Extended Morton Codes for High Performance Bounding Volume Hierarchy Construction

Marek Vinkler
Czech Technical University,
Faculty of Electrical Engineering
Karlovo nam. 13
Prague 12135
vinkler@fel.cvut.cz

Jiri Bittner
Czech Technical University,
Faculty of Electrical Engineering
Karlovo nam. 13
Prague 12135
bittner@fel.cvut.cz

Vlastimil Havran
Czech Technical University,
Faculty of Electrical Engineering
Karlovo nam. 13
Prague 12135
havran@fel.cvut.cz

ABSTRACT

We propose an extension to the Morton codes used for spatial sorting of scene primitives. The extended Morton codes increase the coherency of the clusters resulting from the object sorting and work better for non-uniform distribution of scene primitives. In particular, our codes are enhanced by encoding the size of the objects, applying adaptive ordering of the code bits, and using variable bit counts for different dimensions. We use these codes for constructing Bounding Volume Hierarchies (BVH) and show that the extended Morton code leads to higher quality BVH, particularly for the fastest available BVH build algorithms that heavily rely on spatial coherence of Morton code sorting. In turn, our method allows to achieve up to 54% improvement in the BVH quality especially for scenes with a non-uniform spatial extent and varying object sizes. Our method is easy to implement into any Morton code based build algorithm as it involves only a modification of the Morton code computation step.

CCS CONCEPTS

•Information systems →Multidimensional range search;
•Theory of computation →Sorting and searching;
•Computing methodologies →Ray tracing;

KEYWORDS

ray tracing, bounding volume hierarchy, Morton codes

ACM Reference format:
Marek Vinkler, Jiri Bittner, and Vlastimil Havran. 2017. Extended Morton Codes for High Performance Bounding Volume Hierarchy Construction. In *Proceedings of HPG '17, Los Angeles, CA, USA, July 28-30, 2017,* 8 pages.
DOI: 10.1145/3105762.3105782

1 INTRODUCTION

Spatial data structures play a key role in computer graphics. Any data set containing many details is too complex to be processed without organizing the scene objects in a spatial data structure. Spatial data structures assist for example in performing visibility culling, collision detection, and ray tracing. Particularly for ray tracing based rendering applications including interactive ones it is important to improve on the quality of the spatial data structures while keeping the time of building them low. If the scene contains a dynamic content, it is also important to reconstruct or update the spatial data structure quickly to accommodate the scene changes. In such situations, we seek to balance the construction time and the rendering performance. Particularly in interactive applications, the construction time should match the target frame-rate, which is a difficult task for most of the commonly used scenes.

In this paper, we focus on fast construction of Bounding Volume Hierarchies (BVH) that are one of the most popular spatial data structures nowadays. BVHs have predictable memory footprint, allow for relatively fast construction, and provide very good run-time performance. Most state-of-the-art BVH construction methods exploit Morton codes for spatial sorting of scene primitives [Domingues and Pedrini 2015; Garanzha et al. 2011; Gu et al. 2013; Lauterbach et al. 2009]. Morton codes are employed in these algorithms because they provide a simple yet efficient method for approximate spatial sorting of scene primitives.

We propose a new code, formed as an extension of the Morton code. Our Extended Morton code (EMC) can significantly improve the quality of the BVH constructed with the fastest available BVH construction algorithms, while not increasing the computation time. We show that even for such simple build method (LBVH [Karras 2012]) the quality, expressed as the expected cost of the created BVH, can get close to the high-quality BVHs constructed using a sweeping-based algorithm with the surface area heuristic, but in a fraction of the computation time. Moreover, we show that even the highest quality build algorithms such as ATRBVH [Domingues and Pedrini 2015] can benefit from the use of EMC.

2 RELATED WORK

The build algorithms for data structures used in ray tracing have undergone intensive research, both on multi-core CPU and many-core GPU platforms. Below we summarize the main algorithms for BVH construction on both platforms and discuss how they can benefit from the proposed extended Morton codes.

2.1 CPU based algorithms

One of the first techniques significantly speeding up the build of high-quality data structures on CPUs was the binned BVH build proposed by Havran et al. [2006]. This idea was later used in a parallel implementation by Wald [2007], where top levels of the tree were built by a lower quality but fast grid based splitting. This top level build can be replaced by our extended Morton code build for higher quality data structures. A recent advance in the field of binned BVH building on the CPU is the one by Olivares et al. [2016].

HPG '17, Los Angeles, CA, USA
© 2017 ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of HPG '17, July 28-30, 2017*, http://dx.doi.org/10.1145/3105762.3105782.

Another avenue of research features bottom-up build using agglomerative clustering [Gu et al. 2013; Walter et al. 2008] to improve on the quality of BVHs. This algorithm directly employes presorting by Morton codes and thus can immediately benefit from our extended variant. By considering multiple bits of the Morton code for a split at each level, BVHs with higher arity of nodes can be built [Dammertz et al. 2008].

Finally, optimization of the BVH was proposed by the means of node rotations [Kensler 2008; Kopta et al. 2012] or insertions/removal of whole subtrees [Bittner et al. 2013]. Such algorithms were shown to benefit from starting the optimization on a tree of higher quality and thus they can also benefit from improving the Morton code based construction of the initial BVH. An alternative to optimizing the cost of an existing tree is to better predict the cost of the splits during the build by using temporarily constructed BVHs as in the work of Wodniok and Goesele [2017].

2.2 GPU based algorithms

Morton code based build algorithms were used on GPUs from the beginning because of their easy parallelization. The first algorithm exploiting the code properties is the linear BVH (abbreviated to LBVH) proposed by Lauterbach et al. [2009]. Pantaleoni and Luebke [2010] and Garanzha et al. [2011] proposed hybrid build algorithms (known as HLBVH) that are both fast and produce high-quality data structures by building parts of the tree using the surface area heuristic. Karras [2012] proposed an improved build algorithm for LBVHs based on radix trees to further reduce the build time. This method was improved by Apetrei by reducing the number of passes of the algorithm [Apetrei 2014]. An algorithm for massively parallel optimization of BVHs on a GPU was given by Karras et al. [2013]. In their work the BVH topology is changed by rearranging subtrees, similarly to the method of Kopta et al. [2012]. This algorithm was further optimized by Domingues [2015] for even faster builds. All the above-mentioned methods use Morton codes and can thus be easily upgraded to extended Morton codes without much implementation effort.

3 MORTON CODE BASED SPATIAL SORTING

3.1 Morton codes in 3D

Morton code is a way of mapping quantized n-dimensional vectors into integer scalar values, i.e., codes. Morton code induces a space filling curve that provides relatively coherent ordering of the quantized vectors, meaning the vectors with subsequent Morton codes are spatially close to each other. This curve is also referred to as Z-curve because of the shape of the curve in the 2D case (see Figure 1). Although other space filling curves exist that yield even more coherent spatial ordering [Samet 2005], Morton code gained popularity because of the simplicity of its computation: the Morton code can be computed using simple bit interleaving operations.

A major parameter of the Morton code is the number of bits used for the code. In particular an n bit Morton code of a three dimensional vector $\mathbf{v} = (v_x, v_y, v_z) \in \langle 0, 1\rangle^3$ is computed by first determining the quantized coordinates $\mathbf{v}^* = \{v_x^*, v_y^*, v_z^*\} \in \langle 0, 2^{n/3}\rangle \times \langle 0, 2^{n/3}\rangle \times \langle 0, 2^{n/3}\rangle$. The Morton code is then evaluated by interleaving bits of the components of v^*. For example, a 24-bit Morton code for the quantized coordinates $v_x^* = x_7x_6x_5x_4x_3x_2x_1x_0$, $v_y^* = y_7y_6y_5y_4y_3y_2y_1y_0$, and $v_z^* = z_7z_6z_5z_4z_3z_2z_1z_0$ is then $m(v^*)$= $x_7y_7z_7x_6y_6z_6x_5y_5z_5x_4y_4z_4x_3y_3z_3x_2y_2z_2x_1y_1z_1x_0y_0z_0$. The bit interleaving operation can be performed using algebraic operations such as multiplication and sums [Morton 1966]. Alternatively, look-up tables can be used to precompute the codes for certain bit ranges and then combined at runtime by simple additions and shifts.

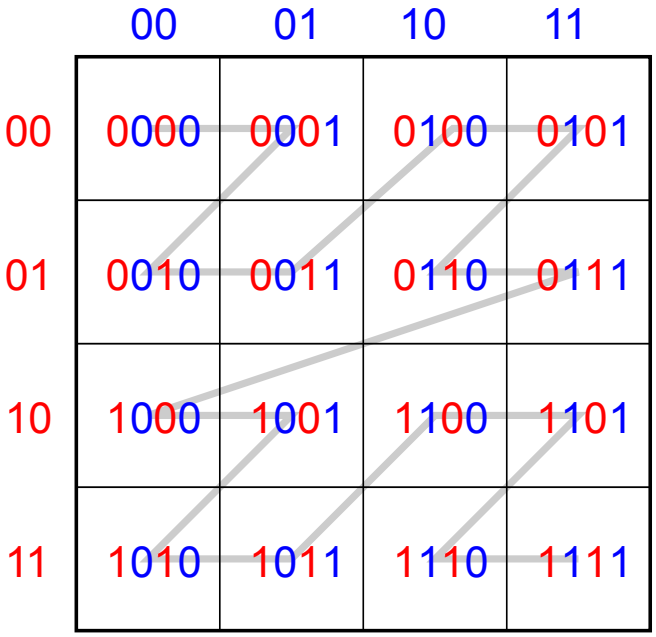


Figure 1: Visualization of the Morton code based space filling curve in 2D.

3.2 Constructing BVH using Morton Codes

Since Morton codes encode the spatial location of vectors (points), they can be easily used for the construction of spatial data structures. By representing scene primitives as points and sorting them by their Morton code, multi-scale clusters of primitives can be formed. The boundaries between the clusters are indicated by the change of a bit in the Morton code on certain bit position.

This property is exploited by the algorithms using Morton codes for spatial sorting of scene primitives. The LBVH method [Lauterbach et al. 2009] sorts primitives according to the Morton codes of their centroids. Starting from the highest bit, the hierarchy is constructed by finding the positions of bit changes and applying this procedure recursively on the two subsets. Thus the BVH topology is solely given by the computed Morton codes.

The HLBVH method [Pantaleoni and Luebke 2010] performs the BVH construction in two steps. The highest levels of the BVH are built using the Morton codes similar to the algorithm of Lauterbach. When the individual subsets become small enough, they are further subdivided using the surface area heuristic (SAH) for improved quality. Because the quality of splits matters mostly near the top of the hierarchy, Garanzha at al. [2011] proposed to turn this two-level build around. They first identify clusters of primitives that share a common prefix of the Morton code of a given length. For these clusters, the subtrees are constructed using the remaining

bits of the code, but the top part of the tree is constructed using the binning SAH method [Wald 2007] executed on the bounding boxes of the clusters. Thus, the Morton codes directly influence how these clusters look like and also how the topology of the lower levels of the tree is formed. Probably the fastest algorithm to date for constructing BVHs based on Morton codes is the one of Apetrei [2014] on GPU and the Embree [Wald et al. 2014] implementation on CPU.

The AAC method [Gu et al. 2013] uses Morton codes to recursively subdivide objects into clusters on which the agglomerative clustering is applied in a bottom-up fashion. The Morton codes directly influence how these clusters are formed, but the method has a potential to postpone clustering towards higher levels of the tree and thus to limit the potentially inefficient clusters resulting from the Morton codes.

4 EXTENDED MORTON CODES

Since any BVH topology can be embedded in the space of bit codes and built efficiently with the algorithm of Karras [2012], it is natural to search for good embeddings. In the rest of this section, we propose several extensions of the standard Morton code that lead to embeddings for higher quality data structures with practically no increase in the build time.

4.1 Encoding object size

Many real-world scenes contain geometry primitives of different sizes. When sorting such primitives based on the centroids of their bounding boxes, the resulting object clustering can be inefficient for the following reason: even though the centroids of the objects can be very close (and thus assumed coherent) their common bounding volume is not a good fit, especially for smaller objects in the cluster. In other words, large objects enforce large bounding volumes not only at a single node in the tree, but on the whole path from the leaf to the root. Thus, it is beneficial to separate such objects and possibly keep them at higher levels of the tree so that they do not influence the rest of the tree.

The decision on how to separate the large objects has to be done with care. Too early and too late separation of large objects could actually increase the overlaps of bounding volumes and in turn, decrease the BVH quality. We propose to inject extra bits into the Morton code that encode the size of the object. In particular, we use the length of the diagonal of the object. In the simplest form, the size is encoded using the same number of bits as the other axes. Thus, the Morton code is then computed by interleaving bits of 4 quantized coordinates (x, y, z, s), where x, y, z encode the spatial position of geometric primitive centroids and s encodes the size of the geometric primitive. The size s is normalized so that the highest value represented using the given number of bits corresponds to the diagonal of the scene bounding box.

When using regular bit interleaving ($xyzsxyzs$) the bounding box corresponding to a particular Morton code prefix follows a regular octree-like subdivision of the scene volume. In other words, each triple of spatial splits subdivides the original volume into eight equally shaped sub-volumes (boxes) with their diagonal size equal to the half of the original box diagonal size. Therefore the bits encoding the size of the object can be directly used to determine the objects with their size smaller/larger than the current volume of the centroids. In particular, the objects with the size bit set to 1 are larger than the current box and will be separated into a different subtree (see Figure 2).

We have also experimented with encoding the object size using its surface area, but the diagonal length achieved slightly better overall results in our tests.

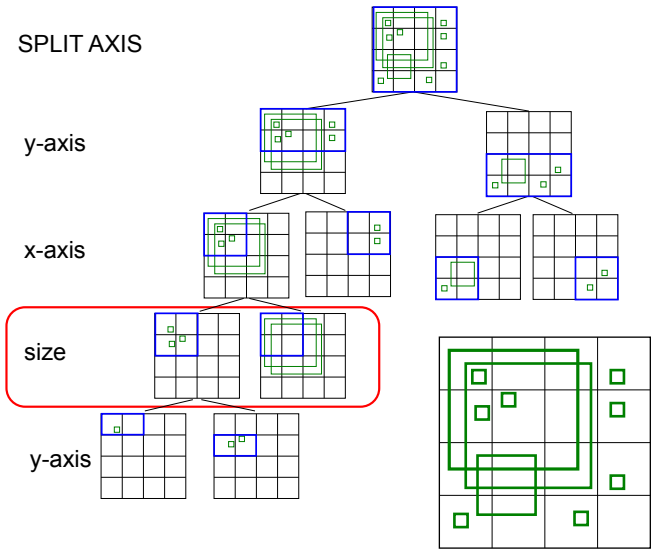


Figure 2: Illustration of the BVH construction on a simple scene using the extended Morton code. Note the third level of the hierarchy in which the size bit is used to subdivide objects into those smaller and larger than the diagonal of the corresponding centroid volume (shown in blue).

4.2 Using fewer size bits

In some situations, it might be beneficial to encode the object size in a coarser manner not to loose the main potential for subdivision given by the object positions. This applies especially for shorter bit codes (32-bit Morton codes) or for large scenes with many primitives (e.g. more than 10M triangles).

We propose a method that reduces the number of bits used to encode the object size while keeping its main benefit of separating small and large objects at appropriate levels. The main idea is to inject the size bits only every seventh bit, thus after two series of splits in each axis. In this case, however, we should make sure that the encoded object size corresponds to the size of the centroid bounding box corresponding to the given bit position. After six spatial splits, the size of the diagonal becomes one-fourth of the size of the diagonal of the box at which the previous size split was performed. At the same time, the following bit in a binary coded number represents half of the previous (higher) bit value. This discrepancy can be easily resolved by encoding a quantized value that corresponds to the *square root* of the diagonal of the object size.

Algorithm 1 Pseudocode of the 64-bit EMC generation using the adaptive axis order and size bits injected every 4th bit. The EXPAND function inserts three zeros before each bit of the input number. Notation: X ≪Y corresponds to shift left bitwise operation of integer X by Y bits, | corresponds to bitwise binary operation OR, and & corresponds to bitwise binary operation AND.

```
1:  function INIT(scene)
2:      ▷ default axis order
3:      a_0 = x, a_1 = y, a_2 = z, a_3 = size
4:      ▷ descending axis order using scene dimensions
5:      sort(a_{0..2})
6:      ▷ Compute quantization scales
7:      s_0 = 2^16/scene.box.size[a_0]
8:      s_1 = 2^16/scene.box.size[a_1]
9:      s_2 = 2^16/scene.box.size[a_2]
10:     s_3 = 2^16/scene.box.diagonal.length
11: end function
12: function EXPAND((integer X))
13:     integer v=0, mask=1
14:     for i=0 to 15 step 1 do
15:         v = v | ((X & mask) ≪(3.i))
16:         mask = mask ≪1
17:     end for
18:     return v
19: end function
20: function CODE(triangle)
21:     v = triangle.box.center-scene.box.min
22:     size = triangle.box.diagonal.length
23:     integer v*_0 = s_0 · v[a_0]
24:     integer v*_1 = s_1 · v[a_1]
25:     integer v*_2 = s_2 · v[a_2]
26:     integer v*_3 = s_3 · size
27:     return ((Expand(v*_0) ≪ 3 ) | (Expand(v*_1) ≪ 2)
                | (Expand(v*_2) ≪ 1 ) | Expand(v*_3))
28: end function
```

4.3 Adaptive axis order

During our tests, we discovered that the order in which the bits corresponding to different spatial axes are used in the code influences the resulting BVH quality. In particular, ordering the axes by putting the ones with the largest extent at the beginning of the code had a consistently positive effect on the results. We expect that this follows from the fact that by splitting the larger axes first, the centroid clusters are closer to the cuboid shape and thus more spatially compact than always keeping a fixed (xyz) order.

Algorithm 1 shows a pseudocode of a simple variant of the algorithm using the adaptive axis order and size bits injected at every 4th bit. The actual code is very similar to standard Morton code encoding, with a few modifications. During initialization additional sorting of axes is performed and the *Expand* function injects three zeros before each bit of quantized coordinates unlike two bits injected in standard 3D Morton coder. Finally, the size bits are computed from the diagonal of the triangle and inserted into the resulting code.

4.4 Variable bit count

The adaptive axis order can further be generalized by allowing a variable number of bits to be used for different axes. If the scene extends mainly in two dimensions (such as terrain or city models), it might be beneficial to subdivide in these two dimensions multiple times and subdivide in the remaining dimension only after the subdivided volume becomes close to a cube. Inspired by the major-axis based splitting used for BVH construction, we propose a simple method that determines the order of axes encoding. Unlike the classical major-axis based splitting, where the decision is made for each node separately, here the proposed method is solely based on the knowledge of the scene bounding box. Starting from the scene bounding box, we determine the largest axis according to which a split should be done at the first bit position and then cut the bounding box in half according to the selected axis. The selection of the axis for each bit is stored in an auxiliary array and in this way we determine the axis corresponding to all spatial bits of the extended Morton code. Note that during this process we optionally also inject the size bits (every 4th or 7th bit). When all bits are assigned, we can count the number of bits used for different axes and use these counts for computing appropriate quantization scales for the input points (see Algorithm 2).

Note that the resulting code is no longer the traditional Morton code that uses a regular bit interleaving, however, the main features of the code for the BVH construction are kept (the bit based subdivision of primitives). The adaptive axis order typically influences only the first two to four bits of the code – the rest of the code contains regular triples (or quadruples in the case of injected size bit) of the spatial axes, i.e., a 16-bit code might look like $xyxyzxyzxyzxyzxy$. As soon as the shape of the bounding box approaches the cube and no axis is twice larger than any other axis, selecting the largest split axis will produce a regular subdivision pattern.

4.5 Implementation

For maximizing the performance of the Morton code computation step, we use look-up tables (LUTs) for all axes that contain values with accordingly shifted bits. The LUTs are precomputed before evaluating the Morton codes of scene primitives. Then the scene data is processed by simple look-ups followed by summing the partial codes representing different axes. Note that for the variable bit counts the LUTs have to be recomputed when the scene bounding box changes. For all other codes, the LUTs are independent of the scene bounding box and remain fixed for a particular Morton code setup.

In our implementation, the square root needed for the 7th-bit encoding described in Section 4.2 is actually avoided by encoding it in the LUT. To avoid loss of precision resulting from the square root we use a LUT of twice the normal size $2^{2|s|}$, where $|s|$ is the number of size bits used. Note that since the encoded size uses the triangle bounding box diagonal, we still need one square root to compute the size of the diagonal. This computation could also be replaced by another access to the LUT, but we have not implemented this feature in our code as it would require a further increase of the LUT size.

We provide a sample implementation of the Extended Morton coders for download at: http://dcgi.fel.cvut.cz/projects/emc.

Algorithm 2 Pseudocode of the 64-bit EMC generation using the variable bit order, adaptive axis order and size bits injected every 7th bit. The EXPAND function inserts variable number of zeros before each bit of the input number. Notation: X ≪Y corresponds to shift left bitwise operation of integer X by Y bits, | corresponds to bitwise binary operation OR, and & corresponds to bitwise binary operation AND.

```
1:  function INIT(scene)
2:      for i=0 to 63 step 1 do
3:          if i is 7th bit then
4:              ▷ insert size bit
5:              axes[i] = 3
6:          else
7:              ▷ pick the largest axis
8:              axes[i] = scene.box.size.maxAxis
9:              ▷ halve the largest axis
10:             scene.box.size.max/ = 2
11:         end if
12:     end for
13:     ▷ number of bits in each axis
14:     bits_{0..3} = axes.sum(0..3)
15:     ▷ shifts between bits in each axis
16:     shifts_{0..3} = axes.shifts(0..3)
17:     ▷ Compute quantization scales
18:     s_0 = 2^{bits_0}/scene.box.size[a_0]
19:     s_1 = 2^{bits_1}/scene.box.size[a_1]
20:     s_2 = 2^{bits_2}/scene.box.size[a_2]
21:     s_3 = 2^{bits_3}/scene.box.diagonal.length
22: end function
23: function EXPAND(axis A, integer X)
24:     integer v=0, mask=1
25:     for i=0 to bits_A-1 step 1 do
26:         v = v | ((X & mask) ≪(shifts_A[i]))
27:         mask = mask ≪1
28:     end for
29:     return v
30: end function
31: function CODE(triangle)
32:     v = triangle.box.center-scene.box.min
33:     size = triangle.box.diagonal.length
34:     integer v*_0 = s_0 · v[a_0]
35:     integer v*_1 = s_1 · v[a_1]
36:     integer v*_2 = s_2 · v[a_2]
37:     integer v*_3 = s_3 · size
38:     return ((Expand(a_0, v*_0) ≪3 ) | (Expand(a_1, v*_1) ≪2)
                | (Expand(a_2, v*_2) ≪1 ) | Expand(a_3, v*_3))
39: end function
```

5 RESULTS

We evaluated the extended Morton codes in a fast parallel build algorithms running on the GPU. The primary goal of the extended Morton codes is to improve on the build algorithms that heavily use Morton code based ordering to construct the BVH. This is the case of the LBVH build algorithm [Karras 2012; Lauterbach et al. 2009], which is one of the fastest to date. Since the LBVH algorithm tends to build data structures with only mediocre ray tracing performance, we also explore the state-of-the-art approach of fast build followed by tree topology optimization [Domingues and Pedrini 2015; Karras and Aila 2013]. In particular, we used the method with the ATRBVH algorithm proposed by Domingues and Pedrini [2015].

We ran our evaluation on a system with Intel Xeon E3-1246 processor and the GeForce GTX 960 graphics card. We used eight scenes for the measurements. Seven scenes represent architectural models, and one scene represents a complex object. Our hypothesis is that the architectural scenes typically contain triangles of more varying sizes and non-uniform distribution, and thus the potential impact of the proposed method will be higher for this type of scenes. This hypothesis is supported by the fact that SAH build algorithms typically achieve higher performance improvement on such scenes.

5.1 EMC code layout

Table 1 gives the overview of the tested scenes and shows the actual layout of the EMC-64-var encoding. We can observe that the 6 size-bits are injected at every 7-th bit. We can also see that the prefix of the code differs depending on the shape of the bounding box of the scene. For example for the Pompeii scene, the first 12 spatial splits are performed in x and z axes, which is the consequence of the *flat* structure of this large architectural scene.

5.2 EMC performance

The results comparing different Morton encoding schemes using two BVH build algorithms (LBVH, ATRBVH) are summarized in Table 2. We used the same build settings as in the paper of Domingues and Pedrini [2015] (triangle intersection cost $C_i = 1$ and node traversal cost $C_t = 1.2$) and the same ray traversal algorithm to allow for more direct comparison of our results with theirs.

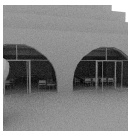
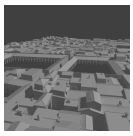
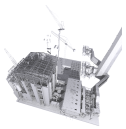
From all the possible variants of EMC codes, we chose to compare the standard 64-bit Morton codes (MC-64) with the extended Morton codes featuring regularly interleaved 16 size bits and sorting of axes (EMC-64-sort), and the extended Morton codes with variable bit count featuring all our improvements with 6 size bits (EMC-64-var). We can observe that the quality of the data structures is improved in the vast majority of cases when using the extended Morton codes. The SAH cost improvement ranges from 0% to 52% with an average of 20% in the case of LBVH, and by -2% to 26% with the average of 7% in the case of ATRBVH. The improvements in ray tracing performance follow closely the improvements in cost. The highest speedup can be seen for terrain-like scenes where the scene bounding box is far from a cubic shape.

From the measurements, we can also observe that the extended Morton codes do not increase the build time compared to standard Morton codes. In our implementation, the precomputation of the LUT for the codes is done on the CPU, but its computation time is negligible compared to the build time (about 2μs for EMC-64-sort and 50μs for EMC-64-var in a single thread). The results confirm that the extended Morton codes can significantly improve the performance of the data structures. Interestingly, they not only have a positive impact on the purely Morton code based LBVH algorithm but also on the more complex BVH build algorithm such as the ATRBVH. Although this impact is smaller than for LBVH (which was expected), it can deliver the highest quality data structures

Table 1: Properties of the tested scenes and the corresponding extended Morton code layouts for EMC-64-var encoding.

scene	triangles	norm. bounding box	X-bits	Y-bits	Z-bits	S-bits	EMC-64-var layout
							6666555555555544444444443333333333222222222211111111110000000000
#bit (0-63)							3210987654321098765432109876543210987654321098765432109876543210
sponza	76,154	(1, 0.4756, 0.4480)	20	19	19	6	xxyzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxyzxyzxyzxyzxyzxyz
conference	331179	(1, 0.2414, 0.6372)	20	18	20	6	xzxzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxyzxyzxyzxyzxyzxyz
sodahall	2,169,132	(0.9556, 1, 0.5483)	19	20	19	6	yxzyxzSyxzyxzSyxzyxzSyxzyxzSyxzyxzSyxzyxzSyxzyxzyxzyxzyxzyxzyxzy
hairball	2,880,000	(0.9908, 1, 0.9963)	19	20	19	6	yzxyzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxyzxyzxyzxyzxyzxy
pompeii	5,646,041	(1, 0.0291, 0.9740)	21	16	21	6	xzxzxzSxzxzxzSyxzyxzSyxzyxzSyxzyxzSyxzyxzSyxzyxzyxzyxzyxzyxzyxzy
san-miguel	7,880,512	(1, 0.2170, 0.3907)	20	18	20	6	zxzxzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxyzxyzxyzxyzxyzxy
city	8,737,047	(1, 0.0299, 0.7272)	21	16	21	6	xzxzxzSxzxzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxyzxyzxyzxyzxyzxyz
powerplant	12,759,246	(1, 0.4073, 0.3038)	20	19	19	6	xxyzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxyzxyzxyzxyzxyzxyz

Table 2: Overview of the results of the EMC method evaluated within a GPU ray tracing framework. The trace performance was evaluated using diffuse ray distribution with 32 secondary rays per primary ray.

	sponza	conference	sodahall	hairball	pompeii	san-miguel	city	powerplant	Avg. ratio
									
Scene size	76k	331k	2,169k	2,880k	5,646k	7,880k	8,737k	12,759k	–
LBVH - Build Time [ms]									
MC-64	5.1 (1.00)	13.0 (1.00)	65.3 (1.00)	88.5 (1.00)	158.5 (1.00)	234.4 (1.00)	261.4 (1.00)	355.3 (1.00)	(1.00)
EMC-64-sort	5.2 (1.02)	12.6 (0.97)	64.6 (0.99)	87.6 (0.99)	158.4 (1.00)	225.8 (0.96)	255.6 (0.98)	354.2 (1.00)	(0.99)
EMC-64-var	5.1 (1.00)	13.1 (1.01)	64.4 (0.99)	89.9 (1.02)	162.7 (1.03)	232.1 (0.99)	257.7 (0.99)	353.5 (0.99)	(1.00)
LBVH - BVH Cost [-]									
MC-64	97 (1.00)	65 (1.00)	101 (1.00)	542 (1.00)	180 (1.00)	94 (1.00)	121 (1.00)	49 (1.00)	(1.00)
EMC-64-sort	86 (0.88)	50 (0.78)	92 (0.91)	537 (0.99)	150 (0.84)	82 (0.87)	100 (0.83)	44 (0.90)	(0.88)
EMC-64-var	93 (0.96)	50 (0.78)	95 (0.94)	541 (1.00)	95 (0.53)	76 (0.82)	58 (0.48)	44 (0.91)	(0.80)
LBVH - Trace performance [Mrays/s]									
MC-64	75 (1.00)	80 (1.00)	113 (1.00)	7 (1.00)	19 (1.00)	19 (1.00)	30 (1.00)	25 (1.00)	(1.00)
EMC-64-sort	76 (1.01)	93 (1.17)	114 (1.01)	7 (1.02)	21 (1.11)	22 (1.12)	36 (1.18)	28 (1.13)	(1.09)
EMC-64-var	71 (0.95)	98 (1.23)	112 (0.99)	7 (1.01)	28 (1.47)	23 (1.17)	61 (2.01)	27 (1.09)	(1.24)
ATRBVH - Build Time [ms]									
MC-64	10.7 (1.00)	31.9 (1.00)	184.3 (1.00)	242.1 (1.00)	478.5 (1.00)	669.3 (1.00)	750.6 (1.00)	1029.0 (1.00)	(1.00)
EMC-64-sort	10.8 (1.01)	30.3 (0.95)	185.3 (1.01)	244.1 (1.01)	475.5 (0.99)	665.8 (0.99)	740.7 (0.99)	1023.2 (0.99)	(0.99)
EMC-64-var	10.5 (0.97)	32.2 (1.01)	182.9 (0.99)	242.3 (1.00)	479.1 (1.00)	675.0 (1.01)	713.7 (0.95)	1027.2 (1.00)	(0.99)
ATRBVH - BVH Cost [-]									
MC-64	70 (1.00)	37 (1.00)	72 (1.00)	476 (1.00)	101 (1.00)	60 (1.00)	62 (1.00)	35 (1.00)	(1.00)
EMC-64-sort	69 (1.00)	37 (0.98)	72 (1.00)	475 (1.00)	94 (0.93)	61 (1.01)	56 (0.92)	34 (0.98)	(0.98)
EMC-64-var	71 (1.02)	36 (0.98)	72 (1.00)	476 (1.00)	76 (0.75)	60 (1.00)	46 (0.74)	34 (0.99)	(0.93)
ATRBVH - Trace performance [Mrays/s]									
MC-64	97 (1.00)	123 (1.00)	144 (1.00)	7 (1.00)	28 (1.00)	27 (1.00)	57 (1.00)	37 (1.00)	(1.00)
EMC-64-sort	100 (1.02)	125 (1.02)	128 (0.89)	7 (1.01)	30 (1.04)	28 (1.02)	61 (1.07)	39 (1.05)	(1.01)
EMC-64-var	94 (0.96)	121 (0.98)	148 (1.03)	7 (1.00)	35 (1.23)	28 (1.04)	74 (1.31)	38 (1.03)	(1.07)

without increasing the build time. The best results regarding the BVH cost were achieved with the 64-bit Morton code that uses all the modifications proposed in the paper (EMC-64-var). The benefit of using fewer bits for the size component (i.e. 6 instead of 16) is especially evident on the terrain-like scenes, where the geometry is finely tessellated and the extra size bits do not help much.

The 32-bit counterpart of (EMC-64-var) (not shown in Tables) also performs very well regarding the cost and leads to lower build times. Therefore the 32-bit EMC is a very good choice for small

to medium sized scenes, for approximately 100k triangles (like sponza).

5.3 Influence of different optimizations

The proposed Morton optimizations are not orthogonal and their combined benefit is scene dependent. In some cases the optimizations support each other in other cases one of the optimizations (e.g. injecting size bits) provides most of the benefit. We provide a brief discussion of the influence of different components of EMC in combination with the LBVH algorithm.

The adaptive axis order and variable bit count provides a significant BVH cost improvement in large scenes with mostly flat structure while providing no benefit in other scenes. In particular, the adaptive axis order alone decreased the BVH cost by 12% in the pompeii scene and by 15% in the city scene. The adaptive axis order combined with the variable bit count provides further cost reduction in such scenes (44% for pompeii and 49% for city).

Injecting size bits improved the BVH cost in all tested scenes. The cost reduction ranged from 23% for conference to 10% for sodahall. In most scenes (particularly in those without the flat structure) injecting size bits was the dominant component of EMC improving the BVH cost.

The influence of using fewer size bits varies. For 32-bit encoding this method was beneficial in majority of cases as it leaves more bits for spatial sorting within the limited codes size. For 64-bit encoding it was able to further reduce the cost particularly in combination with the adaptive axis order. In some scenes (sponza, conference, powerplant) using fewer size bits led to slightly worse results than using the fix length encoding of object size. We assume that this is because the size based subdivision remains under-explored when using just a few size bits in these scenes.

5.4 Using EMC with other build algorithms

We also verified the EMC behavior with other BVH build algorithms in an experimental CPU-based ray tracing framework. We tested the EMC method within CPU implementations of LBVH [Lauterbach et al. 2009], HLBVH [Garanzha et al. 2011], and AAC [Gu et al. 2013] methods. We also compared the EMC based build algorithms with respect to full sweep SAH which does not use Morton code based build and provides a high-quality reference.

On average the EMC-64-var improves the SAH cost by 20% for LBVH, which is the same result as achieved for the GPU implementation (Table 2). For the AAC method, the EMC-64-var encoding improves the BVH cost by 11% on average. Finally, for HLBVH it improves the BVH cost by 16%.

The LBVH build algorithm with EMC-64-var encoding leads to BVH with the cost just 20% higher than the much slower full-sweep SAH build (taking the average ratio for all tested scenes). Similarly, the AAC build algorithm yields 3% higher BVH cost and the HLBVH build algorithm yields 11% higher cost than full-sweep SAH. In a number of test scenes, the improvement in SAH cost provided by EMC is actually enough to bridge the gap between the fast build algorithm (LBVH, AAC, HLBVH) and the full-sweep SAH build algorithm. This suggests that most of the overlap between bounding boxes in a BVH can be prevented already in the precomputation phase when looking at individual primitives independently.

6 CONCLUSIONS

We proposed a method to extend the Morton code for spatial sorting of scene primitives. The extended Morton code allows for improving the quality of bounding volume hierarchies for ray tracing constructed using the Morton codes as the primitive sorting step. In fact, simple highly parallel build algorithms such as LBVH can then build hierarchies which in terms of quality get closer to the high-quality methods such as the binned or sweep-based top down SAH build algorithms. We evaluated the impact of different components of the proposed method on two fast BVH build algorithms (LBVH, ATRBVH) on a GPU and showed that without modifying the build algorithms themselves, significant improvements in data structure quality can be achieved. We also evaluated the method in CPU implementations of high-performance BVH build algorithms (LBVH, HLBVH, AAC) where the EMC also shows its benefits in most test cases.

In future work, it would be interesting to embed other heuristics into the bit codes for even higher quality, and to test the other search algorithms than ray tracing with hierarchies built with the extended Morton code.

REFERENCES

Ciprian Apetrei. 2014. Fast and Simple Agglomerative LBVH Construction. In *Computer Graphics and Visual Computing (CGVC)*.

Jiri Bittner, Michal Hapala, and Vlastimil Havran. 2013. Fast Insertion-Based Optimization of Bounding Volume Hierarchies. *Computer Graphics Forum* 32, 1 (2013), 85–100.

Holger Dammertz, Johannes Hanika, and Alexander Keller. 2008. Shallow Bounding Volume Hierarchies for Fast SIMD Ray Tracing of Incoherent Rays. *Computer Graphics Forum* 27 (2008), 1225–1233(9).

Leonardo R. Domingues and Helio Pedrini. 2015. Bounding Volume Hierarchy Optimization through Agglomerative Treelet Restructuring. In *Proceedings of the 7th Conference on High-Performance Graphics*. 13–20.

Kirill Garanzha, Jacopo Pantaleoni, and David McAllister. 2011. Simpler and Faster HLBVH with Work Queues. In *Proceedings of Symposium on High Performance Graphics*. 59–64.

Yan Gu, Yong He, Kayvon Fatahalian, and Guy Blelloch. 2013. Efficient BVH Construction via Approximate Agglomerative Clustering. In *Proceedings of the 5th Symposium on High-Performance Graphics*. 81–88.

Vlastimil Havran, Robert Herzog, and Hans-Peter Seidel. 2006. On the Fast Construction of Spatial Data Structures for Ray Tracing. In *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006* (18-20). 71–80.

Tero Karras. 2012. Maximizing Parallelism in the Construction of BVHs, Octrees, and K-d Trees. In *Proceedings of the 4th Symposium on High-Performance Graphics*. 33–37.

Tero Karras and Timo Aila. 2013. Fast Parallel Construction of High-quality Bounding Volume Hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*. 89–99.

Andrew Kensler. 2008. Tree Rotations for Improving Bounding Volume Hierarchies. In *Proceedings of the Symposium on Interactive Ray Tracing*. 73–76.

Daniel Kopta, Thiago Ize, Josef Spjut, Erik Brunvand, Al Davis, and Andrew Kensler. 2012. Fast, Effective BVH Updates for Animated Scenes. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*. 197–204.

Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. 2009. Fast BVH Construction on GPUs. *Comput. Graph. Forum* 28, 2 (2009), 375–384.

G. M. Morton. 1966. A Computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. In *Research Report, IBM Ltd., Ottawa, ON, Canada.*

Ulises Olivares, Hector G. Rodriguez, Arturo Garcia, and Felix F. Ramos. 2016. Efficient construction of bounding volume hierarchies into a complete octree for ray tracing. *Computer Animation & Virtual Worlds* 27, 3-4 (2016), 358–368.

Jacopo Pantaleoni and David Luebke. 2010. HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry. In *Proceedings of the Conference on High Performance Graphics*. Eurographics, 87–95.

Hanan Samet. 2005. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Ingo Wald. 2007. On Fast Construction of SAH-based Bounding Volume Hierarchies. In *Proceedings of the Symposium on Interactive Ray Tracing*. 33–40.

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. 2014. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Trans. Graph.* 33, 4, Article 143 (2014), 8 pages.

Bruce Walter, Kavita Bala, Milind Kulkarni, and Keshav Pingali. 2008. Fast Agglomerative Clustering for Rendering. In *IEEE Symposium on Interactive Ray Tracing (RT)*. 81–86.

Dominik Wodniok and Michael Goesele. 2017. Construction of bounding volume hierarchies with SAH cost approximation on temporary subtrees. *Computers & Graphics* 62 (2017), 41 – 52.

ACKNOWLEDGMENTS

This research was supported by SAMSUNG Electronics Co., Ltd., under the project "Fast BVH Tree Build for Dynamic Ray Tracing on Mobile Environment". We would also like to acknowledge the contributors of the scenes used in our measurements. Marko Dabrovic for the Sponza model, Greg Ward for the Conference model, Prof. Carlo Séquin for the Sodahall model, Samuli Laine and Tero Karras for the Hairball model, Guillermo M. Leal Llaguno for the San Miguel model, and the University of North Carolina for the Power Plant model. Finally we would like to thank Tero Karras, Timo Aila, and Samuli Laine as well as Leonardo R. Domingues and Helio Pedrini for releasing their GPU ray tracing frameworks to public.