DESIGN OF AN EXTENDED 3D GRAPHICS ENGINE

M Sazzad Karim, Muhammad Mahmud Hasan and Emdad Ahmed
Department of Computer Science & Engineering, North South University
12 Kemal Ataturk Avenue, Banani, Dhaka-1213, Bangladesh
Email: skremon@yahoo.com, mmahmudhasan@yahoo.com, emdad@northsouth.edu

ABSTRACT

3D Applications such as games, visualization tools and Virtual Environment renderers use an internal system to manage tasks and to actually render the graphics in the display system. While the intrinsic functionalities and features can be very complex, the high level design of the system should be simple, robust and scalable. This paper presents such a high level design of a scalable and extensible design for a 3D graphics engine that follows the concepts of a general operating system.

1. INTRODUCTION

Computer Graphics technology is evolving very rapidly and impressively with the development of graphics hardware and software, giving the applications more power and capabilities. However the graphics applications need to have a robust, scalable and extensible design to integrate latest rendering and shading systems. Here we present a high-level design of a graphics engine system with some extended features: a Kernel and Rendering system. The objective of such design is to keep a robust, stable internal system for every application so that the engine remains unchanged, while the applications may vary in implementation. This work is extended from our prior work on scene management in 3D graphics engines [1].

2. DESIGN PHILOSOPHY

The design of the graphics engine is very much similar to an operating system we use. The application system of the engine shares some similarities that can be compared with an operating system’s basic design. Firstly, an operating system has a task pool that is driven by the kernel. Secondly, there are some resource managers for the operating system that manage the resources needed for the operating system and the running tasks. And finally, there is a Rendering System capable of drawing graphics to the screen. In addition, operating system must have a memory manager for all tasks and resources using system memory.

In this paper we will stress on the design of the Kernel and the Rendering system of the engine.

2.1 Resource and Memory Management

Resource management means the allocation and deallocation of different kind of resources using memory and devices among the tasks. Mesh geometry data, texture data, file data are example of resources.

The engine has a base memory manager that can allocate the requested memory and can keep track of the memory block. It can de-allocate a memory block as required. It also has garbage collection functionality for the block that is not needed anymore.

2.2 Engine Core

It has a task pool that can hold all the tasks needed at a time. Every task is sorted on a priority basis similar to priority based scheduler. The tasks are scheduled in the task pool to be executed by the application. Figure 1 shows the basic block diagram of the engine core.

3. THE ENGINE KERNEL

Kernel is the heart of the engine. The kernel is responsible for managing execution of the tasks.
Kernel is also used for discrete event simulation [2]; however, we are using it for task management.

3.1 Mechanism

The kernel performs functions like adding and deleting tasks from the task pool. Whenever the application requires a task for execution, it can create the task and can request the kernel to add the task in the task pool for the next phase of execution. The kernel can suspend or resume any task if required. On the other hand, after the execution of the whole application, kernel can terminate all the tasks to free up the memory with the help of the memory manager. The kernel functionality is shown in Figure 2.

In our design, we derived the kernel from a generic singleton. Before we go further, we need a brief understanding of singleton.

3.2.1 Singleton

Singleton is simply an object that can have only one instance in an application’s lifetime. A singleton is implemented as a template class, from which we can derive any type of objects i.e. kernel, setting manager, resource managers etc. The singleton helps us to derive the object as a unique body in the whole object hierarchy. The importance of a singleton is that, when we derived our kernel from singleton, there is no chance of making another instance of kernel in the same application by mistake. One of the most popular implementations appeared in [3].

3.2 Design

The kernel maintains two lists tasks. One list is used for active tasks in the pool for execution. The other list contains suspended or paused tasks. And the kernel has the following functions: Add Task, Delete Task, Resume/Suspend Task & Terminate All tasks.

4. TASK MANAGEMENT IN KERNEL

The task manager executes the tasks in the application. Most of the time tasks are sorted in priority basis. So, each task is executed based on the given priority. In priority concern, to remove the hassle in the execution phase, we set priorities while adding the task in the task pool and sort the list. The kernel adds a task in the task pool in sorted order as showed in Figure 3.

4.1 Mechanism

The task management system is designed in a hierarchical, object oriented manner. Every task in the engine is derived from a basic task object that has the generic function for every task. To give an auto garbage collection feature in each task, the basic task object is also derived from the basic object class. The engine has some basic, essential tasks: Video Update, Input, Time and Render. The application will probably use a sound system; hence a Sound task will accompany the list of tasks for any multimedia application.

The generic task class has some basic function like starting the task, stopping the task, suspend and resuming the task. Moreover in every phase of execution, the task is updated by the kernel by the update function. These basic virtual functions must be implemented in every task derived from the basic task (Figure 4). This is why we made these functions
Figure 4: An Example of Engine’s Class Hierarchy

Figure 5: Video task derivation from generic task. *pure virtual* to make sure about their existence (Figure: 5), For example in the case of video updating, a Video Task will be derived from the generic task and it would implement a customized update function. In the execution phase kernel calls the update function of the Video Task. We place the swap buffering function in the update function to update the new frame.

4.2 An Example: A Game Application
A game application using the engine should have a Game Task in addition. The Game Task would be derived from the generic task and added to the task pool. In every phase of the traversal of the task list, kernel executes the game task and updates its internal application states. The Game Task will update all its entities, perform collision, perform A.I. calculation and add geometry to the Render System. The rendering system (also placed as a task in the task pool) draws the geometry according to scene graph and current game state. And finally the Video Task updates the frame buffer (by double-buffering).

5. GRAPHICS RENDERING SYSTEM
Every graphics engine comprises one or more rendering system to manage and draw the graphics in the screen. In this case the rendering system can consists as one or more task in the task pool. The purpose of the rendering system is to render geometry into the screen using proper images and Shaders. The system also manage geometry data, optimize the data for fast rendering according to hardware limitations and render the data using hardware device i.e. Graphics Processing Unit (GPU) in the system. Typically the renderer is designed to be pluggable to our engine and will use a low level API like OpenGL or DirectX. The renderer mainly deals with Shaders and Geometry and should use a proper scene management system.

5.1 Scene Management
The scene manager manages all objects or entities in the scene. This contains preparing the objects to perform collisions, animating them and send geometry to the Graphics Renderer. The scene manager typically uses a hierarchical scene-graph to represent objects. Details of scene management system is beyond the scope of this paper, hence we recommend interested readers using efficient object management and visibility determination systems discussed in our prior work [1] and also [4] [5].

5.2 Shader System
Shaders are material properties that can be applied to each vertex and pixel/fragment of geometry passed through the rendering pipeline. The simplest Shader could be a color applied to a group of vertices or pixels. However Shaders can contain colors, textures, lights applied to every vertex or pixel. Recently the advent of programmable GPUs let us manipulate the vertex and fragment operations in the GPU.

For implementing and integrating Shader systems some high level *shading languages* are available. We recommend using CG [6] and OpenGL Shading Language (GLSL) [7] when it becomes available. These systems abstract the complex assembly implementations and give programmers an intuitive language interface to write Shaders.

5.3 The Renderer
The Renderer will contain lists of Face Groups (FG) or Geometry Chunks. These are basically a group of faces that can be drawn using one Shader pass.
5.3.1 Design
Renderer contains a list of FGs and functions to add FGs. The class design is shown in Figure 8.
The render algorithm goes through each FG is its list and draws its containing geometry using the proper Shader:

\[
\text{Renderer} (\text{RenderList}) \\
\quad \text{For each FaceGroup fg in RenderList} \\
\quad \quad \text{Apply Transformation} \\
\quad \quad \text{Bind VBO data} \\
\quad \quad \text{Apply Shader} \\
\quad \quad \text{Draw the Faces}
\]

The benefit of the discussed engine design comes from development of applications without changing the engine core. While the internal implementation might differ, it is based on the same design; hence the system remains same for all applications. Any 3D Graphics Engine package can be designed using our proposed model for better scalability of the system and easier development of applications.

REFERENCES