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

$\downarrow$ Texture Transformation

2D Object Parameters

$\downarrow$ Parameterization

3D Object Space

$\downarrow$ Model Transformation

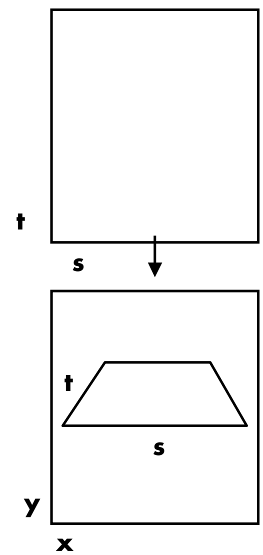
3D World Space

$\downarrow$ Viewing Transformation

3D Camera Space

$\downarrow$ 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

$$[x, y, z, 1, r, g, b, a, s, t, r, 1, \quad]$$

↓ **Transform & Clip**

$$[xw, yw, zw, w, r, g, b, a, sq, tq, rq, q, \quad]$$

↓ **Project (/*w*)**

$$[xw', yw', zw', 1, r, g, b, a, sw', tw', rw', qw', \quad]$$

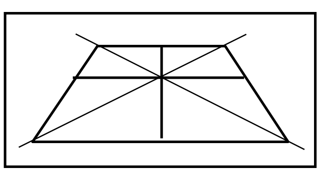
↓ **Rasterize and Interpolate**

$$[xw', yw', zw', w', r, g, b, a, sw', tw', rw', qw', \quad]$$

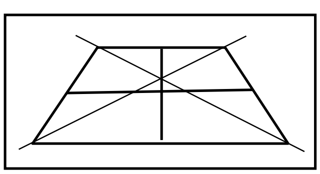
↓ **Texture Project (/*qw'*)**

$$[xw', yw', zw', w', r, g, b, a, sw'/qw', tw'/qw', rw'/qw', 1, \quad]$$

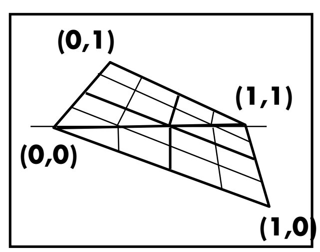
Linear Perspective



Correct Linear Perspective

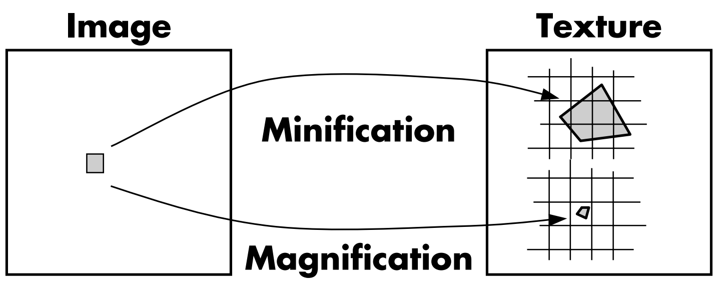


Incorrect Perspective



Linear Interpolation, *Bad*
Perspective Interpolation, *Good*

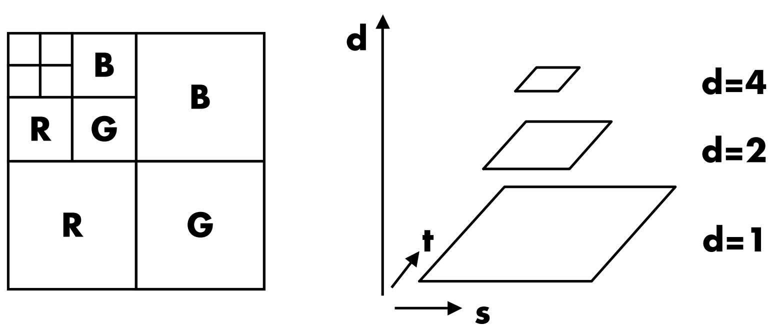
Filtering Textures



- 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

***M**ultum **I**n **P**arvo = Many things in a small place*

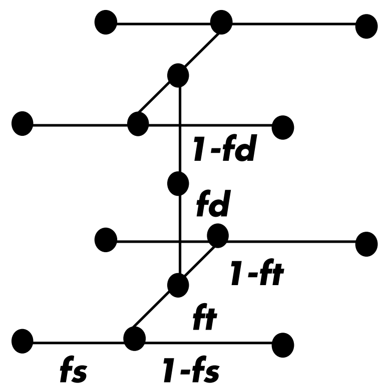


Address: 7 CMP(3 FIX + 4 RANGE), 2 MUL, 2 ADD

s

Texture Filtering

Constant time filtering



$$lerp(t, v_1, v_2) = v_1 + t(v_2 - v_1)$$

Linear (LERP)
- 1 MUL + 2 ADD / comp

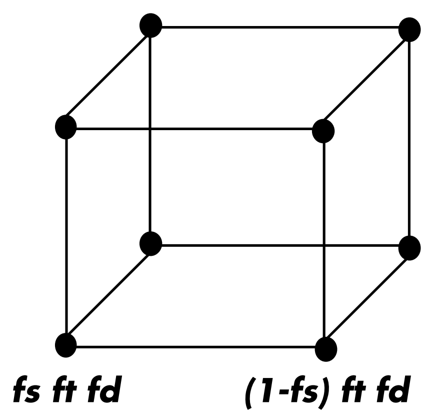
Bilinear
- 3 LERPs
- 3 MUL + 6 ADD / comp

Trilinear
- 7 LERPs
- 7 MUL + 14 ADD / comp

Quadrilinear
- 15 LERPs
- 15 MUL + 30 ADD / comp

Texture Filtering

Constant time filtering



Linear (LERP)
- 1 ADD
- 2 MUL / comp

Bilinear
- 4 MUL + 2 ADD
- 4 MUL / comp

Trilinear
- 12 MUL + 3 ADD
- 8 MUL / comp

Quadrilinear
- 28 MUL + 4 ADD
- 16 MUL / comp

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 t I

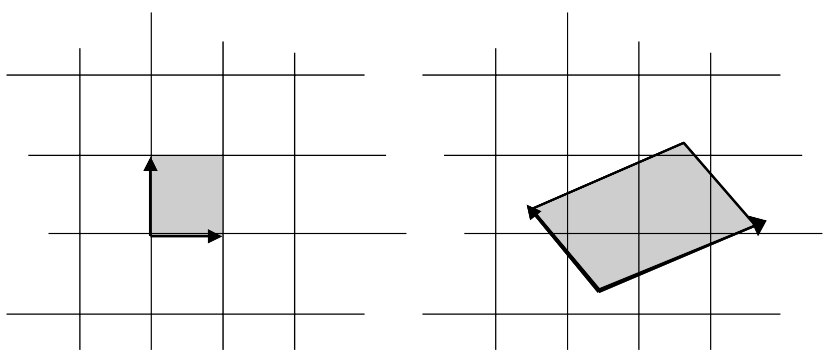
- d – depth complexity
- t – average number of textures per surface

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

Mipmaps

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

Derivatives



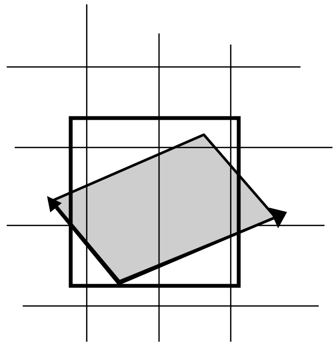
$$\frac{\partial s}{\partial x} = s(x+1, y) - s(x, y) \qquad \frac{\partial t}{\partial x} = t(x+1, y) - t(x, y)$$

$$\frac{\partial s}{\partial y} = s(x, y+1) - s(x, y) \qquad \frac{\partial t}{\partial y} = t(x, y+1) - t(x, y)$$

mipd

Approximates quadrilateral with a square

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



$$d = \sqrt{A}$$

$$mipd = \log_2 d$$

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

[Heckbert]

Compute & Bandwidth Requirements

	ADD	MUL	CMP	DIV	SPE	READ
Project		4		1		
LOD	6	4	1		3	
Address	2	2	7			8
Filter	10	14				
Total	18	24	8	1	3	8

Texture access random (sort texels to fragments)

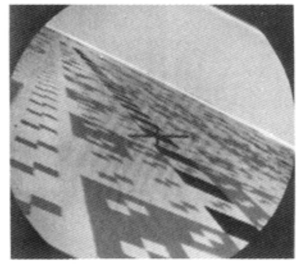
Requires high bandwidth

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

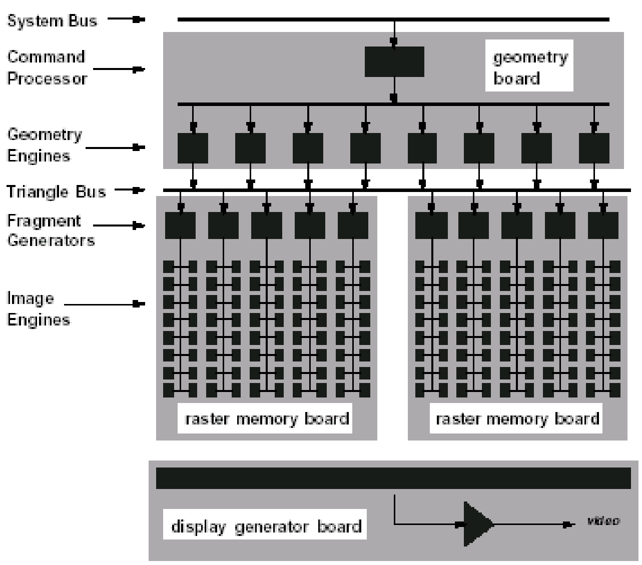
Workstations

- SGI RE and IR and others …

Single-chip PC

- 3DFX and Nvidia and others …

RealityEngine (3rd Generation)

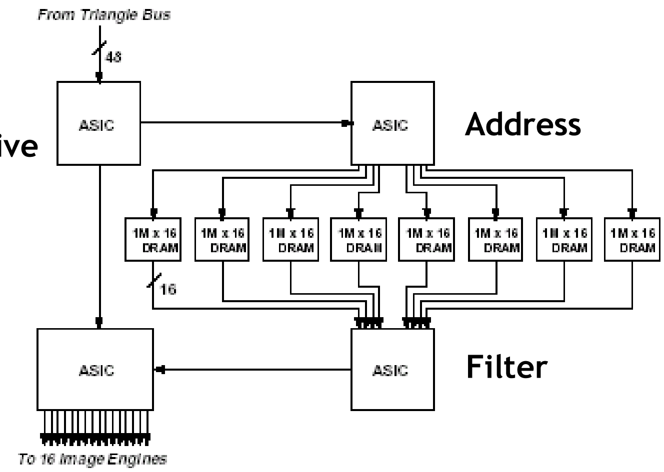


RE Fragment Generator

Project Derivative

Address

Filter

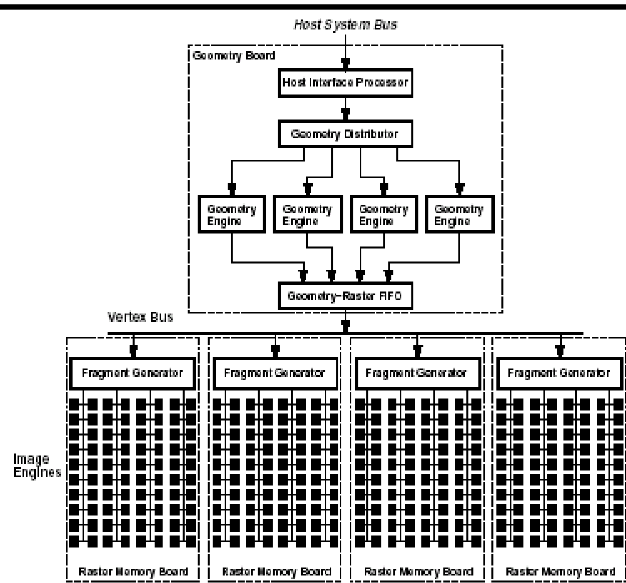


Capacity: 16 MB = 8 MT (>1024^2 mipmap)

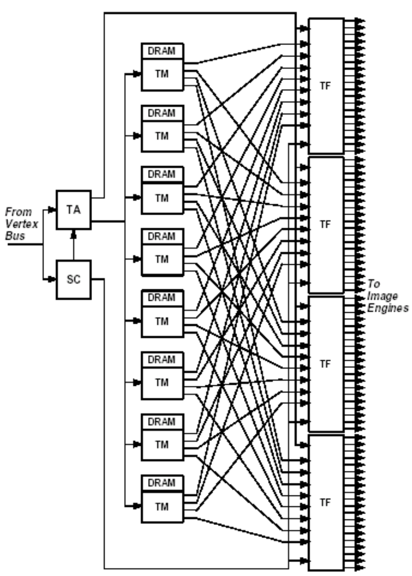
Texture replicated per fragment generator (5,10,20)x16MB

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

	Percent trilinear	Unique T/F	Image
quake	30%	0.033	
quake2x	47%	0.092	
flight	62%	0.706	
flight2x	87%	1.554	
qtvr	0%	0.569	
qtvr2x	100%	2.832	

Texture Locality Measures

Texture locality variables

- Small repeated textures
- Average magnification textures
- Percentage of magnified textures
- Level of detail bias
- Average minification when mipmapping

  Frac(d)~0 $\Rightarrow$ T/F ~1.25

  Frac(d)~1 $\Rightarrow$ T/F ~5

Texture Caching

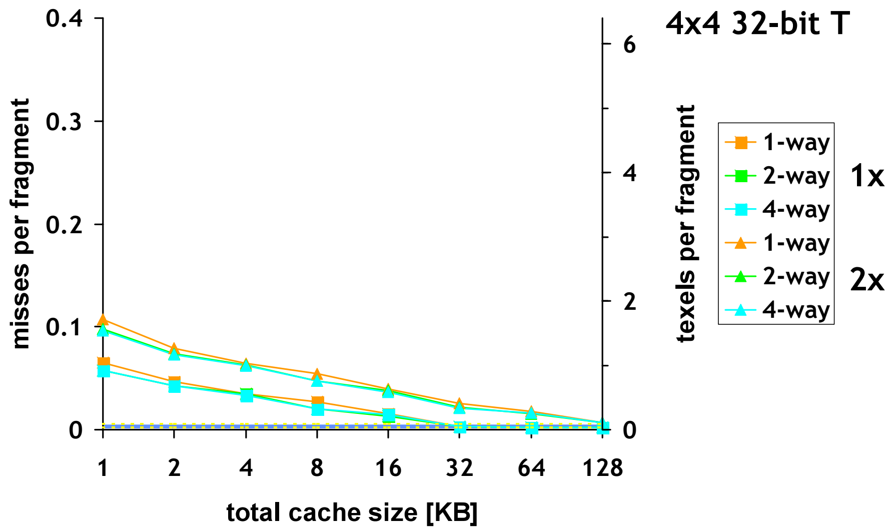
Cache parameters

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

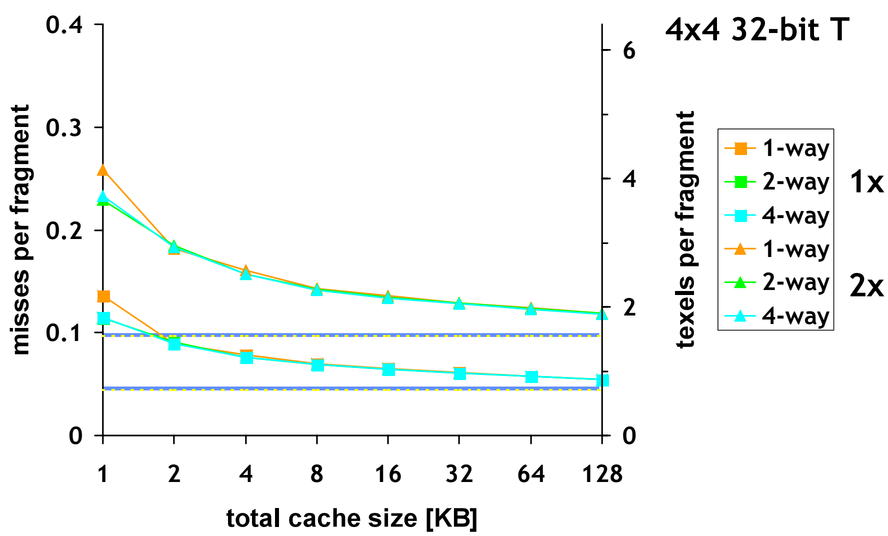
Representation of textures in memory

Rasterization order

Quake Miss Rate

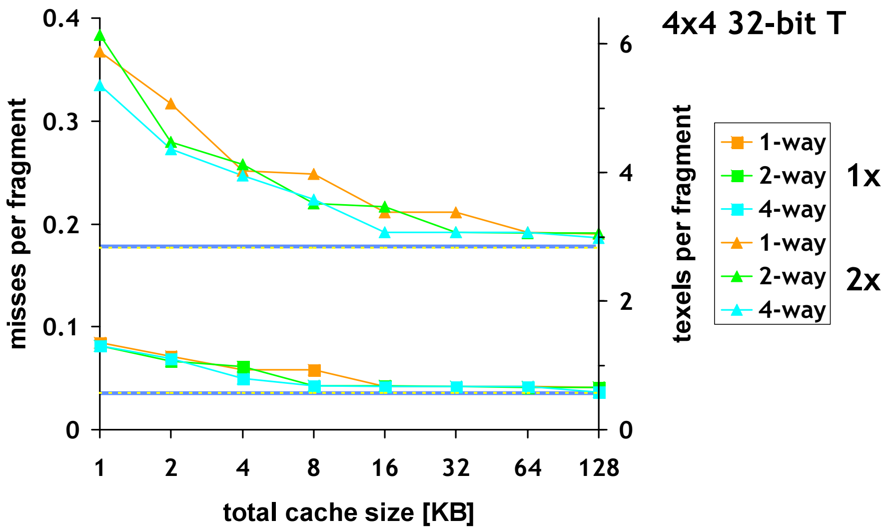


Flight Miss Rate

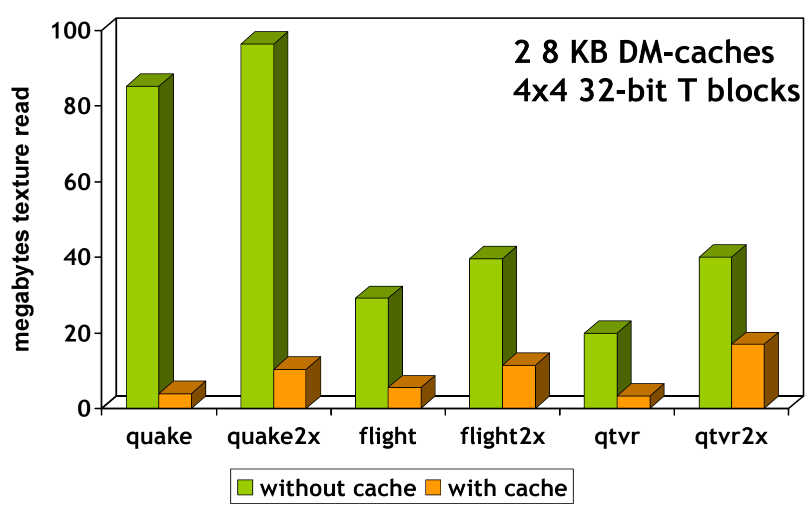


CS448 Lecture 7 Kurt Akeley, Pat Hanrahan, Fall 2001

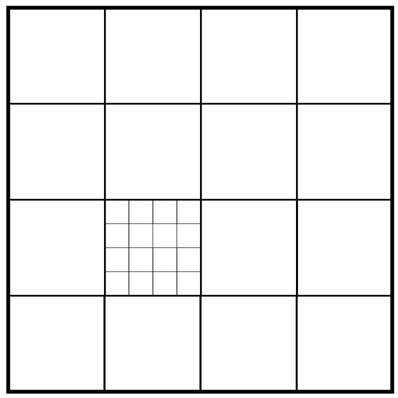
QTVR Miss Rate



Bandwidth Savings



Texture Blocking

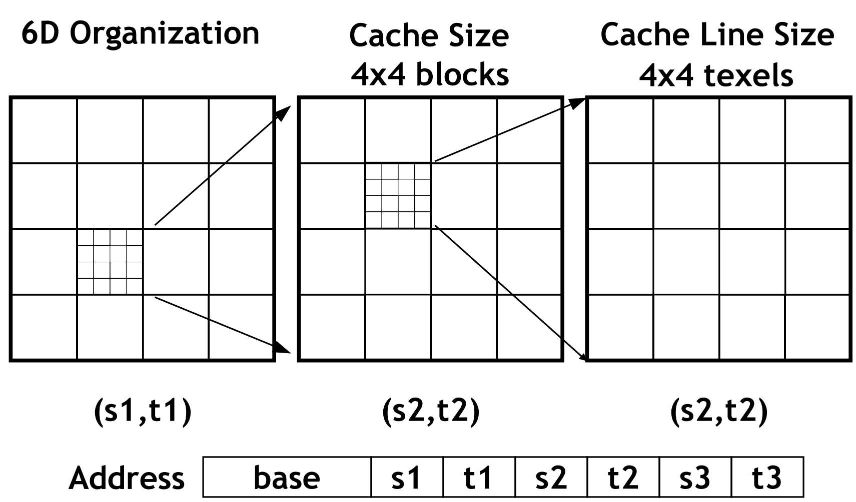


2D blocks

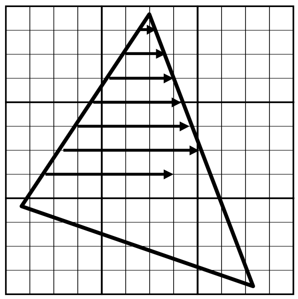
Hide orientation effects

Texture Map

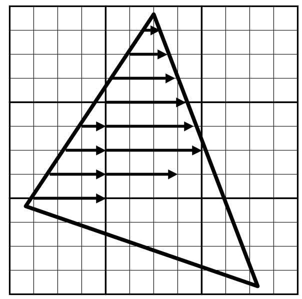
Texture Blocking



Rasterization Order



Scanline Order

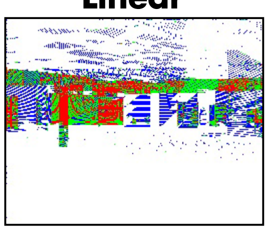


Tile Order

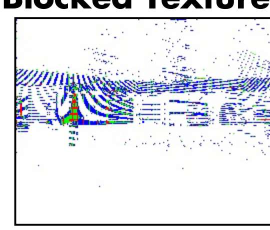
Tiling and Blocking Results

32KB, 2-way Associative, 128 byte lines

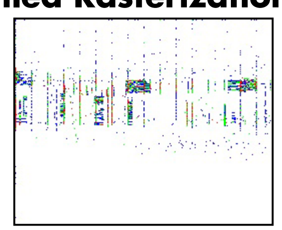
Linear



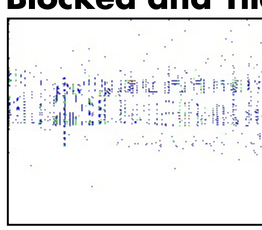
Blocked Textures



Tiled Rasterization



Blocked and Tiled



Misses

- 1
- 2
- 3+

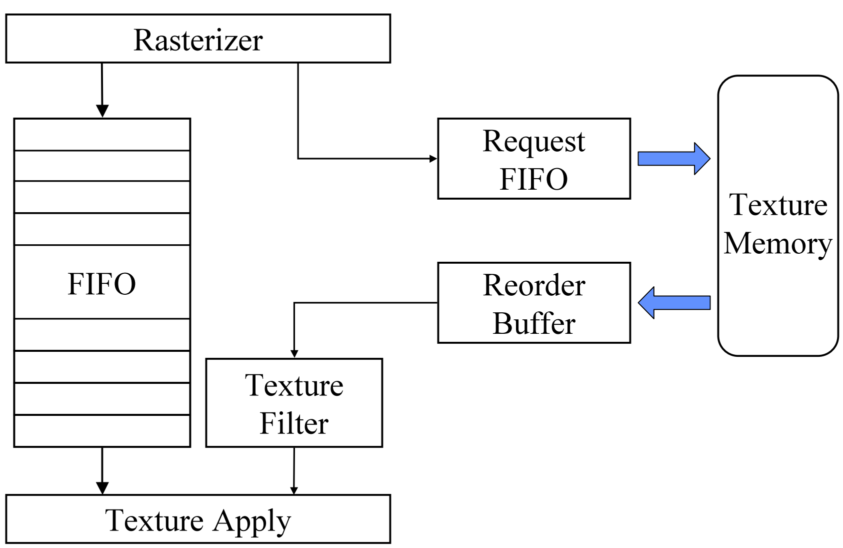
Summary: Texture Caching

- 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

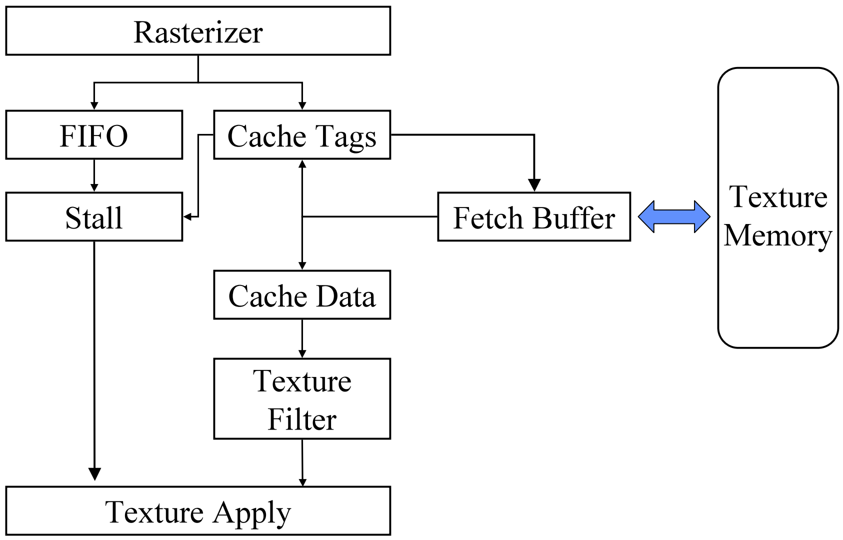
Conclusion

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

Texture Prefetching Architecture



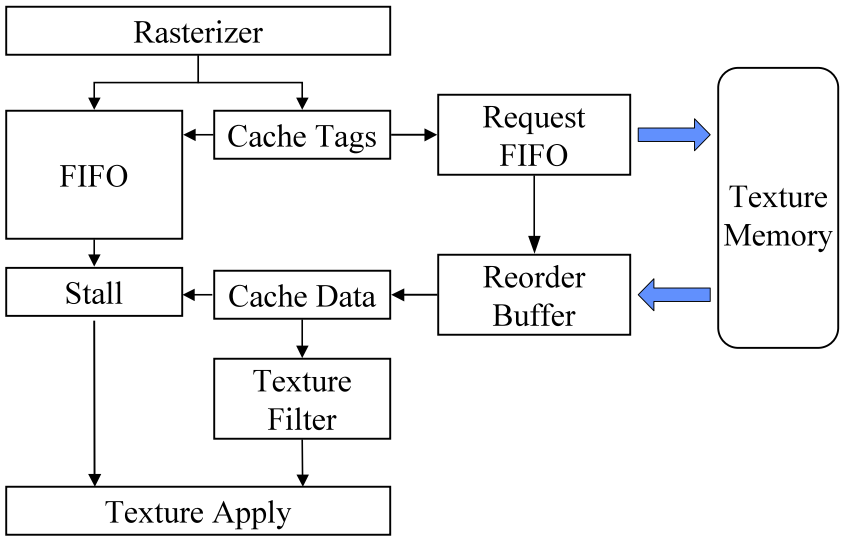
"Microprocessor" Architecture



Disadvantages

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

Texture Prefetching Architecture



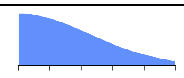
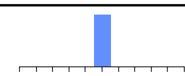
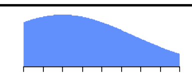
Advantages

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

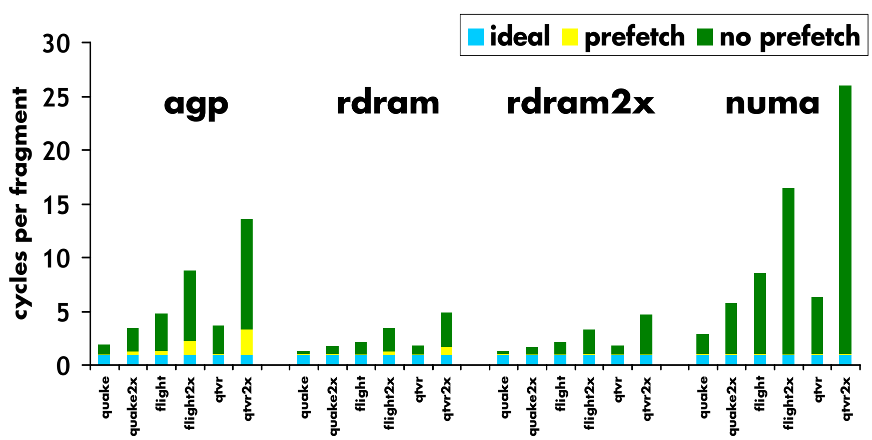
Memory Models

200 Mpixel fragment generator

5 ns cycle time

	Bandwidth (texels/cycle)	Latency (cycles)
agp	1	
rdram	2	
rdram2x	4	
numa	4	

Prefetching Results



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