

Real-time realistic animation in the Crystal Space engine

Christian Van Brussel, Benoit Macq

Communications and Remote Sensing

Laboratory

Université catholique de Louvain

Louvain-la-Neuve, Belgium

{christian.vanbrussel,
benoit.macq}@uclouvain.be

Jorrit Tyberghein, Marten Svanfeldt, Mike

Gist

Crystal Space team

<http://www.crystalspace3d.org>

ABSTRACT

In this paper, we present the recent developments that were made in the Crystal Space game engine (CS – [1]) for the realistic animation of virtual environments. We present the component dedicated to the physical simulation of virtual worlds and its main concepts. We also present the new skeletal animated mesh of CS, its animation blending tree system and the available animation nodes. We show that the animation system of CS is one of the most powerful solutions available in the open-source community, that it can be used to generate realistic animations and behaviors, and can also be used in other domains such as robot simulation or the visualization of motion captured data.

Keywords

Game engine, skeletal animation, physical simulation, virtual reality, video games

INTRODUCTION

Many modern applications use computer graphics to display virtual objects, either for special effects of movies or in video games and virtual reality environments. Many technologies were developed in order to be able to render realistic or even photo-realistic virtual environments and modern personal computers are equipped with graphic cards enabling the ability to render such 3D scenes in real-time. The physics of light propagation are now rather well known and can be approximated efficiently with good overall visual quality. In modern movies such as 'Avatar' it becomes really hard to distinguish the real objects from the synthetic ones.

However, having a good looking image is not sufficient in animated sequences and there is one domain where many productions failed to reproduce coherent behaviors, namely the animation. The human eye is very sensitive to the coherence of the motion of objects that are suggested by the succession of images. In applications such as video games and virtual reality, it is therefore important for the

immersion of the spectator in the virtual scene to have such realistic motions.

For example, the human body has a highly complex skeleton which allows it to generate complex motions, many of which are even used during communication and convey meaning. Even for simple actions such as walking, almost every human will act differently depending on parameters such as gender, size, weight, and mental and physical state (see [2]). Another simple action such as grabbing an object illustrates another difficulty of the animation, which is the relative high degree of 'intelligence' needed to achieve such task. To grab an object it is necessary to move the body to a position where the object is reachable, we may need to kneel down or to stretch the arm for that, we may need to use one or two hands depending on the size and the weight of the object, and we may need to do complex movements to avoid collisions with other objects in the environment. Lastly, the human skin bends in respect to the actions that are made on the skeleton and on the body, and the human face has many muscles allowing the creation of many different facial expressions.

In a virtual environment it is also important either for the interaction of the user with the scene or for the realism of the scene, to have the objects of the virtual world behaving like real objects do, e.g. by moving and falling when we touch them or for deformation or breakage to occur when they collide. Modern commercial games commonly use heavily advanced physical effects such as physical user interaction, vehicles, explosions, fluids or destructible environments. Modern movies also use physical simulation in order to simulate e.g. the destruction of a city.

The Crystal Space game engine is a complete framework for the development of real-time 3D applications such as video games and virtual reality. Recent developments have been made in this game engine to address the task of real-time realistic animation of the objects. In the rest of this paper we will present the results of these developments.

BACKGROUND AND RELATED WORK

The simplest technique for the animation of an object is the key-frame animation, i.e. a sequence of images defined by an artist either in 2D or in 3D, which are displayed successively in order to suggest the motion. This technique is a good general solution for the animation of many simple

*LEAVE BLANK THE LAST 2.5 cm (1") OF THE LEFT
COLUMN ON THE FIRST PAGE FOR THE
COPYRIGHT NOTICE.*

objects, but in the highly dynamic environments of modern virtual worlds it lacks the aptitude to adapt to new situations and events. The highly dynamical aspect of our virtual worlds makes it impossible in practice to prepare in advance an animation for all virtual objects and for all combinations of situations and events.

Procedural animation techniques have therefore been introduced in order to dynamically generate new animations on the base of the actions and events in the virtual world. A first step was the introduction of the real-time physical simulation of the dynamics of the virtual environments, and some middleware libraries are now dedicated to this simulation task. The most popular are NVIDIA PhysX ([3]), Havok Physics ([4]), Bullet ([5]) and the Open Dynamics Engine (ODE - [6]), with ODE and Bullet being the two most widespread open-source libraries. For a long time ODE has been a reference, being used in many games and simulation environments. But these last years it has been overtaken by Bullet which is now far more widely used and has more features such as soft bodies and kinematic objects. The Bullet team is also working on a CUDA and OpenCL implementations of the library, which is an important enhancement needed to take advantage of the general purpose hardware capabilities of today's graphics processors.

For the animation of more complex entities such as human beings, there are some commercial middleware solutions such as Euphoria ([7]) and Havok Behavior ([8]), which are specialized in the domain of the generation of realistic animations. These two libraries are closely linked to both the physics and the artificial intelligence libraries, in order to interact with the dynamic and behavioral simulations. Most of the modern commercial game engine SDK's such as Valve Software's Source Engine ([9]), Epic's Unreal Engine ([10]) and Crytek's CryEngine ([11]), also provide the infrastructure for advanced skeletal animation, with features such as "ragdoll effect", inverse kinematics, parametric skeletal animation, procedural motion warping, facial animation and lip synchronization.

On the open-source side of the middleware and game/3D rendering engines, there was until now almost nothing other than solutions for basic skeletal animation, with libraries such as the Cal3D library [12], allowing only for the basic playing and combination of user-defined animations, as well as for the morphing between several targets (e.g. for facial animation).

Lastly, in order to create animated objects there are some design tools such as AutoDesk MotionBuilder and 3DS Character Studio which are specialized in the creation of skeletal animated meshes and in the offline generation of new animations, by hand or by the processing of motion captured data. There are also other tools such as the open-source project MakeHuman ([13]) which can generate a complete humanoid model from few anatomic and ethnical parameters.

PHYSICAL SIMULATION

Concepts of physical simulation

The physical properties that are used to represent the objects of the virtual environment are mainly their collision shape, their mass and inertia, their position, velocity and acceleration, their friction, elasticity, and softness. The user defines the virtual objects of the environment and the physical library will compute if and how these objects are colliding. It will then apply some forces on the objects according to the collisions, the gravity and other external forces, and will compute the motion of the objects. Some parameters and functionalities allow the user to tweak the behavior of the simulation in order to find a compromise between accuracy and the performance costs.

The physical objects can be in different states: static - they never move (mainly for the environment), dynamic - objects move and collide, and kinematic - the motion of the objects is controlled by the application, but they collide and interact with the dynamic objects. Kinematic objects are important for the physical user interaction as it allows having e.g. the user controlling an avatar that pushes and knocks over the objects it touches.

The collision shapes of the objects are defined with a combination of primitives such as spheres, boxes, cylinders, capsules, cones, convex and concave shapes. The collisions with spheres and capsules are the least costly to compute, while the concave shapes are the worst.

The physical objects can be attached through joints to create more complex structures. The properties of the joints can be set up, such as the rotational and translational degrees of freedom, the minimal and maximal values, and the springiness of the articulation.

Modern physical libraries can also simulate more complex behaviors such as soft bodies, fluids, and destructible objects.

Physical simulation in CS

CS has support for physics (limited to the dynamic of rigid bodies) for both Bullet and ODE through their respective plugins. The CS API abstracts the used physical library and provides higher-level functionalities for defining and manipulating the physical objects and the simulation. The physical library to be used can be selected at runtime, which provides the option of switching them according to need. The API provides automatic definition, configuration and updating of the underlying objects of the library, and automatically synchronizes the motion of the visual 3D meshes with their physical model. It also supports kinematic objects and can automatically update the motion of the bodies in accordance with the actions of the user. Events are generated when collisions occur in order to be able to e.g. play adapted sounds.

The physics plugins of CS allow the user to concentrate on the model and behavior of the virtual environment without having to pay attention to the technical aspects of the

physical simulation. A debug tool allows the visualization of some information such as the collision shapes, the collisions and the joint constraints.

On top of CS, The Crystal Entity Layer (CEL, abstraction layer above CS) adds support for more advanced physical effects such as fluids, explosions, wheeled vehicles, attractors, repulsors and gradual destruction of objects.

The physical properties of the objects can be defined in Blender and exported automatically into CS via its XML compliant description language.

SKELETAL ANIMATION

Presentation

Skeletal animation concerns the art of animating skeleton-based objects, i.e. objects composed of a tree of articulated bones. CS and CEL have traditionally supported basic skeletal animation through the general CS mesh and the Cal3D library. These animation systems are limited in their capabilities and do not meet the needs of modern games which require far more realistic behaviors. A new specialized mesh and animation system were written in CS in order to achieve the degree of quality needed today. The base of the new mesh and animation system was written during the Apricot open-game project [14] set up in collaboration with the Blender Institute, and much other functionality have been added for the need of the 3D-Media academic research project [15].

It is also to be noted that, although these animation techniques are particularly suited for the animation of entities such as humanoids, they can also be used in other contexts, e.g. for vehicles or animated objects such as simple doors or complex castle gates.

The animated mesh

A skeletal animated mesh is defined by the following properties:

- the envelope: the vertices, faces, UV mapping and materials of the mesh.
- the skeleton: the tree of articulated bones.
- the bone weights: the influence of the bones on the deformation of the vertices of the envelope.
- the morph targets: target variations of the position of the vertices of the envelope, used for morphing.
- the bounding boxes of the bones: for efficient tests on collisions and visibility.
- the physical description of the bones and joints: collision shapes, masses, joint constraints and other properties.
- the sockets: emplacements to link some external objects like a hat, some clothes, or an object in the hands.

For each frame the CS animation system updates the pose of the skeleton and deforms the envelope of the mesh according to the bone weights and the active morph targets

(skinning process). It can also add decals on top of the mesh to introduce some variations of the mesh and manipulate the rendering parameters of sub-sections of the mesh.

Animated meshes can be defined in Blender before being exported into CS. There is currently no support for other 3D design tools such as 3D Studio Max or Maya.

Animation blending trees

The core of the animation system uses the technology of animation blending trees, a technology used in many leading commercial 3D engines such as the Unreal Engine and the Source Engine. A blending tree is a structure used for the advanced generation and combination of animations. The combination is made by blending the position and orientation of each bone, hence the name of the method. Each leaf of the tree is either a raw animation sequence (user-defined by an artist or acquired by technologies such as motion capture), or a procedural node, i.e. a node dynamically generating an animation. The other nodes in the tree are used for the combination of sub-trees in order to achieve complex animations and behaviors.

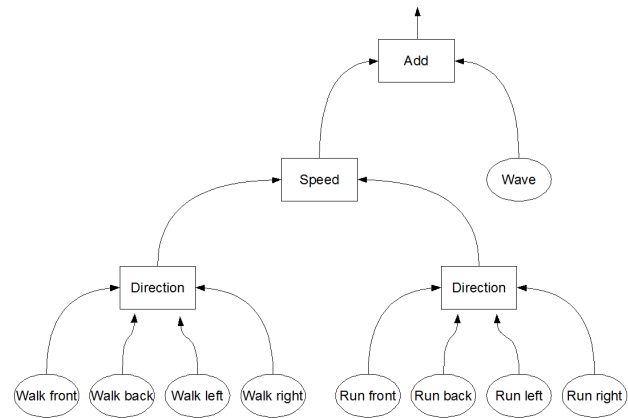


Figure 1: Example of animation blending tree for a walking human

Figure 1 illustrates the method for a walking humanoid. In this example the bottom most leaf nodes of the tree are raw animations for respectively walking and running in front, back, left and right directions. All the walking animations are combined by a 'direction' node, generating an animation of the humanoid walking in any custom direction, same for the running animations. The two 'direction' nodes are then combined by a parent 'speed' node, which generates an animation of the humanoid moving in any custom direction at any custom speed. Lastly, the moving animation is combined through an 'add' node with an animation of the humanoid waving his arm.

This method enables the easy generation of complex behaviors by manipulating the structure of the blending tree or the parameters of the animation nodes. The tree structure also makes it possible to easily replace sub-trees in order to achieve effects such as switching sub-parts of the mesh (e.g.

to change the clothes of the avatar) or for Level Of Detail management.

Animation nodes

A set of animation nodes are available in CS:

- Raw-animation: user-defined animation sequences.
- Multi-blending: simple blending of any number of sub-trees.
- Random: random playing of sub-trees, allowing for more varied animations.
- Priority: blending of sub-trees according to their priority, useful e.g. to add a secondary motion on top of a base animation while preserving most of the base animation.
- Finite State Machine (FSM): the sub-tree to be played is selected according to the state of the FSM. Animations can also be defined for the transitions between the states, in order to keep a smooth motion.
- LookAt: automatic alignment of some bones with a user-defined target. Useful e.g. to breathe life into an avatar by generating an active interaction from the avatar.
- Speed: combination of animations at given speed to achieve any custom speed.
- Ragdoll: interaction with the physical simulation. The bones of the skeleton are automatically converted to rigid bodies colliding and interacting with the physical environment. The bones can be set in kinematic state in order for the mesh to interact with the environment while being animated by the classical animation method, or in dynamic state e.g. to have an avatar falling realistically in a stairwell. Both states can also be combined to achieve effects such as the main body of the avatar classically animated, and sub-trees such as hair, clothes or accessories animated by the physical simulation.
- Inverse Kinematic: Cyclic Coordinate Descent algorithm. Generates an animation that places a bone end effector at a specific position and orientation. Allows e.g. an avatar to grab an object regardless of the relative position of the avatar and the object.
- Debug: interactive visualization of the dynamic of the output of any sub-tree. Enables easy development of a blending tree or of new blending nodes, or even the ability to visualize data e.g. when doing research on the processing of motion capture data.

All these nodes may also generate events, e.g. when the foot touches the ground, in order to play sounds for instance.

CONCLUSION AND FUTURE WORKS

In this paper we have presented the functionalities that were implemented in the Crystal Space engine for the realistic

animation of virtual environments, with both the physics plugins and the new skeletal animation system.

These animation systems in CS provide one of the most advanced open-source solutions for the modern animation of virtual environments, or even for the simulation of robots or the visualization of motion captured data. It has not yet reached the level of functionality achieved in some modern commercial products, but it is far more powerful than most of the other solutions proposed by the other open-source game engines and libraries.

Although already fully usable, the animation system has still many other functionalities that can be added, such as the physical simulation of soft bodies, deformable objects and destructive environments. The skeletal animated mesh needs hardware skinning in order to animate it more efficiently and many more animation nodes can be added for the general movement of the meshes on uneven terrain, the composition and the retargetting of the animations and the automatic generation of transitions between animations. Some work might also be made toward facial animation and lip synchronization, and the support for the new animated mesh should be added to the Crystal Entity Layer in order to link the peripheral events with the control of the meshes, and to establish interaction with the artificial intelligence functionalities of CEL (e.g. path finding and behavior trees).

ACKNOWLEDGMENTS

Part of this work was funded by the 3D-Media sub-project of the MediaTIC portfolio, co-funded by the European Regional Development Fund and the Walloon Region.

REFERENCES

1. <http://www.crystalspace3d.org>
2. J. TILMANNE, R. SEBBE, T. DUTOIT, 2008, "A Database for Stylistic Human Gait Modeling and Synthesis", *Proc. eINTERFACE08*, Paris, pp. 91-94.
3. http://www.nvidia.com/object/physx_new.html
4. <http://www.havok.com/index.php?page=havok-physics>
5. <http://bulletphysics.org>
6. <http://www.ode.org/>
7. <http://www.naturalmotion.com/euphoria.htm>
8. <http://www.havok.com/index.php?page=havok-behavior>
9. <http://source.valvesoftware.com>
10. <http://www.udk.com>
11. <http://mycryengine.com>
12. <http://home.gna.org/cal3d/>
13. <http://www.makehuman.org/>
14. <http://www.yofrankie.org/>
15. <http://mediatic.multitel.be/platforms/3dmedia.html>