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

Massively Parallel Batch Neural Gas for Bounding Volume Hierarchy Construction

René Weller[1], David Mainzer[2], Abhishek Srinivas[1], Matthias Teschner[3] & Gabriel Zachmann[1]

[1]University of Bremen, [2]Clausthal University, [3]University of Freiburg

Abstract

Ordinary bounding volume hierarchy (BVH) construction algorithms create BVHs that approximate the boundary of the objects. In this paper, we present a BVH construction that instead approximates the volume of the objects with successively finer levels. It is based on Batch Neural Gas (BNG), a clustering algorithm that is known from machine learning. Additionally, we present a novel massively parallel version of this BNG-based hierarchy construction that runs completely on the GPU. It reduces the theoretical complexity of the sequential algorithm from $O(n \log n)$ to $O(\log^2 n)$ and also our CUDA implementation outperforms the CPU version significantly in practice.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling —Object hierarchies I.5.3 [Pattern Recognition]: Clustering—Algorithms

1. Introduction

Bounding volume hierarchies (BVHs) are a widely used data structure to accelerate intersection computations in computer graphics and related fields. They have been successfully applied to ray tracing, collision detection, and visibility culling, to name but a few.

The basic idea is very simple: geometric primitives are wrapped into simple shapes called *bounding volumes (BVs)* that allow very fast intersection tests. Common BVs are *axis aligned bounding boxes (AABB)*, spheres, *discrete oriented polytopes (k-DOP)* or *oriented bounding boxes (OBB)*. In a bounding volume hierarchy, we group these bounding volumes into small sets and enclose them within larger bounding volumes recursively. This generates a tree data structure with a single large BV at the root position that encloses all geometric primitives. Obviously, the geometric primitives are the leaves of such a BVH.

When we perform an intersection test, we start at the root node of the BVH and, if it passes the test, continue to recursively traverse its children. If an intersection test fails at some node, we can simply skip the whole underlying branch because the intersection test will also fail for all children.

In order to guarantee a high culling efficiency, the BVH has to fulfill several, quality criteria: it should tightly fit the underlying geometry, provide fast intersection tests, be invariant undergoing rigid motion, not use too much memory, the inner nodes should not overlap heavily, and it should be able to be built automatically and fast. Unfortunately, these factors are partly contradictory. For example, spheres offer very fast overlap and distance tests, they are rotationally invariant and they can be stored very memory efficiently, but they poorly fit flat geometries. AABBs also offer fast intersection tests, but they need to be realigned after rotations. Consequently, choosing the right BVHs is always a compromise and depends on the scenario. There does not exist a *best* BVH for all circumstances. Especially, the quality of the partitioning usually has a significant influence on the performance during queries.

In computer graphics objects are usually represented only by their *surface*, e. g. by a polygonal mesh or as an implicit NURBS surface. Consequently, most work on BVHs has been spent on these object representations. Recently, Weller and Zachmann [WZ09] presented a new *volumetric* method to represent 3D object by a sphere packing. The basic idea is to fill a typical 3D surface representation from the *inside* with a set of *non-overlapping* spheres with different radii. These inside sphere packings allow the computation of the *penetration volume* as penetration measure for collision queries. According to Fisher and Lin [FL01, Sec. 5.1], this penetration measure is “the most complicated yet accurate method” to define the extent of intersection.

However, constructing a BVH of such sphere packings is challenging, because traditional methods that are opti-

Figure 1: Left: a dense polydisperse sphere packing representation of a dragon model. Right: in a wrapped hierarchy, the parent sphere (blue) covers all its leaf nodes (red), but not its direct children (green).

mized for surface representations are not automatically also suited for a volumetric BVH. For instance, BVH construction methods that were designed for classical *outer* sphere trees, like the medial axis approach [BO04, Hub95] work well if the spheres constitute a *covering* of the object and have very similar size, but in our scenario we use disjoint inner spheres that exhibit a large variation in size. Other approaches based on the *k-center problem* work only for sets of points and can be hardly extended to spheres.

In this paper, we extend a method that is previously known from machine learning, the *Batch Neural Gas (BNG)* clustering, for the construction of BVHs on sphere packings. In its pure form, BNG partitions a set of data points into a predefined number of clusters by minimizing the mean squared Euclidean distance of each data point to its nearest center. We adopt an extension called *magnification control* that enables us to take also the spheres' volume into account.

Basically, BNG adds a single point, a so called *prototype*, for each cluster to the sphere packing and moves them iteratively until some convergence criterion is met. The movement depends on the distance of the prototypes to all data points. Unfortunately, this requires a lot of convergence steps and hence, is relatively slow. Moreover, we have to start such a time consuming step for each BV in the BVH individually. Even if the construction of the BVH is a pre-processing step that has to be done only once, it should not be too slow. For instance, if we want to add a new object to an interactive real-time simulation we usually do not want to wait for minutes until the BVH for this object has been constructed.

In order to overcome this limitation, we present a novel massively parallel version of BNG that is especially optimized for the construction of BVHs. Our new algorithm runs completely on the GPU and in general, it does not require any time consuming copy operations between CPU and GPU memory during the whole BVH construction. Moreover, it reduces the theoretic complexity of the hierarchy construction of $O(n \log n)$ for the CPU version to $O(\log^2 n)$ using only $O(n)$ processors. Our novel parallel approach is easy to implement and robust against the start positions of the prototypes. Our CUDA implementation shows a significant speed up compared to the CPU version. Moreover, our results show that the BNG-based BVHs perform much better than BVHs that are constructed using simple heuristics for the sphere partitioning.

2. Previous Work

Wrapping objects in BVs and arranging the BVs into a tree hierarchy is commonly used to accelerate intersection computations in many fields of computer science. Because of their efficiency, BVHs have been extensively researched in the past and they are widely adopted in the area of computer graphics.

Gottschalk, Lin, and Manocha [GLM96] presented a new construction method for the oriented bounding box"=tree (OBB-tree) hierarchy. But the only optimal solution for OBB computation is $\mathcal{O}(n^3)$ and very hard to implement [O'R85]. Lauterbach et al. [LGS*09] used spatial Morton codes to reduce the BVH construction problem to a simple sorting problem, called Linear *Bounding Volume Hierarchy (LBVH)*.

For the case that traversal cost is most important and construction cost is relatively non"=relevant, using a *surface area heuristic (SAP)* as splitting criterion can be well suited. Lauterbach et al. [LGS*09] have recently demonstrated that building SAH BVHs on modern GPUs is possible, however their build times are still significantly higher than those for CPUs [Wal07], but traversal of the BVH is more efficient using GPUs. All these approaches do not consider the volume of the underlaying object, they only focus on the surface of an object, but this is entirely insufficient for volume based approaches, like e. g. inner sphere trees [WZ09] or tetrahedral meshes [THM*03]. Therefore, a different approach to create a hierarchy for volumetric objects

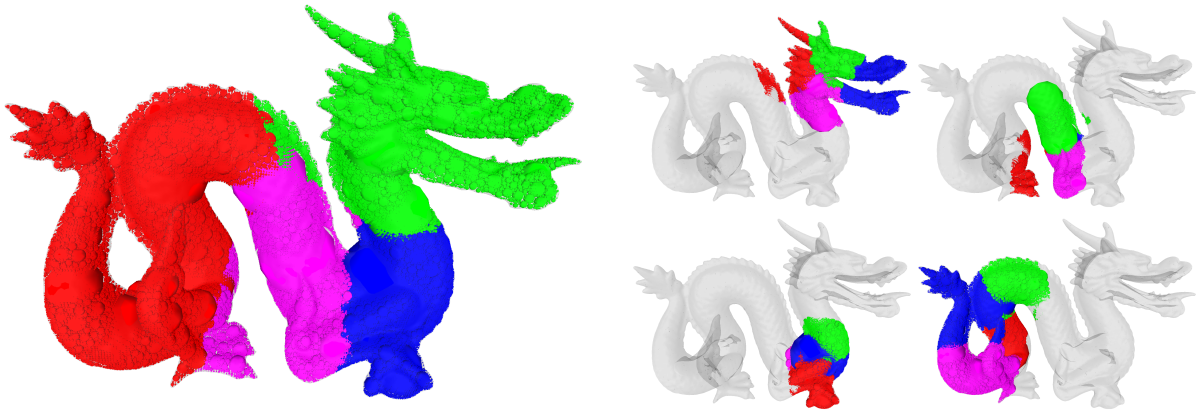

Figure 2: This figure shows the results of our hierarchy building algorithm based on Batch Neural Gas clustering with magnification control. All of those inner spheres that share the same color are assigned to the same bounding sphere. The left image shows the clustering result of the root sphere, the right images the partitioning of its four children.

is needed. A common approach for fitting points into classification is clustering. Since iterative improvement based partitioning approaches [KL70, FM82, WC89] do not perform very well on large input data sets, many different clustering approaches have been developed over the last years. The most common one, named *k-means* [M*67, Web03, DHS12], is a well"=established way of clustering data. Pelleg and Moore [PM99] used a kd-tree to improve the performance of the k-means clustering approach. Weber and Zezula [WZ97] showed that bounding trees do not scale well while the dimension increases.

In the last few years GPU have been the subject of attention and therefore, GPU"=based clustering approaches have been intensively investigated by researcher. Hall and Hart [HH04] used the fragment shader to fetch input data and cluster center for metric evaluation. One downside of this approach was the restriction in dimensionality due to texture memory limitation. Che et al. [CBM*08] and Zechner and Granitzer [ZG09] moved some computation steps on GPU, where every GPU thread is associated with a data point sequentially evaluating its label. The evaluation of determined mean values was done completely on CPU, so many memory transfer operations are required between CPU and GPU.

Hong-Tao et al. [HTLlDt*09] developed an approach where they further moved the new center evaluation partially on the GPU. The rearrangement of input vectors as per labels, however, was further done on CPU. Takizawa and Kobayashi [TK06] presented an effective parallel implementation scheme of k-means clustering. For the subdivision of a large-scale k-means clustering task their approach used a divide"=and"=conquer procedure. Another full CUDA based implementation has been developed by Wu, Zhang, and Hsu [WZH09] and Farivar et al. [FRCC08].

K-means clustering directly tries to minimize the quantization error [BB95]. However, its update scheme works only on a local part of the data set and therefore, it easily gets stuck in local optima. Another algorithm widely used for vector quantization is Kohonen [Koh82] Self Organizing Map (SOM) and the Neural Gas (NG) algorithm described by Martinetz, Schulten, et al. [MS*91]. There exist two classifications for different optimization schemes: online variants and batch variants.

Batch approaches are much faster, since only one adaptation is necessary in each cycle and this approaches converges after fewer steps. Nevertheless, Fort, Letremy, and Cottrell [FLC02] showed that topological order of SOM approach can be destroyed without a good initialization. Cottrell et al. [CHHV05] introduced a batch variant of the Neural Gas clustering algorithm (BNG). Their approach optimized the same cost function as the standard Neural Gas algorithm but converges much faster. In Section 3.1 we give a brief overview of the BNG algorithm and our novel GPU-based implementation for BVH construction.

3. Batch Neural Gas-based Hierarchy Creation

As described in the introduction, our objects are represented by a volumetric polydisperse sphere packing. This means, all spheres are located completely *inside* the object, and they *do not overlap* each other. However, the radii of the spheres varies. This allows us to approximate the object's volume to any required accuracy by using *space-filling* sphere packings.

Based on the sphere packing, we create an *inner* bounding volume hierarchy where the inner spheres are the leaves. In order to construct our hierarchy we use a top-down *wrapped hierarchy* approach according to the notion of Agarwal et al. [AGN*04], where inner nodes are tight BVs for all their

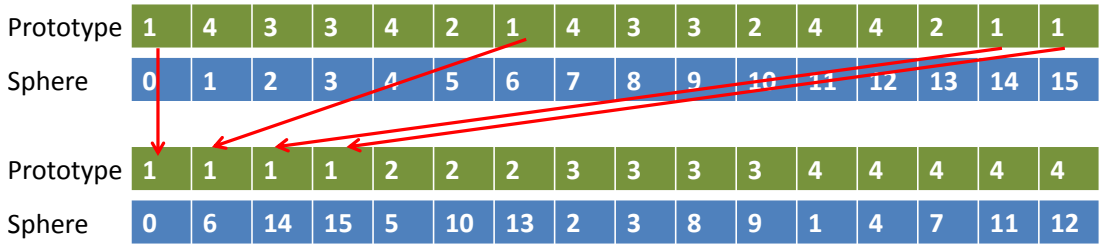

Figure 3: The top array stores the indices of the prototype to which the sphere in the array below has been assigned after the initial BNG clustering. In a first step, we sort the spheres with respect to their prototype index (the two lower arrays). Note, that each sphere is assigned to exactly one prototype.

leaves, but they do not necessarily bound their direct children (see Figure 1). Compared to layered hierarchies, the big advantage is that the inner BVs are tighter. We use a top-down approach to create our hierarchy, i.e., we start at the root node that covers all inner spheres and divide these into several subsets.

3.1. Batch Neural Gas Recap

So we decided to use the *Batch Neural Gas* clustering algorithm (BNG) known from machine learning [CHHV06]. BNG is a very robust clustering algorithm which can be formulated as stochastic gradient descent with a cost function closely connected to quantization error. Like *k-means*, the cost function minimizes the mean squared euclidean distance of each data point to its nearest center. But unlike k-means, BNG exhibits very robust behavior with respect to the initial cluster center positions (the *prototypes*): they can be chosen arbitrarily without affecting the convergence. Moreover, BNG can be extended to allow the specification of the *importance* of each data point; below, we will describe how this can be used to increase the quality of our BVH.

In the following we will give a quick recap of the basic Batch Neural Gas and then describe our extensions and application to building the inner sphere tree.

Given points $x_j \in \mathbb{R}^d, j = 0,\dots,m$ and prototypes $w_i \in \mathbb{R}^d, i = 0,\dots,n$ initialized randomly, we set the rank for every prototype w_i with respect to every data point x_j as

$$k_{ij} := \left|\{w_k : d(x_j,w_k) < d(x_j,w_i)\}\right| \in \{0,\dots,n\} \qquad (1)$$

In other words, we sort the prototypes with respect to every data point. After the computation of the ranks, we compute the new positions for the prototypes:

$$w_i := \frac{\sum_{j=0}^{m} h_\lambda(k_{ij}) x_j}{\sum_{j=0}^{m} h_\lambda(k_{ij})} \qquad (2)$$

These two steps are repeated until a stop criterion is met. In the original publication by Cottrell et al. [CHHV06], a fixed number of iterations is proposed. Indeed, after a certain number of iteration steps, which depends on the number of data points, there is no further improvement. We propose to use an adaptive version and stop the iteration if the movement of the prototypes is smaller than some ε. In our examples, we chose $\varepsilon \approx 10^{-5} \times$ BoundingBoxSize of the object, without any differences in the hierarchy compared to the non-adaptive, exhaustive approach. This improvement speeds up the creation of the hierarchy significantly.

The convergence rate is controlled by a monotonically decreasing function $h_\lambda(k) > 0$ that decreases with the number of iterations t. We use the function proposed in the original publication [CHHV06]: $h_\lambda(k) = e^{-\frac{k}{\lambda}}$ with initial value $\lambda_0 = \frac{n}{2}$, and reduction $\lambda(t) = \lambda_0 \left(\frac{0.01}{\lambda_0}\right)^{\frac{t}{t_{\max}}}$, where $t_{\max}$ is the maximum number of iterations. These values have been taken according to Martinetz, Berkovich, and Schulten [MBS93].

Obviously, the number of prototypes defines the arity of the tree. If it is too big, the resulting trees are very inefficient. On the other hand, if it is too small, the trees become very deep and there exist a lot of levels with big spheres that do not approximate the object very well. Experiments with our data structure have shown that a branching factor of 4 produces the best results. Additionally, this has the benefit that we can use the full capacity of SIMD units in modern CPUs during the traversal.

3.1.1. Magnification Control

So far, the BNG only utilizes the location of the centers of the spheres. In our experience this already produces reasonable results for the query performance. However, it does not yet take the extent of the spheres into account. This is, because Neural Gas uses only the *number* of data points and not their *importance*. As a consequence, the prototypes tend to avoid regions that are covered with a very large sphere, i.e., centers of big spheres are treated as outliers and they are thus placed on very deep levels in the hierarchy. However, it is better to place big spheres at higher levels of the hierarchy in order to get early lower bounds during distance traversal.

Therefore, we use an extended version of the classical Batch Neural Gas that also takes the size of the spheres

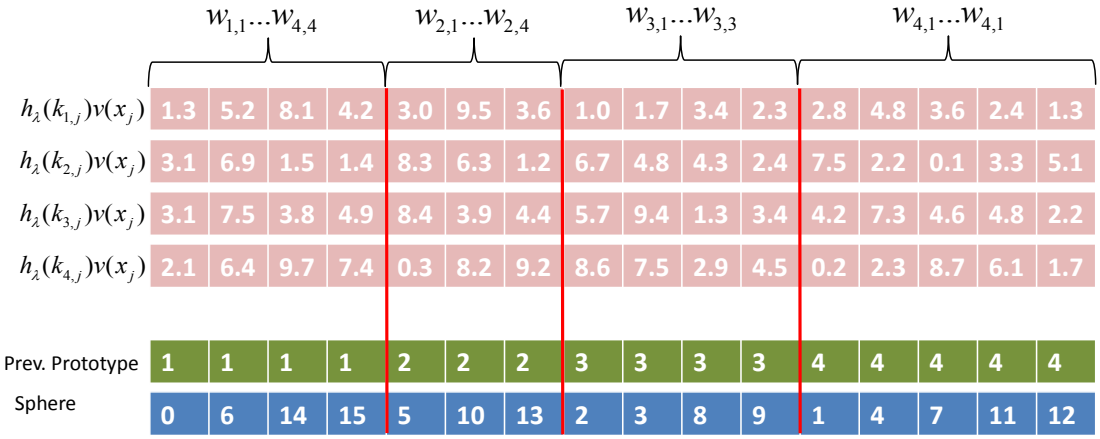

Figure 4: An example for the second level of the hierarchical BNG. According to Figure 3, each sphere has been assigned to a prototype. We insert 16 new prototypes, $w_{1,1}, \cdots, w_{4,4}$, 4 for each prototype $w_1, \cdots, w_4$ from the previous level and compute the values that are required by BNG, e. g. $h_\lambda(k_{ij})v(x_j)$. Please note that we do not have to allocate new memory or copy any values from CPU to GPU. We can simply re-use the memory from the previous level because each sphere was assigned to exactly one prototype. Consequently, we get a constant memory consumption for each level.

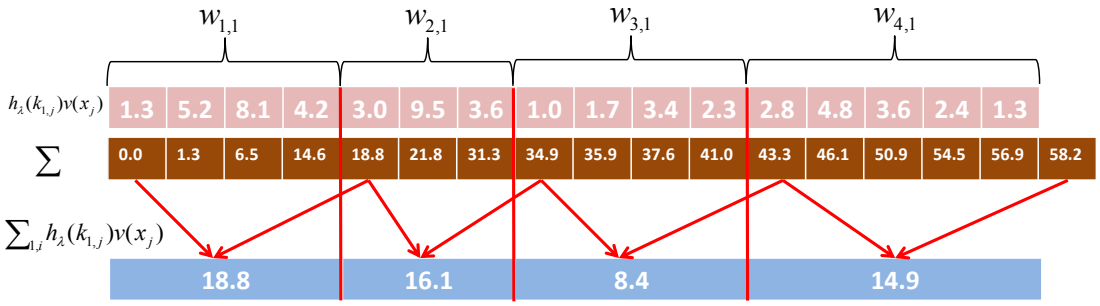

Figure 5: In order to compute the new position of the prototypes for the next iteration, we have to determine $\sum h_\lambda(k_{ij})v(x_j)x_j$. Therefore, we compute the prefix sum (brown array) for each of the four prototype arrays from Figure 4. The differences between the values at the borders directly deliver us the individual sum for each prototype.

into account. Our extension is based on an idea of Hammer, Hasenfuss, and Villmann [HHV06], where *magnification control* is introduced. The idea is to add weighting factors in order to "artificially" increase the density of the space in some areas.

With weighting factors $v(x_j)$, Equation 2 becomes

$$w_i := \frac{\sum_{j=0}^{m} h_\lambda(k_{ij})v(x_j)x_j}{\sum_{j=0}^{m} h_\lambda(k_{ij})v(x_j)} \tag{3}$$

Where $v(x_j)$ identifies a control parameter to take care of the importance. In Hammer, Hasenfuss, and Villmann [HHV06], a function of density is used to control the magnification. In our scenario we already know the density, because our spheres are disjoint. Thus, we can directly use the volumes of our spheres to let $v(x_j) = \frac{4}{3}\pi r^3$.

3.2. Batch Neural Gas Hierarchy Construction

Summing up the findings from before, the hierarchy creation algorithm can be described as follows: we first compute a bounding sphere for all inner spheres (at the leaves), which becomes the root node of the hierarchy. Therefore, we use the fast and stable smallest enclosing sphere algorithm proposed in Gärtner [Gär99]. Then, we divide the set of inner spheres into subsets in order to create the children. To do that, we apply the extended version of Batch Neural Gas with magnification control mentioned above. We repeat this scheme recursively (see Figure 2 for some clustering results).

3.2.1. Parallel Hierarchical Batch Neural Gas

The Batch Neural Gas algorithm produces a very good partitioning of the inner spheres, but as a drawback, it is very slow. Actually, we have to execute $O(n)$ BNG calls – one for

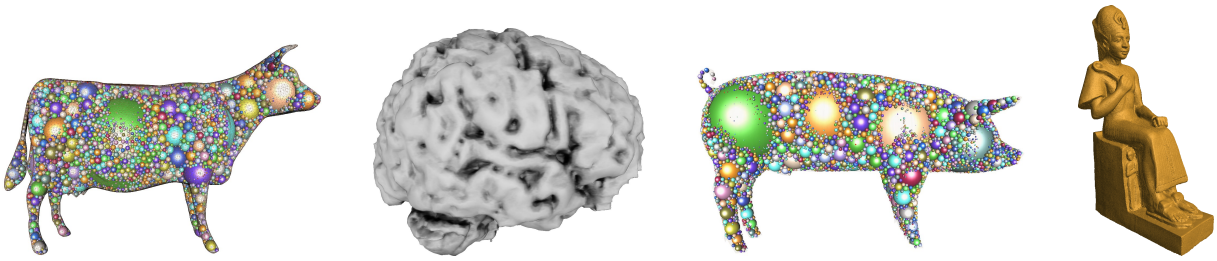

Figure 6: The objects we used in our timings: a cow, a human brain, a pig and a statue.

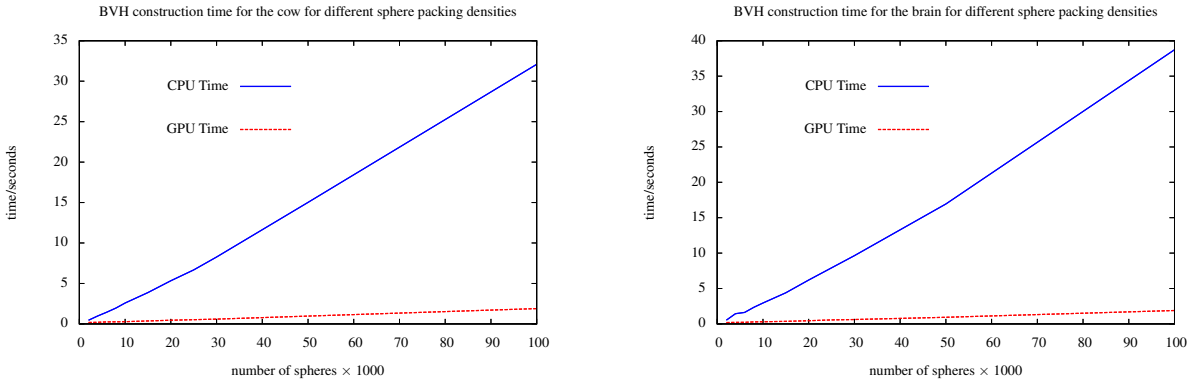

Figure 7: Left: CPU and GPU time for the BVH construction of cow model with different sphere packing densities. Right: the same for the brain model.

each hierarchy sphere – where n denotes the number of inner spheres. In case of a balanced tree with height $O(\log n)$ we have an overall running-time of $O(n \log n)$, but with a relatively high hidden constant factor that results from the number of iteration steps.

However, BNG in its pure form, but also the hierarchical BNG calls of our BVH creation, are perfectly suited for parallelization. Assuming $O(n)$ processors we are able to reduce the asymptotic running-time to $O(\log^2 n)$. In the following we will sketch the details of this parallel hierarchical BNG implementation using the GPU.

Obviously, on the first level of our hierarchy, the ordering k_{ij} and consequently also $h_\lambda(k_{ij})v(x_j)x_j$ can be computed independently for each sphere x_j. Summing up all those values can be implemented in parallel too, by using a parallel scan algorithm [SHG08]. The parallel assignment of spheres to prototypes is straightforward too: we simply have to compute the distances of each sphere to the prototypes. Please note, that each sphere is assigned to exactly one prototype.

In the next level of the BVH creation, we have to add 4 new prototypes for each prototype from the previous level (in case of a branching factor of 4). However, triggering an own parallel process for each sub-set of spheres would shoot down the advantages of parallel computing, especially in the deeper hierarchy levels. Therefore, we decided to chose another way. In the following we will describe its technical details.

First, we sort the spheres with respect to the prototype that the spheres were assigned to (see Figure 3). This can be done in parallel by using a parallel sorting algorithm [SHG09]. This technical detail allows us later to use fast parallel prefix-sum computations. However, after the sorting we virtually insert 4 new prototypes for each prototype from the previous hierarchy level. The fact that each sphere has been assigned to exactly one prototype in the previous level allows us to compute the values that are required for BNG (e. g. k_{ij}) in parallel for each sphere. We simply have to ensure that these values are computed for the *right new* prototypes (see Figure 4).

Finally, we have to sum up the individual values to get the new position of the prototypes; this means we have to compute $\sum_{j=0}^{m} h_\lambda(k_{ij})v(x_j)x_j$ and $\sum_{j=0}^{m} h_\lambda(k_{ij})v(x_j)$. Surprisingly, we can directly re-use the parallel prefix-sum from above [SHG08], even if we now need the sums for each new prototype individually: we simply have to subtract the values at the borders of our sorted prototype array (see Figure 5).

Algorithm 1 summarizes our complete parallel hierarchical BNG implementation.

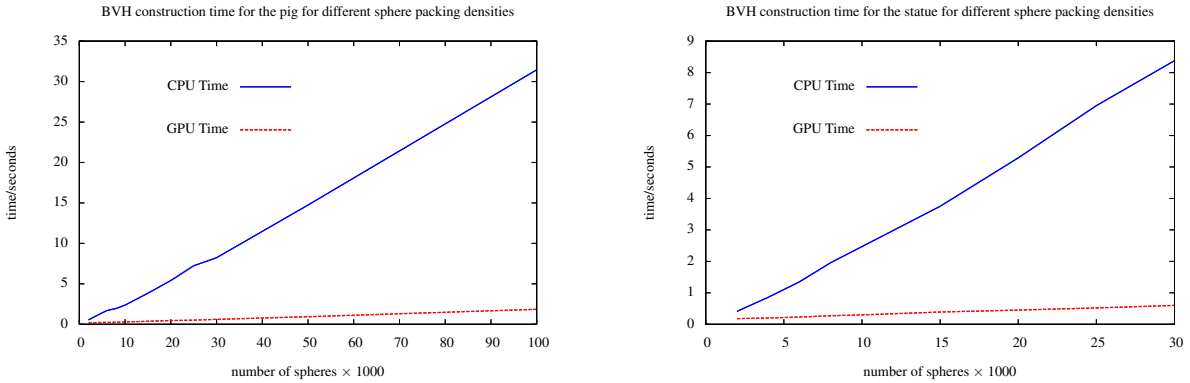

Figure 8: Left: CPU and GPU time for the BVH construction of pig model with different sphere packing densities. Right: the same for the statue model.

Algorithm 1: Parallel hierarchical BNG

while *Not on inner sphere level* **do**
  iteration = 0
  while *iteration<maxNumberIterations* **do**
    iteration++
    In parallel Sort prototype array
    In parallel forall the *Spheres* **do**
      compute $h_\lambda(k_{ij})v(x_j)x_j$
      and $h_\lambda(k_{ij})v(x_j)$
    In parallel Compute prefix sum
    In parallel forall the *Prototypes in level* **do**
      Compute new position
  read back prototype positions

The prefix sum and the sorting of the prototypes for n inner spheres can be computed in parallel using $O(n)$ processors in $O(\log n)$. Basically, both algorithms are based on an implicit balanced binary tree structure (see [SHG09] and [SHG08] for more details). The "per sphere" steps of Algorithm 1 have a complexity of $O(1)$, obviously. If the tree is balanced, the outer while-loop is called $O(\log n)$ times. Overall, we get a parallel time complexity of $O(\log^2 n)$. The memory consumption is $O(n)$.

4. Results

We implemented our algorithms using C++ for the CPU version and CUDA for the GPU version. All tests were performed on an Intel I7 CPU with 8GB main memory and a NVIDIA Gefore GTX 780 GPU with 3 GB of memory.

We used complex 3D models with very different shapes in our timings: in particular, two animal models, a detailed model of the human brain and a statue (See Figure 6). Additionally, we filled all models with different numbers of spheres ranging from 2k up to 100k inner spheres.

Our results show, that our novel hierarchical BNG hierarchy creation algorithms outperforms the CPU significantly. More precisely, we get an acceleration of a factor of 15 for all objects (see Figure 7 and 8). Please note, that our algorithm is not optimized yet, i.e. we do not use advanced CUDA acceleration techniques like shared memory. In practice it is essential that there is not too much traffic between the memories of the CPU and the GPU. In our algorithm there is almost no traffic required. In our current implementation, we only have to save the positions of the prototypes from the last iteration in the outer loop of Algorithm 1. However, this is also not really necessary. In the future, we plan to move the smallest enclosing sphere computation to the GPU too. Then, we only have to read back the whole hierarchy once. We only have to allocate memory for the prototypes once. This memory can be re-used for all iterations.

We also tested the performance of our BNG-based hierarchies for collision detection queries. To do that, we implemented two simple competing partitioning heutistics: First, we greedily choose the four biggest spheres and partition the smaller spheres to the closest of these large elements. Second, we sorted the spheres with respect to the coordinate axis and choose the two axis with the largest extend. Again, we assigned the intermediate spheres to the closest of the four extreme spheres.

In our two test scenes (See Figure 9) we used penetration volume queries. All tests were run on an Intel I7 processor. The collision query algorithm uses hand optimized SIMD code. The results show that the BNG hierarchies performed best in all our query test runs. Actually, they are more than a factor of 4 faster than the greedy choice of outer spheres (See Figure 10). Surprisingly, the simple greedy choice of biggest spheres performs well, but it is still 20% slower than our BNG hierarchies.

Figure 9: The test scenes for collision detection queries. Left: cow and pig. Right: pig and statue.

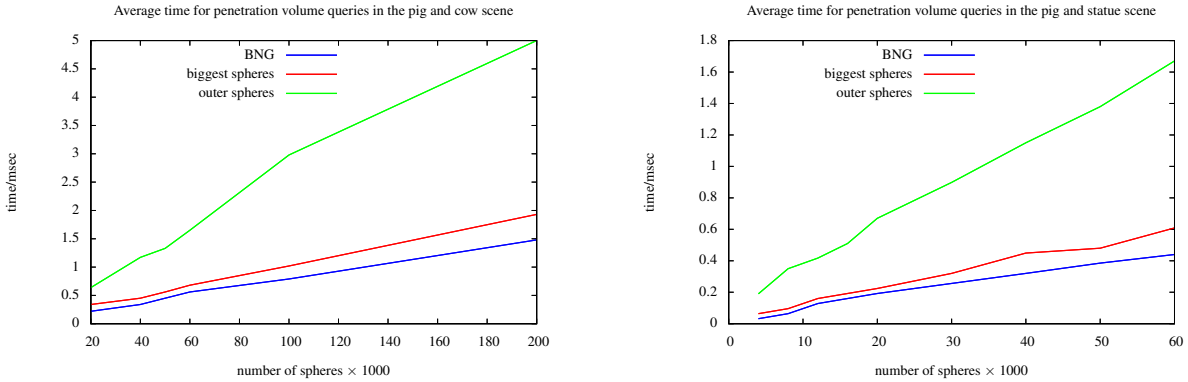

Figure 10: Left: Average time for collision detection queries in the pig and cow scene. Right: the same for the pig and statue scene.

5. Conclusions and Future Work

In this paper, we proposed a partitioning method for the construction of bounding volume hierarchies for volumetric object representations that is based on the Batch Neural Gas clustering known usually from machine learning. Our approach considers the object's volume instead of restricting the partitioning only to the surface, like most other algorithms do. Moreover, we presented a new parallel version of the BNG-based algorithm. It reduces the theoretical complexity from $O(n \log n)$ to $O(\log^2 n)$ using $O(n)$ processors. Additionally, we implemented our new BVH partitioning construction algorithm using CUDA. Our new robust massively-parallel implementation outperforms the CPU version by a factor of 15. During runtime, we recognized a speed-up of up to 4 for collision detection queries compared to simpler partitioning heuristics.

Our novel approach also opens up several avenues for future work. In the previous section we already mentioned the planned parallel implementation of the minimum enclosing sphere computation. However, it would be also interesting to apply our algorithm to other volumetric object representations than sphere packings, e. g. tetrahedra or ellipses. This could improve the quality of the volume covering because spheres do not fit well into some objects, especially if they have many sharp corners or thin ridges. Another option could be the investigation of our clustering-based BVH construction for classical *outer* BVHs. Currently, most implementations of classic BVHs use traditionally a branching factor of two. Due to recent developments in CPU technologies like SSE and AVX, higher branching factors could accelerate queries significantly. However, in this case, also more sophisticated partitioning techniques for the BVH construction are required because traditional heuristics for binary trees may not work anymore. Finally, we would like to explore other uses of *inner bounding volume hierarchies*, such as ray tracing or occlusion culling. Note that the type of bounding volume chosen for the "inner hierarchy" probably depends on its use.

Acknowledgement

This project is supported by the German Research Foundation (DFG) by grant TRR 8/3-2013.

References

[AGN*04] AGARWAL P., GUIBAS L., NGUYEN A., RUSSEL D., ZHANG L.: Collision detection for deforming necklaces. *Com-*

putational Geometry: Theory and Applications 28 (2004), 137–163.

[BB95] BOTTOU L., BENGIO Y.: Convergence properties of the k-means algorithms. In *Advances in Neural Information Processing Systems 7* (1995).

[BO04] BRADSHAW G., O'SULLIVAN C.: Adaptive medial-axis approximation for sphere-tree construction. *ACM Trans. Graph. 23*, 1 (Jan. 2004), 1–26. URL: http://doi.acm.org/10.1145/966131.966132, doi:10.1145/966131.966132.

[CBM*08] CHE S., BOYER M., MENG J., TARJAN D., SHEAFFER J. W., SKADRON K.: A performance study of general-purpose applications on graphics processors using cuda. *Journal of parallel and distributed computing 68*, 10 (2008), 1370–1380.

[CHHV05] COTTRELL M., HAMMER B., HASENFUSS A., VILLMANN T.: Batch neural gas. In *5th Workshop On Self-Organizing Maps* (2005), Citeseer.

[CHHV06] COTTRELL M., HAMMER B., HASENFUSS A., VILLMANN T.: Batch and median neural gas. *Neural Networks 19* (jul 2006), 762–771.

[DHS12] DUDA R. O., HART P. E., STORK D. G.: *Pattern classification*. John Wiley & Sons, 2012.

[FL01] FISHER S., LIN M.: Fast penetration depth estimation for elastic bodies using deformed distance fields. In *Proc. International Conf. on Intelligent Robots and Systems (IROS)* (2001), pp. 330–336.

[FLC02] FORT J.-C., LETREMY P., COTTRELL M.: Advantages and drawbacks of the batch kohonen algorithm. In *ESANN* (2002), vol. 2, pp. 223–230.

[FM82] FIDUCCIA C. M., MATTHEYSES R. M.: A linear-time heuristic for improving network partitions. In *Design Automation, 1982. 19th Conference on* (1982), IEEE, pp. 175–181.

[FRCC08] FARIVAR R., REBOLLEDO D., CHAN E., CAMPBELL R. H.: A parallel implementation of k-means clustering on gpus. In *PDPTA* (2008), pp. 340–345.

[Gär99] GÄRTNER B.: Fast and robust smallest enclosing balls. In *ESA* (1999), Nesetril J., (Ed.), vol. 1643 of *Lecture Notes in Computer Science*, Springer, pp. 325–338. URL: http://link.springer.de/link/service/series/0558/bibs/1643/16430325.htm.

[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* (1996), ACM, pp. 171–180.

[HH04] HALL J. D., HART J. C.: Gpu acceleration of iterative clustering. In *Proceedings of the ACM Workshop on General Purpose Computing on Graphics Processors* (2004).

[HHV06] HAMMER B., HASENFUSS A., VILLMANN T.: Magnification control for batch neural gas. In *ESANN* (2006), pp. 7–12. URL: http://www.dice.ucl.ac.be/Proceedings/esann/esannpdf/es2006-83.pdf.

[HTLlDt*09] HONG-TAO B., LI-LI H., DAN-TONG O., ZHAN-SHAN L., HE L.: K-means on commodity gpus with cuda. In *Computer Science and Information Engineering, 2009 WRI World Congress on* (2009), vol. 3, IEEE, pp. 651–655.

[Hub95] HUBBARD P. M.: Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics 1*, 3 (Sept. 1995), 218–230.

[KL70] KERNIGHAN B. W., LIN S.: An efficient heuristic procedure for partitioning graphs. *Bell system technical journal 49*, 2 (1970), 291–307.

[Koh82] KOHONEN T.: Self-organized formation of topologically correct feature maps. *Biological cybernetics 43*, 1 (1982), 59–69.

[LGS*09] LAUTERBACH C., GARLAND M., SENGUPTA S., LUEBKE D., MANOCHA D.: Fast bvh construction on gpus. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 375–384.

[M*67] MACQUEEN J., ET AL.: Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (1967), no. 281-297, California, USA, p. 14.

[MBS93] MARTINETZ T. M., BERKOVICH S. G., SCHULTEN K. J.: 'Neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Trans. on Neural Networks 4*, 4 (1993), 558–569.

[MS*91] MARTINETZ T., SCHULTEN K., ET AL.: *A" neural-gas" network learns topologies*. University of Illinois at Urbana-Champaign, 1991.

[O'R85] O'ROURKE J.: Finding minimal enclosing boxes. *International journal of computer & information sciences 14*, 3 (1985), 183–199.

[PM99] PELLEG D., MOORE A.: Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* (1999), ACM, pp. 277–281.

[SHG08] SENGUPTA S., HARRIS M., GARLAND M.: *Efficient Parallel Scan Algorithms for GPUs*. Tech. Rep. NVR-2008-003, NVIDIA Corporation, Dec. 2008. URL: http://mgarland.org/papers.html#segscan-tr.

[SHG09] SATISH N., HARRIS M., GARLAND M.: Designing efficient sorting algorithms for manycore GPUs. In *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium* (May 2009).

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANETS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects.

[TK06] TAKIZAWA H., KOBAYASHI H.: Hierarchical parallel processing of large scale data clustering on a pc cluster with gpu co-processing. *The Journal of Supercomputing 36*, 3 (2006), 219–234.

[Wal07] WALD I.: On fast construction of sah-based bounding volume hierarchies. In *Interactive Ray Tracing, 2007. RT'07. IEEE Symposium on* (2007), IEEE, pp. 33–40.

[WC89] WEI Y.-C., CHENG C.-K.: Towards efficient hierarchical designs by ratio cut partitioning. In *Computer-Aided Design, 1989. ICCAD-89. Digest of Technical Papers., 1989 IEEE International Conference on* (1989), IEEE, pp. 298–301.

[Web03] WEBB A. R.: *Statistical pattern recognition*. John Wiley & Sons, 2003.

[WZ97] WEBER R., ZEZULA P.: The theory and practice of searches in high dimensional dataspaces. In *Proceedings of the Fourth DELOS Workshop on ImageIndexing and Retrieval* (1997).

[WZ09] WELLER R., ZACHMANN G.: Inner sphere trees for proximity and penetration queries. In *Robotics: Science and Systems (RSS)* (28 June–1 July 2009). URL: http://cg.in.tu-clausthal.de/.

[WZH09] WU R., ZHANG B., HSU M.: Clustering billions of data points using gpus. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop* (2009), ACM, pp. 1–6.

[ZG09] ZECHNER M., GRANITZER M.: Accelerating k-means on the graphics processor via cuda. In *Intensive Applications and Services, 2009. INTENSIVE'09. First International Conference on* (2009), IEEE, pp. 7–15.