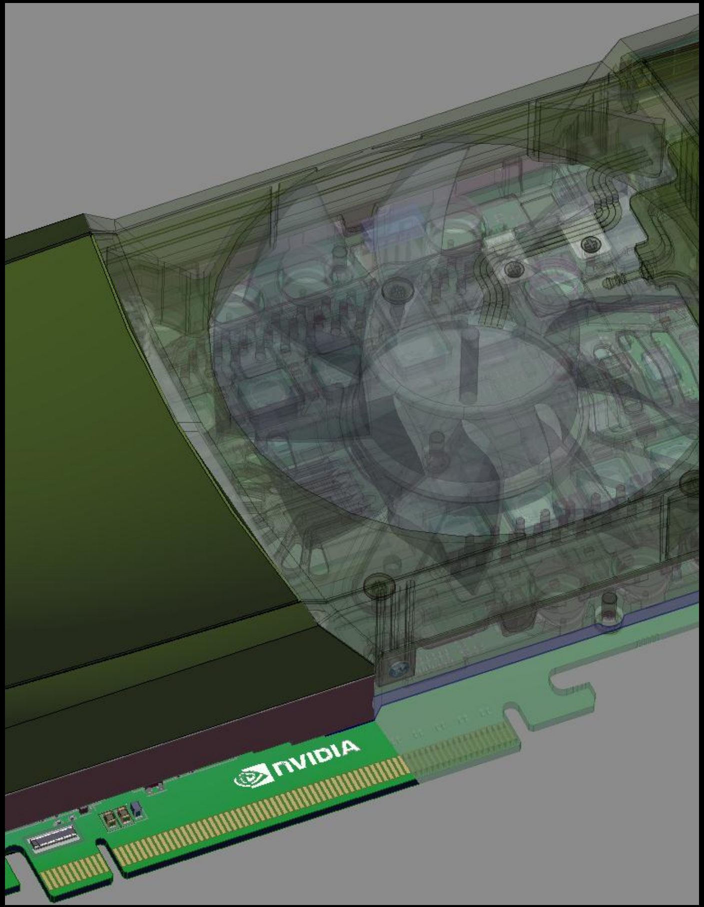
GPU TECHNOLOGY CONFERENCE

Order Independent Transparency In OpenGL 4.x

Christoph Kubisch – ckubisch@nvidia.com

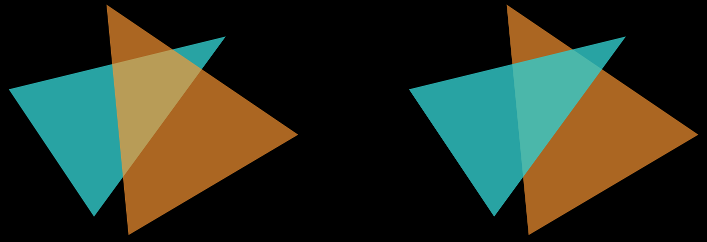
TRANSPARENT EFFECTS

- Photorealism:
  - Glass, transmissive materials
  - Participating media (smoke...)
  - Simplification of hair rendering
- Scientific Visualization
  - Reveal obscured objects
  - Show data in layers

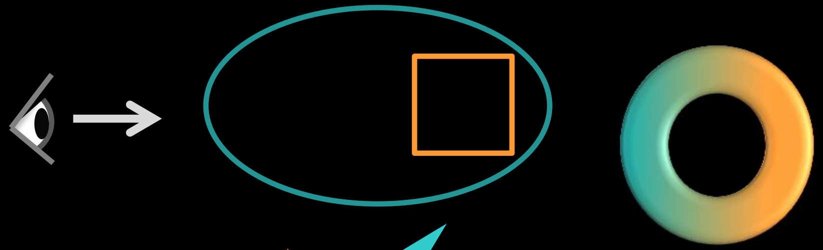


THE CHALLENGE

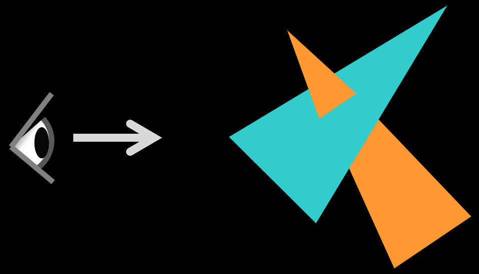
- Blending Operator is not commutative
    - Front to Back
    - Back to Front



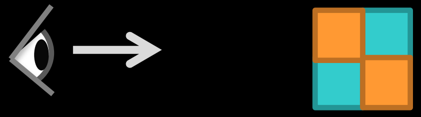
- – Sorting objects not sufficient



- – Sorting triangles not sufficient
    - Very costly, also many state changes



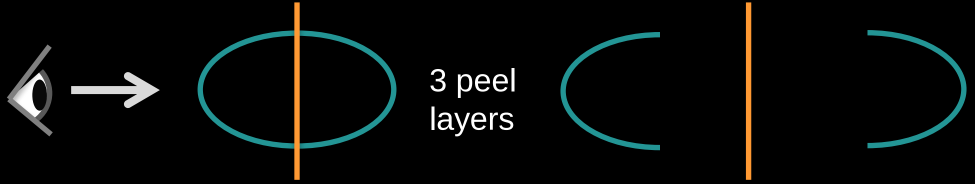
- Need to sort „fragments“



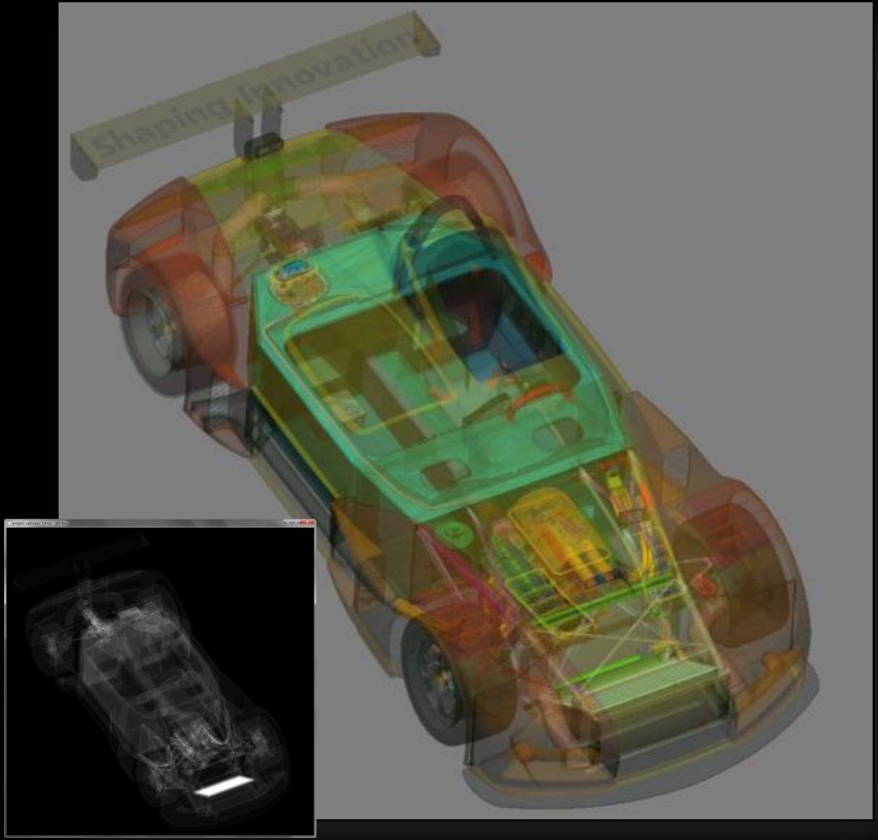
RENDERING APPROACHES

- OpenGL 4.x allows various one- or two-pass variants

- Previous high quality approaches
  - Stochastic Transparency [Enderton et al.]
  - Depth Peeling [Everitt]



  - Caveat: Multiple scene passes required

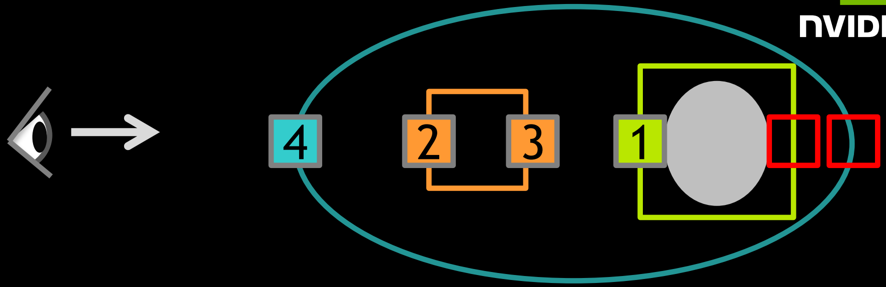


Peak ~84 layers

model courtesy of PTC

RECORD & SORT

- Render Opaque
  - Depth-buffer rejects occluded fragments



```
layout (early_fragment_tests) in;
```

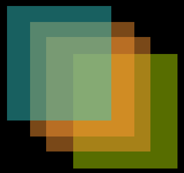
- Render Transparent
  - Record color + depth

1				
				2
			3	
			4	

```
uvec2(packUnorm4x8 (color),
      floatBitsToUint (gl_FragCoord.z) );
```

- Resolve Transparent
  - Fullscreen sort & blend per pixel

1	2	3	4
4	2	3	1



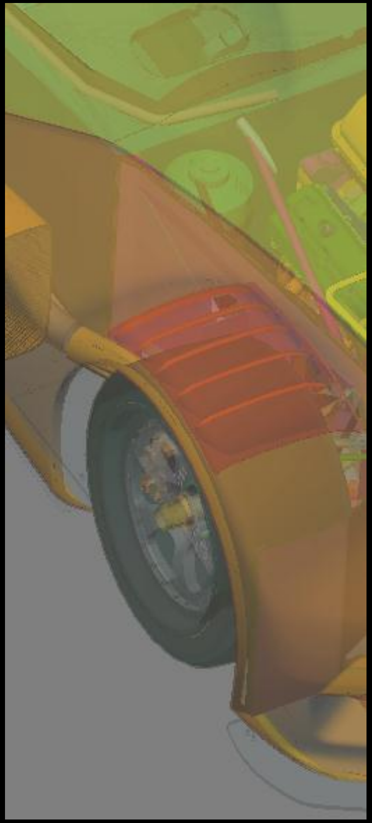
RESOLVE

- Fullscreen pass
  - Not efficient to globally sort all fragments per pixel
  - Sort K nearest correctly via register array
  - Blend fullscreen on top of framebuffer

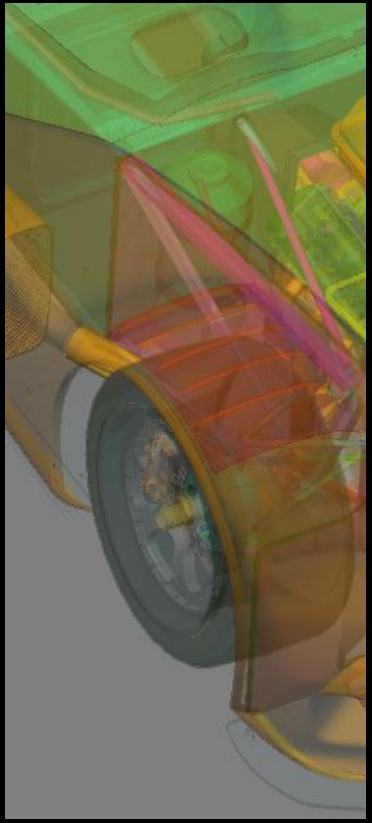
```
uvec2 fragments[K];
// encodes color and depth

n = load (fragments);
sort (fragments,n);

vec4 color = vec4(0);
for (i < n) {
  blend (color, fragments[i]);
}

gl_FragColor = color;
```

TAIL HANDLING

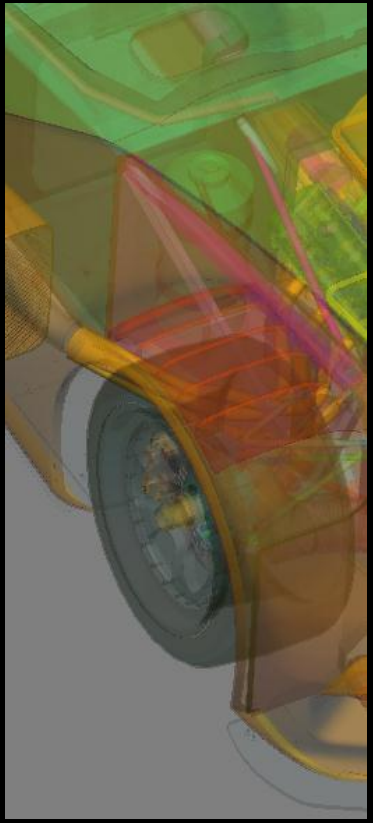
- Tail Handling:
  - Discard Fragments > K
  - Blend below sorted and hope error is not obvious [Salvi et al.]
    - Many close low alpha values are problematic
    - May not be frame-coherent (flicker) if blend is not primitive-ordered



K = 4



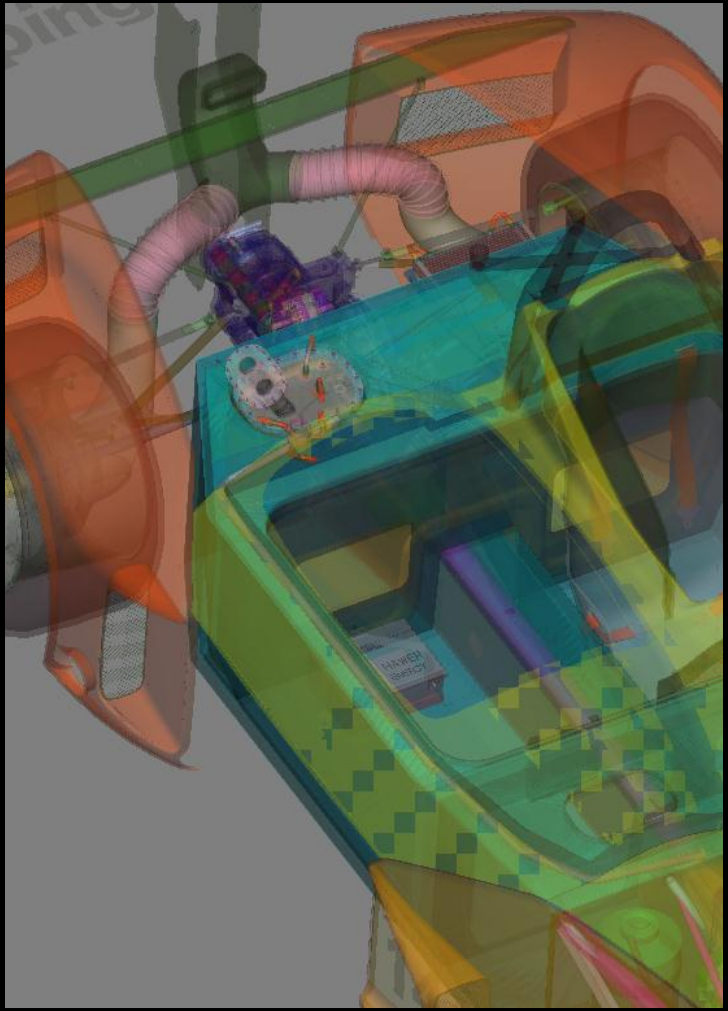
K = 4
Tailblend



K = 16

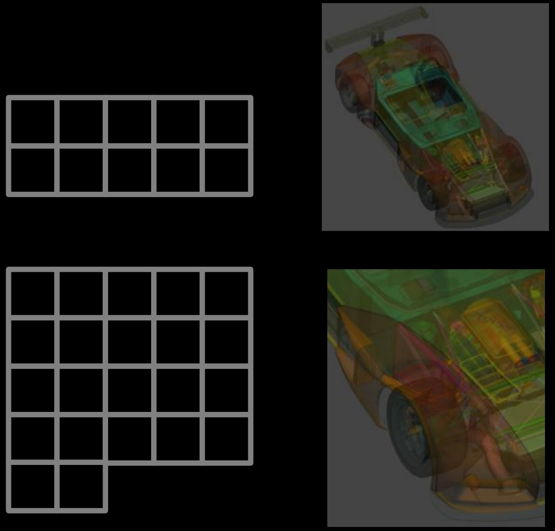
RECORD TECHNIQUES

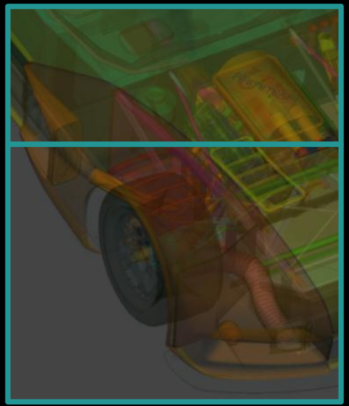
- Unbounded:
  - Record all fragments that fit in scratch buffer
  - Find & Sort K closest later
    - + fast record
    - - slow resolve
    - - out of memory issues



HOW TO STORE

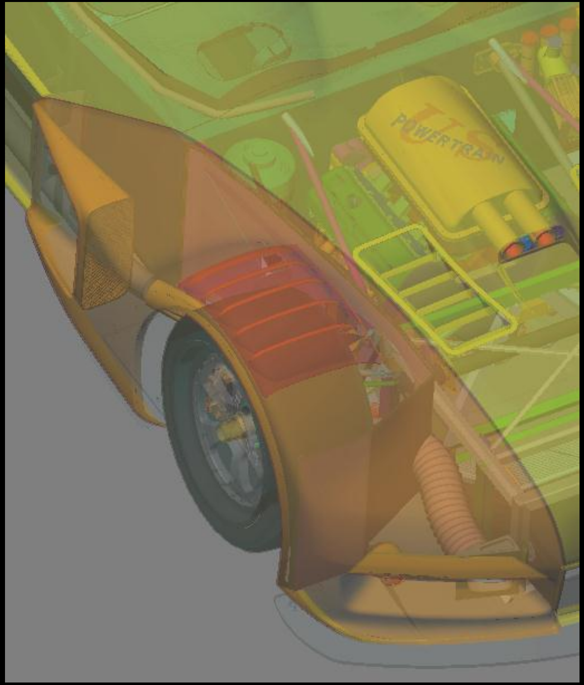
- Unbounded:
  - Resize dynamically based on global counters of past frames (async readback)
    - Avoid glGetBufferData or glMap on counter buffer
    - Use a second dedicated „copy & read“ buffer
  - Consider Tiled Rendering Approach
    - Less overall memory consumption
    - Record & Resolve per Tile



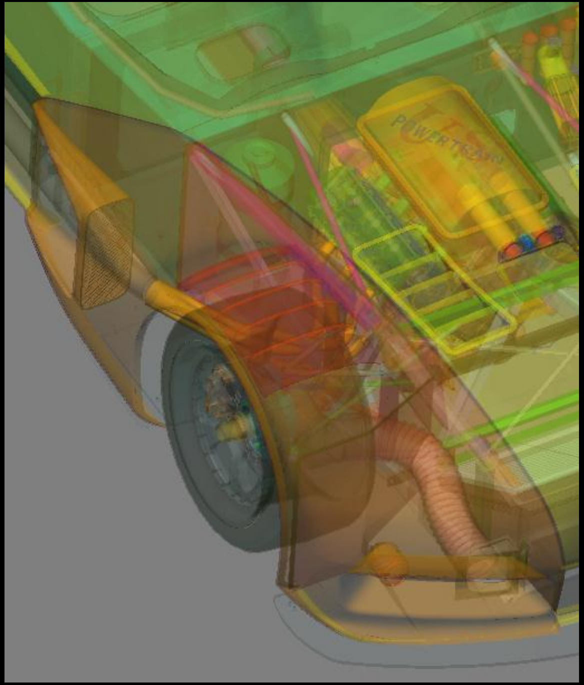


RECORD TECHNIQUES

- Bounded:
  - Record K closest fragments
  - Sort K later
    - - slower record
    - + fast resolve
    - + guaranteed min quality



K = 4

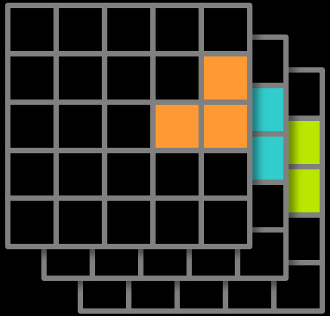


K = 16

HOW TO STORE

- Bounded:
  - Prefer „page“ memory layout

```
listPos(i) =  x + y * width + i * (width * height);
```



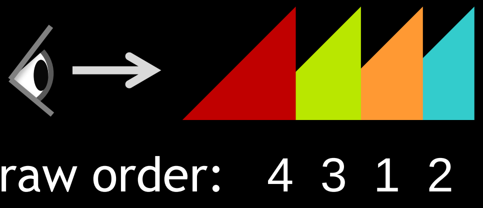
APPROACHES

- Single Pass
  - Simple (least correct)
  - Linked List (unbounded)
  - Spin Lock (not stable)
  - Atomic Loop 64-bit
- Two Pass
  - Offset Array (unbounded)
  - Atomic Loop 32-bit

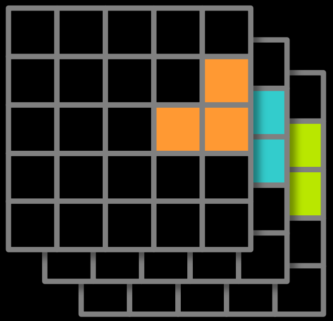
SIMPLE

- Record first K
  - – Highly draw-order dependent
  - – First != nearest
  - – Tail blending not suitful
- Sort & resolve

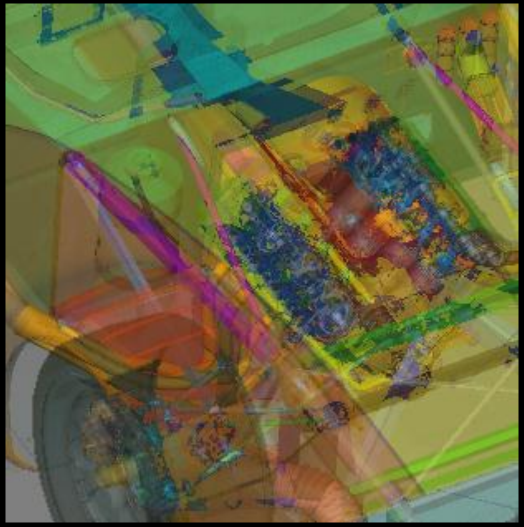
Unsorted	🟧🟦🟩
Sorted	🟩🟧🟦
Sorted + Tail	🟩🟧🟦 🟥



Draw order: 4 3 1 2



K = 3



K = 16

SIMPLE

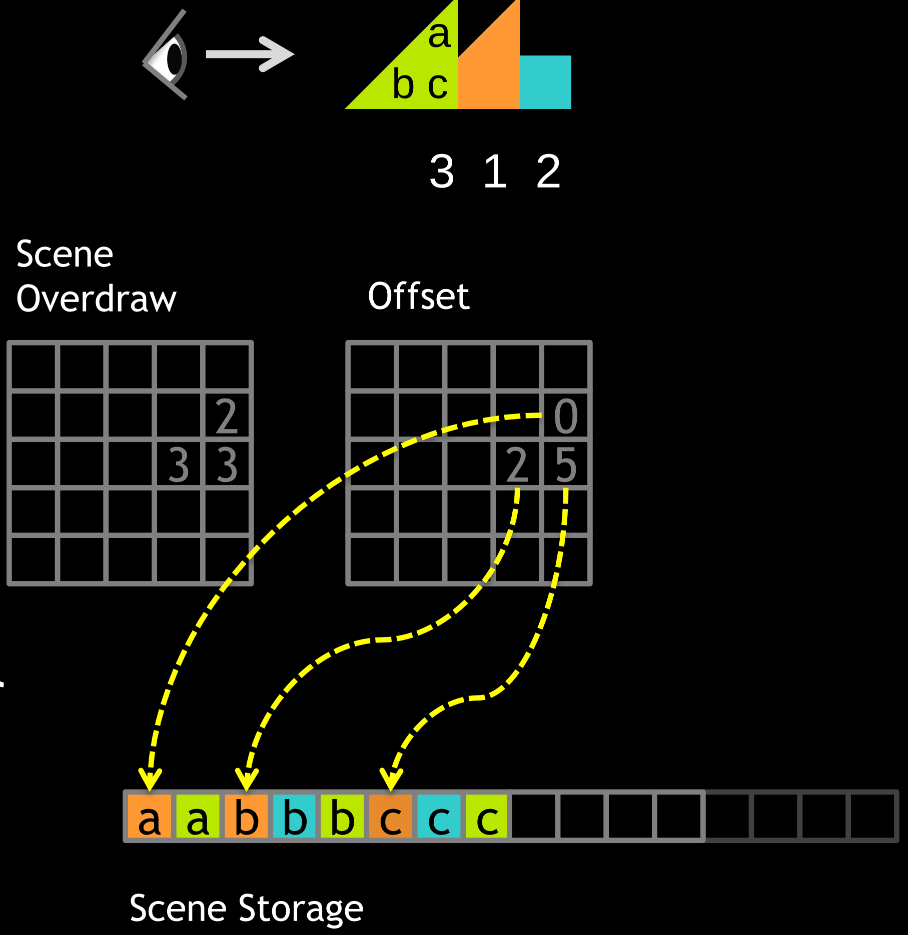
- Record

```
layout (early_fragment_tests) in;

layout(rg32ui) uniform coherent uimageBuffer imgAbuffer;
layout(r32ui)  uniform coherent uimage2D imgCounter;
...

 uint oldCounter = imageAtomicAdd (imgCounter, coord, 1u);


 if ( oldCounter < K ){
    imageStore ( imgAbuffer, listPos (oldCounter),
                 fragment);
 }
```

OFFSET ARRAY

[Knowles et al.]

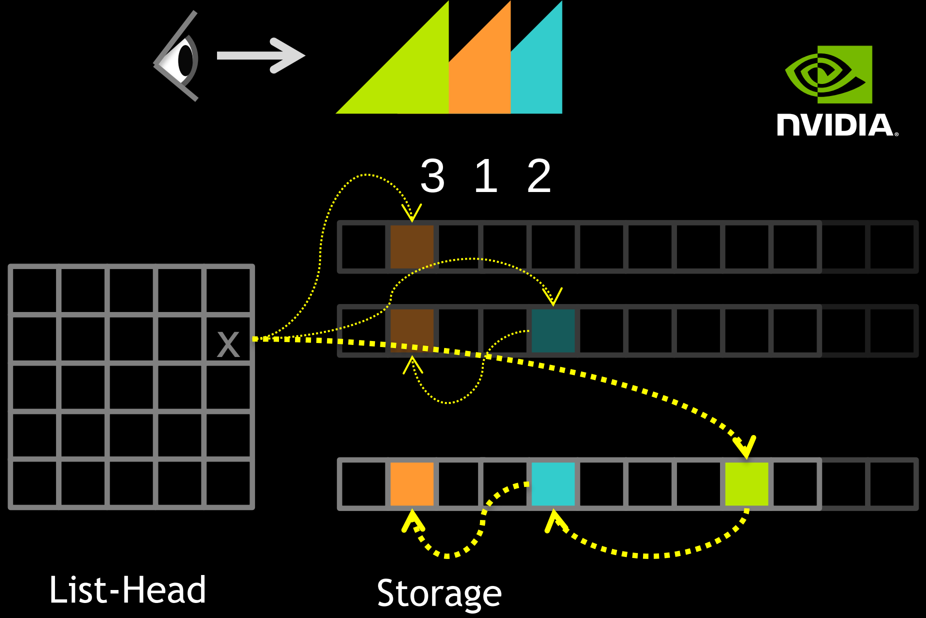
- Count per-pixel overdraw
  - Can use stencil integer texture access for counting
- Generate offsets
- Record lists
  - Requires two geometry passes
  - Can be modified easily for global sort



LINKED LIST

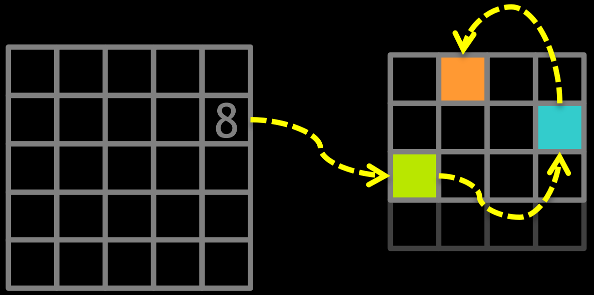
[Yang et al.]

- Try record all
  - Global counter for storage index
  - Storage buffer: fragment + previous
  - Per-pixel list-head



```
...
layout (offset=0,binding=0) uniform atomic_uint counter;

 uint idx = atomicCounterIncrement (counter) + 1u; // zero is list terminator

 if (idx < imageSize(imgAbuffer) ){
   uint prev = imageAtomicExchange (imgListHead, coord, idx);
   imageStore (imgAbuffer, idx, uvec4 (fragment,0, prev ));
 }
```

LINKED LIST



- Resolve
  - Costly, need to run through full list
  - May need insertion sort if K < list

```
idx = getListHead (coord);
while (idx && i < K){
  fragments[i++] = getStored (idx);
  idx = getNext (idx);
}

// beneficial for short lists (majority)
sort (fragments, i);

while (idx) {
  insertionSort (fragments, getStored (idx));
  idx = getNext (idx);
}
...
```

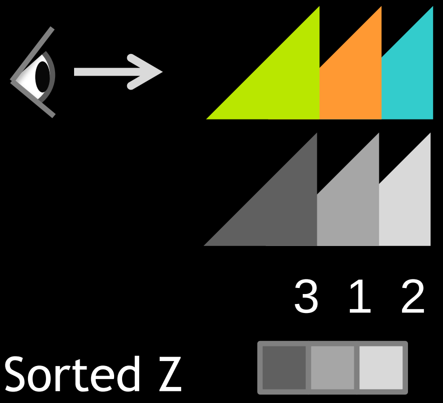
SPIN LOCK

- Manual critical section
  - Record K closest per-pixel
  - not stable (flickers)
  - Often slowest!
  - NOT RECOMMENDED

```
... imgAbuffer;
... imgCounter;
... imgLock;

#extension GL_NV_shader_thread_group : require

// pre-test againt furthest element, skip lock

bool done = gl_HelperThreadNV;
while (!done) {
  if (imageAtomicExchange (imgLock, coord, 1u) == 0u) {
    // add to list or
    // find and replace furthest element in list
    // flicker: list updates not guaranteed consistent
    ...
    // leave section
    imageStore (imgLock, coord, uvec4 (0));
    done = true;
  }
}
```

ATOMIC LOOP 32-BIT

[Liu et al.]

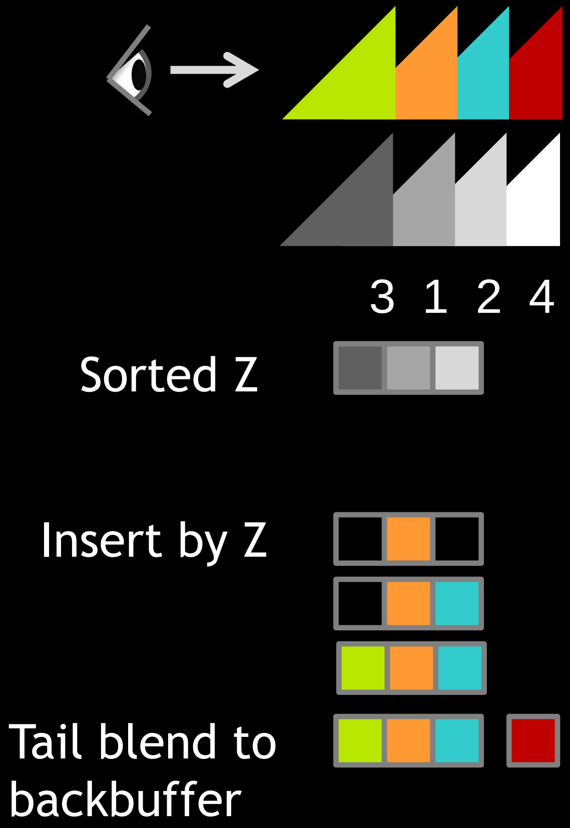


- Two-pass record
  - First Pass: find K closest depth values

```
uint ztest = floatBitsToUint (gl_FragCoord.z);

// for speed-up test against last/middle element of list
// if too far, skip below or start at middle

for ( i < K; i++) {
  uint zold = imageAtomicMin (imgZbuffer, listPos(i), ztest);
  if  (zold == 0xFFFFFFFFu || zold == ztest){
    break;
  }
  ztest = max (zold, ztest);
}
```

ATOMIC LOOP 32-BIT

- Second Pass
  - Insert color based on depth with binary search
  - Tail blend is stable (primitive-order obeyed)
- Resolve
  - Simple already sorted



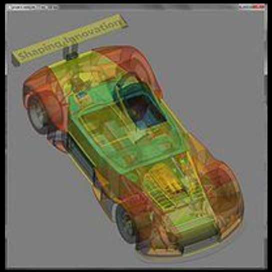
ATOMIC LOOP 64-BIT

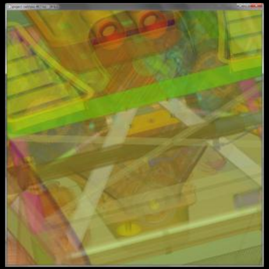
- GK110 and Maxwell
  - NV_shader_atomic_int64 (upcoming) allows single pass!
  - Color in lower-bits (uint64_t via NV_gpu_shader5)

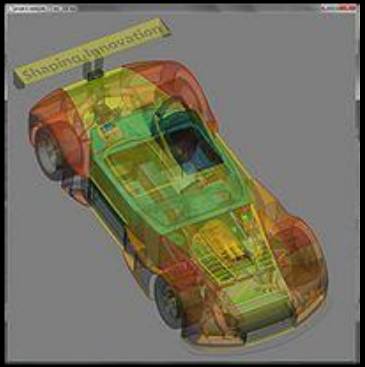
```
buffer myabuffer { uint64_t ssboAbuffer[]; };
...
uint64_t ftest = packUint2x32 (color_as_uint32, z_as_uint32);

for ( i < K; i++) {
  uint64_t fold = atomicMin (ssboAbuffer[listPos(i)], ftest);
  if (hi32(fold) == 0xFFFFFFFFu || hi32(fold) == hi32(ftest) ){
    break;
  }
  ftest = max (fold, ftest);
}
```

PERFORMANCE

- Quadro K6000, 1024 x 1024, GL_RGBA16F
- CAD data and hair
- Varying K, K = 8 often good quality/perf
- Tailblend always on
- Linked List (unbounded)
  - Resized buffer to hold all data
- Offset Array (unbounded)
  - Resized, however capped at 255 overdraw (8-bit stencil)
- „Simple“ approach mostly unreliable due to overdraw

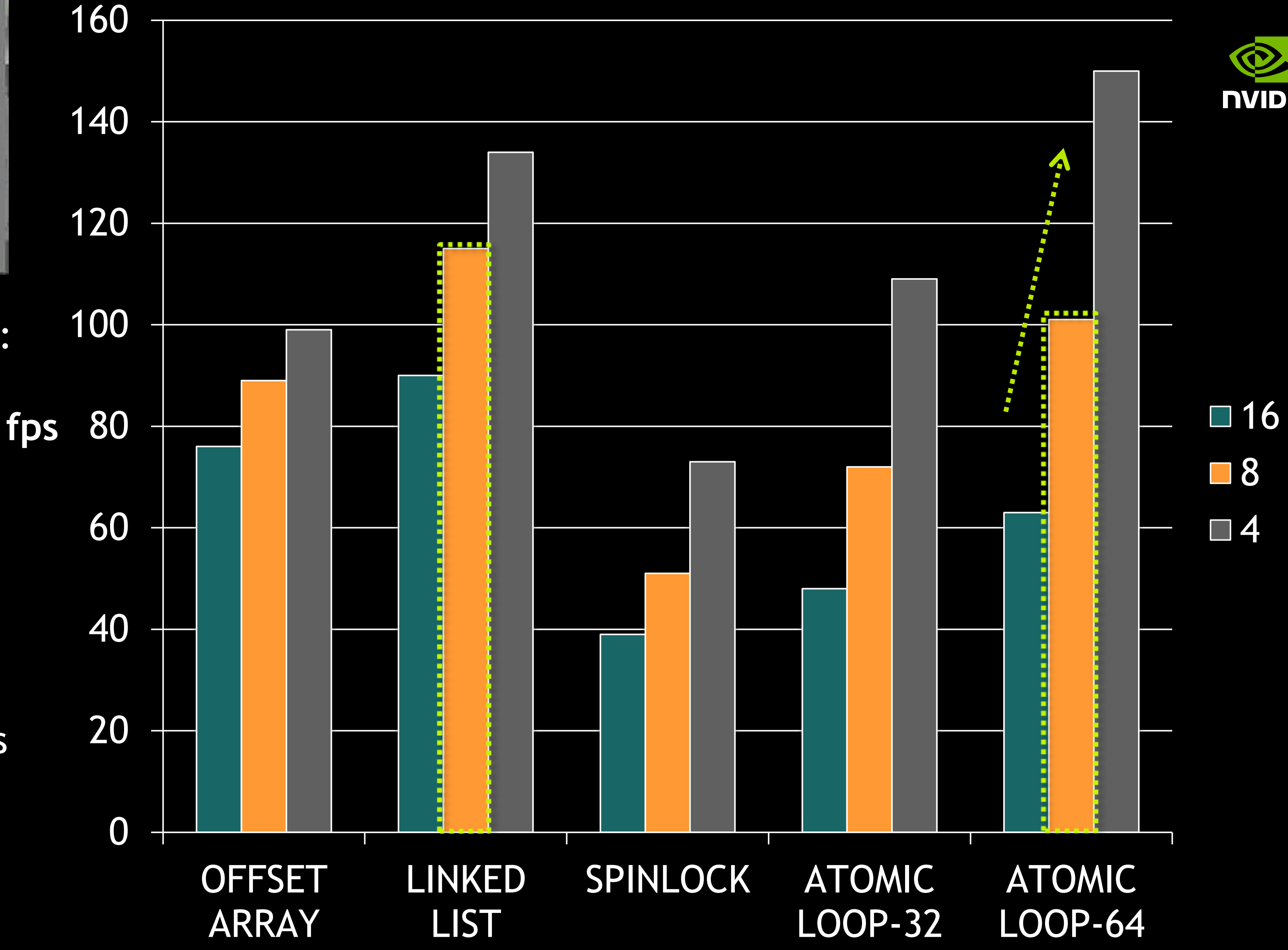


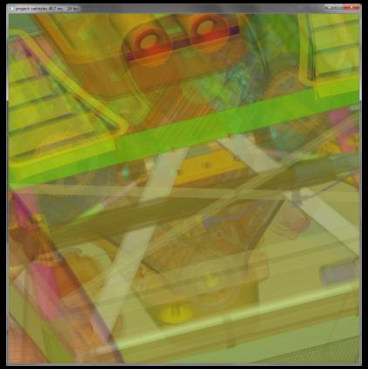




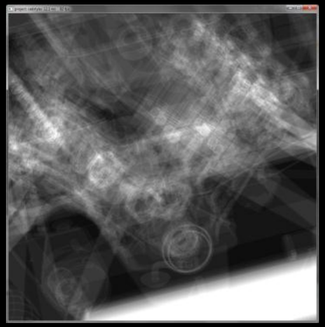
Full global sort:
15 fps

Peak ~84 layers

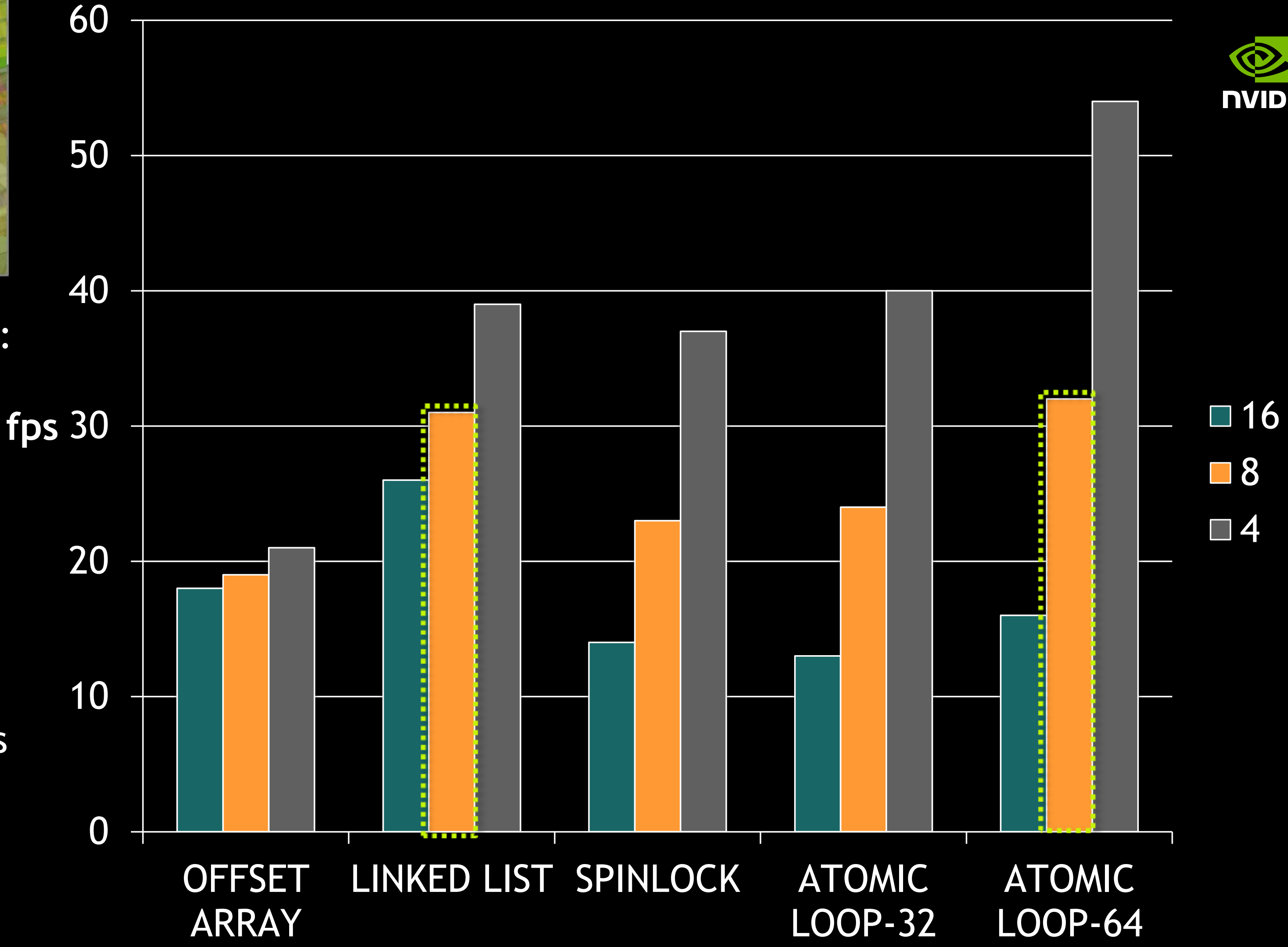




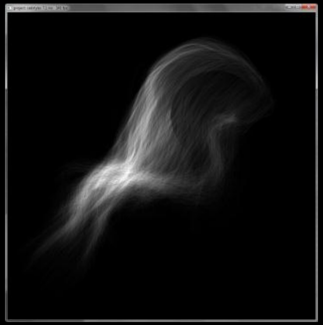
Full global sort:
2 fps



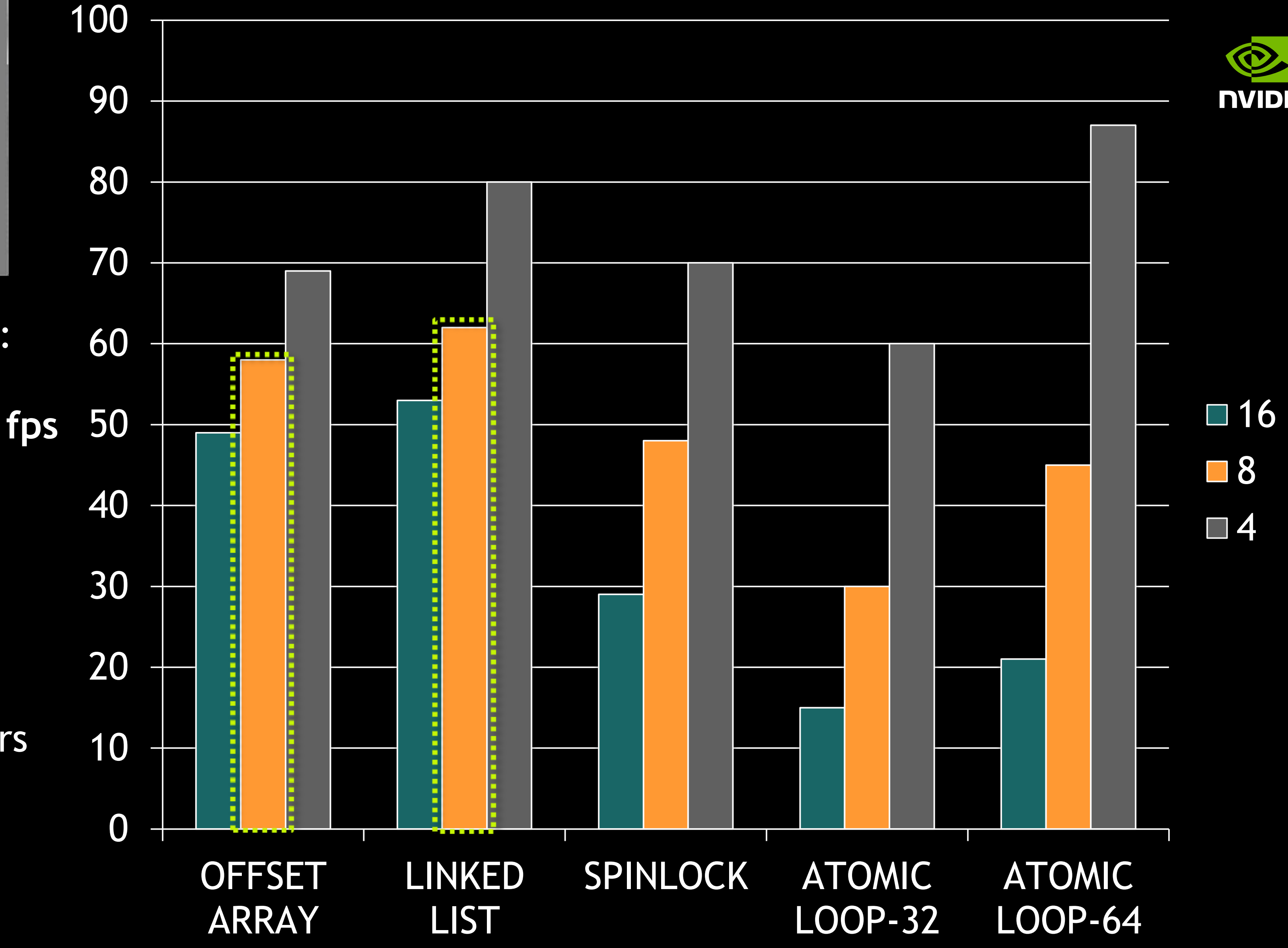
Peak ~74 layers



Full global sort:
4 fps



Peak ~150 layers

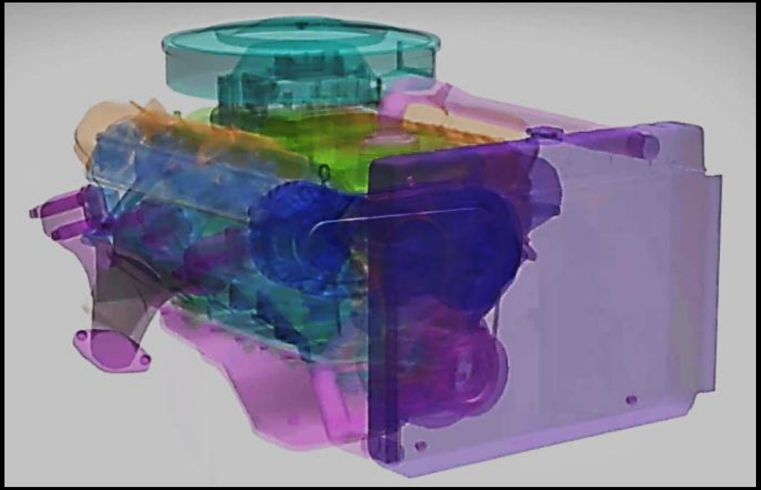


CONCLUSION

- Linked List and Atomic Loop approaches work well
  - 32 – Bit Loop can work well with fast depth-pass (stable tailblend)
  - 64 – Bit Loop for GK110 and Maxwell
- Even simple approach might be sufficient if max depth complexity is known
- Thou shalt not forget „early_fragment_tests“ ☺
  - Otherwise depth-test done „after“ record shader

ALTERNATIVE

- Use commutative blend function
  - Very fast solution (uses mostly classic blendFuncs)
  - Weighted Blended Order-Independent Transparency [McGuire et al.]
  - http://jcgt.org/published/0002/02/09/



THANK YOU & REFERENCES

- Weighted Blended Order-Independent Transparency
  - Morgan McGuire and Louis Bavoil
  - http://jcgt.org/published/0002/02/09/
- Multi-Layer Alpha Blending
  - Marco Salvi and Karthik Vaidyanathan
  - http://software.intel.com/en-us/articles/multi-layer-alpha-blending
- Efficient Layered Fragment Buffer Techniques
  - Pyarelal Knowles, Geoff Leach, and Fabio Zambetta
  - http://openglinsights.com/bendingthepipeline.html#EfficientLayeredFragmentBufferTechniques
- Freepipe: programmable parallel rendering architecture for efficient multi-fragment effects
  - Fang Liu, Mengcheng Huang, Xuehui Liu and Enhua Wu
  - https://sites.google.com/site/hmcen0921/cudarasterizer
- k+-buffer: Fragment Synchronized k-buffer
  - Andreas A. Vasilakis, Ioannis Fudos
  - http://www.cgrg.cs.uoi.gr/wp-content/uploads/bezier/publications/abasilak-ifudos-i3d2014/k-buffer.pdf
- Real-time concurrent linked list construction on the GPU
  - Jason C. Yang, Justin Hensley, Holger Grün and Nicolas Thibieroz
  - http://dl.acm.org/citation.cfm?id=2383624
- Stochastic Transparency
  - Eric Enderton, Erik Sintorn, Peter Shirley and David Luebke
  - http://www.nvidia.com/object/nvidia_research_pub_016.html
- Interactive order-independent transparency (Depth Peeling)
  - Cass Everitt
  - https://developer.nvidia.com/content/interactive-order-independent-transparency