

An Overview of the STEAMiE Educational Game Engine

Scott Nykl, Chad Mourning, Mitchell Leitch, David Chelberg, Teresa Franklin, and Chang Liu
Ohio University, sn361906@ohio.edu, cm492997@ohio.edu, ml951702@ohio.edu, chelberg@ohio.edu, franklit@ohio.edu, liuc@ohio.edu

Abstract - Today, there exists a significant disparity in the degree of technological development between commercial and educational games. The STEAMiE Educational Engine is a cutting-edge system used to create advanced, realistic, immersive learning environments. These environments have been shown to influence comprehension and retention of hard-to-teach science concepts. This paper will present an overview of the features of our STEAMiE Educational Engine and how to use it to create games/simulations capable of enhancing a user's learning experience.

The STEAMiE Educational Engine contains a rich feature set allowing for development of powerful modules within a short period of time. From a development point of view, the STEAMiE Educational Engine's object oriented design is modular and can be easily extended to support new functionality in a timely manner. This significantly shortens development time and abstracts complexities from the developer allowing for clean and efficient implementations of games and simulations.

Index Terms - Educational Games, Game Engine, Immersive Environments, Virtual Worlds

BACKGROUND AND MOTIVATION

There is currently a lack of available software to create educational video games easily and rapidly, especially 3-D immersive games. Studies conducted by the Science and Technology Enrichment for Appalachian Middle-Schoolers (STEAM) program at Ohio University have shown that 3-D immersive games positively affect learning more so than 2-D games. Modern video game development is complicated, and the creation of games that provide both education and entertainment is even more complex. One of the main goals of the STEAMiE Educational Engine (SEE) is to fill the gap of creating educational as well as entertaining video games. SEE provides a framework that abstracts implementation complexities away from the developer allowing developers to focus on educational content creation for their games and simulations.

There are multiple development tools and collaborative environments that hint at educational capabilities and have been used for creating educational games, but no platform has been developed specifically for education. Some popular

tools to create educational games and the tools' shortcomings are described below.

I. Recreational Engines

The Torque Game Engine is one of the leading independent developer game engines [1]. There are multiple ways in which SEE is superior. First, Torque supports basic physics: things can fall, go in a straight line, follow a parabola, and collide with other objects [1]. In SEE the physics engine provides much higher fidelity; SEE can maintain realistic multi-object interactions that cascade indefinitely. Second, Torque, similar to many existing engines, utilizes its own "C++ like" scripting language. Scripting languages allow certain common scenarios to be easily implemented; however, as the complexity of a scenario increases the limitations of the scripting language force the developer to "jump through hoops" to achieve their goal. Conversely, SEE uses C++ as its "scripting language" creating an optimal balance of flexibility and ease of use. SEE is designed to enable creation of highly interactive, immersive, virtual worlds with ease. Highly interactive worlds often require construction of complicated scenarios making SEE ideal for these situations.

The Id Tech 4 engine [2] is a popular commercial game engine, like most commercial game engines its primary focus is creation of first-person shooters. First-person shooters are often not ideal catalysts for educating students. With a flexible infrastructure, SEE is designed to create any genre of video game.

The Id Tech 4 Engine has a single title license cost of \$250,000. This cost may be prohibitive to most educational initiatives. Likewise, the Torque Game Engine can be licensed for \$150, but to provide licenses for an entire computer lab with twenty machines would cost \$3,000 [1]. Often, a key requirement of educational game development is a restrictive budget. All SEE games are made freely available at <http://steam.cs.ohiou.edu/>.

II. Educational Engines

SAGE, Simple Academic Gaming Engine, is a game engine designed to teach computer science and game programming to undergraduate students [7]. The students learn computer science by designing and implementing computer games within SAGE. The games that are created within SAGE are not intended to educate others; the games that are created are

intended to educate the developer. In contrast, the games developed using SEE are designed for educating the player.

Alice is a teaching tool designed to introduce object-oriented programming. It utilizes a drag-and-drop interface and 3-D graphics to simplify the design of games [8]. Again, the games that are created within Alice are not intended to educate others; the games that are created are intended to educate the developer. In contrast, the games developed using SEE are designed for educating the player.

Linden Lab's Second Life is a massively multiplayer online virtual world supporting tens of thousands of online users simultaneously [3]. This collaborative environment can be used to develop educational games that can be used in a classroom setting. However, several issues with Second Life can cause problems within a typical school computer lab. First, each Second Life client requires an upper average of 30 KB/sec to navigate the virtual world [4]. In a computer lab with 20 clients, this is an average upper throughput of 600 KB/sec or roughly three T-1 connections. In many schools, this bandwidth requirement is prohibitively high. In comparison, the school districts, that are using SEE, have at most a single T-1 connection. Second, installation of Second Life on workstations would force many school districts to open additional ports on their firewall; the addition of potential security holes combined with the high bandwidth requirements, may cause IT administrators to resist endorsing Second Life. Third, while Second Life is a three dimensional virtual world, many objects within this world are static and cannot directly interact with a client's avatar. Second Life requires the user to "touch" objects by clicking on them with the mouse and selecting "touch" from a menu, as opposed to naturally pushing the object with their avatar. This creates a disconnect between the user and their avatar thus limiting immersion. In addition to this, the physics within Second Life provides lower fidelity than many educational simulations require.

Macromedia's Flash content creation suite is a popular system used for web content development [5]. Flash is also commonly used for educational software because Flash enables developers to rapidly create and distribute content. One of the major drawbacks, however, is that the vast majority of this software is two dimensional. Data being collected by the GK-12 STEAM project indicates that students working in 3D immersive virtual worlds versus 2D worlds such as Flash have more than twice the mean difference in scores on content achievement tests. Further testing in this area is under development to further examine this unexpected finding in the research of the STEAM project.

II. Key Missing Elements of an Educational Engine

The aforementioned educational platforms are lacking several key elements that should exist for an educational engine:

- Realistic 3D physics behavior

- "Real time" availability of the current physics state
- Highly flexible/modular development framework
- Timely development cycle

Without these elements, the development platform's capabilities hinder the range and quality of educational software that can be developed.

STEAMIE ENGINE OVERVIEW

I. Graphics Engine

SEE is equipped to use the full power of modern graphical systems, both hardware and software. SEE uses OpenGL 2.1 technology to create vivid, immersive, highly interactive worlds that are ideal for educational content. Beyond the normal OpenGL capabilities, SEE provides a flexible system for putting realistic, graphical content into a developer's custom designed virtual environment. SEE fully supports approximately 25 different file formats from which 3D models can be imported. In addition to pre-made models, SEE supports the ability for designers to customize their own objects interacting with OpenGL directly.

SEE has many useful graphical utilities including:

- Customizable automated state preservation system
- Dynamic light management system to keep track of activated and deactivated lights
- Automated object-level sorting mechanism to improve ease-of-use with transparent objects
- Simple frustum culling system to improve overall performance

Additionally, SEE has a fully functional particle system that includes pre-made particle templates, that allows designers to create their own. The particle system allows for emission of any kind of particle from spheres, to tetrahedrons, to textured billboards, or even 3D models.

SEE uses Guichan [6] for GUI widgets such as buttons, labels, text boxes, etc. SEE has extended Guichan to include special purpose widgets such as a multilingual dialog box.

Session F3B



FIGURE 1
IMAGE TAKEN FROM SPACE RACER, A STEAMIE GAME.

II. Animation Engine

SEE provides a mechanism to animate objects within a virtual world. An example of this is having a human appear to walk. SEE accomplishes this via a key frame animation system. Animation adds another dimension of immersion and realism.

The animation uses hierarchical joints, that join pieces of a body together providing a framework for animation. The animation system allows six-degrees of freedom per joint.

III. Physics Engine

SEE uses Russ Smith's Open Dynamics Engine (ODE) for physics simulation and collision detection. ODE is a high fidelity rigid-body Newtonian physics simulator; this provides highly realistic motion of objects and interaction between objects within the virtual world. In complex virtual worlds with many objects, this system is capable of performing all necessary computations quickly enough to simulate "real time".

SEE internally abstracts all the complexities of ODE away from the developer; thus, a developer only needs to tell SEE to create an object such as a car. Upon doing so, SEE internally uses ODE to create all the intricacies of the car such as the suspension, turning radius, torque power for each motor, etc. Of course, SEE easily allows a developer to directly set and manipulate these parameters if desired. SEE will provide intelligent defaults but always allow for a developer to change any parameters as necessary. This intelligent default behavior saves developers time and expedites game development time.

Additionally, SEE allows users to collect physics information about any object at any time during game play. This is useful in scenarios where one would like to observe

the force of a collision, the velocity of a falling object, the angular inertia of a flywheel, the acceleration of a rocket, or almost any other conceivable scenario involving physics.

Furthermore, surface properties such as friction, "bounciness", "hardness", "squishiness", etc, of each object can be specified. Thus, developers can create everything from ice rinks to parking lots covered in super glue.

IV. Force Abstraction Mechanism

SEE has a general force abstraction mechanism. This mechanism is an extension to the physics provided by ODE. ODE, itself, is able to apply forces on objects and subject those objects to Newtonian physics; however, more complex forces such as buoyancy, electromagnetism, gravity, pressure, wind, etc. require more complex governing equations. Hence, the force abstraction mechanism enables developers to define their own forces and how those forces interact with one another. SEE currently implements buoyancy and gravity, and it can be easily extended to support new forces by inheriting and overloading an abstract force base class.



FIGURE 2
IMAGE TAKEN FROM FORCE AND MOMENTUM, A STEAMIE GAME, SHOWING PHYSICS COLLISION DATA.

V. Network Engine

SEE supports a client-server networking mechanism. This allows an entire class of students to occupy the same virtual world and interact with one another. Similar to the physics subsystem, SEE abstracts all the complexities of the network subsystem from the developer. SEE provides intelligent default behavior for all objects across the network, but allows the developer to directly manipulate any of these properties to change the behavior as desired. Each game developed under SEE can automatically run in both server mode and client mode; this means that a computer lab could host one server locally and have all the clients join the local

server. By doing this, issues related to bandwidth consumption and firewall filtering policies are completely avoided.

SEE is also able to communicate with external databases to retrieve data such as questions, sounds, and images, as well as, instantly record student responses. This feature enables teachers to be able to monitor student's progress in "real time" as students play the game. The teacher can view a website that is continually updated as each student submits an answer; the website automatically grades and logs each question and the student's corresponding answer.

VI. Event-Driven Gameplay

SEE features an event-driven gameplay system. This system is used to create events that are often referred to as "waypoints", "triggers", or "scripted events". These events are in contrast to user-driven gameplay. Event-driven gameplay is the main mechanism that drives the plot of the game. SEE's event-driven gameplay system supports a prerequisite system that allows the designer to storyboard a full game and then modularly implement it with ease. SEE implements three basic types of waypoints: global waypoints, standard waypoints, and inverse waypoints. Global waypoints are triggered immediately whenever a set of prerequisites are met. Standard waypoints are triggered when an activator, such as a player, enters into a certain volume of space, called an activation volume. An inverse waypoint is triggered when an activator exits an activation volume. SEE provides spheres, rectangular prisms, and discs to use for an activation volume, but a developer can create any desired arbitrary volume. This volume can be an absolute volume of space or relative to another object. For instance, imagine a hallway that is surrounded by an activation volume. When an activator, such as a player, enters the hallway, the hallway lights turn on. When an activator leaves the hallway, the hallway lights turn off. This exemplifies an absolute waypoint activation volume. If the designer wanted to give the player a mechanism to enter/exit a movable car then the developer could place a relative way point activation volume around that car. This way, no matter where the car is, if the player is within the activation volume, then the player can enter/exit the car.

VII. Cross-Platform Support

SEE has been created with platform-independent C++ code, that allows for cross platform use. The engine has been ported so that it can run on Mac OS X (Intel and Power PC), Linux, and Windows. Cross platform support is not the only goal of SEE, the engine's aim was to provide educational games for schools, and since many public schools lack high-end computer systems, a need for software with low system requirements is necessary. Therefore, SEE performs runtime level-of-detail diagnostics based on the current machine's capabilities and adjusts rendering based on

available resources. This allows games to run on lower end systems, without losing any educational benefit or playability due to poor performance. Another benefit of SEE is that it does not need to be installed. Games created in SEE are stand alone executables, that can be run from a local hard drive, a server shared drive, a CD-Rom, or even a flash drive.

SIMPLIFIED GAME DEVELOPMENT

A goal of SEE is to create an environment that allows for rapid development of games. SEE uses an object oriented approach, that allows a new developer to inherit base template classes to rapidly extend and create modules with little effort. The object-oriented design provides high cohesion and low coupling, that leads to extensibility and scalability. The object-oriented approach inherently provides another benefit, that is maintainability. The code is then easy to manage and future users require little time to accustom themselves to the code base.

SEE abstracts away complexities so that content creation can be the focus during development time. There is intelligent default behavior for all base objects so the user can easily develop games. The objects are designed so that when a developer adds one to their world that object will have default physics, graphics, and networking behaviors. The developer can modify the default behavior by changing the parameters of objects or by writing new functionality for certain parts of the objects. Since there is very low coupling between subsystems (e.g. physics and graphics), if the developer changes part of the functionality of an object in one of those subsystems the other subsystems will be unaffected and act with the default behavior.

The user can easily create a world by using templates. SEE comes with a "New Module" template that the developer can expand upon to create a world. The template serves as a base world. The developer can add new objects to the world by simply: creating the object, positioning it, and adding it to the list of objects in the world.

As mentioned previously, SEE is designed so that the creation of games, simulations, and virtual environments requires little to no training. A motivated middle-school student, using SEE, could create a virtual world within several minutes of opening the "New Module" template. This virtual world could consist of a block that falls onto a plane and bounces according to Newtonian physics. Given several more minutes, a simple machine, such as a functional teeter-totter could be created. Thus, a box resting on one side of the teeter-totter could be launched when a box falls onto the opposite side. With a little more time a controllable car, driving on a teeter-totter, could be created. Within one class period a motivated middle school student could create a multi-stage Rube Goldberg device using SEE. Hence, this shows SEE's power and simplicity.

STEAMIE IN EDUCATION

The following is a summary of a selection of games created using SEE. Each summary will give the name of the module,

October 22 – 25, 2008, Saratoga Springs, NY

the subject matter, and any special features that module may contain:

- **Tide Island:** this module is intended to teach the player about the relationship between the earth, sun, moon, and tidal forces. It also contains two mini-games on the phases of the moon. This game features our buoyancy system.
- **Space Racer:** this module is a three dimensional, multiplayer race through outer space. The content of this game is based on SEE's question set customization system. The game features single and multiple player modes.
- **Force and Momentum:** this module is intended to teach the player about force and momentum. The player controls a car and drives around crashing into objects of varying mass. Each collision is recorded and the corresponding physics statistics are readily available for review via a table.
- **Mass vs. Volume:** this module is intended to teach the player the difference between mass and volume. The game features objects of twelve materials and ten shapes.
- **Digital Geology Lab:** this module is intended to serve as a replacement for anyone who does not have access to a Geology lab. It contains tests for streak, heft, and luster, as well as, different hardness tests. The player must then identify which of eighteen minerals they were testing.
- **Digital Gallery Walk:** this module allows the player to walk through a gallery containing the work of 35 sixth graders from Roseville Middle School. Each student created a "scientist trading card" with a hand drawn picture of the scientist on the front and his critical statistics on the back. The game includes a picture of each student and that student's audio recording about their scientist.

These games are just a sample of those created using SEE. Topics of other games include: position, velocity, acceleration, kinetic & potential energy, projectile motion, transfer of energy, pendulum mechanics, friction, and others. A bilingual game in English and Italian has been created using SEE.

CONCLUSION

SEE is ideal for educational development because of its high-fidelity physics engine and rich set of multi-media capabilities. This allows developers to create modules that exhibit accurate physics in an immersive manner. Additionally, this engine has been designed to run well on lower-end machines without expensive graphics and sound hardware.

ACKNOWLEDGEMENTS

We would like to thank the National Science Foundation for providing us this opportunity to create SEE. This material is based upon work supported by the National Science Foundation under Grant No. 0538588. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Thanks to Russell Smith and the ODE contributors for creating a powerful Open Source Physics Engine.

REFERENCES

- [1] 3D Engines Database [Online]. Available: <http://www.devmaster.net/engines/>
- [2] id Software: Technology Licensing [Online]. Available: <http://www.idsoftware.com/business/idtech4/>
- [3] Second Life Quality Metrics Second Life Grid [Online]. Available: http://secondlifegrid.net/resources/service_metrics
- [4] Multiplayer games and virtual worlds Part-2: Second life [Online]. Available: www.cc.gatech.edu/classes/AY2008/cs7270_fall/7270-lect15-games-2.ppt
- [5] Adobe Flash CS3 Professional [Online]. Available: <http://www.adobe.com/products/flash/>
- [6] GuiChan [Online]. Available: <http://guichan.sourceforge.net/>
- [7] SAGE [Online]. Available: <http://larc.csci.unt.edu/sage/>
- [8] Alice [Online]. Available: <http://www.alice.org/>

AUTHOR INFORMATION

Scott Nykl NSF GK-12 Fellow, MS Computer Science, Ohio University, sn361906@ohio.edu.

Chad Mourning NSF GK-12 Fellow, MS Computer Science, Ohio University, cm492997@ohio.edu.

Mitchell Leitch NSF GK-12 Fellow, MS Computer Science, Ohio University, ml951702@ohio.edu.

David Chelberg Associate Professor, Ohio University, chelberg@ohio.edu.

Teresa Franklin Associate Professor, Ohio University, franklit@ohio.edu.

Chang Liu, Assistant Professor, Ohio University, liuc@ohio.edu.