A Voxel Rendering Pipeline in CUDA for Real-time Indirect Illumination - β

Dave Kotfis

Jiawei Wang

What’s New?

- Voxelization Efficiency – 1024^3 Voxels
- Texture Mapping
  - Mesh Pipeline
    - In Progress - OpenGL Shaders use Texture Coordinates
    - In Test - CUDA Rasterizer does the equivalent
  - Voxel Pipeline
    - Complete - New VoxelPipe shader maps texture color directly into voxels
    - In Test - Additional OpenGL shaders to use CBO created from colored voxels
- Sparse Voxel Octree Construction
    - Complete - Construct Node Pool in GPU memory from VoxelPipe data
    - In Test – Extraction of cubes from SVO leaves
    - Just Starting – Mip-Map RGBA values from leaves into higher levels in brick pool

Sparse Voxel Octree Construction

- 64-bit Nodes for first child address + value or brick pointer

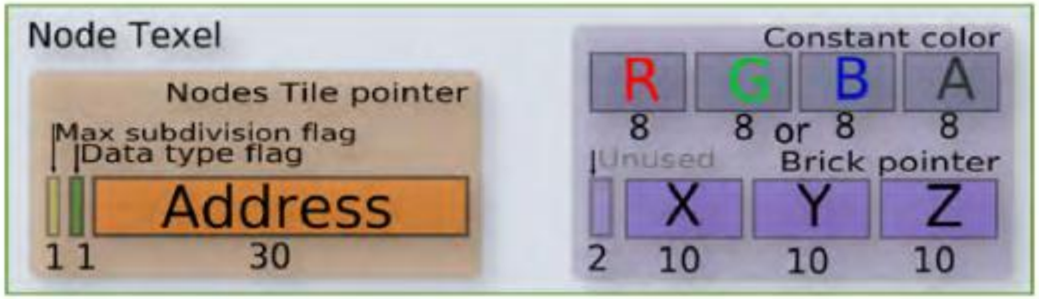
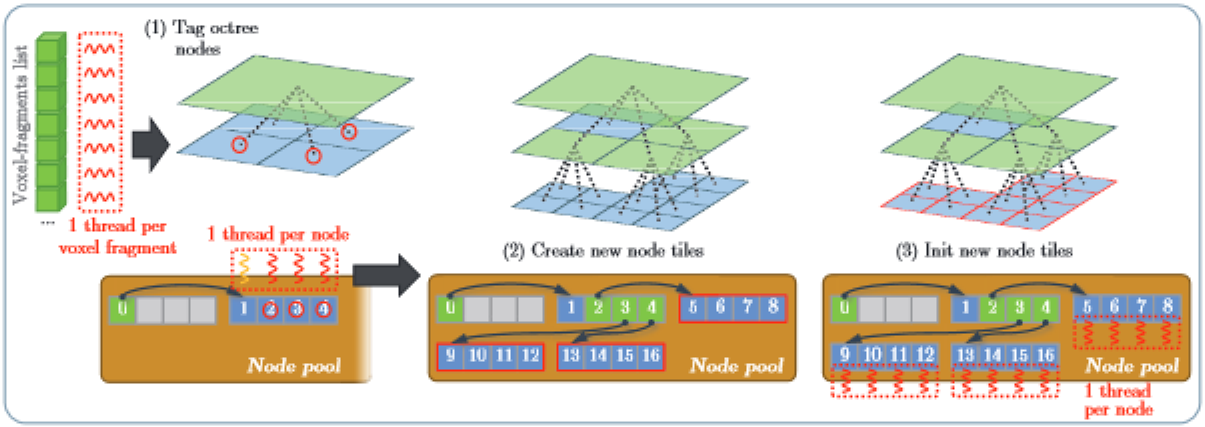


Image Credit: GigaVoxels by Cyril Crassin

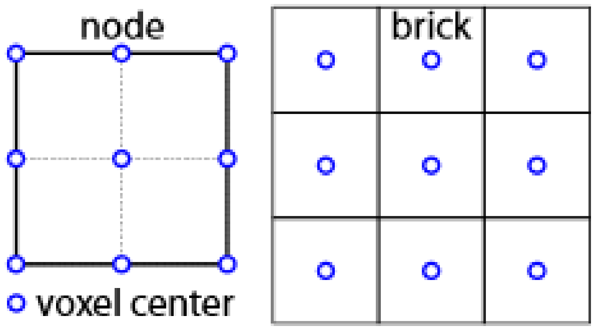
- Top-Down construction of node pool in global memory.

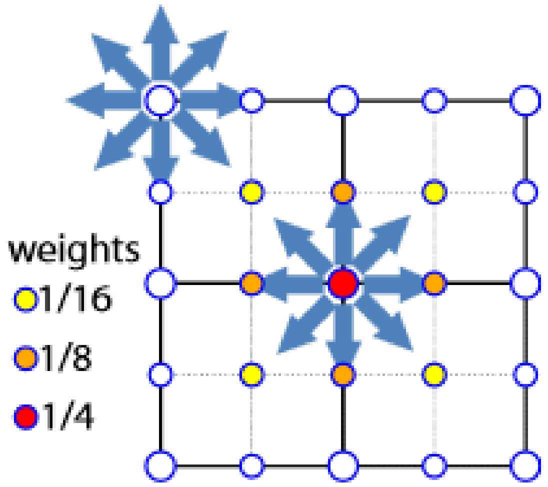


Taken from OpenGL Insights Chapter 22

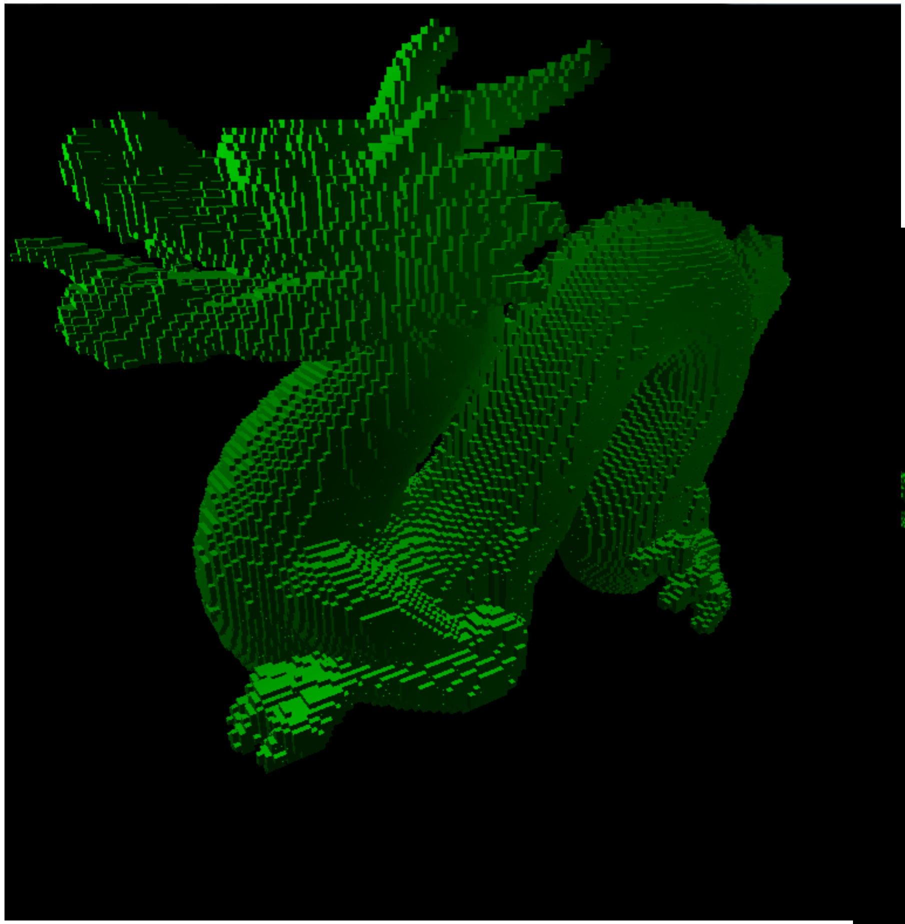
Mip-Mapping the Bricks

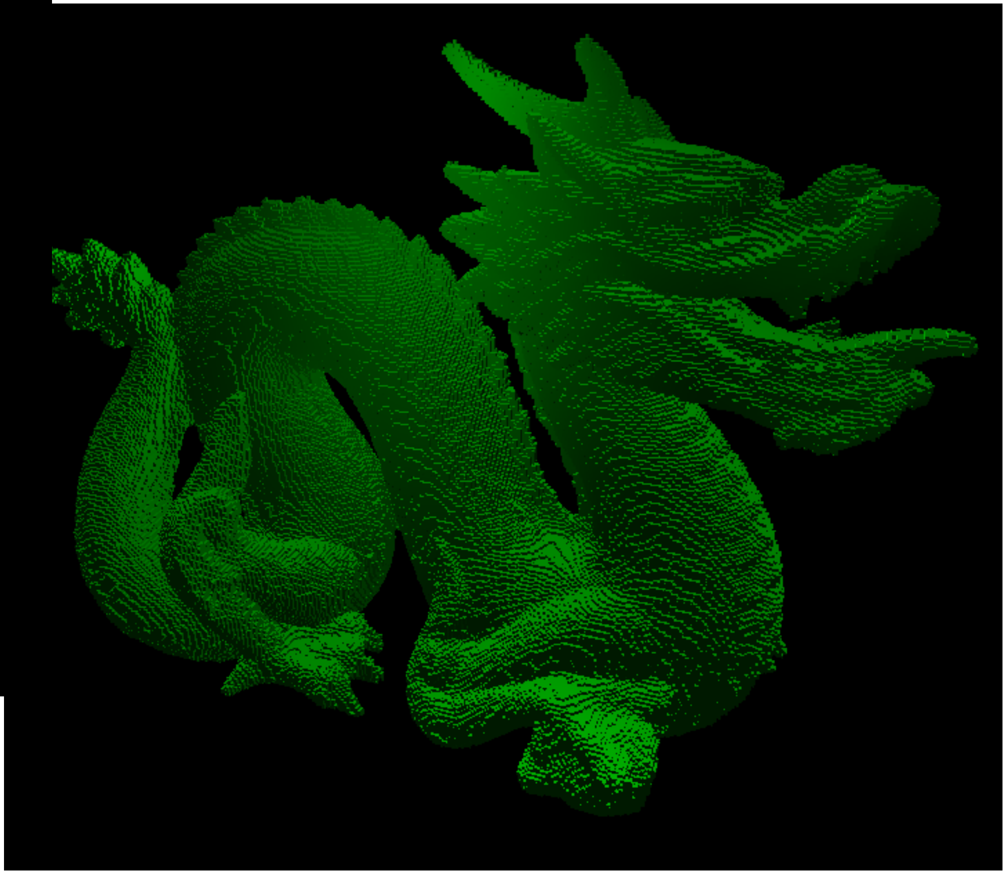
- Use 3x3x3 bricks in texture memory
- Redundant overlapping edges allow for texture interpolation anywhere in the node.
- Use appropriate weights when mip-mapping to account for double-counting.





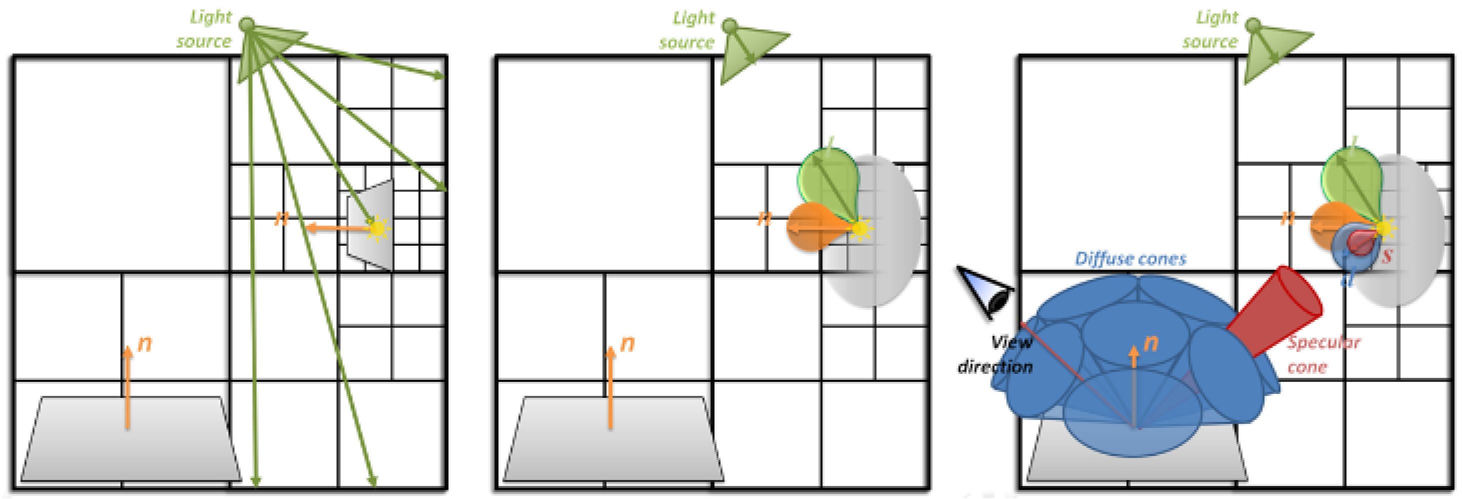
New Results





Next Week…

- Rendering lighting into the SVO



Step 1: Render from light sources. Bake incoming radiance and light direction into the octree

Step 2: Filter irradiance values and light directions inside the octree

Step 3: Render from camera. Sample diffuse + specular BRDF components using voxel based cone tracing