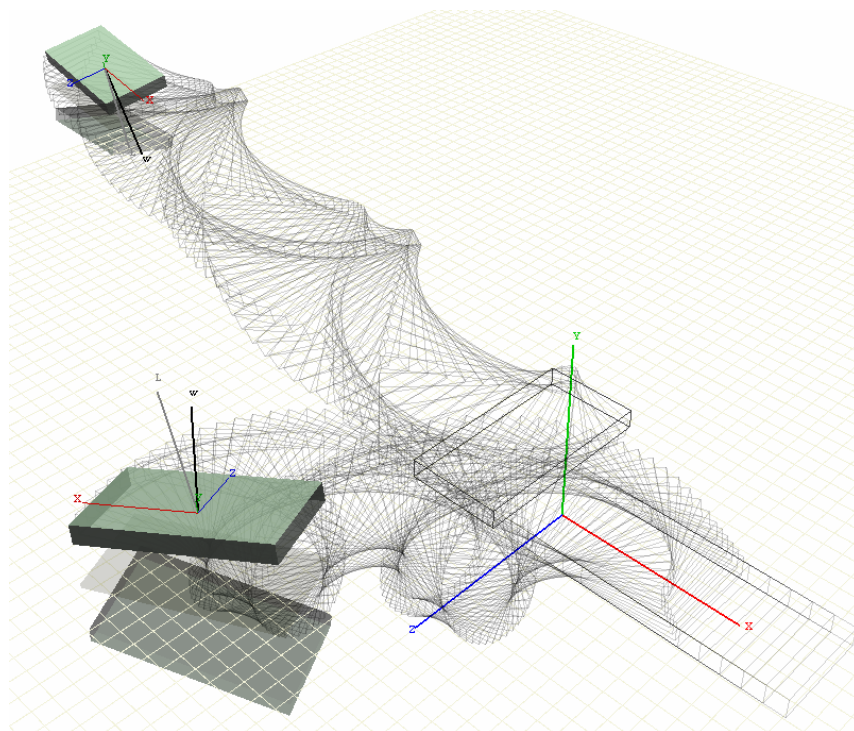# Contacting Rigid Bodies

Mikica B Kocic

2012-05-16, v3

1 Background

The purpose of this computer lab was to simulate collisions between rigid bodies using either the impulse method or the penalty method, and to validate the simulation by studying the conservation of energy, linear momentum and angular momentum. Advanced level tasks were to improve the simulation by including frictional contacts, more objects, more complicated collision geometries, articulated bodies using constraints and user control over the objects.

2 Implementation and Simulation

2.1 Theory

An overall algorithm for a simulation as well the semi-implicit Euler algorithm for solving rigid body equations of motion, are based on Section 2.3 *The Simulation Algorithm* and Subsection 7.3.3 *Time-Integration in 3D* in *Notes on Discrete Mechanics*, respectively.

On the other hand, rigid-body simulation algorithms, where linear and angular momenta are state variables instead of linear and angular velocities, are based on equations of motion found in Chapter 3 *Rigid Body Motion* in [1] and Section 2.11 *Rigid Body Equations of Motion* in [5].

Rigid body collision *detection* algorithms are based on Chapter 5 *Basic Primitive Tests* in [3] and Chapter 13 *Generating Contacts* in [4], whereas rigid body collision *response* algorithms are based on impulse transfer method found in Section 5.2.2 *Collision Response for Colliding Contact* in [1], Chapter 14 *Collision Resolution* in [4] and Section 8 *Colliding Contact* in [5].

Finally, usage of quaternions for representing spatial rotations and derivatives of time-varying quaternions is based on Chapter 10 *Quaternions* in [1].

2.2 Implementation

The resulting rigid body simulation framework, World of Rigid Bodies or WoRB, is implemented in C++ and it uses GLUT and OpenGL API for animations [2]. The WoRB framework can be built either as a stand-alone simulation test-bed application, a MATLAB mex file or as a C++ library.

Quaternions

In this implementation, there are no vectors and matrices as separate entities; the implementation rather uses quaternions to represent all variables, including state variables like position, orientation, linear or angular momentum. Rotation and translation matrices are kept as q-tensors, which are in fact 4-tuples of quaternions. Quaternions and q-tensors are encapsulated as C++ classes and defined in `Quaternion.h` and `QTensor.h` modules, respectively. The theoretical framework for these classes is implemented as '`Quat.m`' Mathematica package and given in Appendix B, with the accompanying notebook given in Appendix C.

Number of Spatial Dimensions

The original request was to implement algorithms for colliding rigid bodies in 2D. In this implementation, however, equations of motion and collisions are always simulated in 3D. Simulations in 2D are considered as a special case where rectangular rigid bodies are represented by very thin plates (cuboids with negligible thicknesses) with motion restricted to the horizontal (*Oxz*) plane only.

Rigid Body Motion Equations

The theoretical framework for rigid body motion algorithms is implemented as Mathematica notebook and given in Appendix A.

Program Code

The WoRB core implementation consists of ~7k lines of code & comments that are distributed in around 20 modules. The WoRB code metrics is presented in Table 2.2-1.

Table 2.2-1:   The WoRB Code Metrics

Module Name	Type	LOC	COM	MVG	L_C	M_C
Const	(S)	30	38	6	0.789	0.158
Quaternion	(S)	193	125	61	1.544	0.488
SpatialVector	(S)	8	8	0	–	–
QTensor	(S)	357	157	46	2.274	0.293
Geometry	(S)	88	31	41	2.839	1.323
Cuboid	(S)	425	253	95	1.680	0.375
Sphere	(S)	100	71	19	1.408	0.268
TruePlane	(S)	11	7	0	–	–
HalfSpace	(S)	11	8	0	–	–
RigidBody	(S)	191	190	18	1.005	0.095
RigidBodies	(S)	43	20	11	2.150	0.550
Collision	(S)	278	235	34	1.183	0.145
CollisionResolver	(S)	197	147	40	1.340	0.272
WoRB	(S)	120	42	10	2.857	0.238
WorldOfRigidBodies	(S)	95	65	8	1.462	0.123
WoRB_TestBed	(G)	717	363	93	1.975	0.256
Ball	(G)	51	18	8	2.833	0.444
Box	(G)	56	16	13	3.500	0.812
GLOrthoScreen	(G)	26	17	0	1.529	–
GLTransform	(G)	23	25	1	0.920	–
GLUT_Framework	(G)	107	31	20	3.452	0.645
GLUT_Renderer	(G)	16	17	0	–	–
WoRB_MexFunction	(M)	158	70	16	2.257	0.229
Mex::Scalar	(M)	52	12	12	4.333	1.000
Mex::String	(M)	49	20	10	2.450	0.500
Mex::Logical	(M)	54	14	14	3.857	1.000
Mex::Matrix	(M)	167	74	50	2.257	0.676

where

- **Type**: (**S**) = simulation related module, (**G**) = animation related module (uses OpenGL), and (**M**) = MATLAB mex-file related module.

- **LOC** = Lines of Code
  Number of non-blank, non-comment lines of source code counted by the analyser.

- **COM** = Lines of Comments
  Number of lines of comment identified by the analyser

- **MVG** = McCabe's Cyclomatic Complexity
  A measure of the decision complexity of the functions which make up the program.

- **L_C** = Lines of code per line of comment
  Indicates density of comments with respect to textual size of program

- **M_C** = Cyclomatic Complexity per line of comment
  Indicates density of comments with respect to logical complexity of program

Program Documentation

The program documentation consists of ~2.5k lines of comment that follows javadoc convention. The resulting documentation, generated from these comments using Doxygen tool, can be found in `doc` subfolder with `doc/index.html` as the main page (opening `README.html` in the top folder will redirect to this page).

The other useful page is 'Files' (`doc/files.html`) which shows the source file list with their brief descriptions. The good point to start browsing the WoRB code are `Main.cpp` and `mexFunction.cpp` files, which both contain main program entry points; the first for a stand-alone application and the second for a MATLAB mex function. Their respective include dependency graphs are shown on Figure 2.2–1.
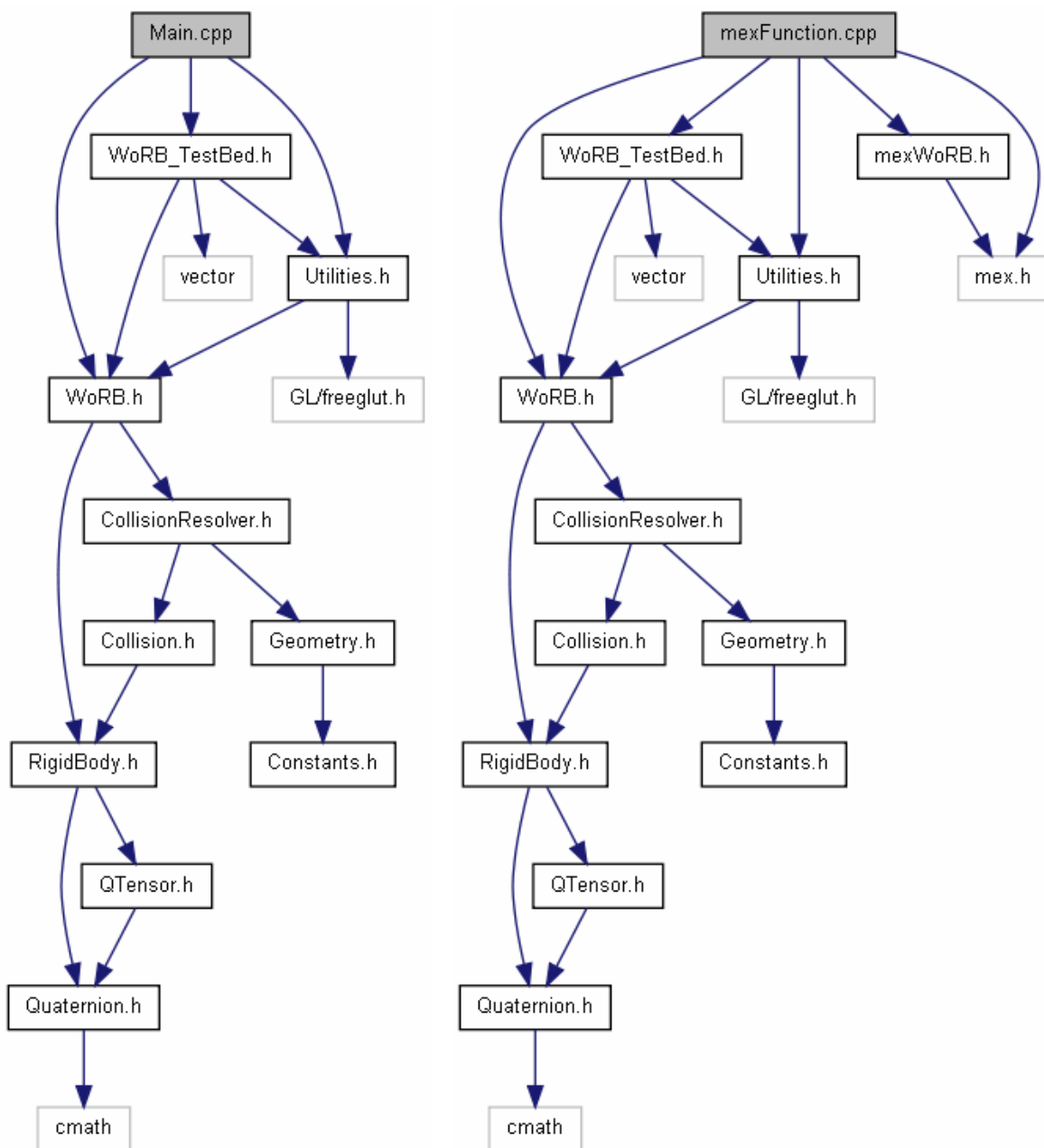

Figure 2.2–1:   Include dependency graphs for `Main.cpp` (left) and `mexFunction.cpp` (right)

2.3 Compiling the Code

The code can be compiled either for stand-alone execution, or to be used as a mex function in MATLAB.

The animation part requires OpenGL libraries, which are usually distributed with an operating system, and the freeglut (GL Utility Toolkit) that can be downloaded from http://freeglut.sourceforge.net/

The default distribution contains `mexw32` and `mexw64` binaries compiled with MS VS 2010 compiler. To rebuild binaries on Windows, use VS solution in folder `vs2010`; the accompanying freeglut 2.8.0 development library and DLL files for win32 and win64 platforms can be found in `freeglut` folder.

To rebuild binaries on Linux, use `Makefile`; just type `make` on the command prompt.

To rebuild binaries under the MATLAB, just execute `make` m-file.

Note: On Linux, if you have non-standard location for freeglut, i.e. if it is not installed as a system package, you will need to configure `GLUT_INC` and `GLUT_LIB` variables in `Makefile` and `make.m`.

Here is a sample log showing the compilation process to get the stand-alone application on Linux:

```
[user@gea WoRB]$ make rebuild
 [RM   ]   obj/*.o
 [C++  ]   obj/Constants.o
 [C++  ]   obj/WoRB.o
 [C++  ]   obj/CollisionDetection.o
 [C++  ]   obj/ImpulseMethod.o
 [C++  ]   obj/PositionProjections.o
 [C++  ]   obj/Platform.o
 [C++  ]   obj/Utilities.o
 [C++  ]   obj/WoRB_TestBed.o
 [C++  ]   obj/Main.o
 [LD   ]   bin/WoRB
[user@gea WoRB]$
```

Here is a sample log showing the compilation process to get mex-file on Windows:

```
>> make rebuild
mex -O -outdir obj -c src\Constants.cpp
mex -O -outdir obj -c src\CollisionDetection.cpp
mex -O -outdir obj -c src\ImpulseMethod.cpp
mex -O -outdir obj -c src\PositionProjections.cpp
mex -O -outdir obj -c src\WoRB.cpp
mex -O -outdir obj -c src\Platform.cpp
mex -O -I./freeglut/include -outdir obj -c src\Utilities.cpp
mex -O -I./freeglut/include -outdir obj -c src\WoRB_TestBed.cpp
mex -O -I./freeglut/include -outdir obj -c src\mexFunction.cpp
mex  -output bin\WoRB.mexw32 ...
 obj\mexFunction.obj ...
 obj\Constants.obj ...
 obj\CollisionDetection.obj ...
 obj\ImpulseMethod.obj ...
 obj\PositionProjections.obj ...
 obj\WoRB.obj ...
 obj\Platform.obj ...
 obj\Utilities.obj ...
 obj\WoRB_TestBed.obj ...
 ./freeglut/lib/freeglut.lib
>>
```

2.4 Running Stand-Alone Simulations

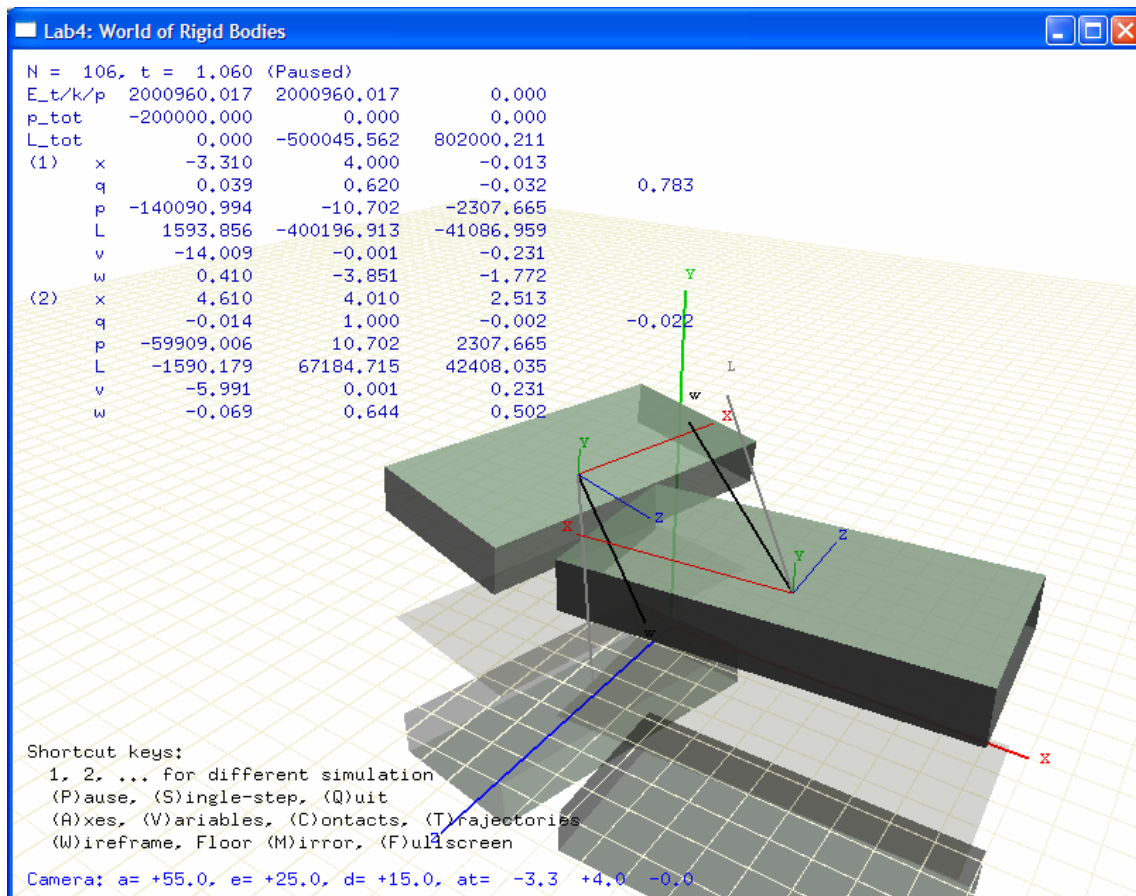To run a stand-alone simulation, just start the WoRB executable. The gui should appear:

Figure 2.4–1:   Snapshot of the WoRB simulation window.

The application window is logically divided into three areas:

central area	Contains the simulated system with the ground floor, bodies and trajectories
top-left area	Contains local time, total energy levels and momenta of the system, as well the state variables of up to 4 bodies. The body state variables are displayed only while simulation is paused or single-stepped.
bottom-left area	Contains camera view position with the short help about shortcut keys above

One can use the following short-cut keys to control the application:

1, 2, …	Start different predefined simulation
A	Toggle displaying axes (and angular velocity and momentum) of the bodies
C	Toggle displaying contact normals during collisions
F	Toggle full-screen mode
H	Toggle displaying help and camera position
M	Toggle displaying rigid bodies mirrored in the ground floor
P or \<space\>	Pause/continue simulation (toggle simulation execution)
S or \<enter\>	Simulation single-step; keep holding \<enter\> for slowed-down simulation
T	Toggle displaying/tracking trajectories of the bodies
W	Toggle displaying objects as wire-frames vs. solid bodies
F1, F2 …	Follow the body #1, #2 … with the camera (while it moves)
F11	Point the camera towards the coordinate origin, at 55 degrees from *Oxz* side
F12	Point the camera towards the coordinate origin, at 90 degrees from above

2.5 Running MATLAB Simulations

To run MATLAB lab4 simulation scripts, you need to have `bin` folder (holding mex-file) in your MATLAB search path.

To ensure this, be sure to execute `make` at the beginning of the simulation session. This will guarantee both that you have the most recent executables, as well that you have `bin` folder in the search path. The other way is to manually add `bin` to search path using `addpath` command.

To start a lab4 simulation, just call `lab4` function with the simulation config-id as the arugment. The gui simulation window will appear, similar to the one shown on the previous Figure 2.4–1.

For example, to start lab4 simulation with config-id '2c' issue command

```
>> lab4 2c
```

Executing lab4 without arguments will start a default 'demo' simulation of two colliding bodies.

The simulation parameters (together with simulated bodies in the system) are configured (constructed) by the `lab4_defs` function, according to the selected config-id. For the full list (and descriptions) of possible configuration config-ids, see `lab4_defs.m`.

3 Results

3.1 Basic Level Tasks

The default simulation parameters were $L = 10$ m, $m = 0.1$ kg and $v = 1$ m/s, without any dissipation or friction. Since the simulation was performed in 3D, it was assumed that bodies had negligible thickness of $d = 0.01$ m.

The resulting trajectories of the rigid bodies during the collision are shown on the top on Figure 3.1–1 where the point of contact, at the time of the collision, is highlighted on the bottom image.
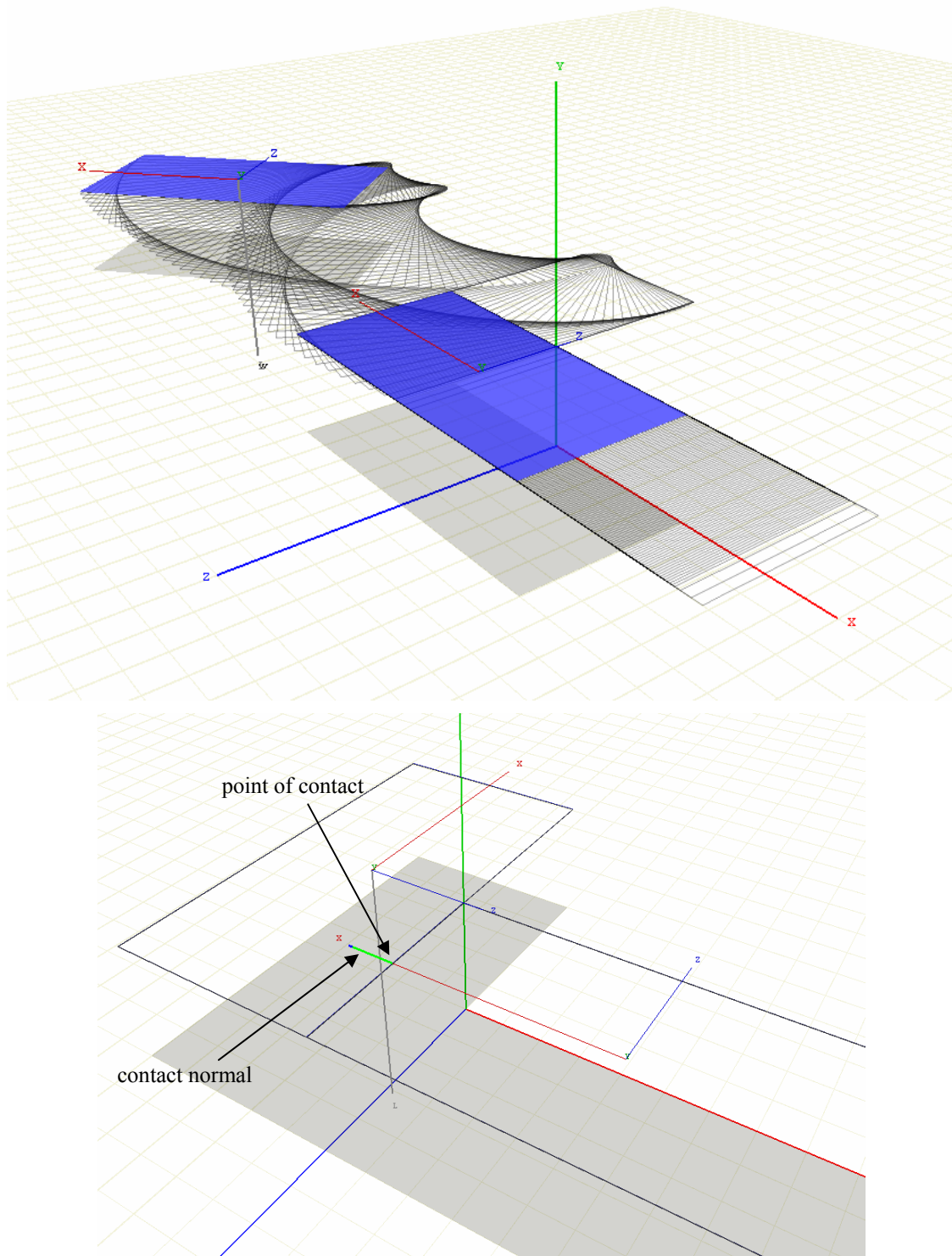

Figure 3.1–1:   The resulting trajectories of the rigid bodies during the collision (config-id '1a')

The corresponding change in the linear and angular momentum and velocities of the bodies are shown in Table 3.1-1. (Note that the angular momentum of the bodies, as shown in the table, is their total angular momentum consisting of the center of mass angular momentum summed with the angular momentum induced by the angular velocity.)

The total linear momentum, angular momentum and energy levels of the system over the time are also shown on Figure 3.1–2.

From data in Table 3.1-1 and Figure 3.1–2, it can be concluded that the total linear momentum, angular momentum and kinetic energy are all preserved during the collision.

Table 3.1-1: Changed physical quantities during the collision (config-id '1a').

Quantity		Before collision			After collision *)		
		body 1	body 2	system	body 1	body 2	system
Kinetic energy	E_k / J	0.000	0.050	0.050	0047	0.003	0.050
Linear momentum	p_x / $(\text{kg}\cdot\text{m}\cdot\text{s}^{-1})$	0.000	−0.100	−0.100	−0.077	−0.023	−0.100
	p_y / $(\text{kg}\cdot\text{m}\cdot\text{s}^{-1})$	0.000	0.000	0.000	0.000	0.000	0.000
	p_z / $(\text{kg}\cdot\text{m}\cdot\text{s}^{-1})$	0.000	0.000	0.000	0.000	0.000	0.000
Angular momentum	L_x / $(\text{kg}\cdot\text{m}^2\cdot\text{s}^{-1})$	0.000	0.000	0.000	0.000	0.000	0.000
	L_y / $(\text{kg}\cdot\text{m}^2\cdot\text{s}^{-1})$	0.000	0.000	−0.250	−0.192	-0.058	−0.250
	L_z / $(\text{kg}\cdot\text{m}^2\cdot\text{s}^{-1})$	0.000	0.300	0.300	0.000	0.300	0.300
Velocity	v_x / $(\text{m}\cdot\text{s}^{-1})$	0.000	−1.000	n/a	−0.769	−0.231	n/a
	v_y / $(\text{m}\cdot\text{s}^{-1})$	0.000	0.000	n/a	0.000	0.000	n/a
	v_z / $(\text{m}\cdot\text{s}^{-1})$	0.000	0.000	n/a	0.000	0.000	n/a
Angular velocity	ω_x / s^{-1}	0.000	0.000	n/a	0.000	0.000	n/a
	ω_y / s^{-1}	0.000	0.000	n/a	−0.185	0.000	n/a
	ω_z / s^{-1}	0.000	0.000	n/a	0.000	0.000	n/a

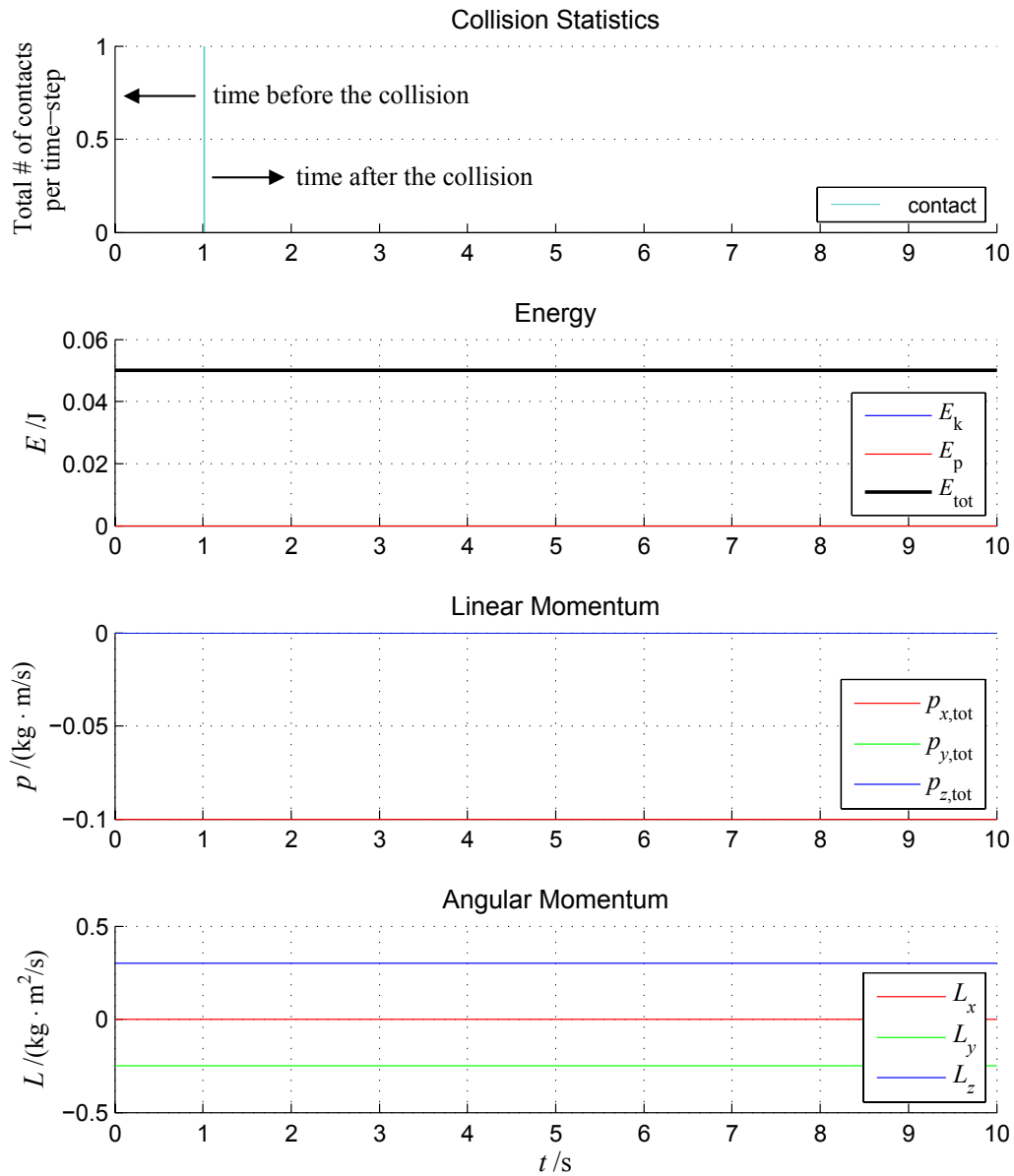*) Quantities changed during the collision are highlighted in grey.

Figure 3.1–2:   The total energy, linear momentum and angular momentum of the system over time (config-id '1a').

Note that the resulting trajectories of the rigid bodies are quite sensitive to detected contact point.

If the *x*-axis of an oncoming cuboid is not completely normal to the *z*-aligned edge of a stationary cuboid during the collision, i.e. if, for example, orientation of the oncoming rigid body is ±0.01 % around *y*-axis, then one or the other vertex of the oncoming cuboid will be detected as the contact point, and the resulting trajectories will look as on Figure 3.1–3.
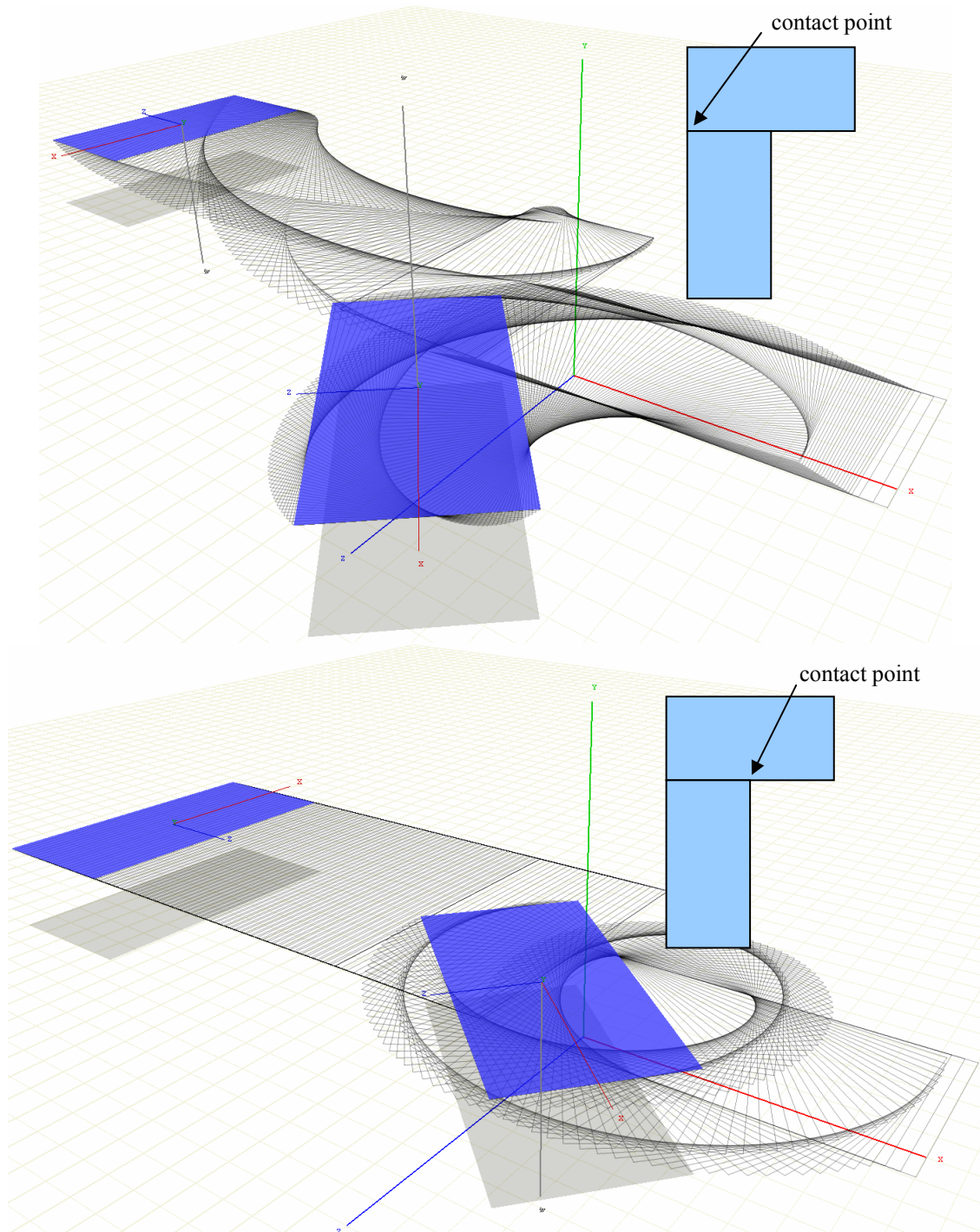
Figure 3.1–3: Config-ids '1b' and '1c' – The resulting trajectories of the rigid bodies during the collisions with slightly disturbed rotation around *y*-axis (config-ids '1b' and '1c')

The snapshot of a more realistic simulation, where cuboid bodies each has thickness $d = 1$ m and mass $m = 10^4$ kg, is shown on Figure 3.1–4. Initial velocity of the oncoming body is $v = 10$ m/s. The total linear momentum, angular momentum and kinetic energy levels of the same system over time are shown on Figure 3.1–5.

```
N =  769,  t =  7,690  (Paused)
E_t/k/p    501009,589     501009,589          0,000
p_tot     -100000,000          0,000          0,000
L_tot          -0,036    -249821,926     309992,728
(1)    x      -55,190          2,514        -11,741
       q       -0,096          0,064          0,038       -0,993
       p    -79439,135       -797,905     -19303,563
       L     4336,357    -112755,275     -16214,082
       v       -7,944         -0,080         -1,930
       w        0,479         -1,123         -0,092
(2)    x       -9,210          3,586         14,241
       q        0,097         -0,968          0,051        0,226
       p    -20560,865        797,905      19303,563
       L     -4282,101     110605,481      16053,410
       v       -2,056          0,080          1,930
       w        0,220          1,115          0,093
```
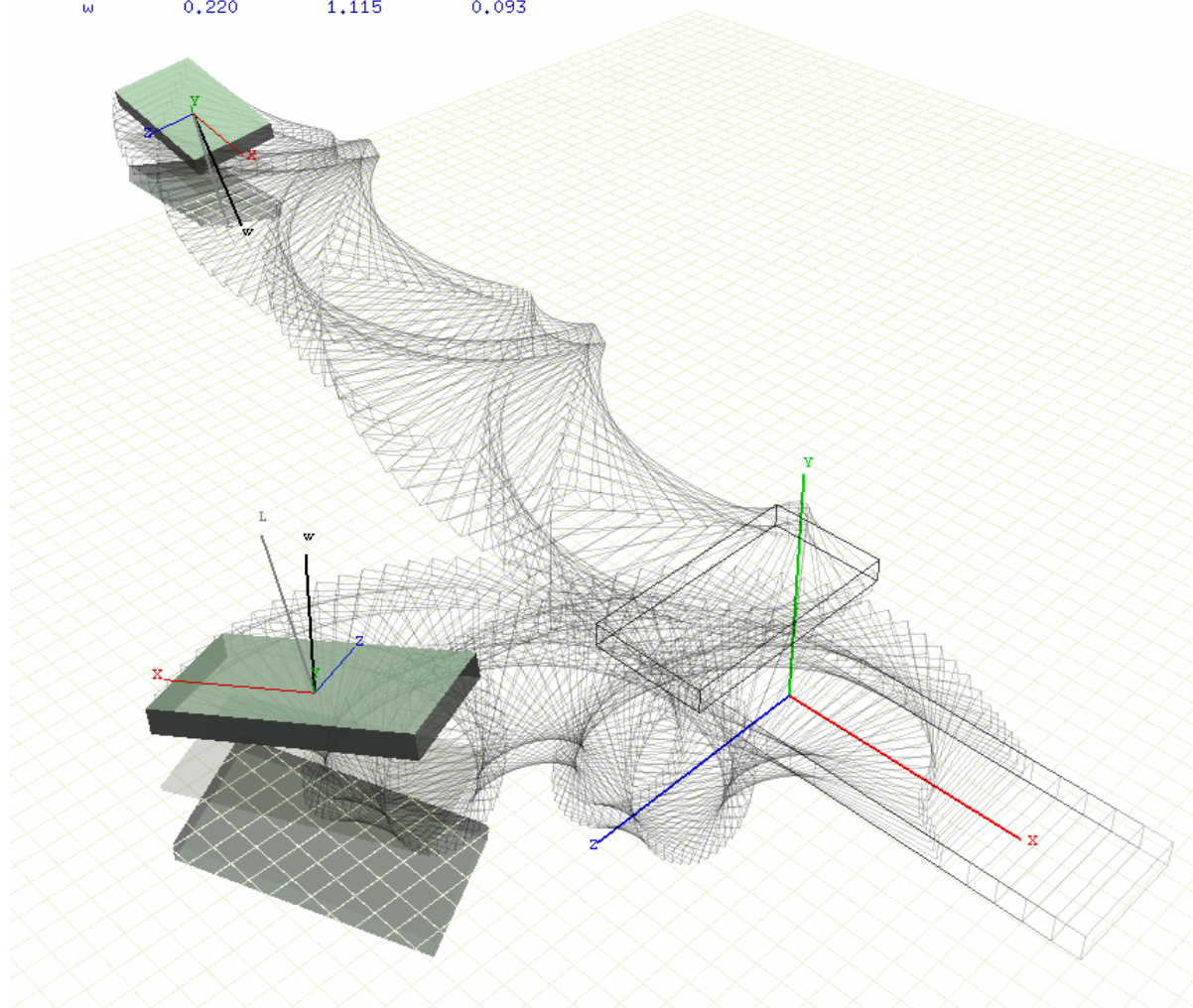

Figure 3.1–4:   The resulting trajectories of the rigid bodies during a more realistic collision (config-id '2d').
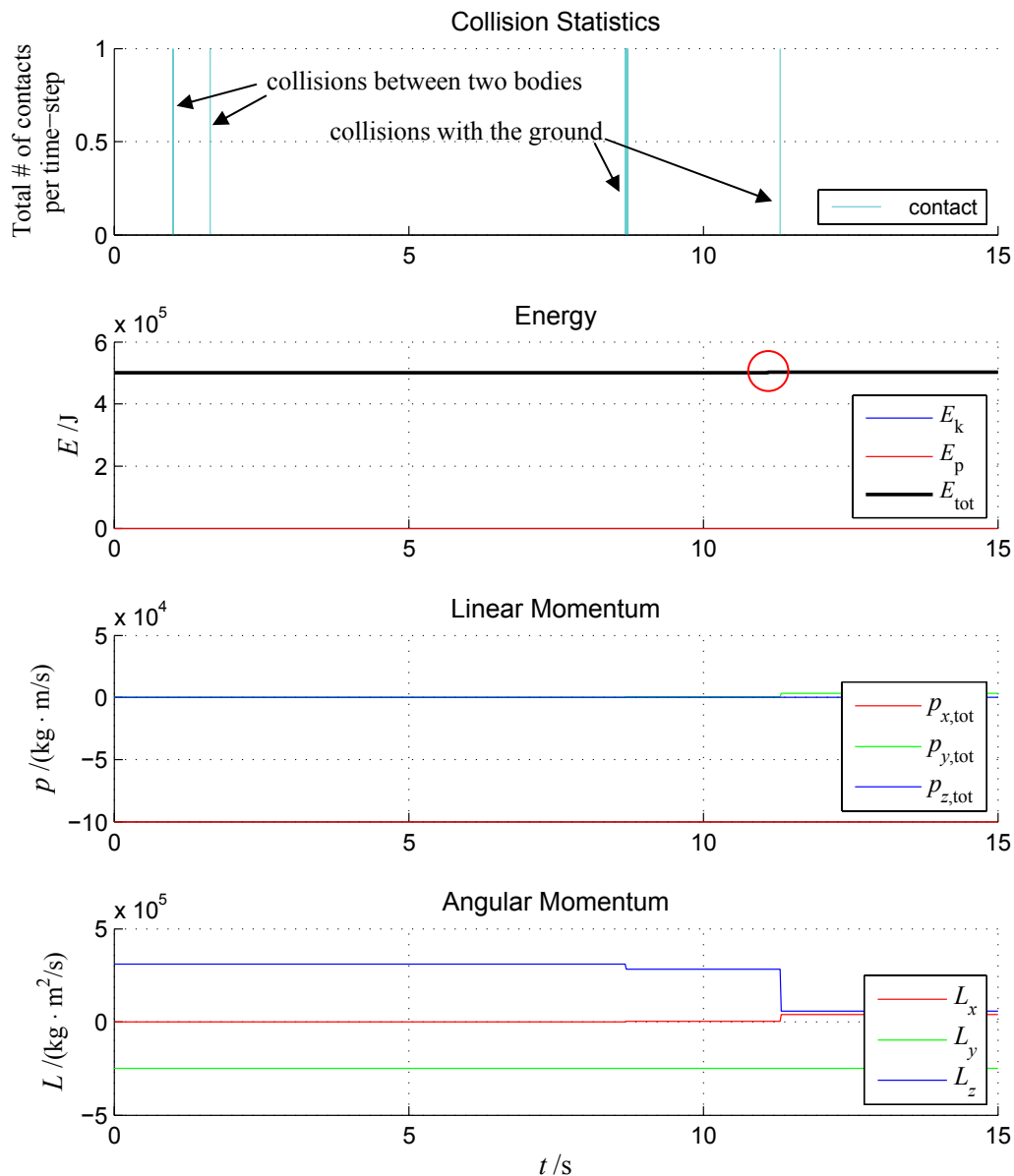
Figure 3.1–5:   The total energy, linear momentum and angular momentum of the system over
time during a more realistic collision (config-id '2d').

In this case it can be concluded that the total linear momentum and angular momentum are preserved during collisions between bodies, and not preserved during collisions between bodies and the ground plane. The kinetic energy is 'almost' preserved in all cases with the slight increase in kinetic energy (noticeable at $t \sim 11.5$ s) caused by the used semi-implicit Euler integrator which is known to increase angular kinetic energy of the system over time (thus requiring a certain kinetic energy dumping [4]).

The reason why linear and angular momenta are not preserved during collisions with stationary objects (like the ground) is, first, stationary objects are assumed to have *infinite* masses and second, stationary objects are not allowed to move. If stationary objects were allowed to move and if they masses were finite (but very large), the total momenta of the system would be preserved.

This is demonstrated on Figure 3.1–6, which shows the simulation results of the collision between an oncoming cuboid having mass 10^3 kg hitting a stationary cuboid with mass 10^9 kg. Here, it can be concluded from Figure 3.1–7, that the total linear momentum, angular momentum and kinetic energy of the system are all preserved; the collision also results in an unnoticeable (but finite) change in linear and angular velocities of the stationary body (see state variables on Figure 3.1–6).
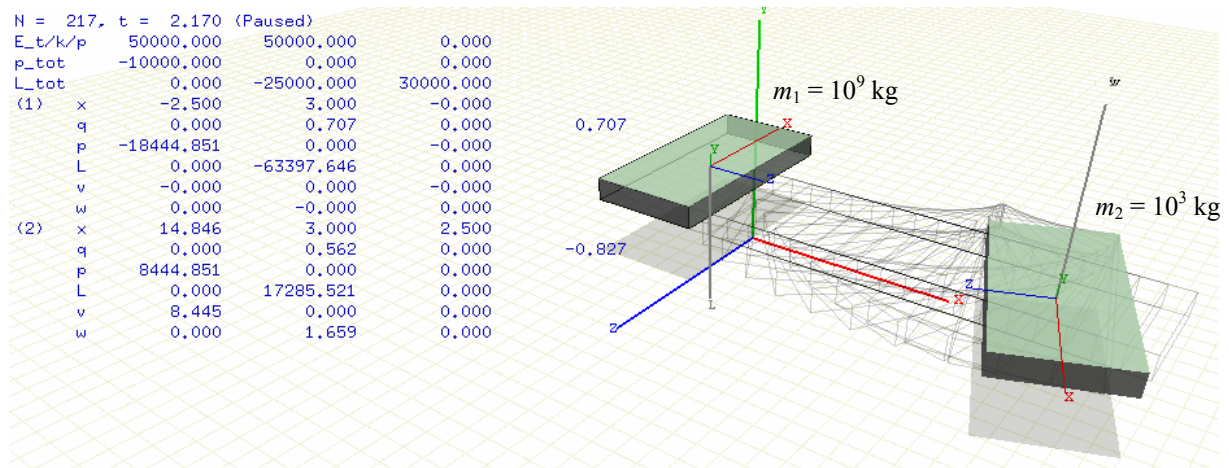
```
N =  217,  t =  2.170  (Paused)
E_t/k/p    50000.000    50000.000        0.000
p_tot     -10000.000        0.000        0.000
L_tot          0.000   -25000.000    30000.000
(1)    x       -2.500        3.000       -0.000
       q        0.000        0.707        0.000
       p   -18444.851        0.000       -0.000
       L        0.000   -63397.646        0.000
       v       -0.000        0.000       -0.000
       w        0.000       -0.000        0.000
(2)    x       14.846        3.000        2.500
       q        0.000        0.562        0.000
       p     8444.851        0.000        0.000
       L        0.000    17285.521        0.000
       v        8.445        0.000        0.000
       w        0.000        1.659        0.000
```

$m_1 = 10^9 \text{ kg}$

$m_2 = 10^3 \text{ kg}$

0.707

-0.827

Figure 3.1–6: The resulting trajectories of two colliding rigid bodies, where the stationary body has a finite but very large mass (config-id '2e'). The stationary body does not (noticeably) move while the oncoming rigid body bounces off.
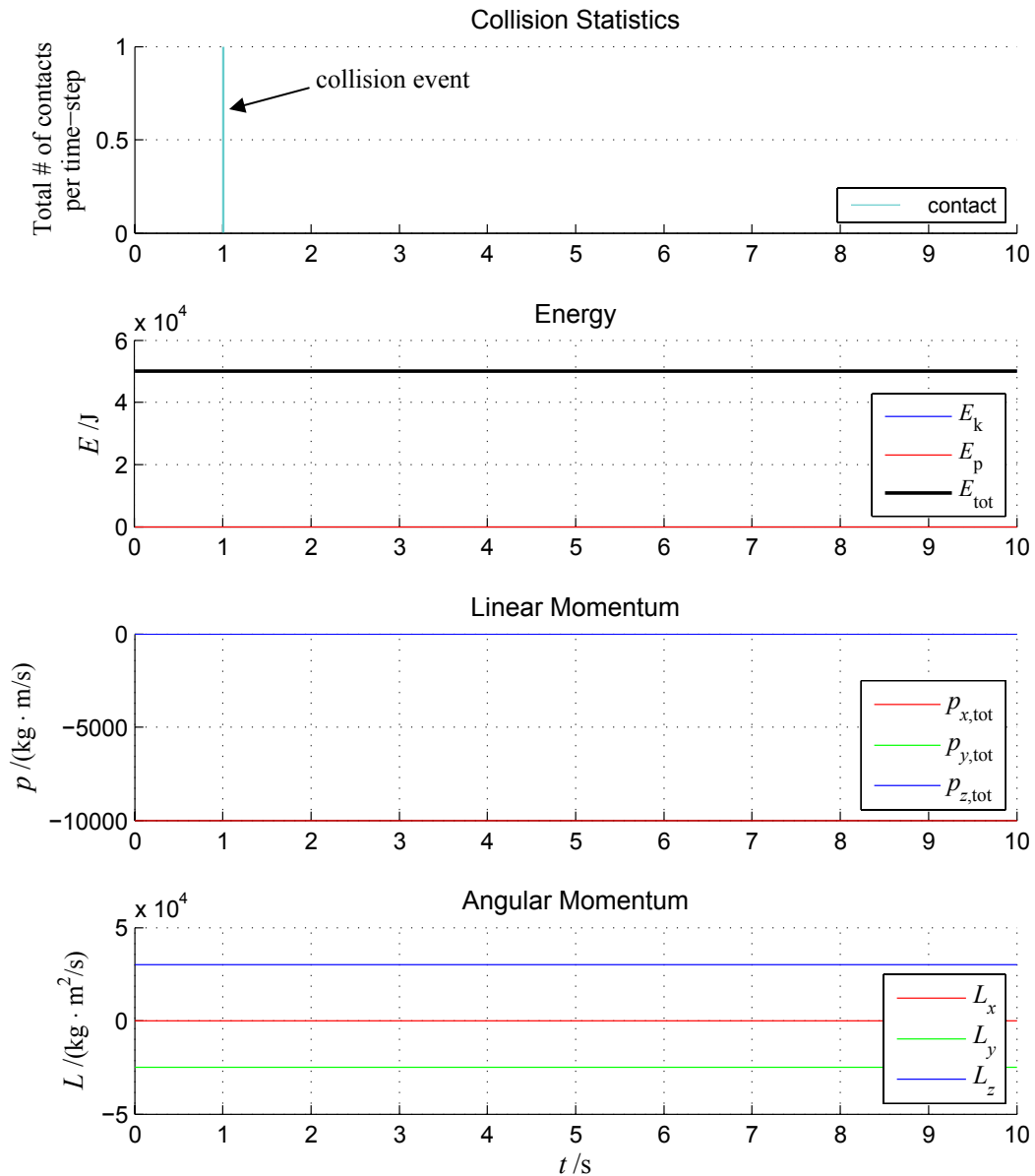
Figure 3.1–7: The total energy, linear momentum and angular momentum of the system over time, where the stationary body has a finite but very large mass (config-id '2e').

3.2 Advanced Level Tasks

Effects of Dissipation and Friction

Snapshots of different simulations that demonstrate effects of dissipation and kinetic friction are shown on Figure 3.2–2 (with dissipation, without friction), Figure 3.2–3 (without dissipation, with kinetic friction) and Figure 3.2–4 (with dissipation, with kinetic friction). The snapshot of the reference simulation is shown Figure 3.2–1 (without dissipation and friction). In all these simulations, two cuboid rigid bodies, having equal masses 10^4 kg, are colliding in the presence of the gravity field.

From Figure 3.2–1, it can be concluded that bodies are loosing the total energy and will eventually stop bouncing even with no dissipation and kinetic friction. The total energy decrease is caused by the semi-implicit Euler integrator in the presence of the gravity field. If the used integrator was rather the leap-frog integrator, the bodies would bounce forever (refer to figures 3.1-2 and 3.2-3 in the "Conservation of the Total Energy in the Impulse Collision Model" section in the earlier lab2 report).

From Figure 3.2–2, it can be concluded that, with dissipation but without kinetic friction, bodies on ground will glide ad infinitum.

From Figure 3.2–3, it can be concluded that, with friction but without dissipation during the collision with the ground, bodies will bounce for ever, until they lose energy from kinetic friction when occasionally colliding with the horizontal plane tangentially.
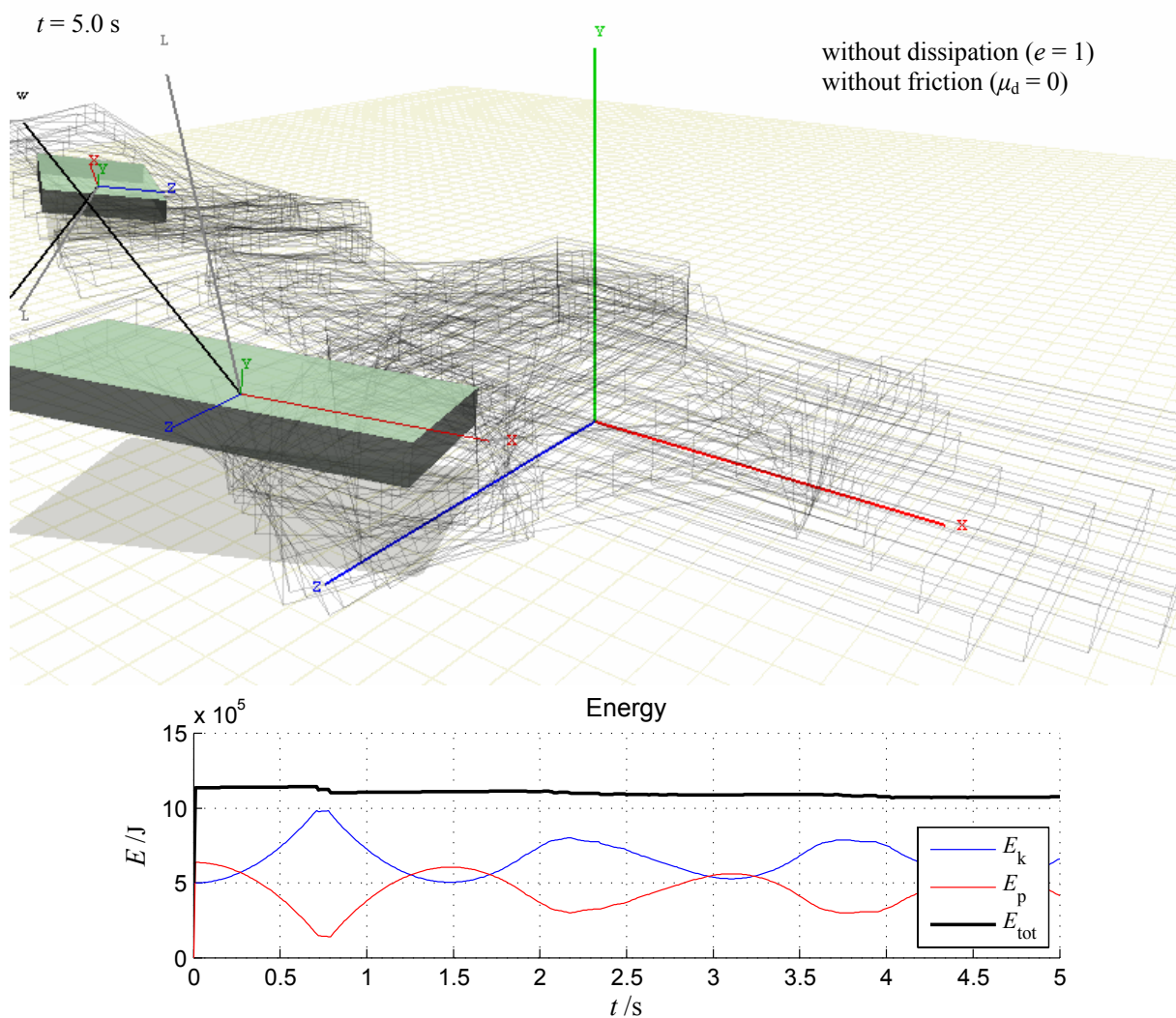

Figure 3.2–1: The snapshot of two colliding cuboids in the gravity field with the full restitution and without a friction (config-id '3a').

$t = 5.0$ s

with dissipation ($e = 0.6$)
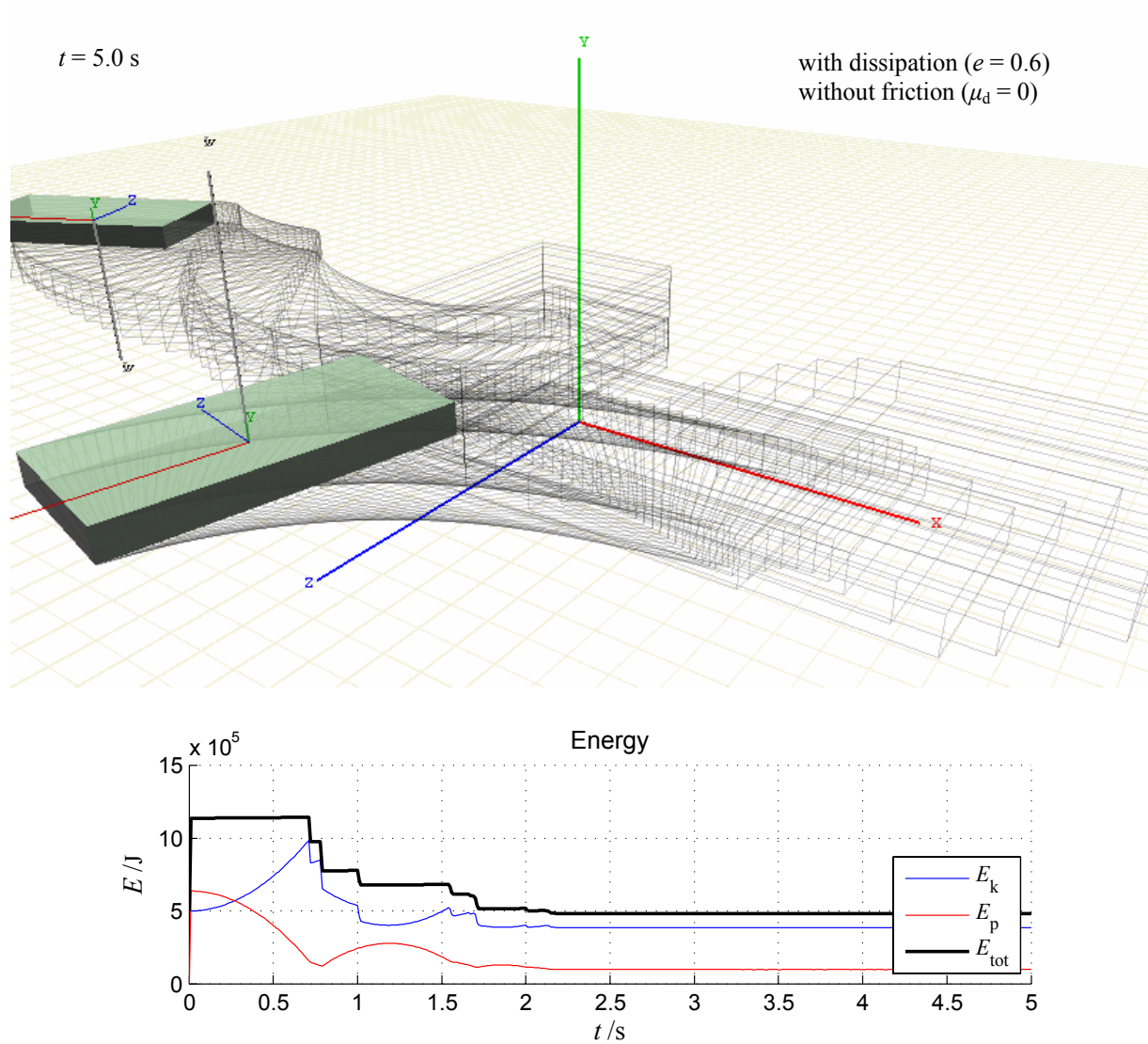without friction ($\mu_d = 0$)

Figure 3.2–2:   The snapshot of two colliding cuboids in gravity field with dissipation but without friction (config-id '3b'). Bodies will continue to glide for ever (blue line).
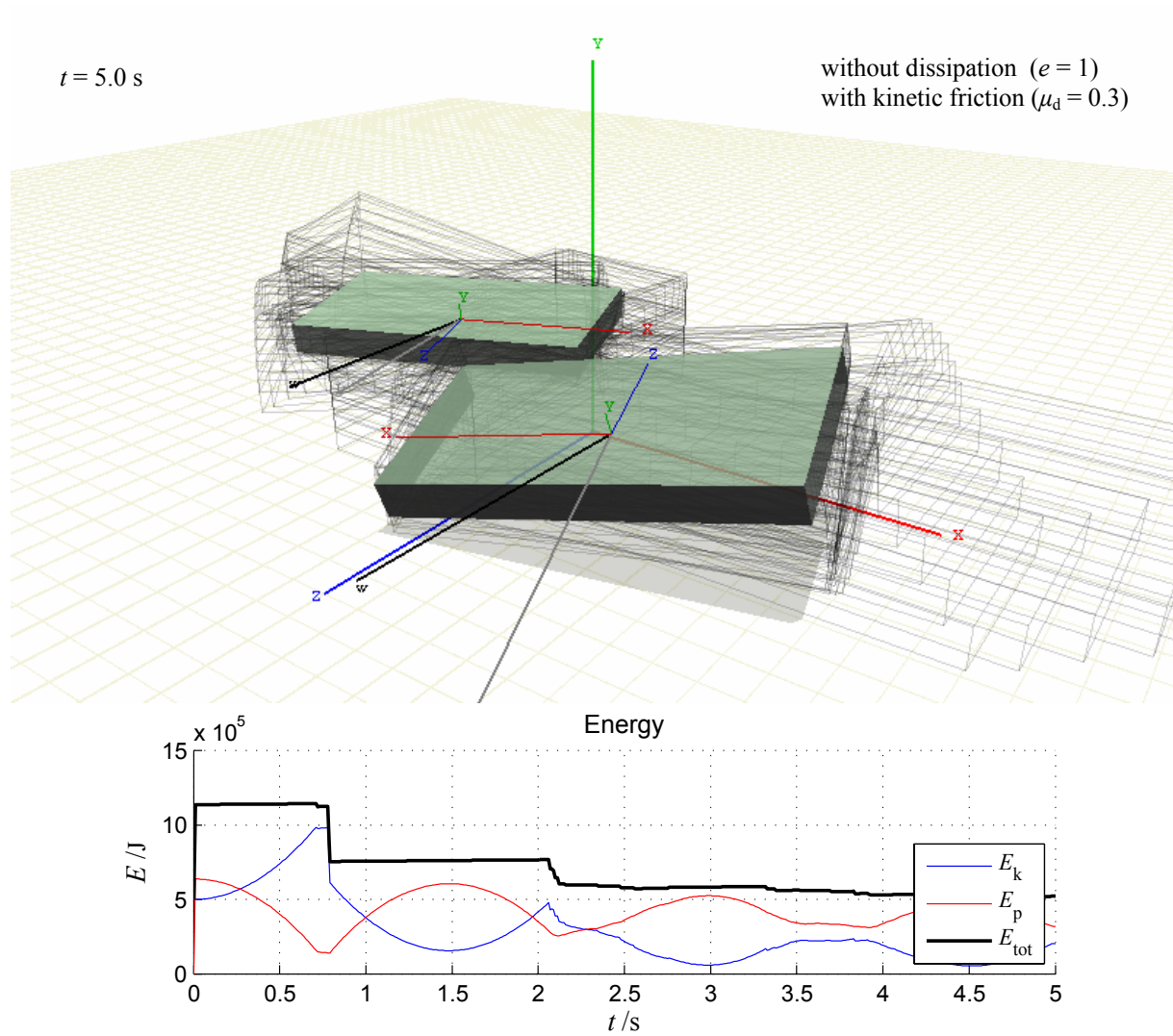
$t = 5.0$ s

without dissipation ($e = 1$)
with kinetic friction ($\mu_d = 0.3$)

Figure 3.2–3:   The snapshot of two colliding cuboids in gravity field without dissipation but with kinetic friction (config-id '3c'). Bodies will continue to bounce until they lose energy from kinetic friction during tangential collisions with the ground plane.

$t = 5.0$ s

with dissipation ($e = 0.6$)
with kinetic friction ($\mu_d = 0.3$)
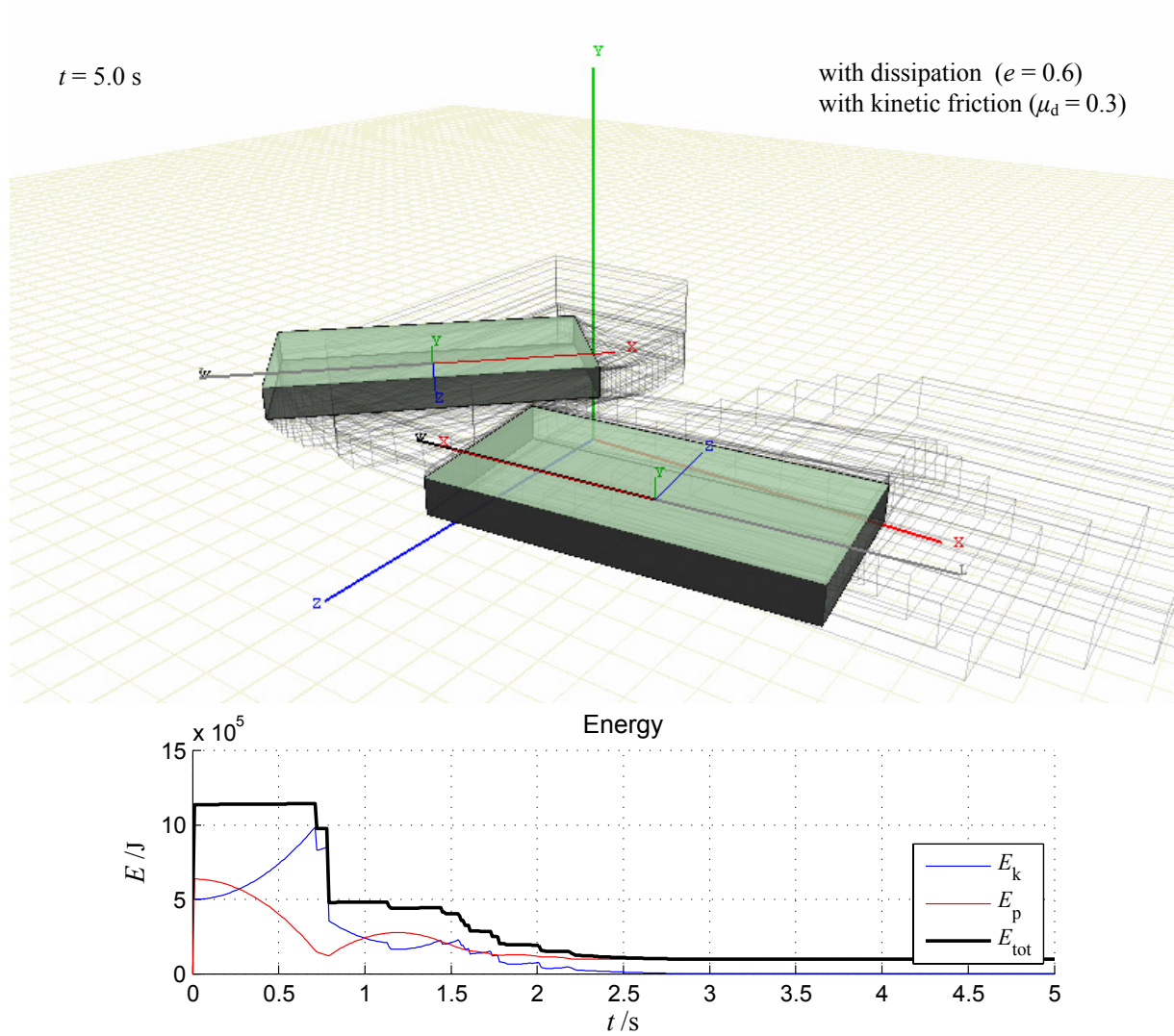
Figure 3.2–4:   The snapshot of two colliding cuboids in gravity field with both dissipation and kinetic friction (config-id '3d'). Bodies lose their kinetic energy after ~2.5 s.

Collisions with Static Ground Plane

The snapshot of the simulation of random cuboid rigid bodies, falling and colliding with the ground plane as well with each other, are shown on Figure 3.2–5.
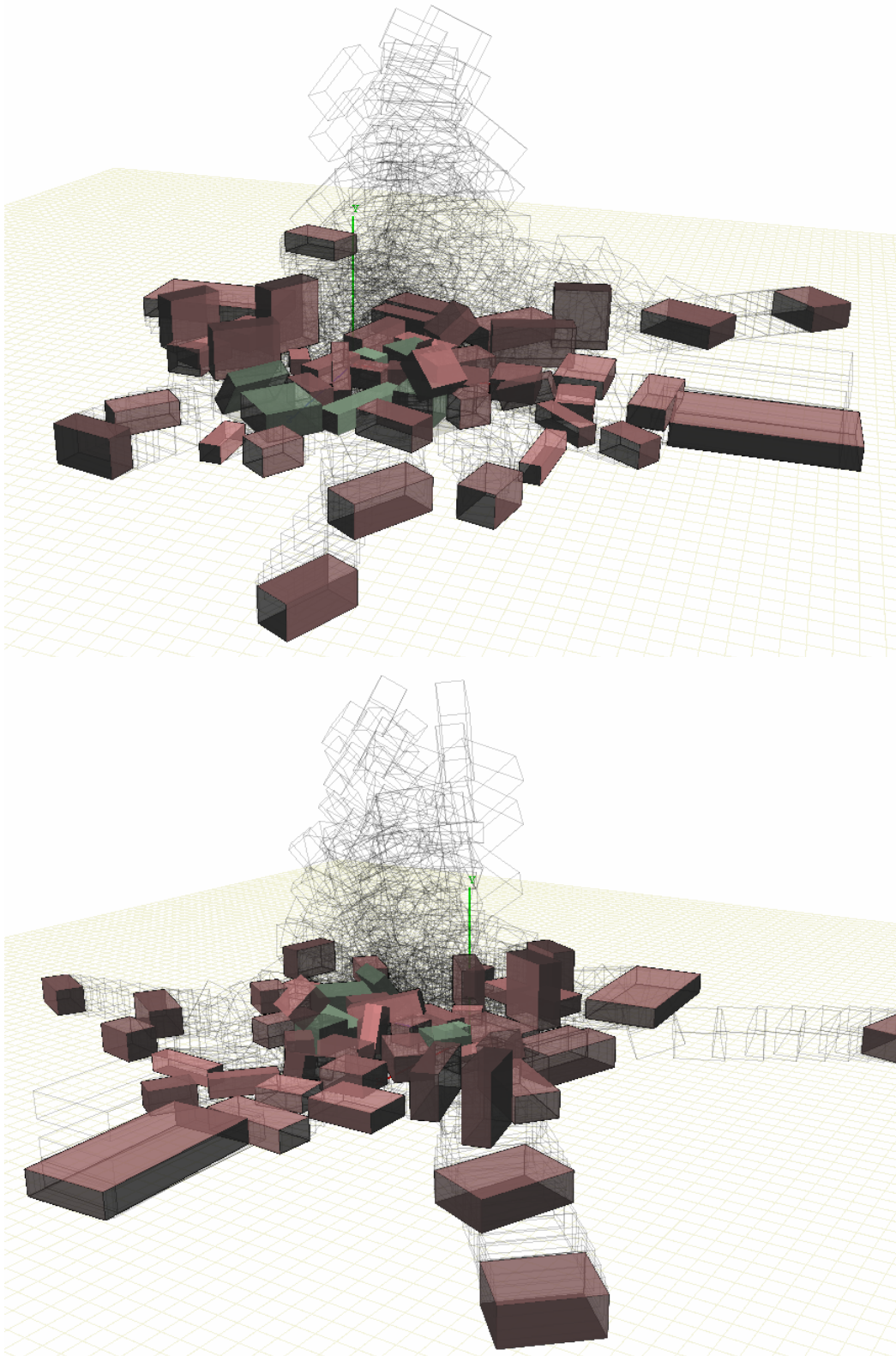

Figure 3.2–5:   Snapshots from two different angles of the same simulation where cuboids are falling onto the ground in the gravity field (config-id '4a'). Inactive objects (objects not-moving at the snapshot time) are highlighted in light-red.

Collisions with Staircase

The snapshots of the simulation of 50 random cuboid rigid bodies each having mass 10 kg, falling down the staircase of 10 m height, are shown on Figure 3.2–6.
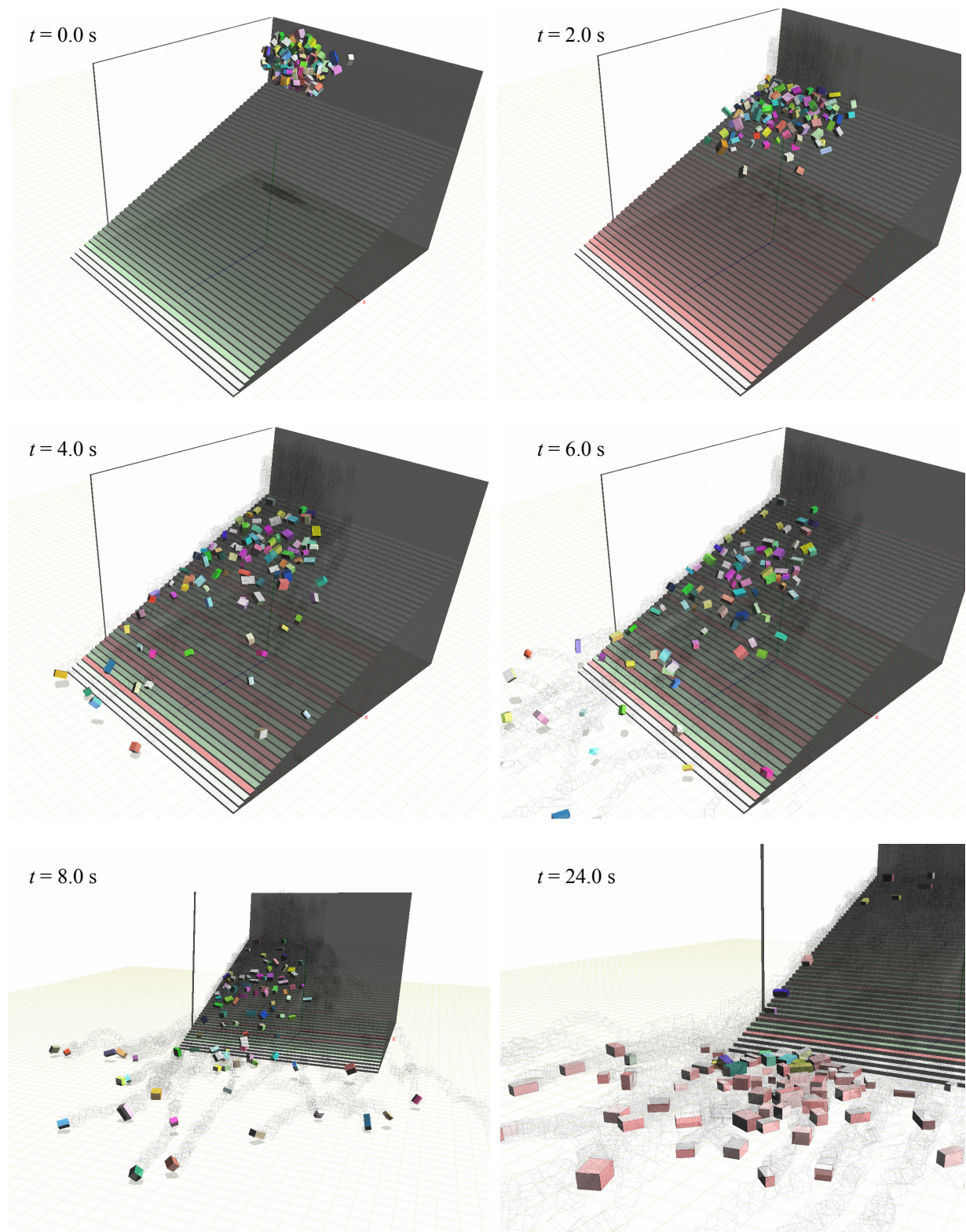

Figure 3.2–6:   Snapshots from different angles of the simulation where random cuboids are falling onto a staircase in the gravity field (config-id '4b'). Inactive objects (objects not-moving at the snapshot time) are highlighted in light-red.

Object Piling

The stability of the impulse method was studied by piling cuboid rigid bodies (blocks) in different configurations and by varying some of the reference simulation parameters. As the measure of method stability was taken stability of the vertical position of the center of mass of the top-most block (i.e. the pile height). The reference simulation parameters are given in Table 3.2-2 and particular simulation configuration IDs for different observed simulations are summarized in Table 3.2-2.

Table 3.2-1:   The reference simulation parameters

Number of blocks:	$N_b = 100$
Block mass:	$m = 100$ kg
Block dimensions:	$h = 0.4$ m height, $d = 2$ m width and depth
Coefficient of restitution:	$e = 0.8$
Position projections relaxation:	$r = 0.2$
Integrator time-step length:	$h = 0.01$ s

Table 3.2-2:   Observed piling blocks configurations

config-id	N_b	e	r	h / s	*Description*
5a	**100**	**0.80**	**0.2**	**0.010**	**Reference simulation**
5b	50	0.80	0.2	0.010	Varying number of blocks
5c	200	0.80	0.2	0.010	
5d	100	0.70	0.2	0.010	Varying restitution
5e	100	0.99	0.2	0.010	
5f	100	0.80	0.1	0.010	Varying relaxation
5g	100	0.80	0.2	0.010	
5h	100	0.80	0.2	0.020	Varying time-step
5i	100	0.80	0.2	0.001	

Snapshots of simulations for different number of blocks in a pile are shown on Figure 3.2–7.
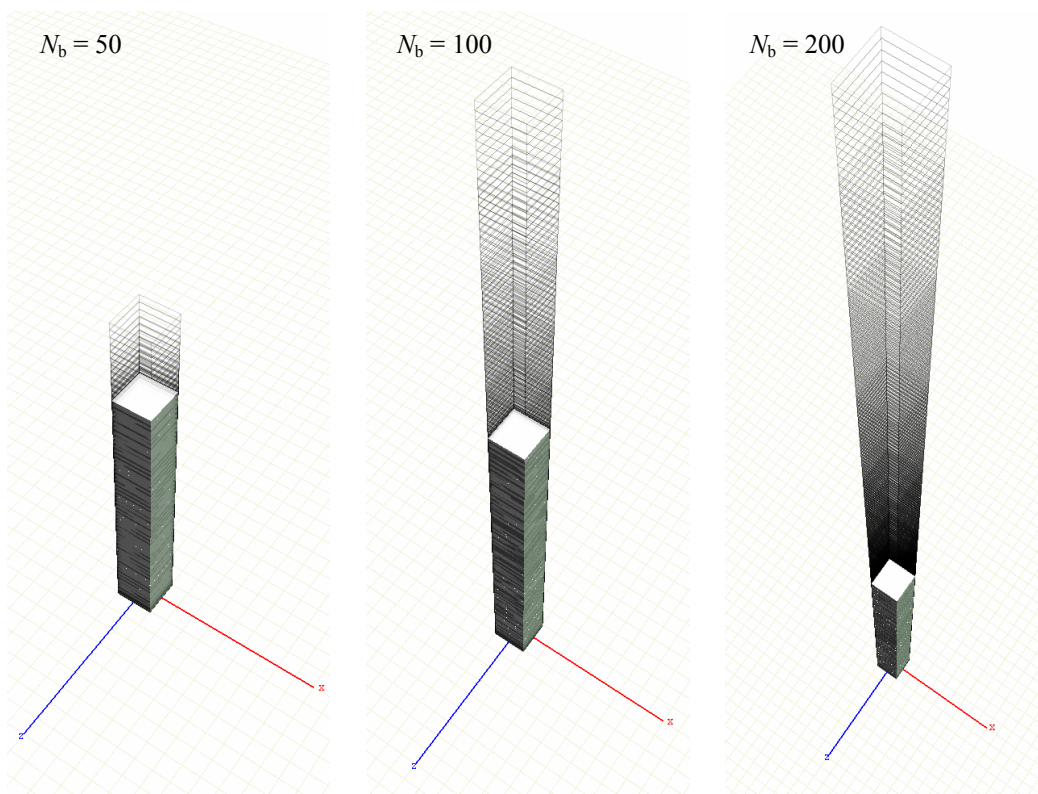

Figure 3.2–7:   Snapshots of piling blocks simulation for different number of blocks at $t = 10$ s (config-ids, from left to right: '5b', '5a' and '5c').

Pile height stability for different number of blocks is shown on Figure 3.2–8. It can be observed that when number of piled blocks reaches a certain amount, compression ratio of the pile stabilizes around a certain value. For low number of blocks in the pile, this ratio is minor. By increasing number of blocks in the pile, the compression ratio reaches its maximum which is dependent on the number of blocks in pile. This is summarized in the following table:

Table 3.2-3:   Relationship between pile height compression and number of objects in the pile

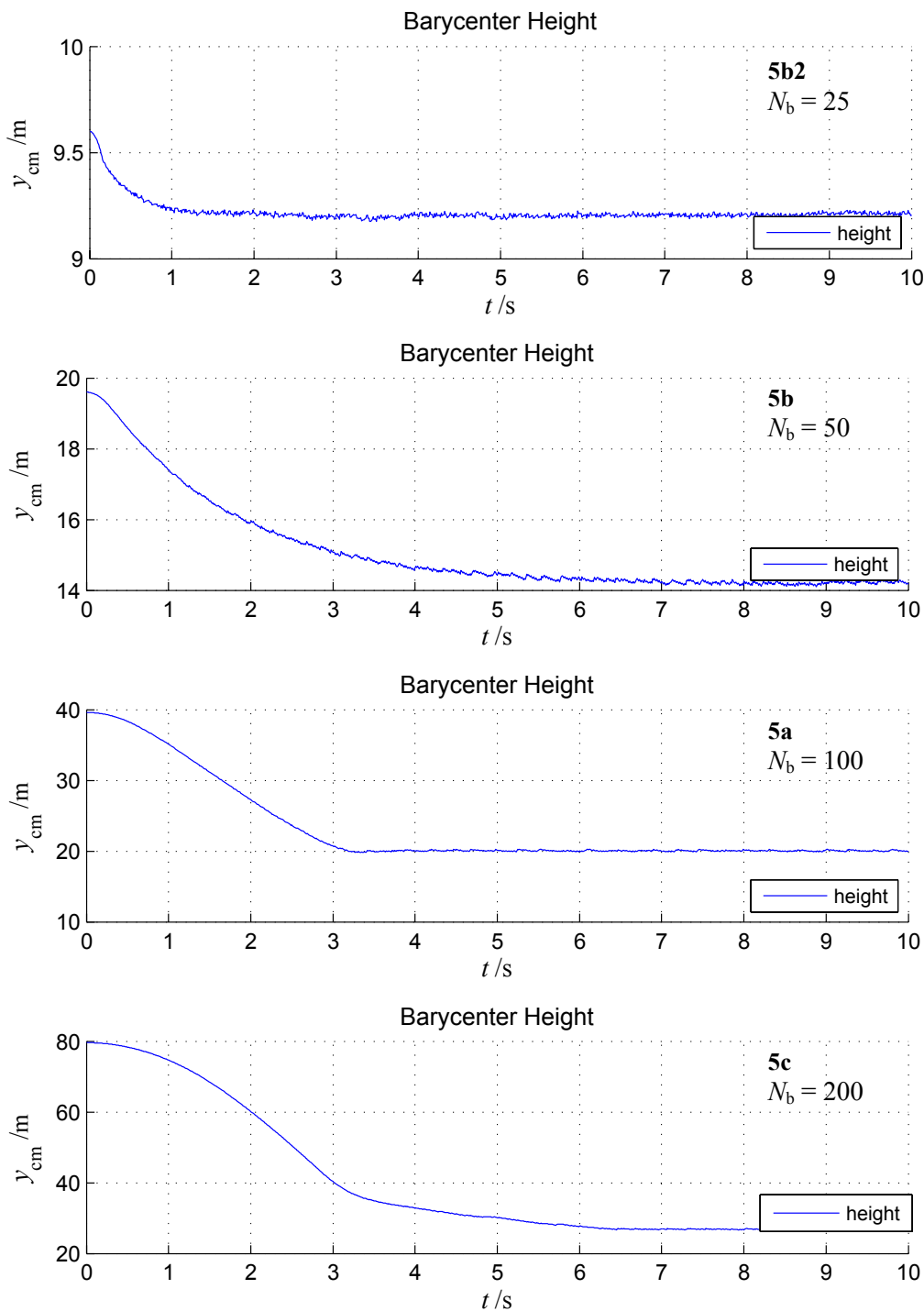| Number of objects	25	50	100	200
Pile height / %	97	80	50	32

Figure 3.2–8:   Pile height over time for different number of blocks
(config-ids: '5b2', '5b', '5a' and '5c').

Pile height stability for different coefficients of restitution is shown on Figure 3.2–9. It can be concluded that stability of the impulse method increases when coefficient of restitution decreases.
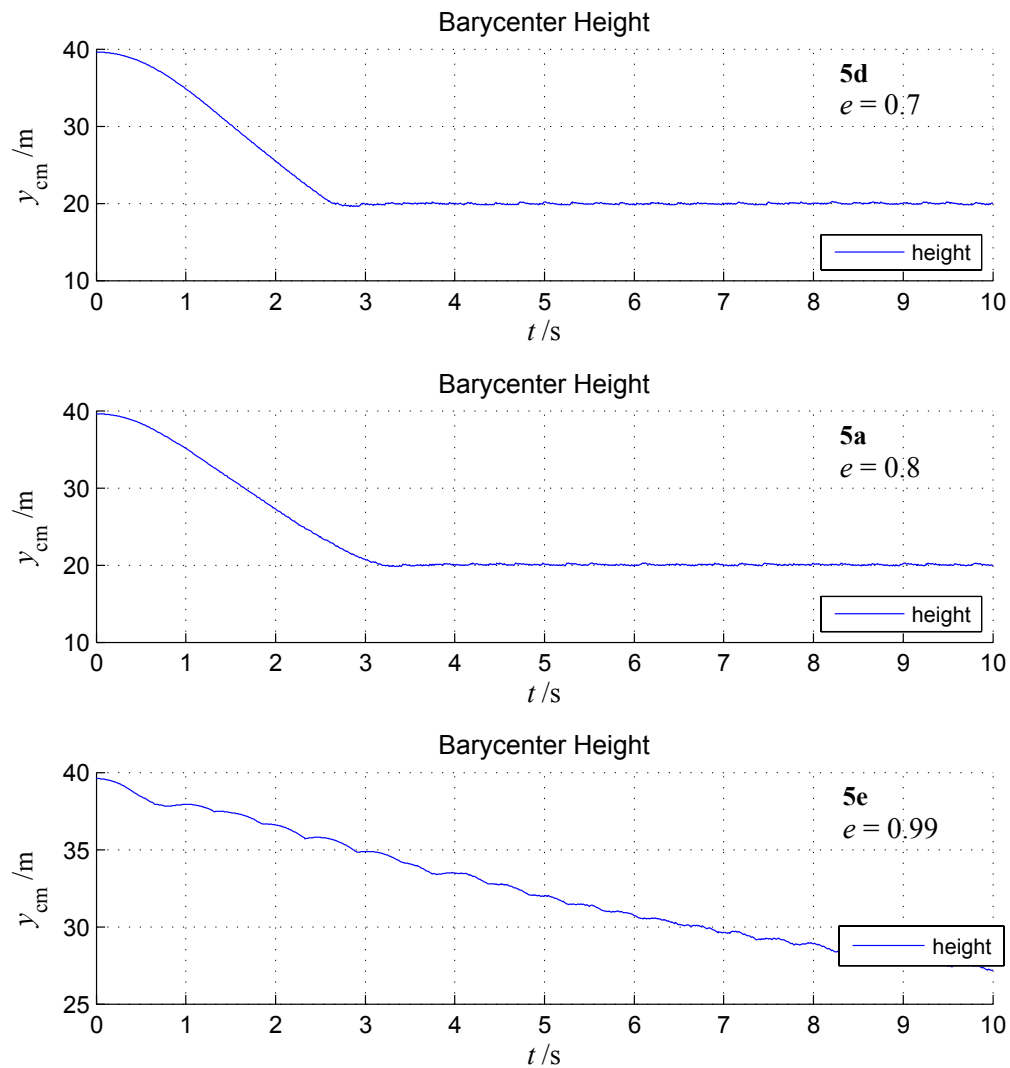

Figure 3.2–9: Pile height over time for different coefficient of restitution (config-ids: '5d', '5a' and '5e').

Pile height stability for different position projections relaxation coefficient is shown on Figure 3.2–10. It can be concluded that stability of the impulse method increases when position projections relaxation coefficient increases.

> The position projections relaxation coefficient $r = 0$ means that position projections are applied in full extent, while the coefficient $r = 1$ means that position projections are not applied at all. Note that implemented collision response algorithm requires $r < 1$.
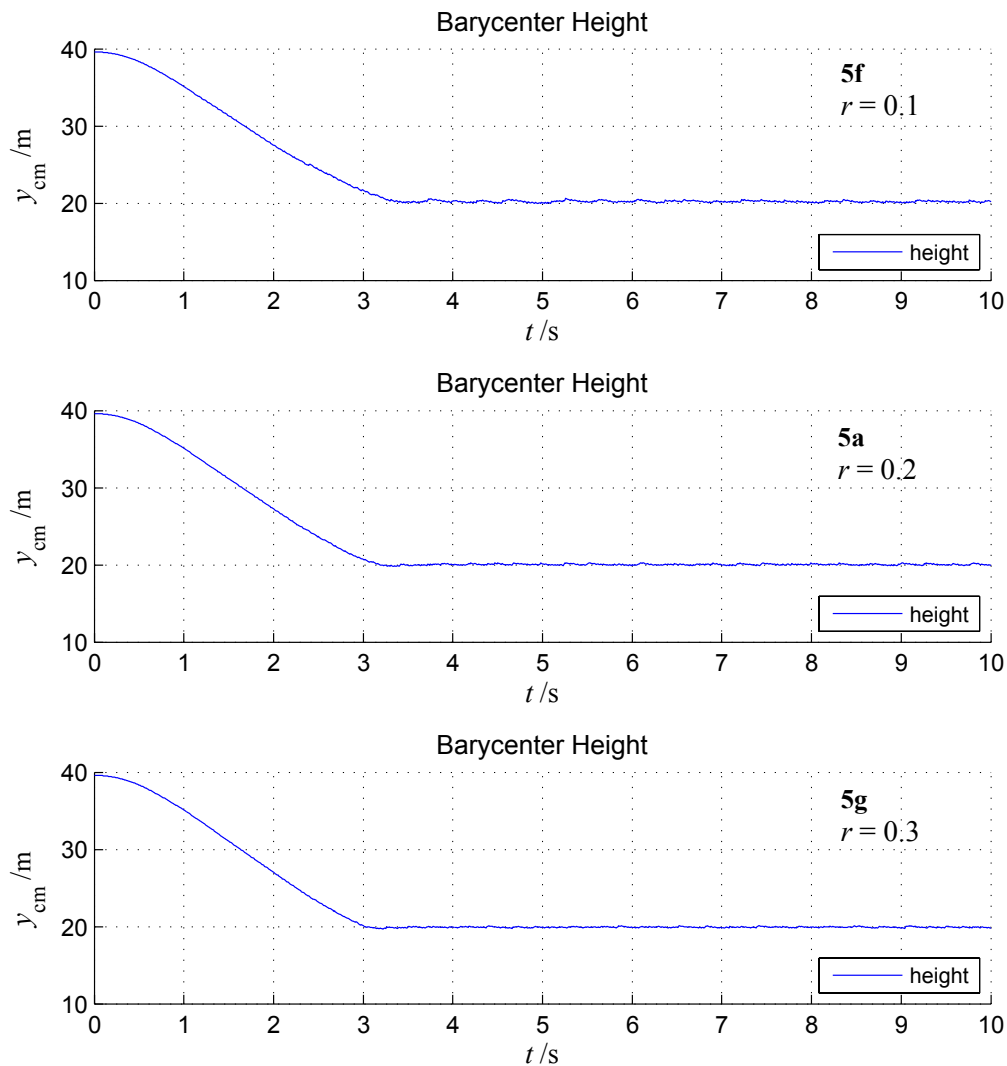

Figure 3.2–10: Pile height over time for different position projections relaxation coefficients (config-ids: '5f', '5a' and '5g').

Pile height stability for different integrator time-step lengths is shown on Figure 3.2–11. It can be concluded that stability of the impulse method increases when integration time-step decreases.
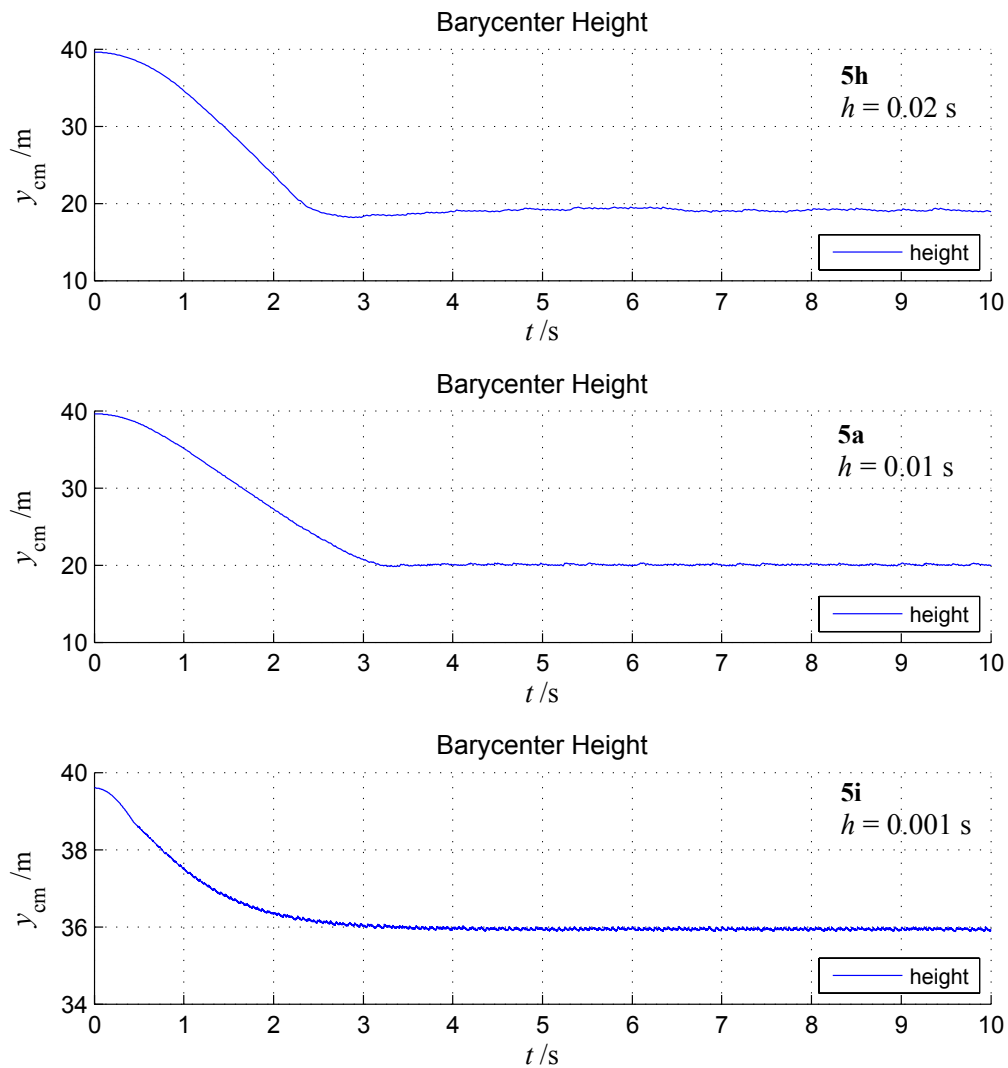

Figure 3.2–11: Pile height over time for different integrator time-step lengths
(config-ids: '5h', '5a' and '5i').

4 Extensions

One of the consequences of the selected approach to use quaternions to represent all physical quantities is that the WoRB framework could be extended to simulations in special relativity [6]. In such extension, position quaternion will include local time for example, and linear momentum and velocity quaternion will include total energy and Lorentz boost, respectively. Still, there are few obstacles to overcome in such endeavor; firstly, forces are not invariant in special relativity, meaning that Newton's third law and our rigid body equations of motion do not hold, so the interactions should be restated in terms of e.g. electromagnetic field, and secondly, quaternions should be allowed to have complex components (thus becoming biquaternions) which would require modifications to `Quaternion` and `QTensor` classes.

5 References

[1] Eberly, David H. – *Game Physics*, Morgan Kaufmann publ., 2004; Contrib. by Shoemake, Ken

[2] Eberly, David H. – *3D Game Engine Architecture*, Morgan Kaufmann publ., 2005

[3] Ericson, Christer – *Real-Time Collision Detection*, Morgan Kauffmann publ., 2005

[4] Millington, Ian – *Game Physics Engine Development*, Morgan Kaufmann publ., 2007

[5] Baraff, David – *Physically Based Modeling, Rigid Body Simulation*, SIGGRAPH, 1997

[6] Silberstein, Ludwik – *Quaternionic Form of Relativity*, Phil. Mag. 23:790-809, 1912