Kinetic Data Structures

Parker Given, Alex Flores, and Emmett Tran

May 4, 2017

Overview

A Kinetic Data Structure (KDS) is a data structure who's objects are changing in time according to a given function. These objects can be anything that has a value, but most of the time in computational geometry these objects are points.

Motivation & Applications

The main reason to use KDS's is the large performance increase that they bring to moving systems. KDS are used in computational geometry problems, computer vision problems, animation, collision problems, and many other things.

Operations

A KDS has two operations that it needs to support, advance(t) and change(v,f(t)). Advance advances the data structure to the given time. This function can usually be used to move time either forwards or backwards. Change allows you to change the function that the object v is following. The KDS must also have several underlying functions that do things such as; compute the certificates, compute when the certificates are invalidated, also called events, and functions for rebuilding the data structure when the certificates are invalidated. The KDS must also support the normal operations of the data structure.

Efficiency

There are four metrics of performance and efficiency when implementing a KDS. The first one is responsiveness. Responsiveness is the worst case time needed to fix the data structure after a certificate has become invalidated. If this amount of time is small, the KDS is described as responsive. The second metric is called locality, which is the maximum number of certificates that a single object or point is in. If this number is small then the KDS is described as local. The third metric is compactness. Compactness is the maximum number of certificates in use. A KDS is described as compact if the number of certificates is $O(n * polylog(n))$ or $O(n^{1+\epsilon})$ for an ϵ slightly larger than 1. The last metric is the worst case ratio of the number of events from t=0 to t = inf to the worst case number of changes to the data structure as t goes to inf. The KDS is efficient if this ration is small.

As an example, A sorted list KDS is described as responsive, local, compact, and efficient. It is responsive because it only takes ***O(log n)*** to fix the data structure. The sorted list is local because each object is only involved in at most 2 certificates. It is compact because there are *n-1* certificates for *n* objects. Finally, the KDS is efficiency because each time the objects change order, only one certificate fails.

Our Code

We planned on implementing a Delaunay triangulation KDS and then visualizing it. However, we were unable to successfully implement the KDS. We did produce a visualization of it, however. The visualization was coded in Python 3, and the visual part was written in Processing's Python mode. We first ported the DCEL data structure given to us in lab from Java to Python. We then programmed the visualization part. We had all of the points of the triangulation move on a random linear trajectory. When an event occurred we had the animation stop and highlight the offending points and edges. The triangulation was then fixed and the animation continued.