G-PCC:
Review of octree parallel substreams

m53684 — D. Flynn, K. Mammou

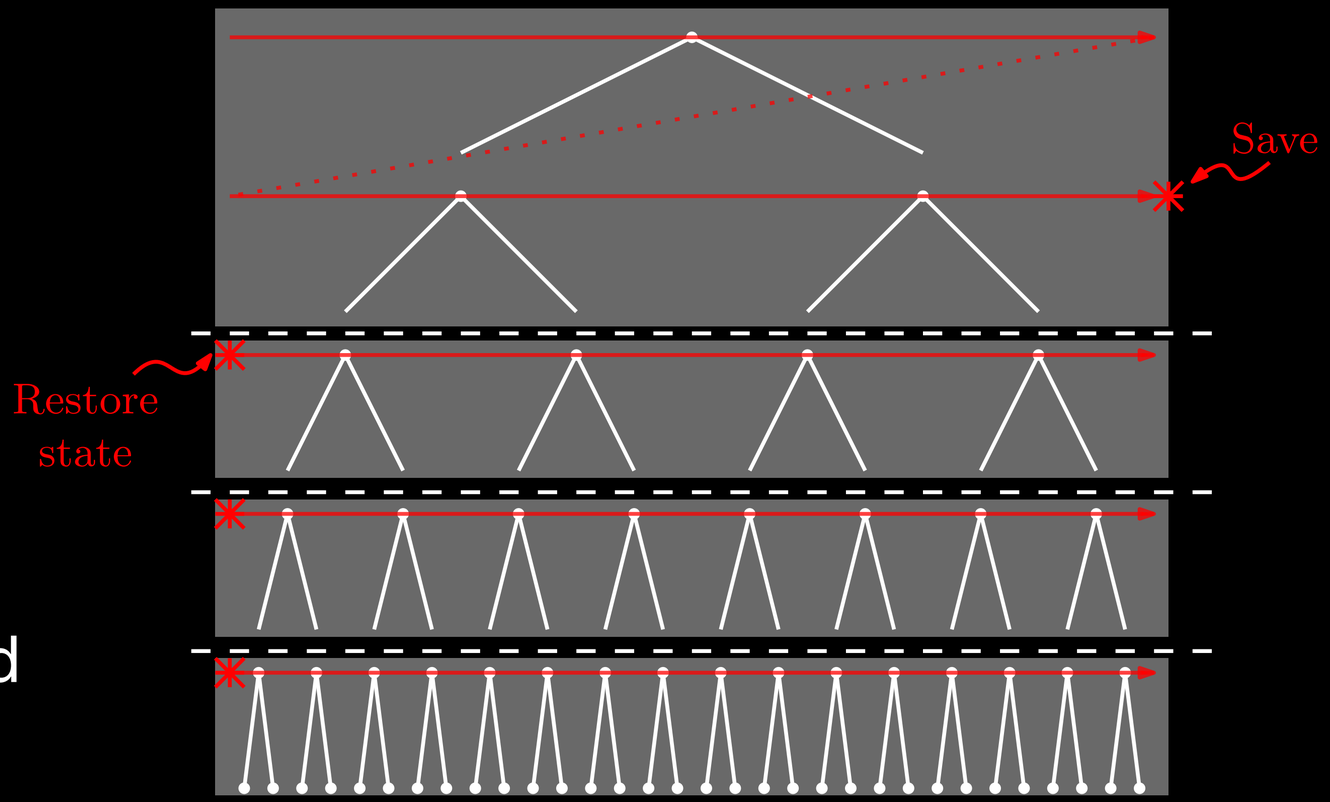
Overview

Octree parallel sub-streams

- At end of a given tree level:
  - Save entropy state
  - Terminate first entropy sub-stream

- At each subsequent level:
  - Restore entropy state at start
  - Terminate entropy sub-stream at end

- Write sub-stream lengths to data unit header



Proposal

Avoid confusion between node size and level

- Original proposal used the node size to signal termination

- This used to be equivalent. But: —
  - max/min node size can be identical in multiple levels due to QtBt tweaks

- Avoid ambiguity by referring to the number of parallel layers

Proposal

Remove non-parallel stream

- The second sub-stream cannot be parallel encoded or decoded:
  - Data dependency on the context state at end of first-stream

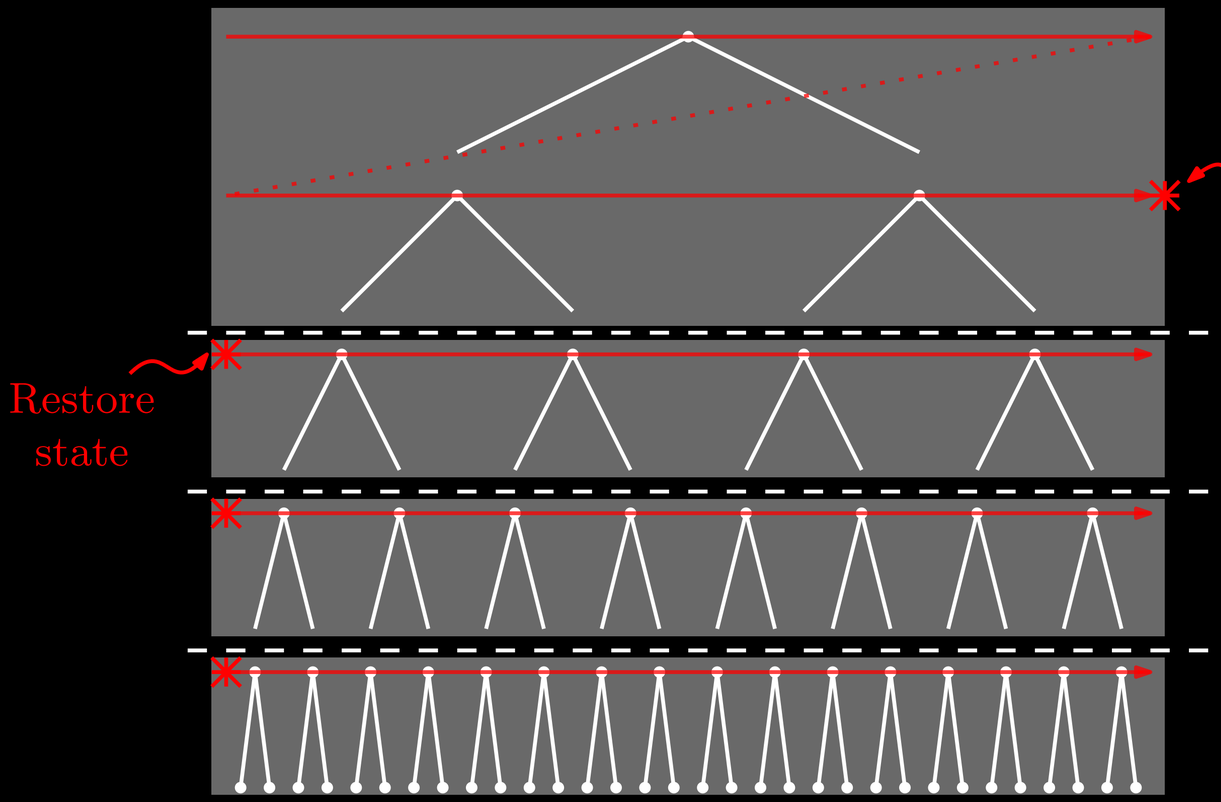
- Awkward signalling to avoid permitting the two sub-stream case

- Proposal: merge the first two streams
  - Save state at same place
  - Terminate first-stream one level later

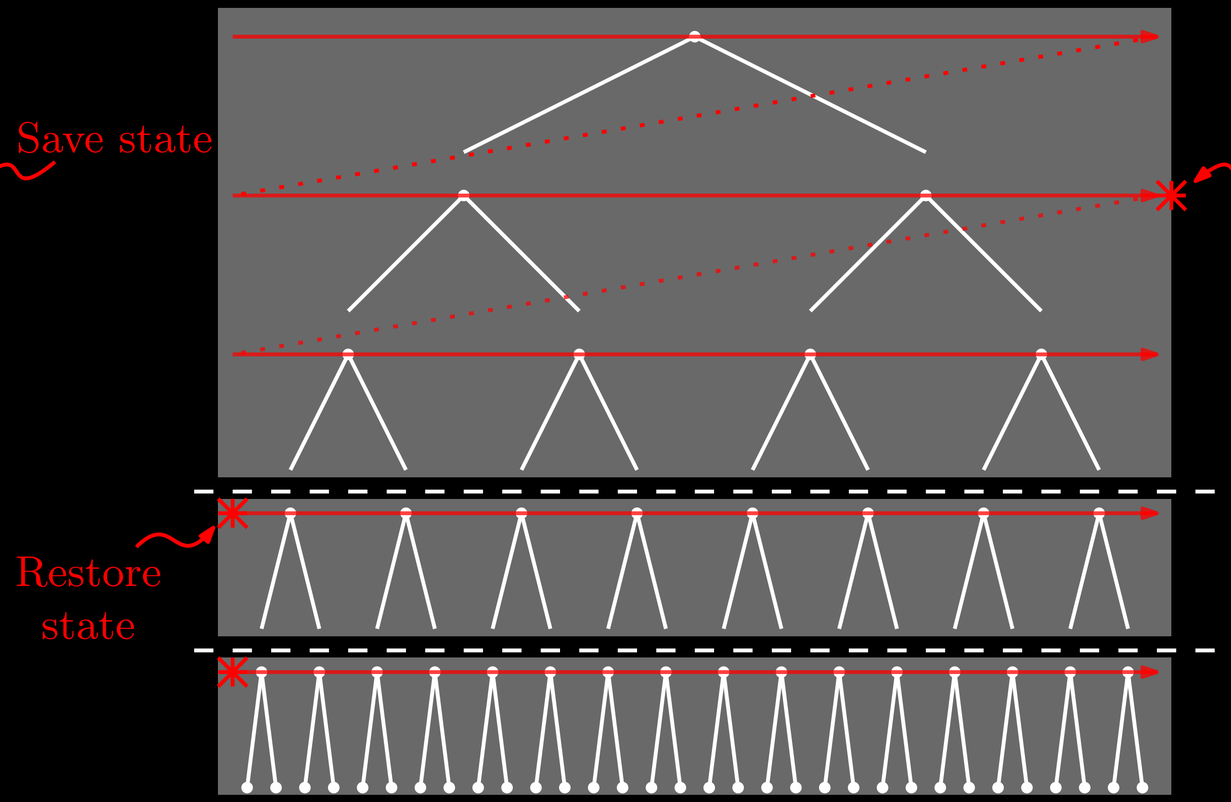
Proposal

Remove non-parallel stream

Current



Proposed



Question:
Is this practical for a decoder?

- To exploit this parallelism is complicated:
  - Data dependencies have variable latency
    - Octree nodes in Morton order <> Neighbours need spatial adjacency
  - Mismatched input/ouput rates
    - Octree nodes expand the tree
    - Special case of matched input / output better handled by IDCM
- Feature is optional
  - Cannot be relied upon by a decoder: Would it ever be used?
- Encoder: maybe, but still has issues

Proposal

Don’t signal offsets

- Instead, signal a per tree level flag:
  - At the end of each level
  - Indicates the save state point

- Single entropy stream:
  - Flush and byte-align arithmetic decoder at start of subsequent levels
- Bypass entropy streams:
  - Flush/discard and truncate the current chunk