Advances in Real-Time Skin Rendering

Natalya Tatarchuk
ATI Research

Overview

- Subsurface scattering simulation
  - Texture Space Lighting
  - Irradiance Gradients
  - Precomputed Radiance Transfer
  - Additional tricks for simulating light scattering through skin

Why Skin is Hard

- Most lighting from skin comes from sub-surface scattering
- Skin color mainly from epidermis
- Pink/red color mainly from blood in dermis
- Lambertian model designed for “hard” surfaces with little sub-surface scattering so it doesn’t work real well for skin

Approximate Skin Cross Section

Air

Epidermis

Dermis

Bone, muscle, guts, etc.

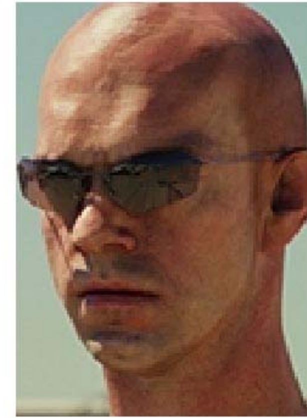
Subsurface Scattering

[Jensen01]

- Subsurface scattering is when light is reflected and refracted internally within a material.
  - Required to accurately portray semi-transparent or turbid materials.
  - Skin is comprised of multiple semi-transparent layers which contribute to the overall appearance.
  - The appearance of skin is dominated by subsurface scattering.
  - Humans are very well conditioned to seeing skin, so subtle nuances in appearance are important.

Texture Space Subsurface Scattering

- **From** Realistic Human Face Rendering for “The Matrix Reloaded” **@ SIGGRAPH 2003**



- **From Sushi Engine: Ruby1 Lighting**

Current skin in Real Time

Real-Time Texture Space Lighting

- Render diffuse lighting into an off-screen texture using texture coordinates as position
- Blur the off-screen diffuse lighting
- Read the texture back and add specular lighting in subsequent pass
- We only used bump map for the specular lighting pass

Results: Standard Lighting Model

Results: Blurred Lighting Model

New Approaches in Skin Rendering in Real-Time: Ruby2 and more

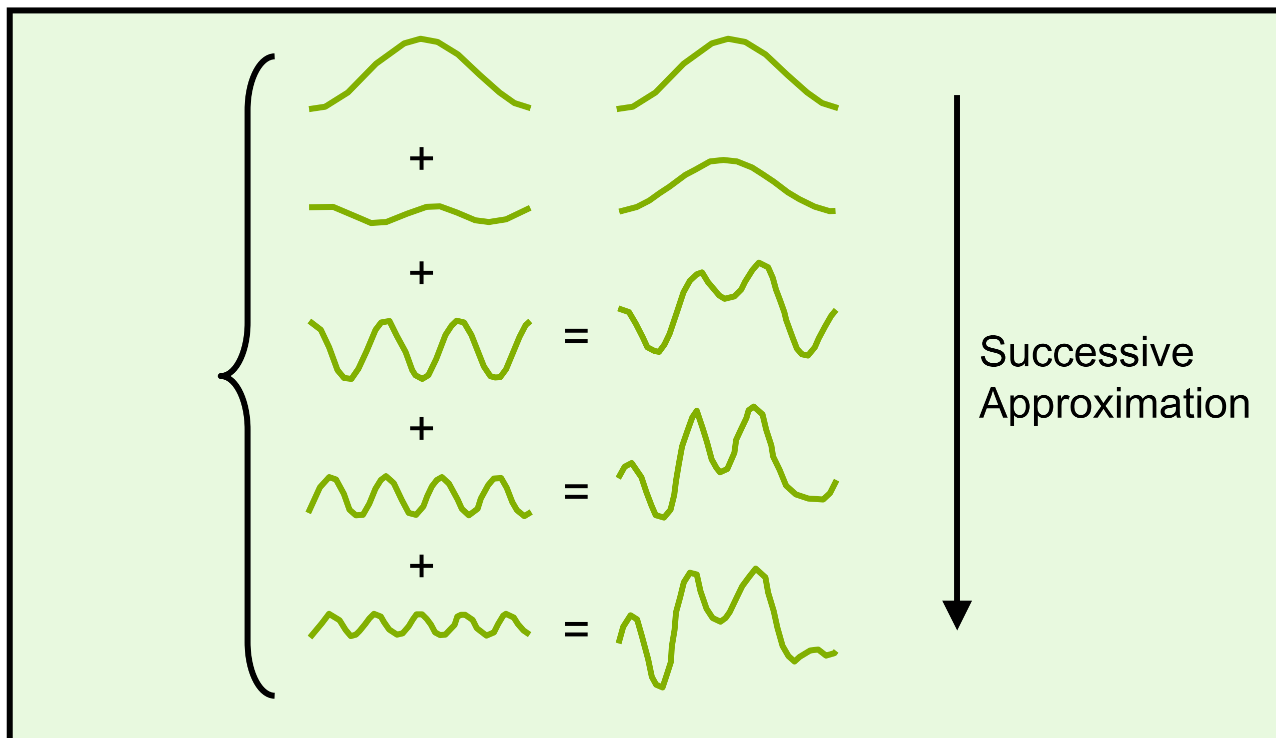
- Overview of PRT lighting
- Irradiance volumes
- Irradiance gradients
- Combining PRT lighting with standard rendering techniques in Ruby2
- Combining Ruby1 and Ruby2 style lighting

Ruby: Dangerous Curves (aka Ruby2)

- Precomputed Radiance Transfer with Spherical Harmonics
  - Allows for sub-surface scattering and global illumination effects
- Irradiance volumes
  - Allows for changing incident lighting as Ruby moves through the tunnel
- Irradiance gradients
  - Allow for variation in the incident radiance over Ruby's extent in the scene

Fourier Theory

- Recall that it is possible to represent any 1D signal as a sum of appropriately scaled and shifted sine waves



Spherical harmonics are the same idea on a sphere!

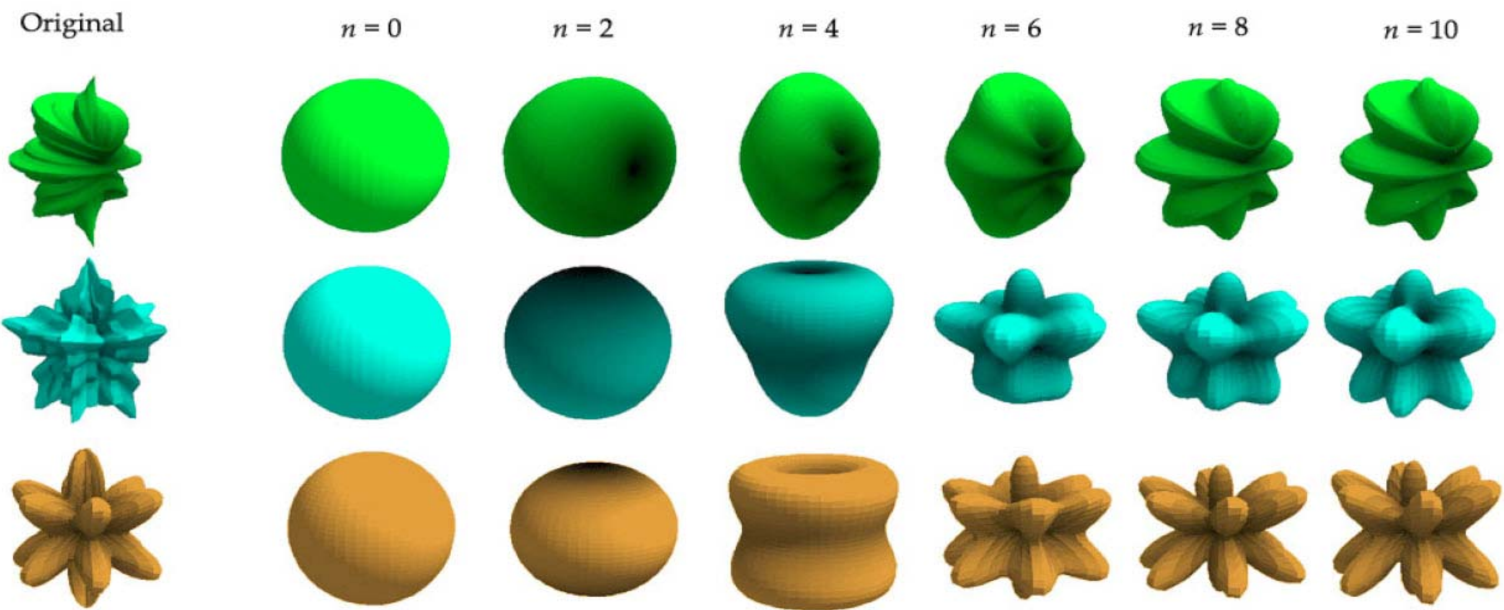
Spherical Harmonic Basis

Positive Values

Negative Values

From [Green03]

SH Example Approximations



From [Green03]

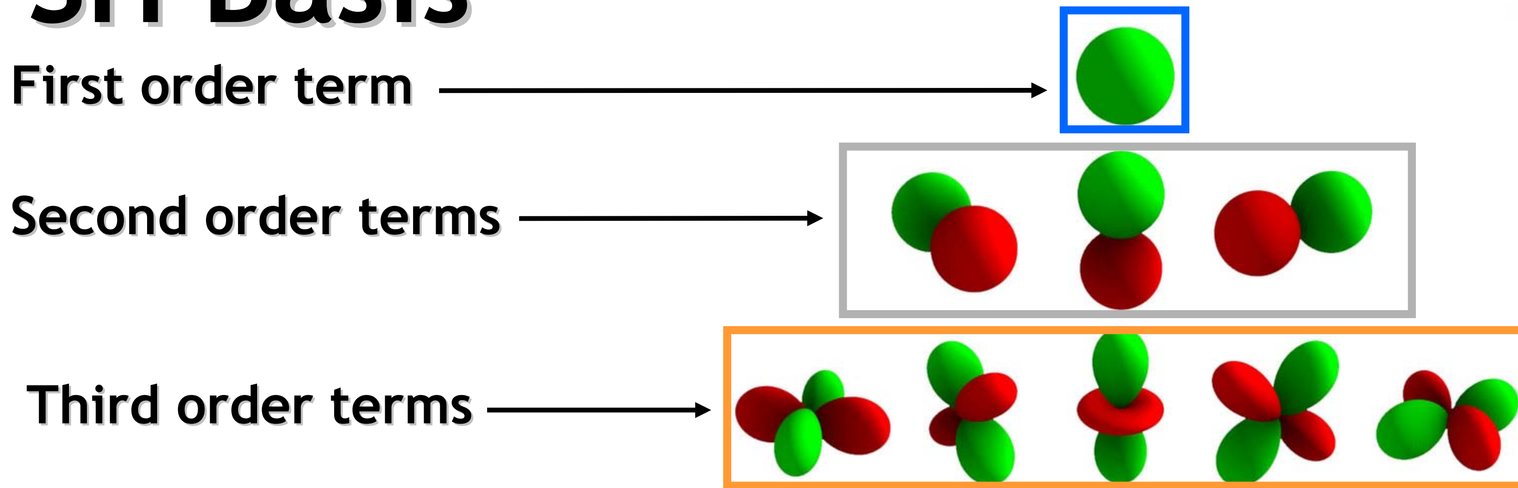
SH to approximate a lighting environment

Original Environment Map	Filtered Environment Map	SH Representation

Images from [Ramamoorthi01]

GameDevelopers Conference

SH Basis

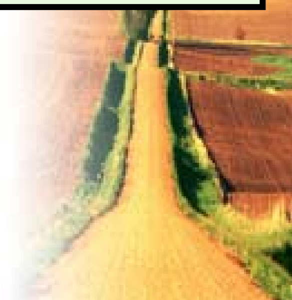


- Allows you to represent functions on the spherical domain.
- Series is infinite
  - Choose a range that fits our storage and approximation needs
  - (6th order for skin / 4th order for other stuff)
  - Each function in the truncated series is assigned to an element in a vector.
- Each element stores its associated SH function's contribution to the overall signal (basis weight)
  - Building your original (arbitrary) spherical signal out of a fixed set of scaled, predefined spherical signals
  - The larger the "fixed set" the closer the approximation will be

PRT Lighting Shader

```
//--------------------------------------------------------------------
// Computing the PRT lighting integral via a sum of dot products
//--------------------------------------------------------------------
for (int index = 0; index < (numSHCoeff/4); index++)
{
    o.cRadiance.r += dot(i.vSHTransCoef[index], g_vIrradianceSampleRedOS[index]  );
    o.cRadiance.g += dot(i.vSHTransCoef[index], g_vIrradianceSampleGreenOS[index]);
    o.cRadiance.b += dot(i.vSHTransCoef[index], g_vIrradianceSampleBlueOS[index] );
}
```

$< C_0\ C_1,\ C_2,\ C_3,\ C_4,\ C_5,\ C_6,\ C_7,\ C_8,\ C_9,\ \ldots\ C_n >$



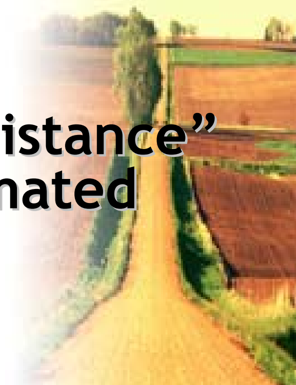
Extending PRT Techniques For Motion Through Complex Environments

Case study: Ruby2:

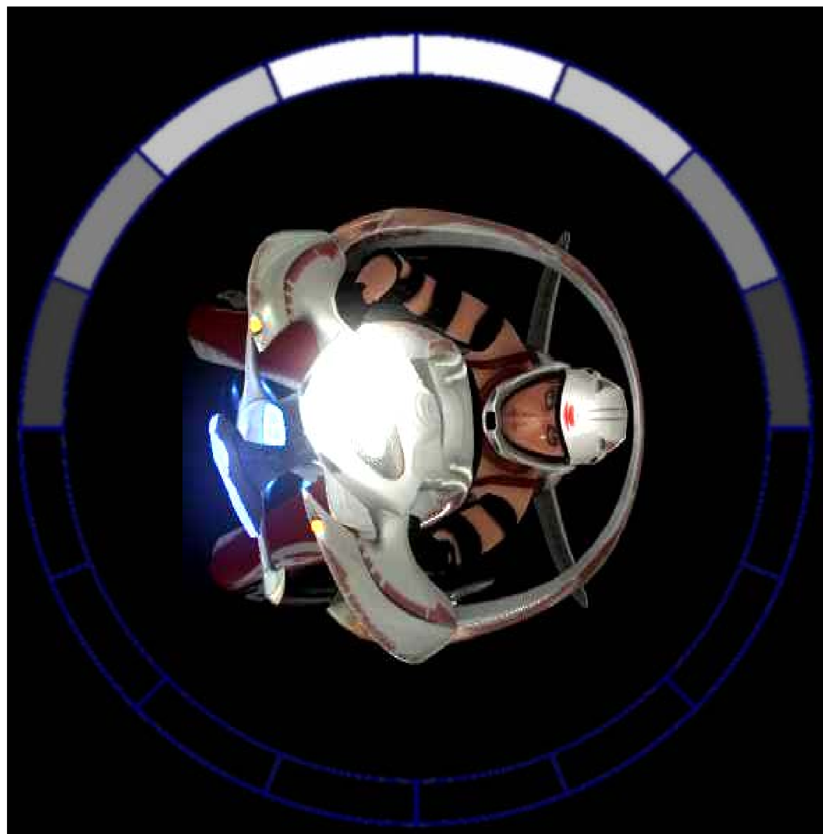
- Canonical pose lighting
- Irradiance Volumes
- Irradiance Gradients
- Integration with various material shaders

Computing PRT for a Canonical Pose

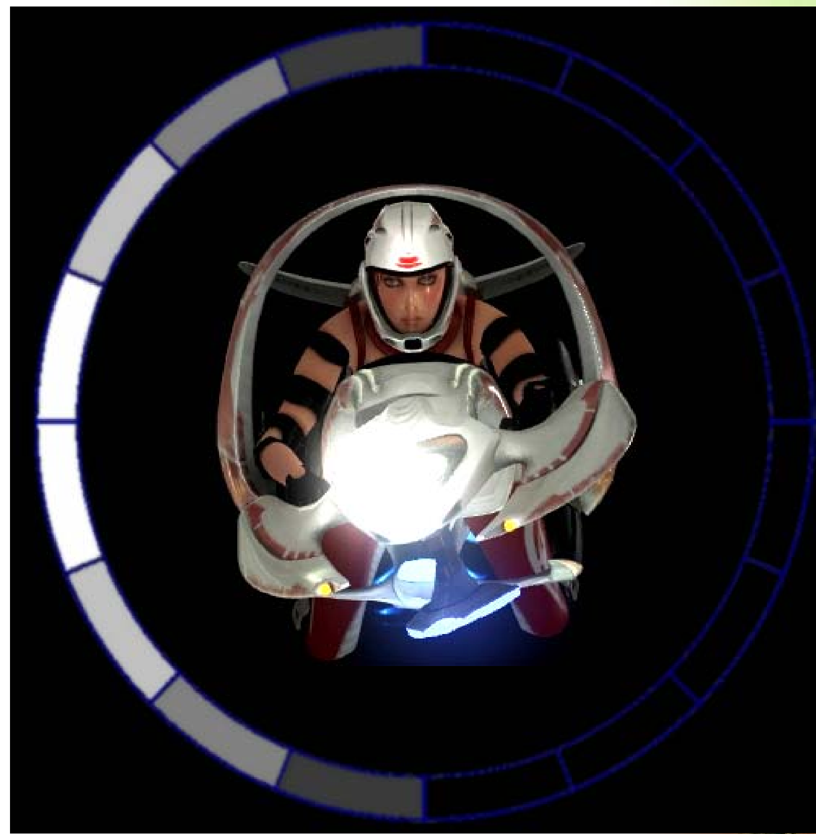
- Only if there isn't a great deal of articulated motion
  - Can pre-compute per-vertex PRT for the model for a single a canonical pose
  - In Ruby2 this worked well
- Choose the pose to minimize shadowing effects
  - They could change during the course of the demo!
  - Remember: PRT is mostly for ambient occlusion + sub-surface scattering effects
- Caveat: Assumes no large changes in occlusion and /or visibility.
- Latest DirectX PRT simulator can take a "max visible distance" value to minimize the error in static occlusion for animated meshes



Incident lighting rotation



Lighting from above in world space, but Ruby is sideways on her bike.



Perform lighting in object space by rotating the world space incident lighting into canonical object space.

The Problem: Spatially Varying Illumination Throughout the Scene.

- A limitation of PRT based lighting in its basic form is that the light sources are assumed to be at infinity.

  - Single lighting environment (irradiance sample) per scene.
- What can we do to get around this limitation?

Irradiance Volumes [Greger98]

- A grid of irradiance samples taken throughout the scene
- For a point in the scene, the irradiance can be computed by trilinear interpolation of the sampled irradiance within the scene.

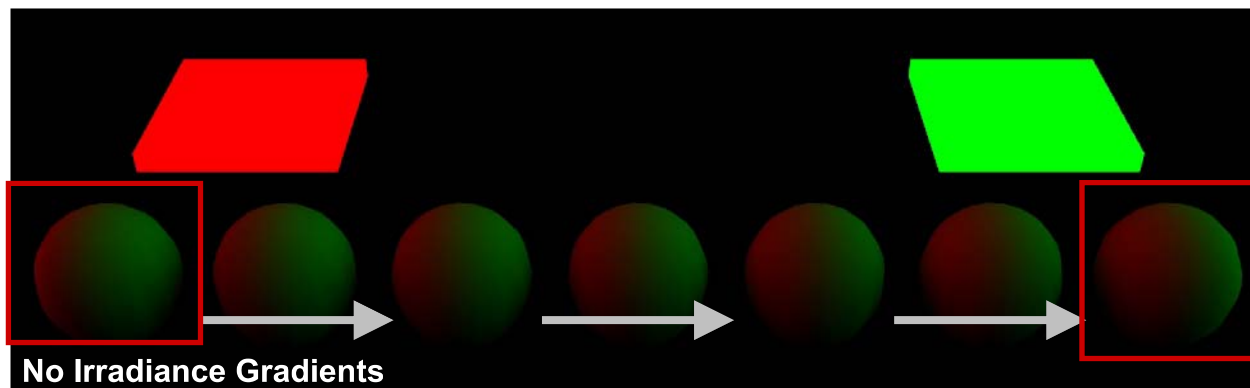
Generating Irradiance Volumes

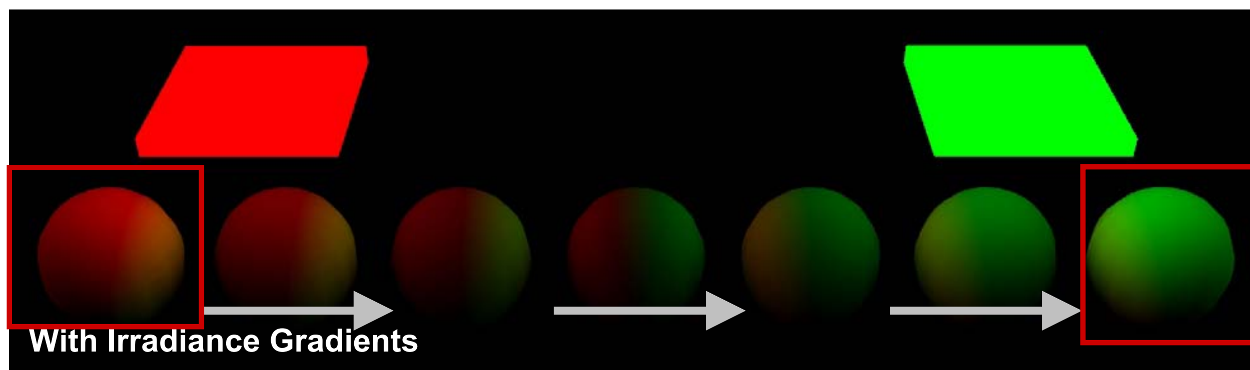
- Sample irradiance by rendering lit scene and light emitters into a cube map at each point
  - For SH-based PRT lighting, the SH coefficients are generated from the cube map
  - Best performed at preprocess time
- Spacing between samples depends on detail in scene, size of objects, legal positions for a character to move, etc.

Irradiance Samples Along a Path

- In Ruby2, since the motion is constrained to the inside of a tunnel, thousands of irradiance samples are taken along the path that Ruby's bike follows in the scene
  - Just an optimization - The math is the same as if we had distributed the samples in a more general way

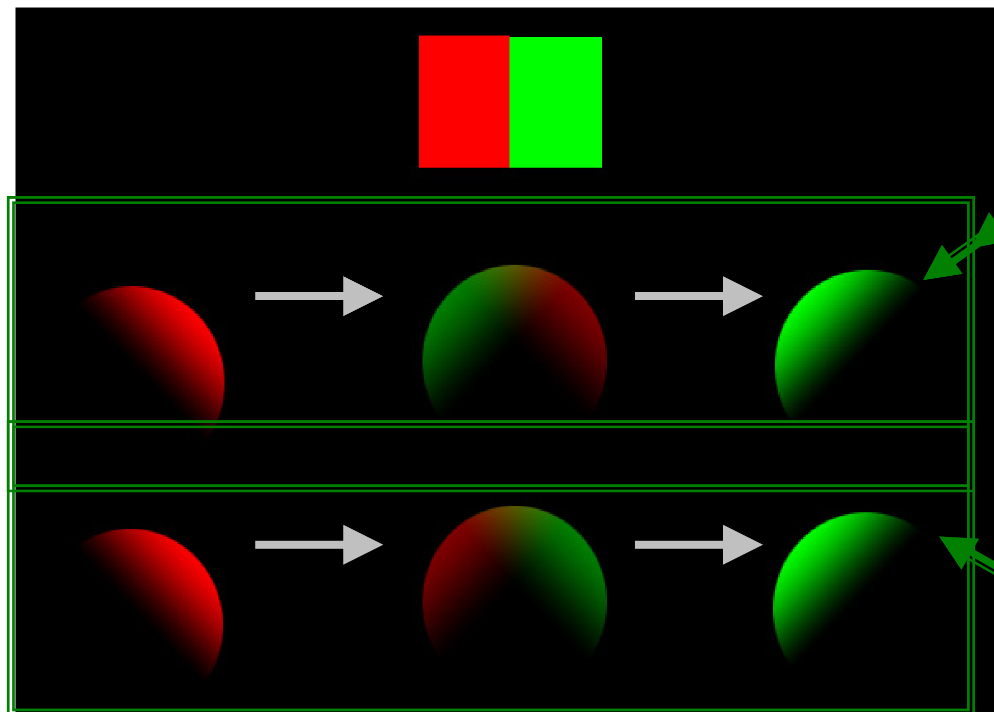
Irradiance Gradients





If irradiance varies greatly near a sample point, can store irradiance gradients along with each irradiance sample. [Ward 92][Annen04]

Irradiance Gradients: Interpolation

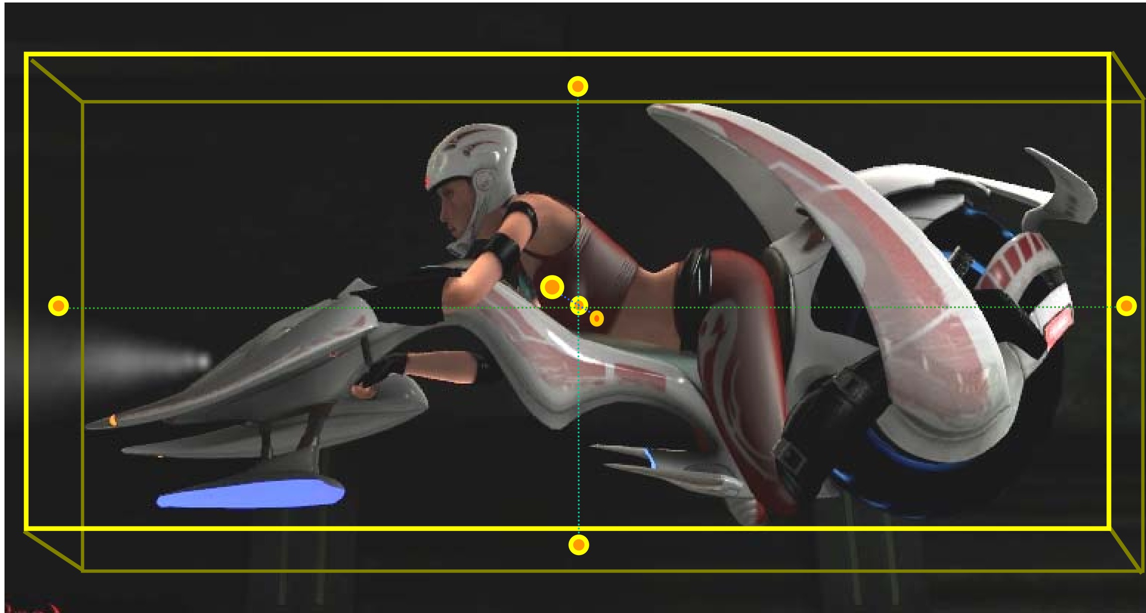


Linear interpolation between two irradiance samples

Gradient used for first-order Taylor expansion of irradiance coefficients

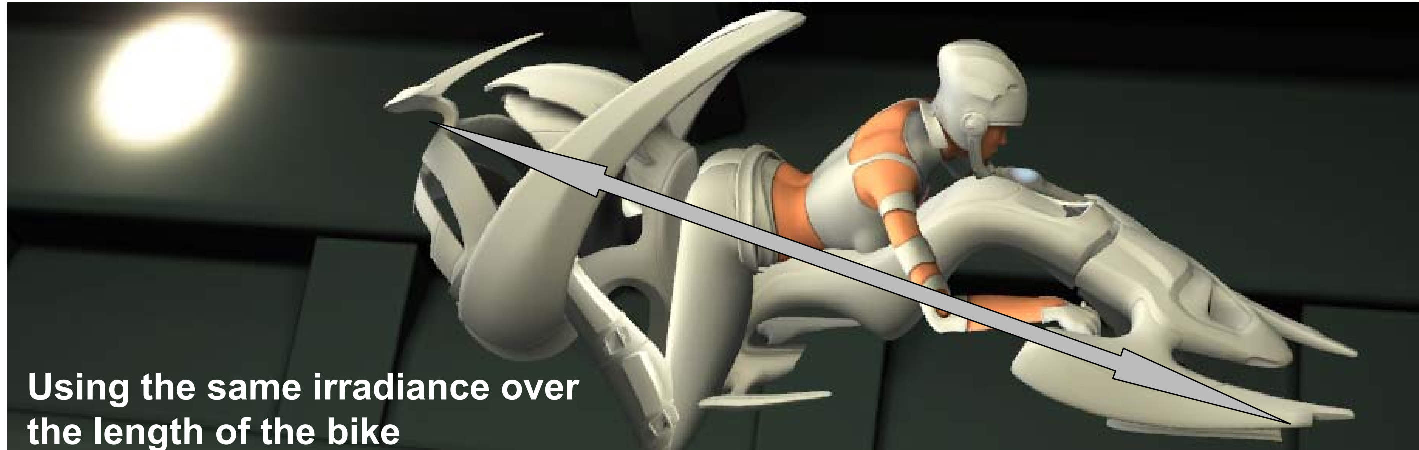
- Linear interpolation assumes irradiance changes piece-wise linear through space… **it doesn’t**

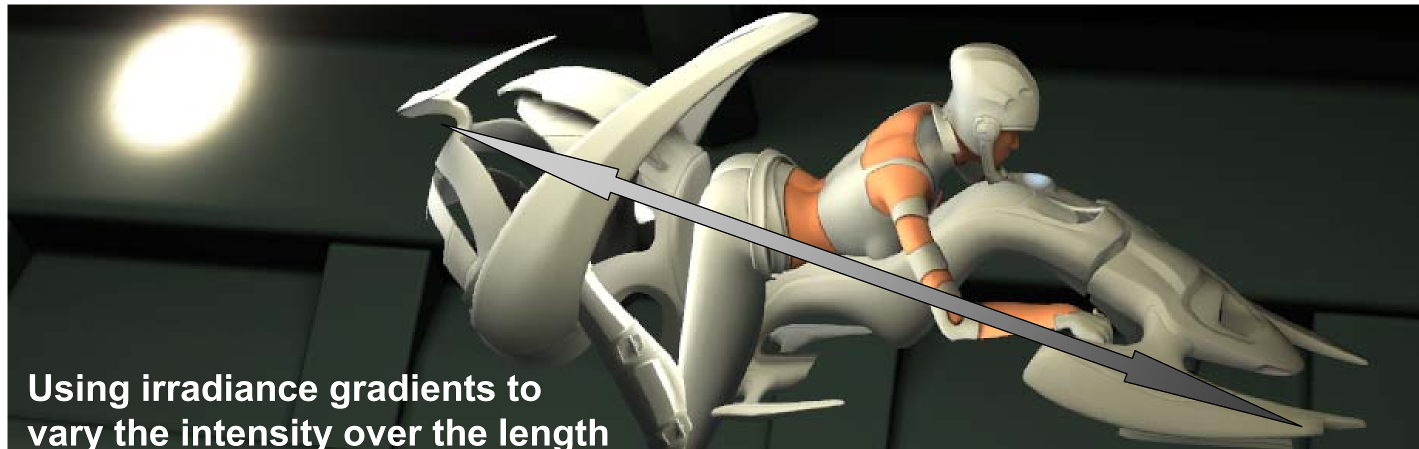
Sampling for Irradiance Gradients



- For Ruby2, at preprocess time, we compute spatial derivatives in *x*, *y*, and *z* using finite differences.
  - Samples are placed at the center of each face of Ruby's world-space bounding box
    - Irradiance is computed for 6 different offsets and derivatives are computed using these offsets.
    - At run time, irradiance and its gradients are rotated into object space for each object being rendered.

Irradiance Gradient Examples





of the bike

Implementation Details: The Simulator

- Computing PRT for Ruby during preprocessing took around 1 hour
  - Model has 80K polygons
- We shoot 2048 rays per-vertex with 3 ray bounce recursion depth

Memory Footprint

- **Skin:**
  - Used 6th order spherical harmonics with CPCA compression
  - **CPCA compression: 12 PCA weights per-vertex, 16 clusters**
  - 49 bytes per-vertex
    - 12 float values for weights plus ubyte for cluster index
- **Other materials:**
  - Color irradiance and irradiance gradients are stored in the vertex constant store
  - Only grayscale PRT coefficients are stored per vertex
  - Used 4th order PRT: 16 coefficients
  - Results in 12 float4 vectors for irradiance and 48 float3 vertex shader constants for irradiance gradients (w unused)

Constant store: 160 floats (2560bytes)

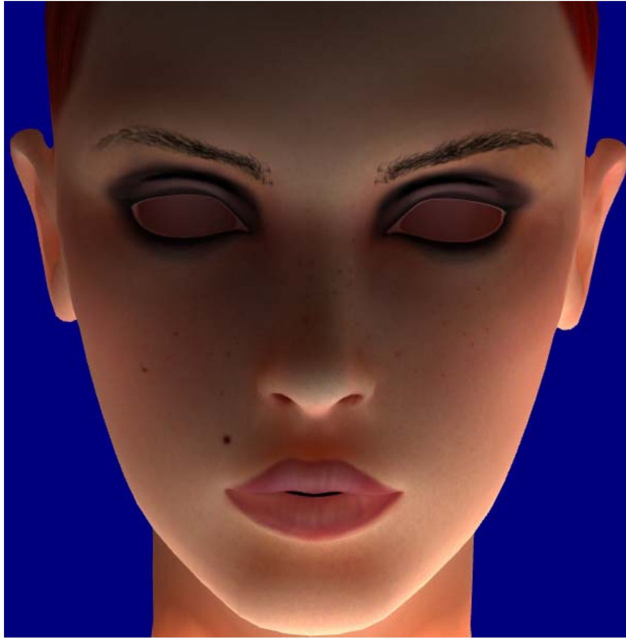
Implementation Details: Shader Details

- In the vertex shader:
  - The point's irradiance is computed using the positional offset, center position's irradiance and its gradients
  - Then the PRT lighting integral is computed using dot products

Two skin rendering approaches: Combining Ruby1 & Ruby2

- Texture space lighting (Ruby 1)
  - Controls lighting from a single light source
  - High frequency variations in the lighting
  - Light source is generally nearby
  - Independent of material, and lighting model, and animation technique
  - Shadow blur technique
- Enhanced PRT based lighting (Ruby 2)
  - Can be used to get effects such as light shining through the ears, and nostrils
  - Can model the effects of sub-surface scattering
  - Light sources assumed to be at infinity
  - Low frequency variations in lighting
  - Pre-process step
  - Animation unfriendly in basic form

How to incorporate PRT lighting?

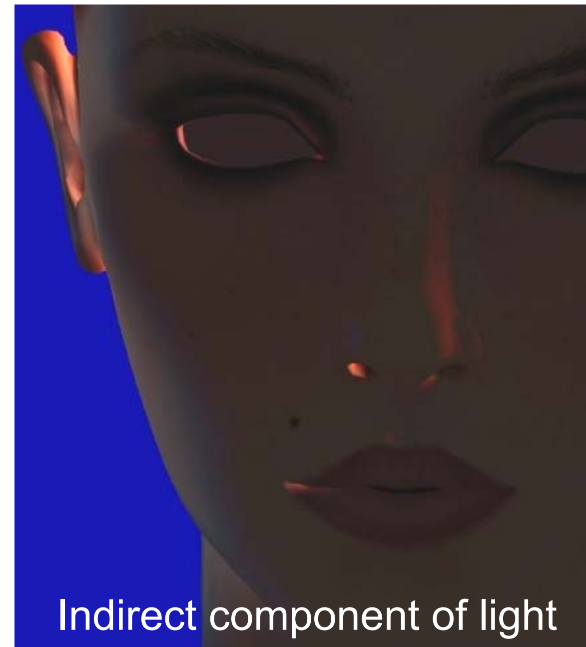


PRT only: Light from below

PRT only: Light from behind

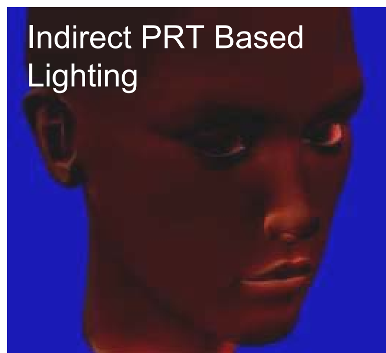
- We would like to apply the subsurface scattering effects using PRT to our shadow mapped lighting.
- *Key idea*: subtract direct illumination from PRT lighting, and add result to Ruby1 style shadow mapped lighting.

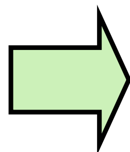
Indirect PRT lighting



- Break incident light into per-light SH coefficients.
  - E.g. Multi-light PRT shaders
- Subtraction of $f(\mathrm{N} \cdot \mathrm{L})$ term from PRT lighting per light.
  - Attenuates light shining directly onto surface
- Use per-light rim lighting term $g(-\mathrm{V} \cdot \mathrm{L})$ to accentuate light bleeding through thin surfaces (backlighting).
  - In rim-lighting configuration, use PRT lighting as is for indirect lighting.

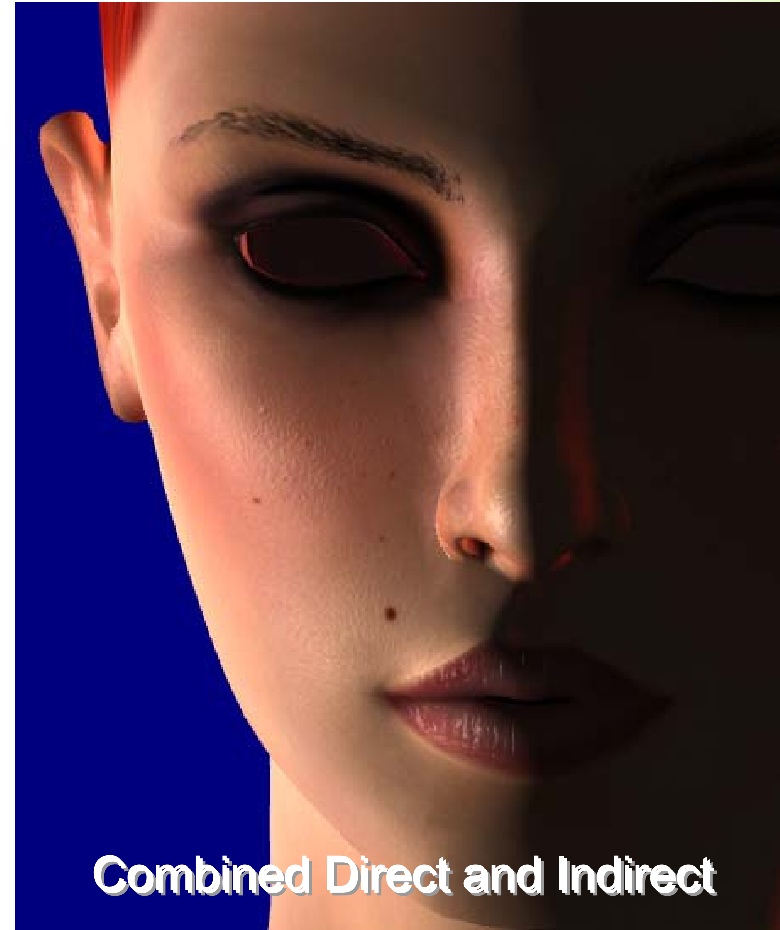
Direct and Indirect Illumination





- How can we combine the two:
  - Use shadow mapping with standard lighting to account for “direct” illumination
  - Use PRT based lighting to account for “indirect lighting” (subsurface scattered light)

Direct+Indirect Lighting Terms



More Interesting Ideas

- Blur visibility per-channel
- Use Spectral LUTs

Blurring visibility

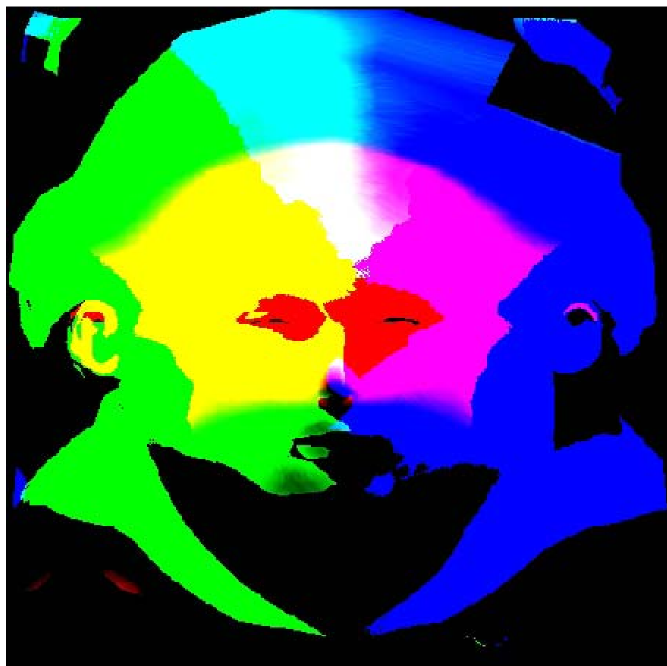
Visibility in light map space for 1 light

Blurring for sub-surface scattering effect

- **Use shadow mapping to determine shadowed regions in light space**
- **Lightmap space blurring of visibility rather than lighting.**
  - **Each light uses one channel of a visibility map**

SAN FRANCISCO CA MAR 7-11 GDC ›05

Blurring visibility



Visibility in light map space for 3 lights

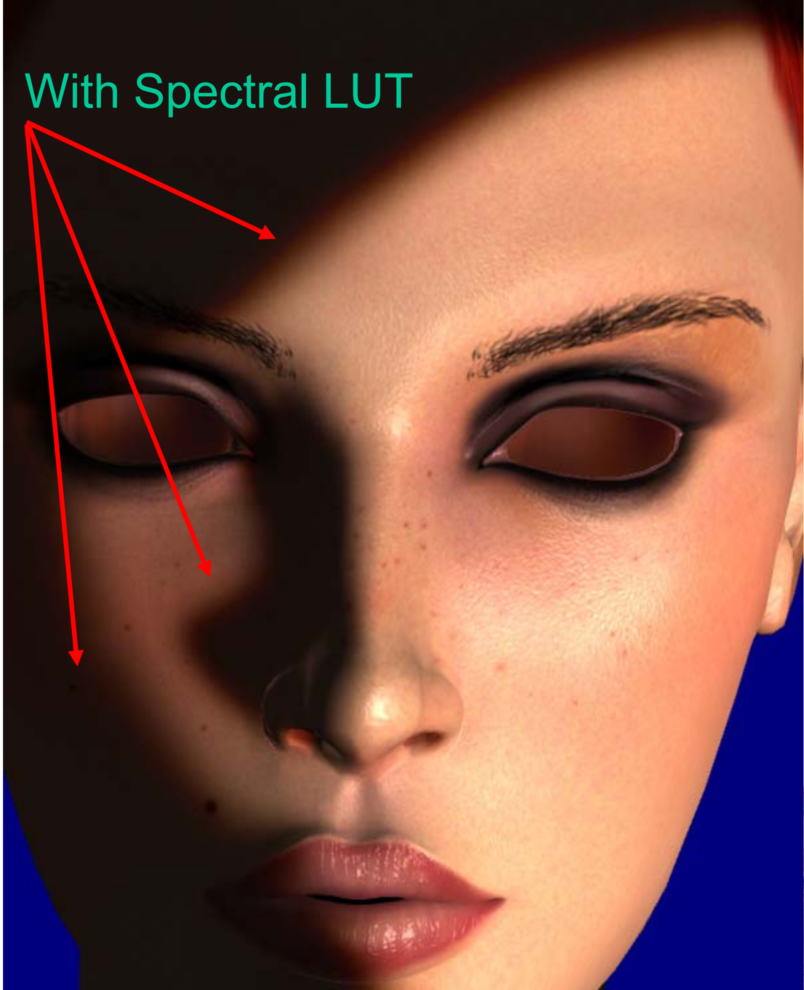
Blurring for sub-surface scattering effect

- Seen as each light uses one channel of a visibility map:
- We can still blur shadows from four lights at a time if using an .rgba texture.

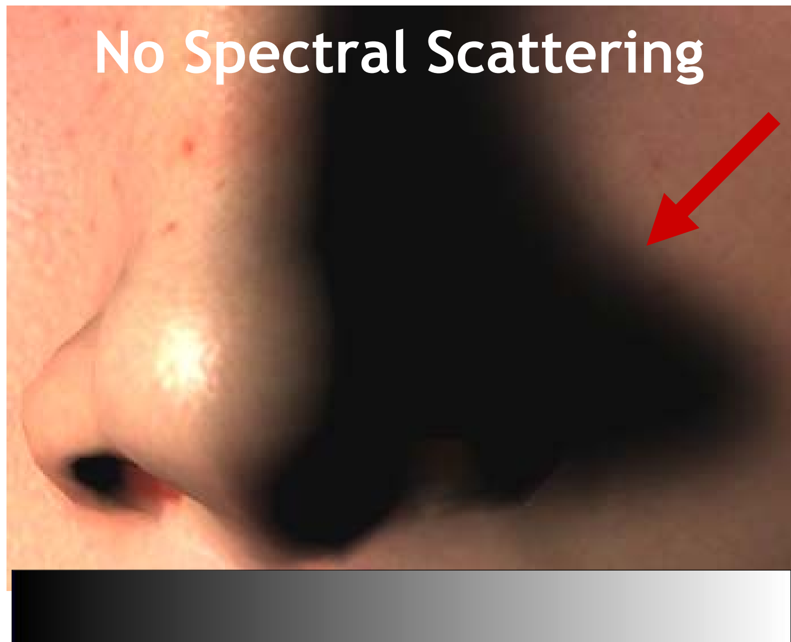
Why Do We Need Spectral LUT?

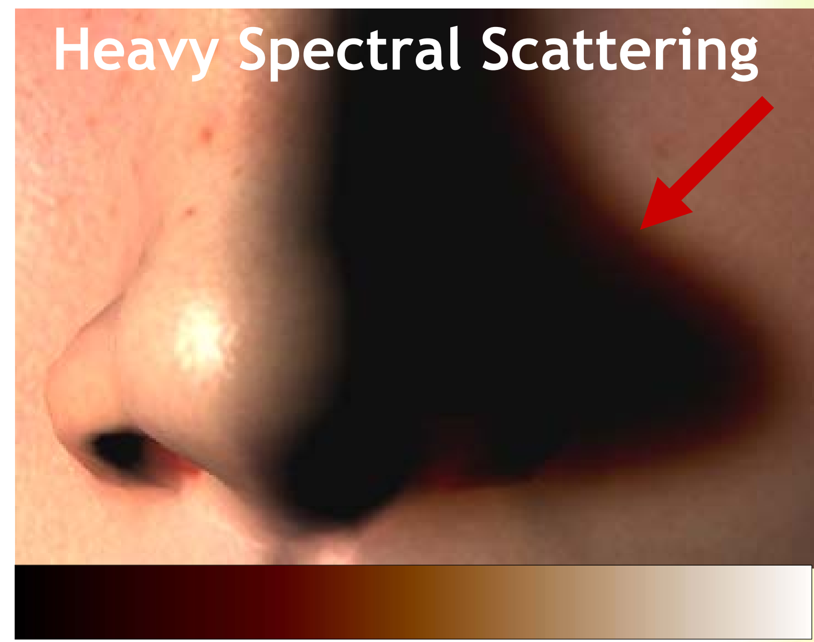
- Two effects which soften the appearance of skin:
  - Subsurface Scattering
  - Area Light Sources
- The amount of spectral scattering can be thought of as controlling the contribution of each.
- Spectral scattering approach:
  - Apply 1D color LUT to blurred visibility edges
  - Mimic effects of red light scattering in skin more than green light and blue light.

Spectral Scattering LUT



Example1D LUTs for Spectral Scattering

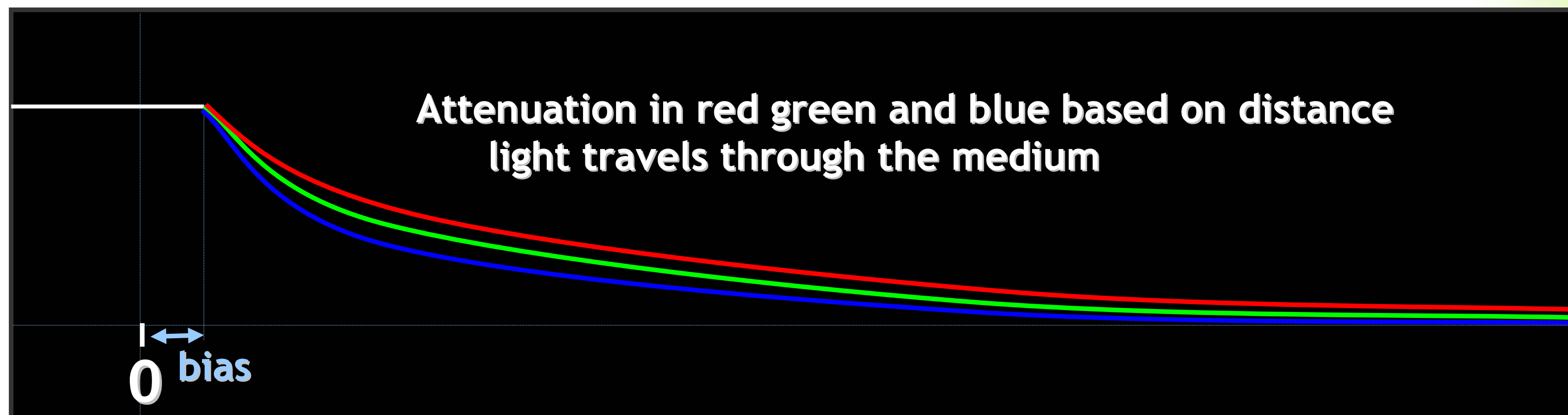




Shadow Map Lighting

- Standard shadow mapping uses a step function based on difference in depth values from the shadow map and from the light

Light Bleed Mapping



- Light Bleed Mapping replaces the step function with a negative exponential of the distance.
- This function represents how much the light is attenuated as it passes through the medium
- The basic idea: Look at the distance light has traveled though the skin to model the light bleeding though thin regions of the skin

Benefits of Light Bleed Mapping

- Not only is it inexpensive, but it also helps to alleviate two common problems with shadow mapping!
  - Smoothly varying function helps bias artifacts. Bias now has the effect of globally thinning the surface, so bias amount should be less than in standard shadow mapping.
  - Allows for the use of lighting models which bleed around to the backfaces of the object to some degree.
  - Too weak of an extinction coefficient can lighten shadows too much.

Light Bleed Mapping Applied

Without Bleed Mapping

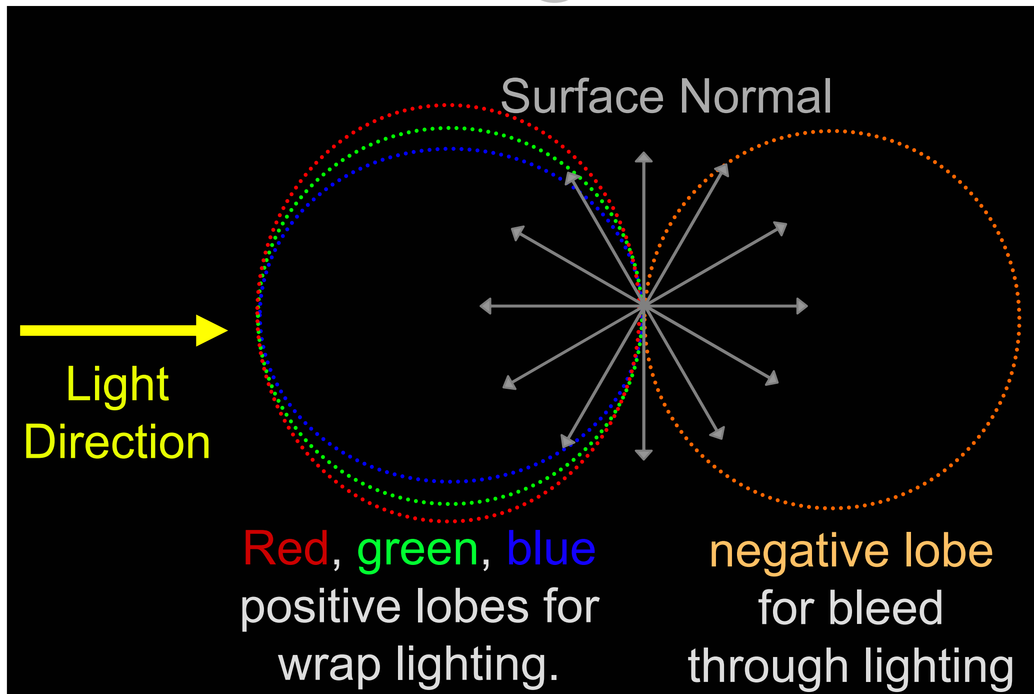
With Bleed Mapping

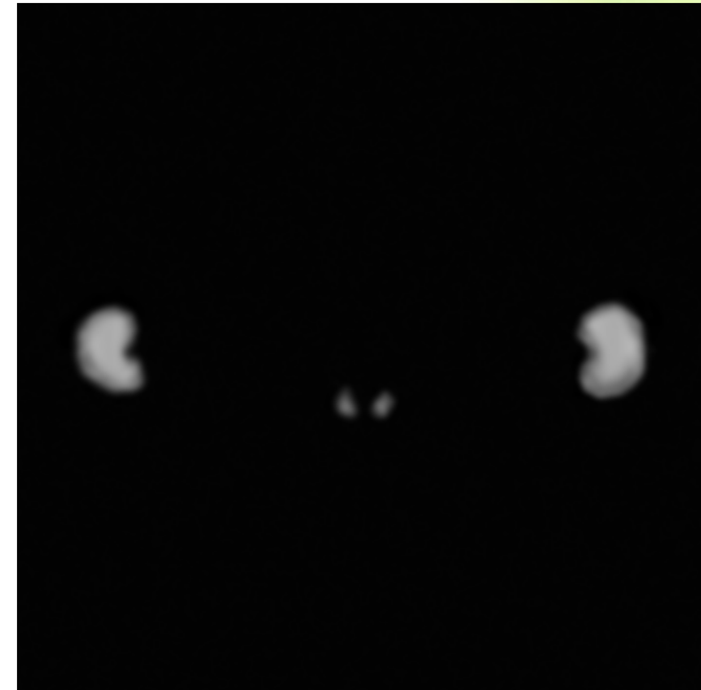
- **It turns out that this a simply a small variation on shadow mapping.**

Controlling Light Bleed Through

- Negative lobe for light that bleed through to the other side of objects.
- Amount of light bleeding through to the other side of the skin depends heavily on the material properties.
  - Bone, and fat makes most regions on the body more opaque.
- Use a light bleed map to mask out these opaque regions.
  - Can be used to control the thickness / light bleed amount of the skin also.
  - Very fast to author, and visualize the results. (as opposed to tweaking the geometry and passing it through the PRT pipeline multiple times)

Negative Lobe and Bleedmap for Light Bleed Through





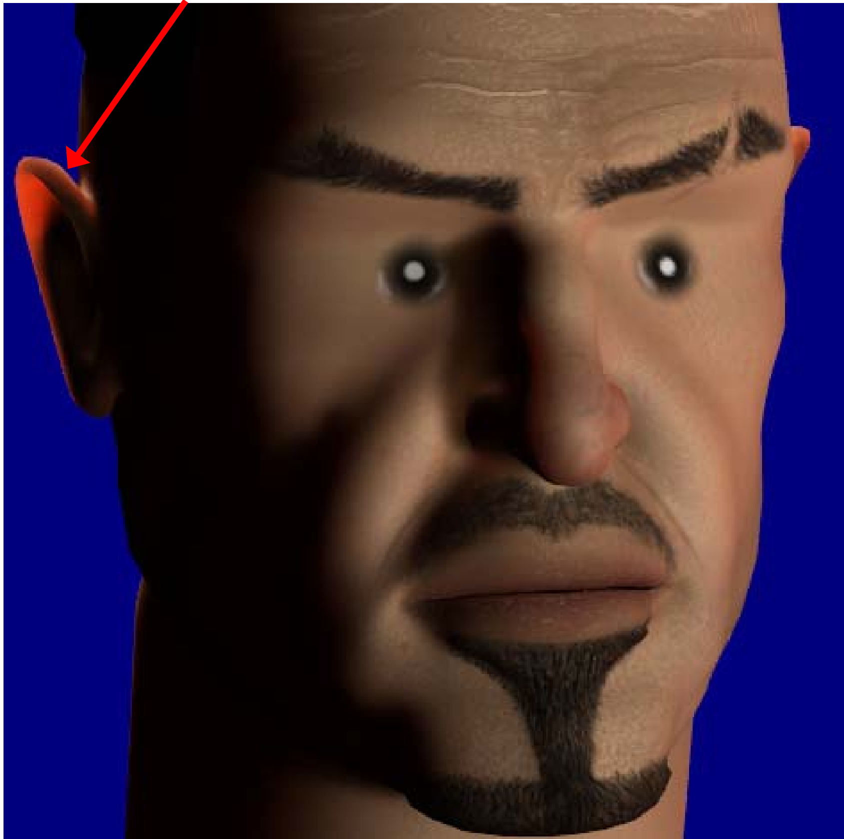
Ruby's Bleedmap to isolate ears and nostrils

- **Use Negative Lobe and Bleedmap**

Separate Bleed Through Blur Radius

- Light bleeding completely through objects will scatter more than localized scattering.
- One solution is to blur the bleed through channel more than the others.

Separate Bleed Through Blur Radius



Same blur kernel across all channels

Wider blur for bleed through channel

Overview of Real-Time Skin Approaches in Our Demos

- Ruby1: Image space lighting approach
  - Phong model, with additive (-L·V) rim lighting term.
  - Subsurface scattering modeled using image space blur with variable width kernel.
    - Larger blur radius for thinner regions like ears and nostrils
- Ruby2: PRT based approach
  - Uses PRT computed using a measured skin model.
  - Exhibits color shift and light propagation through thin structures.

Acknowledgements

- Big thanks to John Isidoro and Chris Oat for their great work

References

- [Jensen01] H.W. Jensen, S.R.Marschner, M. Levoy, P. Hanrahan, "A Practical Model for Subsurface Light Transport", SIGGRAPH 2001
- [Borshukov03] G. Borshukov and J.P. Lewis, “Realistic Human Face Rendering for *The Matrix Reloaded*,” Technical Sketches, SIGGRAPH 2003
- [Green04] Simon Green, “Real-Time Approximations to Subsurface Scattering,” GPU Gems 2004
- [Annen04] Tomas Annen, Jan Kautz, Fredo Durand, and Hans-Peter Seidel, “Spherical Harmonic Gradients for Mid-Range Illumination,” Proceedings of Eurographics Symposium on Rendering, June 2004
- [Mertens03] Tom Mertens, Jan Kautz, Philippe Bekaert, Hans-Peter Seidel and Frank Van Reeth, “Efficient Rendering of Local Subsurface Scattering ,” Proceedings of Pacific Graphics 2003
- [Sander04] Pedro V. Sander, David Gosselin and Jason L. Mitchell "Real-Time Skin Rendering on Graphics Hardware," SIGGRAPH 2004 Technical Sketch. Los Angeles, August 2004
- [Sloan04] Peter-Pike Sloan and Jason Sandlin, “Practical PRT” Microsoft DirectX Meltdown 2004

Questions?