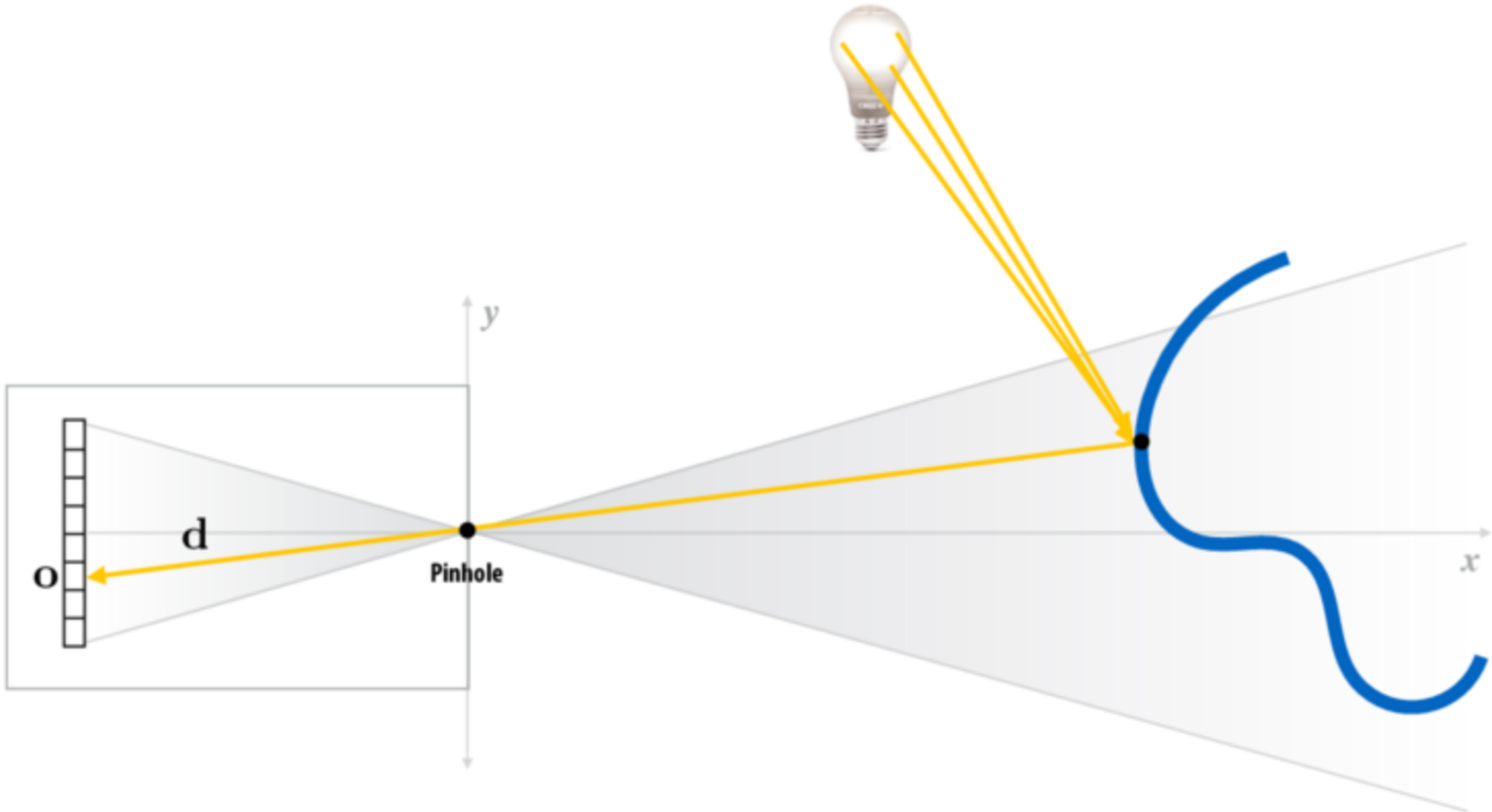
Distributed GPU Ray Tracing

Ye Yuan (yyuan2@andrew)
Ken Ling (kling1@andrew)

Parallel Tree Construction

What is Ray Tracing



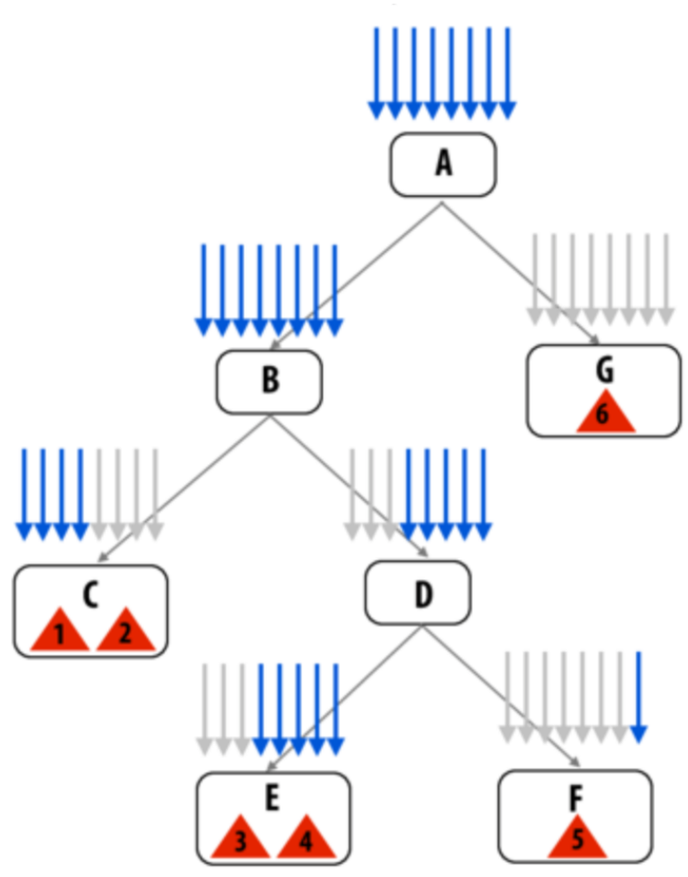

```
For each pixel p
    color(p) = traceRay(cameraRay of p)
```

Image credit to Prof. Fatahalian

Parallel Ray Tracing

```
traceRay(ray)
    intersect = traverseRay(ray, BVH)
    if(intersect)
        shadowRay = sampleLight
        intersect = traverseRay(shadowRay, BVH)
        if(!intersect)
            calculate direct lighting
        ray = generateSecondaryRay
        if(ray.depth < max_depth)
            traceRay(ray)
```

Divergence!



Assign each camera ray to a thread

Image credit to Prof. Fatahalian

GPU Ray Tracing Optimization

Persistent thread for better work distribution

```
PersistentWorkerThread
    __shared__ local
    while(true)
       if(threadId.x == 0)
           blockNextRay = atomicAdd(globalNextRay, BLOCK_SIZE * K)
           blockRayCount = BLOCK_SIZE * K
    __syncthreads()

    threadNextRay = blockNextRay + threadId.x
    if(threadNextRay > GlobalRayCount) return;

    if (threadId.x == 0)
        blockNextRay += BLOCK_SIZE
        blockRayCount -= BLOCK_SIZE
    __syncthreads()

    ray = fetchRay(threadNextRay)
    traverseRay
```

Problem with recursive traversal

```
traverseRay(ray, BVHnode)
    if(node is a leaf)
        for all triangle in node
            do ray-triangle test
    else
        if(ray intersect node->lChild.bbox)
            traverseRay(ray, node->lChild)
        if(ray intersect node->rChild.bbox)
            traverseRay(ray, node->rChild)
```

Divergence!

Solution - stack

```
traverseRay(ray, BVH node)
    Stack S
    while(node)
        if(node is a leaf)
            for all triangle in node
                do ray-triangle test
        else
            if(ray intersect node->lChild.bbox)
                S.push(node->lChild)
            if(ray intersect node->rChild.bbox)
                S.push(node->rChild)
            node = S.pop()
```

Divergence!

while-while structure

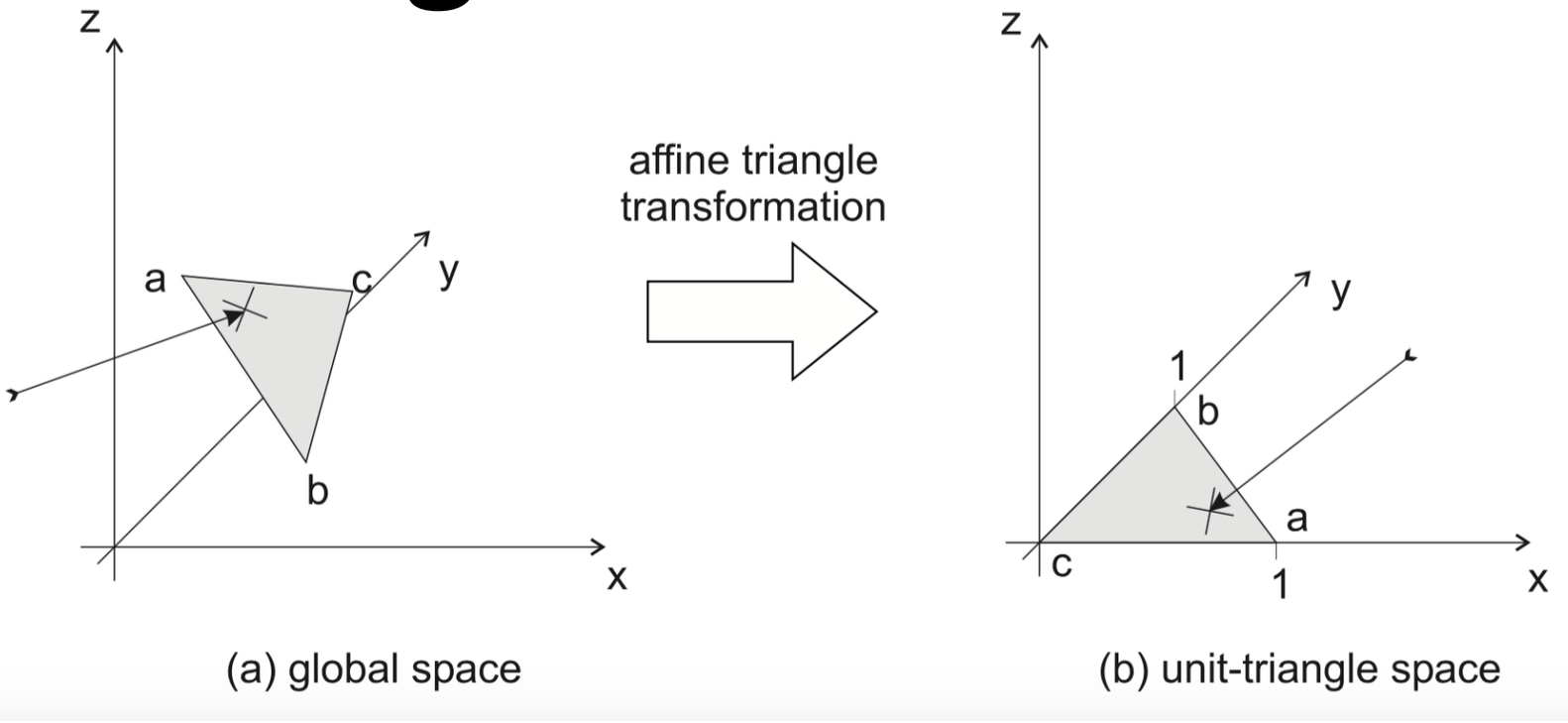
```
traverseRay(ray, BVH node)
   while(ray not terminated)
      while(node is not a leaf)
           traverse ray
      while(triangle to be test)
           do triangle test
```

speculative method

When finding a leaf, keep on traversing until all lanes find a leaf.

Full SIMD utilization in the first while loop!

Unit triangle intersection test



(a) global space (b) unit-triangle space

Require pre computation of transformation.

Perform less computation in the test.

GPU Ray Tracing Result

Scene	TriNum	Primary (M ray/s)	Ambient Occlusion (M ray/s)	Secondary (M ray/s)	GPU FPS	CPU FPS	GPU rendering time 256 sample (s)	CPU rendering time 256 sample (s)	Speedup against single thread
CBcoil	7884	321.97	129.76	34.33	67.32	0.48	3.80	537.32	141x
CBdragon	100012	244.02	161.94	24.45	67.56	0.53	3.79	483.33	128x
Dragon	105120	193.54	92.05	53.92	53.31	0.30	4.80	850.07	177x
CBlucy	133796	276.40	98.83	30.11	56.37	0.42	4.54	608.03	134x
Blob	196608	159.32	53.54	33.92	36.53	0.26	7.01	993.97	142x
Wall-e	240326	87.03	45.56	11.12	25.63	0.17	9.99	1492.79	149x

Single threaded CPU. GTX 680 GPU.

FPS denotes the frame rate of rendering a 1000x1000 picture with shadow using 1 sample per pixel.

Parallel Tree Construction

Traditional Tree Construction is inherently inefficient for Parallelization

- Sequential Dependency
  - Level 2 depends on level 1

- Low Utilization
  - Tree height is limited.
  - For many levels of tree, the small width leads to very low utilization.
  - GTX 680 has 16K threads.
  - On 14th level, there are 16K nodes.

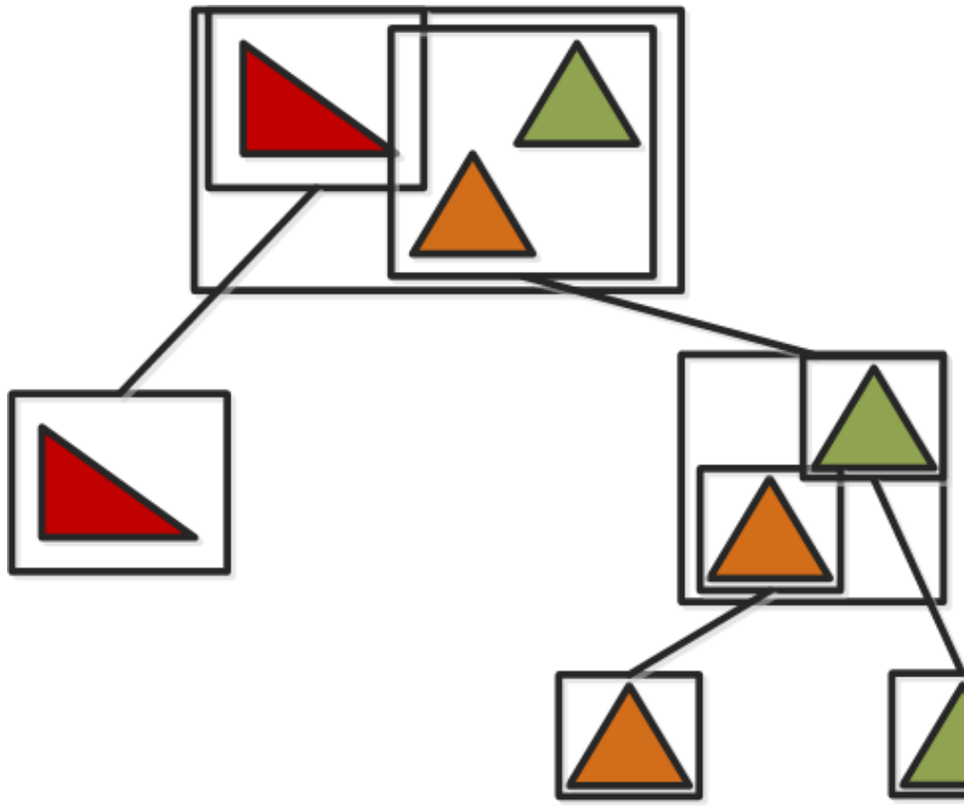


Image Credited to: Abhishek Chugh, Chenxi Liu / CPU-Based Parallel Ray Tracer

Linearly Arrange Triangles in Space

Property of z-order curve.
Traverse in space in z-order.
Near objects in z-order are near in space.

Encode triangles with Morton code.

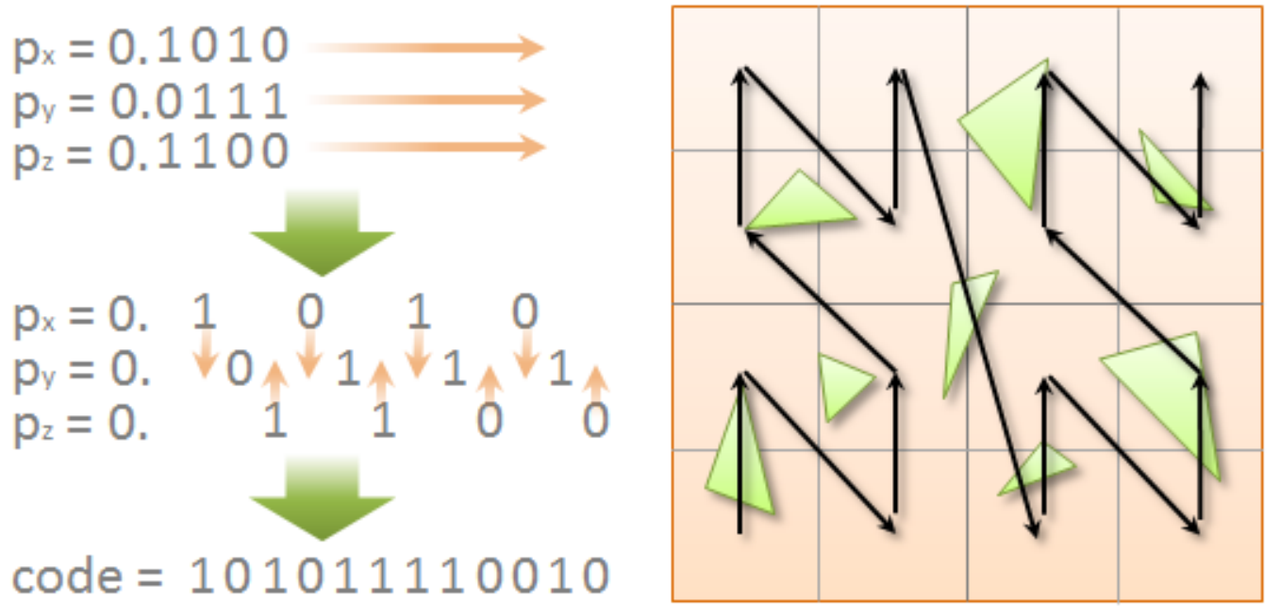
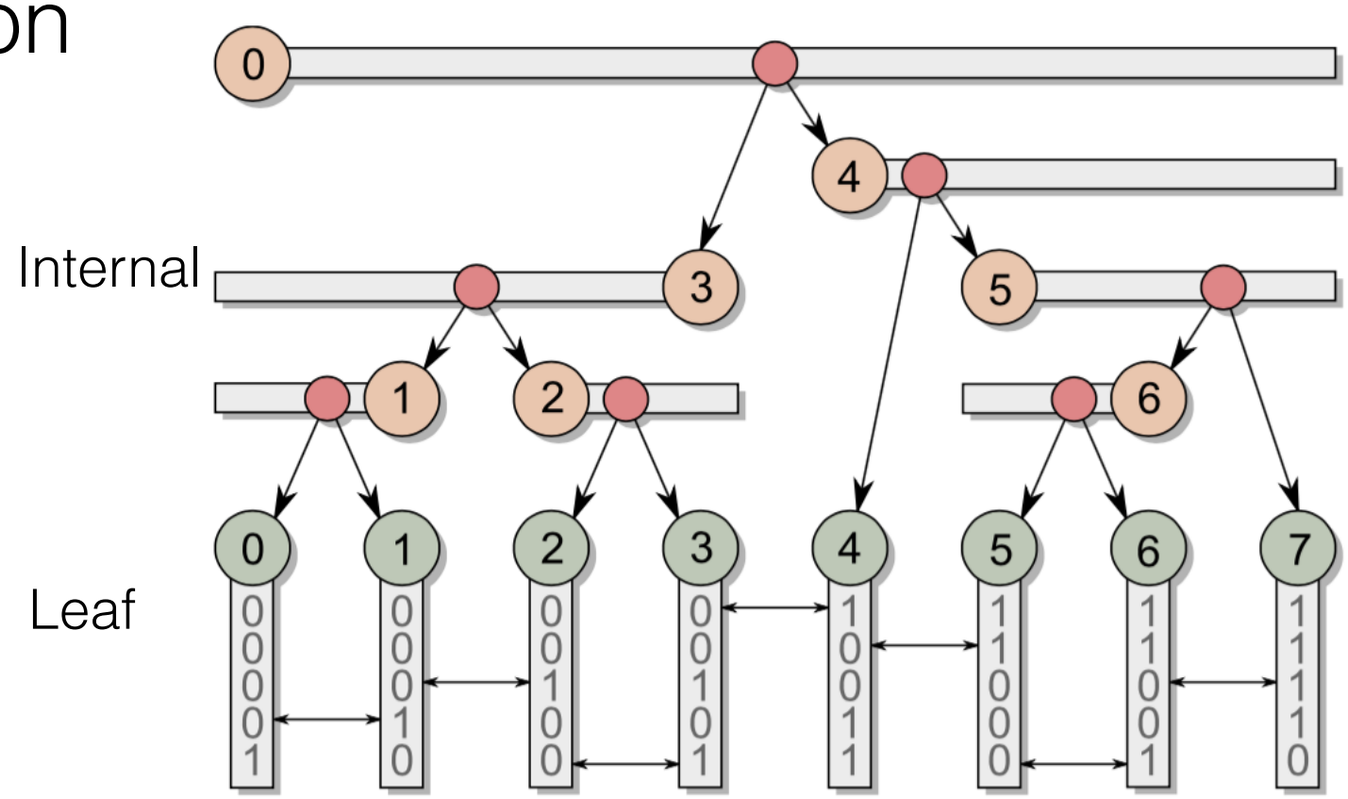


Image Credited to: Tero Karras / Thinking Parallel, Part III: Tree Construction on the GPU

Locate position of each node directly

Tree splits where the first difference in bit representation occurs.



A child node is either at the beginning or at the end of the range it belongs to.

Binary search to find the other end.

Parallel BVH Construction Result

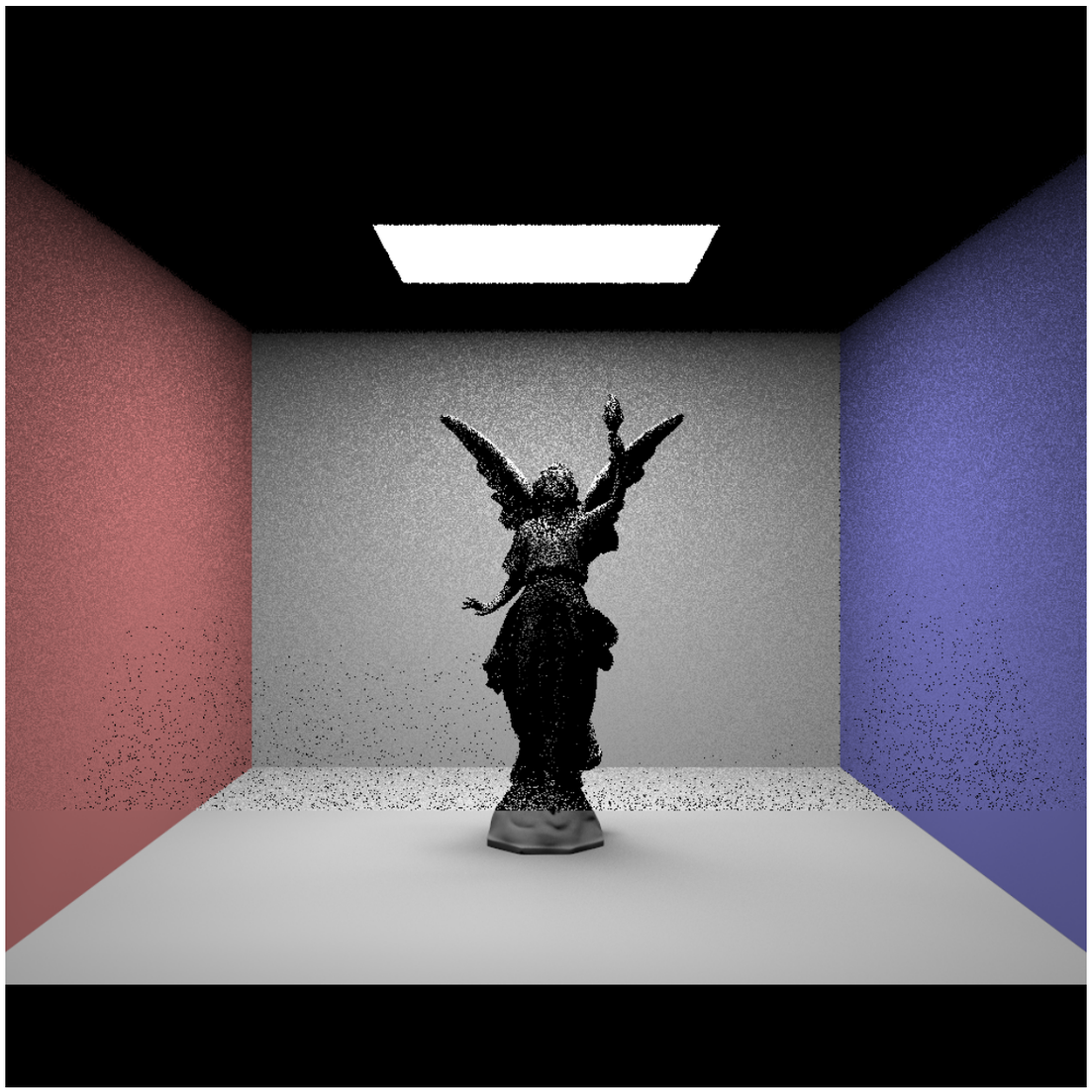
Scene	TriNum	GPU Build (s)	CPU Build (s)	Speedup
CBcoil	7884	0.001546	0.0415	27x
bunny	33696	0.002128	0.2235	105x
Dragon	105120	0.004377	0.8543	195x
CBlucy	133796	0.005089	1.1321	222x
Wall-e	240326	0.007785	1.9346	248x

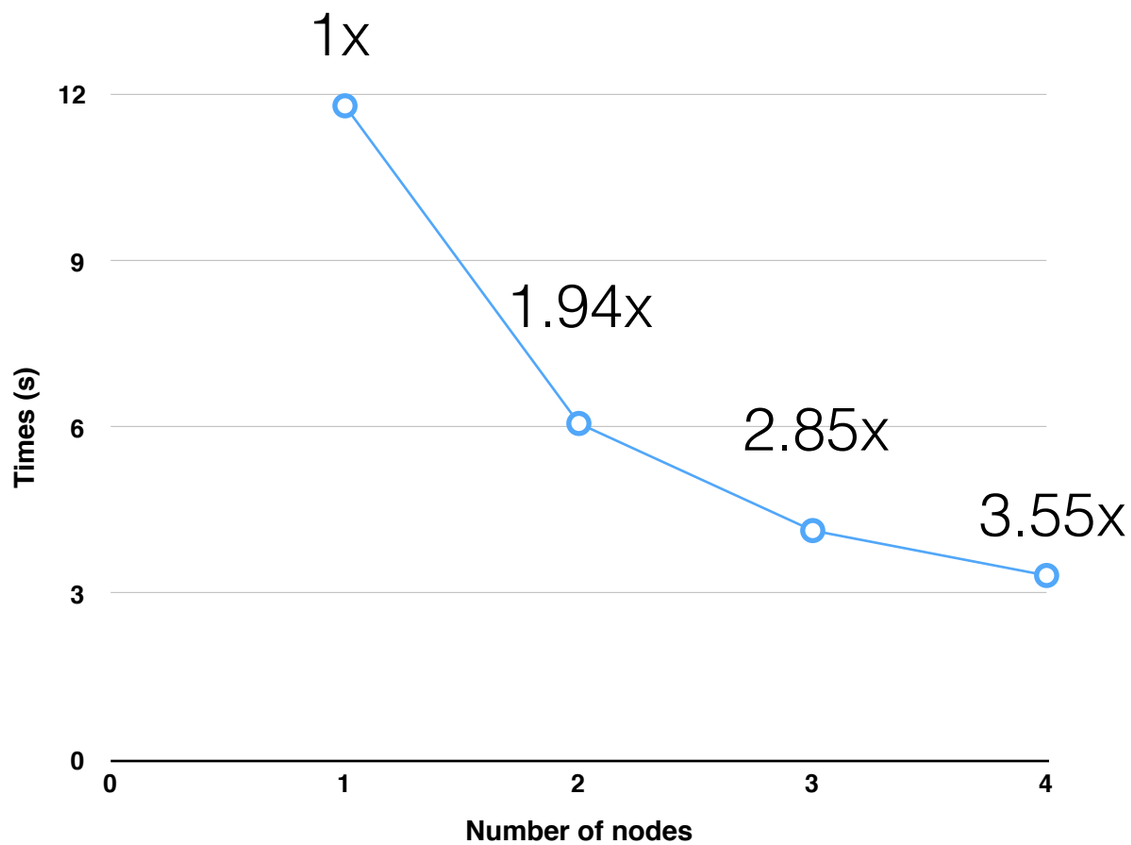
GPU: NVIDIA GTX 680
CPU: Intel Core i5 2.6 GHz (single thread)

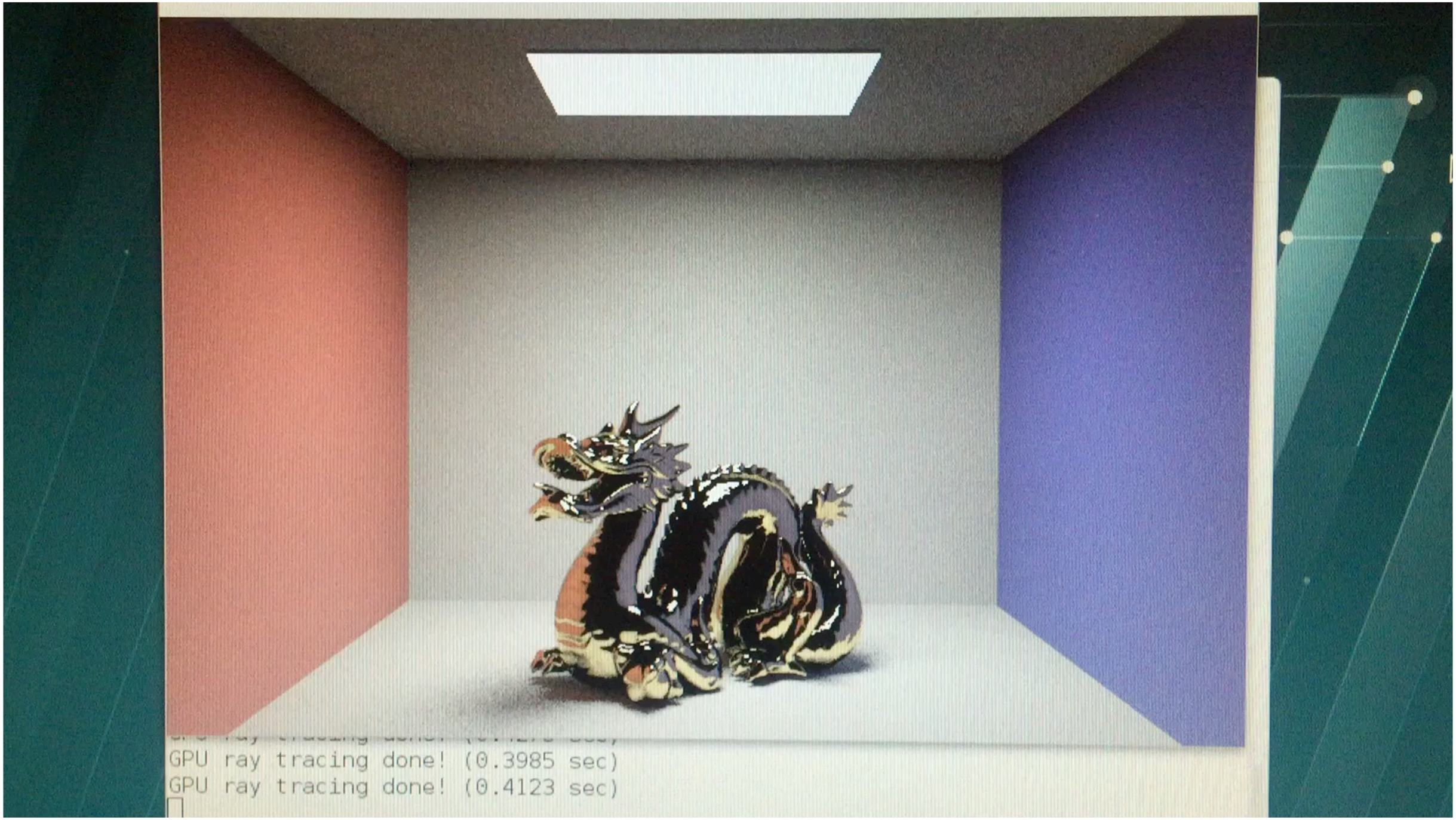
Distributed Ray Tracing

Ray Tracing is computation bound.

Distribute tiles of an image among worker nodes.







100K triangles 4 nodes 32 samples / pixel

It takes 20s to render 1 image on Intel i7 CPU with 8threads.

Questions?

Ye Yuan (yyuan2@andrew)
Ken Ling (kling1@andrew)