Order Independent Transparency via Linked List Sorting

Gang Liao

Computer, Electrical and Mathematical Sciences & Engineering Division
King Abdullah University of Science and Technology

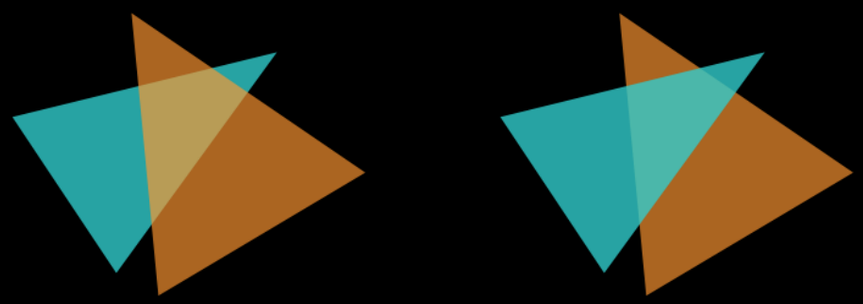
Motivation

Transparent Effects:

- Photorealism
- Scientific Visualization

The Challenge:

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



flickr.com

Sorting Strategy

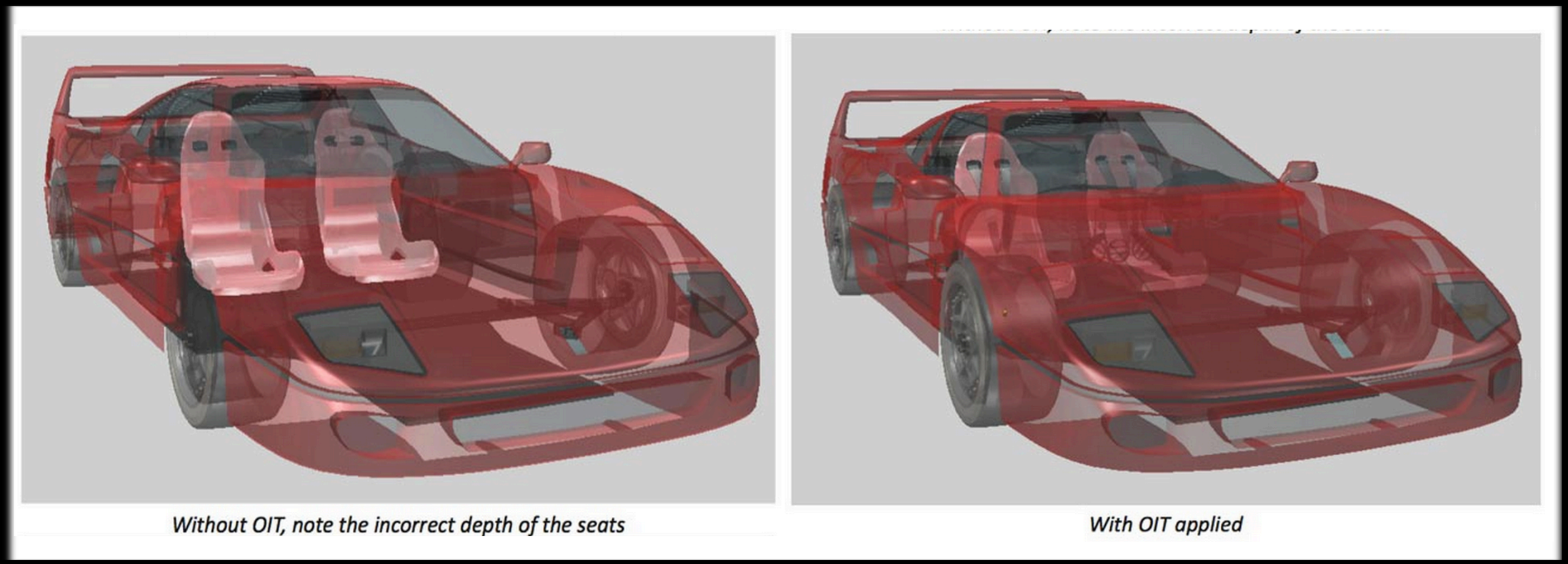
How to Sort?

- Sorting objects not sufficient

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

- ☑ Sorting fragments

Order Independent Transparency



Without OIT, note the incorrect depth of the seats

With OIT applied

The most common one is to keep a list of colors for each pixel, sort them by depth, and then blend them together in the fragment shader.

Algorithm Overview

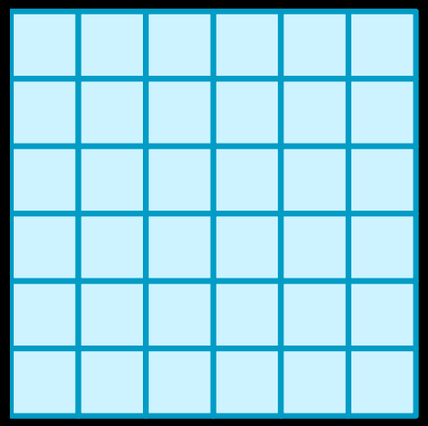
- Render opaque scene objects.
- Render transparent scene objects.
  - All fragments are stored using per-pixel linked lists.
  - Store fragments: color, alpha, & depth.
- Screen quad resolves and composites fragment lists.
  - each pixel in fragment shader sorts associated linked list.
  - blending fragments in sorted order with background.
  - output final effects.

Linked List Creation (1)

start offset buffer

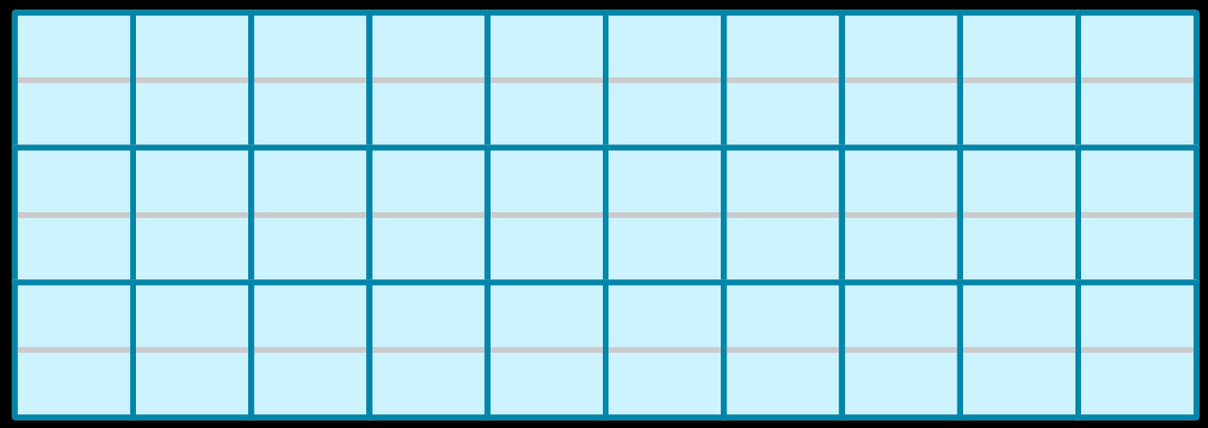
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1

viewport



Counter = 0

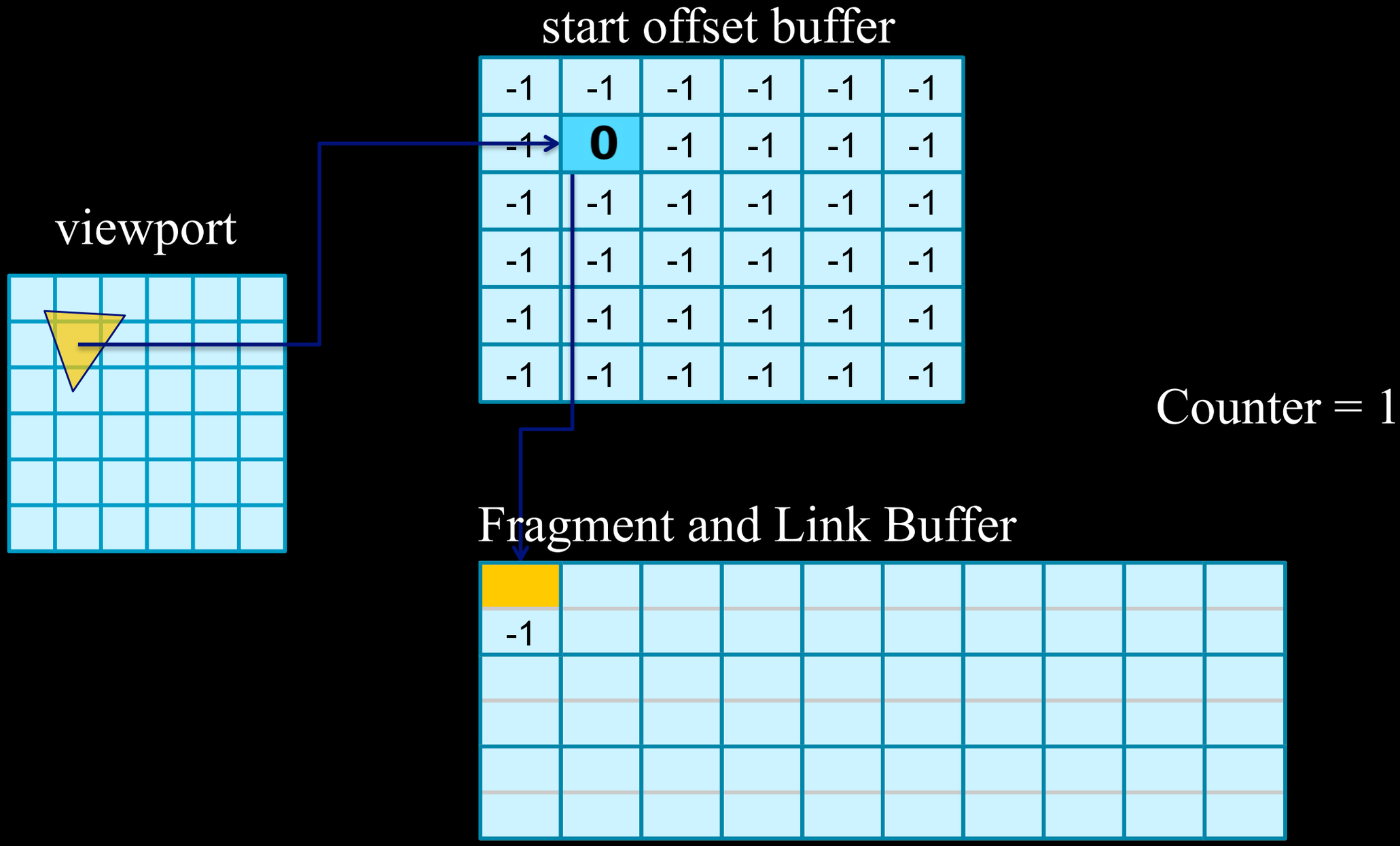
Fragment and Link Buffer



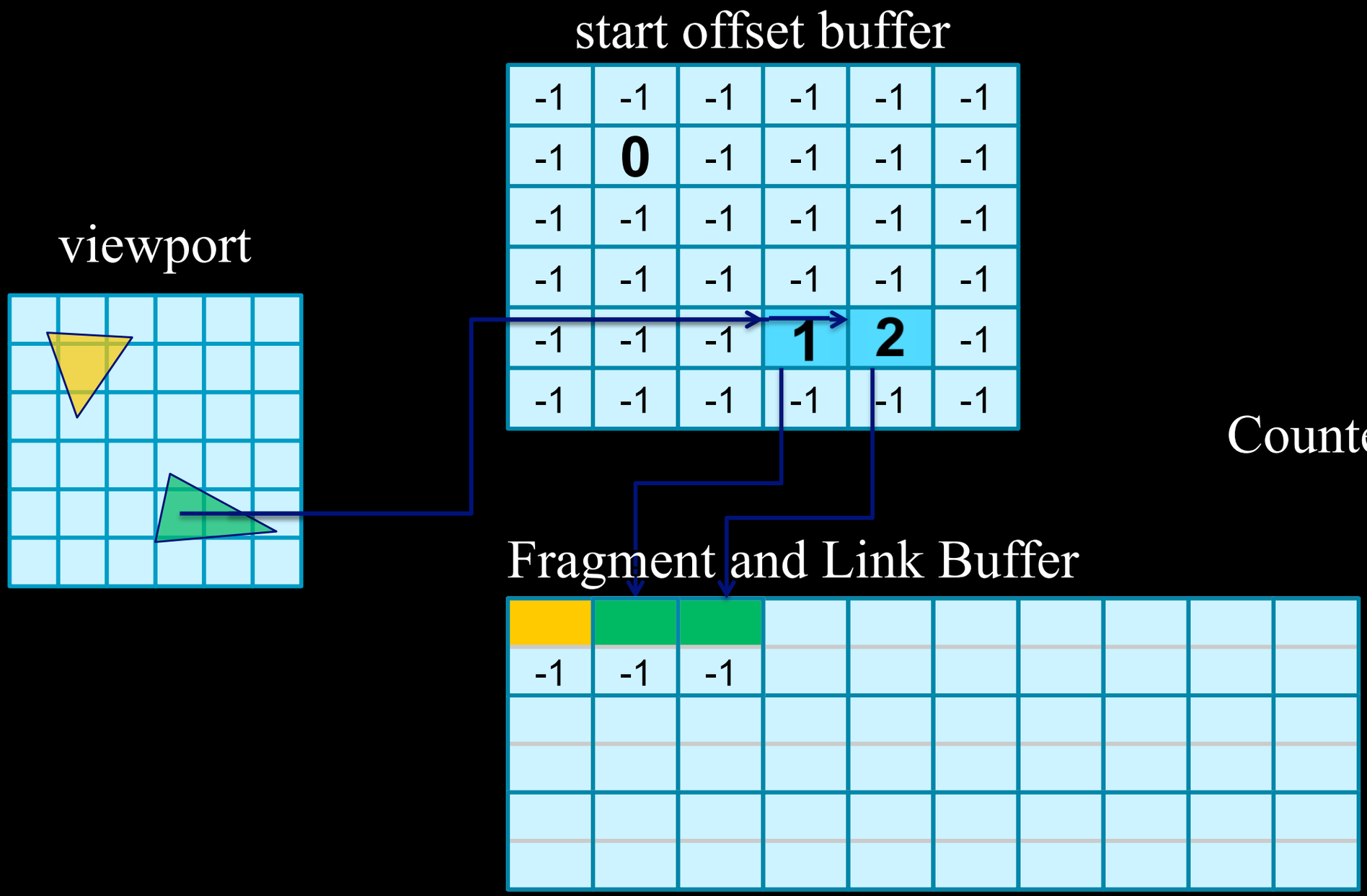
Atomic counter and Buffers

```
1  enum BufferNames {
2    COUNTER_BUFFER = 0,
3    LINKED_LIST_BUFFER
4  };
5  ...
6   // Our atomic counter
7   glBindBufferBase(GL_ATOMIC_COUNTER_BUFFER, 0, buffers[COUNTER_BUFFER
       ]);
8   glBufferData(GL_ATOMIC_COUNTER_BUFFER, sizeof(GLuint), NULL,
       GL_DYNAMIC_DRAW);
9
10  // The buffer for the head pointers, as an image texture
11  glGenTextures(1, &headPtrTex);
12  glBindTexture(GL_TEXTURE_2D, headPtrTex);
13  glTexStorage2D(GL_TEXTURE_2D, 1, GL_R32UI, width, height);
14  glBindImageTexture(0, headPtrTex, 0, GL_FALSE, 0, GL_READ_WRITE,
       GL_R32UI);
15
16  // The buffer of linked lists
17  glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 0, buffers[
       LINKED_LIST_BUFFER]);
18  glBufferData(GL_SHADER_STORAGE_BUFFER, maxNodes * nodeSize, NULL,
       GL_DYNAMIC_DRAW);
```

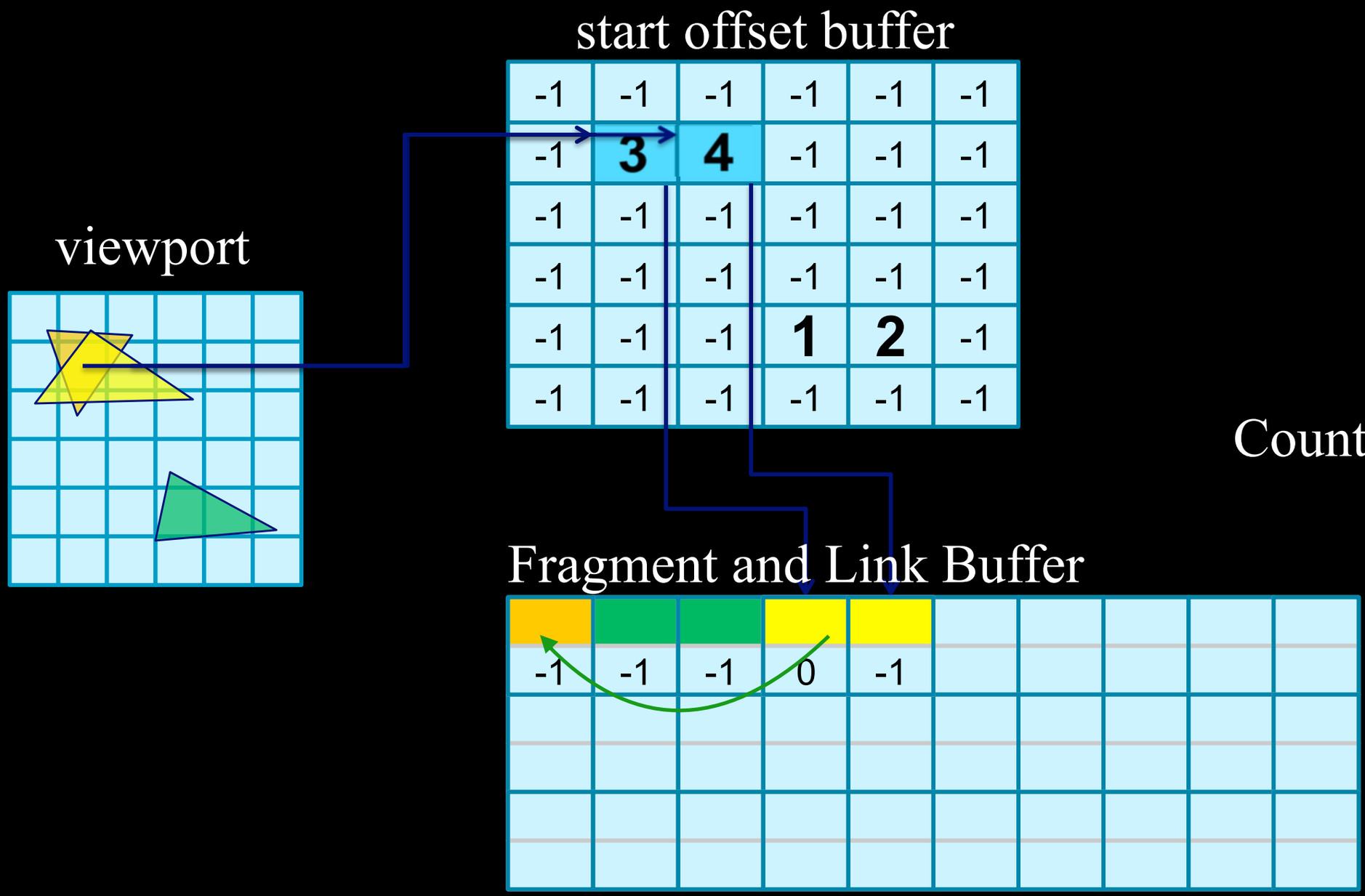
Linked List Creation (2)



Linked List Creation (3)



Linked List Creation (4)



Atomic Operations and Synchronization

Atomic Operations

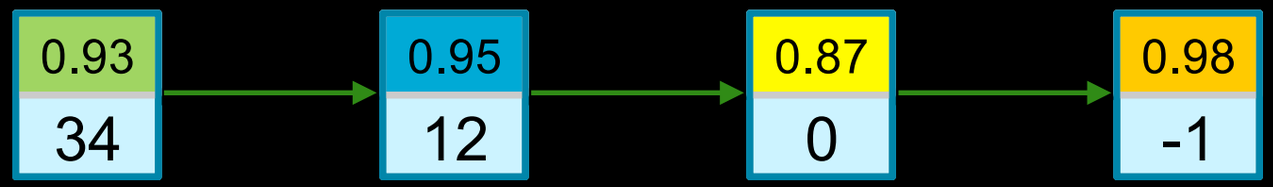
```
// Get the index of the next empty slot in the buffer
 uint nodeIdx = atomicCounterIncrement(nextNodeCounter);
```

```
uint prevHead = imageAtomicExchange(headPointers, ivec2(gl_FragCoord.
    xy), nodeIdx);
nodes[nodeIdx].color = vec4(diffuse(), Kd.a);
nodes[nodeIdx].depth = gl_FragCoord.z;
nodes[nodeIdx].next = prevHead;
```

Synchronization

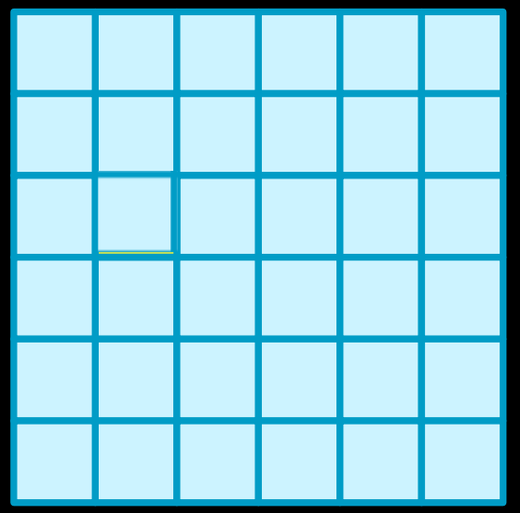
```
glMemoryBarrier( GL_SHADER_STORAGE_BARRIER_BIT );
```

Linked List Sorting (1)



FS color

Render Target

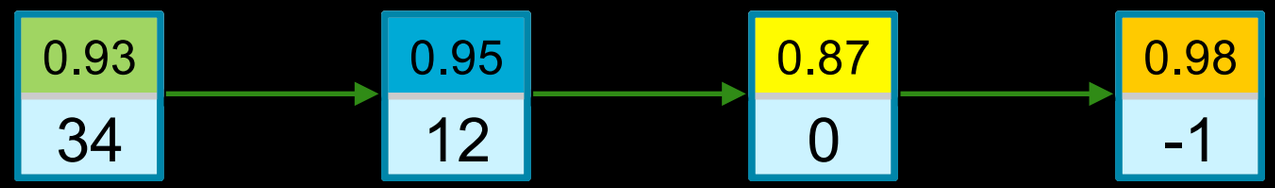


Transferring Temp Array

For each pixel, according to the coordinate, we can get the start index in the linked list buffer. If index is not equal to -1 (unsigned int: 0xffffffff), the nodes of each linked list can be stored into an array.

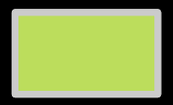
```
1 // Copy the linked list for this fragment into an array
2  while( n != 0xffffffff && count < MAX_FRAGMENTS) {
3    frags[count] = nodes[n];
4    n = frags[count].next;
5    count++;
6  }
```

Linked List Sorting (2)



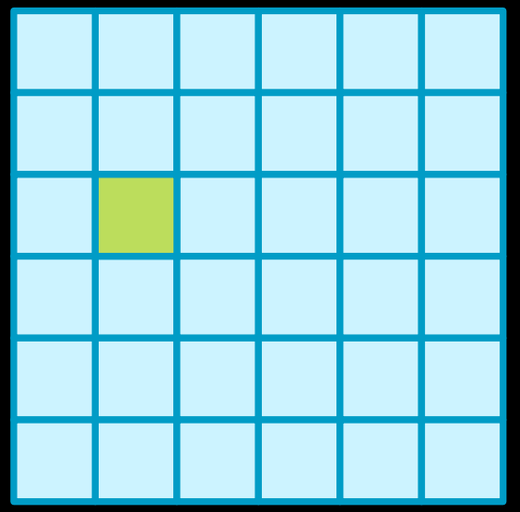
Temp Array

Background color



FS color

Render Target



Sorting Algorithm

Implemented Sorting Algorithm:

- [x] Insertion Sort
- [x] Selection Sort
- [x] Bubble Sort
- [x] Shell Sort
- [x] Merge Sort (iteration version !!!)
- [ ] Quick Sort (TBD)
- [ ] Radix Sort (TBD)

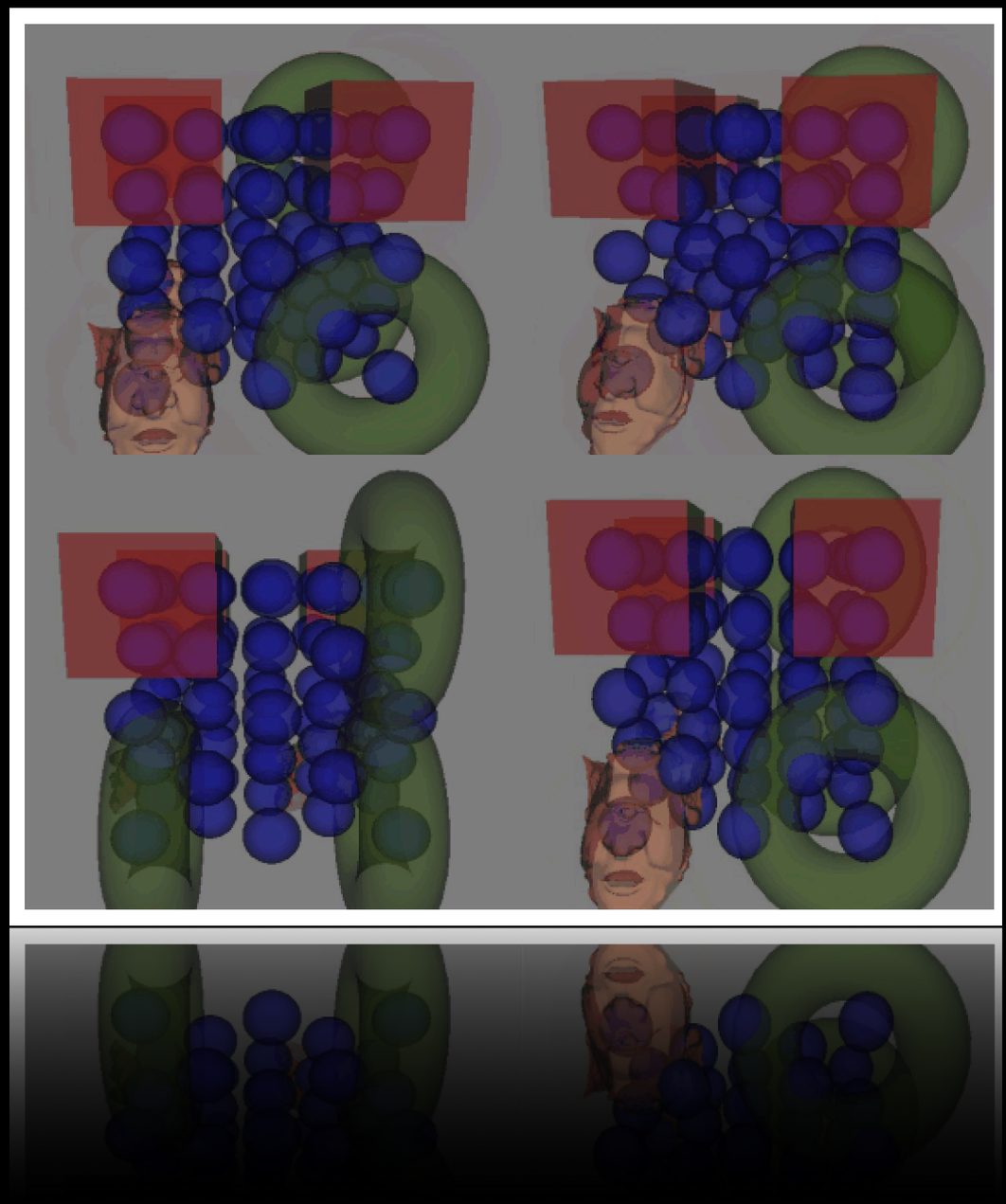
Iteration Merge Sort

```
    int i, j1, j2, k;
    int a, b, c;
    int step = 1;
    NodeType leftArray[MAX_FRAGMENTS/2]; //for merge sort

    while (step <= count)
    {
        i = 0;
        while (i < count - step)
        {
            a = i;
            b = i + step;
            c = (i + step + step) >= count ? count : (i + step + step);

            for (k = 0; k < step; k++)
                leftArray[k] = frags[a + k];

            j1 = 0;  j2 = 0;
            for (k = a; k < c; k++)
            {
                if (b + j1 >= c || (j2 < step && leftArray[j2].depth >
                    frags[b + j1].depth))
                     frags[k] = leftArray[j2++];
                else
                     frags[k] = frags[b + j1++];
            }
            i += 2 * step;
        }
        step *= 2;
    }
```

DEMO

The Platform

- ☑ GTX 750M
- ☑ Visual Studio 2013
- ☑ OpenGL 4.3 or later



Thank you & Reference

- ☑ Christoph Kubisch, *Order Independent Transparency In OpenGL 4.x*, GTC 2014
- ☑ Pal Barta, Balazs Kovacs, *Order Independent Transparency with Per-Pixel Linked Lists*, CESCG 2011
- ☑ David Wolff, *OpenGL 4.0 Shading Language Cookbook*
- ☑ Jay McKee, *Real Time Concurrent Linked List Construction on the GPU*, AMD Technique Notes
- ☑ Nicolas Thibieroz, *Order-Independent Transparency using Per-Pixel Linked Lists*, EGSR 2010