# FLEX - User Manual

NVIDIA

August 21, 2013

## Contents

# 1 Rigid Bodies

## 1.1 Overview

Rigid bodies in FLEX are handled quite differently from a traditional rigid body solver such as PhysX. Instead of convex shapes such as hulls, boxes or capsules, FLEX represents shapes as a collection of particles defined by an array of indices into the scene's global particle list.

At creation time FLEX calculates the particles' center of mass and the local offset of each particle, which is stored as the shape's "rest pose".

During simulation collisions are still handled for each particle individually. This means that after a collision the particles are typically no-longer in their rigid configuration (Figure 1). In order to restore rigidity, the least squares best rigid transform is found to match the deformed particle positions back to the rigid shape.

Note that the solver does not even explicitly track the linear or angular velocity of the body, these quantities are simply derived from the particles referenced.

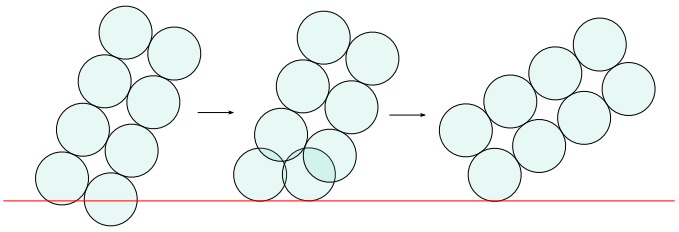

Figure 1: Showing the steps of collision detection and response.

## 1.2 Limitations

### 1.2.1 Structured Stacking

Dense stacking is possible in some situations, such as when shapes are perfectly aligned, however, as soon as particles become *even slightly* unaligned, the collision normal will rotate downward and quickly cause the stack to collapse.

This can make setting the initial conditions for a stack of objects tricky to do by hand. Apart from generating stacks programatically, some solutions may be to freeze shapes until their first collision, or to glue shapes in place with breakable constraints.

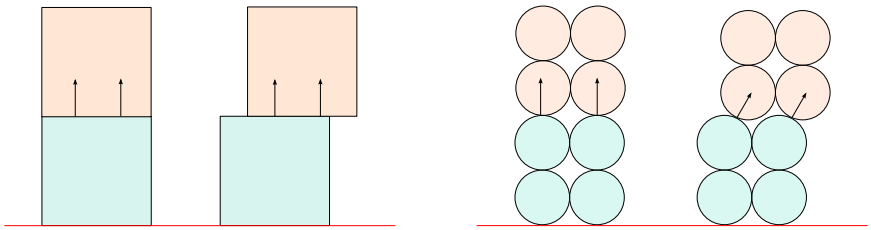

Figure 2: The difference in collision normals between a rigid body solver (left) and a particle based solver (right). As the objects move out of alignment the particle based normals rotate, making stacking difficult.

### 1.2.2 Interpenetration

Collision tests between particles are done discretely and without any global knowledge of the rigid shape. This means that if particles pass through each other the rigid shapes can interpenetrate, and in the worst case may become locked together.

At present the best recommendation is to try and avoid interpenetration through a combination of the following:

- Increase the overlap between particles in the shape
- Increase the particle radius
- Increase the number of substeps
- Limit the maximum particle velocity

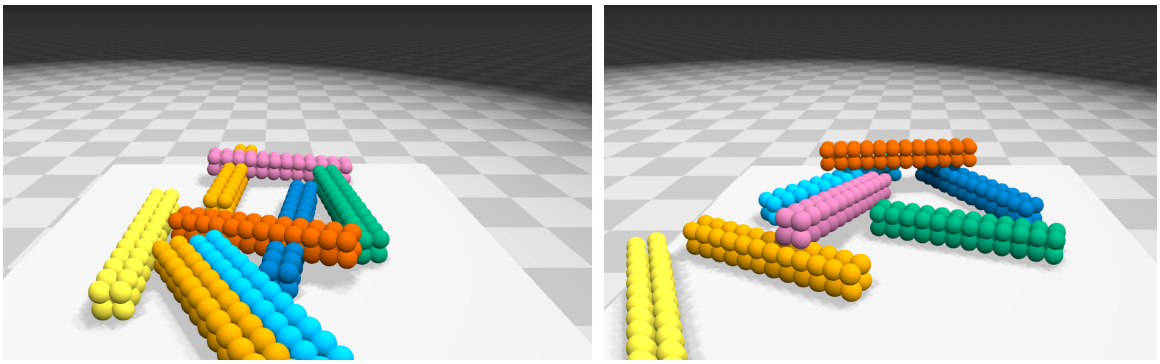

Figure 3: Shapes simulated with a large time step may interpenetrate and fail to separate. **Left:** 1 step/frame, **Right:** 2 steps/frame.

### 1.2.3 Relative Size

FLEX uses the same radius for all particles, this places a lower bound on the smallest possible feature that can be represented in the simulation. But it also places some limitations on the *largest* object that can be efficiently represented.

As the scale of an object increases the number of particles required to represent it grows in proportion to the cube of the scale factor. For example, if the smallest object consists of 1 particle with a radius of 1cm, then a square object with dimensions of 10cm will require 1000x the number of particles ($10 \times 10 \times 10$).

Because of this high growth-rate we recommend the following guidelines when designing content:

- Limit the ratio of the smallest feature to largest dimension to 10:1
- Try sampling only the shape's surface with particles
- Use traditional collision shapes for large / flat objects (e.g. convex hulls)

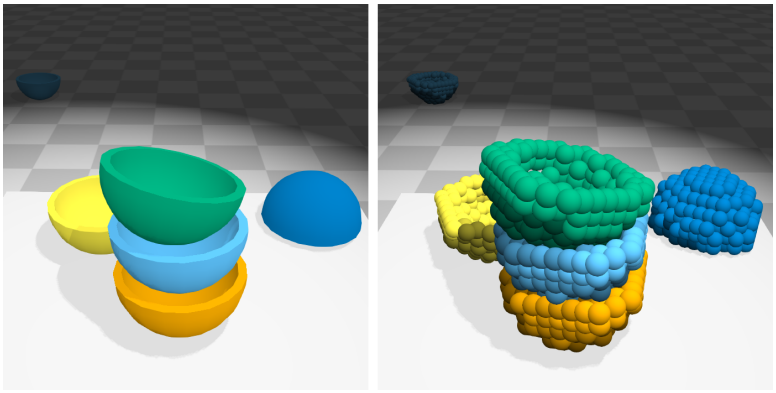

Figure 4: Thin features like the edge of this bowl are not well represented using particles. Note the visible gap between graphical meshes in contact.

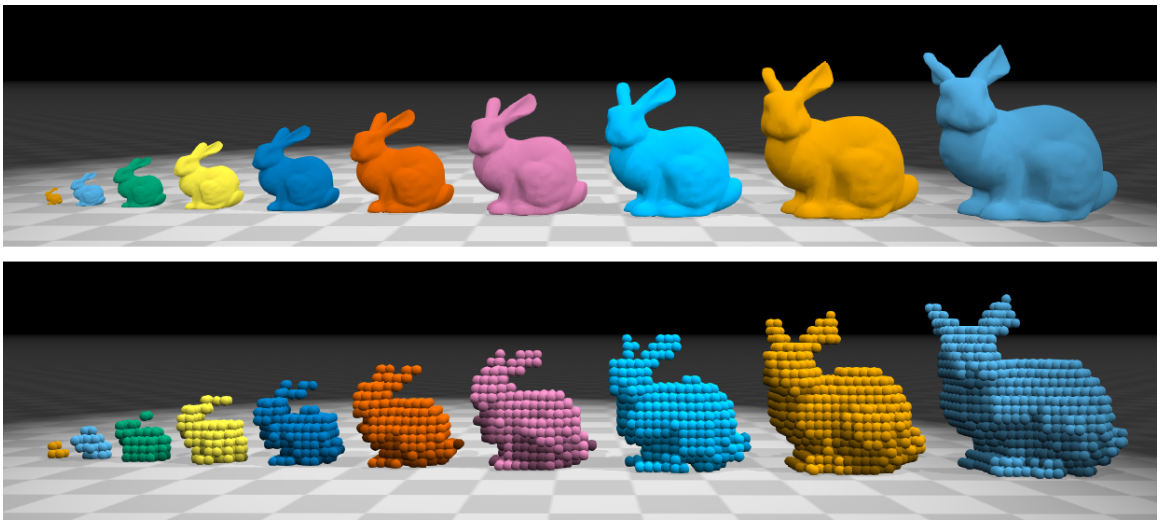

Figure 5: Showing the increase in particle count as shape size increases.

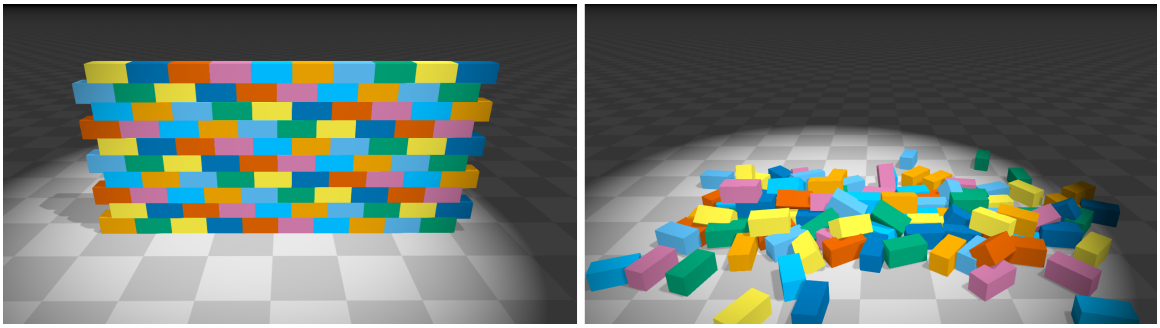

Figure 6: Brick work, when perfectly aligned, stacks stably (left). However small perturbations in initial conditions easily collapse the stack (right).

## 1.3 Applications

Taking into account these limitations, the table below summarizes the strengths and weaknesses of FLEX rigid bodies and suggests some suitable applications:

| **Strength** | **Example** |
|---|---|
| Environmental debris | Bullet hits, rocks, bottles and cans on a table |
| Unstructured piling | Piles of trash, driving through a fruit stand |
| Non-convex shapes | Irregular stones, bananas |
| Two-way interaction | Parachuting bunnies |

| **Weakness** | **Example** |
|---|---|
| Thin or sharp objects | Floor tiles, glass shards |
| Structured stacking | Brickwork, arches |
| Large relative sizes | Arena destruction |
| Collision filtering, raycasts | Character controller |

## 1.4 Examples

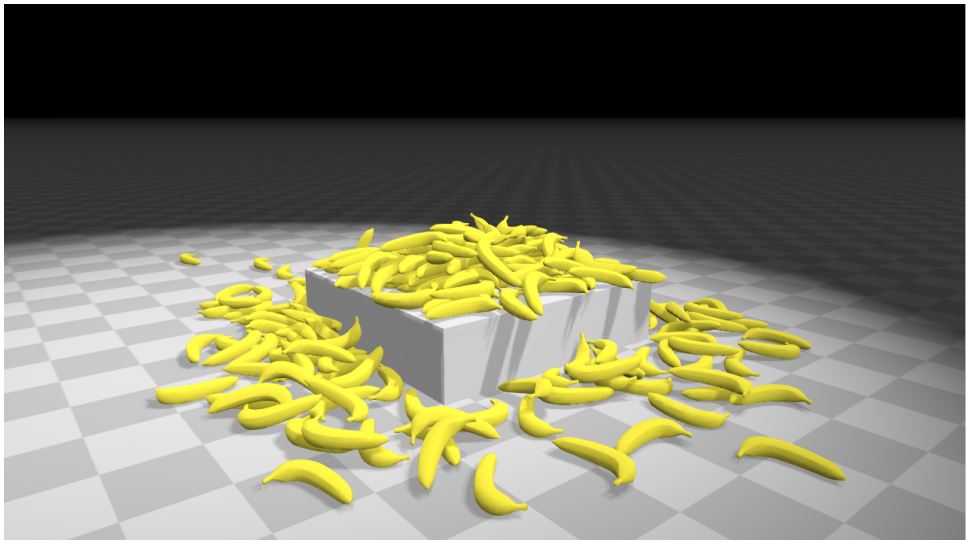

Figure 7: Piling non-convex bananas, each banana consists of around 14 particles.

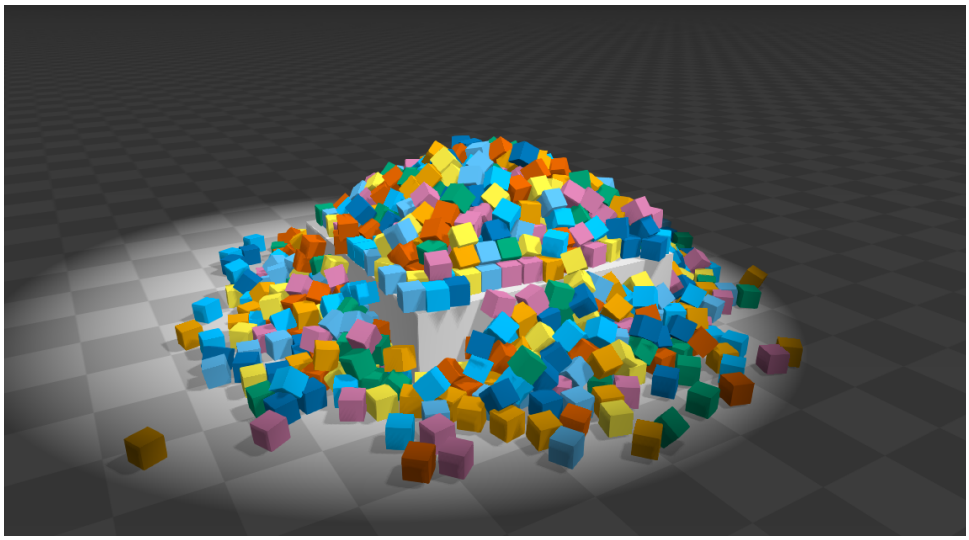

Figure 8: Piling blocks, each block consists of 4 particles.