Those Delicious Texels

or...

Dynamic Image-Space Per-Pixel Displacement Mapping with Silhouette Antialiasing via Parallax Occlusion Mapping

Natalya Tatarchuk
3D Application Research Group
ATI Research, Inc.

Overview of the Talk

- The goal
- Overview of approaches to simulate surface detail
- Parallax occlusion mapping
  - Theory overview
  - Algorithm details
  - Performance analysis and optimizations
  - In the demo
  - Art considerations
  - Uses and future work
- Conclusions

What Exactly Is the Problem?

- We want to render very detailed surfaces
- Don’t want to pay the price of millions of triangles
  - Vertex transform cost
  - Memory
- Want to render those detailed surfaces correctly
  - Preserve depth at all angles
  - Dynamic lighting
  - Self occlusion resulting in correct shadowing
- Thirsty graphics cards want more ALU operations (and they can handle them!)
  - Of course, this is a balancing game - fill versus vertex transform – you judge for yourself what’s best

This is Not Your Typical Parallax Mapping!

- *Parallax Occlusion Mapping* is a *new* technique
- Different from
  - Parallax Mapping
  - Relief Texture Mapping
- Per-pixel ray tracing at its core
  - There are some very recent similar techniques
- Correctly handles complicated viewing phenomena and surface details
  - Displays motion parallax
  - Can render complex geometric surfaces such as displaced text / sharp objects
  - Surfaces with self-occlusion result in correct self-shadowing
  - Uses flexible lighting model

We Want the Details… but Not All the Work!… (the early days)

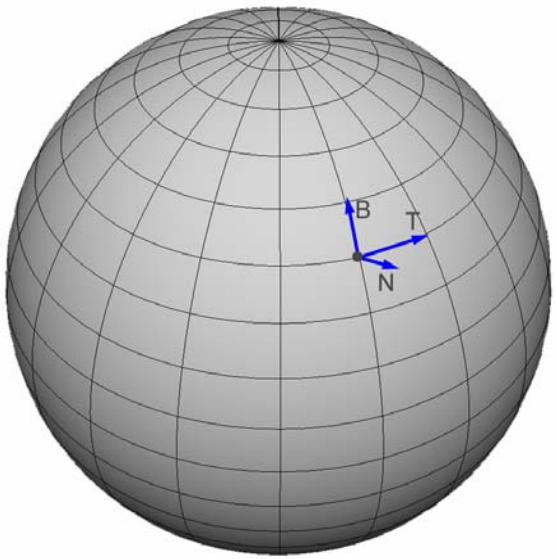
- First there was bump mapping… [Blinn78]
  - Detailed and uneven surfaces in some pre-determined manner
  - Perturbs surface normal using a texture
  - Popularized as normal mapping – *per-pixel*

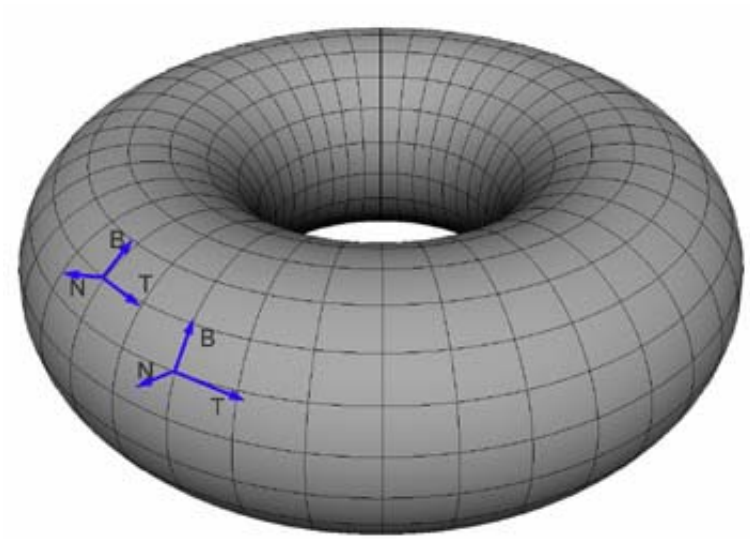
- Doesn't take into account geometric depth of surface
  - Does not exhibit **parallax**
  - No self-shadowing of the surface
  - Smooth silhouettes tell the truth about the surface…

apparent displacement of the object due to viewpoint change

Bump Mapping and Tangent Space

- Normal vectors are computed in the tangent space for each vertex
- Tangent space orients the normal map at each vertex
  - Technically ***u*** and ***v*** can be arbitrarily oriented
  - Tangent space basis vectors ***T*** and ***B*** orient ***u*** and ***v***



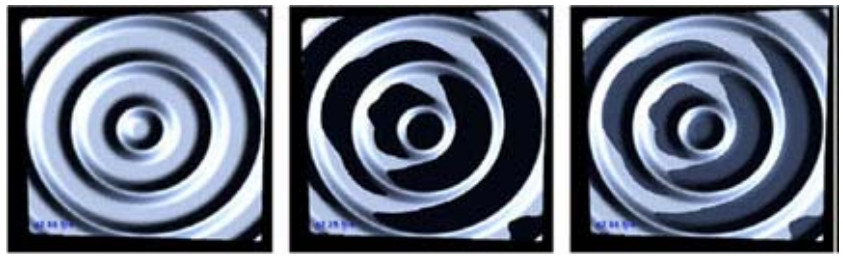


Lighting in Tangent Space

- To compute lighting from normals in tangent space, we must transform the light vector into tangent space
  - *This must be done for each vertex*
  - Tangent space is computed at preprocessing time

$$\begin{pmatrix} t_x & t_y & t_y & 0 \\ b_x & b_y & b_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

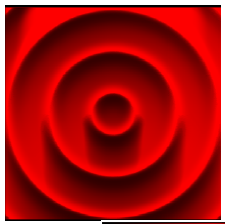
Modeling Self-Shadowing



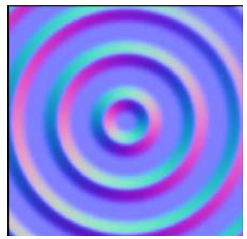
[Sloan00]

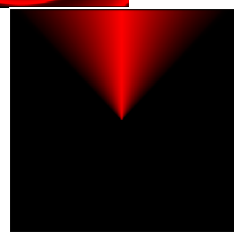
- Interactive Horizon Mapping
  - Based on horizon mapping [Max88]
  - Encodes the height of the shadowing horizon at each point on the bump map in a series of textures for 8 directions
  - Determines the amount of self-shadowing for a given light position
  - At each frame project the light vector onto local tangent plane and compute per-pixel lighting
  - Draw backs:
    - Texture memory
    - Original approach: Multipass

North Horizon Map



Normal map





North Basis Texture

More Self-Shadowing Efforts

- Illuminating micro geometry
  - Precomputes visibility
  - Computes light scattering in height fields
  - Uses the Method of Dependent Tests (Monte Carlo technique) to map it to multitexturing hardware

[Heidrich00]

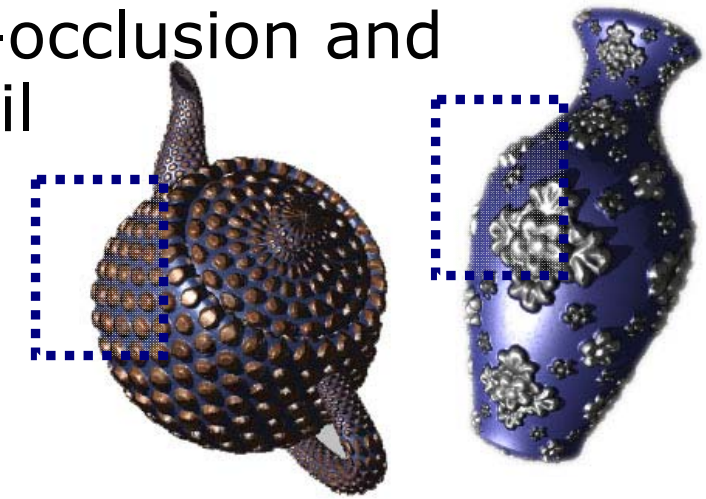
The Holy Grail: Displacement Mapping

- Tessellates the original surface into a large number of small triangles
  - Vertices displaced by a height field
  - RenderMan-quality displacement mapping – sub-pixel triangles
- Looks great!
  - Provides self-shadowing / self-occlusion / correct silhouettes
- Comes at a cost:
  - Large number of triangles → memory / transform
  - Not currently available on consumer hardware

View-Dependent Displacement Mapping

- Per-pixel technique using precomputed surface description (from a given height map)

[Wang03]



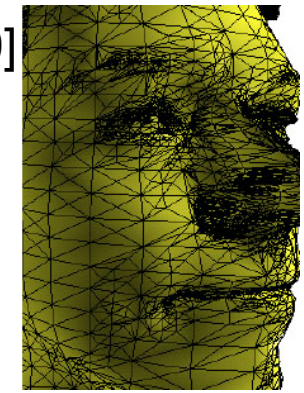
- Handles self-shadowing and self-occlusion and correct display of silhouette detail
- View-dependent approach gives convincing parallax effect
  - Stores texel relationship from several viewing directions
- Pros:
  - Convincing results at high frame rates
  - Good parallax effects

View-Dependent Displacement Mapping Costs

- Multiple texture maps to store per-pixel surface description
- Preprocessing time: computing the description texture maps from the height map
- Uses pixel shader: 40 instructions and 14 texture lookups
- 128 x 128 height fields w/ 32 x 8 view directions = 64Mb uncompressed / 4Mb compressed textures
  - But 128 x 128 resolution for height / normal map is not typical for complicated objects! It's usually higher

View-Dependent Tessellation of Displacement Maps

[Doggett00]



- Recursive tessellation on the fly
  - Uses displacement map to decide tessellation level
- View-dependent resolution of the displaced mesh
  - Uses screen size of the current edge to stop tessellation
- Avoids generating cracks on the displaced surface
  - Tessellation based only on edge information – not on neighbor information → friendly to current hardware
- Uses vertex transform units to calculate view-dependence test during tessellation
- In the end... render many, many triangles

Getting That Feeling... of Motion, of Course

- Wanted: Motion Parallax
  - Surface should appear to move correctly with respect to the viewer
- Recently many approaches appeared to solve this for rendering
  - Relief Texture Mapping by Oliveira et al in 2000
  - Parallax Mapping was introduced by Kaneko in 2001
  - Popularized by Welsh in 2003 with offset limiting technique
  - Yerex discussed displacement mapping with ray casting in hardware in a Siggraph 2004 sketch

Relief Texture Mapping

- Supports surface details in 3D and view motion parallax
- Uses a relief texture – orthogonal displacements per texel
  - Rendered as a regular texture when viewed from far away
  - When viewer is close, the mapping is modified ("warped")
  - Can be rendered as a mesh of micro polygons when extremely close
- Uses 1D image operations to apply pre-warp transform to map the relief texture to the polygon
  - Pre-warp happens in software
  - Essentially determines the portion of the surface that would be visible when viewed from that angle instead of what is actually visible)
- Reconstructs the pre-warped texture with two passes to be viewed as a texture mapped polygon

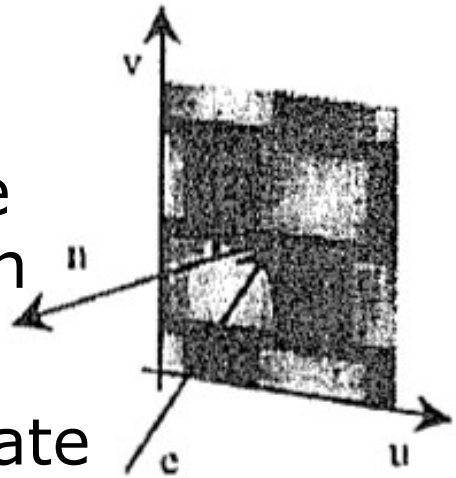
Relief Texture Mapping (cont.)

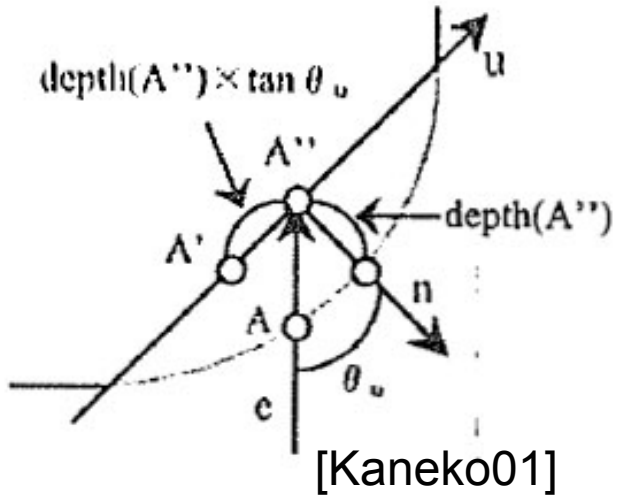
- Pros:
  - Quality results: correctly handles self-occlusions
  - Produces accurate silhouettes
- Cons:
  - Software operations for image pre-warp
  - Memory footprint for relief textures
  - Additional preprocessing time to compute the relief textures

[Oliveira00]

Parallax Mapping

- Introduced by Kaneko in 2001
- Simple way to approximate motion parallax effects on a given polygon
- Dynamically distorts the texture coordinate to approximate motion parallax effect
- Shifts texture coordinate using the view vector and the current height map value





[Kaneko01]

Parallax Mapping (cont.)

- Issues:
  - Doesn't accurately represent surface depth
  - Swimming artifacts at grazing angles
  - Flattens geometry at grazing angles
- Pros:
  - No additional texture memory and very quick (~3 extra instructions)

Parallax Mapping with Offset Limiting

- Same idea as in [Kaneko01], implemented as OpenGL ARB_*_program
- Uses height map to determine texture coordinate offset for approximating parallax
- Uses view vector in tangent space to determine how to offset the texels

- Reduces visual artifacts at grazing angles ("swimming" texels) by limiting the offset to be at most equal to current height value
  - Flattens geometry significantly at grazing angles
  - Just a heuristic

Demo

Parallax Mapping with Offset Limiting

Parallax Occlusion Mapping

- First ideas were introduced in ShaderX^3 in 2004 ("Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing", Z. Browley, N. Tatarchuk [Browley04])
- Current algorithm has several significant improvements over the earlier technique
- Designed to simulate the effects of per-pixel displacement mapping with correct motion parallax

Demo

Parallax Occlusion Mapping circa 2004

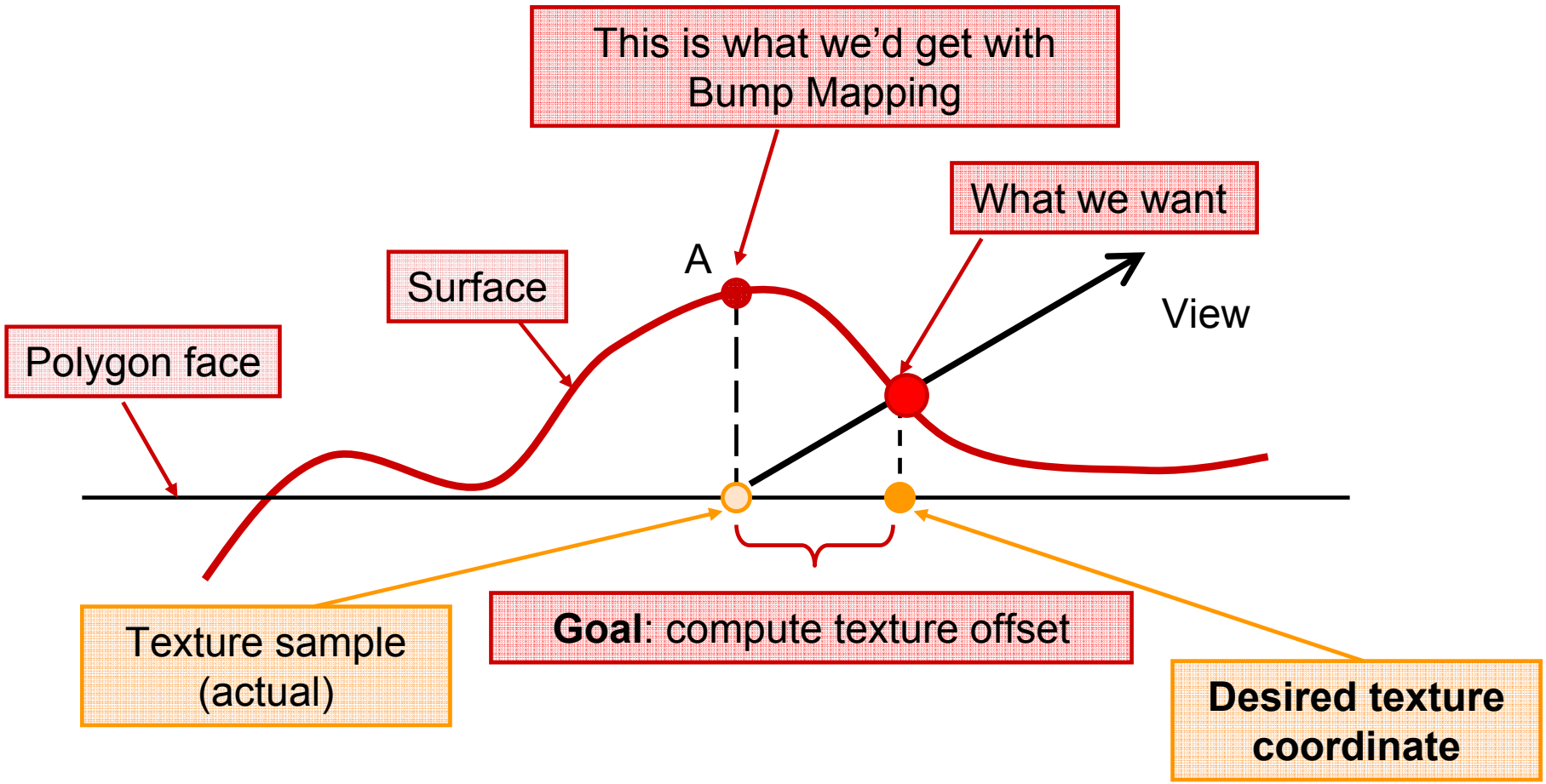
That's All Great, But...

- We've got even better stuff!!..
  - New algorithm developed by N. Tatarchuk (aka: me)
- And that's what we're going to focus on next:

POM Algorithm Overview

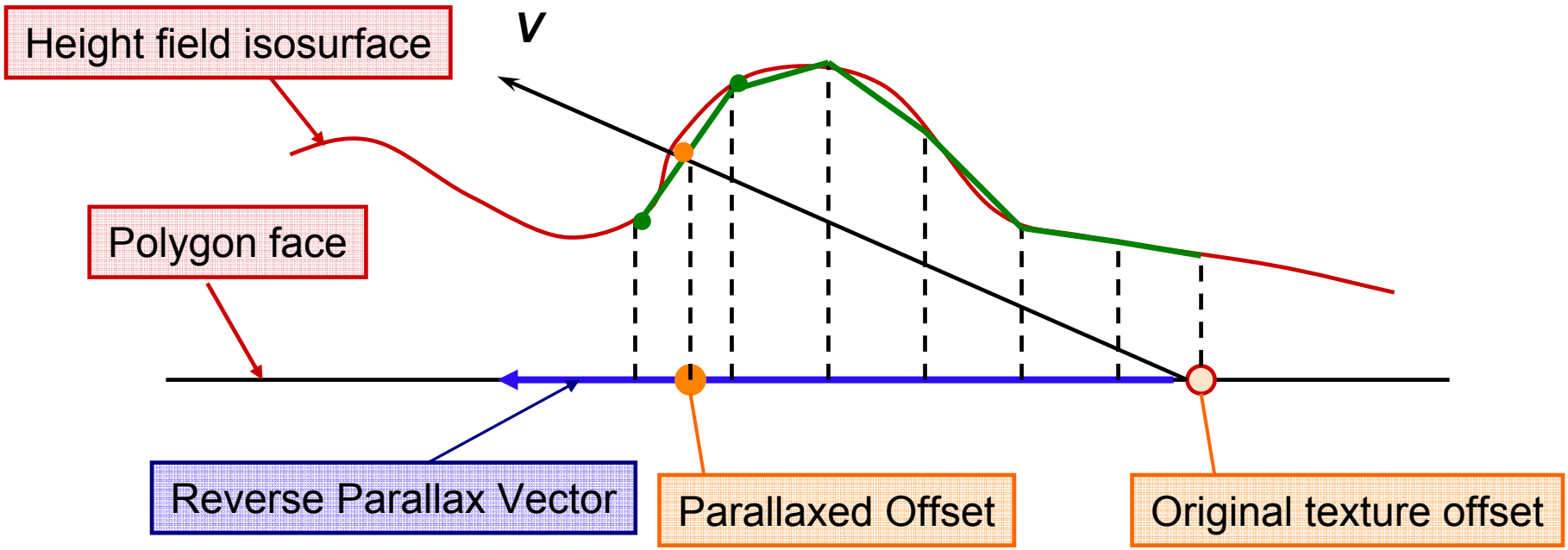
- Compute parallax offset vector
- Determine the intersection of the height map isosurface with the view ray in tangent space using the reverse parallax offset vector
  - This is called *Reverse Height Map Tracing*
  - Computes texture offset coordinates
- Determine visibility using the light vector in tangent space and currently visible point from above
- Compute lighting

Watch Those Texels Shift



Reverse Height Field Tracing

- Ray trace along the reverse parallax vector looking for an intersection with the height field
- Results in a texture offset corresponding to point of surface visible to the viewer adjusted for parallax



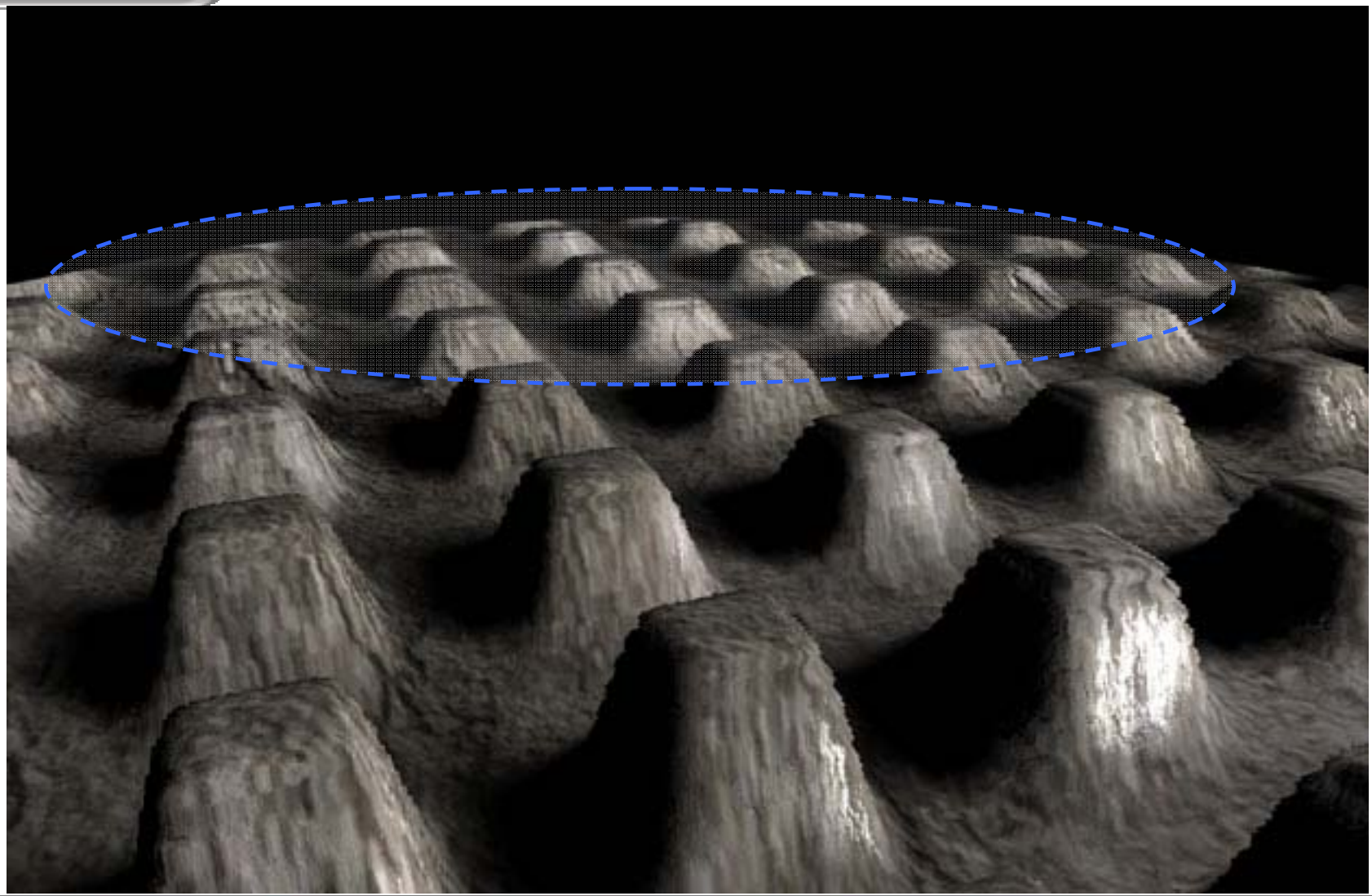
Those Darn Oblique Angles…

- Any sampling-based algorithm is prone to aliasing
- We control the sampling rate based on length of the parallax offset vector and surface / view angle relationship
  - View-dependent sampling rate for correct intersections at grazing angles
  - Do not sample past the length of the parallax vector
  - Use $N \bullet V$ to determine the optimal number of samples
    - This can be artist-controlled by setting min / max samples
    - Just a simple lerp instruction

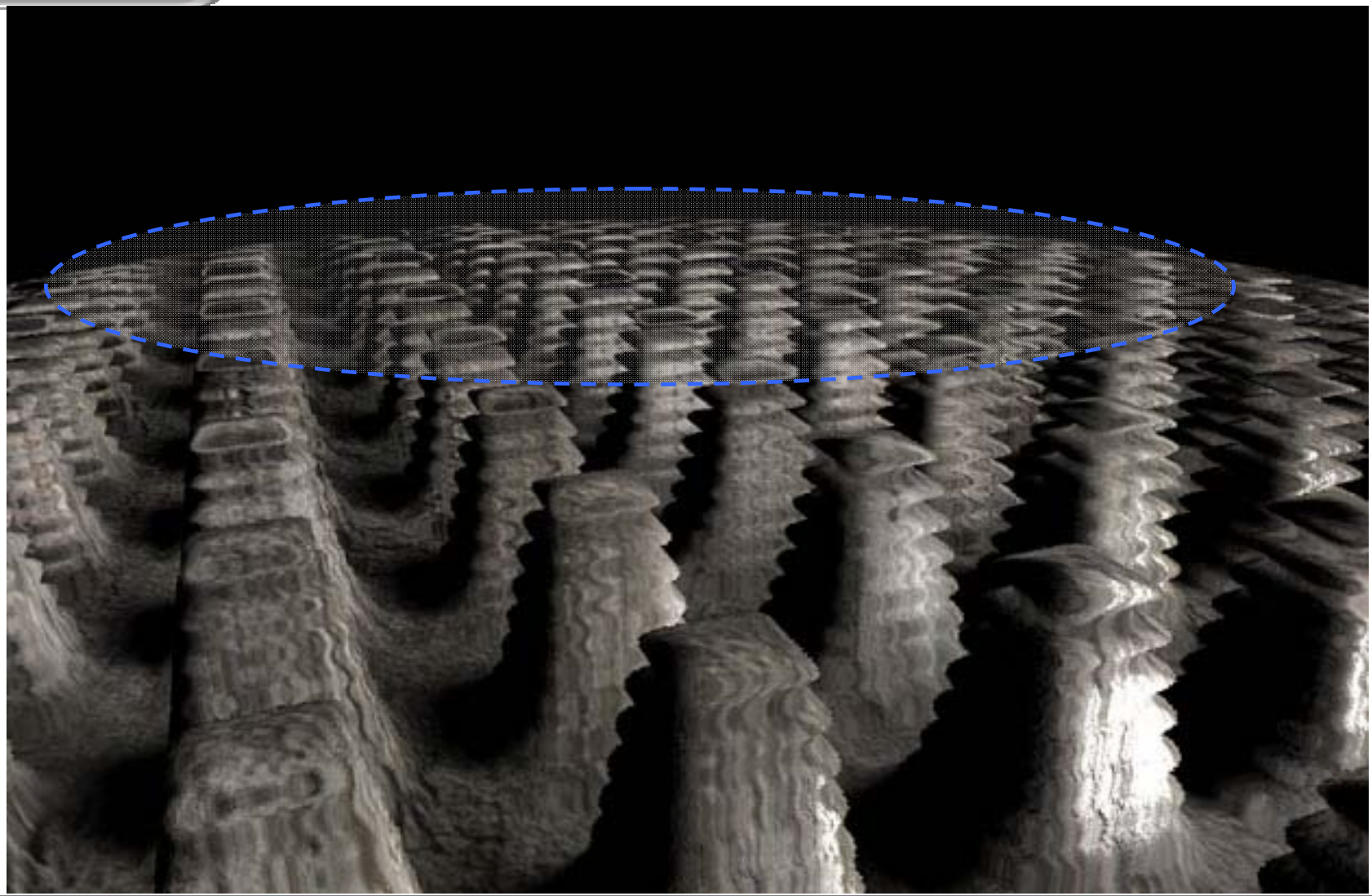
Or else!..

- Otherwise many missed intersections
  - Previous work [circa 2004] flattened bumps due to perspective biasing
  - Without perspective biasing, sampling planes were obvious
  - With view-dependent sampling, no flattening and shape is well-preserved

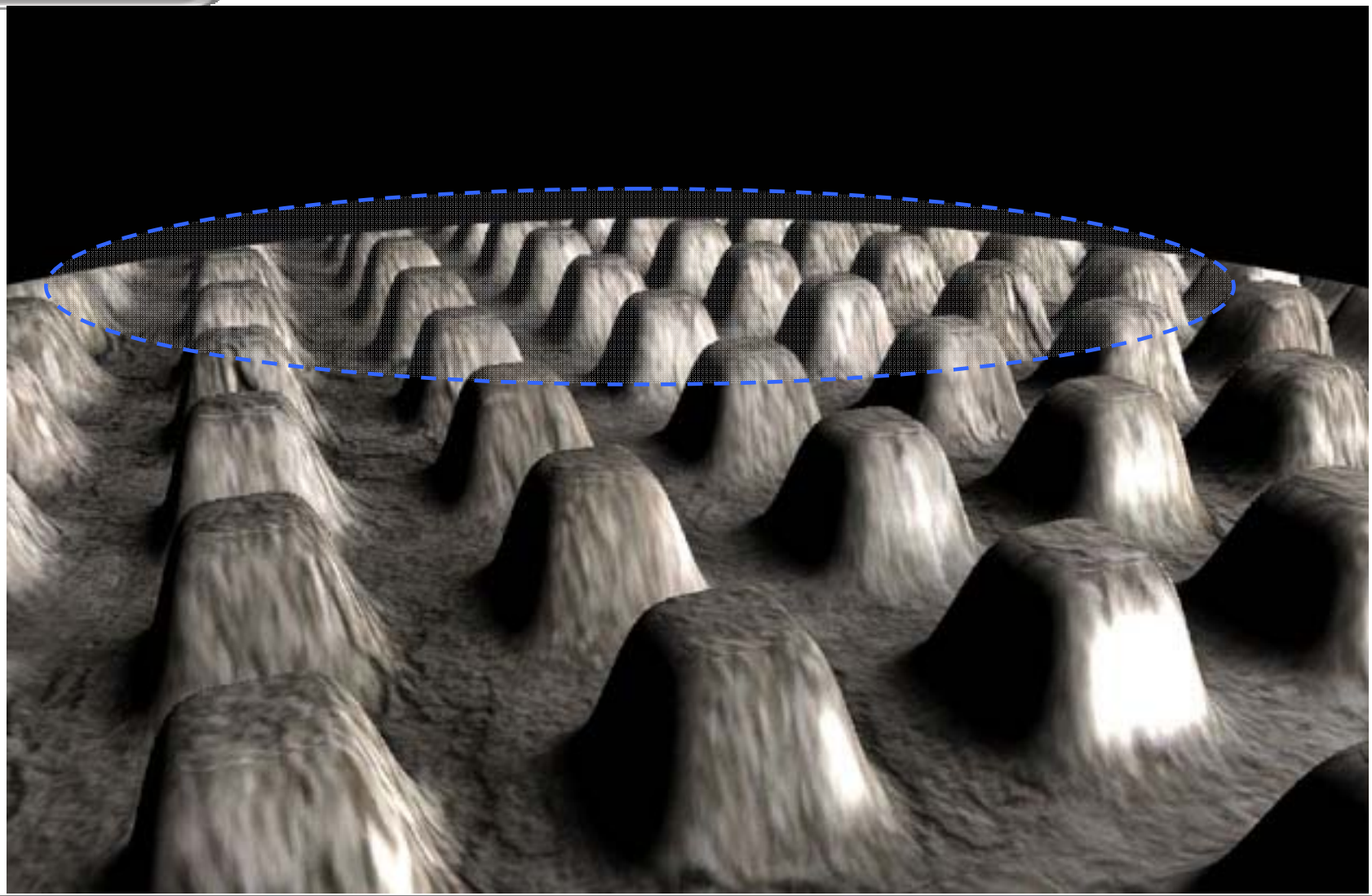
To Lerp or Not to Lerp: The Comparison: Perspective Bias



To Lerp or Not to Lerp: The Comparison: No Perspective Bias



To Lerp or Not to Lerp: The Comparison: The Right Way



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
  - 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

Is It Perfect?

- 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
    - Oblique angles pose problems for all similar approaches
- This approach does not yield correct silhouettes
  - Not the end of the world: Author art to hide the corner case
  - Not exactly displacement mapping
  - Could encode height field description for each texel and use this information to compute silhouettes
    - Prohibitive cost for games (Might as well use geometry in many cases)

Some Misbehaved Texels

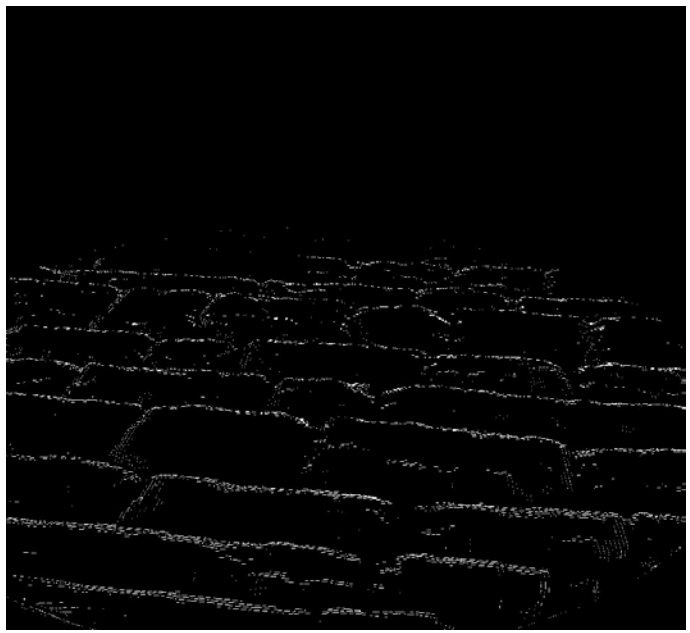
- Texture-space silhouette edges alias
  - Especially bad when viewed from further away and animated camera paths
  - Need to design the algorithm to correct this problem
- Hardware guys spend time doing better multisampling algorithms…. Is it all for nothing?
  - Doesn't quite apply here – we are not rendering polygon edges!
  - The aliasing occurs in texture space – the edges are all in your mind!

Correcting Aliasing Artifacts

- The texture-space edges occur because we are not performing ray-height map intersections at sub-pixel precision
  - The problem is very obvious in production quality demos
- Need to correct this for good results
  - Determine edges
  - Jitter rays and compute separate ray-height field intersections
  - Not enough to just combine the offsets – must compute full new intersections and lighting model for all offsets for good results

Determining Silhouette Edges

- Want to do this during the run of the algorithm and quickly
  - Not with post-processing techniques
- Use height gradient when found a ray-height field intersection
  - When previous step is a plateau and the next step exhibits large gradient the pixel is at the edge
  - Very good results
  - Also can use $N \bullet V$ to improve fidelity
    - At higher cost



Use The Power of 3.0 Shaders

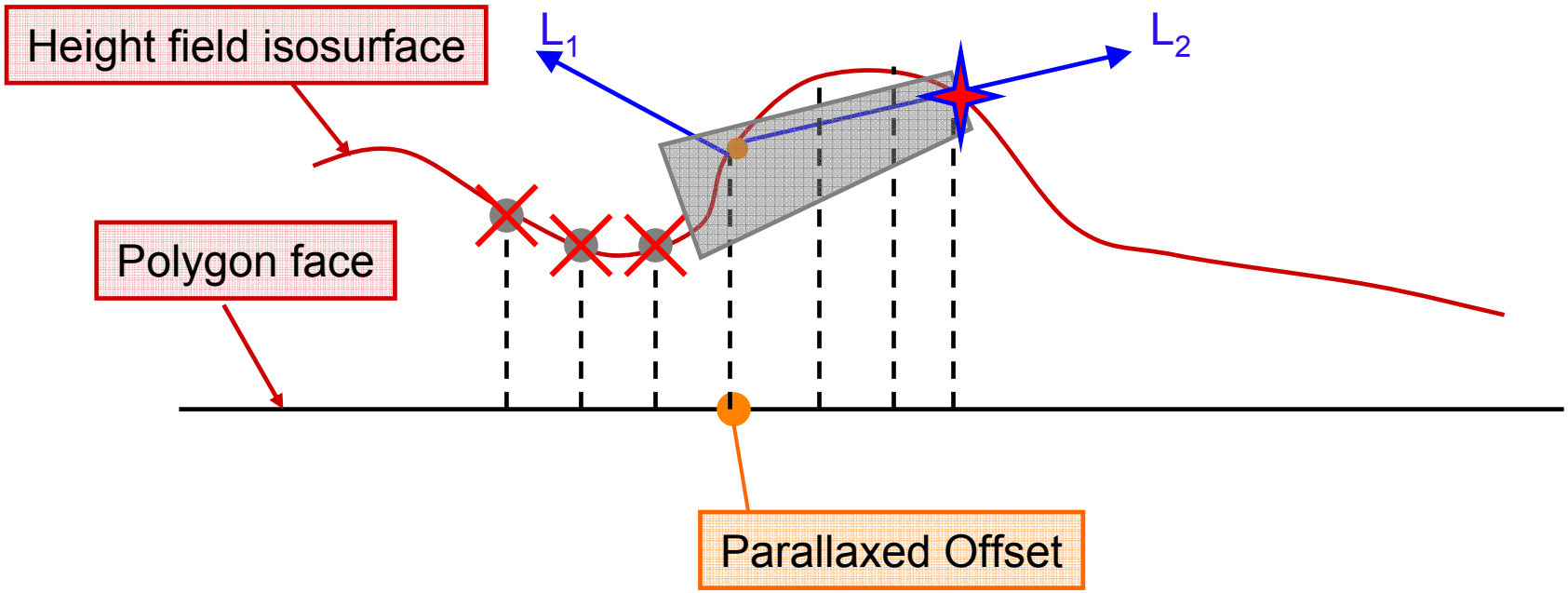
- We are performing additional antialiasing work for edge pixels only
- In a typical scene that means that only at most 10% of your pixels are going through heavier path
- Only possible with dynamic flow control
  - Otherwise you always pay the cost

Self-Occlusion and Shadows

- Apply the same concept as for reverse height map tracing
  - Use the light vector (in tangent space) to ray-trace from the newly found intersection offset to determine its visibility
- This works for self-occlusions of the surface
- Shadows are computed dynamically
  - Object correctly displays self shadowing effects
- This must be done for the POM-displaced point (new texture offset)
  - In order to compute correct visibility results

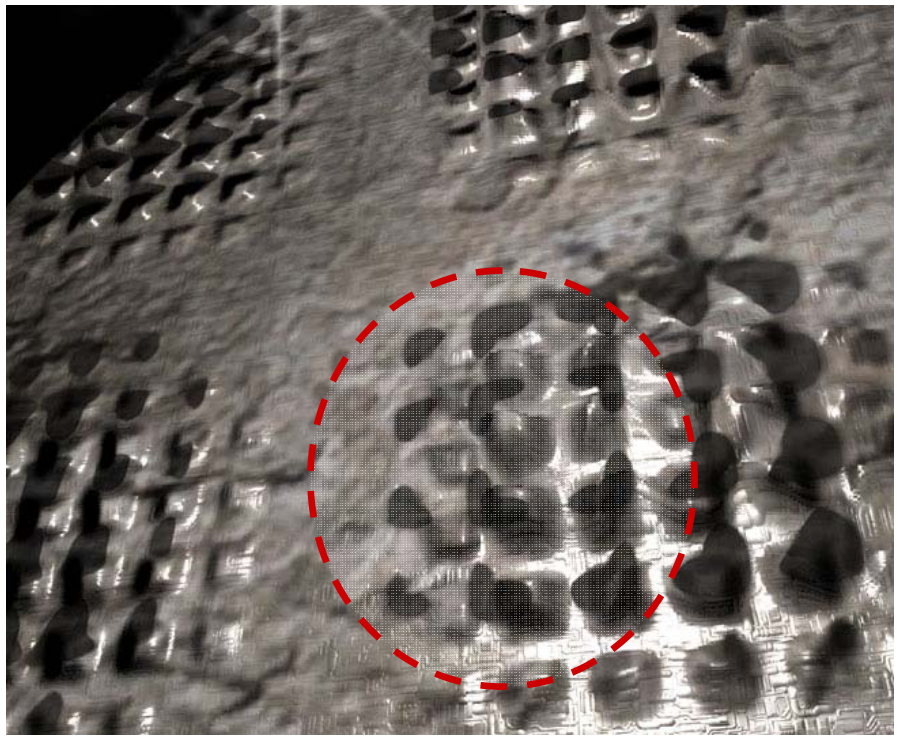
Computing Visibility

- Use the new parallaxed offset – which will be the visibility candidate
- Trace the light vector from that point and check for any occluders (Height field is your friend!)



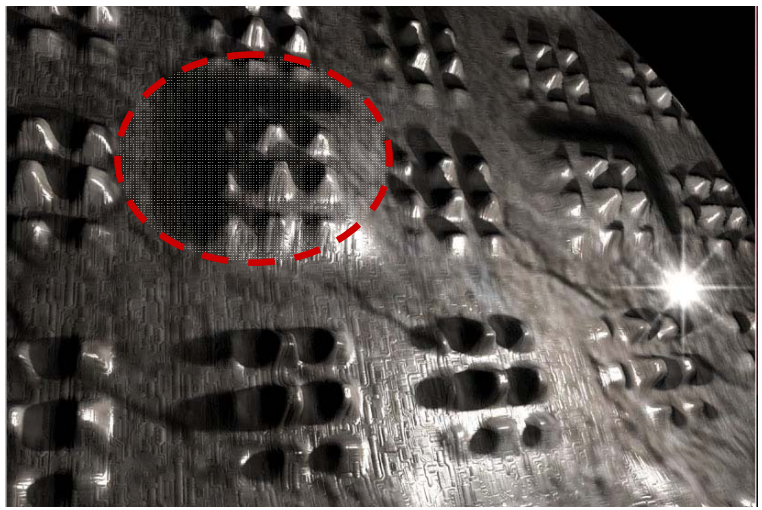
If the Start isn't Right?

- Important to use the parallaxed offset from height field computed earlier
  - Otherwise shadows will be floating in space



Soft Shadows

- Can create soft shadows by filtering the visibility samples as you compute them
- Reduces aliasing artifacts noticeable in many other "fake" displacement mapping approaches
- Creating soft shadows only adds a few additional ALU ops if you are already computing light visibility for current point
  - But looks significantly better!



Lighting the Displaced Pixels

- 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
- 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

POM: Great for Scalability

- Easy LOD scheme for textured surfaces
- The idea:
  - Compute the current mip map level in your shader
  - If rendering from far away, render the low res geometry with bump mapping (Based on LOD level)
  - Scale the number of samples during height field ray tracing as a function of the current mip map level
  - As you get closer to the surface, increase the number of samples

Additional Optimizations

- Use the length of the parallax offset vector to control the sampling rate for ray-tracing
  - Never sample past what will be visible
- View-dependent lerping aids depth fidelity
  - More samples at grazing angles – watch out
  - May want to restrict viewing angles if necessary

If You Use Post-Processing…

- May be cheaper to blur edges at post-process time than jitter rays and re-compute full lighting model
  - Render displaced texels to one buffer and output the edge map as a second buffer
  - During post-processing, blur the samples from the back buffer for edge pixels

You Can Render Dynamic Height Fields!

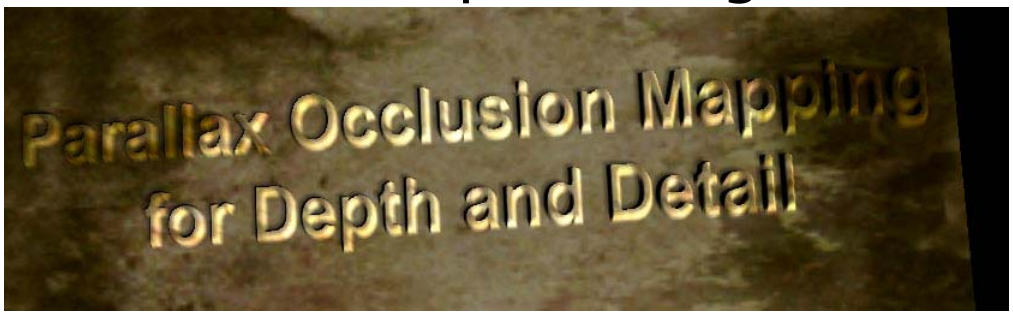
- Great advantage of this algorithm lies in ability to work with *dynamically rendered* height fields
  - Render to height fields texture dynamically
    - Water waves / procedurally generated objects / noise
  - Apply parallax occlusion mapping computations as regular computations
  - Compute the normals directly in the shader as necessary
- Approaches that use precomputed textures (ala distance maps, surface descriptions, etc) cannot support dynamic height field rendering in real-time
- Combine this algorithm with the computation of wave heights for interaction of parallax occlusion mapped objects with liquids

Does The World Have to Be Flat?

- Not at all! Columbus said so..
- Since the computation is in tangent space, we don't have the planarity restriction that many current approaches have
  - Works equally well on curved objects
  - Beware of silhouettes
    - See art considerations later on

What About Problematic Cases?

- Can handle traditionally hard displacement mapping cases – no restriction for low pass height fields
  - Raised text



  - Fine features

Not Just Low-Detail Height Maps!

- This means that the height field does not have to be low frequency (aka: low detail)
  - However, typically for high frequency (high details) fields the ray intersection routine will require higher sampling rate → higher cost
  - But still compute shadows and self-occlusion tests

The Juicy 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 really well
  - Doesn’t use dynamic number of iterations so the number of samples for height field tracing is constant
  - Can use more than one pass to sample height map at higher frequencies
    - 2-3 passes 8 samples each gives very good results
    - Makes oblique angles look better! 

PS_2_b Shader Details

- Single pass to compute the parallaxed offset, lighting and self-shadowing
- Uses static number of iterations to compute height field intersections
- Great performance
- Use as many samples as needed for your art / scene
  - Pay in form of instructions

Where is the Goodness…

- It's in **SM 3.0!**
- Uses dynamic flow control and early out during ray-tracing operations
  - Dynamic flow control is tricky – it helps to know what you're doing
  - Quick path to disassembly is your friend
- All of the important optimizations / quality improvements happen here
- Nicely balances ALU ops with control flow instructions and texture fetches
- 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% quicker!

How Is This Possible?

- But you thought it wasn't possible to do this at interactive rates
  - IT IS! *Good hardware architecture helps it*
- **Cheaper** than rendering same as geometry (remember - less polys)
  - Naturally, case by case basis
  - But – to achieve the same fidelity for some of the harder cases (text, fine features) we would need to have mesh resolution to match per-pixel displacement… COSTLY!
  - Shadow pass for geometry is expensive
  - Soft shadows – even more
- Ultimately – content and engine performance profile determine whether to use

Authoring Art for POM

- 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

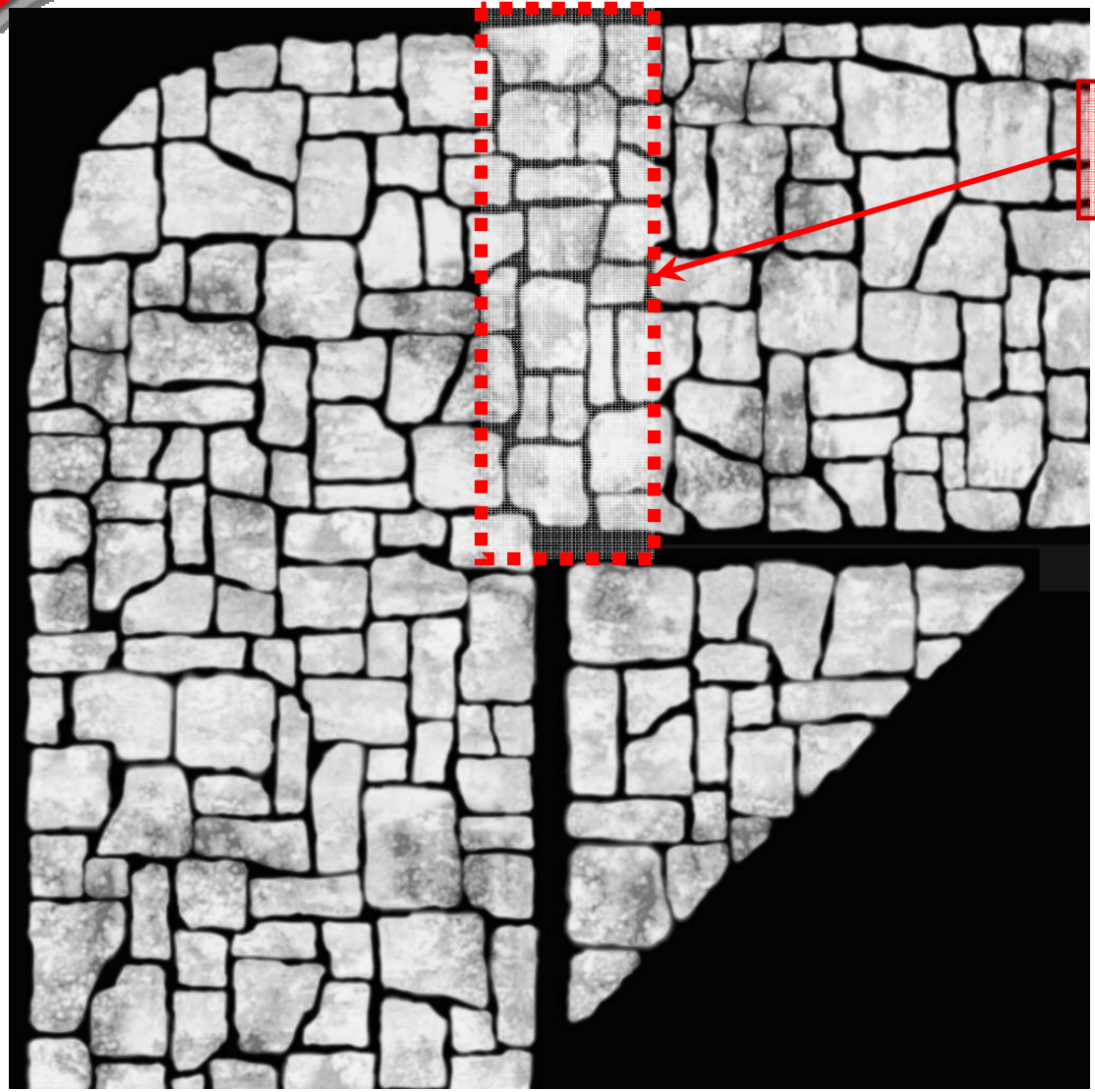
When Using Non-contiguous UVs

- Some complicated objects may require the use of non-contiguous UVs
- If not careful, this may result in a visible seam at run time

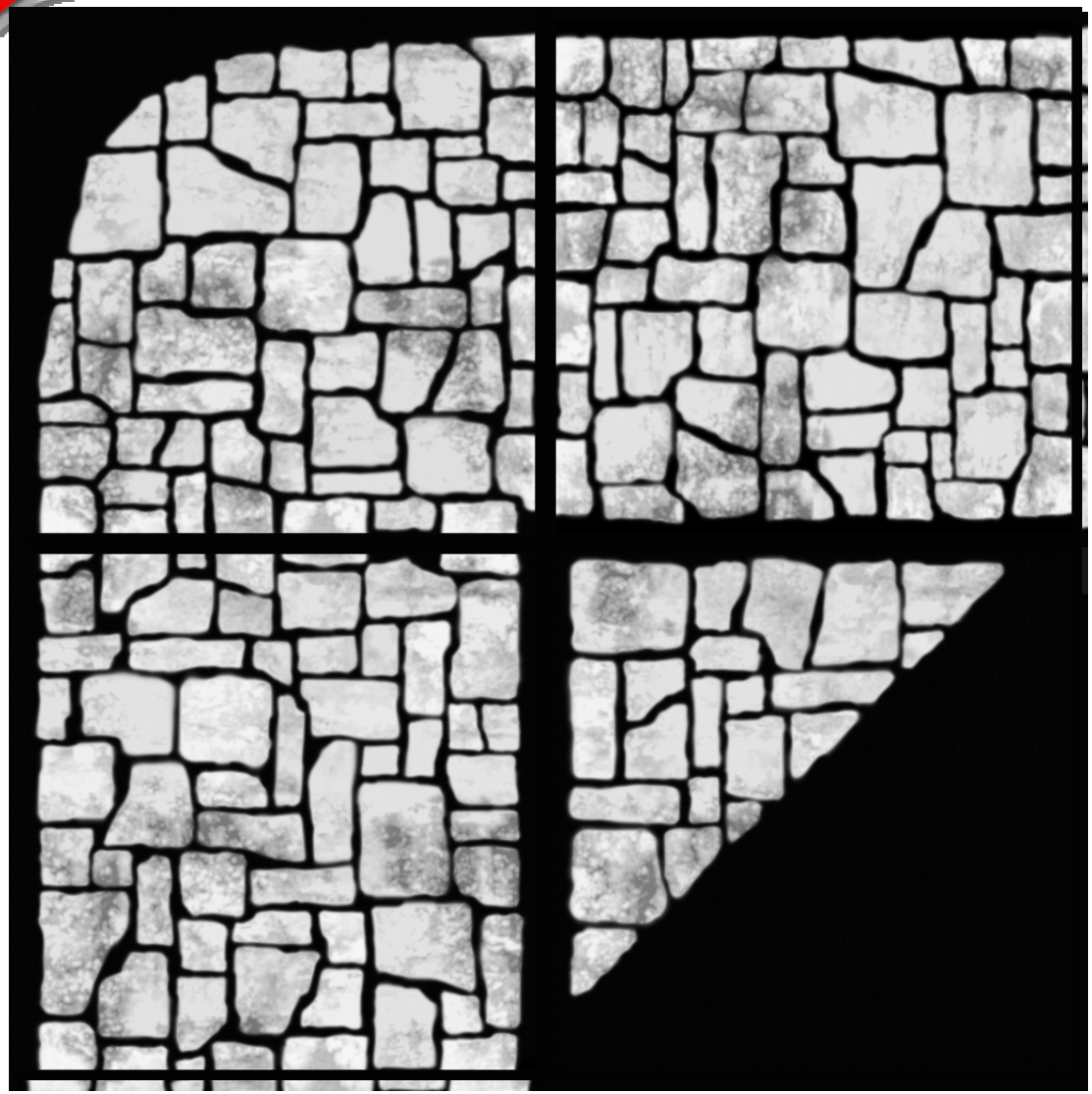
Removing the Seam

- This is an easily solvable problem – when authoring texture maps, add padding of ~10-20 pixels around the border for intersecting regions
  - Important to do this process for *all* texture assets used to render this object (Normal maps / albedo / detail, etc)

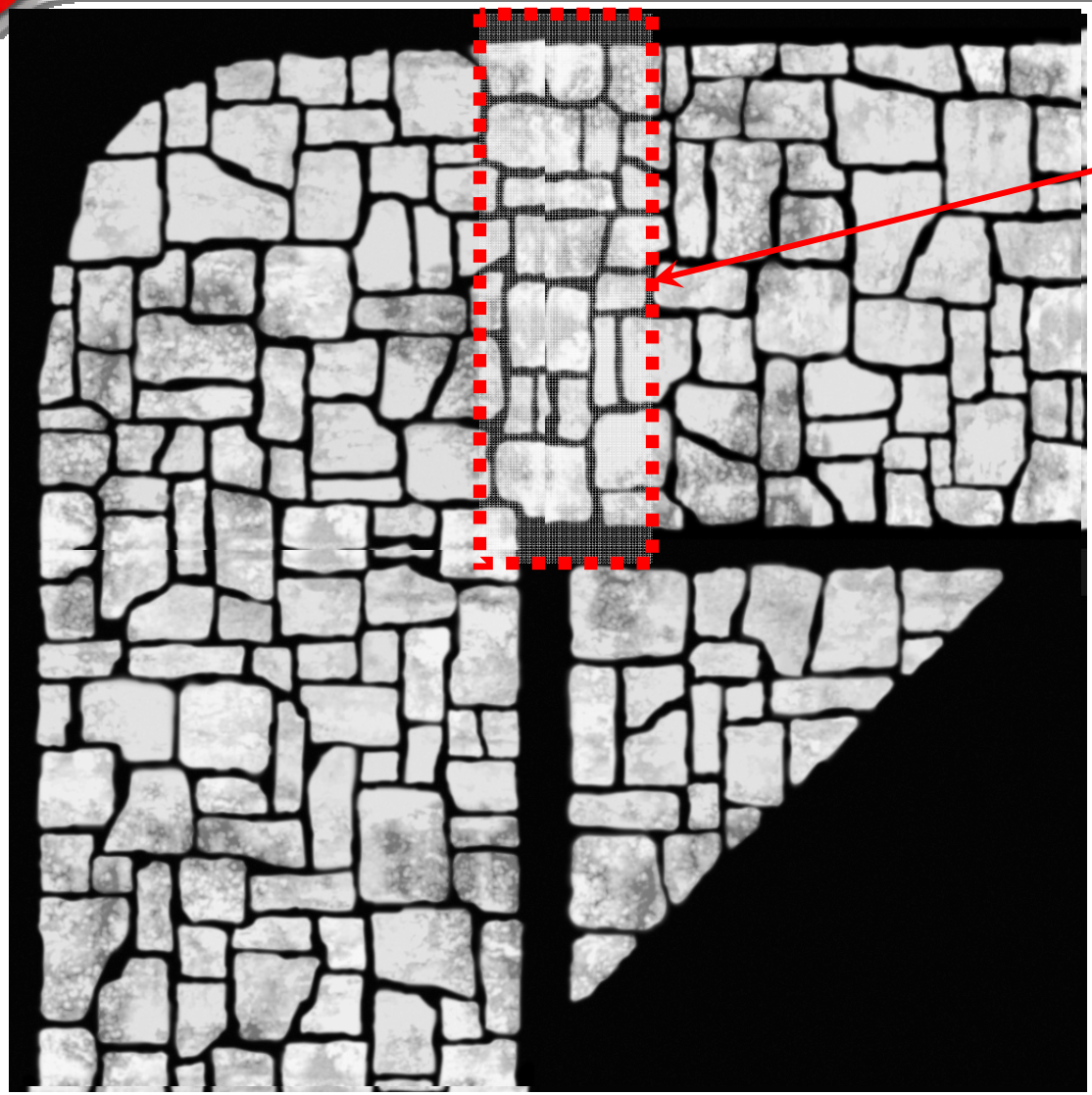
Padding the Maps: Original Art



Padding the Maps: Intermediate Step

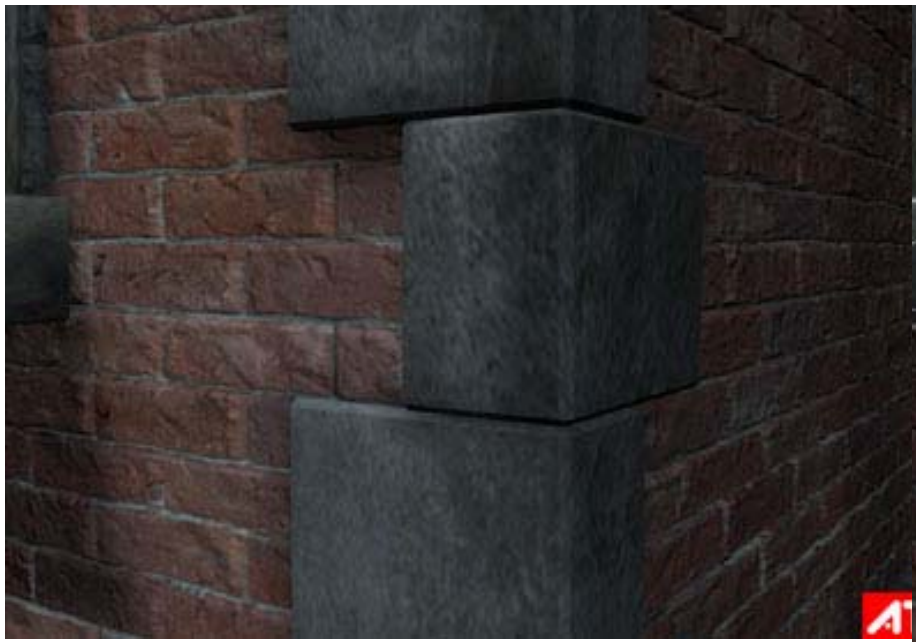


Padding the Maps: Final Art



What About Those Silhouettes?

- Can hide smooth silhouettes with special silhouette geometry
  - A row of bricks on the corners of the building
  - Curb of the sidewalk



A Few More Hints

- For very detailed, complex surfaces, height map / normal map textures need to have enough resolution

- If computing specular term use gloss maps to reduce specularity in the valleys

Demo

Parallax Occlusion Mapping

Uses and Future Ideas

- Everybody gets bumpy tonight!
  - Bumpy surfaces galore - Bricks, cobble stones, manhole covers, grates, you name it!
- Water waves animation
- Water moving over occlusion-mapped objects
- Bullet holes animation
  - Animating POMapped objects is not a problem, in fact works very well
- Many many more…

Conclusions

- *Parallax Occlusion Mapping* is a powerful technique for rendering complex surface details in real time
  - Image-space algorithm
- Has modest texture memory footprint
  - Normal map and height map (tile-able)
- Produces excellent lighting results
  - Self-shadowing for self-occlusion in real-time
  - Soft dynamic shadows
  - Flexible lighting model
  - Antialiased texture-space edges
- Great LOD rendering technique for textured scenes
- Easily supports dynamic rendering of height fields
- Efficiently uses existing pixel pipelines for highly interactive rendering
  - Well-designed for future hardware architectures

References

- [Blinn78] Blinn, James F. "*Simulation of wrinkled surfaces*", Siggraph '78
- [Max88] N. Max "*Horizon Mapping: shadows for bump-mapped surfaces",* The Visual Computer 1988
- [Sloan00] P-P. Sloan, M. Cohen, "*Interactive Horizon Mapping*", Eurographics 2000
- [Welsh04] T. Welsh, "*Parallax Mapping with Offset Limiting: A Per Pixel Approximation of Uneven Surfaces*", 2004
- [Wang03] L. Wang *et al*, "View-Dependent Displacement Mapping", Siggraph 2003
- [Doggett00] M. Doggett, J. Hirche, *"Adaptive View Dependent Tessellation of Displacement Maps",* Eurographics Hardware Workshop 2000
- [Kaneko01] Kaneko *et al*., "Detailed Shape Representation with Parallax Mapping", *ICAT 2001*

More References

- [Brawley04], Z. Brawley, N. Tatarchuk, "*Parallax Occlusion Mapping: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing*", ShaderX^3, 2004
- [Oliveira00] M. Oliveira et al, "*Relief Texture Mapping*", Siggraph 2000
- [Yerex04] K. Yerex, M. Jagersand, "*Displacement Mapping with Ray-casting in Hardware",* Siggraph 2004 Sketches
- And some very latest references
  - W. Donnelly, "*Per-Pixel Displacement Mapping with Distance Functions",* GPU Gems2, 2005

Acknowledgements

> Zoe Brawley, Relic Entertainment

> 3D Application Research Group