Structure-Aware Procedural Destruction

submitted by

Thomas AE. Smith

for the degree of Doctor of Engineering

of the

University of Bath

The Centre for Digital Entertainment

May 2014

Summary

This document presents a brief overview of the procedural destruction academic research field and related areas, and proposes an avenue of investigation for a new approach: structure-aware procedural destruction via annotation metadata and Answer Set Programming.

As the sophistication and quality of graphical rendering within games increases, there is growing demand for increased fidelity of all aspect of in-game environments. This fidelity typically comes at a cost however, particularly when the visual representation of in-game assets is subject to gradual degradation as a result of damage applied during gameplay. A number of approaches to providing these variant visual representations have been attempted, which typically rely on increasing artist investment to produce multiple permutations of each asset. In many cases, aspects of this investment could be automated through use of a procedural destruction system. Although a range of real-time procedural destruction systems of varied sophistication already exist, they typically do not account for the larger-scale structure of the assets being damaged, and there is little research in this area. In some contexts, ignoring relevant information about the larger structure of assets may lead to unrealistic destruction outcomes using existing systems, and so an approach that is able to integrate this information and produce identifiably appropriate damage representations would be desirable. In this proposal and literature review a possible method for developing a structure-aware procedural destruction system is described, and an approach developed to guide practical research and development in this area.

Chapter 1

Introduction

With improved hardware and techniques, the graphical fidelity of games in general is steadily increasing. However, there are some elements of interaction fidelity that have not increased at the same rate as overall graphical fidelity — notably the response of in-game objects to the application of virtual damage. While it remains infeasible to accurately physically simulate the destruction of in-game objects and environments in response to damage, a number of initial approaches have been made towards improving on the traditional abstract or smoke-and-mirrors renderings of damage within games.

1.1 Procedural Destruction

In many cases, while accurate physical simulation is still out of reach, systems have been developed to approximate appropriate behaviour for many kinds of in-game assets. Two popular subjects for this approach are large, planar structures such as windows or wooden barriers, which may exhibit realistic shattering and splintering behaviours, and smaller objects such as low walls, concrete barriers and ceramics, that typically fracture into unrecognisable component fragments.

Some genres of game — notably space- or naval-combat sims, feature large physical structures that can in the real world be subject to idiosyncratic recognisable forms of destruction — for example, hull denting and deformation as a result of impact, or separation of smaller components such as conning towers. The types of damage that each portion of such a ship may be subject to are dependant not only on the kind of damage applied, but also on the physical structure of the area affected and the supporting physical structures surrounding it. No existing procedural destruction systems apply knowledge about the structure of the affected object when generating possible damage outcomes, and so in this document a review of related research is

presented, and a promising avenue for further investigation is suggested.

1.2 Development Motivation and Context

The system will be developed within the context of a contemporary commercial space combat game, featuring a range of in-game destructible assets (spaceships, space station, and potential asteroids and other debris) and potential damage types (explosive missiles, beam energy weapons and directed electromagnetic pulse). Research will be motivated by a range of academic and commercial concerns — the primary benefits of the system to the industry partners will arise from reduced asset production cost overheads, and increased in-game visual fidelity and information affordances. From an academic perspective, it will be instructive to investigate the application of existing techniques in new areas, and to develop the optimisations and adaptations that will potentially make this possible.

Under the existing system assets are produced with multiple variations representing progressively increasing levels of damage received - ranging from pristine to completely destroyed. This incurs considerable duplication of effort on behalf of the artists producing the content, and thereby requires significant investment for each additional asset produced. It also results in a predictable, identical succession of visual renderings for an asset undergoing continuously increasing damage — regardless of the kind of damage or location of impact. Each asset has an internal 'structural integrity' measure that is appropriately decreased whenever it takes damage. At particular thresholds, the current mesh and/or texture is replaced by a variant displaying more damage effects. This has a number of negative consequences: the representation of damage is the same regardless of the actual damage received, harming the impression of fidelity. Similarly, within a threshold band the actual representation remains constant despite potential large changes in actual integrity, reducing the accordance of useful tactical information to the player. Finally, each additional ship variant that must be produced to represent a damage level represents additional artistic investment, and increases the overall production cost of an individual asset.

Each of these areas could be improved upon by a system that rendered damage procedurally according to location and kind. An overall structural integrity metric could still be tracked for gameplay purposes, but the overall rendering of damage could be more accurately representative of the actual damage received. This would improve the overall fidelity of the gameplay experience, and afford increased tactical information to players via the increased granularity of the damage representation. By allowing a system to dynamically generate damaged variants of pristine assets, the

artistic investment and overall production cost required per asset would go down.

1.3 The Problem of Scale

Providing high-fidelity destruction requires investment. Some approaches to rendering damage applied to in-game assets scale better than others. There are a number of different axes on which investment may scale - by the level of fidelity provided by the system, or by the number of assets which may be damaged. In an environment where a variety of assets may have a wide variety of different types of damage inflicted, attempting to realistically render the results of these damage events results in a combinatorial explosion that quickly becomes unmanageable both in terms of processing it in real time and when attempting to supply the assets that support rendering damaged structures.

For each addition asset created within a context that supports in-game destruction, it is important to ensure that the new asset is capable of expressing the full range of damaged states supported by other assets within the context — in order to enforce consistent and believable damage behaviour. In a system that uses two thresholds and pristine, damaged and destroyed variants of each asset, each new asset must also have a pristine, damaged and destroyed variant. This can make increasing the fidelity of a game prohibitive, as a new variant must be made for every single existing asset, as well as increasing the production costs of every future asset.

1.4 Related Research

A number of approaches have been take in order to attempt to automate the process both of determining what sort of damage should be rendered, and also procedurally generating the assets (meshes, textures, VFX) necessary in order to faithfully render this damage. To date, the majority of such procedural systems focus on a per-material or per-object approach that responds to a single kind of impact damage and merely varies the magnitude of the effect applied in order to respond to differences in damage kind — i.e. the difference between a single stray bullet impact and a nearby high-yield explosive detonation.

Where larger scale destruction effects are necessary, these are typically handled by a combination of small-scale procedurally destructible elements, and custom designer-written scripts that make use of higher level knowledge about the physical structure of the object being destroyed and the 'intent' of the destruction - whether it is simply intended to provide a visual effect, or will have some ludic ramifications on the players'

abilities or accessible areas within the game.

1.5 Constraint-based Solvers

One problem with any attempt to model the realistic destruction of large composite structures in 3D space — especially in an environment with multiple distinct damage kinds — is the combinatorial explosion that occurs when considering the possible outcomes of any particular damage event. In order to attempt to circumvent this issue, a declarative constraint-based solver may be used to select appropriate abstract damage consequence descriptions, according to specified constraints on the response of particular structures to particular damage types, and provided information about the structure of an asset, and the kind(s) of damage applied.

1.6 Implementation Constraints

There are a range of important constraints that are likely to be relevant to the research and implementation of such a structure-aware procedural destruction system. From an academic perspective, it is important to ensure that the system design maintains a loose coupling with the game context, in order to aid potential reuse within other contexts. The implementation, however, should be tightly integrated with the host system in order to allow effective utilisation by the developers that will comprise part of the primary end-users. Another important aspect of the implementation is the limited processor time available — the solution must run in real-time in a fraction of a frame, or calculations may be amortised across multiple frames. It is important to ensure that the system remains responsive, whilst also not interfering with the multitude of other processing requirements imposed by the game. As the renderings are likely to provide tactically-useful information to players during runtime, it is important to ensure that the appearance of damage is consistent across each clients' rendering. This is particularly challenging given the constrained network resources — it will be essential to minimise the amount of additional data sent over the network merely to ensure client synchronisation.

1.7 Document Overview

Having briefly introduced the relevant context and concerns, this document presents an overview of related work within both the academic and industry implementations in Chapter 2. There appear to be no direct precedents for this particular approach,

and so the final method is likely to be a synthesis of previous procedural destruction techniques and approaches borrowed from other fields — notably the structural-aware constraints-based system developed by Novelli et. al. using ASP [NDVPD12].

Based upon the previous work detailed, Chapter 3 outlines an approach to developing a project that suitably addresses the research goals for the system, which are also discussed. Chapters 4, 5 and 6 present some initial ideas about what aspects of the system will benefit from thorough evaluation and how to go about this process; a proposed timeline for the project based upon the needs of the enclosing project, and a suggestion as to the types of conclusion that may be drawn from investigation into the outcome and process of the project.

Chapter 2

Literature Review

To inform the development of the project, it is worth considering research in related areas. First, a range of existing approaches to procedural destruction systems within commercial games are considered — accompanied by academic overviews of particular techniques where available. Then, the concept of declarative solvers is introduced, with particular reference to existing projects and approaches that might provide promising initial directions. Finally, there is a brief overview of some of the other techniques that may be necessary in order to bind the abstract output of a solver system to a concrete visual representation within a game.

2.1 Existing Implementations

Any particular procedural destruction system is distinguished by the implementation chosen for two primary considerations: the internal metric or process used to select an appropriate visual representation of damage, and the means by which these selecte representations are displayed to the player(s). The internal model is often defined by the ludic environment of the system — design decisions will be made at the gameplay level about the particular model of damage most suited to the desired in-game experience. A suitable way of communicating this information to a player via visual (rarely, audible) representation can then be developed.

In order to provide context for the description of the proposed system, it is instructive to investigate the range of methods that have previously been used to model and represent damage and destruction within games to date. A number of the approaches described below are mutually incompatible, however there are a small number of subsets that are often used in combination with each other, and which may indicate useful techniques that could be applicable in the present domain. The sections below are

arranged roughly in order of increasing implementation complexity, and therefore also chronologically in order of representation fidelity.

2.1.1 No Destruction

In early and/or simple games, it is commonly the case that all game objects are simply indestructible, which requires no in-game damage metric or alternative visual rendering.
Examples: Pong, Passage

2.1.2 Scripted Destruction

The simplest possible form of destruction is the simple removal of an object in response to a particular in-game event — typically collision with a projectile fired by the player. More advanced implementations may replace the object with a visual effect such as an explosion.
Examples: Breakout, Galaga

2.1.3 Triggered Destruction

The overall 'health' or 'structural integrity' of an asset is tracked by an internal in-game metric. This may be exposed to the player via a user-interface element such as a health bar, but typically the visual representation does not vary as the health degrades.
Examples: Age of Empires, Homeworld

2.1.4 Art Swap

As above, an internal metric tracks damage received, but at various thresholds predefined thresholds the visual representation of an asset is replaced with a variant that displays more damage effects, resulting in a predictable degradation that is identical each time.
Examples: Strike Suit Zero, Red Faction Guerilla

2.1.5 Voxel-based Destruction

In games where the ludic environment is composed of, or can be easily approximated by, voxels (regular three-dimensional elements, typically cubes or tetrahedra), object or terrain may be destroyed or deformed by (re)moving individual voxels in response to damage [DT01].
Examples: Minecraft, Starmade

2.1.6 Destructible Materials

In the same way that per-material metadata may be stored about the render behaviour of an asset (specular/displacement maps), the fracture behaviour may be defined per-material and used by an in-game destruction system. Meshes may be realistically fractured into separate physics objects in response to damage [vG11].
Examples: The Force Unleashed, Gears of War 2

2.1.7 Scripted Systems

For larger-scale destruction in contemporary implementations, typically a mixture of handwritten scripts and destructible objects are used — for example, a collapse animation may be triggered once a pre-defined number of destructible supports have been removed.
Examples: Battlefield: Bad Company 2, The Cave

2.1.8 Academic Implementations

There appears to be very little research into real-time procedural destruction approaches in academic literature, with the topic most often arising as a suggestion for future work based upon the affordances provided by the output of a given procedural generator — c.f. potential voxel-based destruction of the rocks generated by [DDRT11]; a technique that may be useful for providing destructible asteroids. An effort has been made to survey existing industry implementations and provide a physically-accurate materials-based destruction approach [vG11], however as described the system is only suitable for comparatively small assets composed of homogeneous material (i.e. thin planar objects where varied material composition and structural information is not a relevant concern for destruction purposes).

2.2 Declarative Solvers

Though not typically applied to the destruction of in-game assets, there is a wealth of academic literature surrounding the use of constraints-based solvers and truth maintenance systems for problem solving [Doy78]. Given an appropriate representation of a search space in the form of constraints on the possible set of valid answers, a declarative solver is potentially able to discern reasonable solutions to the problem so described, and do so faster and more efficiently than alternative approaches [Bar03].

This facility has already been used by the Perpetuate project in order to predict the structural effects of earthquake damage for architectonic asset preservation. The

LOG-IDEAH tool replaces the weighty decision trees previously used for this task with a declarative approach that involves specifying the structural features of a given asset and then using an ASP solver to predict the most likely consequences of any form of earthquake damage applied [NDVPD12].

In order to replicate this approach for a real-time system in a space combat game there is a need for analogous damage-consequence information appropriate to the project context. Due to the safety requirements of actual space travel, there is a large existing body of physically-accurate prediction data about spacecraft damage behaviours. The European Space Agency provides a summary of such information [KFK00], albeit focusing on the prediction of spacecraft destruction processes specifically during re-entry. The consequences of receiving direct, intentional military aggression are less thoroughly covered, and the data pertains to existing spacecraft rather than the fictional spaceships that will be represented by assets within the game, but it is likely to provide a useful approximation as a starting-point for further development.

2.3 Relevant Rendering Techniques

In addition to the selection of an appropriate internal metric or process for determining the consequence of damage, as described above, it will be necessary to investigate suitable rendering techniques in order to clearly display that outcome to the player. There are a number of established games rendering techniques that may be appropriate, as described below.

2.3.1 Decal Application

One common approach to superficial damage representation is the application of simple damage decals such as bullet holes or scorch marks to asset surfaces. Modern decal application systems can cope with rendering many additional decals within a single frame [JYA06]. This is likely to be a suitable approach for rendering light damage with low processing load.

2.3.2 Procedural VFX Placement

Another popular method for clearly visually indicating the presence of damage on an asset is the use of VFX particle systems to represent fire, impact fragments, smoke or other emissions. In traditional implementations this can damage belief in the fidelity of a system if the smoke is emitted from a location physically distant from the actual point of damage — this can be solved in a procedural system by placing VFX only

within close proximity of impact.

2.3.3 Mesh Deformation

Since it will occasionally be necessary to render assets that are severely damaged but not entirely inoperative, mesh deformation may be a useful approach for representing heavy degradation of structure. There are a number of existing implemented systems, that represent a range of trade-offs between speed and physical accuracy. An important consideration is the effect of deformation on collision detection within the game — unlike the previous techniques, deformation may have potential ludic consequences if it alters the collision profile of an asset, and therefore care must be taken to ensure that modifications are synchronised across all connected clients. One possible approach to quickly rendering deformed structures is described by Morris et. al. in [MAP12], however since all output is determined at runtime via GPU shaders, it would be difficult to ensure cross-client synchronisation of damage appearance. In contrast, Stegmayr [Ste08] describes a deterministic physically-based method that reportedly still provides interactive framerates, however this is unlikely to be fast enough given the relatively small processor slice available during each frame. It may however be worth investigating with a view to constructing a library of pre-computed deformations for each mesh, if it becomes apparent that an optimisation of this nature would prove beneficial.

A number of generalised important considerations relating to procedural deformation — such as the potential of self-intersection — are presented by Zmugg et. al. in their work on procedural architecture using deformation-aware split grammars [ZTK*13]. Finally, Galoppo et.al. provide a fast deformation simulation approach with appropriate deformation behaviours encoded in dynamic textures [GOM*06].

Chapter 3

Proposed Approach

To deal with the combinatorial explosion of possible damage outcomes in an environment with multiple structure and damage types, a declarative approach may be taken to modelling structures, damage and outcomes. Assets may be annotated with material and structural information, and an ASP solver used at runtime to convert information about actual damage received into abstract descriptions of damage consequences — e.g. as a description: 'partial starboard lateral hull deformation, starboard minor fire and venting, dorsal laser scarring'. The game engine may then contain systems that are able to reify these damage consequence descriptions into concrete visual representations — in this example via dynamic mesh deformation, procedural VFX placement, and application of laser scarring decals to the appropriate areas.

3.1 Tool Integration

As a first step, it will be necessary to allow for artists and designers to annotate in-game assets — by providing meta-data detailing structural information such as hollow hulls or potentially weak connections. As a possible extension it may be feasible to analytically determine these features from mesh information alone, without requiring user interaction, or as part of a mixed-initiative interface that allows developers to curate aesthetically suitable solutions [YLA14].

3.2 Solver System

A loosely coupled solver system may then be implemented, that contains prior knowledge (constraints)about appropriate destruction behaviour, and is able to receive individual structure and damage information at run-time in order to generate abstract

damage consequence descriptions. These may be communicated with the full game engine using an appropriate independent protocol in order to maintain suitable implementation independence [LBP13].

3.3 Damage Rendering

Once an appropriate abstract consequence description has been selected by the solver, the game engine needs to be capable of rendering these effects to the player. The precise method used will depend on the kind and degree of damage, and is likely to range from procedural decal and VFX application to selection of deformed or separated mesh variations. In order to ensure real-time performance of the system, it is probable that a range of pre-calculation / pre-generation optimisations will need to be considered

3.4 The Problem of Scale — Revisited

In contrast to the previous art-swap approach, the procedural system requires a different investment focus, and incurs smaller production costs for each additional asset or fidelity increase. In a basic art-swap scenario with $s = 12$ ships and $f = 3$ fidelity thresholds (pristine, damaged, destroyed), $s \times f = 36$ assets must be created, and an additional $a \times f$ assets for every further a ships. If a fidelity increase is desired (say, increasing granularity from 'damaged' to 'light damage' / 'heavy damage') then t new assets are needed, where t is the total number of ships.

The procedural destruction system itself will require an initial implementation investment i — however assuming the implementation supports the same three fidelity levels then the total investment for the first s ships is merely $i + s$; a additional ships require only a new assets; and a fidelity improvement requires investment in the system alone, and will apply both retrospectively to existing ships, and automatically to all new ones. This means that the procedural system provides a much better return on investment, and lowers the production cost of additional in-game assets.

Chapter 4

Evaluation

There are a number of possible methods for evaluating the effectiveness and appropriateness of the approach, as the developed system has two distinct groups of primary stakeholders: the developers, and the players. The developers will wish to ensure that the system is 'sufficient' in some sense — that is, that it is able to provide an appropriate damage rendering for any given in-game scenario, and fulfils the desire to provide increased fidelity and tactical information to the players. The developers will also care about the usability of the artists' or designers' annotation tools, and the tunable parameters of the system. The players, in contrast, are merely likely to care that the system is 'performant' — that it does not have a noticeable negative impact on other aspects of the game, particularly speed. They may also appreciate the increased variety, tactical information and visual spectacle provided by the system, but these are likely to be secondary concerns.

4.1 Expressive Range and Fidelity

To ensure that the system improves upon the levels of fidelity available via simpler methods, it is important that each generated rendering accurately represents the outcome of the specific kind of damage applied to that specific structure. To a degree, these relationships will be encoded within the constraints initially provided to the system, but evaluation will be necessary to ensure that it is responding as intended. There is little academic literature concerning this aspect of evaluating procedural systems — the only published article in the area highlights the importance of this area of evaluation, and the lack of wider awareness of the issue of successfully analysing expressive range [SW10].

4.2 User Acceptance Testing - Developers

As the system will be developed in co-operation with the developers that are likely to become some of the primary end-users, it will be possible to take advantage of a live feedback cycle throughout development. The primary concerns are likely to be about responsiveness, effectiveness and ease of use, and whilst informal feedback will be useful for guiding development related to these concerns, it would also be instructive to conduct a round of evaluation questionnaires in order to provide a degree of quantitative data relating to the use of the system. The ideal time for this would be before the system approaches completion, in order to allow time for modification and improvements to be made in response to feedback. Once the project itself approaches completion and the system has been finalised, it would also be useful to arrange a final evaluation questionnaire for comparative purposes.

4.3 Performance

integration with a real-time game means that the timely performance of the system is as serious concern. It may become appropriate to perform compile time optimisations, pre-calculation and pre-caching of possible solutions in order to ensure that during a match the system is able to provide appropriate damage solutions in real time. Writing in 2008, Boenn et. al. [BBDV*08] suggest that while some of the faster ASP solvers at the time were responsive enough to provide an interactive experience while generating melodies, they did not at the time provide real-time performance. There appears to be no more contemporary research that indicates significant recent speed increases, and so this is likely to remain an active research area for the project.

Evaluation of the performance of the system within the runtime context of the game environment is most likely to be performed via frame-rate comparisons throughout development. As the project should maintain a loose coupling with the game itself, it should remain relatively easy to selectively disable it for the purpose of comparison tests across similar in-game scenarios. A range of game events may be investigated both with and without the procedural destruction system enabled, and then logs of in-game metrics including frame-rate may be analysing in order to determine the impact of the processing load required by the implemented solution.

4.4 User Acceptance Testing - Players

As the solution will initially be developed within a game that is being released on an Early Access platform, it will be possible to perform A/B testing and receive user

feedback on varied incarnations of the system during the project. This may range from informal feedback via community channels to focused user surveys designed to elicit specific impressions of all aspects of the game — including the procedural destruction system. In particular, it would be useful to evaluate the impact of the system on user enjoyment and effectiveness, specifically possible increased satisfaction when applying damage to 'enemy' ships and structures, and increased passive awareness of the status of their own and other visible ships for tactical decision purposes.

Chapter 5

Timeline

Due to the commercial nature of the project context, is is likely that a completed version of the system will be needed within the next year and a half, and an initial working prototype will be required much sooner. The game that the system is initially being developed for is following an 'Early Access' release strategy — that is, a small number of interested players will be experimenting with early versions of the game from a point well before its official release. Their feedback will help to shape the development of the game as a whole, and possibly will also have an impact on the functionality required of the procedural destruction system. As it will be instructive to begin receiving feedback as soon as possible, a sensible development strategy would be to follow an Agile methodology — develop a minimal functional prototype initially, and then incrementally improve it in order to approach the final system.

The Early Access period begins in six months, and based upon the proposed approach above it seems reasonable to aim for a minimal damage system that tracks the location and kind of damage, and applies appropriately selected decals to the correct parts of ships in response. Further development can be guided by player and designer feedback, but is likely to follow the prerequisite ordering previously observed — i.e. provide an integrated approach for structural annotation, develop an ASP-based system that can combine this with the damage information to generate abstract consequence descriptions, and then implement an in-engine solution to visually render these appropriately. Each objective will necessarily introduce a number of sub-tasks required in order to complete them fully, but these are difficult to accurately specify at this initial stage. Once the system is completed and the game is released there will be a necessary period of support and further development, in order to provide for unexpected cases and projected future requirements, and thereby allow continued use of the system for ongoing asset production — which will extend well after the game's release date.

Chapter 6

Conclusion

At this early stage of the project it is difficult to suggest any definite conclusions, however it is hoped that the process of developing the system will help to answer a range of interesting research questions. One major issue is the question of whether ASP will be viable in real-time for solving the combinatorial explosion problem, and if not which optimisations will be necessary in order to benefit from the power of ASP solvers without an associated run-time impact. Investigation will hopefully reveal appropriate methods for rendering ASP-provided solutions in a convincing manner.

Bibliography

[Bar03] Baral C.: *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press, 2003.

[BBDV*08] Boenn G., Brain M., De Vos M., et al.: Automatic composition of melodic and harmonic music by answer set programming. In *Logic Programming*. Springer, 2008, pp. 160–174.

[DDRT11] Dart I. M., De Rossi G., Togelius J.: Speedrock: procedural rocks through grammars and evolution. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* (2011), ACM, p. 8.

[Doy78] Doyle J.: Truth maintenance systems for problem solving.

[DT01] Davison C., Tang W.: Deformable terrain generation for real-time strategy game. *Proc. of Eurographics Short Presentations, Manchester, UK* (2001), 1–8.

[GOM*06] Galoppo N., Otaduy M. A., Mecklenburg P., Gross M., Lin M. C.: Fast simulation of deformable models in contact using dynamic deformation textures. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2006), Eurographics Association, pp. 73–82.

[JYA06] Jing W., YingHui C., AiMin H.: A post-processing decal texture mapping algorithm on graphics hardware. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications* (2006), ACM, pp. 99–104.

[KFK00] Klinkrad H., Fritsche B., Kashkovsky A.: Prediction of spacecraft destruction during uncontrolled re-entries. *EUROPEAN SPACE AGENCY-PUBLICATIONS-ESA SP 468* (2000), 485–492.

[LBP13] Lee J., Baines V., Padget J.: Decoupling cognitive agents and virtual environments. In *Cognitive Agents for Virtual Environments*. Springer, 2013, pp. 17–36.

[MAP12] Morris D. J., Anderson E. F., Peters C.: A modular framework for deformation and fracture using GPU shaders. In *Virtual Systems and Multimedia (VSMM), 2012 18th International Conference on* (2012), IEEE, pp. 267–274.

[NDVPD12] Novelli V., De Vos M., Padget J., D'Ayala D.: Log-ideah: ASP for architectonic asset preservation.

[Ste08] Stegmayr C.: Procedural deformation and destruction in real-time.

[SW10] Smith G., Whitehead J.: Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (2010), ACM, p. 4.

[vG11] van Gestel J.: *Procedural destruction of objects for computer games*. PhD thesis, Department of Mediamatics Faculty of EEMCS, Delft University of Technology, 2011.

[YLA14] Yannakakis G. N., Liapis A., Alexopoulos C.: Mixed-initiative co-creativity. In *Proceedings of the ACM Conference on Foundations of Digital Games* (2014).

[ZTK*13] Zmugg R., Thaller W., Krispel U., Edelsbrunner J., Havemann S., Fellner D. W.: Procedural architecture using deformation-aware split grammars. *The Visual Computer* (2013), 1–11.