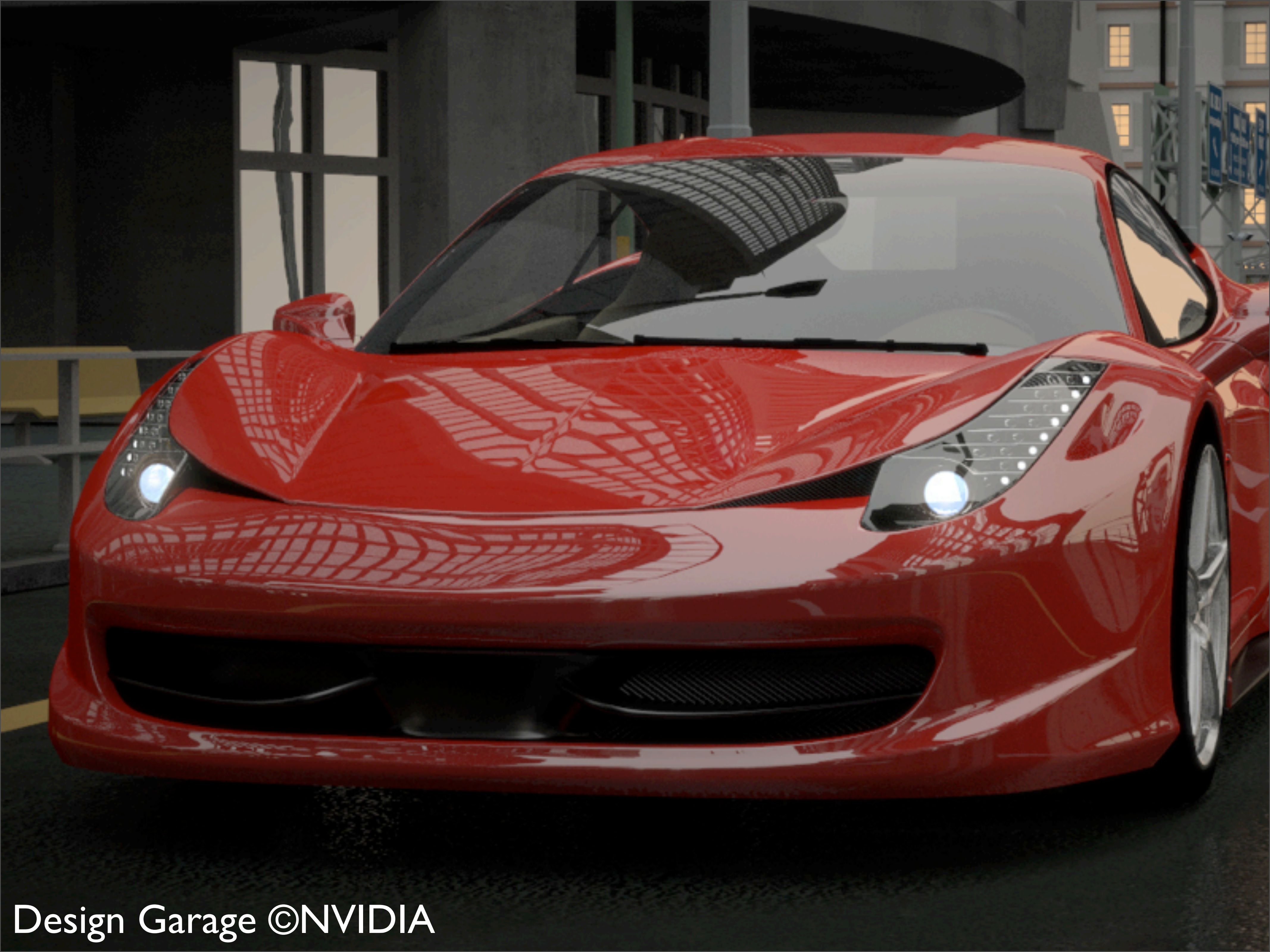
Parallel Progressive Photon Mapping on GPUs

Toshiya Hachisuka   Henrik Wann Jensen

University of California, San Diego

StarCraft II ©Blizzard Entertainment



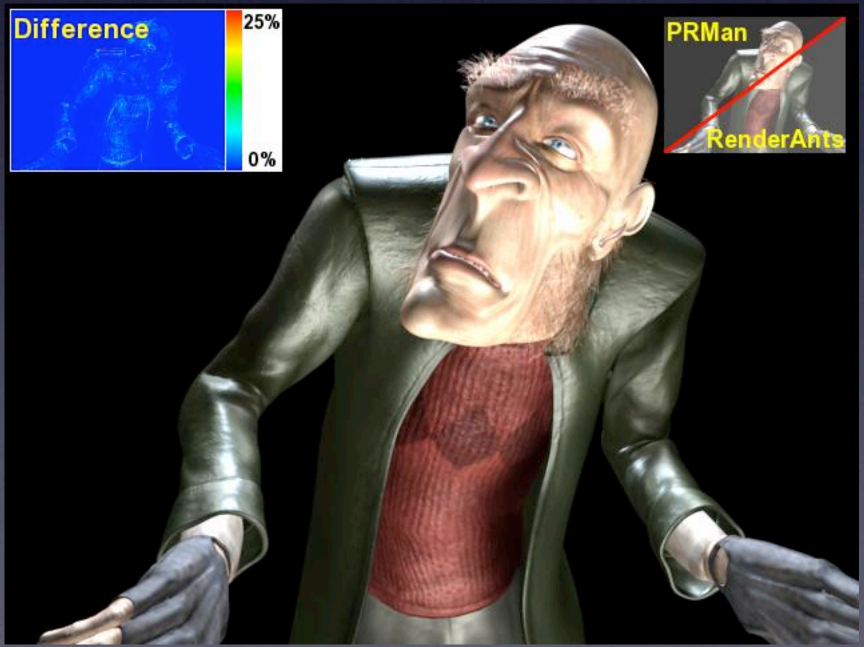
Design Garage ©NVIDIA

Offline Rendering on GPUs

- Growing interests across communities
  - Rendering is highly parallel computation
  - GPU is a massively parallel processor

```
V-Ray RT 2.00.01
2389.099 KPaths/s
2m 23.6s elapsed
Engine: OpenCL (simple)
```

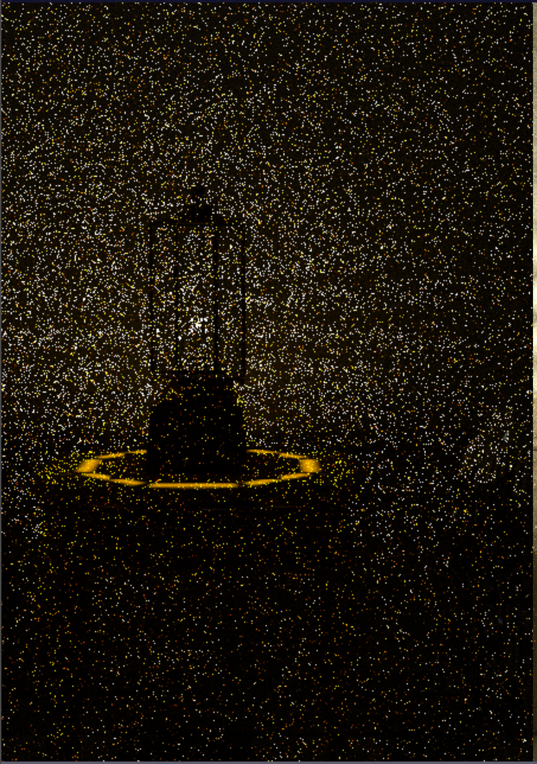
V-Ray RT GPU ©Chaos Group



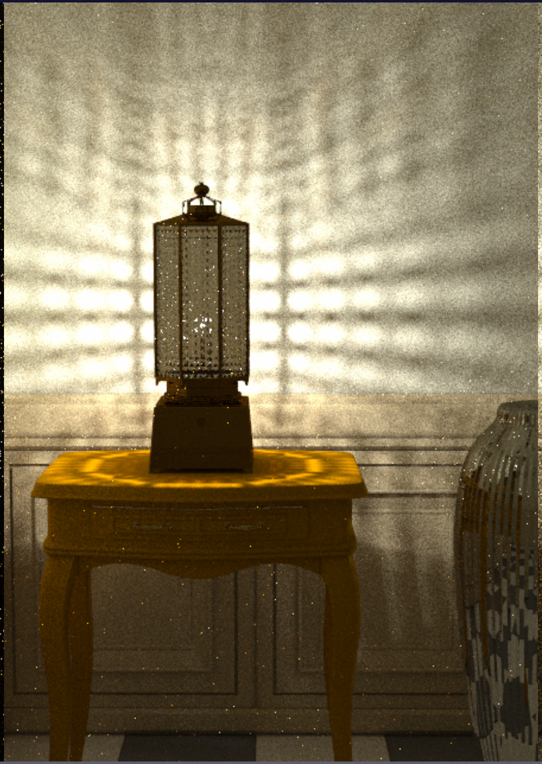
RenderAnts [Zhou 09]

Progressive Photon Mapping (PPM)

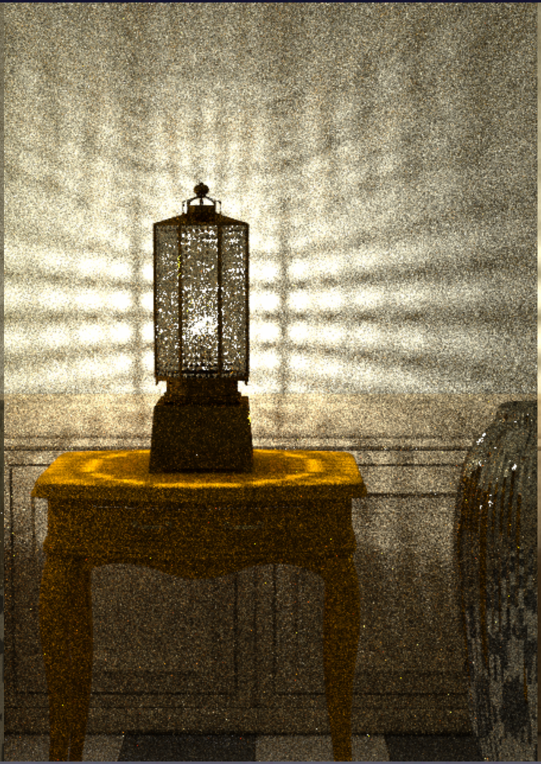
- New rendering algorithm [Hachisuka 08, 09]
- Only method that handles specular-diffuse-specular



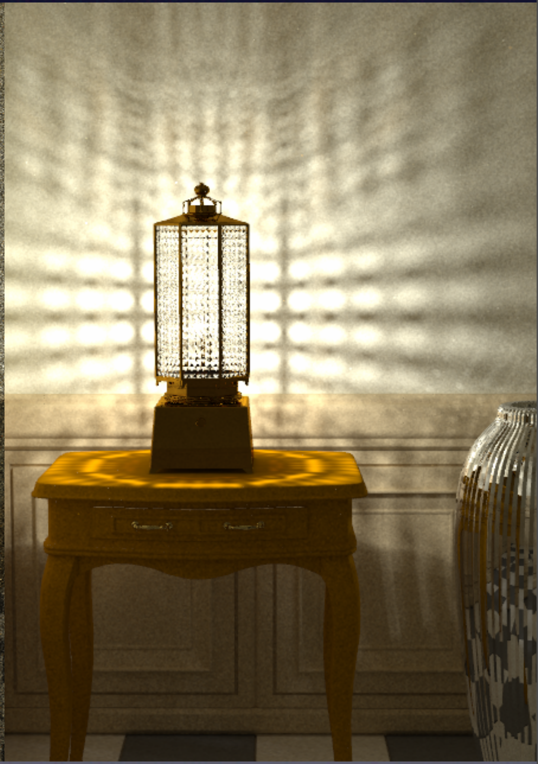
Path Tracing



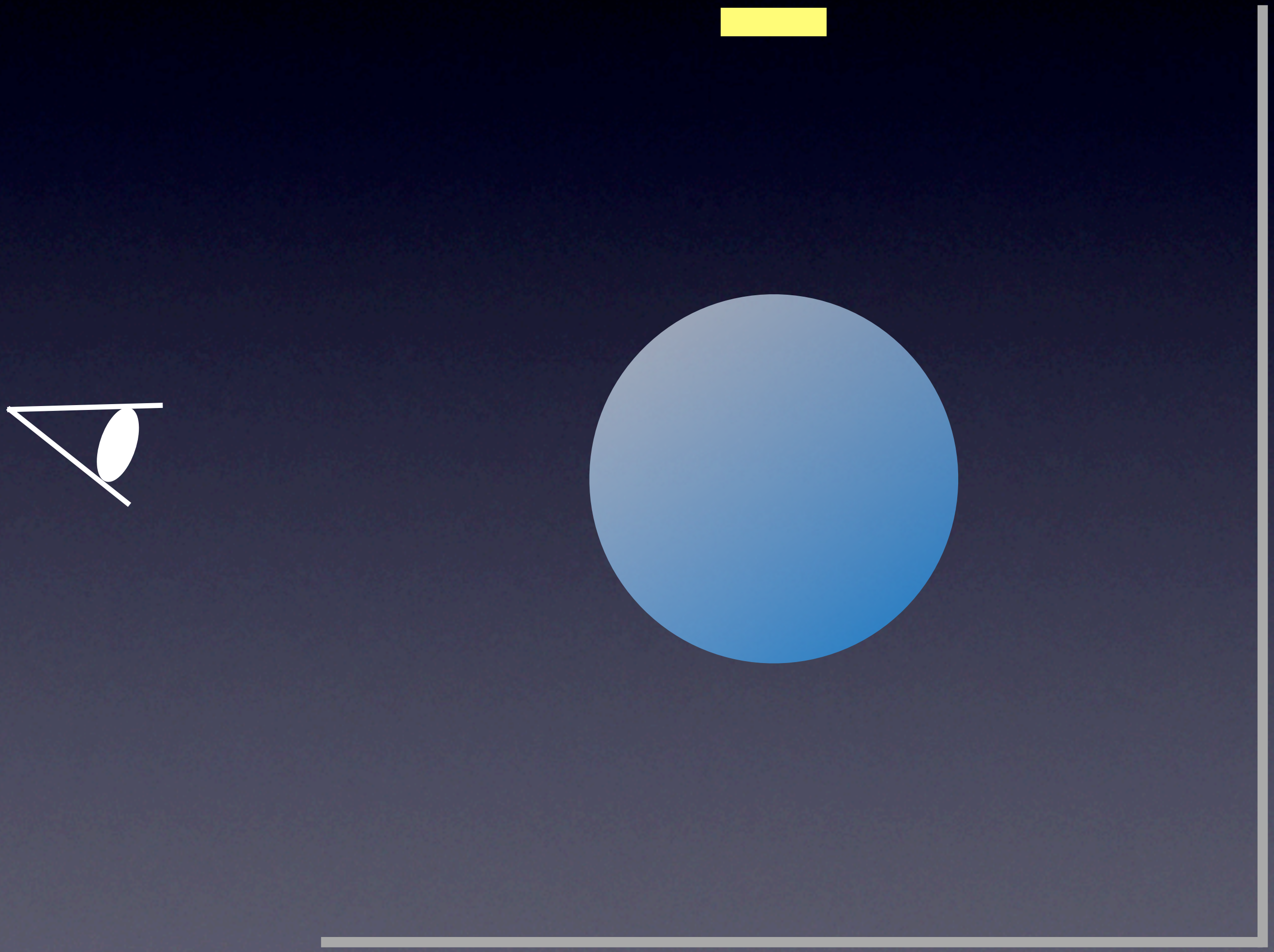
Bidirectional Path Tracing



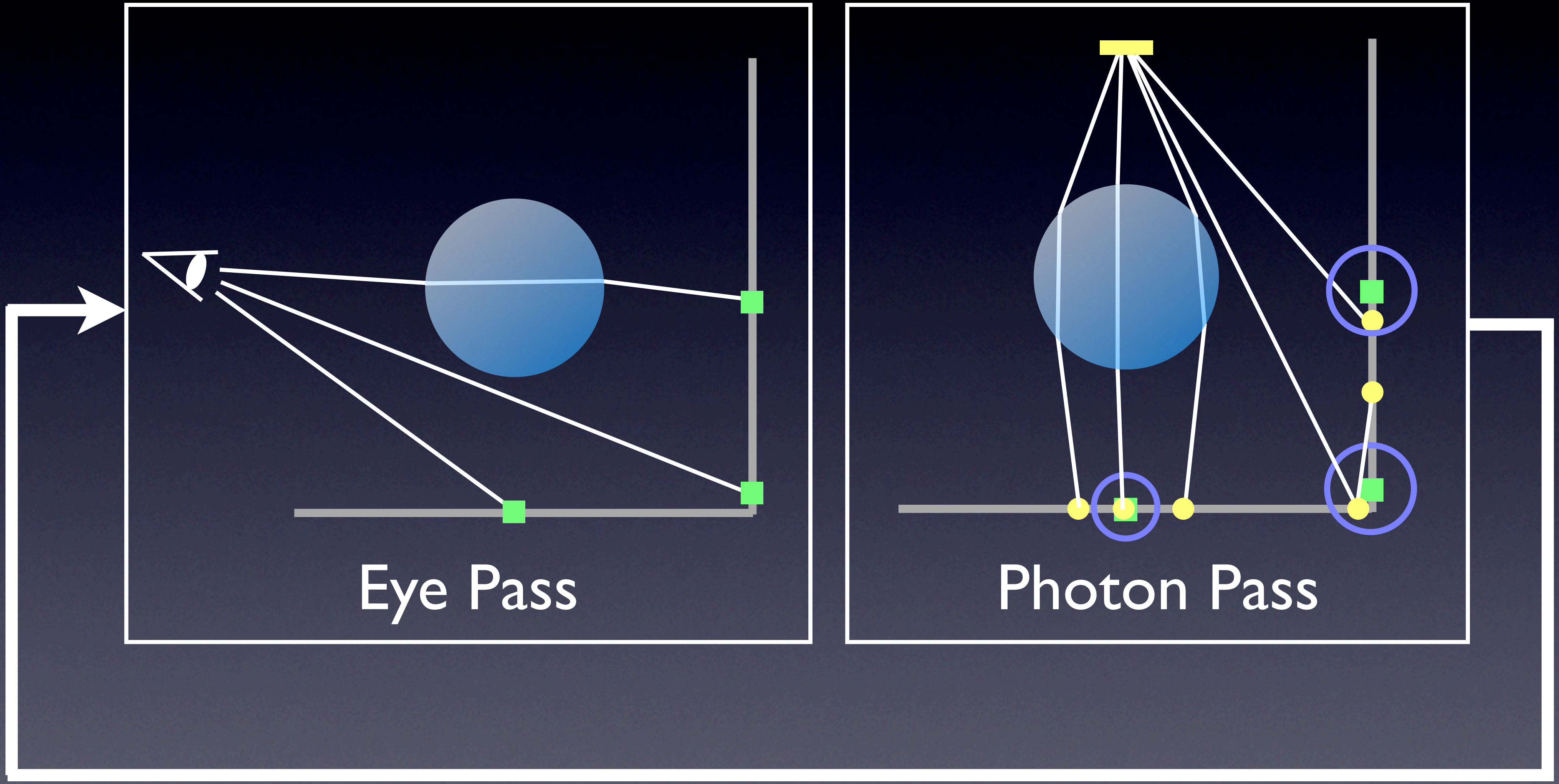
Metropolis Light Transport



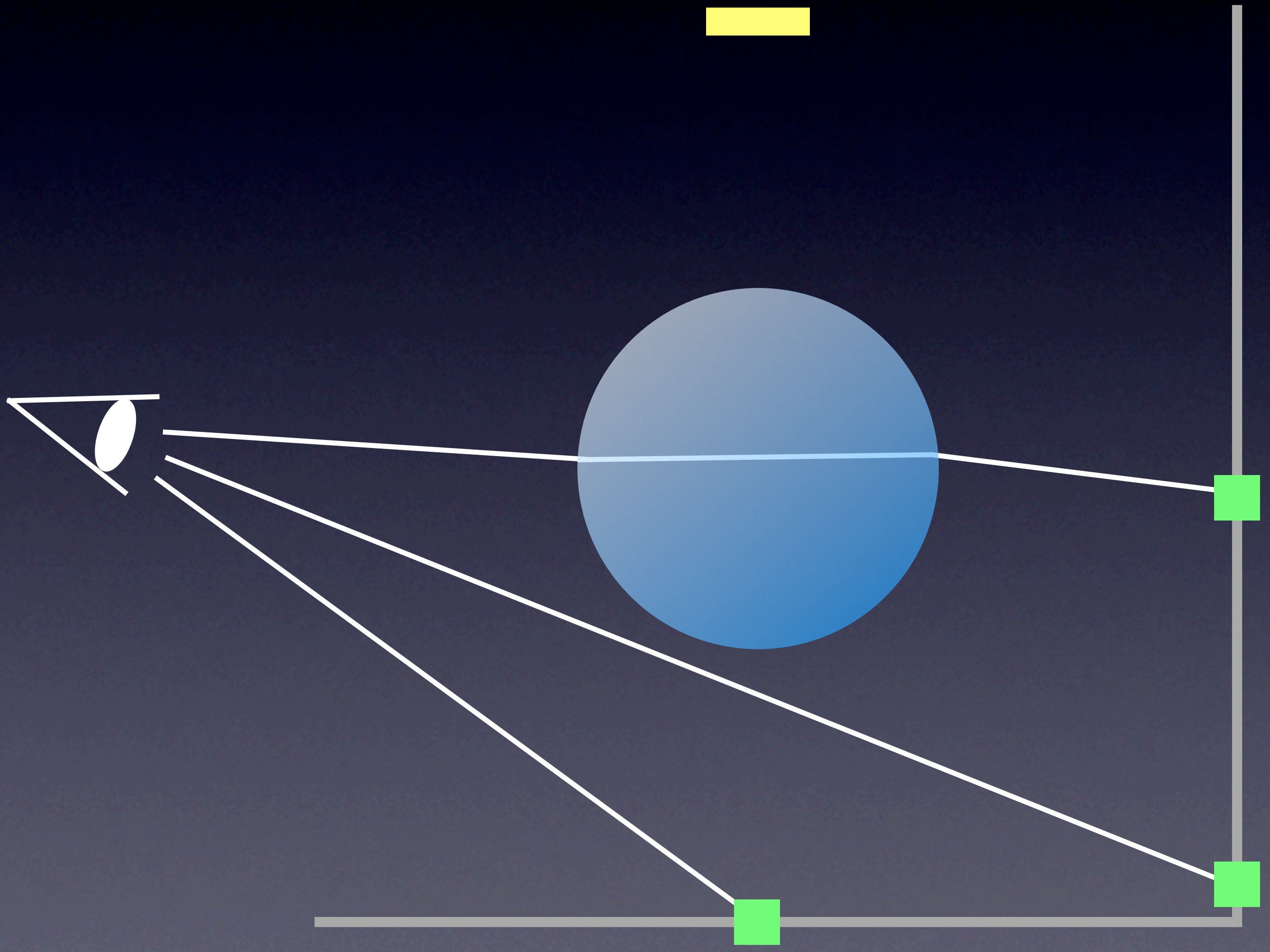
Progressive Photon Mapping



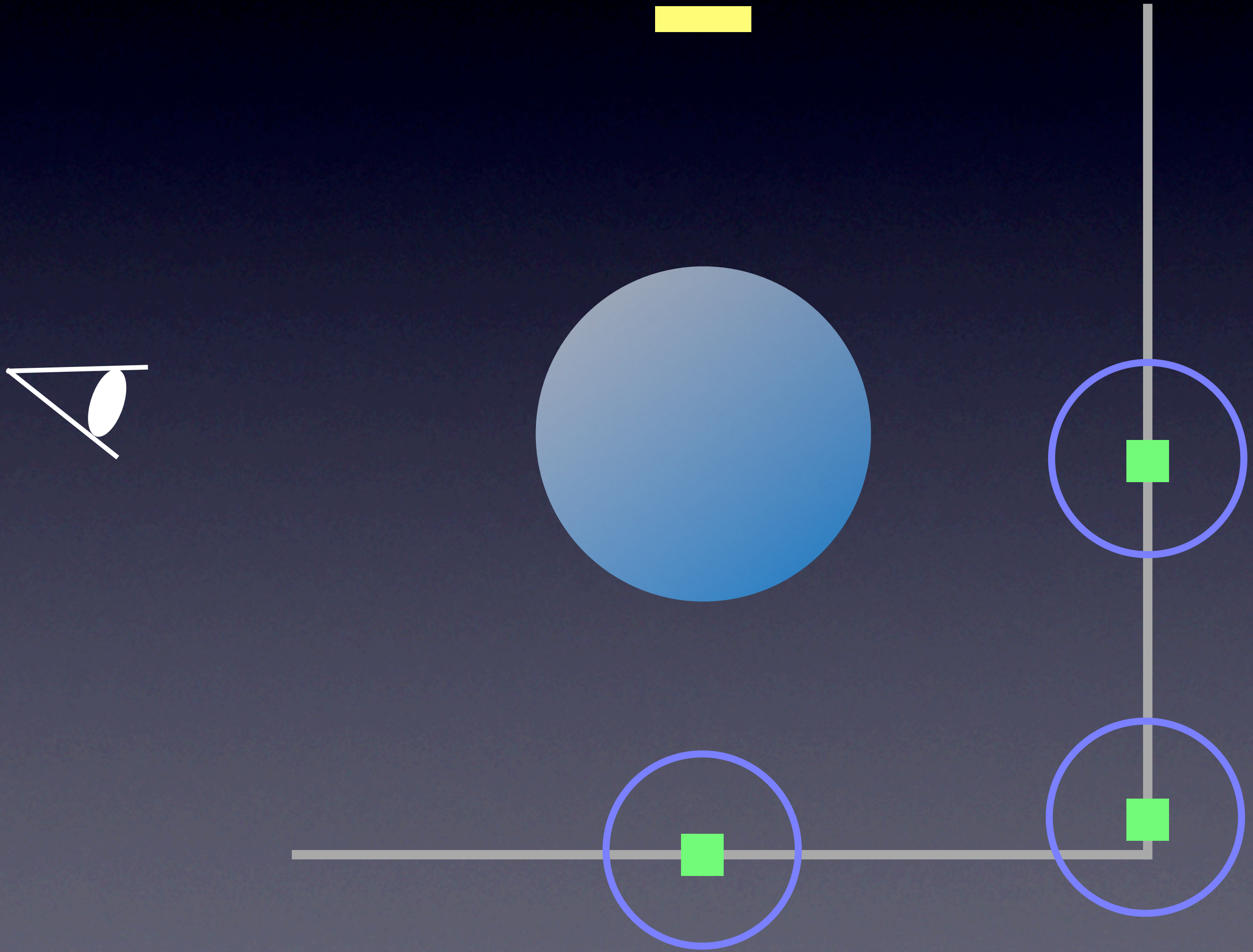
Overview



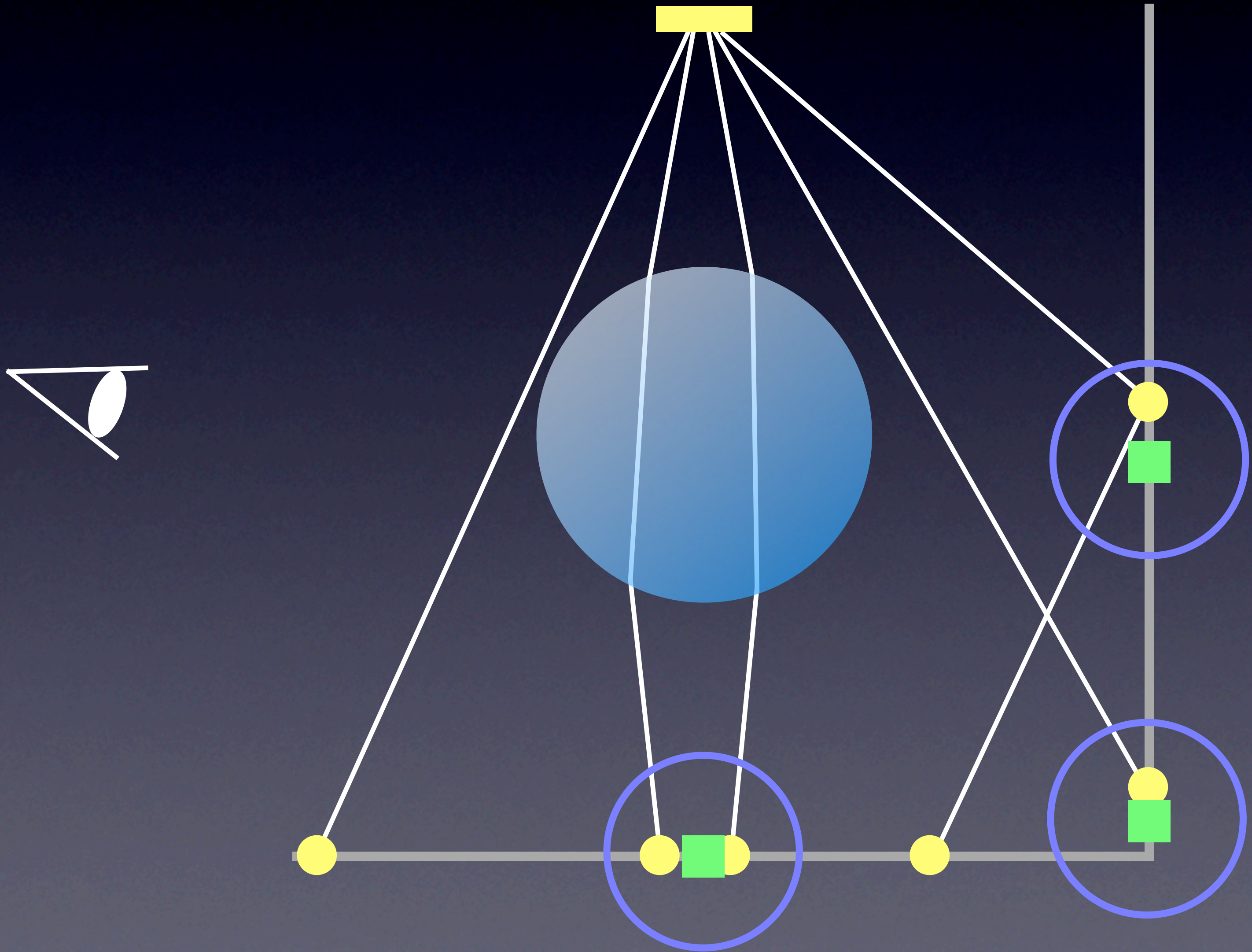
Eye Pass



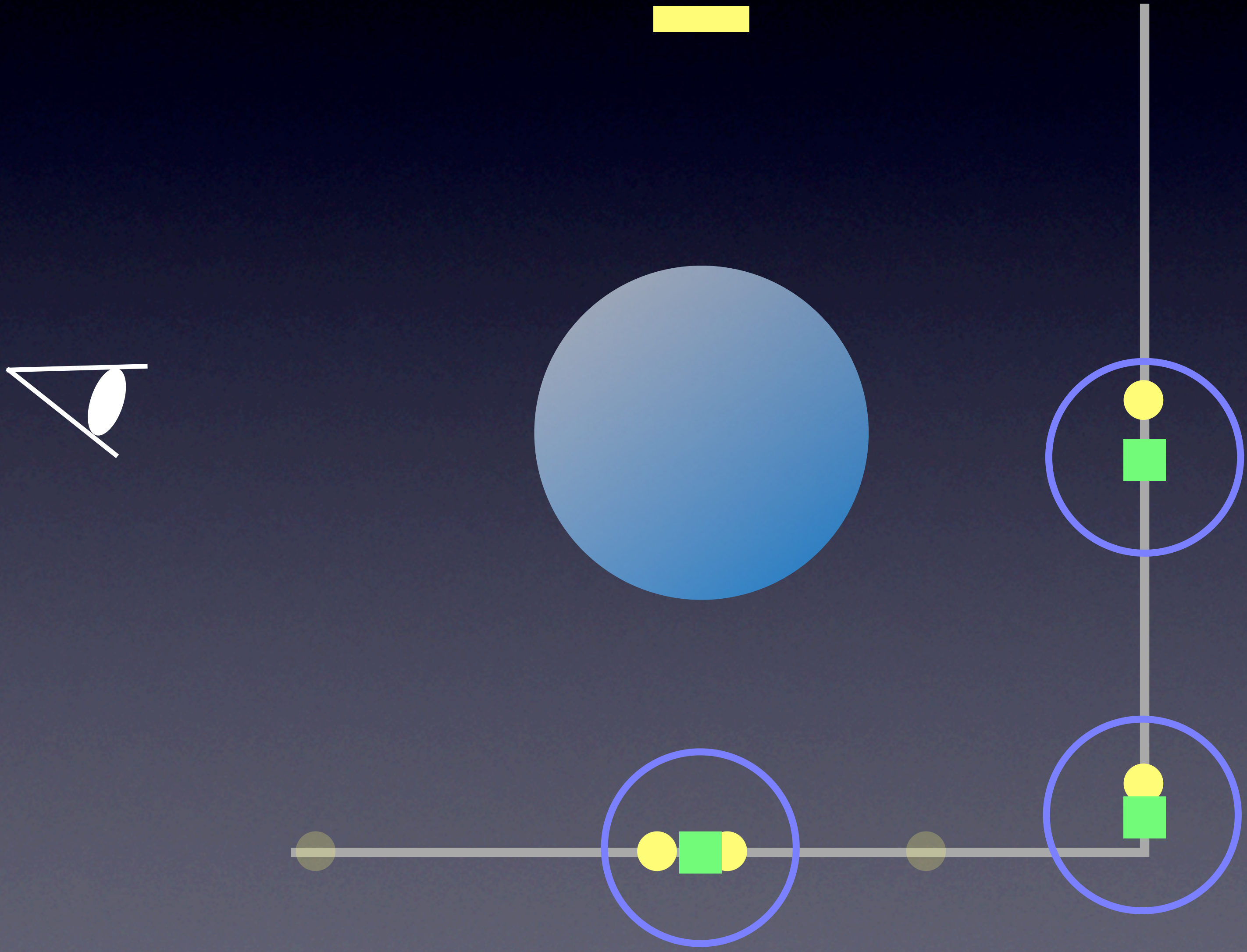
Eye Pass



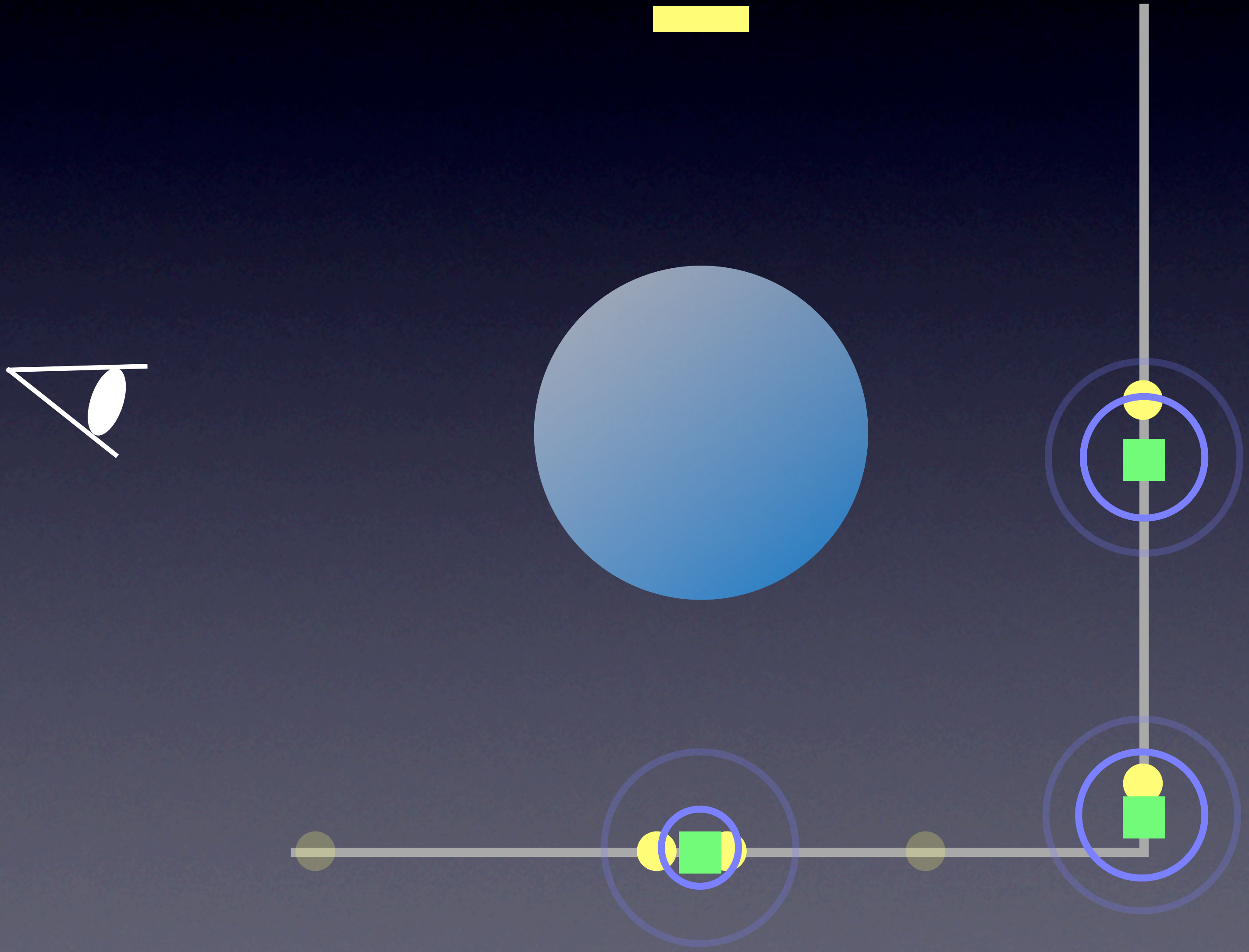
Photon Pass



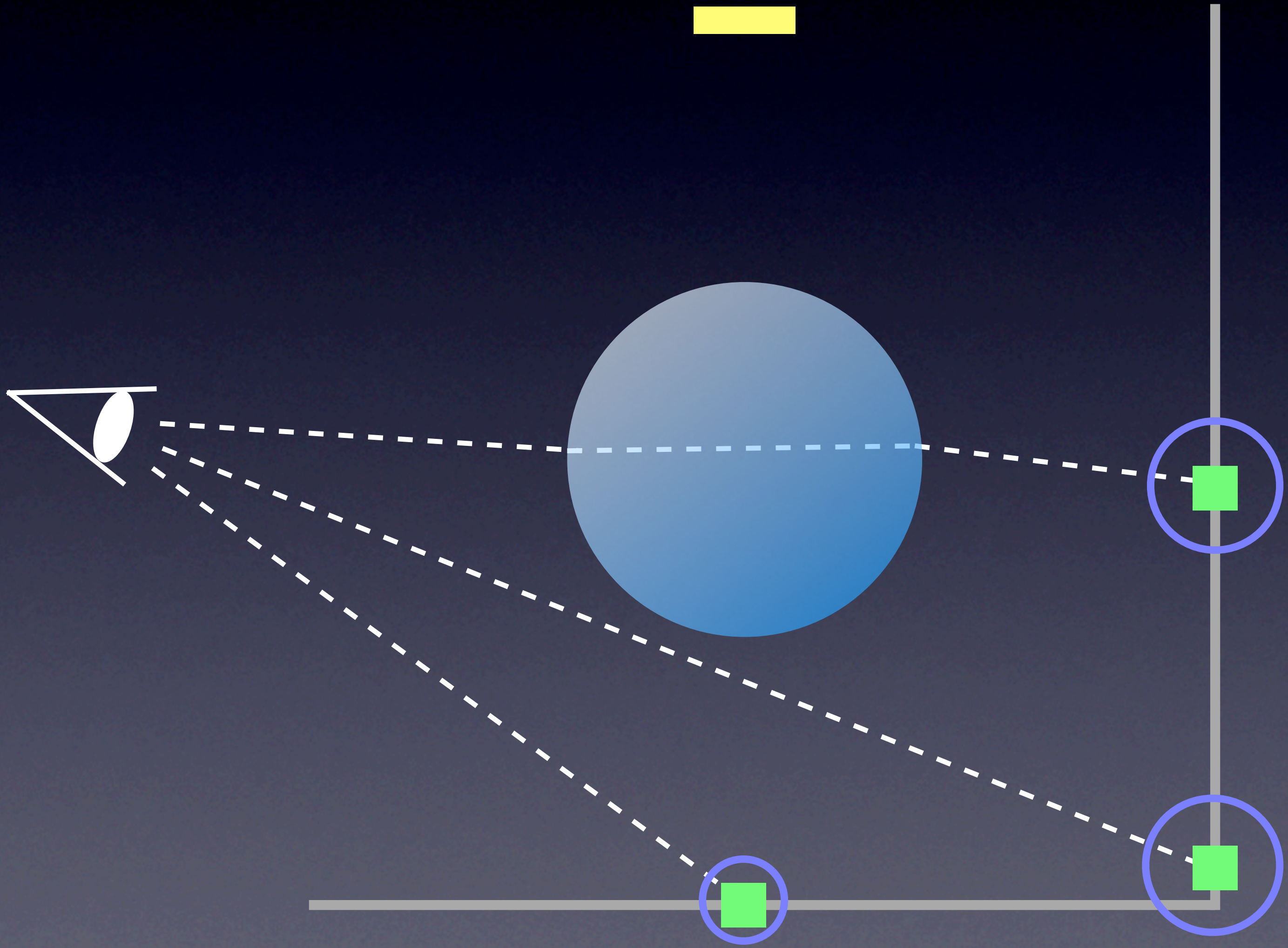
Photon Pass



Photon Pass



Rendering



Parallelism in PPM

- Many parts are highly parallel
  - Eye ray tracing
  - Photon tracing
  - Rendering

...but not everything

- Collecting photons

Contribution

Entirely parallel progressive photon mapping algorithm

Method

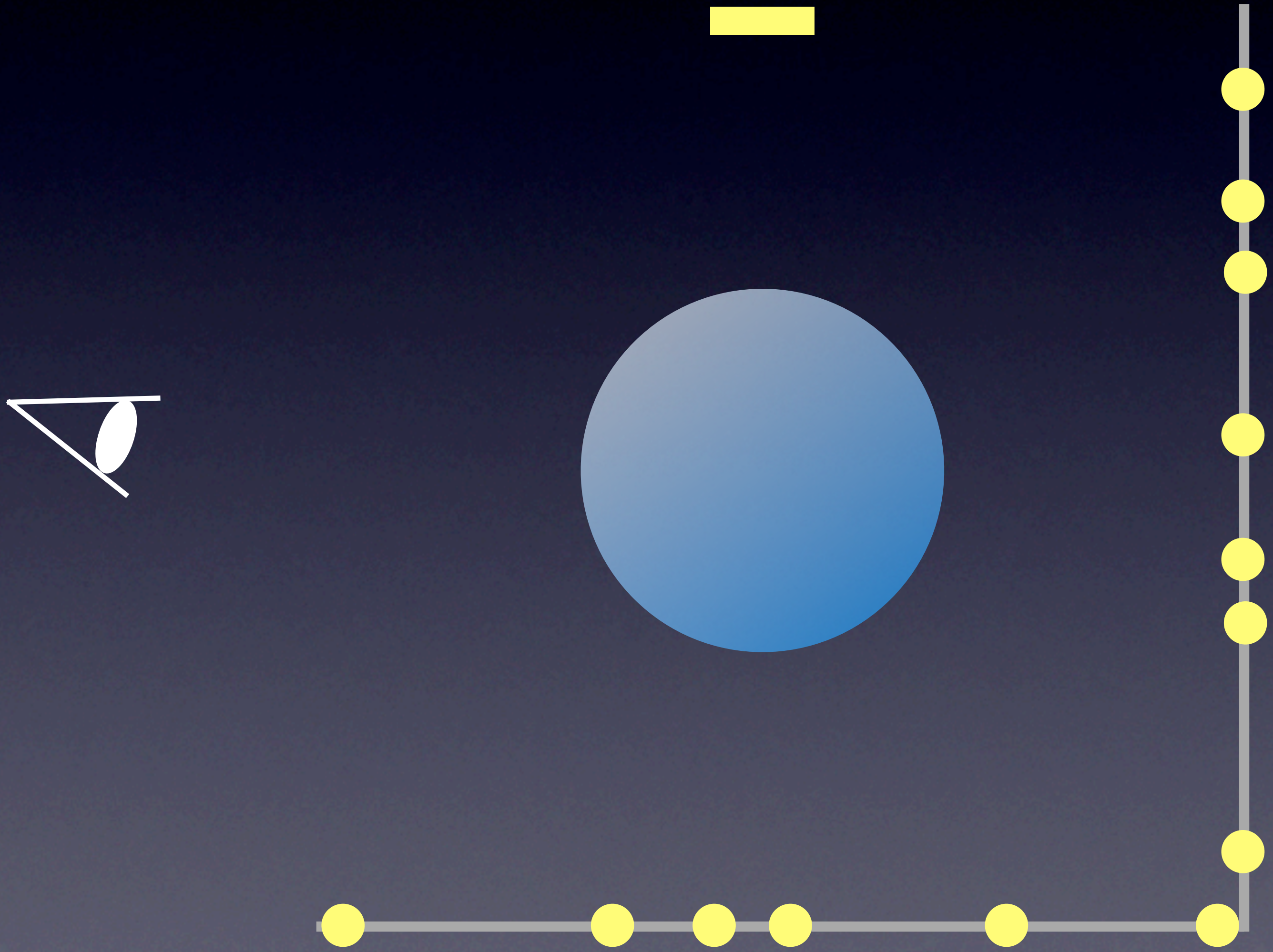
Parallelism in PPM

- Many parts are highly parallel
  - Eye ray tracing
  - Photon tracing
  - Rendering

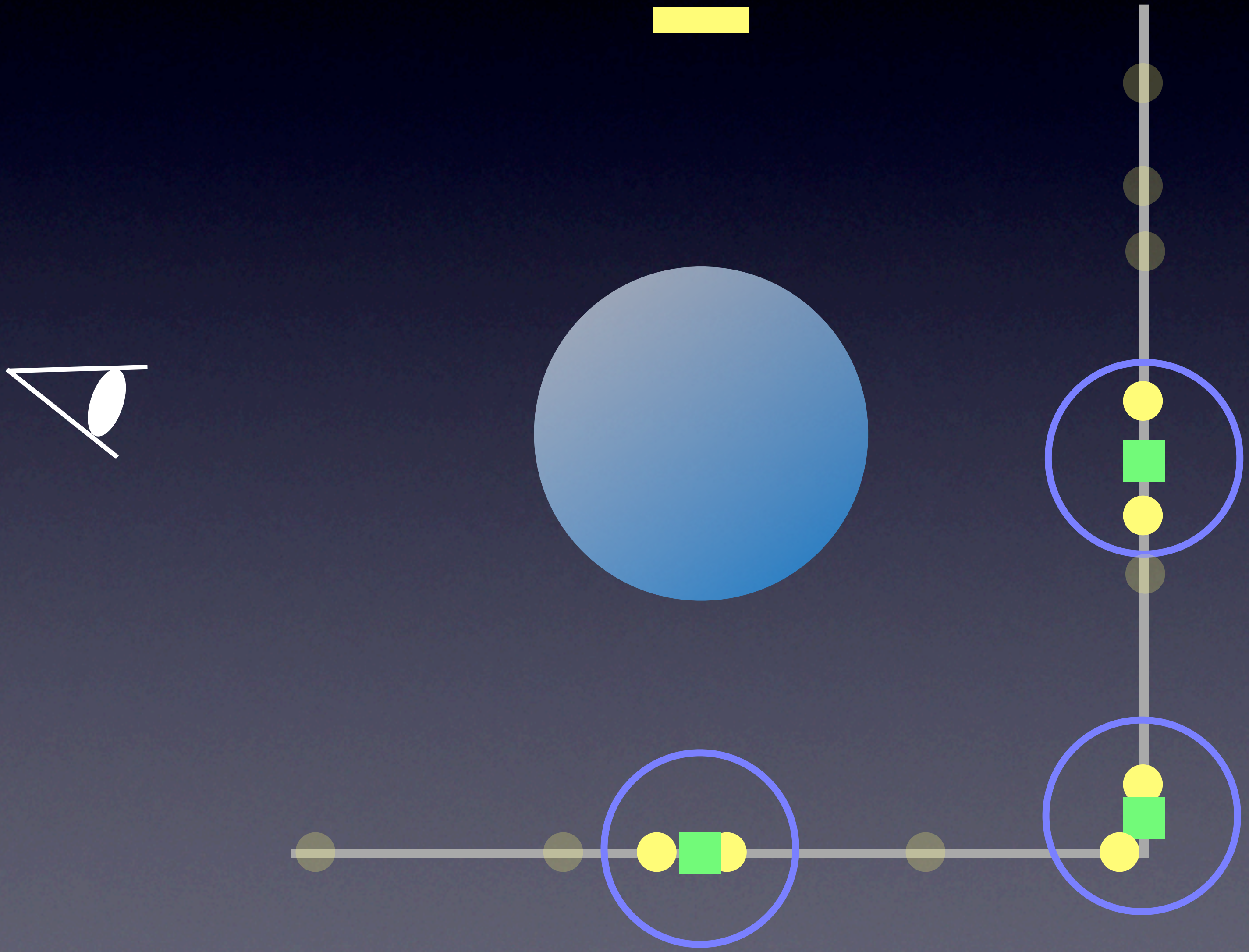
...but not everything

- Collecting photons

Problem



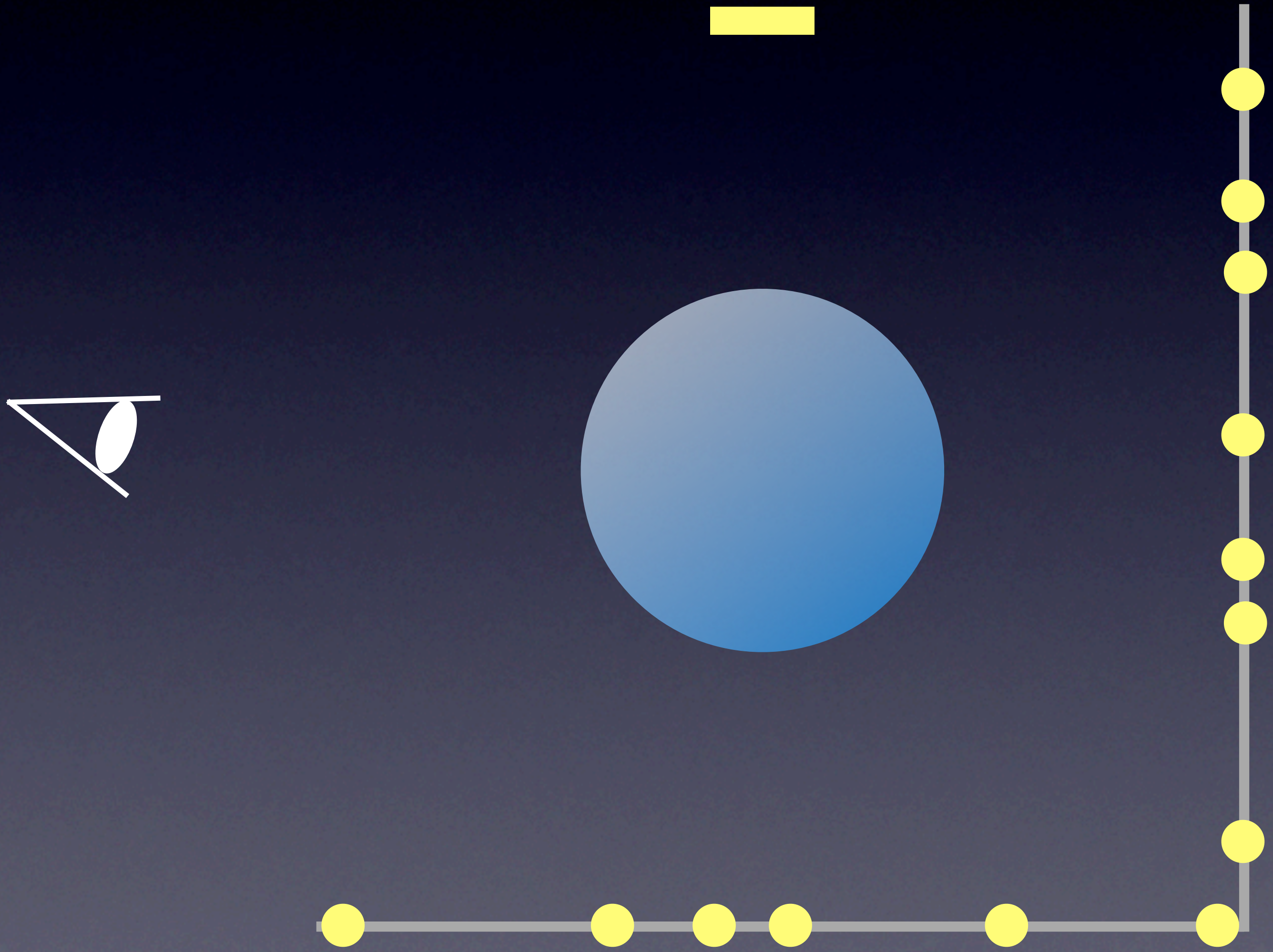
Problem



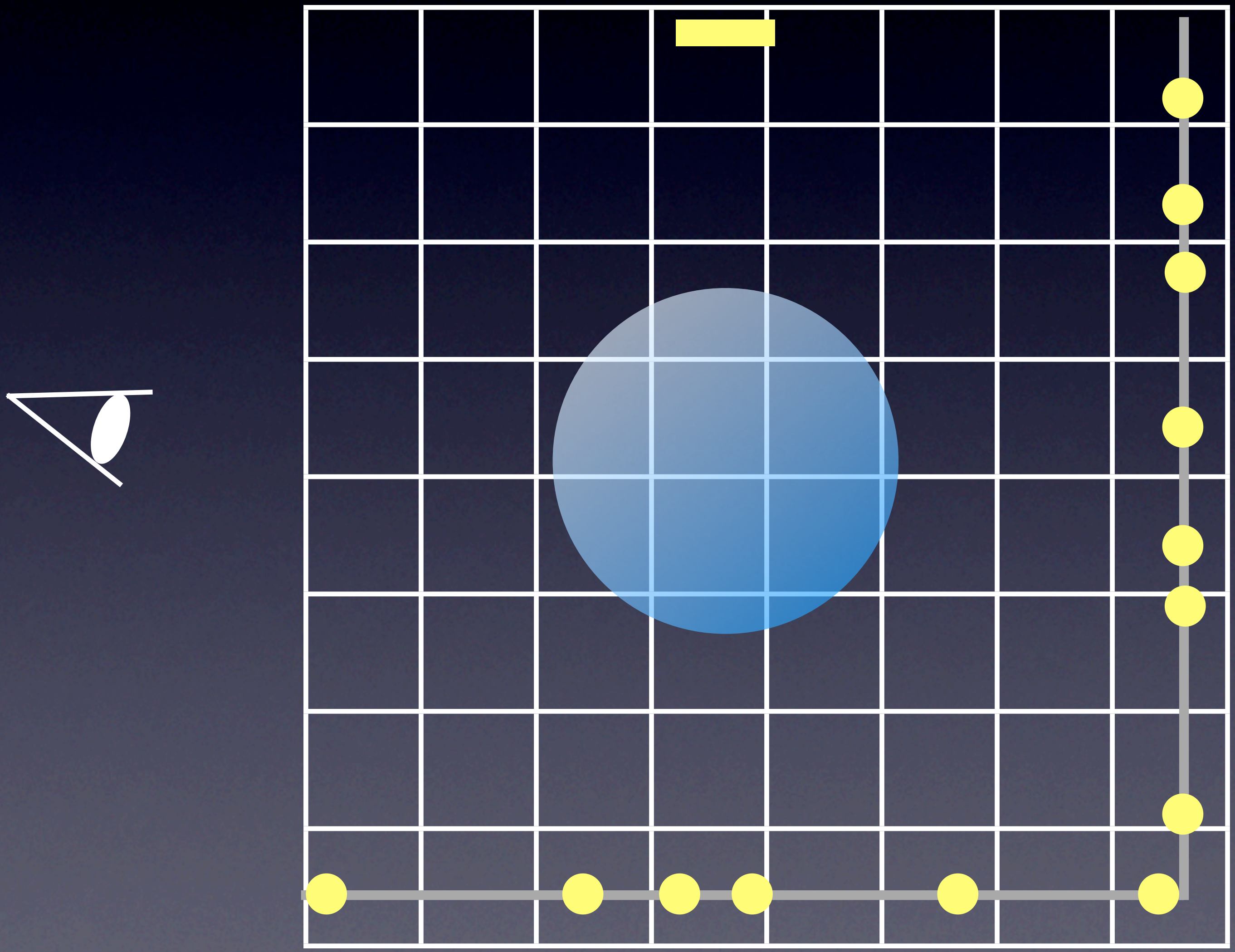
Spatial Hashing

- Construction
  - Discretize space into cells
  - Construct a hash table with lists
- Query
  - Look up overlapping cells
  - Traverse lists

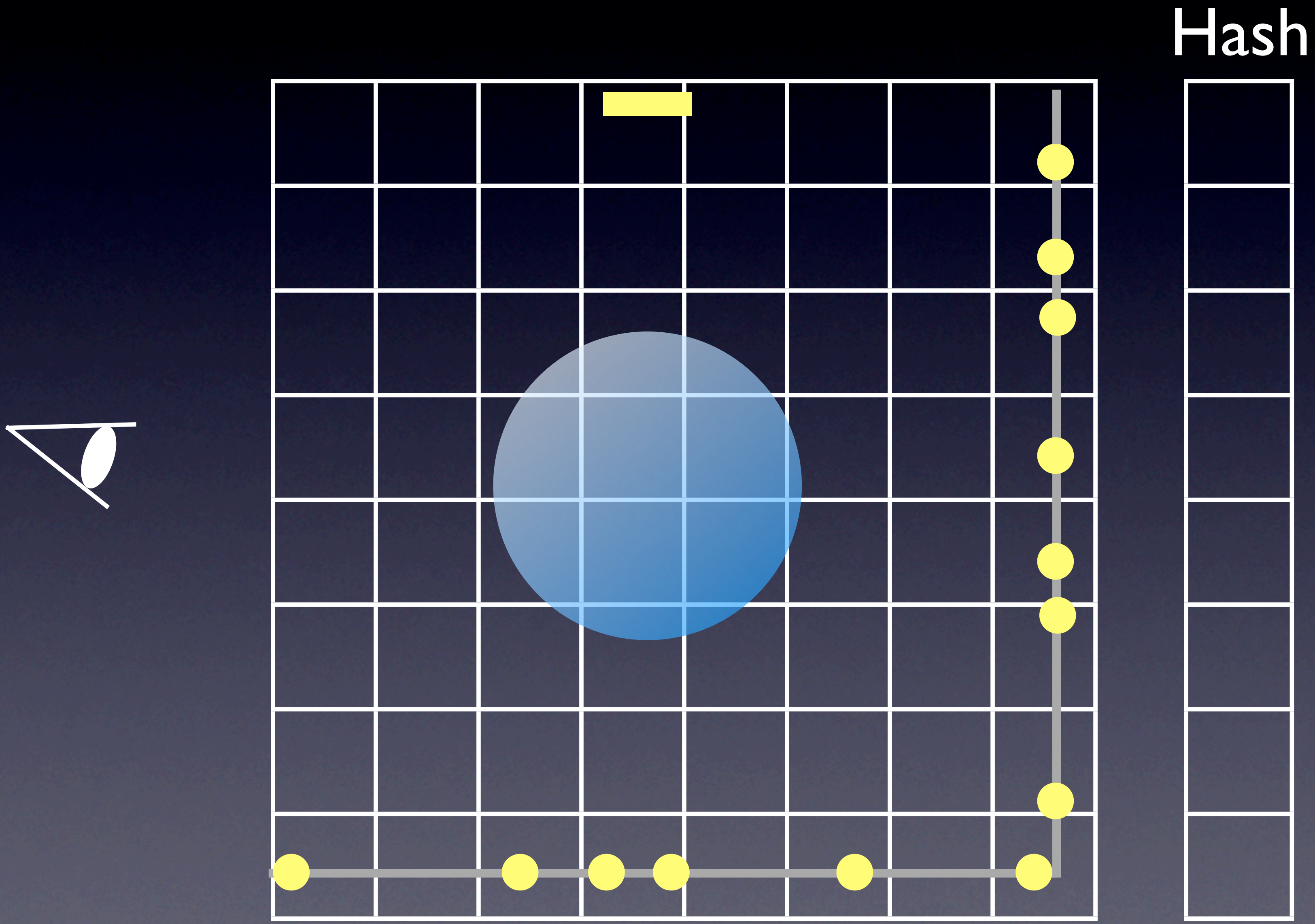
Spatial Hashing: Construction



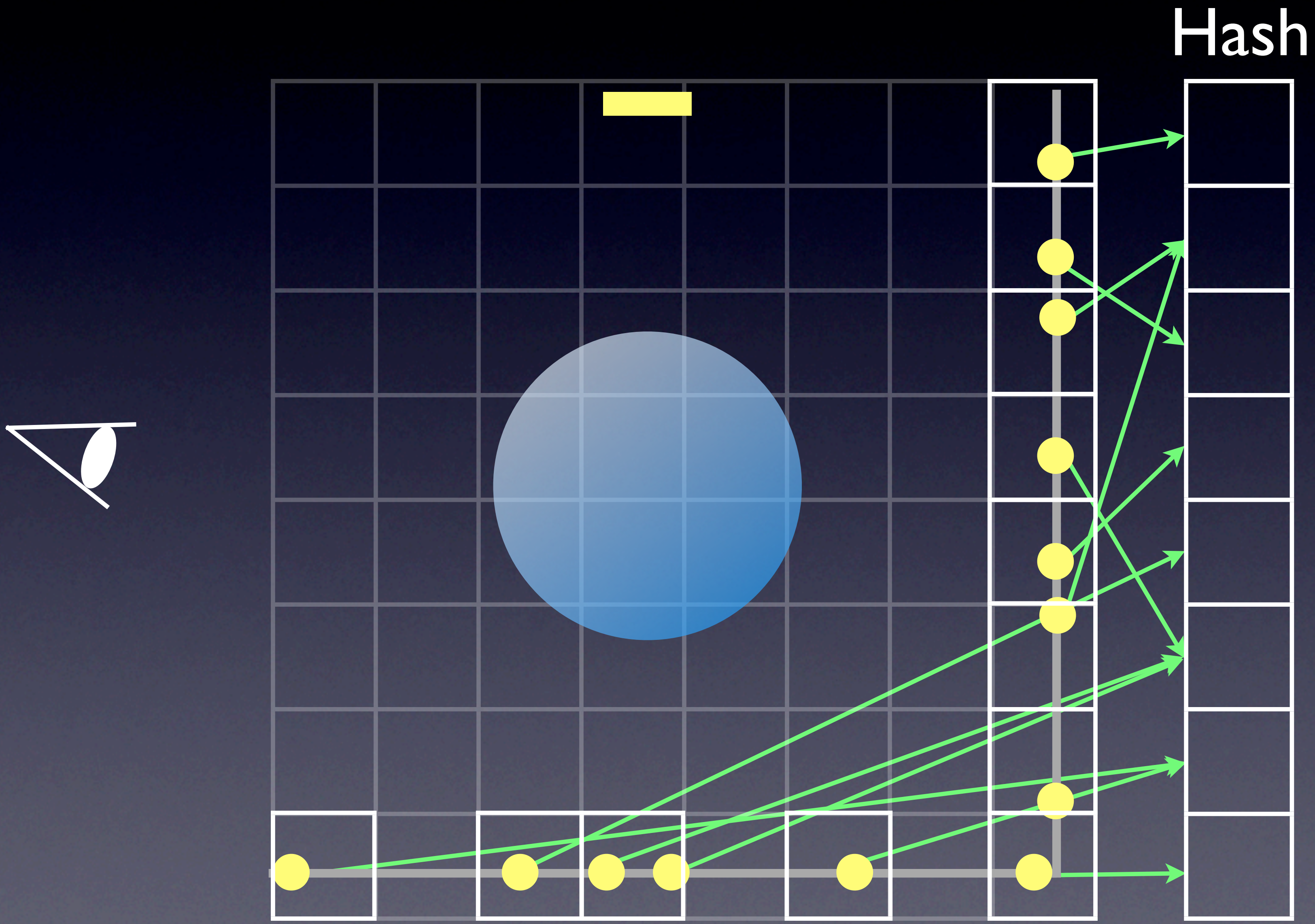
Spatial Hashing: Construction



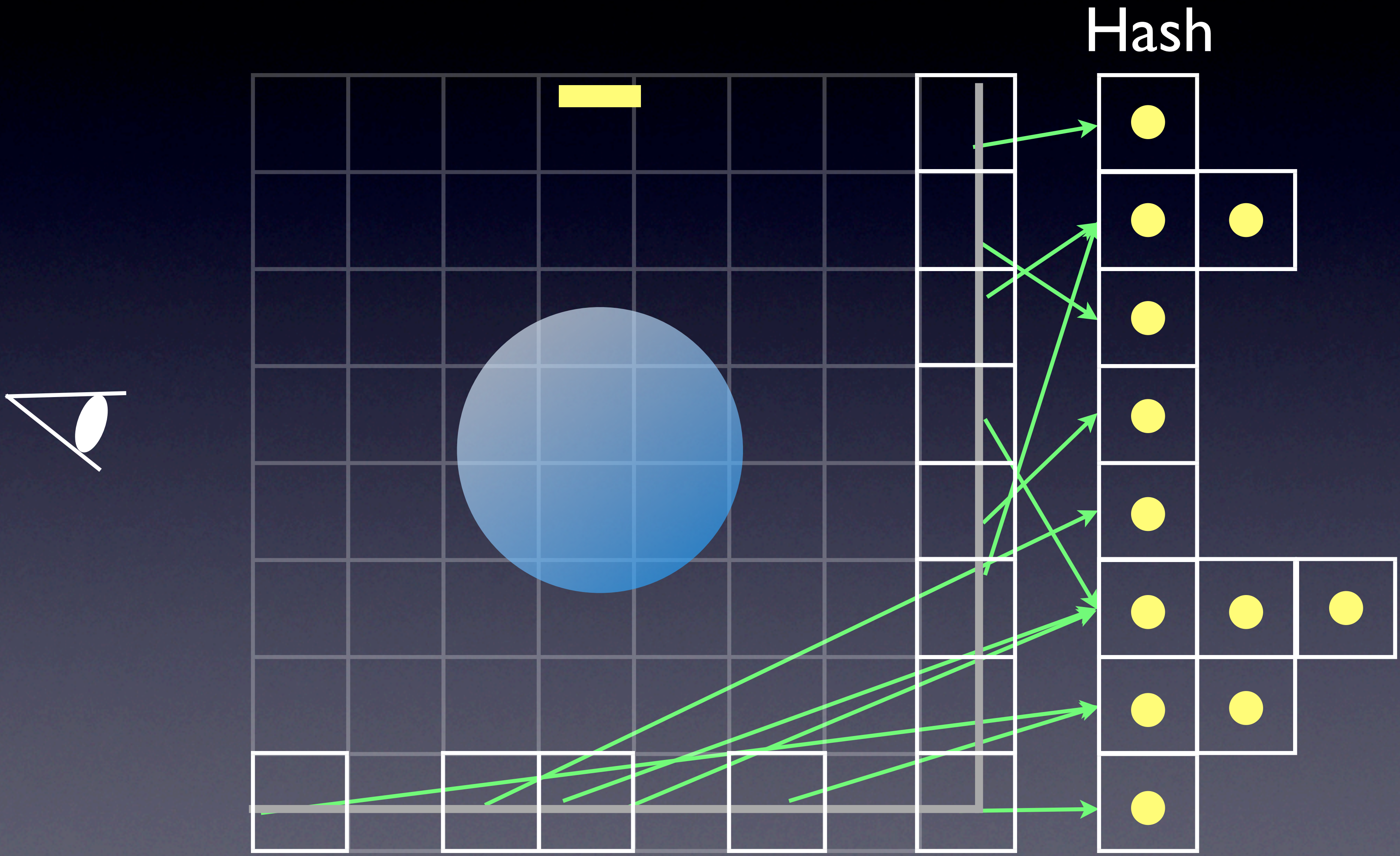
Spatial Hashing: Construction



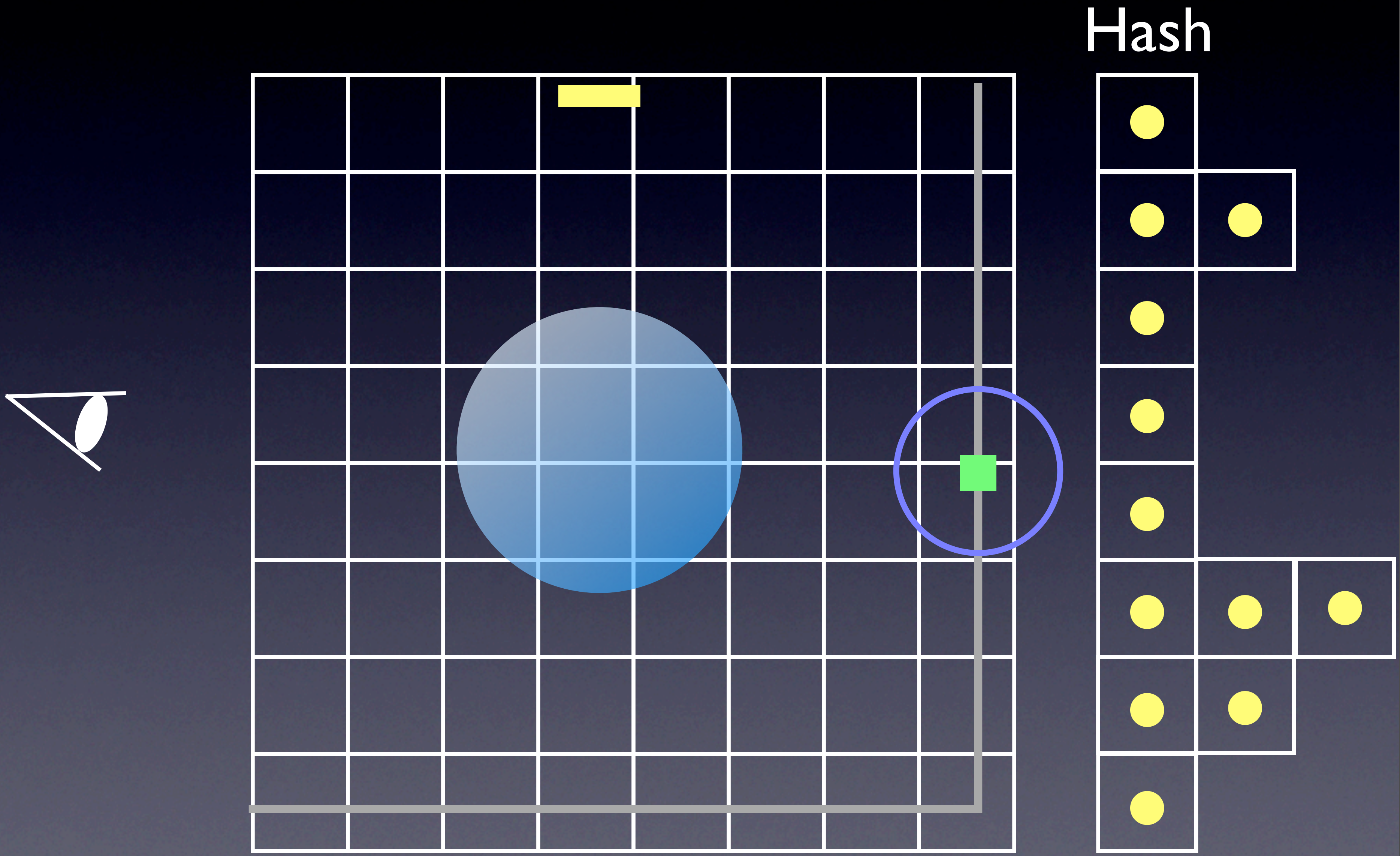
Spatial Hashing: Construction



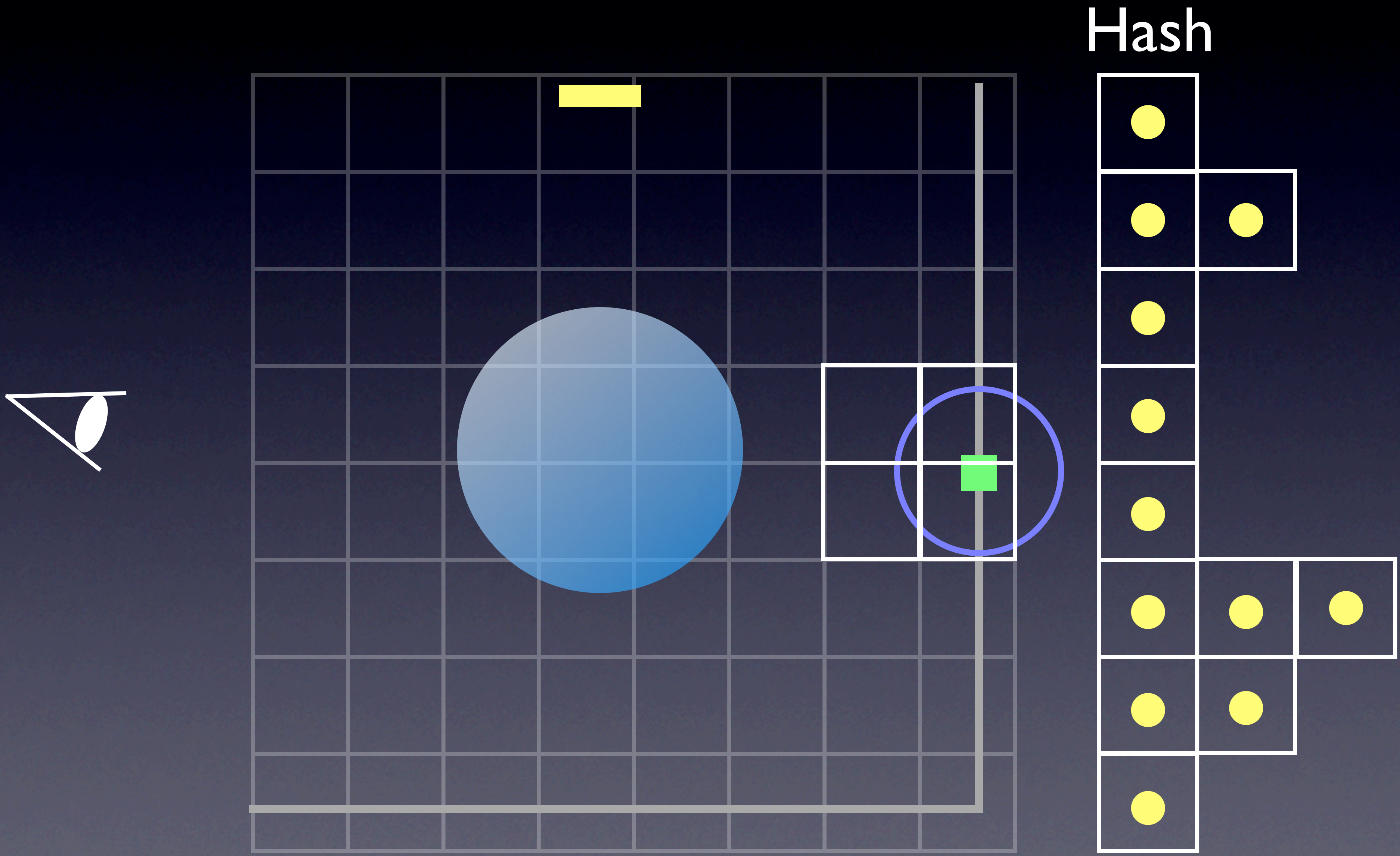
Spatial Hashing: Construction



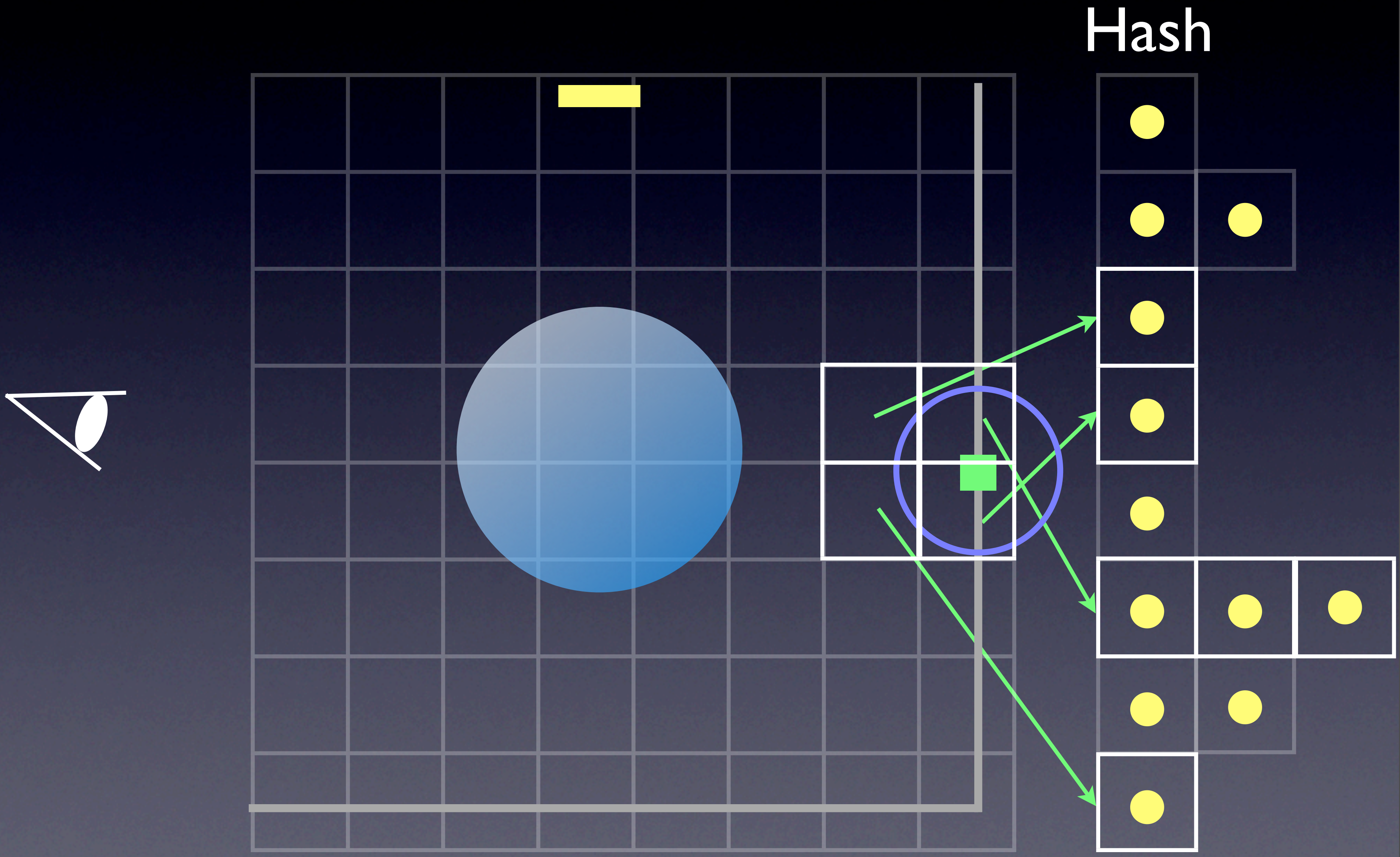
Spatial Hashing: Query



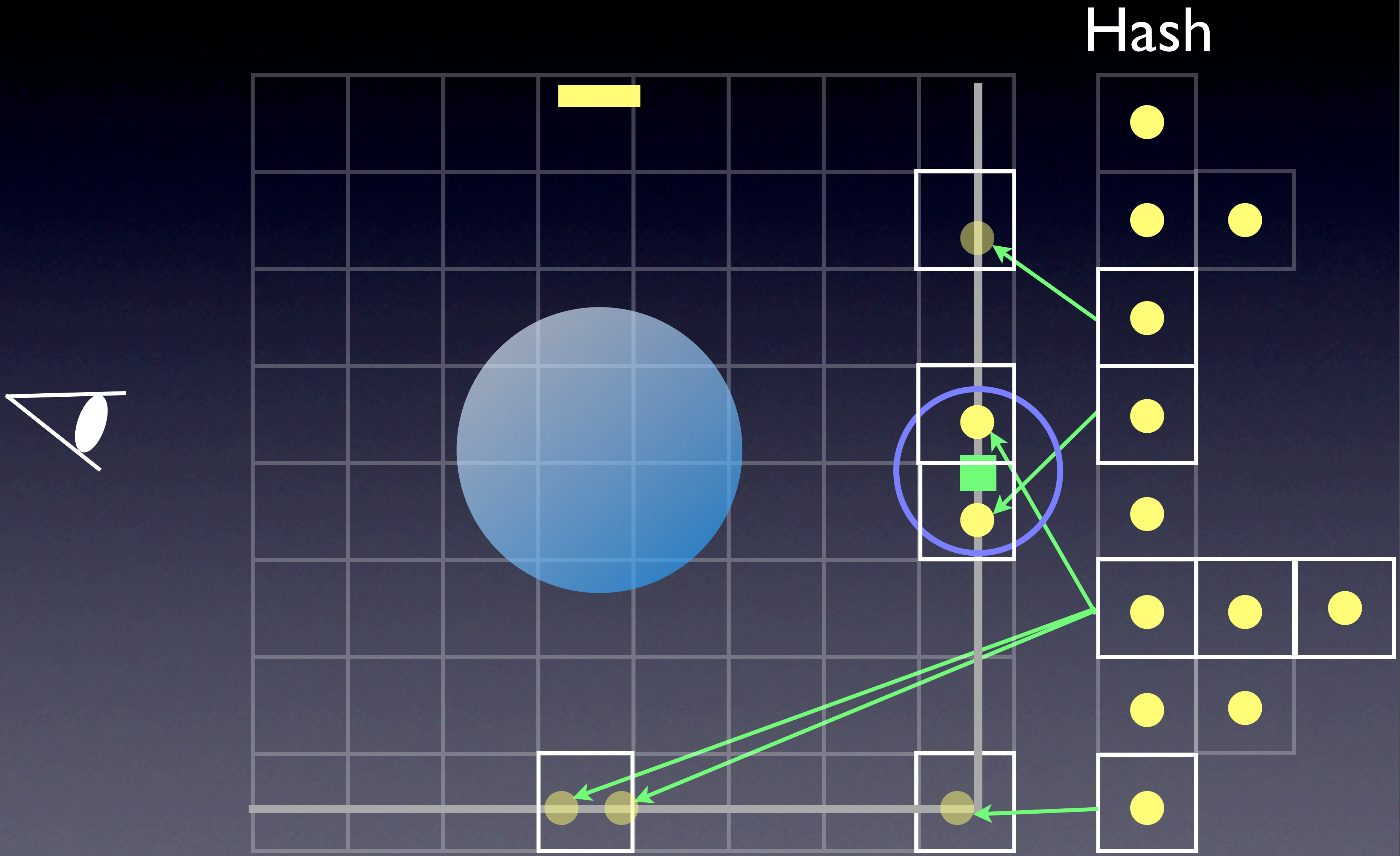
Spatial Hashing: Query



Spatial Hashing: Query



Spatial Hashing: Query



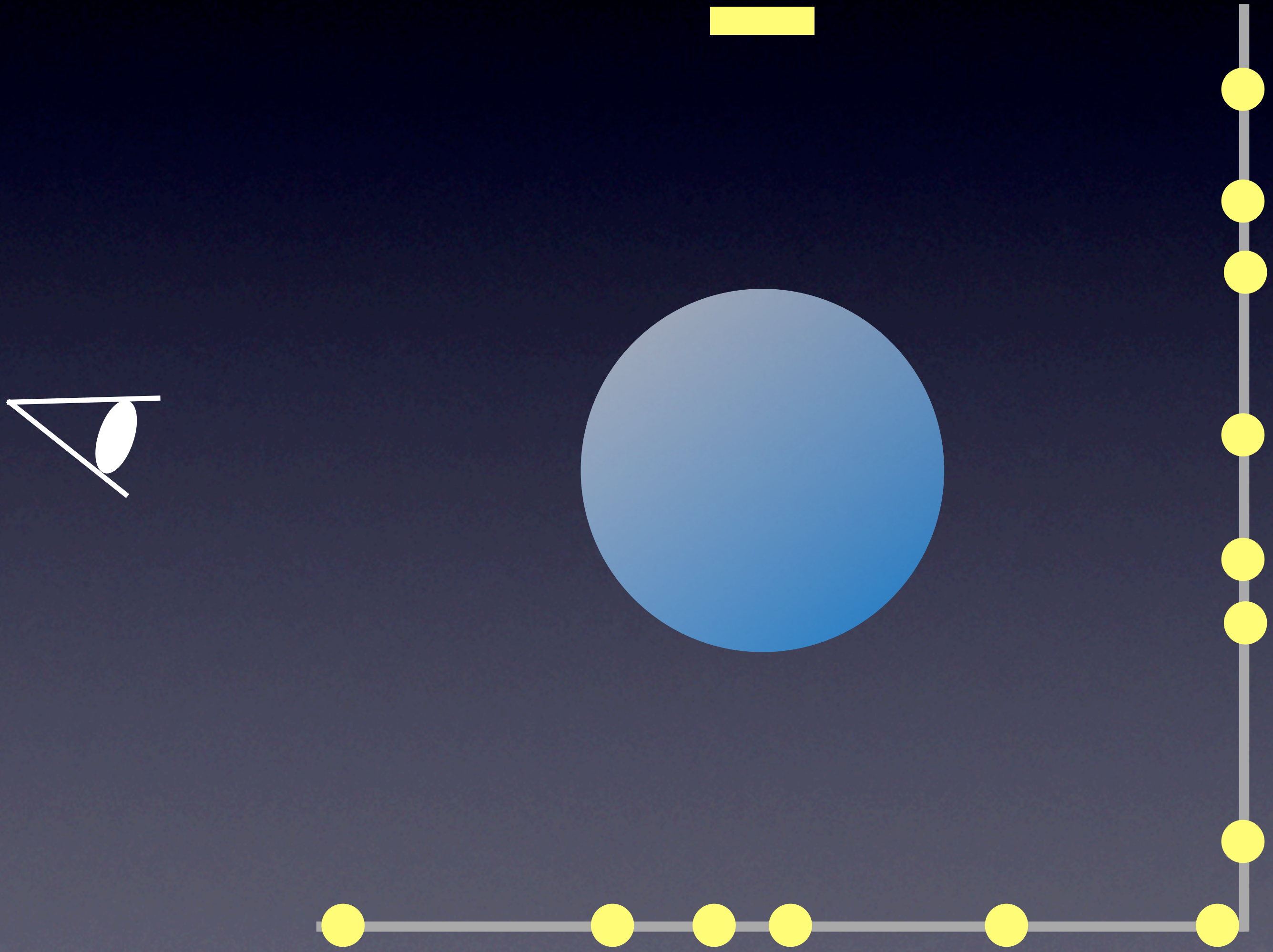
Issues

- Two fundamental issues
  - Construction of list is a serial process
  - Number of data fetches varies per cell

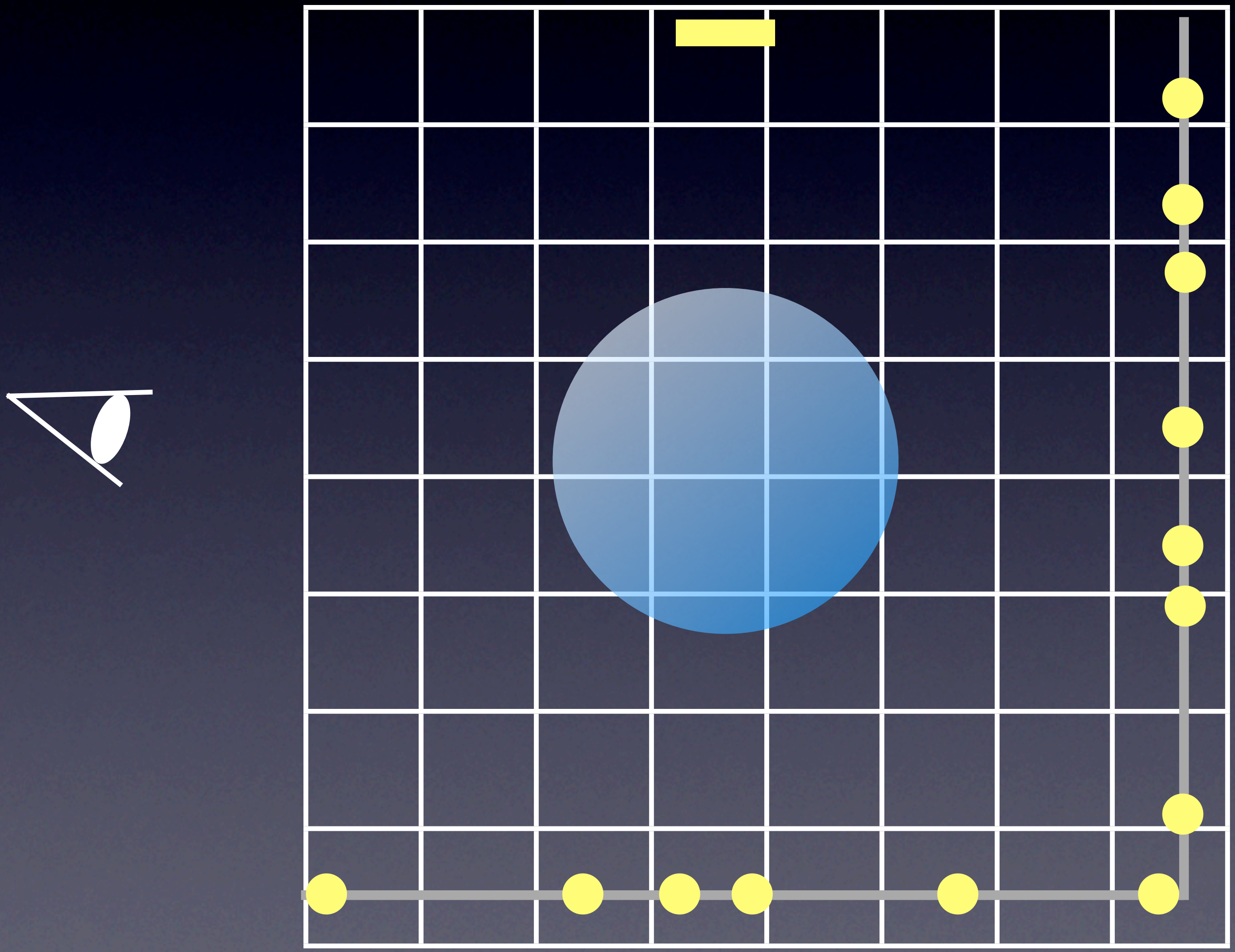
Our Solution

Keep only a single element stochastically

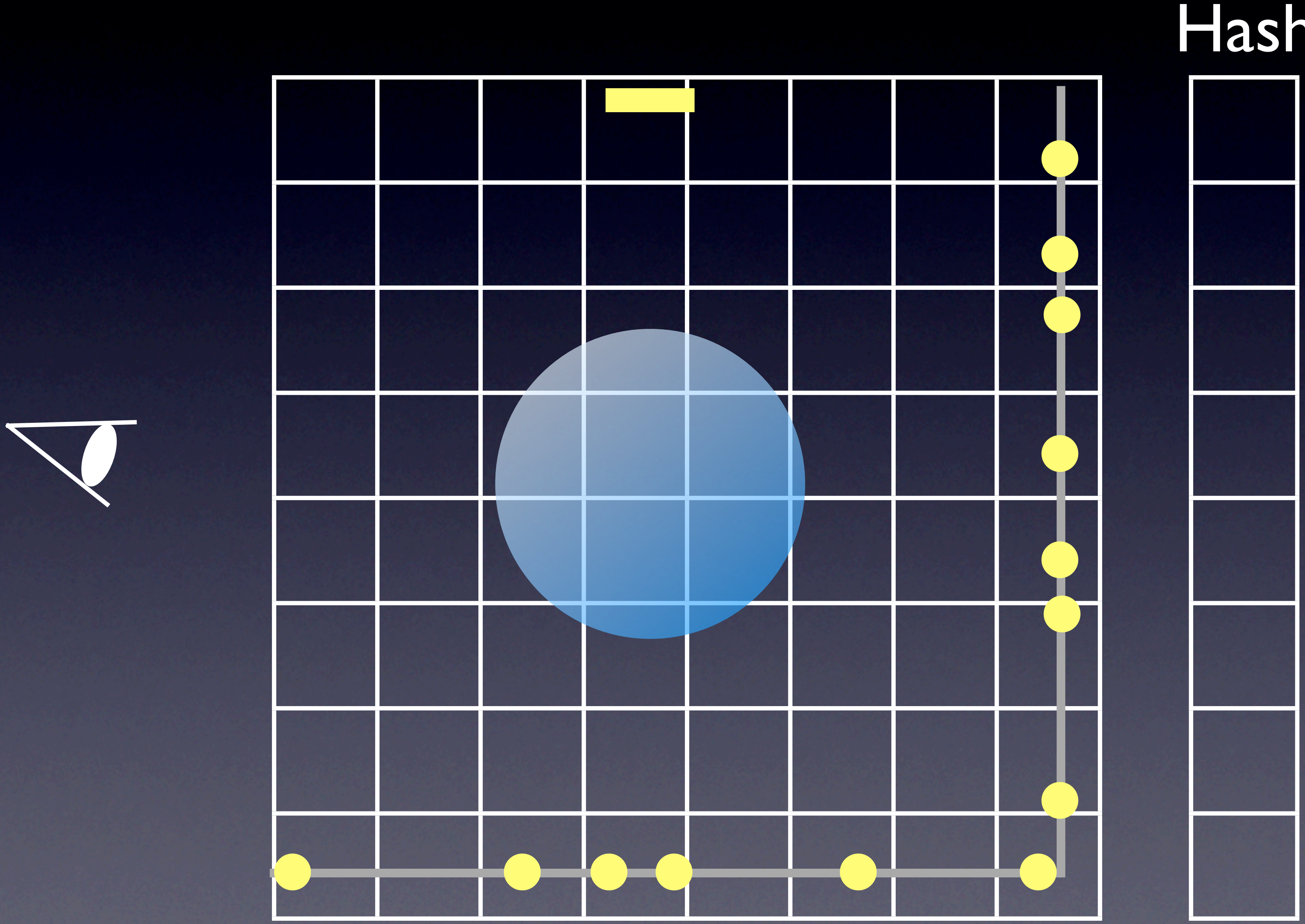
Stochastic Hashing: Construction



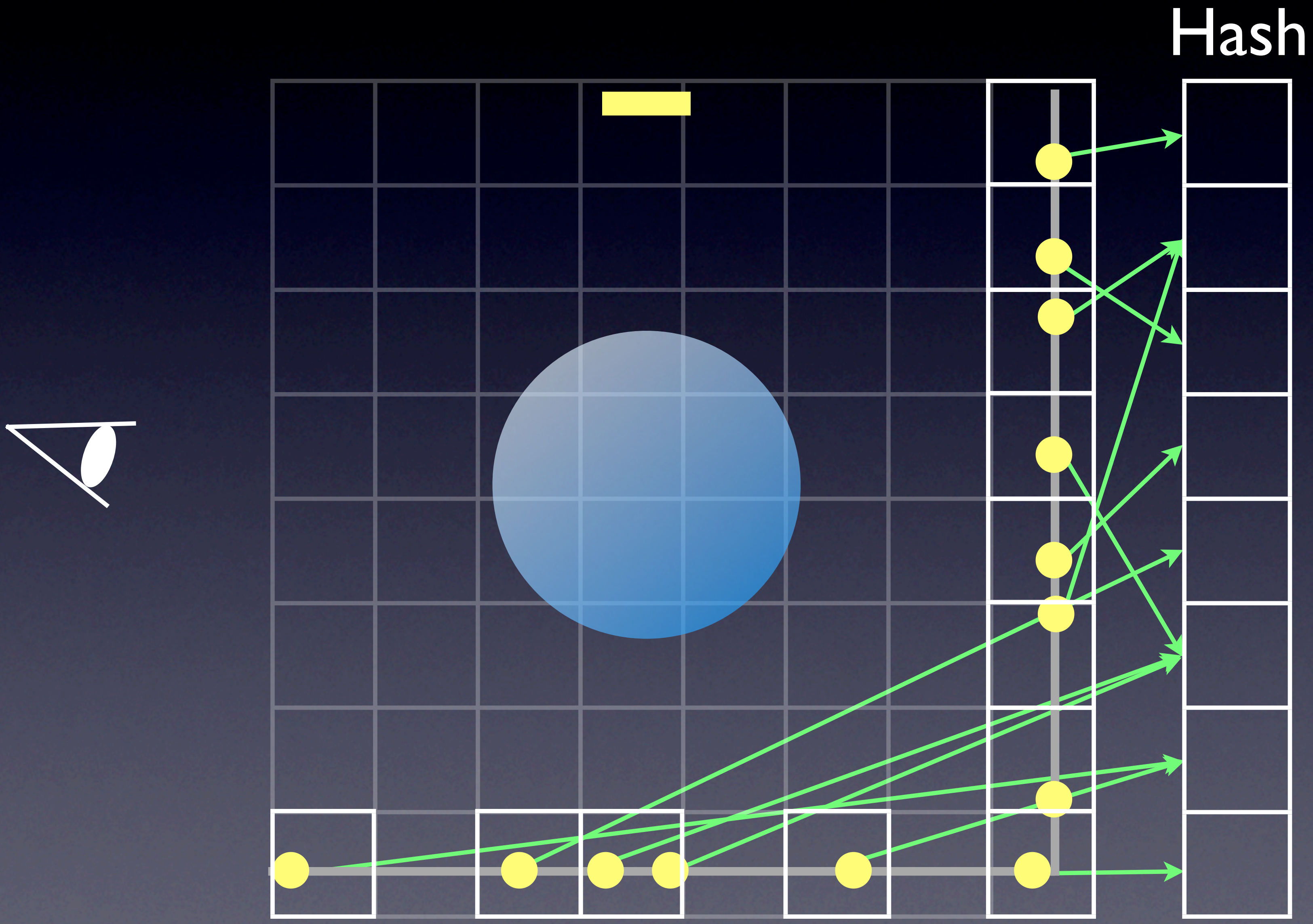
Stochastic Hashing: Construction



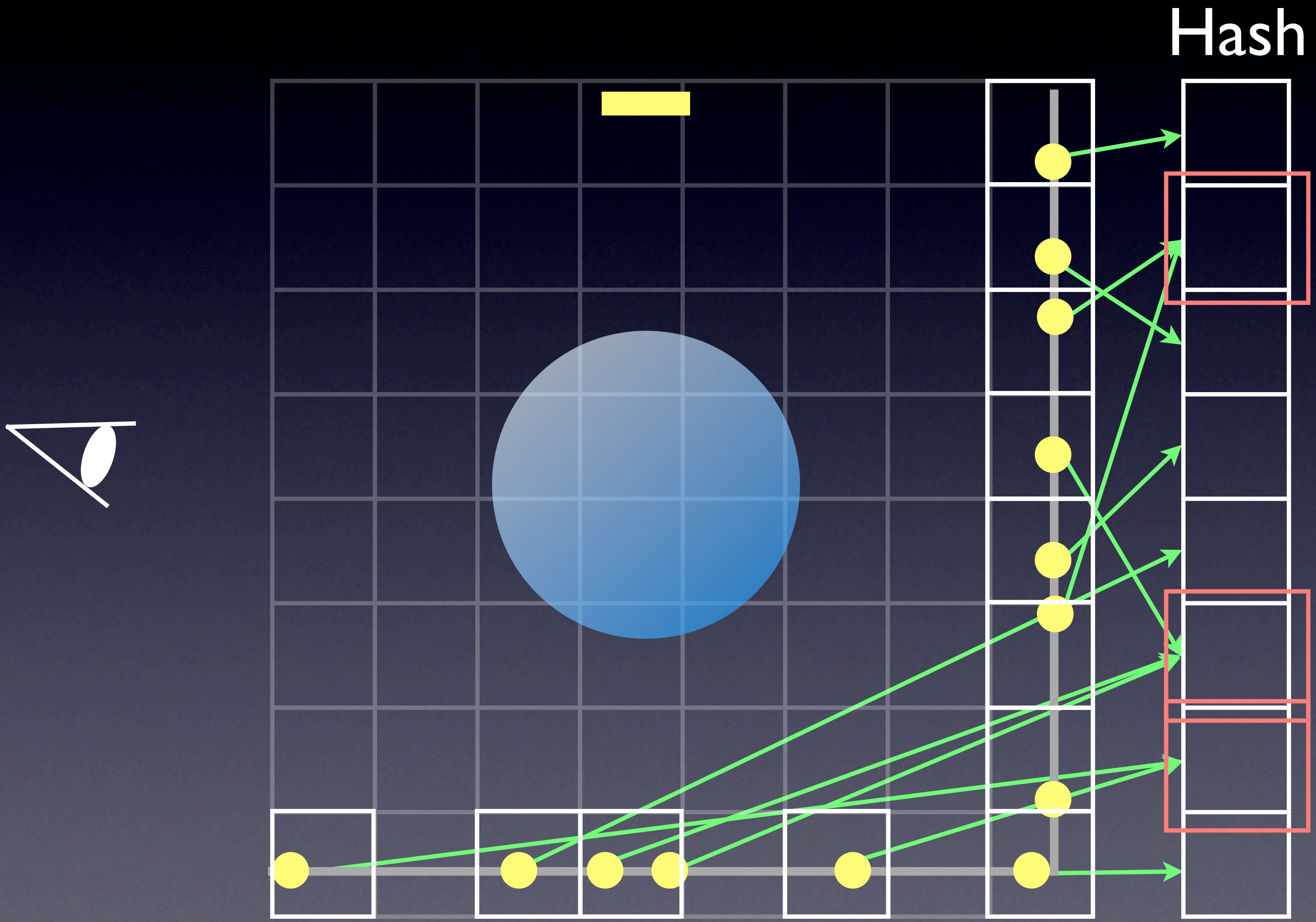
Stochastic Hashing: Construction



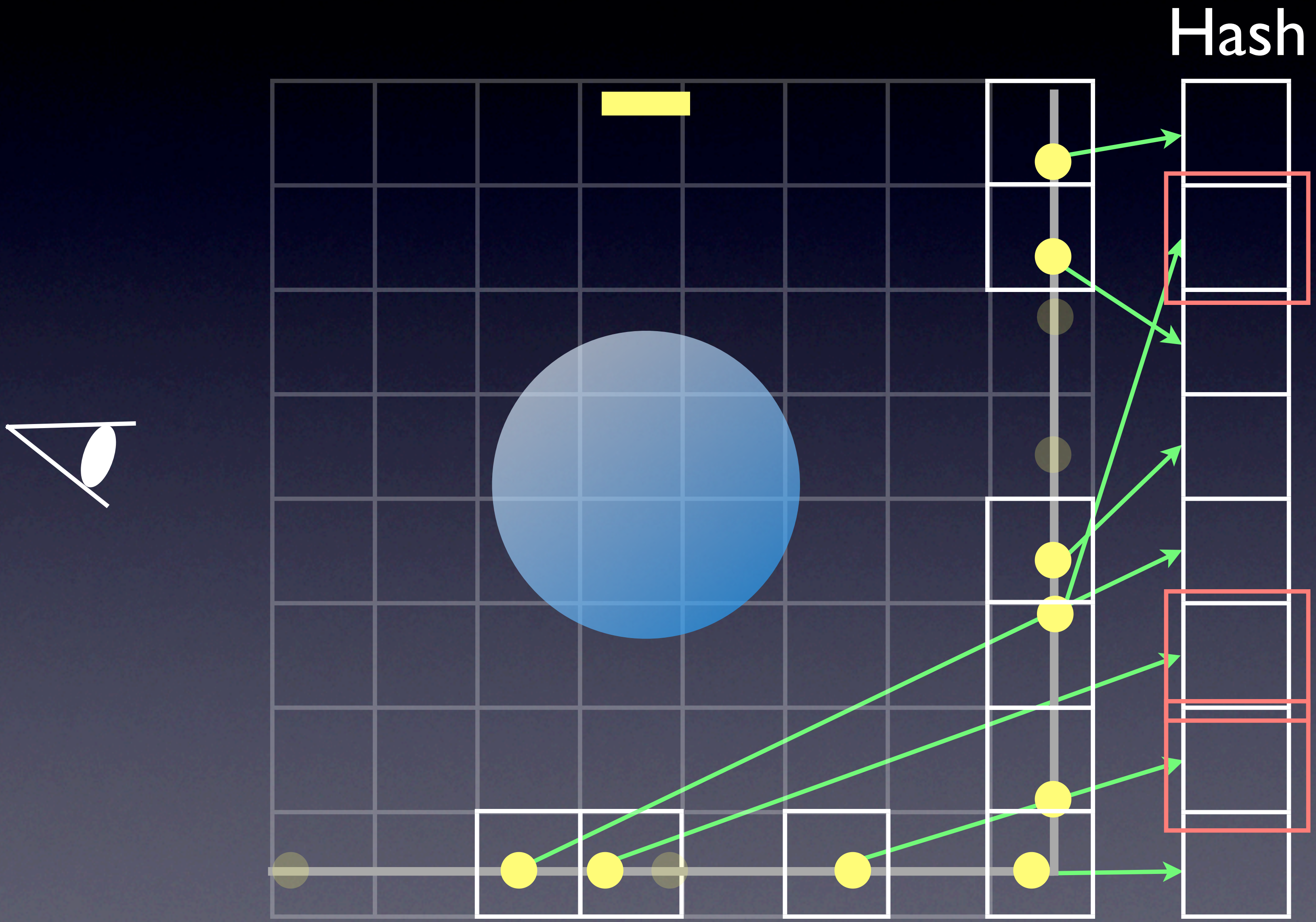
Stochastic Hashing: Construction



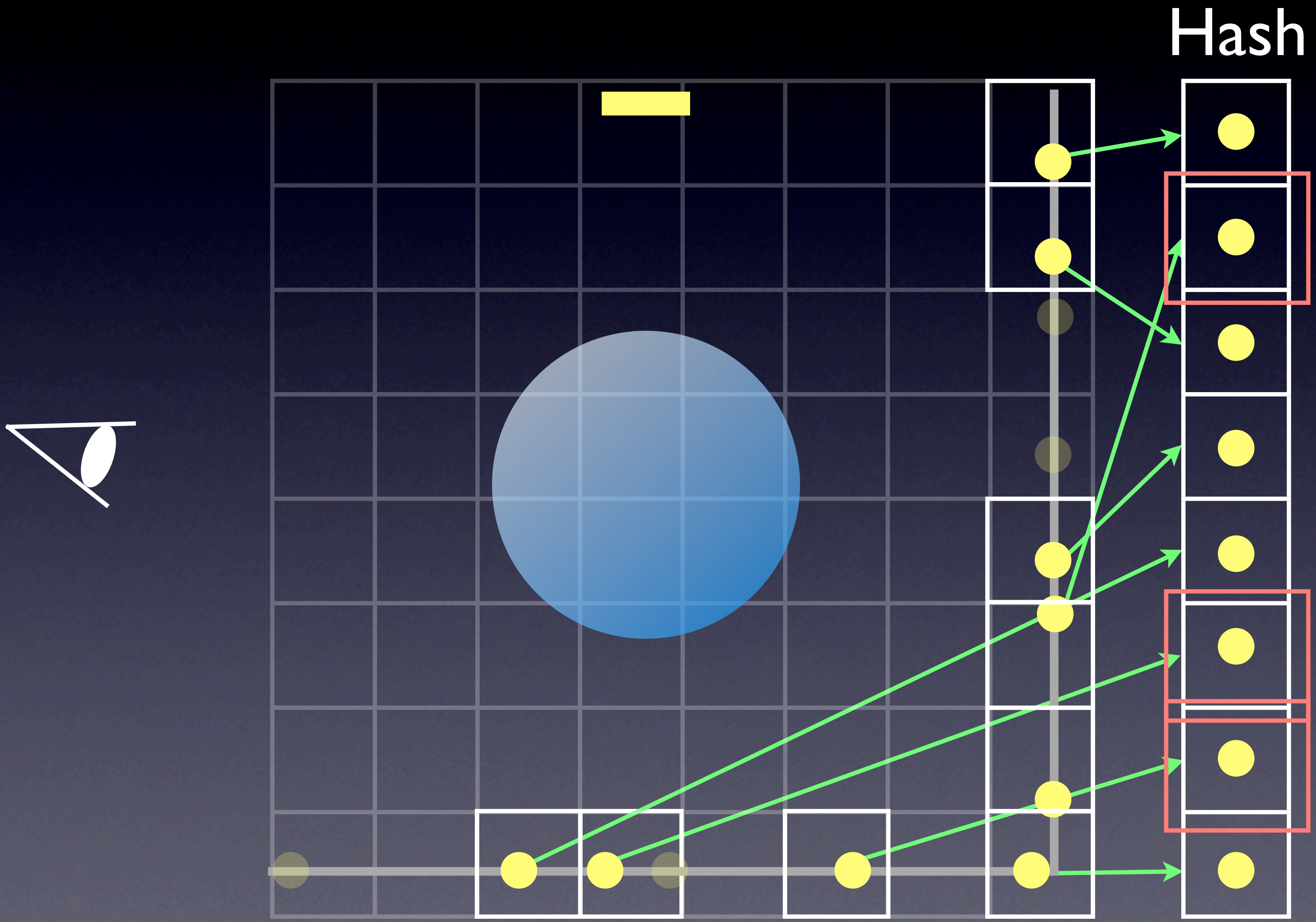
Stochastic Hashing: Construction



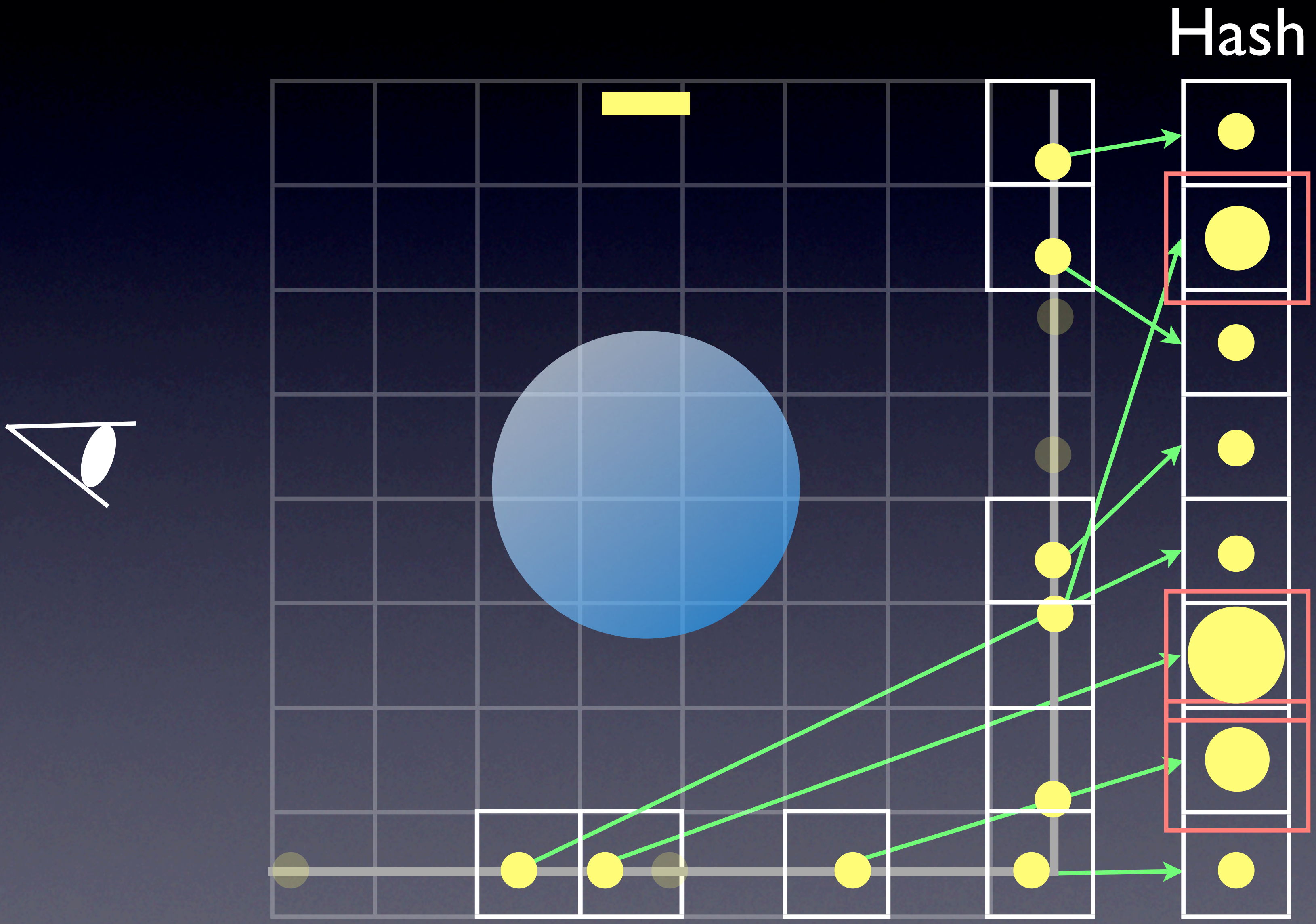
Stochastic Hashing: Construction



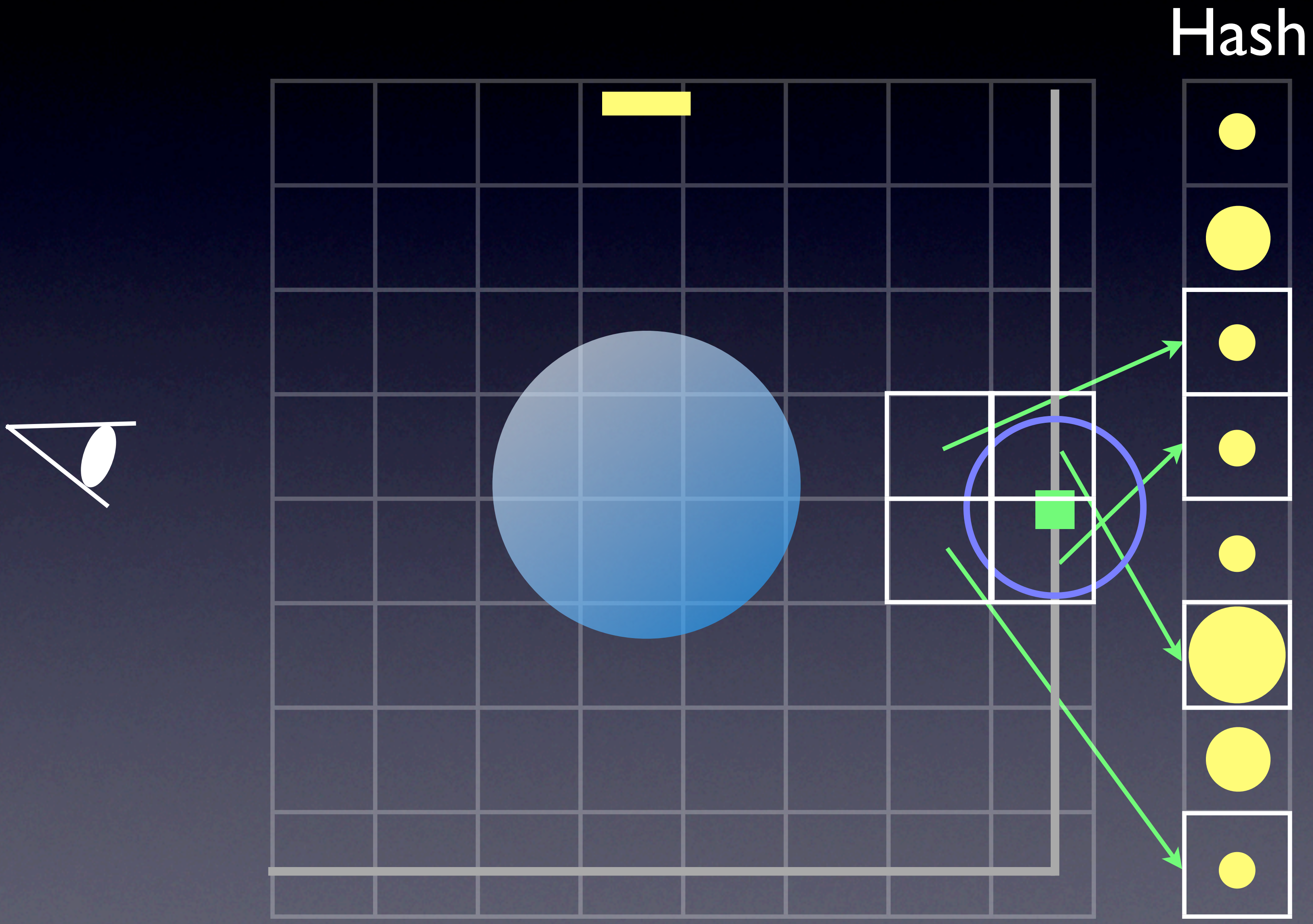
Stochastic Hashing: Construction



Stochastic Hashing: Construction



Stochastic Hashing: Query



Implementation

- Do we need a list to select an element?

Implementation

- Do we need a list to select an element?

No

Implementation

- Do we need a list to select an element?

No

- Just overwrite to the same place in parallel
  - Assume independent photon tracing
  - One of them should survive in the end

Implementation

```
For all photons in parallel
  HashIndex = Hash(Photon.Position)
  Table[HashIndex] = Photon
  AtomicInc(Count[HashIndex])
```

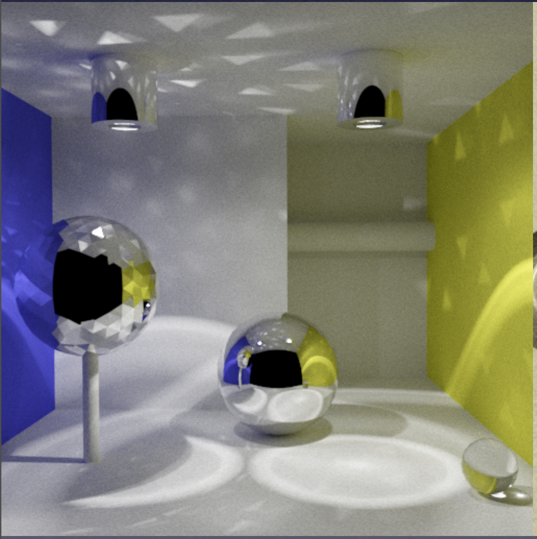
Related Work

- Photon splatting [Lavignotte 03]
- Uniform grid [Purcell 05]
- Cuckoo hashing [Alcantara 09]
- Tree data structure [Zhou 08][Fabianowski 09]
- Linked list [Thibieroz 09]

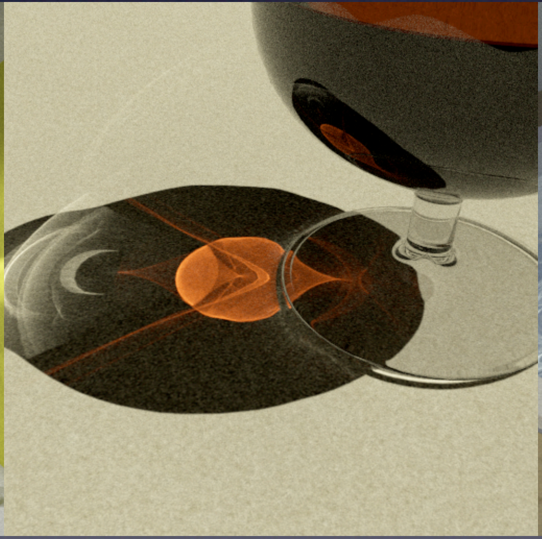
Results

Experiments Setup

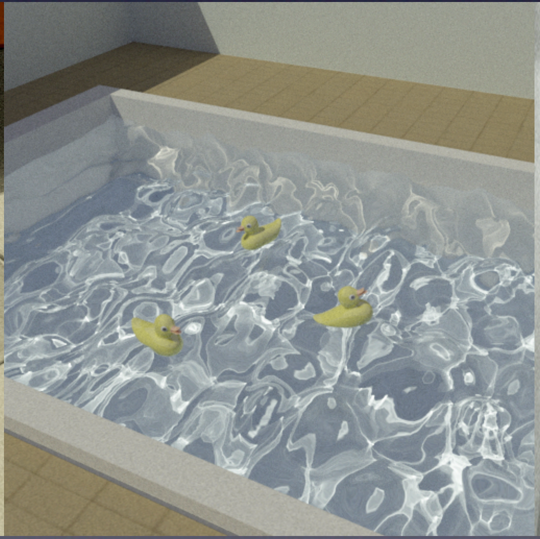
- Hash table size = Number of points
- Implemented using GLSL and NVIDIA OptiX
- Radeon HD 4850 and GeForce GTX 290



Box (4k)



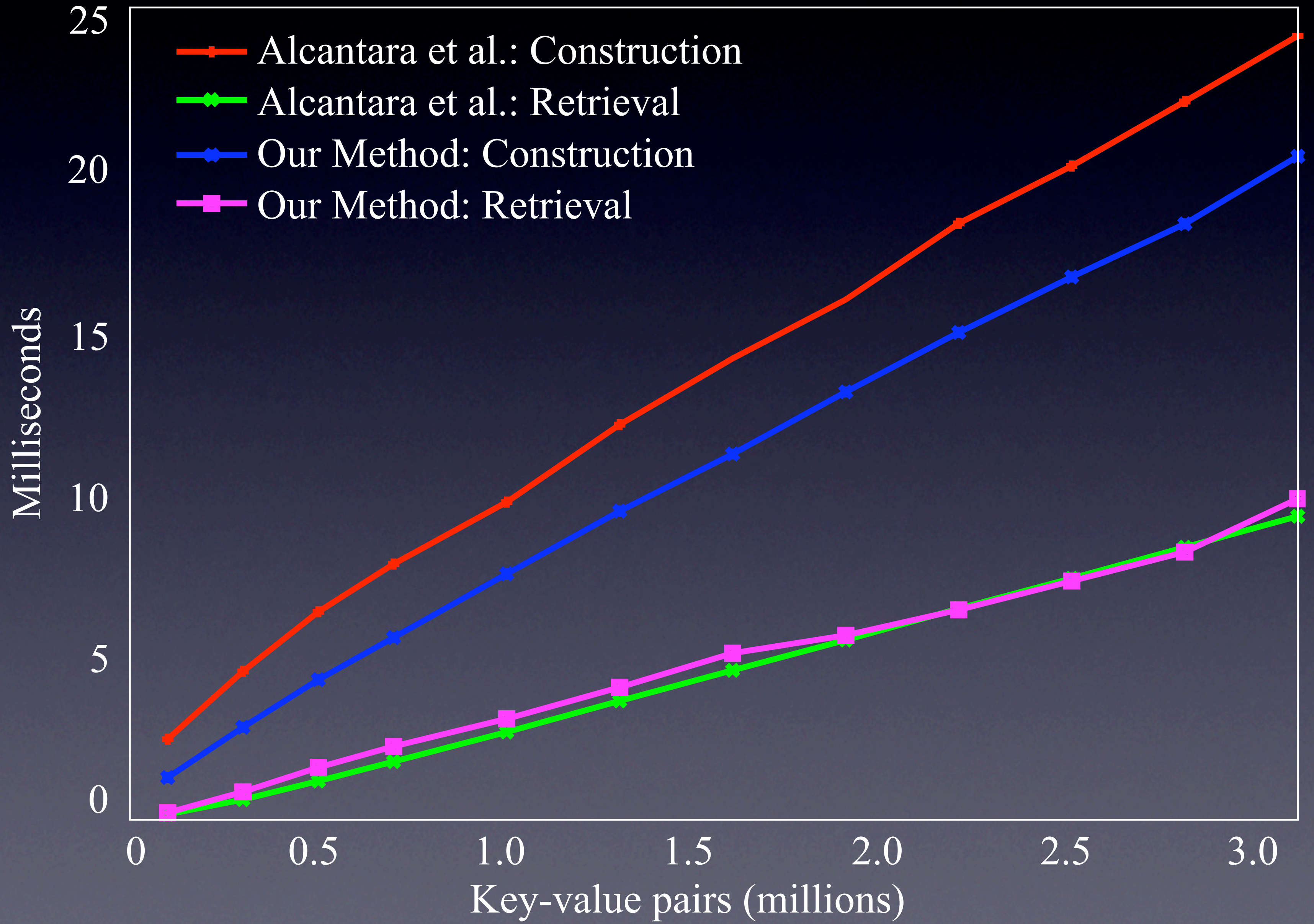
Cognac (16k)



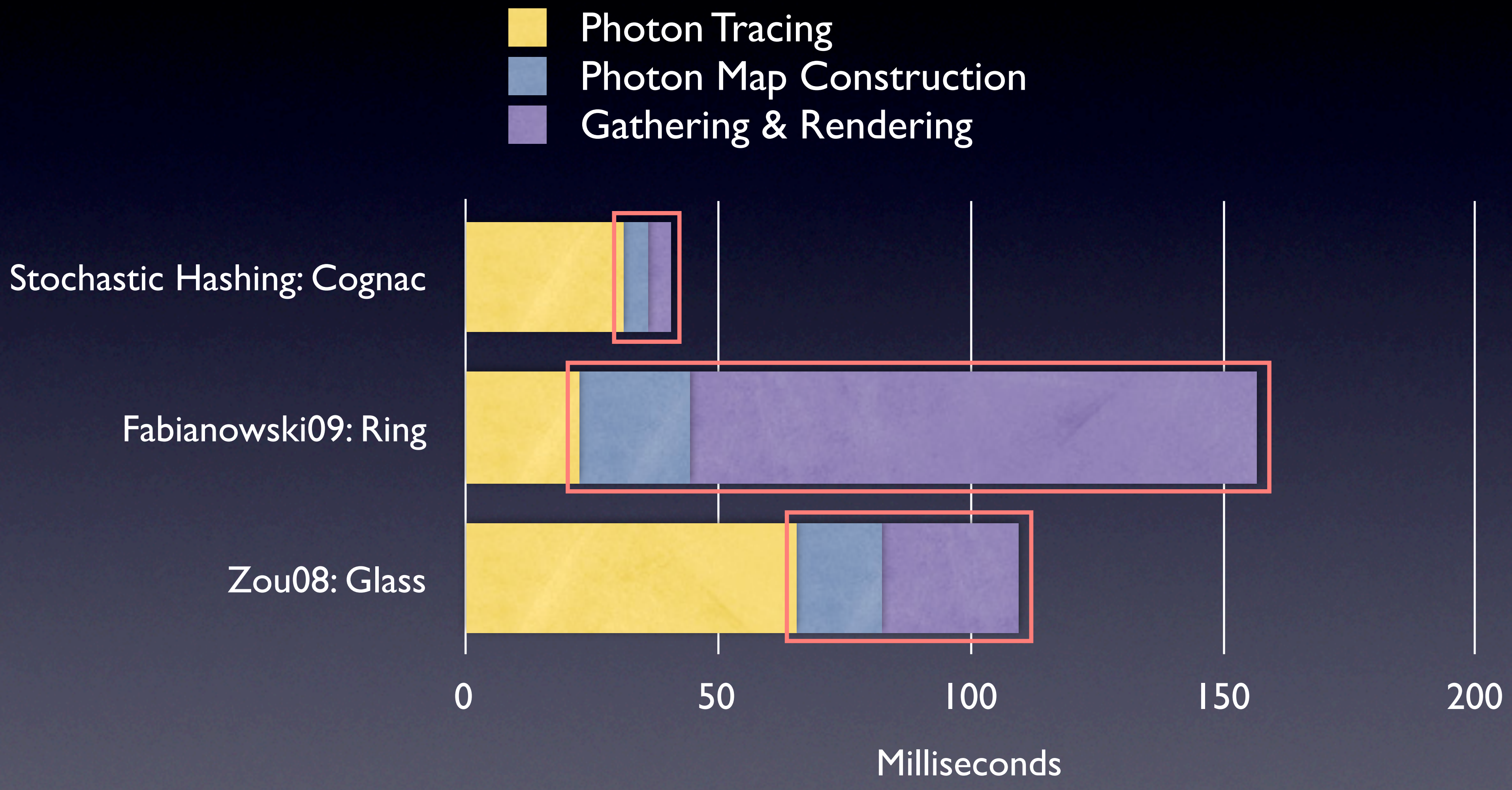
Pool (122k)

Kitchen (44k)

Random Points Test

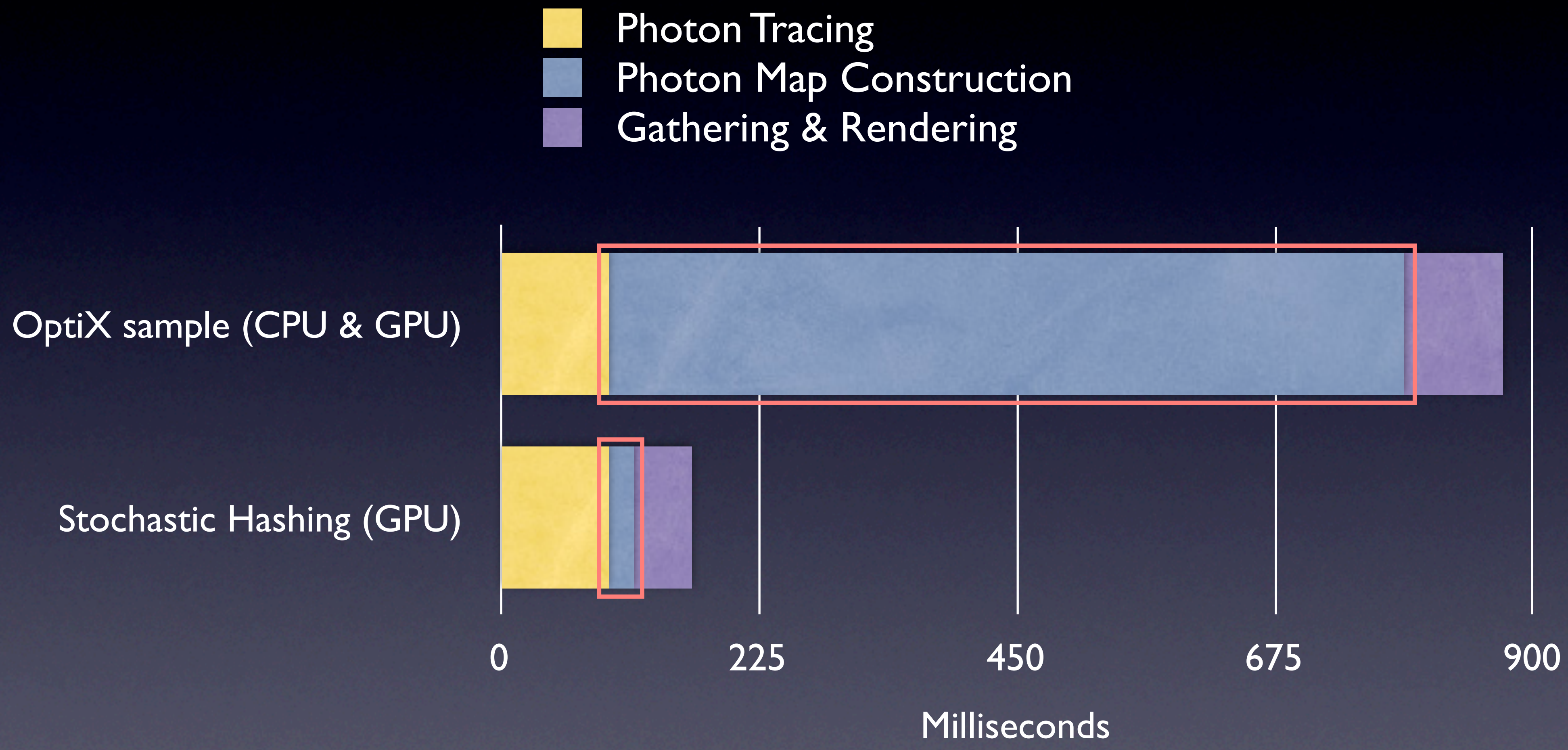


Rendering Time: Tree



Faster construction & gathering

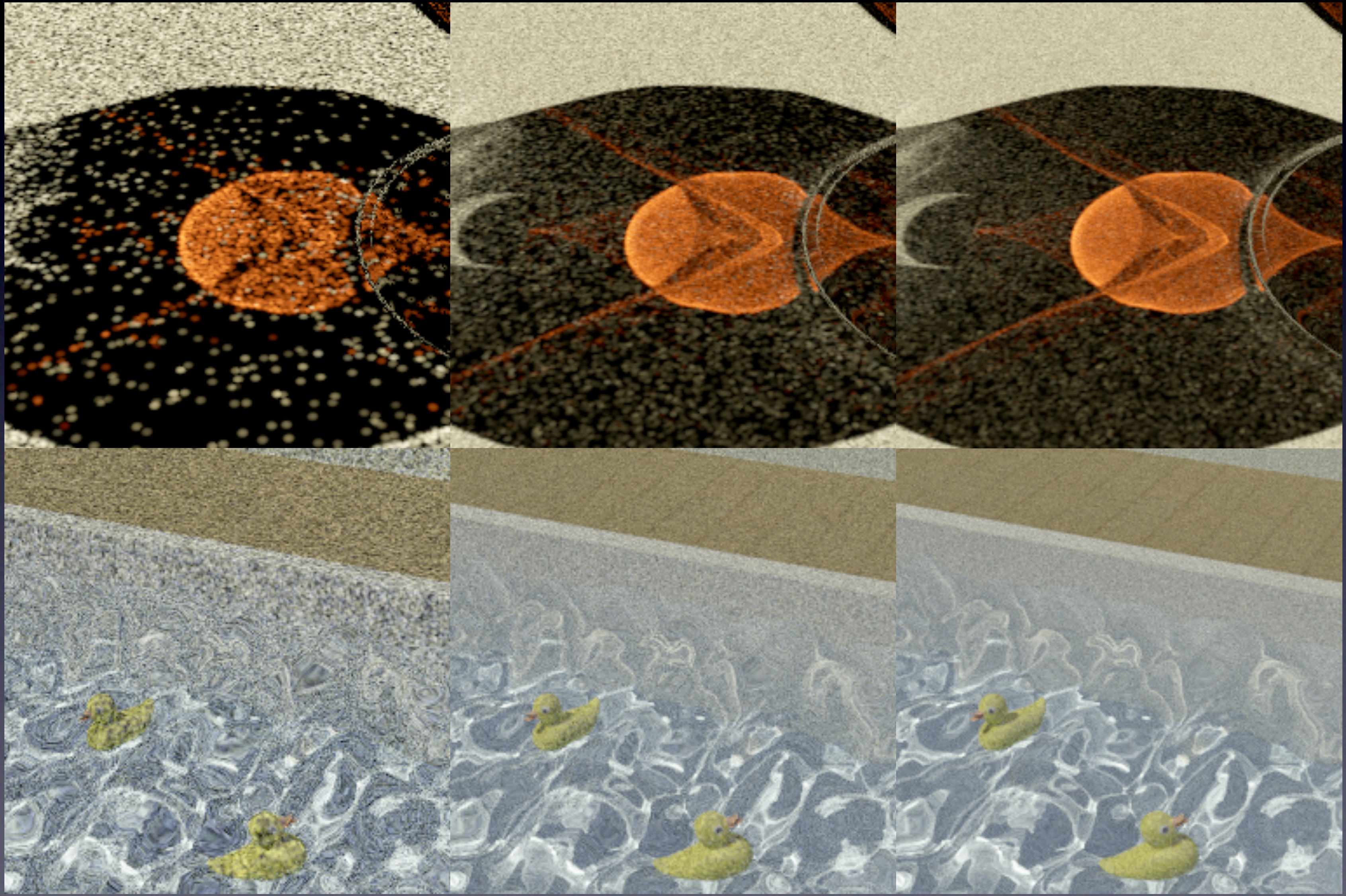
Rendering Time: CPU



Construction alone: 30x

Total: 5x

Additional Noise

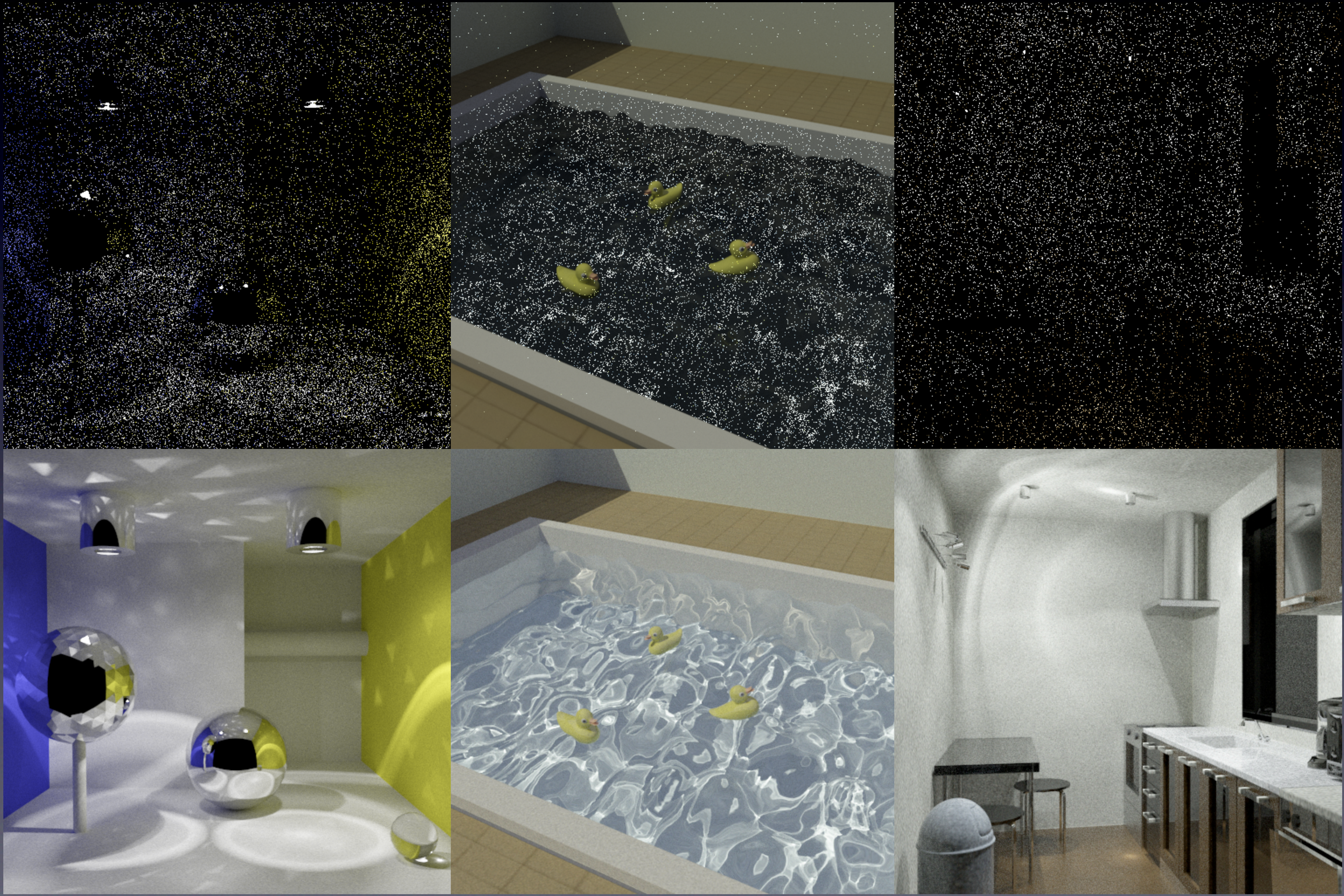


1:64 table

1:1 table

Full list

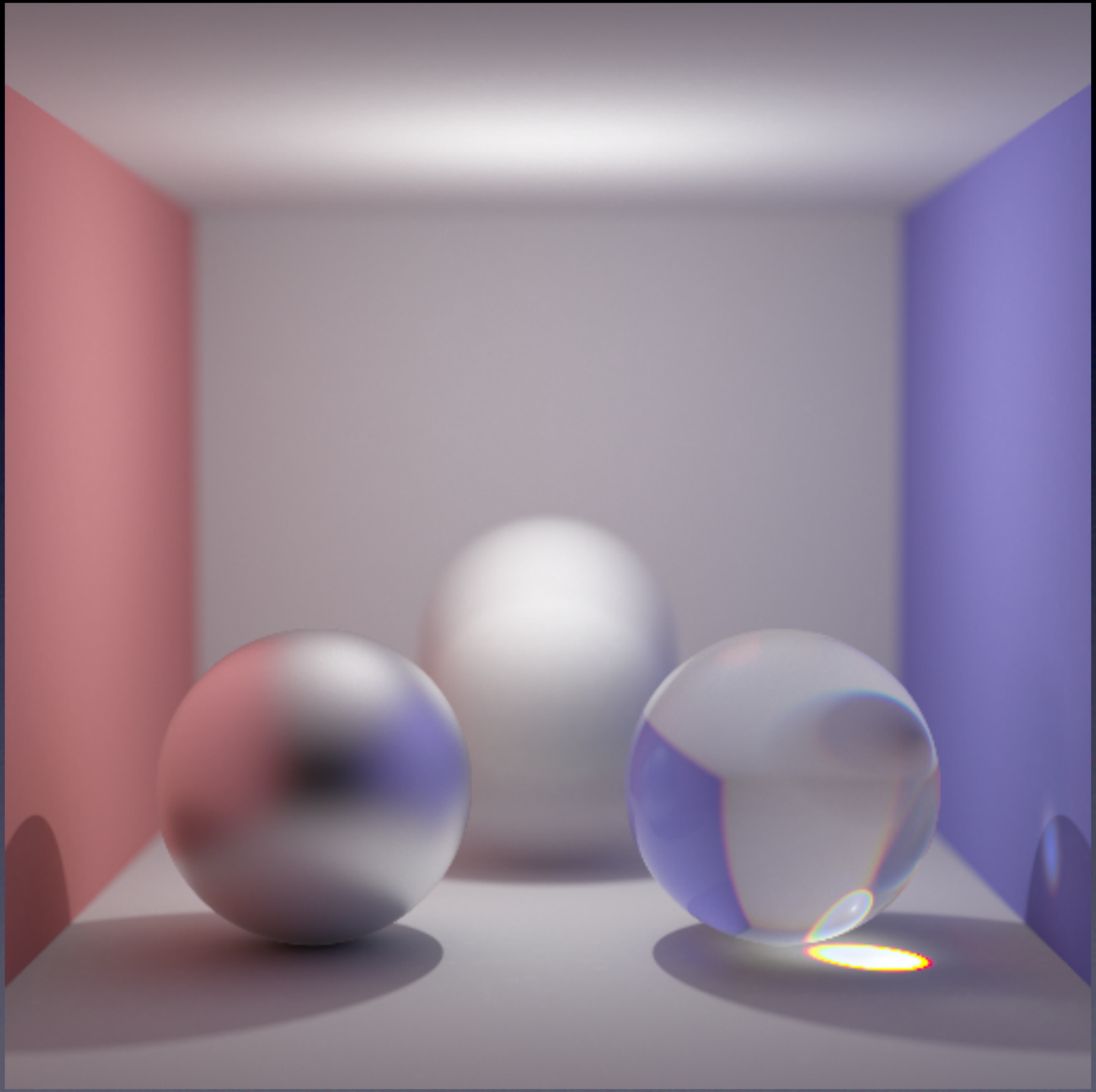
Robustness



Path tracing

PPM

GPUSPPM



graphics.ucsd.edu/~toshiya

Conclusion

- Parallel progressive photon mapping
    - Fast construction using stochastic hashing
    - Suitable for parallel processors (aka GPUs)
    - Easy to implement

"Please ~~do not~~ try this at home"

Acknowledgements

- Dan Alcantara
- NVIDIA Fellowship 2010-2011
- ompf.org forum members