International Journal of Research Studies in Science, Engineering and Technology [IJRSSET]
Volume 1, Issue 4, July 2014, PP 36-41
ISSN 2349-4751 (Print) & ISSN 2349-476X (Online)

3D Game Engines as a New Reality

Deepak Modi[1], Ajay Jaiswal[2], Mayank Gupta[3],
Mayank Mandloi[4], Mohit Mehta[5], Dhairya Mukheja[5]

Department of Computer Science & Engineering
Chameli Devi School of Engineering, Indore, India
[1]dmrohitmodi62@gmail.com, [2]ajay.jaiswal55555@gmail.com
[3]mayank.gupta2805@gmail.com, [4]mayank.mandloi@gmail.com
[5]mehtamohit0766@gmail.com, [5]dhairyamukheja@gmail.com

Abstract: *Understanding players' visual attention patterns within an interactive 3D game environment is an important research area that can improve game level design and graphics. Several graphics techniques use a perception based rendering method to enhance graphics quality while achieving the fast rendering speed required for fast-paced 3D games. Game designers can also enhance game play by adjusting the level design, texture and color choices, and objects' locations, if such decisions are informed by a study of players' visual attention patterns in 3D game environments. This paper seeks to address this issue. We present results showing different visual attention patterns that players exhibit in two different game types: action-adventure games and first person shooter games. In addition, analyzing visual attention patterns within a complex 3D game environment presents a new challenge because the environment is very complex with many rapidly changing conditions; the methods used in previous research cannot be used in such environments. In this paper, we will discuss our exploration seeking a new approach to analyze visual attention patterns within interactive 3D environments.*

Keywords: *Design, Experimentation, Graphics, Perception, Rendering, 3D Game, Visual Attention.*

1. INTRODUCTION

Visual attention has long been an important topic in psychology and cognitive science. Visual attention is gaining more importance in graphics rendering. Recently, results from visual attention research are being adopted by computer graphics research. Due to speed limitations, there has been a movement to use a perception-based rendering approach where the rendering process itself takes into account where the user is most likely looking. Examples include trying to achieve real-time global illumination by concentrating the global illumination calculation on salient parts of the scene or varying the level of detail of terrain rendering depending on salience of locations. Games have achieved a high degree of popularity because of such advances in computer graphics. These techniques are also important because they allowed adoption of game environments in health and training applications. In addition, we believe that research on visual attention can further improve the design of game environments, by decreasing frustration and increasing engagement. Many non-gamers get lost in 3D game environments, or they don't pick up an important item because they don't notice it. A study of visual attention patterns in games can be used to inform designers where to place items in a level or how to choose colors and use other visual tools to stimulate attention and eliminate these problems. While the visual attention process has been extensively researched and is still under much research within psychology and cognitive science, most experiments within these fields employ simplified strictly controlled composition of 2D objects, e.g. combination of lines and blocks. While some researchers have used photographs as well as simple 3D synthetic objects, these environments are no match to game environments. 3D games employ complex 3D architectures, objects, and characters that move and interact in 3D space. While some of the concepts generated by previous visual attention research may transfer to game environments, an experiment is needed to confirm this hypothesis. In this paper, we seek to study the visual attention process within a game environment. We focus on exploring whether visual attention within a 3D game follows the bottom-up or top-down visual attention theories. We also seek to explore the difference between visual attentions patterns exhibited in two game genres:

First person shooters and action-adventure games. Our hypothesis is that different game genres will stimulate different visual attention patterns depending on the activities the player is engaged in. We also

seek to identify saliency of the visual cues present in the environment. Studying visual attention within a 3D environment as the one employed by games is hard. While there are methods established that study visual search through the use of eye tracking techniques within interactive environments, most of these methods focus on web applications, which are 2D, slow-paced (i.e. no rapid movements), and static in terms of their interface (e.g., buttons mostly stay in the same place on the monitor). By comparison a game environment is highly complex using 3D graphics with many moving elements and almost no static interface.

In this paper, in addition to discussing some preliminary results showing visual attention patterns in 3D game environments, we will also present a new method by which eye-tracking data is analyzed within such complex 3D environments. We hope this method can stimulate researchers to begin to formalize methods for evaluating visual attention in 3D complex environments, and thus informing research in psychology and cognitive science as well as enhancing 3D graphics rendering and game level design. In this paper, we first briefly introduce important visual attention theories that are relevant to our work. Then we describe the experiment design that we undertook in detail with game engine like JMONKEY formerly known as JME3 (version 3 is a latest version) and game libraries like OpenGL

2. PREVIOUS WORK

Much research evidence are in favor of a two-component framework for visual attention: bottom-up and top-down. The bottom-up approach identifies several features or properties that subconsciously attract attention, including movement, color, contrast, and brightness. For example, a red rose in a green bush will automatically attract attention because the rose is of a warmer color than the green bush and because of the contrast in color. On the other hand, the top-down mechanism is under voluntary control, and biases the observer towards selecting stimuli that are related to the observer's goal when perceiving a scene. Studies have shown that there is a strong correlation between a viewer's eye movements and the visual task that he is performing.

Several visual features were identified to have a major influence on the bottom-up visual attention process. These features include color, brightness, contrast, orientation, shape, size, lines, and motion. In our study, we concentrate our experiment on testing two features: color contrast and motion. Color has long been accepted as a pre-attentive feature. Search for a target color among homogeneous distracters is efficient as long as the target and distracter colors are not too similar. Because it is so effective, selection by color is common in the design of visual displays. There is evidence for a pre-attentive ability to detect a change in color. D'Zmura and Manga lick found that subjects can search efficiently for a target undergoing a smooth color change when such color is different from (and often complementary to) that of the distracters. When it comes to guiding attention towards more complex targets, motion is a very effective feature. Moving targets pop-out of a field of stationary distracters. Search for random or linear motion among stationary distracters is very efficient. However, search for the absence of motion is less efficient.

High-level top-down driven visual attention covers goal-directed factors; it is essentially a task-dependent visual attention model. Wolfe and Gancarz produced a biologically inspired guided search model that controls the bottom-up features that are relevant to the current task by a top-down mechanism, through varying the weighting of the feature maps. However, it is unclear what the task relevant features are, particularly in the case of action recognition.

3. GRAPHICS AND ANIMATIONS

Graphics are used to render the world in which the player sees. The BSP file format stores information used for rendering levels, while the MD3 file format is used for character animations. Also, the Targa image file format was used for our textures.

Various editors that support these file formats could be used to create content for the game. Also, the level editor that supports BSP generates the information used in algorithms for efficient rendering and collision detection with BSP.

3.1 BSP

The BSP file format stores data structures in contiguous sections of the file. The vertex and texture coordinates used by OpenGL are read into buffers, the rest of the data structures are stored in Haskell lists. The lists are sequentially accessed, and not randomly indexed.

The data structures are assembled into nodes and leaves. These are then assembled into a binary tree.

Visibility tests are performed to reduce the amount of geometry that has to be rendered. Pre-calculated visibility information is stored within the BSP file as bit fields. Bit masking is performed to see if a leaf is potentially visible from the player's location. If a leaf is potentially visible, the bounding box of the leaf is tested to see if it lies within the view frustum.

If a leaf has passed the visibility tests, the leaf Faces are drawn. Each polygon face has pointers that refer to the vertex and texture information stored in the buffer. These pointers are passed to OpenGL, and the vertices the pointer refers to are rendered with OpenGL vertex arrays. Vertex arrays allow for efficient rendering, as they eliminate repeated calls to functions in OpenGL.

3.2 MD3 Animation Format

MD3 is an animation format for Quake. In this game, the

MD3 animation format is used for character animations. Playback of animations is performed with vertex interpolation. Animations are stored as a sequence of key frames. Each key frame is composed of vertices. By interpolating between key frame vertices, the pose of the model between frames can be approximated. The frame approximated by interpolation is rendered between key frames so animations appear to be smoother.

MD3 files store vertex information and transforms. There are separate MD3 files for the legs, body, weapon and head. Shader files specify which textures are applied to the model. An animation cfg file specifies the number of frames and the rate at which each animation is played.

Sections of the model stored in separate md3 files are assembled into a hierarchy. The legs form the root of the hierarchy. This is followed by the torso. Lastly the weapon and the head are placed at the bottom of the hierarchy. Tags are the joints of the model. Each tag has a translation and rotation that has to be applied to join separate sections of model. There are separate tags for every frame of animation. The translation and rotations of a tag affects the position and orientation of sections lower in the hierarchy. There are separate sets of animations for the torso and legs of a model. For example, an attack animation can be played for the torso and running or walking animations can be draw. State information stores what animation is being played, which frames are used for interpolation and the last time the animation was played.

3.3 Textures

Textures are images used to decorate the surface of 3D geometry. Haskell doesn't have libraries for loading images, as it is relatively new language. The Targa image format is a format for describing bitmap images. It supports 24-bit colour and transparencies. It is relatively easier to load compared with JPEG images and many image editors support it.

To load the image, the header of the image file is read to determine the dimensions of the image and the number of bits per pixel. The image is loaded into a temporary buffer. An OpenGL texture object is created and the contents of the buffer are copied into OpenGL's memory.

4. GAME ENGINE – JMONKEY

"jME (jMonkey Engine) is a high performance scene graph based graphics API. jME was built to fulfill the lack of full featured graphics engines written in Java. Using an abstraction layer, it allows any rendering system to be plugged in. Currently, both LWJGL and JOGL are supported. jME is completely open source under the BSD license. You are free to use jME in any way you see fit, hobby or commercial. All we ask is a little footnote (donations are nice too.)"

jMonkey Engine (jME) is a game engine made especially for modern 3D development, as it uses shader technology extensively. jMonkey Engine is written purely in Java and uses LWJGL as its default renderer. OpenGL 2 through OpenGL 4 is fully supported. jMonkey Engine is a community-centric open source project released under the new BSD license. It is used by several commercial game studios and educational institutions. The default jMonkeyEngine 3 download comes readily integrated with an advanced SDK.

4.1 jMonkeyEngine 0.1 - 2.0

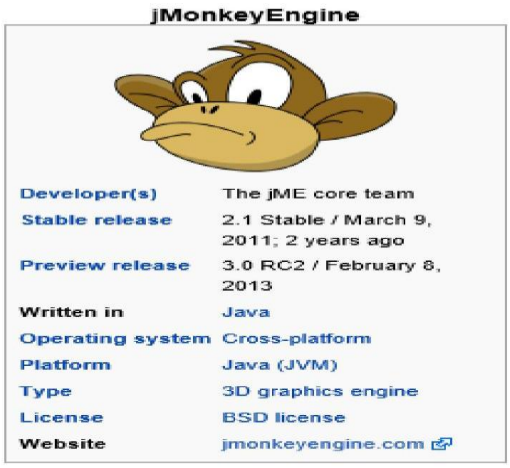
Version 0.1 to 2.0 of jMonkeyEngine marks the time from when the project was first established in 2003, until the last 2.0 version was released in 2008. When the core developers at that time gradually discontinued work on the project throughout the end of 2007 and the beginning of 2008, the 2.0 version had not yet been made officially stable. Regardless, the codebase became adopted for commercial use and the community actively supported the 2.0 version more than any other.

Initial work on jMonkeyEngine was begun by Mark Powell (aka MojoMonkey) as a side project to see if a fully featured graphics API could be written in Java. Much of the early work on the API was inspired by David Eberly's C++ book 3D Game Engine Design. January 2004Mark was joined by Joshua Slack (aka Renanse) and together over the following two years, with the help of other community contributors, a commercially viable API was developed. August 15, 2008 Joshua Slack announces to step back from active development of the jMonkeyEngine. after, the branch was renamed to reflect its "test" status. June 24, 2009 the project sees a new beginning in the official jMonkeyEngine 3.0 branch, initially designed and developed solely by Kirill Vainer.

4.2 jMonkeyEngine 3.0

Since the departure of jME's core developers in late 2008 the codebase remained practically stagnant for several months. The community kept committing patches, but the project was not moving in any clear direction. Version 3.0 started as nothing more than an experiment. The first preview release of jME3 in early 2009 created a lot of buzz in the community, and the majority agreed that this new branch would be the official successor to jME 2.0. From there on all the formalities were sorted out between the previous core developers and the new. The jME core team is now composed of eight committed individuals. April 1, 2009Kirill Vainer "shadowislord" starts a new branch in the official jMonkeyEngine repository and commits the first publicly available code for jMonkeyEngine 3.0. Soon

A 3D Image Created by Jmonkey



The architectures of current game engines limit the chances of a virtual world replicating the real one the limitations and layers of abstraction make it hard to reproduce our world as richly as it needs to be. While the engines are good at estimating behavior and physics, they are not yet completely realistic. Interfacing technology is basic relying primarily on sound and vision; advanced interfaces with the ability to manipulate our senses are not yet available. Virtual worlds are not as immersive as the real world and more research in the areas of taste, smell and balance will lead to far more immersive technology.

Virtual reality systems powered by high-performance game engines can provide reasonable emulations of the real world. The technology needs to mature before it can begin replacing aspects of the real world, but even current technology (demonstrated by VRND) can model our environment to a high degree of detail. No matter at what pace the technology develops, a virtual world while not universal is almost certain. The technologies, approaches and designs used by current and future 3D game engines will prove invaluable in creating a virtual world similar to our own.

Judging by the increase in computing power each year and the similar increase in the power of game engines, there will be a point possibly within ten years when game engines will create photographic reproductions of the real world. Creating a virtual world is merely an extension of this practice but the privacy and security aspects will be the major hurdles, despite providing the largest benefits in future.

5. ALTERNATIVES

The key to a virtual world may actually be creating a new alternate world based on different laws and expectations. Our world exists as it is because we live within its capabilities and limitations. A virtual world would have different limitations - but one based on the real world would need to adhere to the limits of both. An alternate world provides just as many issues as a virtual one. Interfacing with the world poses the same challenges, as does transferring sensation, but there are new issues too: whether we could live in a world designed by us, rather than one we are designed for. Humans have evolved to live within the bounds of the real world – limitations that moderate and control our evolution rather than limit it. Changing these limitations and adding new ones could damage us as a species in this world.

No one lives in a virtual world, and as far as we are aware, no virtual world matches the real world. However, engines capable of serving up a massive world capable of handling many simultaneous users do exist but none use 3D game engines as powerful as those described in this paper. Projects such as the VRND (3.4) demonstrate promise. With 2004-class engines, it will be possible to create very realistic models of the world, combined with accurate visual and aural features. Creating a virtual world is merely an extension of this practice.

6. CONCLUSIONS

The architectures of current game engines limit the chances of a virtual world replicating the real one the limitations and layers of abstraction make it hard to reproduce our world as richly as it needs to be. While the engines are good at estimating behavior and physics, they are not yet completely realistic. Interfacing technology is basic relying primarily on sound and vision; advanced interfaces with the ability to manipulate our senses are not yet available. Virtual worlds are not as immersive as the real world and more research in the areas of taste, smell and balance will lead to far more immersive technology.

Virtual reality systems powered by high-performance game engines can provide reasonable emulations of the real world. The technology needs to mature before it can begin replacing aspects of the real world, but even current technology (demonstrated by VRND) can model our environment to a high degree of detail. No matter at what pace the technology develops, a virtual world while not universal is almost certain. The technologies, approaches and designs used by current and future 3D game engines will prove invaluable in creating a virtual world similar to our own.

Judging by the increase in computing power each year and the similar increase in the power of game engines, there will be a point possibly within ten years when game engines will create photographic reproductions of the real world. Creating a virtual world is merely an extension of this practice but the privacy and security aspects will be the major hurdles, despite providing the largest benefits in future.

ACKNOWLEDGMENT

As mentioned above, there is disagreement about exactly what a game engine is, with sometimes fundamental differences between definitions. Simpson [Simpson 2002] reports on the confusion between game engines and games themselves, as well as the erroneous description of game engines as the game's component for displaying graphics. Game engine definitions that are not limited to individual engine components are often very broad and vague. The main issue appears to be the question of where the boundary lies between the game engine and the game itself. There are signs that a common consensus may be emerging for the definition of a game engine, however. More concrete descriptions have been offered, such as that by Lewis and Jacobson of modules of simulation code that do not directly specify the game's behavior (game logic) or game's environment (level data)".

REFERENCES

[1] Barwood, H. & Failstain, More than th Design Rules" lectureamedevelopers'atg conference http://www.gdconf.com/archive/2002/hal.image_ran.ppt last accessed Nov 2003.

[2] DeLeon V. and Berry R. (Jr.), Bringing VR to the Desktop: Are You Game? IEEE Multimedia, v 7, n 2, p 68-72 (2000)

[3] Henderson E., How Real is Virtual Real, IEE Colloquium Digest, n 454, p 4/1-4/8 (1998)

[4] Hodge L.F. et al, Treating psychological and Physical disorders with VR, IEEE Computer Graphics and Applications,

[5] Kennedy R. S. and Lilienthal M.G., Implications of Balance Disturbances Following Exposure to Virtual Reality Systems, Proceedings of the Virtual Reality Annual International Symposium, p 35-39 (1995)

[6] Maes P., Darrel T., Blumberg B., Pentland S., ALIVE: Artificial Life Interactive Video Environment, Proceedings of the National Conference on Artificial Intelligence, v 2, p 1506 (1994)

[7] Pearson I.D. and Neild, I, Technology Timeline, BT Technology Journal (2002)

[8] Planet Half-Life, Half-Life 2 Screēnshots, http://www.planethalflife.com/halflife2/ screenshots, last accessed 22nd November 2003

[9] PlanetSide, http://www.planetside.com last accessed 21st December 2003

[10] Ultimate Gamer, Half-Life Screenshots, http://www.ultimate-gamer.com/halflife/images/ last accessed 22nd November 2003