Physically Based Real-time Simulation of an Automation Plant

Stefan Rilling and Gerrit Lochmann
Institute for Computational Visualistics
University of Koblenz
Germany
Email: rilling@uni-koblenz.de

KEYWORDS

digital factory; phyics simulation; real-time simulation

ABSTRACT

We present a system to simulate the dynamic object behavior of an automation plant within an interactive virtual environment. The area of application is the field of industrial training and educational software systems. Our contribution is a method to simulate the material flow through the plant in real time using a time discrete game physics simulator in combination with a petri net based state model. The dynamic behavior of virtual objects and the user interaction with the virtual environment is described using a component- and data flow based approach. We conducted several test series to assess the physical plausibility of our model.

INTRODUCTION

The adoption of virtual technologies has made its way into almost every area of industrial production. These technologies have been running through a considerable evolution, leading to a multitude of real time simulation and visualization techniques. Using these technologies for employee training has gained an increasing interest by the industry within the last years, leading to various research projects focusing on the topic, such as (Gerbaud et al., 2008), for example.

Within our research project, we investigate to which extent computer game technology is suitable to implement a realistic simulation of an automation plant and how a virtual training environment can be embedded in such systems.

Training simulations for the automation industry demand several requirements from the underlying software system. From a didactical point of view, it is desirable to have a lifelike and realistic virtual counterpart of the teaching content (Rilling et al., 2010), which implies, in the case of the automation industry, having a system which simulates and visualizes the dynamic behavior of an automation plant in real time. Besides the simulation aspect, interaction is the second requirement arising from the scenario. The virtual automation plant has to replicate all relevant user interactions of its real counterpart and respond to these user interactions.

We have a real automation testing plant at our disposal which gives us the opportunity to validate the outcome of our developed training environment within a realistic setting. This testing plant simulates procedures of the automation industry and consists of several modules responsible for the filling of small glass-bottles with solid parts, their closure with a cap, and their commission. The movement of the bottles through the plant is realized by conveyor belts, the flow control of the bottles is achieved by the means of track switches, stopping-, and separating stop elements. A single bottle is identified by the plant management system by a bar code which is attached on the outside of the bottle read by a bar code scanner. Bar code scanners are placed at discrete positions within the plant and serve to determine the position of the bottles.

RELATED WORK

Simulation models can be classified (Zeigler, 1985), amongst others, by the used time model (discrete or continuous), by the involved state variables (discrete, continuous or a combination of both) and by their interactivity (autonomous or non-autonomous). According to (Klingstam and Gullander, 1999), two main simulation approaches within the field of factory simulations can be found: discrete event simulation (DES) and geometric simulation.

Three-dimensional virtual environments (VE) are classified by the presentation form, the object behavior (Watt and Watt, 1991) and the interactivity. A realistic appearing VE involves an immersive presentation, dynamic object behavior, and is fully interactive. These systems are usually referred to as virtual reality.

The work of Choi et. al. (Choi et al., 2003) describes a virtual prototyping tool for an automated manufacturing system (AMS) simulator based on a DEVS and a 3D-visualization. The DEVS results from a preliminary conversion of a graphical modeling language (JR-net). The JR-net framework has been extended by virtual resources, a visual (3D model) and physical description (animation data) for the simulated factory modules. Virtual Resources comprise a state model as well as an animation controller, which implements the 3D visualization of the simulation results based on the approach of (Hwang and Choi, 2001), where the sequence of simulation events is synchronized with the animation data. A simulated stacker crane can be subject to user-defined

Proceedings 25th European Conference on Modelling and Simulation ©ECMS Tadeusz Burczynski, Joanna Kolodziej Aleksander Byrski, Marco Carvalho (Editors)
ISBN: 978-0-9564944-2-9 / ISBN: 978-0-9564944-3-6 (CD)

control by a scripting language.

The work of Moon et. al. (Moon et al., 2007) is another example of the virtual prototyping and system design field of application. A DES is used to simulate the interaction of resources, buffers, transporters and workers within an automotive transmission case line. The authors point to the usefulness of the involved 3D-visualization to evaluate spacial constraints and working areas.

Within the work of Mueck et. al. (Mueck et al., 2002), the problems arising from the connection of an interactive 3D visualization (called walkthrough system) to a DES concerning time synchronization are shown. The authors propose an architecture which separates the simulation time from the visualization time. An intermediate layer arranges for the interchange of messages between the simulation system and the visualization system. To provide a fluent visualization of the simulation results, a forerun time is given to the simulator. The algorithm presented in the paper makes sure that the time forerun is small enough to respond to user interaction, but large enough so that the visualization time does not outrun simulation time.

Thapa et. al. (Thapa et al., 2008) show a comparable approach to the connection of 3D visualization to a DES. For each simulation event, the involved graphical representation is selected and updated until visualization time meets the simulation time.

The previous work showed in this section has the application of a DES to describe the plants behavior in common. The problems arising with this approach are the synchronization of a 3D visualization and the incorporation of user interactions. Within the field of training simulations, these two factors are crucial.

1 SYSTEM DESCRIPTION

We formulated an entity-based description methodology in which atomic elements, called *Dynamic Object (DO)*, form the basic structural element. In the field of game engines, an entity-based classification of the virtual world is common practice. In our system, a DO is defined as an item in the virtual world, which has a perceivable, distinct behavior, resp. an influence on the behavior of the virtual world. The DO acts as a container element which encapsulates several *components*. The components themselves comprehend the according functionality.

Data flows

Alongside components, *data flows* are another key element within our system. Data flows are used to implement communication among the various components of a DO and actually among DOs themselves. For example, the position and orientation of a physics component can be written to the position and orientation of a graphics component using the data flow mechanism. A data flow connects one *input slot* with one *output slot* at a time (c.f. Figure 1), whereas one input- or output slot can take part in several data flows. A component can send data via an input slot, and receive data via an output slot. Input and output slots are typed, only slots of the same type can be directly interconnected. In addition, there is a *void slot*, which does not transport any data and which can be used to implement event mechanisms.

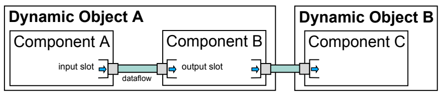


Figure 1: Data flow between objects and components. Data flows are marked as blue boxes and connect one input slot with one output slot at a time. The arrows within the slots denote the data flow direction. Arrows pointing outwards of the slot denote output slots, arrows pointing inwards denote input slots.

Data flows, input- and output slots are identified via a unique ID, so that the input- and output slots of a specific data flow can be denoted by a combination of the DO's ID, the component's ID and the slot ID. This makes the description of the whole data flow network on an ID-basis possible and hence arranges for the extensibility of the architecture.

The execution of a data flow can depend on the activity state of the involved in- and output slots. Four possible execution conditions can be defined: The input slot needs to be active, the output slot needs to be active, the input slot and the output slot need to be active, he input slot or the output slot needs to be active.

Furthermore, the execution of a data flow can occur in a *continuous* or *singular* manner. Continuous data flows are executed with each update step of the runtime environment, singular data flows are executed only once and then deactivated until their reactivation is triggered.

Several data flows can be connected by logical *data flow connectors*. A data flow connector contains two output slots serving as signal input and one input slot serving as signal output (cf. fig. 2). Each of the two signal inputs can be negated. In- and output slots are void slots, which means that no data is transported through the connector. The connector performs a logical operation (AND, OR) on the activity status of its two signal inputs within each update step. If the logical operation results in TRUE, the signal output is activated.

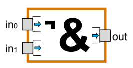


Figure 2: Logical data flow connector. The input signal in_0 is inverted. The connector activates its output as soon as in_0 is deactivated and in_1 is activated simultaneously.

Components

Components encapsulate functionality of the different middleware used within a virtual environment, such as 3D-graphics engines or physics engines, and provide an interface to these self-contained software systems using the input- and output slot system. Each component comprises basic functionality, e.g. registering and querying input- and output slots. The implementation and integration of own specialized components can be realized via a plugin-system, which arranges for the extensibility of the system and enables the integration of various middleware systems.

We developed different components to needed to model the factory simulation with the project-specific software rendering and simulation systems. We give an overview of these components below.

The graphics component describes the visual properties of the virtual object, such as geometry data, position and orientation. A DO can consist of several graphics components, arranged in a transformation hierarchy. The actual geometry data is not included into the dynamic object description. Instead, the data is referenced by a simple alphanumeric URL, which has to be interpreted by the run-time environment parsing the object description. This mechanism eases the integration of various rendering systems. At a minimum implementation, the graphics component provides input- and output slots for position and orientation.

Simulation components describe time-discrete simulation objects, whose state variables are advanced over the simulation time by their supervising simulation system. Within our system, simulation components generally include position and orientation within their set of state variables. Thus, the simulation component provides input- and output slots for position and orientation, whereas the position is stored as a three-dimensional vector and the orientation is expressed as a rotation quaternion. A *physics simulation component* is a specialization of a general simulation component, which encapsulates the physical description of a virtual object. The physical description includes rigid body properties, the collision model as well as mechanical constraints between two physics simulation components. Furthermore, physics simulation components can be equipped with box shaped force field volumes with an extension $\vec{e} \in \mathbb{R}^3$, a direction $\vec{d} \in \mathbb{R}^3$ with $\left|\vec{d}\right| = 1$, and a target speed $v_{tar} \in \mathbb{R}$.

The physics simulation component provides several input slots to provide access to the several simulation state parameters like position, orientation or linear and angular velocity. Furthermore, the component's ability to collide with other physically simulated objects can be enabled and disabled via a data flow.

A DO's state component represents a set of finite states and the transitions between these states. Our system implements a Petri-Net based state machine. A Petri net is a tupel $(P, T, \mathbb{F}, \mathbb{B})$ where P is the set of places, T is the set of transitions, $\mathbb{F}$ and $\mathbb{B}$ are the *forward-* and *backward-matrix*. We refer to literature (e.g. (Priese and Wimmel, 2008)) for a more comprehensive overview on the topic. A DO's state is defined by the marking of the corresponding state component's net. Within our system, the petri net's firing behavior in the case of forward conflicts (cf. (Nielsen et al., 1981)) defers from the specifications found in literature: Active transitions always fire at the same time, resulting in a deterministic behavior of the net (cf. fig. 3).

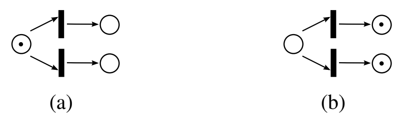


Figure 3: Firing behavior for forward conflicts: A token on an input place activates both transitions shown in (a), which thereafter fire simultaneously, resulting in the marking shown in (b).

Within the state component, an output slot which triggers the placement of a token, and an input slot which is activated by a placement of a token, is provided for each place. With this system, state transitions can be triggered via data flows, and data flows can be triggered via state transitions. Figure 4 shows an example representing an on / off state component. The state transition is triggered by a control state using its assigned output slot.

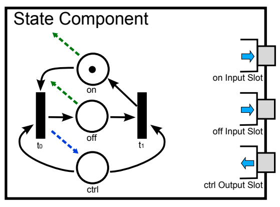


Figure 4: An example state component with the corresponding petri net. For the sake of clarity, only the relevant in- and output slots are shown. Places with outgoing dotted arrows are related to an input slot, places with incoming dotted arrows are connected to an output slot.

The trigger component reacts whenever a DO resides within the trigger's area of influence. The component provides output slots, which allow the connection of specific trigger mechanisms within the connected middleware, as well as input slots, which are activated as soon as the trigger component notices a DO. The trigger component reacts whenever a DO resides within the trigger's area of influence. As specific trigger mechanisms can be found within a multitude of existing middleware like 3D engines or physics engines, an abstract interface to connect these specific trigger mechanisms is provided by the component. A volume trigger component informs the virtual environment whether an object is entering, staying inside, or leaving the volume defined by the trigger component. In each case, one of the component's corresponding input slots is activated, thus a data flow can be initiated by a volume trigger. The slots are named *"'on enter"'*, *"'on stay"'* and *"'on leave"'*, respectively.

MODELING THE AUTOMATION PLANT

Basically, two types of objects determine the plant's behavior: actuating elements and the sensory system. The group of actuating elements includes the system of 18 interconnected conveyor belts, switches, separators, and robotic arms. The plant's sensory system comprises binary sensors and ID-readers. In the following sections, we describe how these elements are modeled with our system in detail.

Conveyor Belts And Virtual Bottles

Conveyor belts are made up of a moving rubber band where the bottles stand upon and are moved with the band due to the large amount of friction between the rubber and the glass. Metal guard rails are mounted at the sides of the conveyor belts, preventing bottles from falling off the track.

We modeled the conveyor belts using rigid bodies to model the guard rails and force fields to simulate the friction between the rubber band and the bottles. Although a simple approach, this method turned out stable and showed good results regarding the physical plausibility. The collision model of the guard rails was modeled using only the basic geometric primitives box and sphere. Each conveyor belt is represented by a DO with a force field physics component. Figure 5 shows a schematic overview of the model.

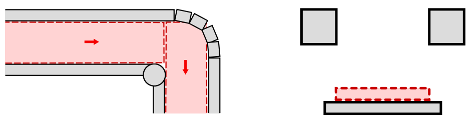

(a) top view (b) section view

Figure 5: Schematic view of a conveyor belt. The rigid body boundary elements are painted gray, the force fields are painted red. The red arrow indicates the force field direction $\vec{d}$.

The force field applies a force $\vec{F}$ at the center of mass of each physics component remaining within the force field's volume of influence. The force $\vec{F}$ is calculated by a proportional control function

$$\vec{F} = k \cdot M_v \cdot (\vec{v}_{tar} - \vec{v}) \qquad (1)$$

with the current physics component's linear velocity $\vec{v}$, the user defined target velocity $\vec{v}_{tar}$ to which the objects are accelerated, the diagonal matrix of the force field's direction vector $M_v = diag(\vec{d})$, and a scalar multiplication factor k.

The DO of each bottle consists of one graphics component and two physics components. Different 3D models are attached to the graphics component depending on the bottle's fill state, i.e. all combinations of filled / empty and capped / uncapped. The collision shape of the primary physics component is represented by the convex hull, while the inertia tensor is approximated by a homogeneous cylindric shape. As the force field volume representing the conveyor belt only affects the physics component's center of mass, the applied force does not grip at the bottle's bottom. To achieve an authentic torque during the acceleration process, the bottle contains a second

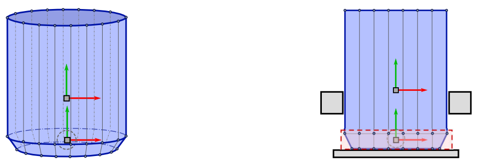

(a) collision model (b) bottle on conveyor belt

Figure 6: The collision model of a bottle is shown on the left. In the right image, the configuration of a bottle on a conveyor belt is shown. The centers of mass of the two physics components are shown by their respective local coordinate system.

physics component in the center of its bottom, bound to a fixed constraint which removes all 6 degrees of freedom of relative movement. (c.f. fig. 6).

ID-reader And Binary Sensors

The plant's sensory system is responsible for the control of the several actuators' movement and thus controls the flow of bottles throughout the plant. Bottles are identified by a bar code ID which is read by the ID-reader sensors and submitted to the plants control software. Binary sensors emit a signal depending on whether a bottle is located within their measuring area. We use dynamic objects equipped with volume trigger components to simulate both types of sensors.

Switches

There are five different switches located at the branch connection points within the plant's conveyor belt system. A switch can take two different positions: The base position and the working position. To each switch, a related ID-reader is located nearby, reporting the incoming bottle ID to the plant's management software, which in return arranges the switch's position depending on the bottle's destination.

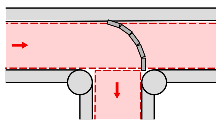


Figure 7: A switch located at a T-junction of two conveyor belts. The switch is shown as an arcuated darkgray rigid body comprised of several box shaped collision shapes.

Switches are modeled as DOs aggregating a graphics component, a physics simulation component and a state component. The graphical representation of the switch includes two predefined animations, the switch's movement from the base position to the working position and vice versa.

The switch's physical representation is made of a physics component made of several box collision shapes arranged to comply with the arctuated shape of the original switch blade (c.f. fig. 7). We choose this simplified collision model instead of a concave collision mesh to simplify the collision test and to increase the physical stability of the simulation, as most real-time physics engines handle simplified collision geometries more effectively. The physical switching behavior is realized by simply enabling or disabling the collision ability of the switch's physics component.

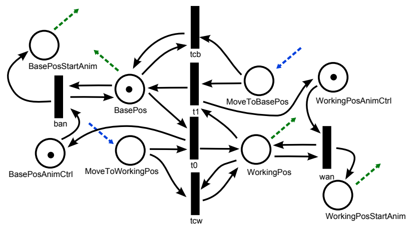


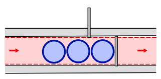
Figure 8: The switch state model shows a configuration where the switch is set to its *BasePos* state and is ready to start the accordant animation which will be played when transition *ban* fires and places a token on *BasePosStartAnim*. The places with blue incoming arrows are controlled by data flows, the places with green outgoing arrows initiate data flows. The transitions *tcw* and *tcb* arrange for no token accumulation on *MoveToBasePos* and *MoveToWorkingPos* respectively.

The state component of the switch is responsible for the animation control as well as for the control of the activation / deactivation of the switches collision representation (cf. fig. 8). The control of the physics component is realized by the places *BasePos* and *WorkingPos*, whose accordant input slots are connected via data flows to the physics simulation component's output slots responsible for the activation or deactivation of the collision ability. The state transition from *BasePos* to *WorkingPos* and vice-versa is triggered by the places *MoveToBasePos* and *MoveToWorkingPos*, respectively, whose activations by token placement is controlled by the plant's transport system simulation. Furthermore, the state component controls for the playback of the graphics component's animation. As soon as a token is placed on *BasePosStartAnim* or *WorkingPosStartAnim*, respectively, the accordant animation is played. The control places *BasePosAnimCtrl* and *WorkingPosAnimCtrl* prevent the playback of the animation if the switch has already reached the accordant state.

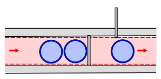
Separators

Separators are actuating elements which separate bottles to the following section of the conveyor belt system. A bottle is let pass as soon as the section behind the separator is ready to carry a new bottle, which is controlled by a combination of binary sensors. Figure 9 shows a schematic overview of the separation process.

Each separator is modeled, similarly to switches, as a DO aggregating various components. Two physics simulation components, whose collision ability is activated in an alternating manner, are used to simulate the gates' behavior. A state component is connected to the different binary sensors via data flows and thus responsible for the control of the two physics simulation components. A timer component induces the speed of a separation cycle.

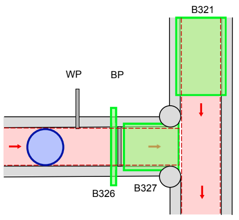


(a) base position

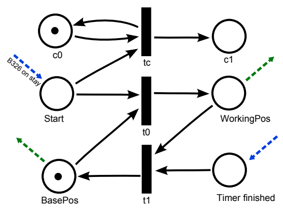


(b) working position

Figure 9: The separation process is realized by two gates with an alternating opening and closing behavior.



(a) separator and sensors



(b) separator state model

Figure 10: A separator at a T-junction with its corresponding two gates (*BP* and *WP*) as well as the associated binary sensors (*B321*, *B326* and *B327*) is shown in the left image. The separator's state model is shown in the right image. The token placement of *Start* is connected to the B326 sensor's volume trigger component.

Figure 10 shows the separator's state model as well as an overview of a separator placed at a T-junction and the configuration of the binary sensors. A sensor placed near the base position gate initiates the separation process as soon as a bottle stays inside the sensor volume. The extension of the sensor volume in conveying direction has to be chosen adequately small in order for the sensor to work correctly.

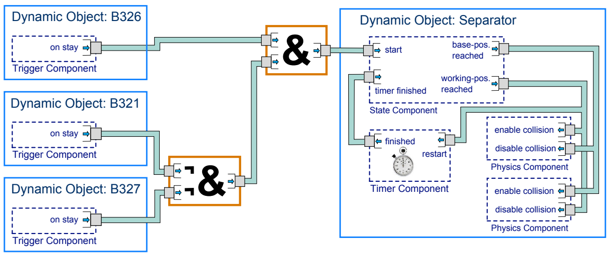


Figure 11: DO-model

Within this example, the separator should start working when sensor *B326* reports a bottle within its volume and sensors *B327* and *B321* report no bottle within their measuring zones. In Figure 11, the DO model of the separator and the sensors, and the data flows involved is shown.

Robotic Arms

A robotic arm transfers bottles from a storage to a conveyor belt. Within our simulator, the robotic arm is animated as a compound of preset skeletal animations and dynamic motion. Positions of the skeletal bones can be recalled and manipulated during the running animation through the graphics component. The robot's gripper is represented as a DO comprising the trigger's graphics component, a volume trigger component and a specialized gripping component. The trigger is attached to the gripper via a data flow connecting the position input- and output slots of the graphics- and trigger component. As soon as a bottle enters the trigger volume, the bottle is reported to the gripping component which takes the physics simulation component out of the dynamics simulation and installs a data flow to make the bottle follow the gripper's position. When the target destination is reached, this data flow is disconnected and the bottle's behavior is reset to dynamic mode.

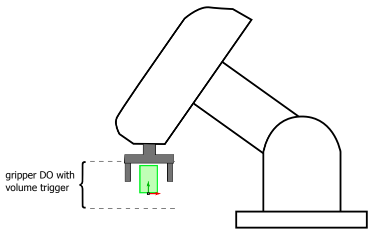


Figure 12: Schematic view of the robotic arm with the connected volume trigger component at the robot's gripper.

RESULTS

Please take a look to our visual presentation of the details at http://uni-koblenz.de/cg/PlantSim . This text will not be included within the final paper (if accepted).

To judge the physical plausibility of the bottle simulation, three scenes have been set up. We compare the bottle transportation process simulated with our model to the real-world situation. In the first test we measure the acceleration in respect to slipping and traction. In the second test we show the appearance of torque when putting bottle a bottle on the running conveyor belt, in a third test we measure the distance between bottles, which are conveyed on a complex track over a longer period. The tests run with a physics simulation frequency of 150 Hz, while all coefficients of friction and restitution are set to 0.0.

Effect of Friction

On a real conveyor belt accelerating an object, the linear momentum of this object increases by the friction between the object and the conveyor belt. The amount of slipping depends on the coefficient of friction. As our model uses force fields, no friction is applied. Instead, the accelerative force is scaled by the multiplication factor k of the force field control function. Set to a low value (e.g. $k = 1.0$), the bottle slowly accelerates until its velocity approximates the target velocity of the force field. In the real transport system, this effect is not noticeable, as the bottles apparently accelerate with an almost non-slip traction. Hence, we set $k = 16.0$ to adapt this behavior.

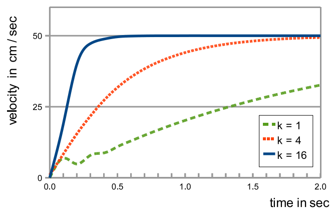


Figure 13: In three test runs we measured the acceleration of a bottle dropped on a conveyor belt simulated by a force field with a maximum velocity of 50 cm/sec. A low multiplication factor $k = 4$ corresponds to a low coefficient of friction, while a high value (e.g. $k = 16$) simulates a non-slip traction.

Torque Caused By Acceleration

The acceleration at the bottom of a bottle causes a torque, which leads to a rocking motion when dropped on the conveyor belt. A large bearing area, a low center of mass and a high mass value increase the creeping strength of the bottle. In this test run, we measured the angle of inclination of the bottle depending on its mass. To enhance the effect, we increased the velocity of the conveyor belt to 100 cm/sec. Figure 14 shows the result of the test run. A plausible behavior can be observed: The lower the mass, the heavier the rocking motion of the bottles gets.

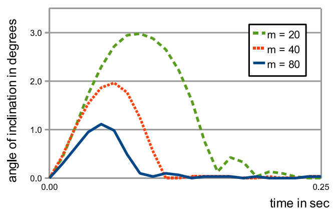


Figure 14: In three test runs we observed a bottle accelerated by the conveyor belt with a target velocity of 100 cm/sec. We measured the absolute angle between the plumb-line and the inclination in the direction of motion. The bottle with a mass of $m = 80$ grams inclines by ca. 1 degree, while a lightweighted bottle ($m = 20$) inclines by ca. 3 degree and by ca. 1/3 degree in a second motion, before it changes to creep behavior.

Constant Distances

In a third test run, we observed the aspect of stability of the simulation over a longer period of time. Two bottles

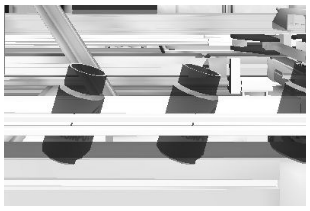


Figure 15: A screenshot of our application showing bottles dropped on a conveyor belt. Parameters are augmented to enhance the effect of inclination for visualization.

are conveyed in a loop with 1 left and 5 right turns as well as 5 straight lines with a length between 20 and 500 cm. We measured the deviation of the bottle's Manhattan distance over 50 seconds starting with a distance of 20 cm. The target velocity of the conveyor belt is set to 50 cm per second in respect to the maximum velocity in the real setting. The results show alternations in distance between 15.8 and 22.4 cm with a variance of 1.6. Thus, the collision behavior is susceptible to small geometric meanderings, especially during turns.

CONCLUSION

We presented methods to simulate the dynamic behavior of an automation plant within a three dimensional virtual environment, as well as a system which implements these methods by the means of components and data flows. We were able to build a lifelike model of an existing automation plant, where the material flow and the plant's actuating elements like conveyor belts, separators, and switches are described by physical properties. The usage of well established middleware from the field of virtual reality and video games allows an interactive simulation of the plant's behavior in real-time. Hence, the simulation speed comes at the cost of simulation precision. However, we showed that our approach at least leads towards a physically plausible behavior of the material flow simulation.

We implemented a training simulator for automation plants using our system. Within this scenario, interactivity and real-time presentation is of larger importance than precise simulation results.

For further research, an integration of the proposed simulation model into existing digital factory concepts. An automated, or at least semi automated conversion of already present factory simulations and mechatronics descriptions of factory components is eligible. Furthermore, an increase in precision regarding the physical simulation while maintaining real-time simulation behavior is desirable, especially the need to rely on simplified collision models has to be evaded.

As the validation of the simulation results is only conducted visually by comparison of the real plant to the simulated one, a profound analysis should be in the scope of future work.

REFERENCES

Choi, B.-K., Park, B.-C., and Park, J.-H. (2003). A formal model conversion approach to developing a devs-based factory simulator. *Simulation*, 79(8):440–461.

Gerbaud, S., Mollet, N., Ganier, F., Arnaldi, B., and Tisseau, J. (2008). Gvt: a platform to create virtual environments for procedural training. In *VR*, pages 225–232.

Hwang, M.-H. and Choi, B.-K. (2001). Gk-devs: geometric and kinematic devs formalism for simulation modeling of 3-dimensional multi-component systems. *Trans. Soc. Comput. Simul. Int.*, 18:159–173.

Klingstam, P. and Gullander, P. (1999). Overview of simulation tools for computer-aided production engineering. *Computers in Industry*, 38(2):173 – 186.

Moon, D. H., Xu, T., Baek, S. G., Lee, J. S., and Shin, W. Y. (2007). A simulation study of the transmission case line in an automotive factory. In *Proceedings of the 2007 spring simulation multiconference - Volume 3*, pages 24–29.

Mueck, B., Dangelmaier, W., Fischer, M., and Klemisch, W. (2002). Bi-directional coupling of simulation tools with a walkthrough-system. In Schulz, T., Schlechtweg, S., and Hinz, V., editors, *Simulation und Visualisierung*, pages 71–84, Ghent, BE. SCS European Publishing House.

Nielsen, M., Plotkin, G., and Winskel, G. (1981). Petri nets, event structures and domains, part i. *Theoretical Computer Science*, 13(1):85 – 108.

Priese, L. and Wimmel, H. (2008). *Petri Netze*. Springer, 2 edition.

Rilling, S., Wechselberger, U., and Mueller, S. (2010). Bridging the gap between didactical requirements and technological challenges in serious game design. *Cyberworlds, International Conference on*, 0:126–133.

Thapa, D., Park, C. M., Han, K. H., Park, S. C., and Wang, G.-N. (2008). Architecture for modeling, simulation, and execution of plc based manufacturing system. In *Proceedings of the 40th Conference on Winter Simulation*, WSC '08, pages 1794–1801. Winter Simulation Conference.

Watt, A. and Watt, M. (1991). *Advanced animation and rendering techniques*. ACM, New York, NY, USA.

Zeigler, B. P. (1985). *Theory of Modelling and Simulation*. Robert E. Krieger Publishing Company, Inc.

AUTHOR BIOGRAPHIES

STEFAN RILLING, born 1979 in Germany, is a computer scientist working at the Institute for Computational Visualistics of the University of Koblenz, Germany. His research interests cover the field of dynamic object behavior and interaction within virtual environments, realtime simulation for the digital factory, as well as game based learning and training.

GERRIT LOCHMANN, born 1986 in Germany, is a master student of computer science at the University of Koblenz, Germany. He works as a student research assistant for the Institute for Computational Visualistics working group since 2007.