R-Trees

Albert-Jan Yzelman

December 10, 2007

Outline

R-trees

Goals

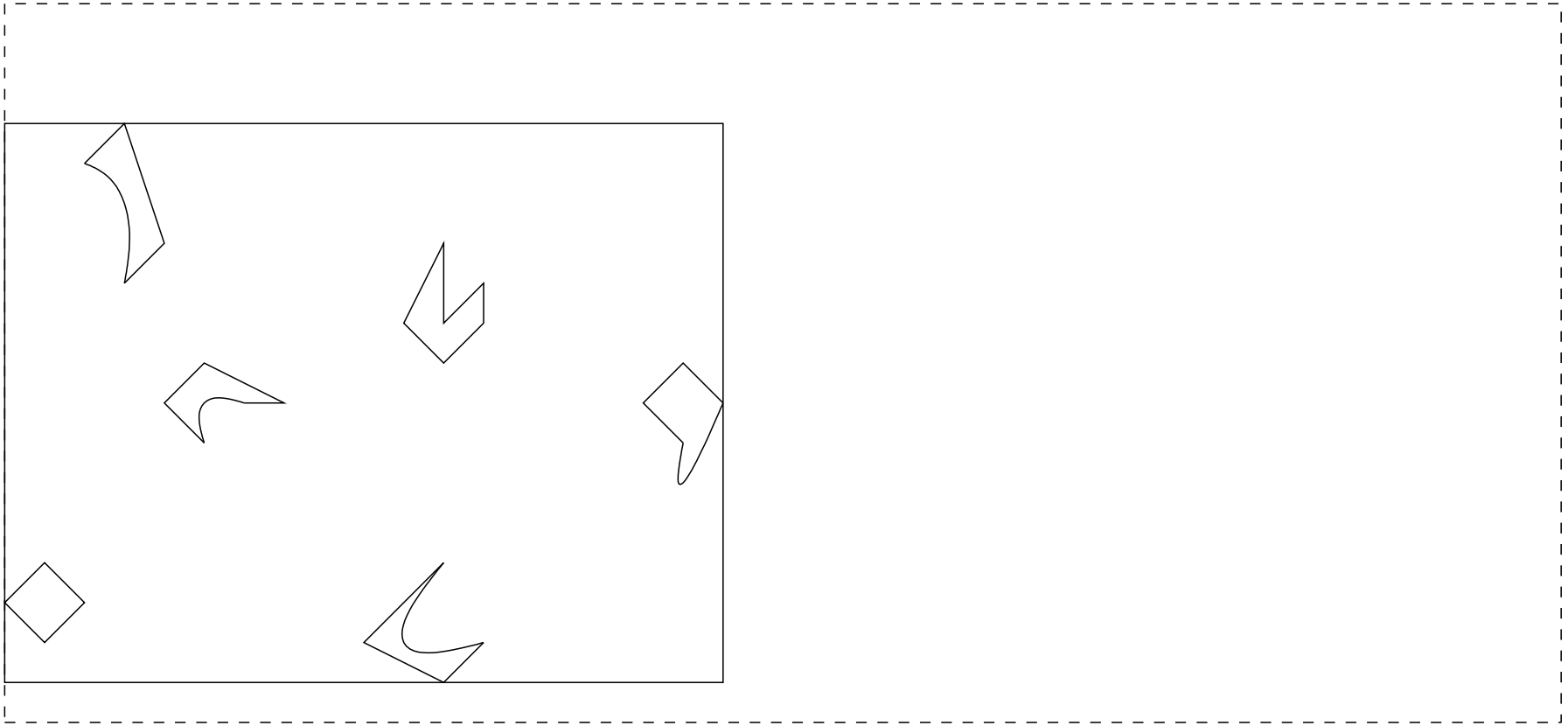
1. Create a datastructure which will enhance Shell's oil reservoir simulation software
2. Obtain a generic, customisable R-tree software library for using R-trees in many other areas

Reminder

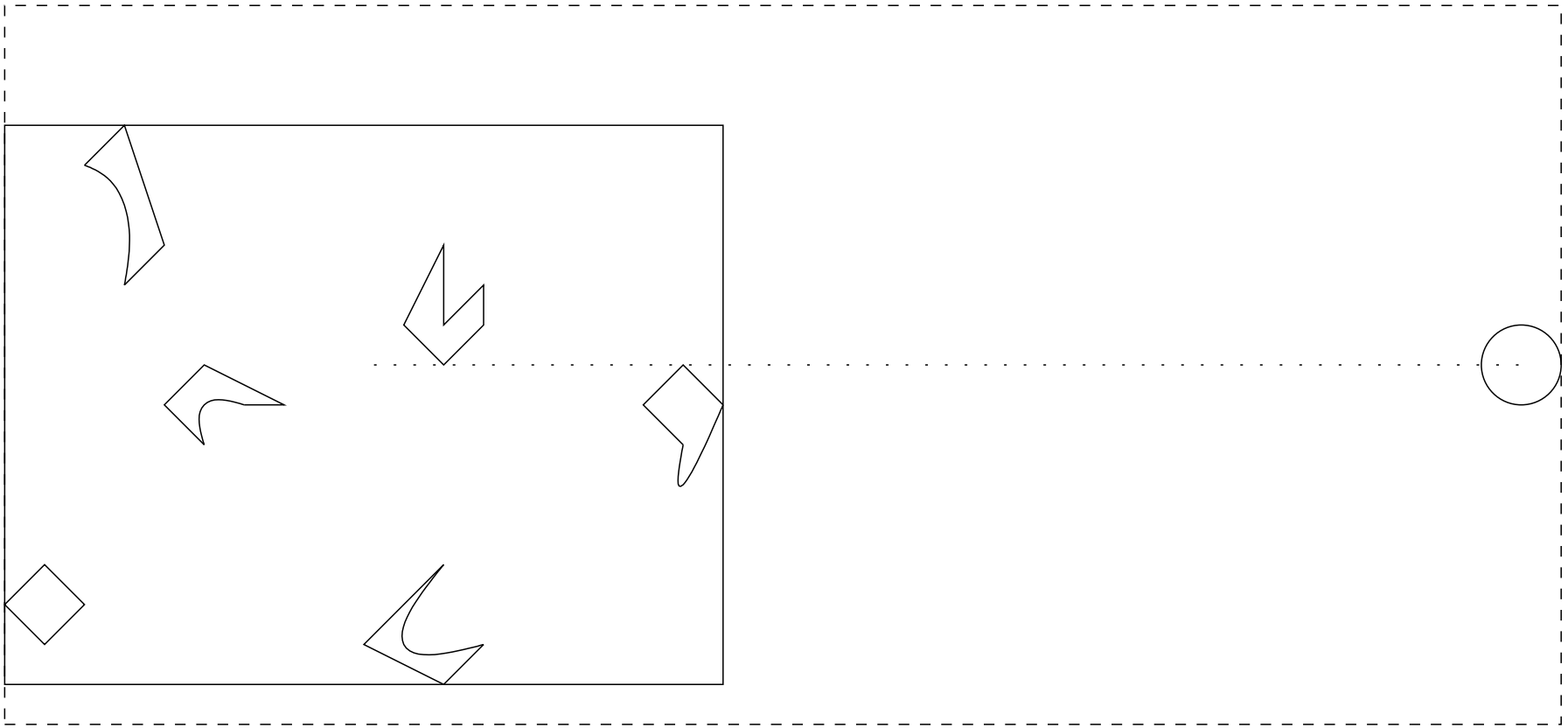
We can let each node of a tree correspond to an MBR and obtain the so called R-trees.

- Let each internal node of the R-tree correspond to the MBR of all MBRs stored in the children of that internal node
- Let each leaf node of the R-tree correspond to the MBR of a single object stored in the R-tree

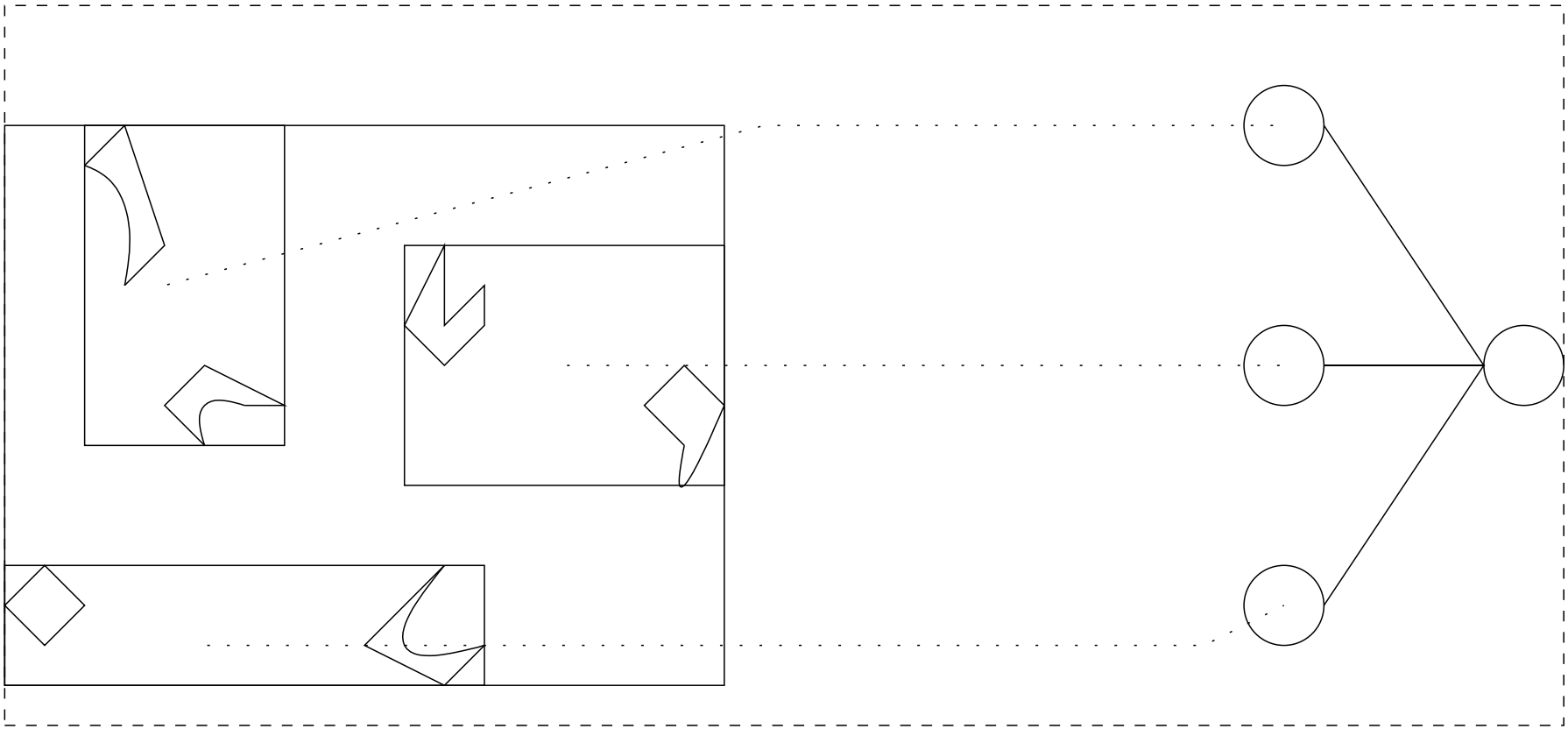
Reminder



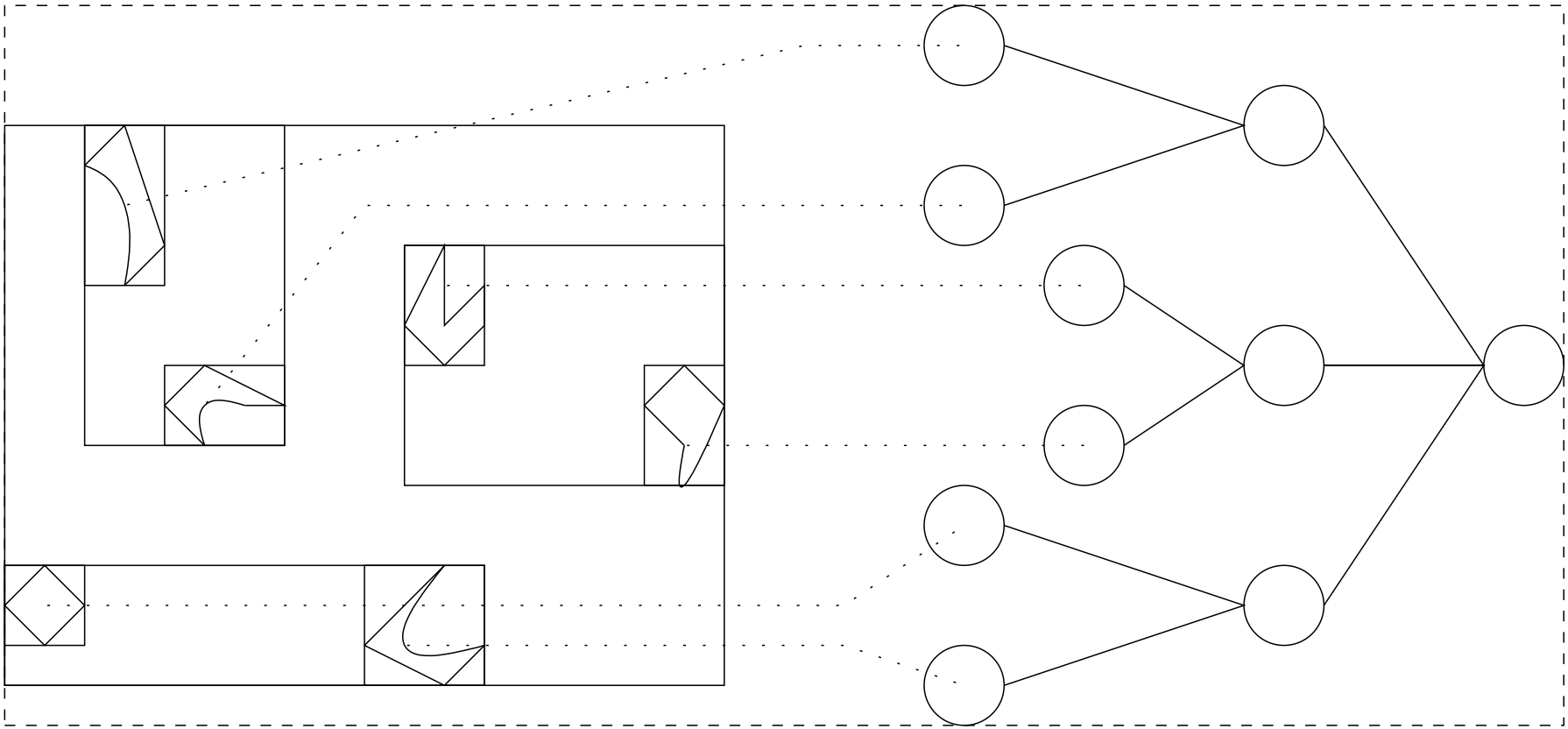
Reminder



Reminder



Reminder



Implemented operations

The R-tree library software presently enables one to obtain efficient answers to the following types of queries:

- Point
- Line
- Box
- k-nearest-neighbourhood

Storage scheme

- Presently stores *container elements* consisting of a d-dimensional *hypercube* with an identification number.

Storage scheme

- Presently stores *container elements* consisting of a d-dimensional *hypercube* with an identification number.
- Can be easily extended to hyperspherical container elements,
- or any other volume.

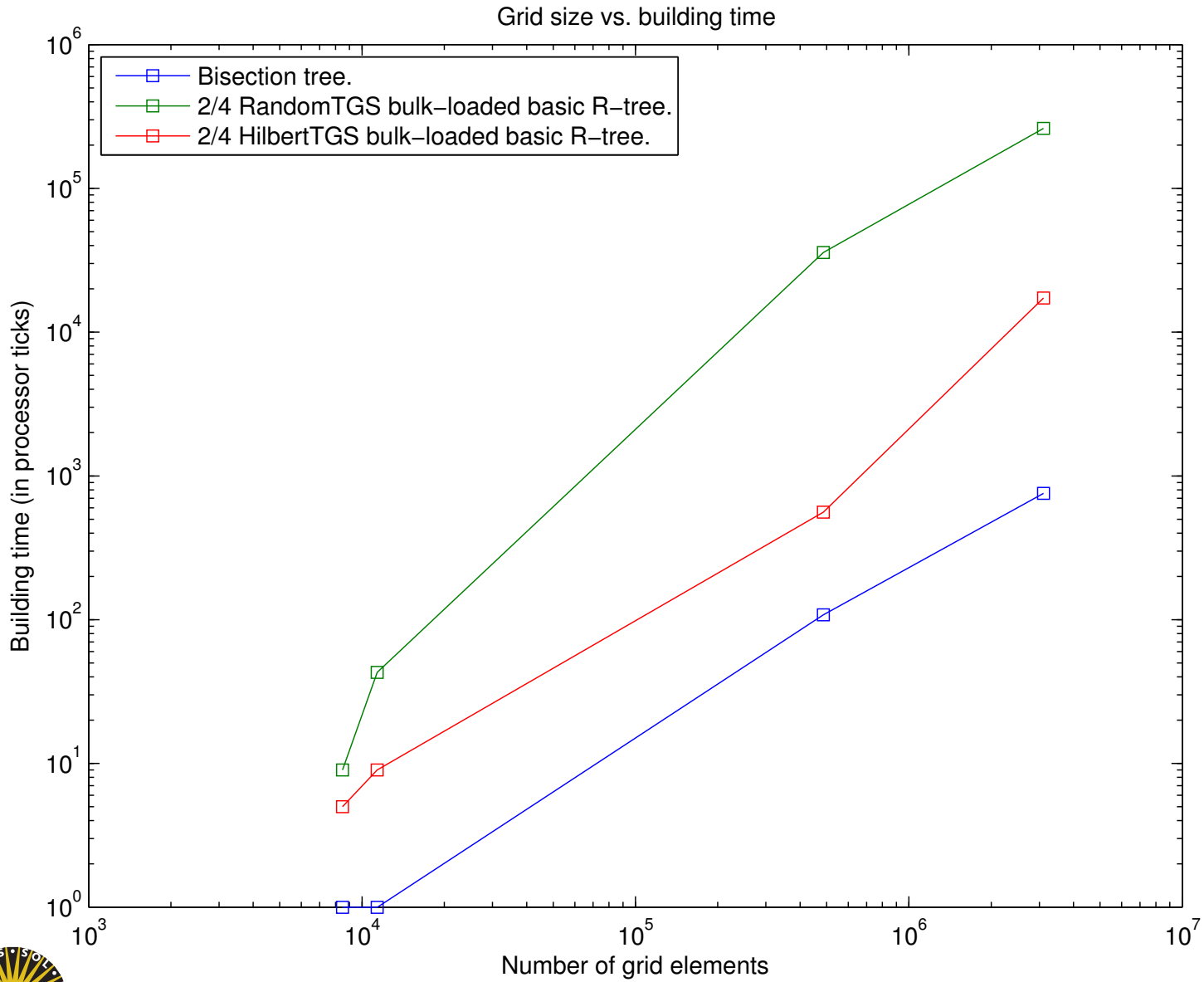
Implemented R-tree variants

- Basic R-tree with linear splitting
- Basic R-tree with quadratic splitting
- Hilbert R-tree

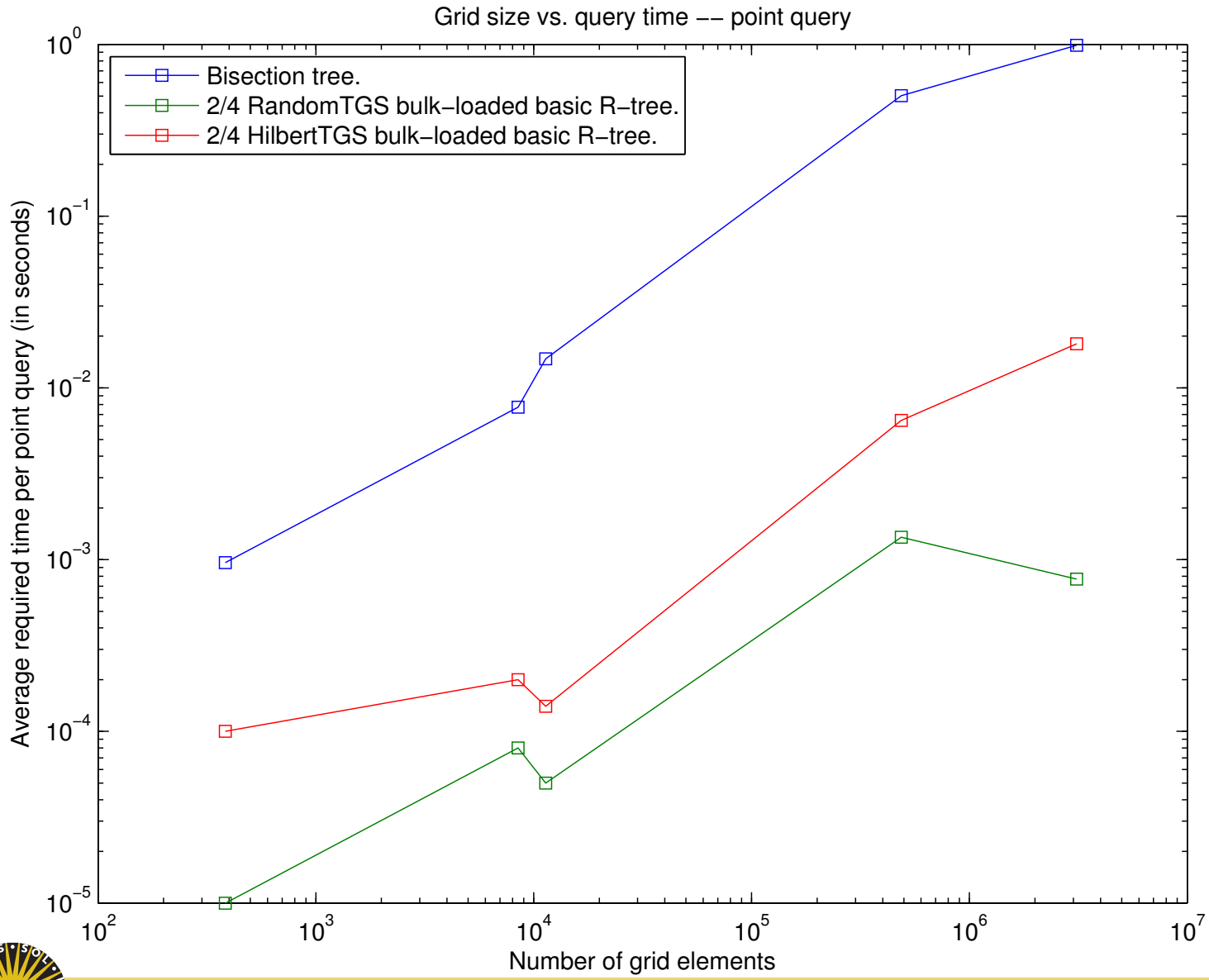
Implemented bulk-loading schemes

- Top-down Greedy Split (TGS)
  - total volume as cost function
  - intersection volume as cost function
- Random TGS
- Hilbert TGS
- Hilbert Packed

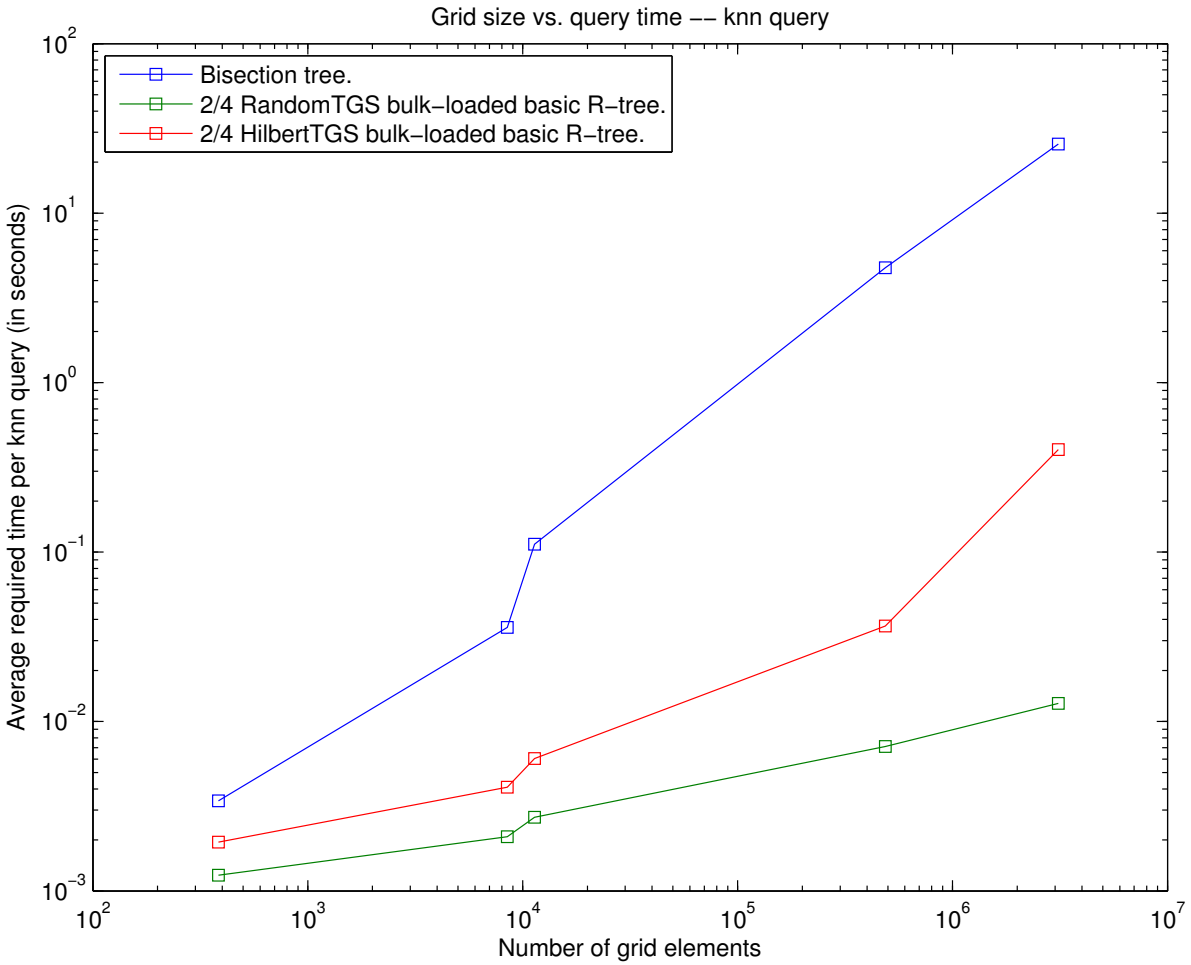
Experiments: construction time



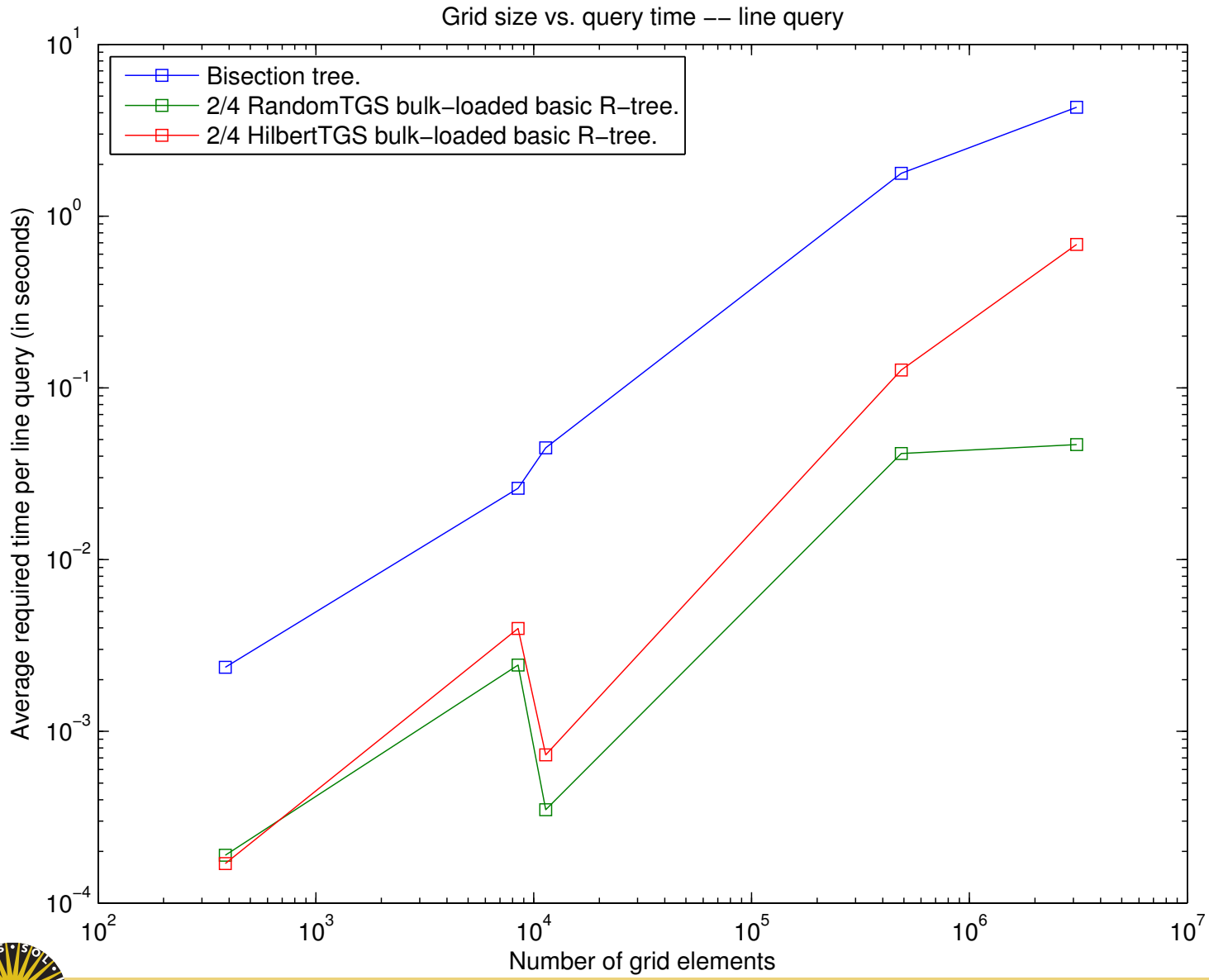
Experiments: point query time



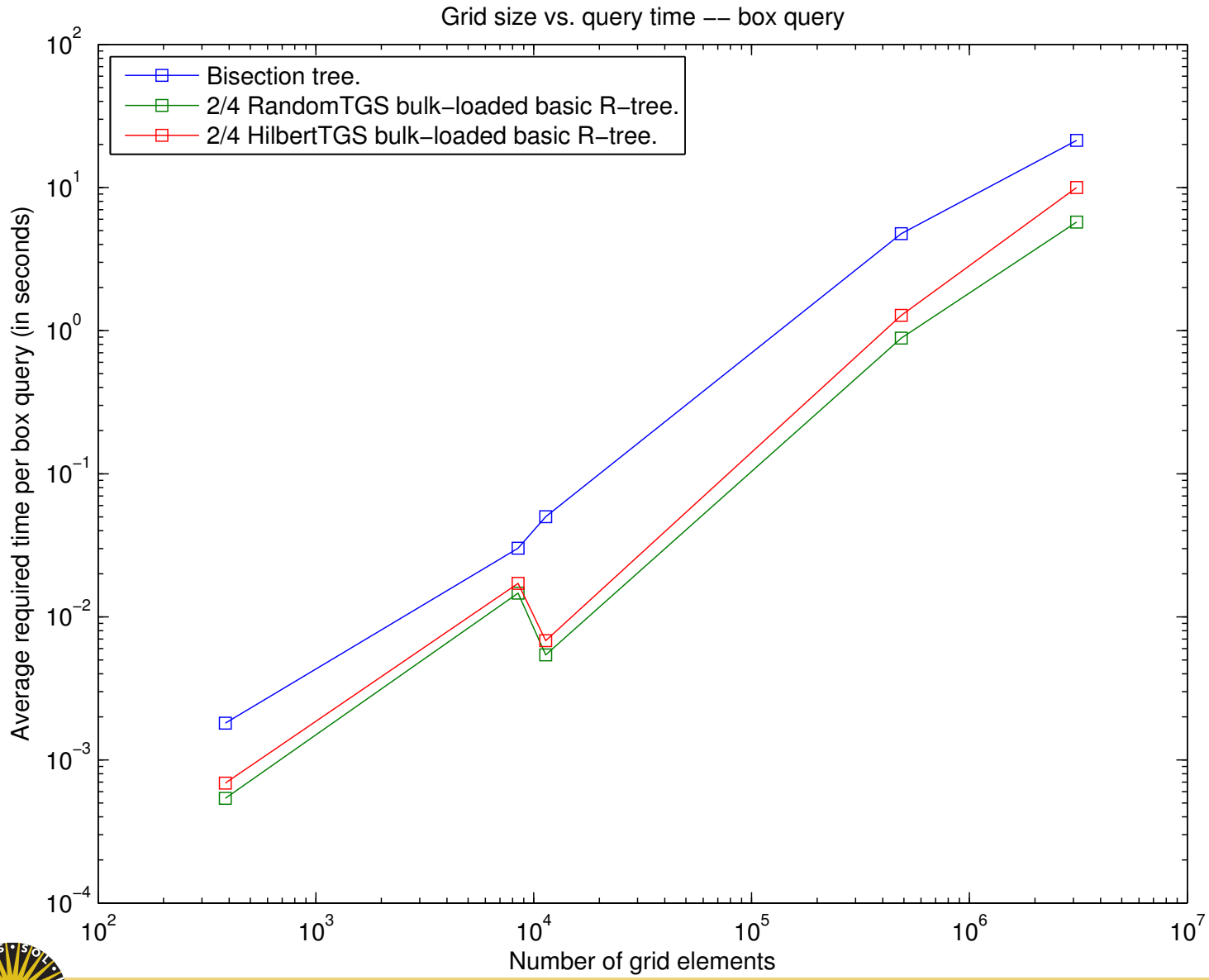
Experiments: knn query time



Experiments: line query time



Experiments: box query time



Outline

R-trees

The previously mentioned variations have been implemented in C++.
The resulting library went public as an open-source project:

`http://www.sourceforge.net/projects/rtree-lib`

The R-tree library as available there only includes the core R-tree sources; experimentation software used with the Shell datasets are *not* included.
The first release available is in effect the same source used for generating the results just seen. The SVN repository however already contains a newer version; Petr Sereda optimised an essential part of the software resulting in large (20 percent?) speedups in TGS bulk-loading.

Coding example

We will now demonstrate the ease of use of the library as is. We will use the release version as available on SF (which is at 54 downloads at the time of writing).
After downloading the tarball we simply unzip it in a directory (./rtree-ex/) and unpack it. We then type "make" to compile the library object files; this runs fine on our test linux machine. Our example application will be aptly called example.cpp.

Pair.h

```
class Pair {
  public:
    double x;
    double y;
    Pair( double _x, double _y ){ x=_x; y=_y; }
};
```

example.cpp

```
#include<iostream>
#include<vector>

std::vector<Pair> data;

int main() {
  createData();
  loadData();
  doQueries();
}
```

example.cpp

```
void createData() {
    double cx, cy;
    Pair* temp;
    while( true ) {
        std::cin >> cx;
        std::cin >> cy;
        if ( cx == 0 && cy == 0 )
            return;
        temp = new Pair( cx, cy );
        data.push_back( temp );
    }
}
```

example.cpp

```
#include "Hilbert_R-Tree.h"
#include "HilbertTGS.h"
typedef Hilbert_TGS_tree< Hilbert_R_tree > tree_type;
tree_type* tree;

void loadData() {
  R_tree_props* props = new R_tree_props();
  props->minimum = 2;
  props->maximum = 6;
  tree = new tree_type( props );
  tree->insertElements( data.begin(), data.end() );
  tree->ensureReady();
}
```

Data translation; utils/BBGenerator.h

```
#include "Polytope.h"
#include "Cubic_Bounding_Box_Container.h"
#include "../Pair.h"

Cubic_Bounding_Box_Container*
extractCubicBoundingBox( const Polytope * const poly );

Cubic_Bounding_Box_Container*
extractCubicBoundingBox( Pair * pair );
```

Data translation; utils/BBGenerator.cpp

```
Cubic_Bounding_Box_Container*
extractCubicBoundingBox( Pair * pair ) {
  Cubic_Bounding_Box_Container *temp;
  std::vector< double > min; min.resize(2);
  std::vector< double > max; max.resize(2);
  min[0]=pair->x-0.05; min[1]=pair->y-0.05;
  max[0]=pair->x+0.05; max[1]=pair->y+0.05;
  temp = new Cubic_Bounding_Box_Container(
      min, max, rand()
  );
  return temp;
}
```

example.cpp

```
void doQueries() {
    double qx, qy;
    while( true ) {
        std::cin >> qx;
        std::cin >> qy;
        if ( qx==0 && qy==0 )
            break;
        Point p(2);
        p.setCoordinate(0, qx); p.setCoordinate(1, qy);
        vector<int> hits = tree->intersects( p );
        if (hits.size()>0)
            std::cout << hits.at( 0 );
    }
}
```

Outline

R-trees

Future plans:

- Even more performance tuning
- Additional query type: *hyperplane query*
- Better build system (automake)
- Contacting `http://www.rtreeportal.org/` for more publicity
- Including example applications on the sourceforge page

Possible other application areas:

- Databases
- Graphics/Gaming
- Chip Design
- Navigation Systems
- Image Processing
- Geographical Information Systems (GIS)
- Robot Control
- PCB Manufacturing
- ...?

See also:

`http://home.altennederland.nl/wiki/index.php/R-Tree_Project#Ideas`

Questions?

Grouping criteria

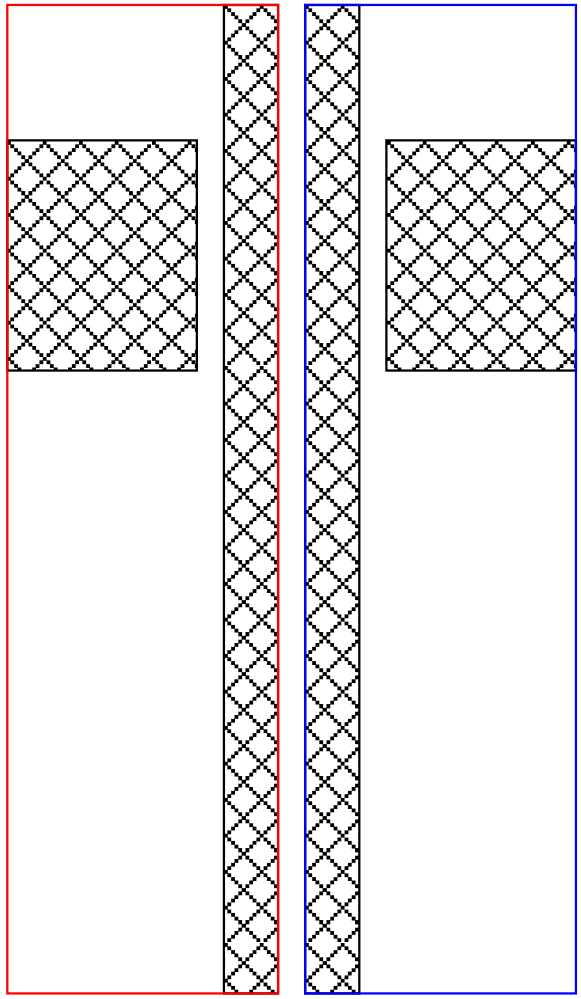


Figure: Minimum overlap criteria

Grouping criteria

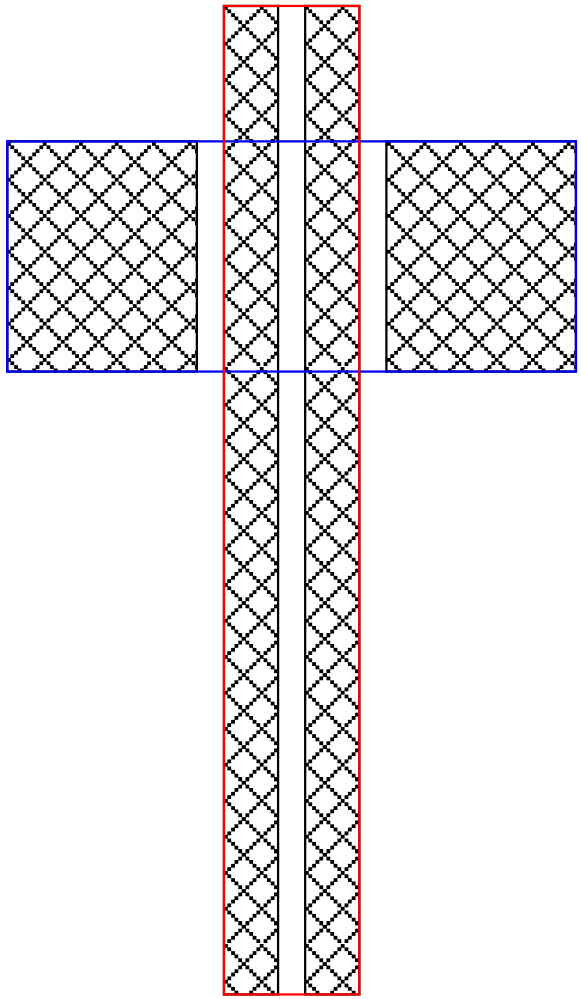
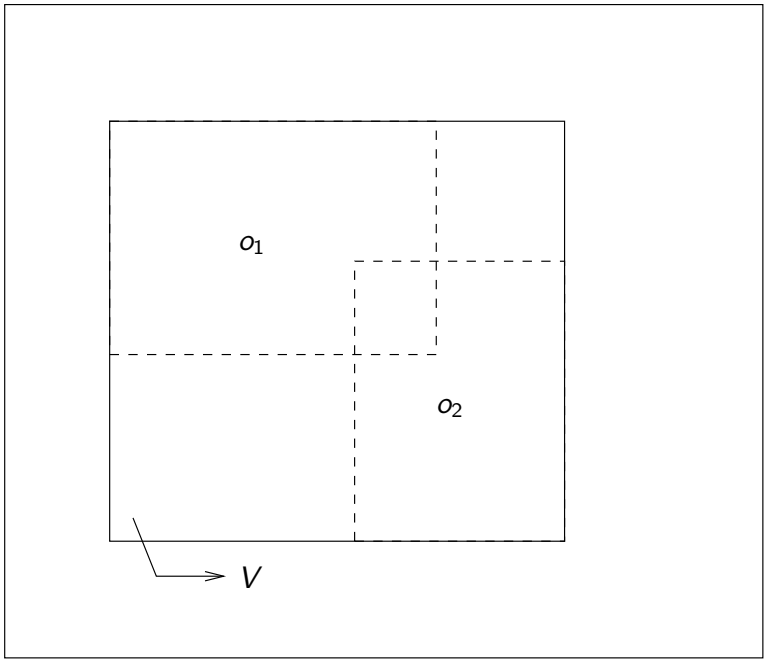


Figure: Minimum total volume criteria

Grouping criteria

Dead space: Volume($V - W$)



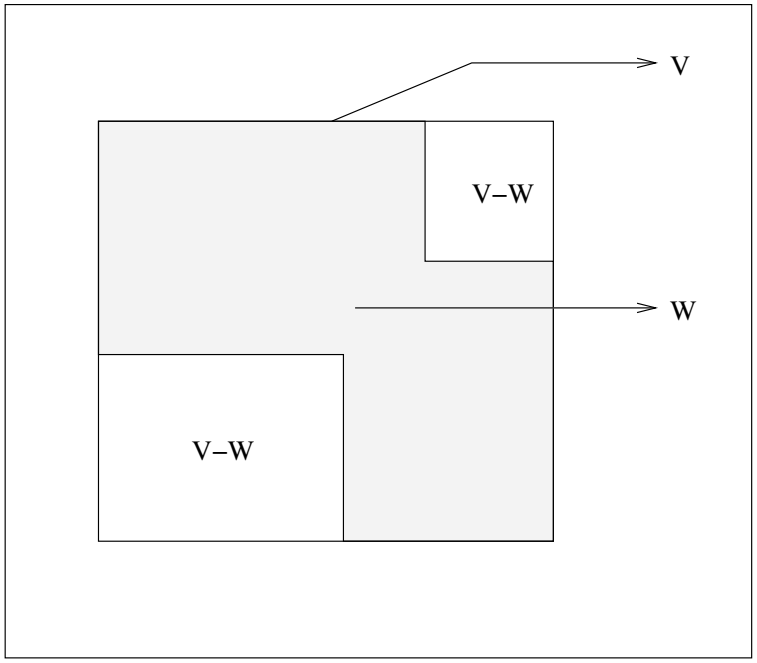


Figure: $V = \text{MBR}(o_1 \cup o_2)$, $W = \text{MBR}(o_1) \cup \text{MBR}(o_2)$

Packed Hilbert R-tree

Observation:

- Which object is in which leaf node defines the whole R-tree

Packed Hilbert R-tree

Observation:

- Which object is in which leaf node defines the whole R-tree
- So if the objects to-be added are known beforehand, why not build the tree ground-up?

Packed Hilbert R-tree

Initially, sort objects on Hilbert coordinates. Let an ordered set N of leaf nodes correspond to those objects.

- Group N into subsets N_i, each of size M
- Let $\tilde{N}$ correspond to an ordered set of internal node, each with children as in a subset N_i
- If $|\tilde{N}| > 1$, repeat process with $N = \tilde{N}$

Packed Hilbert R-tree

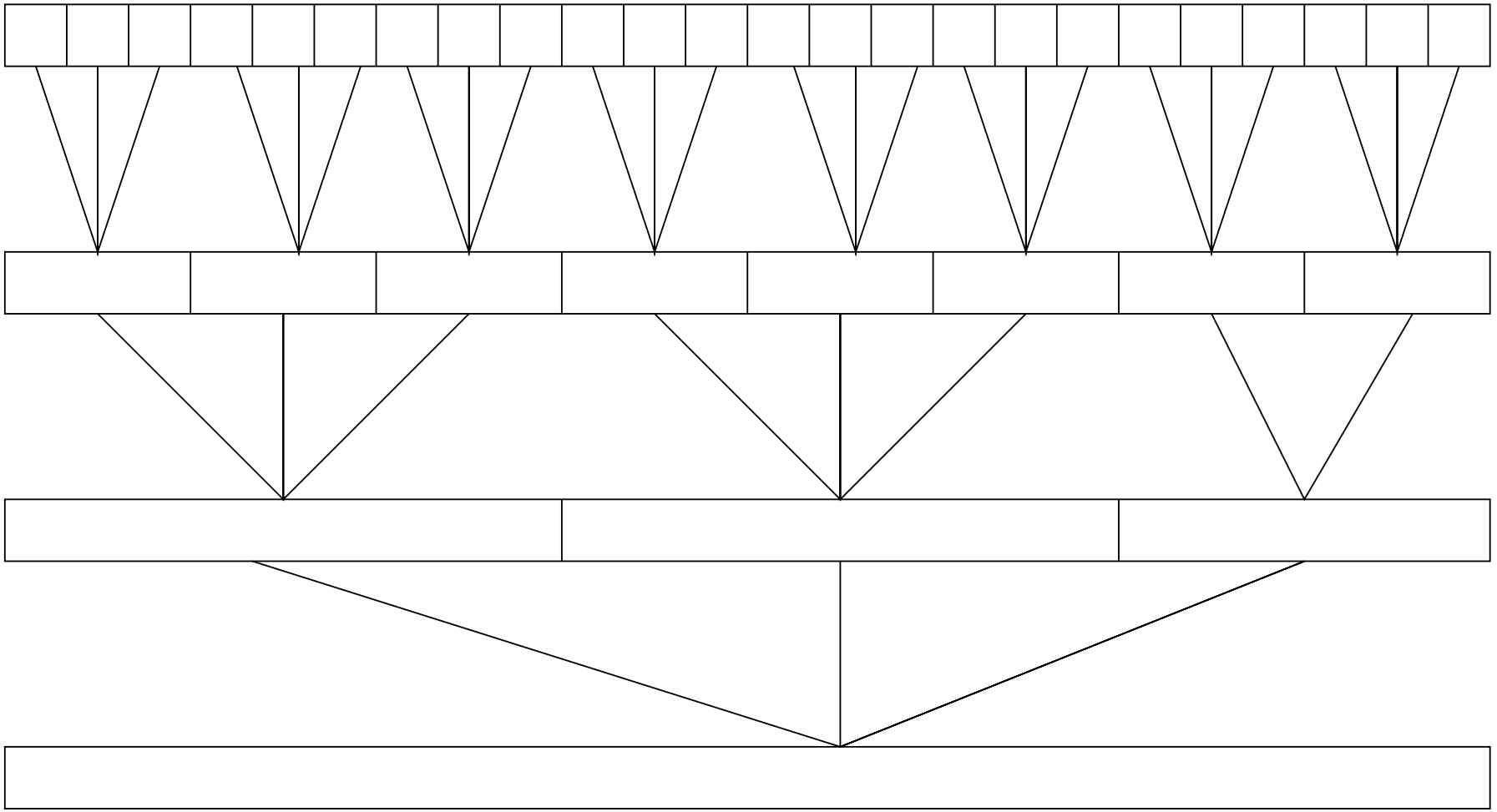


Figure: Packed Hilbert R-tree example with $M = 3$.