Tim Dettmar

Parallel ASTC Texture Compressor

Update

Outline

Recap

- General-purpose image compression algorithms are...
  - Optimized for space efficiency
  - Difficult to determine the output size given only the input size
  - JPEG, PNG, HEIC, AVIF, etc.
- Texture compression is image compression designed specifically for GPUs
  - Balances performance and space efficiency (file and decode HW)
  - Random access ideal for perf sensitive apps: games, CAD etc.
  - ASTC is one of the most complex of these formats
  - ASTC's complexity makes it extremely slow to encode
- mpASTC leverages parallelism, reducing wall-clock encoding time

Outline

Compressor

- Initially a unoptimized search was used
  - Far too slow for blocks with >2 colours!
  - A single block could take hours to encode in the worst case
- astcenc-like implementation infeasible due to time constraints
  - Quality and performance of mpASTC probably will not be as good
  - Lacking the hand rolled assembly, heuristics, etc.
- Compressor will use techniques described in astcrt
  - Lower-complexity implementation with reasonable quality

[Oom]

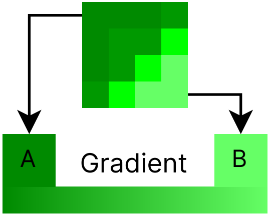
Target Feature Set

Limited support of the ASTC feature set

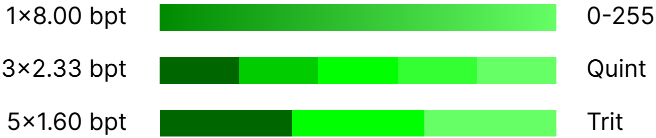
- RGB LDR colour profile
- 4x4 block size
- Fixed texel weight count (16)
- Fixed colour endpoint count (2)
- Limited partitioning support

General Process

Endpoint Selection



Quantization



Of 128 bits, ~96 available for encoding

- Many images are far more complicated than this simple gradient!
- ASTC only allows for specific quantization ranges

[Khr]

Quantization Ranges

- ASTC standard specifies variable encoding for colour and texels
  - Weights
    - Min. 1 bit, max. 5 bits (2 - 32 states)
    - = 16 - 80 bits
  - Colour Endpoints
    - Min. 1.3 bits to 8 bits (2 - 256 states)
    - Fewer bits if more partitions in use
    - = 8 - 48 bits per partition

[Khr]

Parallel Block-Level Search

- Different combinations of colour endpoints and quant levels can be tried in parallel
- This strategy would be very thread-heavy
  - i.e., may only be feasible on GPUs
- In any case, the combination with the highest PSNR is used

Outline

Work Dispatch

- Trivial solution: split work evenly across all threads
- Flawed: not all blocks take the same time to encode
  - Single colour: very fast to encode
  - Shades of a single colour: small search space
  - High-entropy data: large search space
    - Almost guaranteed to be lossy
    - Exhaustive search is infeasible
    - Billions of possible encodings
- Increasing work efficiency requires dispatching work dynamically

Work Dispatch - Node Level

- Avoiding intermediate buffering through sending several rows simultaneously
  - Dispatch size of a 4x4 block = 4 rows x image width
  - Placed into a receive buffer with a preset size
  - Sending partial width increases communication overhead
  - Parallelism sufficient with full-width method

Work Dispatch - Work Unit

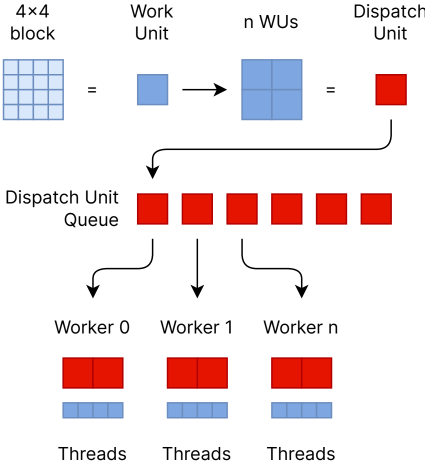
- Split from the Dispatch Unit are Work Units
- A “Work Unit” is a single 4x4 RGB block to be compressed
- Further parallelism theoretically possible but no benefit

Implementation

- Actual block compression functions identical between serial/parallel
  - Work dispatch for parallelism is an additional layer on top
  - Work unit size always blocks of 4x4 pixels
  - Thus, sequential implementation simply loops through all blocks

Parallelization

- Each worker double-buffers two DUs
- Each thread is dynamically allocated a WU from the **worker** DU queue
- When done, worker:
  - Flips buffer
  - Sends compressed result to rank 0
  - Requests another DU from rank 0
  - Uses non-blocking MPI functions



Remaining Components

- Experimentation with encoding parameters
- Search space optimization
- Block-level parallelism
- Visualization

References

[Khr] Khronos Group. *Khronos Data Format Specification Registry*. URL: `https://registry.khronos.org/DataFormat/`.

[Oom] Daniel Oom. *Real-Time Adaptive Scalable Texture Compression For the Web*. URL: `https://hdl.handle.net/20.500.12380/234933`.