Application Note: Design Rule Checking

Written: 6/25/2001
Updated: March, 2002

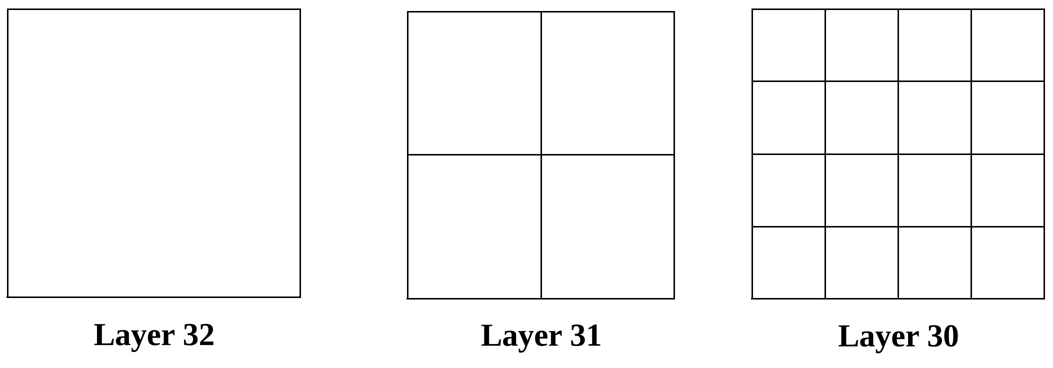
Problem

Given a set of rectangles, how quickly can you find all the rectangles that touch any given rectangle? "Touch" means that if you considered the rectangles to be solid, there would be at least one point in common.

Solution

Use Judy to implement a pointerless quadtree. The goal of a quadtree is to place your spatial data in a coordinate system hierarchy that divides up space in such a way that you only have to look at a relatively small number of divisions to determine a "touch" (or whatever lookup you are performing). When you place a rectangle into this hierarchy, the rectangle resides in the smallest box that is large enough to hold the rectangle (the minimum bounding box).

To divide up a coordinate system, start with a region the size of your universe (i.e. the whole coordinate system) and then recursively subdivide it into smaller and smaller regions. In this example, let's divide up the space by a power of 2 in each dimension:



Layer 32 contains a single grid with an x and y range of 0 to 2^{32}-1 (the universe in this example). Layer 31 contains four grids (subdivisions). Grid (0,0), the lower left, has a usable x and y range of 0 to 2^{31}-1. Grid (1,0) has an x usable range from 2^{31} to $2^{32} – 1$. The layers continue all the way through layer 0 which holds single points.

These layers of grids make up a database in which to put an object. Any object that gets put into this space, starting with layer 32, falls through these subspace regions (i.e. grids) until it hits a divider. An excellent analogy for this is a stacked rock sieve. This is a series of sieves that are stacked one upon another, starting with the coarsest gri d and proceeding down to finer and finer. When you put rocks in the top, they will fall until the grids are too fine for the rock to pass. Fortunately, we are dealing with fixed 2 -D data that doesn't rotate or slide like a rock, so we can calculate on which layer and which grid the rectangle should be stored on. The C macro:

```
#define Layercompute_m(xmin, ymin, xmax, ymax) \
        floor_log2_p1(MAX(((xmin) ^ (xmax)), ((ymin) ^ (ymax))))
```

computes on which layer any rectangle is placed. Rectangles are defined by their lower left and upper right coordinates (xmin, ymin), (xmax, ymax).

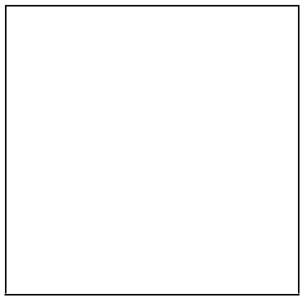
We can also calculate in which grid within a layer a rectangle is placed:

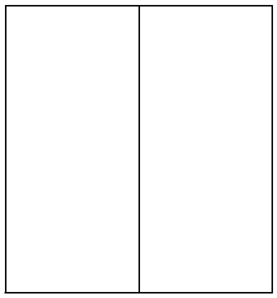
```
#define Gridcompute_m(layer, coord) ((coord) >> (layer))
```

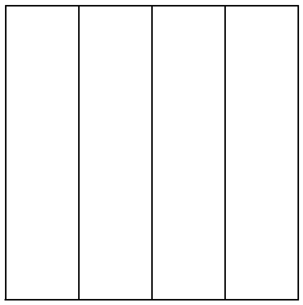
Representing this array of layers of grids of rectangles is frequently done with a quadtree that defines each grid and has four pointers to the grids below it. However, since we can calculate both the layer and the grid, we could represent the whole database as a multidimensional array. Since this array would have to contain more than 2^{62} grids it couldn't be represented with a conventional array. In our implementation we use JudyL.

Further Possibilities

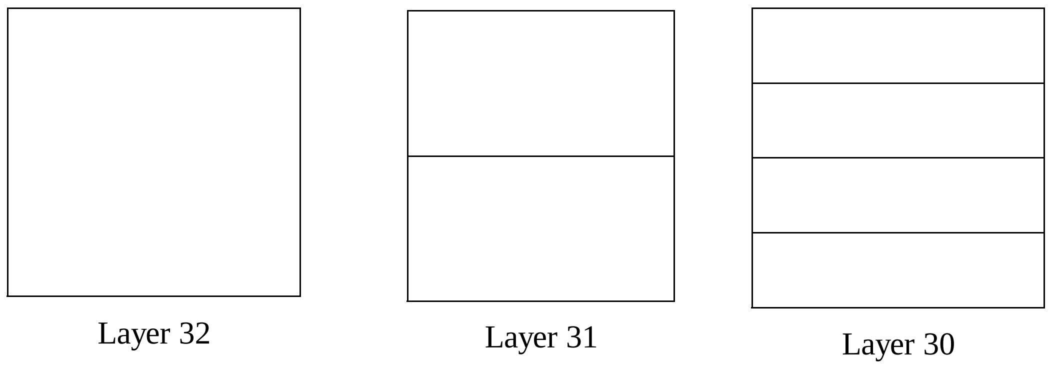
Some enhancements come to mind. One is that the search algorithm could easily be threaded. Another is that there is no reason to limit the division process to a power of 2 in both the x and y direction. Instead of dividing space into squares we could as easily divide it up into rectangles the width of the whole coordinate system. Check the aspect ratio of each rectangle. If it's vertical, put it into a hierarchy like:


Layer 32


Layer 31


Layer 30

If it’s horizontal, put it into a hierarchy like:



Example Code

```
// ===== jxy.h ================================================================

#ifndef JXY_H
#define JXY_H

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>

#define JUDYERROR_SAMPLE  // use default error handling.
#include <Judy.h>

#define NUMLAYERS 33      // in quadtree.

#ifndef MIN
#define MIN(a,b) ((a) < (b) ? (a) : (b))
#endif
#ifndef MAX
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#endif


// MACROS TO COMPUTE LAYER AND GRID (X OR Y):

#define      Layercompute_m(Xmin,Ymin,Xmax,Ymax) \
      floor_log2_p1(MAX(((Xmin) ^ (Xmax)), ((Ymin) ^ (Ymax))))

// Same as Layercompute_m but for one dimension:

#define      Layercompute1d_m(Xorymin,Xorymax) \
      floor_log2_p1((Xoryxmin) ^ (Xoryxmax))

#define      Gridcompute_m(Layer,Coord) ((Coord) >> (Layer))


// GET A LIST OF ALL THE POPULATED GRIDS BY X COORDINATE:
//
// This is a JudyL array for which the key is the X coord of the grid.  The
// value of each of these JudyL elements is a pointer to a JudyL array
// containing all the rectangles at the Y coords of the given X coord.

#define      XGrids_m(Sdbp, I) ((Sdbp)->layer[I])


// CHECK IF A RECTANGLE TOUCHES ANOTHER:
//
```

```
// Note:  xmin, xmax, ymin, ymax are in the context:

#define      iftouch_m(A)        \
       if (((A)->xmax >= xmin)   \
        && ((A)->xmin <= xmax)   \
        && ((A)->ymax >= ymin)   \
        && ((A)->ymin <= ymax))


// RECTANGLE OBJECT:

typedef struct object_t * PObject;

struct object_t {
    uint32_t xmin, ymin, xmax, ymax;
    uint32_t i;
    PObject  all_obj_link;
    PObject  same_bucket_link;
};


// ARGUMENTS OBJECT:

typedef struct args_t * PArgs_t;

struct args_t {
    Word_t xmin, ymin, xmax, ymax;
    Word_t hits, misses, found;
};


// LAYERS (SPATIAL DATABASE):

typedef struct { void * layer   [NUMLAYERS]; }   layers_t;
typedef struct { void * layer[2][NUMLAYERS]; } hvlayers_t; // 0=horiz, 1=vert.


// XYLIB PROTOTYPES:

PPvoid_t XYIns(layers_t * spatdb,
               Word_t xmin, Word_t ymin, Word_t xmax, Word_t ymax);

int XYTouch(layers_t * spatdb,
            Word_t xmin, Word_t ymin, Word_t xmax, Word_t ymax,
            int (* ProcessObjs) (PObject PCollection, Pvoid_t PArgs),
            Pvoid_t PArgs);

#endif // JXY_H


// ===== jxylib.c =============================================================

// XY Library
//
// The lowest and most left coordinate is (0,0).
//
// Terminology:
//    Multiple "layers" hold all rectangles (object_t's).
//    Each layer is composed of 1..n grids, where n = 2**(64 - (2 * layer)).
//    So layer 1 is can contain 2**64 grids with an expanse of 0 to 2.
//    Layer 2 can contain 2**60 grids with an expanse of 0 to 4.
//    Layer 3 can contain 2**58 grids with an expanse of 0 to 8.
//    Layer 32 can contain 1 grid with an expanse of 0 to 2**32.
//
//    Each layer contains a gridset (set of grids).
//    Each gridset contains 0..n grids.
//    Grids are examined for touches.

#include "jxy.h"

#ifdef FILTEROBJSFIRST
```

```
// FIND TOUCHES IN WHOLE GRID:

#define      ProcessGridm_m                            \
      for (PGridobj = *PPGridobj;                     \
           PGridobj != (object_t *) NULL;             \
           PGridobj = PGridobj->same_bucket_link)     \
      {                                        \
          iftouch_m(PGridobj) /* touch found, do something with it: */ \
          {                                    \
            rv = (*ProcessObjs)(PGridobj, PArgs);  \
            if (rv) return(rv);                \
          }                                    \
      }
#else

#define ProcessGrid_m                          \
      rv = (*ProcessObjs)(*PPGridobj, PArgs);\
      if (rv) return(rv);
#endif


// COMPUTE floor(log2(val))+1, 0 if val == 0:

static unsigned int floor_log2_p1(Word_t Val)
{
    unsigned int b = 17;

    if (Val <= 2)        return(Val);
    if (Val <   1<<16)          b  = 1;
    if (Val >= (1<< 7) << b) b += 8;
    if (Val >= (1<< 3) << b) b += 4;
    if (Val >= (1<< 1) << b) b += 2;

    return(b + (Val >= (1<<0) << b));
}
#pragma INLINE floor_log2_p1


// ****************************************************************************
// X Y   I N S
//
// Insert a rectangle defined by the lower left and upper right coordinates
// (xmin, ymin), (xmax, ymax) into the spatial database spatdb:
//
// ACTUALLY, this just returns a pointer to a pointer to the grid.

PPvoid_t XYIns(
    layers_t * spatdb,
    Word_t   xmin, Word_t ymin, Word_t xmax, Word_t ymax)
{
    Word_t   tlayer;           // layer that would contain test coords.
    Word_t   tgridX1, tgridY1;
    PPvoid_t PPJLYGrids;       // JudyL array pointed to from JLXGrids.
                                 //   key: Y1 grid coord, val: PGridobj.
    PPvoid_t PPGridobj;        // pointer to pointer to grid of objects.

    assert(xmin <= xmax);
    assert(ymin <= ymax);
    assert(xmax <= 0xffffffff);
    assert(ymax <= 0xffffffff);

// The layer number:
//
// Layer 31 is the entire expanse, that is, a grid with 2^32 x 2^32 or 2^64 x
// 2^64 units.  Layer 0 is a single point.

    tlayer = Layercompute_m(xmin, ymin, xmax, ymax);
    assert(tlayer <= NUMLAYERS);

    tgridX1 = Gridcompute_m(tlayer, xmin);
    tgridY1 = Gridcompute_m(tlayer, ymin);
```

```
        JLI(PPJLYGrids, XGrids_m(spatdb, tlayer), tgridX1);

    // Get the gridobj array:

        JLI(PPGridobj, *PPJLYGrids, tgridY1);
        return(PPGridobj);

    } // XYIns()

    // ***************************************************************************
    // X Y   T O U C H
    //
    // XYTouch returns ProcessObjs: Value if it is nonzero.
    // Test rectangle coords are (xmin, ymin), (xmax, ymax).

    int XYTouch(
        layers_t * spatdb,
        Word_t   xmin, Word_t ymin, Word_t xmax, Word_t ymax,
        int (* ProcessObjs) (PObject PCollection, Pvoid_t PArgs),
        Pvoid_t    PArgs)
    {
        Word_t   tlayer;      // layer that would contain test coords.
        Pvoid_t  PJLXGrids;   // JudyL array of grids in layer.
                              //  key: X1 grid coord, val: PJLYGrids.
        PPvoid_t PPJLYGrids;  // JudyL array pointed to from JLXGrids.
                              //  key: Y1 grid coord, val: Pgridobj.
        PPvoid_t PPGridobj;   // pointer to pointer to grid of objects.
        PObject  PGridobj;    // pointer to grid objects.
        Word_t   currlayer;   // current layer being examined.
        Word_t   tgridX1, tgridY1, tgridX2, tgridY2;
        Word_t   gridX, gridY;
        int      rv;          // generic int return value.
        int      i;
        struct args_t * Args = (PArgs_t) PArgs;


    // COMPUTE LAYER AND LOWER LEFT GRID OF THE TEST RECTANGLE:

        tlayer = Layercompute_m(xmin, ymin, xmax, ymax);
        assert(tlayer <= NUMLAYERS);


    // FIND ALL TOUCHES IN LAYER 0:
    //
    // Treat layer 0 (the point layer) separately since each grid in the layer that
    // contains any part of the test rectangle must be a touch.  This special
    // treatment isn't necessary but seems like a good optimization.

        currlayer = 0;

    // Compute ending grids (same as coords in this layer):

        tgridX1 = gridX = xmin;
        tgridY1 = ymin;
        tgridX2 = xmax;
        tgridY2 = ymax;

        PJLXGrids = XGrids_m(spatdb, currlayer);
        if (PJLXGrids)
        {
          JLF(PPJLYGrids, PJLXGrids, gridX);

    // Get all the Y grids at this X coord:

          while ((PPJLYGrids != NULL) && (gridX <= tgridX2))
            {
              gridY = tgridY1;
              JLF(PPGridobj, *PPJLYGrids, gridY);

              while ((PPGridobj != NULL) && (gridY <= tgridY2))
              {
```

```
// All the objects in this grid are touches:
//
// Note:  The ProcessObjs() function may check all the rectangles for touches;
// this is unnecessary.

             rv = (*ProcessObjs)(*PPGridobj, PArgs);
             if (rv) return(rv);

             JLN(PPGridobj, *PPJLYGrids, gridY);

          } // Y grid loop.

          JLN(PPJLYGrids, PJLXGrids, gridX);

      } // X grid loop.
    } // non-empty layer 0.


// FIND ALL TOUCHES IN LAYERS BETWEEN 0 AND TLAYER:
//
// FIRST:  check for completely enclosed rectangles
// SECOND: check edges
// THIRD:  check corners
//
// Compute starting and ending grid coordinates:

    tgridX1 >>= 1;
    tgridY1 >>= 1;
    tgridX2 >>= 1;
    tgridY2 >>= 1;

    for (currlayer = 1;
        currlayer < tlayer;
        ++currlayer,
        tgridX1 >>= 1, tgridY1 >>= 1, tgridX2 >>= 1, tgridY2 >>= 1)
    {
      gridX = tgridX1;

      PJLXGrids = XGrids_m(spatdb, currlayer);
      if (! PJLXGrids) continue; // ignore unpopulated layers.

// FIRST:  Find internal grids (grids totally enclosed by the test rectangle in
// this layer -- all the rectangles in these internal grids are touches:

      JLN(PPJLYGrids, PJLXGrids, gridX);

// Get all the Y grids at this X coord:

      while ((PPJLYGrids != NULL) && (gridX < tgridX2))
      {
          gridY = tgridY1;
          JLN(PPGridobj, *PPJLYGrids, gridY);

          while ((PPGridobj != NULL) && (gridY < tgridY2))
            {

// All the objects in this grid are touches:
//
// Note:  The ProcessObjs() function may check all the rectangles for touches;
// this is unnecessary.

             rv = (*ProcessObjs)(*PPGridobj, PArgs);
             if (rv) return(rv);

             JLN(PPGridobj, *PPJLYGrids, gridY);

          } // Y grid loop.

          JLN(PPJLYGrids, PJLXGrids, gridX);

      } // X grid loop.
```

```
// SECOND:  Examine 4 edges:
//
// TOP AND BOTTOM EDGES:

        gridX = tgridX1;
        gridY = tgridY1;

        JLN(PPJLYGrids, PJLXGrids, gridX);
        while ((PPJLYGrids != NULL) && (gridX < tgridX2))
        {

// Get all the grids at each of these X coords:
//
// Check top edge:

            JLG(PPGridobj, *PPJLYGrids, tgridY2);
            if (PPGridobj != NULL) { ProcessGrid_m; /* may return */ }

// Check bottom edge:

            if (tgridY1 != tgridY2)
            {
               JLG(PPGridobj, *PPJLYGrids, tgridY1);
               if (PPGridobj != NULL) { ProcessGrid_m; /* may return */ }
            }

            JLN(PPJLYGrids, PJLXGrids, gridX);

        } // X grid loop.


// LEFT EDGE:

        gridX = tgridX1;
        gridY = tgridY1;

        JLG(PPJLYGrids, PJLXGrids, tgridX1);
        if (PPJLYGrids != NULL)
        {

// Get all the grids at this X coord:

            JLN(PPGridobj, *PPJLYGrids, gridY);
            while ((PPGridobj != NULL) && (gridY < tgridY2))
            {
               ProcessGrid_m;                    // may return.
               JLN(PPGridobj, *PPJLYGrids, gridY);
            }

// Left edge, lower corner:
//
// Get all the grids at this X coord:

            JLG(PPGridobj, *PPJLYGrids, tgridY1);
            if (PPGridobj != NULL) { ProcessGrid_m; /* may return */ }

// Left edge, upper corner:

            if (tgridY1 != tgridY2)
            {
               JLG(PPGridobj, *PPJLYGrids, tgridY2);
               if (PPGridobj != NULL) { ProcessGrid_m; /* may return */ }
            }
        } // X grid.


// RIGHT EDGE:

        if (tgridX1 != tgridX2)          // left and right edges differ.
        {
            gridX = tgridX1;
            gridY = tgridY1;

            JLG(PPJLYGrids, PJLXGrids, tgridX2);
            if (PPJLYGrids != NULL)
            {

// Get all the grids at this X coord:

              JLN(PPGridobj, *PPJLYGrids, gridY);
              while ((PPGridobj != NULL) && (gridY < tgridY2))
              {
                  ProcessGrid_m;        // may return.
                  JLN(PPGridobj, *PPJLYGrids, gridY);
              }

// Right edge, lower corner:

              JLG(PPGridobj, *PPJLYGrids, tgridY1);
              if (PPGridobj != NULL) { ProcessGrid_m; /* may return */ }

// Right edge, upper corner:

              if (tgridY1 != tgridY2)
              {
                  JLG(PPGridobj, *PPJLYGrids, tgridY2);
                  if (PPGridobj != NULL) { ProcessGrid_m; /* may return */ }
              }
            } // X grid.
        } // left and right edges are not the same edge.
    } // loop from layer 0 to tlayer.


// FIND ALL TOUCHES AT AND ABOVE TLAYER:
//
// Since this looks at progressively larger grids, only a single parent grid
// must be examined, but the whole rectangle (both corners) must be checked.

    for (currlayer = tlayer;
         currlayer < NUMLAYERS;
         ++currlayer,
         tgridX1 >>= 1, tgridY1 >>= 1, tgridX2 >>= 1, tgridY2 >>= 1)
    {
        PJLXGrids = XGrids_m(spatdb, currlayer);
        if (!PJLXGrids) continue;       // ignore empty grids.

        JLG(PPJLYGrids, PJLXGrids, tgridX1);
        if (PPJLYGrids != NULL)
        {

// Get the grid at this X coord:

            JLG(PPGridobj, *PPJLYGrids, tgridY1);
            if (PPGridobj != NULL) { ProcessGrid_m; /* may return */ }
        } // X grid.
    } // end of touches above test rectangle layer.

   return(0);

} // XYTouch()


// ****************************************************************************
// P R I N T   S T A T S

int PrintStats(layers_t * spatdb)
{
    Word_t   obj_count;
    Word_t   tlayer;      // layer that would contain test coords.
    Pvoid_t  PJLXGrids;   // JudyL array of grids in layer.
                          //   key: X1 grid coord, val: PPJLYGrids.
    PPvoid_t PPJLYGrids;  // JudyL array pointed to from PJLXGrids.
```

```
                              //   key: Y1 grid coord, val: Pgridobj.
        PPvoid_t PPGridobj;   // pointer to pointer to grid of objects.
        PObject  PGridobj;    // pointer to grid objects.
        Word_t   currlayer;   // current layer being examined.
        Word_t   gridX, gridY;
        Word_t   objcnt, worst;
        Word_t   objtot, gridtot, layertot;

        printf("\n--- Stats ---\n");

    // Compute ending grids (same as coords in this layer):

        worst    = 0;
        objtot   = 0;
        gridtot  = 0;
        layertot = 0;

        for (currlayer = 0, obj_count = 0; currlayer < NUMLAYERS; ++currlayer)
        {
          PJLXGrids = XGrids_m(spatdb, currlayer);
          if (PJLXGrids)
          {
              ++layertot;
              gridX = 0;

              JLF(PPJLYGrids, PJLXGrids, gridX);

              while (PPJLYGrids != NULL)
              {

    // Get all the Y grids at this X coord:

                gridY = 0;
                JLF(PPGridobj, *PPJLYGrids, gridY);

                while (PPGridobj != NULL)
                {
                    ++gridtot;
                    objcnt = 0;

    // Count all the objects in this grid:

                    for (PGridobj  = *PPGridobj;
                        PGridobj != NULL;
                        PGridobj  = PGridobj->same_bucket_link)
                    {
                      ++objcnt;
                    }

                    if (worst < objcnt) worst = objcnt;
                    objtot += objcnt;

                    JLN(PPGridobj, *PPJLYGrids, gridY);

                } // object loop.

                JLN(PPJLYGrids, PJLXGrids, gridX);

              } // Y grid loop.
          } // X grids.
        } // layer.

        printf("Total layers       = %ld\n", layertot);
        printf("Total grids        = %ld\n", gridtot);
        printf("Total objects      = %ld\n", objtot);
        printf("Avg objects/grid   = %ld\n", objtot / gridtot);
        printf("Worst objects/grid = %ld\n", worst);

        return(0);

    } // PrintStats().
```