Spatial Visualization: Volume Rendering

CS 5630/6630, Fall 2016
Alexander Lex
Guest Lecturer: Aaron Knoll

www.sci.utah.edu

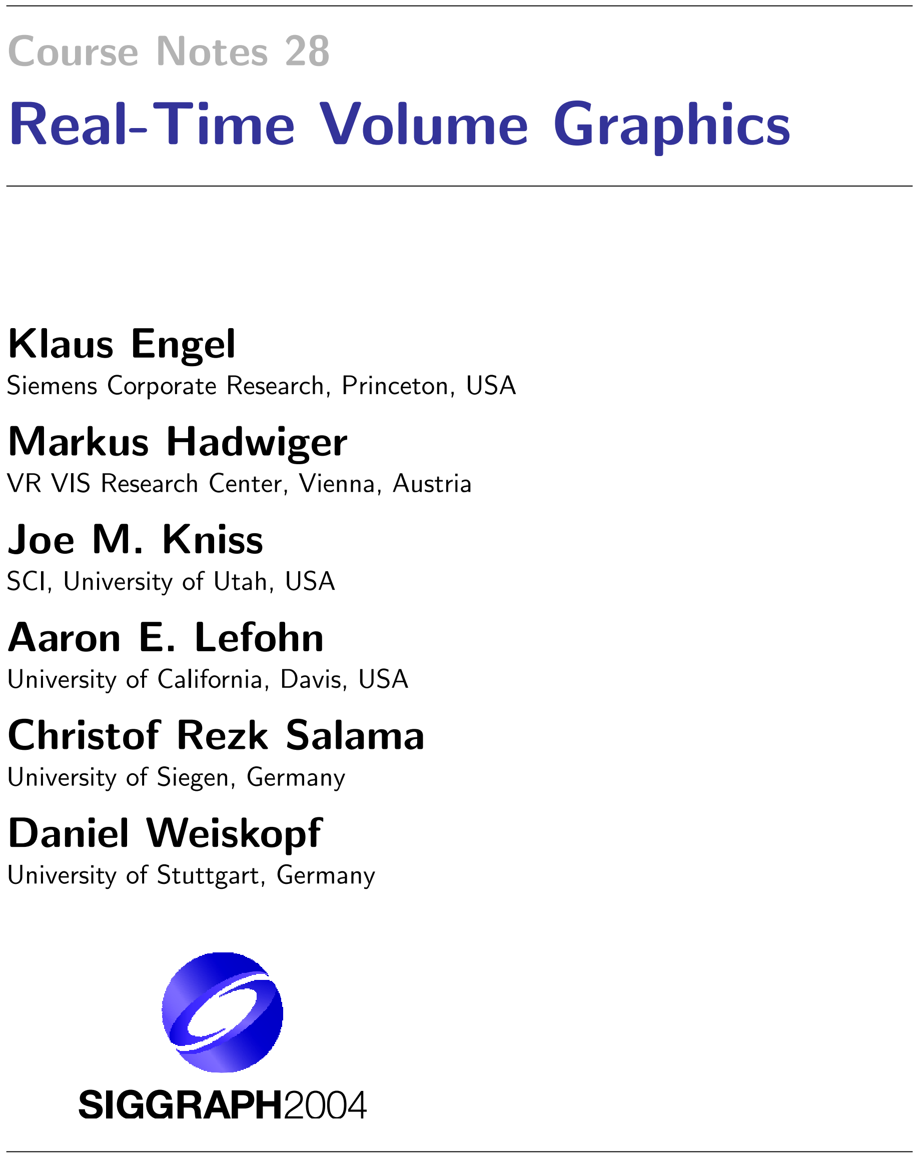
Recap from last spatial vis lecture

- Fields and grids
- Unstructured and structured grids
- Direct and indirect workflows
- Interpolation

Today

- Computer graphics basics
  - Rasterization and ray tracing
  - Shading
  - Alpha blending
- Volume rendering
  - How integration works
  - Code and examples!
- Transfer functions
  - 1D, 2D, and Exotica

Reading



Course Notes 28

Real-Time Volume Graphics

Klaus Engel
Siemens Corporate Research, Princeton, USA

Markus Hadwiger
VR VIS Research Center, Vienna, Austria

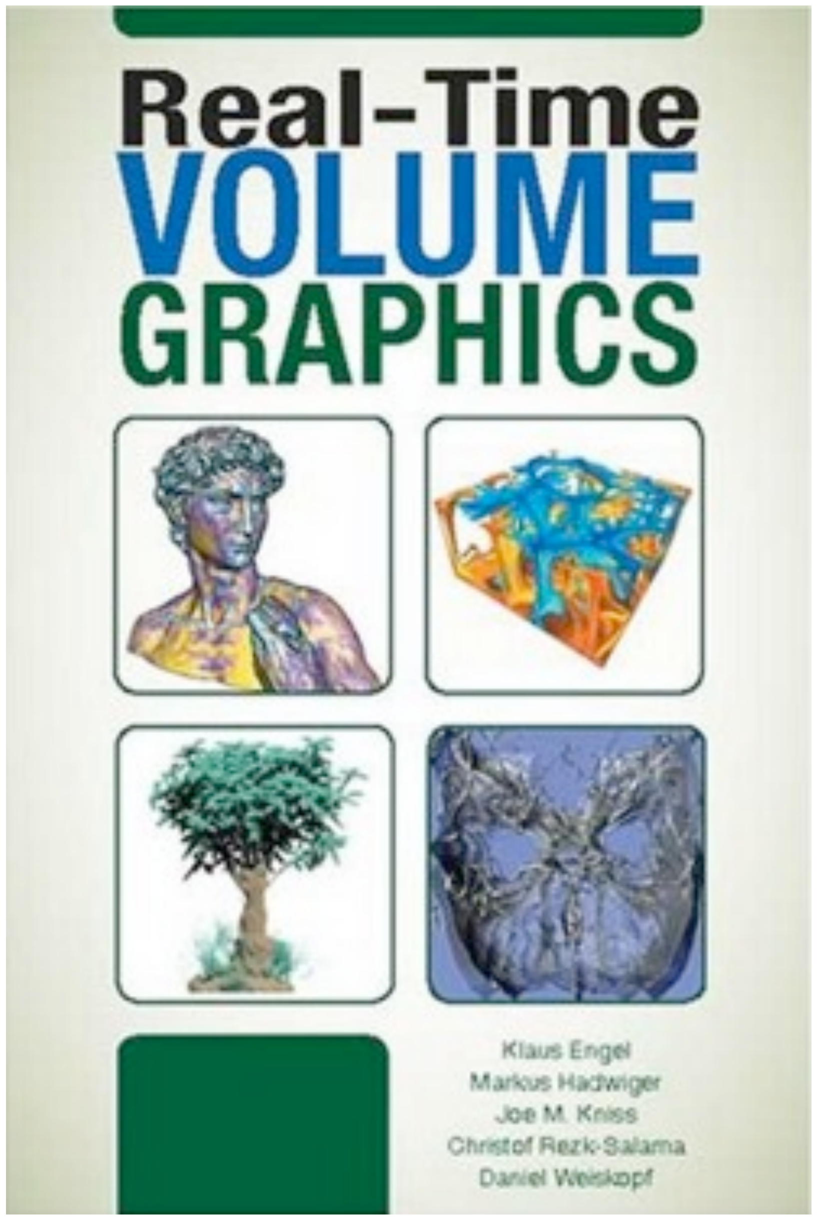
Joe M. Kniss
SCI, University of Utah, USA

Aaron E. Lefohn
University of California, Davis, USA

Christof Rezk Salama
University of Siegen, Germany

Daniel Weiskopf
University of Stuttgart, Germany

SIGGRAPH2004



http://www.real-time-volume-graphics.org/

Tutorials 1,3,4,5

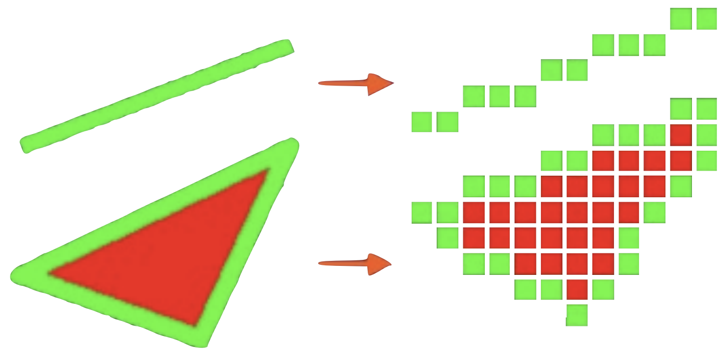
(3D) Computer Graphics

What is graphics?

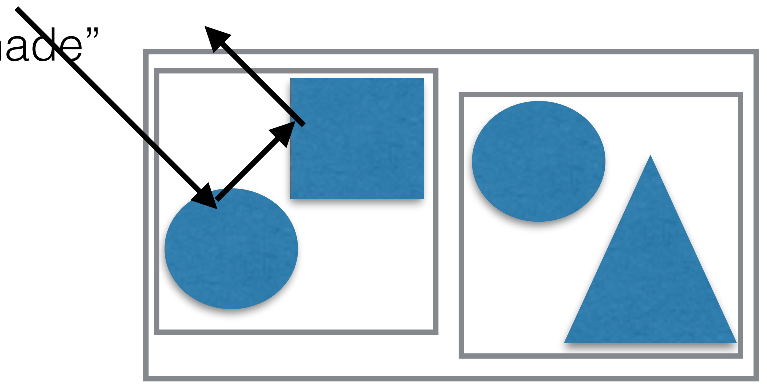
- Computer graphics is the process of converting a 3D scene (model) into a 2D image (frame buffer), via a camera model.

- Principally, there are two ways of doing this:

  - **Rasterization**
    "project the scene, sort and shade textured fragments, and shade"
    The camera transforms the primitives.
    4x4 matrix multiplication, Z-buffer algorithm, scan conversion.
    Cost: O(N)
    APIs: OpenGL / WebGL / three.js, DirectX, Vulcan



  - **Ray tracing**
    Ray casting: "generate rays, search the scene for which primitive a ray hits, and shade"
    Ray tracing: "rinse and repeat."
    The camera defines the ray; primitives remain in native 3D coordinates.
    Many parallel tasks traversing a tree or grid in very different ways.
    Cost: O(P log N).
    APIs: Intel Embree & OSPRay, NVIDIA OptiX & IndeX, write your own!



- Volume rendering can be implemented either via ray tracing (sampling along the ray) or rasterization (textured proxy geometry)

Rasterization

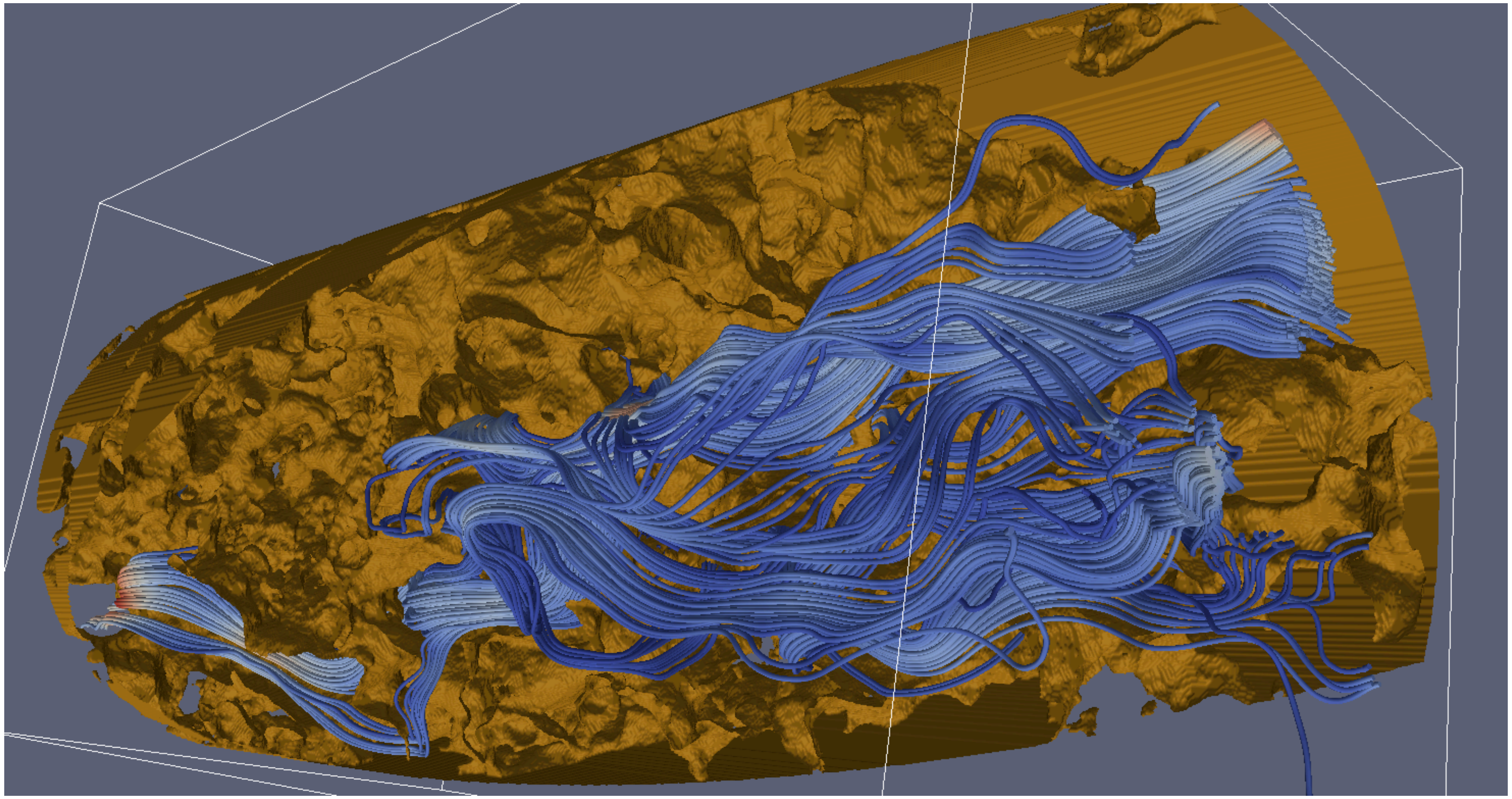


Image: Carson Brownlee, TACC. Data: Michael Sukop, FIU

www.sci.utah.edu

Ray tracing

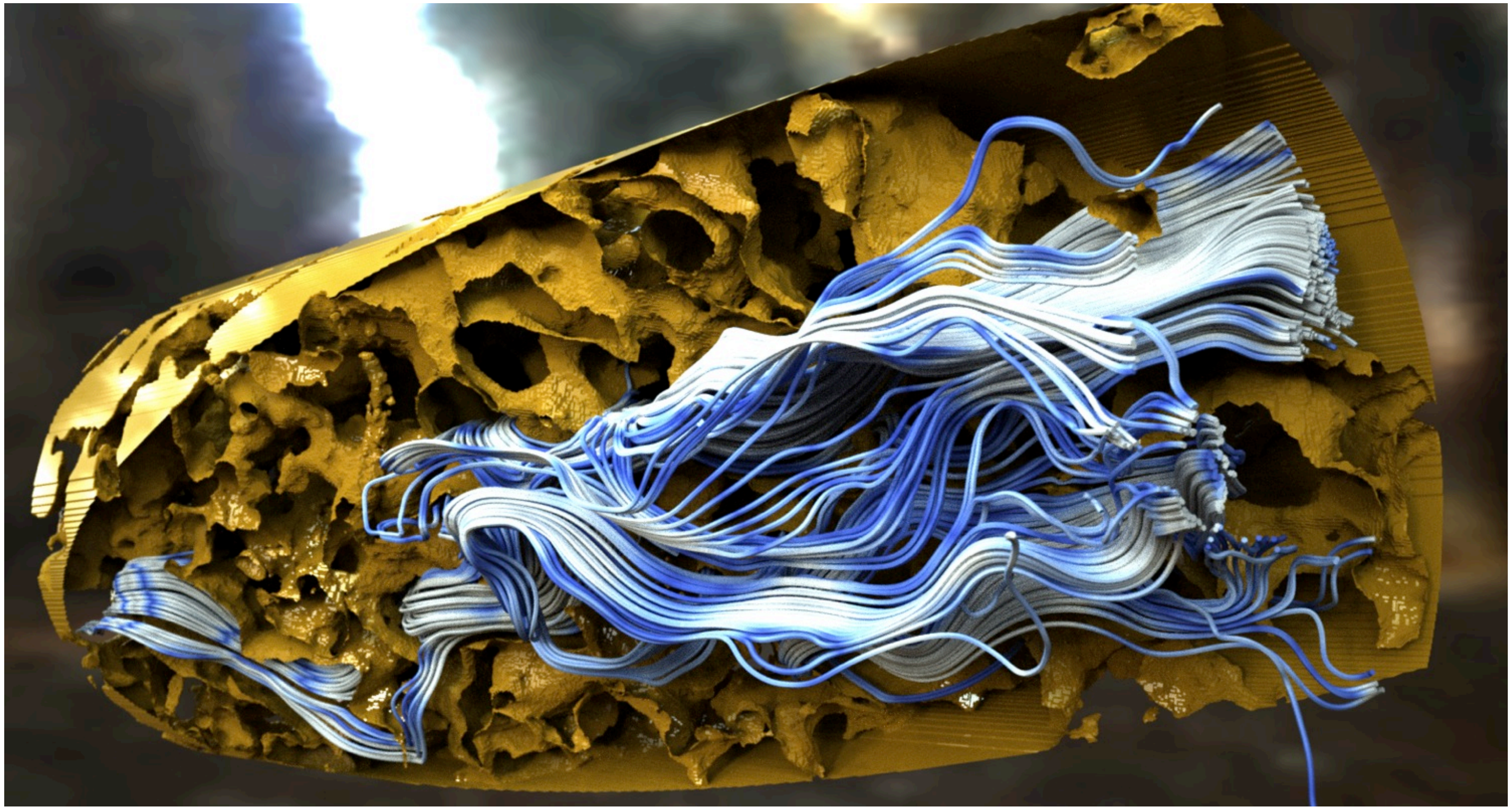
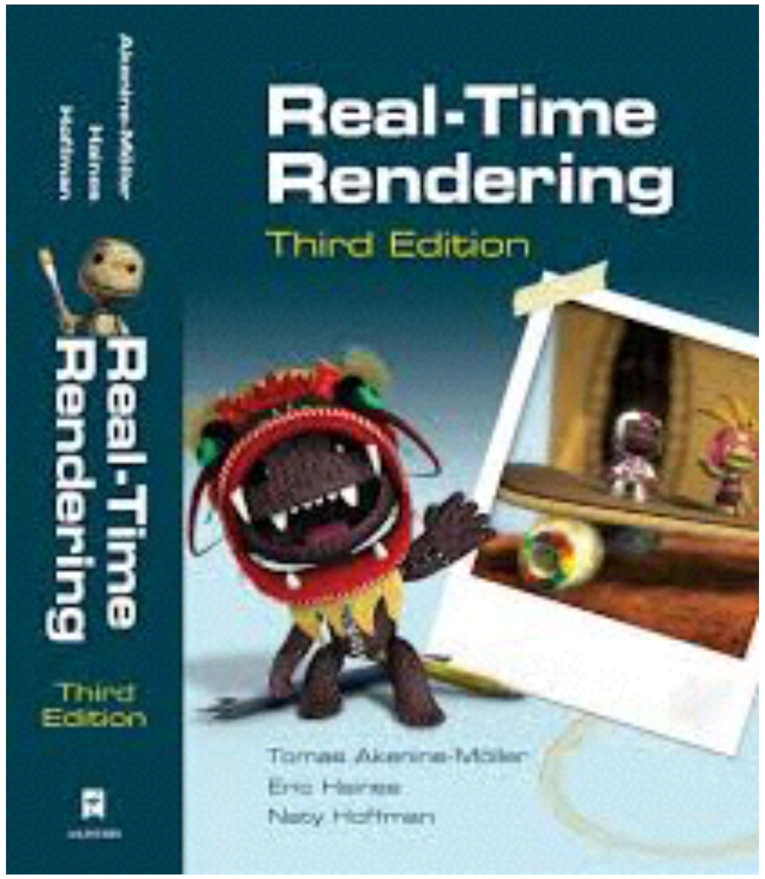
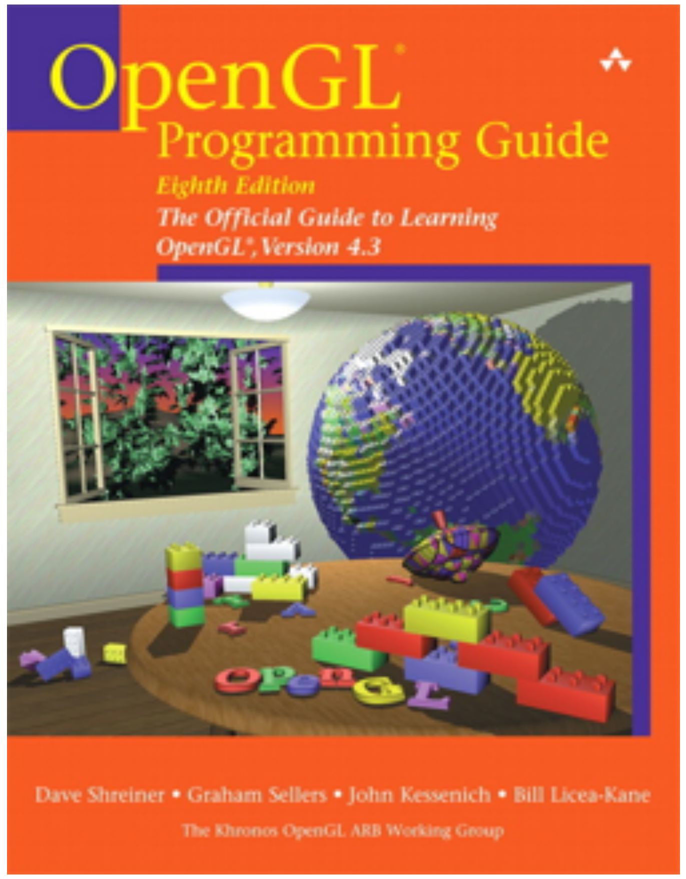


Image: Carson Brownlee, TACC. Data: Michael Sukop, FIU

Rasterization illustrated

http://acko.net/files/gltalks/pixelfactory/online.html#1




OpenGL Programming Guide
Eighth Edition
The Official Guide to Learning OpenGL, Version 4.3
Dave Shreiner • Graham Sellers • John Kessenich • Bill Licea-Kane
The Khronos OpenGL ARB Working Group

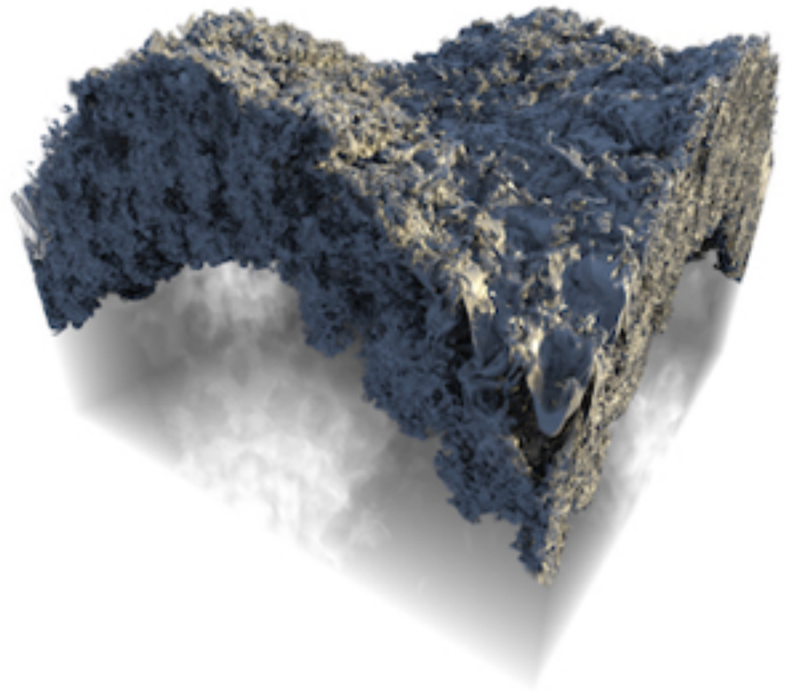
http://www.realtimerendering.com
http://openglbook.com/the-book.html

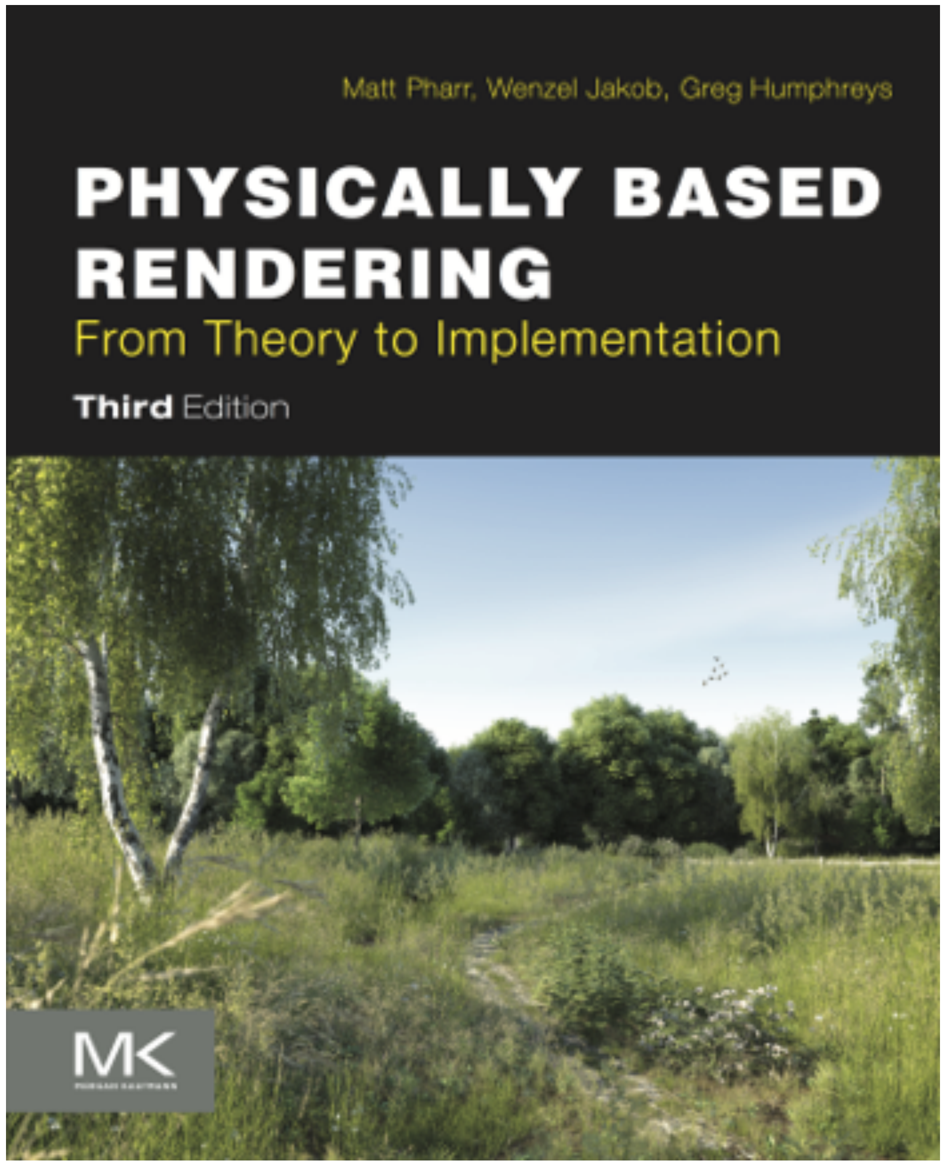
Relevant classes: CS 5600, CS 5610/6610

Ray tracing

http://www.ospray.org

https://developer.nvidia.com/optix

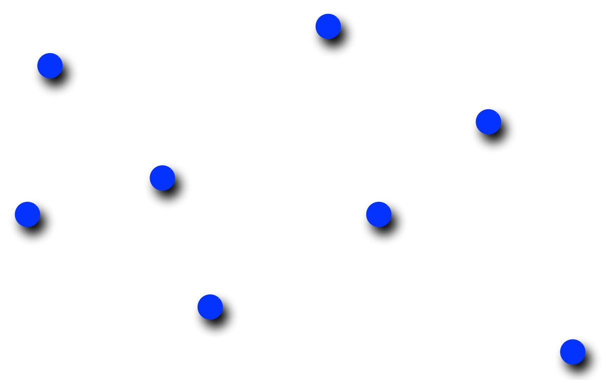



Matt Pharr, Wenzel Jakob, Greg Humphreys
PHYSICALLY BASED RENDERING
From Theory to Implementation
Third Edition
MK

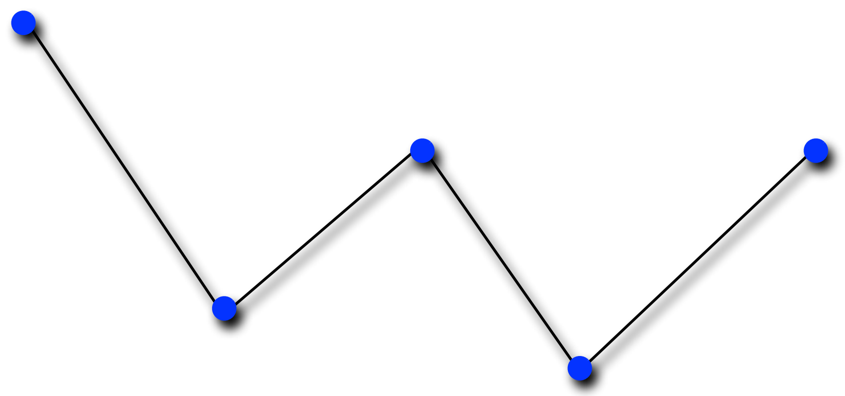
Relevant class: CS 5620/6620

www.sci.utah.edu

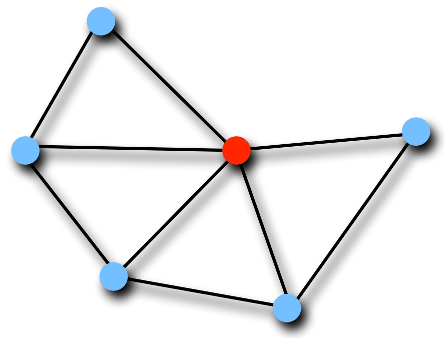
Graphics Primitives



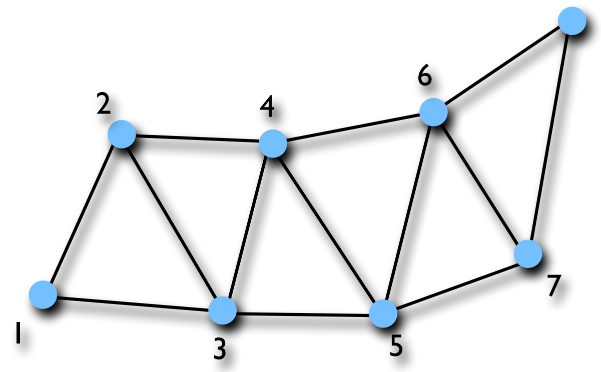
Points



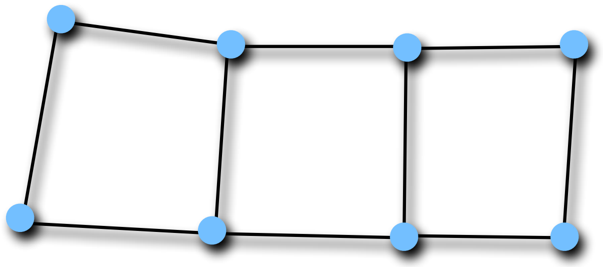
Lines



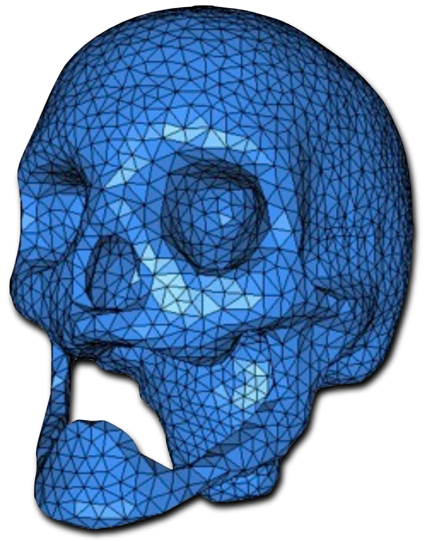
triangle fan



triangle strip



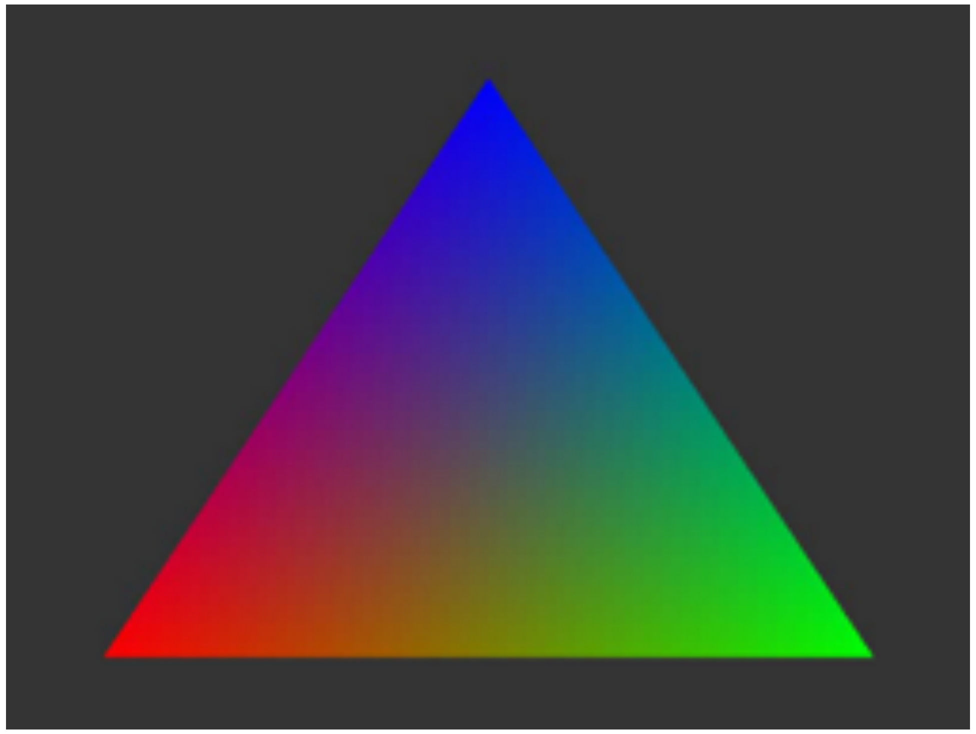
quad strip



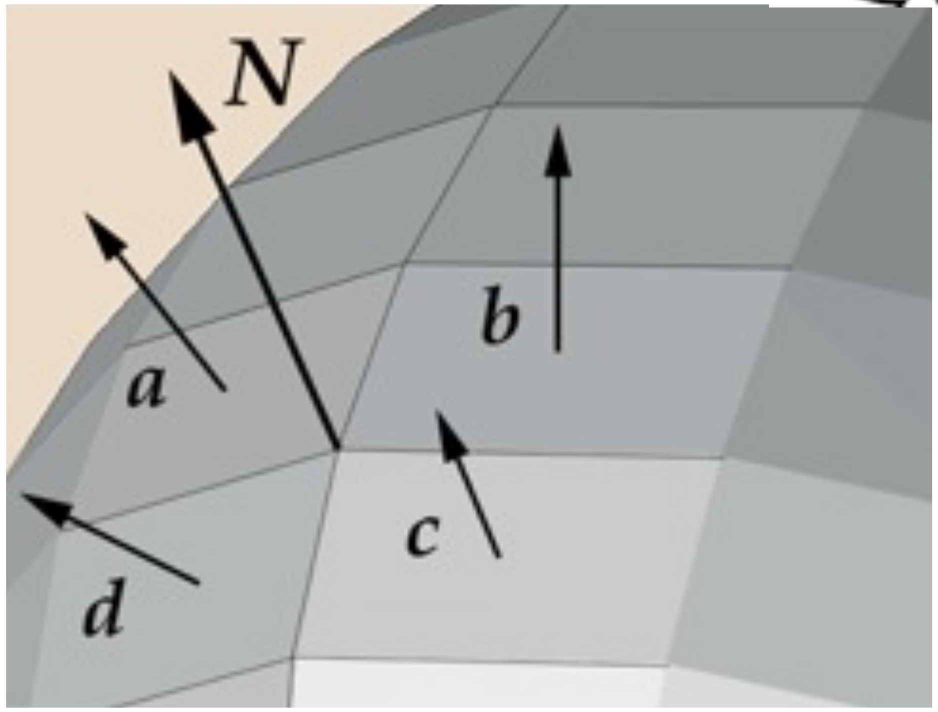
Surfaces

Primitive Attributes

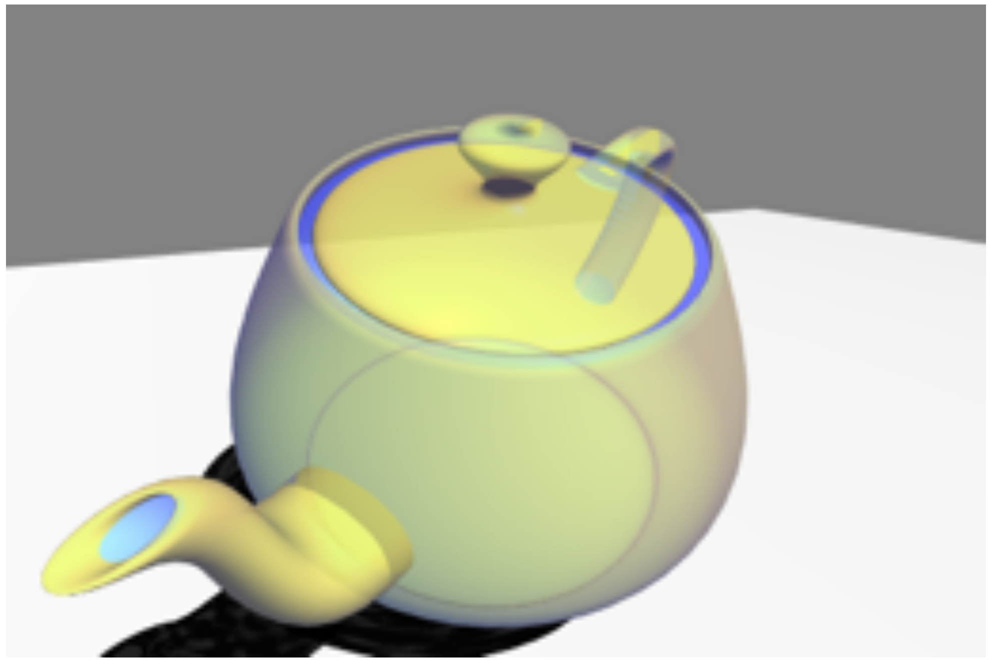
Color

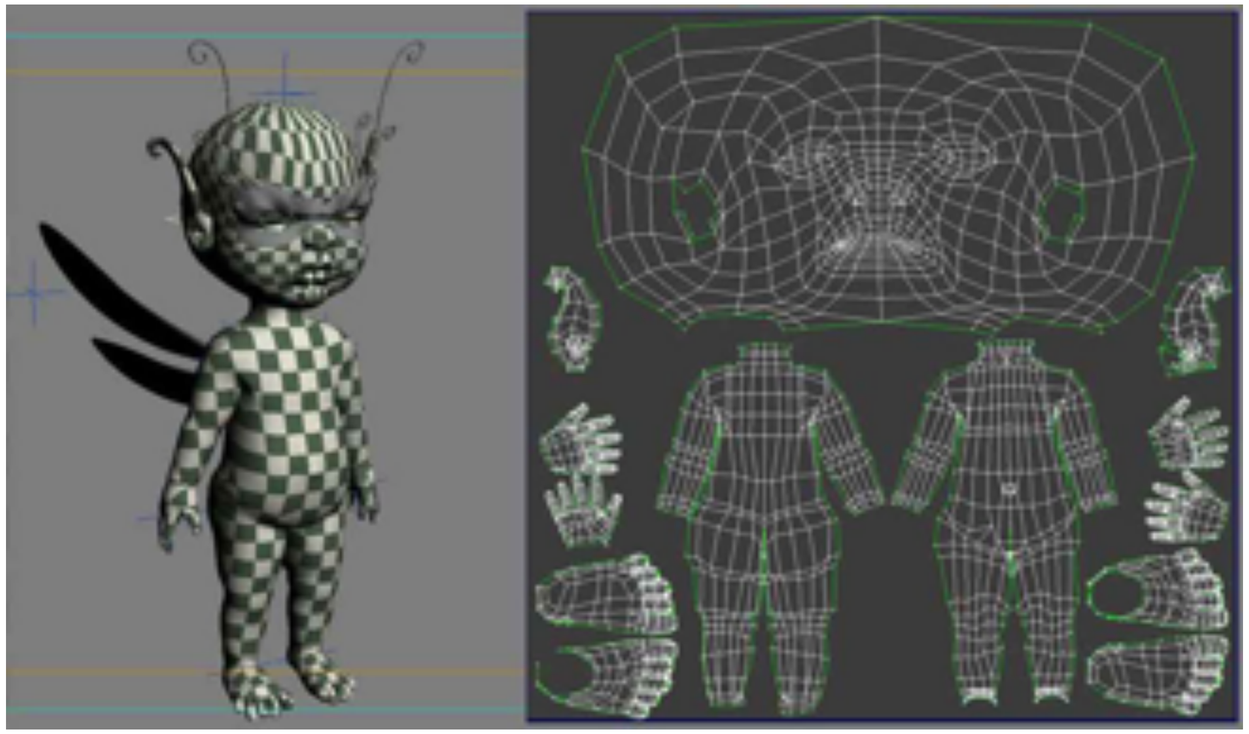


Normal



Opacity

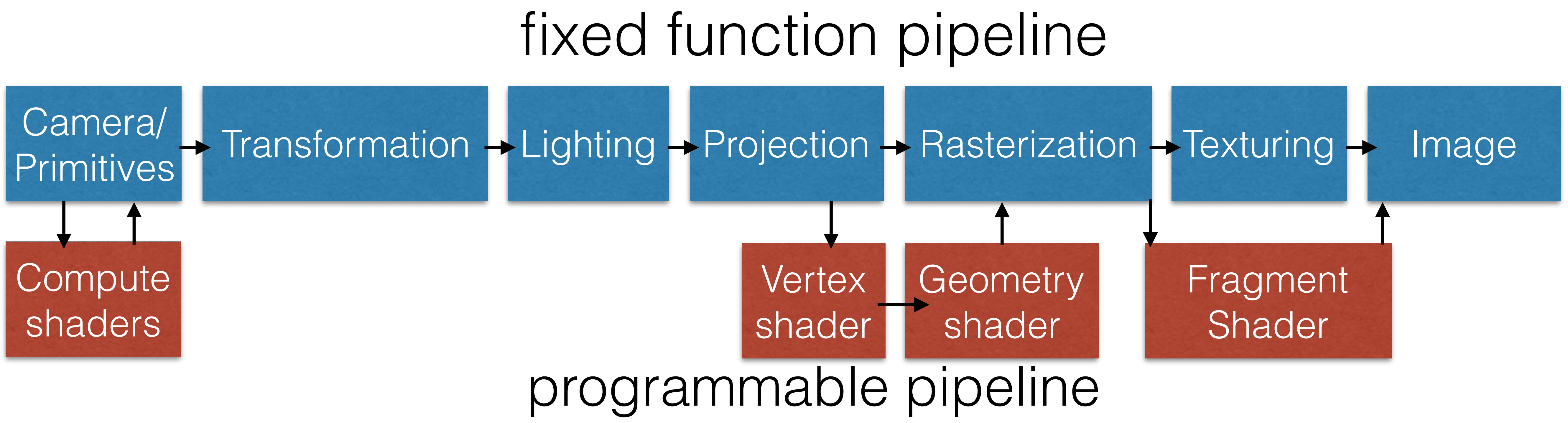




Texture coordinates

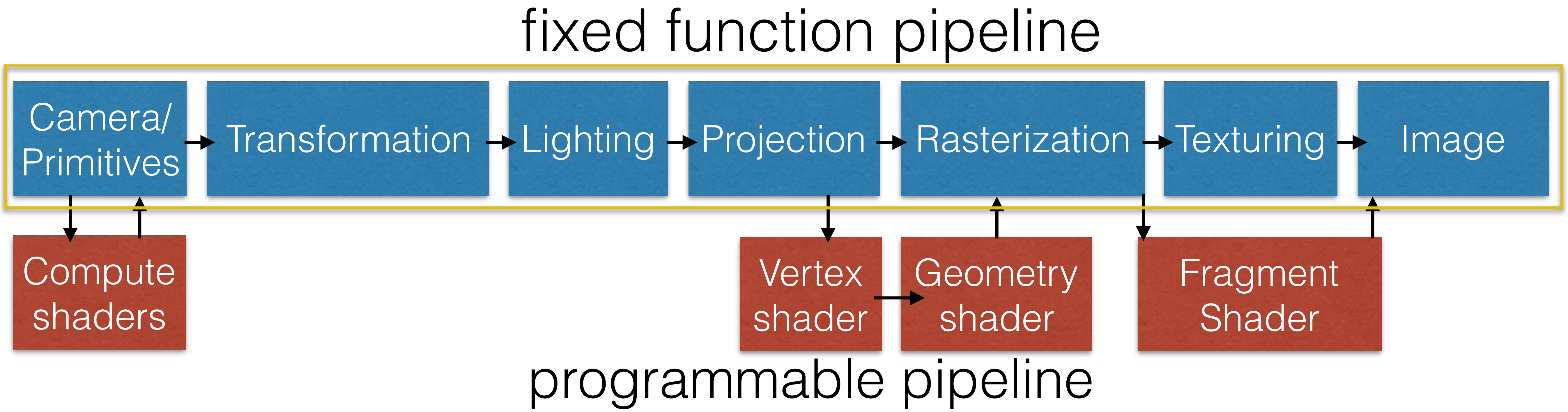
slides: IU Purdue

The rasterization pipeline



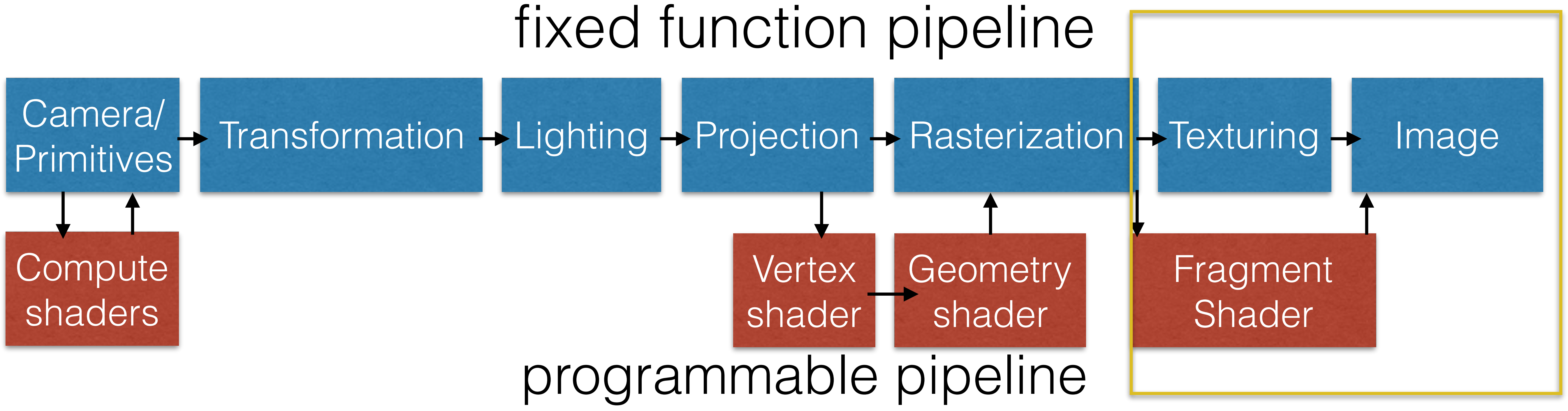
The rasterization pipeline

(e.g., indirect visualization with rasterization)



The rasterization pipeline

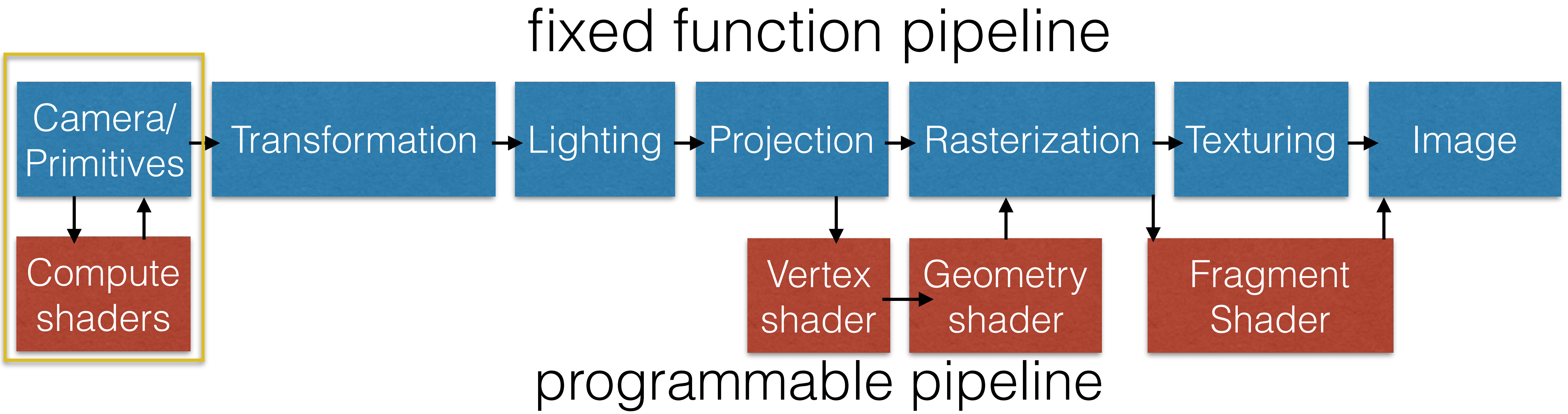
(e.g. direct visualization GPU ray casting)



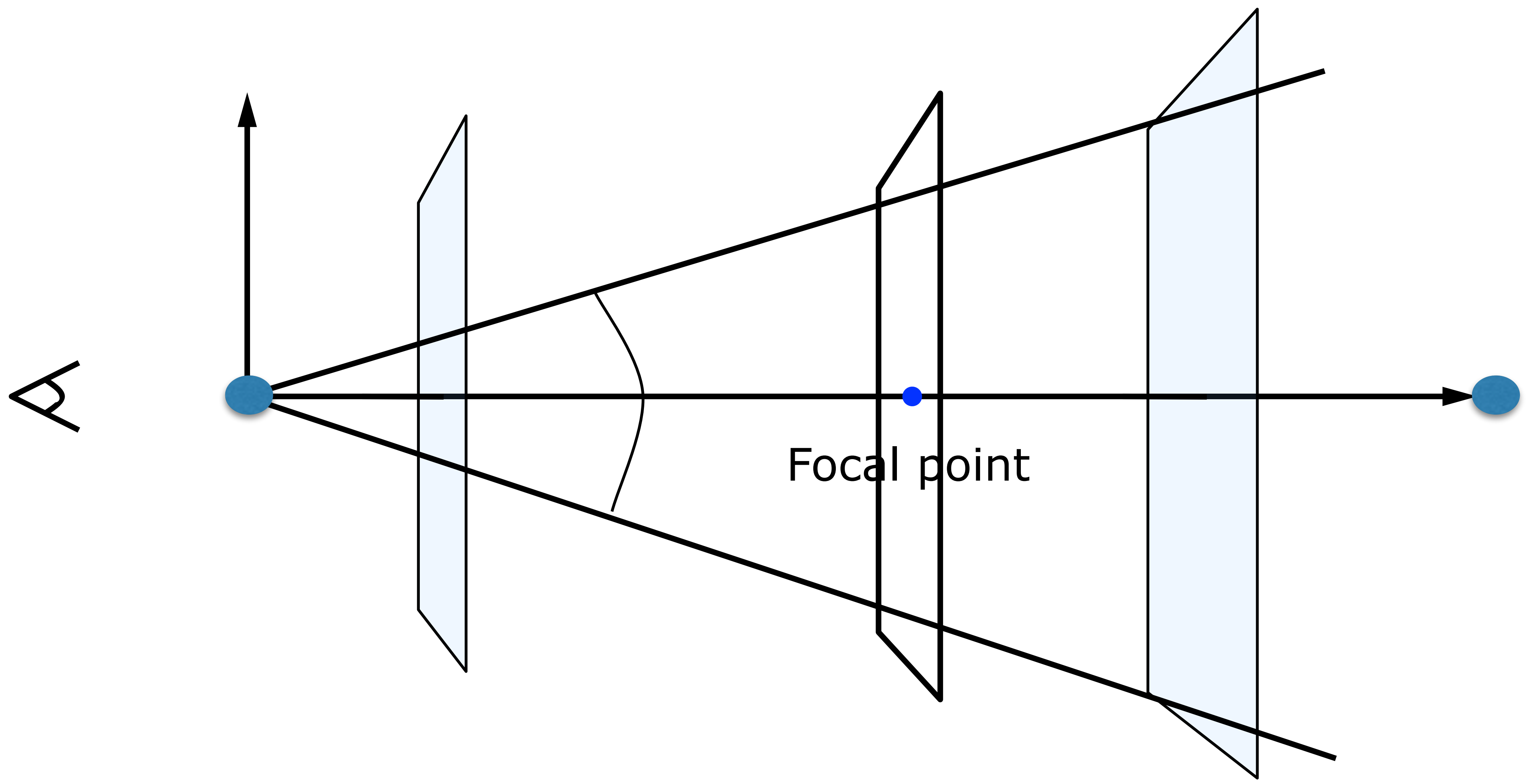
www.sci.utah.edu

The rasterization pipeline

(Much easier way to do direct visualization, but can't do it in WebGL yet)

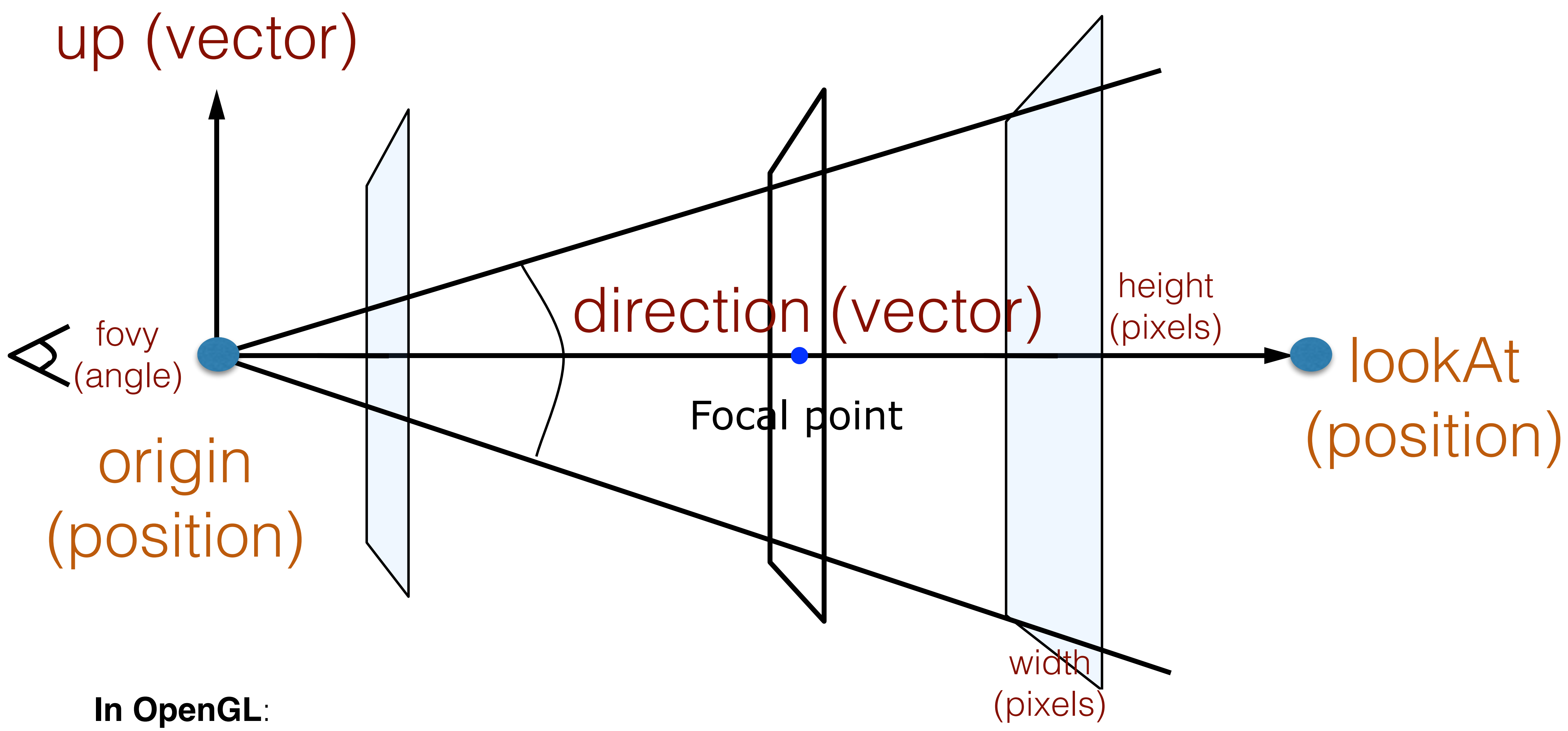


Camera



- Need to specify eye position, eye direction, and eye orientation (or “up” vector)
- This information defines a transformation from world coordinates to camera coordinates

Camera



In OpenGL:

```
vec3 origin, direction, up;
float fovy, aspect;
int width, height;
```

```
glViewport (width, height);
gluPerspective(width, height, fov, near, far);
gluLookAt(origin, direction, up);
```

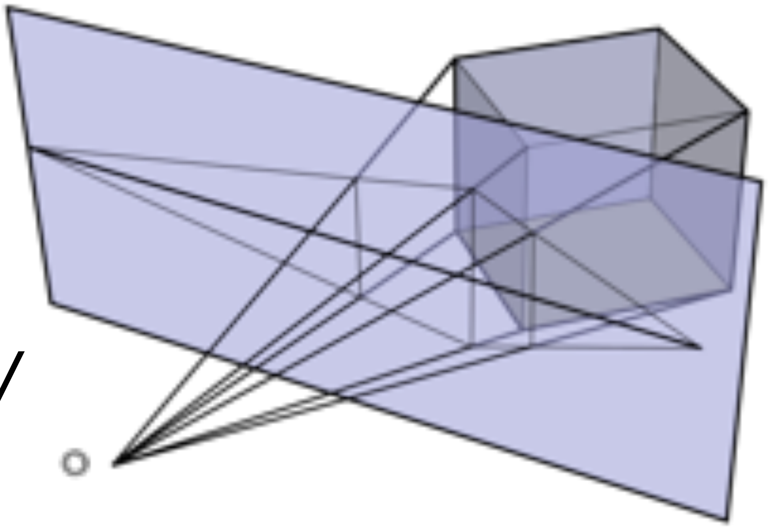
In three.js (done for you in HW6):

```
_gl.viewport( _viewportX, _viewportY, _viewportWidth, _viewportHeight );
var cameraPX = new THREE.PerspectiveCamera( fov, aspect, near, far );
```

Projections

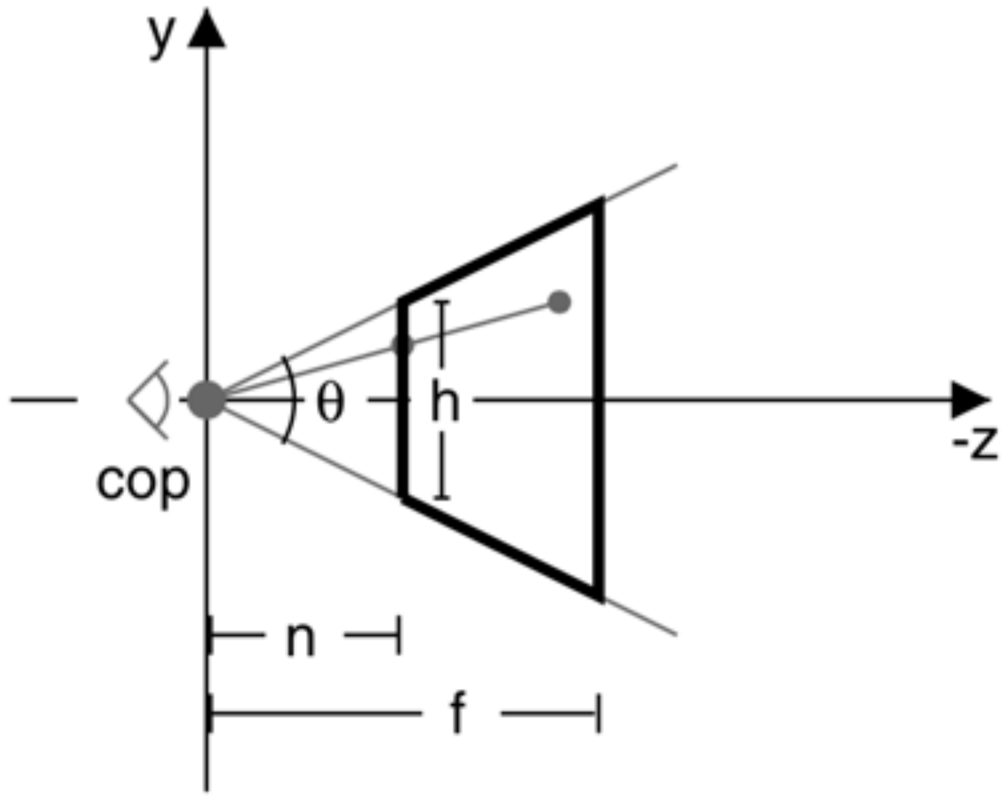
Perspective projection

- parallel lines do not necessarily remain parallel
- objects get larger as they get closer
- fly-through realism



Perspective Projection

- Maps points in 4D (where it is easier to define the view volume and clipping planes) to positions on the 2D display through multiplication and *homogenization*

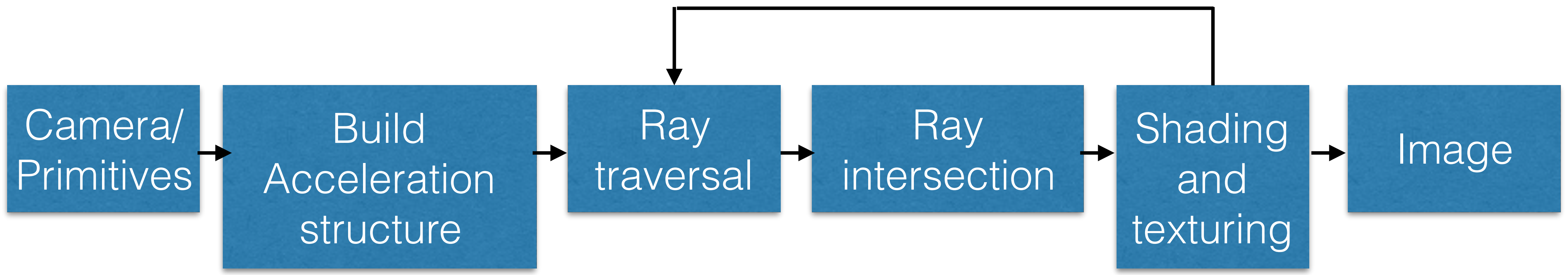


$$M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (n+f)/n & -f \\ 0 & 0 & 1/n & 0 \end{bmatrix}.$$

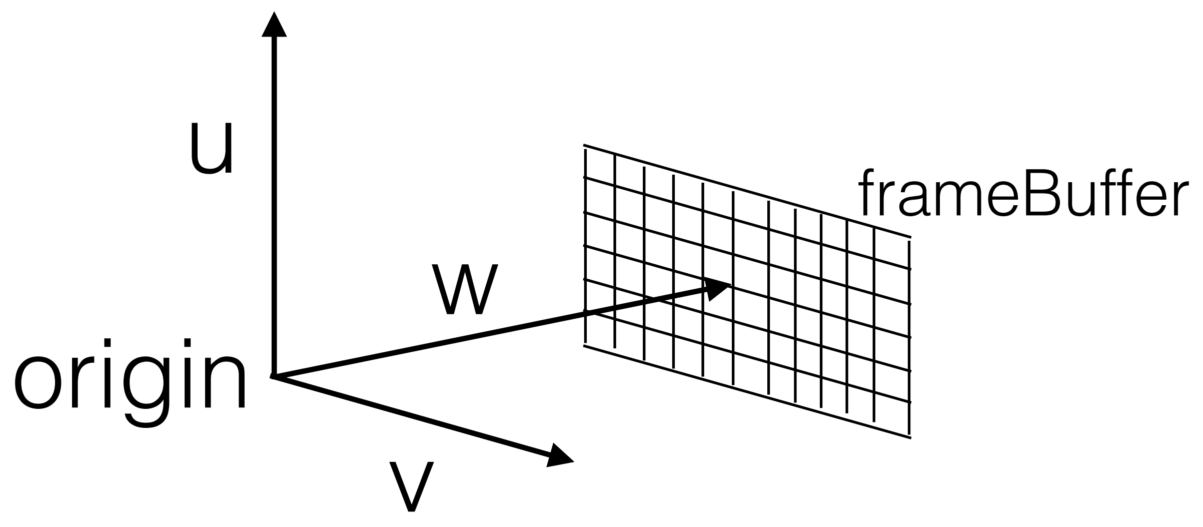
$$M_p\bar{\mathbf{x}} = \begin{bmatrix} x \\ y \\ z(\frac{n+f}{n}) - f \\ z/n \end{bmatrix}$$

Ray tracing pipeline

(direct visualization with ray tracing)



Pinhole camera (GPU ray casting)



- **Camera setup (per frame, on the CPU):**

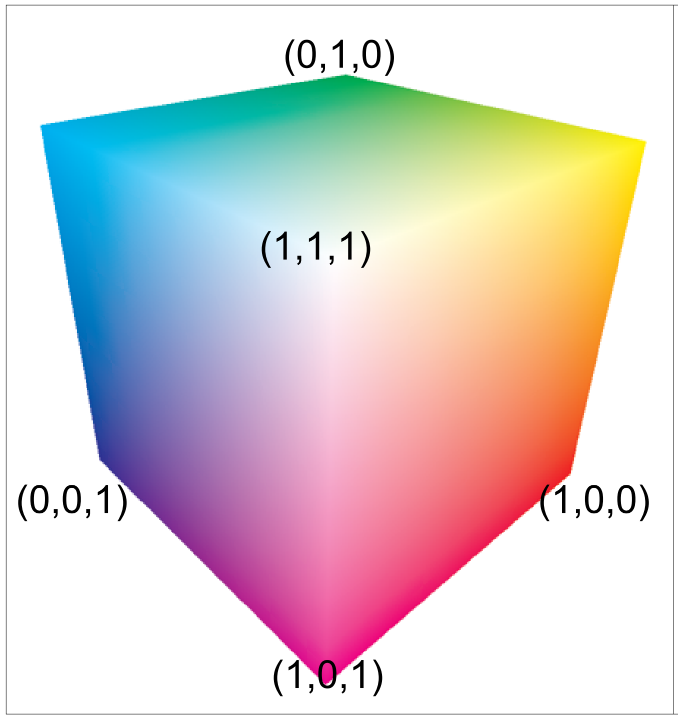
```
vec3 u,v,w;
w = normalize(lookAt - origin);
u = cross(up, w);
v = cross(w,u);
u = normalize(u);
v = normalize(v);

float tanThetaOver2 = tanf(fovy * .5 * PI / 180);
float aspect = width / height;

vec3 frameBuffer_u = u * tanThetaOver2;
vec3 frameBuffer_v = v * tanThetaOver2 / aspect;
```

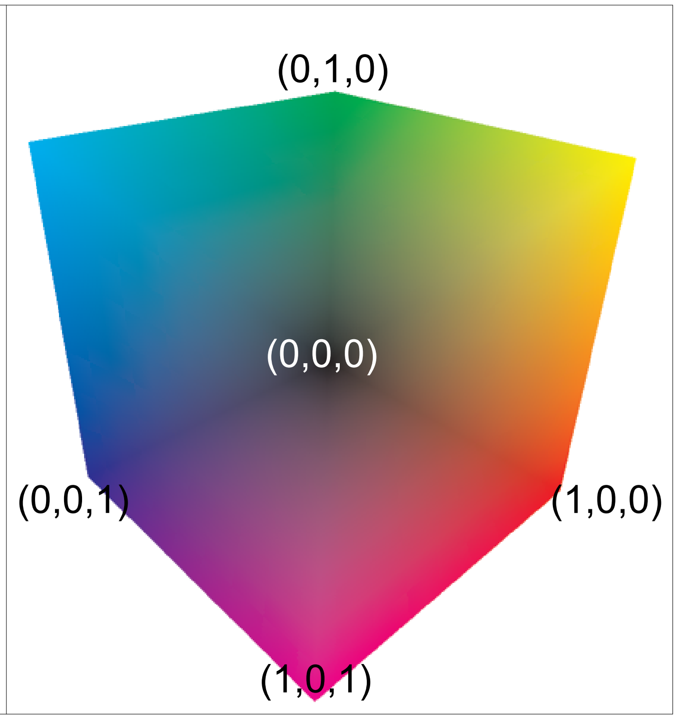
- **Ray generation (per pixel, e.g. in a fragment shader on the GPU, or task per pixel):**

```
varying vec2 pixelPos;   //computed by the vertex shader stage
uniform vec3 origin, w, frameBuffer_u, frameBuffer_v;  //set by the user
vec3 ray_dir = w + (frameBuffer_u * pixelPos.x) + (frameBuffer_v * pixelPos.y);
```

www.sci.utah.edu

Ray casting within a rasterizer





front faces back faces

- Rasterize a 3D bounding box on [0,1]^3.

- **Fragment shader first pass (front faces):**

```
varying vec3 worldSpaceCoords;   //world space coordinates of front faces, from vertex shader
gl_FragColor = vec4( worldSpaceCoords.x , worldSpaceCoords.y, worldSpaceCoords.z, 1 );
```

- ***Fragment shader second pass (back faces):***

```
varying vec3 worldSpaceCoords;   //world space coordinates of back faces, from vertex shader
varying vec4 projectedCoords;   //projected coordinates of this pixel
//Transform the coordinates it from [-1;1] to [0;1]
vec2 texc = vec2(((projectedCoords.x / projectedCoords.w) + 1.0 ) / 2.0, ((projectedCoords.y / projectedCoords.w) + 1.0 ) / 2.0 );
//The back position is the world space position stored in the texture.
vec3 backPos = texture2D(tex, texc).xyz;
//The front position is the world space position of the second render pass.
vec3 frontPos = worldSpaceCoords;
vec3 dir = backPos - frontPos;
```

Krueger and Westermann. Acceleration Techniques for GPU Volume Rendering. IEEE Visualization 2003.

www.sci.utah.edu

Shading

www.sci.utah.edu

Shading

- Shading reveals the shape of 3D objects through their interaction with light
- Shading creates colors as a function of:
  - surface properties
  - surface normals
  - lights
- Rich subject (*we are only interested in basics here*)
- Surfaces show information, lights show surfaces, shading controls how

Shading

Phong lighting model (1975)

- Specular reflection
- Diffuse reflection
- Ambient reflection

Light intensity per light source and per color channel

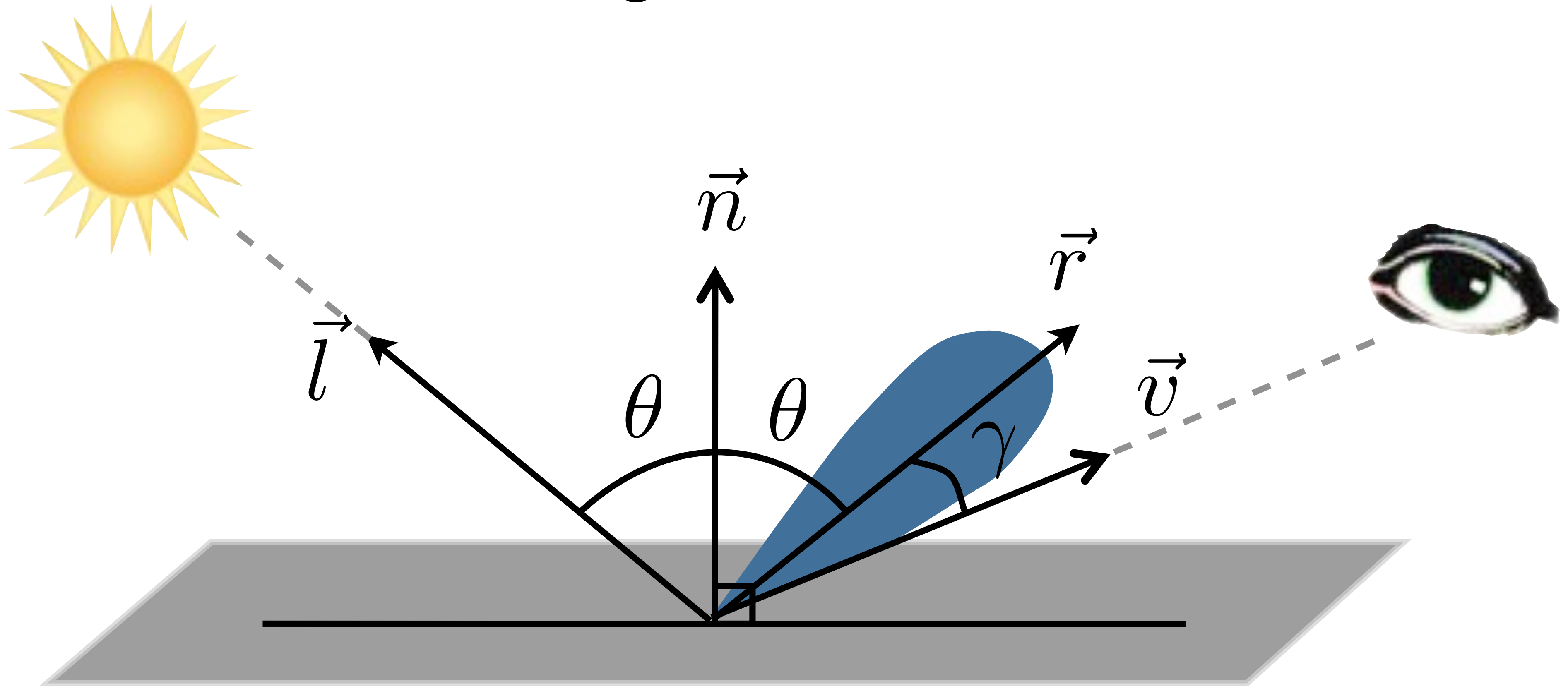
$$I = \underbrace{k_s}_{} I_s + \underbrace{k_d}_{} I_d + \underbrace{k_a}_{} I_a$$

relative contributions
(material specific)

Shading

Phong lighting model: Specular Reflection

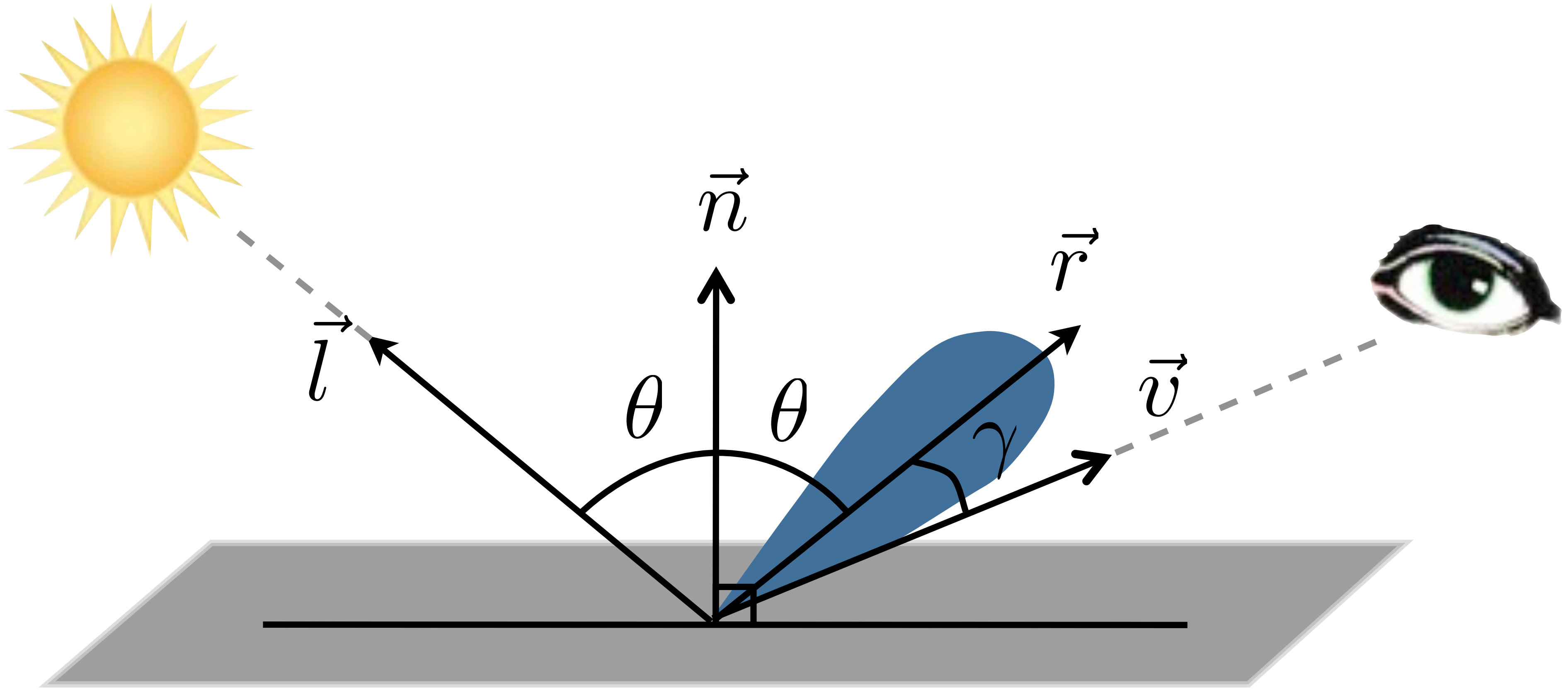
- mirror-like surfaces
- Specular reflection depends on position of the observer relative to light source and surface normal



Shading

Phong lighting model: Specular Reflection

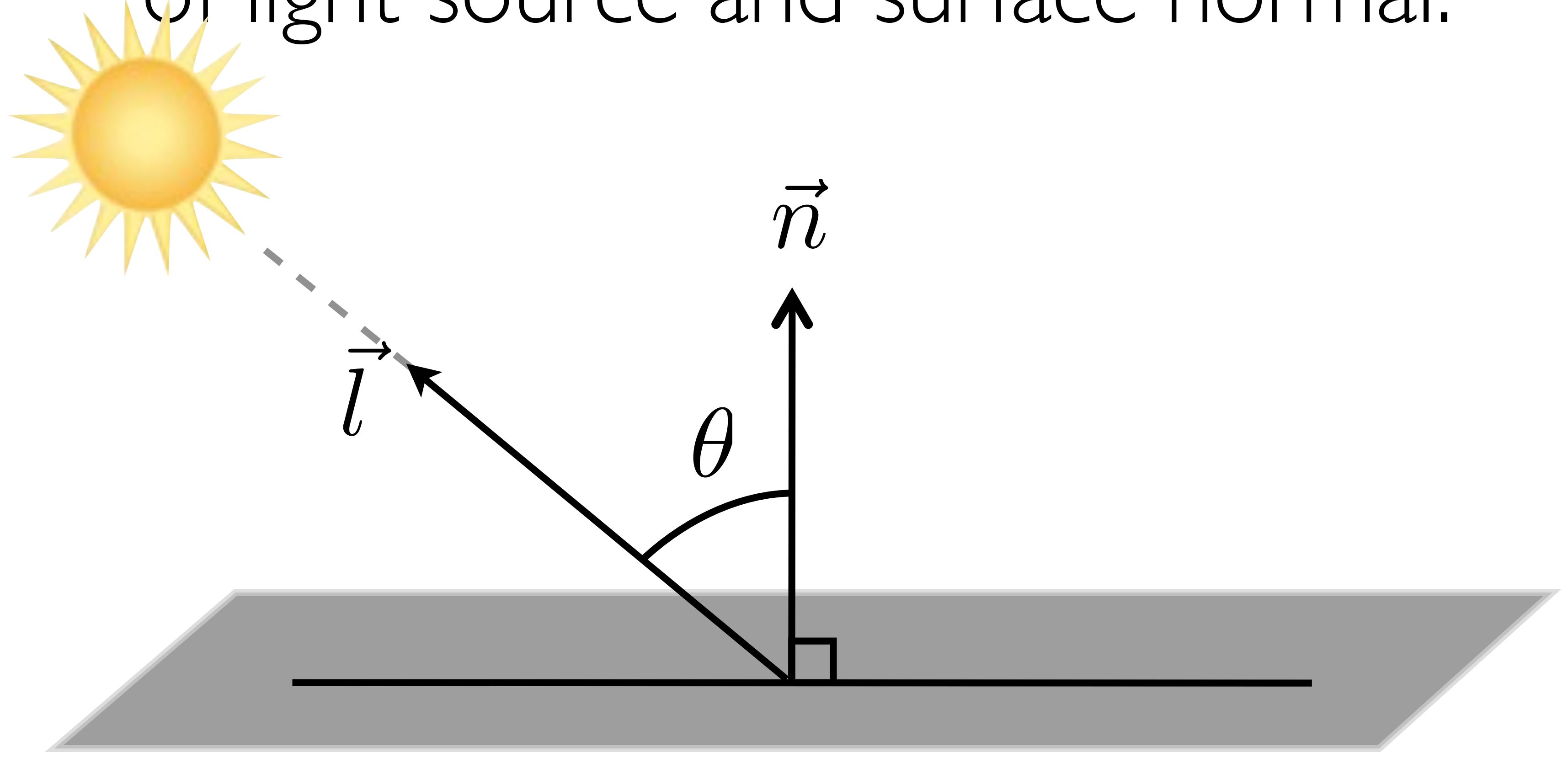
$$\boxed{I_s = I_i \cos^n \gamma = I_i \cos^n < \vec{r}, \vec{v} >}$$

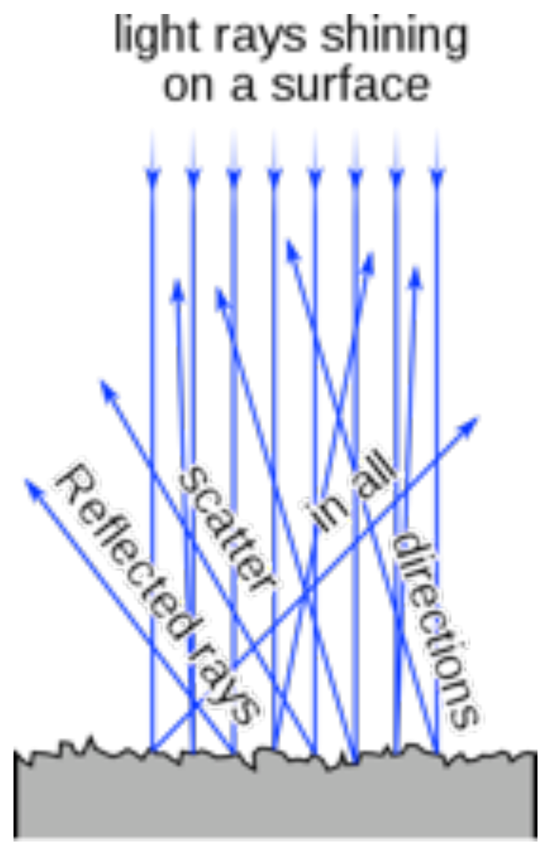


Shading

Phong lighting model: Diffuse Reflection

- Non-shiny surfaces
- Diffuse reflection depends only on relative position of light source and surface normal.

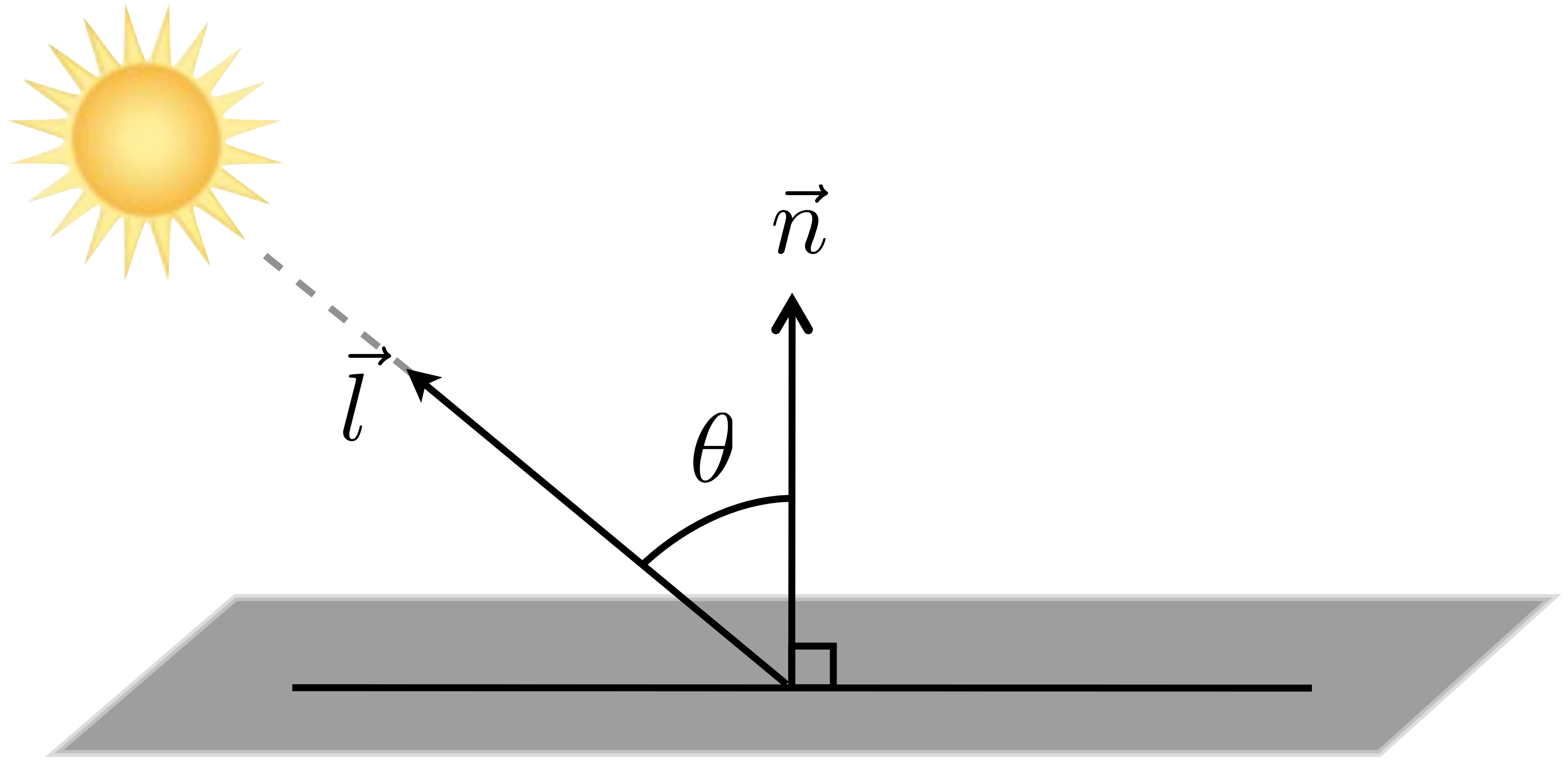


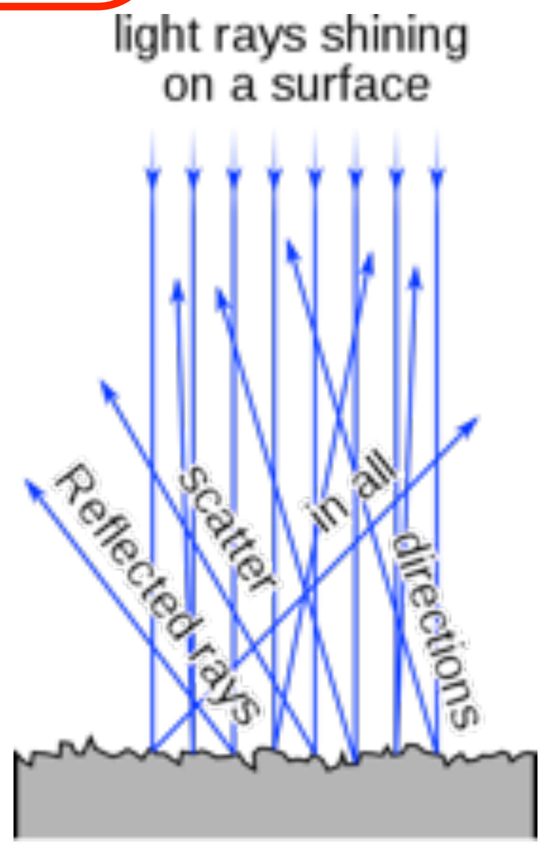


Shading

Phong lighting model: Diffuse Reflection

$$\boxed{I_d = I_i \cos\theta = I_i \ (\vec{l} \cdot \vec{n})}$$





Shading

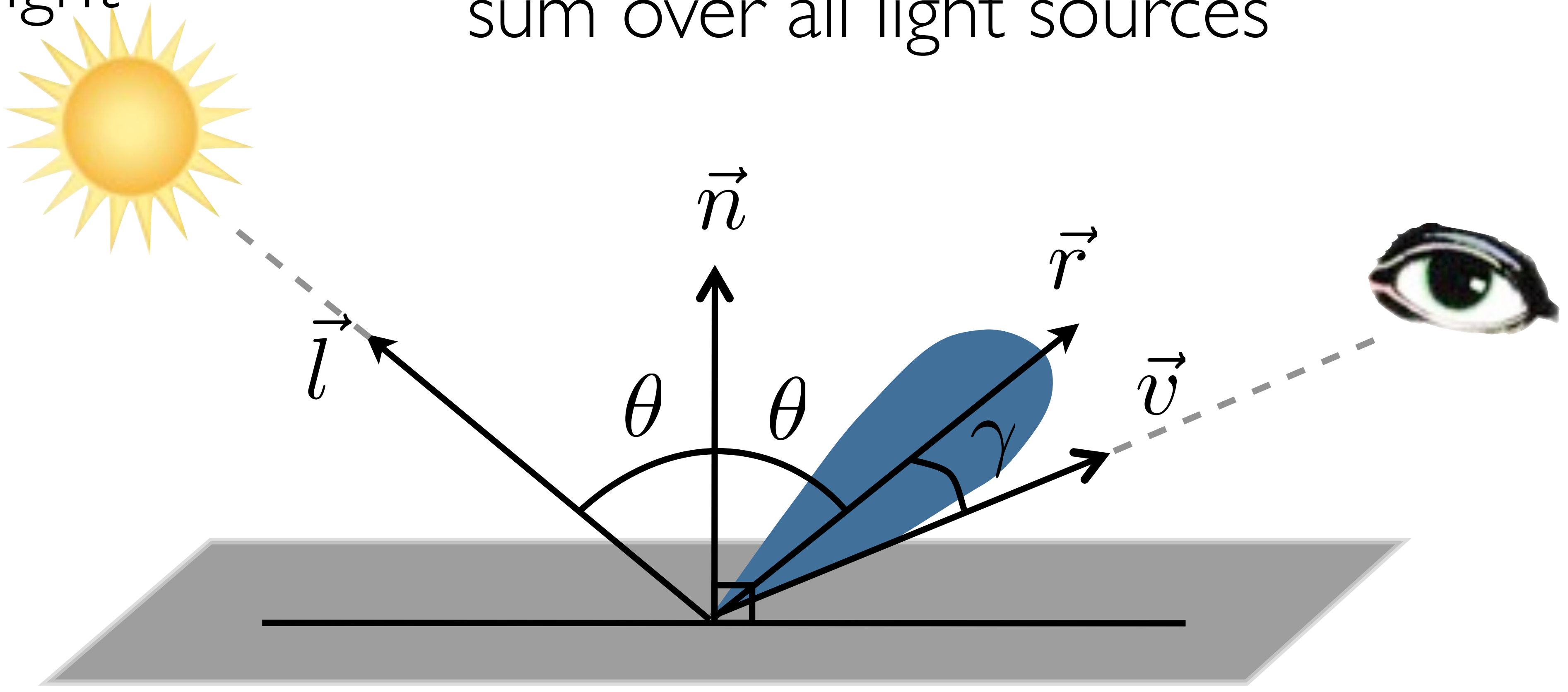
Phong lighting model

per color channel

$$I = k_a I_a + \sum_{i=1}^{N} I_i \left(k_d \cos(\theta) + k_s \cos^n(\gamma)\right)$$

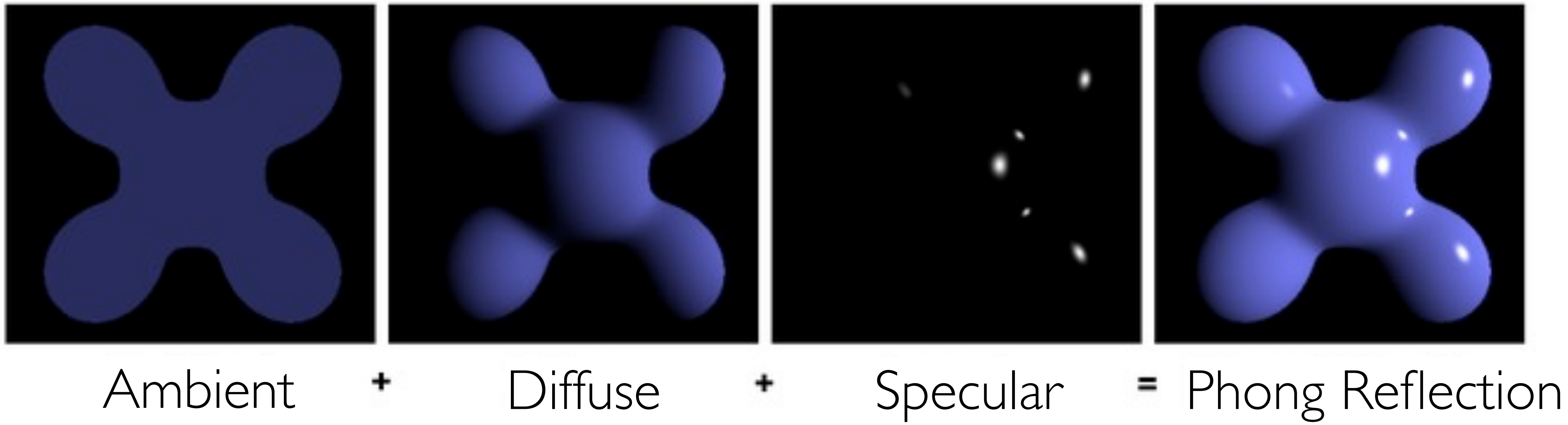
ambient light

sum over all light sources



Shading

Phong lighting model



http://en.wikipedia.org/wiki/Phong_shading

Gradient and Phong shading code

```
vec3 gradient(vec3 psample, float value)
{
  float dcd = .001;
  vec3 p = psample;
  vec3 grad;

  p.x -= dcd;
  grad.x = sampleAs3DTexture(p);
  p.x = psample.x + dcd;
  grad.x -= sampleAs3DTexture(p);
  p.x = psample.x;

  p.y -= dcd;
  grad.y = sampleAs3DTexture(p);
  p.y = psample.y + dcd;
  grad.y -= sampleAs3DTexture(p);
  p.y = psample.y;

  p.z -= dcd;
  grad.z = sampleAs3DTexture(p);
  p.z = psample.z + dcd;
  grad.z -= sampleAs3DTexture(p);

  return normalize(grad);
}
```

Smooth, interpolated normal at an arbitrary point

```
vec3 shade(vec3 material_color, vec3 p, float value, vec3 dir)
{
  vec3 normal = gradient(p, value);
  vec3 light_direction = -normalize(dir);

  vec3 v = -normalize(dir);
  float n_dot_v = dot(normal, v);

  if (n_dot_v < 0.0)
    normal = -normal;

  float n_dot_l = dot(normal, light_direction);
  vec3 color = .15 * vec3(1.0,1.0,1.0);

  if (n_dot_l > 0.0)          //diffuse
  {
    vec3 diffuse;
    diffuse = vec3(min(max(n_dot_l, 0.0), 1.0));
    color += diffuse * material_color;

    //specular
    vec3 half_vector = normalize(v + light_direction);
    float n_dot_h = max( dot(normal, half_vector),0.0);
    color += vec3(pow( n_dot_h, 32.0 ));
 }

  return color;
}
```

Shades your sample “p” like it’s on a surface!

www.sci.utah.edu

The Rendering Equation

$$I(x, x') = g(x, x') \left[\epsilon(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx'' \right]$$

$I(x, x')$	is the related to the intensity of light passing from point x' to point x
$g(x, x')$	is a "geometry" term
$\epsilon(x, x')$	is related to the intensity of emitted light from x' to x
$\rho(x, x'x'')$	is related to the intensity of light scattered from x'' to$x$ by a patch of surface at x'

- James Kajiya, "The Rendering equation", Siggraph 1986. Generalizes all light transport in graphics into one equation!
- Phong shading, the "Utah approximation"

$$I = g\epsilon + gM\epsilon_0$$

- Ray tracing, i.e. Whitted 1980

$$I = g\epsilon + gM_0 g\epsilon_0 + gM_0 gM_0 g\epsilon_0 + \cdots$$

- Radiosity, i.e. Goral et al. 1984

$$dB(x') = \pi[\epsilon_0 + \rho_0 H(x')] dx'$$

- Volume rendering (e.g. Sabella 1988, Kniss et al. "Gaussian Transfer Functions for Multifield Visualization", Vis 03)

$$I(a,b) = \int_a^b C\rho(v(u))\, e^{-\int_a^u \tau\rho(v(t))dt} du$$

www.sci.utah.edu

The Rendering Equation

$$I(x,x') = g(x,x')\left[\epsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')dx''\right]$$

$I(x,x')$	is the related to the intensity of light passing from point x' to point x
$g(x,x')$	is a "geometry" term
$\epsilon(x,x')$	is related to the intensity of emitted light from x' to x
$\rho(x,x'x'')$	is related to the intensity of light scattered from x'' to$x$ by a patch of surface at x'

- James Kajiya, "The Rendering equation", Siggraph 1986. Generalizes all light transport in graphics into one equation!
- Phong shading, the "Utah approximation"

$$I = g\epsilon + gM\epsilon_0$$

- Ray tracing, i.e. Whitted 1980

$$I = g\epsilon + gM_0 g\epsilon_0 + gM_0 gM_0 g\epsilon_0 + \cdots$$

- Radiosity, i.e. Goral et al. 1984

$$dB(x') = \pi[\epsilon_0 + \rho_0 H(x')]dx'$$

- Volume rendering (e.g. Sabella 1988, Kniss et al. "Gaussian Transfer Functions for Multifield Visualization", Vis 03)

$$I(a,b) = \int_a^b C\rho(v(u))\, e^{-\int_a^u \tau\rho(v(t))dt}du$$

www.sci.utah.edu

Alpha blending

Fundamental Algorithms

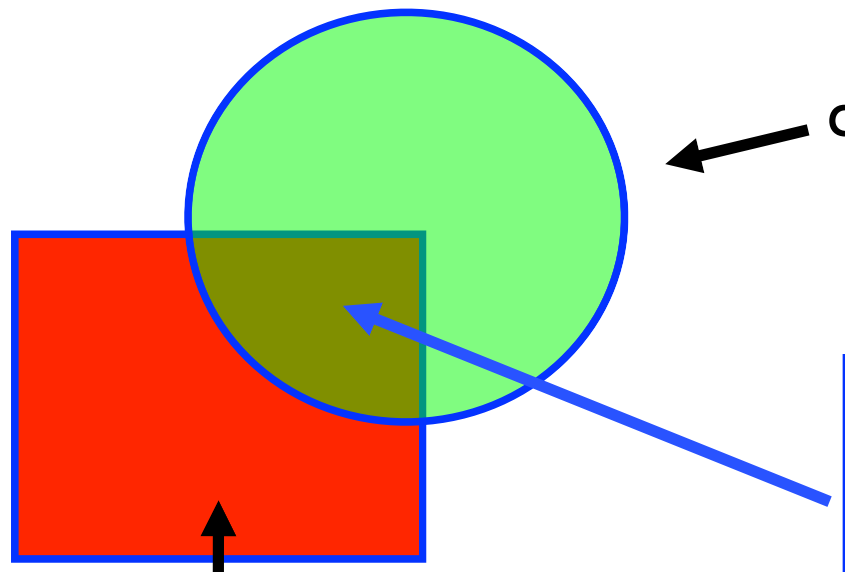
Alpha blending / compositing

- Approximate visual appearance of semi-transparent object in front of another object
- Implemented with OVER operator

Fundamental Algorithms

Alpha blending / compositing

- Approximate visual appearance of semi-transparent object in front of another object
- Implemented with OVER operator



$$c = af*cf + (1 - af)*ab*cb$$
$$a = af + (1 - af)*ab$$

$$c = (0.54,0.4,0)$$
$$a = 0.94$$

Alpha blending code

```
float accumulatedAlpha = 0;
vec3 accumulatedColor = vec3(0,0,0);

//given new alphaSample, colorSample "behind" us, composite as follows:
void blend(vec3 colorSample, float alphaSample)
{
  float ax1msa = (1.0 - accumulatedAlpha) * alphaSample;
  accumulatedColor += ax1msa * colorSample;
  accumulatedAlpha += ax1msa;
}
```

Volume rendering is just doing this over and over again in a loop!

www.sci.utah.edu

Volume rendering

www.sci.utah.edu

The indirect-direct spectrum

isosurface extraction (marching cubes)
+ rasterization

direct isosurface ray casting

segmentation+filtering+
classification+rasterization pipeline

volume rendering
from raw data

polygonal

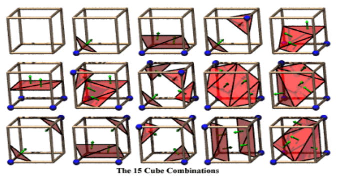
splatting

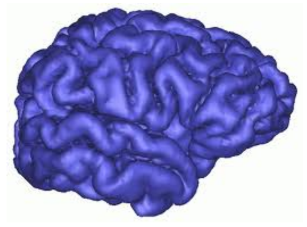
$\longleftrightarrow$

Indirect

$$\begin{tikzcd} \text{Data} \arrow[r] & \text{Filter} \arrow[r] & \text{Render} \end{tikzcd}$$

0	4	8	0
4	14	9	0
6	11	1	0
2	1	0	0

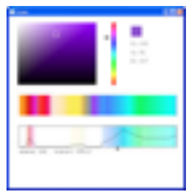


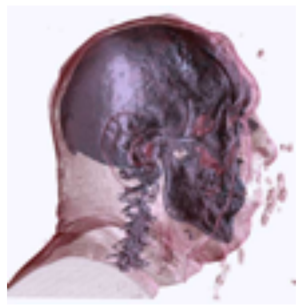


Direct

$$\begin{tikzcd} \text{Data} \arrow[r] & \text{Filter + Render} \end{tikzcd}$$

0	4	8	0
4	14	9	0
6	11	1	0
2	1	0	0

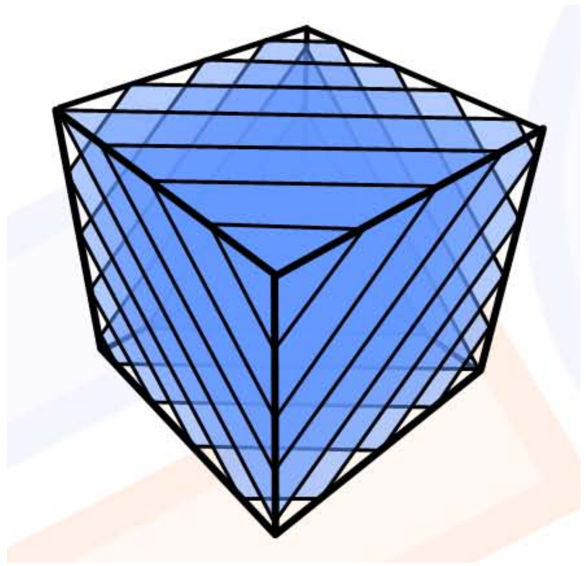




www.sci.utah.edu

Rasterization vs Ray Tracing for *volume visualization*

- Volume rendering can be implemented either via ray tracing (sampling along the ray) or rasterization (with textured proxy geometry)
- Other ways of doing "direct visualization" using the rasterization pipeline:
  - slicing
  - "splatting" and other proxy geometry
  - Ray tracing (or at least ray casting) increasingly common…


3D texture slicing

Rusinkiewicz & Levoy, "QSplat", Siggraph 2000

www.sci.utah.edu

Volume ray casting

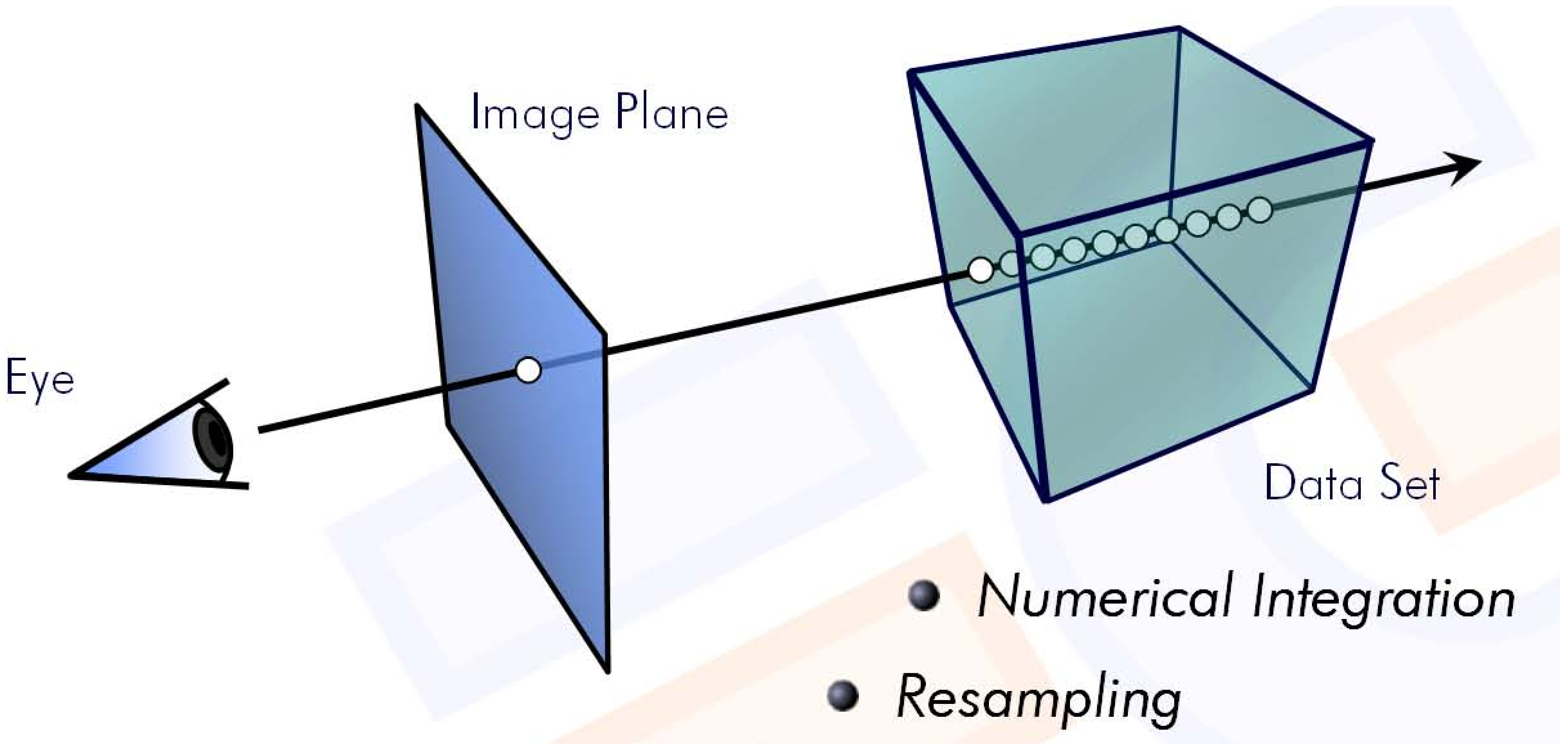
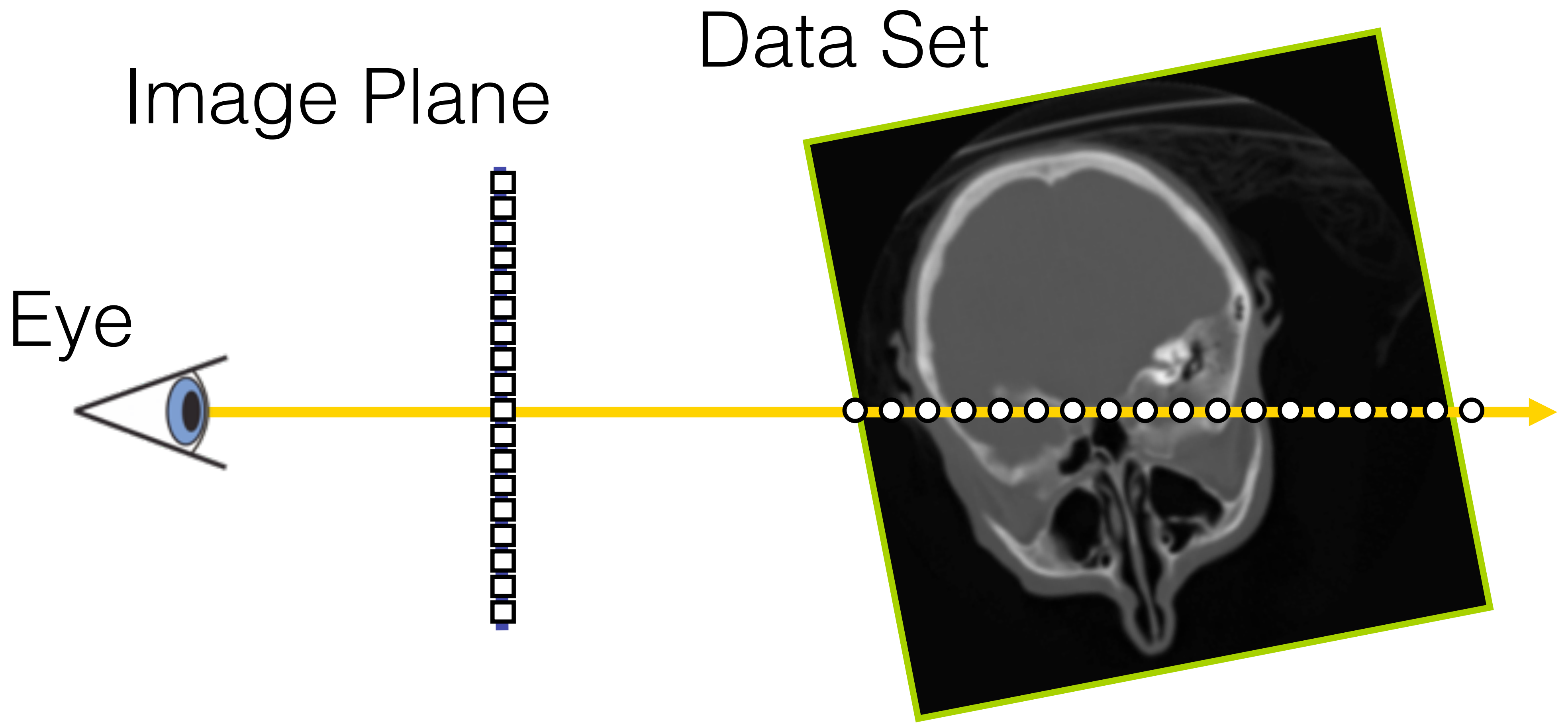
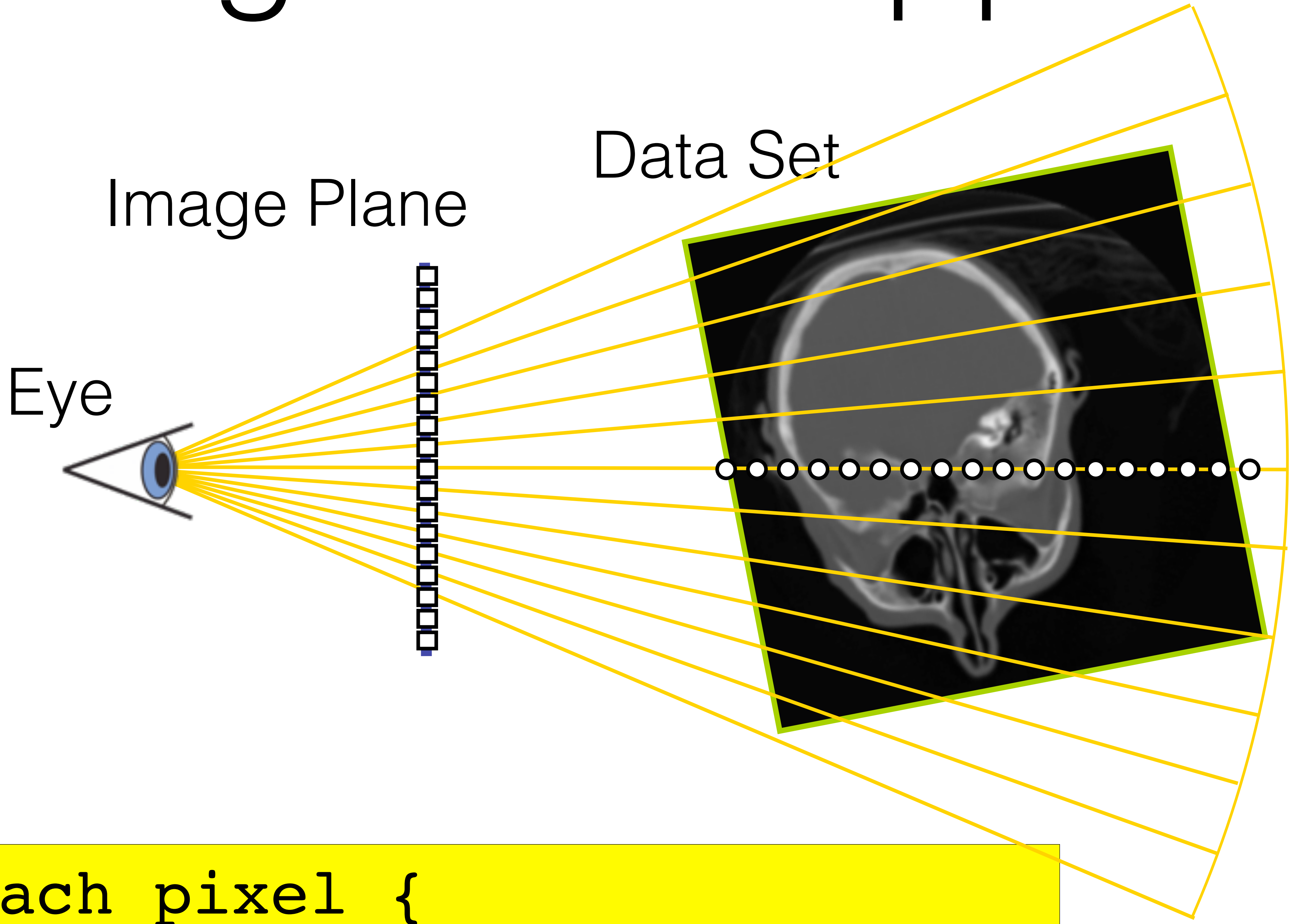


Image order approach



```
For each pixel {
   calculate color of the pixel
}
```

Image order approach

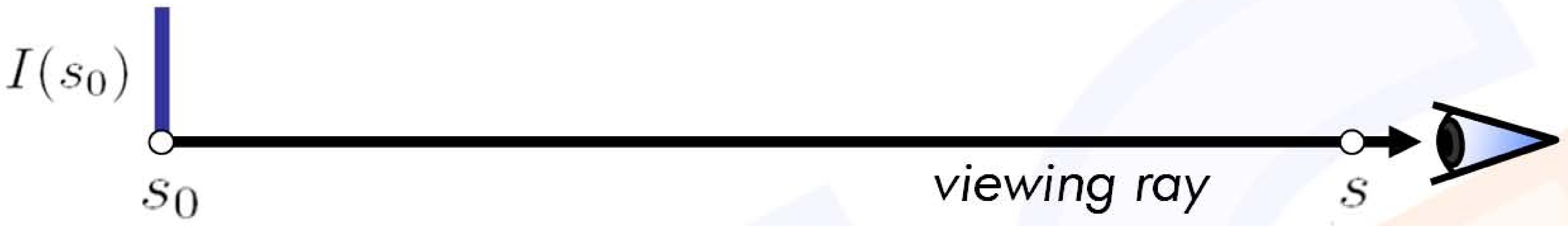


```
For each pixel {
   calculate color of the pixel
}
```

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



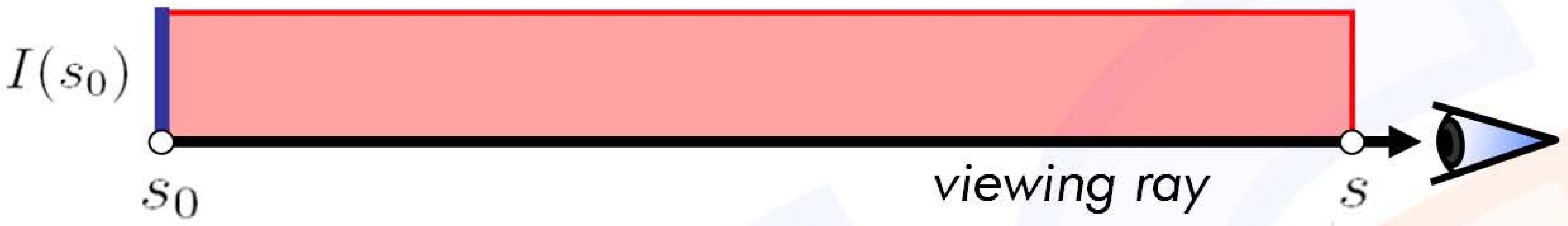
Initial intensity at s_0

$$I(s) = \boxed{I(s_0)}$$

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



Initial intensity at s_0

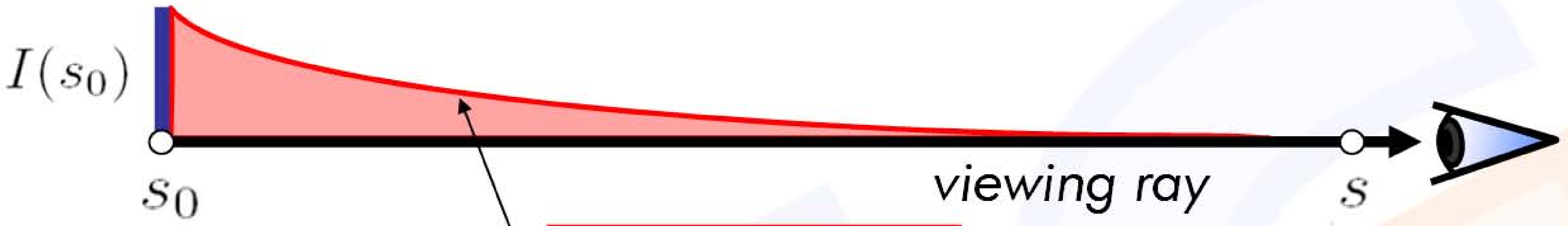
$$I(s) = \boxed{I(s_0)}$$

Without absorption all the initial radiant energy would reach the point *s*.

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



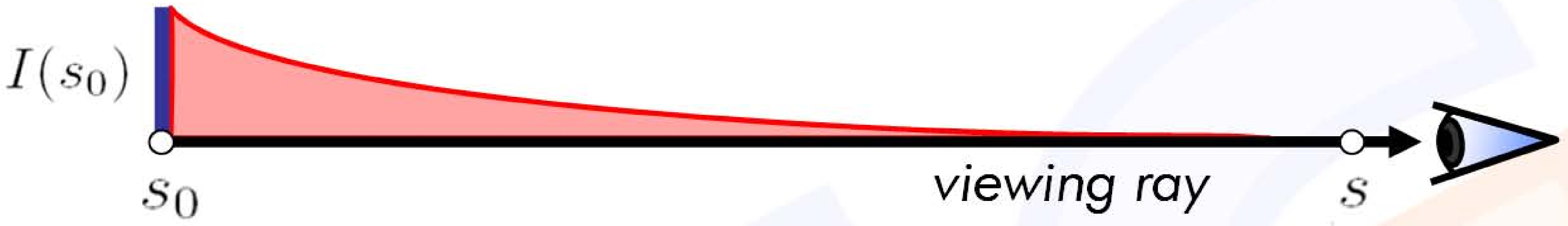
Absorption along the ray segment s_0 - s

$$I(s) = I(s_0) \boxed{e^{-\tau(s_0,s)}}$$

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



$$I(s) = I(s_0)\, e^{-\tau(s_0,s)}$$

Extinction τ

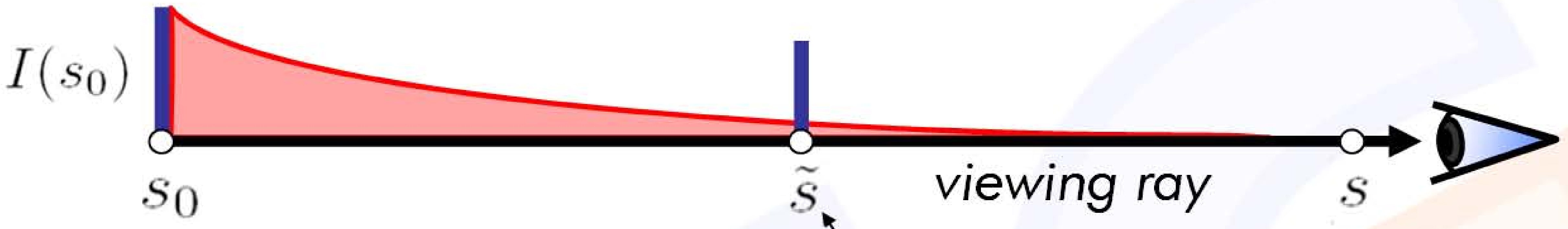
Absorption κ

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(s)\, ds.$$

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



One point $\tilde{s}$ along the viewing ray emits additional radiant energy.

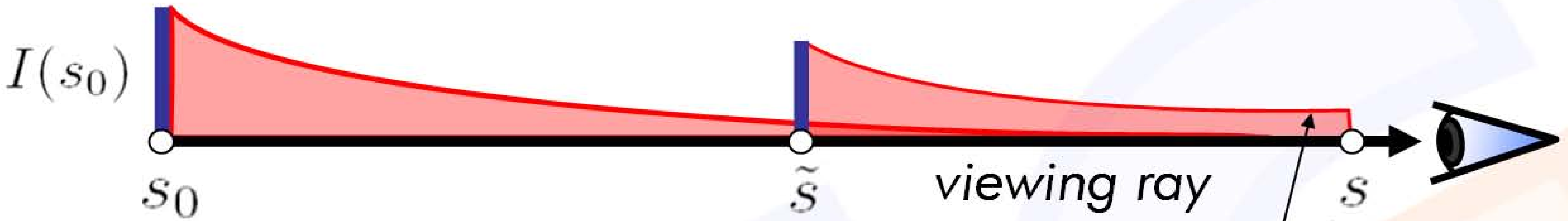
Active emission at point $\tilde{s}$

$$I(s) = I(s_0)\, e^{-\tau(s_0,s)} + \boxed{q(\tilde{s})}$$

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



One point $\tilde{s}$ along the viewing ray emits additional radiant energy.

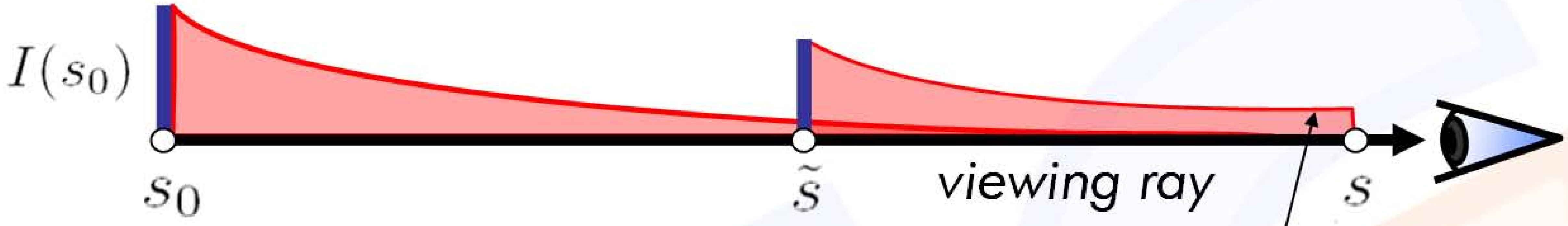
Absorption along the distance s - $\tilde{s}$

$$I(s) = I(s_0)\, e^{-\tau(s_0,s)} + q(\tilde{s})\, \boxed{e^{-\tau(\tilde{s},s)}}$$

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering

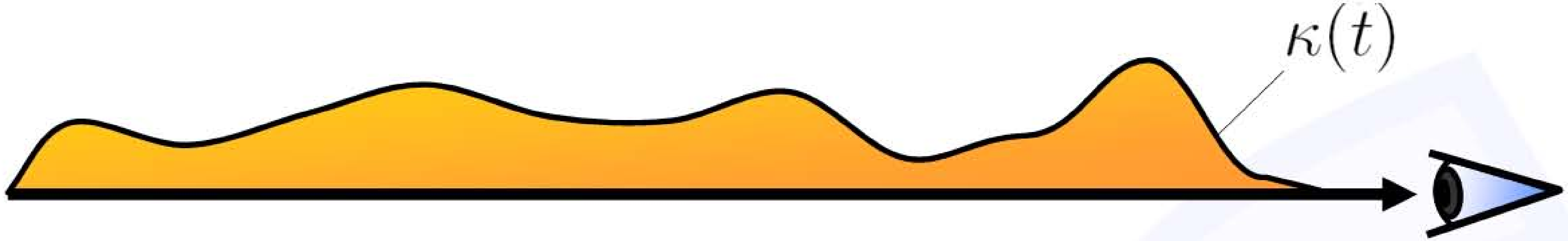


One point $\tilde{s}$ along the viewing ray emits additional radiant energy.

Absorption along the distance s - $\tilde{s}$

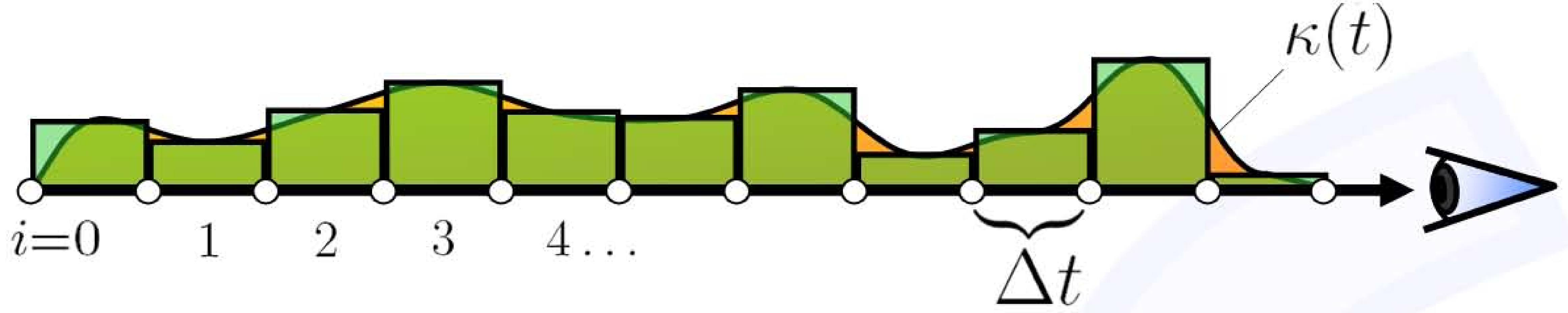
$$I(s) = I(s_0)\, e^{-\tau(s_0,s)} + q(\tilde{s})\, \boxed{e^{-\tau(\tilde{s},s)}}$$

Numerical Solution



Extinction: $\tau(0,t) \quad = \quad \int_0^t \kappa(\hat{t})\, d\hat{t}$

Numerical Solution



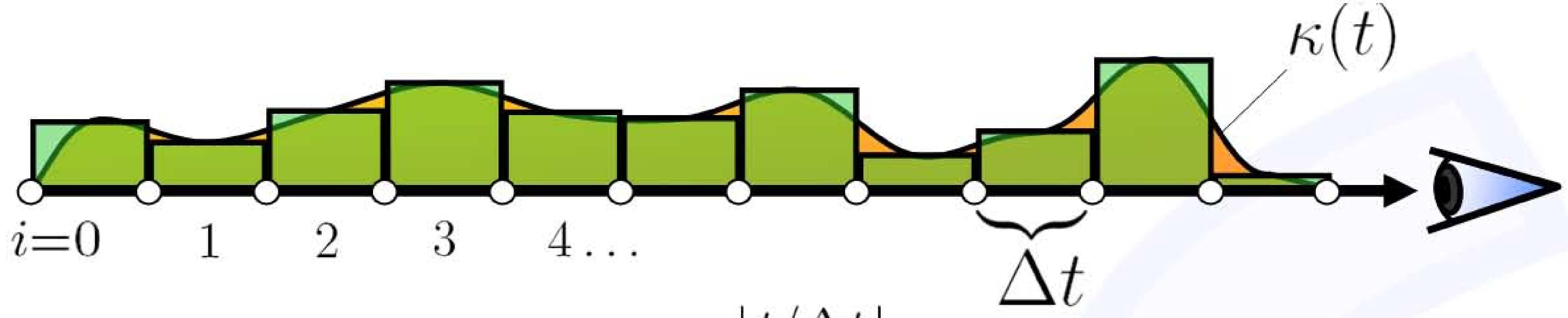
Extinction:

$$\tau(0,t) \quad = \quad \int_0^t \kappa(\hat{t})\, d\hat{t}$$

Approximate Integral by Riemann sum:

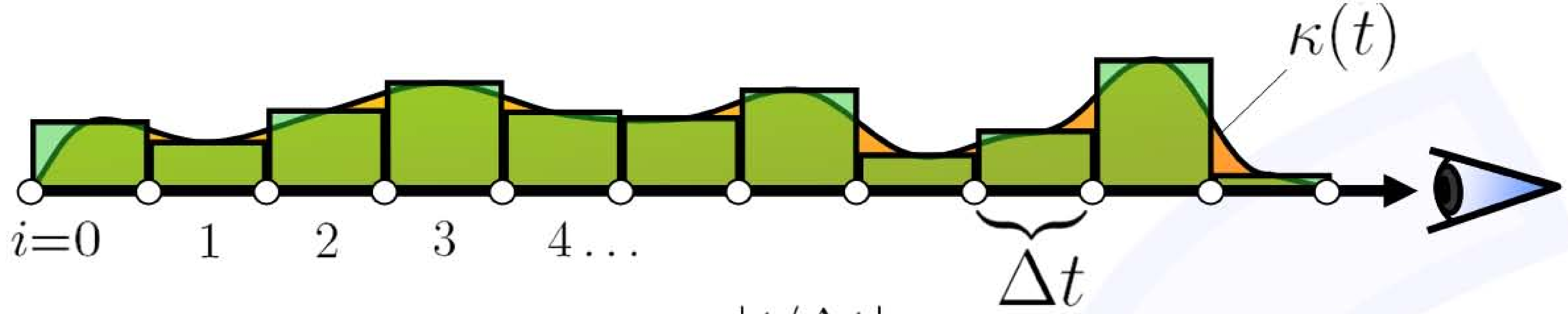
$$\tau(0,t) \quad \approx \quad \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

Numerical Solution



$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

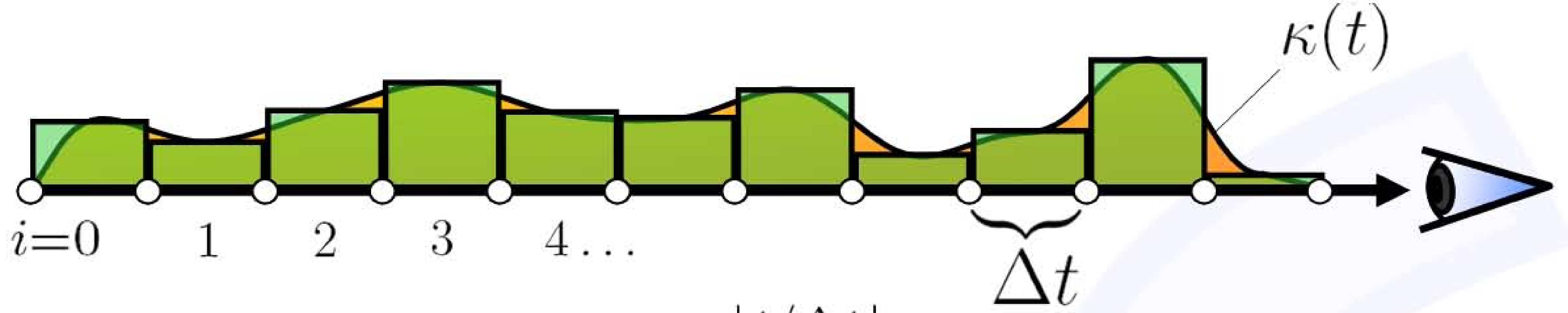
Numerical Solution



$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

$$e^{-\tilde{\tau}(0,t)} = e^{-\sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t}$$

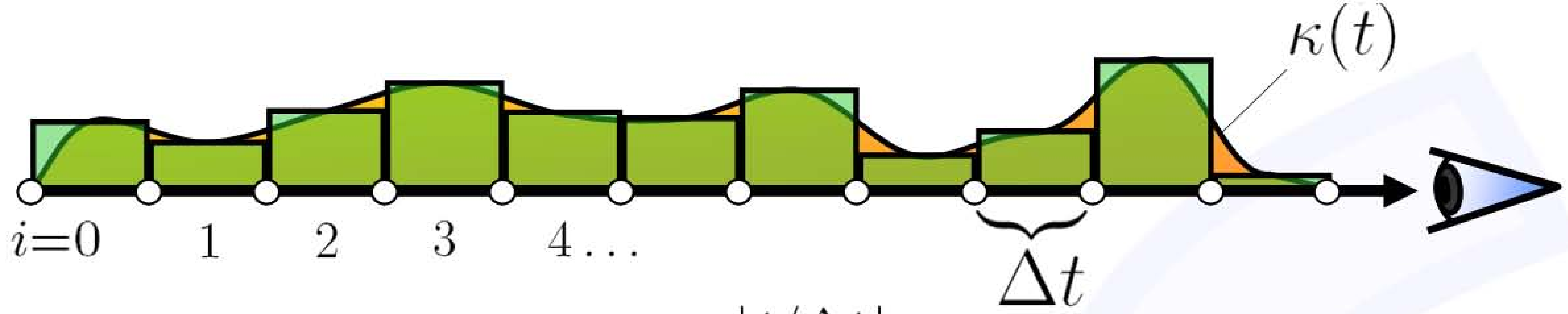
Numerical Solution



$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution



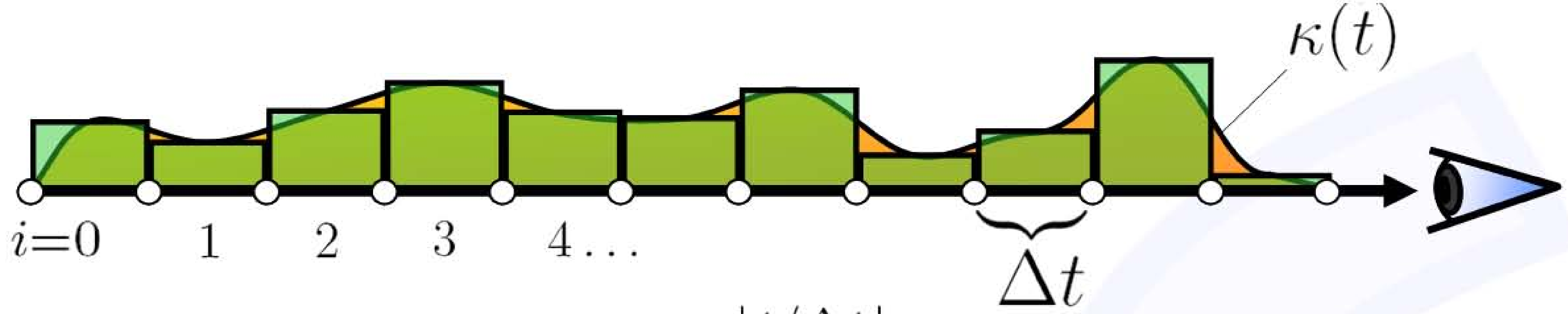
$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Now we introduce opacity:

$$A_i = 1 - e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution



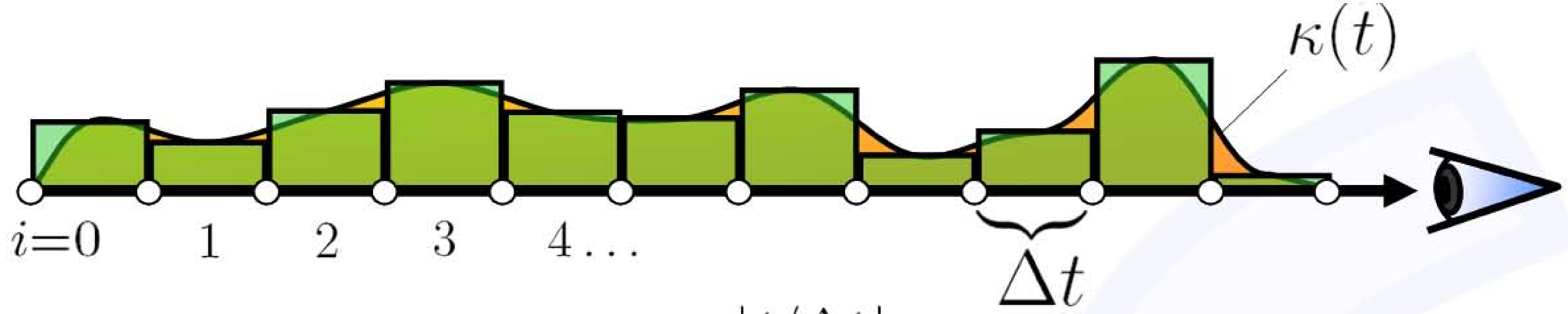
$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Now we introduce opacity:

$$1 - A_i = e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution



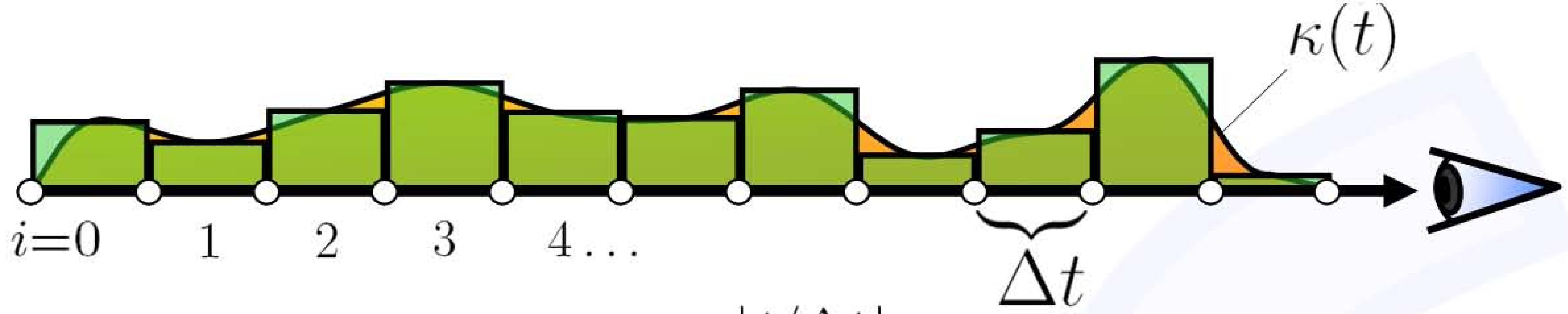
$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} \boxed{e^{-\kappa(i \cdot \Delta t)\, \Delta t}}$$

Now we introduce opacity:

$$1 - A_i = \boxed{e^{-\kappa(i \cdot \Delta t)\, \Delta t}}$$

Numerical Solution



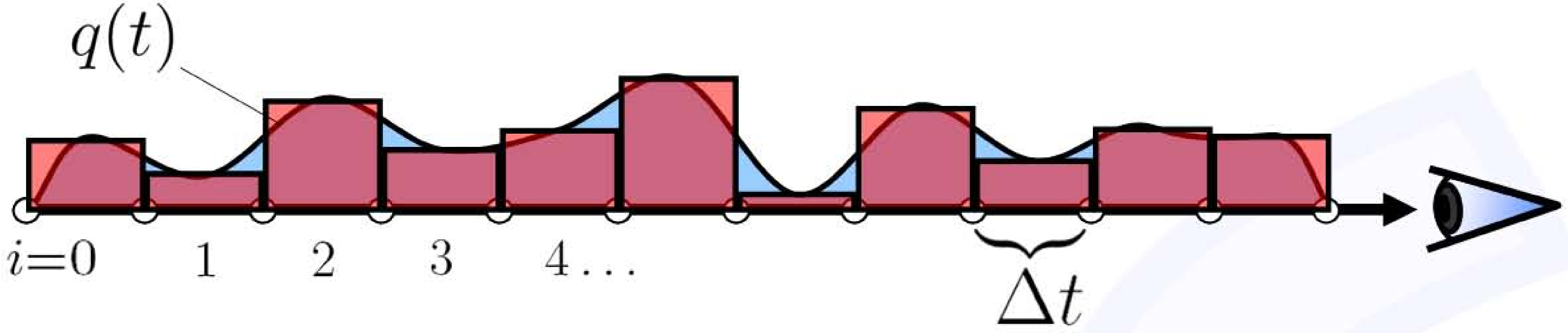
$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

Now we introduce opacity:

$$1 - A_i = e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

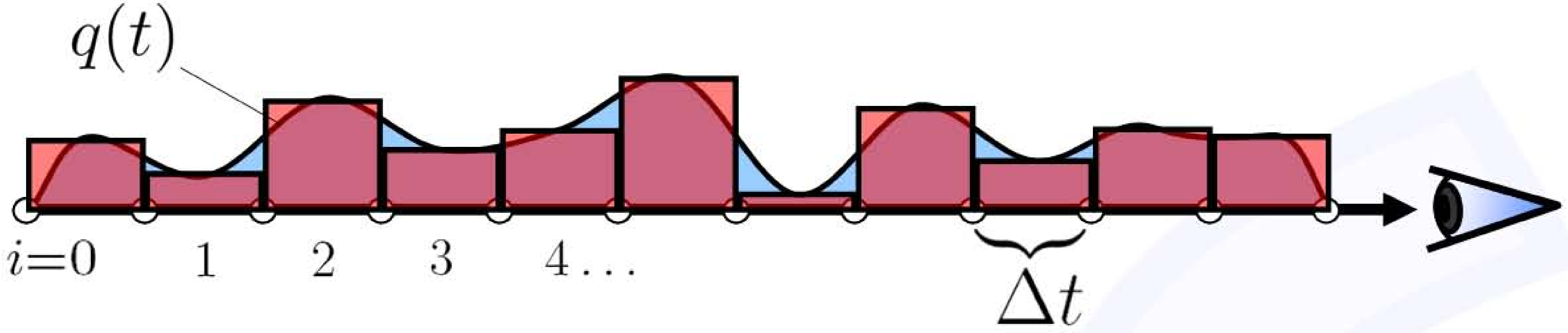
Numerical Solution



$$e^{-\tilde{\tau}(0,t)} \quad = \quad \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 \; - \; A_i)$$

$$q(t) \;\; \approx \;\; C_i \;\; = \;\; c(i \cdot \Delta t)\, \Delta t$$

Numerical Solution

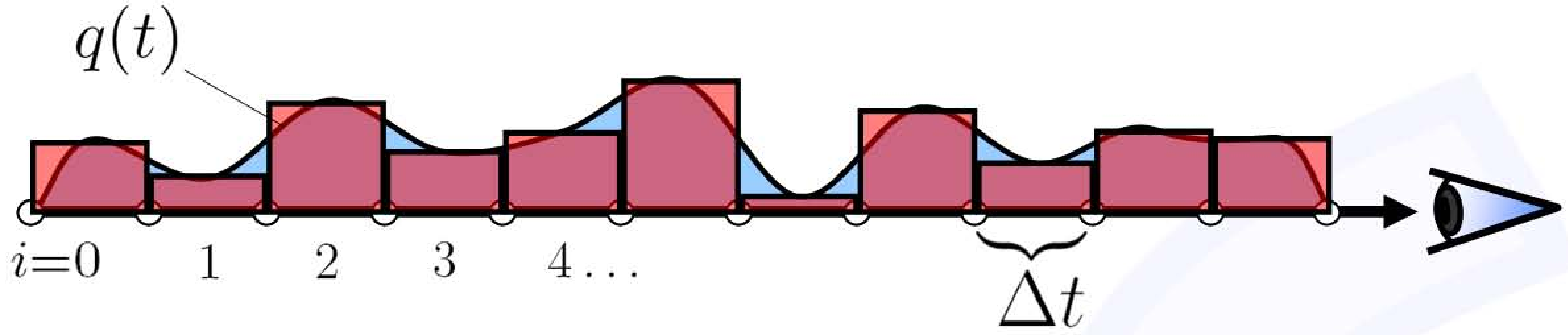


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

$$q(t) \approx C_i = c(i \cdot \Delta t)\,\Delta t$$

$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \, e^{-\tilde{\tau}(0,t)}$$

Numerical Solution

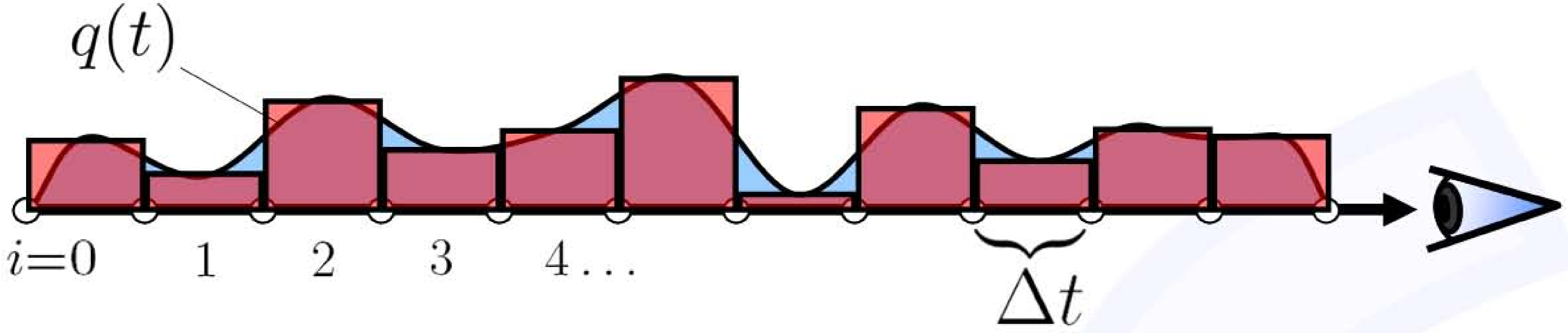


$$\boxed{e^{-\tilde{\tau}(0,t)}} \;=\; \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

$$q(t) \;\approx\; C_i \;=\; c(i \cdot \Delta t)\, \Delta t$$

$$\tilde{C} \;=\; \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \boxed{e^{-\tilde{\tau}(0,t)}}$$

Numerical Solution

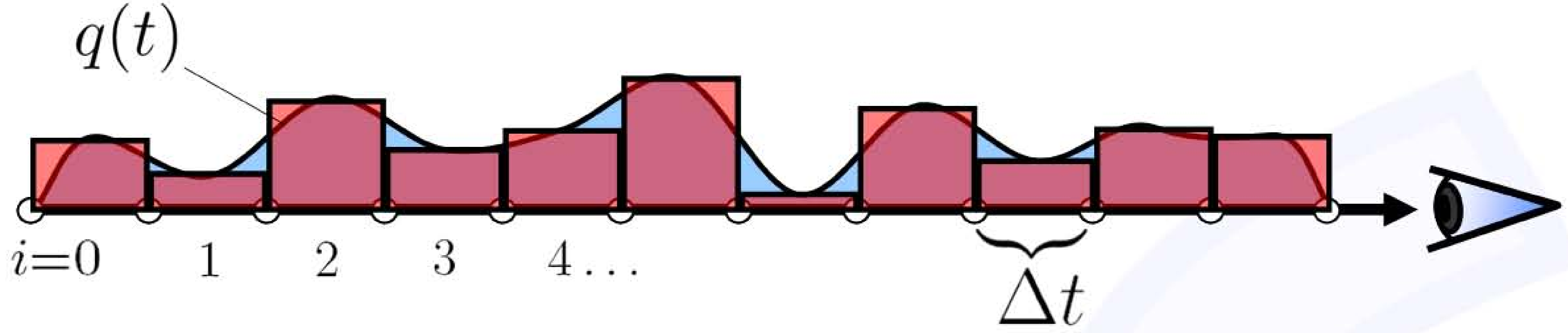


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

$$q(t) \approx C_i = c(i \cdot \Delta t)\,\Delta t$$

$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

Numerical Solution

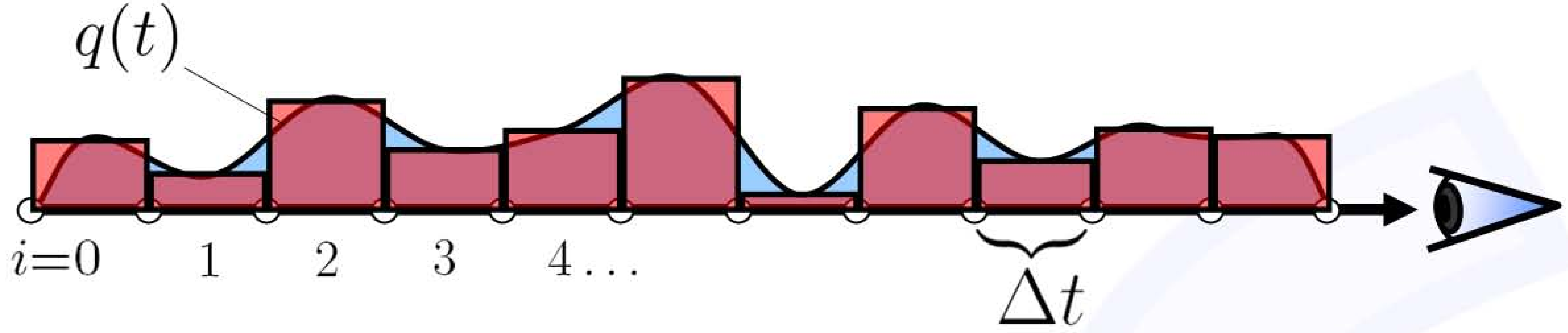


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

$$q(t) \approx C_i = c(i \cdot \Delta t)\, \Delta t$$

$$\boxed{\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)}$$

Numerical Solution



$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

can be computed recursively

$$C'_i = C_i + (1 - A_i) C'_{i-1}$$

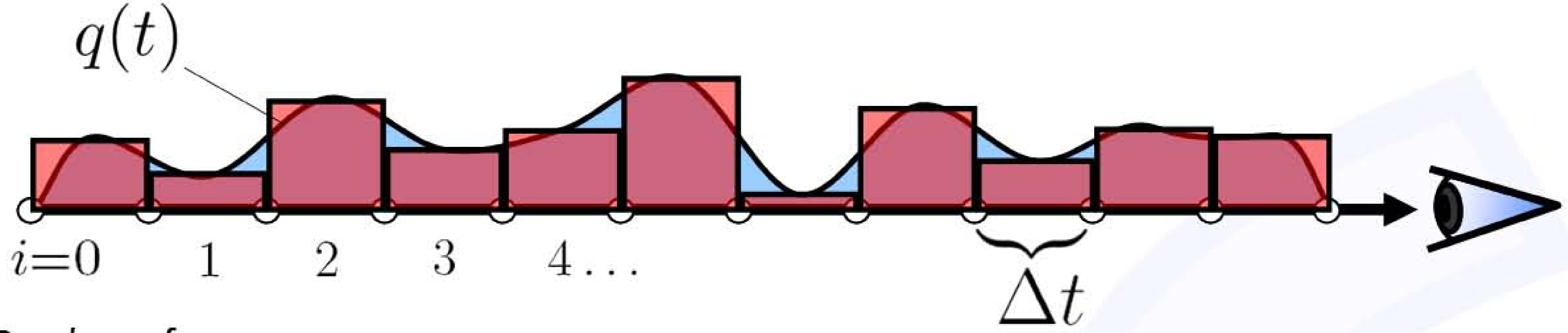
Radiant energy observed at position ***i***

Radiant energy emitted at position ***i***

Absorption at position ***i***

Radiant energy observed at position ***i–1***

Numerical Solution



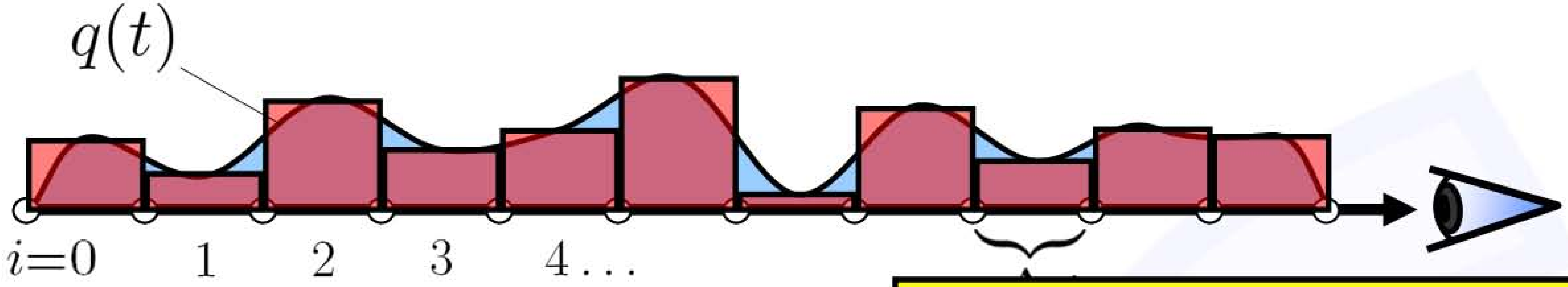
Back-to-front compositing

$$C'_i = C_i + (1 - A_i) C'_{i-1}$$

Front-to-back compositing

$$C'_i = C'_{i+1} + (1 - A'_{i+1}) C_i$$
$$A'_i = A'_{i+1} + (1 - A'_{i+1}) A_i$$

Numerical Solution



Back-to-front compositing

$$C'_i \;=\; C_i \;+\; (1$$

Early Ray Termination:

Stop the calculation when

$$A'_i \approx 1$$

Front-to-back compositing

$$\begin{aligned} C'_i &= C'_{i+1} + \boxed{(1 - A'_{i+1})} C_i \\ A'_i &= A'_{i+1} + (1 - A'_{i+1}) A_i \end{aligned}$$

Summary

- Emission Absorption Model

true emission
true absorption

$$I(s) \quad = \quad I(s_0)\, e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(\tilde{s})\, e^{-\tau(\tilde{s},s)}\, \mathrm{d}\tilde{s}$$

- Numerical Solutions

Back-to-front iteration

$$C'_i \quad = \quad C_i + (1 - A_i)C'_{i-1}$$

Front-to-back iteration

$$\begin{aligned} C'_i &= C'_{i+1} + (1 - A'_{i+1})C_i \\ A'_i &= A'_{i+1} + (1 - A'_{i+1})A_i \end{aligned}$$

Sample ray casting code

```
//the step size, i.e. “delta t” from the Engel slides.
float dt = 1.0 / normalize(volumeGridDimensions);

vec3 dtDirection = normalize(direction) * dt;
vec3 p = frontPos;

vec4 accumulatedColor = vec4(0.0);
float accumulatedAlpha = 0.0;
float t = 0.0;

for(int i = 0; i < 4096; i++)
{
    float value = sampleAs3DTexture(p);
    vec4 colorSample = classify(value);

    //(optional) lighting
    colorSample.rgb = shade(colorSample.rgb, p, value, direction);

    //front-to-back compositing
    blend(colorSample.rgb, colorSample.a);

    p += dtDirection;
    t += dt;

   //exit or early termination
   if(t >= rayLength || accumulatedAlpha >= .97 )
       break;
}
```

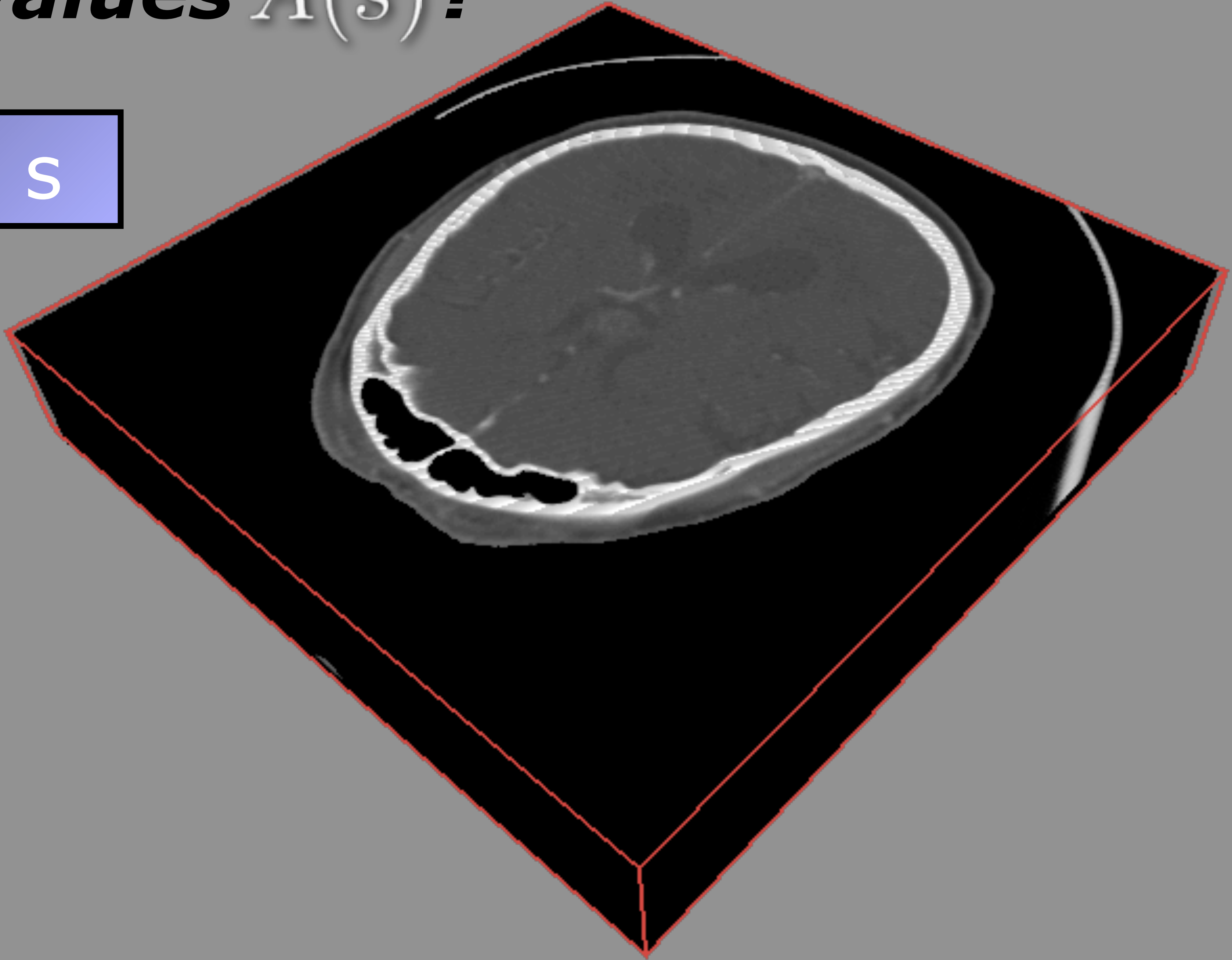
www.sci.utah.edu

Transfer functions

Classification

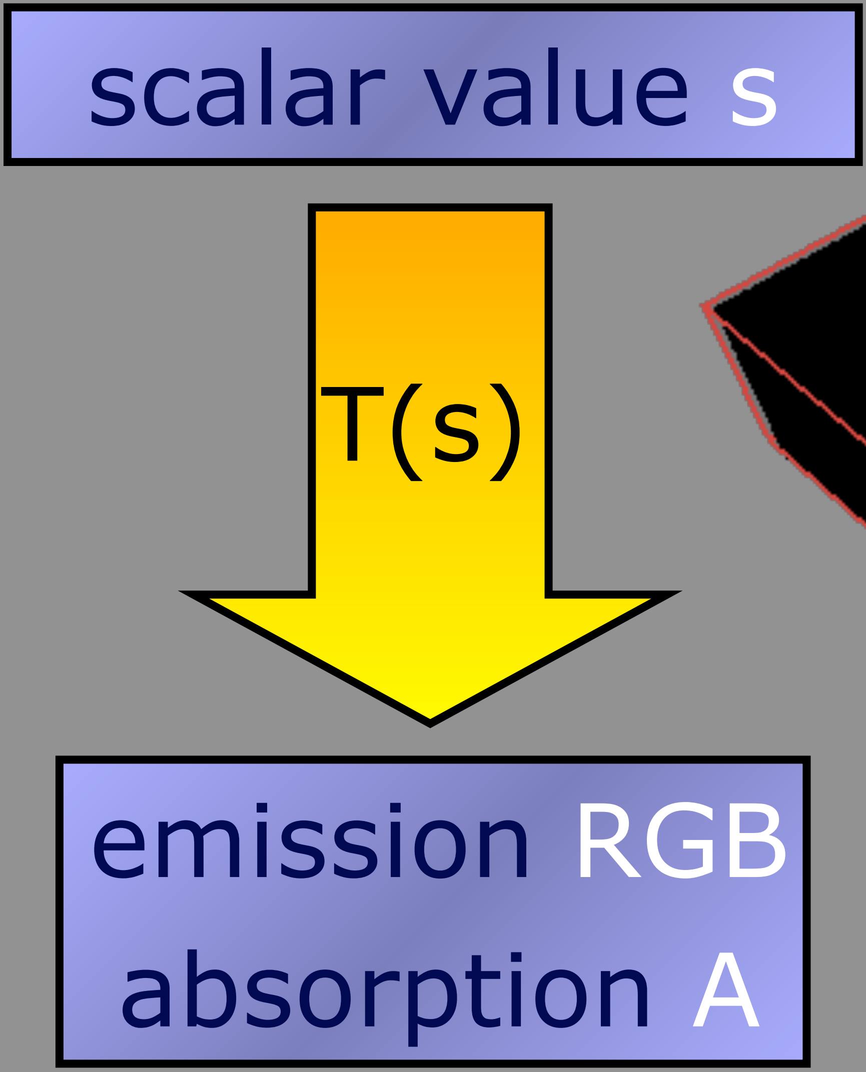
How do I obtain the emission values $\overset{(C_i)}{q(s)}$ ***and Absorption values*** $A(s)$***?***

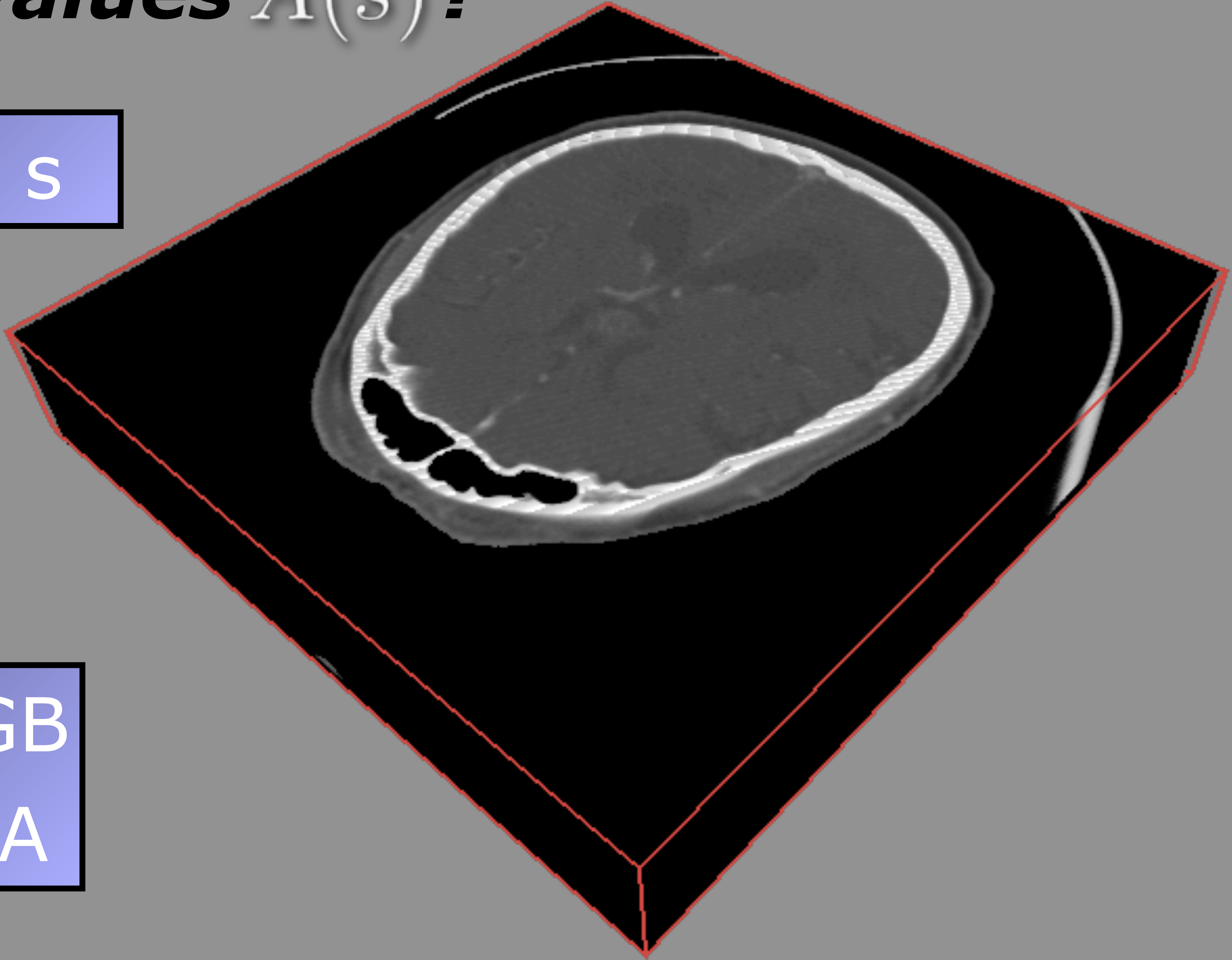
scalar value s



Classification

How do I obtain the emission values $\overset{(C_i)}{q(s)}$ ***and Absorption values*** $A(s)$ ***?***

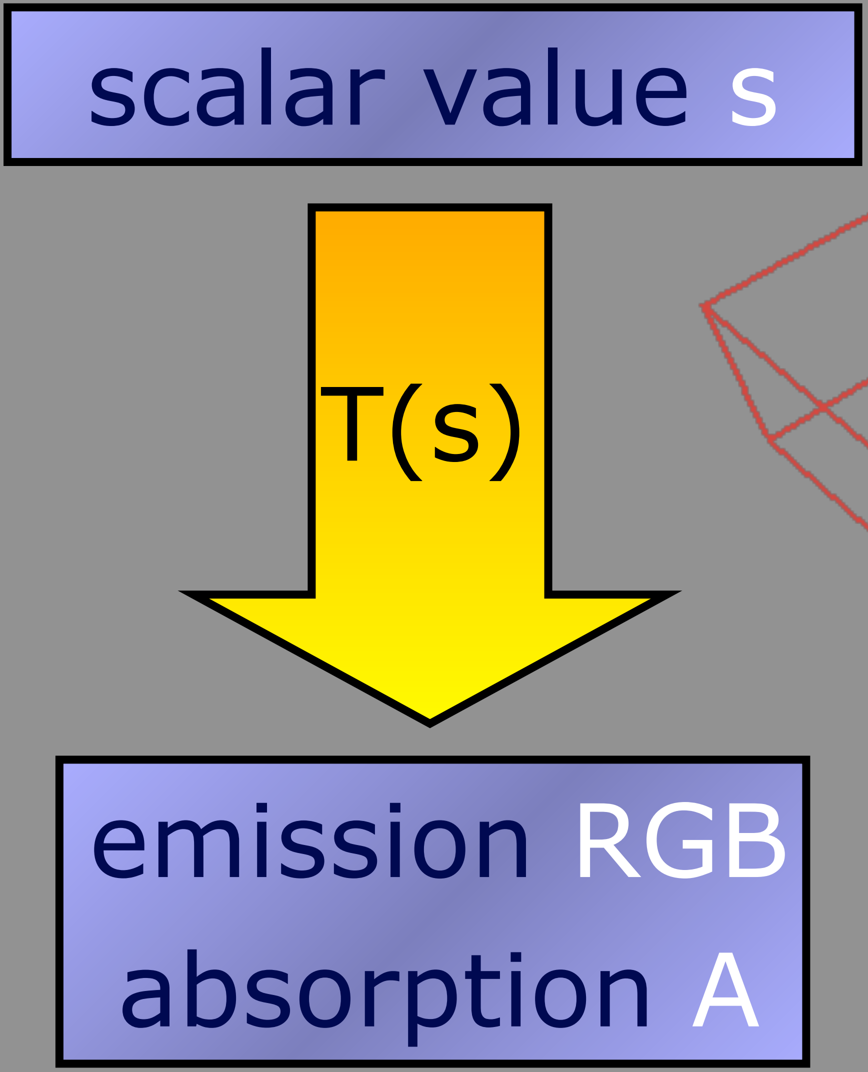


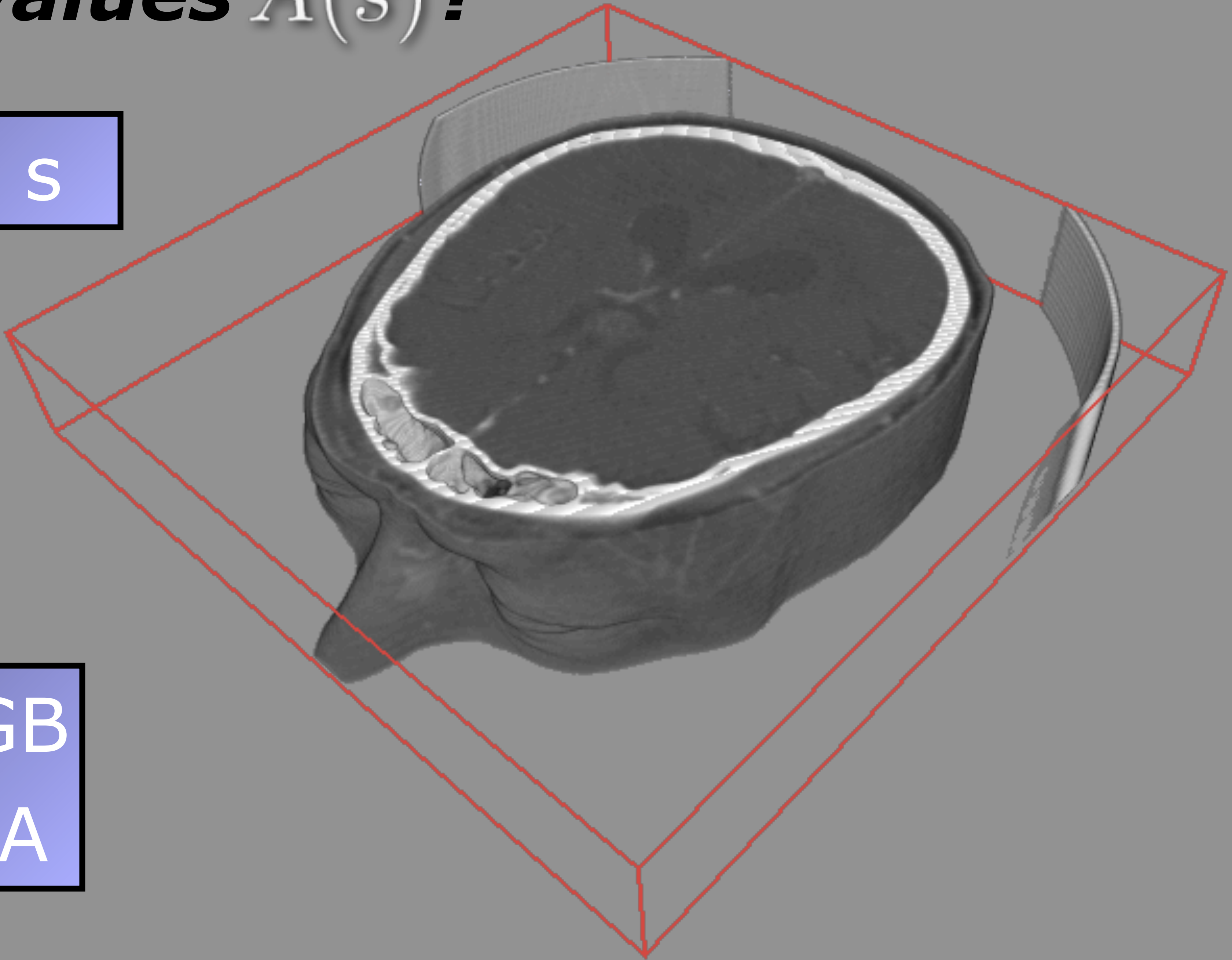


Joshua Levine, Clemson University

Classification

How do I obtain the emission values $\overset{(C_i)}{q(s)}$ ***and Absorption values*** $A(s)$***?***

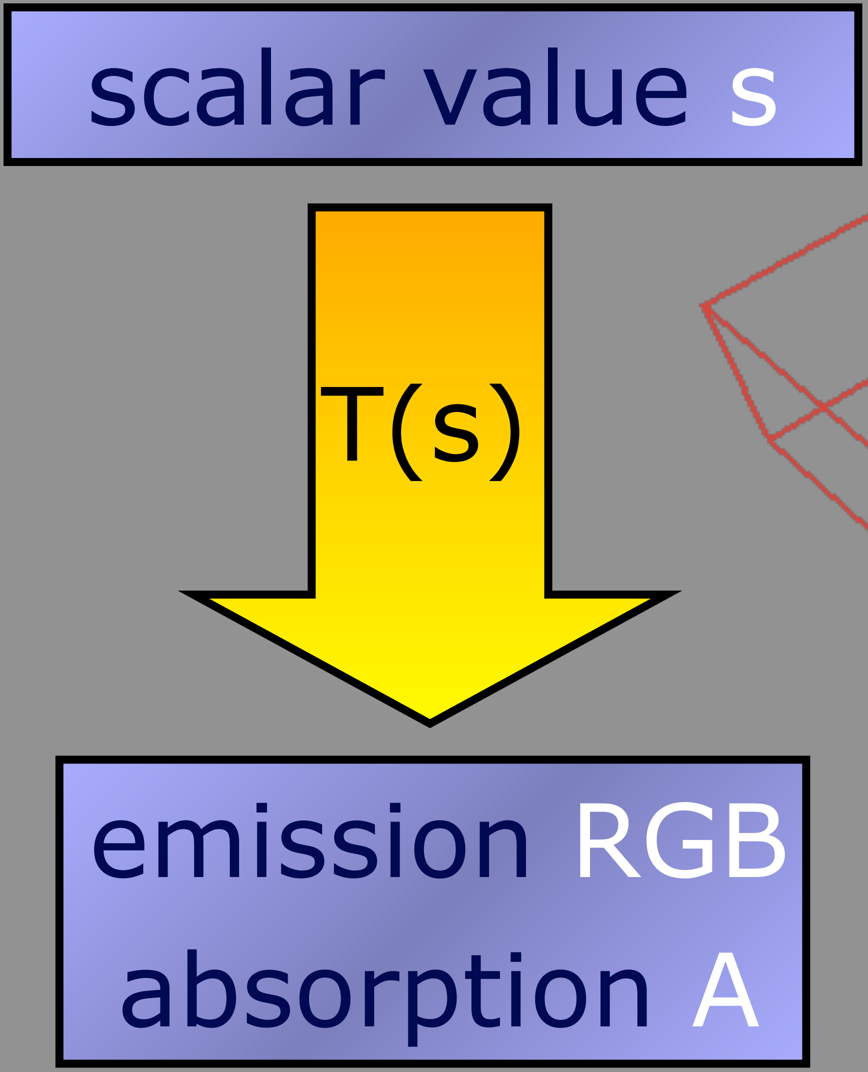


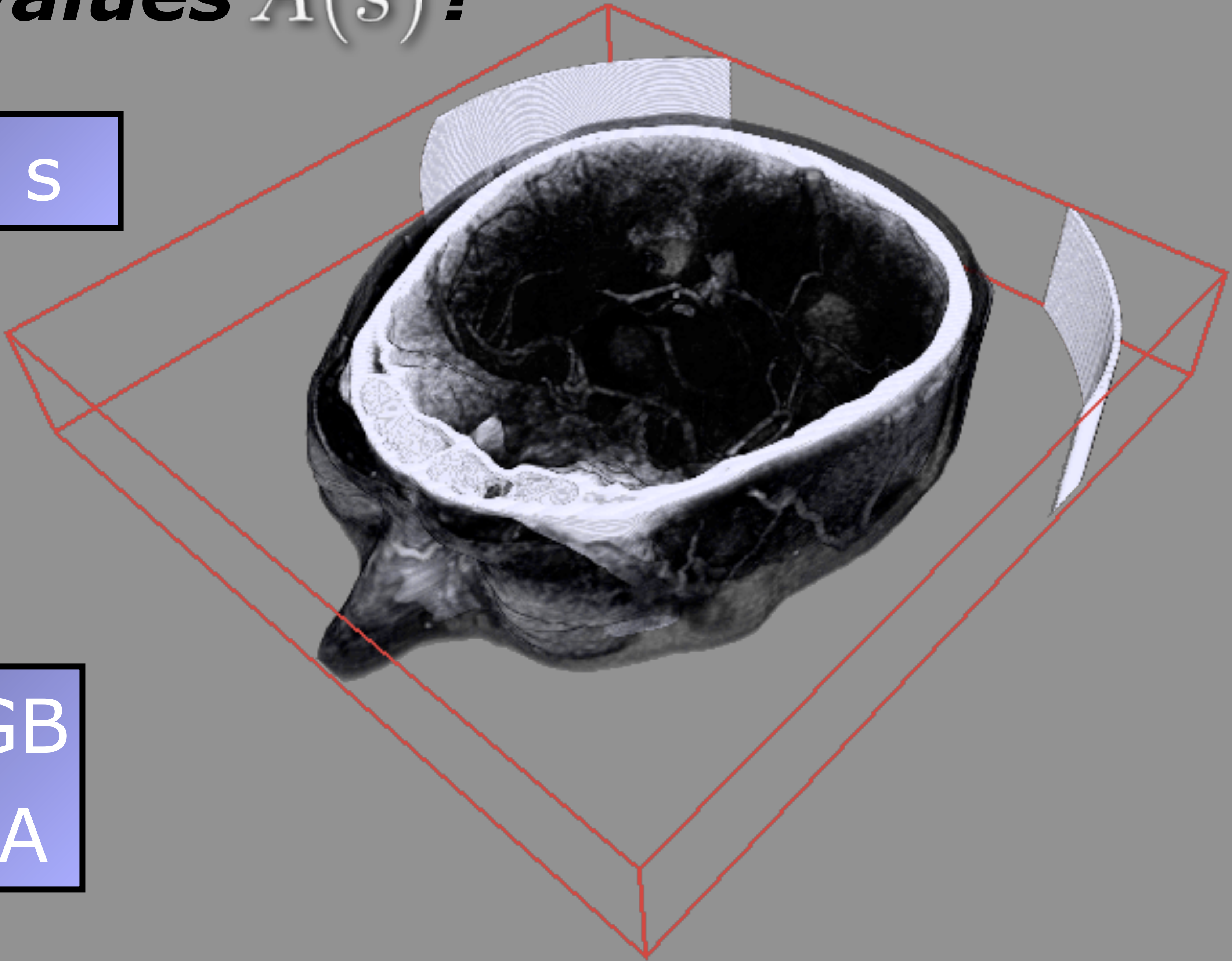


Joshua Levine, Clemson University

Classification

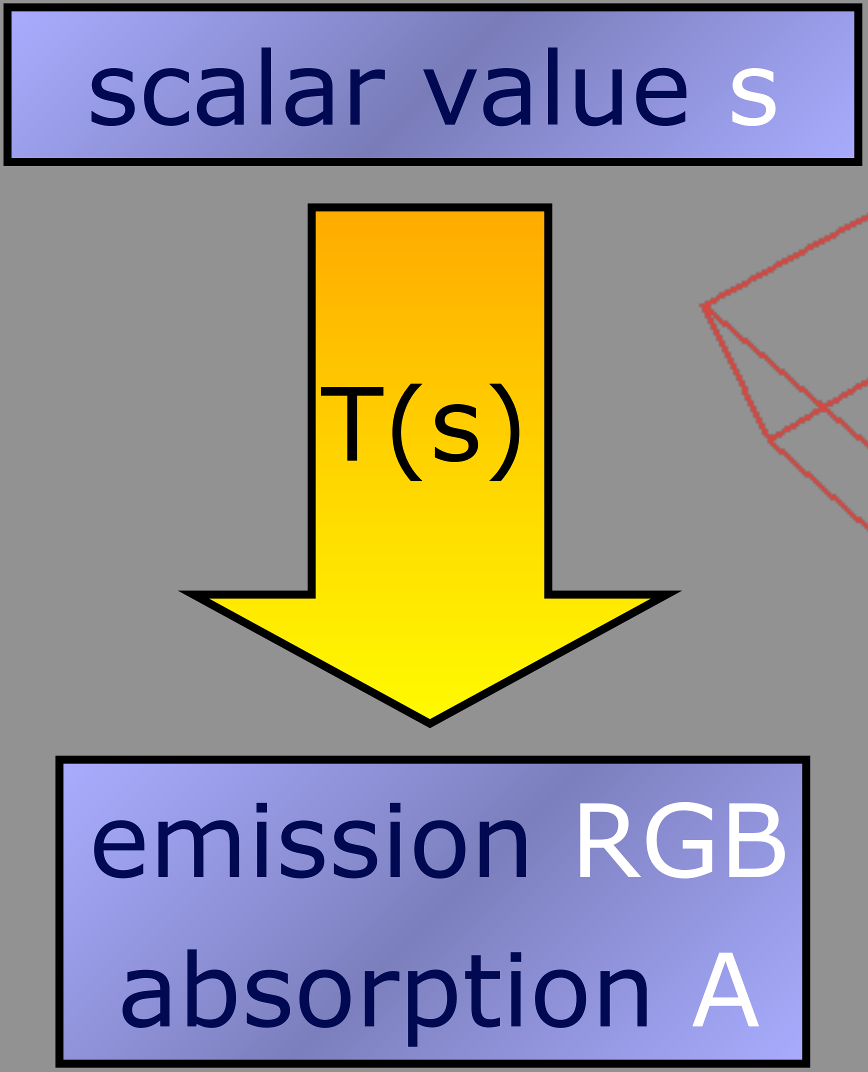
How do I obtain the emission values $\overset{(C_i)}{q(s)}$ ***and Absorption values*** $A(s)$***?***

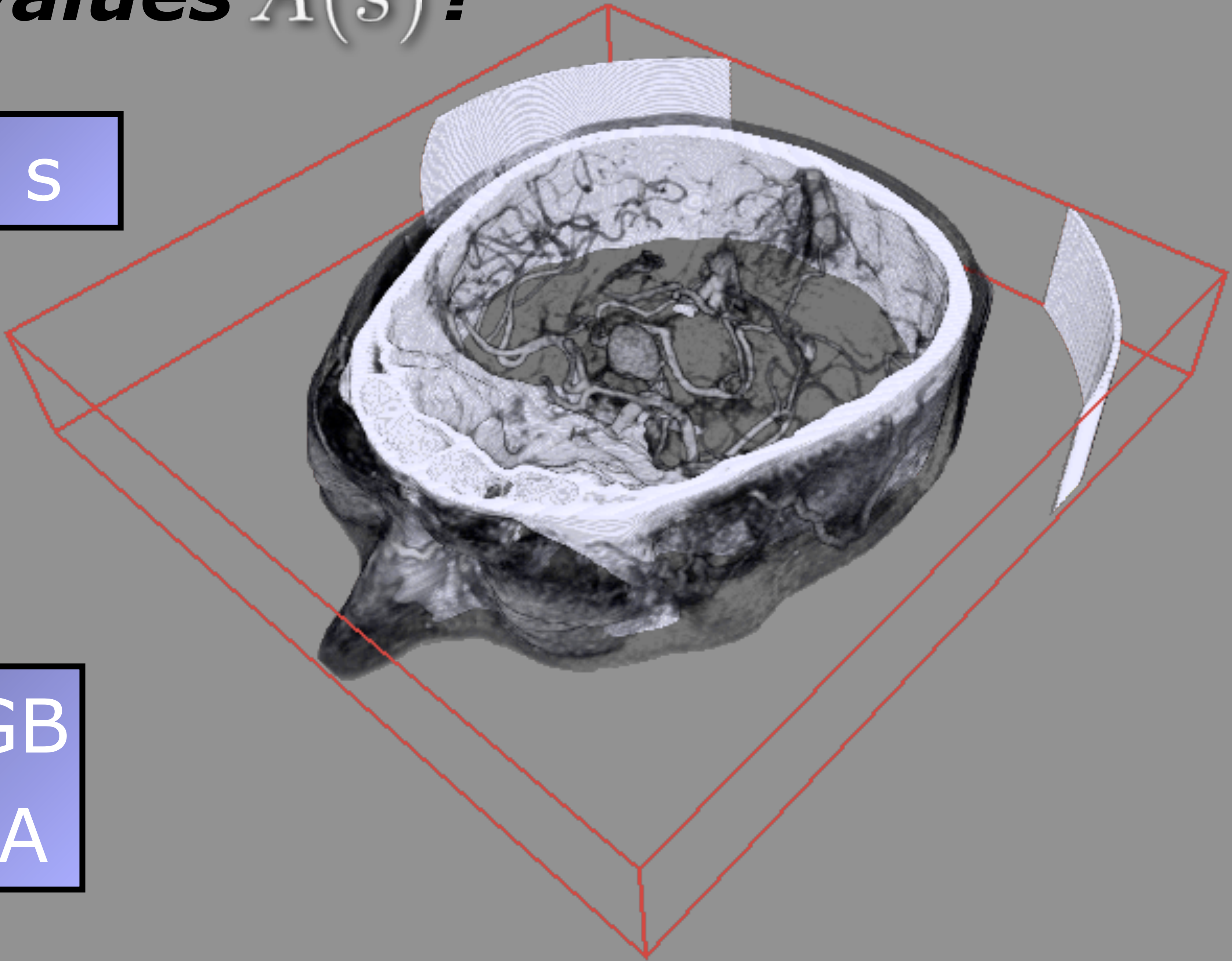




Classification

How do I obtain the emission values (C$_i$) $q(s)$ ***and Absorption values*** $A(s)$ ***?***

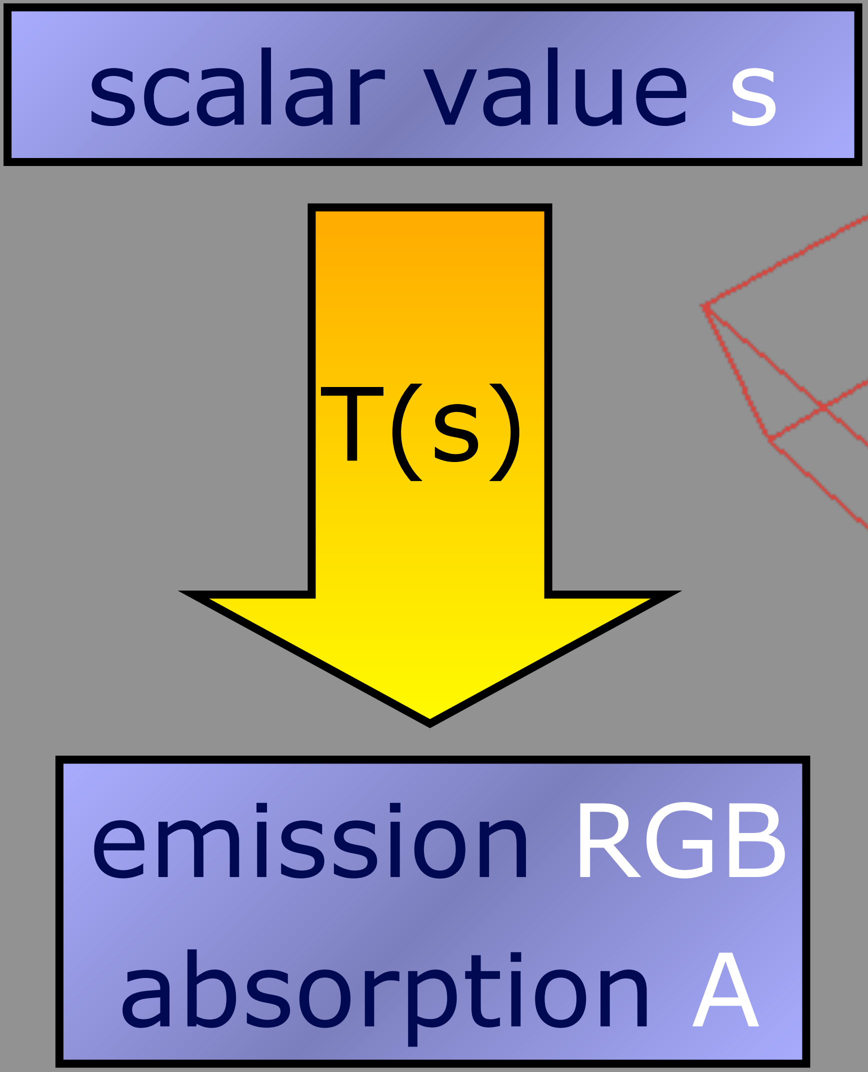


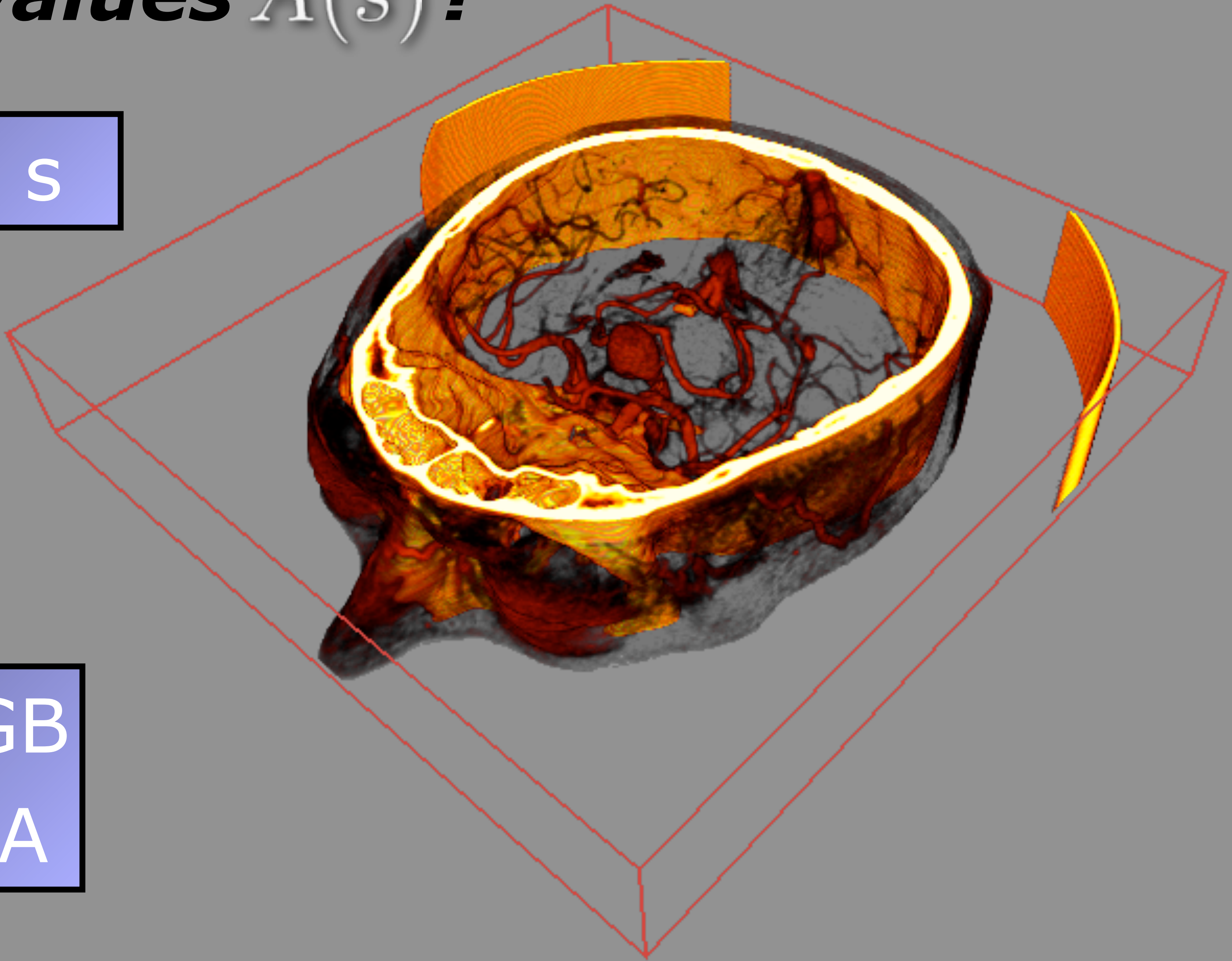


Joshua Levine, Clemson University

Classification

How do I obtain the emission values $\overset{(C_i)}{q(s)}$ ***and Absorption values*** $A(s)$***?***



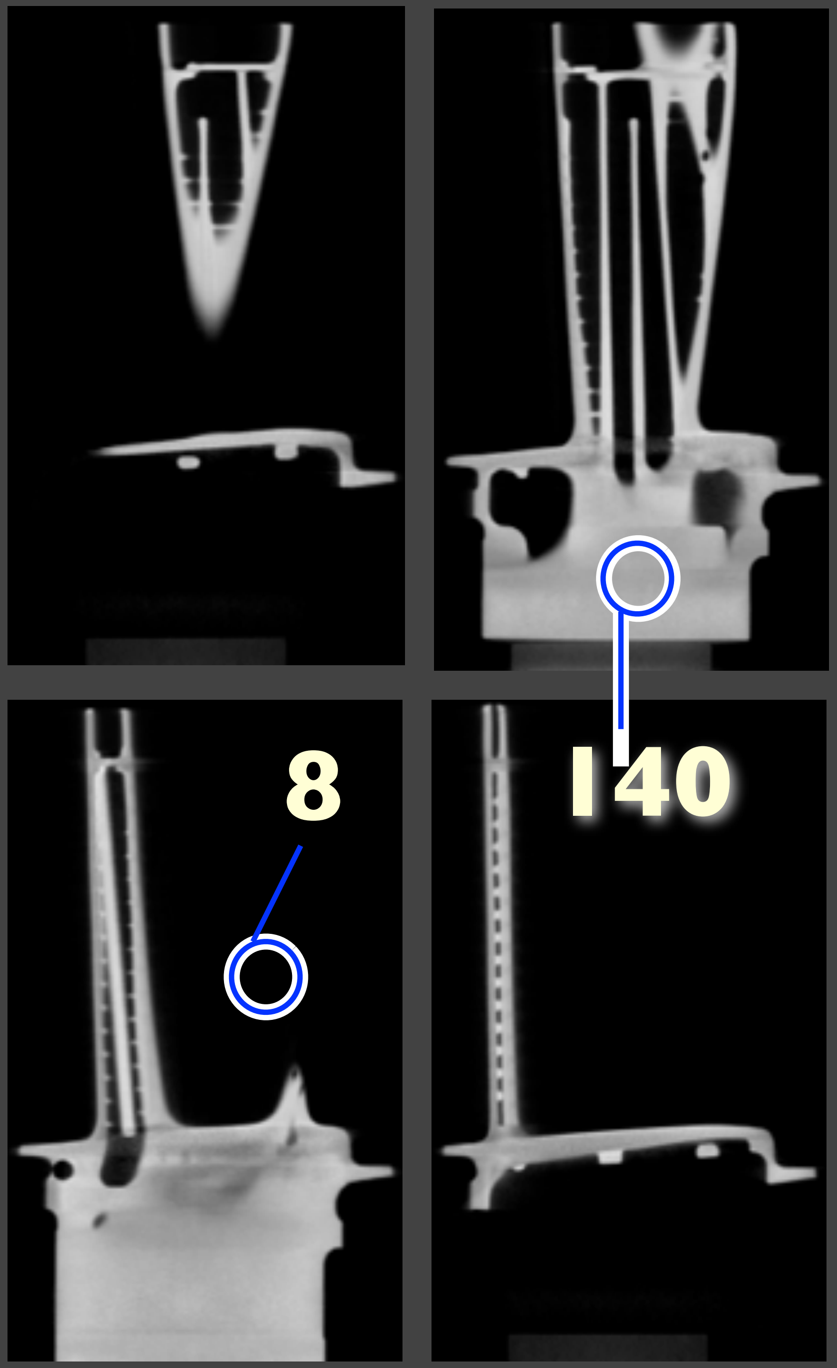


Joshua Levine, Clemson University

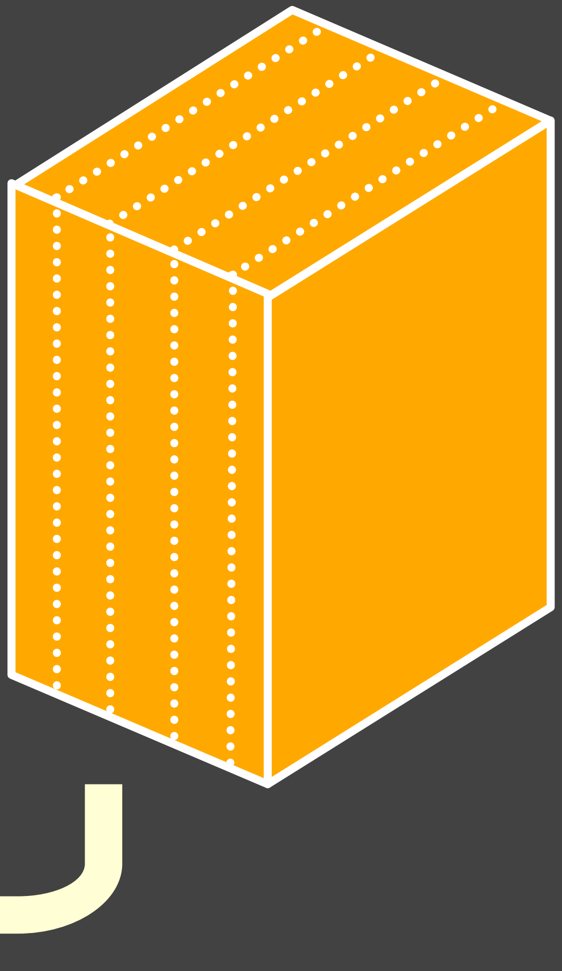
Introduction

Transfer functions make volume data visible by mapping data values to optical properties

slices:



volume data:

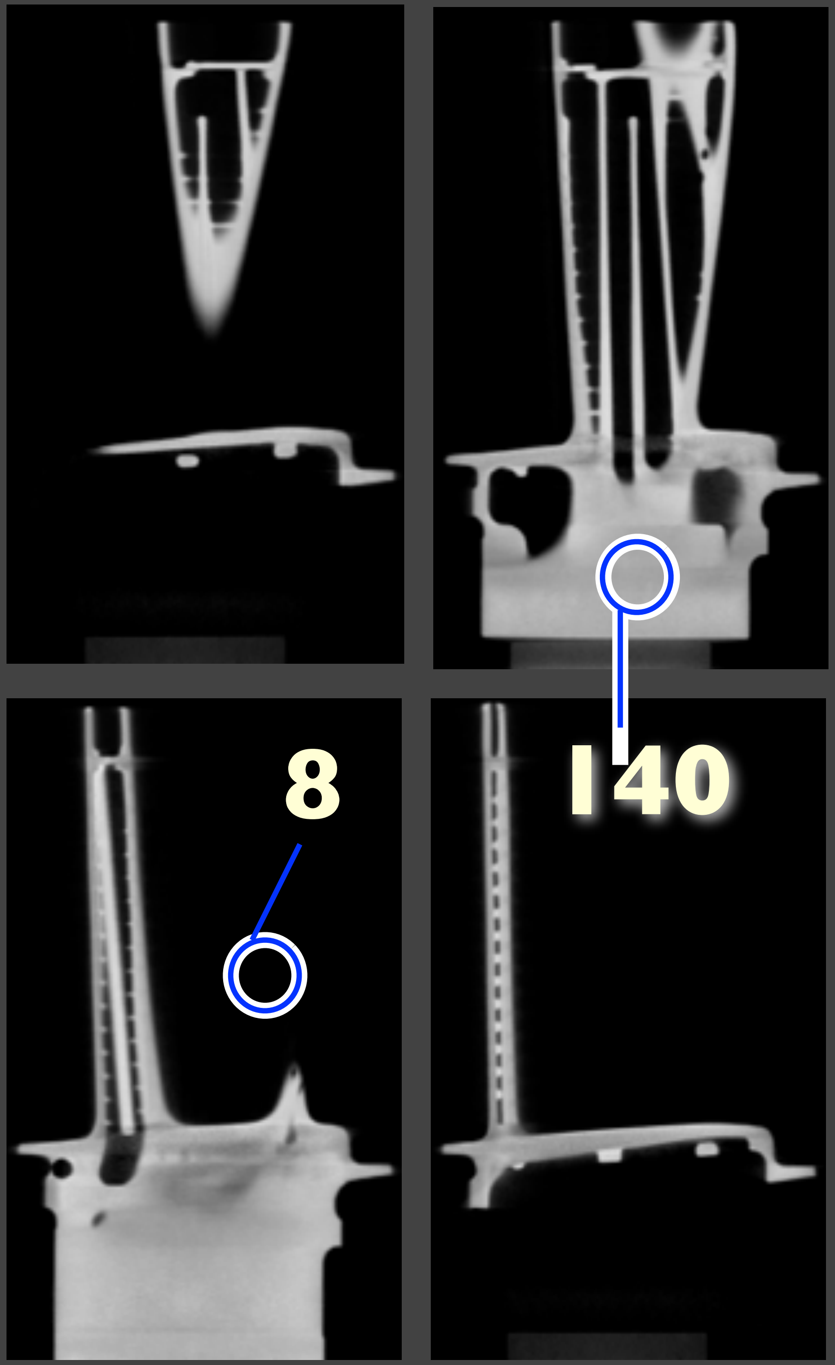


Joshua Levine, Clemson University

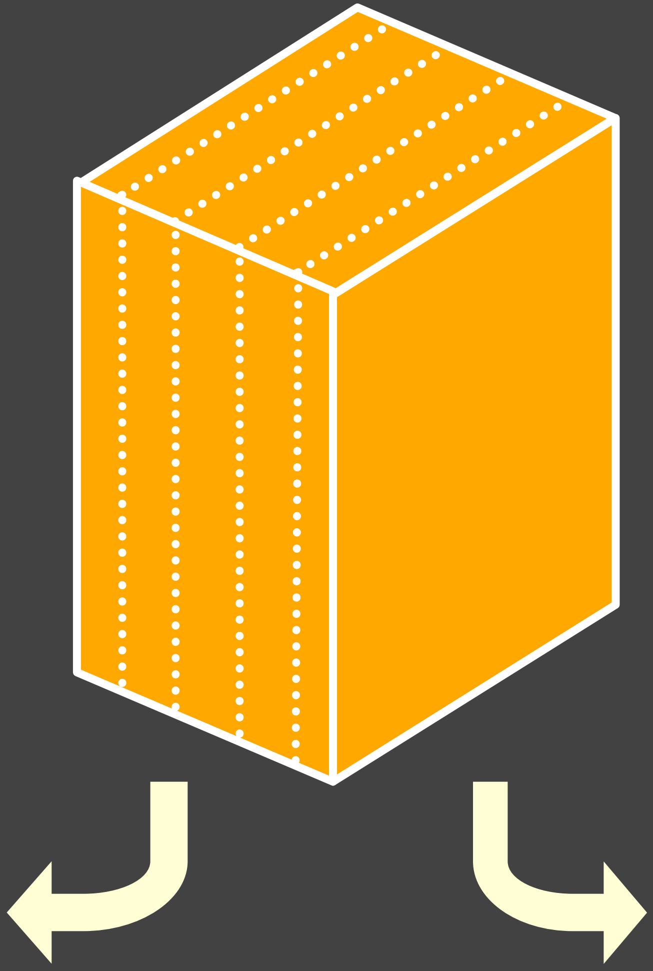
Introduction

Transfer functions make volume data visible by mapping data values to optical properties

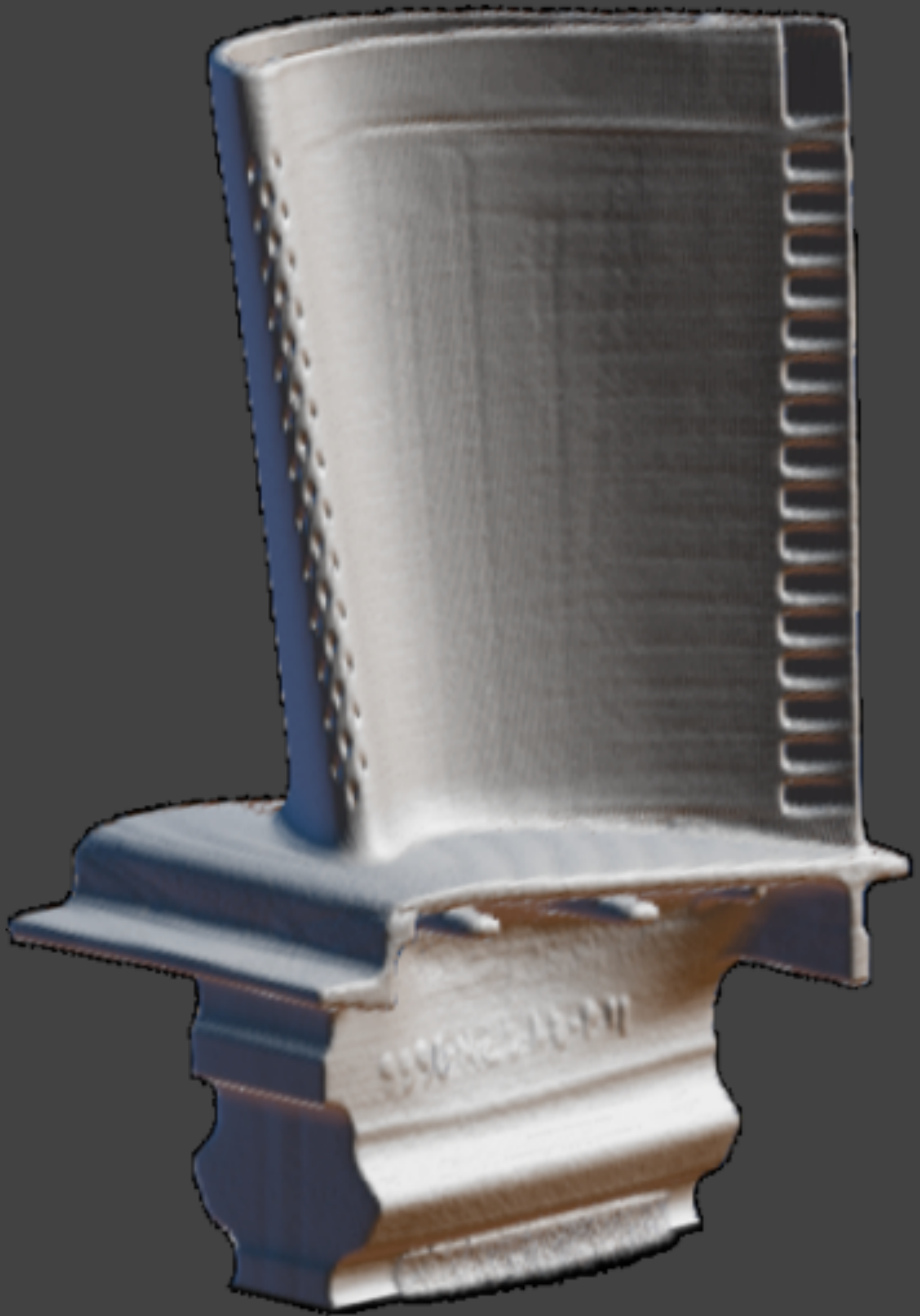
slices:



volume data:

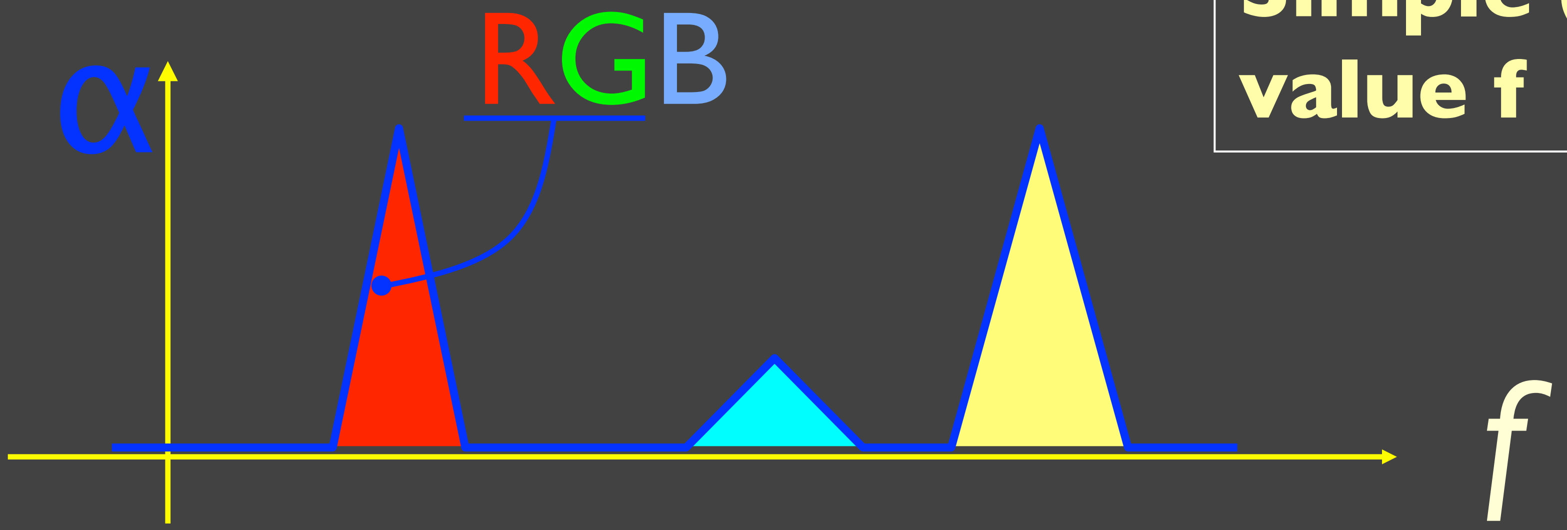


volume rendering:



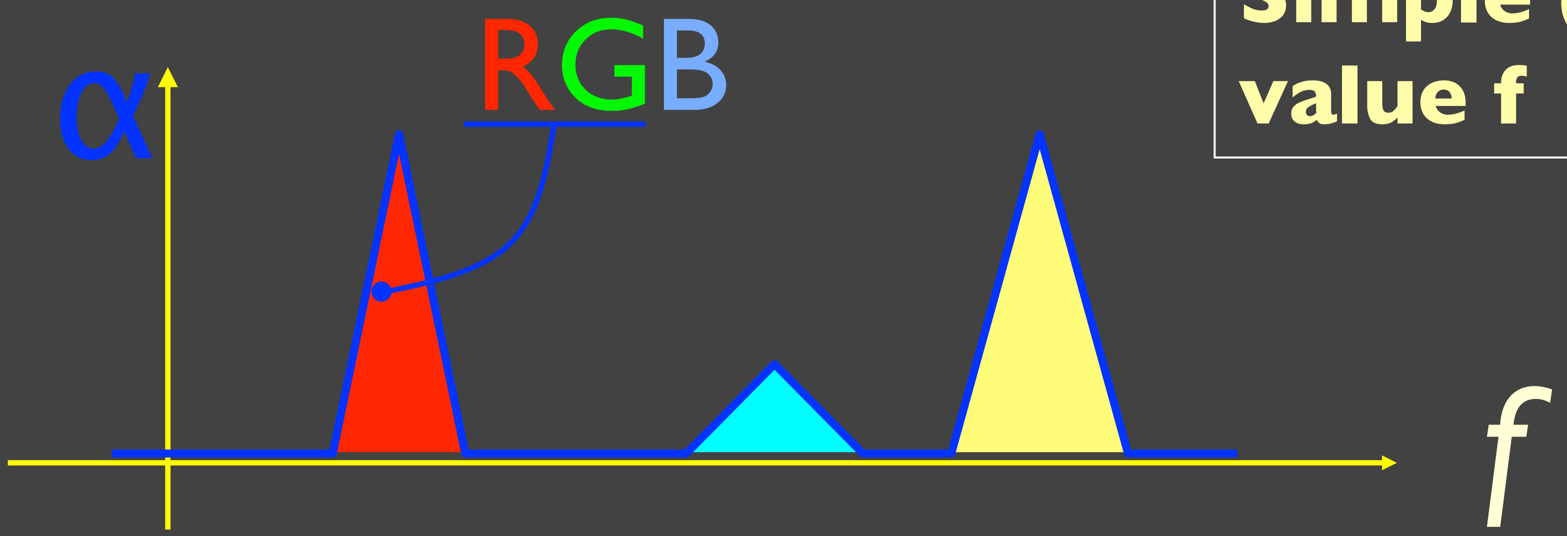
Transfer Functions (TFs)

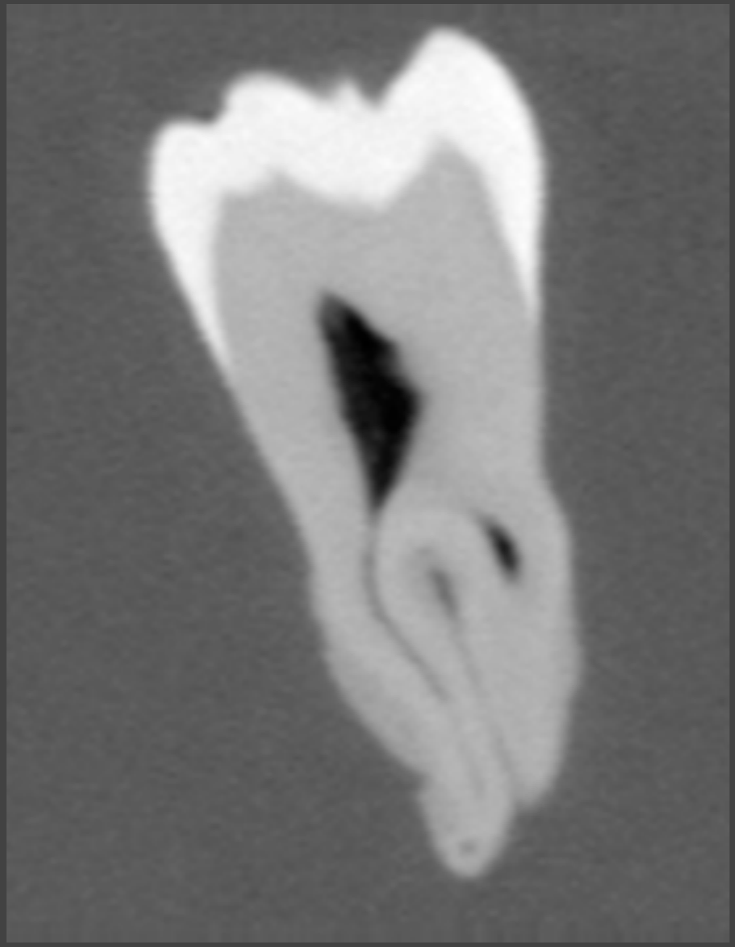
Simple (usual) case: Map data value f to color and opacity



Transfer Functions (TFs)

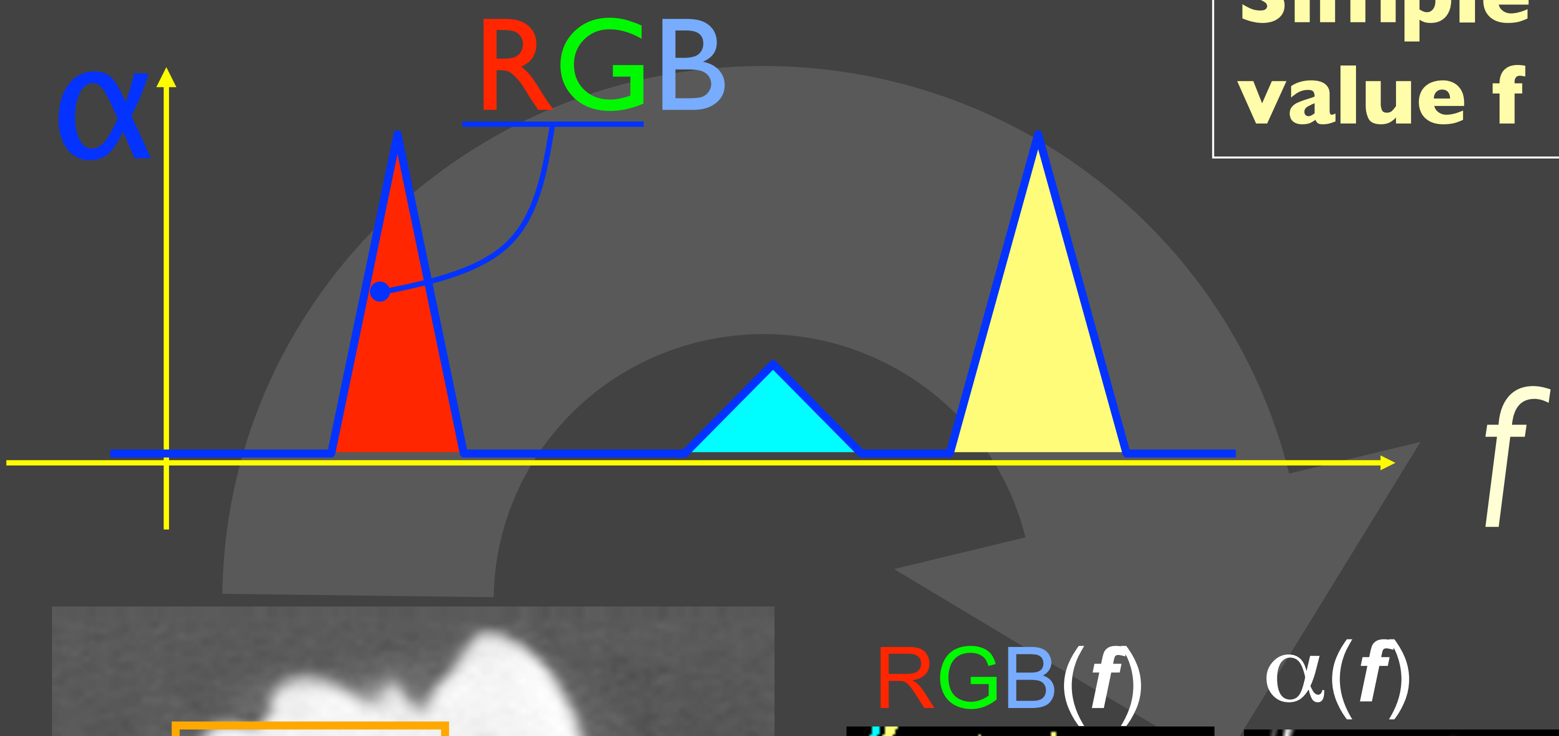
Simple (usual) case: Map data value f to color and opacity

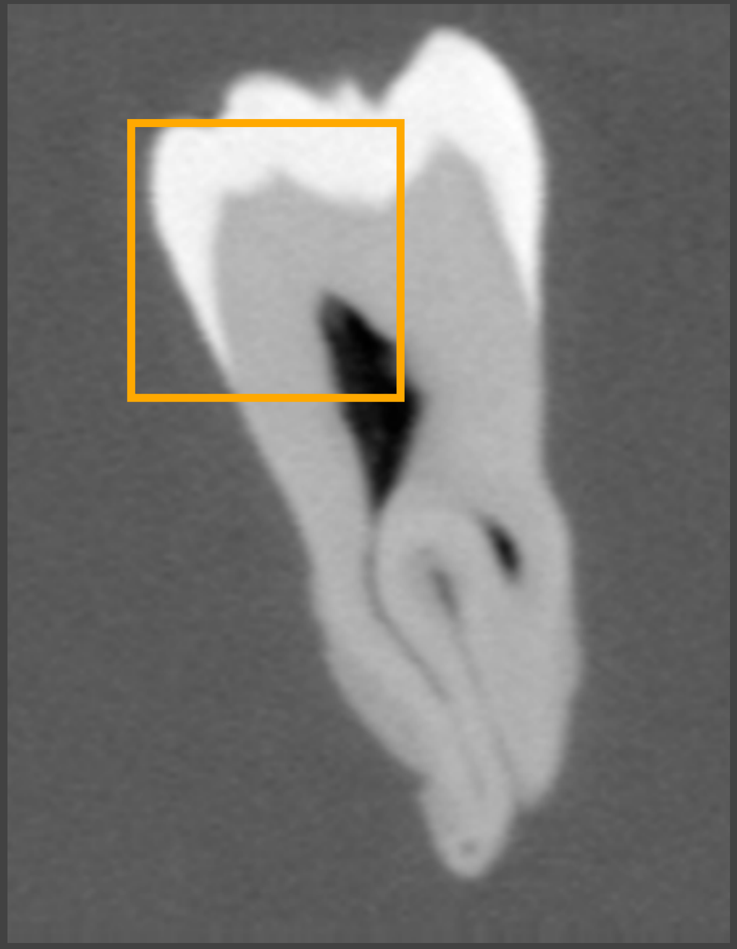



Human Tooth CT

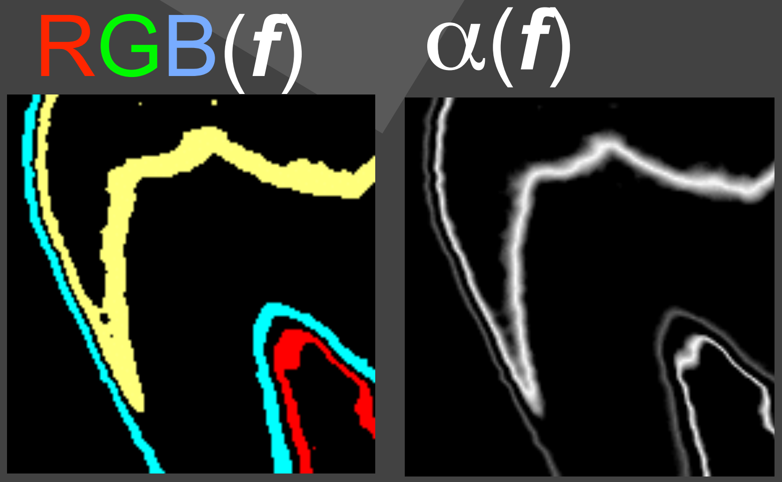
Transfer Functions (TFs)

Simple (usual) case: Map data value f to color and opacity



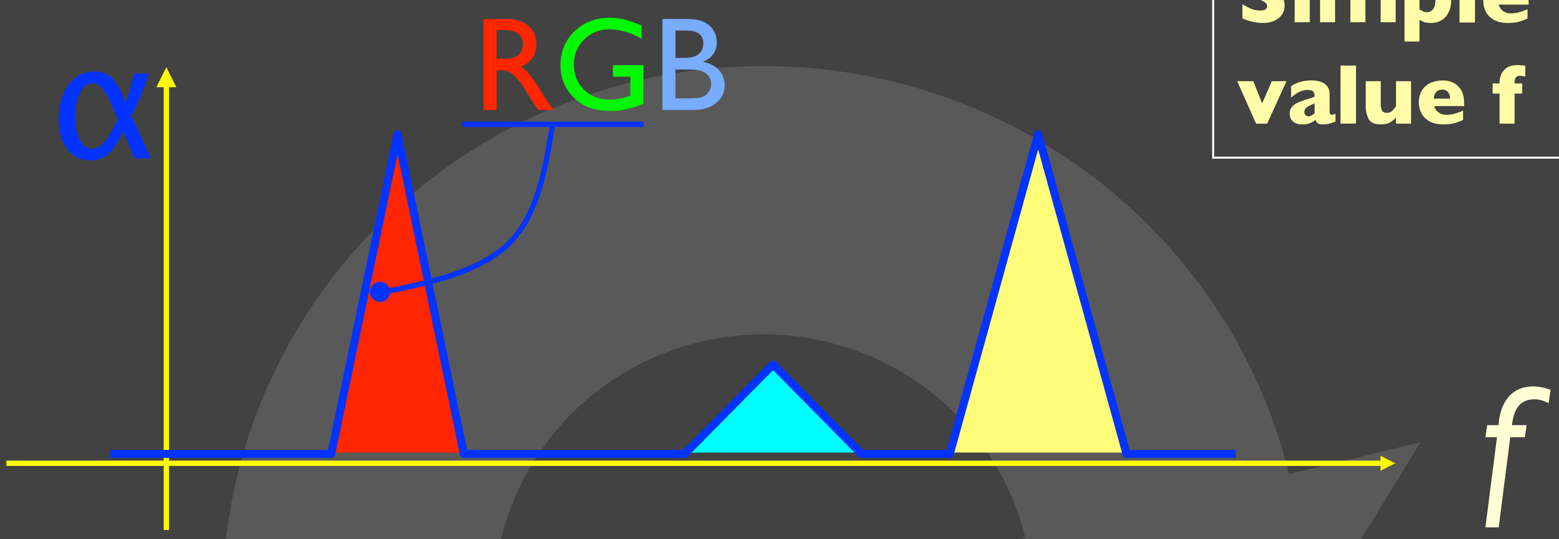


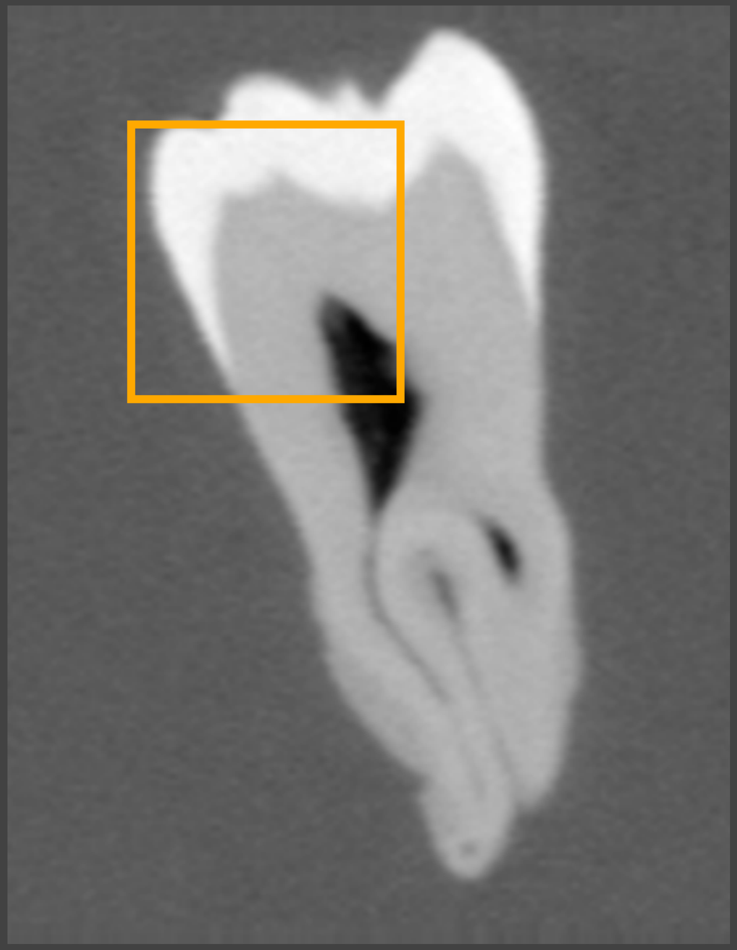
Human Tooth CT

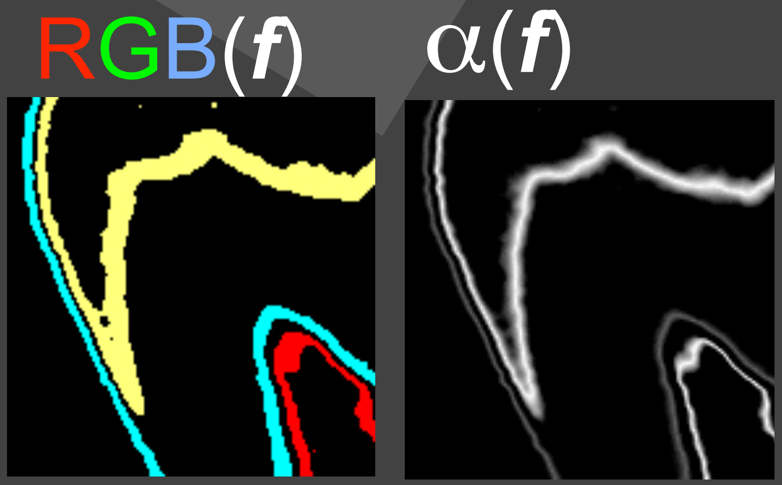


Transfer Functions (TFs)

Simple (usual) case: Map data value f to color and opacity



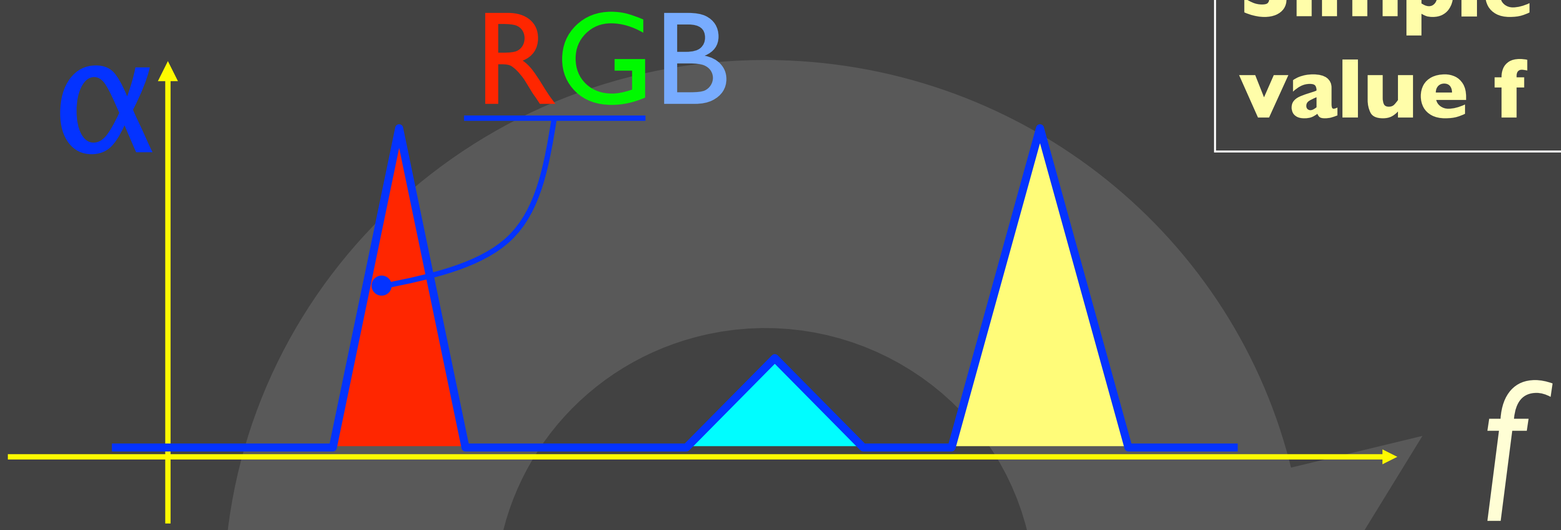

Human Tooth CT

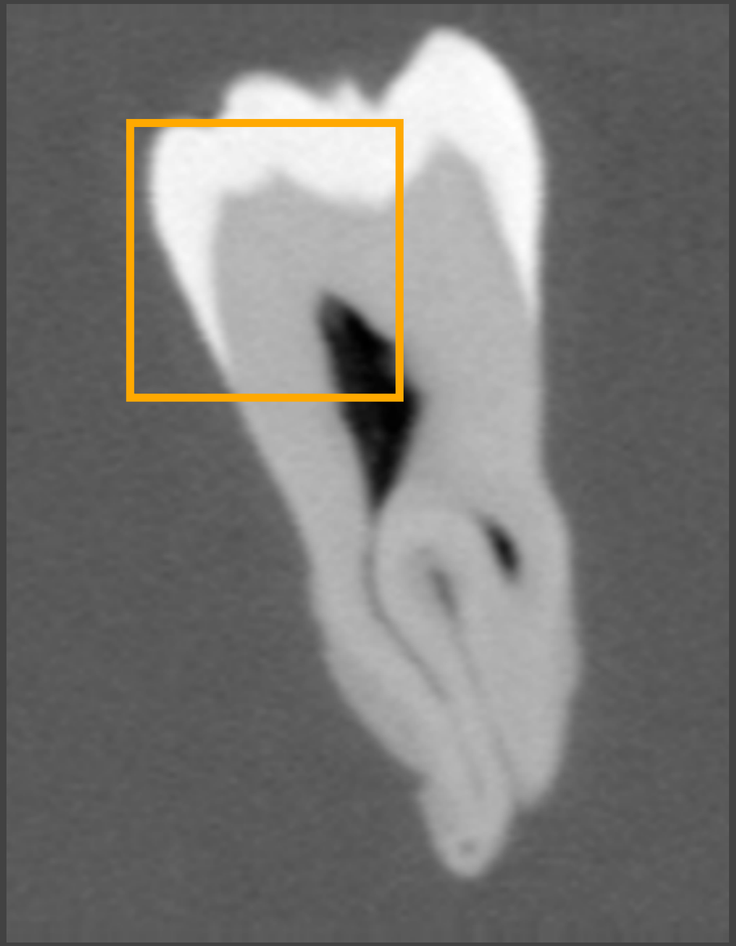


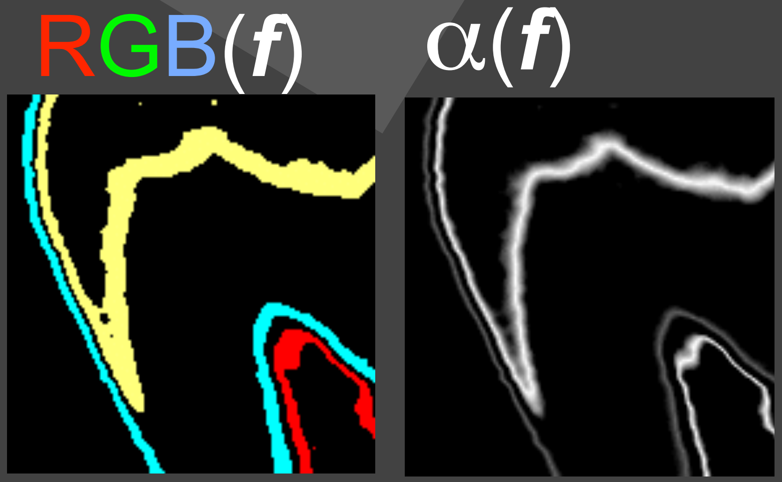
Shading, Compositing…

Transfer Functions (TFs)

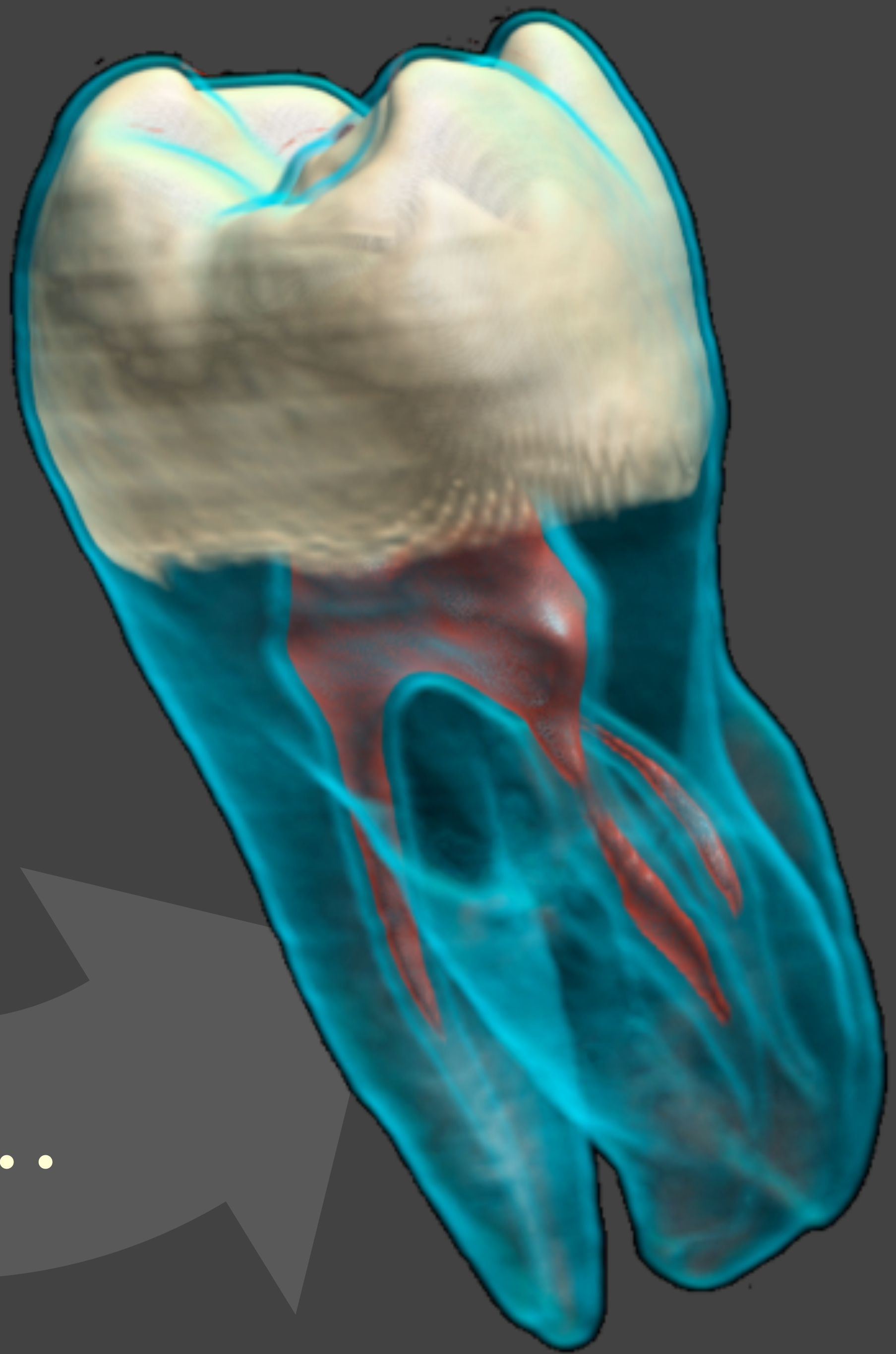
Simple (usual) case: Map data value f to color and opacity



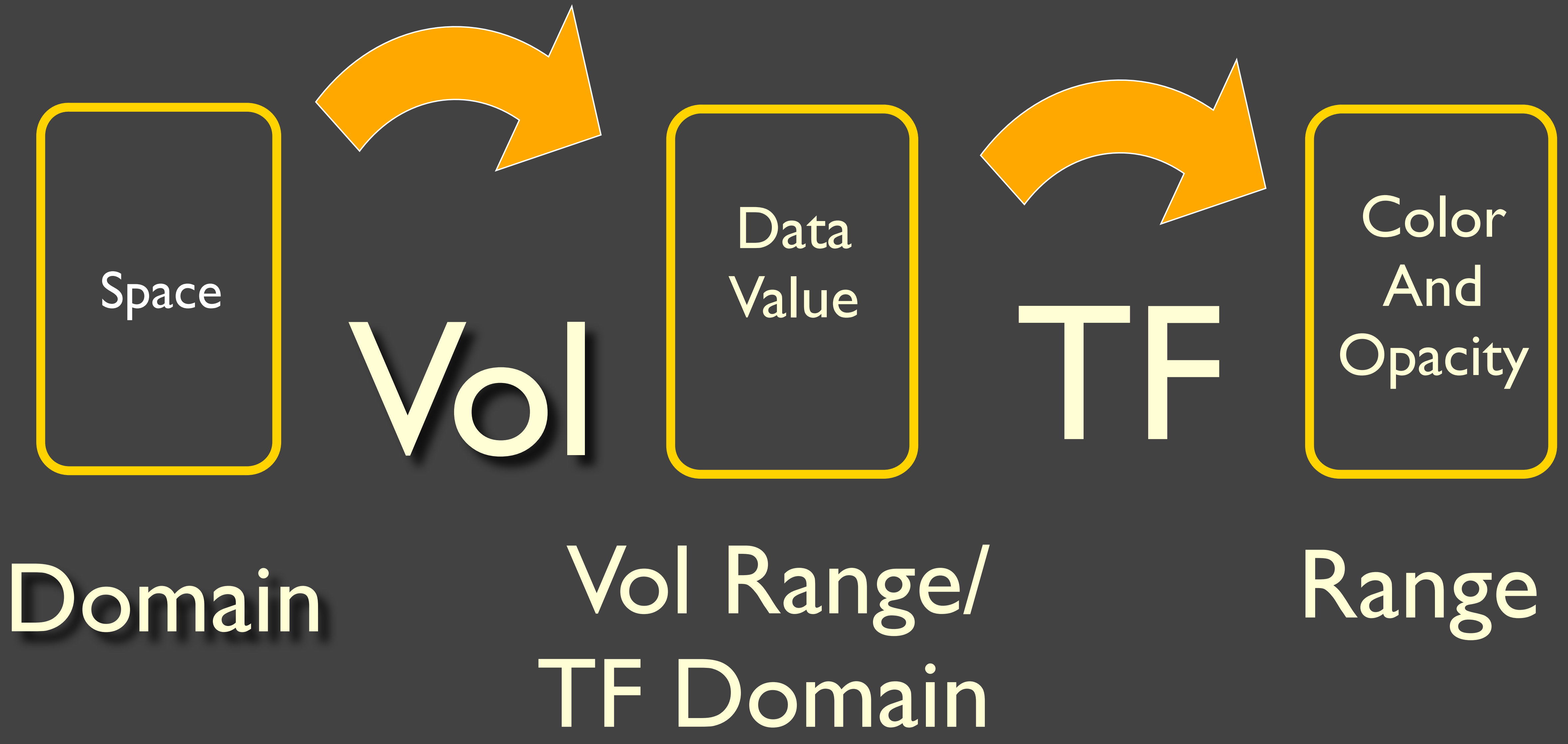

Human Tooth CT



Shading, Compositing…



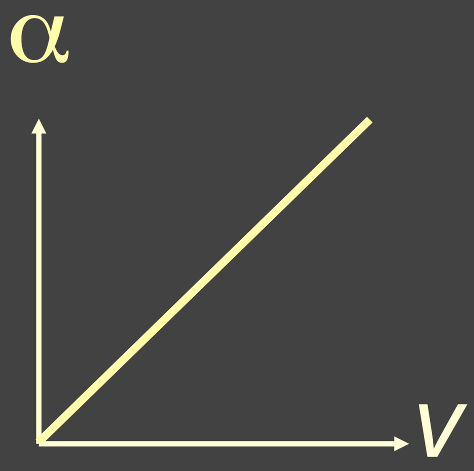
Basic Transfer Functions:

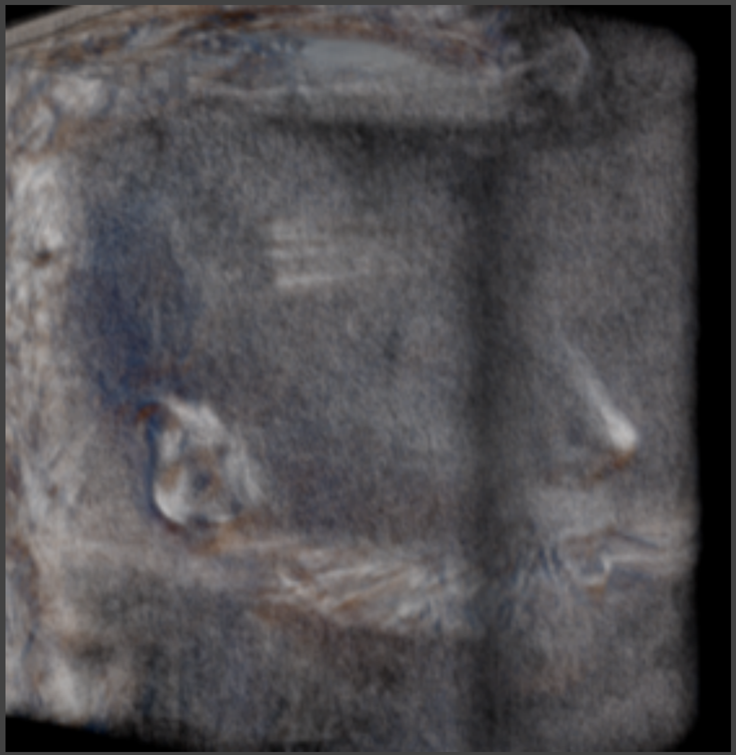


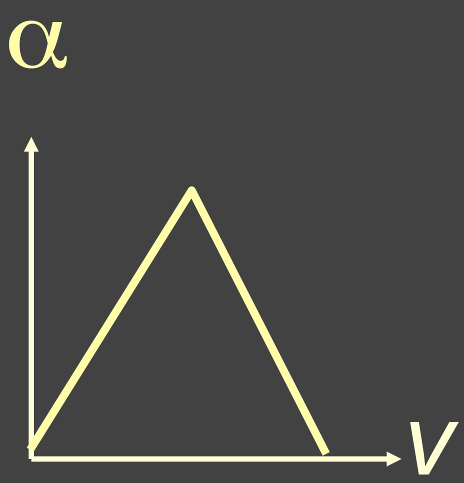
What Can Be Controlled by the Transfer Function?

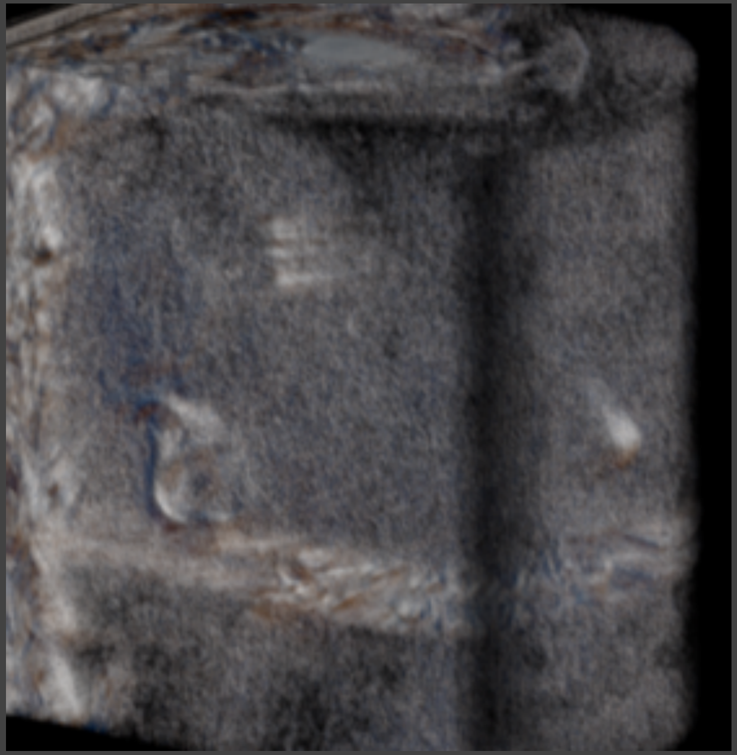
- Optical Properties: Anything that can be composited with a standard graphics operator (“over”)
  - Opacity:“opacity functions”
  - Color: Can help distinguish features
  - Phong parameters (ka, kd, ks)
  - Index of refraction

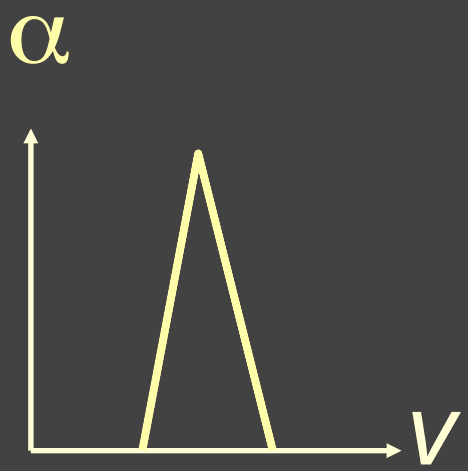
Setting Transfer Function: Hard

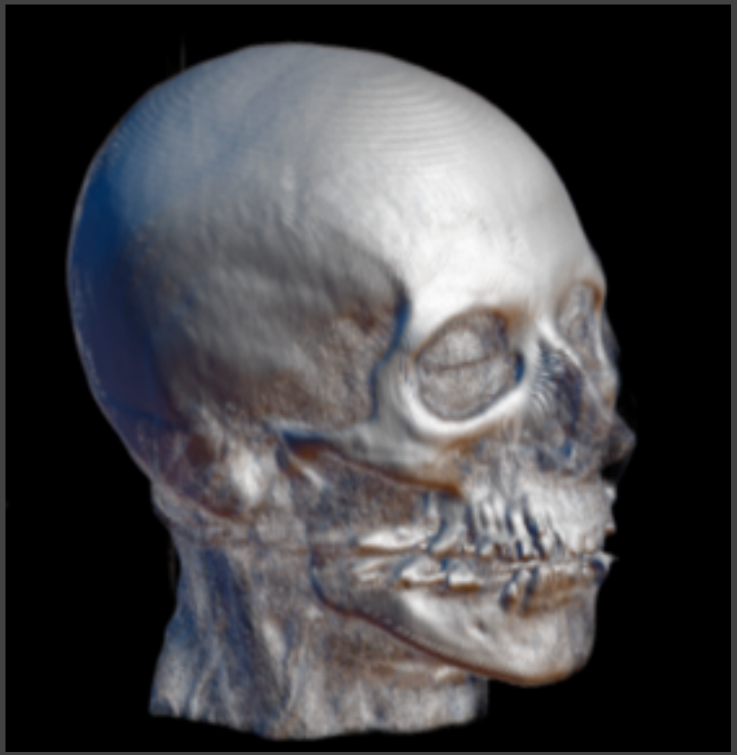


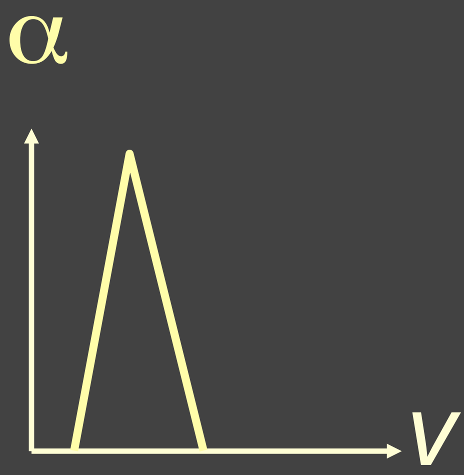


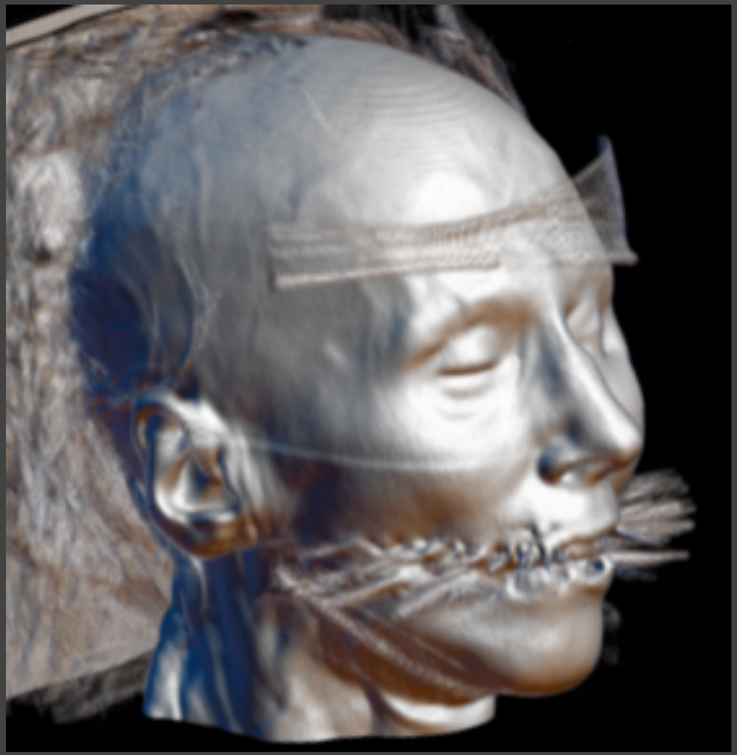




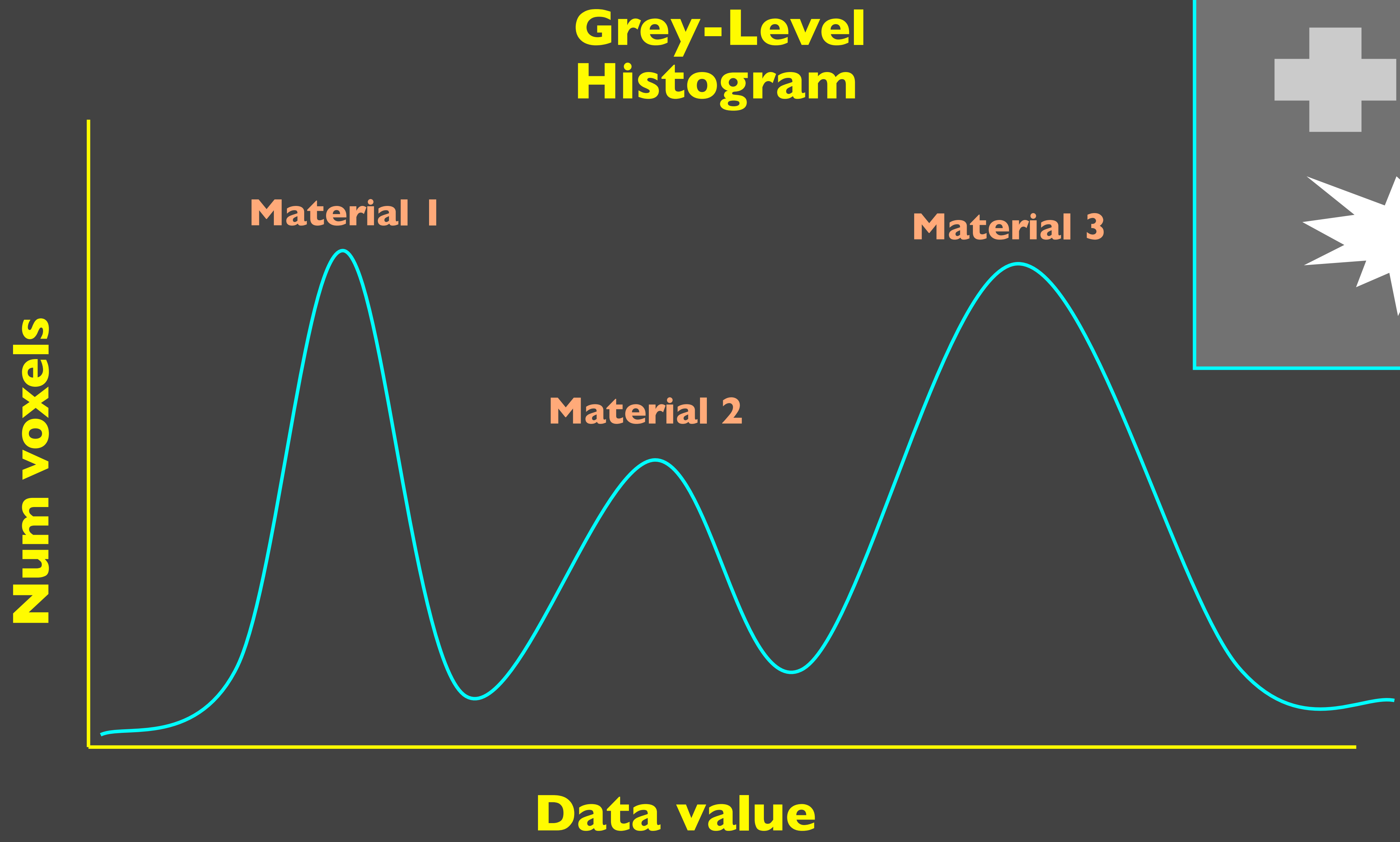




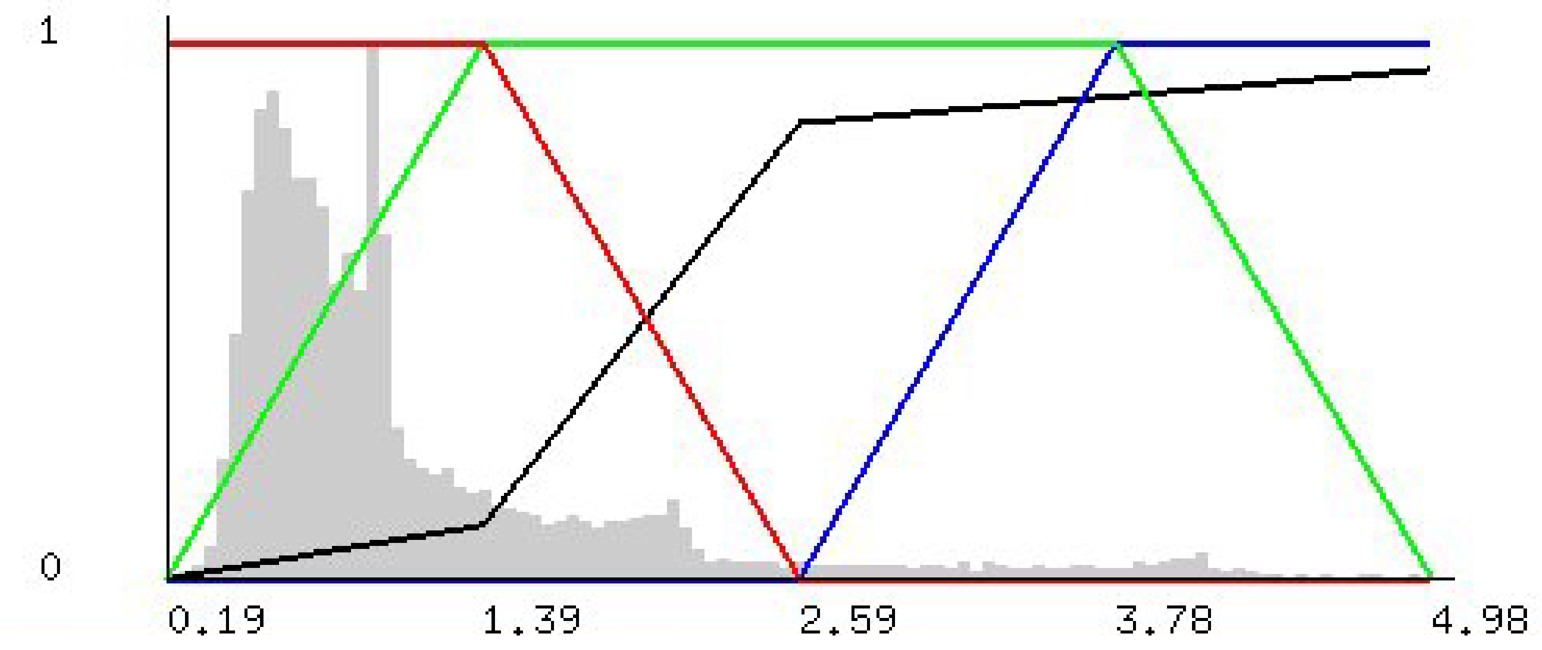




Volumes as Consisting of Materials

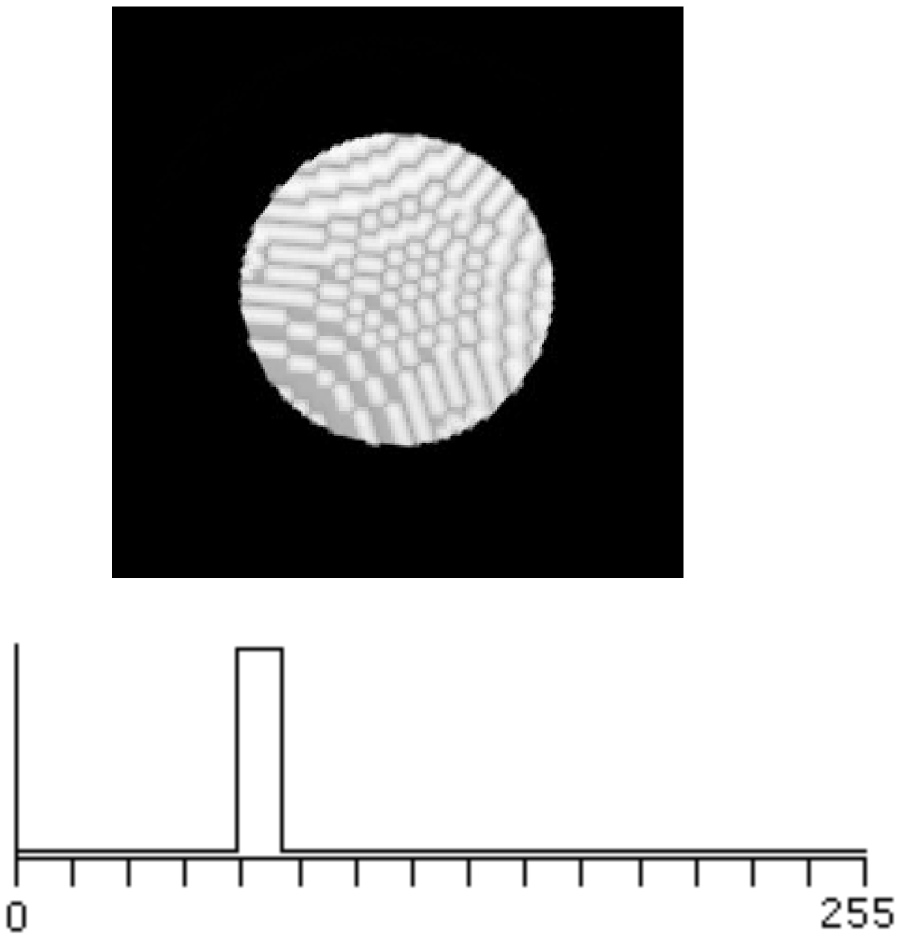


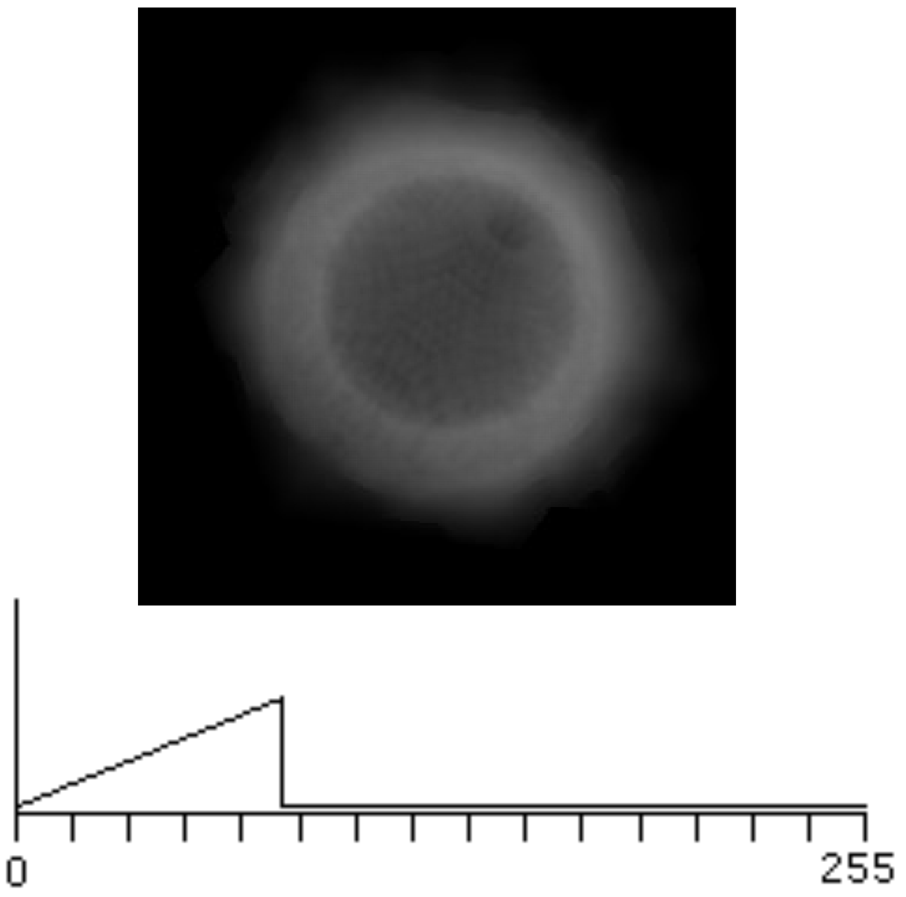
Transfer Function



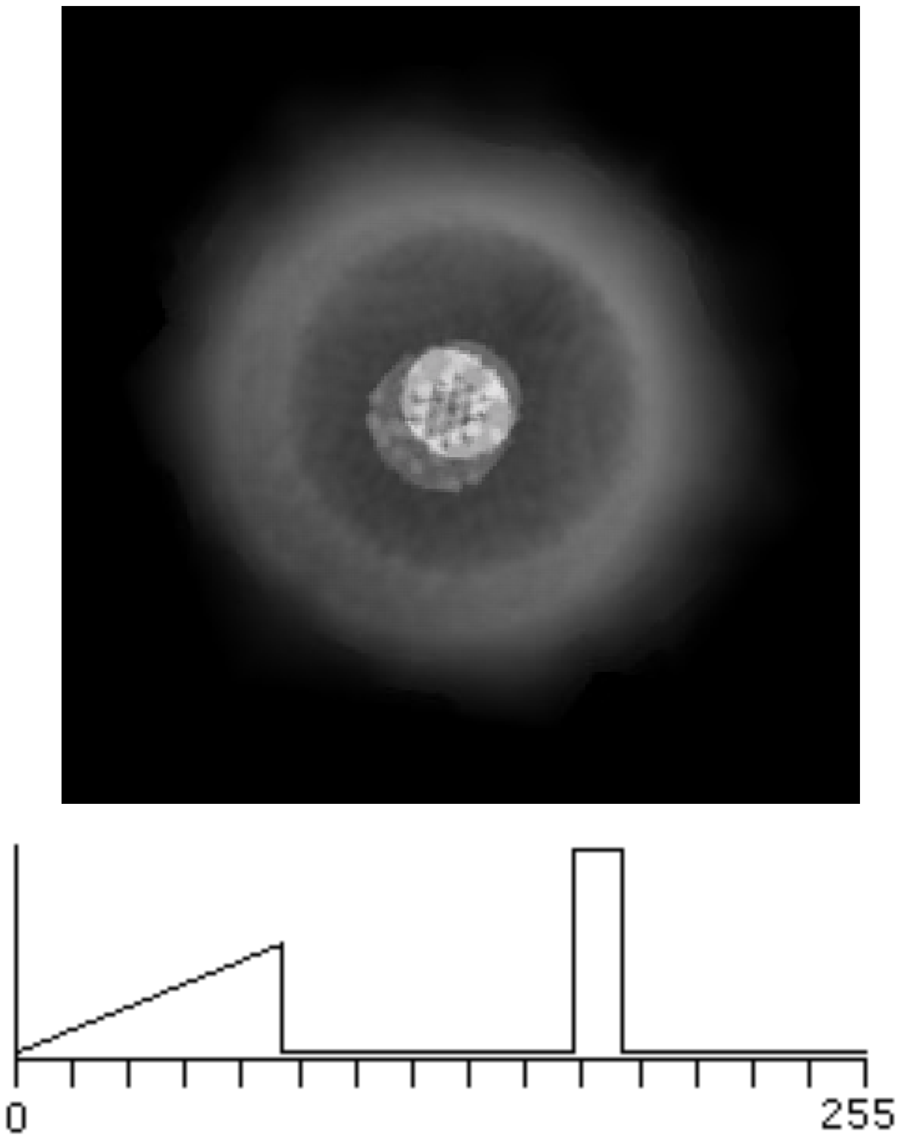
- RGB components
- Opacity
- Histogram helps in designing transfer function

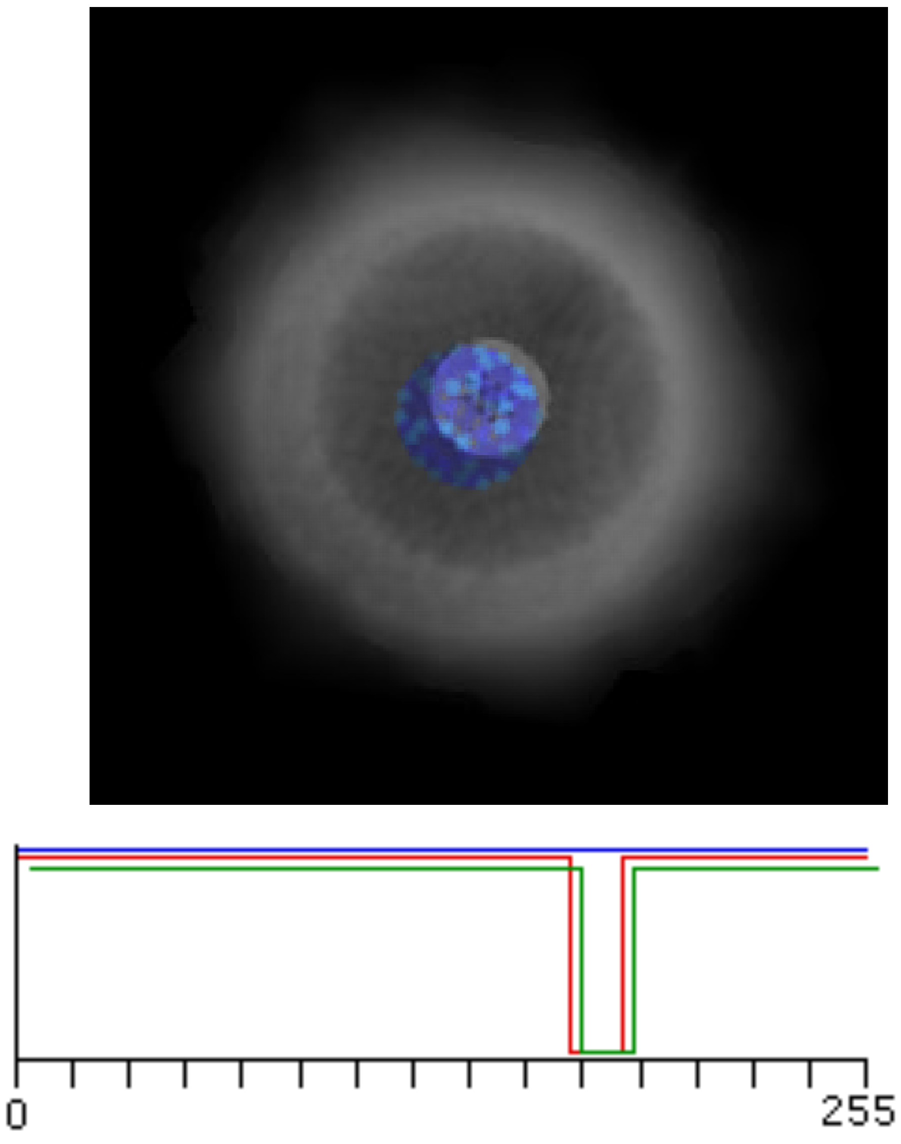
Transfer Function



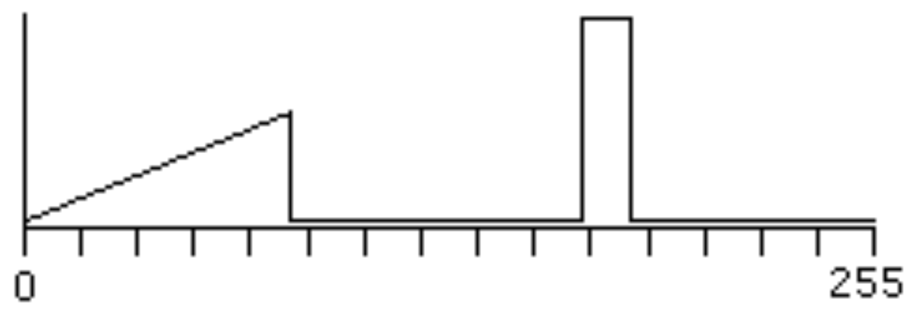


Transfer Function

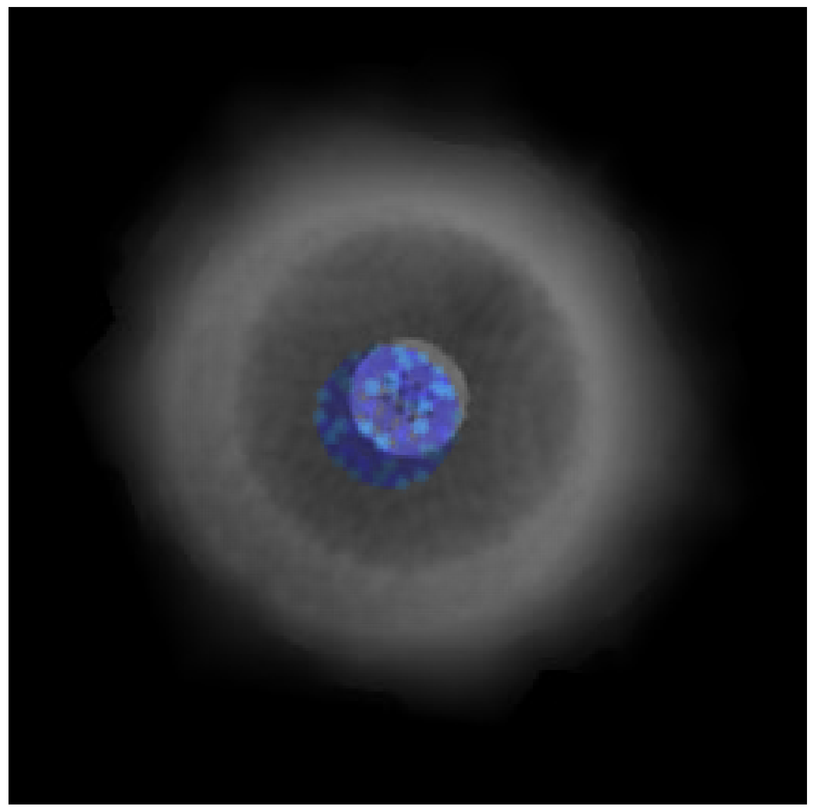


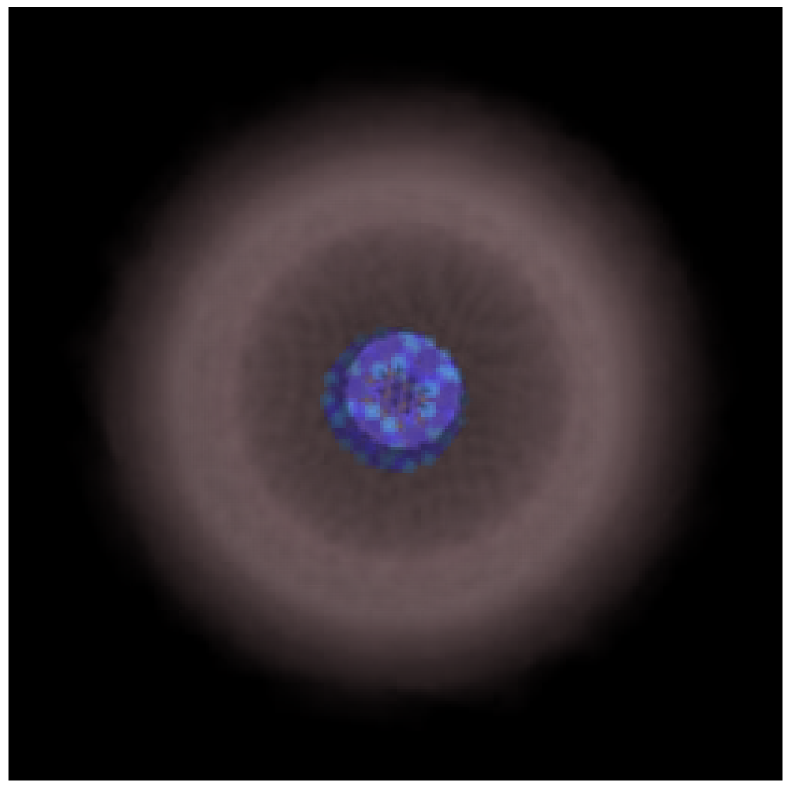


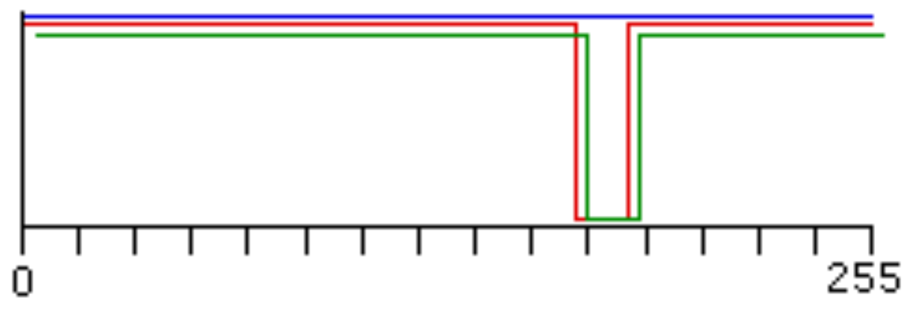
Transfer Function

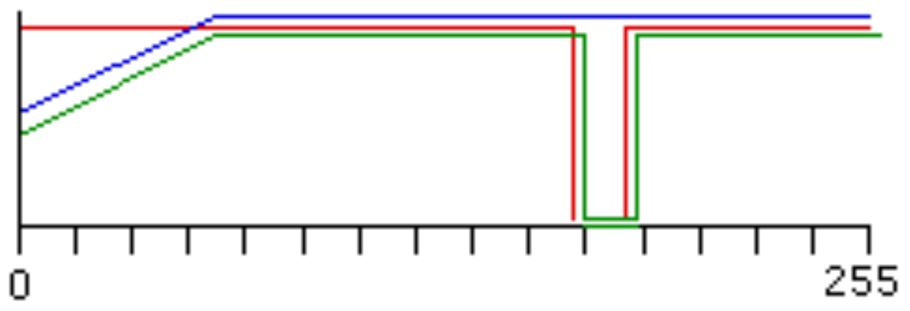


Different colors, same opacity



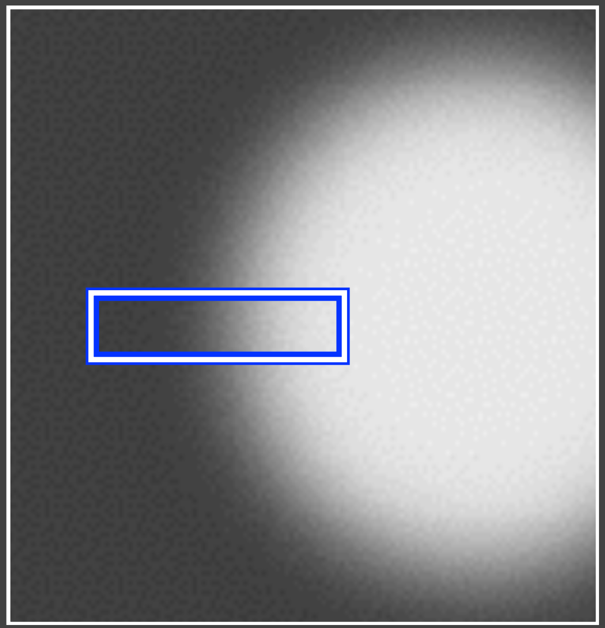


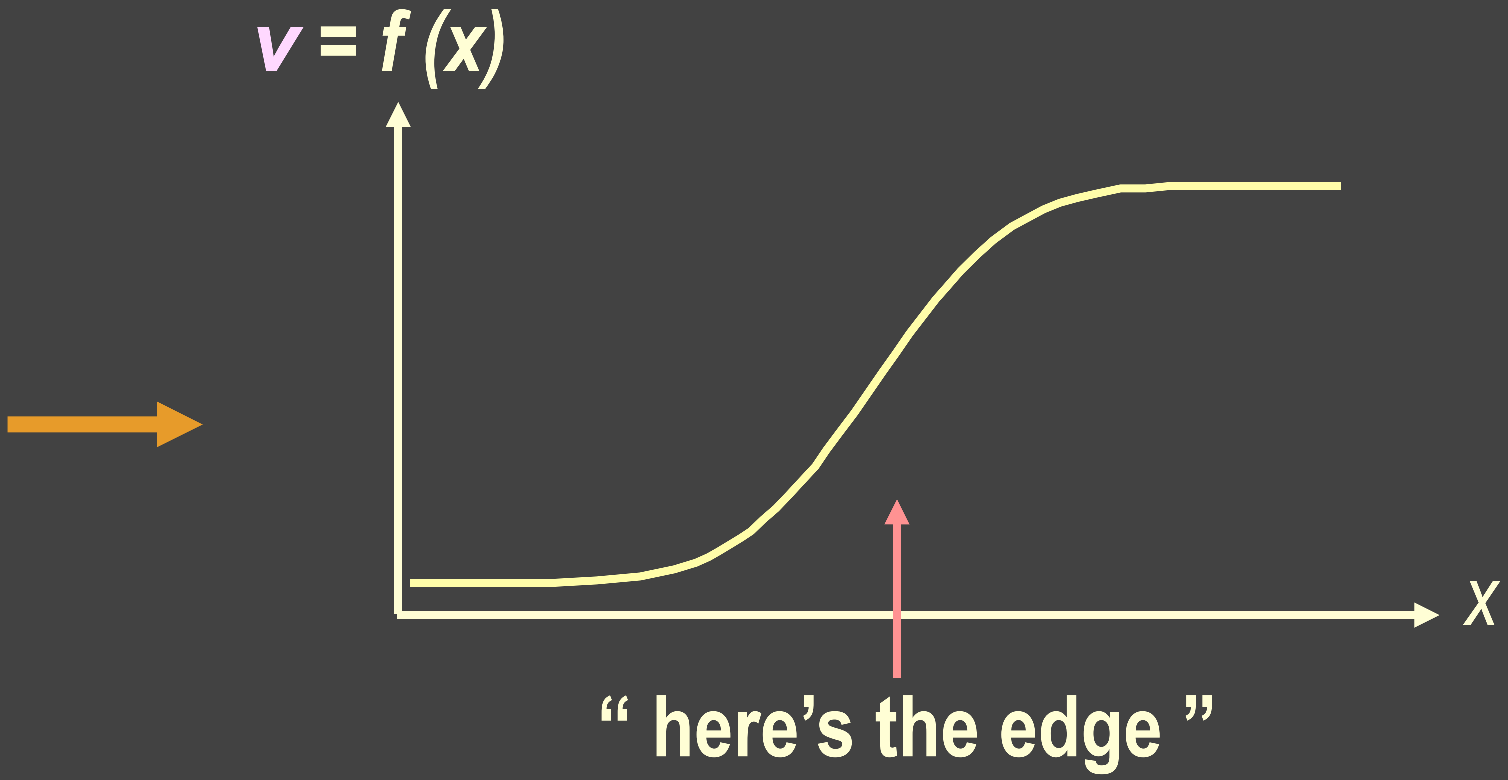




Finding edges: easy

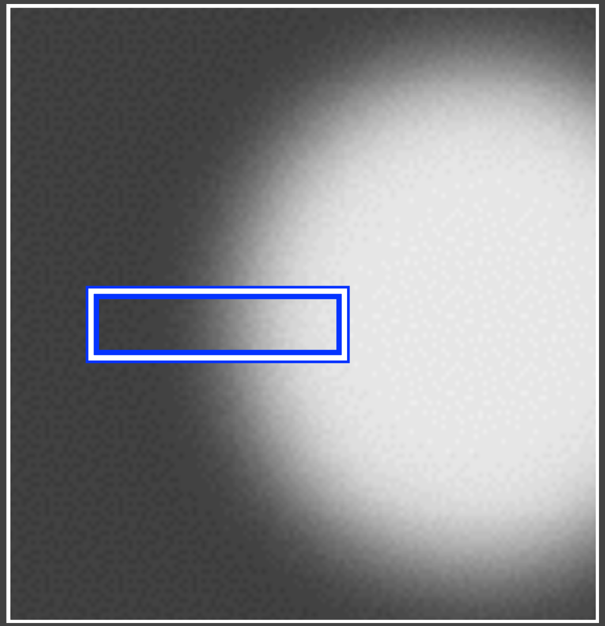
"Where's the edge?"

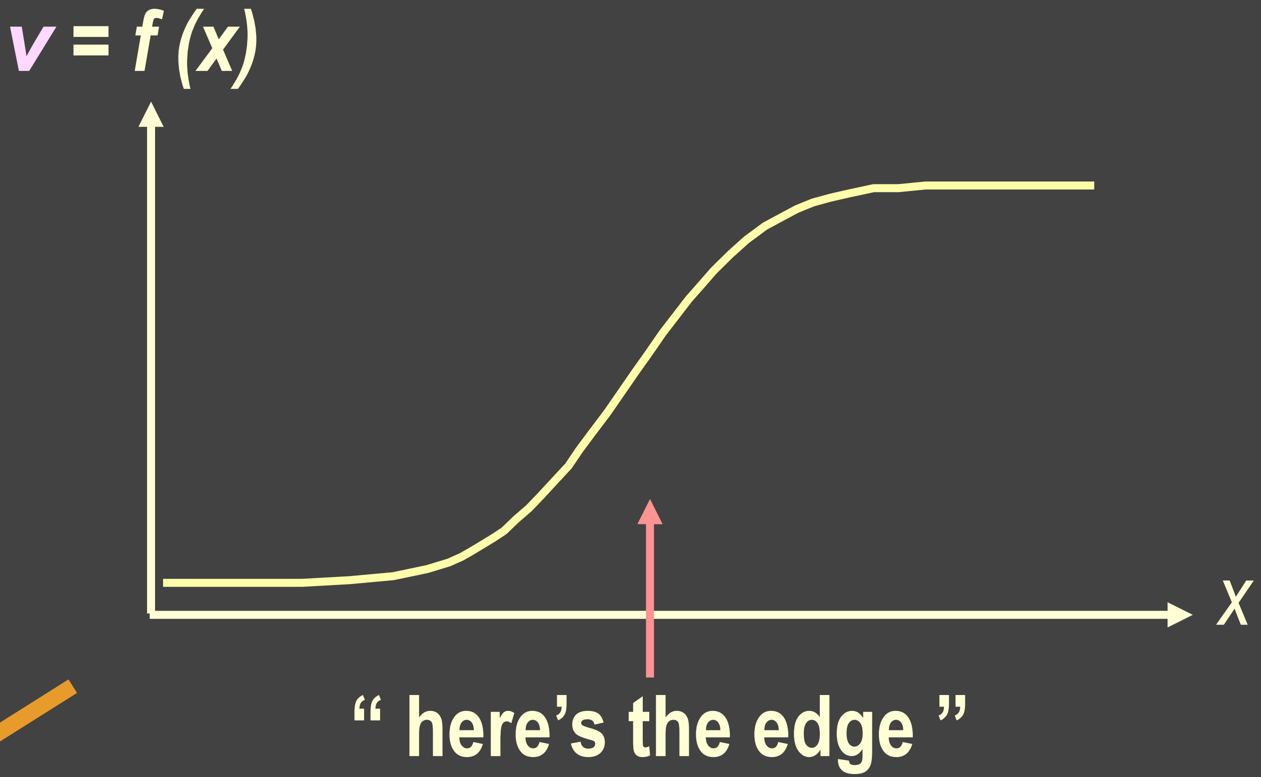


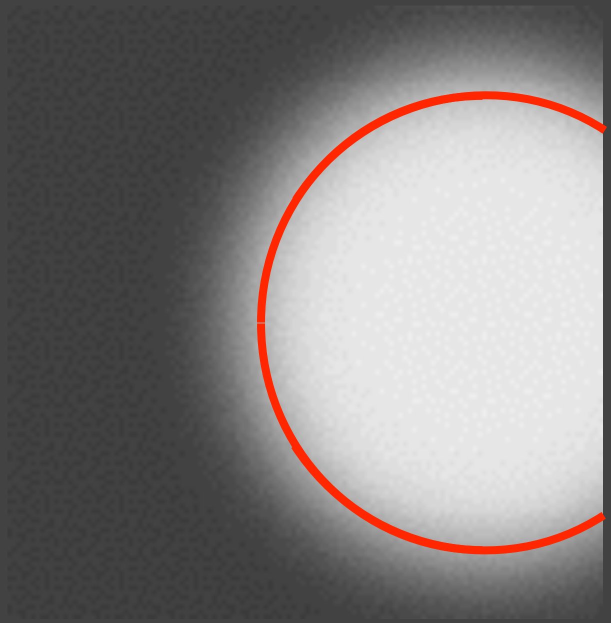


Finding edges: easy

"Where's the edge?"



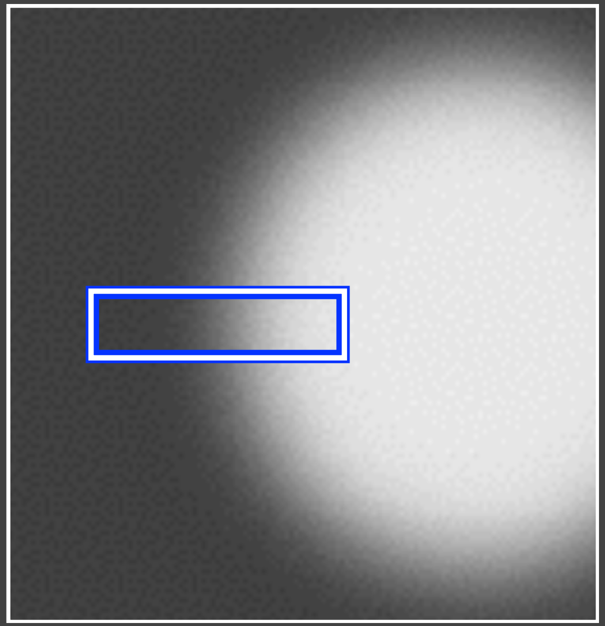


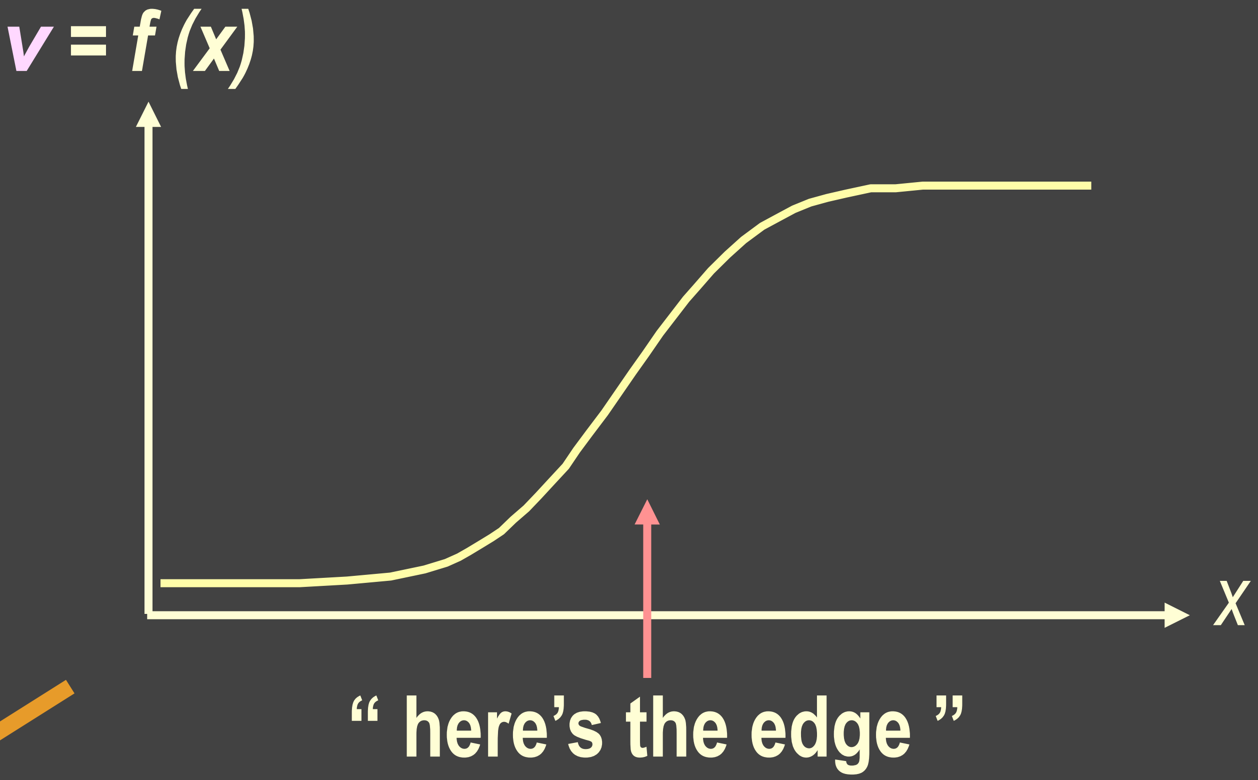


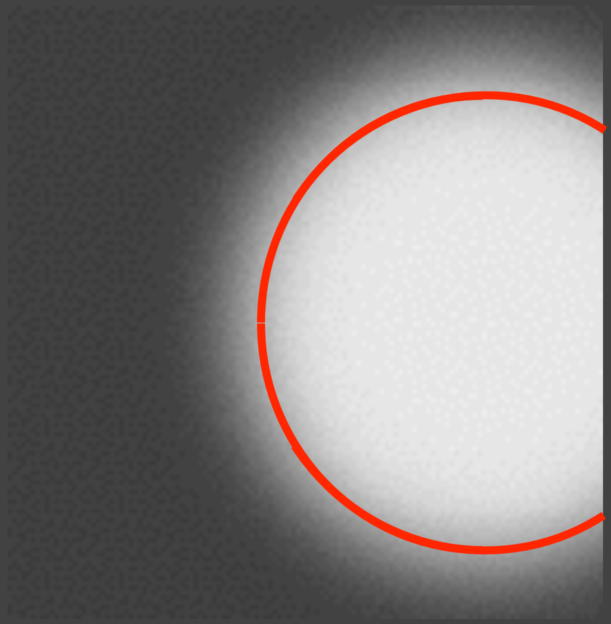
Result: edge pixels

Finding edges: easy

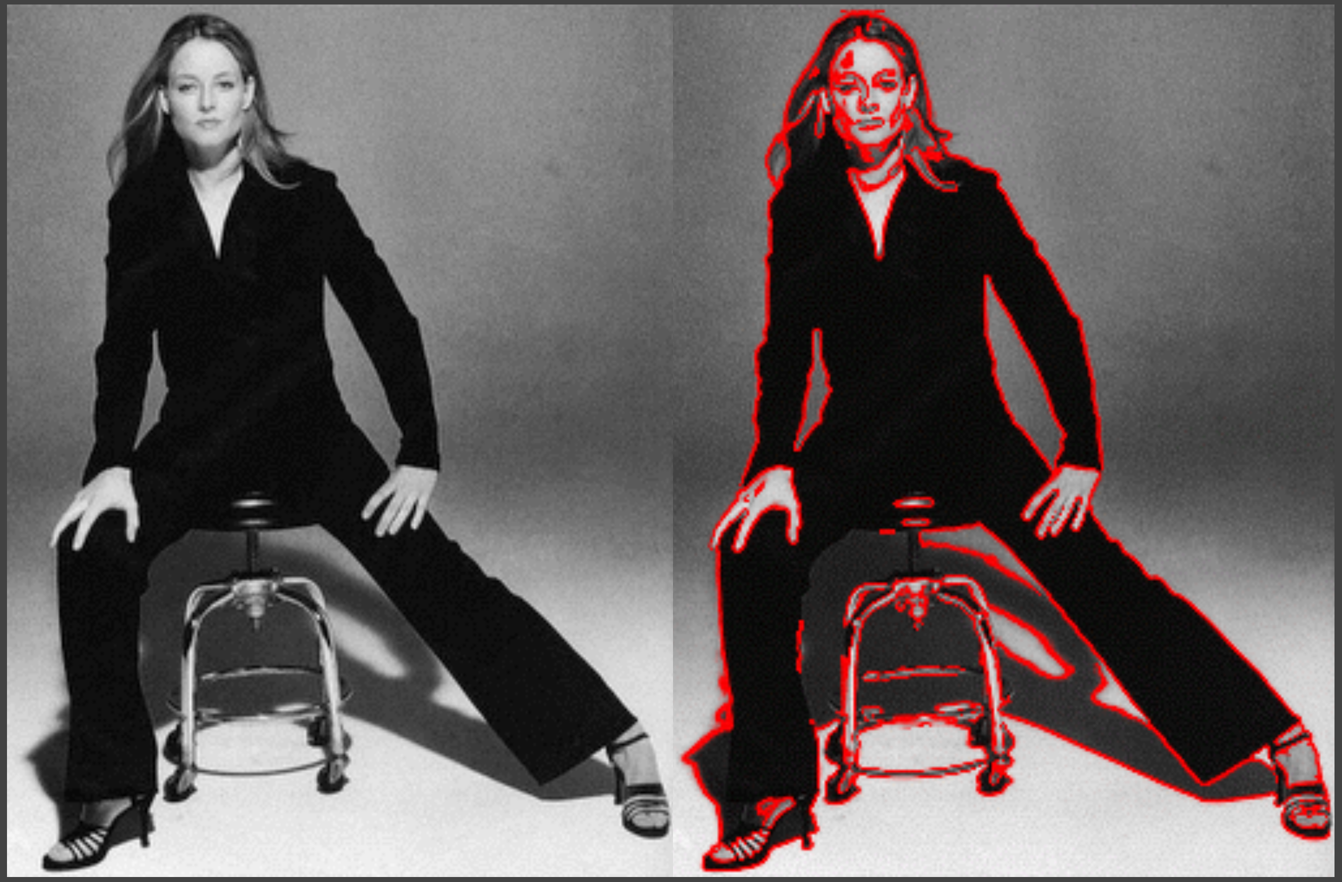
"Where's the edge?"



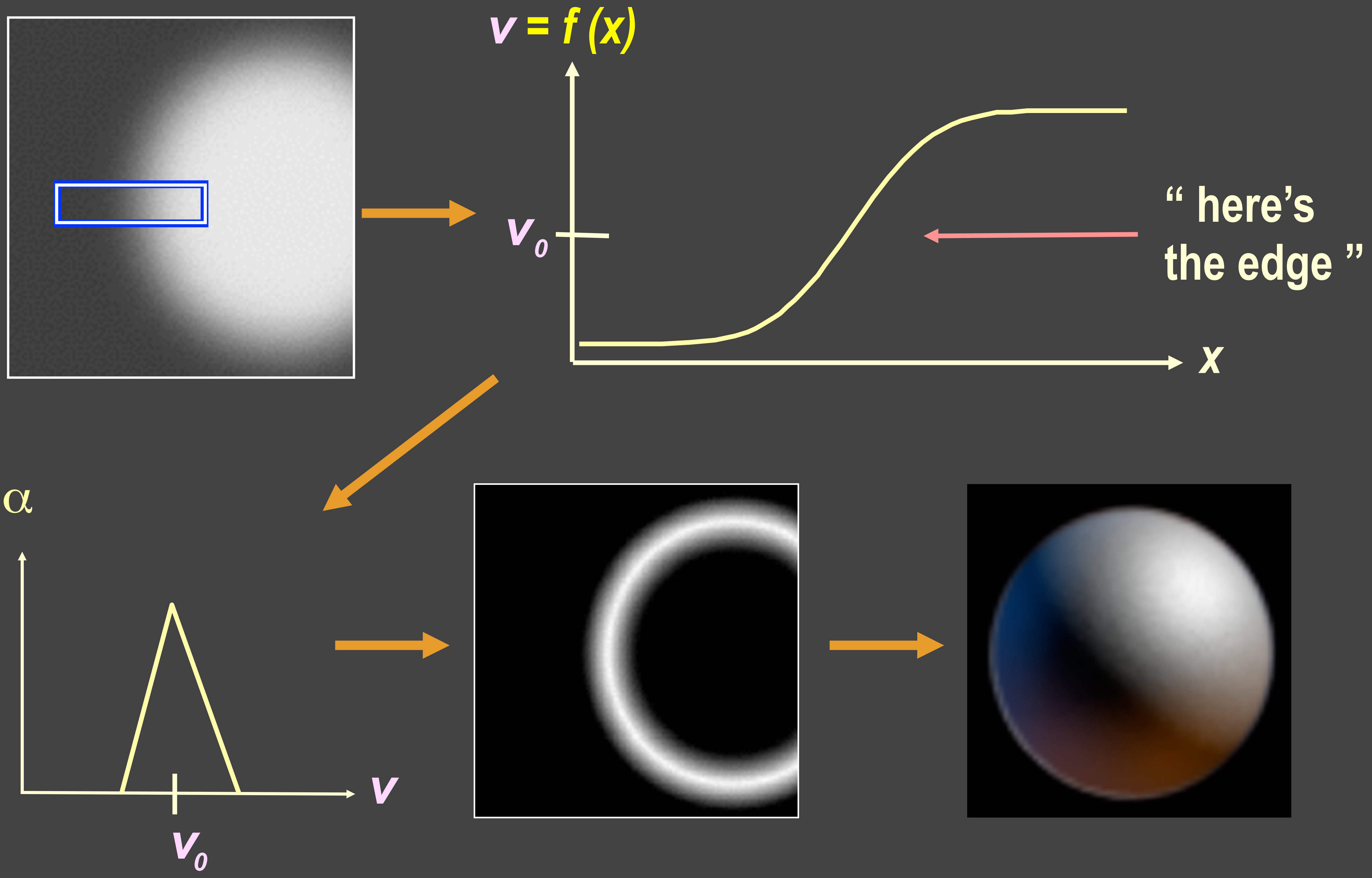




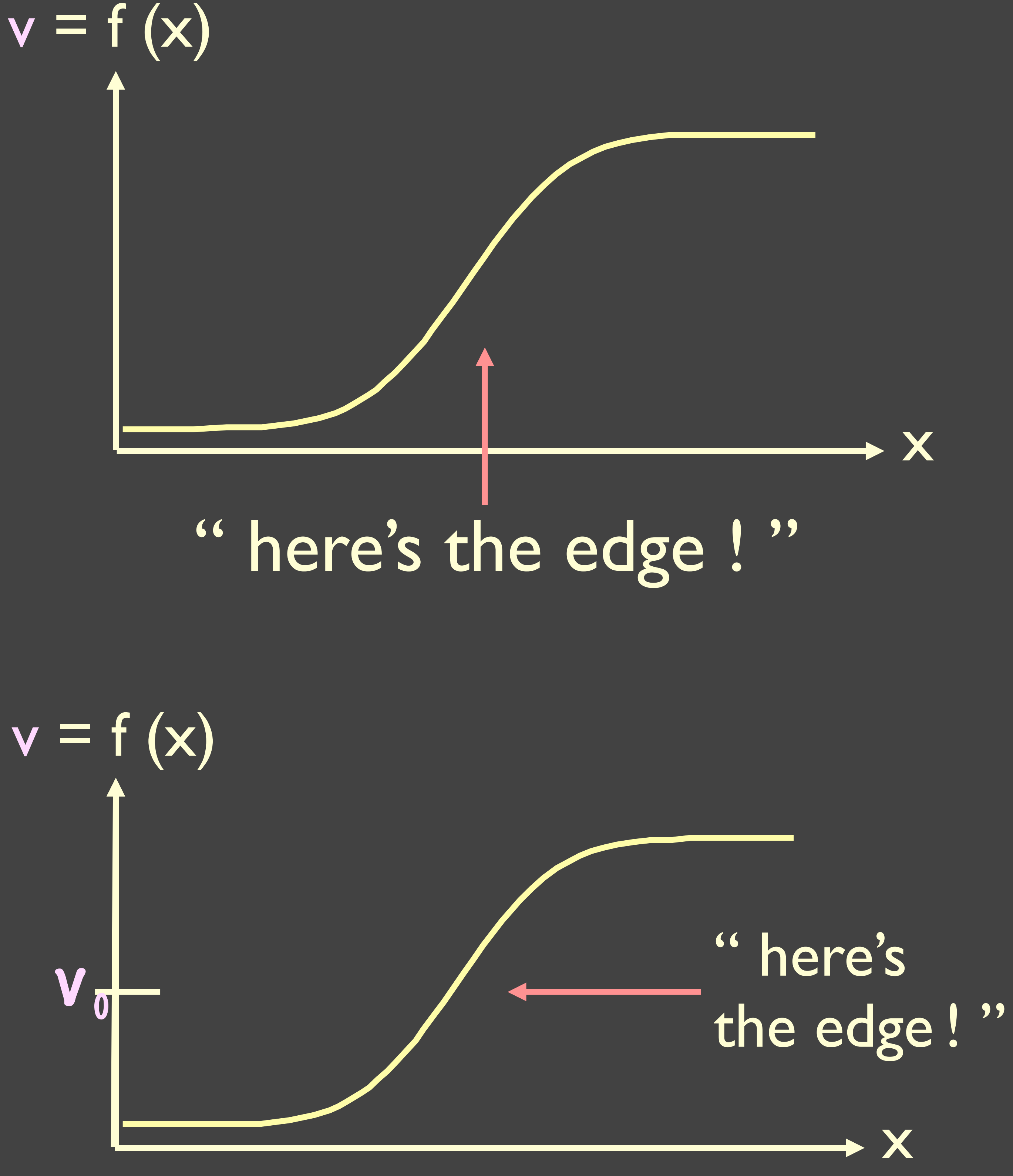
Result: edge pixels



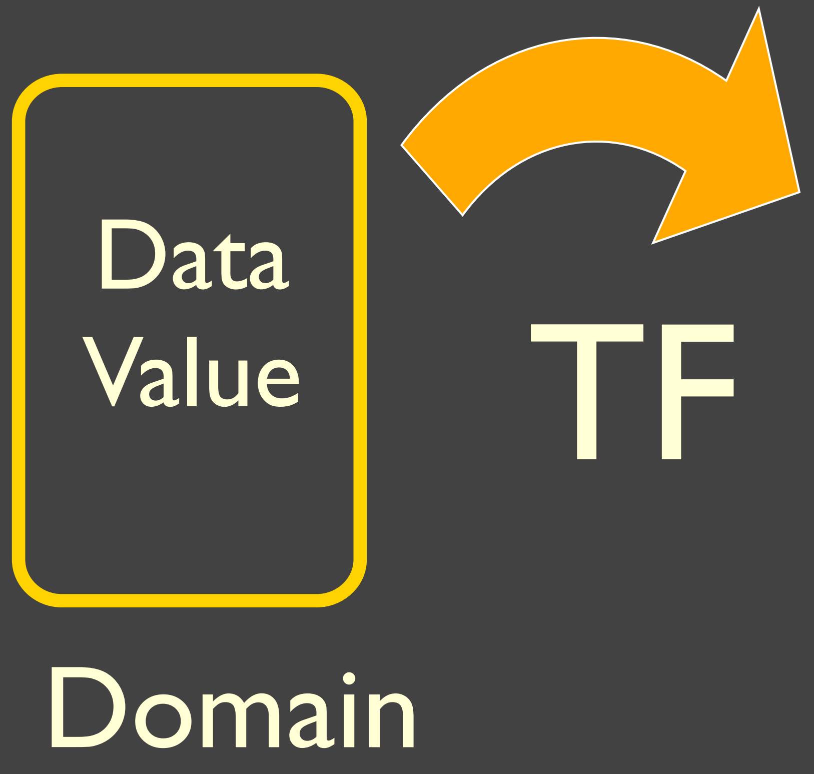
Transfer function Unintuitive



TFs as feature detection



Domain of the transfer function does not include position



What Makes Designing TF’s Challenging?

1. Non-spatial: spatial isolation doesn’t imply data value isolation

2. Many degrees of freedom

3. No constraints or guidance

4. Material uniformity assumption

Goals for TF Design

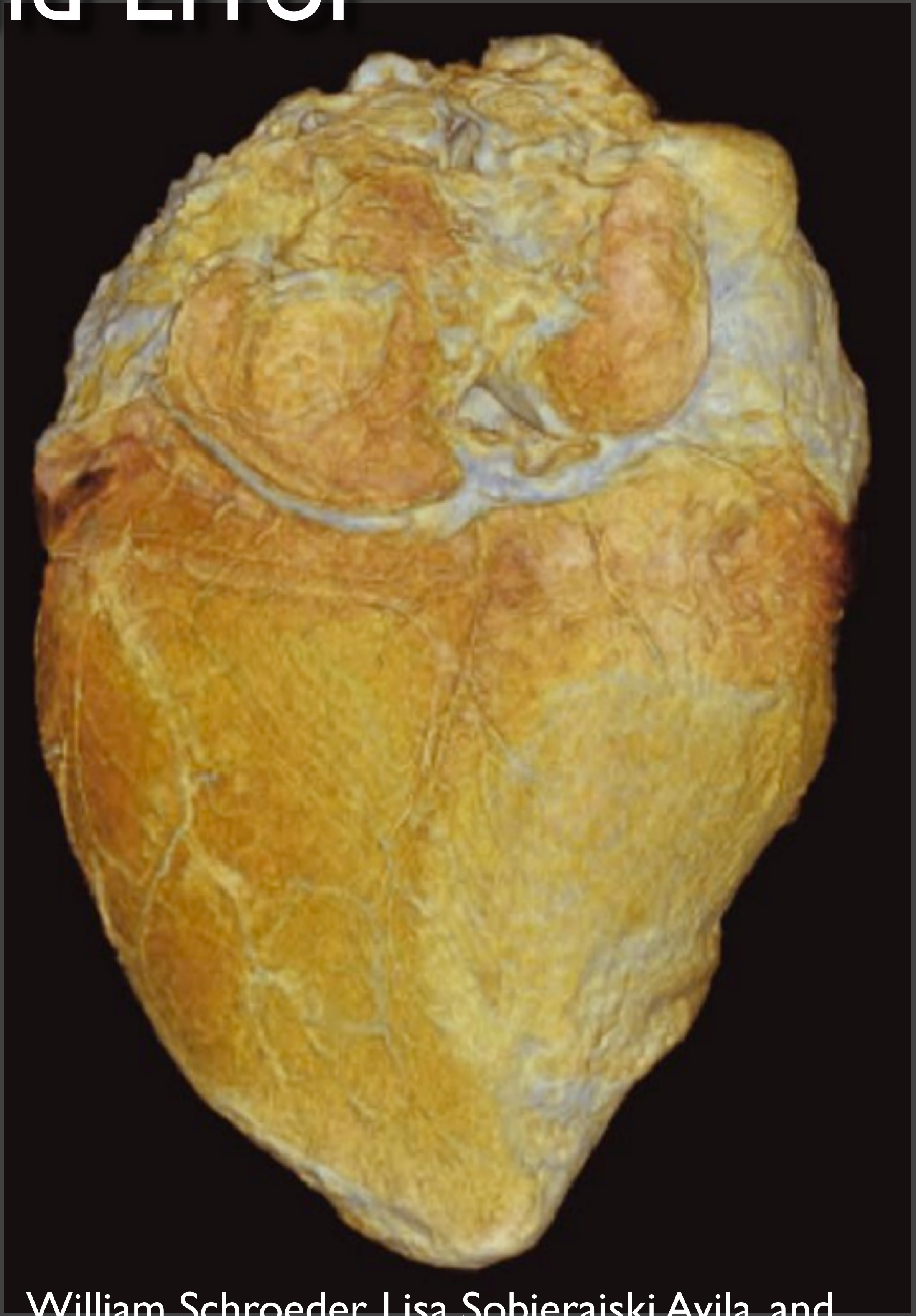
- Make good renderings easier to come by
- Make space of TFs less confusing
- Remove excess “flexibility”
- Provide one or more of:
  - Information
  - Guidance
  - Semi-automation / Automation

TF Techniques/Tools

1. **Trial and Error (manual)**
2. Image-Centric Approach
3. Data-Centric Approach

1. Trial and Error

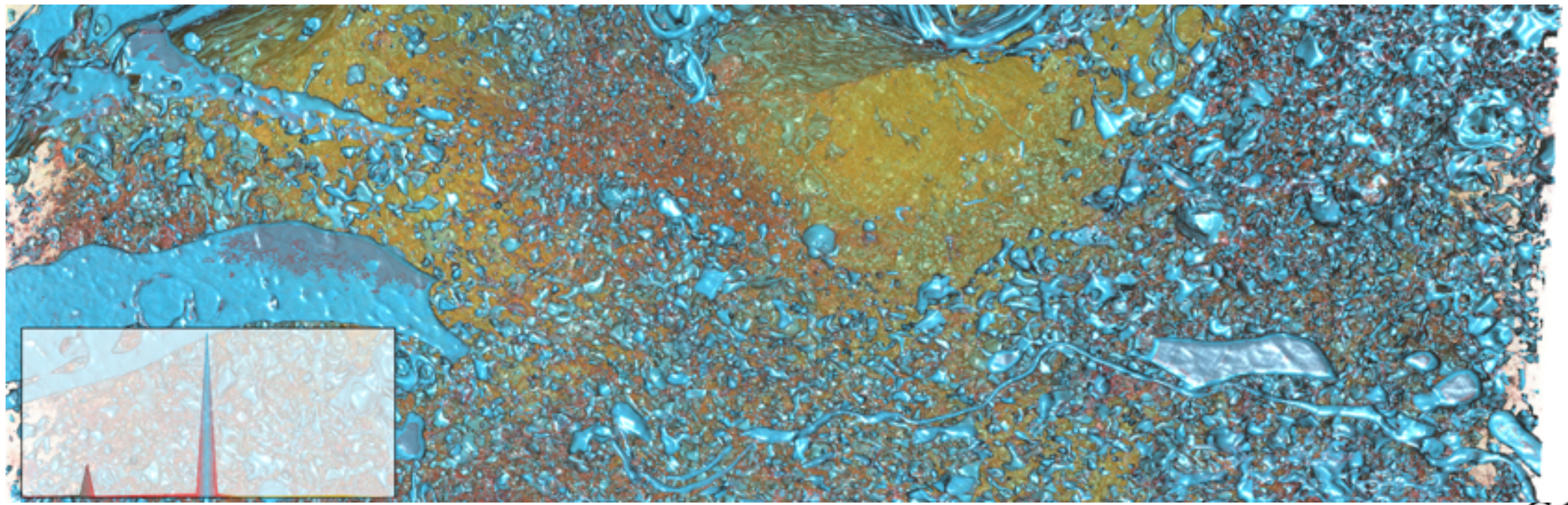
1. Manually edit graph of transfer function
2. Enforces learning by experience
3. Get better with practice
4. Can make terrific images



William Schroeder, Lisa Sobierajski Avila, and Ken Martin; Transfer Function Bake-off Vis '00

Nanovol demo (1D TF's)

- simple structured volume rendering with all the goodies
  - lighting, shading
  - acceleration structure (uniform grid)
  - preintegration (Engel et al.), "peak finding" (Knoll et al. Vis 09)
- C++/OpenGL/GLSL source code:
  http://www.sci.utah.edu/~knolla/nanovol.tgz



Aaron Knoll, Younis Hijazi, Rolf Westerteiger, Mathias Schott, Charles Hansen and Hans Hagen
Volume Ray Casting with Peak Finding and Differential Sampling. IEEE Vis 2009

www.sci.utah.edu

Image-centric

Specify TFs via the resulting renderings

- Genetic Algorithms ("Generation of Transfer Functions with Stochastic Search Techniques", He, Hong, *et al.*: Vis '96)
- Design Galleries (Marks, Andalman, Beardsley, *et al.*: SIGGRAPH '97; Pfister: Transfer Function Bake-off Vis '00)
- Thumbnail Graphs + Spreadsheets ("A Graph Based Interface…", Patten, Ma: Graphics Interface '98; "Image Graphs…", Ma: Vis '99; Spreadsheets for Vis: Vis '00, TVCG July '01)
- Thumbnail Parameterization ("Mastering Transfer Function Specification Using VolumePro Technology", König, Gröller: Spring Conference on Computer Graphics '01)

TF Techniques/Tools

1. Trial and Error (manual)
2. Image-Centric Approach
3. **Data-Centric Approach**

Data-centric

Specify TF by analyzing volume data itself

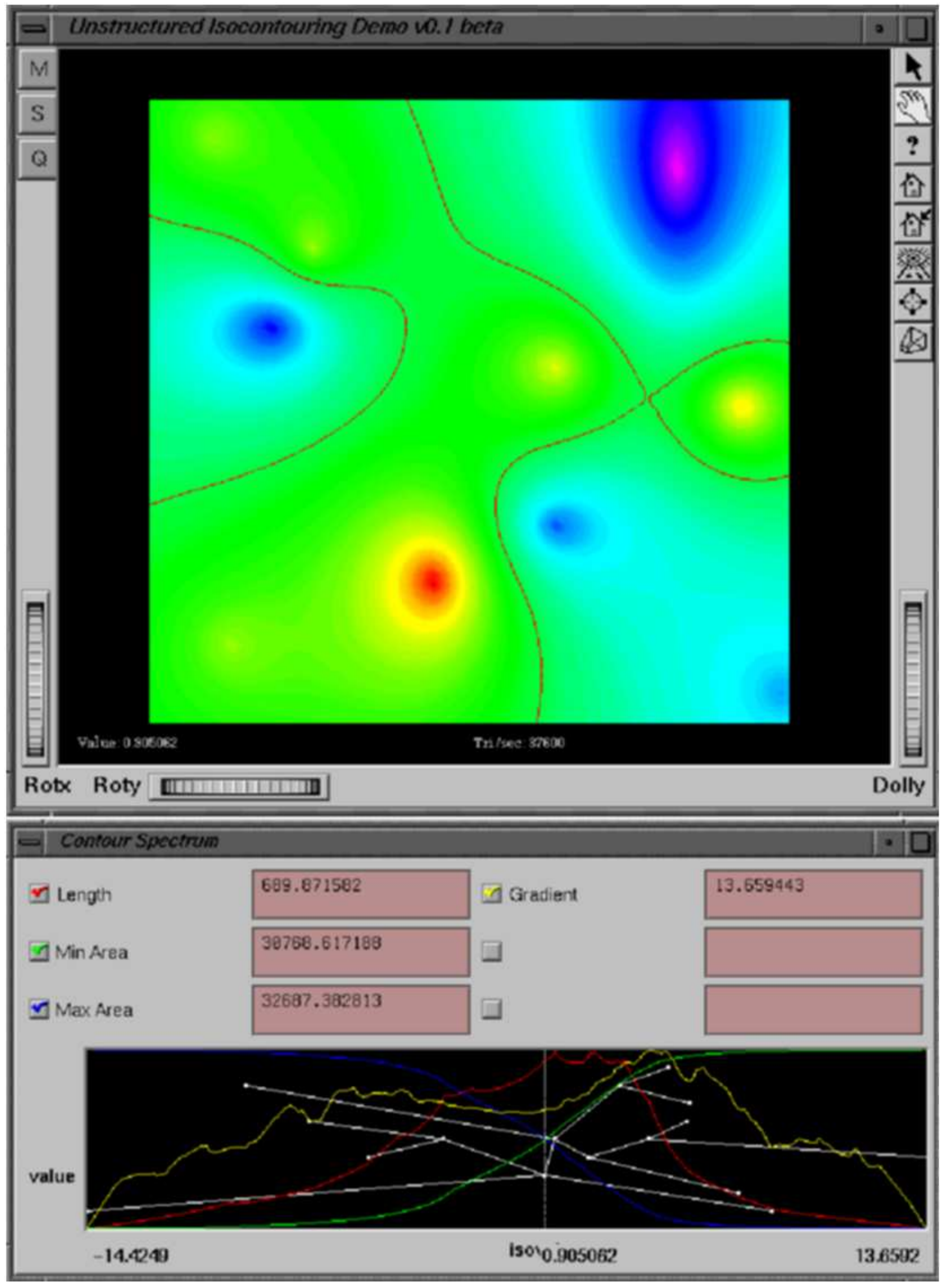
1. Salient Isovalues:
   - Contour Spectrum (Bajaj, Pascucci, Schikore: Vis ’97)
   - Statistical Signatures (“Salient Iso-Surface Detection Through Model-Independent Statistical Signatures”, Tenginaki, Lee, Machiraju: Vis ’01)
   - Other computational methods (“Fast Detection of Meaningful Isosurfaces for Volume Data Visualization”, Pekar, Wiemker, Hempel: Vis ’01)
2. “Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering” (Kindlmann, Durkin: VolVis ’98; Kindlmann MS Thesis ’99; Transfer Function Bake-Off Panel: Vis ‘00)

Salient Isovalues

What are the “best” isovalues for extracting the main structures in a volume dataset?

Contour Spectrum (Bajaj, Pascucci, Schikore: Vis ’97; Transfer Function Bake-Off: Vis ’00)

- Efficient computation of isosurface metrics
    - Area, enclosed volume, gradient surface integral, etc.
- Efficient connected-component topological analysis
- Interface itself concisely summarizes data



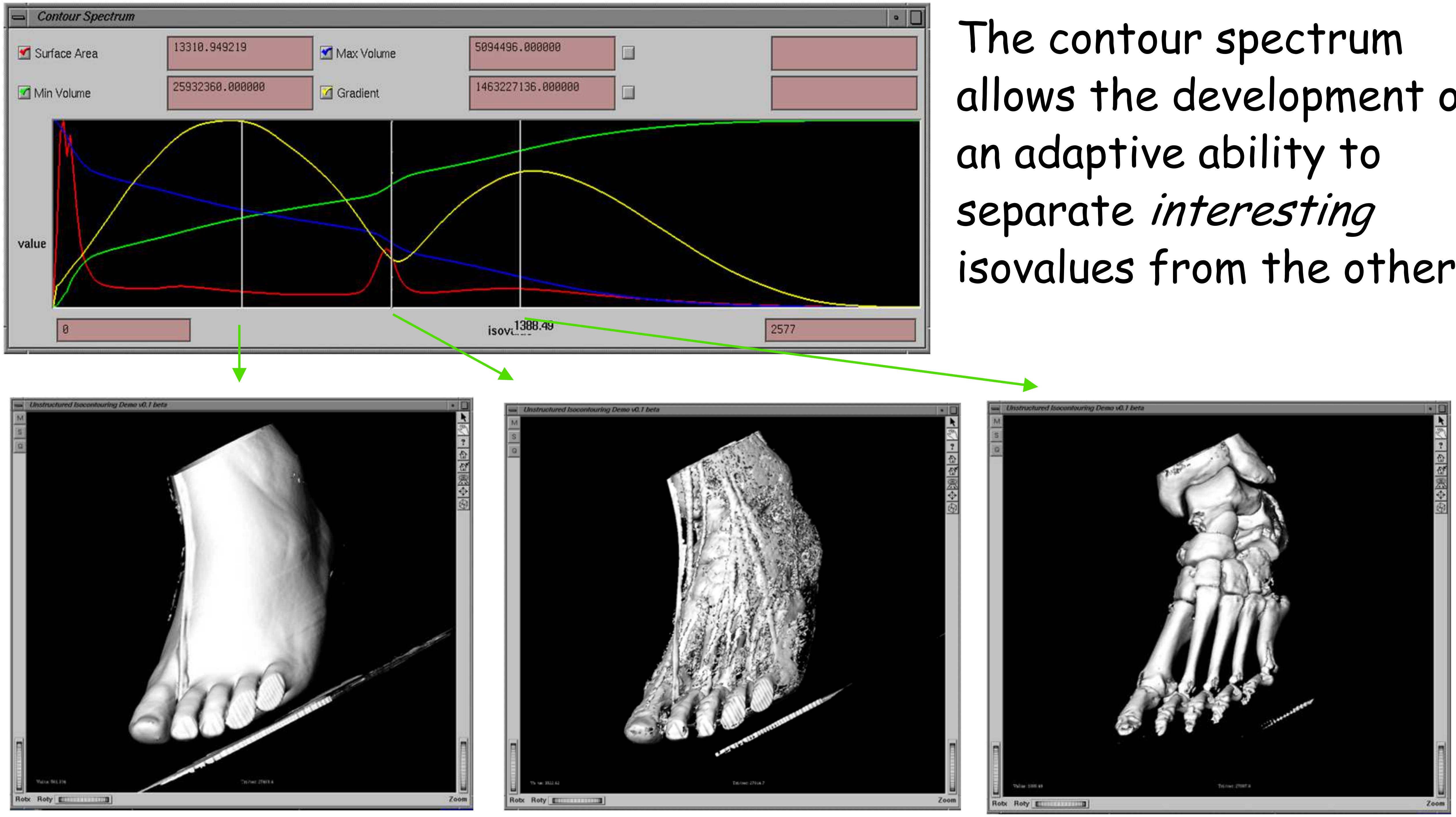
Value: 0.905062 Tri/sec: 37600

✓ Length	689.871582	✓ Gradient	13.659443
✓ Min Area	30760.617188	☐	
✓ Max Area	32687.382813	☐	

-14.4249 iso 0.905062 13.6592

The Contour Spectrum

(colored lines correspond to different isosurface metrics)



Surface Area 13310.949219
Max Volume 5094496.000000
Min Volume 25932360.000000
Gradient 1463227136.000000
0
isovalue 1388.49
2577

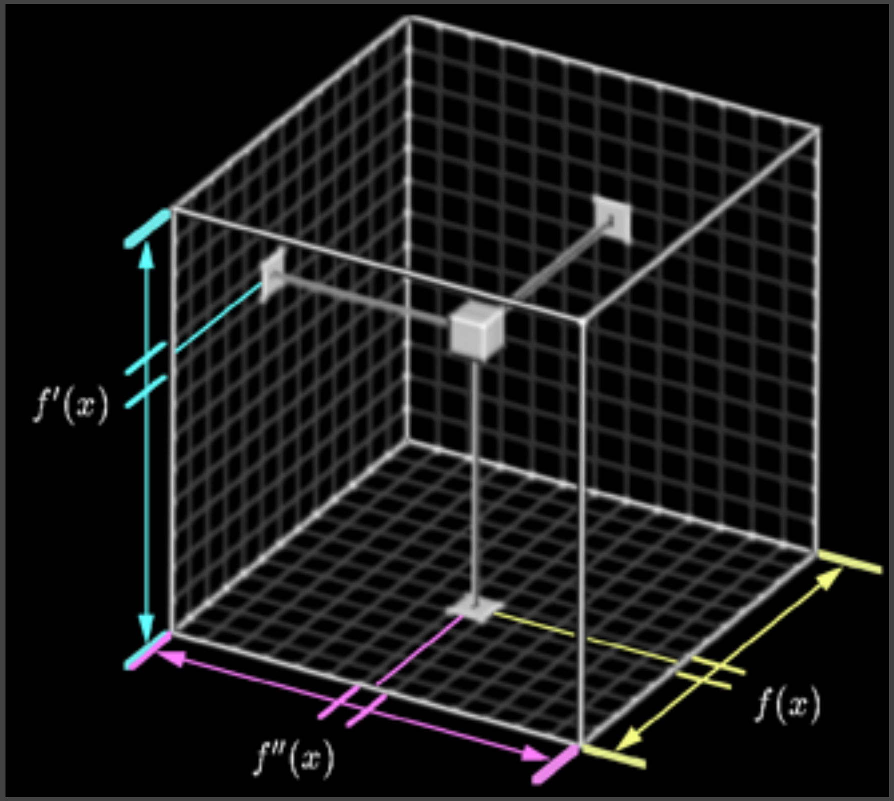
The contour spectrum allows the development of an adaptive ability to separate *interesting* isovalues from the others.

Use derivatives

Reasoning:

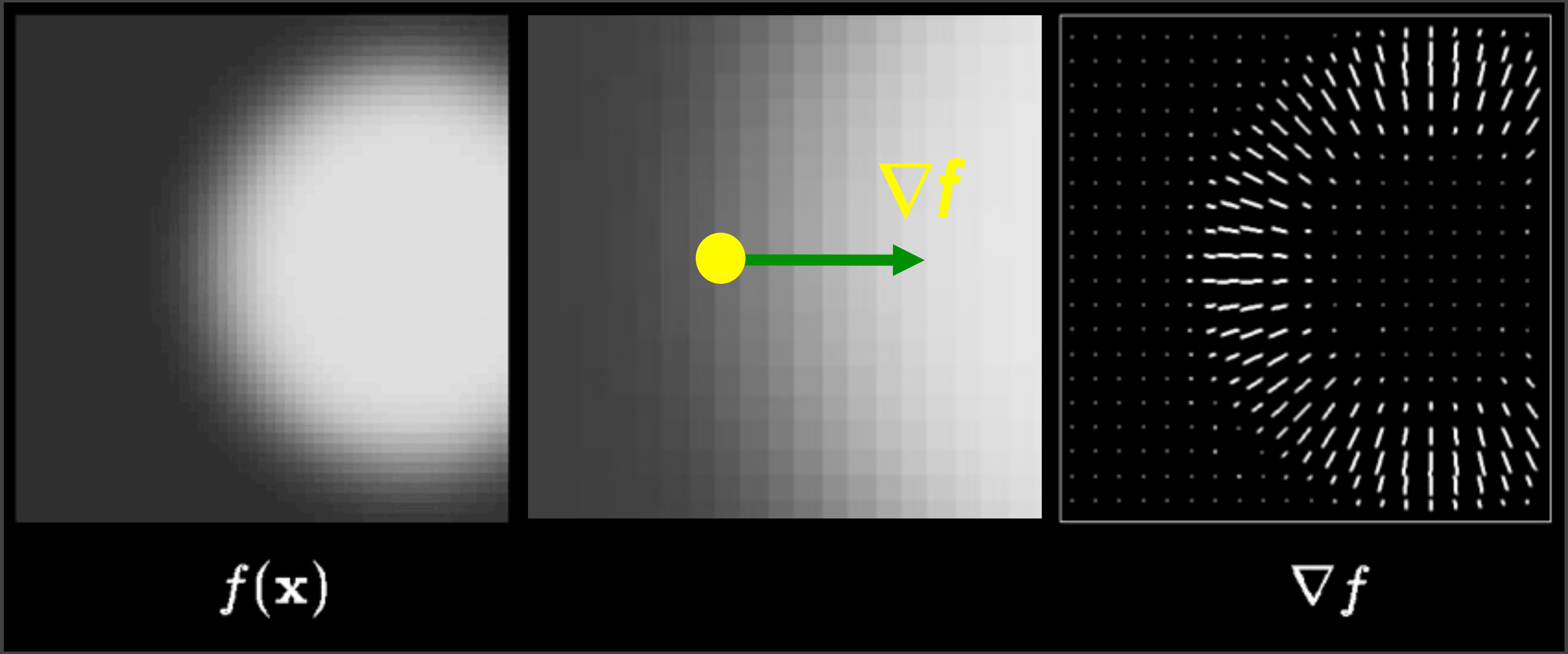
- TFs are volume-position invariant
- Histograms "project out" position
- Interested in boundaries between materials
- Boundaries characterized by derivatives

  Make 3D histograms of value, 1st, 2nd deriv.



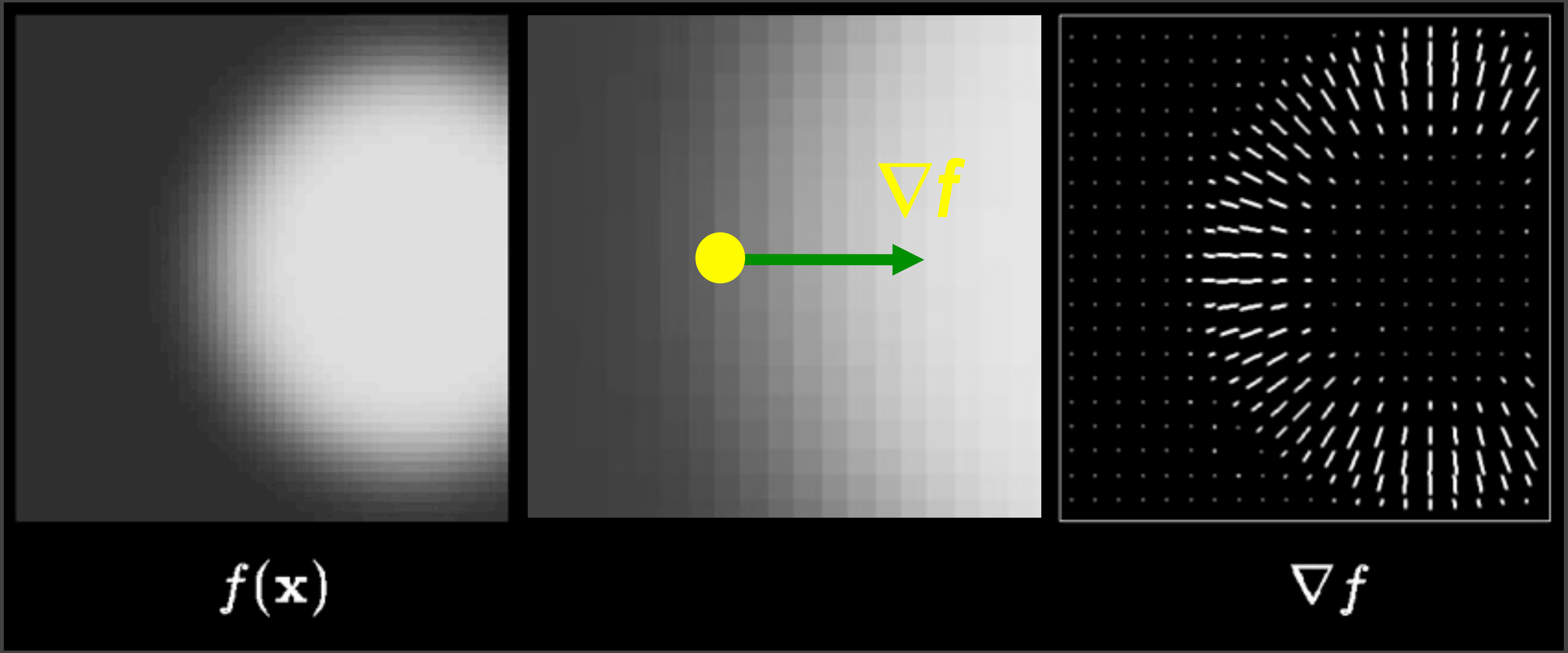
By (1) inspecting and (2) algorithmically analyzing histogram volume, we can create transfer functions

Gradient



Gradient

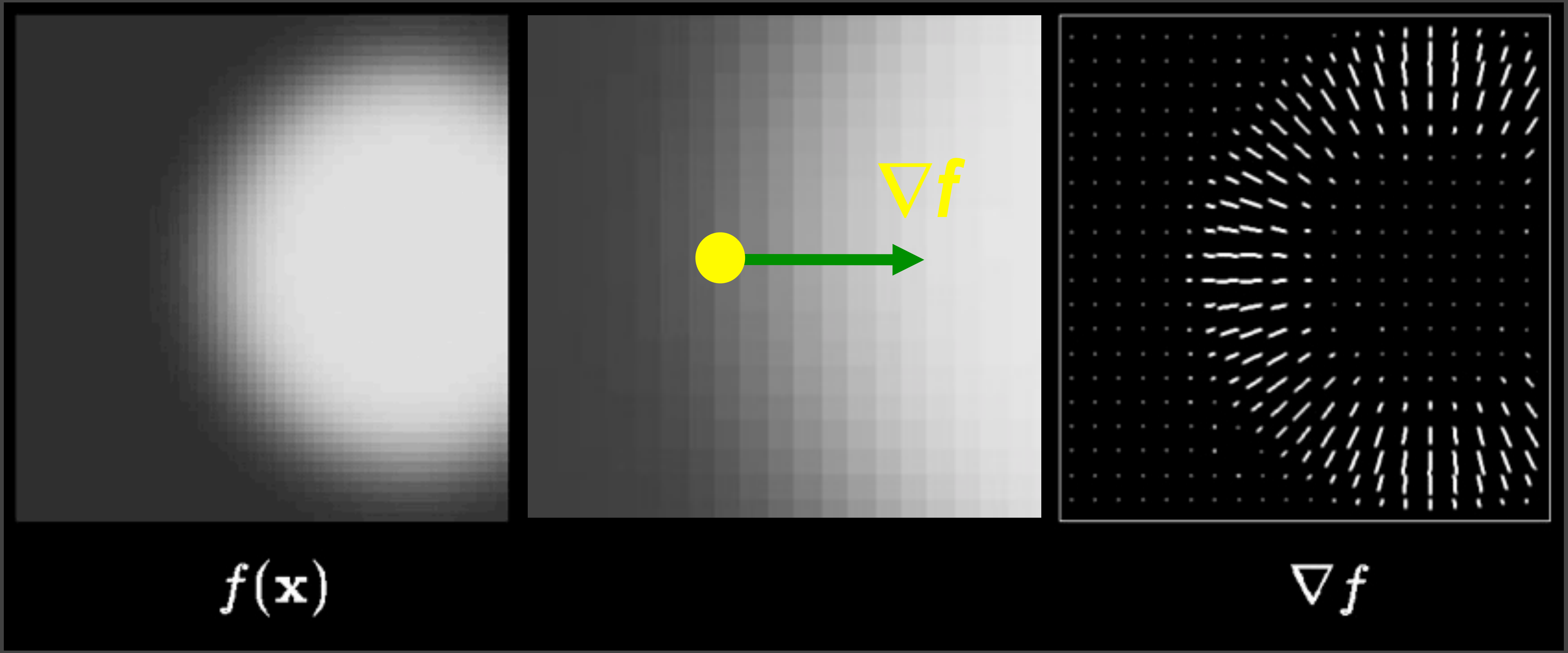
$\nabla f = (dx, dy, dz)$



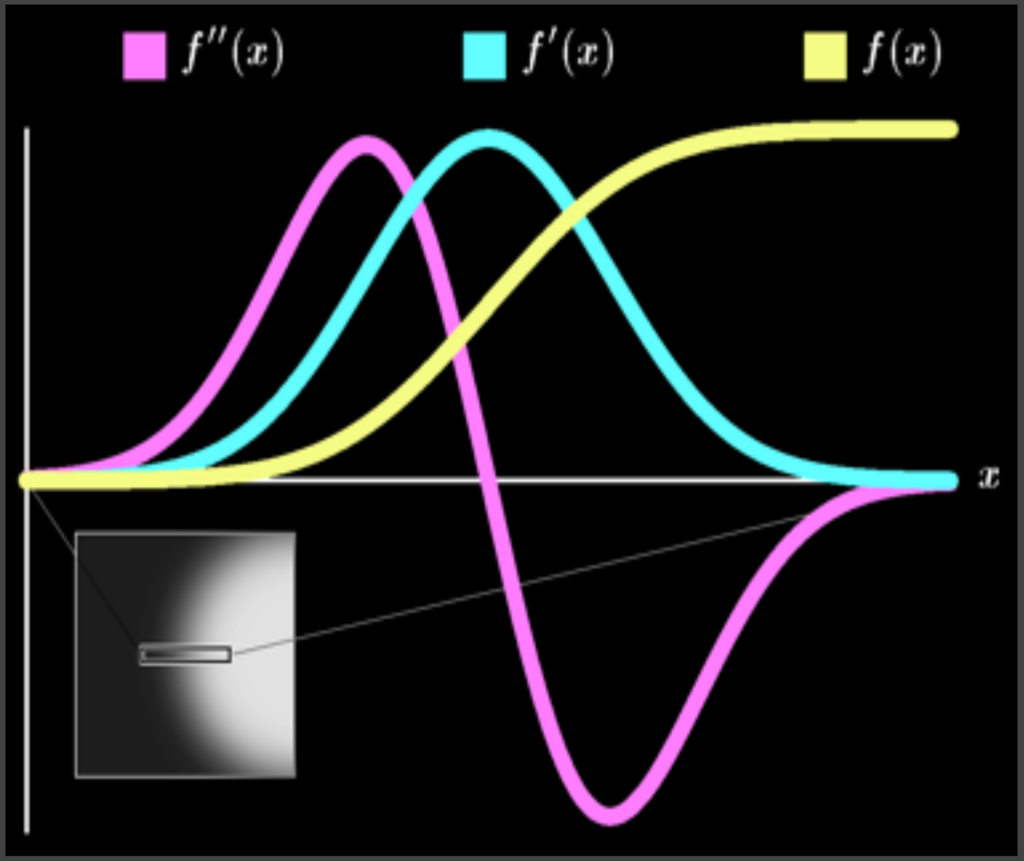
Gradient

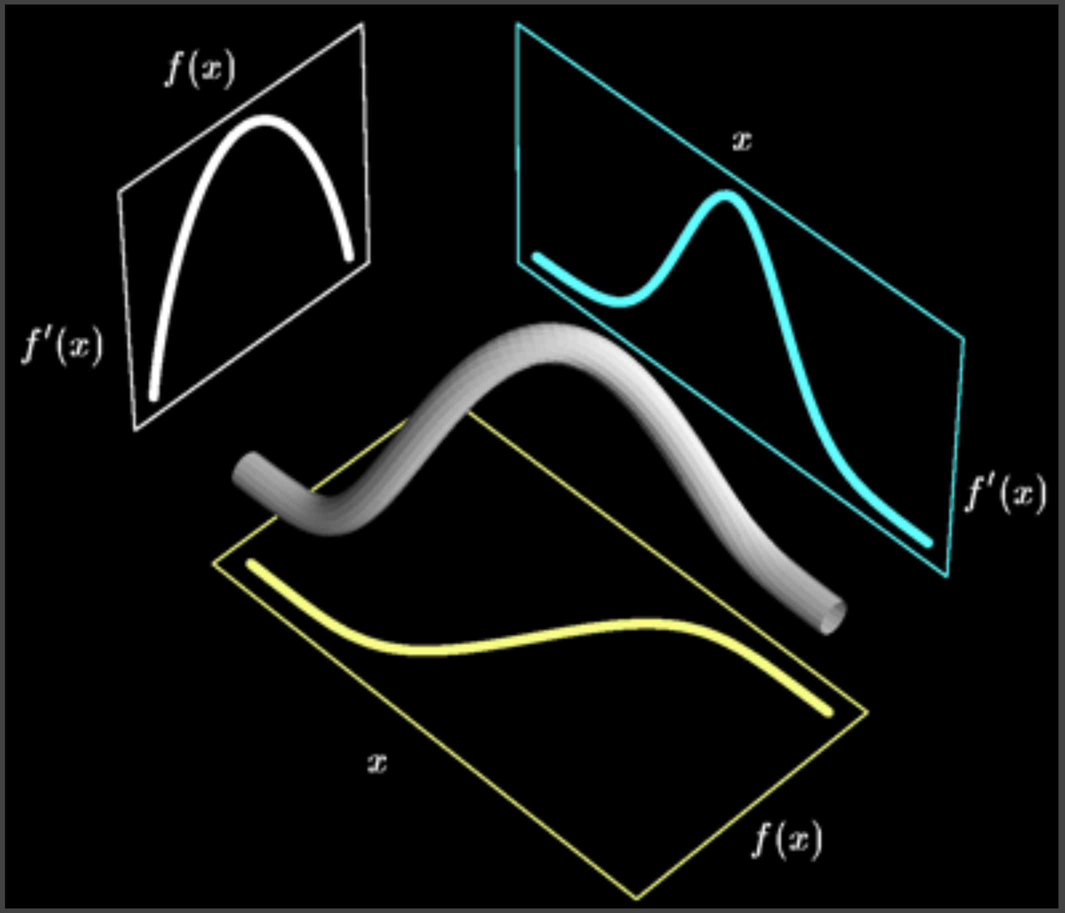
$$\nabla f = (dx, dy, dz)$$
$$= \big((f(1,0,0) - f(-1,0,0))/2,$$
$$(f(0,1,0) - f(0,-1,0))/2,$$
$$(f(0,0,1) - f(0,0,-1))/2 \big)$$

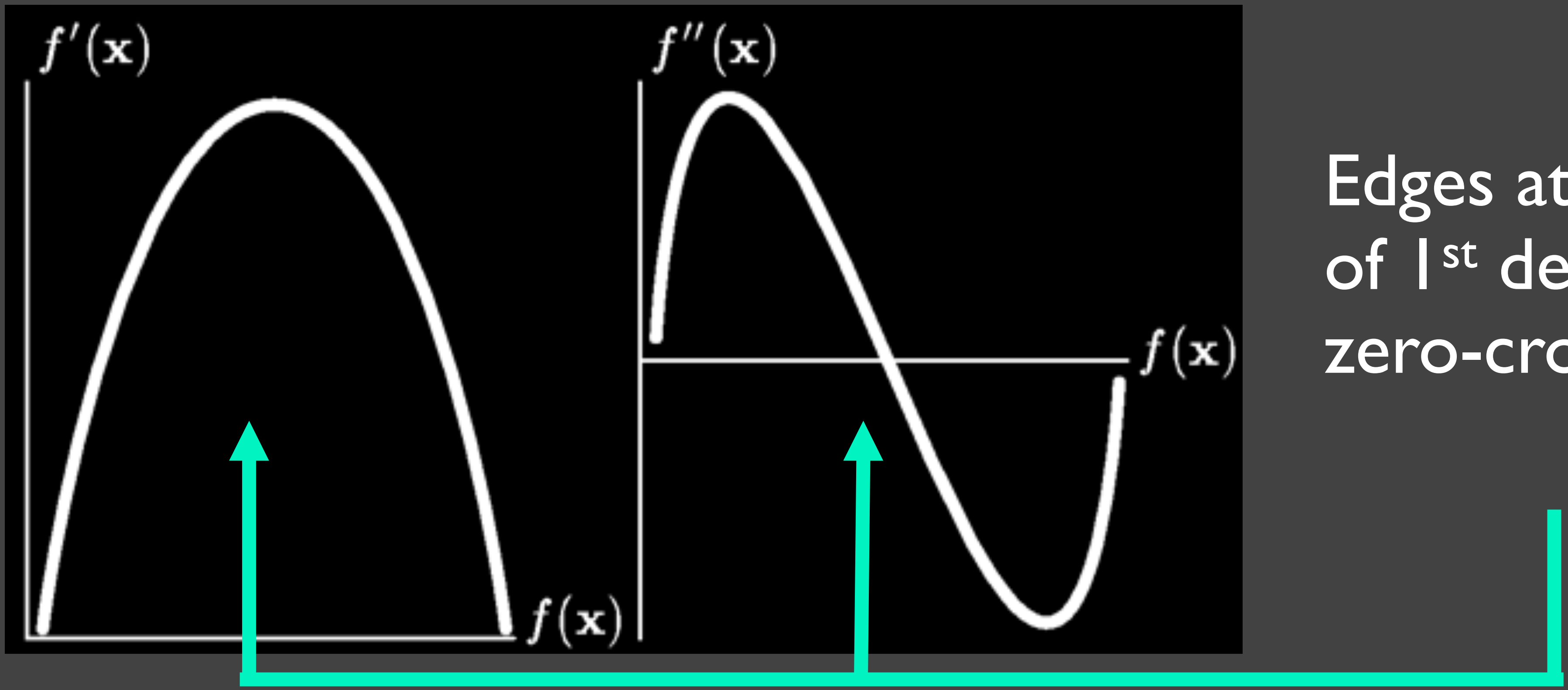
- Approximates "surface normal" (of isosurface)



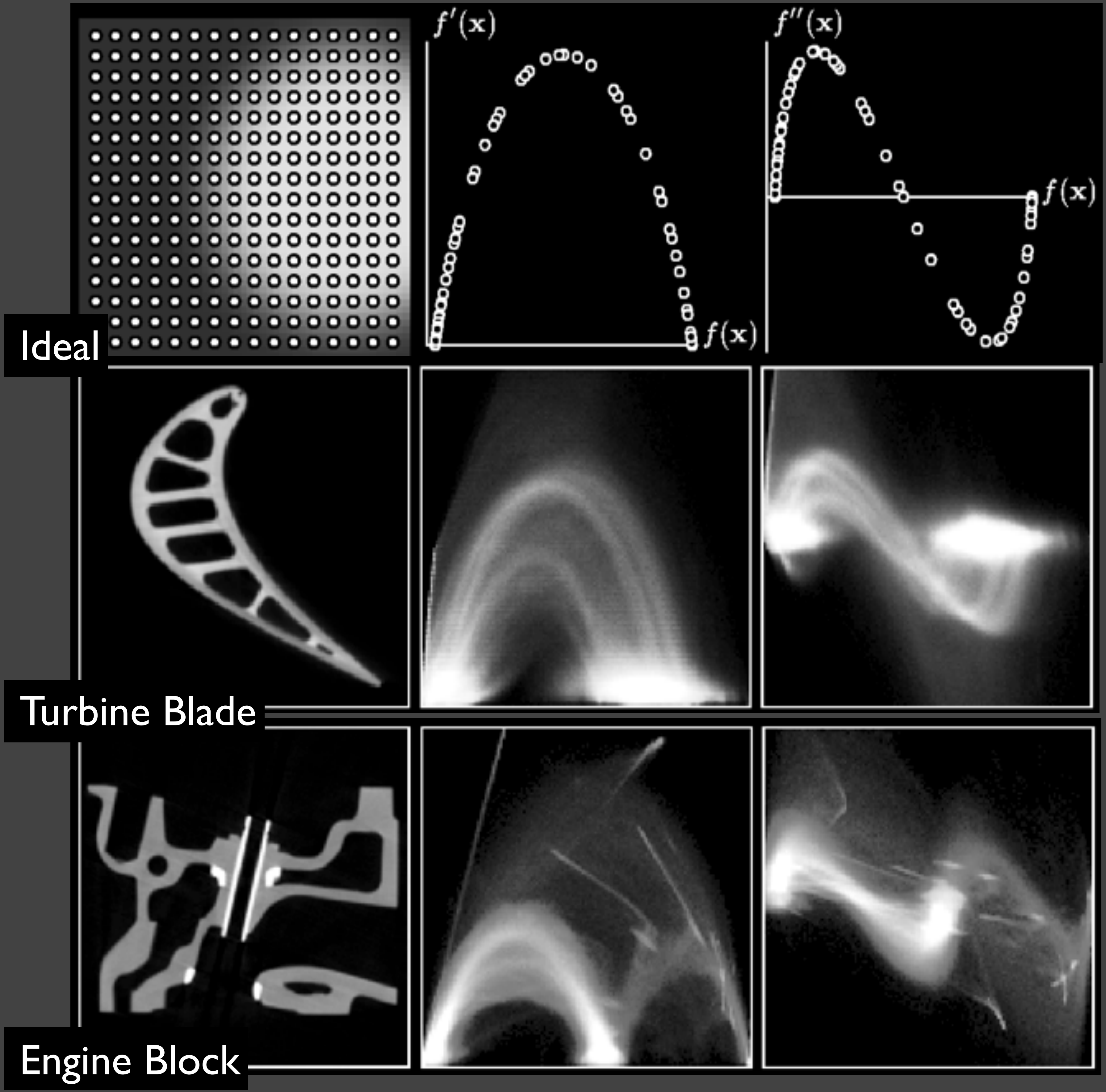
Derivative relationships







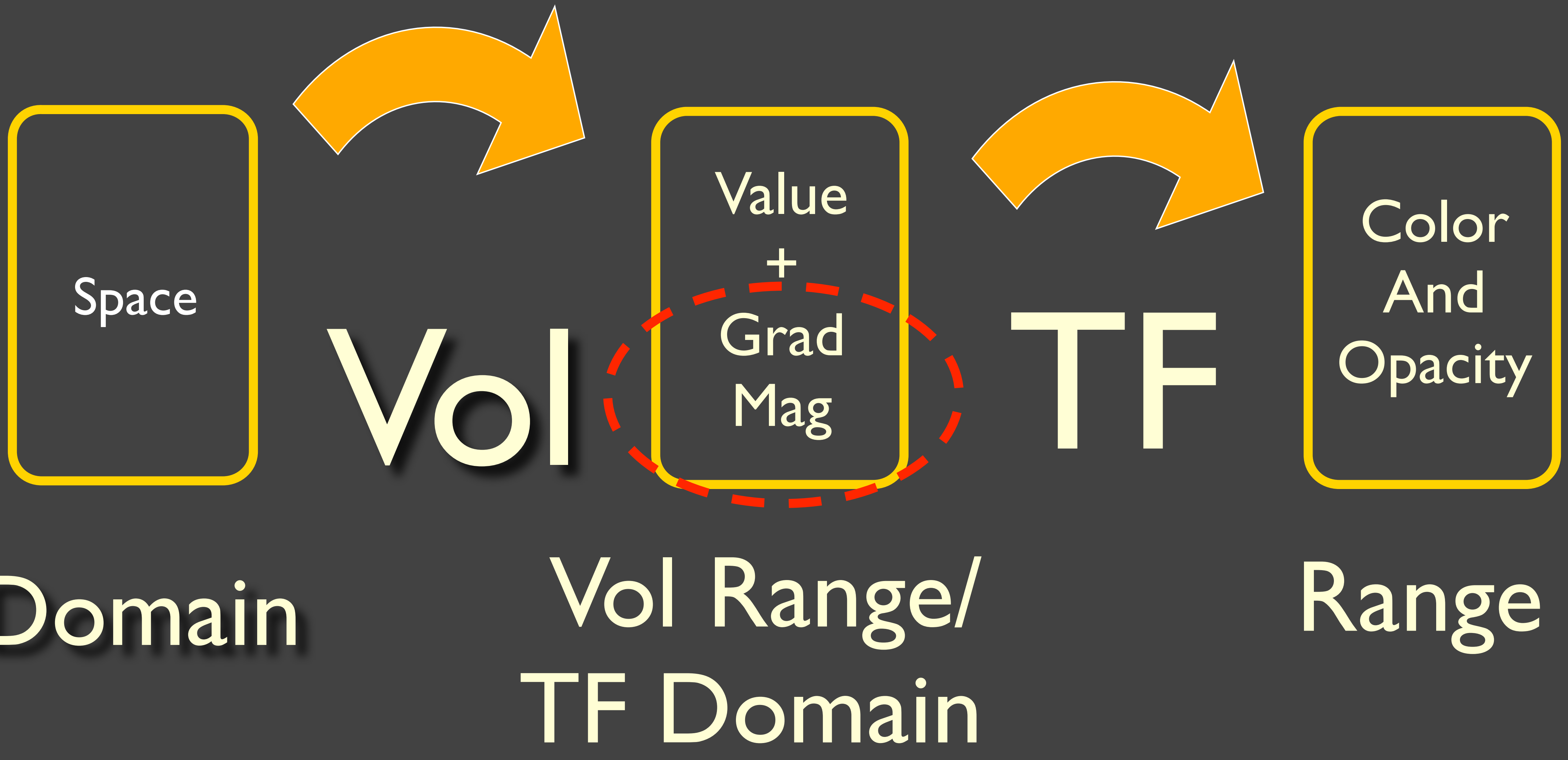
Edges at maximum of 1st derivative or zero-crossing of 2nd



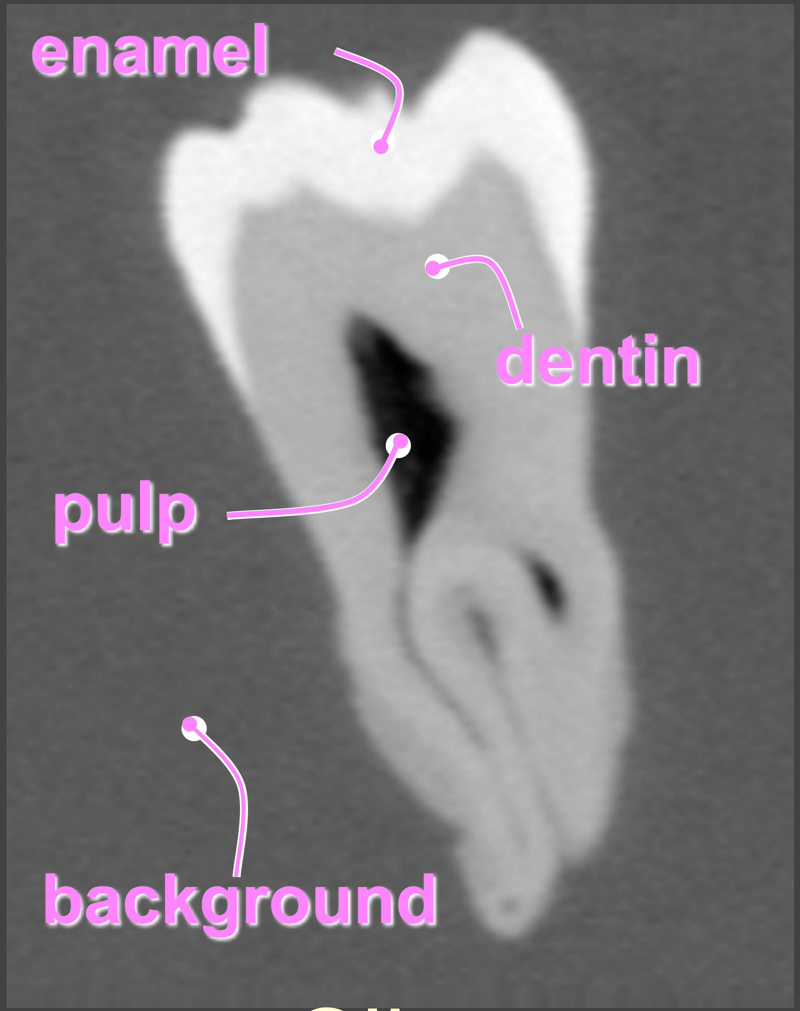
Project histogram volume to 2D scatterplots

- Visual summary
- Interpreted for TF guidance
- No reliance on boundary model at this stage

Basic Transfer Functions:

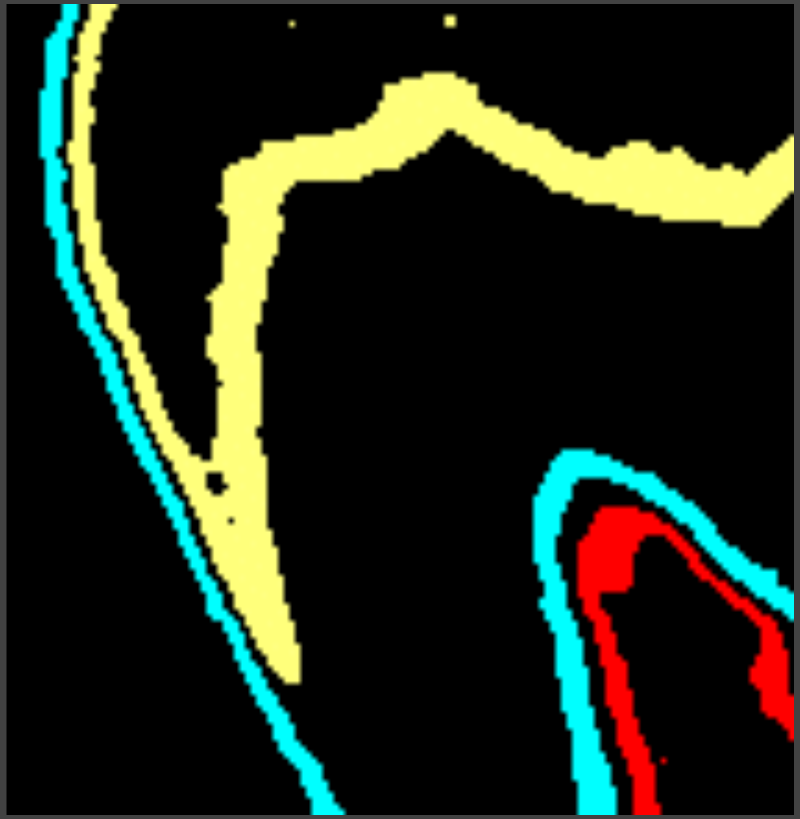


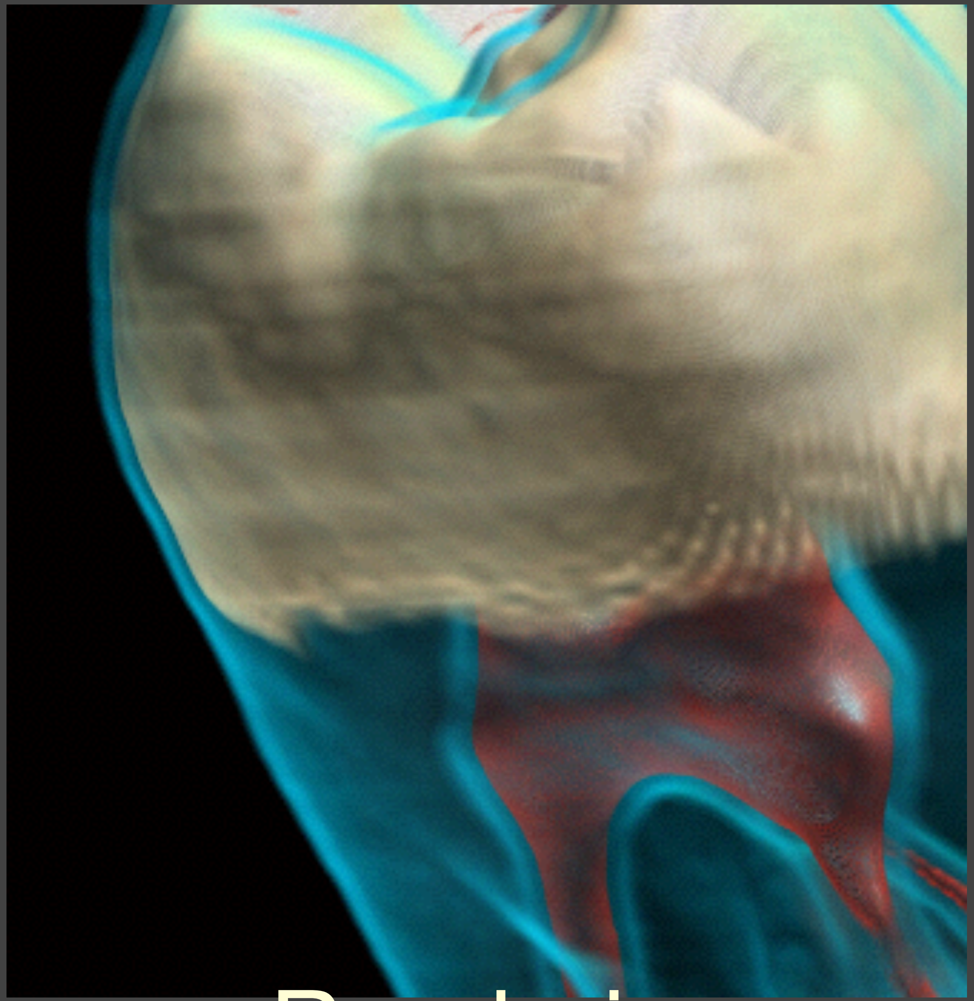
1D TFs: limitation



Slice

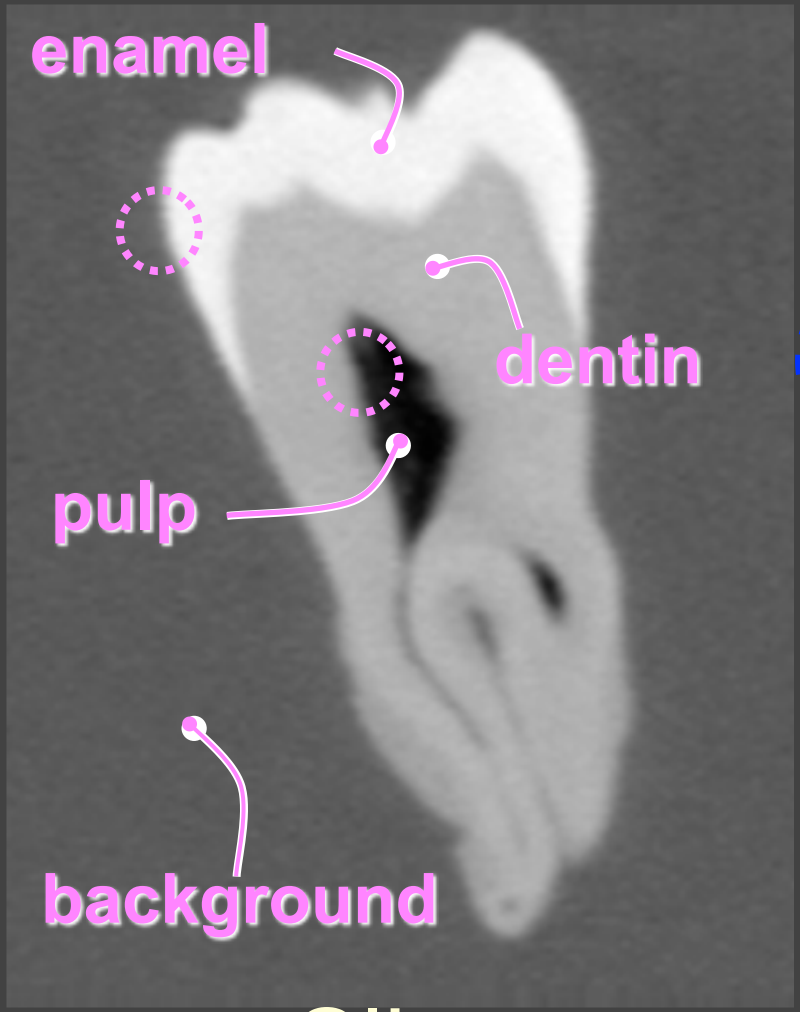
RGB(*f*)


1D TF output


Rendering

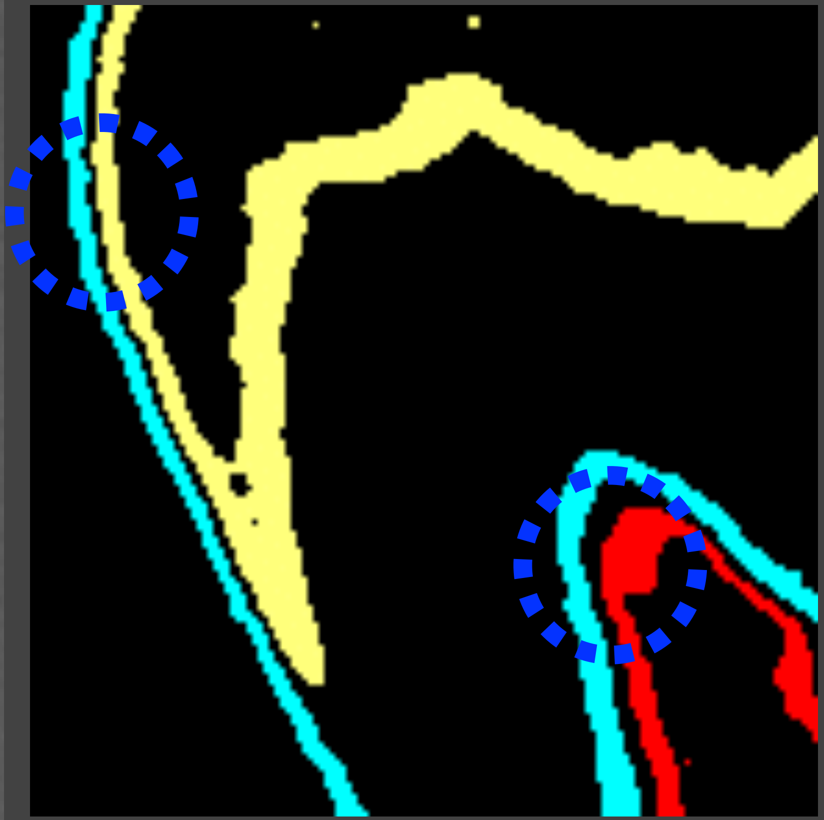
1D transfer functions can not accurately capture all material boundaries

1D TFs: limitation

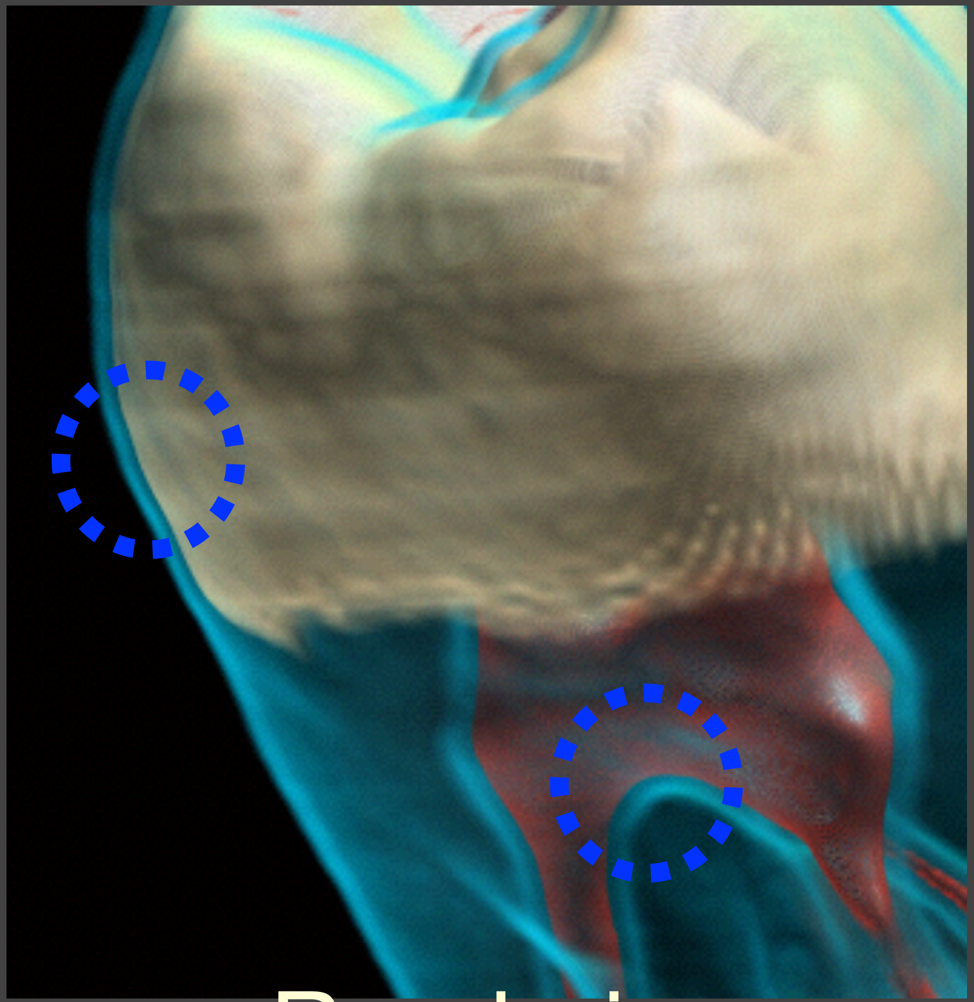


Slice

RGB(f)



1D TF output

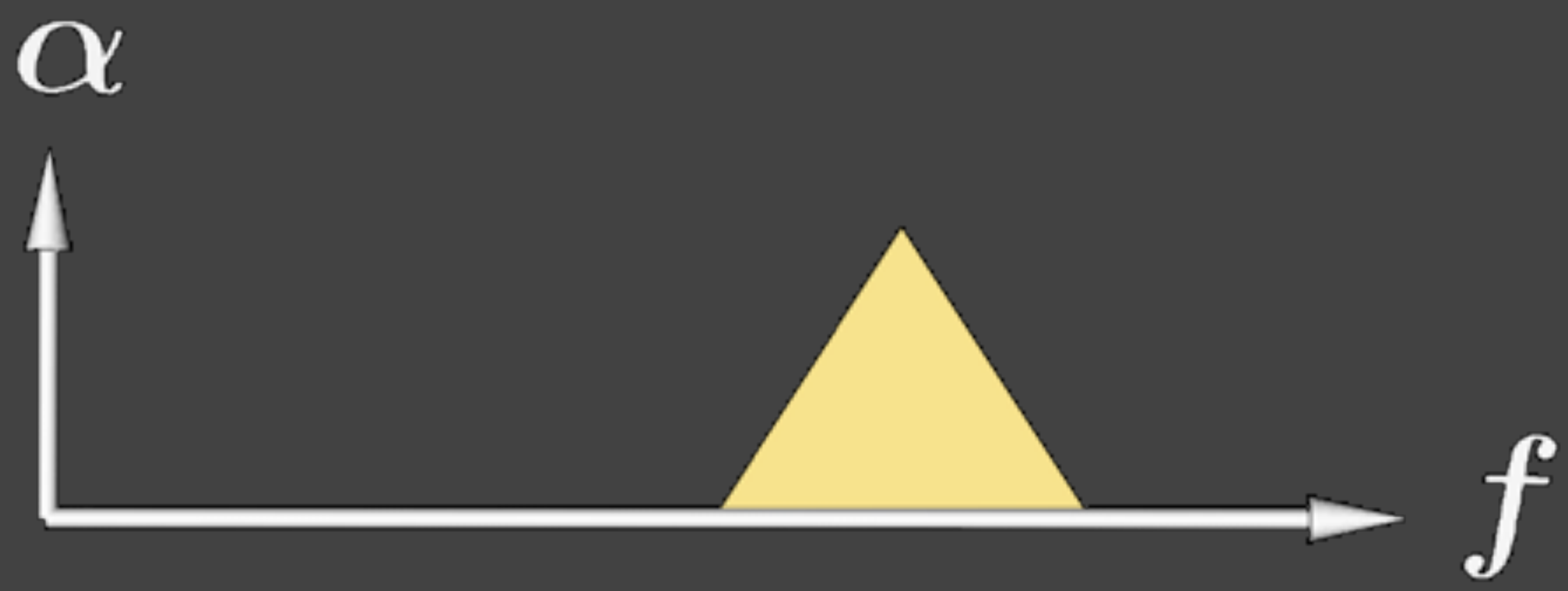


Rendering

1D transfer functions can not accurately capture all material boundaries

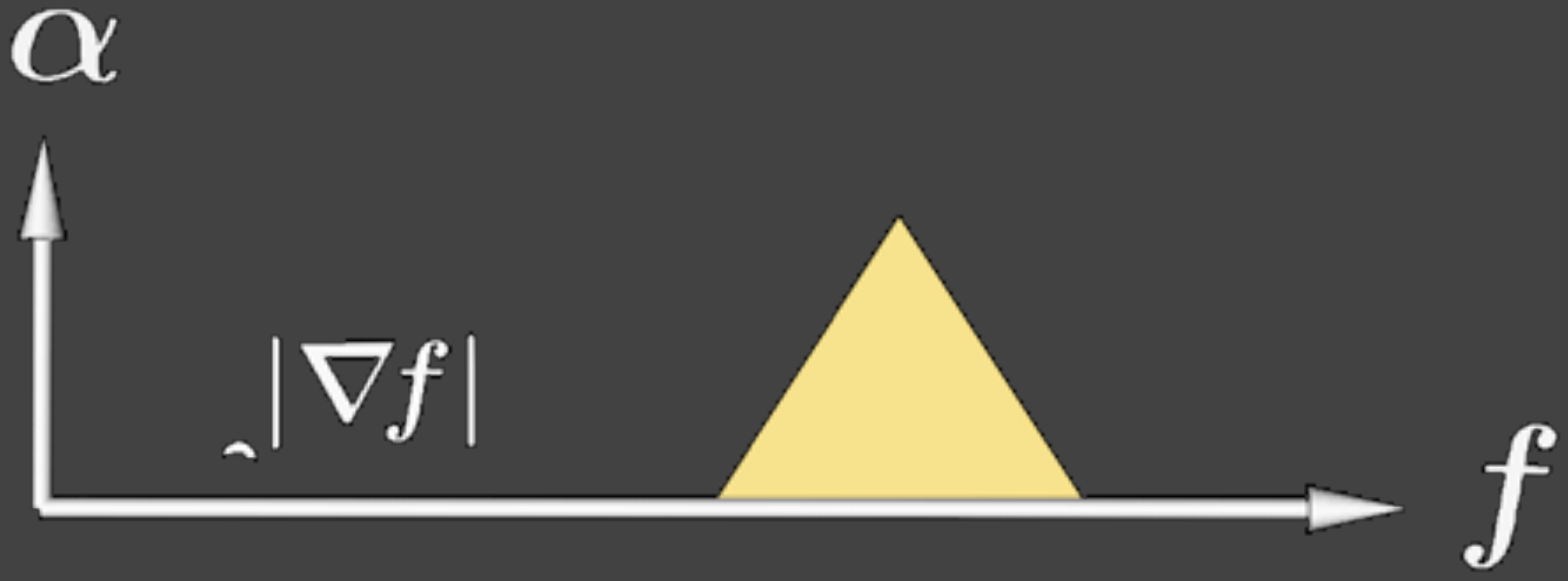
1D → 2D Transfer Function

$$\left.\begin{matrix} \text{RGB}(f) \\ \alpha(f) \end{matrix}\right\}$$ Generalize…



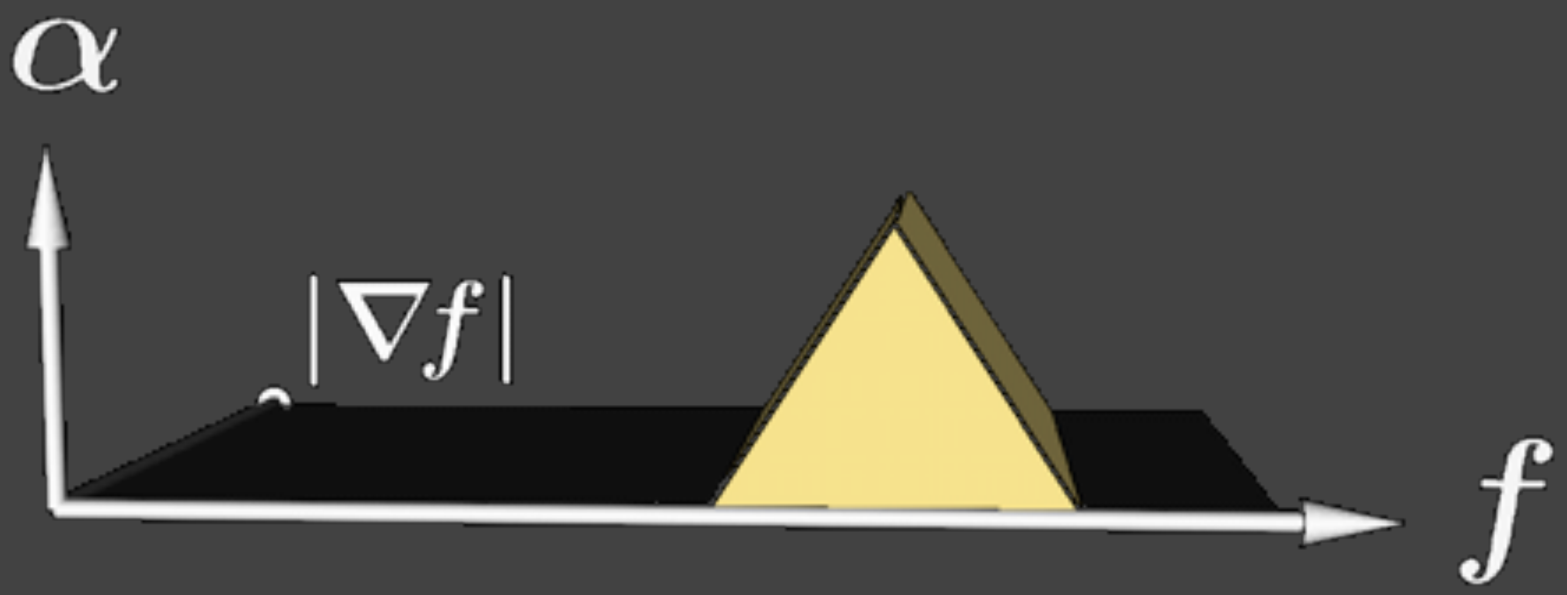
1D → 2D Transfer Function

$$\left.\begin{array}{r} RGB(f) \\ \alpha(f) \end{array}\right\}$$ Generalize…



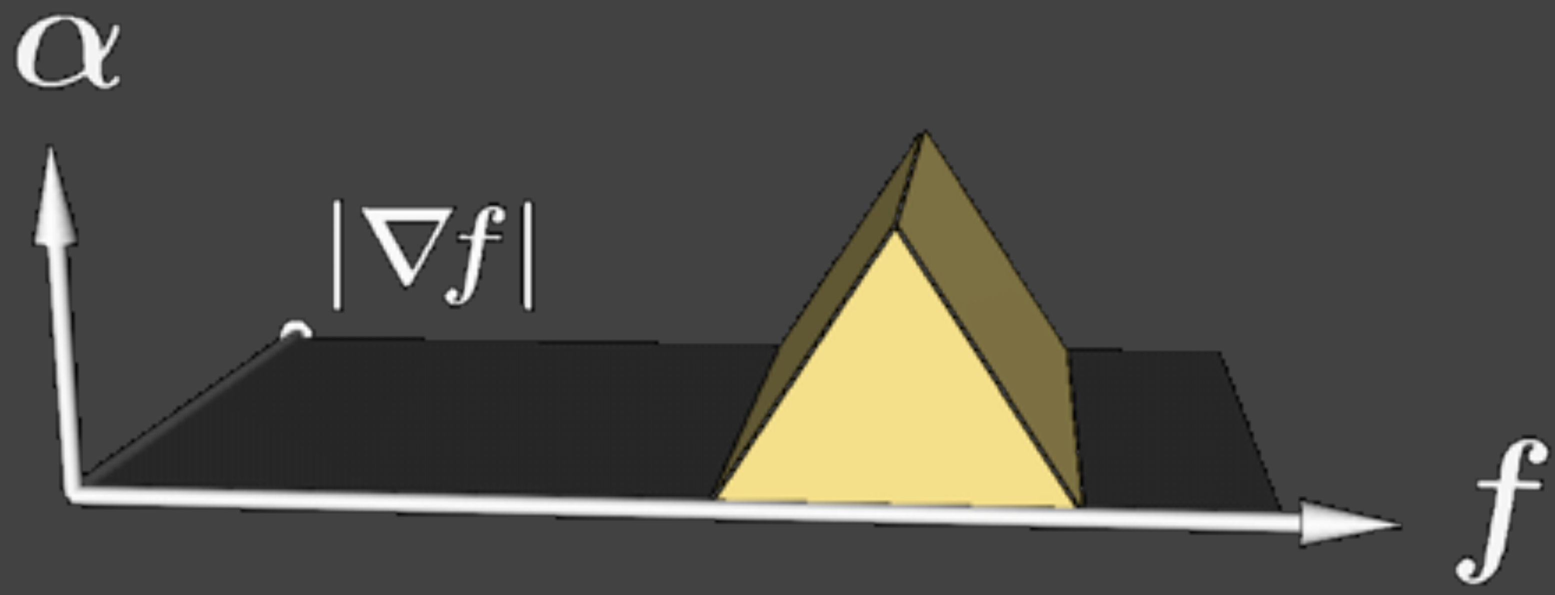
1D → 2D Transfer Function

$$\left.\begin{array}{r} \mathrm{RGB}(f) \\ \alpha(f) \end{array}\right\}$$ Generalize…



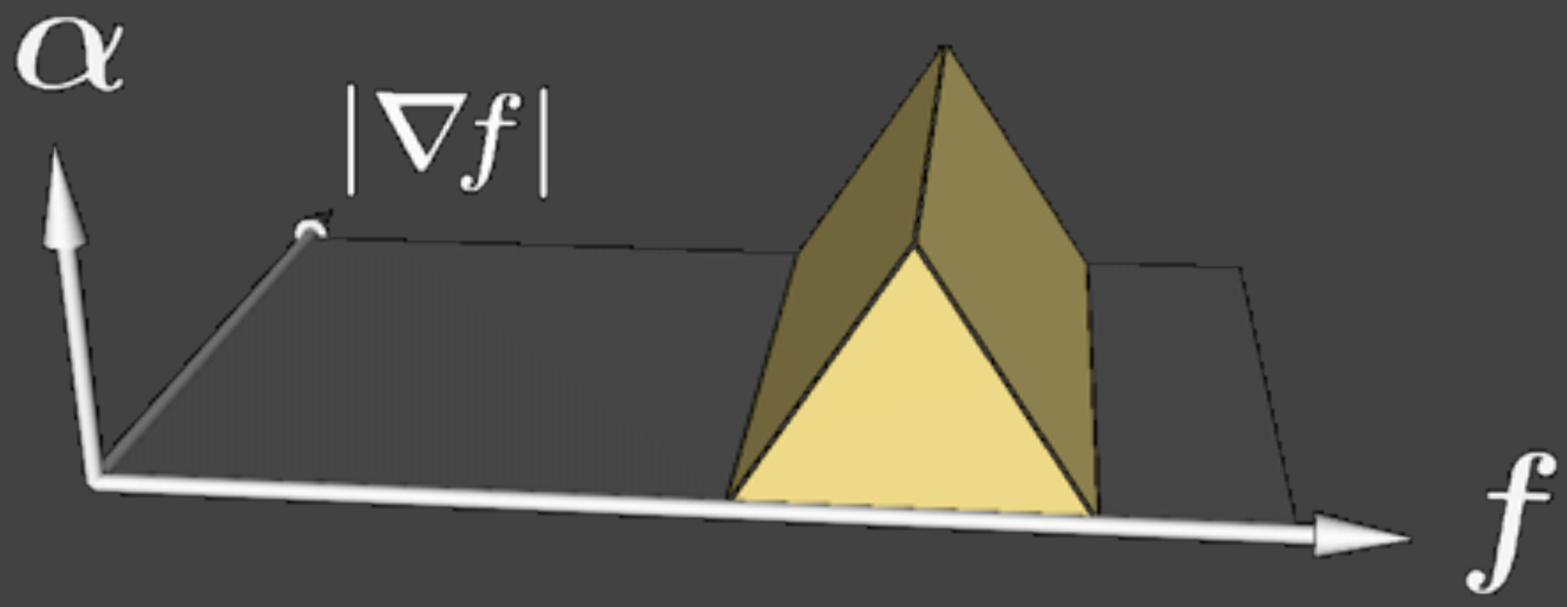
1D → 2D Transfer Function

$$\left.\begin{array}{r}\mathrm{RGB}(f)\\ \alpha(f)\end{array}\right\}$$ Generalize…



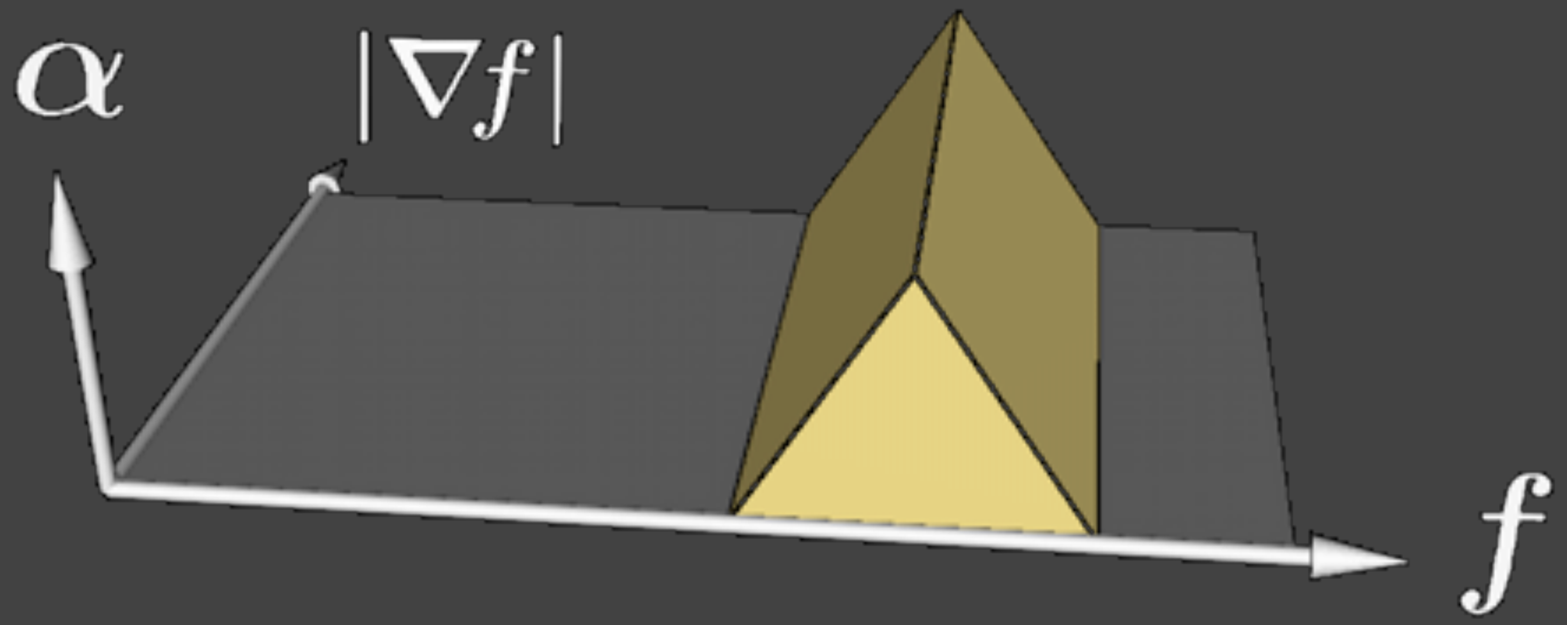
1D → 2D Transfer Function

$$\left.\begin{array}{r}\mathrm{RGB}(f)\\ \alpha(f)\end{array}\right\}$$ Generalize…



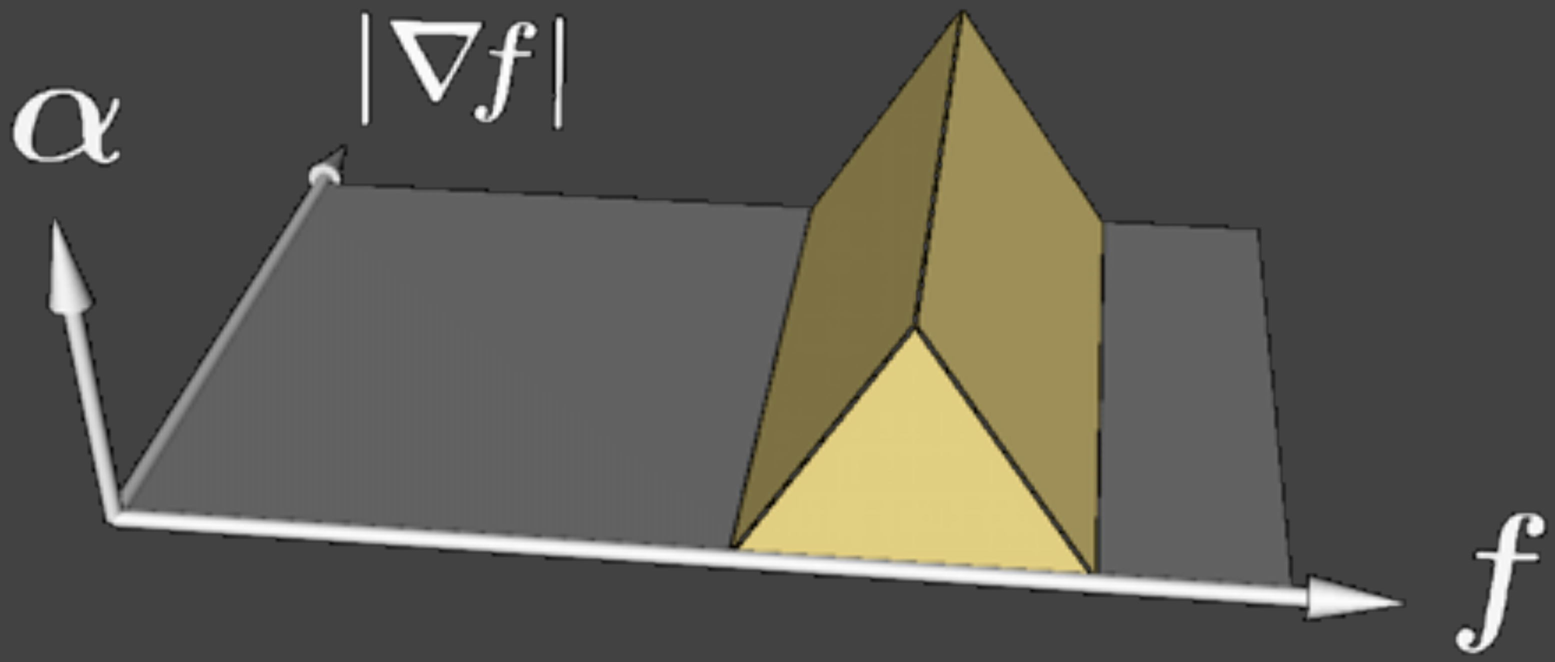
1D → 2D Transfer Function

$$\left.\begin{array}{r}\mathrm{RGB}(f)\\ \alpha(f)\end{array}\right\}$$ Generalize…



2D Transfer Function

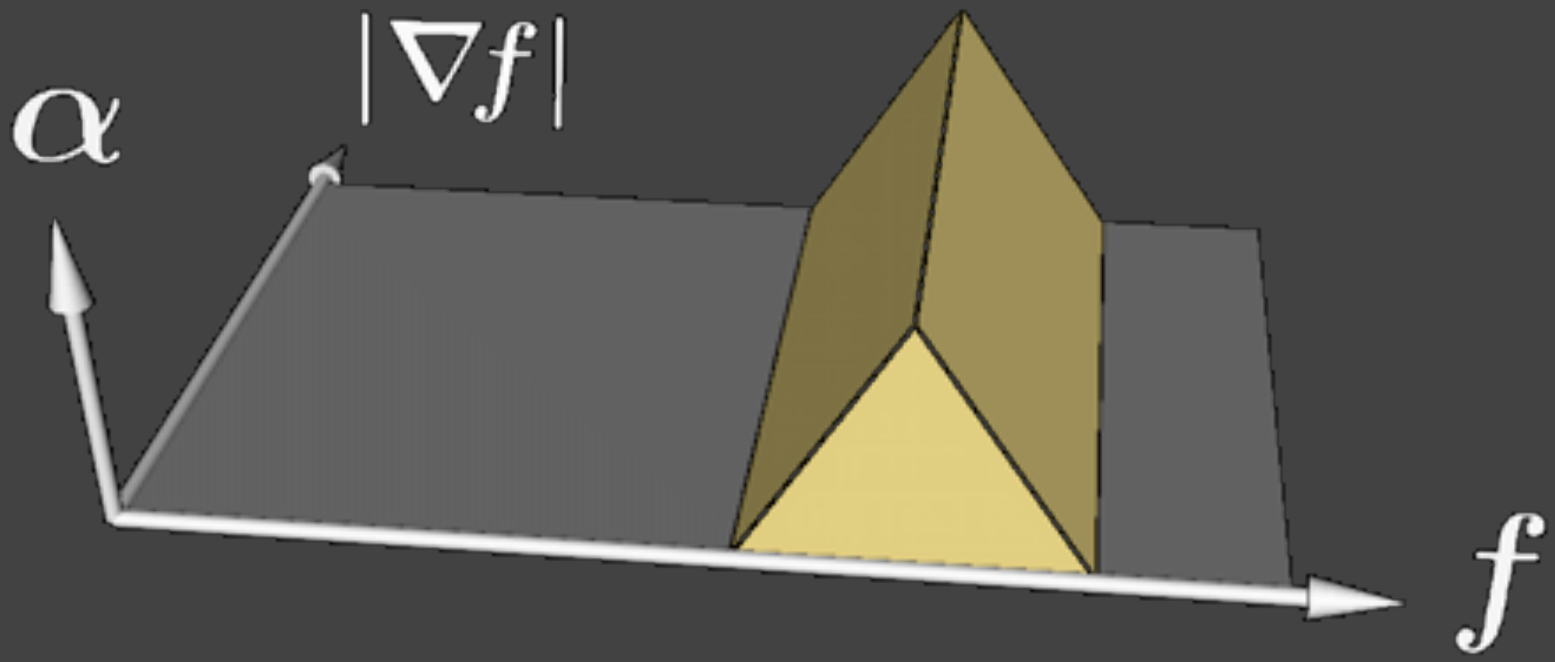
$$\left.\begin{array}{r}\mathrm{RGB}(f)\\ \alpha(f)\end{array}\right\}$$ Generalize…



$$\rightarrow \begin{array}{r}\mathrm{RGB}(f,|\nabla f|)\\ \alpha(f,|\nabla f|)\end{array}$$

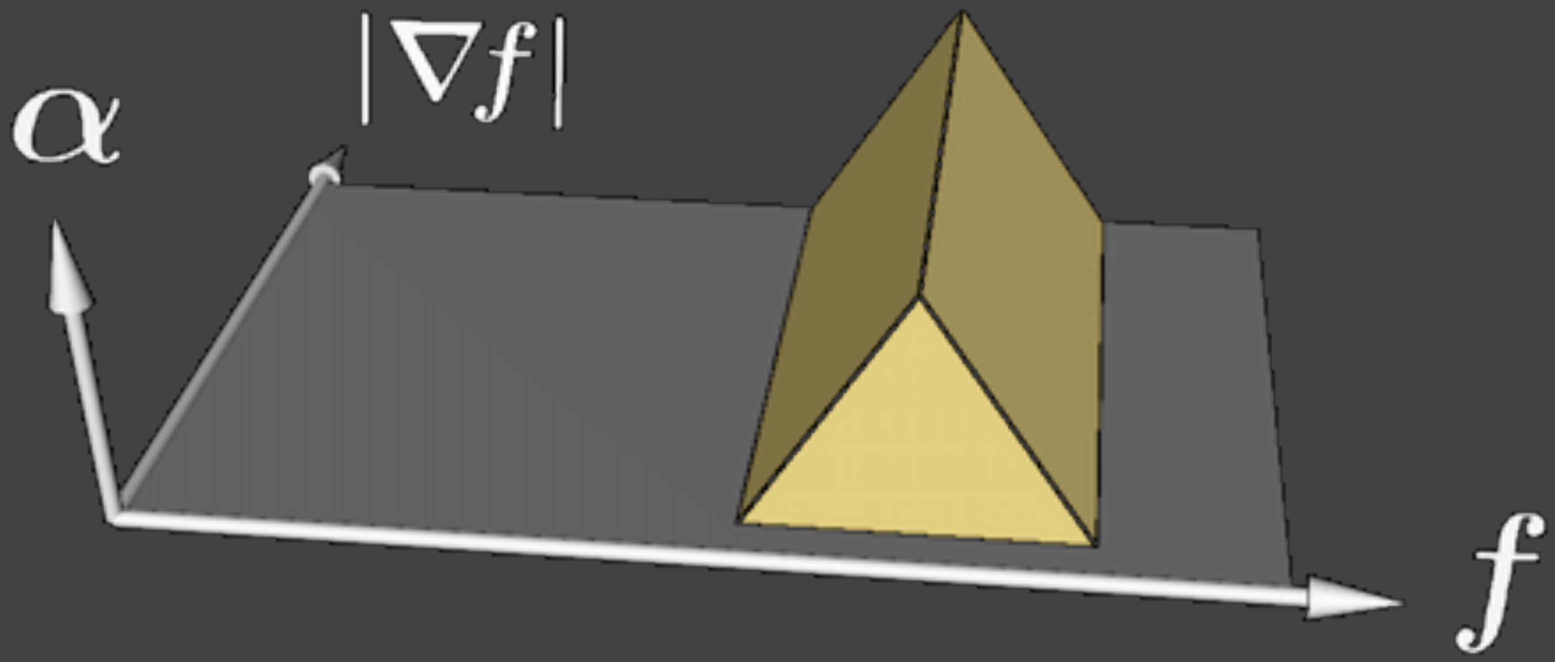
2D Transfer Function

$$\left.\begin{array}{r} \mathrm{RGB}(f, |\nabla f|) \\ \alpha(f, |\nabla f|) \end{array}\right\} \text{Modify...}$$



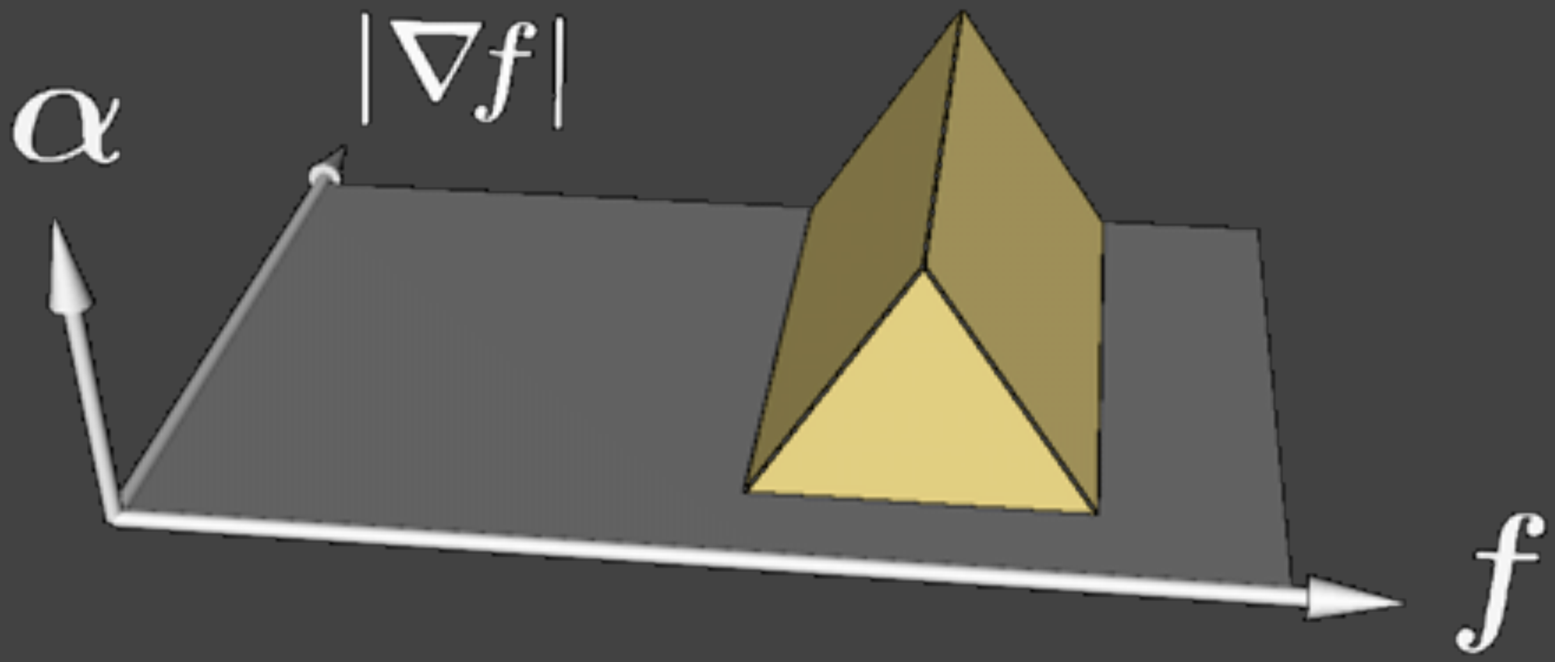
2D Transfer Function

$$\left.\begin{matrix}\mathrm{RGB}(f, |\nabla f|) \\ \alpha(f, |\nabla f|)\end{matrix}\right\} \text{Modify...}$$

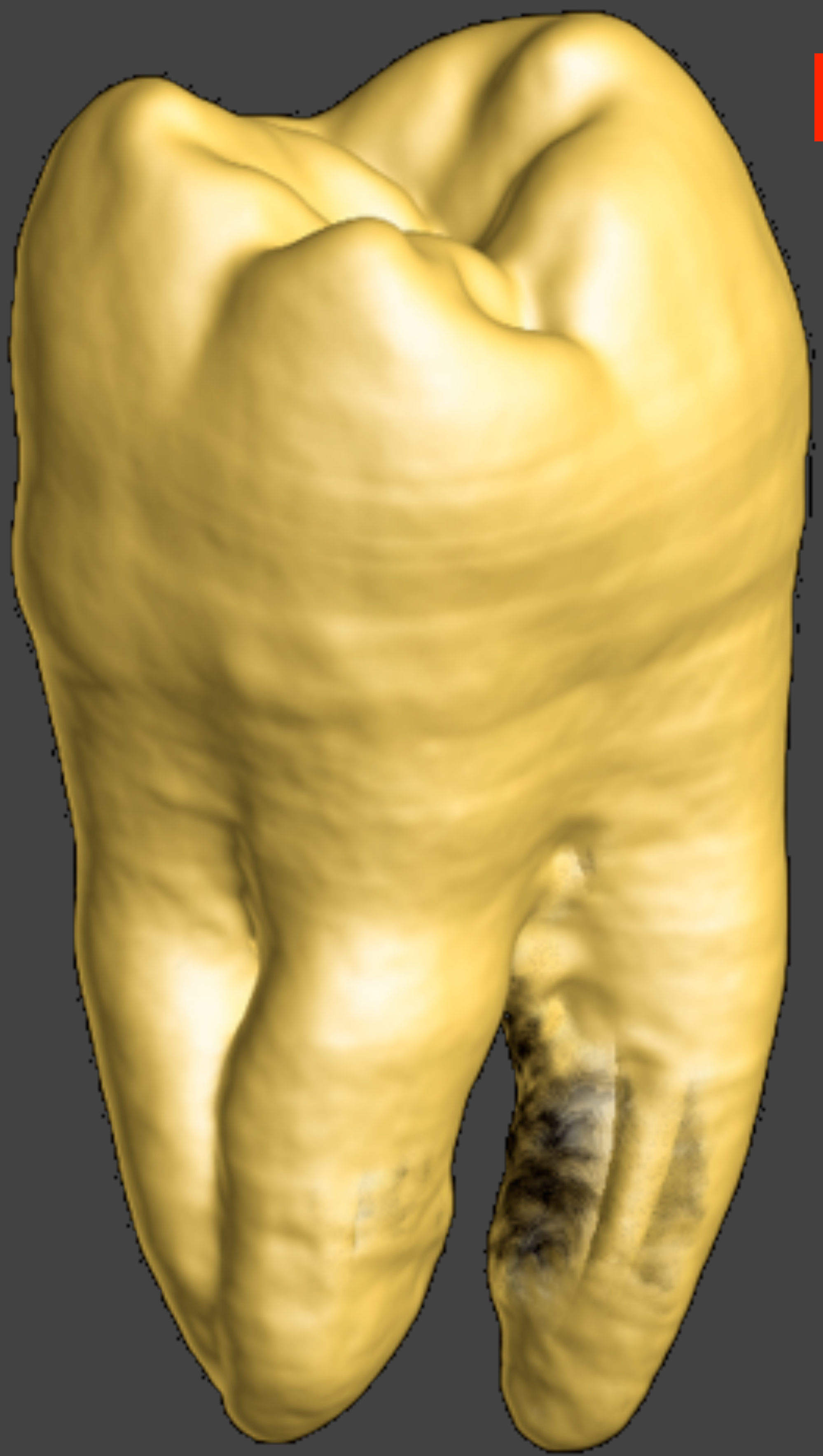


2D Transfer Function

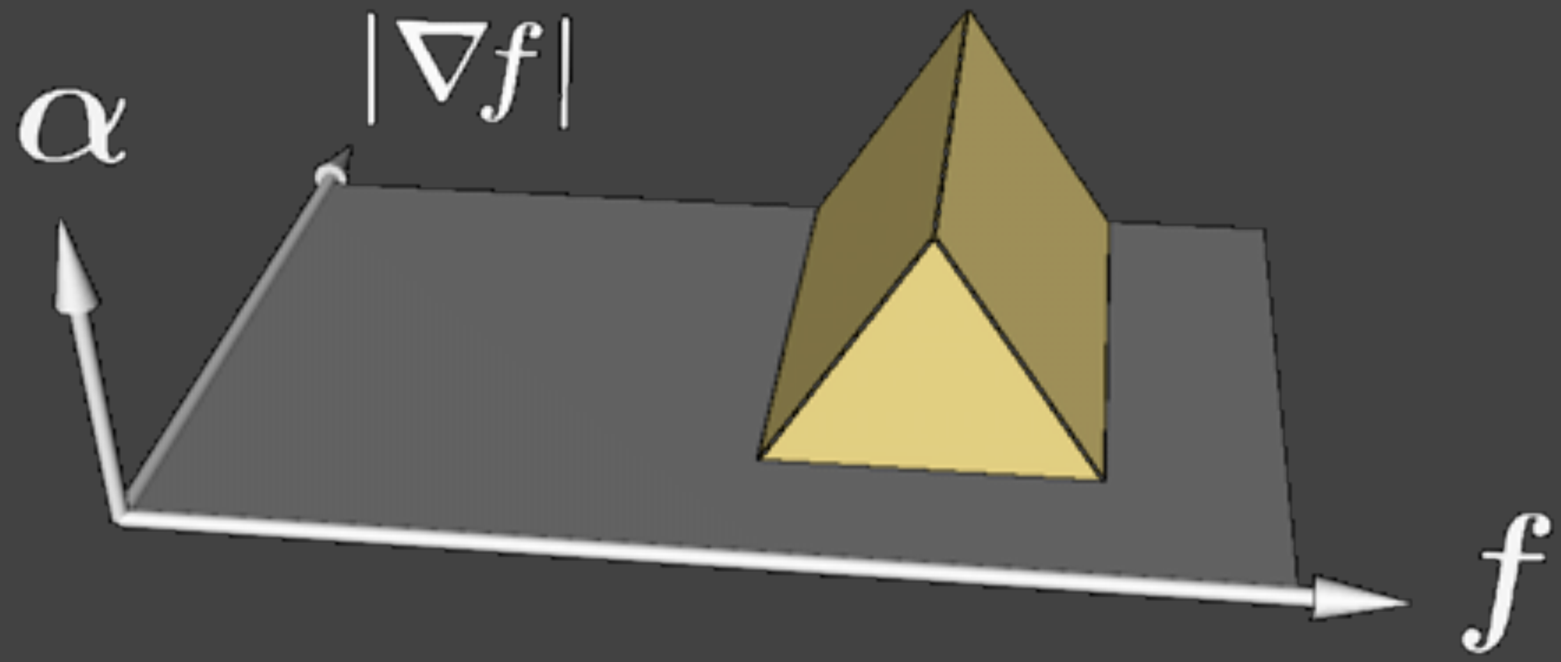
$$\left.\begin{array}{r}\mathrm{RGB}(f, |\nabla f|) \\ \alpha(f, |\nabla f|)\end{array}\right\} \text{Modify…}$$



2D Transfer Function

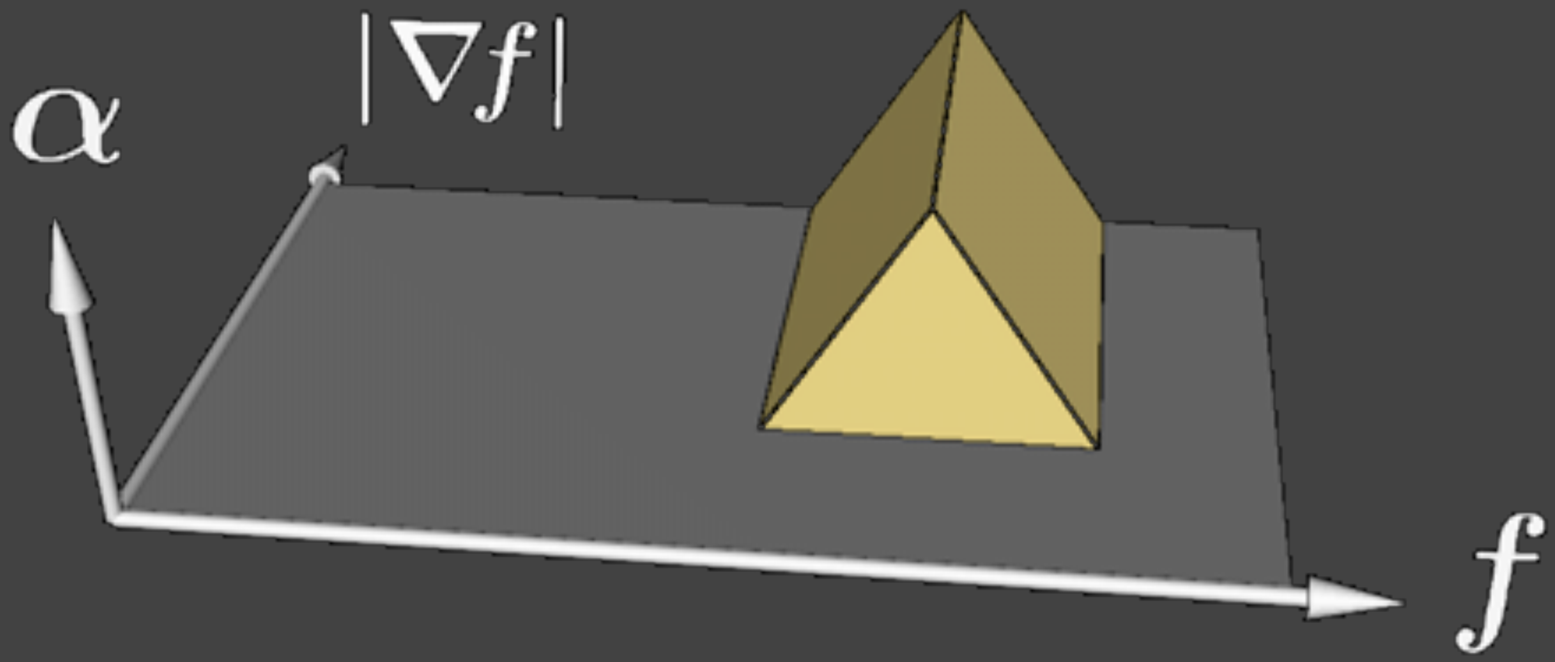


$$\left.\begin{array}{r}\mathrm{RGB}(f,|\nabla f|)\\ \alpha(f,|\nabla f|)\end{array}\right\}\text{Modify…}$$



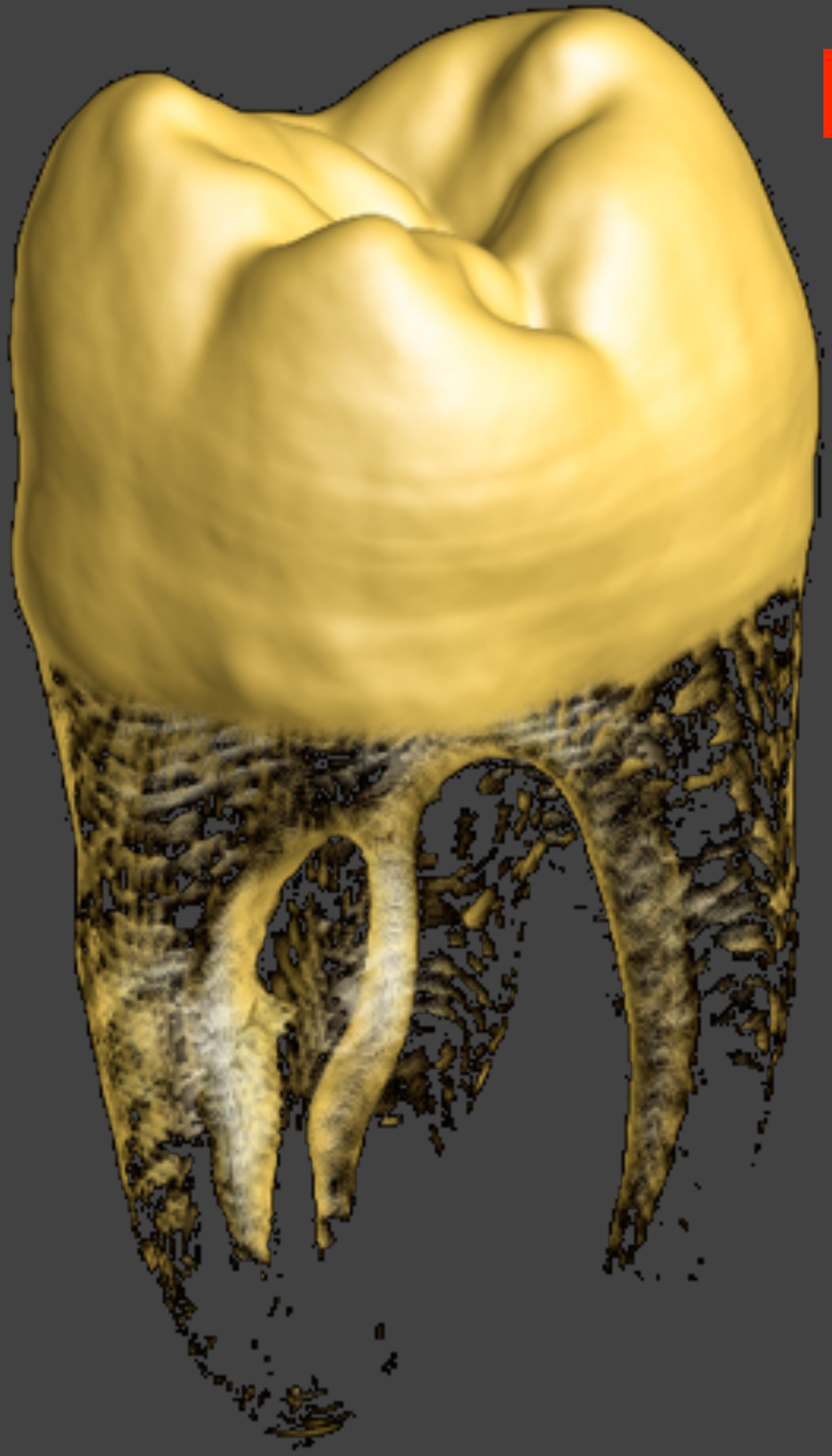
2D Transfer Function

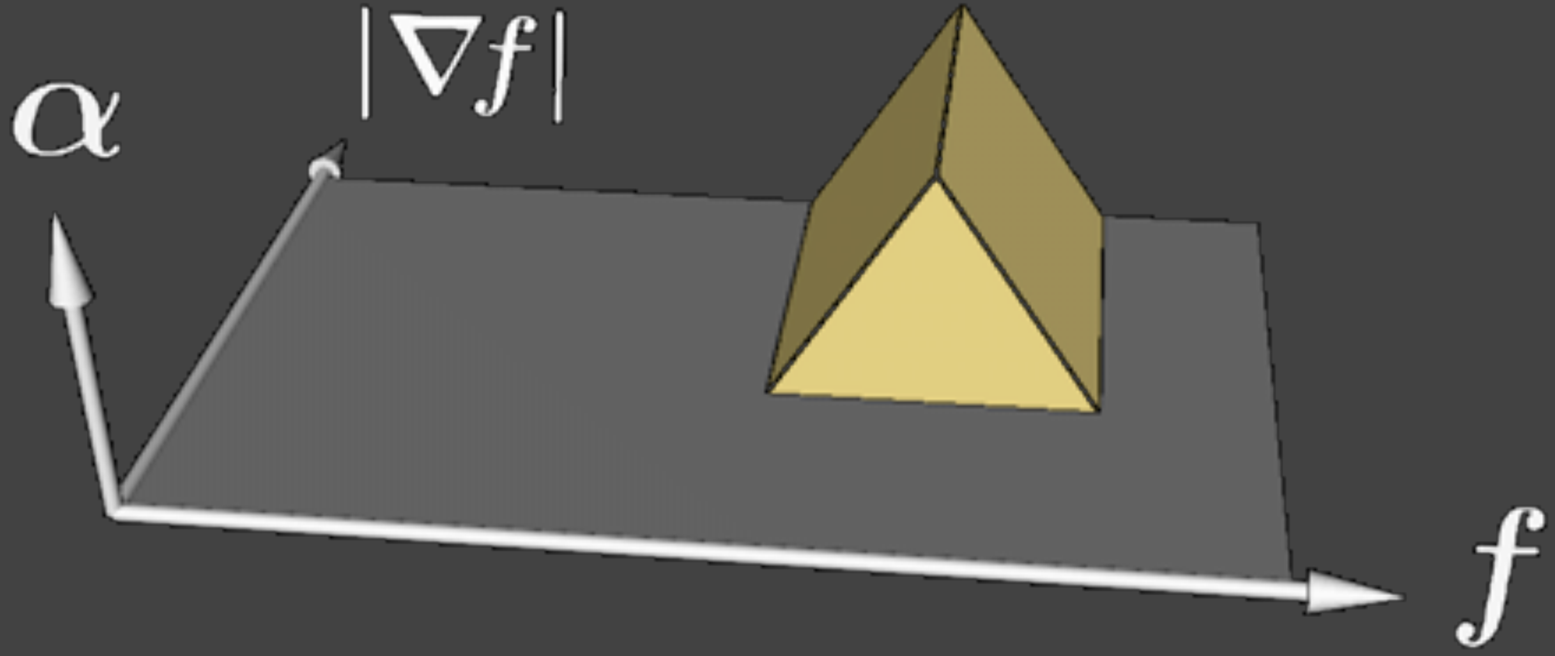
$$\left.\begin{array}{r} RGB(f, |\nabla f|) \\ \alpha(f, |\nabla f|) \end{array}\right\} \text{Modify…}$$



2D Transfer Function

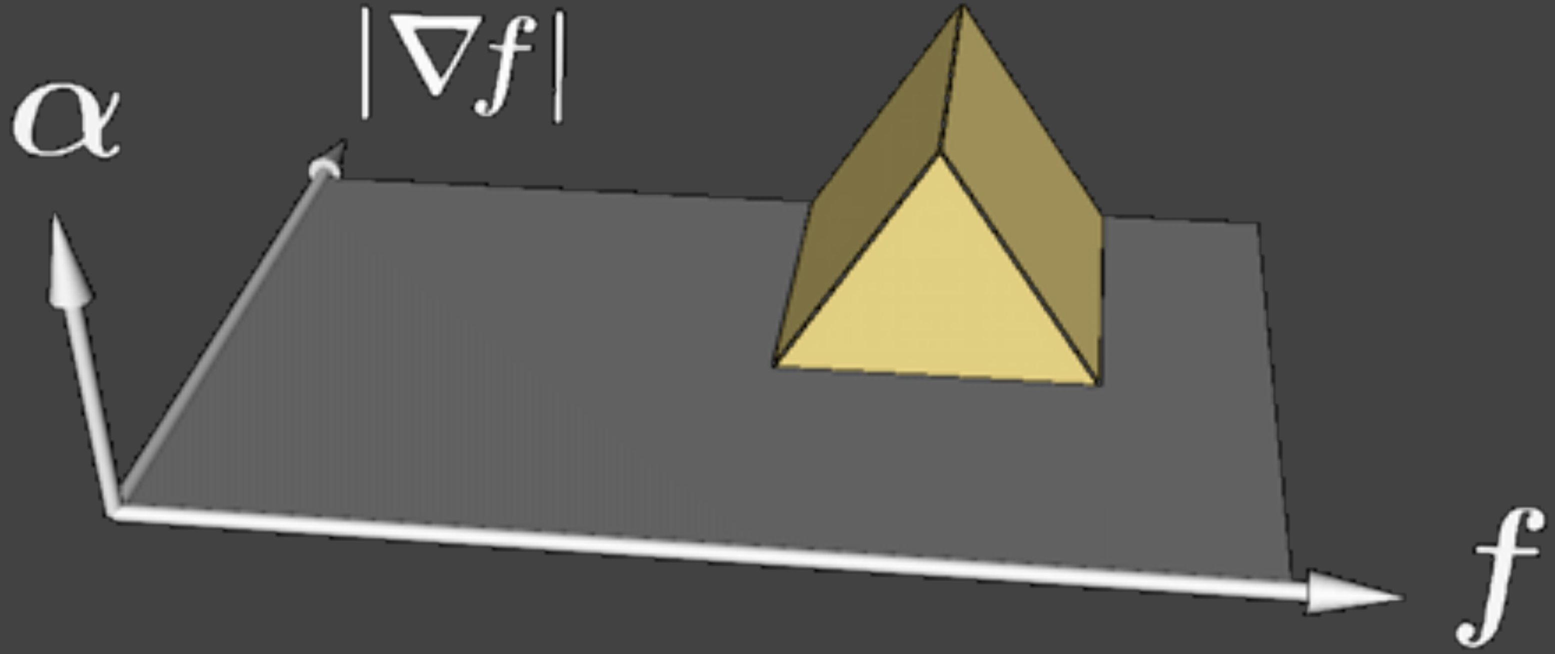
$$\left.\begin{matrix} \mathrm{RGB}(f, |\nabla f|) \\ \alpha(f, |\nabla f|) \end{matrix}\right\} \text{Modify...}$$



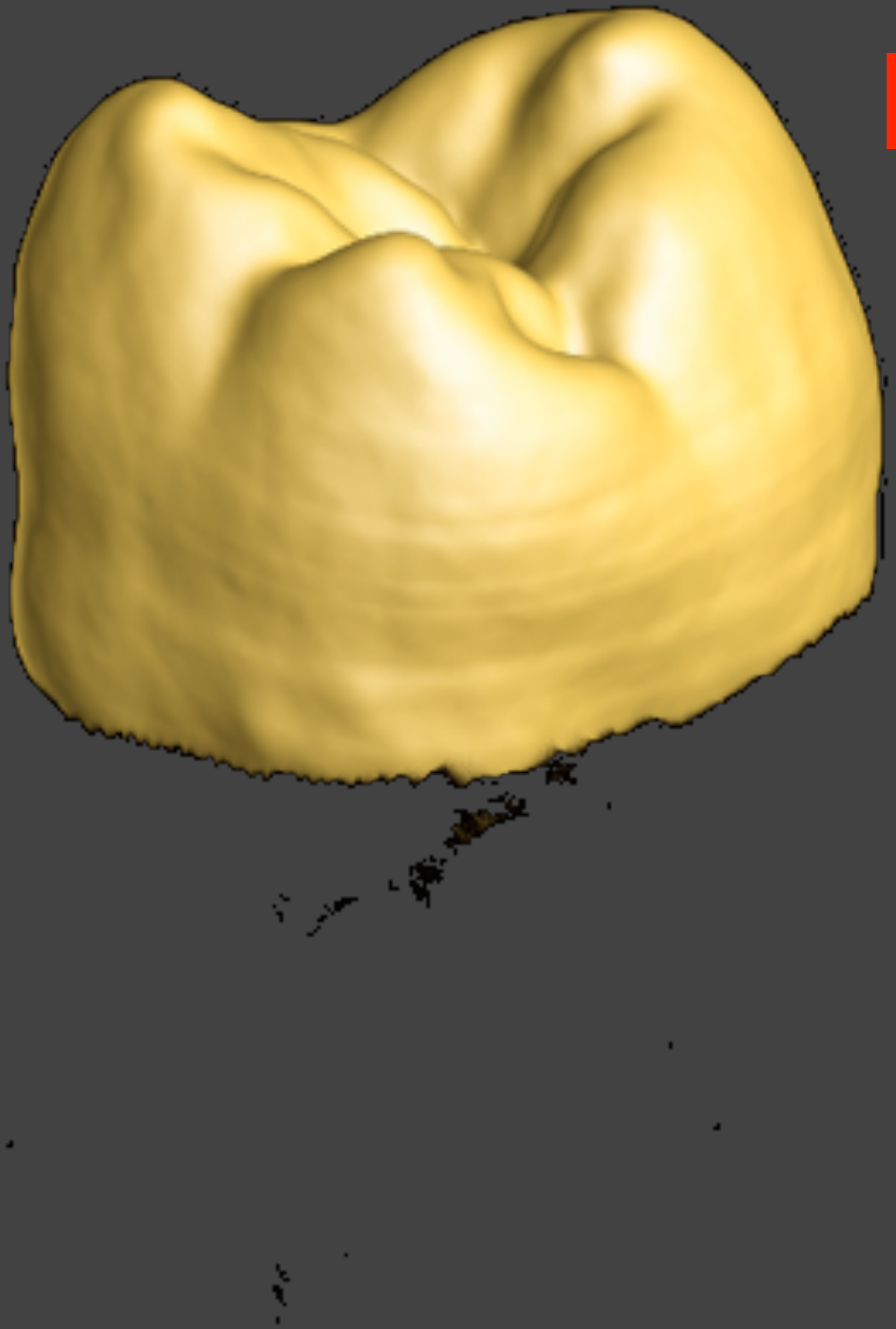


2D Transfer Function

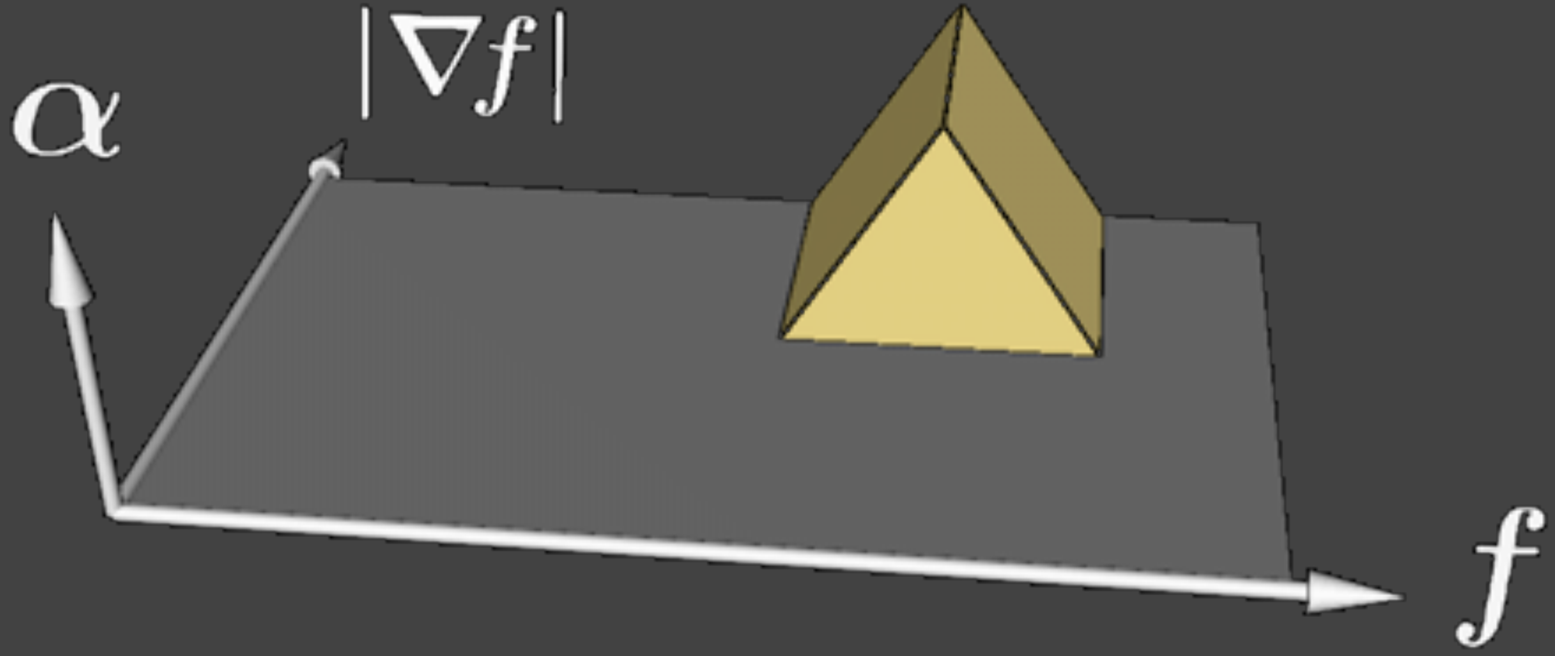
$$\left.\begin{aligned}RGB(f, |\nabla f|)\\ \alpha(f, |\nabla f|)\end{aligned}\right\}\text{Modify…}$$



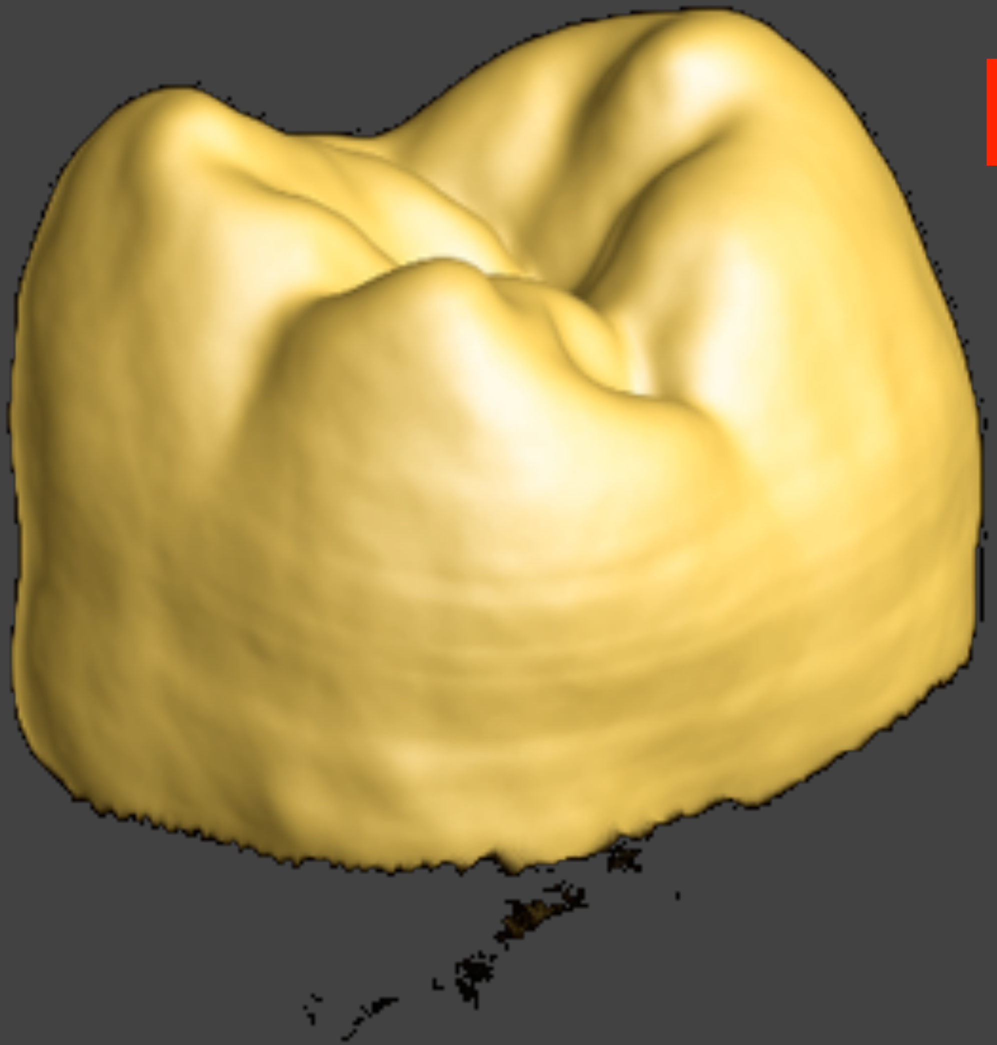
2D Transfer Function



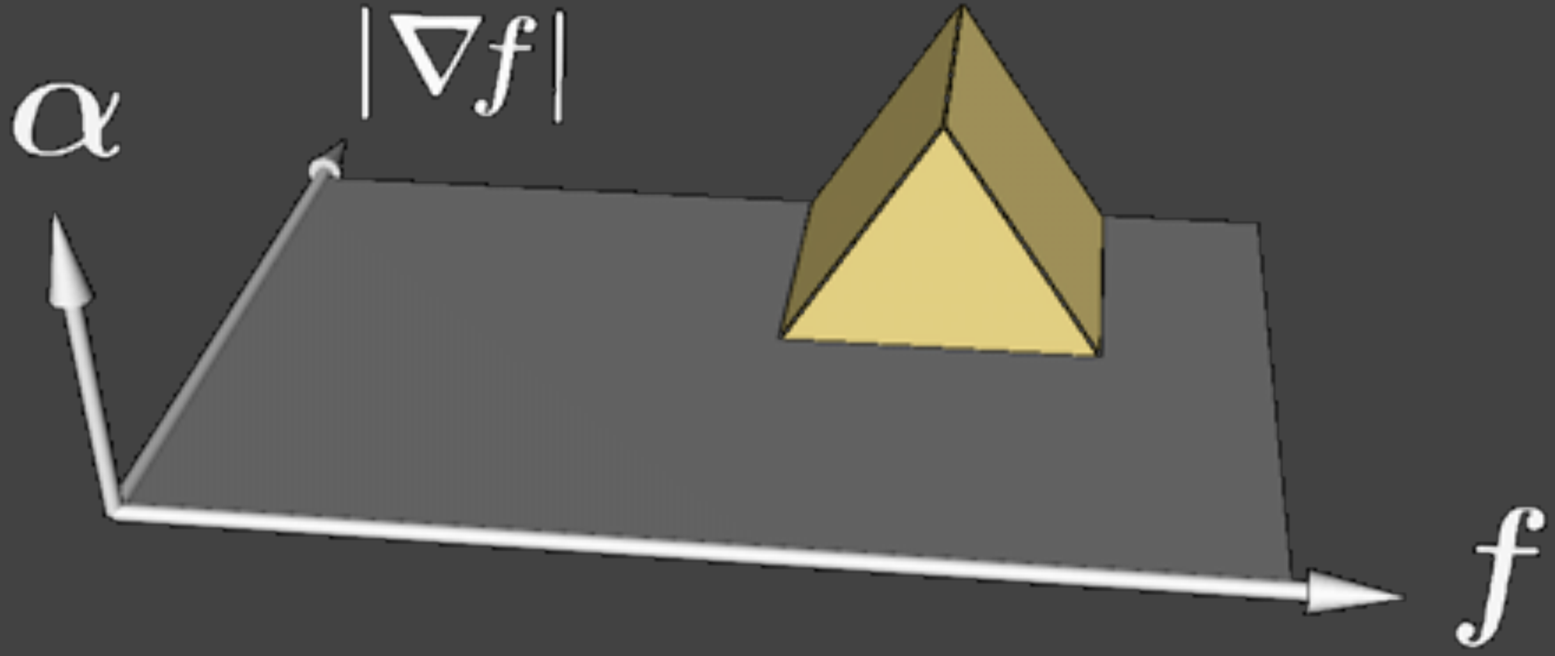
$$\left.\begin{aligned} RGB(f, |\nabla f|) \\ \alpha(f, |\nabla f|) \end{aligned}\right\} \text{Modify…}$$



2D Transfer Function



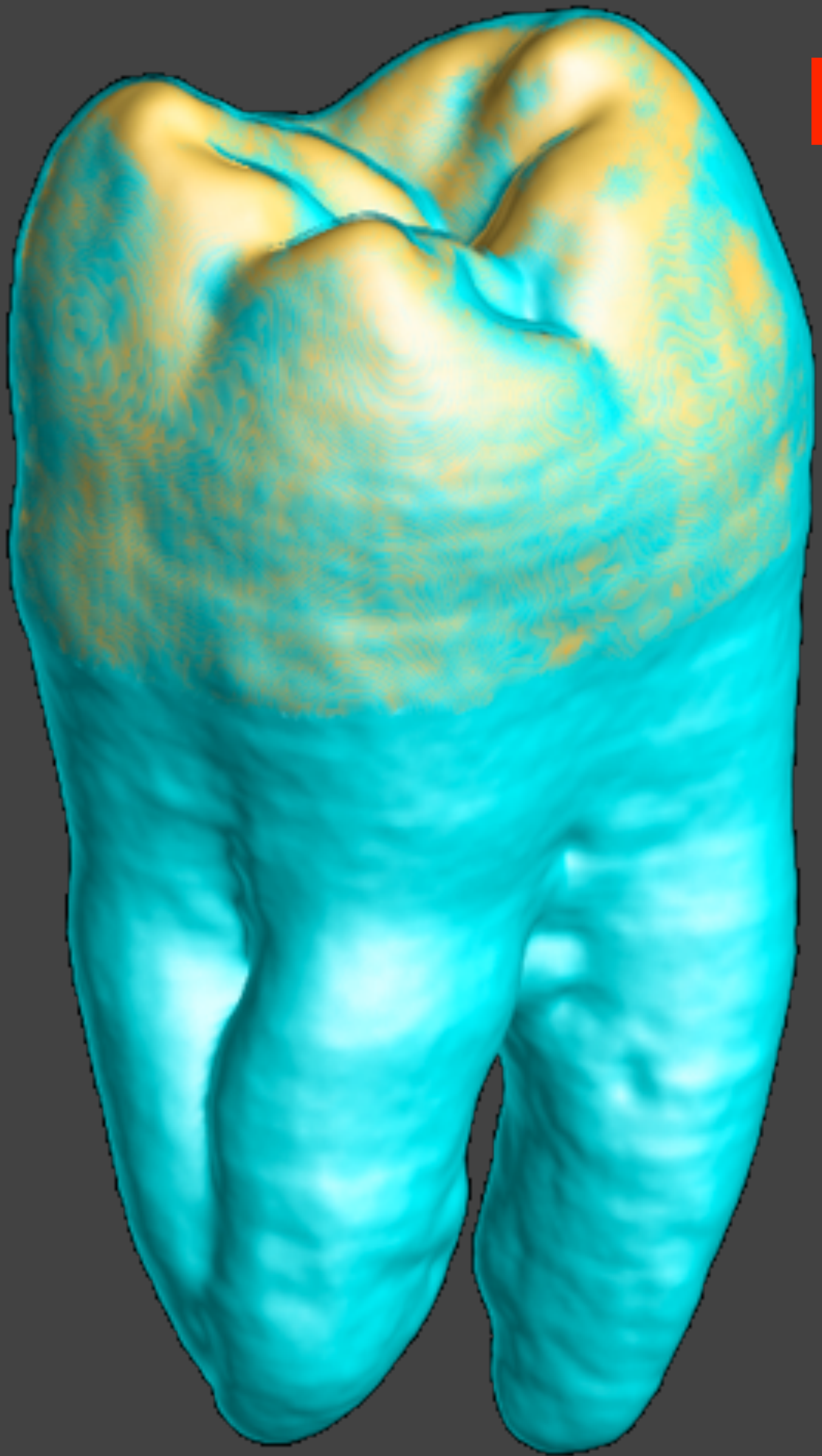
$$\left.\begin{array}{r} RGB(f, |\nabla f|) \\ \alpha(f, |\nabla f|) \end{array}\right\} \text{Modify...}$$



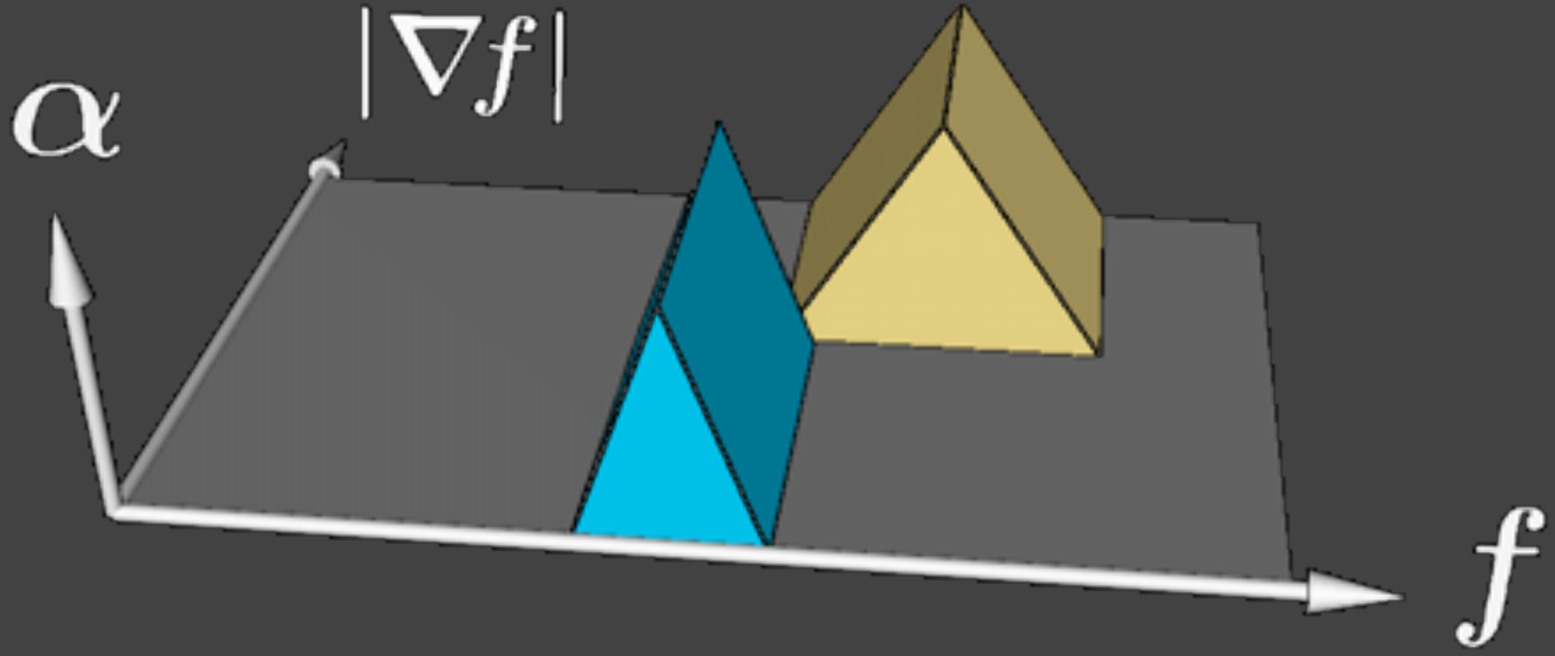
2D transfer functions give greater flexibility in boundary visualization

Display of Surfaces from Volume Data, Levoy 1988

2D Transfer Function

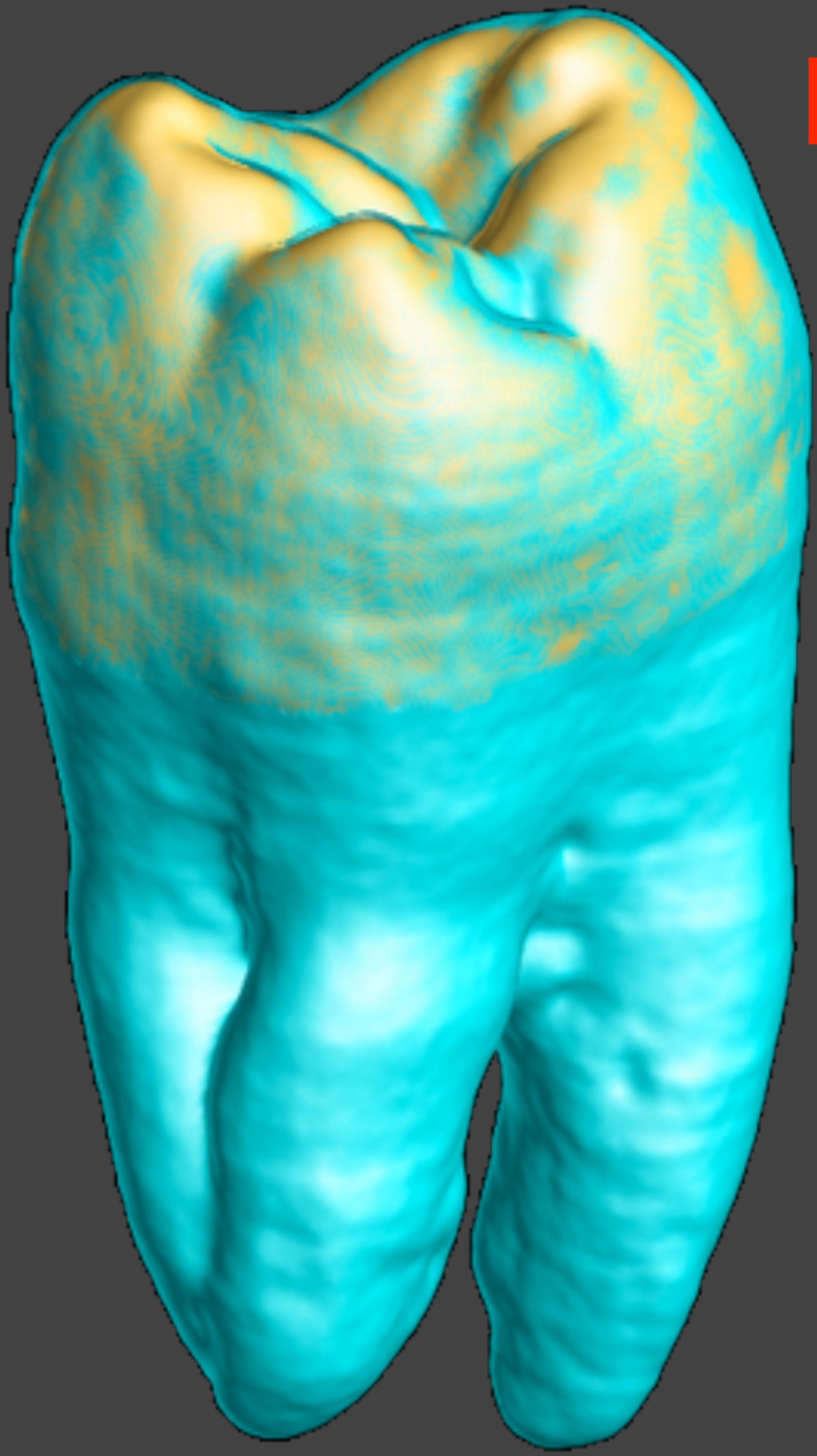


$$\left.\begin{array}{r}\mathrm{RGB}(f, |\nabla f|) \\ \alpha(f, |\nabla f|)\end{array}\right\}$$ Modify…

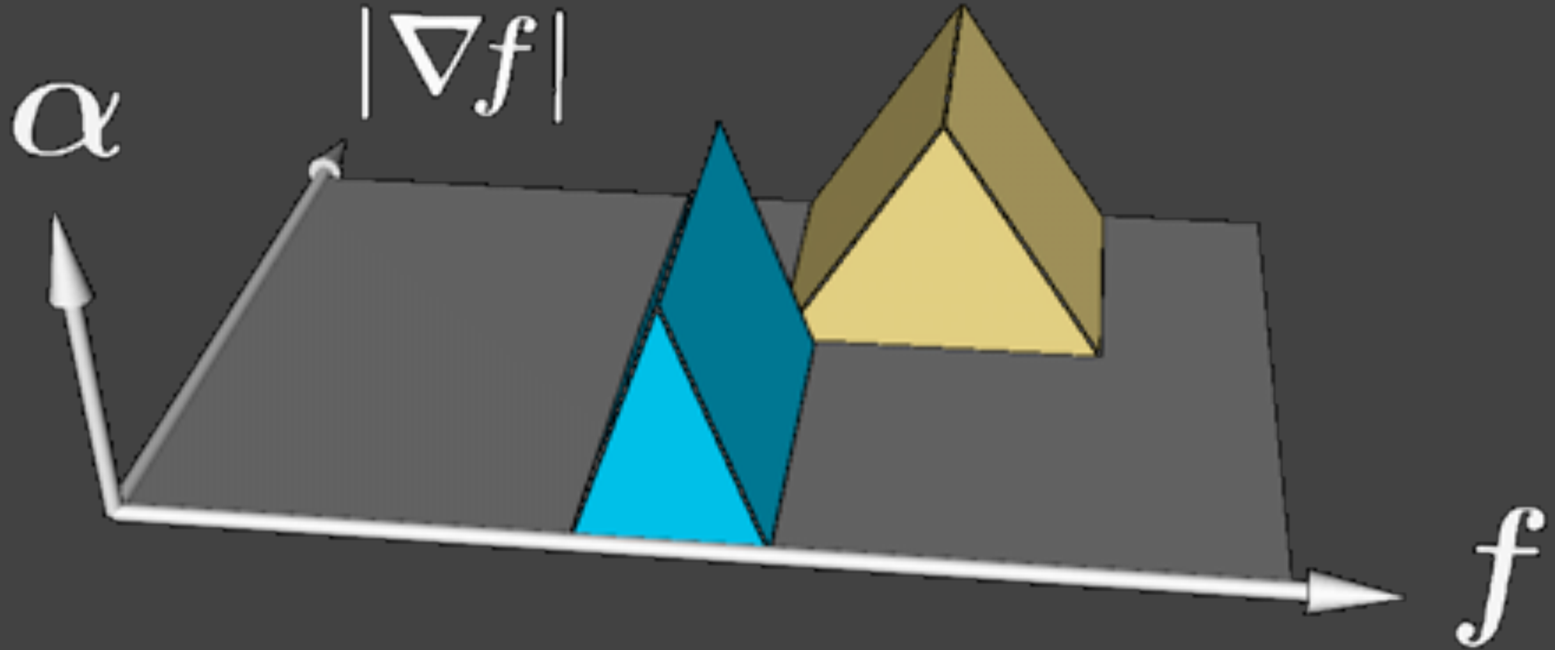


Trying to reintroduce dentin / background boundary …

2D Transfer Function

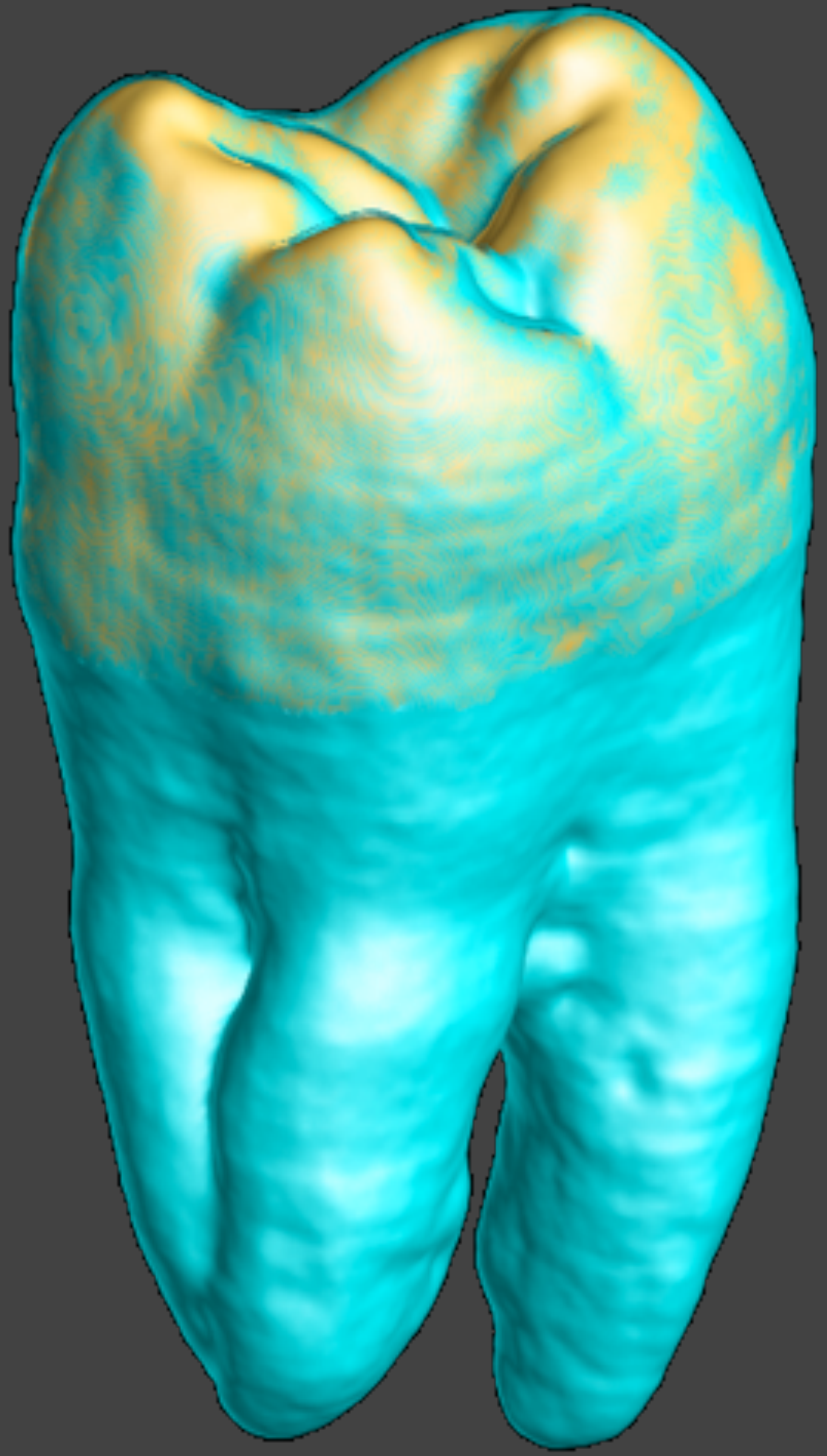


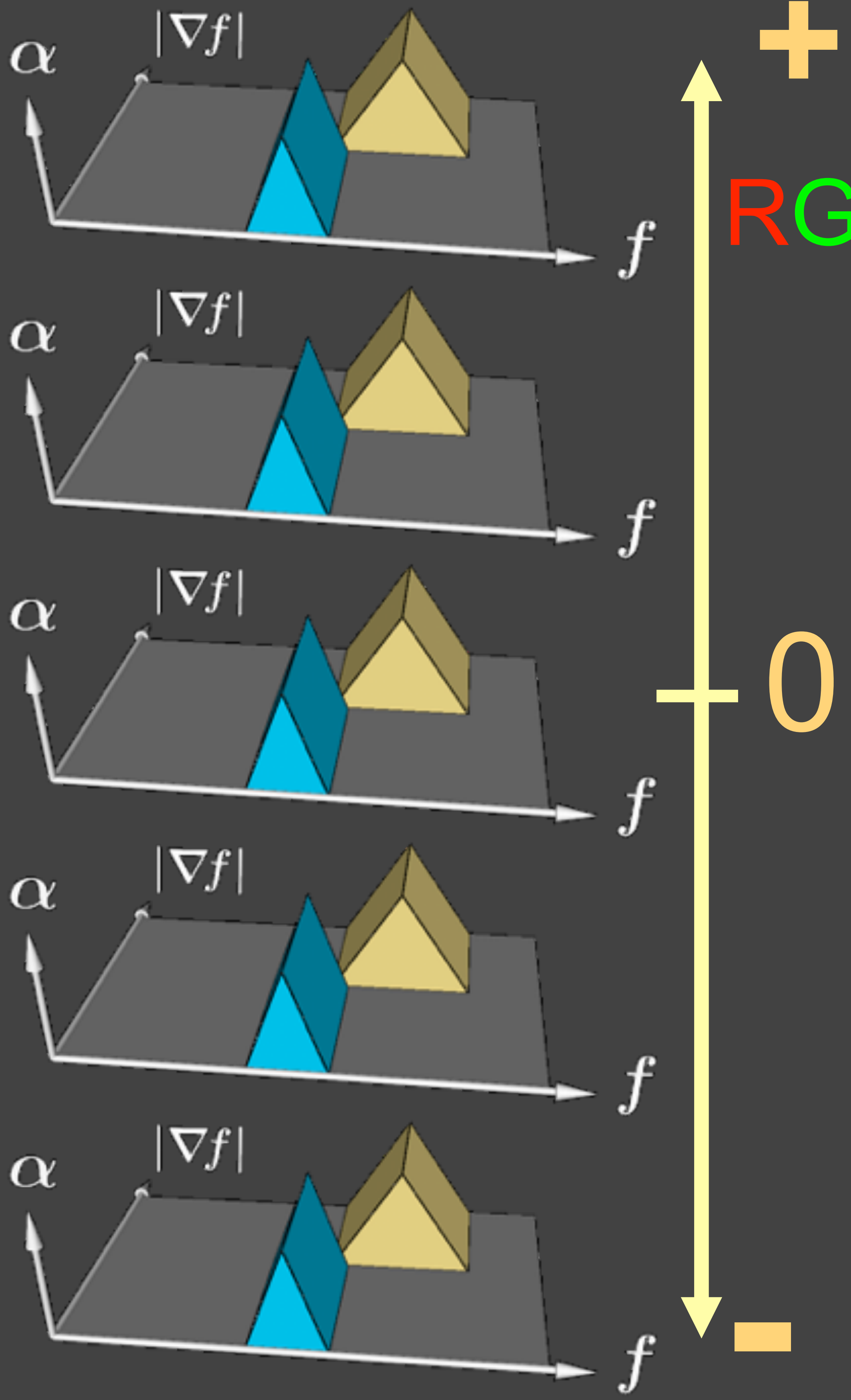
$$\left.\begin{array}{r} \mathrm{RGB}(f, |\nabla f|) \\ \alpha(f, |\nabla f|) \end{array}\right\} \text{Modify…}$$



Trying to reintroduce dentin / background boundary …

2D → 3D Transfer Function





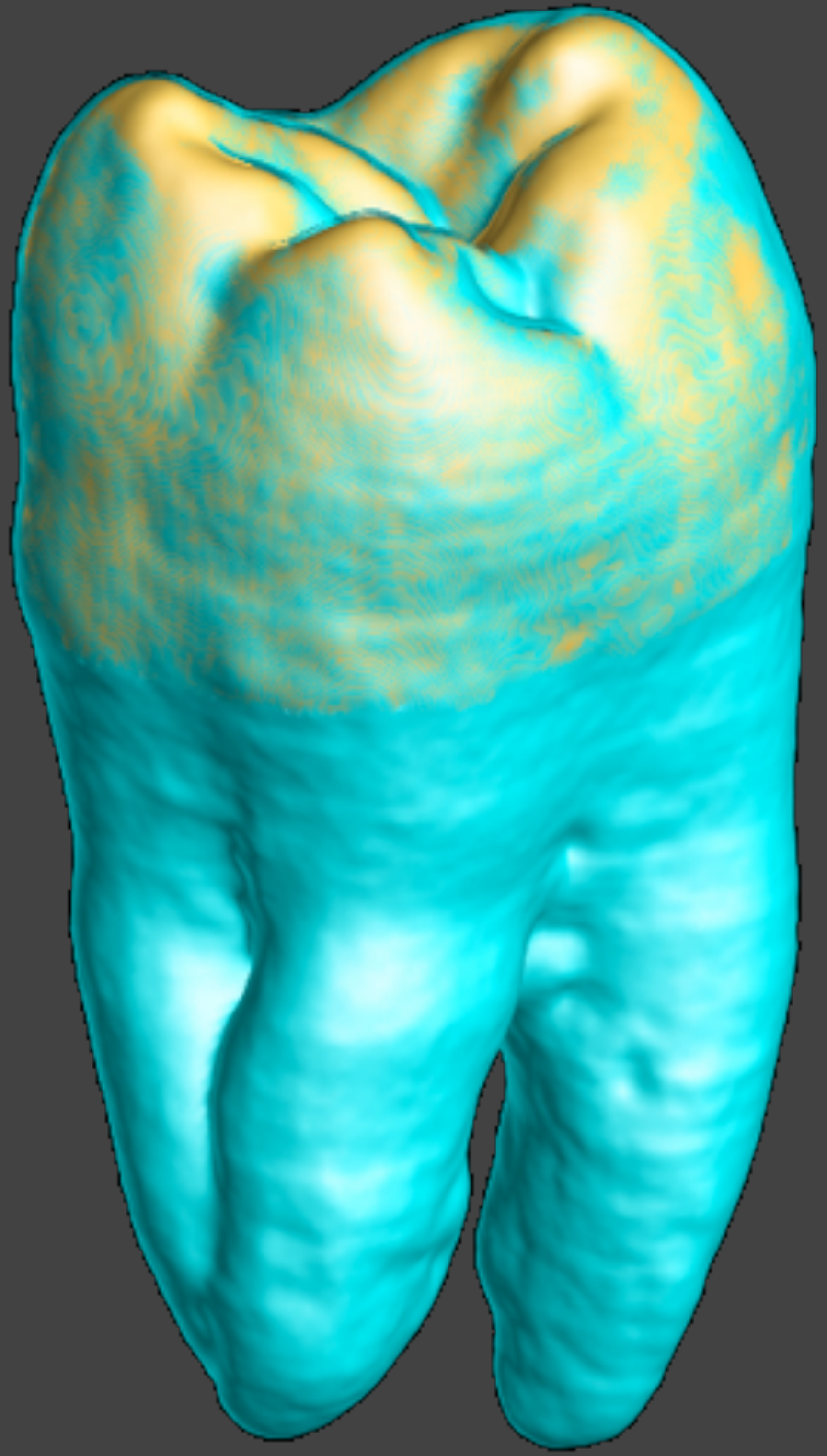
RGB $\alpha(f, |\nabla f|, D^2_{\widehat{\nabla f}} f)$

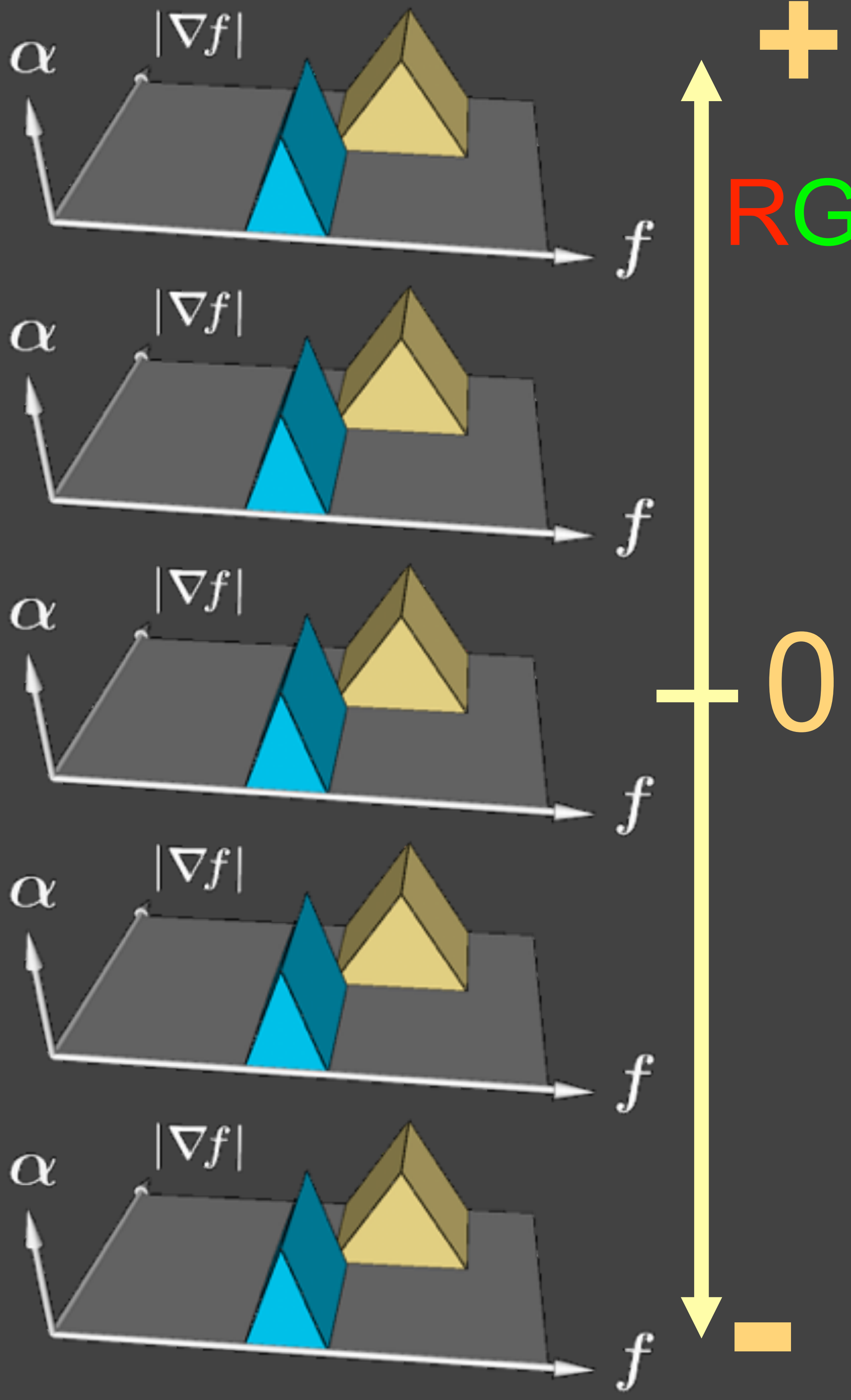
Second directional derivative

$D^2_{\widehat{\nabla f}} f$

measured with Hessian

3D Transfer Function

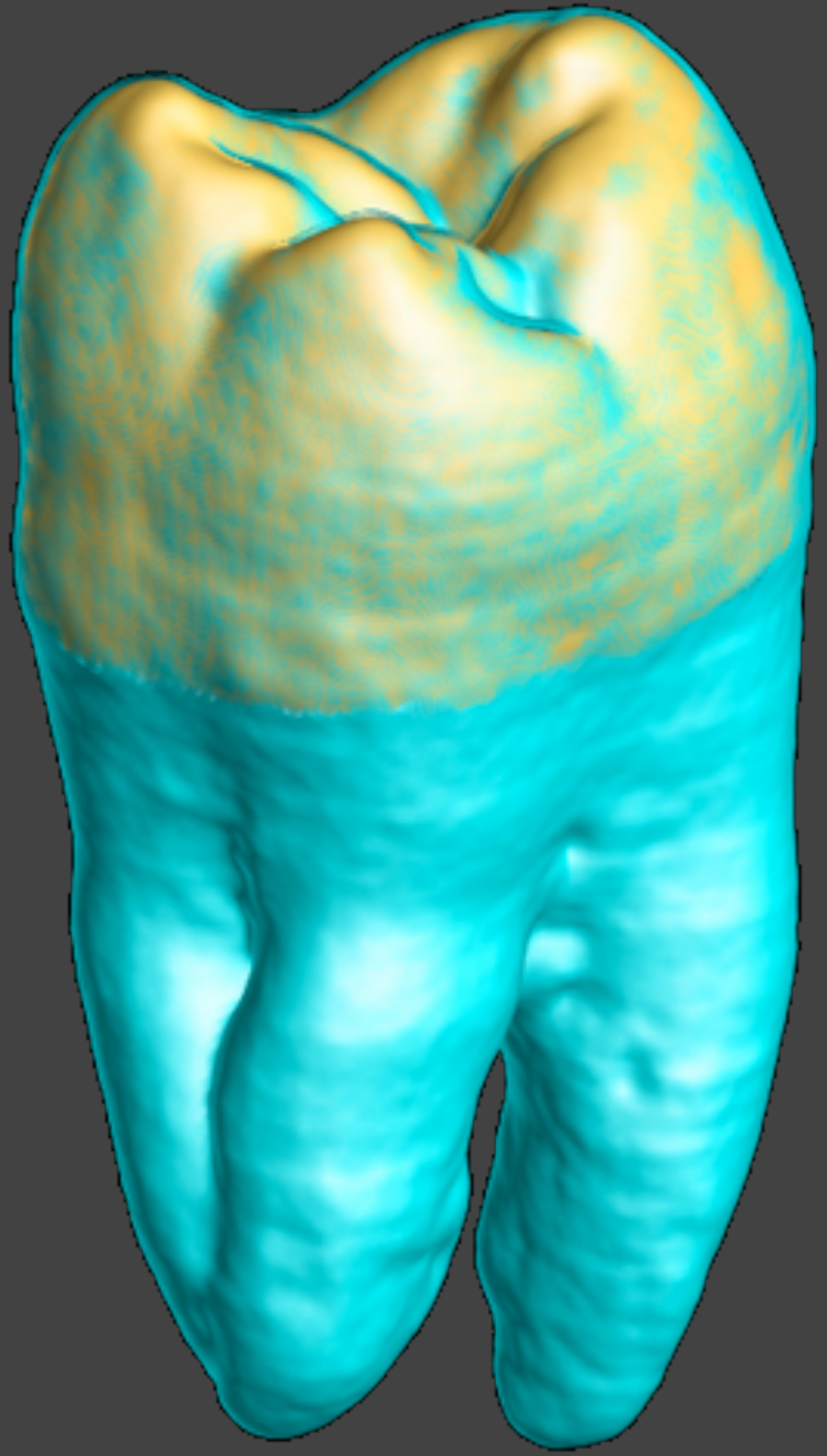


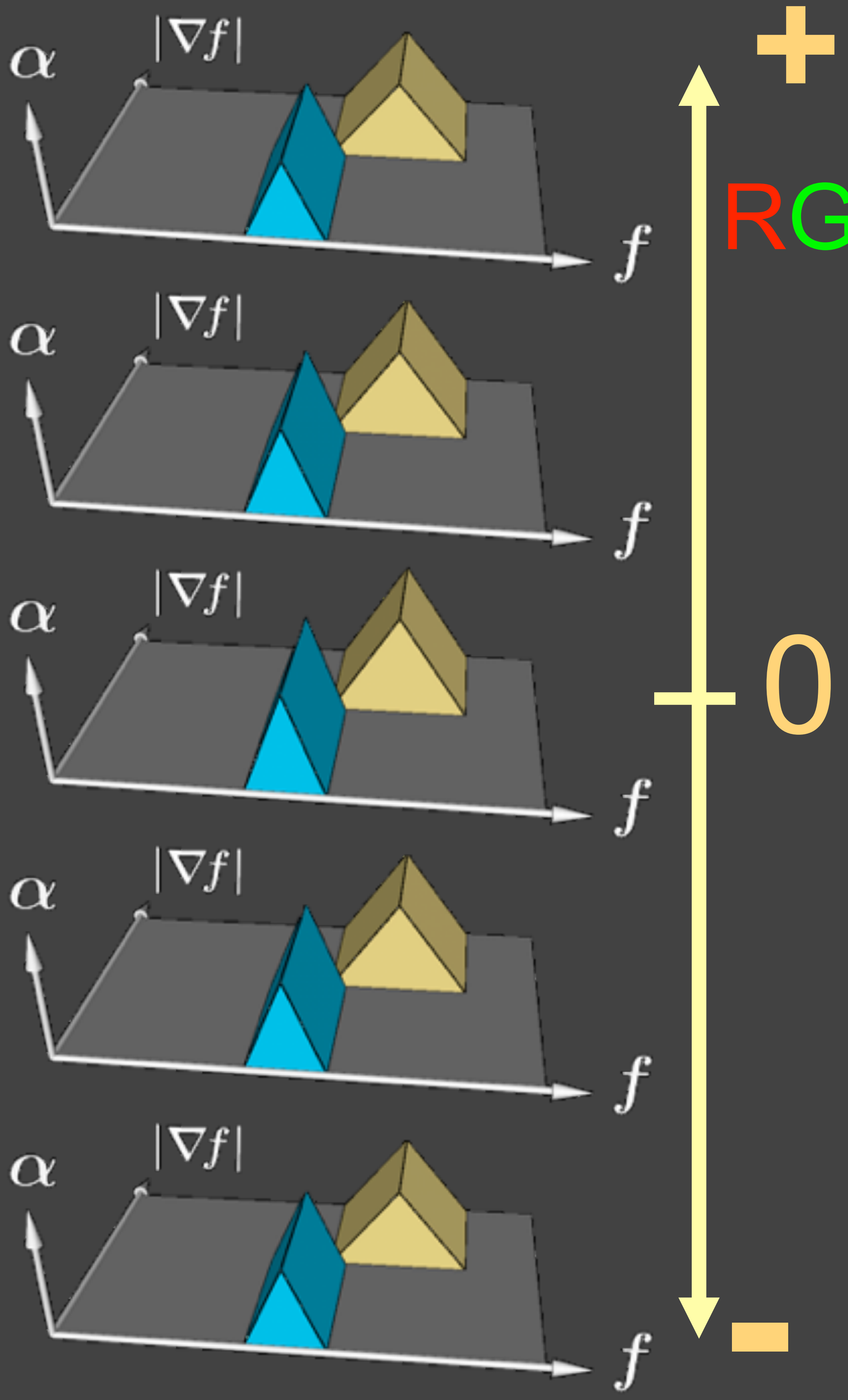


RGB $\alpha(f, |\nabla f|, \mathbf{D}^2_{\widehat{\nabla} f} f)$

Modify…

3D Transfer Function

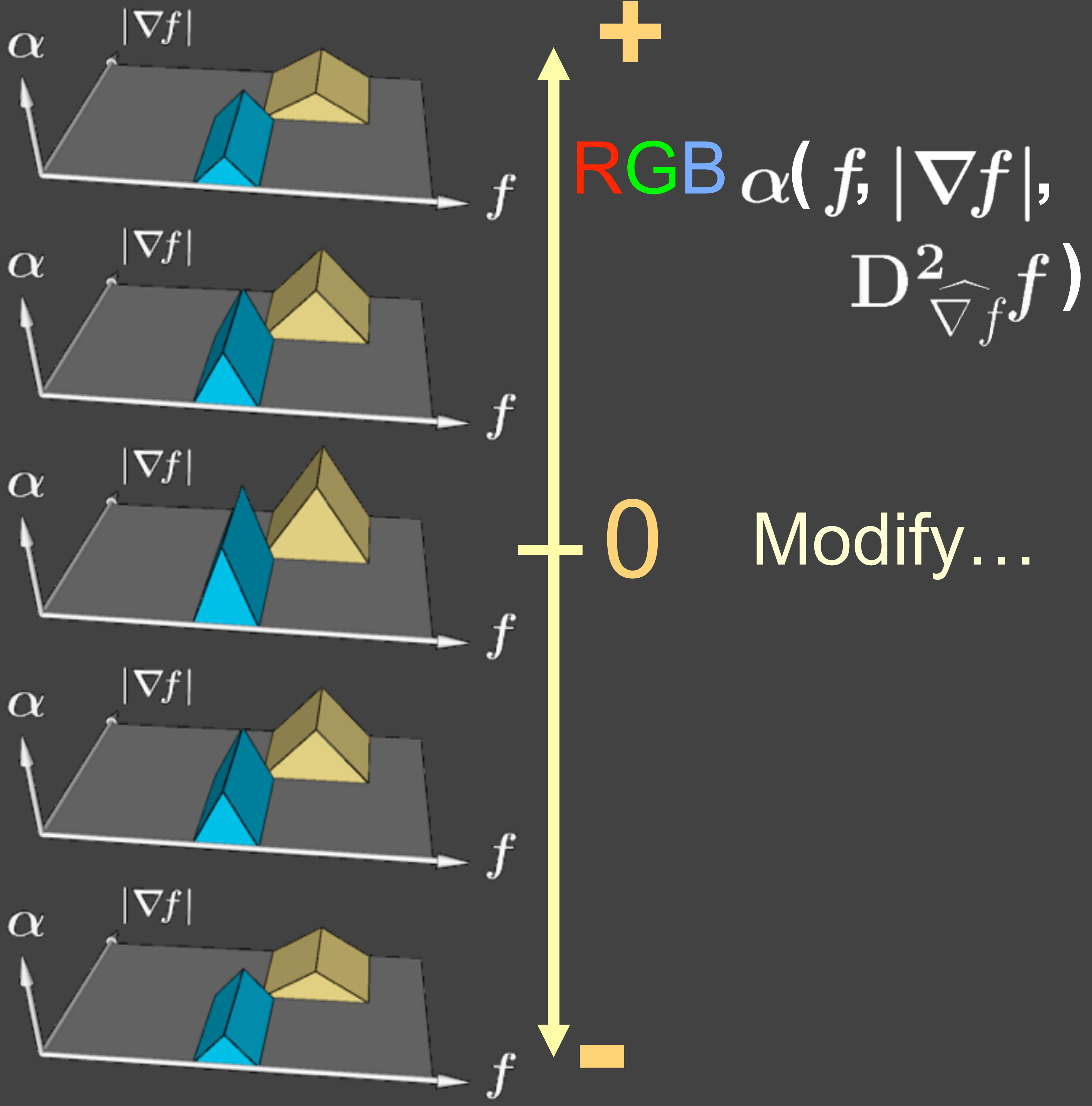




RGB $\alpha(f, |\nabla f|, \mathrm{D}^2_{\widehat{\nabla} f} f)$

Modify…

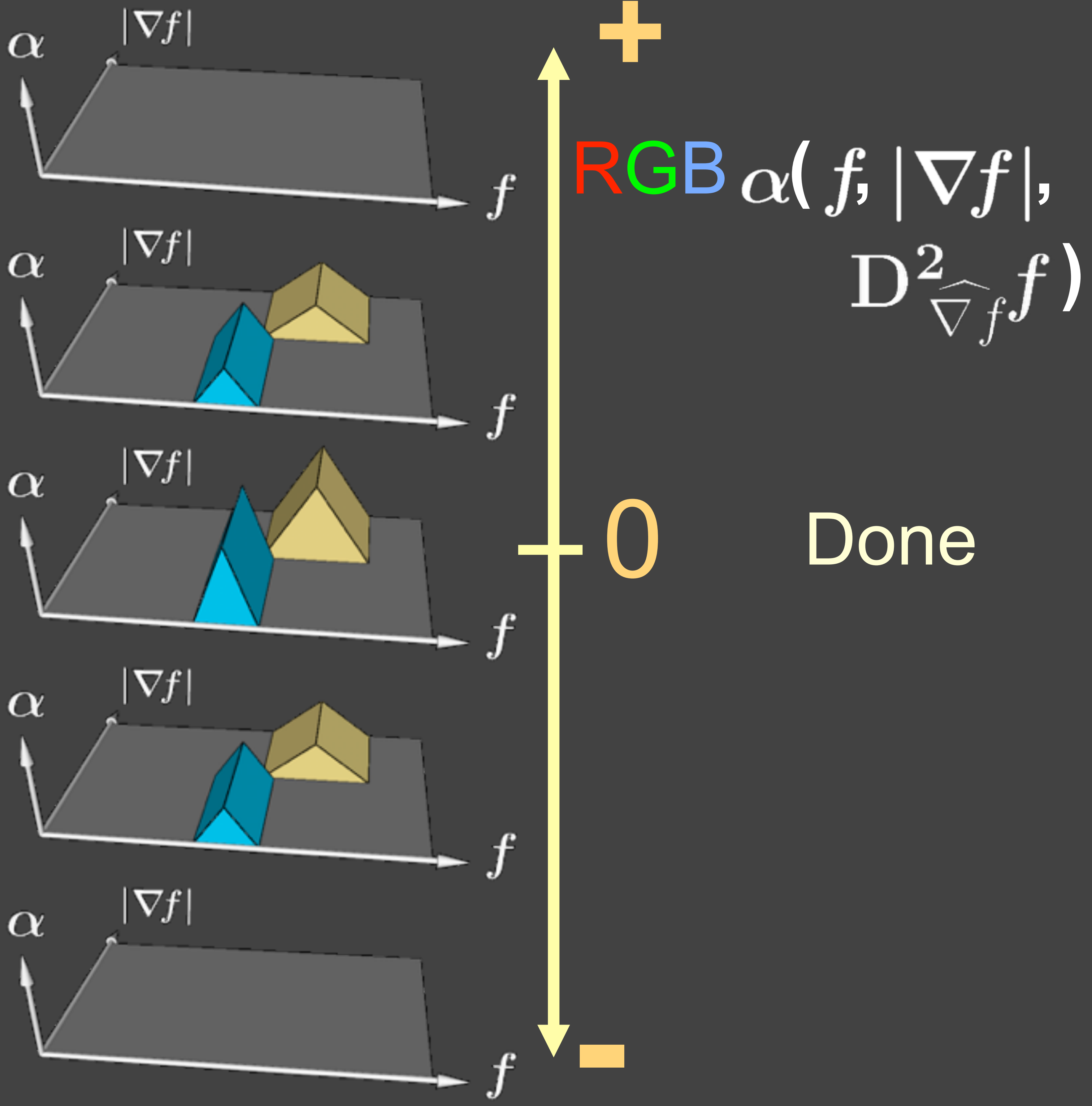
3D Transfer Function



RGB $\alpha(f, |\nabla f|, D^2_{\widehat{\nabla f}} f)$

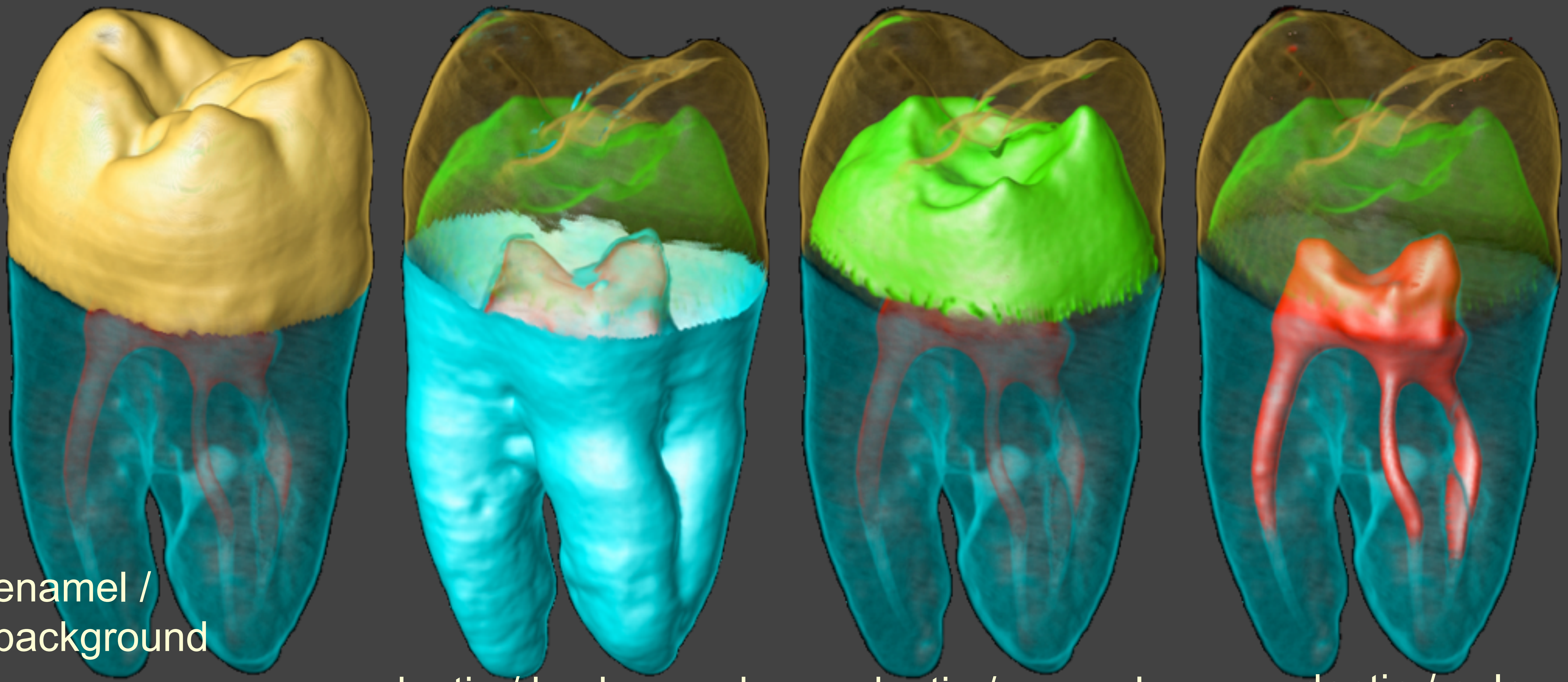
Modify…

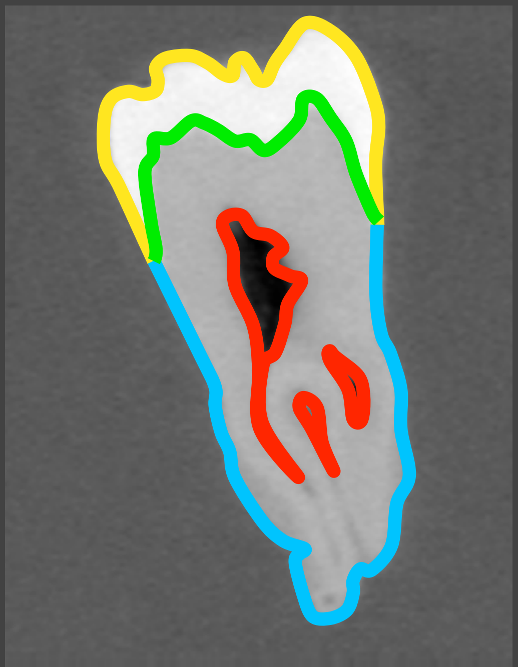
3D Transfer Function



α
$|\nabla f|$
f
+
RGB $\alpha(f, |\nabla f|, \mathbf{D}^2_{\widehat{\nabla} f} f)$
0
Done
-

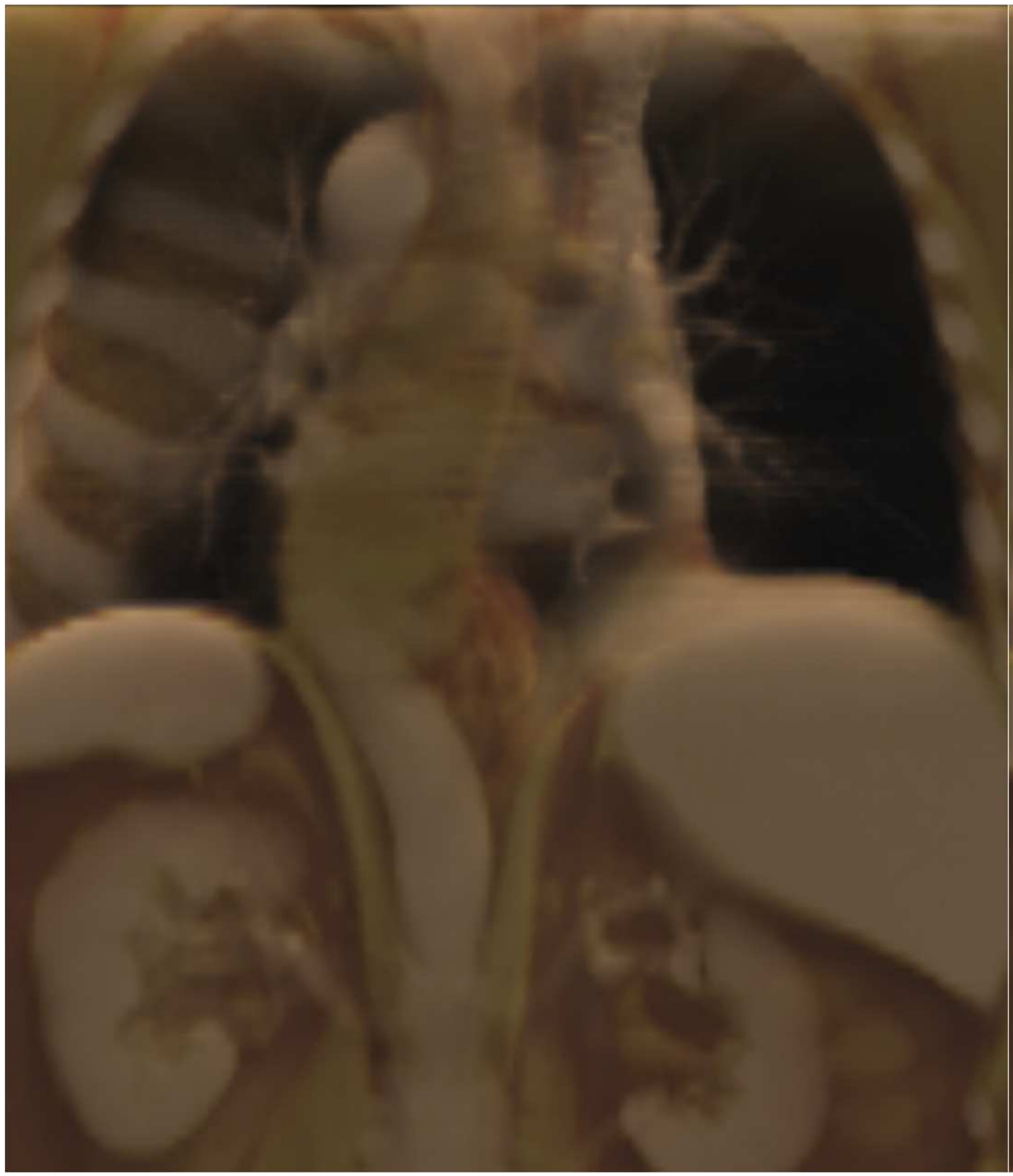
3D Transfer Function



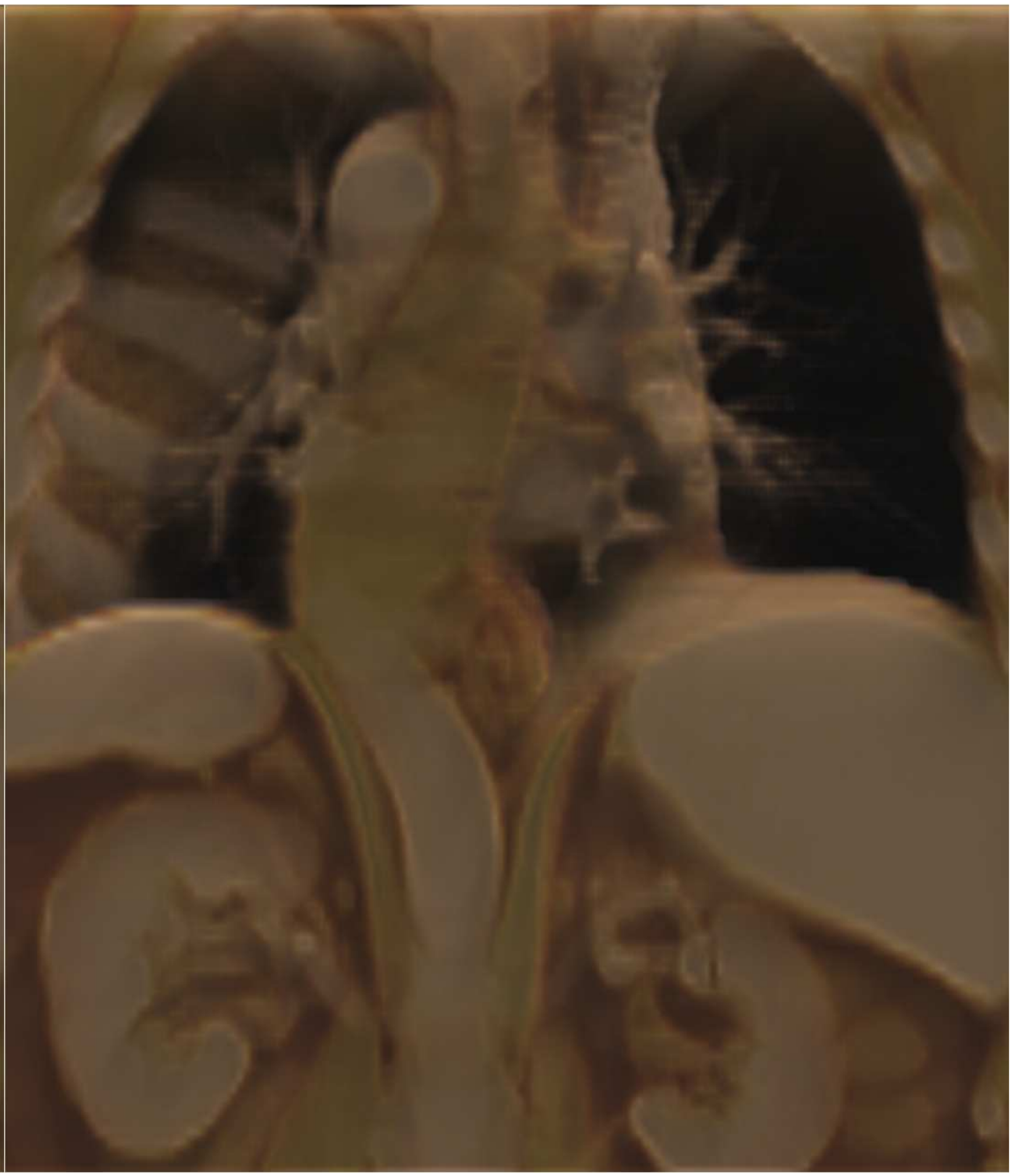


1D: not possible

2D: specificity not as good



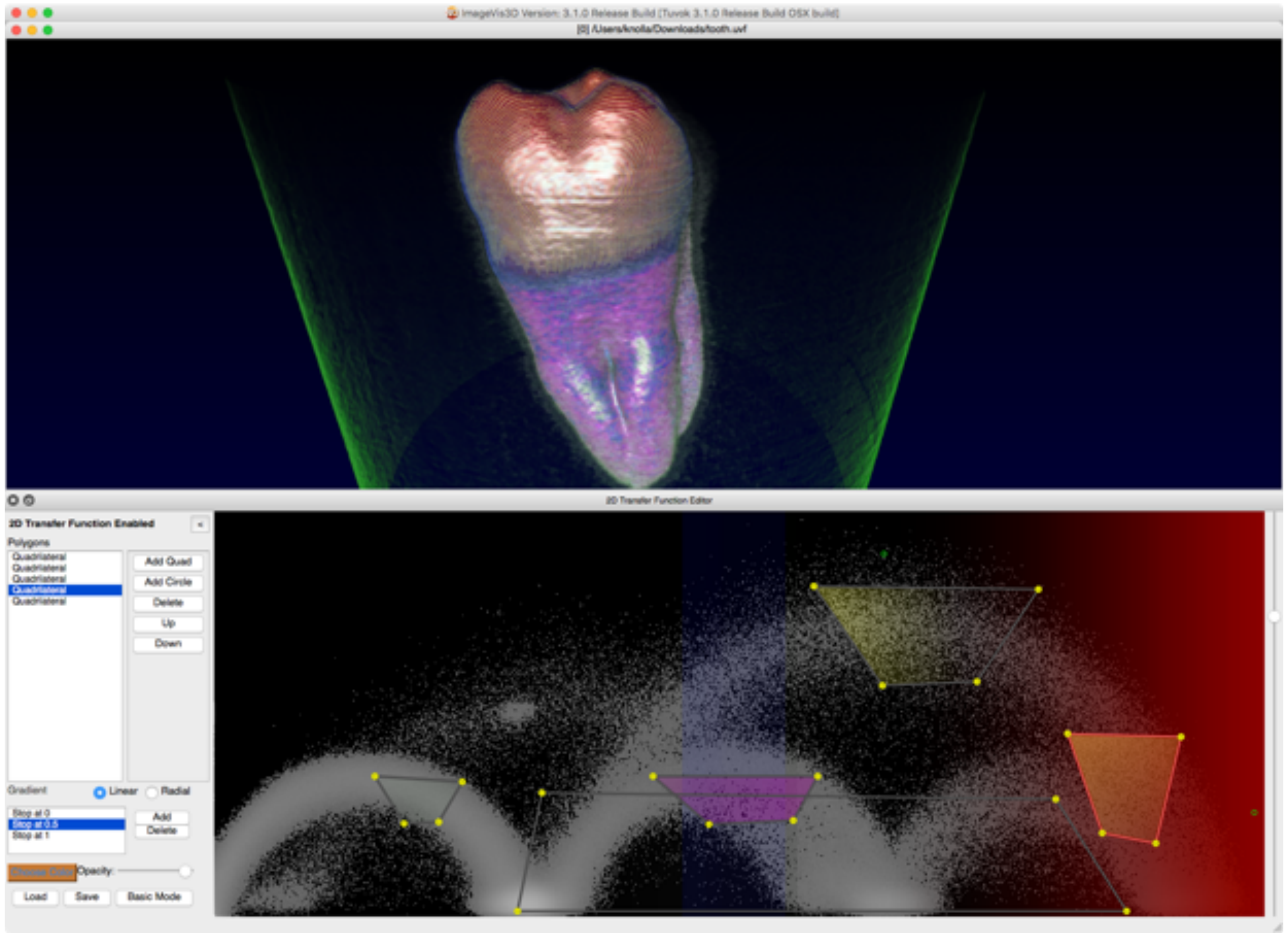
Original TF



Boundaries (gradient)

ImageVis3D demo

- http://www.sci.utah.edu/software/imagevis3d.html



www.sci.utah.edu

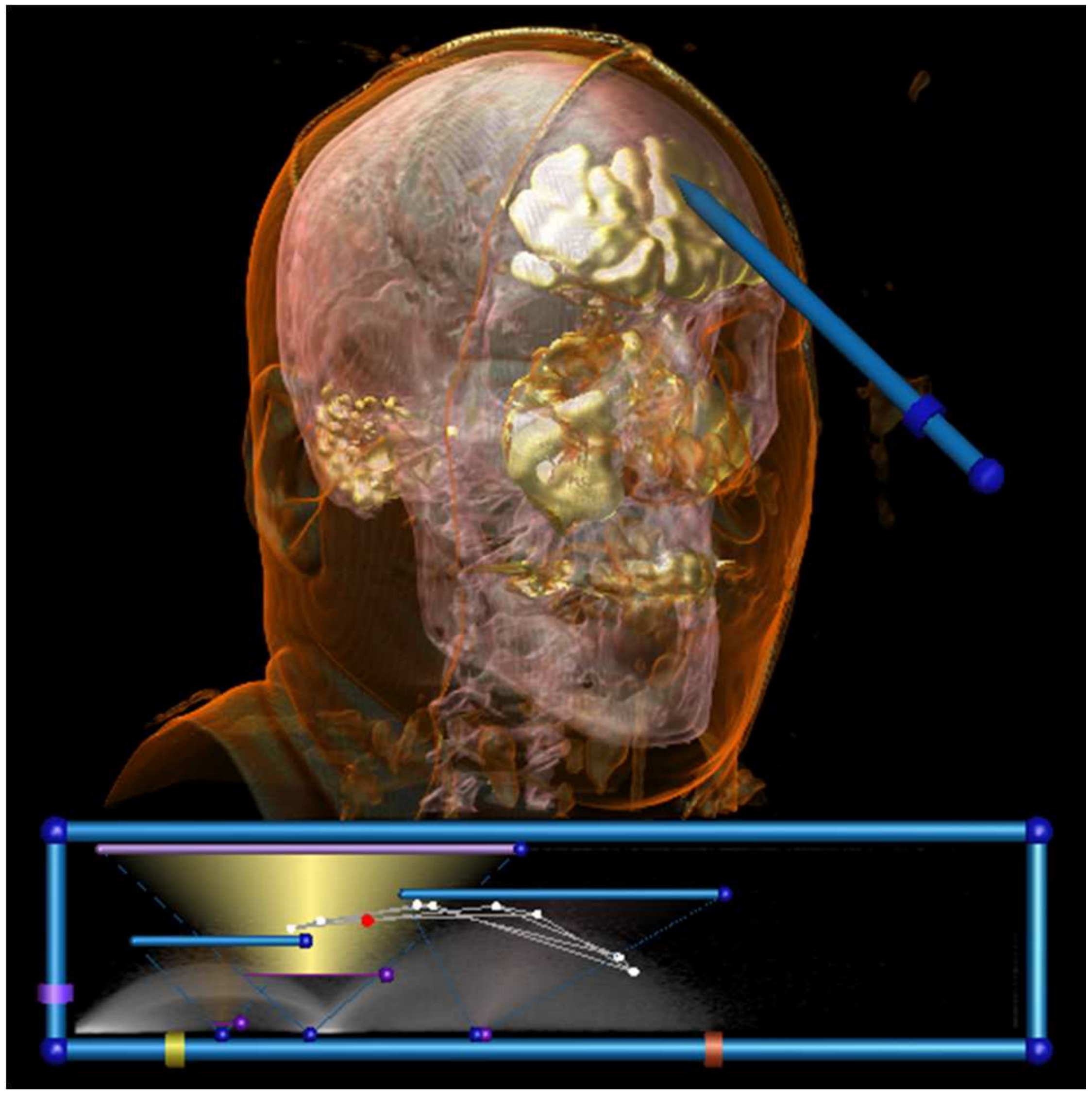
Multi-Dimensional TFs

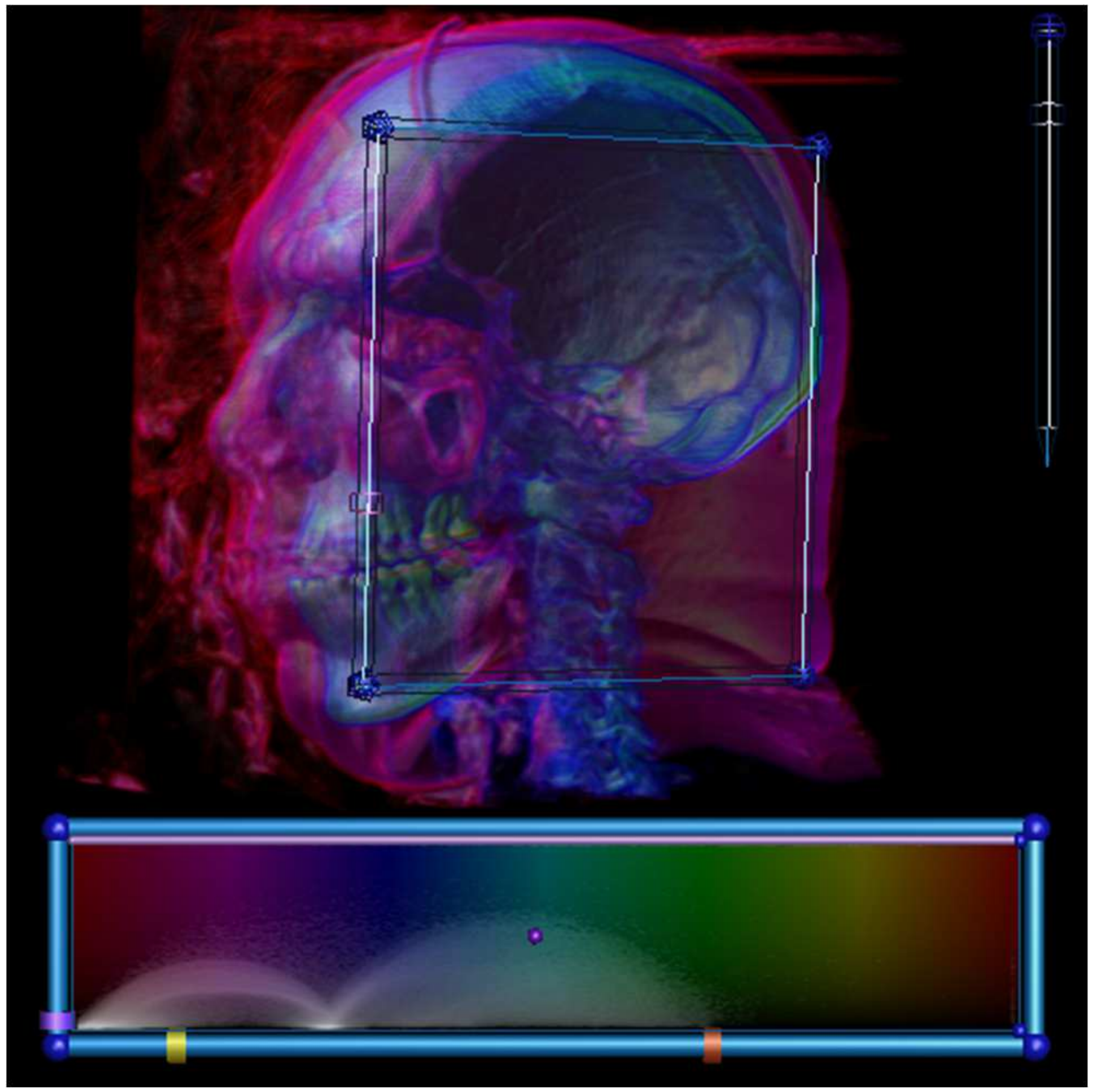
- Strengths:
  - Better flexibility, specificity
  - Higher quality visualizations
- Weaknesses:
  - Even harder to specify
  - Unintuitive relationship with boundaries
  - Greater demands on user interface

Different Interaction

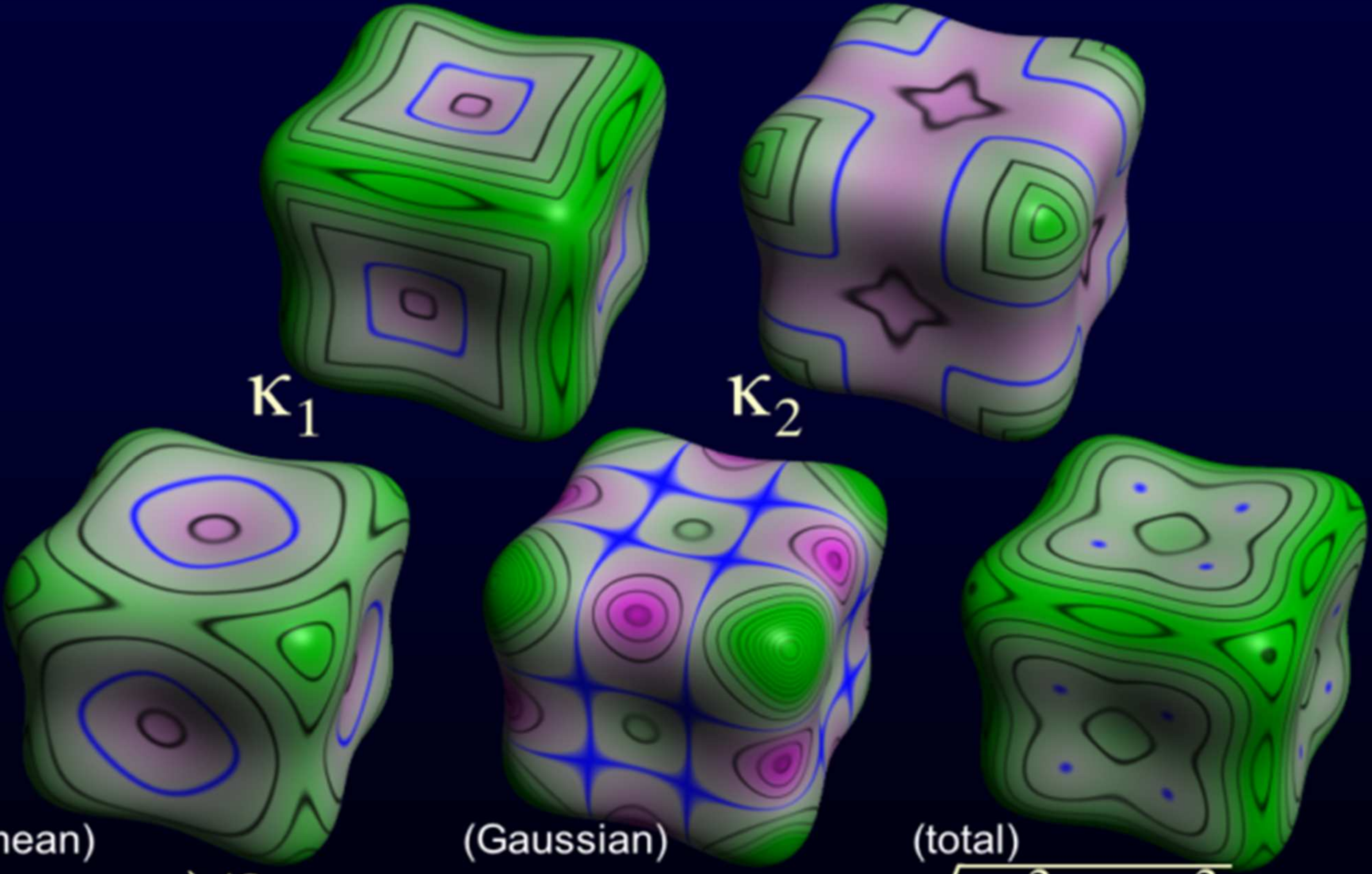
"Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets" Kniss, Kindlmann, Hansen: Vis '01

- Make things opaque by pointing at them
- Uses 3D transfer functions (value, 1st, 2nd derivative)
- "Paint" into the transfer function domain





Curvature measures



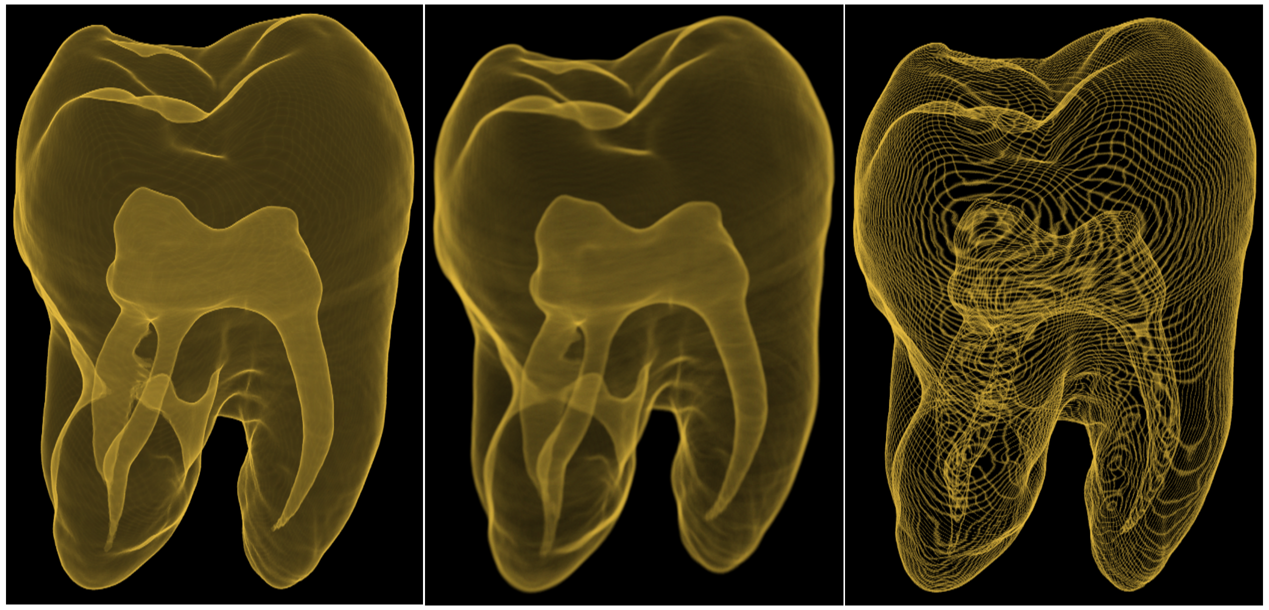
κ_1 κ_2

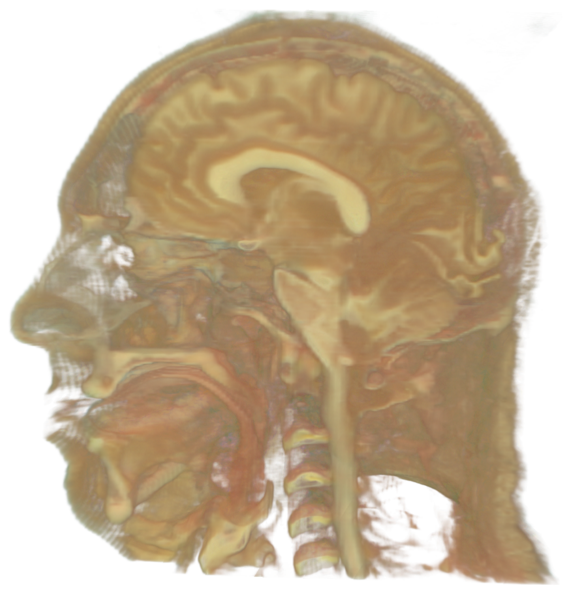
(mean) $(\kappa_1 + \kappa_2)/2$

(Gaussian) $\kappa_1 \kappa_2$

(total) $\sqrt{\kappa_1^2 + \kappa_2^2}$

Multidimensional gaussian transfer functions





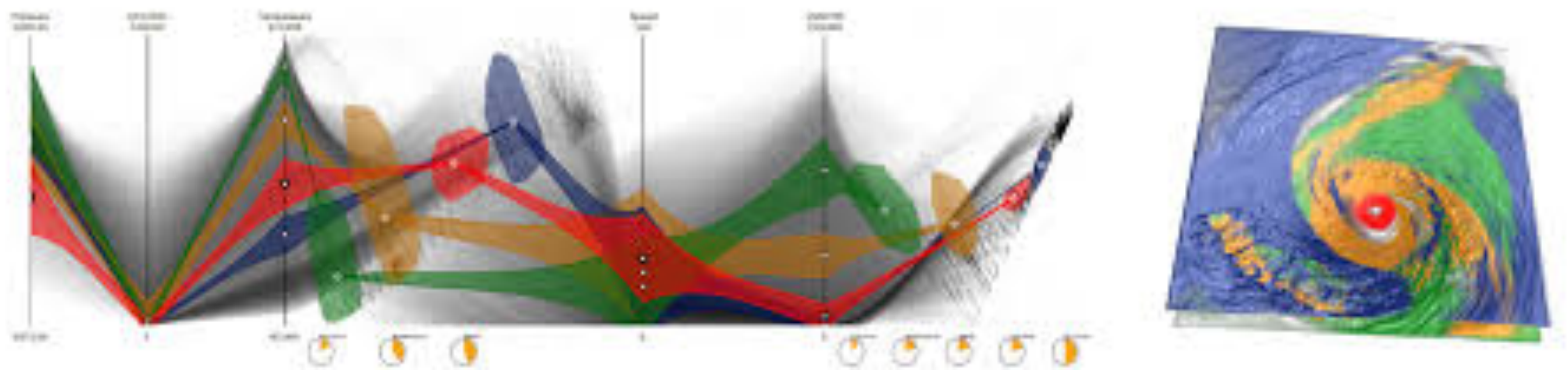
$$\mathrm{GTF}(\vec{v},\vec{c},\mathbf{K}) = e^{-(\vec{v}-\vec{c})^T\mathbf{K}^T\mathbf{K}(\vec{v}-\vec{c})}$$

$$\begin{array}{lll} \mathrm{IGauss}(x_1,x_2) = & \frac{\sqrt{\pi}}{2}\frac{\mathrm{erf}(x_1)-\mathrm{erf}(x_2)}{x_1-x_2} & x_1 \neq x_2 \\ \mathrm{IGauss}(x_1,x_2) = & e^{-x_1^2} & x_1 = x_2. \end{array}$$

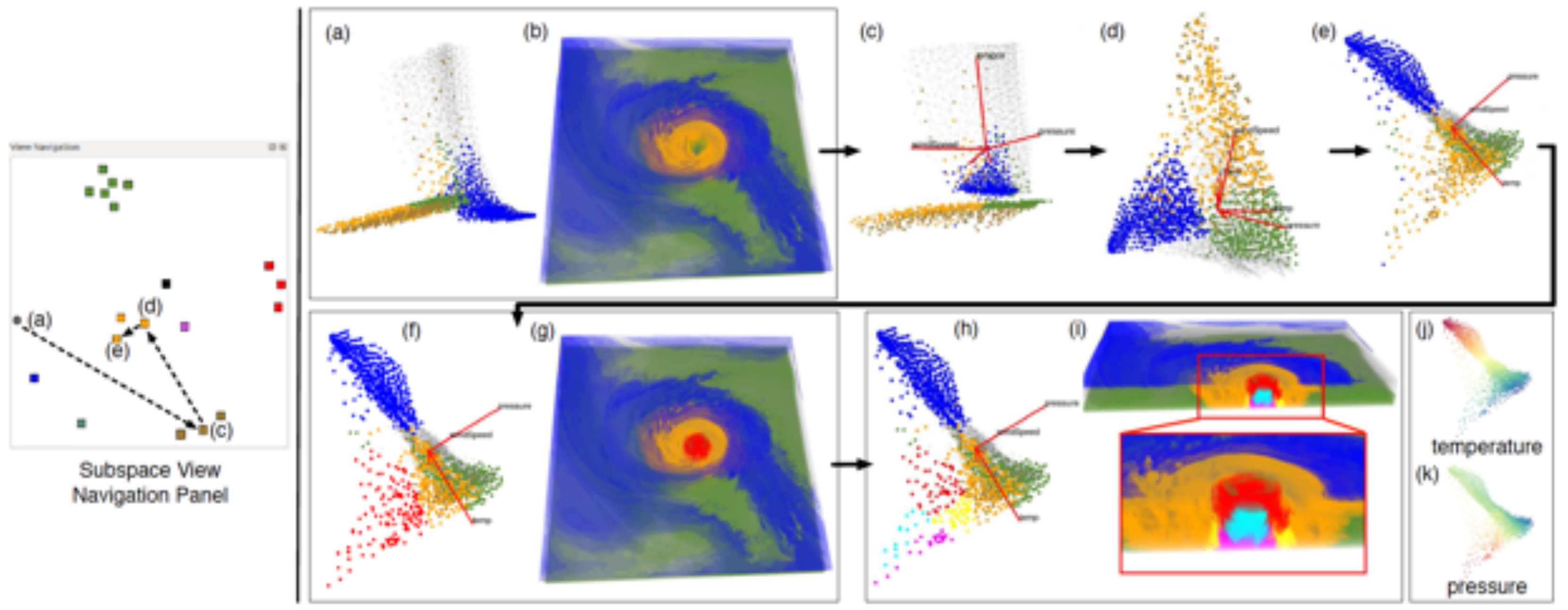
- Analytical integration of any-dimensional transfer functions, summed together as a multivariate Gaussian.
- For data with 2, 3, 4, etc. fields
- Piecewise-linear integration along the ray using compositing

Kniss et al. Gaussian Transfer Functions for Multi-field Volume Visualization. IEEE Vis 2003

Other multidimensional classification

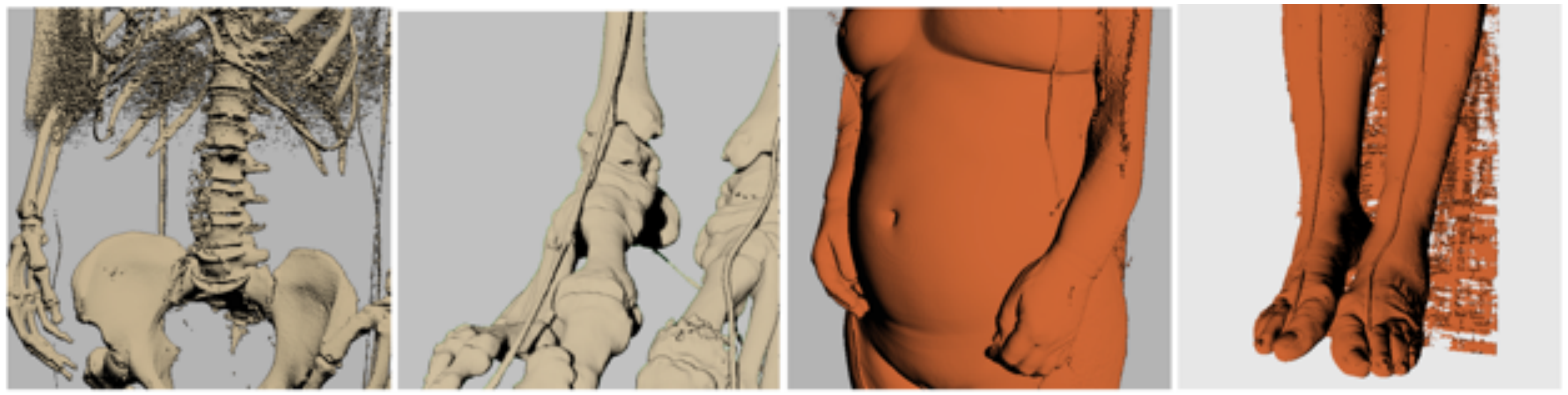


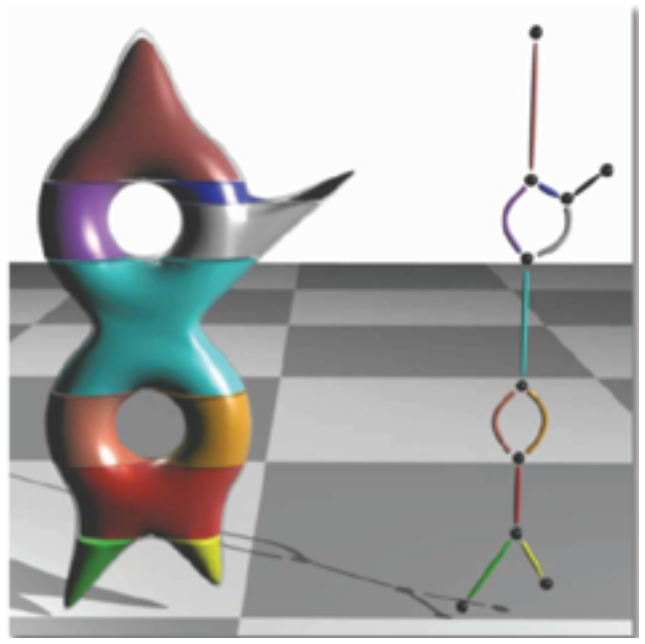
Guo et al. Classifying multi-attribute volume data with parallel coordinates. IEEE Pacific Vis 2011

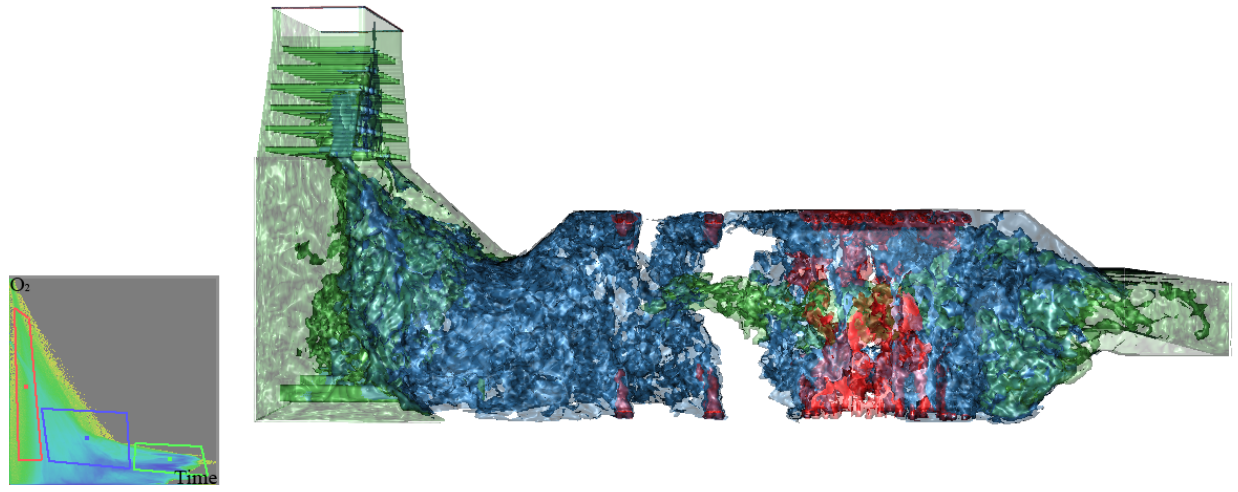


Lui et al. Multivariate Volume Visualization through Dynamic Projections. IEEE LDAV 2014.

Next lectures

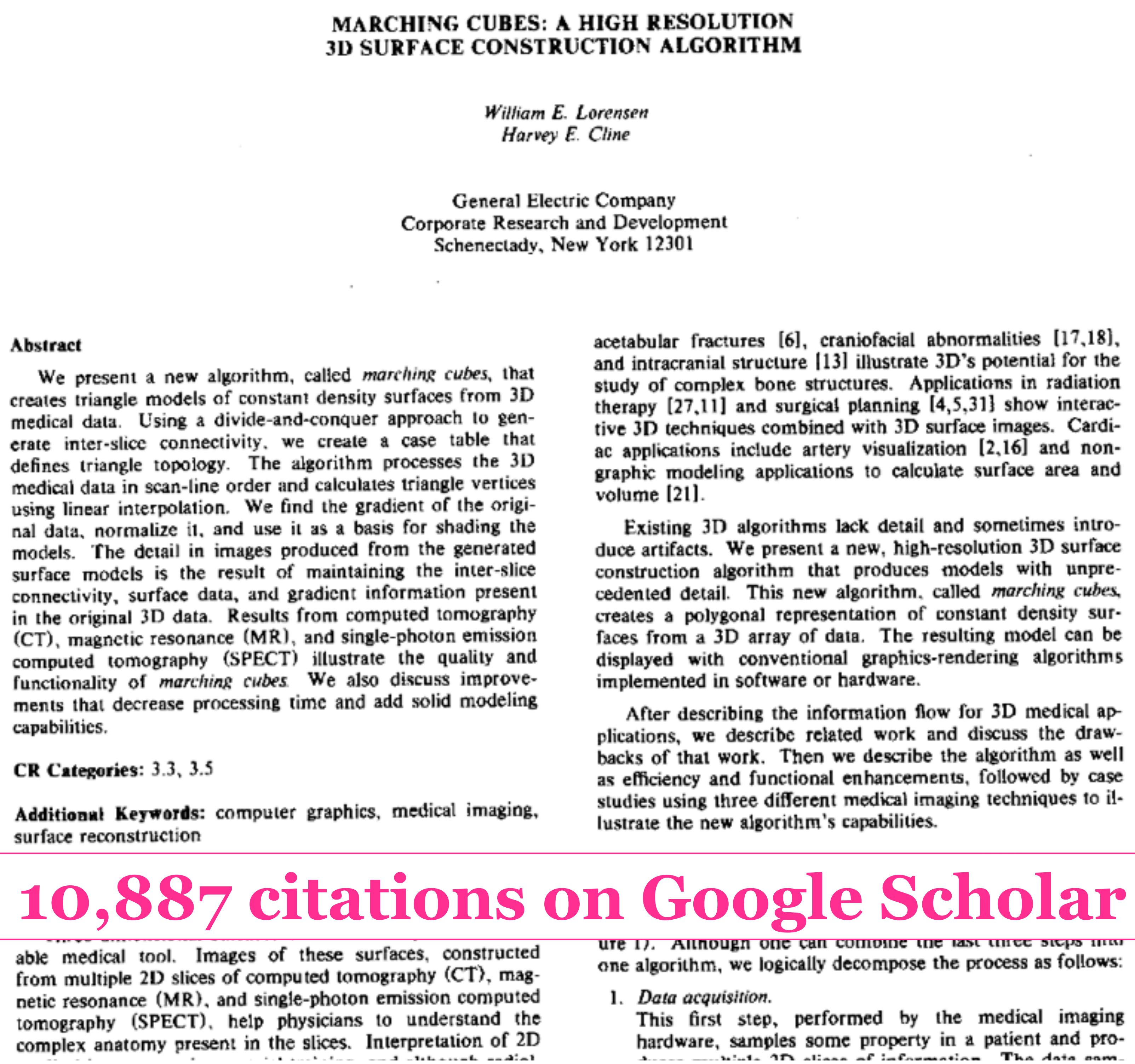






- 11-29: Isosurfaces and Topology
- 12-1: Flow, Vector and Tensor Fields

Reading



MARCHING CUBES: A HIGH RESOLUTION 3D SURFACE CONSTRUCTION ALGORITHM

William E. Lorensen
Harvey E. Cline

General Electric Company
Corporate Research and Development
Schenectady, New York 12301

Abstract

We present a new algorithm, called *marching cubes*, that creates triangle models of constant density surfaces from 3D medical data. Using a divide-and-conquer approach to generate inter-slice connectivity, we create a case table that defines triangle topology. The algorithm processes the 3D medical data in scan-line order and calculates triangle vertices using linear interpolation. We find the gradient of the original data, normalize it, and use it as a basis for shading the models. The detail in images produced from the generated surface models is the result of maintaining the inter-slice connectivity, surface data, and gradient information present in the original 3D data. Results from computed tomography (CT), magnetic resonance (MR), and single-photon emission computed tomography (SPECT) illustrate the quality and functionality of *marching cubes*. We also discuss improvements that decrease processing time and add solid modeling capabilities.

CR Categories: 3.3, 3.5

Additional Keywords: computer graphics, medical imaging, surface reconstruction

able medical tool. Images of these surfaces, constructed from multiple 2D slices of computed tomography (CT), magnetic resonance (MR), and single-photon emission computed tomography (SPECT), help physicians to understand the complex anatomy present in the slices. Interpretation of 2D

acetabular fractures [6], craniofacial abnormalities [17,18], and intracranial structure [13] illustrate 3D's potential for the study of complex bone structures. Applications in radiation therapy [27,11] and surgical planning [4,5,31] show interactive 3D techniques combined with 3D surface images. Cardiac applications include artery visualization [2,16] and nongraphic modeling applications to calculate surface area and volume [21].

Existing 3D algorithms lack detail and sometimes introduce artifacts. We present a new, high-resolution 3D surface construction algorithm that produces models with unprecedented detail. This new algorithm, called *marching cubes*, creates a polygonal representation of constant density surfaces from a 3D array of data. The resulting model can be displayed with conventional graphics-rendering algorithms implemented in software or hardware.

After describing the information flow for 3D medical applications, we describe related work and discuss the drawbacks of that work. Then we describe the algorithm as well as efficiency and functional enhancements, followed by case studies using three different medical imaging techniques to illustrate the new algorithm's capabilities.

one algorithm, we logically decompose the process as follows:

1. *Data acquisition.*
This first step, performed by the medical imaging hardware, samples some property in a patient and pro-

10,887 citations on Google Scholar

www.sci.utah.edu