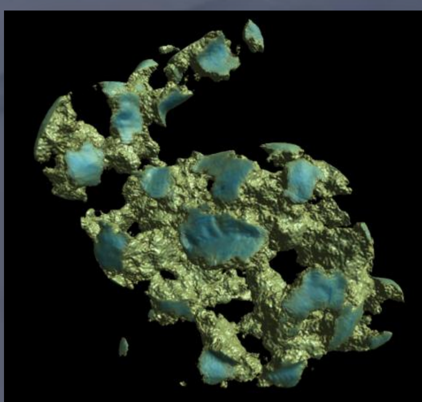
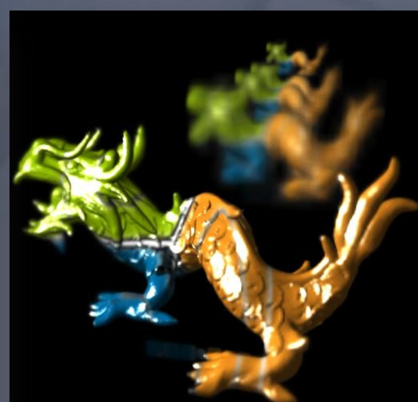
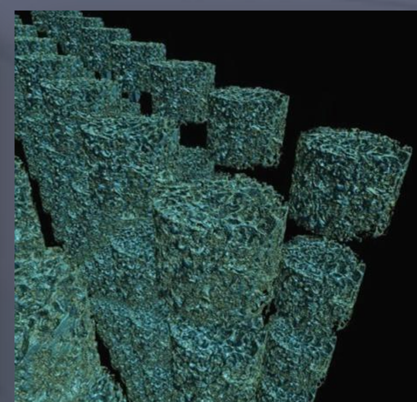
SIGGRAPH2009 NEW ORLEANS

Beyond Triangles

GigaVoxels Effects In Video Games

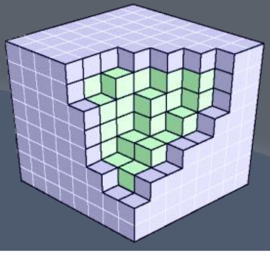





Cyril Crassin, Fabrice Neyret,
INRIA Rhône-Alpes & Grenoble Univ.

Sylvain Lefebvre, *INRIA Sophia-Antipolis*
Elmar Eisemann, *Saarland Univ./MPI*
Miguel Sainz *NVIDIA Corporation*

A (very) brief history of voxels

- Rings a bell?



Voxel grid illustration courtesy of “Real-Time Volume Graphics”

Outcast (Appeal Software)

Comanche (Novalogic)

Voxel Engines in Special effects

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

The Day After Tomorrow, Digital Domain

XXX, Digital Domain

Lord of the Rings, Digital Domain

Voxels in video games ?

- Renewed interest
  - Jon Olick, Siggraph 08
  - John Carmack

[Olick08]
Jon Olick,
John Carmack

Why bother with voxels?

- Exploding number of triangles
  - Sub-pixel triangles not GPU-friendly (might improve but not yet REYES pipeline)
- Filtering remains an issue
  - Multi-sampling expensive
  - Geometric LOD ill-defined
- Clouds, smoke, fluids, etc.
  - Participating media?

The Mummy 3, Digital Domain/Rhythm&Hues

Voxels

- Natural for complex geometries
  - LOD defined
  - “Unique Geometry” (no additional authoring)
- Structured data
  - Convenient to traverse
- But:
  - Memory is a key issue!
    - E.g. 2048 ^ 3 x RGBA = **32 GB!!!**
    - Transfer CPU ⇔ GPU expensive
  - No fast renderer available

GigaVoxels

- I3D2009 paper [CNLE09]
  - Unified geometry & volumetric phenomena
- Full pipeline to render **infinite resolution** voxel objects/scenes


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

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

Figure 1: *Images show volume data that consist of billions of voxels rendered with our dynamic sparse octree approach. Our algorithm achieves real-time to interactive rates on volumes exceeding GPU memory capacity by far, tanks to efficient streaming and ray-casting solutions. Basically, the volume is only used at the resolution that is needed to produce the final image. Besides the gain in memory and speed, our rendering provides is inherently anti-aliased.*

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-casting rendering algorithm. One 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 valuable alternative as a rendering primitive for real-time applications. In this spirit, we allow a quality/performance trade-off and exploit temporal coherence. We also introduce a mipmapping-like process that allows for an increased display rate 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 in several scenarios, like the exploration of a 3D scan (8192^3 resolution), of hypertextured meshes (16384^3 virtual resolution), or of a fractal (theoretically infinite resolution). All examples are rendered on current generation hardware at 20-90 fps and respect the limited GPU memory budget.

This is the author's 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, Cinesite*, or *Rhythm 'n Hues* now massively rely on so-called *voxel engines* [Kis98, BTG03, Kap03, 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 Mummy 3*). Clouds, smoke, foam, and even non-fuzzy 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 previsualization.

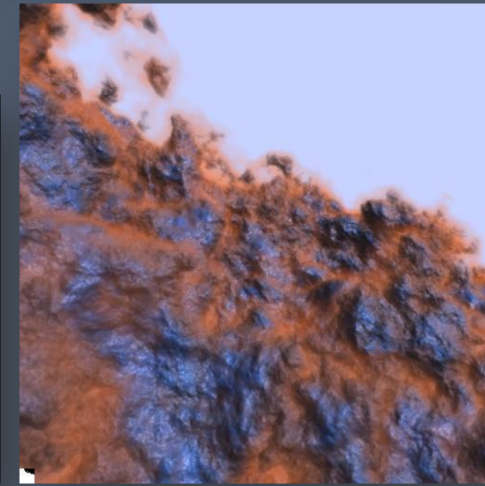
The significant advantage of voxels is the richness of this representation and the very regular structure which makes it easy to manipulate. In particular, filtering operations are well-defined, making it a good candidate to address aliasing issues that are hard to deal with for triangulated models.

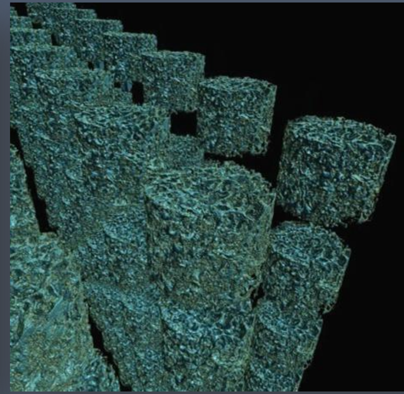
This is one of the reasons voxel data is often used to represent pseudo-surfaces, which is an interface that resembles a surface at a certain distance, but appears complex (non-heightfield, non-connected, or non-opaque) at close view. An example is the foliage of a tree that can be well approximated with volumetric data [RMMD04], but this observation holds for complex surfaces in general. Thus volumetric rendering is also a graceful way of dealing with the level of detail problem.

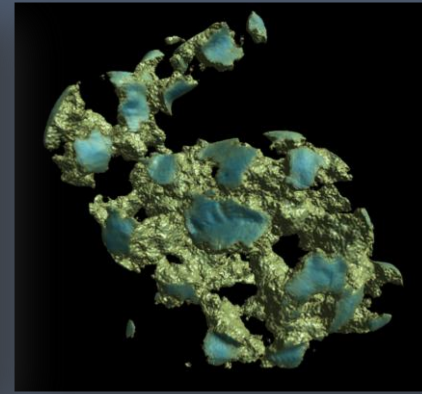
In this paper, we show that the current hardware generation is ready to achieve high-quality massive volume renderings at interactive to real-time rates. Benefits such as filtering, occlusion culling, and procedural data creation, as well as level-of-detail mechanisms are integrated in an efficient GPU voxel engine. This enables us to obtain some of the visual quality that was previously reserved for movie productions and enables the technique to be used to previsualize 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 *intelligent*

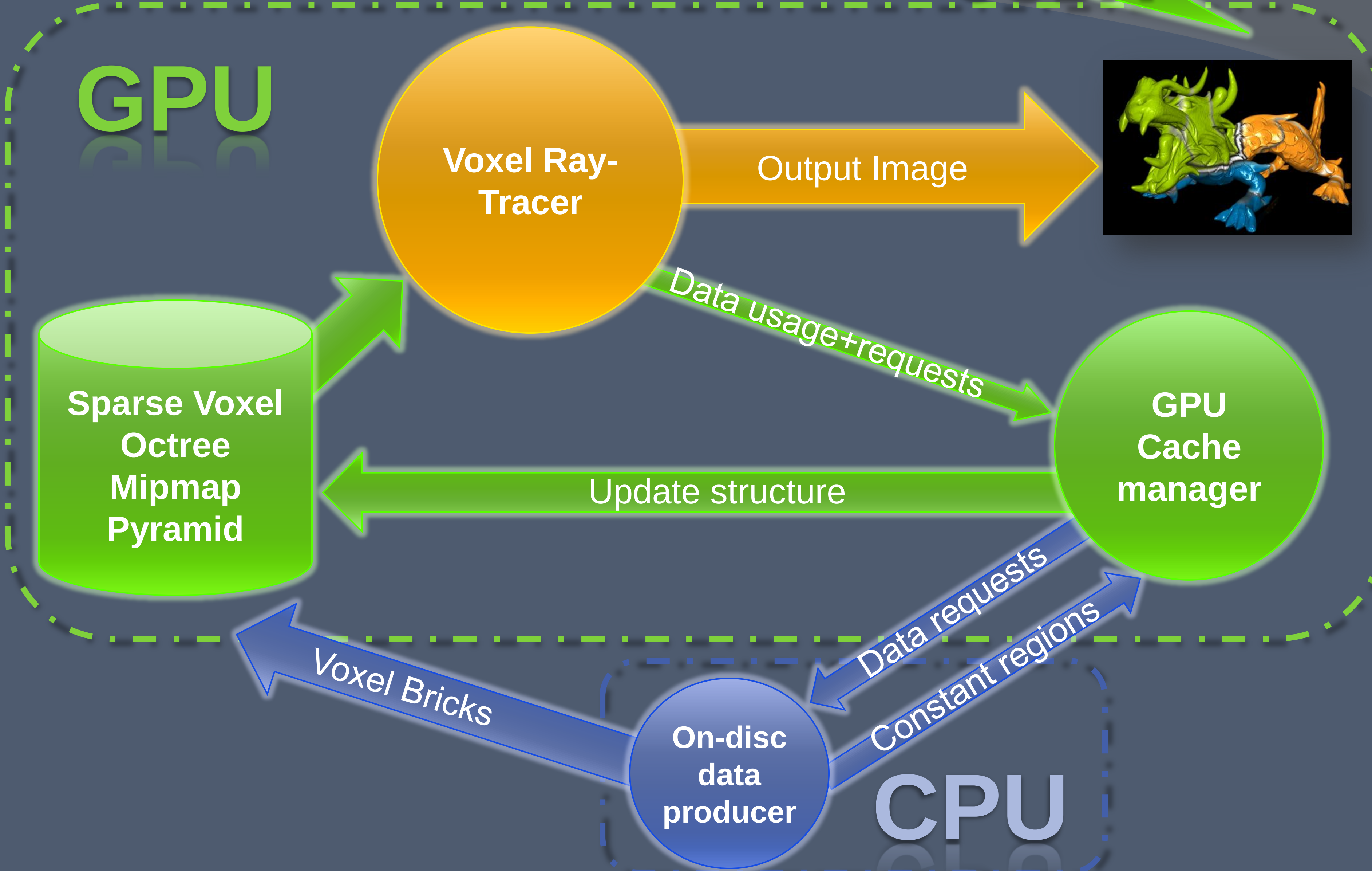


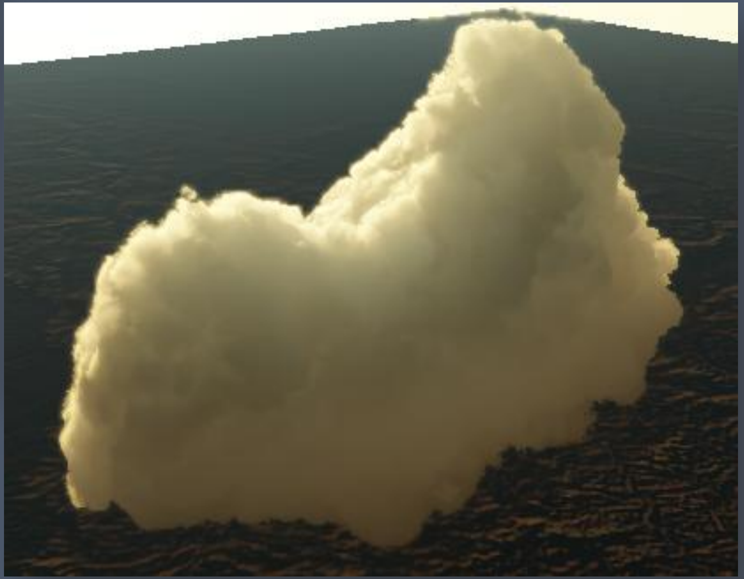




GigaVoxels pipeline

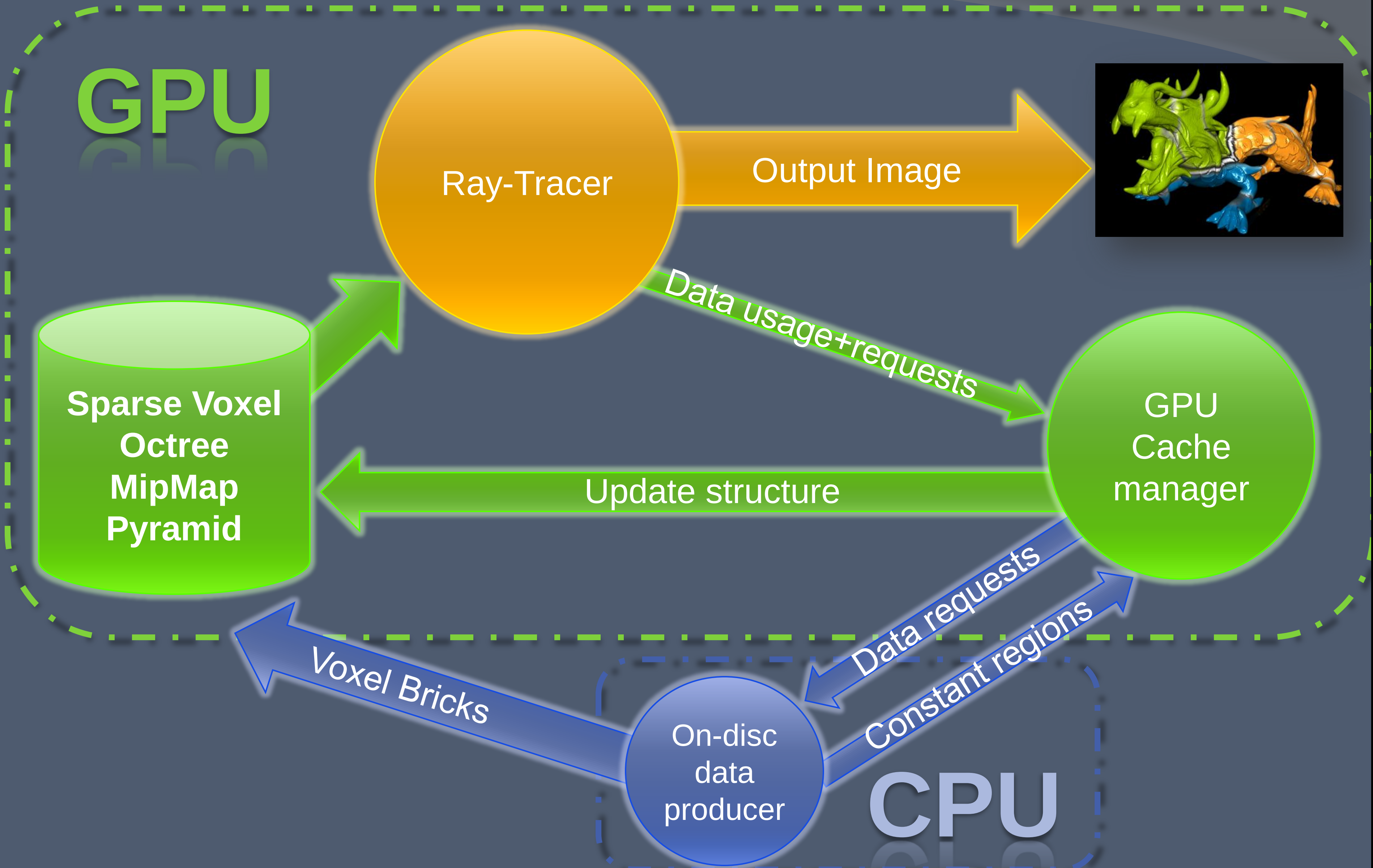
Now implemented with **CUDA**





[BNMBC08]

GigaVoxels Data Structure



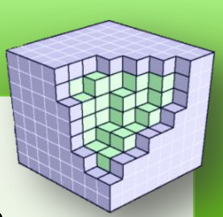
Sparse Voxel MipMap Pyramid

Data structure

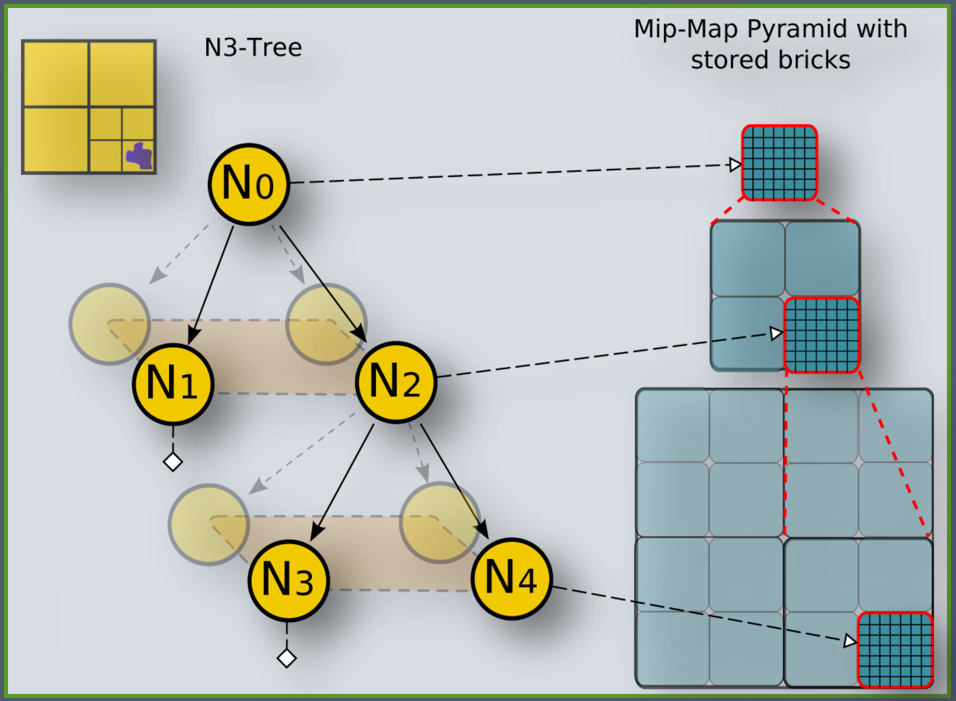
Generalized Octree

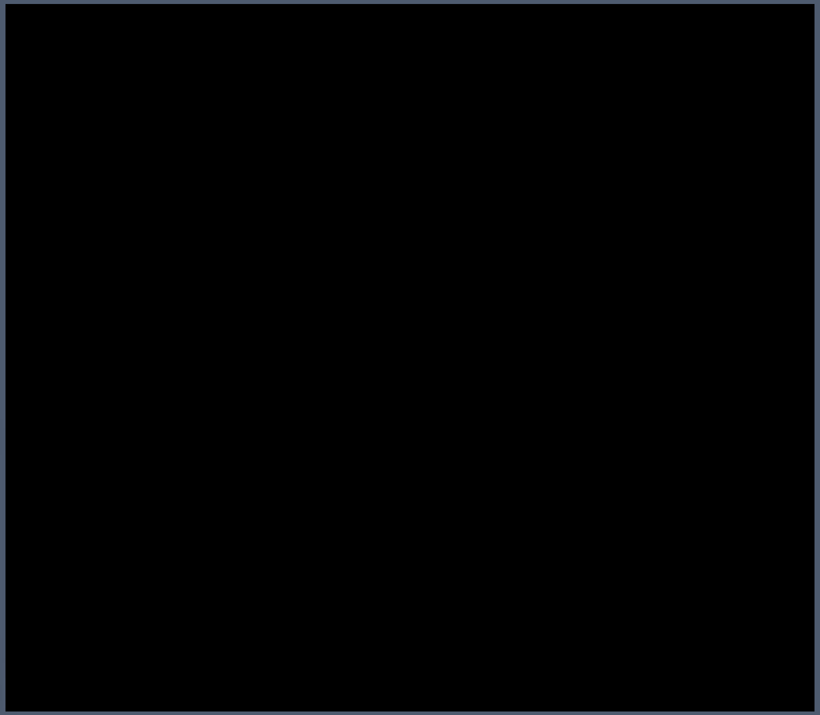
- Empty space compaction

Bricks of voxels



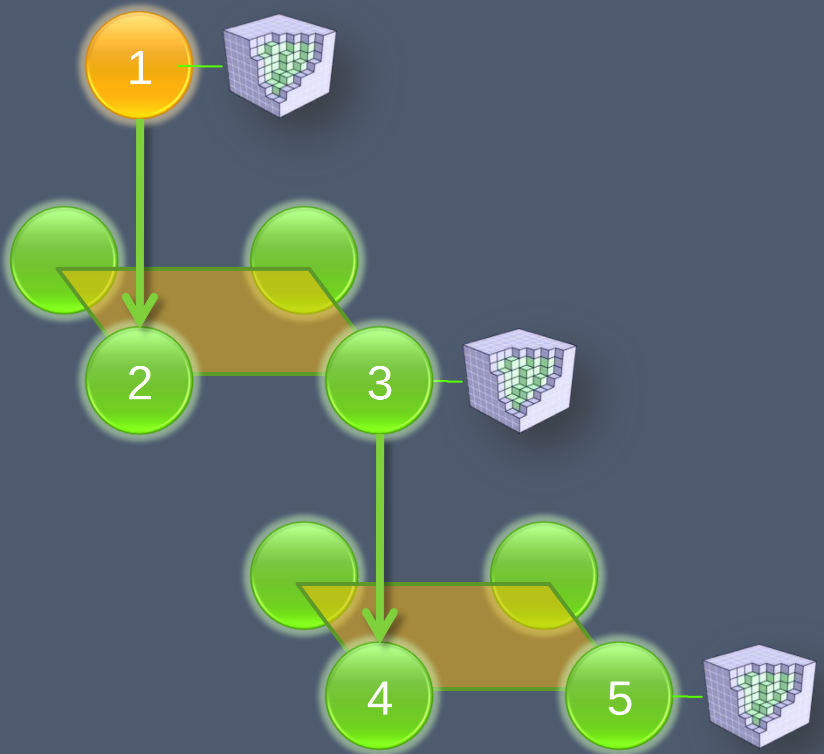
- Linked by octree nodes
- Store opacity, color, normal

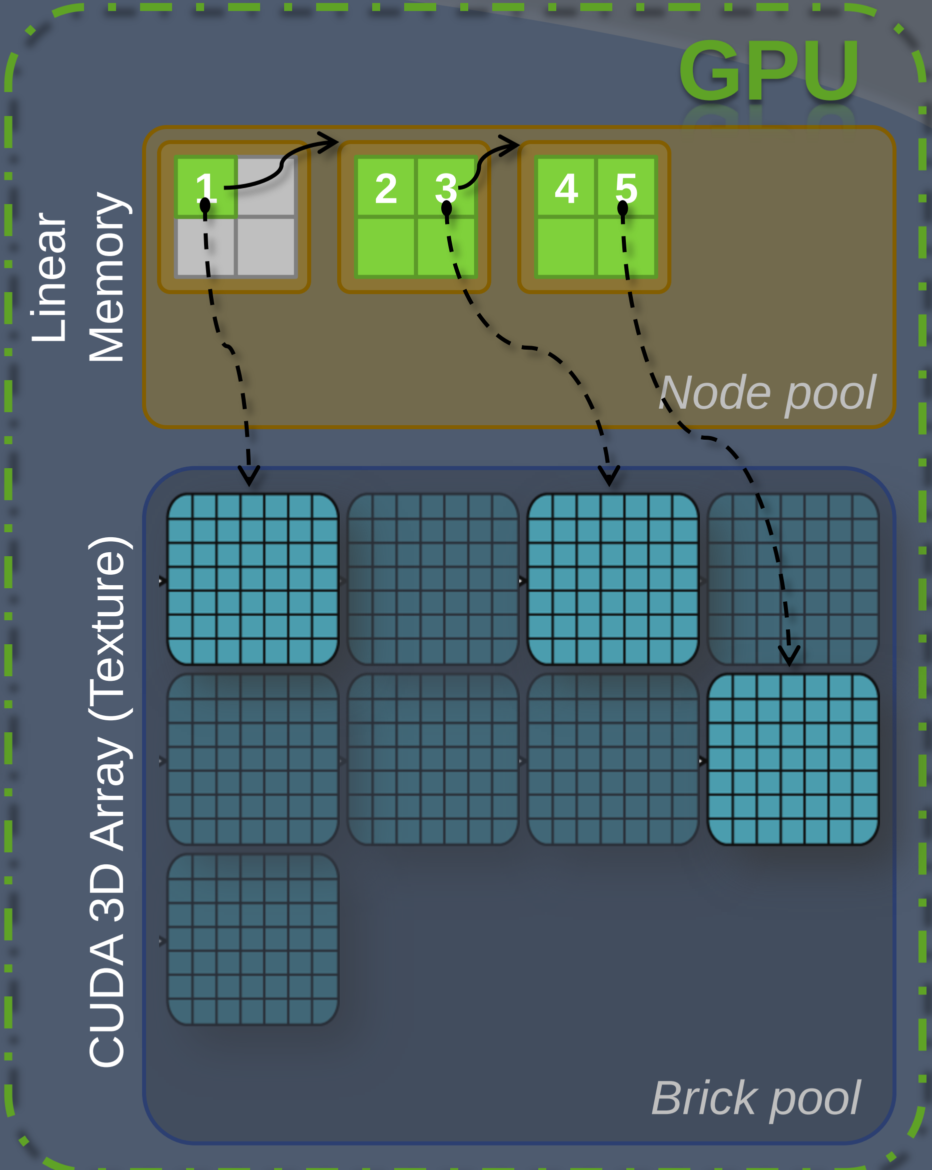




Tower model courtesy of Erklaerbar, made with 3DCoat

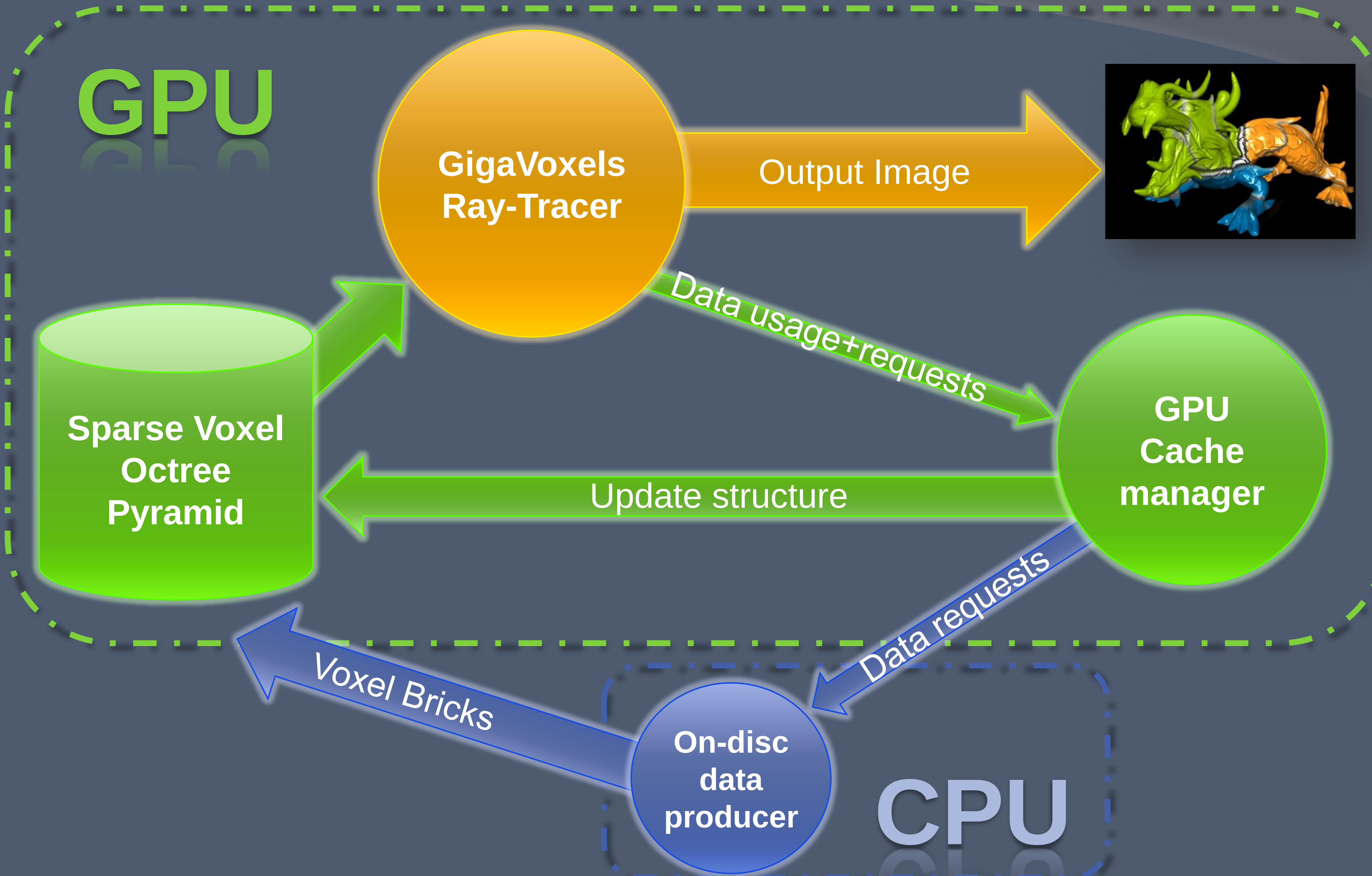
Octree of Voxel Bricks





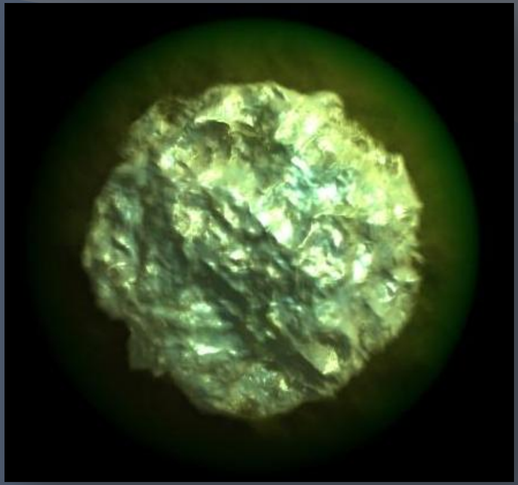
- One child pointer
  - Compact structure
  - Cache efficient

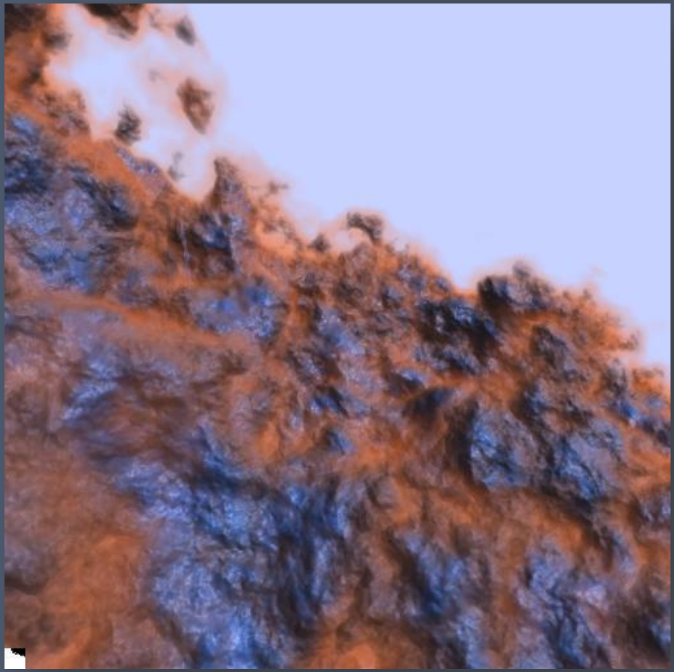
GigaVoxels Rendering



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



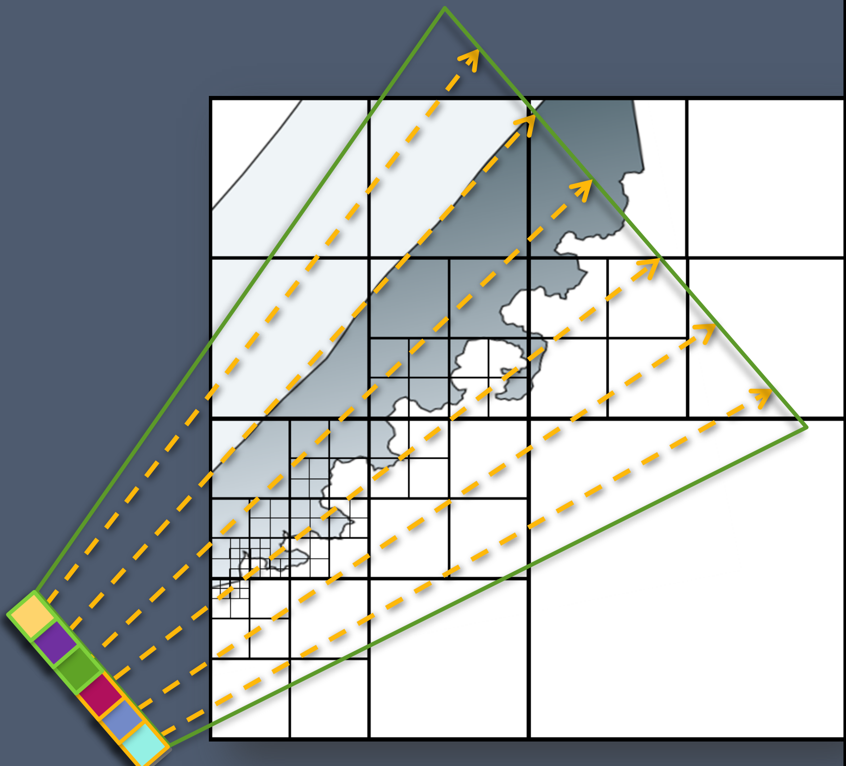


Hierarchical Volume Ray-Casting

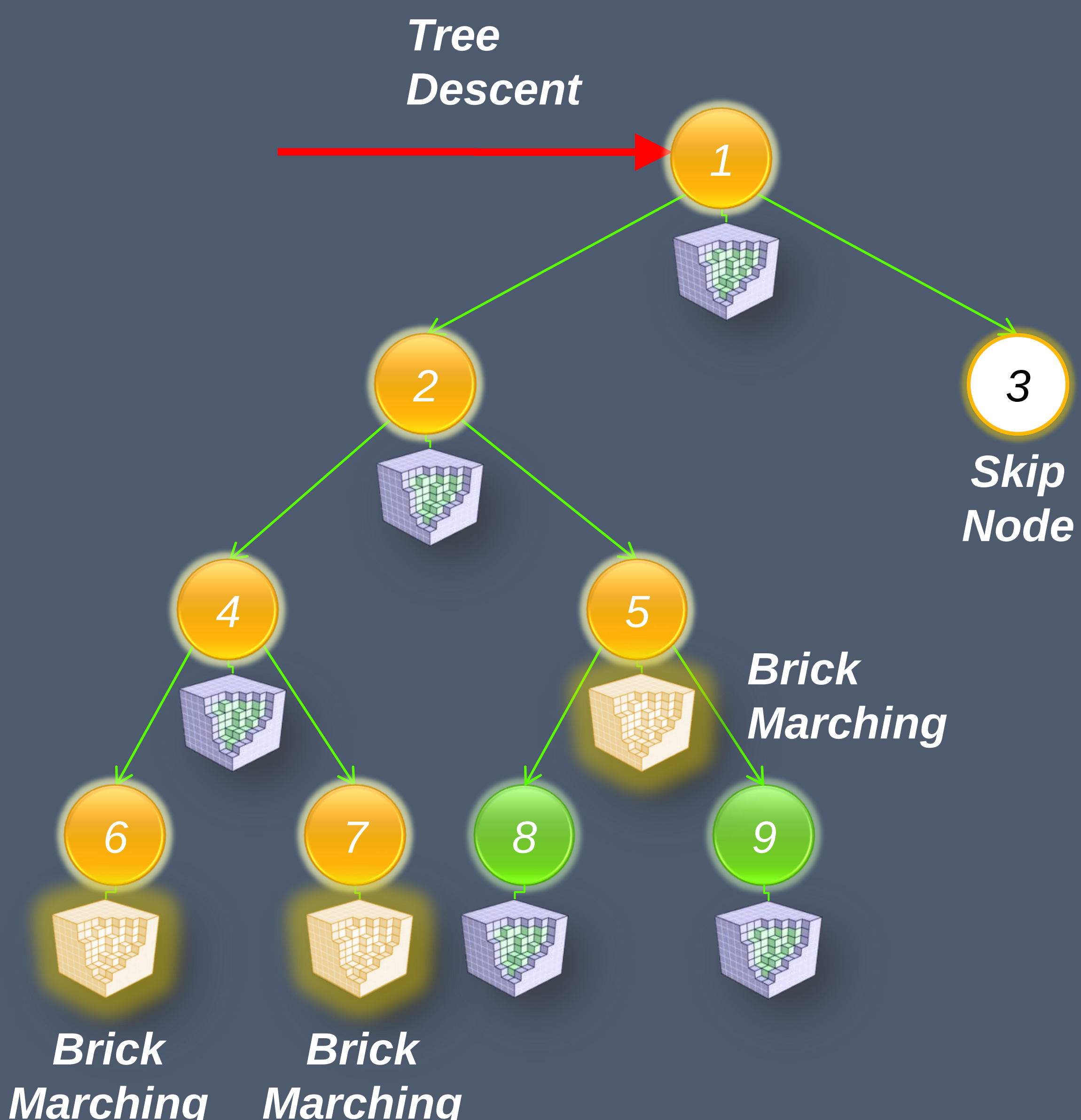
- Volume ray-casting

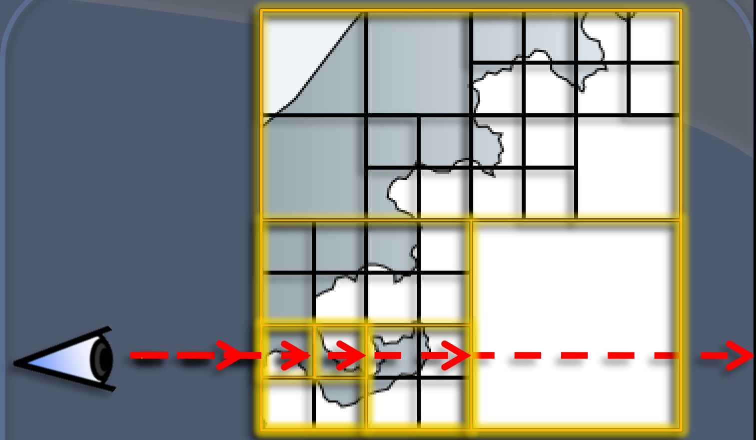
  [Sch05, CB04, LHN05a, Olick08, GMAG08, CNLE09]

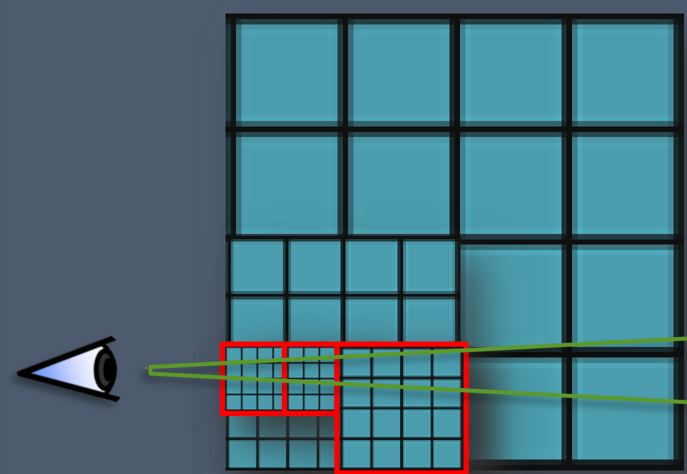
- Big CUDA kernel
  - One thread per ray
  - KD-restart algorithm
  - **Ray-driven LOD**



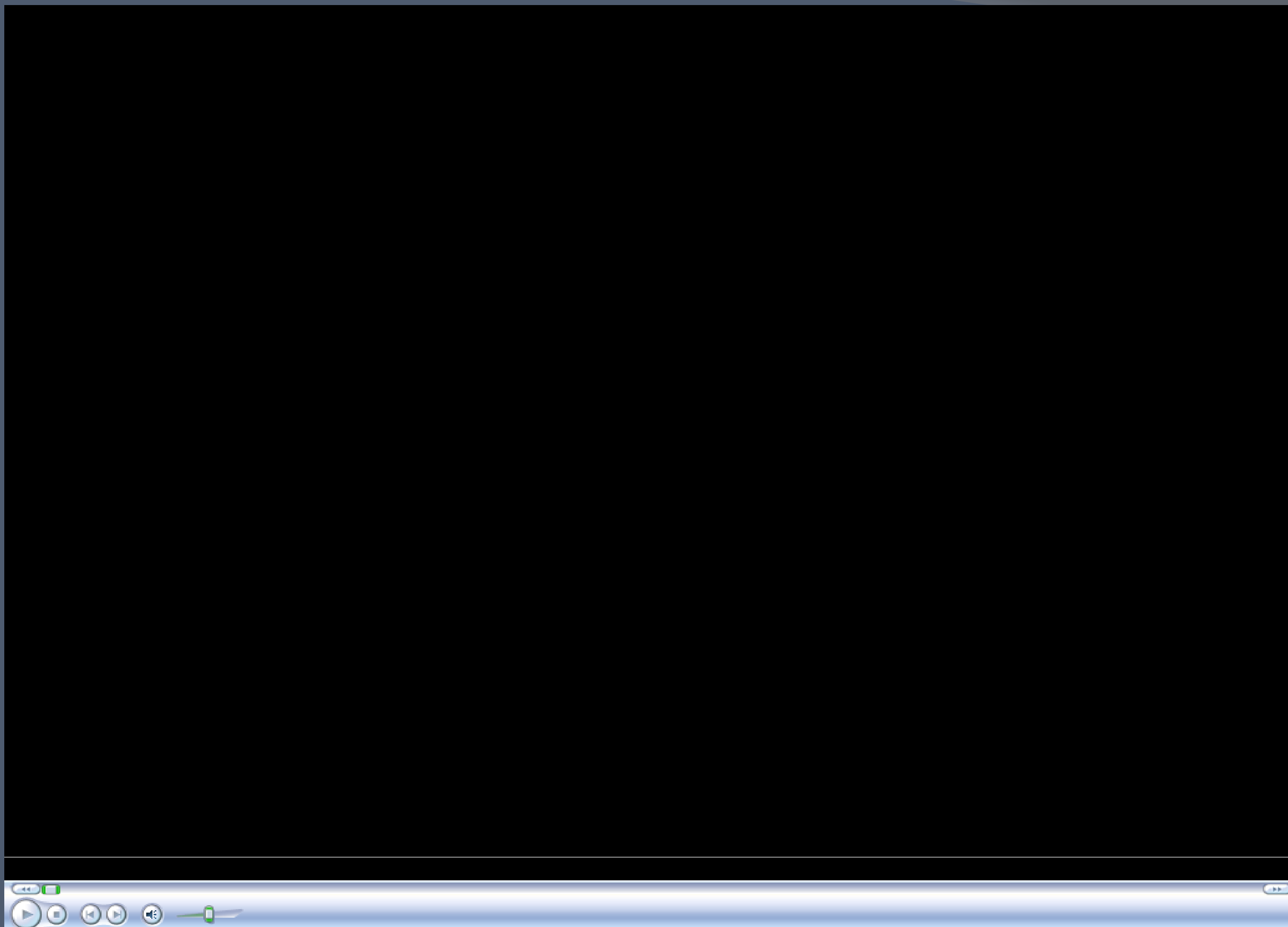
Volume Ray-Casting




Ray traversal


Per-ray LOD
evaluation

Rendering costs



Volume MipMapping mechanism

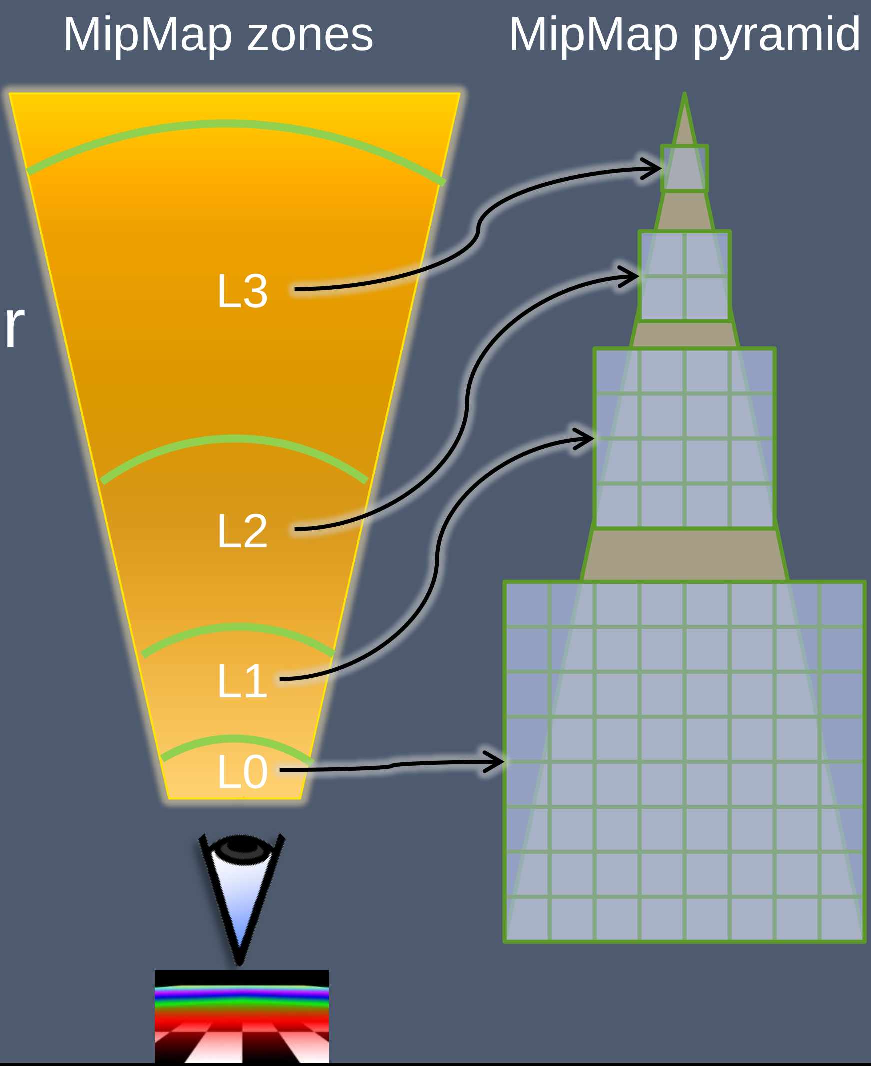
Problem: LOD uses discrete downsampled levels

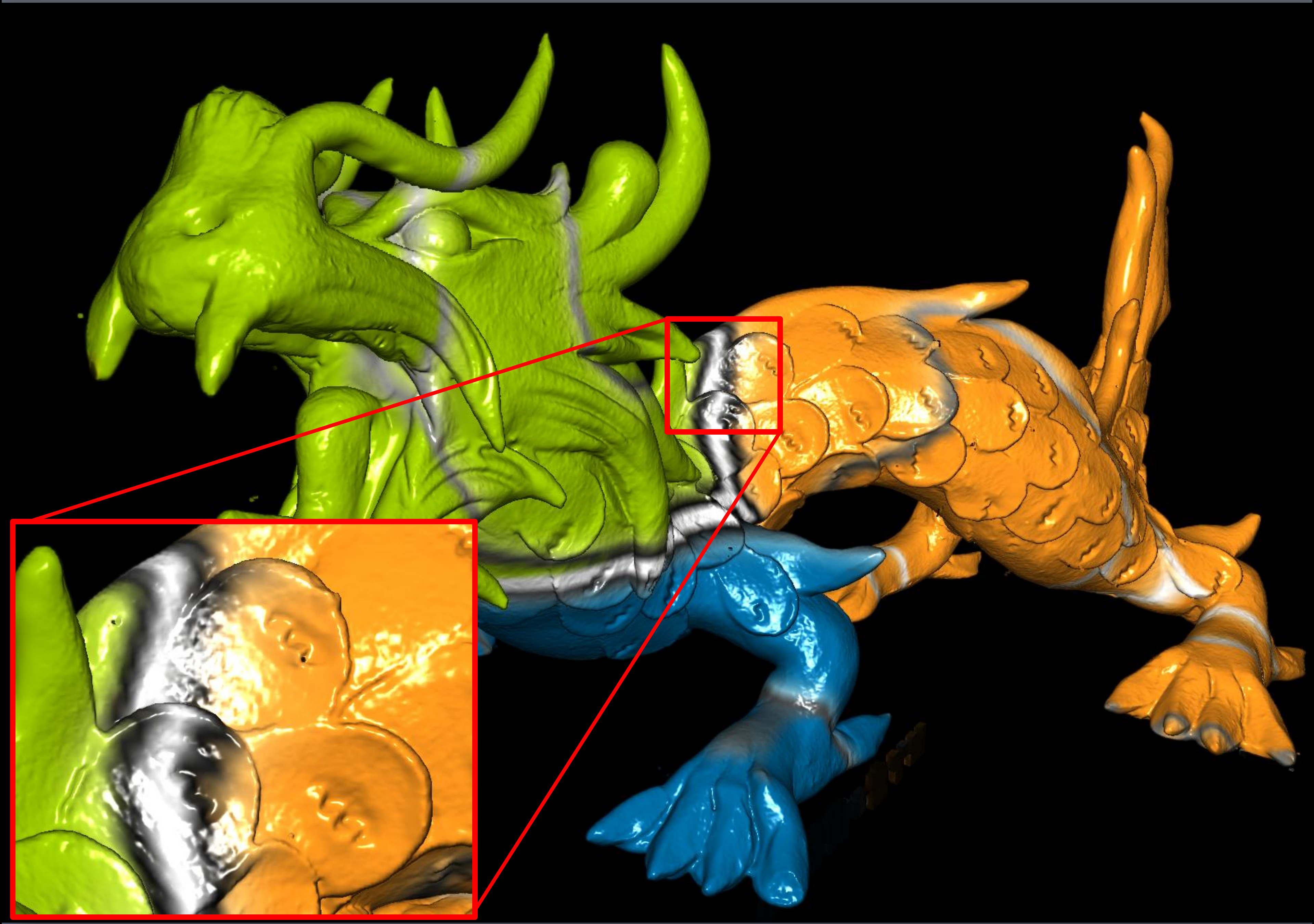
- Popping + Aliasing
- Same as bilinear only for 2D textures

➔ Quadrilinear filtering

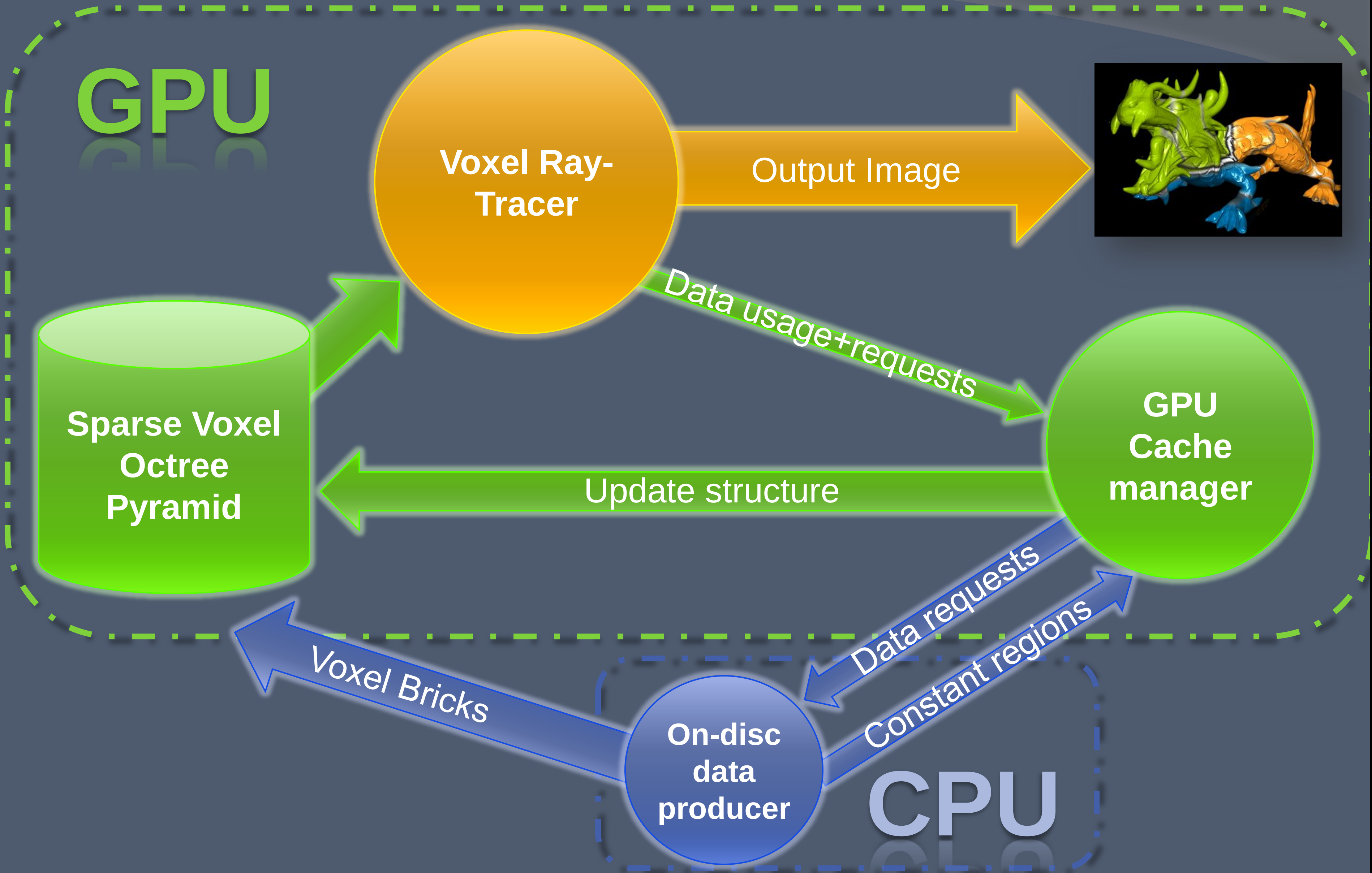
◉ Geometry is texture ☺

- No need of multi-sampling (eg. MSAA)



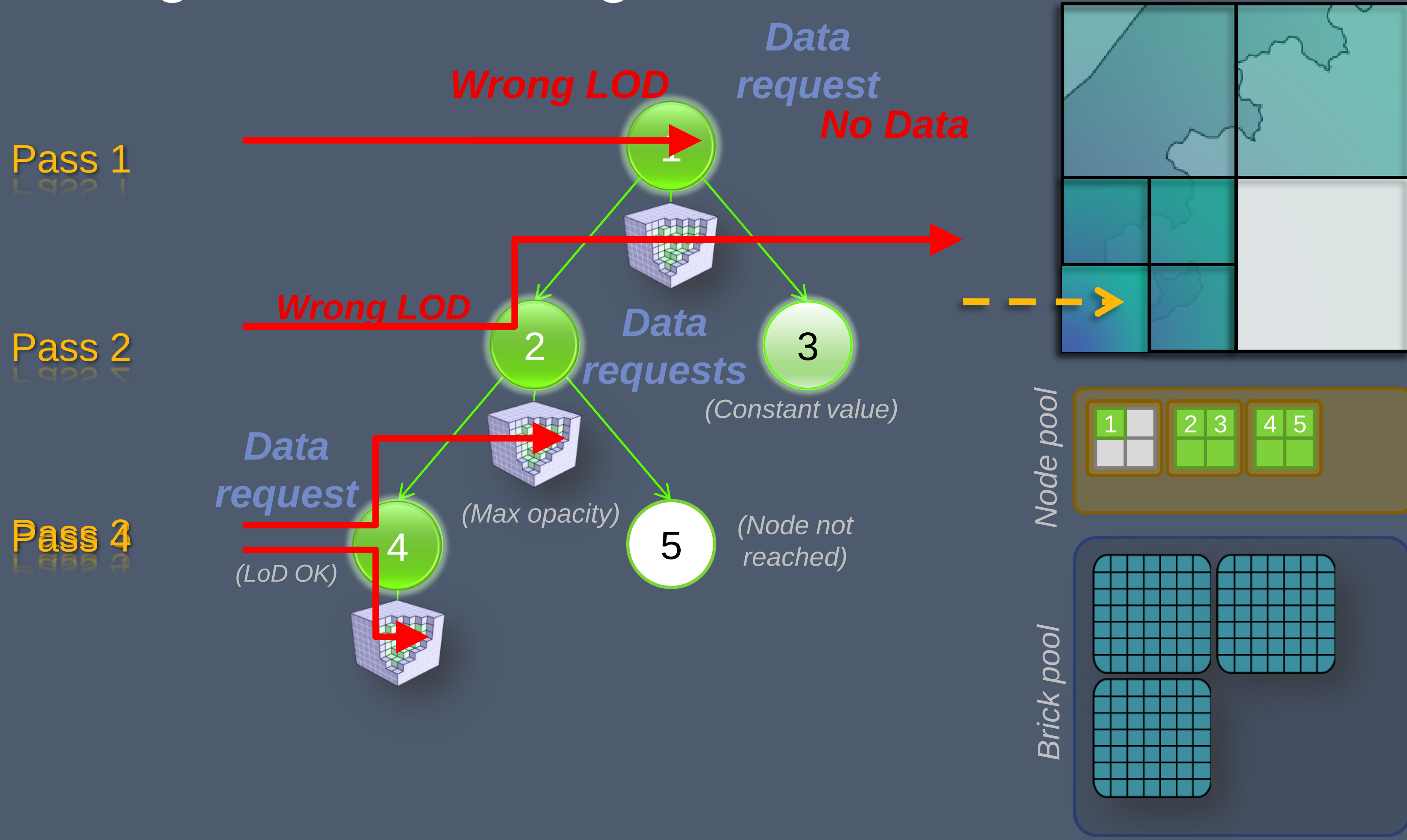


GigaVoxels Data Management



Incremental octree update

- Progressive loading



Ray-based visibility & queries

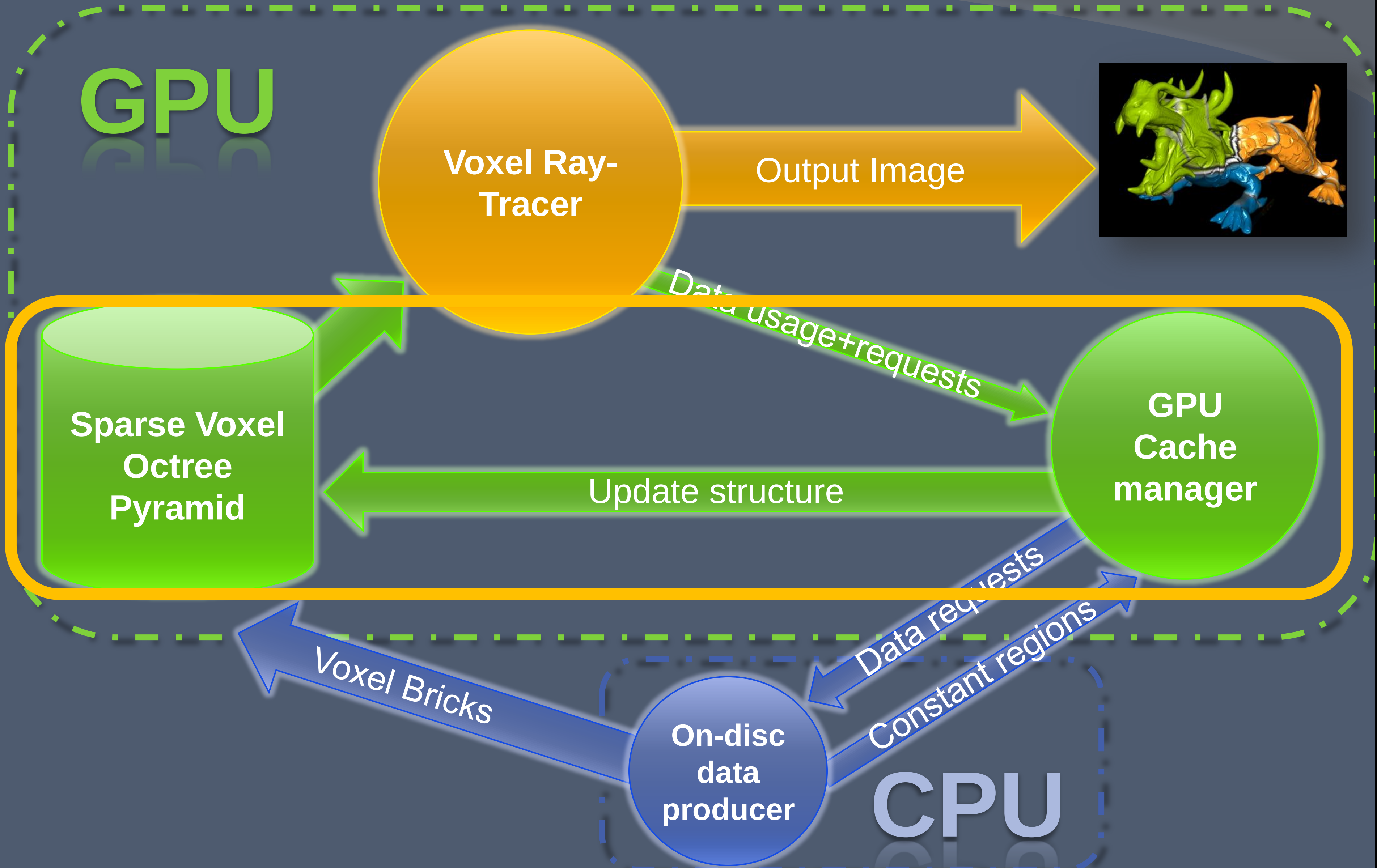
Zero CPU intervention

- Per ray frustum and visibility culling

On-chip structure management

- Subdivision requests
  - LOD adaptation
- Cache management
  - Remove CPU synchronizations

GigaVoxels Data Management



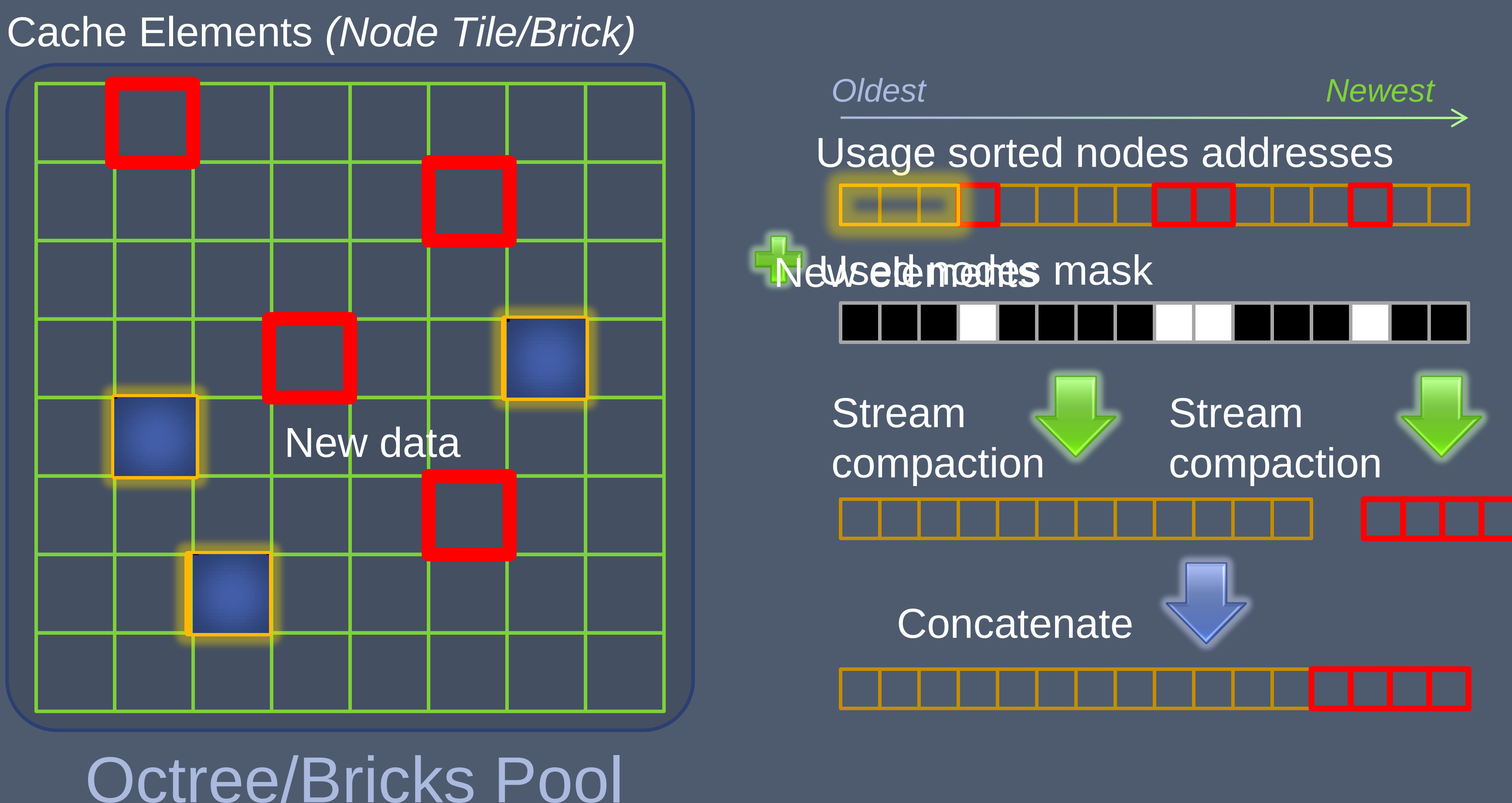
SVMP cache

- Two caches on the GPU
  - Bricks
  - But also tree

→ No maximal tree size

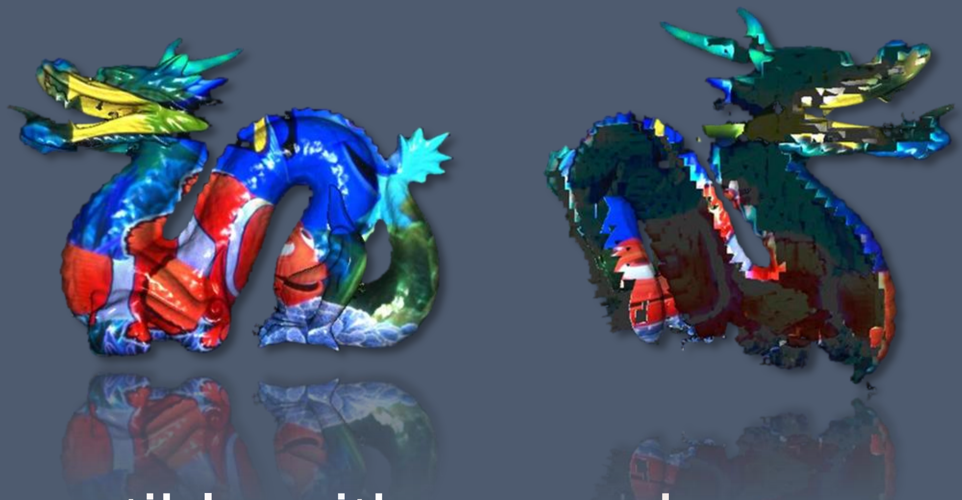
SVMP caches

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



Just-in-Time Visibility Detection

- Minimum amount of data is loaded



- Fully compatible with secondary rays and exotic rays paths
  - Reflections, refractions, shadows, curved rays, …

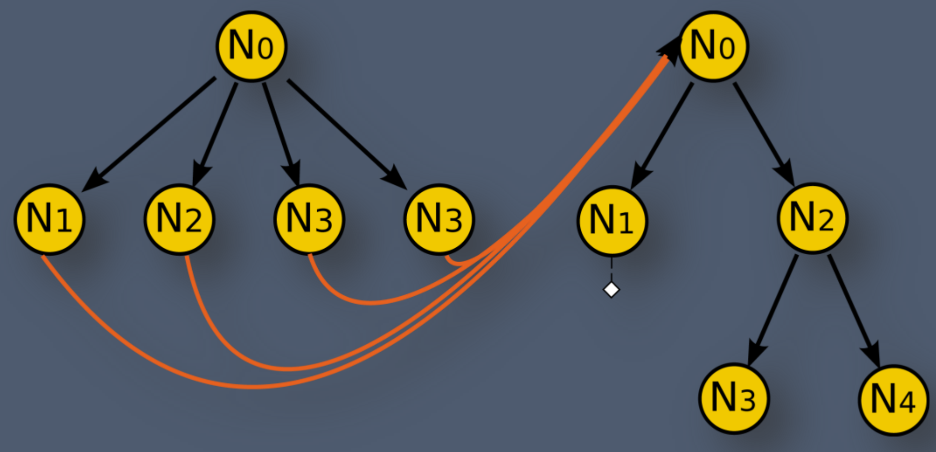
Voxel sculpting

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

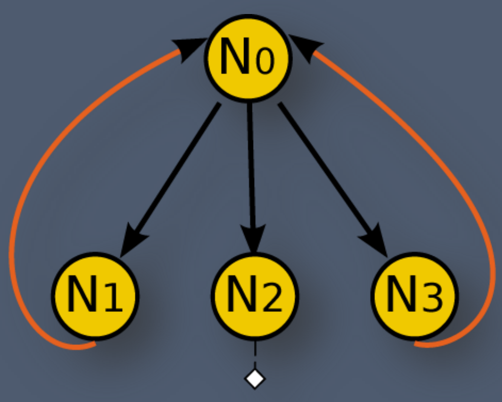
GIGAVOXELS IN VIDEO GAMES

Voxel data synthesis

- Instantiation

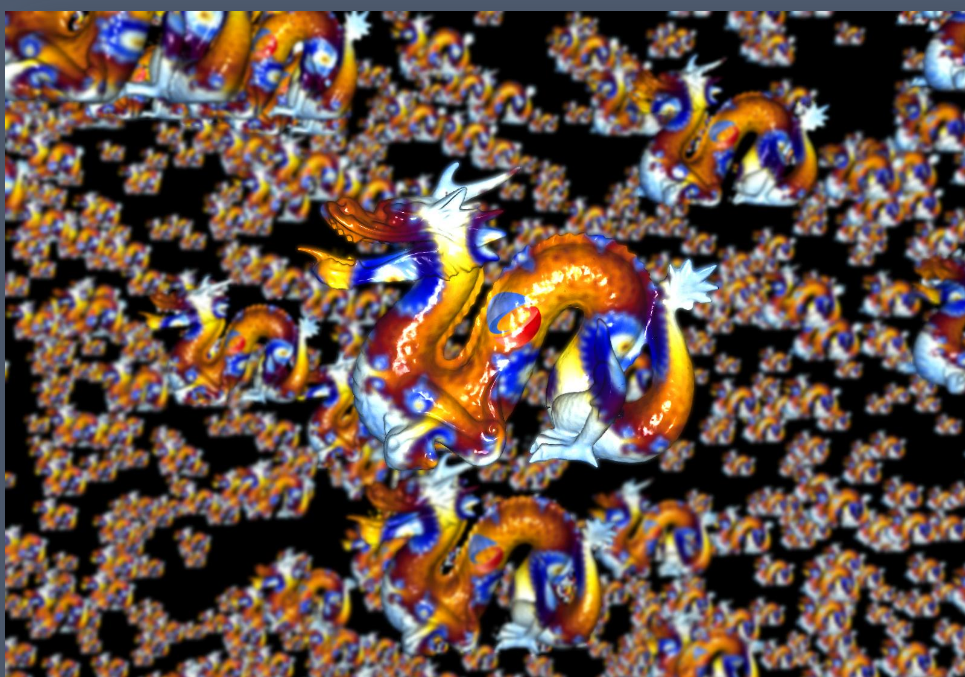


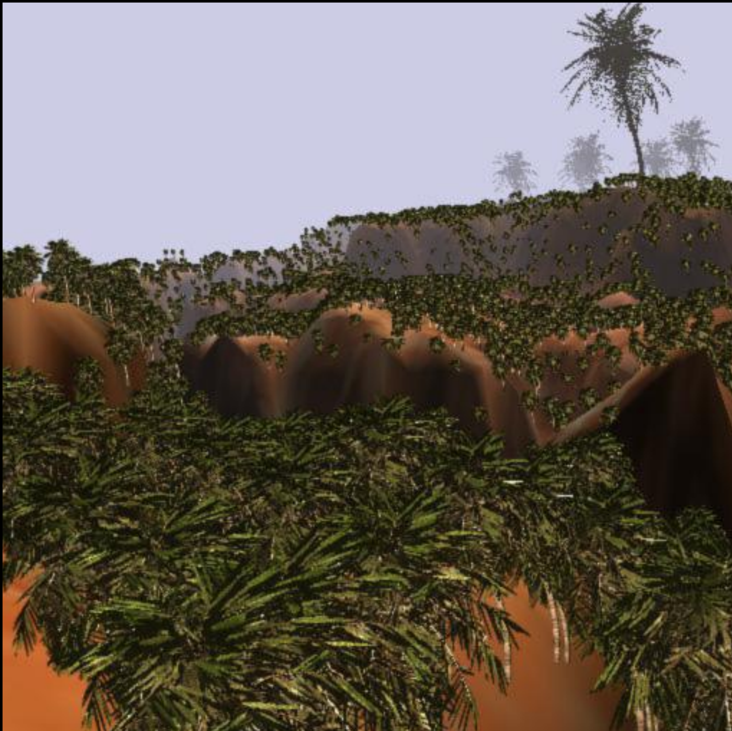
- Recursivity
  - Infinite details

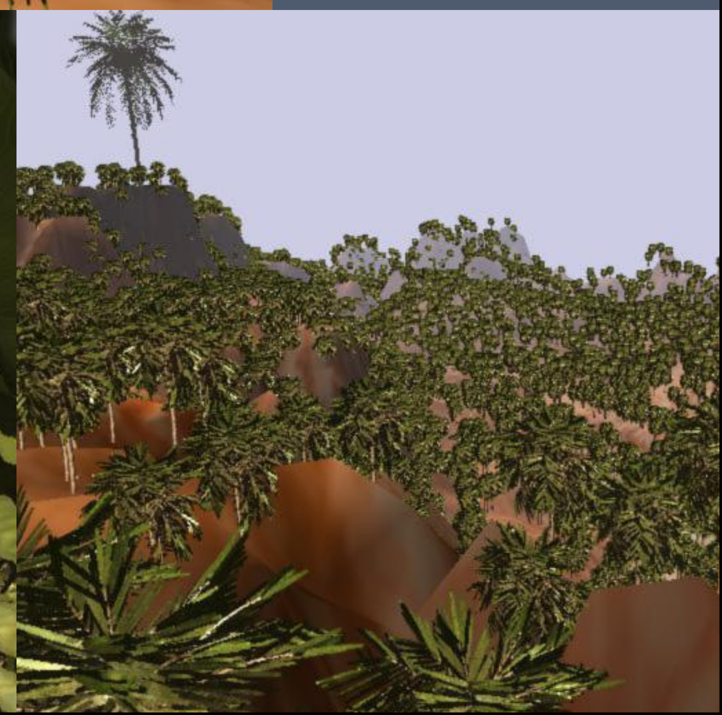


Free voxel objects instancing

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





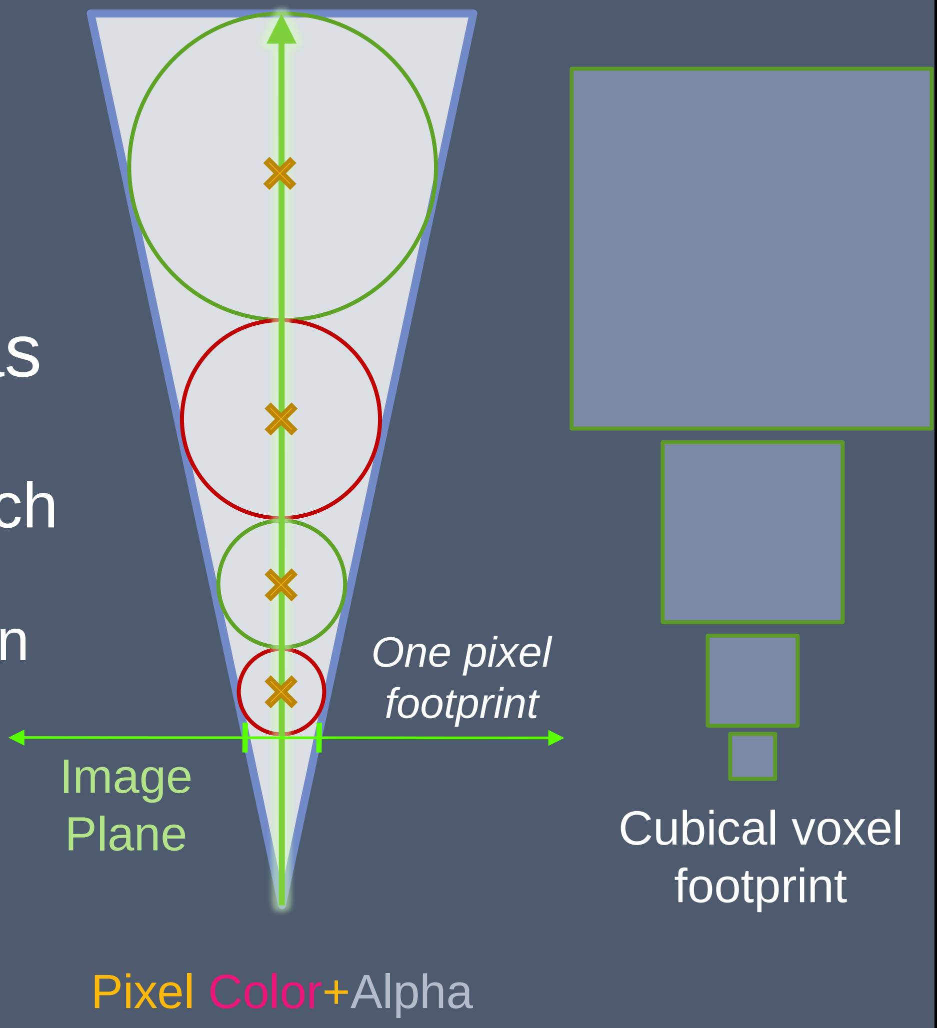


Cool Blurry Effects

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

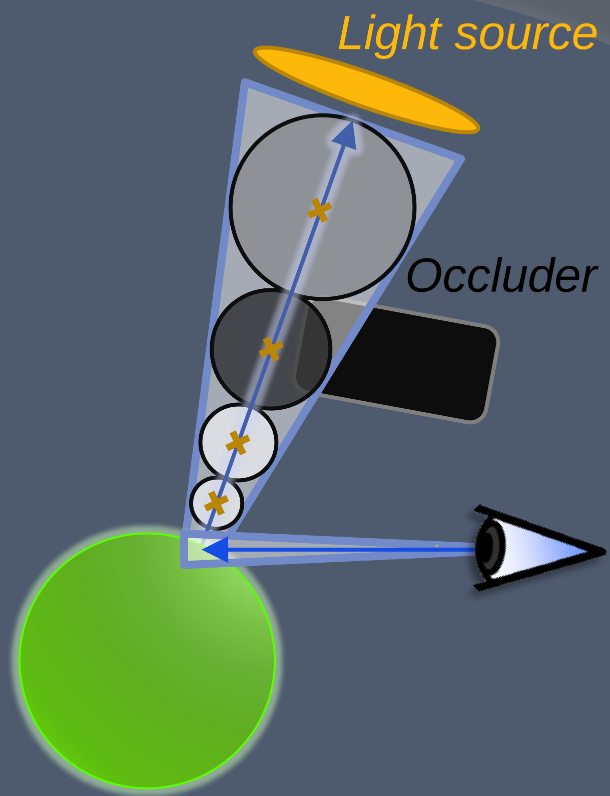
Let's look more closely at what we are doing…

- For a given pixel:
  - Approximate cone integration
    - Using pre-integrated data
    - With only one ray !
- Voxels can be modeled as spheres
  - Sphere size chosen to match the cone
    - Linear interpolation between mipmap levels
- Samples distance *d*
  - Based on voxels/spheres size

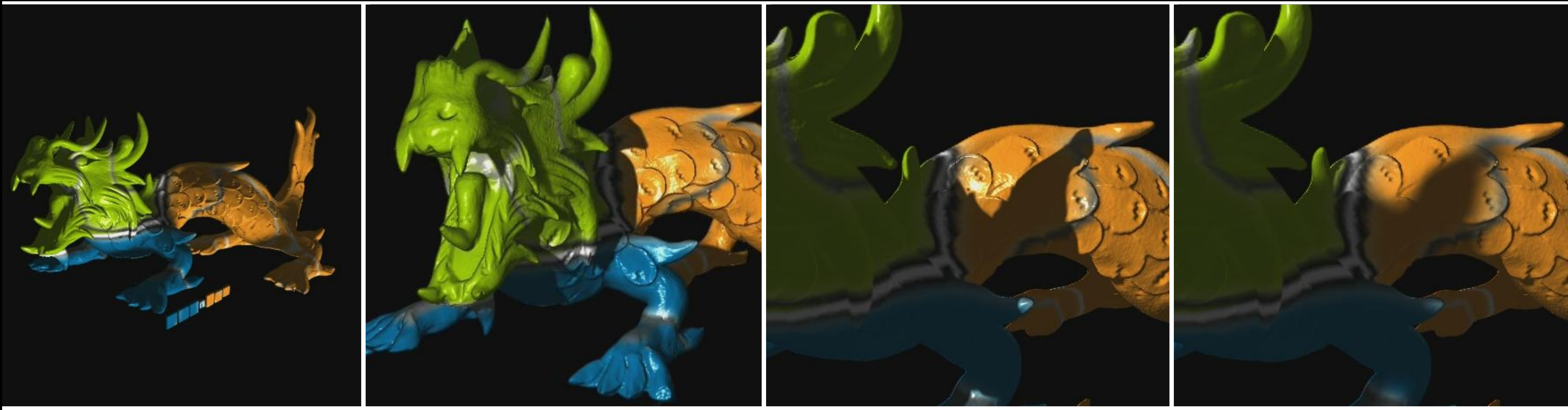


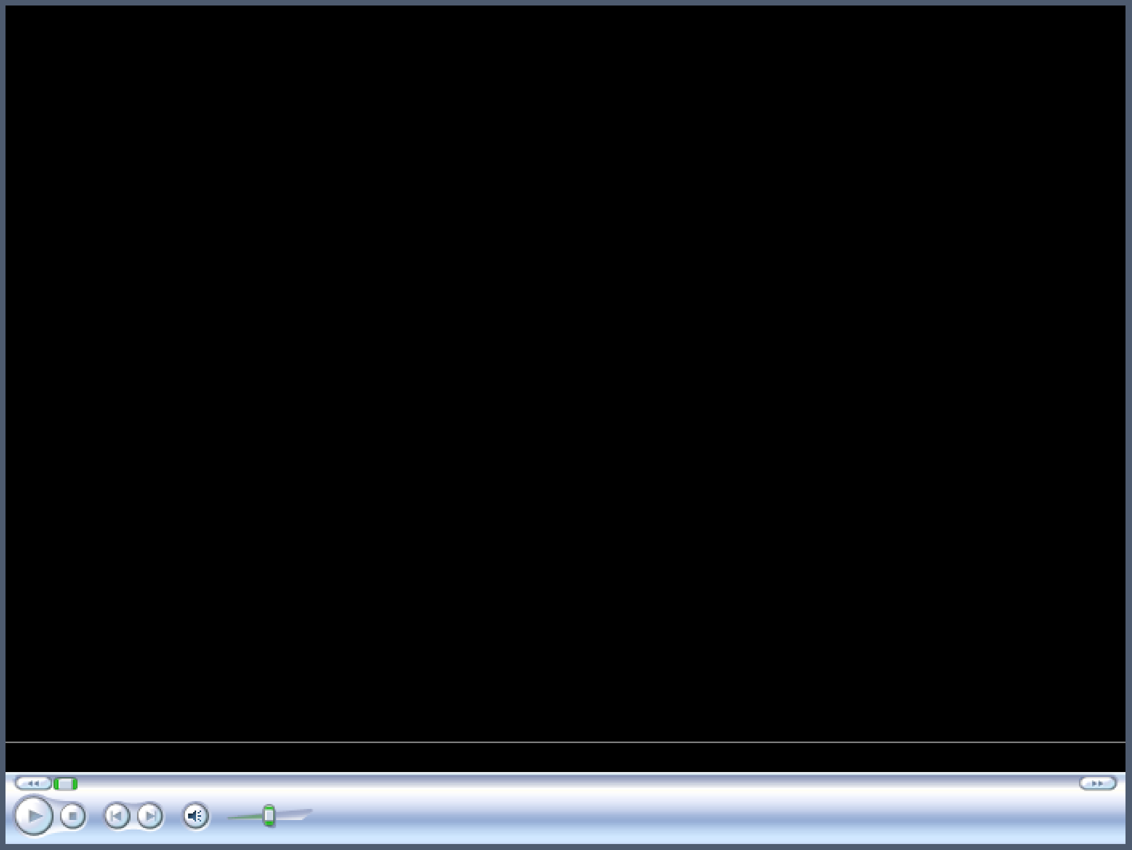
Soft shadows

- Launch secondary rays
  - When ray hit object surface
- Same model as primary rays
  - MipMap level chosen to approximate light source cone
  - Compatible with our cache technique



- Resulting integrated opacity
  - Approximated occlusion





Depth-Of-Field

- Similarly for depth-of-field…
  - MipMap leveld based on circle-of-confusion size

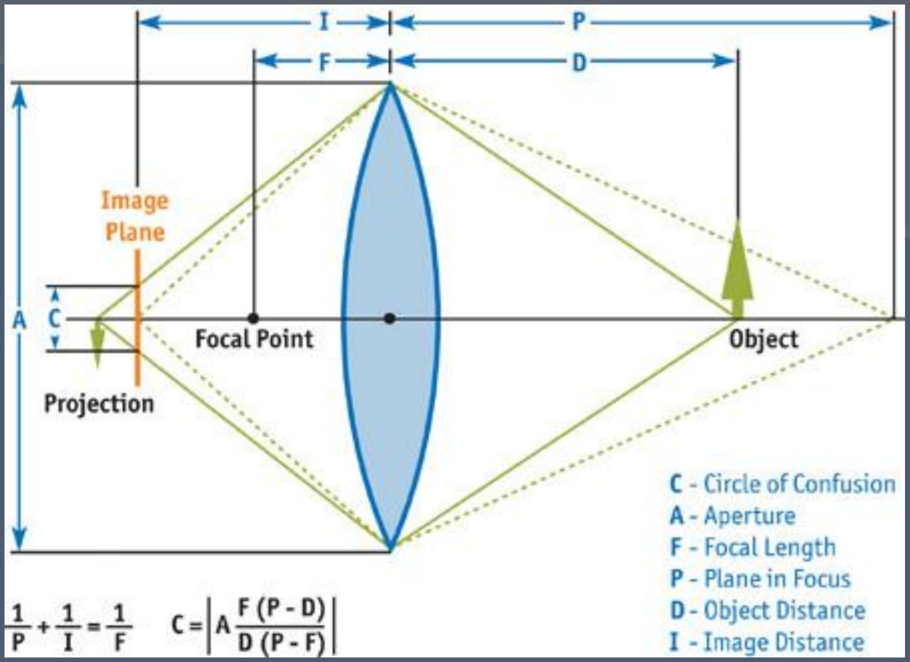
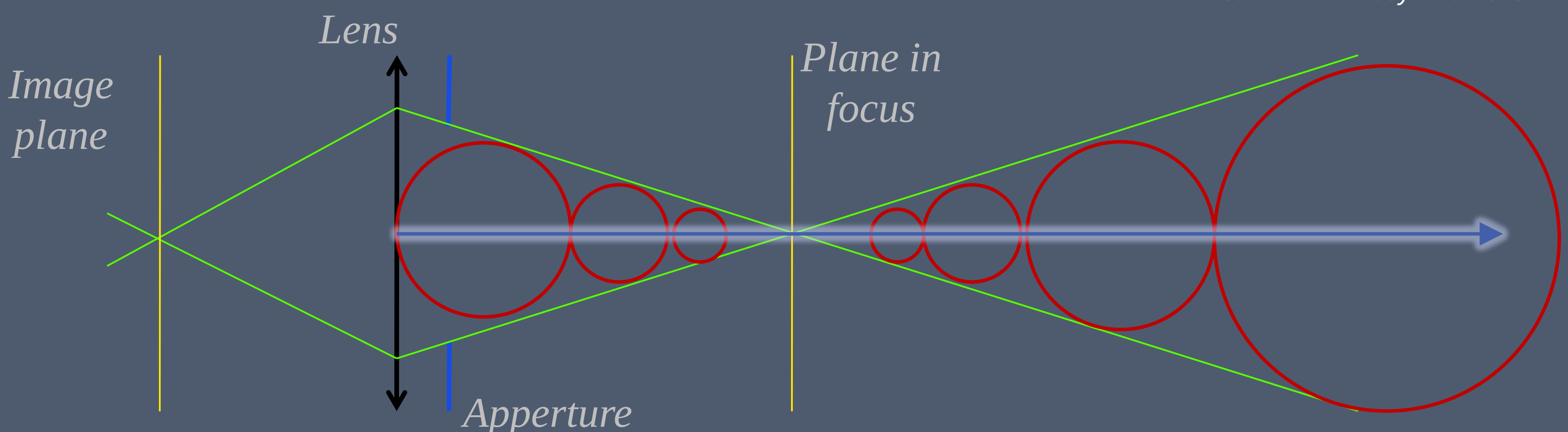
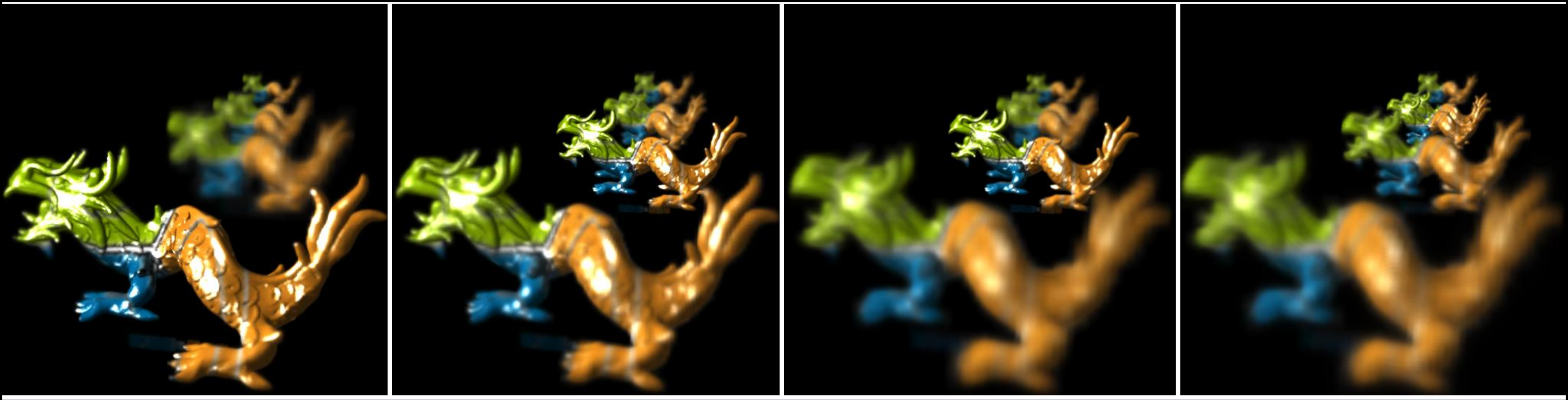
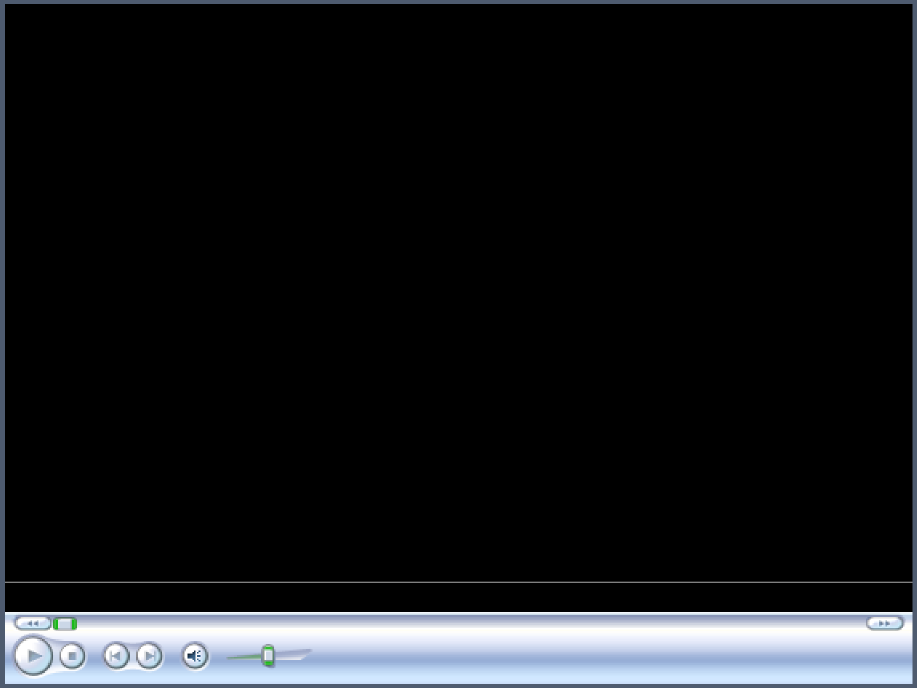


Illustration courtesy of GPU Gems



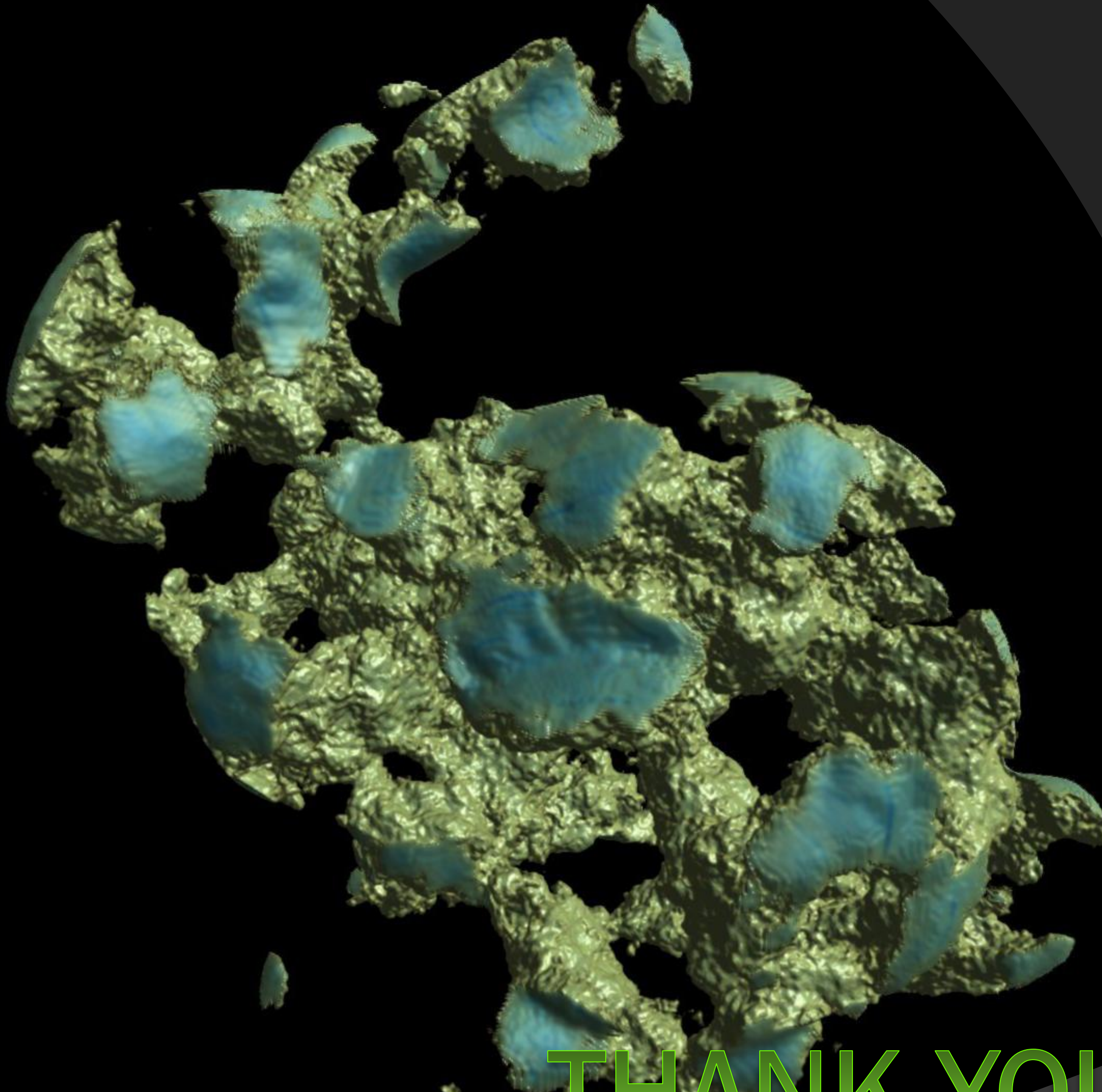



Conclusion

- Unlimited volume data at interactive rates
- Minimal CPU intervention
- Several game techniques can benefit from our algorithm

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
- Erklaerbar

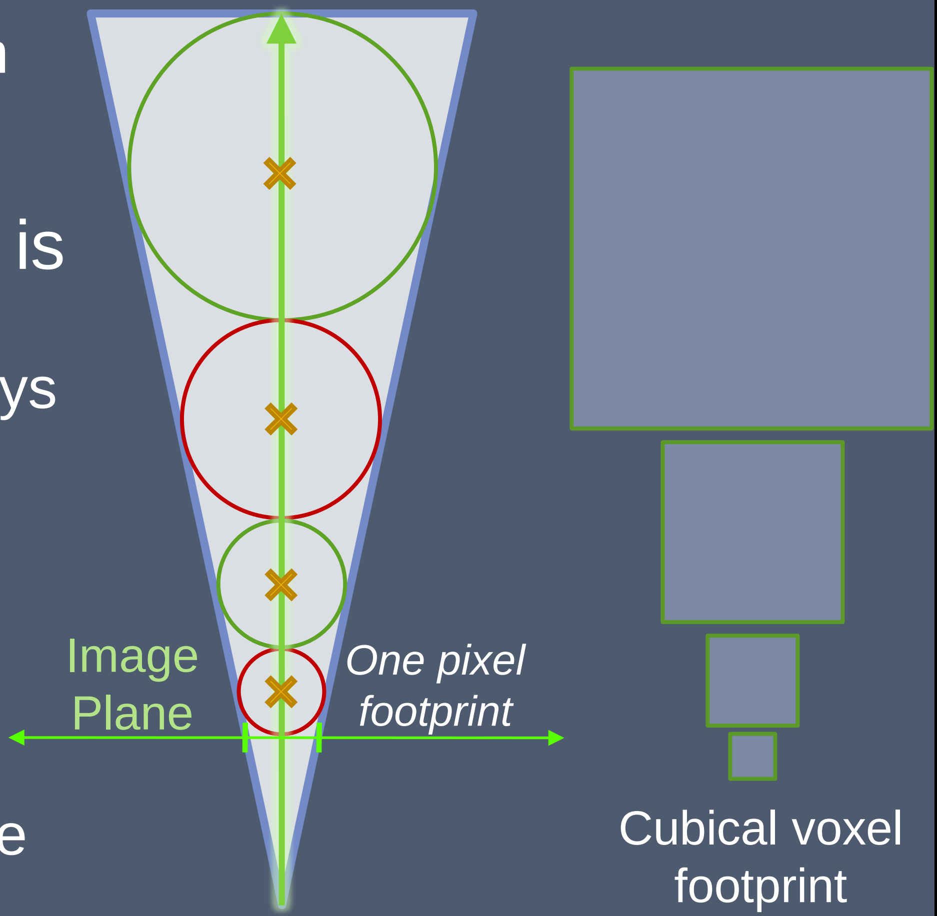


THANK YOU
FOR YOUR ATTENTION

PROBLEMS TO ADDRESS

But there is a little problem…

- Let's see more closely what we are doing:
  - Approximate cone integration
    - Using pre-integrated data
- But the integration function is not the good one !
  - Emi/Abs model used along rays
    - But pre-integration is a simple sum
- Result:
  - Occluding objects are merged/blended
  - Virtually not noticeable for little ray-steps



Emission/Absorption model

- Equation of transfer
  - q : Source term
  - Kappa: absorption

$$I(s) = I(s_0)\, e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(s')\, e^{-\tau(s',s)}\, ds',$$

with optical depth

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(s)\, ds.$$

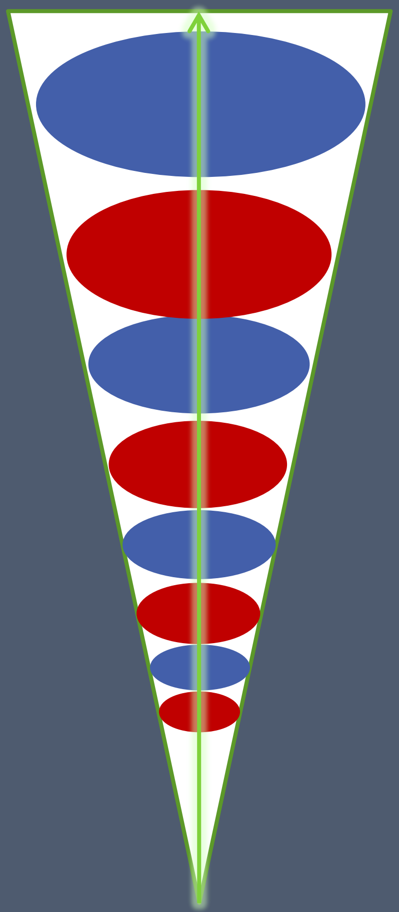
What we would like

- Tangential integration: Sum
- Depth integration: Equation of transfer

- But still avoiding multi-sampling
  - Is it commutative ? Not sure how far we can approximate like this…

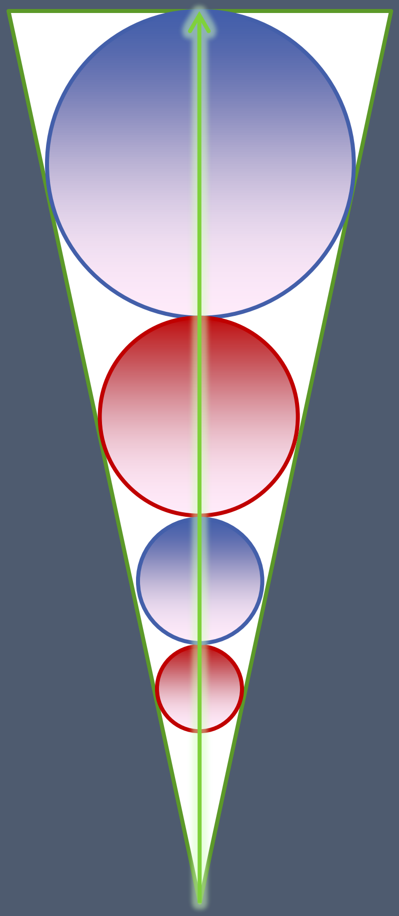
Possible solutions

- Anisotropic pre-integration
  - Similar to early anysotropic filtering methods
  - “2D” mipmapping
    - 1 axis kept unfiltered
- Interpolate between axis at runtime
- Problems:
  - Storage
  - Sampling cost !



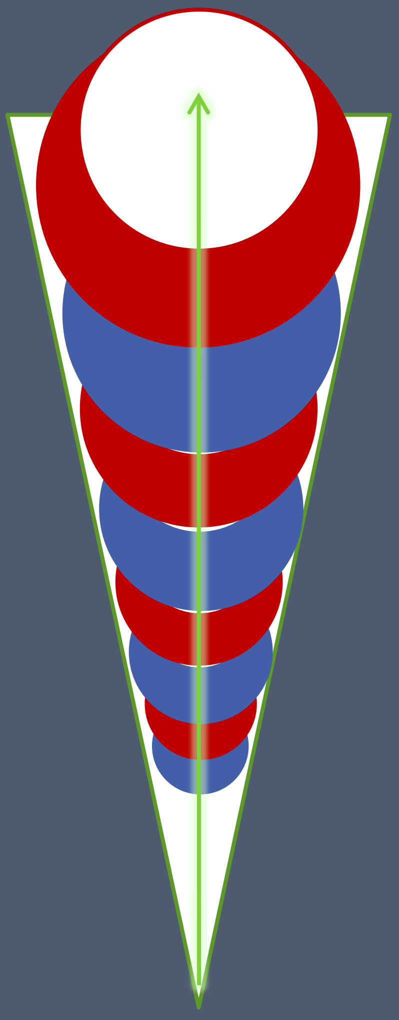
Possible solutions

- Full Anisotropic pre-integration
  - Pre-integrate both parts
    - Light-Transmitance
    - Screen-space average
- Interpolate between axis at runtime
- Problems:
  - Storage !
- We would like to stay anisotropic…
  - Or to reduce storage problem



Possible solutions

- Spheres subtraction
- Problem:
  - Sampling cost
- Any better idea ?



Lighting problem

- How to pre-filter lighting ?
  - Pre-filter Normals
    - How to store them ?
    - How to interpolate them ?
    - Lobes de normales ?
  - Compute gradients on the fly ?