Kinetic Data Structure in Cell Movement Simulation

Kai Zhao
kzhao@gc.cuny.edu
Graduate Center, CUNY

NSF Expeditions in Computing

Problem Statement

The goal is to find an efficient data representation for simulating cell movement. The approach is to adopt kinetic data structures and tree structures from computer video games to speed up collision detection in simulations using the Bead-Spring model .

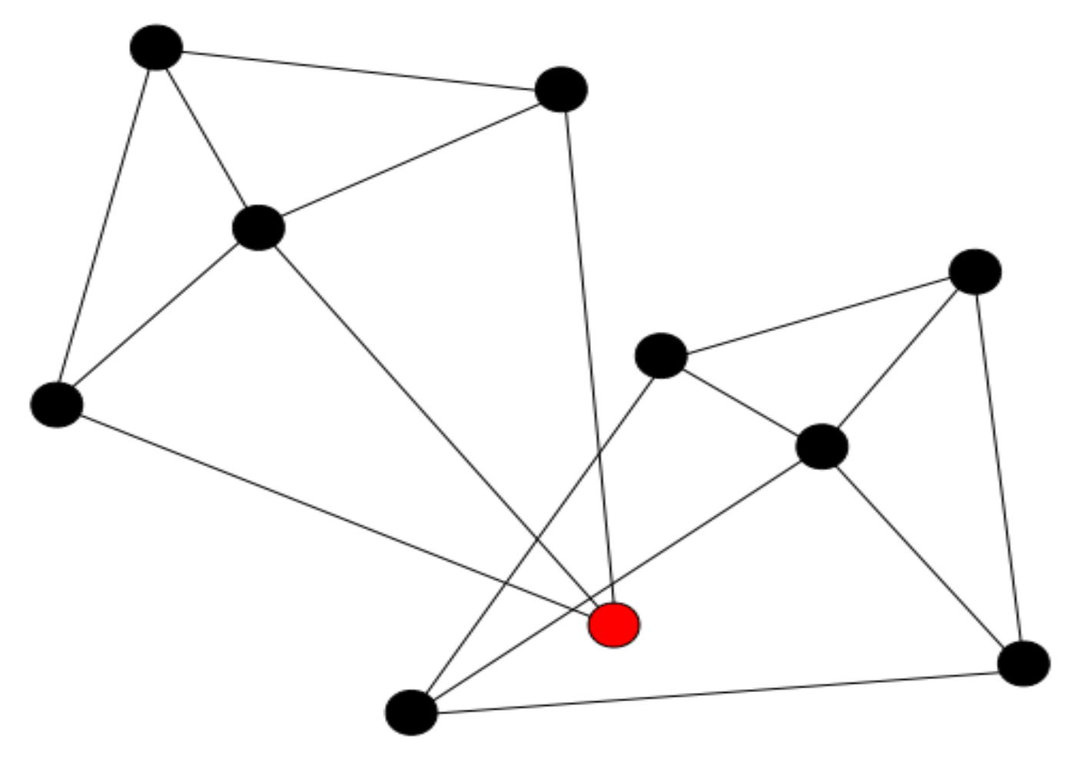
Background

To simulate cell movement, traditional lattice-based methods, such as *Cellular Potts Model* (CPM) based simulations, split the cell into *voxels* of the same size, and compute the movement of each voxel and its interactions with neighbors separately. The proposed Bead-Spring model views a cell like a Tensegrity structure, which can model the cytoskeleton inside a cell. The Bead-Spring model describes a cell as a set of beads connected by a set of springs. The springs mimic the filaments of the cytoskeleton system, and the beads are the connecting points. Each simulation step involves one bead. Compared to the lattice-based model, in the Bead-Spring model, there are fewer simulation objects for each cell, and both short range and long range interactions can be handled, although a more complex data structure means more overhead.

Collision

The major challenge for the Bead-Spring model is to determine whether two cells have physical interactions. As the cells scatter randomly in the space, cell protrusion and rotation may lead to a collision or penetration of one cell by another. The problem is to detect such an event.

This problem has been addressed in the field of computer graphics, specifically in work on representing motion of solid objects. The problem can be broken down into two problems: 1) Given a triangle, how to determine a small set of triangles that might collide with it; 2) Given the motion in a collection of objects, how to determine when the data structure representing them needs to be changed.



Red bead collides with other triangles.

The answer to the first problem is to use of appropriate tree structures to organize triangles in 3D space to locate the neighboring triangles as fast as possible. The answer to the second problem is to use recent ideas from proposals for kinetic data structures to maintain their validity when the objects in them move.

References

[1] P. Agarwal, J. Erickson, and L. Guibas. Kinetic binary space partitions for intersecting segments and disjoint triangles. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 107–116. Society for Industrial and Applied Mathematics, 1998.

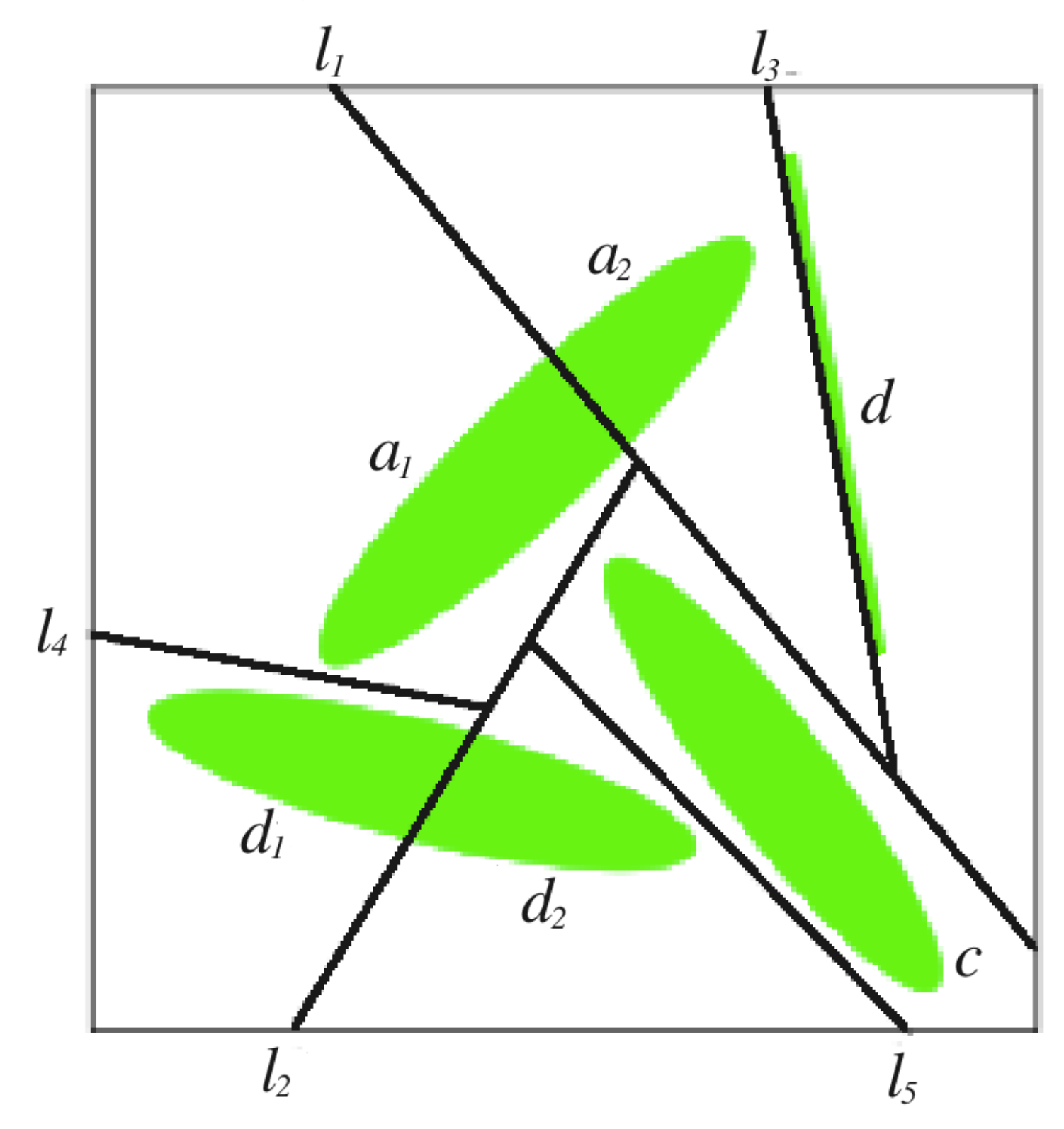
[2] D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. A computational framework for incremental motion. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 200–209. ACM, 2004.

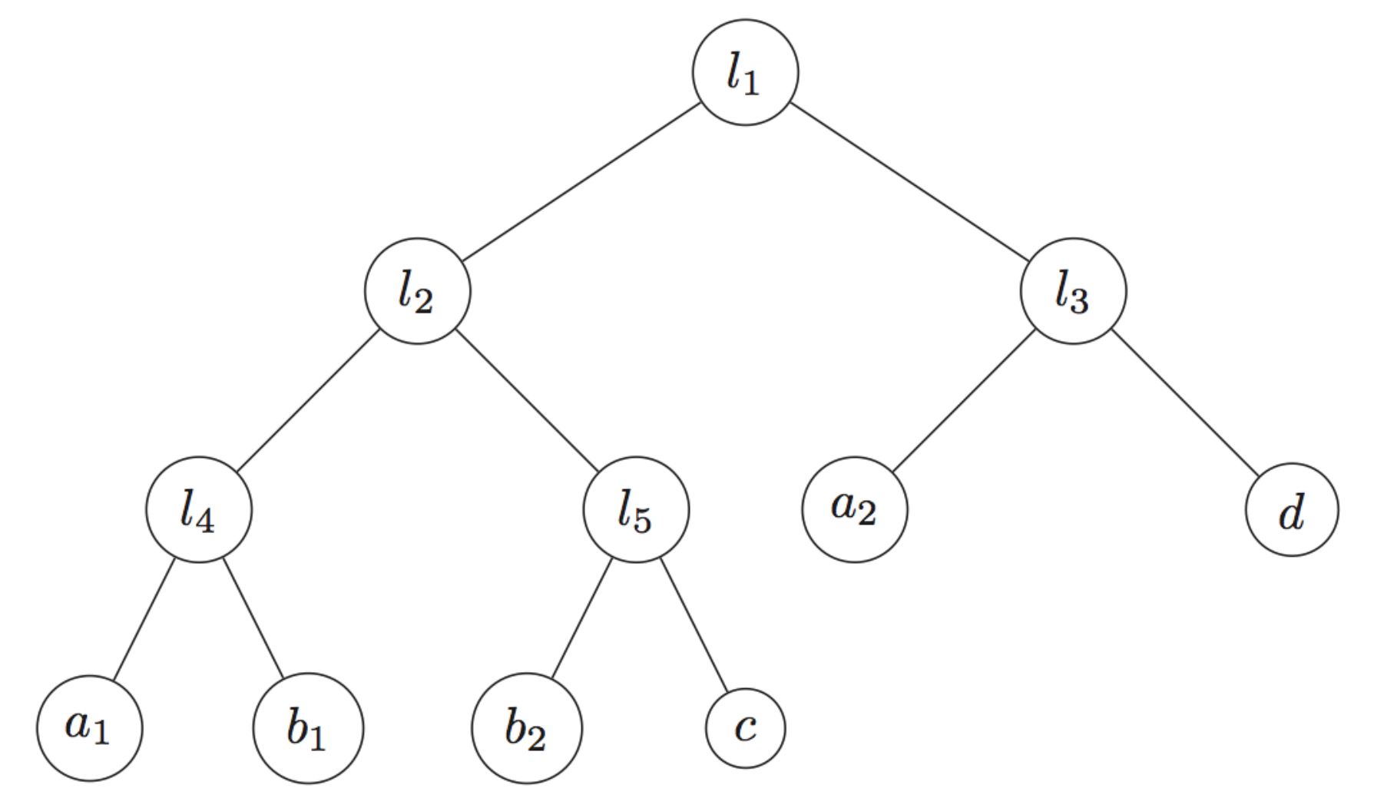
[3] A. Yershova and S. M. LaValle. Improving Motion-Planning Algorithms by Efficient Nearest-Neighbor Searching. *IEEE Transactions on Robotics*, 23(1):151–157, Feb. 2007.

Tree structures for representing moving objects

There are several tree data structures for modeling moving objects, including BSP tree, K-D tree, and R-tree. Here we discuss BSP tree and K-D tree.

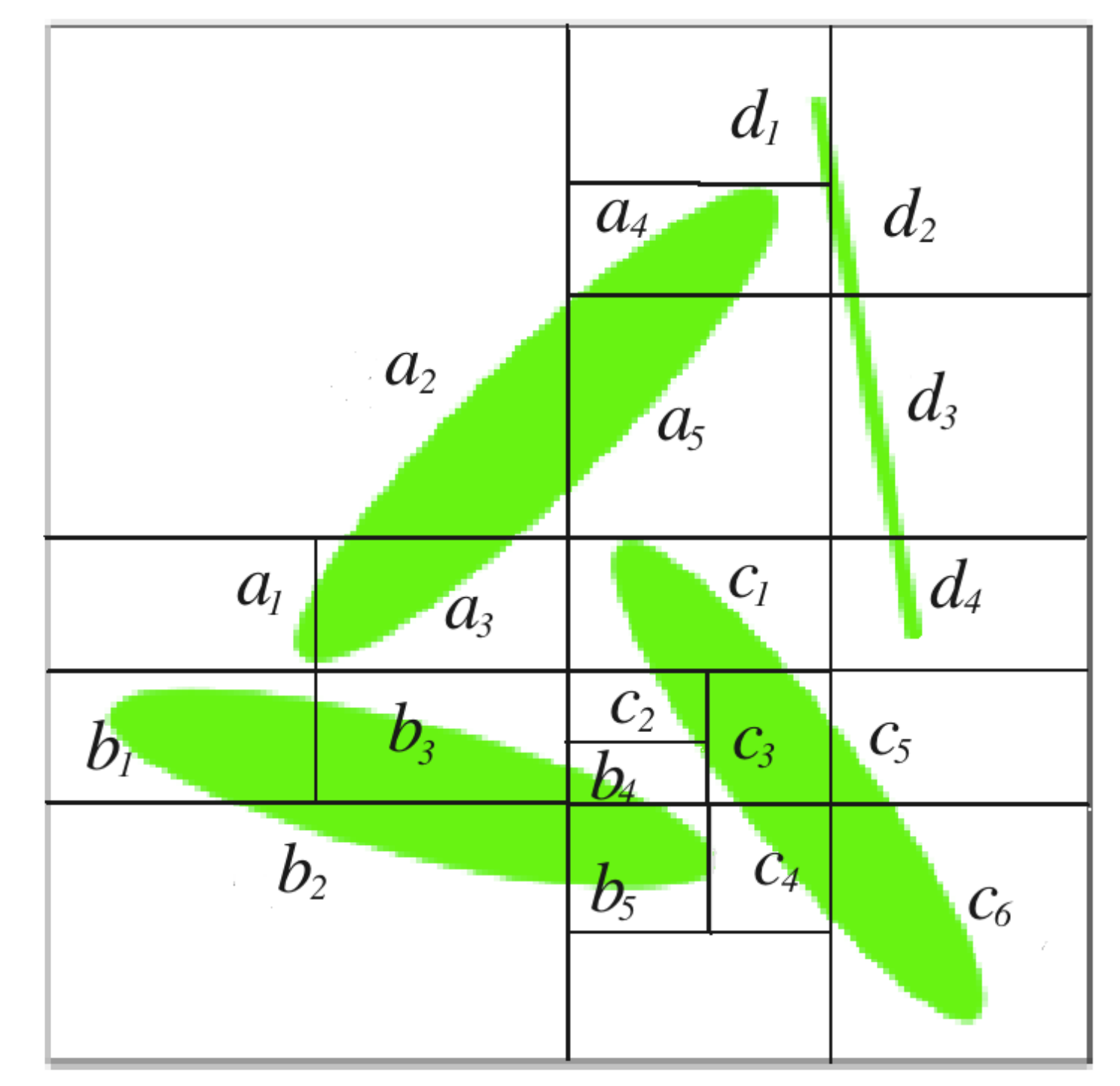
Binary Space Partitioning (BSP)[1] splits a scene into two recursively, until the objects in a partition can be handled individually. In our case the stopping criterion is that the number of triangles in the leaf partitions are small enough to be tested by enumeration. In 3D space at each step a plane is chosen to split a 3D object's surfaces. The optimization goal is to make the tree balanced.

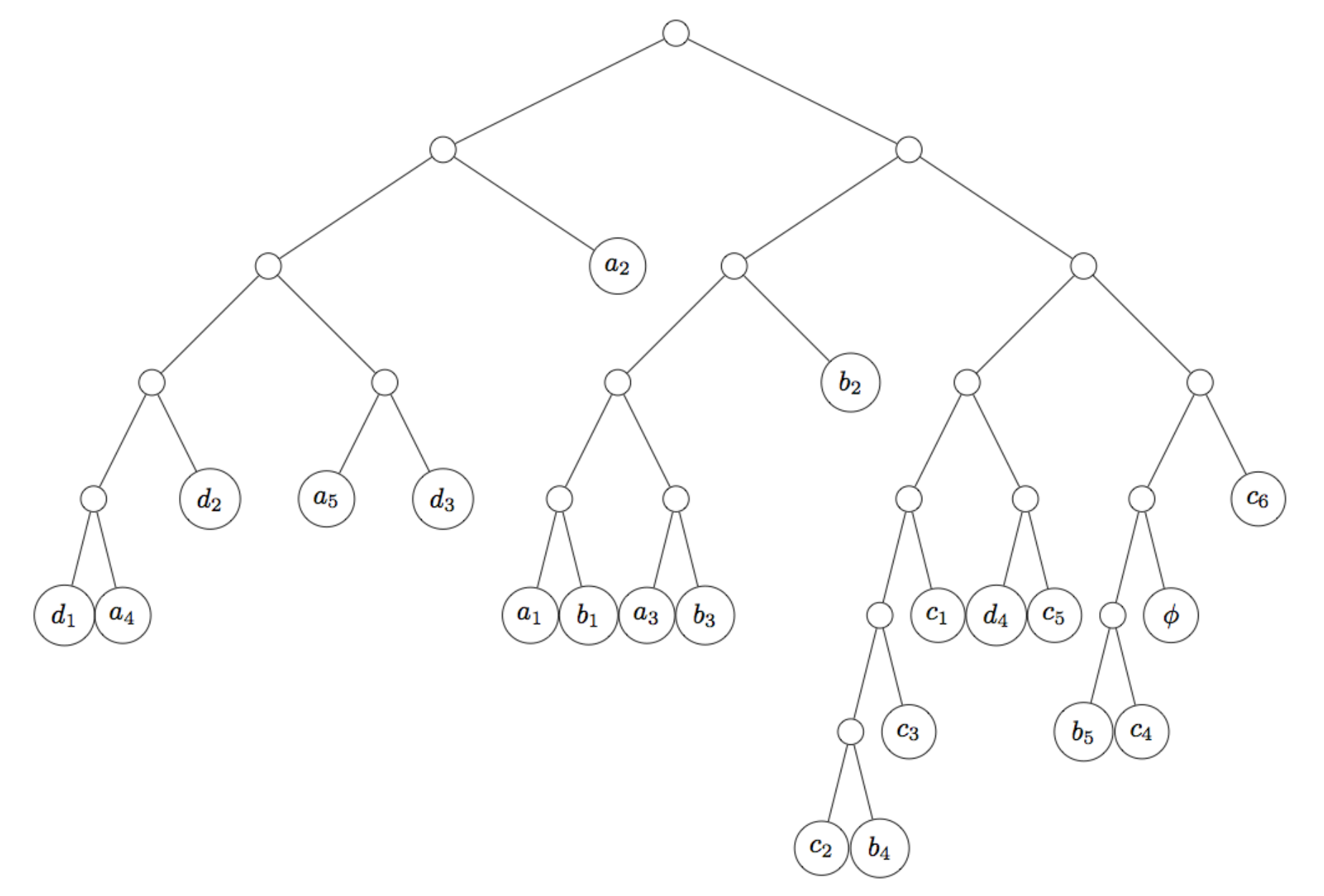




A 2D example of Binary Space Partition

K-dimensional (K-D) tree[3] is simpler than the complex BSP tree. For a K-D tree , each tree node is associated with a dimension, and a splitting plane perpendicular to the axis of the dimension.





A 2D example of K-D Tree with a naive splitting strategy that splits the space half by half each time

Compared to BSP tree, K-D tree is easier to construct and analyze, while providing a comparable performance.

Maintain the Tree for Kinetic Scene

As the tree structure is associated with the positions of the triangles in 3D space, we may need to update the tree structure when cell moves. We do this by maintaining a set of certificates for each triangle.[2] If the movement of a triangle violates a certificate for example, by moving past a splitting plane), then we do the update.