Points to Remember when Using MathEngine Karma

Points to Remember When Using Karma

The following is a simple list of points to remember when working with MathEngine Karma. This list may also prove useful as a troubleshooting guide. Each point is further explained on the subsequent pages of this document.

1. Always use the check build when developing.

2. The most common reason for a 'crash' is that the pool of bodies or constraints in your MdtWorld is not big enough.

3. Care must be taken when setting a range of certain object parameters.

such as mass and size so as not to lose accuracy.

4. If large masses are needed or large forces used the parameter epsilon may need to be decreased.

5. Ensure that an objects inertia tensor corresponds to its mass, collision size and hence is sensible for the torque applied to it.

6. Applying forces or torques to bodies will not slow down a simulation.

7. Adding forces that change rapidly, for example springs, between timesteps can cause simulations to gain energy and become unstable. Use constraints instead.

8. To speed up your simulation use small, separate partitions.

9. To move a body you can:

- apply a force to it
- set its velocity
- set its position

You should always use a force where possible. You can set velocity or position directly. However, care should be taken when doing this.

10. The visualized physical object you create is composed of SEPARATE dynamic, collision and graphics objects.

11. Use a single dynamic body. Do not try to fix dynamic bodies rigidly together.

12. Mass distribution, Moment of Inertia, Inertia Tensor, Mass Matrix refer to the same property. Set it using MdtBodySetInertiaTensor()

Here is a more detailed list of most of the points addressed above.

1. *Always use the check build when developing.*

Always use the check or debug builds when developing because they provide lots of useful warnings. Note that MathEngine does not provide debug builds itself. Debug builds for publicly released libs should be built yourself.

2. *The most common reason for a 'crash' is that the pool of bodies or constraints in your MdtWorld is not big enough.*

If your simulation does crash, check that you are using the check / debug library and look for any warnings. The most common problem (the most common reason for a 'crash') is that the pool of bodies or constraints in your MdtWorld is not big enough. You should increase this. Use the release libraries to get a measure of performance of a working check / debug build only.

3. Care must be taken when setting a range of certain object parameters.

Note that problems can occur if you have a large range of values in the
case of certain object properties. For example, certain ratios relating to mass must be close to one.   Here are some guides to world construction that should be followed:

- Largest mass to smallest mass ratios should ideally not exceed 100.

- It is sensible that the velocity of an object is such that the distance
it moves in a time step is less than the object size. The reason for this is such that collisions are more easily detected.

- The angular velocity should be such that the angle swept out in a
timestep does not exceed 60 degrees. The exceptions to this are in the
carwheel joint that was specifically designed for high-speed rotation, or for bodies where the fast rotation axis (keaBody.fastSpinAxis) has been set. This is because floating-point is most accurate for values in the middle of its range. In decimal equivalent, the working range is between about 10e-6 & 10e+6. Adding and subtracting values whose exponents differ by more than this results in errors in the mantissa. Similarly multiplying 2 numbers with large exponents and dividing numbers with small exponents causes problems in that the exponent suffers overflow and underflow respectively.

The critical point is that mass scales as the third power of object size
and moments of inertia scale as the 5th power. E.g.: A cube has mass = (density)x(volume) = (density)x(length cubed). This cube has inertia = (mass)x(length squared) where mass is given above. If you have 2 objects, one of size 0.1, the other of size 10, a difference factor of a mere 100, the mass difference is of the order 103/0.13 = 106, and the inertia difference = 105/0.15 = 1010. This shows how accuracy is
lost.

4. *If large masses are needed or large forces used the parameter epsilon may need to be decreased.*

If you use larger masses, you may need to decrease 'epsilon' (using MdtWorldSetEpsilon). Epsilon is a ‘global constraint

softness' and directly effects the constraint solution. If the forces in your simulation are stiff, you will need to make constraints (contacts, joints) 'harder'.
For large or small epsilon, the maths describing the system will be solved, but the way the solution is arrived at results in different visual behavior of the system.
 - As epsilon decreases it takes longer to home in on a mathematical solution within the required bounds. As an example, consider a large mass colliding with the ground. The contact is modelled by a spring. Epsilon relates to the stiffness of the spring. For more realistic behavior a stiff spring is needed i.e. small epsilon. However, it will take longer to arrive at a solution within the bound specified. You might get a warning message saying that the number of LCP (don’t worry about LCP here) cycles have been exceeded. This means that while the solution is almost certainly going to be good enough for your application it might not have fallen into the required range within the number of LCP cycles specified.
 To decrease the time (i.e. number of LCP cycles) to find a solution of the given accuracy you should increase epsilon. However, this makes it more difficult to model systems that require stiff springs - e.g. the large mass system described above. A large mass object in collision with the ground will show ‘springiness’ in the contact. Hence the term ‘global constraint softness' that is used to describe epsilon physically.
 Ideally you should set the largest epsilon possible so that your system behaves properly as visually observed. We recommend that you tweak with epsilon. While epsilon is not limited, a sensible working range is between 10e-5 and 0.1.

5. *Ensure that an objects inertia tensor corresponds to its mass, collision size and hence is sensible for the torque applied to it.*

Ensure that an objects inertia tensor corresponds to its mass and collision size. Even if it is a crude approximation such as that of a sphere bounding your object, this will help. If you create a large heavy body with a small inertia tensor, this can cause jittering and odd behavior. This is because physically it would be like, for example, creating a sphere with nearly all of its mass in the centre. It is easy to rotate this object because of its small inertia. If you apply even what appears to be a sensible force (strictly a force that acts to rotate the object i.e. a torque) in comparison to the object size and mass, the small inertia results in rapid rotation.

6. *Applying forces or torques to bodies will not slow down a simulation.*
7. *Adding forces that change rapidly, for example springs, between timesteps can cause simulations to gain energy and become unstable. Use constraints instead.*

Applying forces or torques to bodies will not slow down your simulation. However, adding forces that change rapidly between timesteps (e.g. springs) can cause your simulation to gain energy and become unstable. You should use a constraint for this instead.

8. *To speed up your simulation use small, separate partitions.*

The speed of simulation is, in the worst case, proportional to the number of constraints in a partition* cubed. For example, a stack of 5 boxes can be 8 times faster than a stack of 10 boxes ($10^3/5^3$). Hence you can speed up your simulation by using lots of small, separate partitions.
* A 'partition' is a group of bodies connected by constraints. Constraints to the 'world' do not connect partitions. So two boxes sat on the ground are two partitions, but two boxes in a stack are 1 partition.

9. *To move a body you can:*

- apply a force to it
- set its velocity
- set its position

When you want to reposition or move an object, there are three ways that you can do this using Karma. You can set the object position directly, set the object velocity or apply a force to it. The preferential order for doing this is wherever possible use forces, the next best option is to set the velocity, and finally repositioning directly which will work but is not recommended. To think about this you should consider how objects move in the real world. Leaving quantum behavior well alone, within the Newtonian framework - our perceived reality - objects do not simply move instantly from one place to another (like setting position). Similarly, they don’t suddenly develop a particular speed i.e. they are not stationary one instant and the next they are moving (like setting velocity). Rather there is a smooth increase in the velocity and position changes gradually as a force is applied. It is the same with the simulation software. While you can set position and velocity, you should use forces wherever possible. Mathematically we say that the functions should be continuous i.e. there are no sudden kinks or discontinuities that correspond to position or velocity being set rather than force. Setting position results in bigger discontinuities than setting velocity, hence the reason for ‘preferring’ velocity to position.
A problem from setting position directly is that an object may inadvertently be place inside another object. Or if you reposition an object that is joined to another body or that is part of another structure.
An example of a problem from setting velocity directly is that an object may unintentionally be directed at high speed toward another object. Or if a velocity is given to an object attached to another or that is part of another structure
However, please note that if your object is isolated and is being moved to a position not occupied by another object, then setting position or velocity is okay. Likewise, you can do this as long as you take care of the objects that the object you want to move is attached to. Hence, either detach it or move the entire structure (partition).

11. *Use a single dynamic body. Do not try to fix dynamic bodies rigidly together.*

You should not try to fix dynamic bodies rigidly together, but rather use a single dynamic body. Fixing objects rigidly together causes a large performance hit - relatively speaking, as the fixed joint is not necessary - on the constraint solver. The constraint solver works out the physical properties of the state of the system as the system evolves. A fixed joint constrains the 6 degrees of freedom (3 linear and 3 rotational) between two objects and as a result is the most computationally expensive constraint to deal with. You can create composite structures by attaching a composite collision model and several graphic objects to a single rigid body. The only knowledge that a dynamic body has about its extent is through its mass distribution. You should set the mass matrix of your single dynamic body to something close to that of your perceived structure consisting of multiple rigid dynamic bodies.