GDC 2007 Demo Team Secrets: Advanced Skin Rendering

Eugene d'Eon

Outline

- Demos: Adrianne and Froggy
- Quick Overview of where we're going
- Reflectance properties of real skin
- Review of current techniques
- Ways to improve real-time skin rendering

Demos

Quick Overview

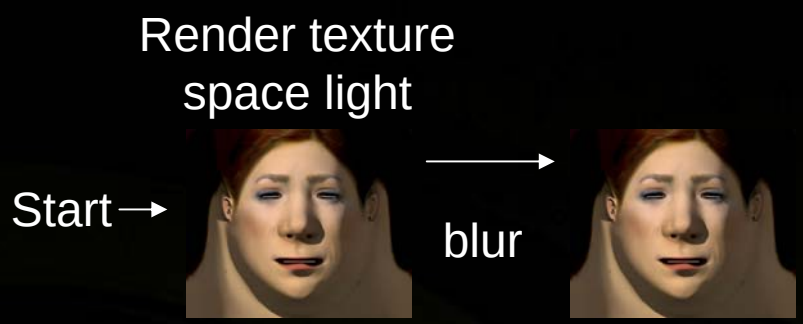
- How are these images generated?

Overview Diagram

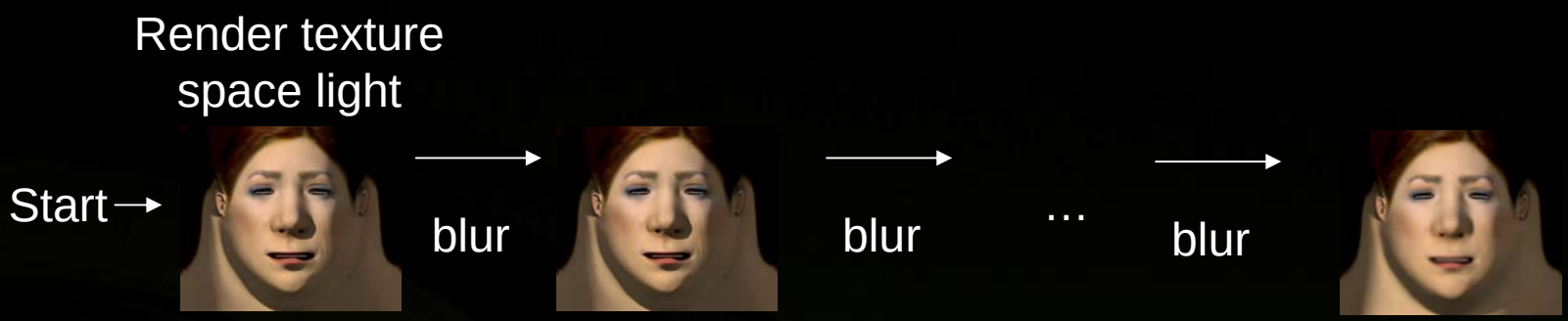
Render texture space light

Start →

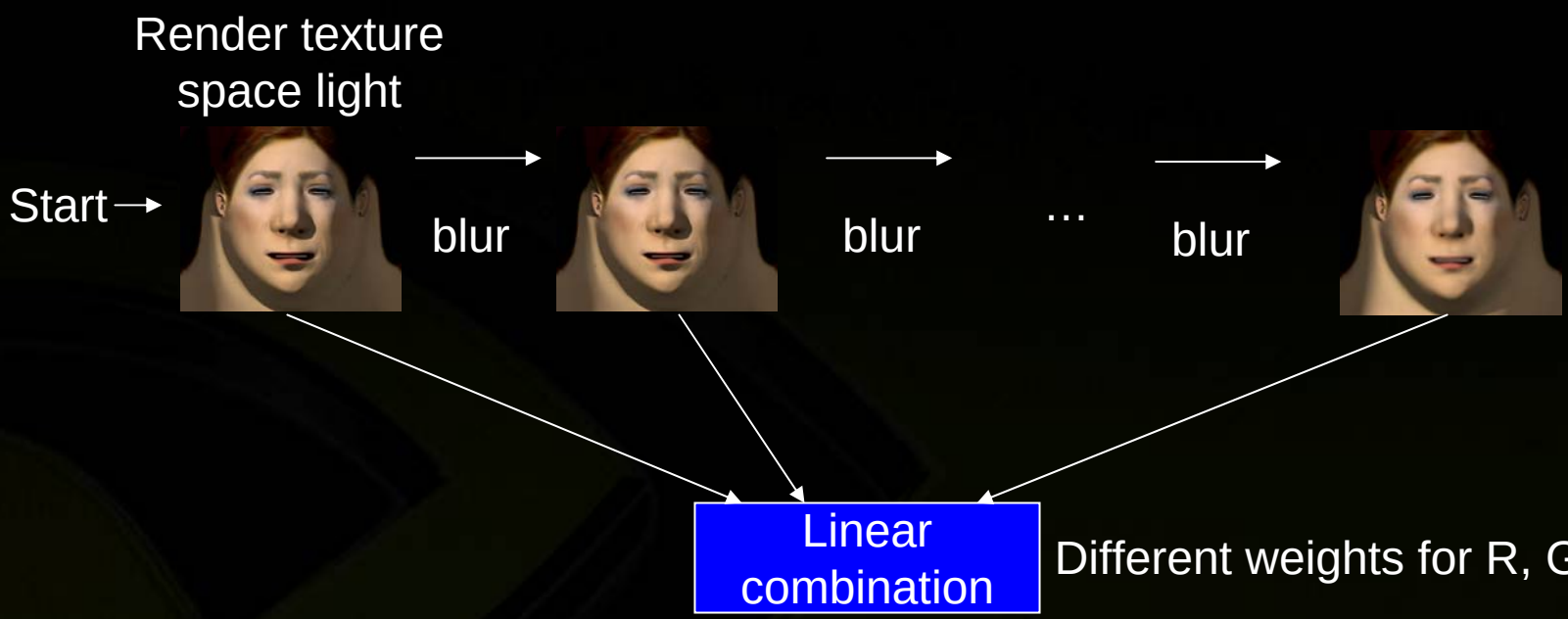
Overview Diagram



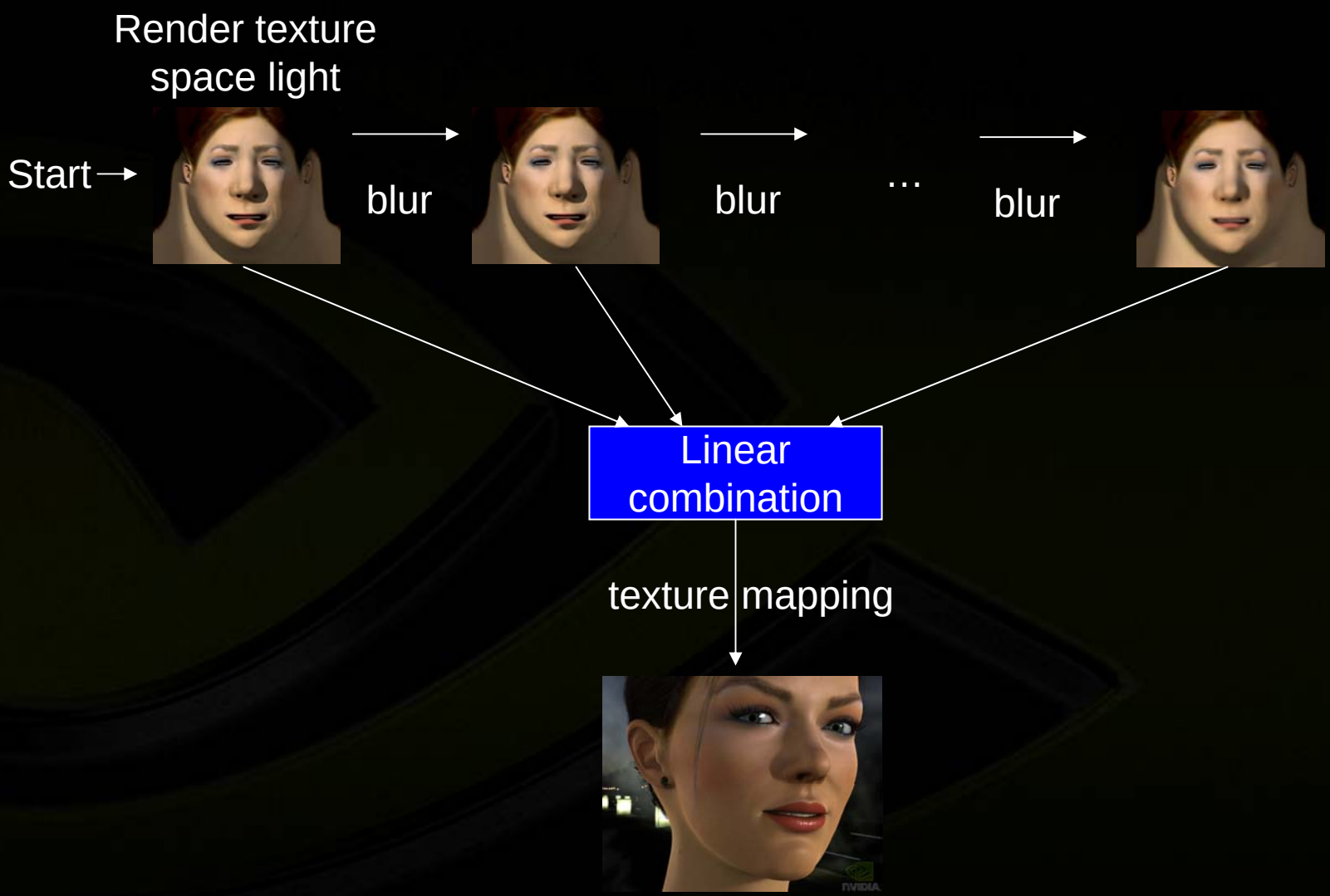
Overview Diagram



Overview Diagram

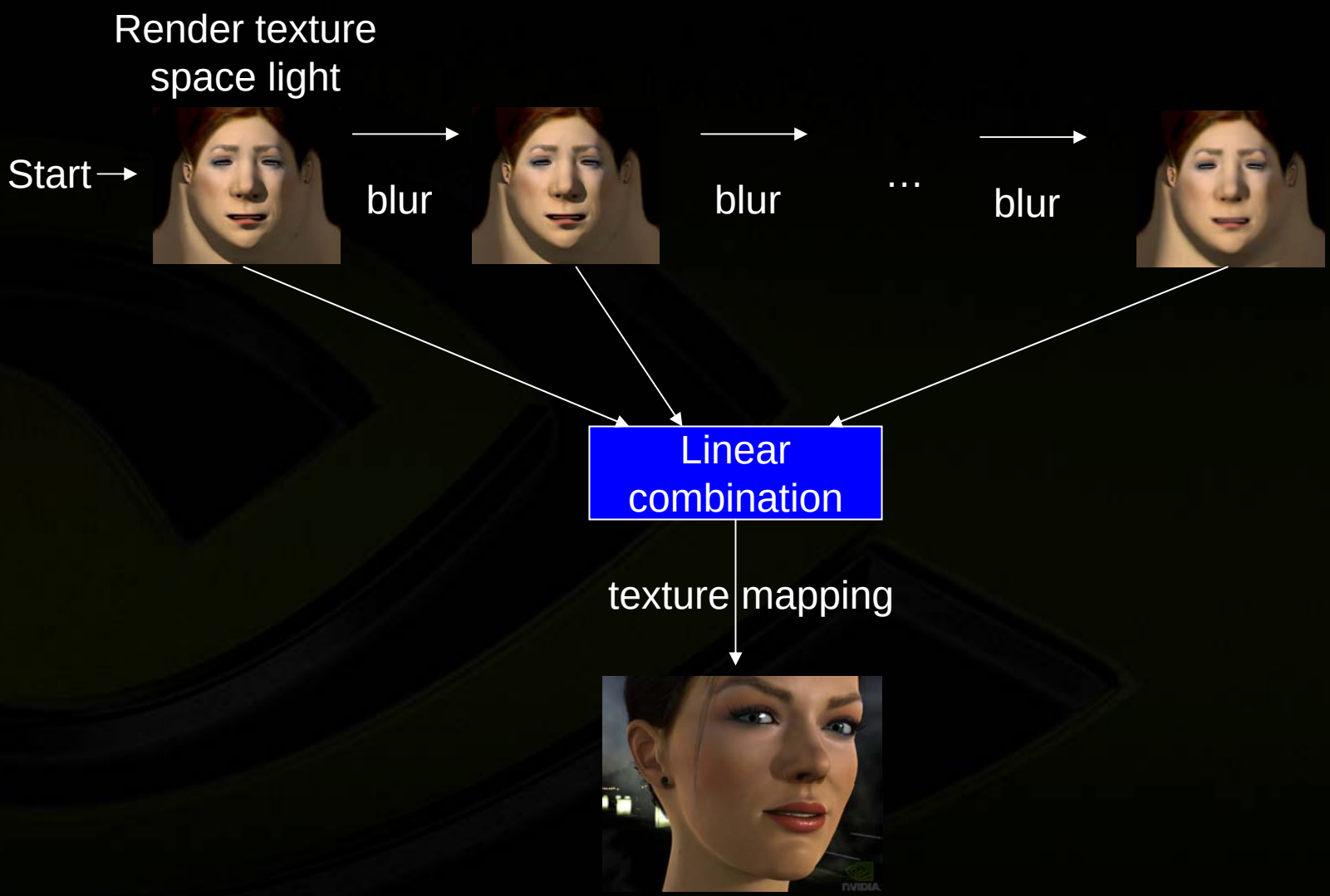


Overview Diagram



Final pass: combine blurs

Overview Diagram



Final pass: combine blurs + specular

Pre-computed Material

- Ambient occlusion map
  - (static)
- Hand painted maps
  - Diffuse Color
  - Bump
  - Specular Brightness

Pre-computed ambient occlusion

Diffuse Color map

Bump map

Specular Brightness map

- (brightened here by 16X for display purposes)

Computed each frame

- Subsurface Irradiance
- Several blurred versions of Subsurface Irradiance
- Blend: Final Subsurface Radiance
- Specular
- Rim light (optional)

Start with incident light (irradiance)

- (Here we show just a single point light)

Compute Subsurface Irradiance

- (explained later – actually computed in an off-screen texture)

Blur Subsurface Irradiance

Blur Subsurface Irradiance Again

Blur Subsurface Irradiance Again

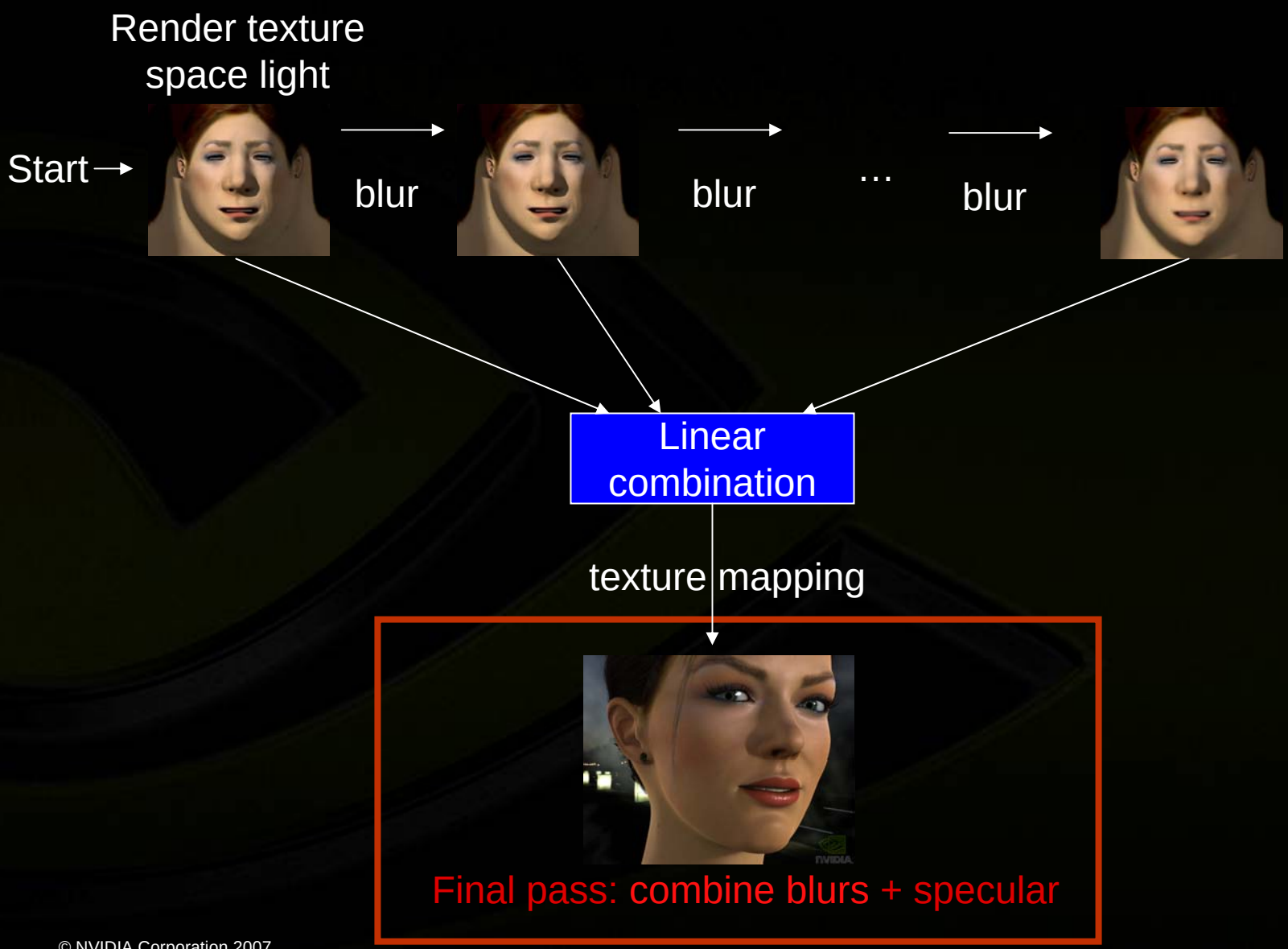
Blur Subsurface Irradiance Again

Blur Subsurface Irradiance Again

Final render pass

- Render mesh in 3D
- Combine blurry textures
- Add specular and rim lighting

Overview Diagram



Final pass

Ummm… What?

- Why blurring?
- Why keep around several blurs?
- Why different recombination weights for R, G, and B?
- Why isn’t specular blurred?

Reflectance properties of skin

Reflectance properties of skin

- Dominated by subsurface scattering
  - ~6% direct reflection, 94% subsurface*

*[KRISHNASWAMY, A., AND BARONOSKI, G. V. G. 2004]

Reflectance properties of skin

- Dominated by subsurface scattering
  - ~6% direct reflection, 94% subsurface*
- Reflectance and *Scattering* are *different* in red, green and blue wavelengths

*[KRISHNASWAMY, A., AND BARONOSKI, G. V. G. 2004]

Reflectance properties of skin

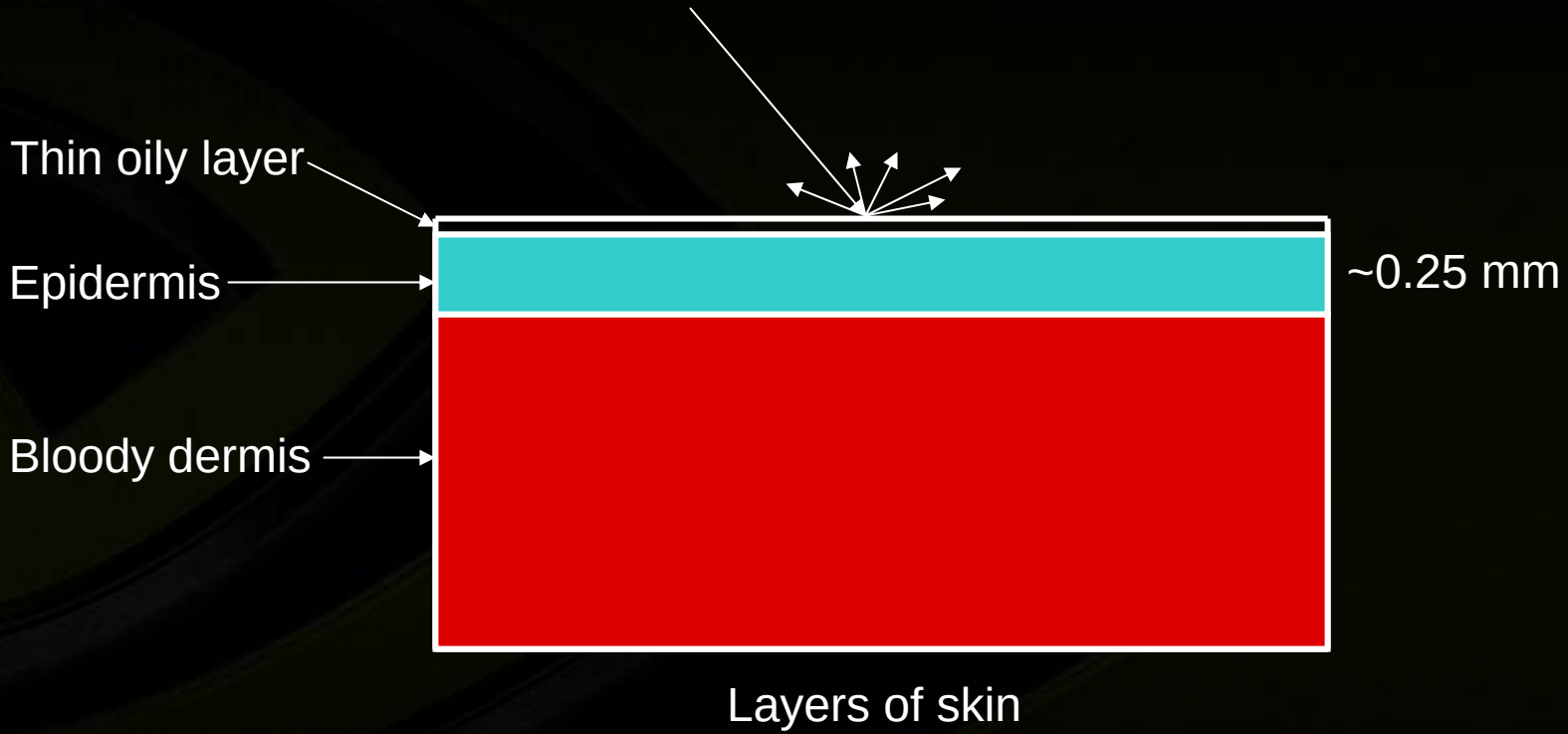
- Dominated by subsurface scattering
  - ~6% direct reflection, 94% subsurface*
- Reflectance and *Scattering* are *different* in red, green and blue wavelengths
- Scattering is poorly modeled by assuming a single layer of roughly uniform material

*[KRISHNASWAMY, A., AND BARONOSKI, G. V. G. 2004]

Why is this?

Direct reflection

- How much light reflects off the oily layer?



Direct Reflection

- ~6% of the light*
- Change of index of refraction between air and skin
- Fresnel reflection & refraction takes place

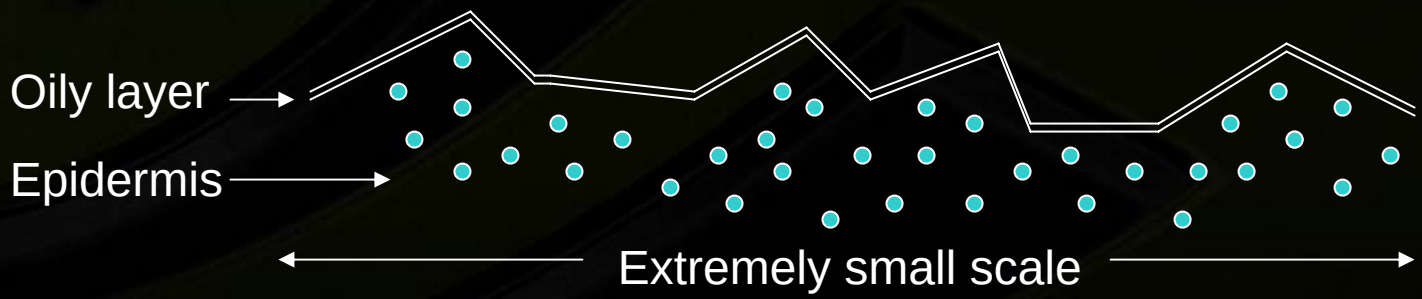
*[KRISHNASWAMY, A., AND BARONOSKI, G. V. G. 2004]

Specular

- This is typically treated as a specular term
- Highly dependant on both view direction and incoming light direction
- Not colored by the surface in any way

Specular - Roughness

- Surface is not a mirror
- Skin has very small scale roughness
- Use a physically based BRDF*



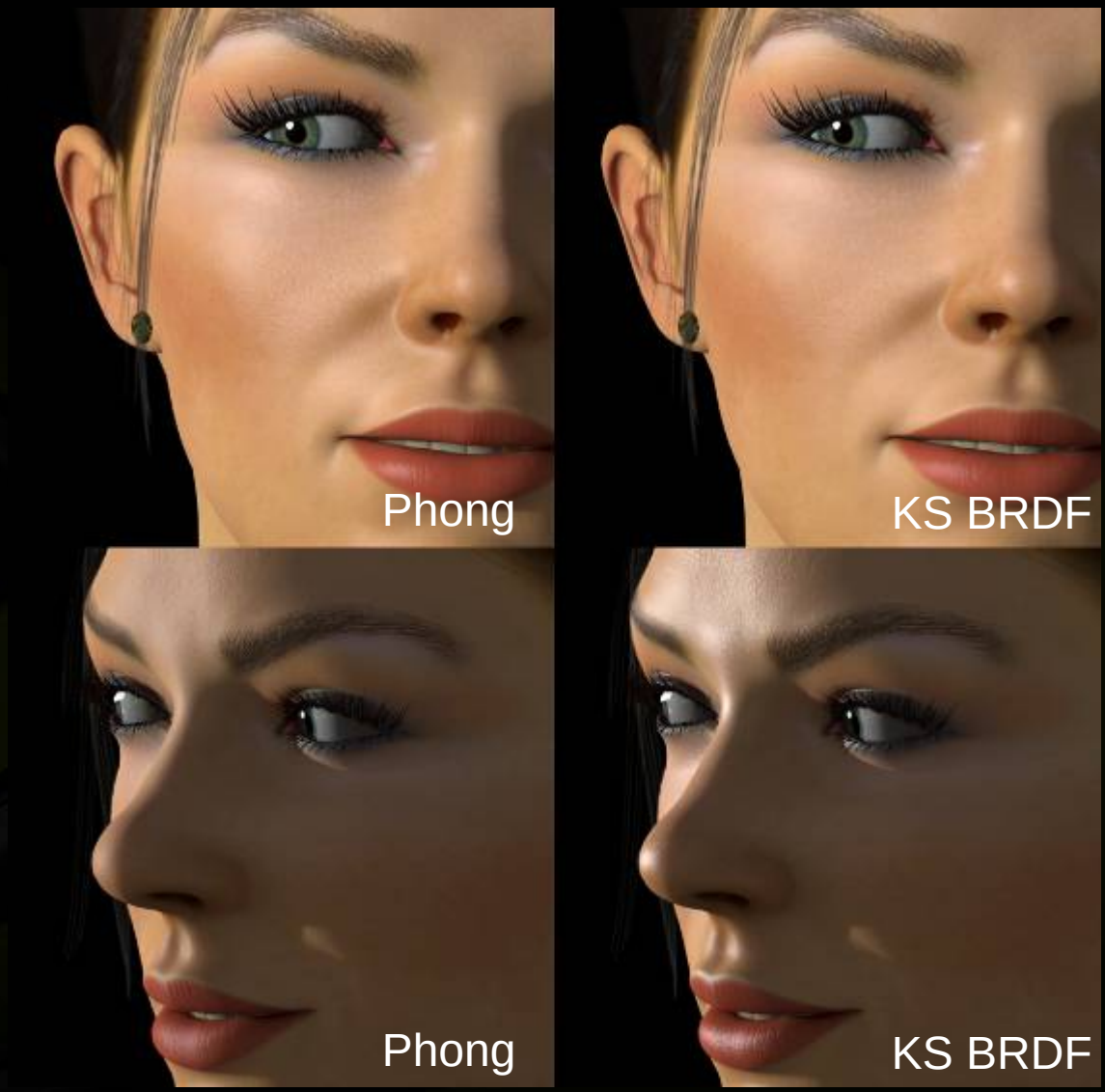
*Bidirectional Reflectance Distribution Function

Specular - Roughness

- Most specular BRDFs have
  - Roughness parameter “m”
  - Index of refraction (use 1.4)*
- Phong and Blinn-Phong aren’t ideal for skin

*[Donner and Jensen 2005]

Phong vs. physically-based BRDF



*[Kelemen and Szirmay-Kalos 2001]

Phong vs. physically-based BRDF

- A physically based BRDF captures increased specularity at grazing angles
- Here we use Kelemen-Szirmay-Kalos 2001

Roughness parameters

- How do we set roughness parameter m?
- 0.3 is a good average value for the face*
- Look to current research (Weyrich et al 2006)

*[Donner and Jensen 2005]

Roughness parameters

- **Average measured values from many real faces – Weyrich et al. 2006 (SIGGRAPH)**

Specular amount Roughness, m

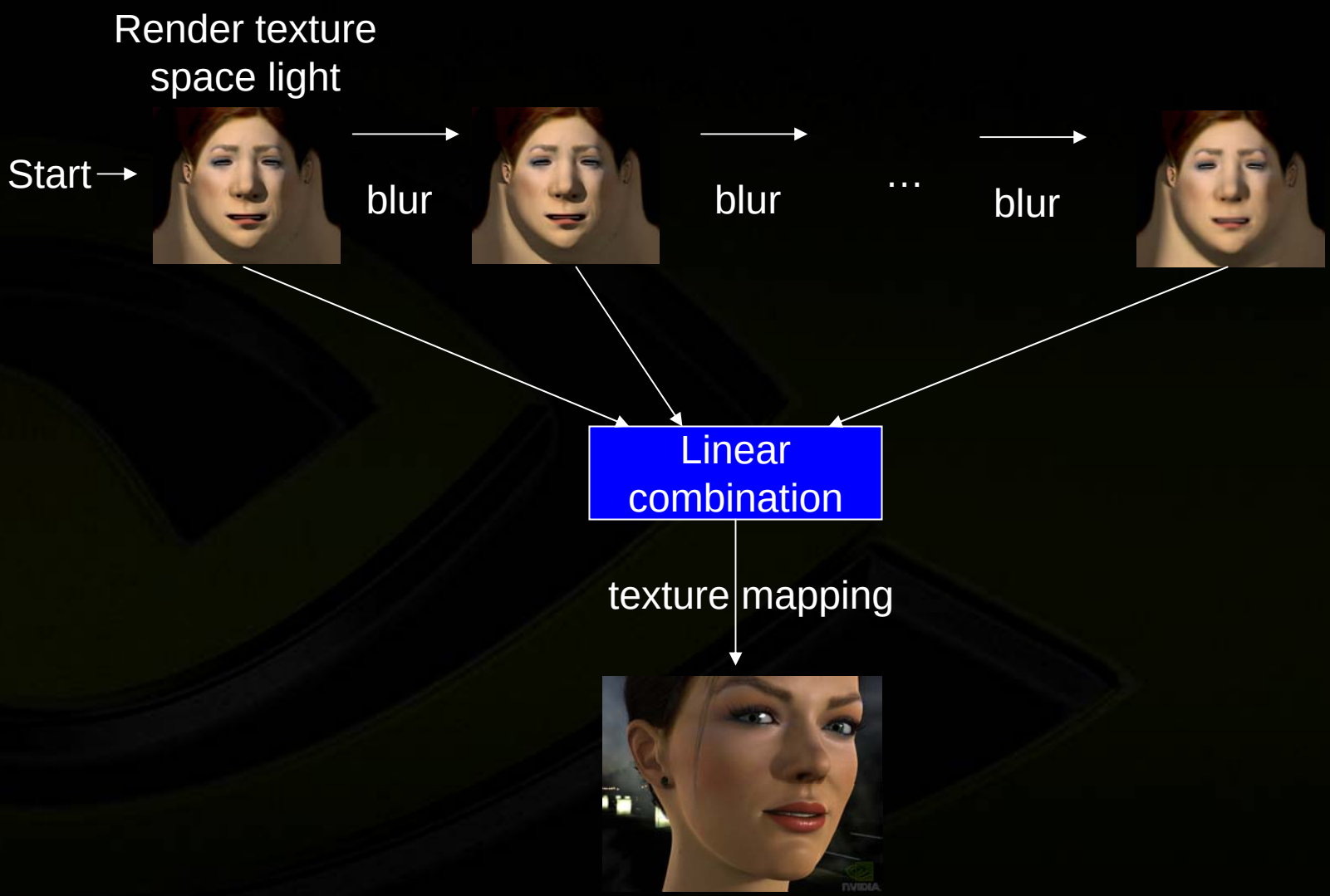
Debevec et al 2000 trick

- Linearly polarized light
- Linearly polarized camera
- Some magic
- Pictures of JUST specular!
- Check out the paper (SIGGRAPH 2000)
- http://www.debevec.org/Research/LS/

*

*Image courtesy of Paul Debevec

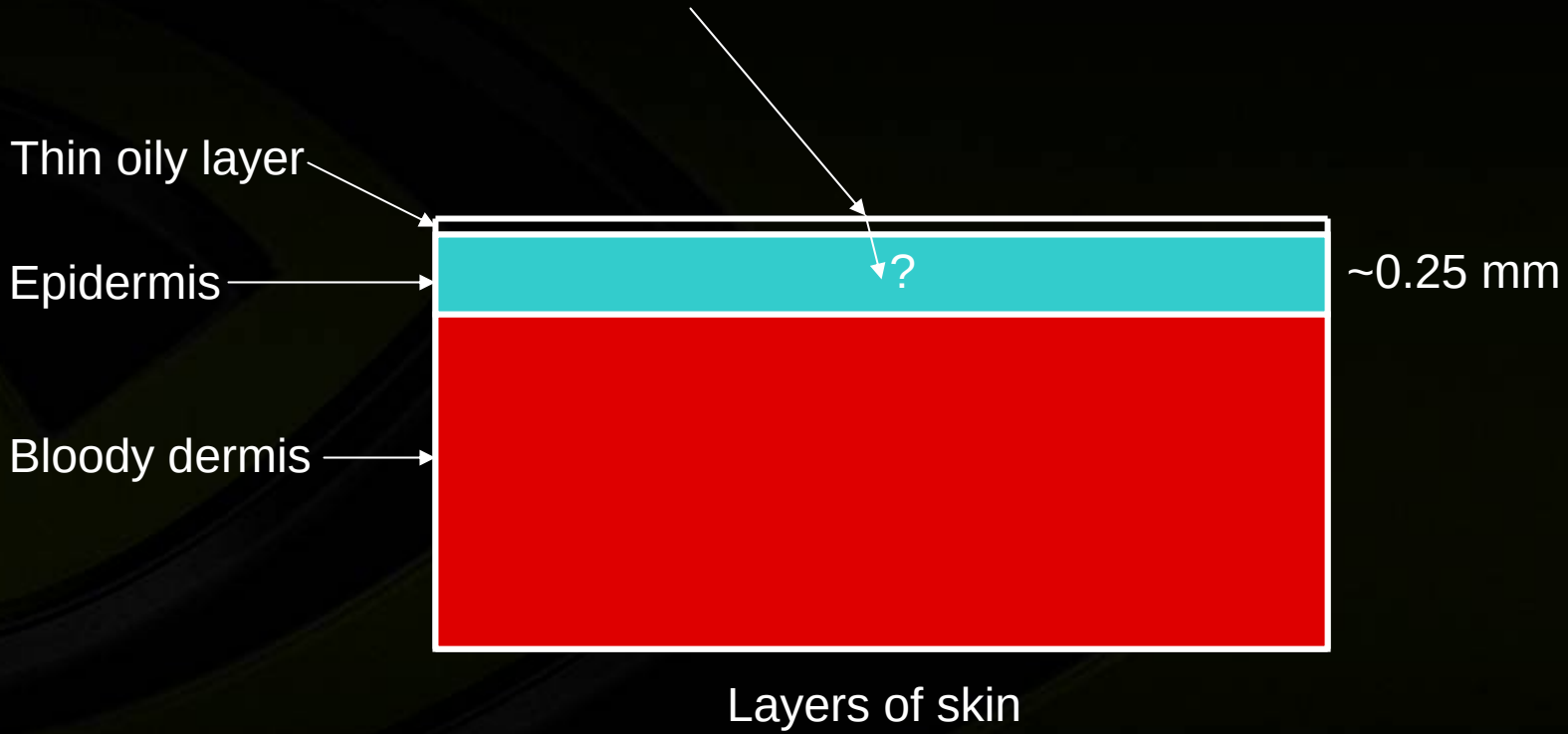
Overview Diagram



Final pass: combine blurs + specular

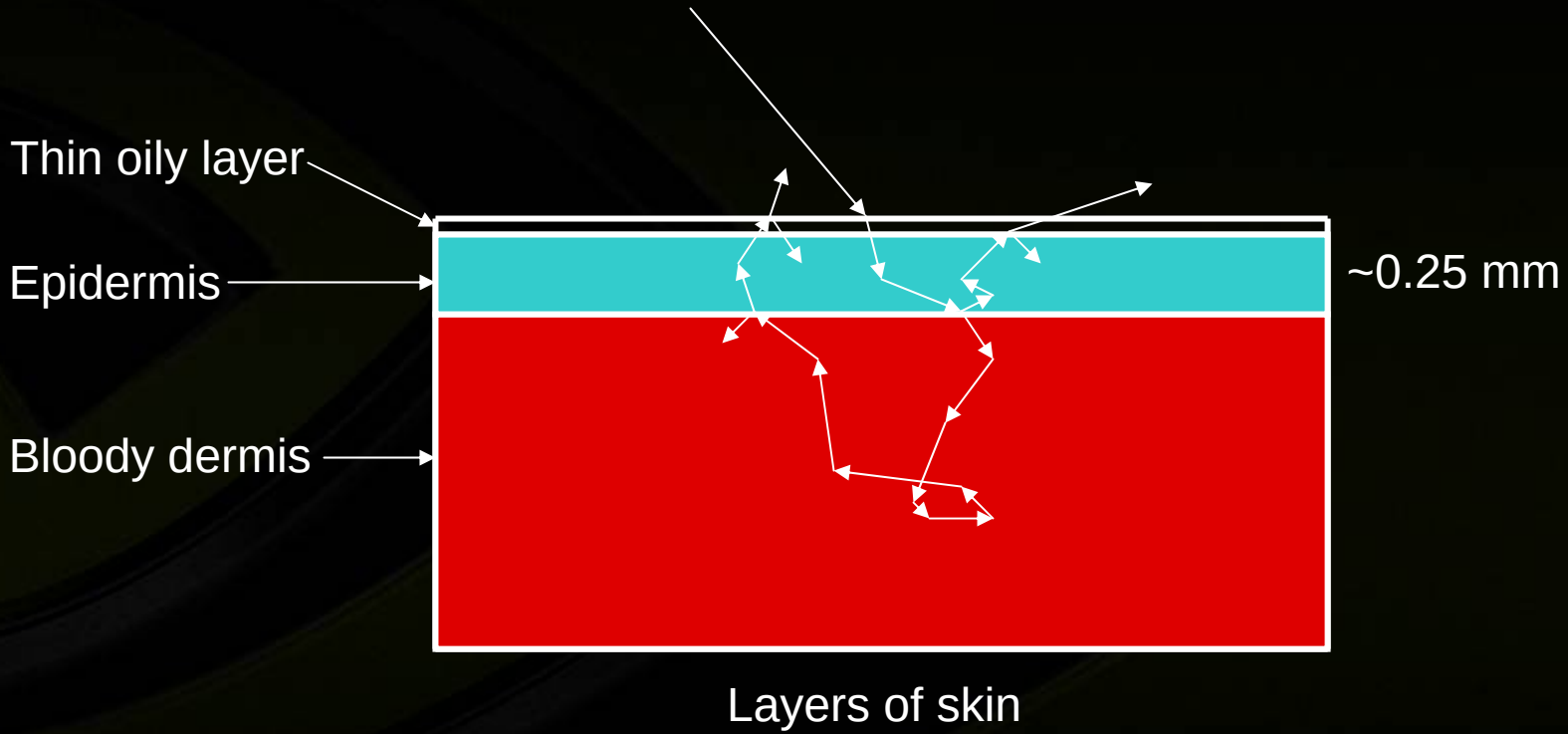
What about the rest of the light?

- How do we handle the subsurface light?



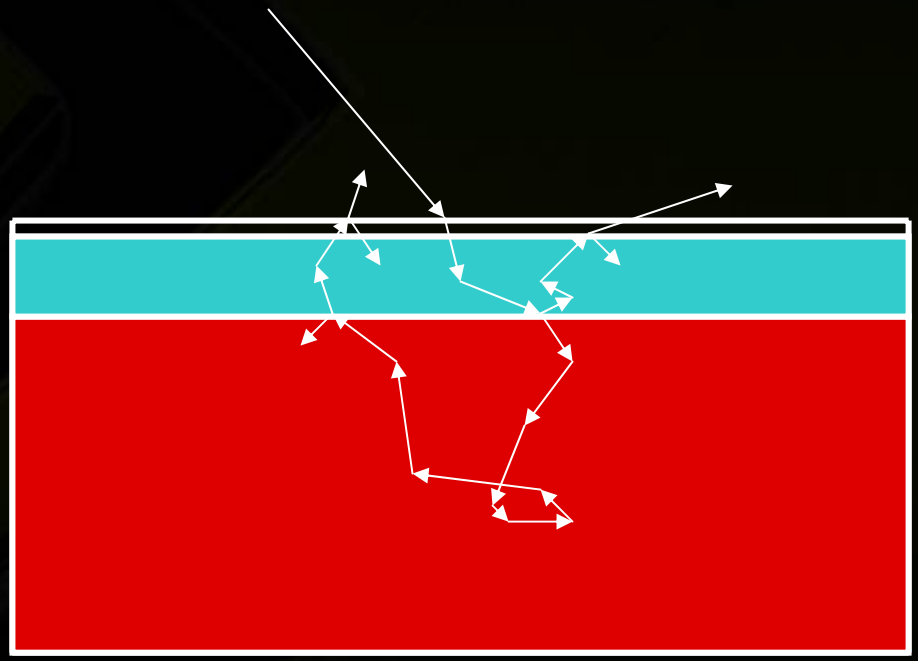
Subsurface scattering

- **underneath layers dominated by subsurface scattering**



Subsurface Scattering

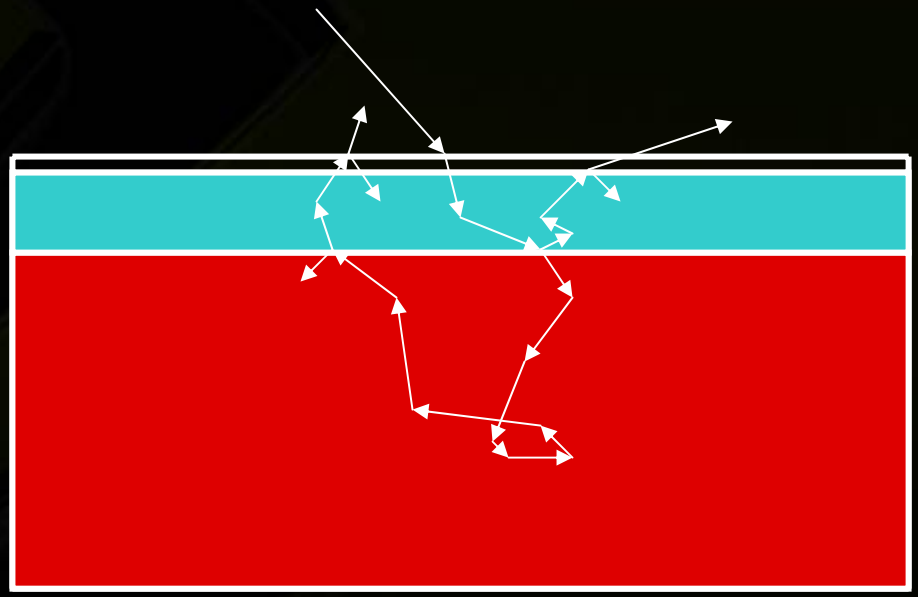
- Light comes out somewhere else
- Light gets colored depending on where it went



Layers of skin

How do we deal with this?

- Seemingly impossible task:
  - infinite number of possible paths
  - What about directional effects?



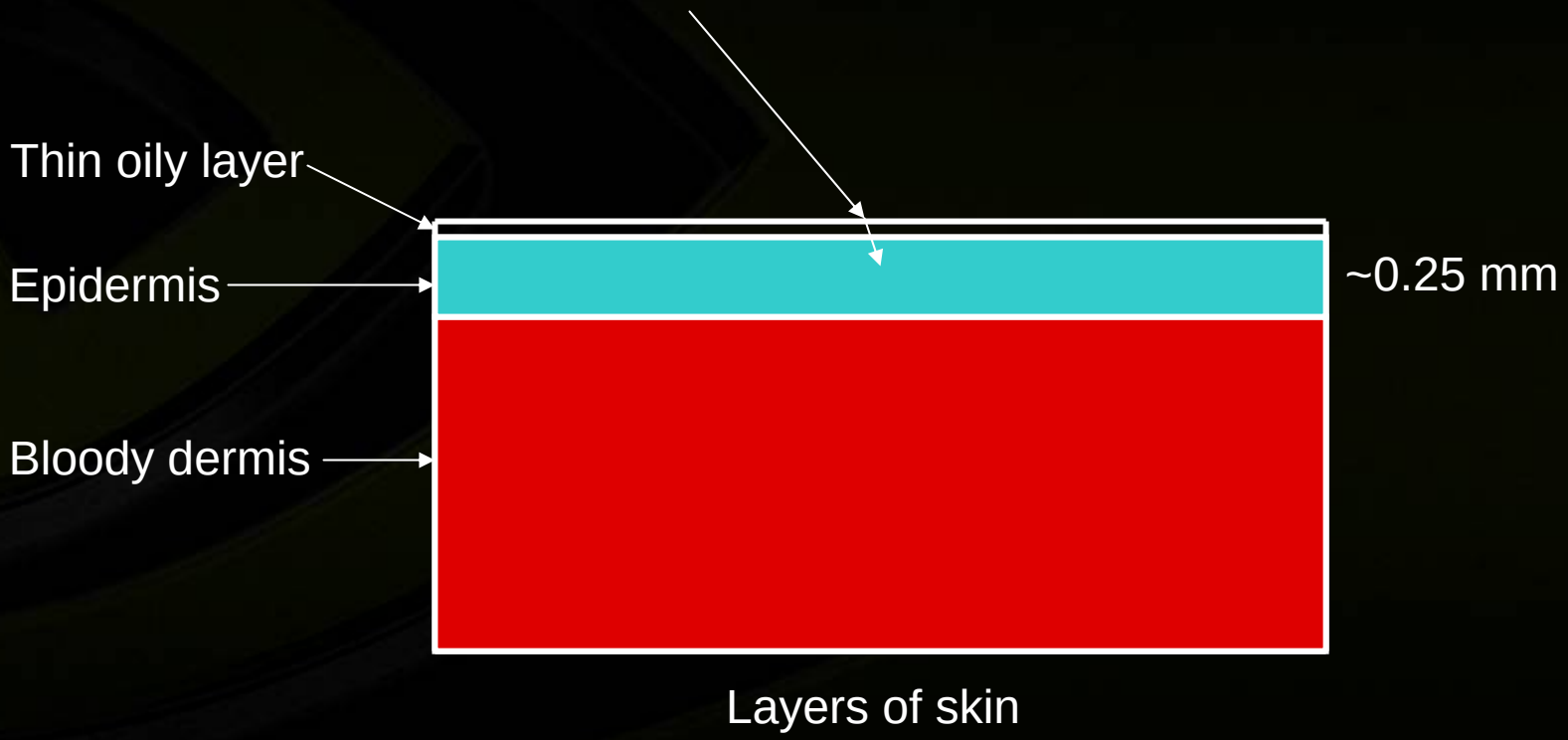
Layers of skin

Transport Theory

- Scattering models from physics
  - How far between scattering events?
  - How far between absorption events?
- Many scattering events make light diffuse
  - How long until that happens?

Transport Theory

- **For skin this distance is quite short:**
  - **0.02083mm (red wavelength)***



*Computed using three-layer skin parameters from [Donner and Jensen 2005]

Transport Theory

- 1/10th through first layer – We're diffuse!
- Track total light – ignore direction
- Any light reflection back is diffuse (equal in all directions)

Sweet, we know how to do this

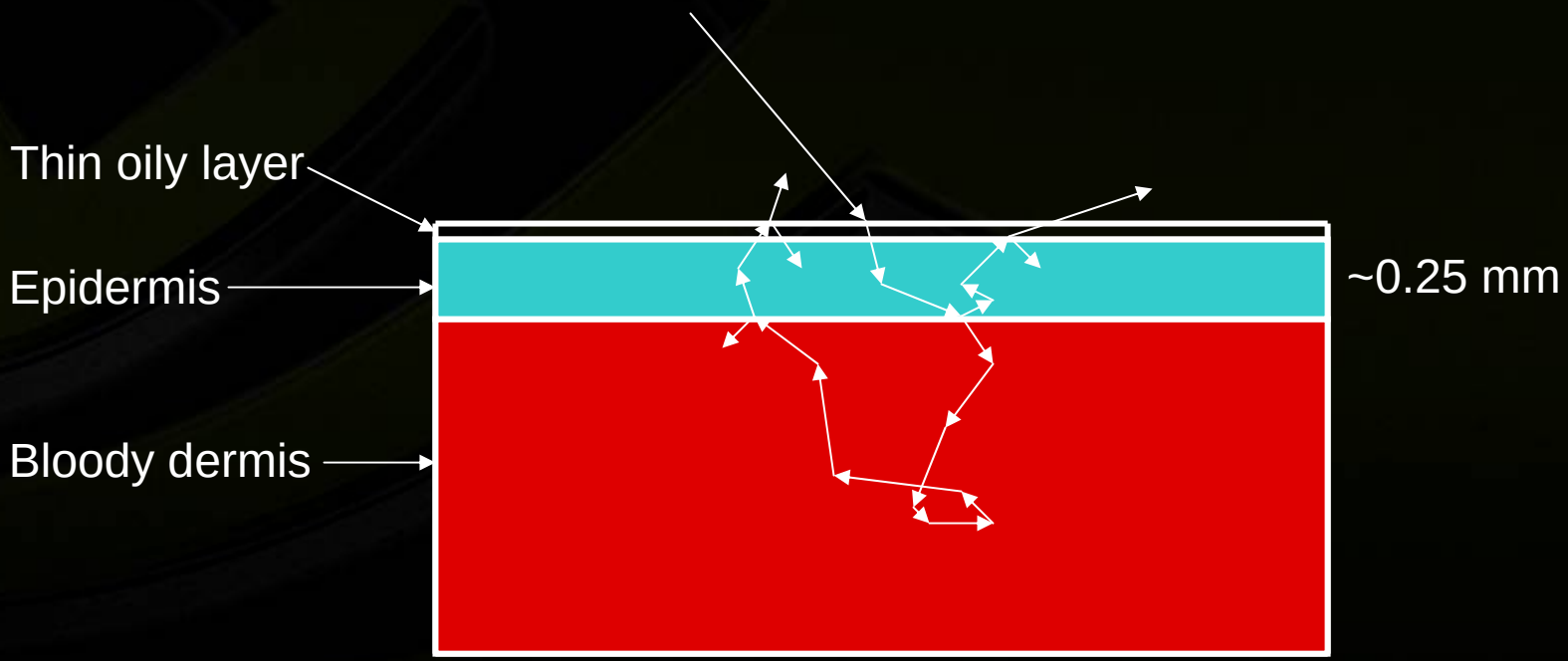
- Do an (N • L) for each light and multiply by the diffuse color
- Then add specular
- Easy, right?

Wrong

- The skin looks dry and hard

Why didn't this work?

- We forgot two things
  - We didn't handle how light enters in one location and exits in another (Important!)
  - The oily rough surface has non-uniform transmittance (subtle)



So what now?

- We need to employ more complicated methods
- Rely heavily on a diffusion approximation

Previous techniques

- Offline rendering:
  - Monte-carlo rendering of multi-layered materials
- Real-time
  - Texture space diffusion

Multi-layer 2005

*Image Courtesy of Craig Donner and Henrik Wann Jensen

Multi-layer 2005

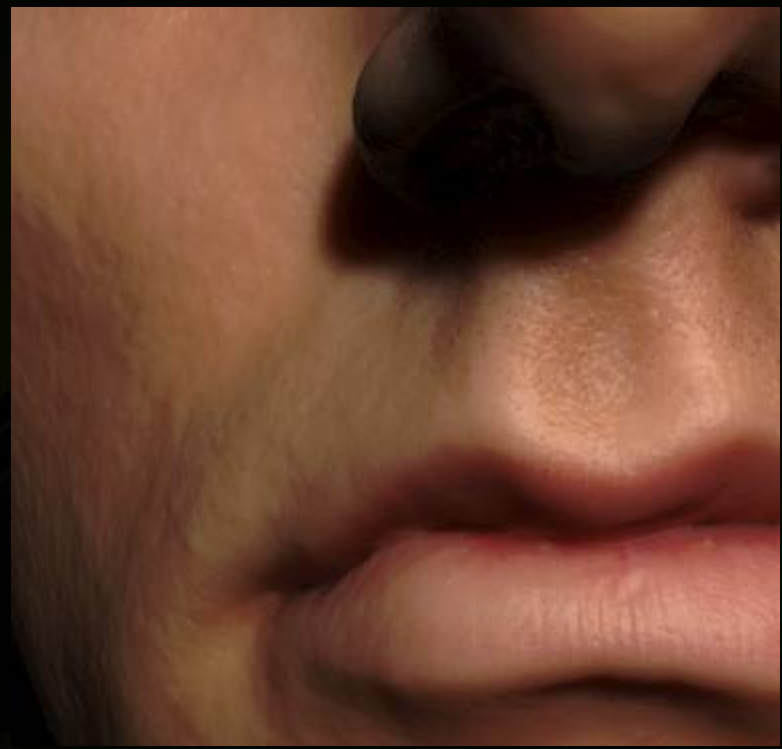
- 2005 SIGGRAPH paper by Craig Donner and Henrik Wann Jensen
  - 3 layer skin model
  - Scattering parameters measured in the medical/optics community
  - High quality head scan (XYZRGB)
  - ~5 minutes rendering time

*Image Courtesy of Craig Donner and Henrik Wann Jensen

Three-Layer vs. dipole

- In particular they showed the importance of a multi-layered skin model

3 layer skin model



1 layer skin model

*Images Courtesy of Craig Donner and Henrik Wann Jensen

What's going on here?

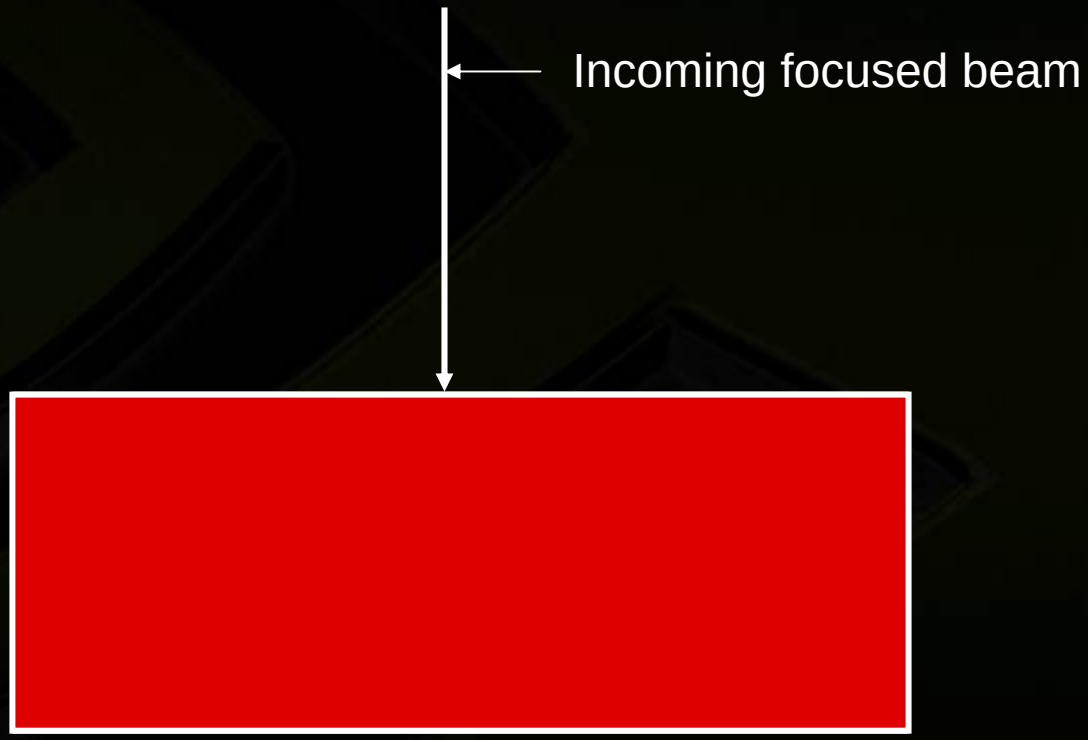
- One-layer skin model looks waxy
- Epidermis
  - Narrow scattering
- Lower layers
  - Wide scattering
  - Mostly red
- Assuming one layer doesn't work

Approximating multi-layer diffusion

- We can approximate this
- Start with incident light
- Blur over surface
- Blur some more
- Mix (linear combination)

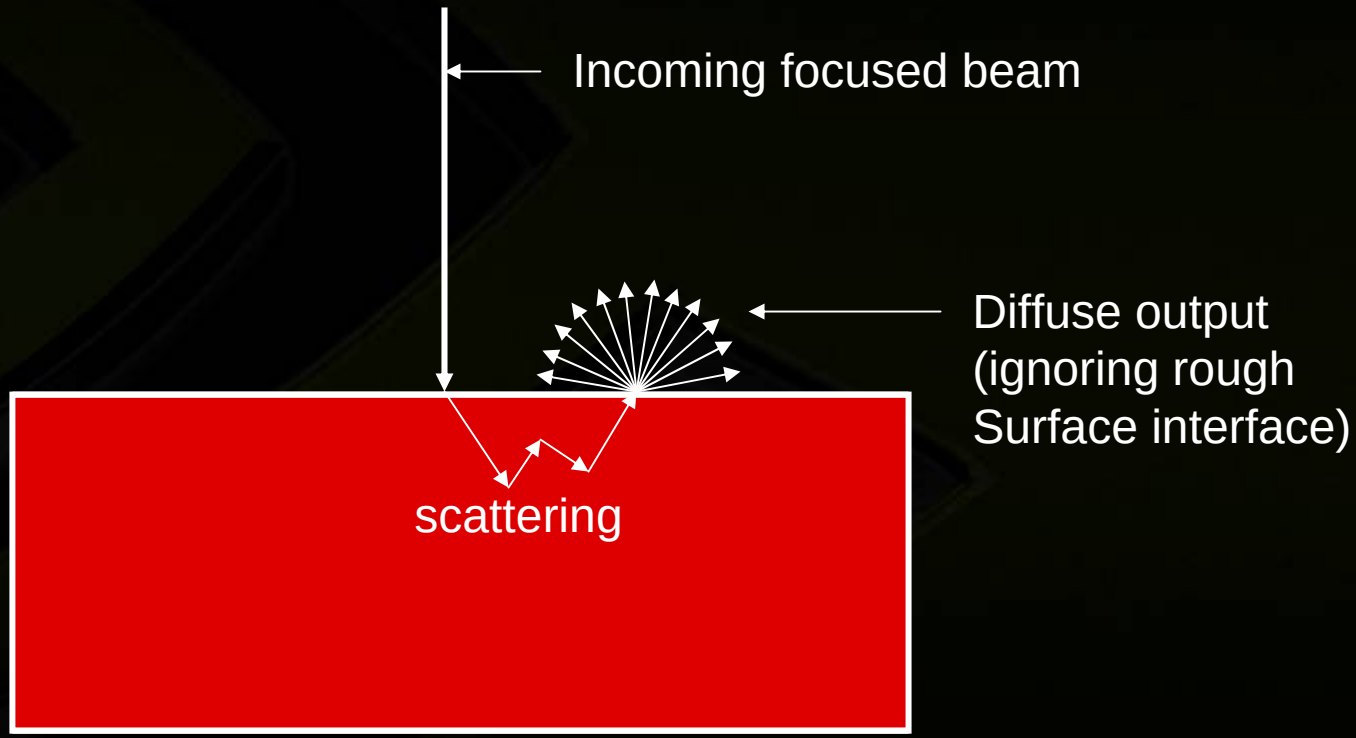
Why a blur?

- Consider a narrow focused white beam of light hitting a flat, highly scattering surface



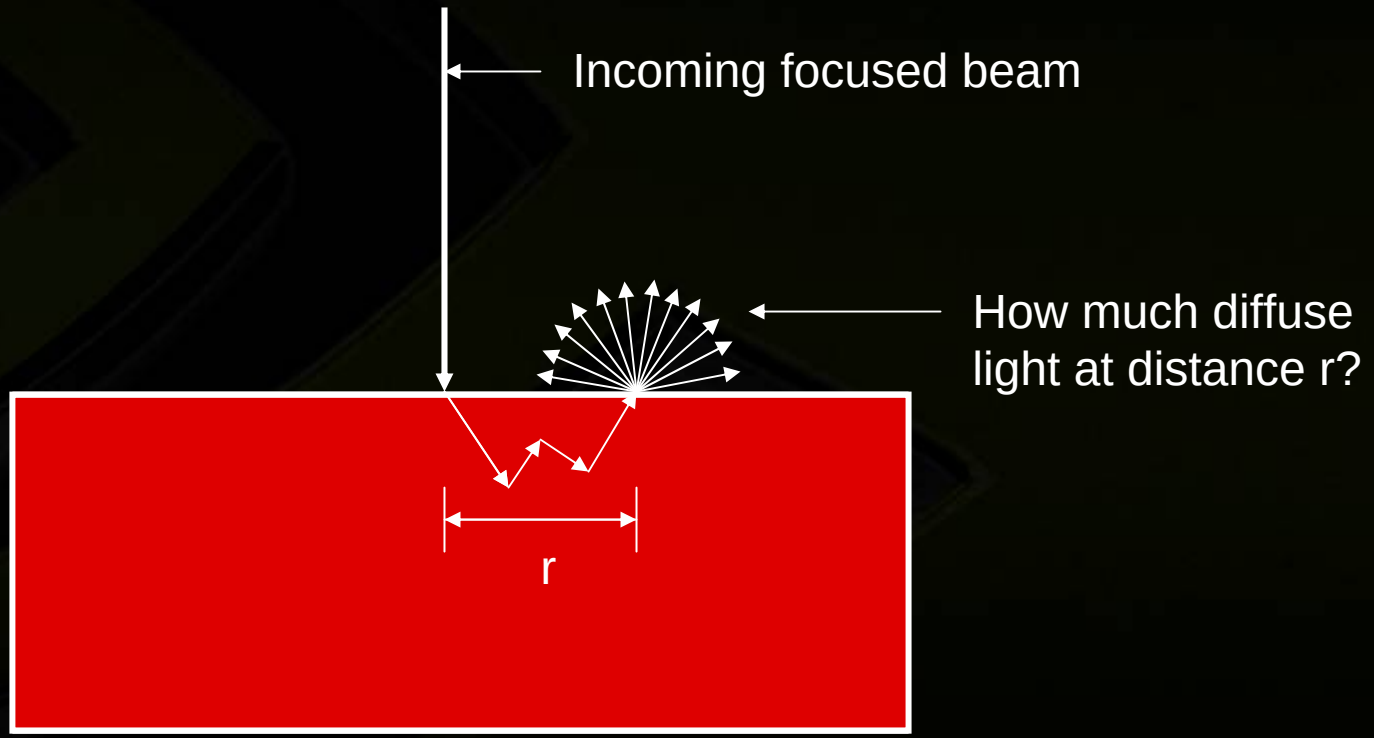
Why a blur?

- “Highly scattering” means diffuse



Diffusion profiles

- How much light at distance r?
- Name: diffusion profile of the material



Diffusion profiles

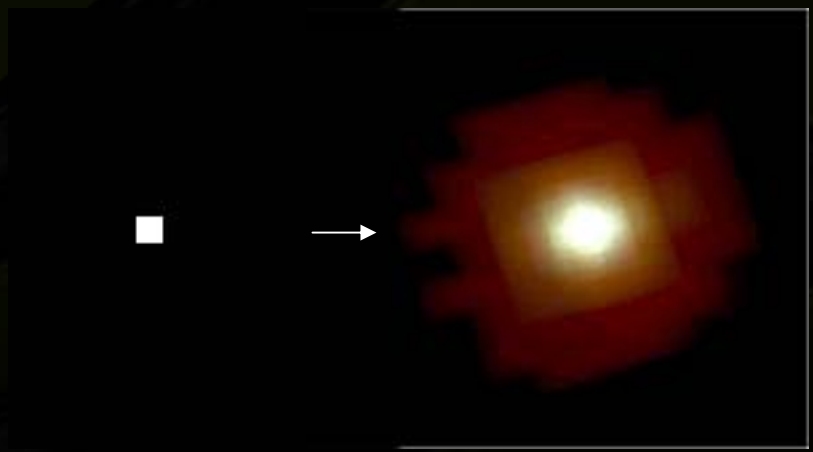
- We could compute this
- What matters for skin?
  - Distinctly different in R, G, B
  - Shape of these profiles means waxy skin vs. real

Main scattering idea

- For every surface point
  - Collect incoming light (color value)
  - Diffusion approximation: sum ignoring direction
  - Scatter into neighboring surface points

Main scattering idea – 2D flat surface

- 2D flat surface
- Each point
  - Gather initial light
  - Scatter into neighboring pixels
  - Depends on separation, r
  - This is a blur!



Diffusion over curved surfaces

- We actually require more complicated surfaces

Vertical laser line reflecting off an ear

Texture space diffusion

- Unfold 3D surface – texture coordinates
- Use this layout to perform blurs
- Map back after using texture mapping
- We call this texture space diffusion*

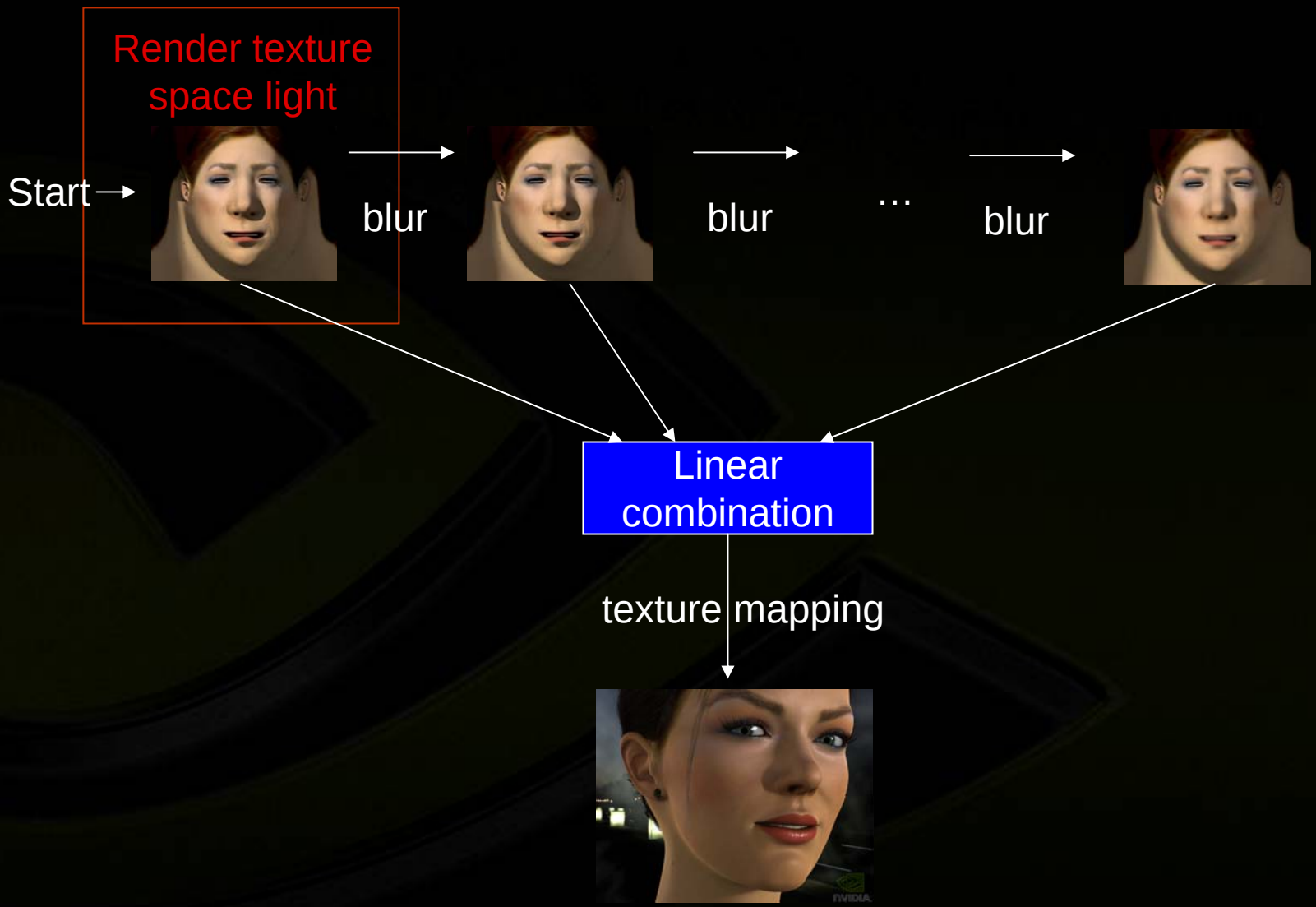
*Invented by George Borshukov and J.P. Lewis 2003

Texture space diffusion

- This has been done in real-time before (Simon Green: GPU Gems 1)*
- We improve upon this technique in several ways
  - Correct distortion
  - Keep several blurs and linearly combine
  - How to use diffuse color

*Invented by George Borshukov and J.P. Lewis 2003

Overview Diagram



Final pass: combine blurs + specular

Rendering using texture coordinates

- Modify the projCoord in the vertex shader using the texture coordinates
- Pass the true worldCoord and worldNormal to the fragment shader for lighting

```
v2f.projCoord = float4( texCoord.x * 2.0 - 1.0,
   texCoord.y * 2.0 - 1.0, 0.0, 1.0 );
```

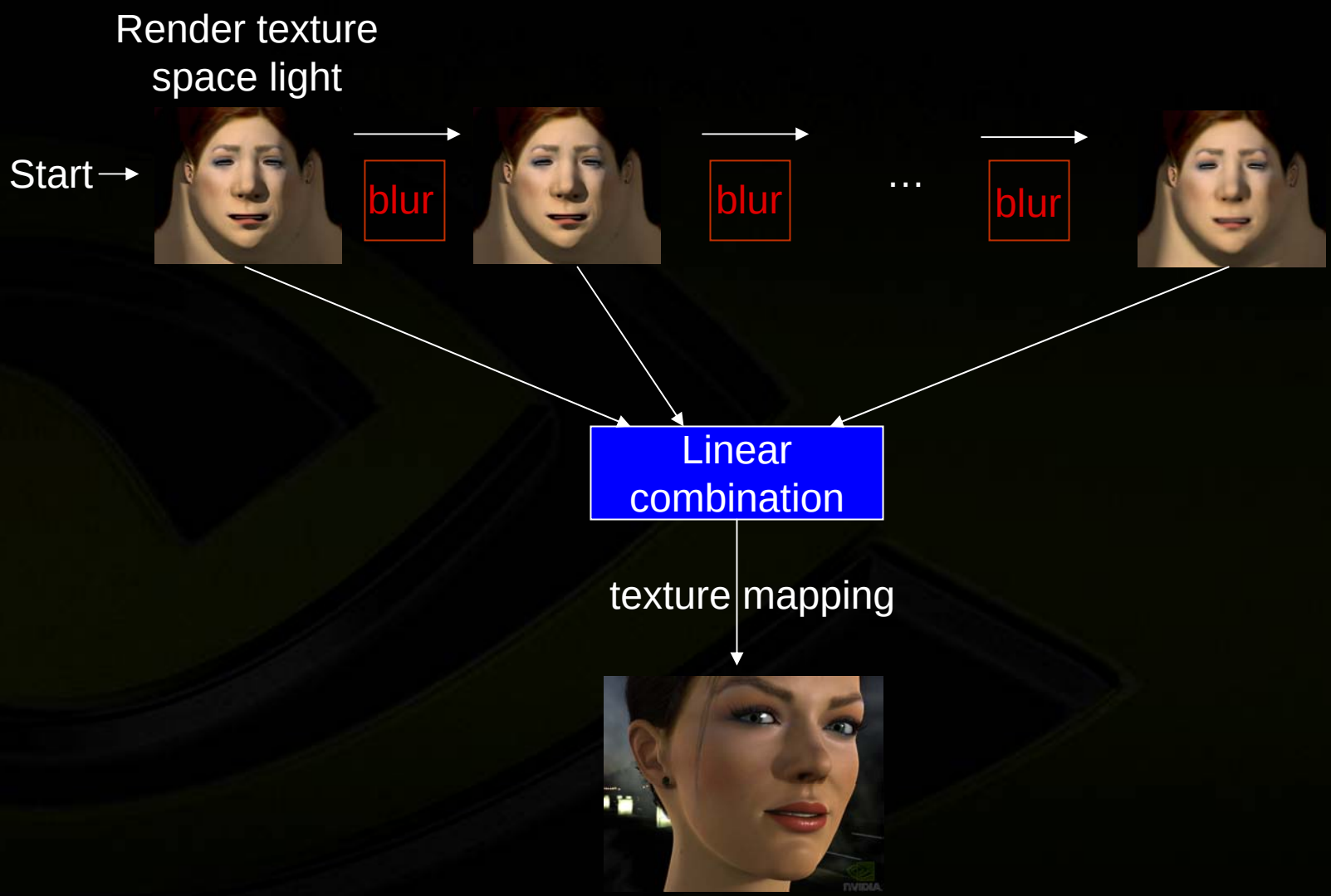
Rendering using texture coordinates

- The fragment shader computes subsurface irradiance (explained shortly)
- Store in an offscreen texture

Rendering using texture coordinates

- Now it can be efficiently blurred

Overview Diagram



Final pass: combine blurs + specular

Each blur operation is 2 passes

Blur weights

.006	.061	.242	.383	.242	.061	.006

Recurse several times building a set of different blurs

Horizontal blur

Temporary buffer

Vertical blur

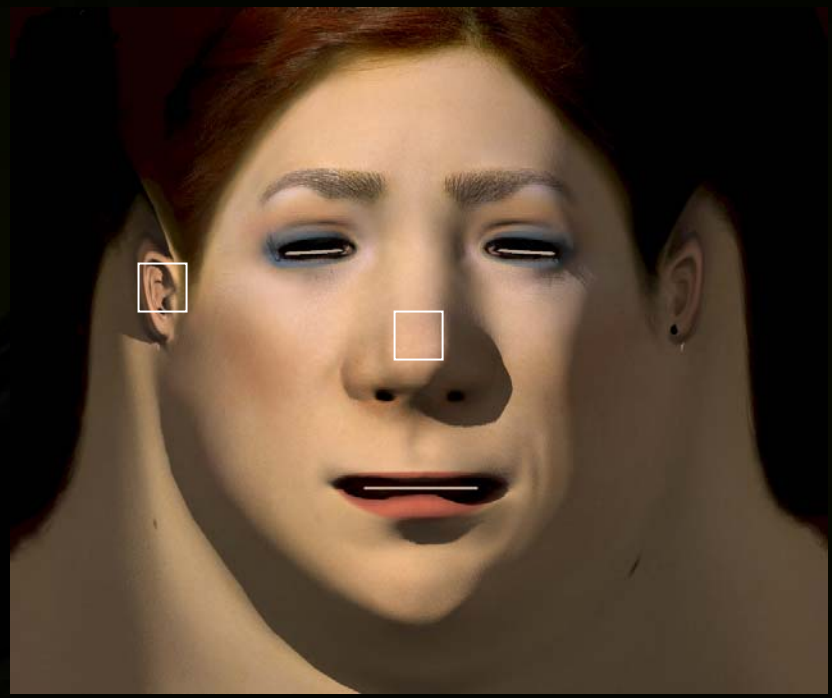
Texture used in final pass

Distortion problems

- Avoid uniform blur in texture space
- One pixel in texture space != constant distance in real world

Uniform blur

Distortion corrected blur



Distortion problems

- This can make the ears look like candle wax
- Very hard dramatic shadows will have an ugly edge

Uniform blurs

Distortion correction

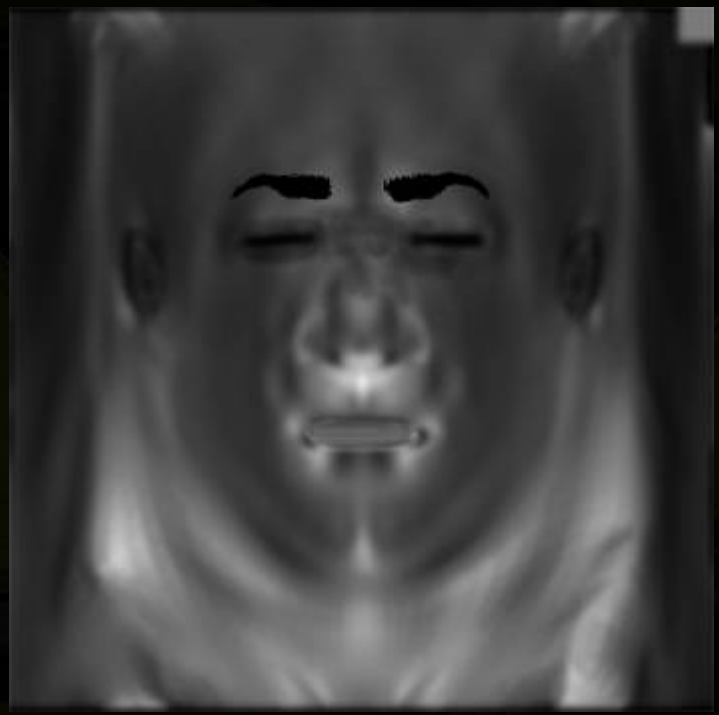
Accurate Distortion Correction

- We can easily estimate distortion
- Compute a map and inversely stretch our blurs

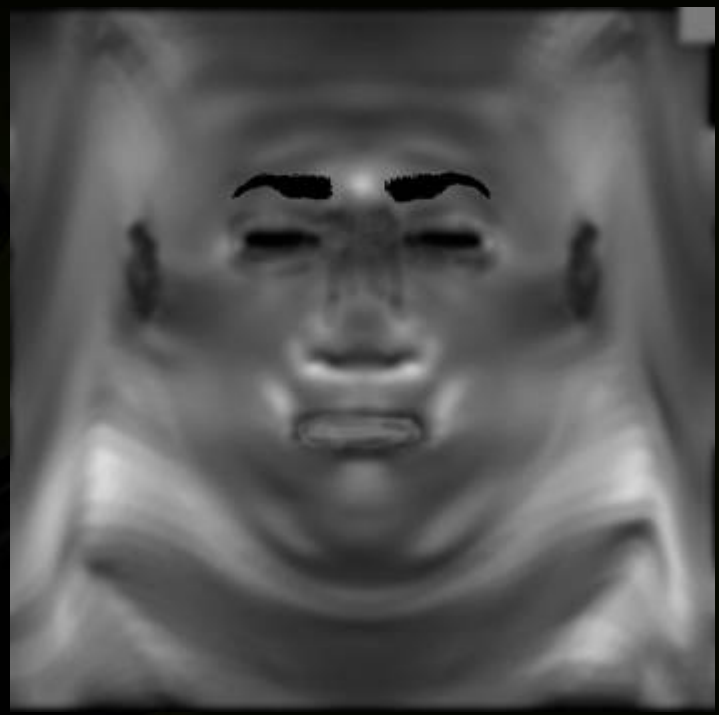
```
float3 derivu = ddx( v2f.worldCoord );
float3 derivv = ddy( v2f.worldCoord );

// 0.001 scales the values to map into [0,1]
// this depends on the model
float stretchU = 0.001 * 1.0 / length( derivu );
float stretchV = 0.001 * 1.0 / length( derivv );
```

Accurate Distortion Correction

- Use these values to scale the directional blurs
- Paint the eyebrows black to eliminate blur there



Blur radius in X



Blur Radius in Y

Accurate Distortion Correction

- In general these distortions are different in X and Y
- Separable blurs advantageous for 2 reasons
  - Can correct distortions in each direction separately
  - More efficient to compute than a 7 x 7 blur
- Build a hierarchy of blurred versions of irradiance
  - Good for mixing to get any diffusion profile you want
  - More efficient than computing a 100x100 blur (even separably)

Dealing with Seams

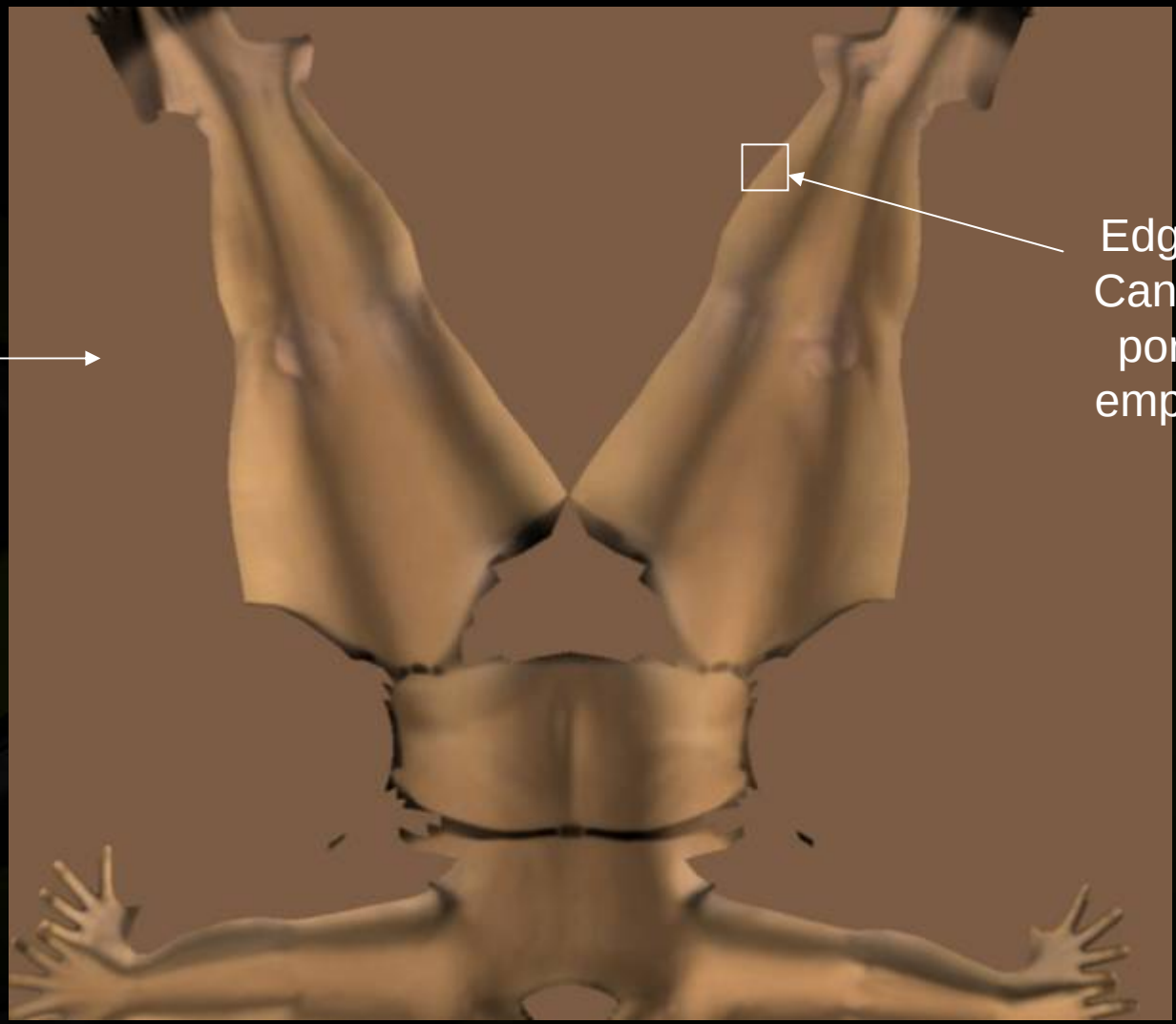
- Depending on the UV layout, there might be seams
- Black regions surround seam edges
  - Black regions will blur into nearby pixels, creating dark seams

Dealing with Seams

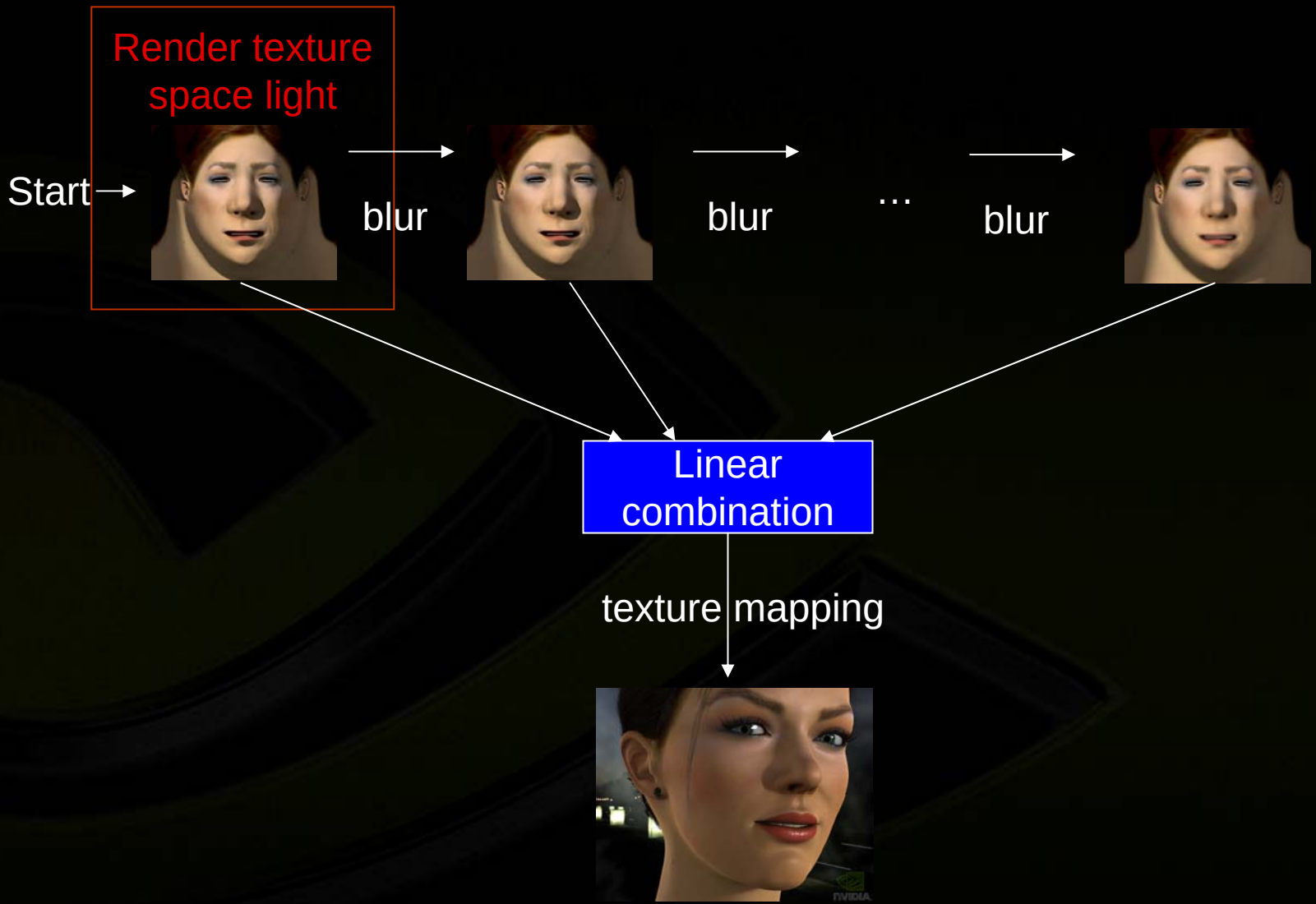
Dealing with Seams

- Ways around it:
  - Change Clear color
  - Edit distortion map
  - Object / no-object alpha map
  - Multiple UV sets (slow)

Dealing with Seams



Computing subsurface Irradiance



Final pass: combine blurs + specular

Computing subsurface Irradiance

- What color do we write here?
- How should diffuse color get used?

- First guess: `dot( N, L ) * diffuseCol`

- After
  - Blur, blur, blur
  - Linearly Combine blurs
  - Add specular

Computing subsurface Irradiance

- First guess: `dot( N, L ) * diffuseCol`

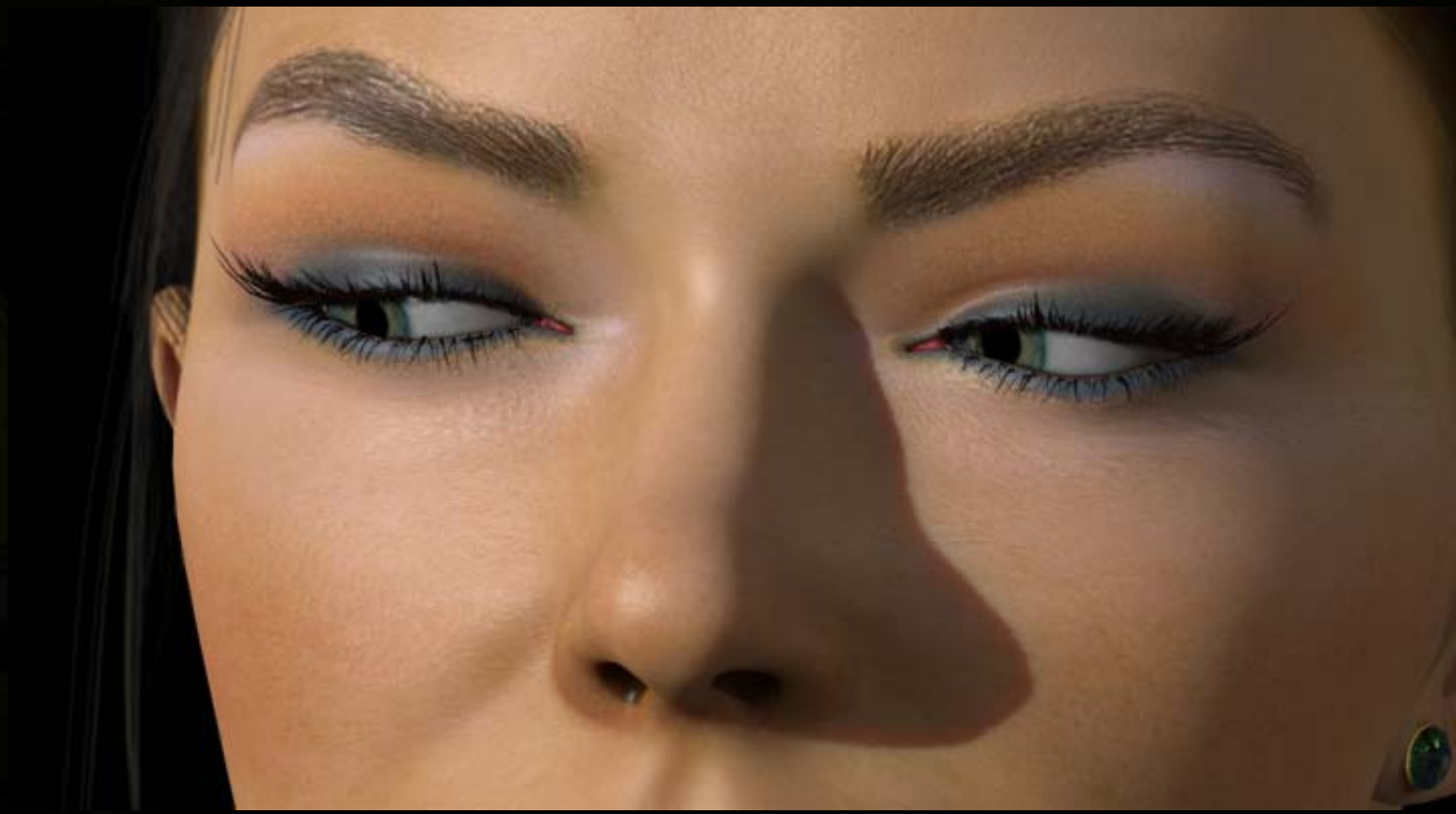
Computing subsurface Irradiance

- This can look ok
- Can lose high frequencies

- Second guess:
  - Store only N • L terms
  - Blur, blur, blur
  - Linearly Combine blurs
  - Multiply DiffuseCol
  - Add specular

Computing subsurface Irradiance

- Second guess: blur dot(N,L), mix, then multiply by diffuseCol



Computing subsurface Irradiance

- This can look ok
- Probably ok for photo-based maps
- No Color bleeding – Too much high frequency content left
- High frequency detail “sits” above underneath layers

- Third guess: can we somehow do both?

Computing subsurface Irradiance

- Want some coloring before and after blur
- Need a multiplication each time
- Can’t multiply DiffuseCol twice

Computing subsurface Irradiance

- Want some coloring before and after blur
- Need a multiplication each time
- Can’t multiply DiffuseCol twice

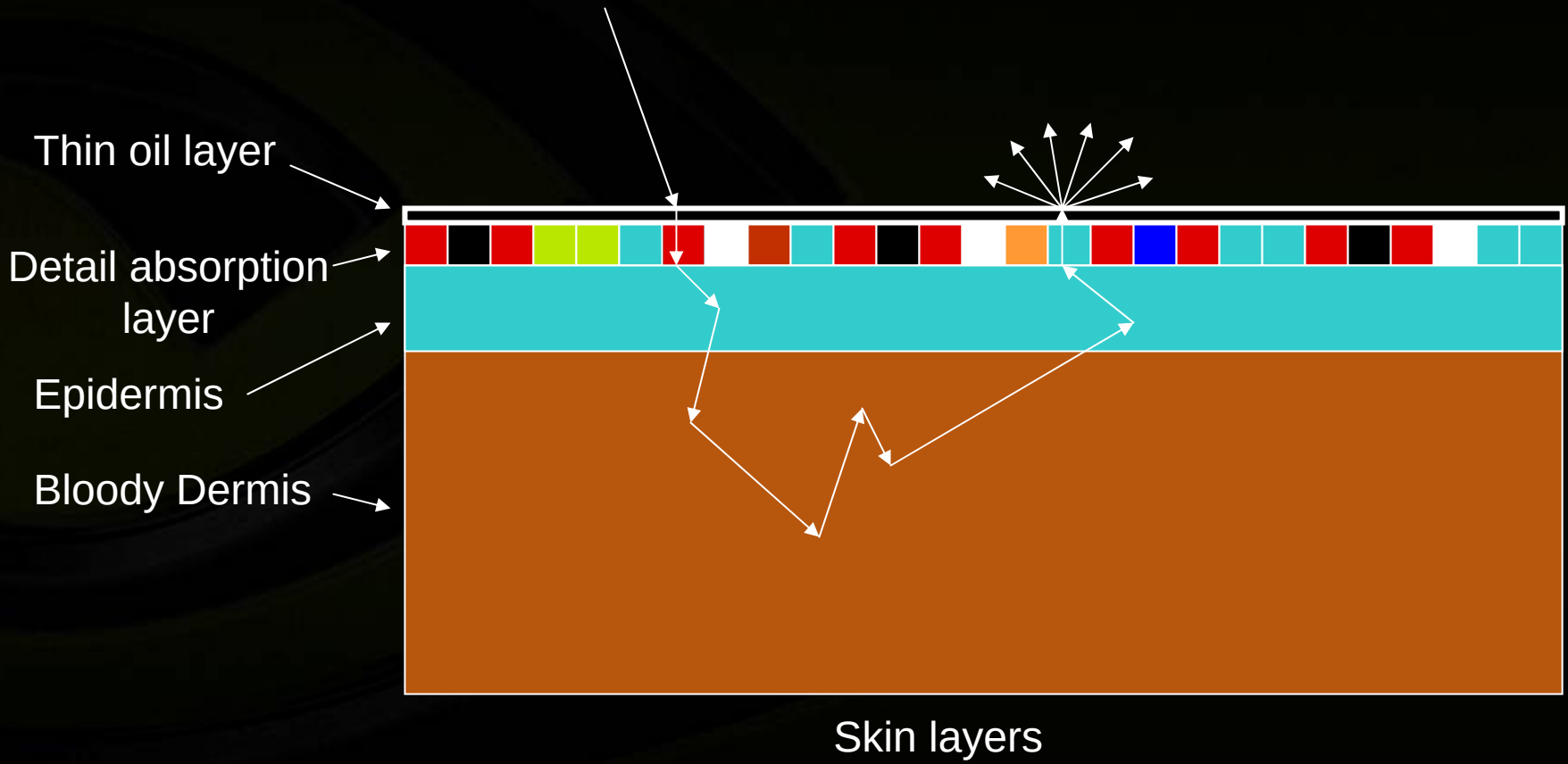
- Multiply sqrt(DiffuseCol) twice
  - Once before blurring
  - Once after blending

Computing subsurface Irradiance

- Half and half: apply sqrt(diffuseCol), blur, combine, then apply sqrt(diffuseCol)

Computing subsurface Irradiance

- **This simulates an infinitely thin absorption layer on the surface which doesn't scatter light**



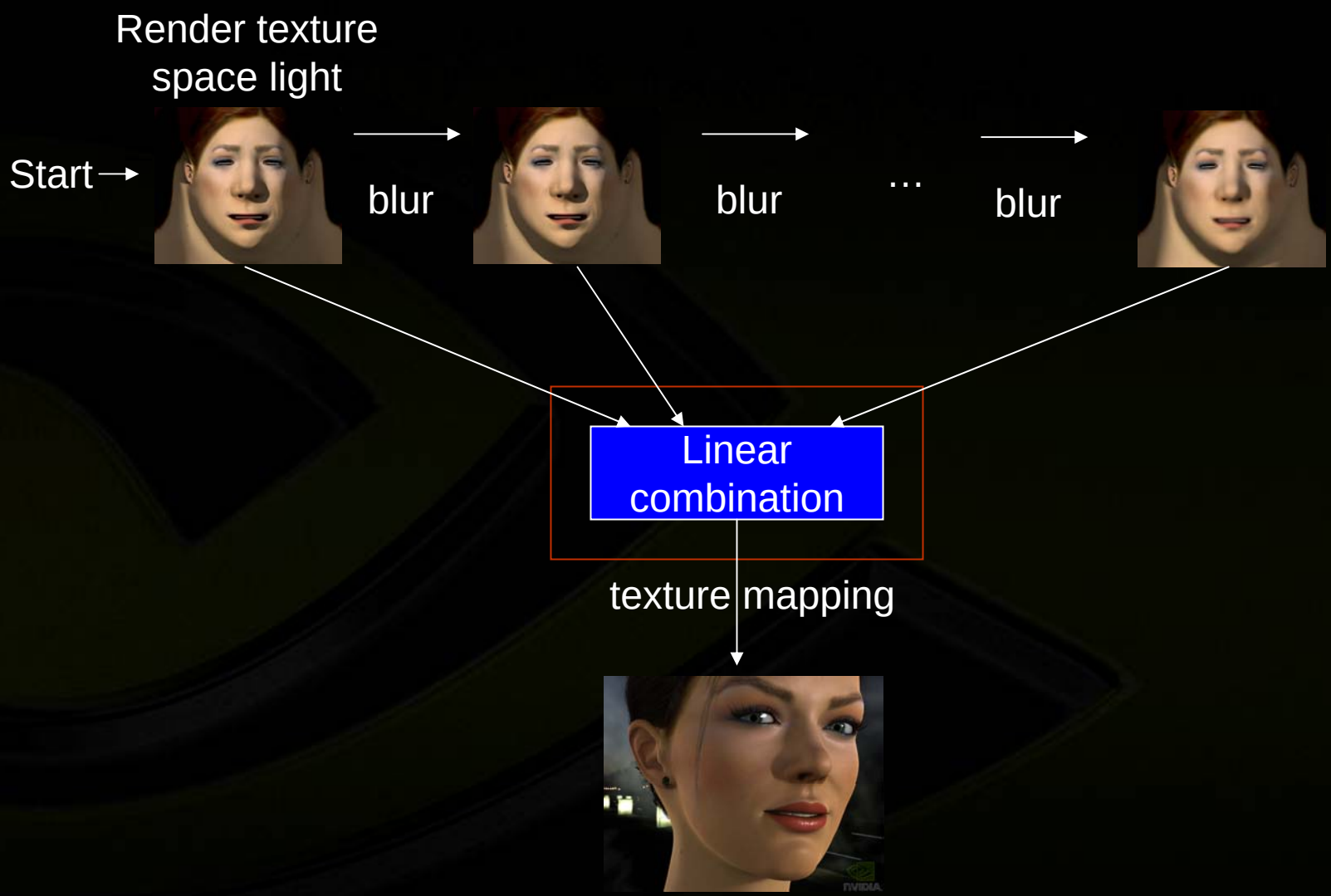
Computing subsurface Irradiance

- This is realistic in that
  - Light must interact with this detail layer twice
  - Once going in
  - Once coming out
- Assumptions
  - Variation in skin lies in a top thin layer

Computing subsurface Irradiance

- Artistic liberty
  - Generalize
  - Instead of
    - * sqrt(diffuseCol) before
    - * sqrt(diffuseCol) after
- Use
  - * pow(diffuseCol, mix) before
  - * pow(diffuseCol, 1.0 – mix) after
- In Adrianne we used mix = 0.82

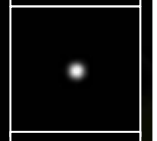
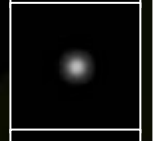
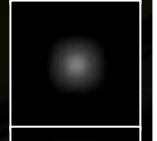
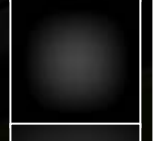
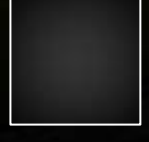
Combining blurs



Final pass: combine blurs + specular

Combining blurs

- For human skin we recommend

	Blur Width(mm)	Blur Weights red	green	blue
	0.042 *	0.22	0.437	0.635
	0.220	0.101	0.355	0.365
	0.433	0.119	0.208	0
	0.753	0.114	0	0
	1.412	0.364	0	0
	2.722	0.080	0	0

*Smallest blur level is lighting re-computed directly in the final render pass (it has not been blurred)

Combining blurs – Choosing weights

- Weights should sum to 1.0
- Somewhat artistic process
- Do it one color at a time
  - return finalCol.xxx
  - return finalCol.yyy
  - Etc
  - Compare to reference photos (red, then green, then blue)
- Avoid:
  - Heavy blur + specular
  - Single gaussian + specular

Combining blurs

- For human skin:
  - Broadest scatter in red
  - Small scatter in green
  - Blue has very little at all
- Single Gaussian is bad
  - Even for one-layered materials

Combining blurs

```
float3 diffuseLight = nonBlur * E1 * pow( diffuseCol, 0.5 );

float3 blur2tap = f3tex2D( blur2Tex, v2f.c_texCoord.xy );
float3 blur4tap = f3tex2D( blur4Tex, v2f.c_texCoord.xy );
float3 blur8tap = f3tex2D( blur8Tex, v2f.c_texCoord.xy );
float3 blur16tap = f3tex2D( blur16Tex, v2f.c_texCoord.xy );
float3 blur32tap = f3tex2D( blur32Tex, v2f.c_texCoord.xy );

diffuseLight += blur2 * blur2tap.xyz;
diffuseLight += blur4 * blur4tap.xyz;
diffuseLight += blur8 * blur8tap.xyz;
diffuseLight += blur16 * blur16tap.xyz;
diffuseLight += blur32 * blur32tap.xyz;

// renormalize weights so they sum to 1.0
float3 norm2 = nonBlur + blur2 + blur4 + blur8 + blur16 + blur32;
diffuseLight /= norm2;
diffuseLight *= pow( diffuseCol, 0.5 );
```

Combining blurs

- Highly scalable process
- Use as many as you need
- Try at least two blurs

Combining blurs

6 levels of blur
physically based specular

3 levels of blur
Blinn-Phong specular

Energy Conservation

- Problems with using only N • L?
  - We didn't conserve energy
  - Ignores directional effects of rough, oily surface
- For skin the difference is quite subtle

dot(N,L)

Careful conservation of energy

Extras

- Gamma correction
- Try several speculars
- Many tileable bump maps

Gamma correction

- Your monitor is lying to you!
- displayed brightness = pixelValue^2.2
- All our displays do this
- Digital Cameras know this
  - They correct for it
  - We should too

Gamma correction comparison

No Gamma correction

Gamma correction

Gamma correction

- Affects image quality greatly
- Screws up lighting and shading
- We need two fixes:

- 1) Correct textures
- 2) Correct framebuffer pixels

Gamma correction – Correct textures

- Pictures & painted maps will have non-linear pixels
- To convert them to linear:

```
diffuseCol = pow( f3tex2d( diffTex, v2f.tex ), 2.2 );

//Or (cheaper)
diffuseCol = f3tex2d( diffTex, v2f.tex );
diffuseCol = diffuseCol * diffuseCol;
```

Gamma correction – Correct Images

- Do this once or automate it upon export/build
  - (avoids the pow() instructions)
- Use sRGB texture format
  - DirectX
    - DXGI_FORMAT_R8G8B8A8_UNORM_SRGB
    - DXGI_FORMAT_BC{1,2,3}_UNORM_SRGB
  - OpenGL
    - GL_EXT_texture_sRGB
- Avoid correcting
  - ambient occlusion
  - Normal maps
  - Alpha channels

Gamma correct the final color value

- Our displays warp pixel values
- When we write to the framebuffer:
  - Invert the warping

```
float3 finalCol = do_all_lighting_and_shading();
float pixelAlpha = compute_pixel_alpha();

return float4(pow(finalCol, 1.0 / 2.2), pixelAlpha);

// or (cheaper)
return float4( sqrt( finalCol ), pixelAlpha );
```

Gamma correction: sRGB

- sRGB framebuffers
  - You write linear pixels
  - Hardware does correction for you
  - Blending is done linearly
  - EXT_framebuffer_sRGB

Try more than one specular at once

- One specular roughness might not be enough
- Skin is a complex material
- Try mixing a few specular calculations together
  - Linear combination
- Many of the intermediate calculations can be re-used
- In Adrianne we used four

Try more than one specular at once

- `float4 specWeights = float4( 0.1, 0.2, 0.3, 0.5 );`
- `float4 specRough = float4( 0.05, 0.1, 0.2, 0.3 );`

- `float spec = QuadSpec( N, V, L, specRough, specWeights );`

- `specLight += spec * lightcolor * shadow;`

Try more than one specular at once




Linear combination
of 4 speculars

Tileable detail bumps

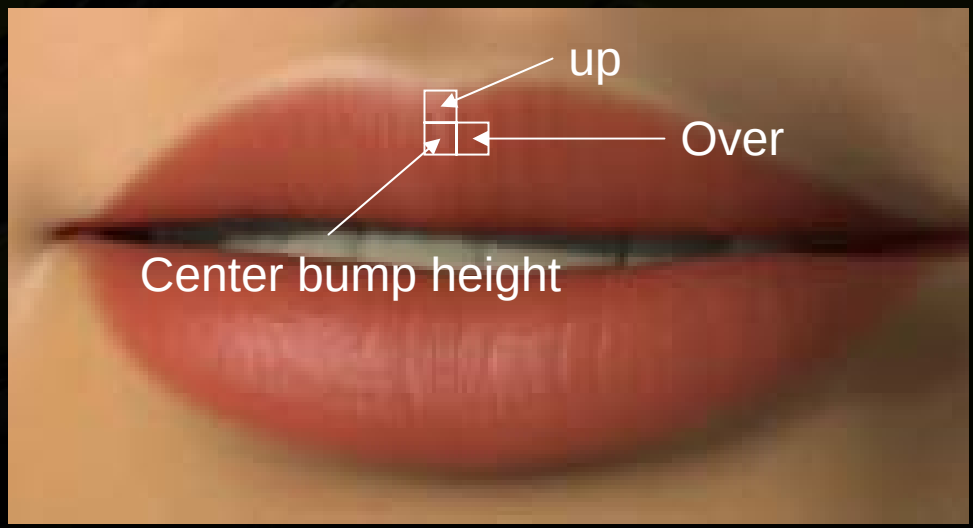
- Bump detail on skin is important
- Creating one large high resolution normal map requires a lot of memory
- Try creating several small tileable bump maps
- Don’t average normals
- Add several bump height values and compute a normal
- Compute normal in shader using bump heights

Don’t average normals

Normals don’t add or average when two bump maps are combined

Tileable detail bumps

- Fragment shader:
  - Find tex coords for pixel neighbors (use ddx, ddy)
  - Compute total bump height for 3 locations
  - Compute tangent space normal

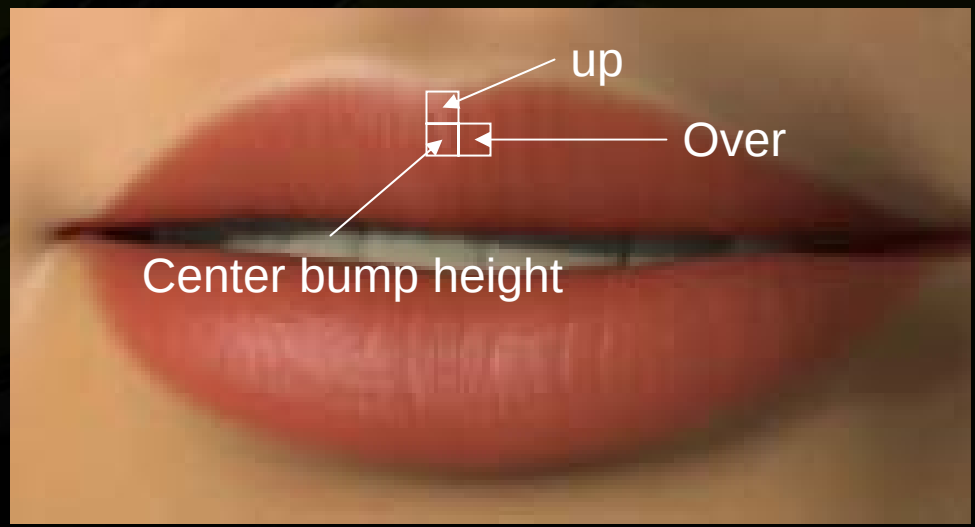


Tileable detail bumps

- Estimate tex_coord for one pixel to the right, and one pixel up

```
float2 uv_offset_over = ddx( v2f.c_texCoord );
float2 uv_offset_up = ddy( v2f.c_texCoord );
float2 texCoordOver = v2f.c_texCoord + uv_offset_over;
float2 texCoordUp = v2f.c_texCoord + uv_offset_up;
```

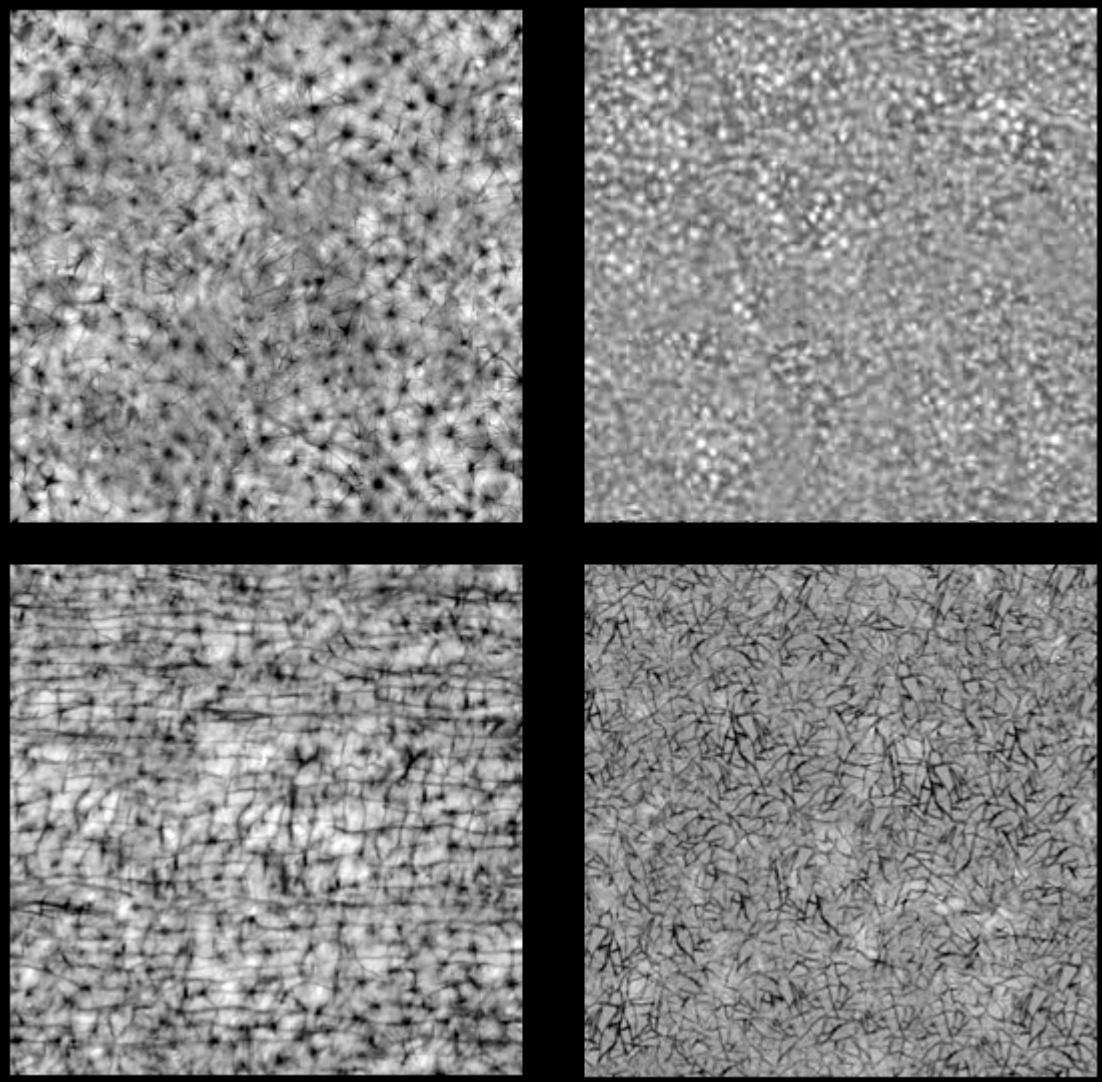
- Use texCoordOver and texCoordUp to evaluate all bump maps



Tileable detail bumps

```
float2 uv_offset_over = ddx( v2f.c_texCoord );
float2 uv_offset_up = ddy( v2f.c_texCoord );
float2 texCoordOver = v2f.c_texCoord + uv_offset_over;
float2 texCoordUp = v2f.c_texCoord + uv_offset_up;
…
// compute finalHeightCenter, finalHeightOver, finalHeight up
…
float yscalefactor = length(uv_offset_over) /
length(uv_offset_up);

float3 tanNormal = normalize(float3(finalHeightCenter -
finalHeightOver, yscalefactor * (finalHeightCenter -
finalHeightUp), length(uv_offset_over) * 4048.0));
```

Tileable detail bumps



Four bump tiles used on Adrianne

Summary

- Improved texture space diffusion
  - Separable blurs, separable distortion correction
  - Build one blur from the last
  - Blend several blurs for more realistic diffusion
  - Blend blurs differently in red, green, and blue
- Gamma Correct!!!
- Layered bump tiles
  - Don’t layer normal maps
  - Layer bump heights and compute an accurate normal
- Many speculars

Play around to get new effects

- Any kind of fleshy surface requires subtle scattering

*Texture maps courtesy of XYZRGB

Future work

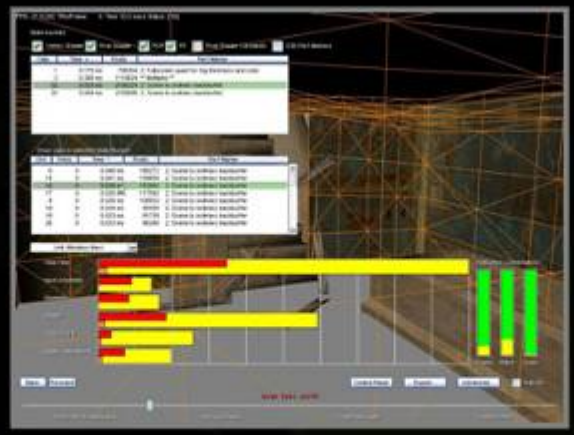
- Skin shading research hasn’t stopped
- Watch for upcoming demos with even more realistic skin shading
- Watch for *GPU Gems 3*

References

- BORSHUKOV G., AND LEWIS, J. 2003. Realistic human face rendering for “the matrix reloaded”. In *ACM SIGGRAPH 2003 Conference Abstracts and Applications (Technical Sketch).*
- DEBEVEC, P., HAWKINS, T., TCHOU, C., DUIKER, H.-P., SAROKIN, W., AND SAGAR, M. 2000. Acquiring the reflectance field of a human face. *In Computer Graphics, SIGGRAPH 2000 Proceedings*, 145–156.
- DONNER C., AND JENSEN, H.W. 2005. Light diffusion in multi-layered translucent materials. *In Proceedings of SIGGRAPH 2005*, 1032-1039.
- GREEN, S. 2004. Real-time approximations to subsurface scattering. In *GPU Gems*, R. Fernando, Ed. Addison Wesley, Mar., ch. 16, 263–278.
- JENSEN, H. W., AND BUHLER, J. 2002. A rapid hierarchical rendering technique for translucent materials. *ACM Trans. Graph. 21*, 3, 576–581.
- JENSEN, H. W., MARSCHNER, S. R., LEVOY, M., AND HANRAHAN, P. 2001. A practical model for subsurface light transport. *In Proceedings of SIGGRAPH 2001*, 511–518.
- KRISHNASWAMY, A., AND BARONOSKI, G. V. G. 2004. A biophysically-based spectral model of light interaction with human skin. *In Proceedings of EUROGRAPHICS 2004*, vol. 23.
- PHARR, M., AND HUMPHREYS, G. 2004. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc.
- WEYRICH, T., MATUSIK, W., PFISTER, H., BICKEL, B., DONNER, C., TU, C., MCANDLESS, J., LEE, J., NGAN, A., JENSEN, H. W., AND GROSS, M. 2006. Analysis of human faces using a measurement-based skin reflectance model. *ACM Transactions on Graphics 25*, 3 (July), 1013–1024.

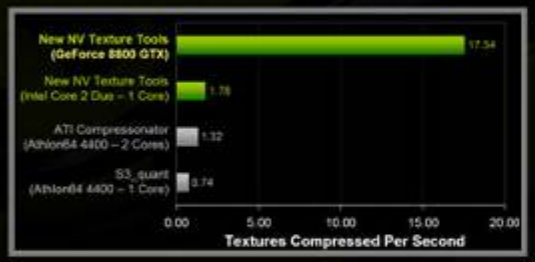
New Developer Tools at GDC 02007

SDK 10



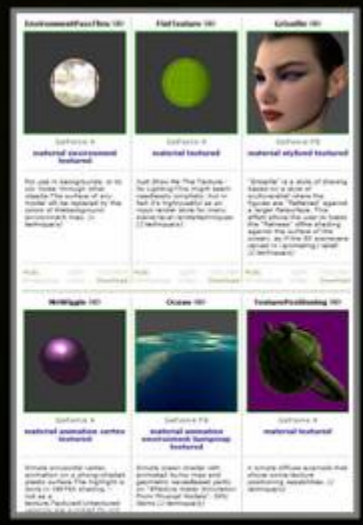
PerfKit 5

FX Composer 2



GPU-Accelerated Texture Tools

ShaderPerf 2



Shader Library

Questions?