



HAL
open science

GigaVoxels: Voxels Come Into Play

Cyril Crassin

► **To cite this version:**

Cyril Crassin. GigaVoxels: Voxels Come Into Play. Crytek Conference, Nov 2009, Frankfurt, Germany. inria-00526646

HAL Id: inria-00526646

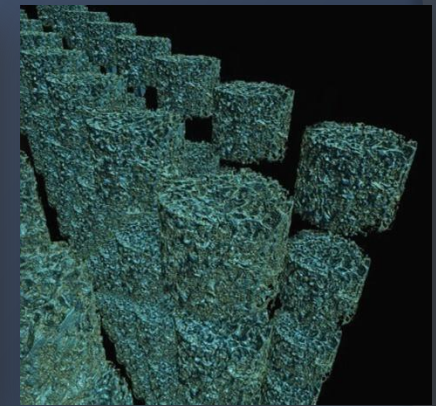
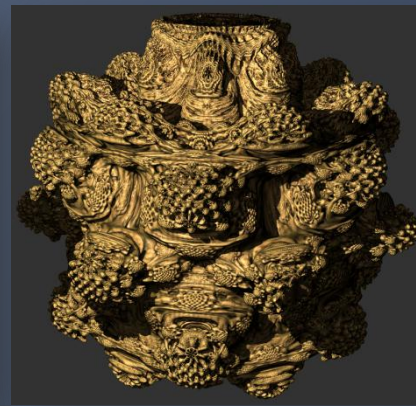
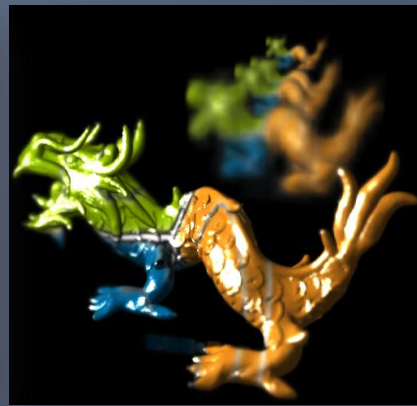
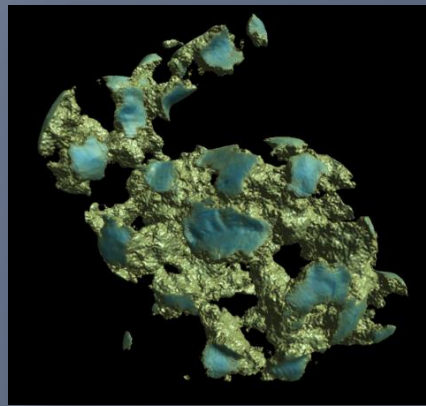
<https://inria.hal.science/inria-00526646v1>

Submitted on 15 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

GIGAVOXELS: VOXELS COME INTO PLAY



Cyril Crassin, Fabrice Neyret,

INRIA Rhône-Alpes & Grenoble Univ.

*Sylvain Lefebvre,
INRIA Sophia-Antipolis*

*Elmar Eisemann,
Saarland Univ./MPI*

*Miguel Sainz
NVIDIA Corporation*



INRIA



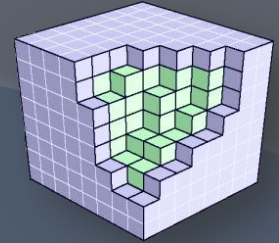
GRENOBLE
UNIVERSITÉS



UNIVERSITÄT
DES
SAARLANDES

A (very) brief history of voxels

- Rings a bell?



Voxel grid illustration
courtesy of "Real-Time
Volume Graphics"



Comanche (Novalogic)

Voxel Engines in Special effects

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



Voxels in video games ?

⦿ Renewed interest

- ID Software

- John Carmack, Jon Olick (Siggraph 08)
- Sparse Voxel Octree ray-casting

- Crytek

- Cevat Yerli

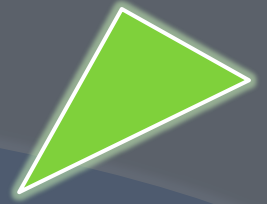
- ...

⦿ Two goals :

- Content generation
- Rendering



Why bother with voxels?



- ⦿ Exploding number of triangles
 - Costly to transform & rasterize
 - Inefficient raster of small triangles on current generation GPUs
- ⦿ Geometric LOD ill-defined
 - Eg. Progressive Meshes
 - Lot of manual intervention for the artist



Why bother with voxels?

- Filtering is an issue
 - Needs massive multi-sampling
 - Multi-sampling is expensive

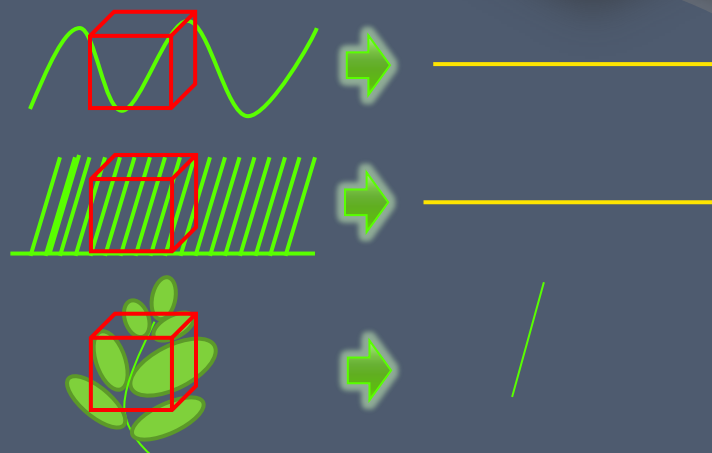
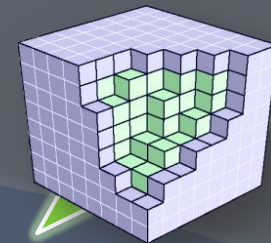


The Mummy 3, Digital Domain/Rhythm&Hues



Why bother with voxels?

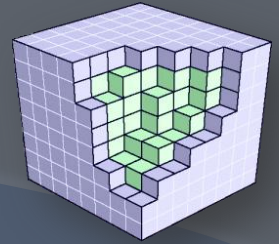
- Unified Geometry + Texture representation
- Avg space **occupancy/density** information
- Avg **color** information



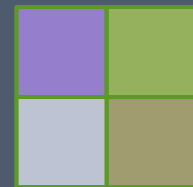
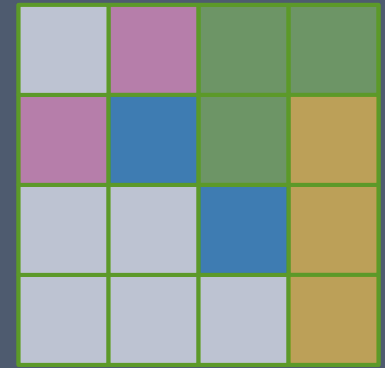
The Mummy 3, Digital Domain/Rhythm&Hues



Why bother with voxels ?



- Filtering is well defined
 - LOD = Mip-Mapping
 - Similarly to 2D textures
- Unique multi-scale representation
 - No additional authoring
- Structured representation
 - Convenient to traverse & edit
 - Efficient to render
 - -> Ray-casting



How to exploit them ?

◎ Main problems:

- How to render voxels quickly on the GPU ?
 - How to exploit these properties ?
- Memory is a key issue !
 - E.g. $4096^3 \times \text{RGBA8} = \mathbf{256 \text{ GB!!!}}$
 - Transfer CPU \Leftrightarrow GPU expensive

GigaVoxels

- Goal: Real-time exploration of very large voxel scenes
- Full GPU rendering pipeline
 - Ray-tracing based approach
 - Fully scalable: Infinite resolution
- Publications:
 - I3D2009 paper [CNLE09]
 - Siggraph 2009 Talk
 - GPU Pro (ShaderX 8) Book Chapter

GigaVoxels: Ray-Guided Streaming for Efficient and Detailed Voxel Rendering

Cyril Crassin Fabrice Neyret Sylvain Lefebvre Elmar Eisenmann
LJK / INRIA / Grenoble Universities / CNRS INRIA Sophia-Antipolis MPI Informatik / Saarland University

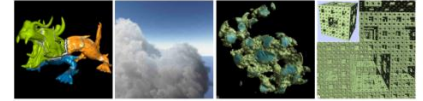
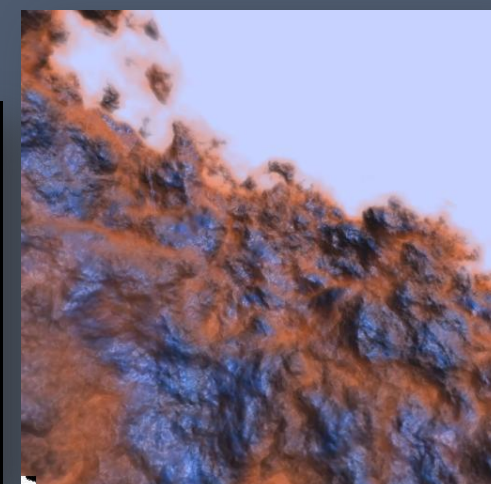
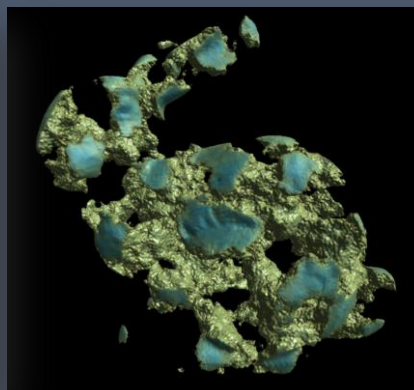
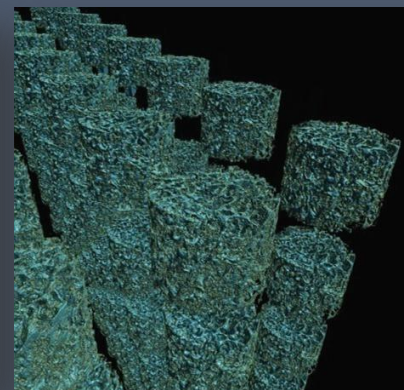
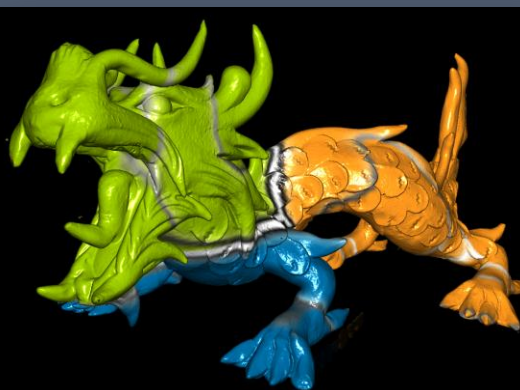


Figure 1: Images show volume data that consist of billions of voxels rendered with our dynamic, sparse voxel approach. Our algorithm achieves real-time interactive rates on modern commodity GPUs, even at rates up to 10x faster than ray-tracing rendering and ray-casting rendering. Ideally, the voxels are only used for the resolution that is needed to produce the final image. Besides the gains in memory and speed, our rendering pipeline is inherently non-linear.

Abstract
We propose a new approach to efficiently render large volumetric data sets. The system achieves interactive to real-time rendering performance for several billion voxels.
Our solution is based on an adaptive data representation depending on the current view and occlusion information, coupled to an efficient ray-tracing algorithm. Our key element of our method is to guide data production and streaming directly based on information extracted during rendering.
Our data structure exploits the fact that in CG scenes, details are often concentrated on the interface between free space and clusters of density and shows that volumetric models might become a viable alternative as a rendering primitive for real-time applications. In this spirit, we allow a quality-performance trade-off and explore temporal coherence. We also introduce a mipmapping-like process that allows for an increased display size and better quality through high quality filtering. To further enrich the data set, we create additional details through a variety of procedural methods.
We demonstrate our approach to several scenarios, like the exploration of a 3D scan (512³ resolution), of hyperscanned meshes (16384³ voxel resolution), or of a fractal (theoretically infinite resolution). All examples are rendered on current generation hardware at 20-30 fps and respect the limited GPU memory budget.
This is the authors' version of the paper. The ultimate version has been published in the I3D 2009 conference proceedings.

1 Introduction
Volume data has often been used in the context of scientific data visualization, but it is also part of many special effects. Companies such as Digital Domain, Creative or Bluebird Effects now mainly rely on so-called voxel engines (Kajiya, BTVOG, Kapri, KH05) to render very complex scenes.
Examples can be found in many recent movie productions (e.g., *XXX*, *Lord of the Rings*, *The Day After Tomorrow*, *Pirates of the Caribbean: The Men of Black*), video games, and even non-fluffy but extremely detailed geometric data (e.g., boats in *Pirates of the Caribbean*) are all represented with voxels and rendered via volume rendering. The scene size and resolution is so large that voxels often do not even fit in the computer's memory. In addition to storage, the rendering of such data is also extremely costly, even for post-visualization.
The significant advantage of voxels is the richness of this representation and the very regular structure which makes it easy to sample. In particular, filtering operations are well-defined, making it a good candidate to address aliasing issues that are hard to deal with for triangulated models.
This is one of the reasons voxel data is often used to represent pseudo-surfaces, which is an interface that resembles a surface at a certain distance, but appears complex (non-hierarchical, non-connected, or non-regular) at close view. An example is the foliage of a tree that can be well approximated with volumetric data [BMM04], but this observation holds for complex surfaces in general. This volumetric rendering is also a graceful way of dealing with the level of detail problem.
In this paper, we show that the current hardware generation is able to achieve high-quality memory volume rendering at interactive to real-time rates. Benefits such as filtering, occlusion culling, and procedural data creation, as well as level-of-detail mechanisms are integrated in an efficient GPU-voxel engine. This enables us to obtain some of the visual quality that was previously reserved for movie productions and enables the techniques to be used to previous special effects.
There are two major issues before making detailed rendering of massive volumes possible: overcome the memory limitations (and propose related update schemes) and the usually costly rendering. Volume data can require large amounts of memory. Thus limiting the scene's extent and resolution of details. The fact that the scene can no longer be held entirely in memory implies the need for an intelligent

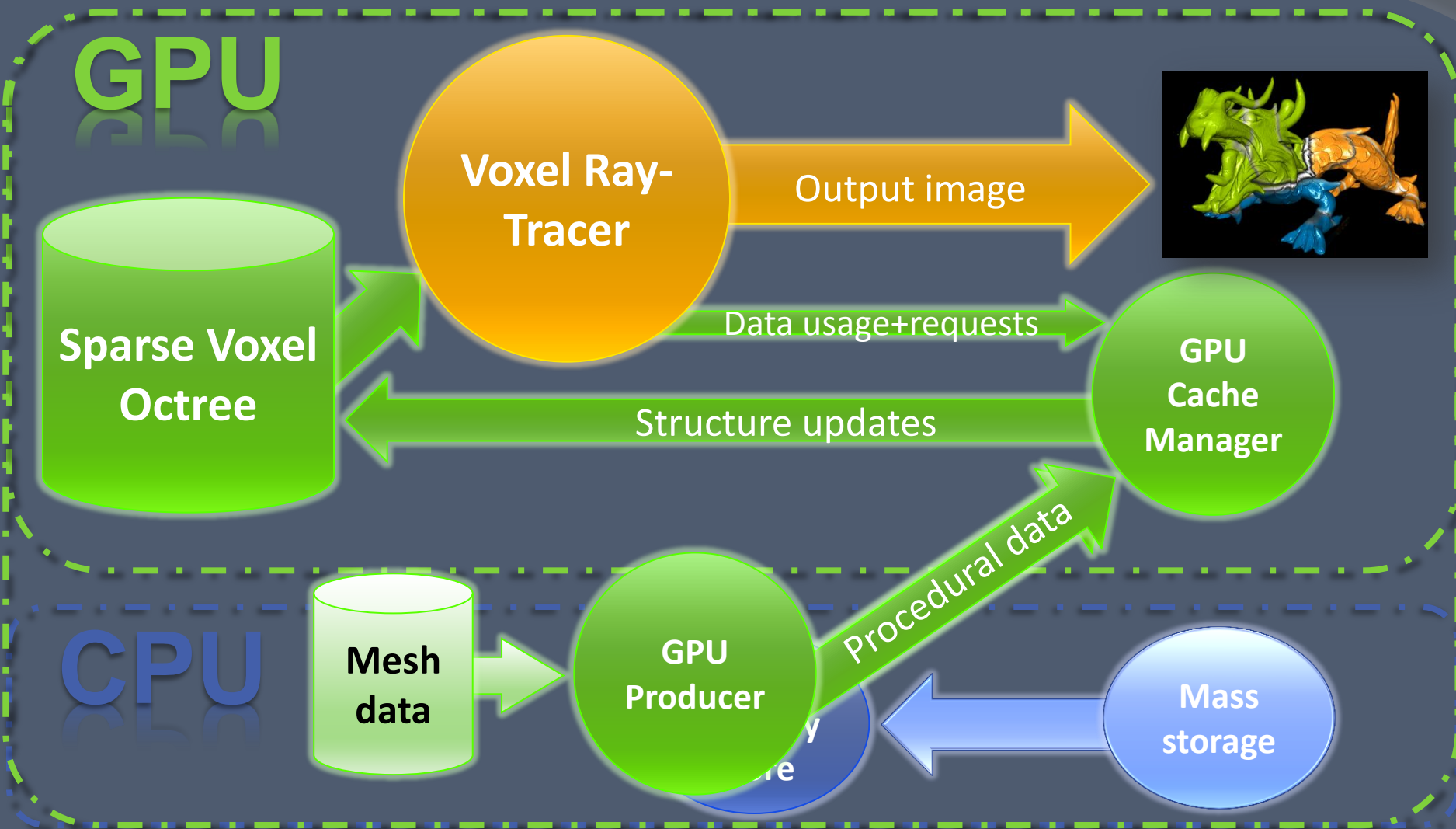


Key ideas

- ⦿ Rendering only dependant on what is visible
 - Ray-tracing approach
- ⦿ Load only needed data, at the needed resolution
 - Occlusion + LOD
 - Ray-guided streaming
- ⦿ Reuse loaded data as much as possible
 - GPU cache mechanism



GigaVoxels CUDA pipeline



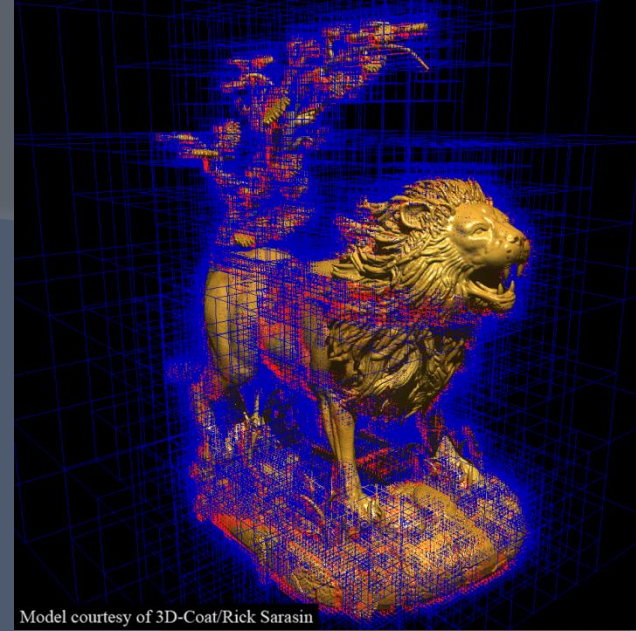


I3D 2008 [BNMBC08]



Voxel sculpting

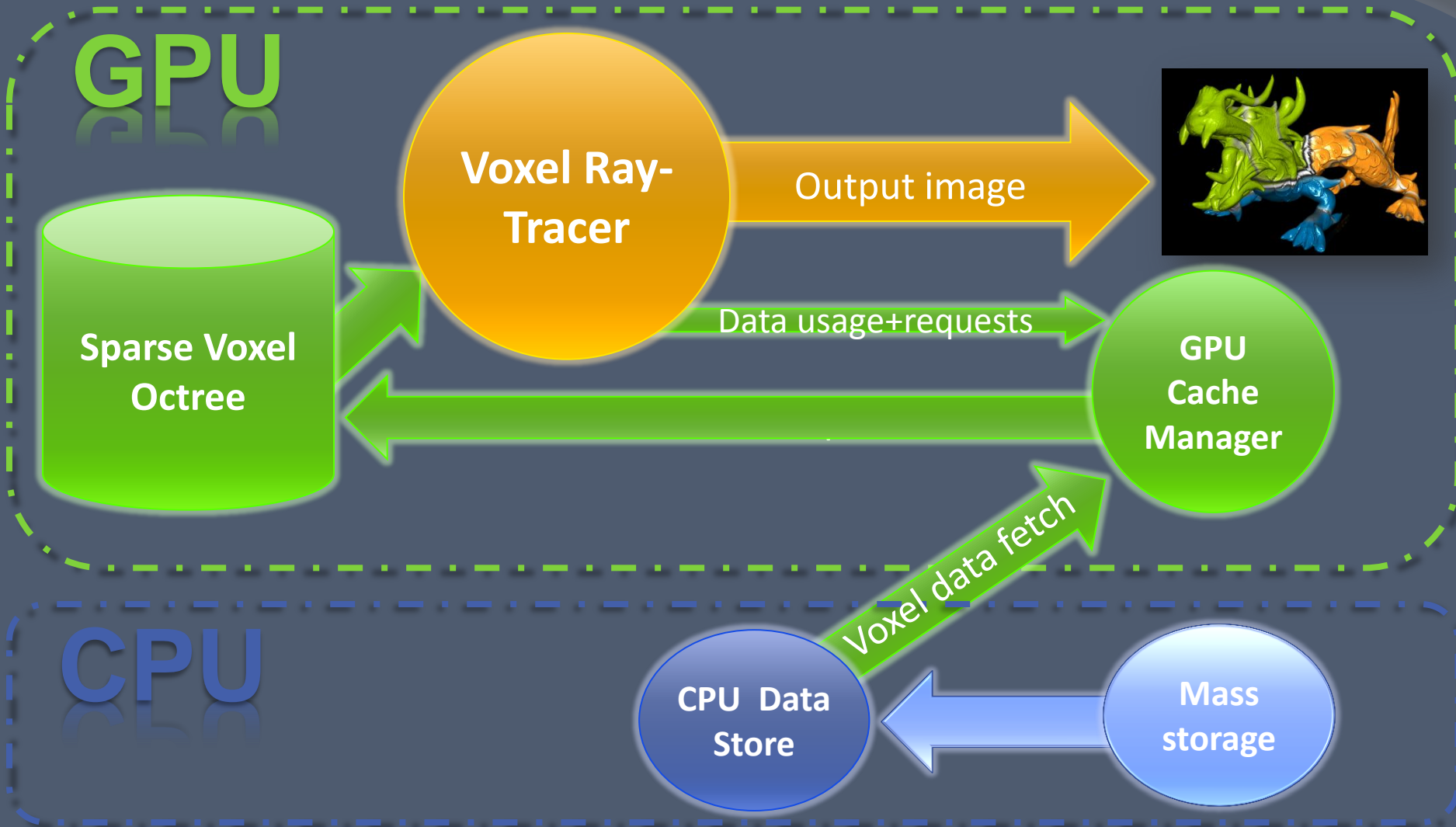
- ① Direct voxel sculpting
 - *3D-Coat*
 - Like *ZBrush*
- ① Generate a lot of details



5-20 FPS

3D COAT

Data Structure



Sparse Voxel MipMap Pyramid

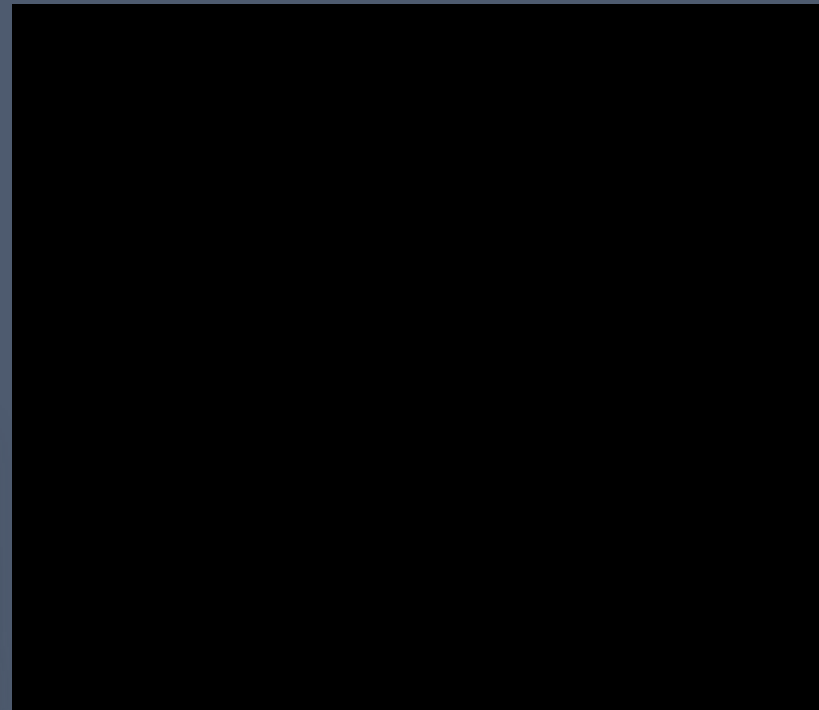
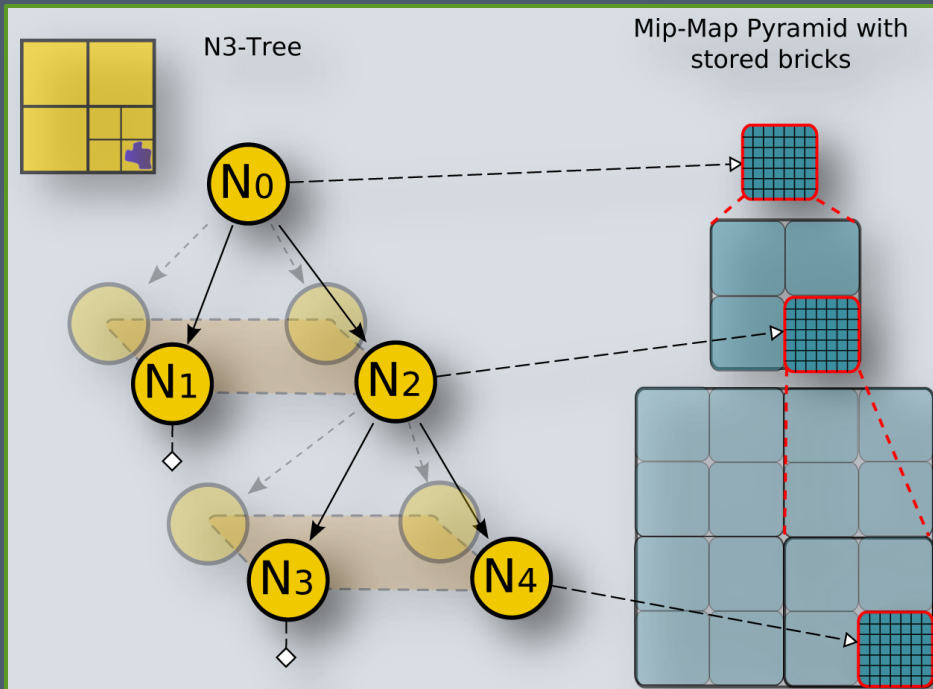
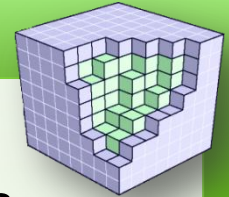
Data structure

Generalized Octree

- Empty space compaction

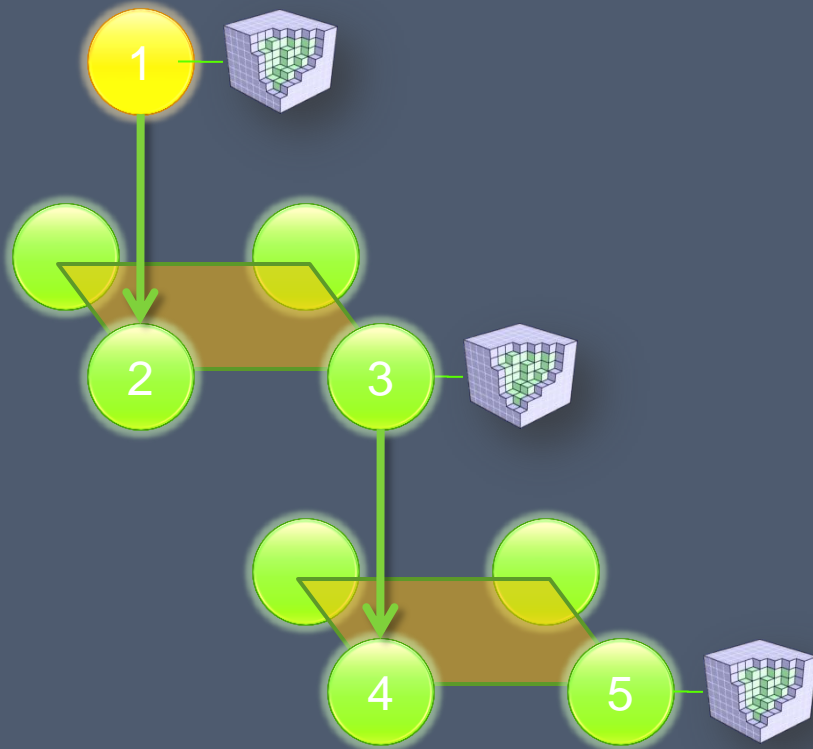
Bricks of voxels

- Linked by octree nodes
- Store opacity, color, normal,...

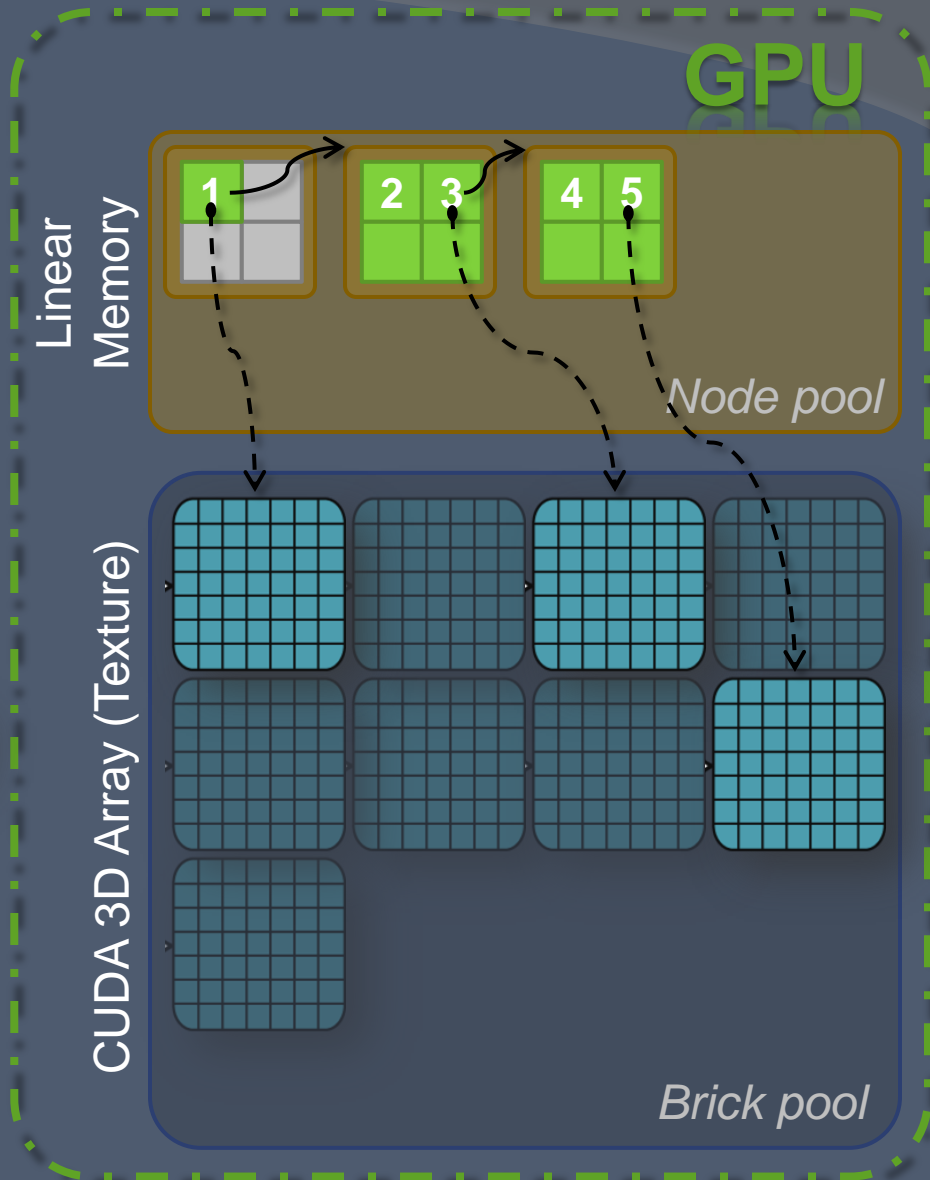


Tower model courtesy of Erklarbar, made with 3DCoat

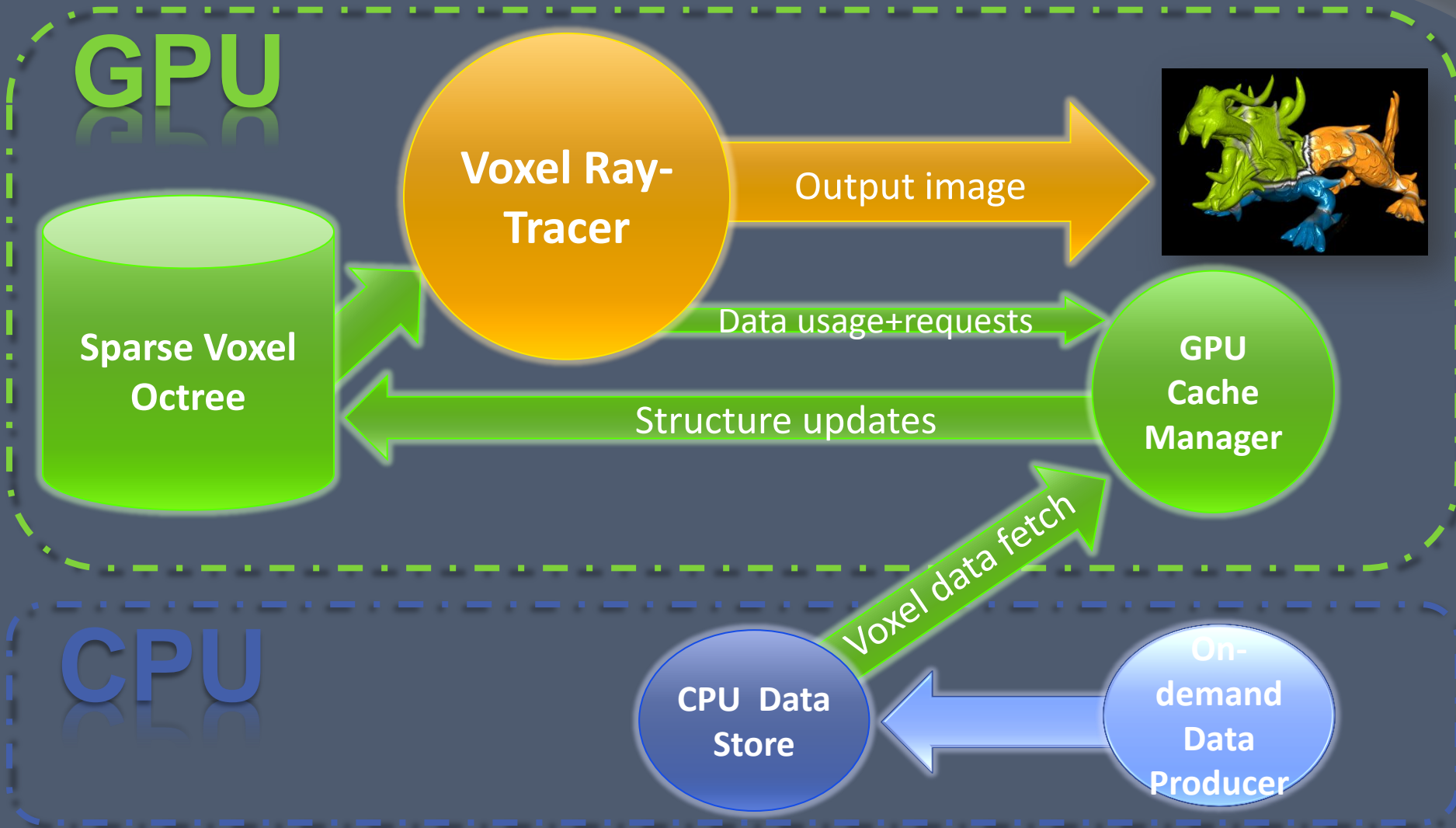
Octree of Voxel Bricks



- One child pointer
 - Compact structure
 - Cache efficient

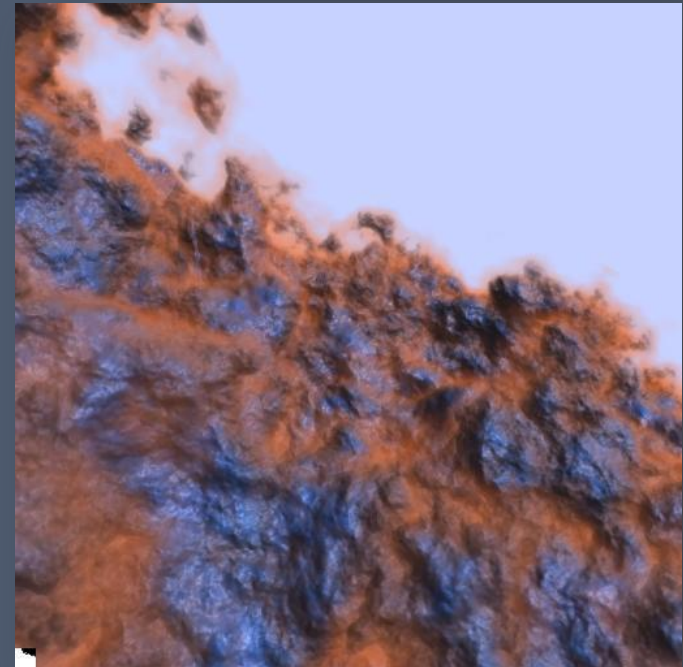
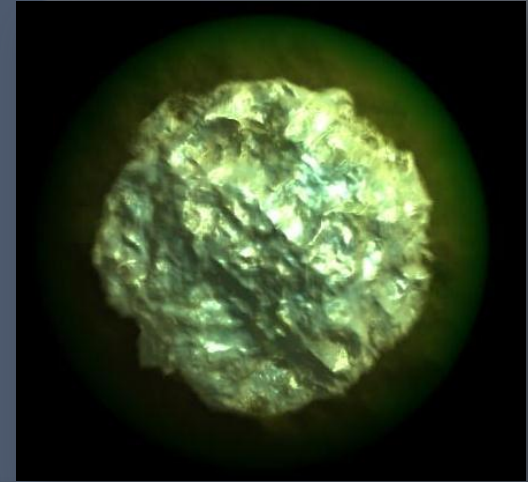


Rendering



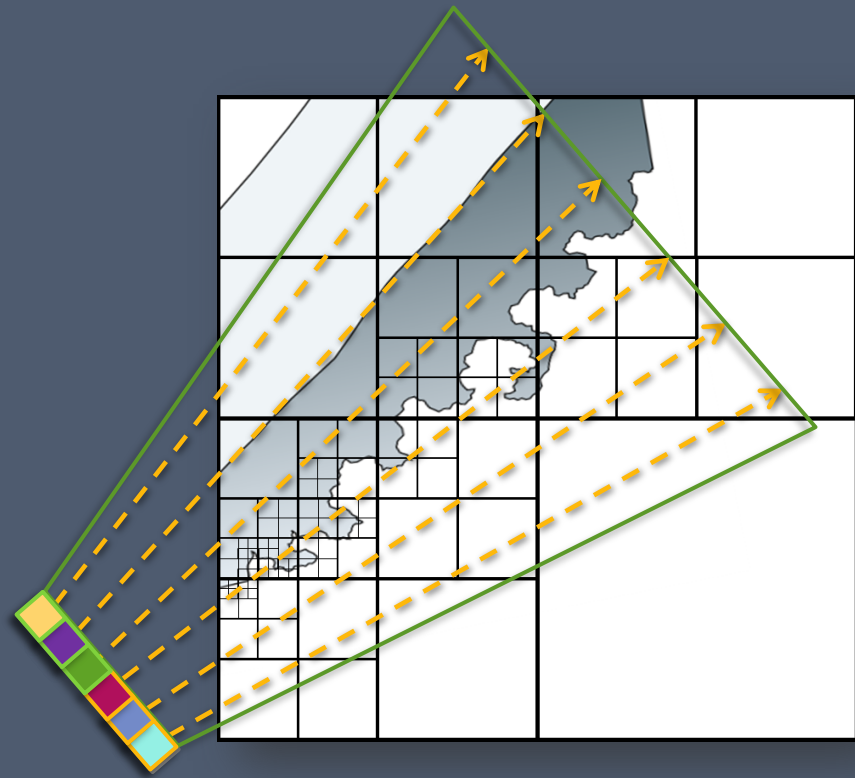
Hierarchical Volume Ray-Casting

- ⦿ Render semi-transparent materials
 - Participating medias
- ⦿ Emission/Absorption model for each ray
 - Accumulate Color intensity + Alpha
 - Front-to-back
 - Stop when opaque



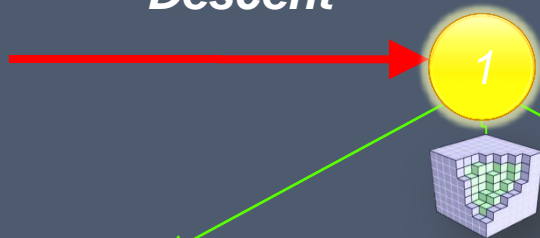
Hierarchical Volume Ray-Casting

- Volume ray-casting
 - [Sch05, CB04, LHN05a, Olick08, GMAG08, CNLE09]
- One big CUDA kernel
 - One thread per ray
- Octree traversal
 - KD-restart algorithm [FS05]
 - Ray-driven LOD
- Bricks marching
 - Regular sampling into the 3D texture



Volume Ray-Casting

*Tree
Descent*



3
*Skip
Node*

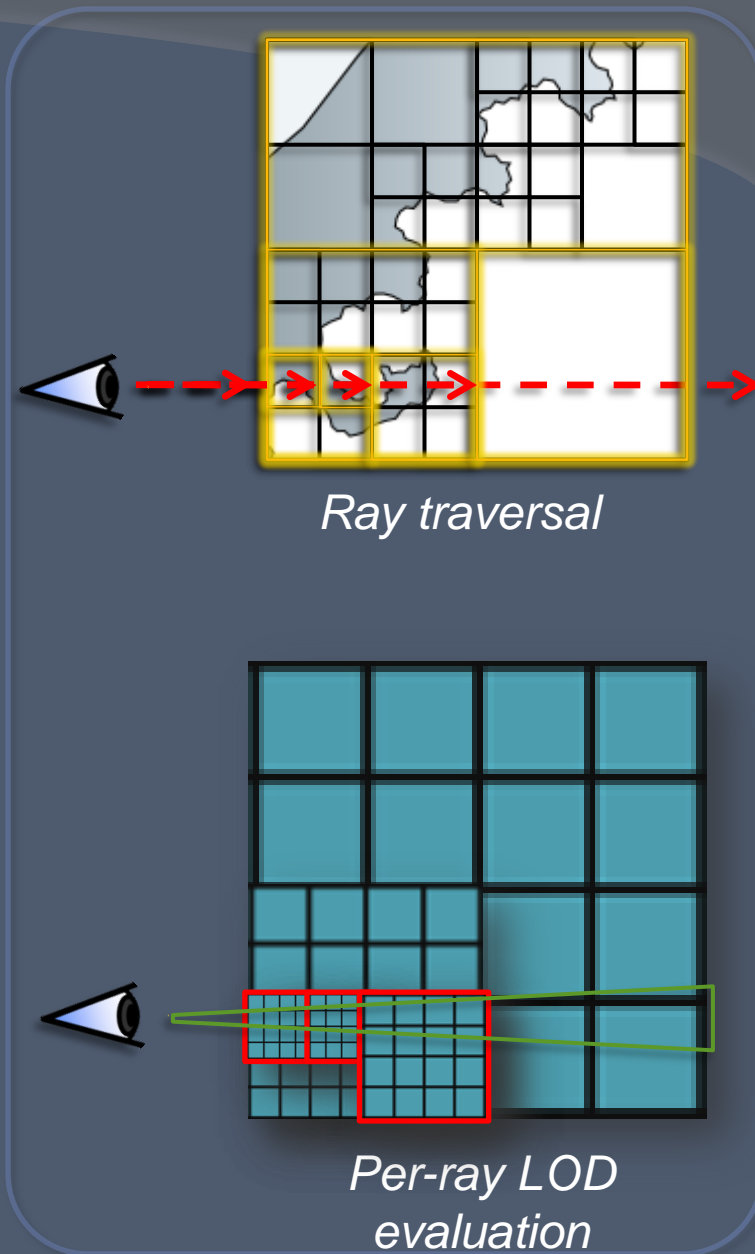


*Brick
Marching*



*Brick
Marching*

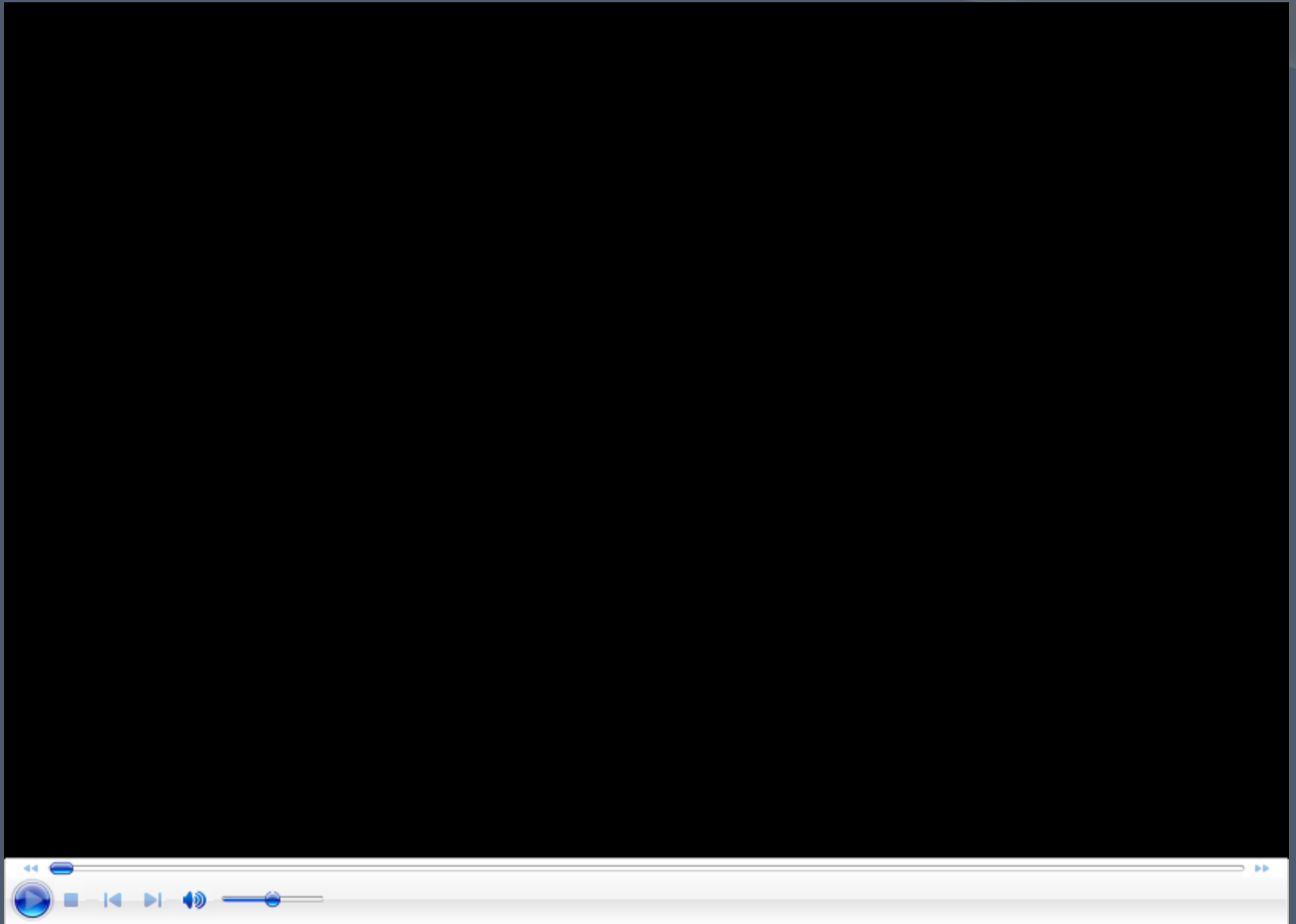
*Brick
Marching*



Ray traversal

*Per-ray LOD
evaluation*

Rendering costs

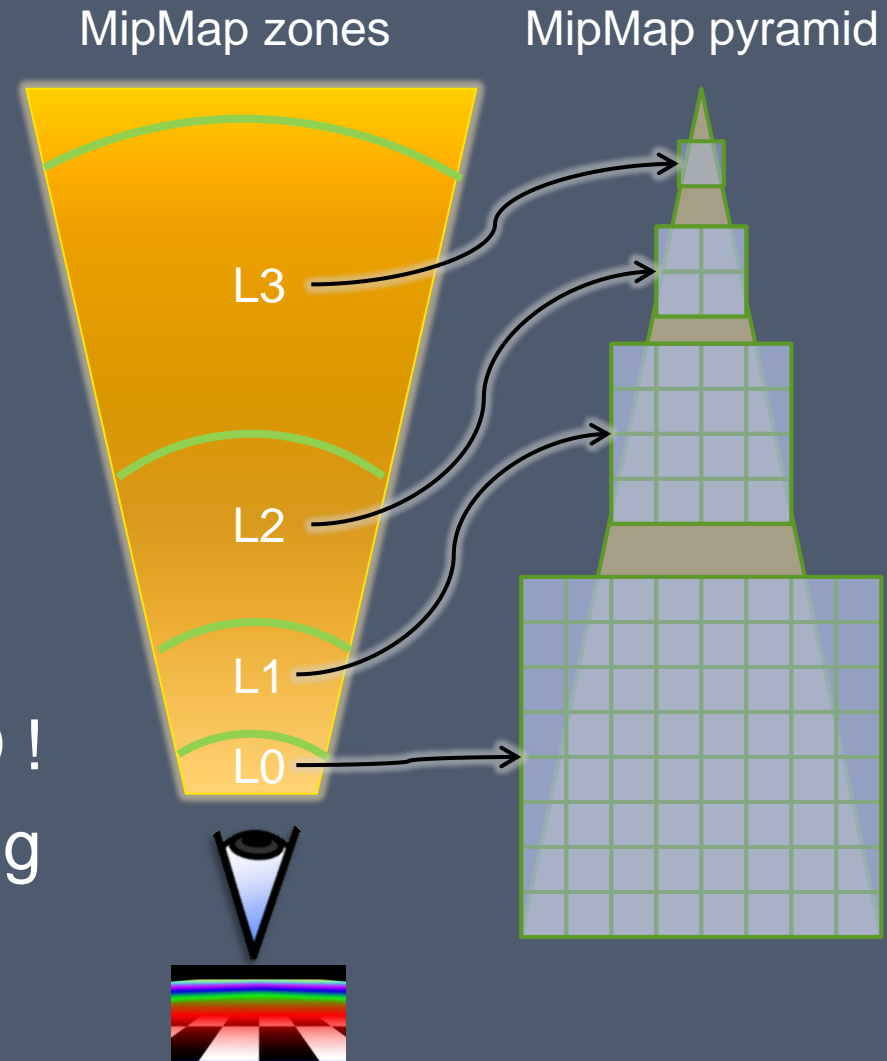


Volume MipMapping mechanism

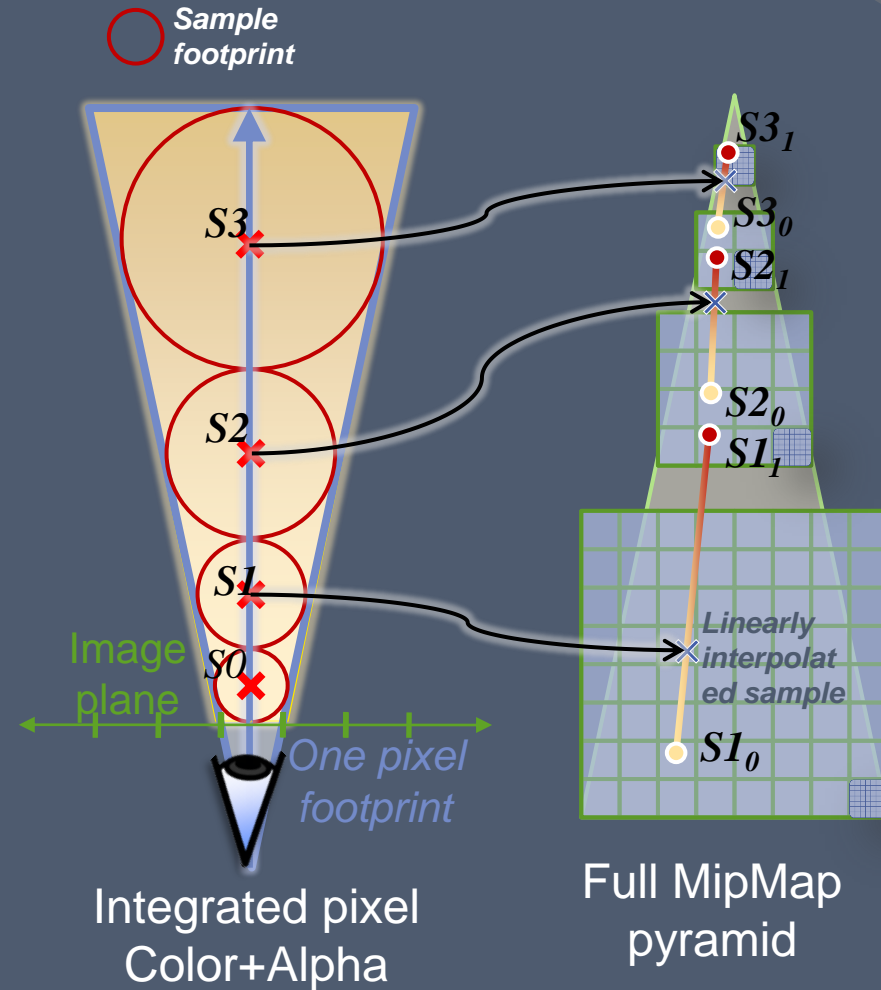
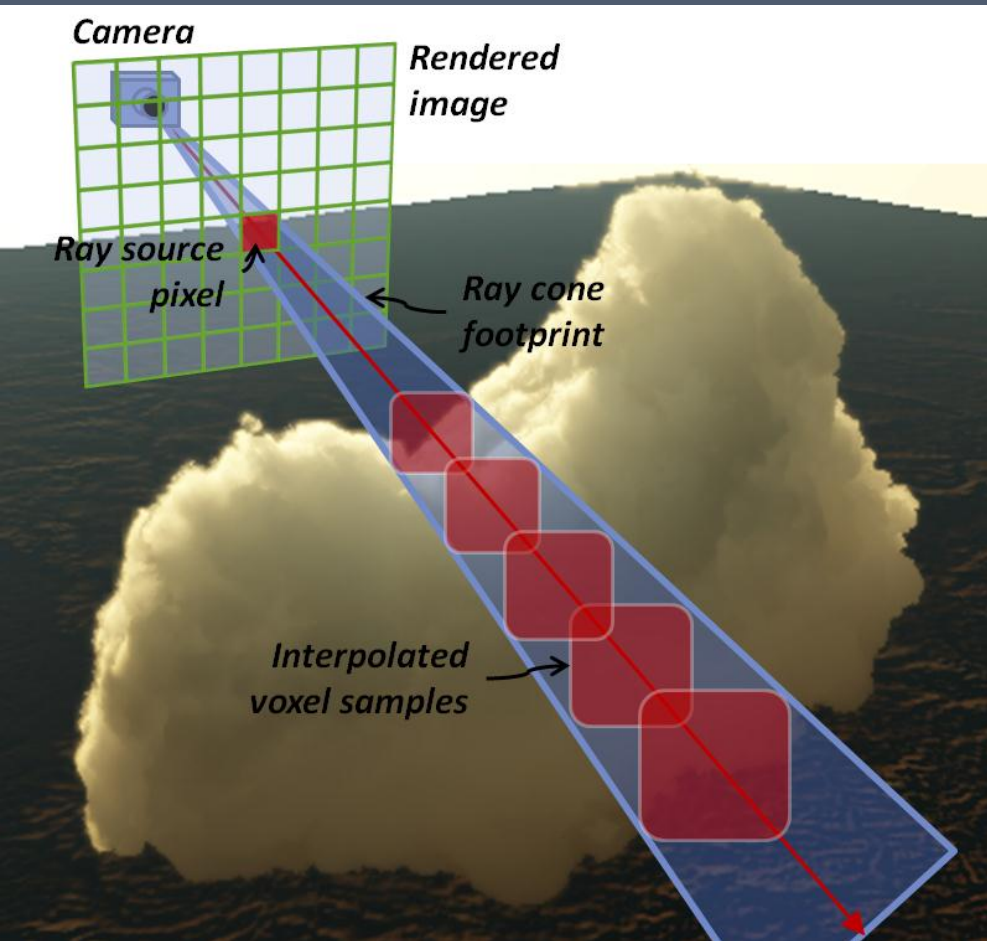
Problem: LOD uses discrete downsampled levels

- Popping + Aliasing
- Same as bilinear only for 2D textures

- Geometry is texture 😊
- Uses pre-integrated LOD !
- No need of multi-sampling (eg. MSAA)



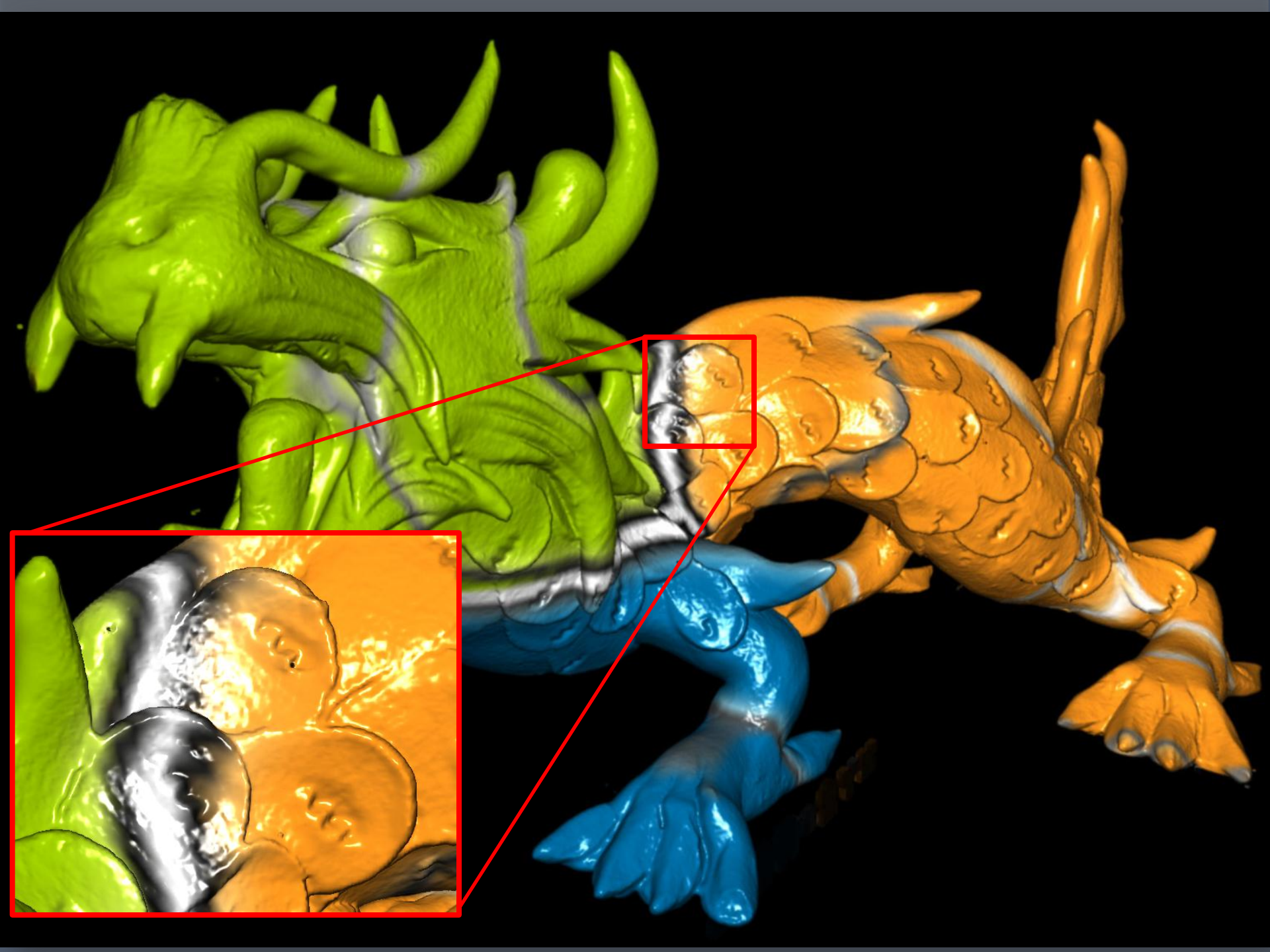
Cone tracing



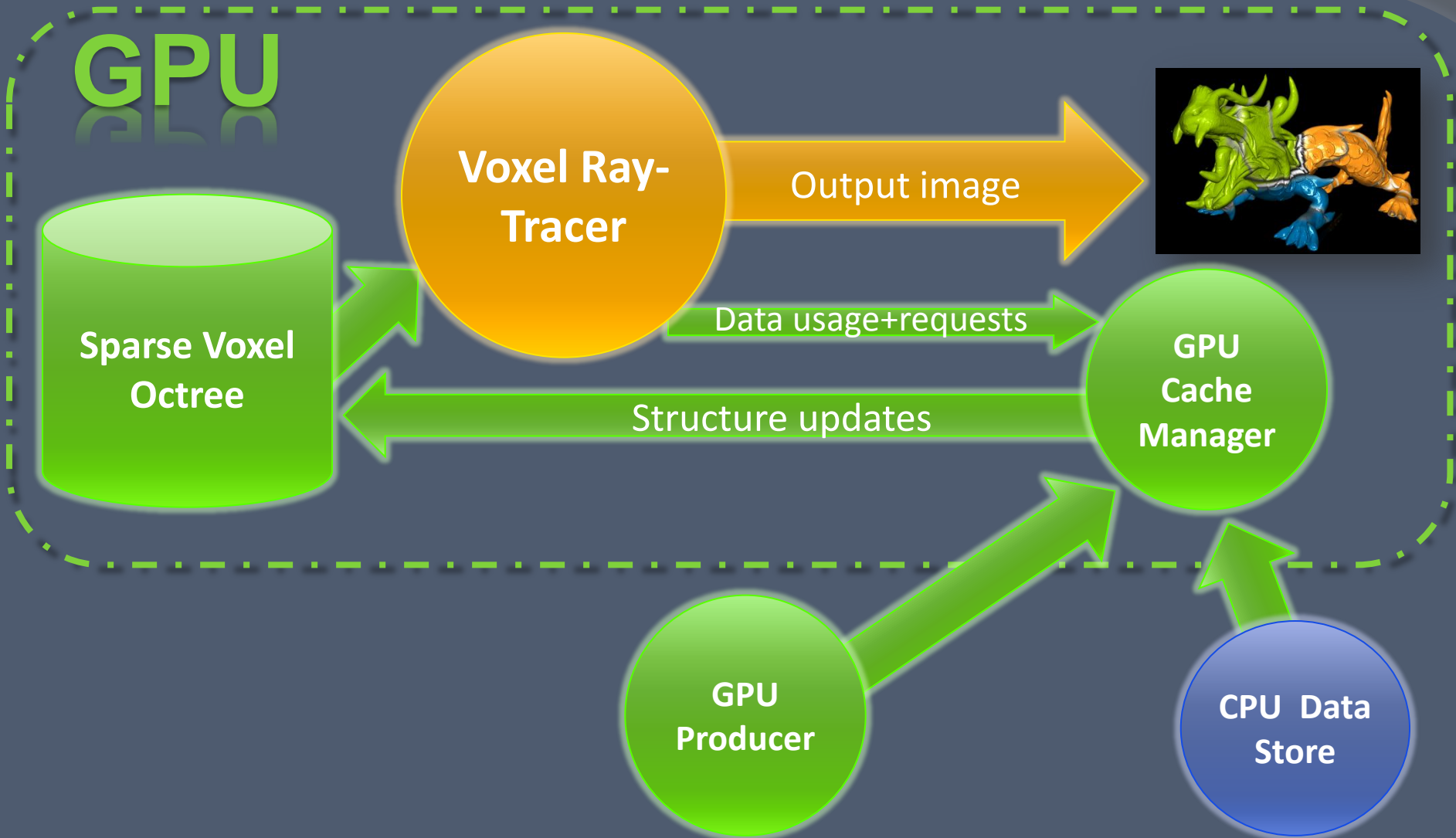
Shading computation

- ◎ Standard Blinn-Phong illumination
 - Per sample
- ◎ Normal information
 - On-the-fly gradient with finite differences
 - Stored normal information
- ◎ Deferred for opaque objects



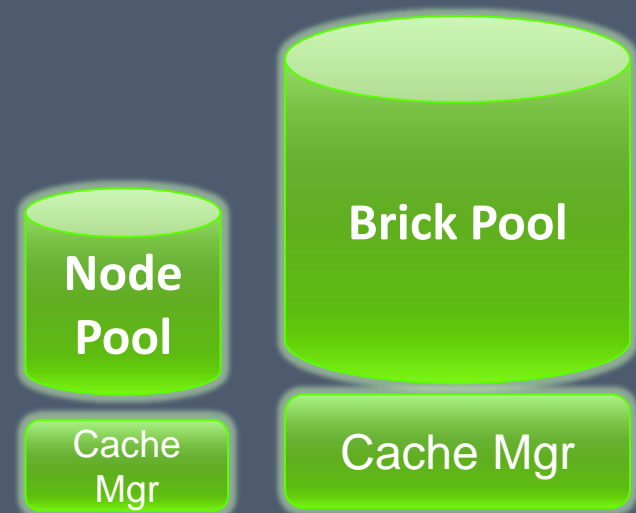


Data Management



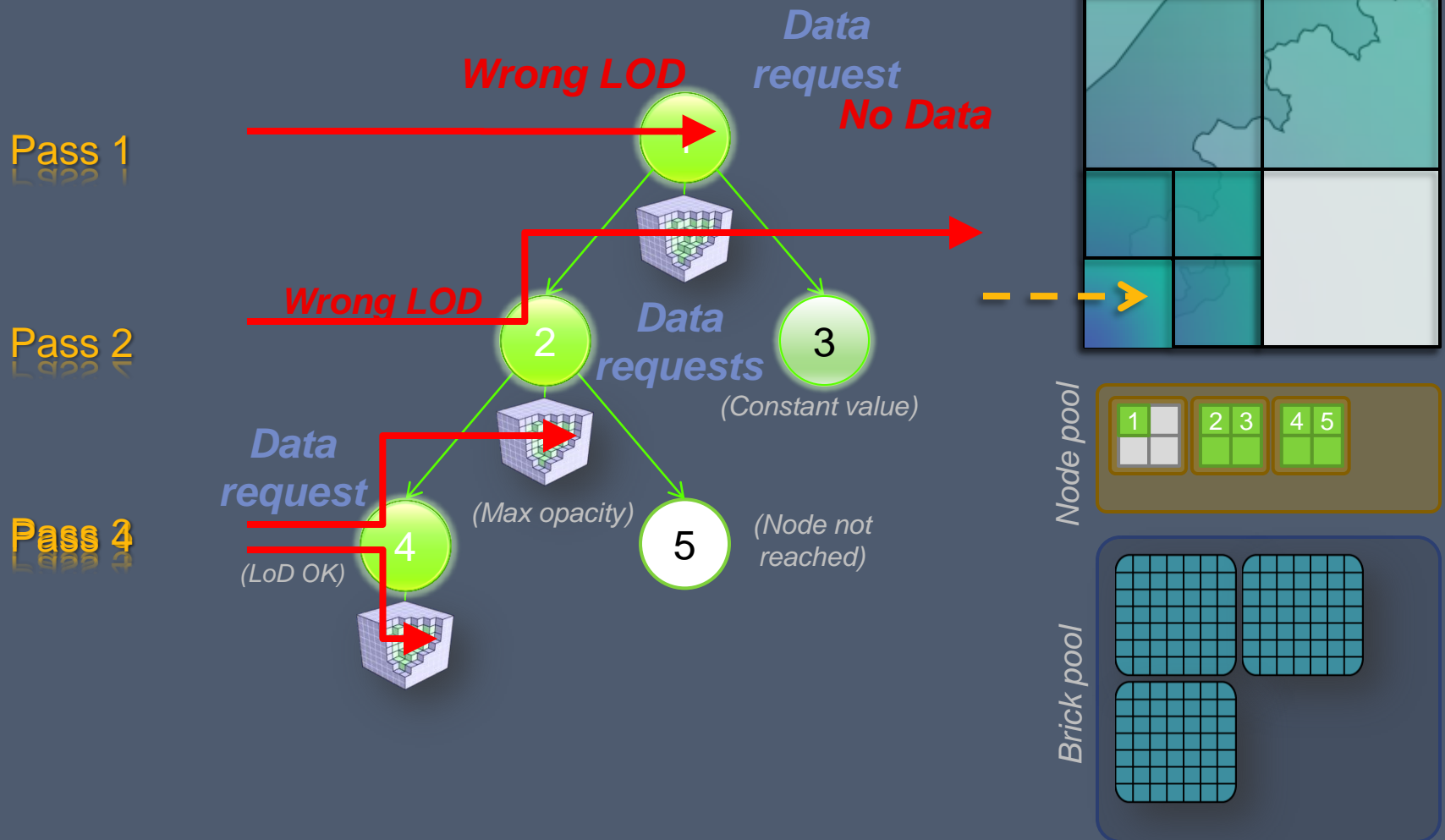
GPU Caches

- ◎ Data management made through a cache mechanism
 - Used for both the **node pool** and **brick pool**
 - Allows full scalability
- ◎ Rely on the octree to address elements
 - The node pool is addressing itself !
 - No page table
- ◎ Data requests generated by the ray-tracing
 - Node subdivision
 - Brick loading



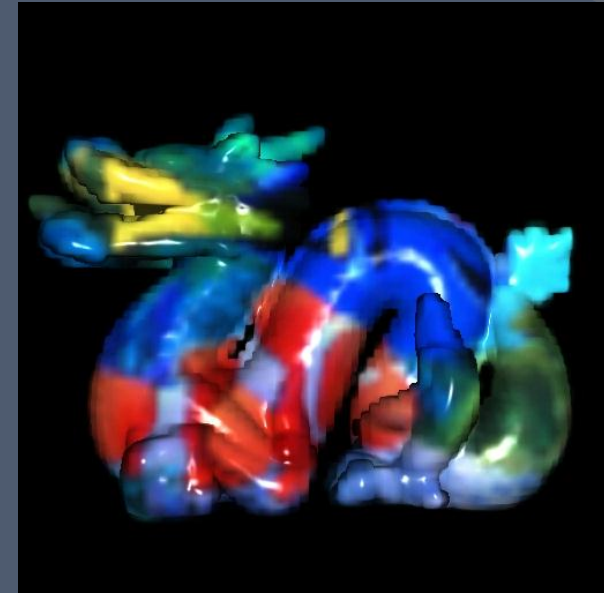
Incremental octree update

Progressive loading



Ray-based visibility & requests

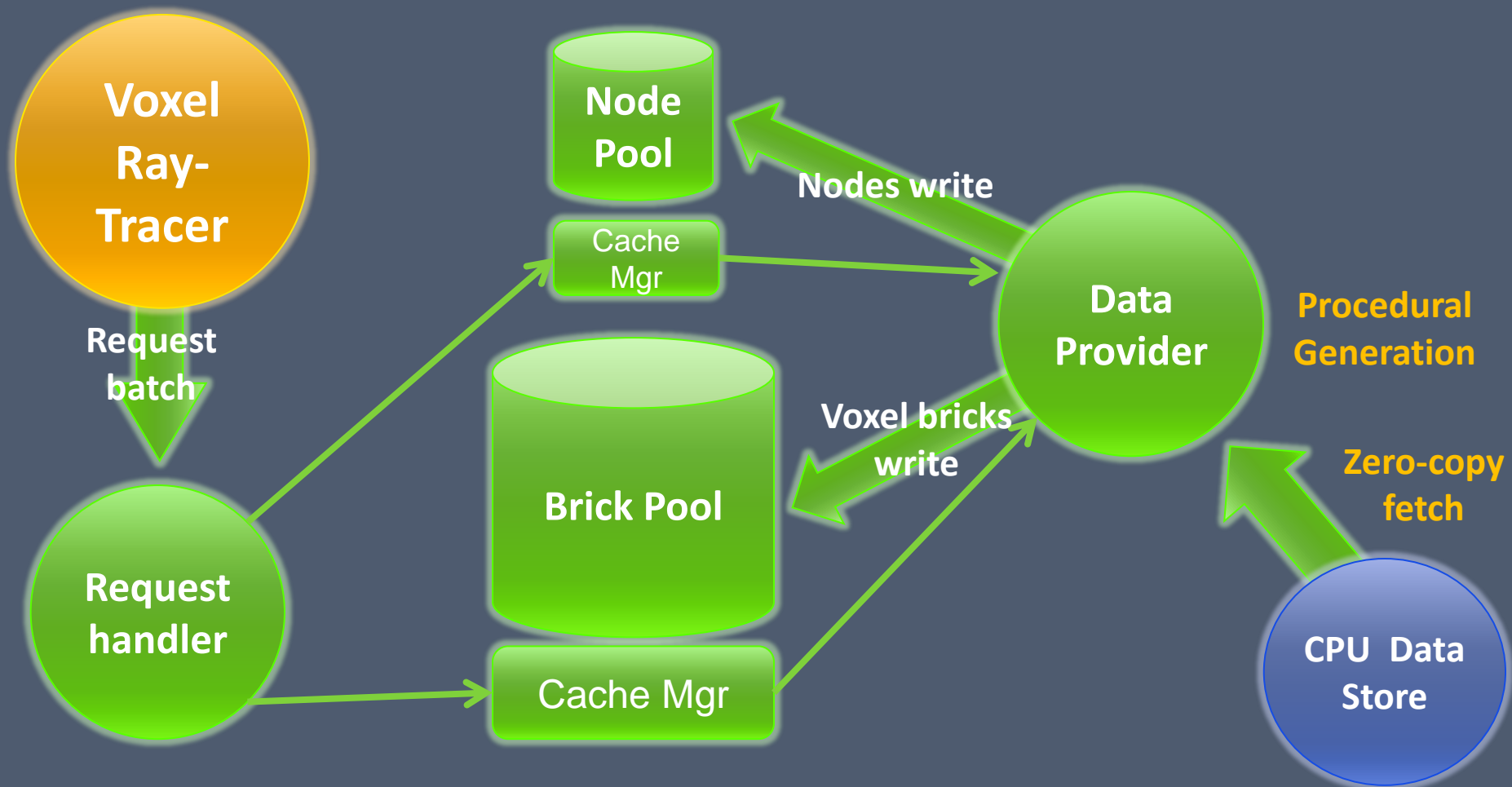
- Minimum amount of data is loaded



- Progressive refinement
 - Always ensure interactivity
- Fully compatible with secondary rays and exotic rays paths
 - Reflections, refractions, shadows, curved rays, ...

Cache requests handling

- Entirely handled on the GPU



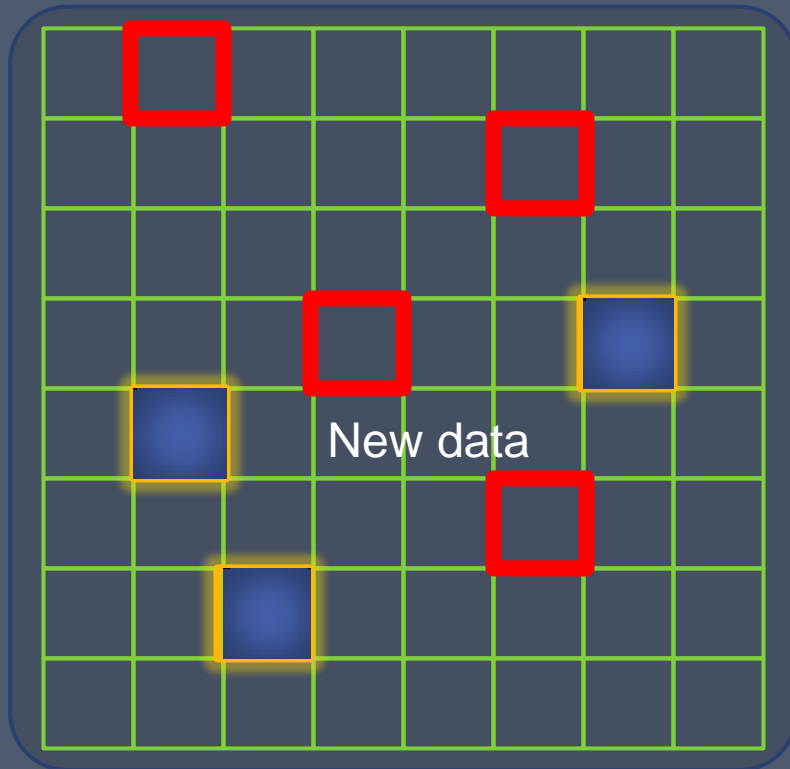
Cache strategy

- ◎ Least Recently Used (LRU) strategy
 - Older elements replaced first
- ◎ Sorted usage list maintained for each cache on the GPU.
- ◎ Usage info provided by the ray-tracer
- ◎ Maintained as a data-parallel process
- ◎ Used when new elements have to be inserted

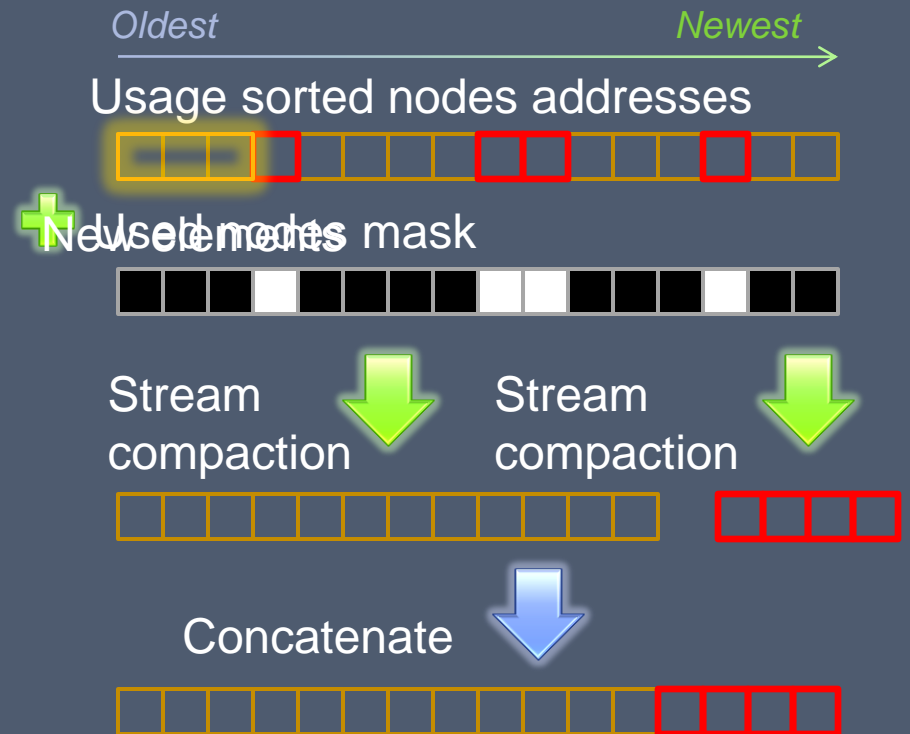
SVMP caches

- LRU (Least Recently Used)
 - Track elements usage
 - Maintain list with least used in front

Cache Elements (*Node Tile/Brick*)

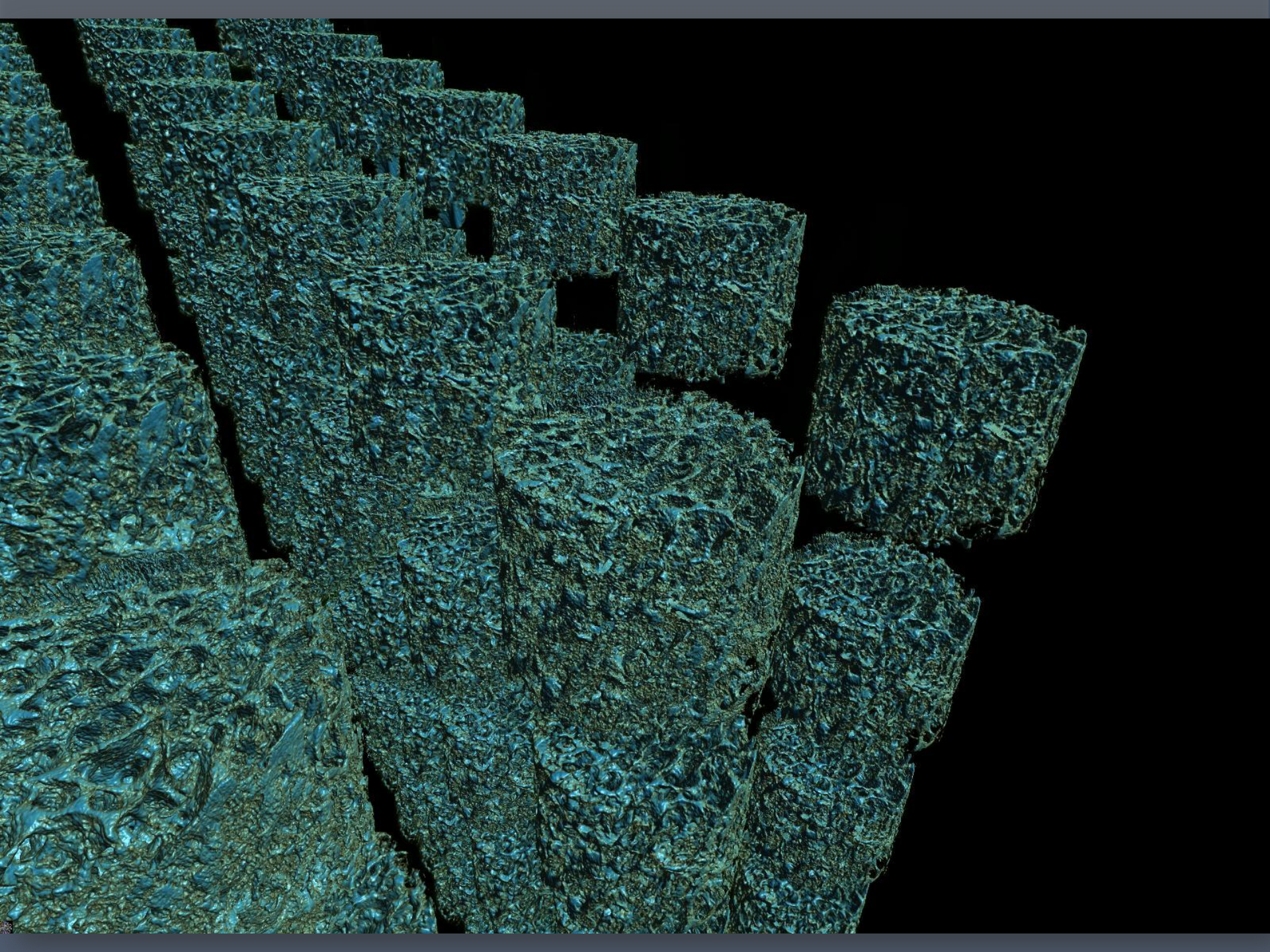


Octree/Bricks Pool



Global cache characteristics

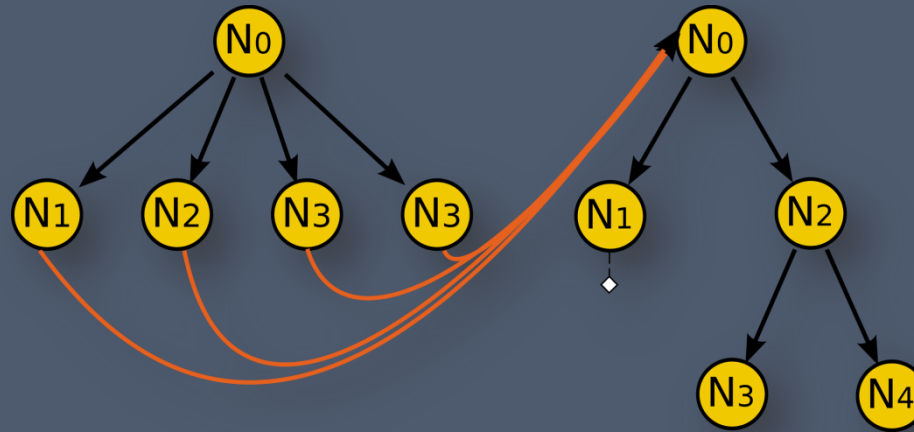
- ① Driven by ray-tracing
- ① Fully managed on the GPU
 - Zero CPU intervention apart kernel launches.
 - Leads to fully on-chip structure management and building
- ① More efficient when large amount of updates



APPLICATIONS

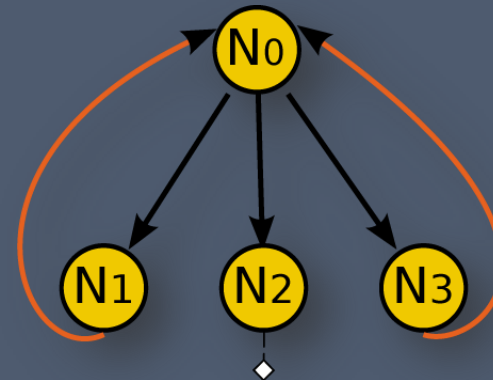
Voxel data synthesis

Instantiation



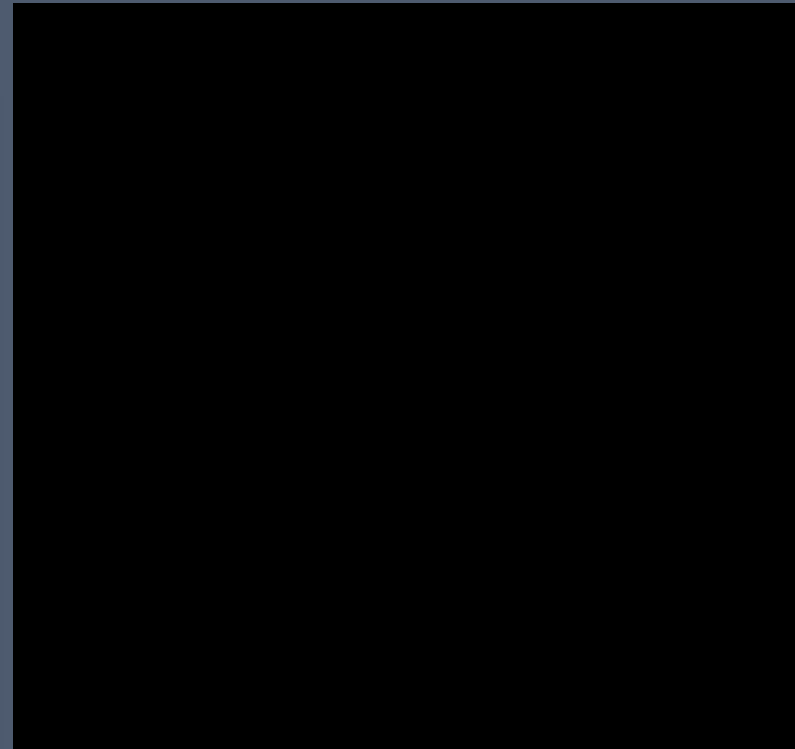
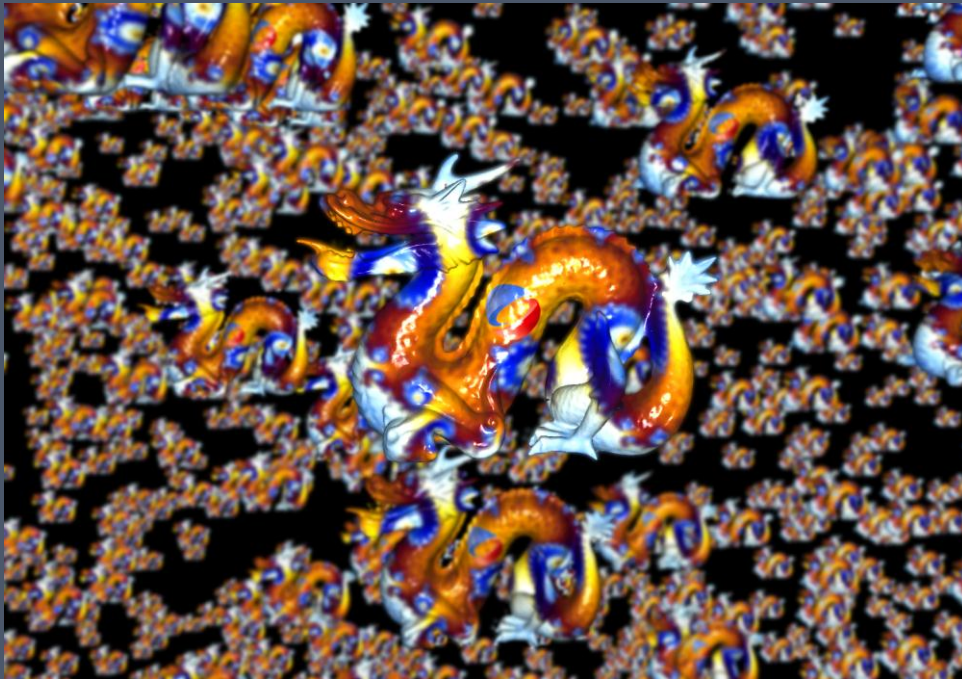
Recursivity

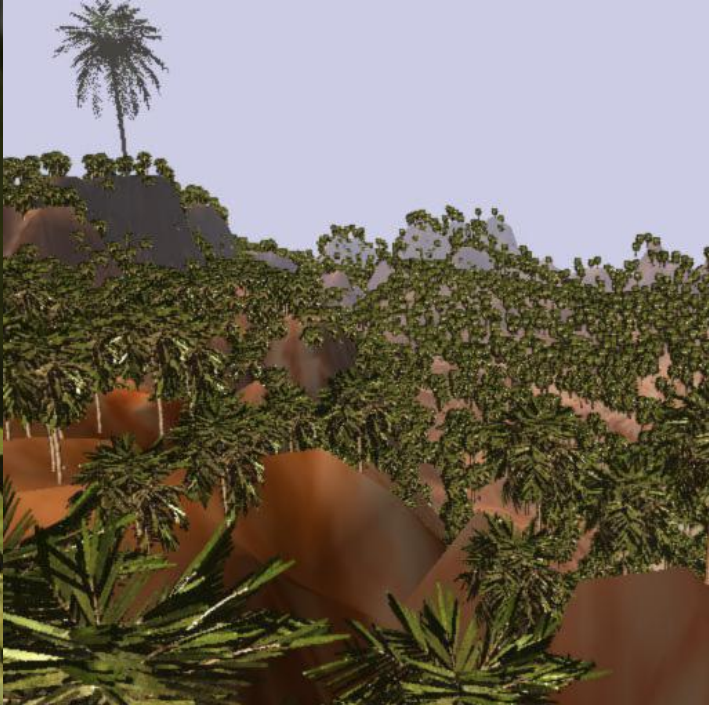
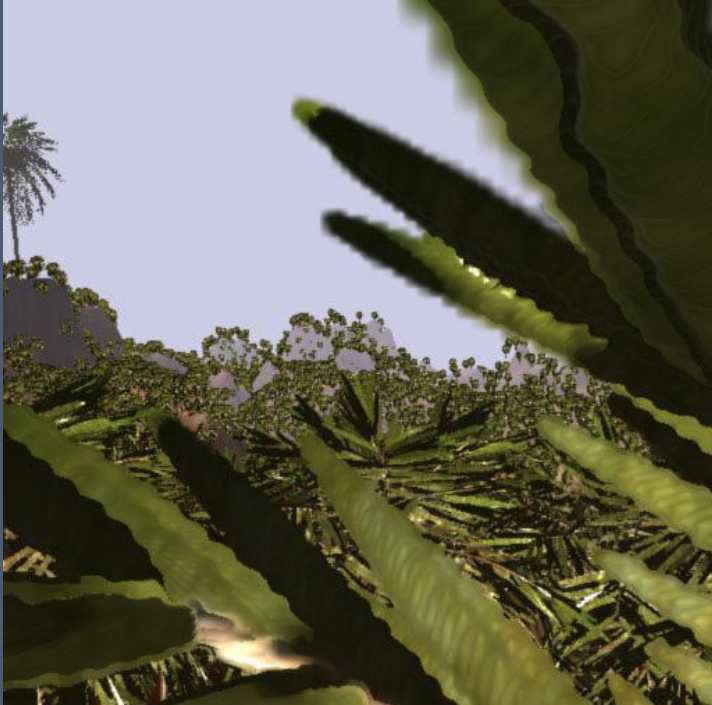
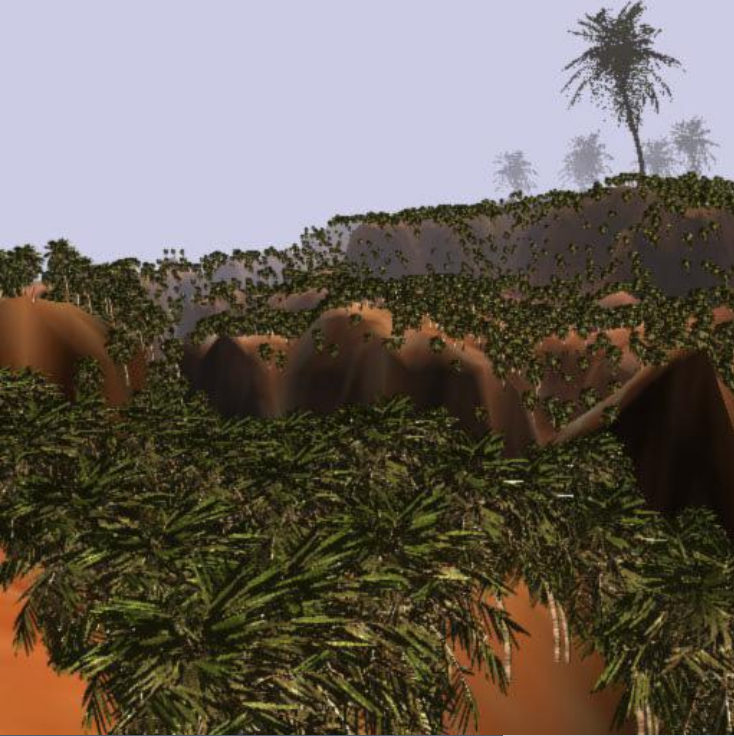
- Infinite details



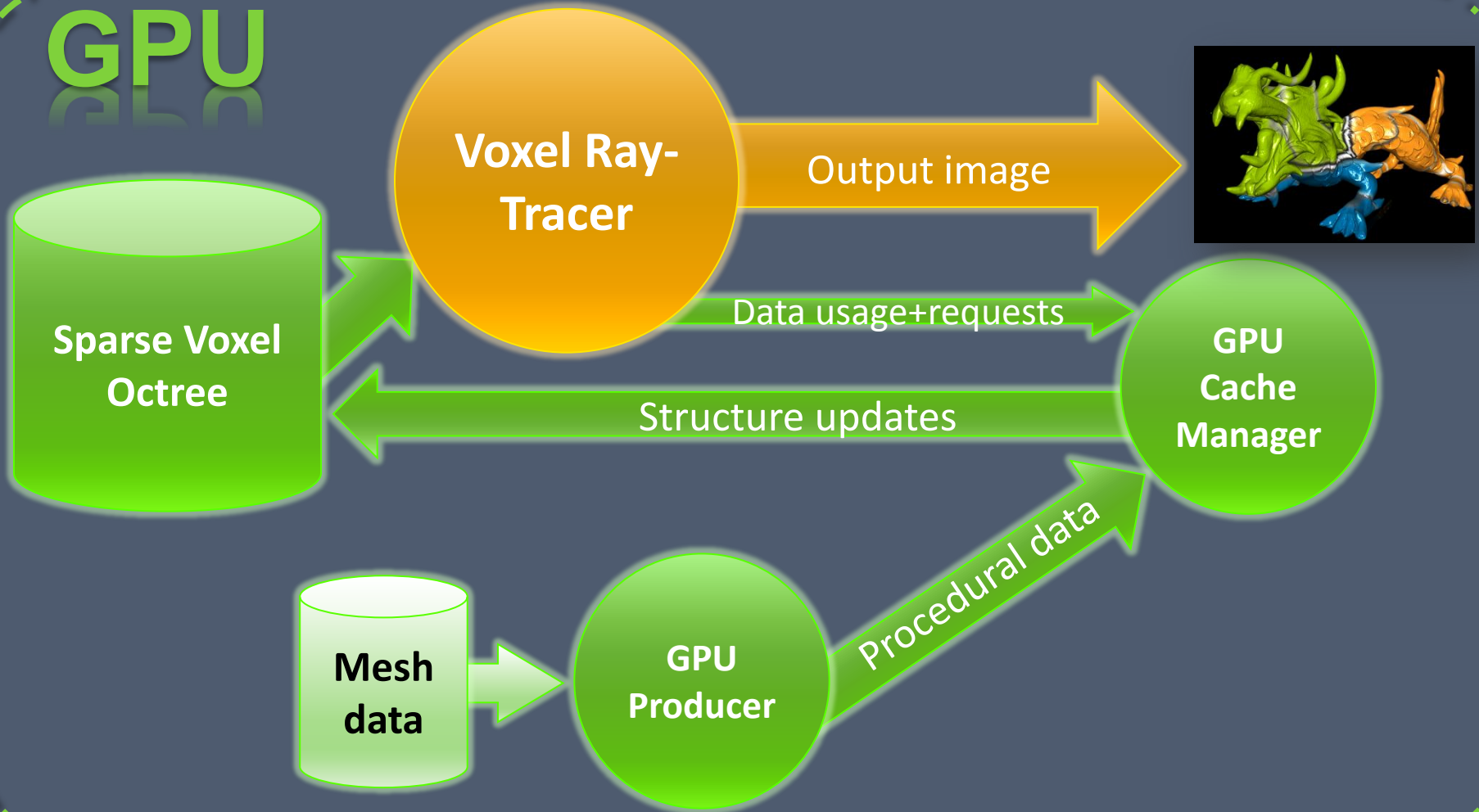
Free voxel objects instancing

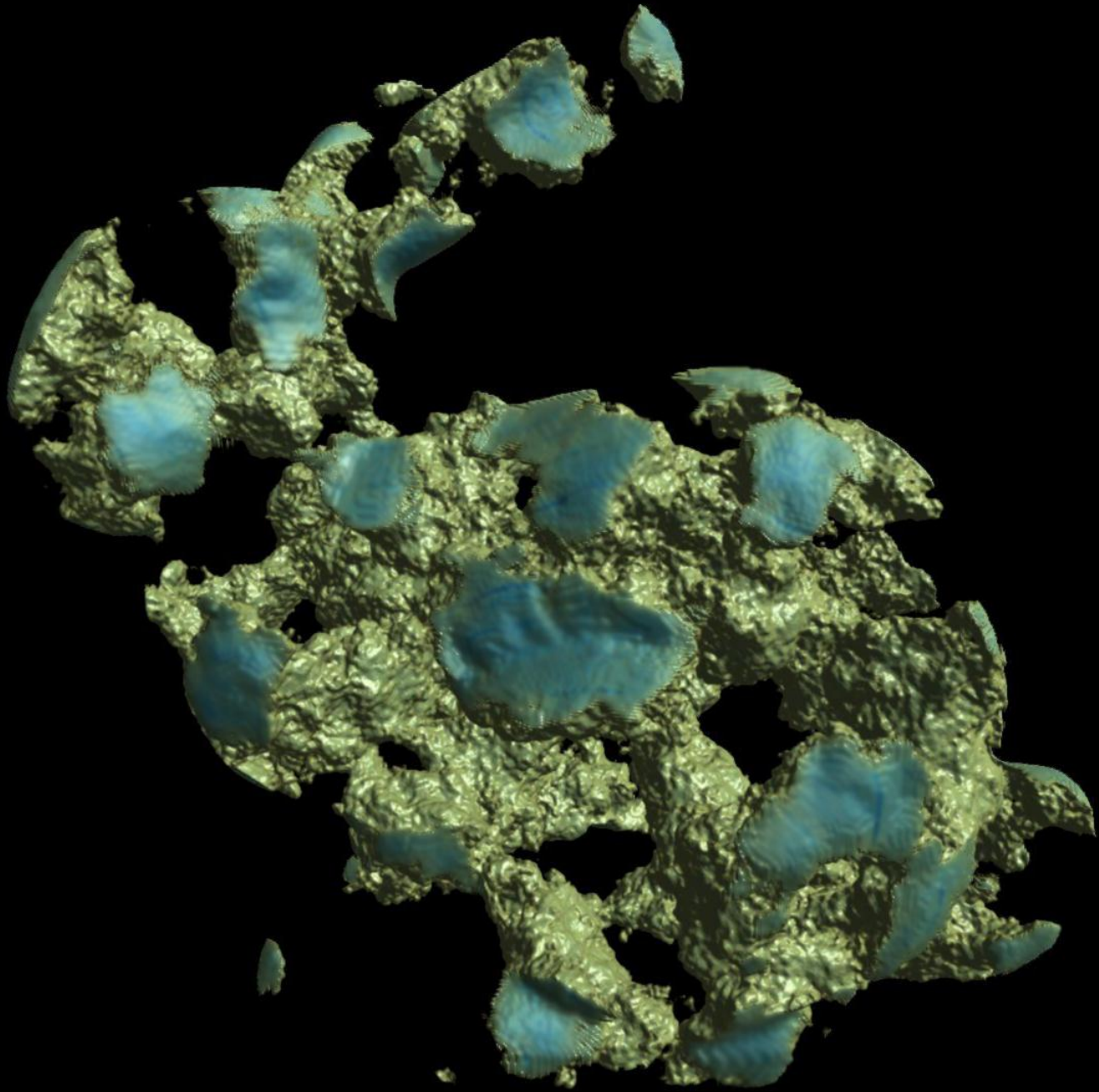
- BVH structure ray-casting
 - Cooperative ray packet traversal [GPSS07]
 - Shared stack
- WA-Buffer
 - Deferred compositing





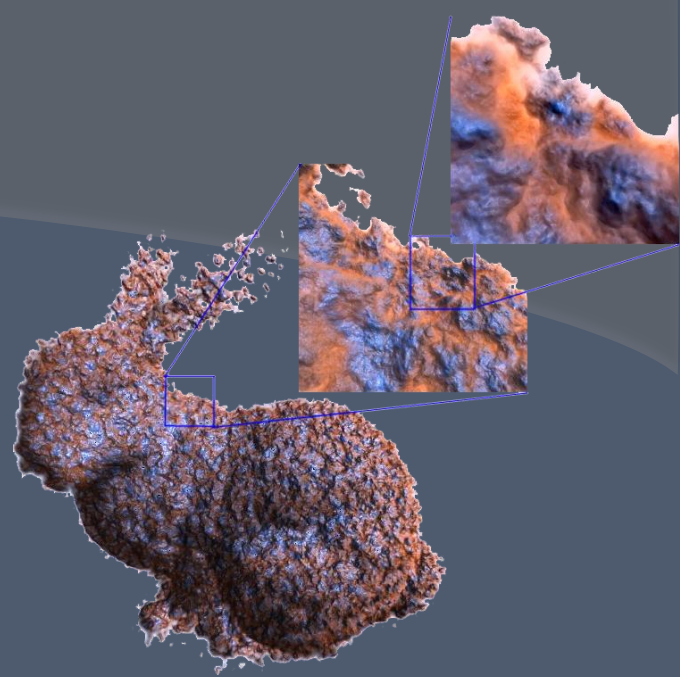
Voxels generation

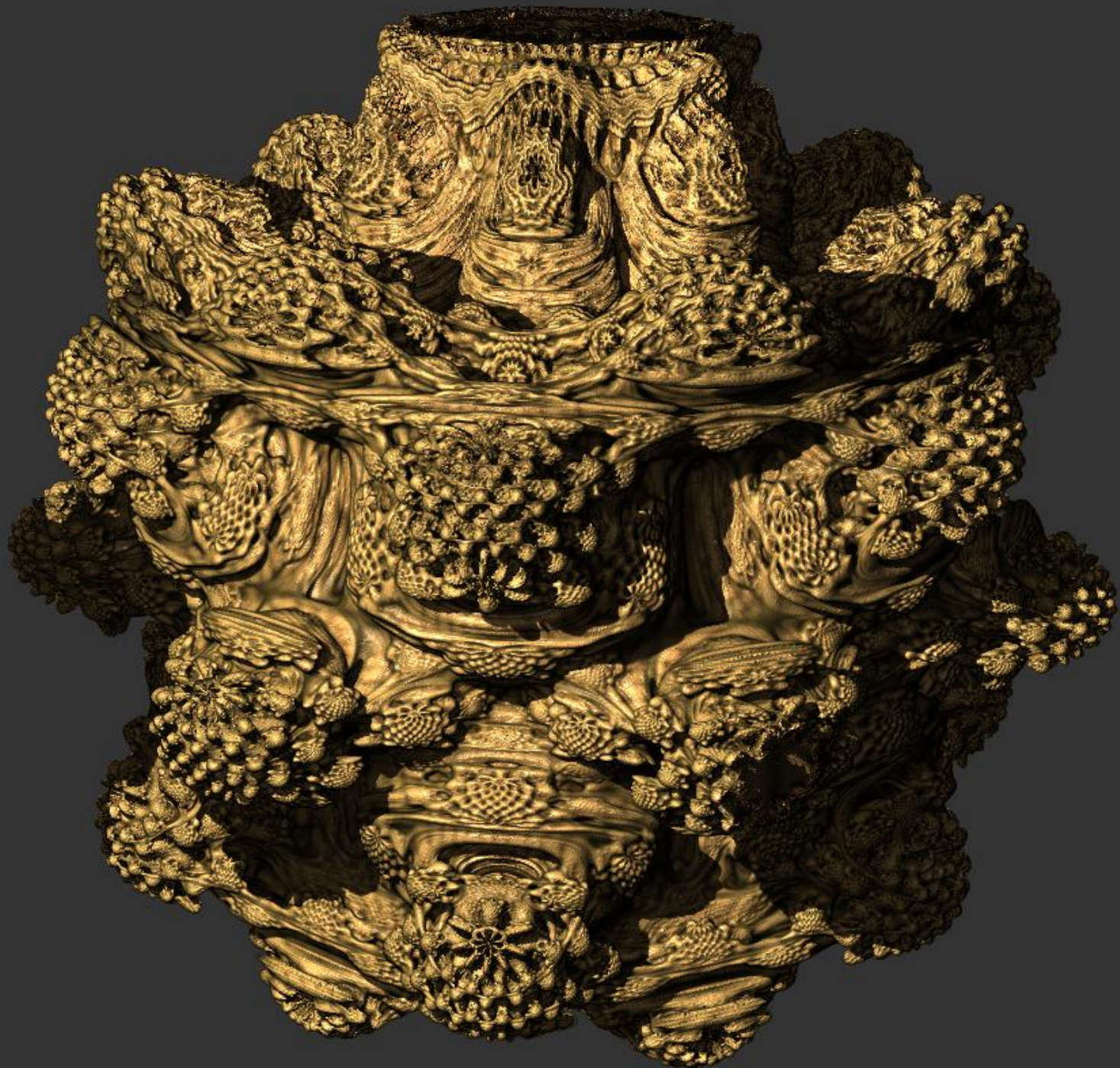


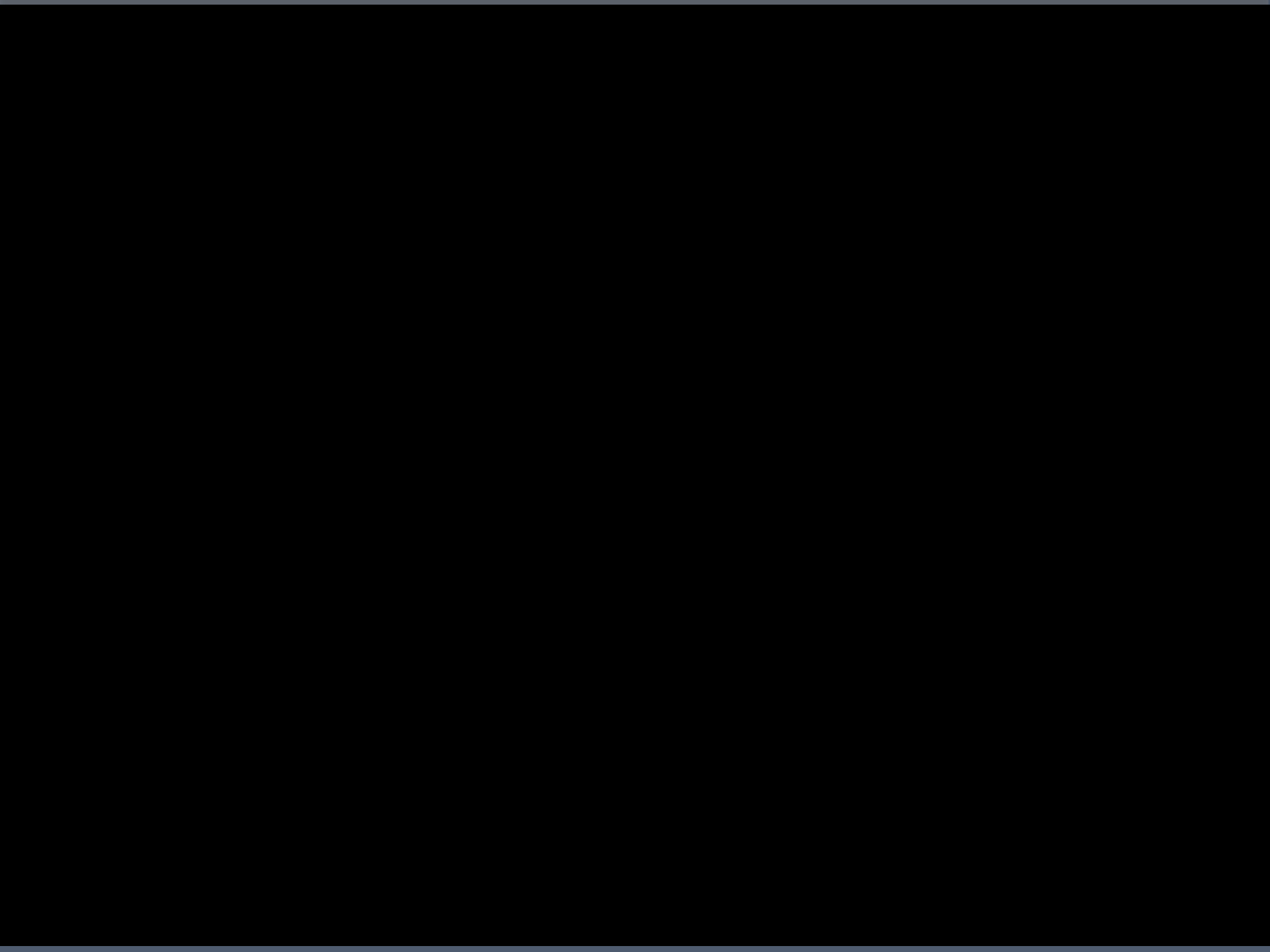


Procedural noise

- ⦿ On-the-fly mesh voxelization
 - Distance field
- ⦿ Procedural noise

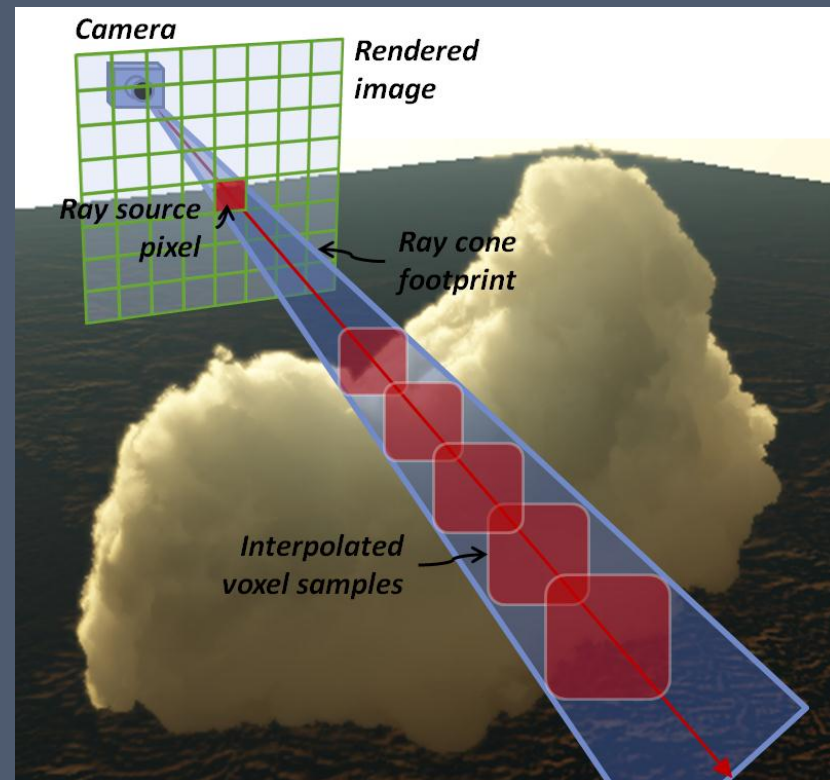






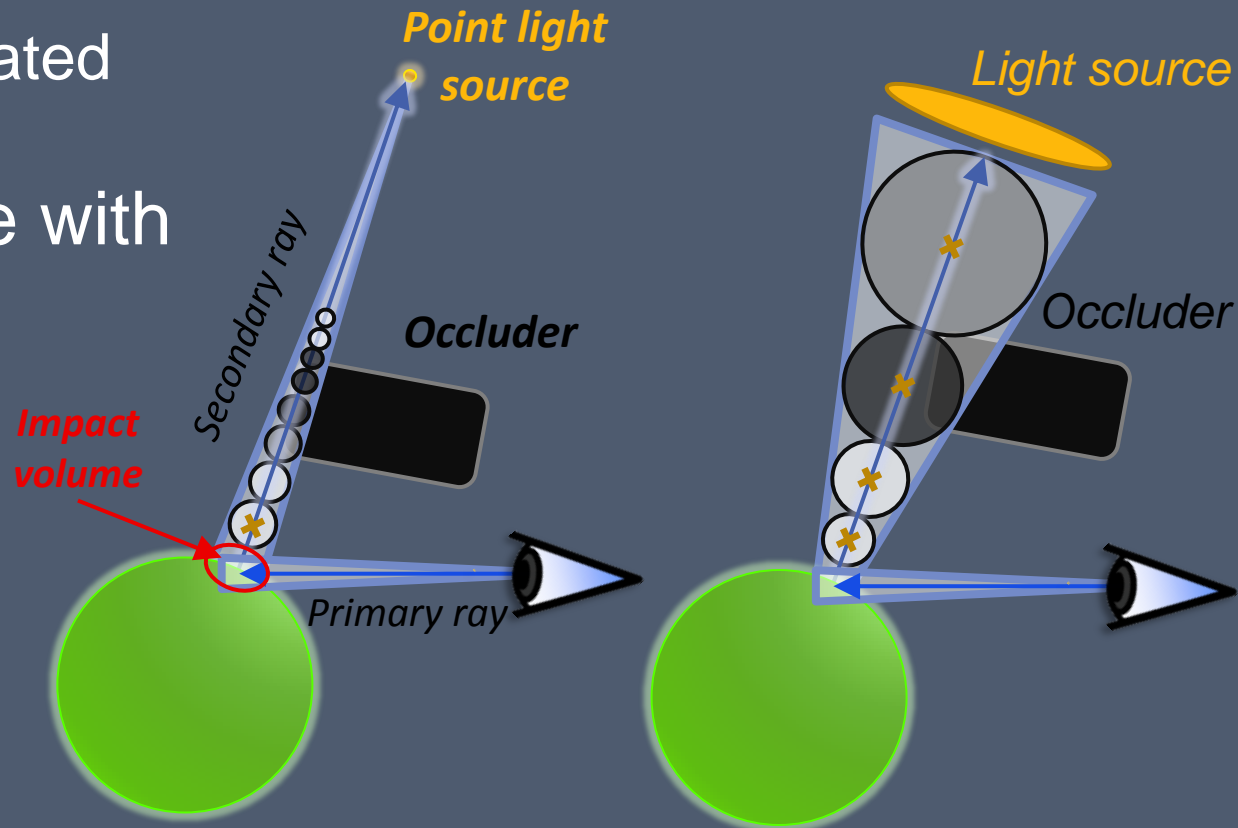
Cool Blurry Effects

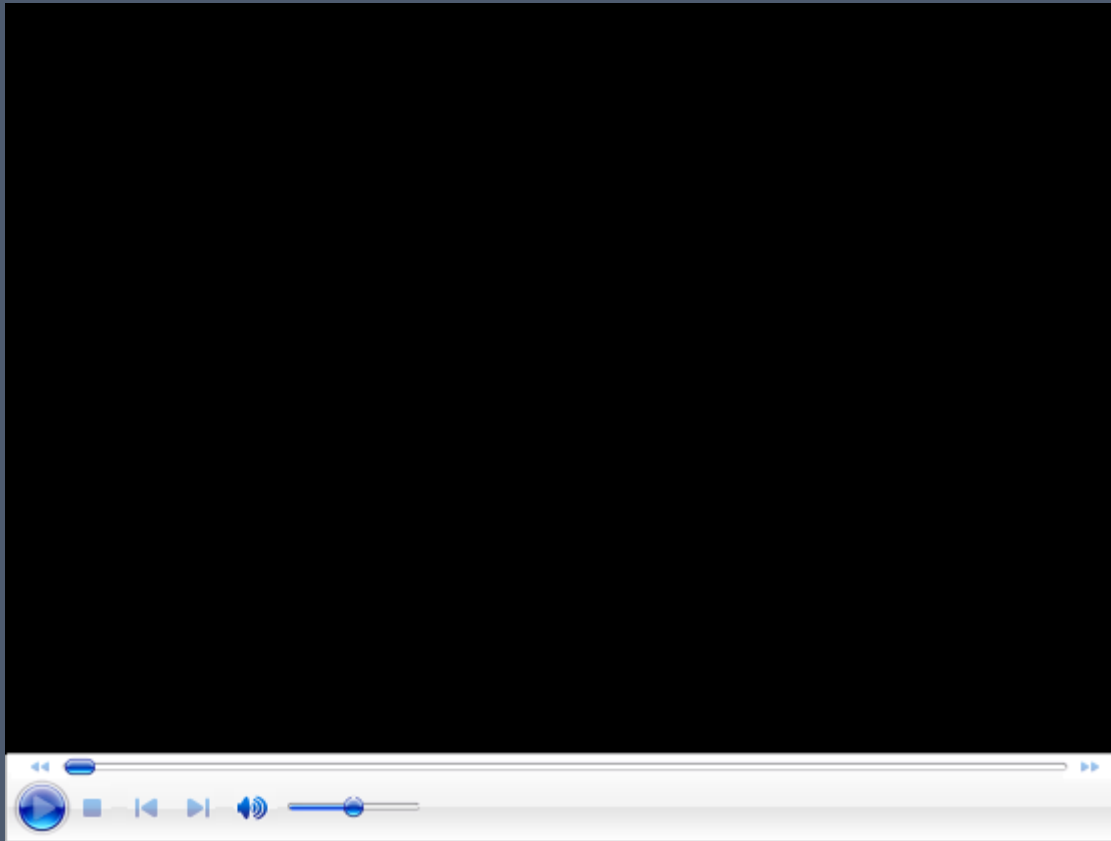
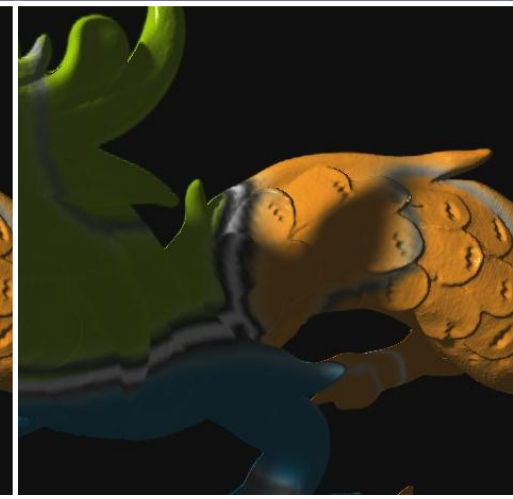
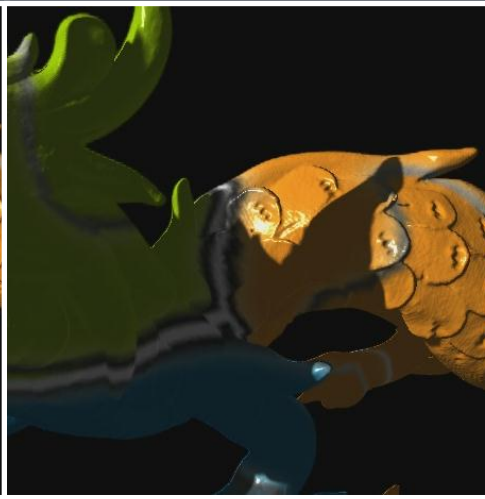
- Going further with 3D MipMapping
 - Full pre-integrated versions of objects
- Idea: Implements blurry effects very efficiently
 - Without multi-sampling
 - Tuning the mipmap level
- Soft shadows
- Depth of field
- Glossy reflections...

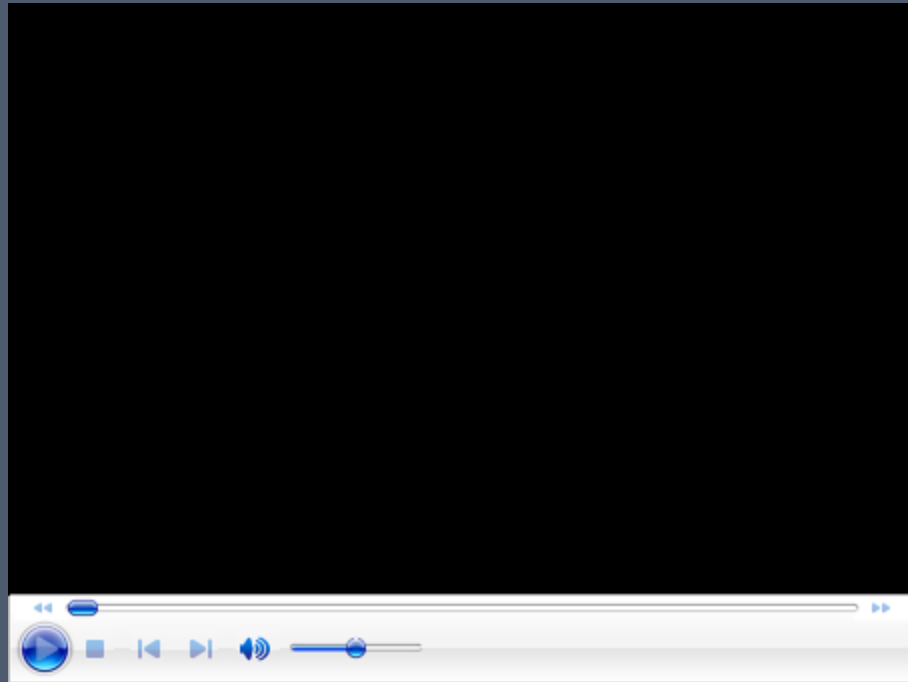
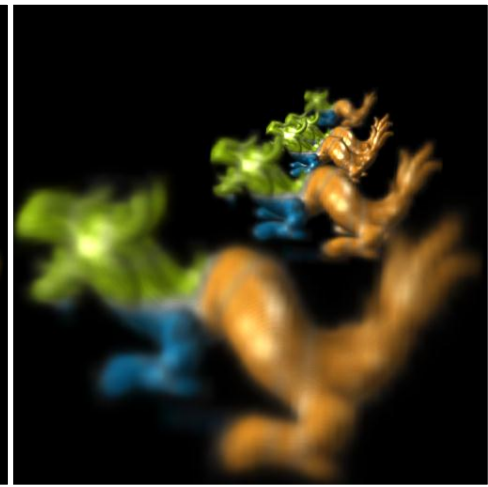
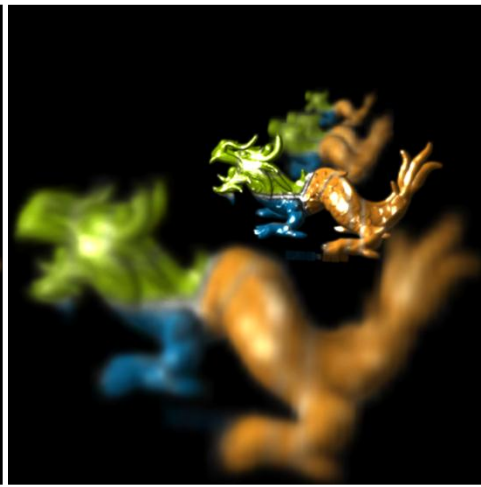
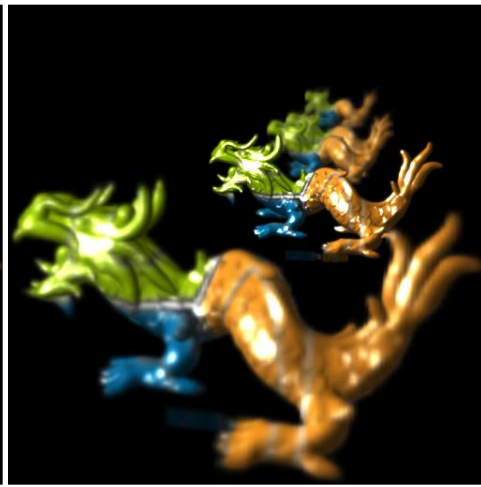


Soft shadows

- Secondary rays
 - When ray hit object surface
- MipMap level chosen to approximate light source cone
 - Resulting integrated opacity
- Fully compatible with the cache

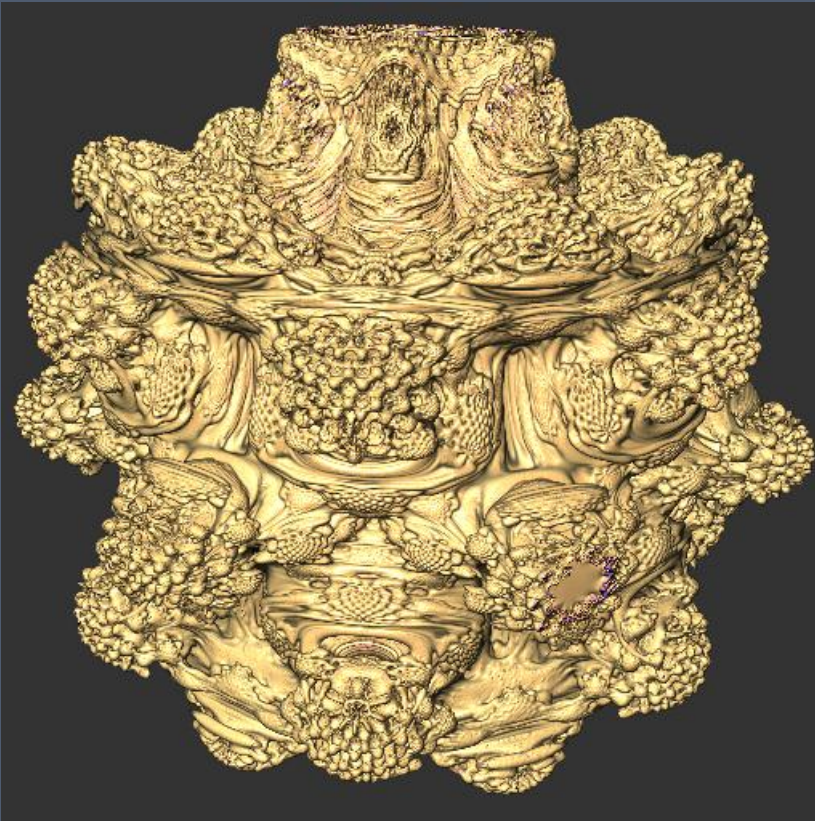




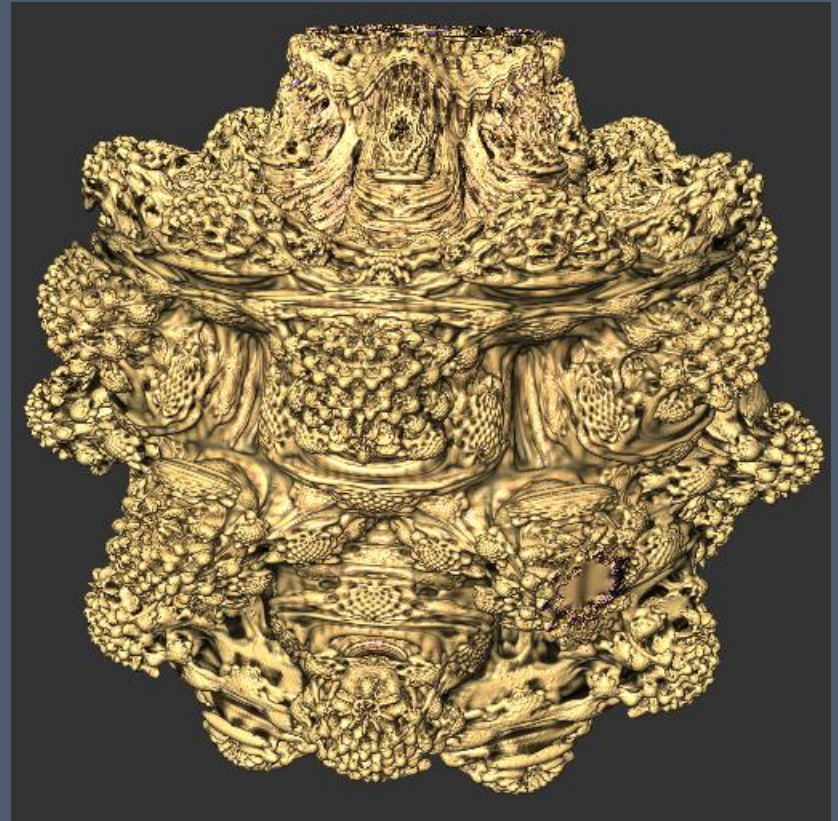


Ambient occlusion

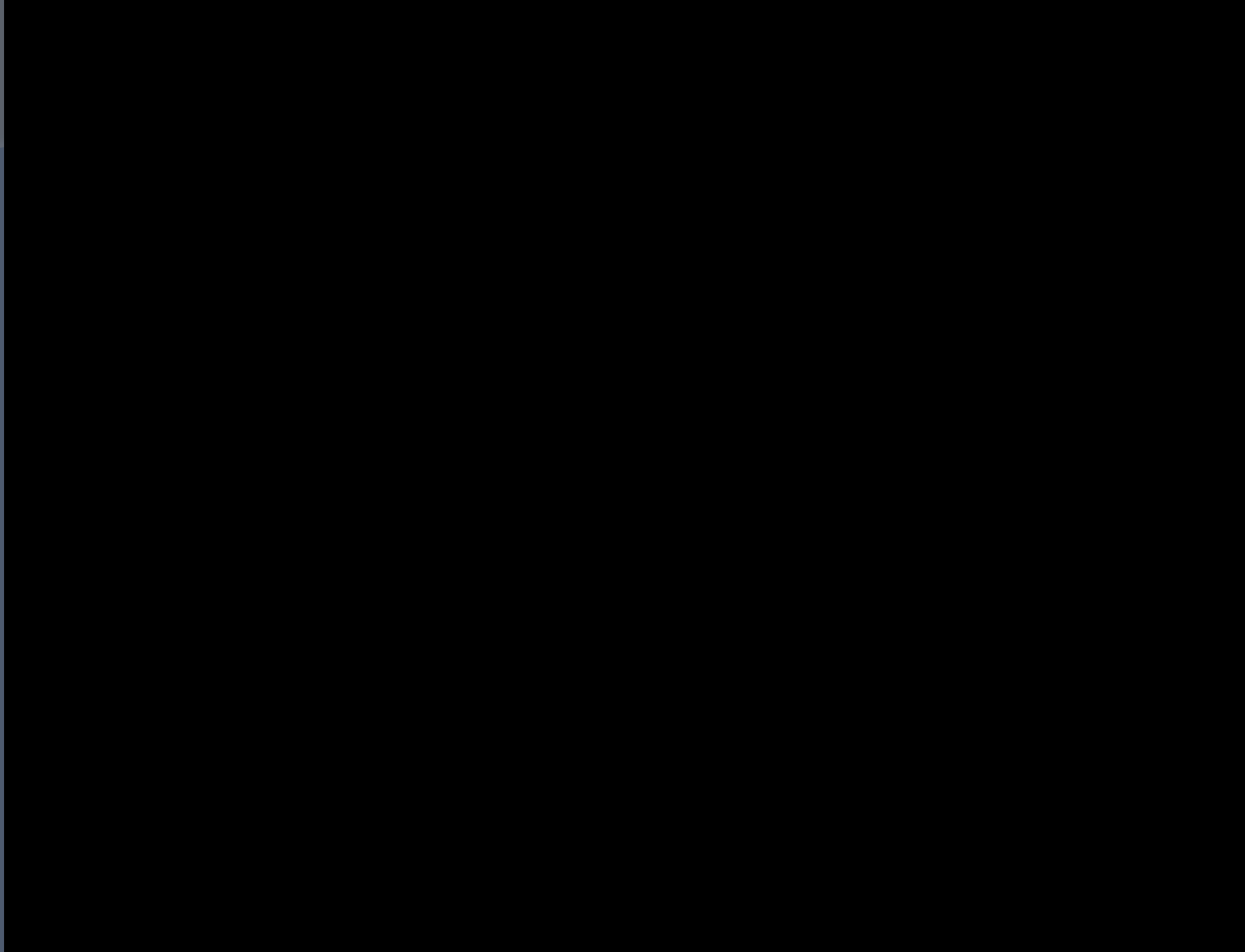
- Uses one filtered sample
 - Covers the surrounding region



Without AO



With AO

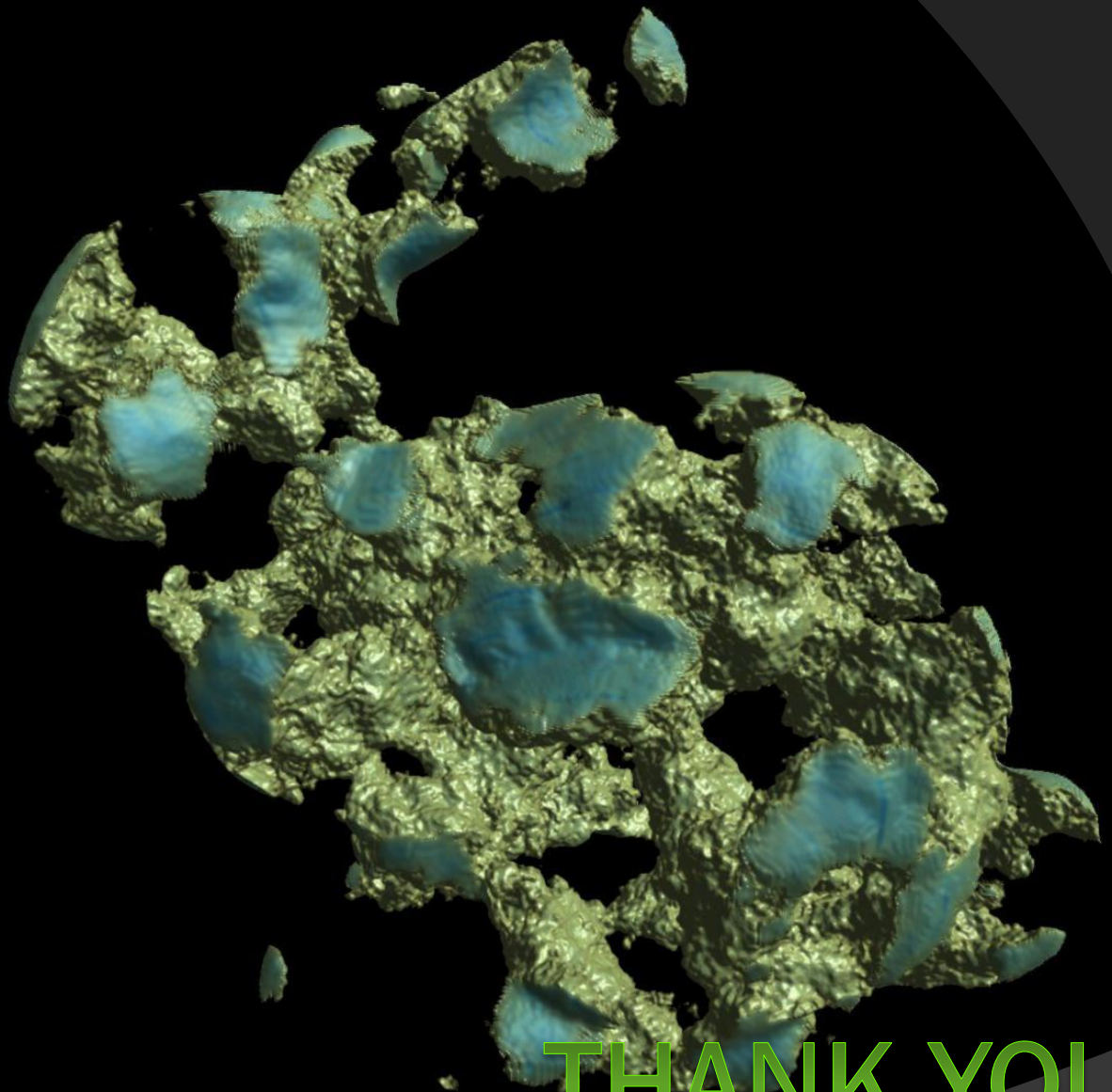


Future work direction

- ◎ Animation
 - Yes, this can be efficiently animated !
 - Volume deformation (skinning)
- ◎ Improved visibility integration
- ◎ Filtering
 - Shading/Normals
 - Isotropic pre-integration
 - Two walls problems

Many thanks go to ...

- ⦿ Digisens Corporation
- ⦿ Rhone-Alpes Explora'doc program
- ⦿ Cluster of Excellence on Multimodal Computing and Interaction (M2CI)
- ⦿ 3D-Coat and Rick Sarasin
- ⦿ Erklarbar



**THANK YOU
FOR YOUR ATTENTION**