

The graphic features a solid red background on the left. On the right, a 3D wireframe structure extends into the distance, composed of red, black, and white rectangular blocks. A white rectangular frame is superimposed on this structure. The AMD logo is positioned within the red area on the left.

AMD

Fusion<sup>11</sup>  
DEVELOPER SUMMIT



**Fusion<sup>™</sup>**  
DEVELOPER SUMMIT

# ***REAL-TIME CONCURRENT LINKED LIST CONSTRUCTION ON THE GPU***

Jay McKee  
AMD  
MTS Engineer

# MOTIVATION

- Take advantage of new DX11 hardware features
  - Shader Model 5.0
  - DirectCompute
  - Append Buffers
- Address transparency problem

# CONTRIBUTION

- Fast creation of linked lists of arbitrary size using existing APIs and GPUs
- Integration into the standard graphics pipeline
  - Demonstrates compute from rasterized data
  - DirectCompute features in Pixel Shader
- Examples:
  - Order Independent Transparency (OIT)
  - Indirect Shadowing

# BACKGROUND

- A-buffer – Carpenter '84
  - CPU side linked list per-pixel for anti-aliasing
  - Hardware – R-buffer and Irregular Z-buffer
- Fixed array per-pixel
  - Hardware – F-buffer, stencil routed A-buffer,  $Z^3$  buffer, and k-buffer
  - Software – Slice map, bucket depth peeling
- Multi-pass
  - Bucket sorting – Rozen '08
  - Depth peeling methods for transparency

# RECENT

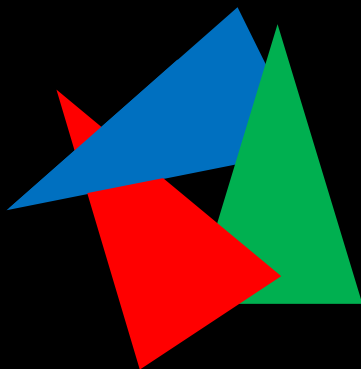
- FreePipe [Liu et. al., I3D '10]
  - Fixed array per-pixel
  - Per-pixel counter to global buffer
- PreCalc [DX11 SDK]
  - Create exact array representation
  - Accumulation buffer and prefix sum

# LINKED LIST CONSTRUCTION

- Two Buffers
  - Head pointer buffer
    - addresses/offsets
    - Initialized to end-of-list (EOL) value (e.g., -1)
  - Node buffer
    - arbitrary payload data + “next pointer”
- Each shader thread
  1. Retrieve and increment global counter value
  2. Atomic exchange into head pointer buffer
  3. Add new entry into the node buffer at location from step 1

# ORDER INDEPENDENT TRANSPARENCY / Construction by Example

- Classical problem in computer graphics
- Correct rendering of semi-transparent geometry requires sorting – blending is an order-dependent operation
- Sometimes sorting triangles is enough but not always
  - **Difficult to sort**: Multiple meshes interacting (many draw calls)
  - **Impossible to sort**: Intersecting triangles (must sort fragments)



Try doing this  
in PowerPoint



# ***ORDER INDEPENDENT TRANSPARENCY WITH PER-PIXEL LINKED LISTS***

- Computes correct transparency
- Good performance
- Works with depth and stencil testing
- Works with and without MSAA

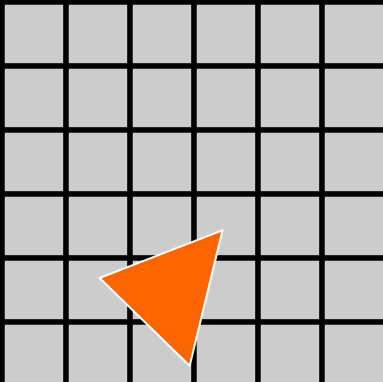
# ALGORITHM OVERVIEW

0. Render opaque scene objects
1. Render transparent scene objects
2. Screen quad resolves and composites fragment lists

## STEP 0 – RENDER OPAQUE

- Render all opaque geometry normally

Render Target



# ALGORITHM OVERVIEW

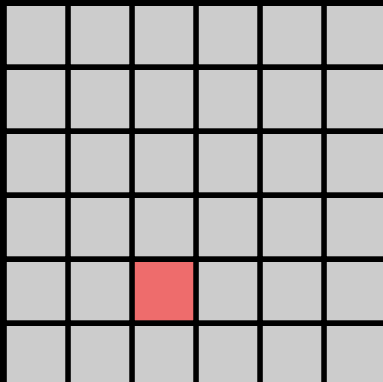
0. Render opaque scene objects
1. Render transparent scene objects
  - All fragments are stored using per-pixel linked lists
  - Store fragments: color, alpha, & depth
2. Screen quad resolves and composites fragment lists

# SETUP

- Two buffers
  - Screen sized head pointer buffer
  - Node buffer – large enough to handle all fragments
- Render as usual
- Disable render target writes

## STEP 1 | Create Linked List

Render Target



Head Pointer

Buffer

-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1

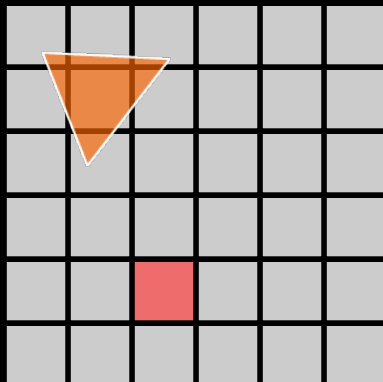
Node Buffer

0	1	2	3	4	5	6	...		

Counter = 0

## STEP 1 | Create Linked List

Render Target



Head Pointer

Buffer					
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1

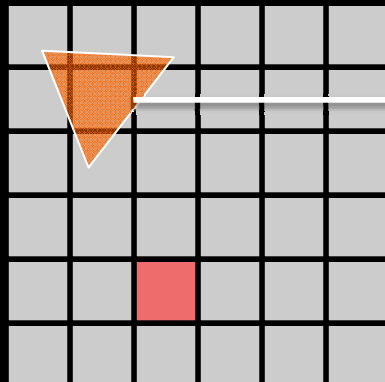
Node Buffer

0	1	2	3	4	5	6	...		

Counter = 0

## STEP 1 | Create Linked List

Render Target



Head Pointer

Buffer					
-1	-1	-1	-1	-1	-1
1	0	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1

Node Buffer

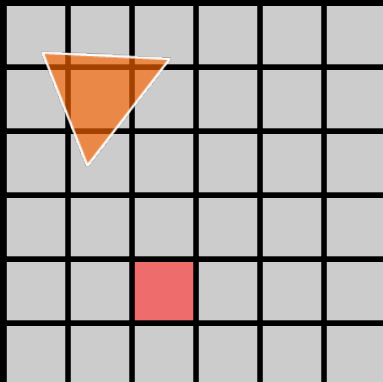
0	1	2	3	4	5	6	...

Counter = 1



## STEP 1 | Create Linked List

Render Target



Head Pointer

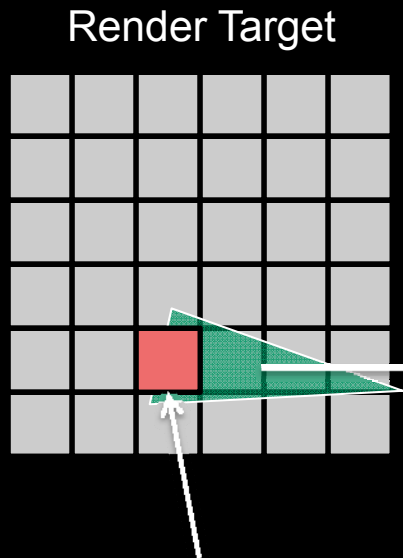
Buffer					
-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1

Node Buffer

0	1	2	3	4	5	6	...
0.87							
-1							

Counter = 1

## STEP 1 | Create Linked List



Head Pointer

Buffer

-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

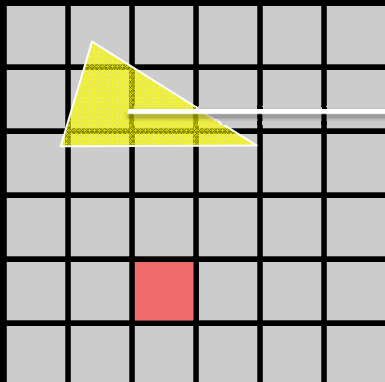
Node Buffer

0	1	2	3	4	5	6	...
0.87	0.89	0.90					
-1	-1	-1					

Counter = 3

## STEP 1 | Create Linked List

Render Target



Head Pointer

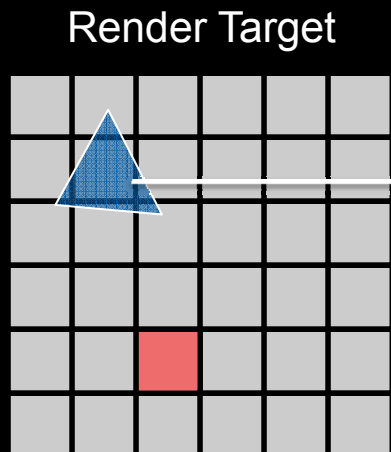
Buffer					
-1	-1	-1	-1	-1	-1
-1	3	4	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

Node Buffer

0	1	2	3	4	5	6	...
0.87	0.89	0.90	0.65	0.65			
-1	-1	-1	0	-1			

Counter = 5

## STEP 1 | Create Linked List



Head Pointer

Buffer

-1	-1	-1	-1	-1	-1
-1	5	4	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

Counter = 6

Node Buffer

0	1	2	3	4	5	6	...
0.87	0.89	0.90	0.65	0.65	0.71		
-1	-1	-1	0	-1	3		

# NODE BUFFER COUNTER

- Counter allocated in GPU memory (i.e. a buffer)
  - Atomic updates
  - Contention issues
- DX11 Append feature
  - Linear writes to a buffer
  - Implicit writes
    - `Append( )`
  - Explicit writes
    - `IncrementCounter( )`
    - Standard memory operations
  - Up to 60% faster than memory counters

# ALGORITHM OVERVIEW

0. Render opaque scene objects
1. Render transparent scene objects
2. Screen quad resolves and composites fragment lists
  - Single pass
  - Pixel shader sorts associated linked list (e.g., insertion sort)
  - Composite fragments in sorted order with background
  - Output final fragment

## STEP 2 | Render Fragments

Render Target

Dark Gray	Dark Gray	Dark Gray	Dark Gray	Dark Gray	Dark Gray
Dark Gray	Light Gray	Light Gray	Light Gray	Light Gray	Light Gray
Light Gray	Light Gray	Light Gray	Light Gray	Light Gray	Light Gray
Light Gray	Light Gray	Light Gray	Light Gray	Light Gray	Light Gray
Light Gray	Light Gray	Light Gray	Light Gray	Light Gray	Light Gray
Light Gray	Light Gray	Light Gray	Light Gray	Light Gray	Light Gray
Light Gray	Light Gray	Light Gray	Light Gray	Light Gray	Light Gray
Light Gray	Light Gray	Light Gray	Light Gray	Light Gray	Light Gray
Light Gray	Light Gray	Light Gray	Light Gray	Light Gray	Light Gray
Light Gray	Light Gray	Light Gray	Light Gray	Light Gray	Light Gray

$(0,0) \rightarrow (1,1)$ :

Fetch Head Pointer: -1

-1 indicates no fragment to render

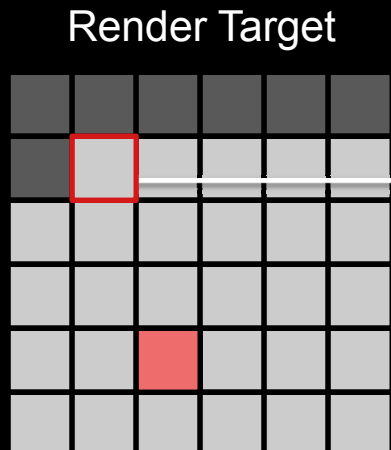
Head Pointer Buffer

-1	-1	-1	-1	-1	-1
-1	5	4	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

Node Buffer

0	1	2	3	4	5	6	...		
0.87	0.89	0.90	0.65	0.65	0.71				
-1	-1	-1	0	-1	3				

## STEP 2 | Render Fragments



(1,1):  
Fetch Head Pointer: 5  
Fetch Node Data (5)  
Walk the list and store in temporary array

0.71	0.65	0.87
------	------	------

Head Pointer Buffer

-1	-1	-1	-1	-1	-1
-1	5	4	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

Node Buffer

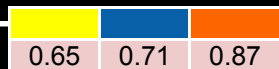
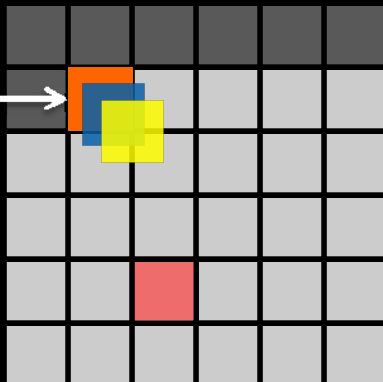
0	1	2	3	4	5	6	...
0.87	0.89	0.90	0.65	0.65	0.71		
-1	-1	-1	0	-1	3		

Counter = 6



## STEP 2 | Render Fragments

Render Target



Head Pointer Buffer

-1	-1	-1	-1	-1	-1
-1	5	4	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

Node Buffer

0	1	2	3	4	5	6	...		
0.87	0.89	0.90	0.65	0.65	0.71				
-1	-1	-1	0	-1	3				

## STEP 2 | Render Fragments

Render Target

Gray	Gray	Gray	Gray	Gray	Gray
Gray	Yellow	Yellow	Gray	Gray	Gray
Gray	Gray	Gray	Gray	Gray	Gray
Gray	Gray	Gray	Gray	Gray	Gray
Gray	Gray	Red	Green	Green	Gray
Gray	Gray	Gray	Gray	Gray	Gray

Head Pointer Buffer

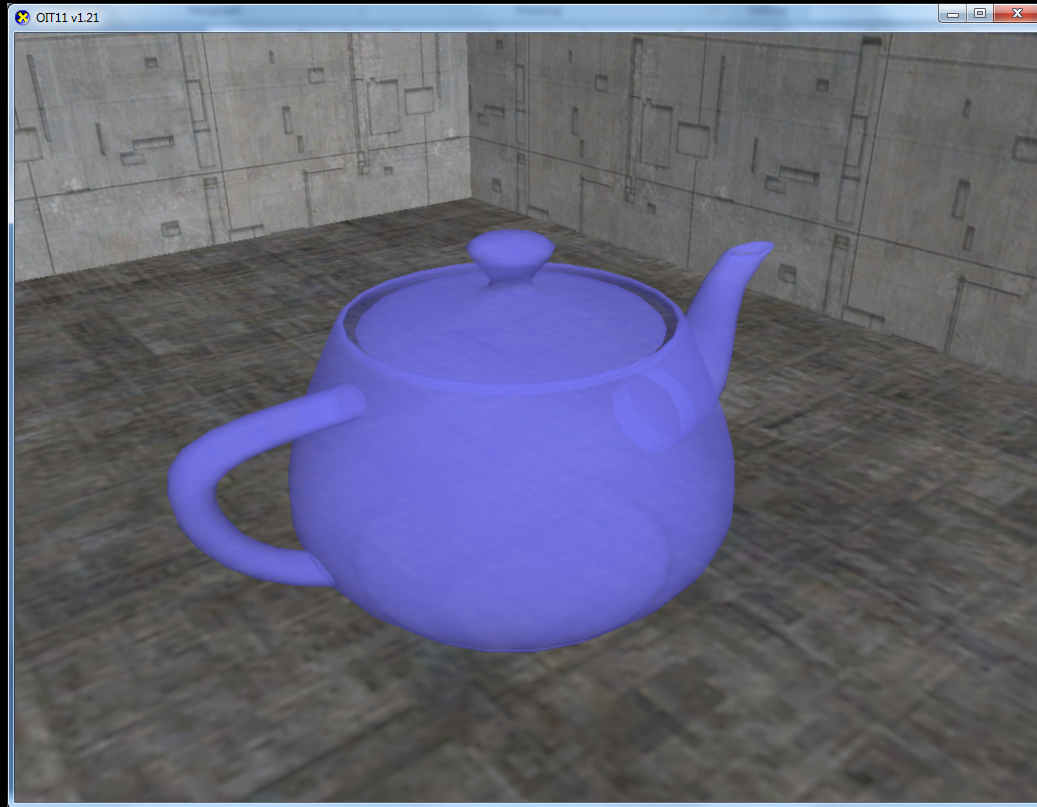
-1	-1	-1	-1	-1	-1
-1	5	4	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

Node Buffer

0	1	2	3	4	5	6	...		
0.87	0.89	0.90	0.65	0.65	0.71				
-1	-1	-1	0	-1	3				

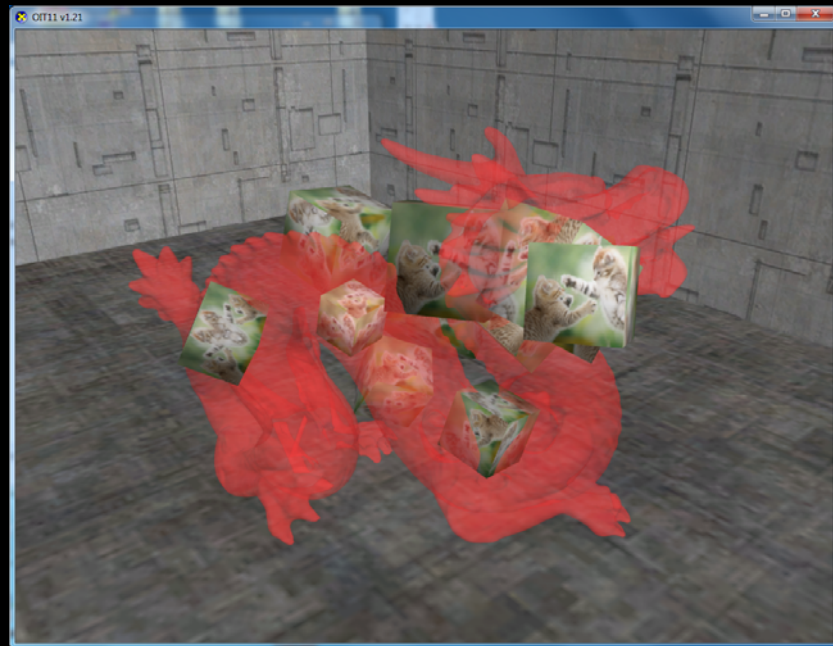
# ANTI-ALIASING

- Store coverage information in the linked list
- Resolve on per-sample
  - Execute a shader at each sample location
  - Use MSAA hardware
- Resolve per-pixel
  - Execute a shader at each pixel location
  - Average all sample contributions within the shader



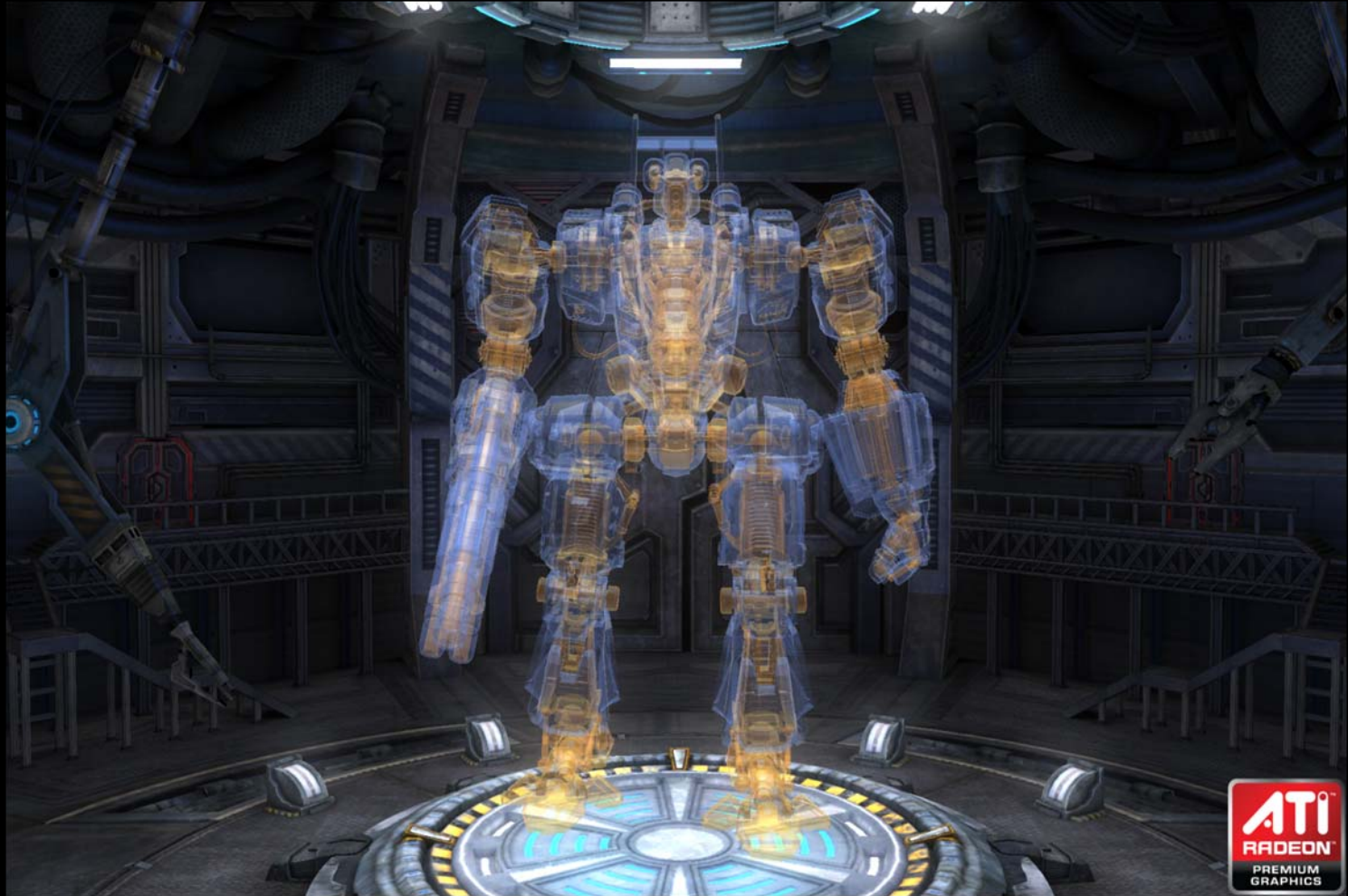
# PERFORMANCE COMPARISON

	Teapot	Dragon
Linked List	743 fps	338 fps
Precalc	285 fps	143 fps
Depth Peeling	579 fps	45 fps
Bucket Depth Peeling	---	256 fps
Dual Depth Peeling	---	94 fps



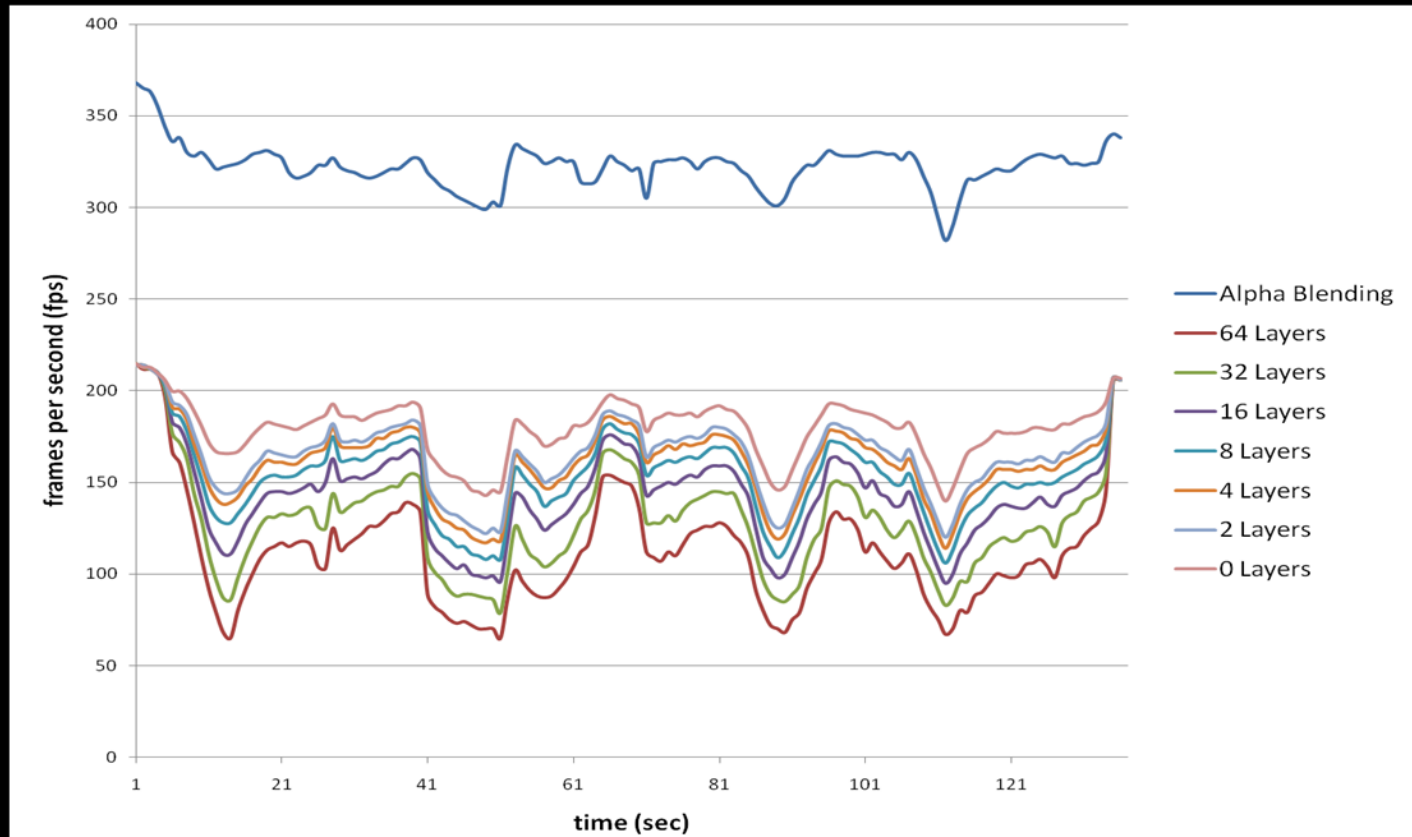
# MECHA DEMO

- 602K scene triangles
- 254K transparent triangles

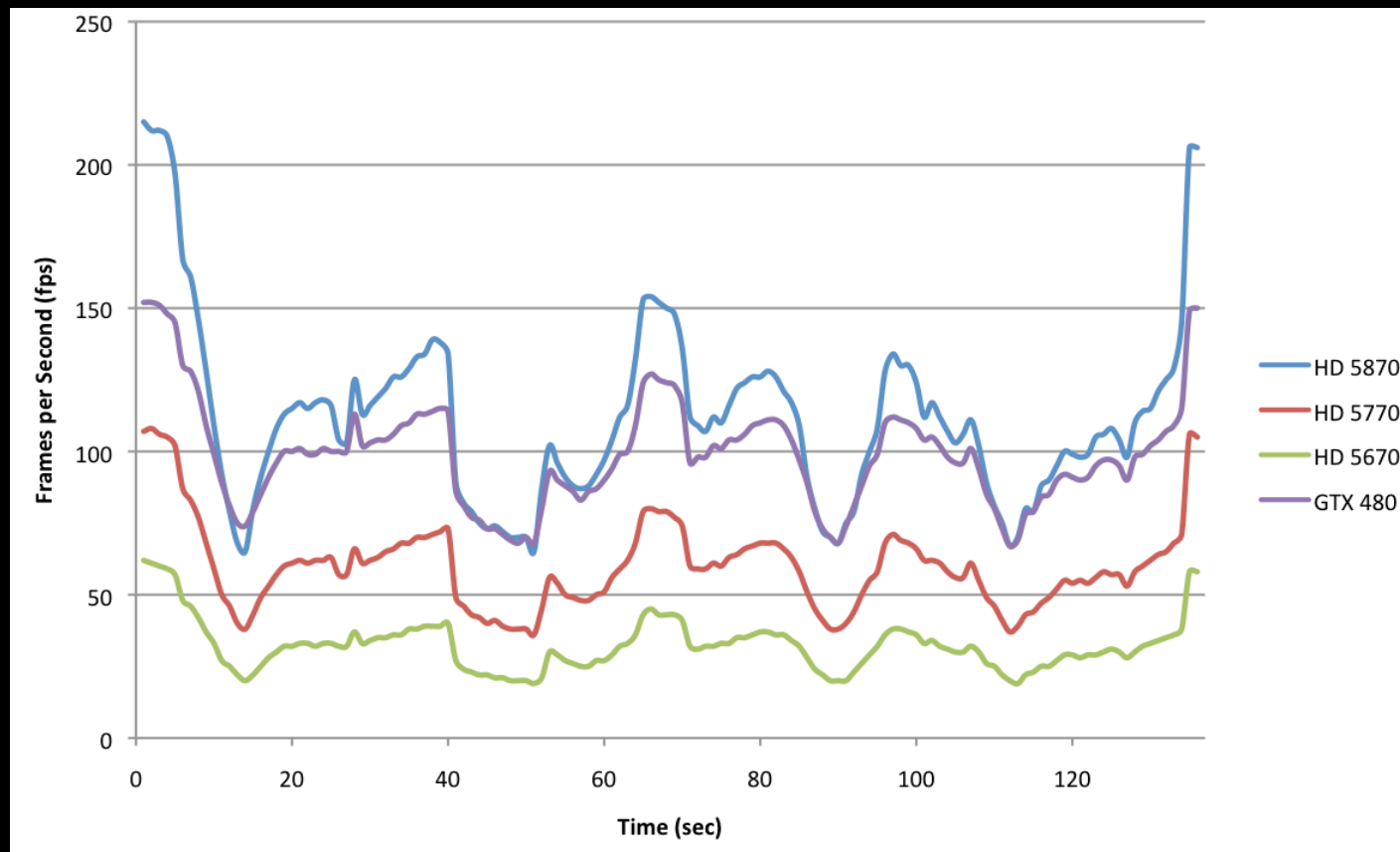




# LAYERS



# SCALING



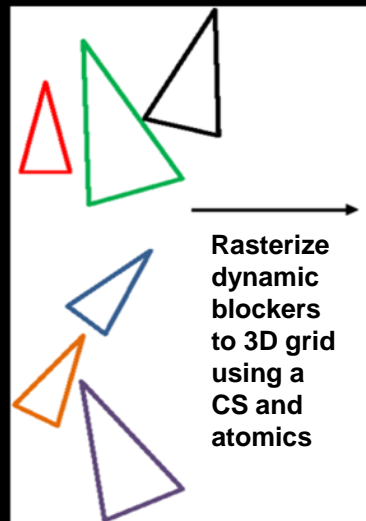


# INDIRECT SHADOWING

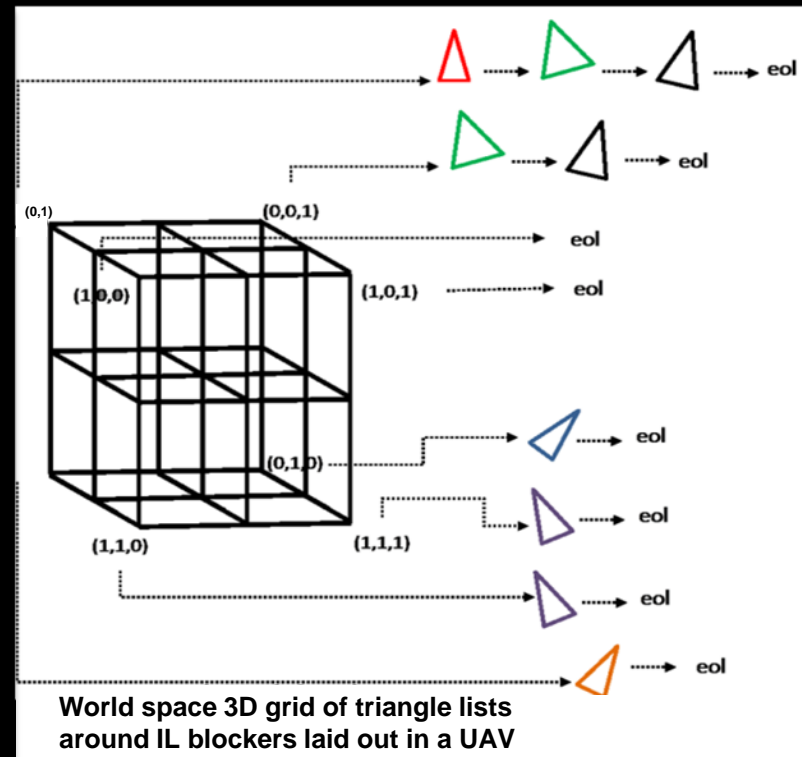
- Real-time one bounce illumination accounting for blocked indirect lights
  - Ray trace fully dynamic scenes
- Reflective shadow maps (RSM) [Dachsbabzcher et al.]
  - One bounce illumination caused by surfaces from light view

# INDIRECT SHADOWING CONSTRUCTION

1. Render a G-buffer from camera
2. Render a RSM from light
3. Construct a grid of dynamic blocker geometry (compute shader)
4. Accumulate indirect light from RSM
5. Determine blocking light by ray tracing through triangle grid
6. Apply to difference to the g-buffer

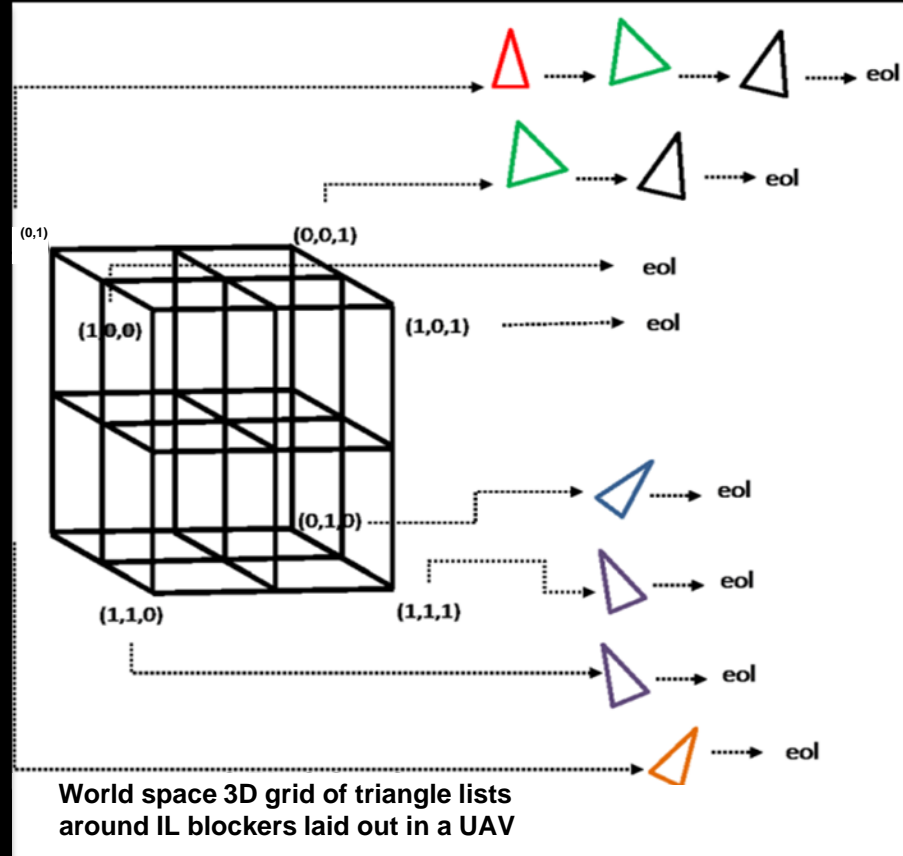
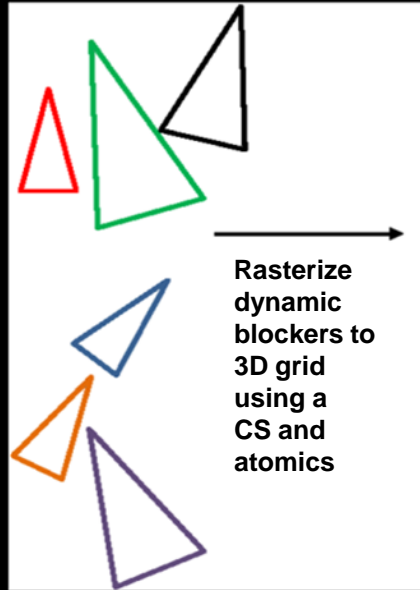


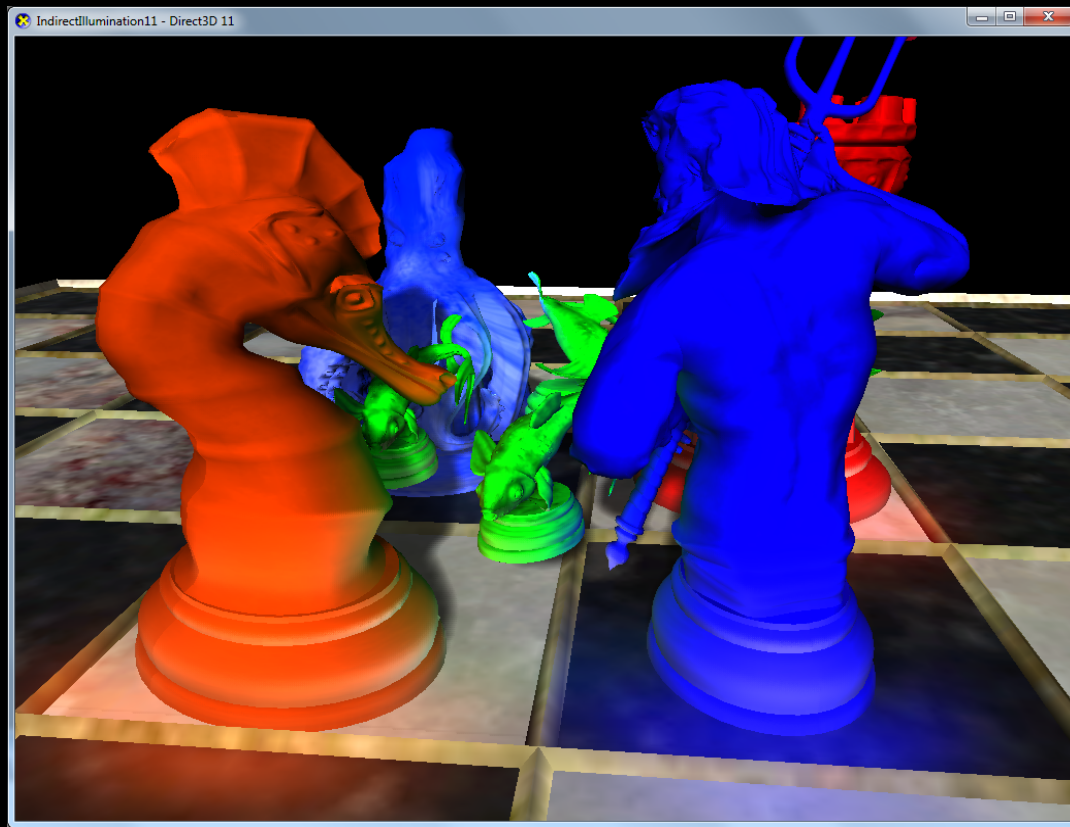
Scene



eol = End of list (0xffffffff)

# INSERT TRIS INTO 3D GRID OF TRIANGLE LISTS





# PERFORMANCE

- 60-200 FPS using a 3D grid
  - 1024x768 – AMD Radeon™ HD 5870
- Cast 300M rays per second (AMD Radeon HD 5890)
- Insert 300K blocker geometry per second

# PERFORMANCE

- 60-200 FPS using a 3D grid
  - 1024x768 – AMD Radeon HD 5870
- Cast 300M rays per second (AMD Radeon HD 5890)
- Insert 300K blocker geometry per second

# CONCLUSIONS

- Drawbacks
  - List ordering
  - Memory (size and allocation)
- Future
  - Programmable blend
  - More general data structures on the GPU

# THANKS

- Jakub Klarowicz – Techland
- Abe Wiley, Dan Roeger, David Hoff, and Tom Frisinger – AMD
- Chris Oat – Rockstar New England
- <http://developer.amd.com/samples/demos/pages/ATI RadeonHD5800SeriesRealTimeDemos.aspx>



# CODE EXAMPLE

```
RWStructuredBuffer    RWStructuredCounter;
RWTexture2D<int>      tRWFragmentListHead;
RWTexture2D<float4>   tRWFragmentColor;
RWTexture2D<int2>     tRWFragmentDepthAndLink;

[earlydepthstencil]
void PS( PsInput input )
{
    float4 vFragment = ComputeFragmentColor(input);
    int2    vScreenAddress = int2(input.vPositionSS.xy);

    // Get counter value and increment
    int nNewFragmentAddress = RWStructuredCounter.IncrementCounter();
    if ( nNewFragmentAddress == FRAGMENT_LIST_NULL ) { return; }

    // Update head buffer
    int nOldFragmentAddress;
    InterlockedExchange(tRWFragmentListHead[vScreenAddress], nNewHeadAddress, nOldHeadAddress );

    // Write the fragment attributes to the node buffer
    int2 vAddress = GetAddress( nNewFragmentAddress );
    tRWFragmentColor[vAddress] = vFragment;
    tRWFragmentDepthAndLink[vAddress] = int2(          int(saturate(input.vPositionSS.z))*0x7fffffff),
    nOldFragmentAddress );

    return;
}
```

***QUESTIONS***



## Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, AMD Radeon, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

© 2011 Advanced Micro Devices, Inc. All rights reserved.