APPLICATION OF A MULTIPLAYER COMPUTER GAMING PARADIGM TO ENGINEERING DESIGN TOOLS
Jeremiah Shepherd, Jian Shi, Tianlei Zhang, Jijun Tang, Roger A. Dougal
University of South Carolina
Columbia, SC 29208

Abstract
Infusion of gaming concepts, ideas, and paradigms into engineering tools can greatly enhance the features of the engineering tool. We have taken concepts from the world of Massively Multiplayer Online (MMO) games and have preliminarily implemented three of them, client-server methods, dynamic interference testing, and multicore computing, into the Virtual Test Bed environment which supports design and analysis of multidisciplinary dynamic systems.

1. Introduction
Massively multiplayer computer games offer lessons in diverse areas, from technical to social, to the designers of engineering tools. We describe here how gaming concepts such as network-centric multiplayer interactions are being introduced into the Virtual Test Bed (VTB) simulation and design environment in order to improve the engineering tool so that it can concurrently involve an entire cohort of engineers, covering many disciplines, into one interactive design environment that allows each engineer to immediately observe the effects of system details or changes introduced by his or her colleagues. The design environment may then become competitive, with each engineer attempting to satisfy his or her own design goals, which often are contrary to those of other engineers. Rapid iteration of a design, guided by human understanding and intuition of the system behavior as revealed by interactive simulations, is expected to permit more rapid convergence to a satisfactory, perhaps even optimal, design.

Our objectives include both exploiting and developing the human intuition that is necessary to rapidly drive a design to a suitable end solution. This intuition can be developed through extensive practice, which can be motivated by using a game-like environment to repeatedly rehearse the process of rapid design iteration in the same way that other “games” (flight simulators, synthetic environments for war games, etc) are now used to develop skills in other professions. Thus the engineering tool can serve as a training aid and the gaming process can enhance engineering abilities. Our work towards these ends has eight formal objectives; in this paper, we will discuss only three, which are enumerated later in this section. This initial version of the software will later be used to more rigorously explore how gaming methods can be applied in the engineering design process.

Our first step towards infusion of this gaming paradigm into an engineering design tool entailed modifying the VTB Pro simulation engine and its associated visualization environment to look like an automobile racing game. Competitors can define their vehicles, including the characteristics of the power storage and conversion subsystems, and then race them against each other in a network-based environment. Development of this game required inserting methods for client-server based interactions between simulation objects, exploitation of multicore computing strategies, and methods for interference detection, which addressed the following three of our formal objectives:

1) New methods for collaboration: The new environment will invite, or perhaps demand, new methods for collaboration. Since changes to one system may immediately impact performance of other systems, it is necessary to communicate these changes to others. Excessive communication will become noise, yet inadequate communication about a rapidly evolving design could lead to confusion or even chaos.

2) Methods for interference testing: A multiuser environment invites “collisions” of many sorts. We have focused first on detecting collisions between parts of the designed system in three-dimensional physical space, then we will later extend the methods to include more abstract concepts of collisions such as infringement of design standards, or interferences between design objectives.

3) Methods to exploit the next generations of multi-core computer processors. We are on the verge of an explosion in the number of computing cores available on computer processors. This opens new opportunities to exploit highly-parallel computing capabilities based on methods for decomposing or decoupling naturally-coupled systems and based on methods for automatic assignment of sub-problems to the available computing resources.

2. Client-Server Architecture
Perhaps the most significant part of a massively multiplayer simulation environment is the means for interaction between multiple users. Direct interaction of the type needed here was not supported by VTB Pro, so we developed an independent networking subsystem to handle this via internet communications. This subsystem allows the users to connect to other users via a server, and to send information pertaining to a local simulation executing on their own computer to other networked computers. The client-server architecture is, of course, not new and has most recently
been adopted by Massively Multiplayer Online (MMO) games such as Second Life. In Second Life, a user’s interactions with other players are managed by a server cluster that sends pertinent information from any one player to all other players whose simulated worlds depend on the player’s actions. In our first implementation we use only a single server rather than a cluster, but the concept remains the same; clients send information to the server and that information is broadcasted to those other clients whose local simulations depend on that information.

Fig 1 shows the architecture of the distributed multi-user “gaming” environment for VTB. Here, VTB Pro operates through a façade in simulation server mode subservient to the application. VTB Pro thus takes commands both from the game player (via a game controller, in this case an Xbox 360 [1] controller) and from the game application client. Results from the simulation server are shuttled through the local game application before being forwarded via a COM object to the visualization engine, along with data from other instances of the distributed game application. While normally VTB Pro would send such simulation results directly to the (local) visualization engine, we instead route these through the game application client in order to integrate results from the local system with those from other game players. This also permits filtering of the data passed to the local version of the visualizer so that it displays only information pertinent to the local user’s view of the system.

![Fig 1. The relationship among server and clients.](image)

The networking subsystem has two vital components -- the server component and the client component.

The server component provides a central point for connections between users. It receives information from the connected users, and then interprets the information using protocols we have developed. The decoded information pertains to the current state of a player’s car in the world, such as its location, orientation, and speed. That information is then taken, repackaged, and sent to all of the other users who are connected to the server, along with any other appropriate information such as collision forces.

The second major part of the networking subsystem is the client component. The client component sends information about the local subsystem to the server and accepts information from the server about other vehicles in the immediate vicinity. Messages from the server are decoded by the client and then sent to one of two places: Most messages just update the position and orientation data of other cars and will be sent only to the visualization engine for rendering animations of the vehicles. Other messages, though, may contain information about collision events between vehicles. This information is forwarded to the pertinent VTB Pro simulation engine so that it can properly compute the physics of the collision.

The networking tool was developed using Microsoft .NET Sockets [2] across the TCP/IP communication protocol. TCP/IP was selected since it assures that all data will arrive intact and at an optimal time. Later we will consider changing this to the UDP networking protocol since that will support a greater number of user connections. By using the .NET Sockets, we took advantage of event driven handling of actions relating to the server. This allowed the clients to connect, send, and receive data at the moment it happened. Using this event driven technology avoids the difficulties and potential headaches of using a polling system to send and receive data.

All of the dynamics for an individual vehicle are embodied in its own VTB Pro system module that describes the dynamics of all of the electrical and mechanical parts of the vehicle. This module can be easily substituted by alternate configurations to define different vehicles or to improve the fidelity of the model of any vehicle. Wrapping up the vehicle as a module with well-defined entry points simplifies standardization of the interaction points between individual vehicles (for interference (collision) detection, and so forth), between vehicles and controllers, or between vehicles and the visualization engine. This makes it possible for different users to define different vehicles, yet still interact through the same controller and with other competitors in the standard game environment.

User interaction with the game environment is via an Xbox 360 controller. While any game controller hardware can serve as the user input device, we specifically chose the Xbox 360 controller for this first implementation. This controller permits the game player to adjust any simulation parameters, but in the particular instance, especially the vehicle acceleration, deceleration, and steering.

We have used the native visualization environment of VTB Pro for the game. It provides mechanisms to define the visual elements of a vehicle and to animate them based on simulation data streaming from VTB Pro. To simplify user interaction, eventually a player will also build her vehicle in a “3D visual sandbox”, as compared to using the current schematic editor. In either case, though, it is assembled from a set of standard elements that can be individually parameterized. For the time being, the vehicle model must be defined directly in the VTB Pro schematic editor environment.

Another major part of the networking subsystem is a specialized protocol that enables clients and servers to decode
and understand the messages. The protocol uses the following form:

\[
\text{<msg type> <user #> <msg data>}
\]

Once an entity (client or server) receives a message, it will first extract the message type and user ID, and then it will parse the data based on the specific type. This procedure is done with the assist of Factory Pattern, a design pattern widely used in network communication. For example, if a message is of type “MOVE”, the data it expects are the spatial and angular coordinates X, Y, Z, roll, pitch, and yaw.

Although the above protocol works well for the demo racing game, eventually a multi-user VTB system will need to transfer more complexed messages. We plan to develop a new protocol based on XML (Extensible Markup Language [3]) or other scripting languages so that structured data can be transferred over the Internet, and new types of data can be easily added by a user since these languages are human-legible and easy to understand.

3. Collision Detection and Response

Neither the simulation engine nor the visualization environment have historically handled interference detection in a generic way, yet this is an important consideration in this sort of multiuser environment, and in the process of engineering design in general. In our racing game, we have identified two different types of interferences that require different handling – collisions between vehicles, and interactions between a vehicle and the terrain. These are differentiated from each other and are handled differently because collisions between two vehicles should be rare and must be detected and analyzed and information must be shared between client stations, whereas interactions between a vehicle and the terrain are continuous and can be handled entirely locally.

Collisions between vehicles are handled in the server and the resulted forces and torques are broadcasted to those entities involved in the collision through the Server-Client connection. In the future, collisions will be handled inside the VTB Pro simulation engine by using a collision manager which takes location and orientation as input and provides forces and torques as outputs. Interactions between vehicles and the terrain can also be handled by the manager, although no detection of the interactions is necessary.

4.1. Solid-Object Collision

Handling of interferences in any simulation environment requires at least two things – detecting a collision and responding to the collision.

Whether or not two objects collide can be accurately determined by checking the intersection of every mesh component from one object against every mesh component in the other object. However, since there may exist hundreds of objects in the system, any such simple test will be very expensive and so algorithms that can efficiently detect potential collisions are required. We explored a number of standard approaches for collision detection, and implemented one of them.

Given a mesh of an object, we can define a simple bounding volume around the mesh that approximates the shape of the object. Since this volume is much simpler than the object itself, the collision detection is correspondingly simplified.

![Fig 2. Example of a sphere and an oriented bounding box defined around a vehicle.](image)

We are currently using two types of bounding volumes to test for collisions: Sphere and Orientable Bounding Box (OBB) [4]. The first type defines a sphere that encloses the whole object. Testing for intersection between two spheres is very simple, but the enclosure is very loose especially for objects like a vehicle, thus it is not tight enough for accurate detection and definition of collision forces. We use this simple method as a preliminary, fast, and efficient method for determining whether a more detailed test is necessary. If so, the Orientable Bounding Box method is then applied. This second test draws a translatable and rotatable box tightly around an object. The OBB method then computes the distance between all of the points of one bounding box and those of another bounding box. When the distance between two of those points falls below a threshold, or at least one point of one box lies inside of another box, then a collision is signaled. This type of interference-testing capability will eventually lead to improvements of simulation-based processes for design validation.

To make the detection more accurate, a hierarchy of OBB are defined so that loose bounding boxes are checked when the objects are far away, (but still inside the spherical limit) while more complex bounding boxes are used when the objects are close. This hierarchy balances speed and accuracy and has been widely used in games.

Once two objects are determined to collide, we must respond to this collision by imposing appropriate forces and torques on each involved object. Such forces and torques are then sent back to the VTB Pro simulation model of each vehicle to determine their effects on the objects, such as new speeds or new directions. Three variables must be known to determine the forces and torques: the collision surface, the collision point, and the minimum translation distance (MTD). This data is calculated using an ad-hoc algorithm that creates a
perimeter of points around the vehicles which adhere to their OBB’s. Using these points distance checks between every point in one box are checked against the other points in the other box to determine which two sets of points are the closest. Once these points are found the mid-point between these two sets of points is used as the collision point, the vector that the two points created becomes the collision surface, and the distance from the two points becomes the MTD.

There are more complex methods used in games that can detect the collision surface and location more accurately, including the Minkowski Difference test [5]. Given two bounding volumes, the Minkowski Difference subtracts the edges of one volume from the other. Taking this difference results in one of the volumes to become a point which is the origin point and the other becoming a volume that encompasses all of the features of two volumes. Fig. 4 shows an example. Then the distance is checked from every edge in the new encompassing volume against the origin point. If the origin point is contained inside the new volume, then a collision has occurred between the boxes. Once a collision is detected then several values are calculated to handle the collision response; the closest edge to the origin is the collision edge, the distance from the origin to the colliding edge is the MTD, and the point where the origin is projected on the collision vector is the collision point. In other words, using Minkowski Difference combines the two procedures of collision detection and response in a uniform way and can be applied to any two complex objects.

When a collision is detected, the corresponding collision forces are generated and sent back to the VTB Pro simulations that pertain to the collision to determine their effects.

Fig 3. Example of the ad hoc method of collision response.

Fig 4. Example of Minkowski Difference. Top: Two cars with bounding boxes defined. Bottom: resulting Minkowski Difference. The graph is shown in 2D to represent the concept, but Minkowski Difference is generalized in the 3D space.

4.2. Terrain Interaction

Terrain interactions (with tires) were identified as a second type of collision because these occur continuously and can be computed entirely by the game client without input from the server pertaining to other game-player’s vehicles. The method permits detection of contact or interference between an object and a rough surface such as terrain or unsettled water. This permits simulation of the physics of an object moving across such rough terrain.

We first defined a height map of the terrain in the form of a top-down topology map [6]. The whole space is discretized as evenly distributed patches, each consists of four corners some or all are shared with other patches. Each corner has two coordinates representing its latitude and longitude, and a height value is defined on a corner to represent its elevation. The height of the terrain for any given latitude and longitude (X, Y) can be easily computed by interpolating the four corners of the patch where (X, Y) resides in. For a vehicle with four wheels, the height of each wheel will then be calculated and presented to the VTB Pro simulation.

Currently all patches possess the same environmental property such as friction coefficient. In the future, each patch will contain different properties, and these properties can be changed dynamically during the simulation.
4. Exploiting Multicore Computing

From 2005, the race between semiconductor manufacturers has changed from achieving higher CPU clock frequency to integrating multiple execution cores onto a single die [7]. This trend provides modern, more powerful multi-core architectures to software developers comparing to the previous hardware platforms on which programmers could only use multi-threading [8] to simulate a multitasking environment. As a result, the computer game industry has started to employ the multi-core technologies for the next generation of computer games [9].

To maximize application efficiency under this trend, new programming models should support the following successful models of parallelism: task-level parallelism, word-level parallelism and bit-level parallelism [10]. In the current version of VTB system, the individual component models are computed sequentially, hence they are good starting points for applying task-level parallelism in multi-core systems. In this section, we will introduce the software and hardware platform to examine the potential improvement to VTB by applying multi-core techniques to the solution of any single model.

4.1. Software and Hardware Platform

We use the TPL package (Task Parallel Library) [11] in Visual Studio 2008 for parallel programming in multi-core systems. Compared to the conventional thread-pool method, TPL has three advantages: (1) developers do not need to consider the hardware difference; TPL will automatically take full usage of the installed processors; (2) the code using TPL is much more concise and much less error-prone; (3) TPL slightly outperforms the thread-pool method.

At this stage, we choose to conduct experiments by using a simulated solver instead of extracting out more complexed solvers used in the VTB system. The simulated solver is simply a loop containing some trigonometric functions. In our experiments, the number of iterations varies from 15 to 2000.

4.2. Experimental Results

We conducted experiments using two different computers: a single-processor dual-core (2.4GHz CPU) computer and a single-processor quad-core (2.6 GHz CPU) computer. Figure 5 shows the results. Here, the ideal value of normalized speedup is 1.0, meaning that speedup is directly proportional to the number of cores utilized. Speedup of less than one means that performance increases more slowly than the number of cores increases.

From this graph we can learn at least five things: First, we see that when the problem has low computational complexity, multi-threading overhead overwhelms the added computing power so the performance gain per added core is meager, only about 0.3 to 0.4. Second, performance gain per added core becomes greater at higher computational intensity, but remains well below unity. Third, the incremental benefit of more cores appears to decrease with number of cores – the apparent asymptotic value of speedup at high computational intensity is only 0.7 for four cores, while it is 0.8 for two cores. We don’t yet know whether it decreases further for increasing numbers of cores. Fourth, the speedup is more “noisy” for fewer cores, indicating that statistical processes show up more strongly. Fifth, depending on computational intensity of the tasks, the normalized speedup might increase with number of cores (low intensity), or it might decrease (high intensity). Not necessarily apparent from this graph, but apparent from the raw data, is that the computing time increases linearly with the increasing size of the computing job for any number of cores, which is an observation in line with those of other researchers [11].

4.3. Observations

Fig. 6 shows the CPU usage chart for each of the cores in a dual core computer, as the computing process is switched back and forth between sequential and parallel threads. We see that both CPUs are nearly fully utilized while running parallel threads, while in the sequential tests, the system distributes the workload almost equally to the two cores, but neither core is fully utilized. This picture gives us a very straightforward observation that TPL does automatically take full advantage of the processors installed.

Fig. 6 CPU usage chart of a dual-core computer. Point A indicates the usage for the sequential tests, and point B indicates the usage for the parallel tests.

Usually, the component solvers, especially for components that exhibit linear properties, run very fast so we see that we
probably cannot get much benefit from parallelizing component solvers. However, these experiments are not the real VTB solvers; they are just loops of some very computational intensive functions. The CPU(s) are highly occupied doing these calculations. It makes the overhead of handling the management of multi-threads higher than the real application.

Figure 5 shows that more speedup up can be gained with larger problems. Since the most complex and time consuming procedure in the VTB system us matrix inversion, we should be able to gain significant improvement. Thus in the future, we will conduct further tests on matrix inversions so that VTB can be improved on the new multi-core architectures.

5. Conclusions

Many ideas and paradigms can be extracted from Massively Multiplayer Online games and infused into the engineering world. To demonstrate some of the paradigms we have explored, an automobile racing game was created and is continually expanding on these paradigms.

In this early stage of development, we have shown how MMO client-server technology could be used for engineering collaboration across multiple simulations that run concurrently. The automobile racing game demonstrated how users running different simulations could interact with each other to simulate different cars racing on a terrain. This type of interaction is not limited to our automobile racing game. Using client-server interactions between multiple simulations can open up new types of collaborations between engineers and their designs of large systems.

We have also shown the importance of interference detection and response between multiple systems. The idea of interference is an important concept to accurately represent systems in the real world by giving the engineer the option of applying physical simulations based on the body of the object. This was also demonstrated using the automobile racing game in the form of collision detection and response between automobiles and also the terrain. The system has the ability to detect the collision of the automobile’s bodies against either another automobile or the terrain and apply the appropriate forces. The concept of interference detection will be further developed and abstracted into other areas of interest.

Finally, the concept of multicore technology was explored to effectively enhance the speed of the system. Using the TPL package as supplied by Visual Studio 2008, significant speed ups were evident in this early phase. Having VTB take advantage of this booming technology will greatly improve on the speed and testing of engineering technologies.

Using ideas from the gaming world have shown to enhance and expand VTB. More concepts and paradigms are being researched into and infused to further improve engineering tool by new methods of collaboration, new types of interference testing, and bigger speed ups using multi-core technology.

Acknowledgements

This work was supported by the US Office of Naval Research under contract N00014-07-1-0686.

References


