Real-Time Graphics Architecture

Kurt Akeley
Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall

Texture
Topics

1. Review of texture mapping
2. RealityEngine and InfiniteReality
3. Texture caching
4. Texture prefetching
5. Trends and pitfalls

Readings

Required
1. Z. Hakura, A. Gupta, The design and analysis of a cache architecture for texture mapping
2. H. Igehy, M. Eldridge, K. Proudfoot, Prefetching in a texture cache architecture

Background
1. P. Heckbert, Texture mapping polygons in perspective
2. P. Heckbert and H. Moreton, Interpolation for polygon texture mapping and shading
3. J. Blinn, Hyperbolic interpolation
4. L Williams, Pyramidal parametrics
Texture Mapping

2D (3D) Texture Space
  ↓ Texture Transformation
2D Object Parameters
  ↓ Parameterization
3D Object Space
  ↓ Model Transformation
3D World Space
  ↓ Viewing Transformation
3D Camera Space
  ↓ Projection
2D Image Space

Texture Mapping Polygons

Forward transformation: linear projective map

\[
\begin{bmatrix}
x \\
y \\
w \\
\end{bmatrix} =
\begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i \\
\end{bmatrix}
\begin{bmatrix}
s \\
t \\
r \\
\end{bmatrix}
\]

Backward transformation: linear projective map

\[
\begin{bmatrix}
s \\
t \\
r \\
\end{bmatrix} =
\begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i \\
\end{bmatrix}^{-1}
\begin{bmatrix}
x \\
y \\
w \\
\end{bmatrix}
\]
Perspective-Correct Interpolation

\[
\begin{bmatrix}
x, y, z, 1, r, g, b, a, s, t, r, 1, \\
\end{bmatrix}
\]

↓ Transform & Clip

\[
\begin{bmatrix}
xw, yw, zw, w, r, g, b, a, sq, tq, rq, q, \\
\end{bmatrix}
\]

↓ Project (/w)

\[
\begin{bmatrix}
xw', yw', zw', 1, r, g, b, a, sw', tw', rw', qw', \\
\end{bmatrix}
\]

↓ Rasterize and Interpolate

\[
\begin{bmatrix}
xw', yw', zw', w', r, g, b, a, sw', tw', rw', qw', \\
\end{bmatrix}
\]

↓ Texture Project (/qw')

\[
\begin{bmatrix}
xw', yw', zw', w', r, g, b, a, sw' / qw', tw' / qw', rw' / qw', 1, \\
\end{bmatrix}
\]

Linear Perspective

Correct Linear Perspective

Incorrect Perspective

Linear Interpolation, Bad Perspective Interpolation, Good
Filtering Textures

- Texture footprint
  - Footprint changes from pixel to pixel
    - i.e. not shift-invariant
- Resampling theory: two cases
  1. Magnification => Interpolation
  2. Minification => Filter (averaging)

MipMaps - L. Williams

*Multum In Parvo = Many things in a small place*

Address: 7 CMP(3 FIX + 4 RANGE), 2 MUL, 2 ADD
Texture Filtering

Constant time filtering

Linear (LERP)
1 MUL + 2 ADD / comp

Bilinear
3 LERPs
3 MUL + 6 ADD / comp

Trilinear
7 LERPs
7 MUL + 14 ADD / comp

Quadrilinear
15 LERPs
15 MUL + 30 ADD / comp

\[ lerp(t, v_1, v_2) = v_1 + t(v_2 - v_1) \]
**Principle of Texture Thrift**

Given a scene consisting of 3D textured surfaces, the amount of texture information minimally required to render an image of the scene is proportional to the resolution of the image and is independent of the number of surfaces and the size of the textures.

\[ T = d \cdot t \cdot l \]

- **d** - depth complexity
- **t** - average number of textures per surface

D. Peachey, Texture on demand, PIXAR Technical Memo, 1990

**Mipmaps**

1. Constant time to filter a textured fragment
2. Output sensitive algorithm

**Derivatives**

\[
\begin{align*}
\frac{\partial s}{\partial x} &= s(x+1, y) - s(x, y) \\
\frac{\partial s}{\partial y} &= s(x, y+1) - s(x, y) \\
\frac{\partial t}{\partial x} &= t(x+1, y) - t(x, y) \\
\frac{\partial t}{\partial y} &= t(x, y+1) - t(x, y)
\end{align*}
\]
mipd

Approximates quadrilateral with a square

\[ A = \begin{vmatrix} \frac{\partial s}{\partial x} & \frac{\partial t}{\partial x} \\ \frac{\partial s}{\partial y} & \frac{\partial t}{\partial y} \end{vmatrix} \]

\[ d = \sqrt{A} \]

\[ \text{mipd} = \log_2 d \]

Common formula: \( A \approx \max \left( \sqrt{\frac{\partial s^2}{\partial x} + \frac{\partial t^2}{\partial x}}, \sqrt{\frac{\partial s^2}{\partial y} + \frac{\partial t^2}{\partial y}} \right) \)

[Heckbert]

---

Compute & Bandwidth Requirements

<table>
<thead>
<tr>
<th></th>
<th>ADD</th>
<th>MUL</th>
<th>CMP</th>
<th>DIV</th>
<th>SPE</th>
<th>READ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>4</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOD</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Filter</td>
<td>10</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>18</td>
<td>24</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Texture access random (sort texels to fragments)

Requires high bandwidth

Address computation and filtering arithmetic intensive

Performance goal: 1 billion fragments per second

Most demanding stage of the graphics pipeline
History

Flight simulators
- GE Apollo Simulator 1963
- Clever method for procedurally generating textures

Workstations
- SGI RE and IR and others …

Single-chip PC
- 3DFX and Nvidia and others …

RealityEngine (3rd Generation)
RE Fragment Generator

Capacity: 16 MB = 8 MT (>1024^2 mipmap)
Texture replicated per fragment generator (5,10,20)x16MB
Fill Rate: 12 MT/s x (5,10,20) = (60,120,240)

InfiniteReality (3rd Generation)
InfiniteReality

- 2x2 fragment quads (TA)
- 4x8 texture memories (TM)
- 2x2 texture filterers (TF)
- 32 by 80 crossbar TF->IE

Capacity: (16 MB, 64, 256)
Fill: 200 MT/s x (1, 2, 4)
Memory Access

High bandwidth required
- 1 GF/s $\Rightarrow$ 32 GB/s texture read (8 16-bit texels)

Mip map accesses
- Small granularity when interleaving

Memories
- Large granularity
- High latency

Solutions

Caching
- Reduce the bandwidth requirement
- Match granularity of accesses to memory

Prefetching
- Hide the high latency of memory accesses
- Handle highly variable latency

Compression (not covered)
Unique T/F Ratio

Key statistic: unique texel to fragment ratio

- Average memory bandwidth required

Definition:

Total texels accessed / Total fragments generated

Texture Locality

<table>
<thead>
<tr>
<th></th>
<th>Percent trilinear</th>
<th>Unique T/F</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>quake</td>
<td>30%</td>
<td>0.033</td>
<td>quake2x</td>
</tr>
<tr>
<td>quake2x</td>
<td>47%</td>
<td>0.092</td>
<td>flight2x</td>
</tr>
<tr>
<td>flight</td>
<td>62%</td>
<td>0.706</td>
<td>qtvr</td>
</tr>
<tr>
<td>flight2x</td>
<td>87%</td>
<td>1.554</td>
<td>qtvr2x</td>
</tr>
<tr>
<td>qtvr</td>
<td>0%</td>
<td>0.569</td>
<td></td>
</tr>
<tr>
<td>qtvr2x</td>
<td>100%</td>
<td>2.832</td>
<td></td>
</tr>
</tbody>
</table>
Texture Locality Measures

Texture locality variables
- Small repeated textures
- Average magnification textures
- Percentage of magnified textures
- Level of detail bias
- Average minification when mipmapping
  \[ \text{Frac}(d) \approx 0 \Rightarrow T/F \approx 1.25 \]
  \[ \text{Frac}(d) \approx 1 \Rightarrow T/F \approx 5 \]

Texture Caching

Cache parameters
- Line size (blocking)
- Cache size (working set)
- Direct-mapped or associative

Representation of textures in memory
Rasterization order
Quake Miss Rate

Flight Miss Rate
## QTVR Miss Rate

### Bandwidth Savings

2 8 KB DM-caches
4x4 32-bit T blocks
Texture Blocking

2D blocks
Hide orientation effects

Texture Map

Texture Blocking

6D Organization Cache Size Cache Line Size
4x4 blocks 4x4 texels

(s1,t1) (s2,t2) (s2,t2)
Address base s1 t1 s2 t2 s3 t3
**Rasterization Order**

Scanline Order  
Tile Order

**Tiling and Blocking Results**

32KB, 2-way Associative, 128 byte lines

Linear  
Blocked Textures

Tiled Rasterization  
Blocked and Tiled

Misses
- 1
- 2
- 3+
Summary: Texture Caching

- Reasonably small workings sets
  16-32 KB caches has 95% hit rate
- Separate caches for even and odd mip-levels to prevent conflicts
  Alternatively 2-way associative cache
- Blocked textures further reduces miss rate
- Tiled rasterization further reduces miss rate
- 6D tiling minimizes working set

Conclusion

Caches highly effective for reducing texture memory bandwidth (roughly 5-10:1)
Disadvantages

1. Prefetches may generate conflict misses
2. Cache tags accessed twice
3. Large, fully associative fetch buffer
Texture Prefetching Architecture

Advantages

1. No conflicts caused by prefetching
2. Cache tags accessed only once
3. No fully associative fetch buffer
4. Reorder buffer tolerates out-of-order memory replies
## Memory Models

200 Mpixel fragment generator
5 ns cycle time

<table>
<thead>
<tr>
<th>Bandwidth (texels/cycle)</th>
<th>Latency (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>agp</strong> 1</td>
<td><img src="#" alt="Graph" /></td>
</tr>
<tr>
<td><strong>rdram</strong> 2</td>
<td><img src="#" alt="Graph" /></td>
</tr>
<tr>
<td><strong>rdram2x</strong> 4</td>
<td><img src="#" alt="Graph" /></td>
</tr>
<tr>
<td><strong>numa</strong> 4</td>
<td><img src="#" alt="Graph" /></td>
</tr>
</tbody>
</table>

97% stall free unless limited by memory bandwidth
Buffering requirements modest compared to cache size
Summary: Texture Prefetching

Prefetching effectively hides memory latency
- Early calculation of texture coordinates

Tolerating latency
- FIFO implements “context switch”

Trends and Pitfalls

Trends
- Programmable texture coordinate generation
- Multitexture (4) and dependent textures (1)
- Programmable texture combination
- Better quality filters

Pitfalls
- 2D texture memory allocation; use 1D!
- Texture thrashing (draw in texture order)
- Handling borders is complicated
- Precision: very large textures (swimming)
Additional Topics

CATS and RATS; RIP-MAPS
Anisotropic filtering
Detail textures
Texture compression
Texture management; clip-maps
Parallel texture caching