Virtual VMASC: A 3D Game Environment

Suchitra Manepalli, Project Scientist; Yuzhong Shen, Assistant Professor; Hector M. Garcia, Senior Project Scientist; Kaleen Lawsure, Project Scientist
Virginia Modeling, Analysis, and Simulation Center
smanepal@odu.edu, yshen@odu.edu, hgarcia@odu.edu, klawsure@odu.edu

Abstract The advantages of creating interactive 3D simulations that allow viewing, exploring, and interacting with land improvements, such as buildings, in digital form are manifold and range from allowing individuals from anywhere in the world to explore those virtual land improvements online, to training military personnel in dealing with war-time environments, and to making those land improvements available in virtual worlds such as Second Life. While we haven't fully explored the true potential of such simulations, we have identified a requirement within our organization to use simulations like those to replace our front-desk personnel and allow visitors to query, navigate, and communicate virtually with various entities within the building. We implemented the Virtual VMASC 3D simulation of the Virginia Modeling Analysis and Simulation Center (VMASC) office building to not only meet our front-desk requirement but also to evaluate the effort required in designing such a simulation and, thereby, leverage the experience we gained in future projects of this kind. This paper describes the goals we set for our implementation, the software approach taken, the modeling contribution made, and the technologies used such as XNA Game Studio, .NET framework, Autodesk software packages, and, finally, the applicability of our implementation on a variety of architectures including Xbox 360 and PC. This paper also summarizes the result of our evaluation and the lessons learned from our effort.

1. INTRODUCTION

Interactive 3D virtual environments present a unique scope allowing both individuals and organizations to analyze and practice methods that are otherwise difficult. Those methods may range from the military studying geographical regions represented virtually in 3D to launch or defend attacks, to realtors advertising real estates in 3D virtual environments to their clients. Further, the success of online versions of 3D environments, such as Second Life and OpenSimulator, attest to the applicability and power of the virtual environments. While the use cases are many, the process of designing and developing such environments is the same at many levels. To evaluate the effort required and to study the problems that might arise when designing such environments, we have designed and implemented "Virtual VMASC", a 3D game, with the goal to replace front-desk personnel with an Xbox console assisting the guests arriving at the VMASC facility. The implementation provides a visual interface to search and browse for various pieces of information including faculty and staff directory, navigational maps to individual offices, and presents ongoing research information. Guests, who are represented as avatars in the virtual world, are free to walk within the building, perhaps following a map to a specific room, interact with various entities on the way, and eventually be able to talk to individuals through video conferences from the console.

The higher-level goals of this effort are to study and evaluate the level of effort needed to model, design, and implement 3D games, and also to study the effectiveness of the various software and platforms chosen for this implementation. We believe our study provides useful information to the Modeling and Simulation community in dealing with similar efforts.

2. MODELING

In this section, we discuss our modeling approach and the software and toolset we used for creating the 3D model for the gaming environment. We also highlight the solutions implemented to deal with the problems encountered while constructing, texturing, and prepping the model. Finally, we discuss the areas to improve in the model, which we tabled for the future.

3. VMASC 3D MODEL

The VMASC facility, located in Suffolk, Virginia, is a two-floor building, divided into east and west wings by a large atrium, with 120 rooms including office, lab, conference, supply, and utility spaces along with 5 restrooms and 3 kitchens. In order to construct a 3D virtual model of the facility, a variety of 3D digital content creation software technologies were used such as Autodesk 3ds MAX [2], Autodesk Maya [3], and Google

SketchUp Pro [5] as well as 2D image editing software products such as Adobe Photoshop CS4 [1] and Luxology's imageSynth [7].

The process of creating a 3D model of the VMASC building began by importing 2D AutoCAD drawings of the floor plan into 3ds Max. After importing the floor plan, walls were generated by using the extrusion tool which generates 3D extruded objects from 2D lines, in this case the 2D lines from the floor plan as illustrated in Figure 1.

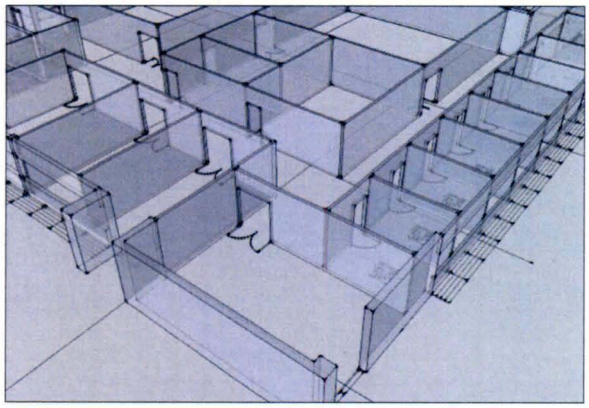

Figure 1: 3D objects extruded from 2D floor plan

Wall height, window placements, doors, and other elevation features of the building that are required for performing extrusion were extracted from digital elevation drawings, pictures, and actual measurements of the building. Additionally, certain geometric and floor plan layout features had to be corrected due to changes in the real world building that had not been reflected in the available 2D AutoCAD drawings. In order to make those corrections, the latest paper based floor plans were used along with a physical walk through of the building. In addition to the geometric corrections, certain other features were added such as the glass for the windows enclosing the atrium, east stairs, west stairs, interior windows, benches along the hallways, a roll up door at the loading dock, and a revolving door at the front entrance.

In order to make these corrections and additions, some objects were edited, while others were created from scratch. Some objects were converted into editable meshes for the purpose of modifying their geometry, and a variety of 3ds Max's tools were used for editing including tools that allow the objects to move, rotate, scale, extrude, bevel, clone, align, and attach. For creating missing features, 3ds Max was used as it allows creating simple geometric objects such as boxes and cylinders, complex objects such as knots and spindles, and architectural objects such as doors, windows, stairs, and handrails. Those standard primitives, complex objects and architectural objects were used to add features that would replicate the VMASC building in the model.

3.1 Textures

After the initial 3D building structure was created, we added textures and other materials to the model to capture the interior and exterior design of the building. Some existing textures from the 3ds Max texture library were used. However, given the high level of detail desired, it was necessary to use as many realistic textures as possible for the building's exterior, interior, and contents. In order to facilitate this, a Samsung SL310W 13.6 megapixel digital camera was used to photograph the building (in the real world). The images were processed using Adobe Photoshop CS4 and imageSynth before applying it to the model.

Photoshop CS4 offers a variety of tools for adjusting hue, saturation, contrast, brightness, exposure, and color of the images. It also provides tools for cropping, rotating, erasing, color sampling, and layering. Those image manipulation tools were used to crop and properly align the images, remove any undesirable blemishes or shadowing, and correct problems with colors or shading. However, after improving the images in Photoshop, we still faced problems tiling the textures. This problem was eliminated by imageSynth software, which creates seamless textures from the processed images, as illustrated in Figure 2.

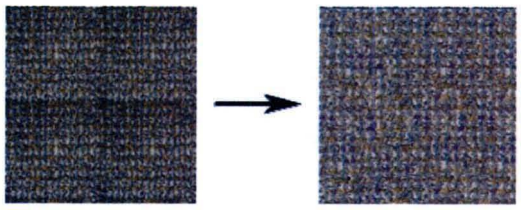

Figure 2: Uncorrected carpet tile (left) and Corrected carpet tile (right)

After correcting the textures, we created a palette of materials for the entire building, some of which are illustrated in Figure 3. The UVW texture mapping tool from 3ds Max was used to place the textures correctly in the 3D model.

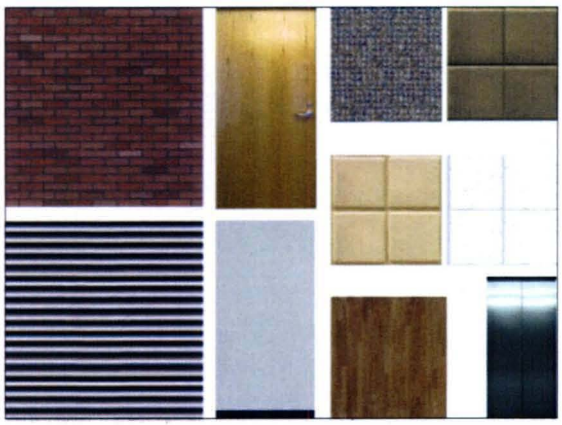

Figure 3: Sample material palette used for the VMASC 3D building

The next step in designing the model was to export it to a FBX file format, which is compatible with XNA – our gaming platform. Google SketchUp Pro was used to import the model from 3ds Max and then export to a FBX file as it provided a better FBX file that XNA is compatible with than the one generated from 3ds Max.

3.2 Hierarchical 3D Model

After designing the model, we transformed it into a hierarchical one using Autodesk Maya's hypergraph hierarchy tool. This tool allows the user to arrange the objects hierarchically one within the other, as illustrated in Figure 4 that optimizes the collision detection mechanism as described in the Technical Approach section.

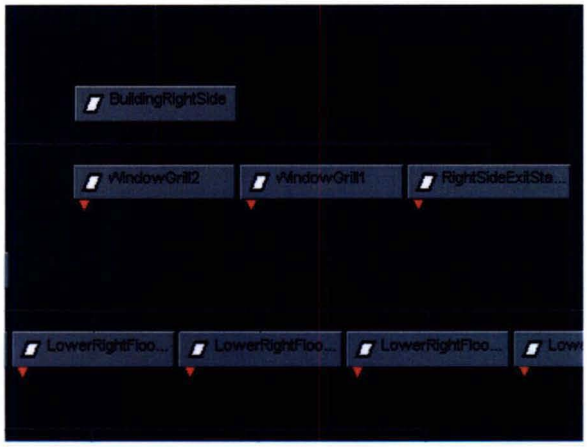

Figure 4: Generated hierarchy using the hypergraph tool (partial hierarchy shown)

Snapshots of the final model are illustrated in Figures 5 and 6.

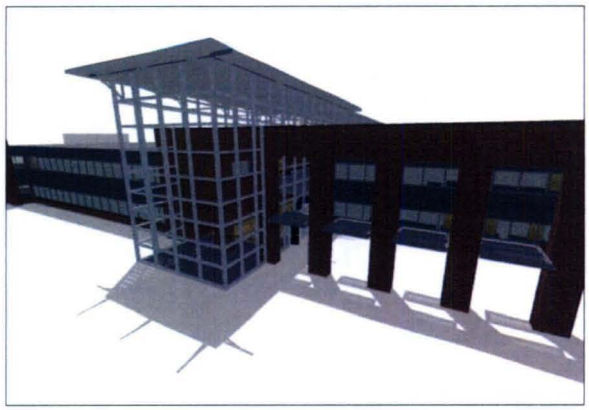

Figure 5: Front view of the VMASC 3D model

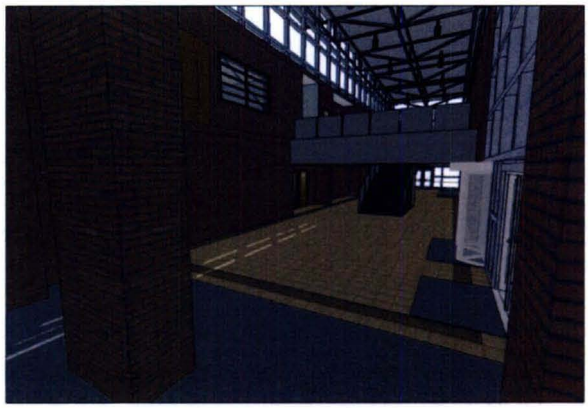

Figure 6: VMASC 3D building with textures

3.3 Model Refinement

There are still some additional details that we would like to add in the future. Those details include adding sinks, commodes, and stalls for the restrooms along with cabinetry and appliances for the kitchens.

Additionally, furniture including chairs, tables, desks, bookshelves, cabinets, and other office needs are to be added to the model. Those will have to be created using digital photographs from the real world.

4. IMPLEMENTATION FRAMEWORK

We have implemented the 3D game using Microsoft XNA Game Studio [8], which is a software library and toolkit targeted at independent and small game studios, academics, and hobbyists. XNA supports cross-platform game development for personal computers, Microsoft Xbox 360, and Zune media player.

At the core of XNA is the XNA Framework, a set of C# libraries for game development based on the Microsoft .NET Framework. C# is an object-oriented programming language drafted by

Microsoft and approved by ISO as a standard. The XNA Framework encapsulates low-level details involved in developing games and allows game developers to focus more on the content and high-level gaming experience. Game developers can more rapidly learn the truly important and difficult parts of game development without dealing with the low-level details such as lighting, shadow effects, etc. Developers can use both the XNA Framework and the .NET Framework in a game with the former for game-specific tasks such as graphics rendering and managing inputs and the latter for more general programming tasks.

The XNA Device Center lets developers manage and connect to multiple XNA devices, including Xbox 360s and Zune devices. The XNA Game Studio supports features including avatars, animations, and embedded videos. XNA can compress and decompress content such as meshes and textures automatically to reduce storage space usage and deployment time. XNA also supports content access from a device's media library such as songs, pictures, and playlists. The ClickOnce deployment technology can create self-updating Windows based applications that can be installed and run with minimal user interaction.

5. TECHNICAL APPROACH

The Virtual VMASC 3D simulation, designed as an Xbox game, involved designing a 3D model, developing game heuristics, and designing an interface to meet our front desk requirements. Transforming the VMASC 3D model, the design of which is discussed in the Modeling section, into a game using XNA while still coherently presenting a real world experience to the player through the avatar resulted in many challenges including presenting a responsive and realistic 3D world, detecting and handling collisions, and providing various viewing (aka camera) modes. Additionally, optimization of the model and texture rendering and collision detection techniques proved to be pivotal for designing a responsive game. The following sub-sections highlight some of those technical challenges and the solutions we adopted for resolving them.

5.1 Rendering

A model is usually comprised of a composite of multiple sub-models. While different mechanisms may be used to render those sub-models, it is important that any model rendering mechanism employed should eliminate lag, jitter, flickers, and other un-real artifacts.

The Virtual VMASC 3D model uses a variety of textures for realistically representing the VMASC building, as discussed in the Modeling section. XNA renders those textures automatically if referenced in the FBX model [8]. However, the building includes a large number of meshes and textures, the rendering of which is process intensive resulting in unrealistic lag and artifacts during model representation as part of the game. In order to mitigate this issue, a known solution based on the *octree* mechanism is implemented [6]. The crux of this solution is to recursively divide a model into eight equally sized cubes until the leaf cubes contain a specified number of spatial objects. Once a model is thus split, thereby resulting in a hierarchical graph (or tree), only those sub-cubes that are in the viewable area (based on the field of view) are processed for rendering, resulting in a cleaner, faster and responsive game. Figure 7 illustrates the visual clarity we achieved after incorporating the octree solution.

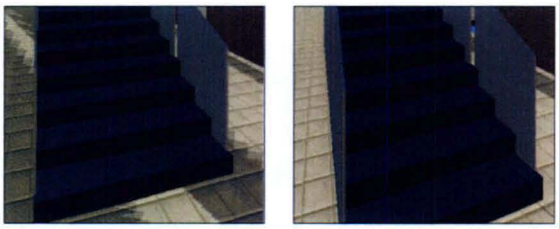

Figure 7: Unwanted artifacts resulting from the default XNA rendering (left) and Clean display resulting from the integration of the octree technique (right)

5.2 Collision Detection and Handling

XNA is in-built with content pipeline architecture for importing *art assets* from the model as binary objects (aka mesh-parts) that may, then, be processed and controlled as required by the game. The content pipeline converts art assets into binary objects using four components [4]:

a. Importer: XNA supports and provides a number of importers. One such importer for Autodesk is the *FBX* importer, which is used in our implementation. Importers convert a model into managed objects conforming to the Content Document Object Model that is processed further by the content processor.
b. Content Processor: Content processors process the managed objects generated

by importer and creates custom managed objects, if required for special gaming requirements.

c. Content Compiler: The compiler bundles together the managed objects generated by the content processor into a compact binary asset for faster run-time loading.
d. Content Loader: The loader is responsible for locating and loading the compiled asset into memory.

Related to the content pipeline architecture is the process of collision detection. XNA associates bounding spheres with meshes to deal with the location and arrangement of those meshes [6]. It is easy to see, however, that not all meshes are spherical in shape (walls, furniture blocks, doors, avatars, etc.); as such, enclosing those meshes in spheres especially to process the location and detect collisions is unrealistic, although simpler. In our model, we identified that most of the spatial features are box-like structures and enclosing those features within bounding boxes is ideal.

The FBX model we designed also associated bounding boxes with each of those meshes. To derive the bounding boxes information from the model into the gaming runtime, instead of using the default bounding spheres created by XNA, a custom content pipeline is implemented. At the time of rendering, the retrieved information is used to load bounding boxes for the meshes. Those bounding boxes are then used for detecting and handling collisions.

Avatars, which can move around the VMASC building model, are the reason to perform collision detection and handle those collisions. Normally, collisions occur when the avatar hits blocked surfaces like a wall or a closed door. However, handling collisions between an avatar and stairs is complicated. Instead of not allowing the avatar to proceed further on its path during such collisions, it needs to climb up or down depending on the direction of the stair mesh. This requires identifying the direction of the avatar and the orientation of the stair meshes before handling those types of collisions.

Handling collisions may be simply done by verifying if the avatar's bounding box is colliding with any of the building's bounding boxes [6]. However, performing detection in a brute force fashion by checking one bounding box after the other from the building is process intensive, and, given the number of meshes in the VMASC building, has resulted in a substantial lag after every move the avatar makes. To deal with this issue, we redesigned the FBX model by creating a hierarchy of meshes. That is, as per this re-design, the entire building is a mesh; each of the floors in the buildings is represented as a separate mesh within that building mesh. The rooms are sub-meshes within those floor meshes, and this process was continued until every object is modeled. The advantage of this hierarchical representation is that the number of collision detections is reduced logarithmically compared to the brute-force approach. This is because, in the hierarchy of meshes (logically represented as an n-tree), the avatar may collide only with a particular path leading from the root to the leaf node, and as such all other computations are not necessary to detect the actual source of collision. Although, we designed and implemented this novel way of collision detection recently, early results seem to have corrected the lag problem dramatically.

5.3 Camera Modes

We implemented three camera modes giving multiple views of the model as the avatar walks through the building. A first person camera mode, in which the world is viewed through the avatar's eyes, results in a very realistic experience. We also implemented a chase camera mode in which it appears as if someone with a camera is following the avatar. An additional elastic effect where the camera slowly comes to a stop although the avatar had abruptly stopped gives a realistic experience for the viewing user. Finally, a static camera mode that just displays a constant field of view is also implemented. This camera mode is best used when multiple cameras are fixed, thereby, allowing multiple fields of view. Figure 8 illustrates the distinction between the first person and chase camera modes.

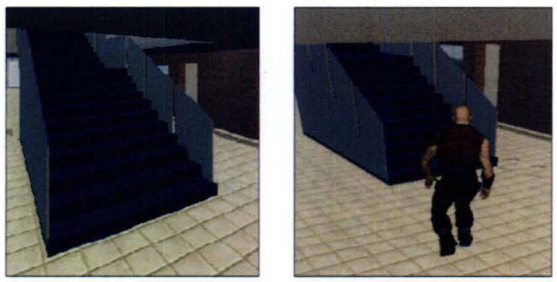

Figure 8: First person camera mode (left) and Chase camera mode (right)

5.4 Audio and Visual Interface

Since the goal of the game is to achieve virtualized front-desk features vis-à-vis providing

faculty and staff directory, ongoing research and projects, and navigational maps to individual rooms within the facility, a variety of menus and screen flows are implemented. Initially, the game is loaded with a welcome screen, as illustrated in Figure 9, which provides the following menu options:

a. Tour of the building

b. Personnel Directory

c. Staff Search

d. Cluster Information

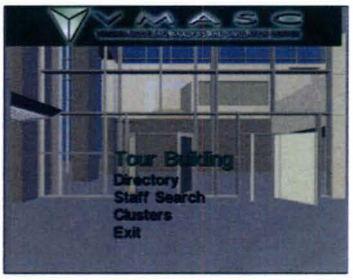

Figure 9: Virtual VMASC visual interface

Users interacting with the game may tour the building as an avatar, browse through the floors by going up and down the stairs, and visit the offices. Users have the choice of various camera modes to flip through during the tour. The second option offers the users a personnel directory to browse and select any office personnel from the system to get additional information, which includes the selected person's office address, email address, and directions to his/her office. The third option lets the users query a personnel database based on a name. The fourth option provides information about the different research areas that VMASC focuses on.

The game also incorporates audio capability, implemented using the Microsoft Cross Platform Audio Creation Tool [8], which enables users to hear any recorded messages or music configured while interacting with the system.

6. CONCLUSION

We found that the XNA game studio coupled with the Autodesk and other software technologies we employed provides a good environment for modeling and developing interactive 3D simulation environments. Autodesk software provides a variety of tools to deal with many modeling issues, and XNA provides functionality to integrate typical gaming scenarios easily. The obvious advantage is the integration of the game developed using XNA into Xbox 360 consoles. However, there are many challenges that we faced as discussed and those challenges required custom implementations and techniques to be employed.

We believe our study, which resulted in valuable lessons that we learned, would also benefit the Modeling and Simulation community.

7. REFERENCES

1. Adobe Photoshop CS4. *http://www.adobe.com/products/photoshop/photoshop/*
2. Autodesk 3ds Max. *http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=5659302*
3. Autodesk Maya. *http://usa.autodesk.com/adsk/servlet/index?id=7635018&siteID=123112*
4. Carter, Chad. "Microsoft XNA Unleashed: Graphics and Game Programming for Xbox 360 and Windows," *Sams.* August 5, 2007, pp. 113-119.
5. Google SketchUp Pro. *http://sketchup.google.com/*
6. Grootjans, R. (2008). "XNA 2.0 Game Programming Recipes: A Problem-Solution Approach," *Apress.* July 11, 2008, pp. 63-66, 88-110.
7. imageSynth. *http://www.luxology.com/whatismodo/imageSynth/*
8. *Microsoft, "XNA Developer Center,"* 2009. *http://msdn.microsoft.com/en-us/xna/default.aspx*