A 3D Pipeline on the Foenix A2560U/K/X(68k)

v0.1-alpha

Brief overview

A 3D pipeline takes a 3D ‘mesh’ – a structure of vertices and their topological connectivity, as an input, and produces a rasterized ‘projection’ of the mesh in an output frame buffer (fb); as such a basic pipeline comprises of two domains:

- Vertex processing – vertex attribute transformations; as a minimum: Cartesian-position transformations
- Primitive (read: triangle) rasterisation; as a minimum: ‘flat shading’ – some constant color over the primitive area

Brief overview, cont’d

A 3D pipeline designed for the 68K Foenix series is under develpment, currently meeting the minimal definition, and tuned for A2560U baseline:

- 68000 CPU – fx arithmetics, two’s complement
  - fx16.14 for full-range values – top two MSBs replicate sign in 32bit int
  - fx1.14 for normalized-range (±[0, 1]) values – top MSB replicates sign in 16bit int
- Dual-ported SRAM pool in the Altera Cyclone FPGA, responsible for the text buffers in Vicky II; used as output fb (low-res 80x60..100x75)
- Vicky II VBLANK interrupt; ‘Start-of-Frame’ (SOF) in Foenix documentation

The dual-ported feature of fb SRAM is important when using single-buffer rasterisation or when keeping a back-buffer in the same pool as front-buffer, subject to sink scan-out access; less so when using a back-buffer elsewhere (e.g. CPU RAM), with subsequent copying to front buffer. More on that next.

Render to back fb vs. render to front fb

- VBLANK window is really small against the entire frame duration
  - entire frame: 16.6ms @60Hz
  - VBLANK window: about 9% of frame time = 1.42ms, on A2560U 640x480@60Hz
- Performing the entire scene rasterisation at VBLANK not practical outside of very limited scenarios (read: truly low-poly wireframe models)
- Beam-racing techniques – rasterization follows the top-to-bottom frame scan-out order, while predating the scan-out by some small margin
  - 3D scene/mesh definition does not naturally follow a top-to-bottom order
  - extra work required at rasterisation – collecting scanlines from primitives and plotting them top-to-bottom; at the very least screen-space sorting of primitives
- Render to back-fb independent of fb scan-out order or VBLANK window size; copying of the back-fb to front-fb is fast – can complete during VBLANK window

Overview of the current pipeline w/ back fb

1. static (pre-frame) xform (all mesh verts)

   frame-loop-iteration-begin

2. dynamic (per-frame) xform (all mesh verts)
3. for-each tri
   a. compute tri parallelogram/barycentric basis (PB)
   b. if-visible (positive PB area) rasterise tri to back fb w/ fb scissoring
4. wait for VBLANK (SOF)
5. copy back fb to front fb

   frame-loop-iteration-end

Current pipeline limitations

- Target-fb scissoring, AKA scissoring-by-rect, insufficient to provide a universal 3D pipeline; traditional vertex pipelines perform a 4D-frustum-clipping step post-vertex-transform, producing ‘Normalized Device Coordinates’ (NDC) space; that can be expensive in both ALU and mem accesses on 68000 (non-pipelined) hw.

What is the minimal work we can do to still offer a universal pipeline?

- Perspective projection requires a div op by a value proportional to vertex distance from the virtual camera origin
- Unless all scene content is guaranteed to reside in the ‘good’ half-space in front of the virtual camera, such a division can cause havoc with out-of-bound ‘bad’ divisors: near-zero, zero, negatives distances
- A vertex ‘near-clipping’ step is required to eliminate such hazards: complete or partial primitives falling ‘behind’ the camera; read: falling behind a virtual-camera projection plane
- Vertex clipping is both a subtractive and generative step – it can produce new primitives.

How to implement vertex near-clipping at minimal cost?

Triangle clipping at the last moment

Traditional 3D pipelines would take the post-vertex-transform arrays (vertex and index) of a mesh and produce new such arrays at vertex clipping. On the 68000 we solve that differently:

- a post-vertex-indexation JIT approach, minimizing the memory accesses
- a tri primitive is clipped right before rasterisation; clipping can produce:
  - zero primitives – all three post-indexation vertices clipped away
  - one (new) primitive – two vertices clipped away, two new vertices formed at clipping plane
  - two (new) primitives – one vertiex clipped away, two new vertices formed at clipping plane; resulting quad trivially subdivided into two tris

Overview of the current pipeline w/ back fb (old baseline)

1. static (pre-frame) xform (all mesh verts)

   frame-loop-iteration-begin

2. dynamic (per-frame) xform (all mesh verts)
3. for-each tri
   a. compute tri parallelogram/barycentric basis (PB)
   b. if-visible (positive PB area) rasterise tri to back fb w/ fb scissoring
4. wait for VBLANK (SOF)
5. copy back fb to front fb

   frame-loop-iteration-end

Overview of a pipeline w/ back fb & vertex near-clipping

1. static (pre-frame) xform (all mesh verts)

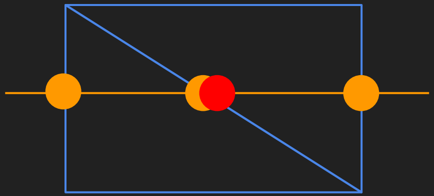
   frame-loop-iteration-begin

2. dynamic (per-frame) xform (all mesh verts)
3. for-each tri
   a. clip tri by near plane
   b. for-each-new-tri
      i. optional per-vertex perspective division (N/A for parallel projection)
      ii. compute tri parallelogram/barycentric basis (PB)
      iii. if-visible (positive PB area) rasterise tri to back fb w/ fb scissoring
4. wait for VBLANK (SOF)
5. copy back fb to front fb

   frame-loop-iteration-end

Tradeoffs of the new pipeline

Clearly a perspective division happening post-indexation poses work duplication along shared edges, which can be taxing on the non-pipelined architectures:



(dots mark newly-formed vertices from clipping, requiring perspective division)

Tradeoffs of the new pipeline, cont’d

We can partially amortize the perspective divisions

- Vertex transforms yield unnormalized vertex data in fx16.14 format
- When doing the actual scan conversion we round that to whole numbers – no sub-pixel correctness (sorry)
- The conversion fx ⇒ whole can be combined with the perspective division into a single div op:

  fx16.14 / fx16.14 ⇒ s16

  (as long as we can guarantee the magnitude of the divisior, which we do after the near-clipping step)

As div(s32,s32) not found on 68000, amortisation may not be viable on the most constrained baseline platform. More on that next.

Future considerations

- The original 3D pipeline doesn’t use the fxmath copro in Gabe; consider using div(s32,s32) op on the copro for amortising the persp division into the fx16.14 ⇒ s16 conversion.
- Moving the pipeline from low-res fb in dual-ported FPGA SRAM to high-res fb in ‘proper’ Foenix VRAM (again, low-lat *write* SRAM) would not require much effort, once the VDMA mechanism is in place and both back-fb and front-fb reside in VRAM. As that memory is not dual-ported, examine access times by the CPU.
- On the DRAM-equipped A2560K/X, the back-fb can reside in DRAM, if low-lat *write* access is provided, and DRAM ⇒ VRAM DMA is availble.