Dynamic terrains applying pseudo-infinity and synthesized by Bezier curves

Rodrigo de Toni* Fernando Marson Vinicius J. Cassol

Universidade do vale do Rio dos Sinos(UNISINOS), Escola Politécnica, Brazil

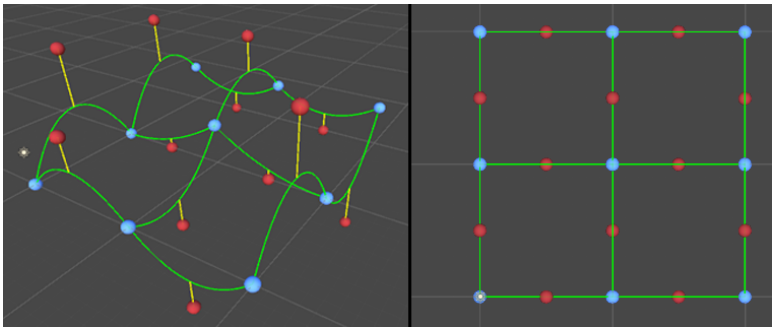

Figure 1: Representation of Bezier curves as a terrain

ABSTRACT

This paper describes the development of a procedural pseudo-infinite deformable terrain system. The proposed method intends to deal with the high performance required by games. In this work the terrains are synthesised using Bezier curves and are loaded and saved automatically as needed, that is according to the camera position. The Heightmap's generating process is, at some moments of a game, an expensive task. Considering such aspect we perform the Heightmap's synthesis in the background of the application, restricting it to a specific FPS target. Moreover, we developed a deformation model which allows alterations in the HeightMap and Texture. We have used texture arrays at the GPU level to prevent sending unnecessary data. The performance of the system is measured both in the average FPS during the deformation of the terrain and the time taken for the synthesis

Keywords: Real-time dynamic terrain, synthesis, procedural.

1 INTRODUCTION

Realism is a key feature of many modern games. Realistic graphics and realistic physics are ways of immersing the player into the game, another part of this realistic environment is interactivity, the environment has to change according to the player's actions. Some of this interactions are easily done, such as opening a door but changes that require a fundamental alterations to the game environment are more complex (such as the explosion from a grenade deforming the ground the player walks on).Often developers resort to simplified versions of this change, such as swapping predetermined geometries or simply painting a texture on the location.

In this work is presented a framework for a middle-ware to create deformable terrains that using inheritance allows developers to create any type of terrain deformation, and using markers a developer can define how a terrain will deform on specific regions of the terrain. For example defining specifically how a sand region of the terrain will be deformed.

Furthermore, the terrain and deformation model are integrated with pseudo-infinity. Maintaining seamless consistency in an ever expanding environment, and generating this terrains in a fast manner so that it does not interfere with game-play.

The remaining of this paper is organized as the following: 2)Basic Concepts: A explanation of some of the basic concepts involved in this paper, 3) Related Work: A brief overview of the scientific literature surrounding this paper, 4)The Proposed Model: Presents in detail the model being proposed, 5) Results: Shows the results obtained using the proposed model and 6)Conclusion and Future Work: Gives our final thoughts and possible continuations on this line of work.

2 BASIC CONCEPTS

Heightmap is the primary concept behind most terrain systems[7, 10], it is a greyscale texture where each pixel represents the the heights of the vertices's within a 3D mesh-grid. For each vertex of a mesh there is a pixel that alters the vertex's position along the y axis based on how close to black or white the pixel is.

Bezier curves are commonly used in computer graphics to create smooth surfaces or paths, this are parametric curves taking control points as input and averaging its position based on a value t, where t is a value of 0 to 1, 0 being the starting point of the curve and 1 being the ending[8].

3 RELATED WORK

Many terrain systems seek to simulate infinite worlds large environments[10, 2]. A truly infinite terrain system is impossible[6] so this systems limit themselves to pseudo-infinity that being the system will create new terrains as needed, and load the same terrain at the same location to maintain consistency.

*e-mail: rodrigodtoni@gmail.com

Some terrain systems seek to simulate deformations, this terrains are called dynamic terrains. In general there are two categories of dynamic terrains: physics based and appearance based. Physics based dynamic terrains simulate surface deformation on Soil Mechanics. Over the years several articles have been published describing the development of a physics based approach to dynamic terrains[5], however the cost of this type of realistic simulation is the high computational cost involved in using such complex physics based algorithms. Appearance based dynamic terrains disregard physics based calculations and try to create deformations that look realistic, this is generally done using parameters that modify how pre-determined behavior will occur[1],typically this approach obtains greater performance than physics based methods but is more limited on the number of different soils that it can simulate due its simplified approach.

Another common feature of terrains in the scientific literature is the procedural generation of its contents. There has been several researches that developed techniques to synthesize Heightmaps, such as Saunders[9], he proposes a genetic algorithm approach that uses both real-world sample data in combination with user input, the user gives a seed of what type of Heightmap they want(in the form of a 2D map of polygonal regions), and the algorithm selects from public available data which DEMs fit the user requested Heightmap. Another method proposed by Ariyan [2] utilizes a web of curves to synthesize mountains with the center point of the web being the peak, the user can define the peak and the algorithm simulates the rocky formations of a typical mountain.

3.1 Contextualization

This paper focuses primarily on real-time modification to the height-map and the terrain's textures using a appearance based approach.The main contributions from this paper being: 1)A structured middle-ware that allows for the use of textures as markers to alter how collisions and deformations will change on specific areas of the terrain, 2)The deformation of terrains in a pseudo-infinite environment, requiring seamless deformation in several terrains and 3)A grid of Bezier curve approach to synthesizing terrains.

The objective in this work is to create a environment that allows for any type of deformation to be developed, with the main focus being towards video game. Since the focus in this paper is on real-time processing and games a strong focus is placed on GPU and Shader processing.

4 Proposed Model

The model proposed in this work uses pseudo-infinite terrains in combination with appearance based deformation techniques. All the terrains are synthesised using a curve based approach [2], more specifically quadratic Bezier curves. The further sub-sections are divided as follows: 4.1) Performance manager: This sub-section explains our model for dividing processing over time so more computationally expensive functions do not interfere with game-play, 4.2) Terrain creation: explains how the Heightmap and texture are synthesized, 4.3) Deformation Model: elaborates on how the terrain is altered on collision or by command and 4.4) Pseudo Infinite: this final section explains how both prior sections are put together in a ever expanding environment.

4.1 Performance manager

Several processes are too expensive to be executed in its entirety during a game, that is without creating a noticeable FPS spike, processes like synthesizing a terrain or saving this terrain's information into the hard-drive. So in our model such features are divided and executed over a longer period of time, rather than executing it all at once.

The performance manager is controlled by a predetermined amount of time it is allowed to occupy in a single frame, that is the class has a list of delegated functions to execute, when it executes a function the time it took to perform its task is added to a timer, when the timer is equal or greater than the predetermined time the manager waits for the next frame to continue to execute its delegates. The form this delegated functions can vary, for example to synthesize the terrain we execute a function SetHeight(int x,int y) to generate a single pixel of the terrain's Heightmap, so when passing the SetHeight function to the manager we declare it to be executed from x=0 to x=256 and the same for y.

4.2 Terrain creation

The terrain is synthesized using 3 point Bezier curves organized in grid network, much like a Heightmap, the control points are equally distributed along the X and Z axis with variation only on the Y, where the P1(the first control point) is equal to P3(the last control point) of the curve before it on the relevant axis (x or z), see figure 1. To have a smooth and seamless transition between one curve to another, the middle control point (P2) has only to be in the line created from the previous curve's P2 and its curves P1. Than P1 and P3 are the only ones that are randomized, in this project we used pseudo-randomization i.e each control point receives only a small variation from the last one, with the exception of the first one.

So our model only randomizes the odd numbers control points, with P2 being always smooth. The randomization process is pseudo-random, that is, apart from the very first control point, every control point is a small random increment from the previous control point. So after the first control point(P1) of the first curve(C1) is randomized from 0 to 1, its final control point receives the Y position of P1 plus a small random value from -r to r(where r is a number smaller than 0.1).

With a synthesis based around curves connected in a grid format, pseudo-infinity is a simple matter of connecting more curves to grid (they do however must maintain a consistent size, a curve grid of 16x16 can only connect with another curve grid of 16x16). After the generating the terrain's Heightmap the terrain's texture is generated based on its results, that is according to the height of a vertex the pixels surrounding it will be of one terrain's TerrainTexture's tile or another. The outcome here is based on developer input, the model allows so that a texture has a range of height, so a texture of rock may occupy the height space of 200 to 500.

4.3 Deformation model

Each vertex on the terrain mesh has several weights from 0 to 1 that represent its connection to a TerrainTexture, we will call this values VT, so a vertex might have VT1= 0.5, VT2=0.25, VT3=0 and VT4=0.25 (all the values put together must add up to 1), each one representing its association with a different TerrainTexture. VT is determined after the terrain creation and is based around the pixels surrounding the vertex.

The VTs will determine how the vertex will be deformed. Every TerrainTexture added to the terrain has a series of functions that defines how the deformation will occur, the results are altered multiplied by the VTs associated with that TerrainTexture and added up before altering the terrain.Than the deformation process is defined by two functions from the TerrainTexture: 1)SetHeight: receives the vertex and deformation info and returns a float. 2) SetPixel: returns a color, receives the deformation info and the pixel information as it relates to the vertex nearby i.e if there are 32x32 pixels within a square in the mesh grid the function will receive its position within this 32x32 coordinates.

Each one of this functions are virtual, meaning they will be overwritten by inheritance, so a developer can create as many types of deformations as it pleases them on top of this class. Each TerrainTexture object can an instance of the same class or several different ones. They receive as a parameter a object called ImpactInfo which contains the following information about the collision: 1)Position:

a 2D vector containing the X and Y indexes of the mesh grid or the texture. 2)Strength: The magnitude of vector that caused the collision. 3) Area: A integer area in vertexes or in pixels of that is being altered in this impact, equal for width and height. 4) Impact Position: a 2D vector containing the X and Y indexes of the mesh grid or the texture, so when Impact Position is 0,0 it is the bottom left corner of the area affected by the impact. 5) Current Value: the current pixel of value of the texture.

Outside of the process of altering the vertexes and texture some other aspects of the terrain are relevant to the Deformation Model, mainly the collision testing and how to treat it in a constantly altering environment and the rendering process for a terrain with constantly changing mesh and texture, hence the following remaining of this subsection is divided to deal with Collision and Rendering respectively.

4.3.1 Collision

With the terrain's shape being constantly altered it's collider also must be, however constantly recalculating a mesh collider and for a large terrain is expensive. In order prevent this cost we only calculate colliders where they may be relevant, that is whenever a object (that may potentially collide with the terrain) is found within the terrain's bounding box, using fast indexation a collider is generated for the area directly below this object, the area that this collider occupies depends on the bounding box of the object.

4.3.2 Rendering

The terrain's mesh is a grid of 254 by 254 and it is rendered using shaders, in the shader the mesh is adapted to a Heightmap of 256 by 256(the mesh is limited to 254 to keep its total number of vertexes within the range of a unsigned short integer, and the Heightmap is kept with its dimensions at 256 to maintain it under the potency of 2 which prevents unnecessary copying for the GPU). Still in the shader the normals are calculated based on the vertexes adapted to the Heightmap.

Since the proposed model presented in this work also seeks to paint the terrain's texture on real time it cannot send large textures to the GPU, transferring a a 2048 by 2048 texture to the GPU can be a computationally expensive task and a unnecessary one, if a collision occurs and it paints a single pixel it is unnecessary to send the entire texture again to the GPU. In this model, in order to prevent this unnecessary cost, we divide the terrain's texture into 16 smaller texture each representing a 4 by 4 division of the original texture, so since the terrain would use a 2048 by 2048 texture instead now it uses 16, this textures are placed in a 4 by 4 grid which are than accessed in using fast indexation both in the GPU and in the process of painting it in the CPU.

4.4 Pseudo-Infinite

Typically pseudo-infinite terrain systems use several small terrains that surround the rendering camera [3], this terrains will constantly re-position themselves and load(or if needed generate) Heightmaps and textures as the camera moves, hence simulating a infinite world. As good practice it is wise to keep this loading and generating process invisible for the user (seeing it occur may break the illusion of infinity) so while terrains are going trough this process they should be invisible.

Using fast indexation we can map regions to a single terrain that uses a grid like global positioning system, this is based on a consistent pattern of width and height for all the terrains. So if a camera is positioned in a vector (x,z) it is positioned in the terrain (x/width, z/length). When a deformation occurs at the border of a terrain instance, if the area of the deformation would overflow this terrain, the Pseudo-Infinite system managing the terrains will apply the remaining area of deformation in the appropriate neighboring terrain.If the deformation occurs and the neighbour is not yet synthesized or even existent, the pseudo-infinite manager holds on to that information so that is applied later.

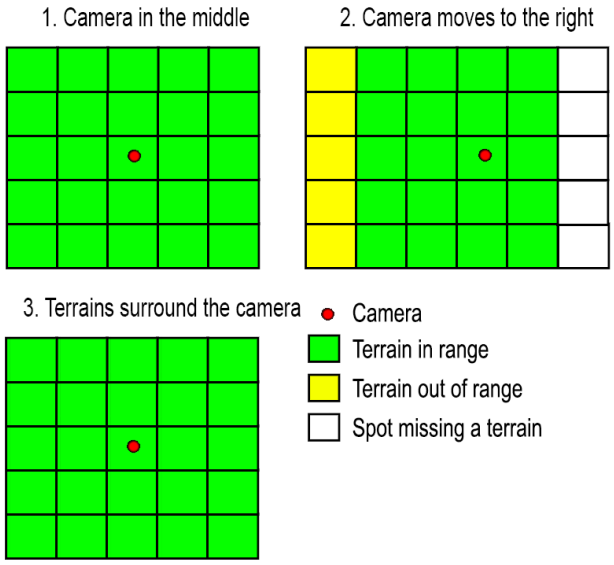

Figure 2: Repositioning Terrains as camera moves

Working with a 5 by 5 grid of terrains surrounding the camera, where 0,0 is the bottom left, 4,4 is the top right and 2,2 is where the camera is localized, in order to prevent unnecessary loading and synthesizing processes, only the terrains that are outside of the area of the camera go trough this, see figure 2. When the pseudo-infinite system sees that a terrain is out of range of the camera and it will be used elsewhere, the information of the the terrain is stored in a file, a file unique to the terrain's global positioning.All the terrains that are on the borders of the grid (i.e the terrains that have x or y =0 or equal to 4) are always invisible and non-interactable by anyone other than the Pseudo Infinite system, this is because rendering such terrains is unnecessary as they are barely visible, being so distant, and they are often the ones going trough the synthesis process .

5 Results

In order the to validate our model we developed an application in Unity3D [1], in the game engine's version 5.4.1b. Due to several performance issues with systems that use Unity's native terrain system, we opted to develop our own system. Such tests involve a number of collisions and the area of this collisions(measured in equal dimensions of a square, so a collision with area 5 altered a square of 5pixels by 5 pixels) in varying amounts.

We aim here to get the minimum of 120 FPS, as previously established by Crause as a standard for terrain deformation [4]. This value is set as the bare minimum for video game development because most games aim for 60 FPS, and they also will involve a lot more than just the terrain system, so some breathing room is necessary.

Figure 3 specifies the tests and it's results.Each tests are executed during a period of 1 minute and the resulting FPS is the average FPS measurement over this 1 minute, the column Objects describes the number of objects colliding and causing destruction simultaneously, and area describes the fixed area of impact for all of this deformations.

[1] www.unity3d.com

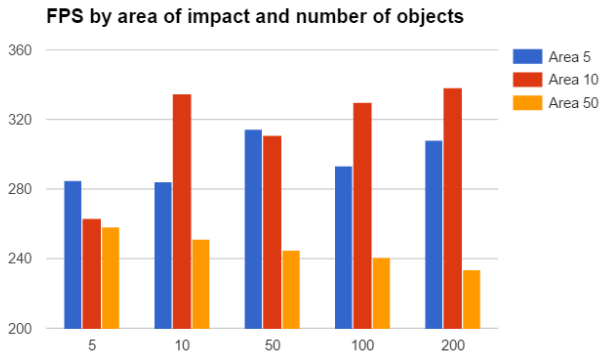

Figure 3: Deformation measurements

Outside of testing the performance of the deformation, the process of creating the terrain is measured, figure 4 describes the set of tests and their results that measure this aspect of our application. The Target FPS is the parameter set in our Performance Manager (see section 4.1) so that the synthesis is not allowed to occupy more than this time, dimensions is the width and height of the terrain's Heightmap, the results are measured in seconds.All the synthesis used 4 TerrainTextures and a texture of 1024 by 1024.

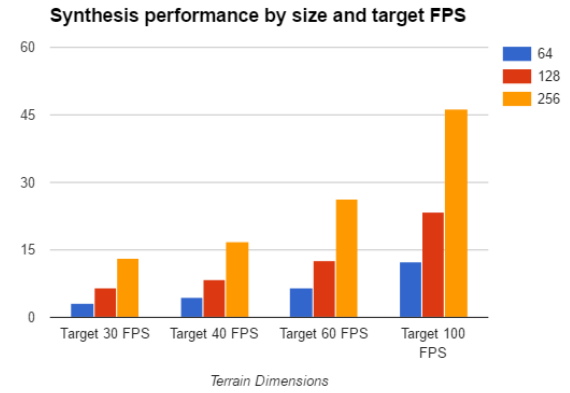

Figure 4: Synthesis measurements

Both of this set of tests were performed in a Dell computer, with Intel(R) Core(TM) i5-4200-1.600 GHZ, a GeForce GT 740M and 6GB of RAM.

As seen on Figure 3 the system surpasses the minimum FPS even in its most rigorous stress test, maintaining 233.35 Frames Per Second. In both columns of area of 5 and 10, the system barely affected the performance,overall in most situations maintained a 270 to 330 FPS variation.As expected, we can see on Figure 4 that the time consumed synthesizing the terrain is proportional to its limitation and terrain scale. A terrain with 64 by 64 synthesizing with 60 FPS limitation takes twice as long to be created as a terrain with 64 by 64 with a 30 FPS limitation, and the same happens with the terrain's dimensions.

6 Conclusion and Future Work

We have presented a model for pseudo-infinite deformable terrain, being synthesized by Bezier curves. Using curves to synthesize the terrain, pseudo-infinity becomes a simple matter adding more curves to the network.Futurely more work should be done to improve the performance of this process, and combine it with other filters effects to give better visual quality to the Heightmap.

By reducing unnecessary communication with the GPU, our model can take advantage of its processing power to render the terrains, while allowing for it to be altered in real-time, without creating a bottle-neck. Further work is needed on integrating this grid based approach with the Heightmap aspect of the pseudo-infinite terrain, perhaps than only a single larger terrain with several Heightmaps is enough to perform a pseudo-infinite system.That is in Figure 2, one single terrain would occupy all the slots in the grid, or perhaps a combination of the two approaches.

References

[1] A. S. Aquilio, J. C. Brooks, Y. Zhu, and G. S. Owen. Real-time gpu-based simulation of dynamic terrain. In *Proceedings of the Second International Conference on Advances in Visual Computing - Volume Part I*, ISVC'06, pages 891–900, Berlin, Heidelberg, 2006. Springer-Verlag.

[2] M. Ariyan and D. Mould. Terrain synthesis using curve networks. In *Proceedings of the 41st Graphics Interface Conference*, GI '15, pages 9–16, Toronto, Ont., Canada, Canada, 2015. Canadian Information Processing Society.

[3] Bevilacqua, C. T. Pozzer, and M. C. Ornella. Charack: tool for real-time generation of pseudo-infinite virtual worlds for 3d games. *Symposium on Games and Digital Entertainment*, 2009.

[4] J. Crause, A. Flower, and P. Marais. A system for real-time deformable terrain. In *Proceedings of the South African Institute of Computer Scientists and Information Technologists Conference on Knowledge, Innovation and Leadership in a Diverse, Multidisciplinary Environment*, SAICSIT '11, pages 77–86, New York, NY, USA, 2011. ACM.

[5] X. Li and J. M. Moshell. Modeling soil: Realtime dynamic models for soil slippage and manipulation. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 361–368, New York, NY, USA, 1993. ACM.

[6] J. Pouderoux and J.-E. Marvie. Adaptive streaming and rendering of large terrains using strip masks. In *Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE '05, pages 299–306, New York, NY, USA, 2005. ACM.

[7] H. P. Ranjali. *Terrain Rendering and Collision: Managing and Rendering Large Environments in Games*. LAP Lambert Academic Publishing, Germany, 2010.

[8] D. Salomon. *Curves and Surfaces for Computer Graphics*. 2006.

[9] R. L. Saunders. Terrainosaurus Realistic Terrain Synthesis using Genetic Algorithm. Master's thesis, A&M University, Texas, 2006.

[10] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes. A survey on procedural modelling for virtual worlds. *Computer Graphics Forum*, 33(6):31–50, 2014.