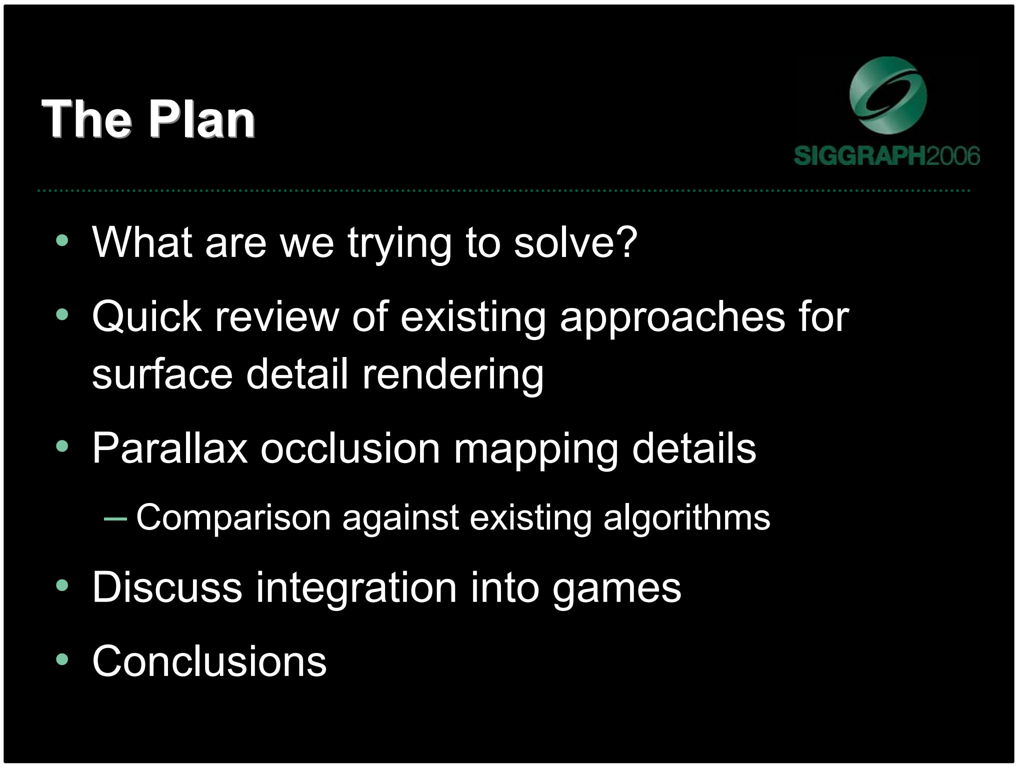
SIGGRAPH2006

SIGGRAPH2006

Practical Parallax Occlusion Mapping For Highly Detailed Surface Rendering

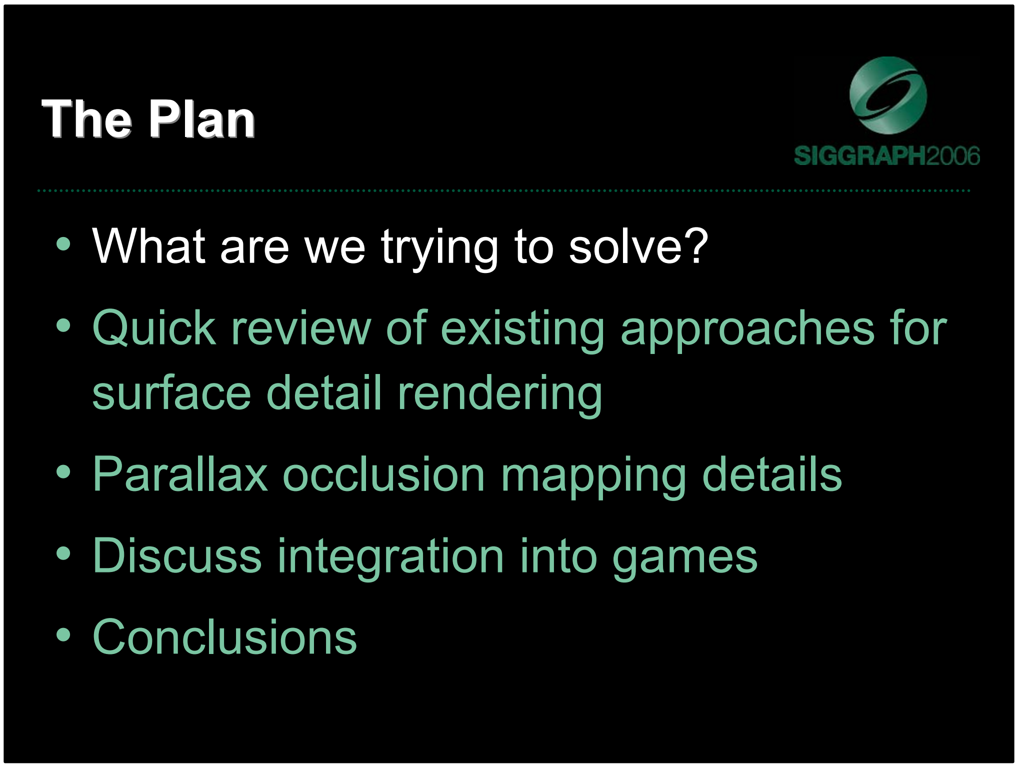
Natalya Tatarchuk

3D Application Research Group
ATI Research, Inc.



The Plan

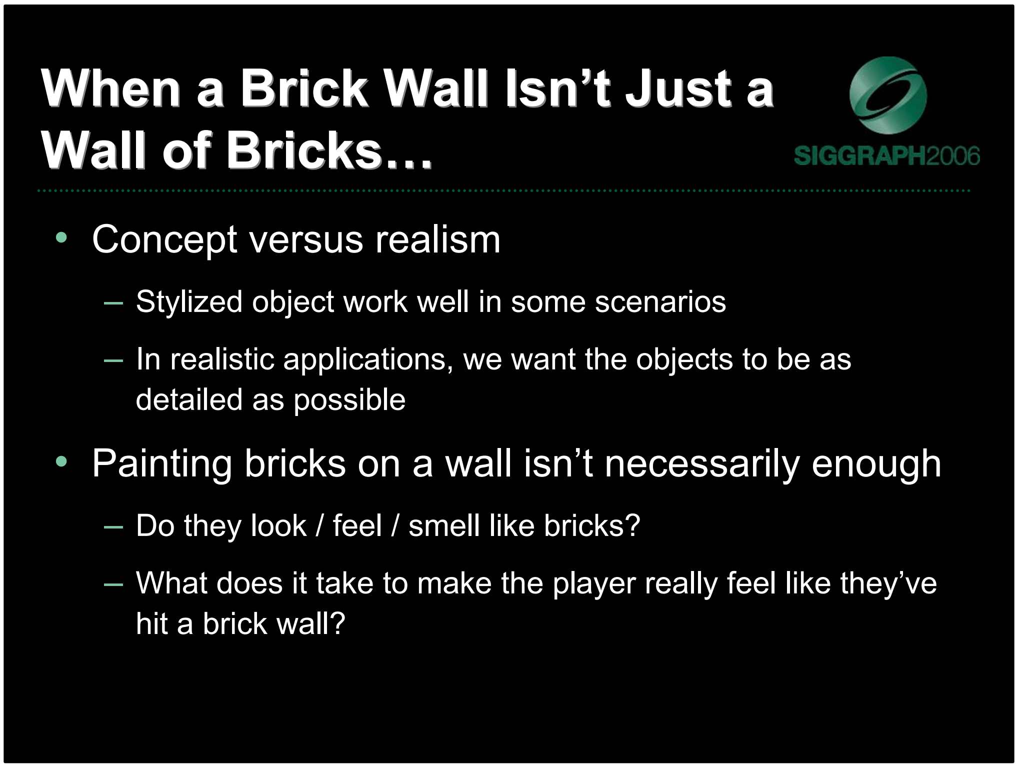
- What are we trying to solve?
- Quick review of existing approaches for surface detail rendering
- Parallax occlusion mapping details
  - Comparison against existing algorithms
- Discuss integration into games
- Conclusions



The Plan

SIGGRAPH2006

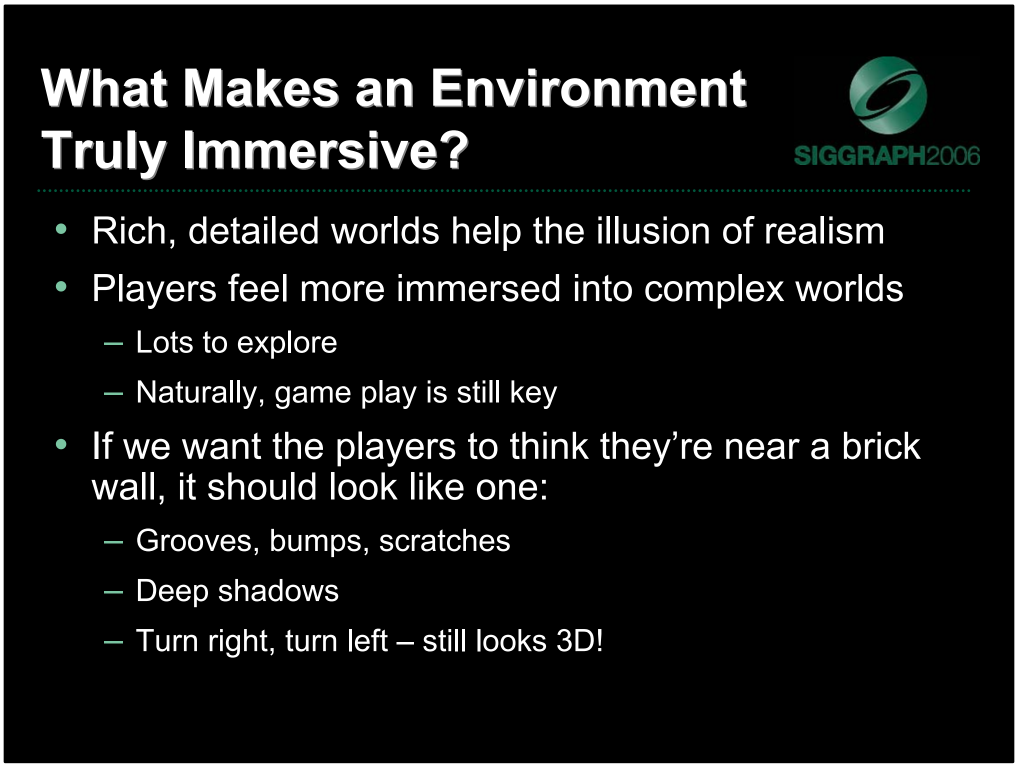
- What are we trying to solve?
- Quick review of existing approaches for surface detail rendering
- Parallax occlusion mapping details
- Discuss integration into games
- Conclusions



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

SIGGRAPH2006

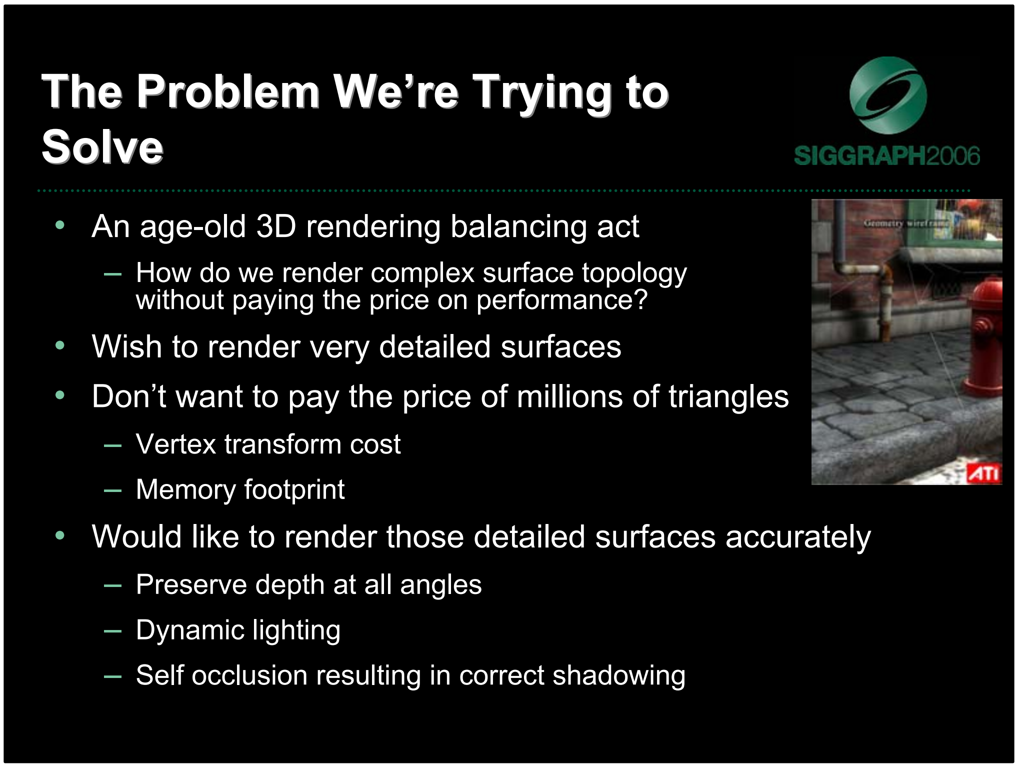
- Concept versus realism
  - Stylized object work well in some scenarios
  - In realistic applications, we want the objects to be as detailed as possible
- Painting bricks on a wall isn't necessarily enough
  - Do they look / feel / smell like bricks?
  - What does it take to make the player really feel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



What Makes an Environment Truly Immersive?

SIGGRAPH2006

- 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!

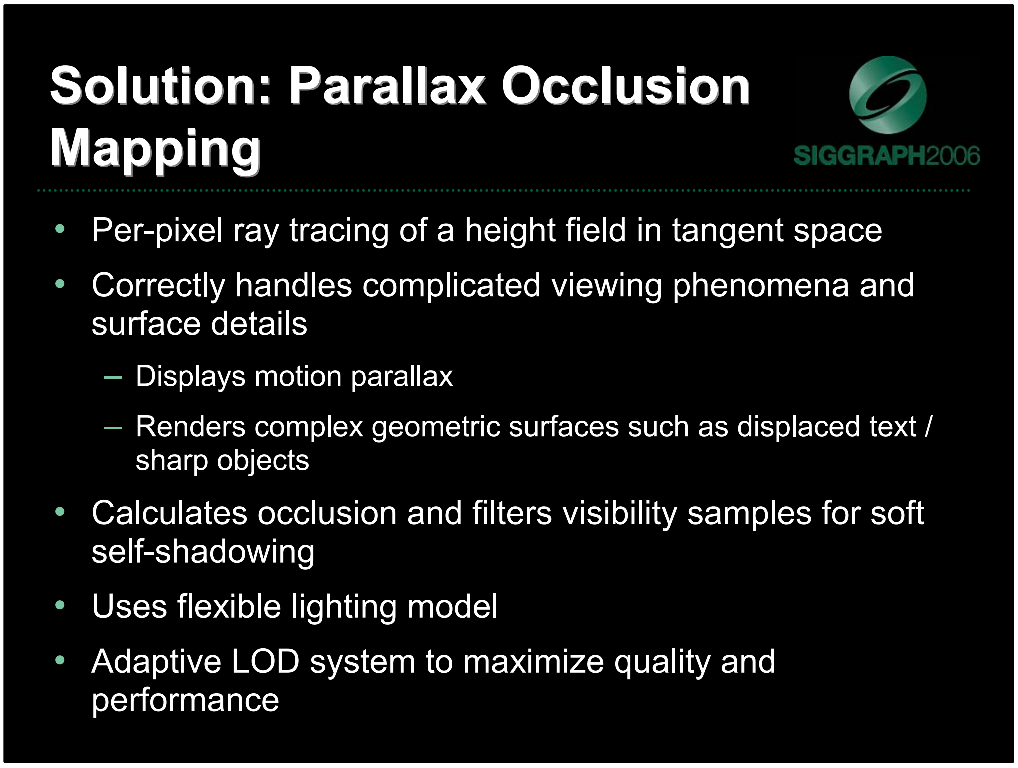


The Problem We're Trying to Solve

SIGGRAPH2006

- 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

How do we render detailed surface topology without paying the price on perf?



Solution: Parallax Occlusion Mapping

SIGGRAPH2006

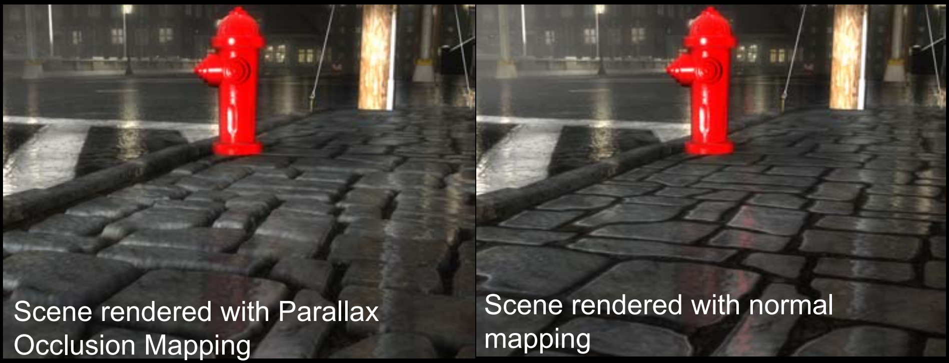
- Per-pixel ray tracing of a height field in tangent space
- Correctly handles complicated viewing phenomena and surface details
  - Displays motion parallax
  - Renders complex geometric surfaces such as displaced text / sharp objects
- Calculates occlusion and filters visibility samples for soft self-shadowing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

The effect of motion parallax for a surface can be computed by applying a height map and offsetting each pixel in the height map using the geometric normal and the eye vector. As we move the geometry away from its original position using that ray, the parallax is obtained by the fact that the highest points on the height map would move the farthest along that ray and the lower extremes would not appear to be moving at all. To obtain satisfactory results for true perspective simulation, one would need to displace every pixel in the height map using the view ray and the geometric normal.

Essentially its a simulated displacement mapping technique that occurs in texture space. Displaces "down" - existing surface has a height value of 1. It is a sampling algorithm – think of it as rendering height slices. Accurately approximates parallax as viewing angle changes. Properly self shadows – "soft" shadows. Integrates with all commonly used lighting models.

Parallax Occlusion Mapping versus Normal Mapping

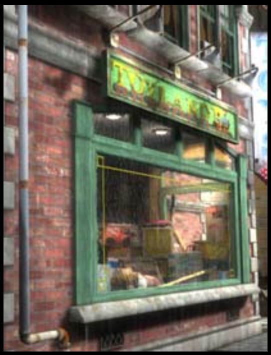
SIGGRAPH2006


Scene rendered with Parallax Occlusion Mapping
Scene rendered with normal mapping

Realistic city scene rendered using parallax occlusion mapping applied to the cobblestone sidewalk in (left) and using the normal mapping technique in (right).

Surface Details in the ToyShop Demo

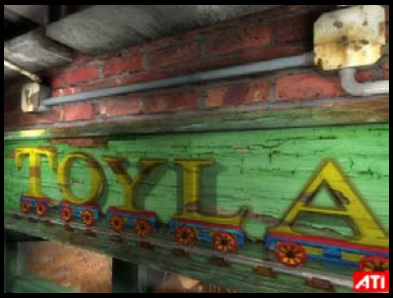
- Parallax occlusion mapping was used to render extreme high details for various surfaces in the demo
  - Brick buildings



Surface Details in the ToyShop Demo

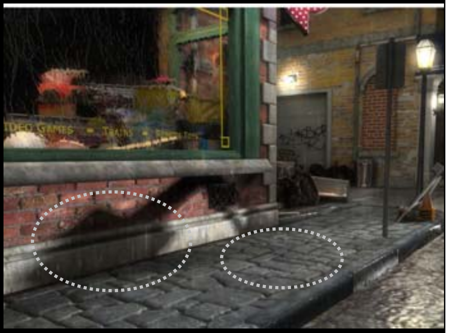
SIGGRAPH2006

- 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

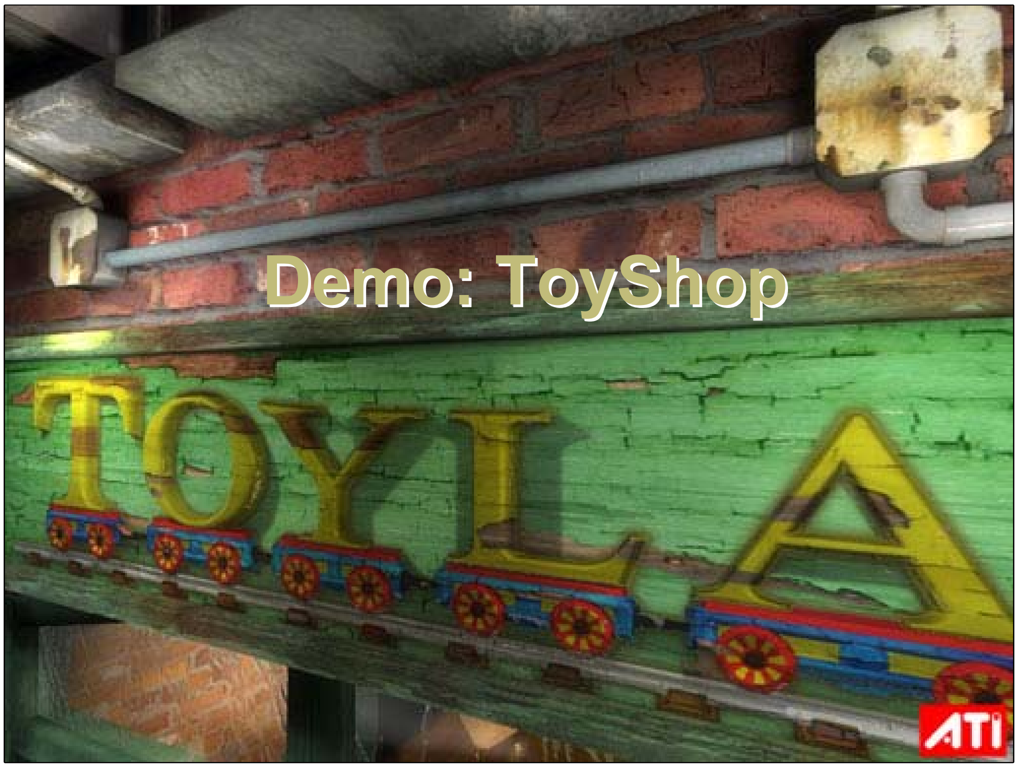


Surface Details in the ToyShop Demo

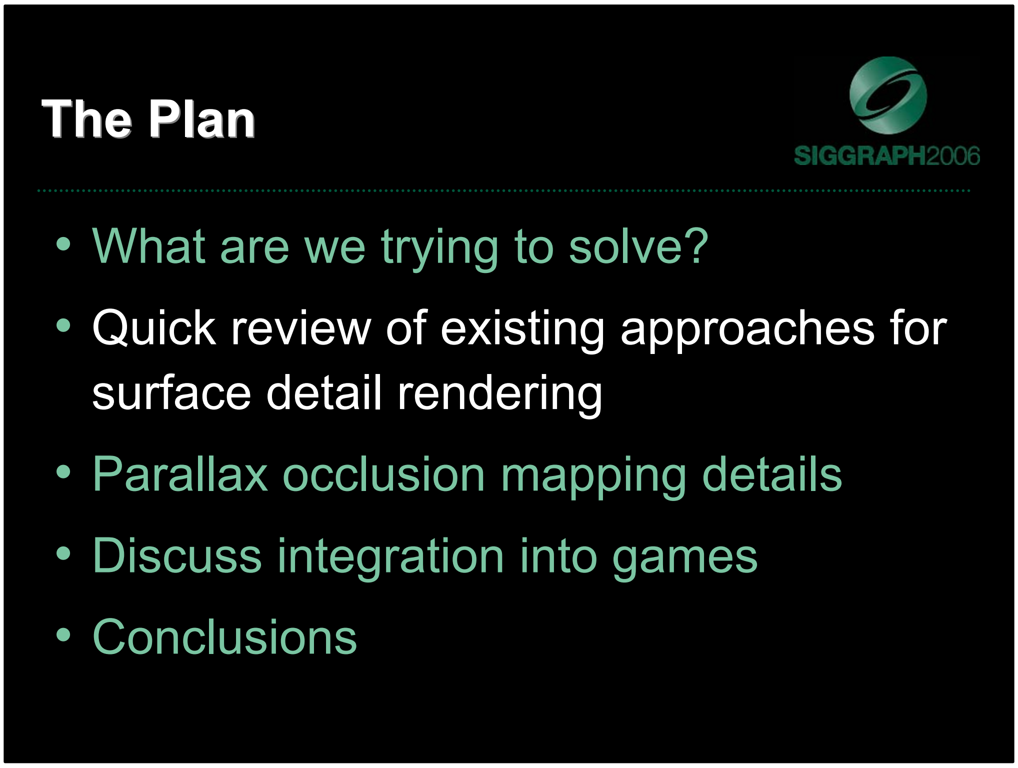
- Parallax occlusion mapping was used to render extreme high details for various surfaces in the demo
  - Brick buildings
  - Wood-block letters for the toy shop sign
  - Cobblestone sidewalk



- Using 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



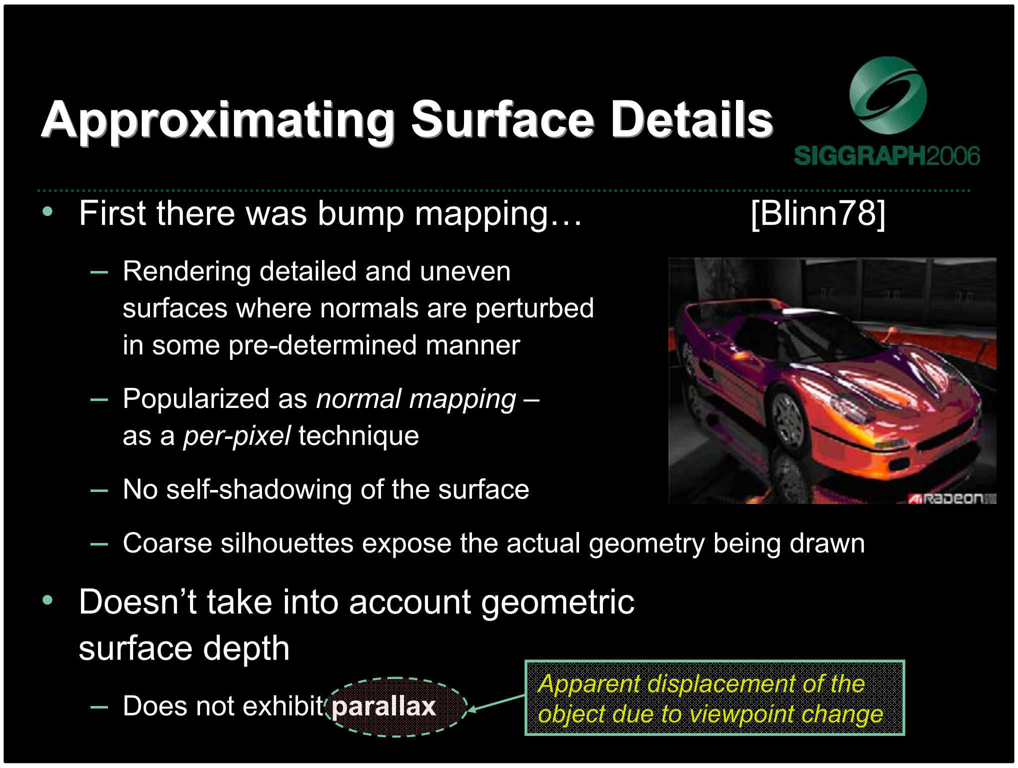
Demo: ToyShop



The Plan

SIGGRAPH2006

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



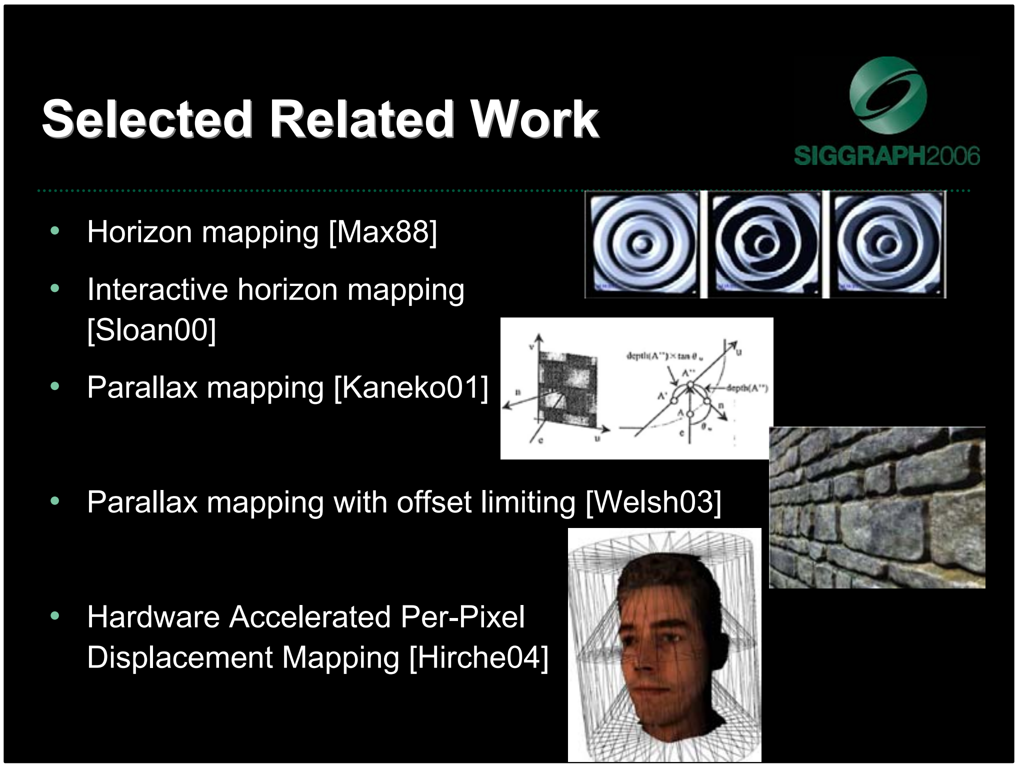
Approximating Surface Details

SIGGRAPH2006

- 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

The surface should appear to move correctly with respect to the viewer



Selected Related Work

SIGGRAPH2006

- Horizon mapping [Max88]
- Interactive horizon mapping [Sloan00]
- Parallax mapping [Kaneko01]
- Parallax mapping with offset limiting [Welsh03]
- Hardware Accelerated Per-Pixel Displacement Mapping [Hirche04]

We would like to generate the feeling of motion parallax while rendering detailed surfaces. Recently many approaches appeared to solve this for rendering. Parallax Mapping was introduced by Kaneko in 2001 and popularized by Welsh in 2003 with offset limiting technique

Parallax mapping

- 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
- 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)

Horizon Mapping:

- 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: additional texture memory

Parallax Mapping with Offset Limiting

- Same idea as in [Kaneko01]
- 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)

The Plan

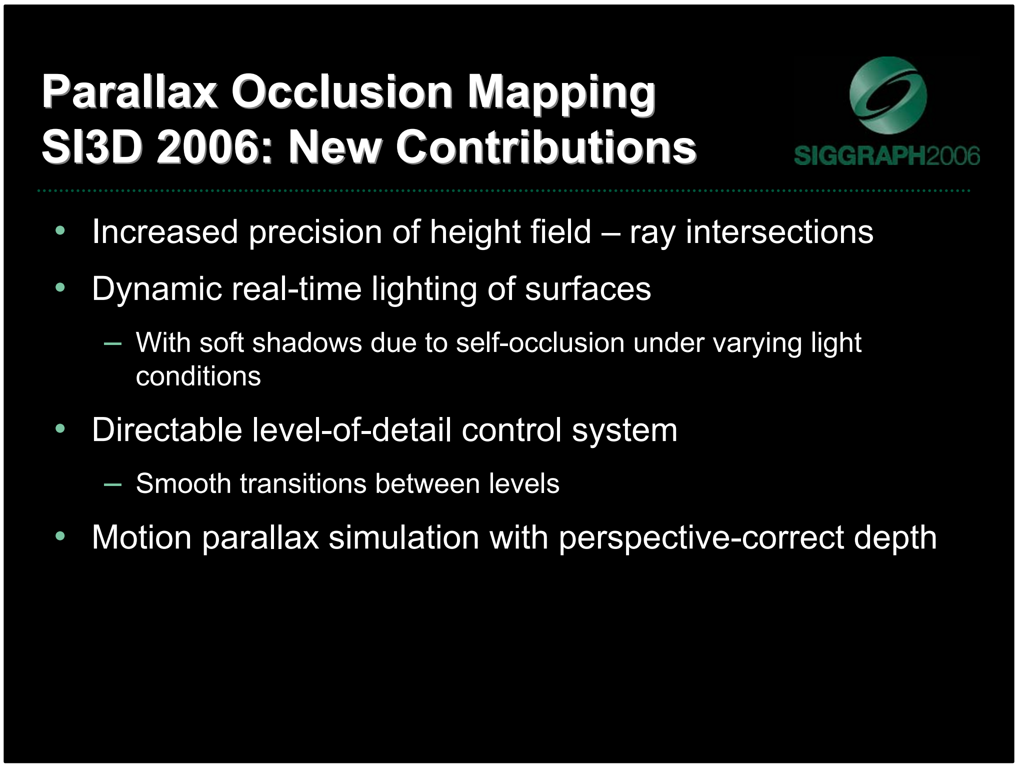
SIGGRAPH2006

- 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

SIGGRAPH2006

- 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

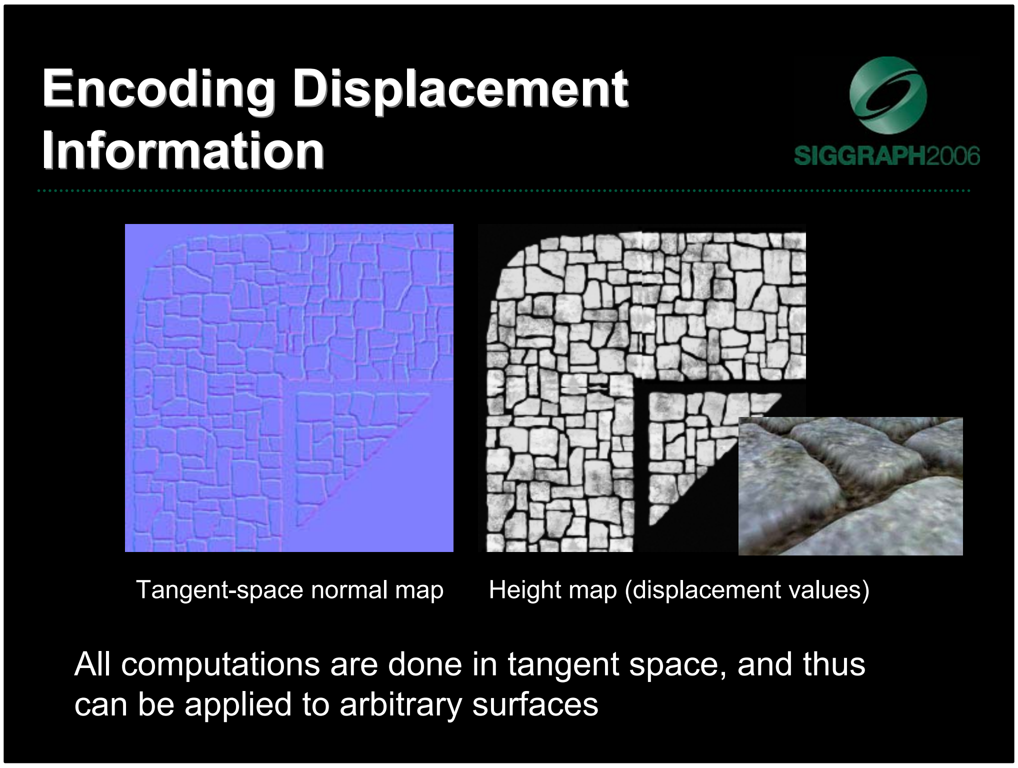


Parallax Occlusion Mapping SI3D 2006: New Contributions

SIGGRAPH2006

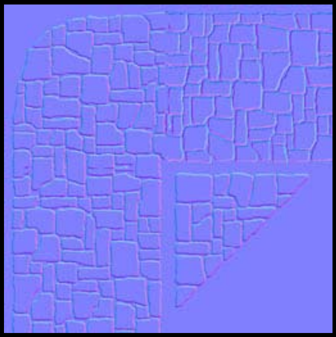
- Increased precision of height field – ray intersections
- Dynamic real-time lighting of surfaces
  - With soft shadows due to self-occlusion under varying light conditions
- Directable level-of-detail control system
  - Smooth transitions between levels
- Motion parallax simulation with perspective-correct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

Our technique can be applied to animated objects and fits well within established art pipelines of games and effects rendering. The implementation makes effective use of current GPU pixel pipelines and texturing hardware for interactive rendering. The algorithm allows scalability for a range of existing GPU products.

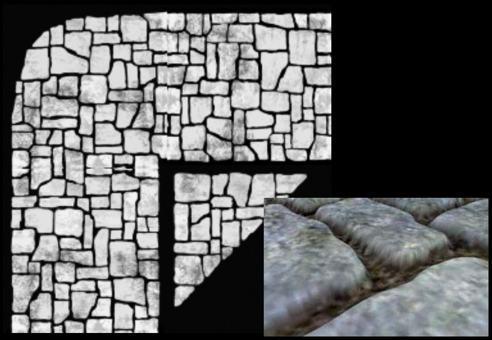


Encoding Displacement Information

SIGGRAPH2006



Tangent-space normal map



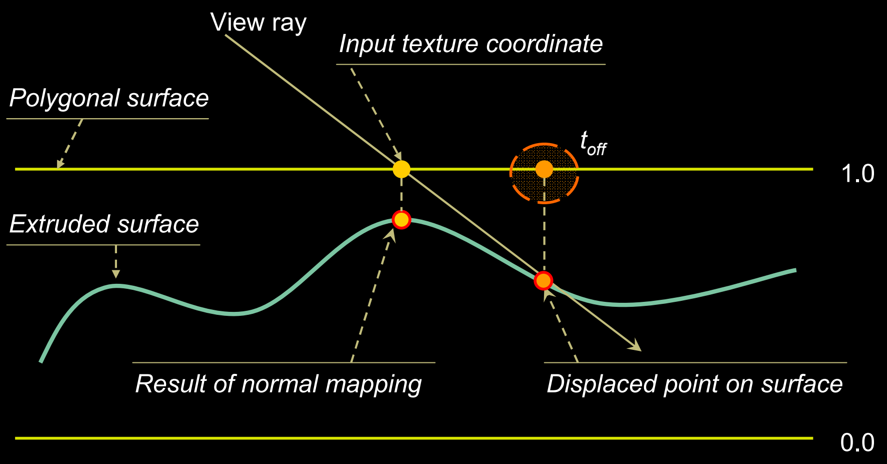
Height map (displacement values)

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

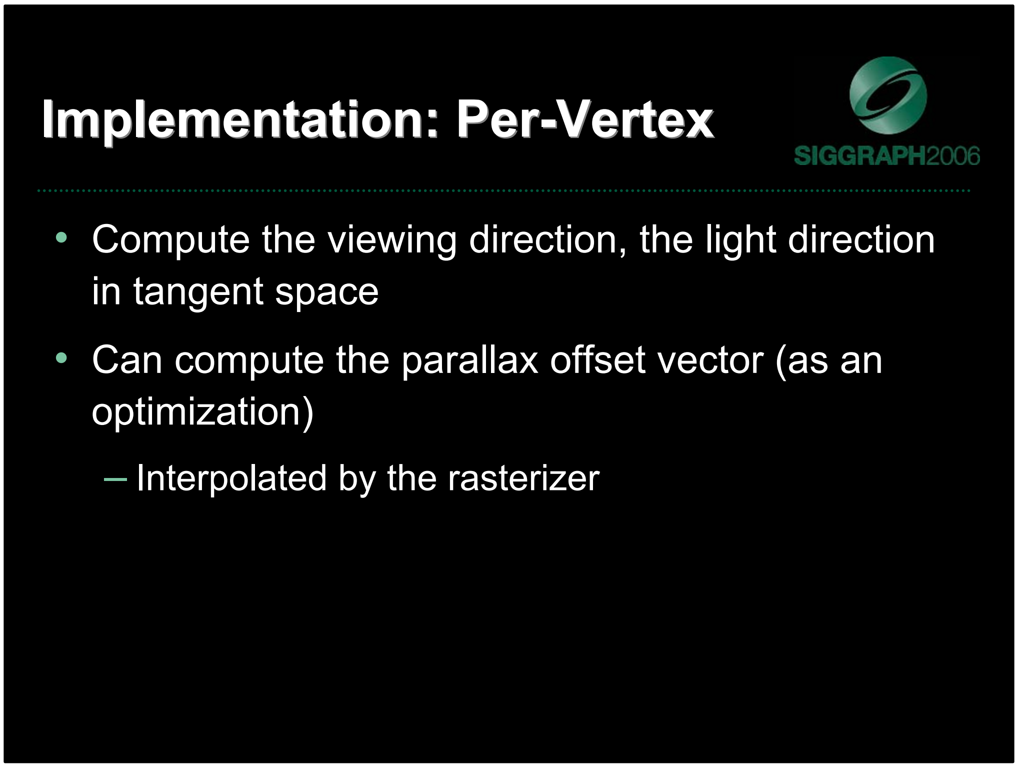
We encode surface displacement information in a tangent-space normal map accompanied by a scalar height map. Since tangent space is inherently locally planar for any point on an arbitrary surface, regardless of its curvature, it provides an intuitive mapping for surface detail information. We perform all calculations for height field intersection and visibility determination in tangent space, and compute the illumination in the same domain.

Parallax Displacement

SIGGRAPH2006



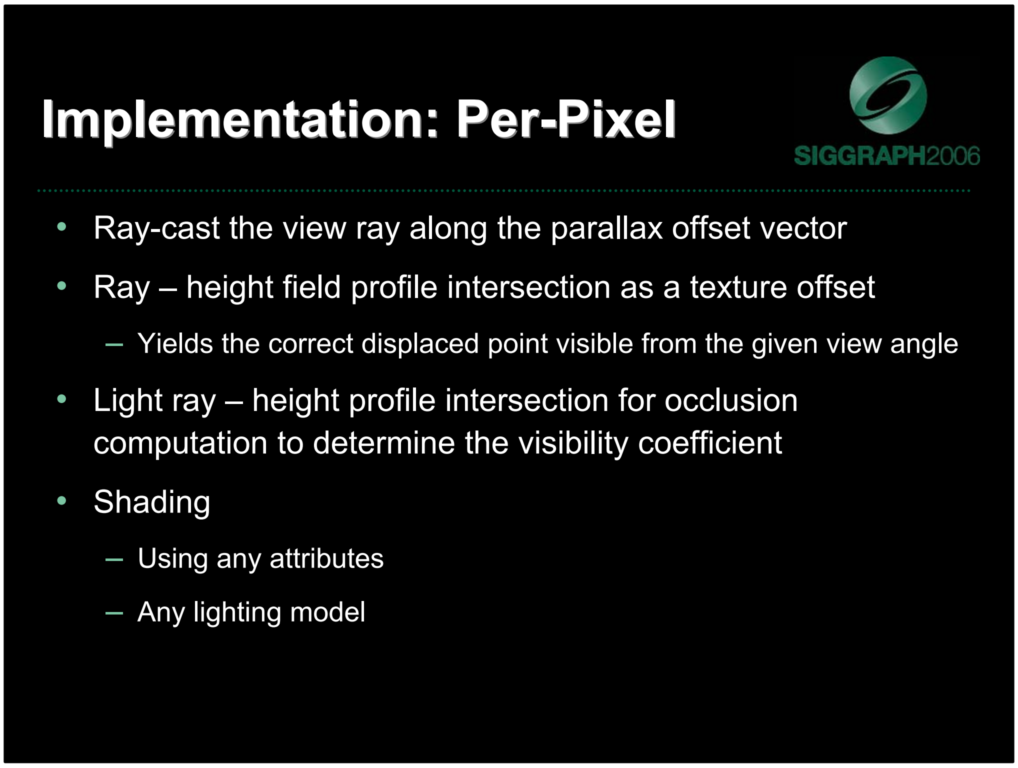
The effect of motion parallax for a surface can be computed by applying a height map and offsetting each pixel in the height map using the geometric normal and the view vector. We trace a ray through the height field to find the closest visible point on the surface. The core idea of the presented algorithm is to trace the pixel being currently rendered in reverse in the height map to determine which texel in the height map would yield the rendered pixel location if in fact we would have been using the actual displaced geometry. The input mesh provides the reference plane for displacing the surface downwards. The height field is normalized for correct ray-height field intersection computation (0 representing the reference polygon surface values and 1 representing the extrusion valleys).



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

Compute the parallax offset vector *P* to determine maximum visual offset in texture-space for current pixel being rendered.



Implementation: Per-Pixel

SIGGRAPH2006

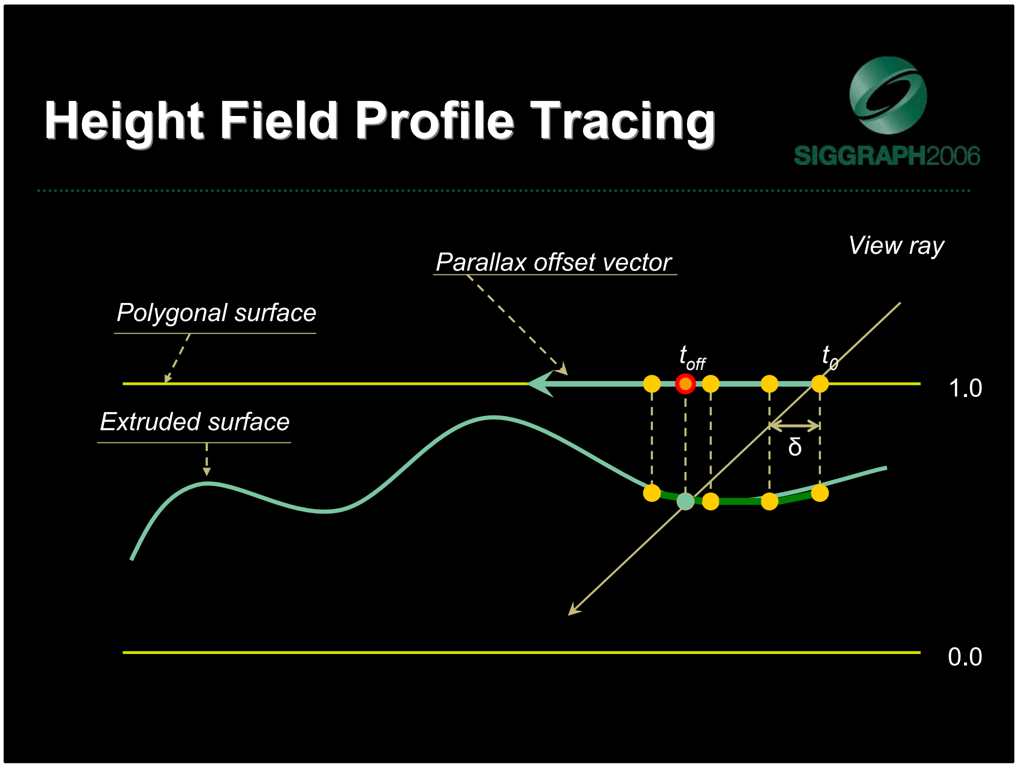
- Ray-cast the view ray along the parallax offset vector
- Ray – height field profile intersection as a texture offset
  - Yields the correct displaced point visible from the given view angle
- Light ray – height profile intersection for occlusion computation to determine the visibility coefficient
-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

Ray cast the view ray along the parallax offset vector to compute the height **profile – ray intersection point**. We sample the height field profile along the parallax offset vector to determine the correct displaced point on the extruded surface. Approximating the height field profile as a piecewise linear curve allows us to have increased precision for the desired intersection (versus simply taking the nearest sample). This yields the texture coordinate shift offset (parallax offset) necessary to arrive at the desired point on the extruded surface. We add this parallax offset amount to the original sample coordinates to yield texture offset coordinates.

If computing shadowing and self-occlusion effects, we can use the texture offset coordinates to perform **visibility computation**

for light direction. In order to do that, we ray cast the light direction ray sampling the height profile along the way for occlusions. This results in a visibility coefficient for the given sample position.

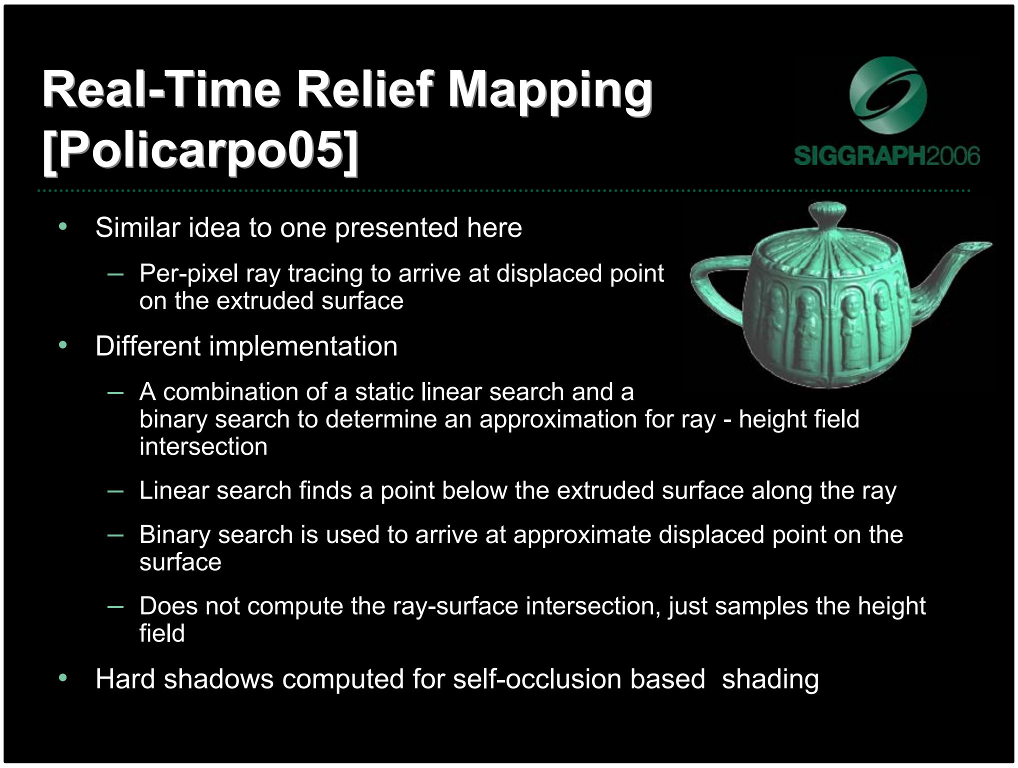
Using the texture offset coordinates and the visibility coefficient, we can **shade the given** pixel using its attributes, such as applied textures (albedo, gloss, etc), the normal from the normal map and the light vector.



Height Field Profile Tracing

SIGGRAPH2006

In order to compute the height field-ray intersection we approximate the height field (seen as the light green curve in this figure) as a piecewise linear curve (seen here as dark green segments), intersecting it with the given ray (in this case, the view direction) for each linear section. We start by tracing from the input sample coordinates t_o along the computed **parallax offset vector *P***. We perform a linear search for the intersection along the parallax offset vector. We sample a linear segment from the height field profile by **fetching two samples** step **size δ** apart. We successively **test each segments endpoints** to see if it would possibly intersect with the view ray. For that, we simply use the height displacement value from each end point to see if they are above current horizon level. Once such pair of end points is found, we compute an intersection between this linear segment and the view ray. The intersection of the height field profile yields the point on the extruded surface that would be visible to the viewer.



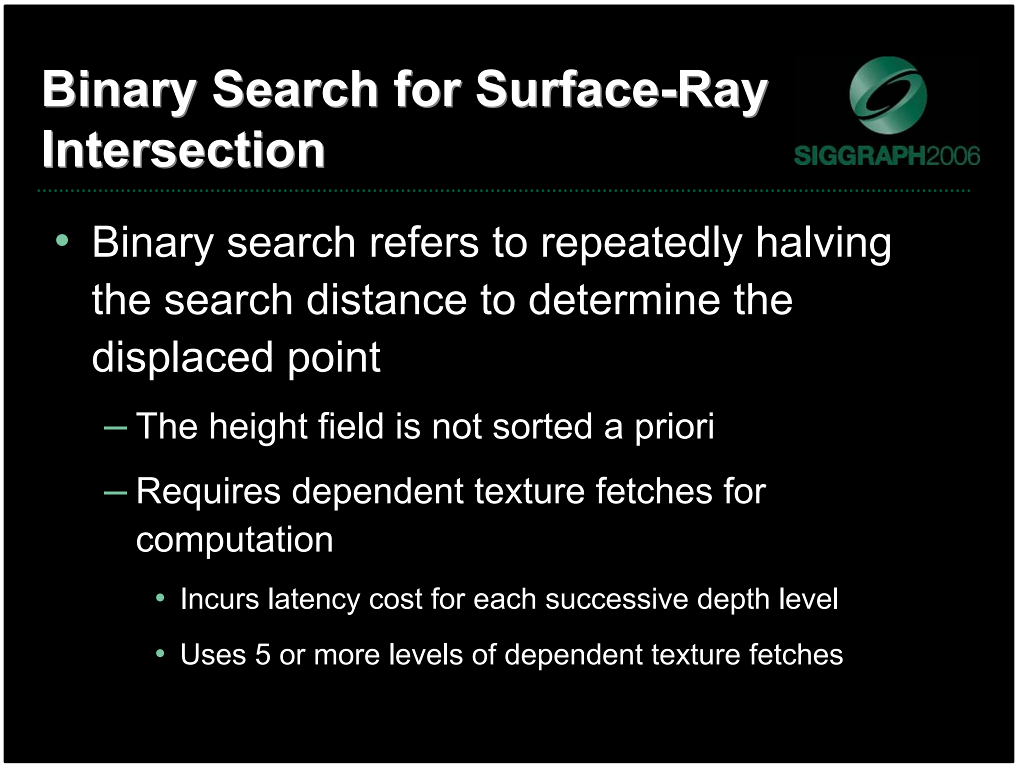
Real-Time Relief Mapping [Policarpo05]

SIGGRAPH2006

- Similar idea to one presented here
  - Per-pixel ray tracing to arrive at displaced point on the extruded surface
- Different implementation
  - A combination of a static linear search and a binary search to determine an approximation for ray - height field intersection
  - Linear search finds a point below the extruded surface along the ray
  - Binary search is used to arrive at approximate displaced point on the surface
  - Does not compute the ray-surface intersection, just samples the height field
- Hard shadows computed for self-occlusion based shading

Techniques such as [Policarpo et al. 2005; Oliveira and Policarpo 2005] determine the intersection point by a combination of a linear and a binary search routines. The relief mapping algorithm approximates the height field with piecewise constant curve, and doesn't actually compute the full intersection, therefore will suffer from aliasing if not enough samples are taken. This technique also computes shadows for surface features, however since it simply tests whether a particular feature is visible or not, this results in hard shadows.

Mapping relief data in tangent space for per-pixel displacement mapping in real-time was proposed in [Brawley and Tatarchuk 2004; Policarpo et al. 2005; McGuire and McGuire 2005] and further extended in [Oliveira and Policarpo et al. 2005] to support silhouette generation. These methods take excellent advantage of the programmable pixel pipeline efficiency by performing height field-ray intersection in the pixel shader to compute the displacement information. These approaches generate dynamic lighting with self-occlusion, shadows and motion parallax. Using the visibility horizon to compute hard shadows as in [Policarpo et al. 2005; McGuire and McGuire 2005; Oliveira and Policarpo 2005] can result in shadow aliasing artifacts. All of the above approaches exhibit strong aliasing and excessive flattening at steep viewing angles. No explicit level of detail schemes were provided with these approaches, relying on texture filtering capabilities of the GPUs.

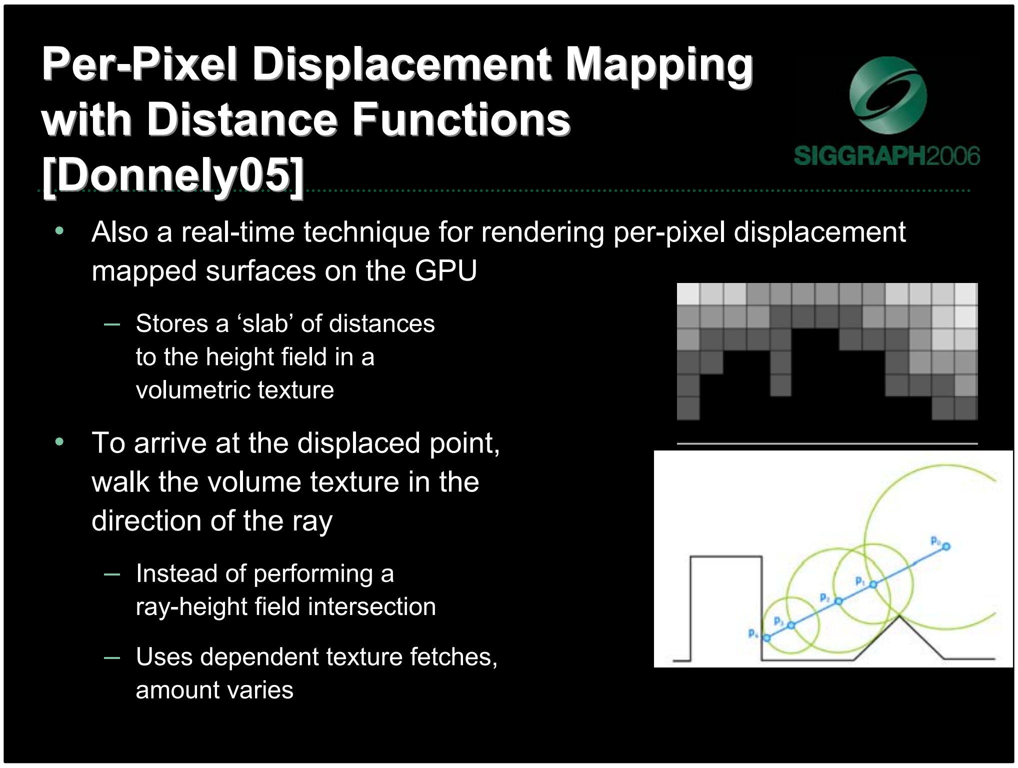


Binary Search for Surface-Ray Intersection

SIGGRAPH2006

- 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

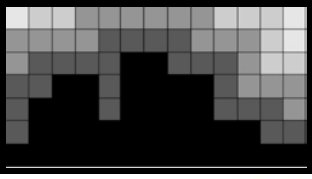
The binary search helps finding an approximate height field intersection utilizing bilinear texture filtering to interpolate the intersection point. The intersection of the surface is approximated with texture filtering, thus only using 8 bit of precision for intersection computation. This results in visible stair-stepping artifacts at steep viewing angles. Depth biasing toward the horizon hides these artifacts but introduces excessive feature flattening at oblique angles

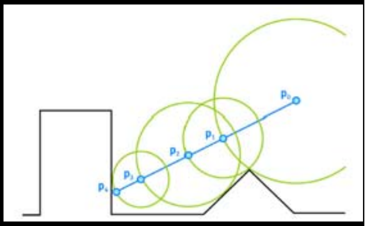


Per-Pixel Displacement Mapping with Distance Functions [Donnely05]

SIGGRAPH2006

- 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



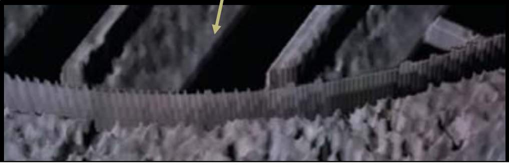


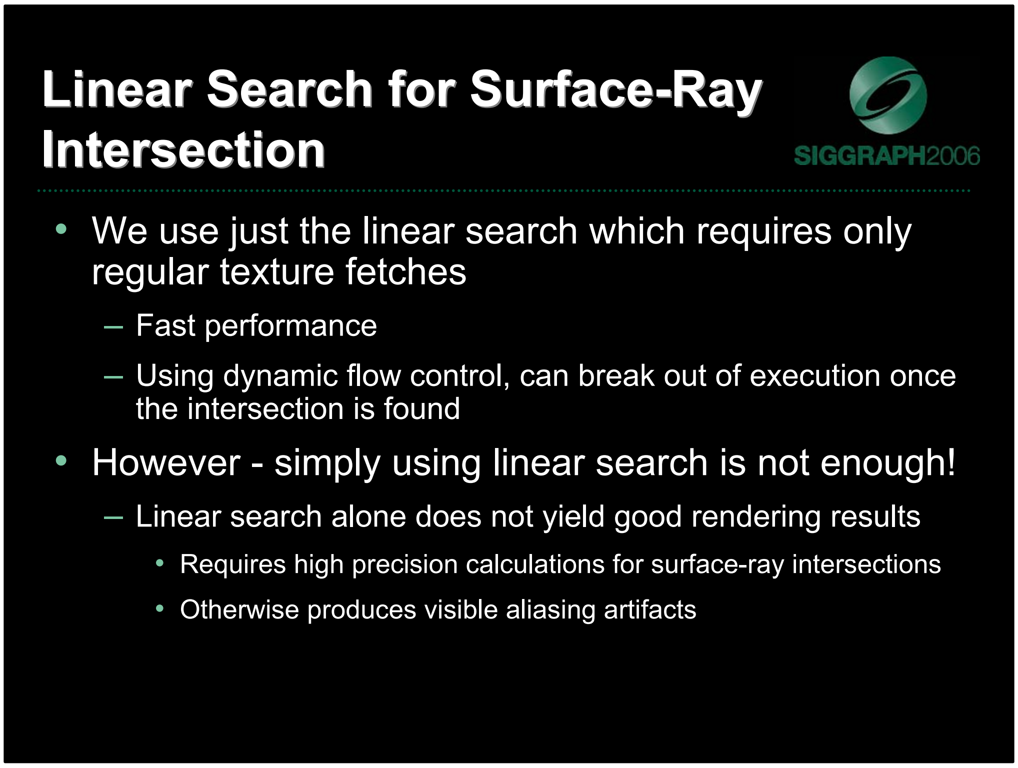
A precomputed three-dimensional distance map for a rendered object can be used for surface extrusion along a given view direction ([Donnelly 2005]). The cost of a 3D texture and dependent texture fetches' latency make this algorithm not applicable to most real-time applications. Each texture fetch into the distance map is not texture-cache coherent

Per-Pixel Displacement Mapping with Distance Functions

SIGGRAPH2006

- 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



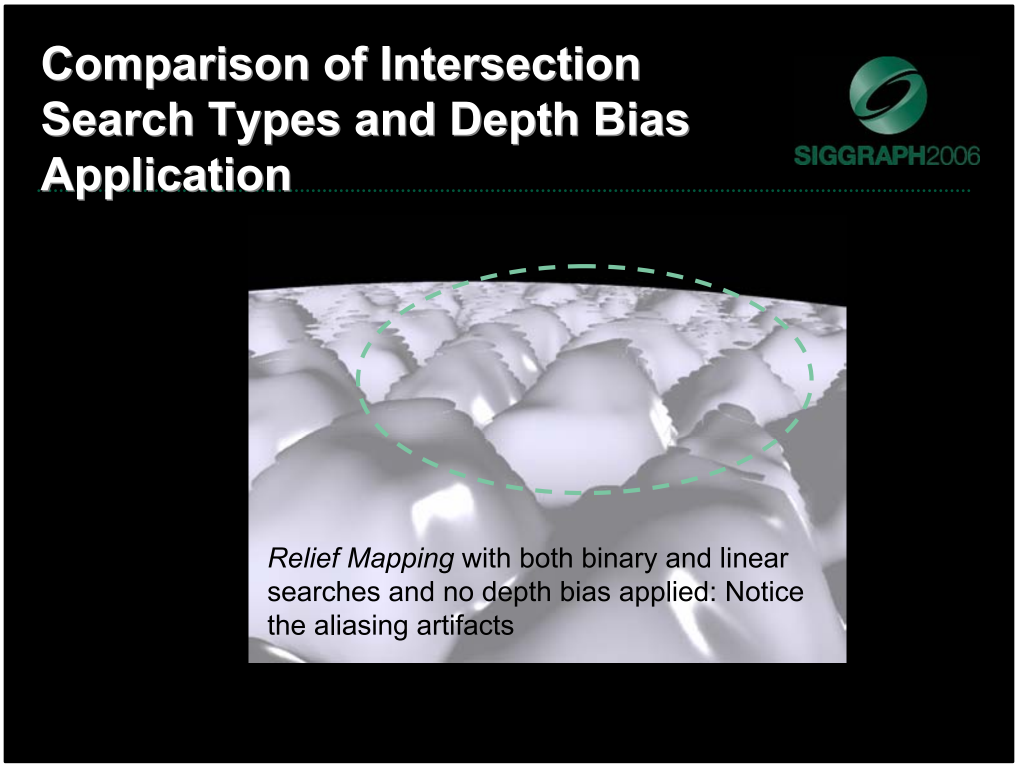


Linear Search for Surface-Ray Intersection

SIGGRAPH2006

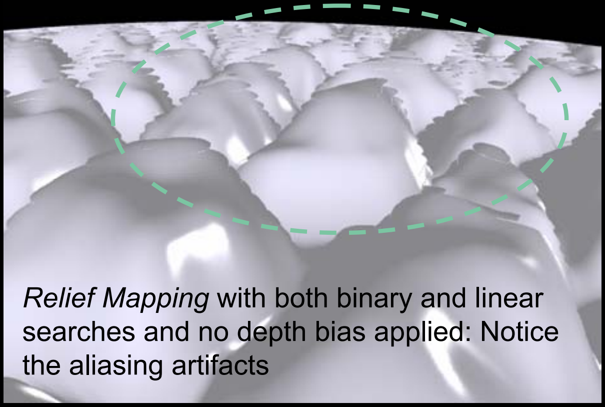
- We use just the linear search which requires only regular texture fetches
  - Fast performance
  - Using dynamic flow control, can break out of execution once the intersection is found
- However - simply using linear search is not enough!
  - Linear search alone does not yield good rendering results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

For our computation we use only just the linear search to arrive at the intersection point. Note that this search utilizes low-latency regular texture fetching and results in good texture cache coherency thus resulting in faster performance. Additionally, using the dynamic flow control feature of the latest consumer GPUs, we can stop tracing the height field profile section once the intersection is found. Note that just simply using a linear search alone is not enough. In order to use just the linear search we must require high precision calculations for surface-ray intersection. Otherwise when using linear search with just bilinear texture fetches for approximating extruded intersection of the height field with a given ray, the results display very strong aliasing.



Comparison of Intersection Search Types and Depth Bias Application

SIGGRAPH2006



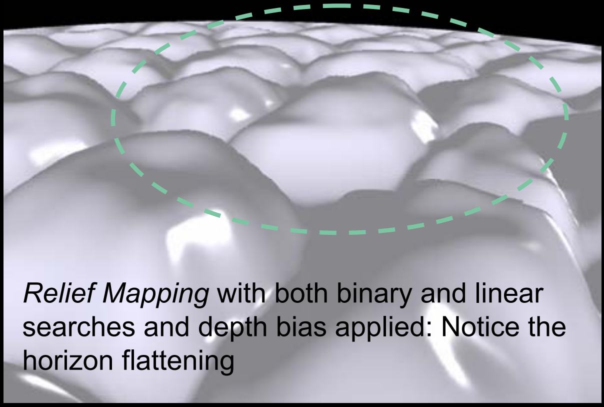
Relief Mapping with both binary and linear searches and no depth bias applied: Notice the aliasing artifacts

Surface approximation methods affect resulting precision for intersection computation. Piecewise constant representation of the surface yields incorrect intersection results just with linear search. Techniques such as [Policarpo et al. 2005; Oliveira and Policarpo 2005] determine the intersection point by a combination of a linear and a binary search routines. These approaches sample the height field as a piecewise constant function. The linear search allows arriving at a point below the extruded surface intersection with the view ray. The following binary search helps finding an approximate height field intersection utilizing bilinear texture filtering to interpolate the intersection point. The intersection of the surface is approximated with texture filtering, thus only using 8 bit of precision for intersection computation. This results in visible stair-stepping artifacts at steep viewing angles (as seen in first figure). Even a combination of binary and linear search with piecewise constant representation does not yield good results - unsuitable for production quality rendering. Significant aliasing at grazing angles makes it unusable. That's why binary search is introduced in [Policarpo05]. Depth biasing toward the horizon hides these artifacts but introduces excessive feature flattening at oblique angles (seen in this figure)

With our algorithm we perform only a linear search combined with a high precision intersection computation for extruded surface-ray intersection. This allows us to preserve perspective-correct depth even at oblique angles as well as display very little or none aliasing due to missed intersections of the extruded surface with the view ray.

Comparison of Intersection Search Types and Depth Bias Application

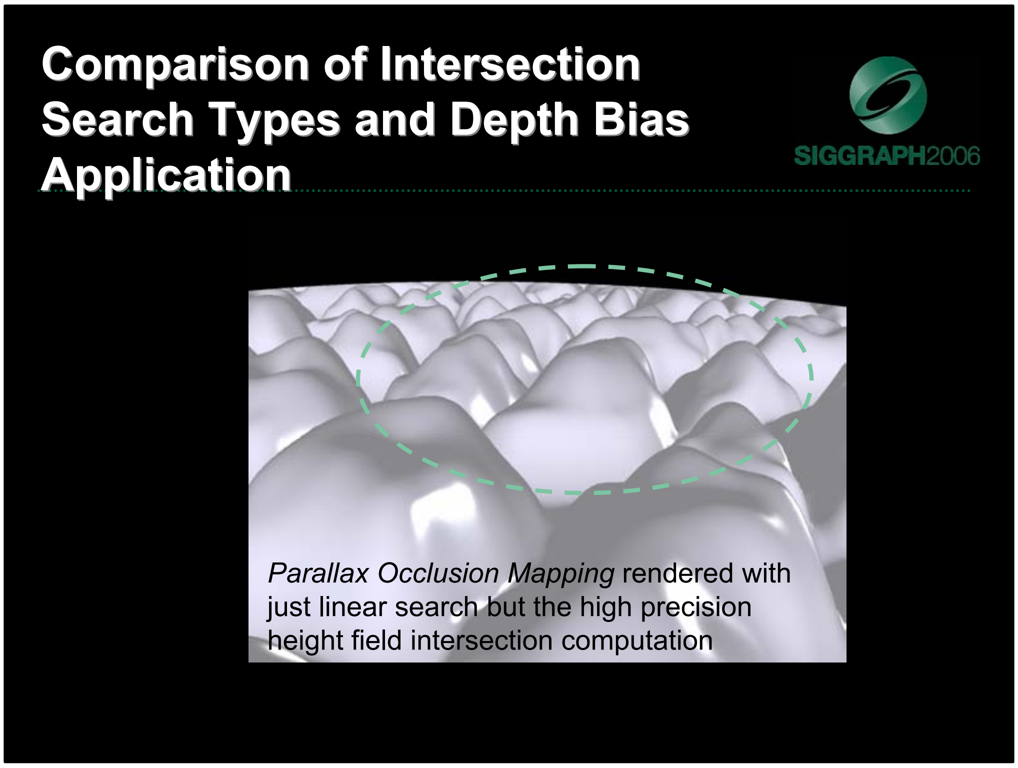
SIGGRAPH2006



Relief Mapping with both binary and linear searches and depth bias applied: Notice the horizon flattening

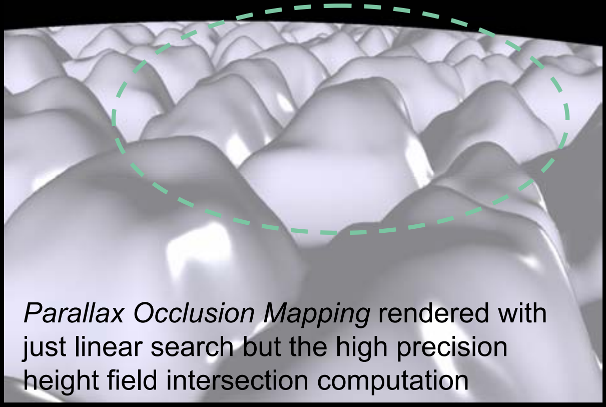
Surface approximation methods affect resulting precision for intersection computation. Piecewise constant representation of the surface yields incorrect intersection results just with linear search. Techniques such as [Policarpo et al. 2005; Oliveira and Policarpo 2005] determine the intersection point by a combination of a linear and a binary search routines. These approaches sample the height field as a piecewise constant function. The linear search allows arriving at a point below the extruded surface intersection with the view ray. The following binary search helps finding an approximate height field intersection utilizing bilinear texture filtering to interpolate the intersection point. The intersection of the surface is approximated with texture filtering, thus only using 8 bit of precision for intersection computation. This results in visible stair-stepping artifacts at steep viewing angles (as seen in first figure). Even a combination of binary and linear search with piecewise constant representation does not yield good results - unsuitable for production quality rendering. Significant aliasing at grazing angles makes it unusable. That's why binary search is introduced in [Policarpo05]. Depth biasing toward the horizon hides these artifacts but introduces excessive feature flattening at oblique angles (seen in this figure)

With our algorithm we perform only a linear search combined with a high precision intersection computation for extruded surface-ray intersection. This allows us to preserve perspective-correct depth even at oblique angles as well as display very little or none aliasing due to missed intersections of the extruded surface with the view ray.



Comparison of Intersection Search Types and Depth Bias Application

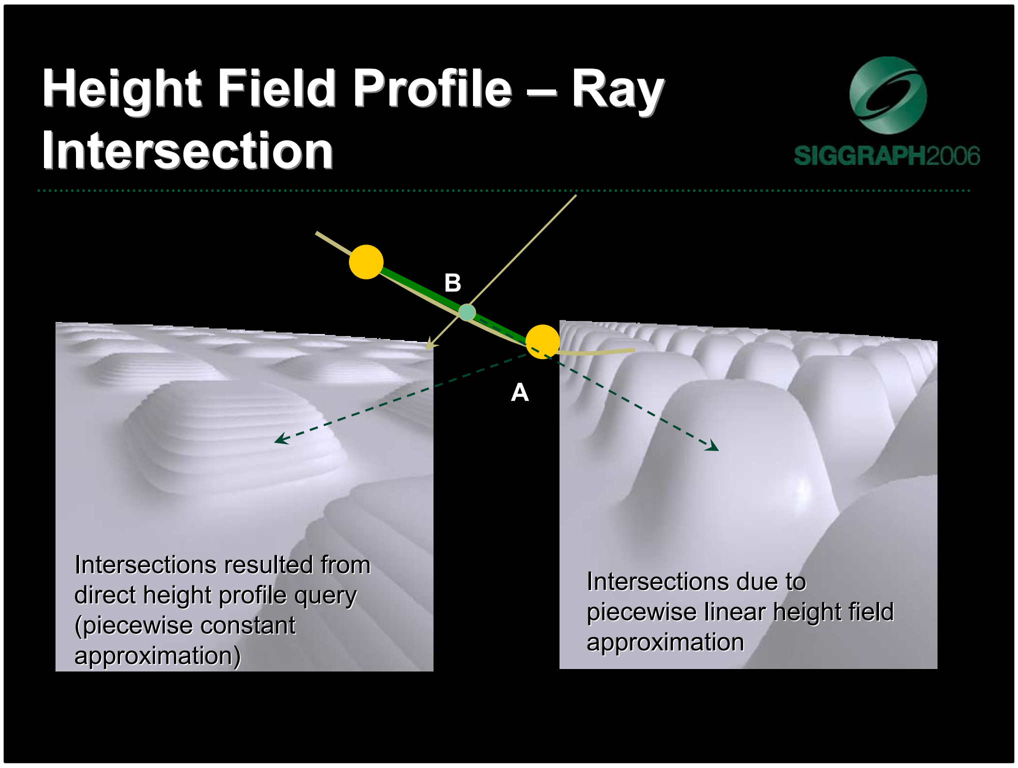
SIGGRAPH2006



Parallax Occlusion Mapping rendered with just linear search but the high precision height field intersection computation

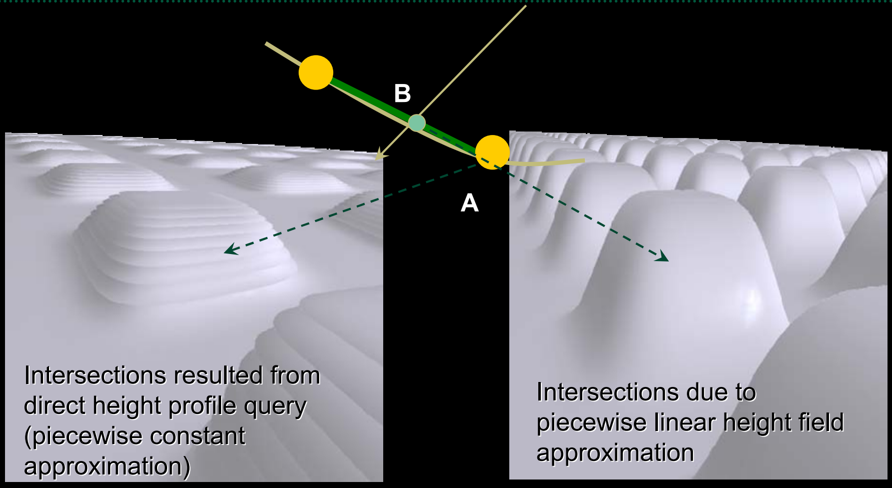
Surface approximation methods affect resulting precision for intersection computation. Piecewise constant representation of the surface yields incorrect intersection results just with linear search. Techniques such as [Policarpo et al. 2005; Oliveira and Policarpo 2005] determine the intersection point by a combination of a linear and a binary search routines. These approaches sample the height field as a piecewise constant function. The linear search allows arriving at a point below the extruded surface intersection with the view ray. The following binary search helps finding an approximate height field intersection utilizing bilinear texture filtering to interpolate the intersection point. The intersection of the surface is approximated with texture filtering, thus only using 8 bit of precision for intersection computation. This results in visible stair-stepping artifacts at steep viewing angles (as seen in first figure). Even a combination of binary and linear search with piecewise constant representation does not yield good results - unsuitable for production quality rendering. Significant aliasing at grazing angles makes it unusable. That's why binary search is introduced in [Policarpo05]. Depth biasing toward the horizon hides these artifacts but introduces excessive feature flattening at oblique angles (seen in this figure)

With our algorithm we perform only a linear search combined with a high precision intersection computation for extruded surface-ray intersection. This allows us to preserve perspective-correct depth even at oblique angles as well as display very little or none aliasing due to missed intersections of the extruded surface with the view ray.



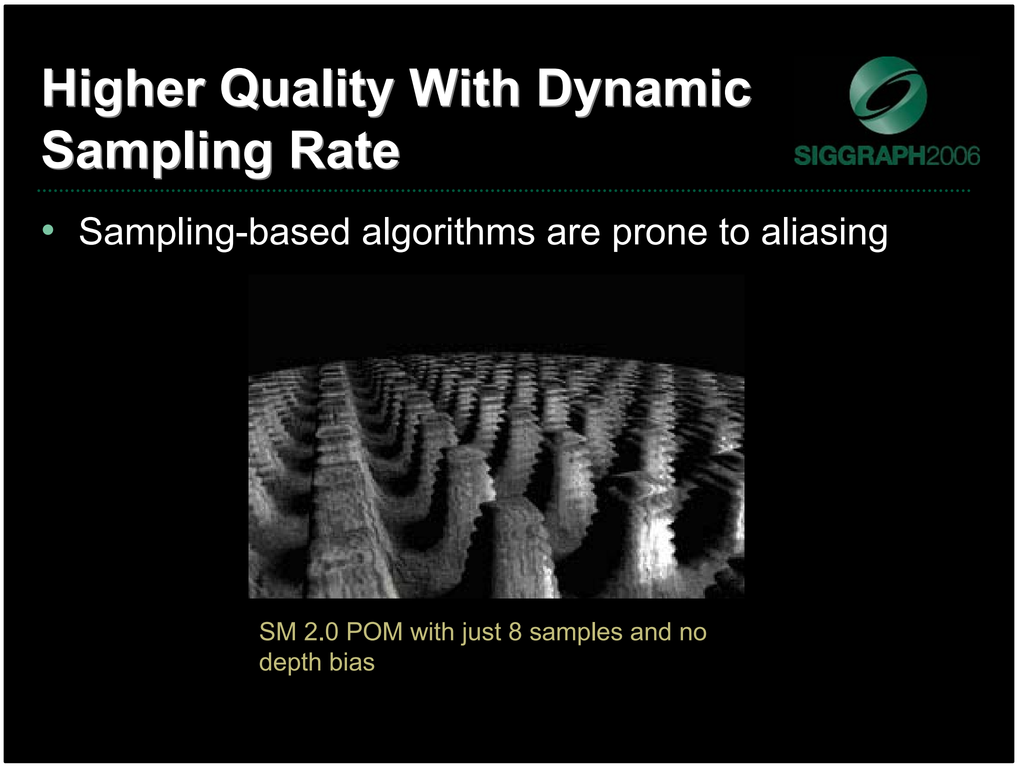
Height Field Profile – Ray Intersection

SIGGRAPH2006


B
A
Intersections resulted from direct height profile query (piecewise constant approximation)
Intersections due to piecewise linear height field approximation

Production quality results require more precise intersection. Other ray tracing-based mapping techniques query the height profile for the closest location to the viewer along the view direction. In the case presented here, these techniques would report point *A* as the displacement point. This results in the stair stepping artifacts visible in the picture on the left. The artifacts are particularly strong at oblique viewing angles, where the apparent parallax is larger. We perform actual line intersection computation for the ray and the linear section of the approximated height field. This yields the intersection point B*.*

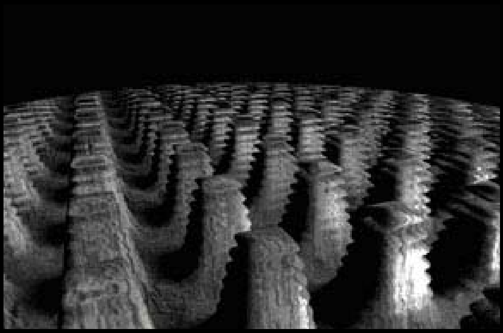
In the figure on the right, you see the smoother surface rendered using higher precision height field intersection technique. In both figures the identical number of samples was used during tracing view direction rays.



Higher Quality With Dynamic Sampling Rate

SIGGRAPH2006

- Sampling-based algorithms are prone to aliasing

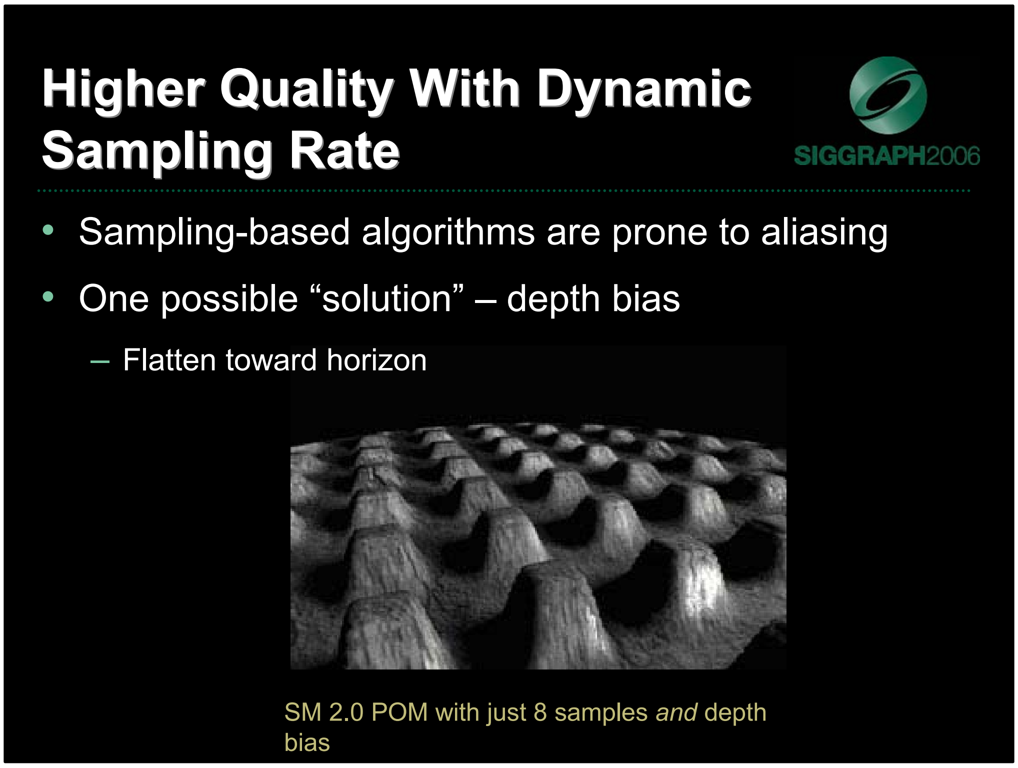


SM 2.0 POM with just 8 samples and no depth bias

One of the biggest problems with the aliasing algorithms exists due to aliasing artifacts. Here you see **the result** of our 2004 technique intersecting the height field with a fixed sampling rate. Note the aliasing artifacts visible with this technique at the grazing angle. To fix this we applied perspective bias to reduce the aliasing artifacts, as visible in **the picture here**. This results in strong flattening of the surface details along the horizon, which is undesirable.

Dynamically scaling the sampling rate ensures that the resulting extruded surface is far less likely to display aliasing artifacts and certainly does not display any flattening as in **this** figure. Therefore the surfaces rendered with our approach display perspective-correct depth at all angles.

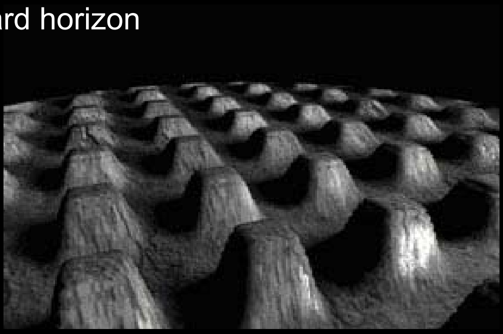
On the latest GPUs we can utilize dynamic flow control instructions to dynamically scale the sampling rate during ray tracing. We express the sampling rate as a linear function of the angle between the geometric normal and the view direction ray. This ensures that we take more samples when the surface is viewed at steep grazing angles, where more samples are desired.



Higher Quality With Dynamic Sampling Rate

SIGGRAPH2006

- Sampling-based algorithms are prone to aliasing
- One possible "solution" – depth bias
  - Flatten toward horizon

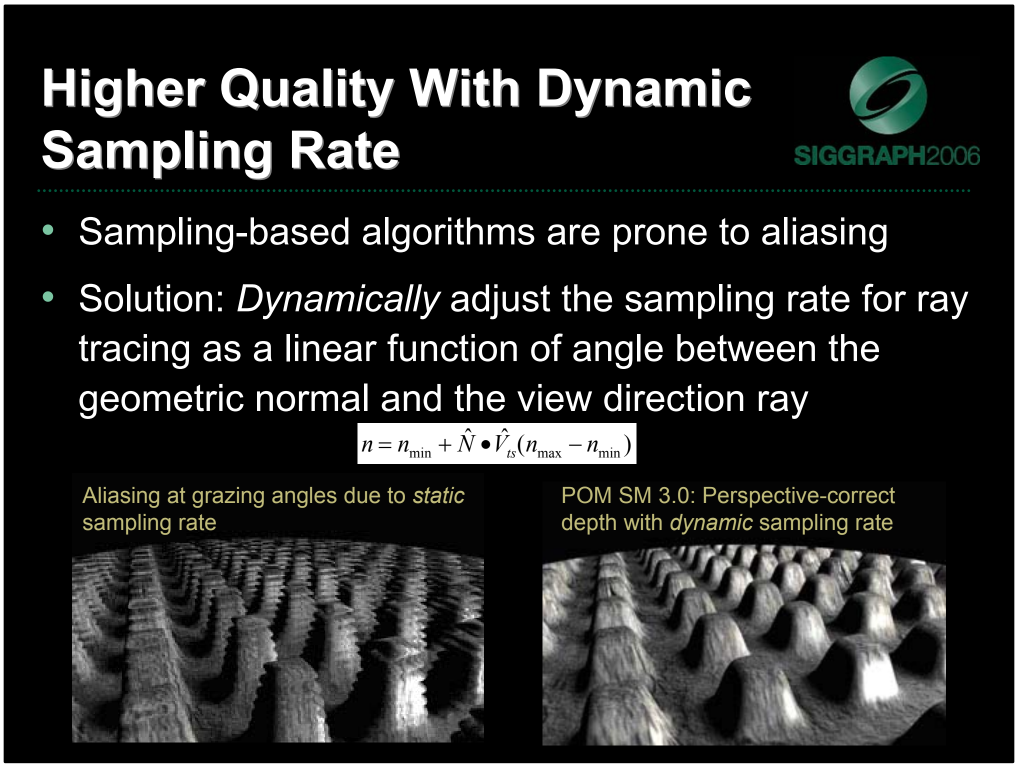


SM 2.0 POM with just 8 samples *and* depth bias

One of the biggest problems with the aliasing algorithms exists due to aliasing artifacts. Here you see **the result** of our 2004 technique intersecting the height field with a fixed sampling rate. Note the aliasing artifacts visible with this technique at the grazing angle. To fix this we applied perspective bias to reduce the aliasing artifacts, as visible in **the picture here**. This results in strong flattening of the surface details along the horizon, which is undesirable.

Dynamically scaling the sampling rate ensures that the resulting extruded surface is far less likely to display aliasing artifacts and certainly does not display any flattening as in **this** figure. Therefore the surfaces rendered with our approach display perspective-correct depth at all angles.

On the latest GPUs we can utilize dynamic flow control instructions to dynamically scale the sampling rate during ray tracing. We express the sampling rate as a linear function of the angle between the geometric normal and the view direction ray. This ensures that we take more samples when the surface is viewed at steep grazing angles, where more samples are desired.



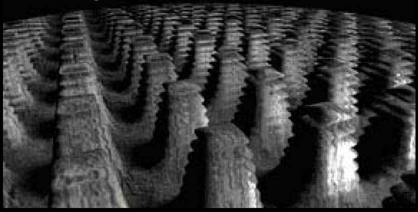
Higher Quality With Dynamic Sampling Rate

SIGGRAPH2006

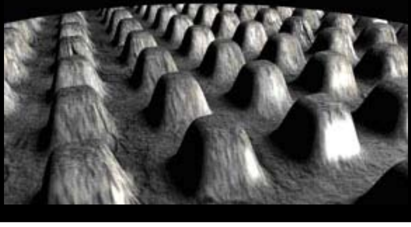
- Sampling-based algorithms are prone to aliasing
- Solution: *Dynamically* adjust the sampling rate for ray tracing as a linear function of angle between the geometric normal and the view direction ray

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

Aliasing at grazing angles due to *static* sampling rate



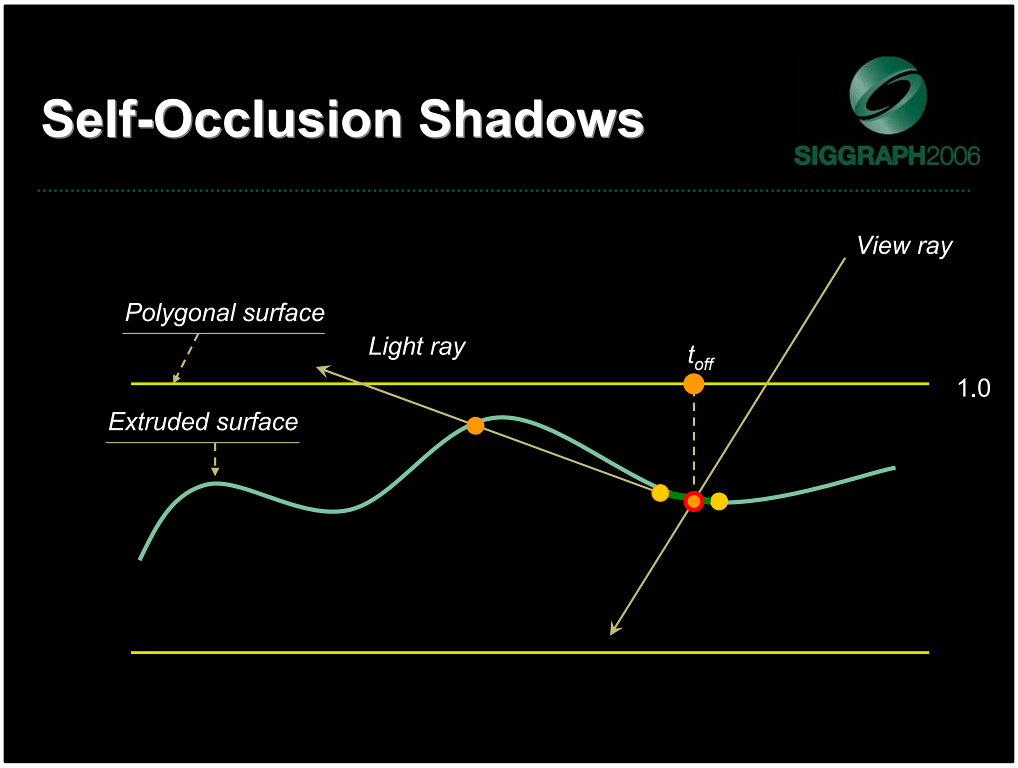
POM SM 3.0: Perspective-correct depth with *dynamic* sampling rate



One of the biggest problems with the aliasing algorithms exists due to aliasing artifacts. Here you see **the result** of our 2004 technique intersecting the height field with a fixed sampling rate. Note the aliasing artifacts visible with this technique at the grazing angle. To fix this we applied perspective bias to reduce the aliasing artifacts, as visible in **the picture here**. This results in strong flattening of the surface details along the horizon, which is undesirable.

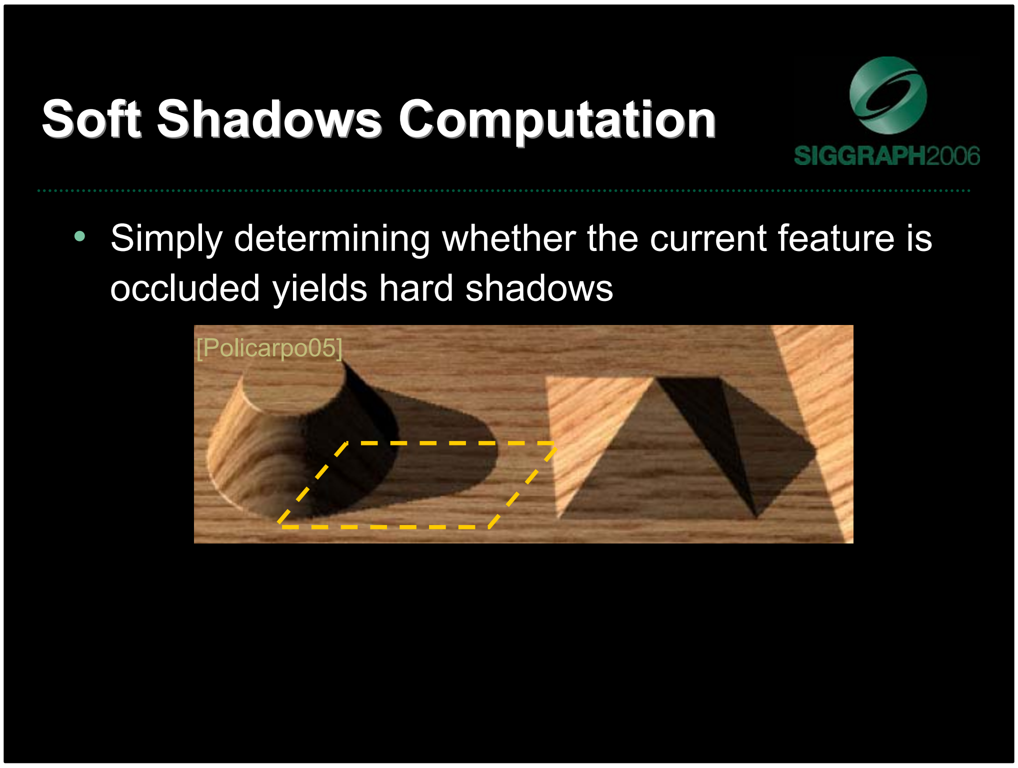
Dynamically scaling the sampling rate ensures that the resulting extruded surface is far less likely to display aliasing artifacts and certainly does not display any flattening as in **this** figure. Therefore the surfaces rendered with our approach display perspective-correct depth at all angles.

On the latest GPUs we can utilize dynamic flow control instructions to dynamically scale the sampling rate during ray tracing. We express the sampling rate as a linear function of the angle between the geometric normal and the view direction ray. This ensures that we take more samples when the surface is viewed at steep grazing angles, where more samples are desired.



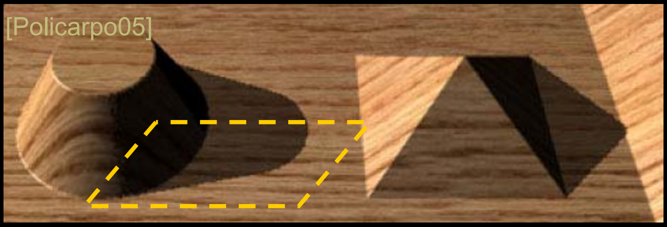
Self-Occlusion Shadows

The features of the height map can in fact cast shadows onto the surface. Once we arrive at the point on the displaced surface (**highlighted here)** we can compute its visibility from the any light source. For that, we **cast a ray toward the light source** in question and perform horizon visibility queries of the height field profile along the light direction ray. If there are intersections of the height field profile with the light vector, then there are occluding features and the point in question will **be in shadow**. This process allows us to generate shadows due to the object features' self-occlusions and object interpenetration.

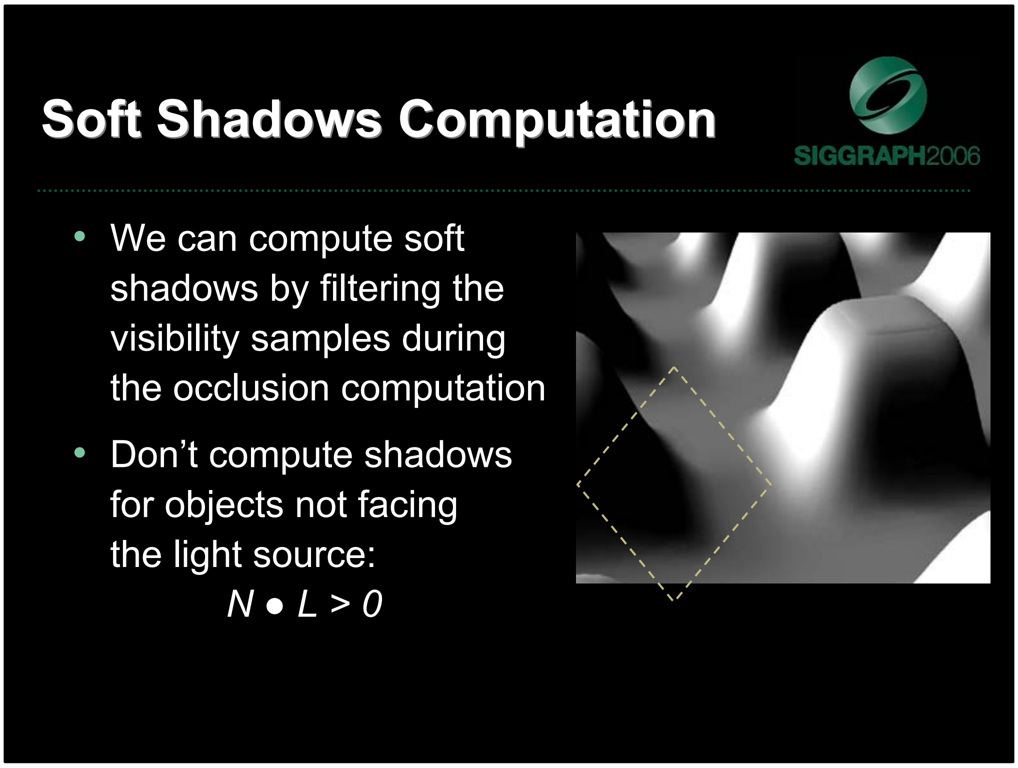


Soft Shadows Computation

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



While computing the visibility information, we could simply stop at the first **intersection blocking** the horizon from the current view point. This yields the horizon shadowing value specifying whether the displaced pixel is in shadow. Other techniques, as seen in this picture, use this approach. This generates **hard shadows** which may have strong aliasing artifacts as you can see in the high-lighted portion.

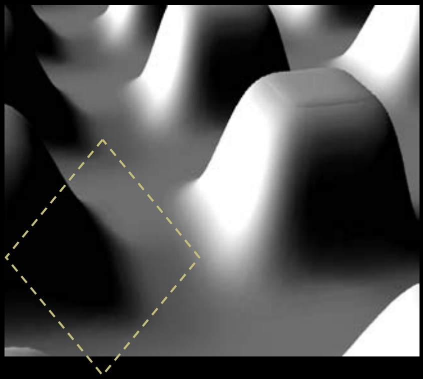


Soft Shadows Computation

SIGGRAPH2006

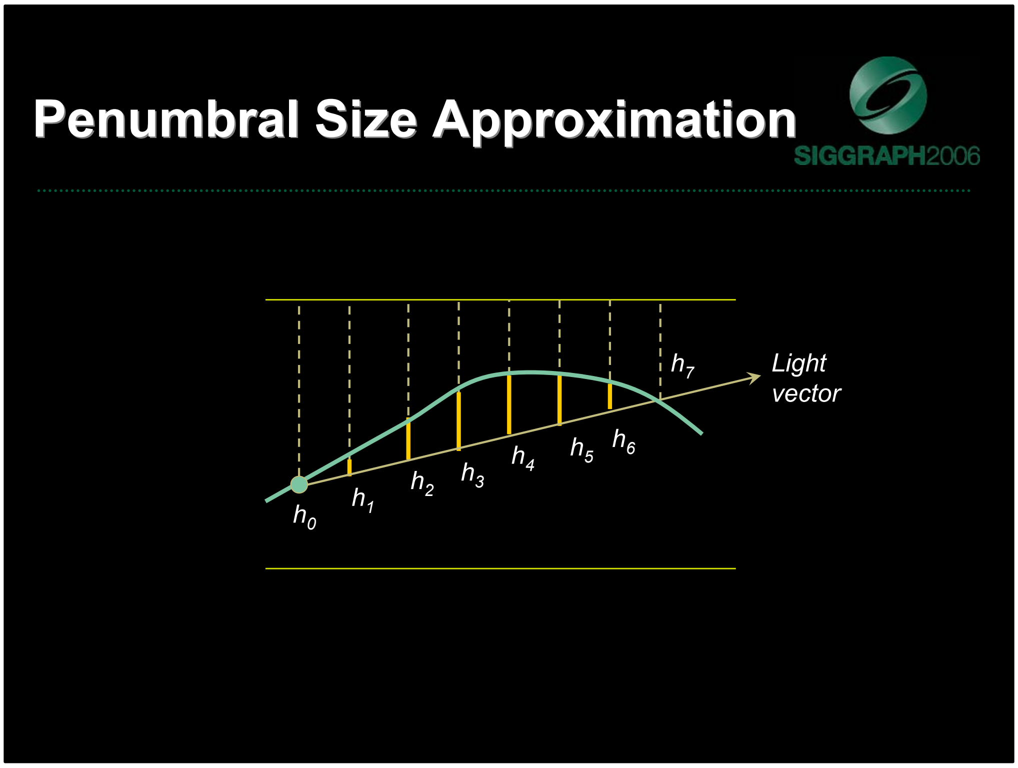
- We can compute soft shadows by filtering the visibility samples during the occlusion computation
- Don't compute shadows for objects not facing the light source:

  $N \bullet L > 0$

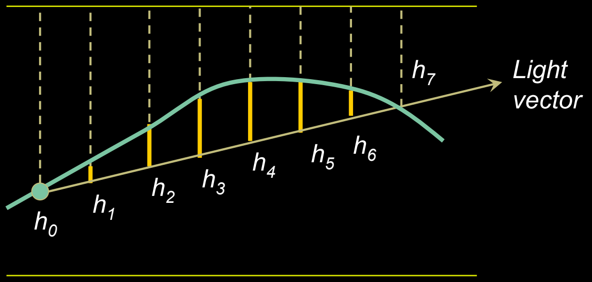


In our algorithm, we continue sampling the height field along the light ray past the first shadowing horizon until we reach the next fully visible point on the surface. Then we filter the resulting visibility samples to compute soft shadows with smooth edges.

We optimize the algorithm by only performing visibility query for areas which are lit by the given light source with a simple test.

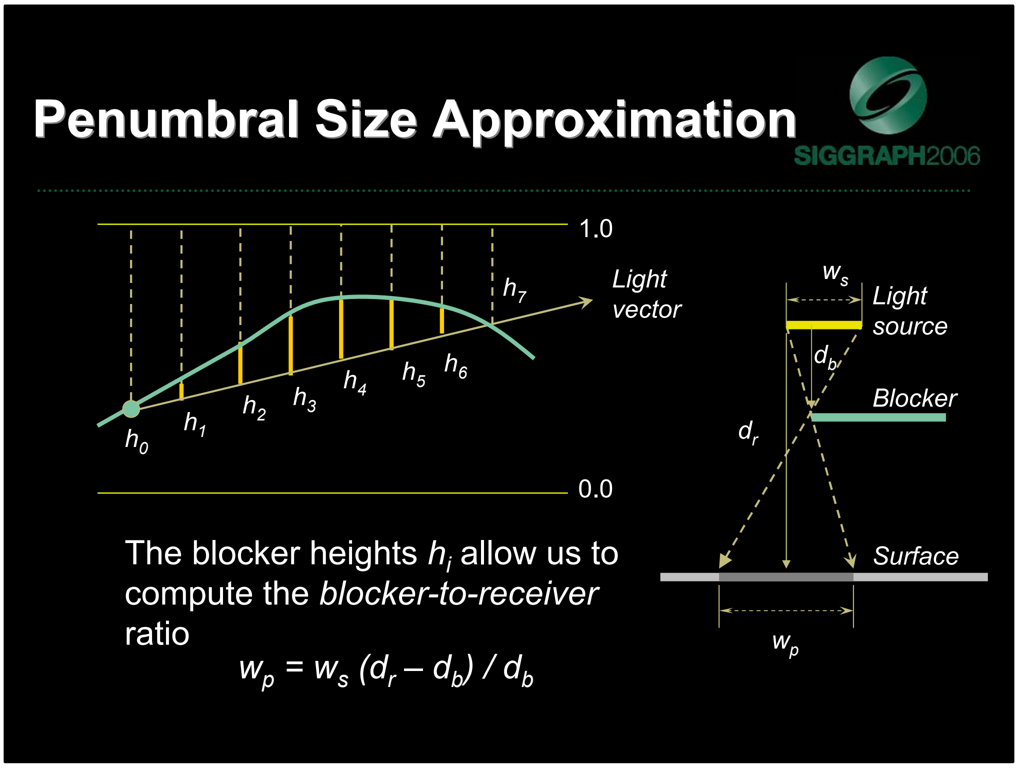


Penumbral Size Approximation

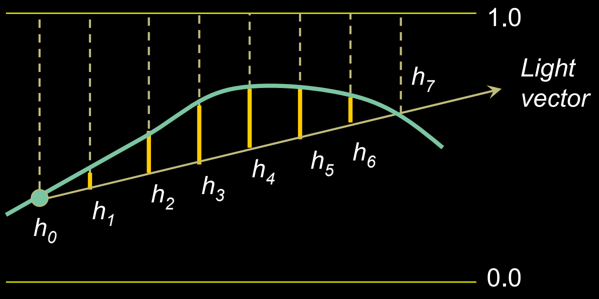


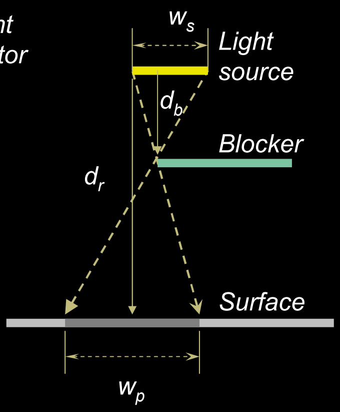
We sample the height value h_0 at the shifted texture coordinate t_{off}. The sample h_0 is our reference ("surface") height. We then sample *n* other samples along the light ray, subtracting h_0 from each of the successive samples h_i. This allows us to compute the blocker-to-receiver ratio as in figure.

We note that the closer the blocker is to the surface, the smaller the resulting penumbra. We compute the the visibility coefficient by scaling the contribution of each sample by the distance from the reference sample. We apply this visibility coefficient during the lighting computation for generation of smoothly soft shadows. In combination with bi- or trilinear texture filtering in hardware, we are able to obtain well-behaved soft shadows without any edge aliasing or filtering artifacts present in many shadow mapping techniques.



Penumbral Size Approximation



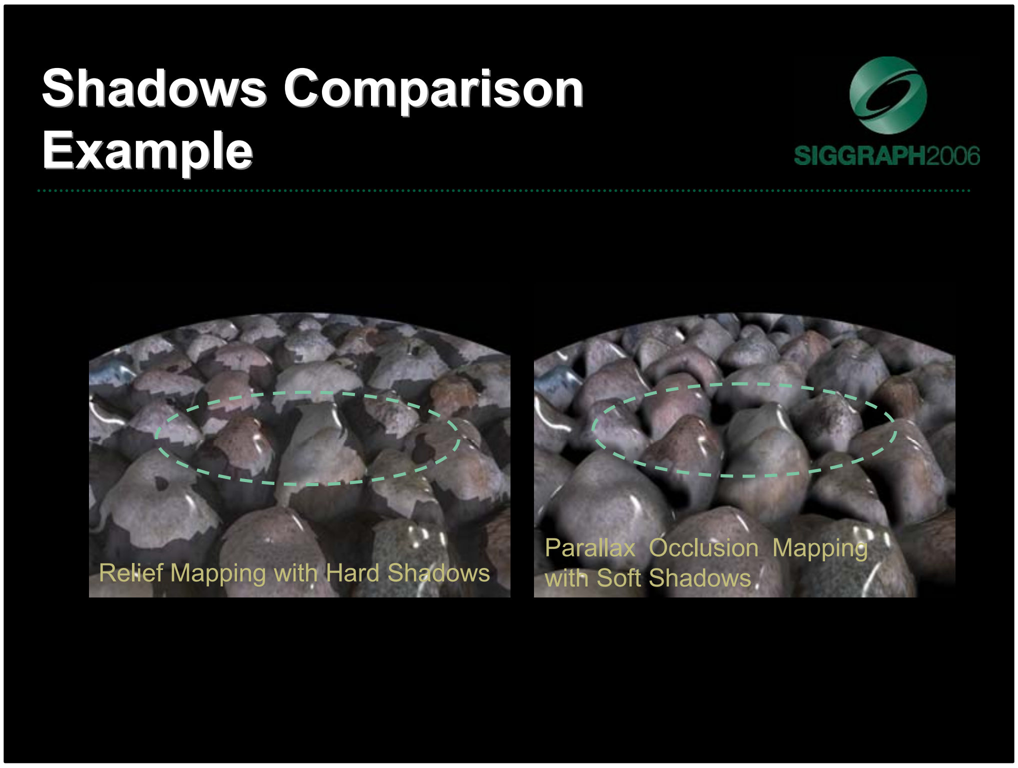


The blocker heights h_i allow us to compute the *blocker-to-receiver* ratio

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

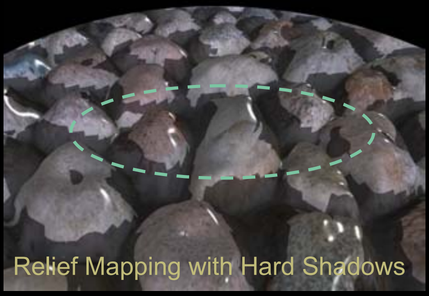
We sample the height value h_0 at the shifted texture coordinate t_{off}. The sample h_0 is our reference ("surface") height. We then sample *n* other samples along the light ray, subtracting h_0 from each of the successive samples h_i. This allows us to compute the blocker-to-receiver ratio as in figure.

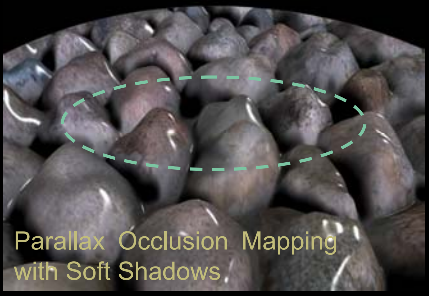
We note that the closer the blocker is to the surface, the smaller the resulting penumbra. We compute the the visibility coefficient by scaling the contribution of each sample by the distance from the reference sample. We apply this visibility coefficient during the lighting computation for generation of smoothly soft shadows. In combination with bi- or trilinear texture filtering in hardware, we are able to obtain well-behaved soft shadows without any edge aliasing or filtering artifacts present in many shadow mapping techniques.



Shadows Comparison Example

SIGGRAPH2006



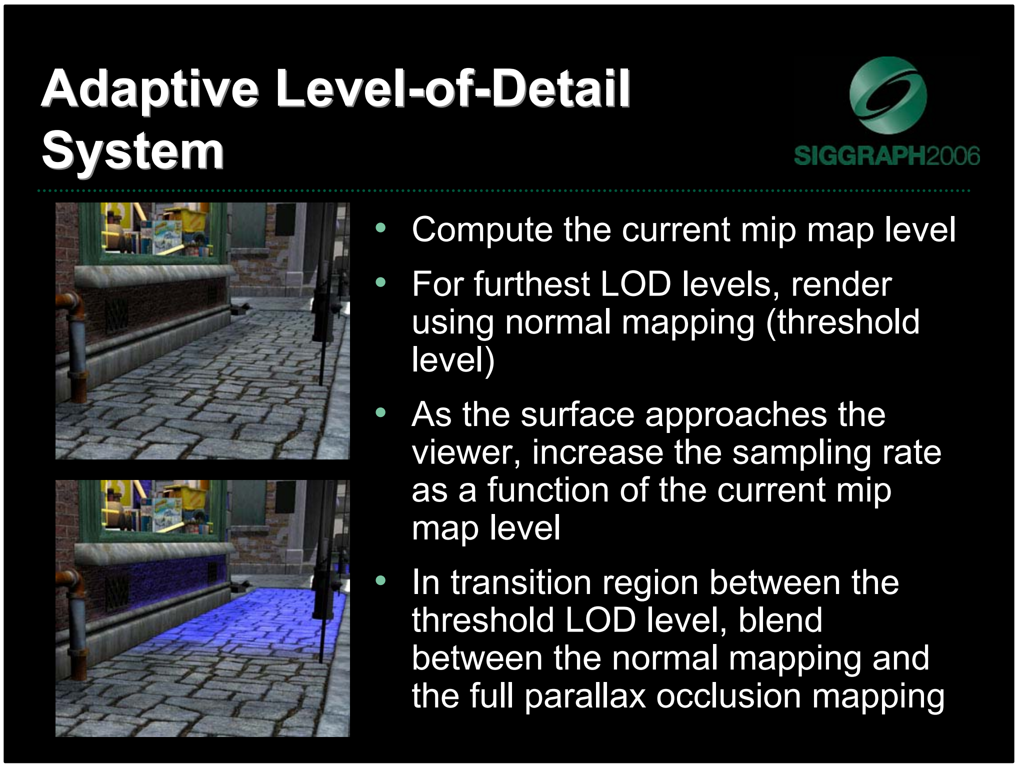


Here you see a comparison of rendering the same scene with relief mapping with hard shadows (on the left) and with parallax occlusion mapping with approximate soft shadows (on the right). We note that the closer the blocker is to the surface, the smaller the resulting penumbra. We compute the visibility coefficient by scaling the contribution of each sample by the distance from the reference sample. We apply this visibility coefficient during the lighting computation for generation of smoothly soft shadows.
In combination with bi- or trilinear texture filtering in hardware, we are able to obtain well-behaved soft shadows without any edge aliasing or filtering artifacts present in many shadow mapping techniques.

Illuminating the Surface

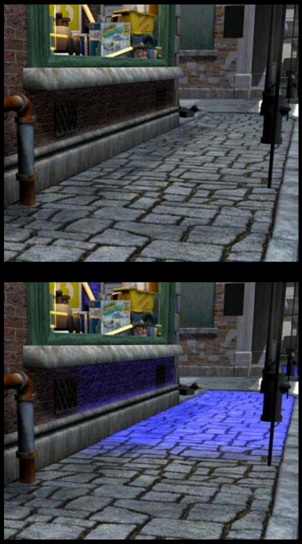
SIGGRAPH2006

- 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



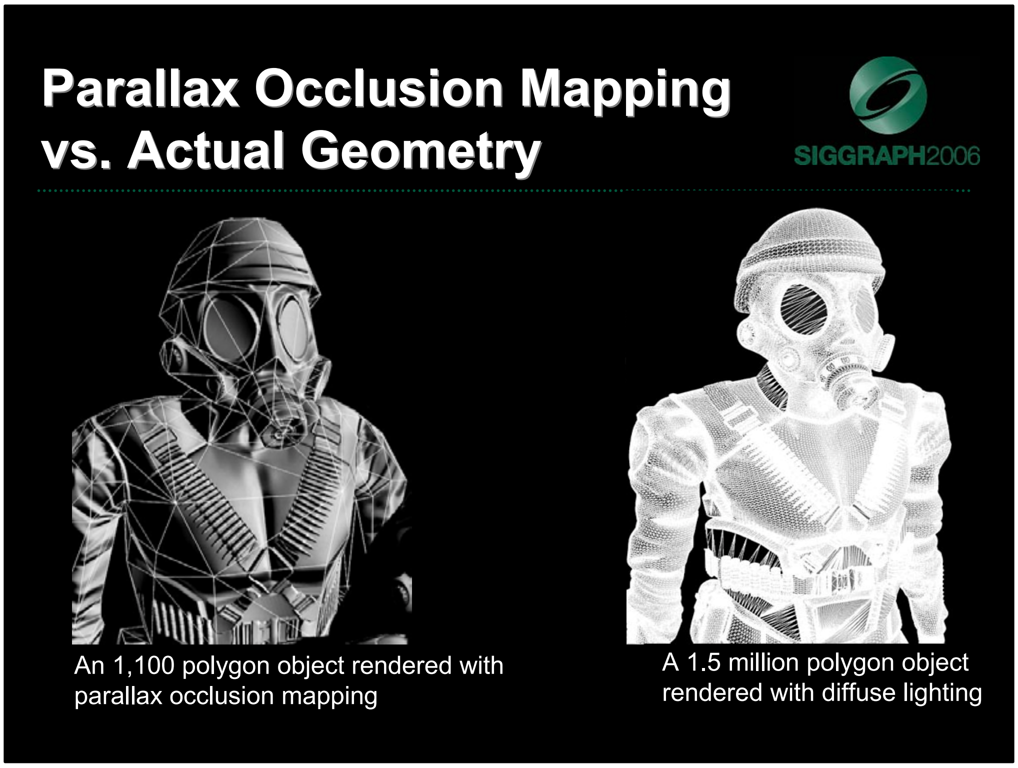
Adaptive Level-of-Detail System

SIGGRAPH2006



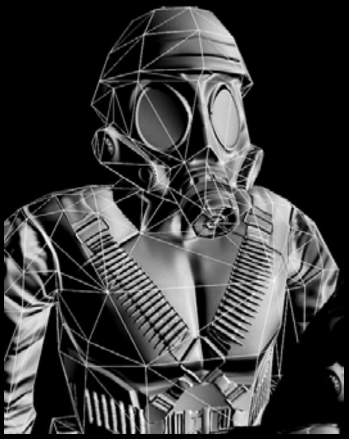
- Compute the current mip map level
- For furthest LOD levels, render using normal mapping (threshold level)
- As the surface approaches the viewer, increase the sampling rate as a function of the current mip map level
- In transition region between the threshold LOD level, blend between the normal mapping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

We designed an explicit level-of-detail control system for automatically controlling shader complexity. We **determine the current mip map level** directly in the pixel shader and use this information to transition between different levels of detail from the full effect to simple normal mapping. We render the **lowest level of details** using regular normal mapping shading. As the surface **approaches the viewer**, we increase the sampling rate for the full parallax occlusion mapping computation as a function of the current mip level. We specify an artist-directable **threshold level** where the transition between the parallax occlusionmapping and the normal mapping computations will occur. When the currently rendered surface portion is in the transition region, we interpolate the result of parallax occlusion mapping computation with the normal mapping result. We using the fractional part of the current mip level computed in the pixel shader. As you can compare between these **two figures,** there is no associated visual quality degradation as we move into a lower level of detail and the transition appears quite smooth.

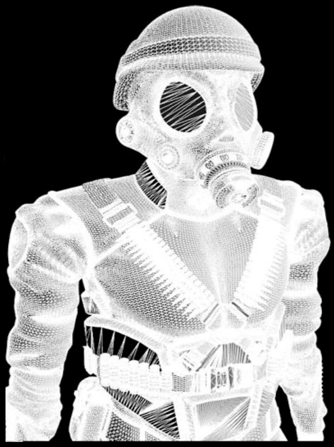


Parallax Occlusion Mapping vs. Actual Geometry

SIGGRAPH2006

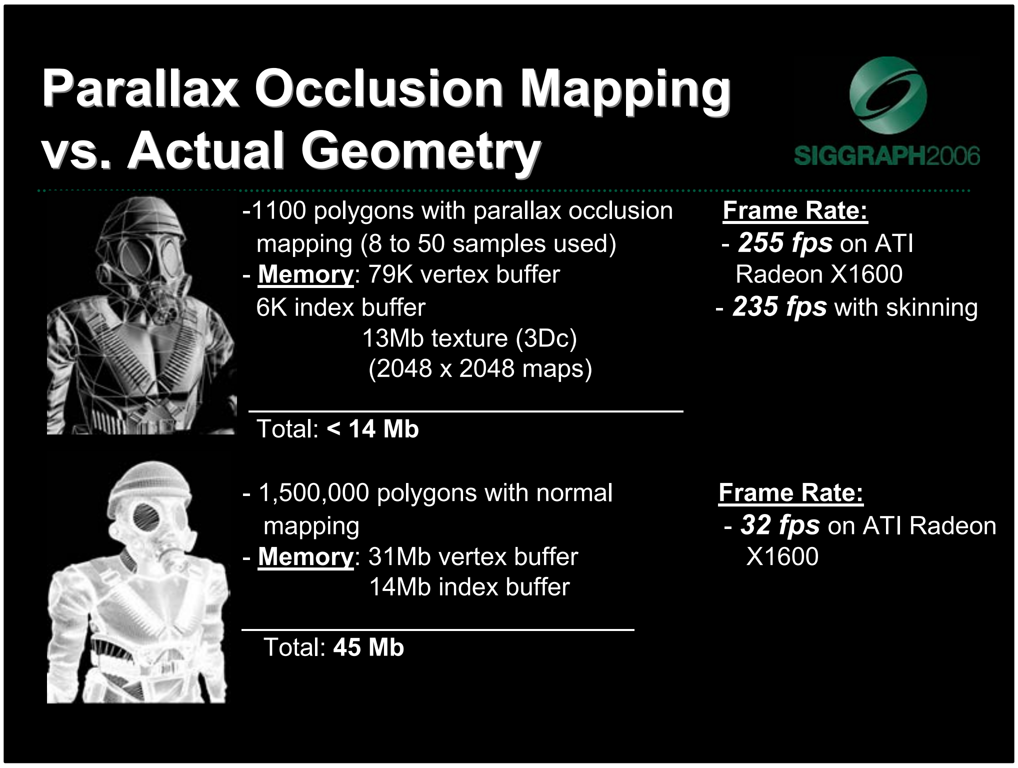


An 1,100 polygon object rendered with parallax occlusion mapping



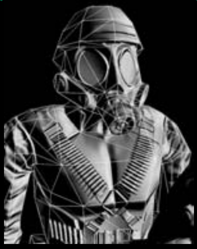
A 1.5 million polygon object rendered with diffuse lighting

We applied parallax occlusion mapping to an 1,100 polygon soldier character displayed on the left. We compared this result to a 1.5 million polygon soldier displayed on the right used to generate normal maps for the low resolution model. We use the same lighting model on both objects. We apply a 2048x2048 RGBα texture map to the low resolution object.



Parallax Occlusion Mapping vs. Actual Geometry

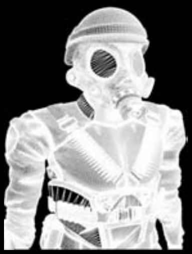
SIGGRAPH2006



-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 X1600
- ***235 fps*** with skinning

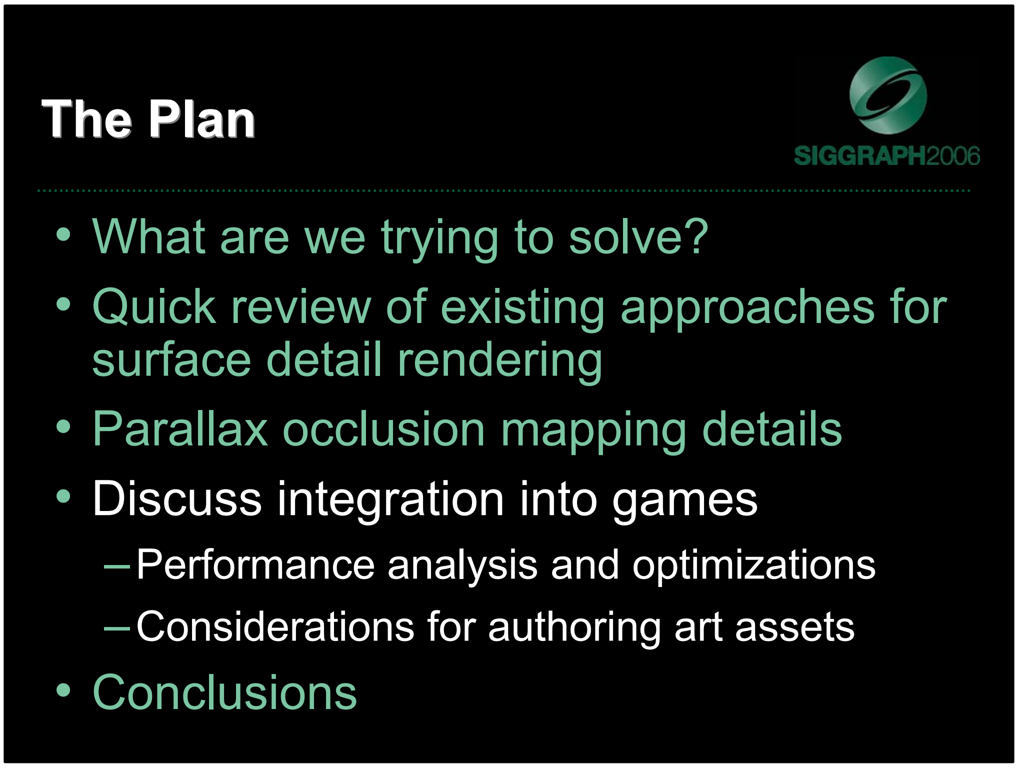


- 1,500,000 polygons with normal mapping
- **Memory**: 31Mb vertex buffer
  14Mb index buffer

Total: **45 Mb**

Frame Rate:
- ***32 fps*** on ATI Radeon X1600

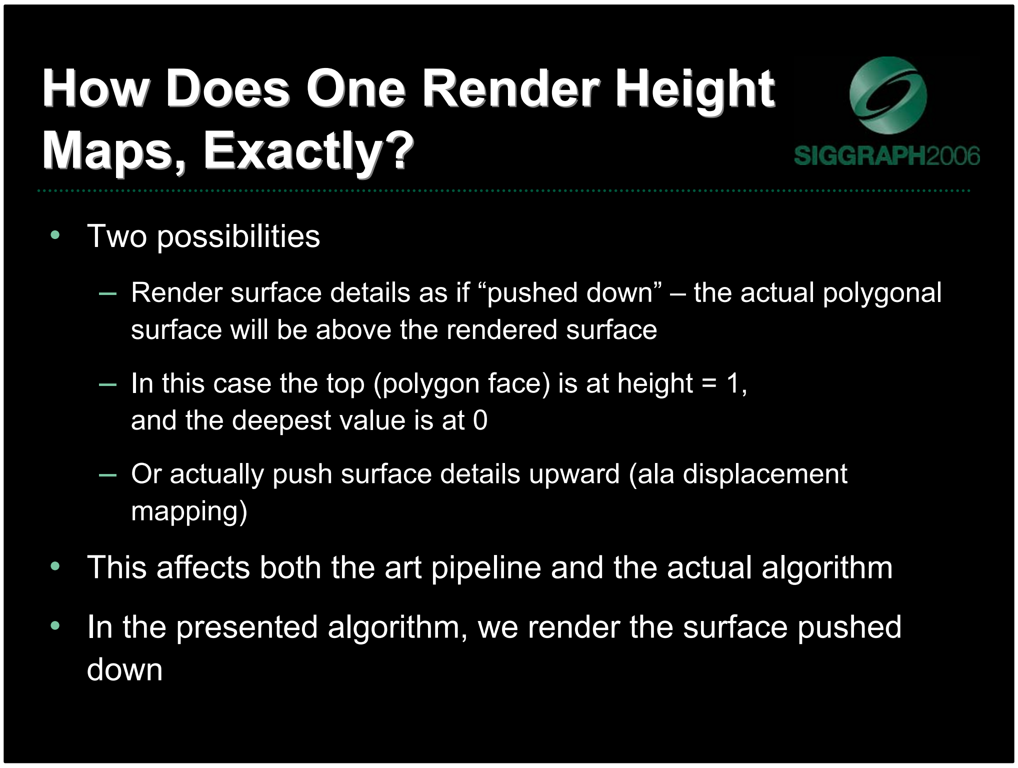
We render the low resolution soldier using DirectX on ATI Radeon X850 at 255 fps. From 8 to 50 samples were used during ray tracing as necessary. The memory requirement for this model was 79K for the vertex buffer and 6K for the index buffer, and 13Mb of texture memory (we use 3DC texture compression). The high resolution soldier model rendered on the same hardware at a rate of 32 fps. The memory requirement for this model was 31Mb for the vertex buffer and 14Mb for the index buffer. However, using our technique on an extremely low resolution model provided significant frame rate increase with 32Mb memory saving at comparable quality of rendering. Notice the details on the bullet belts and the gas mask for the low polygon soldier. We also animated the low resolution model with a run cycle using skinning in vertex shader rendering at 235 fps on the same hardware. Due to memory considerations, vertex transform cost for rendering, animation, and authoring issues, characters matching the high resolution soldier are impractical in current game scenarios.



The Plan

SIGGRAPH2006

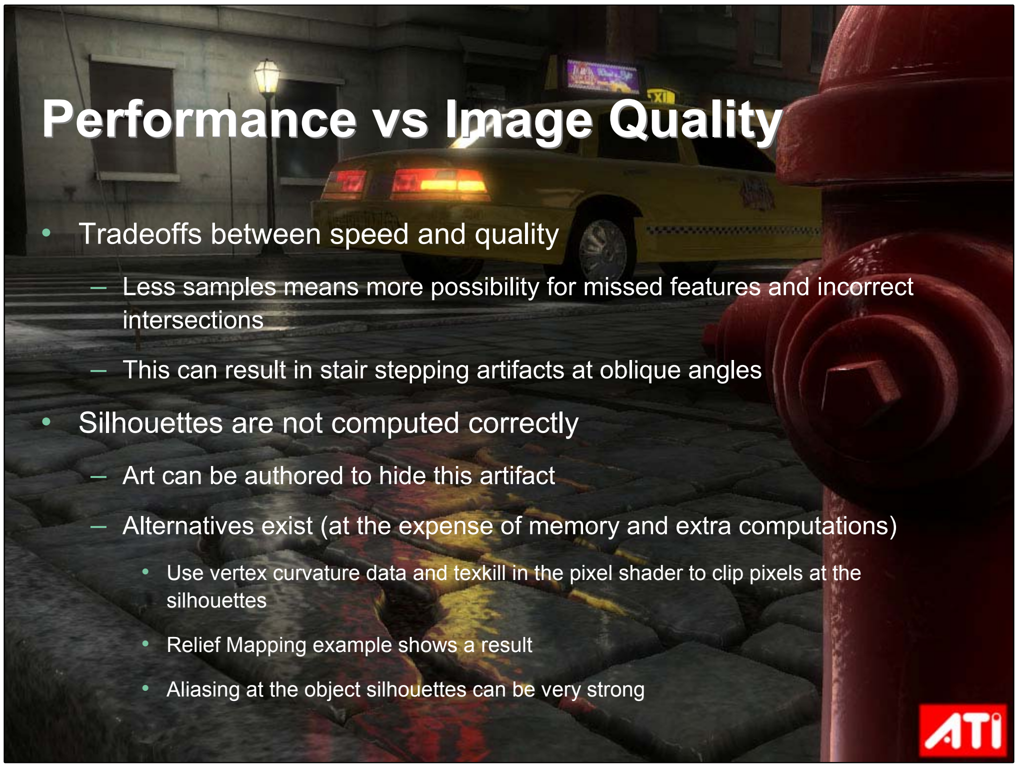
- What are we trying to solve?
- Quick review of existing approaches for surface detail rendering
- Parallax occlusion mapping details
- Discuss integration into games
  - Performance analysis and optimizations
  - Considerations for authoring art assets
- Conclusions



How Does One Render Height Maps, Exactly?

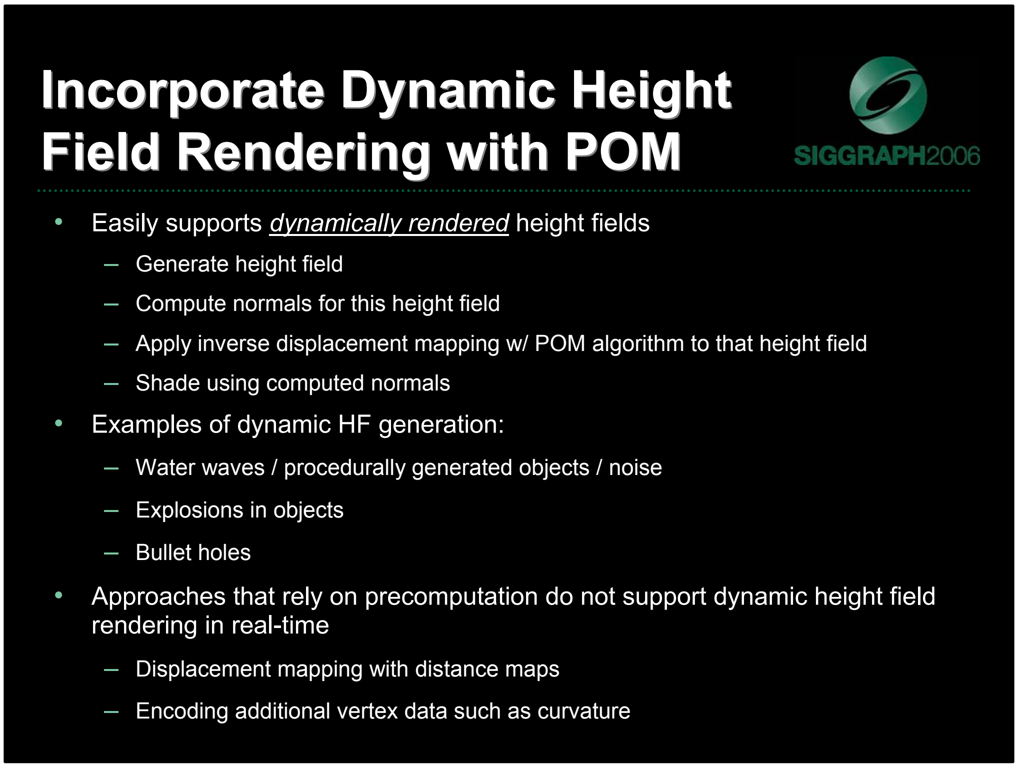
SIGGRAPH2006

- 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



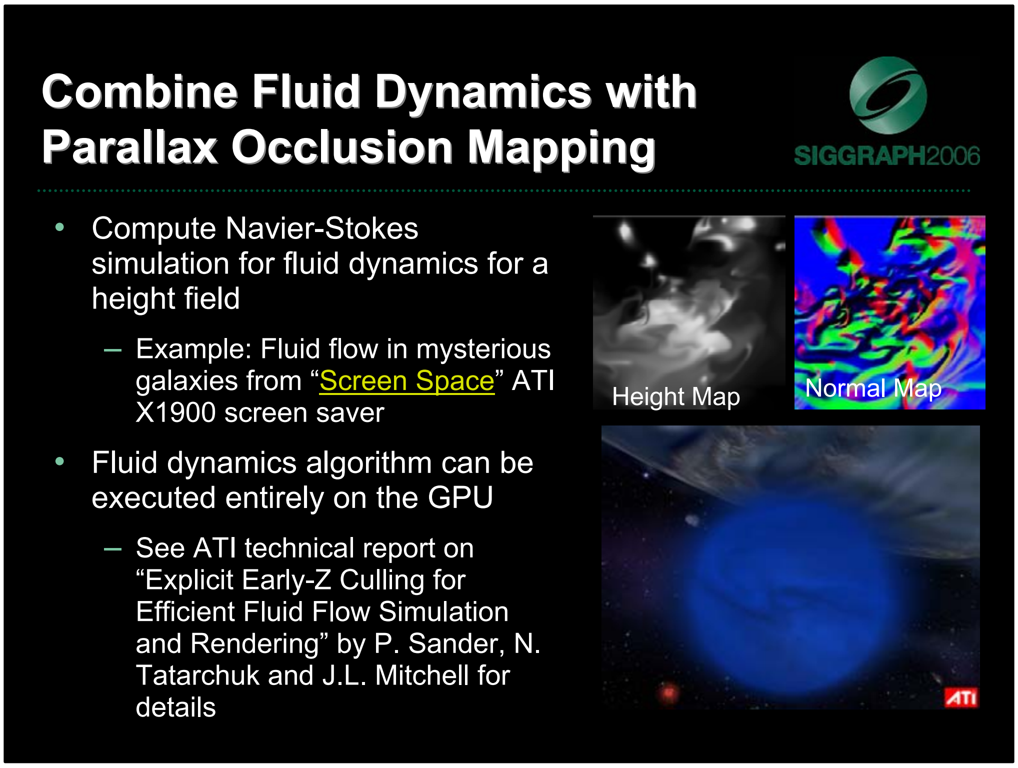
Incorporate Dynamic Height Field Rendering with POM

SIGGRAPH2006

- 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

Our method can be used with a dynamically rendered height field and still produce perspective-correct depth results. In that case, the dynamically updated displacement values can be used to derive the normal vectors at rendering time by convolving the height map with a Sobel operator in the horizontal and vertical direction. The rest of the algorithm does not require any modifications.

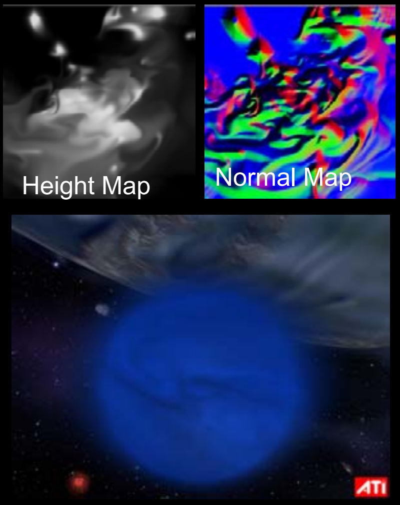
This can be used in games to improve visual quality of interactive scenes. For example, parallax occlusion mapping can be successfully used on procedurally generated height fields. It can be used to render explosions in objects or dynamic bullet holes. Note that other approaches that require precomputed qualities do not support dynamic rendering to height fields.



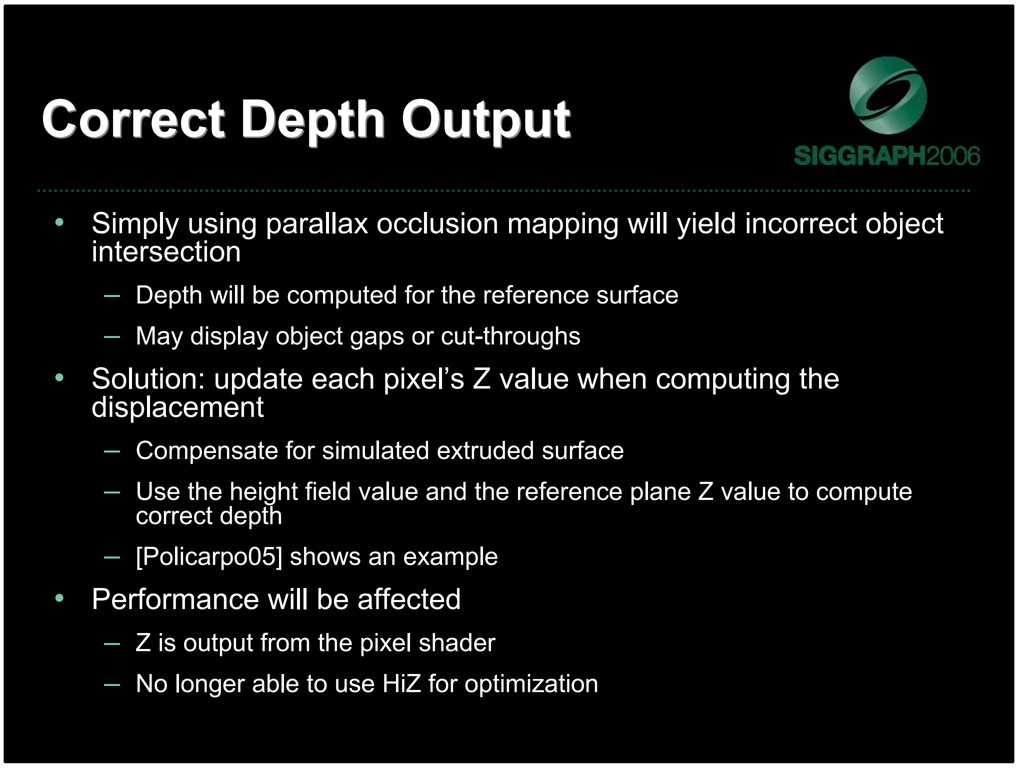
Combine Fluid Dynamics with Parallax Occlusion Mapping

SIGGRAPH2006

- 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



We able to use physics-based Navier-Stokes fluid dynamics simulation as the basis for rendering a height field of a distant gaseous planet in ATI's "ScreenSpace" screen saver. There the entire fluid dynamics simulation is performed entirely on the GPU (see our technical report from 2004).



Correct Depth Output

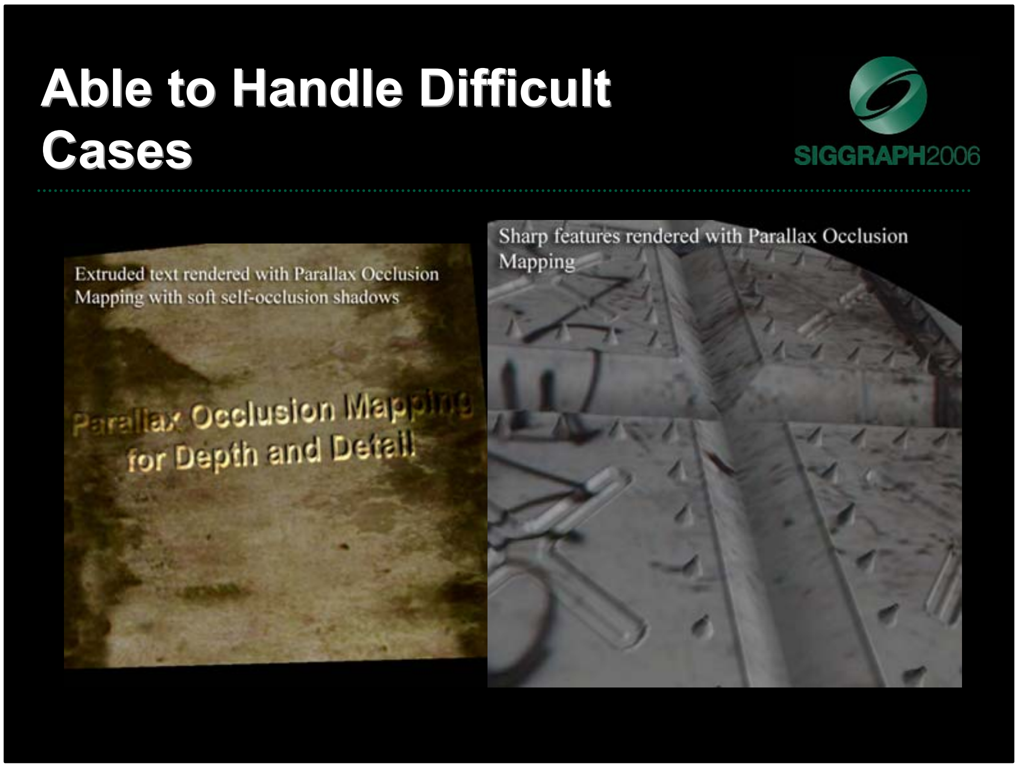
SIGGRAPH2006

- 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

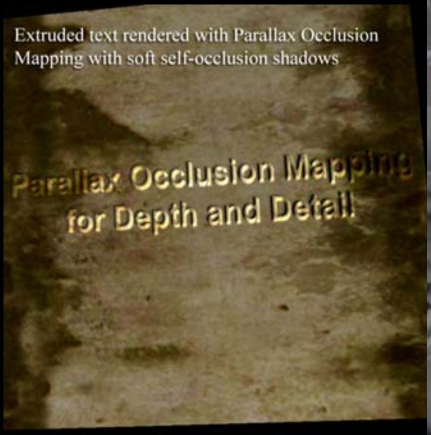
SIGGRAPH2006

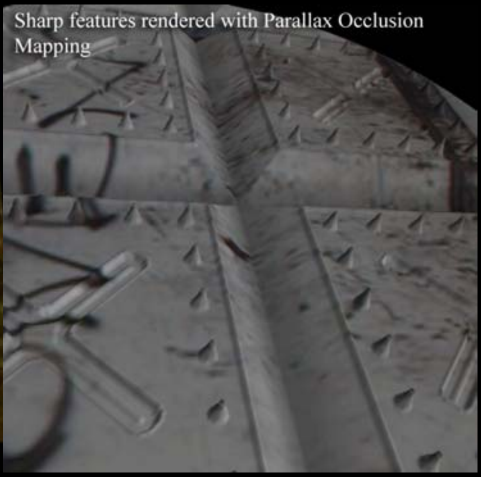
- Since the computation is in tangent space, the approach can be used with any surfaces
  - Works equally well on curved objects
  - Beware of silhouettes
- If vertex curvature can be encoded into vertex data
  - Extend current algorithm to use that data to improve height-field intersection using the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



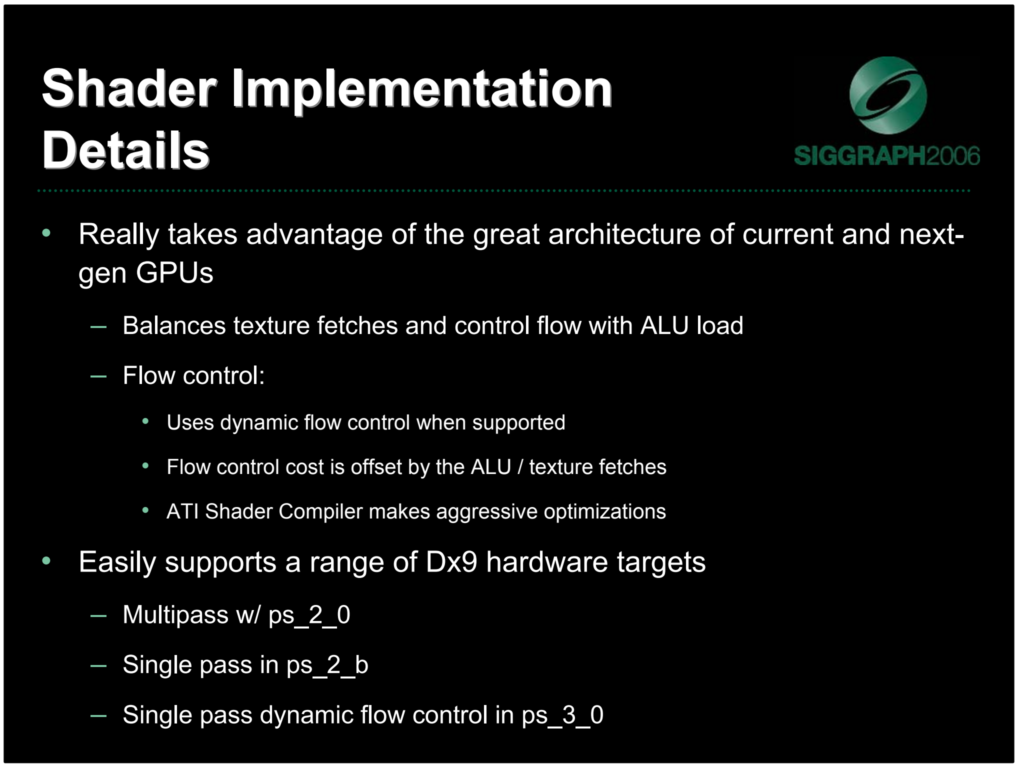
Able to Handle Difficult Cases

SIGGRAPH2006


Extruded text rendered with Parallax Occlusion Mapping with soft self-occlusion shadows
Parallax Occlusion Mapping for Depth and Detail



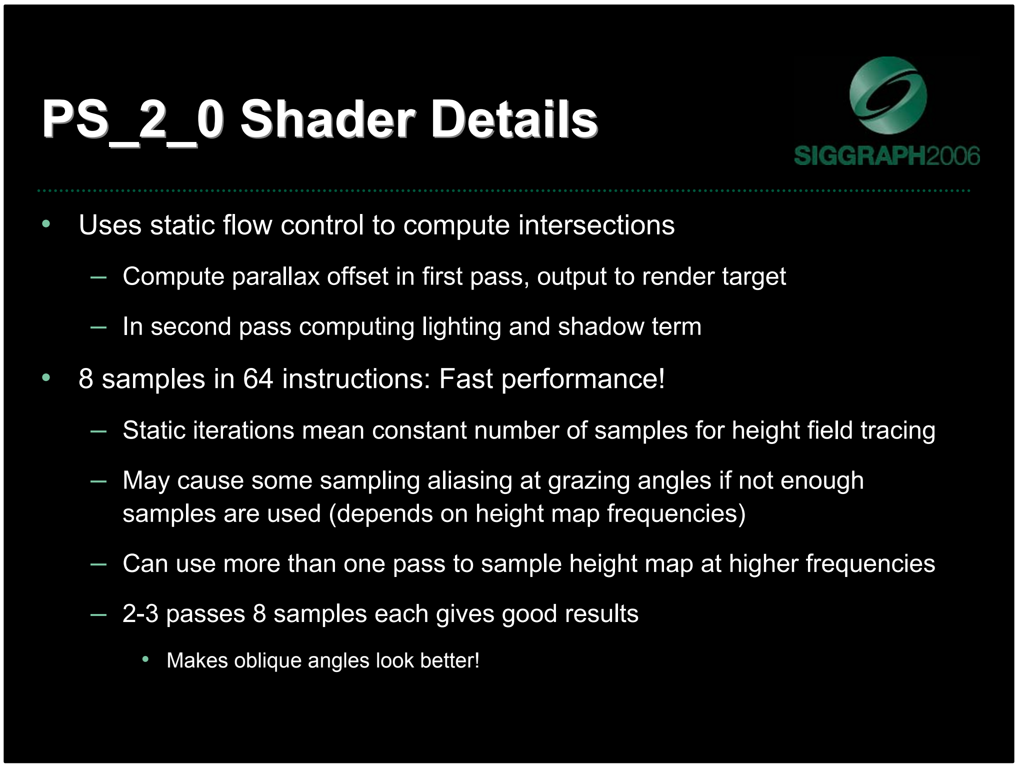
The parallax occlusion mapping technique provides the ability to render such traditionally difficult displacement mapping cases such as raised text or objects with very fine features. In order to render the same objects interactively with equal level of detail, the meshes would need an extremely detailed triangle subdivision (with triangles being nearly pixel-small), which is impractical even with the currently available GPUs.



Shader Implementation Details

SIGGRAPH2006

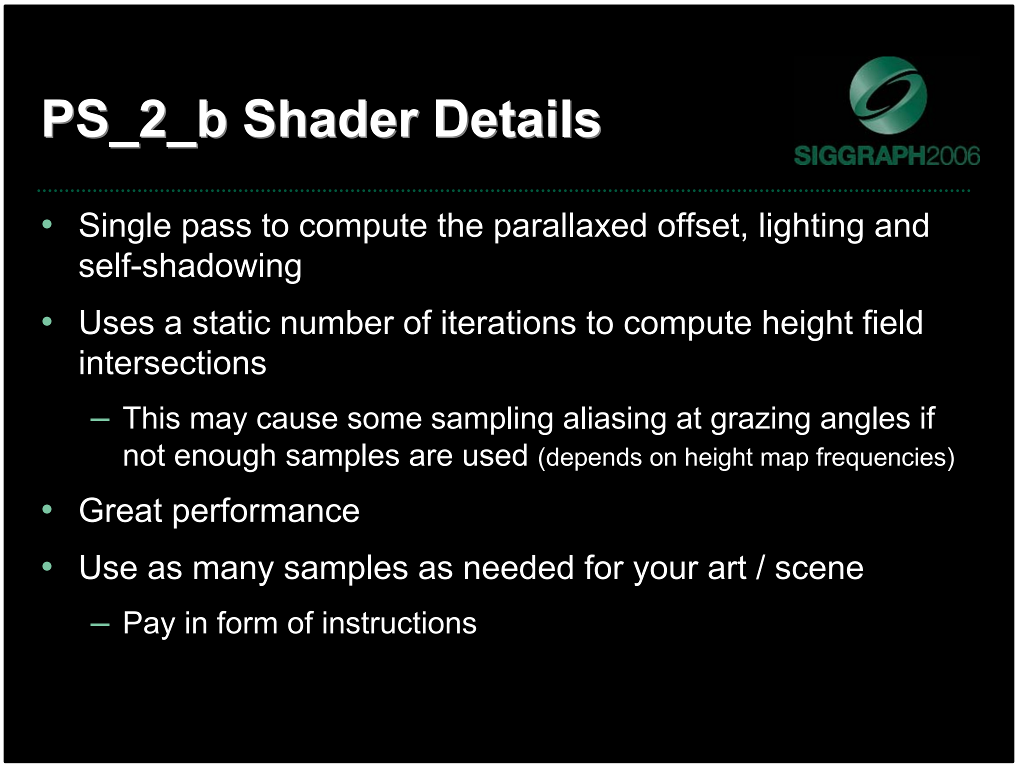
- Really takes advantage of the great architecture of current and next-gen GPUs
  - Balances texture fetches and control flow with ALU load
  - Flow control:
    - Uses dynamic flow control when supported
    - Flow control cost is offset by the ALU / texture fetches
    - ATI Shader Compiler makes aggressive 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

SIGGRAPH2006

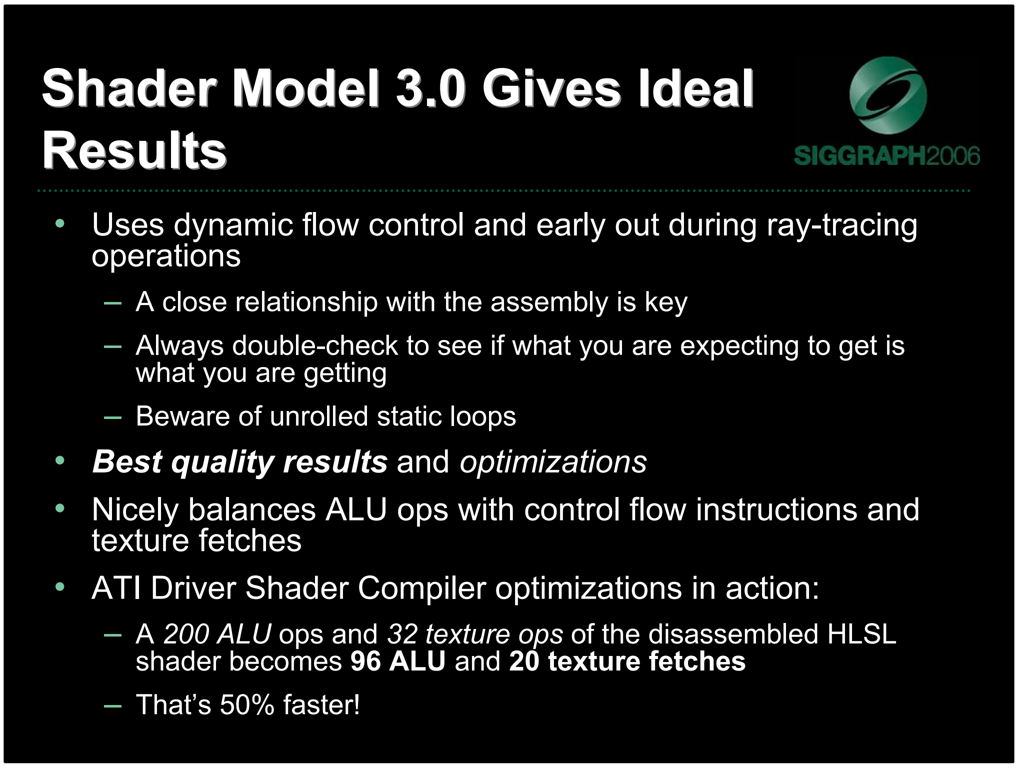
- Uses static flow control to compute intersections
  - Compute parallax offset in first pass, output to render target
  - In second pass computing lighting and shadow term
- 8 samples in 64 instructions: Fast performance!
  - Static iterations mean constant number of samples for height field tracing
  - May cause some sampling aliasing at grazing angles if not enough samples are used (depends on height map frequencies)
  - 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

SIGGRAPH2006

- 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 (depends on height map frequencies)
- Great performance
- Use as many samples as needed for your art / scene
  - Pay in form of instructions



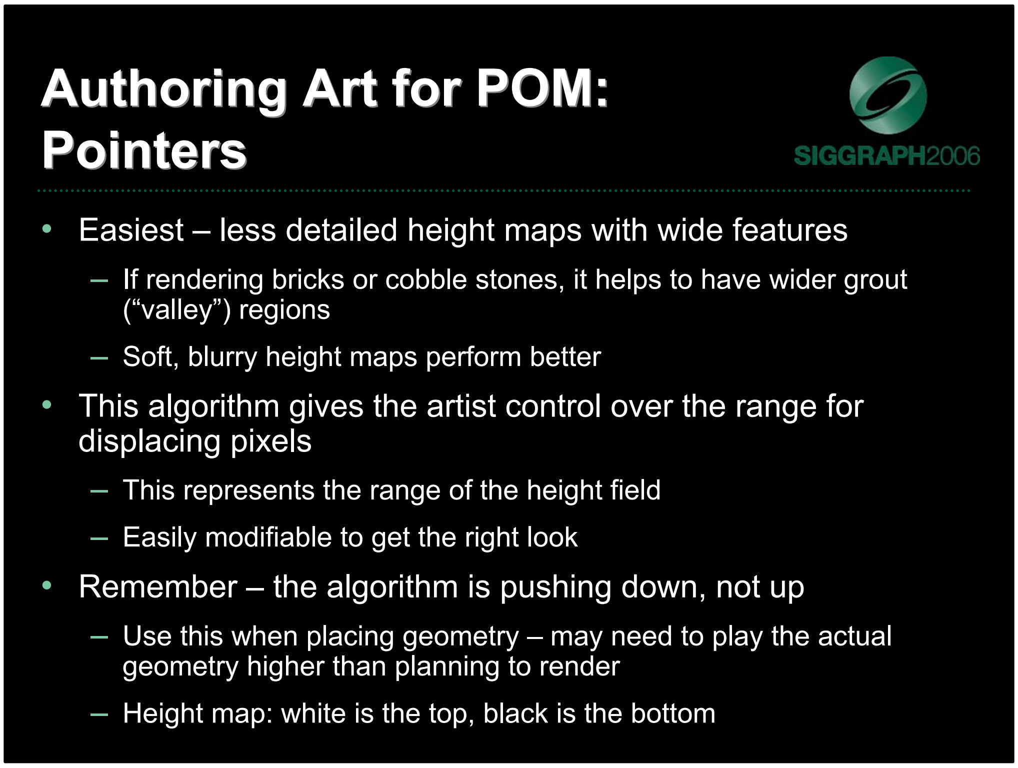
Shader Model 3.0 Gives Ideal Results

SIGGRAPH2006

- 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!

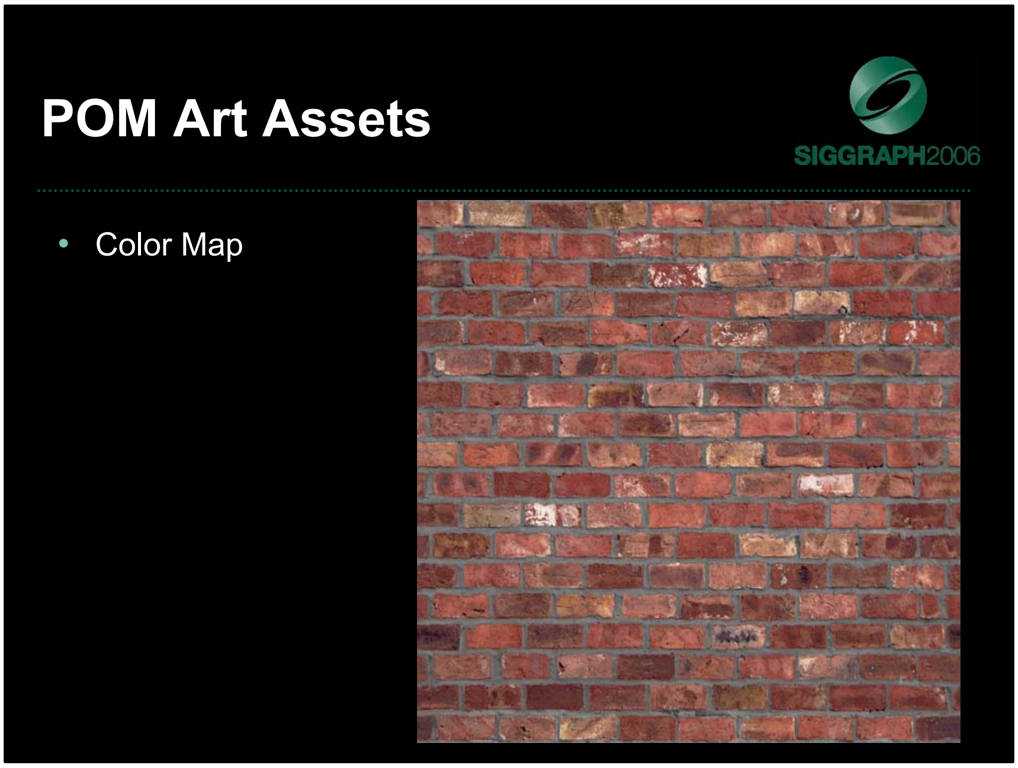
Uses dynamic flow control and early out during ray-tracing operations. Note: dynamic flow control in HLSL can be tricky to achieve. Develop in close relationship with the assembly – always double-check to see if what you are expecting to get is what you are getting. Beware of unrolled static loops. All of the important optimizations / quality improvements happen here (in SM 3.0):

•Nicely balances ALU ops with control flow instructions and texture fetches



Authoring Art for POM: Pointers

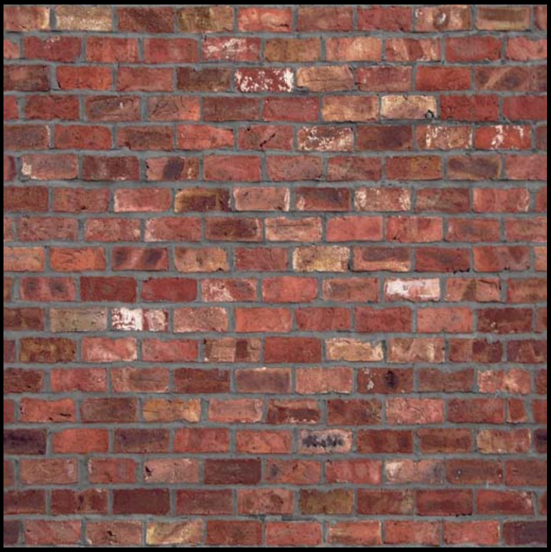
- Easiest – less detailed height maps with wide features
  - If rendering bricks or cobble stones, it helps to have wider grout (“valley”) regions
  - Soft, blurry height maps perform better
- This algorithm gives the artist control over the range for displacing pixels
  - This represents the range of 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

SIGGRAPH2006

- Color Map



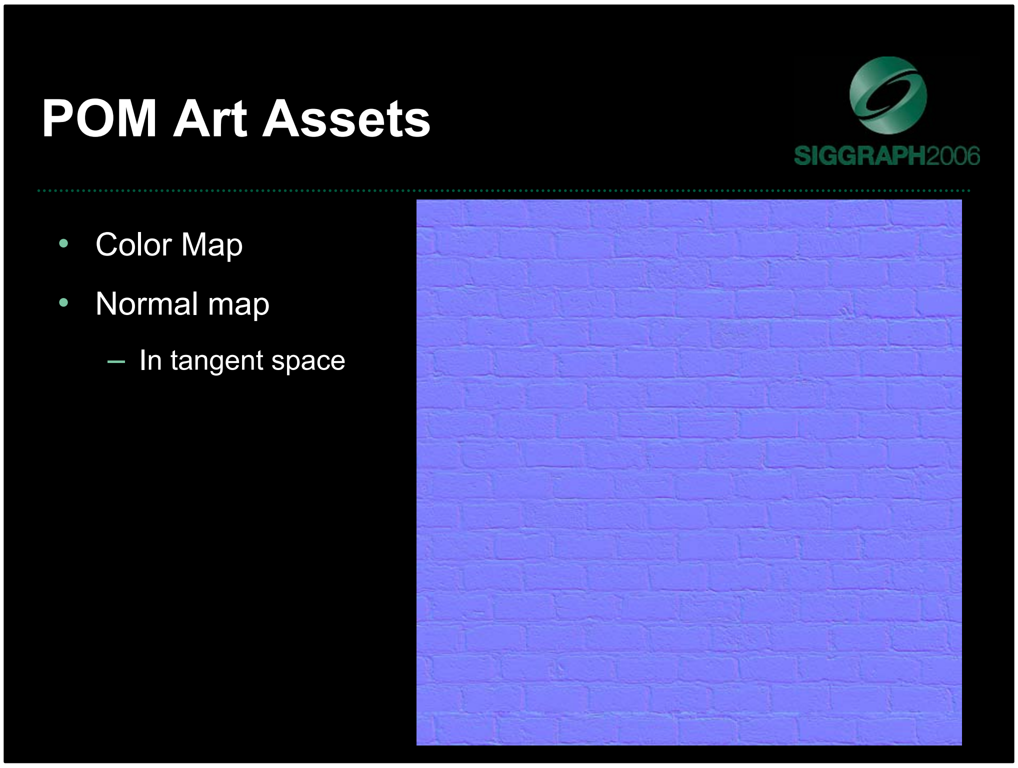
Required art assets:

•Color Map (Obviously)

•Normal Map (Must be a Tangent Space normal map, All computations are done in tangent space, so the shader could be applied to any surface. The shader derives all SHADING (self shadowing) information from the normal map).

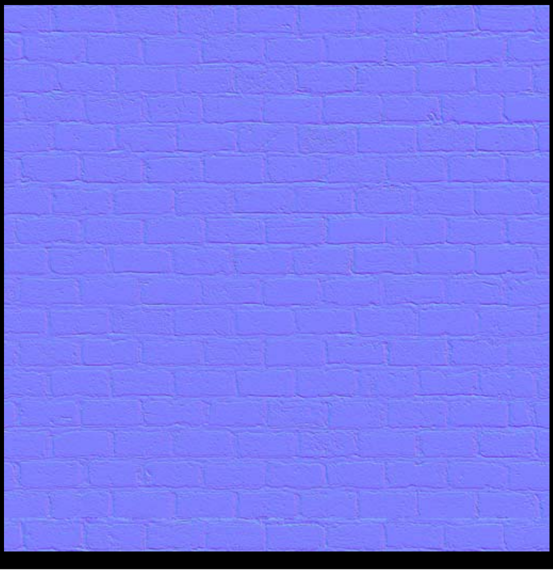
•Height Map (8-bit (grayscale), this map encodes the displacement info)

That's it! Minimal increase in memory usage. Only a small increase in memory footprint over traditional normal mapping technique. Recommend stuffing this into an available channel of one of your RGB textures (colormap). Either manually (by artists) or during export/pre-process stage. Considering that POM was a showcase feature of The Toy Shop demo we invested in high quality maps. We used 2048x2048 for maximum visual quality.



POM Art Assets

- Color Map
- Normal map
  - In tangent space



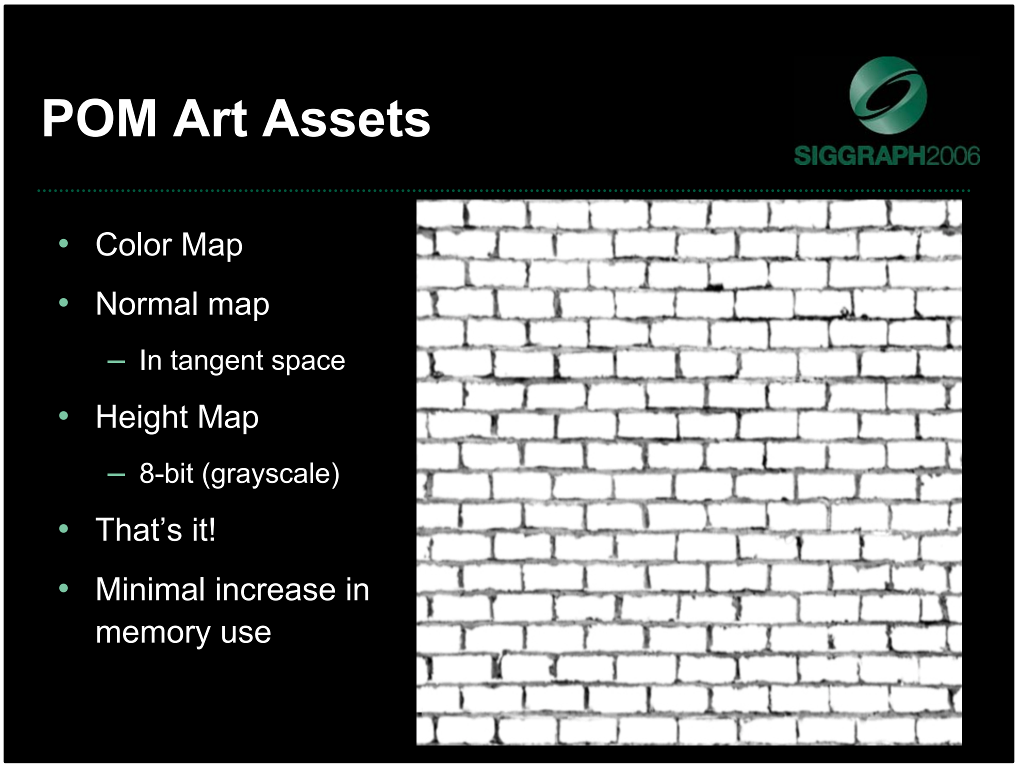
Required art assets:

•Color Map (Obviously)

•Normal Map (Must be a Tangent Space normal map, All computations are done in tangent space, so the shader could be applied to any surface. The shader derives all SHADING (self shadowing) information from the normal map).

•Height Map (8-bit (grayscale), this map encodes the displacement info)

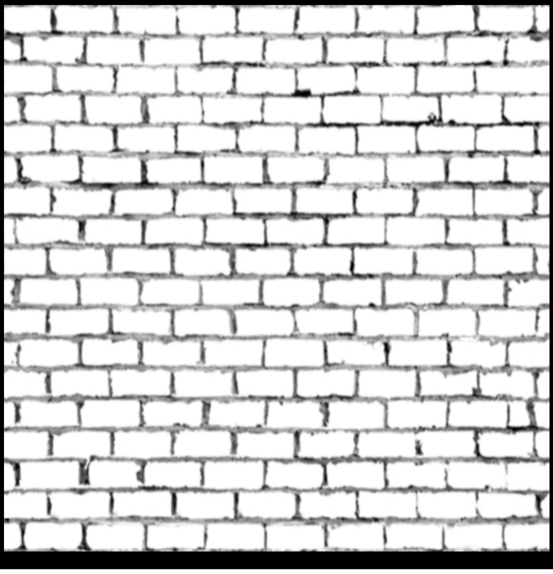
That's it! Minimal increase in memory usage. Only a small increase in memory footprint over traditional normal mapping technique. Recommend stuffing this into an available channel of one of your RGB textures (colormap). Either manually (by artists) or during export/pre-process stage. Considering that POM was a showcase feature of The Toy Shop demo we invested in high quality maps. We used 2048x2048 for maximum visual quality.



POM Art Assets

SIGGRAPH2006

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



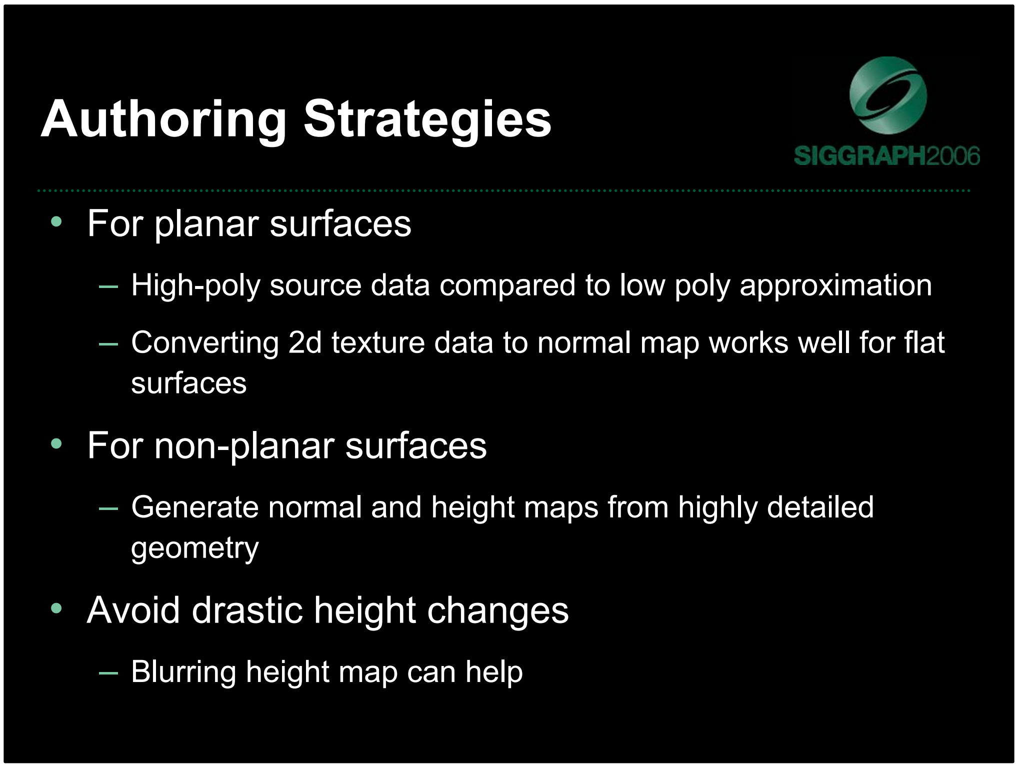
Required art assets:

•Color Map (Obviously)

•Normal Map (Must be a Tangent Space normal map, All computations are done in tangent space, so the shader could be applied to any surface. The shader derives all SHADING (self shadowing) information from the normal map).

•Height Map (8-bit (grayscale), this map encodes the displacement info)

That’s it! Minimal increase in memory usage. Only a small increase in memory footprint over traditional normal mapping technique. Recommend stuffing this into an available channel of one of your RGB textures (colormap). Either manually (by artists) or during export/pre-process stage. Considering that POM was a showcase feature of The Toy Shop demo we invested in high quality maps. We used 2048x2048 for maximum visual quality.



Authoring Strategies

SIGGRAPH2006

- 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

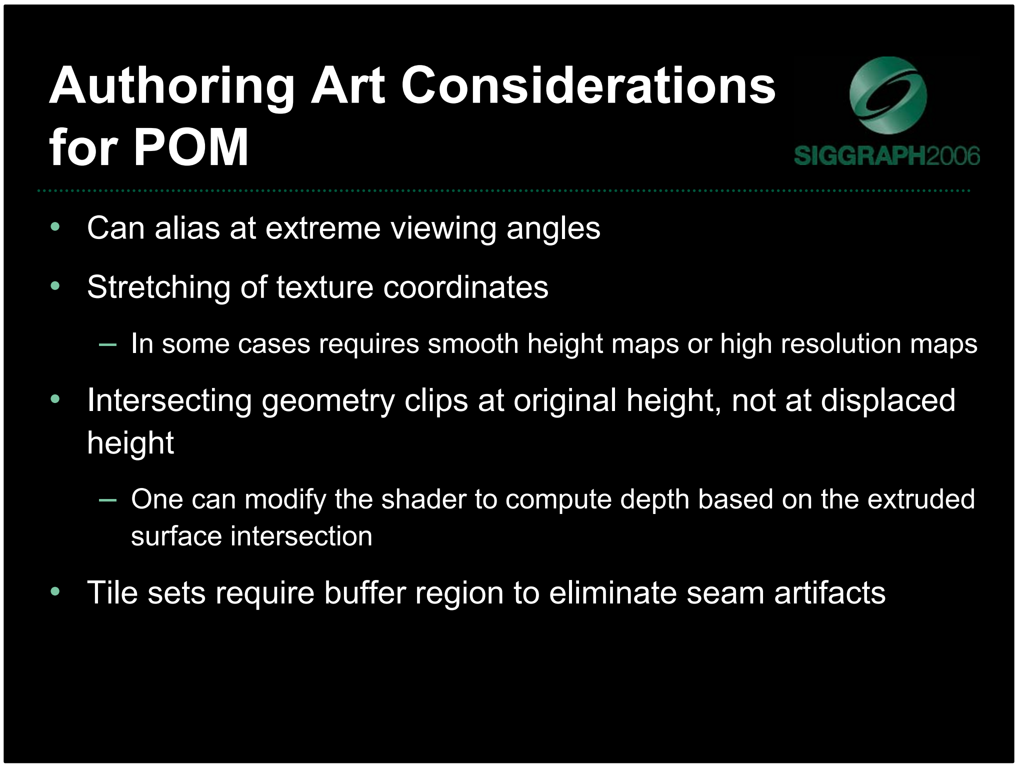
Planar Surfaces: Either method is fine and will generate good results

Non-planar Surfaces:

•For “physically correct” results you must generate your tangent space normal maps from geometry

•You can apply texture derived normal and height maps, but you will not get the best results. It won’t completely break… you will get something parrallax-ish… But generally, not the best idea

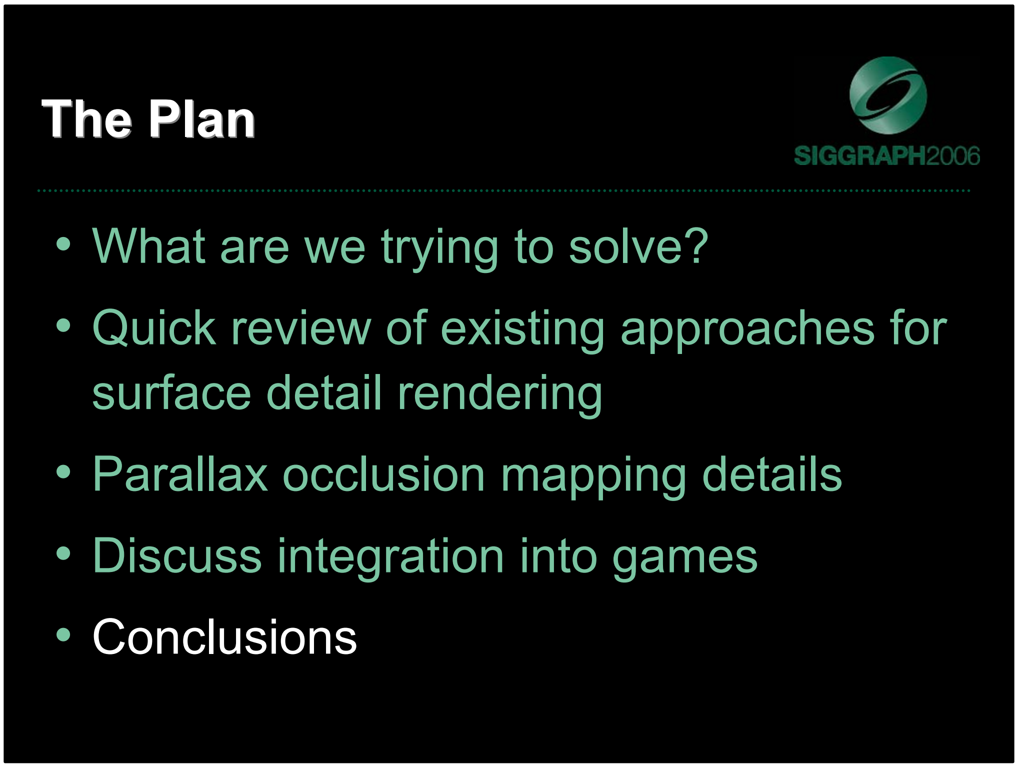
•Avoid drastic height changes: This relates back to limitation of “stretching” of texture coordinated. The more gradual the height change… the less noticeable this will be. If you have a height map that is causing texture stretching… try blurring it in the problematic areas.



Authoring Art Considerations for POM

SIGGRAPH2006

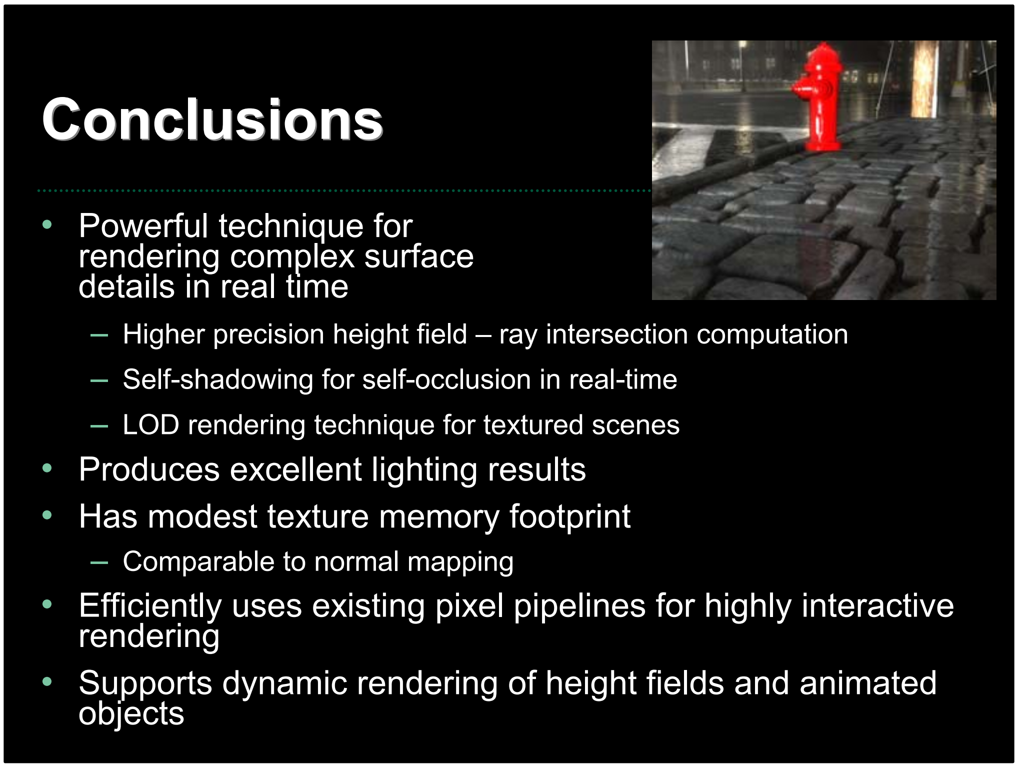
- Can alias at extreme viewing angles
- Stretching of texture coordinates
  - In some cases requires smooth height maps or high resolution maps
- Intersecting geometry clips at original height, not at displaced height
  - One can modify the shader to compute depth based on the extruded surface intersection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



The Plan

SIGGRAPH2006

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



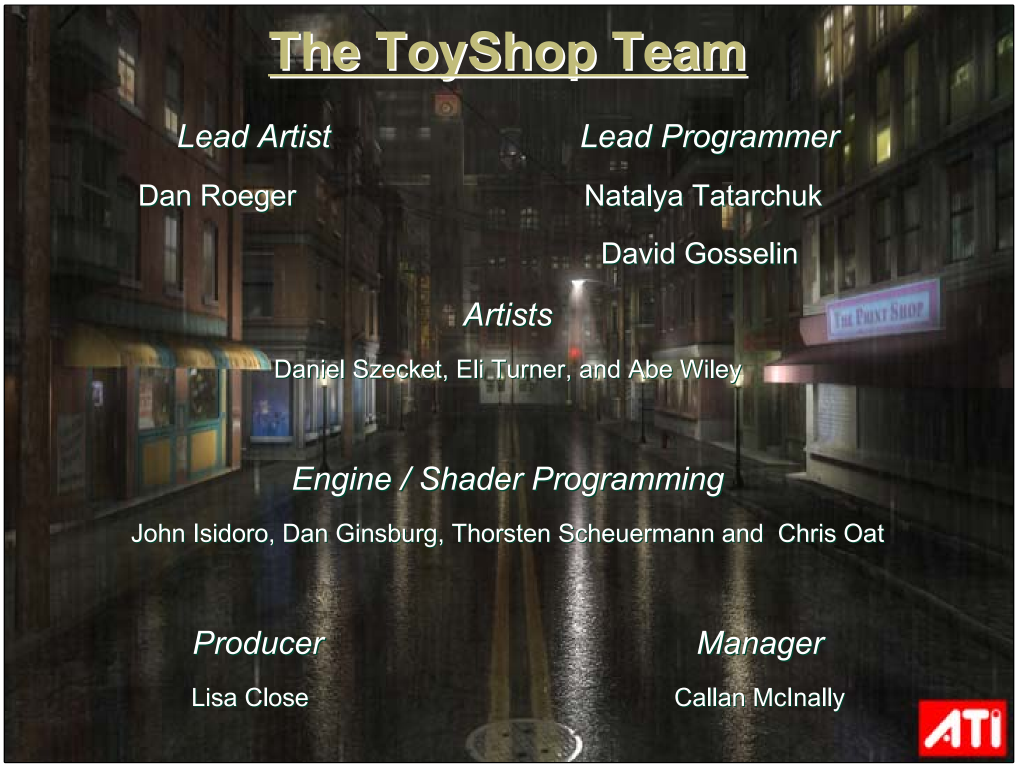
Conclusions

- 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

We have presented a novel **technique** for rendering highly detailed surfaces under varying light conditions. We have described an efficient algorithm for **computing intersections** of the height field profile with rays with high precision. We presented a algorithm for generating **soft shadows** during occlusion computation. An **automatic level-of-detail control system** is used by our approach to control shader complexity efficiently. A benefit of our approach lies in a **modest texture memory footprint,** comparable to normal mapping. It requires only an grayscale texture in addition to the normal map. Our technique is designed to take advantage of the **GPU programmable pipeline** resulting in highly interactive frame rates. It efficiently uses the **dynamic flow control feature** to improve resulting visual quality and optimize rendering speed. Additionally, this algorithm is designed to easily support **dynamic rendering to height fields** for a variety of interesting effects. Algorithms based on precomputed quantities are not as flexible and thus are limited to the static height fields

Acknowledgements

- Zoe Brawley, *Relic Entertainment*
  - **Brawley, Z. and Tatarchuk, N. 2004. Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing. *ShaderX^3: Advanced Rendering Techniques with DirectX and OpenGL.* Charles River Media, Cambridge, MA**
- Pedro Sander, for **ScreenSpace** screensaver work and related slides
- The ScreenSpace screensaver team



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

Truly a team effort of which we are all very proud of.

Reference Material

- Demos, GDC presentations, papers and technical reports, and related materials: www.ati.com/developer
- Downloadable publications and videos from ATI Research
  - http://www.ati.com/developer/techreports.html
  - Tatarchuk, N. Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows. *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.* Redwood City, CA
  - 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
- Parallax Occlusion Mapping DirectX 9.0c SDK sample:
  http://msdn.microsoft.com/directx/

Questions?

SIGGRAPH2006

natasha@ati.com