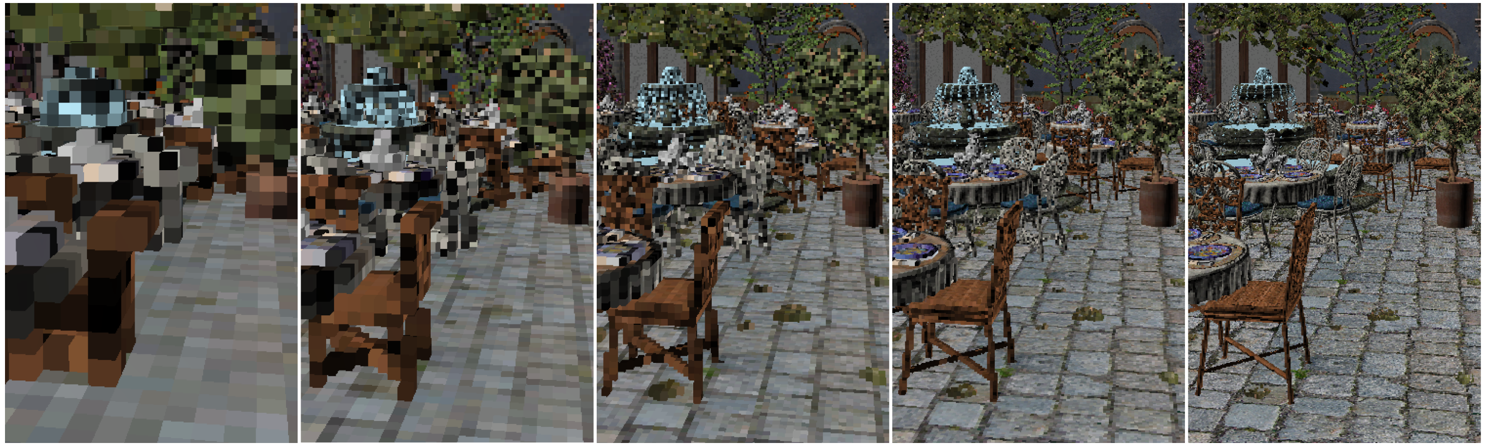


Grid-Free Out-Of-Core Voxelization to Sparse Voxel Octrees on GPU

Martin Pätzold and Andreas Kolb,
Computer Graphics and Multimedia Systems Group,
University of Siegen

UNIVERSITÄT SIEGEN

CG

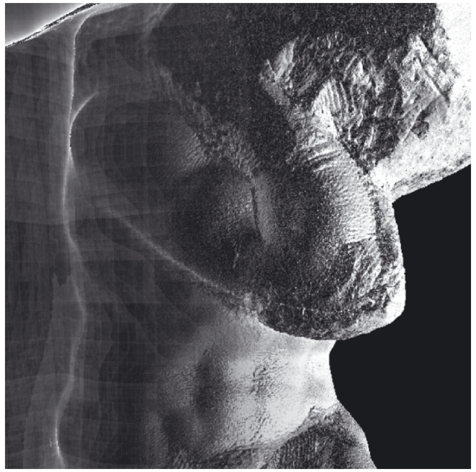
Imaging New Modalities

Motivation

- **Sparse Voxel Octrees** (SVOs) are promising to represent massively large and detailed scenes
- Exploit the **performance** of the GPU and allow an **out-of-core** voxelization with sophisticated **attribute** creation

[Crassin & Green 2012]

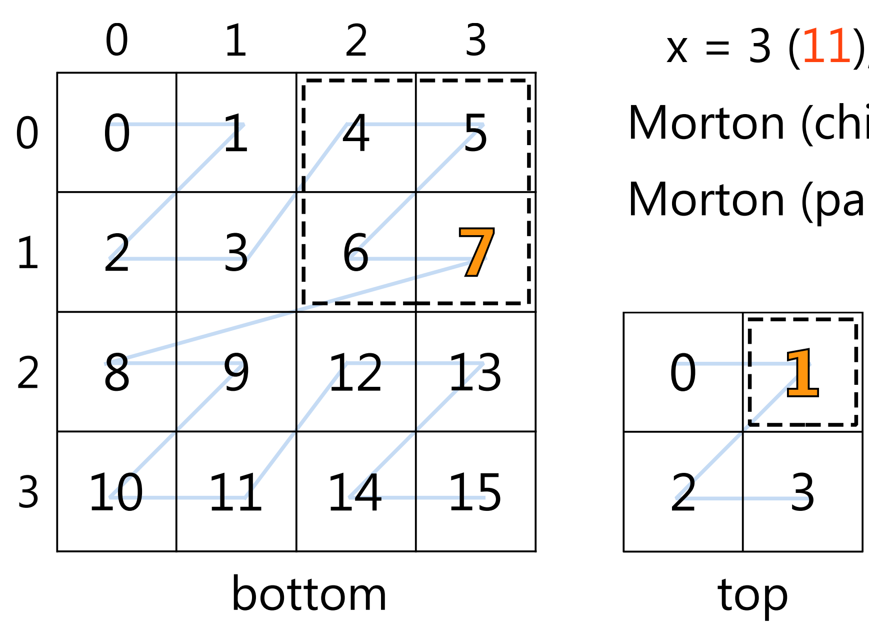
[Laine & Karras 2010+2011]



[Baert et al. 2013+2014]

Main Question

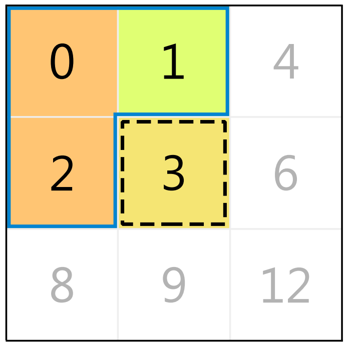
- How do we achieve a performant **out-of-core** processing that uses parallelism of **GPU**? → **stream batches** (subsets) of triangles & voxels
- Triangles need to be **sorted** in the **same order** as nodes of the SVO are created
  → **Morton** order maps multi-dimensional data to **linear index** and preserves **locality** for SVO-creation



Bit-interleaving (2D):
x = 3 (11), y = 1 (01)
Morton (child) = 7 (0111)
Morton (parent) = 1 (7/4)

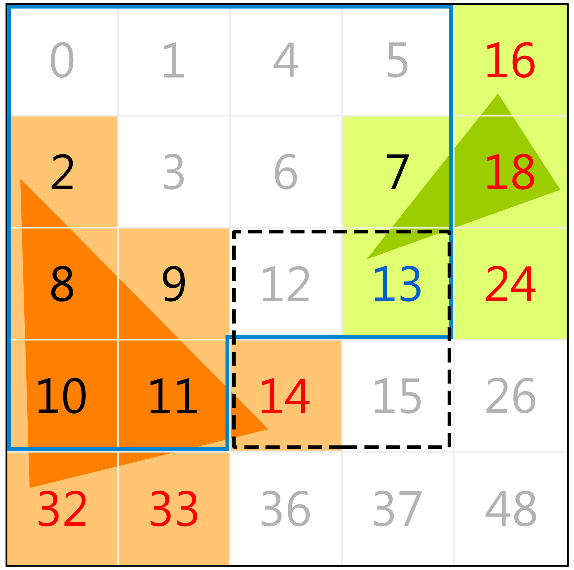
More Questions

- **Which SVO nodes** can be created?
- **Where** do we need a **triangle first**?
  → determine Morton indices



top

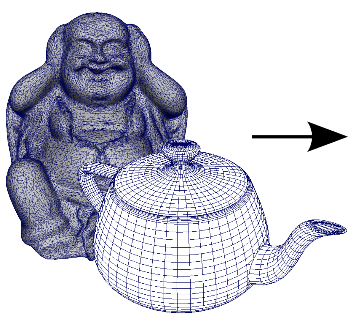
- For **efficient CPU/GPU-transfer**, each triangle should be processed only once: What to do with **unprocessable voxels**?
- How do we create **parent attributes for incomplete child nodes**?
  → store them for later processing



bottom

Overview

- Out-of-core voxelization approaches require a **streamed** processing of **triangles and voxels**

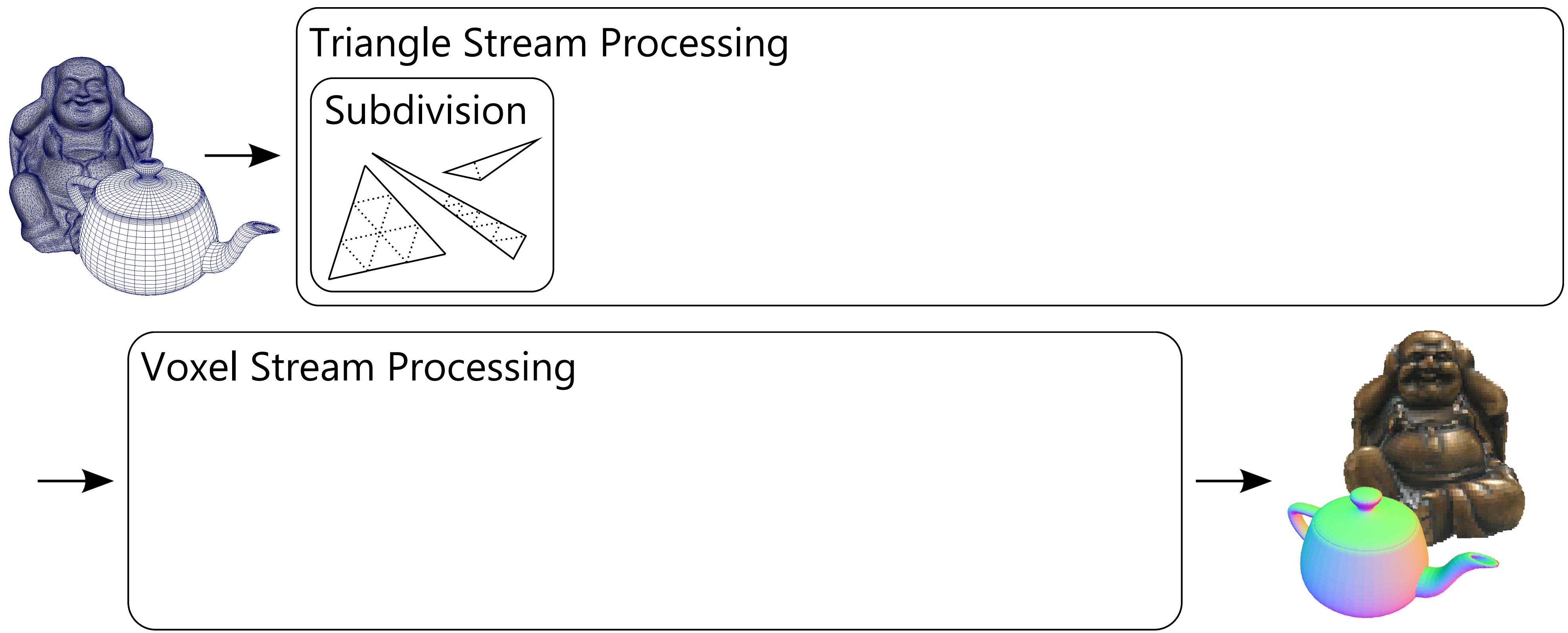


Triangle Stream Processing

Voxel Stream Processing

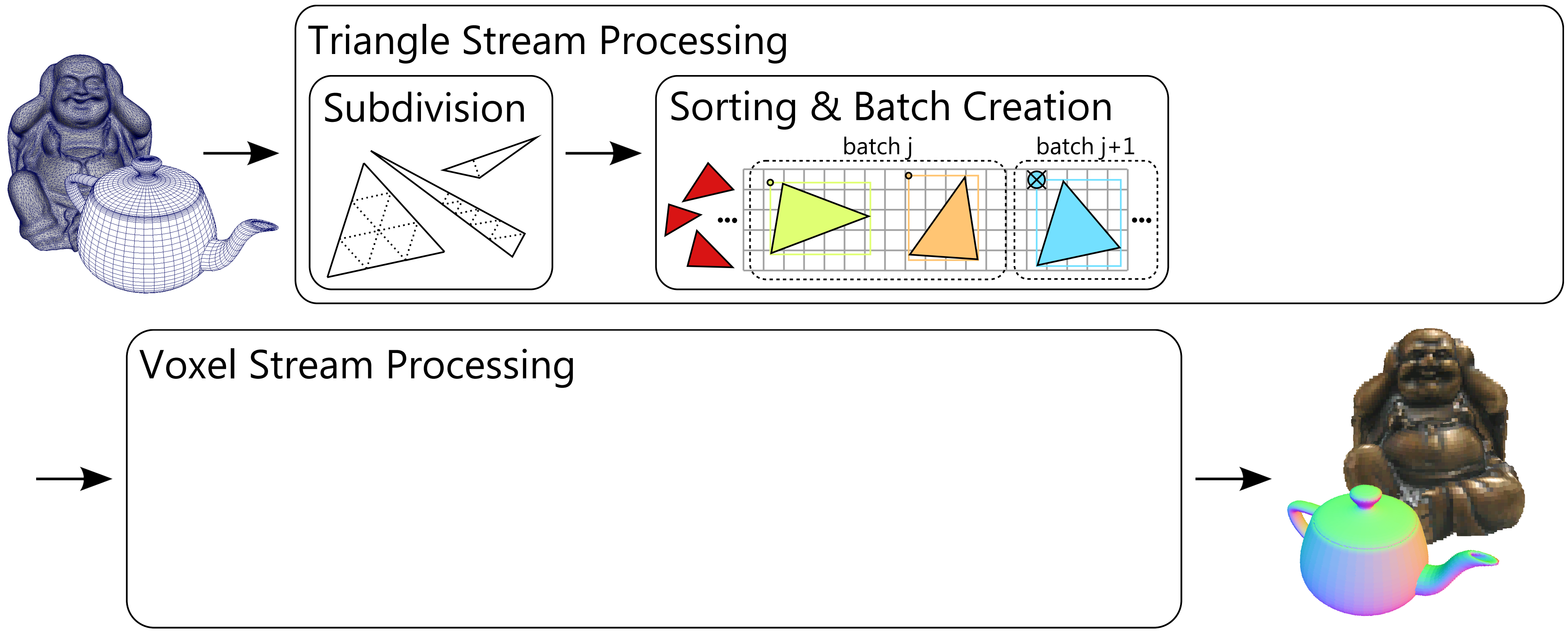
Overview

- Optimized processing on GPU needs a **workload balancing** depending on the **created voxels per triangle**



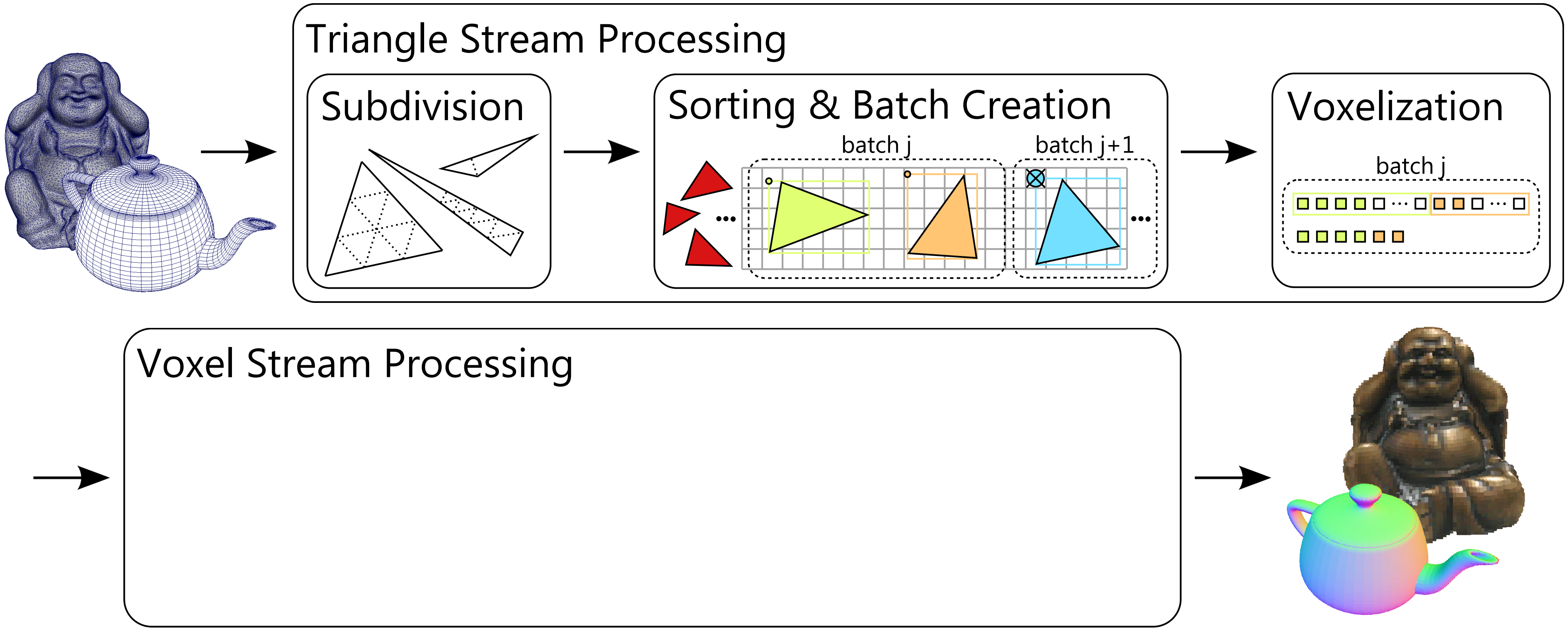
Overview

- Creating **triangle batches** that the GPU can handle at once
- **Sequential** process requires a triangle **order** for voxel streaming



Overview

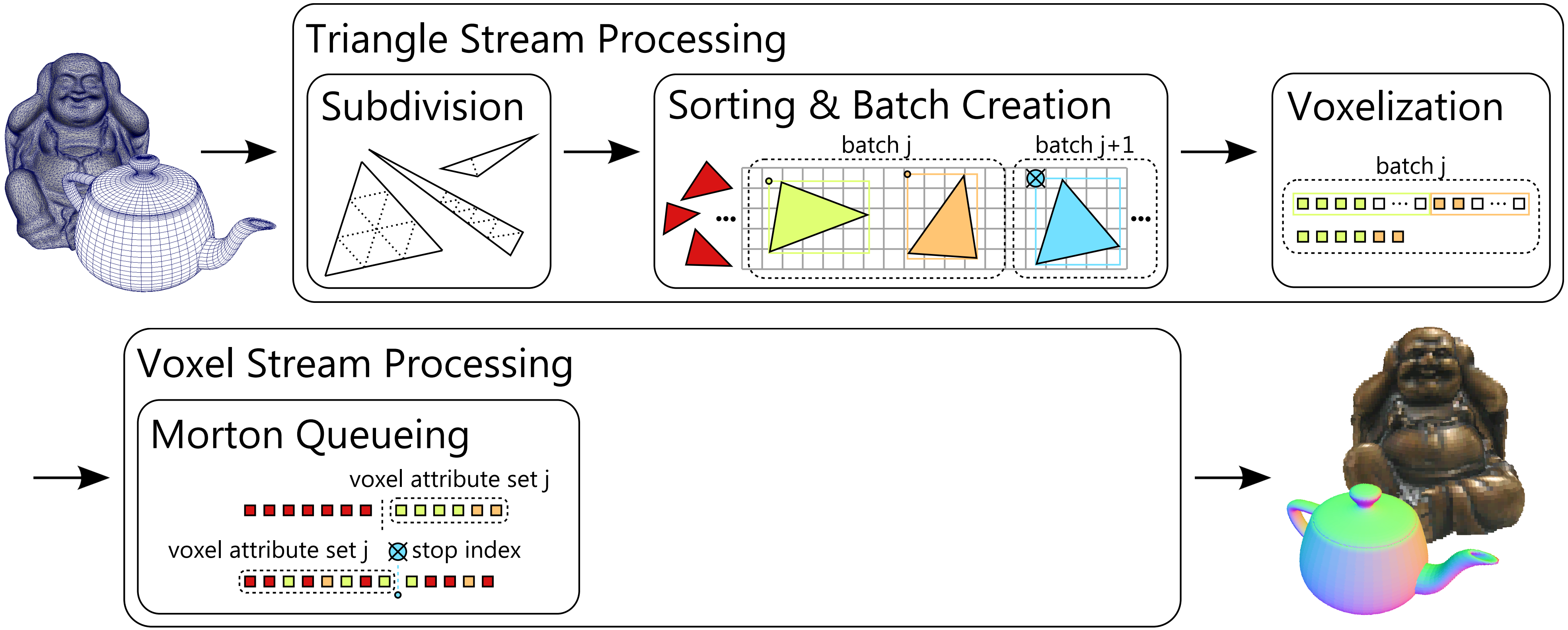
- Creating **voxel attribute sets** from the current triangle batch
- **Predicted** number of voxels per triangle → **no atomic** operation



Overview

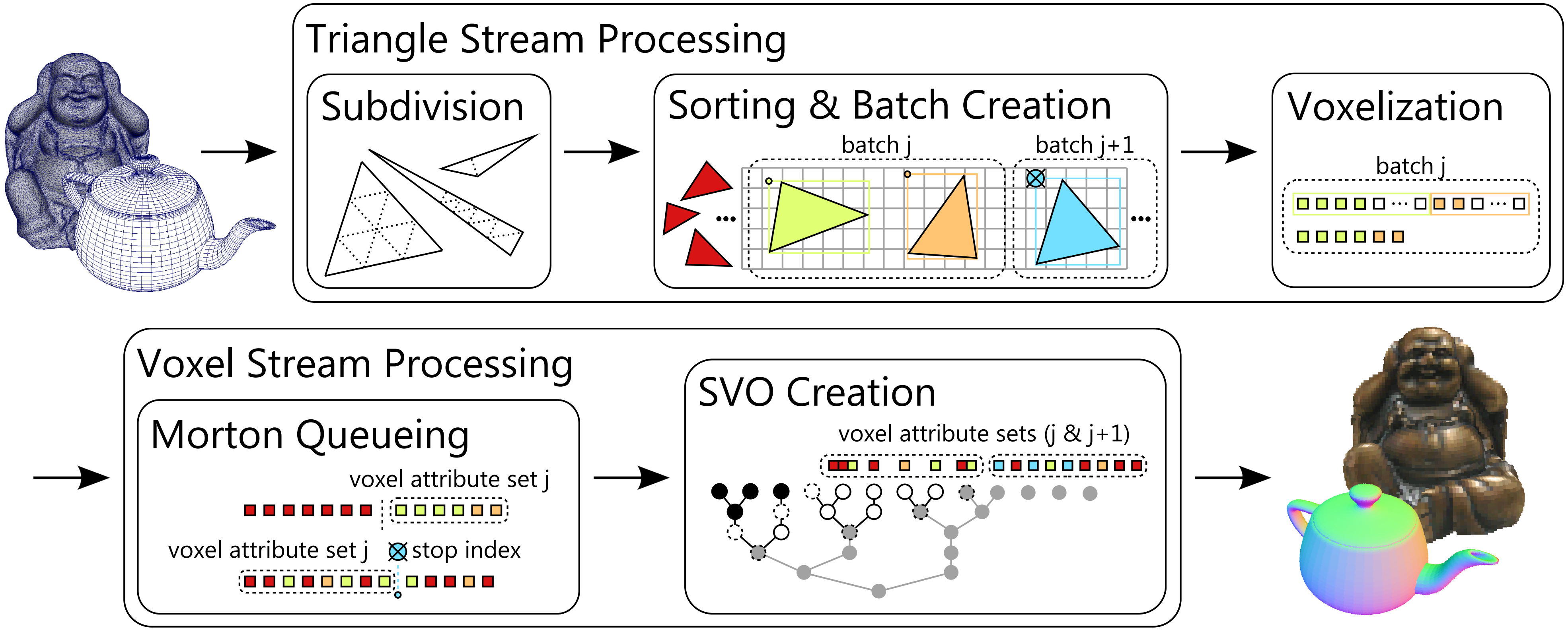
- **Not all voxels** will be **processable** for streamed SVO creation

→ **Store** voxels between iterations and **extract** processable voxels



Overview

- Current voxel attribute set is used for a **bottom-up** creation of **parts** of the **SVO** by **parallel compaction** methods

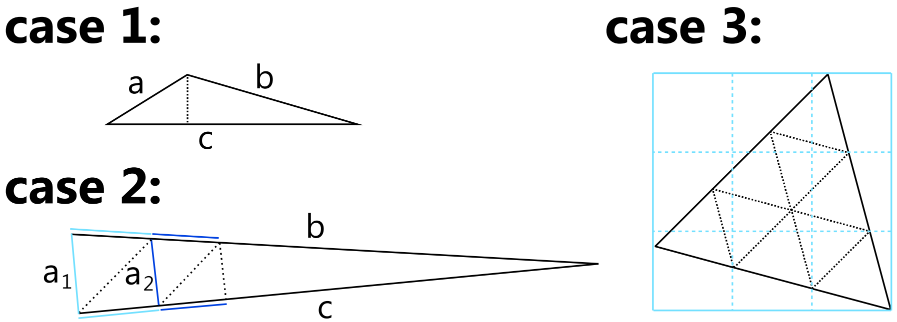


Subdivision of Triangles

- First step consists of a „homogenization“ of triangles
  - **Size** limit → **balanced workload** on GPU (1 triangle per thread)
  - **Locality** of Morton ranges → **limit voxels** that need to be maintained over the sequential batch iterations
- Apply **subdivision rules** if equation below is not fulfilled
- 3 cases: long thin triangles (angles: >90°, <20°), all other triangles

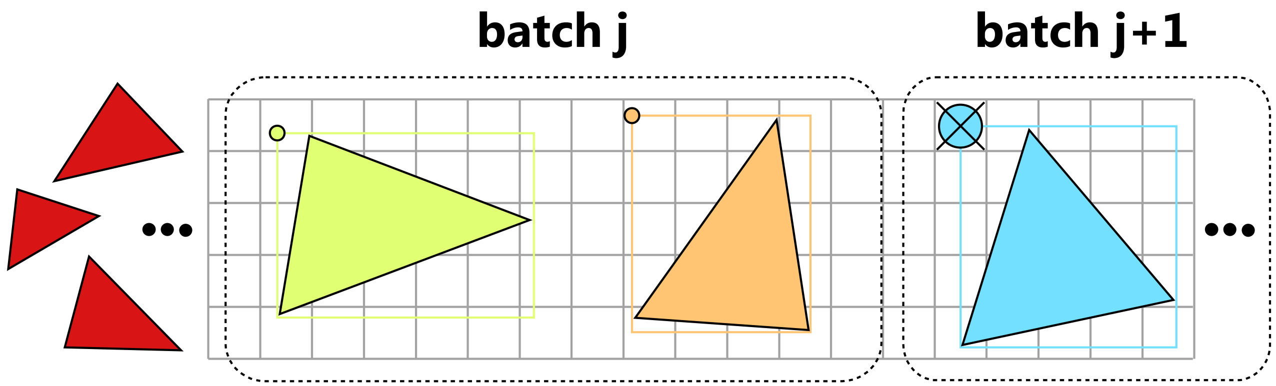
$$A_{\text{bbox}}(T_i) \leq K^{\text{max}}_{\text{vox/tri}} \cdot A_{\text{vox}}$$

- $A_{\text{bbox}}(T_i)$: area of largest face of bbox
- $K^{\text{max}}_{\text{vox/tri}}$: user-defined value
- A_{vox}: area of a voxel face



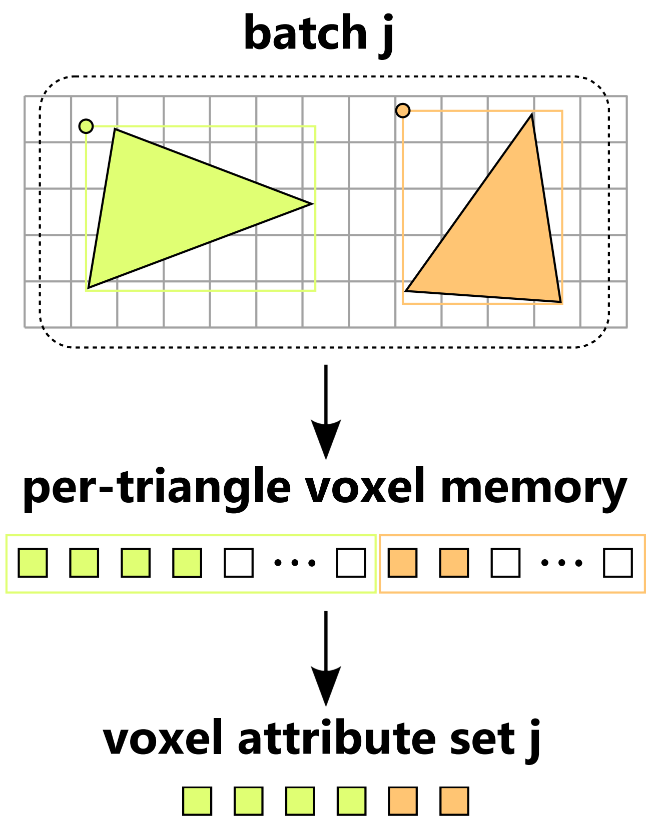
Sorting & Batch Creation

- Sort triangles according to the minimum Morton index of their bounding boxes → **earliest possible need for a triangle**
- Create batches according to voxel count prediction → **processable triangles per iteration** by GPU (max. voxel count as user-defined value)
- Store minimum Morton index of 1st triangle of next batch → **valid Morton range** for creation of SVO-nodes



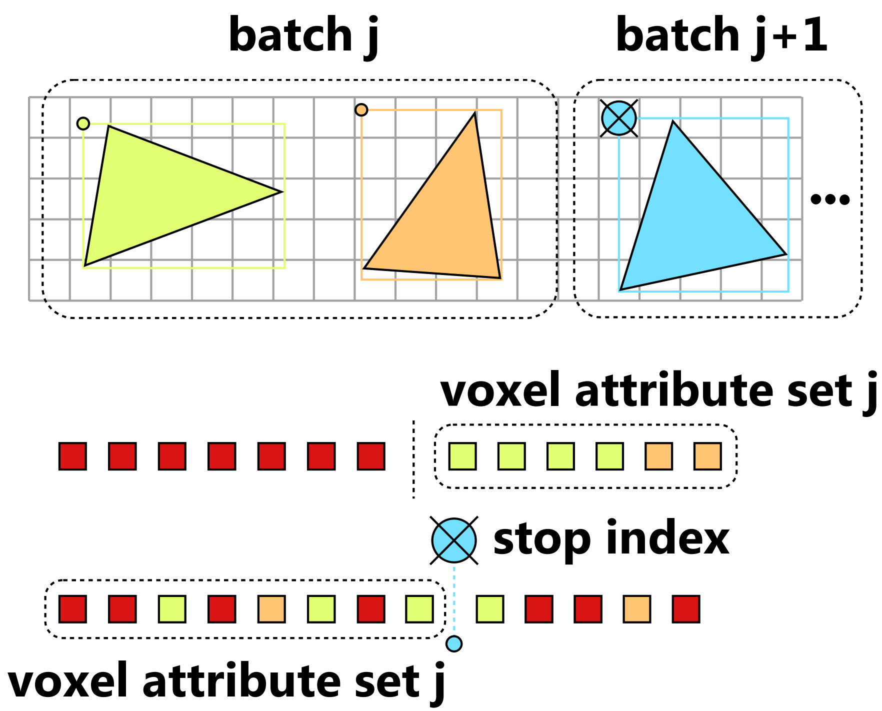
Voxelization

- Triangle batch is voxelized to a **„per-triangle voxel memory"** (offsets given by prediction) → **no atomic** operations
- Method of Schwarz and Seidel [2010]
  - Each thread processes one triangle
  - Conservative surface voxelization
- **Attribute creation** in the same step → project voxel center to **uv-coords**.
- Set of valid voxel-attribute pairs is obtained by **removing placeholders** and copied to Morton queue



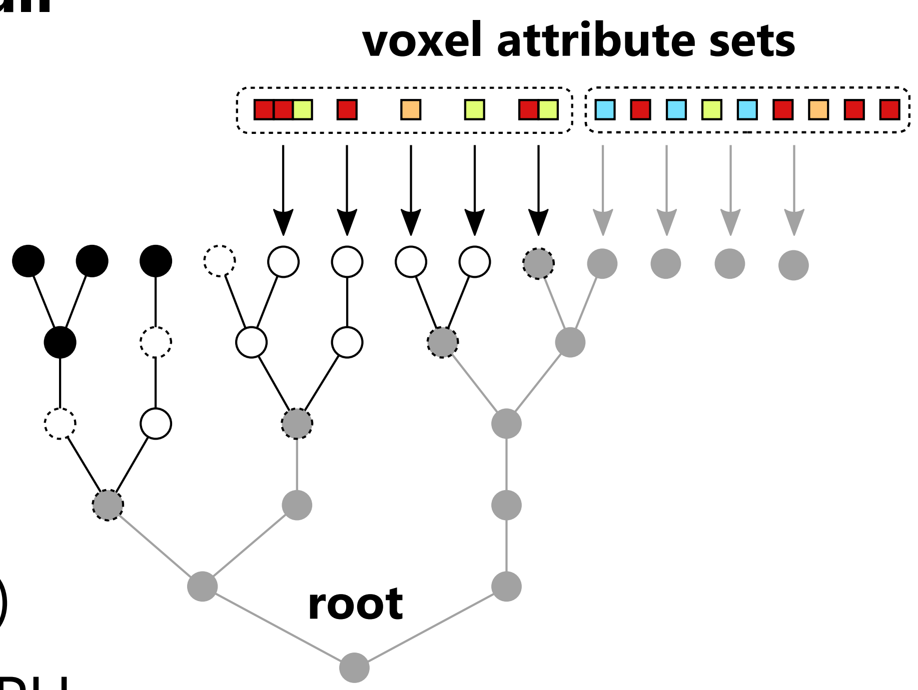
Morton Queueing

- After voxelization of a triangle batch, **processable voxels** for creation of the SVO need to be **determined**
- Morton queue stores unusable voxels from previous iterations & all voxels of curr. iteration
- After sorting of Morton queue, the **1st Morton index** of batch j+1 is used as **stop** to extract a set of voxel-attribute pairs for **SVO creation**



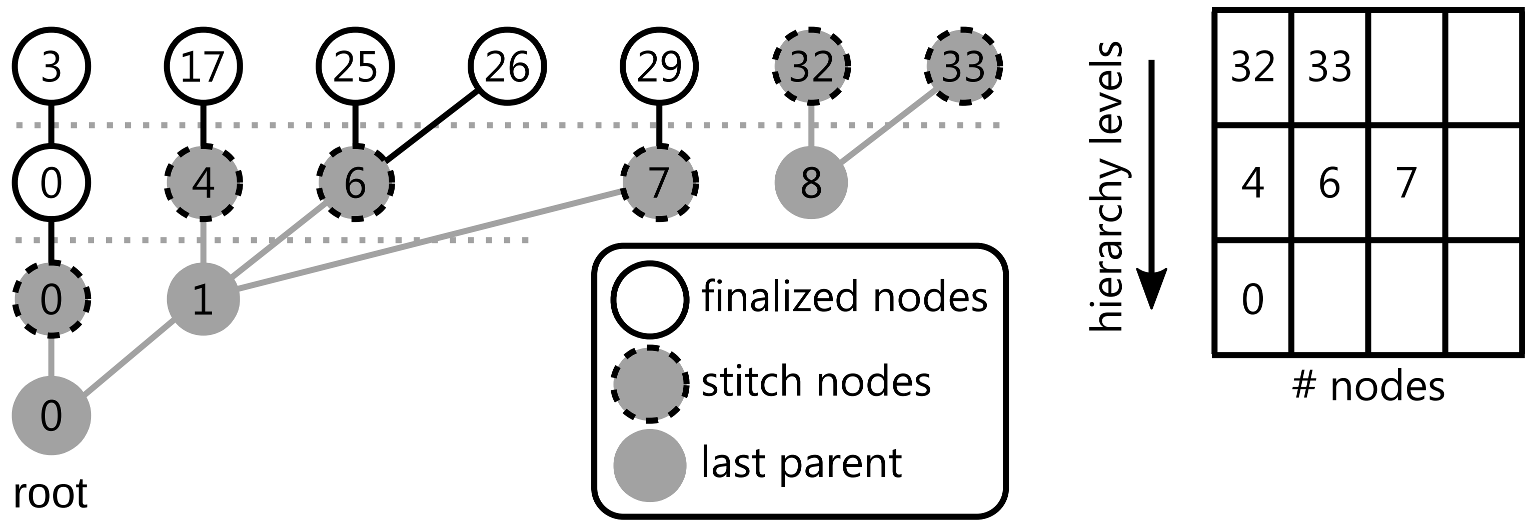
SVO Creation

- Valid set of voxel-attribute pairs is used to create parts of SVO **bottom-up** with parallel stream compaction (parent = child/8)
- Each GPU **thread** processes **all child nodes** of one parent
- Data structure, similar to [Laine and Karras 2010]:
  - **Bitmasks** to address the non-empty voxels (mod)
  - Voxel **attributes** (e.g. color)
  - indices (child-pointer) on CPU



Post-Order Attribute Creation

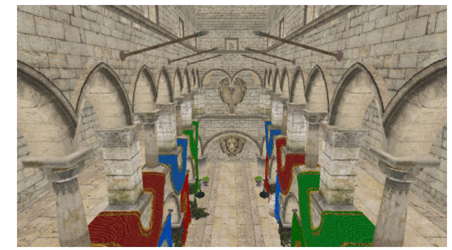
- Parent attributes are created only if **all child nodes available**
- Use of a **stitch queue** on each hierarchy level buffers voxel-attribute pairs until all nodes are given
- Attributes can be determined by **multipass-operations**, etc.



Results (performance)








Scene	**Hairball** (2.8 M triangles)			**Lucy** (28.0 M triangles)			**Atlas** (506.5 M triangles)		
Resolution	2048	4096	8192	2048	4096	8192	2048	4096	8192
[Laine/Karras 2011]	274.4 s	763.7 s	2657.8 s	964.3 s	1001.9 s	1097.4 s	-	-	-
[Baert et al. 2014]	134.4 s	759.2 s	4459.9 s	17.5 s	40.7 s	97.9 s	223.3 s	351.4 s	676.3 s
Our algorithm	83.0 s	281.9 s	1195.5 s	11.7 s	16.9 s	30.3 s	270.0 s*	239.8 s*	345.7 s*

	Scene	**Buddha** (30.3 K triangles)			**Sponza** (262.3 K triangles)			**San Miguel** (10.1 M triangles)		
	Resolution	2048	4096	8192	2048	4096	8192	2048	4096	8192
w/o col.	[Laine/Karras 2011]	14.3 s	59.9 s	243.2 s	24.8 s	92.1 s	364.2 s	(140.6 s)	(153.39 s)	(165.9 s)
	[Baert et al. 2014]	15.4 s	66.6 s	372.1 s	23.1 s	83.6 s	437.1 s	9.5 s	26.5 s	107.4 s
	Our algorithm	4.4 s	14.0 s	49.4 s	6.8 s	24.1 s	97.9 s	6.2 s	12.0 s	32.6 s
w col.	[Laine/Karras 2011]	16.5 s	65.3 s	262.6 s	31.3 s	112.4 s	428.7 s	(171.7 s)	(187.5 s)	(203.9 s)
	[Baert et al. 2014]	55.4 s	166.3 s	611.6 s	52.1 s	363.7 s	1416.9 s	13.0 s	37.6 s	228.3 s
	Our algorithm	5.1 s	17.9 s	50.4 s	11.0 s	30.8 s	111.6 s	13.2 s	20.4 s	44.3 s

* : average over three runs, (...) : scene could be voxelized, but not rendered

Results (performance)

Scene	Hairball (2.8 M triangles)			Lucy (28.0 M triangles)			Atlas (506.5 M triangles)		
Resolution	2048	4096	8192	2048	4096	8192	2048	4096	8192
[Laine/Karras 2011]	274.4 s	763.7 s	2657.8 s	964.3 s	1001.9 s	1097.4 s	-	-	-
[Baert et al. 2014]	134.4 s	759.2 s	4459.9 s	17.5 s	40.7 s	97.9 s	223.3 s	351.4 s	676.3 s
Our algorithm	83.0 s	281.9 s	1195.5 s	11.7 s	16.9 s	30.3 s	270.0 s*	239.8 s*	345.7 s*

	Scene	Buddha (30.3 K triangles)			Sponza (262.3 K triangles)			San Miguel (10.1 M triangles)		
	Resolution								4096	8192
w/o col.	[Laine/Karras 2011]								(153.39 s)	(165.9 s)
	[Baert et al. 2014]								26.5 s	107.4 s
	Our algorithm								12.0 s	32.6 s
w col.	[Laine/Karras 2011]								(187.5 s)	(203.9 s)
	[Baert et al. 20								s	228.3 s
	Our algorithm								s	44.3 s

t not rendered

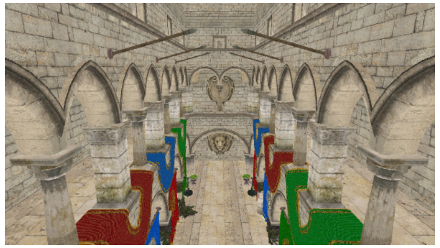
Sparsity of scene has more influence on performance than triangle count

Ours: regular triangles of Lucy are more suitable than long thin triangles of Hairball

Results (performance)








Scene	**Hairball** (2.8 M triangles)			**Lucy** (28.0 M triangles)			**Atlas** (506.5 M triangles)		
Resolution	2048	4096	8192	2048	4096	8192	2048	4096	8192
[Laine/Karras 2011]	274.4 s	763.7 s	2657.8 s	964.3 s	1001.9 s	1097.4 s	-	-	-
[Baert et al. 2014]	134.4 s	759.2 s	4459.9 s	17.5 s	40.7 s	97.9 s	223.3 s	351.4 s	676.3 s
Our algorithm	83.0 s	281.9 s	1195.5 s	11.7 s	16.9 s	30.3 s	270.0 s*	239.8 s*	345.7 s*

Scene		**Buddha** (30.3 K triangles)	**Sponza** (262.3 K triangles)	**San Miguel** (10.1 M triangles)
Resolution				
w/o col.	[Laine/Karras 2011]			
	[Baert et al. 2014]			
	Our algorithm			
w col.	[Laine/Karras 2011]			
	[Baert et al. 2014]			
	Our algorithm			

Sub-partioning of Baert et al. [2014] is more performant than our triangle sorting if triangle count is high and grid resolution is low
(but better scalability for higher resolutions)

Results (performance)

Scene	Atlas (506.5 M triangles)		
Resolution	2048	4096	8192
[Laine/Karras 2011]	-	-	-
[Baert et al. 2014]	223.3 s	351.4 s	676.3 s
Our algorithm	270.0 s*	239.8 s*	345.7 s*

Better scalability if workload on GPU is higher (color attributes)

	Scene	Buddha (30.3 K triangles)			Sponza (262.3 K triangles)			San Miguel (10.1 M triangles)		
	Resolution	2048	4096	8192	2048	4096	8192	2048	4096	8192
w/o col.	[Laine/Karras 2011]	14.3 s	59.9 s	243.2 s	24.8 s	92.1 s	364.2 s	(140.6 s)	(153.39 s)	(165.9 s)
w/o col.	[Baert et al. 2014]	15.4 s	66.6 s	372.1 s	23.1 s	83.6 s	437.1 s	9.5 s	26.5 s	107.4 s
w/o col.	Our algorithm	4.4 s	14.0 s	49.4 s	6.8 s	24.1 s	97.9 s	6.2 s	12.0 s	32.6 s
w col.	[Laine/Karras 2011]	16.5 s	65.3 s	262.6 s	31.3 s	112.4 s	428.7 s	(171.7 s)	(187.5 s)	(203.9 s)
w col.	[Baert et al. 2014]	55.4 s	166.3 s	611.6 s	52.1 s	363.7 s	1416.9 s	13.0 s	37.6 s	228.3 s
w col.	Our algorithm	5.1 s	17.9 s	50.4 s	11.0 s	30.8 s	111.6 s	13.2 s	20.4 s	44.3 s

* : average over three runs, (...) : scene could be voxelized, but not rendered

Results (performance)

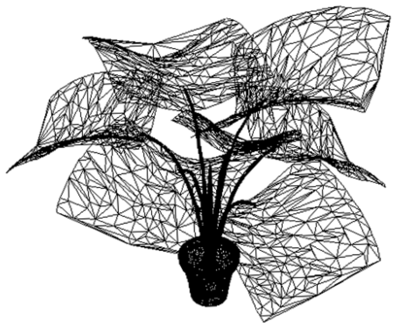
Scene	Hairball (2.8 M t	
Resolution	2048	4096
[Laine/Karras 2011]	274.4 s	763.7 s
[Baert et al. 2014]	134.4 s	759.2 s
Our algorithm	83.0 s	281.9 s

High initialization cost (e.g. texture loading) is bad for small resolution but neglectable for higher resolution

	Scene	**Buddha** (30.3 K triangles)			**Sponza** (262.3 K triangles)			**San Miguel** (10.1 M triangles)		
	Resolution	2048	4096	8192	2048	4096	8192	2048	4096	8192
w/o col.	[Laine/Karras 2011]	14.3 s	59.9 s	243.2 s	24.8 s	92.1 s	364.2 s	(140.6 s)	(153.39 s)	(165.9 s)
	[Baert et al. 2014]	15.4 s	66.6 s	372.1 s	23.1 s	83.6 s	437.1 s	9.5 s	26.5 s	107.4 s
	Our algorithm	4.4 s	14.0 s	49.4 s	6.8 s	24.1 s	97.9 s	6.2 s	12.0 s	32.6 s
w col.	[Laine/Karras 2011]	16.5 s	65.3 s	262.6 s	31.3 s	112.4 s	428.7 s	(171.7 s)	(187.5 s)	(203.9 s)
	[Baert et al. 2014]	55.4 s	166.3 s	611.6 s	52.1 s	363.7 s	1416.9 s	13.0 s	37.6 s	228.3 s
	Our algorithm	5.1 s	17.9 s	50.4 s	11.0 s	30.8 s	111.6 s	13.2 s	20.4 s	44.3 s

* : average over three runs, (...) : scene could be voxelized, but not rendered

Results (attributes)


Mesh


Textured


[Laine/Karras 2011] contour


[Baert et al. 2014]


Ours


[Laine/Karras 2011] voxel

Conclusion

- **Out-of-core** voxelization on **GPU** with workload balancing
- Processing of non-empty voxels only → **grid-free**
- Possibility to create **attributes in post-order**

- Future Work:
  - **Adaptive batch determination** → size of Morton queue (performance vs. out-of-memory)
  - Create more **sophisticated voxel attributes** (statistics of underlying attributes, sorting)

Thank you for your attention!

e-mail: {martin.paetzold, andreas.kolb} @uni-siegen.de

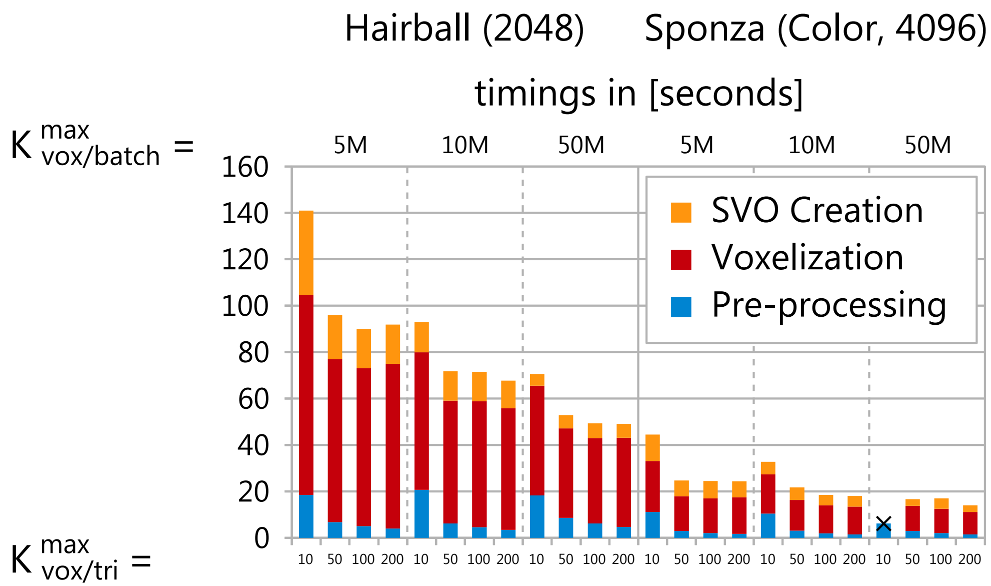
Acknowledgments

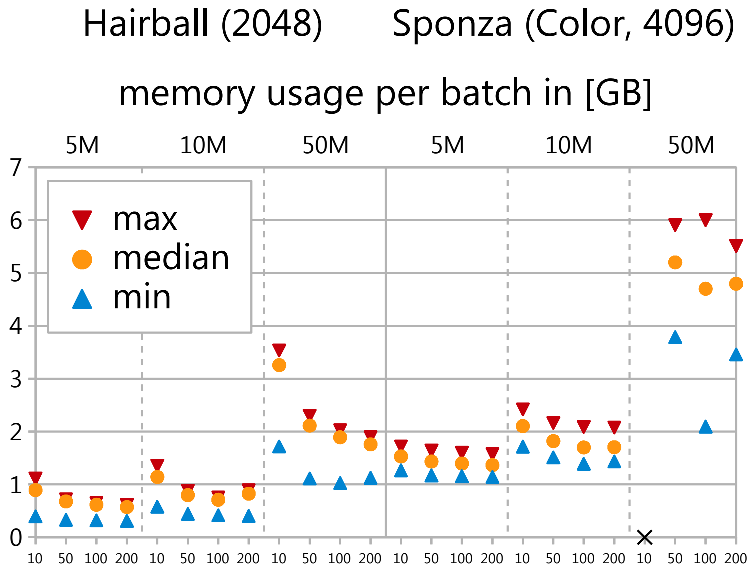
The research project is funded by the German Research Foundation (DFG)
as part of the research training group **GRK 1564 "Imaging New Modalities"**

Hairball: Nvidia Research. **Lucy**: Stanford 3D Scanning Repository.
Atlas: The Digital Michelangelo Project. **San Miguel**: Guillermo M. Leal Llaguno.
Crytek Sponza: Frank Meinl. **Buddha**: Kun Zhou. **Teapot**: Martin Newell.

Results (influence of user-defined values)

- **Performance** increases with more voxels per batch and remains constant for voxels per triangles but drops for smallest value
- **Memory** usage increases with more voxels per batch and slightly decreases with more voxels per triangle





Results (influence of user-defined values)

- **Triangle count** increases with more voxels per batch and decreases with more voxels per triangle
- **Number** of generated **voxel-attribute pairs** increases with more voxels per batch & remains constant with more voxels per triangle