k^+-buffer: Fragment Synchronized k-buffer

I3D 2014

Andreas A. Vasilakis & Ioannis Fudos

{abasilak,fudos}@cs.uoi.gr

Department of Computer Science & Engineering,
University of Ioannina, Greece

16 March 2014

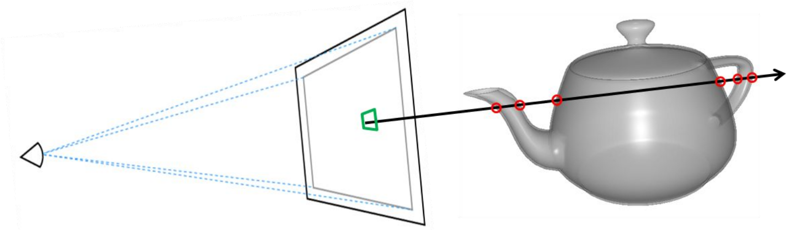
Outline

Outline

Visibility Determination

A number of image-based applications require operations on more than one (maybe occluded) fragment per pixel:

- transparency effects[1] - volume & CSG rendering[2]
- collision detection[3]
- visualization of coplanar[4] & self-trimming surfaces[5]
- shadows[6] - hair rendering[7]



[1] [Maule et al., CG'11],[2] [Bavoil et al., I3D'07],[3] [Jang et al., VC'08], [4] [Vasilakis et al., TVCG'13],[5] [Rossignac et al., CAD'13],[6] [Yang et al., CGF'10],[7] [Yu et al., I3D'12]

Prior Art - Contribution

A-buffer[1] & variants (FreePipe[2],LL[3],LL-Paged[4],SB[5])

- capture [all] fragments per pixel $\longmapsto$ sort them by depth

[1] [Carpenter, SIGGRAPH'84],[2] [Liu et al., I3D'10],[3] [Yang et al., CGF'10],[4] [Crassin, Icare3D'10],[5] [Vasilakis et al., EG'12]

Prior Art - Contribution

A-buffer[1] & variants (FreePipe[2],LL[3],LL-Paged[4],SB[5])

- capture [all] fragments per pixel $\longmapsto$ sort them by depth
- suffer from [memory overflow][2] & [fragment contention][3,4,5]

[1] [Carpenter, SIGGRAPH'84],[2] [Liu et al., I3D'10],[3] [Yang et al., CGF'10],[4] [Crassin, Icare3D'10],[5] [Vasilakis et al., EG'12]

Prior Art - Contribution

A-buffer[1] & variants (FreePipe[2],LL[3],LL-Paged[4],SB[5])

- capture [all] fragments per pixel $\longmapsto$ sort them by depth
- suffer from [memory overflow][2] & [fragment contention][3,4,5]

k-buffer (KB)[6] & variants (KB-SR[7],KB-MDT[8],KB-AB$_{LL}$[9],KB-LL[9],KB-PS[10])

- capture the [k-closest] fragments $\longmapsto$ reduce memory & sorting costs

6 [Bavoil et al., I3D'07], 7 [Bavoil et al., ShaderX6'08], 8 [Maule et al., I3D'13], 9 [Yu et al., I3D'12], 10 [Salvi, SIGGRAPH'13]

Prior Art - Contribution

A-buffer[1] & variants (FreePipe[2],LL[3],LL-Paged[4],SB[5])

- capture [all] fragments per pixel $\longmapsto$ sort them by depth
- suffer from [memory overflow][2] & [fragment contention][3,4,5]

k-buffer (KB)[6] & variants (KB-SR[7],KB-MDT[8],KB-AB$_{LL}$[9],KB-LL[9],KB-PS[10])

- capture the [k-closest] fragments $\longmapsto$ reduce memory & sorting costs
- suffer from
  1. RMW hazards[6] - $k \leq 32$[6,7] - geometry pre-sorting[6,7]

[6] [Bavoil et al., I3D'07], [7] [Bavoil et al., ShaderX6'08], [8] [Maule et al., I3D'13], [9] [Yu et al., I3D'12], [10] [Salvi, SIGGRAPH'13]

Prior Art - Contribution

A-buffer[1] & variants (FreePipe[2],LL[3],LL-Paged[4],SB[5])

- capture [all] fragments per pixel $\longmapsto$ sort them by depth
- suffer from [memory overflow][2] & [fragment contention][3,4,5]

k-buffer (KB)[6] & variants (KB-SR[7],KB-MDT[8],KB-AB$_{LL}$[9],KB-LL[9],KB-PS[10])

- capture the [k-closest] fragments $\longmapsto$ reduce memory & sorting costs
- suffer from
  1. RMW hazards[6] - $k \leq 32$[6,7] - geometry pre-sorting[6,7]
  2. additional rendering pass & depth precision conversion[8]

[6] [Bavoil et al., I3D'07], [7] [Bavoil et al., ShaderX6'08], [8] [Maule et al., I3D'13], [9] [Yu et al., I3D'12], [10] [Salvi, SIGGRAPH'13]

Prior Art - Contribution

A-buffer[1] & variants (FreePipe[2],LL[3],LL-Paged[4],SB[5])

- capture [all] fragments per pixel $\longmapsto$ sort them by depth
- suffer from [memory overflow][2] & [fragment contention][3,4,5]

k-buffer (KB)[6] & variants (KB-SR[7],KB-MDT[8],KB-AB$_{LL}$[9],KB-LL[9],KB-PS[10])

- capture the [k-closest] fragments $\longmapsto$ reduce memory & sorting costs
- suffer from
  1. RMW hazards[6] - $k \leq 32$[6,7] - geometry pre-sorting[6,7]
  2. additional rendering pass & depth precision conversion[8]
  3. unbounded memory[9] $\longmapsto$ KB-AB$_{Array}$ & KB-AB$_{SB}$

[6] [Bavoil et al., I3D'07], [7] [Bavoil et al., ShaderX6'08], [8] [Maule et al., I3D'13], [9] [Yu et al., I3D'12], [10] [Salvi, SIGGRAPH'13]

Prior Art - Contribution

A-buffer[1] & variants (FreePipe[2],LL[3],LL-Paged[4],SB[5])

- capture [all] fragments per pixel $\longmapsto$ sort them by depth
- suffer from [memory overflow][2] & [fragment contention][3,4,5]

k-buffer (KB)[6] & variants (KB-SR[7],KB-MDT[8],KB-AB_{LL}[9],KB-LL[9],KB-PS[10])

- capture the [k-closest] fragments $\longmapsto$ reduce memory & sorting costs
- suffer from
  1. RMW hazards[6] - $k \leq 32$[6,7] - geometry pre-sorting[6,7]
  2. additional rendering pass & depth precision conversion[8]
  3. unbounded memory[9] $\longmapsto$ KB-AB_{Array} & KB-AB_{SB}

Our contribution: k^+-buffer

1. overcomes all limitations of existing k-buffer alternatives

Prior Art - Contribution

A-buffer[1] & variants (FreePipe[2],LL[3],LL-Paged[4],SB[5])

- capture [all] fragments per pixel $\longmapsto$ sort them by depth
- suffer from [memory overflow][2] & [fragment contention][3,4,5]

k-buffer (KB)[6] & variants (KB-SR[7],KB-MDT[8],KB-AB$_{LL}$[9],KB-LL[9],KB-PS[10])

- capture the [k-closest] fragments $\longmapsto$ reduce memory & sorting costs
- suffer from
  1. RMW hazards[6] - $k \leq 32$[6,7] - geometry pre-sorting[6,7]
  2. additional rendering pass & depth precision conversion[8]
  3. unbounded memory[9] $\longmapsto$ KB-AB$_{Array}$ & KB-AB$_{SB}$

Our contribution: k$^+$-buffer

1. overcomes all limitations of existing k-buffer alternatives
2. memory-friendly variation (extra pass needed)

Prior Art - Contribution

A-buffer[1] & variants (FreePipe[2],LL[3],LL-Paged[4],SB[5])

- capture [all] fragments per pixel $\longmapsto$ sort them by depth
- suffer from [memory overflow][2] & [fragment contention][3,4,5]

k-buffer (KB)[6] & variants (KB-SR[7],KB-MDT[8],KB-AB$_{LL}$[9],KB-LL[9],KB-PS[10])

- capture the [k-closest] fragments $\longmapsto$ reduce memory & sorting costs
- suffer from
  1. RMW hazards[6] - $k \leq 32$[6,7] - geometry pre-sorting[6,7]
  2. additional rendering pass & depth precision conversion[8]
  3. unbounded memory[9] $\longmapsto$ KB-AB$_{Array}$ & KB-AB$_{SB}$

Our contribution: k$^+$-buffer

1. overcomes all limitations of existing k-buffer alternatives
2. memory-friendly variation (extra pass needed)
3. supports Z-buffer and A-buffer functionalities

Outline

1. Introduction
2. Framework Overview
3. Experimental Study
4. Conclusions & Future Work

k^+-buffer Pipeline

Store & Sort solution:

1. captures fragments in an unsorted sequence

k^+-buffer Pipeline

Store & Sort solution:

1. captures fragments in an unsorted sequence
2. reorders stored fragments by their depth

[1] [Crassin, Icare3D'10], [2] [Salvi, SIGGRAPH'13]

k^+-buffer Pipeline

Store & Sort solution:

1. captures fragments in an unsorted sequence
2. reorders stored fragments by their depth

1. Store Rendering Pass:

- spin-lock strategy ([binary semaphores][1] or [pixel sync][2])
  - avoids 32-bit atomic operations

[1] [Crassin, Icare3D'10], [2] [Salvi, SIGGRAPH'13]

k^+-buffer Pipeline

Store & Sort solution:

1. captures fragments in an unsorted sequence
2. reorders stored fragments by their depth

1. Store Rendering Pass:

- spin-lock strategy ([binary semaphores][1] or [pixel sync][2])
  - avoids 32-bit atomic operations
- two bounded array-based structures:
  - [early-fragment culling] - $O(1)$

example

[1] [Crassin, Icare3D'10], [2] [Salvi, SIGGRAPH'13]

k^+-buffer Pipeline

Store & Sort solution:

1. captures fragments in an unsorted sequence
2. reorders stored fragments by their depth

1. Store Rendering Pass:

- spin-lock strategy ([binary semaphores][1] or [pixel sync][2])
  - avoids 32-bit atomic operations
- two bounded array-based structures:
  - [early-fragment culling] - $O(1)$
  - if $k < 16$: [max-array]-(K$^+$B-Array), else [max-heap]-(K$^+$B-Heap)
  - [*insert*()]: $O(1)$ vs $O(\log_2 k)$ - [*find_max*()]: $O(k)$ vs $O(\log_2 k)$

example

[1] [Crassin, Icare3D'10], [2] [Salvi, SIGGRAPH'13]

k^+-buffer Pipeline

Store & Sort solution:

1. captures fragments in an unsorted sequence
2. reorders stored fragments by their depth

1. Store Rendering Pass:

- spin-lock strategy ([binary semaphores][1] or [pixel sync][2])
  - avoids 32-bit atomic operations
- two bounded array-based structures:
  - [early-fragment culling] - $O(1)$
  - if $k < 16$: [max-array]-(K$^+$B-Array), else [max-heap]-(K$^+$B-Heap)
  - [*insert*()]: $O(1)$ vs $O(\log_2 k)$ - [*find_max*()]: $O(k)$ vs $O(\log_2 k)$

example

2. Sort Full-screen Pass:

- if $k < 16$: [insertion-sort], else [shell-sort]

Extending k^+-buffer

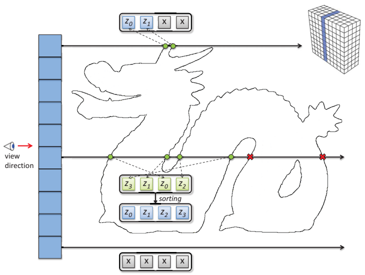
Precise memory allocation

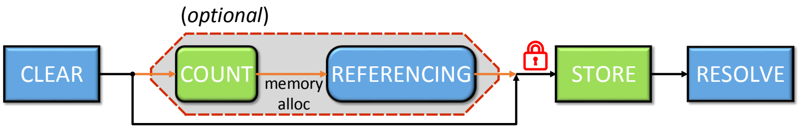
- k is the same for [all] pixels

Extending k^+-buffer

Precise memory allocation

- k is the same for [all] pixels
- [Idea]: S-buffer(SB)[1] - (K^+B-SB)
  - count pass $\longmapsto$ [hybrid scheme]
  - if [counter] reaches k stop
  - extra pass & shared memory



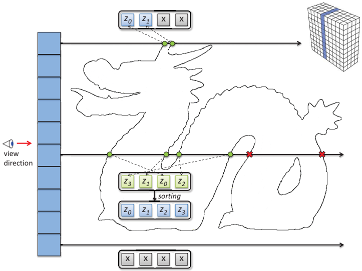


[1] [Vasilakis et al., EG'12], [2] [Liu et al., I3D'10]

Extending k^+-buffer

Precise memory allocation

- k is the same for [all] pixels
- [Idea]: S-buffer(SB)[1] - (K^+B-SB)
  - count pass $\longmapsto$ [hybrid scheme]
  - if [counter] reaches k stop
  - extra pass & shared memory



Unified framework

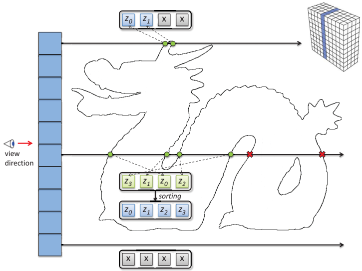
- adjust the value of k:
  - $k = 1$: [Z-buffer] behavior

[1] [Vasilakis et al., EG'12], [2] [Liu et al., I3D'10]

Extending k^+-buffer

Precise memory allocation

- k is the same for [all] pixels
- [Idea]: S-buffer(SB)[1] - (K$^+$B-SB)
  - count pass $\longmapsto$ [hybrid scheme]
  - if [counter] reaches k stop
  - extra pass & shared memory



Unified framework

- adjust the value of k:
  - $k = 1$: [Z-buffer] behavior
  - $k = \max_p\{f(p)\}$: [A-buffer] behavior:
    - K$^+$B $\longmapsto$ FreePipe[2]
    - K$^+$B-SB $\longmapsto$ SB[1]

[1] [Vasilakis et al., EG'12], [2] [Liu et al., I3D'10]

Outline

1. Introduction
2. Framework Overview
3. Experimental Study
4. Conclusions & Future Work

Testing Environment

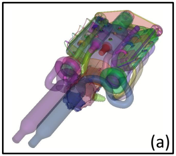
- [artificially generated scenes]: $n = r \cdot k, r \geq 1$
- [screen resolution]: 854×480 (16:9) - [pixel density]: p_d
- OpenGL 4.3 API - NVIDIA GTX 480 (1.5 GB memory)

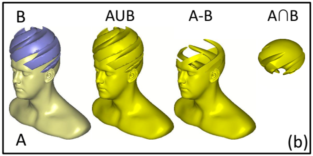
Testing Environment

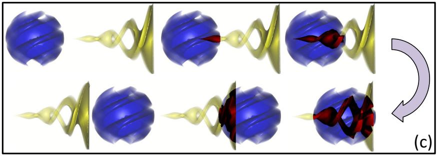
- [artificially generated scenes]: $n = r \cdot k, r \geq 1$
- [screen resolution]: 854×480 (16:9) - [pixel density]: p_d
- OpenGL 4.3 API - NVIDIA GTX 480 (1.5 GB memory)

Applications

- (a) OIT[1], (b) CSG[2], (c) Collision Detection[3]





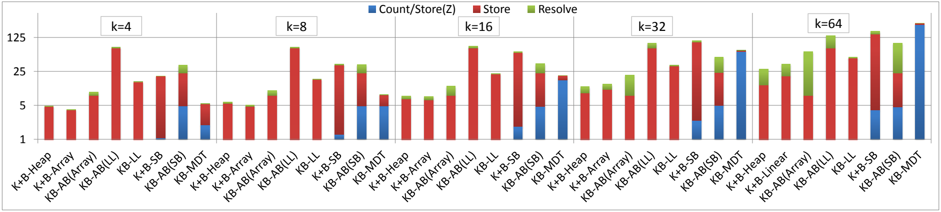


1 [Bavoil et al., I3D'07], 2 [Rossignac et al., CAD'13], 3 [Jang et al., VC'08]

Performance Analysis - *k*-buffer

Impact of *k* (milliseconds)

- [Scene]: $n = 128$, $k = \{4, \ldots, 64\}$
- K^+B-Array [$k \downarrow$] vs K^+B-Heap [$k \uparrow$]
- KB-MDT[1] (two-pass method): future 64-bit atomic operations?
- KB-AB_{Array}: storing [$k \uparrow$] - sorting [$k \downarrow$]

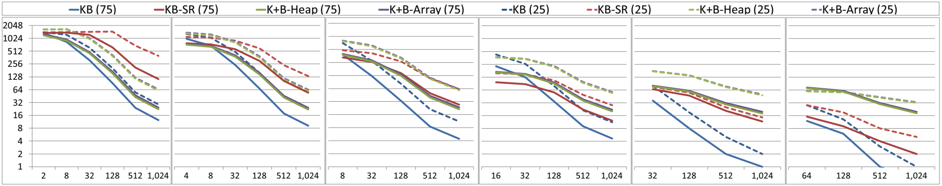


[1] [Maule et al., I3D'13]

Performance Analysis - *k*-buffer

Impact of Sorting (fps)

- [Scene]: $n = \{k, \ldots, 1024\}, p_d = \{25\%, 75\%\}$, *[depth sorted]*
- K^+B-Array ($O(1)$) >K^+B-Heap ($O(\log_2 k)$) - ([$p_d \uparrow$]: linear behavior)
- [$k \uparrow$] $\longmapsto$[multi-pass rendering]: K^+B > KB-SR[2] > KB[1]
- K^+B > KB[1] > KB-PS[3]

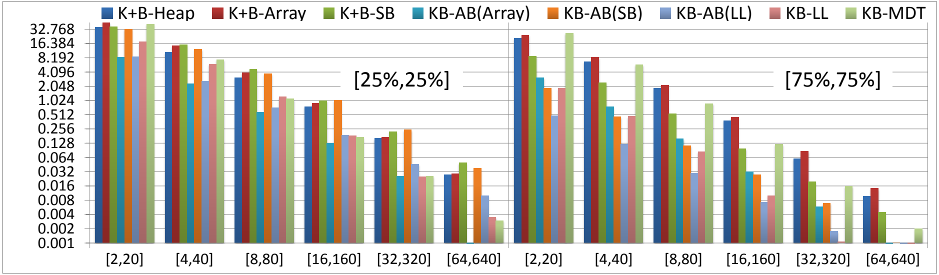


[1] [Bavoil et al., I3D'07], [2] [Bavoil et al., ShaderX6'08], [3] [Salvi, SIGGRAPH'13]

Performance Analysis - *k*-buffer

Impact of Memory (fps/MB)

- [Scene]: $n = \{k, \ldots, 10 \cdot k\}, [p_d, f_p]$
- K$^+$B-SB $[p_d \downarrow, f_p \downarrow]$ vs K$^+$B $[p_d \uparrow, f_p \uparrow]$
- $k = 64$: A-buffer-based solutions[1] fail (memory overflow)

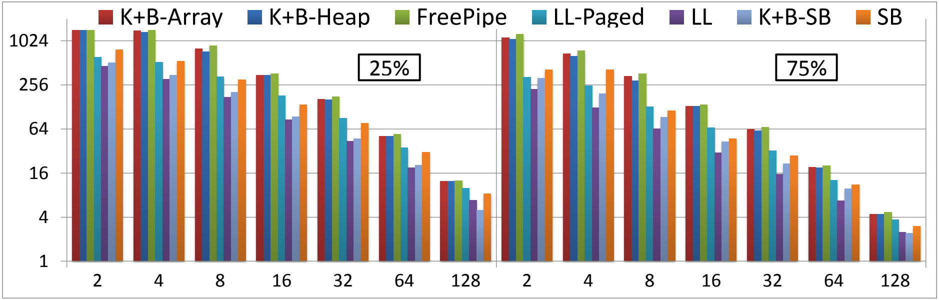


[1] [Yu et al., I3D'12]

Performance Analysis - A-buffer

Impact of k (fps)

- [Scene]: $k = n, p_d = \{25\%, 75\%\}$
- FreePipe[1] $>$ K^+B-Array $>$ K^+B-Heap $>$ rest methods[2,3,4] (culling)
- SB[4] $>$ K^+B-SB $>$ LL[2] (if condition at counting pass)



[1] [Liu et al., I3D'10], [2] [Yang et al., CGF'10], [3] [Crassin, Icare3D'10], [4] [Vasilakis et al., EG'12]

Memory Allocation Analysis

table

Comparison between [bounded-buffers]

- KB-AB$_{Array}$ needs huge resources
- K$^+$B vs {KB[1],KB-PS[3]}: more storage (8-byte/pixel)
- [pixel sync] avoids semaphore allocation (4-byte)
- [data packing] employed: $\forall k > 1 : 4k > 2k + 2$
- K$^+$B vs KB-SR[2]: $\forall k > 2 : 3k > 2k + 2$

[1] [Bavoil et al., I3D'07], [2] [Bavoil et al., ShaderX6'08], [3] [Maule et al., I3D'13], [4] [Yu et al., I3D'12], [5] [Liu et al., I3D'10], [6] [Vasilakis et al., EG'12]

Memory Allocation Analysis

table

Comparison between [bounded-buffers]

- KB-AB$_{Array}$ needs huge resources
- K$^+$B vs {KB[1],KB-PS[3]}: more storage (8-byte/pixel)
- [pixel sync] avoids semaphore allocation (4-byte)
- [data packing] employed: $\forall k > 1 : 4k > 2k + 2$
- K$^+$B vs KB-SR[2]: $\forall k > 2 : 3k > 2k + 2$

Comparison between [unbounded-buffers]

- K$^+$B-SB requires:
  1. [equal]: $f(p) \leq k$
  2. [less]: $f(p) > k$

  $\}$ KB-AB$_{LL}$[4],KB-LL[4],KB-AB$_{SB}$
- A-buffer simulation: K$^+$B = FreePipe[5] & K$^+$B-SB = SB[6]

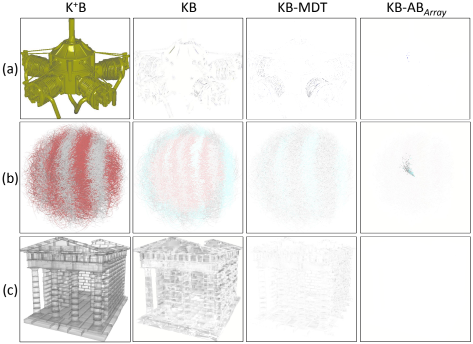
1 [Bavoil et al., I3D'07], 2 [Bavoil et al., ShaderX6'08], 3 [Maule et al., I3D'13], 4 [Yu et al., I3D'12], 5 [Liu et al., I3D'10], 6 [Vasilakis et al., EG'12]

Image Quality Analysis

Noticeable image differences from K^+B

Scenario (a) Z-buffer, (b) k-buffer, (c) A-buffer

Method KB[1]: [RMW hazards], KB-MDT[2]: [depth conversion], KB-AB_{Array}: [fragment overflow]



1 [Bavoil et al., I3D'07], 2 [Maule et al., I3D'13]

Outline

Conclusions & Future Work

Bounded multi-fragment storage using k^{+}-buffer

- alleviates prior *k*-buffer limitations and bottlenecks by exploiting fragment culling and pixel synchronization

Conclusions & Future Work

Bounded multi-fragment storage using k^{+}-buffer

- alleviates prior *k*-buffer limitations and bottlenecks by exploiting fragment culling and pixel synchronization
- introduces an extension to avoid wasteful memory consumption

Conclusions & Future Work

Bounded multi-fragment storage using k$^+$-buffer

- alleviates prior *k*-buffer limitations and bottlenecks by exploiting fragment culling and pixel synchronization
- introduces an extension to avoid wasteful memory consumption
- can also simulate the behavior of Z-buffer or A-buffer

Conclusions & Future Work

Bounded multi-fragment storage using k^+-buffer

- alleviates prior *k*-buffer limitations and bottlenecks by exploiting fragment culling and pixel synchronization
- introduces an extension to avoid wasteful memory consumption
- can also simulate the behavior of Z-buffer or A-buffer

Directions for future work

1. performance evaluation/comparison on Haswell GPU

Conclusions & Future Work

Bounded multi-fragment storage using k^{+}-buffer

- alleviates prior *k*-buffer limitations and bottlenecks by exploiting fragment culling and pixel synchronization
- introduces an extension to avoid wasteful memory consumption
- can also simulate the behavior of Z-buffer or A-buffer

Directions for future work

1. performance evaluation/comparison on Haswell GPU
2. reduce cost of additional accumulation step:
    1. lower-detailed subdivision of the initial scene
    2. exploit temporal coherence solutions

Conclusions & Future Work

Bounded multi-fragment storage using k$^+$-buffer

- alleviates prior *k*-buffer limitations and bottlenecks by exploiting fragment culling and pixel synchronization
- introduces an extension to avoid wasteful memory consumption
- can also simulate the behavior of Z-buffer or A-buffer

Directions for future work

1. performance evaluation/comparison on Haswell GPU
2. reduce cost of additional accumulation step:
    1. lower-detailed subdivision of the initial scene
    2. exploit temporal coherence solutions
3. explore dynamic k$^+$-buffer: *k* value is not the same for all pixels

Thank you! - Questions?

Downloadable Source Code

GLSL shaders for all presented & tested methods are available at:
http://cgrg.cs.uoi.gr/k+-buffer.php

Acknowledgements

This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program Education and Lifelong Learning of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

European Union
European Social Fund

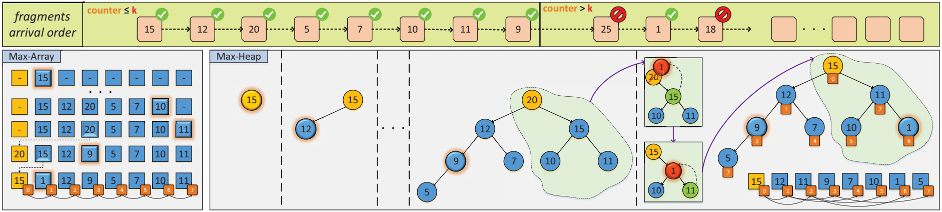
OPERATIONAL PROGRAMME
EDUCATION AND LIFELONG LEARNING
investing in knowledge society
MINISTRY OF EDUCATION & RELIGIOUS AFFAIRS
MANAGING AUTHORITY

NSRF 2007-2013
programme for development
EUROPEAN SOCIAL FUND

Co-financed by Greece and the European Union

k-buffer store example

go back



k-buffer methods details

go back

Algorithm		Performance	Sorting need		Peeling Accuracy		Memory	
Acronym	**Description**	**Rendering Passes**	**on primitives**	**on fragments**	**Max *k***	**32bit Float Precision**	**Per Pixel Allocation**	**Fixed**
KB	Initial *k*-buffer implementation [Callahan2005,Bavoil2007]	1	√	√	8; 16	√	2k; 4k	√
KB-Multi	Multi-pass *k*-buffer [Liu2009a]	1 to k	√	√			2k; 4k	
KB-SR	Stencil routed *k*-buffer [Bavoil2008]	1	√	√	32		3k	
KB-PS	*k*-buffer using pixel synchronization extension [Salvi2013]	1	x	√	-		2k	
K^+B-Array	k^+-buffer using max-array	1	x	√	-		2k + 2	
K^+B-Heap	k^+-buffer using max-heap	1	x	√	-		2k + 2	
KB-MDT	Multi depth test scheme [Liu2010,Maule2013]	2	x	x	-	x	2k	
KB-MHA	Memory-hazard-aware *k*-buffer [Zhang2013]	1	√	√	8; 16		2k; 4k	
$KB\text{-}AB_{Array}$	*k*-buffer based on A-buffer (fixed-size arrays)	1	x	√	-	√	2n + 1	
$KB\text{-}AB_{LL}$	*k*-buffer based on A-buffer (dynamic linked lists) [Yu2012]	1	x	√	-		3f + 1	x
KB-LL	*k*-buffer based on linked lists [Yu2012]	1	x	x	-		3f + 6	
$KB\text{-}AB_{SB}$	*k*-buffer based on S-buffer (variable-contigious regions)	2	x	√	-		2f + 2	
K^+B-SB	Memory-friendly variation of k^+-buffer	2	x	√	-		$2f_k + 3$	
f(p) = # *fragments at pixel* p[x,y]		$n = \max_{x,y}\{f(p)\}$			In A ; B, A denotes the layers/memory for the basic method and B for the variation using attribute packing			
$f_k(p) = (f(p) < k)\ ?\ f(p) : k$		$f_k(p) \le k$						