GameDevelopers Conference

MARCH 20-24
SAN JOSE, CALIFORNIA

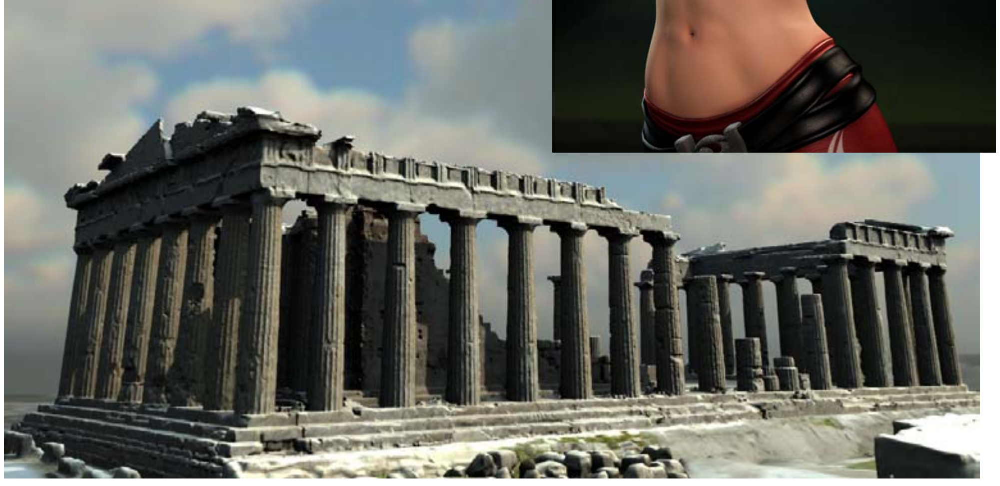
Practical Parallax Occlusion Mapping For Highly Detailed Surface Rendering

Natalya Tatarchuk

3D Application Research Group
ATI Research, Inc.

Let me introduce… myself

- Natalya Tatarchuk
  - Research Engineer
  - Lead Engineer on ToyShop
  - 3D Application Research Group
  - ATI Research, Inc.
- What we do
  - Demos
  - Tools
  - Research



The Plan

- What are we trying to solve?
- Quick review of existing approaches for surface detail rendering
- Parallax occlusion mapping details
- Discuss integration into games
- Conclusions

The Plan

- What are we trying to solve?
- Quick review of existing approaches for surface detail rendering
- Parallax occlusion mapping details
- Discuss integration into games
- Conclusions

When a Brick Wall Isn't Just a Wall of Bricks…

- Concept versus realism
  - Stylized object work well in some scenarios
  - In realistic games, we want the objects to be as detailed as possible
- Painting bricks on a wall isn't necessarily enough
  - Do they look / feel / smell like bricks?
  - What does it take to make the player really feel like they've hit a brick wall?

What Makes a Game Truly Immersive?

- Rich, detailed worlds help the illusion of realism
- Players feel more immersed into complex worlds
  - Lots to explore
  - Naturally, game play is still key
- If we want the players to think they're near a brick wall, it should look like one:
  - Grooves, bumps, scratches
  - Deep shadows
  - Turn right, turn left – still looks 3D!

The Problem We're Trying to Solve

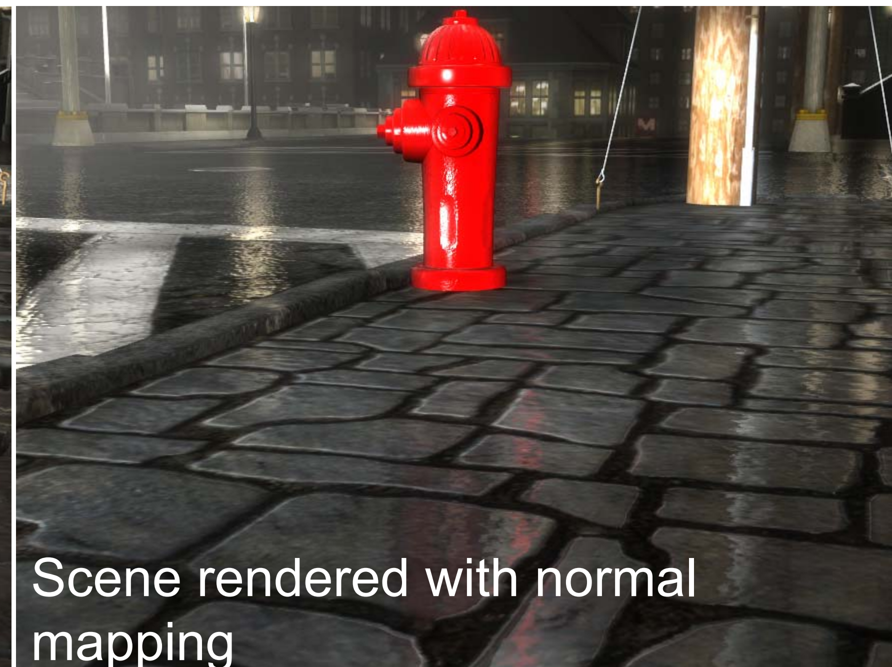
- An age-old 3D rendering balancing act
  - How do we render complex surface topology without paying the price on performance?
- Wish to render very detailed surfaces
- Don't want to pay the price of millions of triangles
  - Vertex transform cost
  - Memory footprint
- Would like to render those detailed surfaces accurately
  - Preserve depth at all angles
  - Dynamic lighting
  - Self occlusion resulting in correct shadowing

Solution: Parallax Occlusion Mapping

- Per-pixel ray tracing of a height field in tangent space
- Correctly handles complicated viewing phenomena and surface details
  - Displays motion parallax
  - Renders complex geometric surfaces such as displaced text / sharp objects
- Calculates occlusion and filters visibility samples for soft self-shadowing
- Uses flexible lighting model
- Adaptive LOD system to maximize quality and performance

Parallax Occlusion Mapping versus Normal Mapping


Scene rendered with Parallax Occlusion Mapping


Scene rendered with normal mapping

Surface Details in the ToyShop Demo

- Parallax occlusion mapping was used to render extreme high details for various surfaces in the demo
  - Brick buildings
  - Wood-block letters for the toy shop sign
  - Cobblestone sidewalk

Surface Details in the ToyShop Demo

- We were able to incorporate multiple lighting models
  - Some just used diffuse lighting
  - Others simulated wet materials
  - Integrated view-dependent reflections
  - Shadow mapping was easily integrated into the materials with parallax occlusion mapped surfaces
- All objects used the level-of-details system

Demo: ToyShop

The Plan

- What are we trying to solve?
- Quick review of existing approaches for surface detail rendering
- Parallax occlusion mapping details
- Discuss integration into games
- Conclusions

Approximating Surface Details

- First there was bump mapping… [Blinn78]
  - Rendering detailed and uneven surfaces where normals are perturbed in some pre-determined manner
  - Popularized as *normal mapping* – as a *per-pixel* technique
  - No self-shadowing of the surface
  - Coarse silhouettes expose the actual geometry being drawn

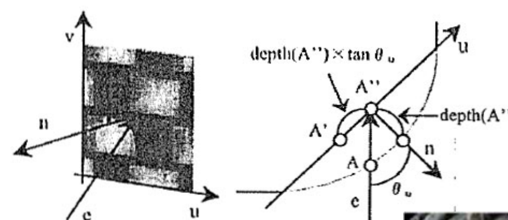
- Doesn't take into account geometric surface depth
  - Does not exhibit **parallax**

apparent displacement of the object due to viewpoint change

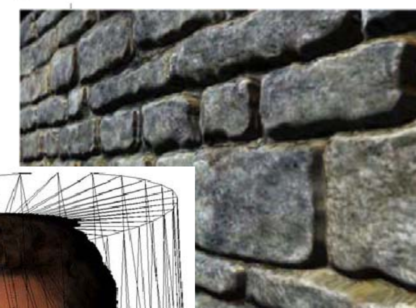
Selected Related Work

- Horizon mapping [Max88]
- Interactive horizon mapping [Sloan00]

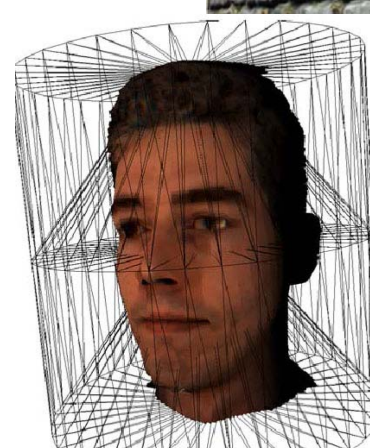
- Parallax mapping [Kaneko01]



- Parallax mapping with offset limiting [Welsh03]



- Hardware Accelerated Per-Pixel Displacement Mapping [Hirche04]



The Plan

- What are we trying to solve?
- Quick review of existing approaches for surface detail rendering
- Parallax occlusion mapping details
- Discuss integration into games
- Conclusions

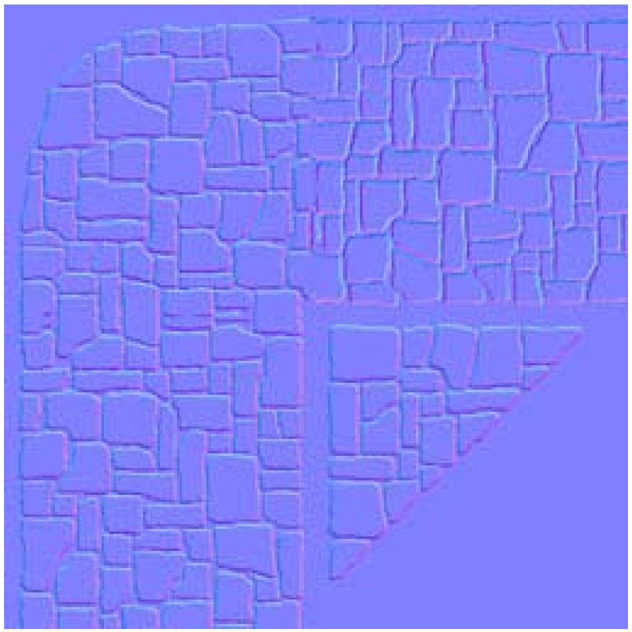
Parallax Occlusion Mapping

- Introduced in [Browley04] “Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing”
- Efficiently utilizes programmable GPU pipeline for interactive rendering rates
- Current algorithm has several significant improvements over the earlier technique

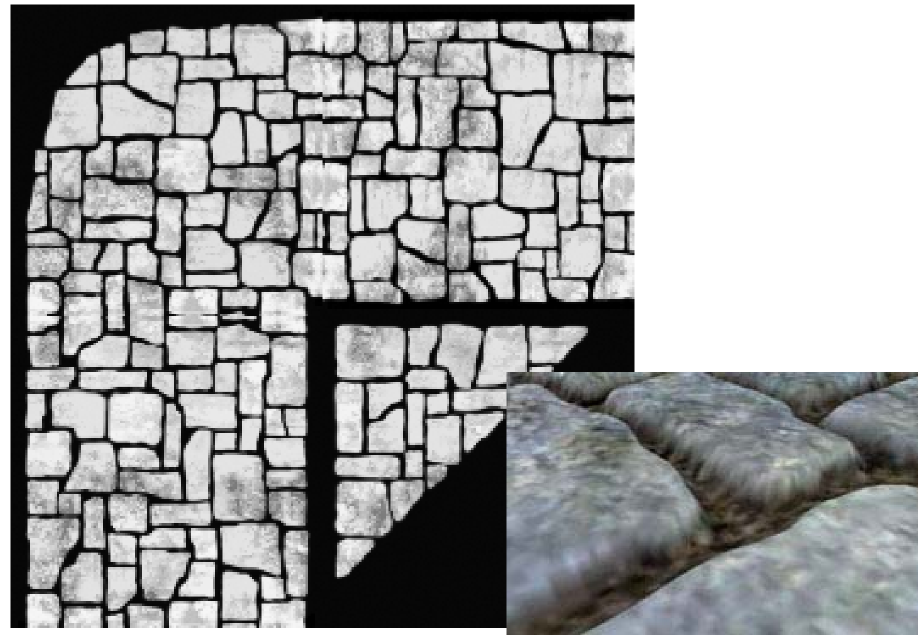
Parallax Occlusion Mapping: New Contributions

- Increased precision of height field – ray intersections
- Dynamic real-time lighting of surfaces with soft shadows due to self-occlusion under varying light conditions
- Directable level-of-detail control system with smooth transitions between levels
- Motion parallax simulation with perspective-correct depth

Encoding Displacement Information



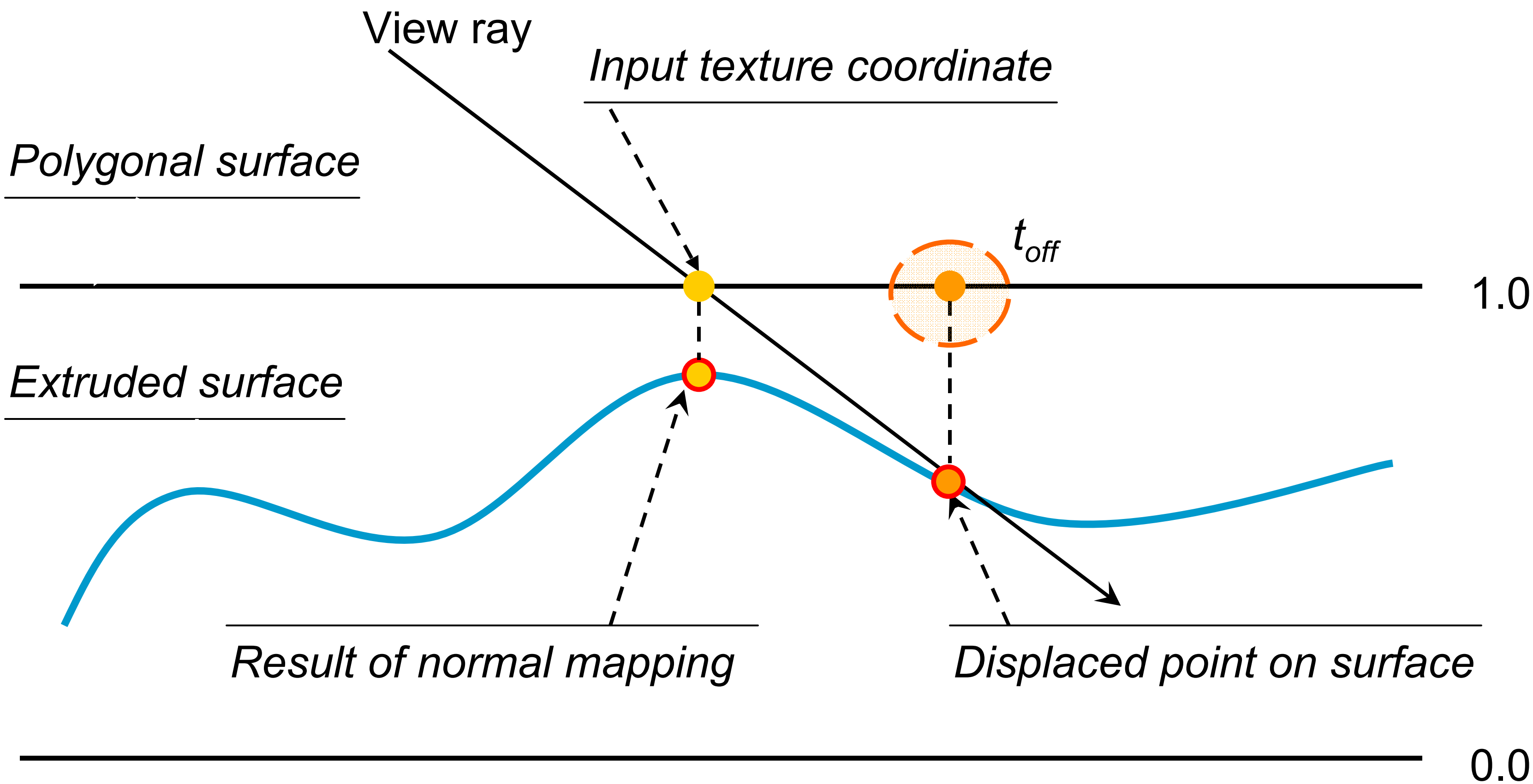
Tangent-space normal map



Height map (displacement values)

All computations are done in tangent space, and thus can be applied to arbitrary surfaces

Parallax Displacement



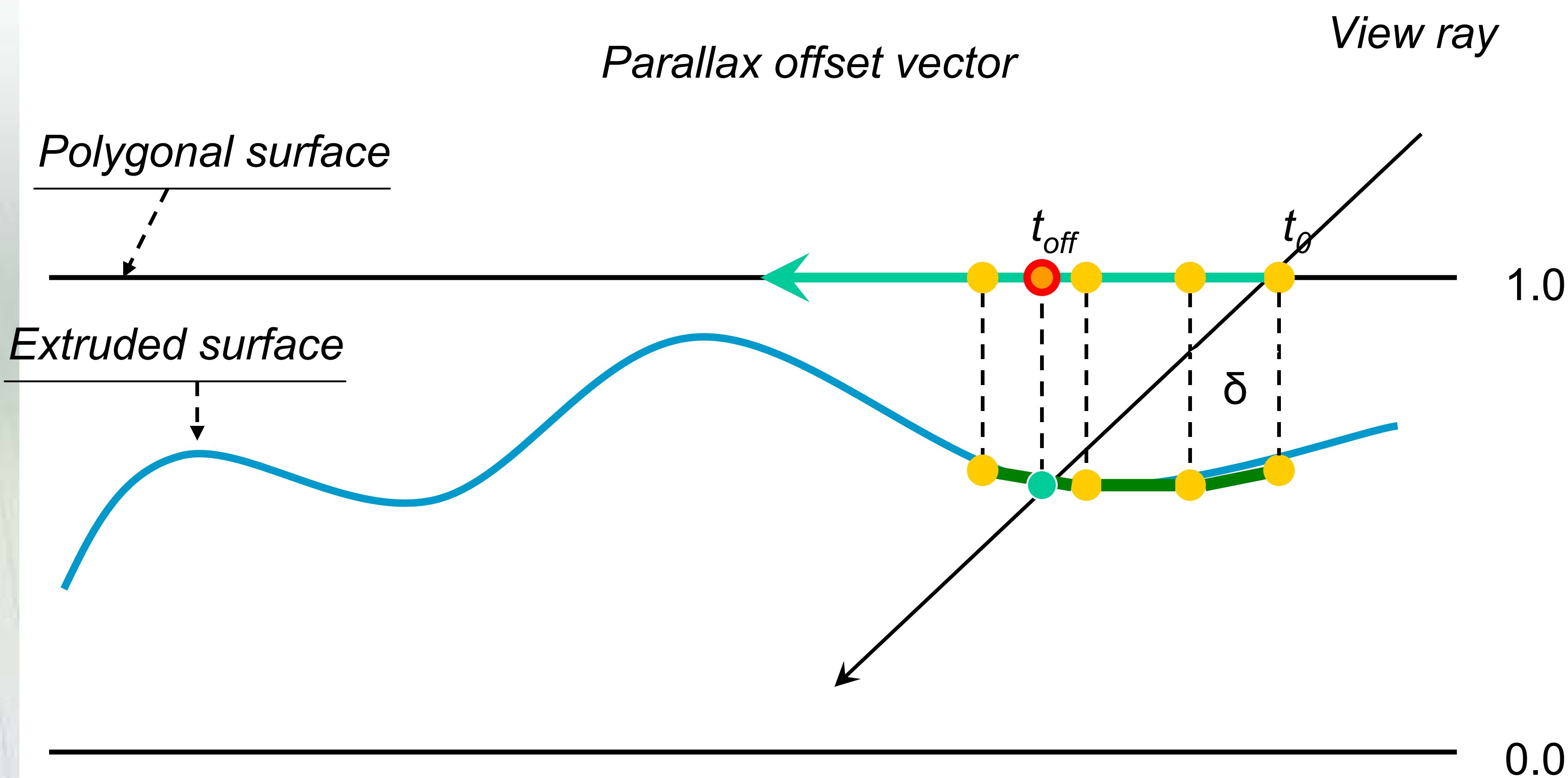
Implementation: Per-Vertex

- Compute the viewing direction, the light direction in tangent space
- Can compute the parallax offset vector (as an optimization)
  - Interpolated by the rasterizer

Implementation: Per-Pixel

- Ray-cast the view ray along the parallax offset vector
- Ray – height field profile intersection as a texture offset
  - Yields the correct displaced point visible from the given view angle
- Light ray – height profile intersection for occlusion computation to determine the visibility coefficient
- Shading
  - Using any attributes
  - Any lighting model

Height Field Profile Tracing

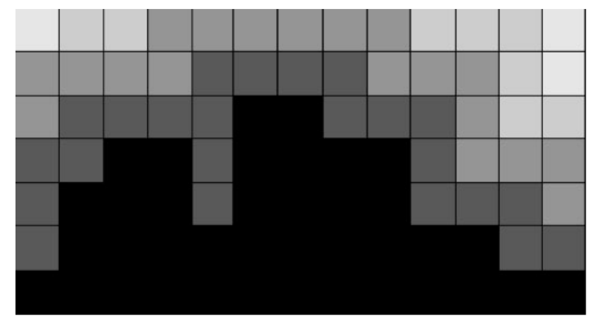


Binary Search for Surface-Ray Intersection

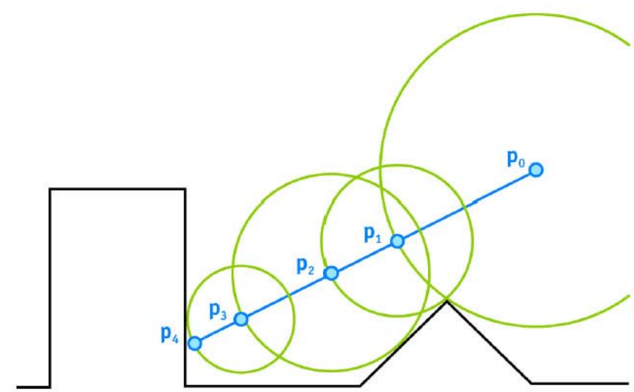
- Binary search refers to repeatedly halving the search distance to determine the displaced point
  - The height field is not sorted a priori
  - Requires dependent texture fetches for computation
    - Incurs latency cost for each successive depth level
    - Uses 5 or more levels of dependent texture fetches

Per-Pixel Displacement Mapping with Distance Functions [Donnely05]

- Also a real-time technique for rendering per-pixel displacement mapped surfaces on the GPU
  - Stores a ‘slab’ of distances to the height field in a volumetric texture

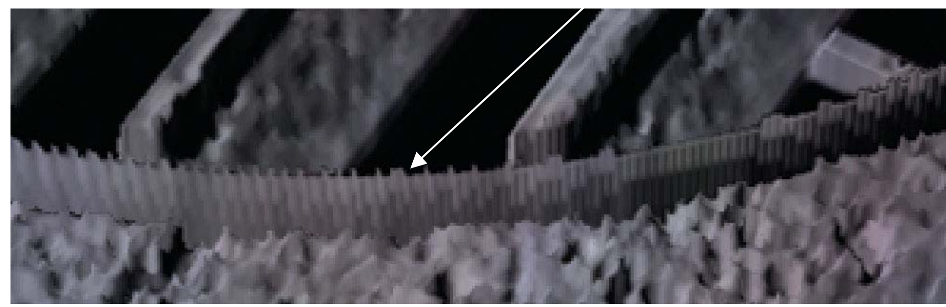


- To arrive at the displaced point, walk the volume texture in the direction of the ray
  - Instead of performing a ray-height field intersection
  - Uses dependent texture fetches, amount varies



Per-Pixel Displacement Mapping with Distance Functions [Donnely05]

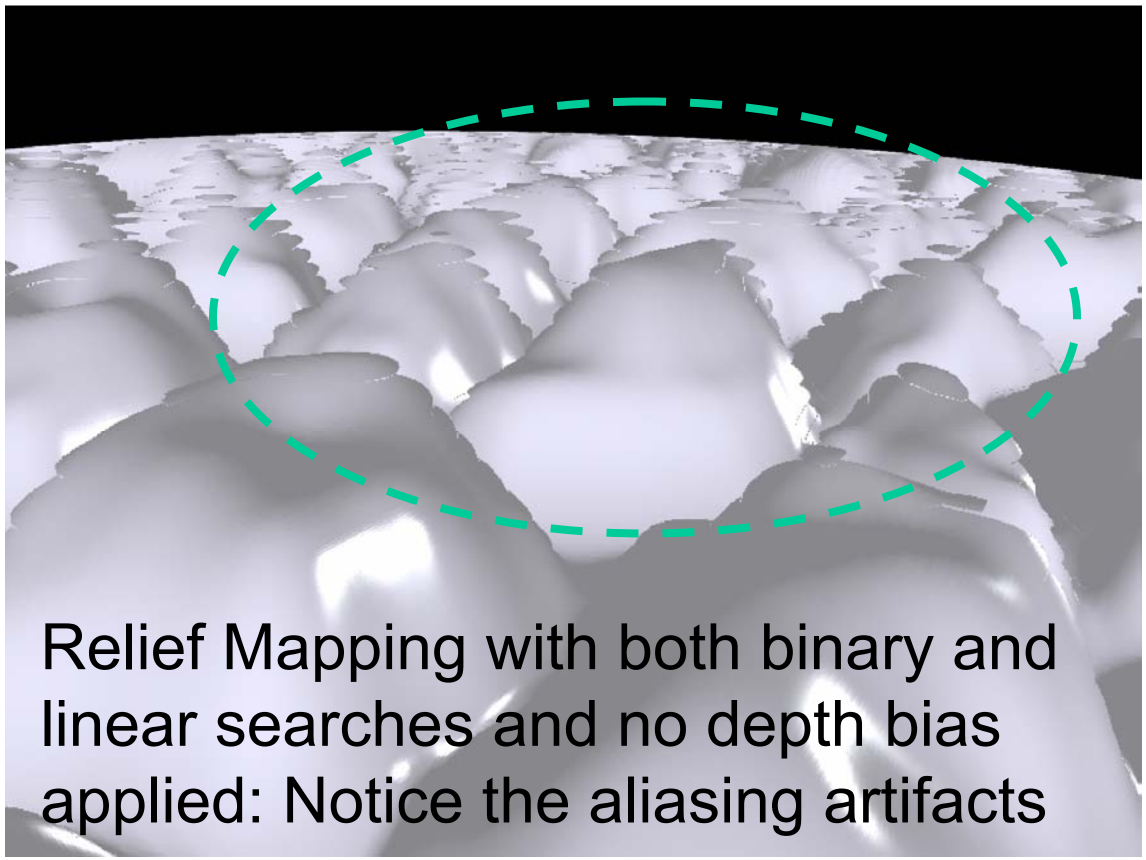
- Visible aliasing
  - Not just at grazing angles
- Only supports precomputed height fields
  - Requires preprocessing to compute volumetric distance map
  - Volumetric texture size is prohibitive
- The idea of using a distance map to arrive at the extruded surface is very useful



Linear Search for Surface-Ray Intersection

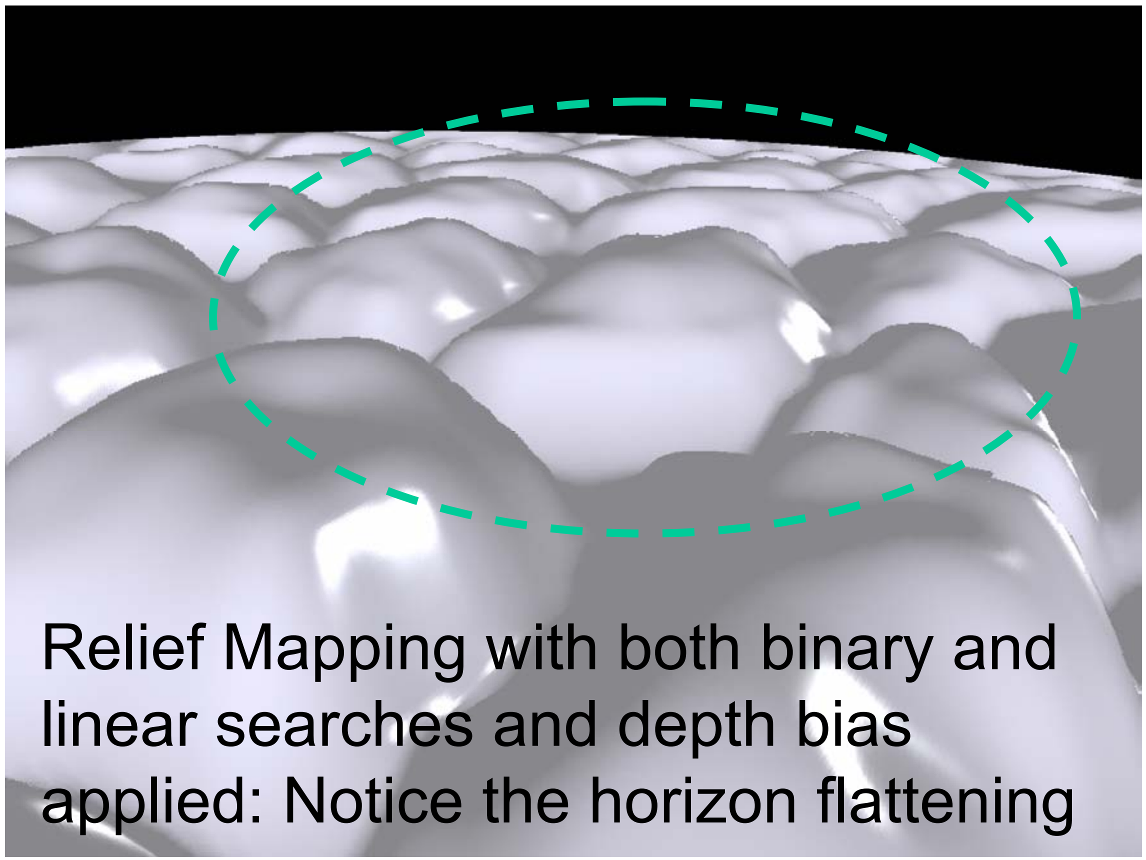
- We use just the linear search which requires only regular texture fetches
  - Fast performance
  - Using dynamic flow control, can break out of execution once the intersection is found
- Simply using linear search is not enough
  - Linear search alone does not yield good rendering results
    - Requires high precision calculations for surface-ray intersections
    - Otherwise produces visible aliasing artifacts

Comparison of Intersection Search Types and Depth Bias Application

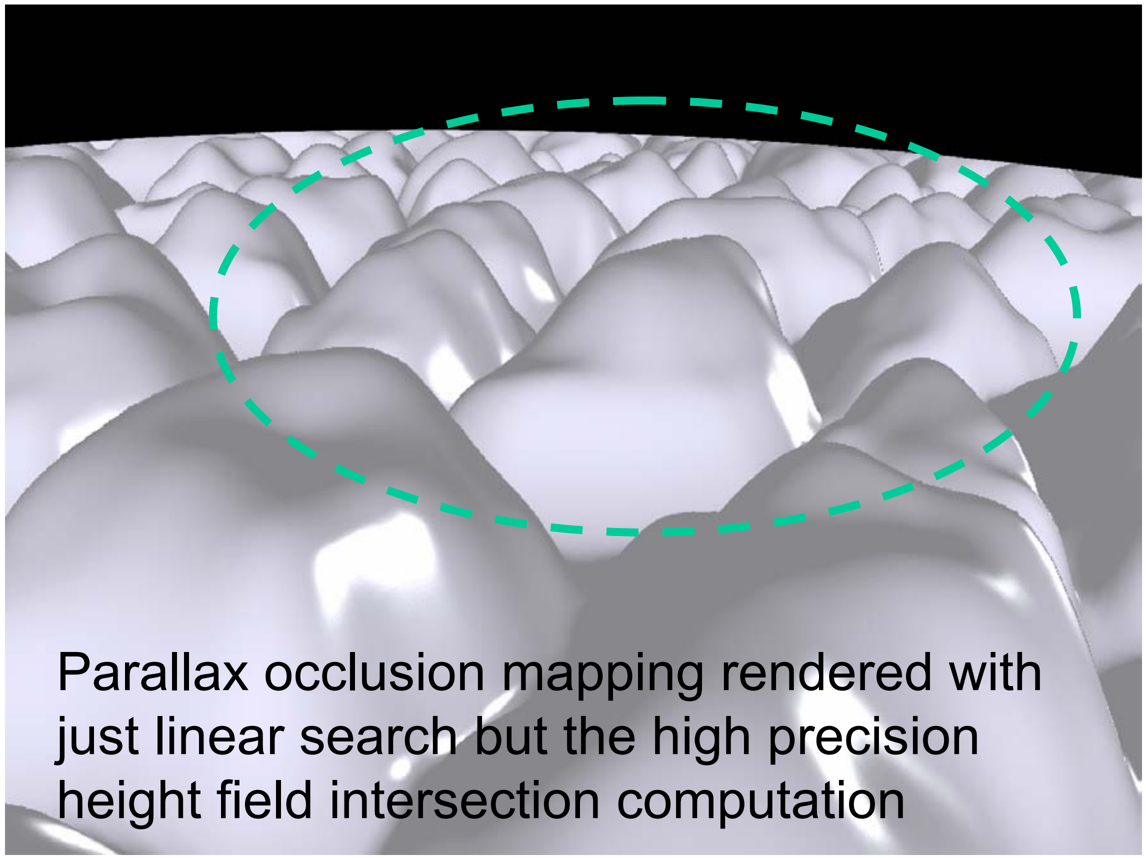


Relief Mapping with both binary and linear searches and no depth bias applied: Notice the aliasing artifacts

Comparison of Intersection Search Types and Depth Bias Application

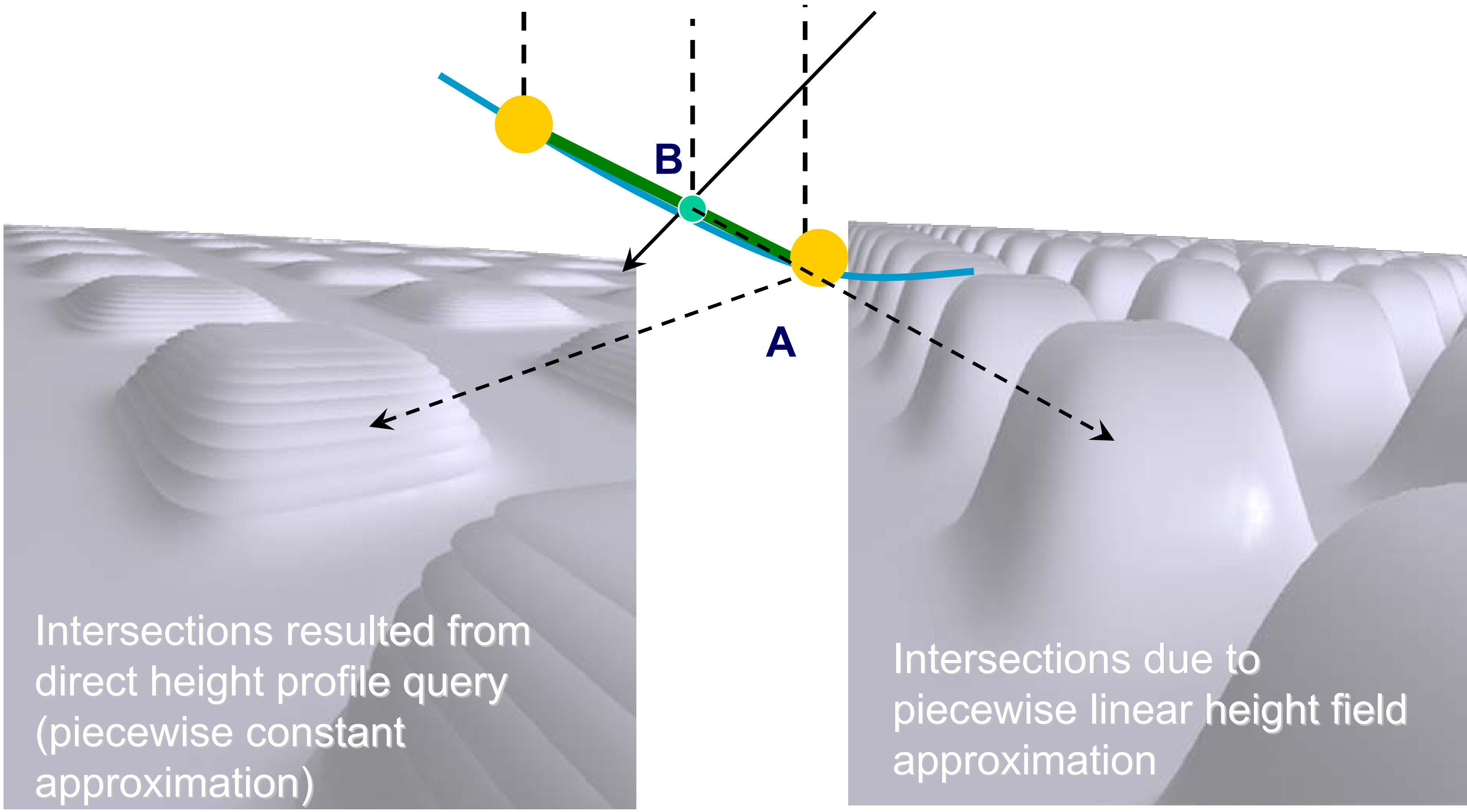


Comparison of Intersection Search Types and Depth Bias Application



Parallax occlusion mapping rendered with just linear search but the high precision height field intersection computation

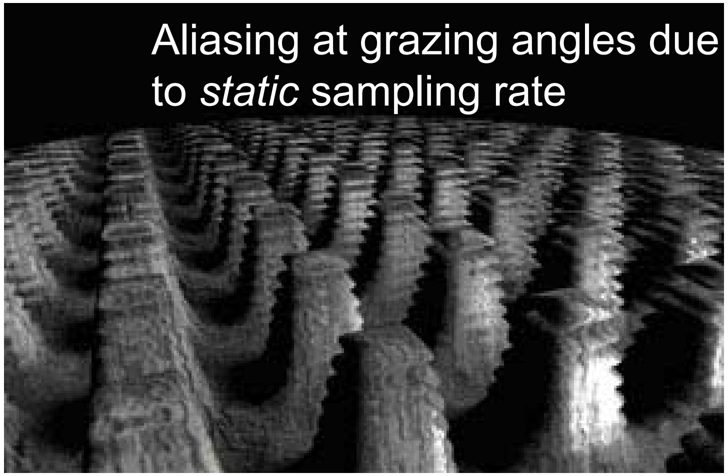
Height Field Profile – Ray Intersection

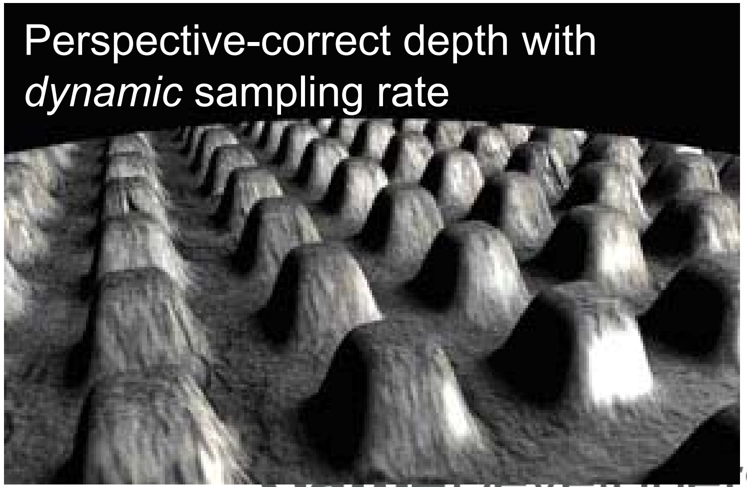


Higher Quality With Dynamic Sampling Rate

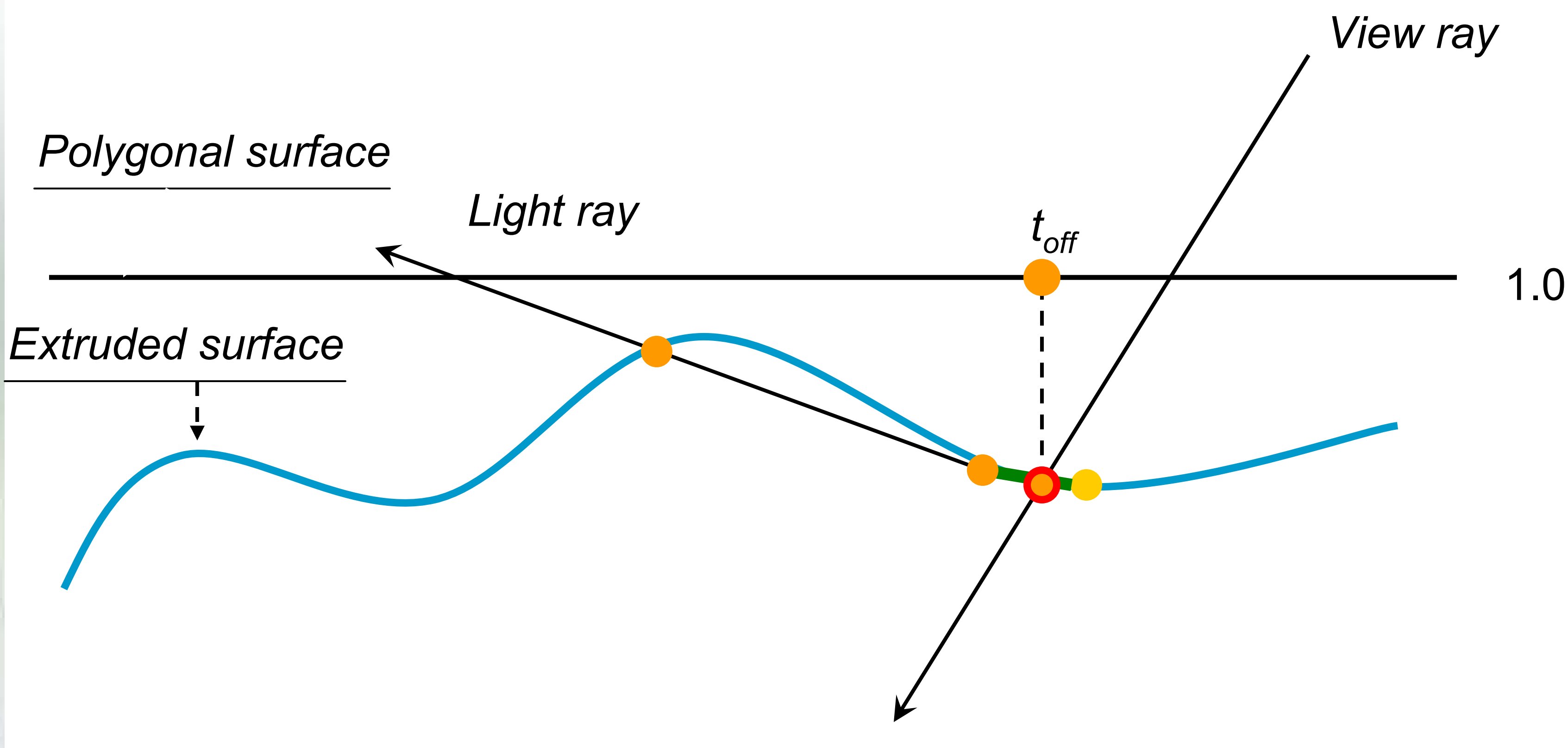
- Sampling-based algorithms are prone to aliasing
- Solution: *Dynamically* adjust the sampling rate for ray tracing as a linear function of angle between the geometric normal and the view direction ray

$$n = n_{\min} + \hat{N} \bullet \hat{V}_{ts}(n_{\max} - n_{\min})$$



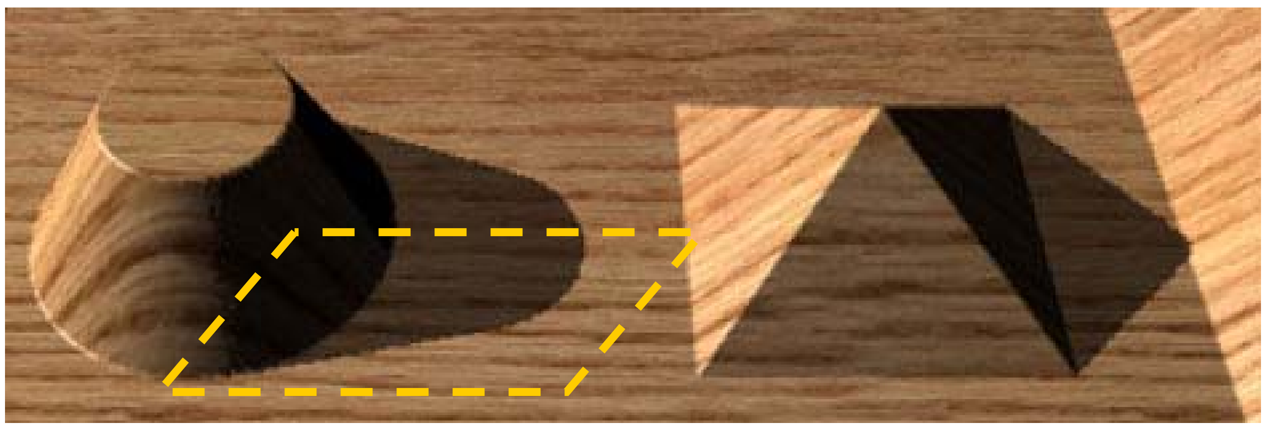


Self-Occlusion Shadows



Hard Shadows Computation

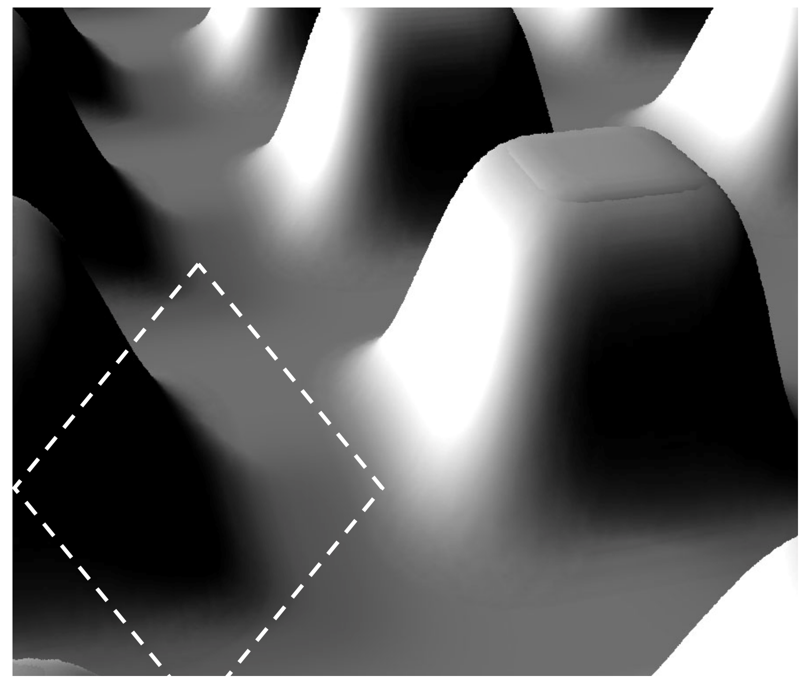
- Simply determining whether the current feature is occluded yields hard shadows



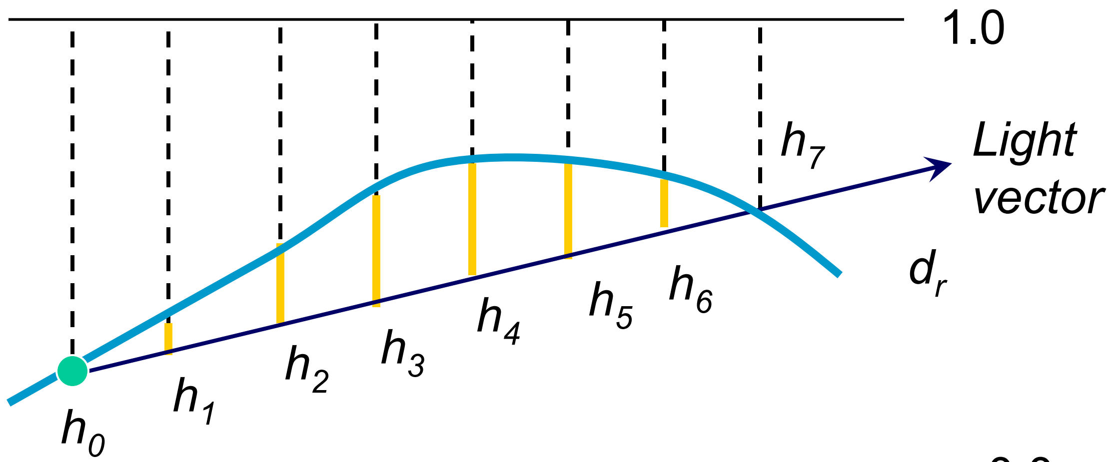
Soft Shadows Computation

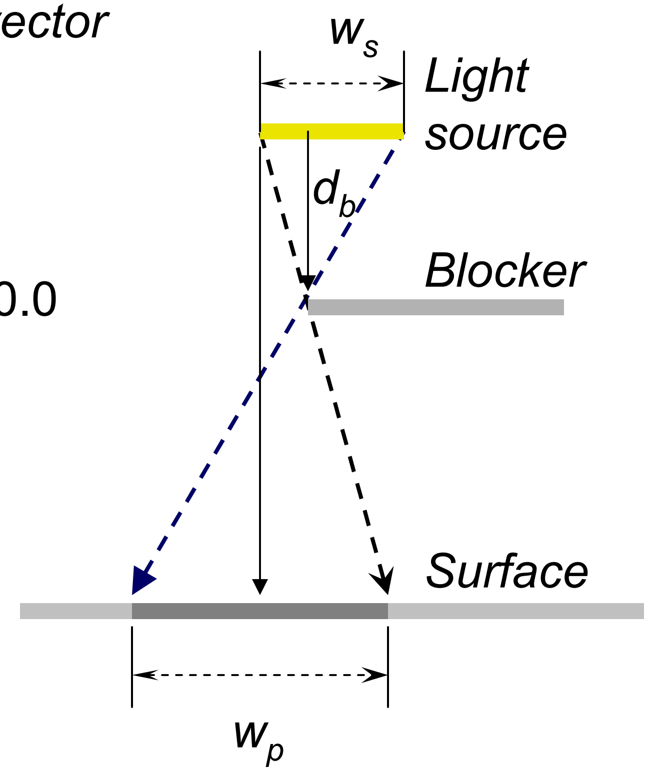
- We can compute soft shadows by filtering the visibility samples during the occlusion computation
- Don't compute shadows for objects not facing the light source:

$$N \bullet L > 0$$



Penumbral Size Approximation

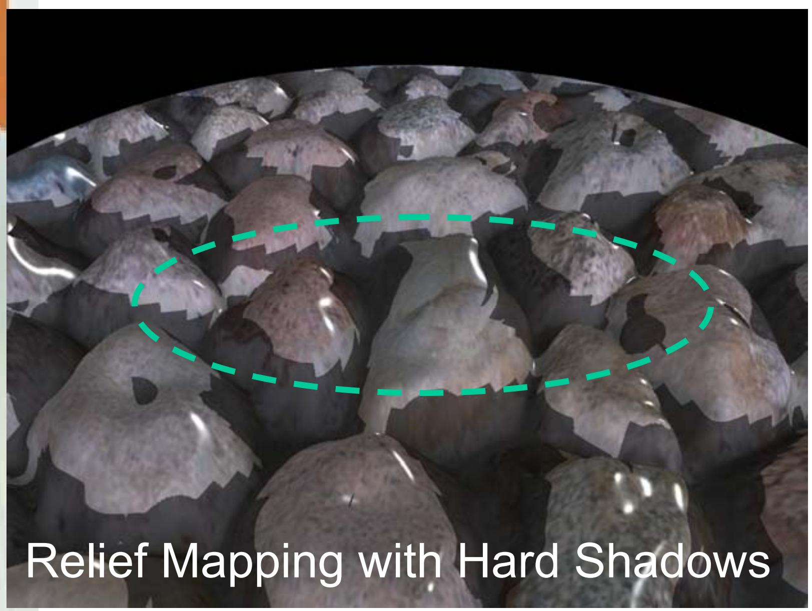


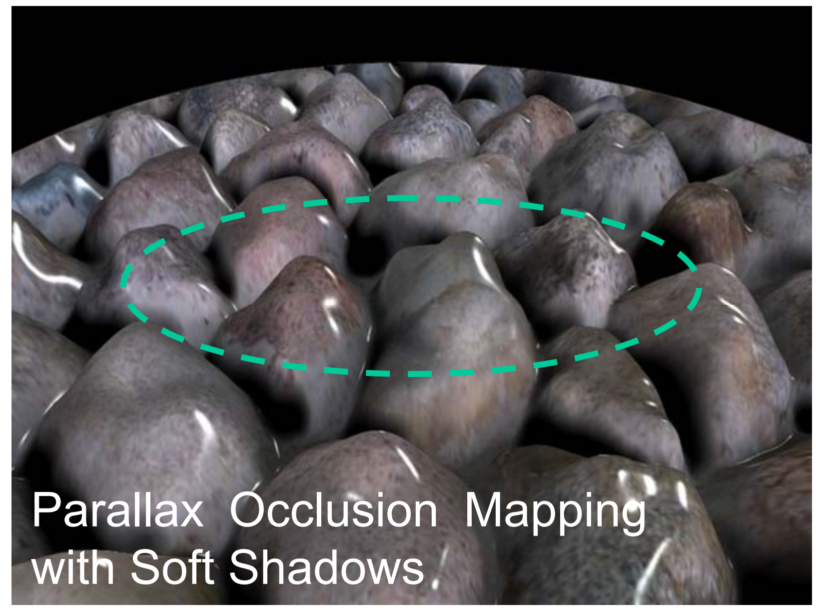


The blocker heights h_i allow us to compute the *blocker-to-receiver* ratio

$$w_p = w_s\,(d_r - d_b)\,/\,d_b$$

Shadows Comparison Example





Illuminating the Surface

- Use the computed texture coordinate offset to sample desired maps (albedo, normal, detail, etc.)
- Given those parameters and the visibility information, we can apply any lighting model as desired
  - Phong
  - Compute reflection / refraction
  - Very flexible

Can Use A Variety of Illumination Effects

- For many effects, simply diffuse lighting with base texture looks great
  - Diffuse only suffices for many effects
- Glossy specular easily computed – can use gloss maps to reduce specularity in the valleys

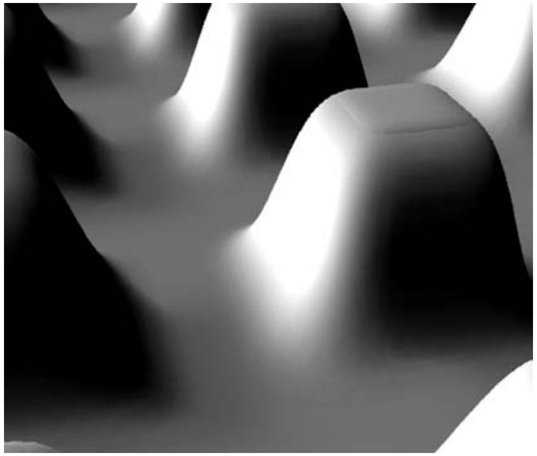
Adaptive Level-of-Detail System

- Compute the current mip map level
- For furthest LOD levels, render using normal mapping (threshold level)
- As the surface approaches the viewer, increase the sampling rate as a function of the current mip map level
- In transition region between the threshold LOD level, blend between the normal mapping and the full parallax occlusion mapping

Results

- Implemented using DirectX 9.0c shaders (separate implementations in SM 2.0, 2.b and 3.0)

RGBα texture: 1024 x 1024, non-contiguous *uvs*



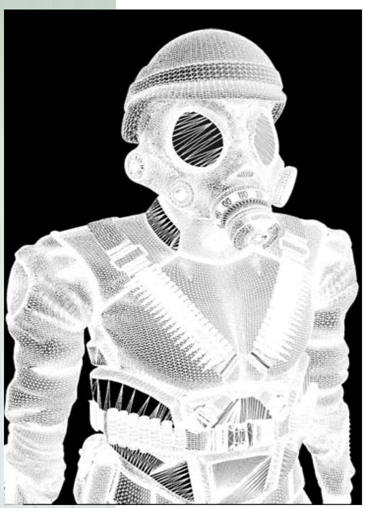
RGBα texture: tiled 128 x 128

Parallax Occlusion Mapping vs. Actual Geometry

-1100 polygons with parallax occlusion mapping (8 to 50 samples used)
- **Memory**: 79K vertex buffer
  6K index buffer
  13Mb texture (3Dc)
  (2048 x 2048 maps)

Total: **< 14 Mb**

Frame Rate:
- ***255 fps*** on ATI Radeon hardware
- ***235 fps*** with skinning



- 1,500,000 polygons with normal mapping
- **Memory**: 31Mb vertex buffer
  14Mb index buffer

Total: **45 Mb**

Frame Rate:
- ***32 fps*** on ATI Radeon hardware

Demo

The Plan

- What are we trying to solve?
- Quick review of existing approaches for surface detail rendering
- Parallax occlusion mapping details
- Discuss integration into games
  - Performance analysis and optimizations
  - Considerations for authoring art assets
- Conclusions

How Does One Render Height Maps, Exactly?

- Two possibilities
  - Render surface details as if “pushed down” – the actual polygonal surface will be above the rendered surface
  - In this case the top (polygon face) is at height = 1, and the deepest value is at 0
  - Or - actually push surface details upward (ala displacement mapping)
- This affects both the art pipeline and the actual algorithm
- In the presented algorithm, we render the surface pushed down

Performance vs Image Quality

- Tradeoffs between speed and quality
  - Less samples means more possibility for missed features and incorrect intersections
  - This can result in stair stepping artifacts at oblique angles
- Silhouettes are not computed correctly
  - Art can be authored to hide this artifact
  - Alternatives exist (at the expense of memory and extra computations)
    - Use vertex curvature data and texkill in the pixel shader to clip pixels at the silhouettes
    - Relief Mapping example shows a result
    - Aliasing at the object silhouettes can be very strong

Incorporate Dynamic Height Field Rendering with POM

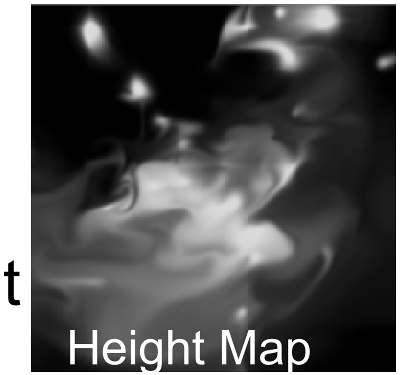
- Easily supports *dynamically rendered* height fields
  - Generate height field
  - Compute normals for this height field
  - Apply inverse displacement mapping w/ POM algorithm to that height field
  - Shade using computed normals
- Examples of dynamic HF generation:
  - Water waves / procedurally generated objects / noise
  - Explosions in objects
  - Bullet holes
- Approaches that rely on precomputation do not support dynamic height field rendering in real-time
  - Displacement mapping with distance maps
  - Encoding additional vertex data such as curvature

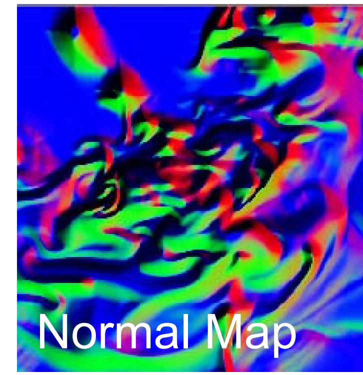
Combine Fluid Dynamics with POM

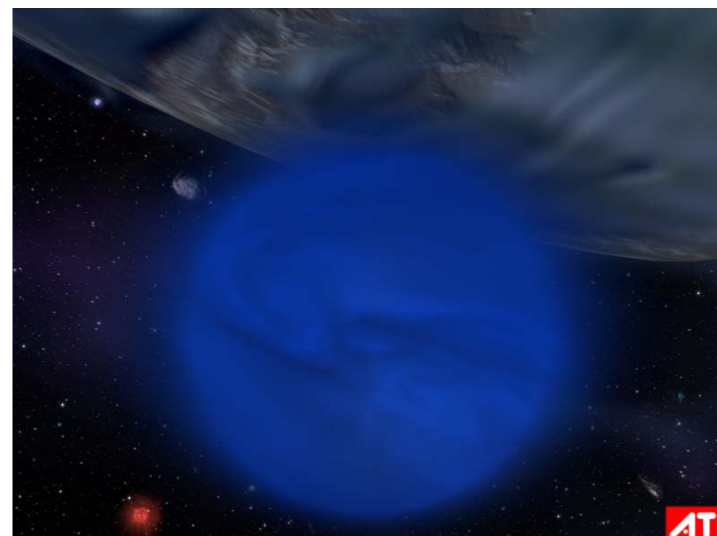
- Compute Navier-Stokes simulation for fluid dynamics for a height field
  - Example: Fluid flow in mysterious galaxies from "Screen Space" ATI X1900 screen saver
- Fluid dynamics algorithm can be executed entirely on the GPU
  - See ATI technical report on "Explicit Early-Z Culling for Efficient Fluid Flow Simulation and Rendering" by P. Sander, N. Tatarchuk and J.L. Mitchell for details

Example: Gas Planet Scene

- Random particles in texture space emit flow density and velocity
- Flow used to compute height field for parallax occlusion mapping
- Compute dynamic normals for the flow height field
- Parallax occlusion mapping used to simulate cloud layer on large planet

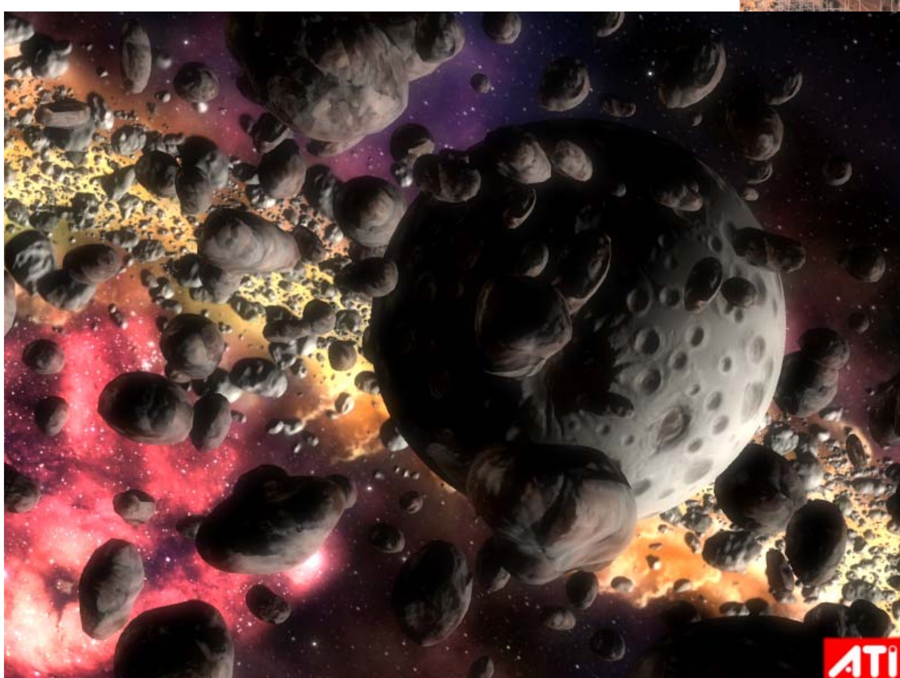


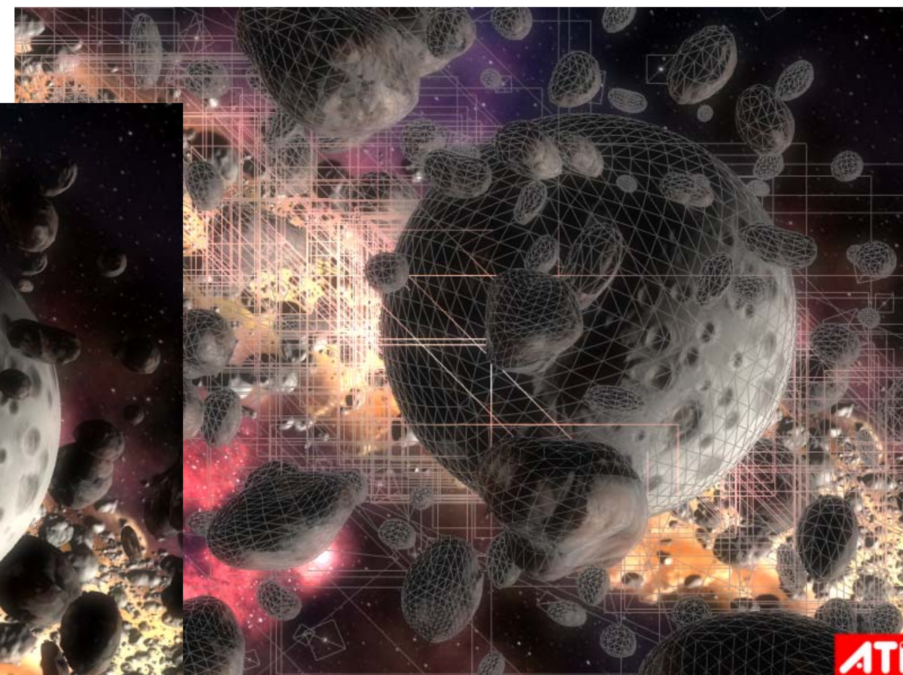




Other Examples: Asteroids scene

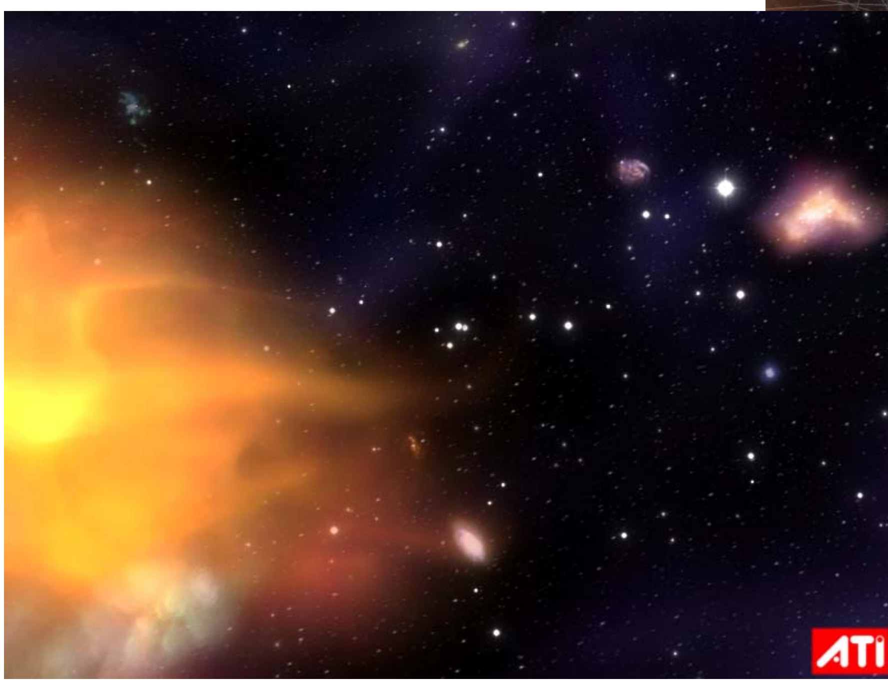
- Scene with several parallax mapped asteroids
- Billboards used for faraway nebulae

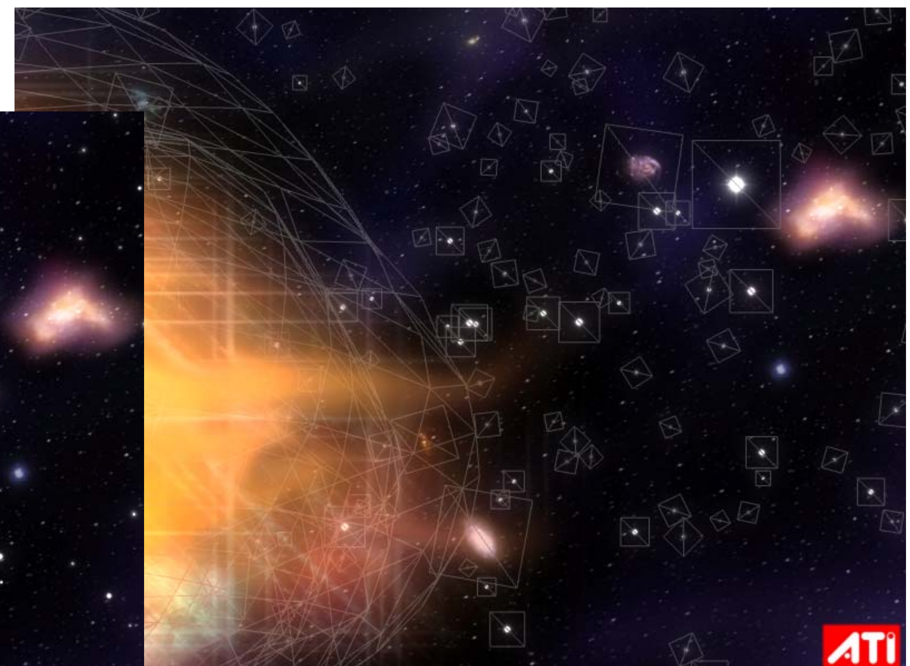




Nebula scene

- Several layers of parallax mapped geometry
- Flow density and velocity emitted in screen space at all layers





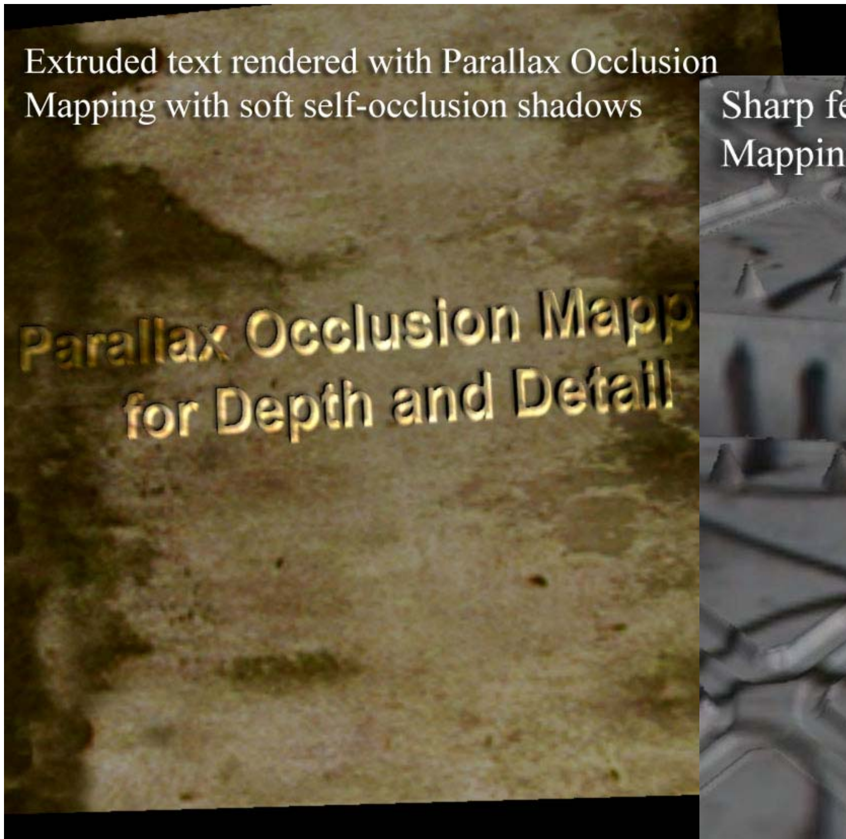
Correct Depth Output

- Simply using parallax occlusion mapping will yield incorrect object intersection
  - Depth will be computed for the reference surface
  - May display object gaps or cut-throughs
- Solution: update each pixel's Z value when computing the displacement
  - Compensate for simulated extruded surface
  - Use the height field value and the reference plane Z value to compute correct depth
  - [Policarpo05] shows an example
- Performance will be affected
  - Z is output from the pixel shader
  - No longer able to use HiZ for optimization

Parallax Occlusion Mapping with Curved Surfaces

- Since the computation is in tangent space, the approach can be used with any surfaces
  - Works equally well on curved objects
  - Beware of silhouettes
- If vertex curvature can be encoded vertex data
  - Extend current algorithm to use that data to improve height-field intersection using the curvature
  - This reduces aliasing and potential misses at steep grazing angles

Able to Handle Difficult Cases



Extruded text rendered with Parallax Occlusion Mapping with soft self-occlusion shadows

Sharp features rendered with Parallax Occlusion Mapping

Game Developers Conference

Shader Implementation Details

- Really takes advantage of the great architecture of current and next-gen GPUs
  - Balances texture fetches and control flow with ALU load
  - Flow control:
    - Uses dynamic flow control when supported
    - Flow control cost is offset by the ALU / texture fetches
    - ATI Shader Compiler makes aggressive optimizations
- Easily supports a range of Dx9 hardware targets
  - Multipass w/ ps_2_0
  - Single pass in ps_2_b
  - Single pass dynamic flow control in ps_3_0

PS_2_0 Shader Details

- Uses static flow control to compute intersections
  - Compute parallax offset in first pass, output to render target
  - In second pass computing lighting and shadow term
- 8 samples in 64 instructions
  - Performs quite fast
  - Doesn't use dynamic number of iterations so the number of samples for height field tracing is constant
  - This may cause some sampling aliasing at grazing angles if not enough samples are used
  - Can use more than one pass to sample height map at higher frequencies
  - 2-3 passes 8 samples each gives good results
    - Makes oblique angles look better!

PS_2_b Shader Details

- Single pass to compute the parallaxed offset, lighting and self-shadowing
- Uses a static number of iterations to compute height field intersections
  - This may cause some sampling aliasing at grazing angles if not enough samples are used
- Great performance
- Use as many samples as needed for your art / scene
  - Pay in form of instructions

Shader Model 3.0 Gives Ideal Results

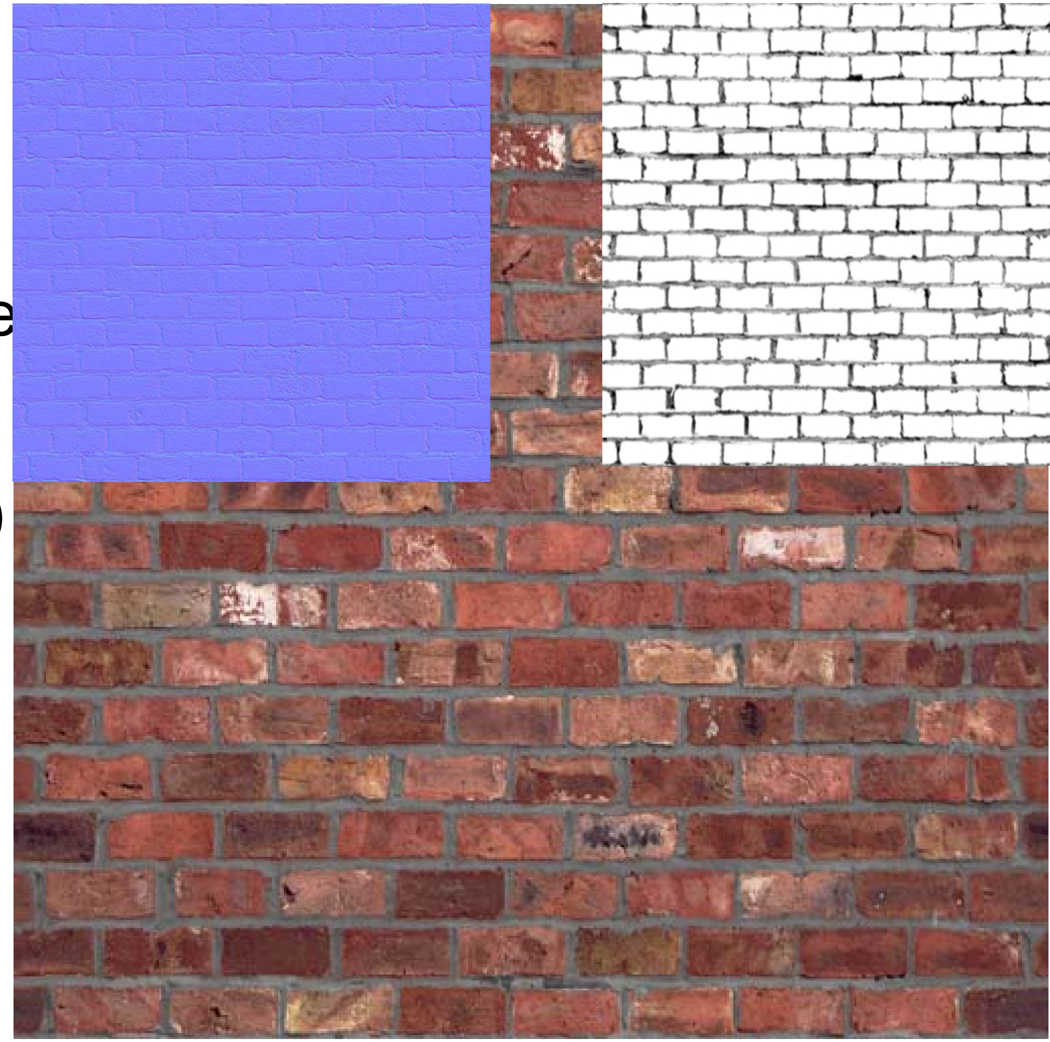
- Uses dynamic flow control and early out during ray-tracing operations
  - A close relationship with the assembly is key
  - Always double-check to see if what you are expecting to get is what you are getting
  - Beware of unrolled static loops
- ***Best quality results*** and *optimizations*
- Nicely balances ALU ops with control flow instructions and texture fetches
- ATI Driver Shader Compiler optimizations in action:
  - A *200 ALU* ops and *32 texture ops* of the disassembled HLSL shader becomes **96 ALU** and **20 texture fetches**
  - That’s 50% faster!

Authoring Art for POM: Pointers

- Easiest – less detailed height maps with wide features
  - If rendering bricks or cobble stones, it helps to have wider grout (“valley”) regions
  - Soft, blurry height maps perform better
- This algorithm gives the artist control over the range for displacing pixels
  - This represents the range of the height field
  - Easily modifiable to get the right look
- Remember – the algorithm is pushing down, not up
  - Use this when placing geometry – may need to play the actual geometry higher than planning to render
  - Height map: white is the top, black is the bottom

POM Art Assets

- Color Map
- Normal map
  - In tangent space
- Height Map
  - 8-bit (grayscale)
- That’s it!
- Minimal increase in memory use



Authoring Strategies

- For planar surfaces
  - High-poly source data compared to low poly approximation
  - Converting 2d texture data to normal map works well for flat surfaces
- For non-planar surfaces
  - Generate normal and height maps from highly detailed geometry
- Avoid drastic height changes
  - Blurring height map can help

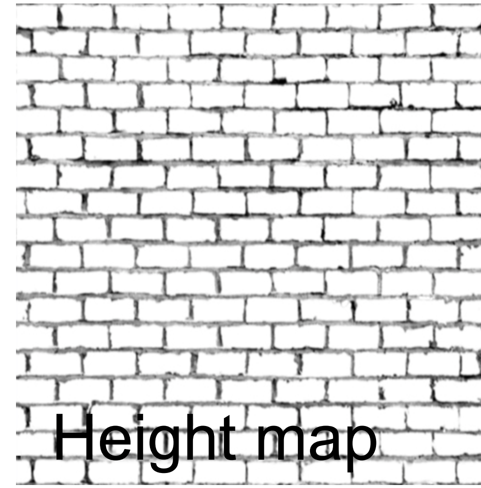
Authoring Art Considerations for POM

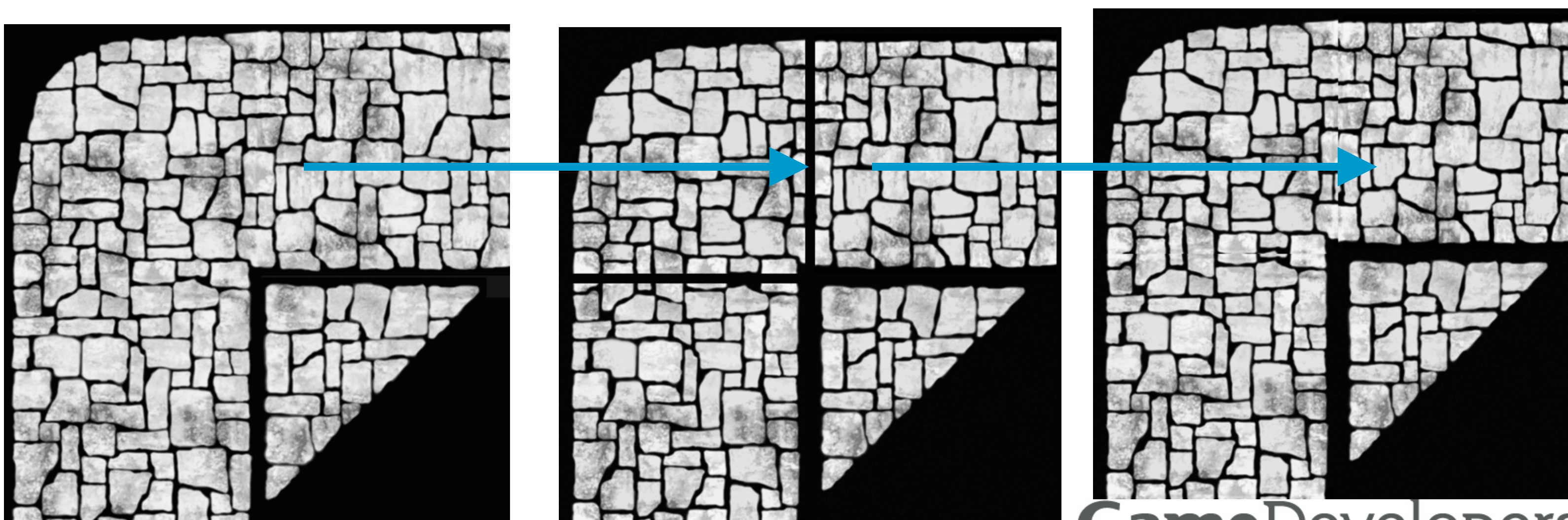
- Can alias at extreme viewing angles
- Stretching of texture coordinates
  - In some cases requires smooth height maps or high resolution maps
- Intersecting geometry clips at original height, not at displaced height
  - One can modify the shader to compute depth based on the extruded surface intersection
- Tile sets require buffer region to eliminate seam artifacts

WHAT'S NEXT :: GDC:06

POM and Tilesets

- Need Buffer Regions
- 10-20 pixel buffer region
- Authoring POM tilesets can must be done with care





The Plan

- What are we trying to solve?
- Quick review of existing approaches for surface detail rendering
- Parallax occlusion mapping details
- Discuss integration into games
- Conclusions

Conclusions

- Powerful technique for rendering complex surface details in real time
  - Higher precision height field – ray intersection computation
  - Self-shadowing for self-occlusion in real-time
  - LOD rendering technique for textured scenes
- Produces excellent lighting results
- Has modest texture memory footprint
  - Comparable to normal mapping
- Efficiently uses existing pixel pipelines for highly interactive rendering
- Supports dynamic rendering of height fields and animated objects

Acknowledgements

- Zoe Brawley, *Relic Entertainment*
- Pedro Sander, for ScreenSpace screensaver work and related slides

The ToyShop Team

Lead Artist

Dan Roeger

Lead Programmer

Natalya Tatarchuk

David Gosselin

Artists

Daniel Szecket, Eli Turner, and Abe Wiley

Engine / Shader Programming

John Isidoro, Dan Ginsburg, Thorsten Scheuermann and Chris Oat

Producer

Lisa Close

Manager

Callan McInally

Reference Material

- www.ati.com/developer
  - Demos, GDC presentations, papers and technical reports, and related materials
- N. Tatarchuk. 2006. "Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows", *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*
- P. Sander, N. Tatarchuk, J. L. Mitchell. 2004. "Explicit Early-Z Culling for Efficient Flow Simulation and Rendering", *ATI Research Technical Report,* August 2004.
- ATI ToyShop demo: http://www.ati.com/developer/demos/rx1800.html
  ATI ScreenSpace screen saver: http://www.ati.com/designpartners/media/screensavers/RadeonX1k.html

Questions?

- natasha@ati.com