Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2014)
J. Bender, C. Duriez, F. Jaillet, and G. Zachmann (Editors)

Massively-Parallel Proximity Queries for Point Clouds

Max Kaluschke[1], Uwe Zimmermann[2], Marinus Danzer[2], Gabriel Zachmann[1] and Rene Weller[1]

[1]University of Bremen, Germany [2]KUKA Laboratories, Germany

Abstract

We present a novel massively-parallel algorithm that allows real-time distance computations between arbitrary 3D objects and unstructured point cloud data. Our main application scenario is collision avoidance for robots in highly dynamic environments that are recorded via a Kinect, but our algorithm can be easily generalized for other applications such as virtual reality. Basically, we represent the 3D object by a bounding volume hierarchy, therefore we adopted the Inner Sphere Trees data structure, and we process all points of the point cloud in parallel using GPU optimized traversal algorithms. Additionally, all parallel threads share a common upper bound in the minimum distance, this leads to a very high culling efficiency. We implemented our algorithm using CUDA and the results show a real-time performance for online captured point clouds. Our algorithm outperforms previous CPU-based approaches by more than an order of magnitude.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality

1. Introduction

During the last years we observed that humans and robots move more and more to close ranks. For instance autonomous robotic vacuum cleaners have already entered our living rooms. In the future the importance of such tasks that unify human and robotic workspaces will increase significantly, not only for small service robots, but also in in healthcare, manufacturing, and everyday life. As a consequence, safety measures protecting humans will become paramount in the software of robots employed in those areas (which is, of course, not as critical in small service robots).

This means, unexpected collisions between humans and robots have to be avoided under all circumstances. This challenge can be solved by the design of the robotic manipulators and on appropriate development of robust collision avoidance methods. Actually, collision avoidance includes three major parts: the perception of the environment, the algorithmic detection of collisions based on environment information and finally, the appropriate control of the robot [FKLK12]. All those parts must be solved in real-time because people tend to behave unpredictably.

Several different sensor types have been proposed for sensing the environment, including monocular cameras [CDR*07], stereoscopic cameras [KA08], laser scanners [WGS04], time-of-flight cameras [PMR*08] and especially, Microsoft's inexpensive depth camera Kinect. All these sensor types record and output some kind of point cloud data, most often an pixel image with additional depth information for each pixel. On the other hand, most fast collision detection algorithms usually require a polygonal mesh representation of the objects to work properly.

Obviously, it is possible to simply reconstruct a mesh from the sensed point cloud and than use traditional collision detection methods to determine the minimum distances between the robot and its environment. Unfortunately, recent reconstruction methods are relatively slow and can be hardly processed in real-time for large point clouds. Additionally, almost all traditional collision detection algorithms rely on acceleration data structures like bounding volume hierarchies (BVHs) that have to be constructed in a time-consuming pre-processing step. These two limitations make it impossible to use the mesh reconstruction method under real-time constraints.

In this paper, we propose a new method that allows us to compute the minimum distance between a robot and its dynamic environment in real-time. In our approach, the robot is modelled by a polygonal mesh and the environment is represented by a point cloud like it is output by most sensor

DOI: 10.2312/vriphys.20141220

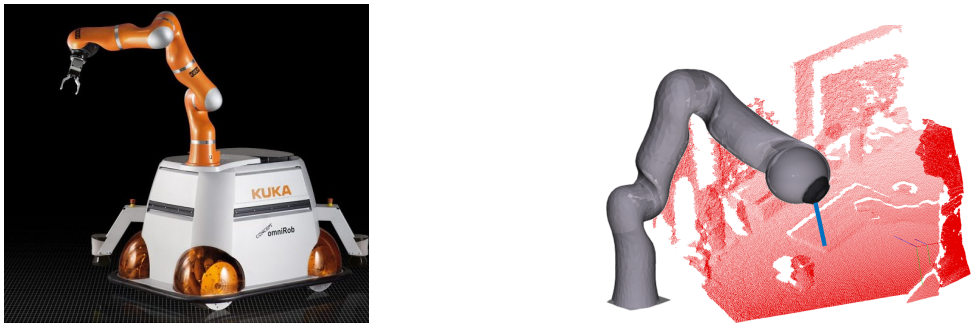

Figure 1: *Left: a KUKA Omnirob robot. Right: our scenario: a KUKA robot senses its environment via a Kinect and computes the minimum distance (blue line) to the point cloud.*

types. A new point cloud is recorded by the sensor in each frame. We compute a BVH for each part of the robot's geometry. More precisely, we adopted the Inner Sphere Trees data structure that has proven to be very fast for distance computations [WZ09], but our approach can be easily used with any other BVH data structure. The volumetric object representation of ISTs does not only support distance computations, but it also reports collisions if a point of the point cloud is located *inside* the object. However, we do not require any complicated data structures or pre-processing of the point cloud; we simply use it directly as it is output by the sensor.

In principle, we test the robot's BVHs against all points in the point cloud using traditional BVH-traversal methods. Such a point cloud usually consists of a large amount of points. For instance, a single Kinect image contains approximately 300.000 points. For a robot that is modelled of eight rigid parts this would result in 8×300.000 BVH traversals. This could be hardly processed sequentially on the CPU in real-time. Fortunately, the BVH can be almost trivially parallelized; all points can be tested individually, i.e. independent from all other points. In our massively parallel algorithm, all threads share additionally the same upper bound computed so far during the traversal. This results in a dramatically better culling efficiency than in a simple sequential computation.

One of the main purposes of this work is to explore the large design space on both the algorithmic and the implementation level, and to find the optimal solution. For instance, we applied an octree pre-filtering of the point cloud or we implemented a non-recursive version of the BVH-traversal that should be better suited for the GPU. Surprisingly, most of these "improvements" resulted in dead ends – they simply run slower than the naive approach. Nevertheless we include their description in our paper in order to prevent other researchers from running into the same dead ends.

Because of its structural simplicity, e.g. the lack of complicated data structures for the point cloud, our novel approach is simple to implement and it works very robustly. We tested it in a real-world scenario using a KUKA Omnirob: the Omnirob consists of a 7-DOF KUKA Light-Weight-Robot (LWR) mounted on an autonomously driving car that adds even more degrees of freedom to the platform. We mounted a Kinect on the end-effector to sense the environment. Our algorithm is able to answer distance queries in less than 3 msec for full resolution Kinect depth images and in 10 msec for point clouds consisting of 5M points. To our knowledge, this is the first description of an algorithm that can perform this task in real-time. However, our algorithm is not restricted to this robotic environment. Basically, the same problem appears also in virtual environments where depth sensors are used to track objects or users such as games or virtual reality applications.

2. Related Work

The topic of collision detection is an essential part in most interactive simulations and computer graphics and it has been extensively researched in the literature. Usually, 3D objects in these scenarios are represented by polygonal meshes. Hence, most work on collision detection has been spend to accelerate queries for this kind of object representation. Often, some kind of bounding volume hierarchy is used in order to early prune parts of the geometry that can not collide. Such hierarchies have been described for different bounding volumes that all have their unique strengths and weaknesses, including axis-aligned bounding box boxes [vdB98], orientated bounding boxes [GLM96], spheres [Hub93] or discrete oriented polytopes [ZL11]. Basically, these data structures are used for simple boolean collision queries, but they can be easily extended to compute minimum distances as well. Johnson et al [JC98] describes a generalized basic BVH-based distance computation in a framework for minimum distance computations.

There also exist more specialized data structures for distance computations: for instance the Linn-Canny algorithms

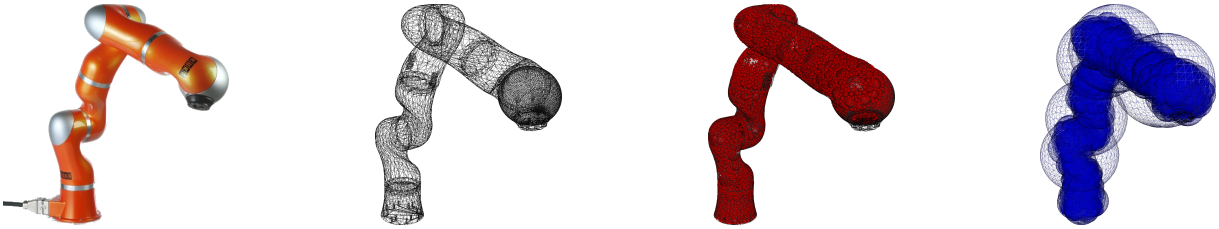

Figure 2: *Left: the real robot. Half left: the wireframe model of the robot. Half right: the model of the robot filled with spheres. Right: some hierarchy levels of the IST.*

performs a local search using pre-computed Voronoi diagrams [LC91] but it works only for convex polyhedra. Other approaches are able to additionally provide a lower-bound, like the spherical sector representation presented by Bonner et al [BK88], or the inner-outer ellipsoids [JLS*01,LKC06]. Another alternative for distance computations are distance fields [FSG03] that can be also combined with BVHs [FUF06].

However, all these approaches were designed for sequential processors. Implementations that use parallel CPU instructions like OpenMP [ZpTL08] or SSE give considerable speedups of around 2.7 compared to sequential algorithms, but there is more potential in modern GPUs. For triangle mesh representations there already exist a few approaches that make use of massively parallel processing of GPUs. For example [Kar12] used the graphics card for collision detection between multiple objects with a single common object. Lauterbach et al [LMM10] implemented a distance computation using OBB trees on the GPU. Some methods have been described that do not require BVHs: for instance Faure et al [FBAF08] used layered depth images, Mainzer and Zachmann [MZ13] proposed a parallel sweep-and-prune approach and Weller et al [WFZ13] showed an approach that is based on hierarchical grids.

Compared to mesh representations, the literature on collision detection for point clouds is relatively sparse. One of the first approaches to detect collision between point clouds was developed by [KZ04]. They use a bounding volume hierarchy in combination with a sphere covering of parts of the surface. [KZ05] proposed an interpolation search approach of the two implicit functions in a proximity graph in combination with randomized sampling. [EFGS07] support only collisions between a single point probe and a point cloud. For this, they fill the gaps surrounding the points with AABBs and use an octree for further acceleration. [FOAM10] used R-trees, a hierarchical data structure that stores geometric objects with intervals in several dimensions [Gut84], in combination with a grid for the broad phase. [PCM11] described a stochastic traversal of a bounding volume hierarchy. By using machine learning techniques, their approach is also able to handle noisy point clouds. In addition to simple collision tests, they support the computation of minimum distances [PCM12].

Also some methods for *online collision avoidance* in robotics has been described. Some authors simply include a high number of additional sensors like infra red or ultrasound to the robots or the environment. These sensors have a limited range of view or produce only coarse data but their combined output can be used to avoid collisions with abruptly popping up objects [HG05]. Other works use neural networks [BJ11], behavioural bayesian networks [YBOB11] or optical flow algorithms for sequences of images [LW05] that can be further improved by also including depth images [RTT09]. [KH07] introduced the idea to compute distances directly from single images of the environment using computer-vision classification techniques. However, they did not include depth values.

Especially the release of Microsofts inexpensive depth camera Kinect inspired many researchers to new online collision avoidance algorithms that work directly on the depth image. For example, [BV12] proposed an error minimization method providing real-time robot pose estimation. However, their approach is restricted to ground robots moving in a 2D space. Also [BMR*10] represented the robot only by a single point in order to simplify the distance computation. [SBF09] compared the obstacle and the robot depth maps by an image plane projection in 3D. The approach that is closely related to our method was presented by [FKLK12]. They also use a KUKA Light-Weight-Robot and a Kinect for the data retrieval. Their primary focus is the computation of the collision responses based on distances and velocities and less the acceleration of the distance queries. Actually, the distance computation is derived from a simple spherical approximation of the robot's surface. But, they do not describe any acceleration data structures for the distance queries. Like us, [PSCM13] used a BVH representation of the robot's geometry. They applied a pre-filtering of the point cloud that is based on an octree to reduce the number of BVH-point tests. However, the construction of the octree is not included in the timings their primary application was offline path-planning for static scenes rather than online distance computation for dynamic scenes. Actually, we have presented a similar octree-based algorithm before [Wel12]. Our experi-

ments have shown that an online construction of an octree is too slow for real-time collision avoidance.

3. Our Approach

Algorithm 1: getMinDist(Set of ISTs R, point cloud P)

```
minDistance = maximum float
forall the ISTs r ∈ R do
    forall the Points p ∈ P do
        distance = traverseIST(Root sphere of r, p,
        minDistance)
        if distance < minDistance then
            minDistance = distance
```

In this Section we will describe the basic mechanisms of our new distance computation scheme. In our scenario, we assume that the robot is represented by a polygonal mesh. The robot can consist of different rigid parts that can be moved using arbitrary rigid object transformations. However, we do not use the actual surface representation of the rigid parts for our distance computation, but we adopted the Inner Sphere Tree (IST) data structure [WZ09]. ISTs have proven to be extremely fast for distance queries. Moreover, they are independent of the object's complexity (e.g. its polygon count) and they can be computed for almost all surface representations, including polygonal meshes, CSG, NURBS, etc. The ISTs gain their efficiency from filling the objects' interior with sets of *non-overlapping* spheres instead of using its surface. These *inner spheres* are used to construct a traditional bounding volume hierarchy. The only difference to BVHs that are based on the object's polygons is, that the leaves consists of the inner spheres instead of the polygons. Actually, our algorithm can be easily extended to all other types of BVHs, it does not rely on any special features of this inner BVH.

As mentioned in the introduction, the environment is represented by a point cloud. This point cloud can be directly derived from a Kinect input image. Basically, such a point cloud is simply a set of points P where each point has a 3D coordinate in space. The goal is to find that point in P that is closest to any part of the robot.

3.1. Basic Concept

In order to describe the basic concept, we start with a short description of a sequential version of our algorithm.

We simply compute a minimum distance of each IST with each point in the point cloud. We are only interested in the global minimum distance. Hence, we can accelerate the algorithm slightly by additionally passing the minimum distance computed so far to the traversal. This allows us to interrupt the traversal when the distance of a hierarchy sphere is larger than this value (See Algorithm 1).

The sequential traversal can be done as with a traditional recursive BVH traversal algorithm: we check whether two bounding volumes overlap or not. If this is the case, we recursively step to their children. In order to compute upper bounds for the minimum distance, we simply have to add an appropriate distance test at the right place. This has to be done, when we reach a pair of inner spheres (i.e. the leaves of the ISTs) during traversal. During traversal there is no need to visit branches of the bounding volume test tree that are farther apart than the current minimum distance because of the bounding property. This guarantees a high culling efficiency (See Algorithm 2).

Algorithm 2: traverseIST(Sphere s, point p, minDist, d)

```
if s is Leaf then
    return d
forall the Children s_c of s do
    d = distance( s_c, p )
    if d < minDist then
        minDist = traverseIST( s_c, p, minDist, d )
return minDist
```

3.2. Parallelization

In principle, the parallelization of Algorithm 1 is straight forward: we can simply process all parts of the robot and all points in parallel because the traversals are almost independent of each other. If we would use this naive implementation, we would compute an individual minimum distance for each part of the robot and each point. In case of a Kinect depth map and a robot that is constructed by seven parts this would result in 8×300.000 local minima. In order to get the global minimum we would have to search for it in all these local minima. In other words, we would lose the possibility to early prune points that are far away. This results in a high number of unnecessary computations.

Consequently, it is better to keep a common *global* minimum that is shared by all threads (See Algorithm 3).

During the traversal we have to be careful when updating the global minimum because concurrent threads could try to write it simultaneously. This would result in typical race conditions that appear often in massively parallel algorithms. Consequently, we have to restrict the writing access.

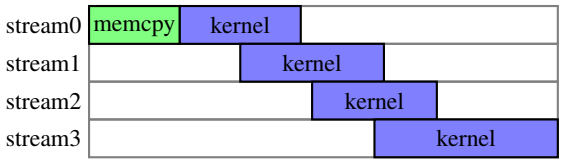

Figure 3: *Task scheduling on the GPU when data dependency is specified using streams.*

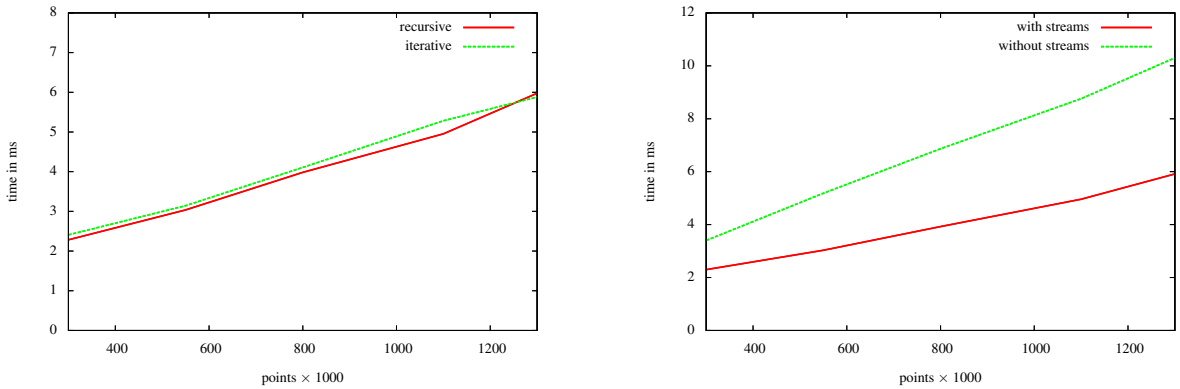

Figure 4: *Left: timings for the two different traversal algorithms. The iterative traversal is slower, except for very large point clouds. Right: timings for CUDA streams and sequential processing of the BVHs. Streams add a significant performance boost.*

Algorithm 3: getMinDist(Set of ISTs R, point cloud P)

```
minGlobalDistance = maximum float
In parallel forall the ISTs r ∈ R do
    In parallel forall the Points p ∈ P do
        traverseIST(Root sphere of r, p)
```

Basically, an atomic compare-and-swap (CAS) in the traversal algorithm solves this challenge. If we simply adopt the recursive traversal of the ISTs this results in the following algorithm:

Algorithm 4: traverseIST(Sphere s, point p, distance d)

```
if s is Leaf then
    atomicMin( d, minGlobalDistance )
forall the Children s_c of s do
    d = distance( s_c, p )
    if d < minGlobalDistance then
        traverseIST( s_c, p, d )
```

4. Implementation

The basic parallel implementation is relatively straight forward. However, current GPUs have their limitations with respect to recursion or memory management. Therefore, we tested different implementation details and compared their effects on the performance.

For instance, in the algorithms from the previous section we assume that the global minimum distance resides in global GPU memory. In current GPUs the access to global memory is known to be relatively slow. Moreover, GPUs are not optimized for recursion. We will discuss these issues in this section.

4.1. Recursion

For a long time GPUs were not able to perform recursive algorithms. When recursion was first introduced, it was extremely slow, because recursion introduces additional execution divergence. To overcome this limitation, we additionally implemented an iterative traversal algorithm for the BVH. It is based on a stack data structure (See Algorithm 5).

Algorithm 5: traverseIST(Tree t, Point p)

```
Stack s
s.push(root of t)
while s not emtpy do
    Sphere s = s.pop()
    forall the Children s_i of s do
        d = distance( s, p )
        if d < minGlobalDistance then
            if s_i is leaf then
                atomicMin( d, minGlobalDistance )
            else
                s.push(s_i)
```

The concept to manage an explicit stack to compensate for recursion was inspired by [Kar12]. However, our experiments have shown that this limitation does not seem to hold anymore. Actually, our recursive implementation performs best (See Figure 4).

4.2. Streams

In our scenario, a robot consists of several rigid parts. Each part has its own transformation matrix to the world coord system. Obviously, we have either to transform the points of the point cloud into the robot's part coord system or vice versa. In each thread, we test a single point against the whole BVH. Consequently it would be extremely inefficient

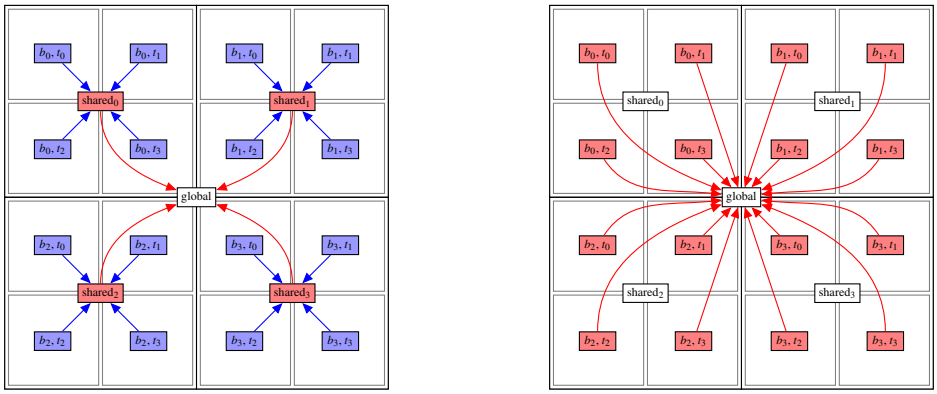

Figure 5: *Left: hierarchical minimum distance computation using shared memory and local memory. Right: a single global variable in global memory stores the global minimum distance.*

to transform the bounding volumes during the traversal into the point's coord system instead of simply transforming the point only once into the BVH's system. However, we could translate the BVHs in a pre-processing step once, before we start the tests. Unfortunately, this would require additional memory to store the transformed BVHs. Consequently, we decided to simply transform the points into the BVH's coord system.

However, this strategy makes the implementation of Algorithm 3 complicated because if we process all parts in parallel in a single kernel call, we would have to include all transformation matrices for all parts. First we thought, that a sequential processing of the parts would not affect the performance significantly. Surprisingly, it turned out that this is not true. Therefore, we decided to use CUDA streams to launch concurrent kernel for the parts as illustrated in Figure 3. This results in a huge performance boost of up to 50% (See Figure 4).

Since the individual robot parts are rigid, the geometry does not change. The bounding volume hierarchies are precomputed and all the BVHs can be transferred to the GPU memory at the start of the application. The movement of the robot is represented solely by the transformation matrix of each rigid part. In contrast, the point cloud that represents the surroundings changes each frame. Consequently, each point needs to be transferred to the memory of the GPU on-the-fly prior to the distance query. In both cases we store the data in global memory, since the amount of data is too large to store in constant memory. However, for each kernel call, the arguments are automatically transferred to constant memory by CUDA, so that consecutive accesses can be cached. By launching kernels concurrently, we maximize the occupancy of the GPU, explaining the huge increase in performance.

These streams also have another advantage: if we combine our algorithm with pre-filtering techniques like an octree, each part of the robot will get an individual set of pre-filtered points from the point cloud. In this case, streaming allows a much more efficient memory transfer from the CPU to the GPU memory. This comes from the fact that streams allow us to express data dependency, such that early finishing data transfers allow certain kernels to start their execution as soon as their data is copied onto GPU memory. Moreover, this implementation scales perfectly with more powerful hardware like multiple GPU setups.

4.3. Fast memory access vs. global communication

In our traversal algorithms we use a global variable in order to communicate the current minimal distance. Global memory access is about 100 times slower than accessing shared memory [Har13]. Hence, we saw a potential improvement here.

In order to make better use of CUDAs memory design, we added a solution that relies on the faster *shared memory* access of CUDA. Basically, shared memory is kind of a user- managed cache. Unlike global memory, however, shared memory is accessible only by one block of threads at a time. † Basically, we simply store an individual local minimum for each such block. When all BVH traversals of all blocks are finished, we combine these local minima to the global minimum. Figure 5 illustrates the idea. Consequently, the potential global accesses will be reduced to the amount of blocks b. In the naive solution, it is $t \cdot b$ with t being the amount of threads in a block.

Surprisingly, our results show a performance loss of approximately 30% if we apply this strategy, compared to simple global memory access (See Figure 7). The reading operations during the traversal seem to be cached and the number of atomic writing operations is relatively low in our distance

† The set of threads launched by one kernel are partitioned into a set of blocks.

Figure 6: *Left: test scene from real kinect images (first scenario). Right: artificial test scene (second scenario).*

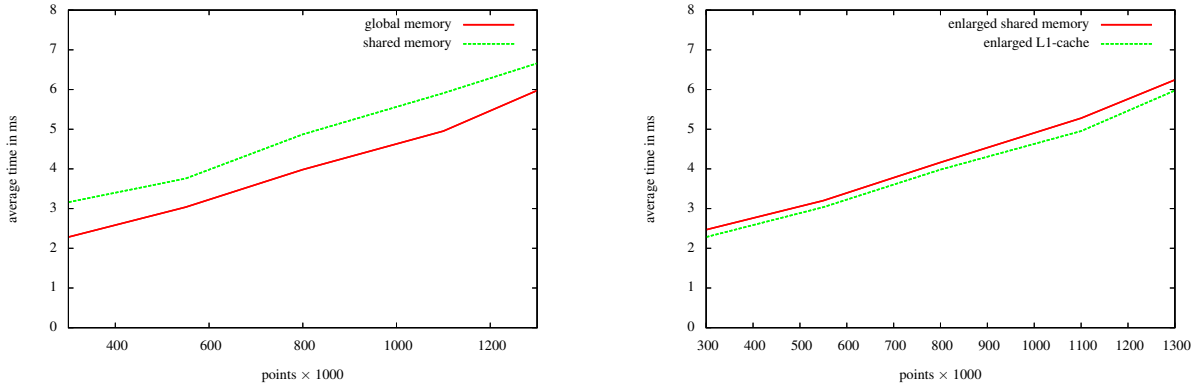

Figure 7: *Left: timings for the shared memory approach and the global memory access. The shared memory strategy is significantly slower in all cases. Right: timings with en- and disabled shared memory caching. Enabling caching results in a 7% performance gain.*

computations. Moreover, the local minima result in worse upper bounds of the distance and hence, also on more traversals of the BVH.

4.4. Cache configuration

Due to the performance loss when using our shared-memory-strategy described above, our implementation does not use any shared memory. However, CUDA offers the ability to use these hardware resources in different ways.
By enabling call $cudaDeviceSetCacheConfig(PreferL1)$ we allow the GPU to use a chunk of shared memory for additional caching This optimizes the access of both, global and local memory, which we utilize in our algorithm. Our timings show a performance gain of around 7% by adding just a single line of code (See Figure 7).

5. Results

We implemented all approaches in a proof-of-concept application. All tests were performed on an Intel Core i7 CPU with 4GB main memory and a NVIDIA Gefore GTX 780 GPU with 2GB of memory.

We used two different test scenarios to test our algorithm. However, the basic test setup was the same in both scenarios: We use a model of a 7-DOF KUKA Light-Weight-Robot (LWR) to represent the geometric model. The robot consists of seven rigid parts and consequently we got seven ISTs, one for each of the robot's parts. The total triangle count of the model was 12k and the number of inner spheres was 15k. Actually, the ISTs compute only an approximation of the distance. In our results we recognized an distance error in the range less than floating point accuracy. This is much smaller than accuracy of the depth maps generated by a Kinect.

In our first scenario, the robot was placed in front of a workspace with boxes and tools laying on a table (See Fig. 6). In order to guarantee a fair comparison between different algorithms we did not use live recordings from a Kinect for the timings but we used pre-recorded paths of the robot and also pre-recorded point clouds. The data was recorded by a Kinect mounted on the end effector of the real robot using OpenNI and the *Point Cloud Library (PCL)* to generate 3D point clouds from the depth images delivered by OpenNI. Additionally, we merged several of these pre-recorded point clouds to a larger point cloud. This allows a more detailed map of the environment that can be applied to path-planning tasks and it shows the scalability of our algorithm. Please note, that we did not require any registration algorithm because the KUKA LWR knows its position and orientation from sensor data.

In our second scenario we used artificial point clouds gen-

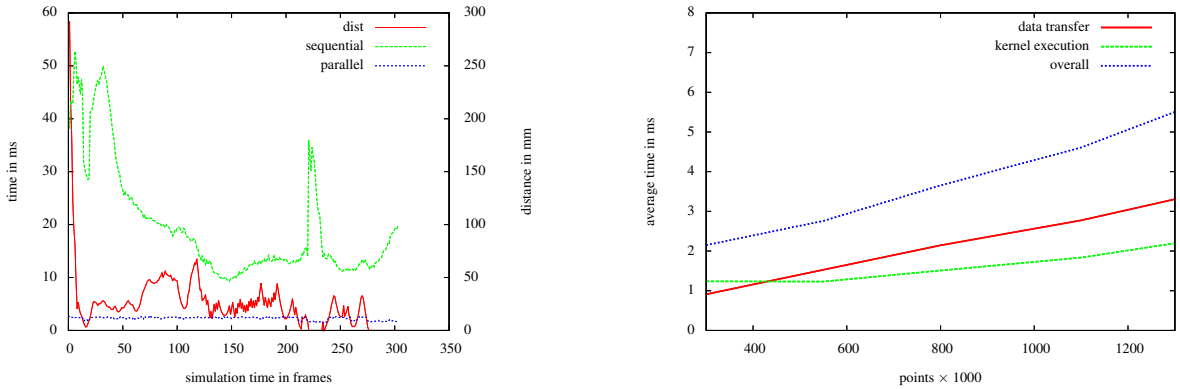

Figure 8: *Left: timings for a single test run with 600k points in the point cloud for our massively parallel algorithm and the sequential octree pre-filtering. The distance computation is almost independent of the actual distance for our approach. In case of large distances, the time for the octree rises. Right: individual timings of data transfer to the GPU memory and pure execution of the kernel, illustrating data transfer being the bottleneck.*

erated from a 3D scene to stress our algorithm (See Fig. 6). The artificial scene allows us to generate larger point clouds of up to 5M points. Moreover, we have a large environment - a whole room - with larger distances than in the first scenario. The resulting point clouds have very different conditions compared to the noisy sensor data from a kinect.

In order to compare the performance of our algorithm, we implemented also some competitors: first, we used a pure CPU version of our approach. The IST implementation for the CPU is hand optimized to support modern SIMD acceleration. Additionally, we implemented a pre-filtering based on an octree. This method is similar to that described by Pan et al [PSCM13]. The only difference is that we used ISTs as BVH.

The results show that our new massively parallel algorithm outperforms all competitors significantly (See Figure 9). For instance, in the first scenario, it is more than two orders of magnitude faster than the pure CPU version that requires up to 1 sec to compute a single distance for a point cloud with 1M points. Our algorithm finds minimum distances even for such large point clouds in less than 10 msec. This factor of more than 100 can not be explained solely by the higher computational power. Additionally, the GPU version provides a higher culling efficiency due to the globally shared upper bound on the minimum distance.

In order to reduce the number of points to tests for the CPU we used an octree for pre-filtering. However, the pure distance calculation for the octree takes already up to 20 msec in average, not including the construction. In our scenario, a new octree has to be computed in each frame because the point cloud changes continuously. If we additionally include this construction time of the octree, our algorithm is more than an order of magnitude faster while the octree is not applicable to real-time scenarios any more (See Fig. 9).

Almost half of the time in our approach is spent on copying the point cloud from CPU memory to GPU memory (See Figure 8). A combination of the octree pre-filtering, that should reduce the number of points to copy to the GPU, and our massively parallel test seems to be an interesting idea. We tested this setup, but our simple non-filtered algorithm is still much faster because most of the time is consumed by the CPU filtering of the point cloud (See Fig. 8).

We believe that a much coarser filtering, that is extremely fast in consequence, would be much better suited for our parallel traversal. Of course, this task should be performed in parallel on the GPU. Another advantage of our algorithm is its independence of the actual distance. Figure 8 shows the timings during a single test run and the corresponding distances. The timings do not change very much as the distance varies. For the octree, the running time increases with increasing distance because of a reduced culling efficiency.

All these observations also hold for the second scenario, a worst-case scenario for our algorithm. It is still faster than the octree, even in very large scenes with up to 5M points. Due to the very even distribution of points in space this scenario is better suited for spatial culling using the octree. However, if we take the construction time into account, our algorithm is still an order of magnitude faster.

6. Conclusion & Future Works

We presented a novel massively-parallel algorithm for real-time collision avoidance of autonomous robot navigation in highly dynamic environments. We extended a typical BVH, the Inner Sphere Trees, to distance computations with point

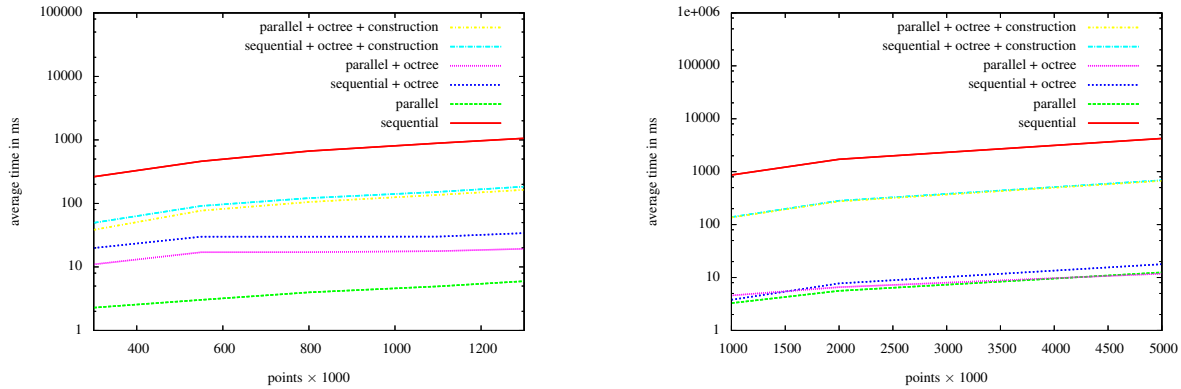

Figure 9: *Left: results from our timing. Our new massively parallel algorithm outperforms all competing approaches by more than an order of magnitude. Please note the logarithmic scaling. Right: timings with point clouds generated from a 3d-mesh. All GPU timings include the transfer of data to GPU memory.*

cloud data that was captured online via a Kinect. Our algorithm is very simple to implement and it can be easily adapted to other BVHs. We implemented our algorithm using CUDA and the results show a real-time performance even for very large point clouds. Moreover, our algorithm is more than an order of magnitude faster than previous CPU-based approaches. To our best knowledge, therefore our algorithm is the first to compute this kind of task in real-time. In addition, we have explored a large number of variations and optimization of both the algorithm and the data structures, thus exploring the design space of the algorithm to a large extent. This provides important insights into what does and, at least as important, what does not increase the overall performance.

However, there are still avenues for future improvements. At the moment, we use our collision detection algorithm only for collision avoidance between the robot and the environment. A better performance would also allow path planning directly on the point cloud data. This offers several challenges for future works: for instance, we need an additional representation of the objects' volumes, instead of only their surface. Probably, a real-time version of the sphere packing algorithms could produce relief. Moreover, we plan to apply our algorithm also to other scenarios that are based on real-time point cloud data like tracking problems in virtual reality systems. For instance, a slight optimization would be able to enable haptic interactions with online captured point cloud data with full 1000 Hz.

Acknowledgement

This work was partially supported by DFG grant TRR 8/3-2013.

References

[BJ11] BENAVIDEZ P., JAMSHIDI M.: Mobile robot navigation and target tracking system. In *IEEE International Conference on System of Systems Engineering* (2011). 3

[BK88] BONNER S., KELLEY R. B.: A representation scheme for rapid 3-d collision detection. In *IEEE International Symposium on Intelligent Control* (1988), pp. 320–325. 3

[BMR*10] BASCETTA L., MAGNANI G., ROCCO P., MIGLIORINI R., PELAGATTI M.: Anti-collision systems for robotic applications based on laser time-of-flight sensors. In *Advanced Intelligent Mechatronics (AIM), 2010 IEEE/ASME International Conference on* (july 2010), pp. 278 –284. 3

[BV12] BISWAS J., VELOSO M. M.: Depth camera based indoor mobile robot localization and navigation. In *ICRA* (2012), pp. 1697–1702. 3

[CDR*07] CLEMENTE L. A., DAVISON A. J., REID I. D., NEIRA J., TARDOS J. D.: Mapping large loops with a single hand-held camera. In *Robotics: Science and Systems* (2007), Burgard W., Brock O., Stachniss C., (Eds.), The MIT Press. 1

[EFGS07] EL-FAR N. R., GEORGANAS N. D., SADDIK A. E.: Collision detection and force response in highly-detailed point-based hapto-visual virtual environments. In *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications* (Washington, DC, USA, 2007), DS-RT '07, IEEE Computer Society, pp. 15–22. 3

[FBAF08] FAURE F., BARBIER S., ALLARD J., FALIPOU F.: Image-based collision detection and response between arbitrary volumetric objects. In *ACM Siggraph/Eurographics Symposium on Computer Animation, SCA 2008, July, 2008* (Dublin, Irlande, July 2008). 3

[FKLK12] FLACCO F., KRÖGER T., LUCA A. D., KHATIB O.: Depth space approach to human-robot collision avoidance. In *ICRA* (2012), IEEE, pp. 338–345. 1, 3

[FOAM10] FIGUEIREDO M., OLIVEIRA J., ARAUJO B., MADEIRAS J.: An efficient collision detection algorithm for point cloud models. In *Proceedings of Graphicon* (2010). 3

[FSG03] FUHRMANN A., SOBOTKA G., GROSS C.: Distance fields for rapid collision detection in physically based modeling. *Proceedings of GraphiCon 2003* (2003), 58–65. 3

[FUF06] FUNFZIG C., ULLRICH T., FELLNER D. W.: Hierarchical spherical distance fields for collision detection. *IEEE Comput. Graph. Appl. 26*, 1 (Jan. 2006), 64–74. 3

[GLM96] GOTTSCHALK S., LIN M. C., MANOCHA D.: Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 171–180. 2

[Gut84] GUTTMAN A.: R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec. 14*, 2 (June 1984), 47–57. 3

[Har13] HARRIS M.: Using shared memory in cuda c/c++, Jan. 2013. 6

[HG05] HU H., GAN J. Q.: Sensors and data fusion algorithms in mobile robotics, 2005. 3

[Hub93] HUBBARD P. M.: Interactive collision detection. *1993 (4th) International Conference on Computer Vision* (1993), 24–31. 2

[JC98] JOHNSON D. E., COHEN E.: A framework for efficient minimum distance computations. In *Proc. IEEE Intl. Conf. Robotics and Automation* (1998), pp. 3678–3684. 2

[JLS*01] JU M.-Y., LIU J.-S., SHIANG S.-P., CHIEN Y.-R., HWANG K.-S., LEE W.-C.: Fast and accurate collision detection based on enclosed ellipsoid. *Robotica 19*, 4 (July 2001), 381–394. 3

[KA08] KONOLIGE K., AGRAWAL M.: Frameslam: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics 24*, 5 (2008), 1066–1077. 1

[Kar12] KARRAS T.: Thinking parallel, part ii: Tree traversal on the gpu, Nov. 2012. 3, 5

[KH07] KUHN S., HENRICH D.: Fast vision-based minimum distance determination between known and unknown objects. In *IEEE International Conference on Intelligent Robots and Systems, San Diego/USA* (2007). 3

[KZ04] KLEIN J., ZACHMANN G.: Point cloud collision detection. In *Computer Graphics forum (Proc. EUROGRAPHICS)* (Grenoble, France, Aug 30 – Sep 3 2004), Cani M.-P., Slater M., (Eds.), vol. 23, pp. 567–576. 3

[KZ05] KLEIN J., ZACHMANN G.: Interpolation search for point cloud intersection. In *Proc. of WSCG 2005* (University of West Bohemia, Plzen, Czech Republic, Jan.31 – Feb.7 2005), pp. 163–170. 3

[LC91] LIN M. C., CANNY J. F.: A fast algorithm for incremental distance calculation. In *In IEEE International Conference on Robotics and Automation* (1991), pp. 1008–1014. 3

[LKC06] LIU J.-S., KAO J.-I., CHANG Y.-Z.: Collision detection of deformable polyhedral objects via inner-outer ellipsoids. In *IROS* (2006), IEEE, pp. 5600–5605. 3

[LMM10] LAUTERBACH C., MO Q., MANOCHA D.: gproximity: Hierarchical gpu-based operations for collision and distance queries. *Comput. Graph. Forum 29*, 2 (2010), 419–428. 3

[LW05] LOW T., WYETH G.: Obstacle detection using optical flow. In *in Proceedings of the 2005 Australasian Conf. on Robotics & Automation* (2005). 3

[MZ13] MAINZER D., ZACHMANN G.: Cdfc: Collision detection based on fuzzy clustering for deformable objects on gpuâĂŹs. In *WSCG 2013 - POSTER Proceedings* (Plzen, Czech Republic, 7 2013), vol. 21, pp. 5–8. Poster. 3

[PCM11] PAN J., CHITTA S., MANOCHA D.: Probabilistic collision detection between noisy point clouds using robust classification. In *International Symposium on Robotics Research* (Flagstaff, Arizona, 08/2011 2011). 3

[PCM12] PAN J., CHITTA S., MANOCHA D.: Proximity computations between noisy point clouds using robust classification. In *RGB-D: Advanced Reasoning with Depth Cameras* (Los Angeles, California, 06/2012 2012). 3

[PMR*08] PRUSAK A., MELNYCHUK O., ROTH H., SCHILLER I., KOCH R.: Pose estimation and map building with a time-of-flight camera for robot navigation. *Int. J. Intell. Syst. Technol. Appl. 5*, 3/4 (Nov. 2008), 355–364. 1

[PSCM13] PAN J., SUCAN I. A., CHITTA S., MANOCHA D.: Real-time collision detection and distance computation on point cloud sensor data. In *ICRA* (2013), IEEE, pp. 3593–3599. 3, 8

[RTT09] RAVARI A., TAGHIRAD H., TAMJIDI A.: Vision-based fuzzy navigation of mobile robots in grassland environments. In *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on* (july 2009), pp. 1441 –1446. 3

[SBF09] SCHIAVI R., BICCHI A., FLACCO F.: Integration of active and passive compliance control for safe human-robot coexistence. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation* (Piscataway, NJ, USA, 2009), ICRA'09, IEEE Press, pp. 2471–2475. 3

[vdB98] VAN DEN BERGEN G.: Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools 2*, 4 (Jan. 1998), 1–13. 2

[Wel12] WELLER R.: *New Geometric Data Structures for Collision Detection*. Dissertation, University of Bremen, Germany, october 2012. 3

[WFZ13] WELLER R., FRESE U., ZACHMANN G.: Parallel collision detection in constant time. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (Lille, France, Nov. 2013), Eurographics Association. 3

[WGS04] WEINGARTEN J. W., GRUENER G., SIEGWART R.: A state-of-the-art 3d sensor for robot navigation. In *In IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* (2004), pp. 2155–2160. 1

[WZ09] WELLER R., ZACHMANN G.: Inner sphere trees for proximity and penetration queries. In *Robotics: Science and Systems Conference (RSS)* (Seattle, WA, USA, June/July 2009). 2, 4

[YBOB11] YINKA-BANJO C., OSUNMAKINDE I., BAGULA A.: Collision avoidance in unstructured environments for autonomous robots: A behavioural modelling approach. In *Proceedings of the IEEE 2011 International Conference on Control, Robotics and Cybernetics (ICCRC 2011)* (New Delhi India, 20 March 2011). 3

[ZL11] ZHAO W., LI L.: Improved k-dops collision detection algorithms based on genetic algorithms. In *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on* (Aug 2011), vol. 1, pp. 338–341. 2

[ZpTL08] ZHAO W., PU TAN R., LI W.-H.: Parallel collision detection algorithm based on mixed bvh and openmp. In *System Simulation and Scientific Computing, 2008. ICSC 2008. Asia Simulation Conference - 7th International Conference on* (Oct 2008), pp. 786–792. 3