Eurographics 2009

Light Interactions

Markus Hadwiger
VR VIS Research Center
Vienna, Austria

Patric Ljung
Siemens Corporate Research
Princeton, NJ, USA

Christof Rezk Salama
Computer Graphics Group
Institute for Vision and Graphics
University of Siegen, Germany

Timo Ropinski
Visualization and Computer
Graphics Research Group,
University of Münster, Germany

Local vs. Global Illumination

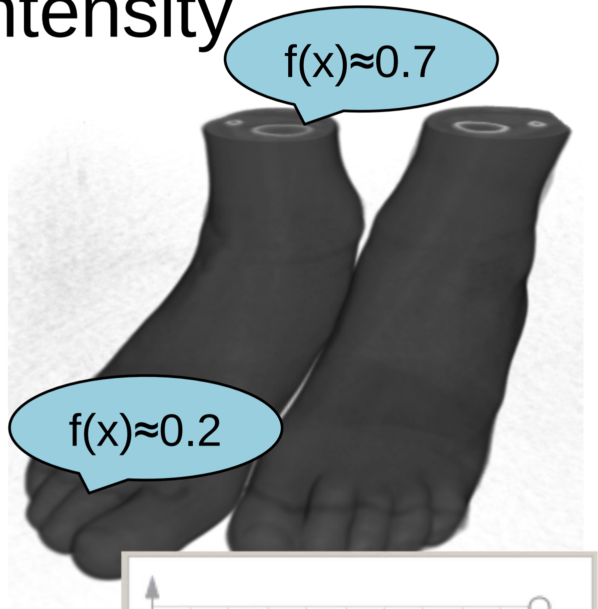
- Local illumination
  - Consider current object only
  - *O(n)* runtime complexity
- Global illumination
  - Consider all scene objects
  - $O(n^2)$ runtime complexity
- Local illumination does not allow to incorporate shadows and reflections

[Ikits et al., GPU Gems 2004]

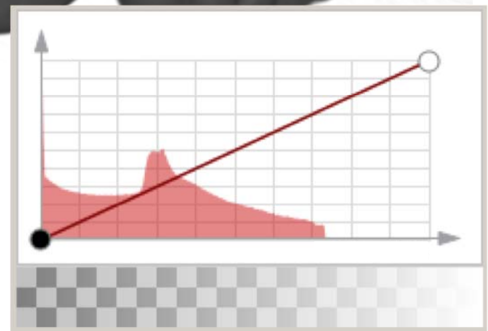
Phong Illumination Model

…in a nutshell

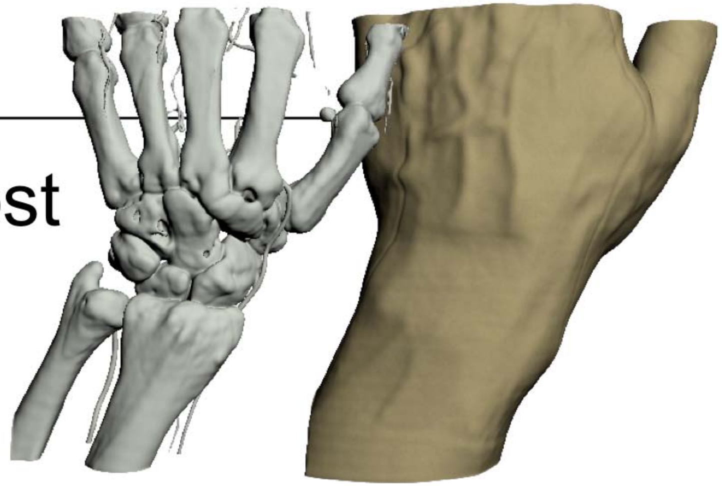
- Assume data set is given as an intensity function **f(x)**



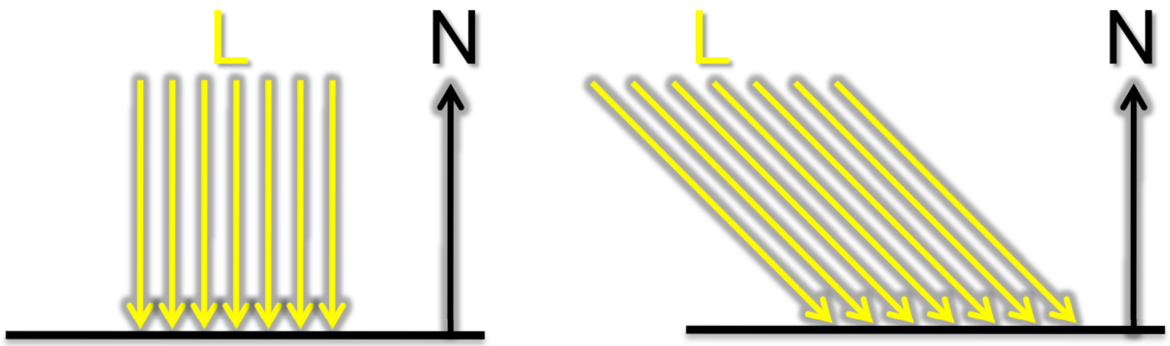
- Parameters used for illumination
  - **Position** of the current voxel x
  - Voxel color as assigned through the **transfer function**
  - Current voxel's **gradient**
  - Position of the **light source**



Diffuse Lighting



- Diffuse lighting effects are most prominent in volume data
- Light calculation is based on Lambert‘s law

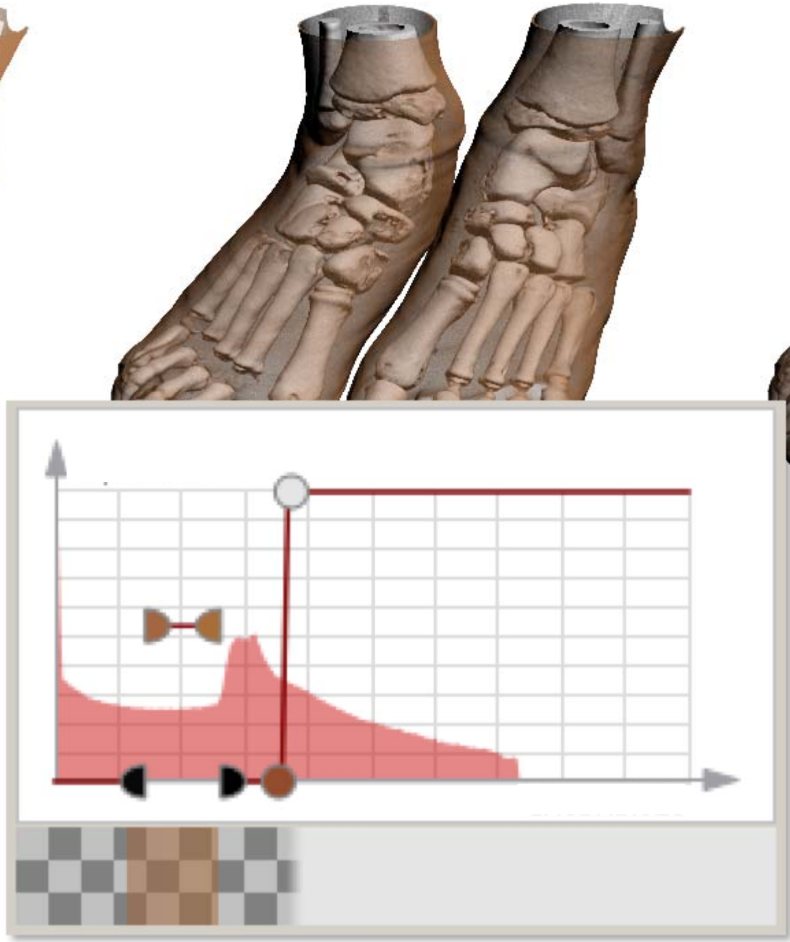


- Ratio can be expressed by dot product

$$I_d(x) = \boxed{L_{d,in}} \cdot \boxed{k_d} \cdot \boxed{max(| \bigtriangledown \tau(f(x))| \cdot L, 0)}$$

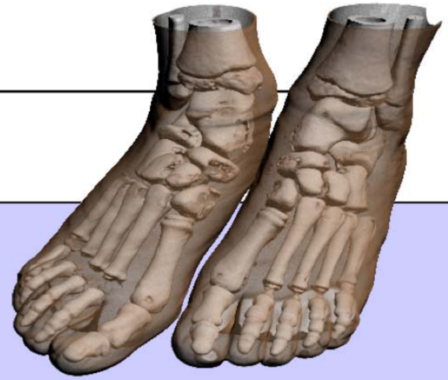
Diffuse Lighting

no shading



diffuse lighting
dark

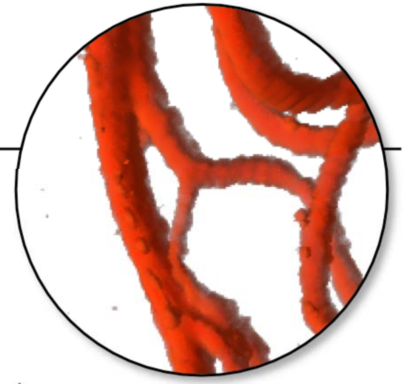
Diffuse Lighting



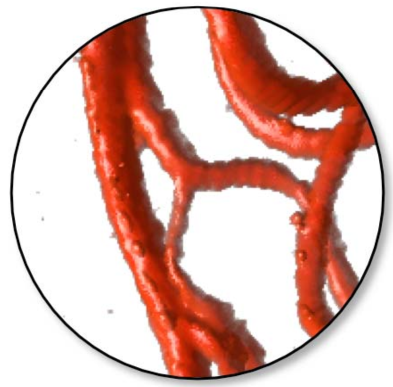
```
/**
 * Returns the diffuse term, considering the
 * currently set OpenGL lighting parameters.
 *
 * @param kd The diffuse color to be used.
 * Usually this is fetched from the transfer
 * function.
 * @param G The computed gradient.
 * @param L The normalized light vector.
 */
vec3 getDiffuseColor(in vec3 kd, in vec3 G, in vec3 L) {
    float GdotL = max(dot(G, L), 0.0);
    return kd * lightParams.diffuse.rgb * GdotL;
}
```

$$I_d(x) = L_{d,in} \cdot k_d \cdot max(| \bigtriangledown \tau(f(x))| \cdot L, 0)$$

Specular Lighting

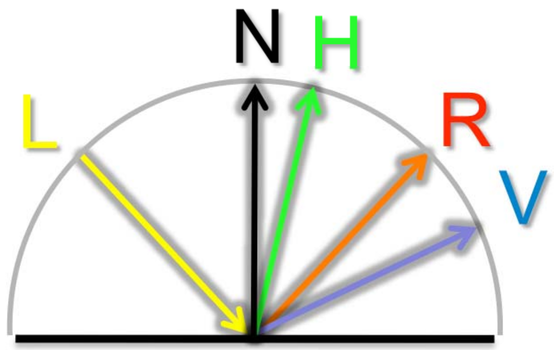


diffuse



diffuse + specular

- Specular highlights can add realism to certain tissues
- Lighting calculation is view dependent

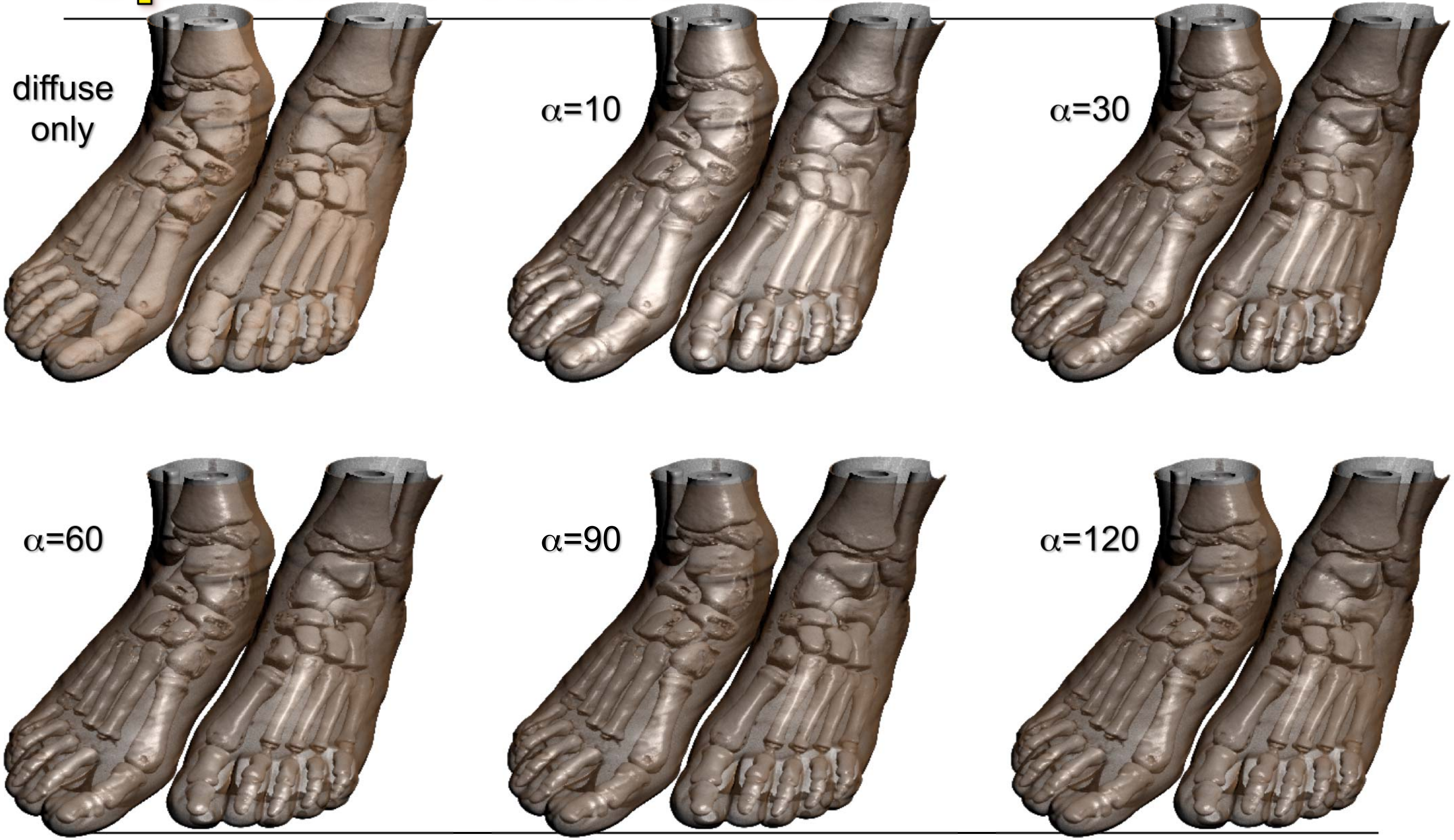


- Can be expre ;

$$I_s(x) = \boxed{L_{s,in}} \cdot \boxed{k_s} \cdot \boxed{max(|\bigtriangledown \tau(f(x))| \cdot H, 0)^{\alpha}}$$

$$H = \frac{V+L}{2}$$

Specular Reflections



Specular Lighting

```
/**
 * Returns the specular term, considering the
 * currently set OpenGL lighting parameters.
 *
 * @param ks The specular color to be used.
 * @param G The computed gradient.
 * @param L The normalized light vector.
 * @param V The normalized view vector.
 */
vec3 getSpecularColor(in vec3 ks, in vec3 N, in vec3 L, in vec3 V) {
    vec3 H = normalize(V + L);
    float GdotH = pow(max(dot(G, H), 0.0), matParams.shininess);
    return ks * lightParams.specular.rgb * GdotH;
}
```

$$I_s(x) = L_{s,in} \cdot k_s \cdot max(|\bigtriangledown \tau(f(x))| \cdot H, 0)^{\alpha}$$

Ambient Lighting

- Add constant light in shadowed regions

$$I_a(x) = L_{a,in} \cdot k_a$$

```
/**
 * Returns the ambient term, considering the
 * currently set OpenGL lighting parameters.
 *
 * @param ka The ambient color to be used.
 * Usually this is fetched from the transfer
 * function.
 */
vec3 getAmbientColor(in vec3 ka) {
    return ka * lightParams.ambient.rgb;
}
```

- Drawback: contrast reduction

Ambient Lighting

ambient dark +
diffuse + specular

ambient medium+
diffuse + specular

ambient bright+
diffuse + specular

Phong Lighting

```
/**
 * Calculates Phong shading.
 *
 * @param G The gradient given in volume object space (does not need to be
normalized).
 * @param vpos The voxel position given in volume texture space.
 * @param kd The diffuse material color to be used.
 * @param ks The specular material color to be used.
 * @param ka The ambient material color to be used.
 */
vec3 phongShading(in vec3 G, in vec3 vpos, in vec3 kd, in vec3 ks, in vec3 ka) {

    vec3 L = normalize(lightPosition - vpos);
    vec3 V = normalize(cameraPosition - vpos);

    vec3 shadedColor = vec3(0.0);
    shadedColor += getDiffuseColor(kd, normalize(G), L);
    shadedColor += getSpecularColor(ks, normalize(G), L, V);
    shadedColor += getAmbientColor(ka);

    return shadedColor;
}
```

Adding Attenuation

```
shadedColor *= getAttenuation(d);
```

```
/**
 * Returns attenuation based on the currently
 * set OpenGL values. Incorporates constant,
 * linear and quadratic attenuation.
 *
 * @param d Distance to the light source.
 */
float getAttenuation(in float d) {
    return 1.0 / (lightParams.constantAttenuation +
                  lightParams.linearAttenuation * d +
                  lightParams.quadraticAttenuation * d * d);
}
```

Phong Shading + Attenuation

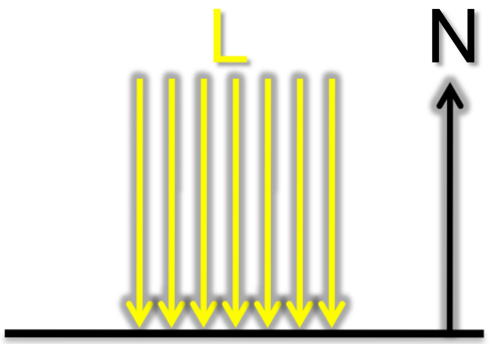
no shading

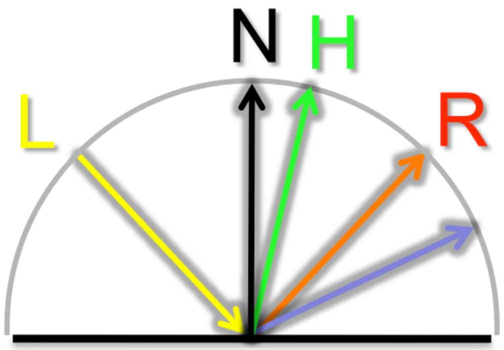
Phong shading

Phong shading
and attenuation

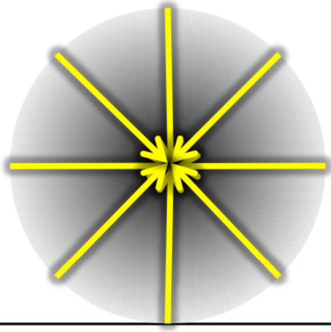
Gradient Calculation

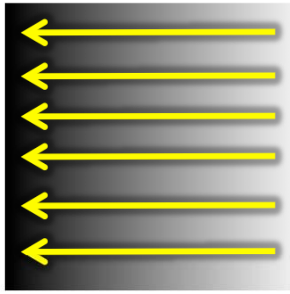
- Surface normal is required for diffuse and specular illumination





- The gradient is a good approximation for a surface normal





[Levoy, CG&A 1988]

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***

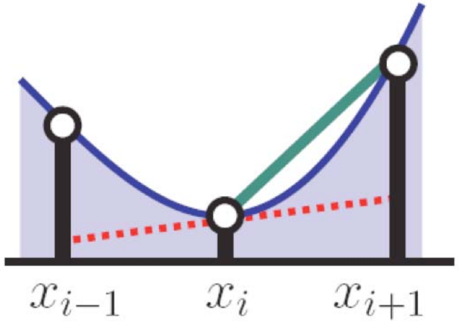
partial derivative ***in z-direction***

- We can estimate the gradient vector using finite differencing schemes
  - Forward/backward differences
  - Central differences

Back-/Forward Differences

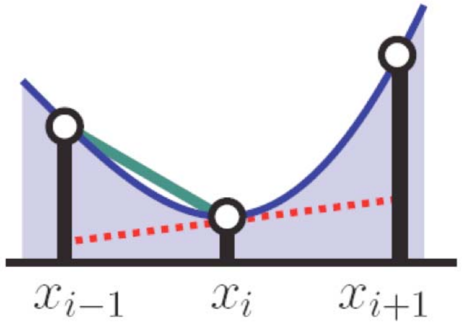
- Forward differences

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



- Backward differences

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



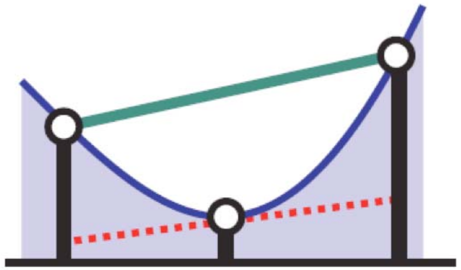
Forward Differences

```
/**
 * Calculate the gradient based on the A channel
 * using forward differences.
 */
vec3 calcGradient(sampler3D volume, vec3 voxPos, float t, vec3 dir) {
    vec3 gradient;

    float v = texture1D(transferFunc_, textureLookup3D(volume, volumeParameters,
voxPos).a);
    float v0 = texture1D(transferFunc_, textureLookup3D(volume, volumeParameters,
voxPos + vec3(offset.x, 0.0, 0.0)).a);
    float v1 = texture1D(transferFunc_, textureLookup3D(volume, volumeParameters,
voxPos + vec3(0, offset.y, 0)).a);
    float v2 = texture1D(transferFunc_, textureLookup3D(volume, volumeParameters,
voxPos + vec3(0, 0, offset.z)).a);

    gradient = vec3(v - v0, v - v1, v - v2);
    return gradient;
}
```

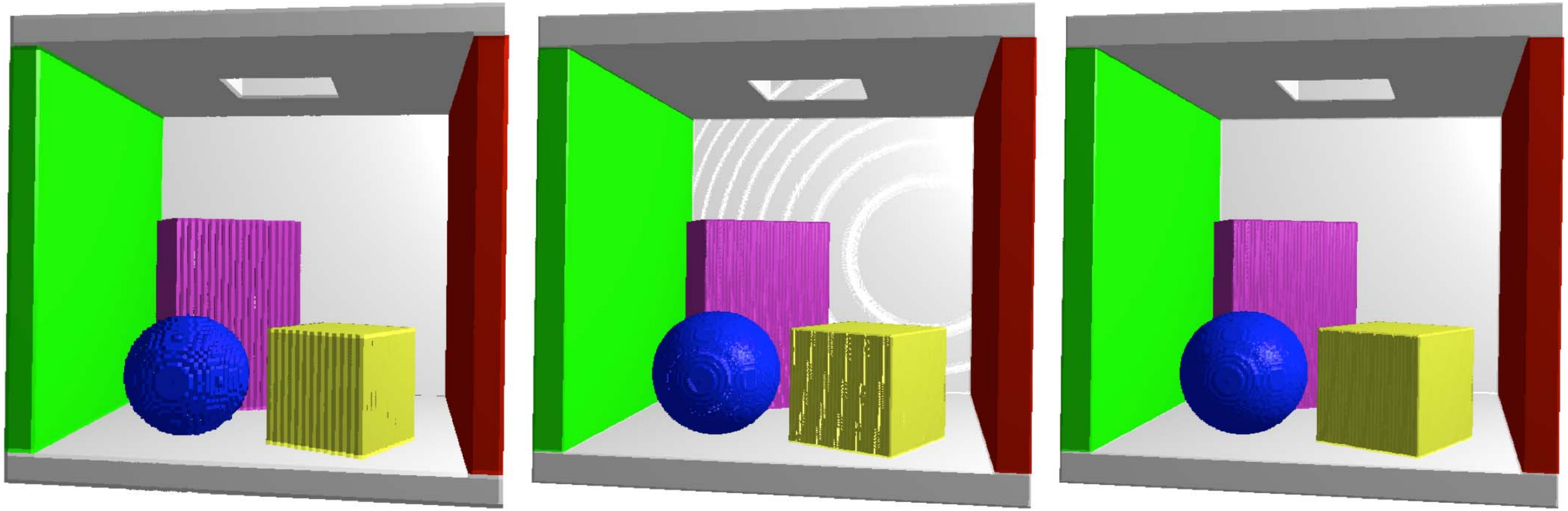
Central Differences



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

Gradient Quality

- Gradient quality is crucial
- Effects are especially visible when rendering binary volumes

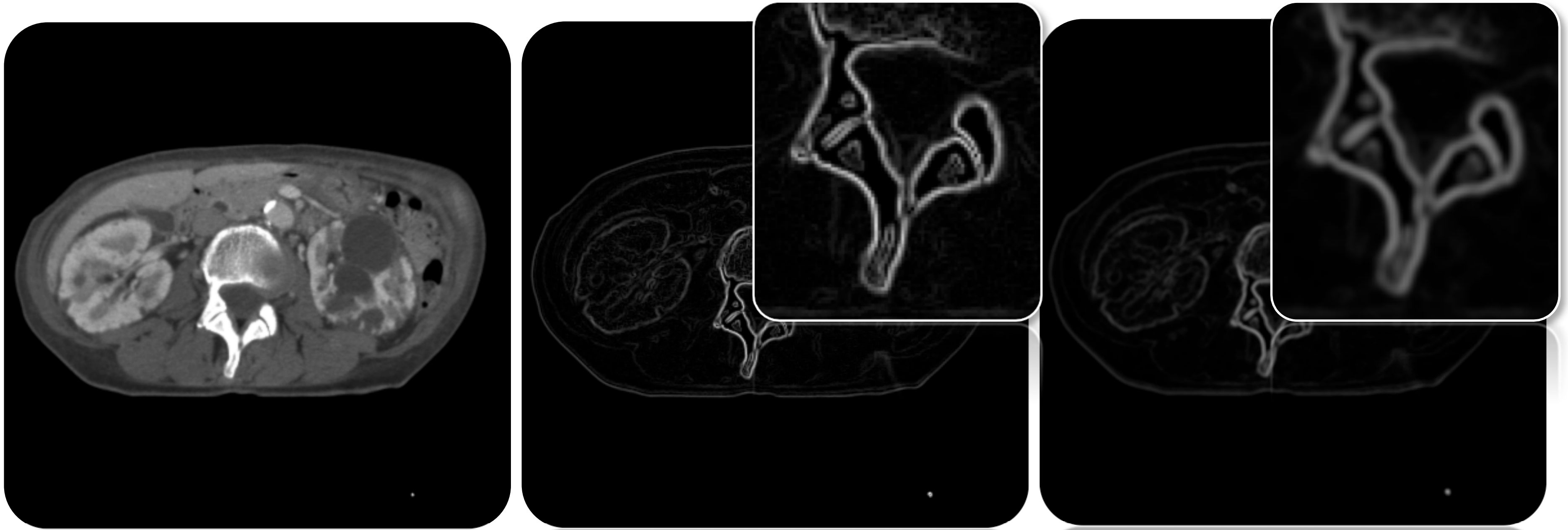


forward | central | central filtered

Sobel Gradients

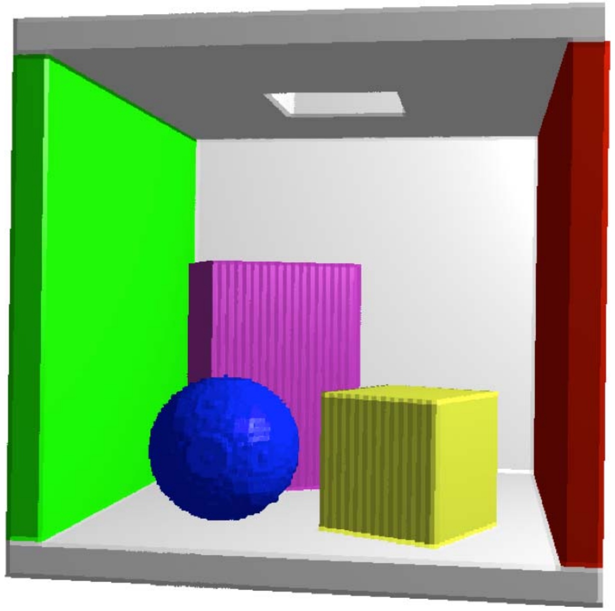
- Alternatively Sobel operator can be used

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

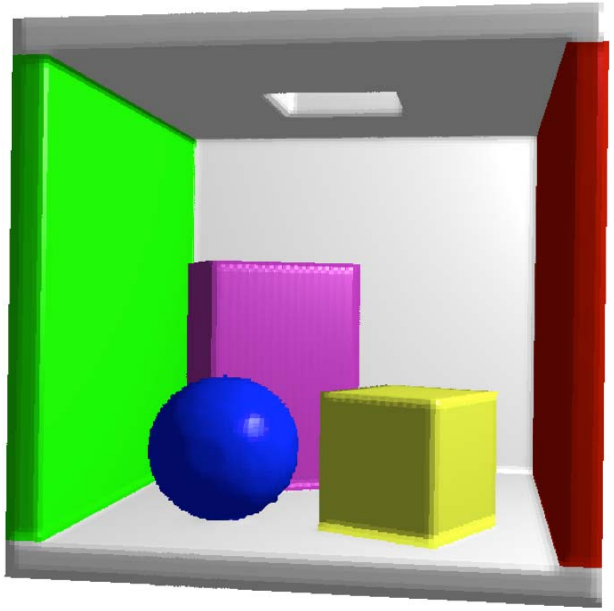


Intensity image | Sobel gradient | Sobel gradient filtered

Sobel Gradients



Sobel



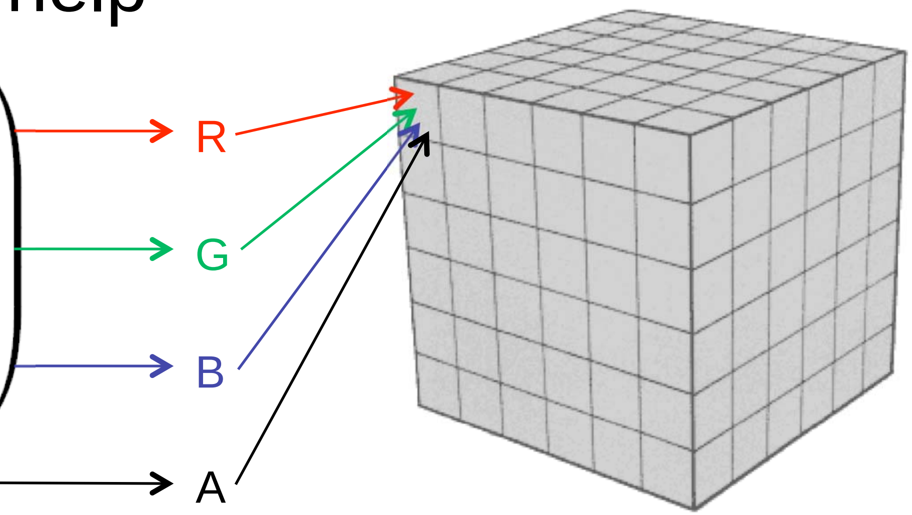
Sobel filtered

Sobel Precalculation

- Sobel filter requires 26(!) additional texture fetches
- Memory access has major performance impact
- Precomputation can help

$$\nabla f(\mathbf{x}) = \begin{pmatrix} (\frac{\partial f(\mathbf{x})}{\partial x} + 1) / 2 \\ (\frac{\partial f(\mathbf{x})}{\partial y} + 1) / 2 \\ (\frac{\partial f(\mathbf{x})}{\partial z} + 1) / 2 \end{pmatrix}$$

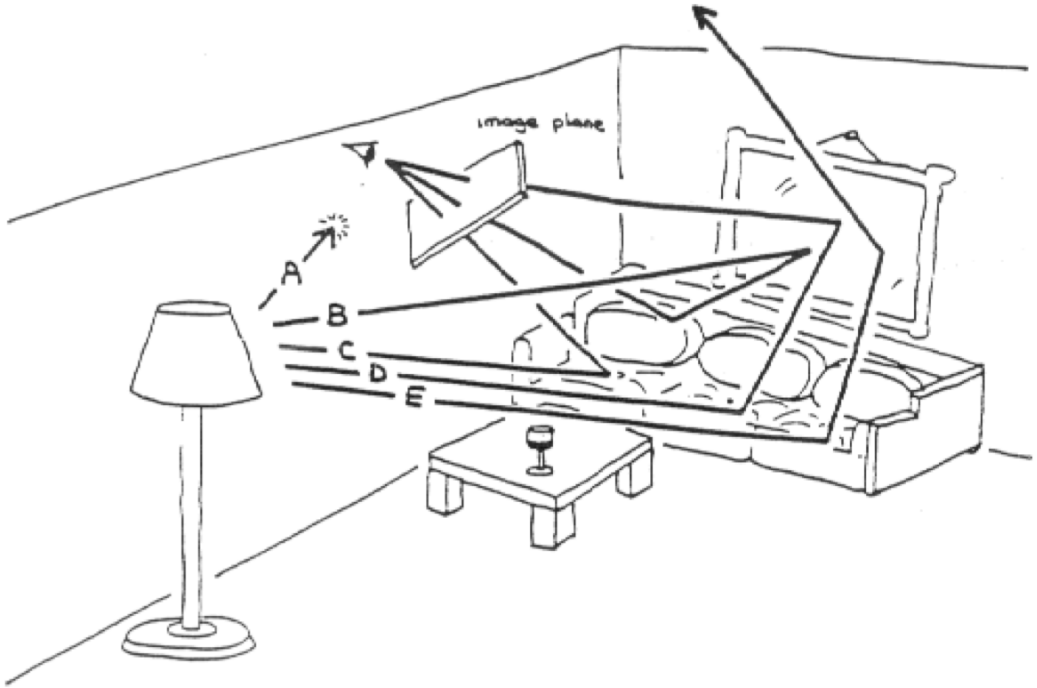
$$f(\mathbf{x})$$



4x8=32 bit data set

Ray Tracing Effects

- Use ray tracing to add *globalism*

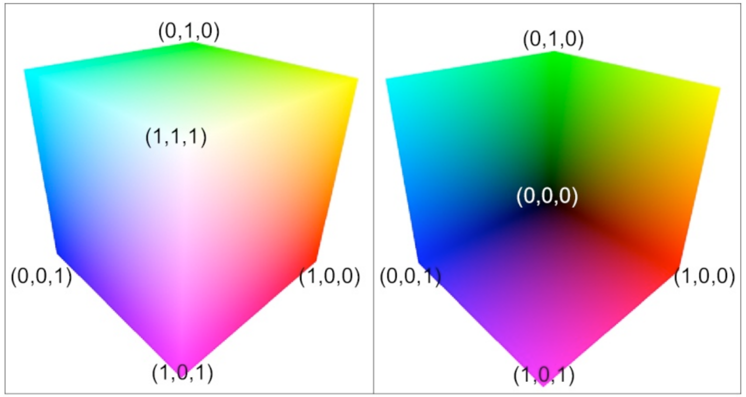


[Glassner: An introduction to ray tracing]

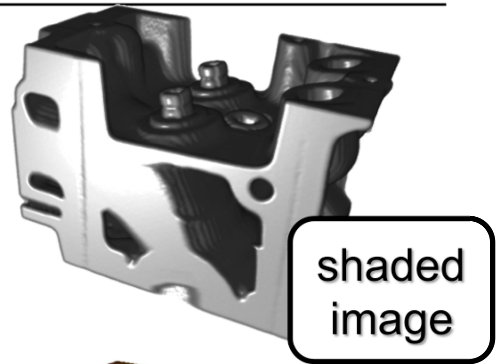
[Stegmayer et al., VG 2005]

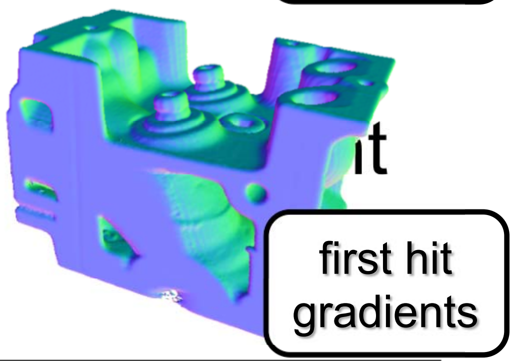
Ray Tracing: Input

- To trace rays, we require
  - Intersection points
  - Gradients at intersection points
  - Material properties

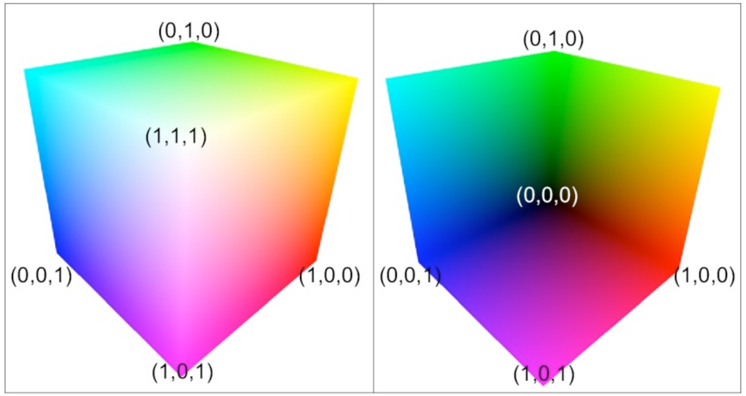


- Based on the intersection point and … it
  a new ray can be computed

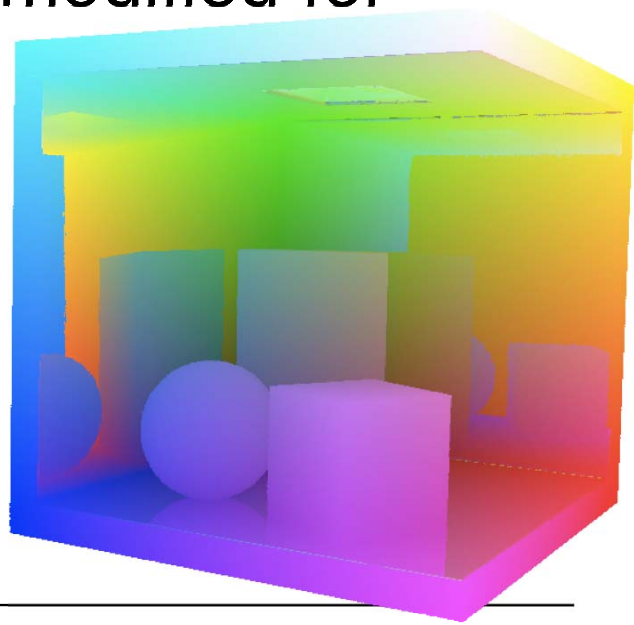




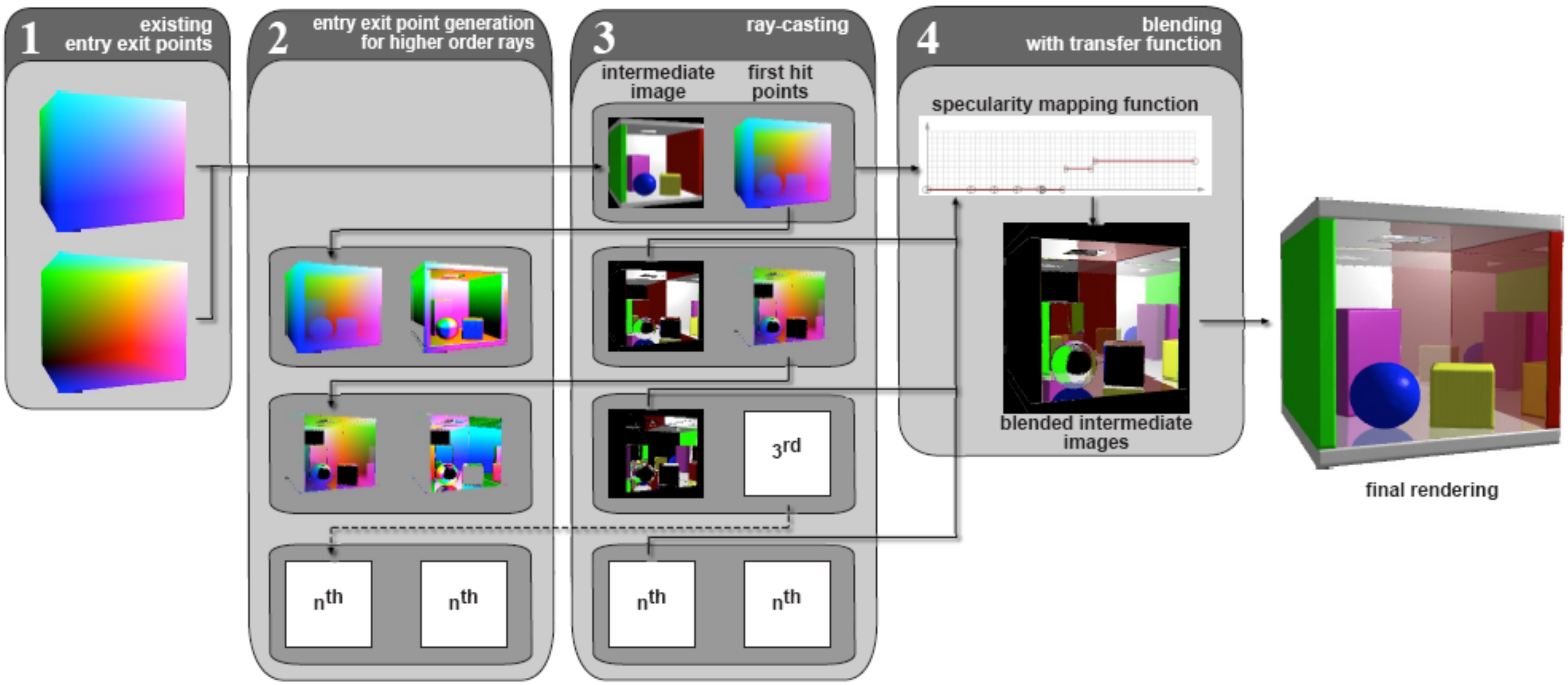
Higher Order Rays



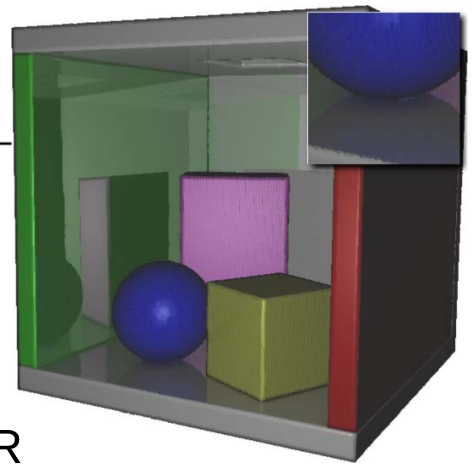
- Entry parameter texture can be modified for tracing higher order rays
  1. Compute first hit point for each pixel
  2. Calculate new ray at each first hit point based on gradient
  3. Generate new exit parameters



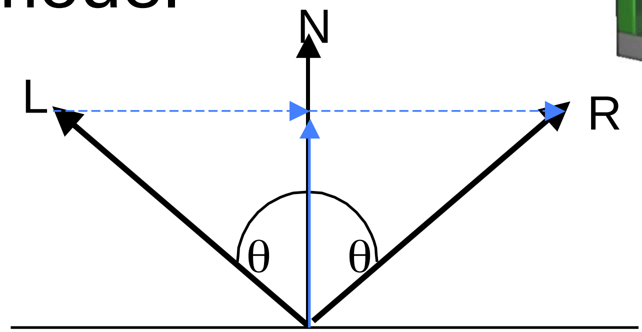
Higher Order Rays



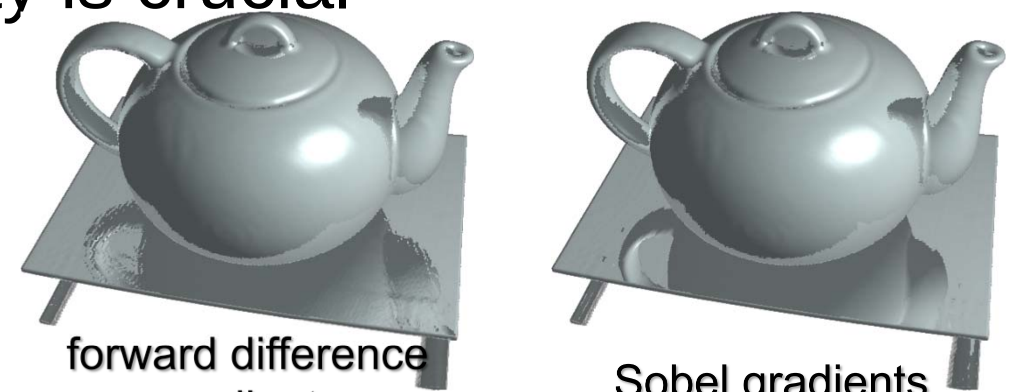
Mirror Reflections



- Compute direction of reflection ray as done in Phong model
  - R = 2 *N* cos θ - *L*



- Again, gradient quality is crucial

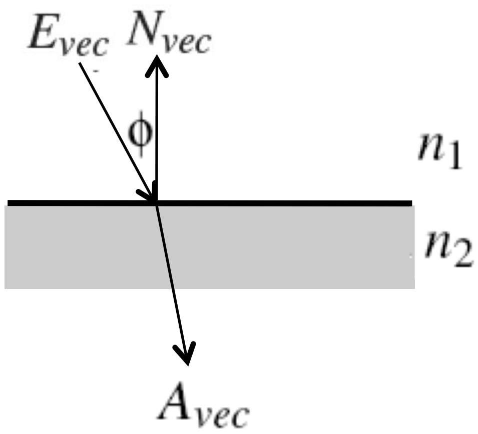


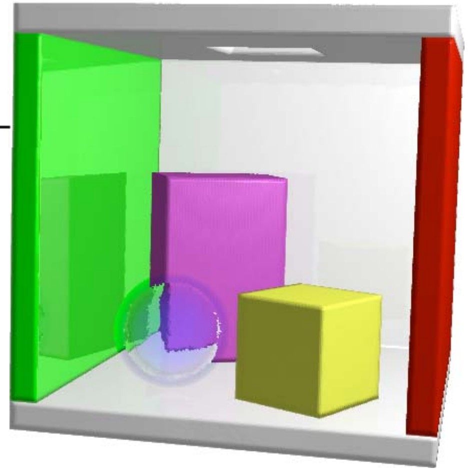
forward difference gradients

Sobel gradients

Refraction

- The refraction indices of the materials have to be known

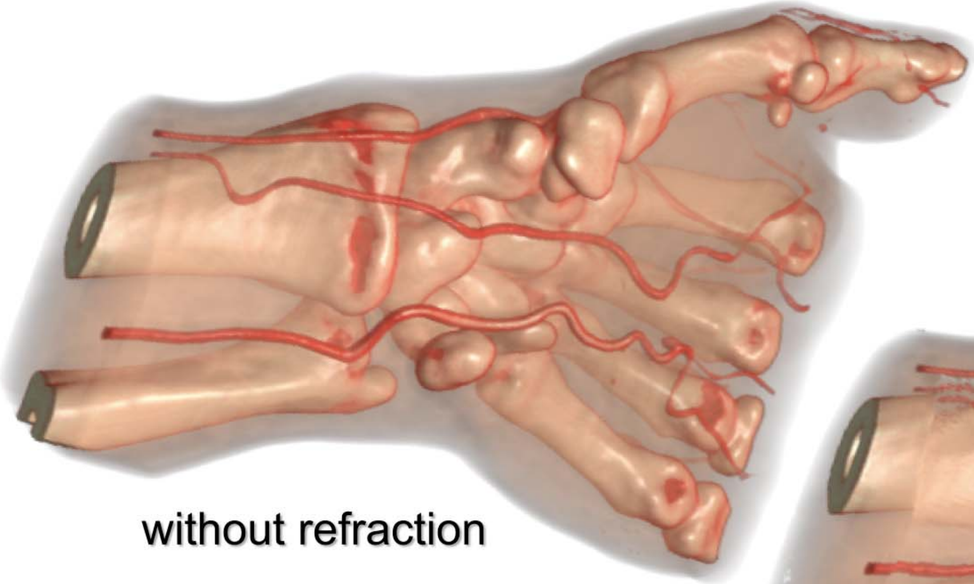




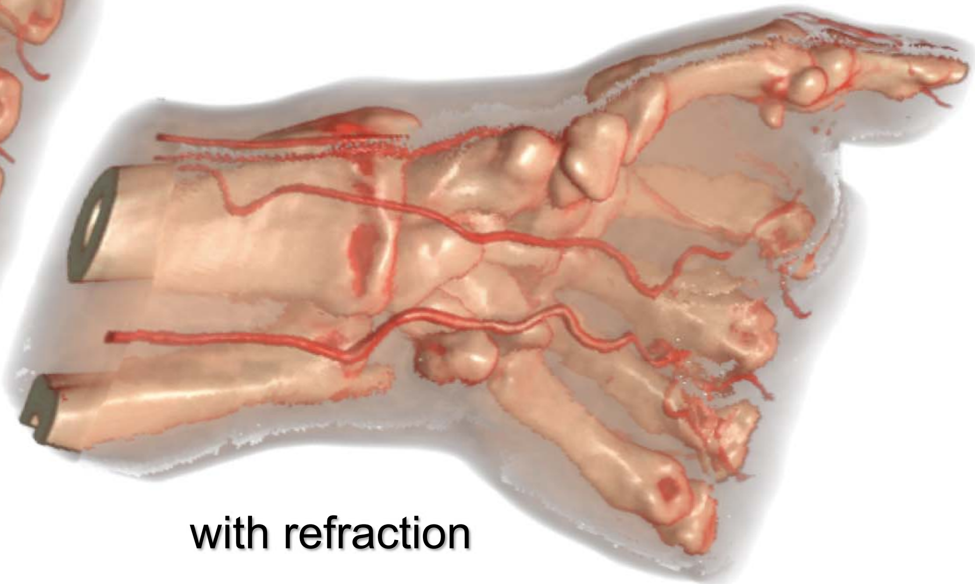
- By exploiting Snell's law

$$\cos(\theta) = \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2 \cdot (1 - (\cos(\phi))^2)}$$

Refraction



without refraction



with refraction

Ray Tracing Performance

- Intel Core2 6600 (2.4GHz), 2GB RAM and an nVidia GeForce 8800GTX

data set	recursion depth	screen resolution 256^2	512^2
Hand $256 \times 256 \times 147$	0	57 fps	39 fps
	1	38 fps	22 fps
	2	30 fps	14 fps
	3	24 fps	12 fps
Teapot $256 \times 256 \times 178$	0	55 fps	38 fps
	1	40 fps	25 fps
	2	33 fps	19 fps
	3	30 fps	17 fps
Cornell Box $256 \times 256 \times 256$	0	59 fps	47 fps
	1	38 fps	19 fps
	2	31 fps	12 fps
	3	18 fps	10 fps

Shadowing

- Adding interactive shadows to volume graphics supports spatial comprehension
- Focus on shadow algorithms integration able into GPU-based ray casters
  - Casting shadow rays
  - Shadow mapping
  - Deep shadow maps

Object- vs. Image-Based

- Object-based
  - object-based shadow algorithms like Crow's shadow volumes
  - require polygonal representation of rendered objects

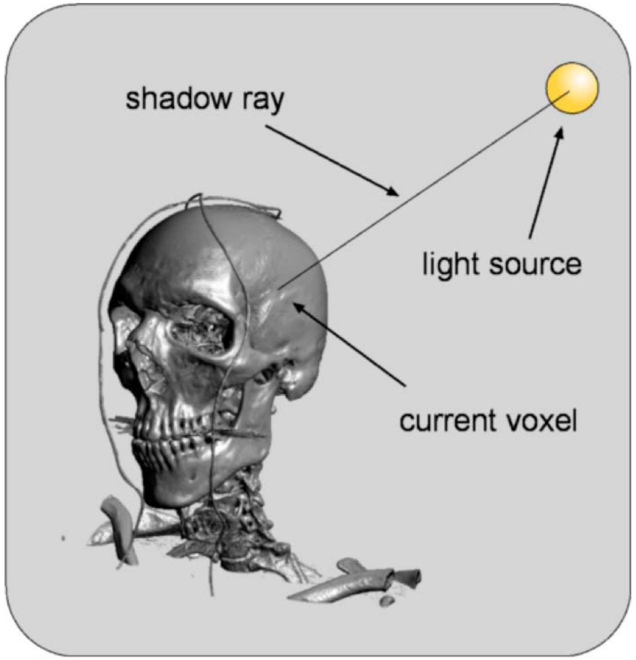
[Crow, Siggraph 1977]

- Image-based
  - representation of shadows in an image
  - shadow mapping
  - opacity shadow maps
  - deep shadow maps (allow transparent objects)

[Williams, Siggraph 1978]

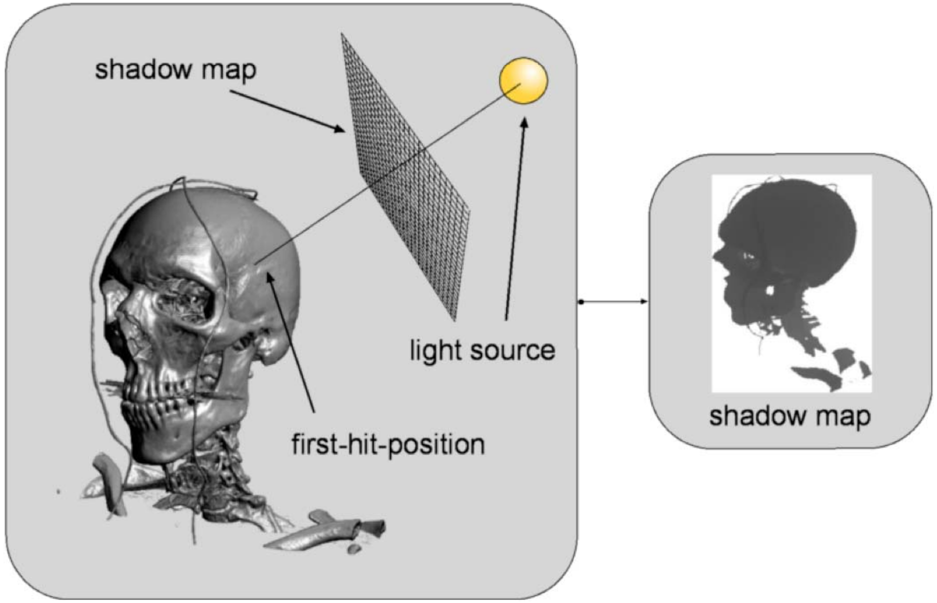
Shadow Rays

- Similar to shadow rays in ray tracing
  - Opaque occluders (similar to first hit raycasting)
  - Alpha raycasting (full volume rendering integral)



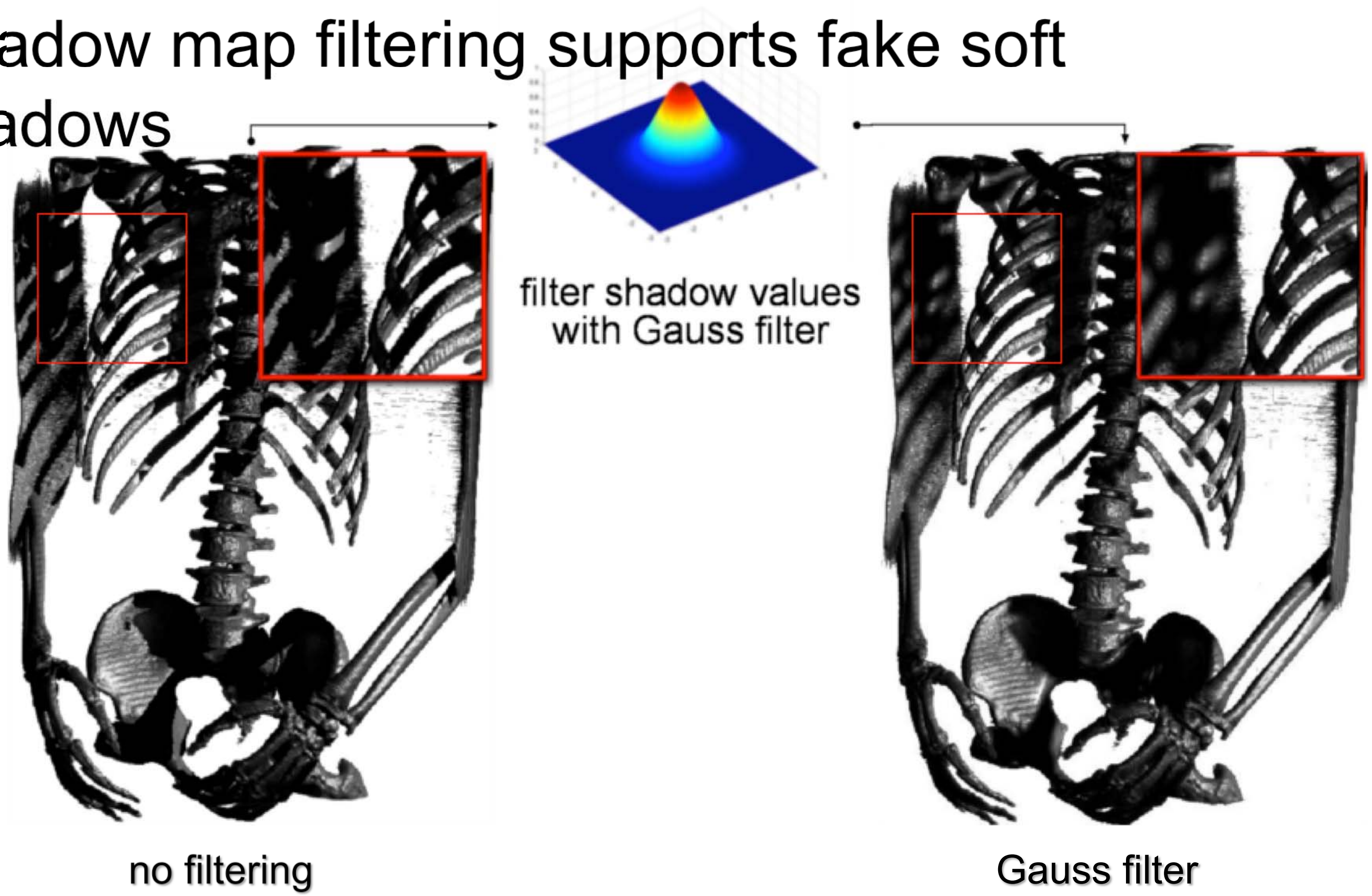
Shadow Mapping

- Shadow map saves depth values of first hit points as seen from the light source
  - Depth comparison during rendering gives binary decision for shadowing
  - Shadow threshold marks intensity limit
  - Supports opaque occluders only



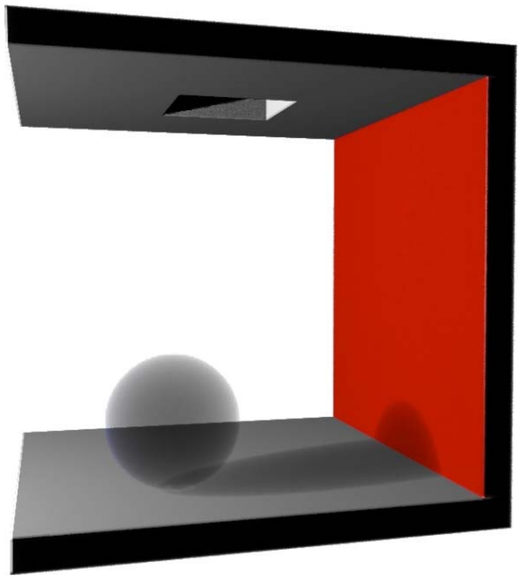
Shadow Mapping

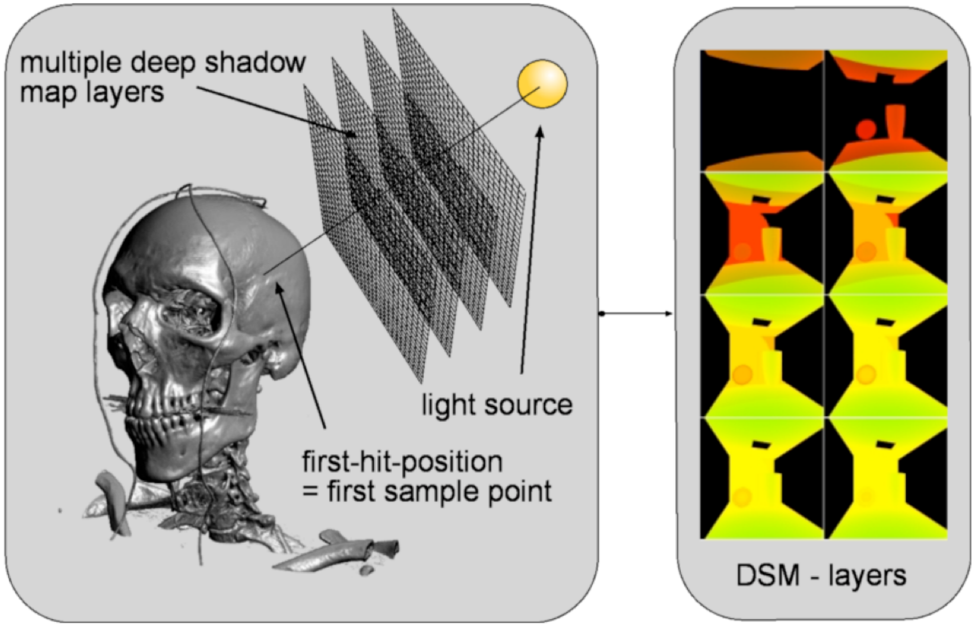
- Shadow map filtering supports fake soft shadows



Deep Shadow Maps

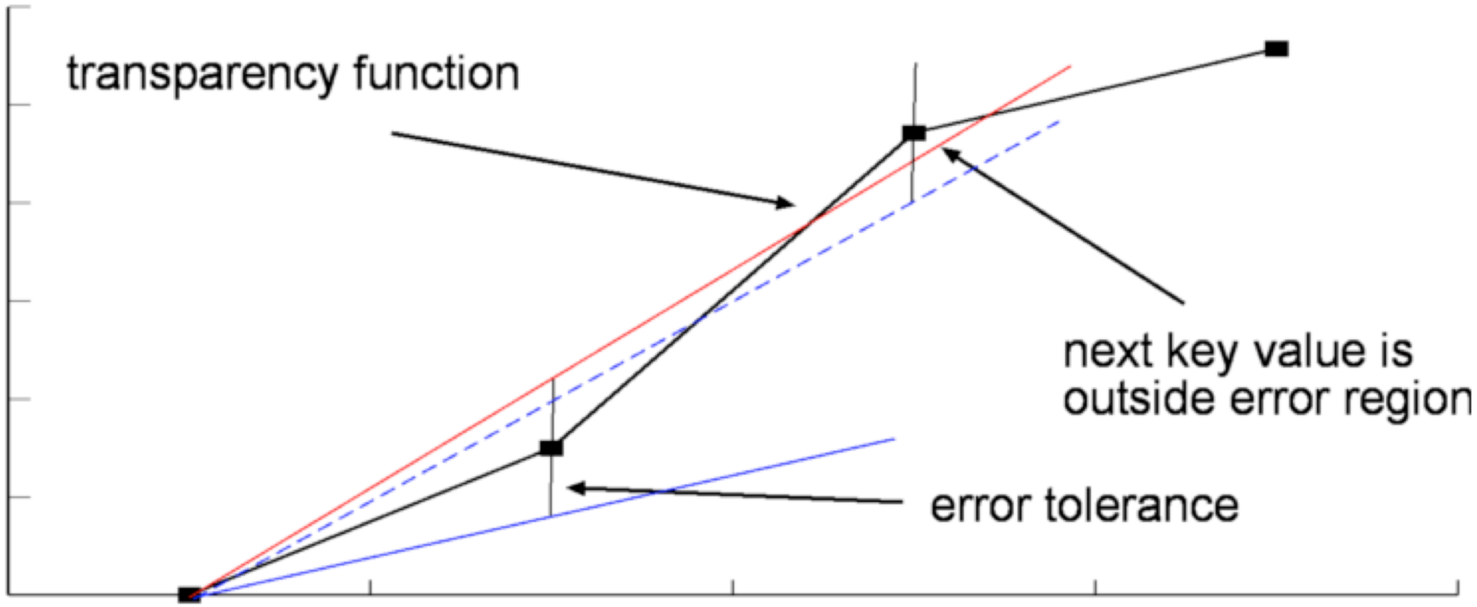
- Support semi-transparent occluders by incorporating multiple layers
- Each layer is a pair of depth and transparency
- For each pixel control points of piecewise linear functions are saved





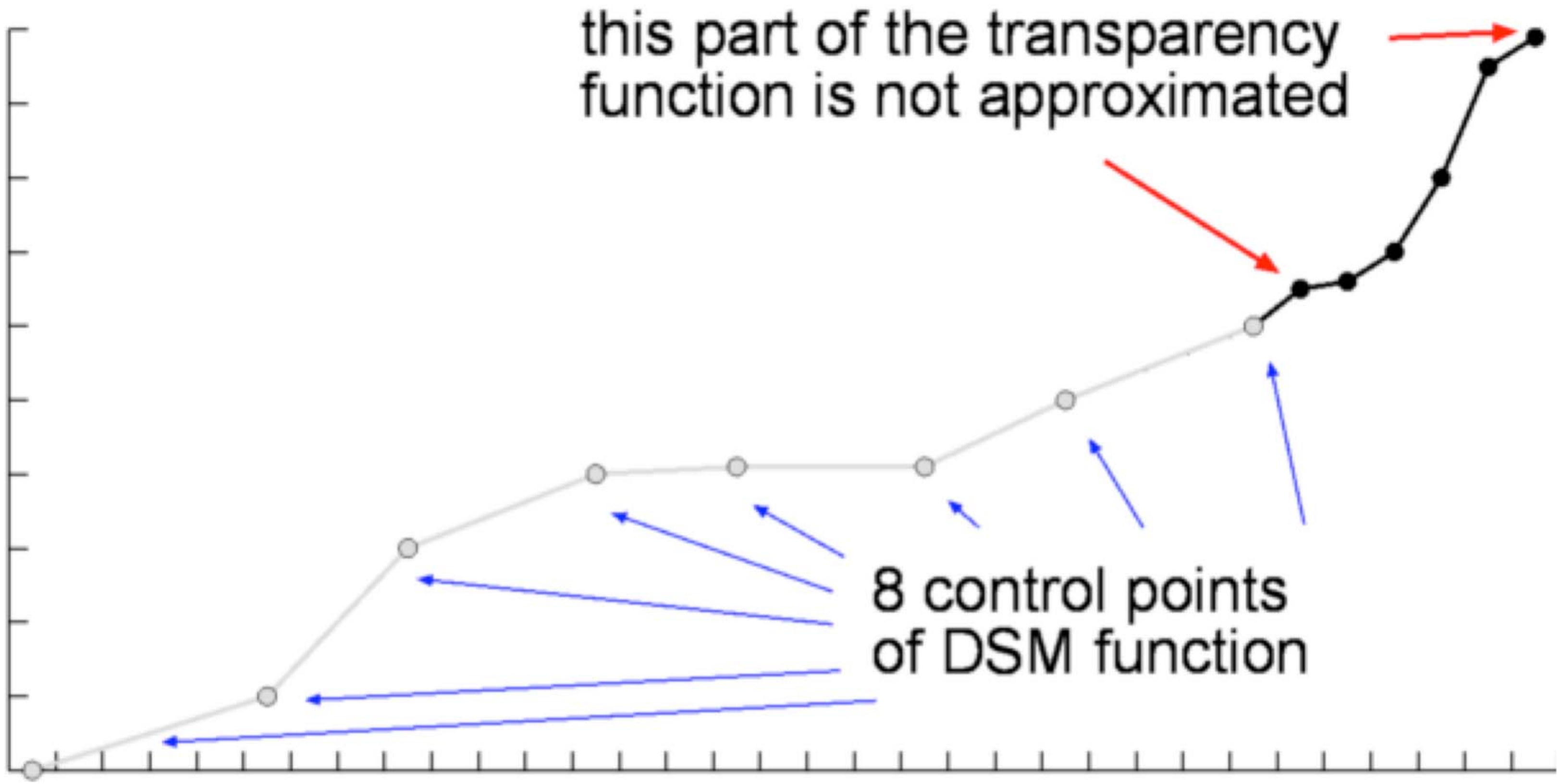
Deep Shadow Maps

- Deep shadow map construction
  - At the first hit point, the first key value is saved
  - Based on a error function, further key values are saved

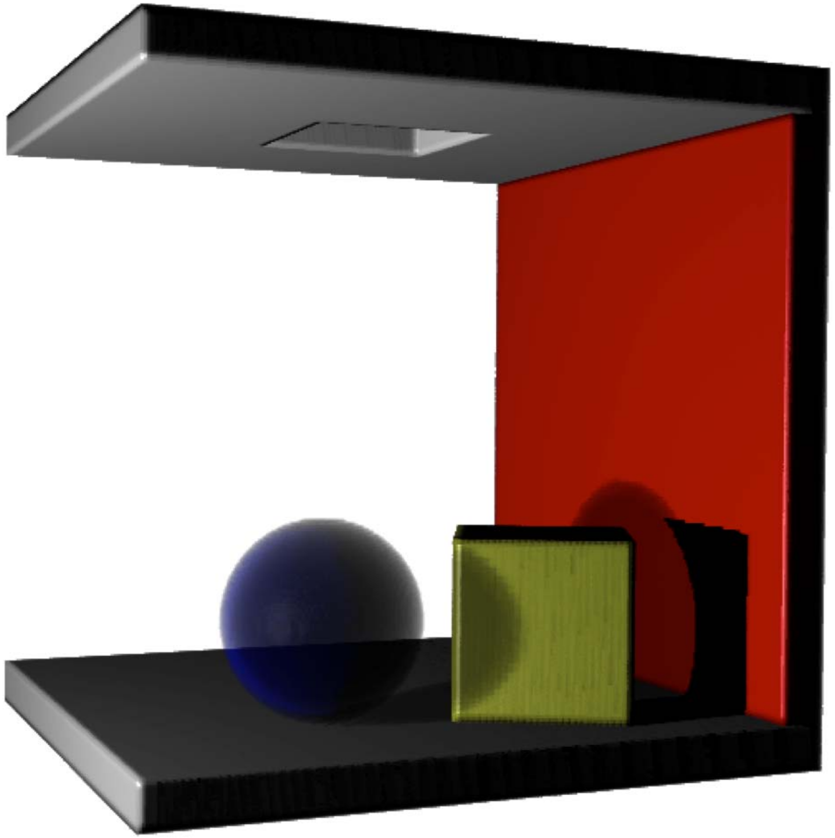


Deep Shadow Maps

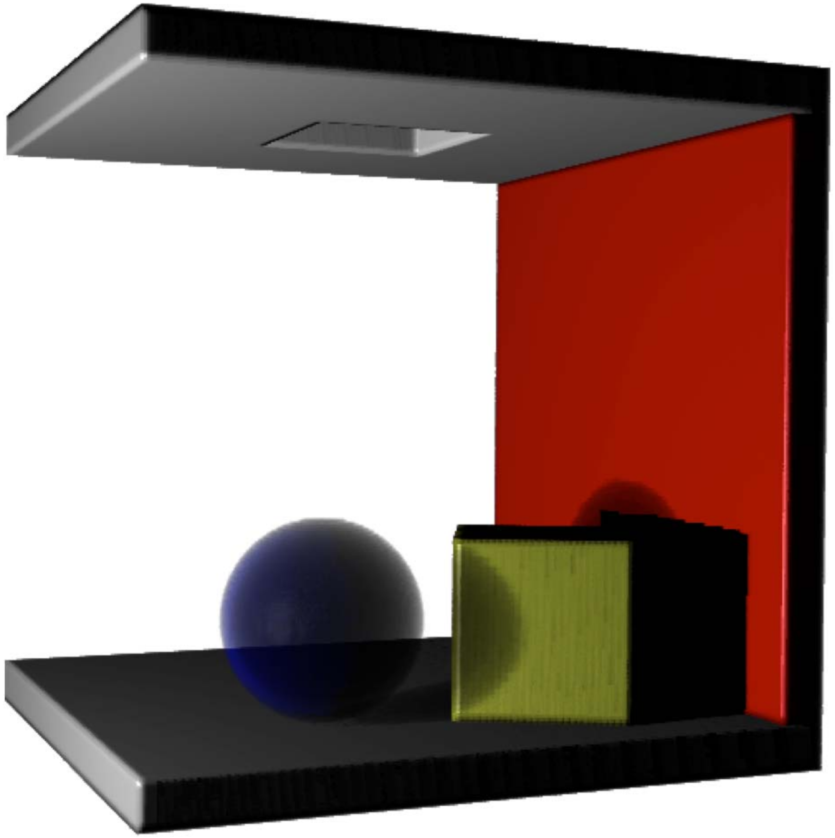
- Original function differs by the chosen error value



Approximation Artifacts



error value of 0.01

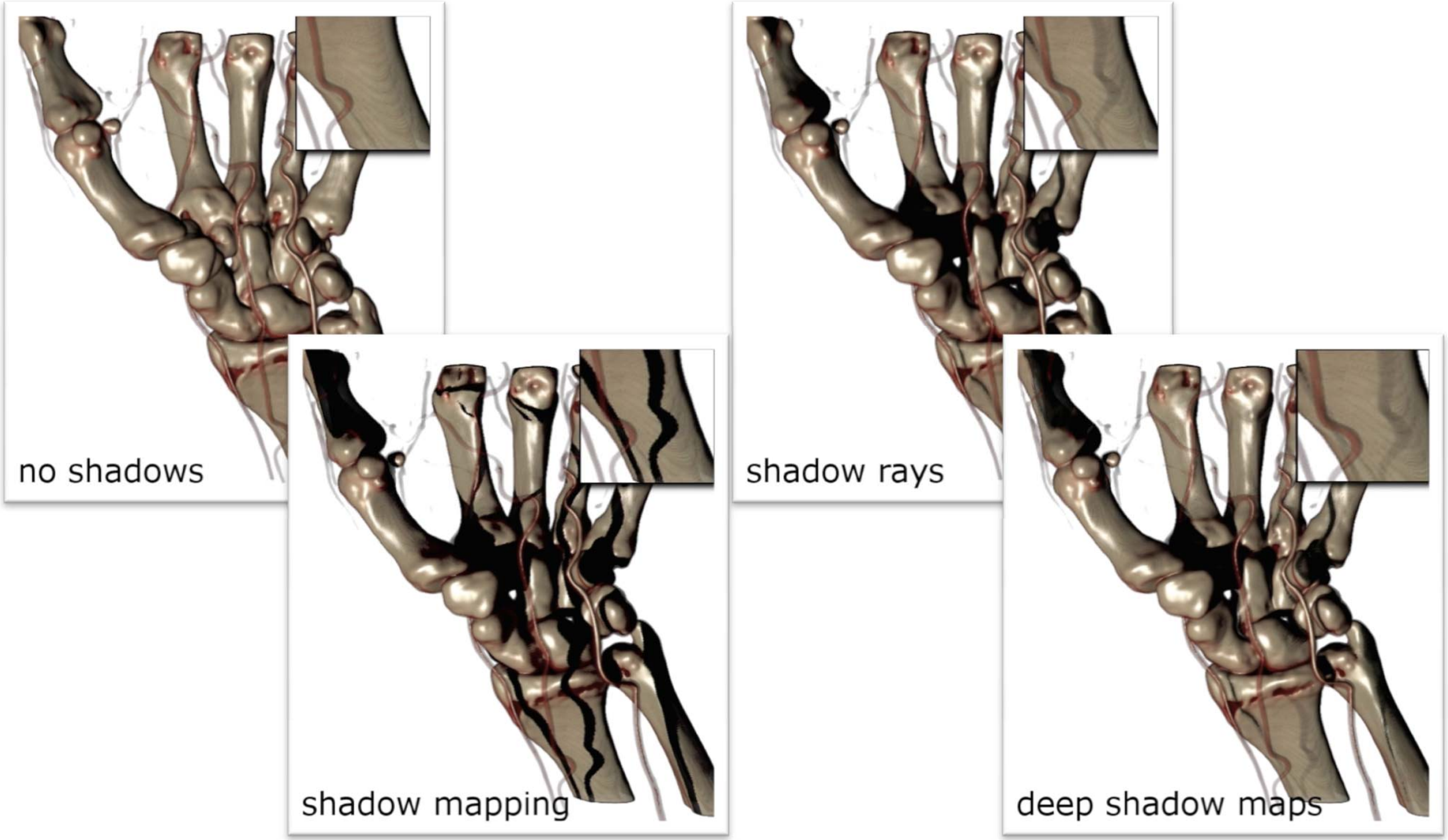


error value of 0.00005

Deep Shadow Mapping

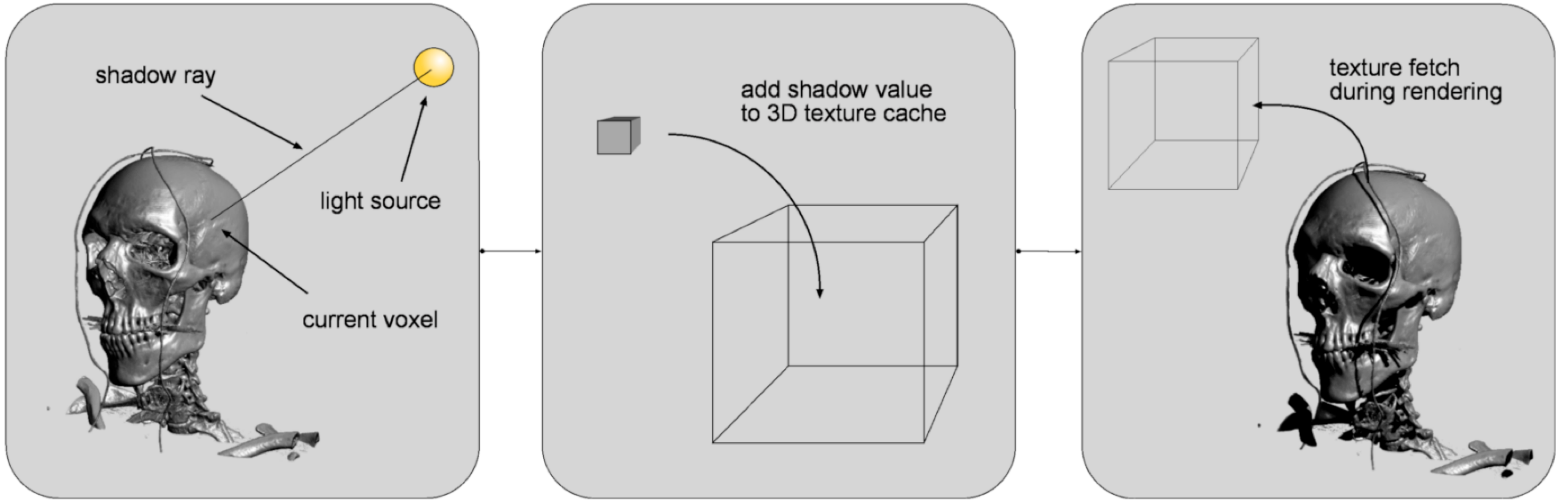
[Hadwiger et al., Graphics Hardware 2006]

Visual Comparison



3D Texture Caching

- Shadows can be cached in 3D textures to gain performance
  - 3D texture for shadow lookup
  - Preprocessing shadow feelers
  - Needs to be recomputed on light source or transfer function change



Shadowing Performance

Shadow Mode	RC	without RC
ShadowRay (B)	10.03	10.03
ShadowRay (A)	10.0	10.0
ShadowRay (B + PP)	5.59	46.0
ShadowMap (B)	29.08	45.5
DeepShadowMap (A)	15.76	34.5
DeepShadowMap (A + PP)	13.06	45.2

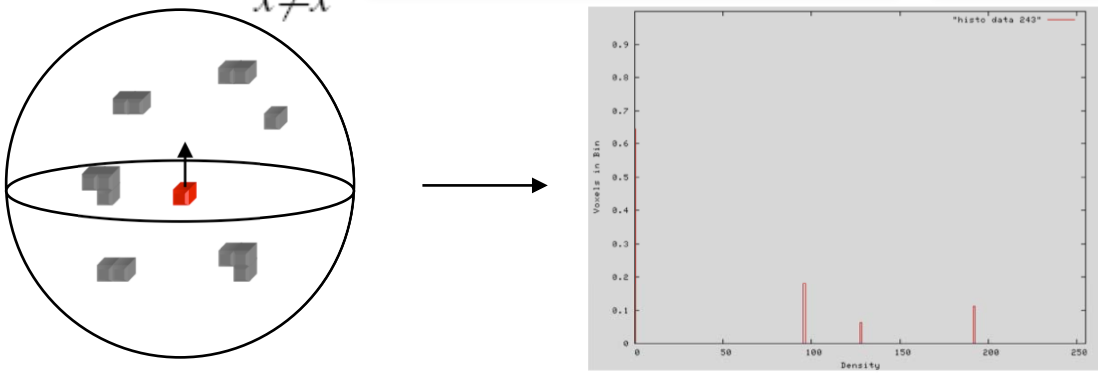
- Intel Core2 6600 (2.4GHz), 2GB RAM and an nVidia GeForce 8800GTX

Color Bleeding

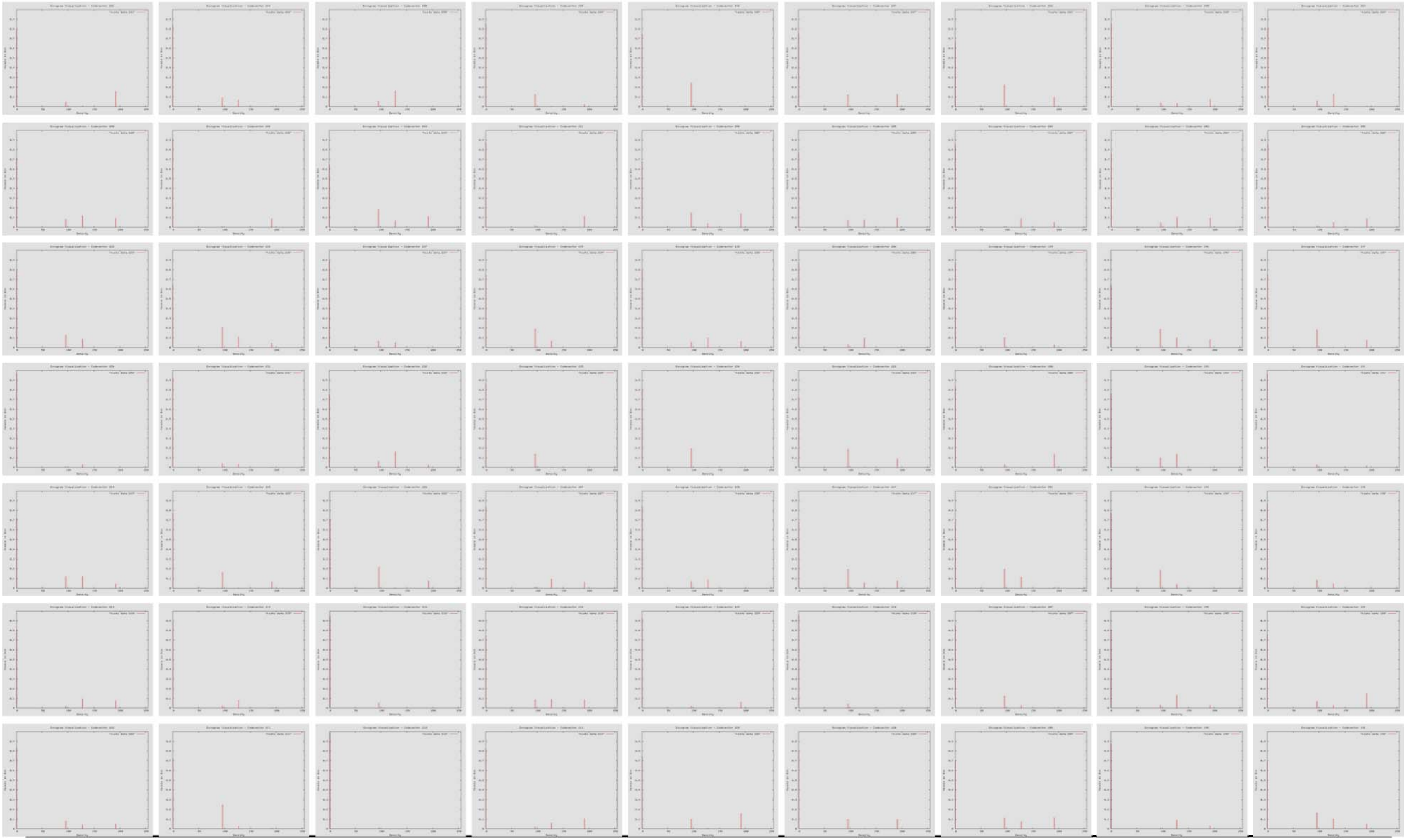
- Caused by vicinity of each voxel
- Compute a normalized local histogram to capture vicinity

$$LH(x) = (LH_0(x), \ldots, LH_{n-1}(x))$$

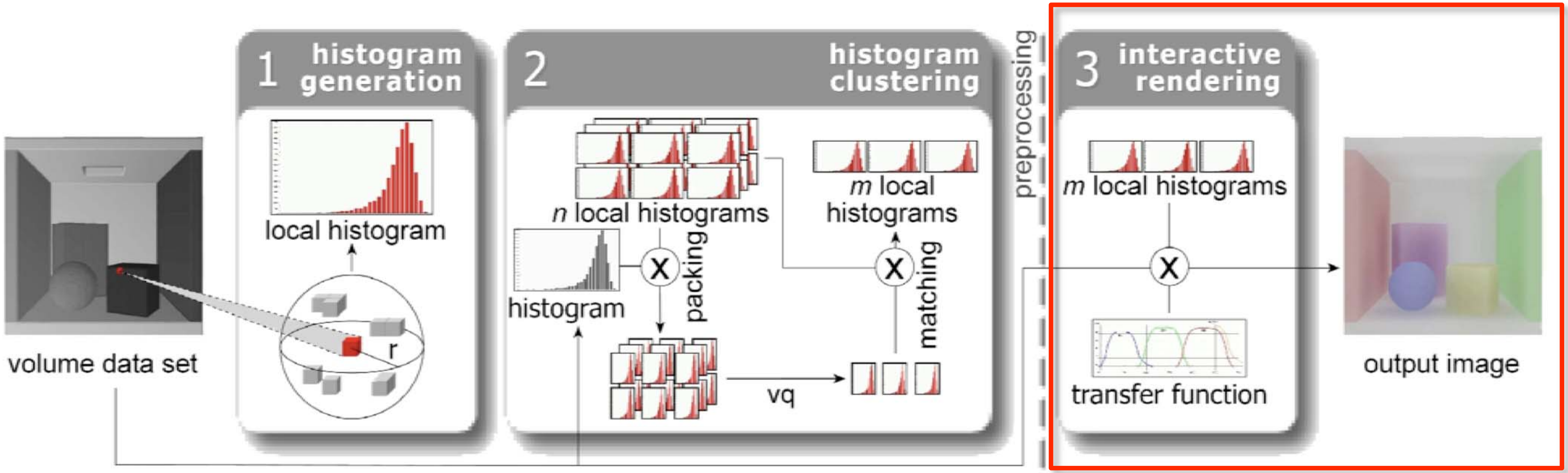
$$LH_k(x) = \sum_{\substack{\tilde{x} \in S_r(x) \\ \tilde{x} \neq x}} f_{dist}\left(\frac{|x-\tilde{x}|}{d_{min}}\right) \cdot g(f(\tilde{x}), k)$$



Histogram Generation

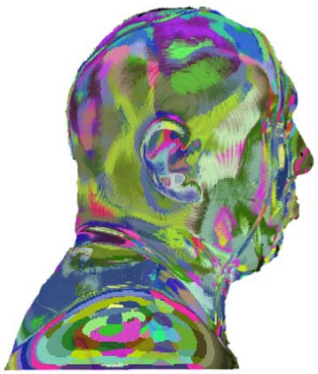


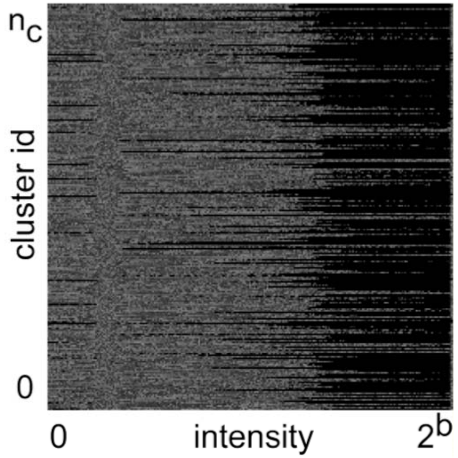
Color Bleeding Workflow



Interactive Rendering

- Two additional texture fetches required
  1. Obtain the cluster ID of the current sample *x*
  2. Fetch the current environment color $E_{env}(x)$





- $E_{env}(x)$ is computed by considering the current transfer function

Color Bleeding: Rendering

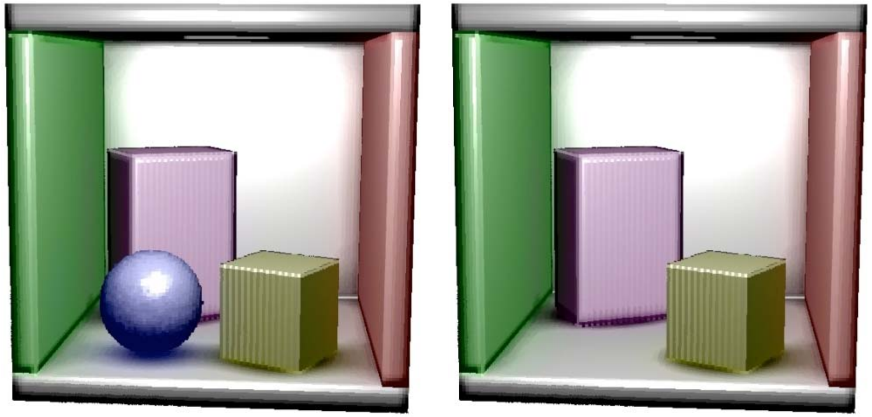
- Combination with the transfer function

$$E_{env}(x, \triangledown \tau(f(x))) = \frac{1}{\frac{2}{3}\pi r^3} \sum_{0 \leq j < 2^b} \tau_\alpha(j) \cdot \tau_{rgb}(j) \cdot LH_j(x)$$

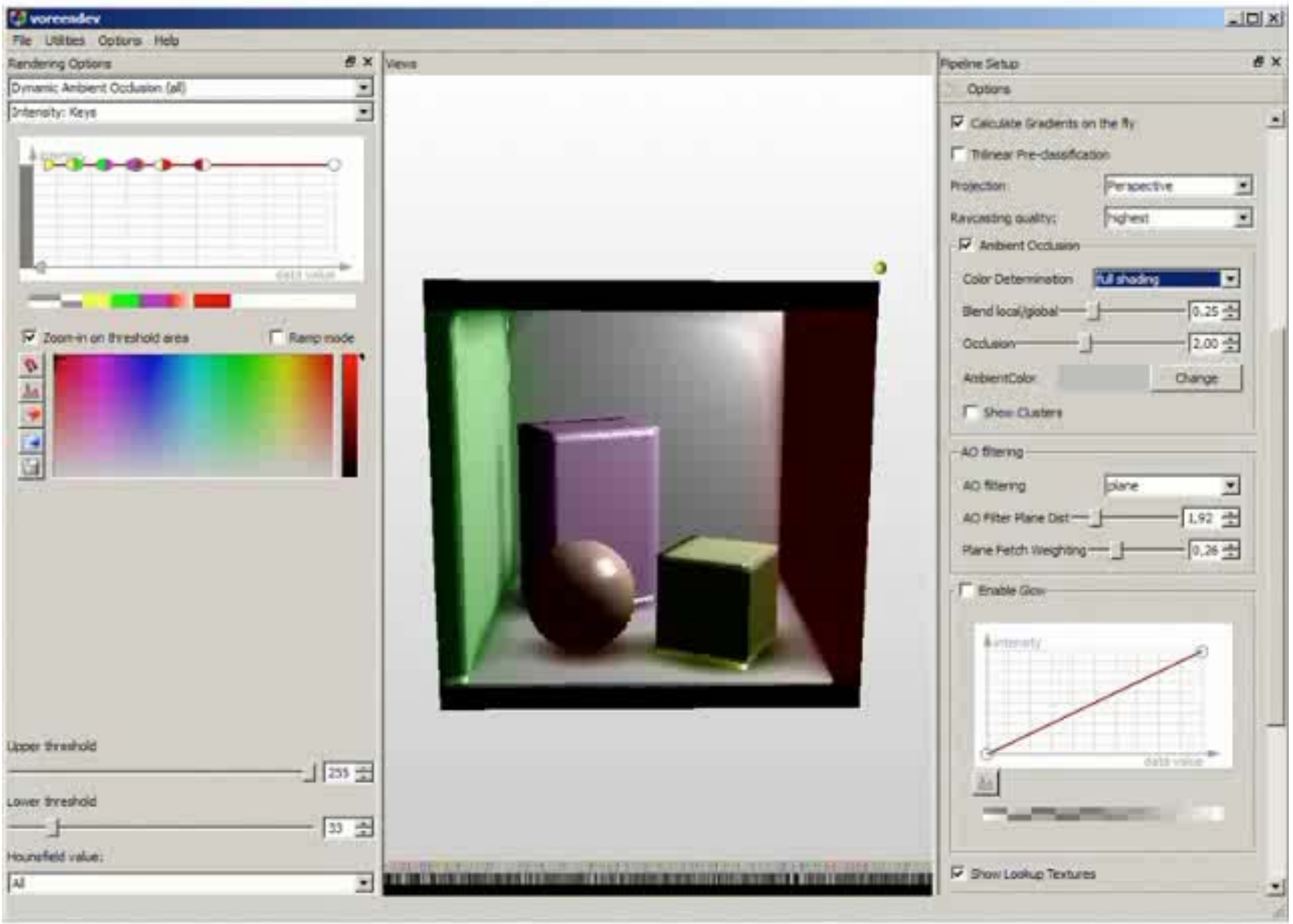
Color Bleeding: Rendering

- Rendering is done in YUV color space
- Color: Interpolate between E_{env} and $\tau_{rgb}(x)$
  - Local occlusion O_{env} used as the interpolation factor
- Luminance: minimum of $1.0 - O_{env}$ and $\bigtriangledown \tau(f(x)) \cdot L$
- Specular highlights can be added

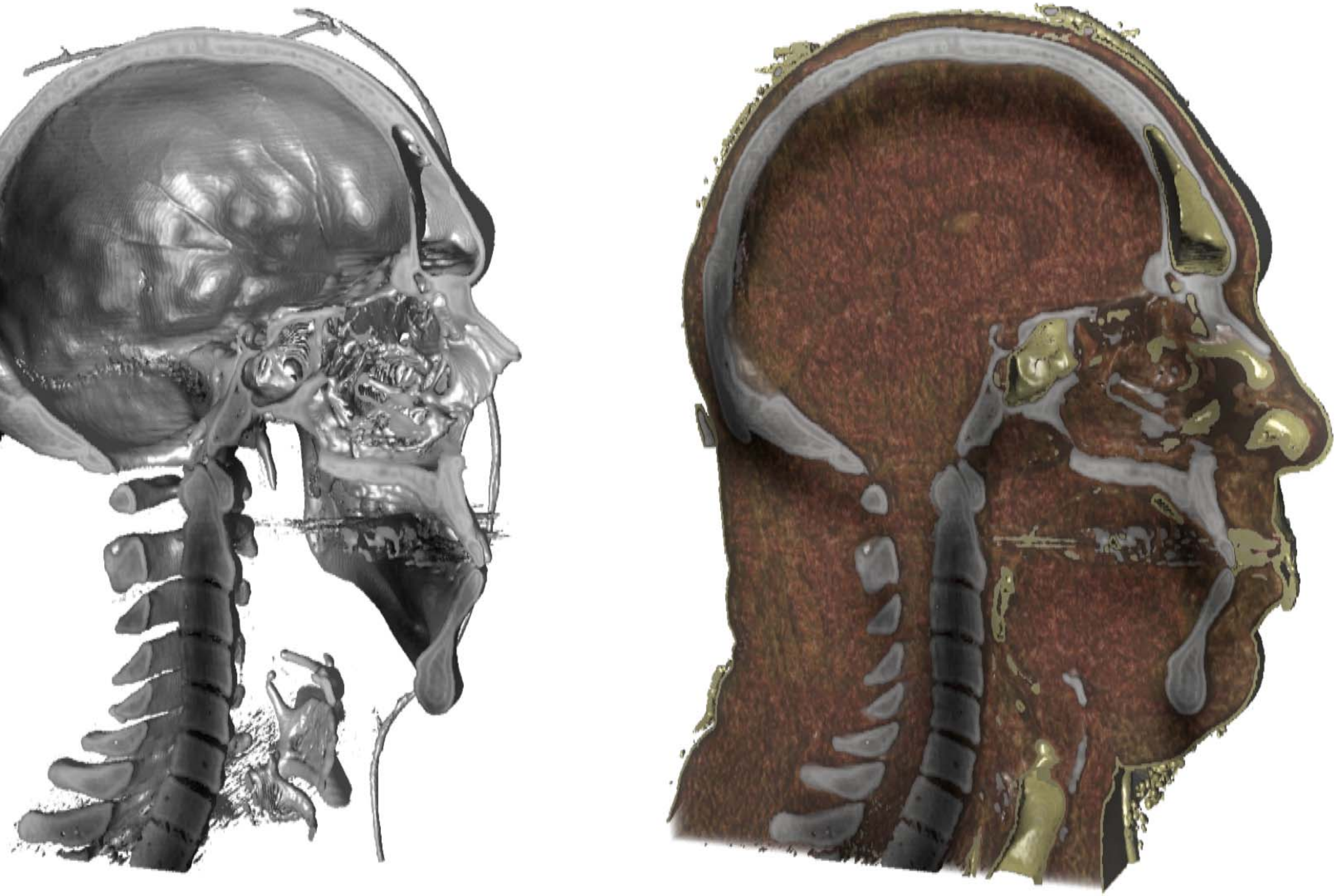
E_{env} only



Demonstration Video

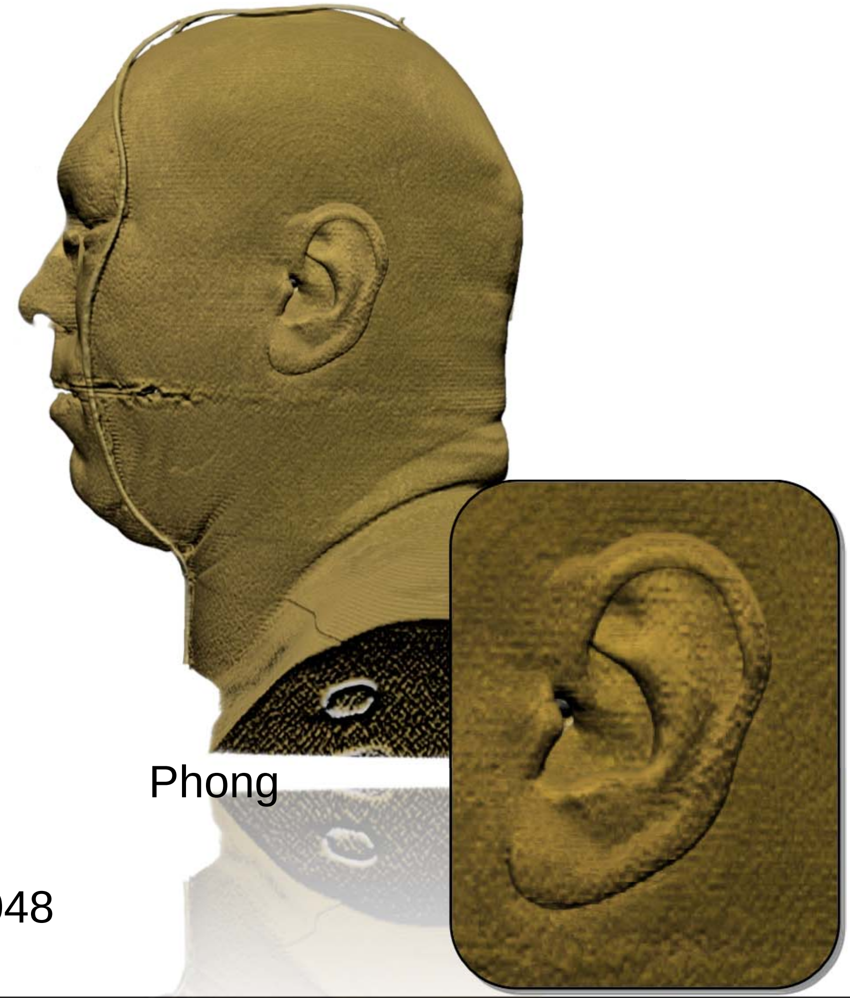


Color Bleeding

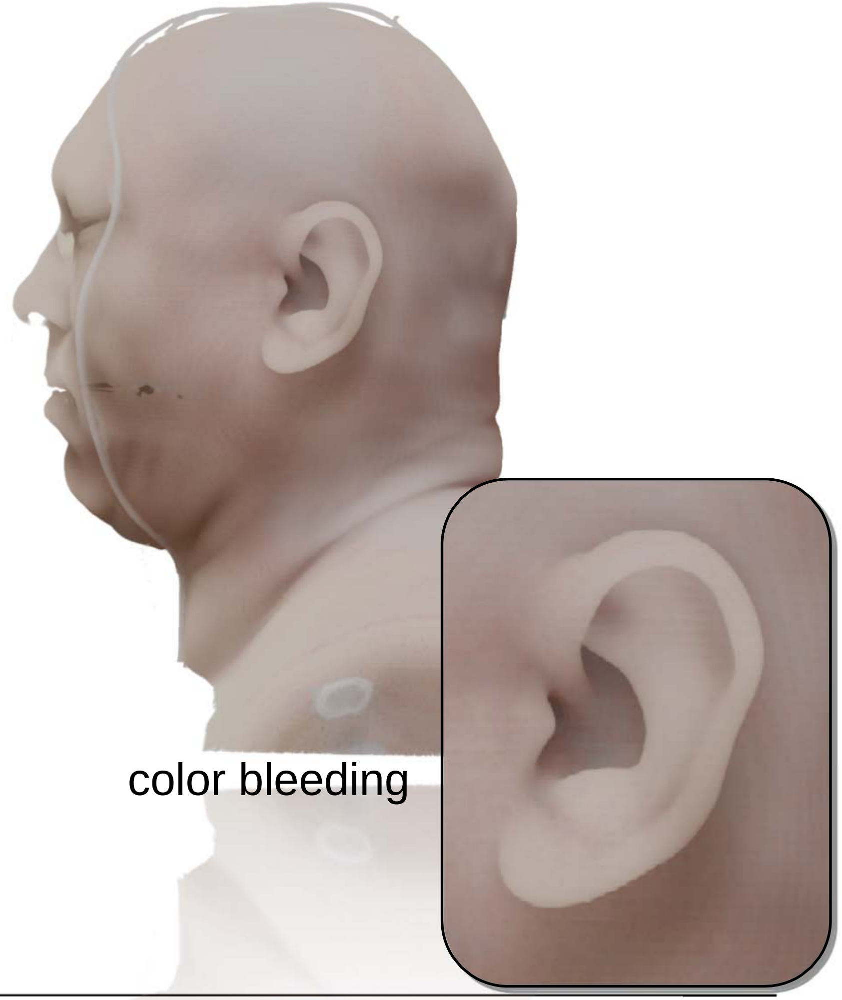


n_c=2048

Color Bleeding



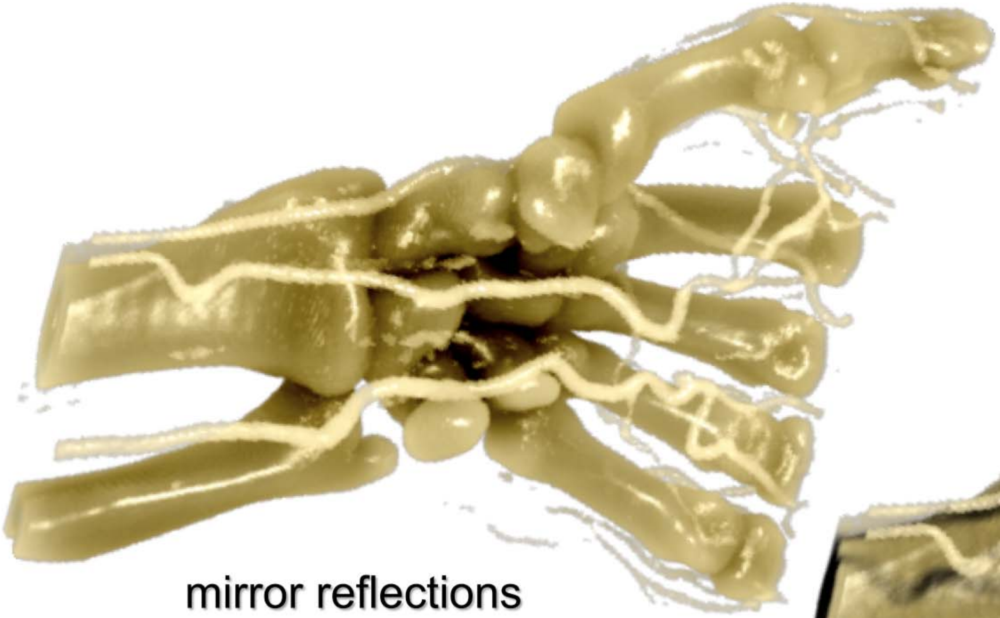
Phong



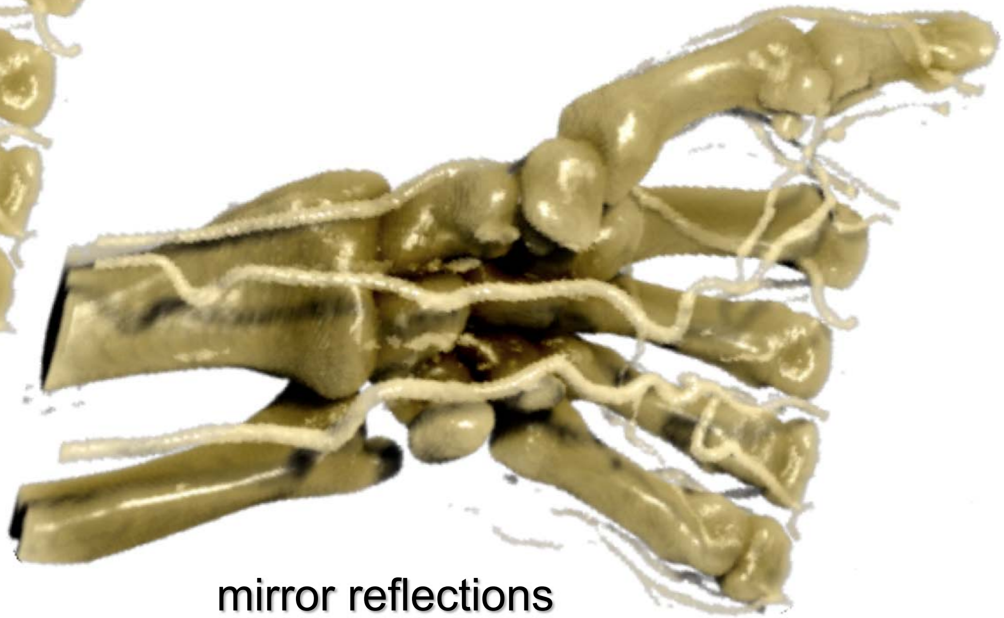
color bleeding

n_c=2048

Combining the Effects



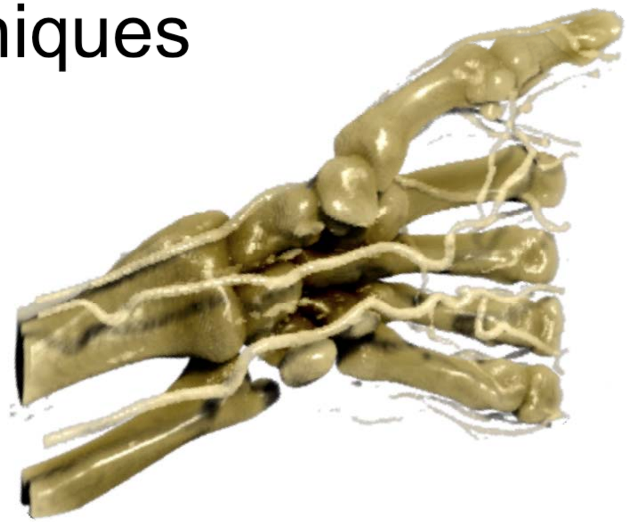
mirror reflections



mirror reflections
and semi-transparent shadows

Summary

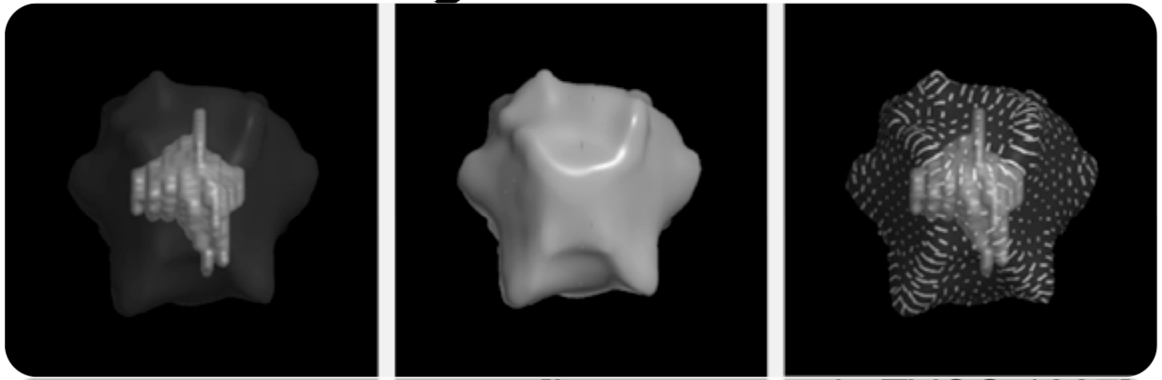
- Local volume illumination
- Gradient computation methods
- GPU based volume ray tracing
  - Refraction
  - Specular reflections
- Interactive shadowing techniques
  - Hard vs. soft-shadows
  - Deep shadow maps
- Color bleeding



Future Work

- Anisotropic lighting models
  - Muscle fibres are anisotropic

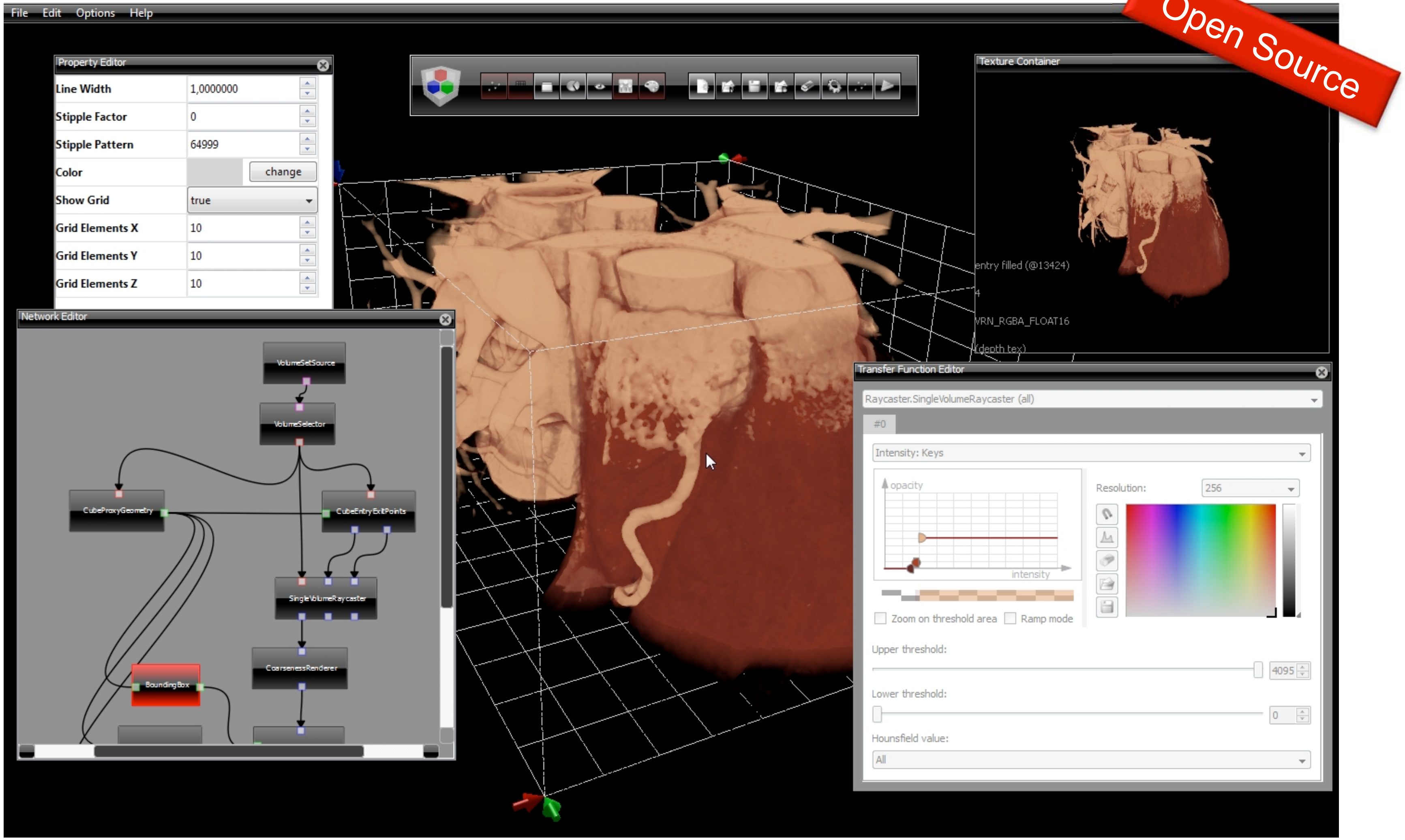
- Improve perception of semi-transparent structures containing each other



[Interrante et al., TVCG 1997]

Voreen

www.voreen.org



References

- Hadwiger, M., Kratz, A., Sigg, C., Bühler, K.. GPU-accelerated deep shadow maps for direct volume rendering. In GH ’06: Proceedings of the 21st ACM SIGGRAPH/ Eurographics symposium on Graphics hardware, pages 49–52, New York, NY, USA, 2006. ACM Press.
- Levoy, M. (1988). Display of surfaces from volume data. IEEE Computer Graphics and Applications, 8(3):29–37.
- Ropinski, T., Meyer-Spradow, J., Diepenbrock, S., Mensmann, J., and Hinrichs, K.. Interactive Volume Rendering with Dynamic Ambient Occlusion and Color Bleeding. *Computer Graphics Forum (Eurographics 2008), 27(2):567–576, 2008.*
- Ropinski, T., Kasten, J., and Hinrichs, K.. Efficient Shadows for GPU-based Volume Raycasting. In *Proceedings of the 16th International Conference in Central Europe on Computer Graphics, Visualization (WSCG08), pages 17–24, 2008.*
- Stegmaier, S., Strengert, M., Klein, T., and Ertl, T. (2005). A simple and flexible volume rendering framework for graphics-hardware–based raycasting. In Proceedings of the International Workshop on Volume Graphics’05, pages 187–195.
- Ropinski, T., Kasten, J., and Hinrichs, K. Gpu-based vol- ume ray-casting supporting specular reflection and refraction. In Proceedings of VisiGrapp ’09, pages 219–222, 2009.

Thanks to Jens Kasten (ZIB, Berlin) for implementing the ray-tracer and part of the shadow algorithms.