NuFrameIT: Serious Games via Computation and Reasoning

Stefanie Heim, Michael Kohlhase, Silas Kuder, and Florian Rabe

Computer Science, FAU Erlangen-Nürnberg

Abstract. In the FrameIT framework for serious games, we have previously experimented with ideas for co-maintaining virtual worlds (such as those maintained by game engines like Unity) with logical worlds (such as those maintained by knowledge representation and reasoning systems). By treating the virtual world as a physical model of the logical world, it achieves a coupling that tracks the player's knowledge about the world. And the player can interactively extend this knowledge with measurements taken in the virtual world and with deductions about them in the logical world.
In this paper we present a major reconceptualization, NuFrameIT. It uses a new representation language for the logical worlds that combines both axiomatic and efficiently executable knowledge descriptions, and allows for a more dynamic and fine-grained maintenance of the logical world. Moreover, it integrates HTML-based techniques for displaying mathematical formulas interactively into the game engine.

Keywords: serious games, FrameIT, UniFormal, virtual world, mathematical knowledge

1 Introduction

The FrameIT framework for serious games [CICM20] explores how to use mathematical knowledge management (MKM) techniques for developing serious—i.e. educational—math games. Such games combine four aspects:
A1 a virtual world which is explored in the game
A2 the player/learner's knowledge about the virtual world
A3 the mathematical background of the topic to be conveyed by the game
A4 the didactic ability to convey the topic effectively.
(**A1**) is usually (and usefully) implemented in a game engine like Unity [UNTY] or Unreal [UNR]. Those employ complete concrete representations including, e.g., positions, velocities, and bounding boxes. These are well-suited for simulating an immersive learning environment for the user.

Most serious games implement the remaining three aspects via custom code in the programming language underlying the game engine. This has the drawback that the abstract mathematical knowledge (e.g., an analytic treatment of trajectories), player knowledge, and the way both can interact are only maintained ad hoc and implicitly in a system designed for fast numeric computation.

Therefore, the FrameIT system separates concerns by using a game engine as **frontend** for the maintenance and display of (**A1**), and a logic-based knowledge management system as **backend** to explicitly formalize (**A2**) and (**A3**) as a set of logical theories. The latter focuses on a knowledge-oriented high-level perspective that is well-suited for logical problem-solving. (Good treatment of (**A4**) is still an open question, but techniques from [Ber+23; ALeA] should apply.)

The concrete implementation UFrameIT [UFO] used the Unity engine and the MMT system [Rab13; RK13], which provides knowledge management services including elaboration of theories, type checking, and simplification. The theory graph paradigm underlying MMT has been an excellent fit for the FrameIT framework: the player knowledge (**A2**) can be represented as a dynamic theory that grows as the player's knowledge about the worlds expands, called the **situation theory**. It can be seen as an abstraction of the virtual world, or a mathematical model of those parts of the world that are relevant for the current learning goal. The background knowledge (**A3**) is represented as a modular knowledge graph in MMT.

To the usual ways for interacting with a virtual world, FrameIT adds two game mechanics to allow for mathematical exploration by the player: **Gadgets** allow actions like defining points, specifying lines, and measuring distances in the virtual world. These generate new **facts**, player knowledge that is represented by visuals in game, and by an extension of the situation theory in the backend. **Scrolls**[1] allow the use of small theories in the background knowledge that represent the abstract mathematical theorems that the game is meant to teach. To apply a scroll, the player instantiates the free variables of the scroll with objects in the situation theory. Then the situation theory is extended by taking the pushout of the scroll along this instantiation in the category of MMT theories and morphisms [CICM20].

We have been able to build several prototypical serious games based on this framework, validating the overall approach. But these prototypes also unveiled serious limitations in the initial design:

L1 The text display facilities in game engines are ill-suited for the mix of prose and formulas, ubiquitous in the communication of mathematical ideas.

L2 MMT's bias towards formal logic allowed computation only via symbolic rewriting and pushout. This was particularly problematic when representing knowledge that is mathematically determined but not known to the player.

(**L1**) fundamentally limited the complexity of topics that were worth attempting, and (**L2**) led to the use of workarounds that made the resulting system too complex to scale up the game development.

Contribution We present a major redesign of the system, which we call *NuFrameIT*. Firstly, we have replaced the MMT backend with the new UniFormal language [CICM25] that was partly inspired by the complexity of the FrameIT design. UniFormal combines both logical/mathematical knowledge representation features using theories with object-oriented mathematical computation features akin to the

[1] The name is intended to invoke the image of a piece of magical parchment.

Axiom computer algebra system [JS92]. Contrary to MMT, UniFormal's implementation is systematically lean and can be compiled into a small, easily packagable binary or JavaScript file. Secondly, we employ the WebView plugin [Ess24] for the dynamic and interactive display of mathematical content and integrate it with Unity in a way that allows seamlessly displaying text, diagrams, and formulas inside the game engine. This enabled us to replace the makeshift web service-based connection between Unity and MMT with a tightly coupled integration of Unity, UniFormal, and WebView. By calling directly into UniFormal's data structures and operational semantics, the representation of the situation theory becomes much simpler and scroll application much smoother.

In the sequel, we explain the key innovations both from the frontend and the backend perspective.

2 Frontend Perspective

We use the well-known word problem of determining the height of a tree via the intercept theorem as a running—didactically motivated—example. Figure 1 shows a diagram of the idea: a person can determine the height h of a given (vertical) tree by sighting the top and bottom of the tree over a (vertical) stick of known length h_a. If we know the distance d between person and tree and the arm length d_a, we can compute the height as $h = \frac{d}{d_a} \cdot h_a$.

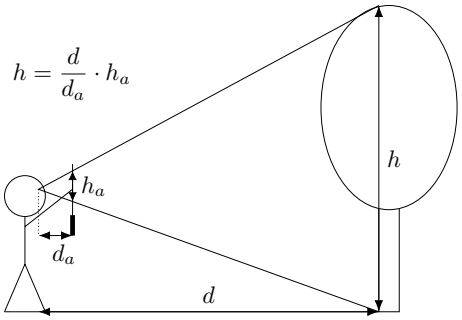

Fig. 1. Tree Height via Intercept Theorem

Figure 2 shows the corresponding construction in the virtual world halfway towards a solution. The right side shows the Unity rendering of the virtual world. The toolbar at the bottom contains the gadgets that can be used for, e.g. defining points and measuring distances. The player has previously employed these to mark the points B through F, and to measure the distances $[DF]$, $[DC]$ and $[EF]$.

The scroll bar on the right of Figure 2 has a selection of the available scrolls. The player has selected the intercept scroll because she deems it useful for solving the tree height problem. A scroll consists of two different representations of the same knowledge, and a formal description of how they are connected. The representation of a scroll in the backend is a UniFormal theory, see Section 3. The representation of a scroll in the frontend is a small narrative document that corresponds to explanations in a mathematics textbook and may also include other media like the diagram on the bottom left. The left side of Figure 2 shows such a document. It describes how to apply the intercept theorem in a concrete situation. Note that it is not a static image but an HTML webpage, using MathML for the formulas, SVG for the triangle, and JavaScript for interactivity. The interactivity is used to support the player in applying the theorem in the

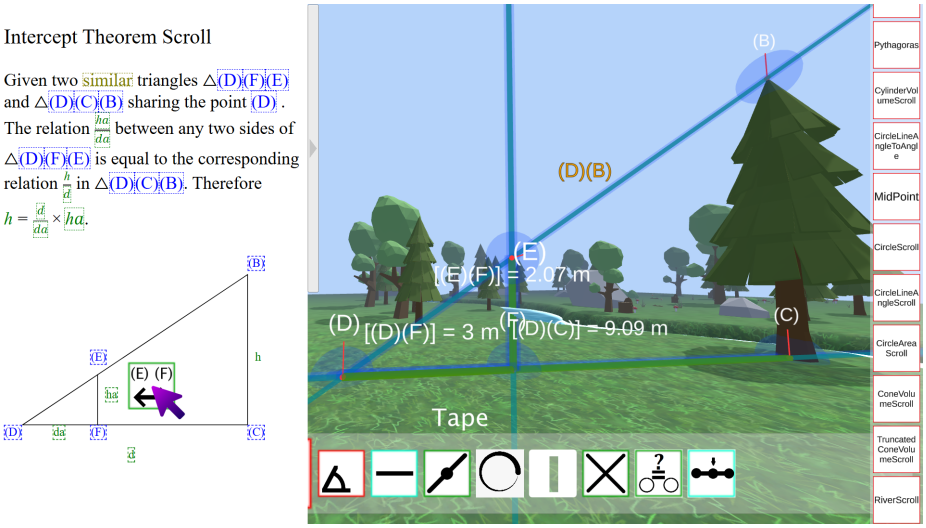

Fig. 2. A Tree Height Problem being solved in FrameWorld

virtual world problem. In our example, this requires first specifying the relevant points, and measuring the distances d, d_a, and h_a.

These are represented as variables of the scroll that must be instantiated. Our implementation allows doing this via drag'n'drop: Figure 2 shows the player in the process of dragging the line segment $[(E)(F)]$ (represented by the square icon below the cursor on the left) from the virtual world onto the variable h_a in the scroll. Once all variables are instantiated, the scroll can be applied.

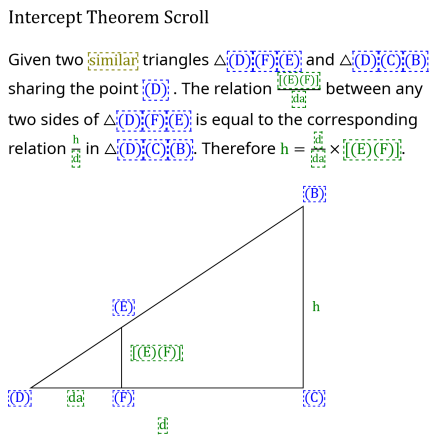

Fig. 3. The scroll after dropping $[(E)(F)]$

Figure 3 shows the resulting display of the scroll after thusly instantiating h_a with the measured length of the line segment $[(E)(F)]$: every occurrence of h_a has been substituted.

This interactive display of formulas was very difficult to achieve, but critical to overcome (**L1**). Previously, Unity could only approximate the display of more visually complex mathematical formulas, requiring ASCII art even for fractions. We now use the WebView plugin [Ess24] for the display of HTML5 (including MathML) in Unity worlds. WebView is essentially a headless Chrome browser that streams images into Unity, and events (like dropping a fact) back out via the Chrome Developer tools Protocol. This solves the

math front-end problems for NuFrameIT in a way that decouples the presentation of the scroll (which is a didactic document that is part (**A4**)) from that of virtual world (**A1**) in the game engine. This has the added benefit that the HTML descriptions and JS interaction scripts could be reused in other game engines that allow for similar webpage display.

3 Backend Perspective

The listing below shows the formalization of the document for the intercept scroll from Figure 2. It declares five points and the lengths of four sides of the resulting triangles. The type `point` is part of the background knowledge. Even though the coordinate triple of a point is fixed and known in the virtual world of Unity, the logical representation can choose flexibly how to represent points: e.g., it can use coordinate triples or a pair of location and height, or leave the type entirely abstract. This corresponds to distinctions between, e.g., synthetic and analytic geometry. Indeed, most applications only ever require measuring *relative* distances, and currently we do not even provide a gadget to measure *absolute* coordinates. For our example, we leave abstract the type `point` and the function `dist` for the distance between two points, because this best represent the players actual knowledge about the game world.

Then individual relative distances effectively become pairs of a variable (whose numeric value may or may not be known in the logical world) and an axiom capturing its definition as the distance between two points. This decouples the representation of the floating point value of, e.g., `d`, which is needed to perform computations, from its geometric meaning. NuFrameIT uses the naming convention of appending `_prop` for the defining axioms, such that the line segment `d` that is visible in the user interface can be connected to its geometry axiom `d_prop` that captures that `d` is the length of the line segment `DC`.

```
theory InterceptTheoremScroll
 // variables
 B, C, D, E, F : point
 similarity: ⊢ similar(Triangle(D,C,B),Triangle(D,F,E))
 d, h, da, ha: float
 d_prop:  ⊢ dist(D,C) == d
 h_prop:  ⊢ dist(B,C) == h
 da_prop: ⊢ dist(D,F) == da
 ha_prop: ⊢ dist(E,F) == ha
 // payload
 intercept: ⊢ h/d == ha/da
```

To instantiate the scroll, all points and three of the four lengths must be instantiated with concrete values, and the proof obligations induced by the corresponding geometry axioms (as well `similarity`) must be discharged by instantiating them with an appropriate proof term. These are usually named properties that were generated when using a gadget or applying a previous scroll. For example, the action in Figure 2 is performed in a situation theory that already

contains the axiom `EF_prop: ⊢ dist(E,F) == 2.07` that was added when the player used the tape measure gadget on E and F. That axiom exposes to the logical world the distance between the two points that was calculated in the virtual world.

The action in Figure 2 of dragging that fact icon onto `ha` extends the situation theory with the instantiation `ha_prop = EF_prop`. Additionally, unification of their types induces the instantiation `ha = 2.07`. Importantly, it is not possible for the player to instantiate `ha` with an arbitrary number or measurement: The interactive instantiation-building only succeeds if the types unify, and that requires that the performed measurement corresponds to the geometry axioms in the scroll.

In a scroll, there is no strict distinction between variables that must be instantiated by the player and fields, whose values are computed by the scroll. The pushout for scroll application is defined for any set of instantiated variables:[2] it can be constructed simply by adding the given instantiations to the situation theory, and then merging in the scroll theory. For didactic reasons, it is practical to attach meta-information about the intended instantiations, like Figure 3 stating which three of the four lengths should be instantiated with concrete values to compute a large vertical distance.

The pushout is only unique up to isomorphism, and it is not easy to choose the most practical representative. The theory resulting from a general pushout construction is usually needlessly complex. While it straightforward to rewrite it into a simper isomorphic one, care must be taken not to employ simplifications that go beyond the player's knowledge or renamings that go against the player's expectations. For this purpose, UniFormal provides a special *theory simplifier* that performs two kinds of operations.

1. The pushout contains the instantiations for the scroll variables, whose right-hand side is usually just a number or an identifier. These are substituted by their definiens everywhere and then dropped from the situation theory entirely.
2. The concrete values of some declarations can be derived using the payload theorem, such as `h` in our case. The simplifier solves for these values, adds them as definitions, and renames them according to the usual naming scheme for new facts. Then it removes the axioms that have become redundant.

In our running example, the scroll application adds uninstantiated declarations for `h` and `intercept`. Step (1) expands all declarations for variable instantiations by expanding them into their definitions everywhere, such as `ha` and `ha_prop`. Then it removes these declarations, which are no longer referenced anywhere. Step (2) solves `intercept` for `h` and then adds the resulting value as the definiens of `h`, which is consequently renamed to `BC` (and `h_prop` is renamed accordingly). At this point, the axiom `intercept` is redundant because all four lengths have numeric definitions and the formula is simply true. It is thus removed. At the end of theory simplification, all names declared in the scroll have

[2] If the instantiations don't allow for a unique solution (or any at all), the pushout is still defined. But rarely useful for our purposes.

disappeared from the situation theory, and the definiens of the variable formerly known as `h` has been added. Not only does this keep the situation theory simple, it also prevents name clashes when performing repeated applications of the same scroll with different instantiations.

4 Outlook

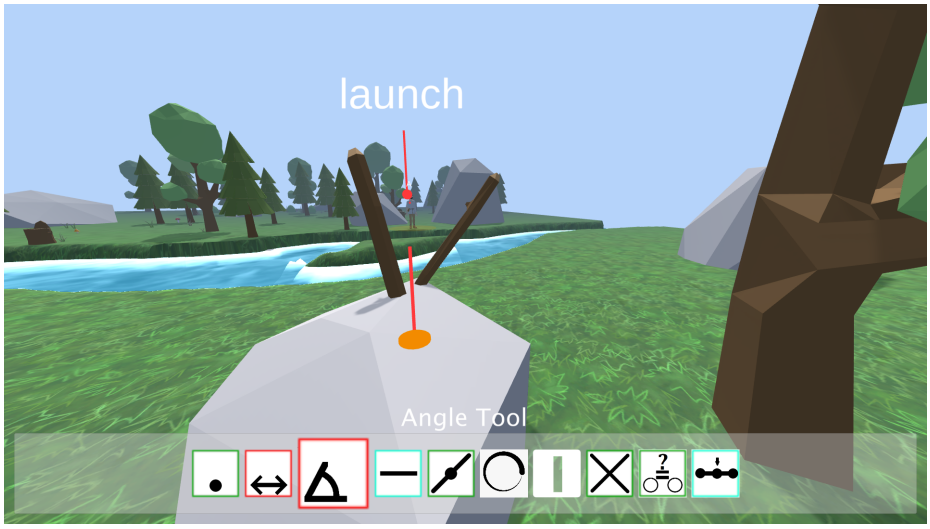

Fig. 4. Initial situation of the river problem

As a more complicated use case, consider moving a small object to a recipient waiting on the over side of an untraversable river. Using a ramp near the river, the player can launch it with a fixed inclination. In the virtual world, this requires a physics engine to approximate the trajectory of the launched object. But given the launch angle and the velocity, calculating the trajectory and the arrival time and point is straightforward in the logical world. The respective scroll looks as follows (using background knowledge declaration `grav` for the constant acceleration due to gravity, and an already simplified quadratic formula for `t_prop`):

```
theory SlingshotScroll
  launch, catch : point
  // launch velocity, and its vertical/horizontal part
  l, v, h: float
  lvh_prop: ⊢ v^2 + h^2 == l^2
  // launch angle
  inclination: float
```

```
inclination_prop: ⊢ v/h == inclination
// catch position and time
d, t: float
d_prop: ⊢ dist(launch,catch) == d
// simplified parabolic vertical movement
t_prop: ⊢ t == 2v/grav
// linear horizontal movement
td_prop: ⊢ h*t == d
```

A narratively more interesting variant of this level is to not compute the point to catch the object from the launch data, but instead to compute the necessary launch data from the distance to the recipient. A problem that is also much harder to solve with iterative simulation approach of a physics engine. The distance could be obtained by applying skills learned in levels similar to the height measurement example above. Importantly, both variants can use the same scroll, just with different instantiations.

5 Conclusion

We have reported on a major re-conceptualization of the FrameIT approach to serious games, fixing the weak points that have been discovered by previous prototypes: We have completely replaced the backend with one based on the new UniFormal language, which is a much better fit for FrameIT. In the front-end we have added a plugin to the game engine that provides responsive rendering of state of the art mathematical notation, and we have build interaction components that allow the player to interactively build the coupling between logical and virtual world.

The new components work well together and provide a much more natural and principled serious game development framework than the original ad-hoc prototype version of FrameIT.

References

[ALeA] *ALeA: Adaptive Learning Assistant*. URL: http://alea.education (visited on 07/16/2024).

[Ber+23] M. Berges, J. Betzendahl, A. Chugh, M. Kohlhase, D. Lohr, and D. Müller. "Learning Support Systems based on Mathematical Knowledge Managment". In: *Intelligent Computer Mathematics (CICM) 2023*. Ed. by C. Dubois and M. Kerber. Vol. 14101. LNAI. Springer, 2023. DOI: 978-3-031-42753-4. URL: https://url.mathhub.info/CICM23ALEA.

[CICM20] M. Kohlhase et al. "FrameIT: Detangling Knowledge Management from Game Design in Serious Games". In: *Intelligent Computer Mathematics (CICM) 2020*. Ed. by C. Benzmüller and B. Miller. Vol. 12236. LNAI. Springer, 2020, pp. 173–189. DOI: 10.1007/978-3-030-53518-6_11. URL: https://kwarc.info/kohlhase/papers/cicm20-frameit.pdf.

[CICM25] F. Rabe. "Global, Regional, and Local Contexts". In: *Intelligent Computer Mathematics (CICM) 2025*. Ed. by V. de Paiva and P. Koepke. Vol. 16136. LNAI. Springer, 2025. URL: https://github.com/UniFormal/UPL/blob/main/doc/levels.pdf.

[Ess24] B. Esswein. "Interactive MathML Formulas in UFrameIT". B.Sc. Thesis. FAU Erlangen-Nürnberg, Dec. 2024. URL: https://gl.kwarc.info/supervision/BSc-archive/blob/master/2024/EsseinBjoern.pdf.

[JS92] R. Jenks and R. Sutor. *AXIOM: The Scientific Computation System*. Springer, 1992.

[Rab13] F. Rabe. "The MMT API: A Generic MKM System". In: *Intelligent Computer Mathematics*. Ed. by J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger. Springer, 2013, pp. 339–343.

[RK13] F. Rabe and M. Kohlhase. "A Scalable Module System". In: *Information and Computation* 230.1 (2013), pp. 1–54.

[UFO] *The FrameIT Project*. Project Website. URL: https://uframeit.org/ (visited on 03/19/2026).

[UNR] *The most powerful real-time 3D creation tool – Unreal Engine*. URL: https://www.unrealengine.com (visited on 03/19/2026).

[UNTY] *Unity Engine: 2D & 3D Development Platform | Unity*. URL: https://unity.com/products/unity-engine (visited on 03/19/2026).