Generative Animation in a Physics Engine using Motion Captures

Brian Wilke and Sudhanshu K. Semwal
Department of Computer Science, University of Colorado, Colorado Springs, CO, 80918, U.S.A.

Keywords: Articulated Motion, Motion Capture, Novel Application, Asymmetric Scaling, Under Controlling.

Abstract: Motion captures are an industry standard for producing high-quality, realistic animations. However, generating novel animations from motion captures remains a complex, non-trivial problem. Many techniques have been developed, including kinematics and manually solving the equations of motion. We present a new technique using a physics engine to generate novel animations. Motion captures are effectively simulated within a popular open-source physics engine, Bullet, and two generative techniques are applied. These generative techniques – asymmetric scaling and under-controlling – are shown to be simple and straight-forward. The techniques and methods were implemented in Python and C++, and show new promising avenues for generative animation using existing motion captures.

1 INTRODUCTION

Motion capture has become an industry standard for producing high-quality, realistic animations in film and games (Schreiner and Gleicher, 2002; Calvert et al., 2002; Multon, 1999; Boulic et al., 1990; Hodgins, 1996; Macchietto et al., 2009; Ryan, 1990), and pivotal papers describing a general purpose rigid body collision and contact simulator (Baraff, 1989; Baraff, 1991). However, once motion capture data is sampled, it is difficult to adjust and re-target. A common example of this distortion is footskate, where the character's feet move when they really should be planted (Schreiner and Gleicher, 2002). Furthermore, what if the director wants the right arm above the head instead of on the waist, and the left leg bent instead of straight? A new motion capture must almost certainly be taken, since adjusting the original capture would be too time-consuming, expensive, and possibly unrealistic.

1.1 Generating Novel Animations with Motion Captures

Adjusting motion captures has been a continuing field of study, especially since captures have inherent obstacles to overcome, e.g., footskate. A main question has been whether a capture or set of captures can be used to generate novel animations. For instance, given a particular walking capture, can we adapt it to a different skeleton walking at a different speed?

This particular type of generative animation is called walking motion synthesis. There have been three main previous research directions in this area, presented in (Multon, 1999): kinematics, solving the equations of motion, and interpolation.

Kinematics is the geometry of motion. This approach specifies key joint angles that determine the rest of the model's configuration. This method can generate various synthesizers, as in (Boulic et al., 1990). However, as in all kinematics, it can require massaging from the animator to provide physical realism and avoid mechanical movements.

Solving the equations of motion attempts to find the correct forces and torques to apply to reach a certain position. (Hodgins, 1996) models humans running physically and directly computes the forces and torques required for running (Hodgins, 1996). Controllers have to be implemented to avoid impossible joint configurations. In (Macchietto et al., 2009), a real-time simulation system automatically balances a standing character, deriving the mechanics of balance from linear and angular momentum. Interpolation edits motion capture data between given motions. (Ashraf and Wong, 2000) interpolate walking and running motions and generate a new motion between four given motions (Ashraf and Wong, 2000). (Rose et al., 1998) use multivariate interpolation on classified motion, generating a new animation by applying polynomial and radial basis function interpolation between B-Spline coefficients under the classification (Rose et al., 1998). Extrapolation is not pos-

Wilke B. and Semwal S.
Generative Animation in a Physics Engine using Motion Captures.
DOI: 10.5220/0006134702500257
In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017)*, pages 250-257
ISBN: 978-989-758-224-0

sible with these techniques, and the parameters can be quite difficult to properly set. PCA has also been introduced into animation synthesis. (Glardon et al., 2004) use principal component analysis (PCA) to create a walking engine based on motion captures (Glardon et al., 2004). This engine computes the walking cycle for different skeleton sizes at different speeds.

Controlling Independent Rigid Bodies. Let the rigid body for joint j be $r_b(j)$. At each frame f, a joint's rigid body $r_b(j)$ has a linear velocity $\mathbf{v}(r_b(j), f)$ computed by the physics engine. This velocity is computed given the forces acting on $r_b(j)$, e.g., gravity. Our target linear velocity from the motion capture for joint j in the next frame $f+1$ is $\mathbf{v}(j, f+1)$. The error term between our current and target velocities is then

$$\mathbf{e}_v(j,f) = \mathbf{v}(j,f+1) - \mathbf{v}(r_b(j),f), \qquad (1)$$

where j is the joint, f is the frame, $\mathbf{v}(j, f+1)$ is our calculated linear velocity in the next frame, and $\mathbf{v}(r_b(j), f)$ is the physics engine's computed linear velocity on the rigid body $r_b(j)$ in f. This error term allows us to control the rigid body $r_b(j)$'s linear velocity in the presence of other forces.

A proportional control loops uses only a proportional term for the next output, that is, $P_{out} = K_p e(t)$, where P_{out} is the loop's output at time t, $e(t)$ is the system's error at t, and K_p is the proportionality constant. Using this kind of control loop, between frames f and $f+1$, we apply a central force to each rigid body

$$\mathbf{F}(j,f) = K_f \cdot m(r_b(j)) \cdot \mathbf{e}_v(j,f), \qquad (2)$$

where j is the joint, $r_b(j)$ is the rigid body corresponding to j, $m(r_b(j))$ is $r_b(j)$'s mass > 0, K_f is an appropriately chosen constant, and $\mathbf{e}_v(j, f)$ is from Equation (1). This equation incorporates all forces acting on the body and applies a force that will correct its motion to our target motion from the motion capture.

Having developed simple controllers for independent rigid bodies, we can consider the problem of creating a physical skeleton. The motion capture joint hierarchy specifies the parents and children of each joint. Typically, it is a rooted tree where each joint has one unique parent. We will assume that is the case, such that $H : J \to G$ where G is a tree and $\exists j \in J$ such that j is the root of G.

Asymmetric Scaling and Under-controlling. Now that we have a physics engine that replicates the original motion capture's motion, we can consider generating novel animations using this data. Here, we will consider two types of generative animation: asymmetric scaling and under-controlling. These techniques can exist on their own or combine together to

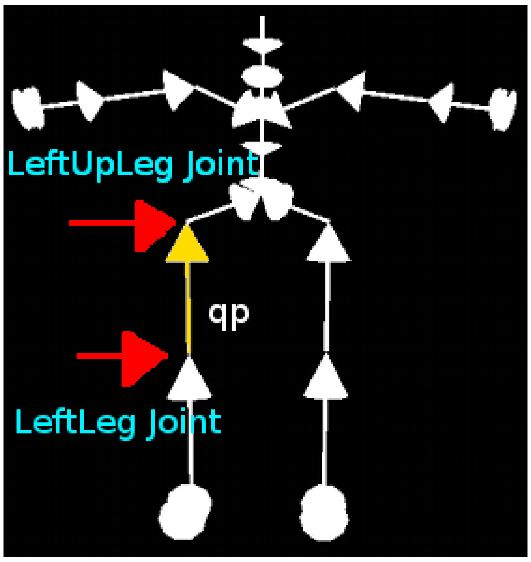

Figure 1: The various $\mathbf{q}_p(j)$ vectors between joints and parent joints in a motion capture hierarchy H. The LeftUpLeg joint and LeftLeg joint are marked in the picture, and the $\mathbf{q}_p(\text{LeftLeg})$ vector between them is highlighted. In this particular hierarchy, the LeftUpLeg joint is a parent of the LeftLeg joint.

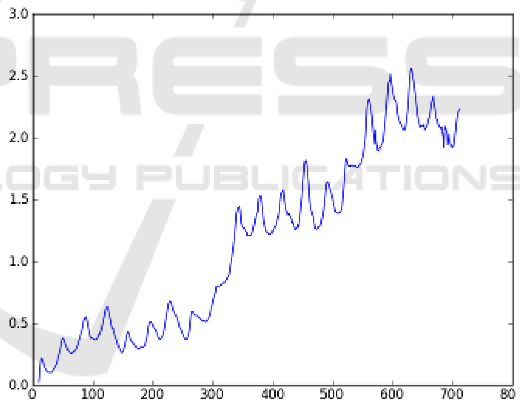

Figure 2: This graph plots $e_p^2(j, f)$ for $j = \text{Hips}$ against the frame numbers. Though there is some oscillation, the errors appear to accumulate over time, causing a left-to-right upwards slope in the graph.

generate new animations. Asymmetric scaling simulates what would happen to the original motion if, e.g., part of the body became stronger. For instance, consider a motion capture of a boxer shadow boxing. The boxer wants to increase the power of his punches. With asymmetric scaling, we can simulate what would happen to his shadow boxing if he increased his biceps muscle mass, or his shoulder muscle mass, or his abdominal muscle mass, etc. By including a proportional term,

$$\mathbf{v}_K(j,f) = K_v(j,f) \cdot \mathbf{v}(j,f), \qquad (3)$$

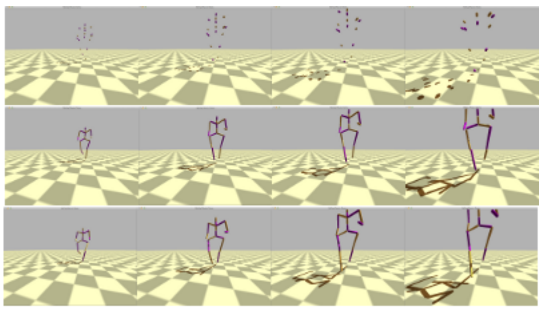
Figure 3: Simulated animation stills of 141_02.bvh using independent rigid bodies, a physical skeleton, and a constrained physical skeleton, respectively from top to bottom, using the tuned parameters.

5

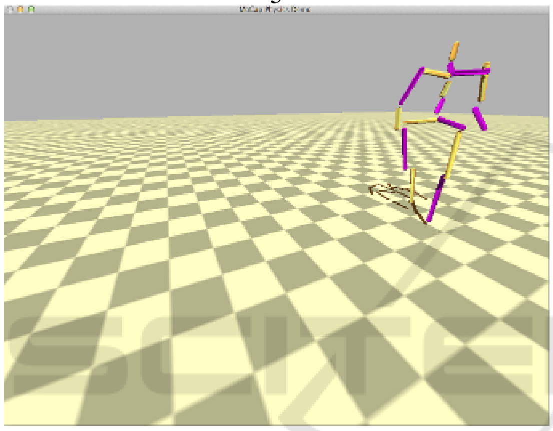
Figure 4: A still from the simulation of 141_30.bvh using physical skeletons after many steps. The error has accumulated enough that joints that should remain locked are drifting away from each other.

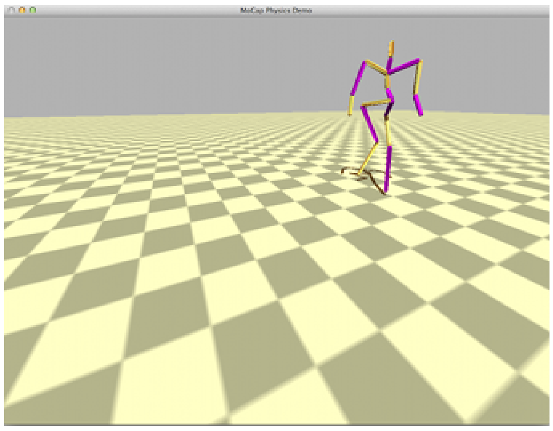
Figure 5: A still from the simulation of 141_30.bvh using constrained physical skeletons after many steps. The animation looks much better because the joints remain locked together.

we simulate a change in joint velocity, where $\mathbf{v}(j, f)$ is joint j's linear velocity in frame f. This velocity scaling also simulates what would happen if, e.g., a tennis player swung her racket harder or softer, or a cyclist pedalled harder or softer during a climb. That is, this approach can simulate a velocity change that is within a person's reach given their current physicality. A tennis player could utilize this technique to determine optimal swinging speed for a given ball, where the velocity change transferred to the racket changes the trajectory of the ball. The ball's trajectory change could also be physically simulated in our physics engine.

1.2 Under-controlling

In our previous techniques, we exactly specified a controller on every rigid body in the physical skeleton. With a physical skeleton operating under constraints, this exact specification is not required. Rigid body constraints determine how two rigid bodies can transform with respect to each other. As mentioned earlier, the shoulder is a type of ball and socket joint that limits translation away or towards the humerus and the shoulder blade, but allows rotation. This ball and socket constraint is also known as a point-to-point constraint. In general, 3d macroscopic rigid bodies have 6 degrees of freedom (DoF): translation along any axis, and rotation about any axis. A point-to-point constraint is a specialization where the translations are locked, i.e., there are only the 3 rotational degrees of freedom. A generic 6 DoF constraint can be created which limits or does not limit any particular DoF. Applying these constraints to each pair of rigid bodies in our physical skeleton allows us to under-control the skeleton. The physics engine will update each rigid body's position based on our control loop *and* the rigid body's constraints. For instance, we could control only the right side of the body, which would create a zombie-like effect where the left side is limply carried along. Our joints – the location where bones connect – have very well-defined constraints. In fact, dislocations occur when we violate these constraints. We have already touched upon the shoulder's constraint, but other major constraints include the elbow, the wrist, and the knee. For instance, as part of the elbow, the radius and ulna (the bones of the forearm) can rotate around the two axes perpendicular to the elbow, but not by more than the proximal semicircle defined by the axis of the humerus, and the humerus, radius, and ulna cannot translate with respect to each other. Most joints in the human body do not allow translations, but allow rotations. By specifying these constraints, we can free ourselves from controlling every rigid body.

2 ALGORITHM DEVELOPMENT AND IMPLEMENTATION

The code developed for this paper uses a mix of Python and C++, and utilizes the Bullet physics engine (Coumans, 2012). Only the BVH motion capture file format is supported, though other formats such as ASF/AMC could be added in the future as necessary. The intent is to work with the large CMU Graphics Lab Motion Capture Database and compare the original motion captures to the generated physics animation (Calvert et al., 2003; Club, 2013). Even though the CMU Database uses the ASF/AMC format originally, their captures have been exported to the BVH format, and BVH is more compatible with certain tools that sped development, such as ply, an implementation of lex and yacc in Python (Beazley, 2011; OpenGL, 2013; SciPy, 2013; Evans, 2011). There is no appreciable difference between the BVH files we use here and the original ASF/AMC files, except for the first frame which includes a T-pose of the capture's hierarchy.

Five motion captures from the CMU motion capture database were replicated physically and had generative techniques applied.

Replicating the Motion Physically and Positional Error. Our controllers depend on two variables: the numerical differentiation technique and the proportionality constant. We evaluated central differences with various orders, $\nabla(n)$, where n is the order and $1 \equiv n \bmod 2$. With each order, we also tuned the proportionality constant K_f to minimize positional error across all joints. Because we also scale our model in our simulation, there is a second tuning constant, K_{vc}, which scales our velocities symmetrically. That is,

$$\mathbf{v}'(j,f) = K_{vc}\mathbf{v}(j,f),$$

for all joints j in all frames f, where $\mathbf{v}(j,f)$ is joint j's linear velocity in frame f.

Our goal is to search for a central difference $\nabla(n)$ and a pair (K_f, K_{vc}) that minimizes differences between the original motion capture and the simulation. For a joint j in frame f, its positional error vector is

$$\mathbf{e}_\rho(j,f) = \mathbf{r}(j,f) - \boldsymbol{\rho}(r_b(j),f),$$

where $\boldsymbol{\rho}(r_b(j),f)$ is the position of the rigid body $r_b(j)$ that corresponds to joint j in frame f, and $\mathbf{r}(j,f)$ is j's position in the motion capture in frame f. In an independent rigid body, $\boldsymbol{\rho}$ is the rigid body's center of mass, whereas for a physical skeleton, it is at one end of the capsule shape we use for $r_b(j)$. The squared error is

$$e^2_\rho(j,f) = \mathbf{e}_\rho(j,f) \cdot \mathbf{e}_\rho(j,f),$$

which is just the dot product of $\mathbf{e}_\rho$ with itself. Averaging across all frames, we get

$$\bar{e}^2_\rho(j) = \frac{1}{|f|}\sum_f e^2_\rho(j,f),$$

where $|f|$ is the number of frames. Finally, averaging across all joints, we achieve an error number for each pair (K_f, K_{vc}) and central difference $\nabla(n)$

$$\bar{e}^2_\rho(K_f, K_{vc}, \nabla(n)) = \frac{1}{|J|}\sum_{j\in J} \bar{e}^2_\rho(j). \tag{4}$$

Minimizing Equation (4) gives us tuned parameters that closely match the simulation to the original motion capture.

Search Algorithm. There are many algorithms we could use to search for the parameters $(K_f, K_{vc}, \nabla(n))$ that minimize Equation (4). One rather simplistic approach is to generate random triples of numbers, calculate Equation (4) for each triple, and compare. This kind of search is known as a Monte Carlo search, and is very applicable when the domain of the triples is limited and it is difficult to obtain or apply a deterministic method that optimizes these parameters. Indeed, formulating a deterministic method appears extremely difficult, since, not only do we apply forces and torques to each rigid body, but Bullet too applies the force of gravity and constraint forces to each rigid body in each frame. We therefore used a modified Monte Carlo search. We limited our central differences to orders $3 \leq n \leq 19$. For certain (K_f, K_{vc}), the rigid bodies can literally fly apart in the simulation, so we seed the search initially with a known good pair, (K^g_f, K^g_{vc}). Algorithm 1 details the search method, where random$[-1,1]$ is a function that produces a random real number, i.e., a floating point number, between -1 and 1 inclusive. It can easily be seen that this algorithm runs in $O(\text{points} \cdot \text{maxOrder} \cdot \text{F(mocap-file)})$, where F is the number of frames in mocap-file.

We found that a good starting point for independent rigid bodies was $(K^g_f, K^g_{vc}) = (100, 75)$, and for physical skeletons was $(K^g_F, K^g_{vc}) = (20, 100)$. These parameters produced a simulation that at least somewhat matched all of the captures. We then tuned to a single motion capture, 141_15.bvh. This capture is by far the longest of our 5, and demonstrates many interactions between joints by going through ranges of motion. From there, we used 100 points per run, each run taking about 14 minutes on a laptop, and a maximum order of 19. We then used the best tuning parameters from each run to seed the good values of the subsequent run. A concern with this particular method is that *runs* will not converge upon a better solution. However, we found that, starting with the

Table 1: Tuned parameters for independent rigid bodies and physical skeletons.

Rigid Body	n	K_f	K_{vc}	$\bar{e}^2_\rho$	σ^2_ρ	points	Runs
Independent Rigid Bodies	5	98.5331	85.46	0.511	0.561	100	5
Physical Skeleton	5	32.5875	86.419	0.874	0.660	100	9
Constrained Physical Skeleton	3	24.1933	120.798	1.472	0.999	100	10

Algorithm 1: Tuning parameters with a Monte Carlo search.

```
function TUNE-PARAMETERS($K^g_f, K^g_{vc}$, points, maxOrder, mocap-file)
    $K_f \leftarrow K^g_f$
    $K_{vc} \leftarrow K^g_{vc}$
    $\nabla \leftarrow \{\}$
    for $i = 3 \rightarrow maxOrder, 1 \equiv i \bmod 2$ do
        $\nabla \leftarrow \{\text{central-difference}(i, \mathbf{r})\}$
    end for
    $K_{f,b} \leftarrow K_f, K_{vc,b} \leftarrow K_{vc}, \nabla_b \leftarrow \nabla(3)$
    $\bar{e}^2_\rho \leftarrow \infty$
    for $i = 1 \rightarrow$ points do
        for $\nabla(n) \in \nabla$ do
            $\bar{es}^2_\rho$ = run simulation of mocap-file with $(K_f, K_{vc})$
            if $\bar{es}^2_\rho < \bar{e}^2_\rho$ then
                $\bar{e}^2_\rho = \bar{es}^2_\rho$
                $K_{f,b} \leftarrow K_f, K_{vc,b} \leftarrow K_{vc}, \nabla_b \leftarrow \nabla(n)$
            end if
            $K_f \leftarrow K_f$+random$[-1,1]$
            $K_{vc} \leftarrow K_vc$+random$[-1,1]$
        end for
    end for
    return $\bar{e}^2_\rho, K_{f,b}, K_{vc,b}, \nabla_b$
end function
```

good points, we achieved convergent results with a small number of runs. The tuning results for 141_15.bvh are listed in Table 1. These results show that we can minimize the error between our simulation and a motion capture simply and effectively. To speed calculations of averages, and avoid errors such as massive cancellation, we used an algorithm due to Knuth and Welford (Knuth, 1998). This algorithm can calculate an average and standard deviation in one pass of the data. We use it during our simulation. In Algorithm 2, we present our use of the method for the positional errors of each joint. The same algorithm is used to aggregate the joint positional errors into our overall average error.

As with any controller, the parameters in Table 1 may work great for a specific instance, but breakdown for another. These results are repeatable, simulation after simulation. Longer captures accumulated greater error, as shown in Figure 2.

Algorithm 2: Online average calculation.

```
$\forall j \in J, \bar{e}^2_\rho(j) \leftarrow 0, n(j) \leftarrow 0, M_2(j) \leftarrow 0$
for $j \in J$ at each step $s, f$ of the simulation do
    $n(j) \leftarrow n(j) + 1$
    $\Delta \leftarrow e^2_\rho(j,f) - \bar{e}^2_\rho(j)$
    $\bar{e}^2_\rho(j) \leftarrow \bar{e}^2_\rho j + \frac{\Delta}{n(j)}$
    $M_2(j) \leftarrow \Delta(\bar{e}^2_\rho(j,f) - \bar{e}^2_\rho(j)$
end for
$\sigma^2_\rho(j) = M_2(j)/(n(j) - 1)$
```

Animations using the Tuned Parameters. In Figure 3, we present animation stills from within a single simulation. Each type of rigid bodies was simulated on a single file, 141_02.bvh. The camera remained at the same place as the independent rigid bodies, the physical skeleton, and the constrained physical skeleton were simulated in turn.

Figure 4 shows the physical skeleton after it has turned around and is walking back towards its original position in the simulation of motion capture file 141_30.bvh. Figure 5 shows the constrained physical skeleton at the same point in 141_30.bvh. The disconnections in Figure 4 are gone from 5 because of the Bullet constraints placed on the joints. Coupled with the tuned parameters, constraints offer a powerful animation enhancement. To demonstrate our two generative animation techniques, we used our five motion capture files. For instance, 141_14.bvh captures punching and kicking, which is a good candidate to show how scaling, e.g., the left foot, changes the kicks, and 141_30.bvh captures a random walk, which is a good candidate for showing how under-controlling can produce, e.g., a zombie-like animation. All of our generative animations used a constrained physical skeleton.

2.1 Asymmetric Scaling

By scaling $K_v(j,f)$, we found that we could make the kicks of 141_14.bvh harder or softer easily. For instance, by scaling the right foot by

$$K_v(\text{RightFoot}, 200 \leq f \leq 600) = 1.2,$$

we simulated a harder snap between the right knee and foot, as compared in Figure 6.

Table 2: Comparison of errors using 141_15.bvh tuned parameters with other captures.

File	Rigid Body	$\bar{e}_\rho^2$	σ_ρ^2
141_02.bvh	Independent Rigid Bodies	3.431	0.194
141_02.bvh	Physical Skeleton	3.262	0.174
141_02.bvh	Constrained Physical Skeleton	0.669	0.022
141_08.bvh	Independent Rigid Bodies	2.992	0.023
141_08.bvh	Physical Skeleton	2.389	0.020
141_08.bvh	Constrained Physical Skeleton	0.237	0.007
141_14.bvh	Independent Rigid Bodies	0.121	0.0005
141_14.bvh	Physical Skeleton	0.234	0.012
141_14.bvh	Constrained Physical Skeleton	0.395	0.042
141_30.bvh	Independent Rigid Bodies	4.612	0.029
141_30.bvh	Physical Skeleton	3.220	0.071
141_30.bvh	Constrained Physical Skeleton	0.893	0.164

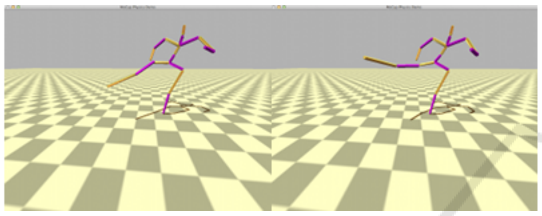
Figure 6: Stills from simulations of 141_14.bvh comparing the original, un-scaled simulation on the left to a simulation using asymmetrical scaling on the RightFoot on the right.

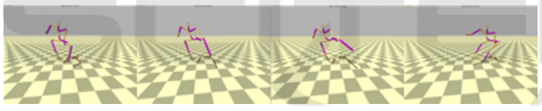
Figure 7: Stills from simulations of 141_15.bvh using the asymmetrical scaling from Table 4.

In the original simulation, at the height of the kick, the right leg stays below ground-parallel, the left leg is bent, the left hip is more ground-parallel, the left elbow is bent into the body, and the spine is bent only slightly towards the left side of the body. In the scaled simulation, at the height of the kick, the right leg is ground-parallel or more, with the knee actually bent past 90 degrees, the left leg is less bent, the left hip is less ground-parallel, the left elbow is bent more away from the body, and the spine is bent more towards the left side of the body. Just by applying an asymmetric scale to the RightFoot, we substantially changed the resultant forces and torques on the entire leg, the Spine, the LeftHand, even the Head! If a kickboxer kicks 1.2 times harder with his feet, this simulation shows what happens to his entire body.

We also applied asymmetric scaling on 141_15.bvh using multiple different scales during the entire simulation. The different scales are listed in Table 4. The resulting animation shows what would happen if the left side of the body became significantly stronger than the right side, and is quite comical. Figure 7 shows stills from a simulation of 141_15.bvh using the scales in Table 4 where the left arm actually goes under the left leg.

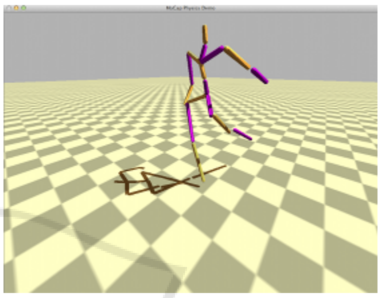
Figure 8: A still from the simulation of 141_30.bvh using an under-controlled constrained physical skeleton where the left hand and left foot are not controlled. The hand and foot lift up because of their constraints with their parent bodies and joints.

While comical, this generation also demonstrates how someone recovering from a serious injury will be impacted. For instance, a person who does injure their right leg severely and must keep off of it will experience decreased muscle mass in the right leg, and potentially increased muscle mass in their left leg to compensate. This imbalance will lead to very unexpected results as the person begins using their right leg again, as demonstrated here.

2.2 Under-controlling: Zombies

Furthermore, a gimp leg does not rotate at the knee. So, instead of a point to point constraint, we used a generic 6 DoF constraint locked in all axes at the knee and hand. We also changed the mass of the three bodies in the chain between the foot and the hip and the hand and the shoulder. We found that the foot mass should be lower than the leg mass, and the hand mass should be lower than the arm mass, to produce

Table 3: Comparison of the forces on certain joints of an un-scaled and scaled 141_14.bvh simulation. The scaling is solely applied to the RightFoot.

Joint j	$\bar{F}^2(r_b^1(j), r_b^2(j))$	$\bar{\tau}^2(r_b^1(j), r_b^2(j))$
RightFoot	15.196	23.643
RightLeg	4.458	1.640
RightUpLeg	1.618	2.548
Spine	0.148	0.554
LeftFoot	0.0987	0.0949
LeftLeg	0.084	0.143
LeftUpLeg	0.307	1.043
LeftHand	0.009	0.145
LeftForeArm	0.002	0.002
LeftArm	0.008	0.002
Head	0.010	0.008

Table 4: The asymmetric scales used to simulate 141_15.bvh. Values for joints are given.

Joint j	K_v
LeftFoot	1.1
RightFoot	0.9
LeftHand	1.5
RightHand	0.5
All other joints	1.0

the desired animation. These bodies are less massive on the human body, and, zombies being anatomically human, this result is most expected. Masses for LeftFoot, LeftLeg, LHipJoint, LeftHand, LeftFOreArm, and LeftArm are: 5, 10, 20. .25, 2.0, and 5 respectively. All other joints are 1 units.

3 CONCLUSIONS AND FUTURE WORK

Using Python, C++, and the Bullet Physics library, a motion capture to linear velocity transformer was implemented. This transformer relied on two techniques with three parameters: numerical differentiation using central differences with one parameter, the order, and proportional controllers with two parameters, the actual proportionality constant for the controller and a symmetric scaling constant. We also showed how, using this transformer, we could achieve two generative animation techniques: asymmetric scaling and under-controlling. The results showed that replicating the original motion capture in a physics engine is feasible, though the proportional controllers tend to accumulate errors over time. Tuning the parameters to a single capture was fairly simple using a Monte Carlo method, and these parameters worked well across other captures. Physical skeletons tended to produce defects at the parent joint of a rigid body over time. Constrained physical skeletons compensated for these defects and created good-looking animations. We found that generating new animations in the physics simulations achieved intriguing results. Asymmetrical scaling can be achieved simply; changing only one joint slightly in a few frames caused noticeable and interesting changes in the simulation. We also showed how under-controlling could achieve a zombie-like effect by under-controlling certain rigid bodies on the left-side of a constrained physical skeleton. By recreating motion captures in a physics engine, we hoped to overcome previous dynamics generation limitations. Possible uses of this work include sports medicine, athletic training, and model animation, especially for video games.

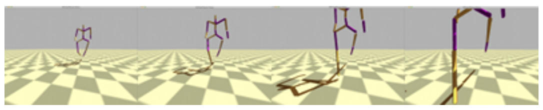
Figure 9: Stills from a simulation of 141_02.bvh using a zombie physical skeleton. We can see that the left leg's knee does not bend at all.

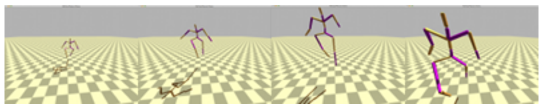
Figure 10: Stills from a simulation of 141_08.bvh using a zombie physical skeleton. The motion capture actor steps up onto a platform and then down. The platform is not shown here. We can see that the skeleton looks awkward stepping down from its left leg onto its right, since the left leg is locked and does not bend at the knee.

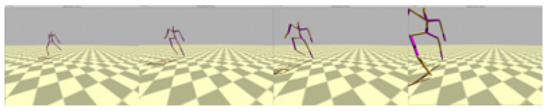
Figure 11: Stills from a simulation of 141_30.bvh using a zombie physical skeleton. We can see that the left side of the body is limp corresponding to a shuffling zombie.

REFERENCES

Ashraf, G. and Wong, K. C. (2000). Dynamic time warp based framespace interpolation for motion editing. In *Proceedings of the Graphics Interface 2000 Conference, May 15-17*, pages 45–52.

Baraff, D. (1989). Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *Computer Graphics*. Vol. 23, Number 3, July 1989, pp. 223-232.

Baraff, D. (1991). Coping with friction for non-penetrating rigid body simulation. In *Computer Grpahics*. Vol. 25, Number 4, August 1991, pp. 31-40.

Beazley, D. M. (2011). Ply python lex-yacc. Web Accessed.

Boulic, R., Magnenat Thalmann, N., and Thalmann, D. (1990). A global human walking model with real-time kinematic personification. In *The visual computer*. Vol. 6, No. 6, 1990, pp. 344-358.

Calvert, T., Chapman, J., and Patla, A. (2002). Aspects of the kinematic simulation of human movement. In *IEEE Computer Graphics and Applications*. Vol 2, No 9, pp. 41-50, 1982.

Calvert, T., Chapman, J., and Patla, A. (CMU 2003). Carnegie mellon university motion capture database. In *Web Accessed*. 9 July 2013.

Club, T. M. C. (July 2013). Web accessed.

Coumans, E. (2012). Bullet 2-80 physics sdk manual. In *I*. Web Accessed 9 July 2013.

Evans, C. (2011). Yaml. In *Web Accessed*. Web Accessed.

Glardon, P., Boulic, R., and Thalmann, D. (2004). Pca-based walking engines using motion capture datat. In *IComputer Graphics International*. Computer Graphics International.

Hodgins, J. (1996). Three-dimensional human runningt. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*. Vol 4,.

Knuth, D. E. (1998). The art of computer programming. In *IEEE Computer Graphics and Applications*. p 232 3rd Edition, Addison Wesley, Boston.

Macchietto, A., Zordan, V., and Shelton, C. (2009). Momentum control for balance. In *IACM Transactions on Graphics*. Vol. 28, No. 3, ACM, 2009.

Multon, F. (1999). Computer animation of human walking: a survey. In *The journal of visualization and computer animation*. Vol. 10, No. 1, 1999, pp. 39-54.

OpenGL (2013). The python opengl binding. Web Accessed.

Rose, C., Cohen, M., and Bodenheime, B. (1998). Verbs and adverbs: Multidimensional motion interpolation. In *IEEE Computer Graphics and Applications Vol. 18 No. 5*, pages 32–40.

Ryan, R. (1990). Multibody systems handbook. In *IADAMS – Multibody System Analysis Software*. Springer Berlin Heidelberg, 1990, pp. 361-402.

Schreiner, L. K. J. and Gleicher, M. (2002). Footskate cleanup for motion capture editing. In *ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM.

SciPy (July 2013). scipy.misc.derivative. In *SciPy v0 12 Reference Guide (DRAFT)*. Web Accessed.