Voxel DAGs and Multiresolution Hierarchies

From Large-Scale Scenes to Pre-computed Shadows

Ulf Assarsson[1], Markus Billeter[2], Dan Dolonius[1], Elmar Eisemann[2], Alberto Jaspe[3], Leonardo Scandolo[2], Erik Sintorn[1]

1) Chalmers University of Technology, 2) Delft University of Technology, 3) CRS4 Visual Computing

EUROGRAPHICS 2018

Voxel DAGs and Multiresolution Hierarchies

- Volumes encoded with very little memory by exploiting redundancy

2^51 voxels =
2,251,799,813,685,248 Voxels

Potential to become a standard representation for real-time rendering

Overview

• **Introduction**	Ulf Assarsson & Elmar Eisemann	30 min
• **Advanced DAG Encodings**	Alberto Jaspe	20 min
• **Attribute Compression**	Dan Dolonius & Ulf Assarsson	30 min
Break		
• **DAG Construction**	Erik Sintorn	20 min
• **DAG Ray-tracing**	Markus Billeter	20 min
• **Applications/Demos**	(all)	10 min
• **Compressed Shadow Volumes**	Erik Sintorn	20 min
• **Compressed Shadow Maps**	Leonardo Scandolo	20 min
• **Conclusion and Q&A**	(all)	10 min
	Total time:	180 min

Representation

Overview

• **Introduction**	Ulf Assarsson & Elmar Eisemann	30 min
• **Advanced DAG Encodings**	Alberto Jaspe	20 min
• **Attribute Compression**	Dan Dolonius & Ulf Assarsson	30 min
Break		
• **DAG Construction**	Erik Sintorn	20 min
• **DAG Ray-tracing**	Markus Billeter	20 min
• **Applications/Demos**	(all)	10 min
• **Compressed Shadow Volumes**	Erik Sintorn	20 min
• **Compressed Shadow Maps**	Leonardo Scandolo	20 min
• **Conclusion and Q&A**	(all)	10 min
	Total time:	180 min

Overview

Representation	• **Introduction**	Ulf Assarsson & Elmar Eisemann	30 min
	• **Advanced DAG Encodings**	Alberto Jaspe	20 min
	• **Attribute Compression**	Dan Dolonius & Ulf Assarsson	30 min
	Break		
Usage	• **DAG Construction**	Erik Sintorn	20 min
	• **DAG Ray-tracing**	Markus Billeter	20 min
	• **Applications/Demos**	(all)	10 min
	• **Compressed Shadow Volumes**	Erik Sintorn	20 min
	• **Compressed Shadow Maps**	Leonardo Scandolo	20 min
	• **Conclusion and Q&A**	(all)	10 min
		Total time:	180 min

Overview

Representation	• **Introduction**	Ulf Assarsson & Elmar Eisemann	30 min
	• **Advanced DAG Encodings**	Alberto Jaspe	20 min
	• **Attribute Compression**	Dan Dolonius & Ulf Assarsson	30 min
	Break		
Usage	• **DAG Construction**	Erik Sintorn	20 min
	• **DAG Ray-tracing**	Markus Billeter	20 min
Applications	• **Applications/Demos**	(all)	10 min
	• **Compressed Shadow Volumes**	Erik Sintorn	20 min
	• **Compressed Shadow Maps**	Leonardo Scandolo	20 min
	• **Conclusion and Q&A**	(all)	10 min
		Total time:	180 min

Speakers

- Ulf Assarsson, Dan Dolonius, Erik Sintorn
  Chalmers University of Technology

- Markus Billeter, Elmar Eisemann, Leonardo Scandolo
  Delft University of Technology

- Alberto Jaspe
  CRS4 Visual Computing

Voxels?

Voxels?

- Why would you even consider something else than triangles?

Voxel Engines in Special effects

- Natural representation
  - Fluid, smoke, scans…
- Volumetric phenomena
  - Semi-transparency…
- Unified rendering representation
  - Particles, meshes, fluids…

Voxels however are still extremely popular in FX companies.
They are a natural fit to simulations of complex phenomena such as fluids, smoke, and scan data.
It integrates very well with particles, meshes, and other representations.

==========

But we are not the first, to talk about voxel engines. They have been used for special effects for years because they achieve high quality effects.
Voxels are the natural representation for fluid simulations and 3D scans data and they are massively used to render volumetric phenomenas thanks to they intrinsic structured ordering.
Voxels are also used to unify many objects representations into a unique structured representation used for lighting computation and rendering.

These off-line engines are producing very realistic looking images, but at the cost of TeraBytes of memory and hours of rendering computations.

****

We are not the first to see that voxels are interesting.
For example for transparency, you would have to sort
####
In special effect, there is more and more impressive image of large and detailed scenes, made using voxel engine.
DD, RH for instance are wildly using VO not necessarily for semi-transparent pheno but also for high quality effects in general, and to unify rendering and illumination computation for composed scenes.

But these engines are very expensive in computation time and manipulated data that represents hundred of gigabytes that don't even fit into system memory.

Object IS the texture !
//, off line computation. Don't fit into cpu memory.
####
+add boat

Unify For rendering and illumination computations

[Legende, dire memoire monstrueuse]

[vous conaissez les voxels pour ca]

Voxels in video games ?

- Renewed interest
  - Jon Olick, Siggraph 08
  - Crytek
  - NVIDIA [Laine & Karras 2010m]
  - …

In video game, we actually observe renewed interest for voxels – in particular last year Jon Olik presented a prototype of a voxel renderer; and of course JCarmack has been mentioning this for years.

==========

[trop de texte]

//Not only usefull for volumetric phenomenas

Even video game companies start to be interested in voxels rendering. Jon Olick from ID software presented an experimental Sparse Voxel Octree renderer at Siggraph last year and voxel representation has been a recurrent idea for John Carmack for years.

Indeed, video games content generation pipeline (using tools like Zbrush) produces more and more detailed geometry that are hard to texture and to prepare for realtime rendering.

#Another point to underline here, is that voxels provide an unified object representation, meaning that we don't have our model that we need to fill around with our texture coordinates, our model IS TEXTURE.

Pourquoi olick le fait

Why bother with voxels?

- Exploding number of triangles
  - Sub-pixel triangles not GPU-friendly
- Filtering remains an issue
  - Multi-sampling expensive
  - Geometric LOD ill-defined
- Clouds, smoke, fluids, etc.
  - Participating media?

So why would we come back to voxels anyway?

Triangles reach their limits, we have to many of them #and very small triangles are currently not GPU friendly at all.
In addition, filtering is really a big issue on such complex geometries.
The last argument for using voxels is that games are now ready to make massive use of participating medias for smoke or fluid simulations for instance.

==========

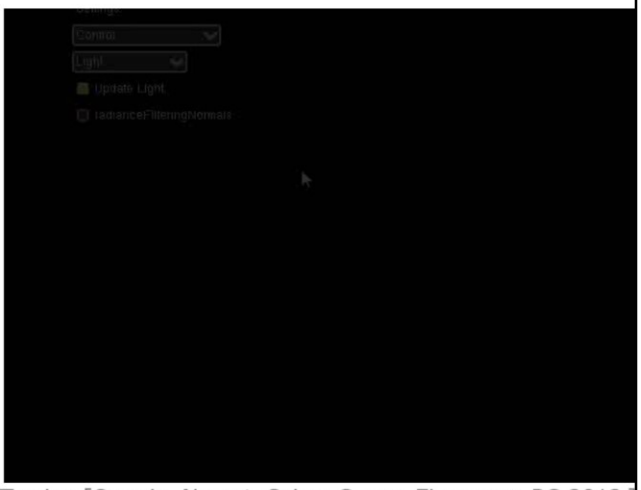
[trop chargé]

But such models are made of Huge number of triangles that can't always be replaced with textures. For instance, this model is made of more than 10Millions of triangles.

Rendering so huge number of micro-triangles is very costly and we can't imagine to use them in video games in the years to come, even if next generation GPU will be more optimized to rasterize small triangles.

Many Applications!

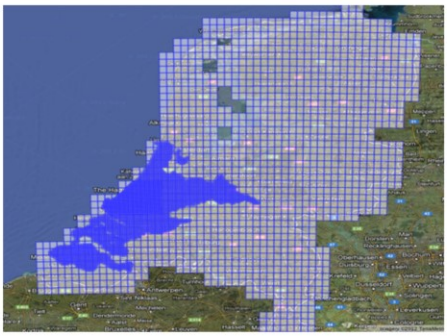
- Collision Detection
- Shadow Calculation
- Global Illumination
- Ambient Occlusion
- …

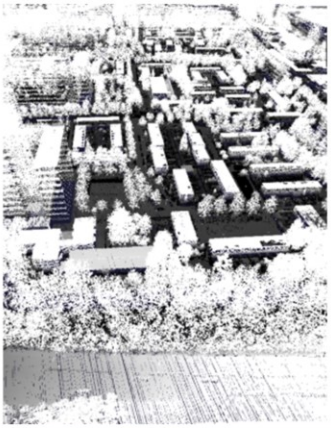


Voxel-Cone Tracing [Crassin, Neyret, Sainz, Green, Eisemann, PG 2012]

Scanned Data Sets

- AHN2 dataset with satellite imagery ~8 TB of data

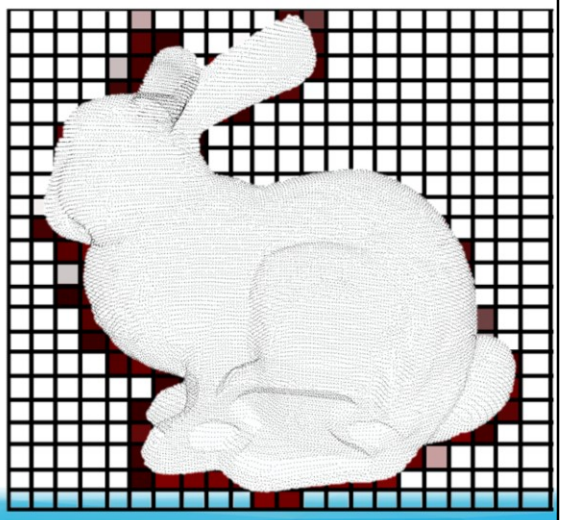




That is a 15m tower of CDs

Add structure…

- Voxel Representations



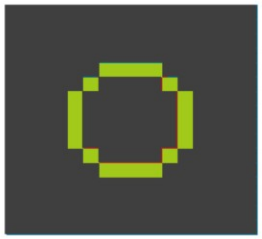
Scanned Data Sets

Voxel Representations

- **Main problem:**
  - Memory is a key issue!
    - E.g. 2048^3 x RGBA = **32 GB**

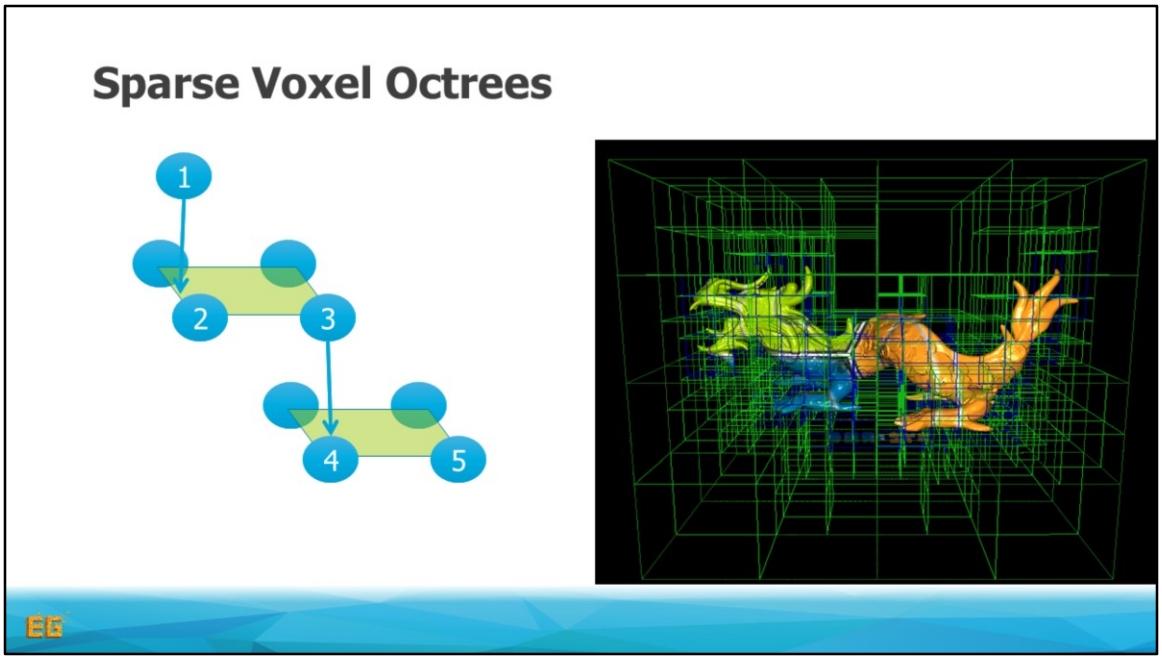
Sparse Encoding

- Group constant valued areas (including empty)

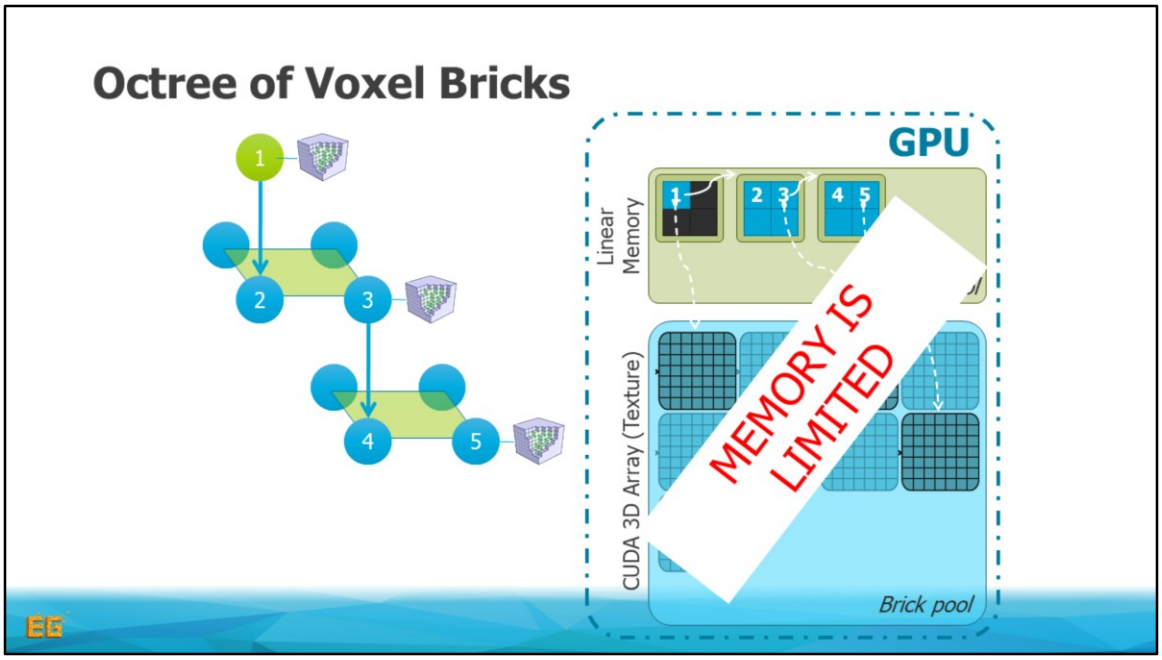


#Well the first thing that we can do, to reduce memory, Is one taking a look at the object place in the voxel grid, is to remove, or to group together values that are constant.
#And in case where a solid object intersects the grid, means that you have the interior part of one density, you have the exterior part that is completely uniform, only on the boundary, things are actually getting a little bit more complicated.
#So you have these large chunks of data, that can be grouped together to reduce the cost in the memory.

Lets take a look on an object into a voxel grid.
Interesting things going on boundaries



Now spatially, this correspond to different volumes. So the root node brick is going to cover the entire scene, #it's children will split this area into 4 regions of same size, where each has the same resolution as the original root node. #Then as you can see, this allow us to further refine those areas where more precision is needed. #This is how it looks in 3D when you apply this to some dragon model.



But how is this structure implemented on the GPU ?
The octree is implemented as a traditional pointer based structure composed of nodes.
#Nodes of the octree are stored in a dedicated video memory region we called the "Node Pool". #On the other side, bricks associated the nodes are stored in another dedicated memory region called the "Brick pool", that is very large 3D texture.

#So here is the root node stored in the node pool,
#and it's associated brick stored in the brick pool.
#it has 8 children, here represented by 4, #stored continuously in the node pool and liked by only one pointer. This nodes representation provides us with a very compact structure, that is inexpensive to store and cache efficient.
#nodes have associated bricks, ##and other subnodes #with bricks

Pool sizes are fixed at initialisation time. In a video-game context, their size would be chosen depending on available video memory. As we will see later, these pools are managed as caches.

==========

Thanks to an efficient nodes representation, the node pool is a very small memory region, while the brick pool is larger since it did store the voxel data. Both sizes are fixed at initialisation time, and in a video-game context, they would chosen depending on available video memory. As we will see later, these pools are managed as caches and so their size only impact bandwidth pressure and so the performance of the streaming system.

This octree is built from top to bottom, on demand, and fully on GPU as I will show later.

Having such dedicated memory regions is necessary to be able to manage both of them as a caches from the GPU. Their size is [variable/chosen depending on objects resolution, screen resolution and available video memory.]

[Octree nodes are very light, Tiles -> very cheap structure]

As I said previously, we want to deal with extremely large voxel objets, or scenes.
So in order to implement a custom data management,
In this context, the structure needs to be dynamically

In order to be able to use hardware tri-linear interpolation inside bricks, as well as the 3D-local texture cache, the brick pool is implemented as a CUDA array, corresponding to an OpenGL 3D texture. The node pool is implemented in standard global linear memory in order to be modifiable directly from on kernel on GPU. But in order to get cached accessed during rendering, it is bound as a linear texture.

As illustrated on this diagram, nodes are grouped 2x2x2 into what we call a "Node Tile". Using node tiles allows us to only store one pointer to the 8 sub-nodes of a given node. In fact, we designed this structure as light as possible by encoding them only on each 64 bits.

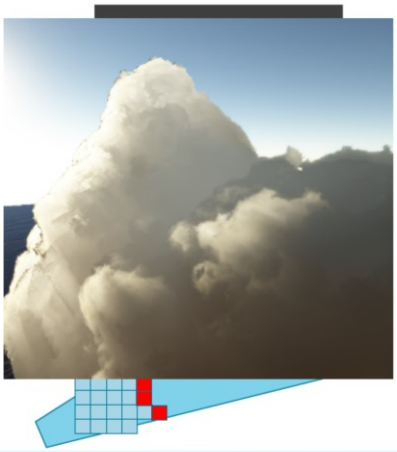
[parler a la fin de la voxelization probleme, texture lineaire]

****
#It's start from the root node, #just here indicated in the node pool, and it has a corresponding texture in the brick pool. #Then if this node is subdivided, #it's gona have 4 children that you can find in a coherent location in the node pool. This coherent location is very important because it allows us to use one single children pointer, to get all the children with just one address. This makes our structure really small, and is very good to improve cache efficiency. #On the other side, texture information is loaded into the brick pool.

#Then you can see the second level of this representation, #again nodes are coherent in the node pool, and the textures is in the other side in the brick pool.

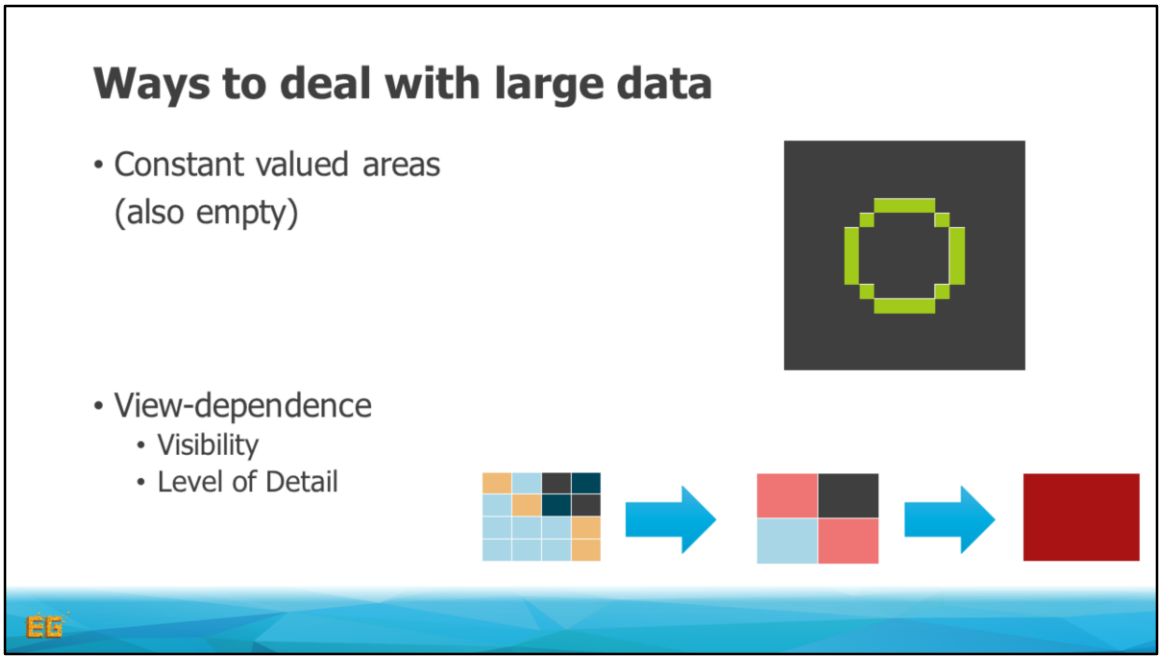
Ways to deal with large data

- Constant valued areas (also empty)
- View-dependence
  - Visibility



Then if you take a look at this volume, #you realize that it's much smaller than the initial thing.

And furthermore, that's not only true for opaque objects, but for all these transparent things that I shown before.
#If you take a look at this cloud, you realize that even if it's transparent everywhere, at some point the rays are just blocked due to opacity.



#We can group together neighboring voxels to produce level of detail representations that then consume less memory and are even faster to render.

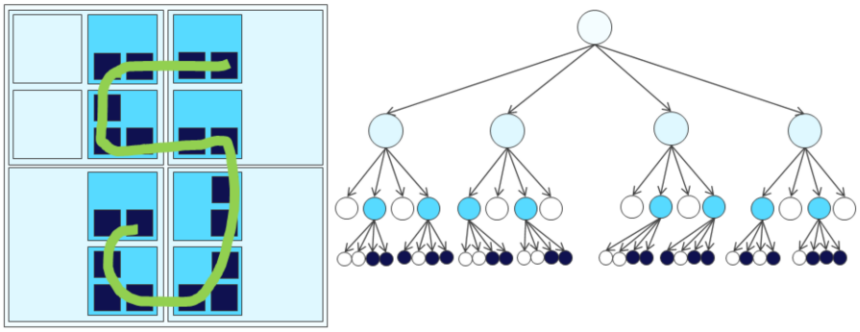
Voxel DAGs=Exploit redundancy reduce memory

- Keep all data in core
- Maintain:
  - Efficient Random Access
  - Hierarchical Data Representation

[Kaempe et al. – SIGGRAPH 2013]

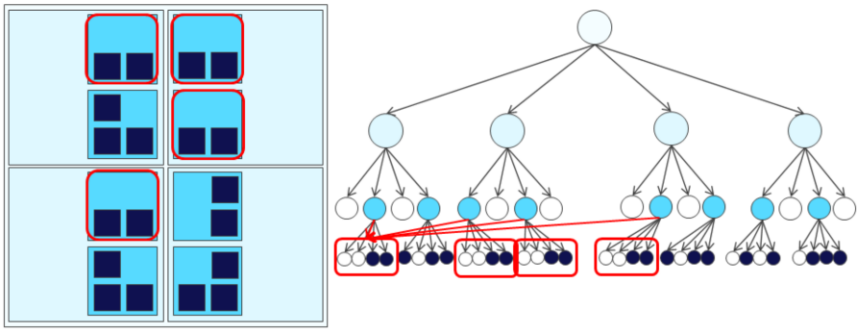
Sparse Voxel Directional Acyclic Graphs (DAG)

Binary Data Example



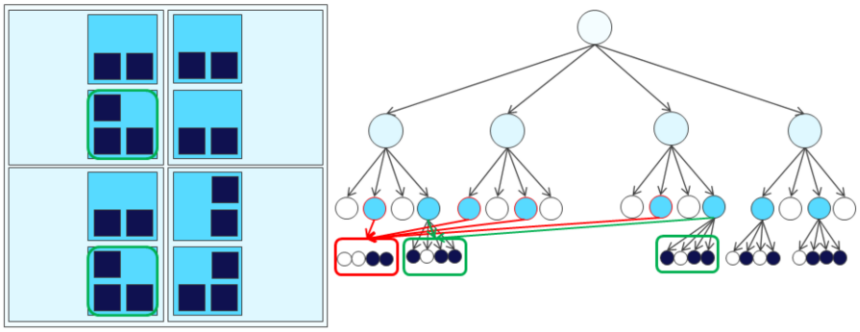
[Kaempe et al. – SIGGRAPH 2013]

Sparse Voxel Directional Acyclic Graphs (DAG)



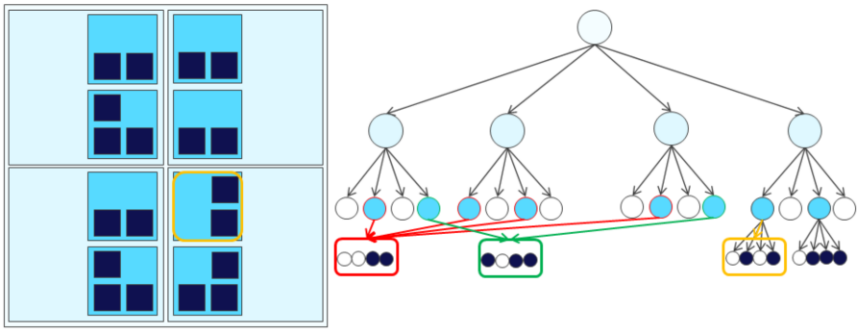
[Kaempe et al. – SIGGRAPH 2013]

Sparse Voxel Directional Acyclic Graphs (DAG)



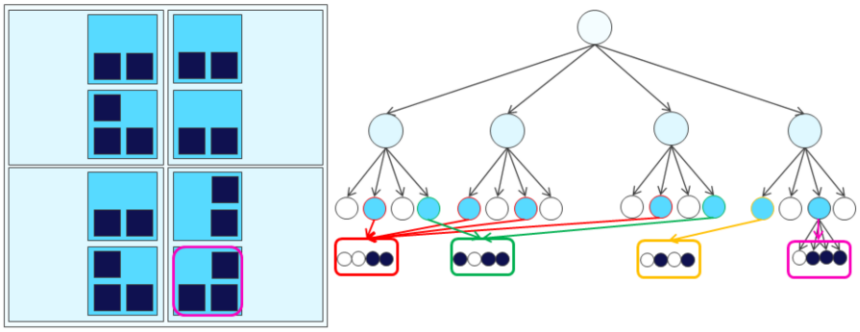
[Kaempe et al. – SIGGRAPH 2013]

Sparse Voxel Directional Acyclic Graphs (DAG)



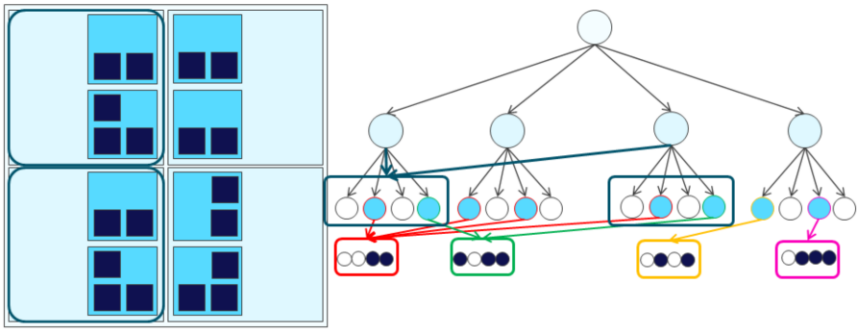
[Kaempe et al. – SIGGRAPH 2013]

Sparse Voxel Directional Acyclic Graphs (DAG)



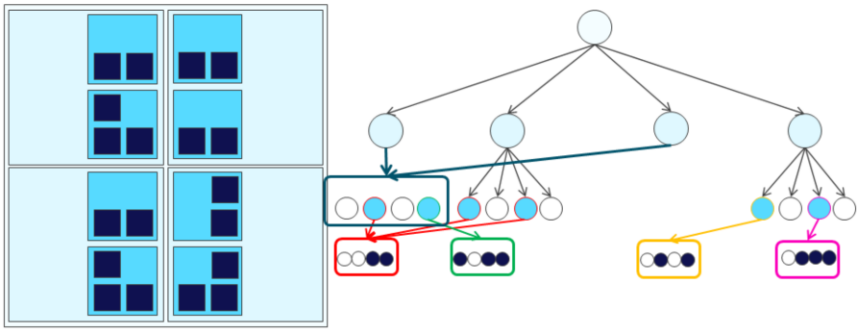
[Kaempe et al. – SIGGRAPH 2013]

Sparse Voxel Directional Acyclic Graphs (DAG)



[Kaempe et al. – SIGGRAPH 2013]

Sparse Voxel Directional Acyclic Graphs (DAG)



Example

64K³, 12-bit colors
15 billion filled voxels
Size: <3.3 GB
SVO/DAG:11.5x

San Miguel has very complex geometry at several locations but also many axis-aligned walls of similar colors, for which compression works very well