The F-Buffer:
A Rasterization-Order FIFO Buffer for Multi-Pass Rendering

Bill Mark and Kekoa Proudfoot

Stanford University

http://graphics.stanford.edu/projects/shading/

HWWS 2001
© 2001, WRM

Motivation for this work

Two goals for real-time procedural shading:

- Hardware independence
- Support arbitrarily complex shaders

HWWS 2001
© 2001, WRM

Must virtualize HW resources

HW resources include:

- Functional units
  - Instructions
  - Texture units
  - Interpolators
- Memory
  - Vertex & fragment registers

CPU Analogy

- Virtual Memory system virtualizes DRAM
- Perfect virtualization not necessary

HWWS 2001
© 2001, WRM

Current approaches to virtualization

- Virtualize functional units
  - Examples:
    - Most multi-texture HW
    - 8 combiners @ 25% fill rate on NV20
  - Implemented with extra HW state
  - Memory virtualization is still a problem

HWWS 2001
© 2001, WRM

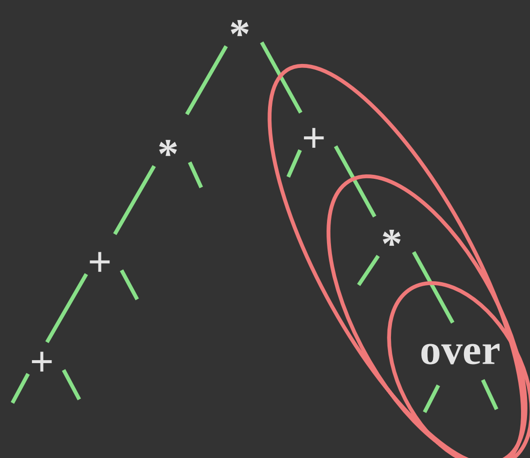
Current approaches to virtualization

- Multi-pass rendering
  - Render several times from one viewpoint
  - Each rendering pass performs one part of complete computation
  - Framebuffer can store one temporary RGBA value.
  - Use render-to-texture to store more temporary values.

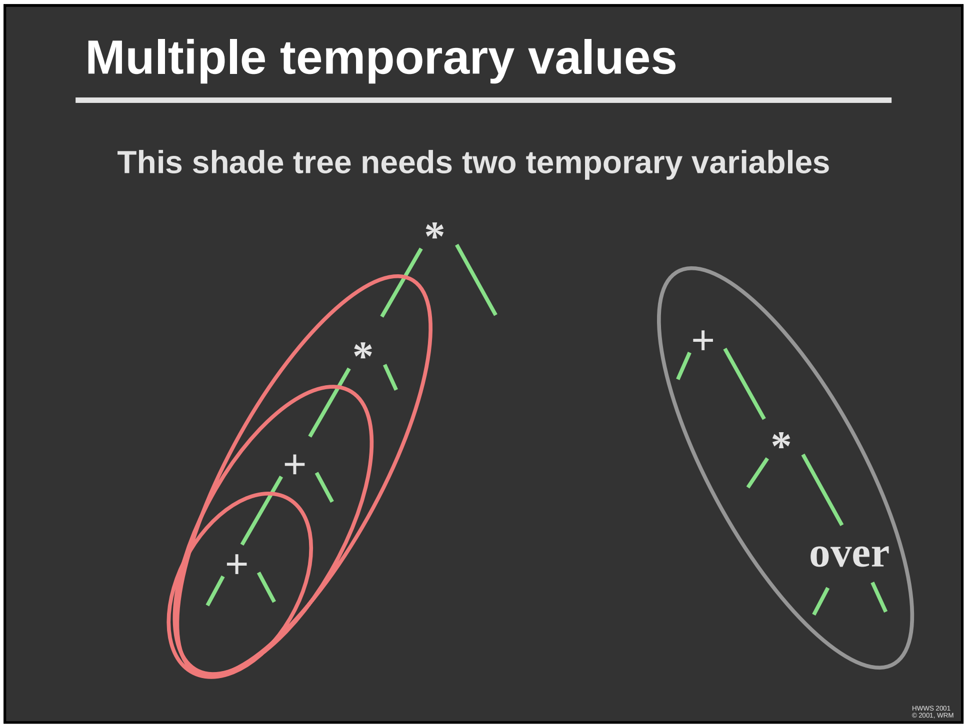
HWWS 2001
© 2001, WRM

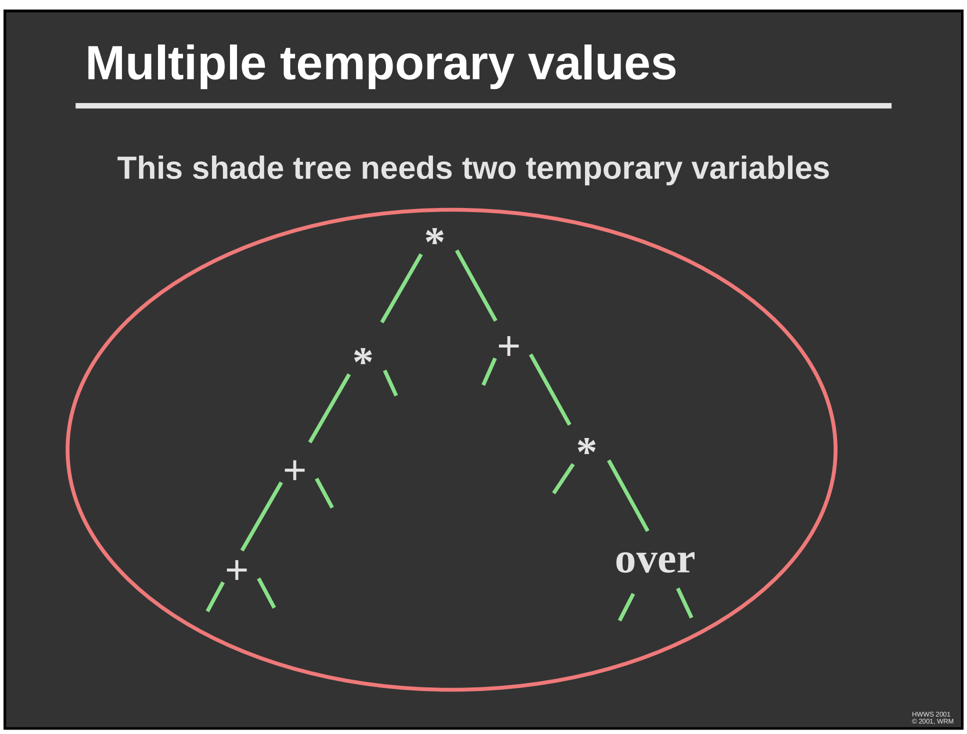
Multiple temporary values

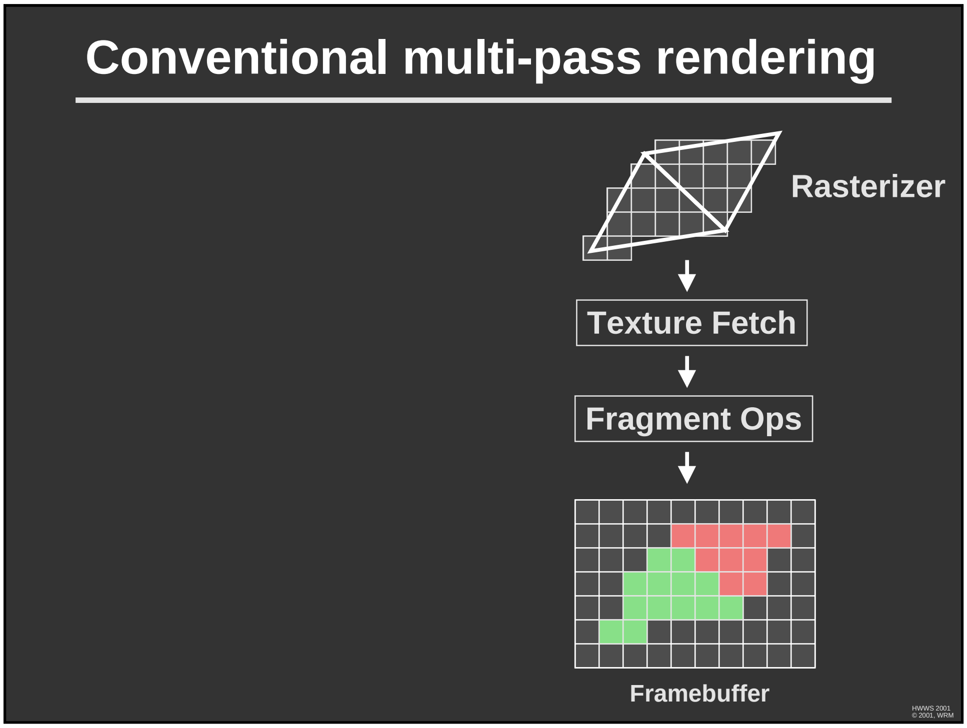
This shade tree needs two temporary variables

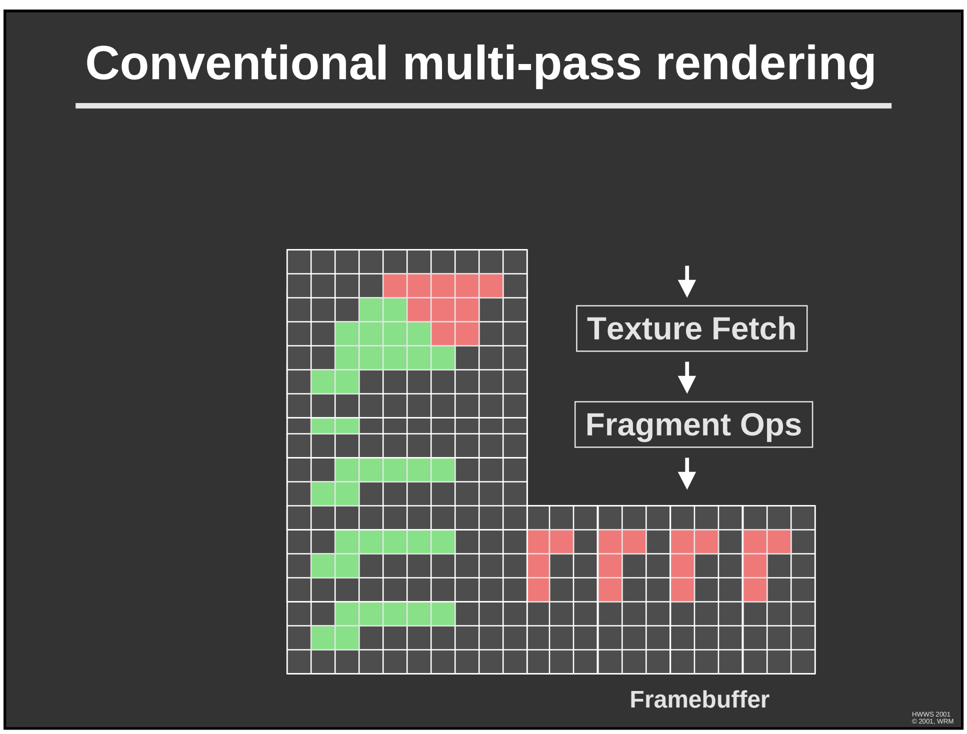


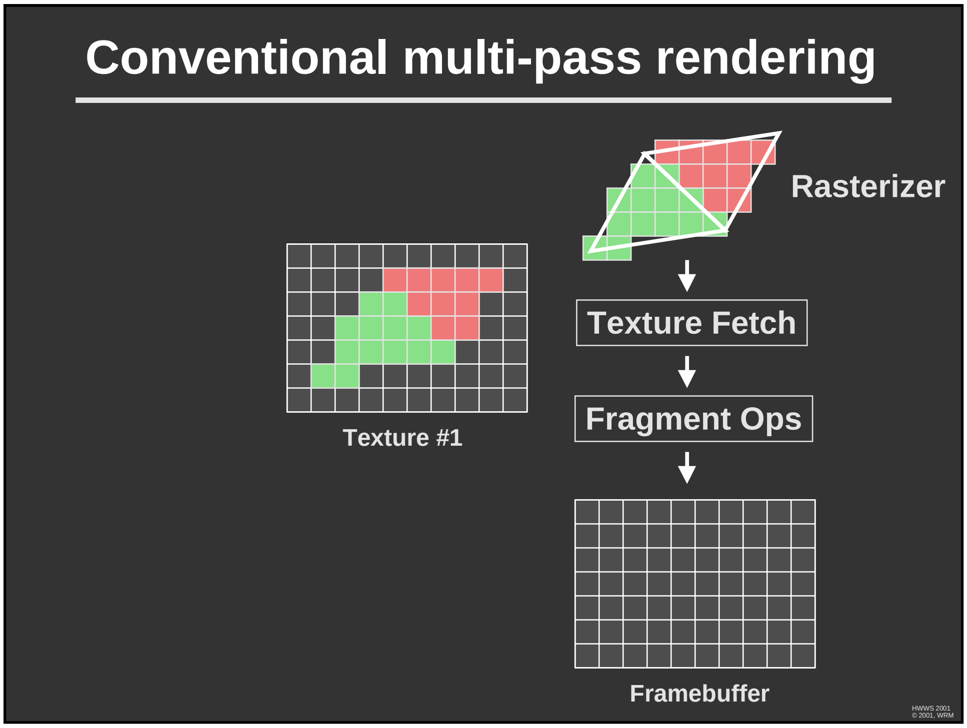
HWWS 2001
© 2001, WRM

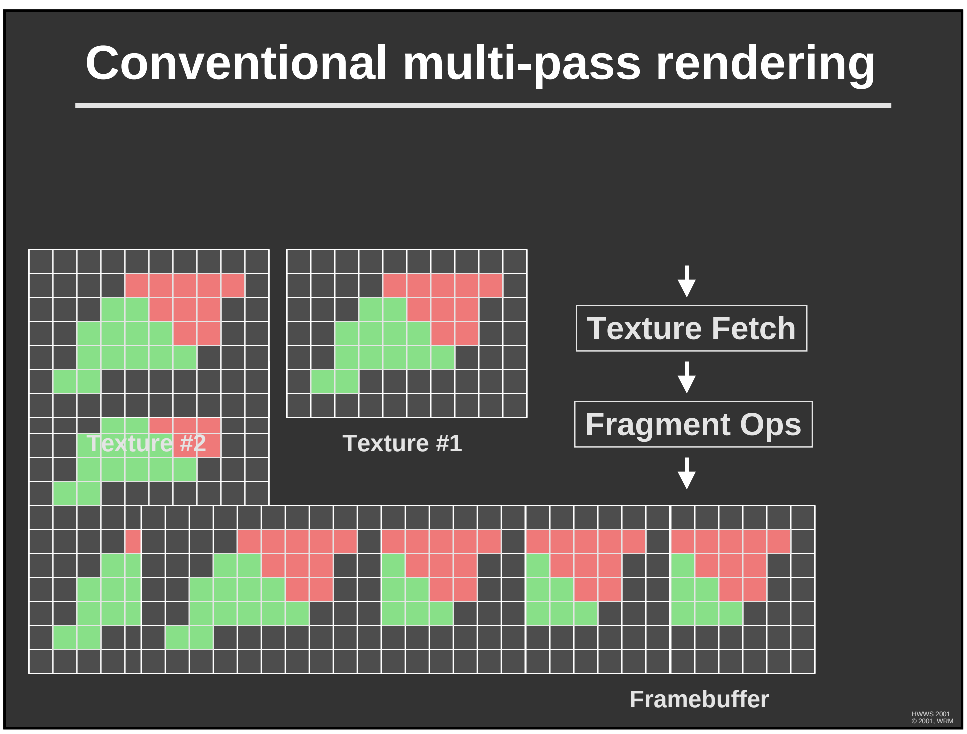

Multiple temporary values
This shade tree needs two temporary variables
*
*
+
+
+
*
over
HWWS 2001
© 2001, WRM

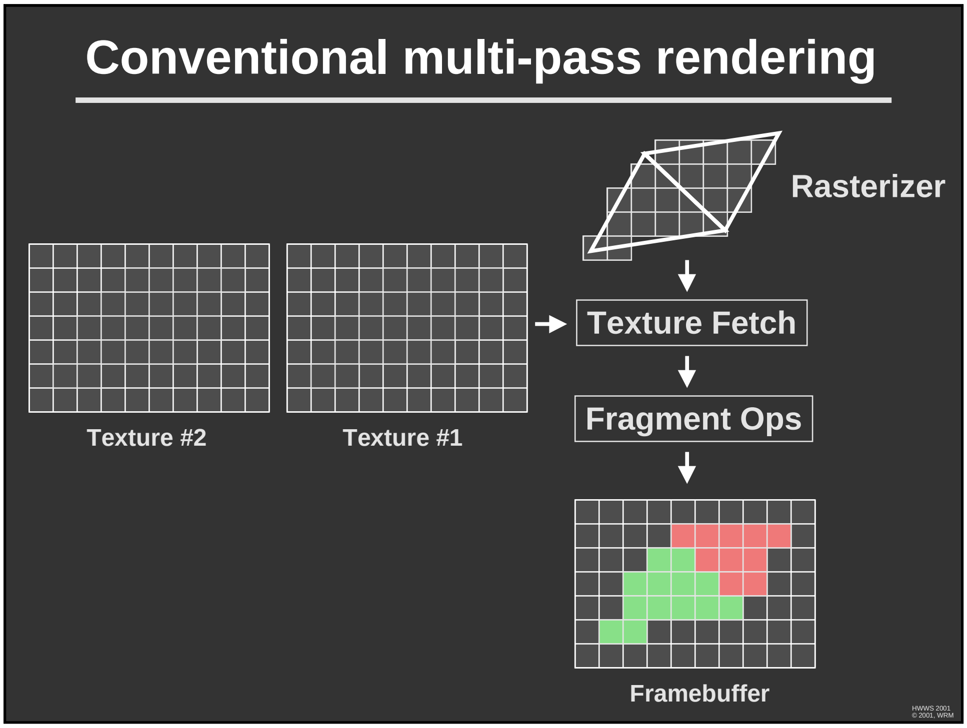

Multiple temporary values
This shade tree needs two temporary variables
*
*
+
+
+
*
+
over
HWWS 2001
© 2001, WRM

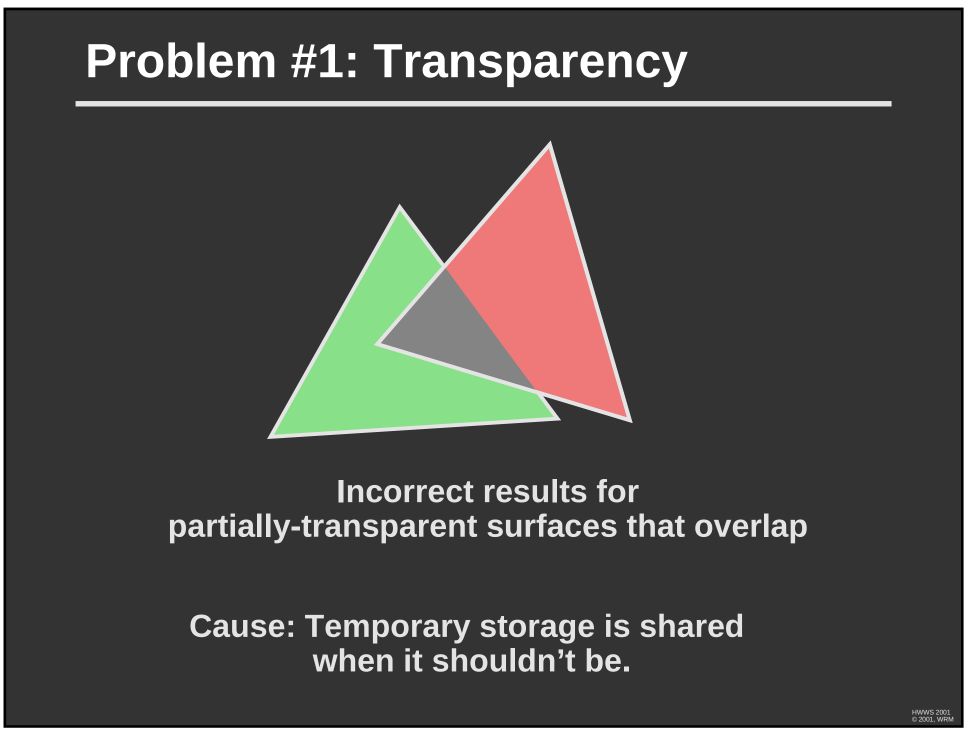

Conventional multi-pass rendering
Rasterizer
Texture Fetch
Fragment Ops
Framebuffer


Conventional multi-pass rendering
Texture Fetch
Fragment Ops
Framebuffer


Conventional multi-pass rendering
Rasterizer
Texture #1
Texture Fetch
Fragment Ops
Framebuffer
HWWS 2001
© 2001, WRM

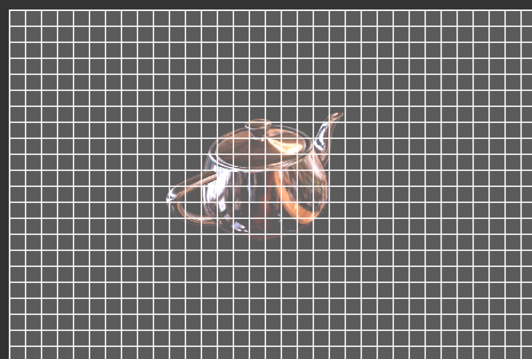

Conventional multi-pass rendering
Texture #2
Texture #1
Texture Fetch
Fragment Ops
Framebuffer
HWWS 2001
© 2001, WRM


Conventional multi-pass rendering
Rasterizer
Texture Fetch
Fragment Ops
Texture #2
Texture #1
Framebuffer
HWWS 2001
© 2001, WRM


Problem #1: Transparency
Incorrect results for
partially-transparent surfaces that overlap
Cause: Temporary storage is shared
when it shouldn't be.
HWWS 2001
© 2001, WRM

Problem #2: Wastes memory

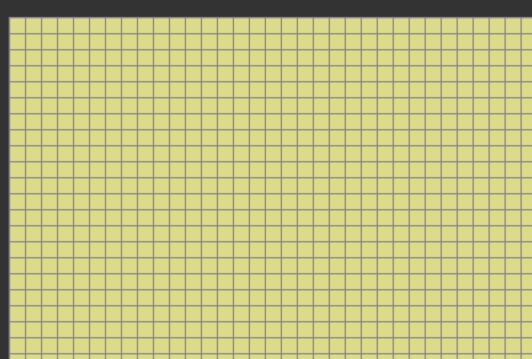
**Each additional
per-pixel temporary variable
requires a screen-sized buffer**



HWWS 2001
© 2001, WRM

Problem #2: Wastes memory

**Each additional
per-pixel temporary variable
requires a screen-sized buffer**

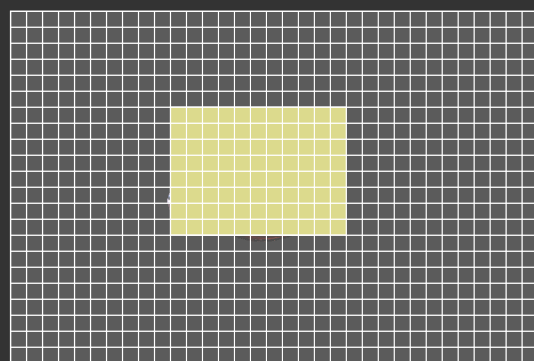


HWWS 2001
© 2001, WRM

Problem #2: Wastes memory

Each additional
per-pixel temporary variable
requires a screen-sized buffer

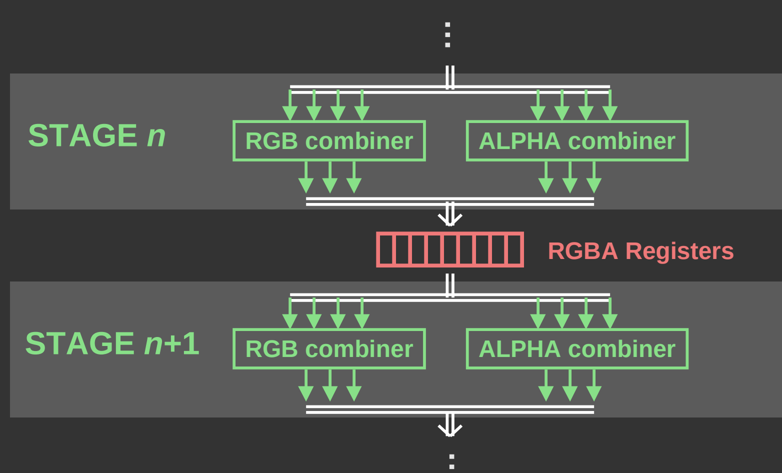
… or, a bounding-box-sized buffer



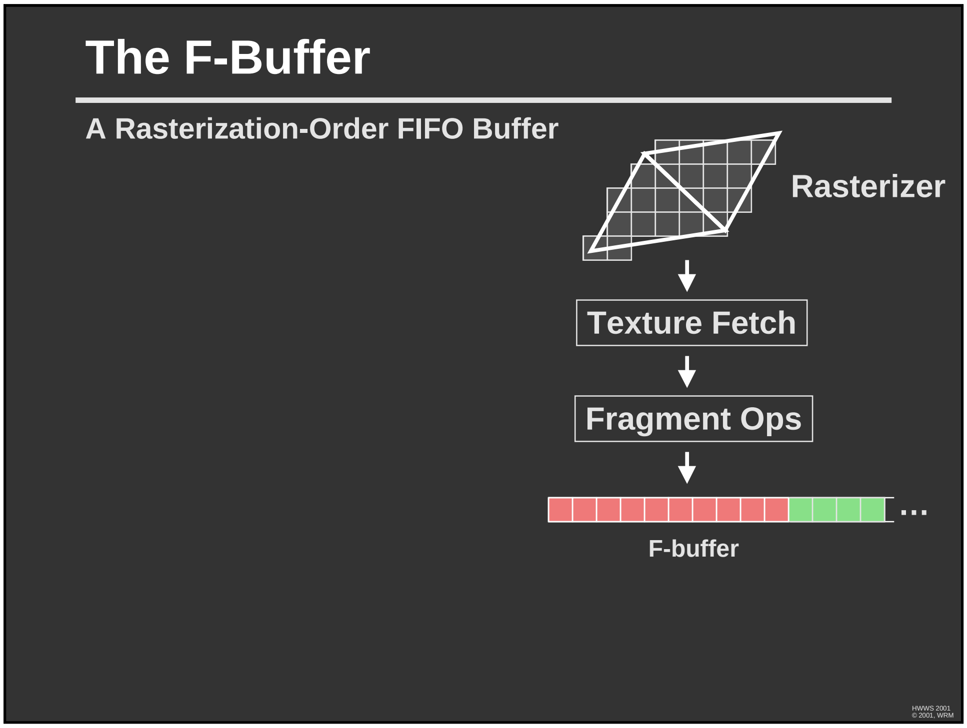
HWWS 2001
© 2001, WRM

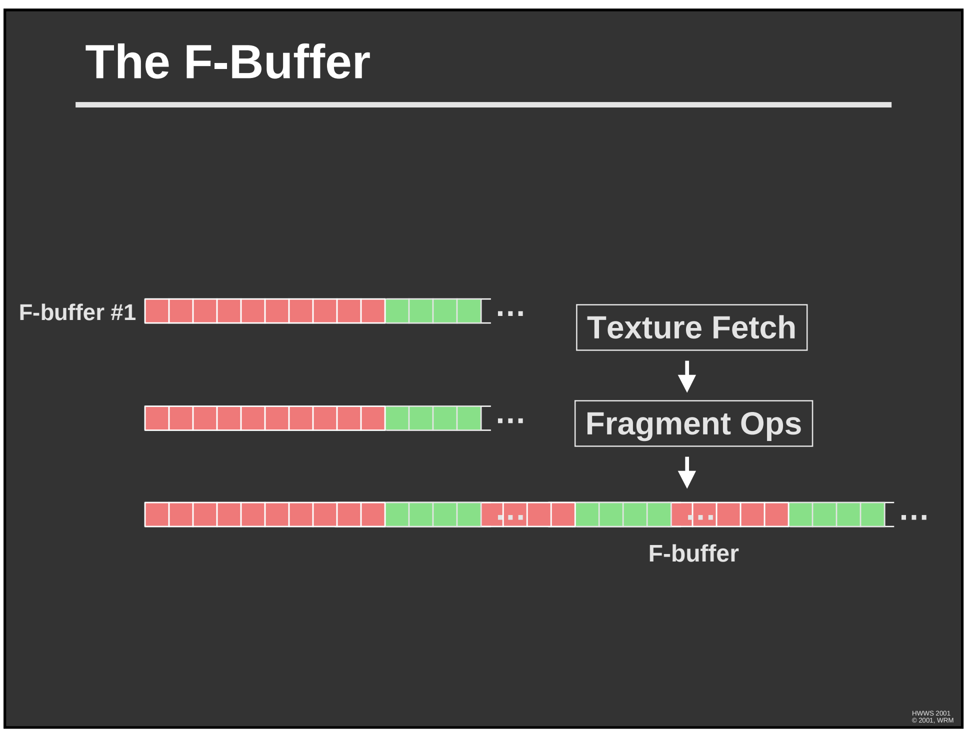
Problem #3: One output per pass

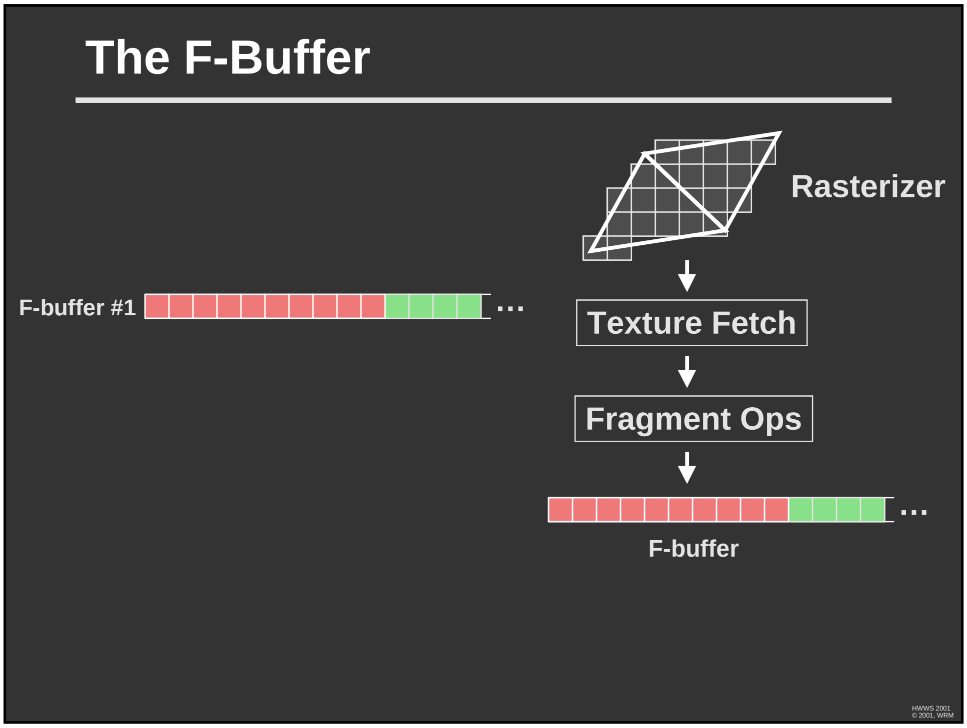
NV20 has 9 RGBA registers, but only one RGBA output
(and, it's lower precision/range!)

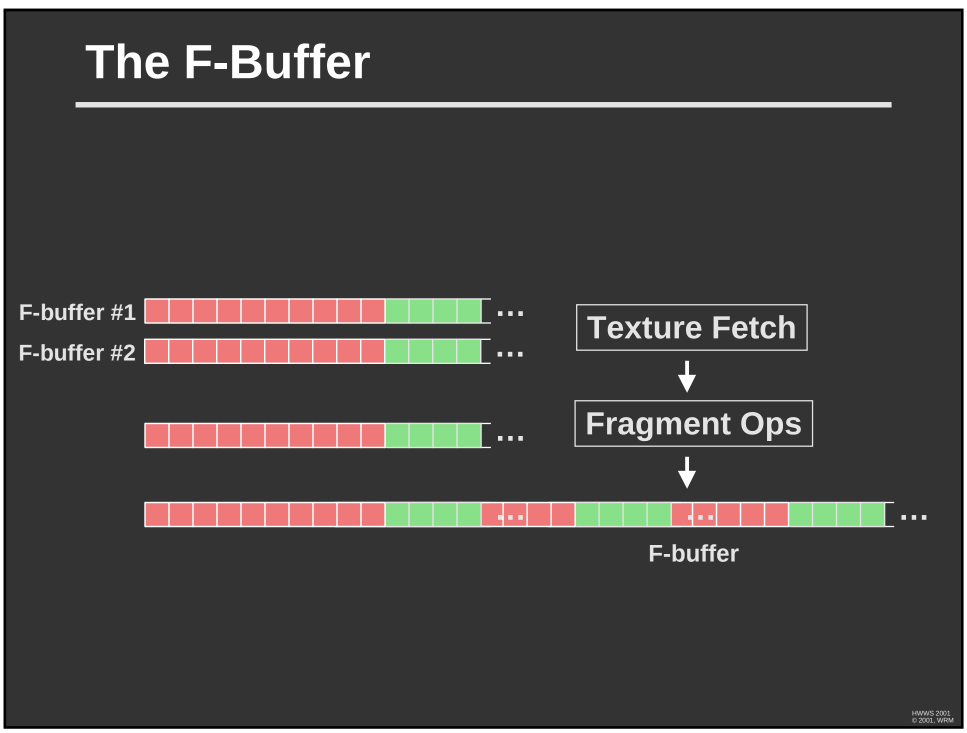


HWWS 2001
© 2001, WRM

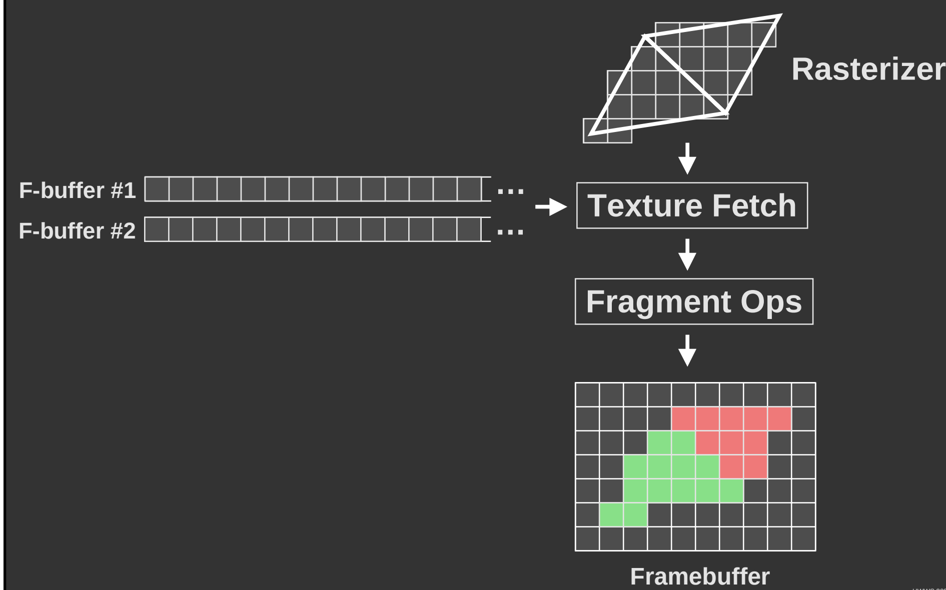

The F-Buffer
A Rasterization-Order FIFO Buffer
Rasterizer
Texture Fetch
Fragment Ops
F-buffer
HWWS 2001
© 2001, WRM


The F-Buffer
F-buffer #1
Texture Fetch
Fragment Ops
F-buffer
HWWS 2001
© 2001, WRM


The F-Buffer
Rasterizer
F-buffer #1
Texture Fetch
Fragment Ops
F-buffer
HWWS 2001
© 2001, WRM


The F-Buffer
F-buffer #1
F-buffer #2
Texture Fetch
Fragment Ops
F-buffer
HWWS 2001
© 2001, WRM

The F-Buffer



HWWS 2001
© 2001, WRM

F-Buffer implementation is simple

Requirements:

- Input: Address counter for “texture” read
- Output: Address counter for “framebuffer” write
- Software or HW to handle overflow

Possibly:

- Guarantee of consistent rasterization order

HWWS 2001
© 2001, WRM

Related ideas

Stream Processing [Rixner98]

- F-Buffer is a type of stream buffer

A-Buffer [Carpenter84]

- Associates storage with each fragment

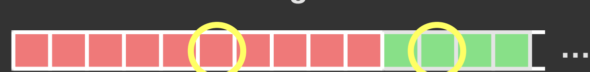
R-Buffer [Wittenbrink01]

- Order-independent transparency

HWWS 2001
© 2001, WRM

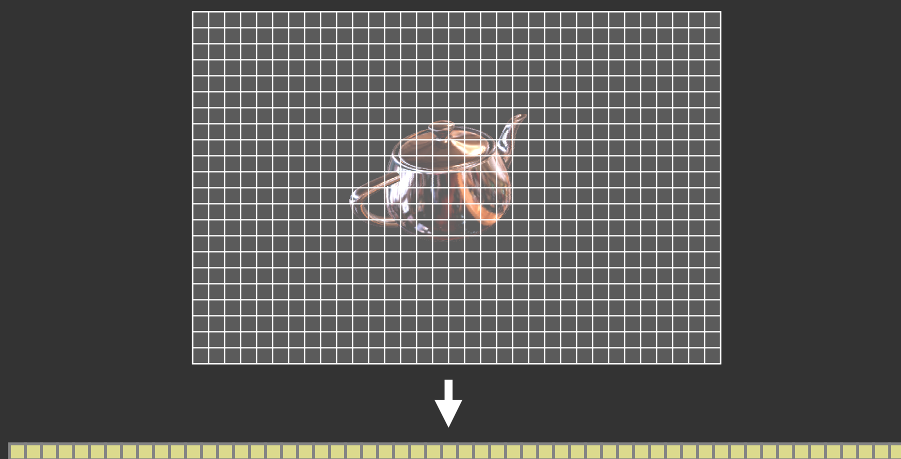
Transparency works with F-Buffer

Overlapping fragments get
their own storage locations



HWWS 2001
© 2001, WRM

F-Buffer can avoid wasting memory



Buffer does not have to be screen-sized

HWWS 2001
© 2001, WRM

Multiple outputs per pass with F-Buffer

F-Buffer makes it simpler to support multiple outputs

- FIFO access
- No read-modify-write blend ops
- Memory-usage efficiency

Extended-precision output is easier for same reasons

HWWS 2001
© 2001, WRM

Many variations of F-Buffer

Where is F-Buffer stored?

How is overflow handled?

Is geometry re-rasterized on every pass?

(if so, rasterization order must be consistent)

When are conventional framebuffer ops performed?

HWWS 2001
© 2001, WRM

Where is F-Buffer stored?

On-chip

Off-chip graphics DRAM

Main system memory

Buffer access is linear → hybrids are relatively easy

HWWS 2001
© 2001, WRM

Options for overflow

- HW-supported virtual memory
  - Removes burden from software
  - Linear access makes it simple
- Pass burden to software – just avoid overflow
- HW-supported batching of geometry
  - Break geometry into batches
  - Render all passes for a batch, then start next batch
  - HW support for starting/stopping batches
  - Fragment-granularity batching is simplest, but inefficient if overflow is common

HWWS 2001
© 2001, WRM

How frequent is overflow?

Statistics from Quake III shaders*

- 10% of shader invokations overflow
  an F-Buffer sized to 10% of screen
- 0.1% of shader invokations overflow
  an F-Buffer sized to 85% of screen
- Rarely, shader invokations overflow
  an F-Buffer sized to 100% of screen!

Note… *future* applications are flexible

* For shaders requiring two passes on a single-texture pipeline.
Two-pass shaders are used for 53% of fragments.
Each shader invokation is counted separately.

HWWS 2001
© 2001, WRM

Shading library can handle overflow

- Added F-Buffer to MesaGL
- Shading system manages F-Buffer overflows
- No changes to application

HWWS 2001
© 2001, WRM

Multi-pass rendering's future

Functional-unit virtualization in HW is great

- Easy for software to use
- Trend is clear – NV10, NV20, R200

But, multi-pass rendering will still be useful

- For further virtualization of functional units
- For virtualization of memory/registers

HWWS 2001
© 2001, WRM

Conclusion

F-Buffer can solve problems with multi-pass rendering

- Works with partial transparency
- Doesn't waste memory
- Can easily preserve multiple results per pass

F-Buffer overflow can be handled in HW and/or SW

F-Buffer could facilitate evolution towards more general stream-processing architecture

HWWS 2001
© 2001, WRM

Acknowledgements

Stanford Shading Group

- Pat Hanrahan, Svetoslav Tzvetkov, Pradeep Sen, Ren Ng, Eric Chan, John Owens, David Ebert

Sponsors

- ATI, NVIDIA, SONY, Sun
- DARPA, DOE

Useful discussions

- Roger Allen, Matt Papakipos

HWWS 2001
© 2001, WRM