Spacetime Constraints for Continuous Gameplay

Akshar Vandara
Khoury College of Computer Sciences
Northeastern University
Boston, Massachusetts, USA
vandara.a@northeastern.edu

Seth Cooper
Khoury College of Computer Sciences
Northeastern University
Boston, Massachusetts, USA
se.cooper@northeastern.edu

Abstract

Recent work has explored the use of spacetime representations in procedural level generation. These represent time as an explicit dimension in a constraint-solving process to generate both a level and a solution. Changes along the time dimension are constrained by gameplay mechanics, ensuring that the generated level is solvable. However, in previous work gameplay mechanics have generally been discrete and tile-based. In this work, we explore the use of spacetime constraints for continuous, physics-based gameplay in a platformer game, using fixed-point in an SMT solver to solve for both the level layout and the player's path through the level. We demonstrate applications in level generation, generating levels from paths, solving levels, and playing the game.

Keywords

procedural content generation, constraint solving, spacetime optimization

ACM Reference Format:
Akshar Vandara and Seth Cooper. 2026. Spacetime Constraints for Continuous Gameplay. In *Foundations of Digital Games (FDG '26), August 10–13, 2026, Copenhagen, Denmark.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3815598.3815715

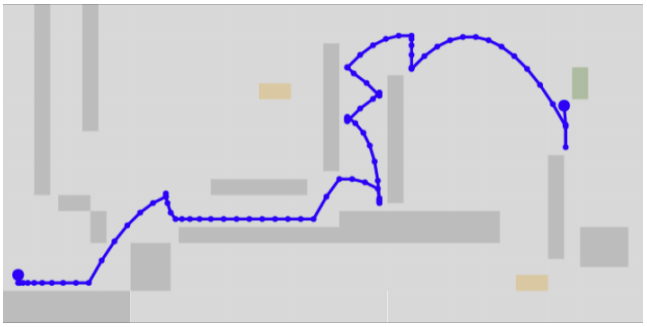

Figure 1: Using spacetime constraints to find a solution to a hand-authored level in a continuous, physics-based platformer game.

1 Introduction

One approach to ensuring the completabiliy of procedurally generated levels is to simultaneously generate an example solution along with the level. To this end, work has explored the use of spacetime representations: representing time as an explicit dimension, which can be used in a constraint-solving process to generate both a level and a solution. In this approach, changes along the time dimension are constrained by gameplay mechanics, and the level must be solved by the end, thus ensuring that the generated level is solvable by the given mechanics.

However, in previous spacetime constraint work, gameplay mechanics have generally been discrete and tile-based [2, 24]. This limits the approach mainly to puzzle-style games such as Sokoban [22].

To expand the types of games that the spacetime approach can apply to, in this work, we explore the use of spacetime constraints for continuous, physics-based mechanics in a platformer game. Inspired by related work in animation [25], the constraints between timesteps are applied to a platformer character with continuous movement and their position, velocity, and acceleration, along with collision handling. We use the Z3 SMT solver [4], with a fixed-point representation, to solve for level layout and the player's path through the level.

We demonstrate applications in level generation, generating levels from paths, solving levels, and playing the game. Although the approach is relatively slow and generates simple levels, we hope to build on these initial steps to expand spacetime constraints to more continuous physics-based games.

2 Related Work

In procedural level generation [16], ensuring completability of generated levels is an important consideration. Some work incorporates a generate-and-test approach, where levels are generated and then tested by an agent [26]. This can often be incorporated into search-based approaches, evolving levels toward agent completability [15, 23]. Games with continuous mechanics often employ this approach.

Also closely related to our work are approaches that simultaneously generate an example solution along with the level. This approach is often taken in a constraint-based framework [1, 2, 10], although has also been used in other techniques such as genetic algorithms [9] and machine learning by embedding path information in levels [20]. This approach often applies to games with discrete mechanics such as puzzles, and even when applied to games like platformers, may discretize their continuous movement to tiles [1, 20].

A number of general models for games have been developed, to allow understanding, playing, and reasoning about games as well as agent development. These include the Game Description Language [7], BIPED [17], GDL-II [21], LUDOCORE [18], Ceptre [8], and Ludii [19]; these are often applied for discrete, board-game-like

FDG '26, Copenhagen, Denmark

ACM ISBN 979-8-4007-2495-4/26/08
https://doi.org/10.1145/3815598.3815715

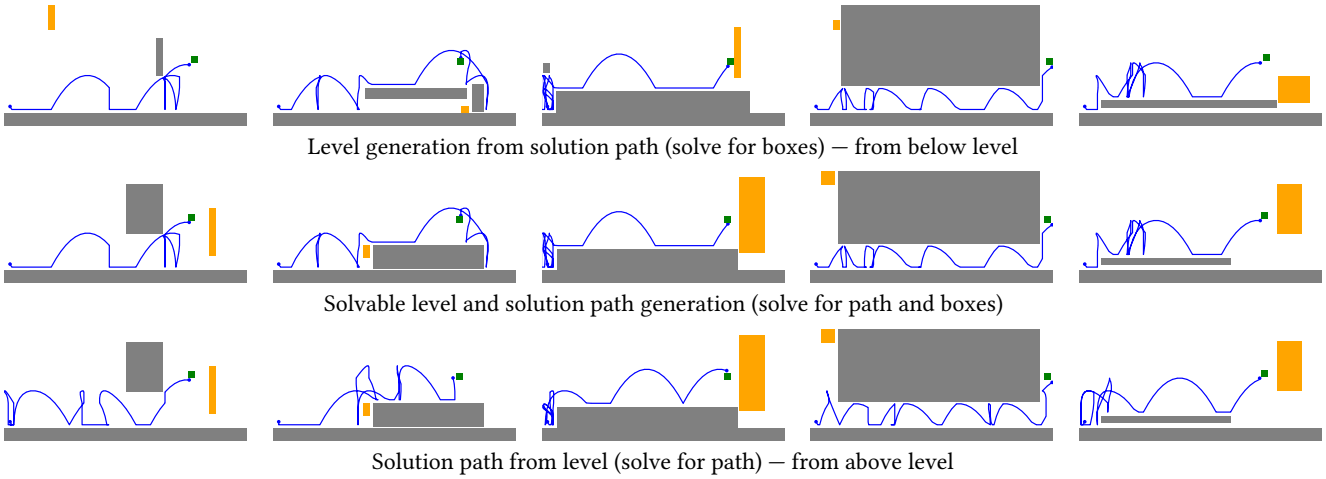

Figure 2: Generating levels. The middle row shows simultaneous generation of levels with a solution path. The top row shows a new arrangement of boxes generated from the solution path of the middle level by constraining the solution path. The bottom row shows a new solution path to the same arrangement of boxes as the middle level by constraining the arrangement of boxes.

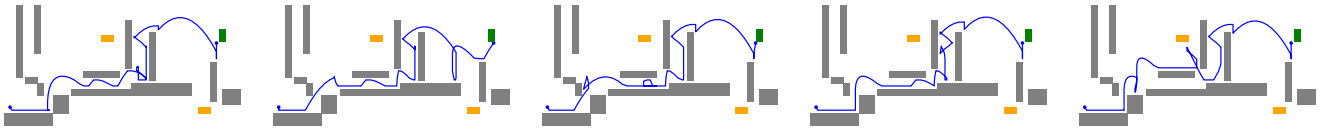

Figure 3: Solution path from level (solve for path) — from hand-authored level. Pathfinding can be done by constraining the boxes to be from a hand-authored authored level and solving for a solution path through the level.

games. Petri nets have also been applied to game analysis [12]. Closely related to our work is HyPED [11], which uses hybrid automata to model action games, for example more precisely modeling platformers with velocity.

Our work also draws on spacetime constraints for animation, developed as complimentary to keyframing with the goal of more physically realistic animations [25]. They have since been expanded to applications such as generating motion transitions [13], motion editing [5], capturing physically-based movement styles [6], and deformable objects [14].

3 Approach

We developed a simple custom platformer game for this work. Code used is at https://github.com/crowdgames/continuous-spacetime/tree/pcg2026. All the elements of the level are represented as axis-aligned boxes for simplicity. The elements are the player, the start, goals, blocks, and hazards. Only the player can move during gameplay. The player can move left and right, jump when they are on the ground, cling to and slide down walls, and wall jump when they are clinging to a wall.

We set up a constraint problem to be solved by the Z3 SMT solver [4]. For the continuous gameplay variables, we used a fixed-point representation with three fractional digits by encoding them in into integer bit-vectors. The use of fixed-point rather than floating-point was done in early tests to improve performance, though further explorations of the tradeoffs remains future work.

The variables to be used in the constraint problem are pre-allocated and constrained before running the solver.

Each potential box — a start (if needed), goal, hazard, or block — had variables for if it is active and included in the level (Boolean) and its 2D coordinates and size (fixed-point). As boxes do not move, there was simply one set of each such variables per potential box regardless of the temporal length of the constraint problem.

The game state, consisting of the player's information, was set up as variables in a number of timesteps. There is one set of timestep variables for each timestep considered in the constraint problem. Each timestep consists of variables for the current status (integer, representing *active*, *won*, or *lost*), the player's 2D *resolved* position and velocity after collision response (fixed-point), the player's 2D *desired* position before collision response (fixed-point), if the player is on the ground, clinging to a wall to the left, or clinging to the right (Boolean), and if the left, right, up or down buttons are pressed (Boolean).

The main constraints on the variables are between timesteps. Timestep constraints constrain how the values of the variables of neighboring timesteps can differ; here we summarize them at a conceptual level. They are essentially a physics update, but expressed as constraints, and make extensive use of Z3's logical connectives such as `Implies`, `If`, `And`, and `Or`.

There is a constraint that if a timestep has the *won* or *lost* status, then the following timestep is also in that status and the player stays in the same place.

Mode	Time, sec.	
Solve for path and boxes	197.7	(112.9)
Solve for boxes (from path and boxes)	6.7	(1.2)
Solve for path (from path and boxes)	47.3	(13.1)
Solve for path (from hand-authored)	771.5	(382.7)

Table 1: Summary of different generation application results (25 runs each). Mean (and standard deviation) given.

Other constraints incorporate the timestep having the *active* status and constrain the next timestep's player update. The player's position is handled by having both a *desired* position — where their velocity would take them if there were no collisions — and a *resolved* position — where their desired position ends up after resolving collisions. The desired position in the next timestep is constrained to be the current position plus current velocity. The most involved part involves collision response. A system of constraints is set up such that if the player's bounding box at their *desired* position overlaps that of a block, the player's *resolved* position is constrained to place their bounding box outside the block; if there are no overlaps, their resolved position is constrained to be the desired position. The next timestep's velocity is also constrained to be zero along the axis of a collision. The on ground and wall cling Booleans have their values constrained based on player contact with blocks. The player's velocity in the next timestep is constrained based on collisions, gravity, on ground and wall cling, and the buttons. The next timestep's status is constrained to be *lost* if the player collides an active hazard, *won* if the player collides with an active goal, or *active* otherwise.

In generation problems, there is also a constraint that the player starts in the start box and the final timestep has the *won* status.

4 Applications

Here we describe some applications of the proposed approach. Different effects can be achieved depending on which variables are constrained to specific values, which are allowed to vary, and how many timesteps are used. In some applications, we used a hand-authored level by constraining box locations. In each case we generated 25 levels to explore variety and timing. Timing information is given in Table 1.

When generating paths, inputs from the player could only be changed every 3 frames, only press one button at a time, and not use the down button. These limitations were used to simplify the constraint problem.

Solvable level and solution path generation (solve for path and boxes) — To generate a solvable level, a level along with its solution path can be generated, by allowing both the boxes defining the level and the solution path to vary. We used 106 timesteps, with 1 potential goal, 5 potential boxes, and 2 potential hazards (note that it is possible to end up with fewer than this if some are made inactive). The start box is constrained to be active in the lower-left, the goal to be active on the right, and one block to active along the bottom. Screenshots are given in Figure 2 (middle).

Level generation from solution path (solve for boxes) — Given a path through a level in the form of a sequence of locations, the timestep part of the constraint problem can be constrained to this while allowing the boxes to vary. This generates a new level for which the given path is a possible path through the level, potentially allowing generating variations on a level. This is similar to the discrete approach taken in previous work [3]. We took paths from the above generated levels and used them to generate new levels. Screenshots are given in Figure 2 (top).

Solution path from level (solve for path) — Complimentarily to the previous, by constraining the boxes and allowing the timesteps to vary, a path through the level can be found. This is similar to normal pathfinding, but we expect much slower and thus may not be preferable. We applied this to both generated levels — screenshots in Figure 2 (bottom) — and the hand-authored level — screenshots are given in Figure 3.

Gameplay updates (solve for next path timestep) — The constraint solver can also be used as an update function. This is done simply by having two timesteps, constraining the first to be the current state of the game and input, and then solving for the following timestep. This may be useful as the in-game update function will then match the one used during level generation and not need to be implemented separately. We found this fast enough to play the level we hand-authored smoothly. Videos of gameplay are provided at https://osf.io/q74jp/.

5 Discussion and Conclusion

Although we have performed an initial exploration of continuous gameplay spacetime constraints, it remains to be seen in future work if they can find useful applications or be meaningfully integrated into tools. Overall, the solving process is fairly slow, and the generated levels are quite simple. These provide areas for future work using the approach. When generating levels, we noted that in some places the solver's paths can slip through or around parts of the level — likely due to the timestep size and solver's precise control over paths; this may be a way that a solver could assist with testing levels.

We envision some applications where spacetime constraints may be useful. Generating a level from a solution path may be applied by, for example, creating developer tools for drawing solution paths and editing them by dragging points or "keyframes" and having the level update accordingly. It could be useful to constrain other aspects of solutions such as player velocities or wall cling, and constrain partial solutions, such as part of the path and some of the boxes, possibly infilling or completing a partially-authored level. If integrated into a game's runtime, there may be applications such as allowing players to move boxes around while ensuring playability, or to generate levels that allow players to practice their own desired sequence of inputs. There may be applications in developer tools such as level testing or debugging by solving for paths through a level with certain properties; though this could likely be done with existing pathfinding techniques.

In the future we would also like to combine with other mechanics, such as moving platforms and enemies or lock-and-key. We are also interested in exploring other physics-based game styles, such as Angry Birds, Cut the Rope, Peggle, or Asteroids.

Acknowledgments

The authors thank Chris Martens for helpful discussions.

References

[1] Seth Cooper. 2022. Sturgeon: Tile-Based Procedural Level Generation via Learned and Designed Constraints. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 18, 1 (Oct. 2022), 26–36. doi:10.1609/aiide.v18i1.21944

[2] Seth Cooper. 2023. Sturgeon-MKIII: Simultaneous Level and Example Playthrough Generation via Constraint Satisfaction with Tile Rewrite Rules. In *Proceedings of the 18th International Conference on the Foundations of Digital Games (FDG '23)*. Association for Computing Machinery, New York, NY, USA, 1–9. doi:10.1145/3582437.3587205

[3] Seth Cooper and Matthew Guzdial. 2023. Path2level: Constraint-Based Level Generation from Paths. In *2023 IEEE Conference on Games (CoG)*. 1–4. doi:10.1109/CoG57401.2023.10333205

[4] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, Berlin, Heidelberg, 337–340. doi:10.1007/978-3-540-78800-3_24

[5] Michael Gleicher. 1997. Motion Editing with Spacetime Constraints. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics (I3D '97)*. Association for Computing Machinery, New York, NY, USA, 139–ff. doi:10.1145/253284.253321

[6] C. Karen Liu, Aaron Hertzmann, and Zoran Popović. 2005. Learning Physics-Based Motion Style with Nonlinear Inverse Optimization. *ACM Trans. Graph.* 24, 3 (July 2005), 1071–1081. doi:10.1145/1073204.1073314

[7] Nathaniel Love, Timothy Hinrichs, and Michael Genesereth. 2006. *General Game Playing: Game Description Language Specification.* Technical Report LG-2006-01. Stanford Logic Group.

[8] Chris Martens. 2015. Ceptre: A Language for Modeling Generative Interactive Systems. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 11, 1 (2015), 51–57. doi:10.1609/aiide.v11i1.12784

[9] Matthew McConnell and Richard Zhao. 2025. From Frustration to Fun: An Adaptive Problem-Solving Puzzle Game Powered by Genetic Algorithm. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 21, 1 (Nov. 2025), 277–286. doi:10.1609/aiide.v21i1.36831

[10] Mark J. Nelson and Adam M. Smith. 2016. ASP with Applications to Mazes and Levels. In *Procedural Content Generation in Games*, Noor Shaker, Julian Togelius, and Mark J. Nelson (Eds.). Springer International Publishing, Cham, 143–157. doi:10.1007/978-3-319-42716-4_8

[11] Joseph Osborn, Brian Lambrigger, and Michael Mateas. 2017. HyPED: Modeling and Analyzing Action Games as Hybrid Systems. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 13, 1 (Oct. 2017), 87–93. doi:10.1609/aiide.v13i1.12937

[12] Christian Reuter, Stefan Göbel, and Ralf Steinmetz. 2015. Detecting Structural Errors in Scene-Based Multiplayer Games Using Automatically Generated Petri Nets. In *Proceedings of the 10th International Conference on the Foundations of Digital Games*.

[13] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. 1996. Efficient Generation of Motion Transitions Using Spacetime Constraints. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 147–154. doi:10.1145/237170.237229

[14] Christian Schulz, Christoph von Tycowicz, Hans-Peter Seidel, and Klaus Hildebrandt. 2014. Animating Deformable Objects Using Sparse Spacetime Constraints. *ACM Trans. Graph.* 33, 4 (July 2014), 109:1–109:10. doi:10.1145/2601097.2601156

[15] Noor Shaker, Mohammad Shaker, and Julian Togelius. 2013. Evolving Playable Content for Cut the Rope through a Simulation-Based Approach. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 9, 1 (Oct. 2013), 72–78. doi:10.1609/aiide.v9i1.12690

[16] Noor Shaker, Julian Togelius, and Mark J. Nelson. 2016. *Procedural Content Generation in Games.* Springer.

[17] Adam Smith, Mark Nelson, and Michael Mateas. 2009. Prototyping Games with BIPED. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 5, 1 (Oct. 2009), 193–194. doi:10.1609/aiide.v5i1.12344

[18] Adam M. Smith, Mark J. Nelson, and Michael Mateas. 2010. LUDOCORE: A Logical Game Engine for Modeling Videogames. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. 91–98. doi:10.1109/ITW.2010.5593368

[19] Matthew Stephenson, Éric Piette, Dennis J.N.J. Soemers, and Cameron Browne. 2019. An Overview of the Ludii General Game System. In *2019 IEEE Conference on Games (CoG)*. 1–2. doi:10.1109/CIG.2019.8847949

[20] Adam Summerville and Michael Mateas. 2016. Super Mario as a String: Platformer Level Generation via LSTMs. *arXiv:1603.00930 [cs]* (March 2016). arXiv:1603.00930 [cs]

[21] Michael Thielscher. 2011. GDL-II. *KI - Künstliche Intelligenz* 25, 1 (March 2011), 63–66. doi:10.1007/s13218-010-0076-5

[22] Thinking Rabbit. 1982. Sokoban.

[23] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. 2011. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186.

[24] Akshar Vandara, Kaylah Facey, and Seth Cooper. 2025. Spacetime Level Generation and Editing with Constraints from Examples. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 21, 1 (Nov. 2025), 142–152. doi:10.1609/aiide.v21i1.36818

[25] Andrew Witkin and Michael Kass. 1988. Spacetime Constraints. *ACM SIGGRAPH Computer Graphics* 22, 4 (June 1988), 159–168. doi:10.1145/378456.378507

[26] Georgios N. Yannakakis and Julian Togelius. 2025. *Artificial Intelligence and Games* (2nd ed.). Springer Nature Switzerland, Cham. doi:10.1007/978-3-031-83347-2