Theme 2 Milestone 3: Krusty's Crazy Collisions

COMS W4167 Physically Based Computer Animation

Due: Wednesday, October 19, 2016, 21:59:59

Academic Honesty Policy

You are permitted and encouraged to discuss your work with other students. You may work out equations in writing on paper or a whiteboard. You are encouraged to use Piazza to converse with other students, the TA, and the instructor.

HOWEVER, you may NOT share source code or hardcopies of source code. Refrain from activities or the sharing of materials that could cause your source code to APPEAR TO BE similar to another student's enrolled in this or previous years. We will be monitoring source code for individuality. Cheating will be dealt with severely. Cheaters will be punished. Source code should be yours and yours only. Do not cheat.

1 Introduction

In Milestone 3, you will design your own algorithm for accelerating collision detection in scenes containing a large number of particles or edges. Test scenes for this milestone are located in a new directory, *assets/t2m3*.

2 Program Submission

Please submit your solution using our submission script on the CLIC machines at /home/cs4167/scripts/grade as before. Detailed instructions are available on Piazza.

3 New XML Features

This milestone adds the following new feature:

1. The new *collisiondetection* node specifies how collision detection is performed during the simulation:

   ```
   <collisiondetection type="allpairs"/>
   ```

   The valid values for *type* are "allpairs" (the default) and "contest". *allpairs* is already implemented in the starter code, and performs a brute-force check of all pairs of objects to see if any two of them are colliding. You will be implementing the contest algorithm in this milestone.

4 Required Features for Milestone 3

4.1 Collision Detection Contest

Edit *ContestDetector.cpp* and implement any algorithm you like for speeding up collision detection when handling collisions using the penalty method with thickness 0. In other words, given start-of-timestep positions, you are to return all particle–particle, particle–edge, and particle–halfplane pairs that might be overlapping at those positions. You do not need to perform continuous-time collision detection, and you do not need to check if particles are approaching, since the penalty method only cares about overlap at the start of the time step. The penalty force will then take your list of potentially overlapping pairs, compute a force for that pair, and apply it to the system. Note that the computed force will be zero if the pair is not actually overlapping.

One solution is to simply return all pairs of objects in the scene as potentially overlapping. This is the solution that is implemented as "allpairs" collision detection in the starter code. It is very slow – $O(n^2)$ – since a gradient has to be computed for all of these pairs, even those that aren't overlapping.

Another solution is to compute the distance between all pairs of objects, and only return those that are actually overlapping. No time is wasted computing unnecessary gradients, but the detection itself is now very slow – again $O(n^2)$. You want to design an algorithm in between these two extremes – one that can *quickly* determine all pairs that are *likely* to be overlapping, without including too many false positives.

We will run your alogrithm on a gauntlet of three very large test scenes, and measure the total time it takes your code to simulate the scenes. We will compare your algorithm's time against that of the other submissions, and your milestone grade will be determined by how well your algorithm performs. Here are the details of the competition.

- You may not sacrifice correctness for speed; in other words, your collision detection must be *conservative.* If two objects actually overlap during a time step, you *must* include that pair in your list of detected pairs. If they do not overlap, you *may* include that pair in the list. The oracle can be used to check that your code does not miss collisions.
- You may not change any files in the starter code except *ContestDetector.cpp* and *ContestDetector.h* (you may also create *new* source files containing helper code, if you'd like). We will overwrite all other files with those supplied in the starter code before compiling your contest entry. There are many inefficiencies in the starter code that could be optimized – but the focus of this milestone is solely on collision detection.
- You are encouraged to use the internet, class notes, research papers, etc. to find potentially useful ideas or algorithms. However, **all non-starter code you submit must be entirely your own.** You may not link to any external libraries except for those already used by the starter code (e.g. Eigen).
- The three test scenes will include
  - A scene very similar to the box-of-balls scene included with the starter code,
  - A scene very similar to the ribbon-pile scene included with the starter code,
  - A scene very similar to the best submitted creative scene; see section 6.

  You should strive to design a well-rounded algorithm that performs well for both provided scenes.
- For each of the three test scenes, we will run your code and record the time taken. If any of the following occurs, we will use the time taken by the oracle's implementation of "allpairs" instead:
  - your code fails the oracle due to unacceptably large residual (ie, you miss a collision),
  - your code fails to compile,
  - your code takes longer than the time taken by the oracle's implementation of "allpairs",
  - your code crashes or fails to complete the simulation for any other reason.

  Each scene is worth 28% of your milestone grade. For each scene, your grade is determined as follows: if T is your total time, A is the total time of the oracle running "allpairs", M is the fastest total time in the class, r the number of submissions slower than yours, and N the total number of submissions, your grade is given by the formula

$$14\left(\frac{A-T}{A-M}\right)^{1/3}+14\frac{r}{N-1}.$$

If you have multiple submissions, the shortest timings from yours in each scene will be used for grading.

5 Leader Board

Your code would be run with timing at some independent CLIC machines in about 20 minutes after your submission. The timing result will then be posted on a *Leader Board* indexed with submission IDs:

`http://www.cs.columbia.edu/~cs4167/index.php`

You may challenge the Champion of Year 2013, whose binary code can be accessed at:

`/home/cs4167/oracle/FOSSSimChampion2013`

and run in a similar way as the Oracle.

6 Creative Scene

As part of your final submission for this milestone, please include a scene of your design that best shows off your program. Based on the quality of your scene, you will have the opportunity to earn up to 15% extra credit. Your scene will be judged by a secret of committee of top scientists using the highly refined criteria of:

1. How well the scene shows off this milestones magic ingredients (a la Iron Chef).
2. Aesthetic considerations. The more beautiful, the better.
3. Originality.

Top examples will be posted to Piazza, and possibly demoed for the class. To submit this scene, place the XML file in the Creative directory of your submission. Please name your scene file youruni_tXmX.xml where youruni is your uni. Note that if you do not follow this requirement your scene may not be picked up by the grading script and may not receive any credit. We also ask that you include a movie of your creative scene in the Creative directory so that it can be posted online, should it be chosen by the judge committee. You can find instructions on how to generate movies from simulations on Piazza.

7 FAQ

Theme 2 Milestone 3 FAQ:

Q: I have touched other files in the code to make functions easier, is this allowed?

A: Unfortunately no. We will be taking solely the files for collision detection and adding them to our own codebase to ensure everyone is on equal footing.

Q: Is there any way to change my algorithm over time?

A: Sure, use static variables.

Q: If I am using AABB on edges should I use the edge radius or particle radius?

A: The edge radius. If the edge radius is larger you will correctly capture collisions. If it is smaller than the particle radius than particle-particle collisions will detect collisions before it reaches the edge.

Q: What exactly are we being asked to do?

A: You need to prune the $O(n^2)$ algorithm from checking every pair of objects using some higher level (broad) algorithm and pass down a reduced list of pairs (through ParticleParticleCallback, ParticleEdgeCallback, ParticleHalfplaneCallback) for last weeks collision detection routines to determine/handle collisions.

Q: My algorithm needs bounds on the position of scene elements, how do I obtain this.

A: The only sure way to do this is to look at all scene particles and determine max/min X/Y values.

Q: How do we test our algorithm and submit it?

A: Use the usual assignment submission system. It will appear on the contest website and inform you of its success.

Q: How can I verify the test scenes are passing correctly.

A: Just as before use the Oracle. Run the Oracle with your scene's output and get a visual display of every missed collision (red circles).

Q: I'm getting "disk quota exceeded" issues, what do these mean?

A: It is most likely an issue with the amount of memory you are limited to on your CLIC account. Try deleting or removing old files and see if that resolves the issue. If it is still present alert a TA and we will assist if it is something on our end.

Q: How long does it take from my submission for the results to go up.

A: It depends on the traffic on the CLIC machines. CLIC machines used by multiple people. So, the wall clock time may be considerably longer than on an isolated single-user machine. But, we don't measure wall clock time. We measure only processor time used by your process. So, the heavier the load (the more students are using the autograder), the more time it will take until your result is posted to the board.

Q: Should our code handle half-planes?

A: Yes.

Q: Should there be an acceptable trade-off between correctness and performance?

A: No. We cannot even start talking about the performance of an algorithm without first confirming its correctness, in other words, no matter how fast a collision detector runs, if it ever misses one legitimate collision, it is not acceptable.

8 Extra Reading

There have been a lot of research papers about collision detection and culling, you may implement one of them or design your algorithm inspired by them. Here are some examples:

Fuchang Liu, Takahiro Harada, Youngeun Lee, and Young J. Kim. 2010. *Real-time collision culling of a million bodies on graphics processing units.* ACM Trans. Graph. 29, 6, Article 154 (December 2010), 8 pages. `http://graphics.ewha.ac.kr/gSaP/`

Chang, Jung-Woo, Wenping Wang, and Myung-Soo Kim. *Efficient collision detection using a dual OBB-sphere bounding volume hierarchy.* Computer-Aided Design 42.1 (2010): 50-57. `http://www.sciencedirect.com/science/article/pii/S001044850900102X`

Fan, Wenshan, et al. *A hierarchical grid based framework for fast collision detection.* Computer Graphics Forum. Vol. 30. No. 5 (2011). `http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2011.02019.x/pdf`

Jae-Pil Heo, Joon-Kyung Seong, DukSu Kim, Miguel A. Otaduy, Jeong-Mo Hong, Min Tang, and Sung-Eui Yoon. 2010. *FASTCD: fracturing-aware stable collision detection.* In Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '10). `http://dl.acm.org/citation.cfm?id=1921450`

Larsson, Thomas, and Tomas Akenine-Mller. *A dynamic bounding volume hierarchy for generalized collision detection.* Computers & Graphics 30.3 (2006): 450-459. `http://www.sciencedirect.com/science/article/pii/S0097849306000689`