Check for updates

JDMS

Technology

Journal of Defense Modeling and Simulation: Applications, Methodology, Technology
2019, Vol. 16(3) 255–271
© The Author(s) 2018
DOI: 10.1177/1548512918811954
journals.sagepub.com/home/dms
SAGE

Dynamic synthetic environments: a survey

Ruben Smelik, Freek van Wermeskerken, Robbert Krijnen, and Frido Kuijper

Abstract

The real environment in which military operations take place is dynamic and ever changing under the influence of natural effects and human activities. In contrast, synthetic representations of these environments used in simulations typically have had little support for dynamic effects and run-time changes. Advances in computer graphics research and innovations in game technology now allow for real-time dynamic simulation of specific features, such as physics-based building destruction or particle-based terrain deformation. However, widespread application of these algorithms and techniques in the context of military training and mission preparation is complicated by the additional requirements imposed by the distributed and heterogeneous nature of military simulation technology, such as the need for correlation and coping with specific limitations of (legacy) simulators. In this survey, we review the state of the art of methods and techniques for dynamic synthetic environments and discuss their applicability and current limitations in the context of distributed interactive simulation.

Keywords

Distributed interactive simulation, synthetic environment, dynamic terrain, deformation, destruction, correlation

1. Introduction

A synthetic environment (SE), that is, a virtual representation of actual geographic or fictional but lifelike terrain, provides the basis of any military simulation scenario. Since its inception, the application of military simulation technology has become increasingly widespread, initially focusing on the education and training domain, but now including Concept Development and Experimentation (CD&E), large international exercises, and mission and operational planning, preparation, and analysis. Because of these new application domains, the demands and requirements with respect to the scale and the level of fidelity and detail of SEs have grown as well, nowadays encompassing large geographical areas (provinces, countries) and being based on sub-meter resolution source data.

Despite these advances, SEs more often than not remain relatively static models of the real world. They represent a snapshot in time, and do not evolve with changes in the real world or during the course of the simulation scenario execution, for example, evolving weather conditions or a change in season, events, or operations, such as weapon effects or combat engineering. The lack of support for dynamic changes to the environment decreases the fidelity of the simulation and can diminish the value of training or preparation sessions.[1] For instance, a static, fair-weather SE cannot prepare a platoon commander for coping with deteriorating weather conditions that affect the trafficability of a chosen route.

Advances in processing power and GPU computing (i.e., algorithms running on Graphics Processing Units) now make it possible to execute complex deformation and fracturing algorithms, originating from computer graphics research and initially developed for application in the

Modelling, Simulation and Gaming Department, TNO, The Netherlands

Corresponding author:
Ruben Smelik, Modelling, Simulation and Gaming Department, TNO, Oude Waalsdorperweg 63, 2597 AK Den Haag, The Netherlands.
Email: ruben.smelik@tno.nl

movie industry, in real-time on desktop systems. As a result, high-end entertainment game developers are steadily incorporating dynamic effects, such as destructible buildings, in their games and game engines. At a somewhat slower pace, some of these effects are finding their way into modern military simulation systems as well.

Even if specific simulation systems include some capabilities to dynamically modify their instance of the SE, the correlation between the SE instances of heterogeneous simulators in a distributed execution is quickly lost as a result of dynamic effects: each simulator implements dynamic effects in a proprietary way or might not support it at all, resulting in differences in, for example, line of sight or cover, which can affect the fair fight principle or training value.

There is a clear need in the military modeling and simulation (M&S) community for more dynamic synthetic environments (denoted here as DSEs) in distributed interactive simulations (DSEs have actually been on the community's wish list since the early 1990s[2,3]), and the required research and technology to support this is slowly becoming available. However, the widespread deployment of DSEs is hindered by a lack of coherence and standardization: what types of algorithms are suitable for simulating dynamic effects, what information should the DSE data model contain to support dynamic modifications, how can changes to the DSE be executed in a correlated way across all federates, and what additional requirements are imposed by sensor simulations and Computer Generated Forces (CGFs)?

The goal of this survey is to provide a representative overview of the state of the art of the methods, algorithms, and techniques applicable to DSEs and discuss open issues, specifically to what extent current methods fulfil the additional requirements and constraints imposed by distributed military simulation, such as the level of fidelity, performance and scalability, and correlation across heterogeneous simulators. The focus of the survey is on two topics within the field of DSE research: terrain deformation and object destruction.

The article is structured as follows: Section 2 introduces the concept of the DSE, its features and requirements, and the scope of the SE chosen for this survey; Section 3 surveys methods related to deformations of the terrain skin as a result of, for example, explosions, weather effects, or engineering; Section 4 focuses on methods for fracturing or destroying materials and objects in real-time; Section 5 discusses to what extent the currently available work is able to fulfil the requirements for DSEs and what avenues for future research there are; and, finally, Section 6 summarizes the findings of the survey.

2. Dynamic synthetic environments

There are many definitions available for the concept of the SE. For the purpose of this survey, we follow the definition used by the NATO Modelling & Simulation Group, tasked with exploring *dynamic synthetic environments* for distributed simulation[4]:

> A Synthetic Environment (SE) is a collection of elements that represents the physical world within which (simulation) models of systems exist and interact (i.e., terrain, weather, oceans, space). It includes both data and models representing the elements of the environment, their effects on systems, and models of the impact of systems on environmental variables.

To get some sense of how a DSE differs from a traditional static SE, consider the following (non-exhaustive) list of examples of dynamic changes to a real-world environment that can happen during the course of a military mission or operation:

- a tracked armored vehicle traverses a sand plain, leaving distinctive tracks in the ground;
- due to heavy rainfall in recent weeks, a wadi is flooded and no longer traversable by ground vehicles;
- a combat engineering unit constructs a network of trenches and tank traps to fortify a position;
- a fire team breaches the wall of a target compound;
- retreating troops blow up a small levee dam, causing the land to become flooded and disrupting the advance of their pursuers;
- a large multi-story industrial facility is hit by an air strike, causing heavy damage to the interior and resulting in a partial collapse of several top floors;
- a building collapses in an uncontrolled way after the detonation of an illegal munition supply, causing heavy damage to nearby buildings and vehicles as a secondary, collateral effect.

As follows from these examples, a DSE differs from a traditional static SE because it can react and adapt to changing configurations (e.g., heavy rainfall) or events (e.g., wall breaching), whereas in a static SE, the environment is essentially frozen and, although it may display superficial effects (fire, smoke, damage decals), on a structural level it is unaffected by any events that occur during the course of the scenario.

The current lack of support for dynamic changes to the SE leads to discrepancies between the simulated outcome of a plan or activity and the outcome it would have in reality.[1] For instance, by neglecting the effect of heavy rainfall on the soil conditions and the trafficability of (especially unpaved) roads and river beds, the route of a convoy that is

planned by a commander might be determined to be fast and safe in simulation, but very slow or even infeasible in practice. If the fire team is not able to breach the wall of the compound in their simulator because of a lack of technical support for this, the team leader is forced to enter through the gate instead, thereby reducing the speed of the operation and possibly increasing risks of failure. More importantly, the training or preparation value of this simulated operation is strongly reduced, as the plan execution in the simulator no longer matches with the operational plan.

It should be noted that increasingly often modern simulators incorporate a part of the full feature set of a DSE, for instance destructible objects with fixed damage states, vehicle tracks (often as a visual effect only), or, in specific cases, support for some types of combat engineering. In general, these dynamic effects are implemented in a constrained, ad-hoc, and proprietary way, and these simulators do not communicate their changes to the SE with other participating systems. Often, such simulators are developed as standalone procedures or operator training tools, for instance, an excavator training simulator that is used as part of a combat engineering training curriculum, and not intended to be employed in a distributed simulation. To the best of our knowledge, there is currently no fully featured implementation of a DSE available.

The primary intended application of DSEs this survey focuses on is mission training or preparation in an interactive (i.e., real-time) and distributed simulation context, with a number of virtual and constructive entities participating in a shared scenario, controlled from a heterogeneous set of simulators coupled using a standard data exchange protocol, such as Distributed Interactive Simulation (DIS[5]) or High-Level Architecture (HLA)[6]. Distributed simulation scenarios are extremely diverse and can include any possible combination of unit types, aggregation and abstraction levels, plans and tasks, and environments and conditions, and, as such, could require any of the features of a DSE.

Different than for standalone simulators, because of the variety in systems involved in scenarios and events, distributed interactive simulations impose two main requirements with respect to DSEs.

Correlation: in a distributed simulation context, the measure of correlation of the SE across multiple heterogeneous systems can have a large effect on the training value and effectiveness of the simulation. This holds even more so for DSEs. For instance, human observers might need to be able to draw the same conclusions of the damage state of a building in different simulators, meaning that the visual appearance needs to match to the extent that is required. As another example, changes in trafficability of a road due to weather influence the need to match between a manned HUMVEE simulator and CGF vehicles both driving on that road. Without a high degree of initial correlation and mechanisms in place to preserve correlation throughout the simulation execution, with each update to the DSE, the states of the different instances of the DSE within the federation will diverge, reducing the fair fight and training value.

Integration: since distributed interactive simulations need a fully featured DSE, it is important that individual methods and techniques are properly integrated in a technical framework, that is, made to work well together. This means, among other things, that the datamodel of the DSE (its terrain representation(s), structure of objects and features) should fit the heterogeneous techniques that operate on it and be kept internally consistent, and that the framework should be able to cope with secondary or delayed effects of certain events and operations. For instance, a small patch of terrain might subsequently or perhaps even simultaneously be deformed as a cause of a heavy tracked vehicle driving through it, an exploding artillery shell, or erosion by an intense rain shower. Each of these deformation types might be implemented in a different way and work on different representations and data structures of the terrain patch. A DSE implementation should be able to cope with these different computational approaches to deformation and preserve the stability and consistency of that particular patch of terrain.

In addition, the wide variety of simulated platforms (space, air, ground, surface, subsurface) participating in the federation as virtual entities (i.e., human-in-the-loop) and the presence of computer-controlled constructive entities (civilians, opposing forces) give rise to two additional high-level requirements.

Sensors: sensors other than the human eye often play an important part in distributed simulations. When using a DSE, the sensor simulation algorithms of the participating virtual and constructive entities should be able to cope with dynamic effects and changes, for example, accurately simulating the temperature differences on debris and craters after an explosion.

CGF: as simulation scenarios often feature a number of CGFs, it is important that these entities are able to sense the dynamic changes in the SE and act accordingly. In particular, navigation and behavior planning should deal with dynamic changes to the environment, for example, by updating navmeshes, updating lines of sight, reassessing suitable cover points, etc.

Zooming in on individual techniques and effects, to assess the applicability of the methods reviewed in this survey to distributed interactive simulations, the following list of detailed requirements and constraints has been considered.

Performance: it should be feasible to execute the method in real-time, given an efficient implementation running on current or near-future hardware.

Scalability: the method should scale well to large datasets, object counts, or geographic extents, as distributed military simulation scenarios most often use large-scale and detailed environments. This constraint makes, for instance, deformation methods that can only operate on tightly constrained sandboxes (e.g., an excavator simulator) not applicable as-is.

Realism: the method should produce results that match the level of fidelity and level of detail desired by the intended application, which in most cases means a plausible result that matches the expectations of the training instructor, scenario developer, or exercise leader. In a distributed simulation context, the focus is not on the precise simulation of a single interaction event or munition effect, but on the performance of an operational plan or the effectiveness of communication and decision making of a (multi-national) task group. This means that often one does not need the most accurate and complex simulation of a dynamic phenomenon available, and abstractions or simplifications can sometimes be justified.

Control: dynamic effects can have far reaching consequences for the course of the scenario, which can introduce risks to an exercise or training session, for instance resulting in a trainee not being able to meet all the training goals because a route has unexpectedly become impassable. To mitigate the risk of derailing a scenario, the method should have a predictable outcome and be stable. In some specific application scenarios, a sacrifice of realism in order to achieve this predictability might be needed or the instructor or exercise controller needs additional tools to control the effects on the scenario.

Authoring: a method might impose additional requirements on the information contained in and the structure of content and data models, for instance a wall object might need pre-computed fracture lines for a particular destruction method. Furthermore, a scenario might need to be extended to include or trigger dynamic effects at certain locations. The required data and authoring activities for content and scenarios to support dynamic effects and interactions should (eventually) be seamlessly integrated into the SE generation pipeline, that is, the process to go from source data to simulator-specific terrain databases and supporting tools for each phase of the process (e.g., a three-dimensional (3D) modeling package, a database generation system (DBGS)).

To provide focus in the survey, we have limited the scope of DSEs to the base terrain model and its natural (soil material, vegetation, hydrography) and man-made features (infrastructure, buildings, other objects). We will not give special attention to the dynamic simulation of deep ocean environments and space. Furthermore, although weather and climate are important factors for the dynamism of the world and can have a major impact on simulation outcomes,[1] we do not treat works dealing purely with its simulation. However, the effects of weather, foremost precipitation, on the soil and terrain are included in this survey. Finally, special effects such as animations and particle systems required to visualize a dynamic effect such as an explosion in a plausible way in real-time are not treated extensively in this survey.

The following sections review in detail the body of work on the two main topics of this survey: terrain deformation and object destruction.

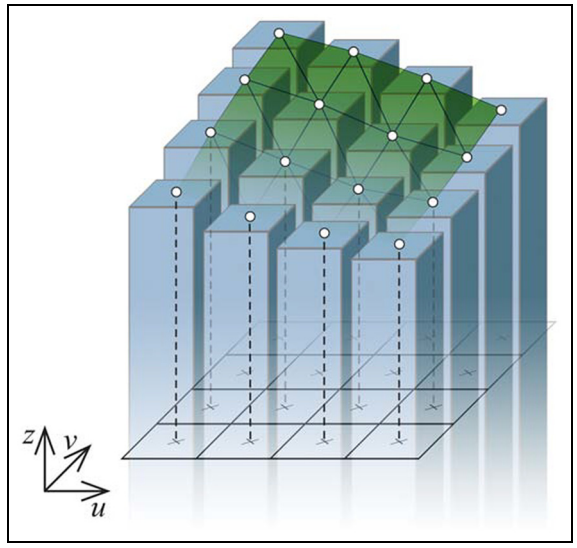

Figure 1. A diagram showing the representation of terrain as a height field from Holz et al.,[8] who implemented the height field in such a way that it can be dynamically updated. Courtesy of D Holz, ©EUROGRAPHICS Association 2009, reprinted with kind permission.

3. Terrain deformation

Terrain deformation in the context of interactive simulations means altering the terrain surface or skin at run-time, including first and foremost the manipulation of values of the surface height field (see Figure 1) or triangulated irregular network (TIN), and also the modification of surface soil materials resulting in changes in texture appearance or in simulation properties (e.g., surface friction). Although there are many possible causes for a terrain to deform, ranging from footsteps in mud to earthquakes, we focus our discussion on the types of deformation most relevant to a military simulation context: vehicle tracks, combat engineering, in particular digging,[7] craters, local erosion, and changes in soil condition and trafficability due to weather effects.

For each of the terrain deformation methods treated here, we consider everything from the concepts and approach of simulating dynamic terrain to the actual implementation. However, we do not discuss visualization algorithms and effects (such as dynamic textures, shaders).

In general, terrain deformation has a clear trade-off between performance and scalability on the one hand, and realism and level of detail on the other. However, because of the ever-increasing computing capabilities of modern hardware and the move to both GPU-based parallel processing and cloud computing, it is clear that the performance

statistics of less than recent publications might be based on outdated or obsolete implementation techniques and hardware. As a result, more detailed and complex methods published some time ago might already be feasible to run at interactive rates in modern implementations. Therefore, we try to give a more qualitative appreciation of the performance and scalability options of each method.

The criteria we use for comparing methods are its capabilities, the computational algorithms and terrain representations employed, and performance possibilities. It should also be clear that many publications focus on *one* specific aspect or capability of dynamic terrain, for which a small test area or sandbox is constructed. This is in contrast to what we are in search of: a unified approach and technique to enable dynamic terrain deformation that can be used for all relevant purposes.

3.1. Deformation capabilities

We discern the following capabilities that terrain deformation methods can support. In italics are capabilities we consider to be especially important for military simulation.

- *Vehicle tracks.*
- *Combat engineering: digging.*
  - Can an excavator create a trench or earth wall?
- *Trafficability affected by soil state.*
- *Explosions and crater forming.*
- *Types of soil materials.*
  - Does the method support several different types of soil materials (gravel, mud, dry earth, asphalt, rock), or is it specialized for one particular material, for example, fine sand?
  - Which parameters can be specified for each supported soil material, for example, the level of moisture?
- *Soil materials distribution over the terrain surface*, for example, surface maps.
- Layered soil materials, for example, sand over bedrock.
- Volume preservation.
- Erosion and material stability over time.
  - Is the angle of repose accounted for?
  - Is erosion by weather supported?

Since modern SEs involve a multitude of different soil materials, typically embedded in a surface mask texture or defined using standard encodings such as Surface Material Coding (SMC), the ability to simulate dynamic terrain for multiple materials is very relevant. The effect of soil material on trafficability and digging operations adds additional scenario possibilities in the military context (see the work of Birkel[9] as part of SEDRIS for an in-depth technical specification of terrain trafficability simulation). In addition to this, the capability to simulate the effects of munitions and explosions on the terrain surface would further enhance the realism and training value of the scenario.

We have categorized the real-time computational approaches that approximate the physics of terrain displacement.

- Solve a partial differential equation.
  - How accurate are the used approximations?
- Use static rules.
  - How many degrees of freedom are supported?
- Apply a parametric effect template.
  - What is dynamically changed by applying the template?
  - How large can the template library be?
- Combinations of the above approximations.

Often these methods are applied simultaneously with or just after the common object-to-object physics update loop. Input to the terrain physics update loop are the internal state of the terrain surface model (which can include internal forces) and external influences (entities, events). Some methods allow for optional performance optimizations, which can be very beneficial to implement to improve scalability. We have identified the following global approaches to performance optimization in the context of terrain deformation:

- locally refining the terrain representation (e.g., increasing resolution);
- simplifying the terrain representation; and
- only calculating physics in active regions and dynamically track and update the active regions.

Also, each method has a representation of the terrain surface. We distinguish between the following:

- particles;
- a height field (grid);
- a surface (some continuous representation); and
- combinations of the above representations and rules for switching between these.

3.2. Deformation methods

We review the terrain deformation methods in roughly chronological order.

The early work of Sumner et al.[10] focuses on footprints and tracks in soft soil and, given the limitations of contemporary hardware, proposed approximations of the physics involved in the deformation, focusing on plausible appearance and animation instead of accuracy. The computational approach is a set of rules involving the detection of the collision area and estimation of the impacted volume. This volume is displaced around the impacted area.

Erosion is applied as a post-processing effect to create a more plausible footprint or track. Soil materials supported are sand, mud, and snow, and the terrain is represented by a dynamically updated height field. Although a good starting point for research in terrain deformations, the method ultimately lacks realism and versatility.

While primarily proposed as a level-of-detail (LOD) approach for large terrain meshes, the ROAM algorithm by He[11] also included basic support for dynamic terrain and textures, although the physical deformation calculation is not discussed in this work. Advances in GPU computing have made this purely Central Processing Unit (CPU)-based scheme obsolete.

Pla-Castells et al.[12–14] present the application of cellular automata to simulate the behavior of granular materials in a number of publications. Firstly, they derive a number of physical functions for the mechanics of granular ground surfaces. These functions are subsequently discretized such that they can be calculated in the form of a cellular automaton (i.e., per cell per time step). The resulting method allows for dynamic changes regarding trafficability and erosion; however, it remains unclear what type of granular materials are supported and if these can be simulated in conjunction, getting mixed up over time. In the follow-up publications,[13,14] the method is tested in a simulation application and external forces on the terrain are incorporated in the method (soil–tool and soil–tire interactions).

The work of Aquilio et al.[15] focuses on deforming clay-like terrain, primarily with vehicle tracks. It was one of the first terrain deformation approaches using GPU-based techniques to efficiently compute the material displacement caused by a rigid body colliding with the terrain, by rendering the object to a displacement buffer that is applied to a height-field texture. As such, it involves a crude approximation of the physical deformation and will not be immediately applicable to different types of soil material. Furthermore, the authors note that they have not implemented erosion of the surface and that the method is not very scalable.

Explosions and craters are supported in the method of Cai et al.,[16] which adapts the above-mentioned ROAM algorithm. The technique involves deploying a parameterized crater template (e.g., size) to a dynamic height field. The downside of this method is that it is limited to crater forming and assumes a uniform soft soil. Brandstetter et al.[17] introduce a different CPU-based terrain refinement and deformation method, which supports deforming the terrain due to, for example, craters and tire tracks both within and outside the active area (out-of-core).

Another example of dynamic terrain effects from explosions is from Andersson,[18] which explains the dynamic terrain available in the Frostbite game engine.[19] This modern engine supports different soil materials and can create craters in the terrain skin using a dynamic height map. It is unclear how the terrain deformation is calculated and if the underlying soil material has any influence on the deformation.

In the paper by Wang et al.,[20] we find another method to dynamically alter terrain as a result of explosions. The method is template-based as well; it consists of pushing a sphere into the terrain and displacing the height map accordingly. Local terrain textures are then adjusted to create a visually appealing crater appearance. This method is limited to soft soil materials.

There are a number of recent approaches that primarily focus on improving deformation performance by devising efficient GPU-based processing schemes, for instance the GPU stamps[21] (i.e., templates), the GPU-friendly dynamic terrain mesh structure by Pangerl,[22] and the GPU framework for deforming terrain modeled as subdivision surfaces.[23,24] With the focus on optimization, the actual deformation algorithm and terrain datamodel are often relatively simple. However, the feasibility of real-time terrain deformation on current hardware is clearly shown. Furthermore, Pennings[25] explore the performance and consistency maintenance challenges of applying deformations in a networked multi-user setup.

Holz et al.[8] are the first to use a particle simulation combined with a dynamic height field to simulate terrain deformations. As long as there are no interactions with objects, the terrain is represented in its most efficient manner, that is, as a grid. When an interaction occurs, the local region affected is transformed into particles and a particle simulation is started to capture the effects of the interaction. When all forces involved have stabilized, the particles are translated back into the grid representation, updating the dynamic height field. The method is able to simulate different soil materials, since the particles can have different (physical) properties. Each soil material generates a (possibly mixed) set of particles, and upon compacting the particles, the grid topmost soil material can be updated as a result of the simulation. The method supports changes in trafficability, simulating erosion and performing digging operations. The performance is improved by tracking active areas; only in these areas is the costly particle simulation performed. In his more recent publication, Holz[26] introduces an improved particle simulation method, including more physical laws and effects and thereby improving the plausibility of the behavior of the terrain. A physical coupling with rigid bodies is embedded into the method, enabling two-way interaction between the terrain and objects.

Another publication in which a particle system and a height field are used in conjunction is from Zhu and Yang.[27] They propose a method with multiple layers stacked vertically, where each layer consists of different kinds of particles. On top is a layer of dynamic particles, followed by a layer of interface particles and at the bottom a layer of static particles (see Figure 2). The static layer can expand and decrease depending on the flow of the

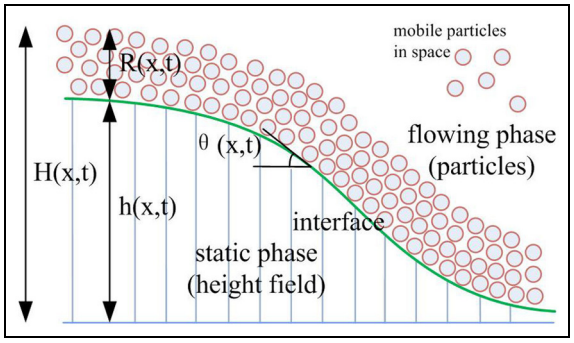

Figure 2. A diagram showing the representation of terrain as a particle system from Zhu and Yang.[27] Courtesy of B Zhu, ©EUROGRAPHICS Association 2010, reprinted with kind permission.

interface and dynamic layers. The method excels when simulating a scene where the flow of granular materials is key and the authors note that the method performs well for large-scale 3D scenes. However, in the context of distributed military simulation, these scenes can still be considered small. Furthermore, it is not clear how the interaction with objects should be modeled.

A completely new representation of the terrain surface and its dynamic properties is given in the thesis of Chen.[28] It is a method tailor-made for vehicles driving through terrain. The terrain is modeled by an elastic layer and a static layer. Between these two layers, a mass spring system captures the dynamics of the terrain. This system is capable of simulating sand, snow, sawdust, and other surfaces by adjusting the elasticity equation. Tire tracks are added by indenting the surface with a specific template for each tire. The limitations of this approach are not fully described, for example, will multiple subsequent vehicles have an increasingly deteriorating effect on the terrain?

In the paper of Gilardi et al.,[29] a deformation method is proposed that applies a discretized drift-diffusion equation to a height field. The height field can be updated using von Neumann neighborhoods and then applying the effect of the angle of repose. Therefore, the method scales linearly with the grid size and good performance is obtained. The method can simulate different surface types by adjusting the advection, diffusion coefficient, or angle of repose constants for a grid cell. However, there is no information on mixing surface types.

3.3. Preliminary conclusions

In this section, we summarize the work on dynamic terrain deformation and suggest avenues of future research.

In general, most publications deal with one particular use-case (see Figure 3), which also limits the broadness in capabilities of the presented method. Combining these single-purpose methods into a generic multi-purpose method that can be applied to large-scale environments has to date been neglected. A logical consequence of this is that proposed performance optimizations are often tailored for the chosen use-case.

The first typical use-case is a vehicle that drives through dynamic terrain, where the capabilities of trafficability, multiple (horizontal) surface types, and erosion are accounted for. The terrain representations employed vary considerably. The physical rules to update the terrain surface are often founded on (partial) differential equations interpreted to fit the chosen representation. The supported types of soil materials typically depend entirely on the type of physical rules implemented. Moreover, it is not clear which types of materials can be simulated and which would require extensions of the methods and further research, since only the implemented soil material parameters are mentioned. With respect to the requirements presented in Section 2, the most promising method for this use-case is from Gilardi et al.[29]

The second common use-case involves an excavator that performs digging in a patch of dynamic terrain (including erosion). The digging operation is modeled as a particle simulation for several types of granular materials, implementing a physically detailed method that can be calculated using particles. When such a method does not include simplifications or is not able to transform its particle representation to a dynamic height field, its scalability is severely reduced. The techniques to switch from particles to a height field and vice versa are still diverse. The most promising method for this use-case is introduced by Holz et al.[8,26] The work by Dukstein et al.[30] is very relevant for this use-case, as it implements excavator training in a military simulation context. Notable topics they cover are the effect of weather on the terrain state and communicating terrain deformations between simulators using experimental DIS protocol messages (including distributing changes in elevation and soil material at regular intervals using compressed grids[31]).

The third use-case is an explosion that creates a crater in the terrain. These types of deformation are often modeled by indenting the terrain surface with a sphere-like object (the size of which depends on the explosion force or munition type). In these publications, only the explosions are supported and no extensions to other capabilities are shown.

Other neglected dynamic changes to the terrain are changes to the soil materials due to weather effects, or mixing of different soil materials. In general, the methods presented do not consider large-scale environments and do not prioritize performance optimizations. Lastly, a currently unexplored area is how to correlate dynamic terrain changes and how to compute dynamic terrain deformation

Figure 3. Examples of methods from the three most common use-cases for terrain deformation: vehicle tracks generated by the method of Gilardi et al.,[29] an excavator simulation by Holz et al.,[8] and crater generation by Brandstetter et al.[17] Courtesy of M Gilardi, D Holz., and W Brandstetter III. ©EUROGRAPHICS Association 2009, 2016, reprinted with kind permission.

at multiple levels of detail for simulators that operate on different abstraction levels.

4. Object destruction

Object destruction in the context of interactive simulations means altering the state and structure of an object that is part of the SE at run-time as a result of blast damage or other physical forces that act upon this object. The alterations could include not only modifications to the visual appearance of the object (meshes, textures, shaders), but also changes in functionality (navigation, cover, interaction). Furthermore, sensors other than the human eye might perceive the object differently due to the alterations made (e.g., a thermal infrared (TIR) sensor might sense local changes in temperature on the object). Lastly, object destruction might entail secondary effects, for example, debris from one object could do significant damage to another object, potentially triggering a chain of object destruction.

In distributed interactive simulations, objects that are part of the SE have traditionally been static. Later on, in line with how damage to entities was typically modeled, important objects such as target compounds were enhanced with a discrete set of pre-computed damage states (e.g., undamaged, damaged, destroyed), where the pre-computed model might be hand-modeled by an artist or the output of a (procedural) generation algorithm. Fixed damage states suffer from clear drawbacks, as the pre-computed damage might have little relation to the actual damage that was inflicted (e.g., different types of weapons and munition, direction of the blast, etc.). Still, they are the prevalent way of modeling object destruction today.

Inspired by the advances in physics-based object destruction in entertainment gaming, some simulators have gradually started implementing more dynamic damage effects to specific objects, such as buildings. However, the technology developed for gaming is aimed at enhancing the player's immersion and experience, and still needs to be extended to better fit the specific requirements of distributed simulation (see Section 2).

As with terrain deformation, discussed in the previous section, for object destruction there is again a trade-off between performance and scalability on the one hand, and not only realism and level of detail, but also increased flexibility and interactivity, on the other hand.

Below, we review relevant methods to dynamically alter objects that are part of the SE, that is, partly or fully destroy objects of different breakable materials. Most of the discussed methods focus on one particular capability, physical process, or material. They differ, among other things, in the extent of pre-computation or modeling that is required. Consequently, authoring requirements, as discussed in Section 2, are highly relevant for object destruction. Object destruction has more industrial support for middleware and extensions of physics engines, and therefore the dominant technology in this field is addressed as well.

4.1. Destruction capabilities

We discern the following capabilities that object destruction methods can support. In italics are capabilities we consider to be especially important for military simulation.

- *Fully destructible buildings*.
- *Destruction of solid objects*, for instance infrastructure: bridges, communication towers, lighting poles.
- *Breaching doors and windows*.
- *Breaching a wall*, that is, blowing a hole in a wall.
- *Secondary effects*, that is, damage from debris.
- Tree felling.
- Collapsible buildings (stress-based collapse).
- Cracking and fracturing of objects (stone, rock, glass, ice, metal, cloth).
- Chippable surfaces (micro destruction).
- Micro effects (particle debris).
- Primary destruction from fire propagation (forest fires).

- Realistic animations, visual, sound, and lighting effects related to damage and destruction.

In this field of research, the terms *cracking* and *fracturing* are commonly used,[32] where *cracking* relates to the formation of lines in a surface without separating it (e.g., cracks in a mud surface), and *fracturing* is the separation of an object or material into two or more pieces under the action of stress. We have limited our focus to the cracking and fracturing of relatively large solid objects, because of the relevance for military simulation, but there exists much more work in the field of object destruction. For an in-depth survey of techniques for simulating numerous phenomena that can affect the state of an object, such as aging, weathering, melting, and corroding, we refer to the extensive research carried out by Frerichs et al.[33]

4.2. Destruction methods

From an entertainment gaming perspective, when developing methods for simulating destruction effects, the real-time performance and graphical quality are of the highest importance.[34–36] Whether the approach actually produces physically accurate results is not as much of an issue as long as the results are perceived as plausible. Other important criteria are the ease of use for the designers and the degree of control it provides them to create a visually appealing effect. As a result, computationally demanding but more accurate methods are being adopted in the industry at a much slower pace.

As discussed above, the traditional method for implementing destructible buildings in computer games is to hand-model one or more variations of a (partly) destructed building. During run-time, when the fracturing occurs, for example, due to an explosion, the model is replaced by a destroyed counterpart. The switching of these models is obscured using special effects (explosion, smoke, dust, debris particles). This approach is scalable, computationally cheap, and predictable. The downside of this approach is, however, that the pattern of destruction does not correspond to the location of impact and the number of damage states is fixed.[37] This pre-fracturing process requires careful preparation of all the destructible models and a level of foresight on how and where a model will be destroyed. Particularly with large objects that are partially destroyed, it is impossible to cover all cases. Secondary effects of debris on adjacent objects, entities, or trafficability are also difficult to implement.

A final important downside of the pre-scripted approach is the increase in content authoring time. Content creation already typically is a major bottleneck in both the game development and SE generation process. By introducing pre-modeled destruction states to every object in the SE, considering that most pipelines for content involve much manual modeling effort, content creation can become very costly indeed. Tools like Maya[38] and Houdini[39,40] support a semi-automated workflow, where fractures can be computed automatically based on user parameters.

Earlier improvements on the model-switching approach were first explored by Mazarak et al.[41] In their method, objects are represented using a connected voxel datastructure, out of which solid pieces of debris can be extracted resulting from explosions, instead of using simple polygons animated using a particle system with no physical relation to the mass and geometry of the original objects. Particles are effective at modeling the special effects part of an explosion, such as fire and smoke. They are not suited for modeling the actual debris of explosions, because unlike particles, which move only in 3D space and can be defined by a vector, debris occupies space and have properties like a center of mass and six degrees of freedom.

The voxel approach by Mazarak et al.[41] uses a blast wave model to calculate the pressure field on an object. Micro-fractures or flaws are present in most materials, and the pressure of the blast wave causes these micro-fractures to grow, eventually fragmenting the object into many parts of debris. Voxels, just as pixels, suffer from ‘‘staircased edges.’’ This aliasing effect can be reduced by decreasing the voxel size, that is, increasing the resolution. Due to computational and memory limitations, a trade-off has to be found between voxel size and scalability.

Fracturing an object is performed by breaking links between voxels (crack formation), whenever the pressure caused by the wave exceeds the maximum pressure the link can withstand, up until the point that segments become separated from the rest of the object (they become debris). Rigid Body Simulation (RBS) is then used to animate the dynamic behavior of the debris. The authors note that the bottleneck of their method is in the collision detection; in the case of sustained fragmentation, the number of debris objects and boundary voxels continues to grow, resulting in increasingly many body-to-body collision checks.

With the advances in GPU processing capabilities, new methods allow for dynamic fracturing of solid objects into a large number of debris pieces at real-time. These methods allow one to apply user-defined fracture patterns at one or more impact locations on the object.[37] These approaches reduce the authoring effort, as only a small library of fracture patterns needs to be modeled, which can then be applied to all objects in the SE. However, it requires dynamic cutting of the object into sub-meshes (fragments) by performing Boolean operations on the objects using the desired fracture pattern. Müller et al.[37] and van Gestel and Bidarra[42] proposed a fast method to cut the model in real-time (see Figure 4).

To enable real-time destruction for complex shapes like buildings, the method uses two separate meshes. A visual mesh for rendering and a compound of convex-shaped

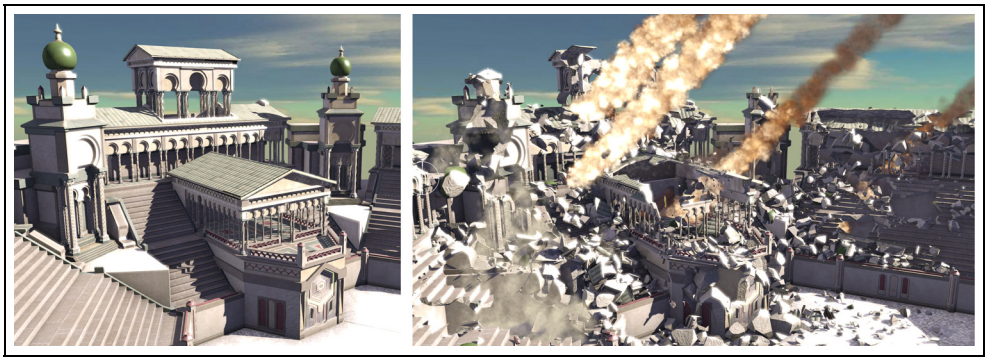

Figure 4. Destruction of a Roman arena using real-time fracture masking techniques, as presented by Müller et al.[37]. Courtesy of M Müller, ©ACM, reprinted with permission.

meshes for the fracturing calculations. Using Volumetric Approximate Convex Decomposition, the model decomposes into a set of convex shapes where each convex shape contains a unique part of the visual geometry. Using these pre-computed decompositions, only the fracture patterns have to be applied to the convex shapes that are affected by the operation. After fracturing of a convex shape the new convex shapes are fit to the visual sub-mesh.

The sub-meshes of the model are then simulated using physics simulation (RBS) to simulate the effects of the destruction. To simulate different types of impacts (size, force), different fracture patterns can be used. To prevent a large number of dynamically generated meshes (debris), thereby impacting performance, care has to be taken to limit the cutting of models into sub-meshes, limit the minimal size of meshes, and remove or simplify meshes/debris after a certain time. Although the technique makes it easy to cut away geometry from an object, for low triangle count objects, texture decals are generally used to enhance the visuals. Such decal textures are placed automatically at the fracture lines of the broken object.

Many current solutions are based on RBS to simulate non-deformable objects like debris. The Finite Element model approach described by Parker and O'Brien[43,44] introduces a solution that is also applicable to soft bodies and deformable bodies in real-time. Finite Element Analysis (FEA) is best known for its application in performing engineering analysis. Typical problem areas include elasticity, structural analysis, and heat transfer. The method includes the use of meshes for dividing a physical object into small elements. This mesh contains the material and structural properties, which define how the object will react to certain physical conditions. It allows the developer to assign different physical properties not limited to structural simulation to calculate the stress, deformation, and fracturing.

The FEA system is based on a tetrahedral finite element method and combines a number of existing simulation techniques into a coherent and performant system. For each object, a tetrahedral mesh is constructed that encompasses the visual model, and each vertex in the visual model mesh is linked to the enclosing tetrahedron. When any of the nodes in the tetrahedral mesh moves, the linked vertex positions are updated. The nodes of the tetrahedral mesh are described using a mass, damping, and stiffness matrix.

Based on the external forces acting on the mesh nodes, the amount of stress is calculated. Once stress reaches its fracture threshold, the connection is broken and the released energy causes a cascade of fractures, resulting in the separation of the tetrahedral mesh into separate nodes.

To limit the performance impact, in the proposed FEA system, tetrahedral nodes are not split into sub-nodes. This limitation can, however, result in visually unconvincing fractures, especially if the tetrahedral mesh is relatively coarse. Depending on the material and the anticipated stress levels of a particular area within the object, a finer or more coarse tetrahedral mesh can be created. To further improve the perceived realism of the fracturing method, the concept of *splinters* is introduced. These splinters are rendered as part of the surfaces, but are not part of the Finite Element simulation. Each splinter is associated with a single node of the tetrahedron mesh based on the splinter mesh centroid. During fracture, separate splinter meshes are created when splinter vertices are no longer connected to the corresponding tetrahedron nodes. Particularly for brittle materials like wood and stone, these splinters increase the visual realism of the fracture.

Although the solution described by Parker and O'Brien[43] refers to the real-time deformation and fracturing of solid objects, the approach is also capable of simulating soft body and deformable bodies and more physical properties than mass and structural stiffness.

Effectively, the FEA approach is much simpler. There is no need to pre-fracture objects and it does not require a soft-body solution to feed a RBS for special cases like

plastics and metal deformations. However, it still comes with a price.

4.3. Destruction engines

With the advancements of computing power and techniques, procedural destruction has found its way into middleware for game development and game engines. These engines provide an off-the-shelf solution for integrating destruction into the game.

The basic support of these destruction engines consists of the following stages.

1. Asset preparation. Preparing the geometry of the object for breaking and shattering, for example, Voronoi construction, tetrahedralization, and pre-fracturing.
2. Applying constraints. In this stage, the artist uses parameters and constraints, such as connectivity trees or specifying the strength of a wall, to influence the destruction simulation and obtain a desired, predictable result.
3. Running the simulation. As part of the simulation, the engine actually fractures the object into separate meshes and solves the object dynamics and collisions using physics libraries like Bullet,[45] the Open Dynamics Engine,[46] PhysX,[47] and Havok Physics.[48]

The following sections give a brief overview of the prominent destruction engines currently available.

4.3.1. Havok Destruction. Havok is an established middleware company, originally focusing on its physics library (Havok Physics) featuring rigid body dynamics, which was licensed and integrated in several games and game engines. The middleware suite they provide now includes AI (path finding and dynamic nav mesh generation), clothing simulation, and, finally, destruction. Havok Destruction 2012[49,50] has several tools to support the required content authoring pipeline. It uses *fracture templates*, which are procedural descriptions of how a fracture will work on a given material. The tools come with several predefined templates but allow the artist to create their own unique fracture template. In addition to the templates it provides a set of predefined *debris templates* to generate debris pieces. To make low poly geometry more interesting, the artist is able to provide decoration pieces and decal maps to enhance destruction masking. The decal textures and decoration pieces are applied to fracture faces at run-time. For example, it can add protruding bricks to pieces of a fractured wall.

For plausible destruction effects, the engine needs information about the connectivity of the pre-fractured pieces and how strong the connections are. Havok Destruction therefore stores connectivity information within the breakable shape. The connectivity ties the physics representations of the pieces together. With this information, the collapse of objects is made possible. For asset preparation, plug-ins for Maya[38] and 3ds Max[51] are available to artists.

4.3.2. NVIDIA APEX. Like Havok, NVIDIA is one of the established suppliers of physics engine middleware (PhysX[47]) for games. Engines like Unreal and Unity3D use NVIDIA's PhysX as their main physics simulator. APEX[52] is a software framework built on top of PhysX, providing authoring tools, custom pipelines, and optimizations with which developers can create physics simulations such as clothing, particles, force fields, and object destruction without the need to build these systems from scratch on top of a low-level physics engine.[53] APEX also comes with plug-ins for common 3D modeling packages such as Maya[38] and 3ds Max[51].

APEX Destruction uses hand-modeled pre-fractured meshes, additional sub-systems to handle aspects such as damage propagation, and a PhysX-driven rigid body solver to calculate interactions between chunks and fragments.[54] To support more complex destruction behavior, APEX includes specific functionality, such as specifying a hierarchical structure of the fracture chunks and defining connectivity constraints between chunks. A standalone editor, named the PhysX Lab Tool, provides a rich set of features to prepare destructible objects and preview and debug their desired destruction behavior.[35]

Starting in 2017, PhysX Destruction has been deprecated to be replaced by NVIDIA Blast, which is, according to NVIDIA, completely redesigned, and features improved performance and scalability.[55]

4.3.3. CryENGINE. CryENGINE[56] is Cryteks gaming engine for the Crysis game series. In addition, the engine has been licensed and extended by other developers, for instance, Amazon.[57] The engine includes an in-house developed physics and destruction module.

CryENGINE features several different types of representations for modeling object destruction, each suited and optimized for a specific type of fracturing or material.[35] The representations that can be used in the engine include (see the engines manual[58]) switchable damage states, joint constraints (i.e., connectivity between parts), pre-computed fracture pieces, deformable objects (based on cloth algorithms), and even real-time Boolean operations on geometry. Further, a complex (cinematic) destruction sequence can be generated offline in a modeling package and executed as an animation to guarantee deterministic and

performant results. Plug-ins for the 3D modeling packages Maya[38] and 3ds Max[51] are available for setting up the destructible objects.

4.3.4. Frostbite. The Frostbite Engine is a game engine used in the majority of the popular Battlefield game series by EA DICE.[19] To date, Frostbite has been exclusively used in games published by Electronic Arts. Destruction capabilities have always been an integral part of the engine and, as such, it is one of the most advanced destruction engines to date. The engine supports destruction both on a very small and a very large scale: small-arms fire can chip off pieces of walls and blocks, exposing cover positions, while tall skyscrapers can collapse due to stress caused by, for example, earthquakes. Furthermore, large infrastructure, such as a dam, can be destroyed, causing the entire map to be flooded by water. Because it is an internal proprietary engine, unfortunately there is a limited amount of up to date information publicly available regarding the technology behind the destruction engine.[59]

4.3.5. DMM engine. The latest middleware supplier in the field of destruction engines is Pixelux,[60] which introduced advanced destruction technology called Digital Molecular Matter (DMM) in 2008. DMM is based on the Finite Element model approach described by Parker and O'Brien.[43] The DMM solution uses tetrahedra to represent a 3D model (for more on the algorithms of DMM in production, refer to fxguide's Art of Destruction[44]). Unlike traditional destruction methods and the other engines discussed in this section, which rely on pre-fracturing and extensive configuration of the destructible material offline, DMM is based on real-world material properties and generates fractures at run-time.

While DMM has been used in many high profile movies for its visual effects, it is also available as a plug-in for 3D modeling packages and integrated into real-time game engines. The engine was used for major entertainment game titles like Star Wars: The Force Unleashed. The middleware physics engine is available for licensing.[60]

4.4. Preliminary conclusions

There is a long history of research on the subject of creating destructible assets for entertainment games, and there are quite a number of technical solutions (tools and engines) commercially available. The main focus of solutions in this domain has been performance efficiency, predictable results, and visual appeal. Physical correctness and secondary effects are of less importance. Following the typical game development workflow, in most cases preparing content for real-time destruction involves considerable manual modeling effort by artists, approximating the physical phenomenon and, at the same time, creating a visually appealing effect. As a result, with the increasing scale and level of interactivity of game worlds, creating (destructible) content consumes a substantial amount of the development budget.

Even for big budget games the ever-increasing effort and costs of content creation are becoming a development bottleneck. There is an urgent need for advanced content creation tools to support artists, employing procedural and/or physics-based algorithms. New solutions like the Finite Element approach implemented by Pixelux[60] in the DMM engine aim to ease and speed up the authoring process and seek to achieve the ultimate goal: a fully destructible game world.

With the amount of research available and the quite extensive tooling and experience available in the entertainment gaming and movie industry, the question becomes to what extent the currently available technology can be applied in the military M&S community for more dynamic SEs in a distributed interactive simulation context. There is no easy answer here.

- With sufficient resources (artists, time, budget), current tools and engines could be applied to make every asset in a SE destructible. However, budgets for the development of SEs are not comparable to the budgets of AAA-game development, especially with respect to content development, and there are often stricter time constraints (e.g., a SE for mission preparation), and therefore parametric or procedural content generation is often the only feasible path. At the same time, we see that even in the game industry, more automated art production tools are slowly being adopted. Therefore, we can expect that the modeling pipeline for destructible content will be made much more efficient in the coming years.
- Performance issues and considerations will continue to severely limit the scale of destruction one can introduce in a distributed simulated scenario. In the foreseeable future, we cannot expect to compute and visualize a physics-based high-ordinance munition detonation over a city in real-time, with multiple federates using heterogeneous simulators connected.
- In particular distributed simulation scenarios, where the visual appearance of the destruction effects needs to match almost perfectly, the required amount of correlation between multiple heterogeneous systems will be nearly impossible to achieve with the current technology, pipelines, standards, and protocols. A cloud-based destruction approach where the server orchestrates effects for the different heterogeneous systems could be one step toward a DSE in distributed simulation.

5. Discussion

In the previous sections, the state of the art in terrain deformation and object destruction was presented and some preliminary observations were made. This section discusses the main concerns and open issues, returning to the requirements specified in Section 2.

As stated in Section 2, the first and foremost requirement for the successful introduction of DSEs in distributed simulations is the preservation of *correlation* of the SE amongst all participating federates during scenario execution. Unfortunately, there seems to be surprisingly little work done on the implications of dynamic changes to the correlation of the SE within the federation. Clearly, these dynamic changes make it very challenging to maintain a correlated SE at run-time.

Even static SEs working from the same geo-dataset and content library is often not enough to guarantee true correlation, because specific limitations and conventions of one participating simulator might force the DBGS to make changes to the baseline data upon exporting to the simulator's proprietary run-time format. However, with the use of offline correlation validation tools and, in some cases, by downscaling the baseline data to suit the least capable simulator, it is typically possible to achieve an acceptable level of correlation between the different simulator-specific versions of the SE across the federation. All of this correlation is lost once the first dynamic change to the SE occurs and each simulator handles this change in a different manner, using proprietary algorithms, or lacks the capability to update its SE at all.

Maintaining correlation for DSEs is clearly an area that requires further research. There seem to be two main solution directions that could be explored.

1. Standardizing and agreeing upon algorithms to be used and expected results in order to increase correspondence in the way in which individual simulators handle dynamic events. The advantage of this approach is that simulators stay relatively unchanged and can still deal with the SE in their proprietary and optimized format and run-time implementation. Communication of changes would typically be done using extensions on the distribution protocols DIS (as already proposed by Lisle[61]) or HLA. Simulator validation and certification might be required in exercises where maintaining correlation and fair fight is key.
2. Moving to an authoritative service to which all participating federates subscribe and that has the final say of the (dynamic) state of the SE. This approach has the potential to ensure correlation and fair fight even under extensive dynamic changes to the environment during the scenario. Furthermore, by implementing this service on a modern, distributed, and scalable cloud architecture, the performance and memory constraints imposed by complex, physics-based destruction and particle-based deformation can be met. Finally, an individual simulator needs not implement all possible dynamic operations that can be executed on the SE (e.g., trench digging) in order to participate in a DSE scenario, as long as it is able to visualize all changes communicated by the SE service. However, this approach requires a fundamental shift in simulator architectures and implementations, which is problematic in practice considering the vast amount of legacy simulators in use, and is not without its challenges on the client side, for example, dealing with latency and network synchronization, area of interest management, and the need for fast online SE (re-)generation algorithms (see e.g., Ellis et al.[62] for work in this direction).

The second direction has the greatest potential for achieving truly correlated DSEs in distributed simulations; however, the move to a service-based DSE will require much more research and development and data model and protocol standardization efforts, and might be slowed by the vast number of legacy simulators and proprietary, ad-hoc implementations existing today. With the current focus and momentum of the *M&S as a service* paradigm (see, e.g., Hannay and van den Berg [63]), it might be the right time to shift toward *dynamic terrain as a service*.

Going back to the surveyed methods for terrain deformation and object destruction, in line with the steady increase in graphics and computing power, we see a slow transition from *appearance-based* techniques, for example, deformations based on fixed terrain offset templates and destruction using fixed damage states, to *physics-based* approaches, for example, particle-based deformation and FEA-based fracturing of objects and materials. An obvious benefit of this transition is that newer methods and techniques offer improved *realism* and, with the move from pre-computation to real-time execution, enhanced flexibility and, potentially, a decrease in *authoring* effort as damage models and effects are generated at run-time instead of modeled beforehand.

However, the transition has additional consequences for the adoption of DSEs in the M&S community. The *integration* of individual techniques and representations into a consistent dynamic world model, which to date is still one of the most important missing pieces of the puzzle, would be very much helped if this world model can be constructed from (an approximation of) real-world laws of physics and material properties, with as few tricks, limitations, and special cases as possible. In addition, *sensor* simulations can work on dynamic, physical changes to

Table 1. Coarse appreciation of the extent to which the requirements specified in Section 2 have been fulfilled by the surveyed methods [–, +/–, +, ++], with noteworthy research from terrain deformation (Section 3) and object destruction (Section 4) included as citations.

	Terrain deformation		Object destruction	
Requirement	Score [–.. + +]	References	Score [–.. + +]	References
Performance	++	21	++	37
Scalability	+/–	8	+/–	64
Realism	++	26,29	+ +	43,60
Control	–	None	+/–	52
Authoring	–	None	+	52,60

objects and materials, resulting in more accurate sensor imagery. Finally, CGFs can more easily be made aware of changes to the environment, as the semantics of these changes are defined more precisely than in the case of appearance-based changes, and, to cope with these changes, can employ dynamic agent planning and navigation mesh generation techniques already extensively explored in the gaming industry (albeit for smaller scale SEs).

As a consequence of this transition to real-time, physics-based deformation and destruction, the physics engine will become a core component of the distributed interactive simulation architecture and will need to assume new responsibilities going beyond the traditional RBS; we have already discussed engine extensions to support object destruction (Section 4.3), but in time also terrain deformation might be integrated as a standard physics engine capability. As a result, physics computations and the required physics representation of the terrain and all objects can quickly become the performance bottleneck of future simulations. Cloud-based physics computation and distribution is already being explored for state of the art entertainment games and large online virtual worlds and frameworks and middleware is becoming available[64,65]; this might also be a relevant direction for DSEs.

Zooming in on the individual methods and techniques and reciting the requirements specified in Section 2, we can conclude that with the current state of the art, one can process terrain deformations and object destruction in real-time at a sufficient level of physical realism (sometimes, even beyond the level one typically needs in distributed military simulation scenarios and exercises). Furthermore, industrial implementations and content pipelines exist, although they were developed with a clear focus on the workflow in use in the entertainment game industry (focus on manual modeling) and need some adaption to work in the typical geographic information system (GIS)-based DBGS workflows used for SE development. However, the scalability of methods to large-scale environments is often not addressed in the research we surveyed; some notable exceptions exist, for example, hybrid data representations to increase scalability. Lastly, by moving from fixed, pre-computed results to dynamic computation, some amount of control over the outcome of a dynamic event is inherently lost, which could introduce the risk of diverging from or blocking the preferred course of the simulation scenario. Table 1 gives an appreciation of the extent to which the surveyed methods fulfil the requirements and provides notable examples. One can conclude from this table that most research effort has focused on devising efficient algorithms that can realistically deform or destroy content in real-time; however, it is not yet clear how to these methods can scale to the large environments of military simulation scenarios, how to prepare the content for dynamic modifications (this is especially true for terrain, although there are many terrain authoring tools out there, they only allow one to define and texture a static terrain skin), and how to give, for example, training instructors or exercise leaders some control over the outcome of dynamic changes to the SE.

Summarizing, although the move to real-time, physics-based dynamic effects in both the academic and entertainment gaming community is promising, what is clearly lacking and hindering wide acceptance and application in the M&S community is a unifying *approach* and agreed-upon SE run-time *data model*, addressing the particular concerns for distributed interactive simulations and, most importantly perhaps, correlation, and also scalability to large-scale environments and long running scenarios, predictability and controllability, and ease and efficiency of authoring. Solution directions, such as the standardized SE development process RIEDP,[66] streaming terrain data serving,[67] moving to service-based SEs,[68,69] and the mentioned cloud-based physics engines,[70] should be explored to determine their suitability for ensuring the scalability and consistency of the DSE, and maintaining correlation between all participating heterogeneous simulators during execution of the scenario.

6. Conclusions

In the M&S community, for over 20 years there has been a strong desire to move from the traditional static and inflexible terrain databases to interactive and dynamic environments that, just as their real-world counterparts, continuously change as a result of natural processes and human interventions. Advances in computer graphics research and technology and the increased availability of computing power and parallel processing capabilities now make it possible to simulate many of the individual features that together make up a DSE in real-time on desktop computers.

In this survey, we reviewed the state of the art of methods and techniques for DSEs, focusing on terrain deformation and object destruction, and discussed to what extent the research and industrial solutions available today are applicable in the general context of distributed interactive simulation. For both terrain deformation and object destruction, we found methods and techniques with a suitable balance between physical accuracy and performance. What is lacking is a unifying approach that encompasses these individual techniques, while at the same time addressing the particular concerns for our application of distributed military simulation: scalability to large-scale environments, predictability, efficiency of authoring, and, finally, maintaining correlation across the federation.

Further research in this direction, focusing on developing such a unifying approach, could help our community to realize the dream of truly interactive and lifelike SEs.

Acknowledgments

We thank the members of the NATO Task Group Dynamic Synthetic Environments for Distributed Simulation (MSG-156) for the constructive discussions on this topic, and the anonymous reviewers for their insightful comments and suggestions that helped to improve this article.

Declaration of conflicting interests

The authors declare that there is no conflict of interest.

Funding

This work was supported by the the Dutch Ministry of Defence through the research program ‘‘Mission Training through Distributed Simulation (MTDS)’’ (V1701).

References

1. Pfeiffer KD and Tamash T. Measuring the impact of natural environment representation on combat simulation outcomes. In: *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC),* Orlando, FL, 1–4 December 2014.
2. Moshell JM, Blau B, Li X, et al. Dynamic terrain. *Simulation* 1994; 62: 29–40.
3. Rybacki RM. *Reliable dynamic terrain updates and fault tolerant terrain servers for distributed interactive simulations*. Master’s Thesis, University of Texas at San Antonio, 1995.
4. Exploratory Team MSG-ET-045. Dynamic synthetic natural environments for distributed simulation. *Final report, NATO Science & Technology Organization*, 2017.
5. IEEE 1278. Standard for distributed interactive simulation, 2012.
6. IEEE 1516. Standard for modeling and simulation (M&S) high level architecture (HLA).
7. Gilbert S, Keren N, Winer E, et al. Evaluating the value of dynamic terrain simulation on training quality. In: *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, Orlando, FL, 28 November–2 December 2016.
8. Holz D, Beer T and Kuhlen T. Soil deformation models for real-time simulation: a hybrid approach. In: Prautzsch H, Schmitt A, Bender J, et al. (eds) *Workshop in Virtual Reality Interactions and Physical Simulation ‘‘VRIPHYS’’*, Karlsruhe, 5–6 November 2009, pp. 21–30. Geneva: The Eurographics Association.
9. Birkel PA. Terrain trafficability in modeling and simulation. *SEDRIS Technical Paper* 2003; 1.
10. Sumner RW, O Brien JF and Hodgins JK. Animating sand, mud, and snow. *Comput Graph Forum* 1999; 18: 17–26.
11. He Y. *Real-time visualization of dynamic terrain for ground vehicle simulation*. PhD Thesis, University of Iowa, 2000.
12. Pla-Castells M, Garca-Fernández I and Martínez RJ. Interactive terrain simulation and force distribution models in sand piles. In: *Cellular Automata, 7th International Conference on Cellular Automata, for Research and Industry (ACRI 2006)*, Perpignan, France, 20–23 September 2006, pp.392–401. Berlin: Springer.
13. Pla-Castells M, Garcia-Fernandez I and Martinez-Dura RJ. Physically-based interactive sand simulation. In: Mania K and Reinhard E (eds) *Eurographics 2008 - Short Papers*, Crete, 14–18 April 2008. Geneva: The Eurographics Association.
14. Pla-Castells M, Garca-Fernández I, Gamón-Giménez MÁ, et al. Interactive earthmoving simulation in real-time. In: *XIX Spanish Computer Graphics Conference (CEIG 2009)*, San Sebastián, Spain, 9–11 September 2009, pp.235–238. Geneva: The Eurographics Association.
15. Aquilio AS, Brooks JC, Zhu Y, et al. Real-time GPU-based simulation of dynamic terrain. In: *International Symposium on Visual Computing*, pp.891–900. Berlin: Springer.
16. Cai X, Li F, Sun H, et al. Research of dynamic terrain in complex battlefield environments. In: *International Conference on Technologies for E-learning and Digital Entertainment*, pp.903–912. Berlin: Springer.
17. Brandstetter III WE. *Multi-resolution deformation in out-of-core terrain rendering*. Master’s Thesis, University of Nevada, Reno, December 2007.

18. Andersson J. Terrain rendering in frostbite using procedural shader splatting. In: *ACM SIGGRAPH 2007 Courses*. New York: ACM, pp.38–58.
19. EA DICE. Frostbite, ea.com/frostbite (accessed 16 October 2018).
20. Wang D, Zhu Qs and Xia Y. Real-time multiresolution rendering for dynamic terrain. *JSW* 2014; 9: 889–894.
21. Crause J, Flower A and Marais P. A system for real-time deformable terrain. In: *Proceedings of the South African Institute of Computer Scientists and Information Technologists Conference on Knowledge, Innovation and Leadership in a Diverse, Multidisciplinary Environment*, Cape Town, 3–5 October 2011, pp.77–86. New York: ACM.
22. Pangerl D. Dynamic GPU terrain. In: Engel W. (ed.) *GPU Pro 6*. CRC Press, 2016. pp.3–17.
23. Schäfer H, Keinert B, Nießner M, et al. Real-time deformation of subdivision surfaces from object collisions. In: *Proceedings of the 6th High-Performance Graphics Conference. EG, Lyon, 23–25 June 2014, pp. 89-96. Geneva: Eurographics Association.*
24. Schäfer H, Nießner M and Stamminger M. Real-time deformation of subdivision surfaces on object collisions. In: Engel W (ed.) *GPU Pro 6*. Boca Raton, FL: CRC Press, 2016. pp.27–50.
25. Pennings S. *Dynamic layering system for real-time interaction between entities and terrain*. Master's Thesis, Utrecht University, 2014.
26. Holz D. Parallel particles (P2): a parallel position based approach for fast and stable simulation of granular materials. In: *VRIPHYS 14: 11th Workshop on Virtual Reality Interactions and Physical Simulations*, 24–25 September 2014, pp.135–144. Geneva: Eurographics Association.
27. Zhu B and Yang X. Animating sand as a surface flow. In: *Eurographics (short papers)*, 3–7 May 2010, pp.9–12. Geneva: Eurographics Association.
28. Chen X. *Real-time physics based simulation for 3D computer graphics*. PhD Thesis, Georgia State University, 2013.
29. Gilardi M, Watten PL and Newbury P. Drift-diffusion based real-time dynamic terrain deformation. In: Bashford-Rogers T and Santos LP (eds) *EG 2016 - Short Papers*, Portugal, 9–13 May 2016. Geneva: The Eurographics Association.
30. Dukstein G, Watkins J, Le K, et al. Extending construction simulators through commonality & innovative research. In: *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2013*, Orlando, FL, 2–5 December 2013. National Training and Simulation Association (NTSA).
31. Tamash T. *SISO PCR 236.A: Dynamic Terrain*. Dignitas Technologies, LLC, 2016.
32. Iben HN and OBrien JF. Generating surface crack patterns. *Graph Model* 2009; 71: 198–208.
33. Frerichs D, Vidler A and Gatzidis C. A survey on object deformation and decomposition in computer graphics. *Comput Graph* 2015; 52: 18–32.
34. Muguercia L, Bosch C and Patow G. Fracture modeling in computer graphics. *Comput Graph* 2014; 45: 86–100.
35. Hettich R. *Approaches to destruction effects in real-time computer graphics*. Master's Thesis, Karlsruhe University of Applied Sciences, 2013.
36. Desbenoit B, Galin E and Akkouche S. Modeling cracks and fractures. *Vis Comput* 2005; 21: 717–726.
37. Müller M, Chentanez N and Kim TY. Real time dynamic fracture with volumetric approximate convex decompositions. *ACM Trans Graph* 2013; 32: 115:1–115:10.
38. Autodesk. Maya, autodesk.com/products/maya (accessed 16 October 2018).
39. Side Effects Software. Houdini, sidefx.com (accessed 16 October 2018).
40. Goffredo E. *A tool for procedural destruction in Houdini shattering + dynamics*. Master's Thesis, NCCA Bournemouth University, 2010.
41. Mazarak O, Martins C and Amanatides J. Animating exploding objects. In: *Proceedings of the 1999 Conference on Graphics Interface '99*, Kingston, Ontario, 2–4 June 1999, pp.211–218. San Francisco, CA: Morgan Kaufmann Publishers Inc.
42. van Gestel J and Bidarra R. Procedural modelling of destructible materials. In: *Proceedings of V Ibero-American Symposium on Computer Graphics*, Faro, Portugal, 1–3 June 2011.
43. Parker EG and O'Brien JF. Real-time deformation and fracture in a game environment. In: *Proceedings of the ACM SIGGRAPH/eurographics Symposium on Computer Animation*, New Orleans, LA, 1–2 August 2009, pp.156–166. ACM: New York.
44. Seymour M. Art of destruction (or Art of blowing crap up), fxguide.com/featured/art-of-destruction-or-art-of-blowing-crap-up (2011, accessed 16 October 2018).
45. Coumans E. Bullet real-time physics simulation, bulletphysics.org (accessed 16 October 2018).
46. Smith R. Open dynamics engine, ode.org (accessed 16 October 2018).
47. NVIDIA. PhysX, developer.nvidia.com/gameworks-physx-overview (accessed 16 October 2018).
48. Havok. Physics, havok.com/physics (accessed 16 October 2018).
49. Havok. Havok Destruction, havok.com/destruction (accessed 16 October 2018).
50. Havok. Havok Destruction 2012 SDK manual. *Product documentation*, 2012 (accessed 16 October 2016)
51. Autodesk. 3ds Max, autodesk.com/products/3ds-max (accessed 16 October 2018).
52. NVIDIA. APEX, nvidia.com/object/apex.html (accessed 16 October 2018).
53. PhysXInfo. Category:APEX, physxinfo.com/wiki/Category:APEX (accessed 16 October 2018).
54. PhysXInfo. APEX Destruction, physxinfo.com/wiki/APEX_Destruction (accessed 16 October 2018).
55. NVIDIA. Blast, developer.nvidia.com/blast (accessed 16 October 2018).
56. Crytek. CryEngine, cryengine.com (accessed 16 October 2018).
57. Amazon. Lumberyard, aws.amazon.com/lumberyard (accessed 16 October 2018).
58. Crytek. CryEngine manual:breakable objects, docs.cryengine.com/display/SDKDOC2/Breakable + Objects (accessed 16 October 2018).

59. Kihl R. Destruction masking in Frostbite 2 using volume distance fields. In: *SIGGRAPH course: Advances in Real-Time Rendering in 3D Graphics and Games*. Los Angeles, CA, 2010.
60. Pixelux. DMM, pixelux.com (accessed 16 October 2018).
61. Lisle C. Architectures for dynamic terrain and dynamic environments in distributed interactive simulation. *Technical report, Institute for Simulation and Training, University of Central Florida*, 1994.
62. Ellis C, Babenko P, Goldiez B, et al. Dynamic terrain for multiuser real-time environments. *IEEE Comput Graph Appl* 2010; 30: 80–84.
63. Hannay JE and van den Berg T. The NATO MSG-136 reference architecture for M&S as a service. In: *M&S Technologies and Standards for Enabling Alliance Interoperability and Pervasive M&S Applications (STO-MP-MSG-149)*, Lisbon, 20-21 October 2017.
64. Cloudgine Ltd. *(now part of Epic Games), Cloudgine, cloudgine.com* (accessed 26 February 2018).
65. Improbable. SpatialOS, improbable.io (accessed 16 October 2018).
66. RIEDP Product Development Group. *Reuse and interoperation of environmental data and processes (RIEDP) data model foundations*. SISO, 2017.
67. Hebert KJ and Sexton D. Dynamic synthetic environments through run-time modification of source data. In: *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2012*, Orlando, FL, 3–6 December 2012. Arlington, VA: I/ITSEC.
68. Siegfried R, van den Berg T, Cramp A, et al. M&S as a Service: expectations and challenges. In: *2014 Fall Simulation Interoperability Workshop (SIW)*, Orlando, FL, 8–12 September 2014. Atlanta, GA: SISO.
69. Watkins J, Tamash T and Campbell C. Future simulation paradigms and synthetic natural environments. In: *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2016*, Orlando, FL, 28 November–1 December 2017. Arlington, VA: I/ITSEC.
70. Sirigampola S, Mondesire S, Martin GA, et al. Development and analysis of a physics server for large virtual worlds. In: *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2016*, Orlando, FL, 28 November–1 December 2017. Arlington, VA: I/ITSEC.

Author biographies

Dr Ruben Smelik has been a scientist at TNO since 2007. He holds a MSc degree in computer science from Twente University. He earned a PhD degree from Delft University of Technology based on his thesis on the automatic creation of 3D virtual worlds. His current work focuses on innovations in the field of automated SE modeling for military simulation applications.

Freek van Wermeskerken has been a scientist at TNO since 2015 and has a MSc degree in Mathematics from the Vrije Universiteit Amsterdam. He specializes in optimization and modeling dynamical systems. His work focuses on creating models and algorithms to aid the field of safety and security on topics of analysis and decision support.

Robbert Krijnen has been a scientist at TNO since 1996 and has a MSc degree in computer graphics from Delft University of Technology. He specializes in the use of virtual environments for serious gaming applications. His current work focuses on innovations in the field of Augmented Reality and Virtual Reality applications for ‘‘Training & Education’’ and ‘‘Concept Development & Experimentation’’ applications in the military domain.

Frido Kuijper has been affiliated with TNO since the early 1990s. Originating from the fields of computer graphics and high-performance computing, he is now a senior research scientist, focusing on innovations for automatic generation of virtual environment models for simulation. He has broad experience in projects that served the military with effective use of simulation technology in training and mission support.