Energy Efficiency in Graphics Rendering

Preeti Ranjan Panda
Department of Computer Science and Engineering
Indian Institute of Technology Delhi

Presentation at TU Dortmund, June 2011
Graphics Power Consumption

Desktop computer

- Monitor: 56%
- Power Supply Loss: 22%
- VR: 1%
- Other: 7%
- HDD/DVD: 4%
- Graphics: 6%

Mobile computer

- Graphics: 14%
- HDD/DVD: 9%
- Cooling Fan: 4%
- Power Supply Loss: 7%
- Rest: 13%
- CPU: 7%
- Chipset: 13%
- 14.1' LCD: 33%

[Ref: PC Energy-Efficiency Trends and Technology, source: intel.com]

B.V.N. Silpa and P.R. Panda, 2011
Observation

- GPU/Graphics rendering power is significant (greater than CPU)
- Yet, very little research on GPU energy efficiency!
  - GPU performance was/is primary
  - Proprietary GPU architectures

B.V. N. Silpa and P. R. Panda, 2011
Graphics Pipeline

From CPU

Command processor

Receives vertices and commands from CPU

Vertex processor

Transform vertices to screen space and Light

Clipping

Delete unseen part of scene

Setup and Rasterize

Generate pixels

Fragment Processor

Pixel Coloring and Z-test

Image Composition

Blend with Frame Buffer

Display

Texture

B.V. N. Silpa and P. R. Panda, 2011
Adding Energy Efficiency

Component Level: TEXTURE MAPPING

- From CPU
- Command processor: Receives vertices and commands from CPU
- Vertex processor: Transforms vertices to screen space and Light
- Clipping: Deletes unseen part of scene
- Setup and Rasterize: Generates pixels
- Fragment Processor: Pixel Coloring and Z-test
- Image Composition: Blend with Frame Buffer
- Display

System Level: DVFS

B.V.N. Silpa and P.R. Panda, 2011
LOW POWER
TEXTURE MAPPING
[ICCAD’08]

B.V. N. Silpa and P. R. Panda, 2011
B. V. N. Silpa and P. R. Panda, 2011

Texture memory consumes 30-40% of total power.
Texture Mapping

- Add detail and surface texture to an object.
- Reduces the modeling effort for the programmer.

B.V. N. Silpa and P. R. Panda, 2011
Texture Filtering

- Texture space and object space could be at arbitrary angles to each other

- Nearest neighbor

- Bilinear interpolation: weighted average of four texels nearest to the pixel center.
Texture Access Pattern

- Texture mapping exhibits high spatial and temporal locality
  - Bilinear filtering requires 4 neighbouring texels
  - Neighbouring pixels map to spatially local texels
  - Repetitive textures

B.V. N. Silpa and P. R. Panda, 2011
Blocking and Texture Cache

- **Blocked Representation**
  - Texels stored as 4x4 blocks
  - Reduces dependency on texture orientation, and exploits spatial locality

- Texture memory accessed through a Cache hierarchy ("TEXTURE CACHE")

- Familiar architectural space

- BUT, application knowledge could help improve the HW over a “standard cache”

B.V. N. Silpa and P. R. Panda, 2011
Predictability in Texture Accesses

- Access to first texel gives information about access to the next 3 texels
- The four texels could be mapped to either one, two or four neighbouring blocks.

B.V.N. Silpa and P.R. Panda, 2011
Low Power Texture Memory Architecture

- Lower power memory architecture than Cache for texturing
  - Use a few registers to filter accesses to blocks expected to be reused
  - Access stream has predictability - controlled access mechanism reduces tag lookups

B.V. N. Silpa and P. R. Panda, 2011
How many blocks to buffer?

- Need to buffer up to 4 blocks

- A buffer is a set of 4x4 registers, each 32 bit
- Texture Buffer Array is a group of 4 such buffers
Texture Lookup

- **Case 1:**
  - Lookup (block0)
    - Get the 4 texels from the block using offsets
    - **SAVING: 3 LOOKUPS**

- **Cases 2 & 3:**
  - Lookup (block 0)
    - Get texel0 and texel1 from this block
  - Lookup (block 2)
    - Get texel2 and texel3 from this block
    - **SAVING: 2 LOOKUPS**

B.V. N. Silpa and P. R. Panda, 2011
Contd..

Case 4:

- Lookup all 4 blocks and get the texels from the respective blocks using offsets

Power Savings from:

Reduced Tag lookups
Smaller buffer than cache

B.V. N. Silpa and P. R. Panda, 2011
### Distribution of access among various cases

<table>
<thead>
<tr>
<th>Case</th>
<th>Access Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>60%</td>
</tr>
<tr>
<td>Case 2</td>
<td>20%</td>
</tr>
<tr>
<td>Case 3</td>
<td>10%</td>
</tr>
<tr>
<td>Case 4</td>
<td>10%</td>
</tr>
</tbody>
</table>

Number of comparisons per access is 1.38 instead of 4

B.V.N. Silpa and P.R. Panda, 2011
Architecture of Texture Filter Memory
Hit Rate into TFM

TFM gives 4.5% better hit rate than a direct mapped filter of the same size

B.V. N. Silpa and P. R. Panda, 2011
Energy per Access

TFM consumes 75% lesser energy than the conventional Texture cache
Texture Filter Memory Summary

- In addition to high spatial locality, texture mapping access pattern also has predictability.
- Replaced high energy cache lookups with low energy register buffer reads.
- TFM consumes ~75% lesser energy than conventional texture mapping system.
- Overheads:
  - TFM access 4x faster than cache access.
  - 0.48% area overhead over texture cache subsystem.

B.V. N. Silpa and P. R. Panda, 2011
DYNAMIC VOLTAGE AND FREQUENCY SCALING (DVFS)

[CODES+ISSS’10]

B.V.N. Silpa and P.R. Panda, 2011
Tiled Graphics Rendering

Vertex Shader → Primitive Assembly → Clipping → Setup & Rasterize → Pixel Shading → Raster Operations

Geometry Pipeline

Geometry Processing → Tiling → Pixel Processing

Bin 1, Bin 2, Bin 3, Bin 4

B.V.N. Silpa and P.R. Panda, 2011
Workload of games

Different games have significant but gradual workload variation within a game

B.V.N. Silpa and P.R. Panda, 2011
Spatial Correlation in frames

Continuity of motion leads to frame level spatial correlation, resulting in slow workload variation

B.V. N. Silpa and P. R. Panda, 2011
Temporal Correlation of Tile Workload

Many tiles are correlated, even if workloads of consecutive frames differ

- 80% tiles within 10% diff
Dynamic Voltage and Frequency Scaling

Predicted Workload – run Tiles-1,2 at V/2

V/2

#1 #2

T 2T

#1 over-predicted

#1 under-predicted

V/2

#1 #2

V/3

T/2 2T

=> slow down #2

V

#1 #2

V/2

3T/2 2T

=> speed up #2

Continuously track and take corrective action after rendering each tile

B.V. N. Silpa and P. R. Panda, 2011
Frame Rank ($R_G, R_p, R_t, R_r$)

- Vertex processing workload of a primitive of $V$ vertices using a Shader $N_v$ instructions long
  - Shader workload $\sim V \times N_v$
  - Clipping and Binning $\sim V$

$$R_g = \sum_{Batches} \text{VertexCount} \times \text{VertexShaderLength} + \text{PrimitiveCount}$$

- Pixel shading workload
  - Number of pixels per primitive $\sim$ Area of bounding box of the primitive

$$R_p = \sum_{Batches} \sum_{Primitives} \text{PrimitiveArea} \times \text{PixelShaderLength}$$

B.V. N. Silpa and P. R. Panda, 2011
Frame Rank \( (R_G, R_p, R_t, R_r) \)

- Texture mapping workload
  - Texture footprint – number of texels to be filtered per pixel

\[
R_r = \sum_{\text{Batches}} \sum_{\text{Primitives}} \text{PrimitiveArea} \times \text{TextureCount} \times \text{TextureFootPrint}
\]

- Raster operations workload
  - Each raster operation results in a read and write to frame buffer

\[
R_r = \sum_{\text{Batches}} \sum_{\text{Primitives}} 2 \times \text{PrimitiveArea} \times \text{RasterOps}
\]

B.V.N. Silpa and P.R. Panda, 2011
Tile Rank \((T_p, T_t, T_r)\)

- Tile rank computation is similar to frame rank computation
- Pixel count is computed as overlap area of the bounding box and the tile.
Rank Based DVFS Scheme

- Divide the tiles into set of Heavy tiles (Tile rank in current frame greater than its rank in previous frame) and Light tiles.

Frame_Rank (current) > Frame_Rank (previous) ?

Yes

Process Heavy tiles at Frequency F=F_{Max}

No

Process Heavy tiles at frequency determined by frame history

Use tile history based scheme for light tiles

B. V. N. Silpa and P. R. Panda, 2011
Tile Rank Based DVFS Summary

- Tile Rank Based DVFS gives 75% better performance than history based scheme.
- Energy/FrameRate minimum for Tile Rank based DVFS scheme.
- Overheads:
  - < 0.01% computation
  - < 0.01% storage

B.V. N. Silpa and P. R. Panda, 2011
Future Work

- Extension to multi-core GPUs
- Other stages of the graphics pipeline
Thank You!