A Comparison of Linear Solver Methods in Object-Collision Simulations

Wen Zhang

Spring 2022 URA Report

Contents

1 Introduction

Modelling the collisions of many objects in an efficient and accurate way is a complex problem, with many applications in computer graphics. The problem becomes more complicated when considering simultaneous collisions of multiple solutions.

The purpose of this report is to study new methods that can solve multiple collisions more efficiently. This builds off the original research from (1), with the original simulation using the IPOPT solver, which can be found at: (https://github.com/breannansmith/scisim).

The research focuses on developing new linear solvers that treat the collision as a Linear Complimentarity Problem (LCP). In addition to previous work done from (2) to develop a Policy Iteration (PI) solver, a new Penalty Method solver was added. The full simulation codebase is located at (https://github.com/wzhang2705/scisim), on branch `penalty1`.

While both PI and penalty solvers are significantly more efficient than the original IPOPT solver, they do not always converge in the case of many objects in the simulation. This research remains open-ended with fixing these convergence issues as the next priority.

2 Code Setup

Working from the readme instructions in the codebase, and in addition to the instructions described in (2), there are some other code setup and running tips below.

2.1 Setup Tips:

- In addition to the instructions described in (2)
- The setup in this version of the code uses Ubuntu 22.04
- To use WSL on a Windows system, you would need to set up an X server to connect the graphics library properly
- Complete the steps in readme_ipopt.md before steps in readme.md

2.2 Running the Code:

Running cmake:

```
CMAKE_PREFIX_PATH=/usr/local/install/ipopt FC=gfortran cmake -DUSE_IPOPT=ON -DUSE_QT4=ON -
    DUSE_HDF5=ON ..
```

Where **/usr/local/install/ipopt** represents the location of your IPOPT library (also should be the value of **IPOPT_DIR**)

Run any test simulation by running the executable, and pass in the argument of the XML file - which defines the shape objects and type of collision solver to use.

Example:

```
./rigidbody2d_qt4 assets/tests_ipopt_serialization/balls_in_box.xml
```

To run the simulation: check "simulate". You can also click step to tick through each timestamp in the simulation.

2.3 Troubleshooting

Some potential problems you may encounter during setup:

Problem: Upon compilation, you may see this warning:

```
-- Found RapidXML: /home/wen/projects/scisim/include/rapidxml
-- Skipping Ball2D tests that require Python and HDF5 and h5diff (USE_PYTHON or USE_HDF5 is disabled or h5diff not found).
CMake Warning (dev) at /usr/share/cmake-3.22/Modules/FindOpenGL.cmake:315 (message):
  Policy CMP0072 is not set: FindOpenGL prefers GLVND by default when
  available.  Run "cmake --help-policy CMP0072" for policy details.  Use the
  cmake_policy command to set the policy and suppress this warning.

  FindOpenGL found both a legacy GL library:

    OPENGL_gl_LIBRARY: /usr/lib/x86_64-linux-gnu/libGL.so

  and GLVND libraries for OpenGL and GLX:

    OPENGL_opengl_LIBRARY: /usr/lib/x86_64-linux-gnu/libOpenGL.so
    OPENGL_glx_LIBRARY: /usr/lib/x86_64-linux-gnu/libGLX.so

  OpenGL_GL_PREFERENCE has not been set to "GLVND" or "LEGACY", so for
  compatibility with CMake 3.10 and below the legacy GL library will be used.
Call Stack (most recent call first):
  ball2dqt4/CMakeLists.txt:18 (find_package)
This warning is for project developers.  Use -Wno-dev to suppress it.
```

Solution: As described in https://cmake.org/cmake/help/latest/policy/CMP0072.html in CMakeLists.txt file, add the line:

```
cmake_policy(SET CMP0072 NEW)
```

Problem: There were some issues with the QT4 graphics not displaying well on the panel, and this was due to the X server permissions not being set.
Solution: https://stackoverflow.com/questions/35486181/x-error-baddrawable-invalid-pixmap-or-window-parameter-when-launching-spyder, then add

```
export QT\_X11\_NO\_MITSHM=1
```

in terminal setup.

3 Penalty Method

3.1 Linear Complimentary Problems

Starting with the LCP from (1): $0 \geq \lambda$ and $G_A^T q^+ \geq -c_r G_A^T q^-$

Using the equation $q^+ = q^- + M^{-1} G_A \lambda$, simplify the second inequality to:

$G_A^T q^+ = G_A^T [q^- + M^{-1} G_A \lambda]$
$G_A^T q^+ = G_A^T q^- + G_A^T M^{-1} G_A \lambda \geq -c_r G_A^T q^-$
$G_A^T q^- [1 + c_r] + [G_A^T M^{-1} G_A] \lambda \geq 0$

This inequality is simplified within the code, by setting: 1) $Q = [G_A^T M^{-1} G_A]$ and 2) $b = G_A^T q^- [1 + c_r]$.

3.2 Implementing Penalty Method

The implementation of penalty method follows the method similar to (3). This implementation works with LCP $x \geq 0$ and $Qx + b \geq 0$, as simplified from section 3.1. For penalty method to run, the initial x is set to the 0 vector and the initial penalty value is set at 10^8.

To run a simulation with penalty method, simply add the following line into the XML file:

```
<solver name="penalty" max_iters="100" tol="1.0e-6"/>
```

Where `max_iters` represents the maximum number of iterations the penalty method goes through before it concludes that it diverges, and `tol` represents the error tolerance for the final solution for x.

Furthermore, to run the simulation and record data from all 3 solvers simultaneously, add the following line to the XML file:

```
<solver name="solvers3" linear_solvers="ma97 ma57 mumps ma27 ma86" max_iters="100" tol="1.0
    e-6"/>
```

Where `linear_solvers` represents the linear solvers that the IPOPT simulation uses

4 Analysis

In both PI and penalty method, there remain issues with convergence in the case of large number of balls. These issues are further examined in the `balls_in_box` simulation, a similar setup as shown below:

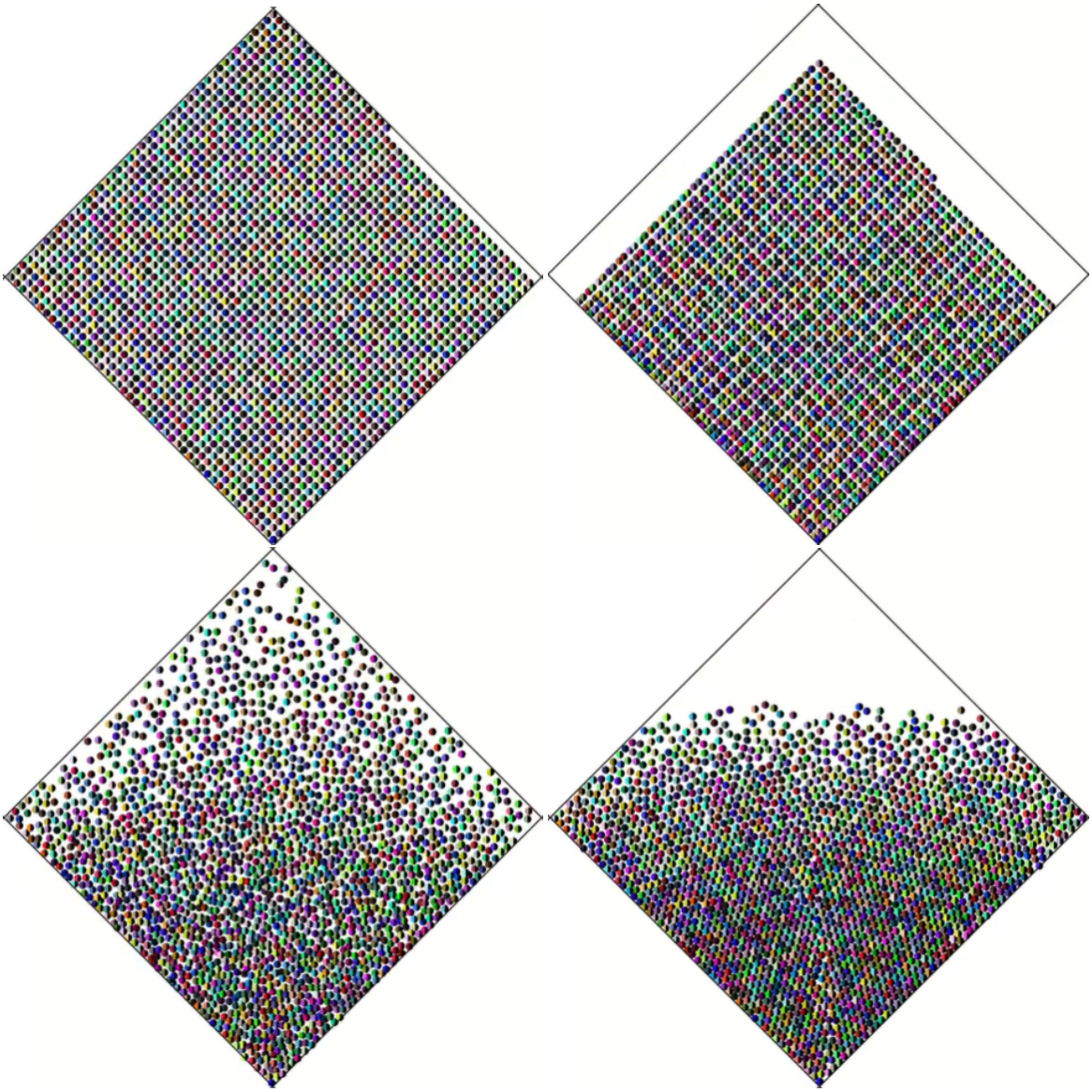

Figure 1: `balls_in_box` simulation over time - top left: start, 0 seconds top right: 10 seconds bottom left: 1 min bottom right: 6 mins

As shown in Figure 1, this type of simulation test begins with and arrangement of balls placed tightly around a square box. When the simulation begins, the balls start to fall down, and bounce back up and collide against one another. After about 5-6 minutes, the balls start to lose velocity and begin falling towards each other.

While both methods converge when the number of balls is under 2000, it appears to diverge for a greater number of balls, with known simulations of 5000, 10000, 16000, or above producing iterations of divergence in as little as a few seconds when the simulation begins.

First, one of the main correlations between divergence is the size of the matrix, in that cases of divergence are generally correlated with a larger matrix size.

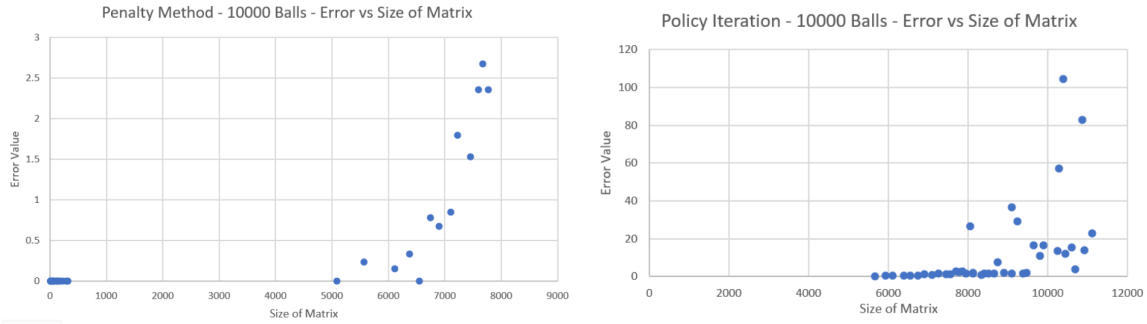

Figure 2: Errors values in divergence from penalty and PI

Figure 2 examines the cases of divergence from the 10000 `balls_in_box` simulation. The error value in this case is measured as the L2 norm between the final iterated value and the value that would solve the LCP, and the size of the matrix represents the number of columns (or rows) present in the $N \times N$ matrix of the LCP. In both penalty and PI, the error values of the solution tend to trend upward as the size of the matrix increases, and error values in PI (up to 105) tend to be larger than in penalty (up to 2.7). Generally, as the simulation progresses, the number of active balls in the collision increases, which in turn increases the size of the matrix. This progress is quite sudden in the simulation, which causes the large gap of values between smaller sized matrices and larger sized matrices in the graphs. While penalty and PI diverge in many larger sized matrices, they often also converge, so the relationship is not fully causal.

In the case of penalty method, there appear to be a series of matrices that still diverge despite not having as large a size. While the size of these matrices are still in the hundreds, they are much smaller than the majority of larger sized matrices that diverge. In PI, it appears that only larger matrices that have more than 5000 rows are diverging. Furthermore, it appears as though many of the error values are still quite small during divergence. While these error values (¡1) are small relative to the much larger error values in the graph, they are still much larger than the error tolerance value, which is generally set at 10^{-6} or 10^{-8}.

It is also important to note that the increase in error does not necessarily follow a nice trend. Among both simulations, there are outliers of the certain sizes producing very large error values. For example, in the penalty method graph, there was a case where the matrix size was 10068 with error 128251, and in the PI, there was a case where matrix size was 7885 with error 312780. In order to not distort the graph, these values were not included.

These issues can be further analyzed by looking at the type of matrix in the LCP. One of sufficient conditions for the LCP solver to converge is when the matrix in the LCP is a M-Matrix. To examine the effect of the matrix type on convergence, particularly comparing differences between penalty and PI, the divergence cases were examined for how close the matrix in the LCP was to an M-Matrix. This M-Matrix deviance is calculated by taking the max of the absolute values of all the diagonal entries that are negative (which should be positive), and the non-diagonal entries that are positive (which should be negative).

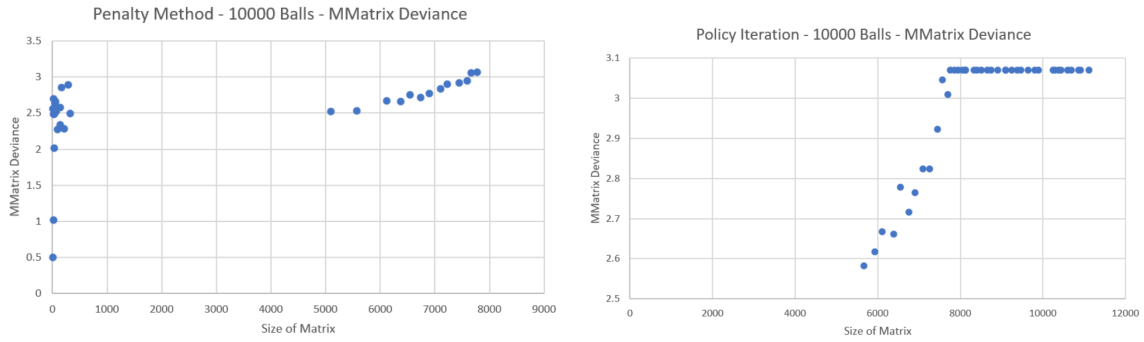

Figure 3: M-Matrix deviance values in divergence from penalty and PI

Comparing with the size of the matrix, the M-Matrix tends to increase with size, but in both cases, are capped at around 3.1. This is not significant in terms of convergence, but may indicate that the simulation pre-processes matrices by scaling them down first before passing them into the solvers.

One other characteristic of a square matrix is whether it is diagonally dominant. That is, whether the sum of the non-diagonal entries in a row is less than or equal to the diagonal entry. This was measured and recorded.

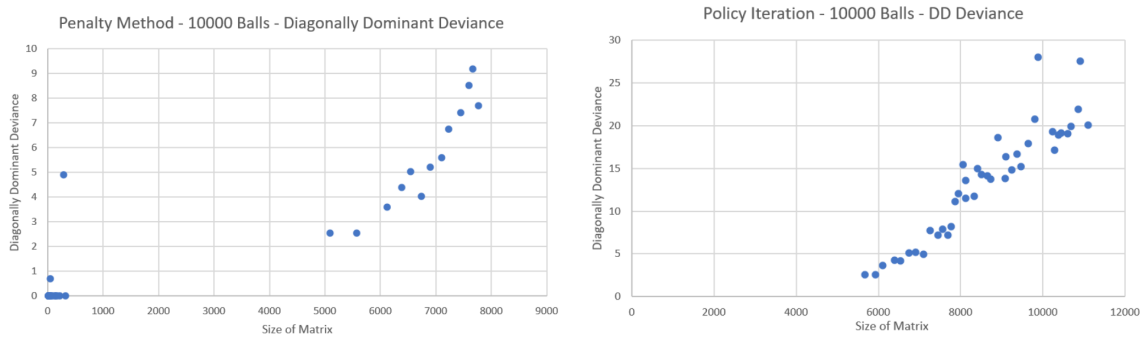

Figure 4: Diagonal dominance deviance values in divergence from penalty and PI

Overall, in most divergence cases, and as the matrix size increases, the matrix does deviate from an M-Matrix or diagonally dominant matrix. In the case of penalty method, there remain some cases when the matrix is small that still lead to divergence, and this in turn also has small diagonal dominance deviance. However, this was also observed in convergence cases, so is not a clear causal indicator of divergence. The scaling down of the values of the matrix indicates that the M-Matrix

deviance is still kept low as the matrix increases, so it is hard to conclude what the potential effect might be on divergence.

To better compare the solutions between the 3 solvers: IPOPT, penalty, and PI, all 3 solvers were run at once to compare the solutions they produced among one another. The l2 norm between the solutions was used as the difference.

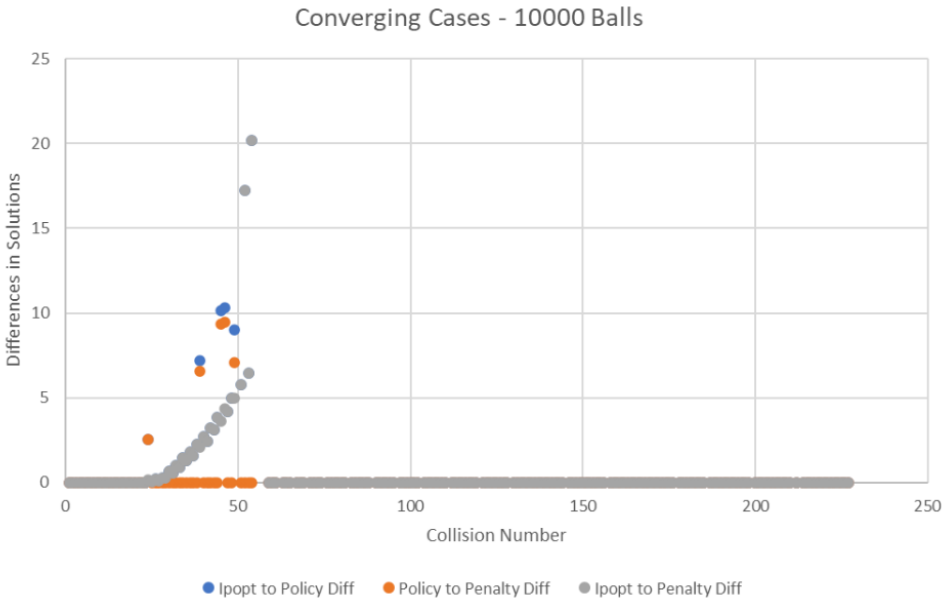

Figure 5: Solution differences in converging cases

First, as shown above, in converging cases, where all 3 solvers produced viable solutions to the LCP, most solutions were minimally different between the 3 solvers, which is expected because they should all be solutions to the LCP. However, there were often great differences between IPOPT and penalty or PI within the first 50 collisions, which suggests that these solvers look for a different solution in the solution space than IPOPT.

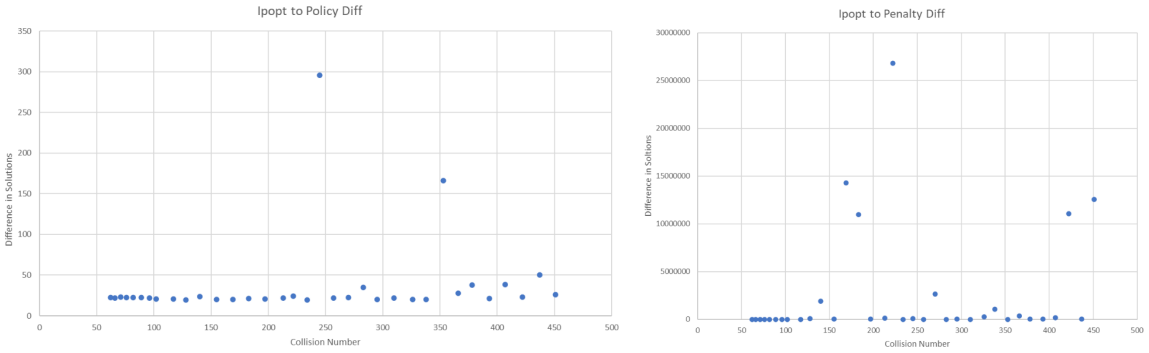

Figure 6: Solution differences in diverging cases

Furthermore, in looking at the divergence cases, the "solutions" examined were simply the last x values of PI and penalty that were set before the solver reached the maximum number of iterations. There were many instances where ipopt and policy solutions differed by around 25. However, IPOPT and penalty would sometimes differ by values in the millions. This indicates that in divergence cases, particularly with penalty, the iterations only further diverge away from the desired solution.

Another causal probability for divergence was in the overlapping of balls. During a collision, there is generally a delay between when the new position of a ball is calculated to when it is updated in the simulation, which is the discrete timestamp, dt. This may cause the balls to overlap as an updated position is out of sync in the simulation.

To combat this issue, the dt values were increased from the original dt=0.01, and the convergence and runtime was compared between dt=0.05 and dt=0.005. Overall, while the dt values varied by 10x, it did not appear to have a significant effect on convergence or runtime, and this is in part because when there are still 10000 balls in, the change in dt may not be enough to reduce overlapping.

All 3 solvers were run simultaneously in the case of 10000 balls to see the results of there corresponding runtimes:

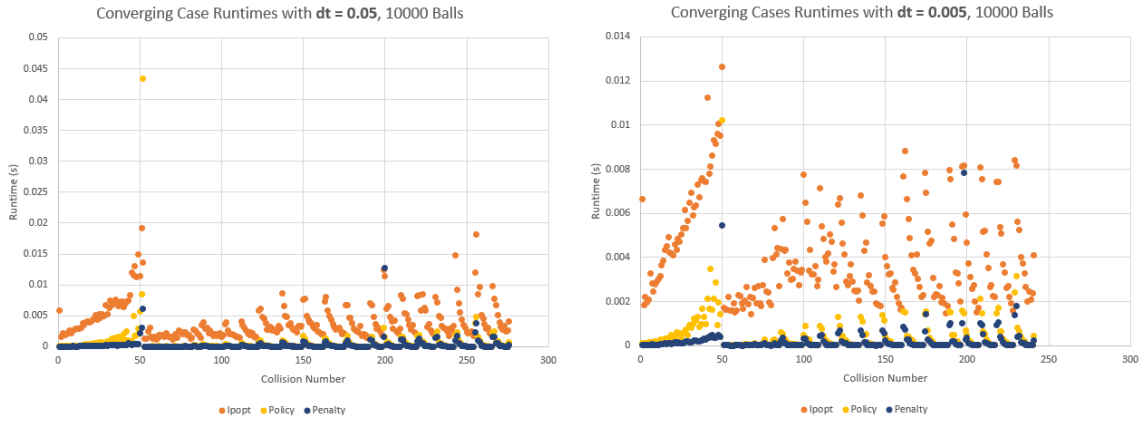

Figure 7: Comparing `dt` impact on runtimes for 10000 balls setup

While both PI and penalty still experience issues with convergence, they both perform significantly faster than the original IPOPT solver provided in the simulation during cases of convergence. For example, when dt=0.05, the IPOPT solver converges, it takes on average 0.004142 seconds, while PI took 0.000641 seconds and penalty took 0.000286 seconds to converge on average.

5 Conclusion and Next Steps

The significance in runtime reduction for both PI and penalty method for solving the cases indicates a large potential for these solvers to be more efficient that current solvers. However, convergence issues still remain a concern for both IPOPT and penalty method.

References

1. SMITH, B., KAUFMAN, D., VOUGA, E., TAMSTORF, R., AND GRINSPUN, E. 2012. Reflections on simultaneous impact. *ACM Transactions on Graphics (Proceedings of SIGGRAPH2012)*, 31(4):106:1–106:12

2. WAN, K. 2022. URA Report for Computer Simulation of Many Object Collisions.

3. LAI, J. 2020. Fast and Scalable Solvers for the Fluid Pressure Equations with Separating Solid Boundary Conditions.