GPU-based Scene Management for Rendering Large Crowds

Joshua Barczak, Natalya Tatarchuk,
Christoper Oat

SIGGRAPHASIA2008
NEW HORIZONS

Outline

- Motivation
- GPU Crowds
  - Management
  - Rendering
- Conclusion

Motivation

Motivation

- Need scalability and stable performance
- Don't want to render thousands of million polygon characters
  - Wasteful if details are unseen
- CPU-side character management is impractical when doing GPU simulation
  - Requires a read-back
- *Solution*: Perform GPU-side scene management

Scene Management

GPU Scene Management

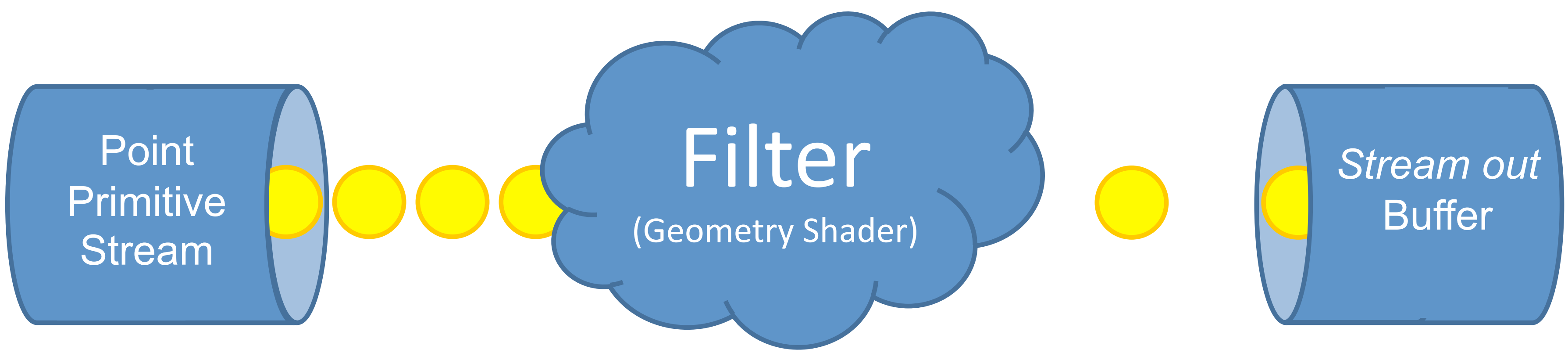
- Vertex buffer containing all per-instance data
  - GPU-based crowd simulation
  - CPU-based simulation works too
- Need to perform typical scene management tasks
  - Frustum cull
  - Occlusion cull
  - Several discrete LODs
  - Parallel split shadow map frustum selection
- How do we move all this to GPU?

Geometry Shaders as *Filters*

- Act on instances
- A set of point primitives (instance data) as input
- Re-emit only points that pass a specific test
  - Discard the rest
  - *DrawAuto* used to chain multiple filters

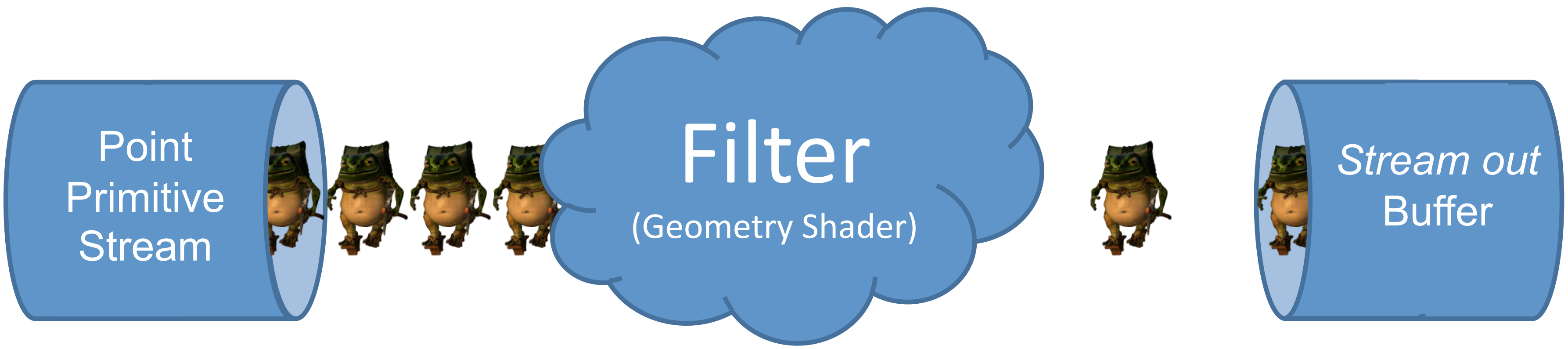
Stream filtering using *Stream Out*

- Act on instances
- A set of point primitives (instance data) as input
- Re-emit only points that pass a specific test
  - Discard the rest
  - *DrawAuto* used to chain multiple filters



Stream filtering using *Stream Out*

- Act on instances
- A set of point primitives (instance data) as input
- Re-emit only points that pass a specific test
  - Discard the rest
  - *DrawAuto* used to chain multiple filters



Filters Manage Crowd Complexity

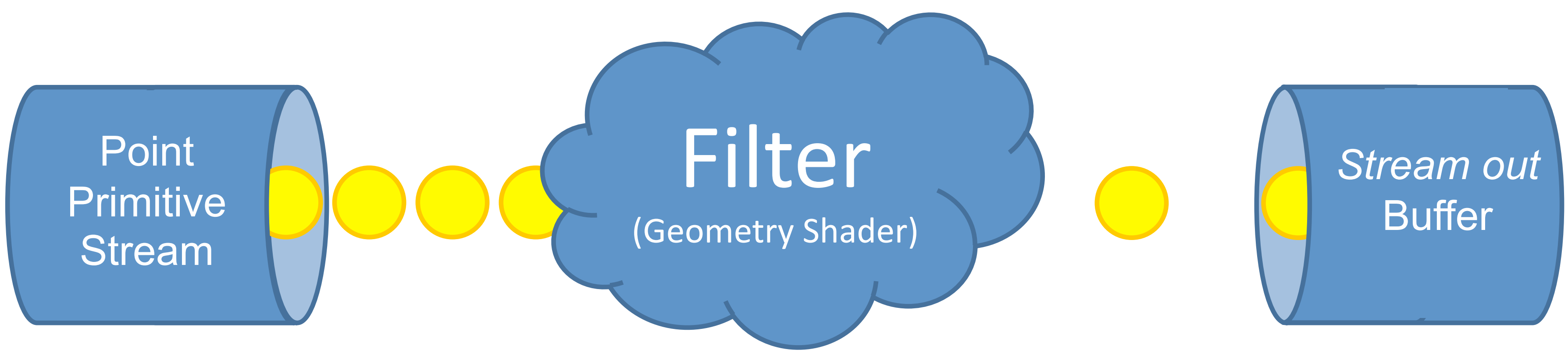
- Different filters for:
  - View frustum culling
  - Occlusion culling
  - LOD Selection
  - Shadow frustum selection

View Frustum Culling

- Filter removes characters outside view frustum
  - Checks for intersection between character’s bounding volume and the view frustum

View Frustum Culling

- Filter removes characters outside view frustum
  - Checks for intersection between character's bounding volume and the view frustum
  - If test **passes**, character is **in view**: <u>emit it</u>
  - If test **fails**, character is **out of view**: <u>discard it</u>

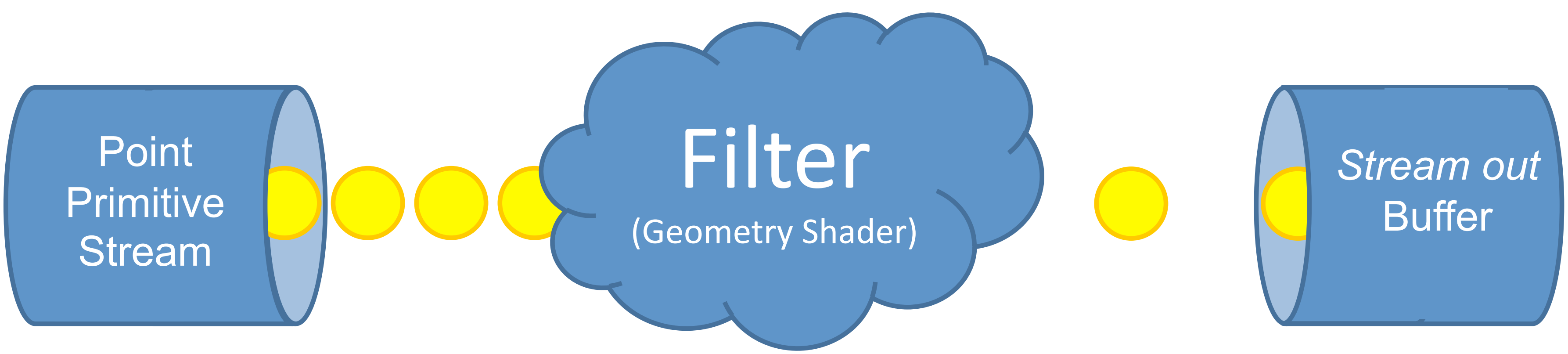


View Frustum Culling

- Filter removes characters outside view frustum
  - Checks for intersection between character's bounding volume and the view frustum
  - If test **passes**, character is **in view**: <u>emit it</u>
  - If test **fails**, character is **out of view**: <u>discard it</u>
- Output is buffer of potentially visible characters
- Output becomes input to subsequent filters

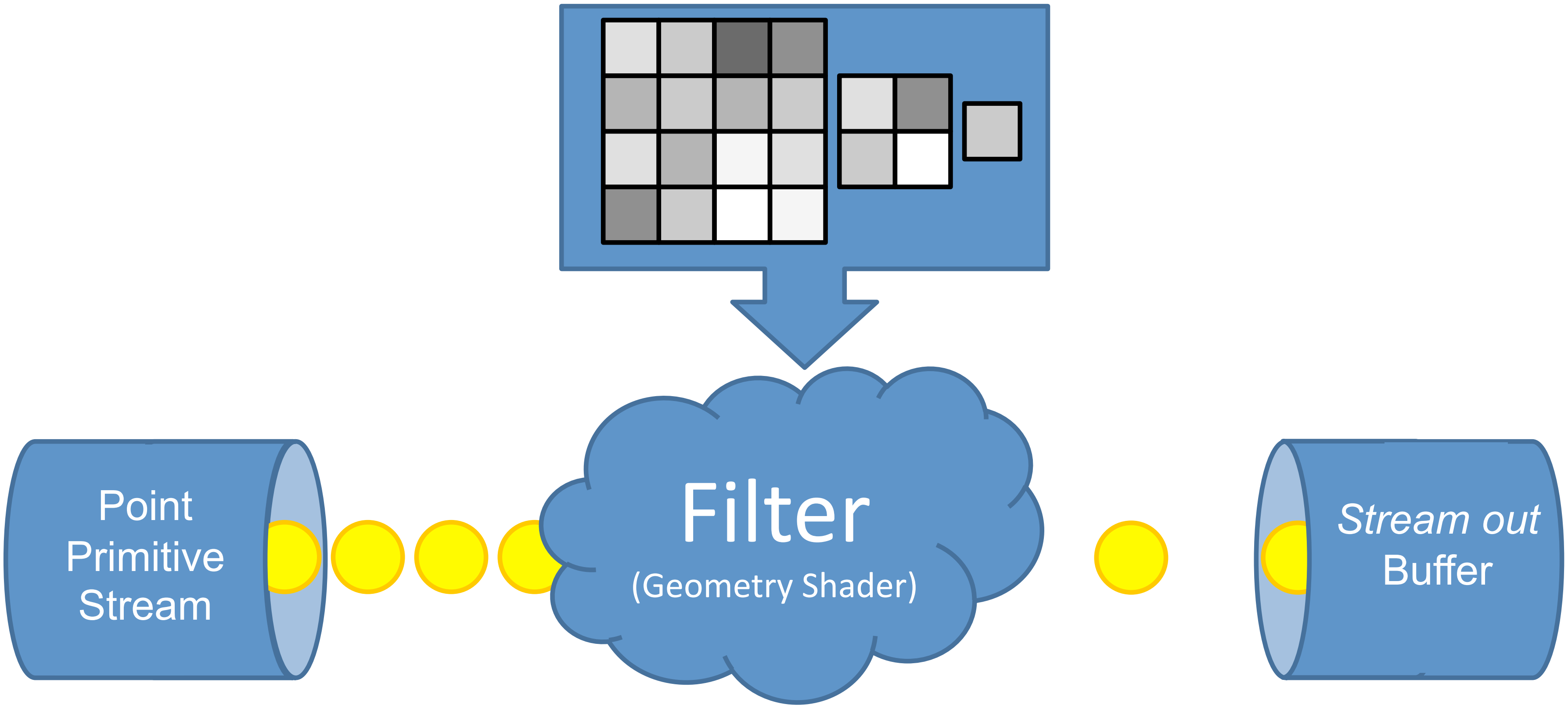
Occlusion Culling

- Determine which characters are occluded by the environment or structures



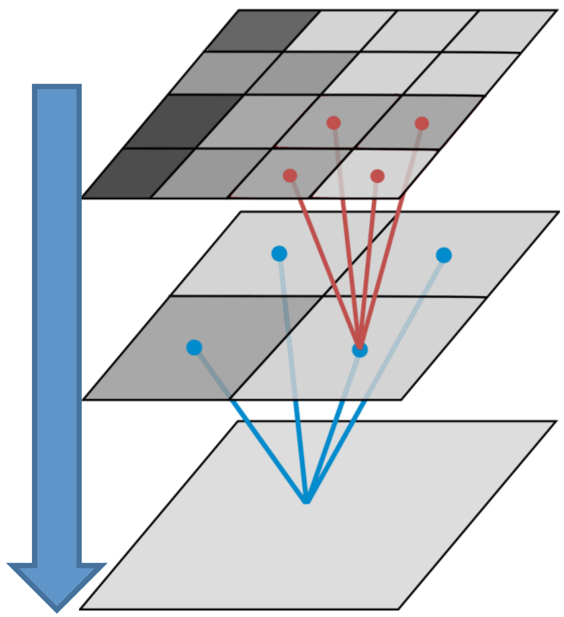
Occlusion Culling

- Determine which characters are occluded by the environment or structures
- Filter requires additional input: *Hierarchical Depth Image*



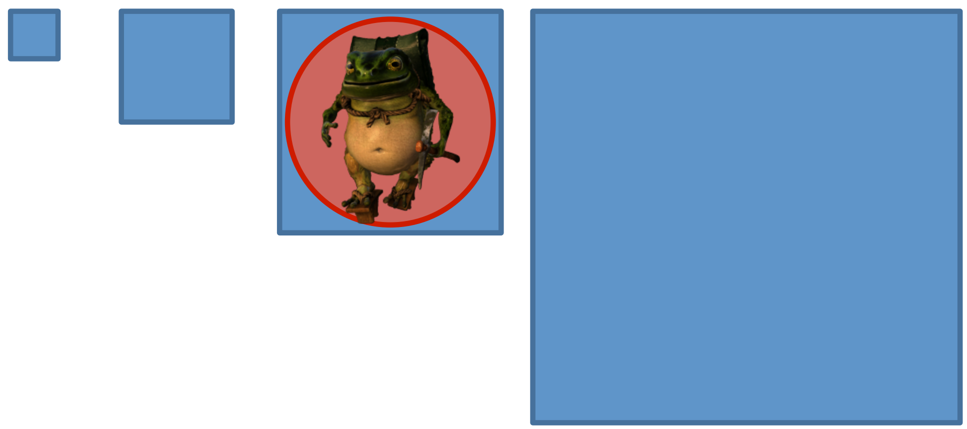
Hierarchical Depth Image

- Occlusion Culling
  - Generate hierarchical Z (Hi-Z) buffer from scene depth buffer [Greene et al 1993]



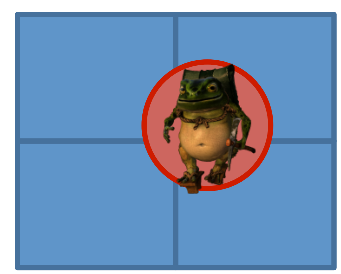
Hierarchical Depth Image

- Occlusion Culling
  - Generate hierarchical Z (Hi-Z) buffer from scene depth buffer [Greene et al 1993]
  - Each character chooses MIP level based on bounding volume



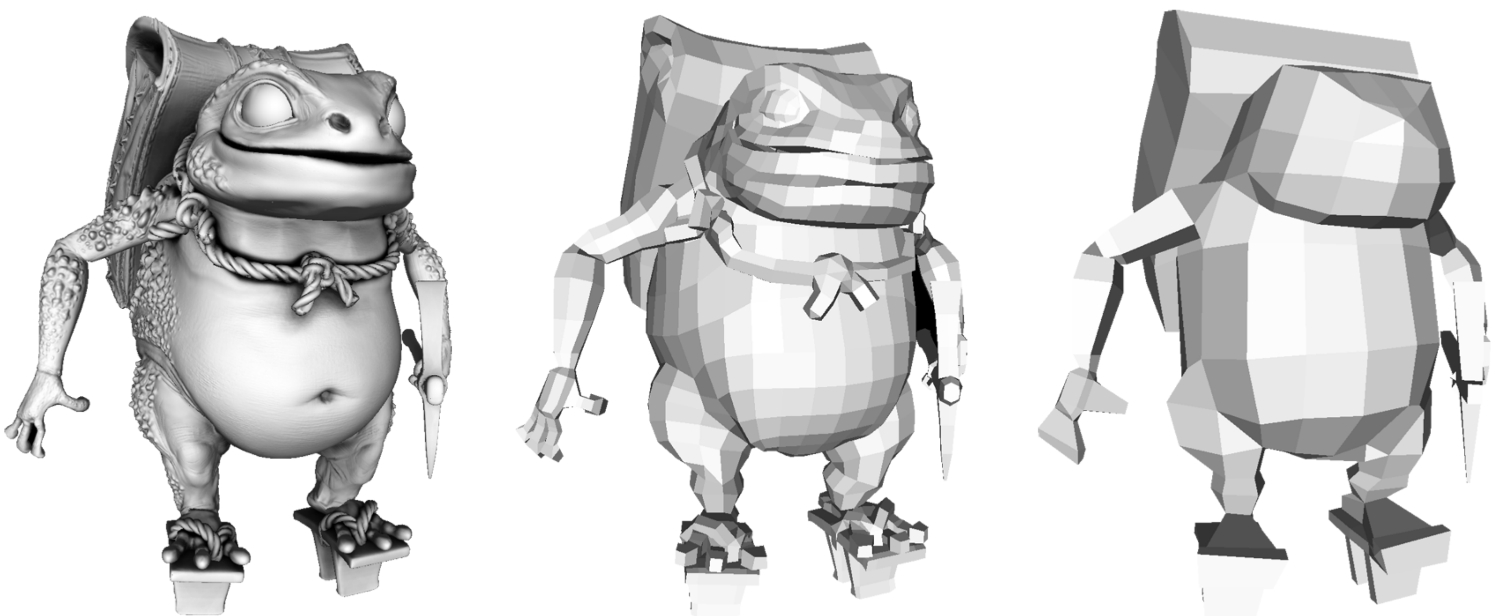
Hierarchical Depth Image

- Occlusion Culling
  - Generate hierarchical Z (Hi-Z) buffer from scene depth buffer [Greene et al 1993]
  - Each character chooses MIP level based on bounding volume
  - Projected depth of character’s bounding sphere tested against four texels in chosen MIP level



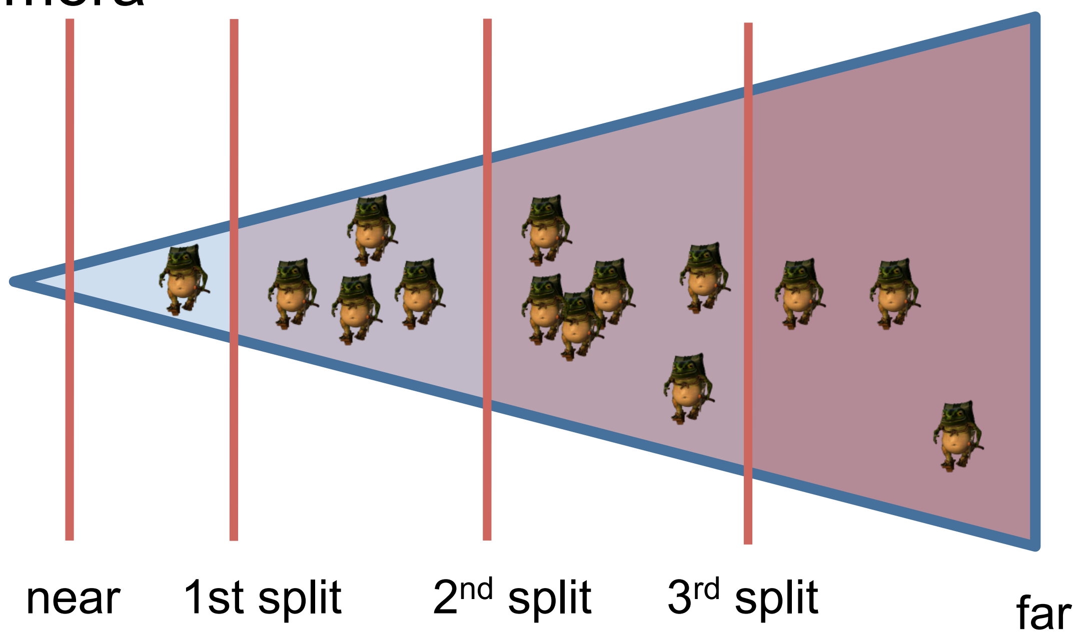
LOD Selection

- Agents filtered using distance from camera to centroid
- Uses results of culling filters buffer
- We use three levels of detail
  – Three filter passes into three buffers



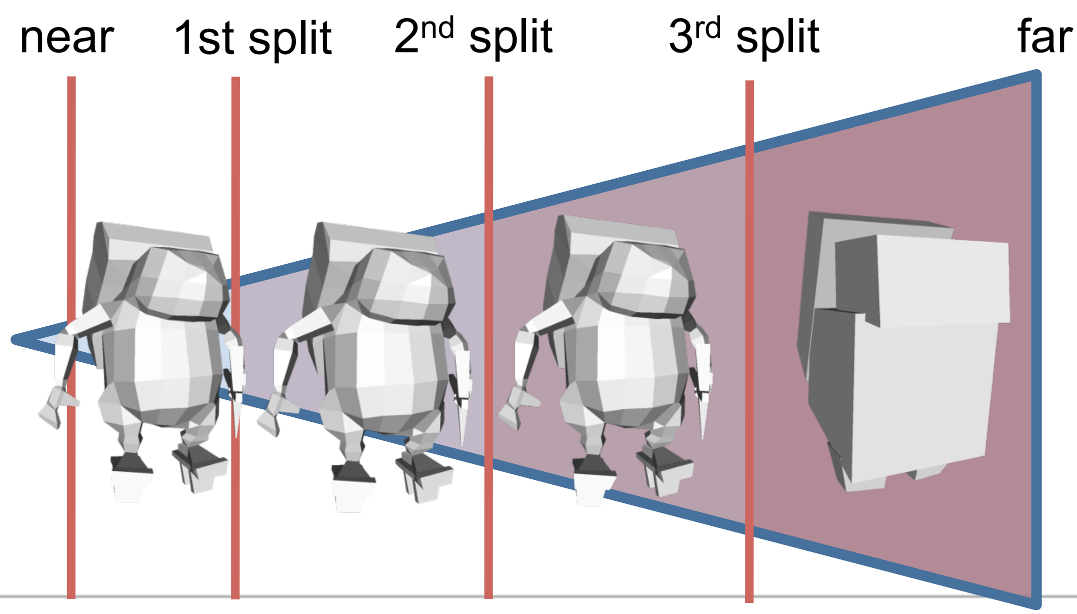
Shadows

- *Parallel Split Shadow Maps* [Zhang et al. 2006]
  – Several shadow maps, selected by distance from camera



Shadows

- Appropriate shadow map chosen per-character based on split distance from camera
- Character LOD based on split distance



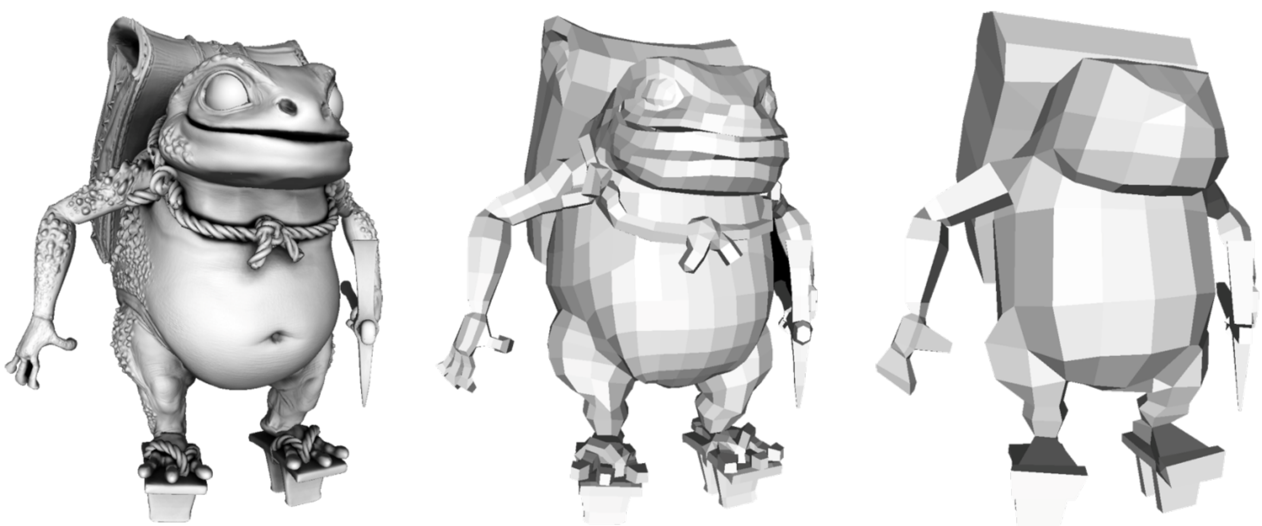
Character Rendering

Organize Draw Calls Around Queries

- Need instance count for issuing the draw call for each LOD
- This requires a stream out stats query
  - Can cause significant stall when results are used in the same frame issuing the query
- Re-organize the draw-calls to fill the gap between issuing the query and using the results
  - We perform AI simulation steps

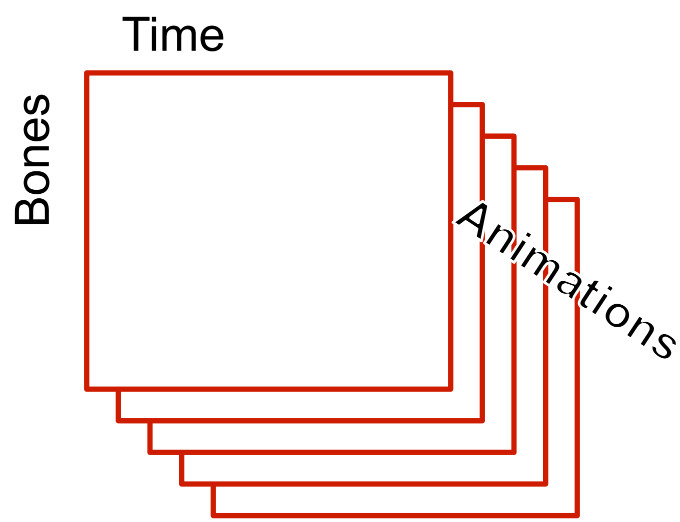
Character Rendering

- *DrawInstanced()* call for each LOD
- Hardware tessellation and displacement mapping for closest LOD
- Conventional rendering for middle LOD
- Simplified geometry for farthest LOD



Character Animation

- Skeletal animations sampled into texture array
- Packed animation data sampled by character's vertex shaders



Conclusions

- Dealing with large crowds of instanced characters can be expensive
- Leverage GPU for crowd management
  - Frustum & visibility culling
  - LOD selection
  - Shadow frustum selection
  - Character animation

Questions?

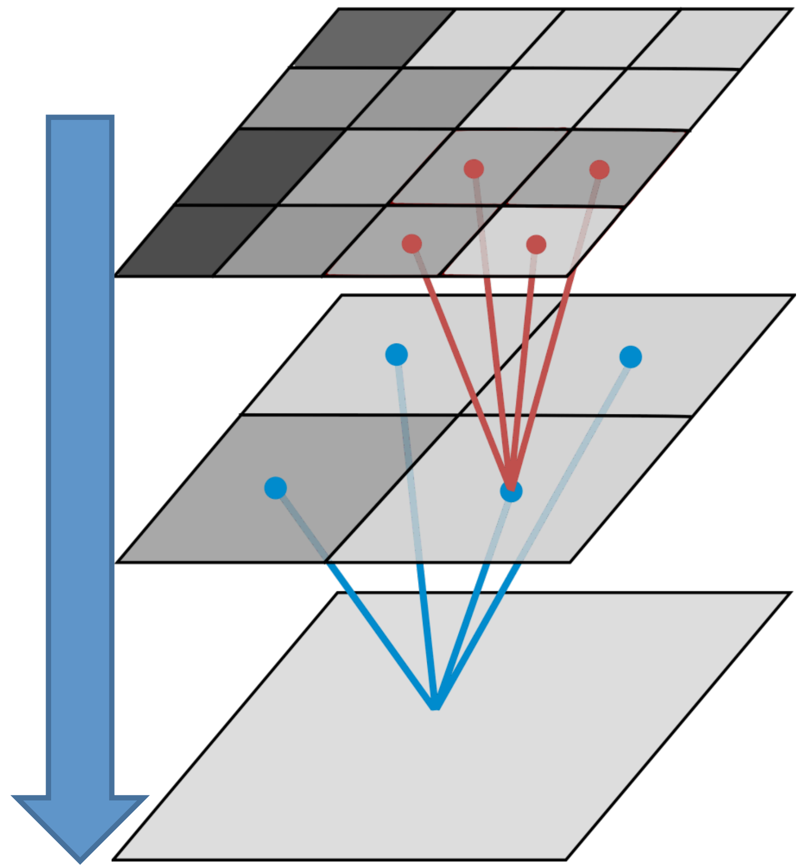
Thank you!

Occlusion Culling

- Render all occluders prior to rendering characters
- Determine which characters are occluded by the environment or structures
- Filter requires additional input: *Hi-Z map of occluders*

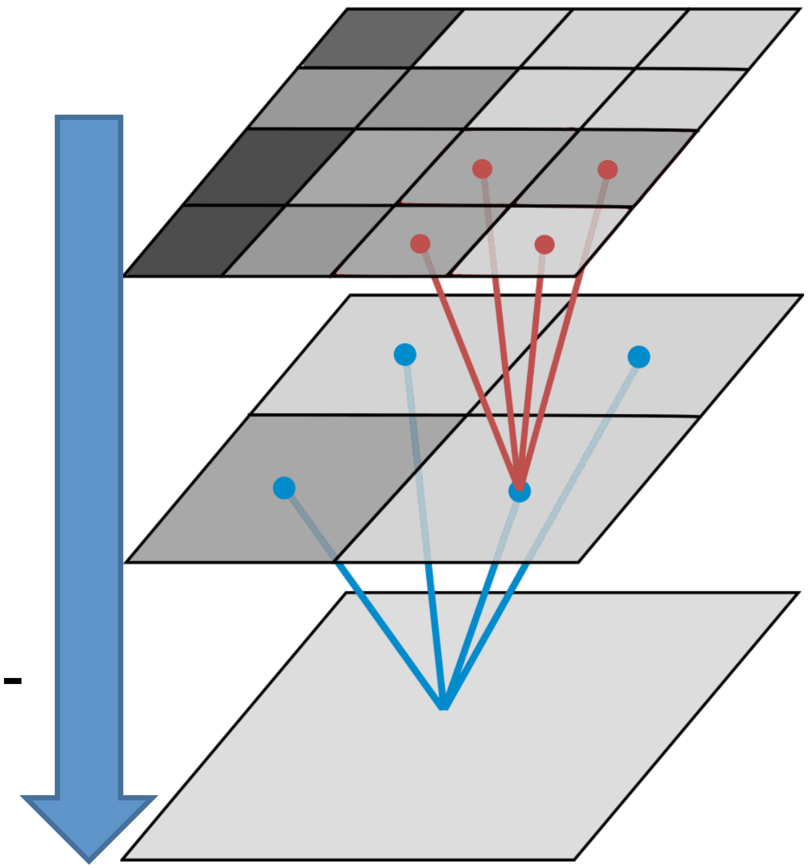
Hierarchical Depth Image

- Hi-Z Map Generation
  - Start with scene's Z buffer
    - Not a separate depth pass
  - Max of neighboring texels
    - Stored in MIP chain



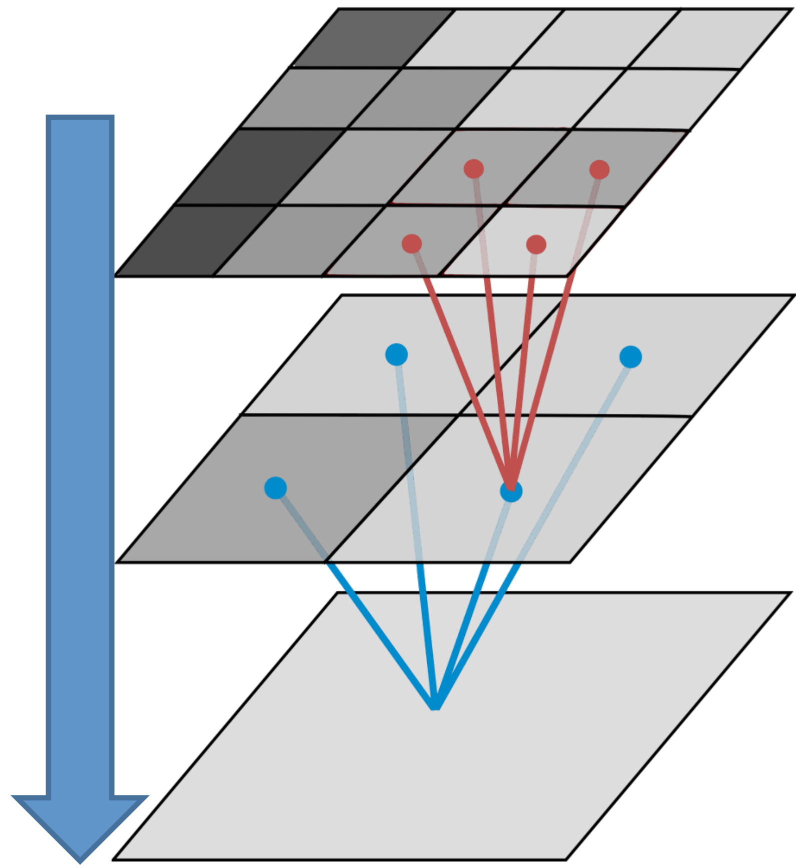
Hierarchical Depth Image

- Render into one MIP level while sampling the previous level
  - Rendering into smaller mip *reducing* the larger one
- Fetch 2×2 neighborhood and compute *max* value
- Fetch additional texels on the odd-sized boundary



Hierarchical Depth Image

- ***Indexing Gotcha!***
  - Careful with texel indexing
  - Use *Load()* with intearray indices



GS Filtering for LOD Selection

- Used a discrete LOD scheme
  - Each LOD is selected by character’s distance to camera
- Three successive filtering passes
  - Separate the characters into three disjoint sets
  - LOD parameters easily specified for each set

GS Filtering for LOD Selection

- Compute LOD selection *post* culling
  - Only process visible characters
  - Culling results are only computed once and re-used
- Render closest LOD using tessellation and displacement
- Conventional rendering for middle LOD
- Simplified geometry and shaders for furthest LOD