SIGGRAPH2004

Real-time skin rendering on graphics hardware

Pedro V. Sander
David Gosselin
Jason L. Mitchell

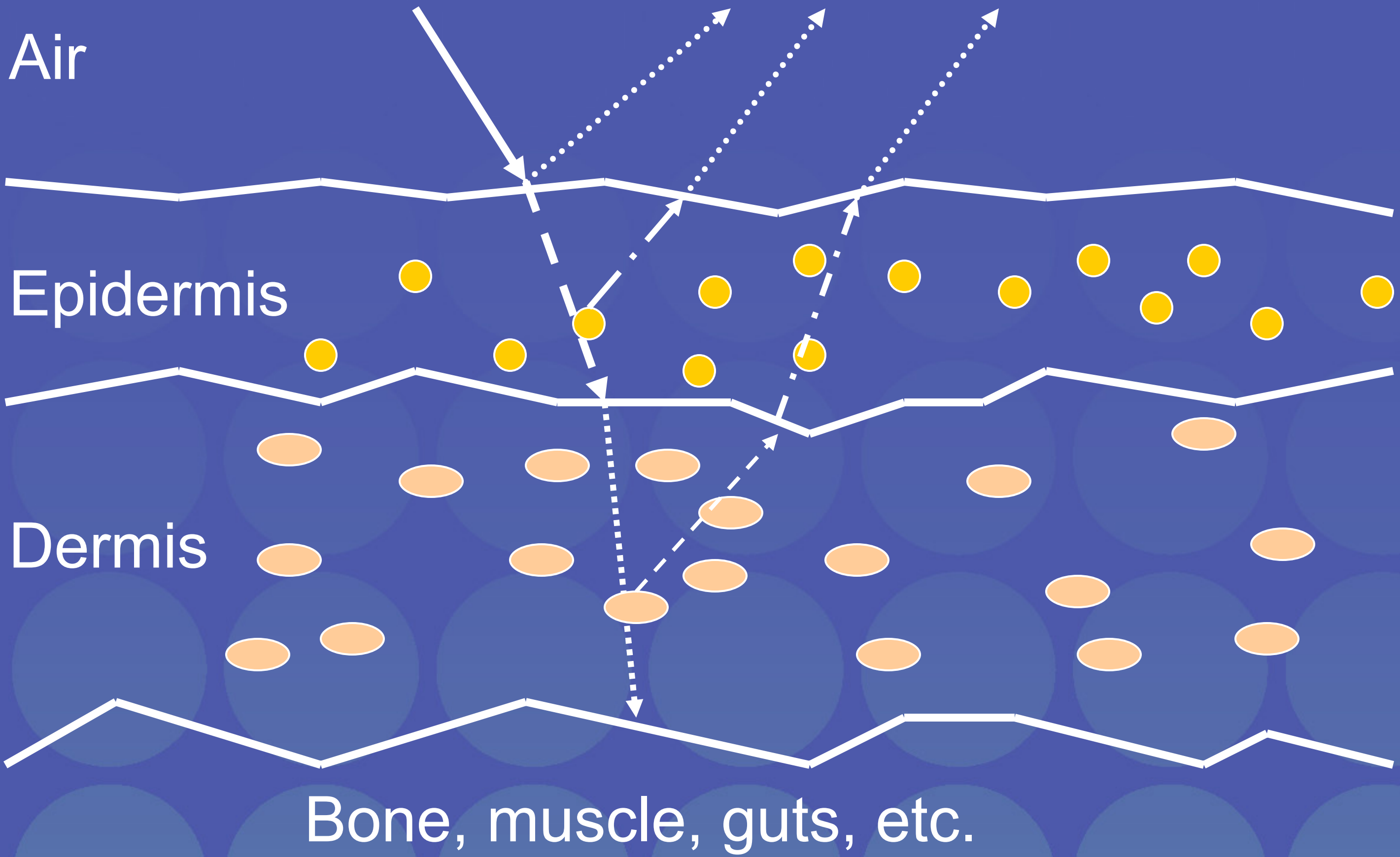
ATI Research

Skin shading

SIGGRAPH2004

- Most lighting comes from sub-surface scattering
- Traditional Lambertian lighting model is designed for hard surfaces with no sub-surface scattering so it doesn’t work well for skin

Rough cross section



Our objective

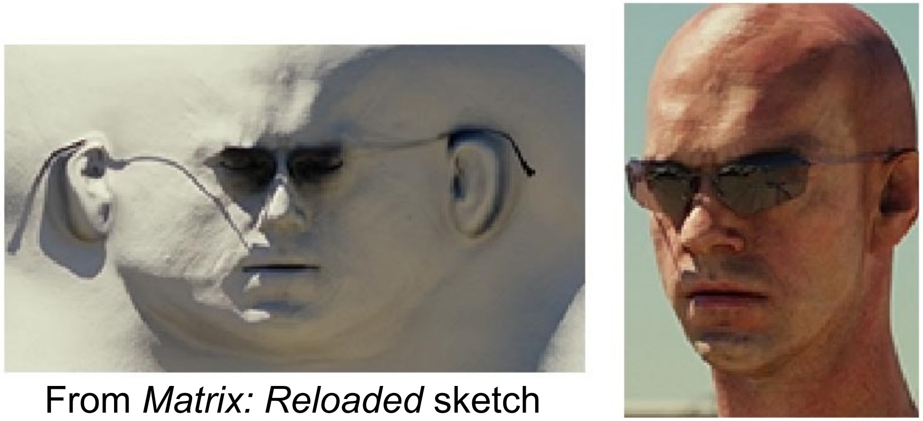
- Develop a simple, efficient skin rendering algorithm for a real-time demo using ATI hardware
- Must be very fast, as it is one of several rendering techniques used concurrently in the demo
- Results that approximate sub-surface scattering

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 a subsequent final pass
- We used a bump map for the specular lighting pass only

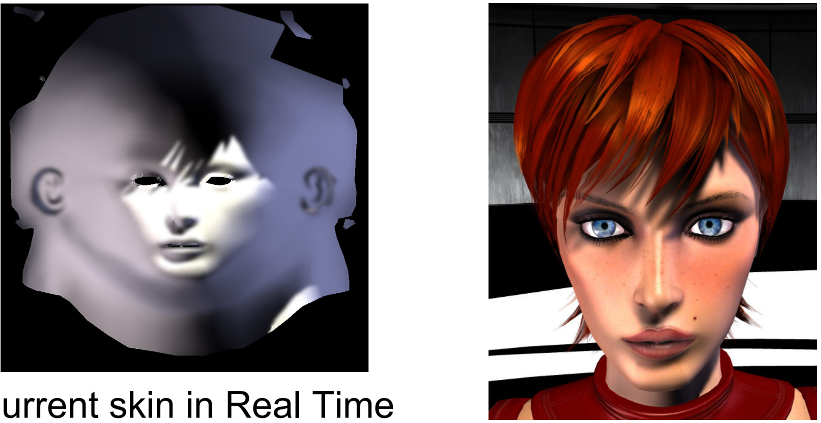
Previous work

- From Realistic Human Face Rendering for "The Matrix Reloaded" @ SIGGRAPH 2003:



From *Matrix: Reloaded* sketch

- Our results:



Current skin in Real Time

Previous work

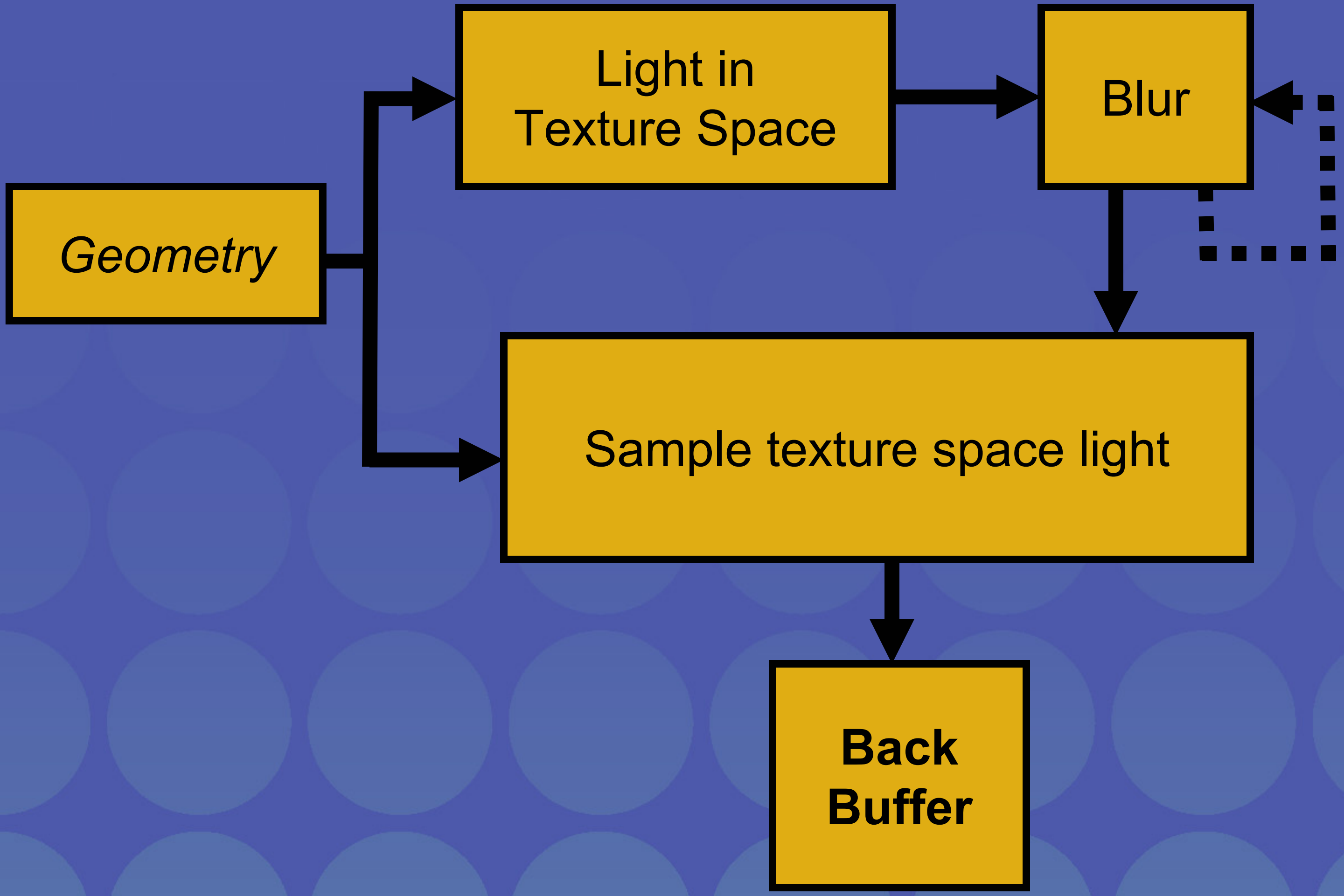
- [Borshukov03]
  - Texture lighting-based method almost everywhere
  - Used traditional ray tracing for areas where light can pass all the way through (e.g. ears)
    - We cannot afford to do that in real-time
- [Mertens03]
  - Similar image-space real-time technique using importance sampling of BSSRDF

Outline

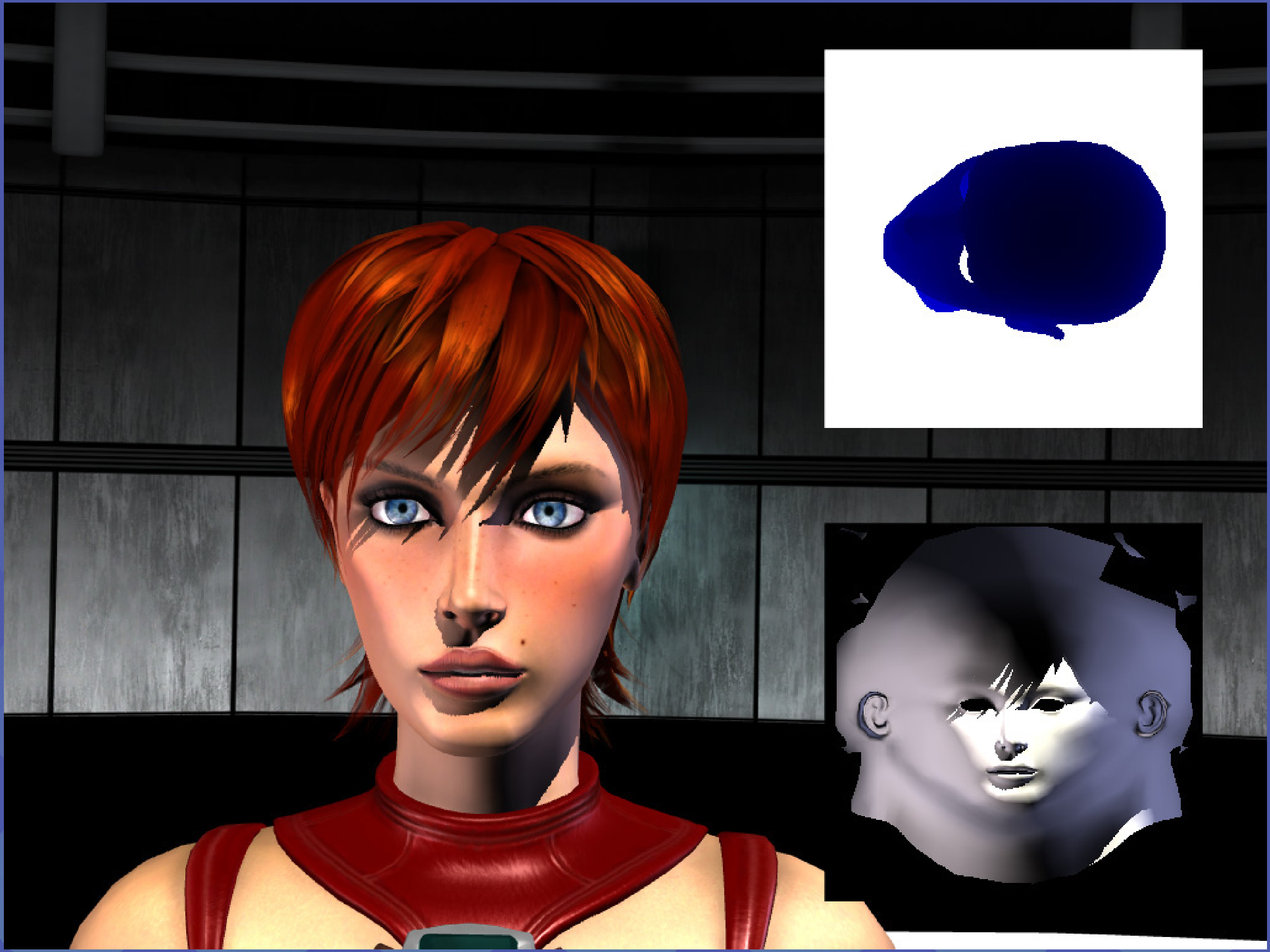
- Algorithm overview
- Texture space lighting
  - Blur operation
  - Dilation
- Shadows
  - Soft shadows
  - Translucent shadows
- Acceleration techniques
  - Early-Z culling

Basic approach

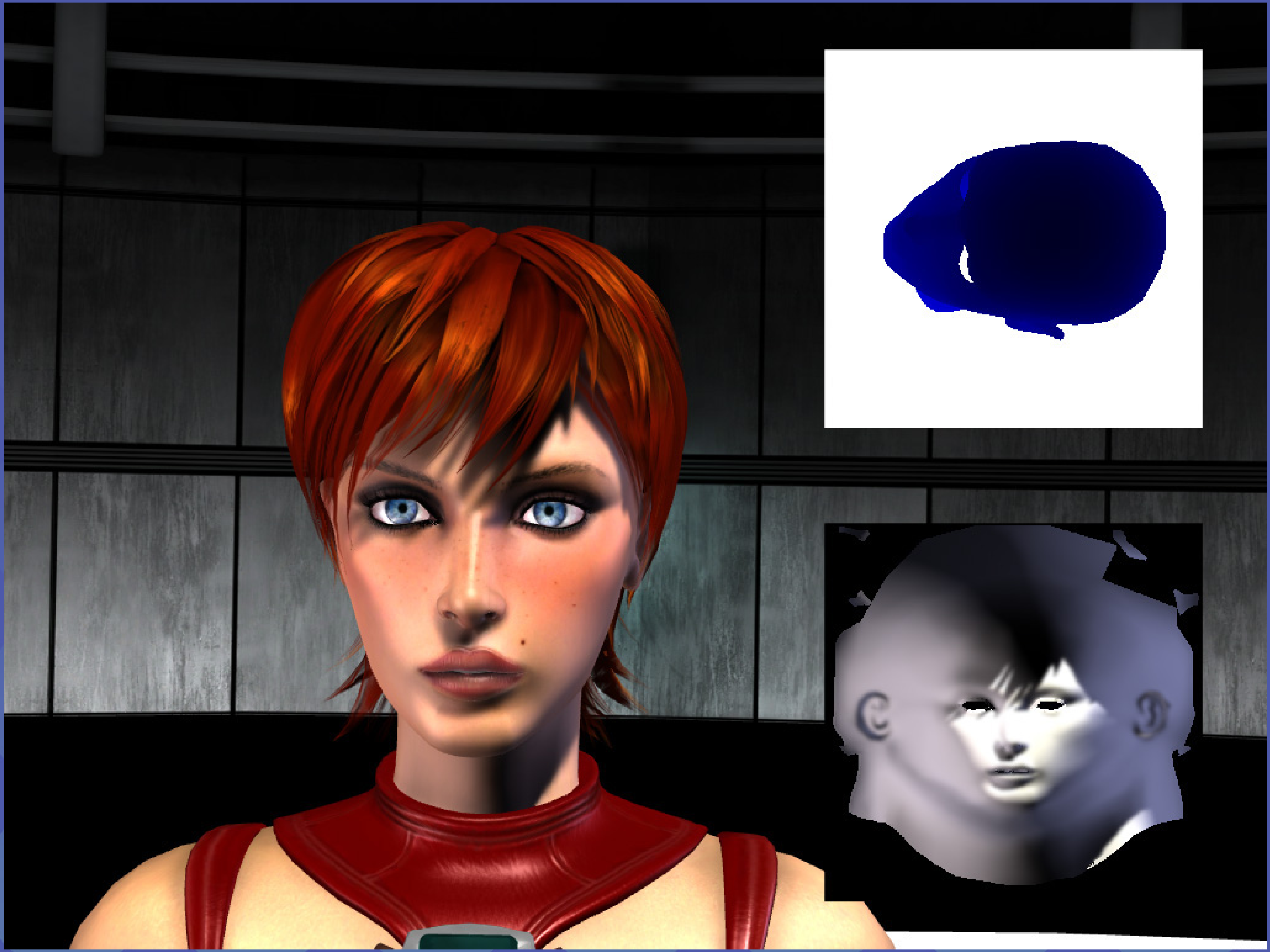
SIGGRAPH2004



Standard lighting model

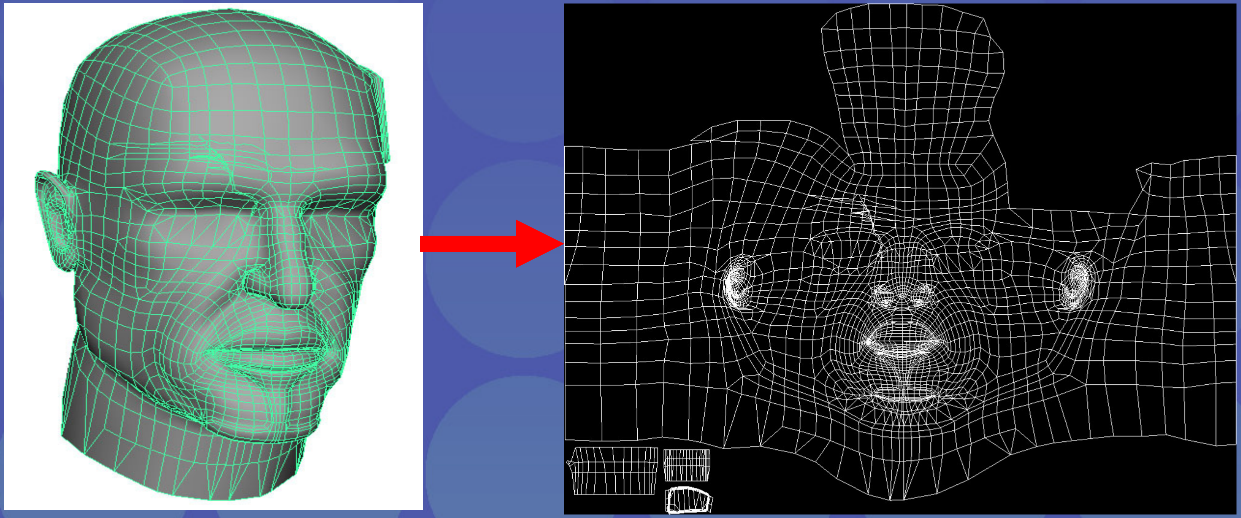


Blurred lighting model



Rendering to texture

- Light as a 3D model
- Draw into texture
  - Pass texture coordinates as “position”
  - Rasterizer does the unwrap

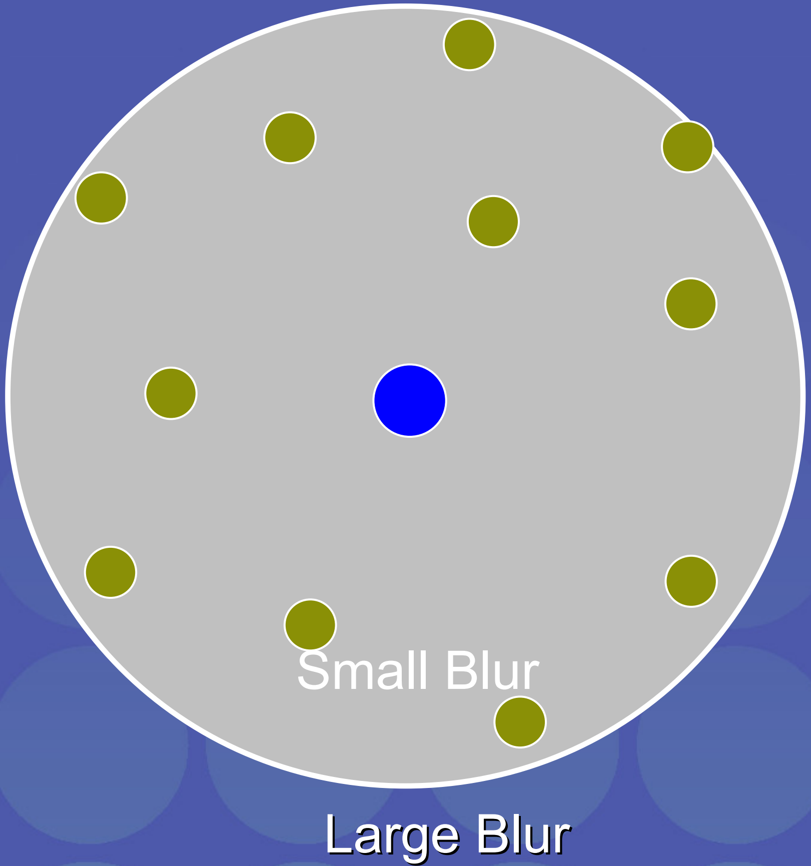


Spatially varying blur

- Used to simulate the subsurface component of skin lighting
- Used a growable Poisson disc filter
- Read the kernel size from a texture
- Allows varying the subsurface effect
  - Higher for places like ears/nose
  - Lower for places like cheeks

Filter kernel

- Stochastic sampling
- Poisson distribution
- Samples stored as 2D offsets from center

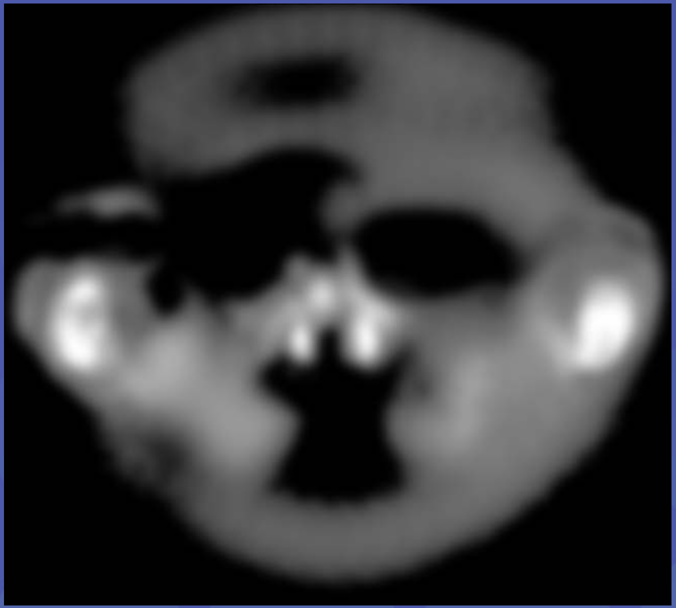


🔵 Center Sample

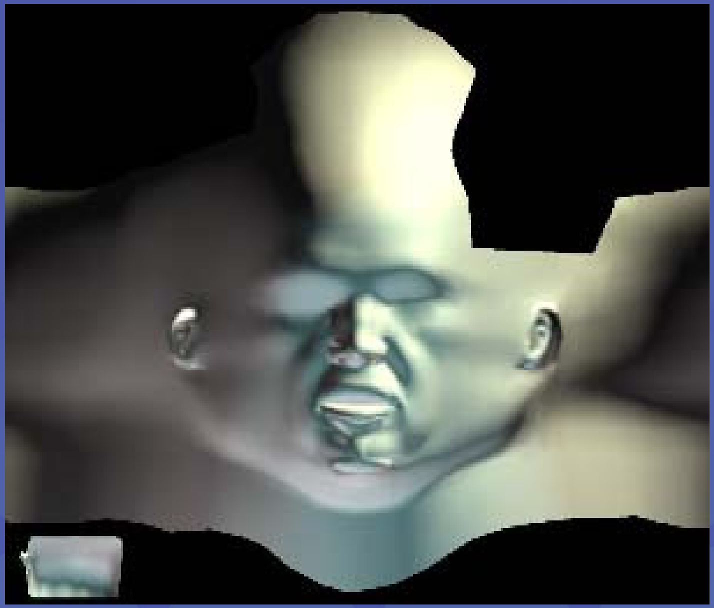
🟢 Outer Samples

Blur kernel size map and blurred lit texture

SIGGRAPH2004



Blur Kernel Size Map

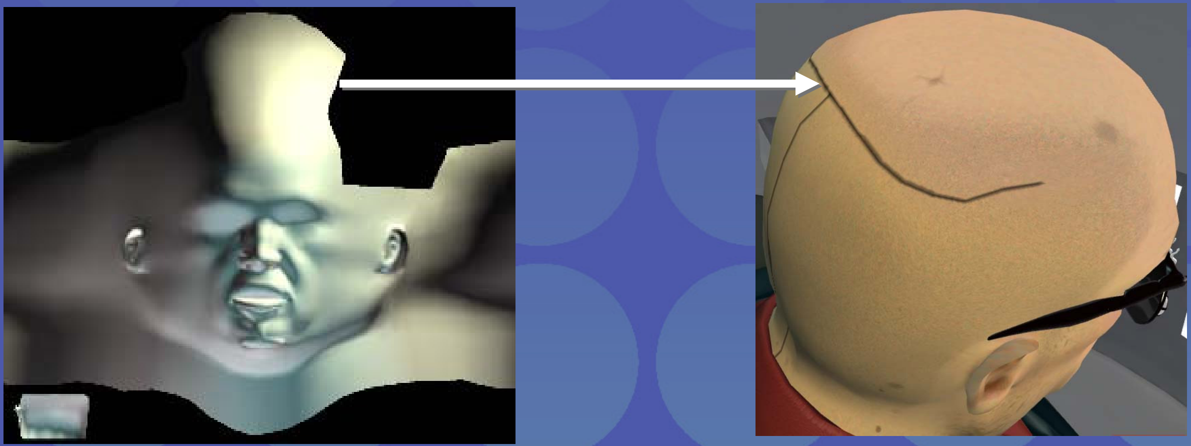


Texture Space Lighting

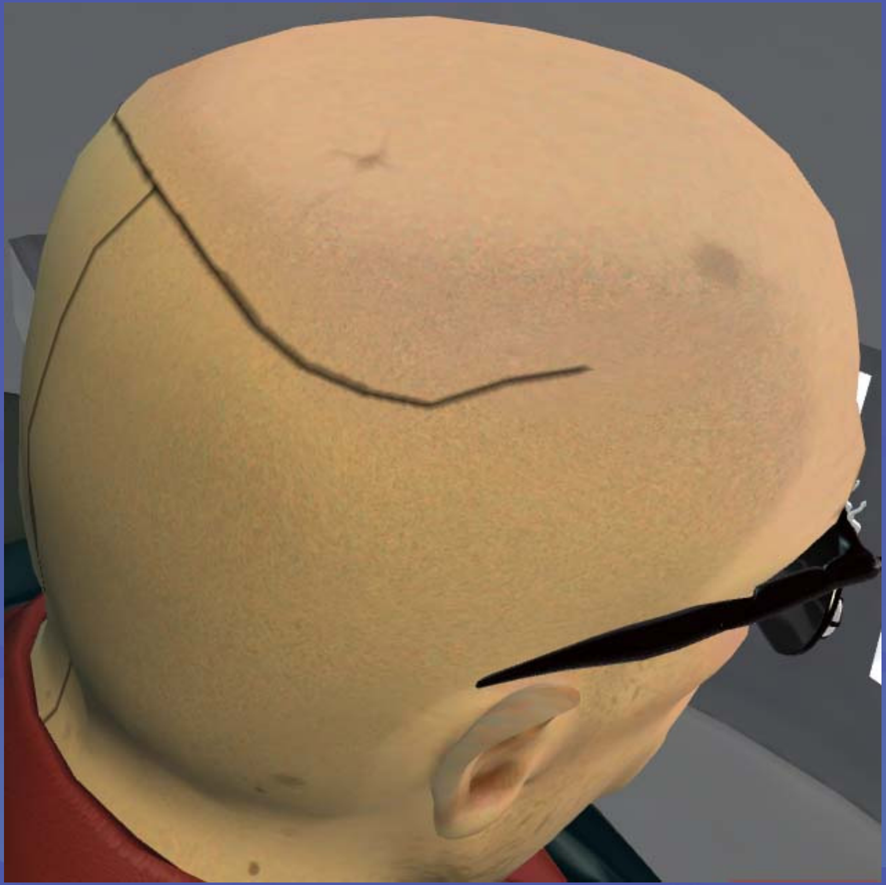
Result

Dilation

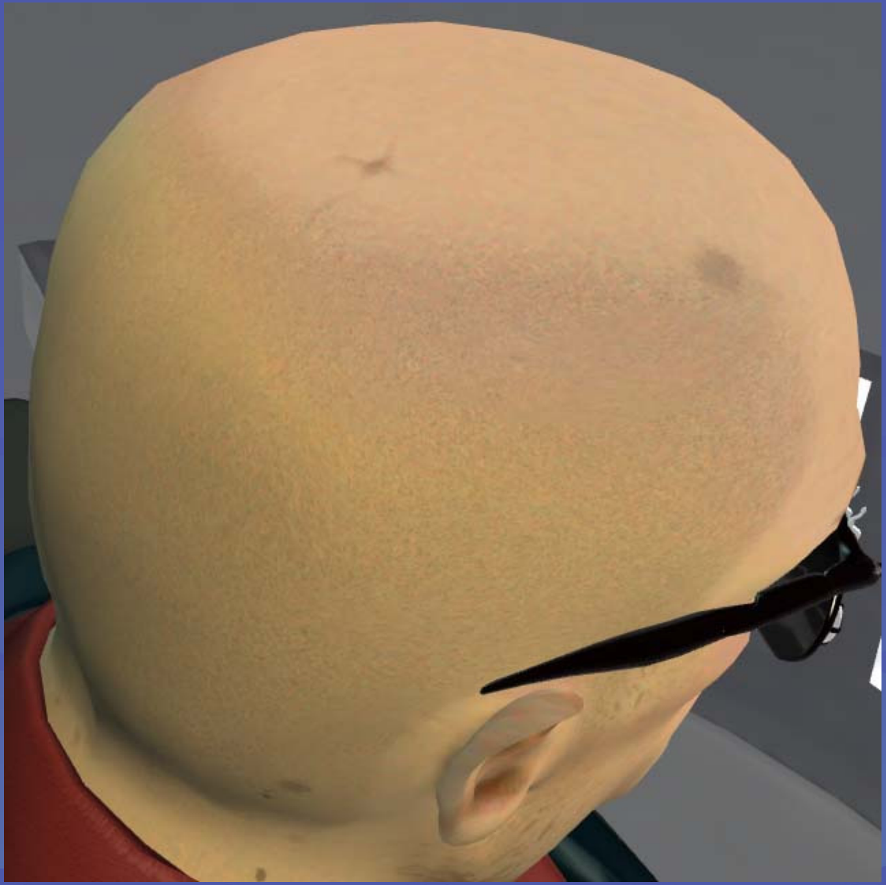
- Texture seams can be a problem (unused texels, bilinear blending artifacts)
- During the blur pass we need to dilate
- Use alpha channel of off-screen texture
- If any sample has 1.0 alpha, just copy the sample with the lowest alpha



Dilation Results



Without Dilation

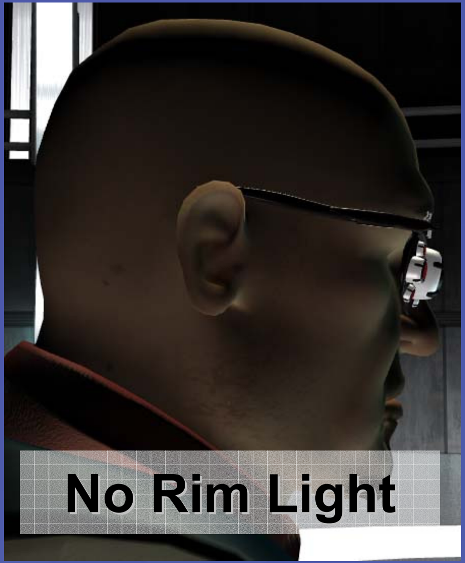


With Dilation

Rim light

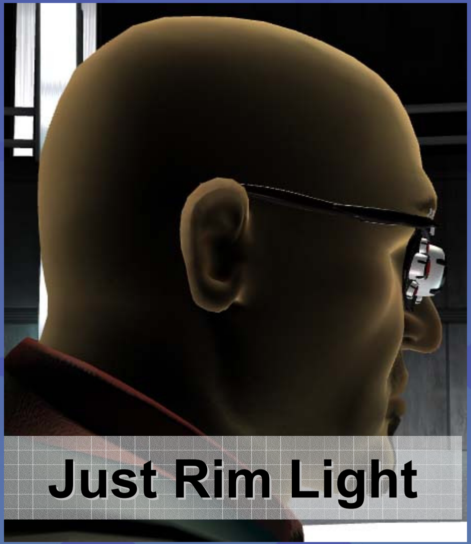
- We wanted to further emphasize the light that bleeds through the skin when backlit
- Compute the dot product between the negative light vector and the view vector
- Multiply result by Fresnel term
- Only shows up if there is a light roughly “behind” the object

Added rim light result



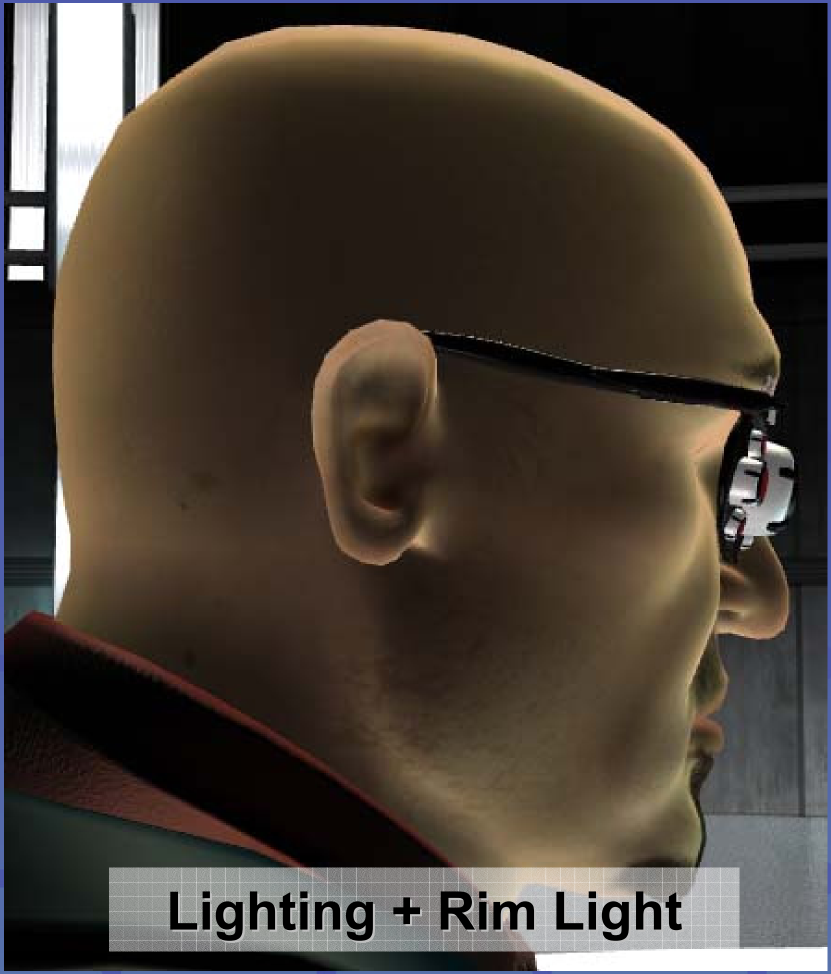
No Rim Light

+



Just Rim Light

=



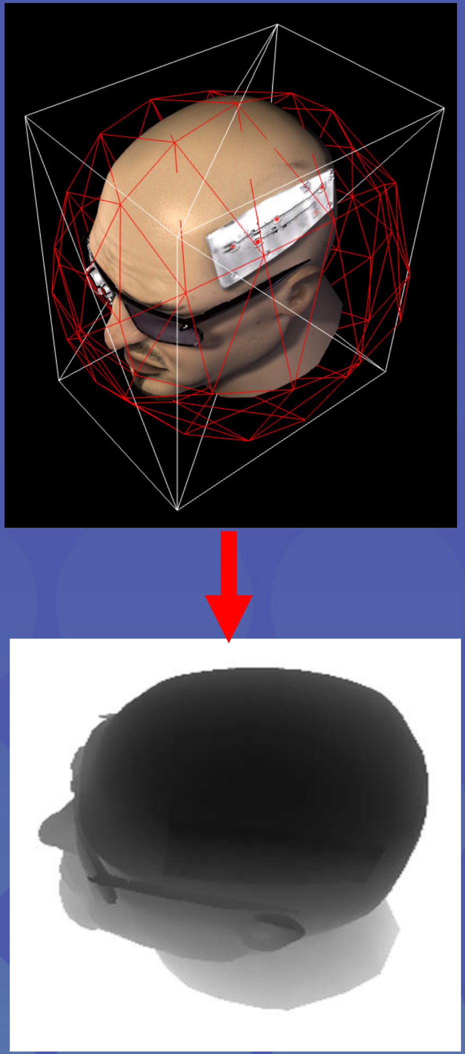
Lighting + Rim Light

Shadows

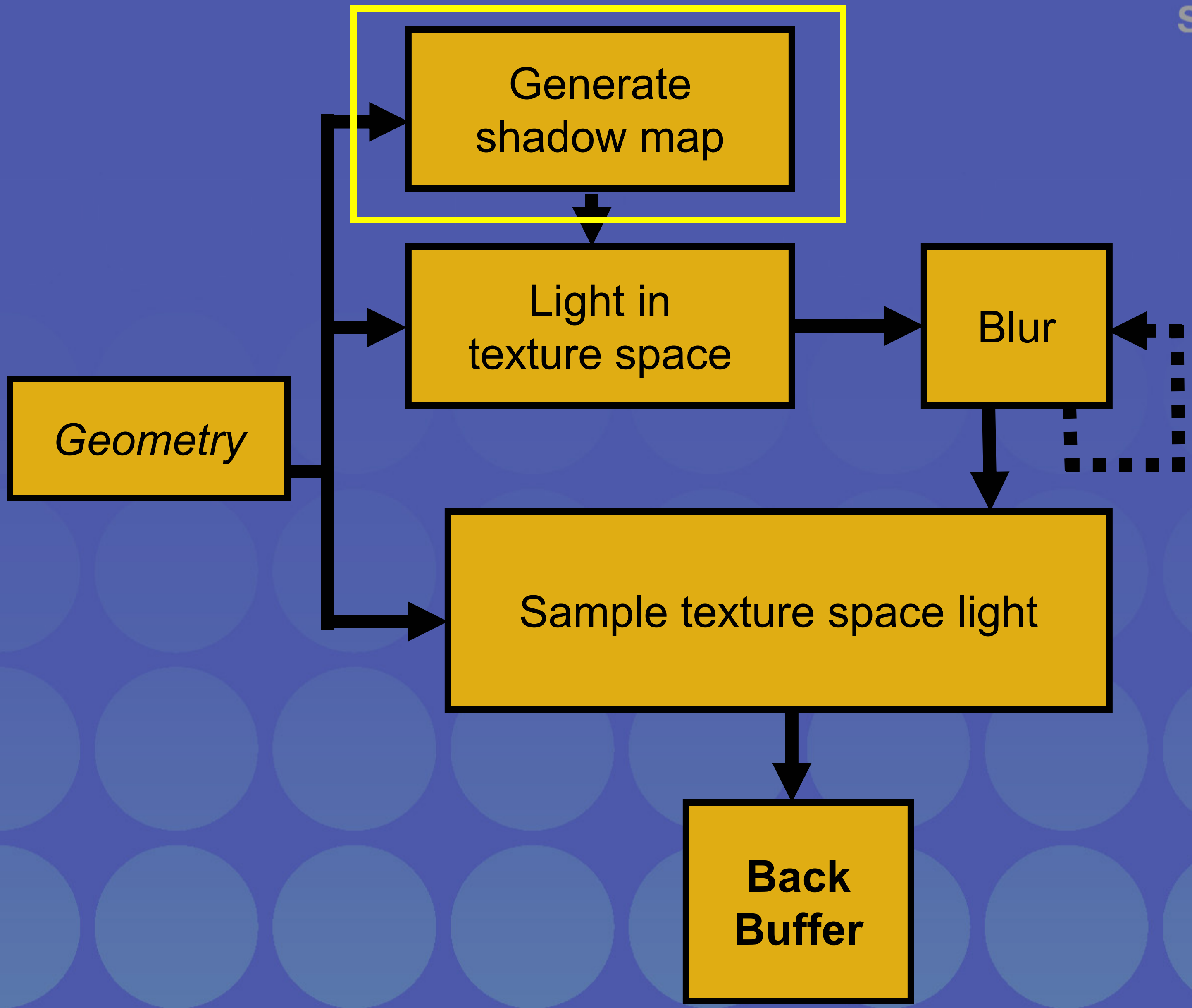
- Used shadow maps
  - Apply shadows during texture lighting
  - Get “free” blur
    - Soft shadows
    - Simulates subsurface interaction
    - Lower precision/size requirements
    - Reduces artifacts
- Only doing shadows from one key light
  - If using multiple lights,
    still same # of blur passes

Shadow maps

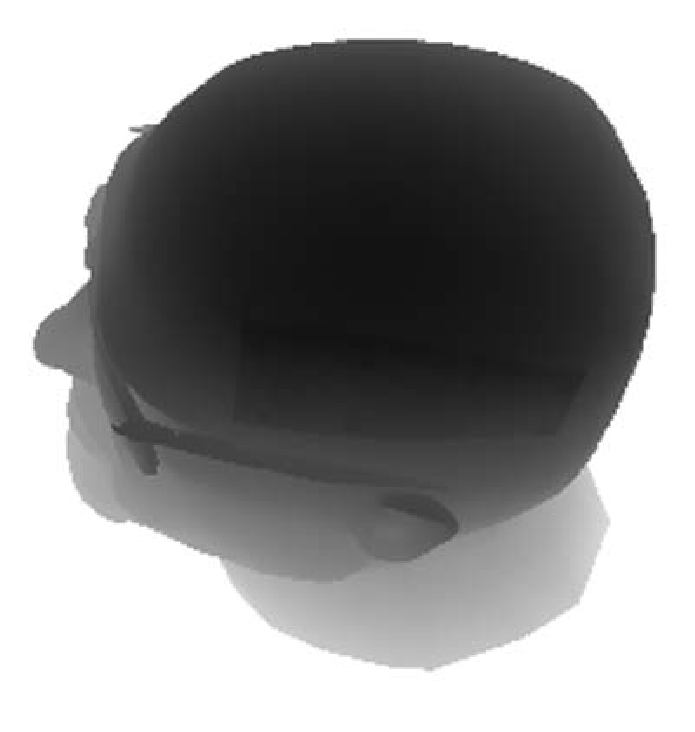
- Create projection matrix to generate map from the light's point of view
  - Used bounding sphere of head to ensure texture space is used efficiently
- Shadow map rendering pass:
  Write depth into off-screen texture
- On texture lighting pass:
  Test depth values in pixel shader



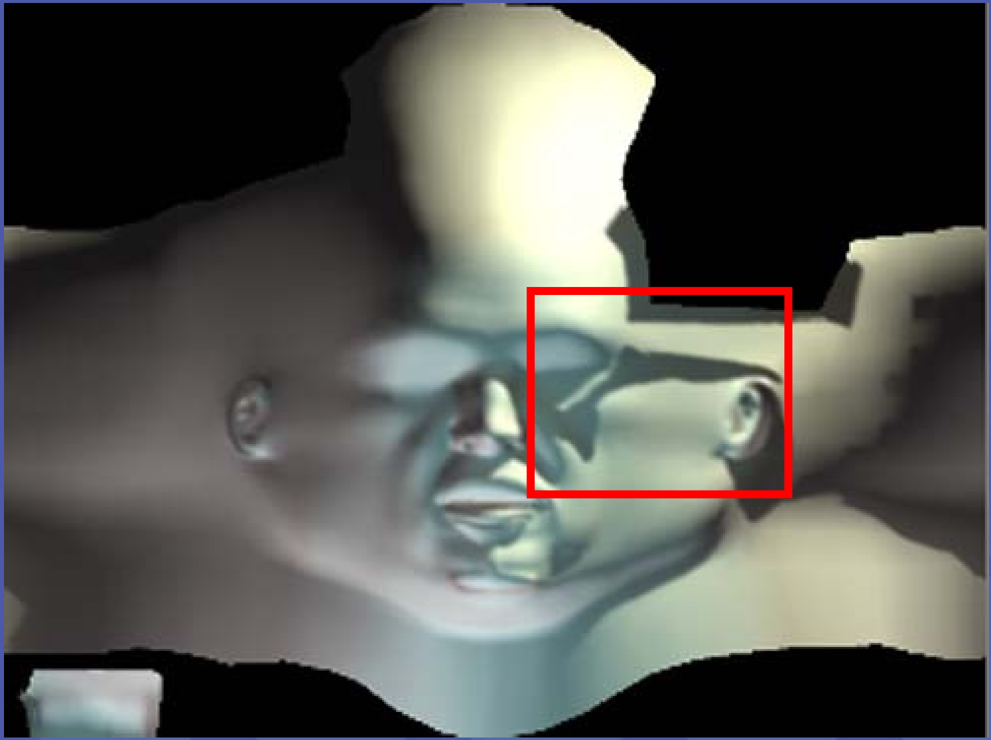
Basic approach + shadows



Shadow map and shadowed lit texture



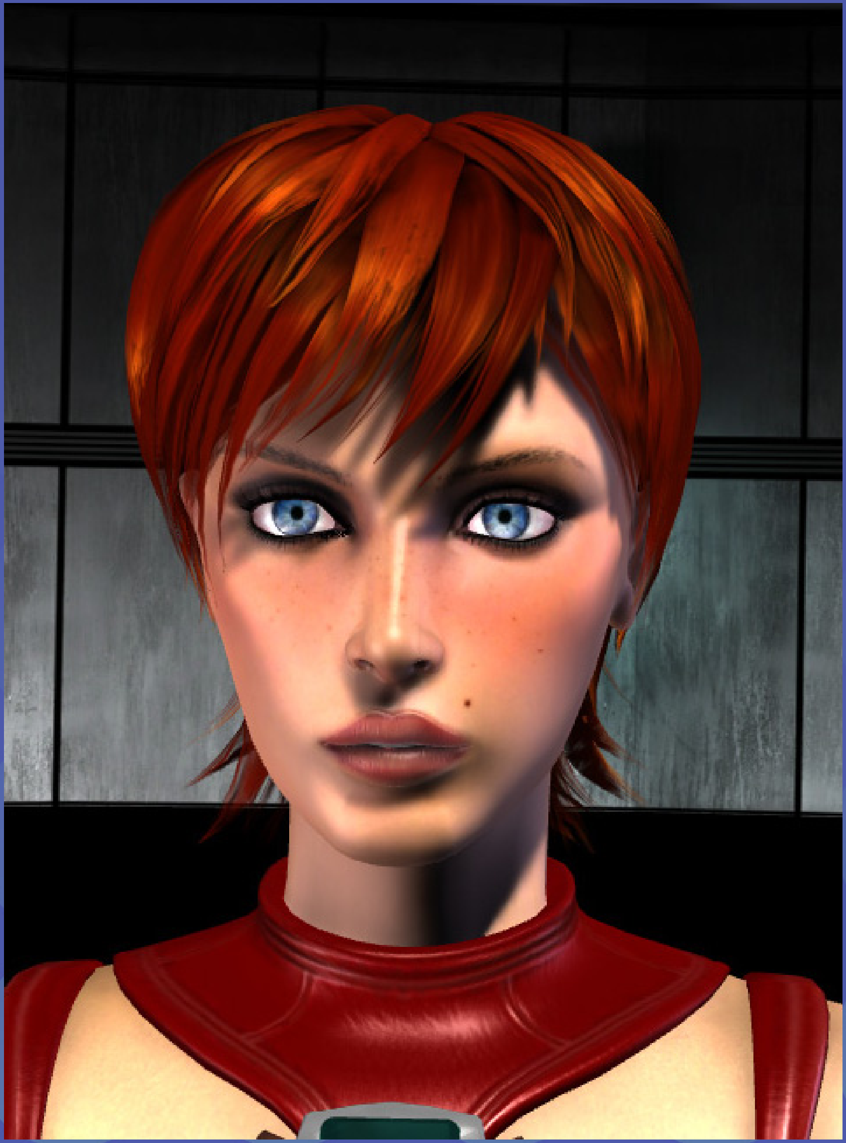
Shadow Map (depth)

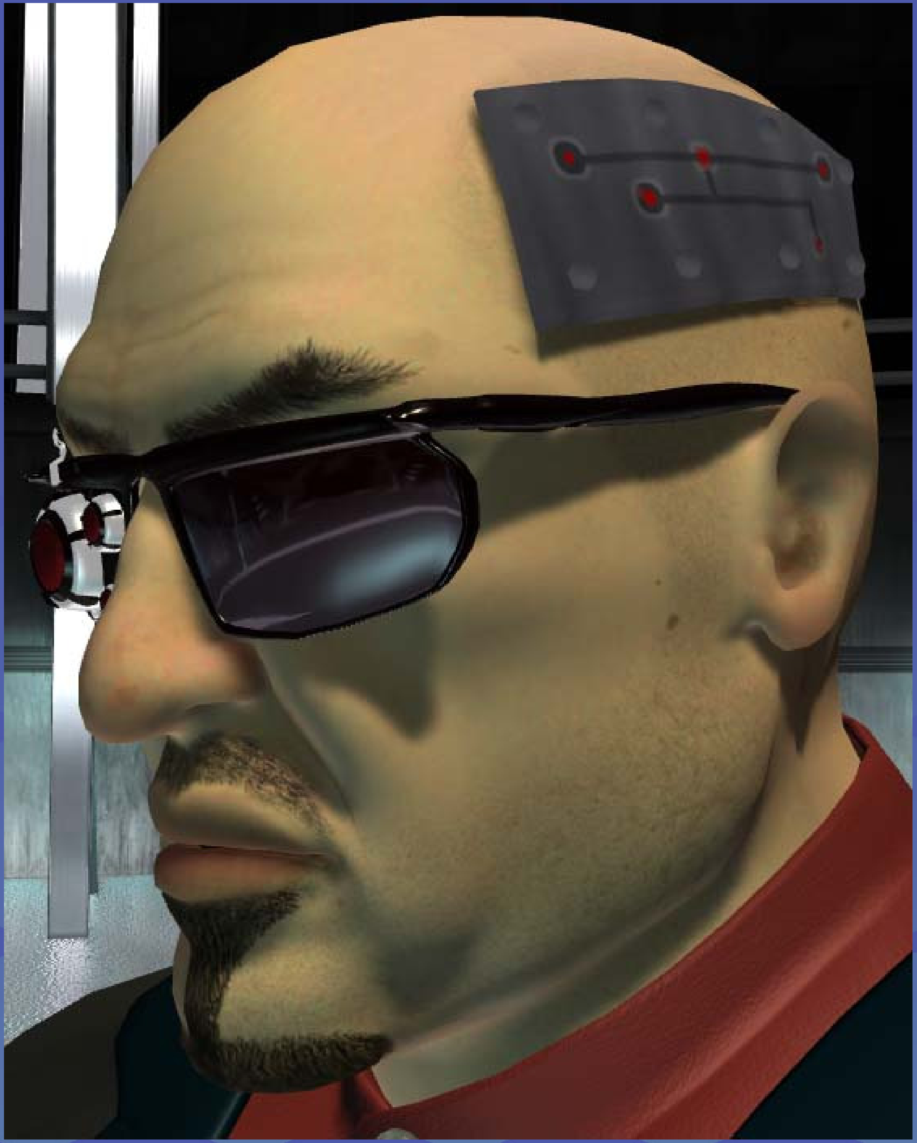


Shadows in Texture Space

Results with shadows

SIGGRAPH2004

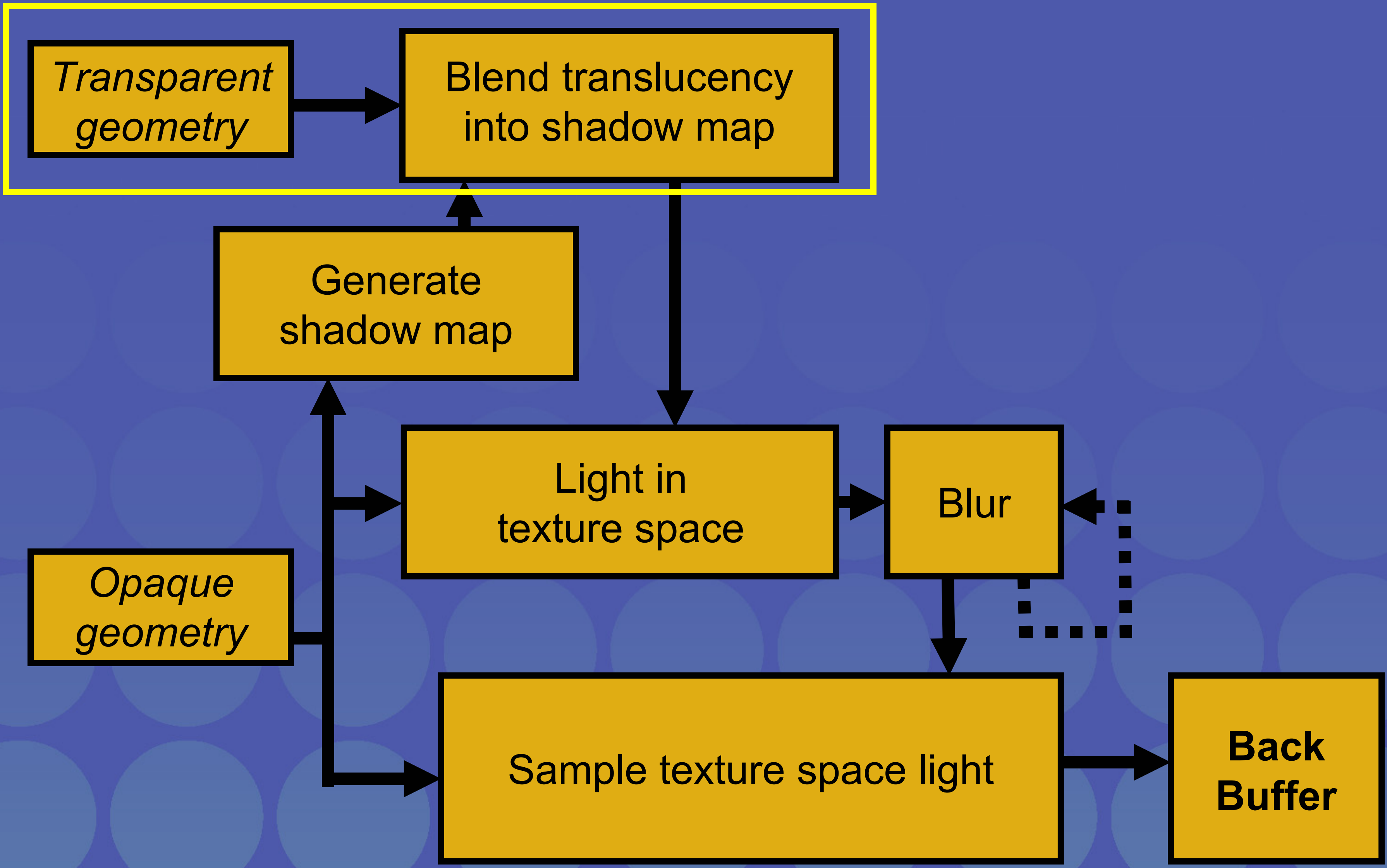




Shadows from translucent objects

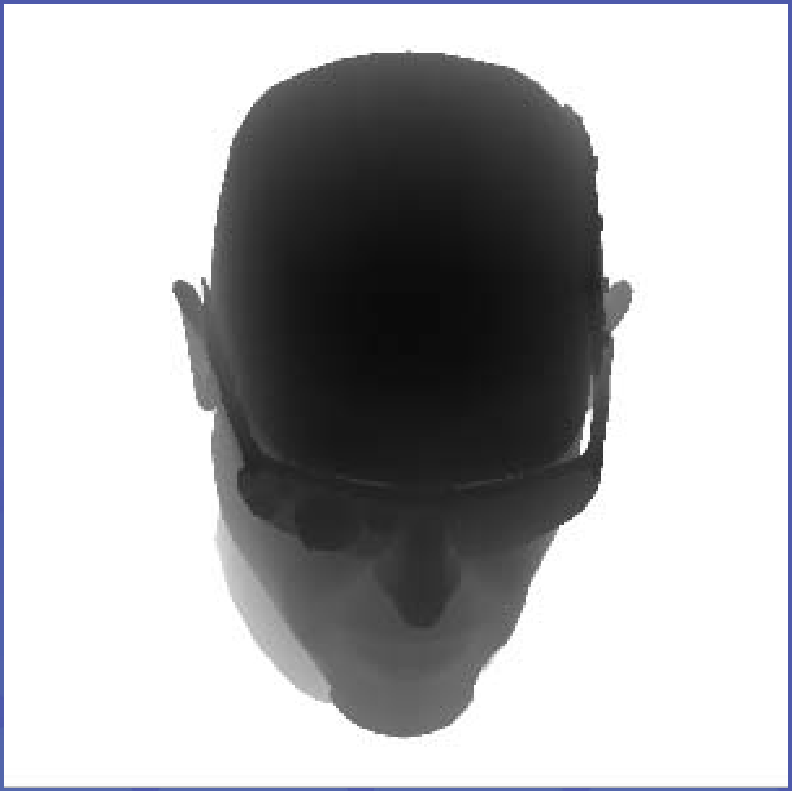
- Algorithm:
  - Draw depth of opaque shadow geometry to shadow buffer alpha channel
  - Aditively blend RGB of translucent shadow geometry into shadow buffer RGB channels
    - Depth test on
    - Depth write off
  - On texture lighting pixel shader:
    non-shadowed pixels are multiplied by the translucent RGB value

Basic approach + shadows

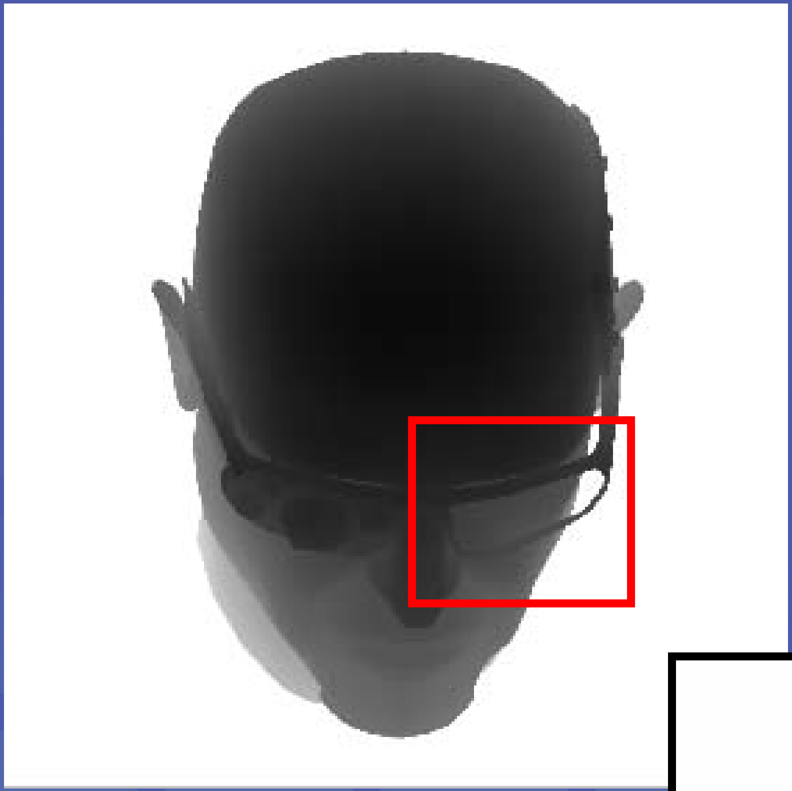


Shadow map for transparent shadows

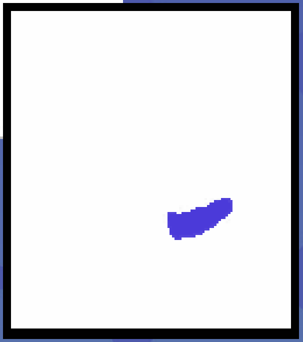
SIGGRAPH2004



Opaque shadow map

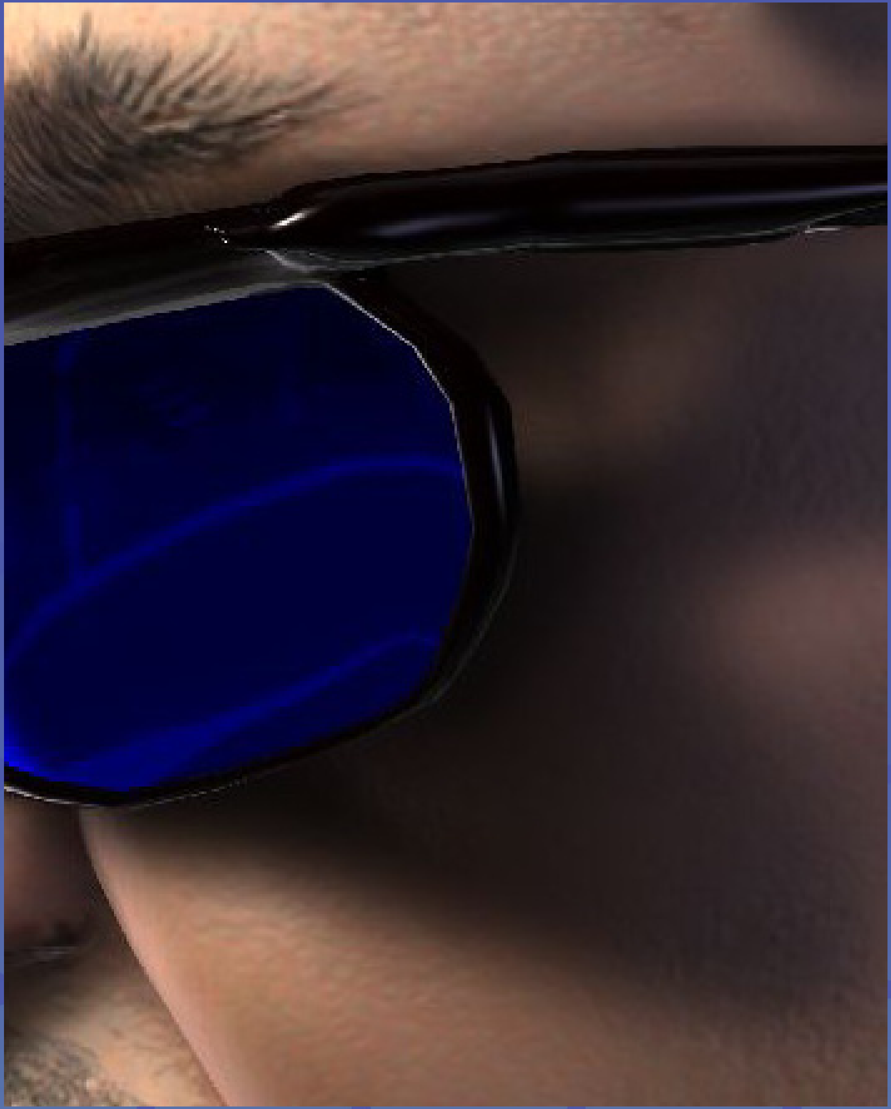


Translucent shadow map

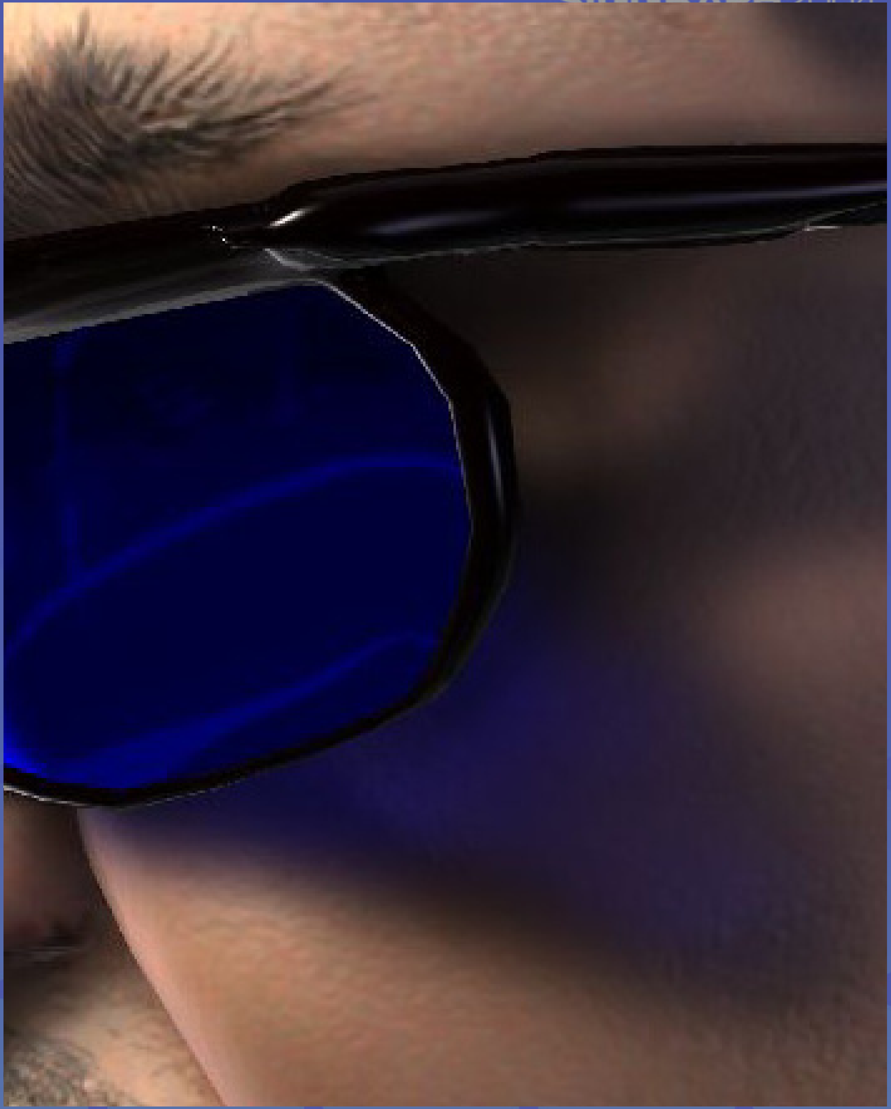


RGB

Translucent shadows results



Opaque Shadows

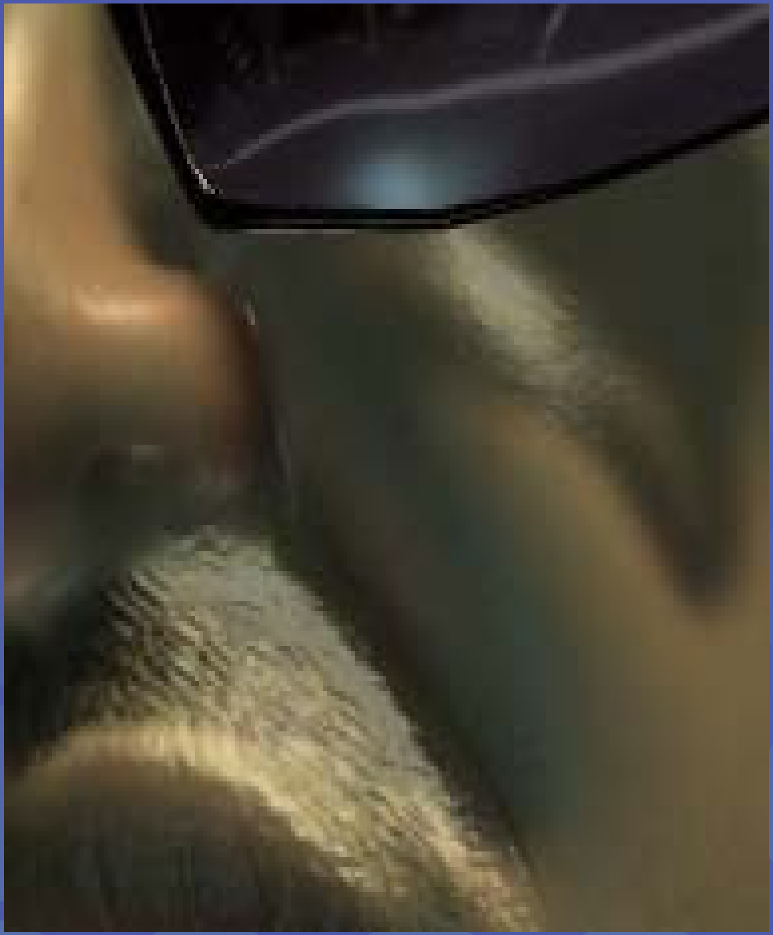


Translucent Shadows

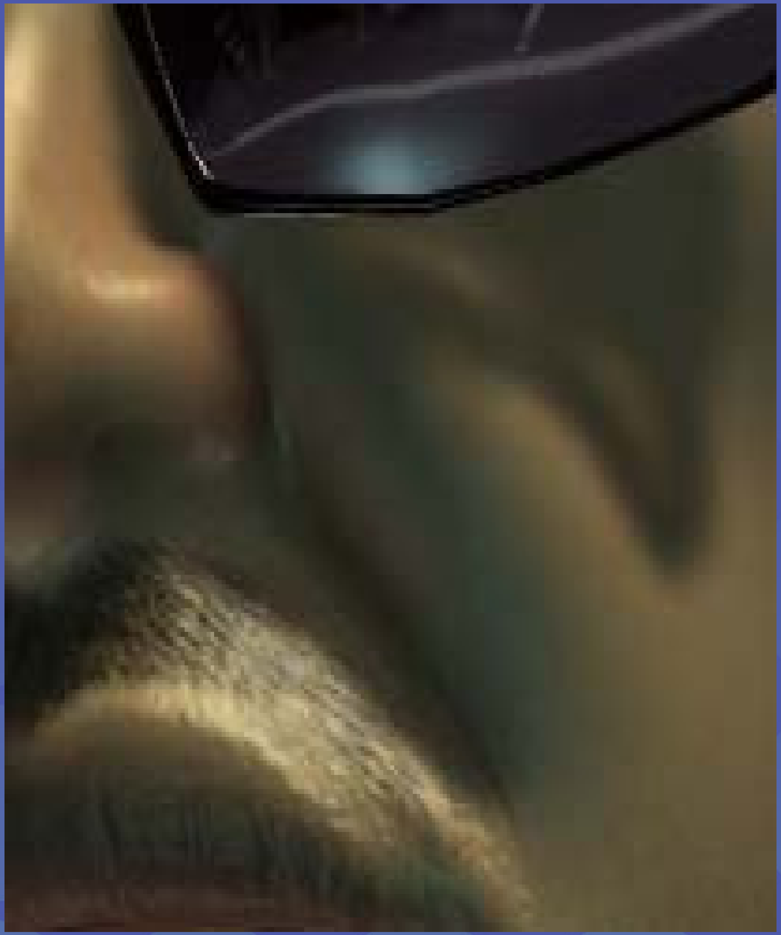
Specular

- Use bump map for specular lighting
- Per-pixel exponent
- Need to shadow specular
  - Can’t have shadowed region with specular lighting spots
  - Expensive to do yet another blur pass for shadows
- Use luminance of blurred diffuse texture
  - Modulate specular from shadowing light by luminance of texture space light (which very low-frequency)
  - Darkens specular in shadowed areas but preserves lighting in unshadowed areas
- Not a limitation, just an optimization
  - One could do a separate blur pass for shadows

Specular shadow dim results



Specular without shadows



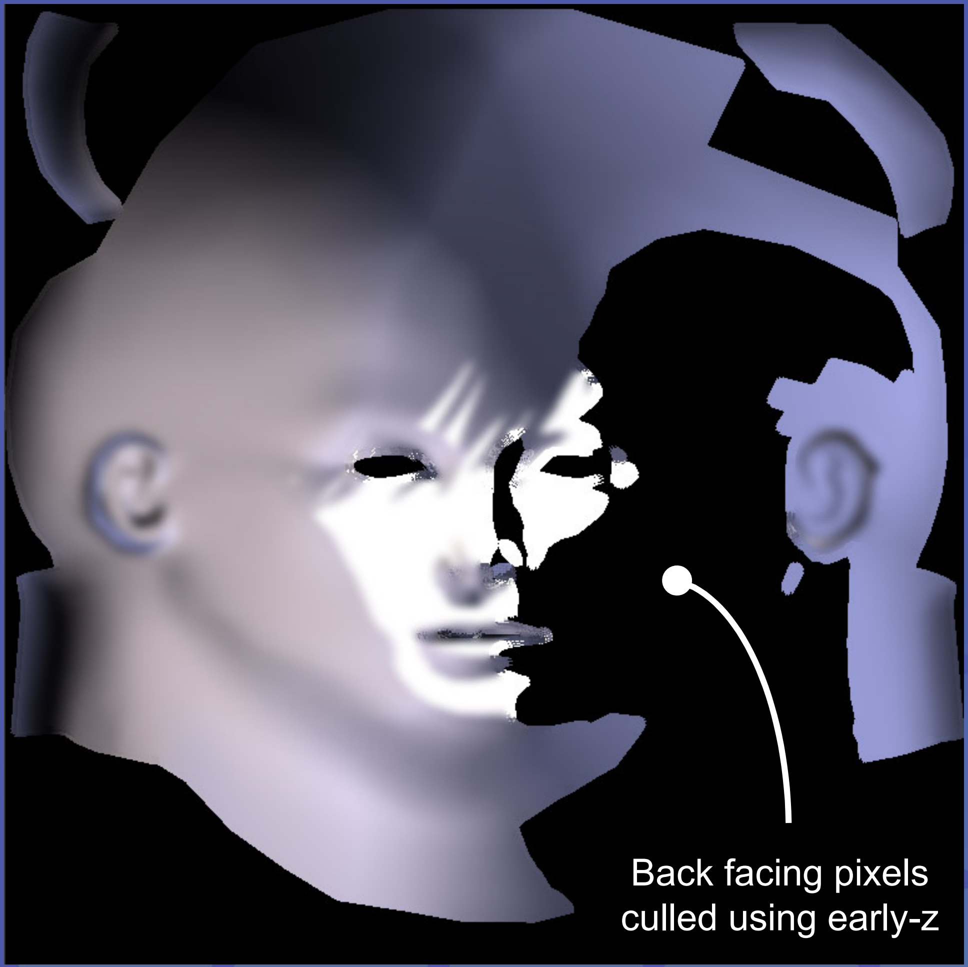
Specular with shadows

Using Early-Z for Culling

- Testing z-buffer prior to pixel shader execution
  - Can cull expensive pixel shaders
  - Only applicable when pixel shader does not output depth
- This texture-space operation doesn’t need the z buffer for hidden surface removal
- Can store any value of Z buffer
- Use Early-Z to cull computations
  - Back face culling
  - Distance and frustum culling
- Set z buffer on lighting pass according to frustum, distance from viewer, and facing-ness of polygons
- Set the z test such that non-visible polygons fail Z test
- Reduces cost of image-space blurs in regions that don’t need it

Back Face Culling

Over the shoulder view of Ruby



Back facing pixels culled using early-z

Summary

- Simple and fast skin rendering algorithm
- Texture-space blur with dilation
- Soft shadows “for free”
- Translucent shadows
- Early-Z culling acceleration techniques

Acknowledgements

- Thanks to Eli Turner for help with the artwork
- Thanks to Chris Brennan for the colored translucent shadows suggestion

References

- [Borshukov03] Borshukov and Lewis. *Realistic Human Face Rendering for “The Matrix Reloaded”*. Technical Sketches, SIGGRAPH 2003.
- [Mertens03] Mertens et al. *Efficient Rendering of Local Subsurface Scattering.* Pacific Graphics 2003.

SIGGRAPH2004

Demo

SIGGRAPH2004

Questions?