DigiPen Institute of Technology

INFUSING LIFE INTO AN ARTICULATED BODY USING SPACE TIME CONSTRAINTS

BY
Amar Chitimalli
Masters of Science in Computer Science

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the graduate studies program
of DigiPen Institute of Technology
Redmond, Washington
United States of America

Spring
2008

DigiPen Institute of Technology

Thesis Advisor: Dr. Xin Li
DIGIPEN INSTITUTE OF TECHNOLOGY

GRADUATE STUDY PROGRAM

DEFENSE OF THESIS

THE UNDERSIGNED VERIFY THAT THE FINAL ORAL DEFENSE OF THE

MASTER OF SCIENCE THESIS OF AMAR CHITIMALLI

HAS BEEN SUCCESSFULLY COMPLETED ON 28TH MARCH 2008

TITLE OF THESIS: INFUSING LIFE INTO AN ARTICULATED BODY USING SPACE TIME CONSTRAINTS

MAJOR FILED OF STUDY: COMPUTER SCIENCE

COMMITTEE:

Dr. Xin Li, Chair	Charles Duba
Gary Herron	Michael Moore

APPROVED :

Xin Li 3/28/2008 Graduate Program Director	Matt Klassen 3/28/2008 Dean of Faculty
Samir AbouSamra 3/28/2008 CHAIR OF COMPUTER SCIENCE	Claude Comair 3/28/2008 PRESIDENT OF DIGIPEN DEPARTMENT

DIGIPEN INSTITUTE OF TECHNOLOGY

PROGRAM OF MASTER’S DEGREE

THESIS APPROVAL

DATE: 28ST MARCH 2008

BASED ON THE CANDIDATE’S SUCCESSFUL ORAL DEFENSE, IT IS RECOMMENDED THAT THE THESIS PREPARED BY

Amar Chitimalli

ENTITLED: INFUSING LIFE INTO AN ARTICULATED BODY USING SPACE TIME CONSTRAINTS

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE FROM THE PROGRAM OF MASTER’S DEGREE AT DIGIPEN INSTITUTE OF TECHNOLOGY.

Dr Xin Li
Thesis Advisory Committee Chair

Xin Li
Director of Graduate Study Program

Matt Klassen
Dean of Faculty

Table of Contents

List of Important Figures

Acknowledgements

This thesis would certainly not have been possible without the support of many people. My sincerest gratitude goes to all of them. I would specially like to thank my advisor Prof. Xin Li who has always supported and pushed me to go ahead and try things until it worked. It has been a great experience working with him.
I am working at a game development company called BigFish Game, the company and the team has been of great support and help is providing all the resources and information required. They have always understood my passion towards my thesis and supported to go ahead even while we had great deliverable to be met in the team.

JME Physics team has bee a great resource; they have a forum that people help and share the knowledge and solve your barriers. This forum has been a great help.

Finally I would like to thank my family, friends for believing in me and supporting me all the way to reach my career prospects. I dedicate my work to my father and mother.

Abstract

Animation has come a long way starting from traditional hand drawn frames to synthesizing the motion as physically realistic as possible, by adapting and evolving with the research and knowledge from areas like robotics, biomechanics, space-time optimization and kinesiology. Animation studios have always been trying to reduce the gap between the animation generated artificially and the natural motion of a living character in a real world.

This paper talks about implemented a simple prototype for animating an articulated character using only its physical characteristics and constraints imposed by the environment and itself, where the user can impose additional constraints both on space and time of the characters animation. The over all idea is to make the animation as interactive and provide the user a physics simulated environment with an easy use interface to compose and control the character motion with out any traditional key framing. There is no interpolation involved and the same animation sequence can be generated uniquely by changing the physical characteristics of the character.

This paper introduce a novel joint correction algorithm, where the joints of the character are corrected in each physics time steps, by calculating the amount and direction of angular velocity to be applied to each joint. The algorithm takes into account the centre of mass of the entire system, the physical mass of each link, the time and poses constraints to generate the animation sequence.

Introduction to study of computer animation

Motion of a creature that can move its body parts are controlled by its brain, the nervous system controls and prepares the creature to make the require moves so that the desired motion and pose is obtained. The nervous system is able to control this motion based on some of the inputs it gets from the environment in the form of visual signals and many other forms. With out essential input and proper goal of the motion, the complex nervous system also fails to get into the desired motion and pose.

Also the system already has the very essential physical information of itself like mass, density, stiffness, constraints and DOF's of each joint and bones in its body. Apart from that, most of the creatures have memory to store and learn the experiences, and reuse them my making minor corrections in the initial conditions based on the current inputs and external factors its getting from its senses. All these factors lead a character to manipulate itself and make required changes to its body before, during and after the course of the motion to reach the required position and target, as long as it is physically possible.

For years scientists have been studying, collecting and applying the learnt data so that they can build a system that can control the body in the same way a creatures can manipulate moves based on the trained data, inputs to achieve the required motion. This has lead to a tremendous research in the areas of robotics, biomechanics to study the biology of the body and mix it up with the mechanical objects that they can create and simulate the control of the creatures control on its body parts to achieve the required motion.

The movie, entertainment and gaming industry has evolved as the computers become powerful, affordable. This has lead into solving and simulating these problems of motion control by using the knowledge from Biomechanics, Kinesiology and Robotics. As the art, imagination, computing and all this research joint together and provided tools for the artists to express their ideas of motion of a character, we started seeing something called computer generated animation.

Generating computer animation with out considering physics only artistic and does not deal with all the complexities of environment and the body/character properties. A cartoon character looks funny and interesting; here the artist does not worry about making the character obey all the physic properties, joints and their degree of freedom, and the environment the character is in.

But when it comes to applications that have a character with bones with DOF's, mass and muscle forces defined, generating a physically realistic animation that should appeal our eyes is not in the hands of the artist alone, it need lot of parameters to be considered. How hard the artist try they will not be able to make realistic. The more realistic and physically possible animation we want the more complex and time consuming the task becomes.

Current animation systems

Animation, once considered merely a source of entertainment, has evolved into a powerful medium of communication in our daily life. The overwhelming need for animation in education, science visualization, architecture, medicine, and even forensics has created a great environment that nourishes research in different disciplines. Despite the prevalence of animation, the process of creating animation is difficult and labor intensive. Character animation poses a still more challenging problem. Unlike passive rigid body motion, animals use muscles to exert self-propelling forces to achieve certain goals when locomotion is undertaken. Moreover, the animator has to portray the style and expression that bring characters alive. Finally, even minor glitches and oddities in the synthetic motions are easily detected by the scrutiny of sensitive human perception of natural motion.

Most character animations appearing in movies or video games products were generated painstakingly by hand. The Key frame technique enables animators to choreograph and build an animation by arranging characters and taking snapshots at key moments during a sequence of movements. The advent of computers alleviates part of the manual process by automatically generating in-between frames in a motion sequence. Since it is still up to the artist to convincingly portray the expressiveness of the movement from one key frame to the next one, computers are not of much use when dealing with complex and intricate motions.

Data-driven methods leverage the high fidelity of motion capture data to produce believable character animations with expressive details by learning a statistical model from a large dataset of existing motion sequences acquired from the real world, data-driven methods can synthesize new motions that are similar to the training motion sequences. Since these methods do not explicitly model physics, the output is limited to direct modifications to the existing motions. If the desired motion is drastically six different from the existing motions, a large number of new motions that are similar to the desired motion need to be acquired. Consequently, extremely large motion datasets may be required for general-purpose motion synthesis.

In contrast to data-driven methods, the space-time constraints framework casts motion synthesis as a variation optimization problem of minimizing some physical measure of energy. In addition to producing physically plausible motion, space-time constraints provide the user an intuitive way to control the output motion through constraints. However, because many aspects of the real-life physics are abstracted away from the model, the optimization tends to produce reasonable results only for high energy motions (jumping, diving, acrobatics, etc.) which can be well defined by straightforward Newtonian dynamics. Low energy motions (walking, reaching, etc.), in contrast, contain more stylistic variations which have not been described in any dynamic models. When synthesizing highly dynamic motion, these methods use a small set of simplified physical rules to ensure the momentum of the motion satisfying Newtonian Laws. For synthesis of low energy motions, the model takes into account the relative strength of muscles, impedance, and neutral position parameters of passive structures around each joint.

Dynamics of Physics

Centre of Mass

In physics, the center of mass of a system of particles is a specific point at which, for many purposes, the system's mass behaves as if it were concentrated. The center of mass is a function only of the positions and masses of the particles that comprise the system. In the case of a rigid body, the position of its center of mass is fixed in relation to the object. In the context of an entirely uniform gravitational field, the center of mass is often called the center of gravity — the point where gravity can be said to act. [See Fig -1]

- Mass $\boldsymbol{m_k}$
- Center of mass $\boldsymbol{C_k(t)}$
- Velocity $\boldsymbol{v_k(t)}$
- Inertia Tensor $\boldsymbol{J_k}$
- Rotation Matrix $\boldsymbol{R_k(t)}$
- Angular Velocity $\mathbf{W}_{\boldsymbol{k}}\boldsymbol{(t)}$

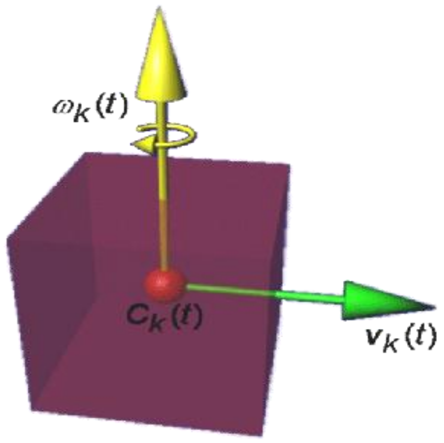


Figure -1

Center of Mass of a system of R particles is defined as the average of their positions (**r**i) weighted by their masses **m**i

$$\mathbf{R} = \frac{\sum m_i \mathbf{r}_i}{\sum m_i}$$

For a continuous distribution with mass density p(r) and total mass M, the sum becomes an integral:

$$\mathbf{R} = \frac{1}{M}\int \mathbf{r}\; dm = \frac{1}{M}\int \rho(\mathbf{r})\,\mathbf{r}\; dV = \frac{\int \rho(\mathbf{r})\,\mathbf{r}\; dV}{\int \rho(\mathbf{r})\; dV}$$

Rotation and Centers of Gravity

The center of mass is often called the center of gravity because any uniform gravitational field g acts on a system as if the mass M of the system were concentrated at the center of mass R. This is seen in at least two ways:

The gravitational potential energy of a system is equal to the potential energy of a point particle having the same mass M located at R.
The gravitational torque on a system equals the torque of a force Mg acting at R:

$$\mathbf{R} \times M\mathbf{g} = \sum_i m_i \mathbf{r}_i \times \mathbf{g}.$$

If the gravitational field acting on a body is not uniform, then the center of mass does not necessarily exhibit these convenient properties concerning gravity

Center of Momentum

The "center of momentum" of a system is not a point in space, but rather a particular inertial frame in which the center of mass is at rest, and the total linear momentum of the system is zero. Thus, the term is generally used in conjunction with the term "frame", as in "center of momentum frame" or COM frame. The center of mass frame, a less-preferred term for the same concept, refers to the fact that in the center of momentum frame, the velocity of the center of mass is zero. However, the center of mass of an object or system of objects is defined as point, independent of inertial frames.

Moment of Inertia tensor

Moment of inertia, also called mass moment of inertia or the angular mass, (SI units kg m^2, Former British units slug ft^2), is the rotational analog of mass. That is, it is the inertia of a rigid rotating body with respect to its rotation. The moment of inertia plays much the same role in rotational dynamics as mass does in basic dynamics, determining the relationship between angular momentum and angular velocity, torque and angular acceleration, and several other quantities. The symbols I and sometimes J are usually used to refer to the moment of inertia.

Moment of inertia was introduced by Euler in his book a *Theoria motus corporum solidorum seu rigidorum* in 1730. In this book, he discussed at length moment of inertia and many concepts, such as principal axis of inertia, related to the moment of inertia

Moment of Inertia Tensor

For the same object, different axes of rotation will have different moments of inertia about those axes. In general, the moments of inertia are not equal unless the object is symmetric about all axes. The moment of inertia tensor is a convenient way to summarize all moments of inertia of an object with one quantity. It may be calculated with respect to any point in space, although for practical purposes the center of mass is most commonly used.

Definition

For a rigid object of N point masses mk, the moment of inertia tensor is given by

$$\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}.$$ Its components defined by $$I_{ij} \overset{\text{def}}{=} \sum_{k=1}^{N} m_k (r_k^2 \delta_{ij} - r_{ki} r_{kj})$$

Where i, j equal 1, 2, or 3 for x, y, and z, respectively, rk is the distance of mass k from the point about which the tensor is calculated, and dij is the Kronecker delta.

Here Ixx denotes the moment of inertia around the x-axis when the objects are rotated around the x-axis, Ixy denotes the moment of inertia around the y-axis when the objects are rotated around the x-axis, and so on…

Principal Moments of Inertia

Since the moment of inertia tensor is real and symmetric, it is possible to find a Cartesian coordinate system in which it is diagonal, having the form

$$\mathbf{I} = \begin{bmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_3 \end{bmatrix}$$

Where the coordinate axes are called the principal axes and the constants I1, I2 and I3 are called the principal moments of inertia. The unit vectors along the principal axes are usually denoted as (e1, e2, e3)

Using the tensor I, the kinetic energy can be written as a quadratic form

$$T = \frac{1}{2}\boldsymbol{\omega}^T \mathbf{I} \boldsymbol{\omega} = \frac{1}{2} I_1 \omega_1^2 + \frac{1}{2} I_2 \omega_2^2 + \frac{1}{2} I_3 \omega_3^2$$

Angular momentum can be written as a product

$$\mathbf{L} = \mathbf{I}\,\boldsymbol{\omega} = \omega_1 I_1 \mathbf{e}_1 + \omega_2 I_2 \mathbf{e}_2 + \omega_3 I_3 \mathbf{e}_3$$

Where W is the angular velocity of the entire system, W1, W2… are angular velocities of each rigid body part of the system

Taken together, one can express the rotational kinetic energy in terms of the angular momentum (L1, L2, and L3) in the principal axis frame as

$$T = \frac{L_1^2}{2I_1} + \frac{L_2^2}{2I_2} + \frac{L_3^2}{2I_3}.$$

The rotational kinetic energy and the angular momentum are constants of the motion (conserved quantities) in the absence of an overall torque. The angular velocity W is not constant.

Source:

http://en.wikipedia.org/wiki/Moment_of_inertia#Principal_moments_of_inertia

Rigid Body Linear Momentum:

The equation for particle linear momentum is $\frac{\mathrm{d}(mv)}{\mathrm{d}t} = \sum_{i=1}^{N} f_i$

- m is the particle's mass
- v is the particle's velocity
- f_i is one of the N forces acting on the particle

Assuming constant mass, this reduces to $m\frac{\mathrm{d}v}{\mathrm{d}t} = \sum_{i=1}^{N} f_i.$

To generalize assume a body of finite mass and size is composed of such particles. There exist internal forces, acting between any two particles, and external forces, acting only on the outside of the mass. Each particle has:

- a mass dm
- a position vector r

Thus, the linear momentum equation of any given particle would look like this:

$$\mathrm{d}m\frac{\mathrm{d}^2 r}{\mathrm{d}t^2} = \sum_{i=1}^{M} f_{i,\text{internal}} + \sum_{j=1}^{N} f_{j,\text{external}}.$$

If the equation for each particle were added together, the internal forces would cancel out, since by Newton's third law, any such force would have opposite magnitudes on the two particles. Also, the left side would become an integral over the entire body, and the second derivative operator could come out of the integral, leaving

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\int r\,\mathrm{d}m = \sum_{j=1}^{N} f_{j,\text{external}}.$$

Letting M be the total mass, the left side can be multiplied and divided by M without changing the validity:

$$M\frac{\mathrm{d}^2\frac{\int r\,\mathrm{d}m}{M}}{\mathrm{d}t^2} = \sum_{j=1}^{N} f_{j,\mathrm{external}}$$

However, $\frac{\int r\,\mathrm{d}m}{M}$ is the formula for the position of center of mass. Denoting this by rcm, the equation reduces to $M\frac{\mathrm{d}^2 r_{cm}}{\mathrm{d}t^2} = \sum_{j=1}^{N} f_{j,\mathrm{external}}.$

Thus, linear momentum equations can be extended to rigid bodies by denoting that they describe the motion of the center of mass of the body.

Source: http://en.wikipedia.org/wiki/Momentum

Angular Momentum and Torque

Similarly, the angular momentum **L** for a system of particles with linear momentum pi and distances ri from the rotation axis is defined

$$\mathbf{L} = \sum_{i=1}^{N} \mathbf{r}_i \times \mathbf{p}_i = \sum_{i=1}^{N} m_i \mathbf{r}_i \times \mathbf{v}_i$$

For a rigid body rotating with angular velocity W about the rotation axis n (a unit vector), the velocity Vi vector may be written as a vector cross product

$$\mathbf{v}_i = \omega \hat{\mathbf{n}} \times \mathbf{r}_i \overset{\text{def}}{=} \boldsymbol{\omega} \times \mathbf{r}_i$$

Substituting the formula for Vi into the definition of L yields

$$\mathbf{L} = \sum_{i=1}^{N} m_i \mathbf{r}_i \times (\boldsymbol{\omega} \times \mathbf{r}_i) = \boldsymbol{\omega} \sum_{i=1}^{N} m_i r_i^2 = I\omega\hat{\mathbf{n}}$$

Where we have introduced the special case that the position vectors of all particles are perpendicular to the rotation axis $\boldsymbol{\omega} \cdot \mathbf{r}_i = 0$.

The torque N is defined as the rate of change of the angular momentum L

$$\mathbf{N} \overset{\text{def}}{=} \frac{d\mathbf{L}}{dt}$$

If I is constant (because the inertia tensor is the identity, because we work in the intrinsically frame, or because the torque is driving the rotation around the same axis so that I is not changing) then we may write

$\mathbf{N} \overset{\text{def}}{=} I\frac{d\omega}{dt}\hat{\mathbf{n}} = I\alpha\hat{\mathbf{n}}$ Where alpha is called the angular acceleration (or rotational acceleration) about the rotation axis

Source:

http://en.wikipedia.org/wiki/Rigid_body_dynamics#Angular_momentum_and_torque

Articulated Character and Degree of Freedom (DOF)

An articulated figure or character is made up of links and joints. The different links are connected by joints which have some degree for freedom (DOF). In gaming terminology these articulated characters are referred as rag dolls. Rag doll are being used in many games to simulate natural motion where physics is involved with the character.

__Link__ can be thought of as a rod, which cannot change its shape and nor length.
__Joint__ can be thought of as a connection between two neighboring links.
A joint has several Degrees of freedom, i.e., it might rotate around one, two, or three axes. Or it might translate along one two or three axes.

Links and joints are numbered from 0,…,N, and an articulated figure always starts with joint0, which is fixed at some stationary base coordinate system. The numbering of the Links is very important in an articulated system. A joint inside an articulated system say joint (i) connects link(i-1) and link(i), where link(i-1) is closer to the base. Figure- 2 shows some of the examples of articulated bodies.

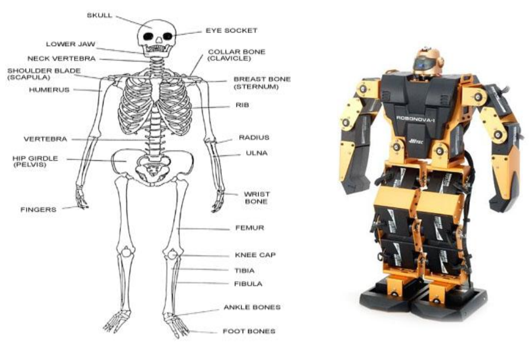


Figure -2

Types of Joints:

1) ***Revolute Joint:*** A joint that can be rotated around one or more axes
    a. They can have up to 3 degrees of freedom, that is rotate in all the three axes [x,y,z]
    b. Example: Ball and socket
2) ***Prismatic Joint:*** A joint that can translate along one or more axis
    a. They can have up to 3 degrees of freedom, that is translate in all the three axes [x,y,z]
    b. Example: Hinge

Ball and Socket Joint:

A ball joint connects two bodies in one common point The bodies can rotate around this common point but they can't move apart in this point (anchor). See Fig -3

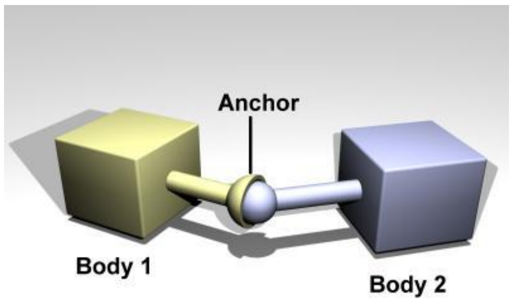


Figure- 3

Hinge Joint:

Hinge Joint eliminates all translational degrees of freedom and two rotational ones. The linked bodies are allowed to rotate around one axis and they can't move relative to each

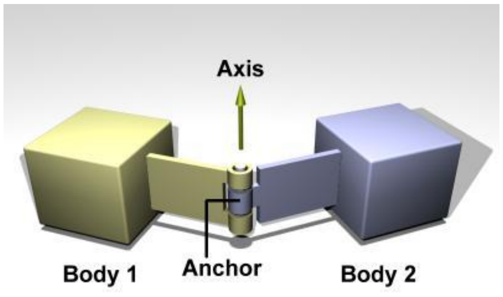


Figure- 4

Slider Joint:

A slider joint eliminates all the rotation of freedom two translational ones. The linked bodies are allowed to translate on axes relative to each other

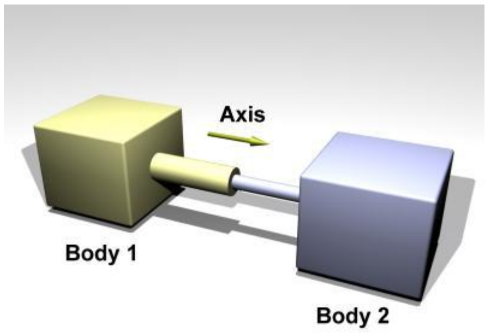


Figure - 5

Universal Joint:

Eliminates all translational and one rotational degrees of freedom. The linked bodies are allowed to rotate around two different axes and they can't move relative to each

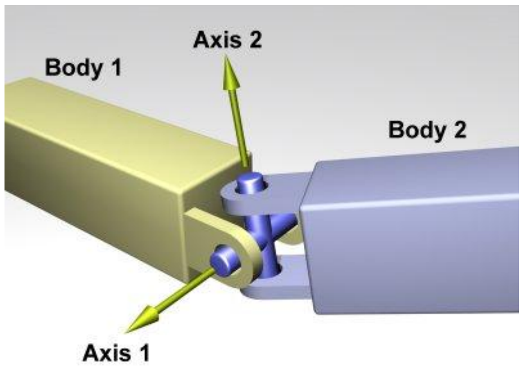


Figure - 6

http://www.ode.org/ode-latest-userguide.html

Hinge Slider Joint:

Eliminates two translational and two rotational degrees of freedom. The linked bodies are allowed to rotate around one axis and they move relative to each on a slider

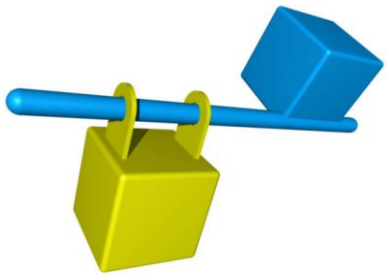


Figure - 7

Motor Hinge Joint:

A Motor Hinge Joint is a hinge joint with a servo motor . The motor is controlled with a PID controller

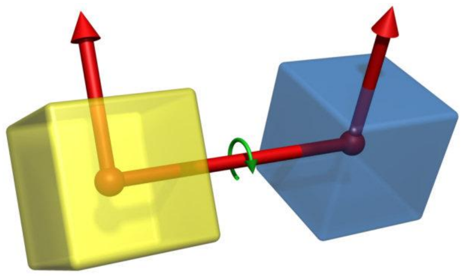


Figure - 8

Motor Slider Joint:

A Motor Slider Joint is a slider joint with a servo motor. The motor is controlled with a PID controller

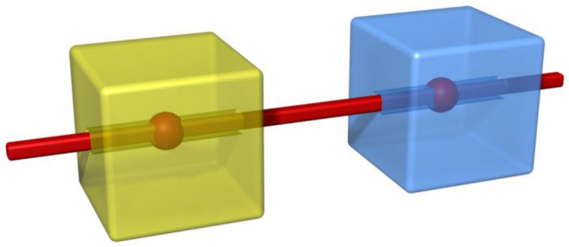


Figure - 9

http://www.impulse-based.de/manual.html

The State Vector:

A state vector is a vector that represents the state of a physical body. Mostly stare vectors are used in representing the Degree of Freedom (DOF) of joints in a particular character.

To represent all kinds of mentioned above we will need 3 elements for translation and 3 for rotation. Hence a vector with a size of 6 is good to represent most of joint configuration. Let **S** be a state vector and it can be defined by:

S= {Xpos, Ypos , Zpos ,Yaw, Pitch ,Roll}

Most of the animation sequences have the trajectories for all the joints in the character tracked for the entire animation sequence, which is basically huge array of state vectors tracked for the entire time of animation.

Survey of previous work in Physics based animation

Space time Constraints – Andrew Witkin and Michale Kass

Introduction:

Space time constraints were first introduced in the year 1988, by Andrew Witkin and Michale Kass. This was the first attempt to animate a character using only physics properties of the characters. The goal of their new method was to eliminate the dependency on the traditional animation techniques like key frame interpolation provided by the artist. This was the first time where some one tried to eliminate static key framing and come up with a method that can generate the animation as realistic as possible.

The two common methods are simulation methods and constraint force methods to generate animation but, Simulation methods solve initial value problem, they are not good for problems like two-point boundary problems. They are good for problems with the starting and the ending conditions are available, where as constraint force methods permit parts of the character to move along predefined key framed trajectories, but cannot be used to define the trajectories itself. Finding the optimal trajectory of the parts is the central problem of character animation.

The Space time optimization introduced in this paper solves for characters motion and time-varying muscle forces over the entire time interval of animation. The model can be extended through time as well as space they call this formulation space time constraints.

- Space-time formulation does the following things
    - Imposes constraints throughout the time course of the motion
    - Constraints on positions and velocities directly encode the *goals* of the motion
    - Constraints limiting muscle forces or preventing interpenetration define properties of the physical situation
    - Newtonian physics provides a constraint relating the force and position functions that must hold at every instant in time.
    - Solving the above constrained optimization problem will result in a physically valid motion

This paper explains the formulation of a motion of a simple particle with some mass and comes up with the equation of motion with respect to time and discusses its extension to a complex model and the numerical solutions available to optimize the solution.

The Numerical solution this paper adapts to use the Sequential quadratic programming [SQP] which involves the derivation if the entire systems equation of motion in complex tensors. This involves the computation of large sets of Jacobian and Hessians for solving the linear system during runtime.

The paper also discusses the extension to complex model where the model they chose is Luxo Lamp. They formulate for the Kinetic energy T. The KE of the entire system is the sum of the KE of each part of lamp.

Over all this method is very complex and involves solving lot of tensor equations and also deals huge sparse matrices of Jacobian and Hessians.

Some of the results that they discuss involve

Simple jumping of the Luxo Lamp

Variation Jumps

Ski Jumping

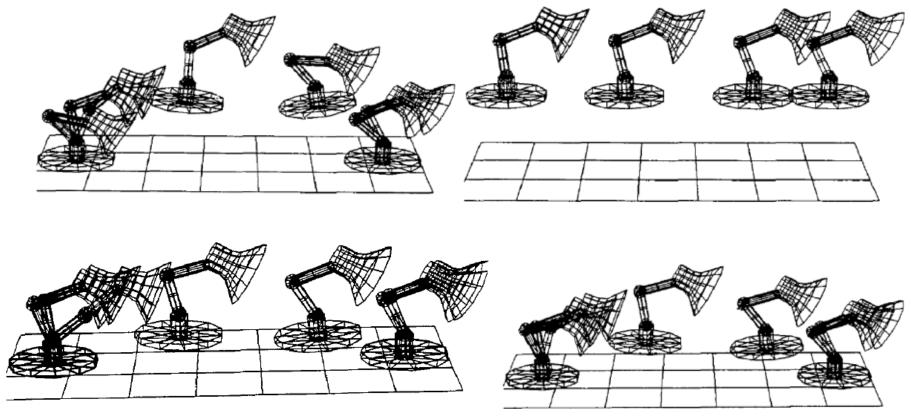


Figure – 10

- Advantages
    - They do much of work only a skilled animator can do
    - No high detail of Key framing is involved
    - Motions can be sketched out as start here and stop
    - New motion control by attributes like force and mass
- Draw Backs
    - Needs formulation of equation of motion for every new body
    - Involves "SQP" Sequential Quadratic Programming where large sets of Jacobian and Hessians are solved offline
    - Involves complex Tensor expressions to be solved
    - Needs Offline computation of these tensors and Jacobian
    - Hardly interactive

Synthesis of Complex animation from simple animation

C. Karen Liu Zoran Popovi´c University of Washington

Abstract **:**

This was paper has a different approach for the generating a realistic animation by using the so called bio mechanics and the momentum constraints

Key features of this method can be put as following

- Method for rapid prototyping of realistic character motion
- Solve from a simple animation provided by the animator
- Describe a method to automatically find constraints from the input motion
- Uses small set of linear and angular momentum constraints
- Avoid the complexities of computing muscle forces

Overview

- System transforms simple animations into realistic character motion by applying laws of physics and the biomechanics domain knowledge
- Input to the system
    - An articulated character with its mass distribution
    - Arbitrary character animation values of joint angles at each frame
- The motion synthesis problem is framed into a space time optimization
- The unknowns to this problem
    - Values of joint angles at each frame
    - Parameters that determine the behavior of angular and linear momentum

Four key stages of this method

- Constraint and stage detection
- Transition pose generation
- Momentum control
- Objective function generation

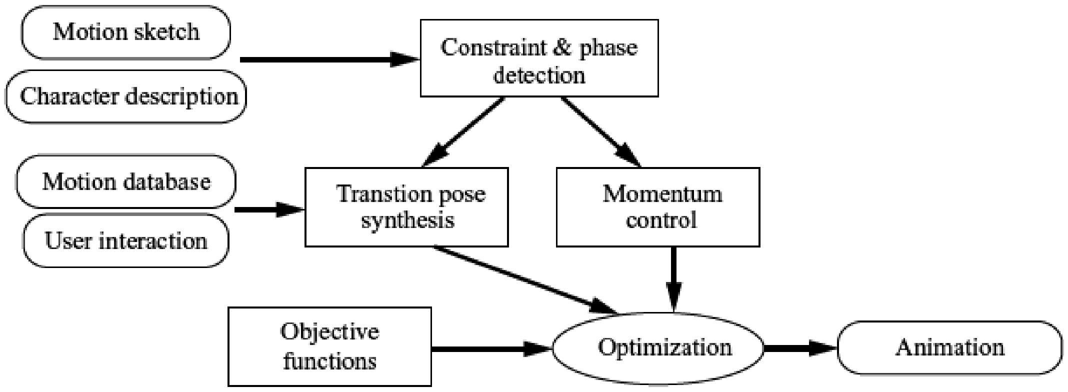


Figure - 11

Putting it all together

- All constraints and the objective function fit naturally within the space time framework
- The unknowns of our system **Q** are the character DOF's **q** *i* for each time *i* and the control point vectors for all constrained-stage momentum curves **q** *j m*
- The optimization needs to enforce three types of constraints:
    - Environment constraints (C*e*)
    - Transition pose constraints (C*p*)
    - Momentum constraints (C*m*)
- The space time constraints formulation finds the unknowns **Q** that minimize the objective function while satisfying all the constraints

<u>Advantages and disadvantages:</u>

- Needs Offline computation of solving all the constraints
- Not real time, but good for quick prototype
- Good for high energy, as governed by Newtonian physics
- Do not work for low energetic motion

My Approach for Infusing Life into Articulated Character

This implementation mainly consists of an articulated Luxo Lamp with three joints connecting between four rigid links. The goal of the implementation is to make this articulated character animate using the knowledge gained from the above papers. The character is places in a fully simulated physics environment with mass values and DOF's values assigned to each of its body parts. Only by using Bio Mechanics knowledge and simple physics dynamics discussed above we try to make the Luxo move for pose to pose smoothly applying required angular velocities to each joint iteratively in the simulation step. The idea is to eliminate the traditional method of key framing the poses and then using some kind of interpolating techniques to simulate the motion.

Since we are having the Luxo in a totally physics simulated environment the animation is physically realistic and most importantly it's real-time and hence interactive. The Lamp reacts to the user's inputs in real time and moves form one pose to other pose. The user has the ability to add a new pose from the interface provided and also modify the joint constraints as the simulation is in progress. Apart from the user can specify a series of poses and the Lamp start moving form one pose to the other pose, which make the user to create interesting moves and animation.

Kinematics Vs Dynamic approach

There can be two ways of approaching this problem of moving the joint angles to the desired angles, while in the process we achieve the animated effect of the skeleton of the articulated body.

Kinematics Approach:

Kinematics is the study of motion without regard to forces that causes the motion.
Kinematics analysis is a simpler task than dynamic analysis and is adequate for many applications involving moving parts. Kinematics simulations show the physical positions of all the parts in an assembly with respect to the time as it goes through a cycle. This approach is useful for simulating steady-state motion (with no acceleration), as well as for evaluating motion for interference purposes, such as assembly sequences of complex mechanical system. Some of the kinematics packages provide "reaction forces," forces that result from the motion.

Dynamic Approach:

Dynamics is the study of motions that result from forces.
Dynamic simulation is more complex because the problem needs to be further defined and more data is needed to account for the forces. Dynamics are often required to accurately simulate the actual motion of a mechanical system. Generally, kinematics simulations help evaluate form, while dynamic simulations assists in analyzing function.

Traditionally, kinematics and dynamics have followed the classic analysis software method of preprocessing (preparing the data), solving (running the solution algorithms, which involve the solution of simultaneous equations), and post processing (analyzing the results). Even though today's programs are much more interactive, most programs follow this basic process since it is a logical way to solve the problem. Most solvers are available as independent software programs.

The basic output of generating motion are numerous, including animation, detecting interference, trace functions, basic motion data, and plots and graphs. Animated motions are the classic output of simple kinematics analyses. Initially, the designer uses simple animation as a visual evaluation of motion to see if it is what is desired. More sophisticated animations can show motion from critical angles or even inside of parts, a definite advantage over building and running a physical prototype.

In my approach for generating animation for the articulated body, I have chosen the non classic and more complex dynamic approach where the motion of the character is generated from the amount of force you apply on the joints of the articulated body. The algorithm that will be described below will be a dynamic approach where we calculate the linear and angular velocities based on the force that has to be applied to a joint based on its physical properties, current position and the desired position. The algorithm is a very simple iterative correction of the joint angular velocities that result from the joint forces and their connectivity with the parent and the child of the joint.

Inspiration for this implementation

My basic interest is physics as a subject and character animation is one of the reasons I started researching the area of physics based animation. I started with looking into the area of space-time constraints and biomechanical laws and wanted to control the character animation in real-time. Pixar's Luxo Jr and the amazing Natural motion software are some of the reasons for me to look into the area of physics based animation.

With the hardware capabilities that a normal user has, it is very much possible to do lots of calculations on in real-time and achieve the natural physical motion. Hence it's a good time to move the character using physics and make the motion as natural and interactive as possible.

- Animation in games and movies
- Natural Motion
- Dance and animation

Luxo Physical Model

My Luxo model has the following configuration:

Joints:

There are there joints namely connecting four links.

- Base Joint
- Mid Joint
- Head Joint

All the joints are modeled as rotational Hinge Joints which connect the two links. As explained a rotational Hinge joint can have at most three DOF.

All the joints in the Luxo have 1 Degree of freedom (DOF) about Z-axes.
Which introduces 2 rotational and there translational constraints at each joint.

The figures below show the complete and physical view of the lamp.
As mentioned above the joints are names and labeled in the below figures.

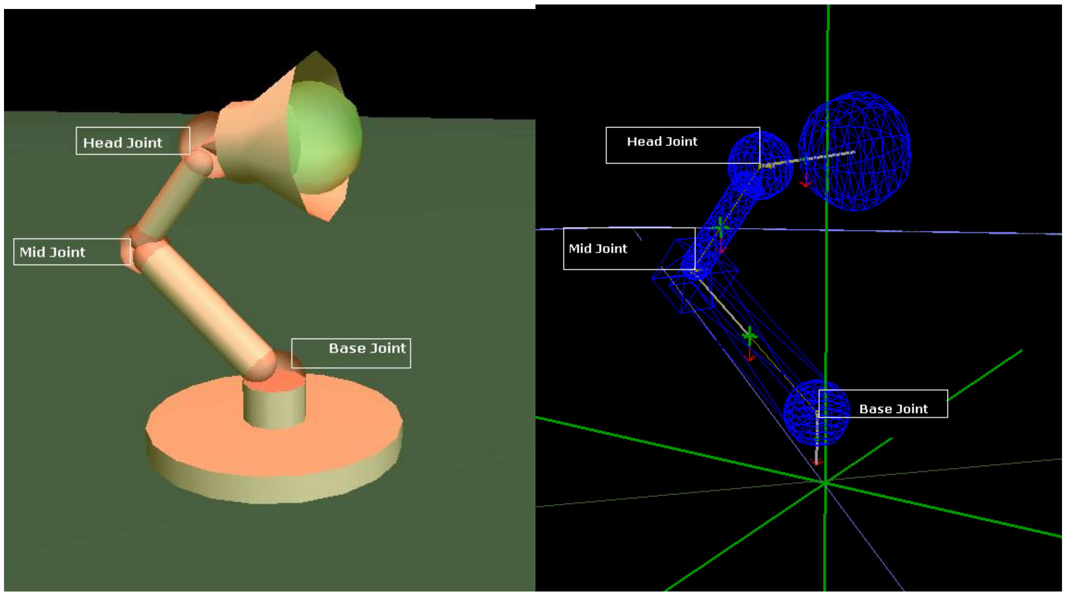


Figure - 12

All the three joints rotational joints having only one Degree Of freedom in the Z axes. The above figures show the resting position of the lamp.

Links:

There are four links (Dynamic nodes) which can move around the world and are connected by these joints.

- Base
- Lower Arm
- Upper Arm
- Head

Each of these have a mass and material associated with them. The material tells the physics simulation about the density and elasticity of the links.

Base Link: It is the made with two cylinders (lower and upper), rotated locally in order to make the cylinder face the ground. Once the geometry is created it is attached to the root dynamic node, which has all the physics properties needed

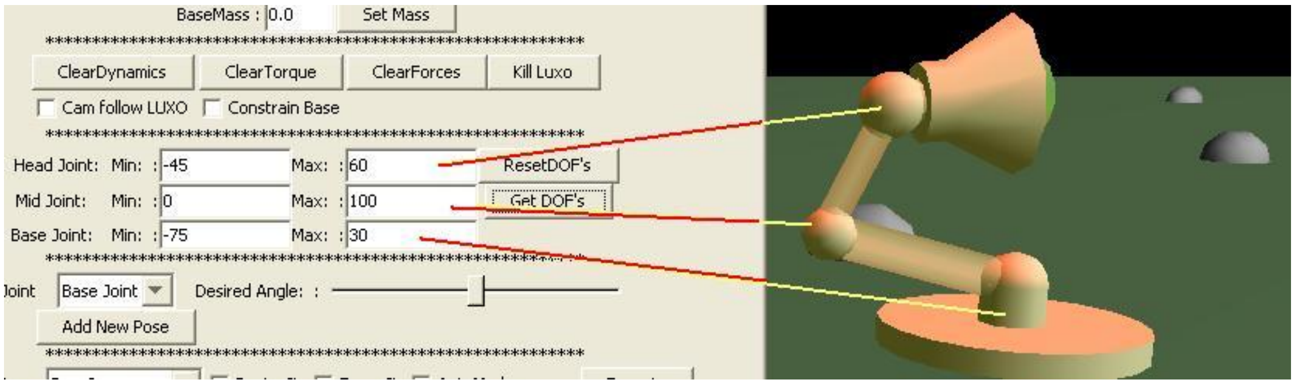
Upper and Lower Arm: These are made with simple capsules that are aligned and given world position relative to its parent. Here the parent of the lower arm is the base and the parent of the upper arm is the lower arm.

Head: The head of the lamp is made with a cone, cylinder and two spheres.

The head has one single dynamic node that can translate and rotate, while has multiple geometries.

The connectivity and the ordering of these links and joints are very important.

- Base Joint connects the Base and lower arm
- Mid Joint connects the Lower and Upper arm
- Head Joint connects the upper and the Lower arm


BaseMass : 0.0

Head Joint:	Min: -45	Max: 60
Mid Joint:	Min: 0	Max: 100
Base Joint:	Min: -75	Max: 30

Joint: Base Joint

Figure - 13

Below are two blocks of code which show the actual functions written in java and using the JMonkey physics (https://jmephysics.dev.java.net) to show how to create joints between two dynamic physics nodes.

```
private void createBaseRigidLink(String name,float lowR,float
upR,float lowH,float upH){

s_baseDPN          = m_physicsSpace.createDynamicNode();
m_baseGeomNode     = new Node(name);

Cylinder baseCyl1 = new Cylinder("lower",2,10,upR,upH,true);
Cylinder baseCyl2 = new Cylinder("upper",2,20,lowR,lowH,true);
Sphere jointSphere = new Sphere("joint",10,10,upR);
Utils.setColor(jointSphere, new ColorRGBA(1,0,0,.2f), 1);

m_baseGeomNode.attachChild(baseCyl1);
m_baseGeomNode.attachChild(baseCyl2);
m_baseGeomNode.attachChild(jointSphere);
m_baseGeomNode.setModelBound(new BoundingBox());
m_baseGeomNode.updateModelBound();
s_baseDPN.attachChild(m_baseGeomNode);
s_baseDPN.generatePhysicsGeometry(true);
s_baseDPN.getLocalRotation().fromAngleAxis(-FastMath.PI/2,
Vector3f.UNIT_X);
attachChild(s_baseDPN);
      }
```

```
private void createHingeLinkJoints() {
            Vector3f anchor2 = new Vector3f(0, 3, 0);
            Vector3f anchor1 = new Vector3f(0, 3.7f, 0);
            Vector3f anchor0 = new Vector3f(0, 0, 1);

s_headJoint = createRevJointAxis("HeadJoint", s_upperDPN,
s_headDPN, anchor2, new Vector3f(0, 0, 1), -45, 45);
s_midJoint = createRevJointAxis("MidJoint", s_lowerDPN,
s_upperDPN, anchor1, Vector3f.UNIT_Z, 0, 90);
s_baseJoint = createRevJointAxis("BaseJoint", s_baseDPN,
s_lowerDPN, anchor0, Vector3f.UNIT_Z, -50, 50);
      }

private Joint createRevJointAxis(String name,
DynamicPhysicsNode nodeA, DynamicPhysicsNode nodeB, Vector3f
anchor,
Vector3f dir, float min, float max) {
Joint joint = m_physicsSpace.createJoint();
joint.attach(nodeA, nodeB);
joint.setAnchor(anchor);

JointAxis jointAxis = joint.createRotationalAxis();
jointAxis.setDirection(dir);
jointAxis.setPositionMinimum((float)Math.toRadians(min));
jointAxis.setPositionMaximum((float)Math.toRadians(max));
return joint;
}
```

Centre of mass of the system

Each of the rigid links connected using the hinge joints have mass associated with them. This will eventually lead to have a center of mass of the entire system COM (system).

Below figures show the centre of mass of each link and the entire system in different poses. The center of mass of the entire system is calculated every frame as the links move dynamically in every frame.

Below are some of the figures which show how the COM is represented in the Luxo system fro each rigid links (yellow) and the COM of the entire system in (green)

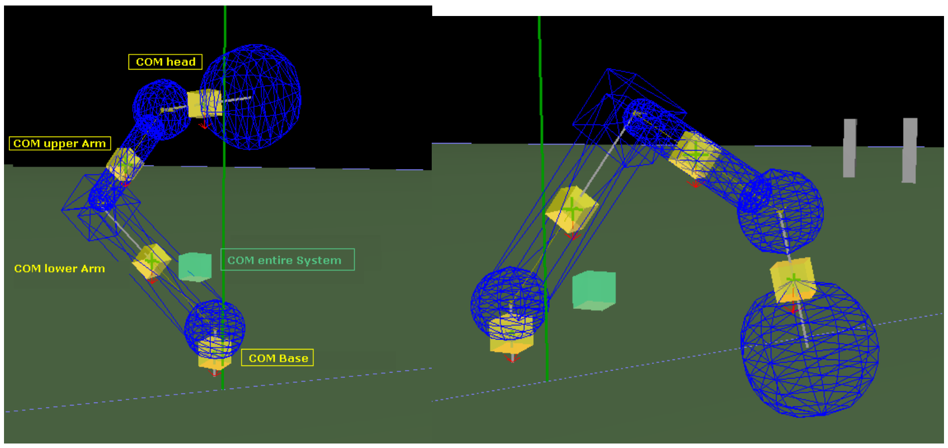


Figure - 14

$$\mathbf{R} = \frac{\sum m_i \mathbf{r}_i}{\sum m_i}$$

For a continuous distribution with mass density $\rho(\mathbf{r})$ and total mass M, the sum becomes an integral:

$$\mathbf{R} = \frac{1}{M}\int \mathbf{r}\, dm = \frac{1}{M}\int \rho(\mathbf{r})\,\mathbf{r}\, dV = \frac{\int \rho(\mathbf{r})\,\mathbf{r}\, dV}{\int \rho(\mathbf{r})\, dV}$$

If an object has uniform density then its center of mass is the same as the centroid of its shape.

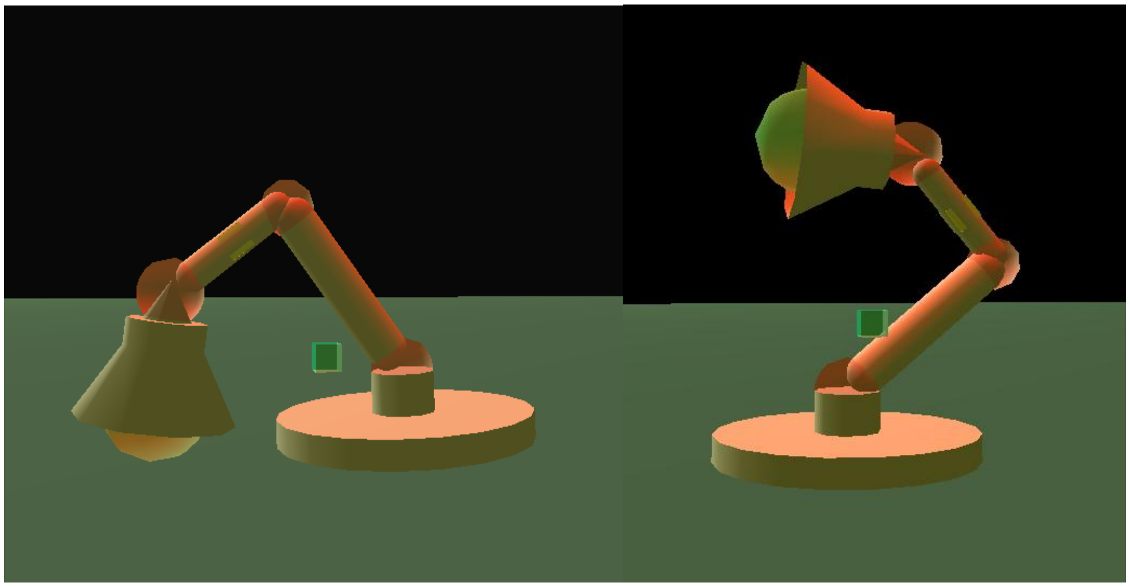


Figure - 15

Below is the code to calculate the COM of the entire system

```
public void updateCOMPositions(){
m_LuxoCOMPos              =
LuxoModel.s_baseDPN.getLocalTranslation().mult(LuxoModel.s_baseDPN.ge
tMass());

m_LuxoCOMPos.addLocal(LuxoModel.s_lowerDPN.getLocalTranslation().mult
(LuxoModel.s_lowerDPN.getMass()));

m_LuxoCOMPos.addLocal(LuxoModel.s_upperDPN.getLocalTranslation().mult
(LuxoModel.s_upperDPN.getMass()));

m_LuxoCOMPos.addLocal(LuxoModel.s_headDPN.getLocalTranslation().mult(
LuxoModel.s_headDPN.getMass()));

m_LuxoCOMPos.divideLocal(LuxoModel.s_baseDPN.getMass() +
LuxoModel.s_lowerDPN.getMass()
                        + LuxoModel.s_upperDPN.getMass() +
LuxoModel.s_headDPN.getMass());

LuxoModel.s_LuxoCOMBox.getLocalTranslation().set(m_LuxoCOMPos);

}
```

Measure of Joint Angle

Since all the angles are having only single DOF and around the Z axes, the system becomes very easy to calculate measure the angle at joint. All the joints around the Z axes, and the default starting position of the lamp where all the joints have a zero angle is facing upwards.

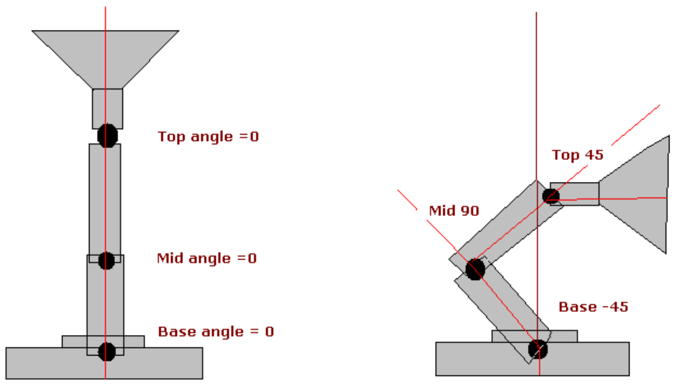


Figure – 16

In the figure above you can see that the angle made by a joint is measured with the axes of its previous parent link

Representing a Pose:

The pose is represented in the system as a data type [class] that holds the information of each joint angle, the position of the base in world coordinate system, the time required to move form one pose to the next pose and finally the name of the pose.

Since we have only 3 DOF fro each joint, a pose can be represented by a data structure that can handle 3 variables that is angle configurations for each joint for that particular pose.

Example of a Pose: Pose2 VectorDOF<-20,0,0>

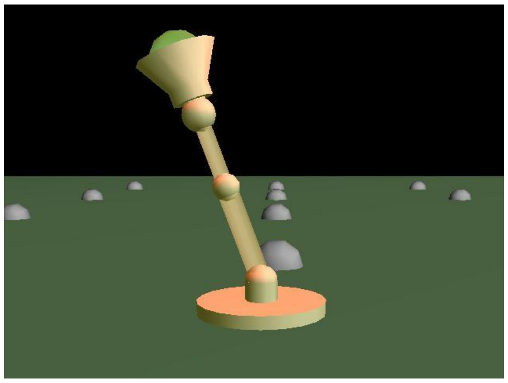


Figure - 17

Example of a Pose: Pose5 VectorDOF<0,60,0>

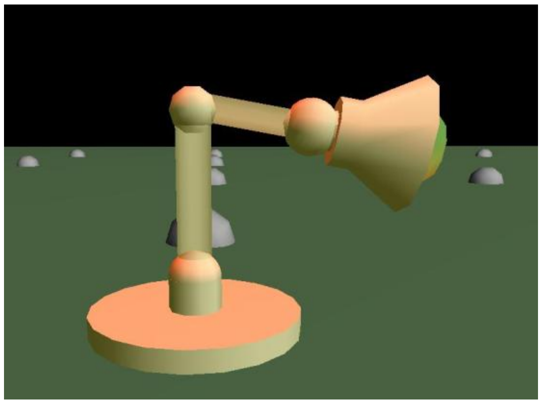


Figure - 18

Representing a Sequence

Sequence element syntax:

[Pose Id, Time for Pose Movement]

A sequence is a group of poses arranged in a particular order by the user of the animator. Each sequence can have any number of poses and can be repeated also.

Apart from taking in the list of poses a sequence takes in an extra parameter called the time frame for each pose. While the simulation is carried the algorithm checks if the user has enabled time constraints in the simulation. If the Time constraints are enabled in the simulation will make sure the pose to pose movement reaches in the amount specified by the user.

The very unique thing about the time constraints imposed at each pose level is that we can have independent times specified for the same pose repeated several number of times in the sequence.

For example: If we have a poses Pose1, Pose2, Pose3, Pose4…
Using above poses we will be able to define a sequence in the following way

Sequence {[Pose1, 1], [Pose2, 3], [Pose3, 4], [Pose1, 3]}

In the above example as you can see that though the Pose1 is repeated twice in the sequence, they have independent times

System overview view

The system can be basically divides into three main sections:

1) Physics Simulation
2) Computing the Joint Torques and Angular velocities
3) Joint correction to meet desired pose from user interface

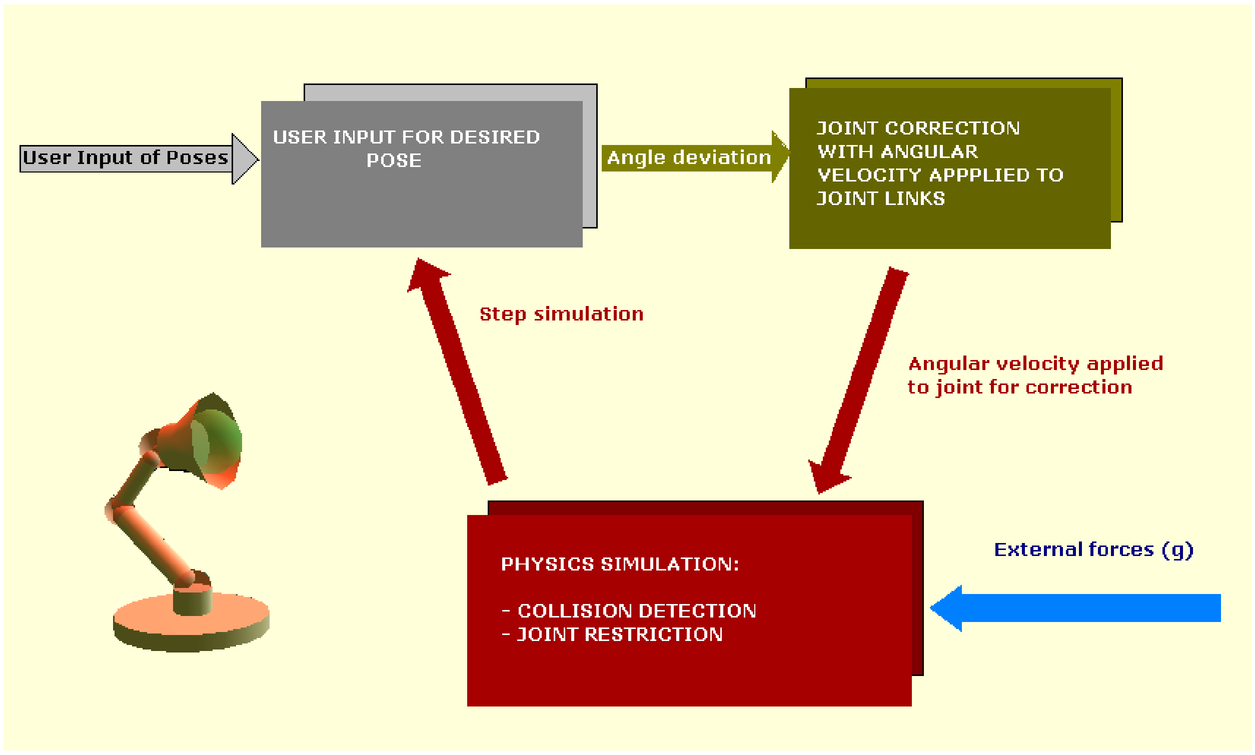

User Input of Poses
USER INPUT FOR DESIRED POSE
Angle deviation
JOINT CORRECTION WITH ANGULAR VELOCITY APPPLIED TO JOINT LINKS
Step simulation
Angular velocity applied to joint for correction
External forces (g)
PHYSICS SIMULATION:
- COLLISION DETECTION
- JOINT RESTRICTION

Figure - 19

Physics Simulation

The physics I am using here for all the Joint restrictions and collision is from an open source physics wrapper written in java on top of ODE (Open dynamics engine http://www.ode.org/) called JMonkey Physics (https://jmephysics.dev.java.net/) Check the Appendix-A for more details on the technologies used.

These systems handle all the collisions and the joint restrictions on the articulated body. JME physics is a scene graph based engine and allows creation dynamic and static nodes. All the links in the Luxo lamp are dynamic nodes.

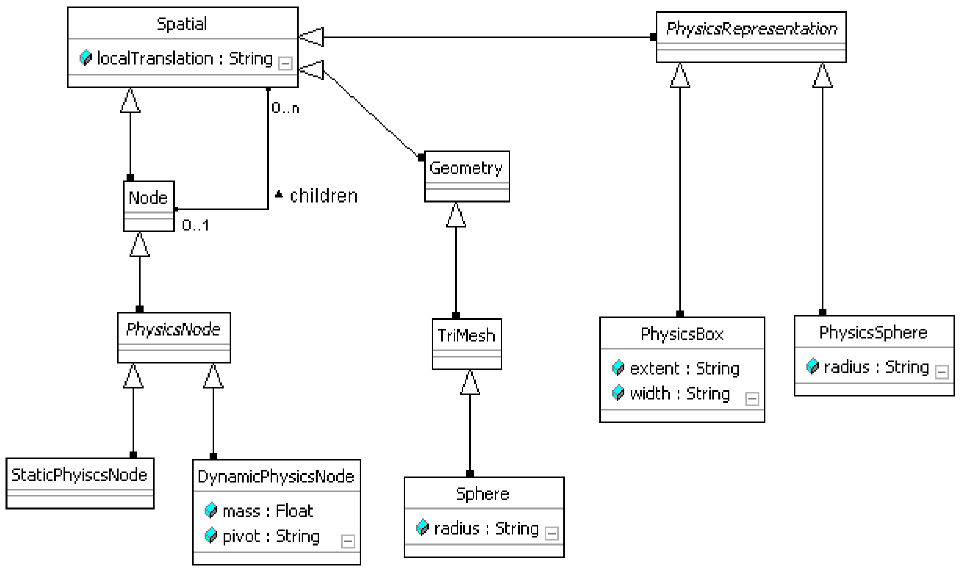


Figure - 20

JME format gets extended to allow storage of such scene graphs, a filter may be provided to store only physics/visuals

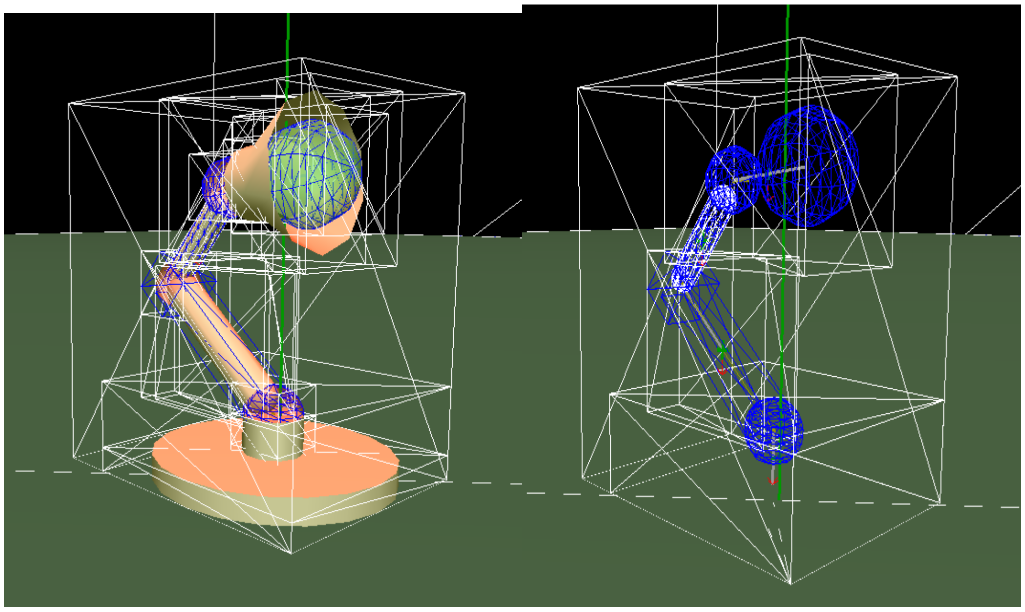


Figure - 21

The image on the left is enabled with all the bounding boxes, for each link and system as a whole. Right the same with the geometry disabled.

The physics simulation takes care of the following things:

1) Collision of the base and other links with the environment
2) Joint restrictions, the way the hinge joints are constrained
3) Applying gravitational force of -9.8 m/sec^2 along the y- axes

The physics calculations are very fast and the whole simulation runs at 330 FPS

Computation of Joint Torques and Angular Velocities

This is the core of the system is this module is responsible for computing the required amount of torque that has to be applied to the links connecting the joints. The torque that is applied to each joint's connected links is directly to the mass its own along with its children links.

During every frame, the algorithm runs through all the joints and computes the joint difference between the desired and the current angle and uses it to compute the requited angular velocity to the joint. Once the difference in the desired and the current joint angle is computed, the algorithm has to decide whether the joint has to be moved in clockwise or anticlockwise direction.

Since all the joints can rotate along Z-axes only, the following can be applied:

Anti clockwise rotation along Z- axes:

Applying positive angular velocity along Z- axes makes the joint rotate in anti Clockwise. This will be applied by the simulation to the joint when the current joint angle is greater than the desired joint angle

Clockwise rotation along Z- axes:

Applying negative angular velocity along Z- axes makes the joint rotate in clockwise. This will be applied by the simulation to the joint when the current joint angle is less than the desired joint angle

Once the difference is computed the joint's angular velocity is applied based on its current and desired, which makes the joint to move in the desired direction. Once the joints start moving the difference between the desired and the current joint angle becomes smaller and smaller and finally comes to a stable equilibrium.

As the joints are moving step by step in the simulation loop, the whole character looks like animating from one pose to the other pose. The physics engine takes care of the joints not crossing the minimum and the maximum (DOF) of the joints while correction is happening in each time step.

The torque is always applied to the connected joints at the center of mass by the physics simulation, which in turn generates the required motion of the joint.

The right hand rule is applied by the simulation to

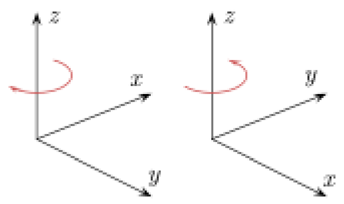


Figure - 22

The left-handed orientation is shown on the left, and the right-handed on the right

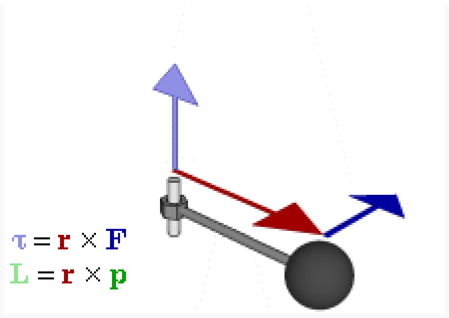

$\tau = \mathbf{r} \times \mathbf{F}$
$L = \mathbf{r} \times \mathbf{p}$

Figure - 23

this diagram shows the relation between force (F), Torque (T) and the momentum vectors in a rotating system

This below code is responsible for calculating the angle difference for each joint in each frame and applies the right hand rule to get the desired motion in the articulated joint of the body

Algorithm and Pseudo code of iterative Joint correction:

- For each joint [Base, Mid and Head]
- Calculate the difference of the angel in current and desired
- Compute the diff of the connected links COM to the entire systems COM
- If time constrained, scale the diff with the “dt”
- Choose the direction of the velocity based on the difference in the angle and Flemings right hand rule
- Calculate the magnitude of Angular velocity and scale it based on the mass of the link to be applied to
- Apply velocity to the connected Link
- If difference less than [thresh] rest the link- desired angle is reached

```
public void setRequiredAngularVelocity(JointType jType){
m_diffRad          = _axis.getPosition() -
m_desiredPose.getDesiredRad(jType);
double dt;
if(null != m_desiredSequence &&
FloatingMenu.getInstance().isTimeConstrained())
            dt = m_desiredSequence.getCurrSeqElemTime();
      else
            dt =1;

      float dist =
_linkToRotate.getLocalTranslation().distance(m_LuxoCOMPos);
                  dist*=dt;

 if(m_diffRad<-errorAllowed){
       _axis.getDirection(_clockWiseW);
      _clockWiseW.multLocal(-_scaleOfRotation);

//Need ClockWise Rotation, hence Apply Negative Angular Velocity

_linkToRotate.setAngularVelocity(_clockWiseW.mult(_linkToRotate.getMa
ss()/dist));
      }else if (m_diffRad>errorAllowed) {
 _axis.getDirection(_antiClockWiseW);
 _antiClockWiseW.multLocal(_scaleOfRotation);
//Need ClockWise Rotation, hence Apply Negative Angular Velocity

      _linkToRotate.setAngularVelocity(_antiClockWiseW.mult(_linkToRo
tate.getMass()/dist));
                  }else{
                        _linkToRotate.rest();
                  }
            }
      }
```

The above code snippet shows the actual code of how the angular velocity that has to be applied to a joint is calculated in a simulation loop of the whole algorithm.

User Interface

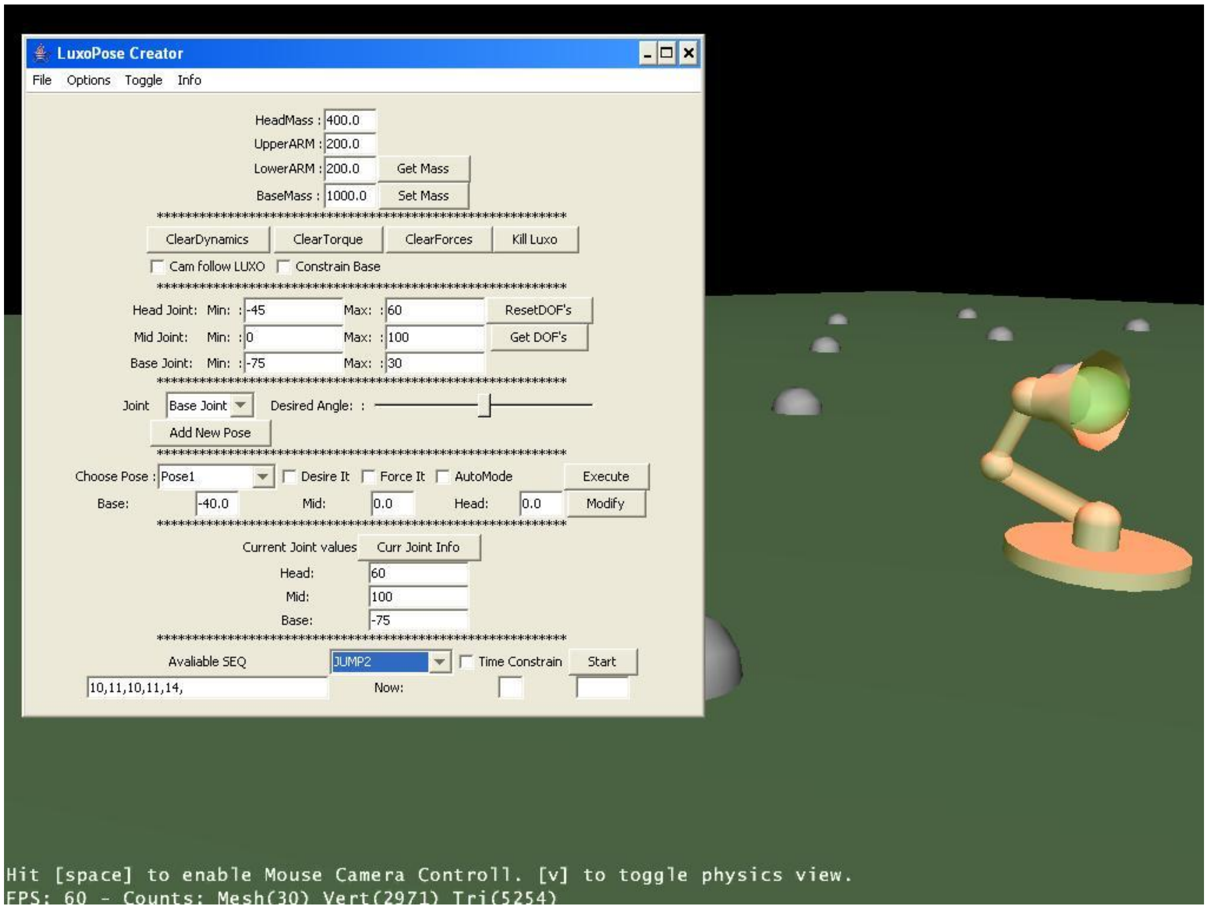
The interface for controlling the lamp is pretty simple. The user can do the following thing from the interface

1) From the menu users can choose to enable and disable the physics view, geometrical view, COM of each link, COM of entire system
2) Mass Panel: This panel can be used to set and get the mass of the each link in the entire system. The links of the rigid body will dynamically react to the change in the mass while the animation is proceeding
3) Dynamics Panel: This panel has the options to clear the following physics elements of the entire system
    a. Clear Dynamics [clears all the torques, velocities and forces on all links and joints of the system]
    b. Clear Torques and Forces: These will clear all the torques and forces acting on each link and joint of the entire system
4) Enable and disable Constraints like
    a. Constraints the base
    b. Make the camera follow the Luxo
5) DOF Pane: Modify or Reset the Degree of freedom (DOF) for each joint
6) Add Pose Pane: Add a new desired pose, by specifying the desired angle for each joint
7) Pose Chooser Pane : Choose a pose and either desire it or force it, also modify the pose details before desiring it or forcing it, and then execute to start motion
8) Current Joint Pane: See the current joint angles from the current simulation in degrees

9) Sequence Pane: This is the most interesting part, you can choose a series of poses and the simulation will start moving for pose to pose

10) Just below the Sequence you can how you can enable and disable the time constraints on the system

11) Num Lock keys, 0,1,2,3,4,5,6,7,8,9 are mapped to the poses. On choosing the pose the lamp stars moving to the desired pose



HeadMass : 400.0
UpperARM : 200.0
LowerARM : 200.0
BaseMass : 1000.0

Cam follow LUXO
Constrain Base

Head Joint: Min: : -45 Max: : 60
Mid Joint: Min: : 0 Max: : 100
Base Joint: Min: : -75 Max: : 30

Joint Base Joint Desired Angle: :

Choose Pose : Pose1 Desire It Force It AutoMode
Base: -40.0 Mid: 0.0 Head: 0.0

Current Joint values
Head: 60
Mid: 100
Base: -75

Avaliable SEQ JUMP2 Time Constrain
10,11,10,11,14, Now:

```
Hit [space] to enable Mouse Camera Controll. [v] to toggle physics view.
FPS: 60 - Counts: Mesh(30) Vert(2971) Tri(5254)
```

Figure - 24

Other Short Cuts keys:

1) WSAD: Use to navigate into the scene
2) Arrow Keys: Use to move the camera up and down
3) Mouse Left Click: Use the mouse to pick the Luxo in the scene and place it any where in the scene
4) Space Bar : Mouse controlled camera mode [toggles]
5) Key V: Toggle the physics debug mode, can see the forces acting
6) Key N: Toggle Normal's of the geometry
7) Key B: Toggle bounding draw bounding
8) Key L: Toggle Lights in the scene
9) Key T: Toggle wireframe mode of the scene

The entire user interface has been designed using Java swings. The actual rendering is done using the OpenGL based JMonkey engine.

Results

Pose to Pose motion:

The lamp can move from one pose to the other pose, with all the joint and the floor constraints satisfied, the motion is all physically realistic as it is all simulated in total physics environment. The whole application runs at a maximum of 350 FPS. The best thing is all the complexities of the computing the joint forces, and solving multiple linear system of equation is avoided. And the real time interaction with the lamp is achieved with out any key framing of joint angles.

When we have a good number of poses defined and mapped to keys as mentioned above. Upon interaction with the keys randomly or with a good sequence the whole pose to pose simulation running in real-time looks like a controlled dance and is totally interactive.

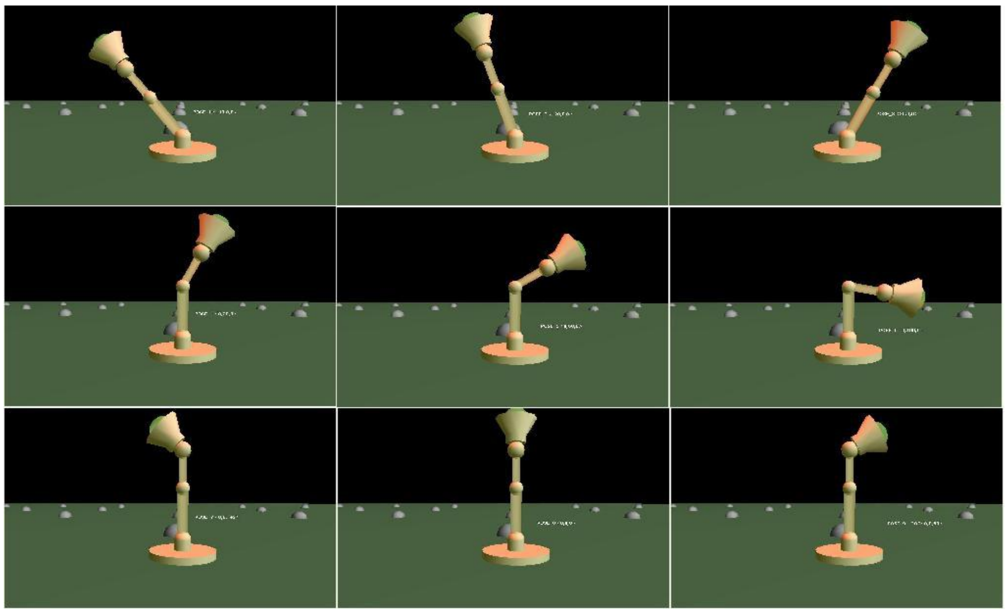


Figure - 25

<u>Luxo Dance [defining a sequence of poses and time intervals]:</u>

The Lamp can move form pose to pose continually by specifying a series of poses it has to under go. Along with the poses the user can specify in seconds, how long the simulation has to take to go from pose to pose in the specified series. Since we are controlling the pose to pose transition along with the time and space basically we are exposing a simpler interface to the user or artist to go ahead in time and space and specify how his animation character has to look at a particular point of time in the entire sequence of animation.

The basic definition of "Space Time" constraints is to allow the constraints to be met by going forward both in space and time. We have a simpler interface created for the user to think ahead in space an time and control both the space and time constraints of the character animation.

Since all the simulation is done in a fully physics simulated environment we have the options of modifying multiple parameters as the simulation proceeds, and expect different results of end motion for the same predefined set of poses [sequence of poses]

Changing the Physical characteristics during animation is the most fun part of the entire algorithm. In the simulation we have tried a demo where the characters mass of the head is increased from 400 units to 1000 units and the animation changed naturally as if it's lifting a heavy object on its head. Also you can notice the change in its COM of the system.

Example: If I have a pool of 20 poses defined by the user say Pose_0, Pose_1….. Pose_20, having the joint configuration values encoded in them. This set of poses can be combined into a series "Sequence", along with the length of time that pose has to stay on the sequence. The simulation can slowly move the character from on pose to the other pose by applying required angular velocities to the required joints and make the whole sequence look like animation. Since we are controlling the characters motion and constraining it both in space an time we can call this a space time approach.

The vast difference of this approach when compared to the traditional animation where the joints are interpolated from one pose to the other does not take into considerations of the physical properties of the character like its mass distribution, environment constraints and most importantly not interactive and dynamic to the user input.

Here if we change the mass of the characters body part, like head or base for the same poses with the same sequence, we come up with a totally new animation which follows the mechanics of physics. Hence we have the control to generate unique motion every time we change the physical property of the character.

Below are some of the predefined poses along with some sequences generated form this poses. See how the lamp reacts to the change in its mass of head, as the COM of the entire system shifts towards the head.

If the user is aware of good poses he can imagine he can end up with a good animation sequence.

Statistics and performance

The whole simulation currently has the following statistics:

Number of Joints:	3 joints
Number of Triangles:	2856 {Luxo}
Number of Vertices:	1621
Number of Lights:	2 lights

System Configuration:
All the simulation has been tested on a Intel Pentium processor T2400 @ 1.83Ghz

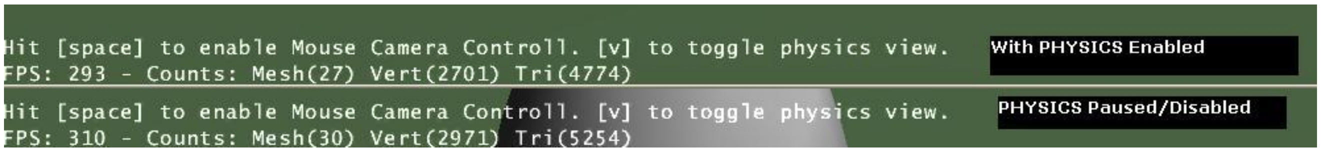
2.00 GB RAM, ATI Graphics

With out Physics:

With Physics disabled the simulation could run at a frame rate of 310-330 FPS

With Physics enabled:

The Simulation runs at a frame rate of 290-300 PFS



```
Hit [space] to enable Mouse Camera Controll. [v] to toggle physics view.
FPS: 293 - Counts: Mesh(27) Vert(2701) Tri(4774)
Hit [space] to enable Mouse Camera Controll. [v] to toggle physics view.
FPS: 310 - Counts: Mesh(30) Vert(2971) Tri(5254)
```

Figure - 26

The physics joint correction itself is very fast and can be extended with out any over head and more constraints to complex creatures like human.

Conclusion and future work

Hence an articulated figure with joints and links setup properly in a fully physics environment can be controlled as per the users desire in real time. We have eliminate all the complex calculations involved in calculation the muscle forces usually represented with springs which are usually very unstable and complex.

Also we have removed the dependency on traditional method of key framing where the artist has to under go a painful procedure in defining all possible scenarios which may not be physically realistic. Apart from that the characteristics of the joints and their DOF and the physical properties of the links can be modified to get different results using the same algorithm.

This simple approach of controlling an articulated character with joint angular velocities can be extended in much way for very complex characters with a larger set of constrains and DOF's defined. Adding different kinds of joints as discussed above will also make the animation more interesting.

The joint correction based on the mass displacement, space and time constraints imposed my user itself is very fast and can be extended to complex characters.
I don't see and huge performance drop with introducing some more joints and constraints.

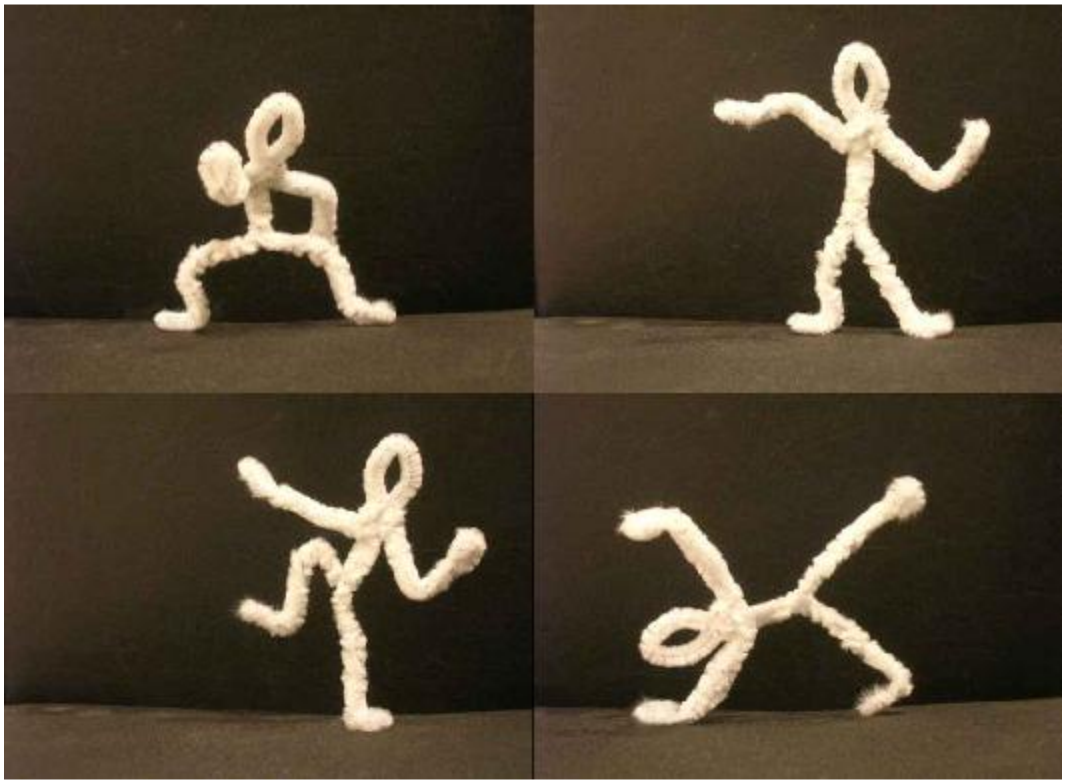


Figure - 27

http://www.davidbessler.com/pulldown/pipecleaner_dance3.swf

My future extension for this project will be extending the same application to a human like model and making him dance based on the music events derived on the fly using a good DSP tool. Adding music along with predefined poses and using a physics simulated environment makes the over animation experience a feel of character dancing to the music. Apart from that the user can define their own moves and poses for the characters at some predefined events as discussed in the "Momentum based parameterization of character animation – Zoran Popovic"

Appendix – A

JMonkey Engine, a Open Source OpenGL based Java Graphic Engine

http://www.jmonkeyengine.com/

Introduction:

jME (jMonkey Engine) is a high performance scene graph based graphics API. Much of the inspiration for jME comes from David Eberly's book 3D Game Engine Design. jME was built to fulfill the lack of full featured graphics engines written in Java. Using a abstraction layer, it allows any rendering system to be plugged in. Currently, LWJGL is supported with plans for JOGL support in the near future.

History

jME was created by Mark Powell in 2003 while he was investigating OpenGL rendering. After discovering LWJGL he decided that Java (his language of choice) would be perfect for his own graphics tools. These tools soon grew into a primitive engine. After reading David Ebery's 3D Game Engine Design, scene graph architecture was implemented. It was then that jME became part of Sun's Java.net software repository. jME soon saw others joining the project to enhance it's capabilities. It has since grown to encompass many advanced modern graphics features and turned into a stable platform for game development. Joshua Slack joined jME at the end of 2003 and became a core member and integral part of the jME team.

<u>Features of jME:</u>

jME is a scene graph based architecture. The scene graph allows for organization of the game data in a tree structure, where a parent node can contain any number of children nodes, but a child node contains a single parent. Typically, these nodes are organized spatially to allow the quick discarding of whole branches for processing. For example, if we build a graph such that all objects in a room share a parent (room), and all rooms share a parent (floor), where all floors share a parent building. Our character is in room 1 of the first floor. We can quickly discard the floor 2 node (which is turn discards every room on the 2nd floor and every object in those rooms). We can then process the floor 1 branch of the tree. All rooms that are not room 1 are discarded (including all objects in that room). We then process room 1 including its objects.

Discarding can mean a variety of things, but the most significant in Graphics programming is culling of data. jME's camera system uses frustum culling to through out scene branches that are not visible. This allows for complex scenes to be rendered quickly, as typically, most of the scene is not visible at any one time.

The leaf nodes of the scene graph are Geometry that will be rendered to the display. There are many supported Geometries, including: Bezier Patches, Line, Points, Models (Milkshake, MD2, ASE, etc), Terrain, Level of Detail, and more.

jME also supports many high level effects, such as: Imposters (Render to Texture), Environmental Mapping, Lens Flare, Tinting, Particle Systems, etc.

jME supplies the user with easy to use, but powerful application classes for building the application. Jumping into jME should be a quick and painless process. With a small learning curve, you can have your game up and running in no time.

JME Physics/ODE

jME Physics was providing an interface between jME (Java Monkey Engine) and ODE (Open Dynamics Engine). It sat on top of a slightly modified version of ode java, and provided a way to very easily set up a physics world and add objects to it - a simple box-falling-on-a-floor application is only a matter of a few lines. Yet, the goal was to provide advanced and powerful features - e.g. vehicle support is implemented. Physics objects and types get specified within the scene graph like this:

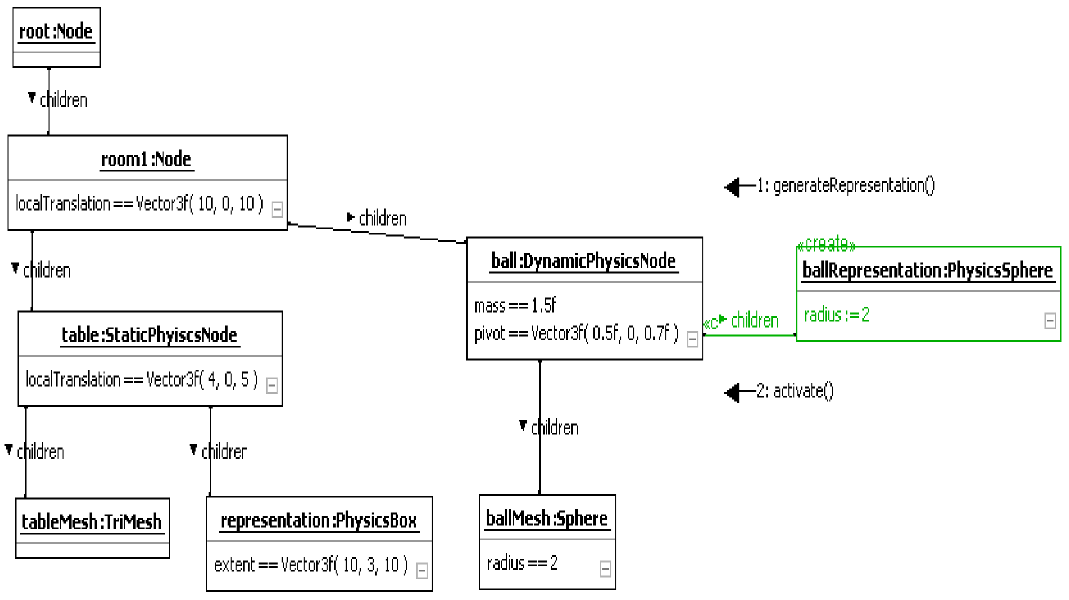

root :Node
children
room1 :Node
localTranslation == Vector3f(10, 0, 10)
children
children
table :StaticPhyiscsNode
localTranslation == Vector3f(4, 0, 5)
children
childrer
tableMesh :TriMesh
representation :PhysicsBox
extent == Vector3f(10, 3, 10)
ball :DynamicPhysicsNode
mass == 1.5f
pivot == Vector3f(0.5f, 0, 0.7f)
children
ballMesh :Sphere
radius == 2
1: generateRepresentation()
«create»
ballRepresentation :PhysicsSphere
radius := 2
«c» children
2: activate()

Figure - 28

References:

[1] Space-time Constraints Andrew Witkin Michael Kass

[2] Andrew Witkin, Kurt Fleischer, and Alan Barr, *Energy constraints on parameterized models,* Computer Graphics, 21 (4) July 1987, pp. 225-232 (Proc. SIGGRAPH '87). 168

[3] Unique Game Moments [2006], Natural Motion Ltd, Oxford, UK

[4] Dynamic Motion Synthesis [2005], Natural Motion Ltd, Oxford, UK

[5] http://www.naturalmotion.com/

[6] http://www.davidbessler.com/pulldown/pipecleaner_dance3.swf

[7] Pixar, *Luzo, Jr.,* (film,) 1986

[8] MIRTICH, B. 1996. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley

[9] Practical Physics for Articulated Characters [**Evangelos Kokkevis**] *Game Developers Conference 2004*

[10] HECKER, C. 1998. Rigid body dynamics.

[11] http://www.d6.com/users/checker/dynamics.htm

[12] *Physically Based Modeling, Course Notes*. ACM Siggraph. BARAFF, D., KASS, M., AND WITKIN, A. 1999.

[13] William W. Armstrong and Mark W. Green, *The dynamics of articulated rigid bodies for purposes of animation* in *Visual Computer,* Springer-Verlag, 1985, pp. 231-240.

[14] Ronen Barzel and Alan H. Barr, *Dynamic Constraints,* Topics in Physically Based Modeling, Course Notes, Vol. 16, Siggraph 1987

[15]Michael Brady et. el., eds, *Robot Motion: Planning and Control,* MIT Press, Cambridge, MA, 1982

[16] Charles E. Buckley, *The Application of Continuum Methods to Path Planning,* Doctoral Dissertation, Dept. of Mechanical Engineering, Stanford University, Steax£ord, CA, 1985

[17] Kurt Fleischer and Andrew Witkln, *A modeling testbed,* Proc. Graphics Interface, 1988.

[18] Phillp Gill, Welter Murray, and Margret Wright, *Practical Optimization,* Academic Press, New York, NY, 1981

[19] Michael Girard and Anthony a Maciejewski, *Computataional Modeling/or the Computer Animation of Legged Figures,* Proc. SIGGRAPH, 1985, pp. 263-270

[20] Herbert Goldstein, *Classical Mechanics,* Addison Wesley, Reading, MA, 1950

[21] David Haumarm, *Modeling the Physical Behavior of Flezible Objects,* Topics in Physically Based Modeling, Course Notes, Vol. 16, Siggraph 1987

[22] Paul Isaacs and Michael Cohen, *Controlling Dynamic Simulation with Kinematic Constraints, Behavior*

[23] Charles Klein and Ching-Hsiang Huang, *Review of Pseudoinverse Control for Use with Kinematically Redundan Manipulators,* IEEE Trans. SMC, Vol. 13, No. 3, 1983

[24] John Lasseter, *Principles of Traditional Animation Applied to 3D Computer Animation,* Proc. Siggraph
1987, pp. 35--44

[25] William Press et. al., *Numerical Recipes,* Cambridge University Press, Cambridge, Engiemd, 1986

[26] Robert S. Stengel, *Stochastic Optimal Control,* John Wiley and Sons, New York, 1986.

[27] Demetri Terzopoulos, John Platt, Alan Barr, and Kart Fleischer, *Elastically Deformable Models,* Pros. SIGGRAPH, 1987.

[28] Jane Wilhelms and Brian Barsky, *Using Dynamic Analysis To Animate Articulated Bodies Such As Humans and Robots,* Graphics Interface, 1985.

[29] http://animationphysics.wordpress.com/

[30] Synthesis of Complex Dynamic Character Motion from Simple Animations -*C. Karen Liu Zoran Popovi´c 2005*

[31] Physically Based Motion Transformation – Zoran Popovic Andrew Witkin

[32] Momentum – based Parameterization of Dynamic Character Motion – Yeuhi Abe C.Karen Liu Zoran Popovic

*[33] Practical Physics for Articulated Characters -**Evangelos Kokkevis GDC -2004***

BOOKS:

[1] PHYSICS – BASED ANIMATION – [ISBN 1-58450-380-7 56995]

[2] Computer Animation Algorithms and Techniques [ISBN-13:978-1-55860-579-4]

[3] Classical Dynamics of Particles and Systems [ISBN: 0-534-40896-6]