rigid body then stores the identifier of its first constituent particle. As each rigid body is made up of a known number of particles, it is then simple to retrieve the other constituent particles.

4.1 Updates

The system maintains two textures for each property that is stored for the particles/rigid bodies. Each frame, one texture acts as the data source, and one texture acts as the data target. The data is processed by the GPU, and the results are written to the target texture, which then becomes the source texture for the following frame. The advantage here is that because the result is written to a different texture, the source data remains intact, so the result for each particle is based on a constant, non-varying set of data. The system uses the fragment shader to process the updates, which allows the system to remain hardware agnostic, as it does not require any hardware specific third party libraries.

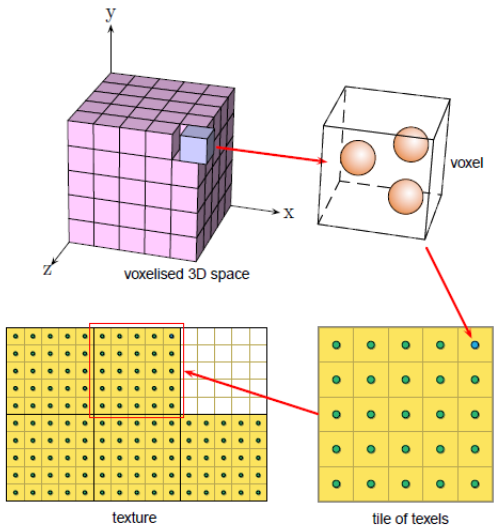


Figure 4: The 3D grid is represented by a 2D texture. Each slice in the 3D grid is stored as a tile within the 2D texture. Each texel represents a voxel within the 3D grid, and can store up to 4 particles (one per colour channel). This diagram, from Longmore et al.[Lon13], shows the layout for a $5 \times 5 \times 5$ voxel area of space.

In order to detect collisions between the various particles in the simulation, a texture representing a 3D grid is used. An example of this texture can be seen in Figure 4. This 3D grid represents the volume in which the particles and rigid bodies exist, effectively discretising the space into a voxelised format. The ID of each particle is added to the grid node which corresponds to it's position in 3-dimensional space. As each texel within the texture is composed of four values, a maximum of four particles may translate to a single grid node. Collisions with adjacent particles are detected by sampling

neighbouring voxels in the grid. This requires 27 texture lookups (i.e. a $3 \times 3 \times 3$ cube). If a particle from another rigid body is found in an adjacent node (including its own grid node), we then perform a collision between the particles. The resultant force of the collision is calculated using the formulas from Bell et al.[Bel05]. Particle systems are sensitive to the time step used. The simulation uses a constant time step of 20 milliseconds.

4.2 Rendering

A splatting technique is employed to render the particles. Splatting is a technique which renders objects using points. This is usually used to render volumetric objects, but has also been adapted to render objects with a large number of vertices[Rus00].

In order to perform the rendering, a vertex buffer object (VBO) is generated. One vertex is added to the VBO for each particle within the simulation. Its position is the texture coordinate which corresponds to the particle within the particle position texture. The vertices are then rendered using the `glDrawArrays` call, with the rendering mode set to `GL_POINTS`. The vertex shader reads the position of the particle from the position texture, and sets the position of the resulting point to that position. The points produced by `GL_POINTS` are square, and thus need to be trimmed by the fragment shader. Any fragments which lie outside of a circle are simply discarded. This method is very efficient, and allows the entire particle system to be rendered with a single call.

However, there are a few downsides to this method. Firstly, the particles each appear spherical, giving the system a uniform look. However, the larger problem for our scenario is that it doesn't integrate well with heightfields. We alpha blend between the two representations, but the difference is still quite obvious. In order to overcome this, a different visualisation is required, which more closely matches the regular structure of heightfields. However, we have chosen to focus on localised particle-based simulation and conversion techniques, and have thus left this as future work.

4.3 Scaling of Particle-Based granular terrain

One of the features of the LOD system is the ability to use coarser particle simulations further away from the observer, and finer grained, more detailed particle simulations closer to the observer. However, Longmore's particle system lacks the ability to scale the particle sizes. Thus, the particle system was adapted to support particles of different sizes. We found that scaling the equations within the updates step tended to introduce instability to the system.

The alternative method we devised is simple and fast. The updates and collisions are processed with particles of the regular size. The renderer then scales the particle sizes and positions, creating the appearance of a larger or smaller particle size. This works well, as the particles are in a stable configuration at the default particle scale. The velocity of the objects need not be scaled, as this is already scaled by the scaling of the position. Only forces external to the terrain must be scaled. The only force that needs to be scaled is thus gravity. This is because the distance covered by a falling object in the virtual world should remain constant, regardless of the scale of the particle system. As gravity results in a constant acceleration, it can be scaled linearly. Additionally, the velocity of objects entering the system should be scaled. This scaled velocity results in appropriately scaled forces when it collides with the terrain, and thus the underlying algorithms can be left untouched. Although smaller particles would usually dictate smaller time steps, in practice we had no problems with instabilities, as the scaling of these velocities did not introduce enough energy into the system to cause any instabilities.

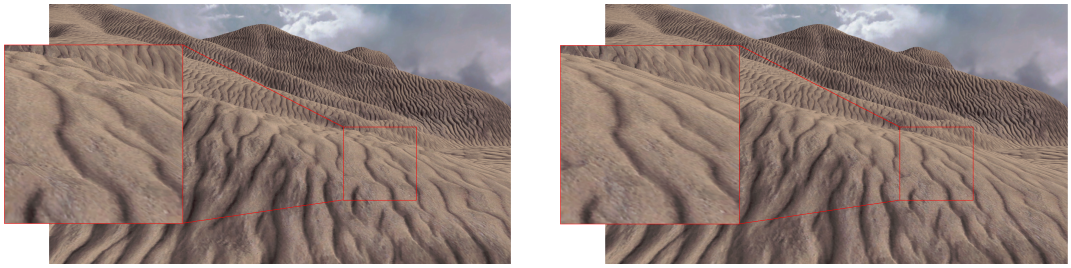
5 CONVERSION

The terrain manager is responsible for switching between the heightfield-based terrain and the particle-based, in both directions, and converting models to a particle-based representation, so that they may interact with the terrain.

5.1 Heightfield to Particle System

In order to convert from a heightfield to the particle-based terrain representation, the system needs to create a volume of particles. The height of the volume at each point on the x-z plane must correspond to the height stored in the heightfield. Thus, the first step in the conversion is dividing up the area covered by the particle system into a grid. The size of each unit in the grid corresponds to the particle system scale. The heightfield is then sampled at each point on this grid. Bilinear interpolation[Len12] is used to infer to the height at points which fall between texels in the heightfield. This height is used to deduce the number of rigid bodies which must be stacked at that point in order to reach that height on the terrain.

Arrays are created to store the rigid body and particle attributes. The grid is iterated over, and at each grid point rigid bodies and their corresponding particles are inserted into the arrays, until the height of the terrain at that point in the grid is reached. Each rigid body and particle is assigned an index, leaving us with the required arrays of particle and rigid body attributes, which are ready to be inserted into the particle system. The topmost particle in each stack is assigned a random orientation, which helps create a more natural looking surface.



(a) (b)

Figure 5: An example terrain with a converted particle system inserted. The inset shows a zoomed in view. (a) Shows the result of the injected heightfield, with no filtering applied. This results in an obviously bumpy surface. (b) Shows the same section, but with Gaussian filtering applied. No discernible discontinuities are present.

These attribute arrays are passed to an inactive particle simulation. However, uploading all this data to the GPU at once results in the CPU – GPU bus becoming bottlenecked, resulting in a noticeable stuttering effect. Instead, the data is uploaded over multiple frames, which significantly reduces the stuttering.

The particle system is then left to settle. Once injected into the system, the rigid bodies shift from their initial positions. This appears unnatural, as the particle system should be representing terrain at rest. Introducing a settling period effectively negates this issue. However, depending on the terrain, particles may continue to move after this settling period, thanks to the "cascading" property of sand. Additionally, in some cases, such as objects moving quickly across the terrain, we cannot afford to wait for the system to settle first before displaying the terrain. Both of these issues could potentially be solved by precalculating various stable particle configurations, and combining them to match the outline of the terrain.

Once the particle system has settled, the particle-based terrain manager alpha blends in the particle system over the course of one second. However, rendering order is an issue. If the terrain is rendered first, particles which lie beneath the level of the terrain will be discarded, yet they will be visible once the blending is complete, which will lead to a noticeable popping effect. Additionally, particles which intersect with the terrain will only be partially rendered, resulting in unsightly artefacts. If the particle system is rendered first, then the terrain will not be rendered behind the particles, and the particles will appear to pop in, and blend between black and their resulting colour. We use the following solution. First, the terrain is rendered as a first pass. The depth buffer is then cleared, and the particle system is rendered. The particles will thus be correctly blended with the underlying terrain. However, particles which should not be visible will be rendered. To address this, the terrain is rendered again, thus occluding particles which should be occluded.

5.2 Conversion from Particle System to Heightfield

In order to convert the terrain from the particle-based representation to the heightfield-based representation, a method is required which is capable of quickly converting the volume of particles to a heightfield. Surface extraction from a set of points is a difficult problem, and many techniques have been developed to solve it, such as Gumhold et al.[Gum01], and Rosenthal et al.[Ros08]. However, these techniques typically take a few seconds, to minutes to complete, and thus are not suitable for real-time applications.

We note that a useful feature of the particle system is that while dynamic updates are expensive, the rendering of the system is comparatively cheap. Also, the particles form a single volume of sand. Based on these two observations, we have developed a novel rendering based technique to perform this conversion. By performing a top-down orthographic projection of the particle system, and extracting the depth buffer from the resulting image, a depth map of the terrain is obtained. As the position of the camera above the terrain is known, it is fairly simple to convert this resulting depth map into a heightmap. As the z-buffer sampling for an orthographic projection is linear, the following formula is used to convert the depth value to a height value:

$$height = y_{camera} - n - z(f - n)$$

where y_{camera} is the height of the camera, n is the distance to the near plane, f is the distance to the far plane and z is the depth value.

Performing this rendering step with a resolution equal to the size of the particle system produced poor results. Instead, the system renders the particle system using a resolution of 1024×1024. This resolution is much higher than the number of visible particles, with each particle covering multiple pixels. The depth map is sampled at points which correspond to points in the heightmap.

Although this produces better results, a stepping effect may occur, since, depending on the positions of the

particles, the same particle may be sampled multiple times. This is corrected by applying a 3×3 Gaussian filter[Sha01], to smooth out sharp transitions (Figure 5). A larger filter kernel size results in over-smoothing, and thus the loss of subtle changes made to the terrain in the particle system.

The resultant heightmap is inserted into the terrains heightfield, over-writing the section of the heightfield which corresponds to the particle system. This causes any changes made to the terrain in the particle-based system to persist in the heightfield-based system. However, there may be a discontinuity along the edges of the newly injected heightmap. This is due to a sudden transition from the original heightfield, to the heightfield representing the particle system, which may lie slightly below or above the level of the terrain at that point. Thus, a Gaussian filter with a 3×3 kernel is also applied along the border of the inserted heightfield.

6 RESULTS

The system was implemented in C++, using the Microsoft Visual Studio 2010 IDE on Windows 7 (SP1) and employs OpenGL 2.1, with GLEW 1.10.0. All results are generated at a resolution of 1920x1080. The hardware testing platform consists of an Intel Core i7 930 processor with 6GB of DDR3 memory. Results are presented for both an Nvidia Geforce GTX 460 with 1024 MB RAM, and an Nvidia Geforce GTX 770 with 2048 MB RAM. These two cards where chosen to represent the performance of mid- and high-end cards. The Nvidia display driver 331.58 was used for both cards.

6.1 GPU Geometry Clipmaps

In order to evaluate the performance of the GPU Geometry clipmaps implementation, we tested the performance across a range of different terrains. Ten different terrains are used in total, and the results are averaged, to provide a reasonable performance measure against which to evaluate system performance. Each terrain was run five times. Two different terrain sizes are used: $512 \times 512(6)$ and $1024 \times 1024(4)$. The view of the terrain is from the corner, and spans the entire terrain. The 512×512 terrains produce 79,886 triangles, whereas the 1024×1024 terrains produce 103,700 triangles. The size of the terrain has a fairly minimal impact on performance, as only a single additional clipmap level is required to render the extra terrain section in the larger terrain, due to the exponential increase in the grid resolution.

The systems performs very well (Table 1), with both the midrange and the high-end graphics cards. Such high performance from the heightfield-based component of the system is necessary, since the particle-based simulation is computationally expensive (see Section 6.3). Finally, the memory usage of the system is measured.

	GTX 460	GTX 770
512×512		
Average (FPS)	393	949
Std Dev (FPS)	23	72
Shadowed Average (FPS)	169	484
Shadowed Std Dev (FPS)	23	36
1024×1024		
Average (FPS)	315	776
Std Dev (FPS)	12	28
Shadowed Average (FPS)	129	382
Shadowed Std Dev (FPS)	7	17

Table 1: GPU geometry clipmap performance results for the Nvidia Geforce GTX 460 and GTX 770.

The system uses a total of 30MB of main memory and 40MB of graphics memory. A low memory footprint is important, as it allows for other high quality assets to be used in the game or simulation.

6.2 Particle-based Simulation

The performance of the particle-based terrain simulation is of greatest importance, as the physical correctness of the system has already been established[Lon13]. To test the particle-based system, particles are injected into a 3D cuboid. Particles are injected in sets of 40,000, i.e. 10,000 granules at a time. The framerate is measured for each set of injected particles.

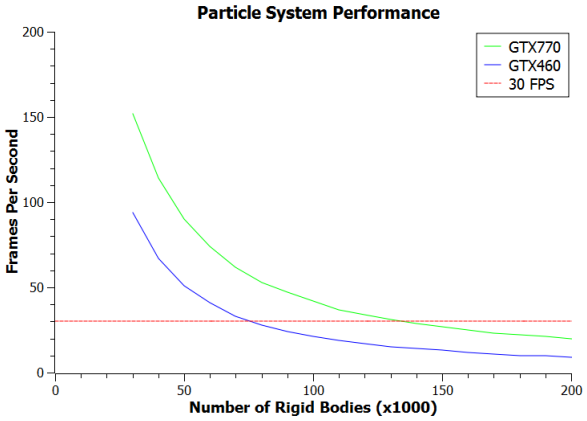


Figure 6: Performance results of the particle-based simulation. Note that each rigid body, or granule, is made up of four particles in a tetrahedral configuration.

The system scales fairly well with the computational power of the graphics card used, with the more powerful GTX 770 achieving almost double the performance of the GTX 460. The framerate is inversely proportional to the number of particles in the system.

The particle-based simulation is comprised of two primary components: the update component, and the rendering component. In order to measure the performance of these individual components, we measure the frame time of the entire system. Then, we disable the simulation, and simply render the system, while recording the

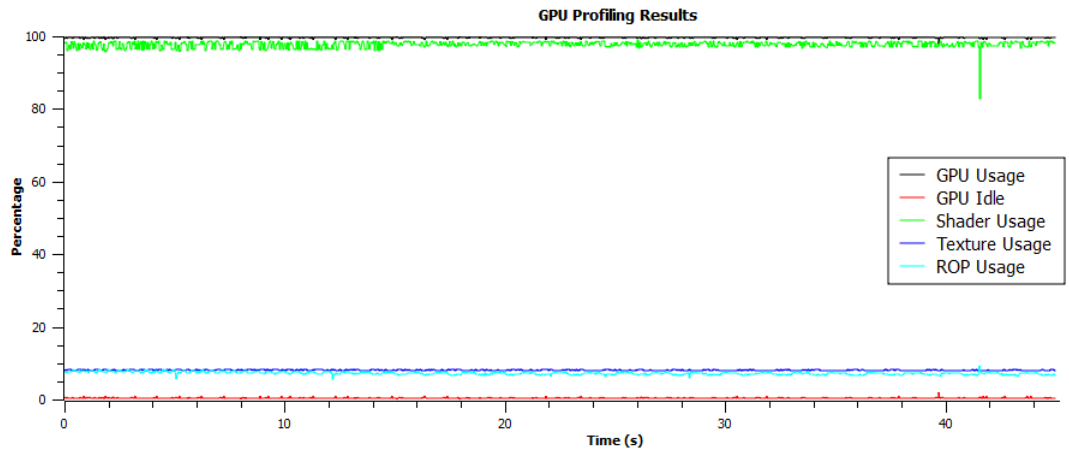


Figure 7: The GPU profiling results for the GTX 770, for a particle system with 100,000 granules, over 45 seconds.

frame time. The difference between these two frame times corresponds to the time taken to process the updates. The result is shown in Figure 8.

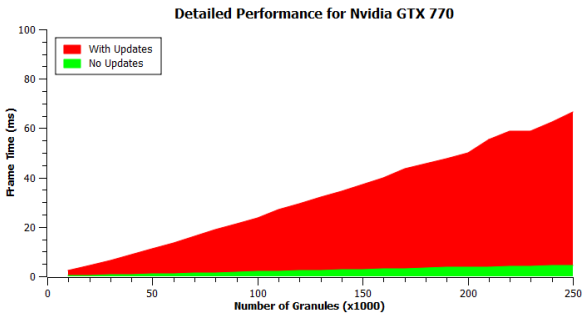


Figure 8: Frame time of the particle-based simulation components, for the GTX 770. The time spent to process updates is indicated by the red area, whereas the time spent on rendering is shown in green.

Clearly, the updates are the major limiting factor behind system performance. This is to be expected, as the updates entail mapping the particles to a 3D grid, then performing collisions for hundreds of thousands of particles, and updating the rigid bodies and particles in response to these collisions. The frame time, both with and without updates, is linearly proportional to the number of particles, which accords with the previous performance results.

We used Nvidia Perfkit 3.1.0.13233 to measure the GPU performance counters, in order to identify any potential bottlenecks. As can be seen from the results shown in Figure 7, we can see that shader usage is extremely high, and is the primary bottleneck behind GPU performance. The next closest performance metric, texture usage, peaks at less than 10%, which means that system performance should scale well with future increases in GPU shader performance.

The system used 140 MB of main memory and 188 MB of GPU memory. This memory usage is mainly due to the number of textures required to store all the particle and rigid body attributes. Fortunately, with the amount of memory available in most modern computers and graphics cards, this memory usage is perfectly acceptable, and allows us to use multiple particle-simulations concurrently, along with other graphical assets. Note, that the number of particles added to the system does not affect the memory usage. Textures of a set size are used to hold the particle and granule attributes. These textures are allocated during the system set up. Thus the memory usage is independent of the particle count.

6.3 Integrated System

Three scenarios were developed in order to test the various facets of the LOD framework.

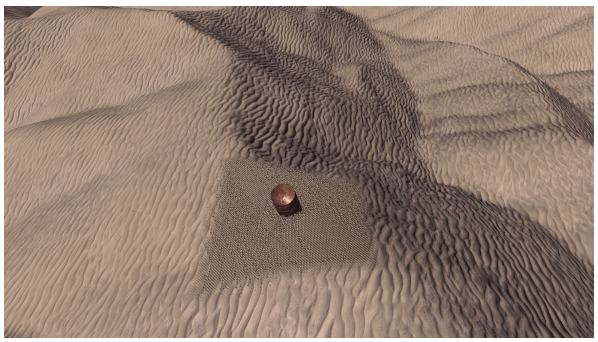


Figure 9: Screen shot of the first test scene. The particle simulation can be seen interacting with a model.

The first scenario contains one very high quality particle system. This is a stress test, to test the systems ability to render and simulate a large particle system, while also rendering the heightfield-based terrain, all the while maintaining real-time performance. The particle system for this scenario consists of 115,502 granules (i.e. 462,008 particles).

In addition, a barrel model is dropped onto the particle simulation. This model is made up of 6,261 particles. This is done to showcase the realistic interactions, and

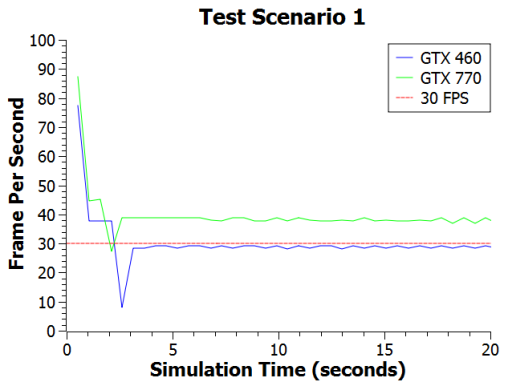


Figure 10: Performance results from the first test scenario.

puts further stress on the system. The performance results for this scenario are shown in Figure 10.

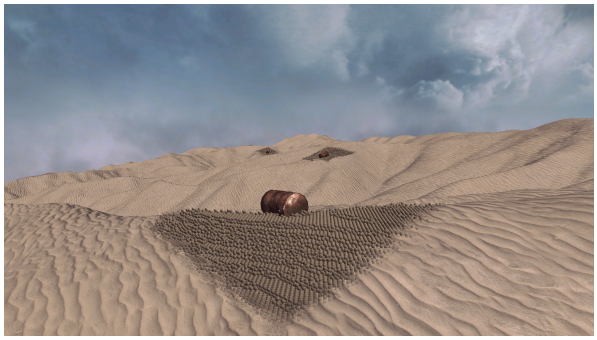


Figure 11: Screen shot of the second test scene. Three simulations can be seen, each with a different scale.

The second scenario contains three particle systems, with three different particle scales. Each scale represents a different distance from the camera. The aim of this test is to verify that multiple particle simulations, of different scales, may be used concurrently, whilst maintaining real-time performance. This scenario represents the general envisaged use case, with a few simulations taking place at different distances from the user.

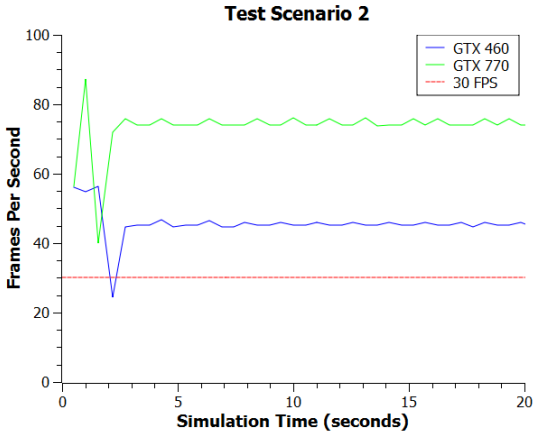


Figure 12: Performance results from the second test scenario.

Each particle system in the scenario has a scale appropriate for the distance from the camera. The foreground simulation contains 202,352 particles, the midrange particle simulation contains 54,876 particles, and the far particle simulation contains 25,776 particles. The performance results for this scenario are shown in Figure 12.

The final test consists of ten particle simulations. The dynamic object for each particle system is added 10 seconds after the last. This shows off the ability of the system to handle many particle simulations, by disabling updates once bodies have come to rest. Each simulation is fairly complex, and contains between 55,000 to 75,000 rigid bodies (220,000 particles to 300,000 particles). A particle scale of 0.33 is used for each simulation. The results are shown in Figure 14.

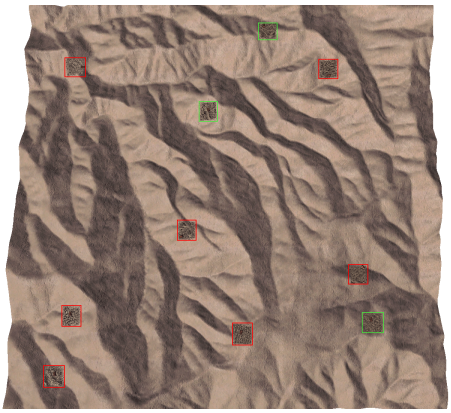


Figure 13: Screen shot of the third test scene. Ten particle simulations can be seen. Active systems are outlined in green, and systems at rest are outlined in red.

While both cards generally maintain real-time frame rates, the GTX 460 falters in scenario 3. Closer investigation showed that due to the slower updates the particle system models took a longer time to come to rest. By this time another particle system had been populated, thus competing for computational resources. This in turn caused the updates for both simulations to slow down, and when another simulation was added, the effect compounded, leading to a cascading drop in performance. The GTX 770 on the other hand, is powerful enough to ensure that this doesn't occur. This suggests that for mid-range and lower level graphics cards, the particle systems used should not be too finely scaled, in order to allow the dynamic objects to come to rest quickly. Additionally, the number of concurrently active particle simulations may also need to be limited.

While the stuttering effect has been reduced somewhat by inserting particle data over multiple frames, it has not been completely eliminated. This can be seen by the downward spike after particle insertion in all the test scenarios. However, the framerate stays high enough that it is not too noticeable.

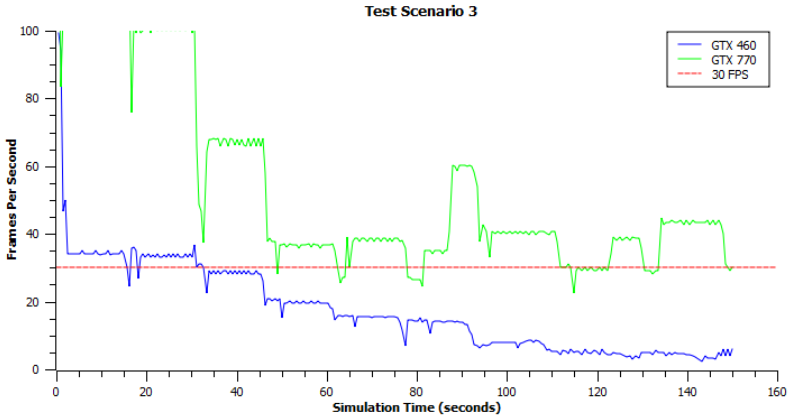


Figure 14: Performance results for the third test scenario.

7 CONCLUSION

In this paper, we have presented our GPU-based LOD technique for large-scale particle-based granular terrains. The resulting system can simulate large-scale granular terrain in real-time, by seamlessly switching between the heightfield-based and particle-based representations. Changes to the terrain in the particle-based simulation persist in the heightfield-based system, by virtue of a novel rendering-based conversion technique.

Performance results are promising with both test GPUs maintaining real-time performance, except for in one of the three test scenarios, where the mid-range GPU produced poor results. Future advances in GPU processing power will allow for even mid-range graphics cards to use the system to simulate large-scale granular terrains, even under the tough conditions of this test scenario.

However, it is important to remember that this paper represents a proof of concept, and simply explores the potential feasibility of such systems. Whilst this paper addresses some of the issues with rendering large-scale granular terrain in real-time, many issues remain, and thus there are many possible avenues for future research in this area. For instance, the rendering technique should be updated to a system which more closely matches the regular grid structure of heightfields. The heightfield to particle system conversion could be optimised with pregenerated particle configurations to stabilise, and allow almost instantaneous conversions, and particle systems could be refined to finer resolutions as the camera approaches them.

8 REFERENCES

[Asi05] Asirvatham, A., and Hoppe, H. Terrain rendering using GPU-based geometry clipmaps. In GPU Gems 2, pp.27–45, 2005.

[Bel05] Bell, N., Yu, Y., and Mucha, P.J. Particle-based simulation of granular materials In Proc. of the ACM SIGGRAPH/Eurographics symposium on computer animation, pp.77–86, 2005.

[Gum01] Gumhold, S., Wang, X., and Macleod, R. Feature extraction from point clouds In Proc. of the 10th international meshing roundtable, pp.293–305, 2001.

[Len12] Lengyel, E. Mathematics for 3D game programming and computer graphics, pp.171–172 2012.

[Lon13] Longmore, J.P., Marais, P., and Kuttel, M. Towards realistic and interactive sand simulation: A GPU-based framework Powder Technology 235, pp.983–1000, 2013.

[Lue02] Luebke, D., Watson, B., Cohen, J.D., Reddy, M., and Varshney, A. Level of detail for 3D graphics, 2002.

[Obr01] O'Brien, D., Fisher, S., and Lin, M.C. Automatic simplification of particle system dynamics, In Computer Animation, pp.210–218, 2001.

[Ros08] Rosenthal, P., and Linsen, L. Smooth surface extraction from unstructured point-based volume data using PDEs IEEE transactions on visualization and computer graphics, pp.1531–1546, 2008.

[Rus00] Rusinkiewicz, S., and Levoy, M. QSplat: A multiresolution point rendering system for large meshes, Proceedings of the 27th annual conference on computer graphics and interactive techniques, pp.343–352, 2000.

[Sol11] Solenthaler, B., and Gross, M. Two-scale particle simulation, ACM transactions on graphics, pp.81, 2011.

[Sig03] Sigg, C., Peikert, R., and Gross, M. Signed distance transform using graphics hardware In IEEE visualization, pp.83–90, 2003.

[Sha01] Shapiro L.G., and Stockman, G.C. Computer Vision, pp.137–150, 2001.

[Tor08] Torchelsen, R., Comba, J., and Bastos, R. Practical geometry clipmaps for rendering terrains in computer games In Shader X6 – Advanced Rendering Techniques, pp.103–114, 2008.