1d Constraint Formulation

PhysX Team

April 29, 2024

Preliminaries

A constraint concerns the relationship of the state of two rigid bodies (body 0 and body 1) with positions $x_0(t)$ and $x_1(t)$; and quaternions $q_0(t)$ and $q_1(t)$.

The state of the body pair $s(t)$ is as follows:

$$s(t) = \{x_0(t), q_0(t), x_1(t), q_1(t)\} \tag{1}$$

If the two rigid bodies have linear velocities $v_0(t)$ and $v_1(t)$; and angular velocities $\omega_0(t)$ and $\omega_1(t)$ then we may define:

$$v(t) = \{v_0(t), \omega_0(t), v_0(t), \omega_1(t)\} \tag{2}$$

The two rigid bodies are subject to external forces f_{ext0} and f_{ext1} applied at the centre of mass; and external torques τ_{ext0} and τ_{ext1}:

$$F_{ext}(t) = \{f_{ext0}, \tau_{ext0}, f_{ext1}, \tau_{ext1}\} \tag{3}$$

The two rigid bodies are also subject to constraint forces f_{c0} and f_{c1} applied at the centre of mass; and constraint torques τ_{c0} and τ_{c1}. The purpose of this document is to detail the constraint force calculation that will result in a constraint force $F_c(t)$ that will satisfy the constraint.

$$F_c(t) = \{f_{c0}, \tau_{c0}, f_{c1}, \tau_{c1}\} \tag{4}$$

To compute the effect of $F_{ext}(t)$ and $F_c(t)$ it is necessary to introduce a mass matrix M:

$$M = \begin{bmatrix} m_0 & 0 & 0 & 0 \\ 0 & I_0 & 0 & 0 \\ 0 & 0 & m_1 & 0 \\ 0 & 0 & 0 & I_1 \end{bmatrix} \tag{5}$$

where m_0 and m_1 are diagonal 3x3 matrices with the respective masses of bodies 1 and 2 along each diagonal; and I_0 and I_1 are 3x3 matrices describing the inertia with respect to the centre of mass of bodies 1 and 2 in the world frame.

To avoid confusion with I_0 and I_1, the identity matrix shall be henceforth denoted by E.

The bodies of the constraint have associated joint frames L_1 and L_1 that are relative to the centre of mass:

$$L_0 = \{Lp_0, Lq_0\} \tag{6}$$
$$L_1 = \{Lp_1, Lq_1\} \tag{7}$$

where Lp_0 and Lp_1 represent the translations of the two joint frames; and Lq_0 and Lq_1 represent the rotations of the two joint frames.

Constrained Dynamics

The purpose of a constraint is to enforce a specific rule placed on the state of the body pair $s(t)$. Such a rule may be that the two bodies are separated by a particular distance or that their separation projected along a direction must be greater than zero. Any combination is possible. To avoid losing generality, the constraint may be expressed as follows:

$$C(s) = 0 \tag{8}$$

It is also possible to express a constraint as an inequality constraint ($C(s) \geq 0$). The remainder of this document, however, shall focus on equality constraints.

$C(s)$ will be maintained for as long as $\dot{C}$ is also zero. Appendix 1 demonstrates that $\dot{C}$ may be recast in the following form:

$$\dot{C} = Jv(t) = 0 \tag{9}$$

In practice, it is not possible to ensure that the constraint is resolved exactly. Indeed, it is typical for errors to accumulate due to time discretization and rounding error. There is also the possibility a body pair will be initially configured in a way that does not satisfy the associated constraint. To avoid drift propagation and accumulation it is necessary to amend Equation (9) so that $v(t)$ accounts for the velocity required to counteract at least some of the error:

$$Jv(t) = -\alpha \frac{C(s)}{dt} \tag{10}$$

where dt is the timestep that will be used to forward integrate $s(t)$ and α is the "Baumgarte" multiplier in range (0, 1). Increasing α will more aggressively resolve the accumulated error but comes with the risk of adding energy to the system because it necessarily overshoots the ideal velocity that would occur with zero error.

Introducing a bias velocity b:

$$b = \alpha \frac{C(s)}{dt} \tag{11}$$

allows the following relationship:

$$Jv(t) + b = 0 \tag{12}$$

It is also useful to introduce a target velocity v_T for the constraint such that:

$$Jv(t) + b = v_T \tag{13}$$

This may be re-rexpressed as follows:

$$Jv(t) + b - v_T = 0 \tag{14}$$

The bodies of the constraint experience external forces $F_{ext}(t)$ and a constraint force $F_c(t)$:

$$v(t) = v(t - dt) + dtM^{-1}\{F_{ext}(t) + F_c(t)\} \tag{15}$$

The goal now is to compute $F_c(t)$ such that $v(t)$ satisfies (14). This is made possible by applying the rule [1] that a workless constraint force $F_c(t)$ has the following form:

$$F_c(t) = J^T\lambda \tag{16}$$

Substituting (16) and (15) into (14) reveals an equation for λ:

$$dt \cdot \lambda = -(JM^{-1}J^T)^{-1}\{b - v_T + Jv_{pre}(t)\} \tag{17}$$

where

$$v_{pre}(t) = J(v(t - dt) + dtM^{-1}F_{ext}(t)). \tag{18}$$

The intermediate state velocity $v_{pre}(t)$ is the state velocity after applying external forces for the timestep dt but before applying the constraint forces.

There is now a recipe for computing and appying the constraint forces:

$$v(t) = v_{pre}(t) + \lambda' M^{-1}J^T \tag{19}$$

where

$$\lambda' = -\frac{(b - v_T + v_n(t))}{r} \tag{20}$$

$$r = JM^{-1}J^T \tag{21}$$

$$v_n(t) = Jv_{pre}(t) \tag{22}$$

In practice, it is not sufficient to resolve a constraint just once because a typical use case is multiple constraints that compete with each other. The solution is to perform multiple passes over the list of all constraints and accumulate Δv each time a constraint is resolved. This observation leads to a generalisation of the recipe for computing and applying the constraint forces:

$$\lambda' = -\frac{(b - v_T + v_0)}{r} - \frac{J\Delta v}{r} \tag{23}$$

$$\Delta v \mathrel{+}= \lambda' M^{-1}J^T \tag{24}$$

$$v_n = v_0 + \Delta v \tag{25}$$

where Δv is the velocity change that accumulates over the solver iterations. The parameter v_0 represents the velocities immediately before the 1st solver iteration; that is, $v_0 = Jv_{pre}(t)$. Finally, the parameter v_n denotes the velocities after the nth solver iteration.

This formulation deserves a couple of remarks. The first thing to note is that $\frac{(b-v_T+v_0)}{r}$ is a constant and can be computed once before beginning the 1st solver iteration. A final note is that the first solver iteration naturally begins with $\Delta v = 0$.

Examples

It is instructive to compute the Jacobian for a small number of simple examples. To introduce basic concepts, a joint that locks relative linear motion along all 3 axes shall be discussed. This example is followed by a quick analysis of a joint that locks angular motion along all 3 axes. With these examples in mind, the concept of the 1d constraint is introduced.

It is worth noting that PhysX describes a joint as a collection of 1d constraints where each constraint may constrain linear or angular motion on a single axis. The sections on the spherical and angular joint are presented solely to introduce concepts that are applied in the treatments of 1d constraints.

Spherical Joint

The spherical joint enforces the rule that the position of each body is maintained at a constant distance from the joint anchor. If r_0 is the vector from body 0 to the joint anchor and r_1 is the vector from body 1 to the joint anchor then we may formulate the following constraint:

$$C(s) = (x_0(t) + r_0) - (x_1(t) + r_1) = 0 \tag{26}$$

The constraint derivative is readily computed as follows:

$$\dot{C} = (v_0(t) + \omega_0(t) \times r_0) - (v_1(t) + \omega_1(t) \times r_1) = 0 \tag{27}$$

The cross product terms may be re-rexpressed by introducing the equivalent matrices R_0 and R_1:

$$\dot{C} = (v_0(t) + R_0\omega_0(t)) - (v_1(t) + R_1\omega_1(t)) = 0 \tag{28}$$

This reveals a form that matches (9).

$$\dot{C} = \{E, R_0, -E, -R_1\}v(t) \tag{29}$$

The Jacobian of the spherical joint therefore has the form:

$$J = \{E, R_0, -E, -R_1\} \tag{30}$$

It is straightforward to compute $JM^{-1}J^T$:

$$JM^{-1}J^T = m_0^{-1} + m_1^{-1} + R_0I_0^{-1}R_0^T + R_1I_1^{-1}R_1^T \tag{31}$$

Angular Joint

Consider a joint that enforces the rule that two bodies do not rotate with respect to each other. This can be achieved if the bodies maintain the same angular velocity:

$$\dot{C} = \omega_0(t) - \omega_1(t) = 0 \tag{32}$$

This can be re-expressed in a form that invokes the following Jacobian:

$$J = \{0, E, 0, -E\} \tag{33}$$

It is straightforward to compute $JM^{-1}J^T$:

$$JM^{-1}J^T = I_0^{-1} + I_1^{-1} \tag{34}$$

1d Linear Constraint

It is important to generalise to the case of a 1d linear constraint because PhysX does not actually resolve groups of constraints as outlined in the earlier discussion of the spherical and angular joint. Instead, constraint types such as a spherical or prismatic joint are constructed from collections of 1d constraints.

While it might be tempting to formulate 1d linear constraints by picking out individual rows from the constraint equations presented for the spherical and angular joint, this does not work. The problem here is that each row of the spherical joint constraint equation represents an axis of the world frame. The same is true of the angular joint constraint equation. A different technique is required, ideally one that operates in the correct frame so that a linear constraint along x (or y or z) has persistent meaning when considering the relative motion of two bodies with arbitrary rotation. The key to understanding 1d linear constraints is an understanding of the frame used for the specification of the Jacobian.

For the spherical constraint the following has already been demonstrated:

$$\dot{C} = (v_0(t) + \omega_0(t) \times R_0) - (v_1(t) + \omega_1(t) \times R_1) = 0 \tag{35}$$

It is straightforward to express Equation (35) in the joint frame L_0 associated with body 0. This is achieved with the introduction of a rotation matrix Q that is the 3x3 matrix equivalent of $q_0(t)Lq_0$. Accounting for a non-zero bias velocity and non-zero target velocity specified in the world frame, we have the following:

$$Q^{-1}\{(v_0(t) + \omega_0(t) \times R_0) - (v_1(t) + \omega_1(t) \times R_1) + b - v_T\} = 0 \tag{36}$$

It is now possible to pick out individual rows from (36) to compute the Jacobian of a 1d linear constraint. The 1st row will correspond to a constraint along the x-axis the joint frame, the 2nd row will correspond to a constraint along the y-axis of the joint frame, and the 3rd row will correspond to a constraint along

the z-axis of the joint frame. Remembering that the inverse of a rotation matrix is equal to its transpose allows us to cast (36) as 3 separate rows:

$$Q_x \cdot (v_0(t) - v_1(t)) + Q_x \cdot (\omega_0(t) \times r_0 - \omega_1(t) \times r_1) + Q_x \cdot (b - v_T) = 0 \quad (37)$$
$$Q_y \cdot (v_0(t) - v_1(t)) + Q_y \cdot (\omega_0(t) \times r_0 - \omega_1(t) \times r_1) + Q_y \cdot (b - v_T) = 0 \quad (38)$$
$$Q_z \cdot (v_0(t) - v_1(t)) + Q_z \cdot (\omega_0(t) \times r_0 - \omega_1(t) \times r_1) + Q_z \cdot (b - v_T) = 0 \quad (39)$$

where Q_x, Q_y and Q_z are the columns of Q and each row represents a constraint along one axis of the joint frame L_0 associated with body 0. The expressions for each row may be recast in a form that matches the expected Jacobian form by applying the triple product rule [2]:

$$\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = \mathbf{b} \cdot (\mathbf{c} \times \mathbf{a}) = \mathbf{c} \cdot (\mathbf{a} \times \mathbf{b}) \quad (40)$$

The result is as follows:

$$Q_x \cdot v_0(t) + (r_0 \times Q_x) \cdot \omega_0(t) - Q_x \cdot \ v_1(t) - (r_0 \times Q_x) \cdot \omega_1(t) + b_x - v_{Tx} = 0 \quad (41)$$
$$Q_y \cdot v_0(t) + (r_0 \times Q_y) \cdot \omega_0(t) - Q_y \cdot \ v_1(t) - (r_0 \times Q_y) \cdot \omega_1(t) + b_y - v_{Ty} = 0 \quad (42)$$
$$Q_z \cdot v_0(t) + (r_0 \times Q_z) \cdot \omega_0(t) - Q_z \cdot \ v_1(t) - (r_0 \times Q_z) \cdot \omega_1(t) + b_z - v_{Tz} = 0 \quad (43)$$

where b_x, b_y, b_z, v_{Tx}, v_{Ty}, v_{Tz} are the individual components of the bias velocity vector and target velocity vector expressed in the joint frame L_0.

This exercise has revealed a general expression for a 1d constraint:

$$\mu_0 \cdot v_0(t) + \kappa_0 \cdot \omega_0(t) + \mu_1 \cdot v_1(t) + \kappa_1 \cdot \omega_1(t) + \alpha * \frac{\Delta}{dt} - vTar = 0 \quad (44)$$

with J taking the form:

$$J = \{\mu_0, \kappa_0, \mu_1, \kappa_1\} \quad (45)$$

and Δ and $vTar$ denoting the relevant element of $C(s)$ and v_T expressed in the joint frame L_0.

1d Angular Constraint

The 1d angular constraint follows a similar pattern to the 1d linear constraint. The key once again is to choose a frame that allows a single row from the Jacobian to be picked out in a meaningful way.

PhysX API

1d constraints are implemented in PhysX using the Px1dConstraint struct. This struct closely follows the theoretical outline of constrained dynamics discussed

in earlier sections. More specifically, the struct specifies the four Jacobian terms $\{\mu_0, \kappa_0, \mu_1, \kappa_1\}$, the geometric error Δ and the target velocity v_T.

The Jacobian specification involves the parameters linear0, angular0, linear1, angular1 such that:

$$\text{linear0} = \mu_0 \tag{46}$$
$$\text{angular0} = \kappa_0 \tag{47}$$
$$\text{linear1} = -\mu_1 \tag{48}$$
$$\text{angular1} = -\kappa_1 \tag{49}$$

The parameter geometricError performs the role of Δ, while the parameter velocityTarget performs the role of v_T.

The solver seeks a new state velocity such that:

$$\begin{aligned} &(\text{linear0} \cdot v_0(t) + \text{angular0} \cdot \omega_0(t)) \\ -&(\text{linear1} \cdot v_1(t) + \text{angular1} \cdot \omega_1(t)) \\ +&\frac{\text{geometricError}}{dt} \\ -&\text{velocityTarget} = 0 \end{aligned} \tag{50}$$

It is worth noting that the PhysX API does not allow users to set the Baumgarte multiplier α. The Baumgarte multiplier α is internally set. When PGS is employed the Bumgarte multiplier has value 1.0. When TGS is employed the Baumgarte multiplier has value 1/sqrt(nbPosIters).

Implementation

Momocity

A key characteristic of the PhysX implementation is an optimisation that reduces the cost of computing $\lambda' M^{-1} J^T$. This optimisation is often referred to as "momocity", a term used to indicate that we compute neither angular momentum nor angular velocity but something in-between. This system requires all constraints to update the angular velocity in a very specific way so that the intermediate angular momocity values may be unambiguously translated back to angular velocity values. This does not affect linear velocities, which are computed as expected.

If the Jacobian J has the mathematical form $\{\mu_0, \kappa_0, \mu_1, \kappa_1\}$ then the expected update order is that the angular velocities will be updated as follows:

$$\Delta\omega_0(t) = \lambda' I_0^{-1} \kappa_0 \tag{51}$$
$$\Delta\omega_1(t) = \lambda' I_1^{-1} \kappa_1 \tag{52}$$

It is also expected that the angular velocities will be projected onto the constraint using the angular terms of the Jacobian:

$$\kappa_0 \cdot \Delta\omega_0(t) + \kappa_1 \cdot \Delta\omega_1(t) \tag{53}$$

PhysX deviates from this prescription. Instead of working with the Jacobian J, PhysX internally works with a modified Jacobian $J^{(m)}$ and a modified state velocity $v^{(m)}(t)$ such that the outcome of each constraint is a change to be applied to the linear velocity and angular momocity.

$$J^{(m)} = \{\mu_0, \kappa_0^{(m)}, \mu_1, \kappa_1^{(m)}\} \tag{54}$$

$$\kappa_0^{(m)} = I_0^{-\frac{1}{2}}\kappa_0 \tag{55}$$

$$\kappa_1^{(m)} = I_1^{-\frac{1}{2}}\kappa_1 \tag{56}$$

$$v^{(m)}(t) = \{v_0(t), \omega_0^{(m)}(t), v_1(t), \omega_1^{(m)}(t)\} \tag{57}$$

$$\omega_0^{(m)}(t) = I_0^{1/2}\omega_0(t) \tag{58}$$

$$\omega_1^{(m)}(t) = I_0^{1/2}\omega_1(t) \tag{59}$$

The angular momocities are updated as follows:

$$\Delta\omega_0^{(m)}(t) = \lambda'\kappa_0^{(m)} \tag{60}$$

$$\Delta\omega_1^{(m)}(t) = \lambda'\kappa_1^{(m)} \tag{61}$$

The terms $\Delta\omega_0^{(m)}(t)$ and $\Delta\omega_1^{(m)}(t)$ no longer represent changes to angular velocity. Instead of using I_0^{-1} and I_1^{-1} to translate from angular momentum to angular velocity, PhysX has used $I_0^{-\frac{1}{2}}$ and $I_1^{-\frac{1}{2}}$ to translate from angular momentum to angular momocity. All that matters, however, is that angular momocity is translated back to angular velocity when computing $J\Delta v(t)$:

$$\kappa_0^{(m)} \cdot \Delta\omega_0^{(m)}(t) + \kappa_1^{(m)} \cdot \Delta\omega_1^{(m)}(t) \tag{62}$$

PGS Implementation

The PGS implementation closely follows the recipe set out in Equation (23). There are minor modifications to the model that introduce maximum impulse strengths; and mass and inertia scaling. Furthermore, each constraint accumulates an angular momocity rather than an angular velocity. In order to work with angular momocities, the angular terms of the constraint Jacobian are modified in a pre-solver computation. Apart from these details, however, the recipe set out earlier is followed as expected.

The key code snippet is as follows:

```
FScaleAdd(vMul, normalVel, constant)
```

with

$$\text{vMul} = -\frac{1}{r} \tag{63}$$

$$\text{normalVel} = J^{(m)} \Delta v^{(m)}(t) \tag{64}$$

$$\text{constant} = -\frac{(b - v_T + J v_{pre}(t))}{r} \tag{65}$$

The impulse applied by each constraint at each solver iteration is then clamped so that the running total does not exceed a maximum specified for each constraint. The impulse that survives the clamp is added to the running total and propagated to the linear velocities and angular momocities of the body pair as described in Section Momocity.

One final remark is that the Baumgarte multiplier has value 1. This means that the geometric error of a 1d constraint will be completely resolved in a single position iteration if there are no other constraints present in the scene.

TGS

TGS has a more complicated implementation due to the need to update the Jacobian J to account for changes to the state $s(t)$ that occur at the end of each solver iteration. In addition to updating the Jacobian it is also necessary to update the geometric error to account for changes to the state $s(t)$. It is tempting to update these with an extra execution of the PxConstraintShaderTable::solverPrep() callback at the beginning of each solver iteration. For performance reasons this does not happen. Instead, the geometric error and the Jacobian J are updated on the fly as required. This necessarily involves approximations that require description and explanation.

The approximations are as follows:

- The inertia of each body remains constant over all solver iterations of a single simulation step. This is required by the momocity optimisation.
- The linear terms of the Jacobian remain constant over all solver iterations of a single simulation step.

The vectors r_0 and r_1 for any 1d constraint require an update every time that constraint is visited because the rigid body poses are updated at the end of each solver iteration. The Jacobian $J^{(m)}$ applied at each solver iteration is therefore a mixture of constant terms computed once in the PxConstraintShaderTable::solverPrep() callback and terms that required an update with each solver iteration.

$$J^{(m)} = \left\{\mu_0, I_0^{-\frac{1}{2}} \left([\kappa_0 + (r_0 \times \mu_0))\right], \mu_1, I_1^{-\frac{1}{2}} \left[\kappa_1 + r_1 \times \mu_1\right]\right\} \tag{66}$$

The momocity Jacobian $J^{(m)}$ is used to compute an update to the geometric error. This is performed by tracking the change in position and angle that

accumulates over the solver iterations. Applying the momocity Jacobian $J^{(m)}$ necessarily requires tracking the integrated angular momocity over the course of the solver iterations rather than the integrated angular velocity. Tracking the position and rotation requires knowledge of the state velocity $v^{(m)}(t)$ rather than just $\Delta v^{(m)}(t)$ that is tracked by PGS. The key point here is that with PGS the body poses are only integrated once at the end of the solver so there is no need to know the absolute state velocity $v(t)$: it is sufficient to track $\Delta v(t)$ during the solver iterations and then apply it to the state velocity $v(t)$ once after the solver is complete. This is not the case with TGS because the body poses are integrated at the end of each solver iteration. It makes sense, therefore, for TGS to track the state velocity $v(t)$.

Prior to the start of the solver, body angular velocity is converted to angular momocity:

```
solverVel.angularVelocity = sqrtInertia * av;
```

The changes contributing to the geometric error are recorded as follows:

```
PxVec3 linearMotionVel = vel.linearVelocity;
const PxVec3 delta = linearMotionVel * dt;
...
PxVec3 unmolestedAngVel = vel.angularVelocity;
...
vel.deltaAngDt += unmolestedAngVel * dt;
vel.deltaLinDt += delta;
```

The code snippet that computes the impulse is as follows:

```
FScaleAdd(vMul, normalVel, constant)
```

This follows the PGS recipe with the exception that the value "constant" must be computed afresh each visit instead of being cached in a pre-solver preparation step. One other change is that because we track $v(t)$ instead of $\Delta v(t)$ there is no need to account for $Jv_{pre}(t)$ when computing "constant":

```
const FloatV constant = FMul(recipResponse, FAdd(bias, targetVel));
```

A final remark is that the Buaumgarte multiplier has value $\frac{1}{\sqrt{\text{nbPosIters}}}$ with nbPosIters denoting the number of position iterations. It might be tempting to code the constraint so that it resolves all of the geometric error in a single position iteration. This, however, will require a temporary velocity $\frac{\text{nbPosIters}*\text{geometricError}}{dt}$. This larger than desired velocity will propagate to all other constraints in the system, which will in turn result in unwanted consequences such as overly powerful friction forces. A compromise is to use $\frac{1}{\sqrt{\text{nbPosIters}}}$ for the Bumgarte multiplier. This undershoots the geometric error and requires more position iterations to approach a resolution to the geometric error but helps avoid the introduction of large velocities on the early position iterations that need resolution on the later position iterations.

Appendix 1: Proof of Jacobian Form

This section shall demonstrate that constrained dynamics may be cast in the form $Jv(t) = 0$.

The proof begins with the chain rule:

$$\dot{C} = \frac{\partial C(s)}{\partial s} \frac{ds(t))}{dt} \tag{67}$$

The following also holds:

$$\frac{ds(t)}{dt} = Sv(t) \tag{68}$$

with

$$S = \begin{bmatrix} E & 0 & 0 & 0 \\ 0 & Q_0 & 0 & 0 \\ 0 & 0 & E & 0 \\ 0 & 0 & 0 & Q_1 \end{bmatrix} \tag{69}$$

With the real part of the ith quaternion denoted by $q_i.w$ and the imaginary part of the ith quaternion denoted by $[q_i.x, q_i.y, q_i.z]^T$, Q_i has the following form:

$$Q_i = \begin{bmatrix} -q_i.x & -q_i.y & -q_i.z \\ q_i.w & q_i.z & -q_i.y \\ -q_i.z & q_i.w & q_i.x \\ q_i.y & -q_i.x & q_i.w \end{bmatrix} \tag{70}$$

Substituting (68) into (67) reveals the following form

$$\dot{C} = Jv(t) \tag{71}$$

with

$$J = \frac{\partial C(s)}{\partial s} S \tag{72}$$

References

[1] Andrew Witkin, "Physically Based Modeling: Principles and Practice Constrained Dynamics"

[2] "Wiki triple product rule"