Real-Time Volume Graphics

[06] Local Volume Illumination

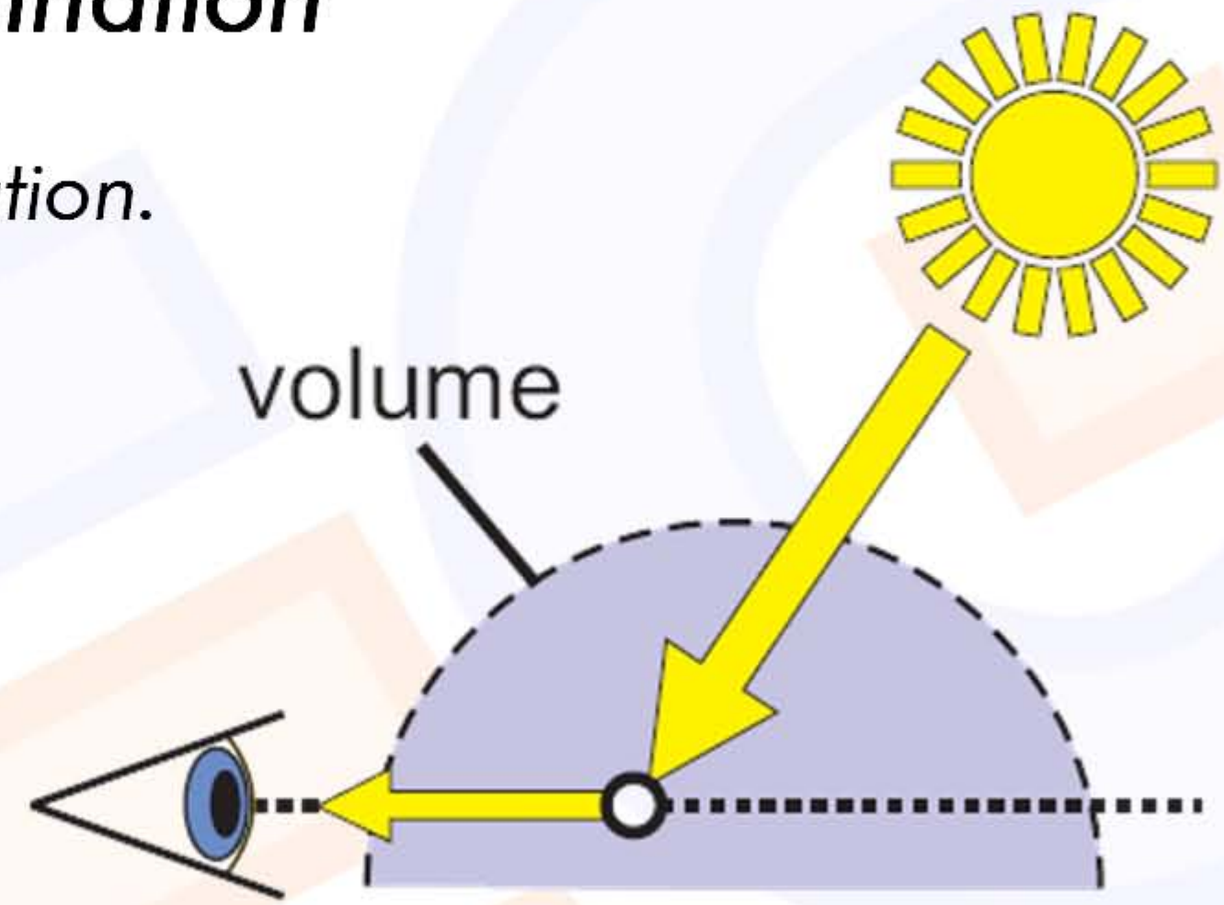
Volume Illumination

- *Up until now:* Light was emitted by the volume
- *Now:* Illumination from external light sources

Types of Volume Illumination

Single scattering, no attenuation.

- Light reaches every point unimpededly
- Light is scattered once before it reaches the eye
- Not physically plausible



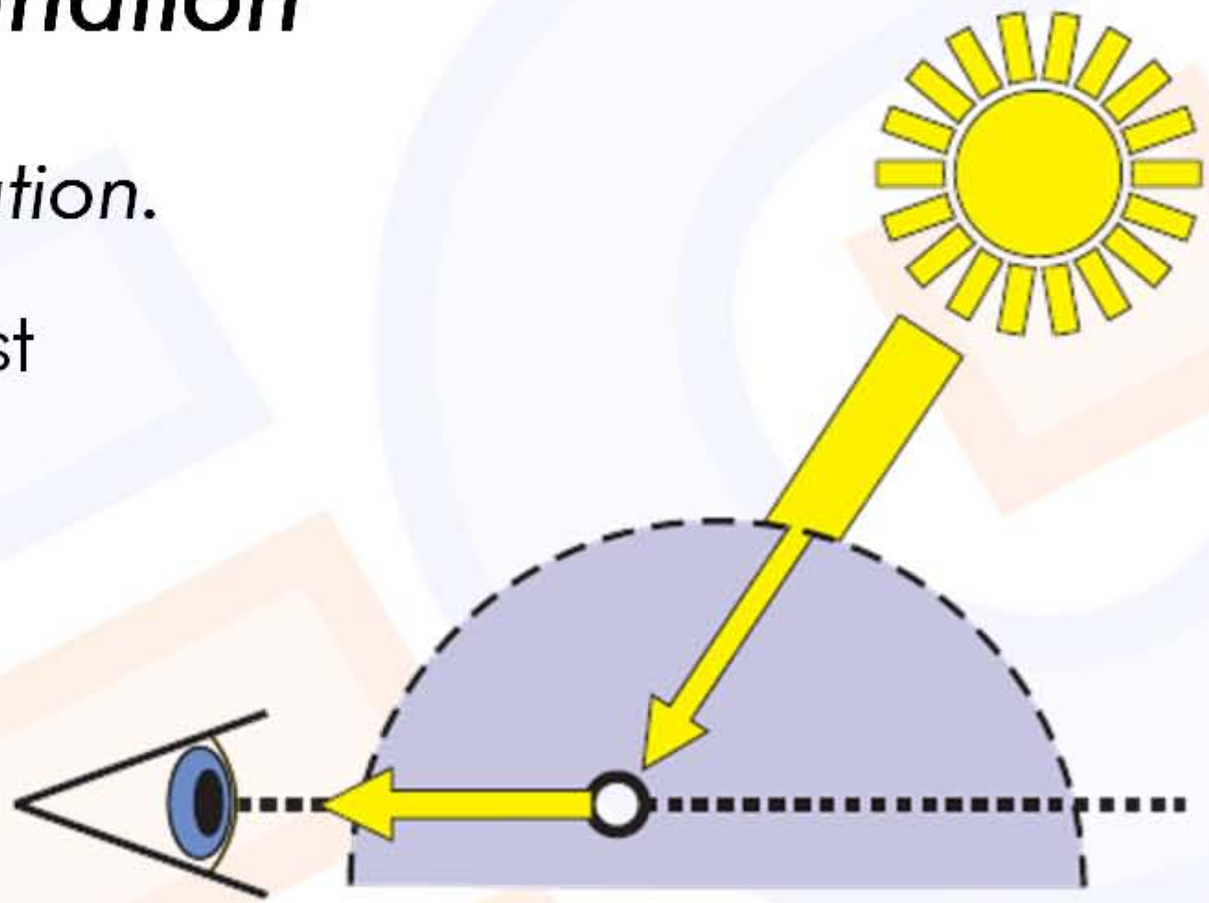
Volume Illumination

- *Up until now:* Light was emitted by the volume
- *Now:* Illumination from external light sources

Types of Volume Illumination

Single scattering with attenuation.

- Light is attenuated along ist way through the volume *(Volumetric shadows)*
- Light is scattered once before it reaches the eye



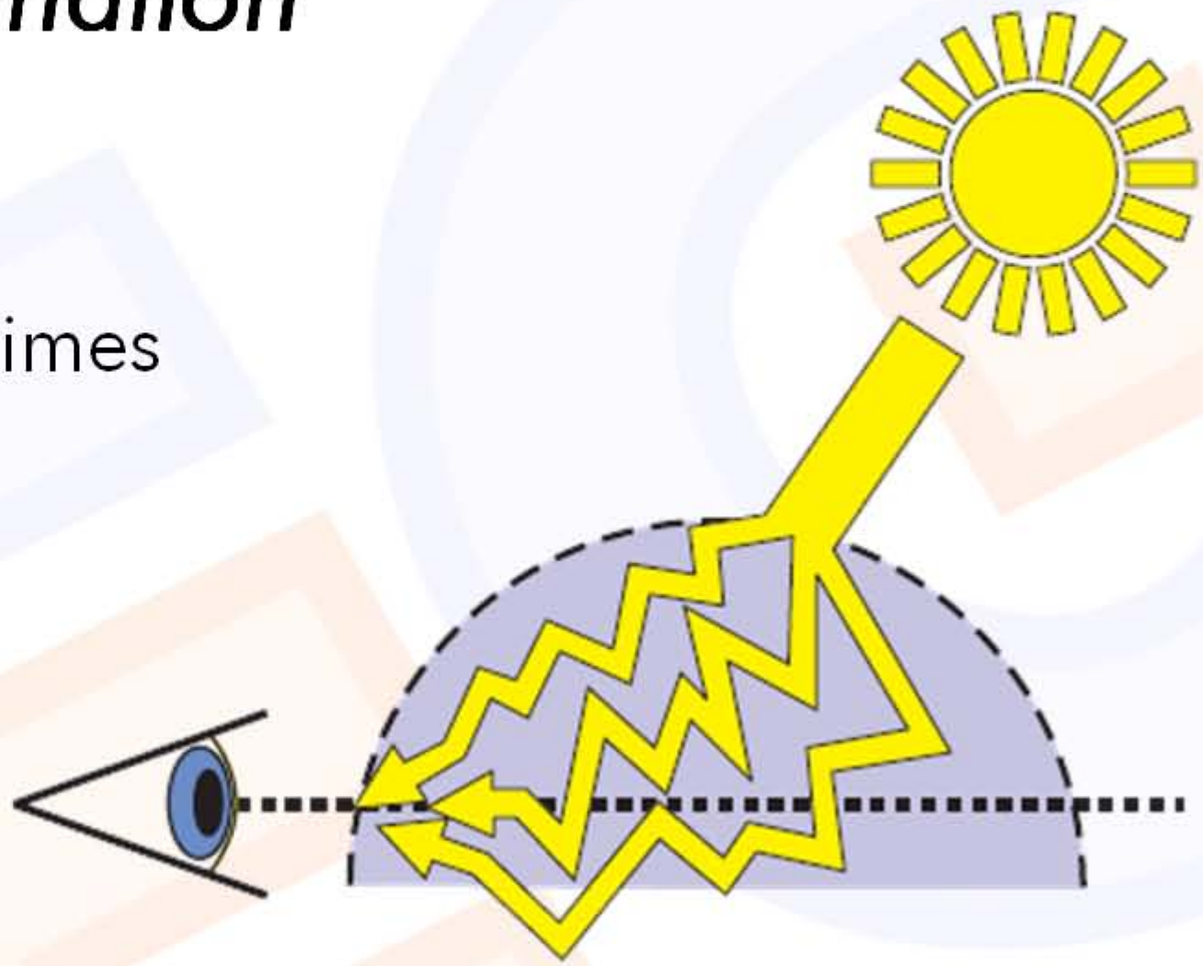
Volume Illumination

- ***Up until now:*** Light was emitted by the volume
- ***Now:*** Illumination from external light sources

Types of Volume Illumination

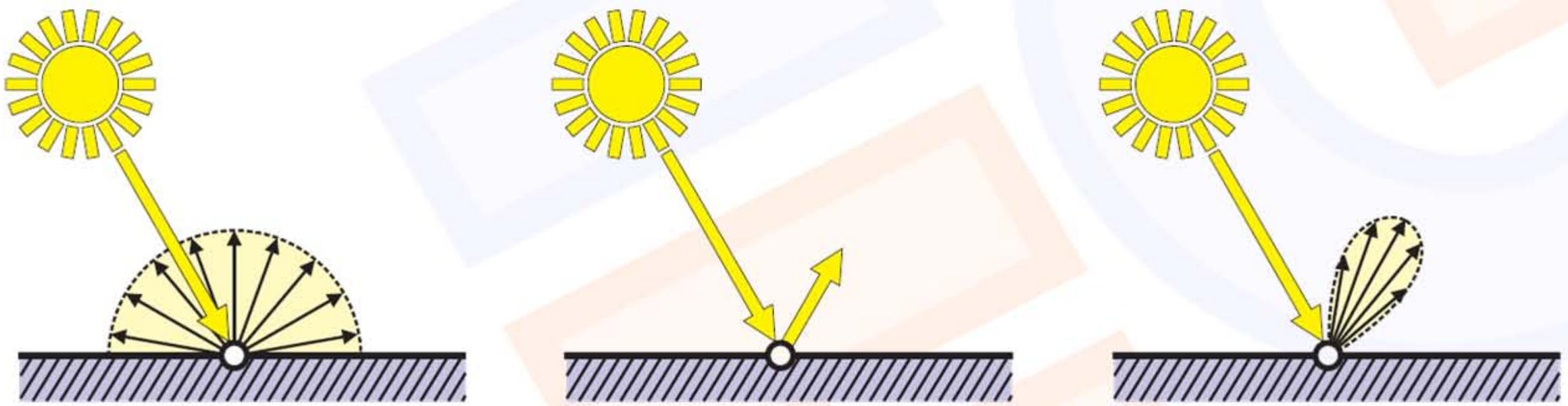
Multiple scattering

- Light is scattered multiple times before it reaches the eye *(Global illumination)*



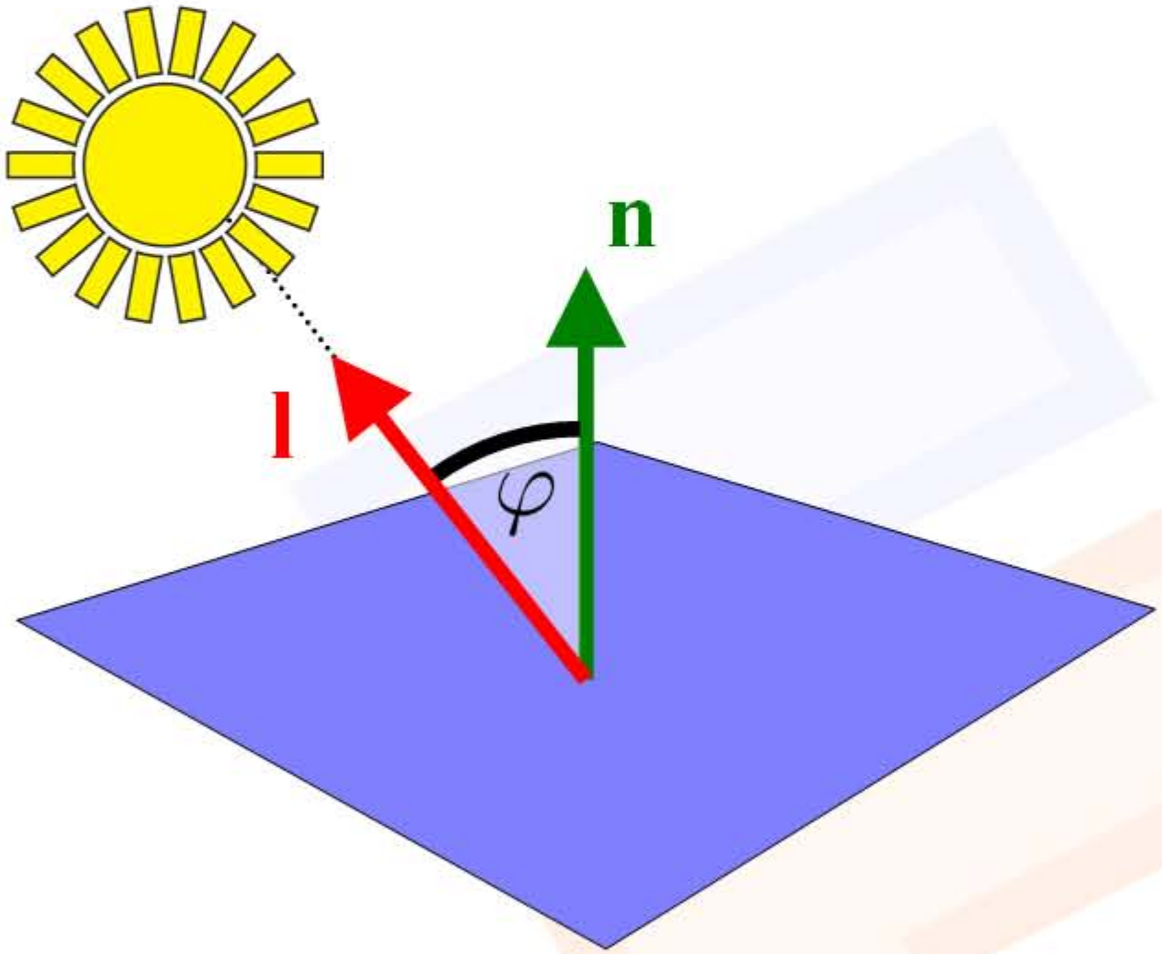
Single Scattering

- Local illumination, similar to surface lighting
  - *Lambertian reflection*
    (light is reflected equally in all directions)
  - *Perfect mirror reflection*
    (light is reflected in exactly one direction)
  - *Specular reflection*
    (light is reflected scattered around the direction of perfect reflection)



Blinn/Phong Illumination

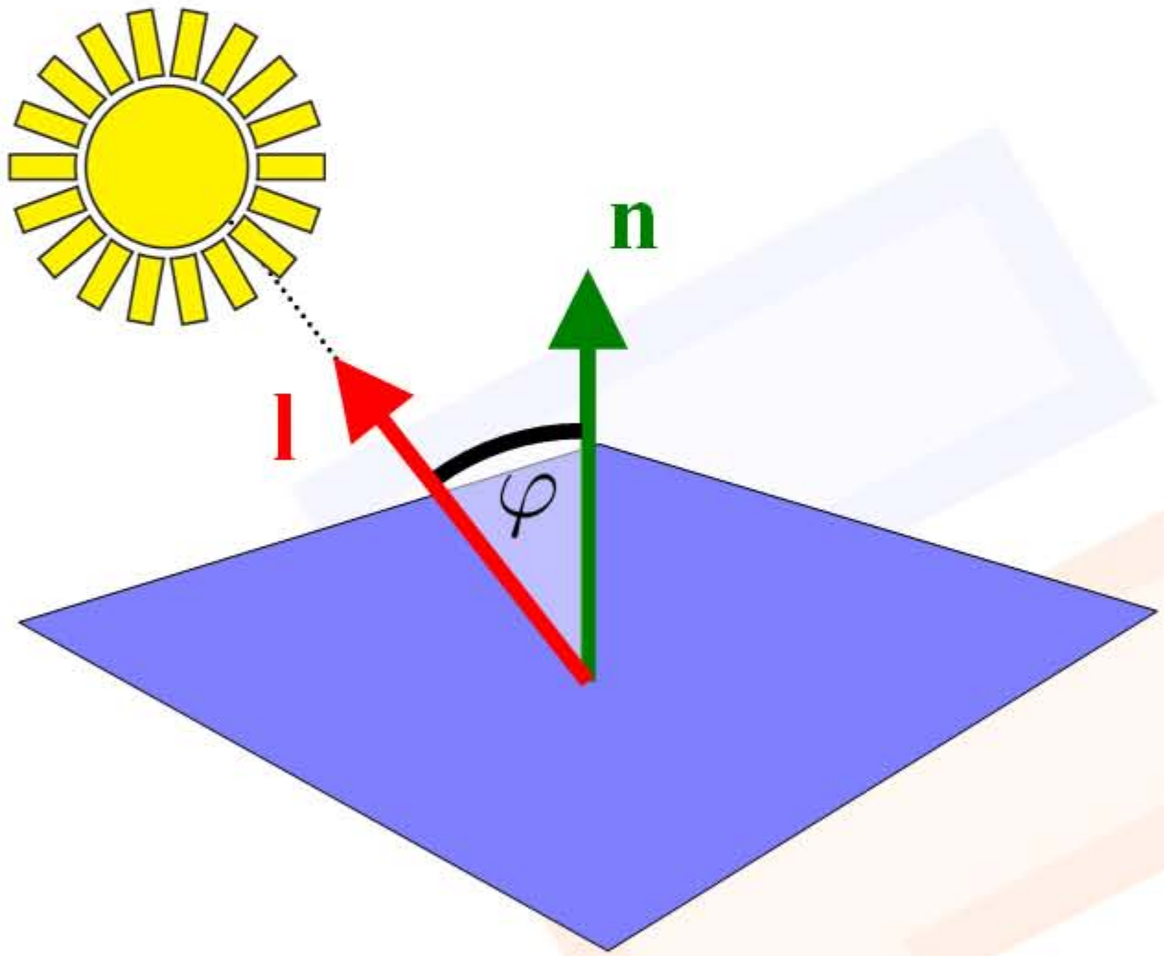
- Diffuse Term (Lambertian reflection)



Blinn/Phong Illumination

- Diffuse Term (Lambertian reflection)

$$\mathbf{I}_{\text{diffuse}} = k_d \, \mathbf{M}_d \, \mathbf{I}_d \cos \varphi$$

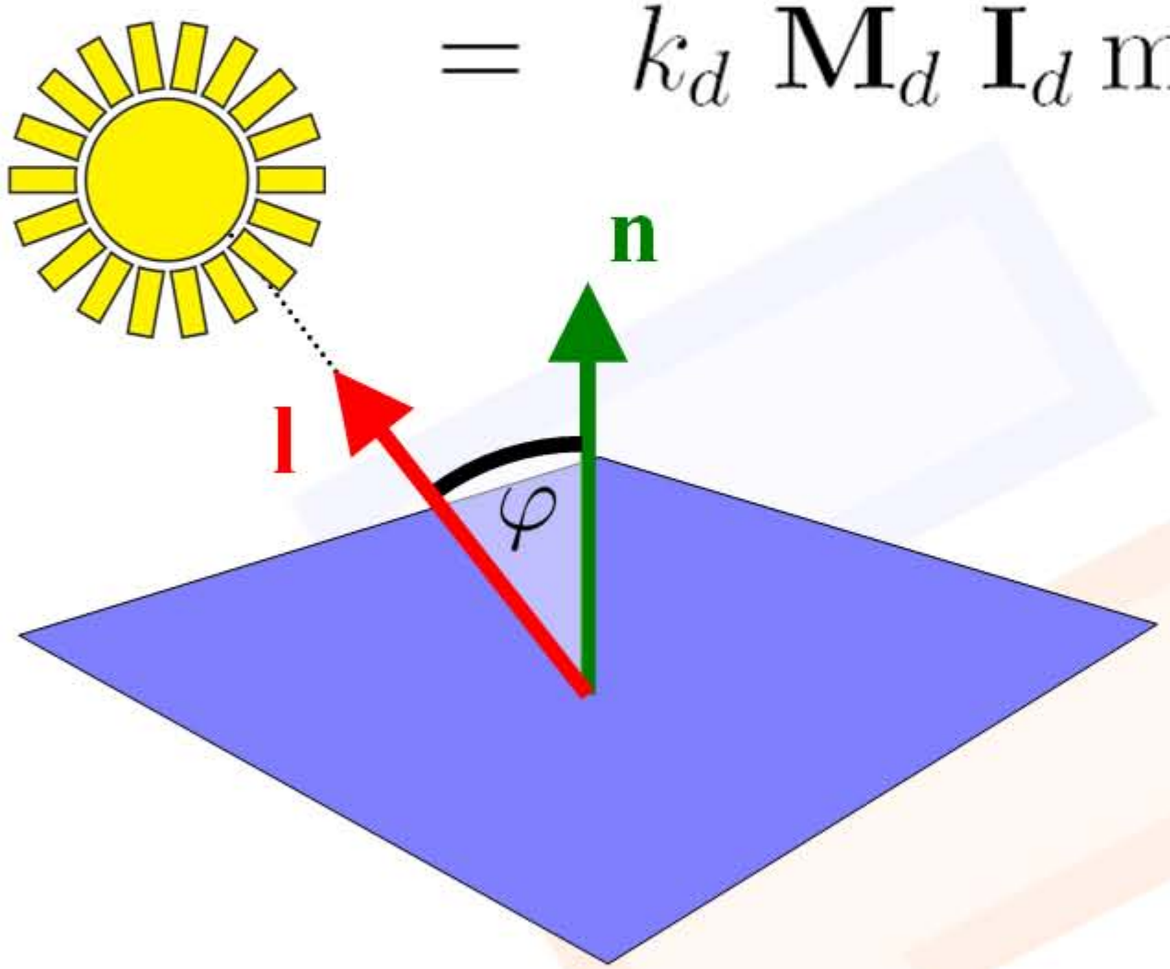


Blinn/Phong Illumination

- Diffuse Term (Lambertian reflection)

$$\mathbf{I}_{\text{diffuse}} = k_d\, \mathbf{M}_d\, \mathbf{I}_d \cos\varphi \qquad \text{if } \varphi \leq \frac{\pi}{2}$$

$$= k_d\, \mathbf{M}_d\, \mathbf{I}_d \max(\langle \mathbf{l} \circ \mathbf{n} \rangle, 0)$$

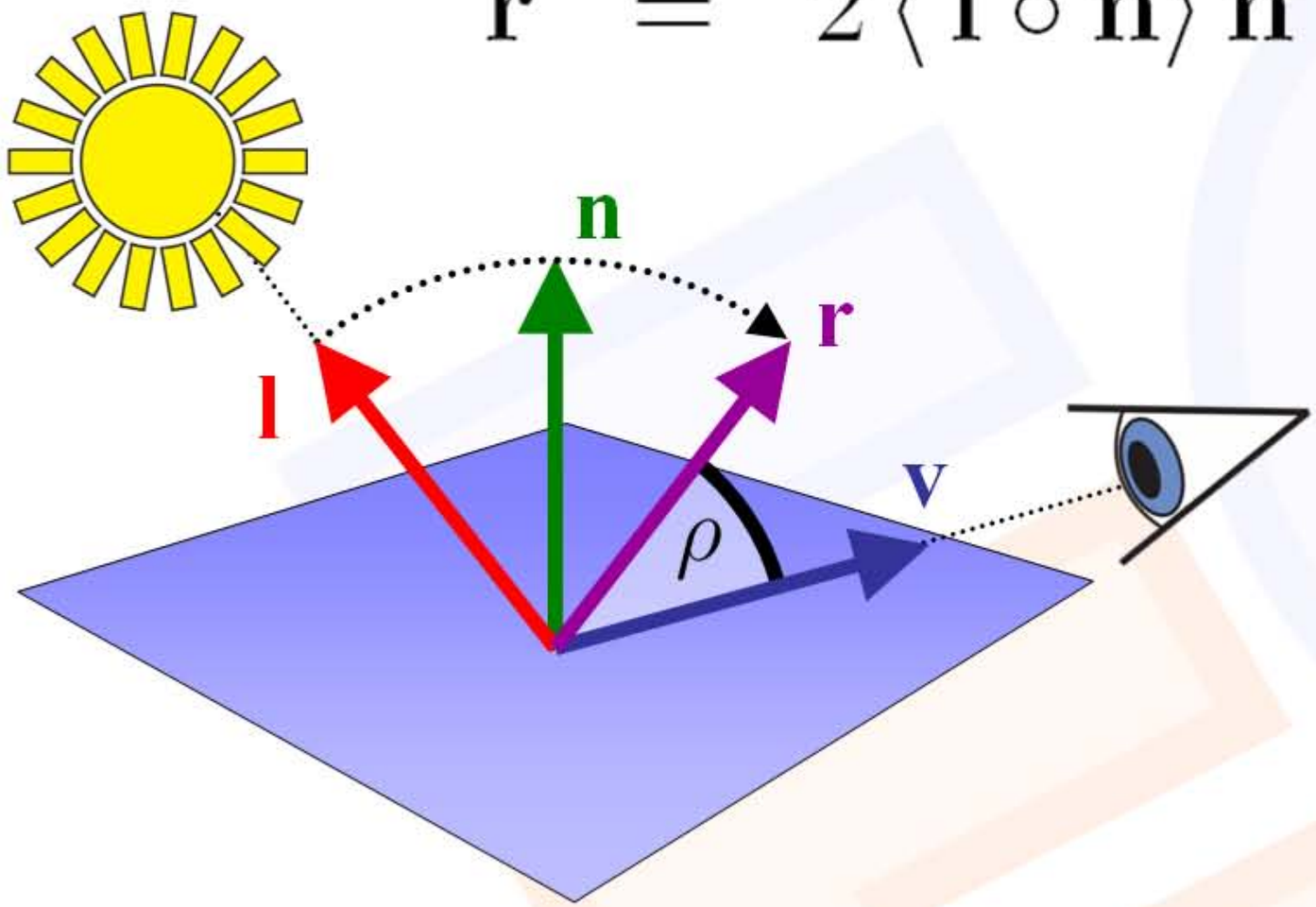


Phong Illumination

- Specular Term (view-dependent)

$$\cos\rho = \langle \mathbf{r} \circ \mathbf{v} \rangle \qquad \text{with}$$

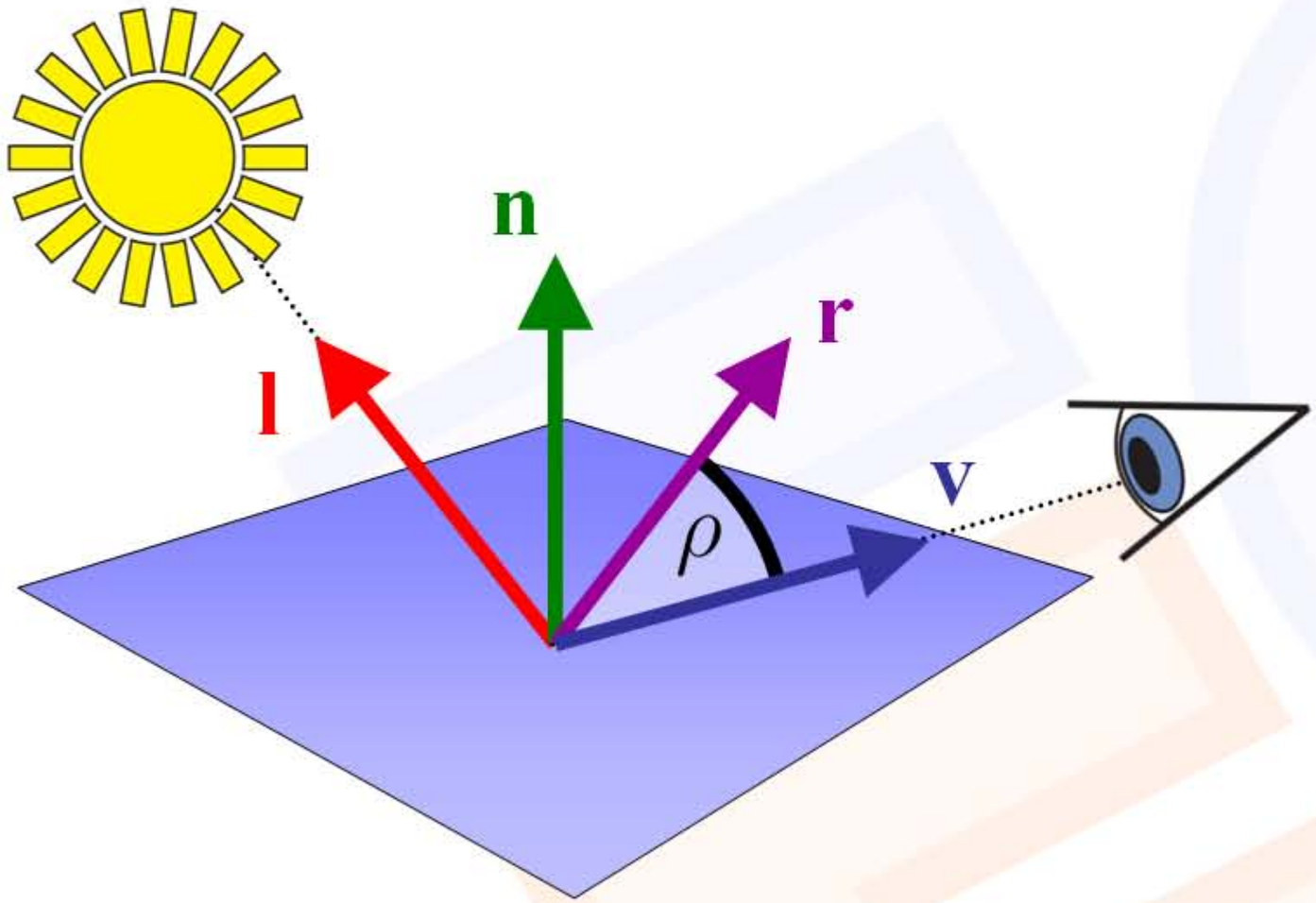
$$\mathbf{r} = 2\langle \mathbf{l} \circ \mathbf{n} \rangle \mathbf{n} - \mathbf{l}.$$



Phong Illumination

- Specular Term (view-dependent)

$$\mathbf{I}_{\text{specular}} = k_s \, \mathbf{M}_s \, \mathbf{I}_s \cos^n \rho \qquad \text{if } \rho \leq \frac{\pi}{2}$$



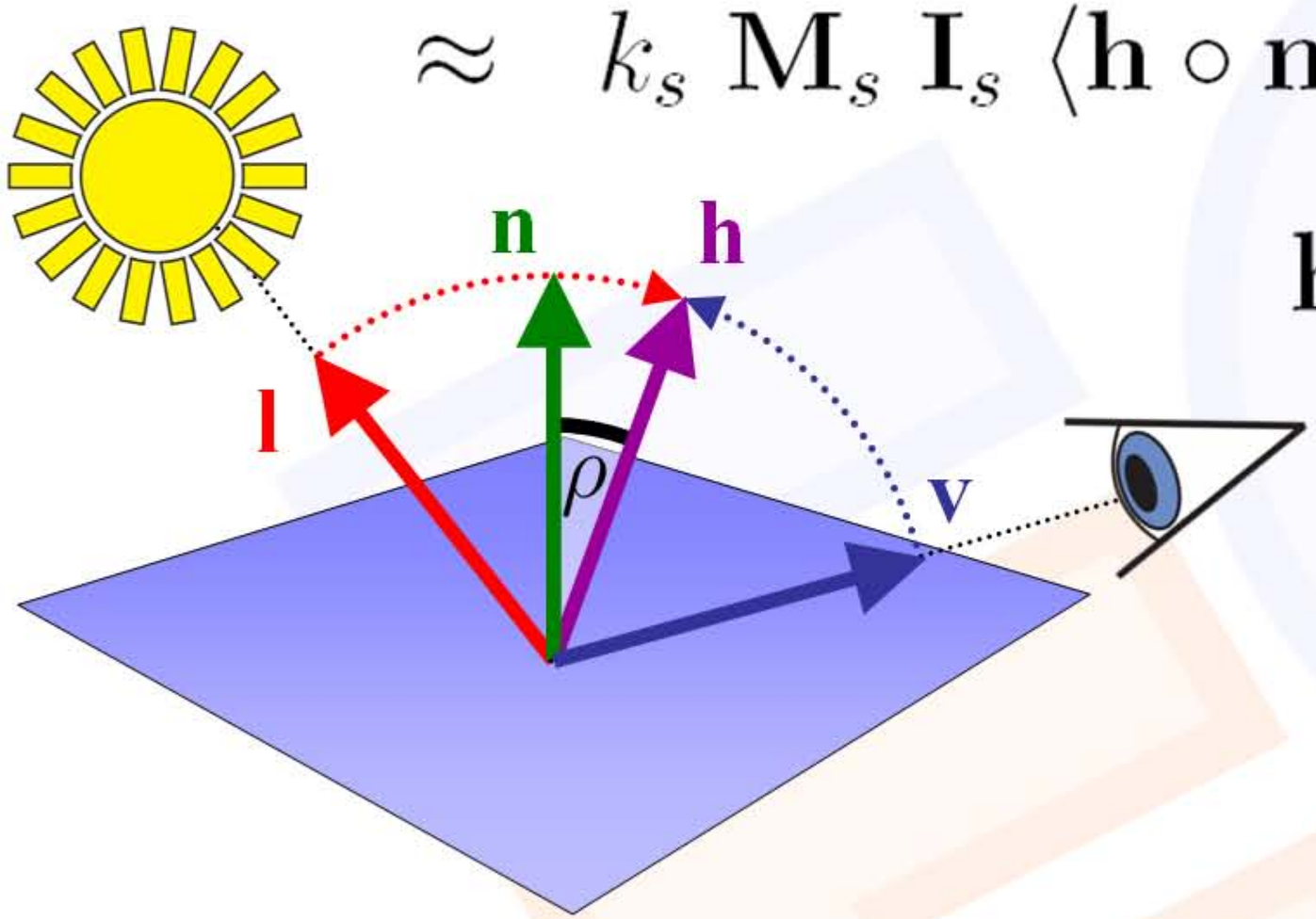
Blinn/Phong Illumination

- Specular Term (view-dependent)

$$\mathbf{I}_{\text{specular}} = k_s\,\mathbf{M}_s\,\mathbf{I}_s\cos^n\rho \qquad \text{if } \rho \le \frac{\pi}{2}$$

$$\approx k_s\,\mathbf{M}_s\,\mathbf{I}_s\,\langle \mathbf{h}\circ\mathbf{n}\rangle^n \qquad \text{with}$$

$$\mathbf{h} = \frac{\mathbf{v}+\mathbf{l}}{\|\mathbf{v}+\mathbf{l}\|}$$



Blinn/Phong Illumination

- Ambient Term (constant illumination)

$$\mathbf{I}_{\text{ambient}} = k_a \, \mathbf{M}_a \, \mathbf{I}_a$$

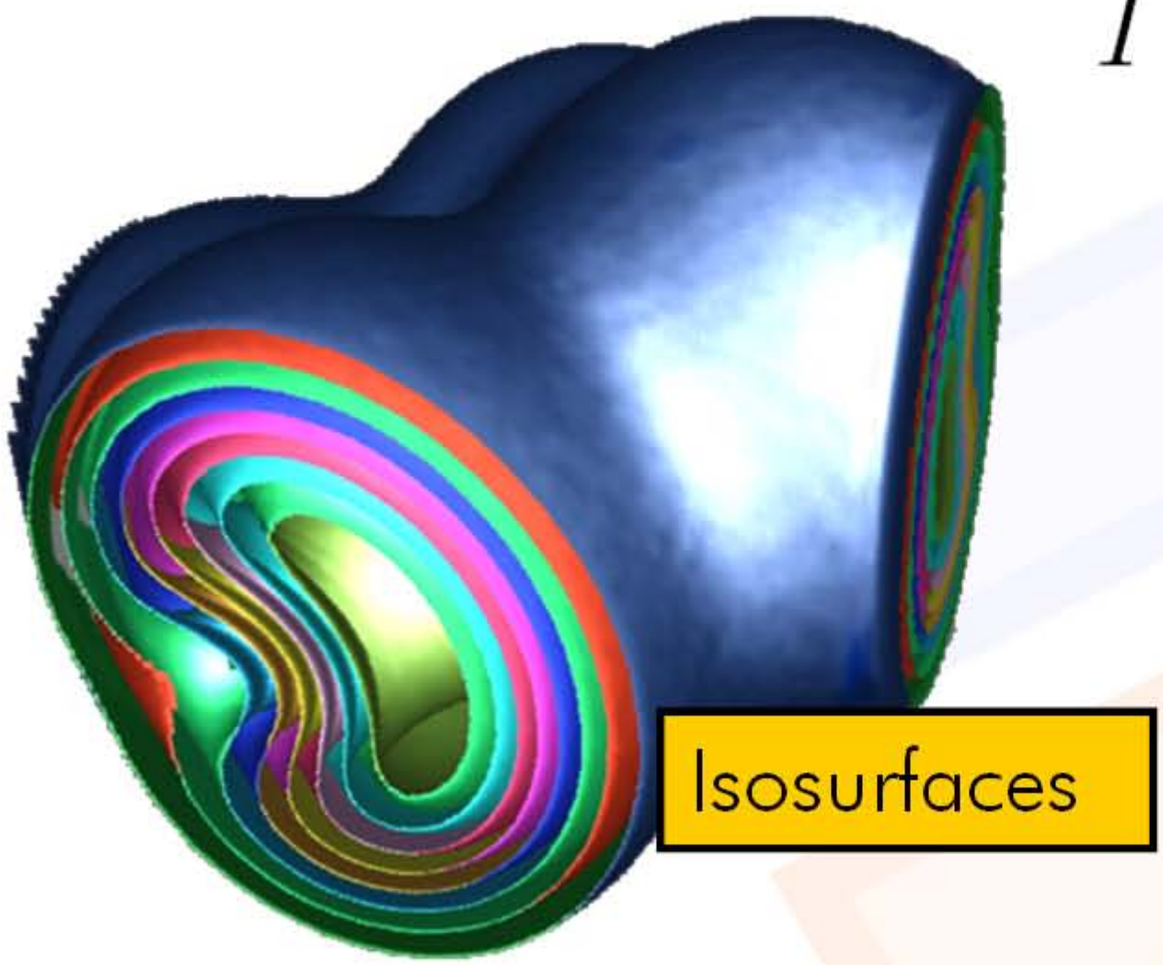
lightens up the shadows, also decreases the contrast!

Consider using a fill light instead of ambient light!

Example images taken from **Jeremy Birn:** *Digital Lighting & Rendering, New Riders Publishing, 2000*

Local Illumination

- Surface lighting: Light is reflected at surfaces
- Volume lighting: Light is scattered at ***isosurfaces***



Isosurfaces

$$I(\mathbf{p}) = \{\, \mathbf{x} \mid f(\mathbf{x}) = f(\mathbf{p}) \}$$

- Isosurface extraction not required!
- We only need the normal vector
- Normal vector of isosurface is equal to (normalized) gradient vector

Gradient Estimation

- The gradient vector is the first-order derivative of the scalar field

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x} \\ \frac{\partial f(\mathbf{x})}{\partial y} \\ \frac{\partial f(\mathbf{x})}{\partial z} \end{pmatrix}$$

partial derivative *in x-direction*

partial derivative *in y-direction*

partial derivative *in z-direction*

- We can estimate the gradient vector using finite differencing schemes

Finite Differences

- Taylor expansion:

$$f(x_0 + h) = f(x_0) + \frac{f'(x_0)}{1!}h + o(h^2)$$

Finite Differences

- Taylor expansion:

$$f(x_0 + h) = f(x_0) + f'(x_0)\,h$$

Finite Differences

- Taylor expansion:

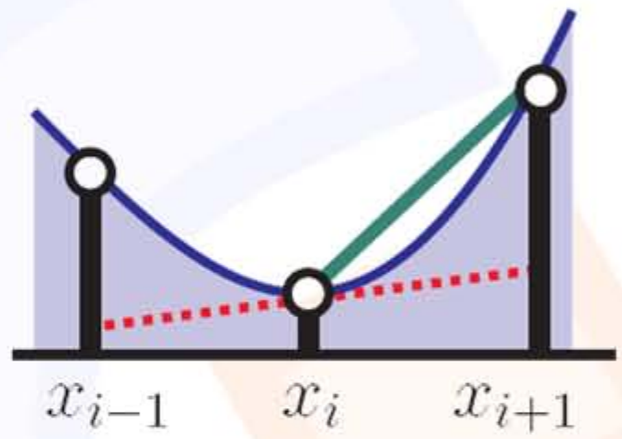
$$\frac{f(x_0 + h) - f(x_0)}{h} = f'(x_0)$$

Finite Differences

- Taylor expansion:

Forward Difference:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}$$

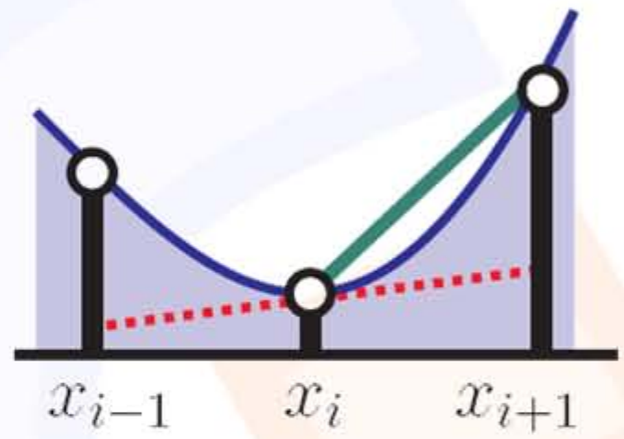


Finite Differences

- Taylor expansion:

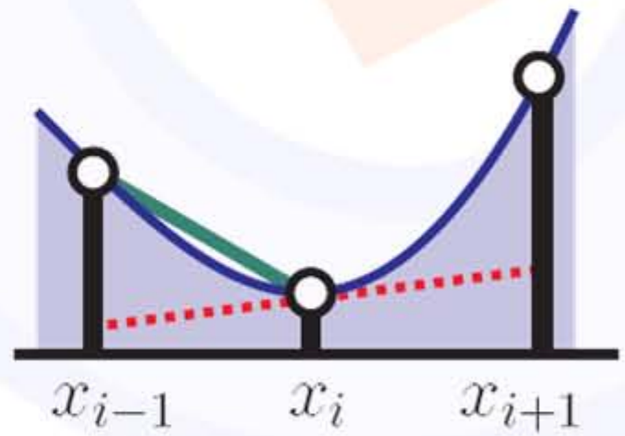
Forward Difference:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}$$



Backward Difference:

$$f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h}$$



Finite Differences

$$f(x_0 + h) = f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f''(x_0)}{2!}h^2 + o(h^3)$$

Finite Differences

$$f(x_0 + h) = f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f''(x_0)}{2!}h^2 + o(h^3)$$

$$f(x_0 - h) = f(x_0) - \frac{f'(x_0)}{1!}h + \frac{f''(x_0)}{2!}h^2 + o(h^3)$$

Finite Differences

$$f(x_0 + h) = f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f''(x_0)}{2!}h^2 + o(h^3)$$

$$f(x_0 - h) = f(x_0) - \frac{f'(x_0)}{1!}h + \frac{f''(x_0)}{2!}h^2 + o(h^3)$$

$$f(x_0 + h) - f(x_0 - h) = 2f'(x_0)h$$

Finite Differences

$$f(x_0 + h) = f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f''(x_0)}{2!}h^2 + o(h^3)$$

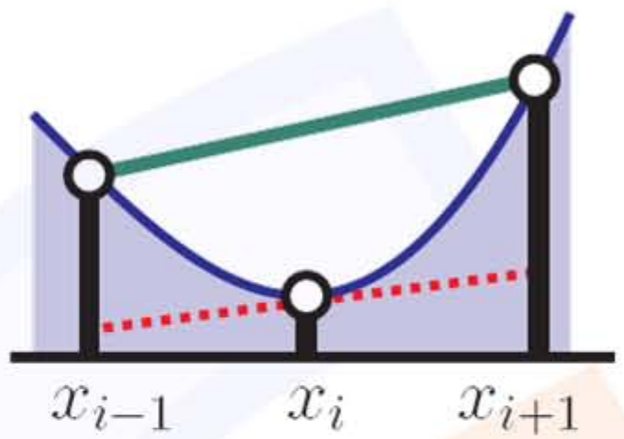
$$f(x_0 - h) = f(x_0) - \frac{f'(x_0)}{1!}h + \frac{f''(x_0)}{2!}h^2 + o(h^3)$$

$$\frac{f(x_0 + h) - f(x_0 - h)}{2h} = f'(x_0)$$

Finite Differences

Central Difference:

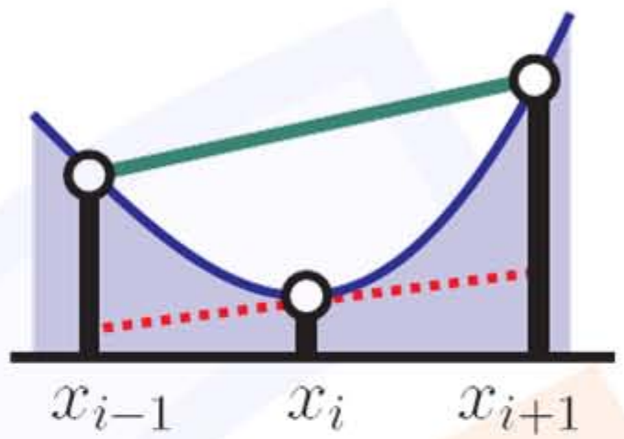
$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h}$$



Finite Differences

Central Difference:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h}$$



Gradient Approximation using Central Differences:

$$\nabla f(x,y,z) \approx \frac{1}{2h} \begin{pmatrix} f(x+h,y,z) - f(x-h,y,z) \\ f(x,y+h,z) - f(x,y-h,z) \\ f(x,y,z+h) - f(x,y,z-h) \end{pmatrix}$$

Pre-computed Gradients

- Calculate the gradient for each voxel
- Store the normalized gradient in an additional texture.

Example: Use an RGB texture:

$$\hat{\nabla} f(\mathbf{x}) = \begin{pmatrix} g_x \\ g_y \\ g_z \end{pmatrix} \begin{matrix} \longrightarrow \\ \longrightarrow \\ \longrightarrow \end{matrix} \begin{matrix} \mathtt{R} = \frac{(g_x+1)}{2} \\ \mathtt{G} = \frac{(g_y+1)}{2} \\ \mathtt{B} = \frac{(g_z+1)}{2} \end{matrix}$$

Pre-computed Gradients

- Calculate the gradient for each voxel
- Store the normalized gradient in an additional texture.

Example: Use an RGBA texture:

$$\hat{\nabla} f(\mathbf{x}) = \begin{pmatrix} g_x \\ g_y \\ g_z \end{pmatrix} \begin{array}{l} \longrightarrow \\ \longrightarrow \\ \longrightarrow \end{array} \begin{array}{l} \mathtt{R} = \frac{(g_x+1)}{2} \\ \mathtt{G} = \frac{(g_y+1)}{2} \\ \mathtt{B} = \frac{(g_z+1)}{2} \end{array}$$

$$f(\mathbf{x}) \longrightarrow \mathtt{A} = f$$

Pre-computed Gradients

```
//fragment program for local illumination and
//post-interpolative transfer function using 3D textures
half4 main (half3 texUV      : TEXCOORD0,
            float3 position :  TEXCOORD1,

            uniform float3 lightPosition,
            uniform float3 eyePosition,
            uniform sampler3D volume_texture,
            uniform sampler1D transfer_function) :  COLOR
{
    float4 sample = tex3D(volume_texture, texUV);

    // expand and normalize the normal vector
    float3 N = normalize(2.0*sample.xyz - 1..xxx);
    // calculate light- and viewing direction
    float3 L = normalize(lightPosition - position);
    float3 V = normalize(eyePosition - position);
```

Pre-computed Gradients

```
uniform sampler1D transfer_function) : COLOR
{
    float4 sample = tex3D(volume_texture, texUV);

    // expand and normalize the normal vector
    float3 N = normalize(2.0*sample.xyz - 1..xxx);
    // calculate light- and viewing direction
    float3 L = normalize(lightPosition - position);
    float3 V = normalize(eyePosition - position);

    // emission and absorption from transfer function
    half4  result  = tex1D(transfer_function, sample.w);

    // add local illumination
    result.rgb += shading(N,V,L);

    return result;
}
```

Pre-computed Gradients

```
//fragment program for local illumination and
//post-interpolative transfer function using 2D multi-textures
half4 main (half3  texUV     : TEXCOORD0,
            float3 position :  TEXCOORD1,

            uniform float3 lightPosition,
            uniform float3 eyePosition,
            uniform sampler2D slice_texture0,
            uniform sampler2D slice_texture1,
            uniform sampler1D transfer_function) :  COLOR
{
    // sample the texture
    float4 sample0 = tex2D(slice_texture0, texUV.xy);
    float4 sample1 = tex2D(slice_texture1, texUV.xy);
    float4 sample  = lerp(sample0, sample1, texUV.z);

    // expand and normalize the normal vector
```

Pre-computed Gradients

```
    float4 sample0 = tex2D(slice_texture0, texUV.xy);
    float4 sample1 = tex2D(slice_texture1, texUV.xy);
    float4 sample  = lerp(sample0, sample1, texUV.z);

    // expand and normalize the normal vector
    float3 N = normalize(2.0*sample.xyz - 1..xxx);
    // calculate light- and viewing direction
    float3 L = normalize(lightPosition - position);
    float3 V = normalize(eyePosition - position);

    // emission and absorption from transfer function
    half4  result  = tex1D(transfer_function, sample.w);

    // add local illumination
    result.rgb += shading(N,V,L);

    return result;
}
```

Non-Polygonal Isosurfaces

```
half4 main (half3 texUV      : TEXCOORD0,
            float3 position :  TEXCOORD1,

            uniform sampler3D volume_texture) :  COLOR
{
   float4 sample = tex3D(volume_texture, texUV);

   // expand and normalize the normal vector
   float3 N = normalize(2.0*sample.xyz - 1..xxx);
   // calculate light- and viewing direction
   float3 L = normalize(lightPosition - position);
   float3 V = normalize(eyePosition - position);

   half4  result;
   result.rgb = shading(N,V,L);
   result.a = sample.a;

   return result;
}
```

Non-Polygonal Isosurfaces

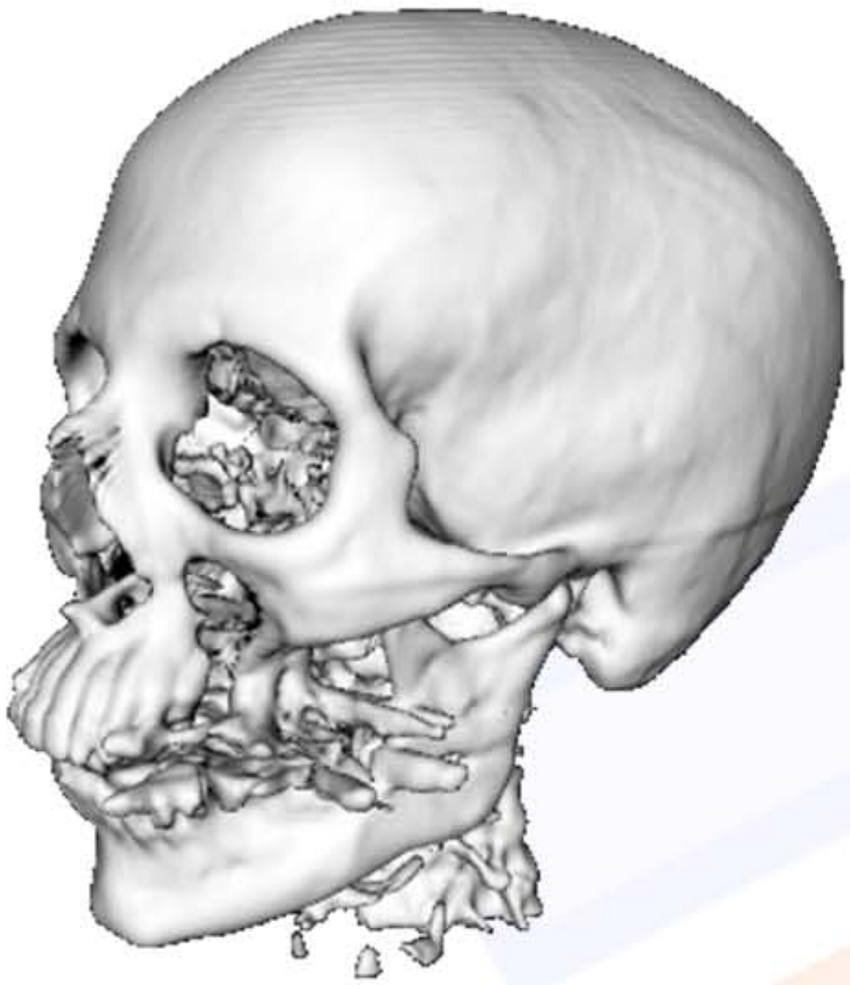
- Replace Blending by Alpha Test

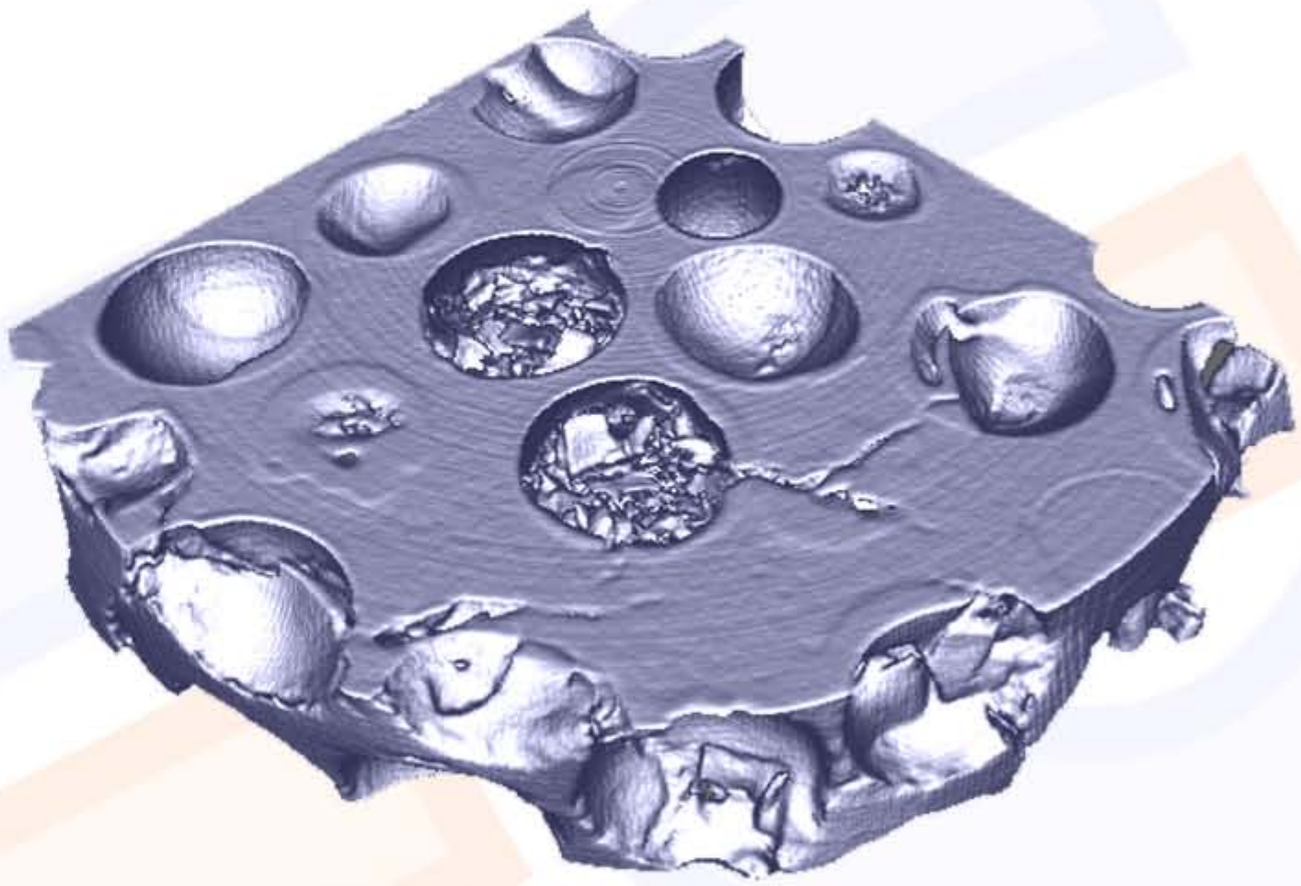
```
// disable alpha blending
glDisable(GL_BLEND);

// enable alpha test for isosurface
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_LESS, fIsoValue); // or GL_GREATER
```

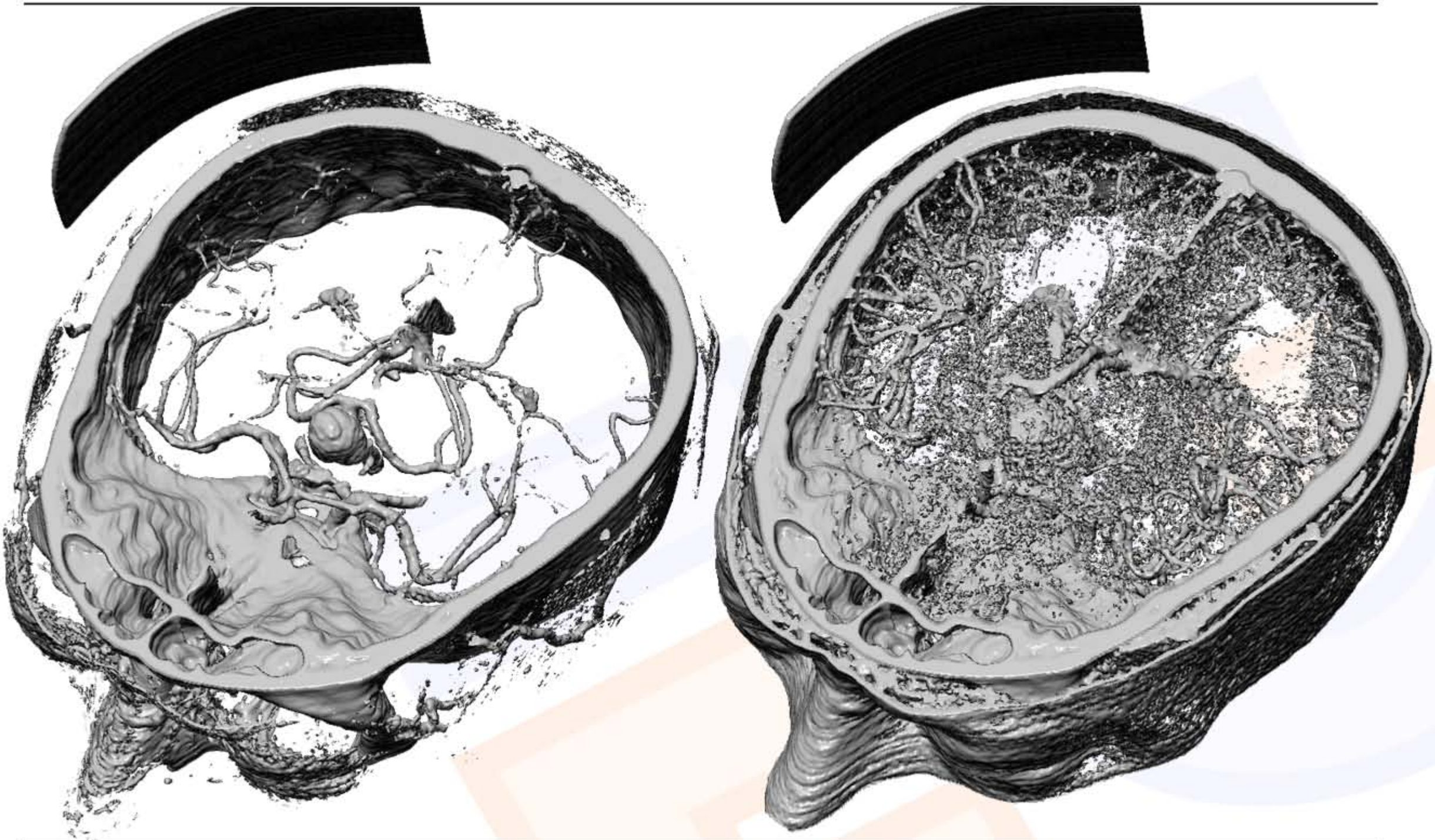
Note: Make sure to render the slices front-to-back in order to exploit the early z-test!

Non-Polygonal Isosurfaces





Non-polygonal Isosurfaces



Pre-Computed Gradients

- Drawbacks of pre-computed gradients:
  Memory Requirements
- Gradients must be stored at least at 3x 8bit
  Texture compression will reduce image quality

- Not applicable to large volume data

On-the-fly Gradient Estimation

$$\nabla f(x,y,z) \approx \frac{1}{2h} \begin{pmatrix} f(x+h,y,z) - f(x-h,y,z) \\ f(x,y+h,z) - f(x,y-h,z) \\ f(x,y,z+h) - f(x,y,z-h) \end{pmatrix}$$

```
float3 sample1, sample2;
// six texture samples for the gradient
sample1.x = tex3D(texture,uvw-half3(DELTA,0.0,0.0)).x;
sample2.x = tex3D(texture,uvw+half3(DELTA,0.0,0.0)).x;
sample1.y = tex3D(texture,uvw-half3(0.0,DELTA,0.0)).x;
sample2.y = tex3D(texture,uvw+half3(0.0,DELTA,0.0)).x;
sample1.z = tex3D(texture,uvw-half3(0.0,0.0,DELTA)).x;
sample2.z = tex3D(texture,uvw+half3(0.0,0.0,DELTA)).x;
// central difference and normalization
float3 N = normalize(sample2-sample1);
```

On-the-fly Gradient Estimation

Drawbacks:

- 3D texture required
- Each additional texture sample is expensive!
  - *Central Differences:* 7 Texture Samples
  - *Forward/Backward Differences:* 4 Texture Samples

Advantages:

- Low memory requirements
- Gradient estimation at floating point precision!
- Gradient estimation can be omitted in FP (using a conditional branch)

On-the-fly Gradient Estimation

Drawbacks:

- 3D texture required
- Each additional texture sample is expensive!
  - *Central Differences:* 7 Texture Samples
  - *Forward/Backward Differences:* 4 Texture Samples

Advantages:

- Low memory requirements
- Gradient estimation at floating point precision!
- Gradient estimation can be omitted in FP (using a conditional branch)

On-the-Fly Directional Derivatives

- Illumination calculation usually involves dot products, such as $\langle \mathbf{n} \circ \mathbf{l} \rangle$
- If we neglect the normalization $\mathbf{n} = \hat{\nabla} f$, we can write the dot product as a directional derivative,

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{l}} = \langle \nabla f(\mathbf{x}) \circ \mathbf{l} \rangle .$$

- Directional derivatives can be approximated directly using finite differences.

On-the-Fly Directional Derivatives

- Forward Difference

$$\langle \nabla f(\mathbf{x}) \circ \mathbf{l} \rangle \approx \frac{f(\mathbf{x} + h\mathbf{l}) - f(\mathbf{x})}{h}$$

- Backward Difference

$$\langle \nabla f(\mathbf{x}) \circ \mathbf{l} \rangle \approx \frac{f(\mathbf{x}) - f(\mathbf{x} - h\mathbf{l})}{h}$$

- Central Difference

$$\langle \nabla f(\mathbf{x}) \circ \mathbf{l} \rangle \approx \frac{f(\mathbf{x} + h\mathbf{l}) - f(\mathbf{x} - h\mathbf{l})}{2h}$$

Environment Mapping

Reflection Map

Perfect mirror reflection

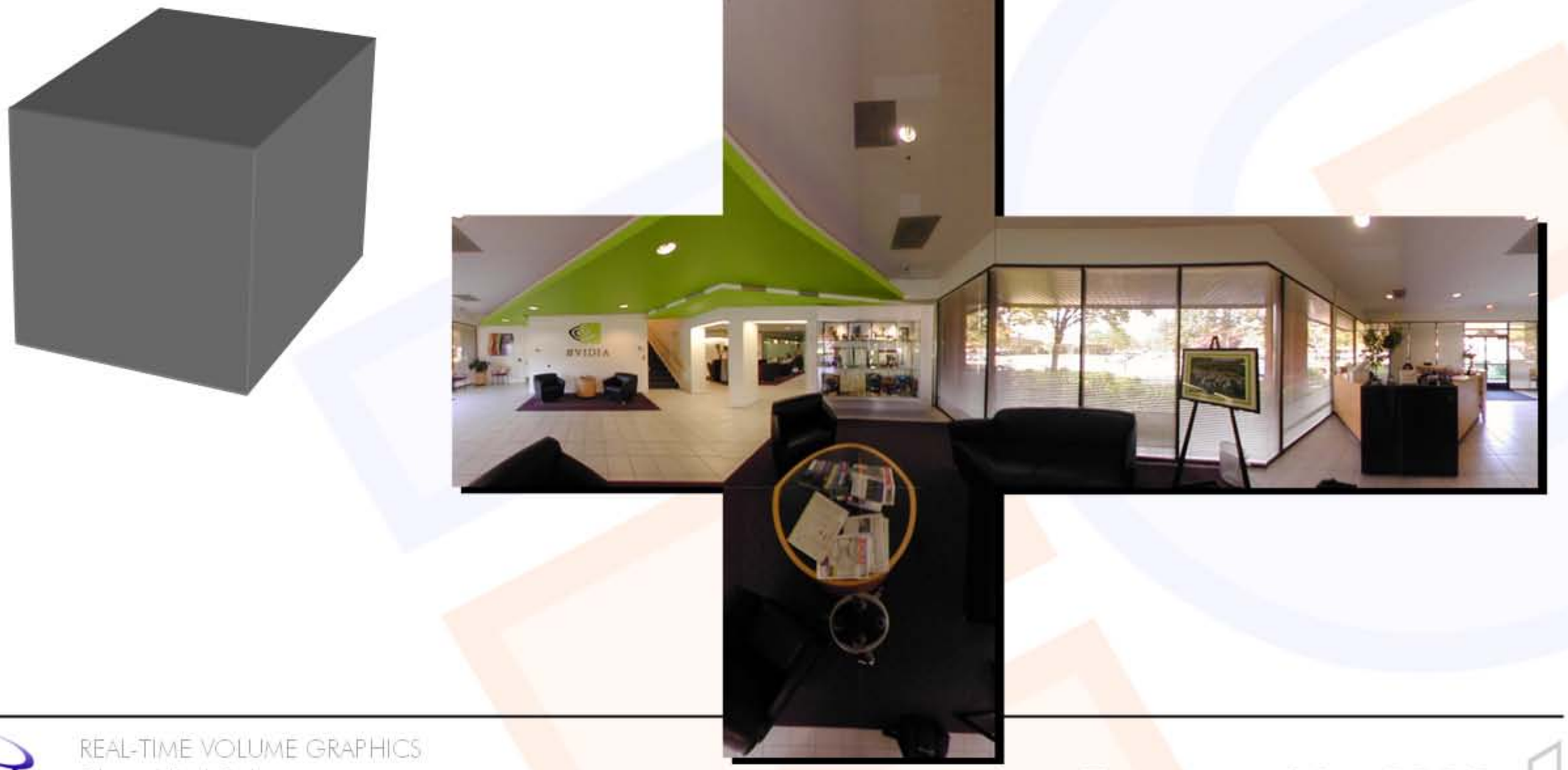
Irradiance Map

Lambertian reflection

Image courtesy of Paul Debevec (www.debevec.org)

Environment Cube Maps

- Use the 6 sides of a cubes to store the light environment.



Environment Mapping

```
//irradiance and reflection mapping with
//post-classification using 3D textures
half4 main (half3 texUV      : TEXCOORD0,
            float3 position :  TEXCOORD1,

            uniform float3 Kd, // diffuse,
            uniform float3 Ks, // specular
            uniform float3 eyePosition,
            uniform sampler3D volume_texture,
            uniform sampler1D transfer_function,
            uniform samplerCUBE irradiance_map,
            uniform samplerCUBE reflection_map) :  COLOR
{
   float4 sample = tex3D(volume_texture, texUV);

   // expand and normalize the normal vector
   float3 N = normalize(2.0*sample.xyz - 1..xxx);
   // calculate viewing- and reflection vector
   float3 V = normalize(eyePosition - position);
```

Environment Mapping

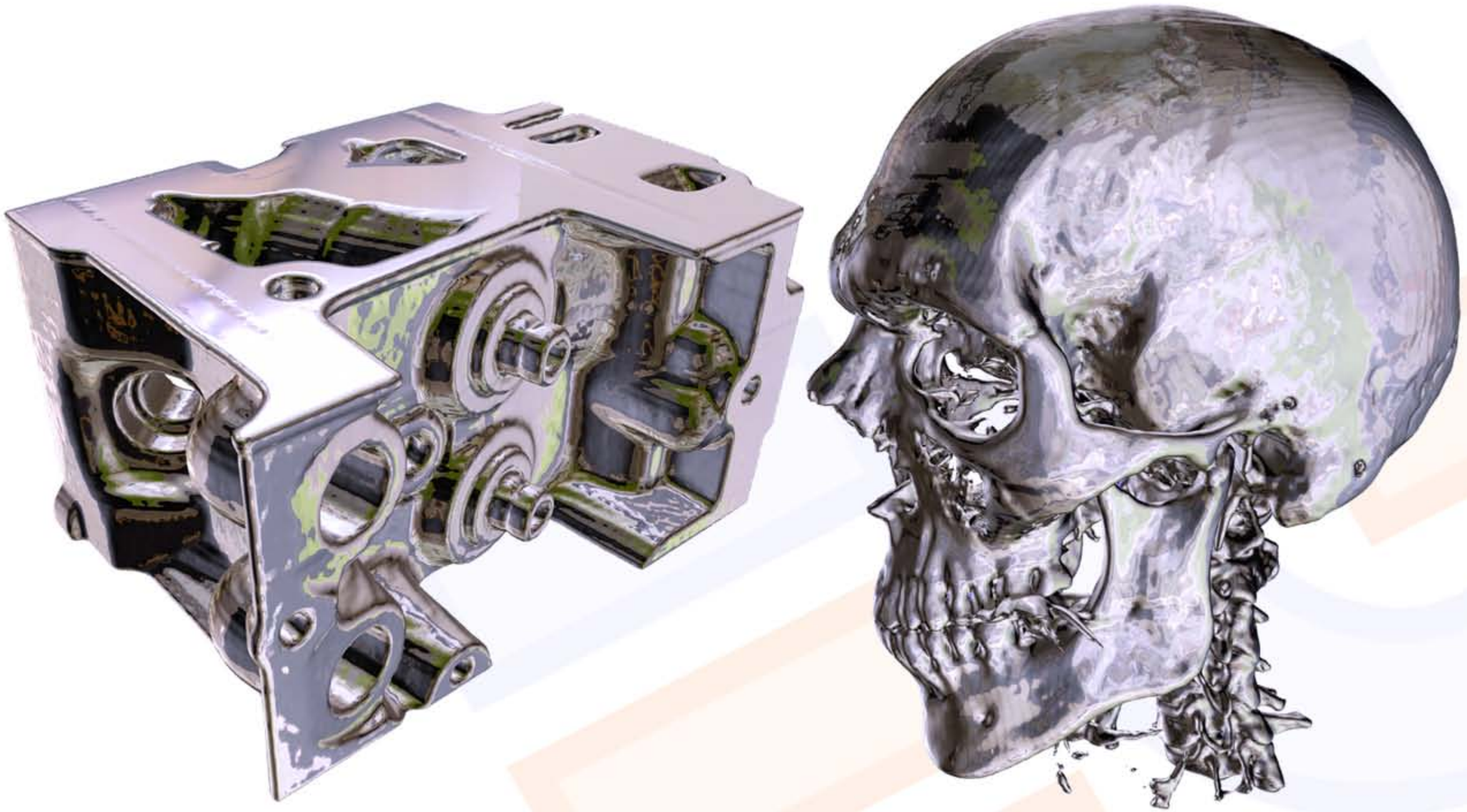
```
{
    float4 sample = tex3D(volume_texture, texUV);

    // expand and normalize the normal vector
    float3 N = normalize(2.0*sample.xyz - 1..xxx);
    // calculate viewing- and reflection vector
    float3 V = normalize(eyePosition - position);
    float3 R = reflect(V,N);
    // sample irradiance map (normal direction)
    float3 diffuse = Kd * texCUBE(irradiance_map,N);
    // sample reflection map (mirrored viewing direction)
    float3 specular = Ks * texCUBE(reflection_map,R);

    // emission and absorption from transfer function
    half4  result  = tex1D(transfer_function, sample.w);
    // add illumination
    result.rgb += diffuse + specular;

    return result;
}
```

Environment Mapping



Summary

- Single Scattering/Local Illumination
- Gradient Estimation
  - Finite Differences
- Gradient-based Illumination
  - Pre-computed Gradients
  - On-the-fly gradient estimation
  - On-the-fly directional derivatives
- Blinn/Phong Illumination
- Environment Mapping
  - (Reflection Map and Irradiance Map)