www.itcon.org - Journal of Information Technology in Construction - ISSN 1874-4753

USING GAME ENGINES FOR PHYSICS-BASED SIMULATIONS – A FORKLIFT

SUBMITTED: April 2010
REVISED: October 2010
PUBLISHED: January 2011

Jhih Ren. Juang, Graduate Student,
Computer-Aided Engineering Group, Department of Civil Engineering, National Taiwan University, Taiwan
jrjuang@caece.net

Wei Han. Hung, Graduate Student,
Computer-Aided Engineering Group, Department of Civil Engineering, National Taiwan University, Taiwan
magic@caece.net

Shih Chung. Kang, Assistant Professor,
Computer-Aided Engineering Group, Department of Civil Engineering, National Taiwan University, Taiwan
sckang@caece.net

SUMMARY*: The detailed simulation of construction operations remains a challenging task in construction practice. A tremendous number of labor-hours are required to set key frames of an animation step-by-step and considerable computational power is needed to render an animation frame-by-frame. This research aims to utilize a game engine, which has been a rapidly evolving field for over a decade, to reduce the effort required by developers. The process of developing a construction simulation was separated into three main steps: 3D model construction, setup of physical properties, and creation of interactive logic. The Blender game engine was selected as the implementation tool, and a forklift simulation was created to demonstrate the concept. With a keypad, users could control a virtual forklift to simulate the manipulation of an actual forklift. The simulation result shows the advantages of developing a game engine-based construction simulation. The quality of the simulation's graphics provides a high degree of realism, and the interactivity requirements between user and computer are satisfied.*

KEYWORDS*: virtual construction, physics-based simulation, physics engine, game engine, Blender*

REFERENCE: *Jhin Ren. Juang (2011)*Using game engines for physical – based simulations – a forklift, *Journal of Information Technology in Construction (ITcon), Vol. 16, pg. 3-22, http://www.itcon.org/2011/2*

1. IMPORTANCE OF PHYSICS-BASED SIMULATIONS

Construction activities are becoming more varied and relied upon in modern construction projects. Individual construction tasks are often very risky and any delay in these can result in an overall delay of the project. High-accuracy planning is necessary to ensure that safety and efficiency requirements are met. However, budgetary and space conditions often do not allow for practical experiments or make them hard to carry out. However, with rapid advancements in computer graphics and hardware, virtual reality technology has the opportunity to provide an environment for testing, experimentation, and simulation before the project starts (Whyte, 2002).

Recently many researchers have developed intelligent methods to efficiently generate animations of construction operations and processes (Kamat, 2003; Dawood *et al*., 2005; Wang and Dunston, 2007; Kamat, 2008). However, the simulations and visualizations generated by these methods introduce logical errors due to the restrictions of efficiency and communication ability, and simulations that ignore the physical details may result in an incorrectly designed plan. To have a realistic and relevant simulation, we have to consider the numerous and complex behaviors that exist in the real world, which means that we must engage branches of physics such as statics, rigid body dynamics (Kang and Miranda, 2009), and fluid dynamics.

A simulation founded on these disciplines is better able to simulate the detailed manipulation of construction machines, which can help to accurately locate potential problems such as spatial conflicts, inefficient machine operations, and over-loading due to incorrect manipulations. These involve object geometry, time factors and dynamic motions that conceptual animations or simulations usually ignore. By simulating detailed and accurate processes, the introduction of physical properties can also help to evaluate the project's constructability. To develop a realistic and reliable virtual construction environment or simulation, accurate physics is a basic and indispensable requirement.

2. PREVIOUS WORKS

To develop useful tools for effective communication and planning, many investigators have employed graphical technologies for visualizing and simulating the construction activities (O'Connor *et al*., 1994; Liu, 1995; Bernold *et al*., 1997; Stone *et al*., 1999). Lipmen and Reed (2000) used Virtual Reality Modeling Language (VRML 97) to provide 3D web-based technologies for managing, accessing, and viewing construction project information. Wilkins and Barrett (2000) applied the concept of a virtual construction site to develop a content-rich web site for teaching and learning construction technology. Waly and Thabet (2002) developed an interactive framework called the Virtual Construction Environment (VCE) to assist planners in visualizing, analyzing, and evaluating construction processes. The VIRCON, a strategic decision support system, has also been developed for practical use to manage construction schedules and, in particular, space planning (Dawood *et al*., 2002). Kamat and Martinez (2005) generated automatic dynamic 3D animations of construction processes using discrete-event simulation (DES) tools. In order to shorten the length of time required to produce animations and simulations, engineers usually have to simplify their complex scenarios and ignore details of the real construction project. Visualizations in these researches are developed for the purposes of presenting and communicating the conceptual idea or working process.

Recently, researchers have tended to introduce physics with the goal of making the visualization more logical and closer to real-world behavior. Huang and Gau (2003) developed a mobile crane simulator which considered collision detection and the inertial oscillation of the cable and the lift hook. Kamat and Martinez (2005) used the principles of forward and inverse kinematics to simulate the motions of articulated construction machines. Kang (2005) combined kinematics and numerical methods to generate detailed motions of cranes. In his research, rigid body dynamics was introduced to deliver a more realistic and accurate operations process which was able to simulate the suspension behaviors of the rigging system. This research has shown the potential of physics-based simulation and the possibility of operating realistic virtual machines in the virtual world.

An accurate simulation of realistic motion of a construction machine requires a lot of computational effort to calculate the position and orientation of each object at each time step. Therefore, researchers begin by using some mathematical approaches to reduce the cost of computation, in order to provide a real-time physics-based simulation (Simlog, 2007; CMLabs, 2007; GlobalSim, 2007). This increases the reliability of the machine

training system, which in turn reduces the operational and sensory differences between the real world and virtual environment manipulations. They developed their systems to cope with the computational problems of physics, making them like the *physics engine* found in the computer game field.

A physics engine is a library or a framework that hides the low-level details of the physics needed in VR applications, thus allowing developers to focus on the higher-level functionalities of the application (Nourian *et al*., 2006). There are many physics engines that have been developed in recent years. Because of the needs of real-time interaction in computer games, physics engines are required to have strong capabilities in speed, stability, and robustness. Hence, researchers have started utilizing the advantages of game physics in simulations in various fields, such as robotics and surgery (MSRDS, 2007; Maciel *et al*., 2009). Chi *et al*. (2007) and Hung and Kang (2009) have used physics engines to model the physics of a crane and simulate erection activities. These researches show the feasibility and potential of developing simulations using physics engines.

Although a physics engine can help developers construct physics-based models of construction machinery and simulate construction operations, developers still need to put a lot of effort in programming to integrate graphics, physics, and other components to complete the simulation, while handling users' inputs, audio cues, and program flow (Popescu *et al*., 2002). Therefore, researchers have started considering developing their simulations by basing them on commercial game engines. Chan *et al*. (2008) developed a simulation of a hydraulic mechanics experiment using Virtools, while Breland and Shiratuddin (2009) described the benefits and potential future of using game engines as a virtual development environment in the construction industry. Lowe and Goodwin (2009) used *Unreal Tournament 3* as a visualization tool for measuring the quantity and quality of pathways. Developing simulations via game technology can provide inexpensive state-of-the art 3D virtual worlds in a short amount of time and without extensive programming (Trenholme and Smith, 2008), which allows engineers to focus on relevant issues and topics of construction.

3. CHALLENGES IN CURRENT PHYSICS-BASED SIMULATIONS

The challenges in current physics-based simulation are encountered during developing virtual environments and physics effects. Lewis and Jacobson (2002) indicated that high quality virtual reality and physical simulation has too high a barrier of entry for most research developers. Developing virtual environments from the ground up is prohibitively costly, complicated and time consuming (Smith and Duke, 2000; Smith and Harrison, 2001; Laird, 2002; Robillard *et al*., 2003; Trenholme and Smith, 2008). The development of physics-based simulations needs interdisciplinary technology, knowledge and skills. In order to implement high-fidelity physics-based simulations, they must address some important requirements: high quality graphics, realistic physics effects and a highly efficient system.

The high-fidelity simulation requires high quality graphics that provide viewers with an effective 3D animation and good quality shading. An effective 3D animation must be smooth and continuous, maintain a correct key frame rate, and be temporally and spatially accurate (Rekapalli *et al*., 2008), and must be constructed with well-designed algorithms to transform the 3D data to the view shown on a screen correctly and efficiently. Shading determines what color will be displayed on each pixel, associated with lights, texture, materials, and 3D objects in the virtual environment. Well-designed shading algorithms and methods can give the viewer a realistic vision in real time. These procedures, which are known as *rendering*, are complicated and time-consuming to develop.

Physics effects, such as gravity, collision response, and other physical behaviors, are simulated with computations. These are based on several mathematical methods or algorithms derived from physical laws. The developers have to develop these methods or algorithms and combine them into the rendering pipeline, which is a critical procedure. The developers need to combine physics knowledge with mathematical skills and concerted programming efforts to complete the system. Even though some physics effects toolkits are available, like physics engines, using these toolkits still requires sound programming skills.

Furthermore, to deliver a greater range of application and usage of the simulation, real-time interaction is also required. In order to support a real-time interactive simulation, the efficiency of the system must be a matter of concern. The computation of 3D graphics, physics effects, and interaction threads must be handled in a very short space of time. Thus, the architecture of the simulation system must be properly designed for effectiveness

and efficiency. Hence, developing such critical and complex simulation systems is usually an important task for researchers or engineers in construction.

Fortunately, game technology has evolved, reaching outstanding performances with computer graphics and interactive user interfaces. Therefore, in this paper we propose a physics-based simulation method using a game engine for simulating realistic and quality construction operations without needing to spend time on low-level detailed programming. The process of development was separated into three steps.: 3D model construction, setup of physical properties, and creation of interactive logic. The provided visual programming interface allows developers to focus on building construction machinery models (graphical and physical) and to design the operation details and scenarios. A virtual forklift and an operation scenario were created to demonstrate the proposed idea. The total development time was approximately 40 labor-hours. This example shows that developing such real-time interactive simulations via game engines is efficient and easy.

4. METHODOLOGY IN A GAME ENGINE

The game engine is the heart of a game and consists of several components (Roosendaal and Wartmann, 2003). Fig. 1 shows the common architecture of a game engine which consists of five major components: the authoring tool, the physics engine, the rendering engine, the user interface (UI), the audio engine, and the authoring tool.

The physics engine, as mentioned above, is responsible for the computation of physics. It is essentially a big calculator for dealing with the low-level details of the physics simulation. It handles the mathematical functions to simulate physics; however it does not know what needs to be simulated (Millington, 2007). A typical physics engine partitions the physical simulation into two phases: collision detection and multibody dynamics. Collision detection is the process that determines contact behaviors between objects, while multibody dynamics calculates the dynamical motions of objects which are connected with each other by joints or exist independently (Erleben *et al*., 2005). Both are based on mathematical methods that are derived or modified from physical laws. However, in the designers' view, the detailed processes of the implementation of a physics engine are encapsulated and invisible; the designers need only focus on what simulation effects are needed, and generate these effects by specifying the required parameters or data to the physics engine.

The goal of a rendering engine is to draw a 3D world and display it on the screen. The work of the rendering engine can be described by three main parts (Eberly, 2001). The first part is to transform 3D data of game objects into 2D data in screen space, which includes two transformations. In the first transformation, the engine converts 3D data in the game world into 3D data in the coordinates of the view world, which is associated with the camera in the game world. It decides which view will be seen by the player on the screen. The second transformation converts 3D data of the view world into 2D data in the screen world. This process is called *projection*. In the projection, the 2D data is made available to be drawn as pixels on the computer screen. The second part of the rendering work is to exclude portions of the data which are invisible to the player. This process includes *culling* and *clipping*, which can decrease the computational costs of a rendering engine. The third part is to draw the 2D data in the screen space onto the computer screen. This process is called *rasterization*, which takes up most time of the time spent in rendering. The rasterization process decides the displayed colors of the pixels through the calculation of several shading effects, including materials, textures, lighting, specular effects, transparency, ray mirror, and other special functional effects.

The function of the audio engine component is to generate sounds while the game is running. Audio can be extremely important for a game's atmosphere, and can heighten player satisfaction and enhance the quality of a game. This audio can include recorded sounds, interface sounds, and sound effects (Novak, 2005). For example, voices and background music are recorded sounds, while button clicks are a kind of interface sound, and explosions and sounds of brakes are sound effects (FXs). To generate these sounds, an audio engine has to load, edit, and mix the sound data. An audio engine is usually provided with several functions to generate game audio, such as playing, mixing, 3D sound (stereo effects), the Doppler effect and the fading-out effect.

The component that represents interaction between game objects and a player is called the *user interface* (UI) which can be separated into invisible interfaces and visible interfaces (Novak, 2005). The invisible interfaces are the UIs which have no on-screen features; for instance, control triggers using a mouse or keyboard, which can be used to select, move items, navigate in a virtual environment, operate character movements, communicate with

NPCs (non-player characters), *etc*. The visible interfaces are the UIs that have on-screen features, such as selective menus, the health meter of a character, and any other on-screen information that is provided for the player.

The physics engine, the rendering engine, the audio engine, and the UI are the basic components required to develop a realistic physics-based simulation or virtual environment. Some frameworks of integrated development environments (IDE), such as XNA (Microsoft, 2008), provide an integrated library for utilizing these four components to develop their simulations or games. However, these still need good programming skills and effort to build up a robust and stable simulation system. Therefore, in order to help game designers avoid taking too much effort and time in developing the complex foundation and structure of a game program, advanced game engines provide authoring tools. The game designers just need to load the game content (3D models, textures, or sounds), specify parameters, and design the game logic via this authoring tool. This allows game designers to use straightforward commands or actions to order the game engine to execute detailed programming jobs. Thus, the game designers can spend their time on more relevant parts. In *section 5*, we will demonstrate the process of using these authoring tools to develop a simulation.

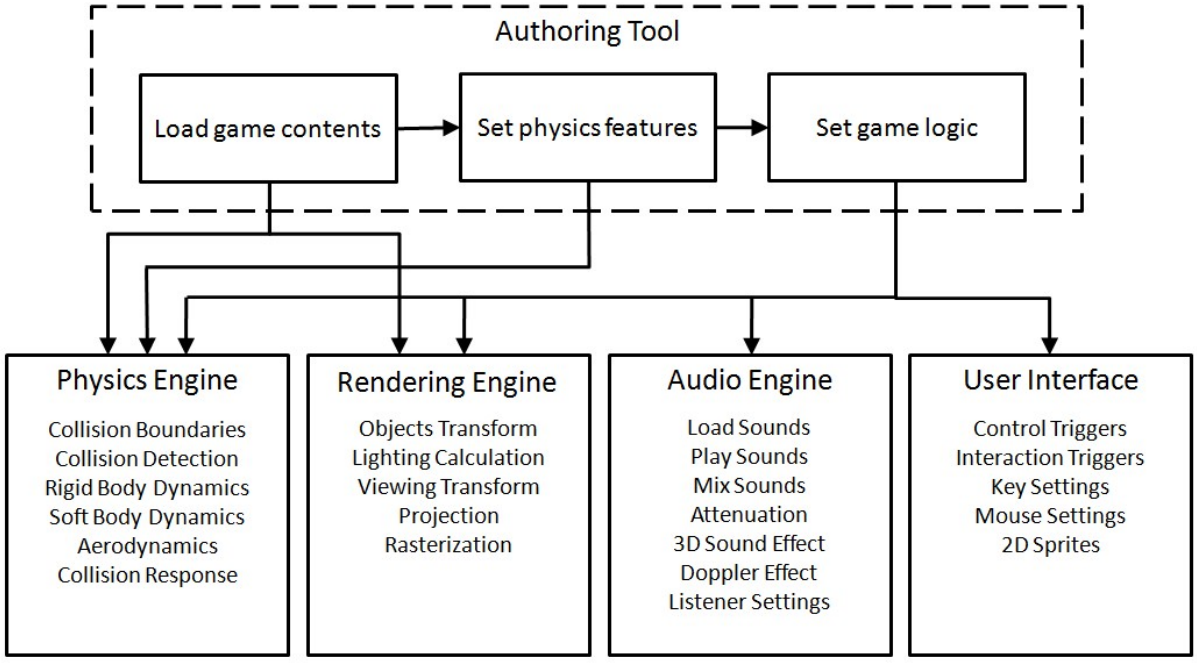


FIG. 1: Architecture of a game engine.

5. MODELLING A FORKLIFT USING A GAME ENGINE

5.1 3D modeling

We used the proposed method to develop a virtual forklift scenario. This scenario simulates the manipulation of a forklift and its dynamical motions. Users can also operate the virtual forklift to perform a specified task. We introduced the use of Blender, a free open-source 3D content creation suite with a built-in game engine, as a sophisticated implementation tool (Blender, 1995). The Blender game engine is the first game engine on which games can be developed without any programming required; it uses only a click-and-drag visual user interface (Roosendaal and Wartmann, 2003). The whole process of developing the simulation can be separated into three main steps: (1) 3D modeling, (2) physics feature setting, and (3) game logic setting. First, we created a forklift model consisting of numerous edited mesh objects, and then we grouped the objects of the forklift model into several components. This action helps us to construct the physical model easily, so that each component has its

own collision boundary, movements or functionality so that developers can manipulate them more intuitively. We modeled the forklift with six components: the forks, the carriage, the mast, the forklift body, the wheels, and the steering system, as shown in Fig 2. The following are the works and relationships of the components:

1) The forks and the carriage are used to load a lifted-object. They can move along the mast vertically.
2) The mast connects the carriage to the forklift body and can rotate vertically (pitch) in a small range of angles.
3) The forklift body is the main component of the entire forklift. It connects the other components together and can be driven by the wheels.
4) The wheels can rotate and move the forklift by adding torques.
5) The steering system containing a pair of uprights and rear axle objects which simulate a simple steering mechanism for the forklift.

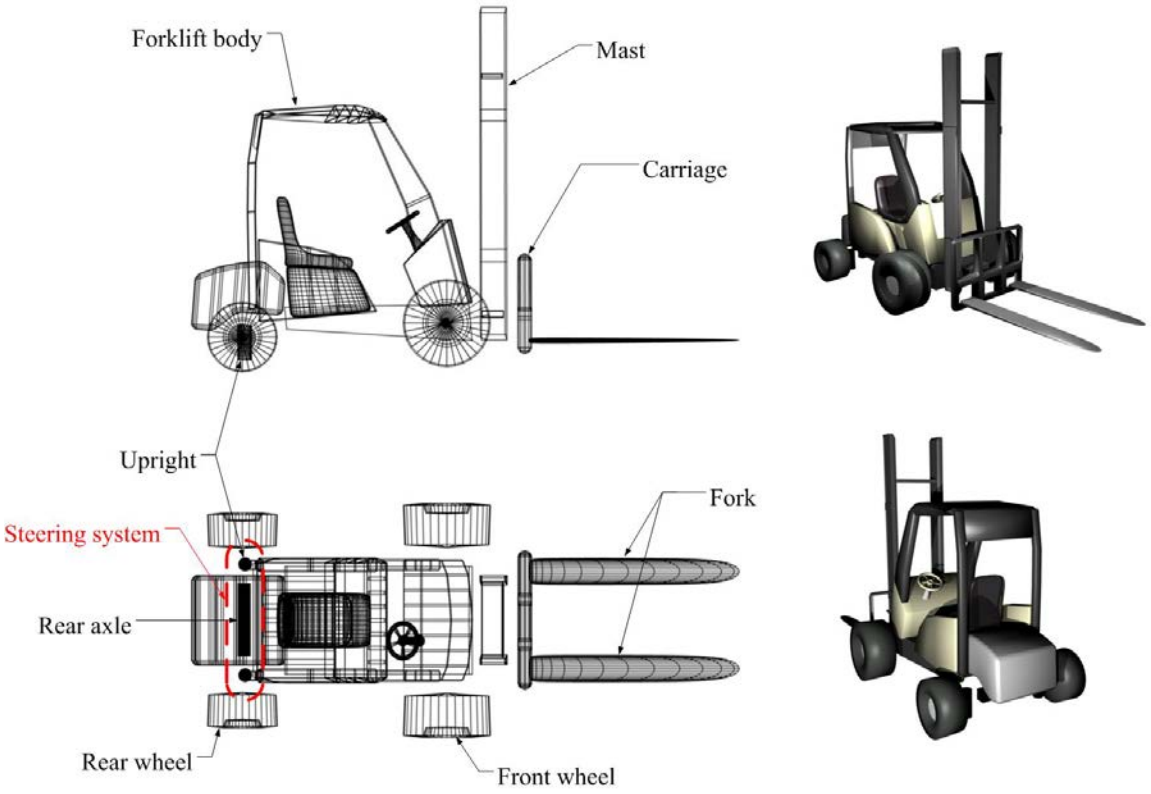


FIG. 2: The right-side wireframe view, bottom wireframe view and 3D shaded views of the forklift model.

In this step, we only constructed the appearance of the forklift and defined the function of each component. In the next step, we will use the defined visual model to construct the physical model to simulate the forklift.

5.2 Setting up of physics features

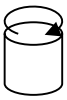
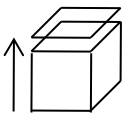
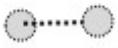
When setting the physics features, we constructed the physical model of the forklift as a multibody that consists of several rigid bodies connected together. We considered each component as a rigid body, and then had to set up the properties of each, such as mass, density and tensors. Then we needed to assign the collision boundary to each rigid body. A collision boundary describes the geometry that is used for collision checking between objects. After that, we assign each *joint*, which is used to connect a pair of rigid bodies, such as the hinge mechanism. These are important features when simulating articulated construction machinery. After finishing the setup of physics parameters, Blender will generate the physical model for the built-in physics engine automatically.

5.2.1 Physics joints

In order to clearly illustrate the forklift physics joint mechanism, we introduced and modified the notation developed in Hung and Kang (2009) for physics models using physics engines, as shown in Table 1. We used this notation to represent the forklift physics model, which presents the relationships between each object of the forklift as shown in Fig. 3. We will illustrate how to set up each joint via Blender in the following instances.

Blender provides an interface to set up physics joints that can simulate multibody behaviors. The engine provides several joint types, such as the *hinge*, the *ball*, and the *generic* joints. In this research, we used the hinge joints and the generic joints to simulate the virtual forklift. The hinge joints are used to connect the wheels and the uprights. A hinge (or revolute) joint has one degree of freedom in rotation along the local x-axis of a jointed object, so the wheels can be attached to the forklift body and revolve on their own x-axes, as shown in Fig. 4(a). The generic joint is useful when a developer needs to define custom joints that are not provided by Blender. We defined a custom revolute joint, derived from the generic joint, to describe the relationship between the uprights and the rear axle of the forklift as shown in Fig. 4(b), which simulate the steering mechanism. This custom joint must be fixed while the forklift is moving forward and backward; otherwise, the joints must be rotatable while the forklift is turning right or left. The method used to define this custom joint includes two parts: (1) assign the joint to the generic joint and constrain all degrees of freedom, and (2) define a game logic command whereby the upright objects rotate a specified degree around its z-axis when the operator triggers the command. Another generic joint application in the forklift is the custom slide (or prismatic) joint. The custom slide joint connects the carriage with the mast. This slide joint allows the carriage to move vertically along the mast. According to the local coordinate of the carriage, we constrained all degrees of freedom except for the translation about its local y-axis, as shown in Fig. 4(c).

TABLE 1: The notation and descriptions of physics model elements (Chi, 2007; NVIDIA, 2008; Hung and Kang, 2009).

Physics model element	Symbol	Description
Actor		A representation of a rigid body; an idealization of a solid body of finite size that is un-deformable.
Revolute joint		A revolute joint has a single rotational degree of freedom from two objects. The axis along which the two bodies may rotate is specified with a point and a direction vector.
Prismatic joint		A prismatic joint permits relative translational movement between two bodies along an axis without relative rotational movement at all. It is usually necessary to add joint limits to prevent the bodies from getting too far from each other along the joint axis.
Fixed joint		A dotted line connected two actors without the above joints means that there are no degrees of freedom between these two actors, and the two actors must behave like one rigid body.

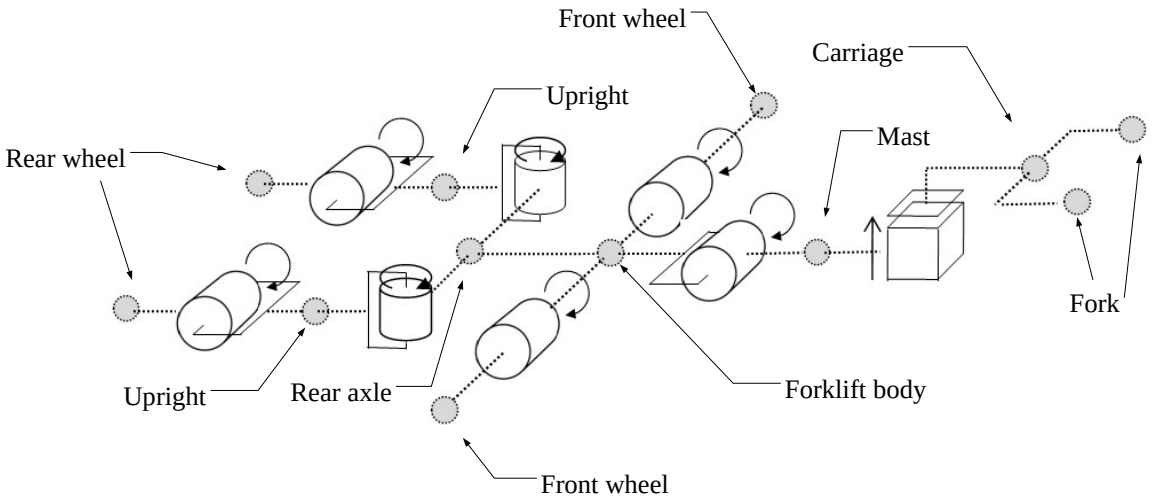


FIG. 3: The physics-based model of a virtual forklift.

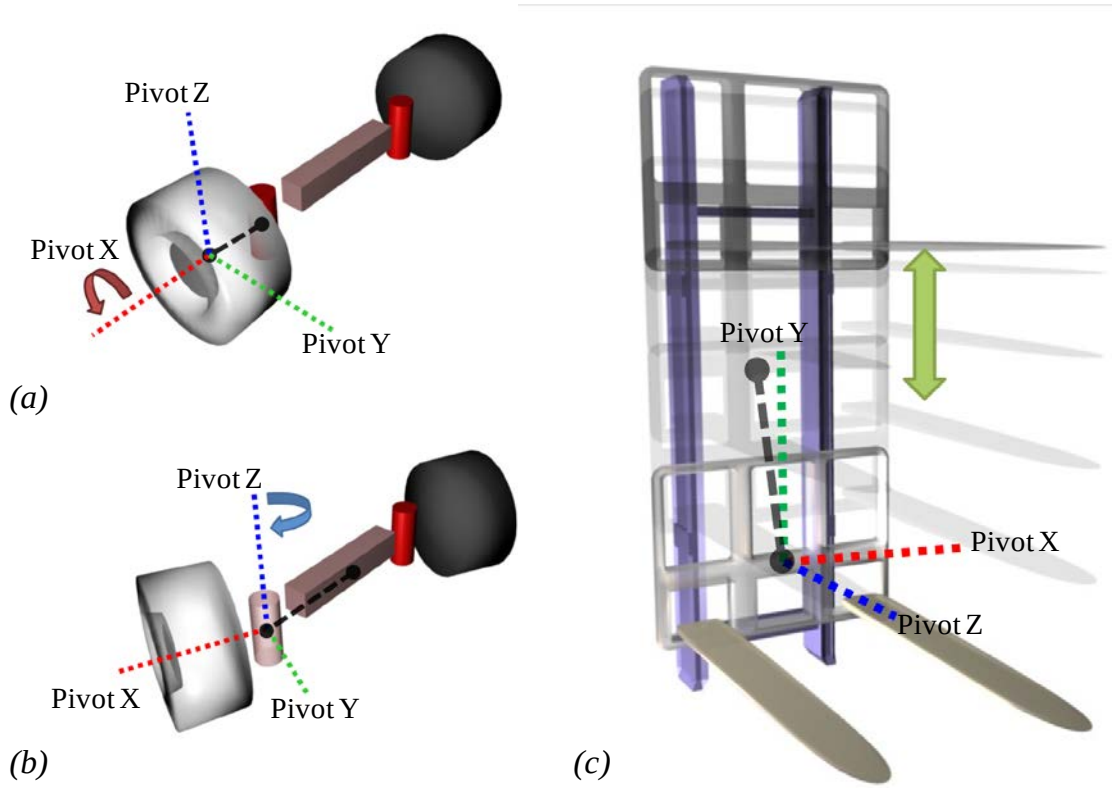


FIG. 4: (a) The hinge joint which connects the wheel and the upright, (b) the custom generic joint which simulates the steering mechanism, and (c) the custom generic joint as the prismatic joint which simulates the lifting mechanism.

5.2.2 Collision boundaries

The Blender game engine provides several types of boundary for developers to select, including *static triangle mesh*, *convex hull polytope*, *cone*, *cylinder* and *box*. Each type has its purpose for different demands of the game scene. Developers only have to assign the appropriate boundary to the object and Blender will automatically compute and construct the mesh in the physics engine for collision detection. In our forklift model, we chose the convex hull polytope type of boundary, which is generated by the mesh of the 3D visual model. The convex hull polytope converts the original mesh to a convex mesh, which can speed up the computation of collision checking and can still retain a certain quality of the boundary. Fig. 5 shows the collision boundary of the virtual forklift. The red boxes represent the approximated border of the collision margin, and the white lines are the mesh shape. The physics engine provides a small collision margin to improve performance and reliability of collision detection (Coumans, 2010). For a convex hull polytope, the collision margin is added on the surface of the mesh, so the collision boundary is slightly larger than the mesh shape and inside the collision margin.

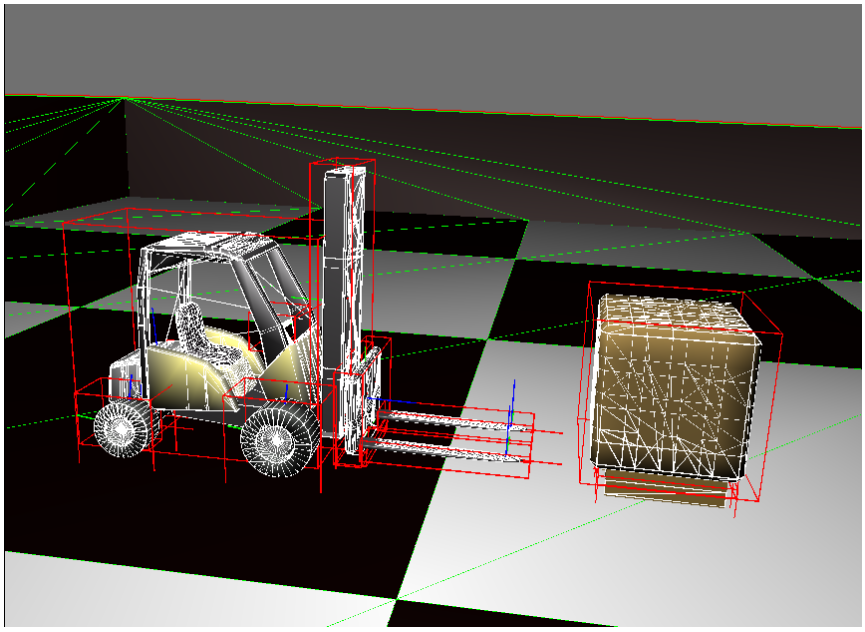


FIG. 5: The visualization of the collision boundaries with the convex hull polytope type.

In this step, we constructed the physics model of the forklift, so the forklift is like a realistic machine in the virtual environment yet without electricity. The next step will be to set up the mechanism that determines how the forklift behaves.

5.3 Game logic setting

In the final step, we set up the game logic to develop interaction with the simulation that allows users to operate the forklift in the virtual environment and accomplish a simple lifting task.

Interactive events (or behaviors) can be developed intuitively using a visual programming environment. The Blender game engine provides such a programming environment, called the Game Logic panel, as shown in Fig. 6, and it allows game designers to develop the event trigger mechanism of a game using drag-and-drop controls. We can define the relationship between user inputs and responses via this panel.

5.3.1 The method of setting the game logic

An *event* is when something happens (*sensor*) that satisfies a certain condition (*controller*), and then someone or something (*actuator*) will respond or execute some actions. For example, the virtual forklift operator pushes a button on the joystick to trigger the forklift to raise its forks. A trigger condition could be any other behavior, not just user inputs, such as when a collision occurs, a time condition, when something enters an area, or any other events that may happen during the simulation.

The Game Logic panel provides three sub-panels for setting up events: Sensors, Controllers, and Actuators. Following are the descriptions of these three sub-panels.

1) Sensors: This interface provides developers with several sensor types and their options. These are the triggers of events, such as keyboard inputs, joystick inputs, object collisions and other game event triggers, which allow developers to define when events will occur. If a sensor is triggered, it will return a “true” value immediately to Controllers (Blender, 2009).

2) Controllers: This receives logical values from sensors and determines if all the sensors would satisfy the defined conditions (such as the A key and the B key are being pressed simultaneously). If the condition is satisfied then the Controller will send a “true” value to Actuators, otherwise a “false” will be sent. The Controllers provide a function that is like a logical operation, such as AND or OR. For instance, when the logical values from two or more sensors are collected by an

AND controller, it will pass a “true” value to the connected actuator if all the logical values received by the AND controller are true (Blender, 2009).

3) Actuators: This takes actions or reactions when it receives “true” values from Controllers. There are several types of actions provided by Actuators, such as motion, sound, game actions and other game events, which can perform different kinds of events; for example, forcing objects to rotate, playing a sound and ending the simulation (Blender, 2009).

The developers have to set up the options and parameters of each sub-panel and then connect each sub-panel by using the mouse to drag the connection lines.

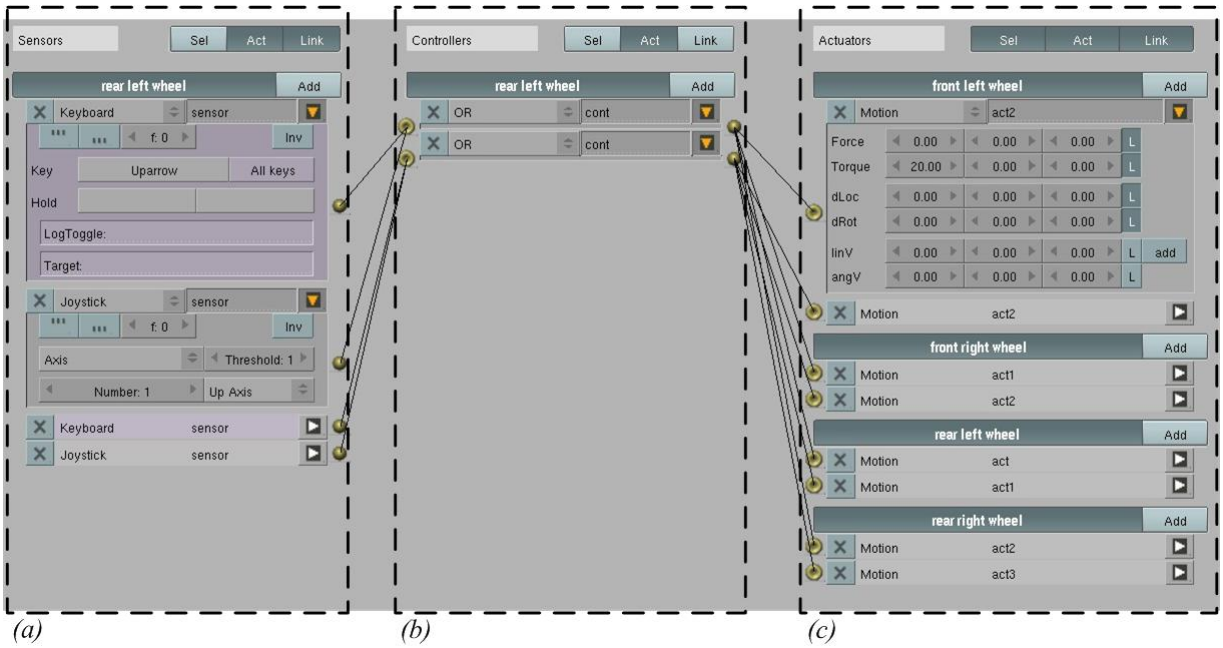


FIG. 6: A screenshot of the Game Logic panel: (a) the Sensors sub-panel, (b) the Controllers sub-panel, and (c) the Actuators sub-panel.

5.3.2 Set up of the interactive logic of the forklift

When setting the interactive logic of the virtual forklift, we use a keyboard and a joystick as input devices to drive and operate the forklift. When the operator inputs to trigger the Sensors, the Controllers then decide which actions parts of the forklift need to take, such as speeding up, braking, or lifting up a load. After these actions are confirmed, the Actuators will start to execute the assigned actions; for instance, adding torque to the wheels to drive the forklift.

The logic for adding torque to drive the forklift is shown in Fig. 6. There, logic settings for two forklift operations for moving forward and backward are set up, and these operations can be triggered using a keyboard or joystick. In detail, both operations have a Keyboard Sensor and a Joystick Sensor, which are specified with input triggers (such as the up-arrow key). These two Sensors of the same operation connect with an OR Controller; that is, either the Keyboard Sensor or the Joystick Sensor are triggered, and the OR Controller will pass a “true” value to the connected Motion Actuators. The Motion Actuator below the forklift front left wheel object, for example, was specified a value under the Torque options, and thus the front left wheel will rotate if either Sensor is triggered. By the same token, the other wheels’ Motion Actuators, which connect with the same OR Controller, will rotate simultaneously with the front left wheel rotation.

In short, in addition to the rigging for driving, all the virtual forklift movements, such as the loading operation, are controlled using a keyboard or a joystick and can be built via keyboard or joystick sensors, controllers, and motion actuators by a developer.

6. IMPLEMENTATION

To test the completed virtual forklift, we created a lifting scenario for experimentation. In the virtual environment, physics effects were built to support realistic dynamical motions. Both the virtual forklift and the lifted objects have collision responses and are influenced by gravity. This simple task also provides an interaction interface such that the forklift can be operated using a joystick or a keyboard through about ten operation commands. In the designed scenario, operators are requested to operate the forklift to complete a simple lifting task.

6.1 The development environment of the virtual scenario

This virtual forklift scenario was developed using version 2.45 of Blender, including a built-in game engine (BGE) and a physics engine. Blender is a free open-source 3D graphics design suite which provides tools for developers to support development of 3D modeling, animation and real-time games. The built-in physics engine is based on Bullet (2005), a free physics library for 3D games. Bullet is widely used by commercial games for PCs and for consoles such as the Xbox 360, Playstation 3, Wii and iPhone. Bullet provides collision detection, rigid body and soft body dynamics solvers to perform physics effects.

The hardware environment for development included a laptop: an Acer Aspire 5920 with Intel® Core™ 2 Duo T7500 (2.20 GHz, 2.20 GHz) and NVIDIA® *GeForce*™ 8600M GT graphics card (512 MB); and a joystick for interacting with the virtual scenario.

6.2 Designing the virtual scenario using a game engine

We designed a task in which an operator needs to operate the forklift to load a lifted object and deliver it to a specified unload position. There are three processes included in this scenario: the first process is the setup of the virtual forklift using game logic, while the second process includes the setup of the lifted object and the shelf for storing the objects. We assume that the shelf in the environment is static in that it always stays at the same position, thus the shelf does not need any settings for rigid body options, and instead uses a *static collision boundary*. Static collision boundaries let simulated objects be like solid rock and always static in a virtual scenario. The third process is the switch of cameras. Via the game logic settings, the operator can switch between different views in run-time with a key or a button. We set up two default cameras for operators. One is fixed on the forklift, which simulates the view from operators in the cabin. Another is a global camera which is located outside the forklift and around the environment, which provides a global view during the forklift simulation for operators to clearly monitor the entire working area. The layout of the working area is shown in Fig. 7.

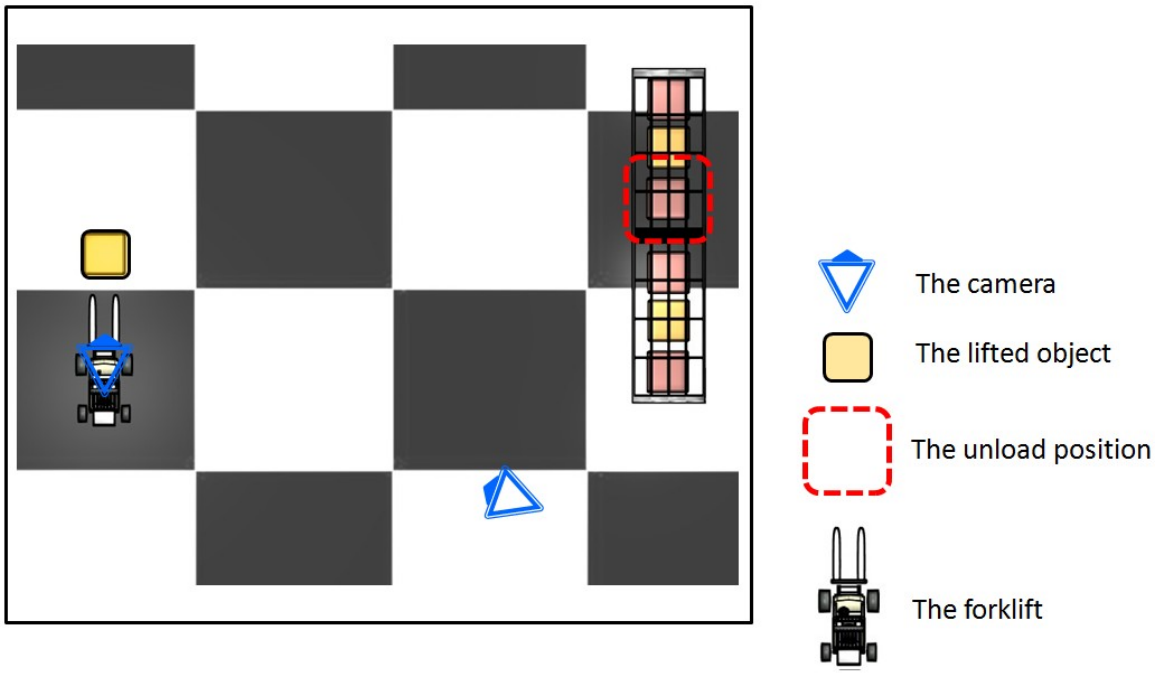
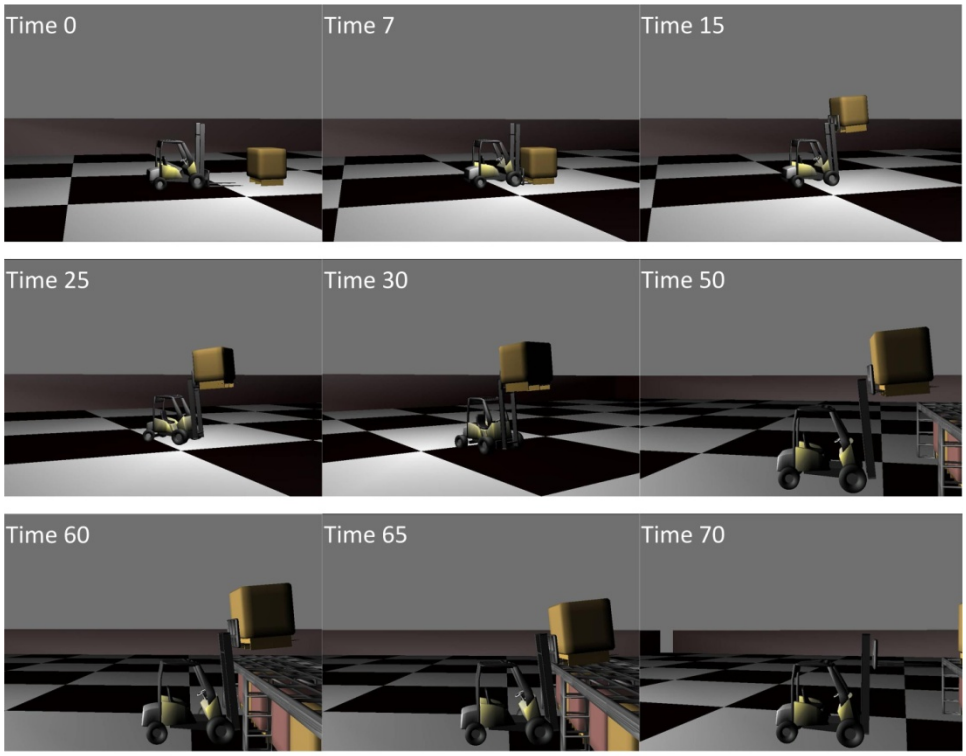


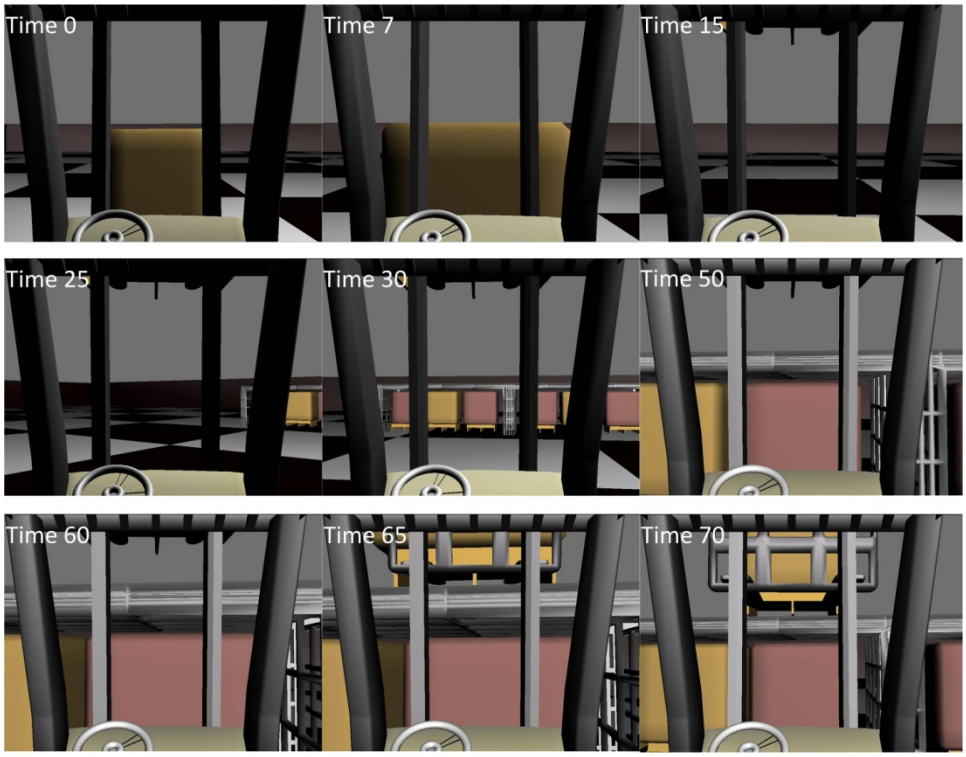
FIG. 7: The virtual forklift scenario.

6.3 Rendering result

The rendering result is shown at Fig. 8, which displays the task process executed by users. The task is to transport the lifted object to the specified unload position on the top of the shelf in Fig. 7. There are two views that can be selected to monitor the operations. In Fig. 8 the upper snapshots were taken from the global camera; the lower ones were taken from the forklift local camera. By switching the views during operation, the operator can make sure from two perspectives that the travel directions and loading/unloading positions are correct. During the task, from time 0s to 15s, the forklift lifted the pallet under the lifted object. From 15s to 50s, the forklift transported the lifted object toward the shelf. From 50s to 70s, the forklift was near the shelf, then the operator slowed down the forklift and unloaded the object at the unload position. The result shows a smooth and realistic simulation of the lifting scenario. Users can be immediately involved in the simulation and easily understand the process of the lifting operations by manipulating and “playing” with the virtual machine in the virtual environment.


Time 0
Time 7
Time 15
Time 25
Time 30
Time 50
Time 60
Time 65
Time 70

(a)


Time 0
Time 7
Time 15
Time 25
Time 30
Time 50
Time 60
Time 65
Time 70

(b)

FIG. 8: The rendering result: the process of implementing a forklift task. The operator was operating the forklift to transport the lifted object to the specified position: (a) the view from the global camera during the task, and (b) the view from the forklift local camera during the task.

7. COMPARISON OF GAME ENGINES AND PROGRAMMING PACKAGES

The results of this research show that developed simulations deliver quality rendering that is efficient and stable. We found that developing interactive, physics-based simulations using game engines has several advantages in comparison with previous methods, in which developers usually program step-by-step to implement and integrate rendering, physics simulation, user interfaces, and other required components using programming packages (e.g. OpenGL). In Table 1, we compare the benefits of using game engines and programming packages to develop simulations.

TABLE 1: Pros and cons of developing physics-based simulations using game engines and traditional methods.

	Using game engine	Programming packages
Effort to deliver high quality graphics	Low	High
Accuracy of simulation of physics behaviors	Limited	Dependent
Variation of simulation purposes	Common	Little limitation
Extendibility and reusability	Easy but limited	Dependent
Required programming skills	Low	High
Prototyping	Easy and Fast	Difficult
Focus on developing simulation methods	High	Low

Using a game engine can reduce the effort required to create a high quality graphical simulation. Most game engines provide many standard and high-quality rendering methods and visual effects which are usually sufficient for most engineering simulation purposes. Meanwhile, to achieve the same rendering quality using a programming package such as OpenGL requires extensive experience and good programming skills in computer graphics, even for a professional programmer. By utilizing a game engine, developers are allowed to develop advanced and special visual effects or rendering method by programming or adding scripts.

The accuracy of a simulation of physical behaviors is limited by the physics engine that the game engine employs, although most can perform stable and realistic simulations. But the core computation of physics in a physics engine is hard to modify or extend by users. In this case, using programming packages or developing one's own physics engine may lead to a more or less accurate simulation, which depends on the developer's programming skill and approach. However, developing an efficient, stable, and accurate simulation to support real-time interaction is a critical task, which is why physics components have become the responsibility of a separate engine in modern, advanced computer games.

Currently, the number of physics behaviors that can be simulated with a game engine is limited. Almost all existing physics engines support multi-rigid-body simulations, which is suitable for modeling construction machines. Some physics engines also support simulations of soft bodies, fluids, and combinations of these. However, to support other types of simulation, such as a pulley system, developers must develop their own components and integrate them into the engine. Programming packages have few limitations when it comes to varying physics simulations, in contrast to using a game engine. Developers are not limited by the frameworks and purposes of existing physics engines and can develop their own simulation approaches.

In the interests of extendibility and reusability, a game engine is easier to extend and add on new components with additional programming, because of the structural software design. However, it is still somewhat limited because the developed methods or functions are based on the core of the game engine. On the other hand, using programming packages to develop simulations may allow high degrees of extendibility and reusability, but still

depends on the skills of the programmer. Developing a highly extendible and reusable program or system is a difficult task, even for a professional programmer.

In the area of programming requirements, using a game engine to develop physics-based, interactive simulations requires less programming skill and effort. Programming is only needed when developers need to create an advanced graphical effect, additional physical behaviors, or to integrate other required components. To develop such a functional simulation system using programming packages, sufficient knowledge of software design and computer graphics and extensive experience in programming are needed. This also makes prototyping the simulation easier and faster with a game engine. Developers do not need to spend time testing many different packages to find a suitable tool for development. Therefore, they can focus more on developing the simulation models and scenarios than on handling the programming problem and troubleshooting.

8. DISCUSSION

We summarize the advantages of using a game engine as follows:

1) *Easy to develop*: The three steps of developing a simulation are intuitive and can be learnt in hours, even without programming experience. Developers just need to use a mouse to click, drag, and connect the relationships between events in the scenarios. The developed simulations are also easy to extend and modify for various purposes.
2) *Shorter development time*: The game engine encapsulates reusable game components that can help developers to save effort on building environments for development. This shortens the process of development so they can focus on building construction scenarios.
3) *Quality*: Thanks to the outstanding progress of game technology, game engines provide high quality graphics pipelines and game physics effects. The fine detail of a scene rendering depends on the developer's modeling efforts. The engine can also import various types of model formats constructed by others. Developers do not have to spend too much time handling model coordination, light shading, or texturing.
4) *Fast and stable simulation*: The 3D visual objects in the environment can be easily combined with physical objects. Developers do not have to write programs to integrate rendering and physics components, which helps to avoid the errors that occur during data transfer and decrease debugging efforts.
5) *Real-time interaction*: Compared to numerical methods, the physics engine inside a game engine uses mathematical approaches to speed up computation, which allows for real-time simulation of physical behavior. The efficiency of a physics engine allows for real-time interaction so that users can manipulate machines constructed in the virtual environment and get immediate feedback.

Although using a game engine to develop physics-based simulations has many advantages, there are however limitations in existing game engines. The accuracy of the physics engine is the major problem that we encountered. Although it delivers a visually plausible simulation, the developed physical models of articulated machines still have unexpected behaviors. This drawback depends on the physics engine that the game engine employs and the physics properties specified by users. There are two common, major physics problems that usually occur in existing game engines:

1) *The crash of physics joints:* The joint constraints between objects are not stiff enough to correctly fix jointed objects together, so they may separate unnaturally while a joint bears a heavy mass or a large impulse (Fig. 9 (a)). The extent of the error is related to the approach which the physics engine uses.
2) *The penetration of collision boundaries:* Penetration may occur when an object contacts a sharp edge of another object. For example, Fig. 9 (b) shows that the forks unnaturally pierce the lifted object. This problem usually happens when the collision boundaries are not precisely specified to fit the visual model (3D model).

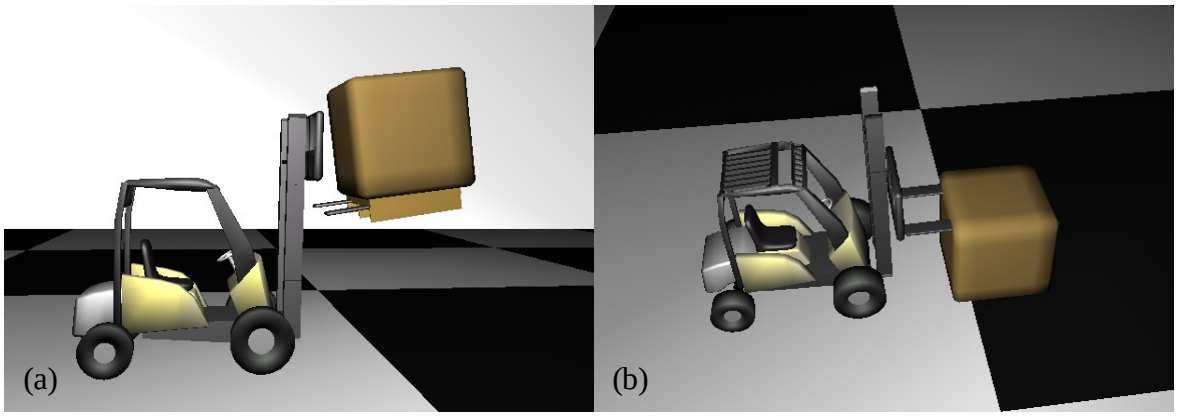


FIG. 9: (a) The joint which connects the forks and the carriage breaks because of the heavy lifted object; (b) the penetration of the forks into the lifted object due to the sharpness of the boundary of the forks.

9. CONCLUSIONS

In this research, we demonstrated the use of a game engine to efficiently develop physics-based simulations. Blender, a free, high-quality, open-source game engine, was employed in this research. Developers who do not have a strong programming background are able to develop simulations with the authoring tool provided by the game engine. We separated the process of building physics-based simulations into three steps: (1) creating the 3D models, (2) specifying physics parameters to switch and control the realistic physics behaviours, and (3) setting up the game logic. These three steps can be completed using the visual programming interface embedded in Blender. This reduces the time and effort for developers to switch between various computer tools and allows them to focus on the simulations. In this research, a forklift scenario was created using these three steps. We found five advantages to using game engines: *easier development, shorter development time, higher graphical quality, more stable simulation,* and *better real-time interaction*, in comparison with the traditional methods. In the future, the construction industry should take advantage of the sophistication of game engines to develop more physics-based models and create a standard library. This will enhance virtual construction to the level of physics-based simulation.

10. REFERENCES

Bernold L., Lorenc S. and Luces E. (1997). On-line assistance for crane operators, *Journal of computing in civil Engineering*, Vol. 11, No. 4, 248-259.

Blender. (1995). Retrieved Mar 09, 2010, http://www.blender.org/.

Blender. (2009). Game engine documentation, Retrieved Mar 09, 2010, http://wiki.blender.org/index.php/Doc:Manual/Game_Engine.

Breland J. S. and Shiratuddin M. F. (2009). Virtual environment on the Apple Iphone/Ipod Touch, *Proceedings of 9th international conference on construction application of virtual reality (CONVR)*, Sydney, AU, 257-265.

Bullet. (2005). Game physics simulation, Retrieved Mar 20, 2010, http://bulletphysics.org/.

Chan Y. C., Chen Y. H., Kang S. C. and Lee T. H. (2008). Development of virtual equipment for a hydraulic mechanics experiment, *Tsinghua science and technology*, Vol. 13, No. 1, 261-265.

Chi H. L., Hung W. H. and Kang S. C. (2007). A physics based simulation for crane manipulation and cooperation, *Proceedings of computing in civil engineering conference*, Pittsburgh, USA, 24-27.

CM Labs. (2007). Vortex simulators, Retrieved Mar 09, 2010, http://www.vxsim.com/en/simulators/index.php.

Coumans E. (2010). Bullet 2.76 physics SDK manual, Retrieved Mar 09, 2010, http://bulletphysics.org/.

Dawood N., Scott D., Sriprasert E. and Mallasi Z. (2005). The virtual construction site (VIRCON) tools: an industrial evaluation, *Journal of information technology in construction (ITcon)*, Vol. 10, 43-54.

Dawood N., Spriprasert E., Mallasi Z. and Hobbs B. (2002). Development of n integration resource base for 4D/VR construction processes simulation, *Automation in construction*, Vol. 12, 123-131.

Eberly D. H. (2001). *3D game engine design*, Morgan Kaufmann, San Francisco, USA.

Erleben K., Sporring J., Henriksen K. and Dohlmann H. (2005). *Physics-based animation*, Charles River Media, Boston, USA.

GlobalSim. (2007). GlobalSim, Retrieved Mar 09, 2010, http://www.globalsim.com/.

Huang J. Y. and Gau C. Y. (2003). Modelling and designing a low-cost high-fidelity mobile crane simulator, *International journal of human-computer studies*, Vol. 58, No. 2, 151-176.

Hung W. H. and Kang S. C. (2009). Physics-based crane model for the simulation of cooperative erections, *Proceedings of 9th international conference on construction application of virtual reality (CONVR)*, Sydney, AU, 237-246.

Kamat V. R. (2003). VITASCOPE - extensible and scalable 3D visualization of simulated construction operations, *Ph. D. dissertation*, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA.

Kamat V. R. (2008). Logical product models for automated scripting of process-level construction animations, *Advances in engineering software*, Vol. 39, 233-241.

Kamat V. R. and Martinez J. C. (2005). Dynamic 3D visualization of articulated construction equipment, *Journal of computing in civil engineering*, Vol.19, No.4, 356-368.

Kang S. C. (2005). Computer planning and simulation of construction erection process using single or multiple cranes, *Ph. D. dissertation*, Stanford University, San Francisco, USA.

Kang S. C. and Miranda E. (2009). Numerical methods to simulate and visualize detailed crane activities, *Computer-aided civil and infrastructure engineering*, Vol. 24, 169-185.

Laird J.E. (2002). Research in human-level AI using computer games, *Communications of ACM*, Vol. 45, No. 1, 32-35.

Lewis M. and Jacobson J. (2002). Game engines in scientific research, *Communications of the ACM*, Vol. 45, No. 1, 27-31.

Lipman R. and Reed K. (2000). Using VRML in construction industry applications, *Proceedings of the Web3D:VRML 2000 symposium*, Monterey, USA, 1-7.

Liu L. (1995). Construction crane operation simulation. *Proceedings of 2nd congress in computing in civil engineering*, Vol. 2, ASCE, Atlanta, USA, 1420-1426.

Lowe R. and Goodwin R. (2009). Computing gaming technlogy and porosity, *Proceedings of 9th international conference on construction application of virtual reality (CONVR)*, Sydney, AU, 127-137.

Maciel A., Halic T., Lu Z., Nedel L. P. and De S. (2009). Using the PhysX engine for physics-based virtual surgery with force feedback, *The international journal of medical robotics and computer assisted sugery*, Vol. 5, No. 3, 341-353.

Microsoft Robotics Developer Studio. (2007). Retrieved Mar 09, 2010, http://msdn.microsoft.com/zh-tw/robotics/defaut.aspx.

Microsoft. (2008). XNA. Retrieved Mar 14, 2010, http://msdn.microsoft.com/en-us/xna/default.aspx.

Millington I. (2007). *Game physics engine development*, Morgan Kaufmann, San Francisco, USA.

Nourian S., Shen X. and Georgana N. D. (2006). Xpheve: an extensible physics engine for virtual environments, *Proceedings of IEEE Canadian conference electrical and computer engineering*, IEEE CS, 1546-1549.

Novak J. (2005). *Game development essentials*, Thomson Delmar Learning, Clifton Park, NY, USA.

NVIDIA. (2008). NVIDIA PhysX SDK 2.8 - Introduction, Santa Clara, USA.

O'Connor J., Dhawadkar P., Varghese K. and Gatton T. (1994). Graphical visualization for planning heavy lifts, *Proceedings of the 3rd Congress on Computing in Civil Engineering* (Khozeimeh K., editor), ASCE, New York, USA, 759-766.

Popescu G. V., Burdea G. C. and Trefftz H. (2002). Multimodal interaction modeling, *Handbook of virtual environments: design, implementation, and applications* (Stanney K. M., editor), Lawrence Erlbaum Associates, New Jersey, USA, 435–454.

Rekapalli P. V., Martinez J. C. and Kamat V. R. (2008). Algorithm for accurate three-dimensional scene graph updates in high-speed animations of previously simulated construction operations, *Computer-aided civil and infrastructure engineering*, Vol. 24, 1-13.

Robillard G., Bouchard S., Fournier T. and Renaud P. (2003). Anxiety and presence using VR immersion: a comparative study of the reations of phobic and non-phobic participants in therapeutic virtual environments derived from computer games, *CyberPsychology and behavior*, Vol. 6, No. 5, 467-475.

Roosendaal T. and Wartmann C. (2003). *The official blender gamekit: interactive 3D for artists* (Roosendaal T. and Wartmann C., editors), William Pollock, San Francisco, USA.

Simlog. (2007). Mobile crane personal simulators, Retrieved Mar 09, 2010, http://www.simlog.com/personal-crane.html.

Smith S. P. and Duke D. J. (2000). Binding virtual environments to toolkit capabilities, *Eurographics*, Vol. 19, No. 3, 81-89.

Smith S. P. and Harrison M. D. (2001). Editorial: user centered design and implementation of virtual environments, *International journal of human-computer studies*, Vol. 55, No. 2, 109-114.

Stone W., Reed K., Chang P., Pfeffer L. and Jacoff A. (1999). NIST research toward construction site integration and automation. *Journal of aerospace engineering*, Vol. 12, No. 2, 50-57.

Trenholme D. and Smith S. P. (2008). Computer game engine for developing first-person virtual environments, *Virtual reality*, Vol. 12, 181-187.

Waly A. F. and Thabet W. Y. (2002). A virtual construction environment for preconstruction planning, *Automation in construction*, Vol. 12, 139-154.

Wang X. and Dunston P. S. (2007). Design, strategy, and issues towards an augmented reality-based construction training platform, *Journal of information technology in construction (ITcon)*, Vol. 12, 363-380.

Whyte J. (2002). *Virtual reality and built environment*, Architectural Press, Oxford, UK.

Wilkins B. and Barrett J. (2000). The virtual construction site: a web-based teaching/learning environment in construction technology, *Automation in construction*, Vol. 10, 169-179.

Wright R. S. and Lipchak B. (2004). *OpenGL SuperBible Third Edition*, Sams Press, Indianapolis, Indiana, USA.