DANCE: Dynamic Animation and Control Environment

Victor Ng-Thow-Hing and Petros Faloutsos
Department of Computer Science, University of Toronto
{victorng|pfal}@dgp.toronto.edu

Introduction

Research in physics-based animation (PBA) and control is currently producing isolated results. Research efforts result in controllers that are bound to customized software supporting predetermined types of object and controllers, thus limiting opportunities to share or exchange results. We believe that an environment that remedies these limitations is essential to the progress of physics-based research. This inspired

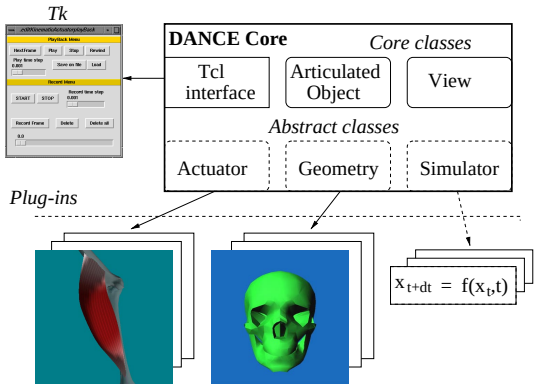

Figure 1: The main components of DANCE.

a joint development project by the authors to build a common environment where multiple controllers and object models could co-exist and interact with each other. The long-term goal is to reduce the startup overhead of future research projects in this area so that investigators can proceed directly to controller design and object model construction. The resulting system, named DANCE, allows practitioners to create their own controllers and physical models as external plug-ins that can be integrated into a common simulation environment. A secondary goal is to give the user the ability to edit controller and actuator parameters interactively through a direct manipulation interface.

Design

Our system is based on an object oriented, plug-in architecture and it is portable across most popular architectures. The main features of DANCE are:

- *Object-oriented and modular design* Can easily be customized and ported.
- *Plug-in open architecture* Controllers in the form of plug-ins can be as independent and complex as desired.
- *Physics-based simulation* Based but not bound to a commercial package.
- *Scripting and window interface* Fully customizable and expandable based on Tcl/Tk.
- *Direct manipulation* Direct manipulation of plug-ins and core objects.
- *Portability* Runs on SGI Irix, Linux, Windows NT, MacOS.
- *Standard keyframing functionality*

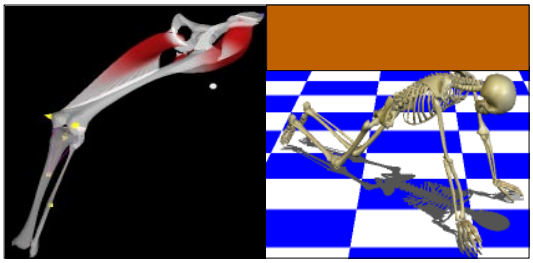

Figure 2: Two actuators: Forward falling and muscles on bones.

We have separated our system into two parts, the *core* and the *plug-ins* as shown in Figure 1. The core classes are built into the main system and implement

the common, yet complex functionality that any animation system needs. We have carefully chosen the core part elements to be those that, most probably, users do not need to modify. The main core classes are the *ArticulatedObject class*, the *View core class* which implements viewing and direct manipulation and the *Dance core class* which manages all event queues, objects and plug-ins.

Plug-ins are represented in the system as abstract classes with the specialized subclass implementation residing in external, dynamically linked executables. These classes impose very few restrictions on the design of specific plug-ins. In addition, they provide the necessary interface for plug-ins to interact with each other, allowing a wide variety of controllers and models to co-exist in the same environment. The main plug-in classes are as follows. *Simulator plug-in class*: The engine that drives the animation is the simulator. We use the commercially-available SD/FAST engine which produces C-code for the equations of motion of a particular AF. However, users can implement and use any simulator they prefer. *Actuator plug-in class*: We defined actuators to be any entity that can exert forces on the articulated figure. They can be a simple gravitation field or a high-level pose controller that can coordinate forces for multiple joints on multiple articulated figures. Given the diversity of control techniques it is preferable not to build a specific type of controller into the main system as users will most likely want to build their own using the plug-in mechanism. *Geometry plug-in class*: Subclasses of this class implement the geometric representation of links in an articulated figure. We currently have implemented an indexed face set plug-in class that allows a large set of VRML polygonal models to be imported into DANCE. This subclass supports oriented bounding boxes (OBB) [1] for collision detection and can automatically compute the mass and a diagonal inertia tensor for polygonal models.

Implementation Details

To achieve platform independence, we chose toolkits that were available on multiple platforms. OpenGL is used for 3-D graphics rendering, GLUT is used for window management and event handling and Tcl/Tk comprises the graphical user interface, Figure 1. We purposely did not integrate the GUI with the system to allow users to customize and develop their own interfaces targeted towards their specialized applications.

Applications

The power of DANCE comes from its ability to support a wide variety of diverse plug-ins. We present two different applications that use DANCE as a common environment.

Anatomically-based modeling. We use DANCE to build articulated structures corresponding to musculoskeletal systems in the human body, Figure 2 left. Digitized bone data can be loaded and their mass properties estimated. We have built an actuator class that uses B-spline solids as a deformable object and that can exert forces on the bones. This actuator is used to model musculotendons and ligaments. Through DANCE's open interface, muscles can be sketched, placed and manipulated directly.

Composeable controllers We use DANCE to implement a framework in which controllers that produce simple everyday motions for human figures can be composed together in a smart and effective way. Currently we focus on producing a composite controller that deals with different cases of loss of balance of human figures. This controller should be able to detect unbalance, and choose between a number of protective moves or natural falling motions depending on what is natural in the specific case, Figure 2 right.

Conclusion

We have built an open, portable and extensible physics-based animation system. Practitioners of PBA can begin to develop controllers without needing to implement all the other subsystems that are necessary for such a system. Furthermore, they can exchange models and controllers and can develop control strategies that allow articulated figures to cooperate or compete with each other. The ability to interact with controllers, articulated figures and actuators with direct manipulation provides quick visual feedback for bringing a human user directly into the process of directing physically-based animation. The challenge remains of devising novel manipulators for controllers that intuitively map to desired motions.

References

[1] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 171–180, 1996.