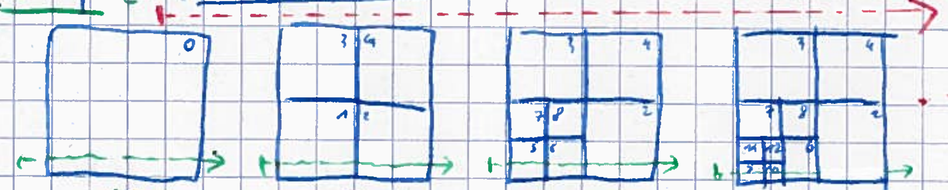
VBO generation

1 Rendering

↳ node visitor::visit() (≠ renderDescentOctree)

Example: ROOT node / renderDescentOctree methode = "Flag EACH node traversed at EACH level of LOD starting from ROOT at LOD = 0"



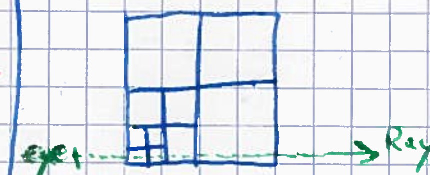
CONCLUSION ⇒ used nodes: $\{N_0, N_1, N_2, N_5, N_6, N_9, N_{10}\}$ ⇒ Flag ALL nodes from hierarchy

⚠ PROBLEM ≠ $\{N_9, N_{10}, N_6, N_2\}$ ⟵ these nodes are at their maximum resolution

⇒ this is View-Dependent

Description of the Problem

Say you have that configuration



⇓

WHAT YOU WANT IS:

↳ "list of traversed nodes"

BUT = with no duplication ⚠

Because we will BUILD the content of a VBO (points) at each frame with points of only used bricks.

2 Data Production Management — Requests handling

[A] ⇒ initialization of the program: 1 BIG VBO ex 10^6 elements (i.e. points)

idea = BRUT force Algorithm ⇒ we're going to fill/update the content of the VBO, given only "VISIBLE" nodes at a given frame.

ALGO → CacheManager::updateVBO() methode

1. Retrieve CURRENT LIST of VISIBLE Nodes
   - d_TempMaskList2 ⟶ list of FLAGS
2. COMPACTION
   - d_ElemAddressListTmp ⟶ list of visible nodes (addresses)
3. Read Nb of points inside each brick (the visible ones)
   - d_vboBrickList ⟶ list of Nb of points
4. Parallel prefix sum (start at 0) ⚠
   - d_vboIndexOffsetList ⟶ list of offset to be able to know where to WRITE in the VBO
   - 3' reduction to sum All nb points
5. ‖ MAP openGL VBO to CUDA Context
   - GL: vbo GLuint ⟶ float3 vboDevicePointer CUDA
6. Update VBO content
   - vboDevicePointer
7. ‖ UNMAP openGL VBO from CUDA context
8. ‖ Render openGL VBO with GLSL ⟶ glDrawArrays(GL_POINTS, 0, nbPoints)