Integrating Advanced GLSL Shading and XML Agents into a Learning-Oriented 3D Engine

Edgar Velázquez-Armendáriz, Erik Millán
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México
Atizapán de Zaragoza, Estado de México, 90210, México
E-mail: {A00464175, emillan}@itesm.mx

Abstract

Most of the existing 3D engines are overwhelmingly complex and do not integrate support for virtual characters. We have developed a teaching oriented 3D engine with support for such tasks as model loading and setup, shadows, level of detail as well as advanced shading techniques using the OpenGL Shading Language (GLSL), developed following Object-Oriented techniques and built upon standard open source components. We have also extended previous work by seamlessly incorporating XML driven crowds which can interact with the elements of the environment. The resulting system is a highly capable, yet easy to learn and use 3D engine which may be used for students in Computer Graphics to quickly build interactive applications and to provide a framework to begin using specialized shading techniques with GLSL. The engine also serves as a motivating development environment for creating visually attractive crowds of agents, aimed at students of introductory Artificial Intelligence courses.

1. Introduction

Computer Graphics has become a very popular topic in the recent years. With the advent of commodity hardware, which is capable of rendering real time graphics that rival the quality of production level software renderers, a great amount of college students show interest in the area. Those who are learning the principles of graphics, however, must take a big leap in moving from the simple and isolated examples that show only one feature at a time to the more complete systems that integrate several of such functionalities together.

To make a more advanced application, users might choose to use one of the available 3D engines, but their use is cumbersome by those less exposed to the graphics programming, given that those engines include a myriad of functionalities, exposed over hundreds or even thousands of API calls. Other option for the initiated graphics practitioners would be to write their own engine. That approach would add the inherent difficulties of developing an entire system architecture to the challenges of implementing the tasks of the main application.

The complexity of both approaches causes that students implementing a project cannot incorporate more advanced features as shadows and custom shader programs [19]. The industry is moving away from the fixed graphics pipelines to custom functionality programmed in high level languages as GLSL, HLSL or Cg, therefore it is valuable to allow them a quick peak into that technology.

Either implementation strategy misses entirely support for artificial intelligence characteristics, which are widely used in interactive applications as games. The user would have to create hand-coded characters. And from the students of AI courses perspective, when developing intelligent agents they face not only the problems of implementing their agent i.e., using the subsumption architecture, but also to create a proper interface to display those agents.

To overcome those limitations, we propose a 3D engine aimed both at AI and CG students simple enough to be used with a small learning curve and that still allows them to use more interesting features as the GLSL shaders. It also integrates the work by Rudomín and Millán in [16], which allows the creation of virtual characters and crowds using images and XML files.

In the following section of this paper, we will make a brief review of the existing related work. In Section 3 we will review the architecture and capabilities of the proposed engine. In Section 4 we will analyze the interactive XML features of the system. Section 5 presents the obtained results and in Section 6 will state our conclusions and future work.

2. Related work

The Generic Rendering System of Döllner and Hinrichs [5], encapsulates different rendering components in an object oriented framework using shapes, attributes, handlers, techniques and engines as the main components. Their system can use different components such as OpenGL, Renderman or POVRay to perform different tasks. This system is able to perform real time effects like Phong shading using multiple rendering passes. Their architecture allows having a platform independent system core, hence it runs using MFC, Qt, or Tcl-Tk for the GUI on several different systems.

Among the several available open source 3D engines, OGRE [1] is one of the best regarded ones. It has extensive support for loading models, textures, animations and materials, as well as the ability to use custom shaders using both high level languages and assembly instructions. It is a multiplatform development toolkit, which runs in Windows, Linux or Macintosh computers. However, the produced applications are intended for high-end computers.

Irrlicht [7] by Gebhardt is another open source engine with increasing support from the community. It is oriented towards high performance on low-end machines, supporting many of the features of OGRE. It has just recently added the capability to use GLSL shaders.

However, none of these engines provides a straightforward way to turn objects into interactive agents. Such functionality must be implemented, if desired, on top of the provided API.

Research has also being made in the area of systems for education. The system Alice [4] developed by Conway, Audia, Burnette et al, is a rendering system intended for novices. They used semantics similar to the LOGO programming language with encouraging results, using notations closer to the user than those which are more technical oriented.

Works in the area of education by Sung and Shirley [19] have found that a popular choice for students projects in courses following a top-down approach is an interactive graphics application, which allows the user to update the state of the application interactively.

3. System's architecture

In this section we will explain the design architecture of our system. The core GLM++ library will be explained, as well as the rendering capabilities built into the engine.

3.1. System's object layout

The main elements in the system are the 3D objects and the cameras. Both elements are built as a C++ class hierarchy which is illustrated in Figure 1, using polymorphism an inheritance to build increasingly specialized classes.

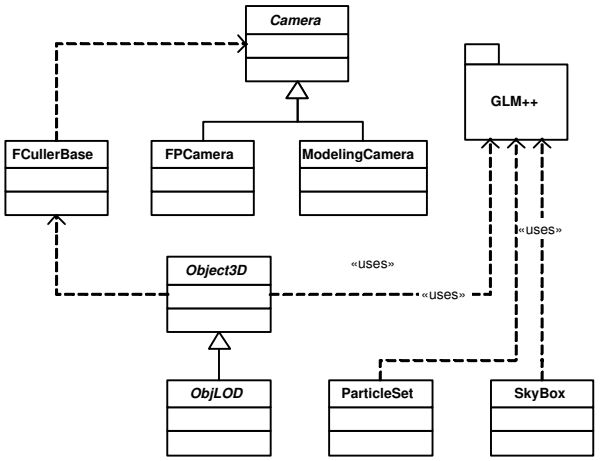

Figure 1. Main engine architecture.

The cameras have an interface to allow interaction by reacting to user input, such as keyboard strokes or mouse movements. They are also able to compute the distance from the camera to any given object. One of the available cameras mimics the views provided by a rendering system, being specifically tied to the movements of the perspective cameras of Maya [3]. The other camera provides a first person perspective that allows the users an immersive view in the virtual world that is being developed, and responses in the same way of popular First Person Shooter games.

The base 3D object class provides a way to draw, scale, move and rotate an object through simple function calls. The object may not be drawn if it is not visible by using a frustum culler object, which receives perspective information from the current camera and the object that is being queried. Specialized objects that support level of detail use the distance from the camera data to calculate a quality factor in the range $[0, 1]$, where 0 means to avoid drawing and 1 requests to use the full detail of the mesh.

Collision detection among objects is performed using the open source library Coldet [8], which was improved to use SSE instructions for its calculations. It may be used to trace collision rays, which may be employed in such actions as shoots or object picking. A simple particle engine has also been added to visualize these collisions. The interaction between these components can be seen in Figure 2.

A base skybox class allows the rendering of panoramic objects, for which the users only need to provide the required textures. Finally, all the objects in the world are drawn using the active camera, through which the user inputs are interpreted. The engine may draw also auxiliary shapes such as the bounding spheres used for the frustum

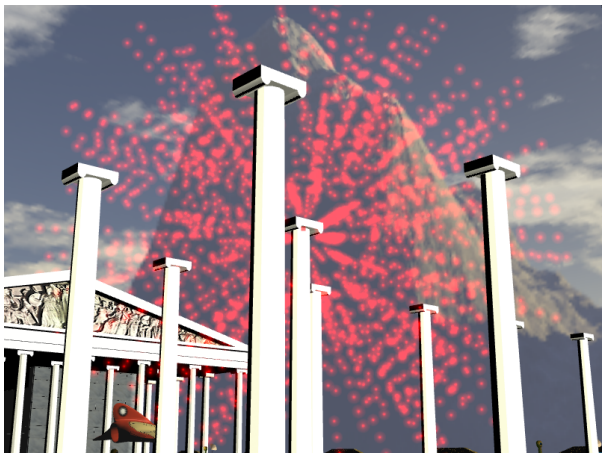

Figure 2. Collision detection and particles.

culling or the wireframe to provide a better understanding of the underlying functionality of the engine.

Computer Science students are more likely to use different operating systems, as Windows or Linux, than the average user. For this reason the framework is built around cross platform open source libraries, such as *xerces* for XML parsing, *GLEW* for OpenGL extensions management, *freeglut* for the windows system, *fmod* for sound support and *libpng* for the textures, using a standard dialect of C++. This way, the source code can be compliled for both Windows and Linux, using the gcc 3.x or Visual C++ compilers, with only some spare lines of non portable code, most of them for the platform dependent PBuffers, required for the shadow maps, which will be explained in Section 3.3. Code readability and maintenance is greatly improved this way.

3.2. GLM++ Library

Although the class hierarchy provides a natural interface to interact with the elements in the world, it does not provide by itself a way to draw a model or load a texture. To perform such tasks we created the GLM++ library, which extends the original *glm* libray by Nate Robins [15].

The original library allows loading models from Wavefront OBJ files, reading their material attributes and drawing them using standard OpenGL calls. It is also capable of performing correction operations to the normals of the objects that allows rendering the objects either with facet normals or as smooth objects.

This functionality was extended to create the vectors required to make the conversion from object space to tangent space. That information is required to use the rendering techniques described in Section 3.3. The library is able to initialize the loaded object for future collision detection.

Textures are supported through objects that encapsulate the process of texture management and loading. The supported formats include PGM, BMP and PNG, using 8, 24 or 32 bit depth. Further formats may be added by extending the corresponding class hierarchy. The library may also create normal maps from 8 bit height maps. In this way, the users can load the same objects, textures and bump maps they have designed in modeling applications and use them into the engine.

Support for GLSL [10] shaders received special attention. This standard language, integrated in OpenGL 2.0 [18] allows the students to interact with custom shader programs, a mayor trend in the industry, but whose setup and implementation details make them too complex for the early practitioners. The shader class performs the tasks of loading the shader programs from the source files, enable and disable their use during program execution and giving to them the uniform parameters they might need.

The shaders can either be incorporated into the 3D objects, so that each object is rendered using a different program, or set within the environment, to render all elements with the same shader. A fully functional shader, described in the following section, is provided with the application to serve as a foundation to the student to begin using GLSL programs. These abilities will allow the user to focus on the logic of the shader program instead of the setup details.

3.3. Rendering features

Shadows are a very important way to add detail and provide volume information within a 3D environment, but they are not usually implemented in the students' projects mostly due to time constraints. The engine provides this feature through shadow maps for a single point light. As this is an image space technique, it does not require knowledge about the object geometry; hence any object in the scene will cast and receive shadows, including self shadowing.

To provide a visually attractive environment, and to encourage further experimentation by the students, a full shader program in GLSL is provided, supporting per pixel lighting, normal mapping, bump mapping and percentage closer filtered shadows, with modulated umbra color.

To perform the normal mapping, the shader is fed with the tangent vectors [12] [17] provided by GLM++, and performs the lighting in tangent space. The lights in OpenGL are defined in eye space, therefore, given the model view transformation matrix M, the normal vector $\vec{N}$, the tangent vector $\vec{T}$ and the binormal $\vec{B}$ defined as $\vec{N} \times \vec{T}$, all of them being column major vectors, the transformation of an eye space vector $\vec{v}$ to the tangent space vector $\vec{v'}$ yields (1):

$$\vec{v'} = ([\ \vec{T} \quad \vec{B} \quad \vec{N}\ ] \circ M^{-1})(\vec{v}) \tag{1}$$

The color of the fragments of a given object is then calculated using the Blinn-Phong lighting equation [6].

Figure 3. Original model (left), its normal map (center) and the final result using normal mapping (right).

Figure 4. Hardware shadows (left) and our custom shaded shadows (right).

$$I_{out} = I_{light} k_d \max(0, \vec{N} \cdot \vec{L}) + I_{light} k_s \max(0, \vec{N} \cdot \vec{H})^n \quad (2)$$

where I_{light} is the color of the light, k_d is the diffuse color, k_s the specular color and n the specular exponent. $\vec{L}$ is the normalized vector to the light source and $\vec{H}$ is the normalized half-way vector defined by (3):

$$\vec{H} = \frac{\vec{L} + \vec{V}}{|\vec{L} + \vec{V}|} \quad (3)$$

The results of this shading technique are illustrated in Figure 3. The final color of the fragment is obtained after applying to I_{out} the percentage closer filtered shadow contribution factor s.

$$I_{frag} = I_{amb} + \frac{1}{2}(1 + s) I_{out} \quad (4)$$

Where I_{amb} is the ambient light contribution. The shadow factor s is within the closed interval $[0, 1]$, where 0 represents a fragment in complete shadow and 1 corresponds to a completely lit fragment and s is the average of the unfiltered binary valued shadow contribution s'_i of the four surrounding fragments. As seen in Figure 4, this way of calculating the final color results in umbra areas where diminished color is seen, like occurs in the real world, instead of the completely black areas of the predefined hardware shadows.

The rendering mode can be alternated between the fixed OpenGL engine and the custom shader at any moment during execution. The shadows may also be activated and deactivated at any time, providing the user an easy way to evaluate the differences among the different shading styles. One of the benefits of using this pixel shader is that the scene requires only one pass to draw it, and one extra colorless pass to update the depth texture needed for the shadow map.

Taking the previous rendering elements as a foundation, the students are able to further experiment with other shading techniques, such as cel shading, parallax mapping or the Fresnel effect.

4. Interactive XML crowds

XML based agents have also been integrated into the engine. As in the work purposed by Rudomín and Millán in [16], crowds of virtual characters which interact with the environment can be created, making different agents classes, types and behaviors through an XML file, using either finite state machines or a subsumption architecture. The 3D models for the characters are loaded from standard Quake 2 files with animation. Static meshes in OBJ and 3DS file formats are supported as well.

These interactive agents can interact with an arbitrary environment using image based collision and height maps. This approach has the advantages of having a very small processing overhead because the collision between agents and the environment requires only a texture read, and it does not require knowledge about the underlying geometry. Using height maps, the agents can move with better realism within a non planar surface. The virtual characters engine may either render a terrain coherent with the provided height map or just displace the elements according to the data read in the map. As with the collision map, each height query needs a single texture lookup.

The seamless integration of the agents with the rendering environment allows them to be shaded using custom GLSL programs, and they also cast and receive shadows, as seen

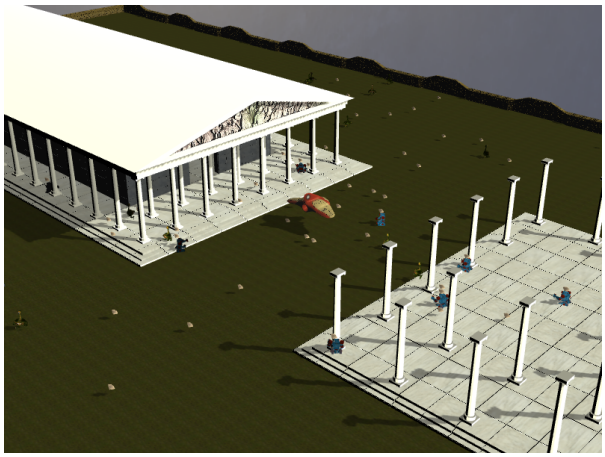

Figure 5. Mars Explorer agents.

Figure 6. Predators and preys agents.

in Figure 5. These effects may be used in order to achieve a highly interactive application which at the same time is visually attractive.

This approach benefits both computer graphics and artificial intelligence students. By simply using plain text XML files, the graphics students will be able to add interactive elements to their applications, and AI students will have a 3D environment to develop and test their algorithms. Constructive learning, which is pursued with the project based courses, evoke the learners spontaneous interest [20], and a platform like the one we are presenting helps achieving this goal.

5. Results

We have built an application using the engine as proof of concept. It consists in an Acropolis with a temple, a statue of Pallas Athena, a limiting wall, a hypostyle hall and the polygon ship from [3] as a reference object. The walls, floors and the ship are bump mapped; the statue has a normal map generated from a high resolution model. Two different XML crowd configurations were created, one that mimics the Mars explorers, seen in Figure 5, and other of predators and preys, observed in Figure 6. Each one uses different models, but interacts with the same Acropolis unmodified environment.

The application proved to run without code changes in Windows XP and Linux, using the Fedora Core 2 distribution. It was executed in very different machines, a legacy 1.13 GHz Pentium III laptop with 512 MB in RAM and a 32 MB Nvidia Geforce 2 Go GPU, a 2.8 GHz Pentium 4 desktop with 512 MB in RAM and a 128 MB Nvidia Geforce FX 5200 GPU, and a high end 3.4 GHz Pentium 4 desktop with 1 GB in RAM and a 256 MB Nvidia Geforce 6800GT GPU.

The legacy laptop was only able to use the fixed rendering pipeline without shadows, because the hardware does not support them. Both desktop systems supported both the hardware integrated shadows and the custom shader programs. The limited number of fragment pipelines in the Geforce FX 5200 caused the application to noticeably slow down when using all the rendering features, while the newer Geforce 6800GT did not show that behavior and performed considerably faster.

The obtained rendering quality is very high as shown by Figure 7. The system is able to generate such images more than 200 times faster than the Maya software renderer using the same computer. The usage of normal mapping allows displaying excellent quality images without requiring extremely detailed meshes.

Both of the virtual characters configurations tested integrated transparently with the environment, moving only within the limits of the Acropolis while they followed rules such as not going through the columns or the temples walls, as well as keeping at the appropriate height at all times. The realism was conspicuously enhanced with the shadows projected by all of the elements, especially with the interactive agents moving around the scene.

6. Conclusions

We have presented a 3D engine geared toward students implementing both computer graphics and artificial intelligence projects, as a simple to use and at the same time easy to use base framework. The achieved visual quality motivates further exploration of the capabilities of the custom shading programs using GLSL and encourages the creation of autonomous crowds of agents that interact in an appeal-

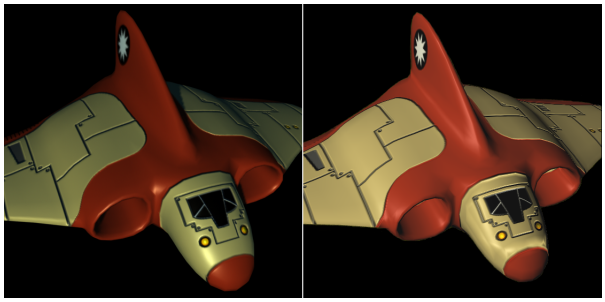

Figure 7. Polygon ship rendered in Maya (left) and in our engine (right).

ing environment. The per pixel shading has performance penalties but those are becoming less apparent with the current generation of GPUs, which provides a great number of fragment pipelines to perform this kind of shading operations better. While in some systems it may not be possible to visualize all of the included features due to hardware limitations, the portability of this library allows the use of a variety of hardware platforms for the supported functionality.

Future work would expand the use of XML files to create complex object from several meshes and with as much fine grained detail as a manually written subclass. To increase the portability of the engine, the PBuffers would be substituted by the newly supported framebuffer objects of OpenGL, which allows direct Render To Texture (RTT) regardless of the underlying windows system.

Another interesting area for future work would be to add communication capabilities between agents to allow for planning and cooperation strategies, using the descriptive XML files to specify them. Networking support would permit interaction among agents running in different environments or even heterogeneous systems, further improving the complexity of the behaviours that can be developed using the XML system.

Acknowledgements

The authors thank Luis Antonio Landgrave Romero of ITESM Campus Estado de México for his support in developing the GLM++ library and integrating the collision detection and sound support into the engine.

References

[1] Ogre 3d: Open source graphics engine. http://www.ogre3d.org.

[2] Nvidia sdk, 2005. http://http://developer.nvidia.com/object/sdk_home.html.

[3] Alias. *Learning Maya 6 — Foundation*. Sybex, 2004.

[4] M. Conway, S. Audia, T. Burnette, D. Cosgrove, and K. Christiansen. Alice: lessons learned from building a 3d system for novices. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 486–493, New York, NY, USA, 2000. ACM Press.

[5] J. Döllner and K. Hinrichs. A generic rendering system. *IEEE Transactions on Visualization and Computer Graphics*, 8(2):99–118, 2002.

[6] C. Everitt. Mathematics of per-pixel lighting, aug 2001. http://developer.nvidia.com/object/mathematicsofperpixellighting.html.

[7] N. Gebhardt. Irrlicht engine. http://irrlicht.sourceforge.net.

[8] A. Geva. Coldet - free 3d collision detection library, 2000. http://photoneffect.com/coldet/.

[9] J. Kessenich. *Features of the OpenGL Shading Language*. 3Dlabs Inc. Ltd., may 2005.

[10] J. Kessenich, D. Baldwin, and R. Rost. *The OpenGL Shading Language*, apr 2004.

[11] P. Kipfer, M. Segal, and R. Westermann. Uberflow: a gpu-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, New York, NY, USA, 2004. ACM Press.

[12] E. Lengyel. *Mathematics for 3D Game Programming & Computer Graphics*. Delmar Thomson Learning, "second" edition, 2003.

[13] E. Lindholm, M. J. Kligard, and H. Moreton. A user-programmable vertex engine. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 149–158, New York, NY, USA, 2001. ACM Press.

[14] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, "first" edition, 2002.

[15] N. Robins. Nate robins - opengl. http://www.xmission.com/~nate/opengl.html.

[16] I. Rudomín and E. Millán. Xml scripting and images for specifying behavior of virtual characters and crowds. In *CASA '04: 17th International Conference on Computer Animation and Social Agents*, Geneva, Switzerland, jul 2004.

[17] G. Schrröcker. Hardware accelerated per-pixel shading. Technical report, Graz University of Technology, 2002.

[18] M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification*, oct 2004.

[19] K. Sung and P. Shirley. A top-down approach to teaching introductory computer graphics. In *GRAPH '03: Educators program from the 30th annual conference on Computer graphics and interactive techniques*, pages 1–4, New York, NY, USA, 2003. ACM Press.

[20] G. Taxén. Teaching computer graphics constructively. In *GRAPH '03: Educators program from the 30th annual conference on Computer graphics and interactive techniques*, pages 1–4, New York, NY, USA, 2003. ACM Press.

[21] L. Wilkens. A multi-api course in computer graphics. In *CCSC '01: Proceedings of the sixth annual CCSC northeastern conference on The journal of computing in small colleges*, pages 66–73, , USA, 2001. Consortium for Computing Sciences in Colleges.