Towards Spatial Databases using Simulation State Transformations

Providing the Means for High Performance Simulation of Digital Twins in Three-Dimensional Scenarios

Moritz Alfrink and Jürgen Roßmann

RWTH Aachen University, Institute for Man-Machine-Interaction,
Aachen 52074, Germany,
alfrink@mmi.rwth-aachen.de,
www.mmi.rwth-aachen.de

Abstract. The growing complexity of industrial plants, enormous advances in autonomous driving and an increasingly digitized world require a quickly rising amount of virtual planning and testing using three-dimensional simulations. In addition to the number of technical advances, the complexity of individual objects and scenarios is increasing, pushing the requirements on simulation technology to generate sufficient realistic data within limited time frames.

Simulation frameworks have their means to store the information necessary to represent the properties of all scenario entities – the simulation state – in their database. However, they lack simulation algorithm-dependent transformations and augmentations of the simulation state that allow maximizing calculation efficiency. This paper introduces a generalized formalism to describe these transformations and illustrates how they help to bring the simulation framework closer to a specialized simulation database that stores spatial information with a focus on performance – the spatial database.

Further, this paper demonstrates the advantages of the approach with the introduction of two concrete transformations. One targets the efficient representation of three-dimensional spatial relations for industrial robots while the other allows generating different levels of complexity in the definition of materials in the context of autonomous driving.

Keywords: spatial database, simulation state transformation, high performance, digital twins, three-dimensional scenarios

1 Introduction

Simulation of large scale, three-dimensional scenarios has become a major demand of Industry 4.0. The introduction of Digital Twins storing complex geometric and dynamic data requires specialized databases, able to store such information efficiently. Furthermore, scenarios as Hardware in the Loop (HIL), including sensors as LIDARs, cameras or other spatial sensors, require sufficient

realistic data while maintaining real-time capabilities. To meet both needs, a combination of ray tracing and rasterization algorithms must be used. Another field which demands efficient spatial data structures are dynamic simulations. Since the geometry approximations in the dynamic simulation have significant influence on precision, complex geometry approximations are required for realistic dynamic simulations. To handle collision detection for such complex geometries, efficient collision queries are a precondition to enable interactive modeling. Furthermore, increasingly specialized hardware is developed to accelerate various tasks in the domain of ray tracing, rasterization, general vector operations and machine learning tasks. To exploit these devices, the input data must be transformed into a device-specific form. This paper introduces a generalized approach to satisfy the proposed constraints on real-time performance by exploiting recent hardware acceleration technologies and efficient data structures leading to a spatial database. To reach this aim, a database transformation formalism is introduced and applied to two challenges of spatial simulation.

2 Related Work and Theory

In this section the theoretical bases for this work are explained. Beginning with the idea of spatial database, continuing with a formalism to describe simulation systems and models and finally introducing a formalism to describe simulation state transformations.

The idea of spatial databases is to transform or augment the information in a database - the simulation state - with respect to hardware capabilities, performance constraints and requirements on precision in a way that either three-dimensional simulation algorithms can process them more efficiently or to enable them in the first place by providing a specialized representation of the information.

In general, a simulation has the purpose to create an environment, in which a system with all its dynamic processes can be analyzed as an experimentable model in order to reach findings which are transferable to reality [10]. Moreover, since simulations can be arbitrarily complex, it is quite difficult to find a common ground or a superordinate system to describe them. One approach proposed by [11] is the formalism of Discrete Event System Specification (DEVS). DEVS can be applied to implement behaviors defined by basic system formalisms such as discrete time and differential equations. The DEVS structure is defined by

$$M = \langle U, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta \rangle \quad (1)$$

where

U	is the set of input values
S	is a set of states
Y	is the set of output values
$\delta_{\text{int}} : S \rightarrow S$	is the internal transition function
$\delta_{\text{ext}} : Q \times X \rightarrow S$	is the external transition function, where
$Q = \{(s, e) s \in S, 0 \leq e \leq ta(s)\}$	is the total state set
e	is the time elapsed since last transition
$\lambda : S \rightarrow Y$	is the output function
$ta : S \rightarrow \mathbb{R}_{0,\infty}^+$	is the time advance function

The proposed system M is in a state $s \in S$ at any given time. In case no external event occurs, the system remains in its current state for a period $ta(s)$. At the end of the period $ta(s)$, the system output value is evaluated by $\lambda(s)$ and the state changes to $\delta_{\text{int}}(s)$. For the case that an external event $u \in U$ occurs during $ta(s)$, the system changes its state to $\delta_{\text{ext}}(u, e, s)$.

In order to introduce a concept of time to the DEVS, [11] defines time as $t = (T, <)$ where T is a set and $<$ is an ordering relation of elements of T with $<$ transitive, irreflexive and antisymmetric [11, p. 128]. Furthermore, the trajectory $\omega : \langle t_1, t_2 \rangle \rightarrow A$ with A an arbitrary set, describes the motion through the set A beginning at t_1 and ending at t_2 for every $t \in \langle t_1, t_2 \rangle$. The motion at time t is defined by $\omega(t)$. The second formalism in [11] describes an input/output system as

$$S_{io} = (T, U, \Omega, Y, Q, \Delta, \Lambda) \quad (2)$$

where

T	is a base time
Ω	is a set - the set of allowable input segments
Q	is a set, the set of states
$\Delta : Q \times \Omega \rightarrow Q$	is the global state transition function
$\Lambda : Q \times U \rightarrow Y$ (or $\Lambda : Q \rightarrow Y$)	is the output function

The essential functions are $\Delta(q, \omega)$ and $\Lambda(q, u)$ with $q \in Q$, $u \in U$ and $\omega \in \Omega$, which describe the state transition function and the output function respectively. In this definition, $\omega(t)$ is the input function. Figure 1 visualizes the system S_{io} .

Fig. 1: Input/output system

The introduced input/output system represents arbitrary simulation functions in

a generalized form. In a simulation framework, both functions map to algorithms implemented in software or specialized hardware, leading to a system or model specific behavior. However, depending on available hardware, requirements on precision or aim for closeness to reality, the specialized forms $\hat{\Delta}$ of Δ and $\hat{\Lambda}$ of Λ should be used to increase performance and to meet the required precision. The application of $\hat{\Delta}$ and $\hat{\Lambda}$ also requires the adapted version $\hat{q}$ of q .

To transform q to $\hat{q}$ this paper introduces the input transformation δ_Δ of the state transition function Δ and the input transformation δ_Λ of the output function Λ as follows

$\delta_\Delta : Q \rightarrow Q_\Delta \subseteq Q \cup I_{env}$, the input transformation of the state transition function
 $\delta_\Lambda : Q \rightarrow Q_\Lambda \subseteq Q \cup I_{env}$, the input transformation of the output function

To augment the state q information I_{env} about the simulation algorithm or the hardware executing the algorithm can be used to transform the internal representation. Transforming q can also mean to remove information irrelevant for the algorithm, while adding information relevant for the algorithm. Figure 2 visualizes the augmented system.

Fig. 2: Augmented input/output system

The last element to complete the picture is the definition of simulation state q . For a simulation framework [7] proposes the generalized model state to be defined as

$$q = (\underline{x}, \underline{a}, \underline{A}) \quad (3)$$

with

$$\begin{aligned} \underline{x} &\in \mathbb{V}_A^{n_A} && \text{model state} \\ \underline{a} &\in \mathbb{V}_a^{n_a} && \text{model parameters} \\ \underline{A} &\in \mathbb{V}_A^{n_A} && \text{model algorithms} \end{aligned}$$

Given the theoretical basis of the DEVS, its specialized form - the input/output system - the transformations δ_Δ and δ_Λ and the definition of q , the next section illustrates with two examples how these transformations would look like augmenting a simulation state. In both examples the aim is to increase performance and efficiency doing the special tasks of spatial collision queries and rendering materials.

3 State Transformations

This section introduces two state transformations showing the superiority of using transformed states $\hat{q}$ in the system functions $\hat{\Delta}$ and $\hat{A}$. The state transformation design is derived from the requirements which three-dimensional simulation scenarios including real-time, sensor simulation, rendering and dynamic simulation impose.

The first simulation state augmentation aims on increasing the efficiency of spatial collision queries, ray tracing and occlusion queries. The common basis of these methods is that they need to determine which objects or geometries are in particular region of the scene. While ray tracing has a lot of application in sensor simulations for LIDARs or ultrasonic sensors, collision queries are a major part of rigid body simulations and occlusion culling in render algorithms. All together they play a primary role in the scenarios mentioned in section 1.

Given the general definition of q from section 3, the model state $\underline{x}$ contains the information about the object geometries, poses, the physical properties such as mass, material, etc. The model algorithms $\underline{A}$ are among others: ray tracing algorithms, collision detection algorithms and occlusion culling algorithms.

Since all algorithms require an efficient representation of spatial relations in the scene, the first proposed augmentation of q adds Space Partitioning Structures (SPS) to the model state $\underline{x}$, leading to the augmented state $\hat{q}_{SPS}$. $\delta_{\Delta,SPS}$ and $\delta_{A,SPS}$ are in this case the same function, containing an algorithm which creates the SPS from the geometry and pose information in $\underline{x}$. An SPS divides the three-dimensional space into a hierarchical structure of geometry groups, which are close to each other as shown in figure 3 left. This allows highly efficient broad intersection queries due to the hierarchic structures, similar to a tree as shown on the right of figure 3, which allows skipping a vast amount of geometry intersection tests.

Fig. 3: Two-dimensional Bounding Volume Hierarchy

While there exist many SPS such as Binary Space Partitioning (BSP) [8], Oc-trees [1] and Bounding Volume Hierarchies (BVH) [3], not all support efficient building algorithms making use of multi-core systems, which is required when dealing with dynamic scenarios. A very efficient BVH construction algorithm was developed by [12]. It allows to build a BVH in linear time, in parallel on CPU or GPU and is therefore be used for further implementation.

BVH recursively enclose larger groups of geometries in axis aligned bounding boxes, beginning with a single geometry and ending with one root box, enclosing the entire scene. Figure 4a and 4b illustrate how the BVH encloses the objects more and more precisely as the tree depth increases. To trace a ray through the BVH, the ray is initially tested against the root box. On intersection, the tree structure is tested recursively, using the depth first approach. The two individual child nodes of each node are traversed in order of the nearest intersection. To find broad intersections between geometries, the geometry to test is enclosed in a box, which is recursively tested against the BVH beginning with the root. A leaf node intersection while testing against the BVH gives a potential collider, which needs to be tested using a narrow collision algorithm. Using an SPS many of broad range problems can be solved efficiently.

Fig. 4: BVH enclosing scene objects

Figure 5 shows how the BVH can be used to accelerate collision tests in the example of a reconfigurable working cell [6], by successively test for collision against smaller groups of objects. The Digital Twins of the robots execute the task of picking and placing work pieces into specialized machines. Those process the work pieces and the robots move the processed items to the target carriage. To determine whether the robot can pick a work piece an intersection query has to be performed. This test is split into a broad and a narrow phase. Before performing the narrow collision test, the broad hierarchical test is performed, testing against each level of BVH from top to bottom as explained before.

The next proposed augmentation of q considers $\underline{A}$ and $\underline{x}$, targeting the materials used to describe the properties of the objects in the scenario. The material description has influence on how sensors perceive objects and how objects are visualized. This is especially important for simulated cameras and LIDARs. However, while these sensors require very complex material definitions, others, like contact sensors, only required very basic material information. This requires a general material definition from which different levels of complexity can be derived using the output transformation $\delta_{A,MDL}$. Since $\delta_{A,MDL}$ depends on the hardware used for rendering, I_{env} contains information about its raster and ray tracing capabilities. For each derivation, the rendering functions expressed in shaders and contained in $\underline{A}$ have to be adapted depending on the complexity of the material in $\underline{x}$ and I_{env} . One material definition, which allows this deriva-

Fig. 5: BVH in reconfigurable working cell

tion, is the Material Definition Language (MDL) by NVIDIA [2]. MDL allows defining arbitrarily complex materials and deriving material models with fixed complexity depending on the users' requirements. Figure 6 shows the usage of different material derivations - generated from a single material definition - in different render techniques.

Fig. 6: Subset of the distilled static material model

4 Implementation

This section describes the implementation of $\delta_{A,MDL}$ using the MDL to transform the simulation state q depending on the available hardware. The transformation is required to provide the necessary information for different rendering techniques such as ray tracing or rasterization, which represent the output functions A_{rt} and A_{ra} , respectively. The proposed implementation allows deriving the required material representations from a single material definition. In the chosen implementation, the rasterizer uses the OpenGL API [4] and the ray-tracer uses the Optix API [5]. Moreover, the implemented rasterization approach is able to

switch between procedural and baked material rendering (as shown at the end of section). The ray tracing is realized with Optix, which is a fully customizable ray tracing framework using GPU hardware and software acceleration. Rasterization is implemented in GLSL using OpenGL as graphics API.

The MDL material definition uses recursively-refined expressions to describe the surfaces, beginning with a basic layer consisting of a surface, volume, index of reflection, thin walled flag and a geometry information. Every surface expression in the MDL can be a bidirectional scattering distribution function (BSDF) leading to a structure shown in figure 7 on the left.

Fig. 7: Different MDL complexity representations

Since these BSDF can only be evaluated in a ray-tracer, the rasterizer requires a different representation. For the chosen implementation, the following expressions are evaluated during rasterization: Anisotropy, anisotropy rotation, attenuation color, attenuation distance, base color, clearcoat normal, clearcoat roughness, clearcoat weight, metallic, normal, opacity, roughness, specular, subsurface color, transmission color, transparency, volume index of reflection. To retrieve the expressions, the NVIDIA MDL distilling API is used. The distiller generates the simplified expressions for the required static material model.

The generated static material model can be used in two ways: The first is the run-time evaluation of the expressions as figure 7 middle shows, leading to materials with a resolution up to the precision of the underlying floating representation. Moreover, run-time evaluation enables the usage of procedural material, which use noise functions to generate permutations of different base material layers, resulting in a very realistic and nonrepetitive structures in the material. The second way is to evaluate the expressions within a fixed range and store the result in a texture as figure 7 right shows. This process is called baking and has the advantage that complex run-time calculation (depending on the material) are reduced to a simple texture lookup. The downside is that the baked textures have a limited maximum size, resulting in a fixed sampling frequency and region to be captured. This is especially problematic for procedural textures, since the baking removes their nonrepetitive structure.

In summary, the simulation state transformation is implemented to derive three

material representations from a single definition, leading to a flexible, target device-specific material representation.

5 Evaluation

In this section, the performance of the output functions A_{rt} A_{ra} introduced in section 4 is evaluated. A_{ra} uses rasterization with either baked or run-time evaluation of the static material model. The measurement consists of two scenarios. The first one is a test scene containing simple geometry with 12016 triangles, which use MDL materials. The second one is the urban test drive scenario shown in figure 6 with complex geometries containing 3085073 triangles. To evaluate the performance the rendering time of one frame Δt is measured. The measured results in table 1 indicate a big difference in render times between ray tracing and rasterization, which renders ray tracing impracticable on weaker GPU devices. The simulation state transformation is therefore the enabling step for the visualization on weaker devices.

Renderer	Method	$\overline{\Delta t}$	min Δt	max Δt	% of max	triangles
GLSL Baked	raster	16	14	20	100	3085073
GLSL Expr	raster	33	28	36	206	3085073
Optix	ray tracing	164	150	190	1025	3085073
GLSL Baked	raster	4	3	4	100	12016
GLSL Expr	raster	5	4	6	125	12016
Optix	ray tracing	34	32	45	850	12016

Table 1: Measured frame times in *ms* using NVIDIA GTX 1080 GPU

6 Summary

This paper introduces the idea of spatial databases as the transformation or augmentation of the information in a database in a way that three-dimensional simulation algorithms can process them more efficiently creating a spatial database. Further the idea is expressed by extending the DEVS formalism by the simulation state transformations δ_Δ and δ_A . Using the formalism two state transformations aiming on increasing computation efficiency and on creating hardware depended representations of q are introduced. For the latter the implementation and evaluation is described. The results gathered from the implementation of the MDL augmentation indicate the need for variable simulation state representations to support multiple levels of hardware compute capabilities. The next steps are to investigate in different levels of geometry representations such as signed distance field or voxelization to provide an even wider range of simulation state representations, enabling more devices to use spatial databases in real-time applications.

References

1. Doctor, L.J., Torborg, J.G.: Display Techniques for Octree-Encoded Objects. *IEEE Computer Graphics and Applications* 1(3) (1981) 29–38
2. Jordan, J.: NVIDIA Material Definition Language One Scene for Different Renderers. (2018)
3. Larsson, T., Akenine-Möller, T.: A dynamic bounding volume hierarchy for generalized collision detection. *Computers and Graphics (Pergamon)* 30(3) (2006) 450–459
4. Martz, P.: The OpenGL shading language. *Dr. Dobb's Journal* 29(8) (2004) 80
5. NVIDIA: NVIDIA OptiX. 1(March) (2019)
6. Priggemeyer, M., Losch, D., Rossmann, J.: Interactive calibration and visual programming of reconfigurable robotic workcells. In: *Proceedings of the 2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, Auckland, New Zealand, July 9-12, IEEE (2018) 1396–1401 ISBN 978-1-5386-1854-7/18 Electronic ISSN: 2159-6255.
7. Schluse, M., Priggemeyer, M., Atorf, L., Rossmann, J.: Experimentable digital twins — streamlining simulation-based systems engineering for industry 4.0. *IEEE IEEE Transactions on Industrial Informatics* PP(99) (2018) 1–9
8. Thibault, W.C., Naylor, B.F.: Set operations on polyhedra using binary space partitioning trees. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987*. Volume 21. (1987) 153–162
9. Thieling, J., Rosmann, J.: Highly-scalable and generalized sensor structures for efficient physically-based simulation of multi-modal sensor networks. In: *Proceedings of the International Conference on Sensing Technology, ICST*. Volume 2018-Decem. (2019) 202–207
10. VDI: Logistics and Production. In: *VDI-Richtline 3633*. Number December. (2010) 1–49
11. Zeigler, B.P., Muzy, A., Kofman, E.: *Theory of modeling and simulation : discrete event & iterative system computational foundations*. (2018)
12. Zellmann, S., Hellmann, M., Lang, U.: A Linear Time BVH Construction Algorithm for Sparse Volumes. (2019) 222–226

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.