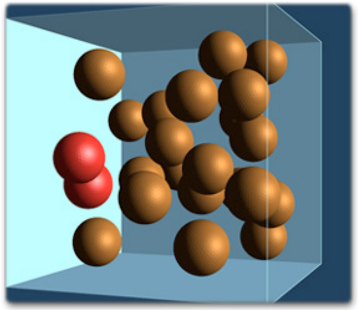
A Benchmarking Suite for Static Collision Detection Algorithms

René Weller
Clausthal University, Germany
weller@in.tu-clausthal.de

WSCG '07, January/February 2007, Plzen

Motivation

- Fast algorithms for collision detection between polygonal objects play an important role in many applications
  - Physically based simulations
  - Entertainment
  - Robotics



- Collision Detection is computational bottleneck

→ Essential to select fastest algorithm

Requirements

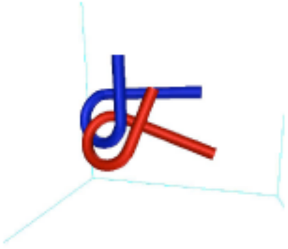
- Collision detection algorithms are sensitive to specific scenarios

  → Difficult to evaluate and compare

- Standardized benchmarking suite for collision detection should make fair comparisons between algorithms much easier
  - Broad spectrum of interesting contact scenarios
  - Easy to use
  - Flexible and robust

Related Work

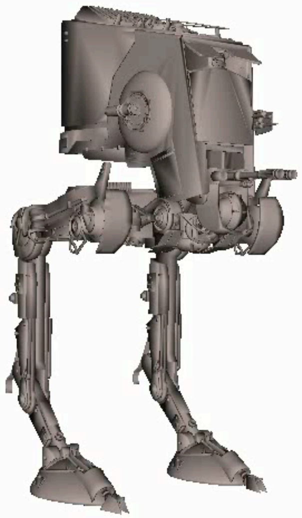
- One object rotates in several fixed distances to another object [Zachmann, 98]
  - objects penetrate heavily
- Benchmark with special focus on motion planning [Caselli et al., 02]
- Physically based simulation [Otaduy & Lin, 03]



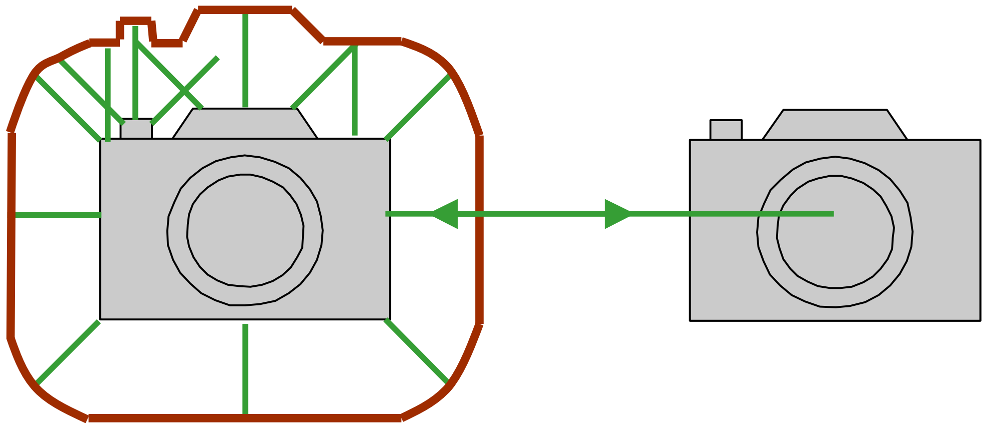
Our Approach

- Running time depends mainly on
  - object shapes
  - objects complexity
  - orientation
  - distance between the objects
- Test as many configurations for a given distance as possible
  - Use a set of different objects in several resolutions
  - Compute user specified number of configurations for a given distance

Benchmark@Work



Distance Computing: Offset Surface



Distance Computing: PQP

Sampling the Search Space

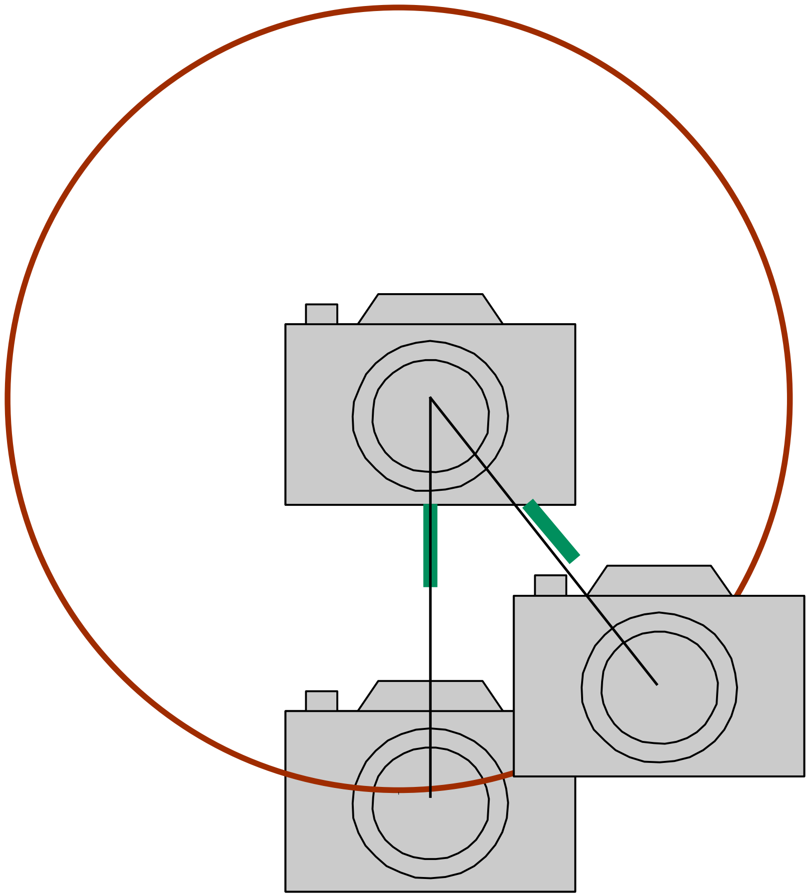
- 6D continuous search space is to big too be tested

  → Sampling

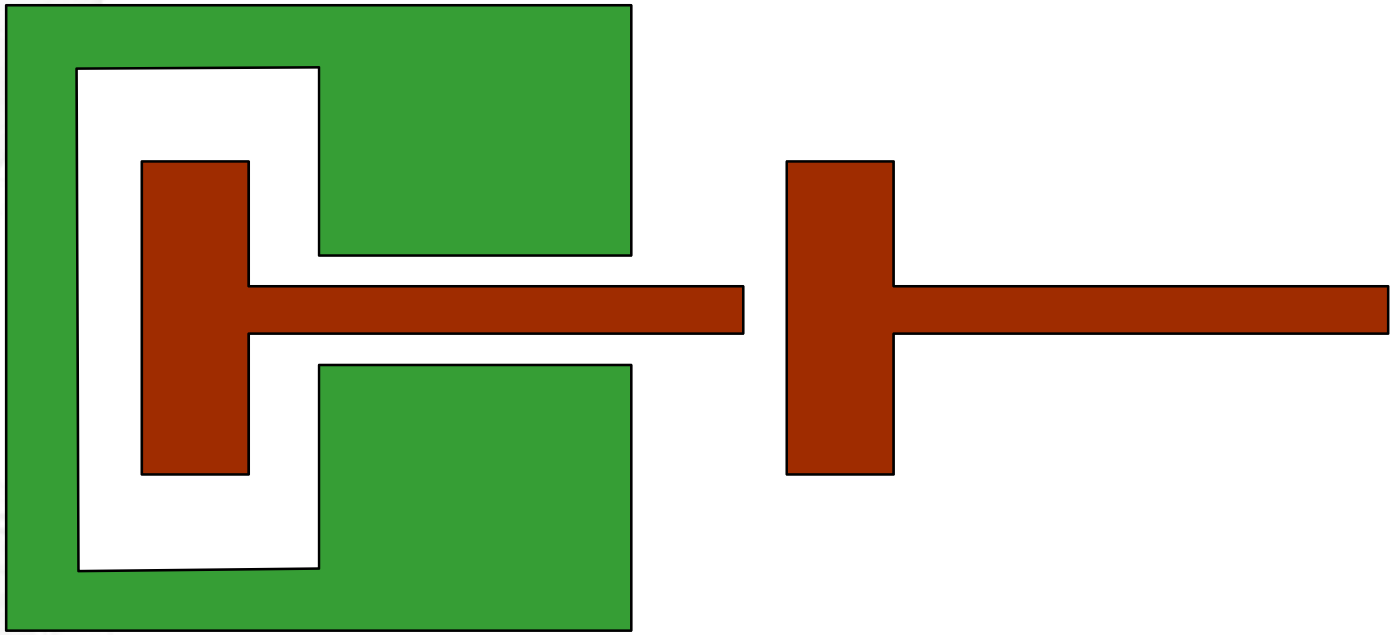
- Two Methods:
  - Grid Method
    - More Accurate
  - Sphere Method
    - Faster
- Main Loop:

```
while #Rotations < Required #Rotations
      Do method specific translations
      Rotate moving object
```

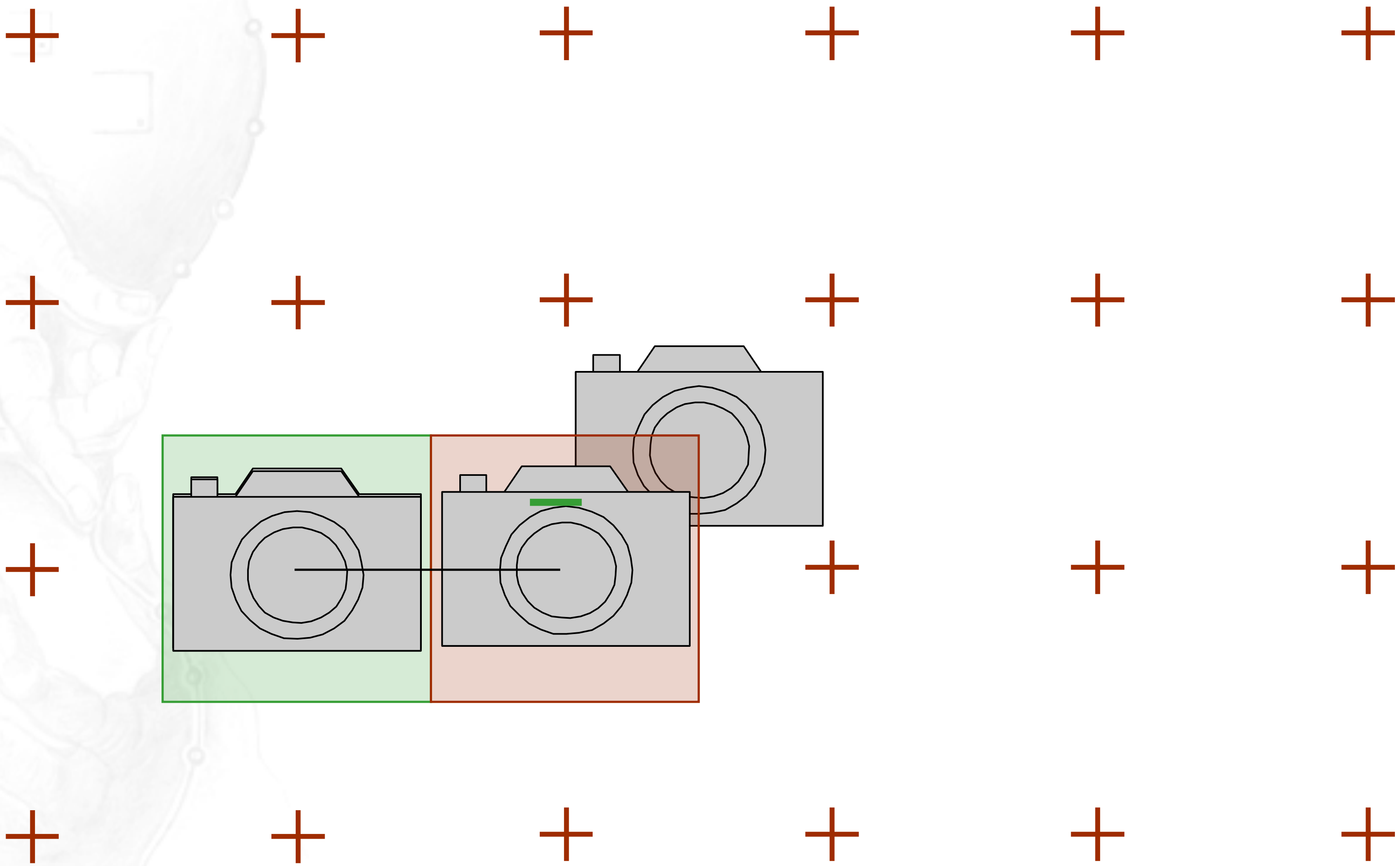
Sphere Method



Problems of the Sphere Method



Grid Method



Implementation

- OpenSG for object management
- Wrapper for several free available CD-libraries
- For configuration space exploration user has to specify:
  - Objects
  - Preferred Method
  - Grid size/ step size for spherical coordinate
  - Step size for rotation of the moving object
  - Set of distances
- Automatical generation of sample points and benchmark of all available algorithms with accuracy of 1 msec

Libraries for Collision Detection

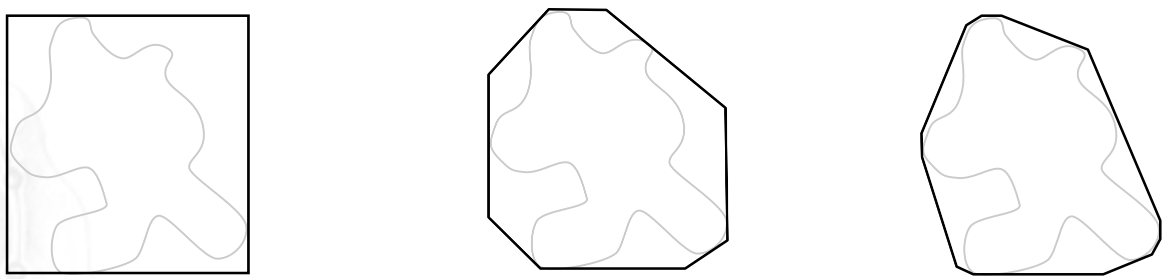
- VCollide
  - Interface for I-Collide and RAPID
  - Sweep-and-Prune (disabled)
  - OBBs
- PQP
  - Bases on RAPID (OBBs)
  - Supports Distance Computing
    - Swept spheres

Libraries for Collision Detection

- FreeSolid
  - AABBs
  - Could handle deformations
- Opcode
  - memory optimized AABBs (no-leaf-trees)
  - Uses primitive-BV-tests
- BoxTree
  - Memory optimized AABBs
  - 2 splitting planes instead of 6 extends
  - Supports grid and convex hull pre-tests for n-body simulations (disabled)

Libraries for Collision Detection

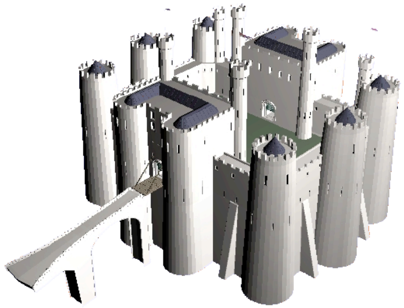
- Dop-Tree
  - Discretly oriented polytopes (k-Dops)
  - k = 24 for highest performance in most cases
  - Supports convex-hull pre-checks and grid for n-body simulation (disabled)

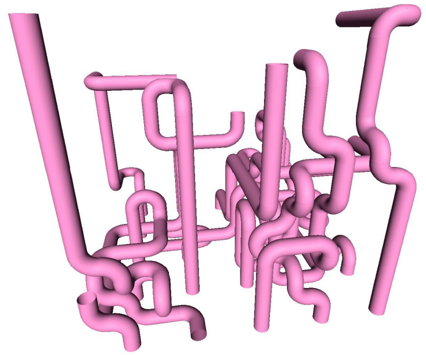


k=4 k=8 k=12

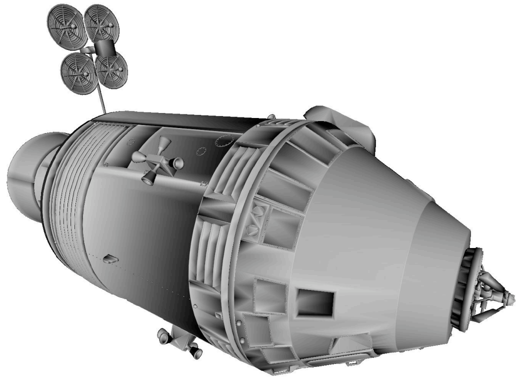
Test Cases

- 20 different kinds of objects in different resolutions
  - Helicopter, lustre, chair, castle, car models, space vehicles, different synthetic objects

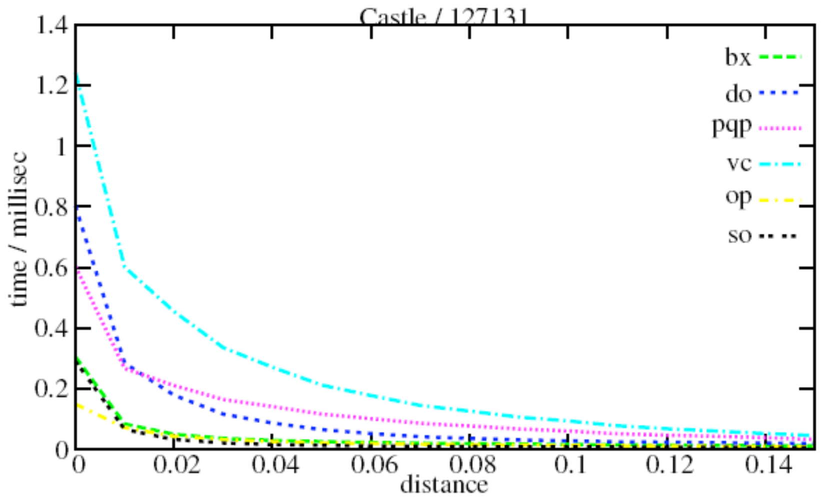


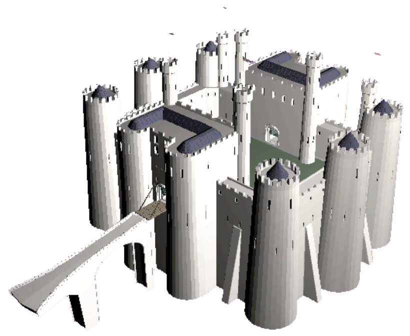


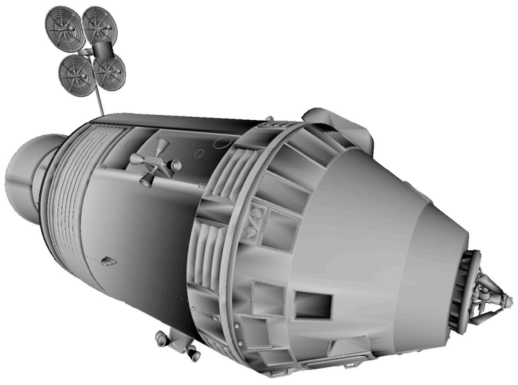
- Sphere method with PQP
  - 15° steps for spherical coordinates
  - 60° steps for rotations
- P4 3.0 Ghz, 1GB, gcc 4.0.2 on Linux

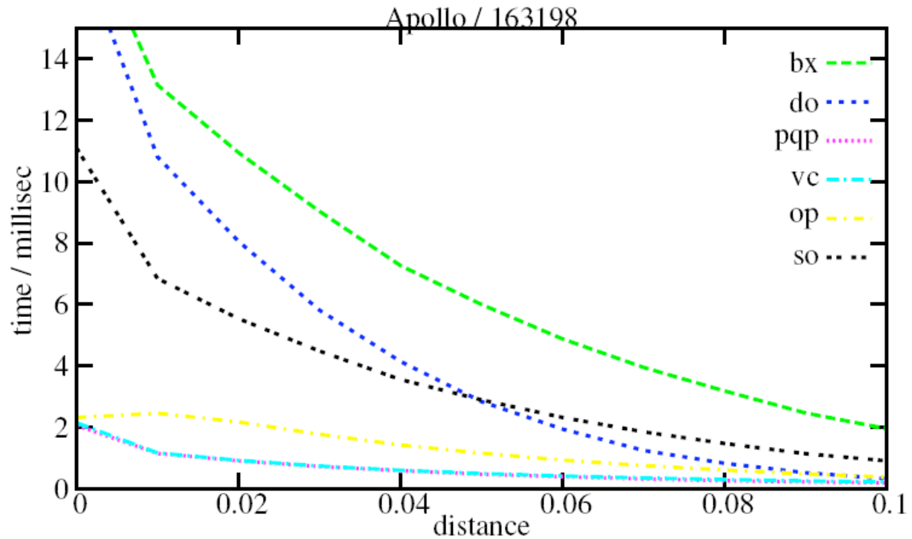


Results

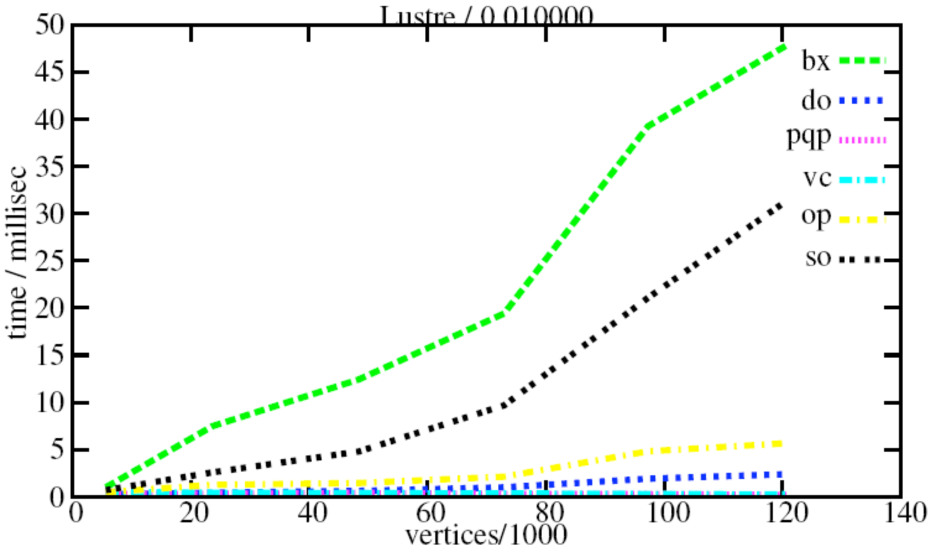

Castle / 127131
time / millisec
distance
bx
do
pqp
vc
op
so
0
0.2
0.4
0.6
0.8
1
1.2
1.4
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14



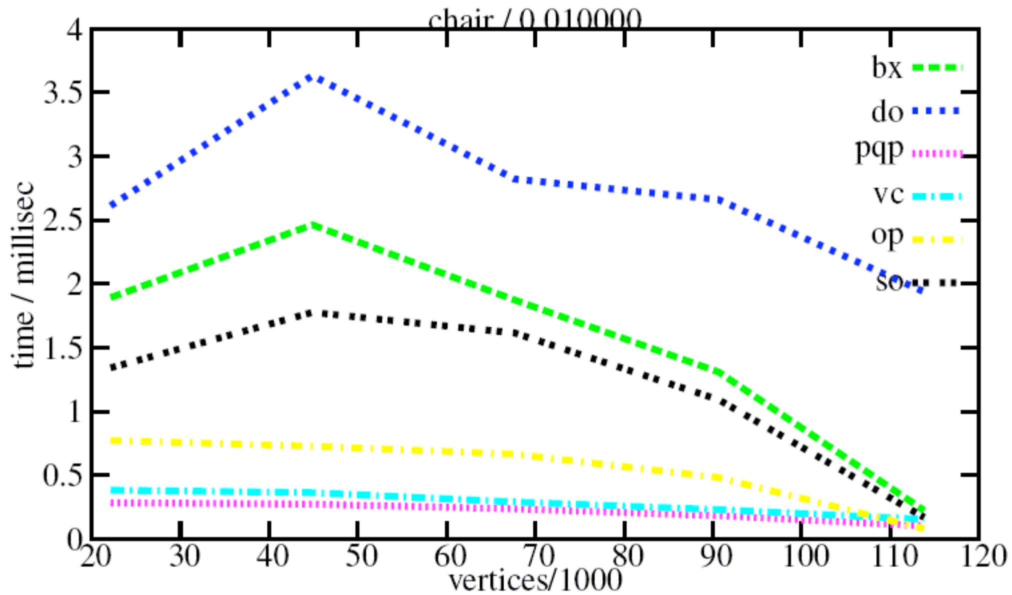



Apollo / 163198
time / millisec
distance
bx
do
pqp
vc
op
so
0
2
4
6
8
10
12
14
0
0.02
0.04
0.06
0.08
0.1

Results



Lustre / 0.010000
time / millisec
vertices/1000
bx
do
pqp
vc
op
so



chair / 0.010000
time / millisec
vertices/1000
bx
do
pqp
vc
op
so

Conclusions and Future Work

- Easy to use benchmarking suite and a set of representative objects for benchmarking CD algorithms for rigid objects
- Robust, fast, flexible and it is easy to integrate other libraries
- In the Future
  - Extend it for penetrating objects
  - Continuous collision detection
  - Deformable objects

Acknowledgements

- Gabriel Zachmann, Clausthal University of Technology
- DFG grant ZA 292/1-1 ("Aktionsplan Informatik")