Interactive Procedural Ground Texturing and Model Scattering for Synthetic Environments

Andrew Tosh
GameSim Inc
Orlando, FL
andrew.tosh@gamesim.com

Eric Snyder
Raydon Corporation
Port Orange, FL
esnyder@raydon

Pedro Ramos
Independent Consultant
Orlando, FL
pedro.pablo.ramos.alonso@gmail.com

ABSTRACT

Developing procedural ground textures and scattering geotypical models based on geospatial surface material data is often a time consuming process of trial-and-error for artists and terrain database engineers. Procedural texturing is the process of using material classification rasters or polygonal land coverage definitions, along with digital elevation models to generate textures that can be applied to the terrain. This technique is used to generate high quality geotypical textures for ground simulations, as well synthetic imagery for air simulations. Likewise, these same techniques are useful for scattering appropriate geotypical models, such as vegetation or urban clutter. While the results of this technique can produce high quality synthetic environments, the process for generating the environment is often tedious and time consuming. The process can typically put artists and database engineers into a long feedback loop of configuring the inputs and evaluating the output. This investment in time, with no guarantee of success, can prevent procedural techniques from being adopted by cost and time sensitive synthetic environment production efforts. This paper details research and development work that addresses the performance and quality issues in building procedural terrains from geospatial source data. The developed algorithms exploit the Graphics Processing Unit (GPU) in order to provide a near real-time visualization of the procedurally generated assets. This interactive mode has the capacity to accelerate the process of building procedural ground texturing and smart model scatter placement by allowing users to immediately see the effects of configuration changes. This research improves efficiency in the usage of procedural generation technology in the production of synthetic environments for the military simulation and training community.

ABOUT THE AUTHORS

Mr. Andrew Tosh is the founder and president of GameSim Inc. Prior to GameSim, Mr. Tosh was the Director of Engineering at the Modeling and Simulation company, AcuSoft, in Orlando, FL. Mr. Tosh left AcuSoft to enter the game industry with Electronic Arts in 2005. While at EA, Mr. Tosh led the development of a number of Electronic Arts games. In 2008, Mr. Tosh founded GameSim with the goal of providing software products and services to the Modeling and Simulation and Video Game Industries. With expertise in 3D rendering, online features, terrain databases and simulations, GameSim is becoming a leader in developing game and simulation software. Mr. Tosh holds a bachelor degree in computer science from the University of Central Florida and an MBA from the University of Florida.

Mr. Eric Snyder is the Chief Science and Technology Advisor for Raydon Corporation and is responsible for Raydon's Applied Research projects. Mr. Snyder has been with Raydon since 1997 and has been specializing in graphics systems and software for most of his career. Since joining Raydon, Mr. Snyder has held the position of Systems/Software Architect until 1999, working on graphics software with one of the first purpose-built PC-based graphics systems, Chief Scientist until 2003, where he and others were responsible for the first successes providing off-the-shelf PC-based graphics systems using "gaming/desktop" graphics. Following that, Vice President of Advanced Development/Technology where Mr. Snyder was in charge of all graphics system development for Raydon's BARE-DI image generator. Mr. Snyder attended Florida State University.

Mr. Pedro Ramos is a consultant to several modeling and simulation organizations with over 33 years of experience in the industry. Mr.Ramos has an extensive background in real-time visual simulation application software, database modeling software, hardware system design and testing as well as independent research and development for real-time geometry processing algorithms for flight and ground based training, some of which were awarded patents and new technology innovation awards. Mr. Ramos is currently the lead software developer on the on SE Core CVEM program and holds a Bachelor of Science in Aeronautical Engineering as well as a Bachelor of Science in Computer Science from the Embry Riddle University.

Interactive Procedural Ground Texturing and Model Scattering for Synthetic Environments

Andrew Tosh
GameSim Inc
Orlando, FL
andrew.tosh@gamesim.com

Eric Snyder
Raydon Corporation
Port Orange, FL
esnyder@raydon

Pedro Ramos
Independent Consultant
Orlando, FL
pedro.pablo.ramos.alonso@gmail.com

INTRODUCTION

The military simulation and training (MS&T) industry has a long history of using procedural techniques to help automate the process of building 3D terrain databases for training systems (Smelik, Tutenel, Kraker, & Bidarra, 2010b). It is common practice to use satellite photo imagery as the ground based texture that is draped on the terrain skin (Ephanov & Coleman, 2006). This strategy has the benefit of being accurate to the real world, which is a major need within MS&T system, as well as scalable to terrain sizes which can reach millions of square kilometers. However, it has a number of drawbacks. Often the imagery resolution (typically 1m per pixel) is not adequate for low flight or ground based training systems. In these situations, the imagery can become extremely blurry and distracting for trainees to understand the environment. Even if the imagery resolution was high enough quality, another drawback remains: the imagery may contain objects that are not meant to be part of the training system. For example, there may be cars on the roads, or tree images in forests, or buildings in the imagery. These objects, which are "burned into" the ground texture, may conflict with 3D features that are added, making it confusing for trainees to understand that they are not part of the simulated system. Removal of the objects from the imagery is possible but can create visual anomalies as well as being a time intensive operation. See Figure 1 for a visual depiction of these two drawbacks to using photo imagery as ground texturing.

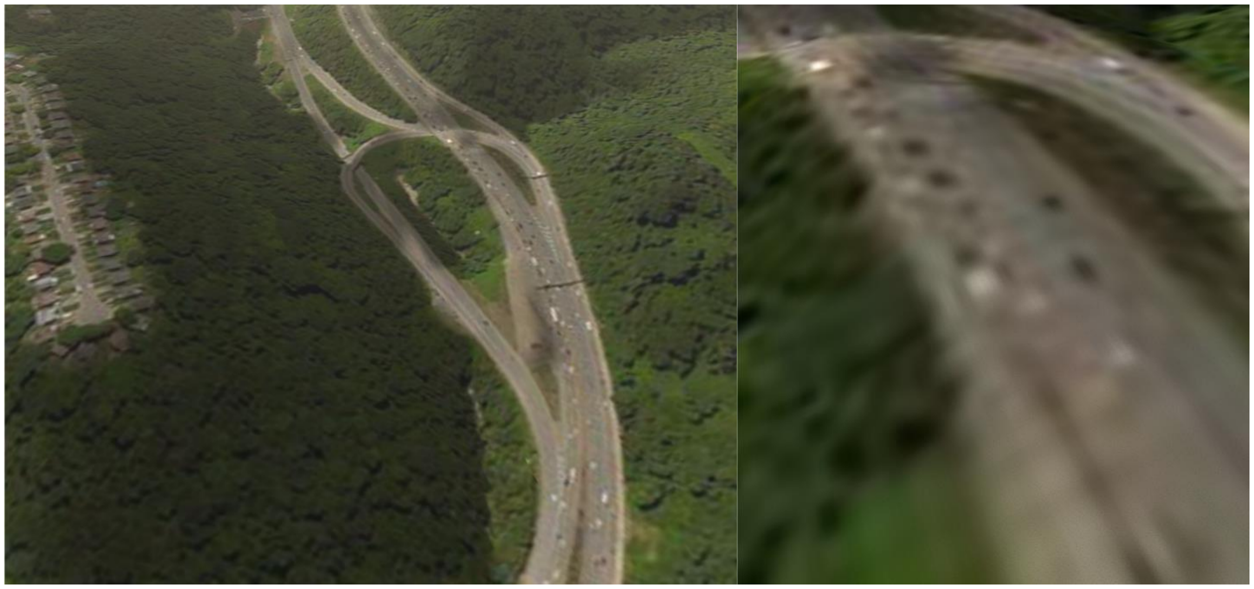

Figure 1. The image on the left shows how vehicles can appear on roads when using photo imagery as the ground texturing, which may create training confusion in MS&T systems. The image on the right demonstrates the blurriness of the photo imagery when visualizing close to the ground, making situational awareness within the MS&T system difficult.

This has led some MS&T terrain database generation systems to rely upon repeating, geotypical texturing based on geospatial land coverage definitions. Often these systems will use geographic information systems (GIS) polygonal land coverage definitions to appropriately texture the terrain skin in the specified area. This technique has been plagued by two major drawbacks: (1) hard edges on coverage definitions boundaries, and (2) long processing times before database generation engineers can view the output. The later issue can have significant impacts on the timeline for building terrain databases, hence, there is a desire to provide capabilities that provide real-time feedback (Bruneton & Neyret, 2008).

The gaming community has long employed a different strategy for ground texturing and model scattering. The gaming community has used real-time generated splatmaps ("Texture splatting" is a term used to describe the computer graphics process of composting multiple textures) as the ground texture (Bloom, 2000). Game engine terrain editors often provide a means to define terrain areas using limited combinations of material definitions. These editors provide terrain painting tools to lay out the material definitions, as well as blend between the areas. For example, you can paint an area of the terrain to use a grass based material and blend that grass area into a mud area. In addition to manually painting, there is also often a means to map these material definitions to terrain elevations and slopes values (Olsen, 2004).

Many game engines are executing the procedural shader splatting at run-time (Andersson & Tatarchuk, 2007). Performing the composition at run-time provides the greatest amount of control for dynamically controlling the ground texturing based on game play, e.g., performing dynamic terrain features. Figure 2 shows how a user of the Unreal game engine can configure surface materials and manually paint them onto the terrain surface.

While performing the compositing at run-time does provide the highest level of control, this is often not an option for constructing terrain databases for MS&T programs for a number of reasons:

1. Many existing programs are using image generators (IG) that do not support the procedural shader splatting, and updating the existing system is not an option
2. The design is meant for artists to manually paint the entire environment (Smelik et al., 2010a), which lacks scalability
3. A limited number of textures can be used based on video card and performance requirements

Therefore, the MS&T industry must look for creative ways to leverage the techniques for procedural shader splatting as part of the offline, terrain database generation process.

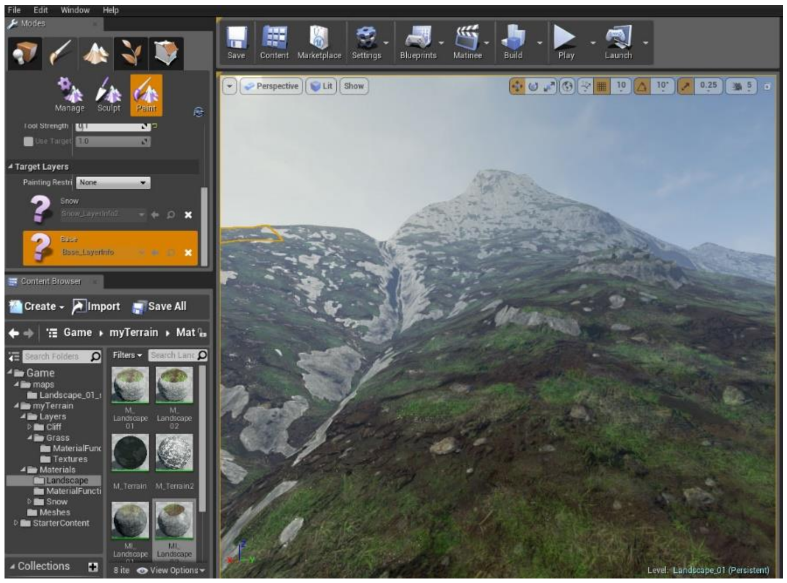

Figure 2. Unreal terrain editor allows users to paint materials onto the terrain skin.

This paper outlines an algorithm that brings the benefits of game engine procedural shader-based splatting to the MS&T community by avoiding the stated drawbacks. The algorithm exploits the GPU to provide MS&T terrain database engineers an interactive experience for configuring the ground texturing and modeling scattering based upon real-world geospatial source data.

PROCEDURAL RESEARCH APPROACH

A cooperative research effort was performed among the authors of this paper to build the ground texturing and model scattering from source geospatial data based on a configuration mapping between the real-world, land usage coverage classifications and a visual coverage definition. The output of the system are textures and geotypical model references that can be integrated into existing, standard MS&T terrain database formats, which can be rendered by current MS&T IGs. Therefore, many existing MS&T systems will not require IG upgrades or game engine replacements to utilize the produced ground texturing. The goals of the procedural approach to processing are to (1) allow a large number of textures to be blended together to produce a high quality visual output, (2) solve the lack of scalability in game engine environments that require hand-editing/painting of the terrain surface, (3) not require any updates to existing MS&T systems in order to prevent costly upgrades (assuming the existing hardware has sufficient video card memory), and (4) provide an interactive tuning experience for MS&T database engineers to configure the procedurally generated output.

With the goals of the research established, an architecture was designed, along with a prototype implementation. The initial design was heavily dependent upon processing within the central processing unit (CPU), but this design lacked the throughput required to achieve the goal of providing an interactive tuning experience. Therefore, the architecture was updated to rely more upon the GPU to provide near real-time performance.

Architecture

As seen in the architecture diagram in Figure 3, there are three sources of input to the system.

1. Land usage coverage definitions; GIS data
2. Elevation data; Digital elevation model
3. Repository of visual coverage definitions

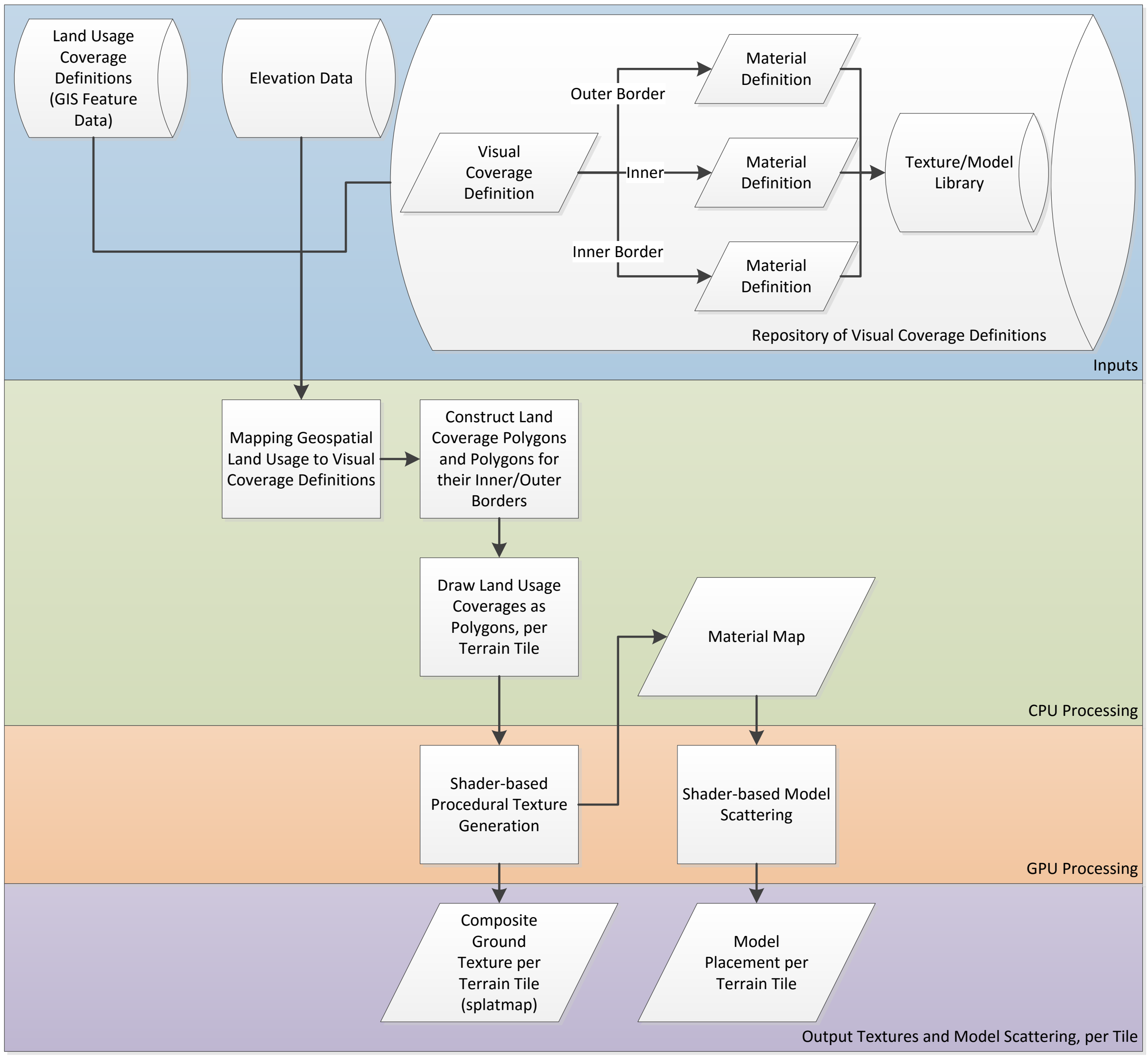
Land Usage Coverage Definitions (GIS Feature Data)
Elevation Data
Visual Coverage Definition
Outer Border
Inner
Inner Border
Material Definition
Material Definition
Material Definition
Texture/Model Library
Repository of Visual Coverage Definitions
Inputs
Mapping Geospatial Land Usage to Visual Coverage Definitions
Construct Land Coverage Polygons and Polygons for their Inner/Outer Borders
Draw Land Usage Coverages as Polygons, per Terrain Tile
Material Map
CPU Processing
Shader-based Procedural Texture Generation
Shader-based Model Scattering
GPU Processing
Composite Ground Texture per Terrain Tile (splatmap)
Model Placement per Terrain Tile
Output Textures and Model Scattering, per Tile

Figure 3. Architecture for how the source geospatial data is used to procedurally build the ground textures and model placements.

Land Usage Coverage Definitions

GIS feature data is used as a primary input to the system for obtaining classification of the terrain surface. This vector-based data can represent the surface using polygonal and linear features. For example, a waterbody may be represented with a polygonal outline and an attribute that classifies the area as being water. Linear features, such as roads and rivers, do not explicitly have an outline. However, these features are often attributed with a *width* attribute, which can be used to derive a polygonal outline of the feature.

The source land coverage definitions from polygon and linear features are used to drive appropriate construction of the procedural ground texturing and model scattering, as opposed to game engine terrain environments that rely upon artists to manually paint the terrain skin.

Elevation Data

The digital elevation models provide the elevation of the terrain skin. The slope of the terrain, as well as the absolute elevation, are used as part of the mapping to the visual coverage definitions. For example, if the land usage polygon classify an area as being a forest region, but the slope is too great within that polygonal area in some regions to support vegetation—the algorithm can use slope thresholds to ensure vegetation is not scattered in those slopped areas. Likewise, if the absolute elevation is high enough to imply ice caps, we can map that region to a visual coverage definition that includes snow texturing.

Visual Coverage Definition

The visual coverage definitions provide a configuration on how an area shall be textured, as well as model scattering. The definition links to at least three material definitions. There are two material definitions for how the inner and outer borders should be visualized, and another for the interior of the area (see Figure 4). Additionally, other materials may be specified based on elevation slope thresholds and absolute elevation.

The borders are constructed based on a configurable width around the source land usage coverage outline.

A visual coverage definition will be required for each unique input land usage coverage definition. For example, a land usage coverage definition that is classified as being a forest shall be mapped to an appropriate forest based visual coverage definition. The visual coverage definitions can reference material coverage definitions for controlling the visual texturing and model scattering for borders, as well as the inner area.

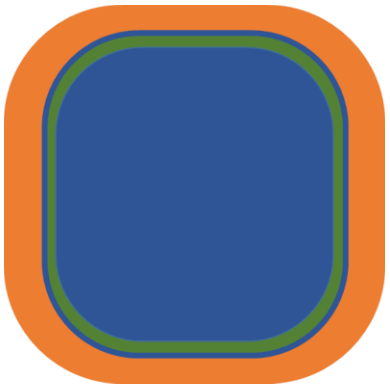

Figure 4. A source land coverage definition is shown in blue, with a generated outer boarder in orange and inner border shown in green. The visual coverage definition can map each of these areas to a separate material definition.

Below is an example visual coverage definition file for a forest region.

```
material: Mixed_Forest

outer_border_enable: true
outer_border_material: Grass_Land
outer_border_width: 15
outer_border_blend_type: noise
outer_border_noise_blend_scale: 1

inner_border_enable: false
inner_border_material: None
inner_border_width: 0
inner_border_blend_type: smooth
inner_border_noise_blend_scale: 1
```

Material Coverage Definition

The material coverage definition stores the necessary attributes and art assets for constructing the visual representation.

- A set of textures that are composited to construct the final texture, based on weight, noise, and scaling attributes associated with each texture

- A set of 3D models that are to be scattered within the coverage, based on density, range, tinting, and positioning attributes associated with each 3D model
- A detail texture
- Surface material codes (SMC) and feature ids (FID) in order to support run-time specific behaviors, such as thermal rendering modes

Below is an example material coverage definition file for a forest region.

```
parameters:
  noise:
    frequency_scale: 0
    sharpness: 1.2
  rows:
    - texture_name: Sand_Gray.dds
      noise_weight: 2
      tex_scale: 400
      real_world_size: 1
    - texture_name: Rocks_Gray_Medium.dds
      noise_weight: 3
      tex_scale: 300
      real_world_size: 1
    - texture_name: Grass.dds
      noise_weight: 3.7
      tex_scale: 100
      real_world_size: 1
models:
  - name: Atlas_1/BlueSpruce_RT.fbx
    density: 0.002
    range: 1000
    tint_min: 0.1
    tint_max: 0.7
    xyscale_min: 1
    xyscale_max: 2
    jitter: 1
    zscale_min: 1
    zscale_max: 1
    z_offset: 0
    is_grass: false
  - name: Atlas_1/FraserFir_RT.fbx
    density: 0.005
    range: 1000
    tint_min: 0.1
    tint_max: 0.7
    xyscale_min: 3
    xyscale_max: 4
    jitter: 1
    zscale_min: 1
    zscale_max: 1
    z_offset: 0
    is_grass: false
detail_texture_name: "detail.dds"
detail_texture_scale: 1
smc: 0
fid: 0
```

Algorithm

Once the mapping between the geospatial land usage coverage and the visual coverage definitions has been established, the procedural generation of the texturing and scattering can be largely executed in the GPU. The benefit to performing this implementation within the GPU is that database engineers that are configuring the mappings and coverage definitions can get near real-time feedback on the final look of the ground texturing and model scattering.

The algorithm for building the procedural ground texture for a terrain tile is shown below.

1. Establish a target frame buffer at the resolution of a terrain tile size.
2. Apply the elevation data as a texture for the given terrain tile.
3. Apply the slope data as a texture for the given terrain tile.
4. Group the source land usage polygons (including laid linear features, e.g., roads, are constructed into polygons using their width attribution) that are contained within the terrain tile.
5. Bind the inner material coverage definition as shader parameters.
6. Draw the land usage polygons.
7. A shader applies the material coverage definition to the drawn polygons.
8. Bind the inner border material coverage definition as shader parameters.
9. Construct and draw the land usage polygons inner border strip along the edge of the source polygon.
10. A shader applies the material coverage definition to the inner border polygons being drawn.
11. Bind the outer border material coverage definition as shader parameters.
12. Construct and draw the land usage polygons outer strip border along the edge of the source polygon.
13. A shader applies the material coverage definition to the outer border polygons being drawn.
14. Read back the final frame buffer (this is the final image to be used as a texture across the terrain tile).

The logic for 5, 8, and 9 is all executed within a pixel shader on the GPU. The logic is able to apply any number of source textures from the material coverage definition, based on the slope and height of the terrain, which is provided in two texture slots to the shader in step 2 and 3. Figure 5 presents an example of configuring the system to use a snow visual coverage definition for high elevations and a rocky, unvegetated visual coverage definition for extreme slopes.

The algorithm for performing the model scattering utilizes an aspect of the procedural ground texturing algorithm. While the shader is building the final visual texture, it is also building a material map into a separate texture slot. This material map will provide the basis of executing the scatter model placement. The material map stores the material coverage definition id for each pixel in the raster. This raster is then fed into a geometry shader program, along with the material coverage definition scatter parameters to place appropriate 3D models across the terrain.

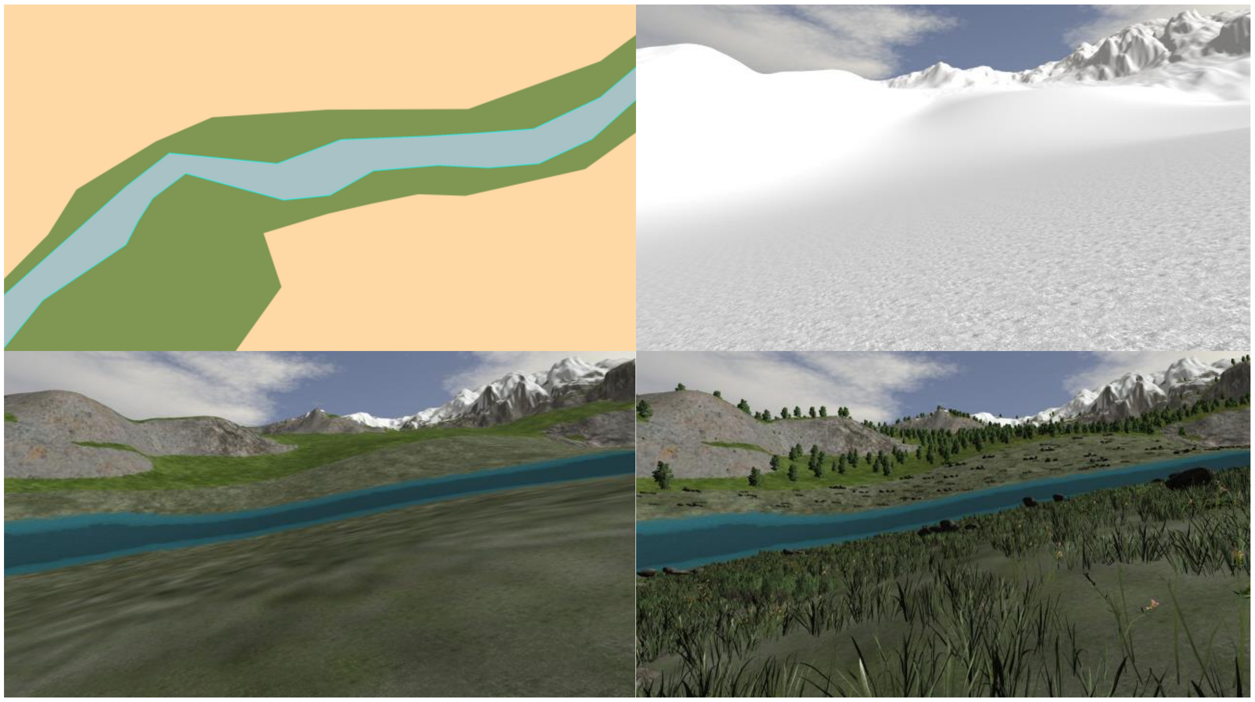

Figure 5. Using the land usage coverage definitions (upper left) and elevation data (upper right) as inputs, ground texturing (lower left) and appropriate vegetation models (lower right) are produced as the outputs.

Overlapping Coverages

It is not uncommon for the land usage coverage definitions to overlap, which is handled by the algorithm prioritizing the definitions. Figure 6 shows a forest areal feature mapped to a forest visual coverage definition. The generated 3D visualization, with appropriate ground texturing and model scattering, is displayed to the user in milliseconds. The second set of images in Figure 6 shows when a higher priority feature is rendered on top of the forest region, a trail that runs through the forest. The trail visual coverage definition is given priority over the forest definition, causing appropriate ground texturing and removal of the vegetation models. It is important that the feature processing be prioritized based on the visual preferences. However, the ground texturing visual coverage definition can use the alpha channel such that overlapping feature types are blended together.

MS&T database engineering using the proposed research can configure the feature priorities, configure the visual coverage definitions, and edit the source vectors, while immediately visualizing the results.

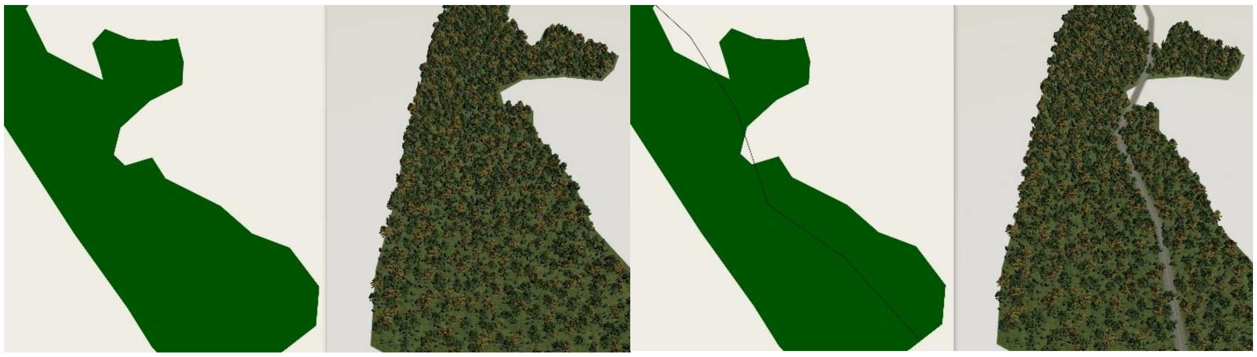

Figure 6. The set of the images on the left shows a source forest land coverage definition and the conversion of that region into an appropriate ground texture along with vegetation models scattered. The set of the images of the right, shows the same area with a trail running through the forest.

Borders

In order to avoid hard edges between land coverage definitions, boarders can be configured to blend with random or smooth techniques from one surface definition to another. These border techniques are implemented in the shader. In order to facilitate rendering of the inner and outer boarders, the edge of the source surface coverage must be constructed into a separate polygon to be rendered by the GPU, with a border specific visual coverage definition. Figure 7 presents how a forest area can blend into a snowcapped mountain area, based on elevation data, using this border technique to blend in a noisy pattern from the forest into snow.

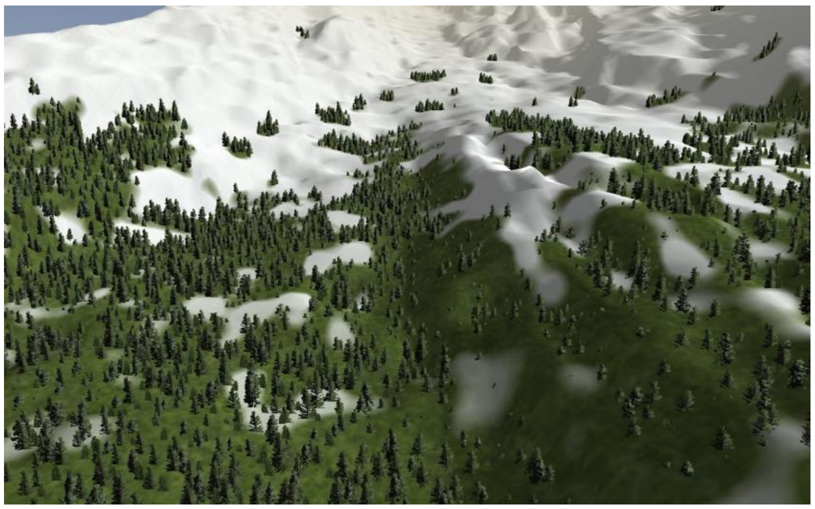

Figure 7. Borders are fused to blend from one land surface coverage to surrounding coverages in order to prevent hard lines.

Multiple Visual Coverage Styles

IGs in MS&T often require multiple versions of the ground texturing and scatter in order to support all seasons of an environment (e.g., winter and summer), as well as the various sensor modes (e.g., thermal). In order to support multiple ground texturing and scatter sets, each visual coverage definition can have several versions. For example, a single forest visual coverage definition may have the following versions.

- Summer
- Autumn
- Winter
- Spring
- Forward Looking Infrared
- Night Vision Goggles

The algorithm executes the same set of steps using the visual coverage version for each style.

Performance

By using a shader based implementation for generating the ground texturing and model placement, the prototype achieved near real-time performance in generating and visualizing the output. This performance is vital for users to quickly preview the coverage mappings and make necessary adjustments to achieve the desired look.

For example, a ground texture and model scattering was configured for downtown Pittsburgh. In the 2500m by 2500m tile below (see Figure 8), there are 783 source land usage coverage features. On an Intel Core i5-3570 CPU with an NVIDIA GTX 970 Windows machine, the algorithm produced the procedural ground texturing in 350 milliseconds and the vegetation model scattering in 150 milliseconds. These times feel nearly instantaneous to users of the system. This facilitates rapid configuration iterations on the visual coverage definitions to achieve the desired look.

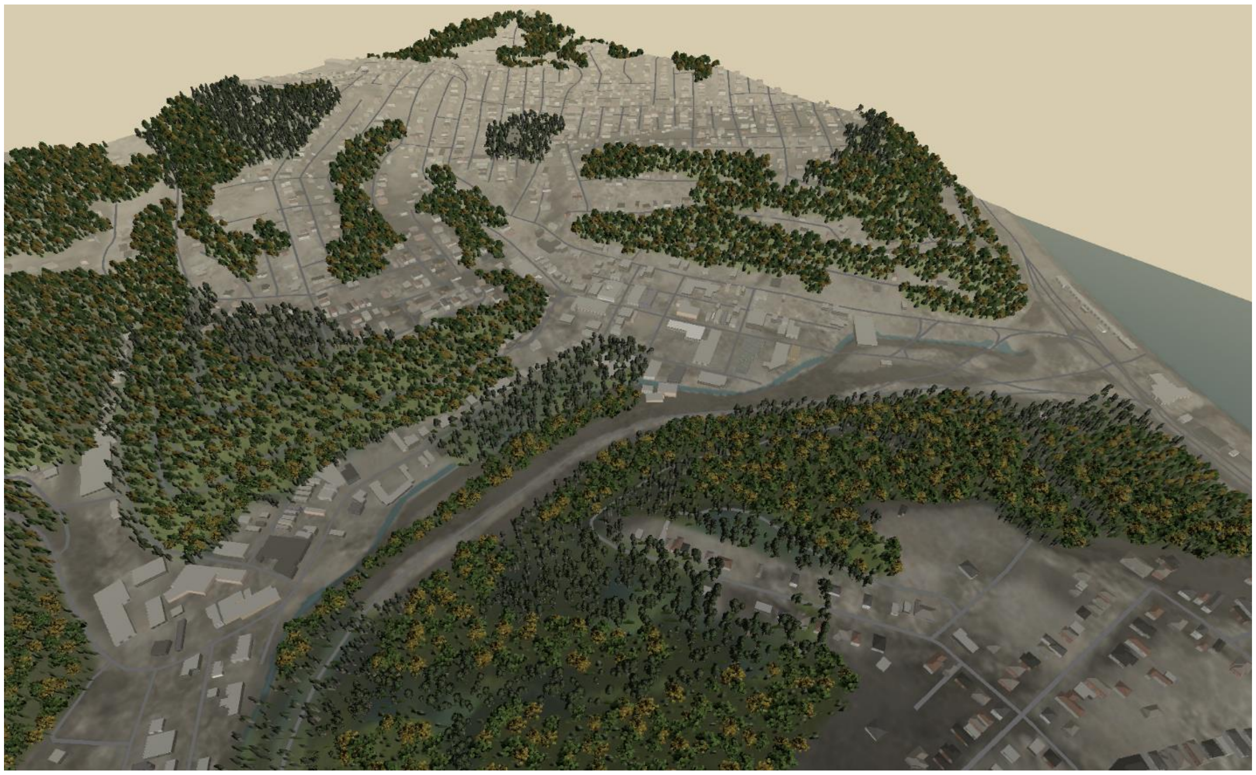

Figure 8. Procedurally generated terrain texturing and model scattering produced for a 2500m by 2500m terrain tile.

Distributed Builds

Once the desired look has been achieved, a full build of the environment can be generated using a distributed, cloud-based architecture. A concern that arises when using distributed multi-machine systems in a secure environment is the nature of the communications and coordination of execution across the participating machines.

Most of the security constraints that impact the system architecture revolve around the nature of the communication connections established between the participating machines, and although constantly evolving, limit the remote execution of processes, the acceptance of remote connections on non-server machines, and limit the environments where multi-machine coordination processes that need to “connect” to multiple machines may run.

In order to satisfy these concerns, user interactive application modes can take advantage of localized, small area on interest environments and generate the data locally by taking advantage of GPU implementations, thus avoiding the multi-machine cross-communication issues.

In cases when multiple machines are needed, as in the case of batch processing of larger areas, the software needs to be set up so that the work load is carefully distributed in such a way that non-server, worker machines initiate their own connections to receive data generation instructions and are able to execute approved, machine-local applications locally in order to avoid remote procedure execution issues. This can normally be achieved through a combination of local and network services as well as server-level data sharing and centralized coordination services, and these distributed systems need to provide a variety of deployment options in order to be able to customize the distribution of work given the local information assurance needs of the organizations.

Drawbacks

One noteworthy drawback of the proposed approach is the increased amount of necessary texture memory of the run-time system, as each tile will have a unique ground texture, as opposed to using repeating, geotypical texturing. Low cost video cards are equipped with a large amount of video card memory, making this a minimal limitation; however, some legacy MS&T systems may not be using modern video card hardware. Therefore, when building the procedural ground texturing, the video card memory budget of the run-time system must be taken into account when selecting the resolution of the generated ground textures. The algorithm can easily support producing any power-of-two resolution, by specifying that resolution when setting up the target frame buffer.

CONCLUSIONS

The algorithm outlined in this paper details an approach for bringing the visual quality of game engine style procedural ground texturing to the MS&T community in order to avoid the pitfalls of using photo imagery as the ground texturing. The process provides a realistic solution for existing training solutions to adopt, as it does not require any runtime modifications to the existing IGs. Additionally, the performance of the system may produce a significant cost savings to programs by supporting an interactive, near real-time configuration capacity. This interactive mode facilitates database engineers to rapidly produce high quality ground texturing and model scattering for their synthetic environments.

ACKNOWLEDGEMENT

This research and development was performed by Raydon Corporation, Pedro Ramos, and GameSim Inc as part of a cooperative initiative to improve visual quality and the production cycle of synthetic environments for their customers.

REFERENCES

Andersson, J., & Tatarchuk, N. (2007, March 5-9). *Frostbite Rendering Architecture and Realtime Procedural Shading and Texturing Techniques*. Retrieved from http://ati.amd.com/developer/gdc/2007/Andersson-Tatarchuk

Bloom, C. (2000, November 2). *Terrain Texture Compositing by Blending in the Frame-Buffer*. Retrieved from http://www.cbloom.com/3d/techdocs/splatting.txt

Bruneton, E., & Neyret, F. (2008). Real-time rendering and editing of vector-based terrains. *Computer Graphics Forum*, 311 - 320.

Ephanov, A., Coleman, C. (2006). Virtual Texture: A Large Area Raster Resource for the GPU. *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*.

Olsen, J. (2004). Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games. *Department of Mathematics And Computer Science (IMADA)*, 1-20.

Smelik, R. M., Tutenel, T., Kraker, K. J., & Bidarra, R. (2010a). Declarative Terrain Modeling fo rMilitary Training Games. *International Journal of Computer Games Technology*.

Smelik, R. M., Tutenel, T., Kraker, K. J., & Bidarra, R. (2010b). Interactive Creation of Virtual Worlds Using Procedural Sketching. *The Eurographics Association*.