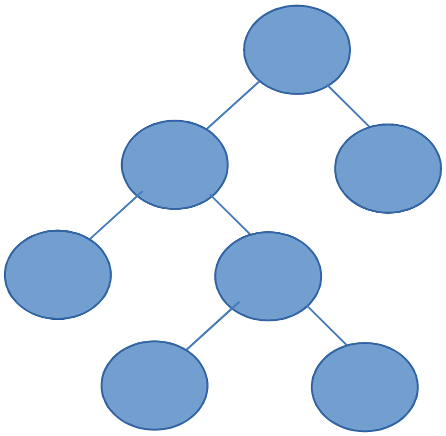
Czech Technical University in Prague
Faculty of Electrical Engineering
DCGI
DEPARTMENT OF COMPUTER GRAPHICS AND INTERACTION

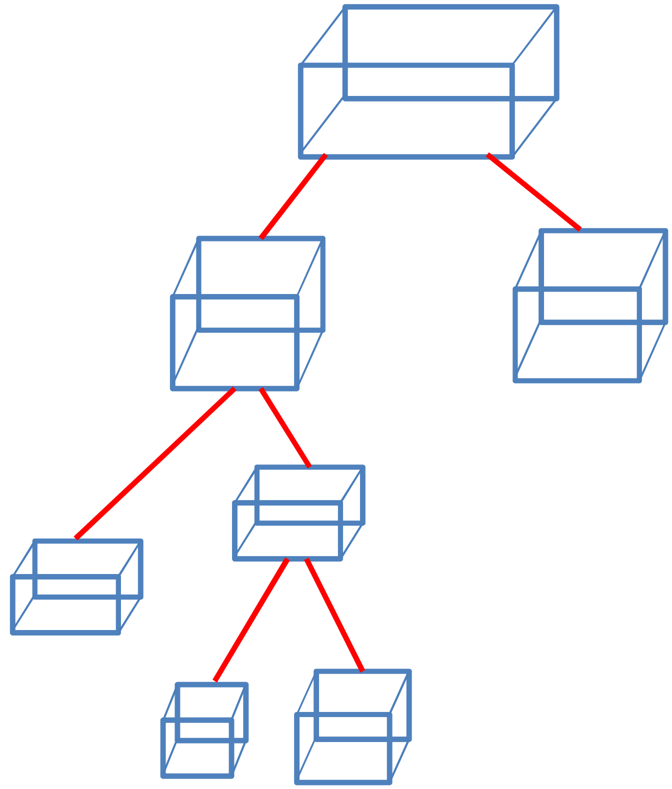
Extended Morton Codes for High Performance Bounding Volume Hierarchy Construction

Marek Vinkler, Jiri Bittner, Vlastimil Havran

BVH – Old but Good Concept of Hierarchy

BVH = bounding volume hierarchy





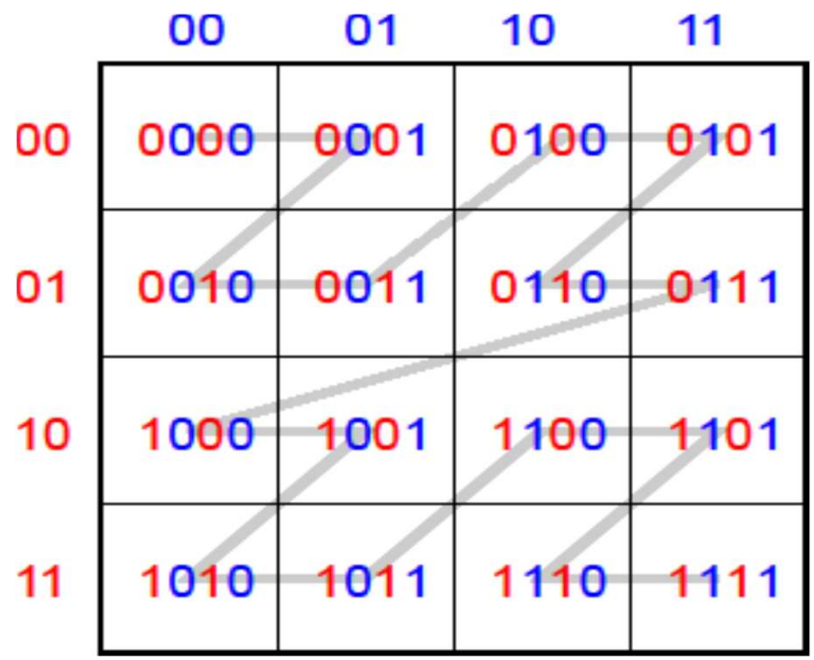
Selected Related Work on BVHs

- [Rubin and Whitted 1980] – first paper, top-down method
- 1980-2005 not so many papers e.g. [Goldsmith/Salmon 87]
- [Havran et al. 2006] – BVH binning+LSD trees
- [Wald 2007] – vertical/horizontal parallel BVH build
- [Walter et al. 2008] – agglomerative BVH build
- [Kensler 2008] – BVH optimization by rotations
- [Lauterbarch et al. 2009] – LBVH - fast build with Morton code
- [Pantaleoni and Luebke 2010] – hierarchical LBVH (HLBVH)
- [Garanzha et al. 2011] – another improvement HLBVH
- [Karras 2012] – faster parallel LBVH build
- [Gu et al. 2013] – efficient approx. agglomerative clustering
- [Bittner et al. 2013] – BVH optimization by insertion
- [Apetrei 2014] – Agglomerative LBVH construction
- [Domingues and Pedrini 2015] – optimization by treelet restructuring
- Etc.

Morton Codes

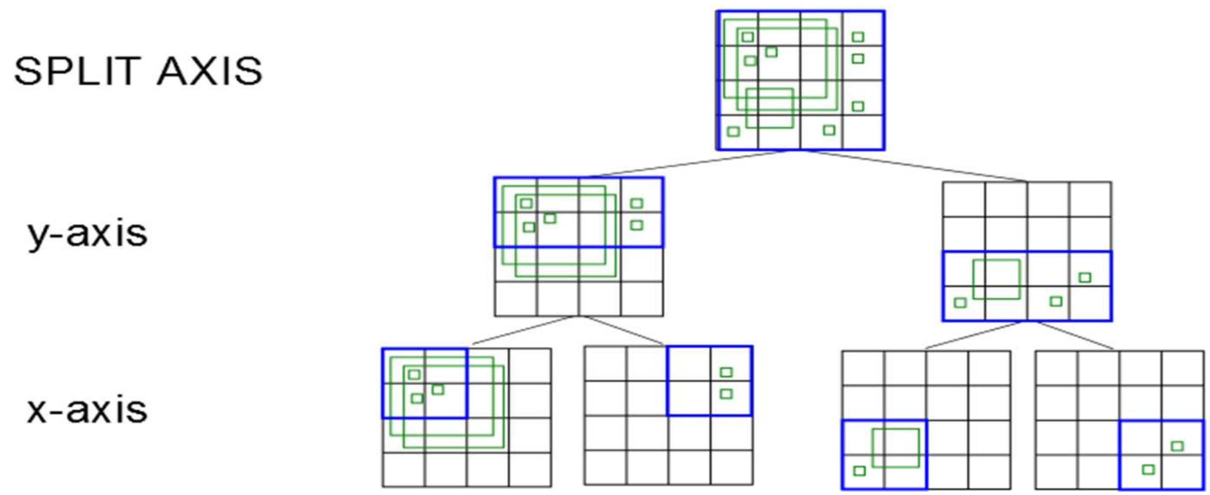
- [G.M. Morton 1966] – A Computer Oriented Geodetic Data Base: and a New Technique in File Sequencing, Research Report IBM Ltd., Ottawa, ON, Canada
- Simply regularly interleaving bits of spatial coordinates
- 2D example:

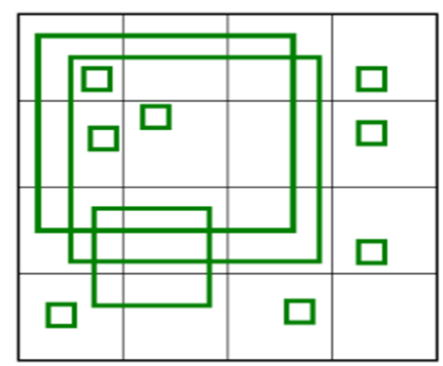
$\color{red}{y_2}\color{blue}{x_2}\color{red}{y_1}\color{blue}{x_1}\color{red}{y_0}\color{blue}{x_0}$



Morton Codes+Ray Tracing $y_2 x_2 y_1 x_1 y_0 x_0$

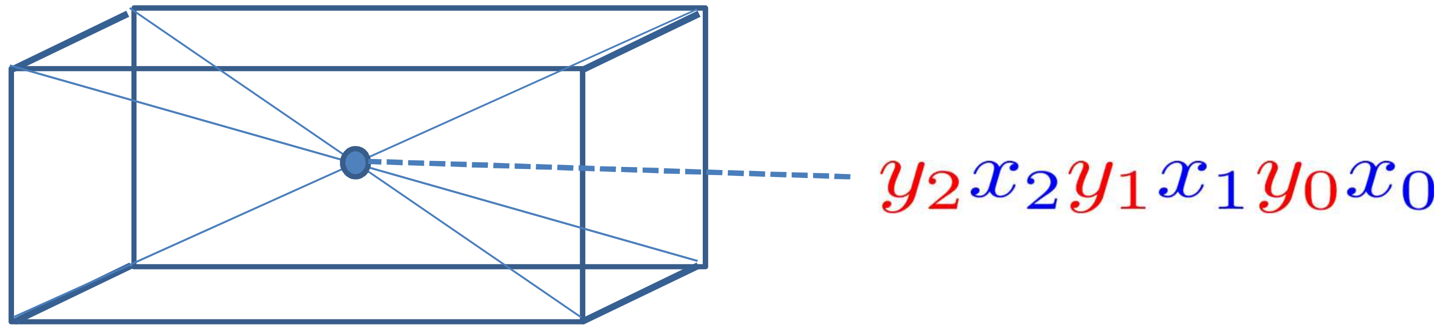
- Direct implementation paper by [Lauterbach et al. 2009]





Morton Codes+Ray Tracing $y_2x_2y_1x_1y_0x_0$

- Algorithm summary – direct build up
  - For each primitive assign the code according to the spatial coordinates of the center of the box around the primitive



  - Sort the primitives according to their Morton Codes
  - Build the BVH by the changes in the bits of the code from root to the leaves
  - Possibly postprocess BVH (e.g. more primitives in a leaf)

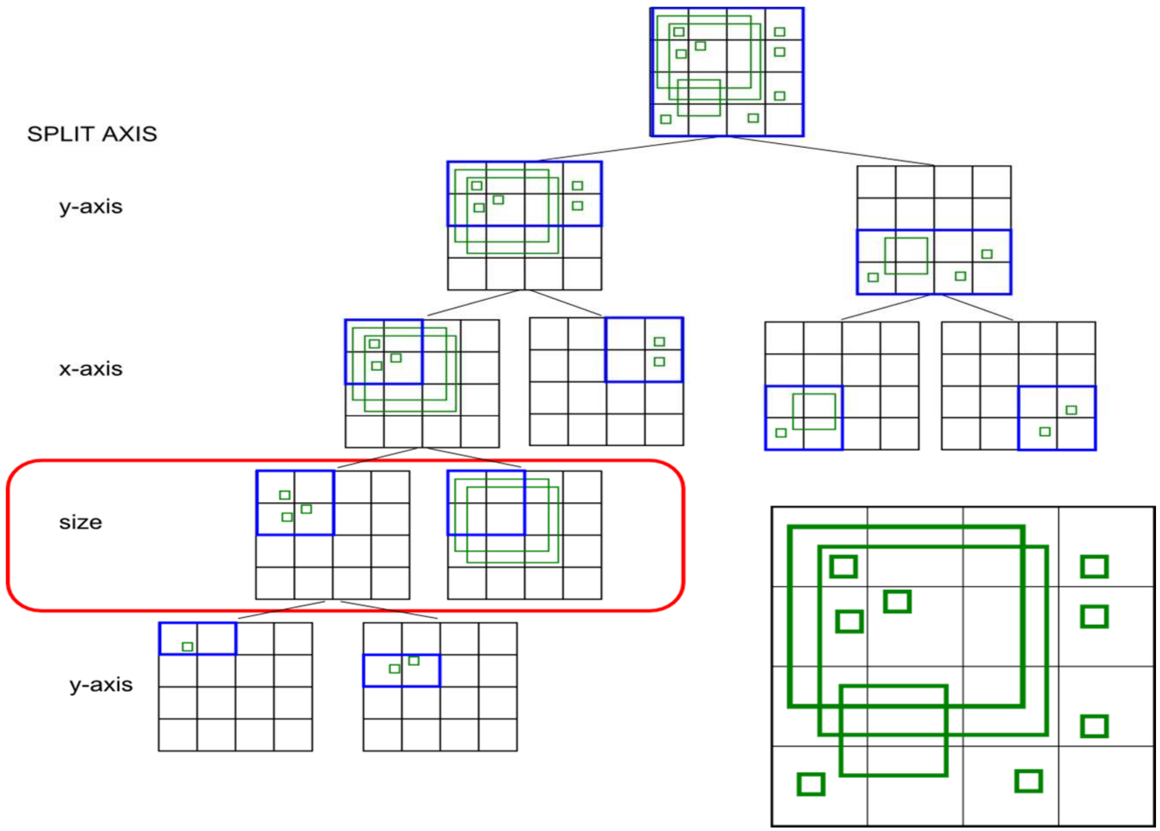
Morton Code (MC) Properties

- Relatively efficient for fast build
- Simple and clear
- Quality is the problem for scenes with non-uniform distribution
- Useful for ray tracing data structures

- **Question:** can we improve on the properties of BVH built with the Morton Code?

1st Idea – Encode Object Size $yxsy$

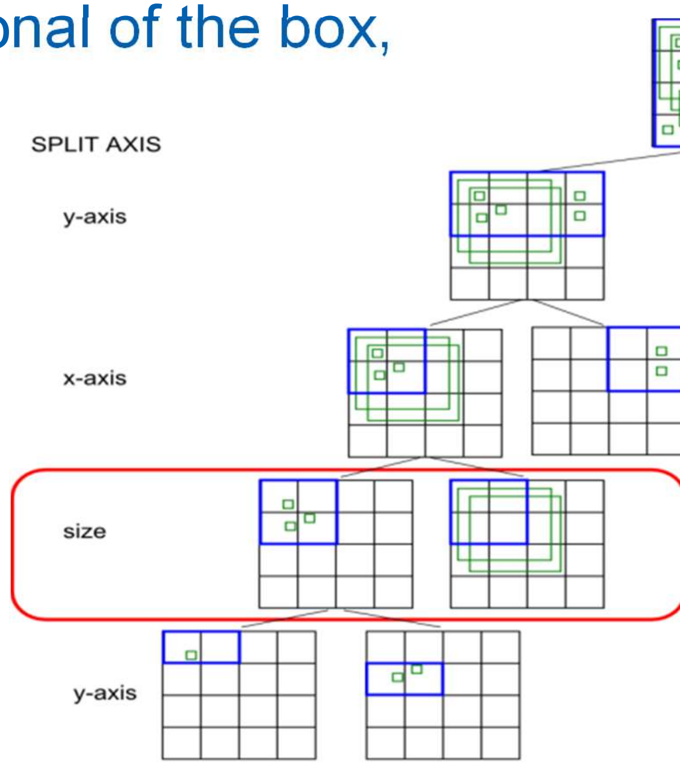
- Morton code designed to interleave bits of spatial coordinates X,Y … Z, we can however encode other properties than only spatial coordinates
- **Separate objects by their SIZE**

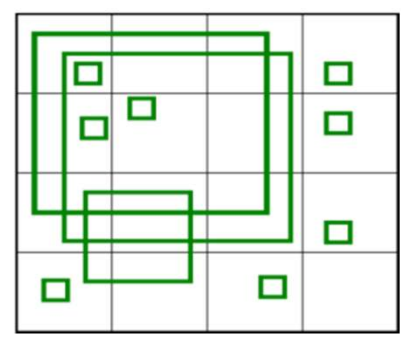


$yxsy$

1st Idea – Encode Object Size $yxsy$

- Size of object and spatial coordinates have to be related!
- Coordinates are normalized for MC before encoding
- We encode the diagonal of the box, its square root





2^{nd} Idea – Fewer Bits for Size $yxsyxyx$

- We can distribute the bit unevenly between coordinates and size, e.g. put less bits to the encoding of size
  - 3 times X
  - 3 times Y
  - 1 times SIZE

- Spatial size encoding the box is not that important as the spatial coordinates

$$yxsyxyx$$

- This allows to use also 32 bit long EMC codes for scenes with up to 200k primitives

3rd Idea – Adaptive Axis Order in 3D

- We can start with another axis than just with x-axis, according to the longest side of the box, the second longest side, only six possibilities: xyz, yzx, zxy, xzy, zyx, yxz

- Note: motivation comes from observation of fast SAH build kd-trees algorithms, use the longest side axis in X% cases and otherwise continue round robin, here globally for the whole tree.

4th Idea – Variable Bit Count $xxyszxxyz$

- We can distribute the bit unevenly between coordinates, e.g.
  - 4 times X
  - 2 times Y
  - 2 times Z
  - 1 times S

$$xxyszxxyz$$

Algorithm Example for EMC

```
function INIT(scene)
    a0 = x, a1 = y, a2 = z, a3 = size ▷ default axis order
    ▷ descending axis order using scene dimensions
    sort(a0..2)
    ▷ Compute quantization scales
    s0 = 2^16/scene.box.size[a0]
    s1 = 2^16/scene.box.size[a1]
    s2 = 2^16/scene.box.size[a2]
    s3 = 2^16/scene.box.diagonal.length
end function
function EXPAND((integer X))
    integer v=0, mask=1
    for i=0 to 15 step 1 do
        v = v | ((X & mask) ≪(3.i))
        mask = mask ≪1
    end for
    return v
end function
function CODE(triangle)
    v = triangle.box.center-scene.box.min
    size = triangle.box.diagonal.length
    integer v0* = s0 · v[a0]
    integer v1* = s1 · v[a1]
    integer v2* = s2 · v[a2]
    integer v3* = s3 · size
    return ((Expand(v0*) ≪ 3 ) |  (Expand(v1*) ≪ 2)
              | (Expand(v2*) ≪ 1 ) |  Expand(v3*))
end function
```

More complete pseudocode in the paper and project webpage contains real code

http://dcgi.fel.cvut.cz/projects/emc

Extended Morton Code – Ideas Summary

- Standard Morton Code – interleaving bits regularly

$$xyzxyzxyzxyz$$

- Extended Morton Code – break of regularity assumption

$$xxyszyxxyszx$$

  - Encoding size of the object
  - Variable bit count for each dimension, including size of box
  - Variable bit count and dimensions order, considering the shape of original scene box

Scenes of Various Size – 100k to 13M

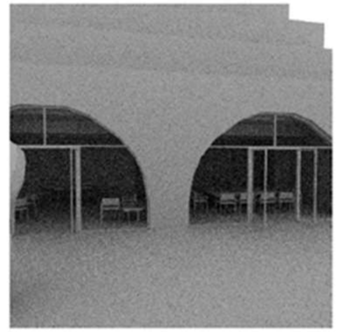
SC1
sponza

76k

SC2
conference

331k

SC3
sodahall

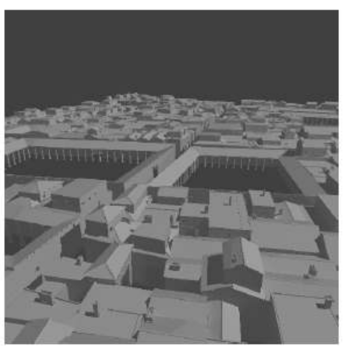


2,169k

SC4
hairball

2,880k

SC5
pompeii



5,646k

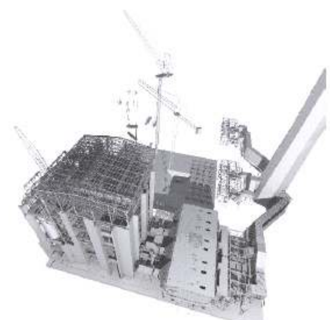
SC6
san-miguel

7,880k

SC7
city

8,737k

SC8
powerplant



12,759k

Results – Example Codes for 8 Scenes

Bit 63 Bit 32 Bit 0

#bits
x/ y/ z/ s

SC1 20/19/19/6
SC2 20/18/20/6
SC3 19/20/19/6
SC4 19/20/19/6
SC5 21/16/21/6
SC6 20/18/20/6
SC7 21/16/21/6
SC8 20/19/19/6

```
6666555555555544444444443333333333222222222211111111110000000000
3210987654321098765432109876543210987654321098765432109876543210
----------------------------------------------------------------
xxyzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxyzxyzxyzxyzxyzxyz
xzxzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxySzxyzxyzxyzxyzxyzxyzxyz
yxzyxzSyxzyxzSyxzyxzSyxzyxzSyxzyxzSyxzyxzSyxzyxzyxzyxzyxzyxzyxzy
yzxyzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxyzxyzxyzxyzxyzxy
xzxzxzSxzxzxzSyxzyxzSyxzyxzSyxzyxzSyxzyxzSyxzyxzyxzyxzyxzyxzyxzy
zxzxzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxSyzxyzxyzxyzxyzxyzxyzxy
```

EMC-64-var code

Numbers for LBVH (lower quality), GPU

- reference – standard Morton code, rendering time/cost
- EMC-64 against MC-64, two variants

	76k	331k	2169k	2880k	5646k	7880k	8737k	12759k	
	SC1	SC2	SC3	SC4	SC5	SC6	SC7	SC8	
Build	5.1	13.0	65.3	88.5	158.5	234.4	261.4	355.3	[ms]
sort									**AVG**
cost	-13%	-22%	-9%	-1%	-16%	-13%	-17%	-10%	**-12%**
Mrays/s	+1%	+17%	+1%	+2%	+11%	+12%	+18%	+13%	**+7.5%**
var									
cost	-4%	-22%	-6%	+0%	-47%	-18%	-52%	-9%	**-27%**
Mrays/s	-5%	+23%	-1%	+1%	+47%	+17%	+101%	+9%	**+19%**

Numbers for ATRBVH (higher quality), GPU

- reference – standard Morton code, rendering time/cost
- EMC-64 against MC-64, two variants

	76k	331k	2169k	2880k	5646k	7880k	8737k	12759k	
	SC1	SC2	SC3	SC4	SC5	SC6	SC7	SC8	
Build	10.7	31.9	184.3	242.1	478.5	669.3	750.6	1029.0	[ms]
sort									**AVG**
cost	+0%	-2%	+0%	+0%	-7%	+1%	-8%	-2%	**-2%**
Mrays/s	+2%	+2%	-11%	+1%	+4%	+2%	+7%	+5%	**+1%**
var									
cost	+2%	-2%	+0%	+0%	-25%	+0%	-26%	-1%	**-7%**
Mrays/s	-4%	-2%	+3%	+0%	+23%	+4%	+31%	+3%	**+7%**

Conclusions on Extended Morton Code

- Useful anywhere Morton Code is used for spatial objects (CPU, GPU, …)
- Requires only to change the code, the rest of algorithm the same as it was (LBVH, HLBVH, AAC, ATRBVH …)
- It is a **heuristic**, does not guarantee the improvement for more sophisticated build algorithms
- Reviewer #4 reimplemented the method in 2 hours with similar results
- Extension on Morton Code summary
  - uses spatial size of object – e.g. box diagonal
  - variable bit count for each axis
  - irregular structure of the code
- Improvement for 8 scenes on GPU +7% for free
- Possible HW implementation perhaps easy

Acknowledgments

- Thanks for scenes: M. Dabrovic (Sponza model), G. Ward (conference model), Prof. Sequin (Sodahall), S. Laine and T. Karras (Hairball scene), G. Llaguno (San Miguel), UNC (Powerplant model),
- Eric Haines for challenge during SIGGRAPH 2002
- T. Karras, T. Aila, S. Laine for 1st GPU ray tracing framework,
- L. Dominges and H. Pedrini for 2nd GPU ray tracing framework,
- Samsung Electronics Co. Ltd. for financial motivation and support, patent pending U.S. Application No. 15/403,612 filed on January 11, 2017

- Project webpage including essential source code

  http://dcgi.fel.cvut.cz/projects/emc

DCGI