SIGGRAPH2008

Simulation and Rendering Massive Crowds of Intelligent and Detailed Creatures on GPU

Christopher Oat
Natalya Tatarchuk
AMD Graphics Products Group

Outline

- Motivation
- Crowd movement simulation on GPU
  - Global and local navigation and avoidance
- Lighting solution
- Crowd rendering and scene management on GPU
- Conclusions

Driving Visual Experience Frontiers

- Our task is to push the envelope for interactive visual experience
  - "What does it mean to have better interactive experiences?"
- Beyond just the pretty face
  - We know we can do that… We've done it!
  - Can we make them think?

AMD ATI RADEON PREMIUM GRAPHICS

SIGGRAPH2008

We Can Even Make Them Dream!

Motivation

- Making the gameplay more fun: A game is a series of choices...
  - It helps when choices are interesting!
- Artificial intelligence is ubiquitous in current games
  - Non-player and player characters require a form of AI
  - Path finding, obstacle avoidance, decisions, etc.

Motivation

- CPU can spend > 50% time on path finding computations alone in games
  - Limits possible decision complexity
  - Thus we frequently see “zombie-like” NPCs
  - Gameplay suffers
- Emergent behaviors improve any game with many characters
  - Making it more fun!

Top Level Overview

- *<u>Goal</u>:* Simulate and render massive crowds of characters on GPU at a scale that's both breathtaking and challenging
- Direct3D® 10.1
- Tessellation
- 4X MSAA
- HD resolution and HDR rendering

AMD ATI RADEON PREMIUM GRAPHICS

SIGGRAPH2008

A Smörgåsbord of Features

- Dynamic pathfinding AI computations on GPU
- Massive crowd rendering with LOD management
- Tessellation for high quality close-ups and stable performance
- HDR lighting and post-processing effects with gamma-correct rendering
- Terrain system
- Cascade shadows for large-range environments
- Advanced global illumination system

Introducing the Froblins

Outline

- Motivation
- Crowd movement simulation on GPU
  – Global and local navigation and avoidance
- Lighting solution
- Crowd rendering and scene management on GPU
- Conclusions

Motivation: Beware of Ghosts!!

GRAPHICS

Dynamic and Engaging World Through AI

- Combine global path finding with local avoidance and individual decisions
- Crowd movement is more stable and realistic solved on a global scale
  - Crowd dynamics similar to fluids
  - Agents “flow” towards the closest goal, along the path of least resistance
  - Froblins follow correct paths and do not “get stuck”

AMD ATI RADEON PREMIUM GRAPHICS

SIGGRAPH2008

Global Path Finding

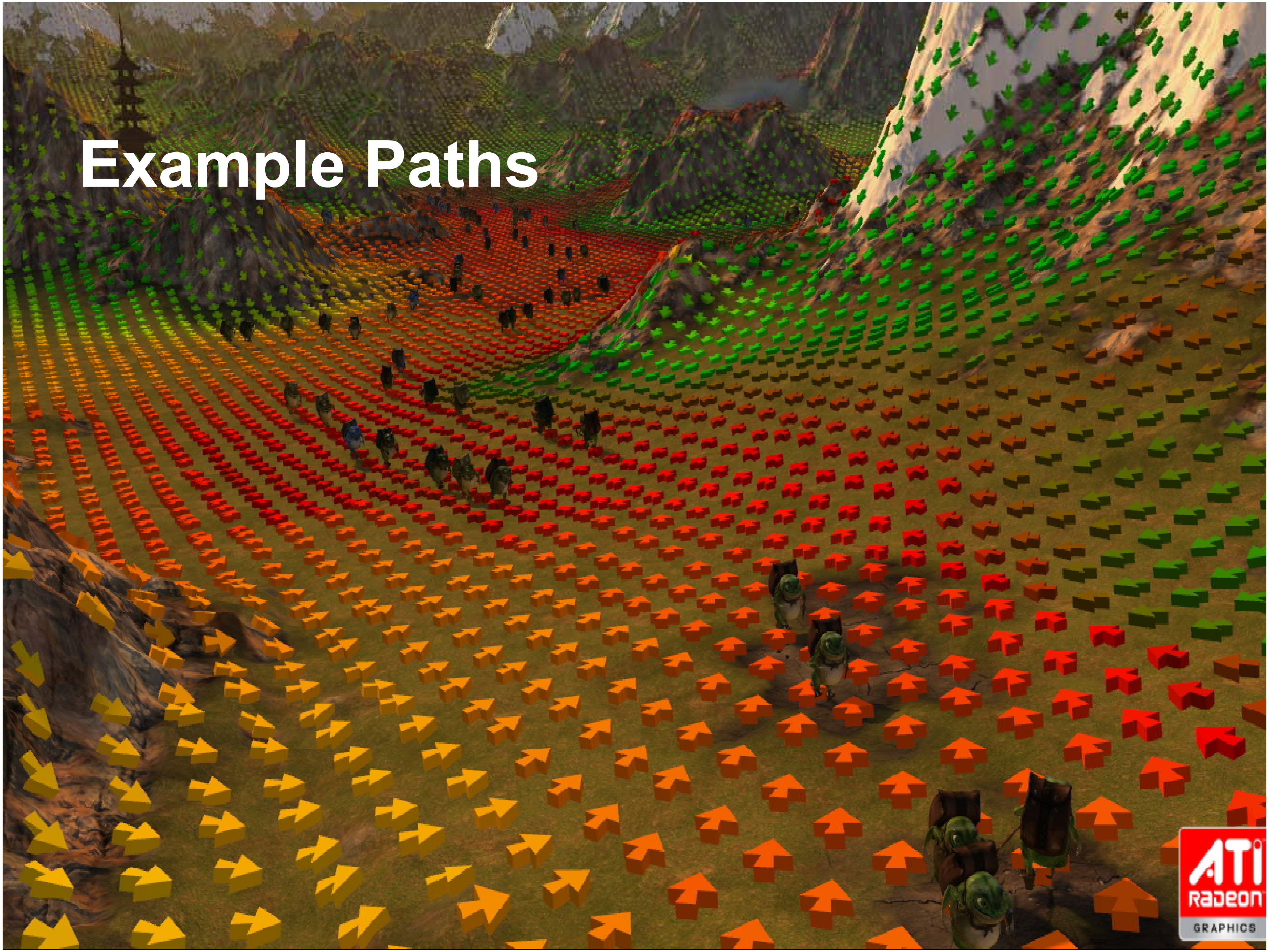
- Based on *Continuum Crowds* [TCP06]
- Convert motion planning into an optimization problem
  - Use continuum model
  - Computational algorithms from optics
- Smooth flow-like crowd movement
  - Congestion avoidance and emergent behavior

Pathfinding on GPU

- Numerically solve a 2nd order PDE on GPU with a computational iterative approach (eikonal solver)
  - Represent environment as a cost field
  - Through discretization of the eikonal equation
- Applicable to many general algorithms and areas

SIGGRAPH2008

Example Paths



Solver: From Cost to Travel Time

- Solve for travel time as a function of potential
- *Potential φ* = integrated cost *F* along shortest path to goal
  - Follow negative gradient

$$\left\| \nabla \varphi(x) \right\| = F$$

- Set potential = 0 at goal, eikonal equation elsewhere
- Comes from optimization algorithms

AMD ATI RADEON PREMIUM GRAPHICS

SIGGRAPH2008

Eikonal Solver via Fast Marching Method

- Emulates Dijkstra's shortest path algorithm
- A finite difference approximation to the continuous eikonal equation
  - Start from known potential ($\varphi=0$ at goal) and propagate to neighbors until convergence
  - Compute potential for neighbor cells based on known cells
- Serial algorithm; requires an ordered data structure

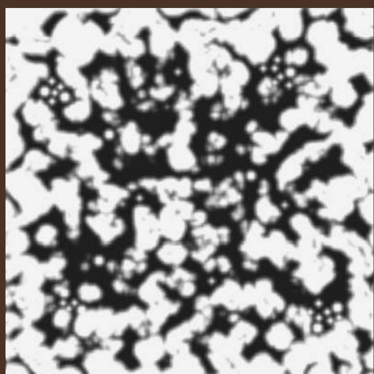
Parallel Eikonal Solver

- Using *Fast Iterative Method* [JW07]
- Use upwind finite difference approximation but no ordered data structures
- We use small grid resolutions (128^2 -256^2)
- Solve for four goals at once
  - Taking advantage of vectorization – use RGBA FP16 texture
- Convergence determined empirically

Environment As Cost Function

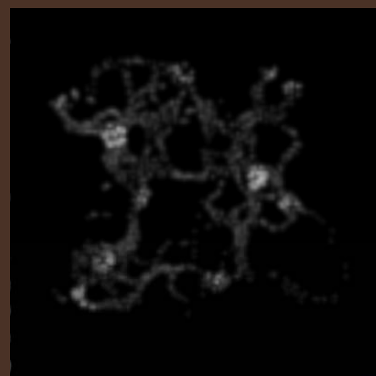
- Continuum Model models environment as *a positive cost function*
  - Can't be zero otherwise agents would move at infinite speed; add some small base amount
- Incorporate information about terrain and obstacles (static and dynamic)
  - Splat dynamic obstacles, including agents into the cost function

Cost Function Formulation



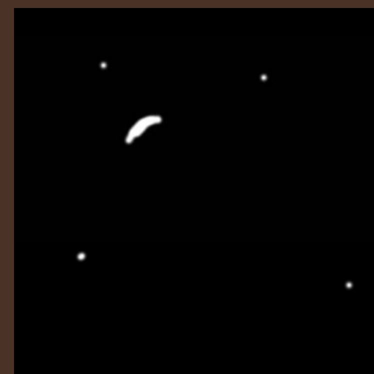
Static Cost

+



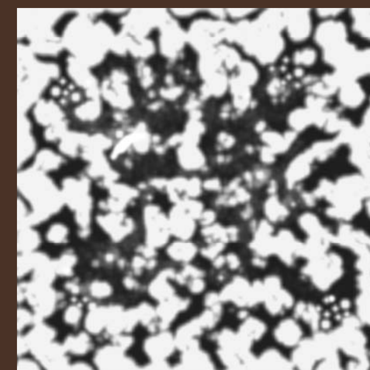
Agent Density

+

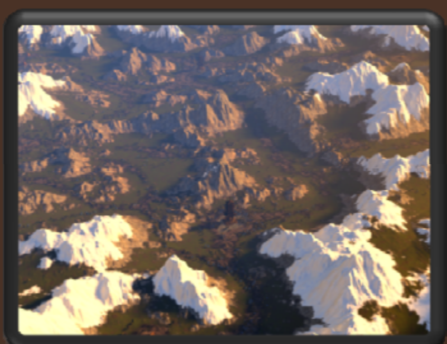


Hazards

=



Total Cost



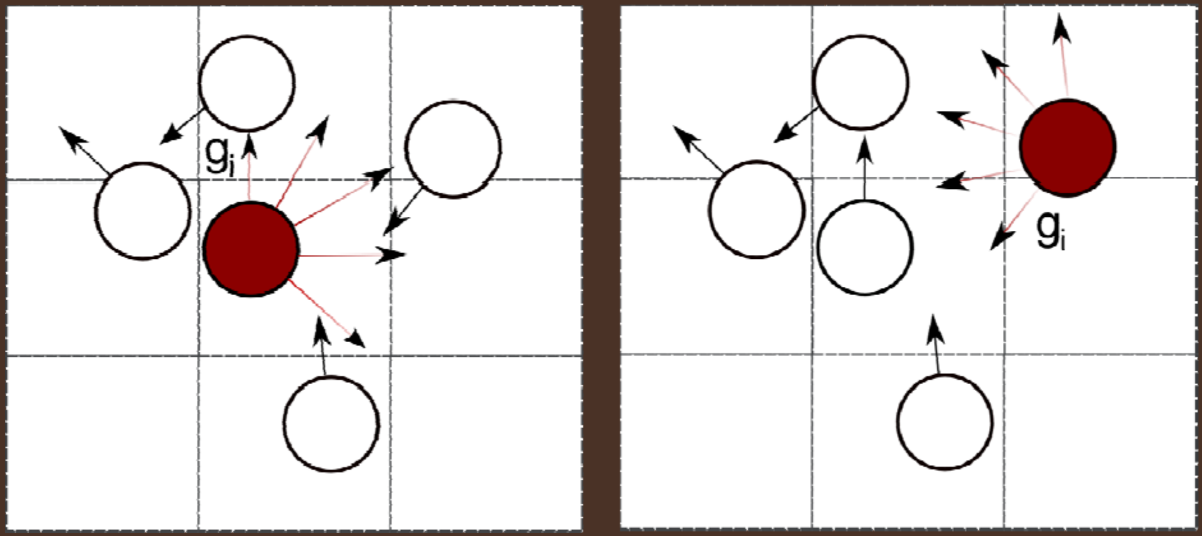
SIGGRAPH2008

Local Navigation and Avoidance

- Global model not great for local obstacles
  - Performance & No need to avoid agent far away
- Update velocity based on positions and velocities of nearby agents/obstacles
  - Based on Velocity Obstacle formulation [Fiorini and Shiller 1998]
  - See course notes for more details & Jeremy Shopf's talk on in ***Beyond Programmable Shading: In Action*** course

Direction Determination

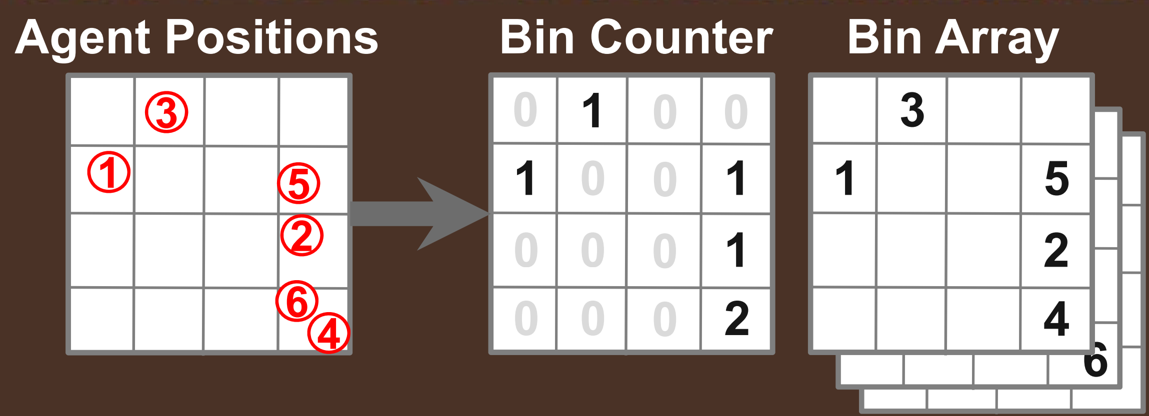
- Evaluate a fixed set of possible directions relative to global navigation direction



Spatial Queries

- Need to query nearby agent positions and velocities
- Spatial data structure
- Agents “binned” by position:
  - Bin Counter : Color buffer
  - Bin Array : Depth Buffer Array

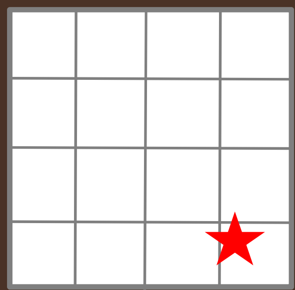
Spatial Data Structure: Bins



- World space position mapped to 2D index
- Bin Counter: color buffer, tracks bin loads
- ID Array: depth array, binned agent IDs

Bin Queries

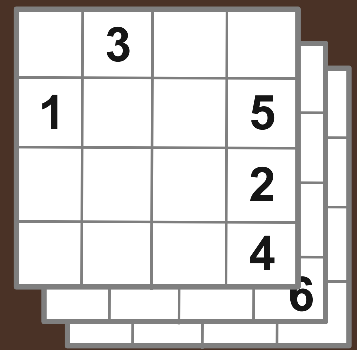
Agent Positions



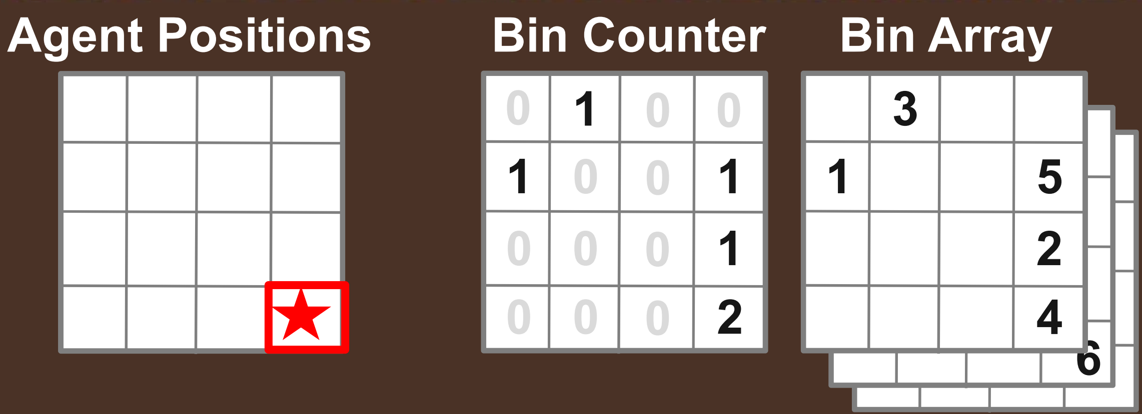
Bin Counter

0	1	0	0
1	0	0	1
0	0	0	1
0	0	0	2

Bin Array

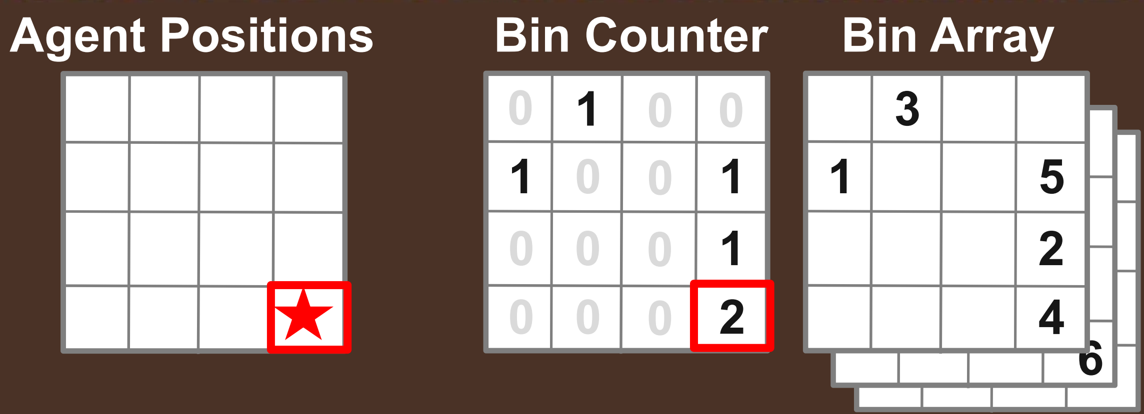


Bin Queries



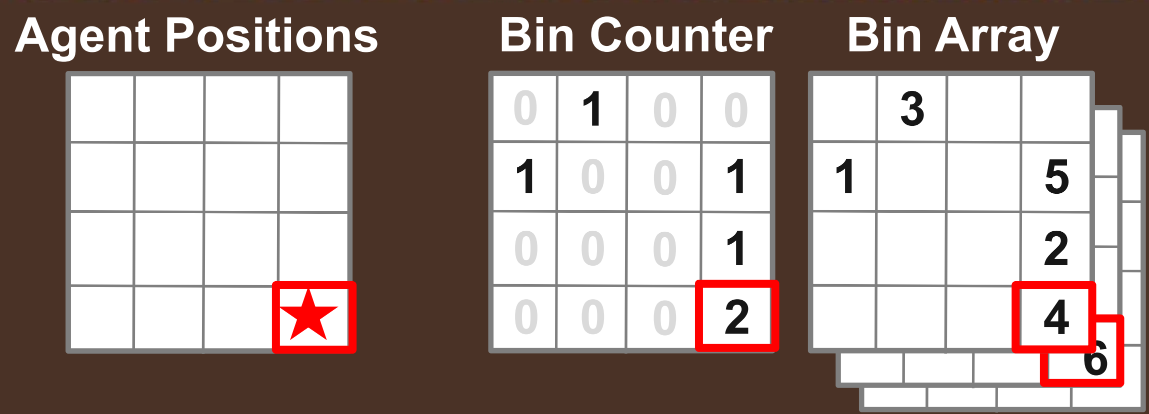
- Translate position to 2D index

Bin Queries



- Translate position to 2D index
- Fetch load from *Bin Counter* (color buffer)

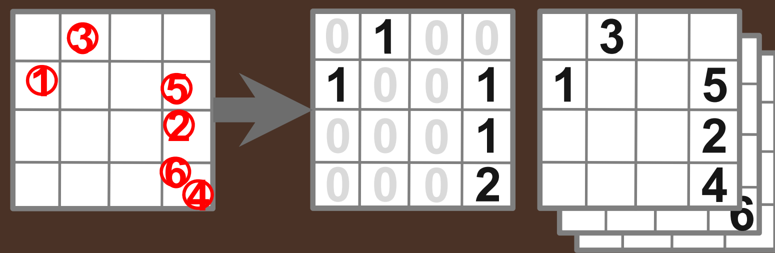
Bin Queries



- Translate position to 2D index
- Fetch load from *Bin Counter* (color buffer)
- Fetch agent IDs from *Bin Array* (depth array)
  – Efficient: *known number* of *sorted* agents

SIGGRAPH2008

Data Structure Updates: Overview



- Binning is a multi-pass algorithm
  - Update one slice of *Bin Array* per pass
  - Once binned, agents removed from working set
  - Algorithm repeats until working set is empty
  - Overflow is possible & detectable

Data Structure Updates: Initialization

- Initialize data structure
  - Clear bin counter (color buffer) to 0
  - Clear bin array (depth array) to **`MAX_DEPTH`**
  - Working set is array of ***all*** agent IDs
    - No VB necessary

Data Structure Updates: Pass 1

- Bind bin counter & first slice of bin array
- Draw working set as point primitives
- Vertex shader:
  - Map agent position to 2D bin array index
  - Set point's depth to normalized agent ID
- Pixel shader: output "1"
- Depth test: **LESS_THAN**

Data Structure Updates: Pass 2…n

- Next slice of bin array bound as depth buffer
- VS: Sample ID from previous slice of bin array
  - Reject points less than or equal to previous
- GS: stream out non-rejected points
- PS: write pass number
- Depth test: **LESS_THAN**

Results of Pass 2..n

- VS test ensures only points that haven't yet been binned get streamed-out and rasterized
- Depth test ensures the point with lowest ID gets binned
- Results in points binned in sorted order
  - Like depth peeling
- Stream-out buffer becomes new working set

Early Termination

- We want to halt the algorithm once all points are binned
  - Do not query size of stream out buffer each pass
  - CPU/GPU synchronization results in pipeline stalls
- Would like to hand the whole thing off to the GPU to control execution

Avoiding Synchronization Stalls

- We know max number of iterations
  - Number of bin array slices
- Make all the draw calls for the max number of iterations
  - Use cascading predicated draw calls
  - Use along with "`DrawAuto`" to issue draws
  - Predicate on number of stream-out elements

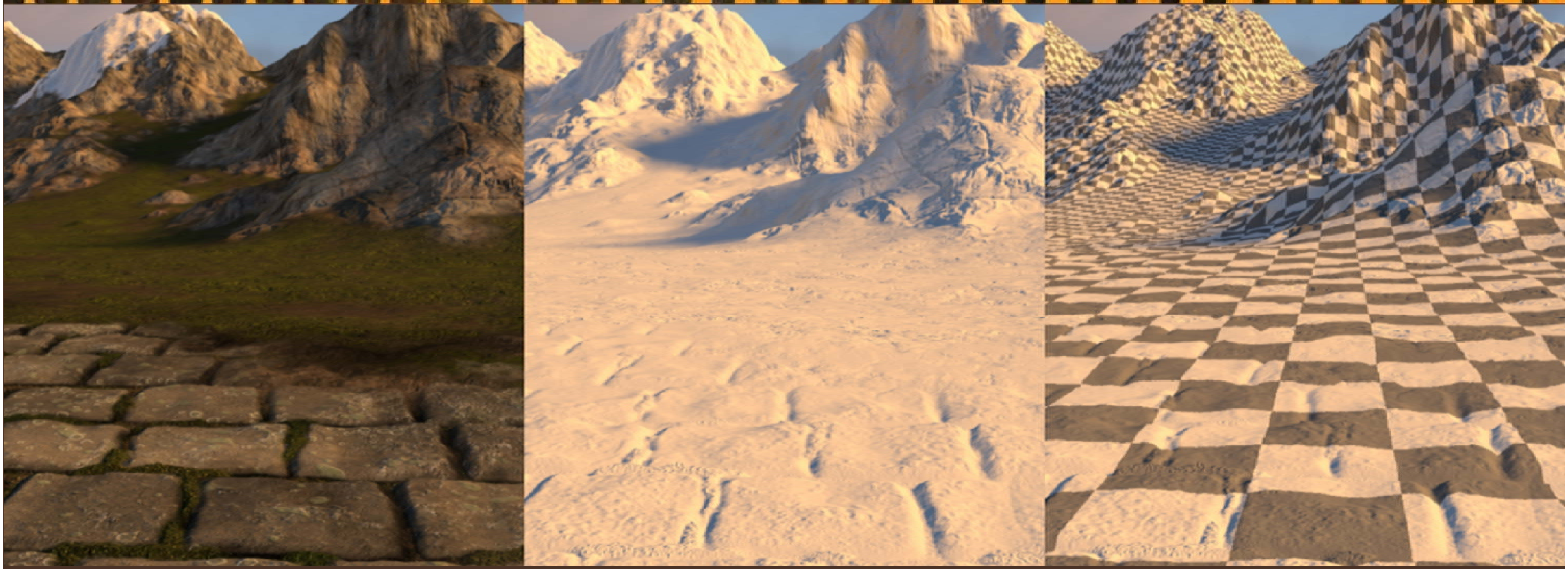
Avoiding Synchronization Stalls

- GPU terminates algorithm once all agents are binned
  - Once working set (stream out buffer) goes to 0
  - Remaining draw calls are skipped
- The algorithm either terminates or overflows

Binning on the GPU

- GPU Binning
  - Efficient queries
    - Early-out on empty bins, known bin load, sorted order
  - Overflow detection
  - Stream-out reduction of working set
  - GPU termination control
- Many applications beyond local avoidance

Spherical Harmonic Light Map



Diffuse lighting with Shadow Map

- Extracted dominant directional light: L_d
  - Dominant directional light color: L_c
- Left over residual lighting environment: L_e
- Dynamic shadow term: V_s
- Surface normal: N

$$\max(N \cdot L_d, 0) * L_c * V_s + \mathrm{SH\_Eval}(N, L_e)$$

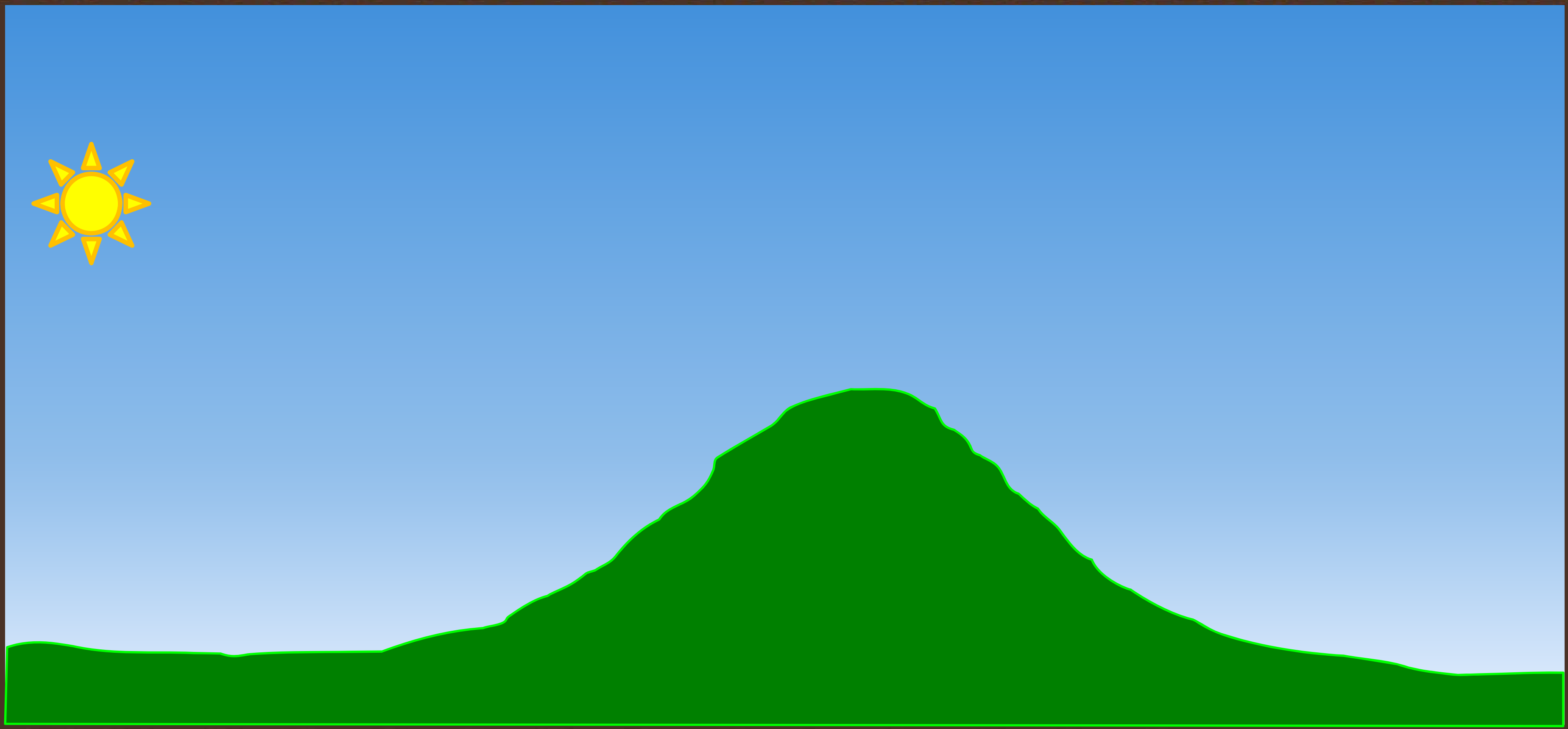
Dynamic Shadows Cast on Terrain

- Dynamic characters cast shadows on terrain
  - Shadow map shadows mingle with SHLM shadows
  - Shadow map attenuates *dominant* lighting term

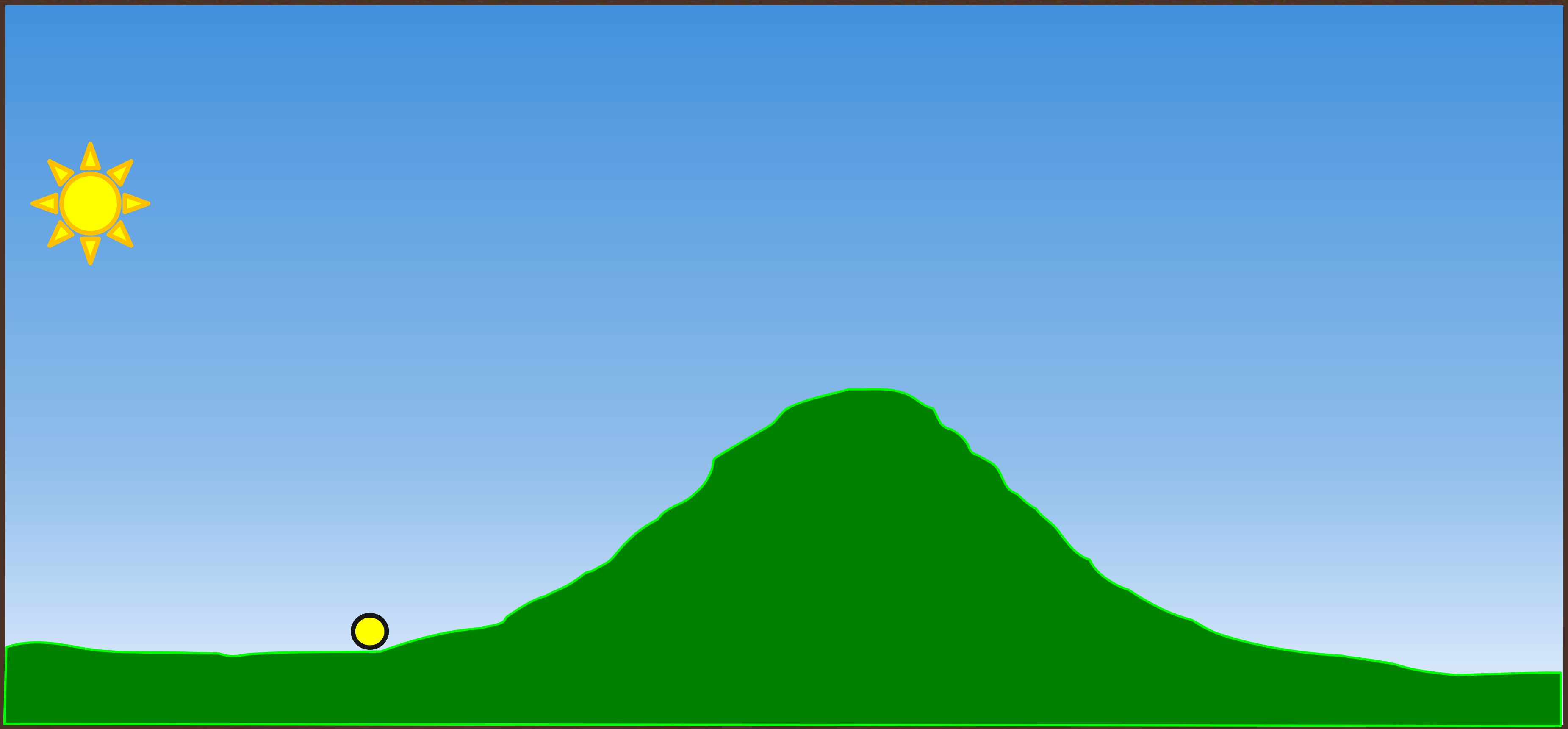
Double Shadows

- Results in double shadows…
  - Characters in shadow still cast shadows

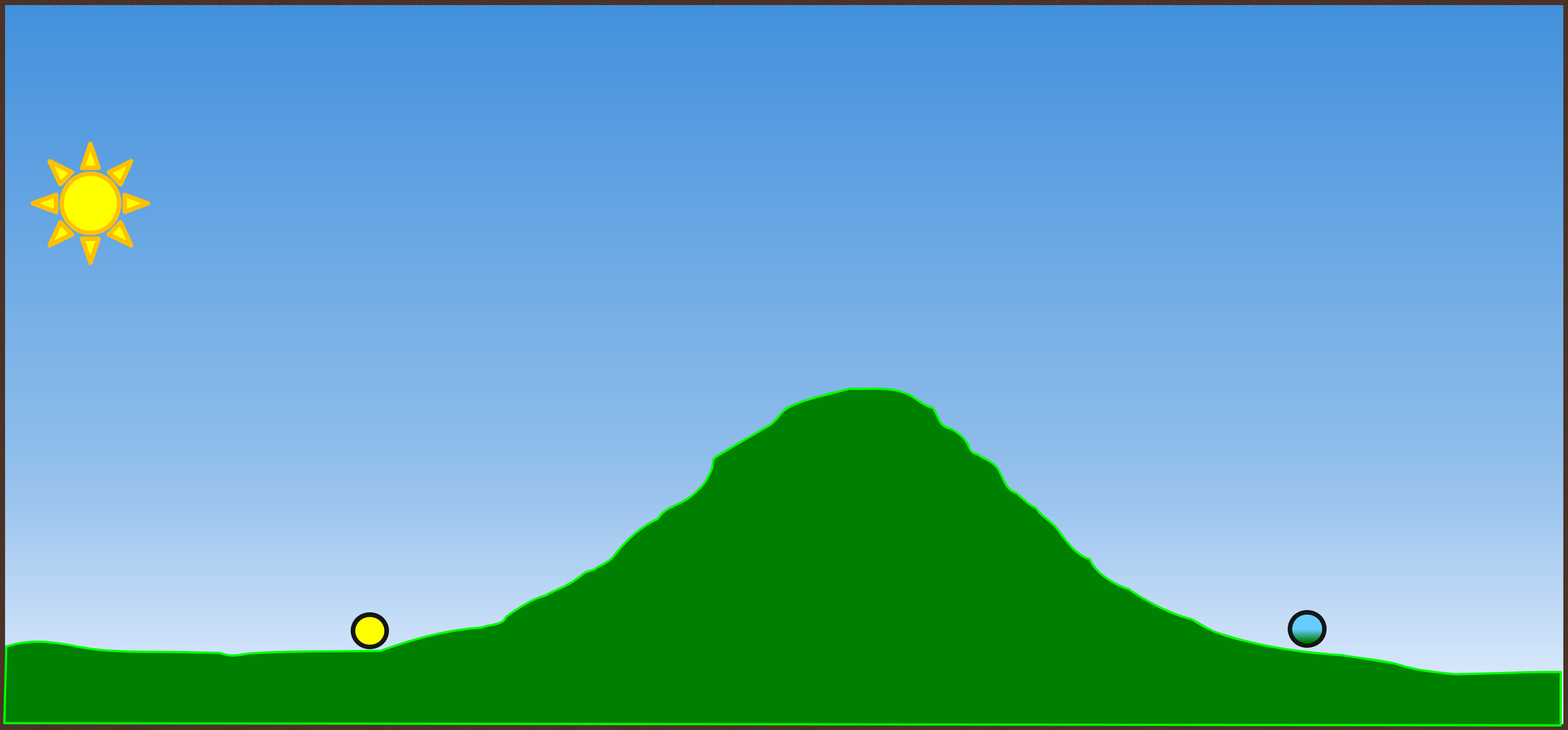
Detecting Direct Sun Light



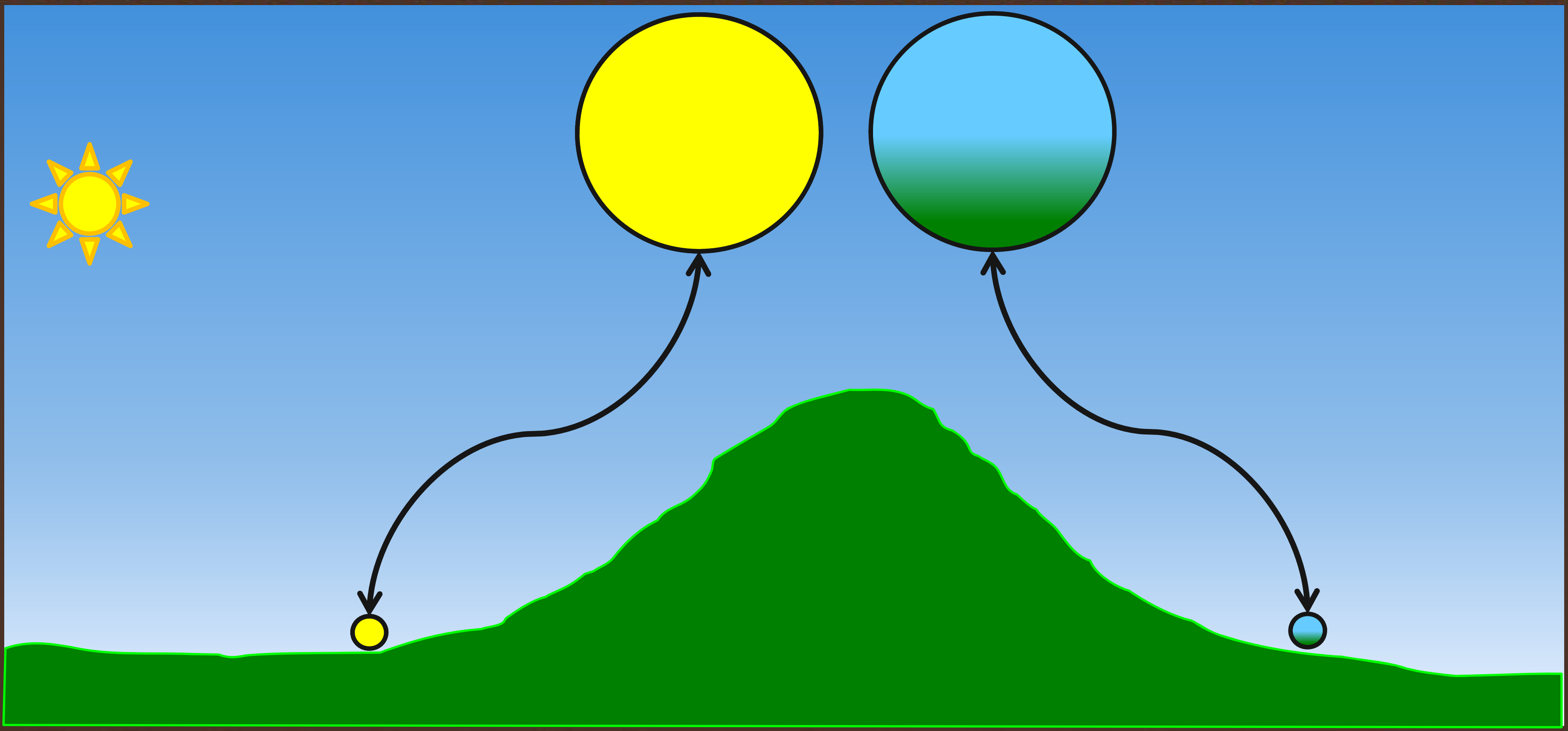
Detecting Direct Sun Light



Detecting Direct Sun Light



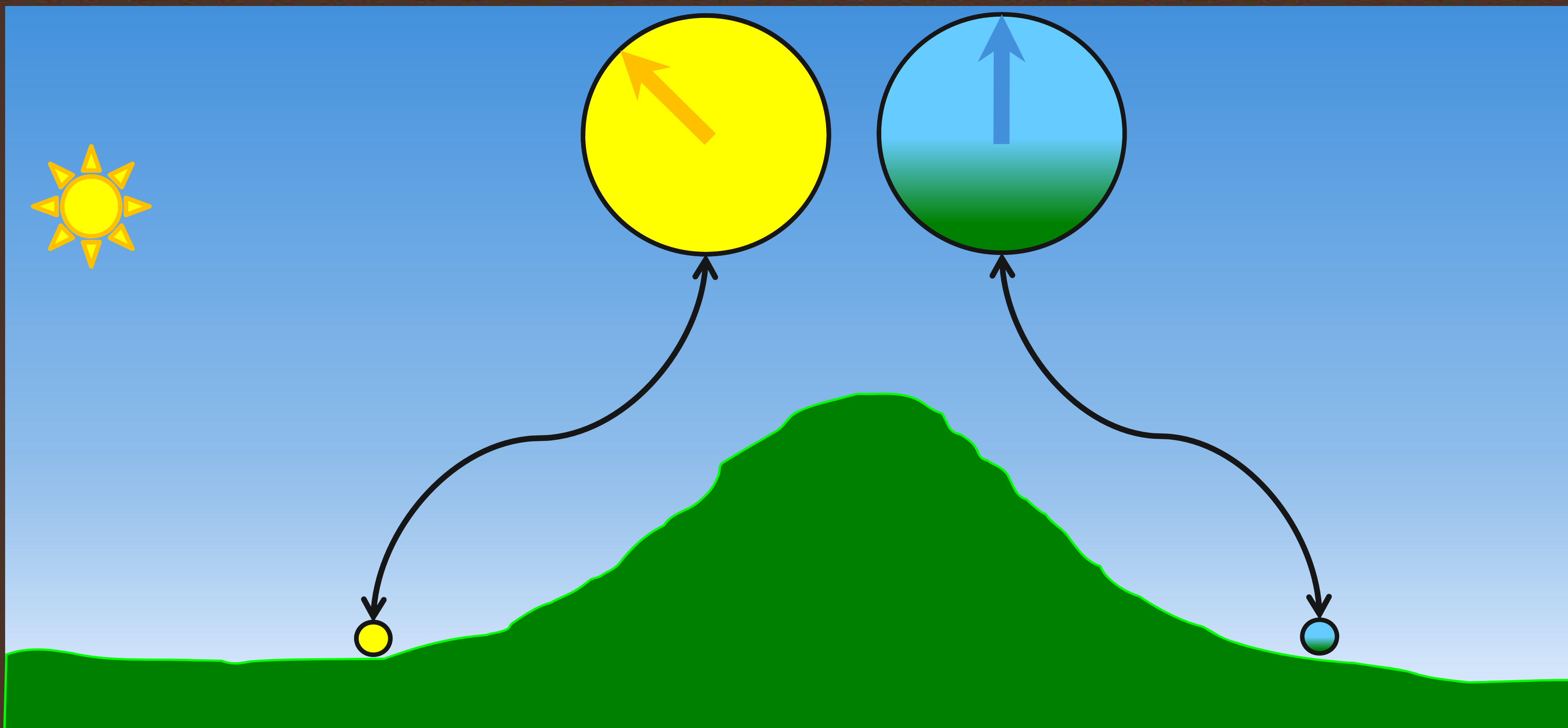
Detecting Direct Sun Light



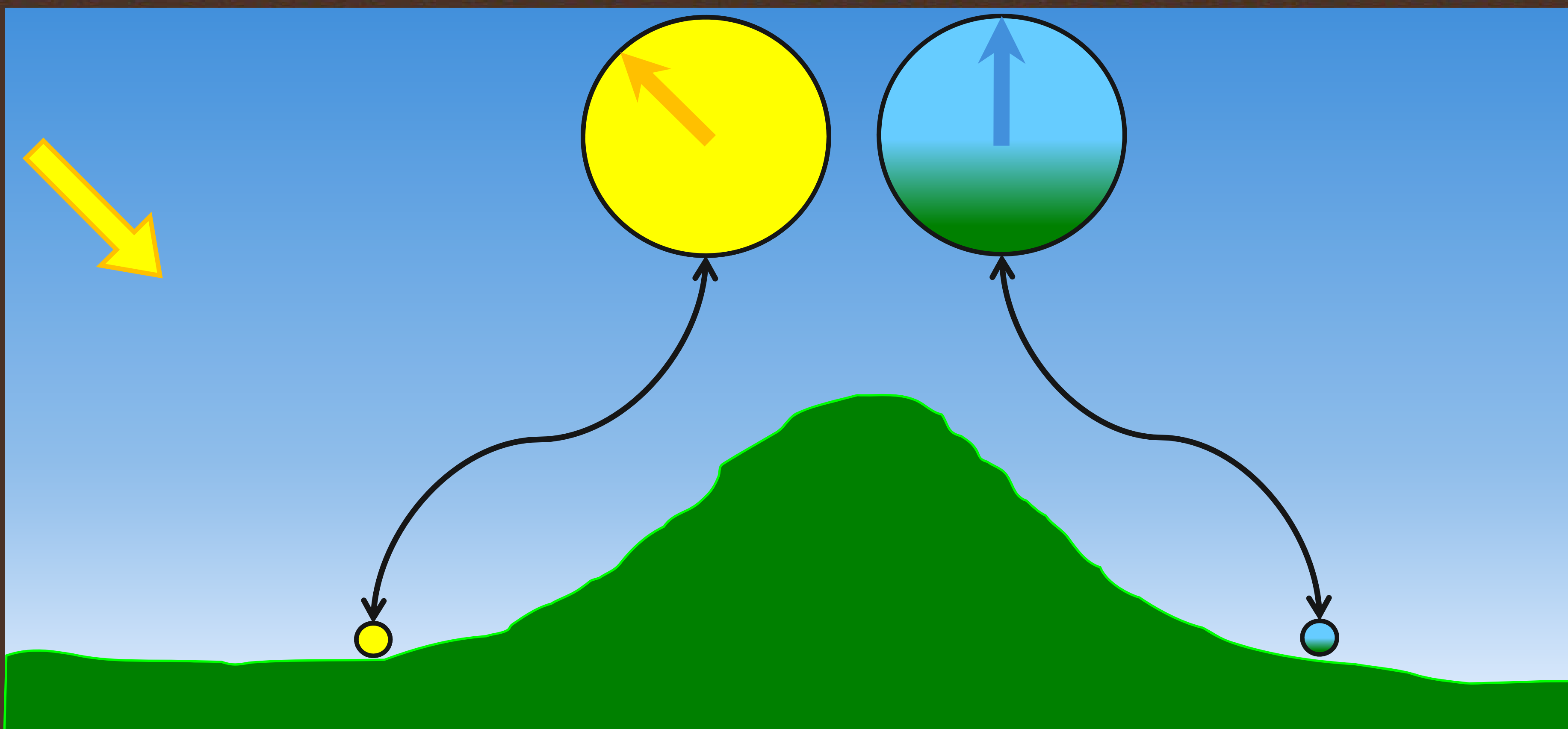
All slides © 2008 Advanced Micro Devices, Inc. Used with permission.

AMD ATI RADEON PREMIUM GRAPHICS SIGGRAPH2008

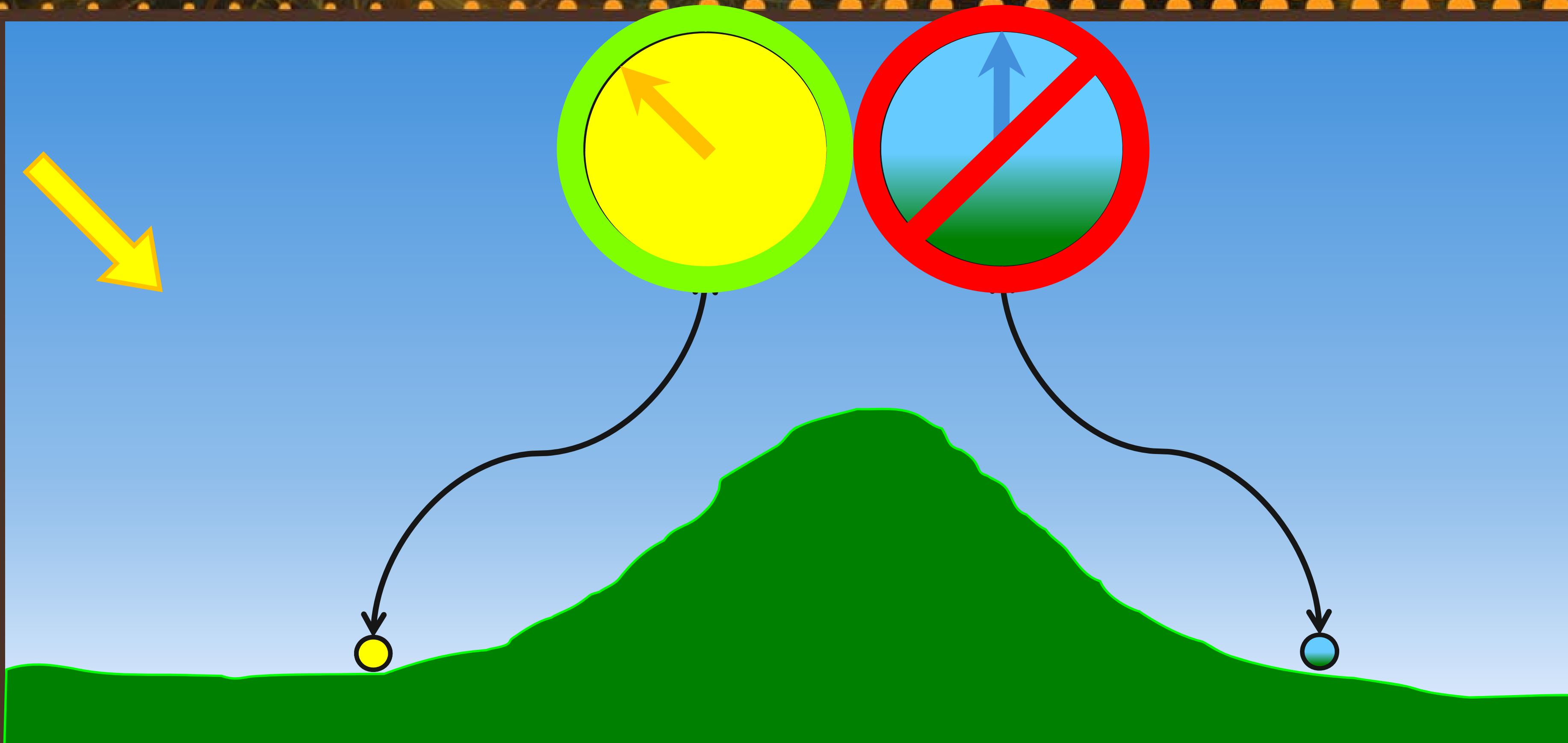
Detecting Direct Sun Light



Detecting Direct Sun Light



Detecting Direct Sun Light



Double Shadow Fix

- Shadow map only used on non-occluded regions of terrain

Outline

- Motivation
- Crowd movement simulation on GPU
  – Global and local navigation and avoidance
- Lighting solution
- Crowd rendering and scene management on GPU
- Conclusions

AMD ATI RADEON PREMIUM GRAPHICS

SIGGRAPH2008

The Need for GPU Character Management

- Need scalability and stable performance
- Don’t want to render thousands million-poly 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

GPU Scene Management

GPU Scene Management

- We use Direct3D® 10.1 features for GPU scene management and level of detail management
- Render froblins as an army of instanced characters
- 3 discrete LODs
- Use tessellation to get maximum amount of details with stable performance for close-ups

AMD ATI RADEON PREMIUM GRAPHICS SIGGRAPH2008

All slides © 2008 Advanced Micro Devices, Inc. Used with permission.

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
  - Use *DrawAuto* for multiple tests or combine into a single GS invocation for efficiency

GPU Character Management Overview

1. Render the occluder geometry
2. Construct the Hi-Z map
3. Run all characters through the view frustum culling filter
4. Results are run through the occlusion culling
5. Run through a series of LOD selection filters
6. Render each LOD

View Frustum Culling

- Using the filtering GS perform view frustum culling
  - Using standard methods
- VS checks for intersection between character bounding volume and the view frustum
  - Regular methods apply
- If the test passes, the character is visible: emit it
  - Otherwise it's culled

Occlusion Culling

- Render all occluders prior to rendering characters
- Determine which characters are occluded by the environment or structures
  - Use resulting depth buffer
  - Cull against arbitrary, dynamic occluders
- Novel formulation: *Hi-Z map*

Hi-Z Map

- A hierarchical depth image
  - Uses the Z buffer information
- A mip-mapped, *screen-resolution* image
  - Each texel at level i = *max* (all texels at level i - 1)
- Does not require a separate depth pass

Occlusion Culling with Hi-Z Map

- Use constructed Hi-Z map
- Examine the depth information for pixels covered by the object's projected bounding sphere
  - Compare the max fetch depth to the projected depth of the point on the sphere closest to the camera
- Conservative culling
  - Does not result in false culling
- Details in the course notes

Hi-Z Map Construction

- Finest level – use the main depth buffer
  - DepthStencil view (with MS DB in Direct3D® 10.1)
- Generate the mip chain levels with *reduction* passes
  - Mind the dimensions – screen-space images don't mip well!
  - Use integer operations to avoid incorrect indexing

AMD ATI RADEON PREMIUM GRAPHICS

Reduction Pass for Hi-Z Levels

- 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

AMD ATI RADEON PREMIUM GRAPHICS

SIGGRAPH2008

GS Filtering for LOD Selection

- Use 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

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

CPU Animation Sampling

- Traditionally matrix palettes are computed on the CPU per character
  - Loaded into constant store
- Limitations on the number of characters handled in one draw-call with this approach
  - Large crowds of characters require numerous draw calls
- GPU character management makes this tricky

Froblin Character Animation

- Agents have a set number of actions with associated animation sequence

GPU Animation Sampling and Control

- At preprocessing time, bake animation data into texture arrays
  - Slice number = animation sequence
  - X and Y = key frame and bone index
- At rendering time, determine per-instance animation sequence index and time offset
  - During simulation in our case

GPU Animation Sampling and Control

- Sample and interpolate animation data when rendering the character
  - Sample the animation textures

Character Animation on GPU

- At preprocessing, flatten the transform hierarchy
- Compute bone transforms for each key frame transforming that bone into object space
- During simulation, assign an animation sequence to each character
  - Including a time offset into that sequence

AMD ATI RADEON PREMIUM GRAPHICS

SIGGRAPH2008

Animation Texture Layout

		Key 0	Key 1	…	→	…	Key N
Bone 1	Matrix Row 0						
	Matrix Row 1						
	Matrix Row 2						
Bone 2	Matrix Row 0						
	Matrix Row 1						
	Matrix Row 2						
	↓						

Character Animation on GPU

- During rendering VS fetches, interpolates, and blends the key frames for each bone
  - Using per-character information
- Each instanced character performs its skinning in object space
- Transforms the result using its position and orientation

Tessellation and Crowd Rendering

- Combine with Direct3D® 10 instancing support
- Render using interpolative planar tessellation
  - Fast evaluation
  - With displacement mapping for fine-scale detail
- Control tessellation level per-draw call
  - Use character location
- The same art assets as conventional rendering

The Benefits of Tessellation

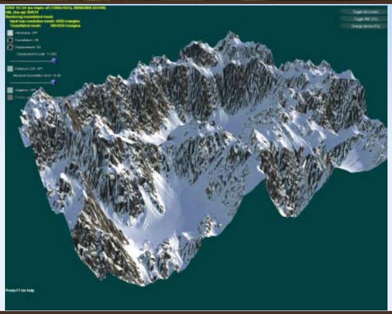
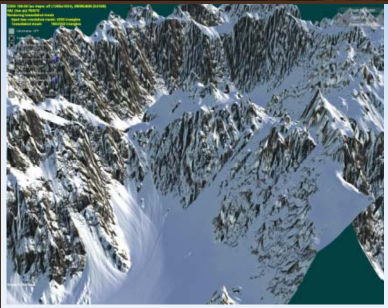
- Tessellation reduces memory footprint and bandwidth
  - Only store low resolution mesh
  - Always relevant, especially for consoles

	Polygons	Total Memory
Low resolution Froblin model	5160 polygons	VB/IB: 100K 2K x 2K 16 bit displacement map: 10MB
ZBrush High res Froblin model	>15M polygons	~270MB VB and 180MB IB storage

The Benefits of Tessellation

- Scalability
- Stable and predictable performance

Tessellation Performance Analysis

Rendering Mode	Num faces:	 Far away view		 Close-up view	
$N_T = 411 \times N_L$		ATI Radeon™ HD 4870	ATI Radeon™ HD 2900 XT	ATI Radeon™ HD 4870	ATI Radeon™ HD 2900 XT
Original **low res mesh (N_L)**	4,050 triangles	852 fps	359 fps	850 fps	275
Continuously tessellated mesh (N_T)	1.6 M triangles	232 fps	143 fps	213 fps	101
Adaptively tessellated mesh N_A	Dynamic, 1.6M > N_A > 4K triangles	845 fps	356 fps	574 fps	207

AMD ATI RADEON PREMIUM GRAPHICS

*Configuration: AMD reference platform with AMD Athlon™ 64 X2 Dual-Core Processor 4600+, 2.40GHz, 2GB RAM. Motherboard: ASUSTek M2R32-MVP. Memory: DDR2-800 400 MHz. Operating System: Windows Vista® SP1."

SIGGRAPH2008

Tessellation Pre Direct3D® 11

- GPU tessellation already available
  - Supported on Xbox™ 360
  - ATI Radeon™ HD 2000 Series and beyond: supported on all models
    - *Even on the integrated chipsets!*
- Subset of Direct3D® 11 tessellation features
- Contact AMD ATI developer relations for details on accessing tessellation

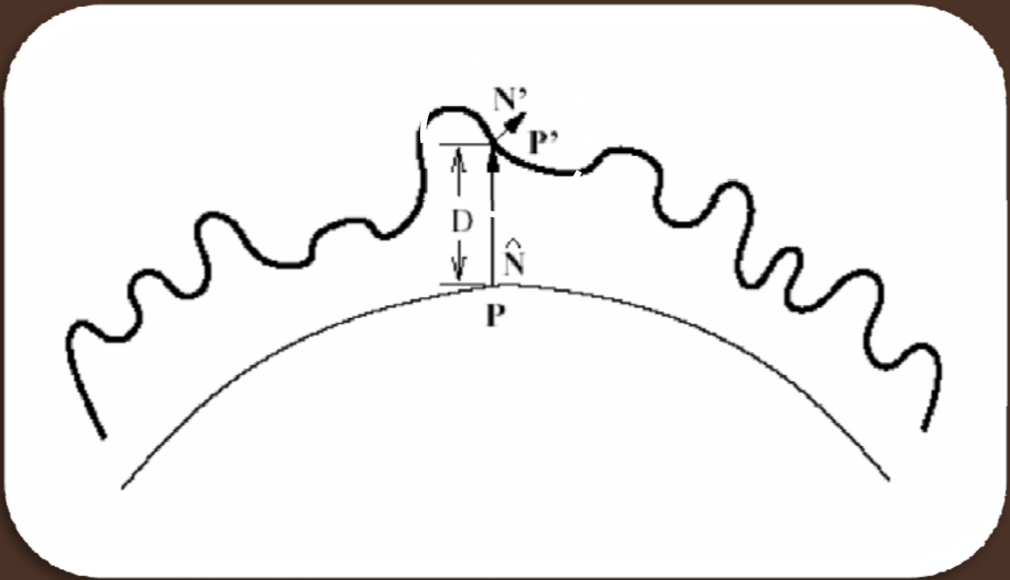
Preview and Prepare for Direct3D® 11

- Support across most APIs gives you best bang for the buck for this feature
  - Support on Microsoft® Windows® XP as well as Microsoft® Windows® Vista
  - PC versions can use Xbox™ 360 native features
  - Reach more players
- Developing artwork takes time
  - Integrating tessellation early gives you and your artists time to design and polish

Tessellation Process

Combining Normal Maps and Displacement Maps

- As we displace, we change the actual displayed normal



SIGGRAPH2008

Lighting with Displacement

- Use TS normal maps *even with displacement*
- Generate displacement and normal maps using the same tangent space
  - The same TS computations for preprocessing
  - The normal encoded in TS normal map will match the normal from displacement

Regularly drawn Froblin

No tessellation used

High Detail Froblin with Tessellation and Displacement Mapping

Froblin Close-up:

No tessellation

SIGGRAPH2008

Froblin With Tessellation and Displacement Mapping Close-up

SIGGRAPH2008

Tessellated Characters LOD Control

- Dynamic number of detailed characters in view
  - Simulation changes interactively, no a priori control
- Need stable frame rate in dense crowd simulations
- Compute tessellation level as a function of tessellated characters in view
  - Avoid polygonal count explosion

Tessellation Level Computation

$$T_l = \text{clamp}(\ M\ T_{max} / N, 1, T_{max})$$

- *N* characters rendered with tessellation
- Bound the number of amplified triangles
- Primitive count never exceeds than the cost of *M* fully tessellated characters

Control Cage Pre-Pass

- Perform animation and transformation on control cage in a *pre-pass*
  - Allows for more complex per-vertex operations
- Combine with vertex (de)compression for reducing bandwidth

Pass 1: Control cage animation and transforms

↓

Stream Out Buffer

↓

Pass 2: Tessellate the control cage

AMD ATI RADEON PREMIUM GRAPHICS

SIGGRAPH2008

PrePass and Vertex Compression

- Shader-based vertex compression / decompression
- Reduce stream out memory footprint between the pre-pass and tessellated pass
- Reduce fetch bandwidth for tessellated pass

PrePass and Vertex Compression

- Pack transformed vertex positions into 128 bit format
  - Lets us load each vertex with one fetch
  - Gives as much as 30% speedup
  - More details in the course notes

Froblin Land: Terrain Rendering

Froblin Land: Terrain Rendering

- Smooth, crack-free LOD without degenerates
- Tessellation and instancing
- Leverages Direct3D® 10.1 functionality to help minimize memory footprint
- Complex material system

AMD ATI RADEON PREMIUM GRAPHICS

SIGGRAPH2008

Froblins Performance Details*

- Staggering polygon count at interactive rates (>20fps)
  – From 900 polygons -> 6K -> 1.6M at the closest tessellated level of details
  – Up to 18M triangles per frame at fast interactive rates
  – 6M-8M triangles on average at 20-25 fps
- Full high quality lighting and shadowing solution
  – Rendering all objects into multiple shadow maps more than doubles polygon count per frame
- Rendering and simulating high quality detailed 3K froblins at 21 fps on average

*Configuration: AMD reference platform with AMD Athlon™ 64 X2 Dual-Core Processor 4600+, 2.40GHz, 2GB RAM. GPU: ATI Radeon™ HD 4870 Grahpics. Motherboard: ASUSTek M2R32-MVP. Memory: DDR2-800 400 MHz. Operating System: Windows Vista® SP1."

Simulation Performance Details*

- All modes render with 4X MSAA HDR and post-processing
- Simulate behavior and render > 65K agents at 30 fps
- AI simulation for 65K agents alone – 45 fps
- Rendering and simulating 65K agents – 31 fps
- Rendering 9,800,000 polygons each frame
- AI simulation executes with high efficiency resulting in ~1 *tera*flops!

*Configuration: AMD reference platform with AMD Athlon™ 64 X2 Dual-Core Processor 4600+, 2.40GHz, 2GB RAM. GPU: ATI Radeon™ HD 4870 Grahpics. Motherboard: ASUSTek M2R32-MVP. Memory: DDR2-800 400 MHz. Operating System: Windows Vista® SP1."

Conclusions

- Practical and efficient simulation and rendering of large crowds of characters on GPU
- New GPU algorithms for
  - Global and local navigation and spatial data structures
  - Scene management
  - etc
- And a number of other advanced techniques!

Learn More about Our Approaches

- All shaders are included in the course notes
- Course notes are available for download:
  - ACM Digital Library: http://portal.acm.org/dl.cfm
  - AMD Graphics Technical Publications webpage: http://ati.amd.com/developer/techreports.html

Acknowledgements

Game Computing Applications Group

Jeremy Shopf

Josh Barczak

Abe Wiley

and

Chaingun Studios

Exigent

Allegorithmic

SIGGRAPH2008

Questions?

References

- [TCP06] TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum Crowds. *ACM Trans. Graph.* 25, 3 (Jul. 2006), pp. 1160-1168, Boston, MA.
- [JEONGWHITAKER07B] JEONG, W.-K, AND WHITAKER, R.T. 2007. A Fast Iterative Method for a Class of Hamilton-Jacobi Equations on Parallel Systems. University of Utah School of Computing Technical Report UUCS-07-010.
- [TATARCHUK08] TATARCHUK, N. 2008. Advanced Topics in GPU Tessellation: Algorithms and Lessons Learned. Presentation. Gamefest 2008, Seattle, WA, July 2008.

Trademark Attribution

AMD, the AMD Arrow logo, AMD Opteron, AMD Athlon, AMD Turion, AMD Phenom, ATI, the ATI logo, Radeon, Catalyst, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Microsoft, Windows, and Windows Vista are registered trademarks of Microsoft Corporation in the United States and/or other jurisdictions. Other names are for identification purposes only and may be trademarks of their respective owners.

AMD ATI RADEON PREMIUM GRAPHICS

SIGGRAPH2008