# ROBOT DESIGN USING UNITY FOR COMPUTER GAMES AND ROBOTIC SIMULATIONS

William A. Mattingly, Dar-jen Chang, Richard Paris, Neil Smith, John Blevins, and Ming Ouyang
Department of Computer Engineering and Computer Science
University of Louisville
Louisville, Kentucky
E-mail: wamatt02@louisville.edu

*Abstract*— **We present a new approach to robot design, using the Unity development environment as the primary authoring tool for a functional virtual robot. We also discuss the development of a collection of design assets in the form of Unity packages which we have found to greatly improve the workflow and accessibility of robot design for computer games and robotic simulations.**

***Keywords- Computer Games, Hierarchical Skeleton Structures, Locomotion Animation, Unity Game Engines, Robotic Simulation.***

## I. INTRODUCTION

In computer games, various kinds of virtual characters are used to make gaming experiences fun and engaging. While the definition of a robot is often varying, they are generally mechanical agents which carry out tasks automatically using some type of programming. Because of their vital role in modern society, and wide use in science fiction, robots are commonly used to fill the role of virtual characters in many different types of games. Robotics generally refers to the design, construction, operation, and applications of modern robots. In the field of robotics, robots are electromechanical systems, which have physical presence in the real world, whereas robots in games are a virtual depiction of a character having robot-like characteristics. Nevertheless, robots in robotics and games share some important design issues related to robot shape, appearance, locomotion, and control programming.

This paper proposes a 3D robot design system based on Unity, a popular game development environment. There are many open source and proprietary robotic simulation software packages offering different features, varying degrees of programmability, and specific application targets [1]. In view of those existing robotic simulation systems, the proposed Unity robot design system provides a rich alternative environment for designing robots for both computer games and robotic simulations. Commercial robotic simulation applications such as V-REF [2] and Webots [3] tend to include an integrated Graphical User Interface (GUI) for robot prototyping and dynamic simulation of robot-environment interaction. To support these functions, they require a basic infrastructure for environment editing, 3D rendering, collision detection, rigid body dynamics, animation, and scripting. Unity includes all of the functions mentioned above in an extremely flexible and easy-to-use editing system. Thus, using Unity for robot design is natural. The proposed robot design system will support the following tasks: Robot assembly, Robot locomotion or animation clip design, and robot control and simulation.

## II. UNITY OVERVIEW

Unity [4] is an integrated development environment used to create computer games and 3D visualization applications. Unity applications can be deployed to the web, game consoles (e.g. Xbox, Wii, and PS 3), mobile devices (e.g. iPhone, iPad, and Android devices), or personal computers as stand-alone applications for Windows and Mac.

### A. Scenes

Unity's basic design concept is called a scene. Usually a scene is a game level or menu. Through scripting, a new scene can be loaded from the current scene based on user input or game state changes. As such, a complicated game level structure and menu system can be constructed and debugged in a modular and logical manner. A scene consists of a collection of game objects which, in turn, contain various other types of components. The type of components included in a game object will determine the object's function and behavior. For example, a robot arm may contain mesh filter and render components for visualization and a collider component for collision detection. Related game objects can be combined in a hierarchical structure to form a single game object. For example, a robot can be built from arms, legs, and torso with an appropriate hierarchy. Object control and interaction between objects is achieved through scripting.

### B. Unity Interface

Unity editing windows include Scene, Game, Hierarchy, Project, and Inspector. The Scene window has a very commonly used GUI for the user to navigate the scene and modify spatial properties of game objects in the scene. New game objects can be added through menu or drag-and-drop interface. Similar interface is applied to adding and modifying game components with the help of Inspector window. The Hierarchy window shows all the game objects present in the scene and the hierarchy among game objects. A very useful

debugging and game testing feature allows selected game objects to be disabled in the Inspector window. Finally, the Game window provides game play testing with the "Maximize on Play" setting. When Maximize on Play is off, all other windows are visible and reflecting progress of the game play. For example, the designer can change the Scene window to see the game progress from different perspectives or how a particular game object behaves even when the game object is not visible in the Game play window. Dynamic game objects, which are created and destroyed during the Game play, can be seen in the Hierarchy window.

### C. *Scripting in Unity*

Unity exposes to its users an object-oriented framework to support runtime and design-time scripting in three languages: Boo (by Unity), JavaScript, and C# (by Mono open-source .NET). Specifically, there are two categories of object classes: Runtime and Editor. Runtime classes are used to write scripts and attach them to game objects as needed. In other words, a script is a type of game component, which can be attached to a game object to control its behavior at game play time. A script can be attached to several different game objects and multiple scripts can be attached to the same game object. This distributed control of individual game objects through attached scripts makes the implementation of game design very intuitive to construct and maintain. On the other hand, the user can use Editor classes to add menu items to the Unity menu system, enhance Inspector interface, and so on, at design time. A useful application of this involves writing a plug-in to extend Unity, such as to add a new type of game object to the Unity editor.

## III. ROBOT ASSEMBLY

Robot assembly focuses on designing the skeletal structure and shape of robots using Unity. We refer to this as robot assembly to emphasize the similarity of assembling robots from pre-built parts. Some robotic simulation software packages refer to this task as robot prototyping.

### A. *Hierarchical Structure*

In the context of games, robots are just rigid bodies composed of articulated joints and possessing a specific hierarchy. Skinned mesh characters have a similar hierarchical structure but usually have a single mesh with some deformation scheme. Unity supports a skinned mesh rendering component using a vertex blending deformation scheme. For robot assembly using Unity, there are two distinct phases: building a hierarchical structure of joints and attaching rigid bodies to joints. Design of hierarchical structure of joints can be done using Unity scene editor or input from an external source. Manual design is usually done incrementally with visual feedback displayed in the Unity Scene window. Alternatively the designer can complement design with analytic techniques and document the design in some form as shown in Fig. 1. To each joint the designer attaches one or more rigid bodies, which may include mesh and material components for visualization and rigid body components for collision detection and dynamic simulation.

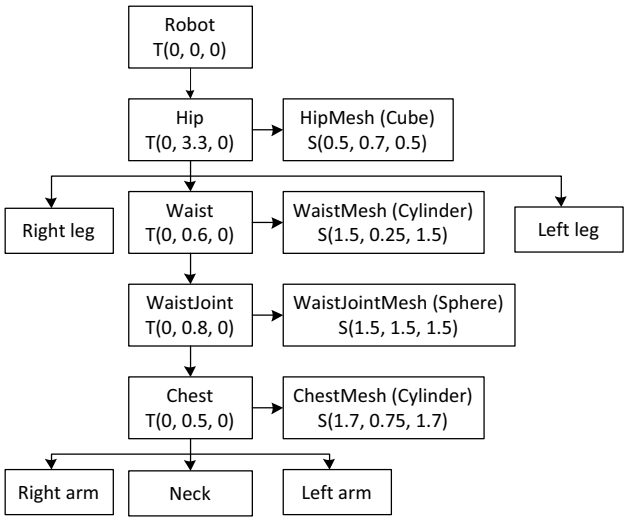


Figure 1. Robot design document

Note the design document given in Fig. 1 is only a partial list of a humanoid robot design. Based on the design document, the robot is created in Unity with a single metallic material, which is displayed in the Unity Game window as shown in Fig. 2. Only the primitive shape game objects (cube, sphere, and cylinder) available in Unity are used to add rigid body (or skin) to the robot. Since the robot is intended for use in games and not for real robot simulation, care is not taken to avoid overlapping of the rigid body parts.

### B. *Joint Creation*

Assembling a robot in Unity requires a variety of materials, mesh parts, and skeletal structures from which the designer can create the robot according to the design blueprint. The standard Unity assets include some materials and mesh game objects that are useful for robot construction, but they are not sufficient

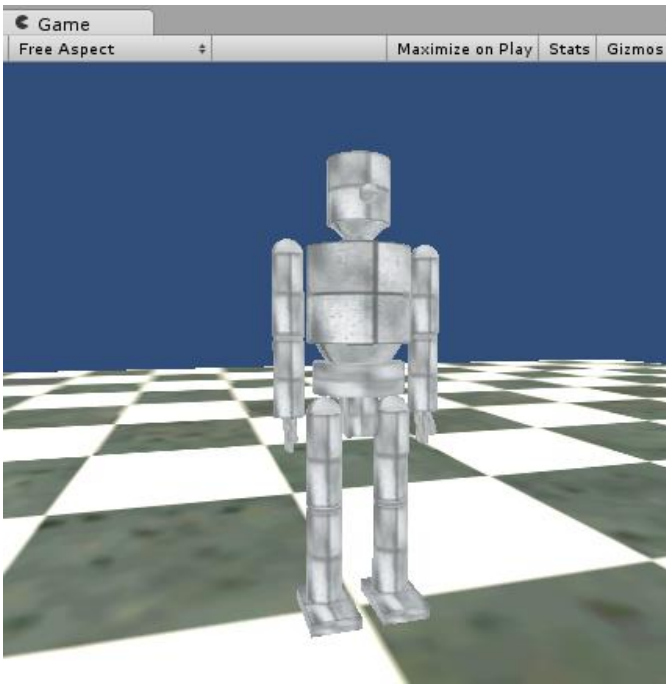


Figure 2. Robot in Unity Game window

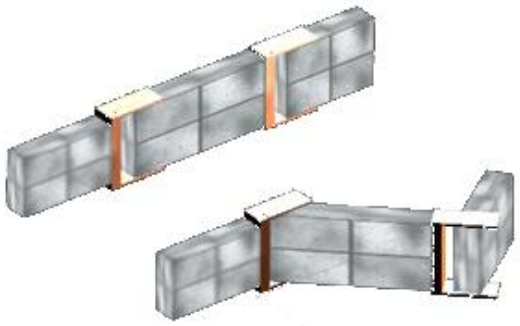

Figure 3. U-shaped joints

for the robot design system. For example, U-shaped rotational joints are commonly used in humanoid robots (e.g. the KHR series humanoid robot kits from KONDO Kagaku Co [6] in Japan). Unity does not include a U-shaped mesh game object, but it can be created as a Unity prefab (pre-fabricated) asset called U-joint and duplicated in the scene to make as many U-joint game objects as needed. Fig. 3 shows how U-joints are used in Unity to make two robots with linearly articulated rotational joints.

The four orange U-joints in Fig. 3 all are made from the same prefab. Even robots composed entirely of this relatively simple U-joint can have complex design and sophisticated behavior. Existing industrial robots provide an important source of inspiration and data for the robot design system. Fig. 4 shows a fairly primitive SCARA-type industrial robot created in Unity. RoKiSim [7], an educational software tool for 3D simulation of six-axis PUMA-type serial robots, includes several popular industrial robot arm meshes (e.g. 3D models of robot joints and end-effectors for grippers and tools). Robots developed for research offer a good source of robot design ideas. For example, robots designed to have snake-like characteristics, or serpentine robots, can be created in Unity based on the ideas given in [8, 9].

## IV. ROBOT ANIMATION

### A. Animation Curves

Unity has an Animation view for the user to author 3D keyframe animation clips and an Animation class for controlling play of animation clips via scripting. An animation clip consists of a set of animation curves, each of which determines how a game object property (e.g. transform, color, and texture) will change over time. Fig. 5 shows the Animation view for the robot shown in Figure 2. In it are four animation curves, which define four leg joint x-axis rotations in a walk cycle animation.

In Fig. 5, near the top and above the curves, there are eight white diamonds which represent keyframes inserted by the user. Based on the keyframes, Unity creates corresponding keys on an animation curve and uses an interpolation scheme to connect keys to build the animation curve.

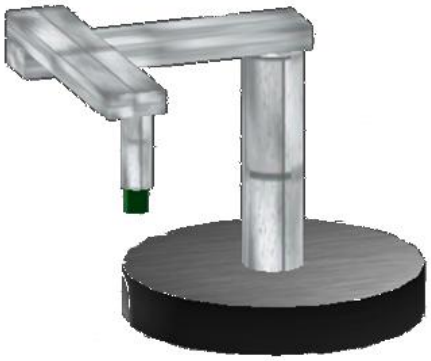

Figure 4. SCARA-type industrial robot

### B. Frame Rates

Unity games typically run at a variable frame rate. The frame rate may be higher on some computers than on others, and it may also vary from one second to the next based on the complexity of the scene update. Therefore when Unity plays an animation, it must sample animation at variable frame rates. Since animation curves are continuous, they can be sampled at any point in time. As a result, an animation can be played with the same speed regardless of the frame rate. Higher frame rates are always preferable, as the resulting animation looks smoother and more fluid.

The Animation class contains methods to play an animation clip with a specified wrap mode (once, loop, etc.), cross fade from one animation clip to another, blend (e.g. add) two animation clips, and so on. Scripts can even be written to alter an animation during its playback.

## V. ROBOT DESIGN LIBRARY

I The core of the proposed robot design system is a library of various assets, which can be organized as a set of Unity packages. A library asset can either be created inside Unity using prefab or procedural scripts with Editor classes or imported from external software applications (e.g. importing FBX files containing 3D models and animation clips created in a modeling package like Maya). The library consists of the following five components:

- Materials
- Mesh parts
- Skeletons
- Animation clips
- Scripting library

The design of the library allows new assets to be added easily. We discuss each component and its related assets.

### A. Materials and Meshes

A material in Unity consists of a shader and properties (e.g. color and texture) used by the shader. Unity comes with many

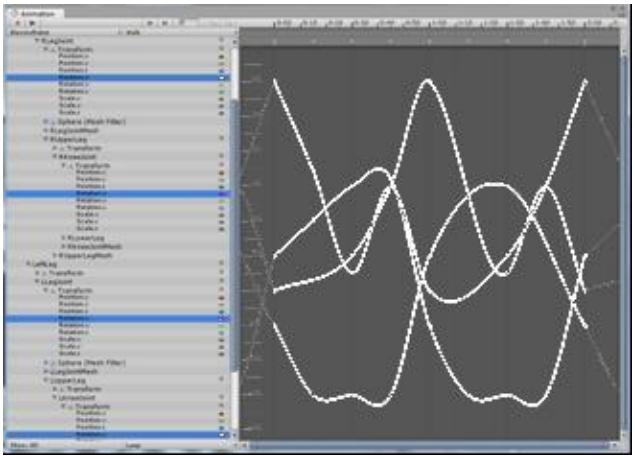

Figure 5. Animation curves of a robot walk cycle in Unity

standard shaders. Textures suitable for robots can be created in any image editing program and imported to Unity. Custom shaders can be implemented in Cg, High-Level Shading Language or OpenGL Shading Language. Although simple mesh parts can be created in Unity using GUI or procedural scripting (like the U-joint used in Fig. 3), more complex meshes need to be imported from FBX files. Many 3D modeling applications supporting FBX file creation and conversion are available including Maya, 3D Studio Max, and Blender.

### *B. Skeleton*

A skeleton is a hierarchical structure of joints like that shown in Fig. 1. A skeleton in the library may or may not contain mesh bodies, and it can be created in Unity or imported from motion capture data. Motion capture data files (e.g. ASF/AMC and BVH files) usually contain a section of skeleton definition. The Carnegie Mellon University Motion Capture Database [10] contains many motion capture data files (ASF/AMC file format) free for all uses. One problem of using a skeleton to start creating a robot in Unity is the skeleton may not have visible components attached. To solve this problem a script can be written to draw "bones" (e.g. lines) to connect joints in the skeleton. The script can be easily attached to and removed from the skeleton as needed.

### *C. Animation Clips*

Animation clips can be created using many different methods. We highlight those animation techniques which can be achieved in Unity: Manual keyframe animation, Motion capture animation, Procedural animation

Manual keyframe animation can be done in Unity using the Animation view. In order to design believable animation, the animation design process needs to take advantages of the basic principles of 2D drawing animation that is commonly practiced by professional human animators [11, 12]. To use motion capture data to create a desirable animation it requires data editing and fitting. Among other techniques, Multon *et al.* survey procedural animation approach to human walking animation [13].

### *D. Scripting Library*

In Unity robot control and simulation can be done through scripting. For example, Unity has a Character Controller class, which includes a script to move the controlled game object in a specified direction. When the game object collides with another object, the move script will adjust the move direction to move the game object away from the collided object. More scripts are needed to control robots in a game playing environment. Although Unity has a Rigidbody component, which supports rigid body dynamics behavior, there are no scripts to support robot control simulation. Robot control simulation is a very complicated task. It requires a library of classes in the chosen scripting language to support the basic functions of robot control simulation such as inverse kinematics and dynamics.

## VI. FUTURE WORK

Future plans for this project involve extending the current library of Unity assets to include more varied types of robotic components and animations.

The arms in Fig. 3 could use a different type of joint which allows rotational movement along a new axis. Different types of robotic locomotion can be added which allow the robots to move around and interact with their environments in a convincing manner. Once the meshes and animation parameters for these components are added to the library, more diverse robots can be constructed and their behavior simulated using Unity.

## REFERENCES


[1] Wikipedia on Robotics Simulation http://en.wikipedia.org/wiki/Robotics_simulator

[2] The Virtual Robot Experimentation Platform (V-REP) http://www.v-rep.eu/

[3] Cyberbotics Webots http://www.cyberbotics.com

[4] Unity http://unity3d.com/

[5] Nvidia Corporation PhysX http://developer.nvidia.com/physx

[6] KONDO Kagaku Co http://kondo-robot.com/

[7] RoKiSim http://www.parallemic.org/RoKiSim.html

[8] Grzegorz Granosik and Johann Borenstein, "Integrated Joint Actuator for Serpentine Robots", IEEE/AME Transactions on Mechatronics, Vol. 10, No 5, Octorber 2005, pp. 473-481.

[9] A.Maity, S.K. Mandal, S. Mazumder and S. Ghosh, Serpentine Robot: An Overview of Current Status & Prospect, 14th National Conference on Machines and Mechanisms, NIT, Durgapur, India, December 17-18, 2009.

[10] Carnegie Mellon University Motion Capture Database http://mocap.cs.cmu.edu/

[11] John Lasseter, "Principles of Traditional Animation Applied to 3D Computer Animation", Computer Graphics, Vol. 21, Number 4, July 1987, pp. 35-44.

[12] Chris Webster, *Animation The Mechanics of Motion*, Focal Press, 2005.

[13] M. Multon, L. France, M-P. Cani-Gascue, and G. Debunne, "Computer Animation of Human Walking: a Survey", *Journal of Visualization and Computer Animation*, 10:39-54, 1999.