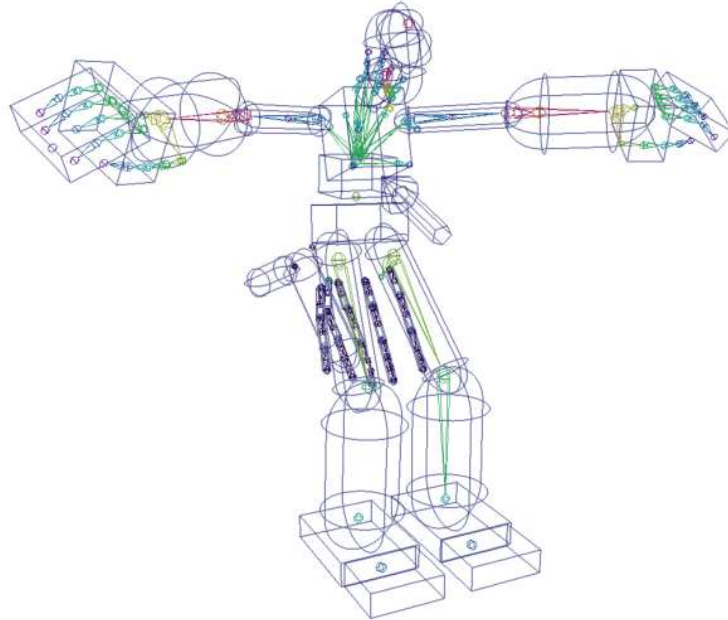


# This Project Presents a Development Framework for Dynamic and Fuzzy Behaviours Focusing on Bipedal Characters using Maya8.0

Luke Titley  
Bournemouth University



**Figure 1:** A Physics Rig In The Laya Framework

## Abstract

This project presents a framework for the development of behaviours within Maya, it also presents a collection of behaviours developed in that environment.

## 1 The Goals and Aims of the Project

The goal of this project is to investigate the current techniques being developed for physically assisted character animation and then develop a working framework which we can support these techniques.

## 2 Introduction

This report documents the research and experimentation undergone for the development of a Physics and Fuzzy logic Behaviour environment in Maya. Ideally with behaviours capable of balancing a bipedal character.

## 3 How this report is structured?

The structure of this document is not chronologically ordered, Rather it covers the theoretical aspects of the subject area and then discusses implementation.

- The First part will discuss the reason for this experiment and its goals and aims.
- The Second will introduce certain theoretical concepts specific to the subject area.

- The Third will suggest a framework.
- The Fourth, will discuss our implementation of the framework.
- The Fifth will discuss behaviours designed using this framework this is where a great deal of experimentation is introduced.
- The Sixth will review our behaviours, and discuss limitations of the framework.
- The Forth further evaluates the project.
- And finally a short personal summary of what has been learnt from the project.  
Which aspects were enjoyable and which were not so enjoyable.

## 4 Why Research Physically Assisted Character Animation?

Traditionally the use of physically based solutions in Film has been restricted to visual effects, such as explosions and breaking [Scheepers and Whittock 2006]. We are now beginning to see the emergence of physically based solutions specifically to assist in character animation.

No discussion on character animation would be complete without first mentioning the 12 principles of animation outlined by [Thomas and Johnston ]. To date these principles serve as a rough guide for producing good character animation.

## 4.1 12 Principles of Animation

1. Squash and stretch
2. Anticipation
3. Staging
4. Straight Ahead Action and Pose to Pose
5. Follow Through and Overlapping Action
6. Slow In and Slow Out
7. Arcs
8. Secondary Action
9. Timing
10. Exaggeration
11. Solid Drawing
12. Appeal

The following principles became less applicable with the introduction of computer animation.

- Solid Drawing  
The Drawing can be part of a separate process.
- Straight Ahead Action and Pose to Pose  
Everything can be modified an infinite number of times. We have more direct control over the characters state in time. Everything becomes pose to pose.
- Arcs  
Shapes can be constrained to their form.

Using a physically based solver we are able to partially with attacking the following principles.

- Follow Through and Overlapping Action  
The weight inherent in rigid body objects automatically computes a follow through and overlapping action.
- Slow In and Slow Out  
Part of the reason for slow in slow out is the physical restrictions on a character. The amount of slow in slow out can be influenced by the mood and temperament of the character however, the animatore still needs control of this.
- Appeal  
The apeal of weight in a characters movement and correct collisions.
- Anticipation  
In the context of a physically correct world, anticipating an action such as throwing would be a neccessity. The animator will need to animate the correct anticipation for the physical simulation to work correctly in the first place.
- Secondary action  
Animation generated as a result of the characters interaction with its environment.

Taking development further we might be able to assist with attacking even more principles.

The use of a physically based solver to assist with these principles should not stifle the creativity of the animator. The solver should serve as tool to minimise more laborious aspects of animation.

## 4.2 Realtime

Considerable research has gone into creating interactive animated characters for computer games. This is a necessity since the animator cannot produce animation while the user plays the game. Both [NaturalMotion ] and [Havok ] provide tools for dealing with interactive character animation scenarios.

A physics solver can help to automate the task of animating weight. Typically we provide an interface for the animator to control positioning of forces and constraints. Physics solvers then produce the animation. [Brice Criswell 2006]

[Mandel ] Explains that we can set goal poses for our skeleton to reach. This is similar to the keyframing approach used in traditional computer animation. However, rather than directly interpolating between keyframes ( FIND A REFERENCE ) a physically based controller will calculate these frames as a result of the accumulation of inertia, constraints and forces. [Wooten 1996] uses posable controllers to animate a diving swimmer.

[J.K. Hodgins and O'Brien. 1995] mentions "Secondary Motions", known as "Secondary Action" in the twelve principles of animation. These are motions which support the primary action of the scene. [J.K. Hodgins and O'Brien. 1995] uses the splash of water from a diver jumping into a swimming pool as an example of a "Secondary Motion" or "Secondary Action".

This approach allows us to use a physics solver to animate the character while still giving a huge level of control to the animator.

A problem arises with such low level access to the solver. Keeping the character upright requires continious and subtle tweaking of our joint rotations. ( REFERENCE ) The animator spends more time trying to keep the character upright than he does trying to make it act.

Ideally a behaviour should keep the character upright while the animator controls the the character's expressive actions.

Techniques have been developed for tracking the trajectory of physically based constraints to moving forward kinematic ( FK ) joints. [Zordan 2002] [Brice Criswell 2006]

Zordan demonstrates that we can use more than just the Physics Solver. Using trajectory tracking we can mix purely procedurally driven animation with FK animation on the same skeleton. One example given is of two characters playing table tennis. The top half of the character is driven by trajectory tracking FK animation. While everything below the hips is controlled by a balancing behaviour.

Our focus then moves on to the development of hybrid behaviours for controlling our physically based rigs.

To summarise, character animation can be broken into two parts.

- Realism of Character ( The character's psychological believability )
- Realism of Movement ( The believability of the character's movement )

Although these parts are not mutually exclusive, a variety of different approaches can be used to tackle each. We will focus on behaviour development as a means to assist in animating Realism of Movement.

## 5 Behaviours and Controllers

This following section will focus on the current techniques being employed in the development of characters behaviours. In the context of this report a Behaviour can be an amalgamation of controllers and mathematical expressions which control part or all of a skeleton.

Definition of Controller: "A piece of equipment or program within a control system which responds to changes in a measured value by initiating a control action to affect that value."

### 5.1 Higher Level Controllers

The following controllers can be considered as higher level controllers, because they rely on primitive lower level controllers to drive the torque of the joints.

#### 5.1.1 Controller Scheduling Using Hierarchical and Non Hierarchical State Machines

[J.K. Hodgins and O'Brien. 1995] proposes the use of state machines to construct Running, Cycling, Vaulting and Balancing behaviours. [Petros Faloutsos and Terzopoulos 2001] builds further on this topic by describing an approach using composable controllers. A hierarchical state machine manages a collection of primitive controllers. Each controller bids for control of the skeleton. The one which bids with the highest priority is given control. It then returns an expected performance, if the performance of the controller degrades, control of the skeleton is opened to bidding and another controller takes control. For the control to succeed the skeleton must be brought to a state in which it satisfies the controller's post conditions. Using this approach it is not possible to run multiple behaviours at once on the same character. Primitive controllers are built from low level PD controllers.

[Havok ] mentions the use of state machines and blend trees for generating dynamic behaviours.

#### 5.1.2 Passive Dynamics

An interesting approach to controller design, known as Passive Dynamics has been proposed by [McGeer 1990]. Passive Dynamics uses the momentum of swinging limbs to better improve efficiency in locomotion. Examples of passive dynamics controllers using the ODE Physics Solver can be found at [DroidLogic ] Currenly these "Dynamic Walkers" rely on a delicate rigs to work, since all motions come from the momentum and gravity acting on the rig. They are unable to walk on anything but slightly inclined flat surfaces. Despite these restrictions an understanding of the theoretics of Passive Dynamics might prove useful when designing behaviours of locomotion.

#### 5.1.3 Cognitive Controllers

Although our research focuses on the low level aspects of behaviour design it is important to mention that a lot of research has been done on the use of soft computing techniques in higher level behaviour development.

#### 5.1.4 Neural Networks and Genetic Programming

[Geng and Bernd Porr ]

### 5.1.5 Fuzzy Logic

[McCuskey ] Introduces us to Fuzzy Logic in game design. The example he gives is that of a driver. Using Fuzzy Logic to control the speed of a car. The GambitSystem for Final Fantasy XII uses fuzzy logic to determine game play, [System ]. [Massive ] software is an example of fuzzy logic being used to control the behaviour of characters on both a low and high level. [Ale ] presents an optimized fuzzy logic architecture for decision making.

EXPLAIN AND GIVE AN EXAMPLE OF HOW FUZZY LOGIC MIGHT BE USED IN DECISION MAKING.

### 5.2 Low Level Controllers

Having looked at the current work in higher level controllers we will now move into the lower level specifics of primitive controllers.

#### 5.2.1 Open and Closed Loop Controllers

Lower level controllers can be broken in two types. Open and Closed loop controllers.

- An Open Loop Controller computes its output using a predefined model of the system
- A Closed Loop Controller will change its output given continuously changing information about the variable it controls.

Closed loop controllers are also known as feedback controllers. We are only concerned with closed loop controllers.

#### 5.2.2 PD and PID Controllers

The high level controllers proposed by [Petros Faloutsos and Terzopoulos 2001] and [Zordan 2002] are reliant on PD Controllers. PD Controllers are Closed Loop or Feedback controllers.

A PD Controller computes an Error value from its inputs. It then computes a solution to the error and outputs that value.

Consider the following scenario.

We have a skelton to control, made of one joint and two rigid bodies.

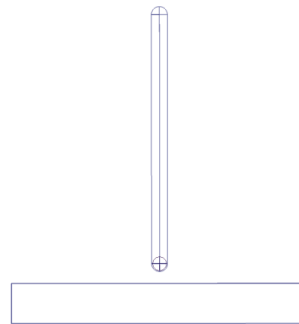


Figure 2: Universe Of Discourse

We have a motor on our joint to allow us to set its Angular Velocity, Assuming our joints have vectors pointing down the Y axis.

A straightforward approach would be to take the angle from the current vector to where we need to be ( Reference Point ) and set that as the Angular Velocity of the joint. The difference between where we are and where we need to be is called the Error.

$$Error = Reference - Current \quad (1)$$

$$Solution = Error \quad (2)$$

We are not guaranteed to be able to reach the reference in one time step. The time taken to get from where we are to where we want to be is called "Deadtime". Our goal is to minimise the deadtime. The solution above will work if the reference never changes. However the destination rotation may move in the next time step. For this reason we need to multiply the calculated AngularVelocity by a constant called the P (Proportional) constant.

$$Solution = Error * P \quad (3)$$

Introducing P produces a second problem. It may overshoot the reference point.

To overcome this we introduce another factor. The rate of change of the Error value. We multiply this rate of change by the constant D ( Derivative ) which acts as a damping parameter on P.

$$Solution = (Error * P) + (ErrorsRateOfChange * D) \quad (4)$$

This is what we use in a PD ( Proportional Derivative ) controller.

Commonly used in engineering is the PID controller. The I constant is the constant we multiply the sum of Errors ( Integral of Errors ) by. Adding the sum of errors can make the controller more responsive to sudden changes in the reference point.

### 5.2.3 fuzzyControl Fuzzy Logic Controllers

Any small amount of research into systems theory will introduce us to Fuzzy Logic. For the purpose of experimentation we will compare the performance of traditional PID controllers with that of Fuzzy Logic controllers.

[L.A.Zadeh 1965] first introduced the concept of fuzzy sets in 1965.

Fuzzy controllers have been used to solve problems in Systems Theory where classical PID controllers fail. They can be more robust than PID controllers. PUT A REASON HERE.

None of the previously mentioned high level controllers use fuzzy logic controllers. [Margaliot and Langholz 2000] mentions the development of fuzzy logic in three different directions.

- Logic Theory  
In which Fuzzy Logic is investigated as a form of multi-valued logic.
- Artificial Intelligence ( AI )  
In which fuzzy logic is used as a tool for handling uncertainty.
- System Theory  
In which fuzzy logic is used for modeling and control using rule-based systems.

Numurus works exist in the area of animation which use Fuzzy Logic for Artificial Intelligence.

We are more interested in its use in System Theory. As a low level controller for the joint rotations of our skeleton.

At this point it is important to mention that [Pinto and Alvares 2006] proposes the use of fuzzy sensors in games AI control, using fuzzy sensors to invoke a state machine behaviour.

A good analogy of the reason for fuzzy logic can be taken from [Margaliot and Langholz 2000]. Using boolean logic we can state that.

Everyone over five ft is tall  
Everyone under five ft is short

According to boolean logic, the rules above state that you cannot be both tall and short. Anyone 4.9 ft tall is short.

The rules above, according to fuzzy logic state that you can be both tall and short, how much you are of each is what becomes important.

Keeping the previous example in mind consider the following.

Height = 2  
Height is over five ft  
Height is under five ft

In boolean logic the above statement would be.

Height = 2  
Height is over five ft ( FALSE or 0 )  
Height is under five ft ( TRUE or 1 )

In fuzzy logic the above statement might be.

Height = 2  
Height is Tall FALSE or ( 0.2 )  
Height is Short TRUE or ( 0.8 )

Now consider the following statements.

Height = 2  
if Height is Tall then Crouch  
if Height is Short then Jump

Here we have a response to the condition. Again in boolean logic the rules would evaluate to one or the other ( TRUE or FALSE ). The result would be a Jump.

Fuzzy logic might evaluate to a 0.2 of a Crouch and 0.8 of a Jump. The result would be a mix of the two.

In the example above Height is known as the Linguistic Variable or Universe Of Discourse. Tall, Short are sets within that Universe of Discourse.

The above analogy serves as an introduction to fuzzy logic. We will now look at creating the rules needed to construct a controller to replace the traditional PID Controller.

A problem commonly used to demonstrate a Fuzzy control system is that of the inverse pendulum. [Wang 1997] GET THIS REFERENCE Solving the inverse pendulum problem is computing the AngularVelocity required to rotate our joint to a Reference angle in the shortest time.

Consider the following situation. An inverse pendulum is attached by a hinge joint to a moving platform.

Below are the Fuzzy Logic rules to balance this inverse pendulum.

if Angle is +High and AngularVelocity is +High then Velocity = -High  
if Angle is -High and AngularVelocity is -High then Velocity = +High  
if Angle is +High and AngularVelocity is -High then Velocity =

Zero

if Angle is -High and AngularVelocity is +High then Velocity = Zero

Balancing an inverse pendulum is a common example given of the use of fuzzy logic.

Consider the same inverse pendulum only we have a motor on the hinge joint.

The problem becomes trivial.

if Angle is High then AngularVelocity = -High  
if Angle is -High then AngularVelocity = +High  
if Angle is Zero then AngularVelocity = Zero

When implementing the controller we further simplify this rule to.

if Absolute Angle is High then Angular Speed = High  
if Absolute Angle is Zero then Angular Speed = Zero

This should not be understood as a direct Linear relationship between the Absolute Angle and Angular Speed. The relationship between these two variables is dependant on how there Sets are defined, it is also dependant on the Inference of the rules and the Defuzzification of Angular Speed.

#### 5.2.4 Set Membership

Evaluation of a fuzzy rule is done by working out the membership value of a variable to one or more of its sets. The membership value is a Real number from 0 to 1 directly proportional to how far the fuzzified variable is within a set. Using our first example.

Height = 2

Height is Tall ( 0.8 ) ; - Membership Value

Height is Short ( 0.2 ) ; - Membership Value

An intuitive way to visualise the Linguistic Variable/Universe of Discourse is in a graph like the one below.

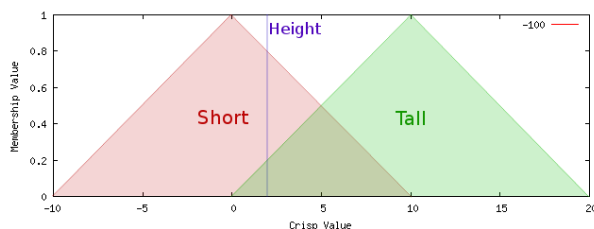


Figure 3: Universe Of Discourse

The area under the red curve is the "Short" set. The area under the green curve is the "Tall" set. The blue line is our crisp value of Height, which is 2.

Both sets are defined by a singleton and a linear curve.

- The "Short" set is defined by the singleton 0 and the linear curve ( 0, 10 ) .
- The "Tall" set is defined by the linear curve ( 0, 10 ) and the singleton 10 .

All variables must have a lower and upper bound. In this case the lower bound is 0 and the upper bound is 10 .

Height contains the sets.

- 'Short'
- 'Tall'

Action contains the sets

- 'Jump'
- 'Crouch'

It is possible to change the curves which define the sets.

To convert our numbers to something meaningful in fuzzy logic we have to fuzzify them. In the previous examples

Height = 2

is the fuzzification of the variable 2.

- Fuzzification is assigning a crisp value to an input variable.
- Defuzzification is retrieving a crisp value from an output variable.

Height.fuzz(2) # Assign our crisp value 2 to the Input Variable Height

... # Evaluate our rules

result = Action.defuzz() # Retrieve a crisp value from the Output Variable Action

#### 5.2.5 Hedges

Fuzzy Logic also defines Hedges for fuzzy rules. They allow us to write rules in the following manner.

if Height is very tall then crouch

Notice the word "very" in the rule. This is a hedge. Hedges are operators that modify the fuzzy values. They allow us to be more or less precise in our rule definitions.

At this point we should now have a rudimentary understanding of the theory from which our low level controllers are based.

## 6 A framework for a physically based character animation system. What is needed and what it must provide

We will now discuss what is required of a framework for the development of these high and low level controllers.

Currently there is no standard tool for the development of physics behaviours for use in computer animation. The following is an analysis of what is currently available and what we deem is necessary for the development of such a framework.

[DANCE ] is a c++ behaviour development environment which implements the controllers presented in [Petros Faloutsos and Terzopoulos 2001]. Dance is comprised of an OpenGL viewport and a modular framework. Support for ODE is built in. Behaviours are compiled then dynamically loaded as plug-ins. Programming in C++ has the advantage of speed. However prototyping in c++ can be very slow. Dance has minimal support for scripting in Python and proves to be a promising tool. Although it's lack of an acceptable compilation setup makes it difficult to compile.

[Euphoria ] is a commerical product from Natural Motion for the development of realtime behaviours. It is modular and capable of supporting commerical physics SDKs such as [Havok ] and [Physx ] It is however a commerical product and the average researcher cannot obtain it.

We will now discuss what is required of an environment for developing behaviours.

## 6.1 A 3D Animation environment

Ideally it should provide a rich 3D Animation environment. Support for setting and getting transformation information is essential. Clear visualisation is also essential.

## 6.2 Physics solvers - Animating Weight and Physical Correctness

It should provide a reasonably fast Physics solver and the behaviours should have the ability to apply force and torque impulses to bodies affected by the solver. Behaviours should also have the ability to query Linear and Angular Velocity information about these bodies. The physics solver should have support for constraints between rigid bodies to act as joints. Although not essential it should be possible to set stops on the degrees of freedom constrained by these constraints. A very useful although also not essential tool is the support for motors, which provide the ability to set the angular velocities on constraints directly.

MENTION MORE ABOUT MOTORS

## 6.3 Cognitive Modelling - The Acting

Additional tools for intelligence systems and soft computing, Fuzzy Logic, Neural Networks and Pattern Recognition can prove useful although again they are not essential.

# 7 The Implementation Of Our Framework

Rather than reinvent the wheel by building a stand alone tool for the development of behaviours.

We suggest a approach in which our development framework is itself a plug-in for a generic animation framework ( Maya ) and behaviours are developed in a higher level scripting language. This has two main advantages. It means that a stable generic animation framework does not need to be developed by the researcher, Working in a higher level scripting language will remove the slow Compile, Load, Cycle. Errors such as the Zero division error will not crash our development framework.

Our framework is called Luya.

## 8 Choice Of Tools

The three programming languages currently available for extending Maya are MEL, Python and C++ . Python was the programming language of choice for the following reasons.

- It supports Object Orientation
- Python has access to both Maya's C++ and MEL Api at the same level. For those familiar with both MEL and the C++ Api, this python example is legal in MayaPython.

```
MMatrix( toMatrix( xform(object,q=N,ws=N,matrix=N) ) )
```

Where MMatrix is a class belonging to the C++ api and xform is a utility function belonging to the MEL interface.

- Python has bindings to the Open Dynamics Engine, A free physics Library.

Before the release of the third party MayaPython plugin this option was not available.

## 8.1 ODE and Bullet

Currently two truly Open Source Physics SDKs exist.

- ODE
- Bullet

### 8.1.1 ODE

- ODE was our second choice of Physics SDK. It has all the features necessary for developing our behaviours.
- Although the ODE manual is very comprehensive, documentation on its Motors and Joint Limits is sparse.
- Development on ODE is very slow and support is non existent.
- Although incomplete, support for Python all ready exists in the form of [pyODE ]

### 8.1.2 Bullet

- Bullet was our first choice of Dynamics Engine. Primarily because it is newer and evolving at an astonishing rate.
- Support is provided by a vibrant [Bullet ] "Community" and the response time to questions are generally very fast.
- However Bullet lacks the Motor support provided by ODE.

## 8.2 Speed VS Development time Tradeoff

It is not necessary for our behaviours to run in real time. So we can sacrifice speed for development time. We bake our simulations into keyframes in Maya.

## 8.3 The Structure Of Our Framework

The following section describes the structure of our framework.

Character physics rigs are setup in Maya using special bounding shapes provided as part of the framework. Traditional rigs can be constrained to these physics rigs.

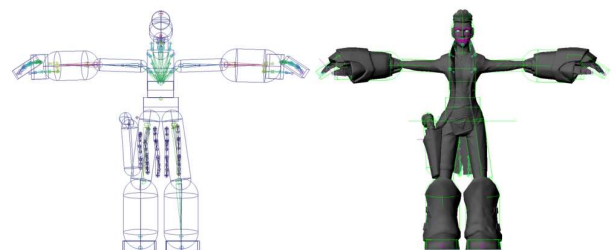


Figure 4: "Luya Physics Rig"

When simulation starts Luya serialises the physics rigs into a form understood by pyODE/pyBullet.

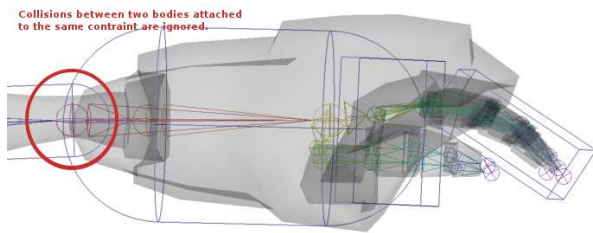


Two interface modules exist to do this.

- LuyaODE
- LuyaBullet.

On initialization the average behaviour creates an instance of the Luya World() class which searches the scene for physics rigs and serializes them into ODE.

On each step of the simulation Luya deserialises the information from pyODE/pyBullet into transformation keys in Maya. Creating a passive collision object is done by setting the objects mass attribute to 0 in Maya. Transform information for passive collision objects is sent from Maya to pyODE/pyBullet on every time step. Collision sets are supported by maintaining a table of object relationships. When two objects collide, our physics solver calls a custom collision callback function which looks for the two objects in our collision sets table. If they are in the same group the collisions are ignored. Luya automatically puts collision objects connected by a constraint into the same collision set. This allows us to overlap our collision objects within a physics rig.

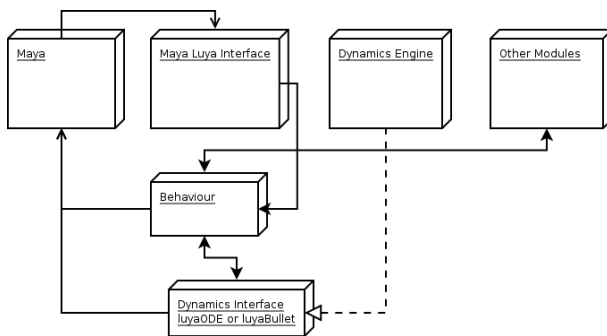


**Figure 5:** "Collision Sets"

Behaviours are modules which Luya calls on every time step of the animation.

The behaviours can interface with both Maya and the interface modules. Currently only one of the two modules can be used per behaviour. It would not take much effort to change this and allow multiple physics engines to interact in the same environment.

A standard construct can be visualised in the following way.



**Figure 6:** "The Luya Framework"

Any behaviour can use third party modules to extend its functionality. For example the behaviour could use a mocap or midi module.

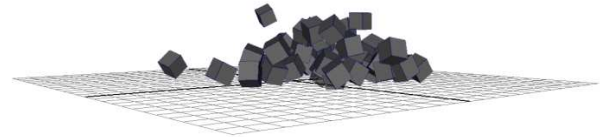
## 8.4 Summary

We now have a framework with which we can develop behaviours.

## 9 Example Behaviours Developed Using This Framework

The following section will present some of the behaviours we developed within our framework. Considering the time restrictions on this project we are only able to develop the simpler low level behaviours. Given more time it would be possible to develop more complex high level behaviours. For consistency all of the following behaviours use the LuyaODE interface although it should be trivial to create similar behaviours with the LuyaBullet interface.

### 9.1 Default Behaviour

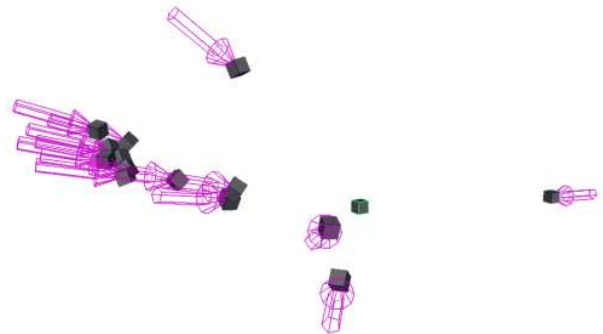


**Figure 7:** "Default Behaviour"

Firstly we present the default behaviour. This provides the minimal set of instructions required for a simulation to work correctly.

The Source Code for the Default Behaviour can be found in the Appendix.

### 9.2 Flocking with Fuzzy Logic



**Figure 8:** "Simple Flocking Behaviour"

This next behaviour displays flocking with fuzzy logic. The Inertia of each Follower and the Collision detection keeps them from ever occupying the same space. Whilst designing this behaviour we discover that the fuzzy rules themselves are very simple to infer. What becomes difficult is how we define our sets. This can become increasingly complex as we try to model the area that we want the followers to stay in. The bulk of behaviour development is not in the rules but in accurately defining our sets.

The Source Code for Simple Flocking Behaviour can be found in the Appendix.

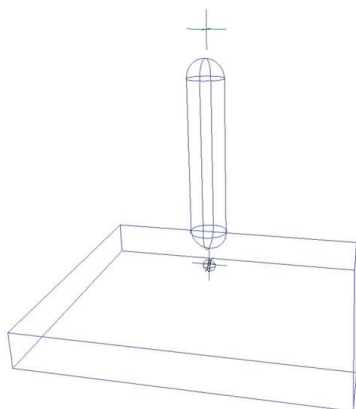
### 9.3 The Inverse Pendulum Controller

We will now consider what must be taken into account when designing a straightforward inverse pendulum controller. The traditional approach to presenting the inverse pendulum problem is with

a 2 Dimensional Inverse Pendulum on a moving platform. The controller must calculate the linear velocity of the moving platform required to balance the inverse pendulum. Considering our research is geared towards physically assisted character animation we presume that calculating the angular velocity required to reach a designated rotation in the shortest time is a more related problem. Therefore our controller will calculate the Angular Velocity of the Pendulum itself required to balance it. Balancing will be the act of reaching the direction denoted by the up vector  $(0, 1, 0)$ .

We present two slightly different approaches to designing the inverse pendulum controller.

- The first is with a traditional PD controller.
- The second is a fuzzy logic controller.



**Figure 9:** "Inverted Pendulum Controllers"

### 9.3.1 Using a PD Controller

The Source Code for PD Inverse Pendulum Controller Can be found in the Appentix.

### 9.3.2 Using a Fuzzy Logic Controller

The Source Code for Fuzzy Inverse Pendulum Controller Can be found in the Appentix.

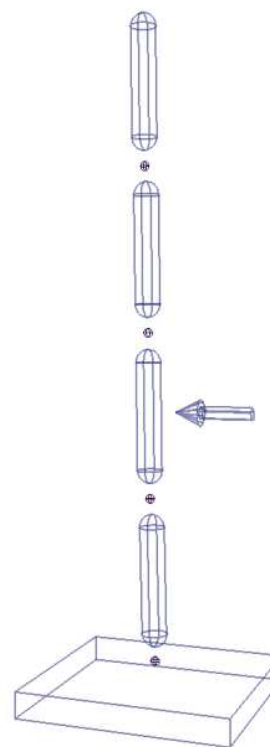
### 9.3.3 Summary

The PD Controller proves to be simpler to implement than the Fuzzy Controller. It runs quicker and uses less resources. The Fuzzy controller however is more stable.

## 9.4 Chain of Inverse Pendulums

The Chain of Inverse Pendulums is a straightforward extension to the Inverse Pendulum problem, in both controllers however an instability is introduced. The lower links are less stable than the higher ones. This same problem has been noted by both [Zordan 2002] and [Paul Wrotek and College 2006]. The solution presented by [Zordan 2002] is to scale the torque of these joints by the effective moment of inertia of the chain of bodies affected by each joint. [Paul Wrotek and College 2006] suggests another approach in which torque forces are calculated by world space transforms rather than local space rotations.

The following are the two behaviours we present for this problem. Both in fuzzy logic and using the conventional PD controller.



**Figure 10:** "Chain Of Inverted Pendulums"

### 9.4.1 Using PID Controller

The Source Code for PID Chain of Inverse Pendulum Controllers can be found in the Appentix.

### 9.4.2 Fuzzy Controller

The Source Code for Fuzzy Chain of Inverse Pendulum Controllers can be found in the Appentix.

## 9.5 Balancing a Double Inverse Pendulum

We presume that with the knowledge gained from balancing a Double Inverse Pendulum we are able to balance the leg of a humoid character. We are unable to find a solution for Balancing a Double Inverse Pendulum with either the PD or the Fuzzy controller, however both [R. C Fitzpatrick and McCloskey 1992] and [Plamen Gatev and Hallet 1999] suggest that balancing can be acheived through controlling the stiffness of the ankles of a humanoid, rather than the knees. Therefore we move on to the Humanoid Directly.

- Using a PD Controller.
- Using an Inverse Pendulum Controller.

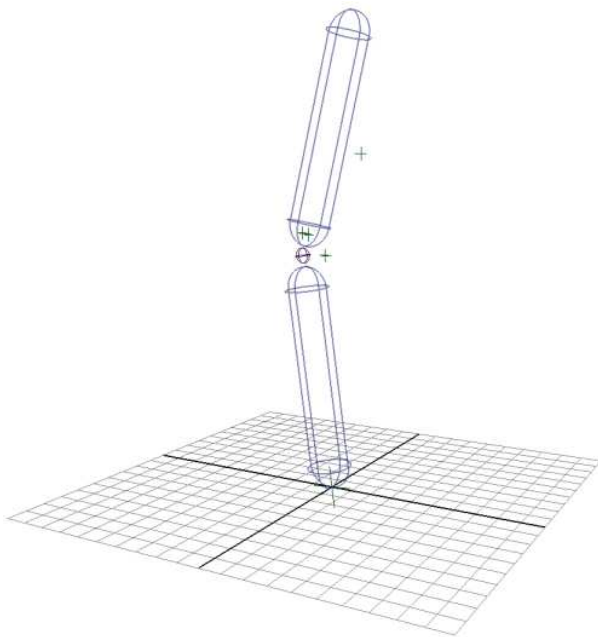
## 9.6 Balancing a person

Several approaches are taken for balancing the Humanoid.

These include.

- Stiffening the the lower leg joints to tend to a particular orientation then applying forces to the upper body to keep it about its projected centre of gravity.
- Stiffening the upper joints.





**Figure 11:** "Pendulum and Inverse Pendulum"



**Figure 12:** "Balancing A Ragdoll Character"

Further approaches are yet to be tried. Amongst these are the following.

- Apply a force in the opposite direction to the lean of the character as it falls over. A PID or Fuzzy controller could be used to determine the amount of force needed.
- A novel approach to balance presented by [Paul Wrotek and College 2006] and mentioned earlier on in this text, is to attach a Weak Root Balance Spring to the root ( hips ) of the character. A weak constraint which is effectively floating in space. The floating constraint's coordinate frame can be modified directly.

## 9.7 Summary of Low Level Controllers

### 9.7.1 Fuzzy Advantage

The advantages of using Fuzzy Logic controllers

### 9.7.2 The Advantages of Using PID Controllers

They were not so useful when it came to controlling complex scenarios. Scenarios which involved mixing two goals, like the Double Inverse Pendulum problem. Using a PD controller to reach our destined rotation proved useful, they were very responsive, elegant and intuitive.

Fuzzy controllers are too bulky and cumbersome, also too slow.

We can mix both PD and Fuzzy Logic Controller for best results.

## 9.8 Summary

The previous behaviours demonstrate the controllers and theory mentioned at the beginning of this article. Using our development framework we are able to directly develop controllers for use in animation. Certain behaviours present a fuzzy logic approach to the low level controller theory. Our use of PID controllers demonstrates the traditional approach of low level controller design. We have introduced an innovative use of Fuzzy Logic in animation controller design.

## 10 Refactors Due to Behaviour Development

Several refactors of the framework were executed due to problems discovered during behaviour development. One example of such a problem was when it came to applying the physics rig to the main character. Originally the physics were keying world space transforms on both the joints and the rigid bodies. When it came to orient constraining the Maya rig to the physics rig this became an issue because the joints in the physics rig were translating and not rotating. To overcome this problem the system was refactored. Joints and unconstrained rigid bodies became the only transforms to have their keys set by the Physics Engine. Joint's local transforms were keyed relative to their child transforms and not the parents as is traditionally the case. This allowed the joints to orient correctly not only aiming at the child, but also rolling with it.

## 11 Further Possible Experiments Using This Framework

Balancing of a person would be achievable within the current framework. Using forces to push the character around.

A more complete set of higher level behaviour tools.

## 12 Optimizations

It is currently not possible to write realtime behaviours within the current framework. Optimisations are necessary if behaviour work on rigs with many degrees of freedom is to be done. We were able to briefly investigate two ways of optimising the running time of our behaviours.

### 12.1 Psyco

The first and easiest way to increase performance was to use Psyco. A python optimisation module. Psyco uses certain tricks to try to optimise the runtime of python scripts, one of which is to dump several versions of the same functions which take different data types as arguments. We found that including psyco in our behaviours with the following two lines of code:

```
import psyco
psyco.full()
```

Sped our behaviours up considerably, however our fuzzyRules module will not work with Pysco. We believe this is due to Pysco's treatment of Python docstrings. A feature which fuzzyRules depends upon heavily.

## 12.2 cFuzzy

The pyFuzzy module is one of the major bottlenecks in our fuzzy behaviours. Pyfuzzy was rewritten entirely in C++ ( cFuzzy ) and an attempt was made to wrap it as a python module both with [SWIG ] and [Boost ] Python. Technical difficulties with SWIG lead us on to use Boost Python for the final version of cFuzzy. However we are unable to report a considerable increase in performance with cFuzzy. Although unproven we believe this may be because cFuzzy relies of slow Pythonic data types provided by Boost Python.

## 12.3 C++ API

We suggest that the use of pyODE and pyBullet rather than the direct C and C++ interfaces may also be a bottleneck in the performance of our framework. We propose a lower level framework which handles communication between Maya and the Physics solver. It should provide a high level interface to python with which behaviours can influence the animation. This lower level interface would communicate on a direct level with Maya, ODE and Bullet's C++ APIs.

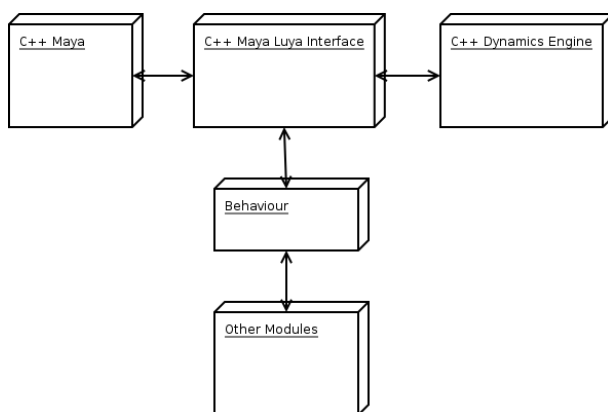


Figure 13: Proposed Optimised Framework

# 13 Analysis Critical Analysis

We critically analyse our framework and our approach to behaviour development.

## 13.1 What works

Tests using this dynamics framework show a greater level of stability than Maya 8.0's own dynamics engine. PROVE IT. We are pleased with the choice of tools and the general design of the framework although a shorter running time for behaviours would be favourable in the future. The system is very easy to use and very stable, it is heavily integrated with the Maya workflow. In the current framework, locators can be placed about the scene as markers for testing. Full use of riging tools such as constraints and expressions can be used on force locators this helps to minimise the need for the behaviour engineer to develop his/her own supporting debugging utilities. We are also pleased with the number of useful tools

which were developed to support this project. pyBullet is the first Python binding available for Bullet and will most likely become a common way to use the Bullet library as pyODE has with ODE. pyBullet is currently hosted on sourceforge as part of the [Biscuits ] project.

## 13.1.1 First Version

The initial version of this framework with far less functionality was written in C++ as a plugin, prototyping in C++ was slow so the decision was made to move over to a higher level language.

## 13.2 How Luya might be improved

### 13.2.1 More Behaviour Development

Using the current framework it would be possible to develop a balancing behaviour as proposed by [Zordan 2002], [Paul Wrotek and College 2006] and [Petros Faloutsos and Terzopoulos 2001], a greater understanding of biomechanics and robotics may be required to develop such a behaviour.

### 13.2.2 A Better Front End

Although our framework is heavily integrated with Maya our tools could be better improved with a more in depth graphical user interface. This interface could give the animator direct control of the simulation timestep.

### 13.2.3 Assignable Behaviours

Displaying the behaviours as DAG nodes might also improve usability. This would allow us to assign behaviours to specific objects within the outliner. It might also allow us to store behaviours within a Maya scene along with the assets it required.

### 13.2.4 Better visualatation of motor strengths and joint limits

It is difficult to visualise the orientation of joint limits and motor strengths. An OpenGL visualisation representing these joint limits would greatly improve debugging of behaviours.

### 13.2.5 More Behavioural Modules

A greater number of modules designed specifically for behaviour engineering would also ease the development of behaviours. Third party modules for pattern recognision such as [Pyro ] are currently available online.

Given more time, work might have gone into developing an implementation of the hierarchical state machine of composable controllers presented by [Petros Faloutsos and Terzopoulos 2001] along with an experimental fuzzy logic extension to that controller to allow multiple behaviours to influence the same constraints at the same time.

### 13.2.6 Blending and Trajectory Tracking

Another feature which would be of great help to animators is blending between FK animation and Dynamically driven animation. This requires a certain element of trajectory tracking to allow the dynamic joints to inherit the same inertia as the FK joints.

### 13.2.7 Floating Joints

At this moment in time there is limited support for floating ( meat hook ) joints. These joints are unable to move during simulation. Support for moving meat hook joints would allow us to implement the Weak Root Balance Spring controller mentioned previously [Paul Wrotek and College 2006].

### 13.2.8 Support for Physx

Support for the Ageia Physx SDK might prove useful when Ageia's hardware accelerated physics cards become more commercially available.

## 14 Summary

I am pleased with the tools which have been developed for this project. A number of them are useable in there current state. I would have liked to experiment with developing more behaviours. The behaviours mentioned previously served as testing sub projects, each introduced new problems with the design of the framework.

Although unsuccessful in developing a balancing behaviour I have been successfull in developing a stable framework from within which a balancing behaviour could be constructed.

I have learn't a great deal about the current techniques and tools available for behaviour development. For that reason I believe this has been a success.

## Acknowledgements

Special thanks to Chris Roberts and Ari Sarafopoulos. Also thanks to Sebastian Huart for the use of his character rig.

## References

Game programming wisdom.

BISCUITS. <http://sourceforge.net/projects/biscuits/>.

BOOST. <http://www.boost.org/libs/python/doc/>.

BRICE CRISWELL, KARIN DERLICH, D. H. 2006. Sketches: rigging the game: Davy jones' beard: rigid tentacle simulation. *SIGGRAPH*.

BULLET. <http://www.continuousphysics.com/bullet/phpbb2/index.php>.

DANCE. <http://www.magix.ucla.edu/dance/>.

DROIDLOGIC. <http://www.droidlogic.com>.

ENGINE, T. O. D. <http://www.ode.org/>.

EUPHORIA. <http://www.naturalmotion.com/euphoria.htm>.

GENG, T., AND BERND PORR, F. A reflexive neural network for dynamic biped walking.

HAVOK. <http://www.havok.com/content/view/285/79/>.

J.K. HODGINS, W. L. WOOTEN, D. C. B., AND O'BRIEN., J. F. 1995. Animating human athletics. *SIGGRAPH*.

L.A.ZADEH. 1965. Fuzzy sets. *Information and Control*.

MAMDANI, E.

MANDEL. Versatile and interactive virtual humans: Hybrid use of kinematic and dynamic motion synthesis.

MARGALIOT, AND LANGHOLZ, G. 2000. *New Approaches to Fuzzy Modeling And Control - Design and Analysis*. World Scientific.

MASSIVE. <http://www.massivesoftware.com/>.

MCCUSKEY, M. Fuzzy logic for video games.

MCGEER, T. 1990. Passive dynamic walking. *International Journal of Robotics Research*.

NATURALMOTION. <http://naturalmotion.com>.

PAUL WROTEK, ODEST CHADWICKE JENKINS, M. M., AND COLLEGE, W. 2006. Dynamo: Dynamic, data-driven character control with adjustable balance. *SIGGRAPH*.

PETROS FALOUTSOS, M. V. D. P., AND TERZOPOULOS, D. 2001. The virtual stuntman - composable controllers for physics-based character animation. *SIGGRAPH*.

PHYSX. <http://www.ageia.com/developers/downloads.html>.

PINTO, H., AND ALVARES, L. O. 2006. *Constructing a Goal-Oriented Robot for Unreal Tournament Using Fuzzy Sensors, Finite-State Machines, and Behavior Networks : Game Programming Gems 6*. Charles River Media.

PLAMEN GATEV, SHERRY THOMAS, T. K., AND HALLET, M. 1999. Feedforward ankle strategy of balance during quite stance in adults. *Journal of Physiology*.

PYODE. <http://pyode.sourceforge.net/>.

PYRO. <http://pyrorobotics.org/?page=pyromoduleneuralnetworks>.

R. C FITZPATRICK, J. L. T., AND MCCLOSKEY, D. I. 1992. Ankle stiffness of standing humans in response to imperceptible perturbation: Reflex and task-dependent components. *Journal of Physiology*.

ROBERTS, C.

SCHEEPERS, F., AND WHITTOCK, A. 2006. The wrecked road in cars - or how to damage perfectly good geometry. *SIGGRAPH*.

SWIG. <http://www.swig.org/doc1.3/python.html>.

SYSTEM, F. F. X. G. <http://www.finalfantasy12.eu.com/>.

THOMAS, F., AND JOHNSTON, O. The illusion of life.

WANG, L. X. 1997. A course in fuzzy systems and control. *Prentice Hall*.

WOOTEN, 1996. Animation of human diving - using an artist to provide (key frames) / (pose targets ) for the pd controllers to get to.

ZORDAN. 2002. Motion capture-driven simulations that hit and react. *SIGGRAPH*.