Beyond Triangles

GigaVoxels Effects In Video Games

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

Sylvain Lefebvre, Elmar Eisemann, Miguel Sainz
INRIA Sophia-Antipolis, Saarland Univ./MPI, NVIDIA Corporation
A (very) brief history of voxels

- Rings a bell?

Comanche (Novalogic)

Outcast (Appeal Software)
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
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?
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. $2048^3 \times \text{RGBA} = 32 \text{ GB}!!!$
    - Transfer CPU $\leftrightarrow$ GPU expensive
  - No fast renderer available
GigaVoxels

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

Now implemented with CUDA

GPU

Voxel Ray-Tracer

Output Image

Data usage+requests

Update structure

Sparse Voxel Octree Mipmap Pyramid

GPU Cache manager

Voxel Bricks

On-disc data producer

Data requests

Constant regions
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

GPU

GigaVoxels Ray-Tracer

Output Image

Sparse Voxel Octree Pyramid

Data usage + requests

Update structure

On-disc data producer

Voxel Bricks

GPU Cache manager

Data requests

CPU
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

Tree Descent

1

2

3

Skip Node

4

5

6

7

8

9

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

→ Quadrilinear filtering

- Geometry is texture 😊
- No need of multi-sampling (eg. MSAA)
Incremental octree update

- Progressive loading

Pass 1

Wrong LOD

Pass 2

Wrong LOD

Pass 4

Data request

(Data requests)

(Max opacity)

(Node not reached)

(LoD OK)

(Data request)

(Node pool)

(Brick pool)
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

- GPU
- Voxel Ray-Tracer
- Output Image
- Sparse Voxel Octree Pyramid
- GPU Cache manager
- On-disc data producer
- Voxel Bricks
- Constant regions
- Data usage+requests
- Update structure
- Data requests
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

Cache Elements (Node Tile/Brick)

Octree/Bricks Pool

Usage sorted nodes addresses

Usage sorted nodes mask

New elements

New data

Stream compaction

Concatenate

Stream compaction
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 sculpting
  - *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 anisotropic 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?