Voxelization – on the fly

FOR EACH brick to produce

- voxelize mesh by computing closest distance from each voxels to mesh

2 pass algorithm :

-- create 3 temporary 3D textures of size 1 brick (+ border) to store distances from mesh to each axis (x,y,z)

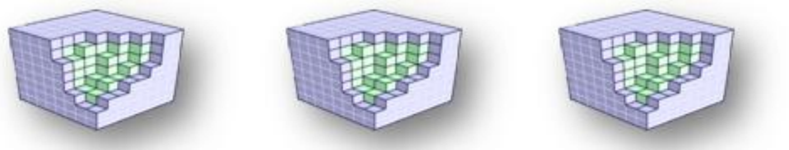
-- [1] on demand, rasterize mesh and store distance to each axis (orthographic projection, camera align to brick, viewport of size of brick)

-- [2] fill « data pool » by storing, at each voxel, shortest distance along the 3 axes (i.e it produces an « approximate » Signed Distance Field)

Normals are then computed from Signed Distance Field with a « gradient » method

Voxelization - on the fly

Voxel : 3 float channels
[normal.xyz]



0 initialization

create spatial acceleration structure
on HOST (« octree » until MAX level)
- vertex array (position, normal, index)

1 HOST producer

2 produce Nodes

check octree if there are triangles inside node
- write info in memory mapped nodes buffer
- If depth more than MAX, ask parent node

« memory mapped »
nodes buffer

write node info
(empty, data, max resolution)

read node info

3 DEVICE producer

4 produce Nodes

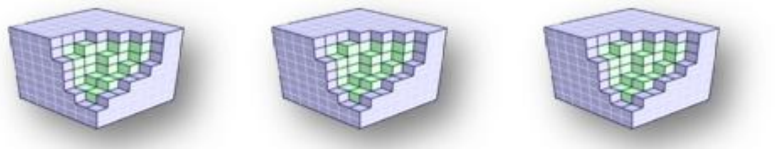
oracle reads the shared
« memory mapped » nodes buffer
and update « node cache »

Voxelization - on the fly

Voxelization

- rasterization with orthographic projection and viewport of size 1 brick (+border)
- camera plan align with brick
- centered at half voxel (texel)
- 1 voxel corresponds to 1 pixel (but many fragments inside)

Voxel : 3 float channels
[normal.xyz]



1 HOST producer → 2 produce Bricks →

Voxelization
write mesh normals in data pool with OpenGL interoperability (GLSL shader with image load/store)
- imageAtomicAdd() is used to smooth normals that will be « normalized » in GigaVoxels shader

↓

3 DEVICE producer → 4 produce Bricks → do nothing…