Realistic Rendering of 3D Avatar Using Physics Simulation and Minimal Sensors

Student Name: Srikrishna Acharya B
Roll Number: MT12106

Thesis report submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Electronics & Communication Engineering with specialization in
VLSI & Embedded Systems
on 16th may,2014

Thesis Committee
Dr. P.B Sujit, IIIT-D
Dr. Kaushik Saha, Samsung R & D
Mr. Amit Kumar, ST Microelectronics

Indraprastha Institute of Information Technology
New Delhi
This research was partially funded by ST Microelectronics Pvt Ltd.

Student's Declaration

I hereby declare that the work presented in the report entitled **"Realistic Rendering of 3D Avatar Using Physics Simulation and Minimal Sensors"** submitted by me for the partial fulfillment of the requirements for the degree of *Masters of Technology* in *Electronics & Communication Engineering in VLSI and Embedded Systems* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Dr. P.B Sujit**. Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

..... **Place & Date: .....**

Srikrishna Acharya B

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

..... **Place & Date: .....**

Dr. P.B Sujit

..... **Place & Date: .....**

Mr. Surinder Pal Singh

Abstract

Physics simulation offers the option of truly receptive and realistic animation. Although wide acceptance of physics simulation for the animation, commercial applications still relies on kinematics-based approaches for the animation of actively controlled characters. In recent times, however, research on interactive character animation using simulated physics has resulted in incredible enhancements in controllability, effectiveness, flexibility and visual fidelity. In this work, the objective is to develop an application to render 3D avatars of users in a realistic way by using the laws of physics to minimize number sensors i.e. using physics based constraints related to animation of the avatars and apply this technique to constrain skeleton(e.g. a knee or an elbow has only some degrees of freedom) for a more realistic simulation.

The application is intended for virtual reality based social networking, such that avatars of the networked users can be seen to interact with each other and their virtual surroundings in a realistic manner. We present a structured evaluation of relevant aspects, approaches and techniques followed in the project and conclude by pointing out some open research areas and possible future directions.

Keywords: Physics Simulations, Interactive Character Animation, Mocap Sensors, 3D avatar Rendering, Virtual Reality, Physics Constraints

Acknowledgments

I would like to first thank my adviser Prof. P.B Sujit, for providing excellent guidance, encouragement throughout the project work. Without his invaluable guidance, this work would never have been a successful one. I take this opportunity to express a deep sense of gratitude towards Dr. Kaushik Saha and Prof. R. N Biswas for continuous support and guidance throughout the process. Thanks to IIIT-D and ST Microelectronics for providing excellent environment and infrastructure. Thanks to my supervisors Mr. Manoj and Mr. Amit Kumar at ST Microelectronics for providing ideas and giving me feedback. I have to thank G.N.S Harsha and J. Shamili to reduce feelings of homesickness. I would also like to thank all my classmates for their valuable suggestions and helpful discussions.I would special mention my co-partner in the project and good friend Supratim Das for his support, guidance and advice throughout the process. Last but not the least, I would like to thank my family for supporting me spiritually and emotionally throughout my life.

> "A mediocre person tells. A good person explains. A superior person demonstrates. A great person inspires others to see for themselves" $\sim HarveyMackay$

Contents

List of Figures

Chapter 1

Introduction

> "It is a capital mistake to theorise before one has data. Insensibly one begins to twist facts to suit theories instead of theories to suit facts." $\sim SherlockHolmes$

1.1 Motivation

Since the earliest days of the art form[1], animators have observed the movement of real creatures in order to create animated motion. Sometimes, this simply takes the form of an artist carefully observing nature for inspiration. Another process is to transfer the movement from a recording of the movement to the animated objects. The earliest mechanism for doing this was the Rotoscope, a device that projected frames of film onto the animator's work space, providing the animator with a guide for their drawings. Many computer graphics applications engross virtual environments in which characters and objects continuously interact with each other and with the environment. Appropriate animation of such interaction is significant for the perceived realism of these virtual environments. Creating such realistic receptive animation is very challenging, because subtle variations in early contact may call for significantly diverse responses.

Kinematics based animation frameworks depend profoundly on existing motion data, either recorded or manually crafted through key-framing. During interactions, appropriate responses are selected based on actions, regulations and scripts, after which representative animations are generated us-

ing motions from a database. However the last decade has brought immense advances both in accessibility and utilization of motion data,but data-driven animation cannot generate realistic and non-repetitive receptive animations due to limitation in the contents of the motion database.

Physics based simulation essentially offers different approach to computer animation. Instead of directly manipulating the motion trajectories of objects and characters, this approach provides motion a product of a physics simulation process i.e physics simulator forms an integral part of animation loop. As a consequence, physics-based characters and objects involuntarily interact in a way that is physically precise, without necessitating supplementary motion data or scripts. i.e, motion of an object in the virtual environment is the direct result of physics simulation, and control within the environment occurs only through the application of forces and torques, similar to real world motion. The result is that all responses of interacting entities are physically realistic by definition. In addition, subtle variations in initial interaction conditions result in unique and original animations.

1.2 Problem Statement

The objective of this dissertation is to generate a full body animation from minimum number of sensors. Fifteen orientation sensing devices are sufficient to provide a full body posture tracking[14] although more can be used to provide increased detail in areas such as the spine and shoulders. Recent advances in microelectromechanical system (MEMS) technologies aided the development of small and inexpensive sensors; still inertial motion capture system is considered costly method. The major factors are number of sensors and a firmware to support the system results in a high maintenance cost.

The main idea discussed in this dissertation is to map the mo-cap sensors[15] to a body model composed of rigid bodies allowing reconstruction of the subject's posture as a result minimizing usage of number of sensors. The objective is to study the effect of minimizing the number of sensors under body model constraints and evaluated the performance.

The solution proposed in the dissertation is to minimize the deviation of the reduced sensor model from the full sensor model as shown in the below equation 1.1 by subjecting to Physics Constraints.

$$error_i = \sum_{i=1}^{T} fullbody_i - reducedbody_i \tag{1.1}$$

$$minimize(error)|PhysicsConstraints \tag{1.2}$$

1.3 Literature Review

Physics simulations to animate virtual characters has been documented in an early phase of 3D computer animation[2] and has incited numerous research publications since 1990. For the animation of unreceptive phenomena, such as cloth, water and rag-doll characters, physics based simulation has been subject to widespread commercial adoption, both in video games and production movies. However, despite more than two decades of research, commercial frameworks still resort to kinematics based approaches when it comes to animating active virtual characters[3]. Primarily physics-based characters have some issues related to controllability. Just like with actual world characters, the pose of a physics-based character is controlled indirectly, through forces and torques generated by actuators that exist inside the character analogous to muscles in biological systems. In addition to this, global position and orientation of a physics-based character are unactuated and can only be controlled through deliberate manipulation of external contacts. This characteristic (in literature also referred to as underactuation) poses a direct challenge to basic tasks such as balance and locomotion, a challenge that has no equivalent in kinematics-based animation. As a result physics-based characters are less reactive during direct control tasks than kinematics-based characters.

Even though professional physics simulation software has become readily available, the animation of active physics-based characters is a compound matter that alludes in various disciplines. Successful execution of a physics-based character animation framework requires at least some quantity knowledge of multi-body dynamics, numerical integration, biomechanics and optimization theory.

To put together, many physics-based control strategies necessitate time consuming manual tuning before they can employ a process often scantily documented in research publications. To conclude, physics-based character animation is computationally more expensive than kinematics based alternatives. It may only be for about a decade that a single passive physics-based character can be simulated in real-time on a consumer grade PC. Currently, real time performance of many recent control strategies[4] is still limited to around a single character at a time on modern hardware.

When the user interacts with the virtual character, such constraints are not accurately known in advance as the user can modify the actions and the environment in an unpredictable way. An approach[5] was proposed to easily design a controller that acts as a parametric model of displacements based on stylized motion capture data contrary to motion graphs that are based on large database of motions. Another alternative approach[6] was proposed for acquiring motions almost anywhere. The proposed wearable system gathers ultrasonic time of flight and inertial measurements with a set of inexpensive miniature sensors worn on the garment. After recording, the information is combined using an Extended Kalman Filter to reconstruct joint configurations of a body. Experimental results show that even motions that are traditionally difficult to acquire are recorded with ease within their natural settings. However, this approach does not cover all the global transformations.

Alternatively a low cost solution was proposed using two webcams[7] then again these approaches are limited to indoor further has occlusion related problems.Traditional vision-based and commercial motion capture systems acquire human body motion within studio. A more detailed approach of ubiquitous human limb motion capture system using wearable micro-inertial-sensors (accelerometers and gyroscopes) was presented[8]. The sensor units are attached to segments of upper limb, and the collected acceleration and angular rate data are transmitted to the PC via USB interface. Motion data are fused by a quaternion-based Kalman filter to reconstruct orientations of body segments in real time. A sensor data-driven layered human motion model is developed for real-time animation. Since the number of sensors are high an attempt to minimize the number of sensors

was proposed in [9]. In this approach series of online local dynamic models from a prerecorded database was utilized to simulate full body motion.

The idea of using body model constraints to improve accuracy of inertial motion capture was proposed in [10]. The proposed method utilizes the correlation between rotational motions and linear accelerations to estimate and correct accelerometer-based orientation estimates.A review of a structured evaluation of physics-based simulations aspects was presented in [11], In this various approaches and techniques regarding interactive character animation using simulated physics based on over two decades of research were discussed. Another attempt was made in minimizing the number of sensors for an interactive control of a full-body human character[12]. The idea is to construct a series of online local dynamic models from a prerecorded motion database and utilize them to construct full body human motion in a maximum a posteriori framework (MAP). Similarly another review on various applications of interactive rigid body simulation was presented in [13]. After studying the physical principles underlying the variety of approaches to motion tracking, No single technology can be used for all purposes, certain methods work quite well for specific applications.

This dissertation is a merger of both motion capture and physics simulation in interactive character animation such that we get an optimal solution using minimum number of sensors by introducing physics laws of motion. This solution targets certain use cases where animations are symmetric like walking, sitting, standing and jumping (conditional) etc on Embedded platforms. The end application is tested for real-time rendering on embedded platforms like Panda Board, Linux/PC (Ubuntu), Snowball and ORLY (ST Product). The results are also compared with the kinnect based motion capture and commercial motion capture system with a full set of motion sensors and it is comparable in quality.

In this work a framework was developed that displays the 3D content along with physical constraints mapped and hereby referred as assView player. A open source library is used to provide body model constraints to the main player. In the next section a brief overview of the entire work

has been supplemented answering choice of sensors, API's, and content generators.

1.4 Overview

In this section various design choices like choices of sensors, 3D content providers, the APIs, physics library and the development board are discussed in detail.

1.4.1 Inertial Mocap Sensor vs Optical Sensors

Motion capture, or mocap, is a technique for digitally recording the movements of humans and animals. Despite their advantages, motion capture has always been considered to be an expensive technology. Traditionally, computer animation techniques are used to create movements of a living being. But with motion capture we can create more realistic animation. There are motion capture systems based on Acoustic, inertial, LED, magnetic or reflective markers, or combinations of any of these. Out of these Optical and Inertial motion capture system are exhaustively used for because of accuracy and non-intrusive nature. In [16] Optical and Inertial motion capture system are compared and evaluated. The major problem with optical based system is the number of markers we need to track are very high and occlusion problem is quite significant. In the last decade articles have addressed this of inertial motion capture (IMC) for Human motion [17]. This is largely due to recent advances in micro-electromechanical system (MEMS) technologies allowing for the development of small and inexpensive sensors such as accelerometers, gyroscopes and magnetometers which form the key components of inertial motion sensor units.

1.4.2 MPEG 3DGC

Embedded application processors have evolved dramatically over the past years, offering ever increasing computation power and hardware support for critical features such as 3D Graphics with programmable OpenGL ES-capable GPUs. Most of these 3D assets are encoded in proprietary formats, limiting the possibility of exchange and dissemination. The MPEG 3DGC standard aims at solving this problem, in offering a compressed and comprehensive way to store and deliver animated, textured 3D assets across different platforms.. The MPEG3DGC standards[18] ISO/IEC 14496-2,

ISO/IEC 14496-11, ISO/IEC 14496-16, ISO/IEC 14496-21 and ISO/IEC 14496-25 cover the compression of 3D graphics primitives (such as geometry, appearance models, animation parameters, etc.), the representation, coding and spatial-temporal composition of synthetic objects. In parallel to these activities, 3DGC is also involved in standardizing the coupling of the 3D compression tools developed within MPEG with other scene representation formats, which is treated in Part 25 of the MPEG-4 standard. The major problem with the 3DGC player is the content generation tools are not open-source so the models that are available are very limited hence open-source 3D contents providers options were explored.

1.4.3 Collada Format

COLLADA[19] is an interchange file format for interactive 3D applications. It is managed by the nonprofit technology consortium, the Khronos Group, and has been adopted by ISO as a publicly available specification, ISO/PAS 17506. COLLADA defines an open standard XML schema for exchanging digital assets among various graphics software applications that might otherwise store their assets in incompatible file formats. COLLADA documents that describe digital assets are XML files. The main advantage is Collada enables easy exchange of 3D data among (industrial) applications.

1.4.4 Assimp Library

The Open Asset Import Library[20], or Assimp, is an open source library that can handle many 3D formats, including the most popular ones. It is very easy to use and integrate into programs written in C/C++. Assimp is independent of the Operating System by nature, providing a C++ interface for easy integration with game engines and a C interface to allow bindings to other programming languages. At the moment the library runs on any little-endian platform including X86/Windows/Linux/Mac and X64/Windows/Linux/Mac. The main advantages are it is portable, open source and available for both Linux and Windows.

1.4.5 Bullet Physics

Bullet Physics[21] Library is an open-source library that provides collision detection, soft body and rigid body dynamics for the simulation of multi-body physical interactions. It is used in gaming and movie making to provide realistic rendering of interactions between graphical objects. It can also be used for rendering of interactions between graphical objects and the environment e.g. gravity, wind etc. It can be used on PCs, mobile terminals and other consumer electronics platforms. The most recent version available for free download is v2.8. An OpenCL accelerated version to exploit data parallelism is planned for v3.x.

Figure 1.1: Bullet Physics Animation Screenshot

Bullet Collision Detection module provides the following functionality:

1) Discrete and continuous collision detection (CCD)

2) Sweep collision queries

3) Ray casting with custom collision filtering

4) Generic convex support (using GJK), capsule, cylinder, cone, sphere, box and non-convex triangle meshes.

5) Support for dynamic deformation of non-convex triangle meshes, by refitting the acceleration structures

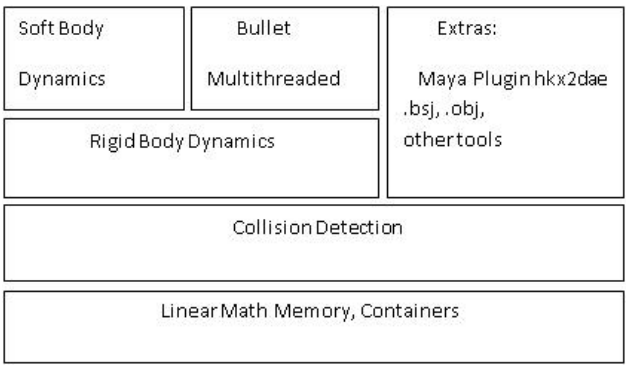


Figure 1.2: Design of Bulletphysics Library

Multi-physics simulation support is provided and includes:

1) Rigid body dynamics including constraint solvers, generic constraints, ragdolls, hinge, ball-socket

2) Support for constraint limits and motors and also soft body support including cloth, rope and deformable

3) Bullet is integrated into Cinema 4D, Lightwave, Blender and Carrara, and plugins for Maya, Houdini and 3ds Max are available

4) Serialization of physics data in the cross-platform binary .bullet file format

The Library is free for commercial use and open source under the ZLib License.
This development will be integrated to the assView player. For example, a complex ragdoll object like a troll object as shown in the figure 1.3 shall be defined with an associated skeleton for real-time animation using data pertaining to the users motions available from motion sensors with a physics kernel associated as a first step.

Figure 1.3: Troll Object

1.4.6 Panda Board

PandaBoard is intended to be used as a platform for software development. Since it supports Android, LinuxARM, Linaro and Fedora distributions, application development on this platform has higher flexibility.

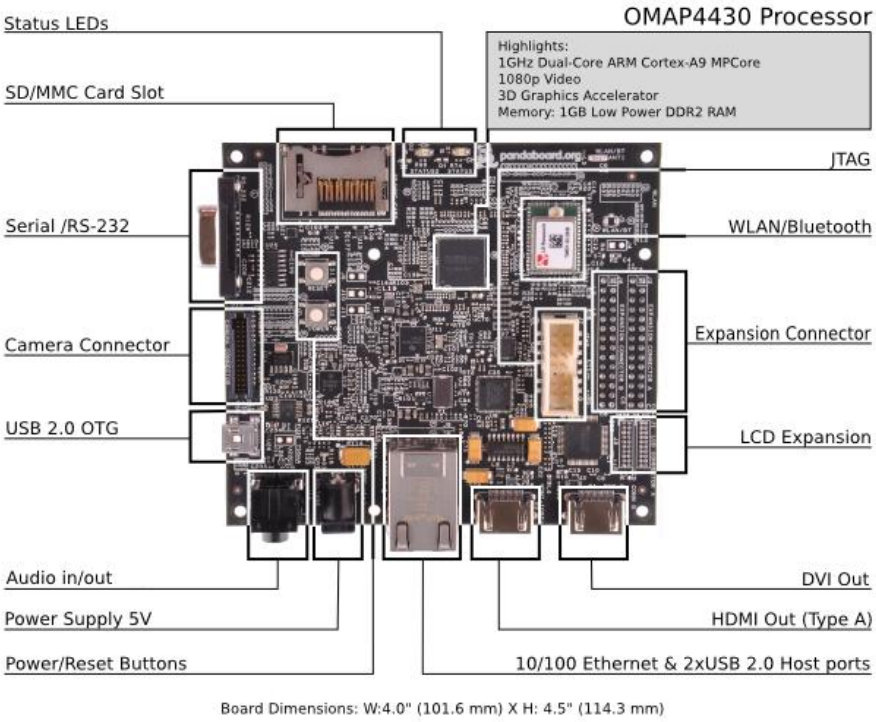


Figure 1.4: PandaBoard

1) The PandaBoard is a development platform based on the Texas Instruments OMAP4430 system on a chip (SoC). It is a low power, low cost single board Computer development platform. PandaBoard ES is newer version with CPU and GPU running at higher clock rates.

2) The PandaBoard ES uses a newer SoC, with a dual-core 1.2 GHz CPU and 384 MHz GPU. Primary persistent storage is via an SD Card slot allowing SDHC cards up to 32 GB to be used.

3) The device runs the Linux kernel, with either traditional distributions or the Android or Mozilla Firefox OS user environment. The PandaBoard has an integrated SGX540 graphics processor and provides 1080p HDMI output.

4) This GPU supports OpenGL ES 2.0, OpenGL ES 1.1, OpenVG 1.1 and EGL1.3. The situation for Linux - x11 utilizing hardware floating point libraries is PowerVR's SGX540 GPU hardware is unusable without a GPU driver.

The above features of PandaBoard best suits to the application that this dissertation was targeting.

1.5 Conclusion

This dissertation is organized as follows. In Chapter 2 assView player development and the architecture of the system was discussed. In Chapter 3 the very important aspect of sensor orientation calibration phase which plays a vital role in calculating the offset rotation was explored. Chapter 4 mainly concentrates on Physics implementation and the SIMBICON paper[22]. Chapter 5 elaborates the the optimum sensor positioning. Chapter 6 will develop abstractions to support the work with appropriate results, conclusions and followed by the further scope of work.

Chapter 2

assView Player

"Speed, it seems to me, provides the one genuinely modern pleasure." $\sim Aldous Huxley$

In this chapter the assView player that has been developed to display 3D content is discussed eloborately. As already discussed COLLADA 3D content providers is used in the mainplayer and the Assimp API is used to develop the player.

2.1 Architecture

An interactive application is composed of two major components:

1) The application, which provides information in real time to the user and the means to interact with it;

2) The content, which contains the information through which the application navigates and provides a view to the user.

COLLADA focuses on the domain of interactive applications in the entertainment industry, where the content is three-dimensional and is a game or related interactive application. Therefore, the user of the application will be referred to as the player.

The types of information that can be provided to the player depends on the output-devices available. Most games use one or several screens to display the visual information, sometimes a system

with stereo visualization, and a set of speakers for the audio information. Often, some physical sensation can be rendered, as simple as a vibrating device embedded in the joystick, or as sophisticated as a moving cabin in arcade settings. The application may output, or render, several sensors at the same time, often in different places. An OpenGL based player named assView(asset viewer) player is designed to provided a platform for the player to interact with the content on the screen.

The content in the COLLADA file has to be converted into engine-specific format for easy and fast every-day-loading i.e. the content in the file has to be loaded into appropriate data structures. ASSIMP API is used to import the 3D assets into engine. The imported data is returns an aiScene structure. This is the root point from where you can access all the various data types that a scene/-model file contain. aiScene forms the root of the data, from here you gain access to all the nodes, meshes, materials, animations or textures that were read from the imported file. Nodes are little named entities in the scene that have a place and orientation relative to their parents. Starting from the scene's root node all nodes can have 0 to x child nodes, thus forming a hierarchy. They form the base on which the scene is built on: a node can refer to 0..x meshes, can be referred to by a bone of a mesh or can be animated by a key sequence of an animation. To apply such an animation you need to identify the animation tracks that refer to actual bones in your mesh.
Then for every track:

1) Find the keys that lay right before the current anim time.

2) Optional: interpolate between these and the following keys.

3) Combine the calculated position, rotation and scaling to a transformation matrix.

4) Set the affected node's transformation to the calculated matrix.

The assView player loads 3D assets into the engine the model is rendered using openGL utility library GLUT. In the next chapter sensor placement and the most important sensor orientation aspect is discussed.

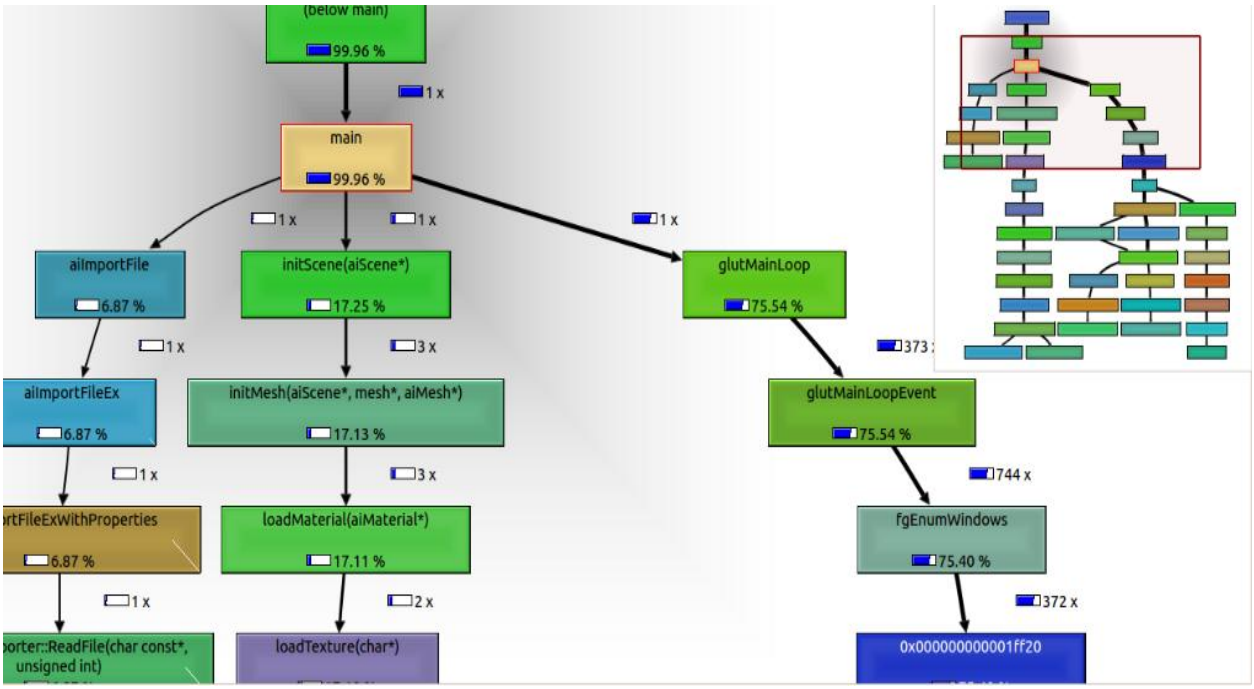


Figure 2.1: function call hierarchy

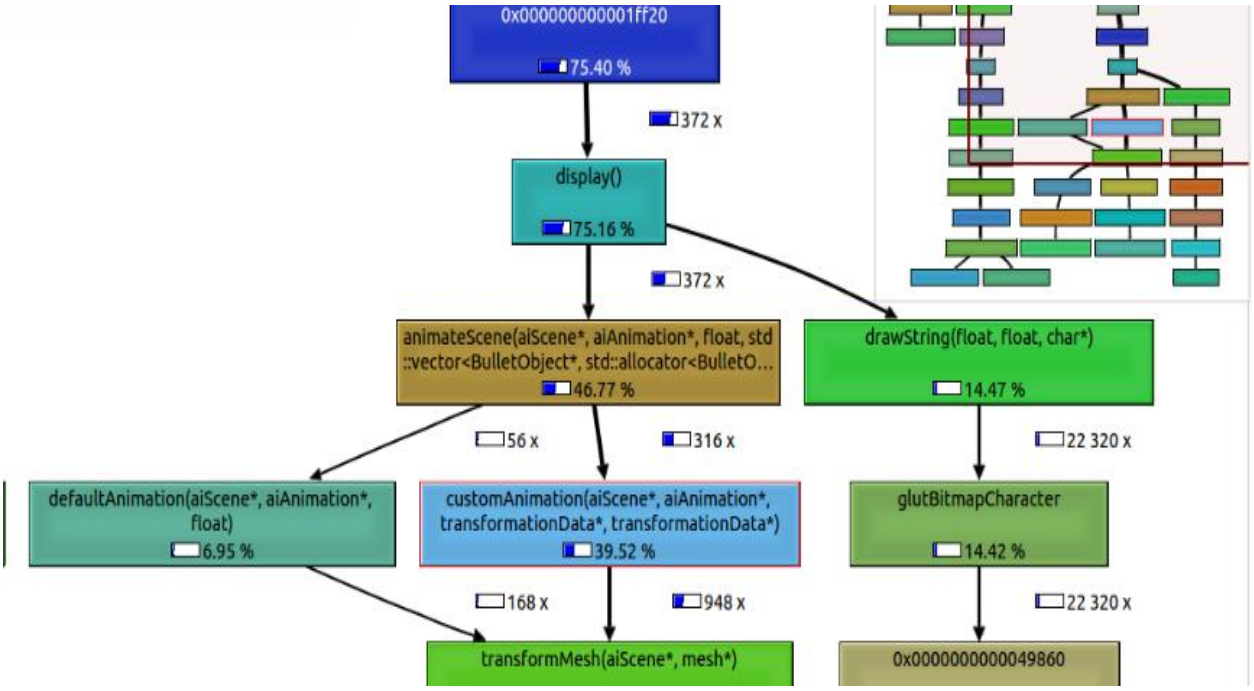


Figure 2.2: function call hierarchy contd

The entire project is profiled using Valgrind[23].

Chapter 3

Sensor Orientation Calibration

> "Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you've proven that's where the bottleneck is." $\sim$ *RobPike*

Inertial Motion Capture technology is based on miniature inertial sensors, biomechanical models and sensor fusion algorithms.The Inertial mo-cap sensors give the absolute rotation around a reference axis at any time. The motion data of the inertial sensors is often transmitted wirelessly to a computer, where the motion is recorded or viewed. Most inertial systems use gyroscopes to measure rotational rates. These rotations are translated to a skeleton in the software. In this chapter the positioning of sensors and the problem orientation of sensors is discussed in detail.

3.1 Sensor Placement

As shown in the Figure 3.1 the yellow cuboids are placed at the major sensor points. One of the major problems faced while our work on integrating wearable sensor data with the player was getting the axis system correctly. In other words it means to have a proper understanding about the axis system of the sensors and that of the 3D avatar and come up with a way that both of them are in agreement with each other.

Most of the time we relied on visual inspection of the character's (The 3D avatar of assView

player) movement with respect to some reference sensor data. It was found during this course of action that although most of the time the degree of movements (rotations) were correct, it was associated with some unnatural twists. Some of these issues are discussed in the next section.

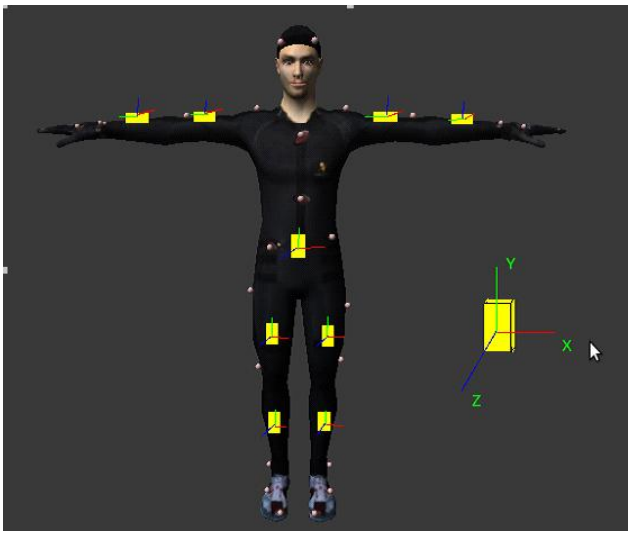


Figure 3.1: placement of sensors

3.2 Orientation Problem and Solution

The sensor orientation problem has double impact:

1) Orientation of multiple sensors with respect to a reference sensor.

2) Orientation of the sensor axis system with respect to the model.

At any point the user will be wearing multiple wearable sensors at various points of the body like the belt region, thighs, legs, upper forearm, lower arm etc. Now these sensors may be placed differently by the users, which is not possible to determine explicitly. It may be made mandatory that every sensor should be placed in a specific fashion, but there will be some form of ambiguity always, and since every part of the 3D avatar follows a uniform axis, this will create unwanted deformities during the process of the animation.

The problem can be better illustrated by the Figure 3.1. This shows a possible placement of

sensors at various regions of the user. The orientation of the sensors positioned at belt, thigh and leg are the same, but the sensors in the upper and lower arm are different.

The sensors placed in the arms have a -90 degree twist around z-axis followed by a +90 degree twist across y axis, with respect to the belt sensor. To bring all the sensors to a common reference frame it is required to find the relative quaternion[1] of each sensors in the T-Pose with respect one reference sensor (belt sensor in our case), then pre-multiply each incoming absolute quaternion with the inverse of the relative quaternion calculated before. The resultant quaternion will be absolute rotations with respect to a common reference.

The second problem is on fixing the differences in the orientation of the axis system of the avatar and the axis of the sensors. This problem again can be solved in a similar fashion. Specifically instead of applying directly the absolute quaternion, it is a better idea to calculate the relative quaternion across different samples and apply these to get the final animation.

To achieve this, we make a copy of the transformation data of the original avatar at T-Position and apply transformation to this instead of the original avatar for a predefined period (calibration / stability period) of around 5 seconds. It is assumed that the user will start making movements only after this defined period is over. The first relative quaternion is calculated against the rotation factor of the avatar in T-pose, which is 0. Subsequent relative quaternion are calculated against the previous transformation. After the calibration period is over the relative quaternion are simultaneously applied to the original animation as well as the copy of it.

The above concept is also illustrated in Figures 3.2, to 3.4. All the mentioned implementation is integrated in the assView player.

[1]Quaternions can be used to represent rotation in 3D

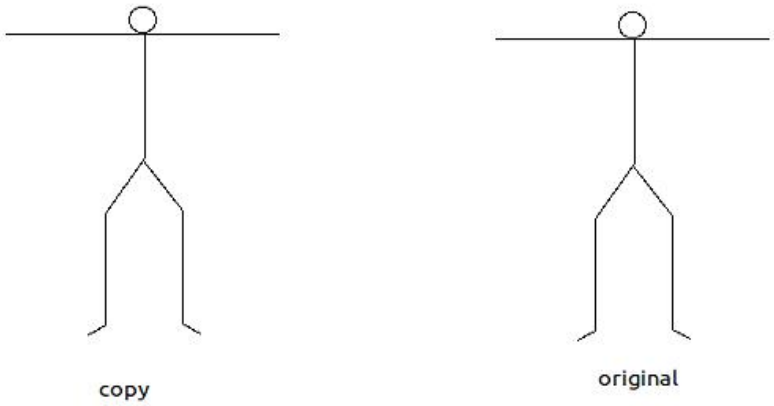


Figure 3.2: Original and a copy of original

Here an assumption has been made that the user will be standing in T-Pose and there will be no movements for 5 seconds the SENSOR_SAMPLE_RATE should be set according to the sampling rate of the sensor. In this case the sensors sample the data at 50Hz. Since the different sensors may not be aligned in a common axis, in the first two seconds the sensors will calibrate with respect to the belt sensor as a reference. Again the animation avatar's axis may not be aligned with the sensors axis system so in the next two seconds all the sensors will align with respect to the axis of the animation avatar's axis system. one second is kept as a margin after which the user may make movements and the animation is expected to be in synchronization.

While obtaining real time motion capture the relative orientations of the sensors experience some offset drift and this orientation offset is corrected by this calibration phase and the animations produced will be synchronization. The main disadvantage is we have to store the transformations in two different buffers resulting in larger memory requirement.

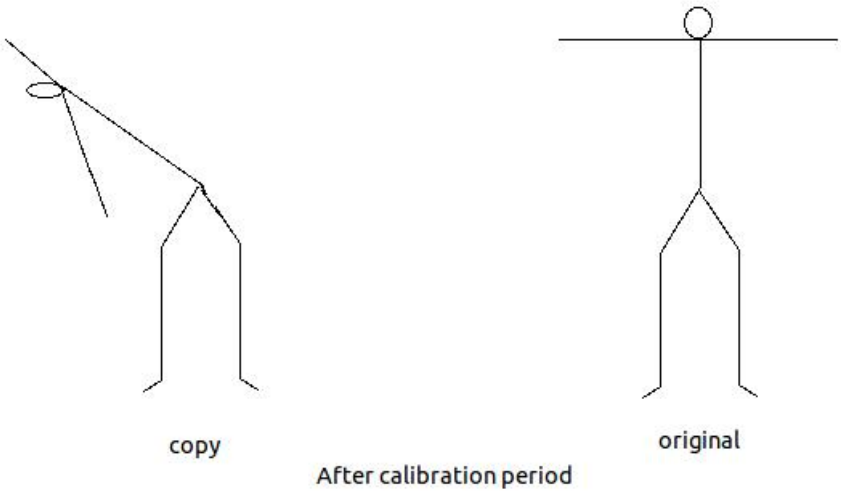


Figure 3.3: After calibration period

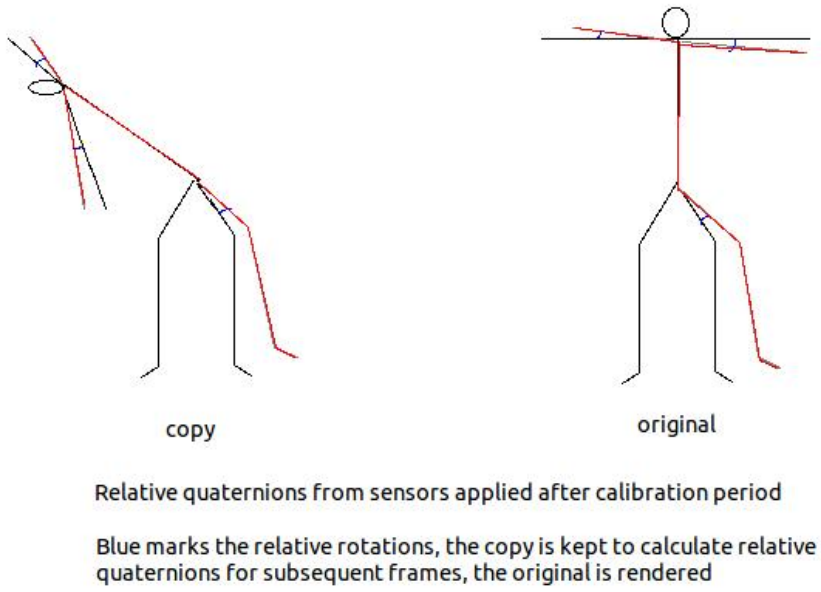


Figure 3.4: Applying animation using relative quaternions

To summarize we have developed an OpenGL based player and loaded a COLLADA model using Assimp library and we identified key sensor points and employed an orientation offset correction. Now let us dwell into the physics based simulations.

Chapter 4

Physics Based Simulations

> "More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity." $\sim W.A.Wulf$

The main task of a physics engine is to perform collision detection, resolve collisions and other constraints, and provide the updated world transform[1] for all the objects. This chapter will give a general overview of the rigid body dynamics rules that are integrated in the player. Finally concludes with physical constraints mapping with assView player.

4.1 Physics Object-Bullet Physics

The primary control object for a Bullet physics simulation is an instance of btDynamicsWorld[2] object. All of our physical objects will be controlled by the rules defined by this object. This World takes Broad Phase Detection (collision detection algorithm), Collision Configuration (box-box, sphere-box, and so on), Collision Dispatcher (constructor) and Constraint Solver as inputs. Bullet maintains the same modular design of its core components even down to individual physics objects. This allows us to customize physics objects through their components by interchanging or replacing them at will.

[1] World transform of the center of mass for rigid bodies, transformed vertices for soft bodies
[2] btDynamicsWorld provide a high level interface that manages physics objects and constraints

Three components are necessary to build a physics object in Bullet:

1) **Collision shape**, defining the object's volume and how it should respond to the collisions with other collision shapes.

2) **Motion state**, which keeps track of the motion of the object.

3) **Collision object**, which acts as a master controller of the object, managing the previously mentioned components and the physical properties of the object.

To encapsulate all these properties from the assView player physics objects with appropriate setter and getter methods are created.

4.2 Physics Implementation

A physics skeleton is constructed with appropriate constraints like ball and socket joint in shoulders, hinge joints near knee etc with the data derived from the T-pose of the assView model. The physics skeleton constructed is illustrated in Figure 4.1.

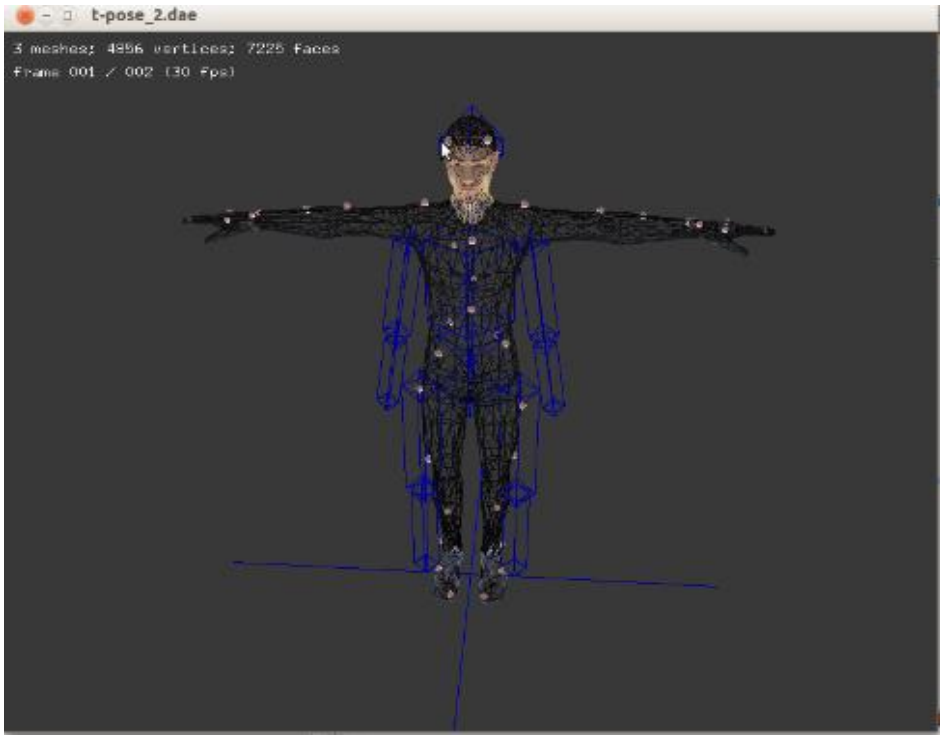


Figure 4.1: Mapping physical bones to key sensors points

After placing all the constraints in appropriate position they have to lock certain orientations using JointHelper Structure, since certain movements are not allowed for human ambulation. Based on the rag-doll model in the bullet demos and Parvez's[3] work[24] as base we create physics objects with appropriate dimensions at appropriate positions like pelvis, spine, head, legs and hands.

These physics objects are connected with appropriate joints at appropriate position and the degree of freedom has been defined such that it mimics human skeleton. Each joint has a JointHelper that defines the degree of freedom at each constraint.

The above code Snippet just explains procedure to setup the constraints[25] for spine bone and head bone. btGeneric6DofConstraint between two rigid-bodies each with a pivotpoint that describes the axis location in local space. It can leave any of the 6 degree of freedom 'free' or 'locked'.

$free : upper < lower$

$locked : upper == lower$

$limited : upper > lower$

A generic joint between two rigid bodies and Angular limits were set for each joint and this process is continued for all the rigid bodies based on human anatomy. The major issue is the 6DOF constraint animations are not "quaternion based" but "Euler axis based". Since in Y-axis once a joint exceeds +/-90 degrees, the constraint attains singularity.

AXIS	MIN ANGLE	MAX ANGLE
X	-PI	PI
Y	-PI/2	PI/2
Z	-PI	PI

Table 4.1: Angulars limits of btGeneric6DofConstraint

All current available solutions involve allowing one axis to freely rotate. We need to lock down all axes, with each axis low/high limit set to a single value in the range [-PI, PI]. In essence, we need

[3]Parvez was intern at STM working on this project prior to us

to constrain the rigid body to a single rotation matrix in a full sphere range of rotation. In Bullet, there is currently no solution for this type of problem, but it would be a common desired feature in many physics simulation projects. So far, it seems the best solution would be a quaternion-based 3DOF (rotation only) constraint. This is discussed by Claude Lacoursire in "Ghosts and Machines"[26], and his is not yet available in the bullet physics library.

In this project since the rotations we get from the sensors are compensated with gravity. we do not require a locomotion controller we just need to lock the pelvis bone. So pelvis bone rotation has to be restricted along the Y axis.This is similar to adding one more invisible joint connecting pelvis with the external world with locking rotations in x and z axis. As a result foot placement and the control strategy[27] is not required. This is not an optimal solution but most adopted solution.

There is another important optimization that Bullet handles internally, which is only visualized through the debug mode, but has a profound effect on CPU usage. Objects whose velocity is below a given threshold for a given amount of time have their activation state set to deactivated. Meanwhile, there are actually two dynamic bounding volume trees created while using a btDbvt-Broadphase object. One stores the active objects (the active tree), and the other stores any static or deactivated objects (the deactive tree). So, when an object is deactivated, it pushes them into the other tree.

This causes Bullet to skip over them when its time for the world to move objects around, and since the broad phase object only compares the active tree against itself, and the active tree against the deactive tree its impact on processing time is reduced even further. Later, when an active object collides with the deactivated one, it is activated once more, pushed back into the active tree, and Bullet performs the necessary calculations until it decides to deactivate it once more. These activation/deactivation states are typically referred to as putting the object to sleep, or waking it up.

This optimization has its drawbacks; sometimes an object may be moving slow intentionally, but if it is moving too slowly, Bullet deactivates it. For example, we might have an object that is very slowly teetering on an edge, which means it has a very low angular velocity for a long time, at which point Bullet may assume that it needs to put the object to sleep. This can cause some bizarre situations, where an object looks like it should be falling over, but is in fact it is frozen in place at an angle that would not be possible in the real world.

The typical workaround is to tweak the sleep threshold of the object, the minimum values of linear and angular velocity, which Bullet considers too low. But this will cost us some performance, since every object will now be a part of the active tree, and hence will be checked every iteration.

4.3 Physics based Animation

Now the most important aspect of physics based animation is introduced. The vital animation class btAnimation is created with btAnimationKeyFrame structure array is declared which has all jointKeyframes. So all the manipulations we set at each joint can be key-framed using btAnimation class. Once we have the entire set up for physics skeleton now we need to provide with some animations.

As the upper part has high degree of freedom,so the work has been divided into two parts Lower half of the body and Upper half of the body. First we concentrate on the lower half of the body. A state model for a basic symmetric animation walking was developed using concepts of SIMBICON 2007[22].

Sensor Position	STATE 1	STATE 2	STATE 3	STATE 4
Right Hip	-40º in x axis	0º in x axis	30º in x axis	-20º in x axis
Right Knee	40º in x axis	0º in x axis	0º in x axis	60º in x axis
Left Hip	30º in x axis	-20º in x axis	-40º in x axis	0º in x axis
Left Knee	0º in x axis	60º in x axis	40º in x axis	0º in x axis

Figure 4.2: State model of walking

As shown in the state model we have developed 4 state to model walking, at each state we give the corresponding joint key frames with the following angles. This in term uses motors to generate appropriate rotations to the physics skeleton at appropriate position. Similarly more symmetric animations like sitting, jumping and running are generated.

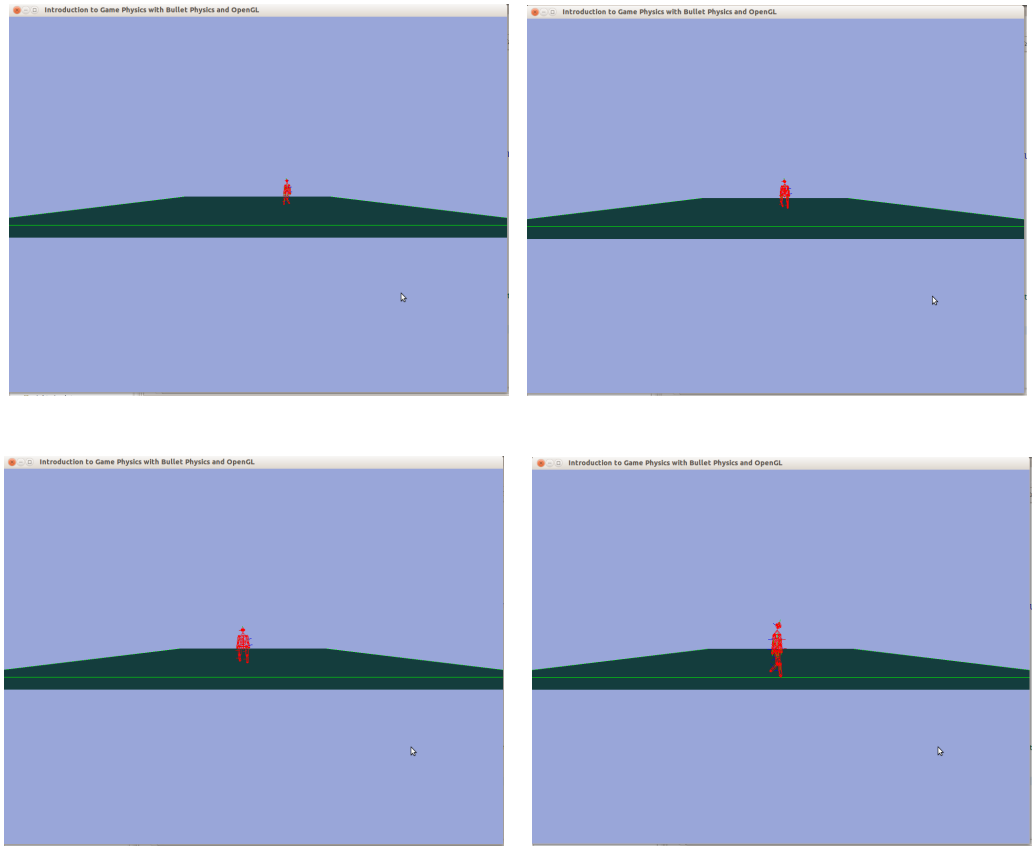
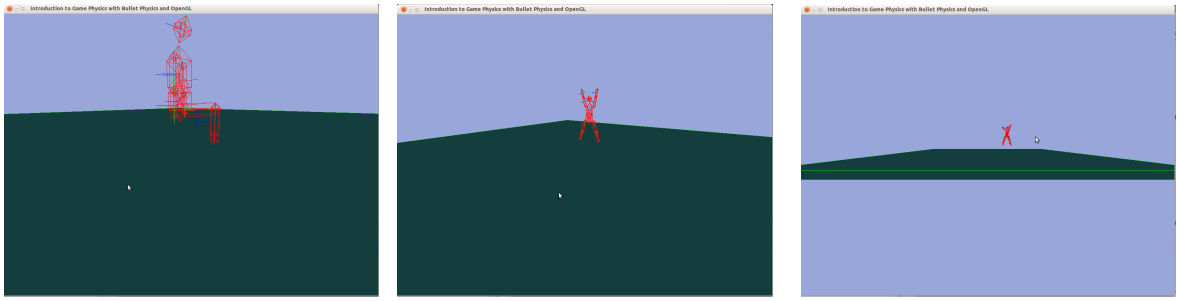


Figure 4.3: Screen shots of Walking animation



(a) Sitting pose (b) Jumping pose (c) Another view of jumping

Figure 4.4: More symmetric animations like sitting and jumping

To summarize till this point we have developed an OpenGL based player and loaded a COLLADA model using Assimp library and we identified key sensor points and employed an orientation offset correction. Using Bullet Physics library we have enabled physics simulations for the player. We

have mapped a ragdoll with humanoid constraints to the character model and added a physic based animation class to support animations to the physics character.

Chapter 5

Optimum sensors and its positioning

"Rules of Optimization: Rule 1: Don't do it. Rule 2 (for experts only): Don't do it yet." $\sim M.A.Jackson$

5.1 Objective

The main objective for this dissertation is to minimize the usage of number of sensors and still reconstruct the full body motion capture. We have identified minimum number of sensors required is 5 to detect human locomotion and ambulation[28], and the positioning of sensors are identified in the below Figure 5.1[1]. The positions of sensors are Right Knee, Left Hip, Belt, Right Arm and Left Forearm. This is paper the have developed and evaluated algorithms to detect physical activities from data acquired using five small biaxial accelerometers. The results show that Decision tree classifiers showed the best performance recognizing everyday activities with an overall accuracy rate of 84% which is very encouraging.

[1] these images are extracted from the same paper

(a) Sensor positions(marked in white circles

(b) Interpolated positions(marked in black circles)

Figure 5.1: Optimum sensors and its positioning

The main plan of execution is to exploit this symmetry to reduce the sensors points to five as shown in above Figure 5.1. We have to interpolate the missing sensor positions value i.e Right Hip and Left Knee as shown in the below Figure 5.1b with black circles.

Now the question is on what basis we have to interpolate? We have numerous possible ways to interpolate the missing sensors position values using machine learning based solutions. But all the methods are biased with the training data that we have provided and we need very large training data base. What we propose is by exploiting the symmetric nature of animations. From this point the symmetric animations like sitting jumping walking will be taken into consideration in the discussion.

5.2 Implementation

Sensor Position	STATE 1	STATE 2	STATE 3	STATE 4
Right Hip	-40º in x axis	0º in x axis	30º in x axis	-20º in x axis
Right Knee	40º in x axis	0º in x axis	0º in x axis	60º in x axis
Left Hip	30º in x axis	-20º in x axis	-40º in x axis	0º in x axis
Left Knee	0º in x axis	60º in x axis	40º in x axis	0º in x axis

Sensor Position	STATE 1	STATE 2	STATE 3	STATE 4
Right Hip	-40º in x axis	0º in x axis	30º in x axis	-20º in x axis
Right Knee	40º in x axis	0º in x axis	0º in x axis	60º in x axis
Left Hip	30º in x axis	-20º in x axis	-40º in x axis	0º in x axis
Left Knee	0º in x axis	60º in x axis	40º in x axis	0º in x axis

Sensor Position	STATE 1	STATE 2	STATE 3	STATE 4
Right Hip	-40º in x axis	0º in x axis	30º in x axis	-20º in x axis
Right Knee	40º in x axis	0º in x axis	0º in x axis	60º in x axis
Left Hip	30º in x axis	-20º in x axis	-40º in x axis	0º in x axis
Left Knee	0º in x axis	60º in x axis	40º in x axis	0º in x axis

Sensor Position	STATE 1	STATE 2	STATE 3	STATE 4
Right Hip	-40º in x axis	0º in x axis	30º in x axis	-20º in x axis
Right Knee	40º in x axis	0º in x axis	0º in x axis	60º in x axis
Left Hip	30º in x axis	-20º in x axis	-40º in x axis	0º in x axis
Left Knee	0º in x axis	60º in x axis	40º in x axis	0º in x axis

Figure 5.2: Symmetry in State model of walking

Based on the state diagram of walking model, we can see the animations are symmetric, and hence we can draw some conclusions. In the above Figure ‘5.2 Right Hip State I and State II rotations are again seen in Left Hip State III and Stage IV respectively and Right Hip State III and State IV rotations are same as Left Hip State I and State II. We can see the symmetry in the movement of right knee and the left knee similar to the pattern observed in right hip and the left hip.

In general we get all the rotations from the sensors through web-sockets.For first few seconds, call it calibration time, we get the sensors data from Right Hip and Left Knee and store it in two separate buffers. Next give references to above buffers to Left Hip and Right Knee as seen in the state model. Similarly this solution can be employed to most of the symmetric animations and the results are evaluated in the next chapter.

Chapter 6

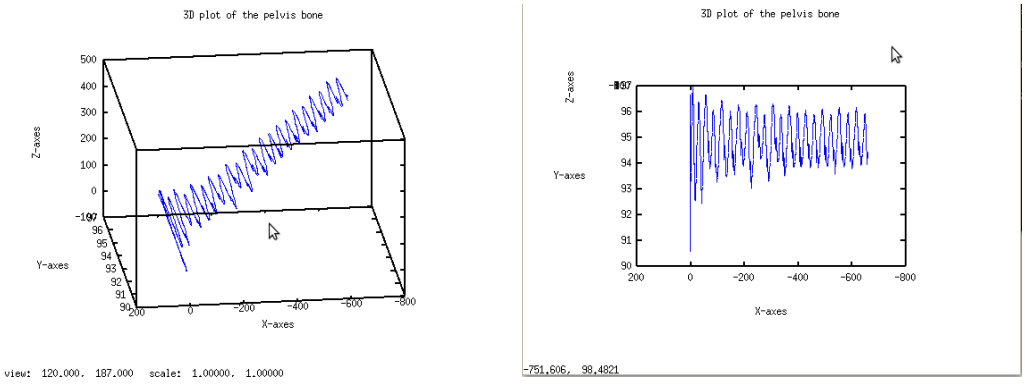
Results

"Don't speculate - benchmark." $\sim$ *DanBernstein*

6.1 Summary of Results and Contributions

Throughout this dissertation we have developed a application framework supporting motion capture sensors with physics simulations. In this final chapter we will discuss some of the key points of our work and provide concluding remarks.

6.1.1 Results Compilation



(a) Pelvis Bone Trajectory in 3d space (b) 2D plane view (i.e XY-Plane)

Figure 6.1: Trajectory of Pelvis Bone in Walking animation

In the Figure 6.1 we have captured the trajectory of pelvis bone sensor point in walking animation. In Figure 6.1b i.e 2D view of the movement of the pelvis in XY - Plane is illustrated.

For testing purpose we have developed an android application which uses mobile in built accelerometer and Gyroscope sensors and tracks the trajectory of the motion[29]. The main implementation aspects of this application is extracted from Freescale Semiconductor application note[30].

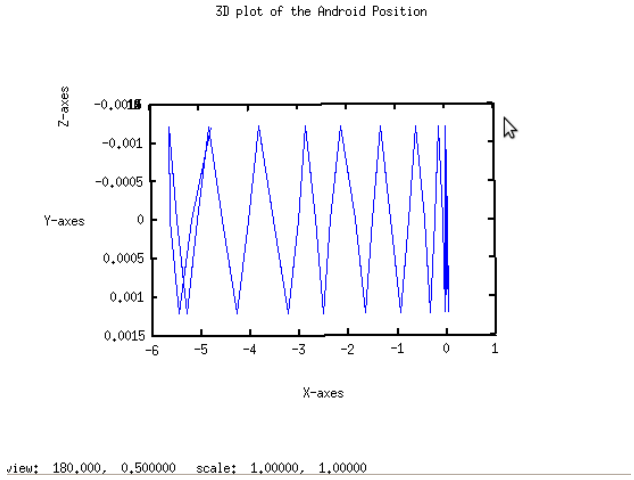
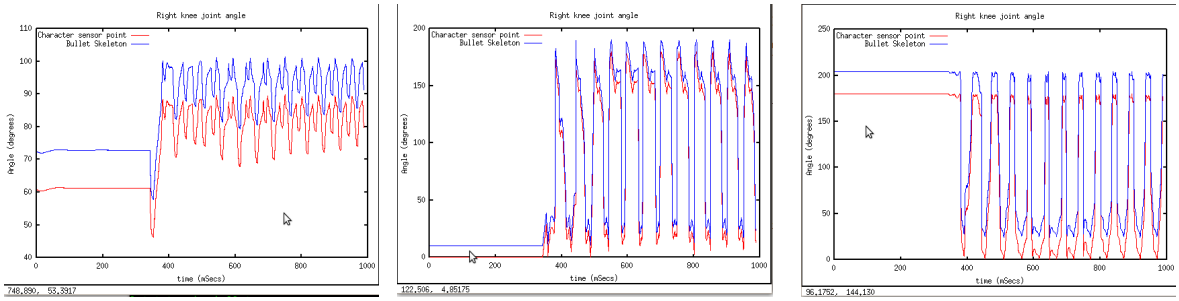


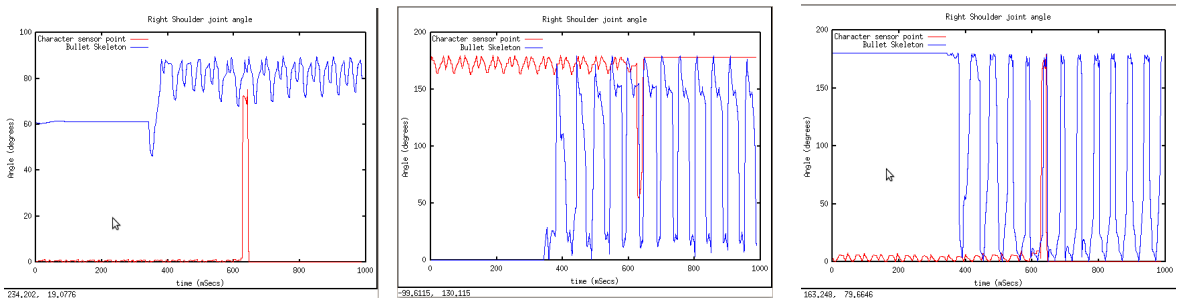
Figure 6.2: Position data from Android Mobile Phone



(a) X direction (b) Y direction (c) Z direction

Figure 6.3: The Trajectory of Right Knee angles captured while walking

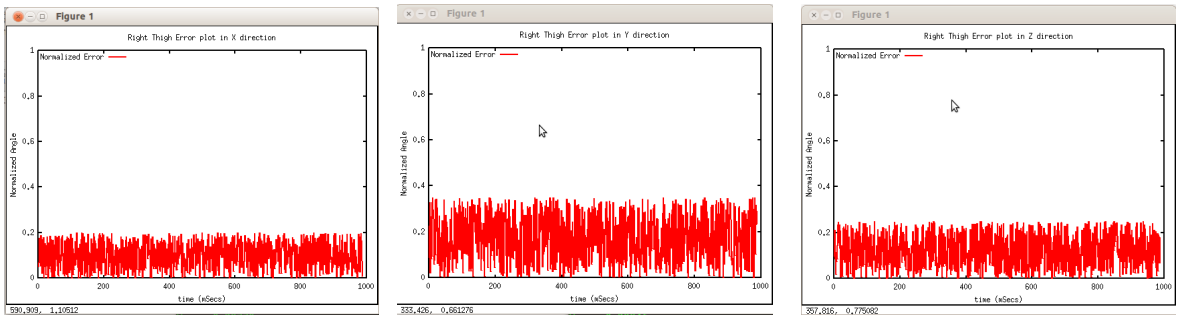
The above figures 6.3 illustrates the trajectory of Right Knee rotation in 3-D. Blue colored line tracks the bullet skeleton and the red tracks the main player sensor point and the mean errors is as follows in X direction $Mx = 11.614\%$, in Y direction $My = 10.297\%$ and in Z direction $Mz = 12.283\%$.



(a) X direction (b) Y direction (c) Z direction

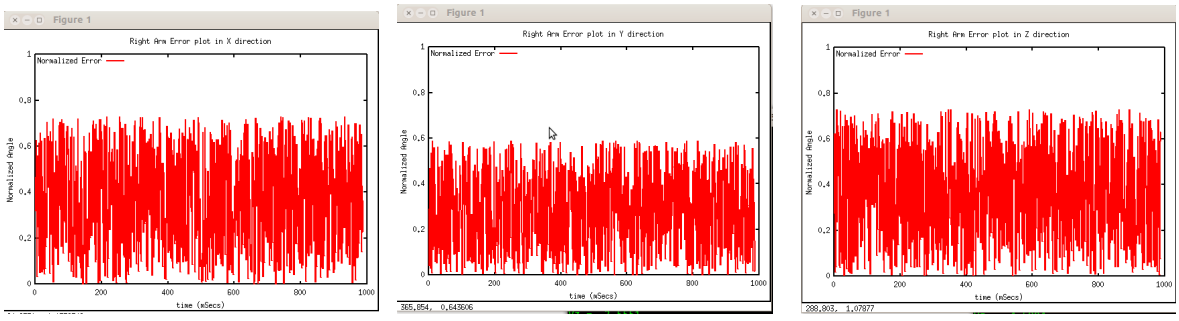
Figure 6.4: The Trajectory of Right Shoulder angles captured while walking

The above figures 6.4 illustrates the trajectory of Right Shoulder rotation in 3-D. Blue colored line tracks the bullet skeleton and the red tracks the main player sensor point and the mean errors is as follows in X direction $Mx = 72.116\%$, in Y direction $My = 58.71\%$ and in Z direction $Mz = 62.88\%$.



(a) X direction (b) Y direction (c) Z direction

Figure 6.5: The Error plot of Right Thigh captured while walking



(a) X direction (b) Y direction (c) Z direction

Figure 6.6: The Error plot of Right Shoulder captured while walking

Testing on PandaBoard and other Platfroms

The Application is tested through manual intervention. To test the demo is robust to hardware changes, It is run on different processors. Three dimensional graphics model is run on PandaBoard with ARM processor, Linux using mesa, windows with ATI Radeon. The frame rate is around 450 500 frames/s on Windows with ATI Radeon, 45-55 frames/s on LinuxPC with Nvidia NVS 295, 10 -15 frames/s on software emulation like MESA, 25-30 frames/s on Panda-Board. And The CPU utilization on PC is around 20-25% and on Panda-Board the CPU utilization is around 60-70%. This Project was also tested on Rasberry PI the CPU utilization was more than 92% and the event handling failed due to these circumstances.

6.2 Discussion

Looking back, we have addressed several key areas on motion capture based interactive character animation. We extended Physics based animations to minimize the usage of number of sensors through supporting contacts similar to humanoid systems. We exploited symmetric aspects of motion to minimize the usage of number of sensors.

6.2.1 Results

In the section 6.1.1 we have compiled all the results now let us discuss each one in detail.
In the figure 6.1b the trajectory of pelvis bone is similar to the trajectory capture by the position captured from the android mobile which is illustrated in the figure 6.2.
In the figure 6.3 the trajectory of bullet physics based Right Knee Joint and the Main Player Right knee sensor position is illustrated. and The mean error is around 10-15% , which is acceptable but in the figure 6.4 The trajectories of bullet physics and the main player is very different this results in the huge errors of around 80-100% which results in disorientation. This is illustrated precisely in the error plots of Right thigh and Right arm. In figure 6.5 the error plots of Right thigh the normalized error is around 0.15-0.25 and the error is Y - directions is little bit high since the gravitaional force acts in Y - direction in the player so it results in some disoreintation. And in

figure 6.6 the error plots of Right arm the normalized error is around 0.6-0.7 which is very high. This is due the higher degree of freedom for upper half of the body. For lower half of the body the the degree of freedowm is limited. So the Optimization proposed in this dissertation works for lower half of the body with an acceptable amount of error(not percieved by visual inspection) but for upper half of the body the proposed solution fails.

6.2.2 GPGPU Physics Synchronization

In the project initial discussions GPGPU Physics Synchronization is one the most interesting aspect that was debated with purpose. As already discussed in Bullet 3.0 an OpenCL accelerated version was proposed. With the advent of gpu accelerated physics it is very interesting to see the effect of the synchronization between the CPU & GPU on performance gain. In a game engine we'd need to synchronize game logic, A.I. etc with the results from a physics step, There was an element of suspense that this requirement might undermine the extra performance gained by simulating on the gpu.

For example in our work physics simulations are generated using the Bullet Physics library which involve the solution of equations of motion under the given physical constraints, collision detection, deformation computation etc. These algorithms are computationally expensive and are expected to slow down the frame rate of the naive 3D application considerably. Therefore, in order to obtain real-time performance with physics simulation integrated, it will be necessary to use some acceleration through hardware. Most present day PC and embedded hardware includes a Graphics Processor Unit (GPU) for rendering graphics and this is a powerful piece of computational hardware. OpenGL-ES[1] provides a programming framework for programming GPUs to accelerate computationally expensive kernels. This methodology shall be used to handle the computational burden posed by the BulletPhysics library.

The computationally intensive kernels of Bulletphysics physical simulation library have been iden-

[1]Khronos Group initiative

tified in our studies as the following:

1) Vector dot product

2) Quicksort

3) Solution of convex hull of the vertices of the avatar

4) Solution of dynamics equations with physical constraint equations (e.g. at joints)

These kernels consume around 40% of the computation time of the application and cause the frame rate to drop significantly when we are targeting embedded platform.

There are two general cases for using a physics system in a game:

1) For gameplay. That is, using forces and collisions to move objects around. The physics system manages the location and movement of game objects.

2) For visualization, things like particle systems. These are pure-rendering things that don't affect gameplay.

Handling game-play on the GPU is a bad idea performance-wise. The problem is that you need to do a CPU - GPU - CPU round-trip. The CPU needs to manage where objects are, so that AI and other systems can manipulate it. Therefore, the CPU may need to update the position of objects. The CPU also needs to read the positions of objects, as well as be notified of any collisions. So you can't just fire off some GPU "rendering" commands and then do something else; you need that information right now for AI and the like.

This forces a GPU/CPU synchronization. A GPU/CPU sync is one of the worst things you can do for performance. You want the GPU to operate as asynchronously as possible. Doing physics of type one on the GPU works against that.

Now, for second type of physics, things are little bit different. In this case, the physics is completely

subservient to the rendering of the object. It basically does physics solely to do rendering. Therefore, everything can effectively live on the GPU. For this, doing the physics on the GPU makes sense, because the AI and game engine don't effect any collisions.

But generally physics engine build concave meshes by taking many convex hulls and using constraints to effectively fuse them together, though how well this works depends on the physics system. And these complex decision making algorithms are very hard to accelerate using GPU and more important aspect is what we gain with respect to performance by porting Physics Constraints solvers to GPU is not clear. This is one of the open area where significant amount of research has to be done. Although we explored some areas since the problem statement that we are addressing is not related to the those aspects we could not move further in that direction.

6.2.3 Constraints Solver Limitation

The constraint solver in Bullet Physic library uses Euler angles,in order to avoid orthogonality problems, one axis must always be limited to +/- 90 degrees.all the proposed solutions involve allowing one axis to freely rotate and lock down all other axes, with each axis low/high limit set to a single value in the range [-PI, PI]. In essence, one need to constrain the rigid body to a single rotation matrix in a full sphere range of rotation. In Bullet, there is currently no solution for this type of problem, but it seems to be common desired feature in many physics simulation projects.

6.3 Conclusion

We have presented an approach for minimizing usage of number of sensors in Interactive Character Animation using Bullet Physics. Our key idea is to exploit the symmetry of motions and correct the predictions by utilizing Bullet Physics to construct full-body human motion in real time. We have demonstrated the effectiveness of our system by controlling a variety of human actions, including walking, sitting, and jumping, in real time. But the main shortcomings of the system is it fails in producing asymmetric motions and mainly in the upper half of the body where the degree of freedom is higher and the results are concur the same. However, our system with the sensor

orientation offset has successfully employed with 7 sensors and without Bullet Physics and the screen-shots of the same is produced in the appendix section.

6.4 Future Work

The further research shall involve finding / designing parallel algorithms mention the above section GPGPU Physics Synchronization that are computationally intensive.

1) Parallel versions of Vector dot product and Quicksort are available in literature[31]. These will need to be implemented on a GPU or customized parallel h/w.

2) There has been considerable work on finding solution of convex hull of the vertices of the avatar e.g. [32],[33]. A survey of the published techniques will need to be made and the most optimal algorithm chosen and implemented on a GPU or customized parallel h/w.

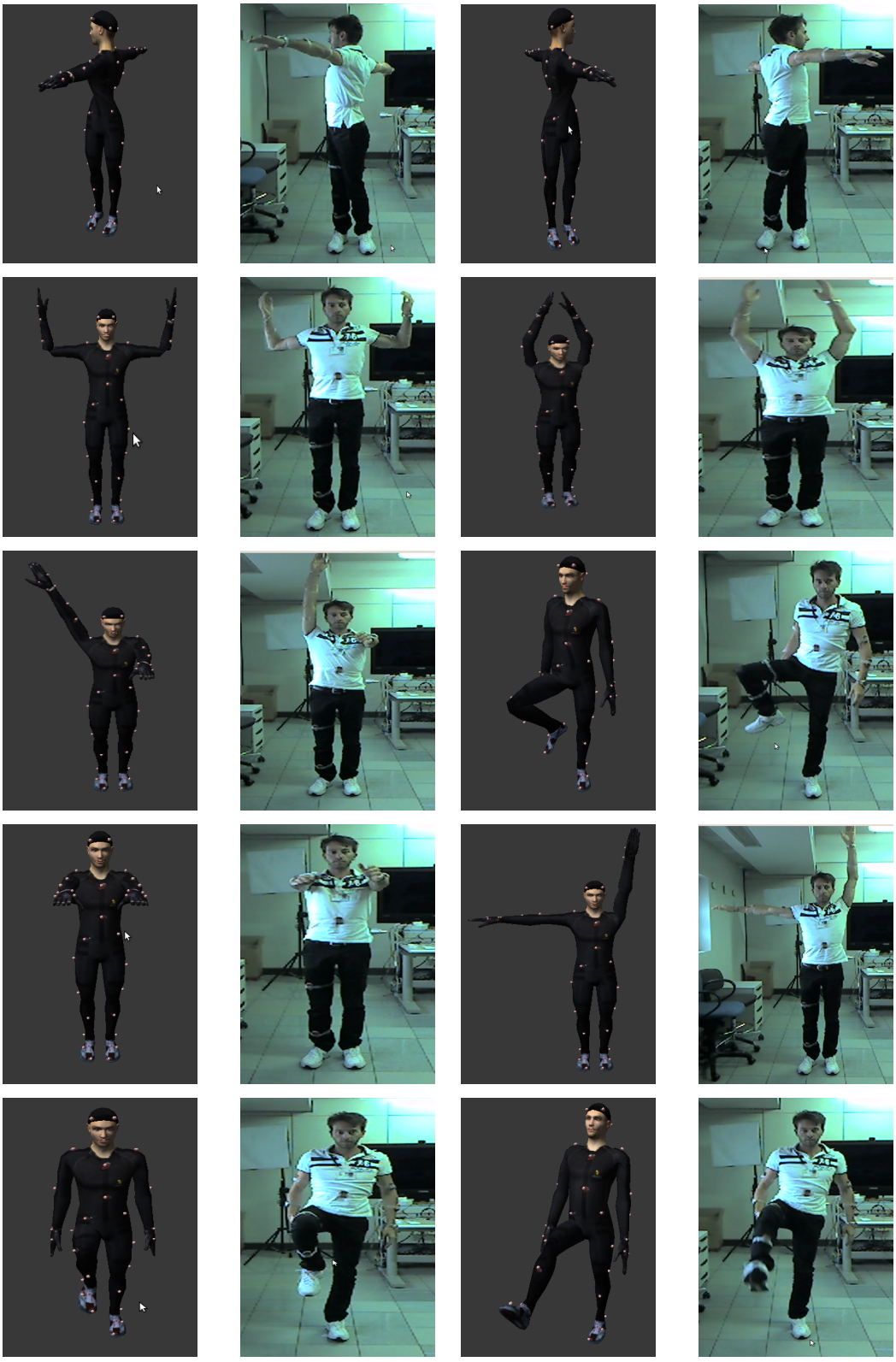
3) Solution of dynamics equations with physical constraint equations shows good potential for SIMD usage and parallelization. The parallel algorithm for this shall be designed and implemented on a GPU or customized parallel h/w.

Appendix

"The most valuable commodity I know of is information." $\sim$ *GordonGekko*

ScreenShots of Mocap suite demonstration

Initial T Pose of man and the character



ScreenShots of Mocap Suite Animation

Bibliography

[1] Michael Gleicher. Animation from observation: Motion capture and motion editing. *ACM SIGGRAPH Computer Graphics*, 33(4):51–54, 1999.

[2] ARMSTRONG W and GREEN M. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer*, 1:231–240, 1985.

[3] PEJSA T and PANDZIC I. State of the art in example- based motion synthesis for virtual characters in interactive applications. *Computer Graphics Forum*, 29:202–226, 2010.

[4] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-Based Locomotion Controllers. *ACM Transactions on Graphics*, 29(3), 2010.

[5] Franck Multon, Richard Kulpa, Ludovic Hoyet, and Taku Komura. Interactive animation of virtual humans based on motion capture data. *Computer Animation and Virtual Worlds*, 20(5-6):491–500, 2009.

[6] Daniel Vlasic, Rolf Adelsberger, Giovanni Vannucci, John Barnwell, Markus Gross, Wojciech Matusik, and Jovan Popović. Practical motion capture in everyday surroundings. In *ACM Transactions on Graphics (TOG)*, volume 26, page 35. ACM, 2007.

[7] R Budiman M Bennamoun DQ Huynh. Low cost motion capture.

[8] Zhiqiang Zhang, Zheng Wu, Jiang Chen, and Jian-Kang Wu. Ubiquitous human body motion capture using micro-sensors. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pages 1–5. IEEE, 2009.

[9] Huajun Liu, Xiaolin Wei, Jinxiang Chai, Inwoo Ha, and Taehyun Rhee. Realtime human motion control with a small number of inertial sensors. In *Symposium on Interactive 3D Graphics and Games*, pages 133–140. ACM, 2011.

[10] Alexander David Young. Use of body model constraints to improve accuracy of inertial motion capture. In *Proceedings of the 2010 International Conference on Body Sensor Networks*, BSN '10, pages 180–186, Washington, DC, USA, 2010. IEEE Computer Society.

[11] Thomas Geijtenbeek, Nicolas Pronost, Arjan Egges, and Mark H Overmars. Interactive character animation using simulated physics. *Eurographics-State of the Art Reports*, 2, 2011.

[12] Thomas Geijtenbeek, Nicolas Pronost, and A Frank van der Stappen. Simple data-driven control for simulated bipeds. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 211–219. Eurographics Association, 2012.

[13] Jan Bender, Kenny Erleben, and Jeff Trinkle. Interactive simulation of rigid body dynamics in computer graphics. In *Computer Graphics Forum*. Wiley Online Library, 2013.

[14] Rong Zhu and Zhaoying Zhou. A real-time articulated human motion tracking using tri-axis inertial/magnetic sensors package. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 12(2):295–302, 2004.

[15] Eric R Bachmann. Inertial and magnetic tracking of limb segment orientation for inserting humans into synthetic environments. Technical report, DTIC Document, 2000.

[16] St

aale Andreas van Dorp Skogstad, Kristian Nymoen, and Mats Erling H

ovin. Comparing inertial and optical mocap technologies for synthesis control. In *Proceedings of Sound and Music Computing*, pages 421–426, Padova, Italy, 2011.

[17] Daniel Roetenberg. *Inertial and magnetic sensing of human motion*. PhD thesis, Enschede, May 2006.

[18] The Moving Picture Experts Group. Part 25 of mpeg-4 standard.

[19] Khronos Group. Digital asset and fx exchange schema.

[20] Assimp. Open asset import library.

[21] Bullet Physics Library. Real-time physics simulation.

[22] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple biped locomotion control. In *ACM Transactions on Graphics (TOG)*, volume 26, page 105. ACM, 2007.

[23] Valgrind. Gpl'd system for debugging and profiling linux programs.

[24] PARVEEZ IQBAL. Porting and integration of bulletphysics physical simulation in to 3dgc through gpu using opengl, June 2013.

[25] Bullet Physics org. mediawiki examples for using constraints.

[26] Claude Lacoursire. *Ghosts and machines : regularized variational methods for interactive simulations of multibodies with dry frictional contacts*. PhD thesis, Ume University, Computing Science, 2007.

[27] Luis Sentis. *Synthesis and control of whole-body behaviors in humanoid systems*. PhD thesis, Citeseer, 2007.

[28] Ling Bao and StephenS. Intille. Activity recognition from user-annotated acceleration data. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 2004.

[29] Bogdan Muset and Simina Emerich. Distance measuring using acceleromreter and gyroscope sensors. *Carpathian Journal of Electronic & Computer Engineering*, 5(1), 2012.

[30] Kurt Seifert and Oscar Camacho. Implementing positioning algorithms using accelerometers. *Freescale Semiconductor*, 2007.

[31] Michael J Quinn. *Parallel Programming*, volume 526. TMH CSE, 2003.

[32] SG Akl. Optimal parallel algorithms for computing convex hulls and for sorting. *Computing*, 33(1):1–11, 1984.

[33] D Srikanth, Kishore Kothapalli, R Govindarajulu, and P Narayanan. Parallelizing two dimensional convex hull on nvidia gpu and cell be. In *International conference on high performance computing (HiPC)*, pages 1–5, 2009.