ECE 571 – Advanced Microprocessor-Based Design Lecture 33

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

4 December 2024

Project/HW Reminder

- Homework #11 will be posted, Nvidia Reading
- Sent the proposed presentation schedule

From Last Time – Flash

- TLC flash
- Regular, MLC, TLC, QLC... the transistor used is exactly the same
- For TLC they just store 3 bits in the same sized transistor
- This is why TLC and QLC are denser but can perform worse
- Essentially what happens, there's a second gate on the transistor. When you program it you inject electrons into the second gate which changes the switching voltage of

the transistor. By allowing more states between on/off you can store more bits

- Storing more bits is tricky, more error prone, and it tends to wear out faster. Also slower to program as you have to make sure you've got the exact level you are aiming for.

A Brief History of Computer Graphics

Old CRT Days

- First graphic displays essentially oscilloscopes
- Electron gun shooting at glass covered in phosphor
- Vector displays: change X and Y of beam with magnets
- TV type displays: beam draws rectangular screen at roughly 60Hz (it's complicated)
- Horizontal Blank at end of line as beam returns to left
- Vertical Blank at end of screen as beam returns to top
- TODO: diagram

Ideally you'd have a Framebuffer

- Bitmap in memory where grid of bits would say what color each pixel on screen should be
- In early days this was expensive.
  - Apple II: 140x192x6 colors = 8k
  - MCGA (VGA mode 13h): 320x200x256 colors = 64k
  - 1024x768x256: roughly 1MB

Early Hardware Limitations

- Atari 2600 – only enough RAM to do one scanline at a time, had to “race beam”
- Apple II – video on alternate cycles, refresh RAM for free, 8k bitmap for 140x192x6 colors

Memory Limitations – Limited Bandwidth

- Use lookup tables. Palettes, a number in the bitmap and can have maybe 256 colors from a palette of millions (if clever, change this per scanline to gets lots on screen at once)
- Memory planes (EGA, VGA), to speed up access can have R/G/B colors in separate memory banks and read out in parallel. Code had to split up drawing across the bitplanes which was complex/slow
- Tilemaps (NES, SNES), instead of framebuffer, have

tilemap with grid of tiles, the value looks up into a tile table that has a set of bitmaps. Faster and use less memory than full framebuffer

Early Graphics Acceleration

- Sprites – small bitmaps that can be moved on screen
- Scrolling – pan across memory to quickly scroll screen
- Copper – small co-processor can modify graphics independent of main CPU
- 2d acceleration – drawing lines, rectangles, mouse cursor

Old 2D Video Cards

- Framebuffer (possibly multi-plane), Palette
- Dual-ported RAM, RAMDAC (Digital-Analog Converter)
- Interface (on PC) various io ports and a 64kB RAM window
- Mode 13h
- Acceleration – often commands for drawing lines, rectangles, blitting sprites, mouse cursors, video overlay

Old 3D History

- At first only in high-end workstations (like SGI)
- Gradually came to PCs for gaming (original 3D games in software, all on CPU)
- 3dfx cards, with passthrough cable
- Became more mainstream
- NVIDIA and ATI (bought by AMD)

3D Graphics

- Two common ways to do 3D graphics
  - Ray tracing, very accurate, but slow
  - Rasterization, low quality, but fast enough to do in real time
- Can do either completely in software on CPU (and people did), but much faster if you accelerate with hardware

Ray-tracing / Ray-casting

- TODO: diagram
- Objects placed in 3d space
- Rays of light traced from eye through each pixel on screen until hit object
- Based on material they hit, reflect, refract, take on color, etc
- Can do reverse where light source sends out rays and you bounce them around until they hit pixel on screen

Question: how does Hardware Raytrace work

- Accelerate in hardware ray-tracing, though usually only partially
- NVIDIA: Optix Library
- You describe how rays behave
- Details are a bit hard to get

Ray-marching

- Just a place holder, it's a related technique often used in size-coded demoscene productions and I've been meaning to learn more about it

Rasterization

- TODO: show diagram
- Objects made up of many triangles (or quads)
- Send vertices to card
- Project to 2d screen
- Broken up to pixels and shaded/textured
  Color/shading based on angle with light source (normals)
- Clipping, depth

Rasterization on GPU

- CPU send list of vertices to GPU.
- Transform (vertex processor) (convert from world space to image space). 3d translation to 2d, calculate lighting. Operate on 4-wide vectors (x,y,z,w in projected space, r,g,b,a color space)
- Rasterizer – transform vertexes/vectors into a grid. Fragments. break up to pixels and anti-alias
- Shader (Fragment processor) compute color for each pixel. Use textures if necessary (texture memory, mostly

read)

- Write out to framebuffer (mostly write)
- Z-buffer for depth/visibility

Rasterization Downsides

- Can't calculate shadows (have to do hacks)
- Can't easily do reflections (mirrors), transparency or refraction (water, lenses, glass spheres)
- On the plus side it is fast

GPU Pipeline

- Old / Traditional
  - Implement rasterization in fixed hardware
  - Fixed pipeline (lots of triangles).
- Modern
  - Much more flexible, programmable almost general-purpose compute units
  - Old pipeline can still be implemented in software via the fancier interface

GPUs

- Display memory often broken up into tiles (improves cache locality)
- Massively parallel matrix-processing CPUs that write to the frame buffer (or can be used for calculation)
- Texture control, 3d state, vectors
- Front-buffer (written out), Back Buffer (being rendered) Z-buffer (depth)
- Originally just did lighting and triangle calculations. Now shader languages and fully generic processing

GPGPUs

- Can we use GPUs as an accelerator?
- Started when the vertex and fragment processors became generically programmable (originally to allow more advanced shading and lighting calculations)
- By having generic use can adapt to different workloads, some having more vertex operations and some more fragment