dolt-guide.cpp
#include <dolt-lite.h>
#include <iostream>
#include "intersection.h"
using namespace dolt;
using namespace std;

objectList sample_list;
list<drawablePoint> Pts;

void DOLT_LITE___QUICK_REFERENCE_GUIDE()
{
    // first, there are geometric representations
    Point p(0.5, 1.9);
    Point q(9.2, 3.7);
    Segment s(p, q); // the first point is the "start", the second is the "end"
    Polygon g;
    g.push_back(p);
    g.push_back(q);
    g.push_back(Point(5.2, 6.8));

    // there are *drawableObject* versions of the above that you can
    // initialize them as above, or with the regular geometric version.
    // Since these are subclasses of the geometric classes, anywhere you
    // can use, e.g., a Point, you can use a drawablePoint.
    drawablePoint dp(p);
    drawablePoint dq(9.2, 3.7);
    drawableSegment ds(p, dq);
    drawablePolygon dg(g);

    // there is also a geometric "lineStrip" class, but you will
    // probably only use the drawable version:
    drawableLineStrip dl; // a "strip" of connected line segments
    dl.push_back(dp);
    dl.push_back(Point(3, 4));
    dl.push_back(Point(5, 8));

    // for a drawableObject to be drawn, you have to put a pointer to it
    // on an objectList, and that objectList must be given to dolt. the
    // sample_list global (and others) are already set up for you in the
    // support code.
    //
    // you can't use a pointer to a temporary drawableObject, of course.
    // one way to get around this is to dynamically allocate the drawableObject
    sample_list.push_back(new drawableObject(dg));
    //
    // However, this obligates you to delete that object when you're
    // through with it. A better practice is to let an STL container do
    // the allocation (and therefore deletion when you, for example,
    // clear the container).
    Pts.push_back(q);
    // Then make sure you give a pointer to the drawableObject that the
    // container created:
    sample_list.push_back(&Pts.back());

    // one of the useful things with (drawable) points is that you can
    // add, subtract, scale, and take the length of them. They're sort
    // of like vectors in that regard. See geometry.h for more procedures.
    Point r = q - p;
    r = r / r.length();
    double c = cross(r, dp); // magnitude or z-component of cross product
    double d = dot(r, dp);

    // the segment class is a little limited right now. all you can
    // really do is access the endpoints:
    double m = (ds.start() - ds.end()).length();

    // Now, on to polygons...

    // There are two ways to access polygons. The first is by going
    // through the vertices.
    for (Polygon::Vertex_iterator k = g.begin();
        k != g.end(); ++k) {
        cout << "Polygon point is: [" << *k << "]" << endl;
    }

    // You can also do this with a constant iterator (in the 2/1 update
    // of dolt-lite)
    for (Polygon::Vertex_const_iterator k = g.begin();
        k != g.end(); ++k) {
        cout << "Polygon point is: [" << *k << "]" << endl;
    }

    // Finally, you can also go through the edges of the polygon, but
    // only with a const iterator:
    for (Polygon::Edge_const_iterator e = dg.edges_begin();
        e != dg.edges_end(); ++e) {
        cout << "The length of this edge is ".
        << (e->start() - e->end()).length() << endl;
        if (test_segmentIntersect(*e, s))
            cout << "This segment intersects with s!" << endl;
        else
            cout << "No intersection with s.\n" << endl;
    }

    // One other thing: there is a bounding box class. There aren't
    // really any operations for this class right now, but you can get
    // the bounding box of polygons which is useful...
    BBox b = g.bbox();
    cout << "bbox: " << b.bmin() << " , " << b.bmax() << endl;
}

// the segment class is a little limited right now. all you can
// really do is access the endpoints:
double m = (ds.start() - ds.end()).length();

// Now, on to polygons...

// There are two ways to access polygons. The first is by going
// through the vertices.
for (Polygon::Vertex_iterator k = g.begin();
    k != g.end(); ++k) {
    cout << "Polygon point is: [" << *k << "]" << endl;
}

// You can also do this with a constant iterator (in the 2/1 update
// of dolt-lite)
for (Polygon::Vertex_const_iterator k = g.begin();
    k != g.end(); ++k) {
    cout << "Polygon point is: [" << *k << "]" << endl;
}

// Finally, you can also go through the edges of the polygon, but
// only with a const iterator:
for (Polygon::Edge_const_iterator e = dg.edges_begin();
    e != dg.edges_end(); ++e) {
    cout << "The length of this edge is ".
    << (e->start() - e->end()).length() << endl;
    if (test_segmentIntersect(*e, s))
        cout << "This segment intersects with s!" << endl;
    else
        cout << "No intersection with s.\n" << endl;
}

// One other thing: there is a bounding box class. There aren't
// really any operations for this class right now, but you can get
// the bounding box of polygons which is useful...
BBox b = g.bbox();
cout << "bbox: " << b.bmin() << " , " << b.bmax() << endl;