A High-level Event System for Augmented Reality

Jean-Luc Lugrin[*]
School of Computing
University of Teesside

Remi Chaignon[*]
School of Computing
University of Teesside

Marc Cavazza[*]
School of Computing
University of Teesside

Abstract

3D graphics systems increasingly rely on sophisticated event systems derived from collision detection mechanisms, which support the discretisation of Physics as well as high-level programming and scripting. By contrast, Augmented Reality systems have not yet adopted this approach. We describe the development of a high-level event system on top of the ARToolkit environment incorporating the ODE Physics engine. We first define a typology of events encompassing interactions between virtual objects as well as interactions involving markers. We then describe how these events can be recognised in real-time from elementary collisions detected by the ODE Physics engine. We conclude by discussing examples of high-level event recognitions and how they can support the development of applications.

CR Categories and Subject Descriptors: H.5.1 [Multimedia Information Systems]: Artificial, augmented, and virtual realities;

Additional Keywords: Augmented Reality, Event Systems, ARToolKit.

1 Introduction

Simulating realistic physical object behaviour is an important aspect of all Virtual Reality systems, even more so when object motion is triggered, directly or indirectly, by user interaction. However, the real-time integration of all physical equations is beyond the reach of current systems and would compromise real-time rendering. The solution adopted by many 3D engines, in particular computer game engines, consists of discretising physical simulation so that only kinematics will be constantly solved. Discretisation takes place at the level of specific events, such as collision between objects, or between objects and volumes. This approach has further advantages in that it offers the possibility of programming system behaviour at the level of object interactions, sometimes in a specialised fashion which depends on the application semantics: in computer games, typical events described are landing, destroy, encroach, but many of them are reusable for other applications. This is probably why more traditional VR systems have also developed event systems [1]. Augmented Reality systems could naturally benefit from a similar approach. Traditionally the complexity of virtual scenes in AR systems has been much lower than in their VR counterparts. However, there has been recent interest in realistic physical simulation, particularly with the integration of advanced Physics engines into AR systems [3] .The concept of an event system is not widespread in AR. One of its first mentions was as part of the DWARF system [5] where one of the examples of "glue logic" was precisely to interpret positional changes in terms of high-level events (such as "entering a room") [6]. However, no systematic event system (of the type of those encountered in computer games) had yet been described as part of these applications.

In this paper, we introduce such an event-system, which is inspired from our previous work in high-level event systems for VR [2]. In the next sections, we show how this detection can be incorporated in the rendering cycle of the ARToolkit without compromising its performance and illustrate the potential uses of the system on implemented examples.

2 System Overview

We use the ARToolkit system [4] as a development environment, in a configuration that includes the OSGART API (ARToolKit for OpenSceneGraph [7]) and the ODE Physics Engine (Open Dynamic Engine [3]). Our Event System, ESAR (Event System for Augmented Reality), which is the main focus of this paper, is an additional software layer developed in C++. ESAR proposes an event programming paradigm (inspired from those used in modern game engines) that facilitates the description of virtual objects behaviour. Following a traditional approach in 3D engines, ESAR continuously intercepts basic object collisions and position changes and analyses them in terms of higher-level events, which we have termed *interaction events*.

Figure 1 represents the integration of ESAR into the rendering and simulation cycle of the ARToolKit. The first sampling phase is performed after the ARToolKit update cycle (providing the marker position for the next frame), whereas the second sampling phase takes place after the collision detection cycle of the Physics Engine. During the first phase, ESAR interprets new marker orientations in terms of *position events*, such as the `Tilted` event**.** The second phase parses the collision contacts generated by the Physics engine during the object's motion into high-level *collision events*. The system uses physical parameters (velocity, momentum) and object categories to add semantics to low-level collisions. The system operates by continuously intercepting primitive events (such as Collision and Position update), and

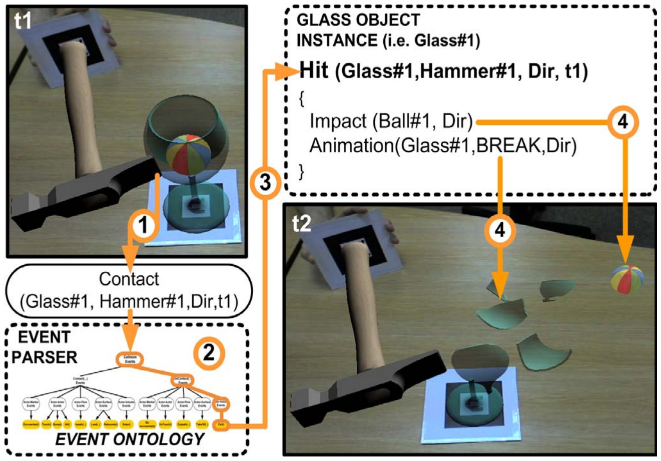

Figure 2. Example of Event-Based Behaviour: Breaking a glass with a virtual hammer.

*University of Teesside, Middlesbrough, TS13BA, UK.
{j-l.lugrin, r.chaignon ,m.o.cavazza}@tees.ac.uk

parsing them into higher-level events. Event interpretation is based on the type of the objects involved in the event. For instance, the `Hit` event is generated from contacts between actors, while the `Inpath` event is activated from a contact between an actor and a flow. ESAR relies on a simple object ontology based on five main categories: `ACTOR`, `MARKER`, `FLOW`, `VOLUME`, and `SURFACE`. Event parsing also uses logical filters based on predicates involving object properties, such as `Velocity (ob#1, LOW)`. Upon successful instantiation of a filter, an event instance is created and the appropriate objects notified. The response will depend on the event-function implementation at the object class level that defines its behaviour. Our complete *interaction event* ontology consists of seventeen events, this set can be used to programme object behaviour as discussed in the following section, in a similar way as implemented in modern game engines.

3 EXAMPLE RESULTS

The first example illustrates how ESAR can be used to encode the manipulation of objects based on interaction with markers, for instance transferring an object from one marker to another interactively. The user inclines a marker holding a sphere (Fig.1, t1). The `Tilt` event is recognized, in response, the control of the sphere behaviour is transferred to the ODE Physics engine. Consequently, gravity makes it falls off of the marker and collides with the marker situated below it (Fig.1, t3). From the collision an `Encroached` event is recognised and, as a further response, this sets the sphere's physical mode as "supported" rather than "free". The user is now able to manipulate the sphere with the other marker. The second example illustrates how events play a role in the articulation of the different elements of the objects behaviour (animation, change state, motion simulation). Here, the user hits a glass using a hammer; a collision is detected by the Physics engine between the two virtual objects, and intercepted by our event system (Fig.2, 1). The collision is then parsed against our ontology of collision events (Fig.2, 2); the high momentum of the hammer results in the recognition of a Hit event. The instance of the Glass object is then immediately notified of the Hit event (Fig.2, 3). Inside this class, a Hit event on a glass instance is interpreted as the shattering of the glass. This in turn triggers an animation and propagates the impact to the object contained within (Fig.2, 4).

4 CONCLUSION AND FUTURE WORK

We have implemented a prototype version of ESAR which is fully functional in our test environment. For scenes of moderate complexity (up to 10 interacting objects, which corresponds to a large proportion of AR applications), the integration of ESAR does not significantly affect the overall performance, operating at an average rendering rate of 58 fps. Current limitations include some unstable behaviour produced by the ODE Physics engine, which will lead us to also explore alternative Physics engines.

ACKNOWLEDGEMENTS

Hartmut Seichter is thanked for providing us with source code exemplifying the integration of the ODE engine. This work has been funded in part by the European Commission through the CALLAS Project (FP6-2005-IST-5).

REFERENCES

[1] Hua Jiang, G. Drew Kessler ,Jean Nonnemaker, "DEMIS: a dynamic event model for interactive systems", Proceedings of the ACM symposium on Virtual reality software and technology, 2002.
[2] J-L. Lugrin, M. Cavazza, "Making Sense of Virtual Environments: Action Representation, Grounding and Common Sense", Proceedings of the International Conference on Intelligent User Interfaces, Honolulu, Hawaii, USA, pp. 225-234, January 2007.
[3] R. Smith, "Open Dynamic Engine", www.ode.org, 2005.
[4] H. Kato, m. Billinghurst, "Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System", Proceedings of the Second IEEE and ACM International Workshop on Augmented Reality, San Francisco, pp. 85-95, 1999.
[5] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reicher, S. Riß, C. Sandor, and M. Wagner, "Design of a Component-Based Augmented Reality Framework", Proceedings of the Second IEEE and ACM International Symposium on Augmented Reality, New York, USA, pp. 124–133, October 2001.
[6] B. Bruegge, A. MacWilliams, and T. Reicher, "Software architectures for augmented reality systems – report to the ARVIKA consortium", Technical Report TUM-I0410, July 2004.
[7] J. Looser, R. Grasset, H. Seichter, M. Billinghurst, "OSGART - A Pragmatic Approach to MR", In Industrial Workshop at ISMAR 2006, Santa Barbara, California, USA, October 2006.

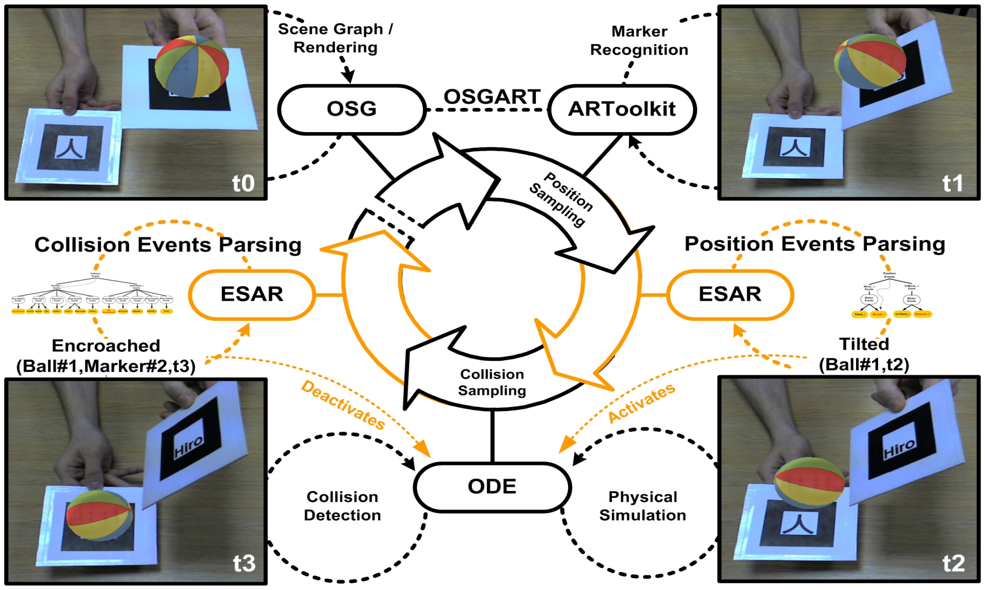

Figure 1. Integration of ESAR in the ARToolkit interaction and rendering cycle: Events can be generated both from interaction with markers and ODE-based physical simulation of object motion.