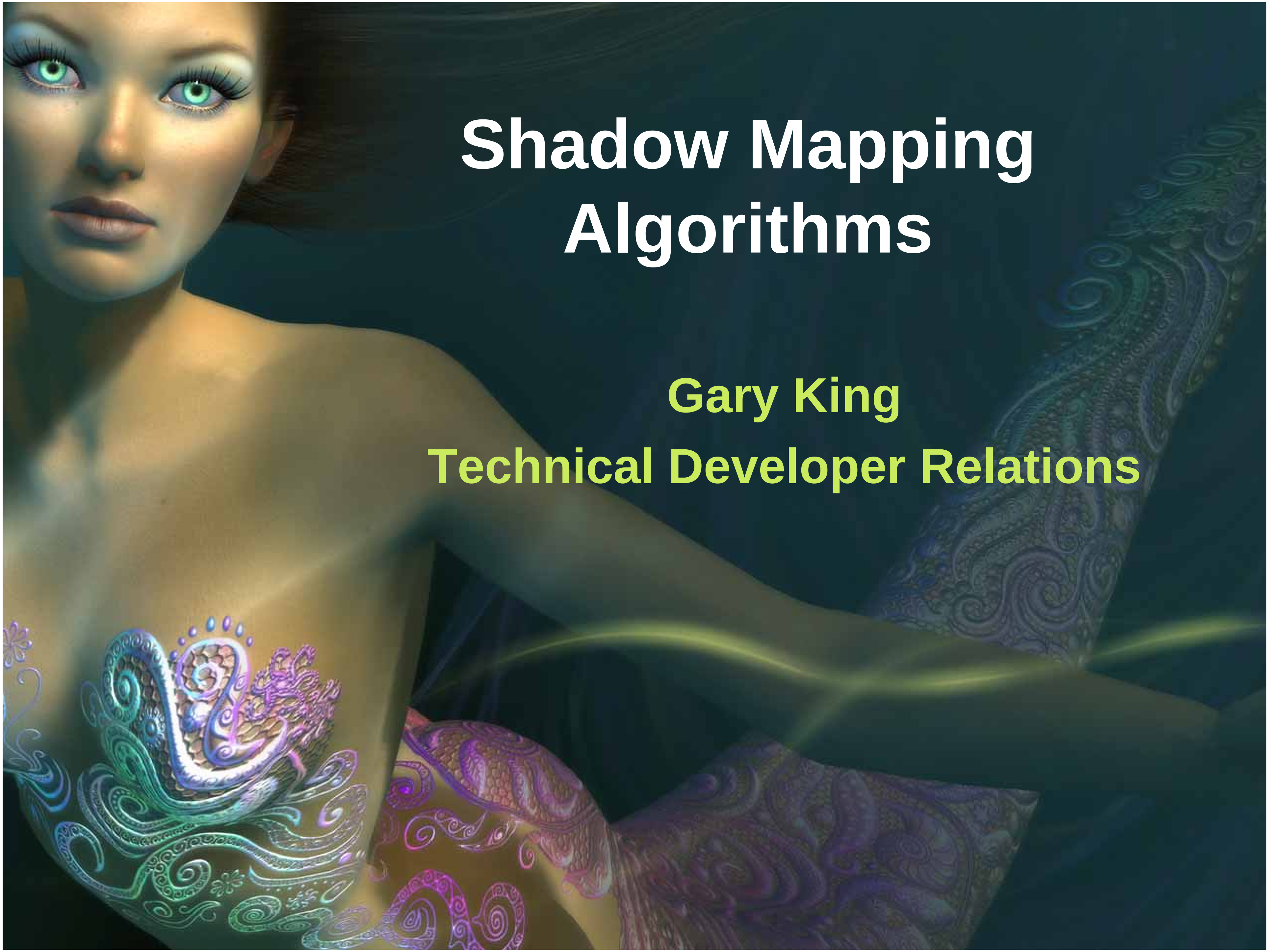


Shadow Mapping Algorithms

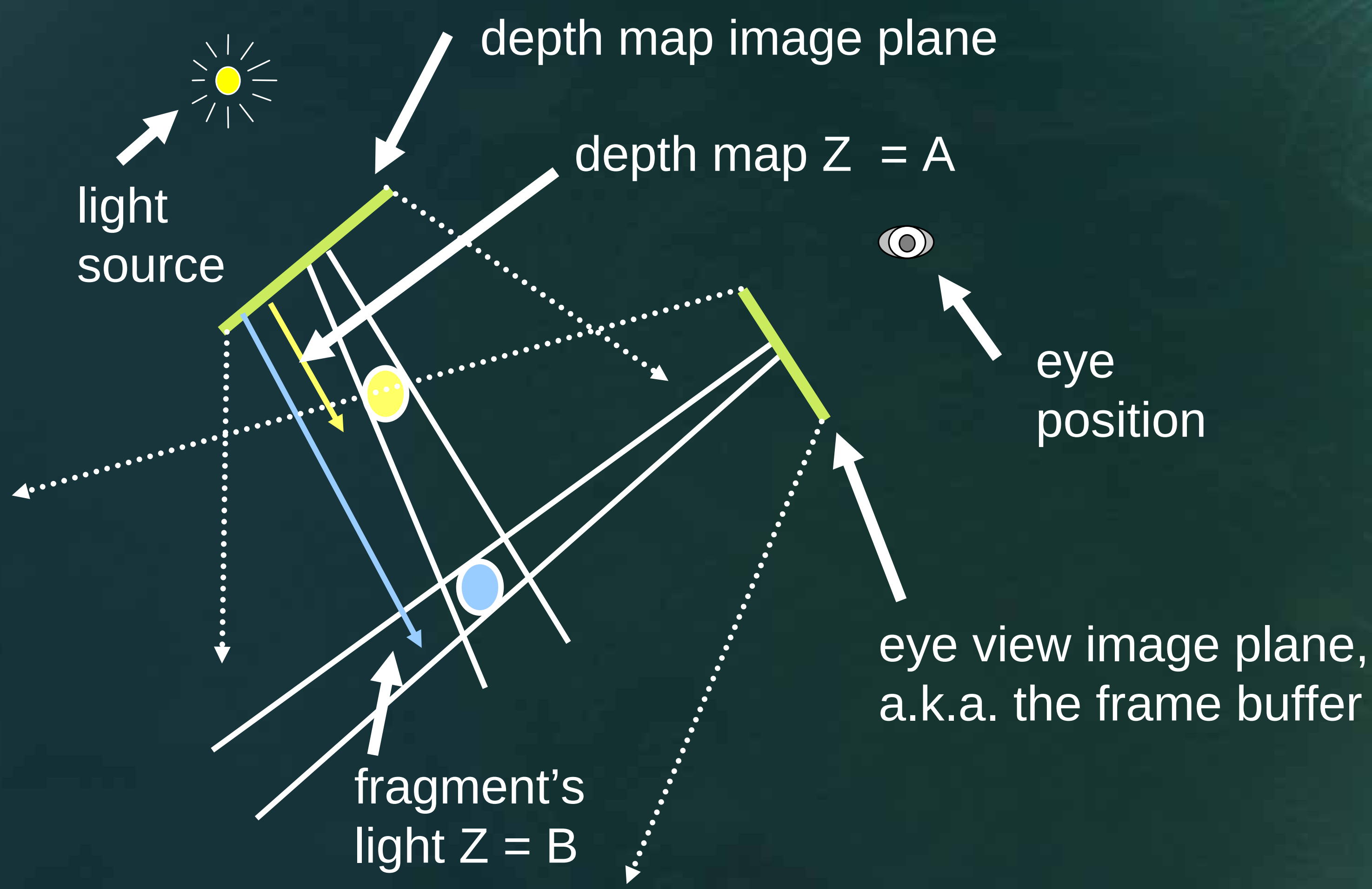
Gary King
Technical Developer Relations

Shadow Mapping Introduction

- Casting Curved Shadows on Curved Surfaces, Lance Williams
- Render scene from light's point of view
  - Closest objects to light are occluders
- Render scene from eye's point of view
  - Project all fragments into shadow map
  - If depth > depth(shadow), object is in shadow

©2004 NVIDIA Corporation. All rights reserved.

Shadow Mapping Diagram



©2004 NVIDIA Corporation. All rights reserved.

Shadow Mapping on GPUs

- On Radeon 9500+
  - Floating-point textures (R32F)
  - Pixel shader filtering and comparison
- On GeForce 3+
  - Native shadow map support (16 and 24-bit integer)
  - 2x2 bilinear percentage closer filtering for free
  - 2x speed rendering on GeForceFX and later GPUs
    - 32ppc on GeForce 6800 GT

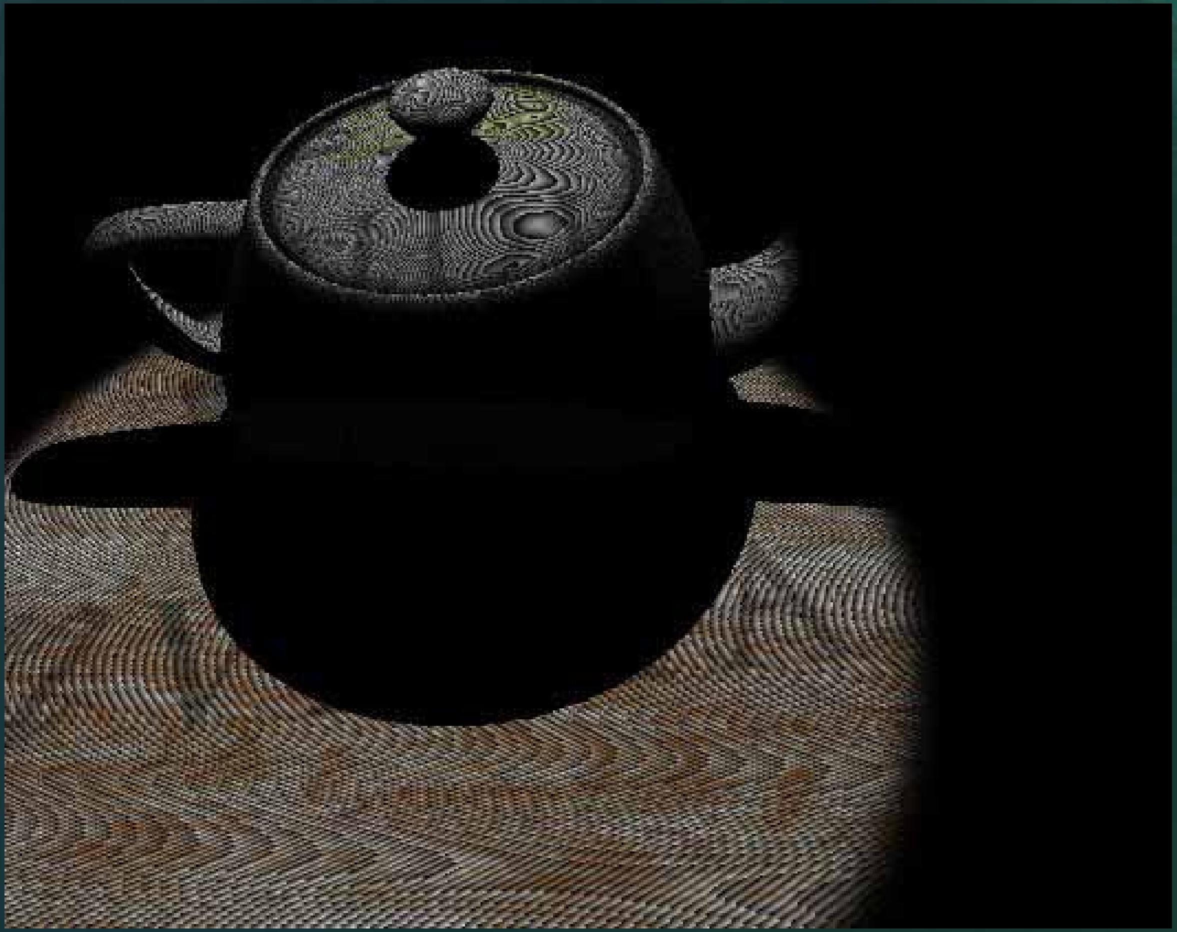
©2004 NVIDIA Corporation. All rights reserved.

Simple, Right?

- Shadow maps are a nice, elegant, easy-to-implement, technique for shadowing…
- Except for…

©2004 NVIDIA Corporation. All rights reserved.

Shadow Acne



- Surface incorrectly self-shadows

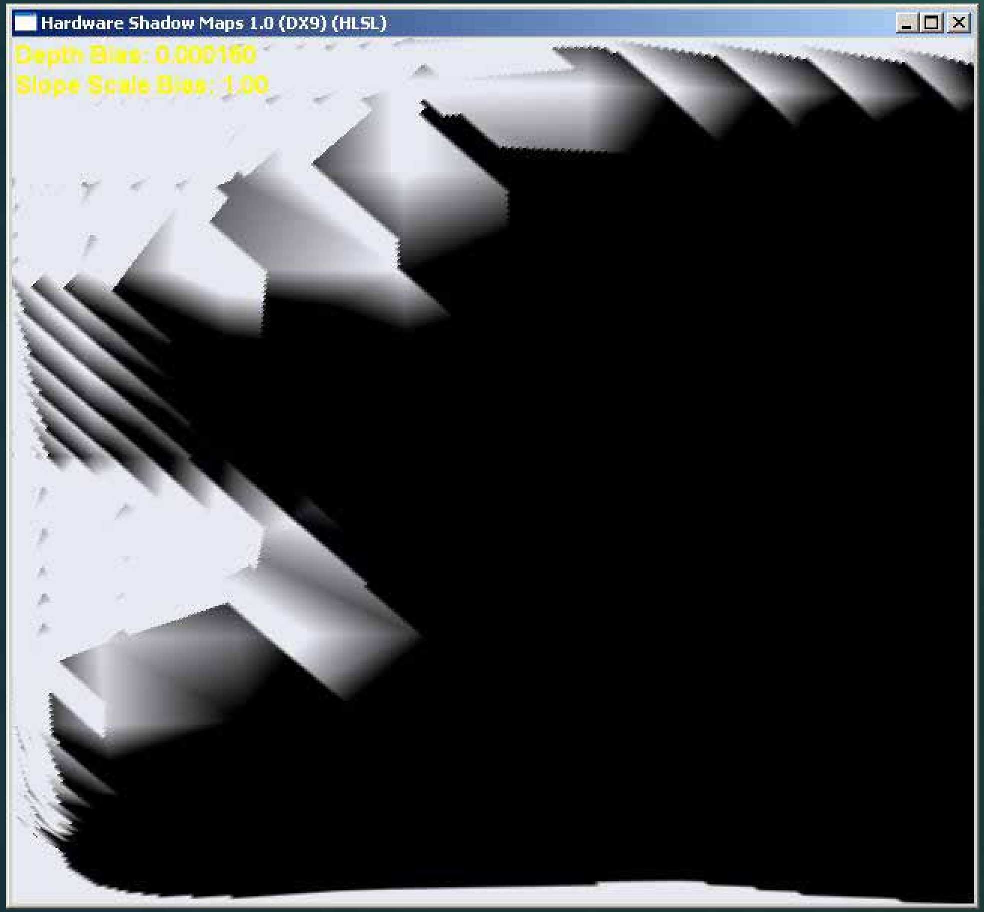
©2004 NVIDIA Corporation. All rights reserved.

Perspective Aliasing

- Blocky shadows cast onto nearby receivers

©2004 NVIDIA Corporation. All rights reserved.

Projective Aliasing



- Streaked shadows cast onto ~parallel receivers

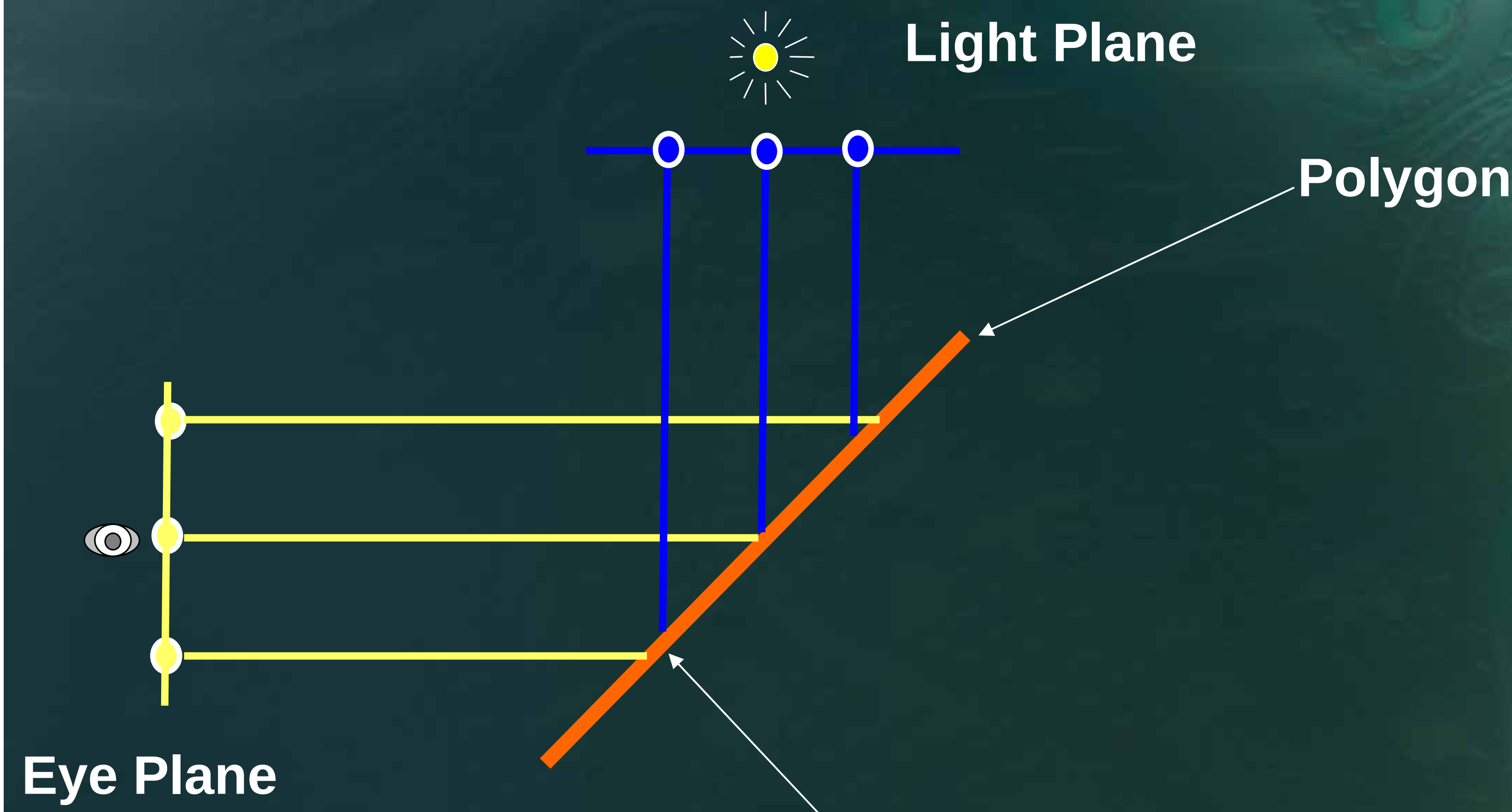
©2004 NVIDIA Corporation. All rights reserved.

Omnidirectional Lights

- Point lights cast shadows in all directions

©2004 NVIDIA Corporation. All rights reserved.

Source of Shadow Acne



Eye's distance > shadow depth

Incorrect Self-Shadowing

©2004 NVIDIA Corporation. All rights reserved.

Depth Bias

- Add epsilon to shadow depths, such that

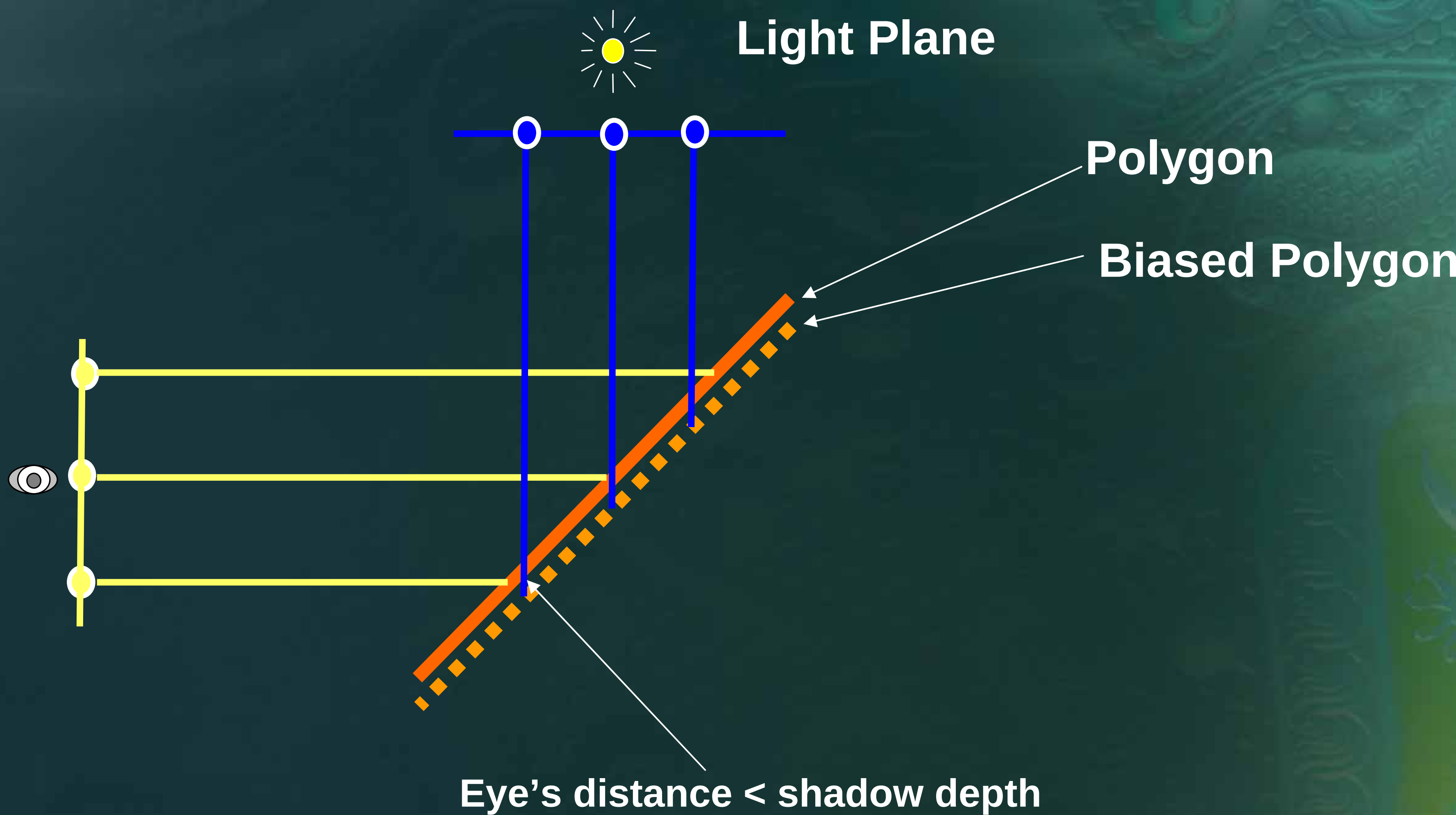
  `eye depth <= shadow depth`

  For all fragments from the same polygon

- How much epsilon is needed?
  - Art, not science
  - Movie houses spend hours/days tweaking shadow parameters frame-by-frame to avoid artifacts

©2004 NVIDIA Corporation. All rights reserved.

Depth Bias Diagram



Shadow acne eliminated

©2004 NVIDIA Corporation. All rights reserved.

Using Depth Bias in Direct3D9

- Ignore DirectX SDK Depth Bias documentation

- Depth bias is a floating point number that is added to the interpolated fragment Z value
  - Can be positive or negative

- To bias by 1 LSB of a 24-bit depth buffer, use

```
float fBias = 1.f / 16777215.f;
DWORD dwBias = *(DWORD*)&fBias;
SetRenderState(D3DRS_DEPTHBIAS, dwBias);
```

©2004 NVIDIA Corporation. All rights reserved.

Resampling Error

- Projected pixels are snapped to nearest texel
  - This is a window of [+/-0.5,+/-0.5] in the texture
  - Up to $| 0.5\, \partial z/\partial s | + | 0.5\, \partial z/\partial t |$
- Since polygons are planar, bilinearly filtering the depth values is *almost* correct
  - $\partial z/\partial s$ and $\partial z/\partial t$ can be inferred from nearby texels
  - Bilinear weights give ∂s and ∂t, **BUT**
  - Polygon edges break the planar assumption
  - Perspective Z is hyperbolic, not linear

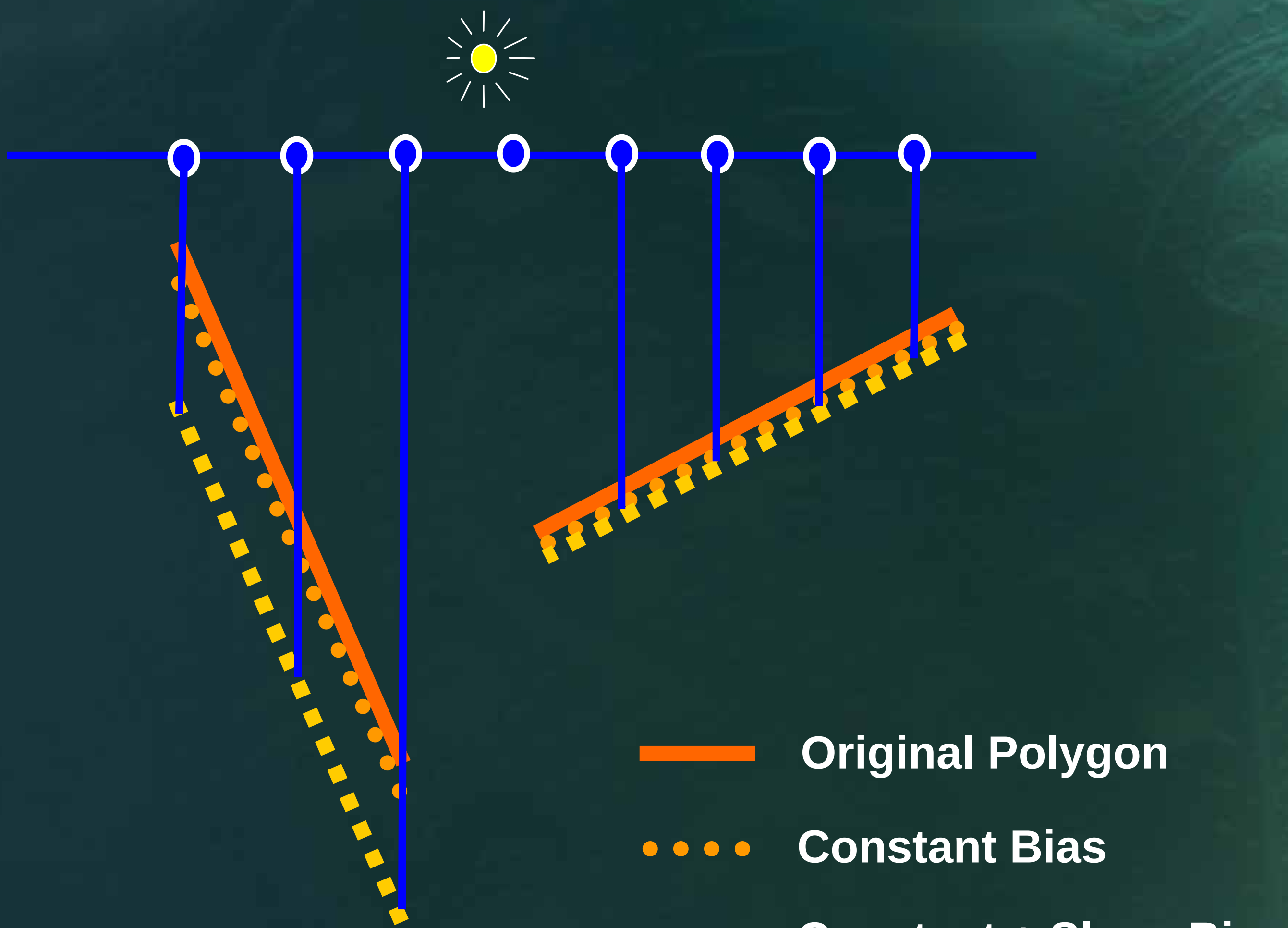
©2004 NVIDIA Corporation. All rights reserved.

Slope-Scale Depth Bias

- Instead, bias the shadow depth based on the post-projective Z-slope
- Slope-scale depth bias lets GPU compute this
  - In D3D, use `D3DRS_SLOPESCALEDEPTHBIAS`
- Total bias is **m*SLOPESCALE + DEPTHBIAS**
  - Where $m = \max(| \partial z/\partial x | , | \partial z/\partial y |)$

©2004 NVIDIA Corporation. All rights reserved.

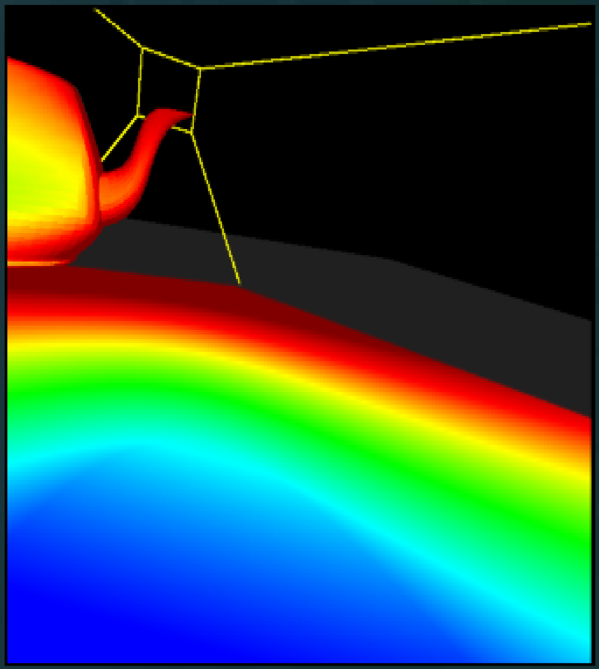
Slope-Scale Depth Bias Diagram



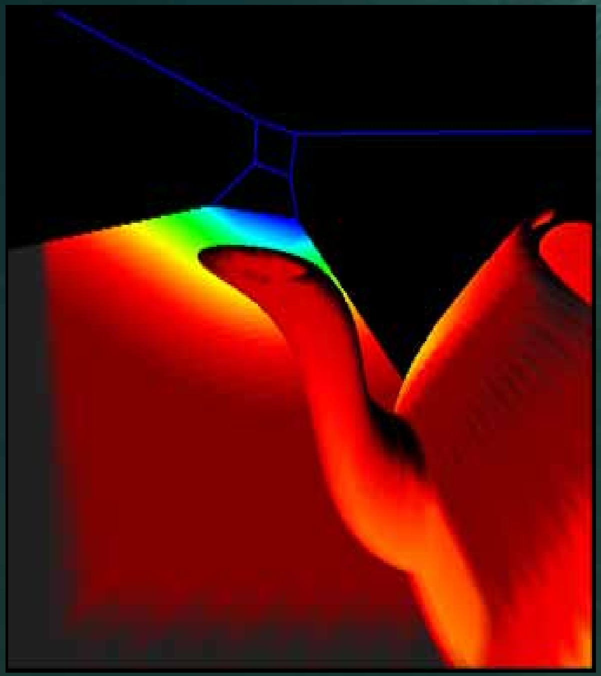
©2004 NVIDIA Corporation. All rights reserved.

Source of Perspective Aliasing

Eye View



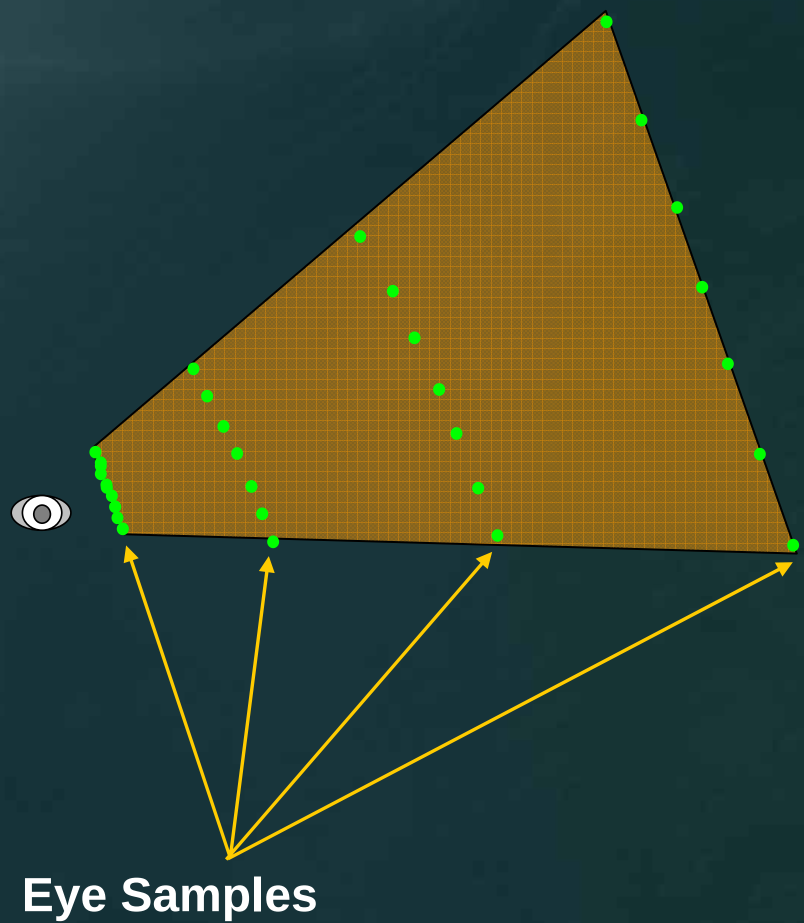
Shadow map



Blue and green region occupy significant portion of eye's view, but receive comparatively few shadow texels

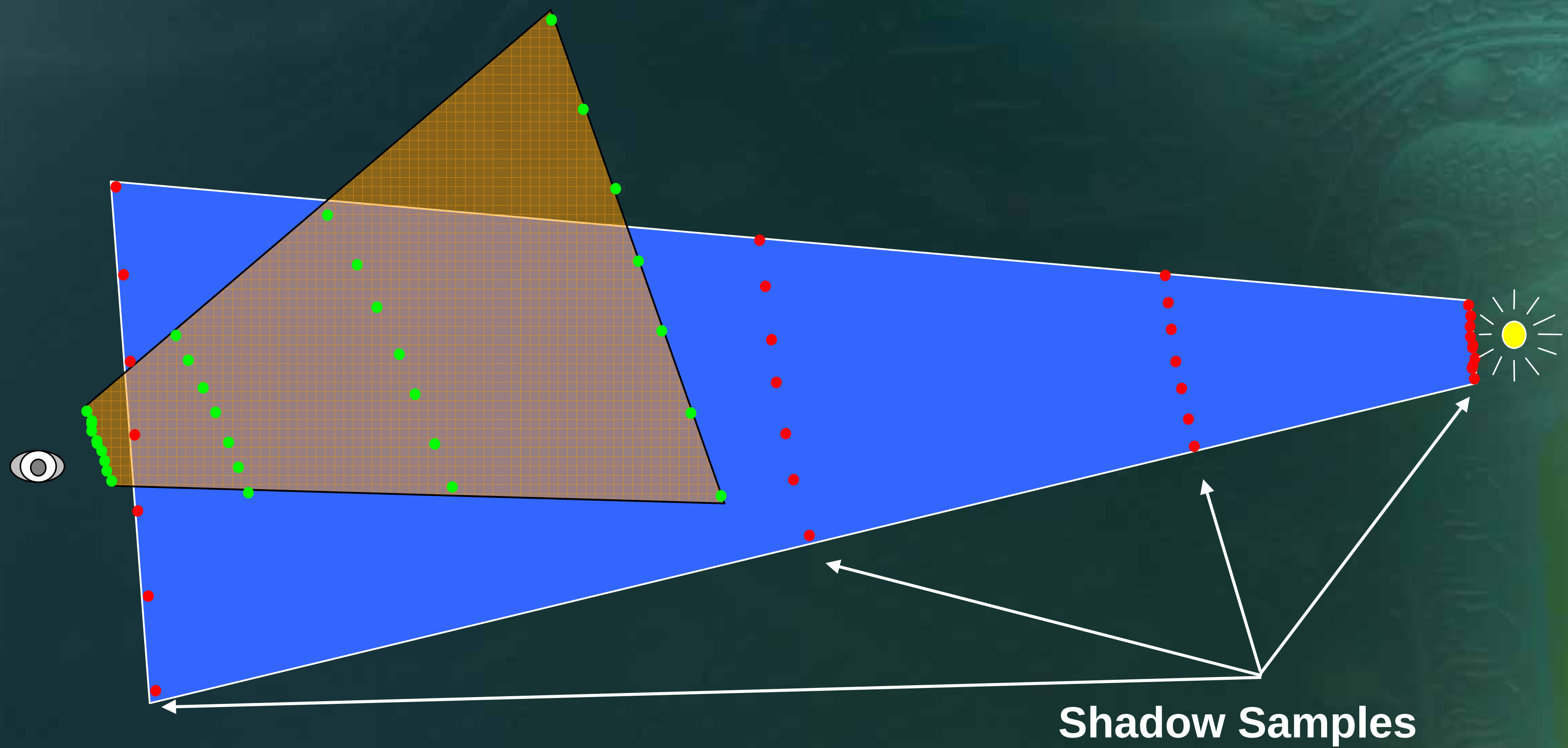
©2004 NVIDIA Corporation. All rights reserved.

Perspective Aliasing in 2D



©2004 NVIDIA Corporation. All rights reserved.

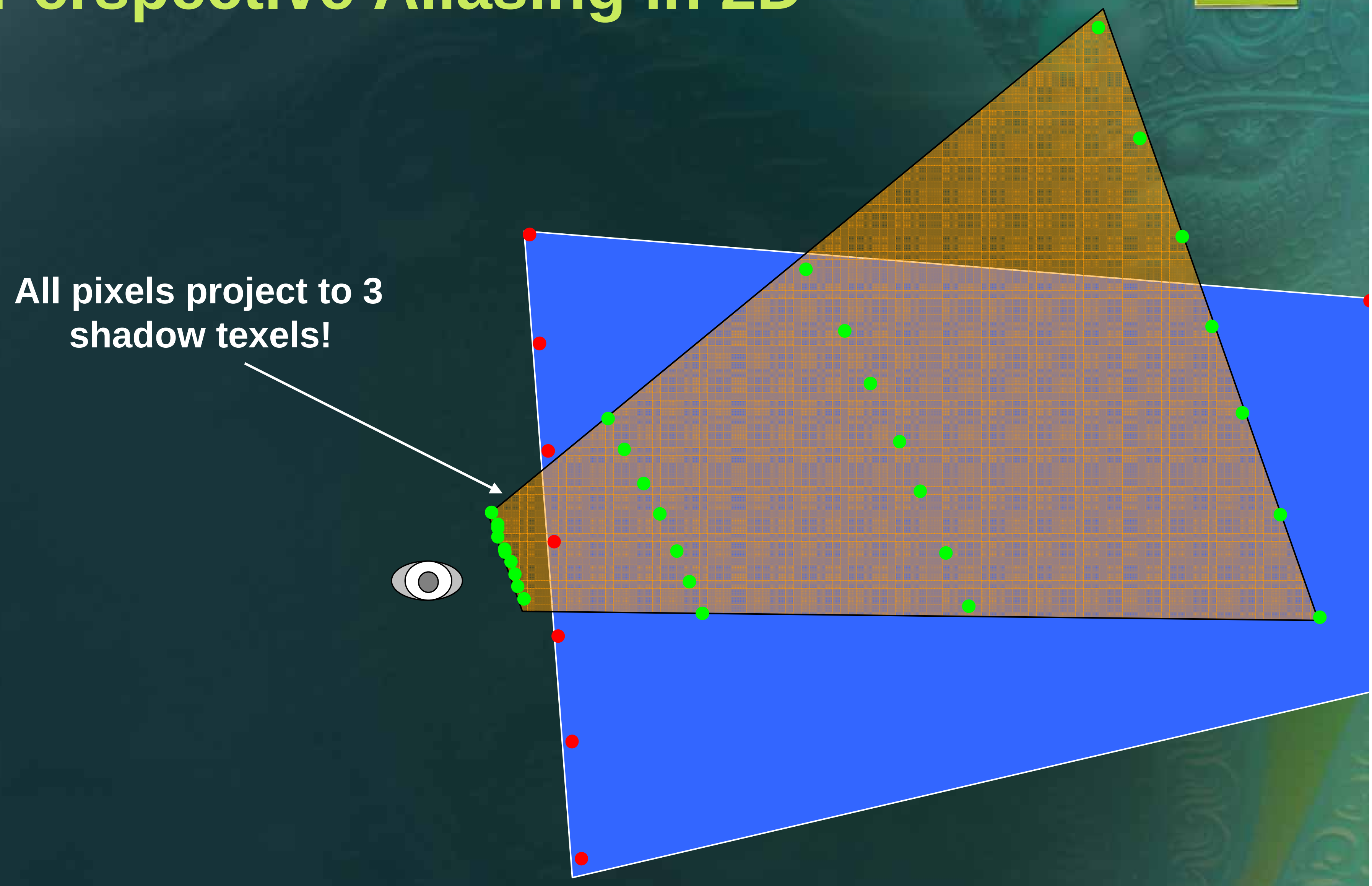
Perspective Aliasing in 2D



Duelling Frusta : Each frustum points toward the other, resulting in perfectly nonoptimal shadow maps

©2004 NVIDIA Corporation. All rights reserved.

Perspective Aliasing in 2D



©2004 NVIDIA Corporation. All rights reserved.

The Sea of Aliasing "Solutions"

- Many ways proposed to reduce aliasing
  - Increase shadow map resolution
  - Unit cube clipping
  - Cascaded Shadow Maps (????)
  - Perspective Shadow Maps (Stamminger, 2002)
  - Light-Space PSMs (Wimmer, 2004)
  - Trapezoidal Shadow Maps (Martin, 2004)

  … Not counting filtering & soft shadow algorithms!

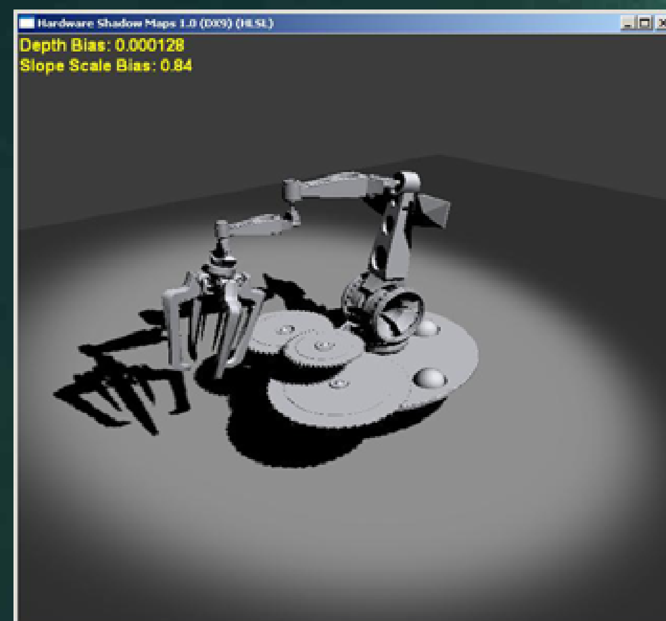
- What works, and what are the problems?

©2004 NVIDIA Corporation. All rights reserved.

Increasing Resolution

- More samples = less aliasing

128x128



512x512

- But how many samples is enough?

©2004 NVIDIA Corporation. All rights reserved.

Sample Density Requirements

- Ideally, we want at least one unique shadow sample for every pixel on the screen
  - Ray tracing and stencil shadow volumes give this
- So, we want to satisfy

```
Area(eyeProj * eyeView * P) <=
Area(lightProj * lightView * P)
```

  for all P, and all view combinations
- This is 22 degrees of freedom

©2004 NVIDIA Corporation. All rights reserved.

Increasing Resolution

- Not possible to satisfy this for arbitrary situations with one shadow map and one linear projection.
- But not all situations are arbitrary
  - Flashlights
    - `lightView*P ~ eyeView*P`
  - Static lights in enclosed environments
    - `Area(lightProj*lightView*P)` is bounded at level-design time
- When DoF can be reduced, resolution tweaking may be sufficient

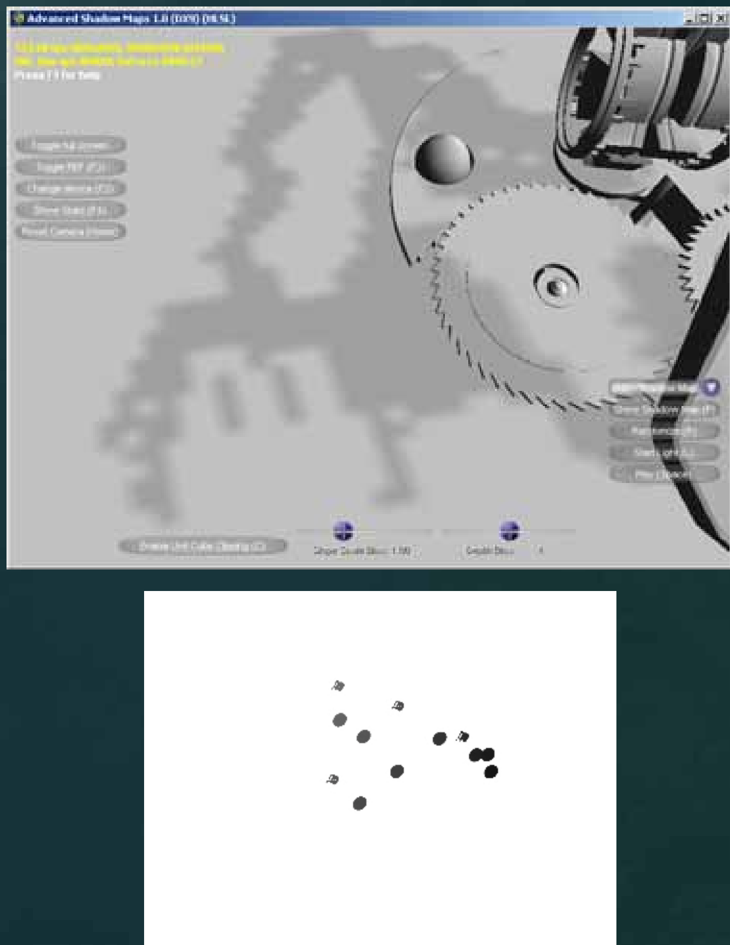
©2004 NVIDIA Corporation. All rights reserved.

Unit Cube Clipping

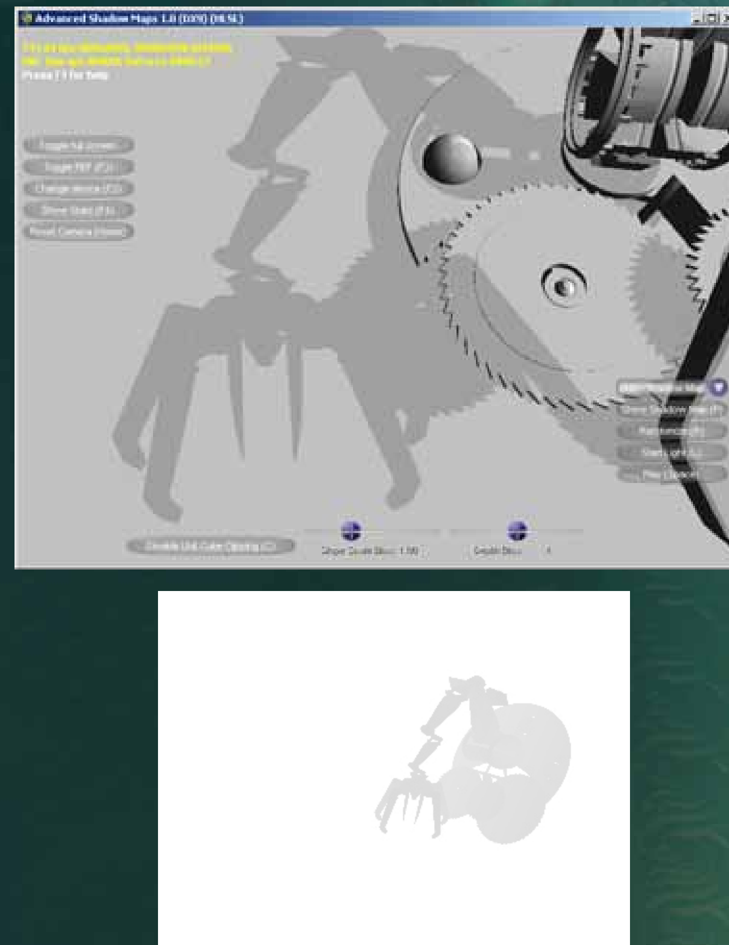
- Only focus shadow map where it is used
  - On the visible receivers
- For large light sources / scenes, this may be a small percentage of the total scene
- Shadow aliasing will depend on the viewer, but quality improvement can be substantial
  - Quality is always at least as good as *not* clipping

©2004 NVIDIA Corporation. All rights reserved.

Unit Cube Clipping



Without Clipping



With Clipping

©2004 NVIDIA Corporation. All rights reserved.

Unit Cube Clipping View Dependence

Viewer focused on small area

Viewer looking at distance

©2004 NVIDIA Corporation. All rights reserved.

Implementation

- For each light, determine which visible objects will receive shadows from it
- Build light projection matrix to maximize the shadow map area used by these objects
  - Transform receiver AABBs by unmodified `lightProj`
  - Merge AABBs into full bounding rectangle
  - Compute ortho transform to bias & scale resulting AABB to cover the entire shadow map
  - `lightProj’ = ortho * lightProj;`

©2004 NVIDIA Corporation. All rights reserved.

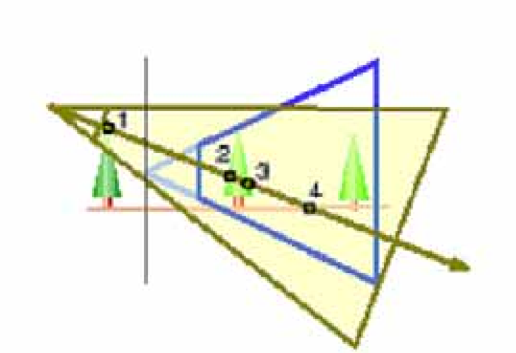
Cascaded Shadow Maps

- Satisfy area constraint using multiple maps
  - Extremely popular technique
- For each receiver in the scene, compute optimal `lightProj*lightView`, given `eyeView*P`.
- Many variants/trade-offs possible, to reduce number of maps required (at expense of quality)
- Handles duelling frusta and omnidirectional lights

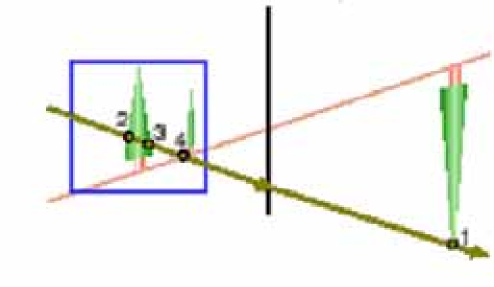
©2004 NVIDIA Corporation. All rights reserved.

PSMs & Variants

- Why not warp shadow map by view projection?
- Perspective transform makes objects close to viewer larger than objects farther away
  - Increasing number of shadow texels focused on them



Eye Space



Post-Projective Space

©2004 NVIDIA Corporation. All rights reserved.

Perspective Variants

- Perspective Shadow Maps
  - Stamminger & Drettakis, 2002
- Light-Space Perspective Shadow Maps
  - Wimmer, 2004
- Trapezoidal Shadow Maps & variants
  - Martin, 2004

©2004 NVIDIA Corporation. All rights reserved.

Implementing PSMs

- Transform scene into post-projective view space
- Construct a projection transform for the light
  - From light's position in post-projective space
- Compose light projection matrix with scene
- Simple in concept, hard in practice
  - Post-projective space is filled with singularities
  - Spends too many shadow texels on nearby objects

©2004 NVIDIA Corporation. All rights reserved.

Implementing LSPSMs

- Construct “Light Space” transformation based on view & light directions
- Compute a 3DH projection point based on frustum Z_{near} and Z_{far}
- Compose perspective transform with light space
- Perspective xform bounds based on CSG ops
  - Easy for directional lights, clipper required for others

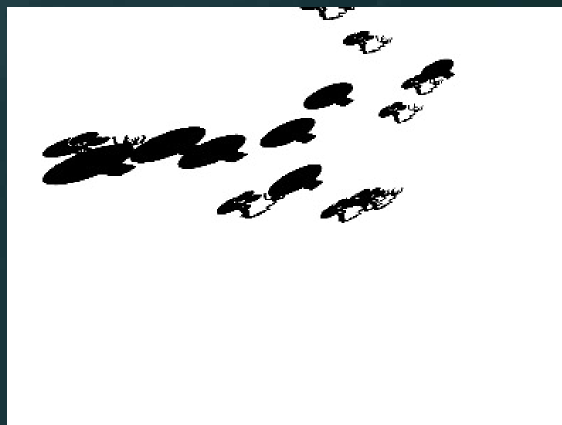
©2004 NVIDIA Corporation. All rights reserved.

Implementing TSMs

- Project view frustum extremities by `lightProj`
  - `lightProj` is conventional shadow mapping matrix
  - Clip view frustum to light frustum for local lights
- Find 2D convex hull of projected frustum
- Construct a trapezoid that tightly bounds hull
- Use 2DH transform to convert trapezoid to square

©2004 NVIDIA Corporation. All rights reserved.

Shadow Map Comparison



TSM

LSPSM

Uniform

PSM

©2004 NVIDIA Corporation. All rights reserved.

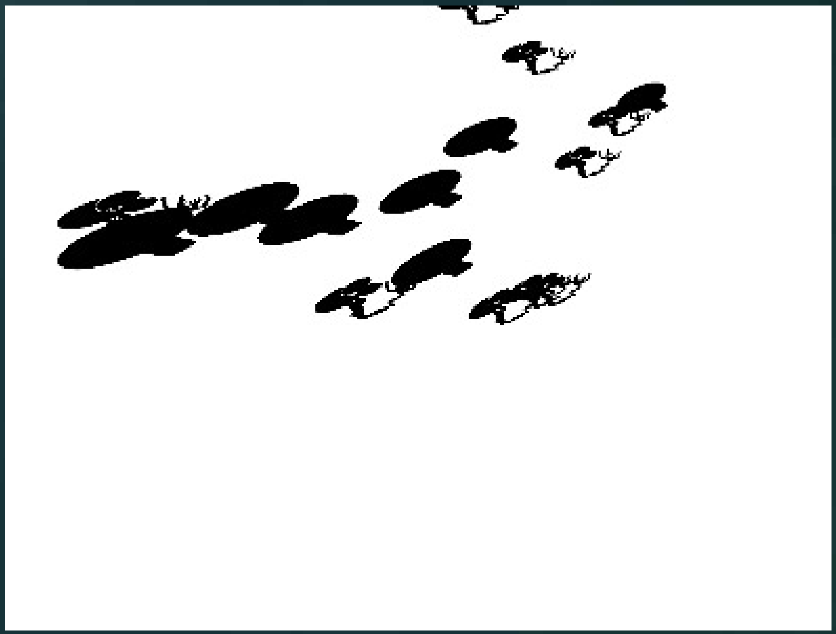
Implementation Issues

- No frame-to-frame caching
- Near/Far Shadow Map Quality
  - All variants have ways to balance this
- Depth Biasing
  - PSM variants much worse than uniform shadow maps
  - Slope/Scale frequently required, may not be enough
  - TSM is extreme case
    - So bad that authors recommend using depth replace to output non-warped Z instead

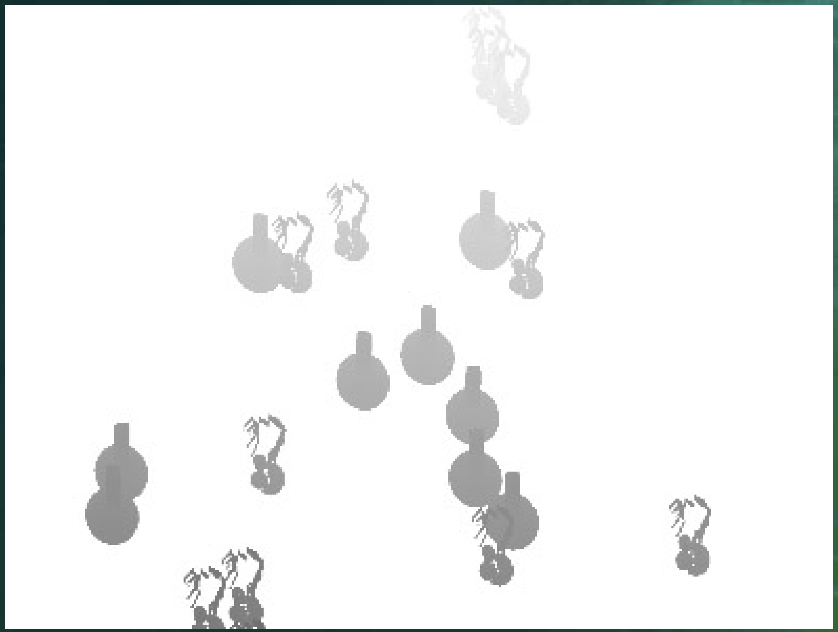
©2004 NVIDIA Corporation. All rights reserved.

TSM Z vs Uniform Z

TSM



Uniform



8 bit depth values are all black for TSMs!

©2004 NVIDIA Corporation. All rights reserved.

Optimization Parameters

- Each PSM variant has a way to trade-off near/far shadow quality
  - PSMs: $Z_{infinity}$
  - LSPSMs: N_{opt}
  - TSMs: 80% line
- All of these control the perspective distortion
- To get best quality, these terms must be adjusted dynamically (demo doesn't do this)

©2004 NVIDIA Corporation. All rights reserved.

Infinite Light Demo

©2004 NVIDIA Corporation. All rights reserved.

Thoughts on PSM Variants

- Only practical for infinite lights
  - Local light implementations require clippers, 3D convex hulls, and many other expensive CPU ops
  - And none can handle duelling frusta
    - Not an issue for infinite lights
- LSPSMs and TSMs are much easier to implement than PSMs
  - Light and Trapezoid Space avoid major singularities
- TSM variants provide the best overall quality
  - Especially with unit-cube clipping

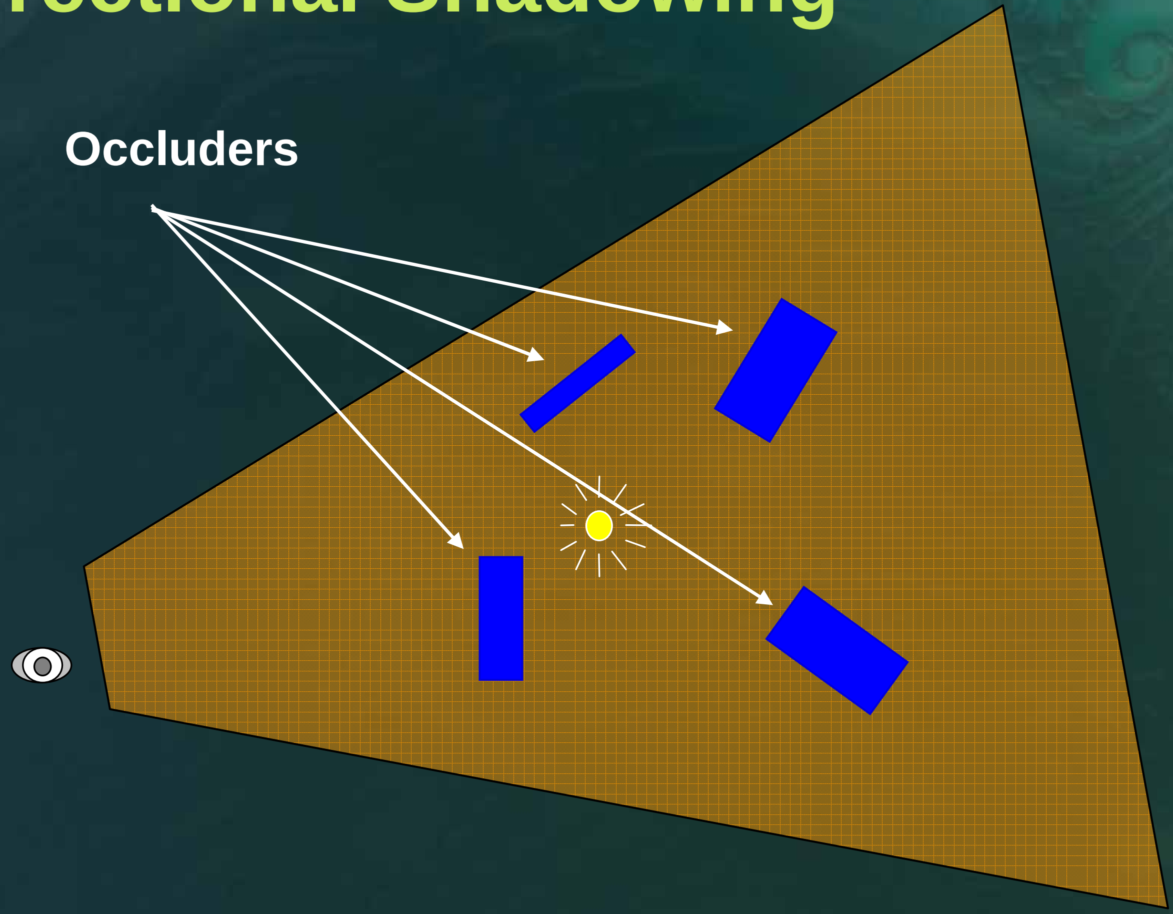
©2004 NVIDIA Corporation. All rights reserved.

Cascaded PSM Variants

- For extreme outdoor scenes, a single PSM may be insufficient
- However, the texel redistribution is still beneficial
- Consider using cascaded PSMs/TSMs
  - Partition view frustum into chunks
  - Compute unique PSM/TSM transform for each chunk

©2004 NVIDIA Corporation. All rights reserved.

Omnidirectional Shadowing



No one linear transform can handle all of the occluders

©2004 NVIDIA Corporation. All rights reserved.

Solutions

- Nonlinear projections
  - Dual-parabaloid
  - Hyperbolic
  - Spherical
- Multiple linear projections
  - Cube Maps

©2004 NVIDIA Corporation. All rights reserved.

Nonlinear Projections

- Hardware interpolation of $^x/_w$ $^y/_w$ is wrong
- Requires high scene tesselation
  - Reducing number of interpolated fragments
- Doesn’t work well with deferred renderers
  - Since exact projection per-pixel during lighting significantly differs from inexact projection per-vertex
- GPGPU “scatter” operations would help, here…

©2004 NVIDIA Corporation. All rights reserved.

Cube Maps

- 6 simple linear projections
  - No interpolation problems
- Single-pass lighting
- Don’t play well with depth textures, so you lose:
  - Free bilinear PCF
  - 32ppc rendering
  - Extra texture memory to support both 2D and omnidirectional shadows

©2004 NVIDIA Corporation. All rights reserved.

Why No DSTs?

- DSTs support is lost since cubemap distance is typically radial, rather than planar.

- However, eye-space Z of a 90 degree projection is just an unsigned permutation of [s t r]
  - If Z_{near} and Z_{far} are equal for all faces, window-Z is:

$$Z_s = \frac{-1}{MA} \times \frac{Z_{far} Z_{near}}{Z_{far} - Z_{near}} + \frac{Z_{far}}{Z_{far} - Z_{near}}$$

  - Where `MA = max( |s|, |t|, |r| )`

©2004 NVIDIA Corporation. All rights reserved.

Virtual Shadow Depth Cube Textures

- This allows us to use planar depth, which can be generated by rendering to a DST

- And if the cube faces are tiled in a DST…
  - Then we can render omnidirectional shadows in 1 pass, just like with cube maps!

- Need a way to efficiently map from (s,t,r) to (x,y)
  - Use a smaller, precomputed G16R16 cubemap to perform this indexing

©2004 NVIDIA Corporation. All rights reserved.

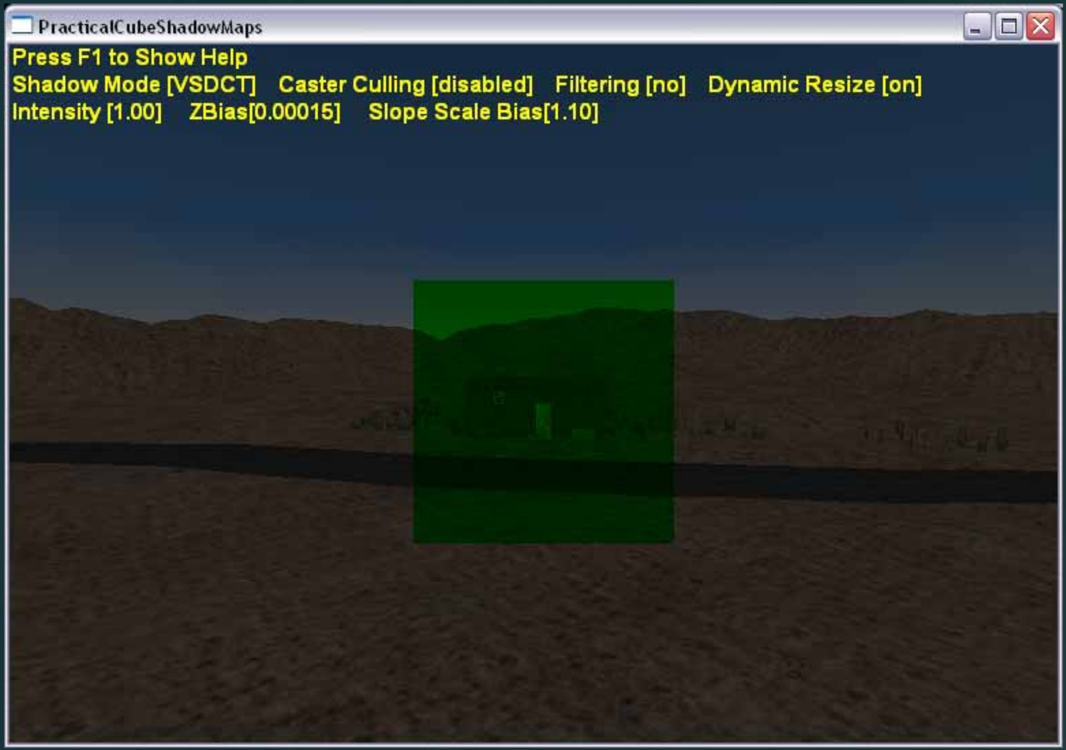
VSDCT Demo

VSDCT filtering > 2x faster than cube filtering

©2004 NVIDIA Corporation. All rights reserved.

Some General Performance Thoughts

- Use scissor to avoid shading pixels outside of light range; good when light is in view



©2004 NVIDIA Corporation. All rights reserved.

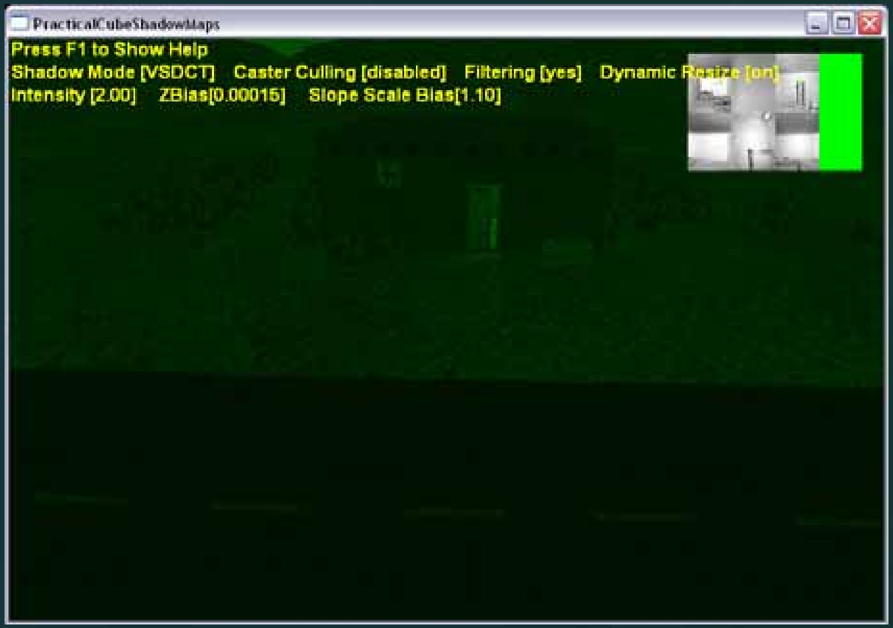
General Performance Thoughts

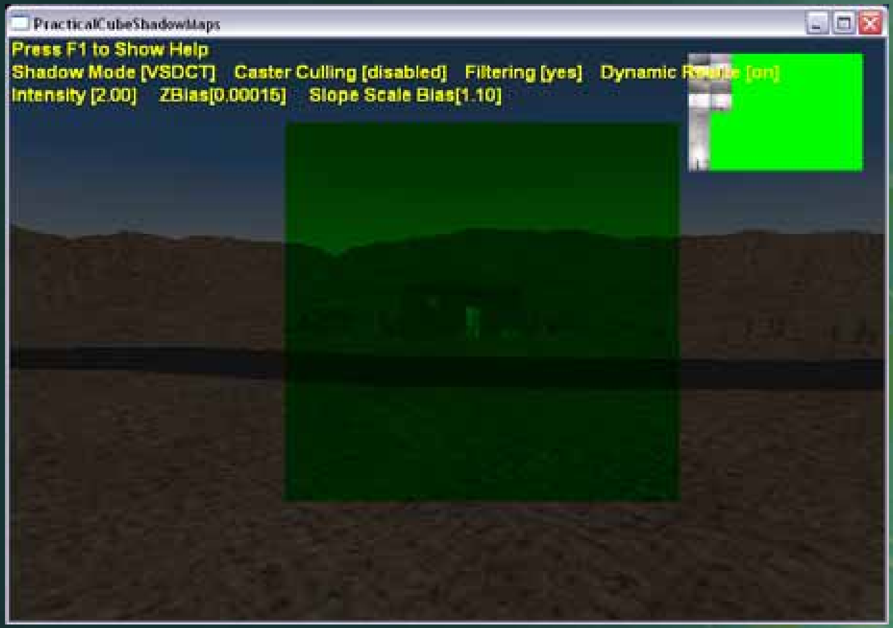
- Cull aggressively: if an omni light is outside the view frustum, at least one face can be culled

©2004 NVIDIA Corporation. All rights reserved.

General Performance Thoughts

- When lights are small on-screen, they don't need high-res shadow maps.
  - Easily accomplished with VSDCTs, or mipmaps on cube maps





©2004 NVIDIA Corporation. All rights reserved.

Summary

- Real-time shadow mapping isn't a solved problem
  - Lots of partial solutions for specific problems
- No one algorithm will work everywhere
  - And sometimes it is beneficial to combine them
- Always squeeze out optimizations
  - Shadows are expensive, don't spend perf when it doesn't help

©2004 NVIDIA Corporation. All rights reserved.

Acknowledgements

- Sim, Ashu, Cass, Cem and other NVIDIANs whose work I've shamelessly stolen
- Will Newhall, for the key VSDCT insight
- Oles Shishkovstov, for convincing me to implement all of this
- Simon Kozlov, whose *GPU Gems* article made Perspective Shadow Maps clear to me
- And everyone else I'm forgetting, for everything I've taken for granted :)

©2004 NVIDIA Corporation. All rights reserved.

Questions?

- gking@nvidia.com

©2004 NVIDIA Corporation. All rights reserved.

The Source for
GPU Programming
developer.nvidia.com
• Latest News
• Developer Events Calendar
• Technical Documentation
• Conference Presentations
• GPU Programming Guide
• Powerful Tools, SDKs and more ...
Join our FREE registered developer program for early access to NVIDIA drivers, cutting edge tools, online support forums, and more.
NVIDIA
developer.nvidia.com
©2004 NVIDIA Corporation. NVIDIA, and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation. Nalu is ©2004 NVIDIA Corporation. All rights reserved.