1996-12-04 Bo Nordin (bosse@cb.uu.se)

Lecture notes, computer graphics (F17) (S, (4.3), IF,)

Superworkstation
Not super anymore, but about how the different tasks can be divided between processors

Parallelism (graphics can be partitioned into functional tasks -> multiple processors)
Sequential -> pipelined

The five fundamental task:

1/ *Generation (G)*
in an interactive environment you must be able to specify/generate the image. CG is used for many applications, hence a general purpose processor is adequate.
2/ *Traversal (T)*
Read through the structures and feed the transformation step. Has been done on both special-purpose and general-purpose.
3/ *Transformations (X)*
Transformation, clipping, projection, illumination, shading
4/ *Scan conversion (S)*
Both general-purpose and special-purpose
5/ *Display (D)*
Read the frame buffer and show it on the screen. Simple task, but must be done fast.
1280*1024 (pixels) * 60 (/sec) * 3 = 236 Mbyte/s. "always" special-purpose

Because these are executed in order they are "always" implemented as
a pipeline of processes: user -> (1) -> (2) -> (3) -> (4) -> (5) -> screen

Processor choice:
1/ GTXS-D (1-4 general, 5 special) Xerox Alto, original Sun
2/ G-TXSD (DEC VAX/E&S PS-390) - only generation on host

Many new systems avoid these extremes.

Instead GTX-SD and GT-XSD are becoming more frequent.

SG uses GT-XSD architecture

Why not G-TXSD: immediate mode (increased performance, but hardwired traversal algorithms, predefined data structures, and clumsy modelling interaction). If we ignore generation this architecture is perfect: *demonstration* programs. Can display complex models fast but offer bad interaction)

PHIGS is a software version of G-TXSD hardware: can be implemented in GT-XSD
but an immediate-mode graphics graphics interface cannot be implemented on G-TXSD platform.

For *transformation* SG uses special processors:
demanding - 40 Mflops
if done in general-purpose this would be too special (for anything *but* graphics)
GT-XSD is poorly used when *not* graphics, GTX-SD poorly used *when* graphics
reduced flexibility (cannot be programmed, manufacturer provides microcode)

Application programmable traversal.

General CPU (1/ and 2/) (page 76)
Four/two parallel CPU's (the buses are important, asynchronous)
each a RISC processor/ FPU caches 2 cache memories. 20 MIPS (4 Mflops)
also hardware to support efficient transfer to transformation engine.
can transfer up to 400 000 position/normal pairs to the transformation engine.

Page 78: twice the load -> double speed.
But the other figure is more relevant: (4 proc/ 1 pgm)

Graphics subsystem
100 floating point operations for transform, light, clip, project, and map
400 000 per second, i.e. 40 Mflops (100K, 135K triangles per second)
Each geometry engine does one part of the process (pipeline)
1/ matrix and normal
2/ light
3/ clip testing
4/ perspective division (and clipping)
5/ viewport transformation, colour mapping, depth-cue

Scan conversion subsystem
Output from transformation: vertices in screen coordinates
scan conversion subsystem -> pixels
polygon scan conversion -> three processors
reduces to trapezoids left and right sides are vertical
edge processor: decomposes the trapezoids into vertical spans
span processors reduces to pixels
2 000 000 polygon spans per second
each span processor reduce spans to pixels at a rate of 16 000 000 pixels per second
i.e. 80 Mpixels per second Line pixels (from the edge processor: 16 Mpixels per second)

Raster subsystem
In the 5 parallel pipelines there are 20 image engines, each controlling 1/20 of the screen
Tiles as in fig 6 (page 81) - works in parallel even for small figures
5x4 (1280*1024)

Bitplane: 5 banks (96 bits per pixel): double buffer (2*32), z_buffer (24), overlay (4), id (4)
all pixels algorithms compare current contents with new; replace, z-buffer, alpha blend

Display subsystem
not much
five in parallel, assigned to one of the span processors.
Interpret the 64 + 8 bits
Interpret as a function of ID (i.e. window independent selection of buffer mode)
colour index also selected independently.

Image formats
Let's assume we've created a beautiful or informative image
Now we talk about how the images we create can be stored &transmitted

JPEG, TIFF, GIF, RLE
MPEG

interplane, interrow, interpixel (in general)
colour tables (24 vs. 8)

GIF: + widely used (across many different platforms)
64k x 64k images supported
reasonably simple to implement
multiple images supported

- only 256 colours (through a colour table)
no colour correction
only RGB, not CMYK/HSI

5 components: header
logical screen descriptor (size for image or images)
global colour table (palette of 24-bit triplets) (not mandatory)
graphics or special-purpose (comments) blocks
graphics: image descriptor, colour table, LZW-compressed bitmap data
trailer block (hex: 3b)

LZ: compression rate between 1:1 and 3:1
adaptive (starts with 256 values & "learns" patterns of values)

TIFF:

+ flexible
- complicated, unreliable between platforms, (can include undefined data)
many compression schemes

PBM:
A family of simple formats

+ very simple, many conversion routines
- bulky, Unix-only, binary version only 8-bit data

Header in ASCII: P1/P2/P3/P4/P5/P6
width
height
max (for P2,P3,P5,P6)

P1 - ASCII PBM P4 - binary PBM
P2 - ASCII PGM P5 - binary PGM

JPEG:

+ the future (and present?) standard
"by far the best" compression
- slow software (de)compression, still evolving

does not necessarily restore original exactly (but GIF only has 8 bits per pixel)
JPEG stores info about *changes* in luminance (in particular)/colour (cf. YIQ,TV)
Images that are nearly indistinguishable from original with 1 bit per pixel.
There are many compression schemes

The most common one:
1/ convert RGB -> luminance, chrominance (intensity, 2 colour change chann)

2/ DCT transform
3/ Quantization (this step makes JPEG lossy)

Decompression - opposite order (inserts 0's for the lost bits)

Colour scheme: YCbCr (sometimes YUV)
can use fewer Cb/Cr pixels (one each per four Y's)

DCT encoding intensity -> frequency (changes) (8x8 subimages)
F(0,0) - mean, F(0,1) slow changes in Y F(7.7) - fast changes in X and Y

Quantization
Linear (divide) - divide differently for different values (0,0) less than (7,7)

Stored as differences from neighbouring values in each 8x8

Compression (entropy coded)

Huffman coding:
binary (varying-length) code for each unique value
often 2 passes
problem: for instance

Converters:

PGMPLUS package to/from most formats
two conversions to convert from TIFF to GIF for instance

djpeg jpeg -> PBMPLUS (PPM,PGM), Targa, GIF, RLE
cjpeg jpeg <- PBMPLUS ()PPM,PGM, Targa, GIF, RLE