Handling of plasticity in physics engine

August 13, 2016

1 Introduction

The application of modern computer game techniques enables the description of complex dynamic systems such as military vehicles with a high level of detail while still solving the equations in real-time. Film production and war games, in particular, is a key area that have benefited from simulation technology. In practice, games are often accomplished using an open-source platform such like ODE - Smith (2001-2007), Bullet Physics - Coumans (2003-2016) and Box2D - Catto (2007-2015).

Computational methods used in physics engines are divided to modules that handle collision detection and contact description and modules that handle solution of equations in real-time. Equations need to be solved can further be subdivided to be associated to motion, constraints and collisions. Velocity-based formulation is typically used in constraint based rigid body simulation. Friction is typically taken into account and mechanical joints are handled by constraint equations. Detailed description of various components can be found in e.g. Erleben (2005).

Plasticity is not typically taken into account in gaming solutions. Breaking of various objects typically takes place based on collision or impulse. Nevertheless, breaking of steel or reinforced concrete structures using this approach is not appropriate making a simulation to look unrealistic. Theory for handling of plasticity has been presented already in Terzopoulos and Fleischer (1998). Müller et al. (2004) and Müller et al. (2005) present a method for modeling and animating of elastic and plastic objects in real-time using point based animation. This approach is not been widely used in simulation applications. On major issue is collision handling of deformable objects.

This study will introduce an approach to account plastic deformation in game applications. In the introduced method, the plastic deformation takes place if force or moment exceeds given limit, deformation absorbs energy and joint breaks if plastic capacity is exceeced. The approach is based on using joint motors to model plasticity. Erleben (2005, p. 90) suggests similar method for modelling friction in joints. Adjacent objects are connected by motors. Motor power production limits are estimated based on plastic section modulus. Joint breaking is accounted by summing plastic deformation and comparing it to predefined material based limit. Elastic part of deformation is modelled by employing spring description which is based on modification of existing constraint in Bullet Physics.

Approach presented in this work can be used in gaming industry to provide more realistic simulations without significant extra work. For gaming purposes presented method works best in scenarios where connected parts are relatively heavy. This allows normal integration timestep to be used without stability issues. This kind of metodology also opens large area of combining old structural analysis methods to modern simulation frameworks.

2 Adding plasticity to physics engine

In this section, key concepts related to the introduced model are explained. Main differences between traditional structural analysis and physics engines are reviewed and discussed.

Velocity-based formulation is not often used in structural or mechanical engineering while, however, it is popular within physics based game developers and film production teams. Erleben (2005, p. 45) provides reasoning and theoretical details on why velocity-based formulation is popular in constraint-based rigid body simulation. Main reason is that collision handling can be done without additional procedures. Acceleration-based formulations need to stop at collision and switch to an impulse based method.

Background for velocity based formulation shown here is based on Erleben (2005, p. 45-50). In following section, these formulations will be clarified by simple example using Bullet Physics implementation. Assuming that force $\vec{f}_{true}(t)$, is known the impulse $\vec{J}$ in the time interval Δt can we written as

$$\vec{J} = \int_0^{\Delta t} \vec{f}_{true}(t)dt \tag{2.1}$$

Using Newton's second law of motion $F = ma$ one can solve for the velocity, $\vec{v}^{\Delta t}$

$$\int_0^{\Delta t} m\frac{d\vec{v}}{dt}dt = \int_0^{\Delta t} \vec{f}_{true}(t) \tag{2.2}$$

$$m(\vec{v}^{\Delta t} - \vec{v}^0) = \vec{J} \tag{2.3}$$

where superscripts denote time, i.e. $v^{\Delta t} = v(\Delta t)$. Next position can be found by integrating the velocity. Derivation of equations of motion is not shown here in detail but results for simple contact can be summarized as equation 2.4 for locations and 2.5 for velocities. Symbols are summarized in table 2.1 and figure 2.1.

$$\vec{s}^{t+\Delta t} = s^t + \Delta t S\vec{u}^{t+\Delta t} \tag{2.4}$$

$$\vec{u}^{t+\Delta t} = u^t + \Delta t M^{-1}(CN\vec{f}^{t+\Delta t} + \vec{f}_{ext}) \tag{2.5}$$

Friction in contacts and joint constraints can be handled in unified way by refactoring equation 2.5 to get 2.6, Erleben (2005, p. 66-67). Jacobian terms are derived by taking time derivates of kinematic constraints.

$$\vec{u}^{t+\Delta t} = u^t + \Delta t M^{-1}(J^T_{contact}\vec{\lambda}_{contact} + J^T_{joint}\vec{\lambda}_{joint} + \vec{f}_{ext}) \tag{2.6}$$

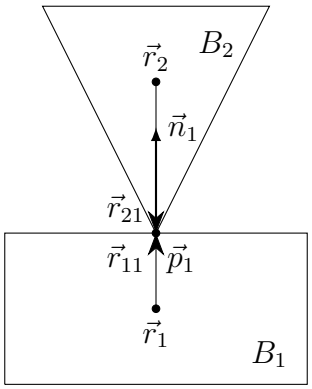

Figure 2.1: Illustration of nomenclature for equations of motion.

In structural analysis, a formulation and associated numerical solution procedure are selected based on needed features. For most complex scenarios finite element method is used. In most cases,

Symbol	Description
$\vec{r}_i$	position of center of mass for body i
$\vec{q}_i$	orientation as a quaternion for body i
$\vec{p}_i$	contact or joint point i
$\vec{r}_{ki}$	$\vec{p}_k - \vec{r}_i$
$\vec{s}$	$[\vec{r}_1, \vec{q}_1, ..., \vec{r}_n, \vec{q}_n]^T$
S	generalized transformation matrix
$\vec{v}_i$	linear velocity of center of mass for body i
$\vec{\omega}_i$	angular velocity of center of mass for body i
$\vec{u}$	$[\vec{v}_1, \vec{\omega}_1, ..., \vec{v}_n, \vec{\omega}_n]^T$
M	generalized mass matrix
C	contact condition matrix
N	contact normal matrix
$J_{contact}$	Jacobian matrix for contacts
$\lambda_{contact}$	vector of lagrange multipliers for contacts
J_{joint}	Jacobian matrix for joints
λ_{joint}	vector of lagrange multipliers for joints

Table 2.1: Nomenclature for equations of motion

static solution with assumption of linear strain-displacement relation small displacement solution using displacement based boundary conditions is used. Bathe (1975) provides description for handling of various nonlinearities. In large displacement analysis, formulation may be based on updated formulation (Eulerian) or Lagrangian formulation where initial configuration is used. Further enhancements are material nonlinearity and dynamic analysis. Physics engine provides dynamic analysis with large large reference translations and rotations.

Material plasticity has typically taken into account in games by using suitable coefficient of restitution. This provides reasonable means to simulate loss of energy in collisions. Simulation of breaking of objects made of ductile material can be made more realistic by splitting rigid bodies to multiple bodies which are connected by energy absorbing joints. Typical engineering stress-strain curve of ductile steel is shown in 2.2. E.g. Dowling (2007) provides detailed descriptions of engineering and true stress-strain curves. Stress-strain curve is not drawn to scale as elastic strain could not be seen as it is typically 0.001 to 0.005.

In this work elastic-fully plastic material model is used in most scenarios. It allows realistic simulations for most scenarios. Having elastic part allows elastic displacements for slender structures. Elastic part is ignored in method suggested in this work if deformation is related to higher frequency than integration stability would allow. Strain hardening part of stress-strain curve could probably be taken into account but it has not been tried in this work. It should be noted that geometry of objects is not updated during analysis and thus engineering stress-strain properties should be used even with strain hardening.

Strain hardening is taken into account in this work mainly by assuming that plasticity in bending expands, Dowling (2007, p. 672). Material that starts to yield first is hardened and yielding moves slightly. This can be seen e.g. by bending paperclip. It does not break at low angles but can take few full bends.

Difference between elastic and plastic section modulus is shown in 2.3. If stress is below yield limit, stress and strain are linear within material. If cross section is fully plastic, stress is assumed to be at yield level over whole cross section and so plastic section modulus is higher than elastic section modulus.

Basic idea in this work can be tested with any framework having motors and hinge constraints. This can be done by setting target velocity of motor to zero and limiting maximum motor impulse to

Figure 2.2: Engineering stress-strain curve of ductile steel (not to scale).

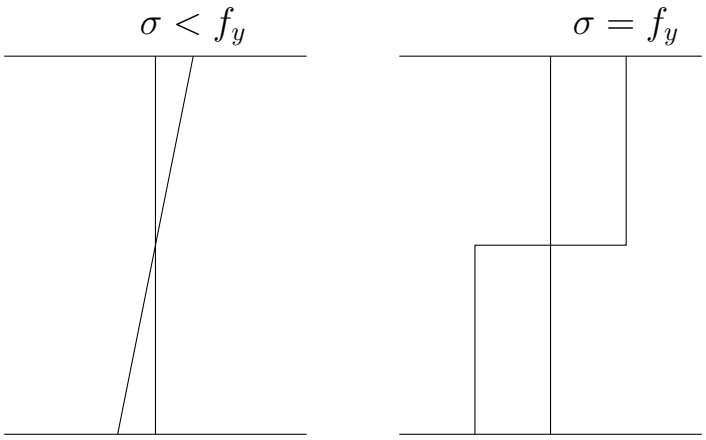

Figure 2.3: Stress distribution under elastic and plastic loads.

plastic moment multiplied by timestep.

Further enhancements were created and tested by forking Bullet Physics source code and adding new constraints Nikula (2014-2016). Constraint processing in Bullet Physics is based on ODE, Smith (2001-2007). Mathematical background and detailed examples are available by Smith (2002). Joints are also discussed in detail in Erleben (2005, p. 60-90). In following section, these equations will be clarified by simple example. Equations 2.7, 2.8 and 2.9 are created for each constraint.

$$J_1 v_1 + \Omega_1 \omega_1 + J_2 v_2 + \Omega_2 \omega_2 = c + C\lambda \tag{2.7}$$

$$\lambda \geq l \tag{2.8}$$

$$\lambda \leq h \tag{2.9}$$

Main parameters and corresponding fields in Bullet Physics are described in table 2.2.

Parameter	**Description**	**btConstraintInfo2 pointer**
J_1, Ω_1 J_2, Ω_2	jacobian	m_J1linearAxis, m_J1angularAxis m_J2linearAxis, m_J2angularAxis
v	linear velocity	
ω	angular velocity	
c	right side vector	m_constraintError
C	constraint force mixing	cfm
λ	constraint force	
l	low limit for constraint force	m_lowerLimit
h	high limit for constraint force	m_upperLimit

Table 2.2: Constraint parameters

3 Processing plasticity

In this section, changes to constraint formulation are clarified by doing few simulation steps of non constrained, rigid and elastic-plastic examples.

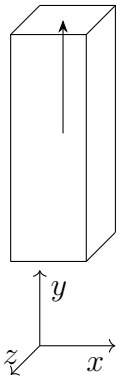

Figure 3.1: Single body model for plasticy processing demonstration.

System has only one dynamic rigid body which is three meters high block of 0.04 % steel enforced concrete. Cross section is one square meter. Constraint is set so that connecting frame is at top of block. In this case frame could be anywhere but if multiple bodies are involved, connecting frame must be defined so that it reflects wanted scenario. Concrete density is $2000\frac{kg}{m^3}$ and steel density is $7800\frac{kg}{m^3}$. Steel is assumed to handle load in elastic-plastic case. Yield stress of steel is 200 MPa. Gravity is $10\frac{m}{s^2}$. Simulation step is $\frac{1}{60}s$ and 10 iterations are done for single step. Body is 1.5 meters above rigid ground. In unconstrained case body will hit hit ground at t=0.548 s ($\sqrt{2 \cdot 1.5/10}$).

Single simulation step is done using following substeps. In this work, changes are made only to constraing solving and update actions.

1. Apply gravity to each non static body. This step just makes programming easier as otherwise each program should add forces due to gravity in each step. In this case, $\vec{f_{ext}}$ gets added by $\{0, -60000, 0\}$.

2. Predict unconstrained motion for each non static body. Prediction is done based on current linear and angular velocities of the body.

3. Predict contacts. In this phase, continuous collision detection is done based on simplified objects. Each rigid body is represented by sphere and contact prediction is done if body moves more than given threshold value during simulation step. Continuous collision detection is configured for each body and as default it is not active. Typical scenario that requires continuous collision detection is fast moving bodies that would otherwise go through walls.

4. Perform discrete collision detection. All overlapping body pairs are processed and manifoldPoints are created for each detected contact at end of simulation step. In unconstrained case, contact is detected after t=0.55 s if continuous collision detection is not used and manifolds distance gets negative value (-0.058 m).

5. Calculate simulation islands(groups). Bodies that are near each other based on contact prediction or discrete collision detection or connected with constraints are grouped in same group.

6. Solve constraints. Both contact and other constraints are processed in this step.

7. Integrate transforms using velocities modified in previous step.

8. Update actions (callbacks) are called. In elastic-plastic case, plasticity is summed and equilibrium point of elastic part is updated if maximum force or moment is exceeded.

9. Activation state of bodies is updated. To avoid extra calculation bodies are as default put to sleeping state if linear and angular velocities of body are less than given threshold values (default 0.8 and 1.0) more than set limit (2.0).

Equation 2.7 is simplified to 3.1 in constrained cases.

- No rotation takes place. ω_1 and ω_2 are zeros.
- Constraint force mixing can be ignored.
- Only vertical velocity is handled.
- Other involved body is rigid and it does not move.

$$mv_y = c \tag{3.1}$$

Equations 2.8 and 2.9 are not active for fixed case. For elastic-plastic case maximum impulse is set to product of yield stress, area of steel enforcement and time step (1330).

Method btSequentialImpulseConstraintSolver::solveGroupCacheFriendlySetup in Bullet Physics was used to pick up values for internal variables.

velError is calculated using velocities and external impulses of connected bodies.

In constraint cases, it is dominant contributor for constraint's impulse.

In contact case, main contributor is bodies relative speed at point of contact. Bodies are not allowed to penetrate each other.

posError is calculated by constraint. It is significant factor in designing stable constraints.

In fixed case, value is about 12 times actual position error. Factor 12 is based on time step (60) and default value of error reduction parameter (erp) which has value of 0.2 in this context.

In elastic plastic case, value is set to zero if impulse would be larger than maximum impulse or spring simulation cannot be done in stable way.

In contact case, value is zero if there is no penetration. For penetration cases it is $\frac{-penetration\,erp}{timeStep}$

rhs (c) is calculated by velError jInv + posError jInv

jInv is calculated using masses and inertias of connected bodies and constraint geometry.

In constraint cases, it is mass of body (6000).

In contact case, it varies below mass of body.

Impulse is impulse applied to body during timestep.

In constraint cases, it is obtained from btJointFeedback structure.

In contact case, it is obtained by summing applied impulses from active manifolds.

erp Error reduction parameter (0...1) is used to handle numerical issues e.q. object drifting. Setting erp to 1 would in theory eliminate error but in practice value of 0.8 is used in most cases.

Actual values for unconstrained case without continuous collision detection are shown in 3.1. Penetration is detected at time 0.567 s when *velError* is 5.67 (5.5+0.167) and *posError* is 2.8 (0.058*0.8/0.0167). Impulse is 34000 and contact force is thus about 2 MN (34000/0.0167). After few steps location and position stabilize although internally calculation is needed for each time step until body is deactivated.

Time	**Location**	*velError*	*penetration*	*posError*	*rhs*	**Velocity**	**Impulse**
0.017	0					-0.17	0
0.550	-1.558					-5.5	0
0.567	-1.511	5.67	-0.058	2.8	21270	0.01	34000
0.583	-1.502	0.14	-0.011	0.54	2570	0.55	420
0.600	-1.496	-0.38	-0.002	0.1	-1000	0.38	0
0.617	-1.492	-0.44	0.004	0	-1600	0.22	0
0.717	-1.497	0.004-0.08	-0.0003-0.001	0-0.01	10-400	-0.01	400
0.817	-1.499					-0.08	700
0.917	-1.500					-0.001	1000

Table 3.1: Simulation values for unconstrained case. For internal contact values typical values are shown as number of contacts and detailed values differ.

Time	**Location**	*velError*	*penetration*	*posError*	*rhs*	**Velocity**	**Impulse**
0.017	0					-0.17	0
0.550	-1.500	3.5	0.033	0	21000	-2	21000
0.567	-1.500	2.17	0	0	8100	0.01	13000
0.583	-1.500	0.15	0	0	600	0	937
0.600	-1.500	0.17	0	0	600	0	1040

Table 3.2: Simulation values for unconstrained case with continuous collision detection.

Actual values for unconstrained case with continuous collision detection (ccd) using 1.5 as radius of ccd sphere and 0.001 as ccd motion threshold are shown in 3.2. Collision is detected at time 0.550 s when *velError* is 3.5 (5.34+0.167)-0.033/0.0167 and *posError* is 0 as collision is detected before penetration. It should be noted that in general ccd sphere should not extend actual body as premature contacts are created if collision takes place in those regions.

Values for fixed constraint are shown in table 3.3. Constraint is activated in second step and positional error is corrected about 20 % in each step as requested by using erp value 0.2.

Time	**Location**	*velError*	*posError*	*rhs*	**Velocity**	**Impulse**
0.017	-0.0028				-0.17	0
0.033	-0.0022	0.33	-0.033	-2200	0.033	2200
0.050	-0.0018	-0.13	-0.027	-960	0.027	960
0.067	-0.0014	-0.14	-0.021	-970	0.021	970
0.35...	0	-0.17	≈ 0	-1000	0.0	1000

Table 3.3: Constraint parameter values for fixed constraint

Values for elastic-plastic case are shown in tables 3.4 and 3.5. There are currently two alternative six-dof-spring constraint implementations in Bullet Physics and in this work elastic-plastic versions of both of them were developed. Constraint activation of older spring and fixed constraints is implemented so that they are activated after constraint is violated. Thus body drops freely during first simulation step and gains enough kinetic energy so that higher impulses are needed in few following steps.

Time	*velError*	*posError*	*rhs*	*velocity*	**Impulse**	**Plastic strain**
0.017				-0.17	0	0
0.033	-0.33	0	-2000	-0.11	1330	0.001
0.050	-0.28	0	-1670	-0.056	1330	0.003
0.067	-0.22	0	-1340	-0.001	1330	0.004
0.083...	-0.17	0	-1000	0.0	1000	0.004

Table 3.4: Constraint parameter values for elastic-plastic constraint

Time	*velError*	*posError*	*rhs*	**Velocity**	**Impulse**	**Plastic strain**
0.017...	-0.17	0	-1000	0	1000	0

Table 3.5: Constraint parameter values for elastic-plastic constraint 2

References

Bathe, K.J. (1975). Finite Element Formulations for Large Deformation Dynamic Analysis. *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING*, pp. 353–386. url: http://web.mit.edu/kjb/www/Publications_Prior_to_1998/Finite_Element_Formula

Catto, E. (2007-2015). *Box2D A 2D Physics Engine for Games*. url: box2d.org.

Coumans, E. (2003-2016). *Bullet Physics Library*. url: bulletphysics.org.

Dowling, N.E. (2007). *Mechanical Behavior of Materials - Engineering Methods for Deformation, Fracture, and Fatigue, third edition*. Prentice-Hall, New Jersey. ISBN 0-13-186312-6.

Erleben, K. (2005). *Stable, Robust, and Versatile Multibody Dynamics Animation*. Ph.D. thesis. University of Copenhagen. image.diku.dk/kenny/download/erleben.05.thesis.pdf.

Müller, M., Heidelberger, B., Teschner, M., and Gross, M. (2005). Meshless deformations based on shape matching. In: *ACM Transactions on Graphics (TOG)*, vol. 24, 3, pp. 471–478. url: www.beosil.com/download/MeshlessDeformations_SIG05.pdf.

Müller, M., et al. (2004). Point based animation of elastic, plastic and melting objects. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 141–151. url: lgg.epfl.ch/publications/2004/mueller_2004_PBA.pdf.

Nikula, S. (2014-2016). *Plasticity extension to Bullet Physics Library*. url: https://github.com/simo-11/bullet3.

Smith, R. (2001-2007). *Open Dynamics Engine*. url: ode.org.

Smith, R. (2002). *How to make new joints in ODE*. url: ode.org/joints.pdf.

Terzopoulos, D. and Fleischer, K. (1998). Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture. *Computer Graphics*, pp. 269–278. doi:10.1145/54852.378522, url: web.cs.ucla.edu/~dt/papers/siggraph88/siggraph88.pdf.