



**NVIDIA**®

# Variance Shadow Maps

Andrew Lauritzen, RapidMind

# Overview of Shadow Mapping



- **Introduced by Williams in 1978**
- **Advantages compared to shadow volumes:**
  - **Cost less sensitive to geometric complexity**
  - **Can be queried at arbitrary locations**
  - **Often easier to implement**
- **Disadvantages:**
  - **Aliasing**

# Shadow Mapping Algorithm

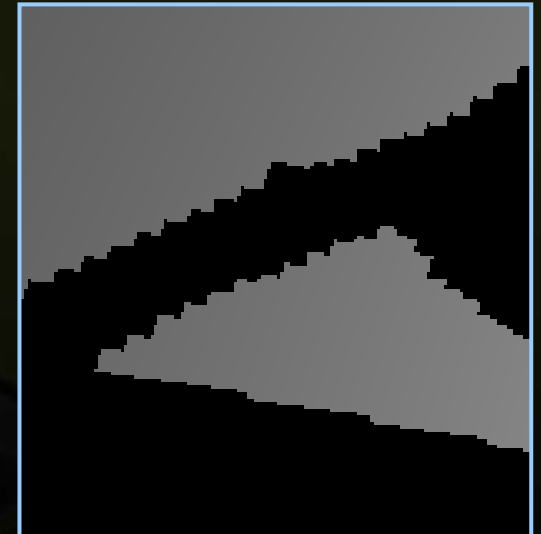
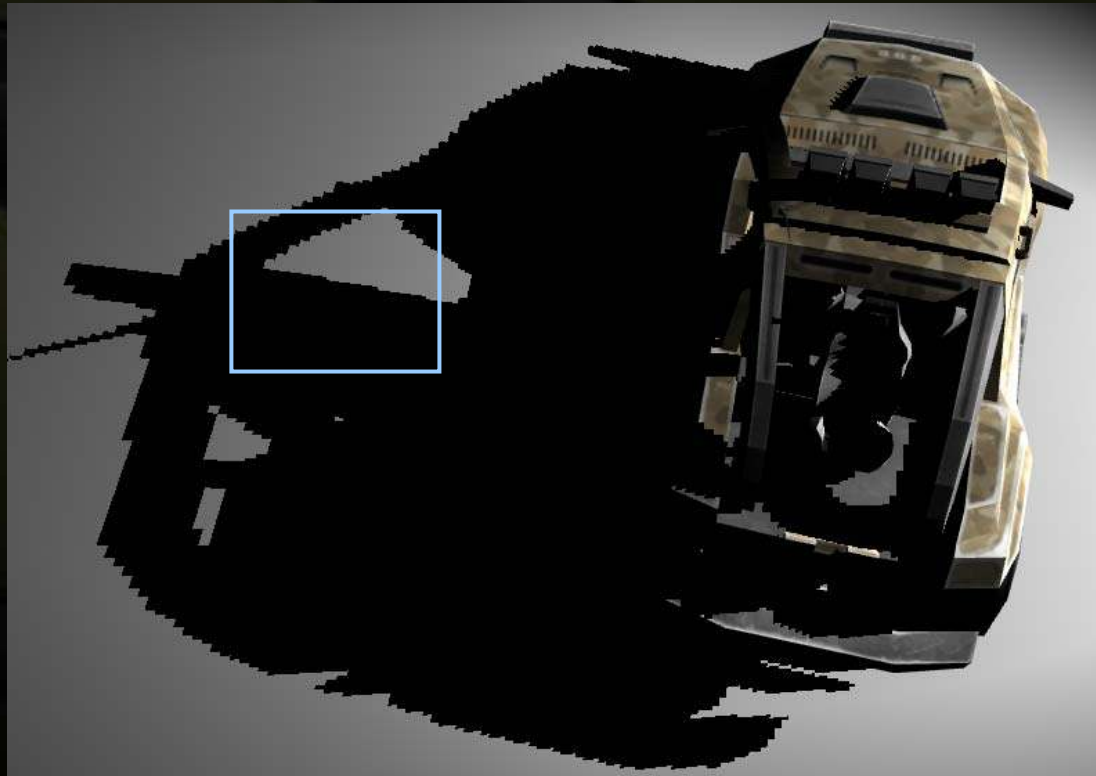


- **Render scene from light's point of view**
  - **Store depth of each pixel**
- **When shading a surface:**
  - **Transform surface point into light coordinates**
  - **Compare current surface depth to stored depth**
  - **If depth  $>$  stored depth, the pixel is in shadow; otherwise the pixel is lit**

# Aliasing Artifacts



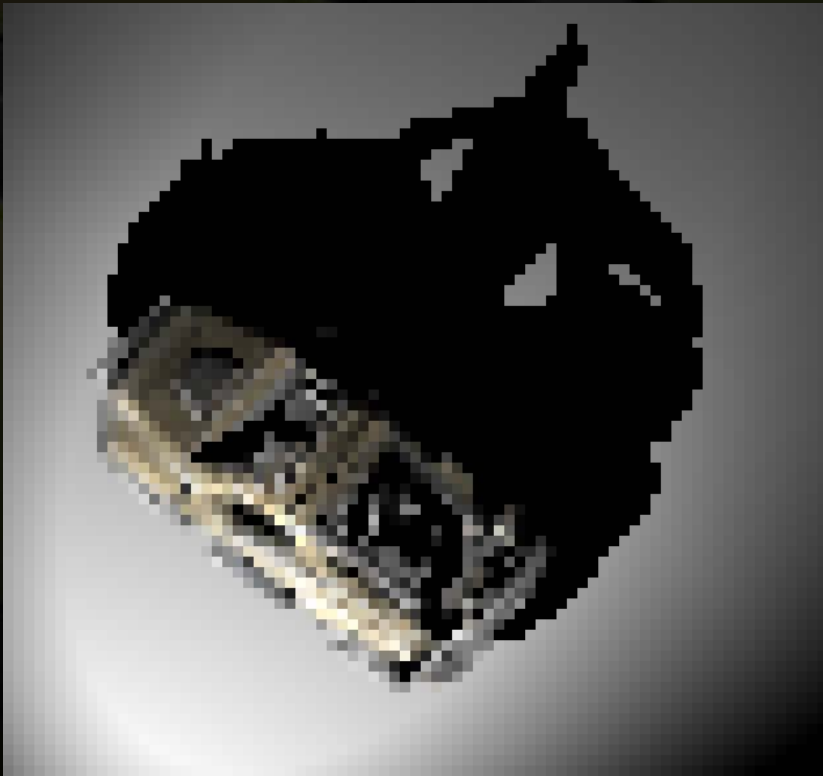
## ● Magnification artifacts



# Aliasing Artifacts



## ● Minification artifacts



- Typically encountered when viewed from a distance
- Produces ugly and distracting “swimming” effect along shadow edges

# Aliasing Artifacts



- **Anisotropic artifacts**
  - A mix of minification and magnification
  - Encountered at shallow angles



# Solutions?



- Also encountered with colour textures
- Reduce aliasing by hardware filtering
  - Magnification artifacts => linear interpolation
  - Minification artifacts => trilinear, mipmapping
  - Anisotropic artifacts => anisotropic filtering

# Solutions?



- **Can we apply these to shadow maps?**
  - Not at the moment
- **Interpolating depths is incorrect**
  - Gives  $\text{depth} < \text{average}(\text{occluder\_depth})$
  - Want  $\text{average}(\text{depth} < \text{occluder\_depth})$



# Percentage Closer Filtering

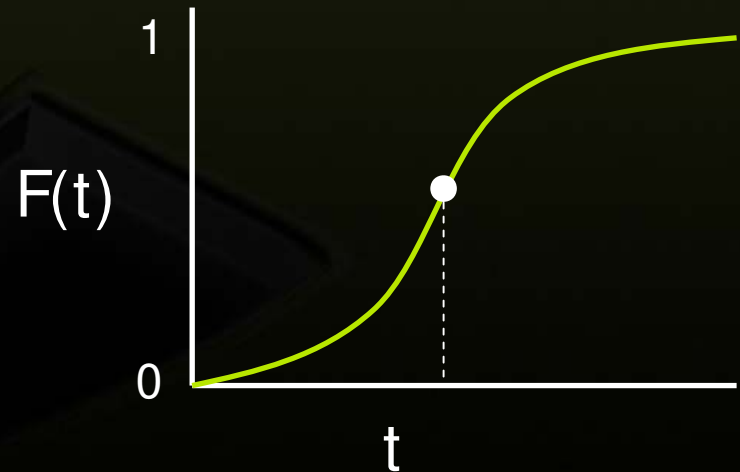
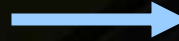
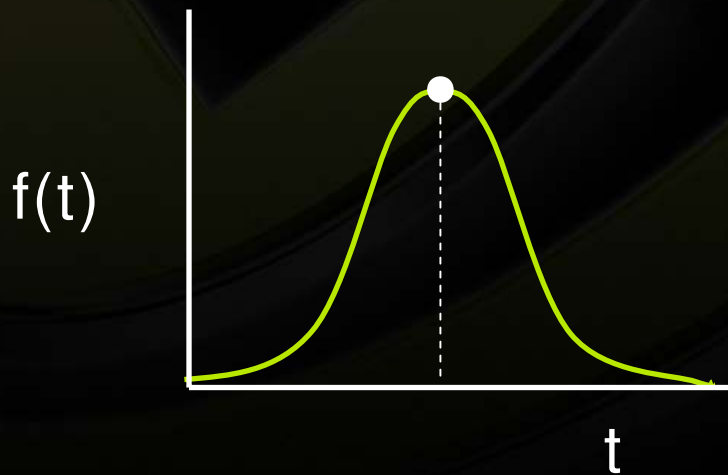


- Proposed by Reeves et al. in 1987
- Filter result of the depth comparison
  - Sample surrounding shadow map pixels
  - Do a depth comparison for each pixel
  - Percentage lit is the percentage of pixels that pass the depth comparison (i.e. are “closer” than the nearest occluder)
- NVIDIA hardware support for bilinear PCF
- Good results, but can be expensive!

# Occluder Distribution



- Really want a cumulative distribution function (CDF) of a set of depths
  - $F(t) = P(x \leq t)$
  - $F(t)$  is the probability that a fragment at distance “ $t$ ” from the light is in shadow



# Deep Shadow Maps



- **Lokovic and Veach, in 2000**
- **Per-pixel piecewise linear function**
- **No hardware filtering**
- **Complex reconstruction**

# Occluder Distribution



- **A representation that filters linearly?**
  - Allows us to utilize hardware filtering
- **Idea: Moments of distribution function!**
  - $E(x)$  is the mean,  $E(x^2)$ ,  $E(x^3)$ , etc.
  - Linear in distribution

# Variance Shadow Maps



- **Store depth squared as well as depth**
  - Gives  $E(x)$  and  $E(x^2)$  where  $x$  is the depth of the nearest occluder
  - Use the moments to approximate the fraction of the distribution that is more distant than the surface point being shaded

# Variance Shadow Maps



- We want to find  $P(x \geq t)$
- We have the mean, and can find variance:
  - $\mu = E(x)$
  - $\sigma^2 = E(x^2) - E(x)^2$
- Cannot compute CDF exactly
- Chebyshev's Inequality states:

$$P(x \geq t) \leq p_{max}(t) \equiv \frac{\sigma^2}{\sigma^2 + (t - \mu)^2}$$

# Variance Shadow Maps



- **Inequality only gives an upper bound**
  - **Becomes equality in the case of single planar occluder and receiver**
  - **In a small neighbourhood, an occluder and receiver will have constant depth and thus  $p_{\max}$  will provide a close approximation to  $p$**
- **So just use  $p_{\max}$  for rendering**

# Implementation



*// Call the parent light shader*

```
light_contrib & dir_to_light & dist_to_light & n_dot_l =  
    spot_light_shader(surf_position, surf_normal);
```

*// Transform the surface position into light space and project*

```
ShAttrib4f surf_light = light_view_projection | surface_position;  
ShTexCoord2f tex_coord = 0.5 * surf_light(0,1)/surf_light(3) + 0.5;
```

*// Query the shadow map*

```
ShAttrib2f moments = shadow_map(tex_coord);
```

*// Standard shadow map comparison*

```
ShAttrib1f lit_factor = (dist_to_light <= moments(0));
```

*// Variance shadow mapping*

```
ShAttrib1f E_x2 = moments(1);  
ShAttrib1f Ex_2 = moments(0) * moments(0);  
ShAttrib1f variance = E_x2 - Ex_2;  
ShAttrib1f m_d = moments(0) - dist_to_light;  
ShAttrib1f p_max = variance / (variance + m_d * m_d);
```

*// Attenuate the light contribution as necessary*

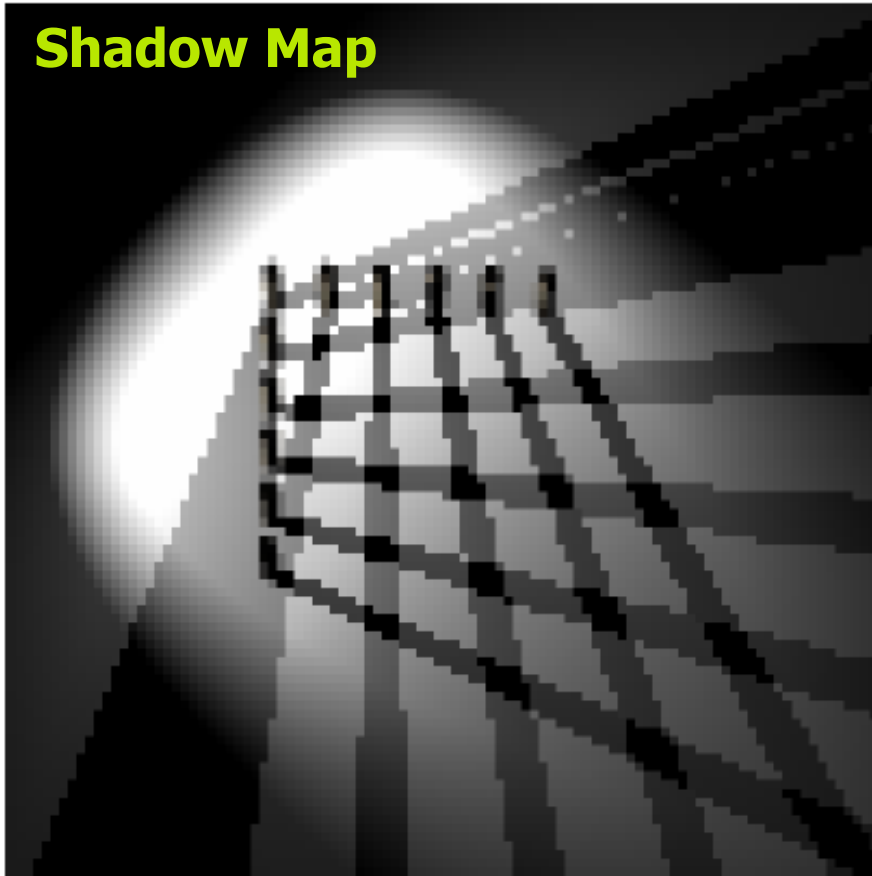
```
light_contrib *= max(lit_factor, p_max);
```



# Mipmapping Results



**Shadow Map**



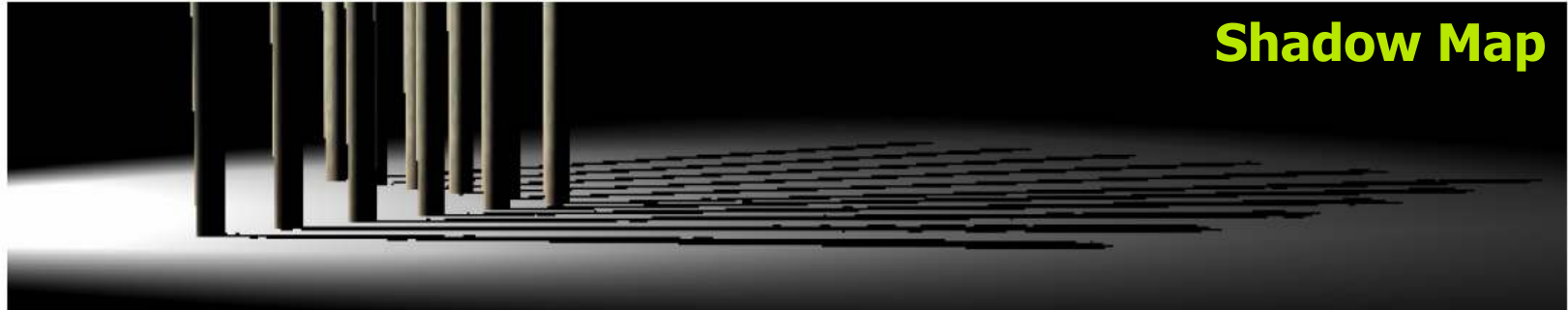
**Variance Shadow Map**



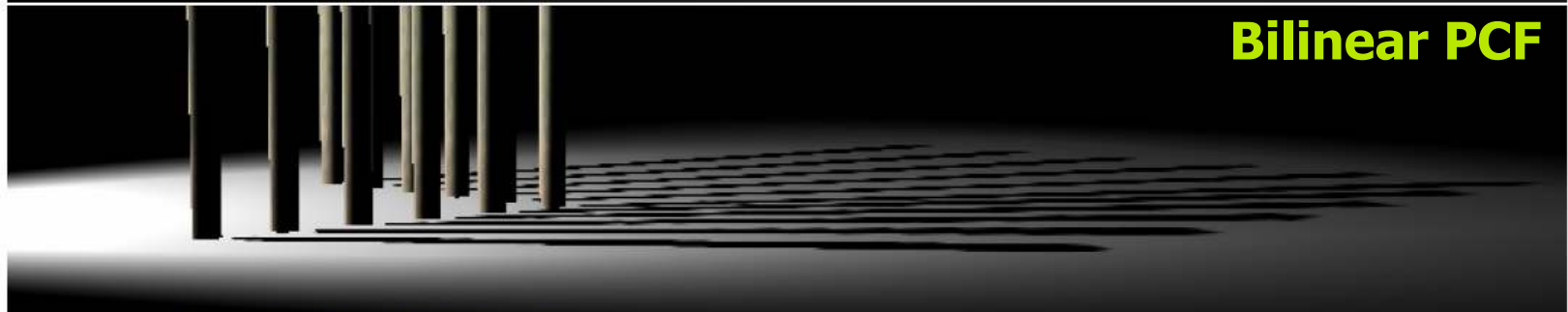
# Anisotropic Filtering Results



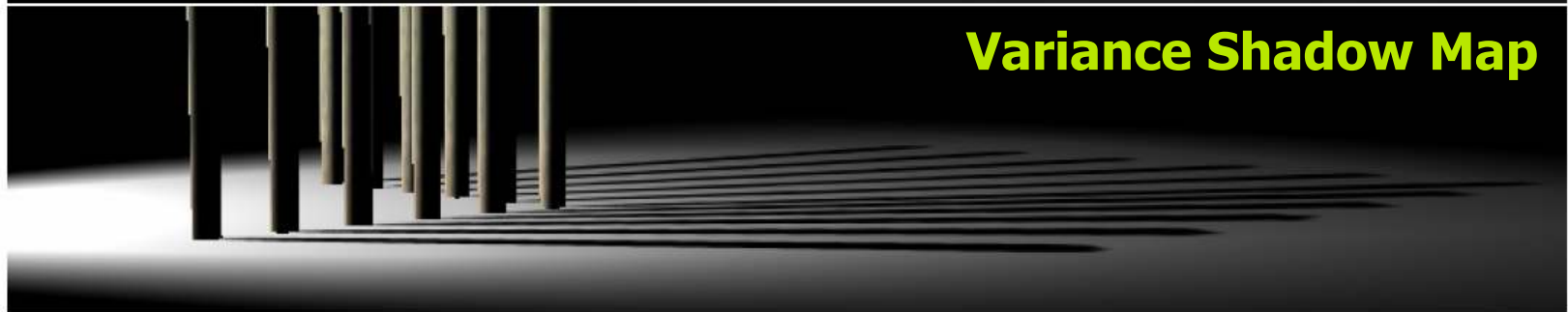
**Shadow Map**



**Bilinear PCF**



**Variance Shadow Map**

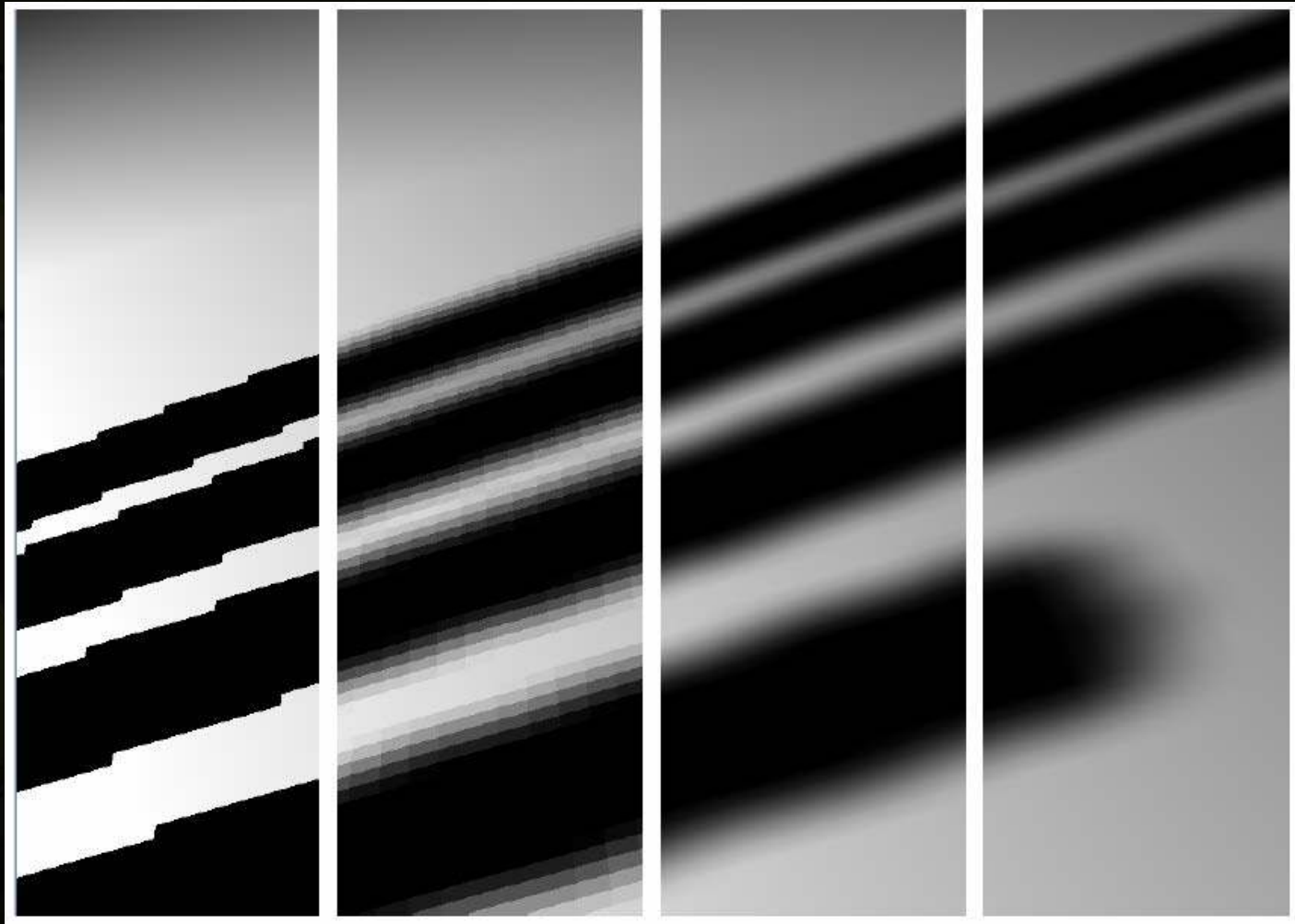


# Variance Shadow Maps



- **Can we do more?**
  - Our shadow maps can be arbitrarily filtered now
- **Pre-filter shadow map using a Gaussian blur**
  - Equivalent to percentage closer filtering
  - Separable convolution  $\Rightarrow O(n)$  on kernel size
  - Much faster than PCF complexity of  $O(n^2)$

# Gaussian Blur Results



**SM**

**PCF 5x5**

**Bil.PCF 5x5**

**VSM**

# Super-sampling



- **Generate more samples and filter**
  - Render large shadow map and down-sample
  - Or simply use texture LOD bias
- **Tiled rendering of a huge shadow map**
  - Render 4 tiles at 4096x4096 each
  - Down-sample to a single texture
  - Gives an anti-aliased 4096x4096 shadow map

# Multi-sampling



- **Simply enable multi-sampling while rendering the shadow map**
- **Support is dependent on chosen texture format**
  - **More of this later...**
- **Notes on gamma correction**
  - **Hardware might “gamma correct” the samples**
  - **This is incorrect for non-colour data!**
  - **Ideally we want to turn this “feature” off...**

# Other Fun Stuff



- **Orthogonal to projection-warping techniques**
  - **Perspective shadow maps (PSM)**
  - **Trapezoidal shadow maps (TSM)**

# Demo





# Texture Formats



- **Ideal texture format:**
  - **Renderable**
  - **Two components**
  - **High precision**
  - **Supports filtering (anisotropic, mipmapping)**
  - **Supports multisampling**

# Depth and Shadow Formats



- **+/- Indirectly renderable**
- **- Single-component**
- **- Often highly non-uniform precision**
- **- Do not support arbitrary linear filtering**
  - Mipmapping, trilinear, anisotropic, etc.
- **- Do not support multisampling**
  
- **Not the way to go...**

# Floating-point Formats



- **No renderable two-component formats!**
- **4x fp16**
  - + NVIDIA GeForce 6/7 supports filtering!
  - +/- Average precision
  - +/- Some hardware supports multisampling
- **4x fp32**
  - + Great precision
  - - No filtering on current hardware
  - - No multisampling on current hardware
- **Probably the best current options**

# Fixed-point Formats



- **8-bit formats?**
  - - Poor precision makes these unusable
- **2x 16-bit (i.e. G16R16)**
  - + Two component
  - + Often supports filtering
  - +/- Renderable on some hardware
  - +/- Acceptable precision (at least as good as fp16)
  - +/- Some hardware supports multisampling
- **Dreaming:**
  - 2x 32-bit filterable fixed point format?

# Texture Format Summary



- **Floating-point formats probably the best**
  - **Ideally we want filterable fp32**
- **16-bit fixed-point formats could work too**
  - **Dependent on what hardware supports**

# Numerical Stability



- Recall the computation of variance:

- $\sigma^2 = E(x^2) - E(x)^2$

- Highly numerically unstable!

- Recall Chebyshev's Inequality:

$$P(x \geq t) \leq p_{max}(t) \equiv \frac{\sigma^2}{\sigma^2 + (t - \mu)^2}$$

- Can be a problem when fragment is near occluder
- Need a high-precision texture format

# Ways to Improve Stability



- **Can use any distance metric that we want**
  - Post-projection “depth” ( $z$ ) is a bad choice
  - Use a linear metric, ex. distance to camera
- **When using floating-point formats**
  - Rescale the numeric range to fall in  $[-1, 1]$
  - Gets an extra bit of precision from the sign

# Ways to Improve Stability



- Four-component floating-point formats

- Store extra precision in extra components
- Must still filter linearly!!!

- Example encoding:

```
ShAttrib2f moments = (...);  
ShOutputAttrib4f output;  
const float factor = 64.0f; // Try to gain 6 more bits  
output(0,1) = frac(moments * factor) / factor;  
output(2,3) = moments - output(0,1);
```

- Example decoding:

```
ShAttrib4f input = shadow_map(tex_coord);  
ShAttrib2f moments = input(0,1) + input(2,3);
```



# Ways to Improve Stability



- Use a 32-bit per component floating-point texture!
- We've had no precision problems with fp32

# Notes on Shadow Bias

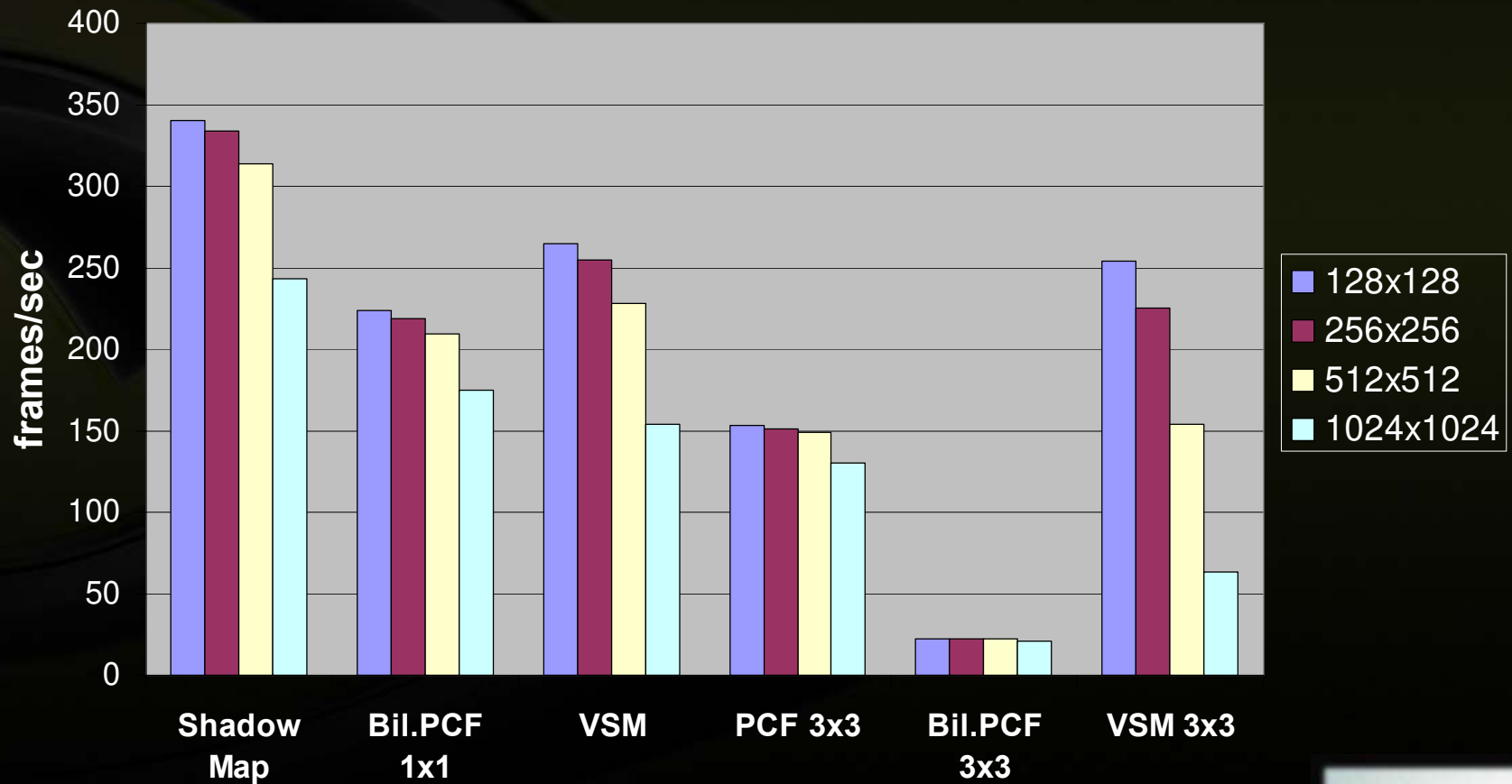


- **Biassing depth comparison usually required**
  - Proportional to slope of polygon (`glPolygonOffset`)
  - Scene dependent and error-prone
- **Not required for variance shadow maps!**
  - If  $(t - \mu) \sim 0$  then  $p_{\max} \sim 1$
- **May want to bias variance very slightly**
  - For numeric stability reasons
  - This is neither slope nor scene dependent!

# How Fast?



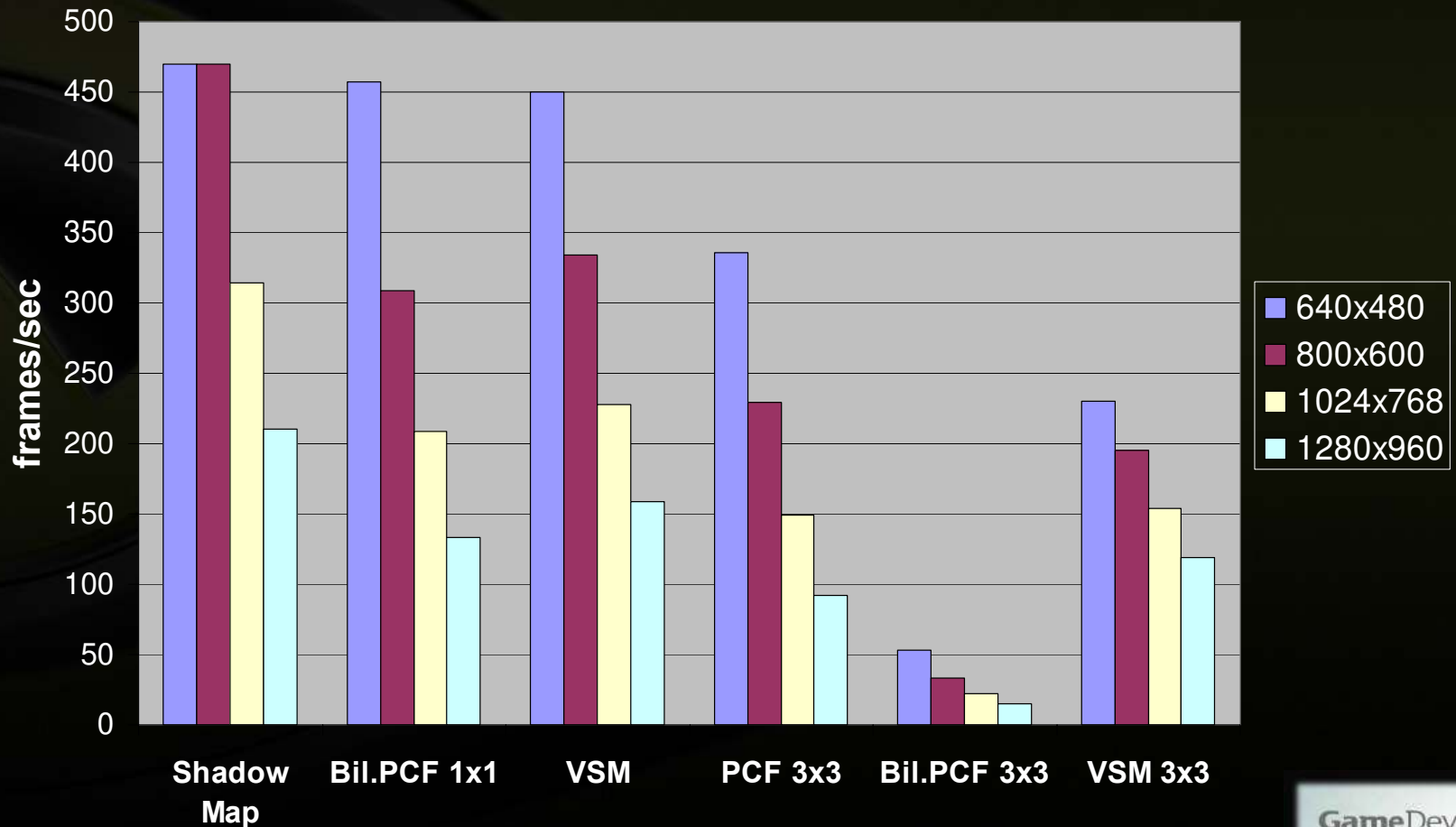
## GeForce 6800GT @ 1024x768



# How Fast?



## ● Fix shadow map at 512x512



# Light Bleeding



- $p_{\max}$  works in many situations, but not all
- When  $\sigma^2$  is large, can get “light bleeding” :



# Ways to Reduce Light Bleeding



- **Lower depth complexity in light space**
  - Ex. Use variance shadow maps for the sun, not headlights
  - Construct scenes with this artifact in mind
  - Control attenuation ranges carefully
- **Use ambient or multiple lights**
  - Contrast will be lessened
- **Use static lights**
  - Moving lights makes the projection obvious
- **Use smaller filter regions**
  - Artifact is only as large as the filter region

# Ultimate Solutions



- Find a higher-order inequality?
- Fast, programmable hardware filtering?
- Combine with percentage closer soft shadows
  - Randima Fernando (NVIDIA), 2005
  - Cheap, perceptually-correct soft shadows?
- Lots of potentially fruitful hybrid techniques!

# Conclusion



- **Introduced a simple solution to many forms of shadow map aliasing**
- **Implemented easily on modern hardware**
- **Compares favourably in both performance and quality to existing techniques**



# For More Information...



- Donnelly and Lauritzen, *Variance Shadow Maps*, ACM Symposium on Interactive 3D Graphics and Games 2006
- <http://www.punkuser.net/vsm/>
- Questions?

# The Source for GPU Programming

[developer.nvidia.com](http://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](http://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.