Real Time Skin Rendering

David Gosselin
3D Application Research Group
ATI Research, Inc.

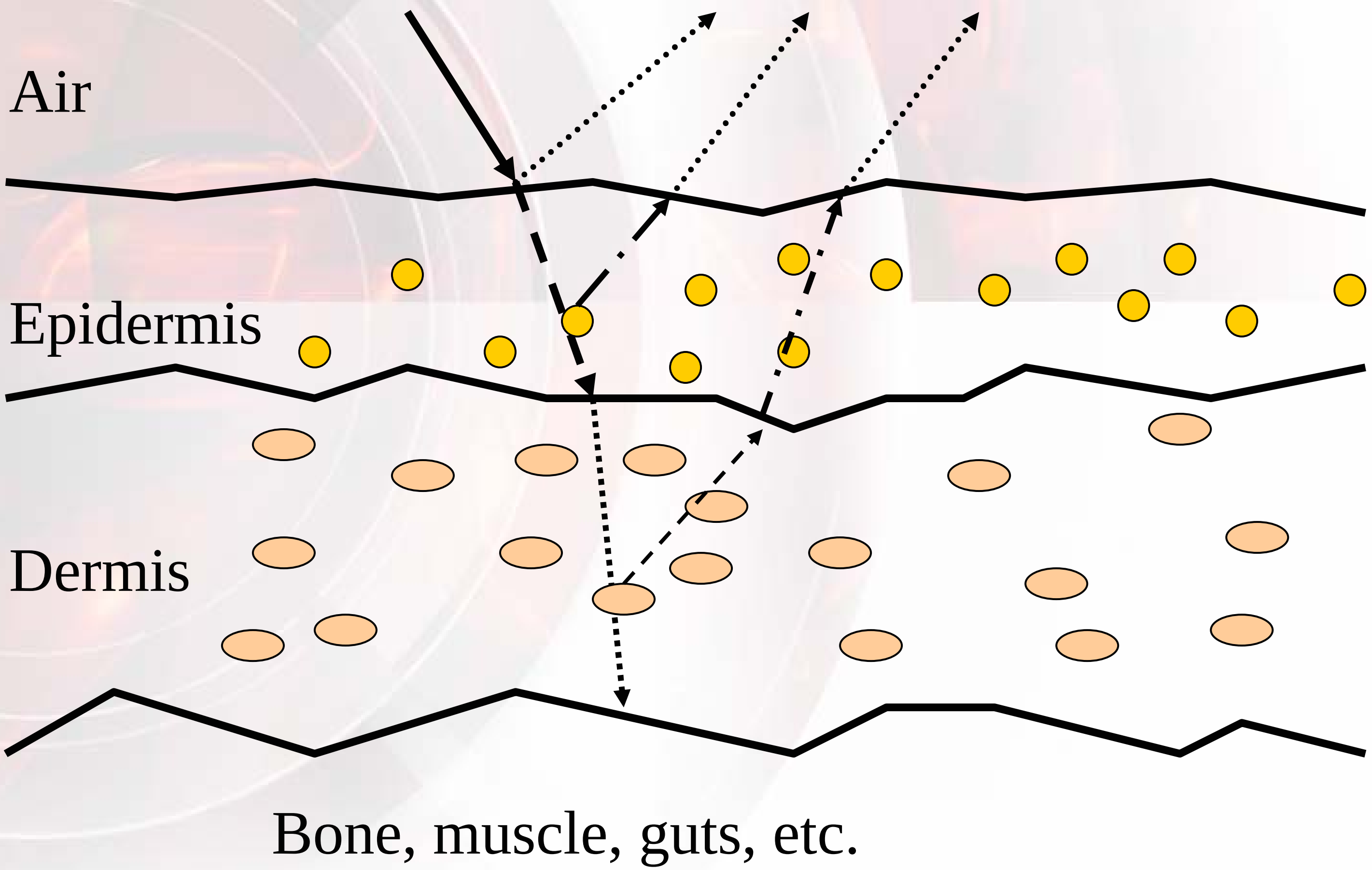
Overview

- Background
- Texture space lighting
- Blur and dilation
- Adding shadows
- Specular with shadows
- Demo

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

Rough Skin Cross Section



Basis for Our Approach

- SIGGRAPH 2003 sketch **Realistic Human Face Rendering for “The Matrix Reloaded”**
- Rendered a 2D light map
- Simulate subsurface diffusion in image domain (different for each color component)
- Used traditional ray tracing for areas where light can pass all the way through (e.g.. Ears)

Texture Space Subsurface Scattering

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

- From Sushi Engine

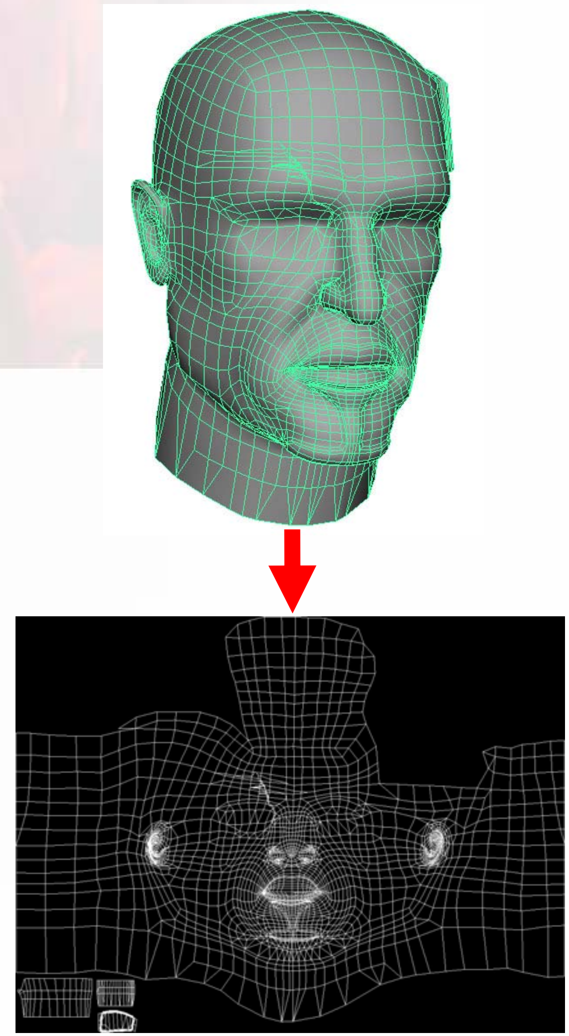
Current skin in Real Time

Texture Space Lighting for Real Time

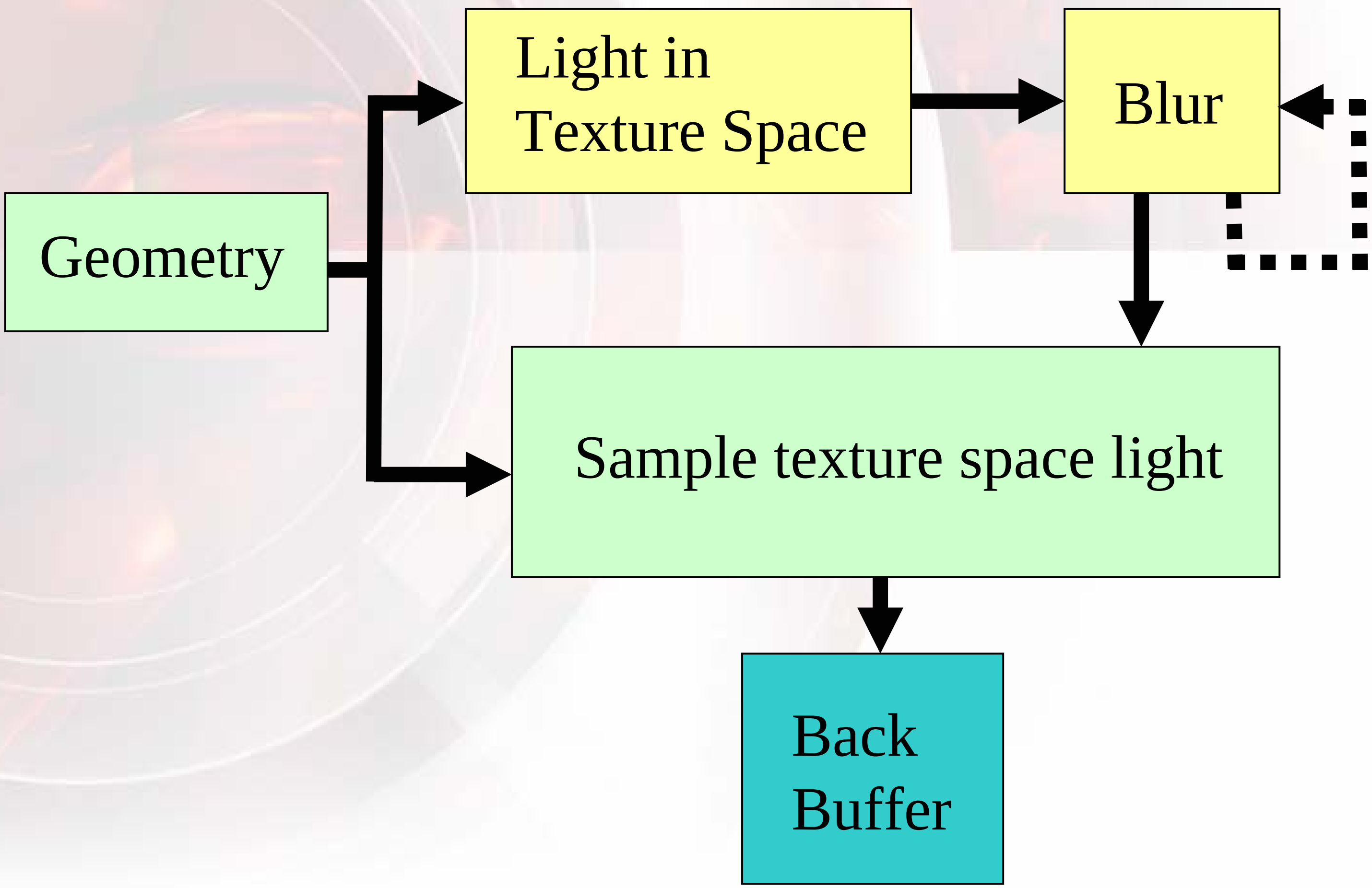
- Render diffuse lighting into an off-screen texture using texture coordinates as position
- Blur the off-screen diffuse lighting
- Read the texture back and add specular lighting in subsequent pass
- We only used bump map for the specular lighting pass

Texture Coordinates as Position

- Need to light as a 3D model but draw into texture
- By passing texture coordinates as “position” the rasterizer does the unwrap
- Compute light vectors based on 3D position and interpolate



Basic Approach



Texture Lighting Vertex Shader

```
VsOutput main (VsInput i)
{
   // Compute output texel position
   VsOutput o;
   o.pos.xy = i.texCoord*2.0-1.0;
   o.pos.z = 1.0;
   o.pos.w = 1.0;

   // Pass along texture coordinates
   o.texCoord = i.texCoord;

   // Skin
   float4x4 mSkinning = SiComputeSkinningMatrix (i.weights,
                                                  i.indices);
   float4 pos = mul (i.pos, mSkinning);
   pos = pos/pos.w;
   o.normal = mul (i.normal, mSkinning);

   // Compute Object light vectors
   // etc.
   . . .
```

Texture Lighting Pixel Shader

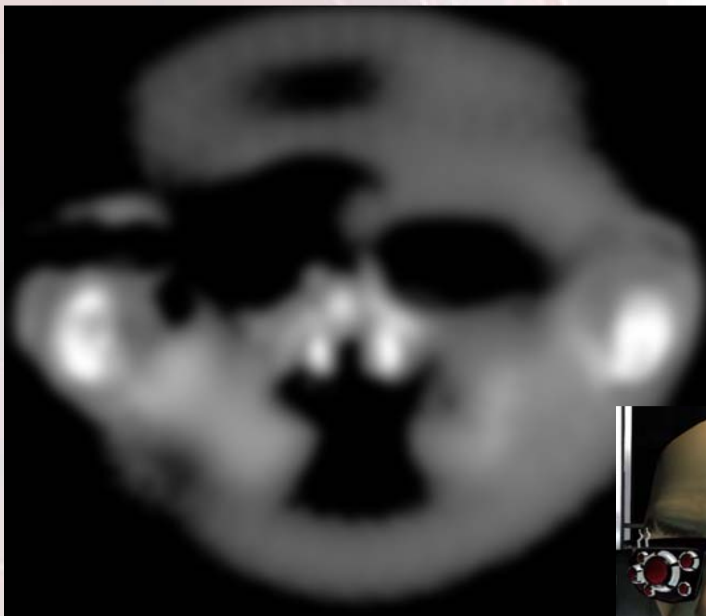
```
float4 main (PsInput i) : COLOR
{
   // Compute Object Light 0
   float3 vNormal = normalize (i.normal);
   float3 lightColor = 2.0 * SiGetObjectAmbientLightColor(0);
   float3 vLight = normalize (i.oaLightVec0);
   float NdotL = SiDot3Clamp (vNormal, vLight);
   float3 diffuse = saturate (NdotL * lightColor);

   // Compute Object Light 1 & 2
   . . .

   float4 o;
   o.rgb = diffuse;
   float4 cBump = tex2D (tBump, i.texCoord);
   o.a = cBump.a; // Save off blur size
   return o;
}
```

Texture Lighting Results

Blur

- Used to simulate the subsurface component of skin lighting
- Used a grow-able Poisson disc filter (more details on this filter later)
- Read the kernel size from a texture
- Allows varying the subsurface effect
  - Higher for places like ears/nose
  - Lower for places like cheeks

Blur Size Map and Blurred Lit Texture



Blur Kernel Size

Result

Texture Space Lighting

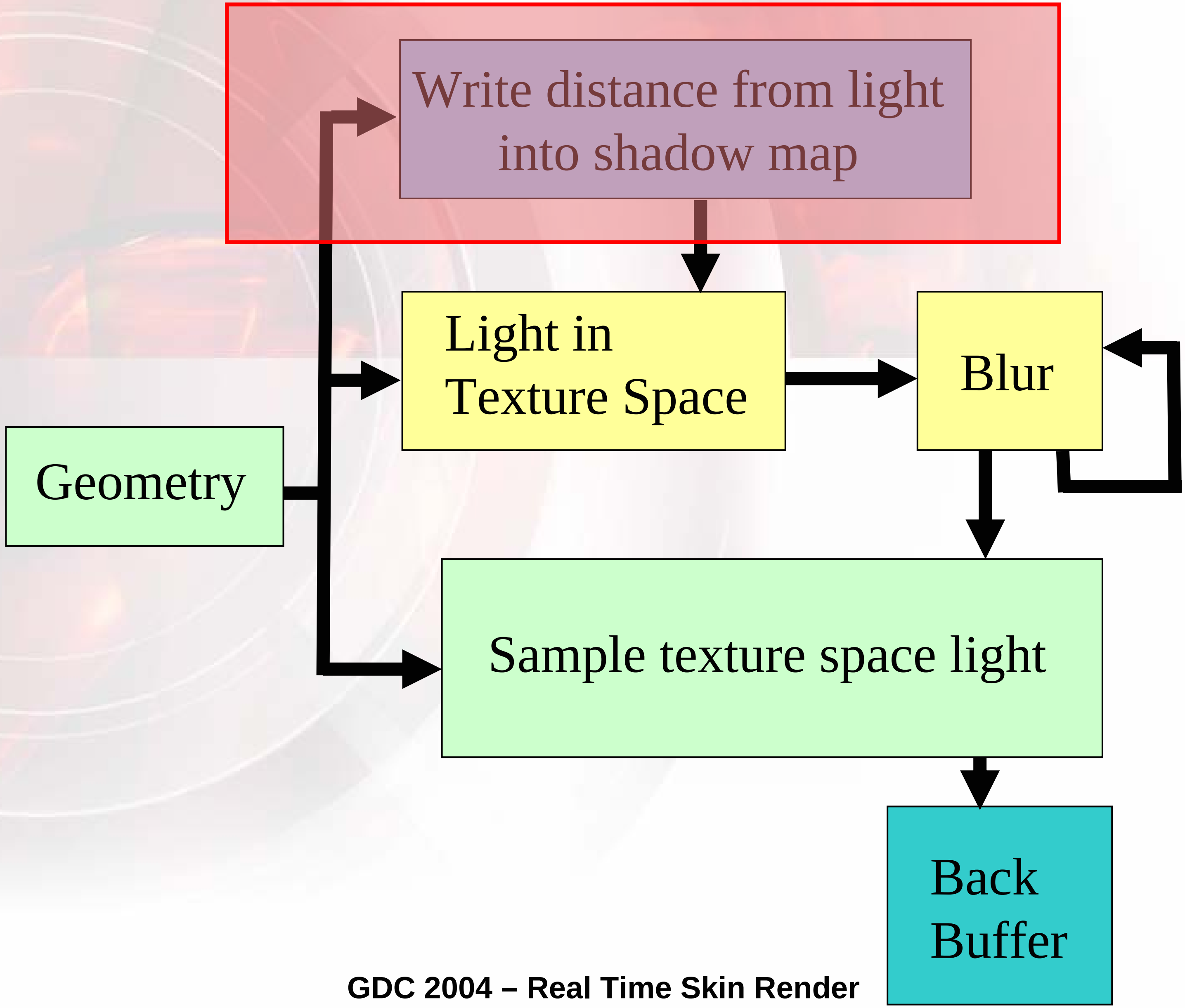
Shadows

- Use 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

Shadow Maps

- Create projection matrix to generate map from the light's point of view
- Use bounding sphere of head to ensure the most texture space is used
- Write depth from light into off-screen texture
- Test depth values in pixel shader

Texture Lighting With Shadows



Shadow Map Vertex Shader

```
float4x4 mSiLightProjection; // Light projection matrix
VsOutput main (VsInput i)
{
   VsOutput o;

   // Compose skinning matrix
   float4x4 mSkinning = SiComputeSkinningMatrix(i.weights,
                                                 i.indices);

   // Skin position/normal and multiply by light matrix
   float4 pos = mul (i.pos, mSkinning);
   o.pos = mul (pos, mSiLightProjection);

   // Compute depth (Pixel Shader is just pass through)
   float dv = o.pos.z/o.pos.w;
   o.depth = float4(dv, dv, dv, 1);
   return o;
}
```

Texture Lighting Vertex Shader with Shadows

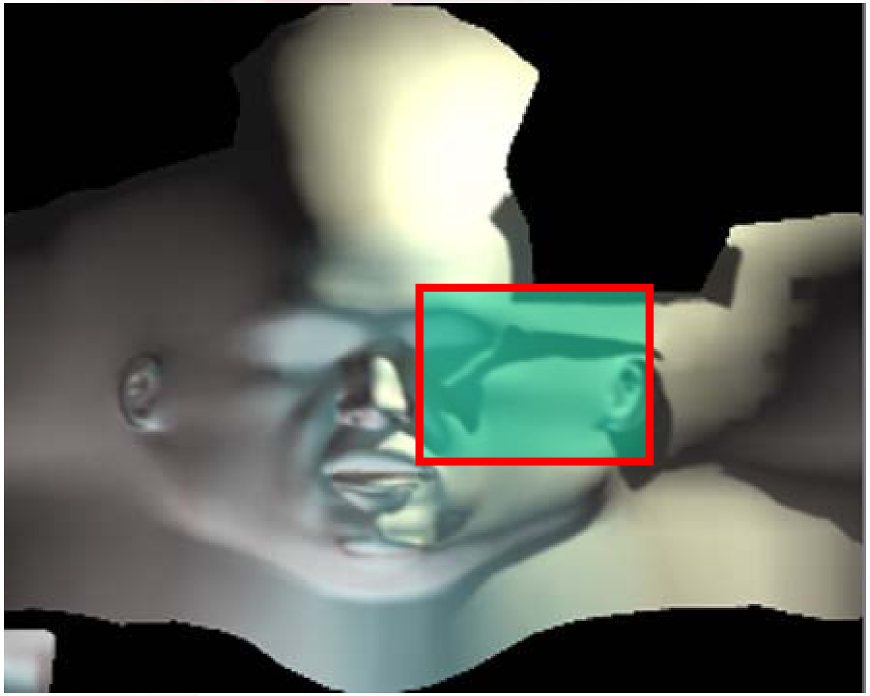
```
VsOutput main (VsInput i)
{
   // Same lead in code as before
   . . .

   // Compute texture coordintates for shadow map
   o.posLight = mul(pos, mSiLightKingPin);
   o.posLight /= o.posLight.w;
   o.posLight.xy = (o.posLight.xy + 1.0f)/2.0f;
   o.posLight.y = 1.0f-o.posLight.y;
   o.posLight.z -= 0.01f;
   return o;
}
```

Texture Lighting Pixel Shader with Shadows

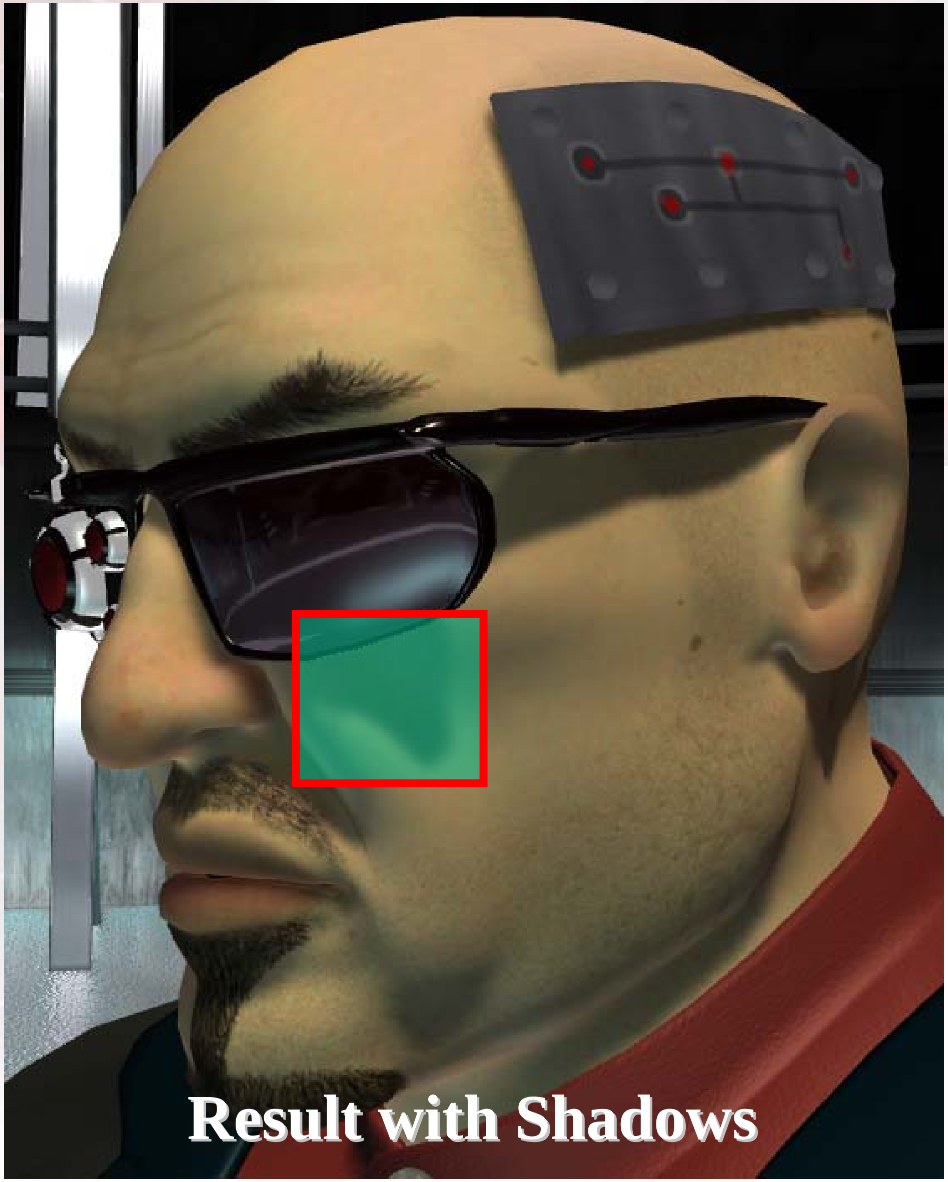
```
sampler tShadowMap;
float faceShadowFactor;
float4 main (PsInput i) : COLOR
{
   // Same lead in code
   . . .

   // Compute Object Light 0
   float3 lightColor = 2.0 * SiGetObjectAmbientLightColor(0);
   float3 vLight = normalize (i.oaLightVec0);
   float NdotL = SiDot3Clamp (vNormal, vLight);
   float VdotL = SiDot3Clamp (-vLight, vView);
   float4 t = tex2D(tShadowMap, i.posLight.xy);
   float lfac = faceShadowFactor;
   if (i.posLight.z < t.z) lfac = 1.0f;
   float3 diffuse = lfac *
                  saturate ((fresnel*VdotL+NdotL)*lightColor);

   . . .// The rest of the shader is the same as before
}
```

Shadow Map and Shadowed Lit Texture

Shadow Map (depth)



Shadows in Texture Space


Result with Shadows

Specular

- Use bump map for specular lighting
- Per-pixel exponent
- Need to shadow specular
  - Hard to blur shadow map directly
  - Modulate specular from shadowing light by luminance of texture space light
  - Darkens specular in shadowed areas but preserves lighting in unshadowed areas

Normal Map Compression

New Feature!

- New texture format in new ATI chip
- Exposed in Direct3D using FOURCC: **`ATI2`**
- Two-channels (x and y)
  - Each channel is compressed separately
  - Each uses a block like a DXT5 alpha block
  - Derive Z in pixel shader

$$+\sqrt{1-x^2-y^2}$$

- Useful for tangent-space normal maps
- Can be used for any 2-channel texture

Final Pixel Shader (with specular)

```
sampler tBase;
sampler tBump;
sampler tTextureLit;
float4 vBumpScale;
float specularDim;
float4 main (PsInput i) : COLOR
{
   // Get base and bump map
   float4 cBase = tex2D (tBase, i.texCoord.xy);
   float3 cBump = tex2D (tBump, i.texCoord.xy);

   // Get bumped normal
   float3 vNormal = SiConvertColorToVector (cBump);
   vNormal.z = vNormal.z * vBumpScale.x;
   vNormal = normalize (vNormal);
```

Final Pixel Shader

```
// View, reflection, and specular exponent
float3 vView = normalize (i.viewVec);
float3 vReflect = SiReflect (vView, vNormal);
float exponent = cBase.a*vBumpScale.z + vBumpScale.w;
```

```
// Get "subsurface" light from lit texture.
float2 iTx = i.texCoord.xy;
iTx.y = 1-i.texCoord.y;
float4 cLight = tex2D (tTextureLit, iTx);
float3 diffuse = cLight*cBase;
```

Final Pixel Shader

```
// Compute Object Light 0
float3 lightColor = 2.0 * SiGetObjectAmbientLightColor(0);
float3 vLight = normalize (i.oaLightVec0);
float RdotL = SiDot3Clamp (vReflect, vLight);
float shadow = SiGetLuminance (cLight.rgb);
shadow = pow(shadow,2);
float3 specular = saturate(pow(RdotL,exponent)*lightColor)
                 *shadow;

// Compute Object Light 1 & 2 (same as above but no
// shadow term)
. . .

// Final color
float4 o;
o.rgb = diffuse + specular*specularDim;
o.a = 1.0;
return o;
}
```

Specular Shadow Dim Results

Specular Without Shadows

Specular With Shadows

Demo

Questions?