Scientific Visualization: Volume Rendering

CS 6630, Fall 2015 — Alex Lex
Aaron Knoll, guest lecturer

Recap from last sci-vis lecture

- Grids
- Unstructured and structured data
- Direct and indirect workflows
- Interpolation

Today

- Two more handy formulae for interpolation
- Computer graphics
  - Rasterization and ray tracing
  - Shading
  - Alpha blending
- Volume ray casting (HW6)
  - Theory
  - Code!
- Transfer functions and transfer function design (HW6)

Interpolation

Bilinear interpolation

Just 3 linear interpolations

```
#define lerp(a,b,t)  (1-t) * a + t*b

//Given voxel vertices vXX and the (x,y) position within the voxel [0,1]^2

//lerp along y direction
float v00_y = lerp(v00, v01, y);
float v10_y = lerp(v10, v11, y);

//lerp along x direction
return lerp(v00_y, v10_y, x);
```

Trilinear interpolation

Just 7 linear interpolations!

```
#define lerp(a,b,t)  (1-t) * a + t*b

//Given voxel vertices vXXX and the (x,y,z) position within the voxel [0,1]^3

//lerp along z direction.
float v000_z = lerp(v000, v001, z);
float v010_z = lerp(v010, v011, z);
float v100_z = lerp(v100, v101, z);
float v110_z = lerp(v110, v111, z);

//lerp along y direction
float v000_yz = lerp(v000_z, v010_z, y);
float v100_yz = lerp(v100_z, v110_z, y);

//lerp along x direction
return lerp(v000_yz, v100_yz, x);
```

Easier formula for trilinear interpolation (3D)

$$f(x,y,z) = \sum_{i,j,k=\{0,1\}} x_i y_j z_k \; v_{ijk}$$

Where $x_0 = i+1-x, x_1 = x-i$, ditto for y and z

And v_{ijk} is the value of the voxel at that vertex.

Even easier formula for general interpolation (3D)

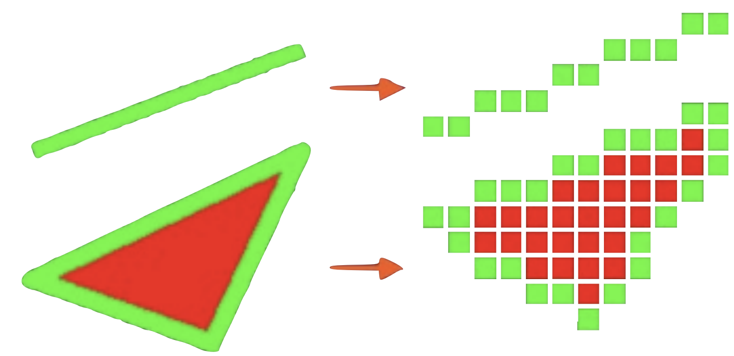
$$f(x, y, z) = \sum_{i,j,k} B_i(x) B_j(y) B_k(z) v_{ijk}$$

Where the B(x) is a general basis function, v is the voxel.

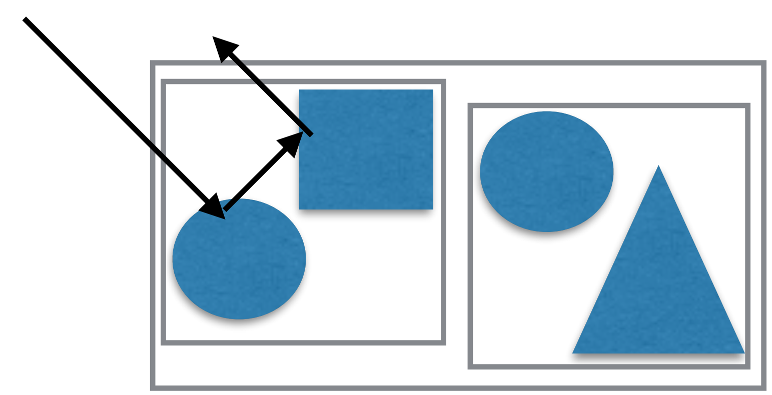
(3D) Computer Graphics

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!

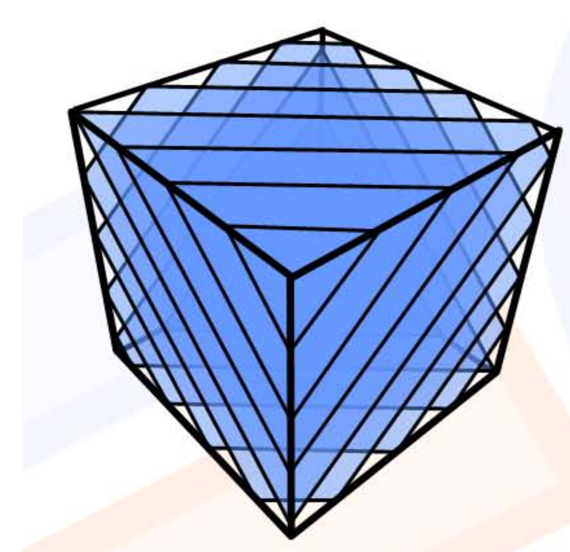


- Our "third way" in HW6:
  Hack the GPU fragment shader to do volume ray casting!
- Volume rendering can be implemented either via ray tracing (sampling along the ray) or rasterization (with textured proxy geometry)

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" in the rasterization pipeline:
  - slicing
  - splatting and other proxy geometry
  - Our method in HW6:
    Use the GPU fragment shader to do volume ray casting!
  - Ray tracing (or at least ray casting) increasingly common…



3D texture slicing

Rusinkiewicz & Levoy, "QSplat", Siggraph 2000

Rasterization

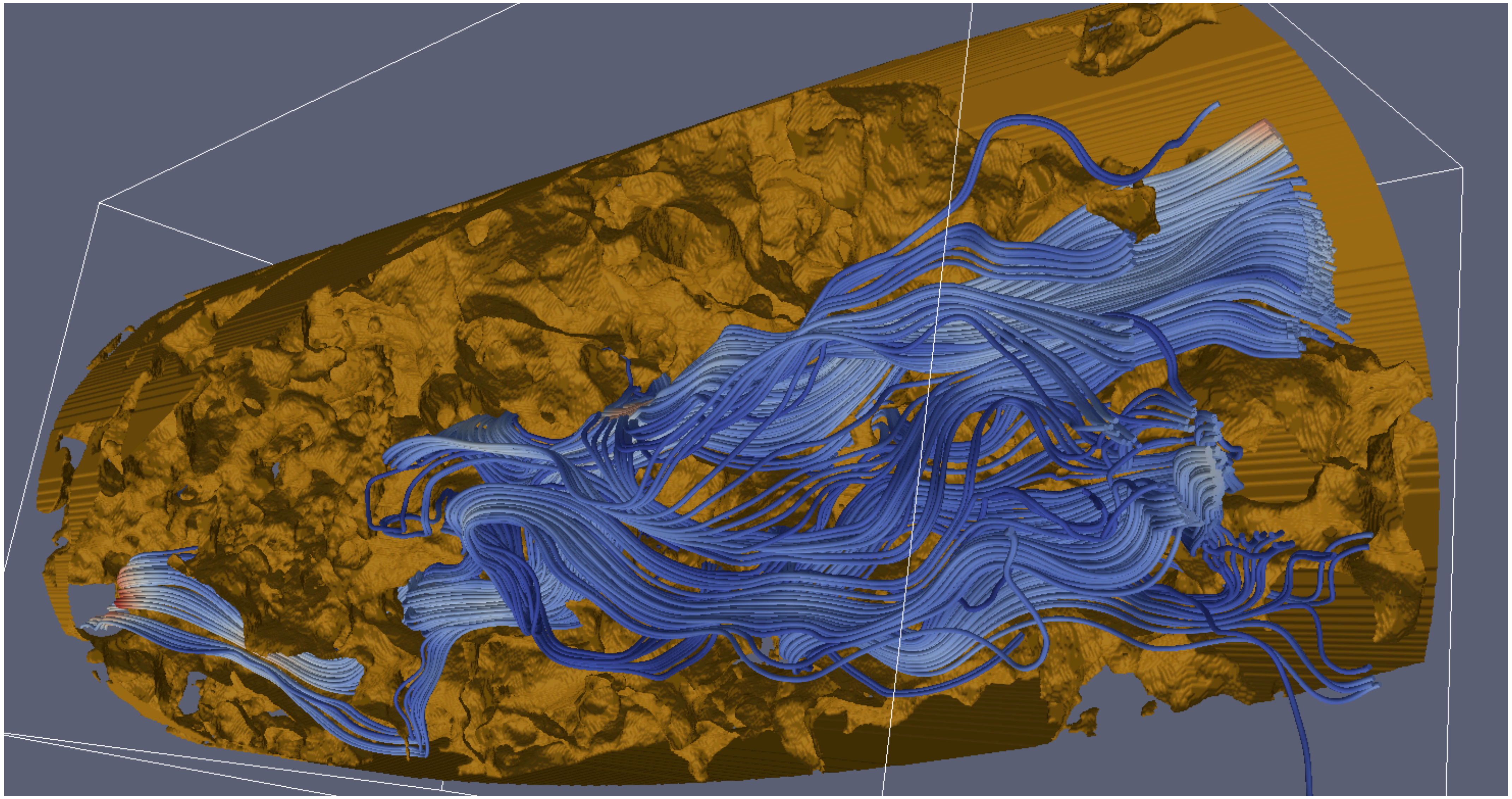


Image: Carson Brownlee, TACC. Data: Michael Sukop, FIU

www.sci.utah.edu

Ray tracing

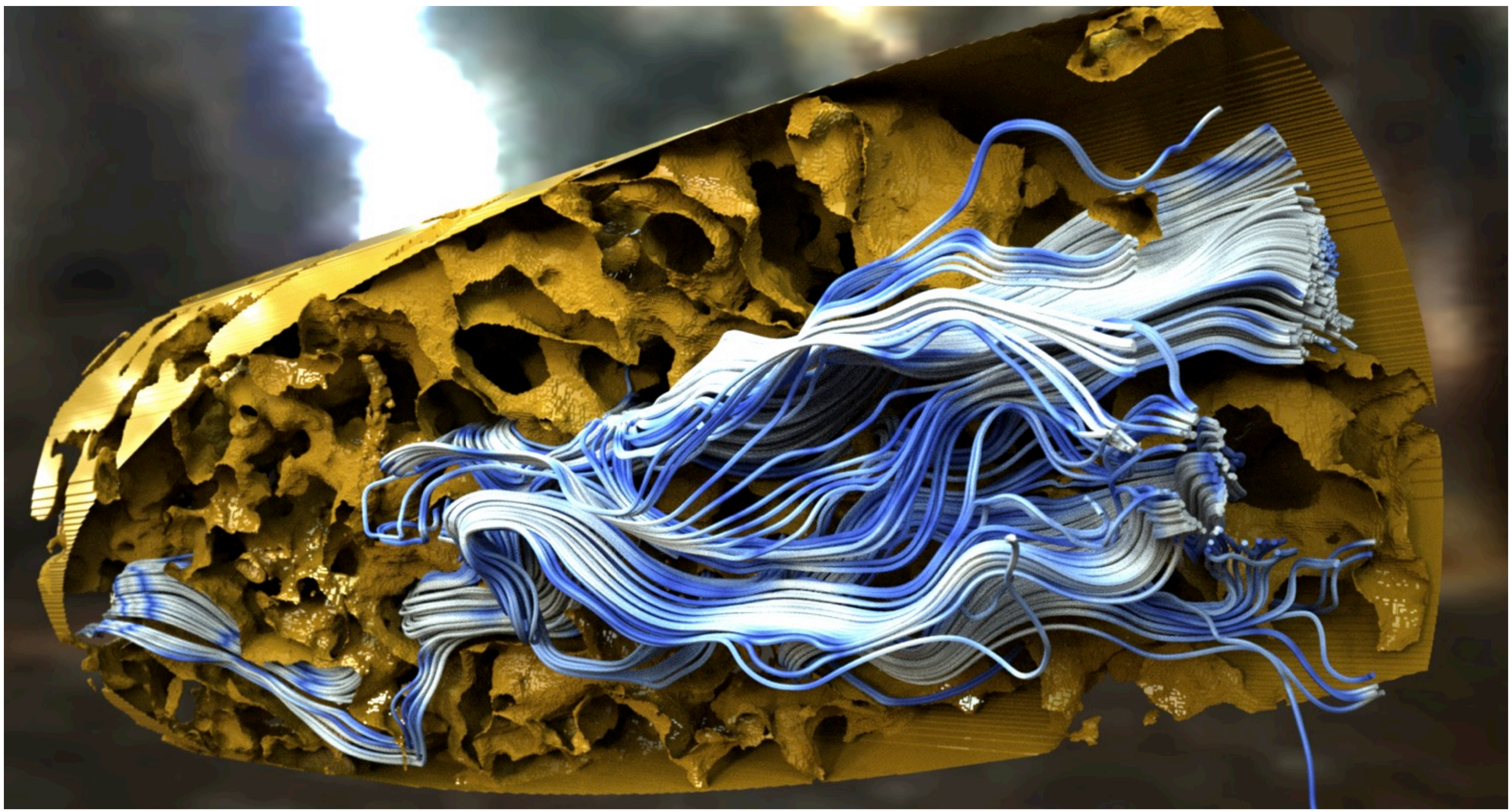
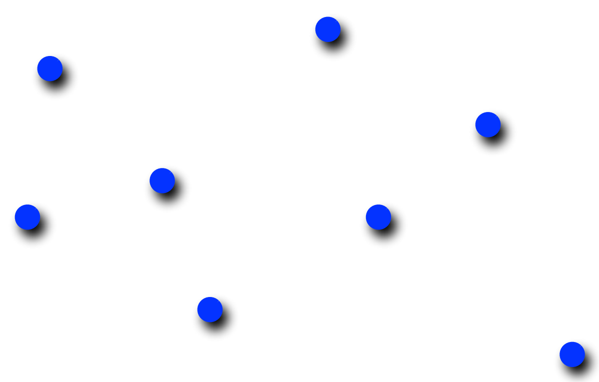


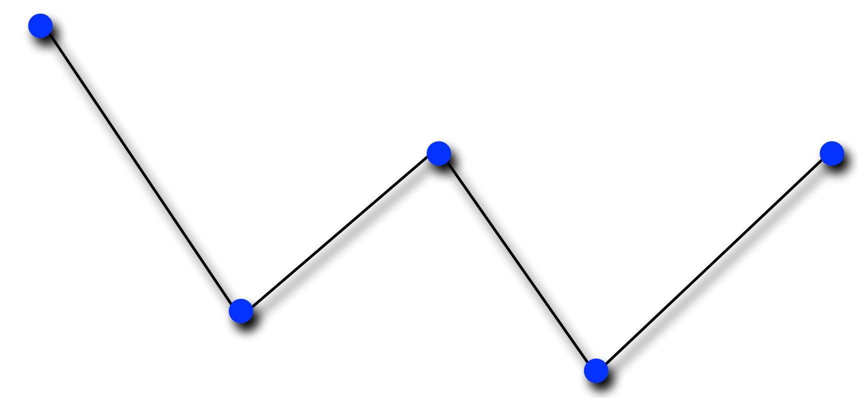
Image: Carson Brownlee, TACC. Data: Michael Sukop, FIU

www.sci.utah.edu

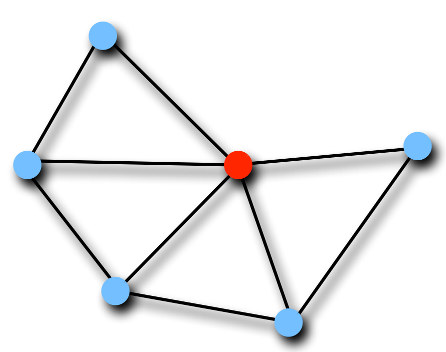
Graphics Primitives



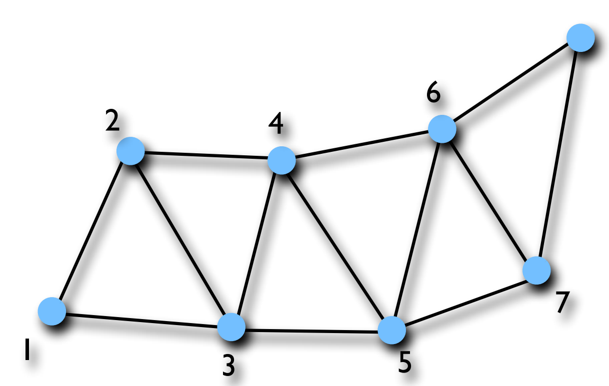
Points



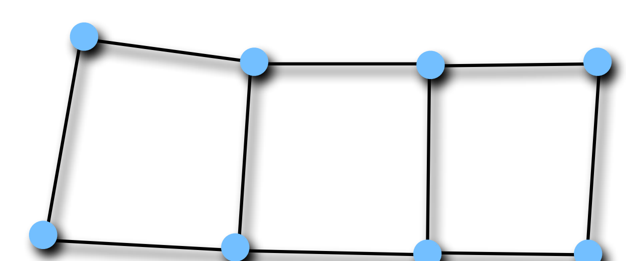
Lines



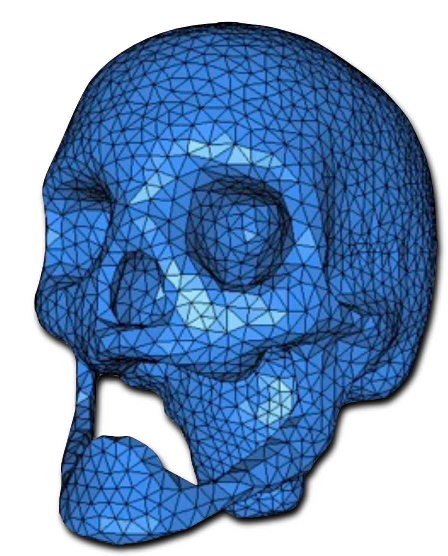
triangle fan



triangle strip



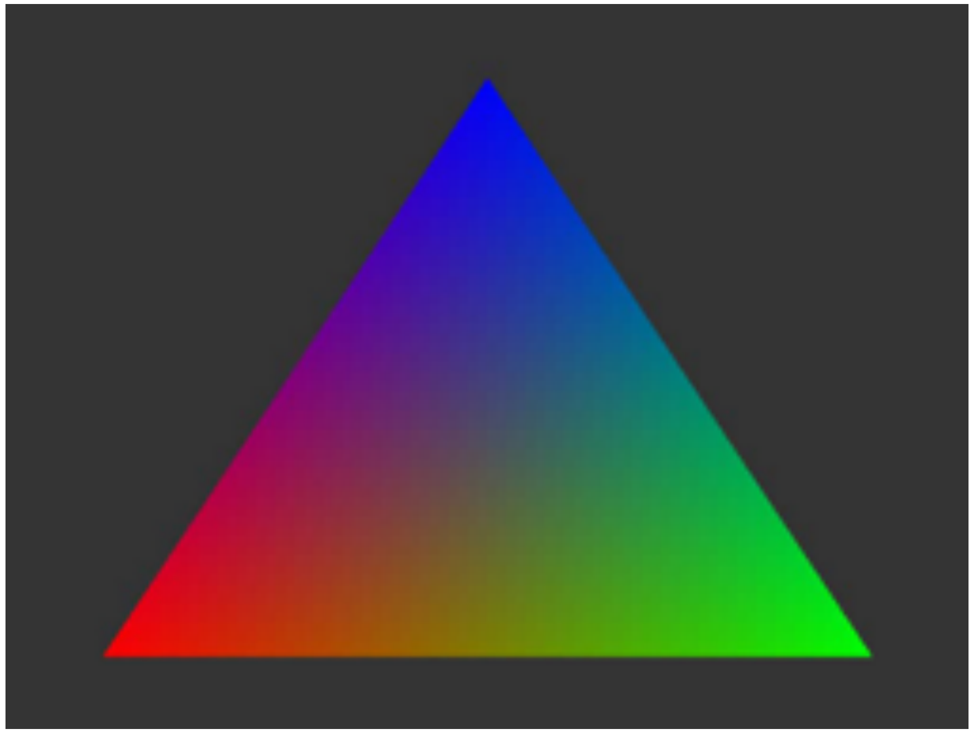
quad strip



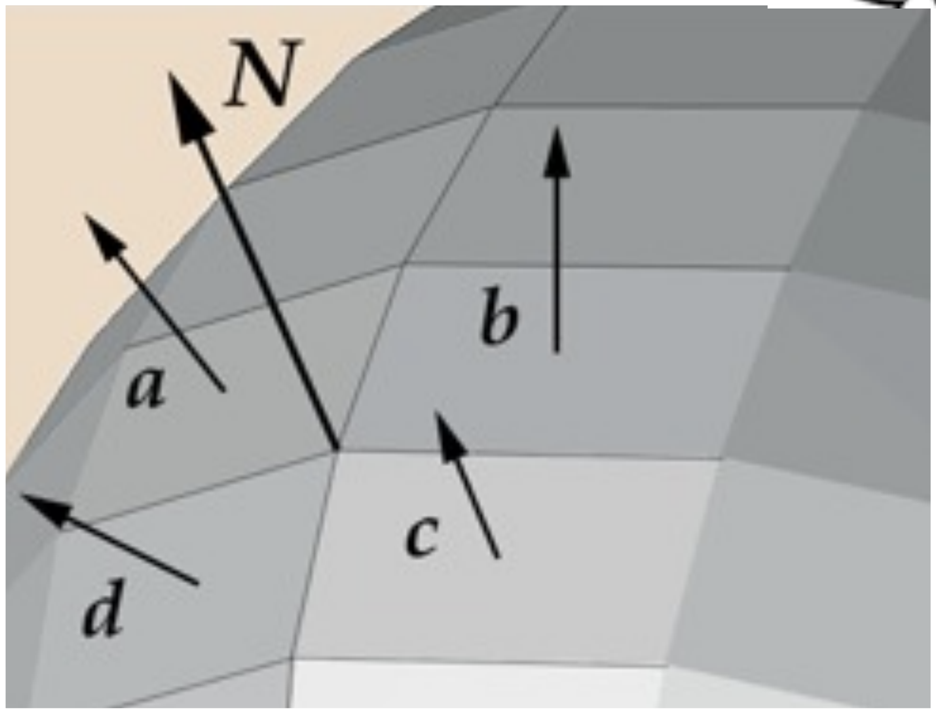
Surfaces

Primitive Attributes

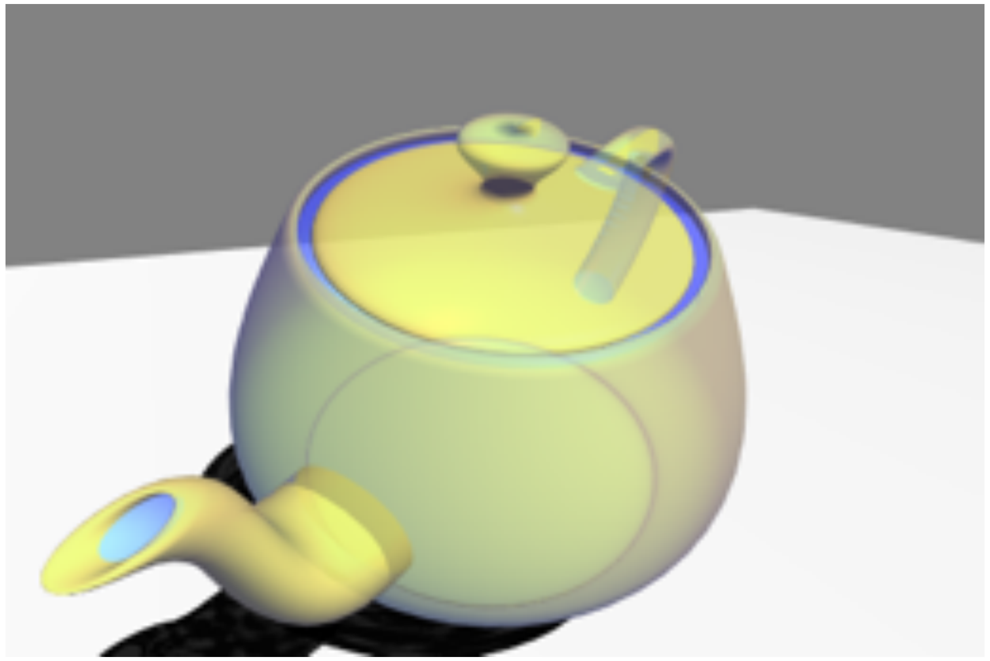
Color

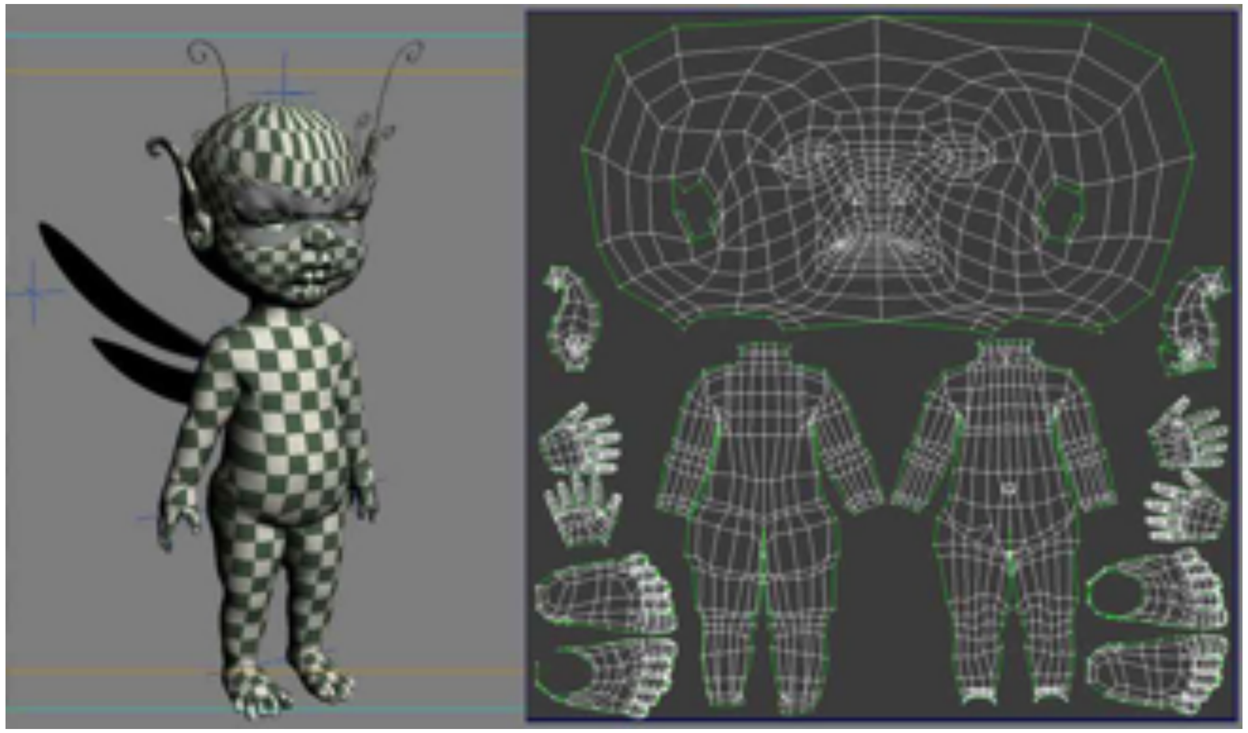


Normal



Opacity

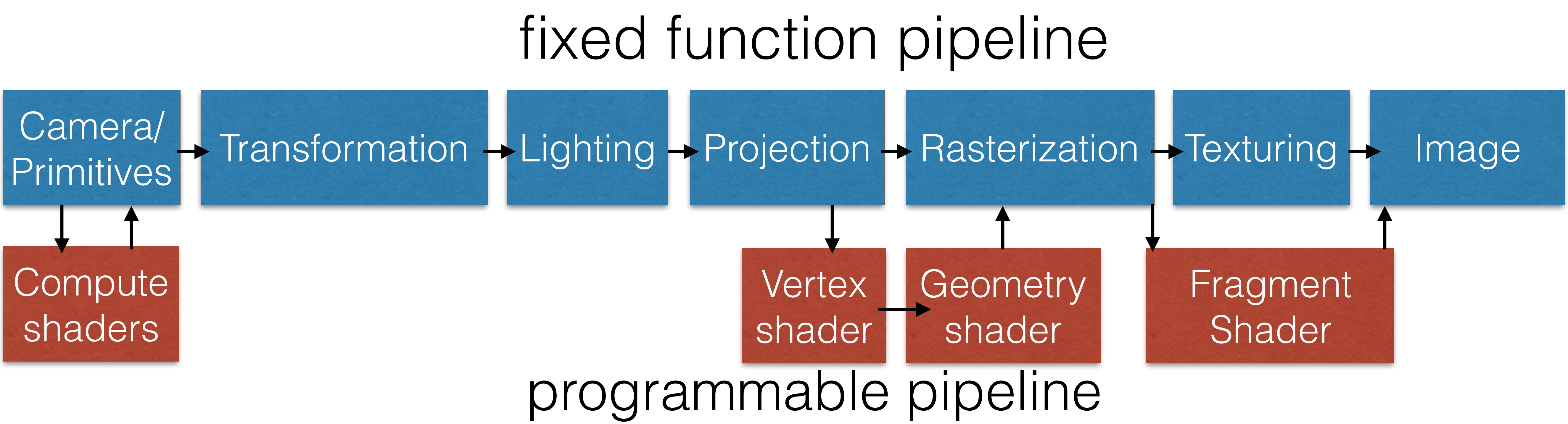




Texture coordinates

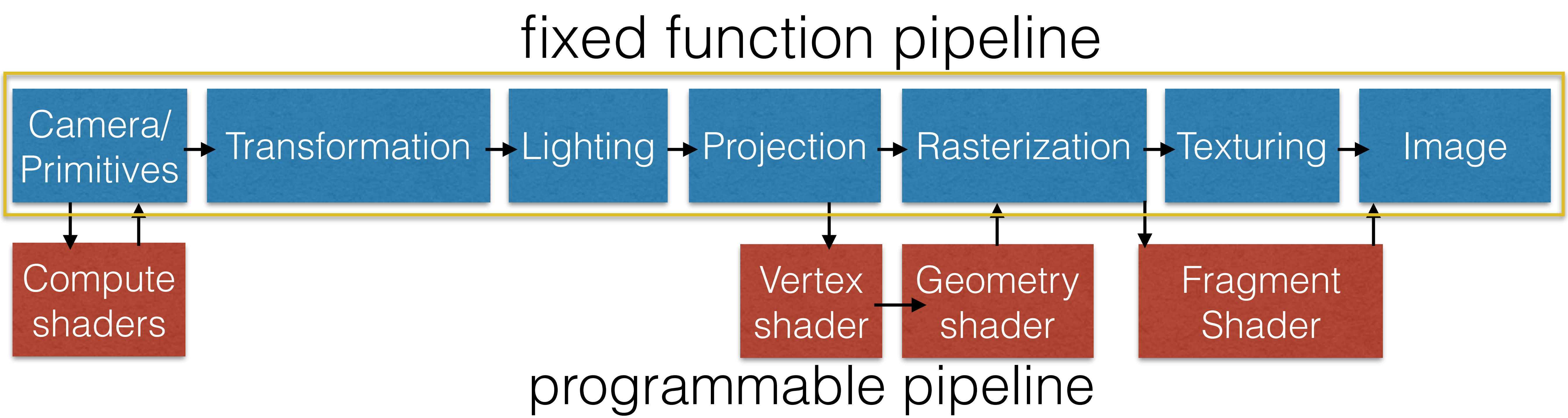
slides: IU Purdue

The rasterization pipeline



The rasterization pipeline

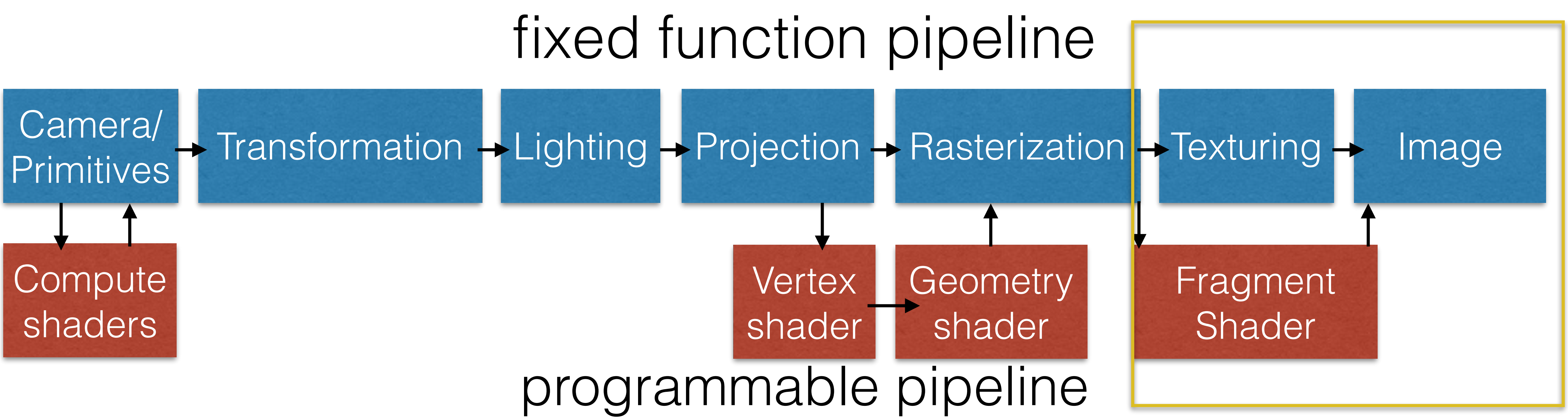
(e.g., indirect visualization with rasterization)



www.sci.utah.edu

The rasterization pipeline

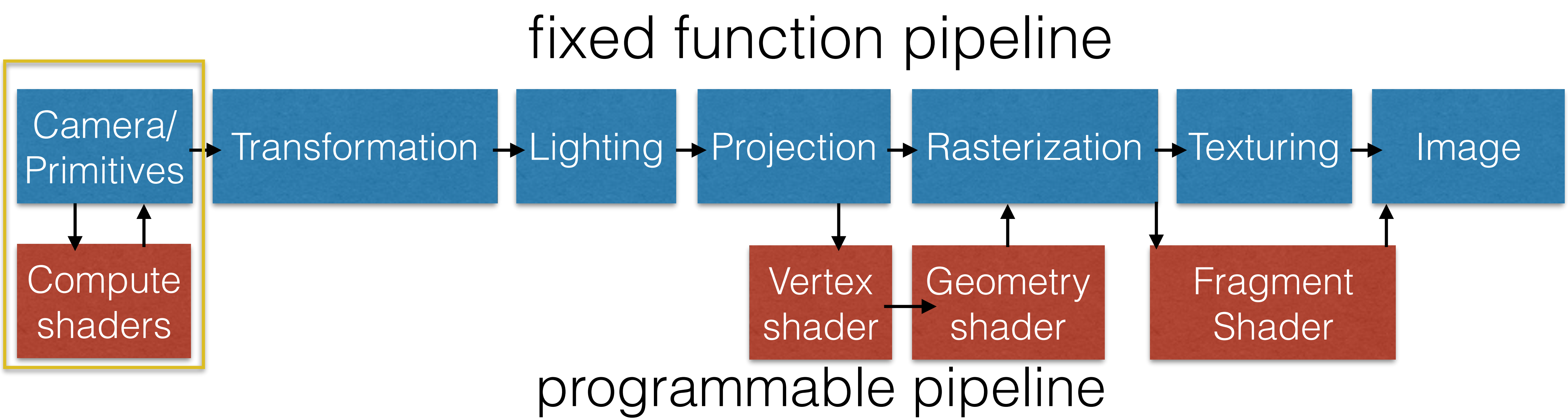
(e.g. direct visualization GPU ray casting)



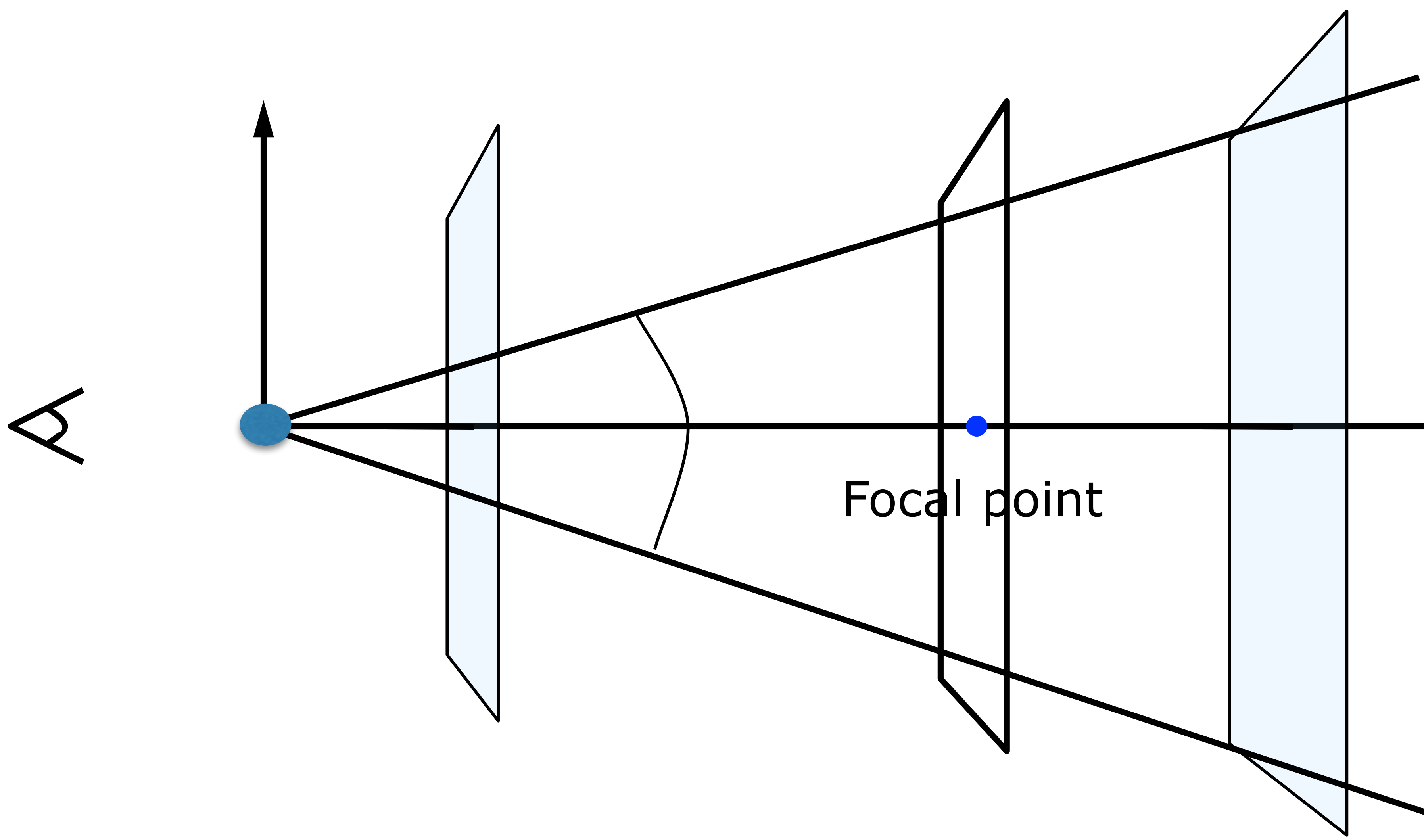
www.sci.utah.edu

The rasterization pipeline

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

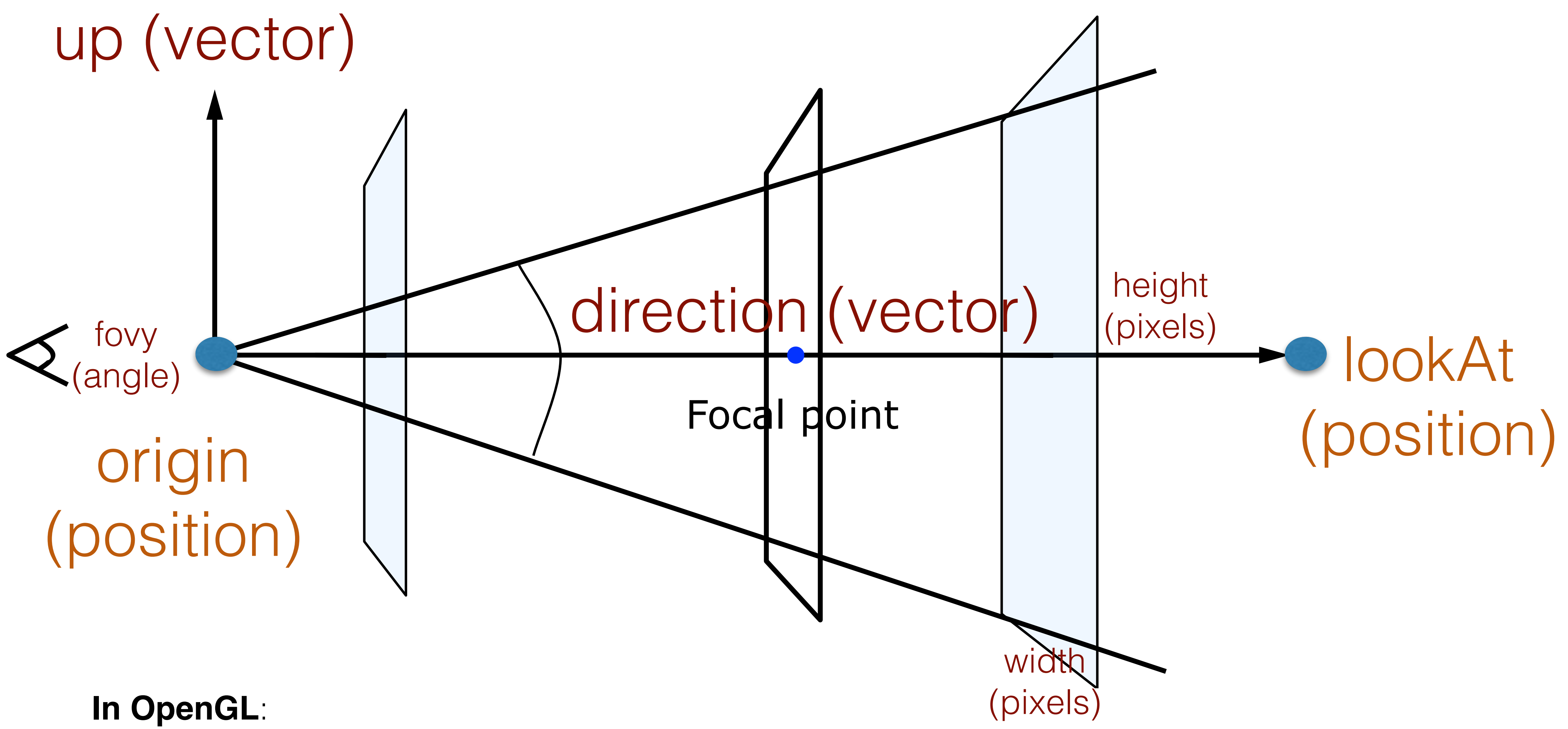


Camera



- 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;

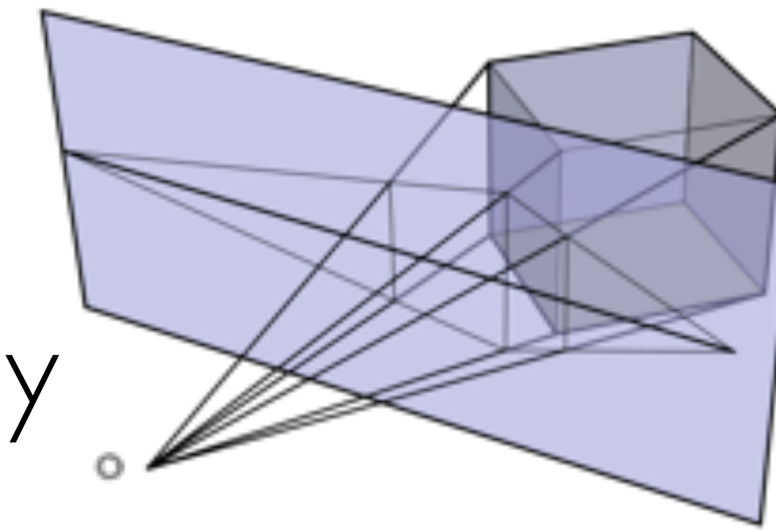
glViewport (width, height);
gluPerspective(width, height, fov, near, far);
gluLookAt(origin, direction, up);

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

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

Projections

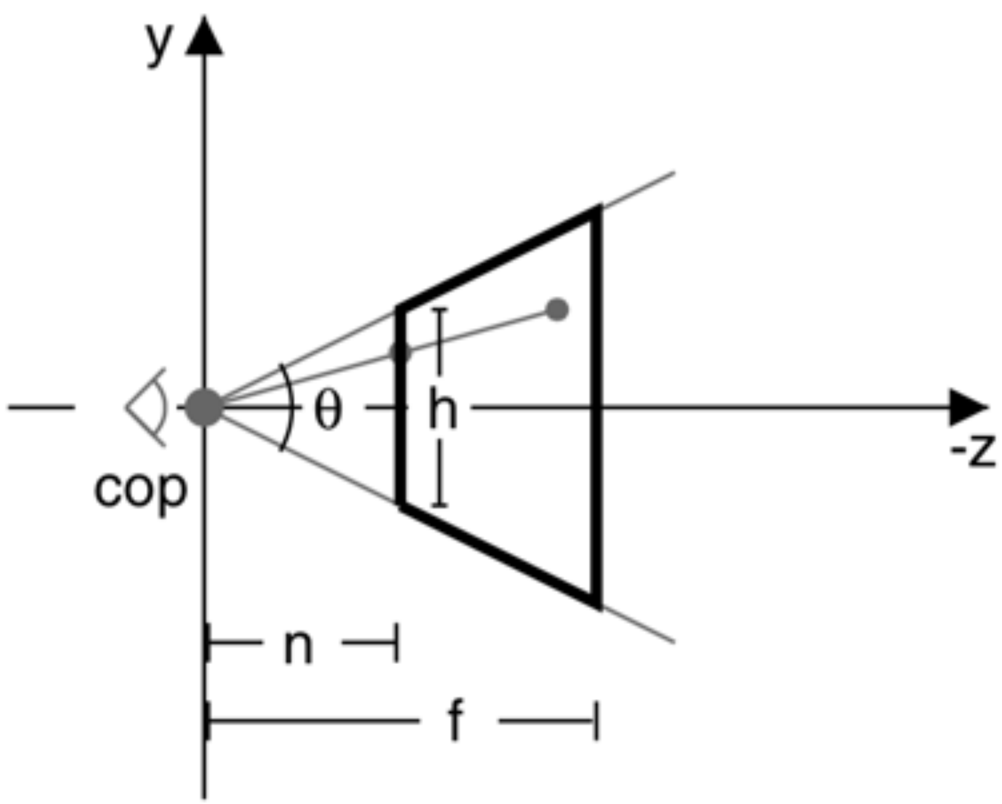
Perspective projection



- 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*

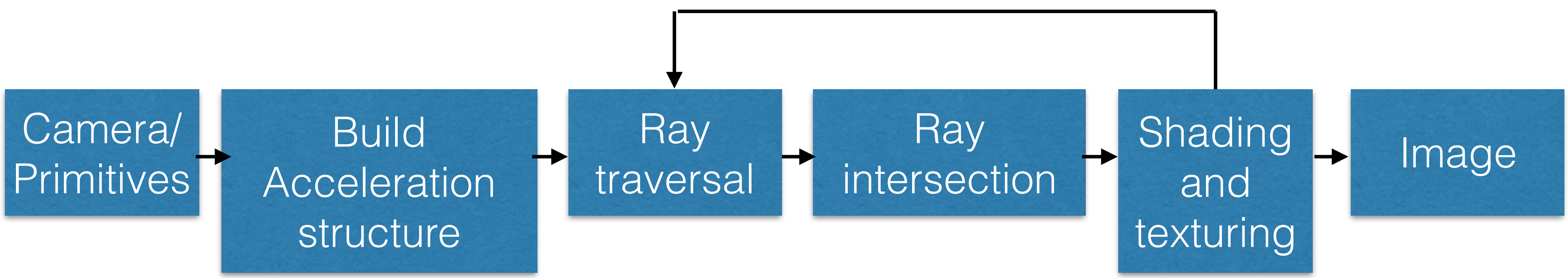


$$M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (n+f)/n & -f \\ 0 & 0 & 1/n & 0 \end{bmatrix}.$$

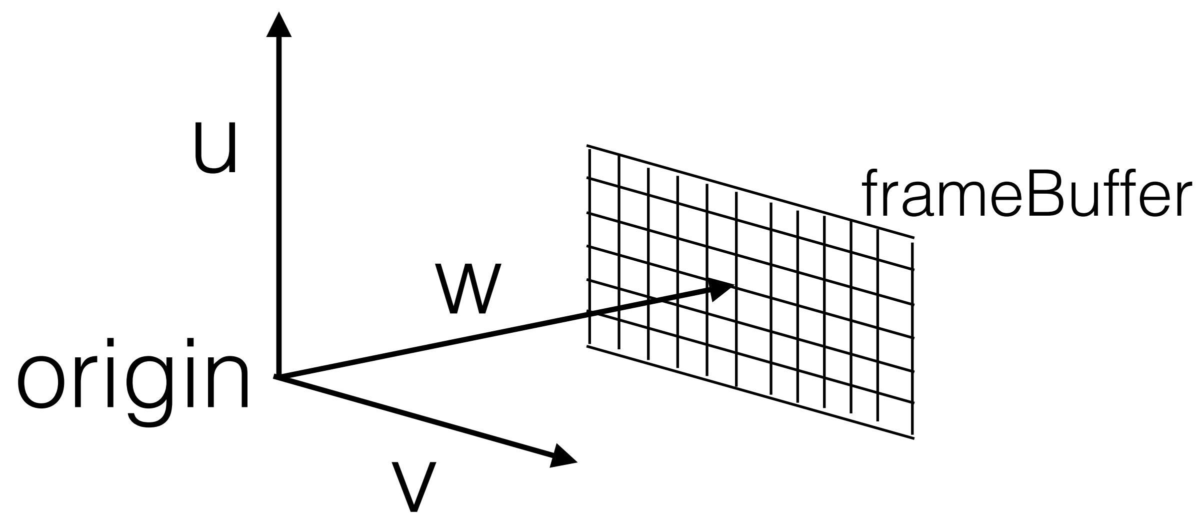
$$M_p \bar{\mathbf{x}} = \begin{bmatrix} x \\ y \\ z(\frac{n+f}{n}) - f \\ z/n \end{bmatrix}$$

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);

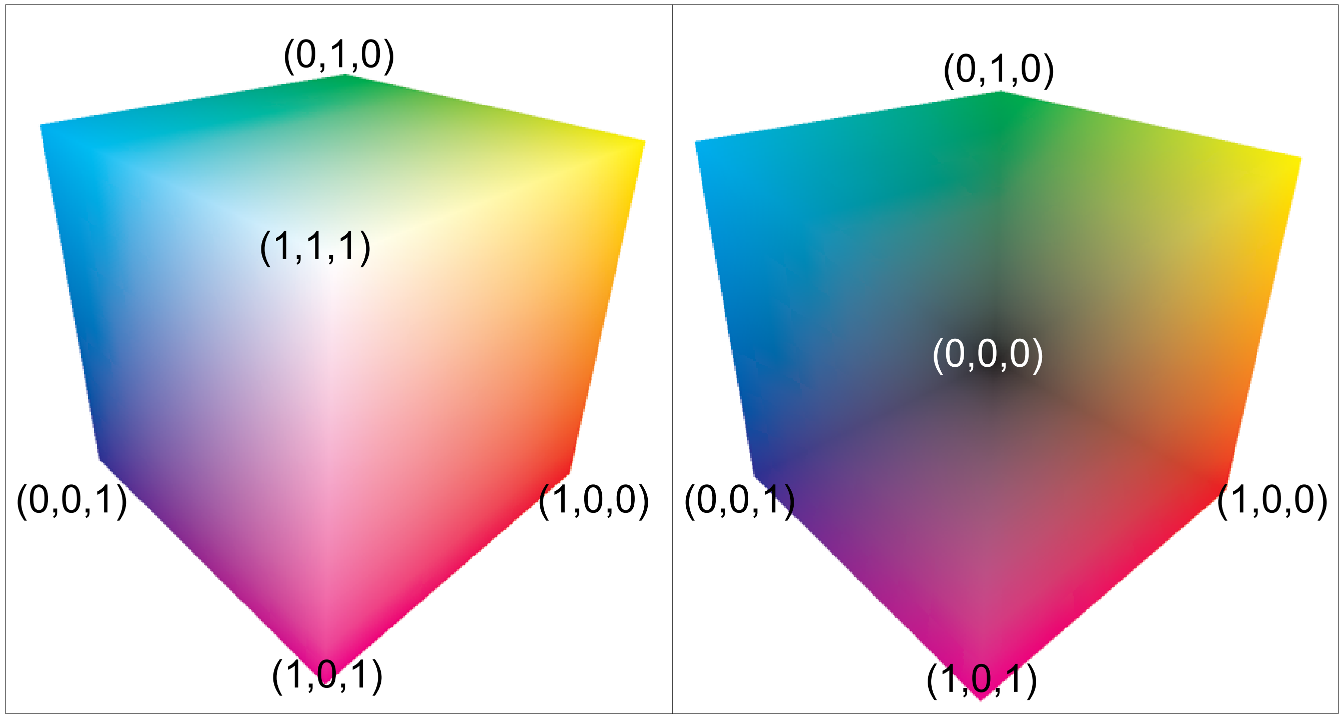
  float tanThetaOver2 = tanf(fovy * .5 * PI / 180);
  float aspect = width / height;

  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

Two-pass rasterization for ray generation (how we do it in HW6 — already done for you!)



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

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

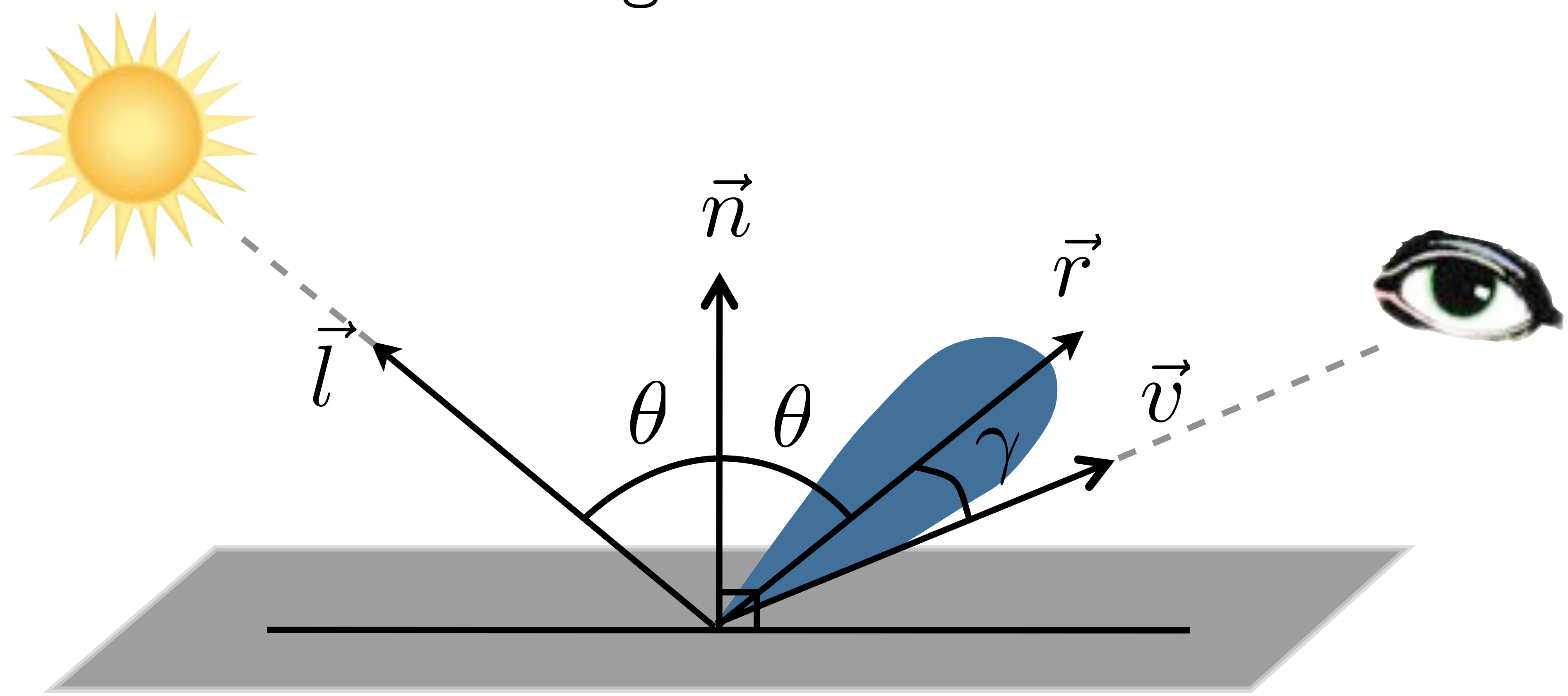
$$I = k_s I_s + k_d I_d + k_a I_a$$

relative contributions
(material specific)

Shading

Phong lighting model: Specular Reflection

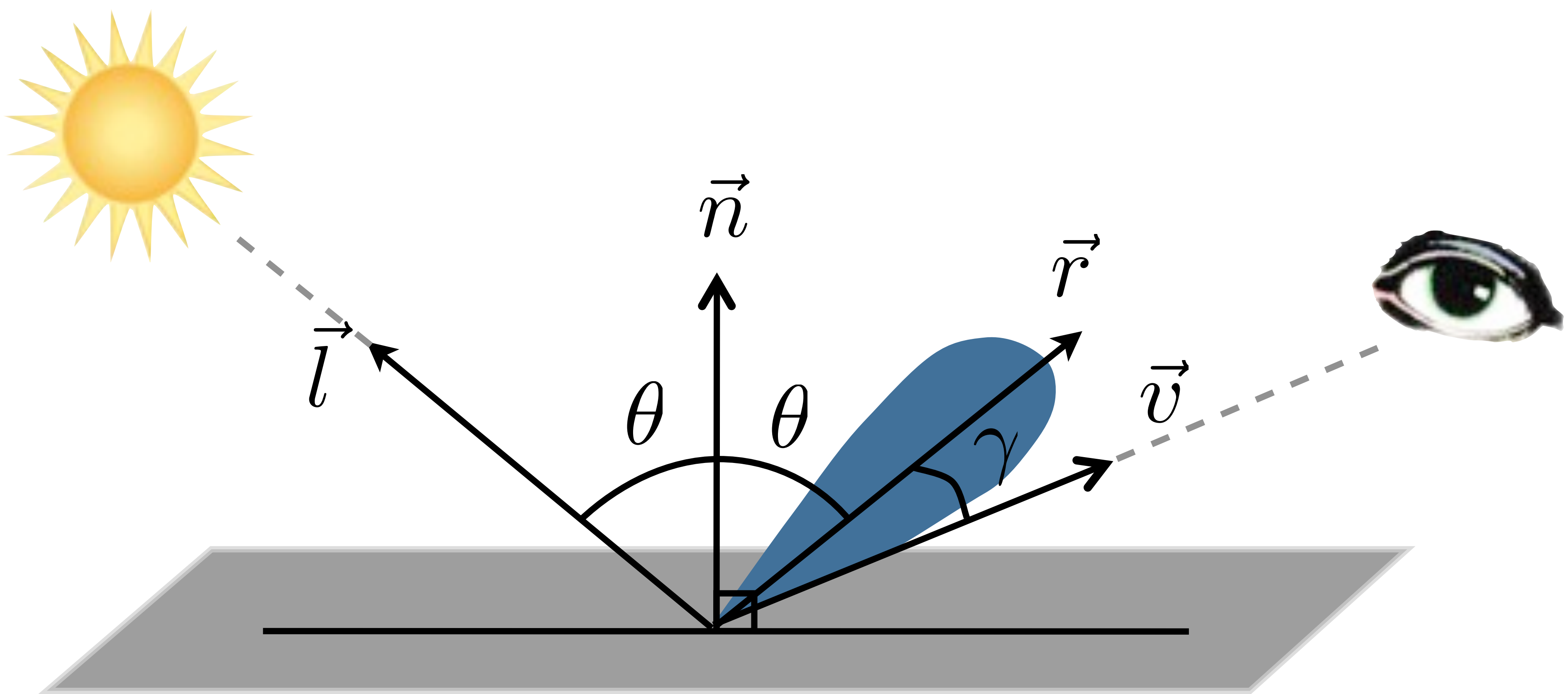
- mirror-like surfaces
- Specular reflection depends on position of the observer relative to light source <u>and</u> surface normal



Shading

Phong lighting model: Specular Reflection

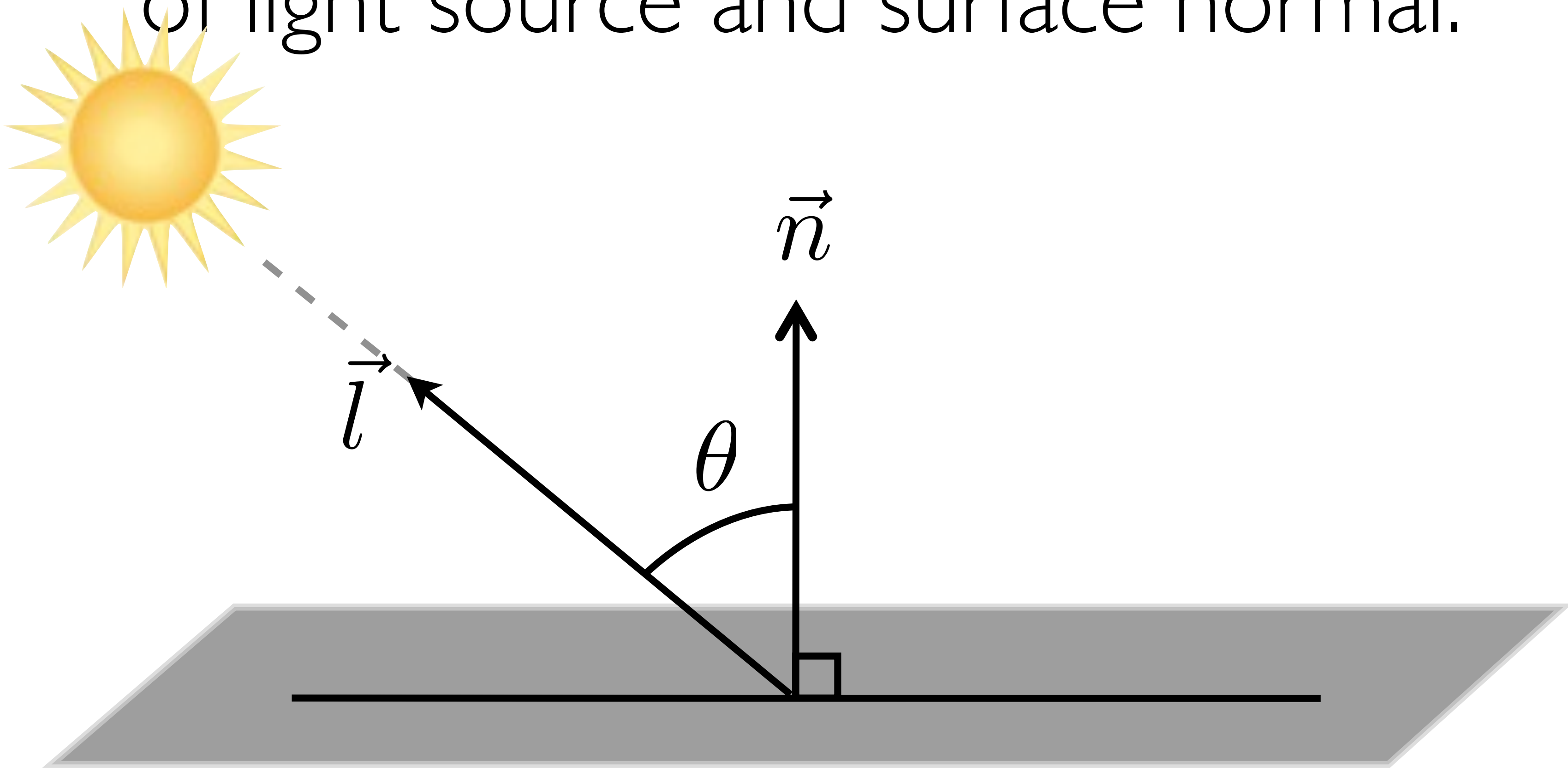
$$\boxed{I_s = I_i \cos^n \gamma = I_i \cos^n < \vec{r}, \vec{v} >}$$

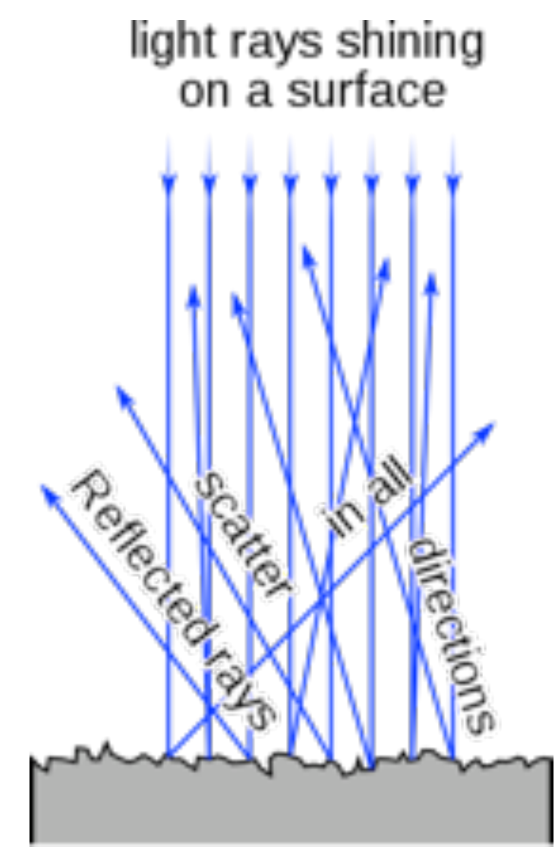


Shading

Phong lighting model: Diffuse Reflection

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

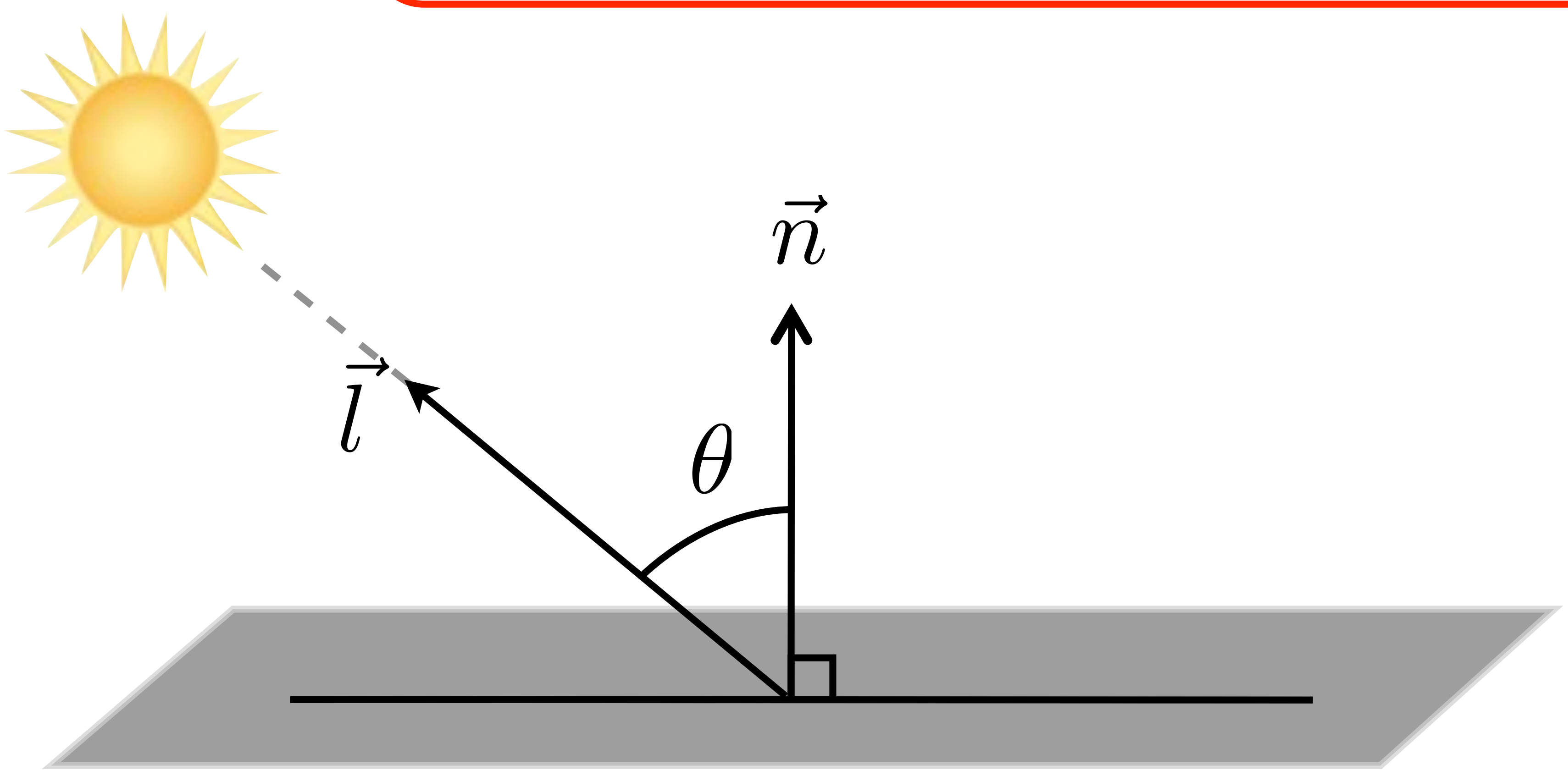


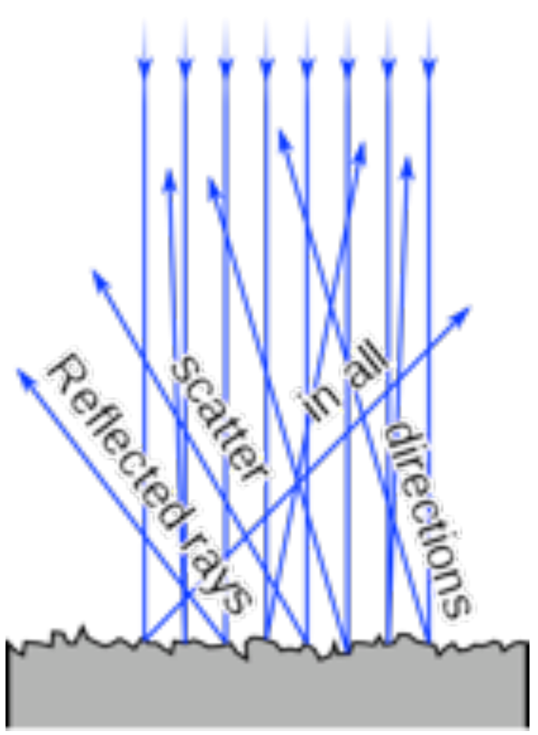


Shading

Phong lighting model: Diffuse Reflection

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





Shading

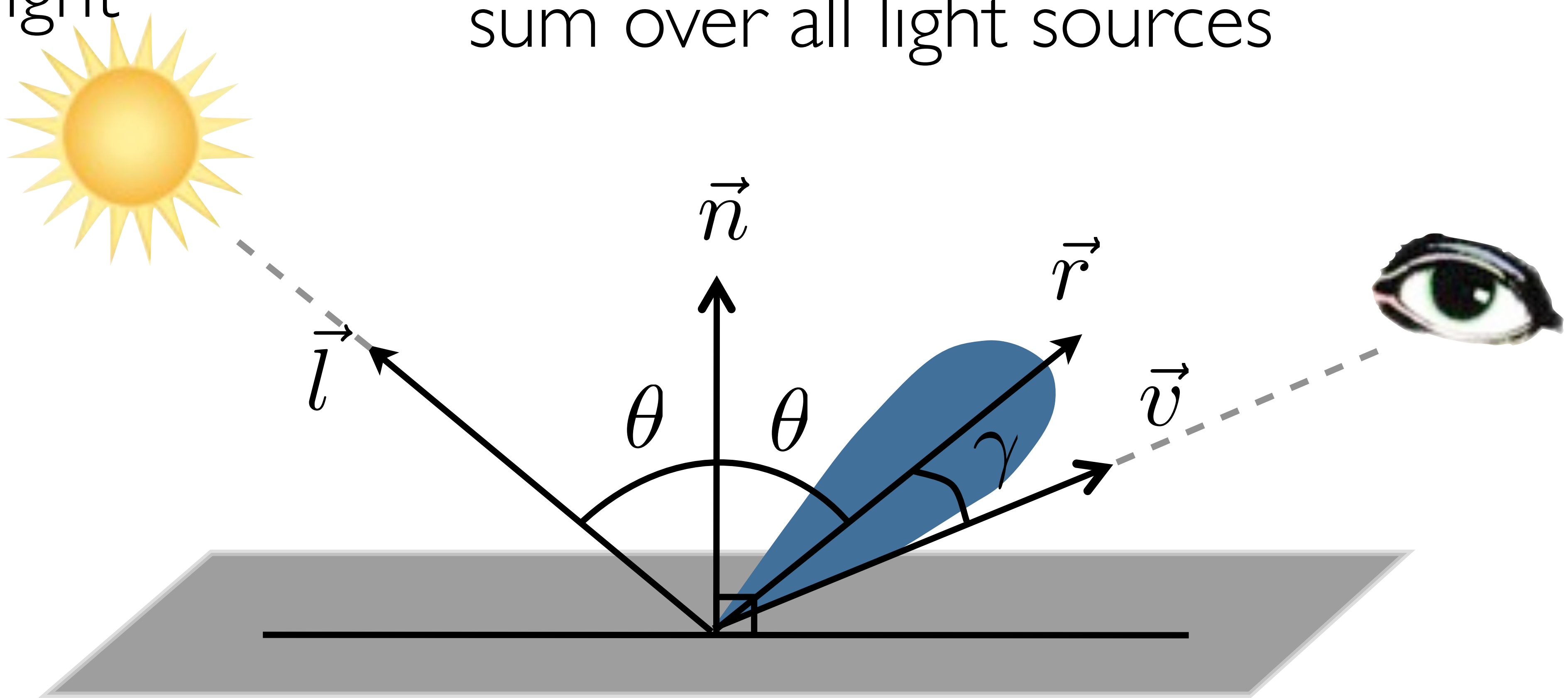
Phong lighting model

per color channel

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

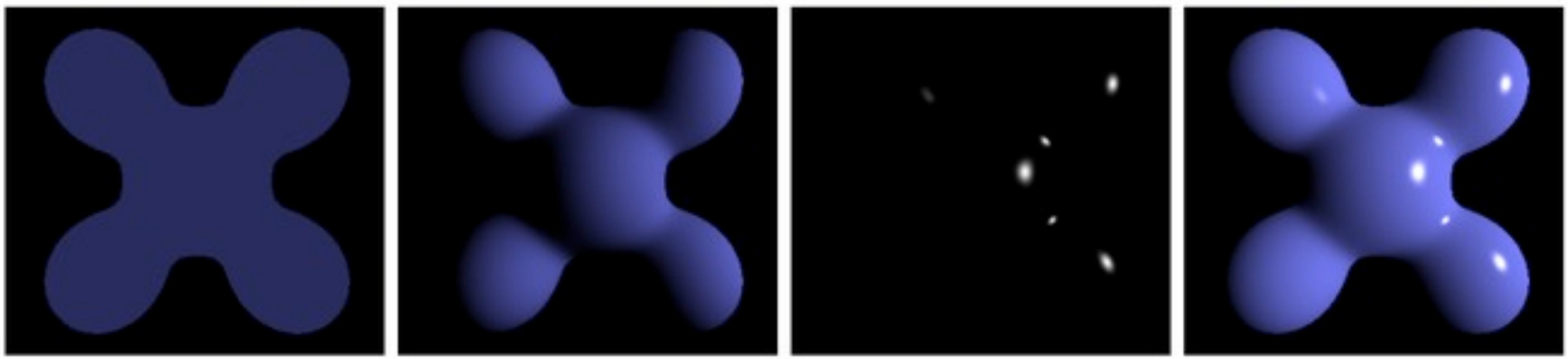
ambient light

sum over all light sources



Shading

Phong lighting model



Ambient + Diffuse + Specular = Phong Reflection

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!

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)

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

This is the only one you need to remember for vis.

www.sci.utah.edu

Alpha blending

Fundamental Algorithms

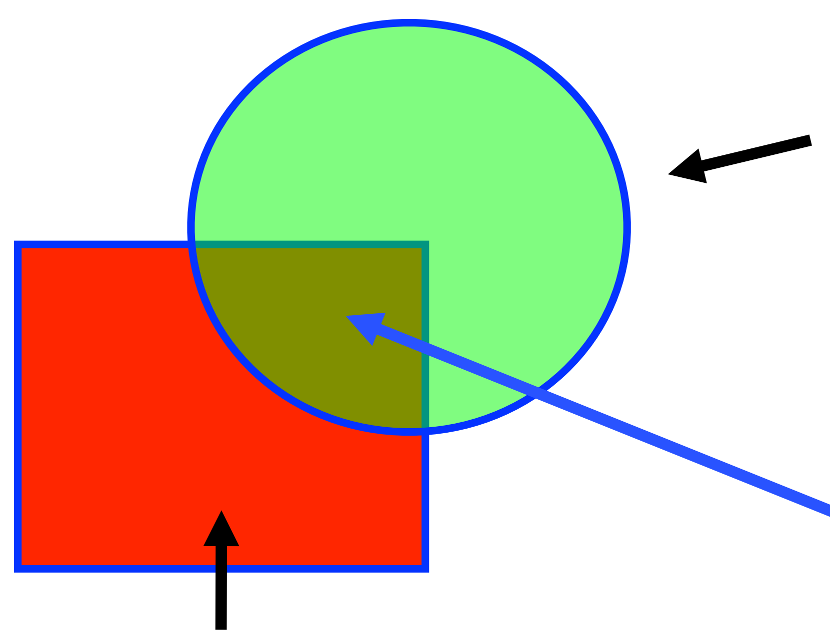
Alpha blending / compositing

- 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)
{
  accumulatedColor += (1.0 - accumulatedAlpha) * colorSample * alphaSample;
  accumulatedAlpha += alphaSample;
}
```

Hint: volume rendering is just doing this over and over again in a loop!

Volume rendering

Volume ray casting

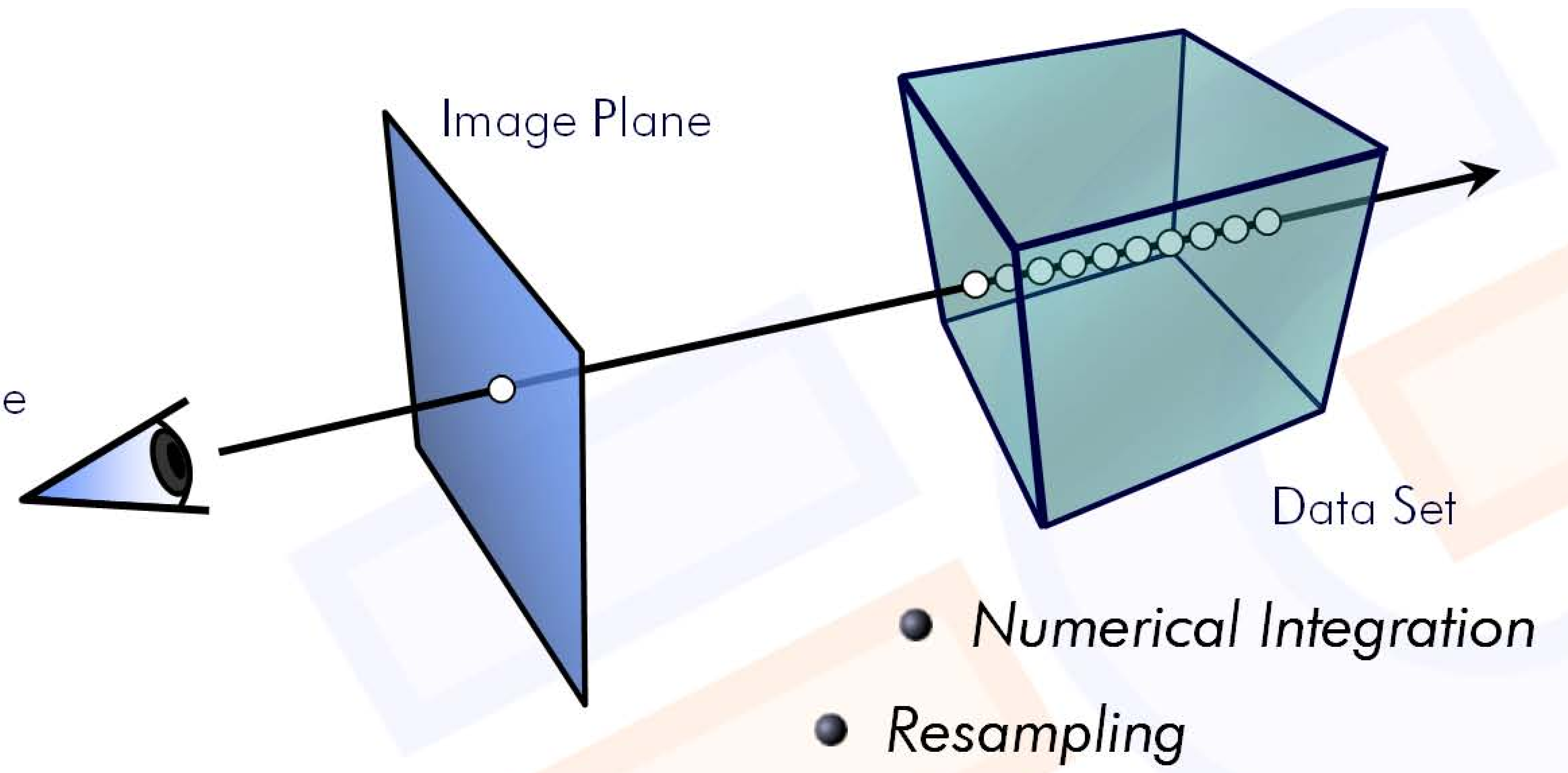
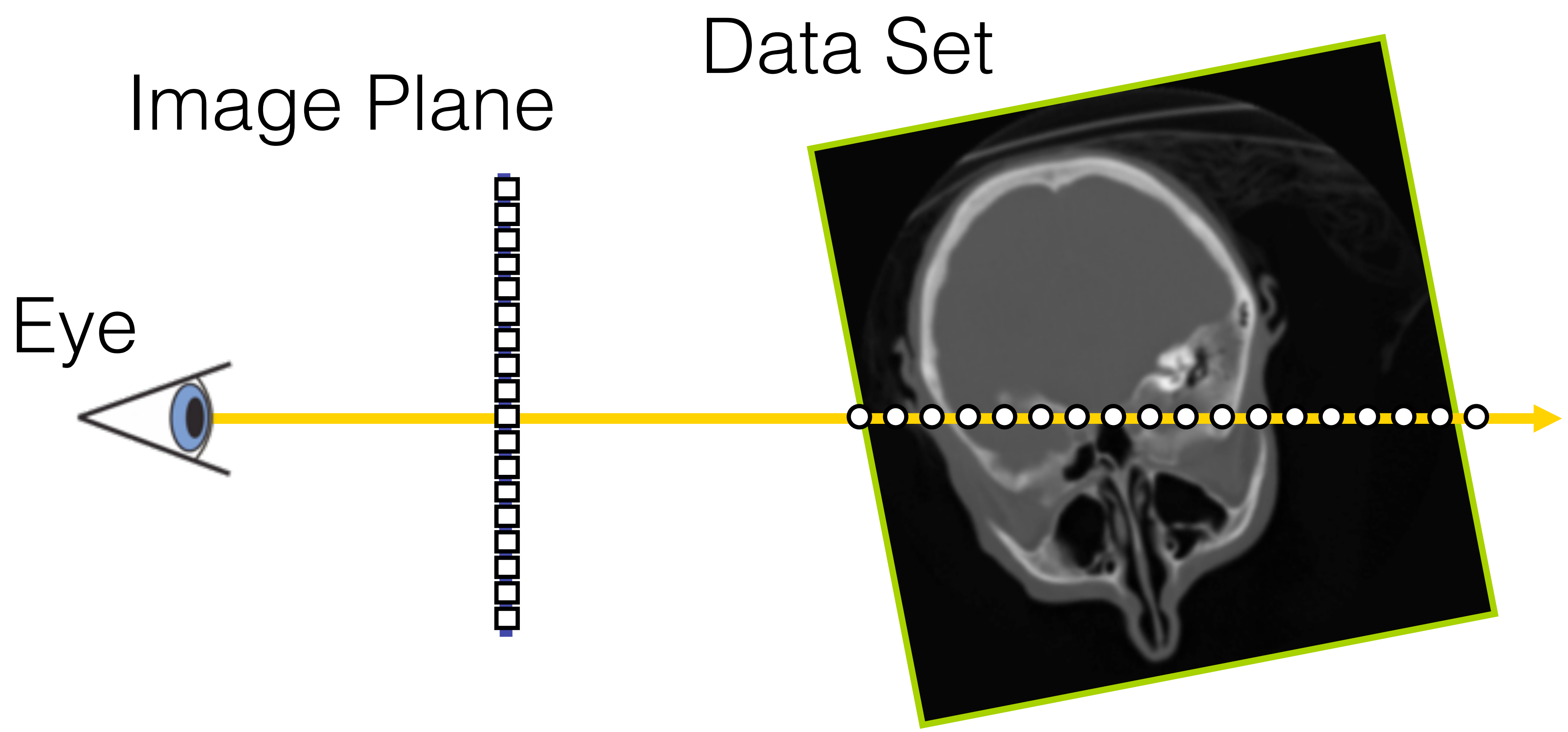
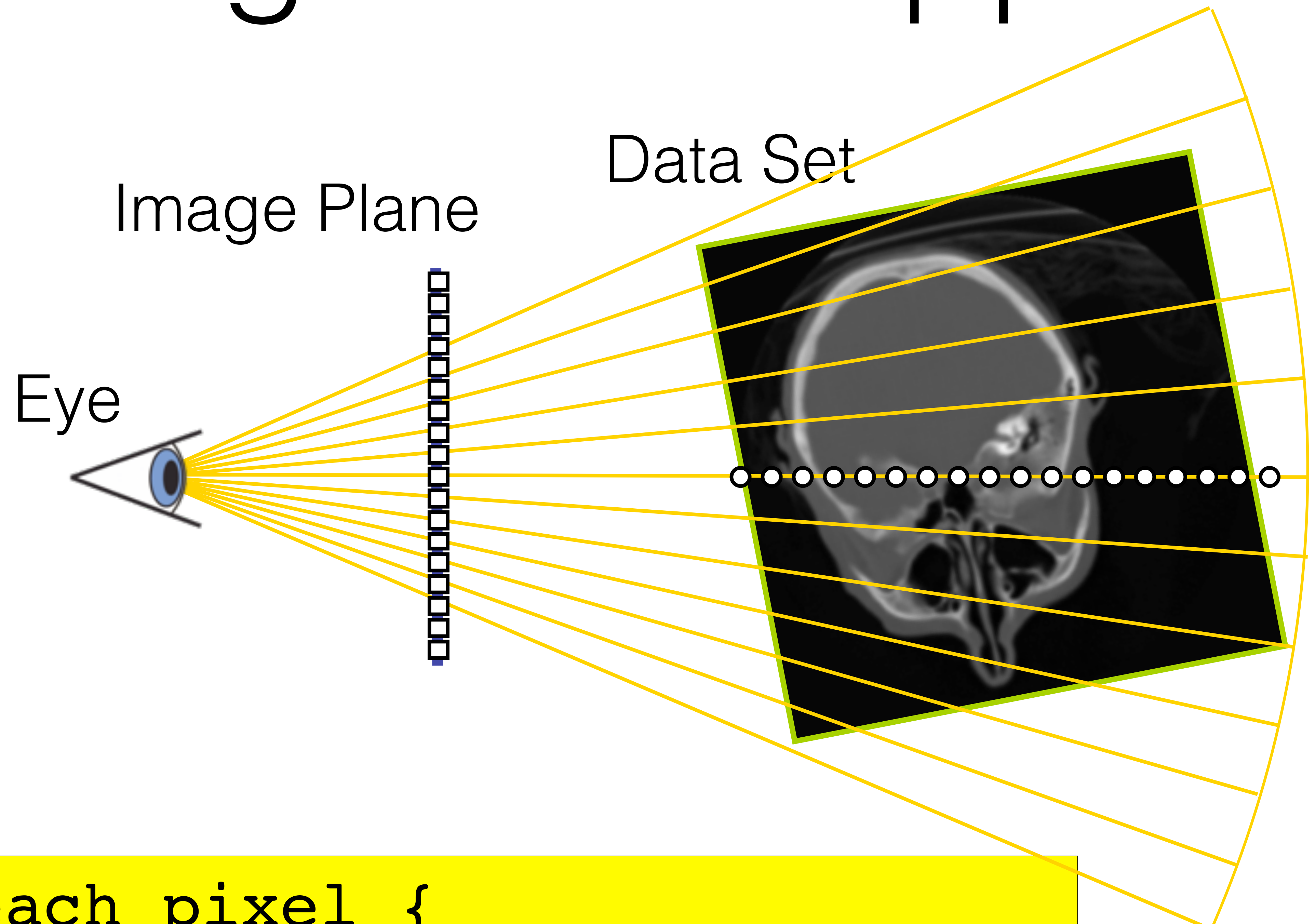


Image order approach




```
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

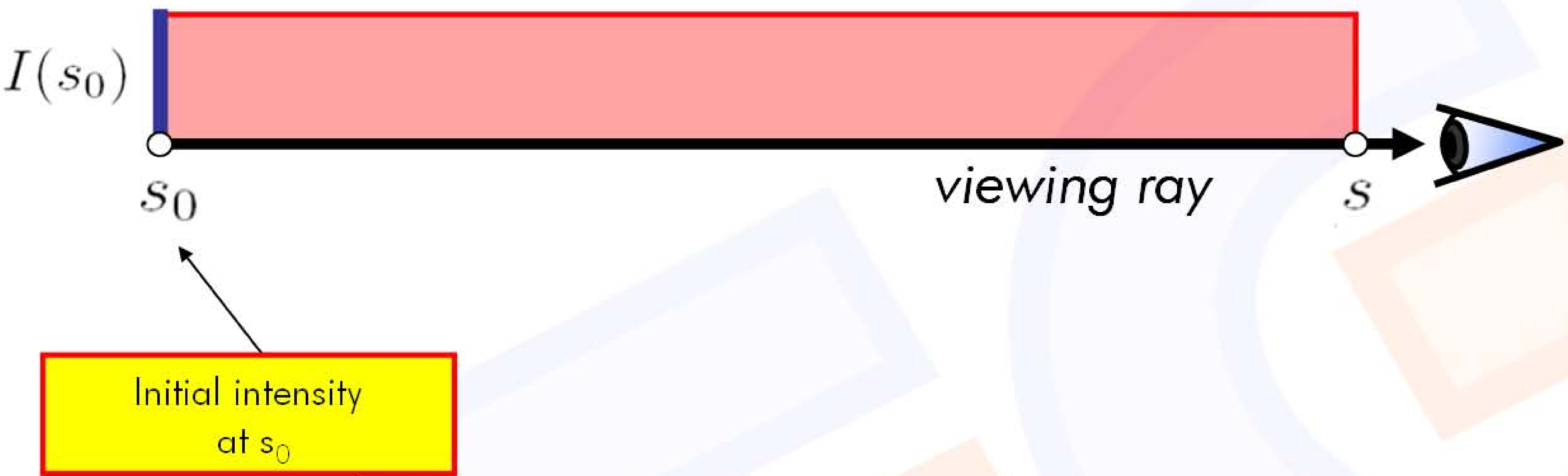
Initial intensity at s_0

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

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



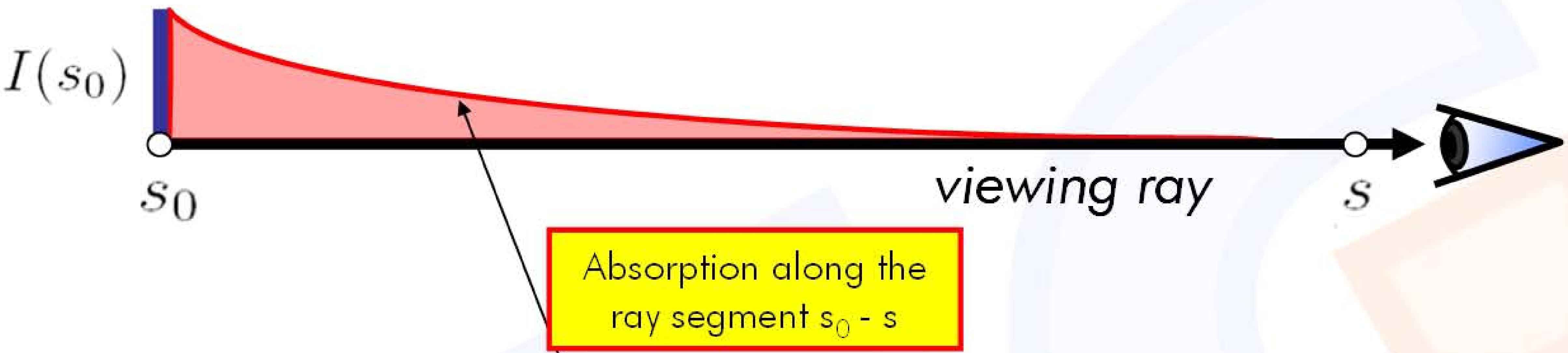
$$I(s) = \boxed{I(s_0)}$$

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



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

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

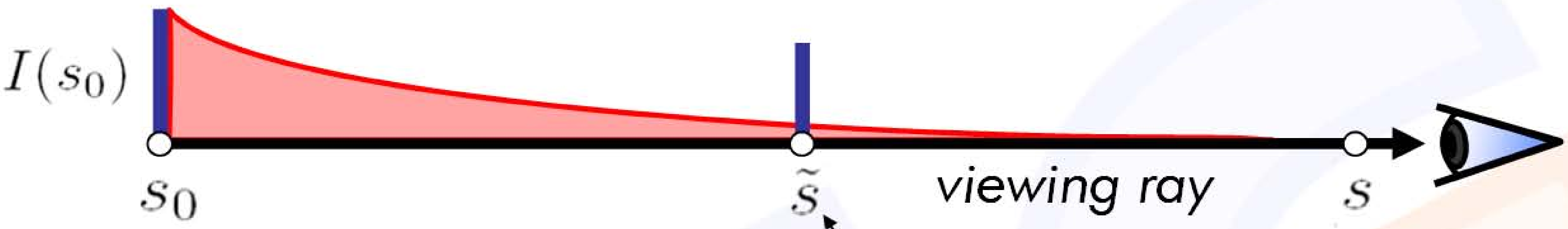
Absorption κ

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

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.

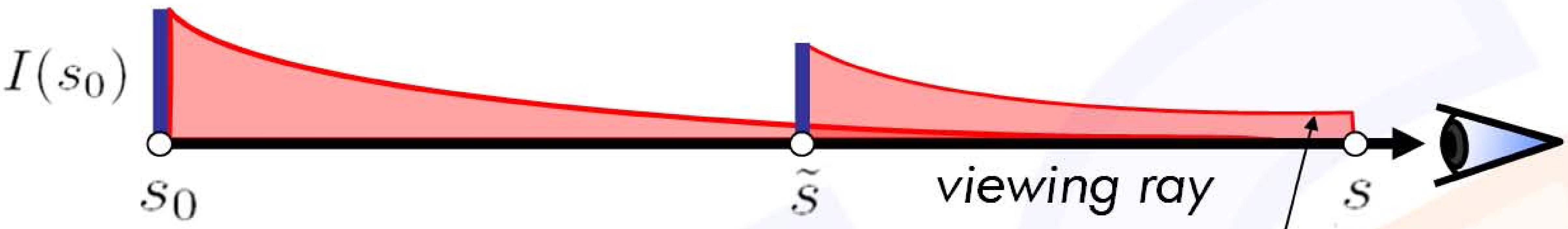
Active emission at point $\tilde{s}$

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

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.

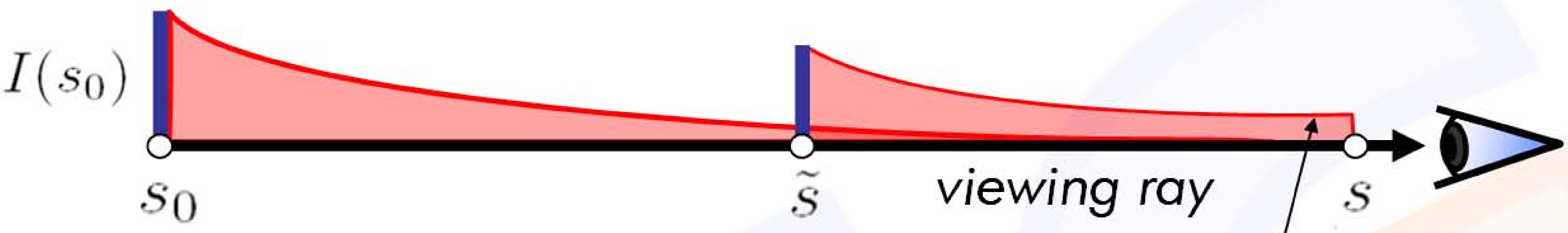
Absorption along the distance s - $\tilde{s}$

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

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering

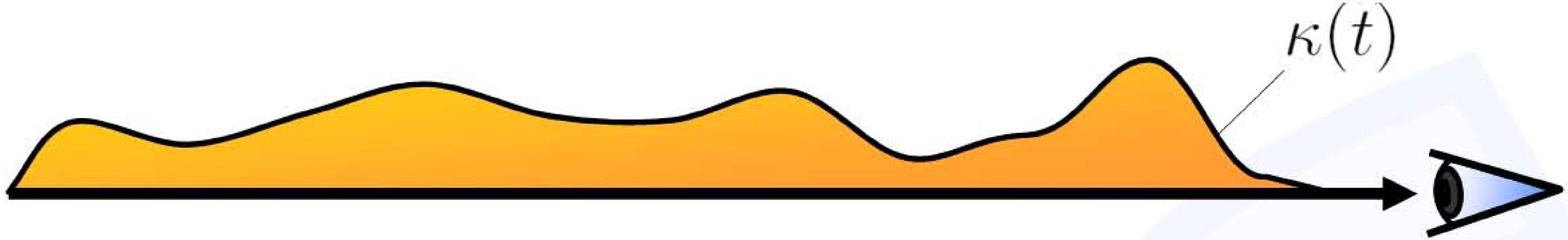


One point $\tilde{s}$ along the viewing ray emits additional radiant energy.

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

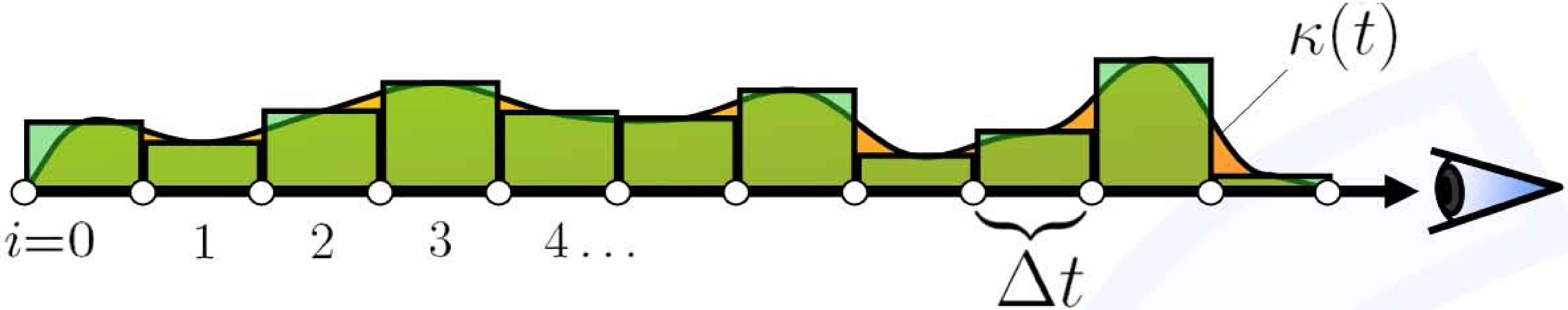
$$I(s) = I(s_0)\, e^{-\tau(s_0,s)} + q(\tilde{s})\, \boxed{e^{-\tau(\tilde{s},s)}}$$

Numerical Solution



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

Numerical Solution



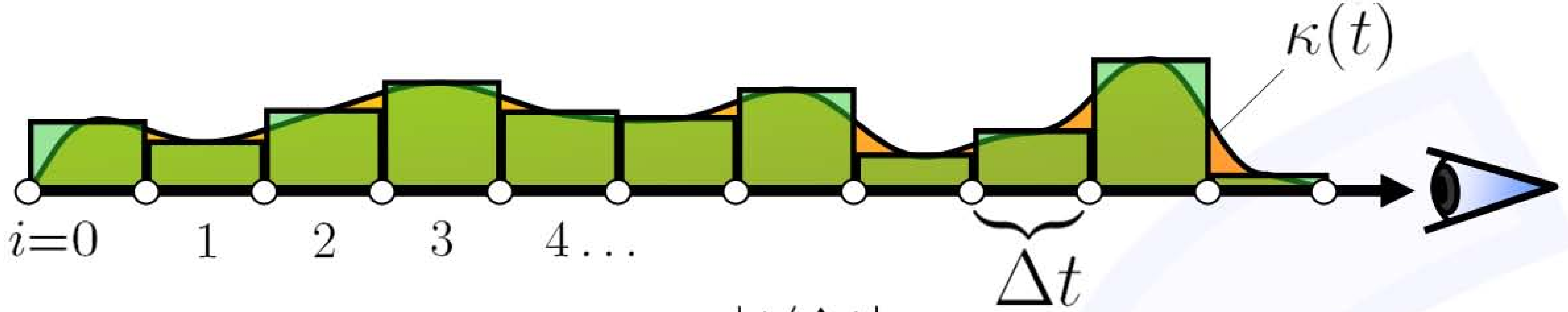
Extinction:

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

Approximate Integral by Riemann sum:

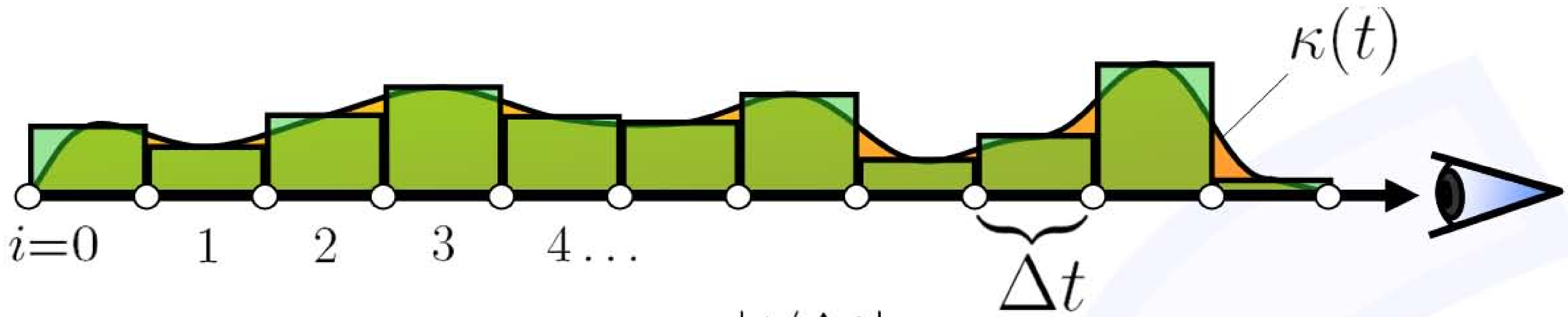
$$\tau(0,t) \;\approx\; \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

Numerical Solution



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

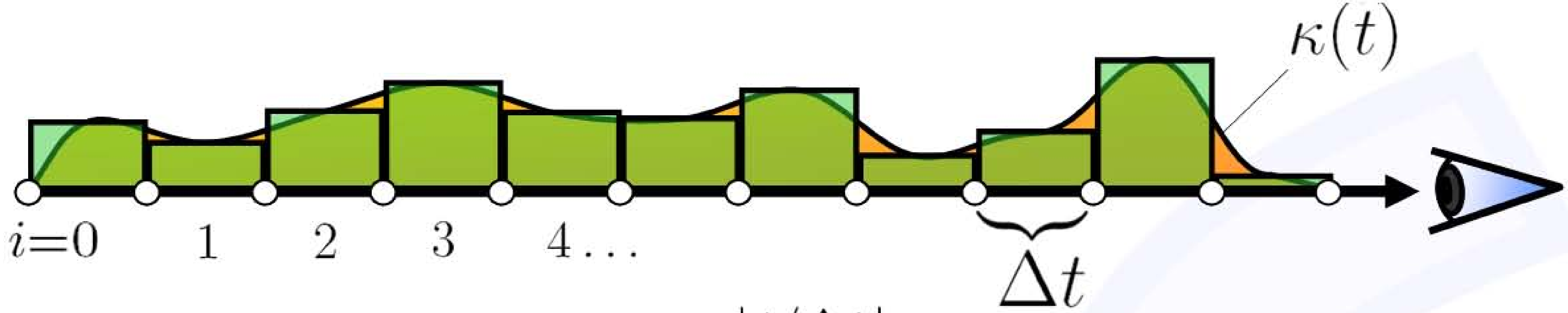
Numerical Solution



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

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

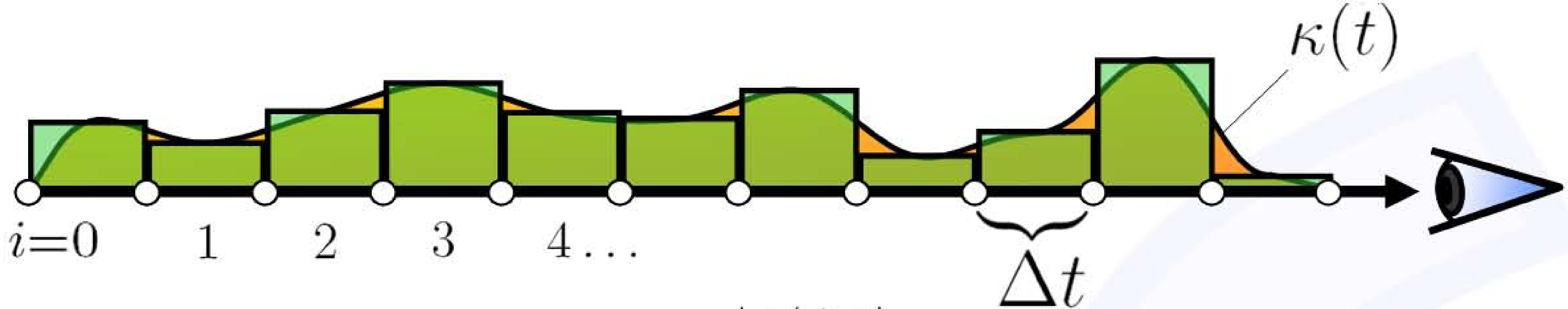
Numerical Solution



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

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

Numerical Solution



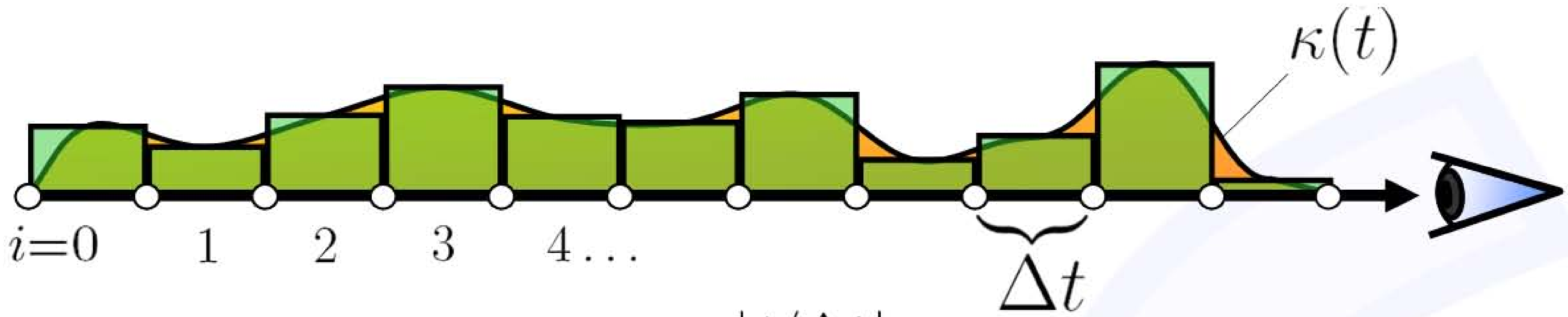
$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

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

Now we introduce opacity:

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

Numerical Solution



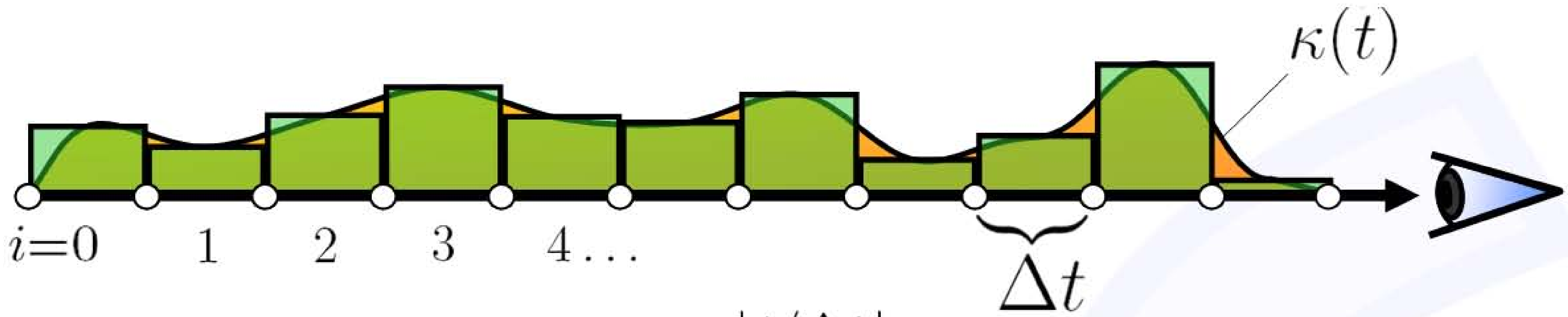
$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

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

Now we introduce opacity:

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

Numerical Solution



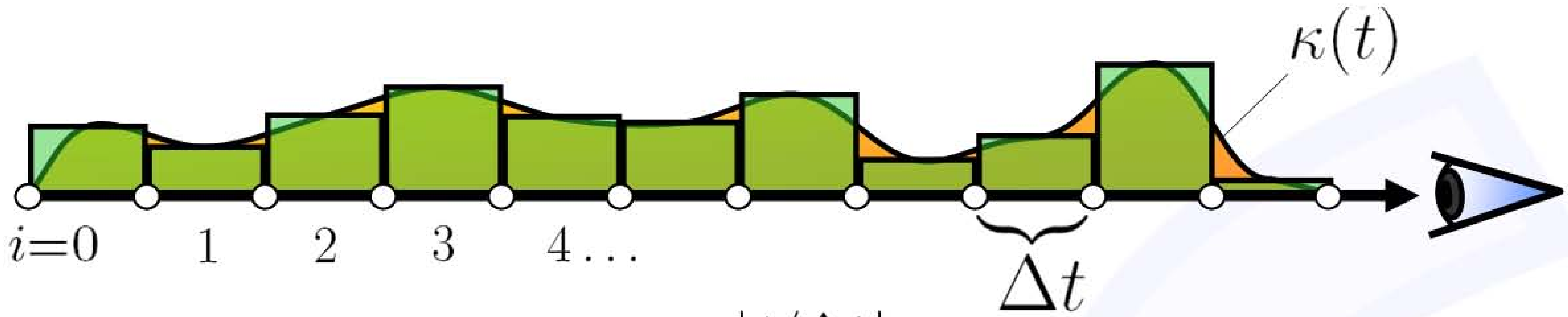
$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

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

Now we introduce opacity:

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

Numerical Solution



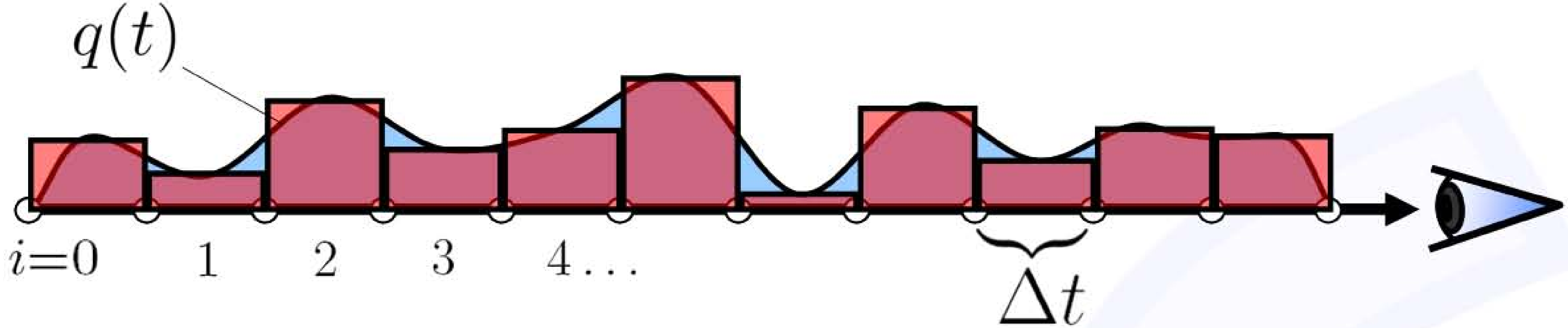
$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

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

Now we introduce opacity:

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

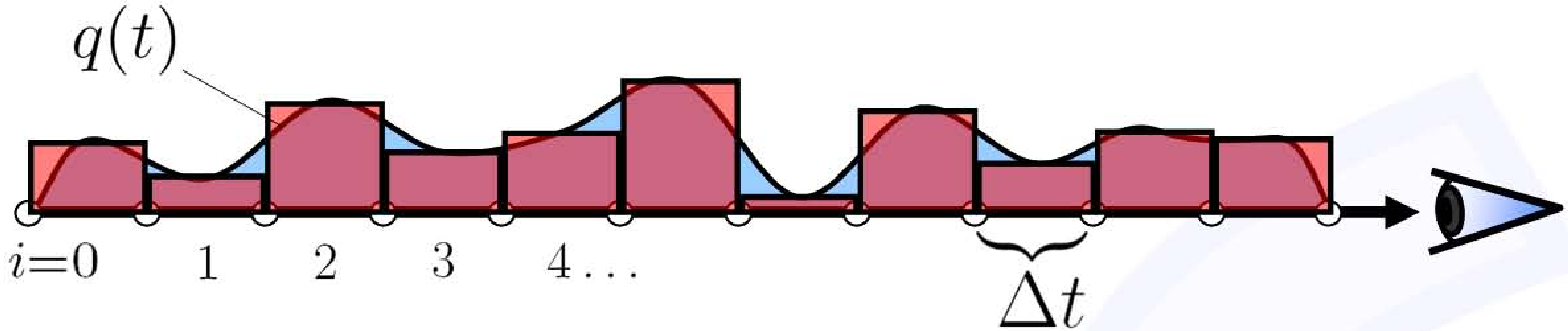
Numerical Solution



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

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

Numerical Solution

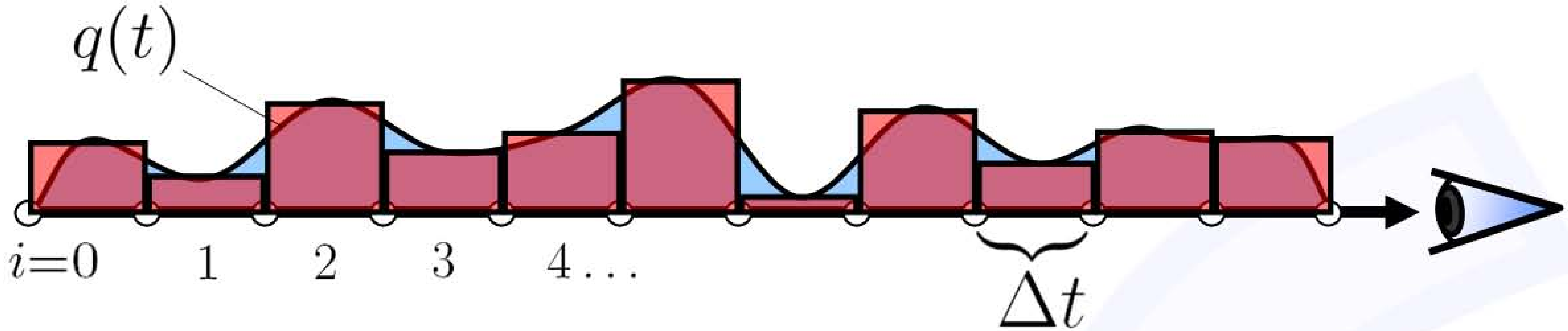


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

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

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

Numerical Solution

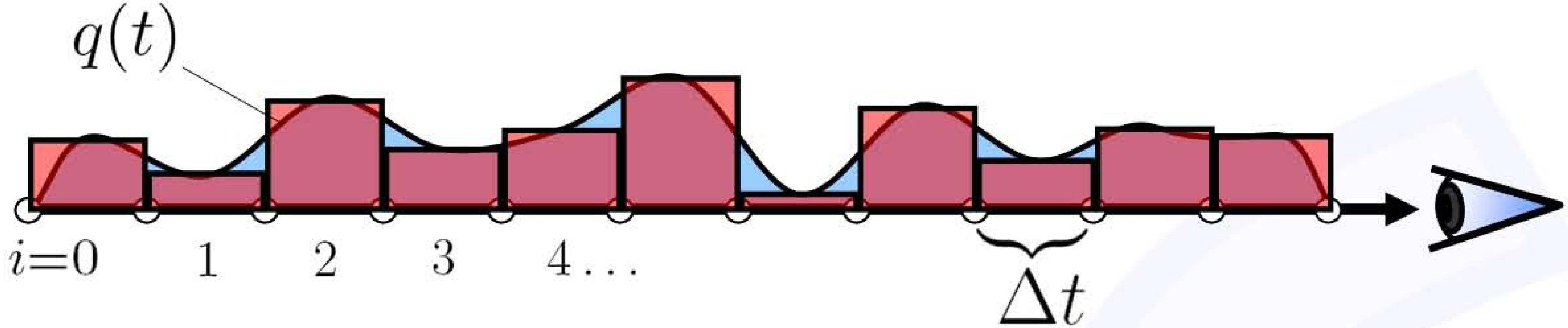


$$\boxed{e^{-\tilde{\tau}(0,t)}} \;=\; \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

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

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

Numerical Solution

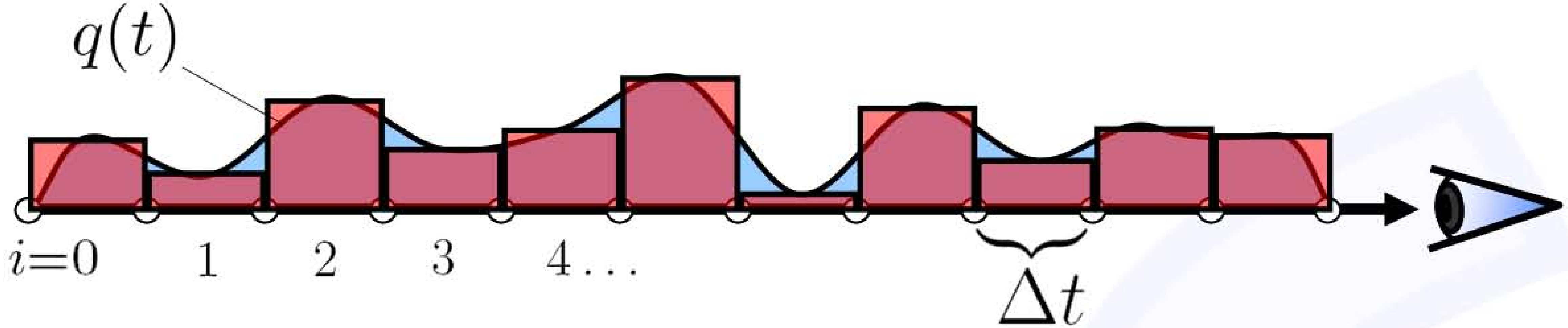


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

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

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

Numerical Solution

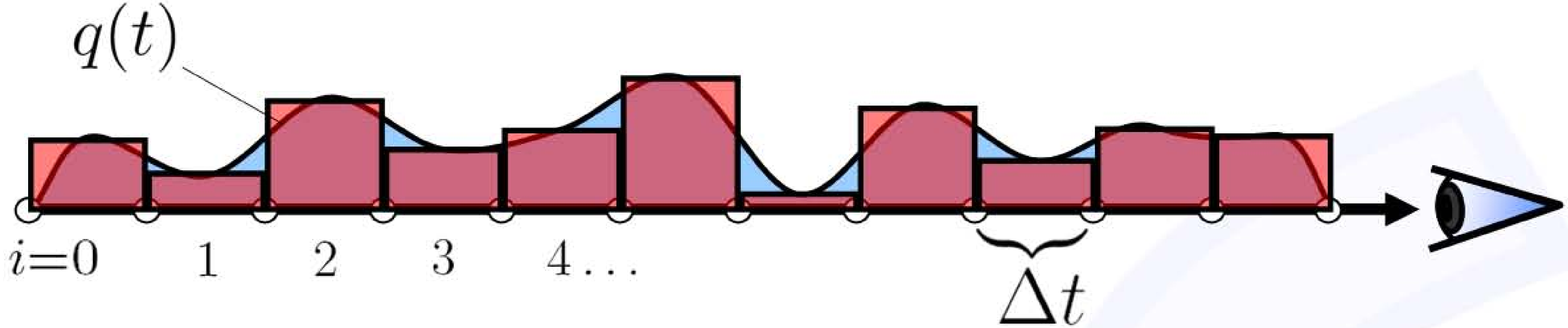


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

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

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

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

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

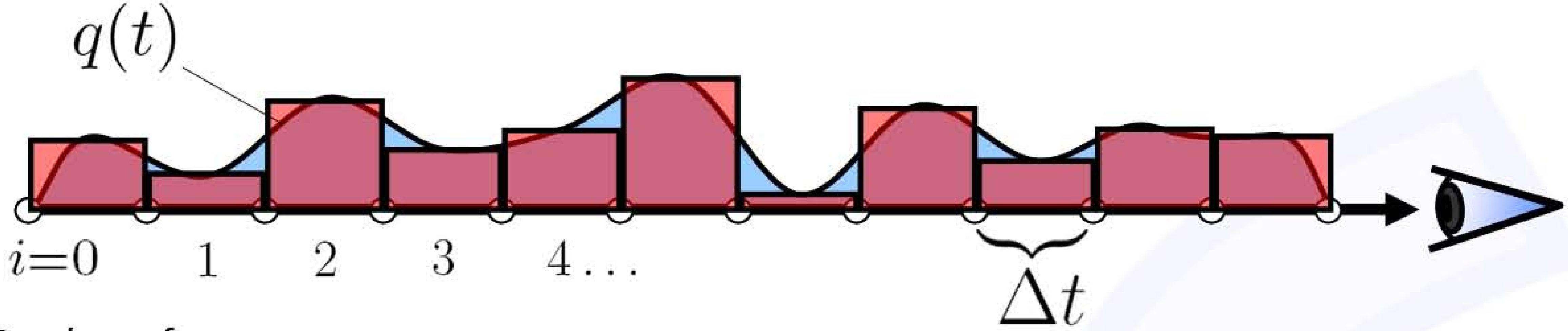
Radiant energy observed at position *i*

Radiant energy emitted at position *i*

Absorption at position *i*

Radiant energy observed at position *i*–1

Numerical Solution



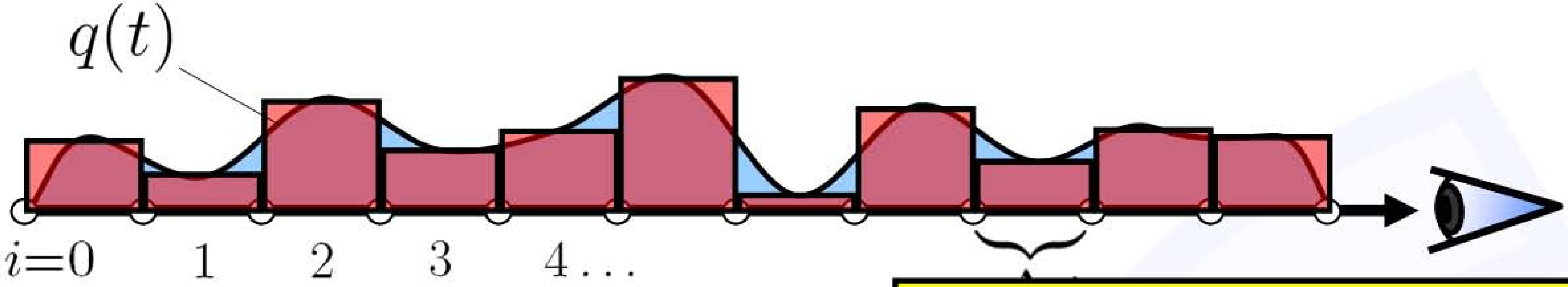
Back-to-front compositing

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

Front-to-back compositing

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

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

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

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

Summary

- Emission Absorption Model

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

- Numerical Solutions

Back-to-front iteration

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

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

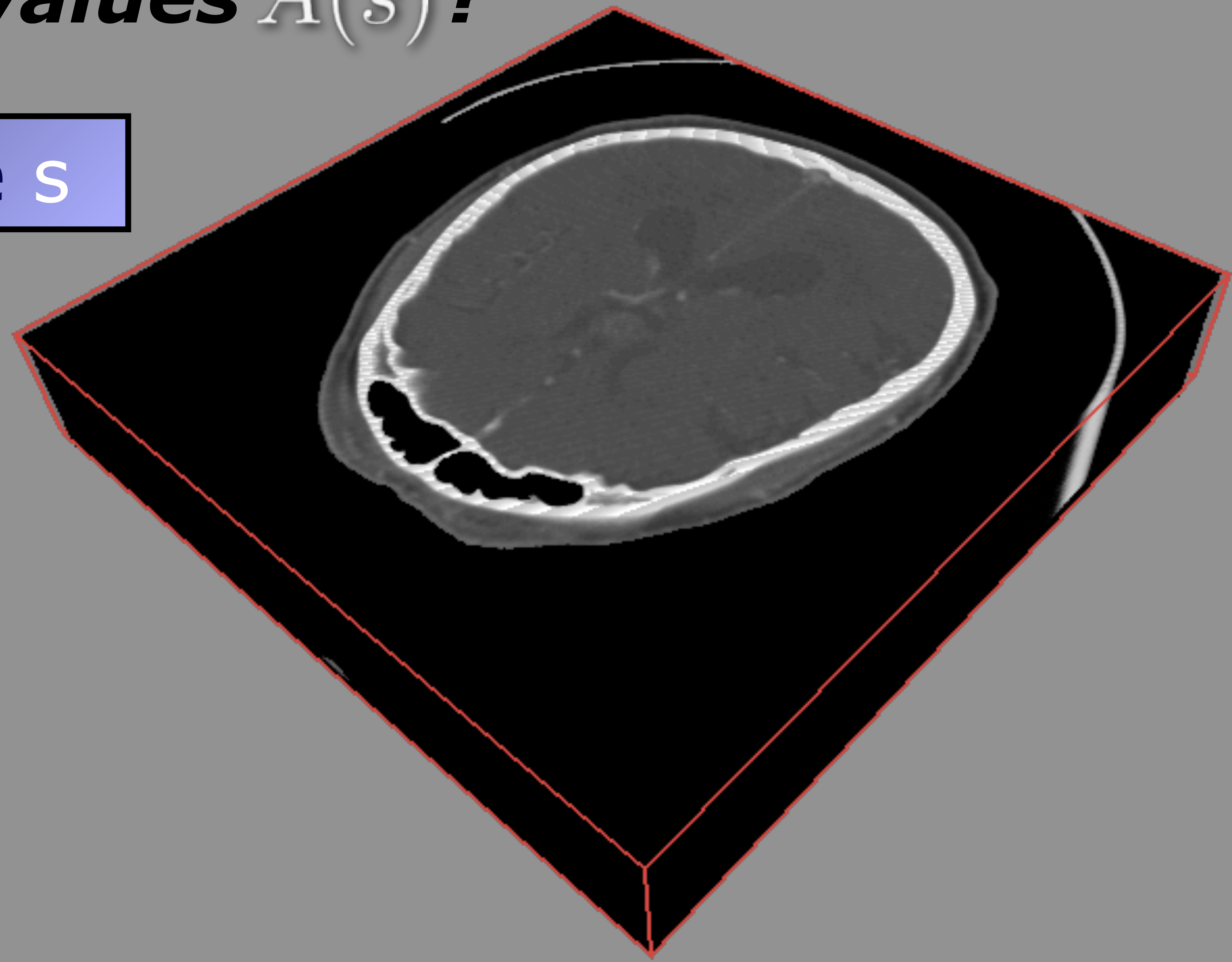
```
//the step size, i.e. "delta t" from the Engel slides.
float dt = 1.0 / normalize(volumeGridDimensions);

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;
}
```

Transfer functions

Classification

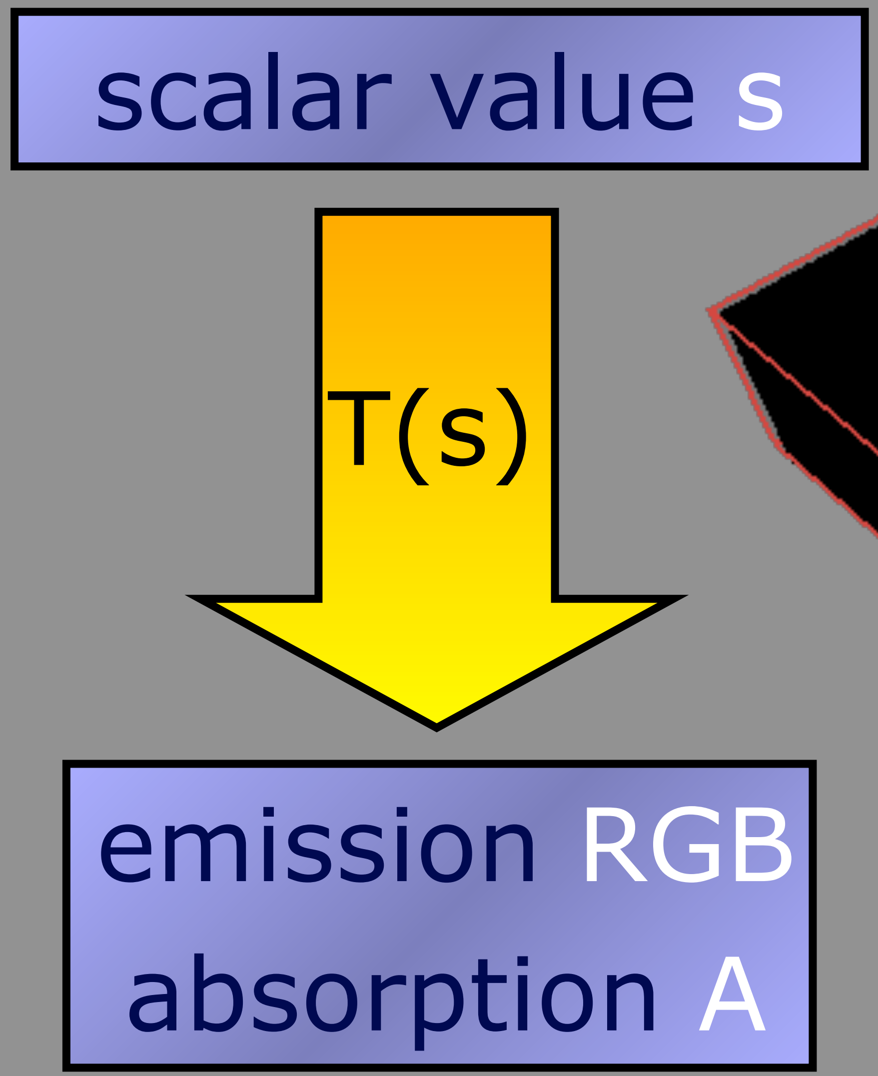
How do I obtain the emission values $q(s)$ (C_i) ***and Absorption values*** $A(s)$***?***

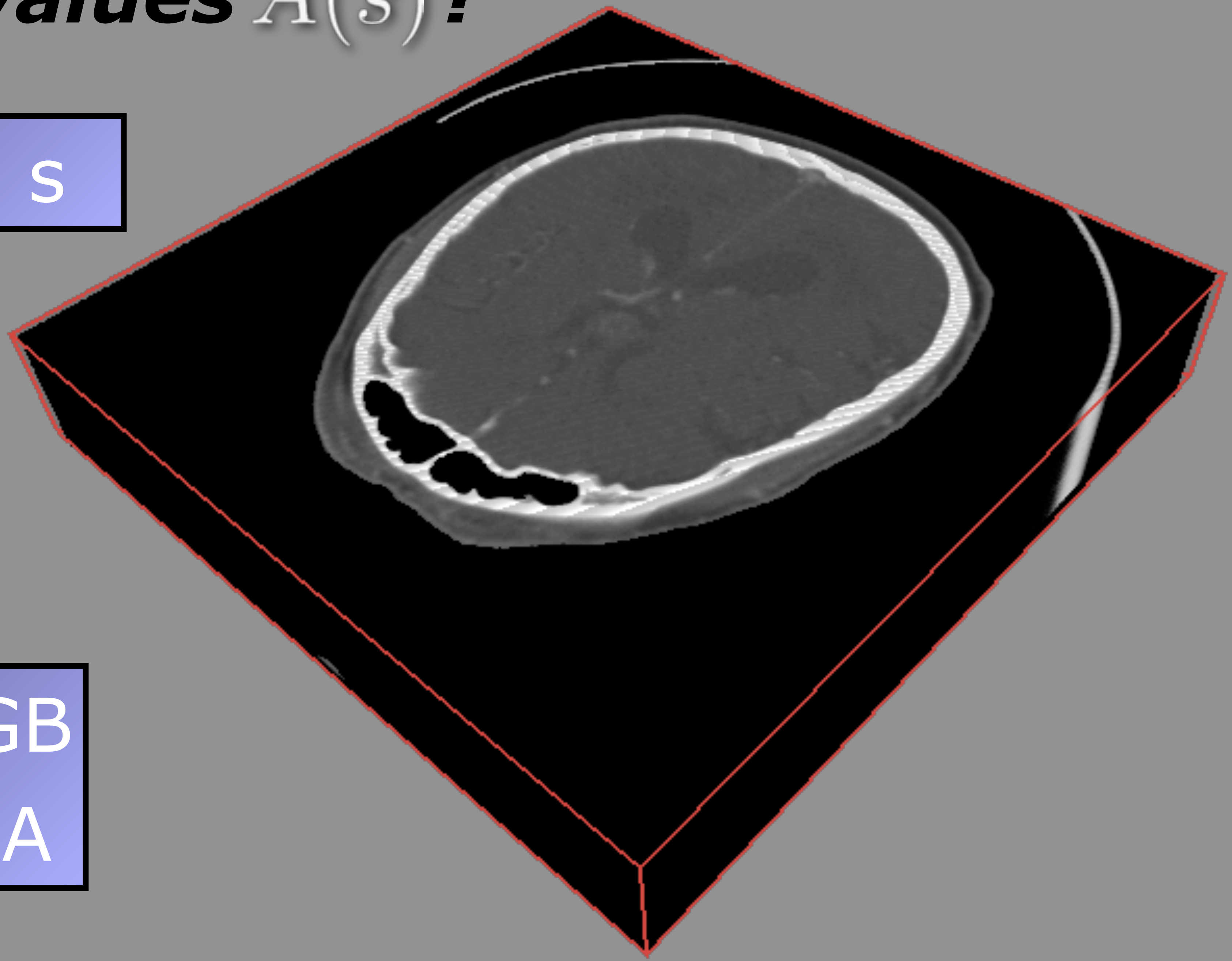
scalar value s



Classification

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





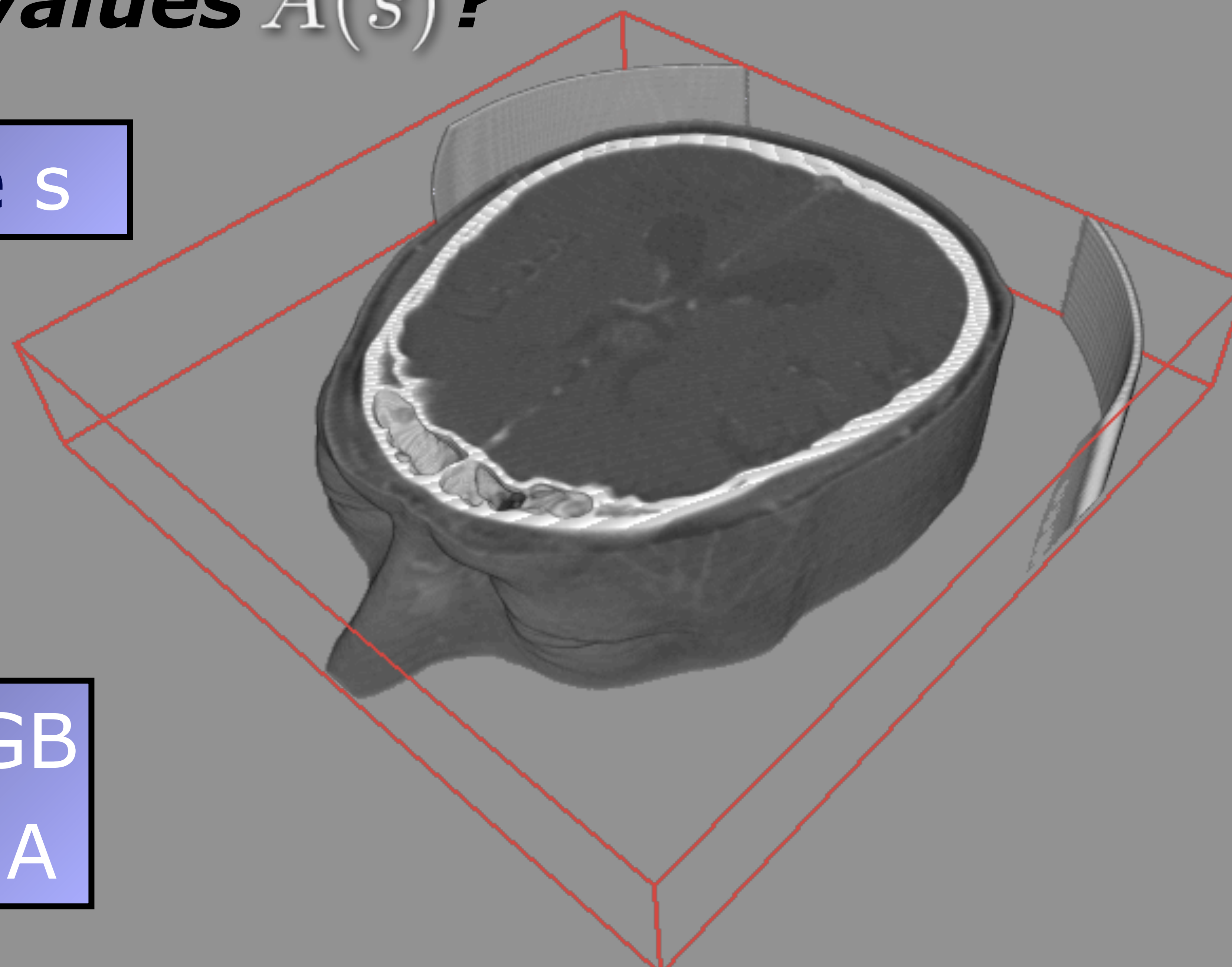
Classification

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

scalar value s

T(s)

emission RGB
absorption A

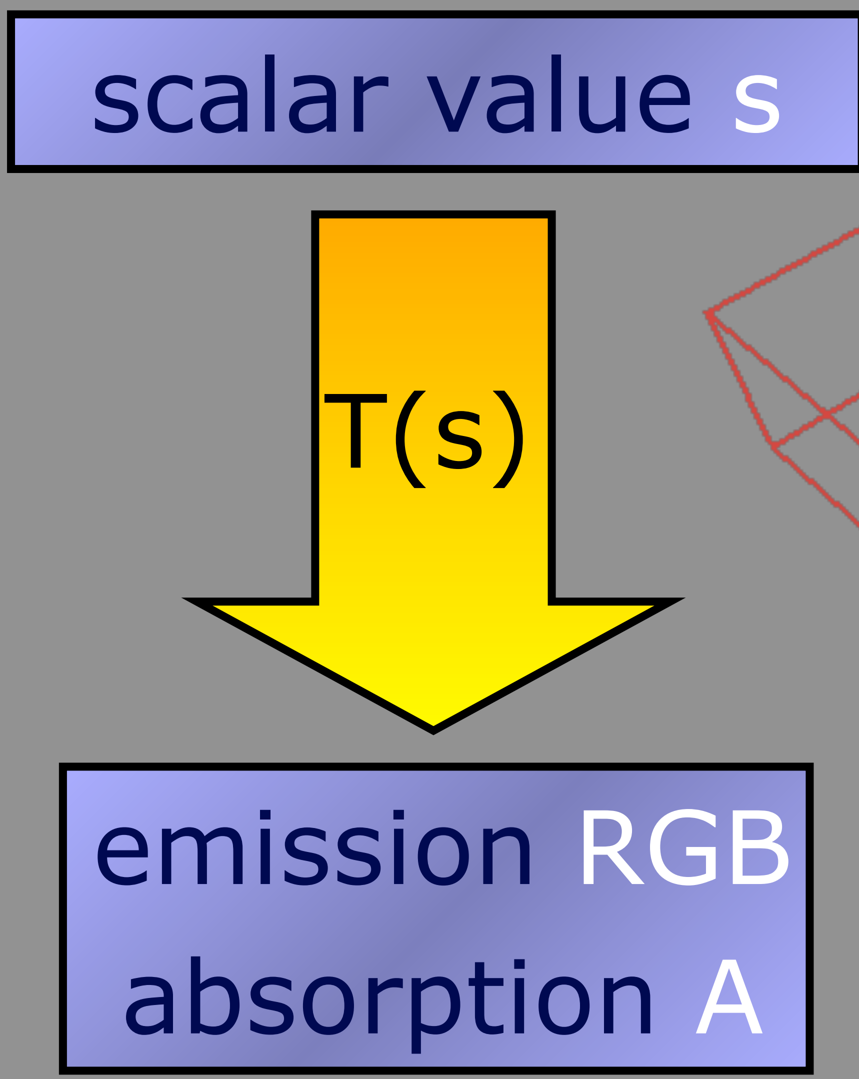


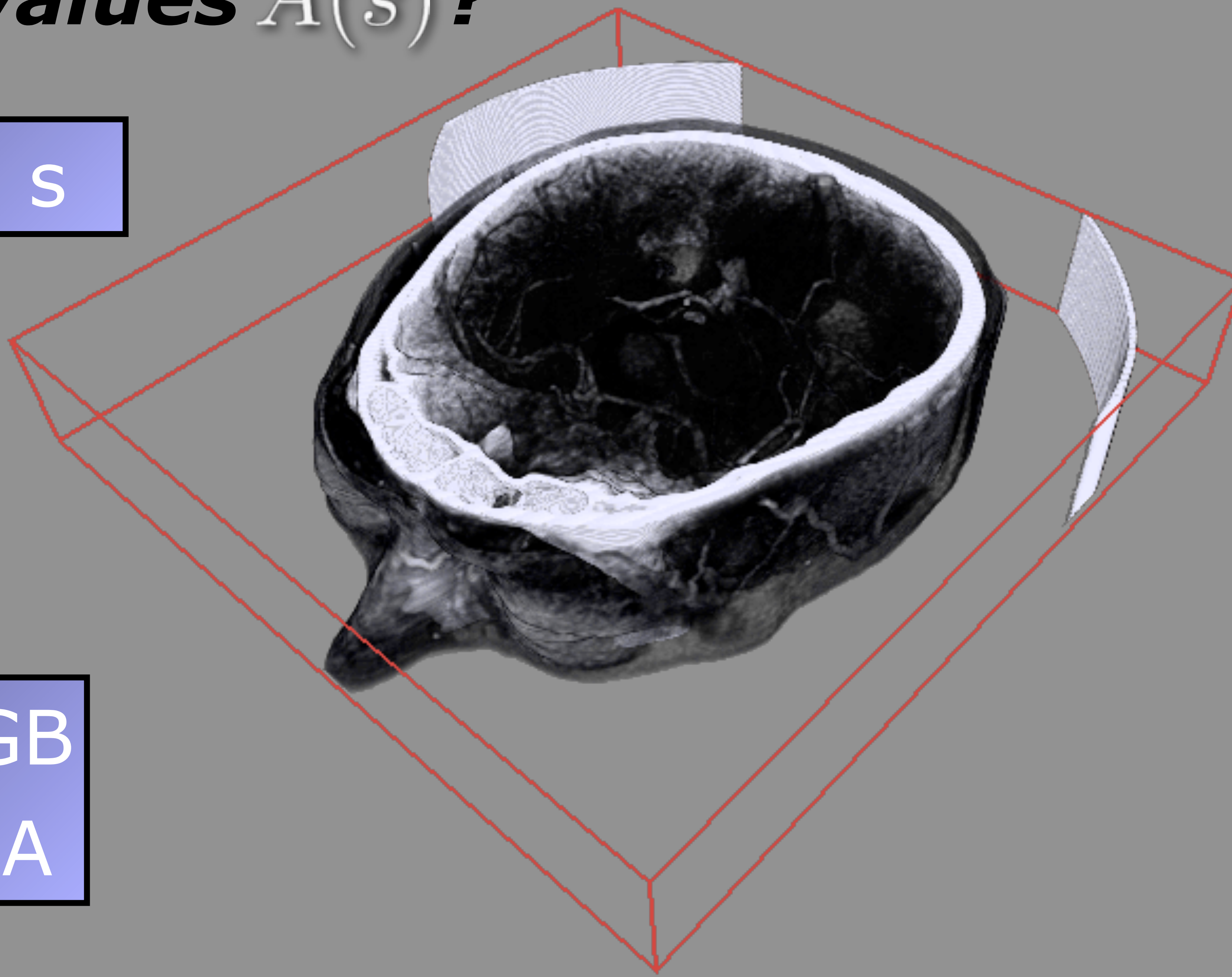
Joshua Levine, Clemson University

Classification

(C_i)

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

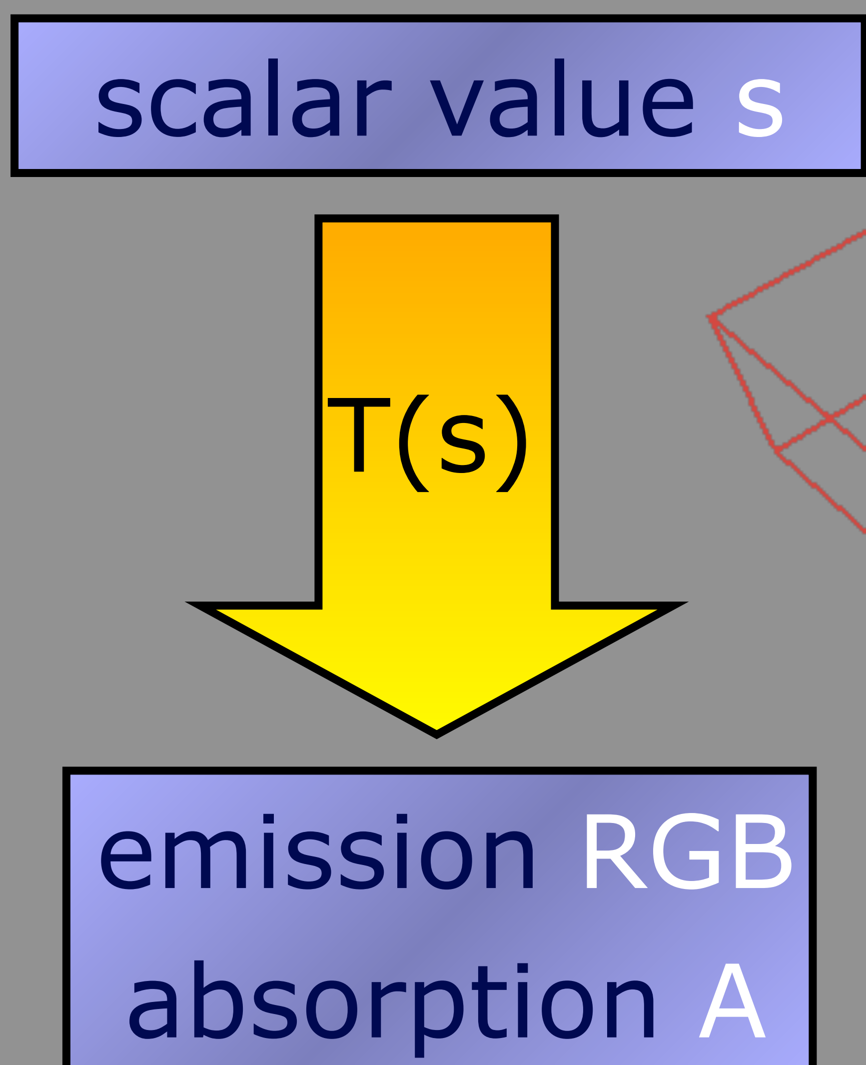


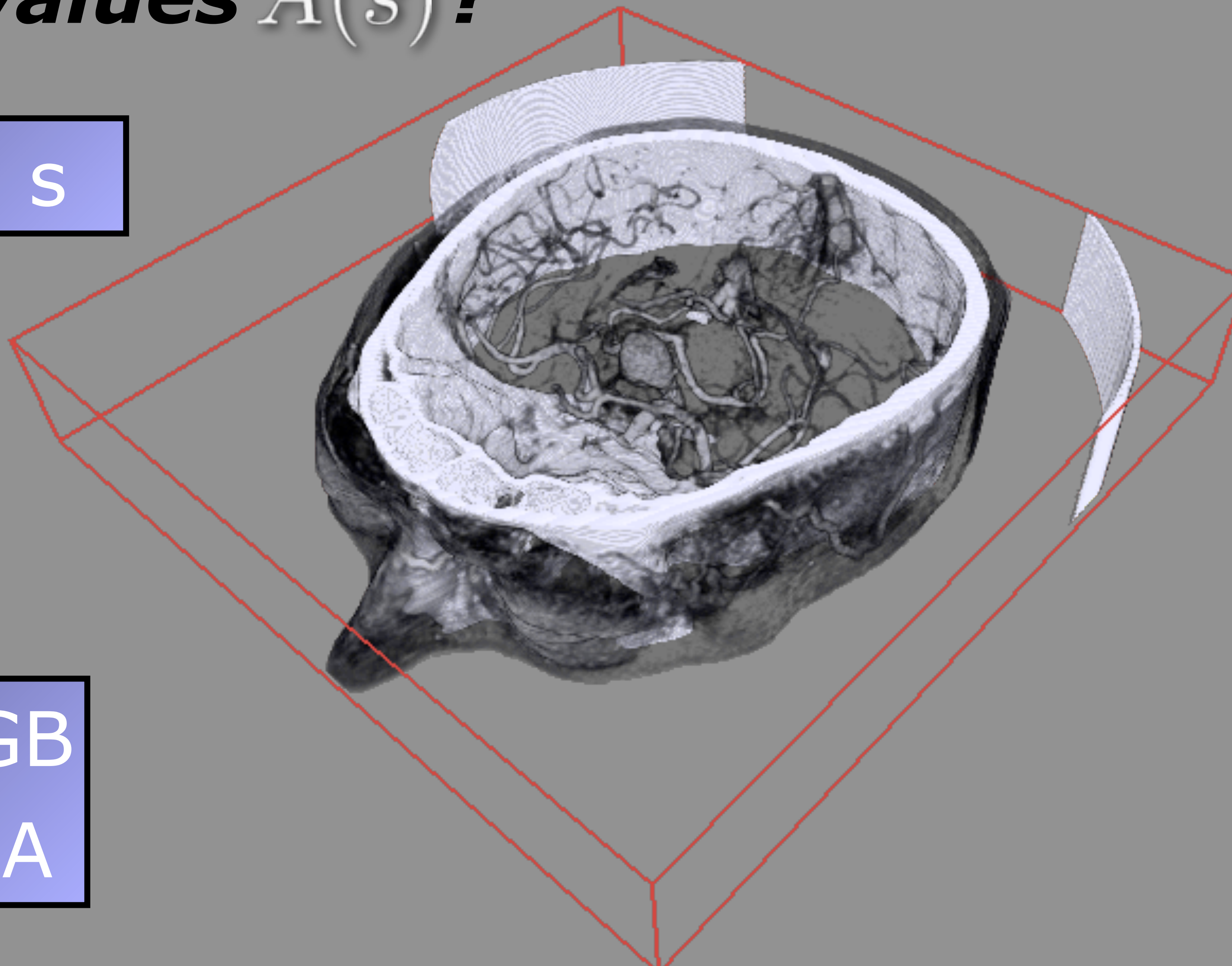


Joshua Levine, Clemson University

Classification

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





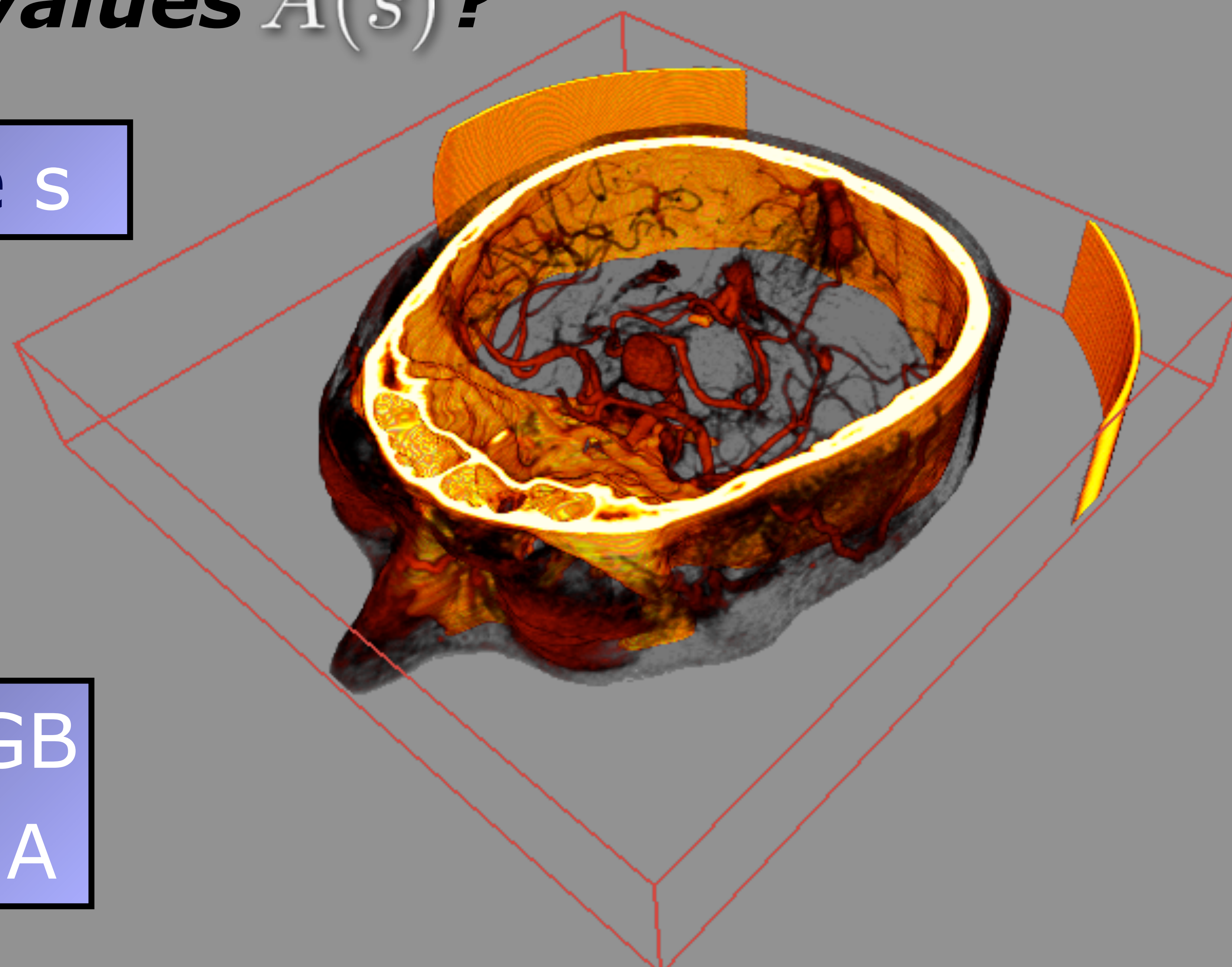
Classification

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

scalar value s

T(s)

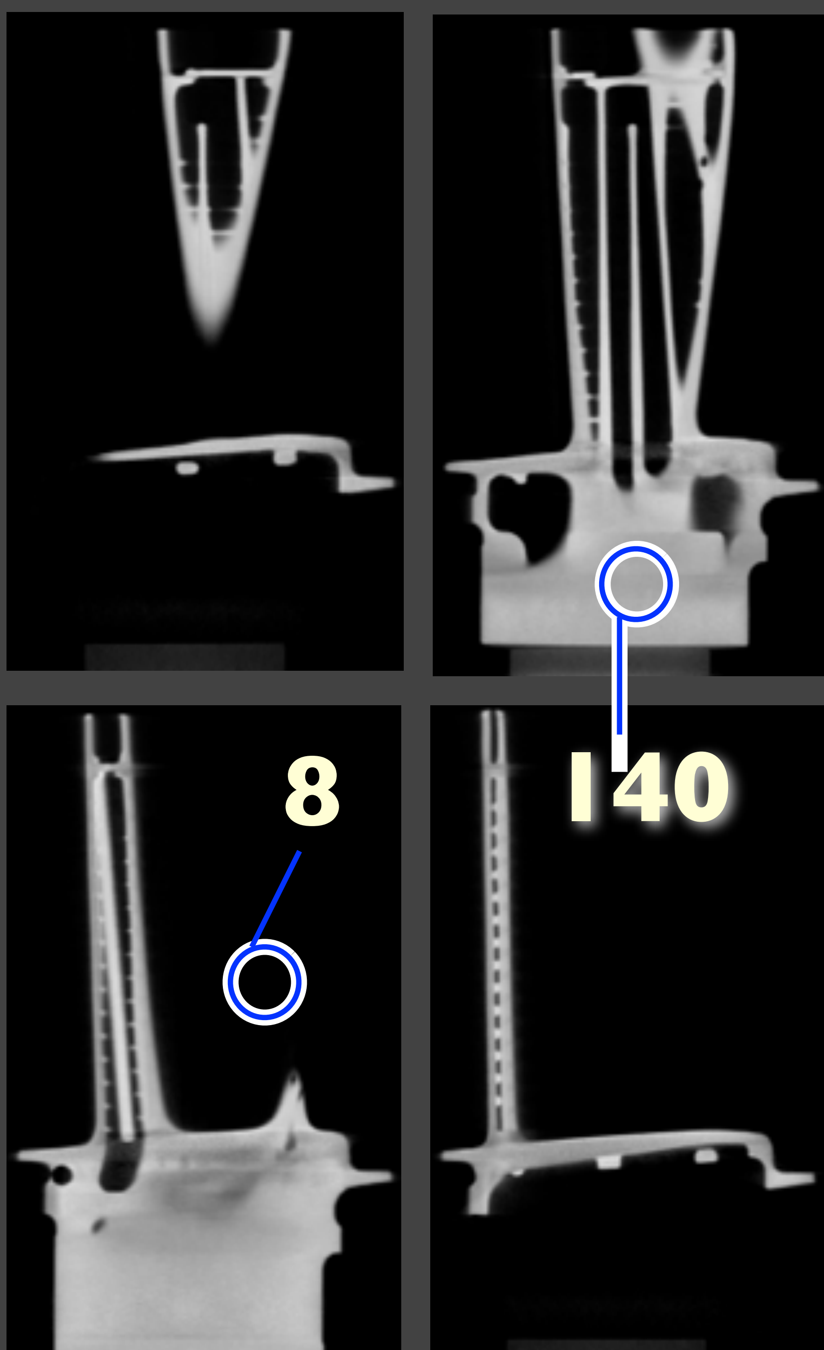
emission RGB
absorption A



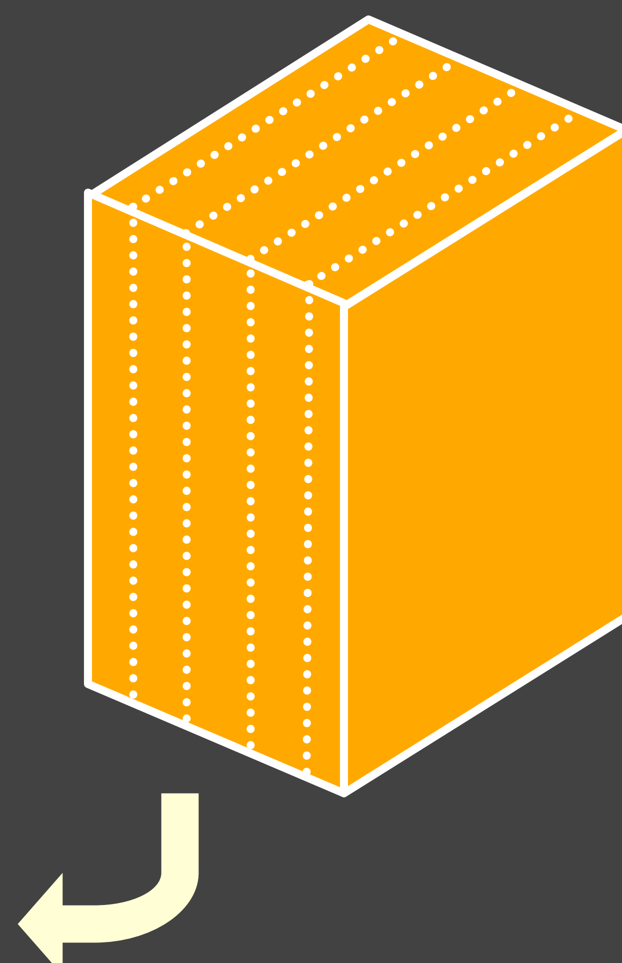
Introduction

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

slices:



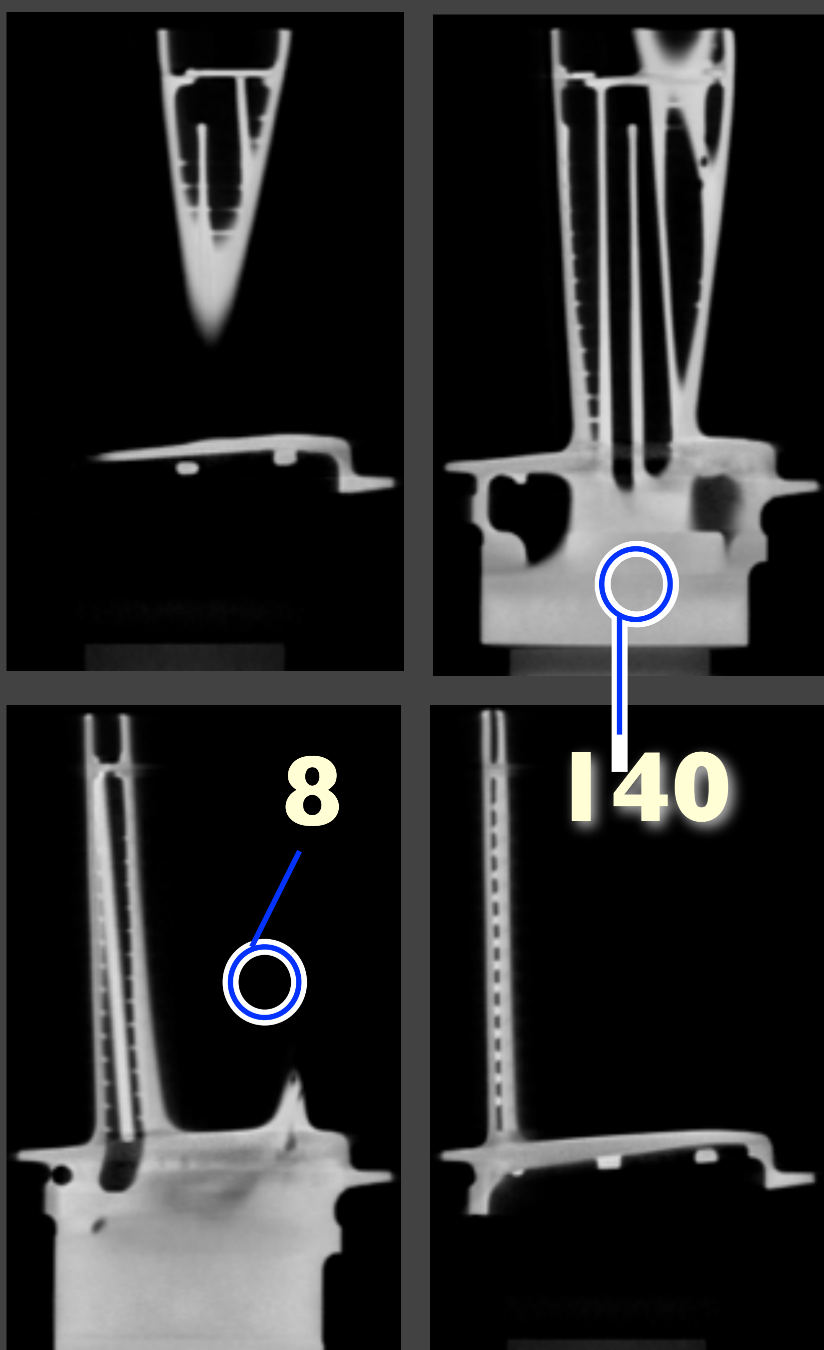
volume data:



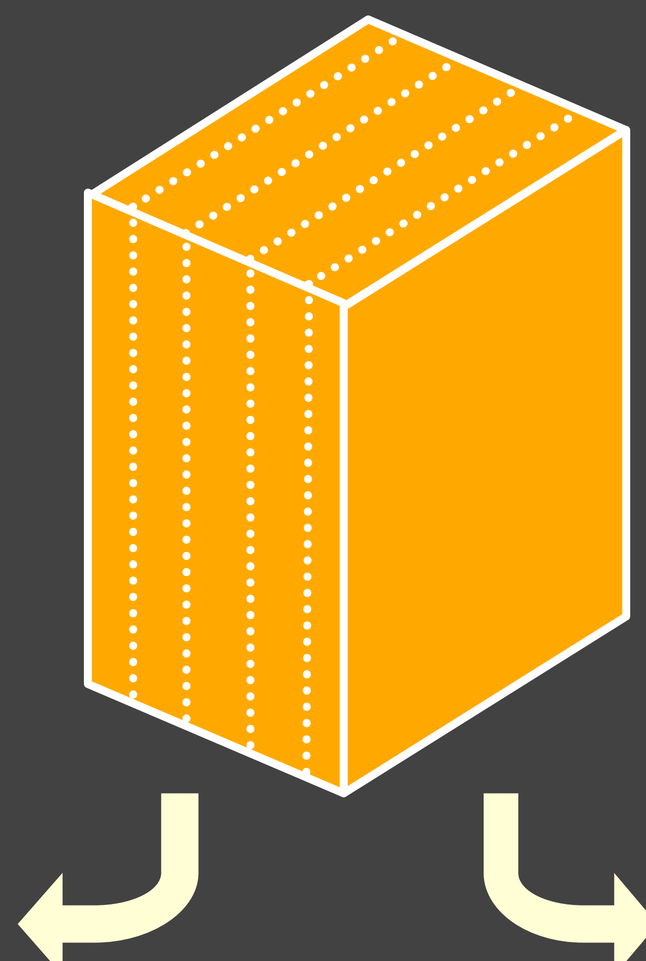
Introduction

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

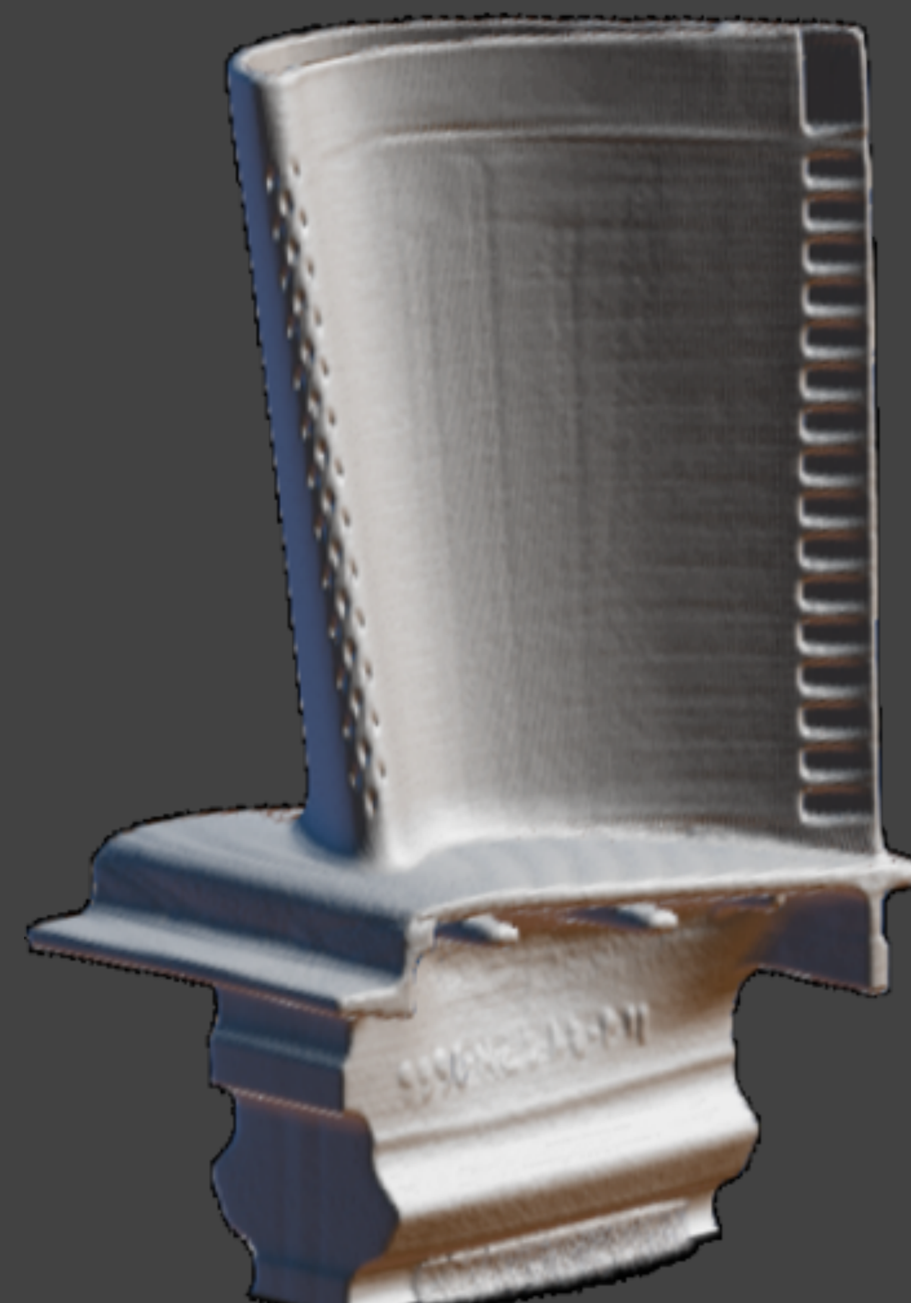
slices:



volume data:

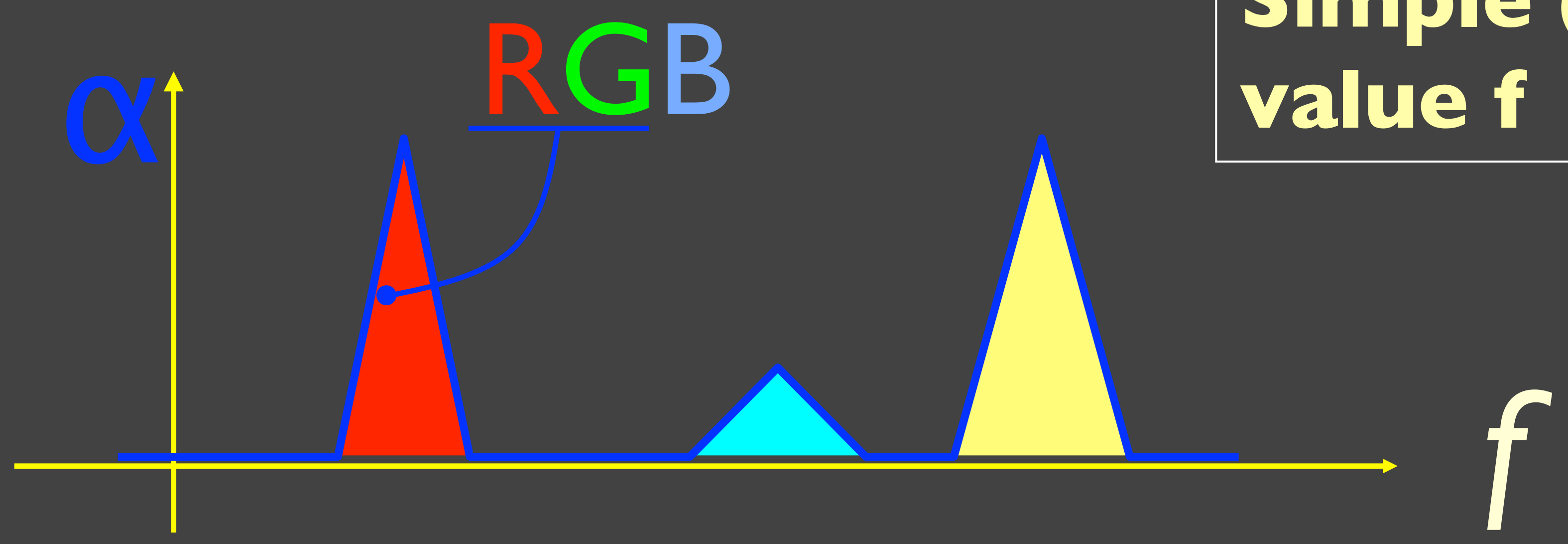


volume rendering:



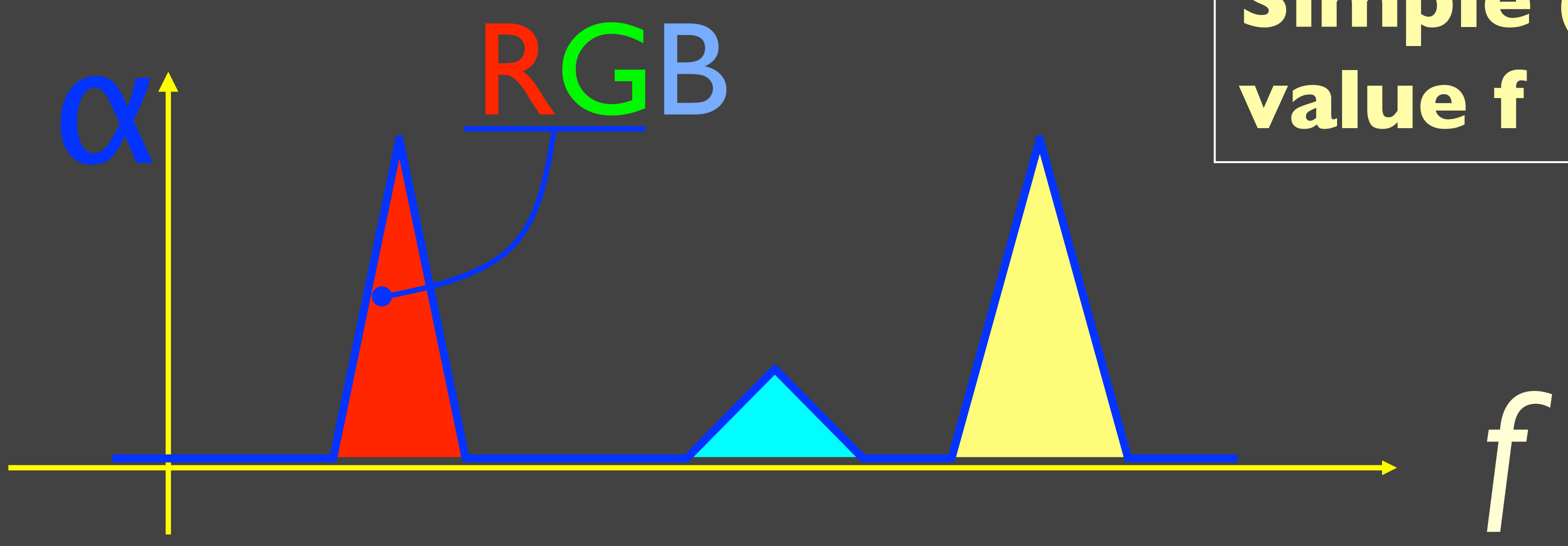
Transfer Functions (TFs)

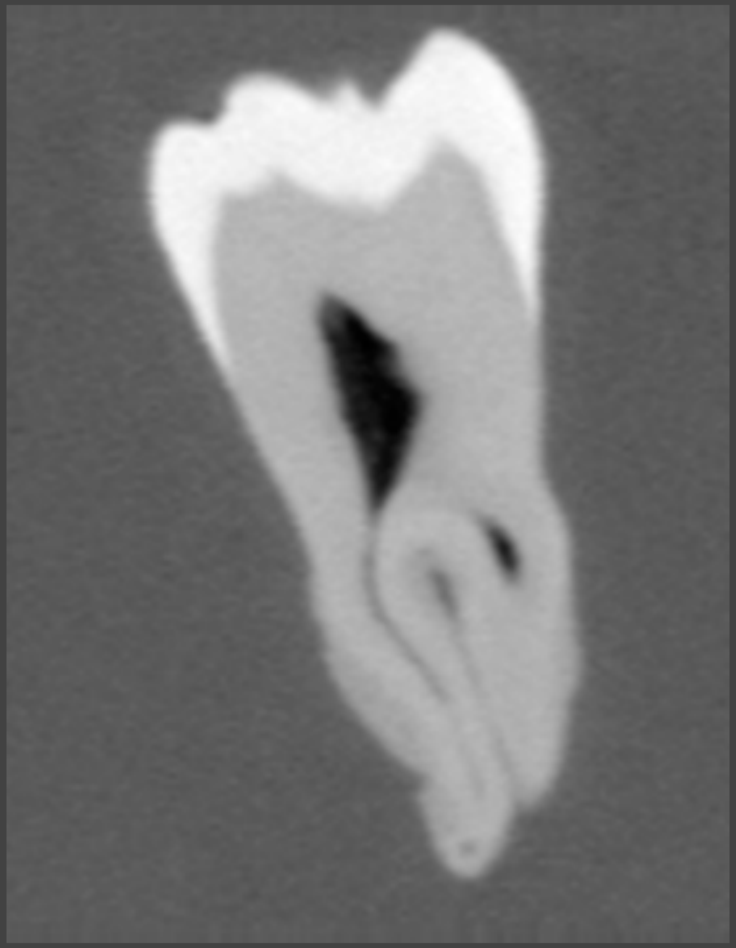
Simple (usual) case: Map data value f to color and opacity



Transfer Functions (TFs)

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

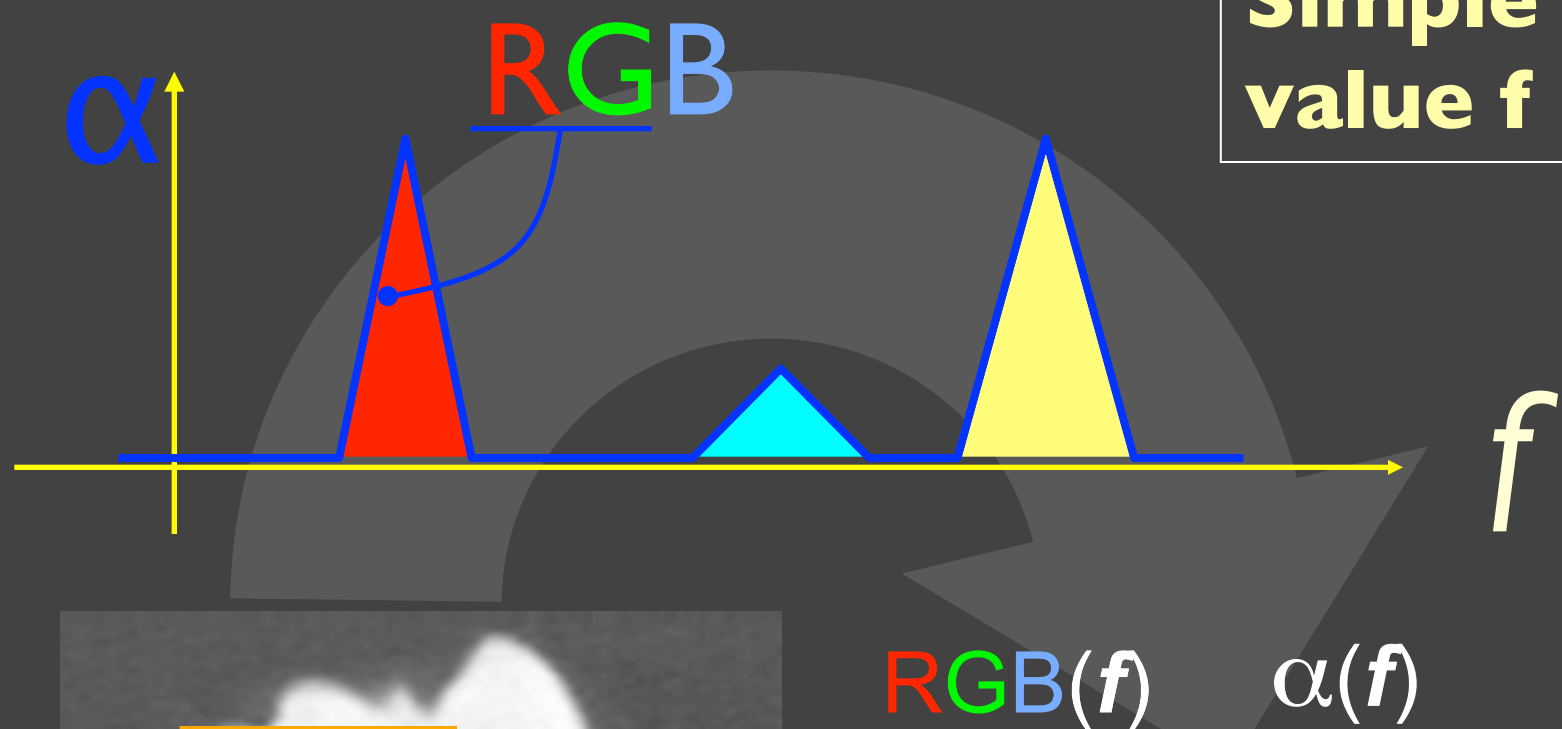


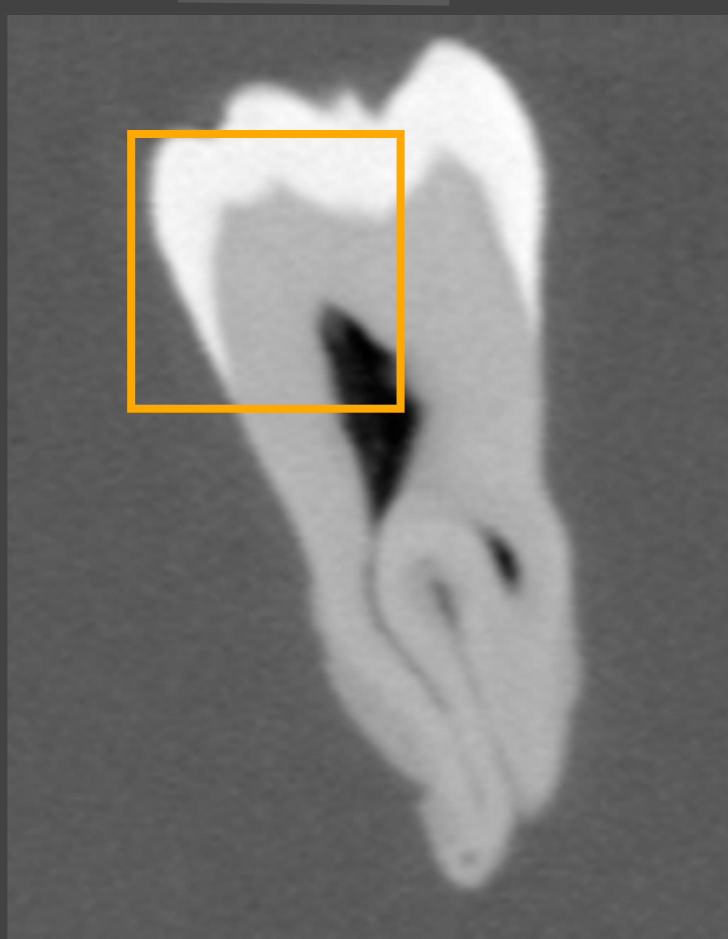


Human Tooth CT

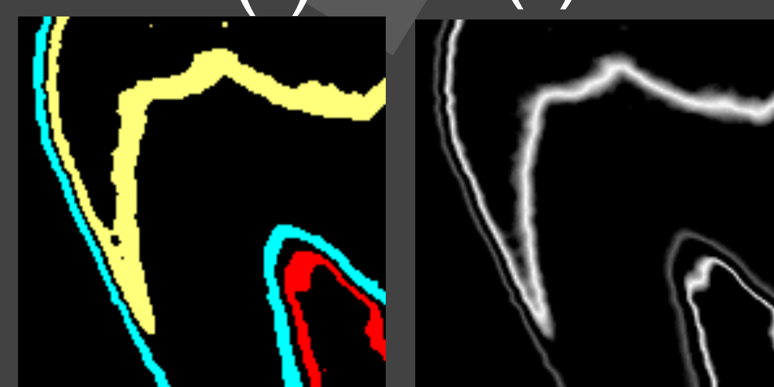
Transfer Functions (TFs)

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



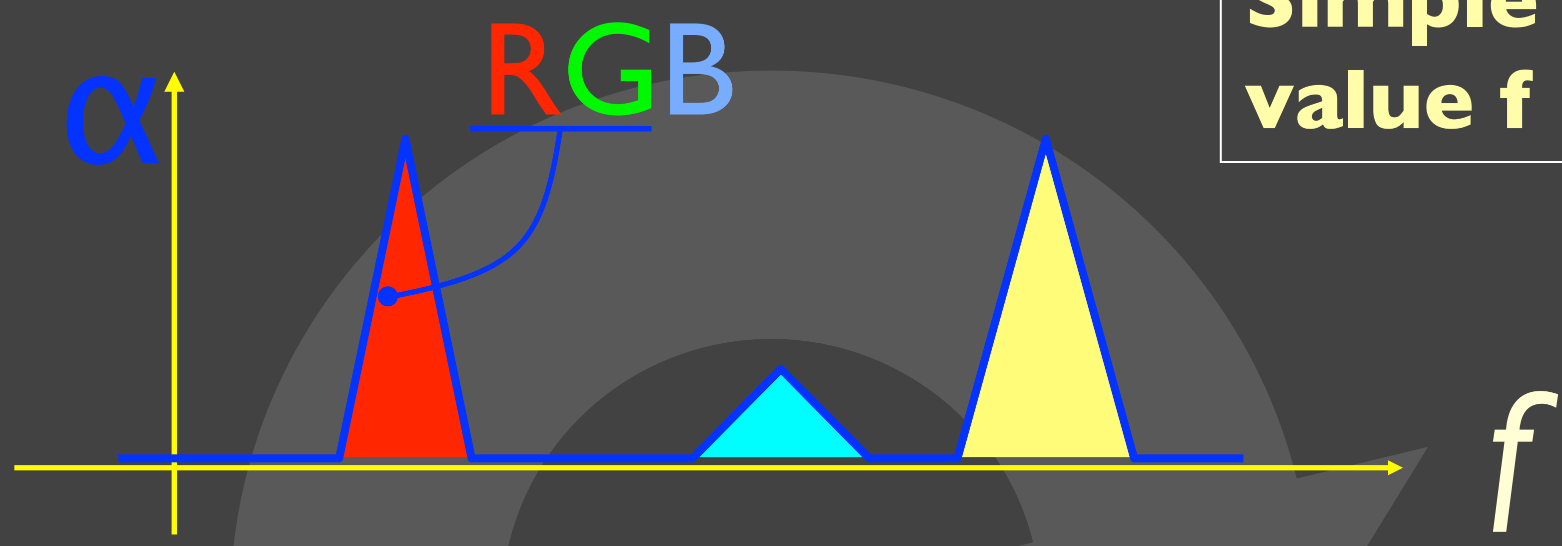


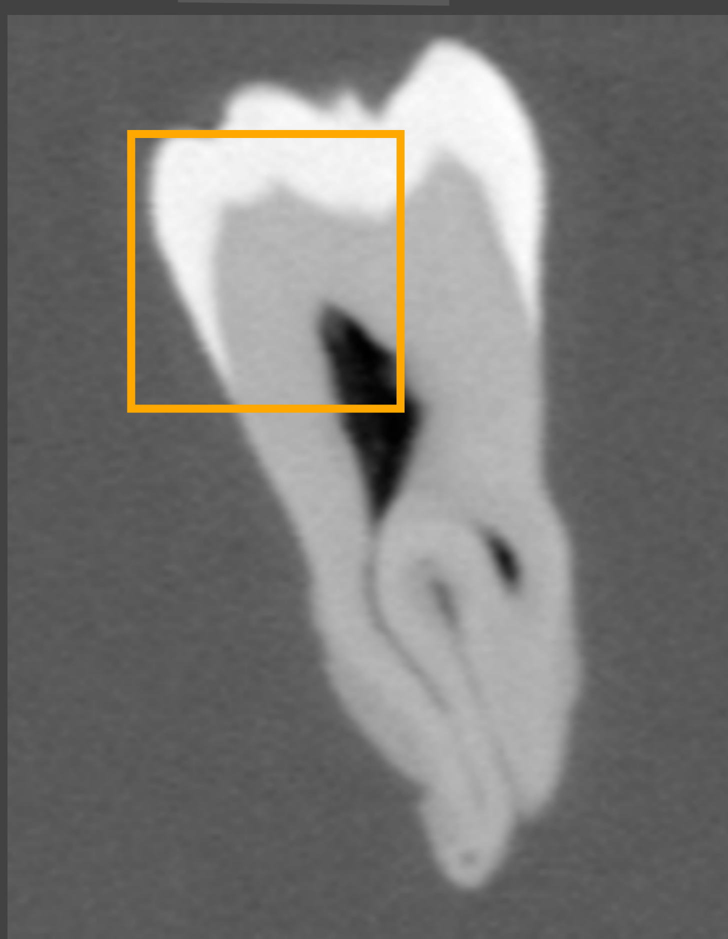
Human Tooth CT



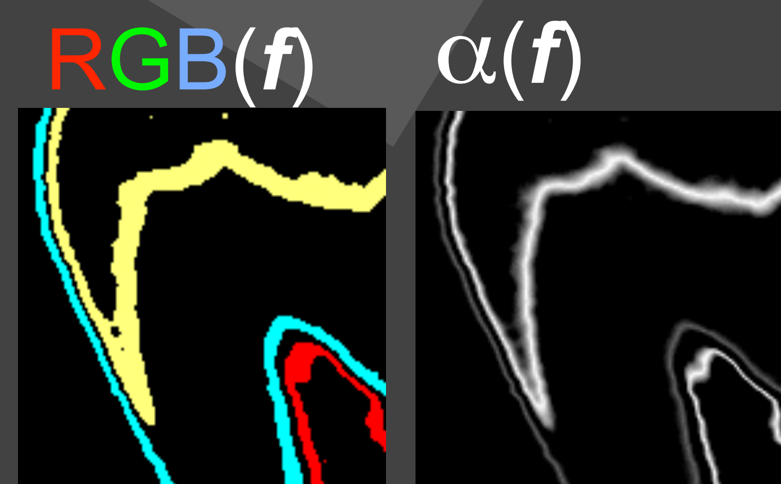
Transfer Functions (TFs)

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





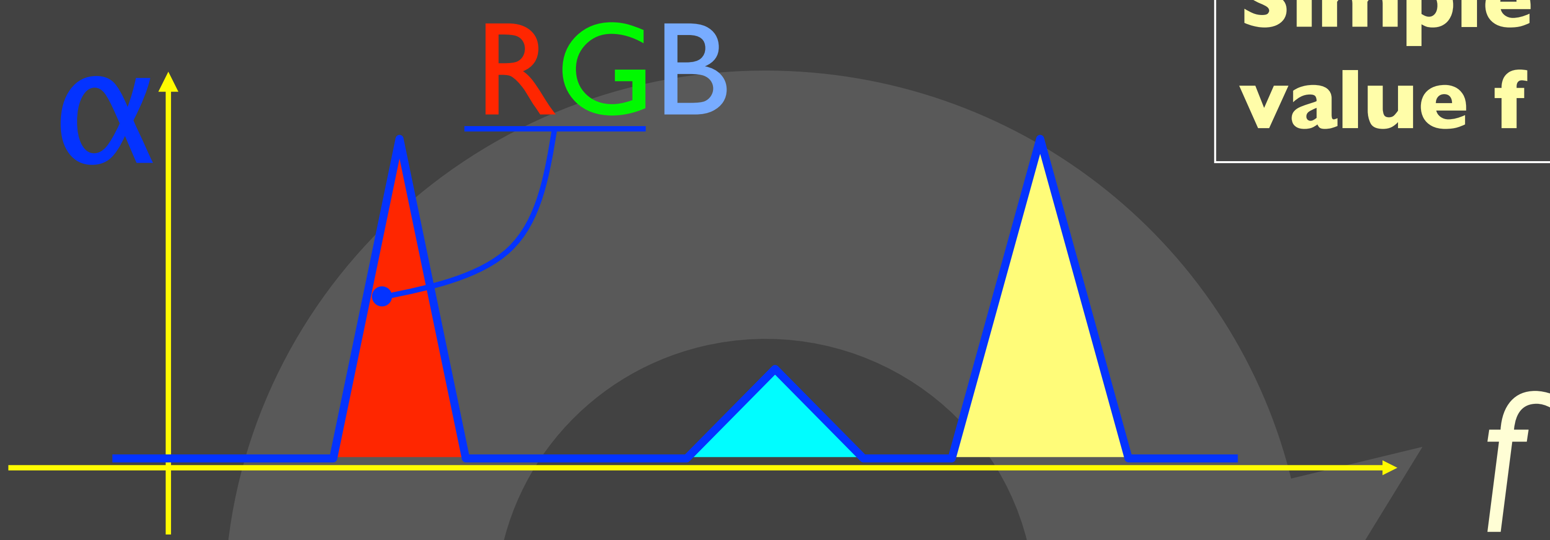
Human Tooth CT

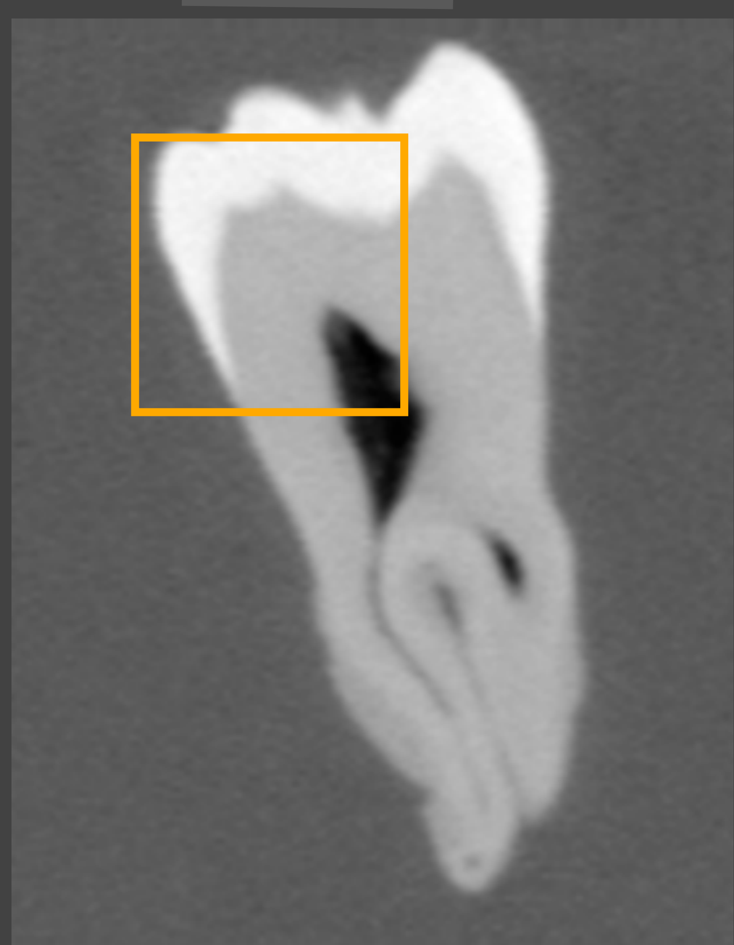


Shading,
Compositing…

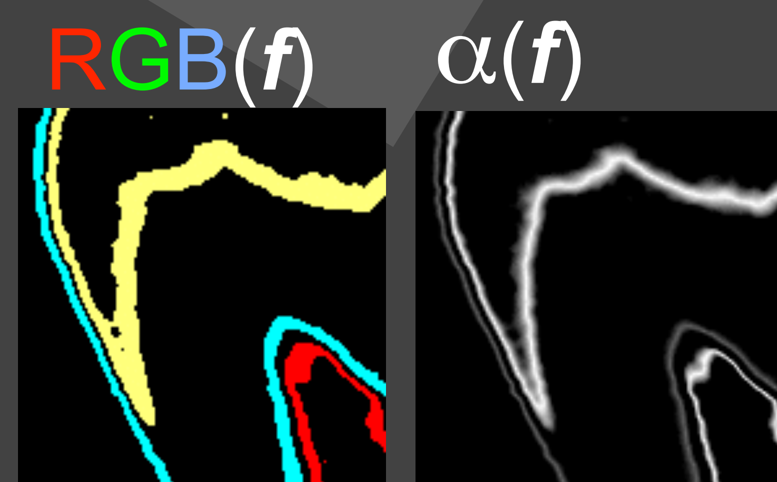
Transfer Functions (TFs)

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

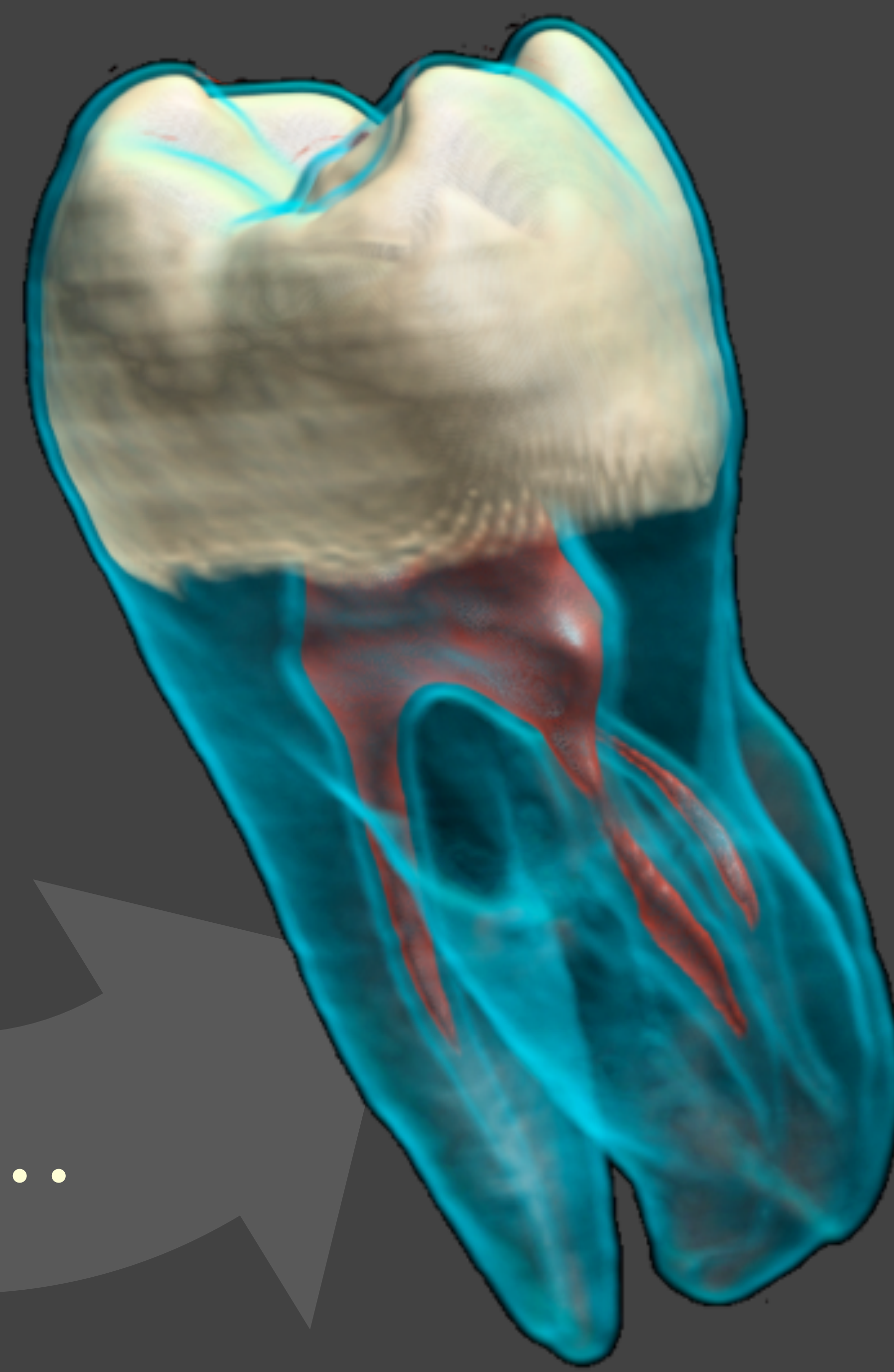




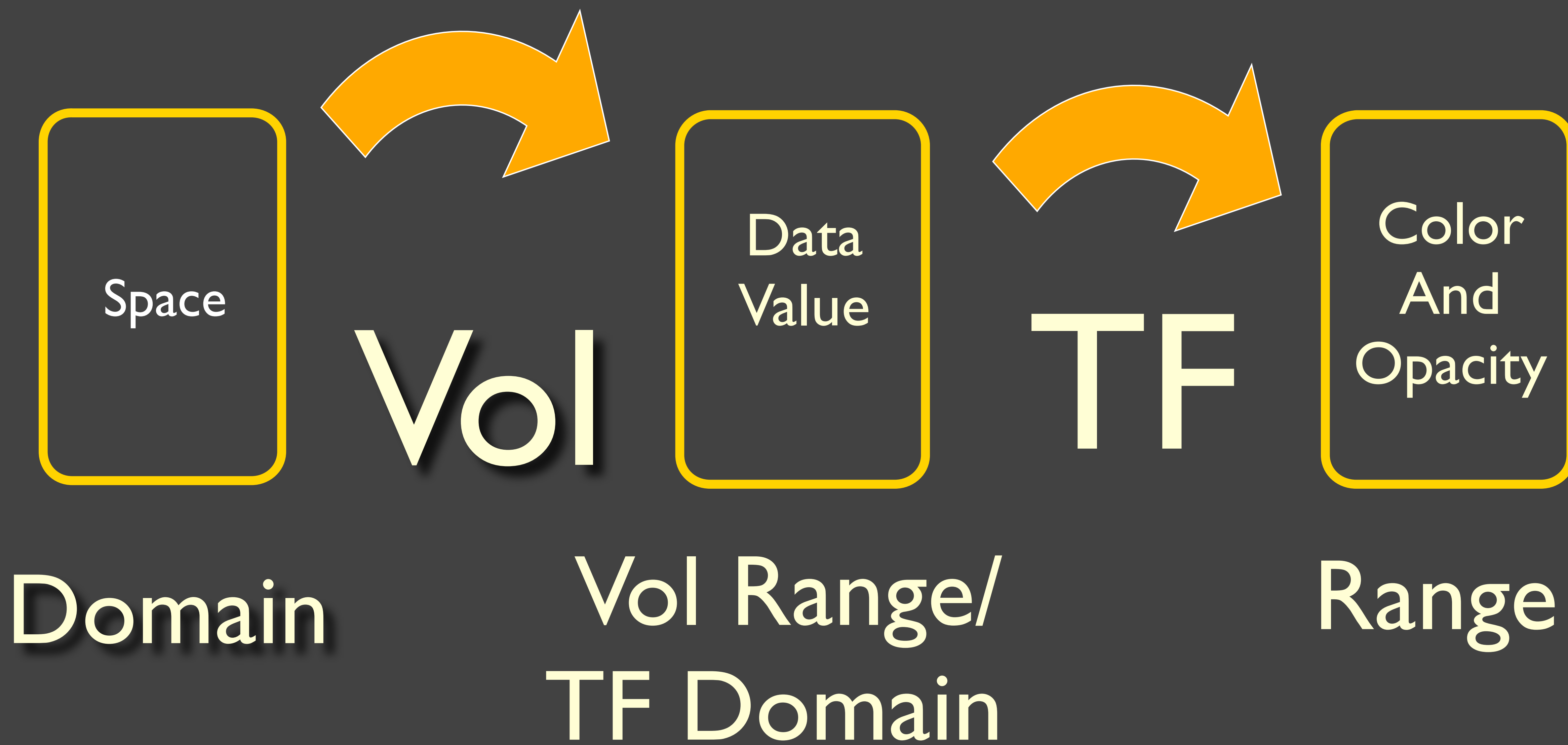
Human Tooth CT



Shading, Compositing…



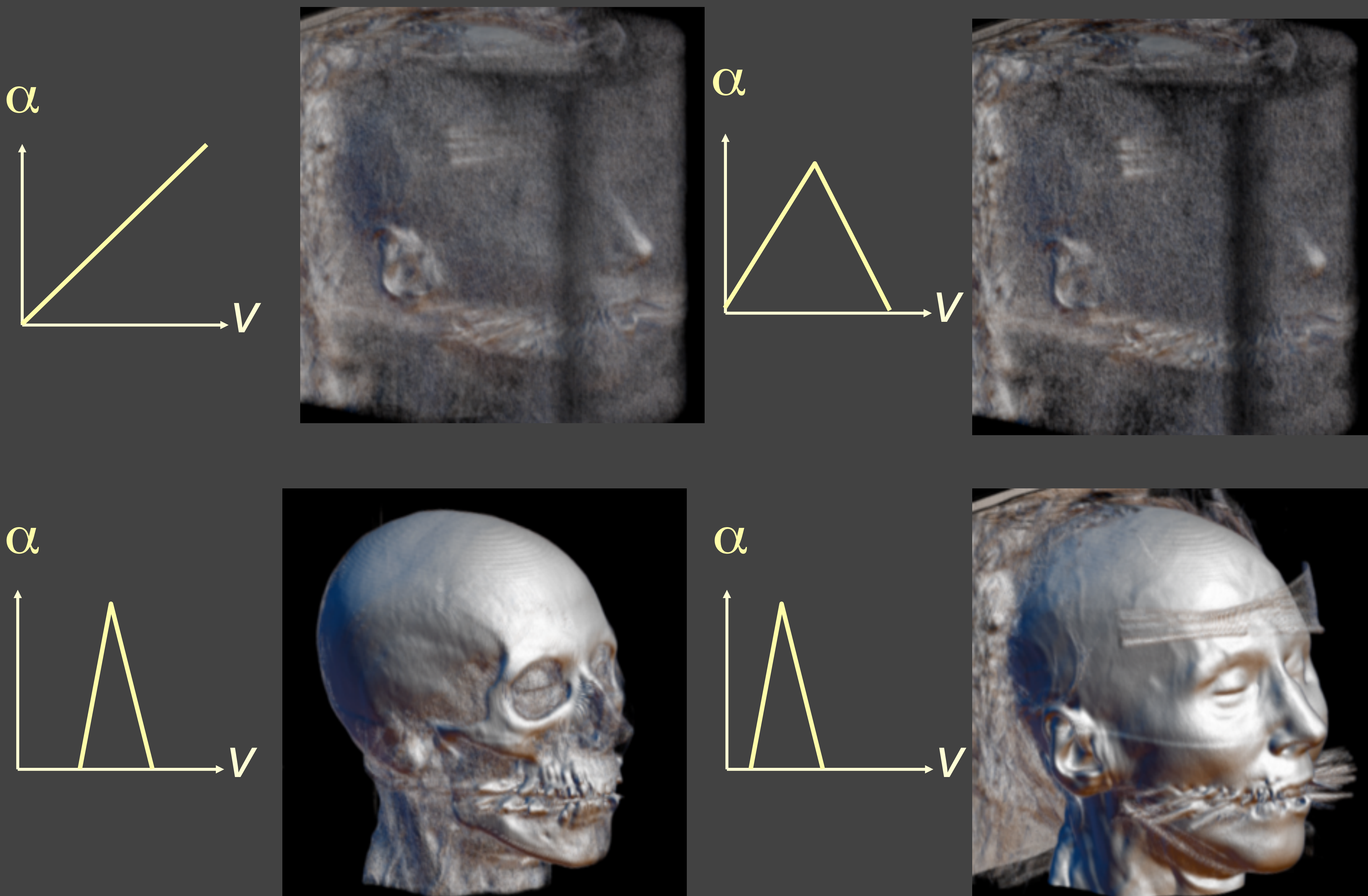
Basic Transfer Functions:



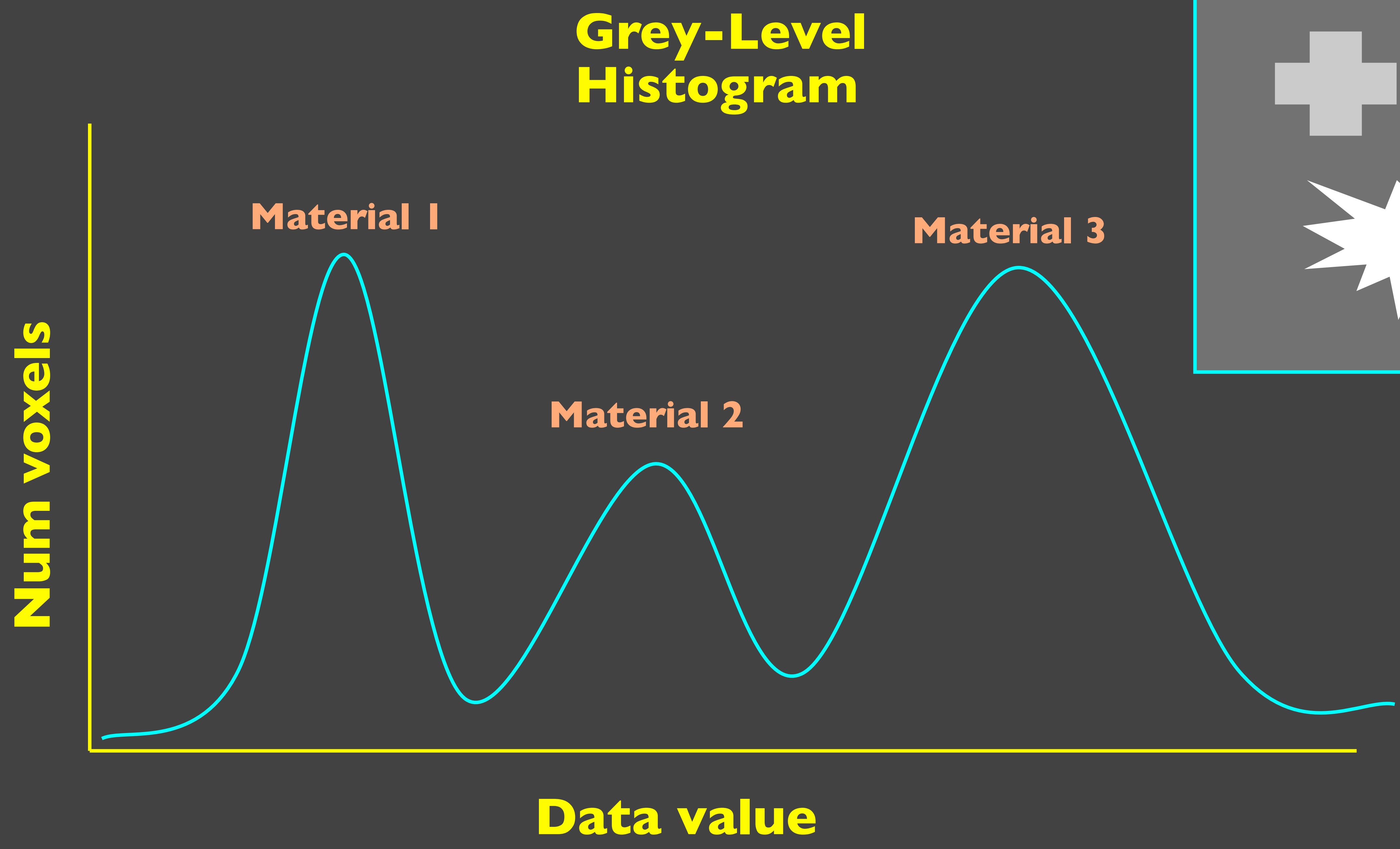
What Can Be Controlled by the Transfer Function?

- 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

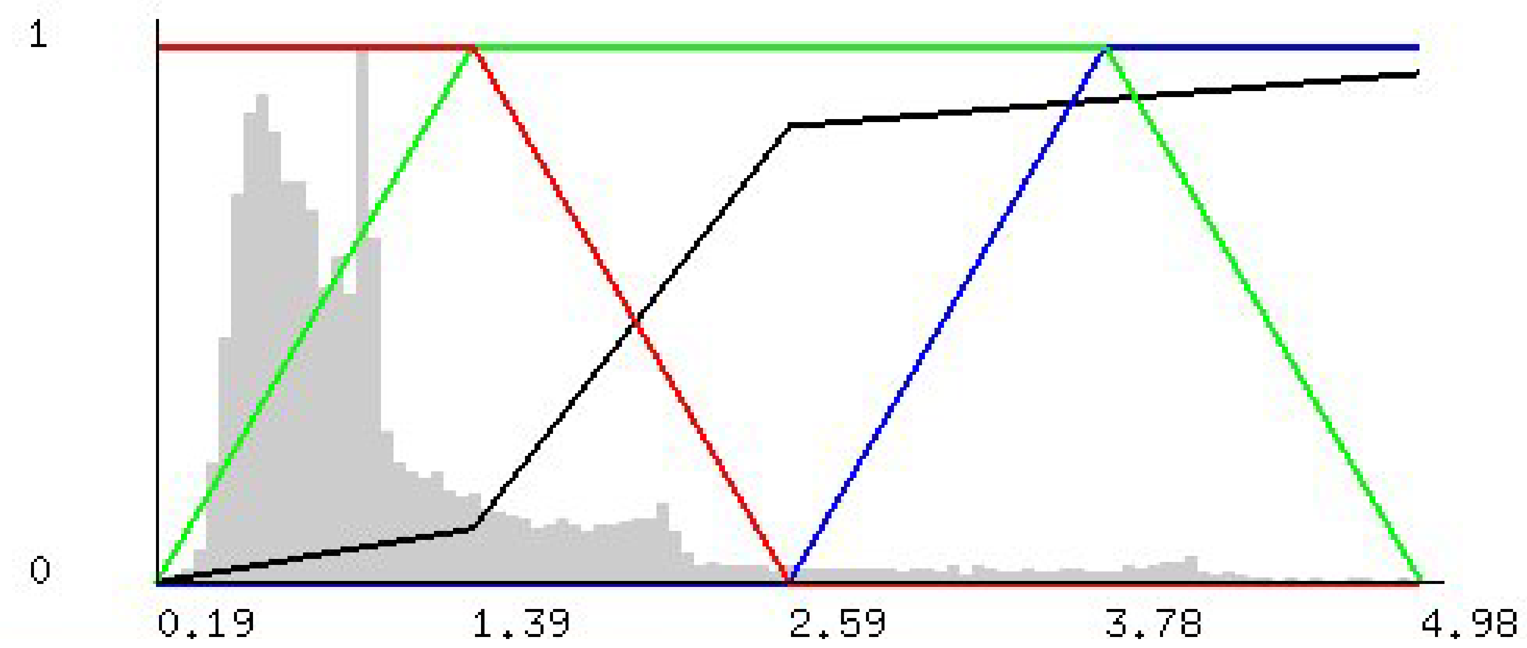
Setting Transfer Function: Hard



Volumes as Consisting of Materials

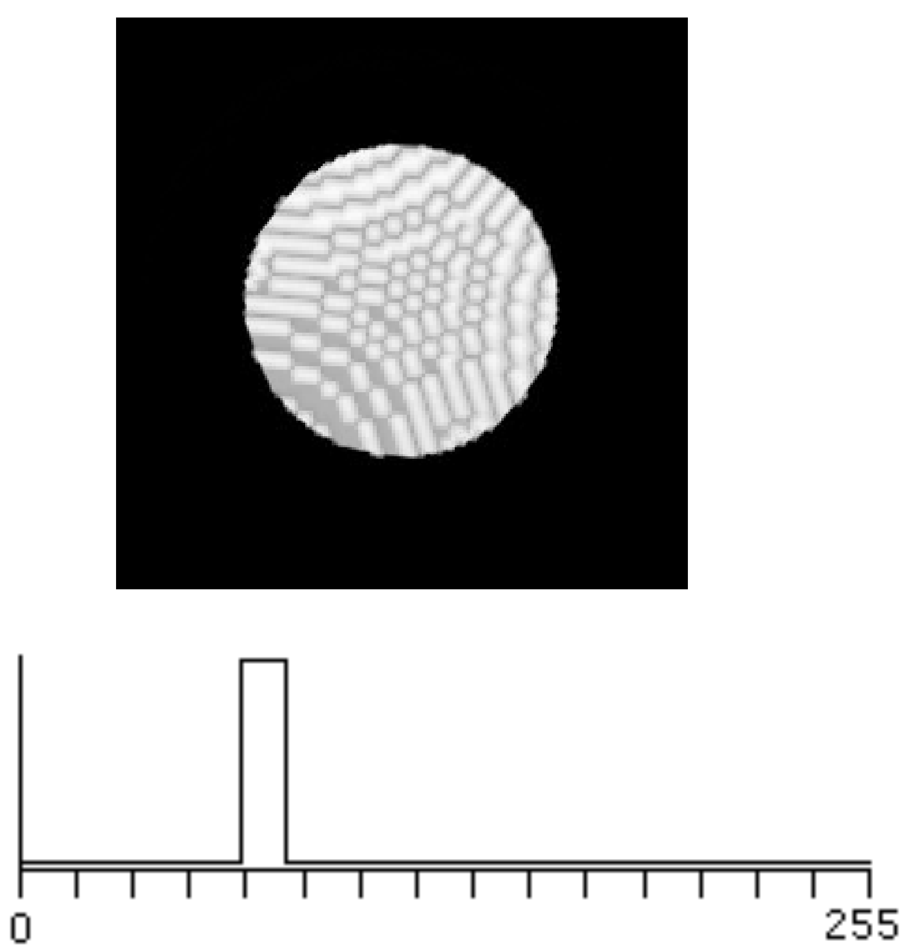


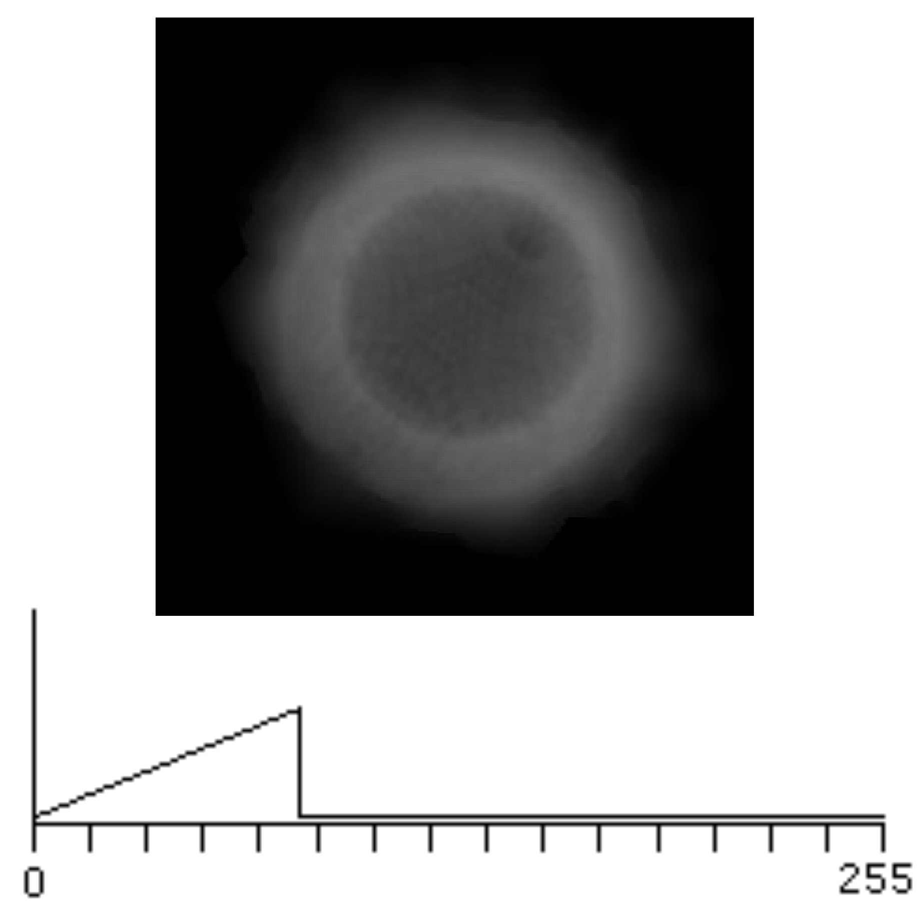
Transfer Function



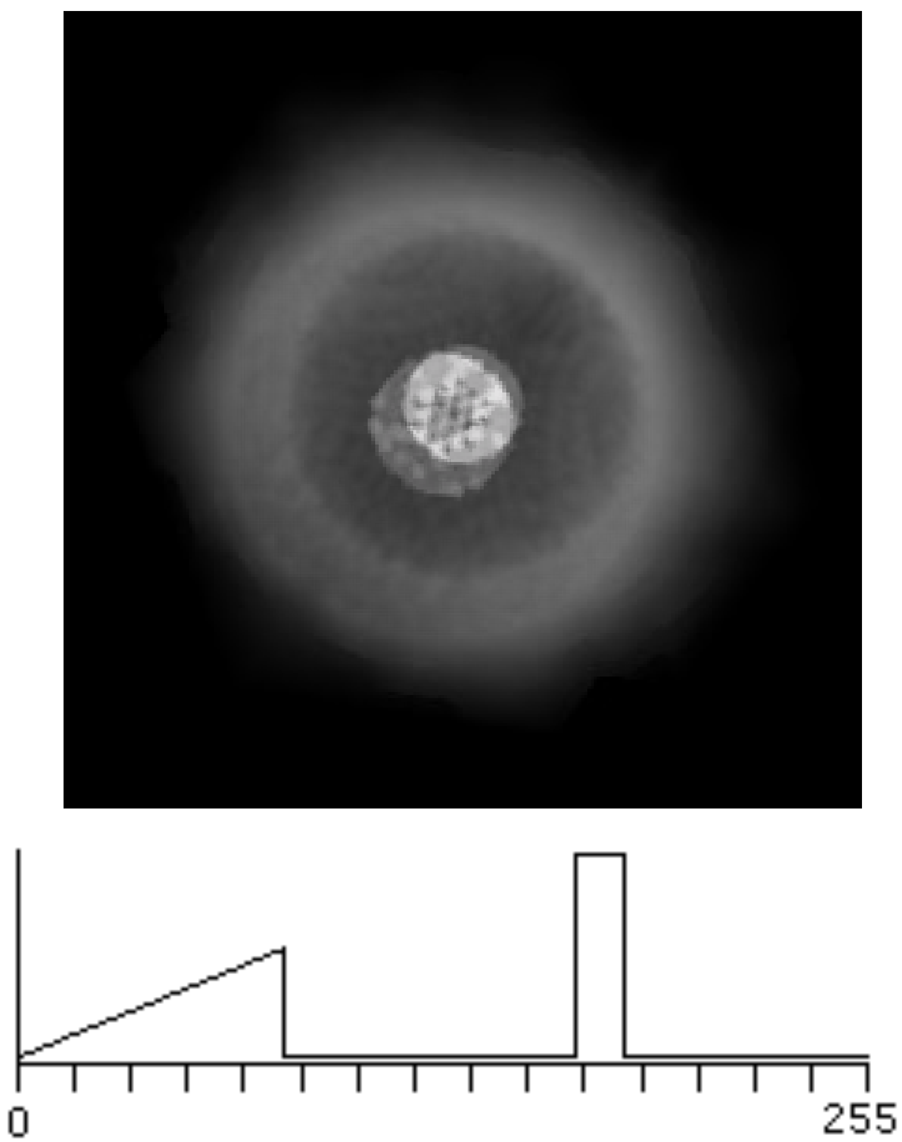
- RGB components
- Opacity
- Histogram helps in designing transfer function

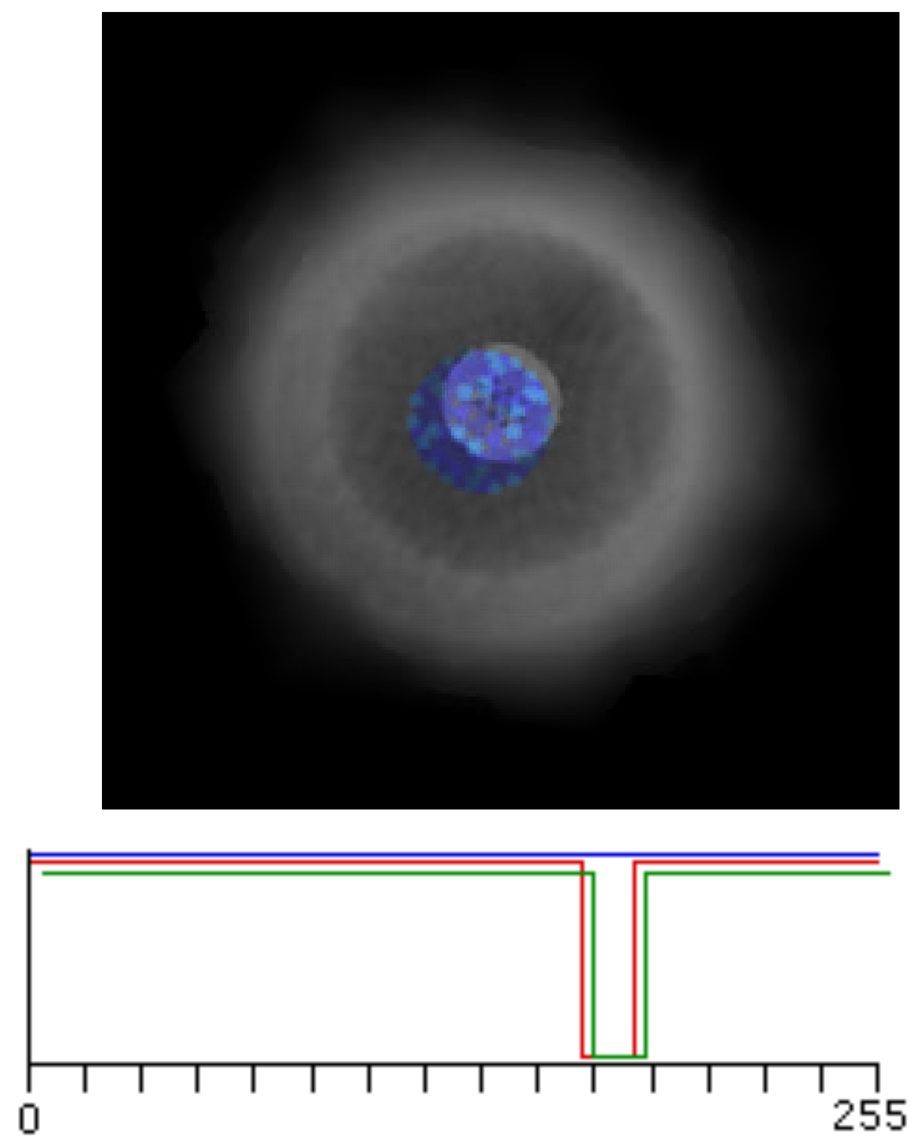
Transfer Function



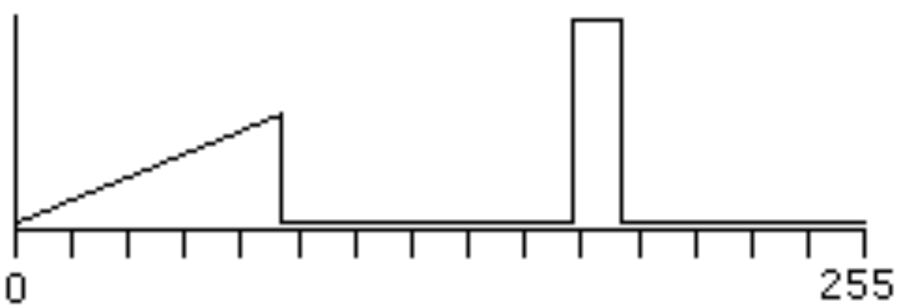


Transfer Function

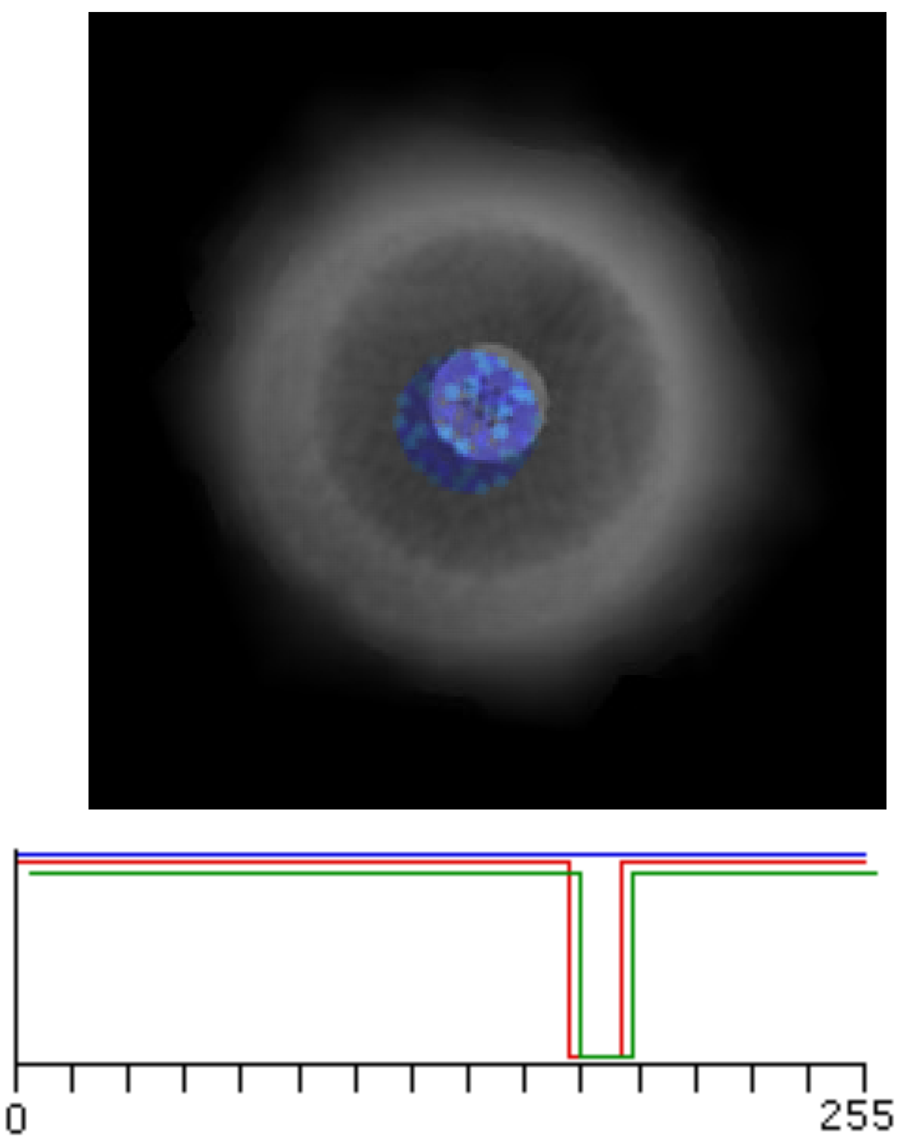


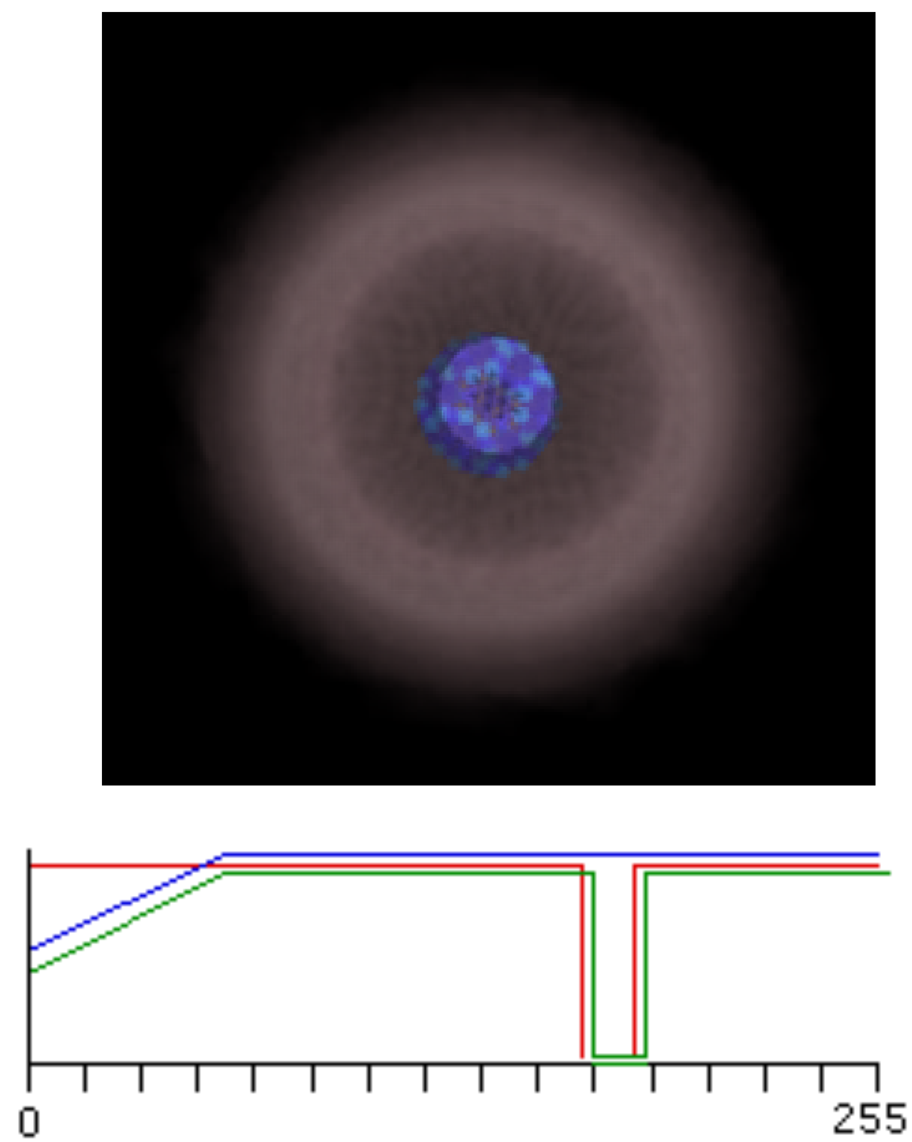


Transfer Function



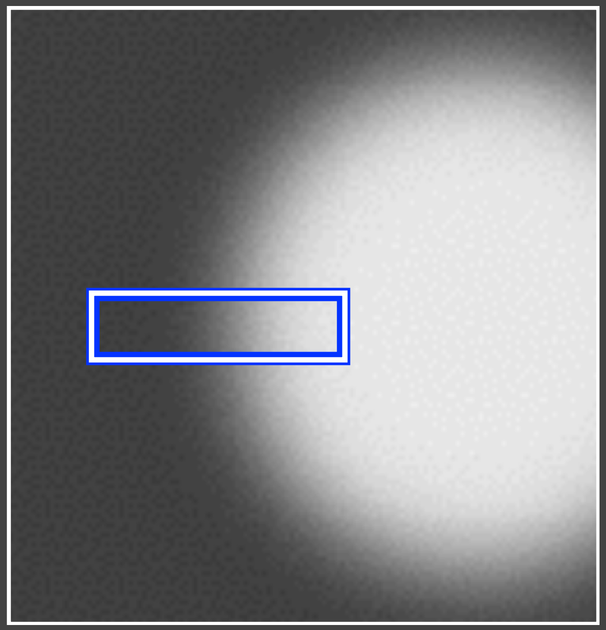
Different colors, same opacity

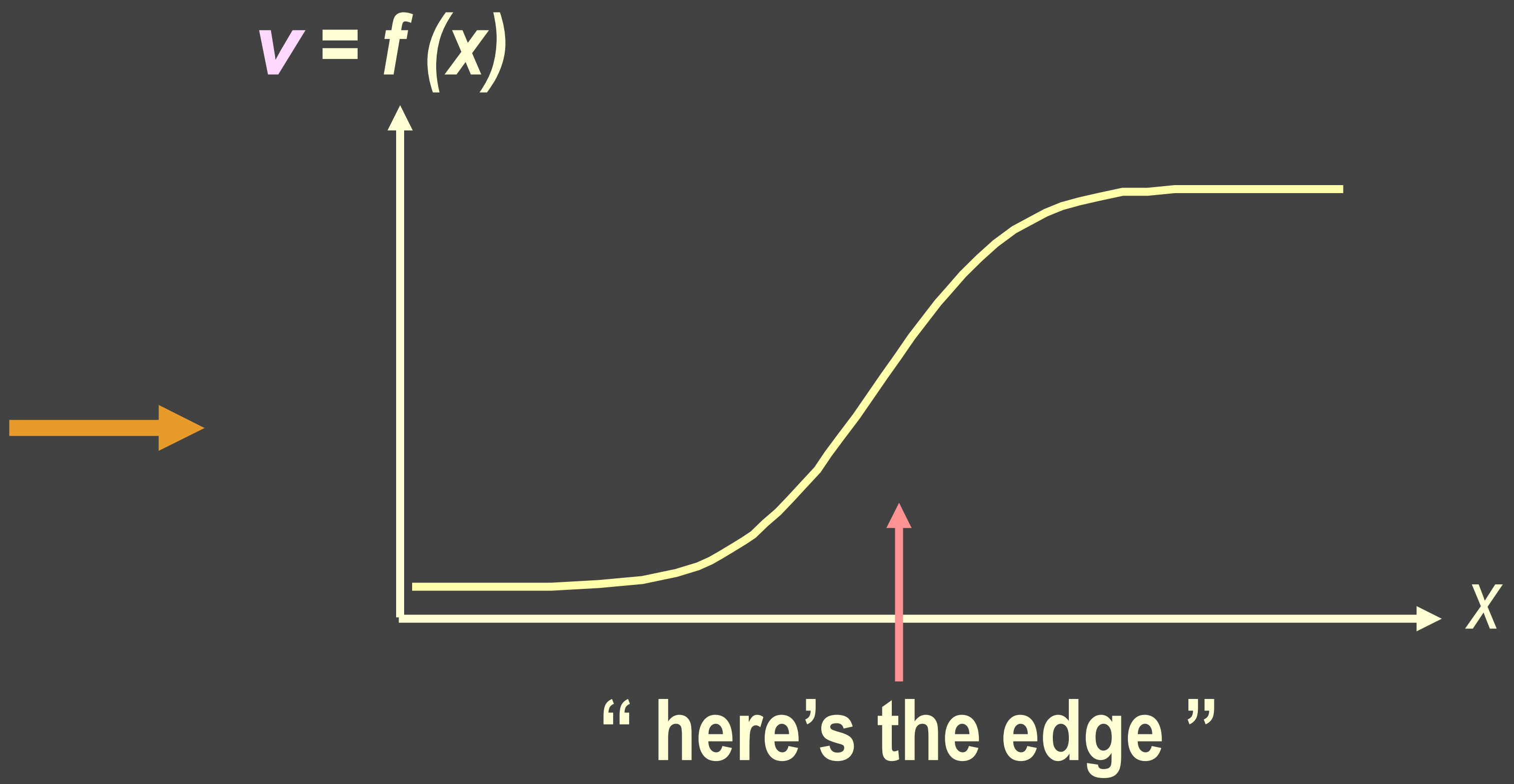




Finding edges: easy

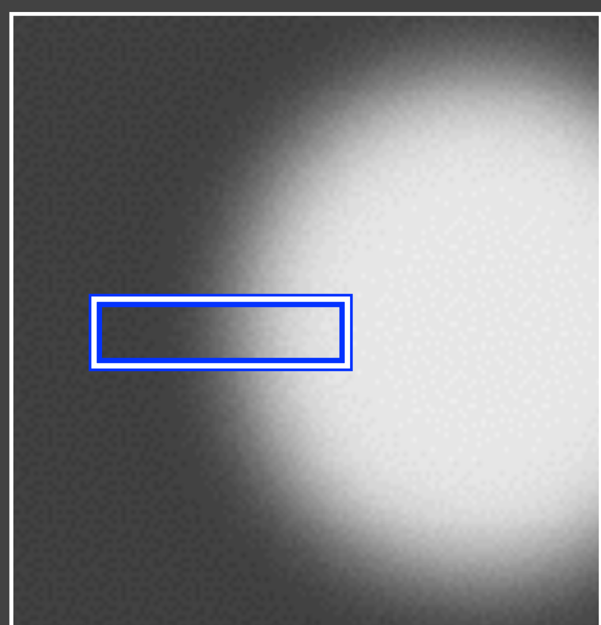
“Where’s the edge?”

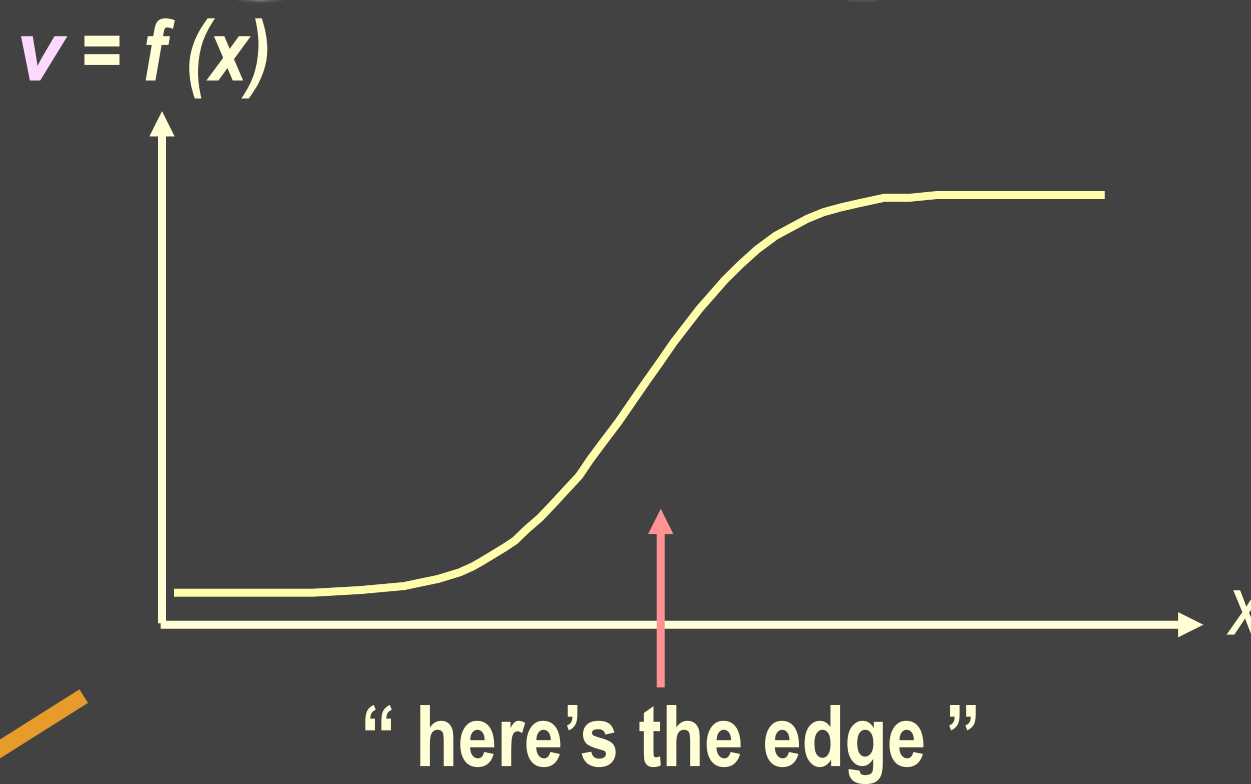


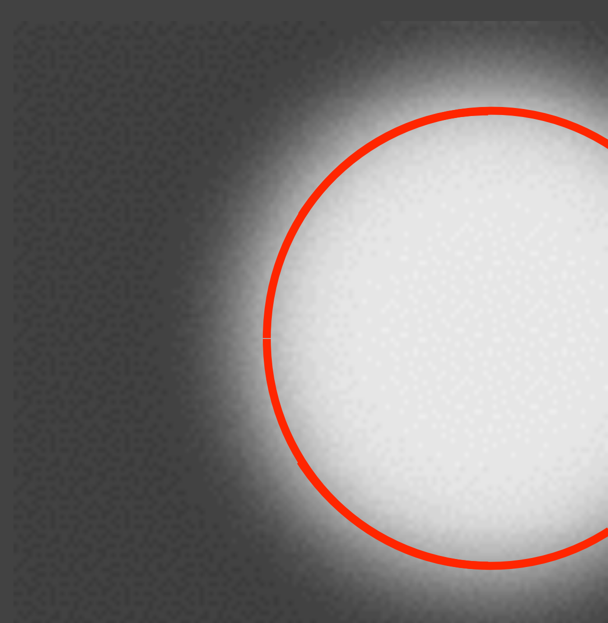


Finding edges: easy

"Where's the edge?"



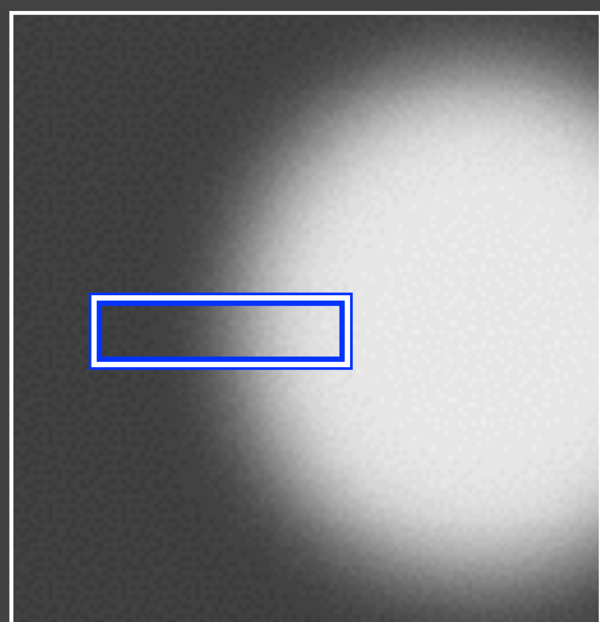


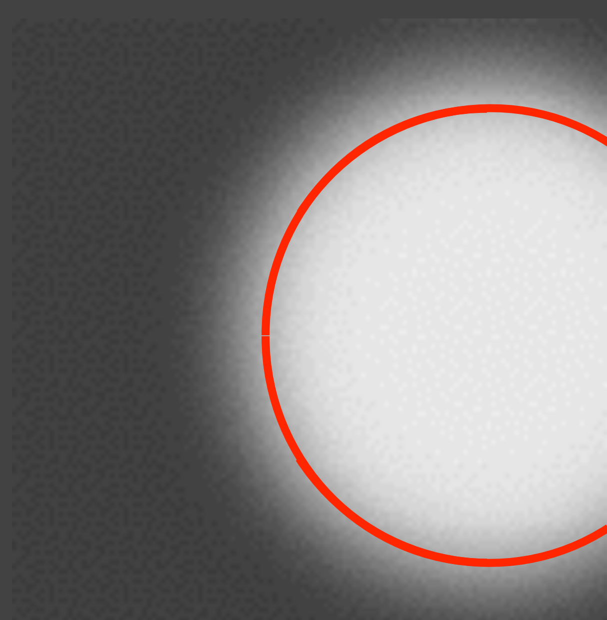


Result: edge pixels

Finding edges: easy

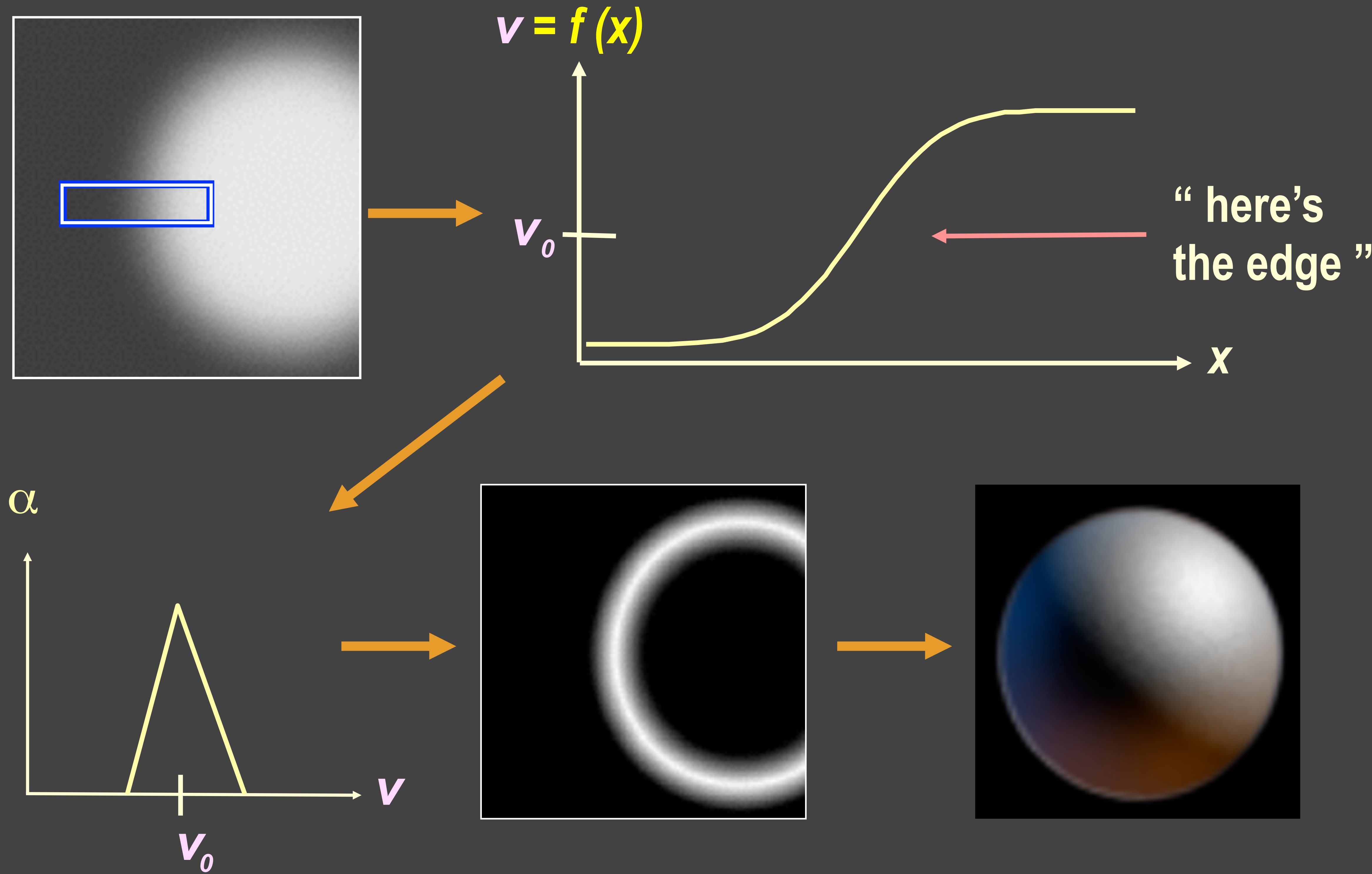
"Where's the edge?"



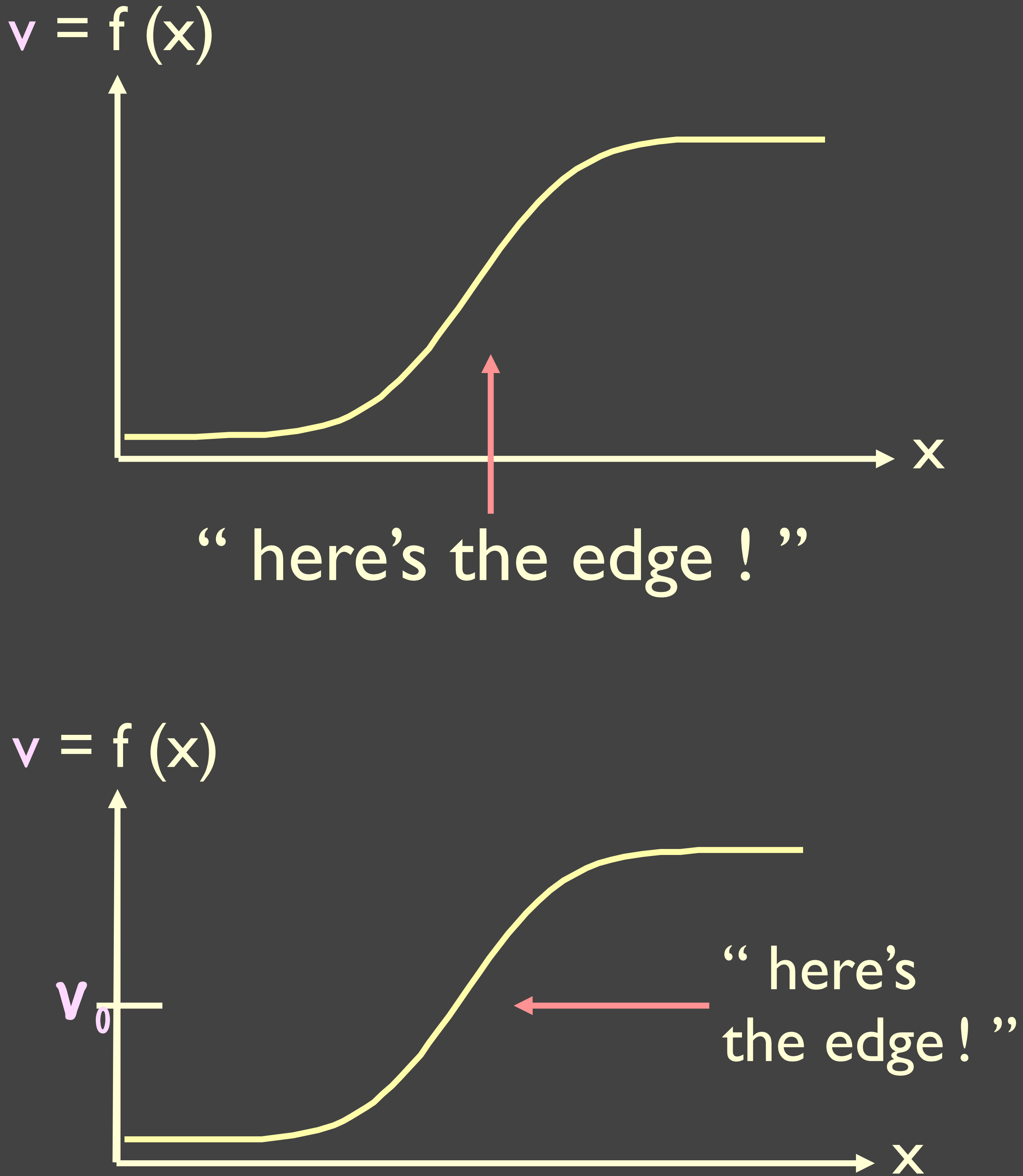


Result: edge pixels

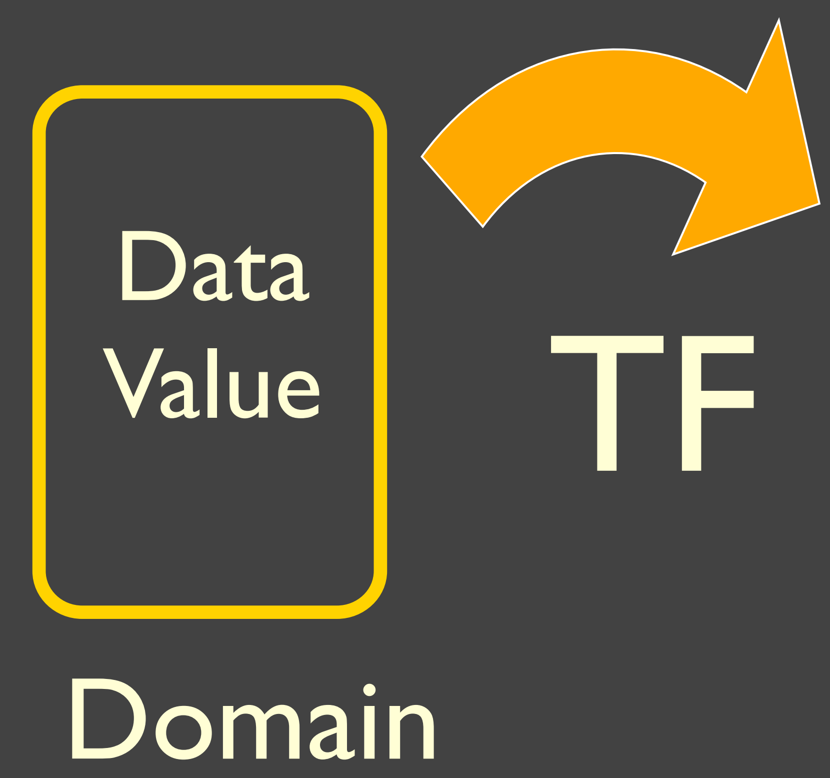
Transfer function Unintuitive



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

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

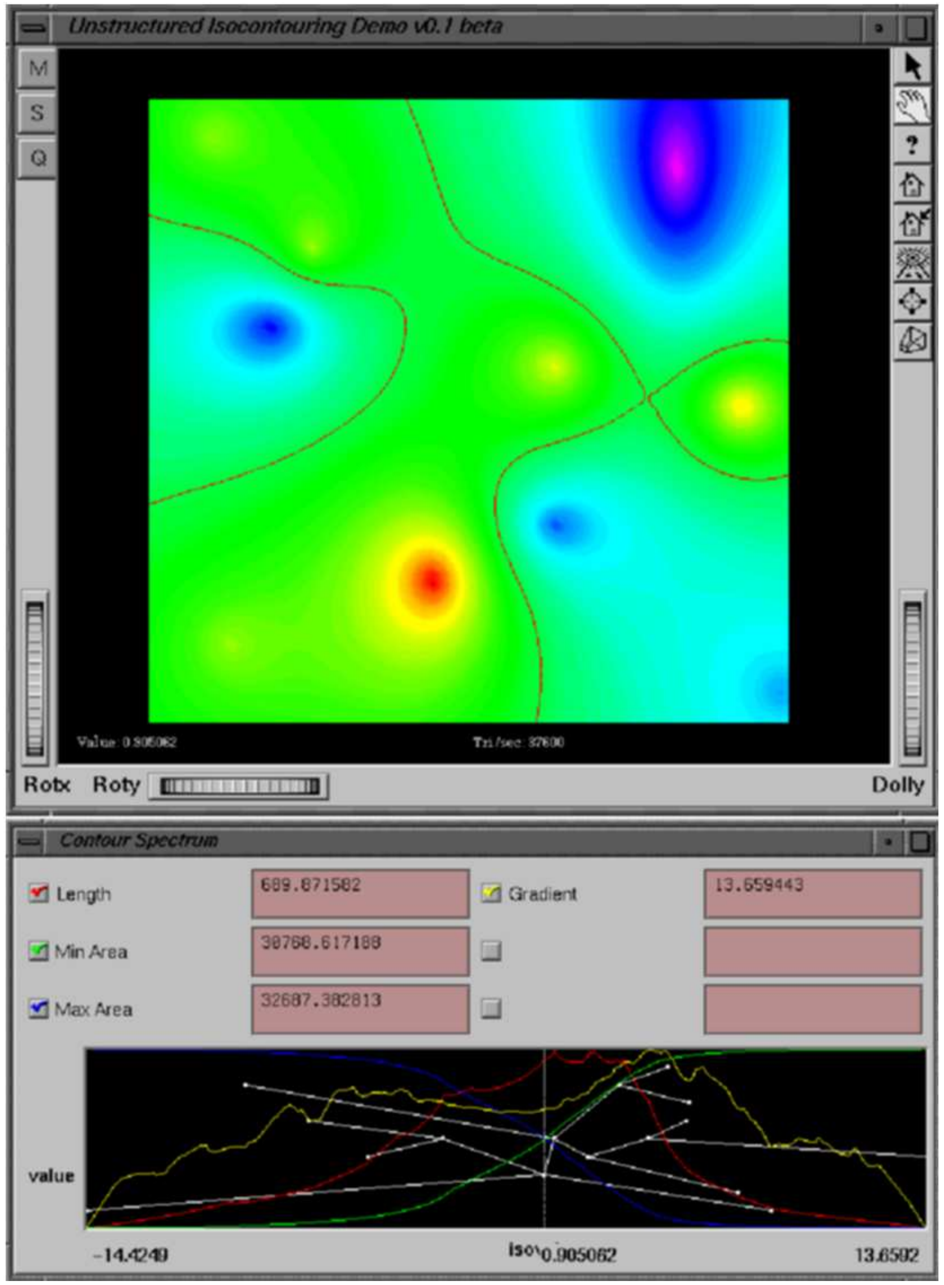
1. Salient Isovalues:
   - Contour Spectrum (Bajaj, Pascucci, Schikore: Vis ’97)
   - Statistical Signatures (“Salient Iso-Surface Detection Through Model-Independent Statistical Signatures”, Tenginaki, Lee, Machiraju: Vis ’01)
   - Other computational methods (“Fast Detection of Meaningful Isosurfaces 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?

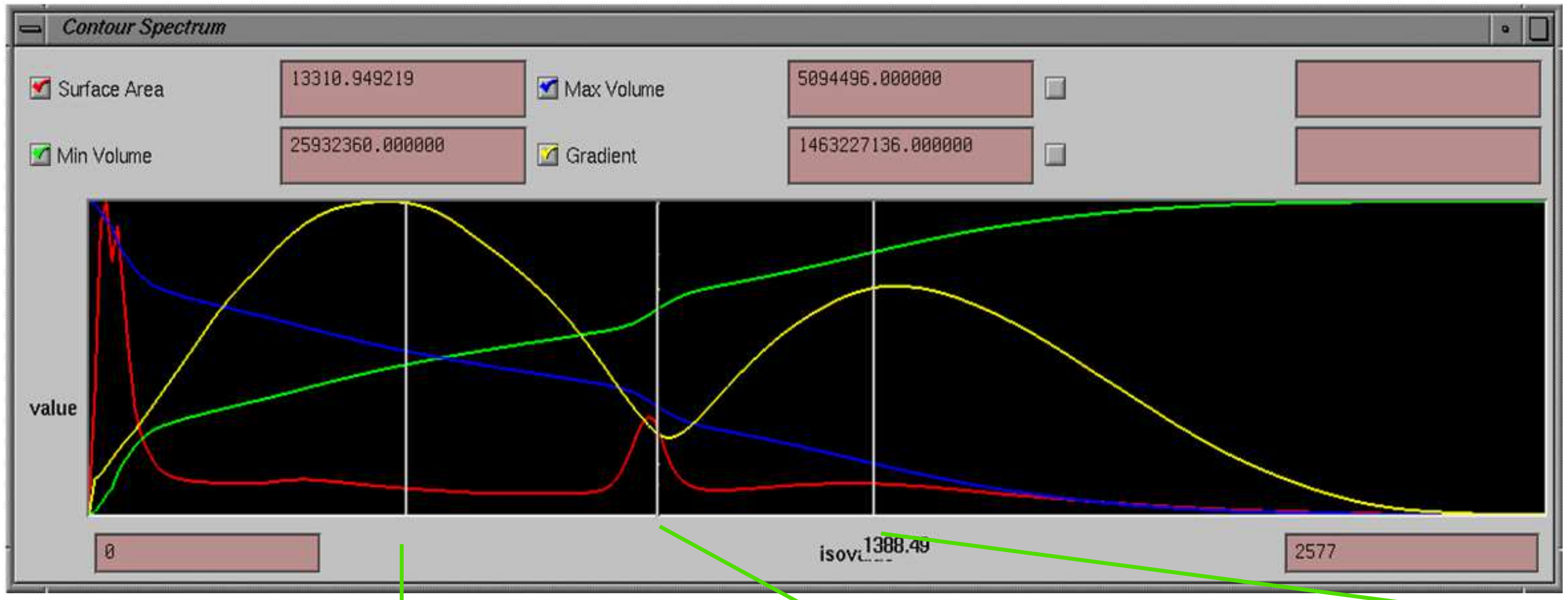
Contour Spectrum (Bajaj, Pascucci, Schikore: Vis ’97; Transfer Function Bake-Off: Vis ’00)

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

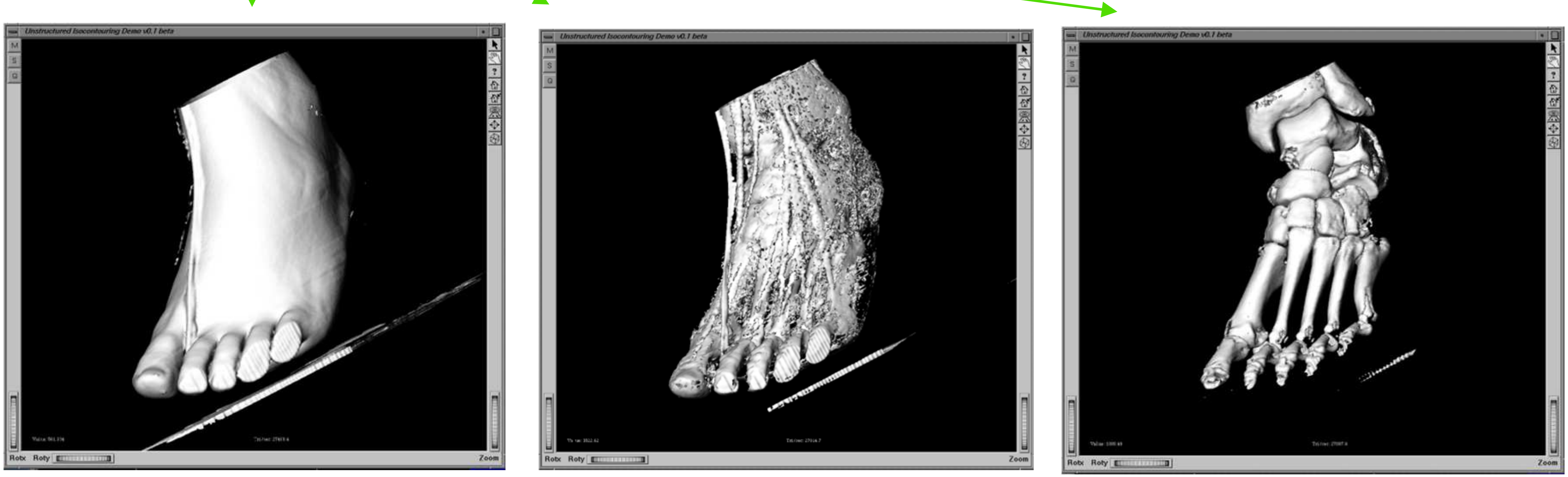


The Contour Spectrum

(colored lines correspond to different isosurface metrics)



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

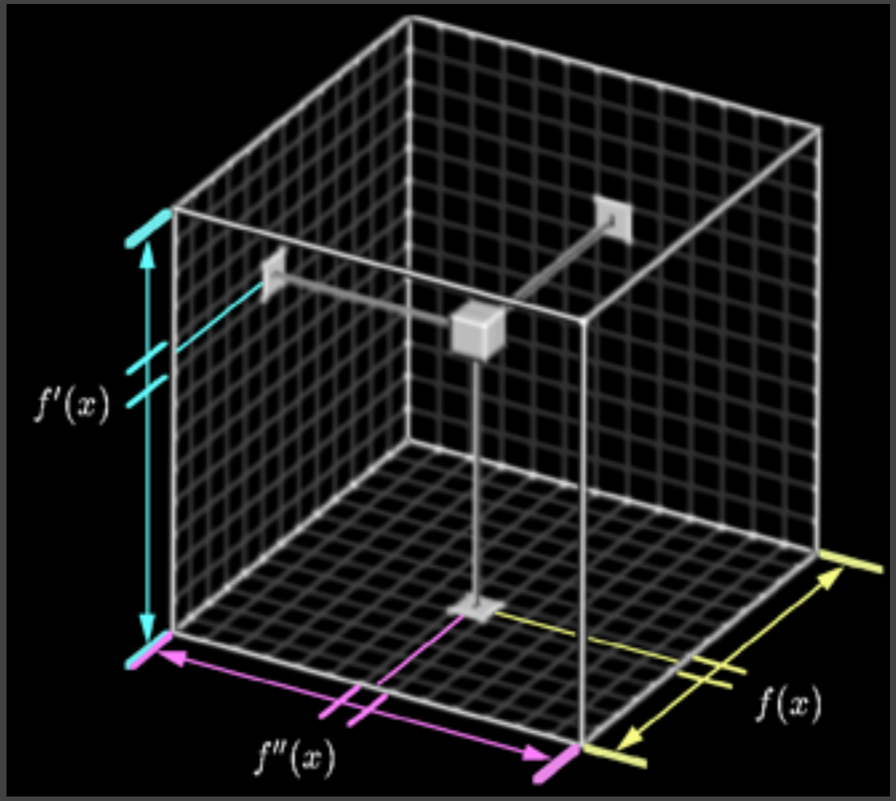


Use derivatives

Reasoning:

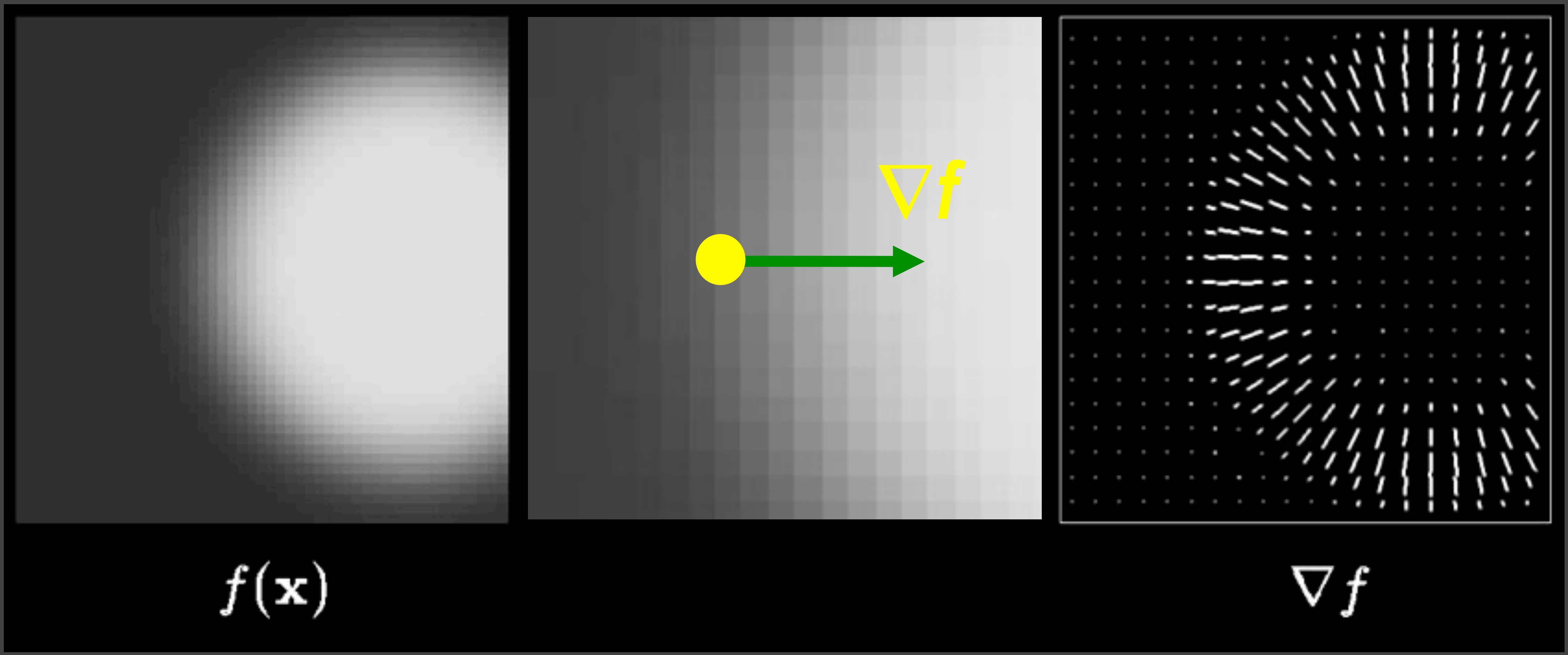
- TFs are volume-position invariant
- Histograms "project out" position
- Interested in boundaries between materials
- Boundaries characterized by derivatives

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



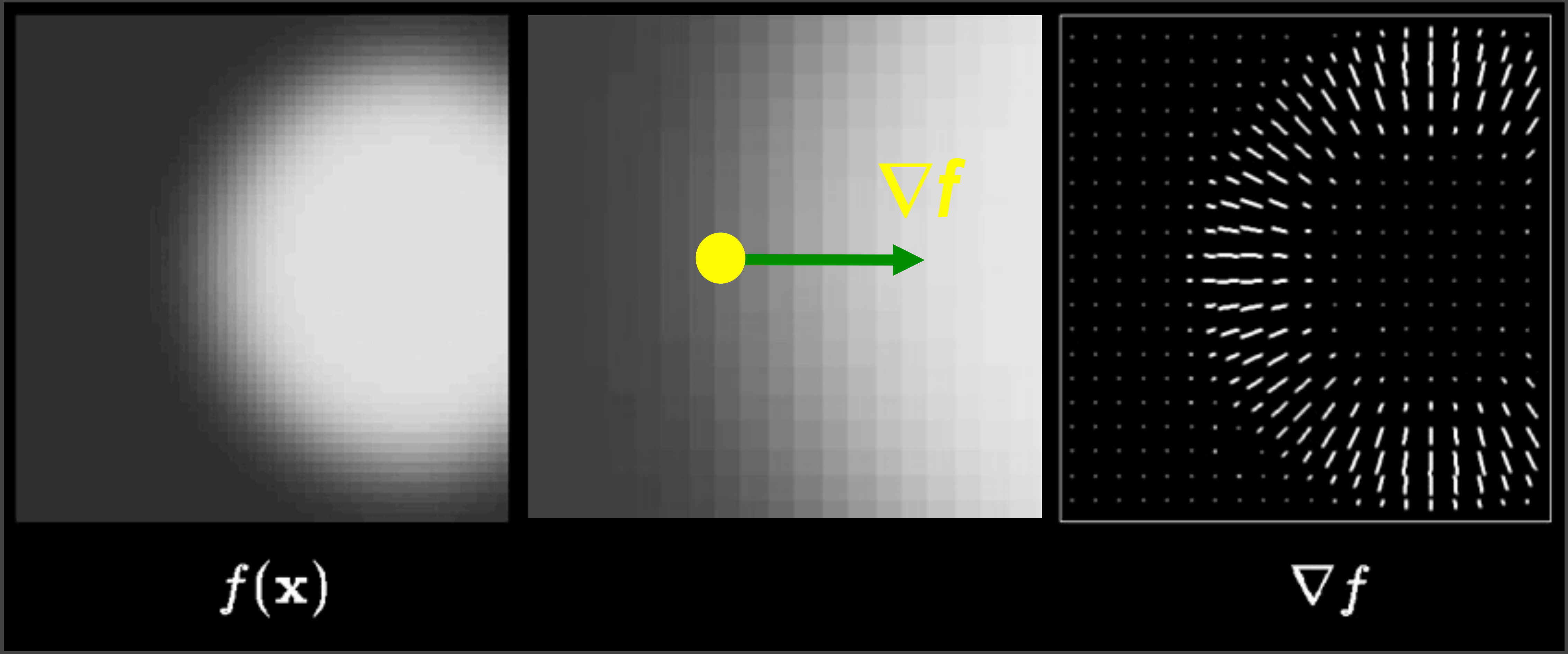
By (1) inspecting and (2) algorithmically analyzing histogram volume, we can create transfer functions

Gradient



Gradient

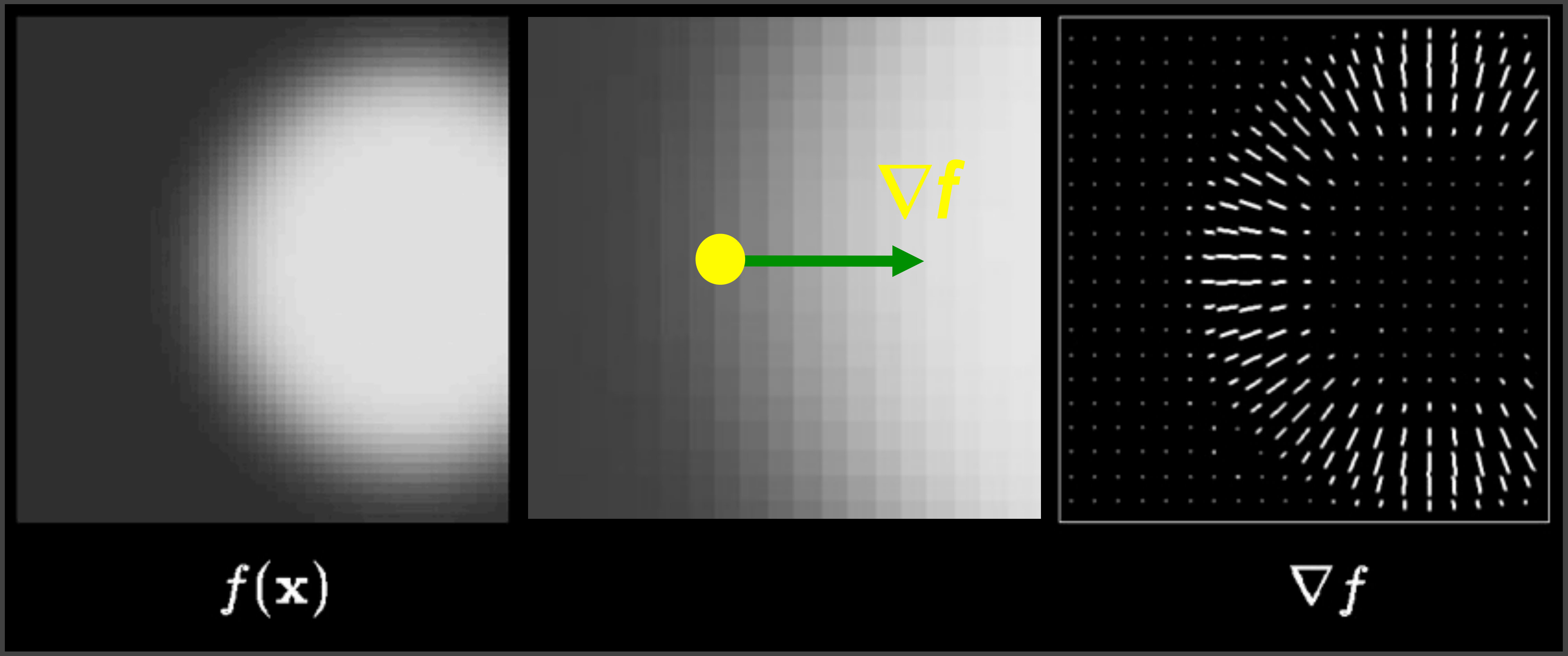
$\nabla f = (dx, dy, dz)$



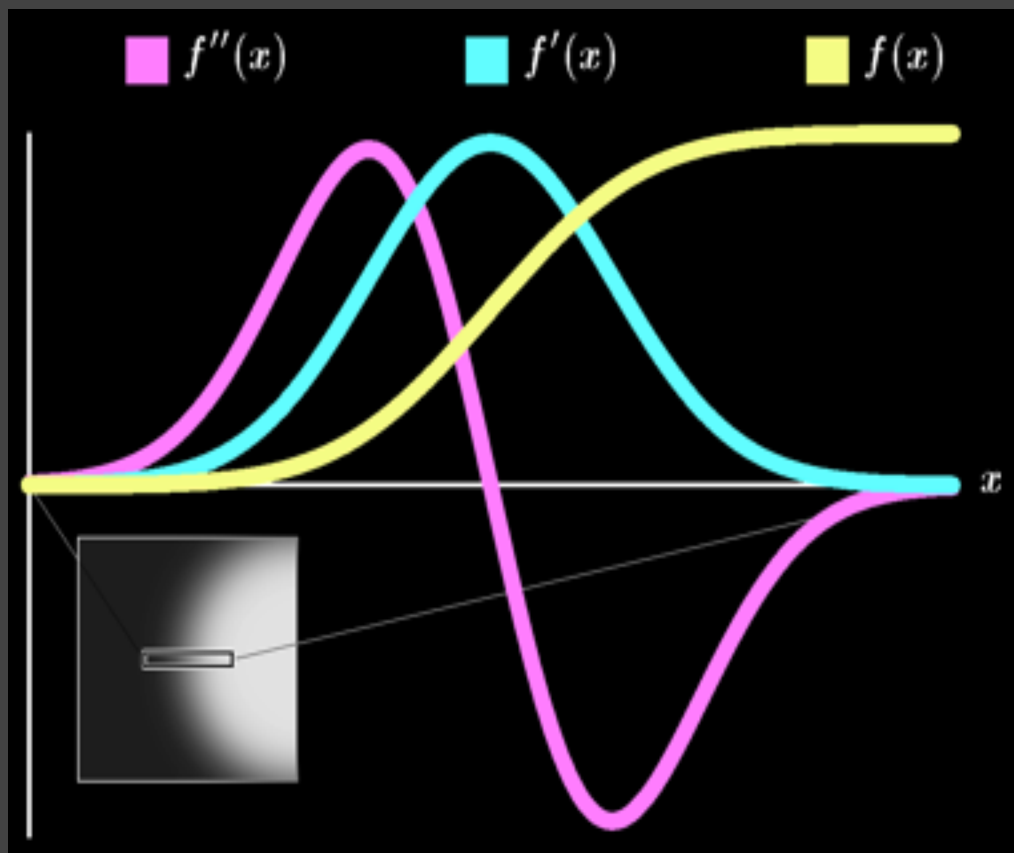
Gradient

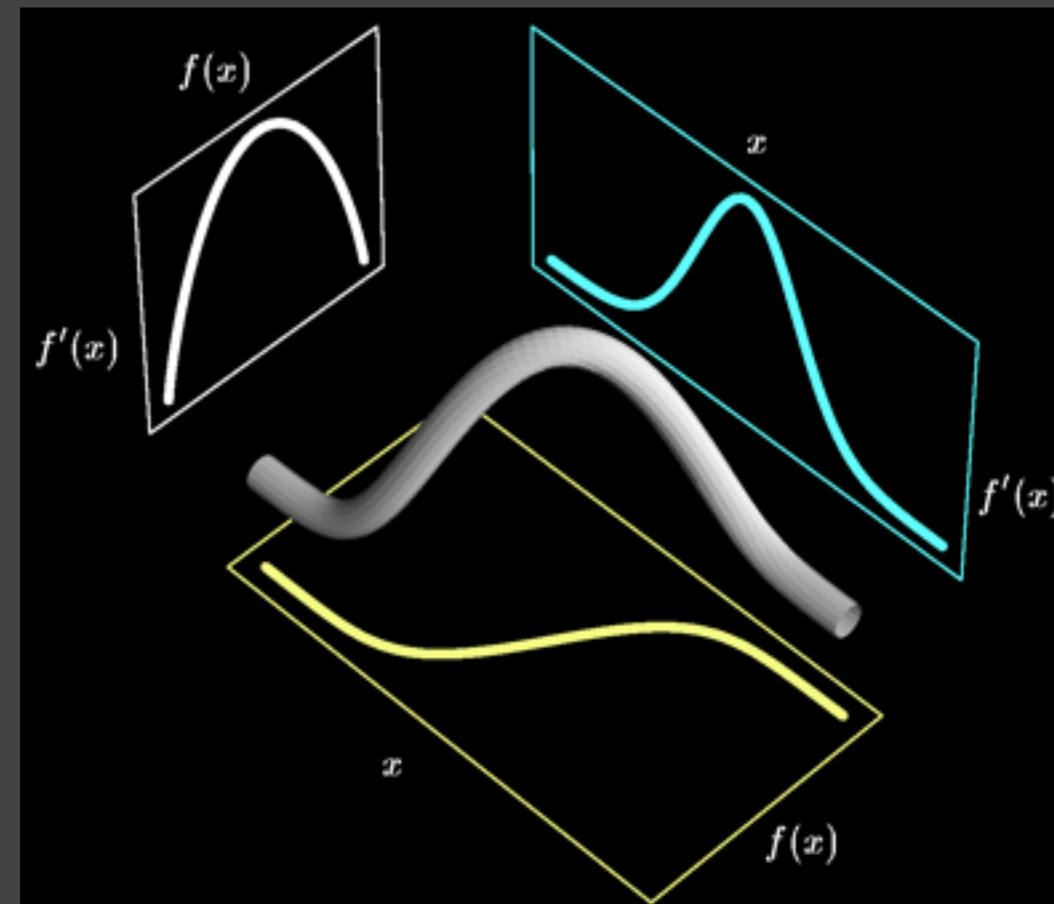
$$\nabla f = (dx, dy, dz)$$
$$= \left((f(1,0,0) - f(-1,0,0))/2, \; (f(0,1,0) - f(0,-1,0))/2, \; (f(0,0,1) - f(0,0,-1))/2 \right)$$

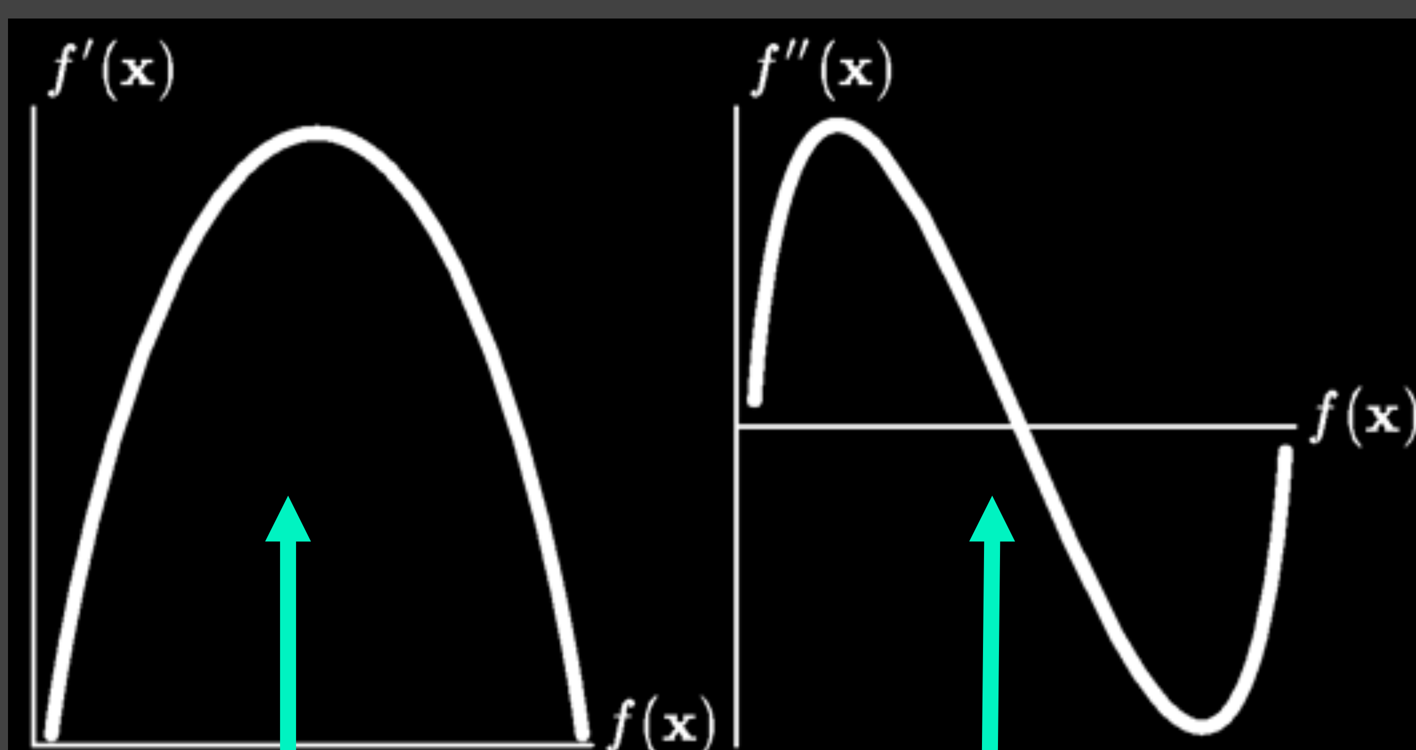
- Approximates "surface normal“ (of isosurface)



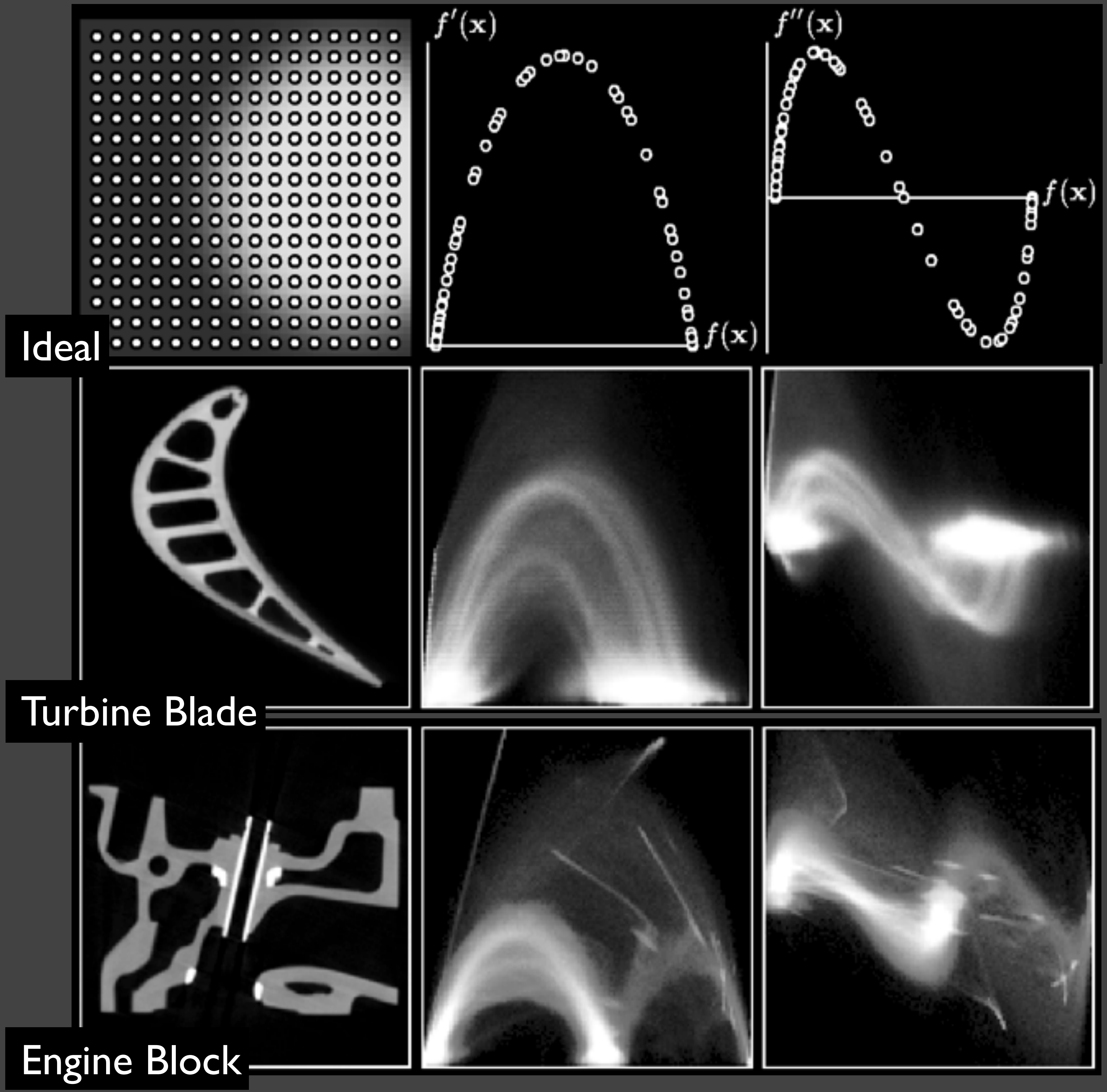
Derivative relationships







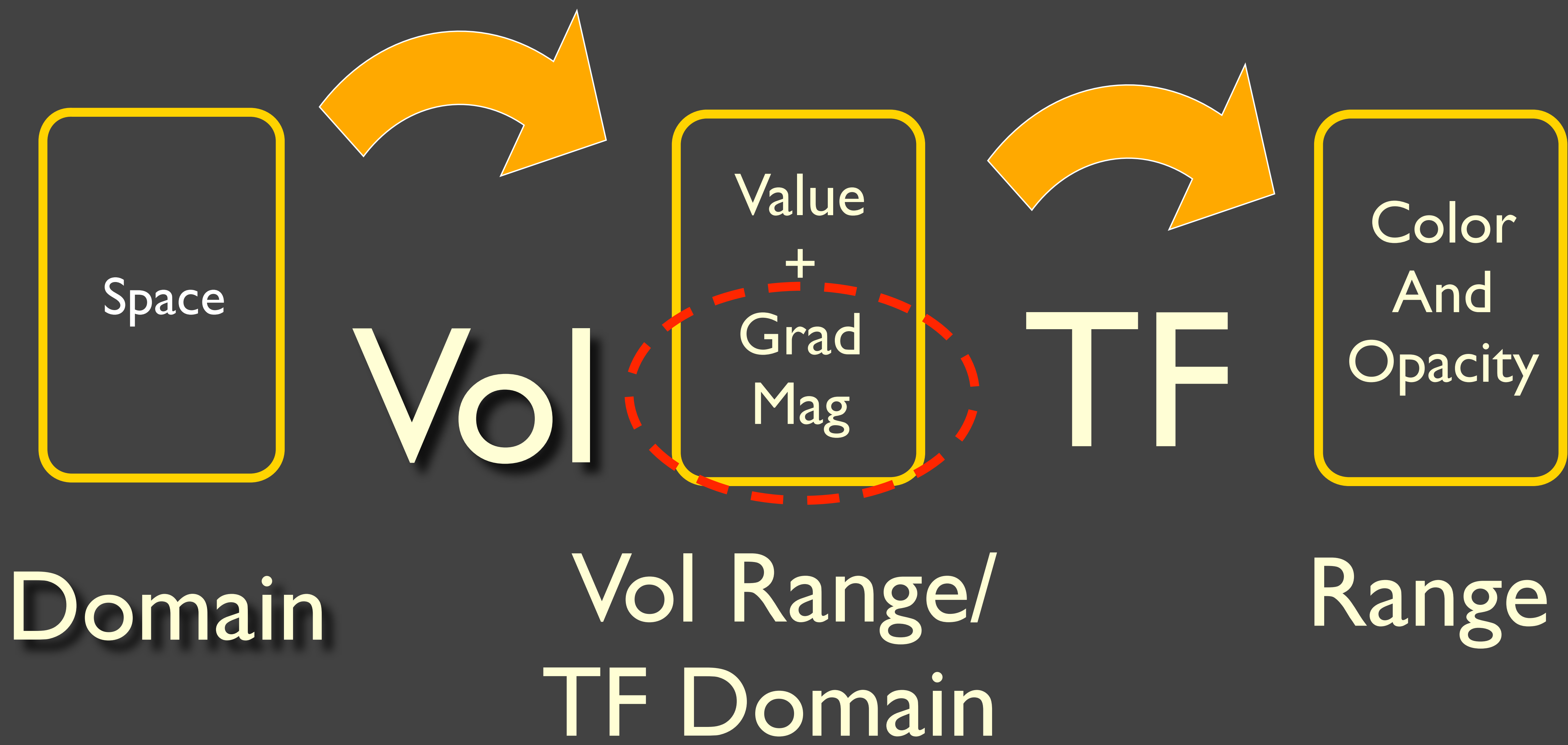
Edges at maximum of 1st derivative or zero-crossing of 2nd



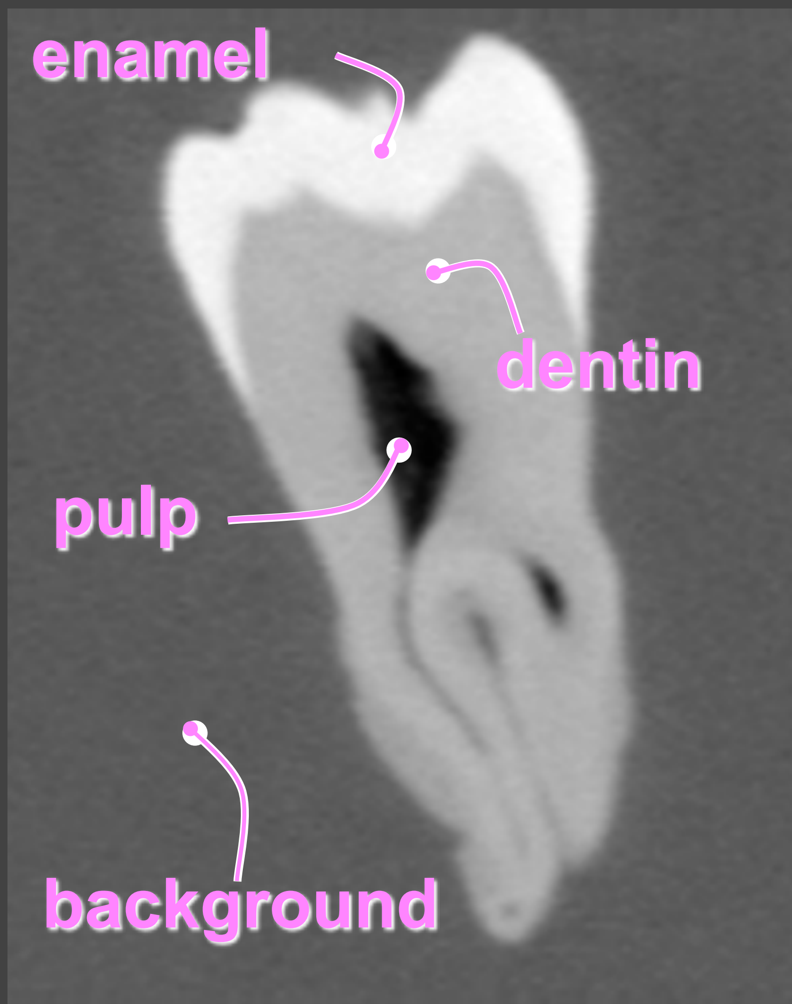
Project histogram volume to 2D scatterplots

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

Basic Transfer Functions:



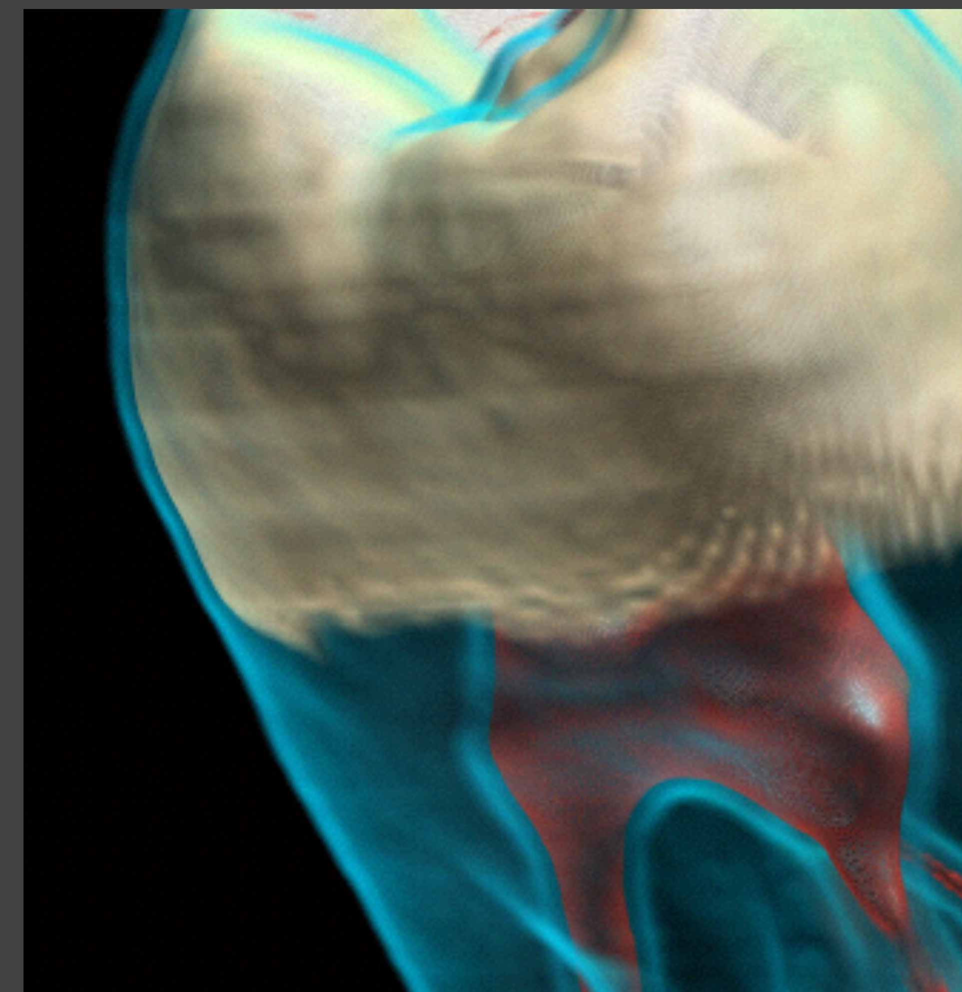
1D TFs: limitation



Slice

RGB(f)

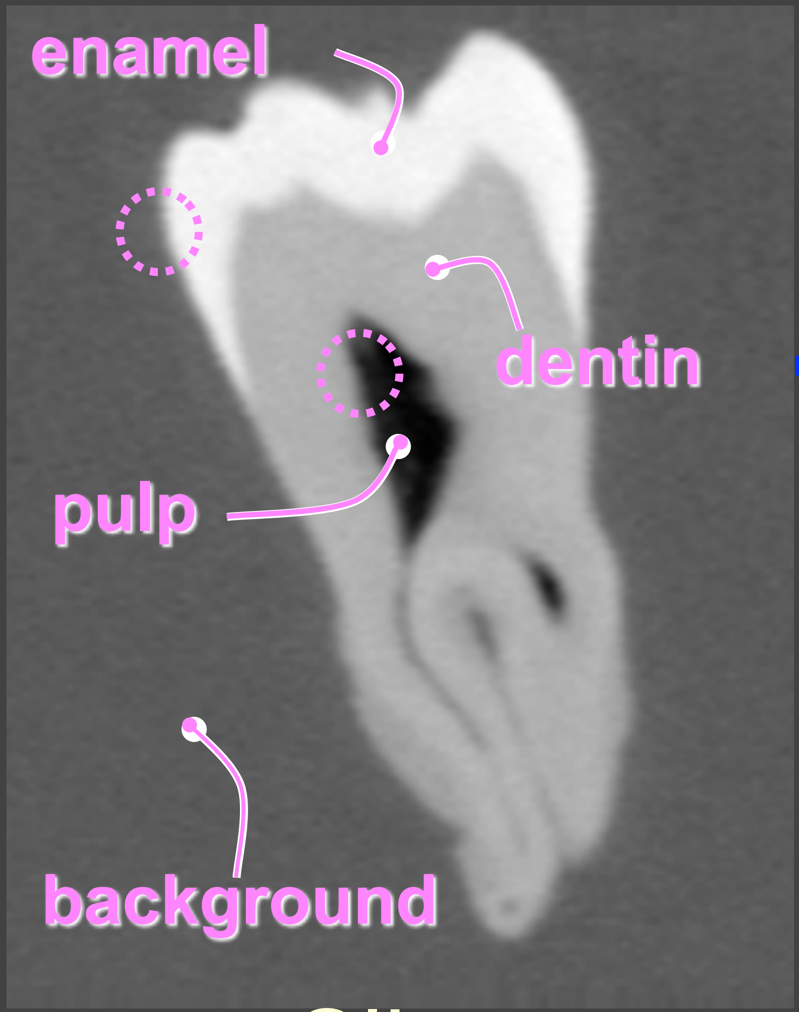
1D TF output



Rendering

1D transfer functions can not accurately capture all material boundaries

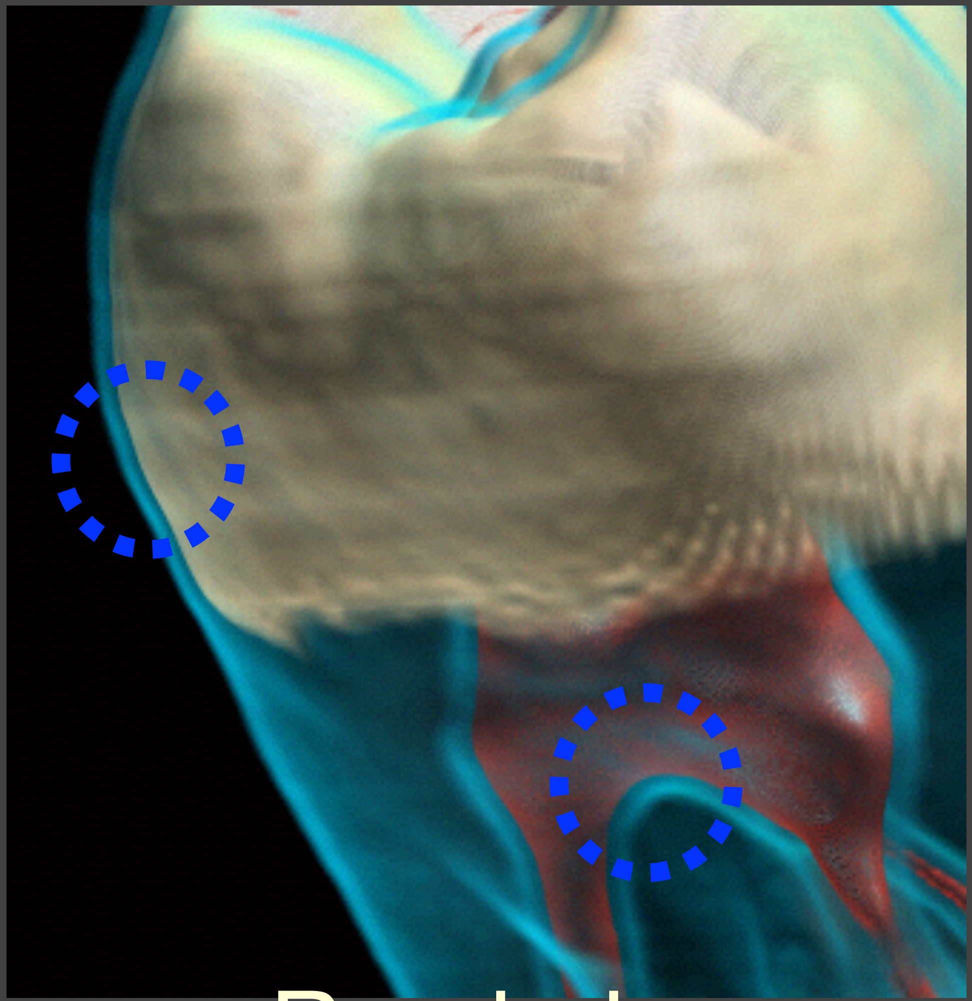
1D TFs: limitation



Slice

RGB(*f*)

1D TF output



Rendering

1D transfer functions can not accurately capture all material boundaries

1D → 2D Transfer Function

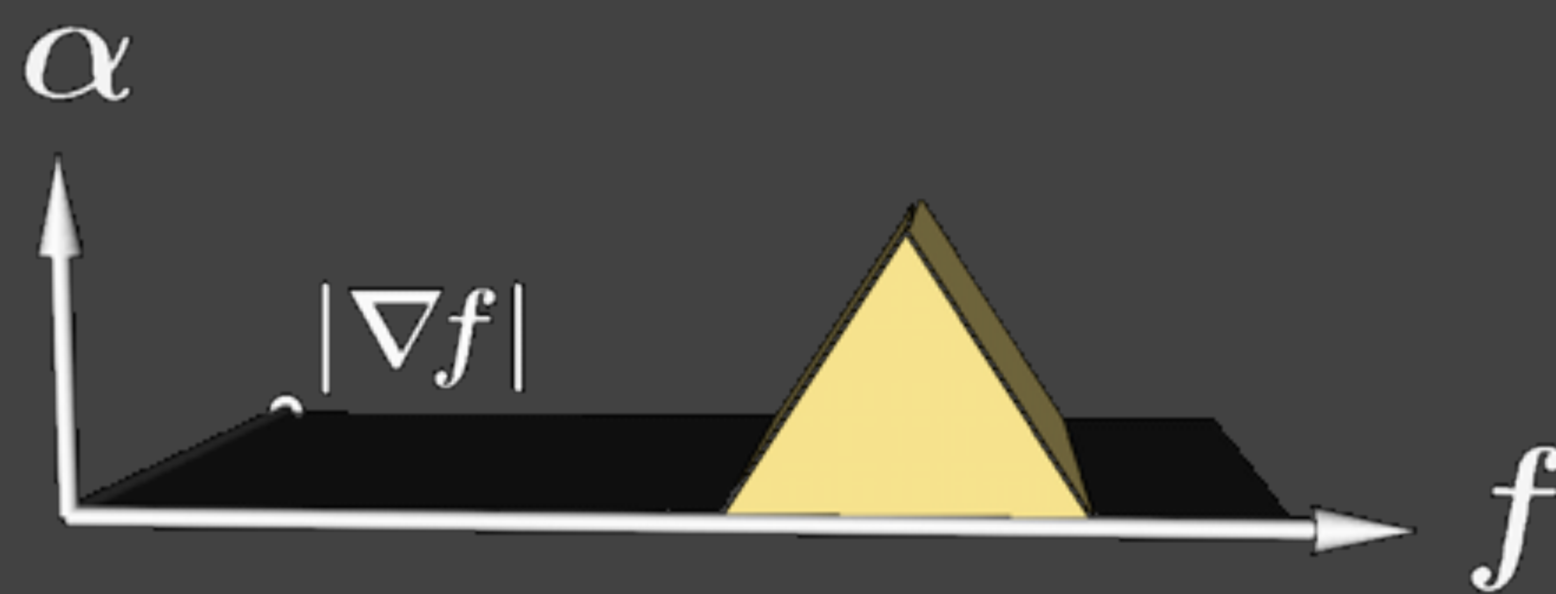
$$\left.\begin{array}{r}\mathrm{RGB}(f)\\ \alpha(f)\end{array}\right\}\text{ Generalize…}$$

1D → 2D Transfer Function

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

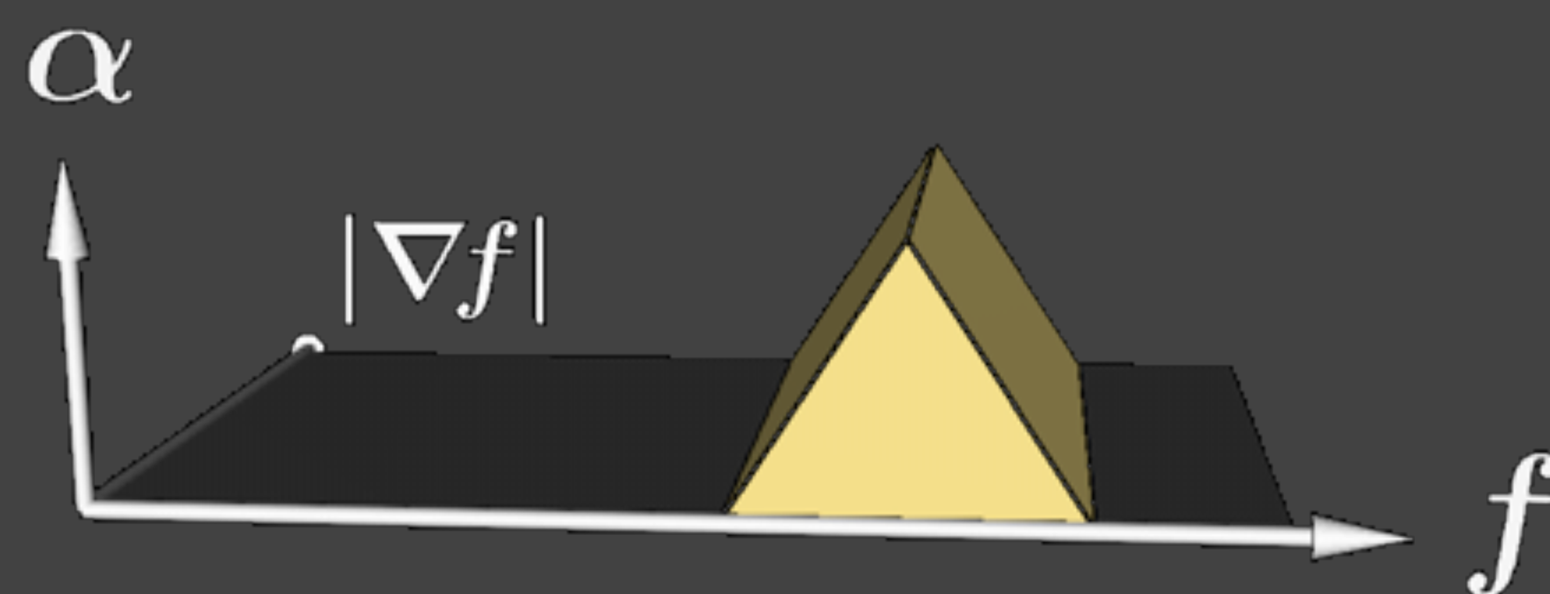
1D → 2D Transfer Function

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



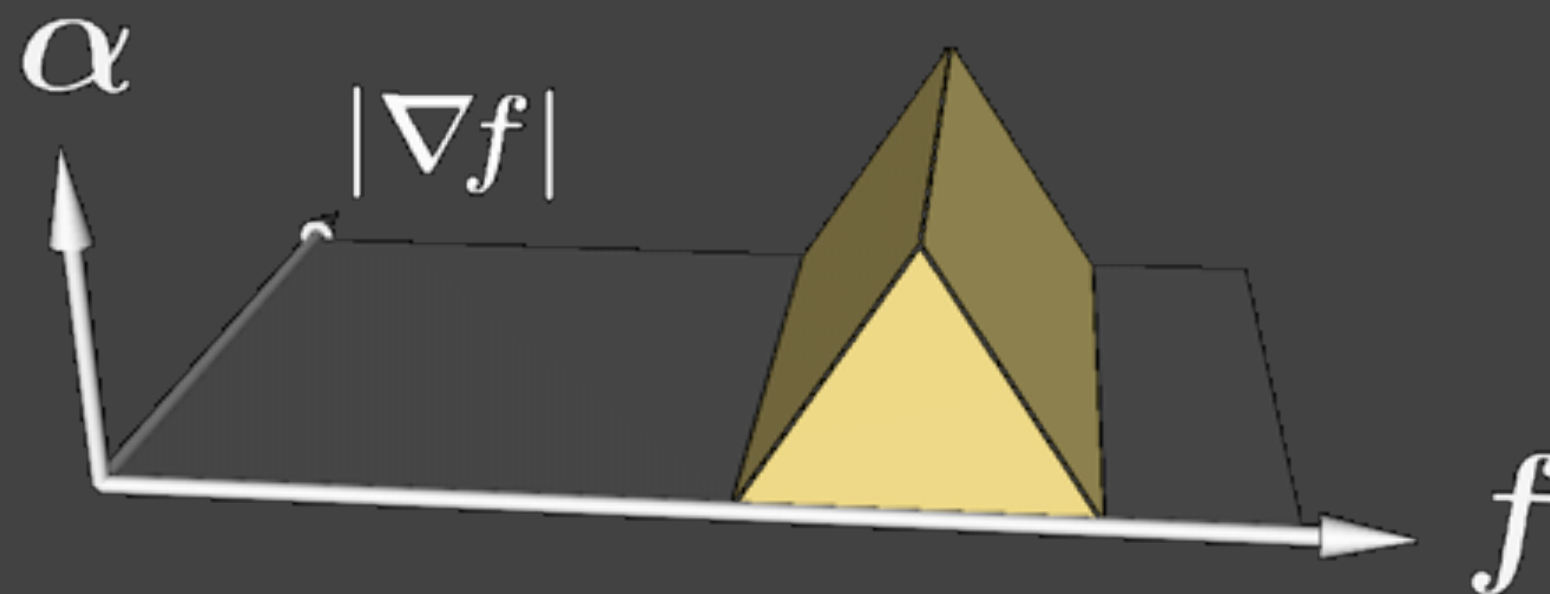
1D → 2D Transfer Function

$$\left.\begin{array}{r}\color{red}{R}\color{green}{G}\color{blue}{B}(f) \\ \alpha(f)\end{array}\right\}$$ Generalize…



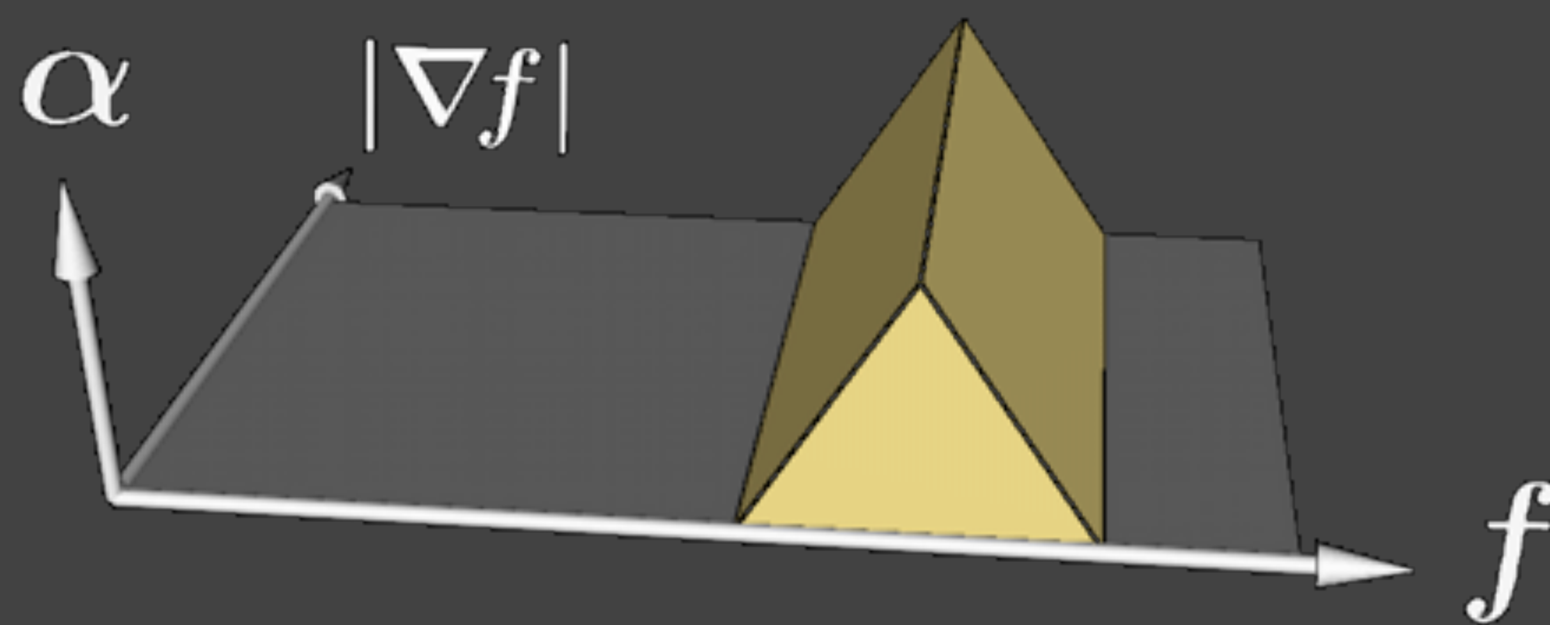
1D → 2D Transfer Function

$\mathrm{RGB}(f)$
$\alpha(f)$ } Generalize…



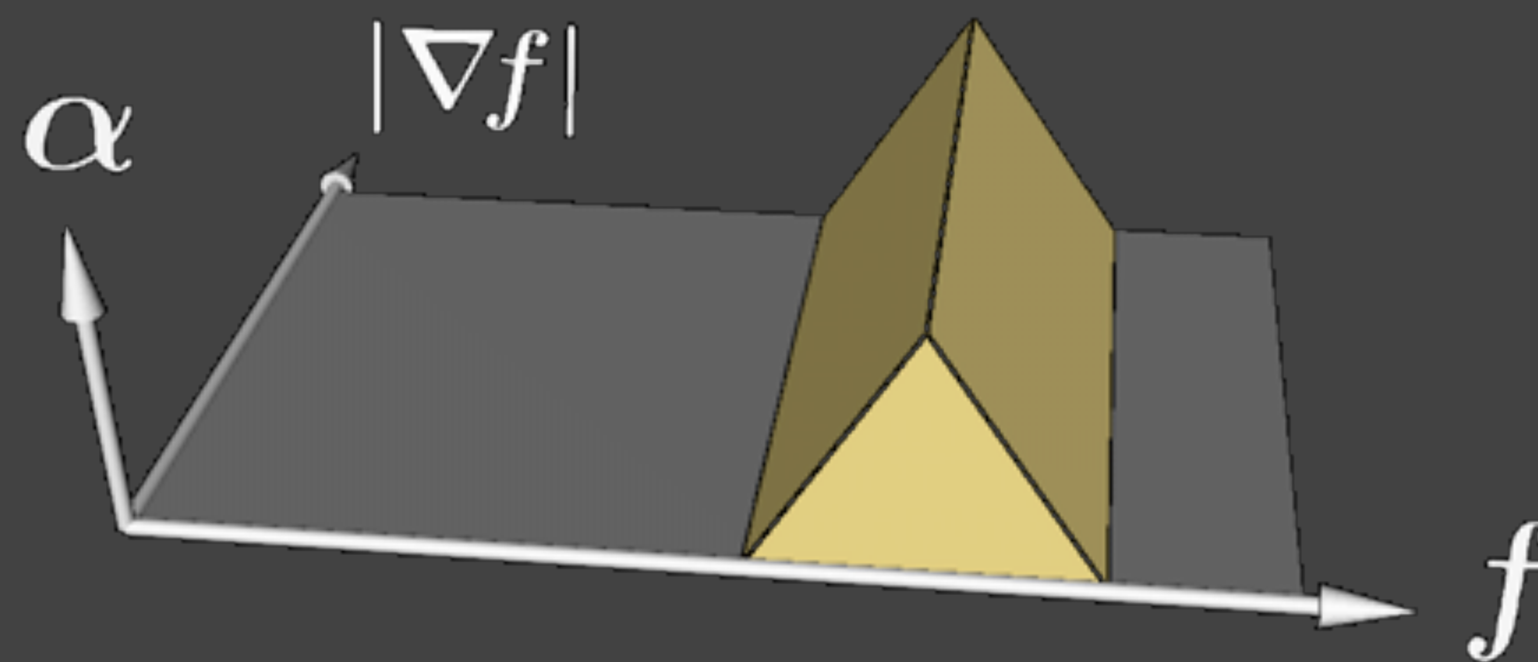
1D → 2D Transfer Function

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



2D Transfer Function

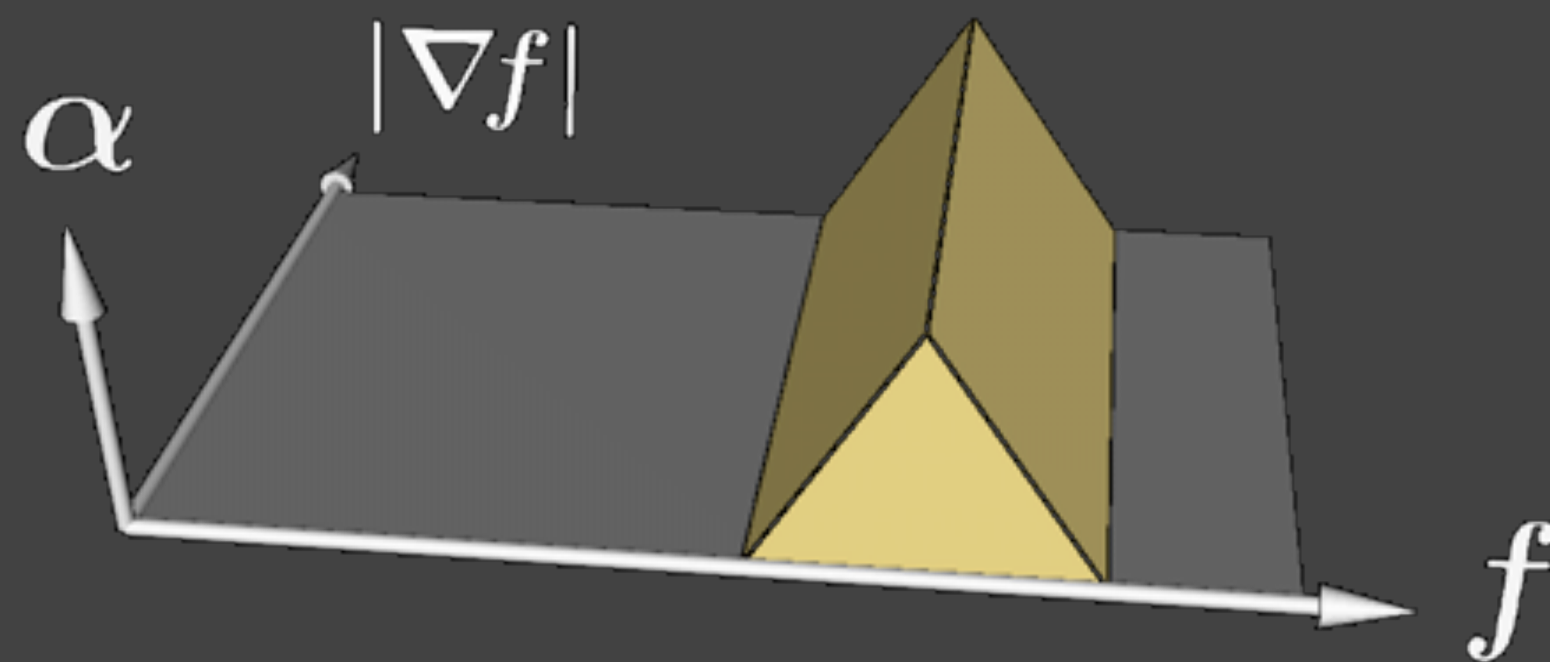
$$\left.\begin{array}{r}\mathrm{RGB}(f)\\ \alpha(f)\end{array}\right\}\ \text{Generalize...}$$



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

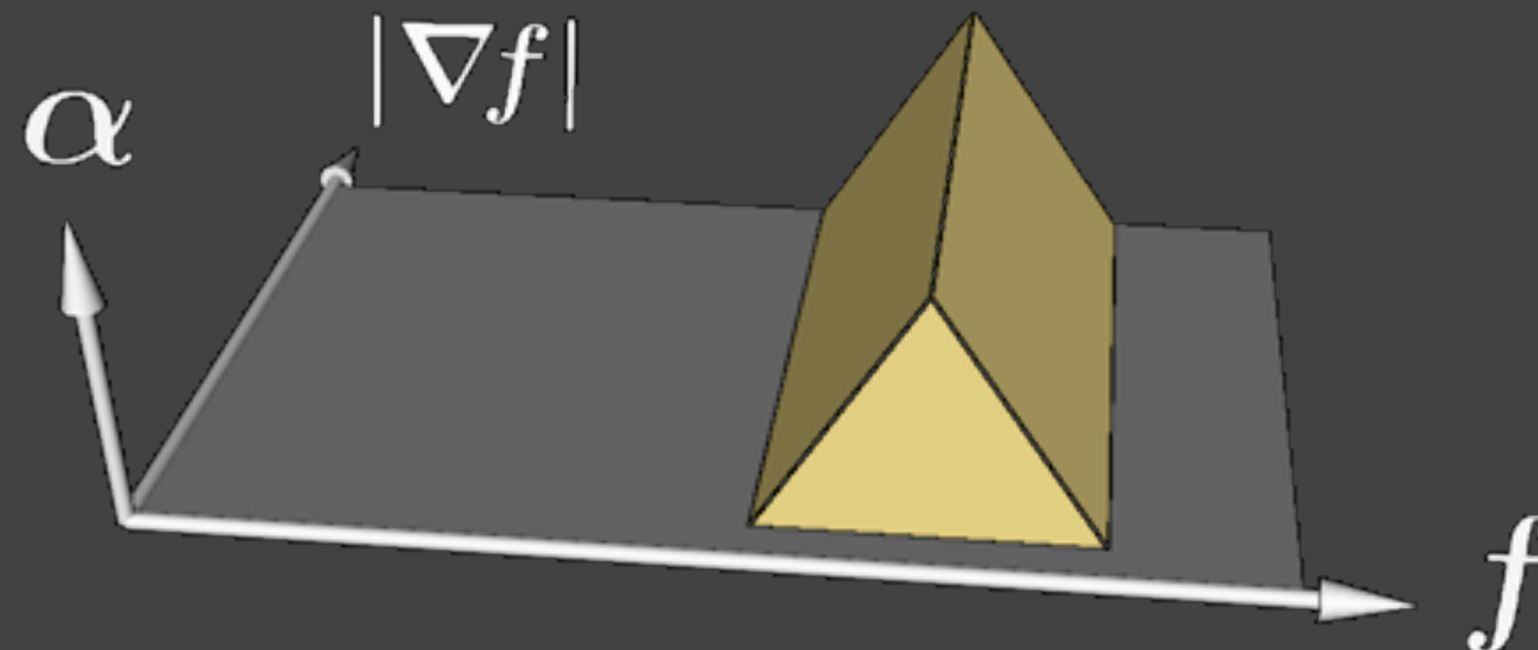
2D Transfer Function

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



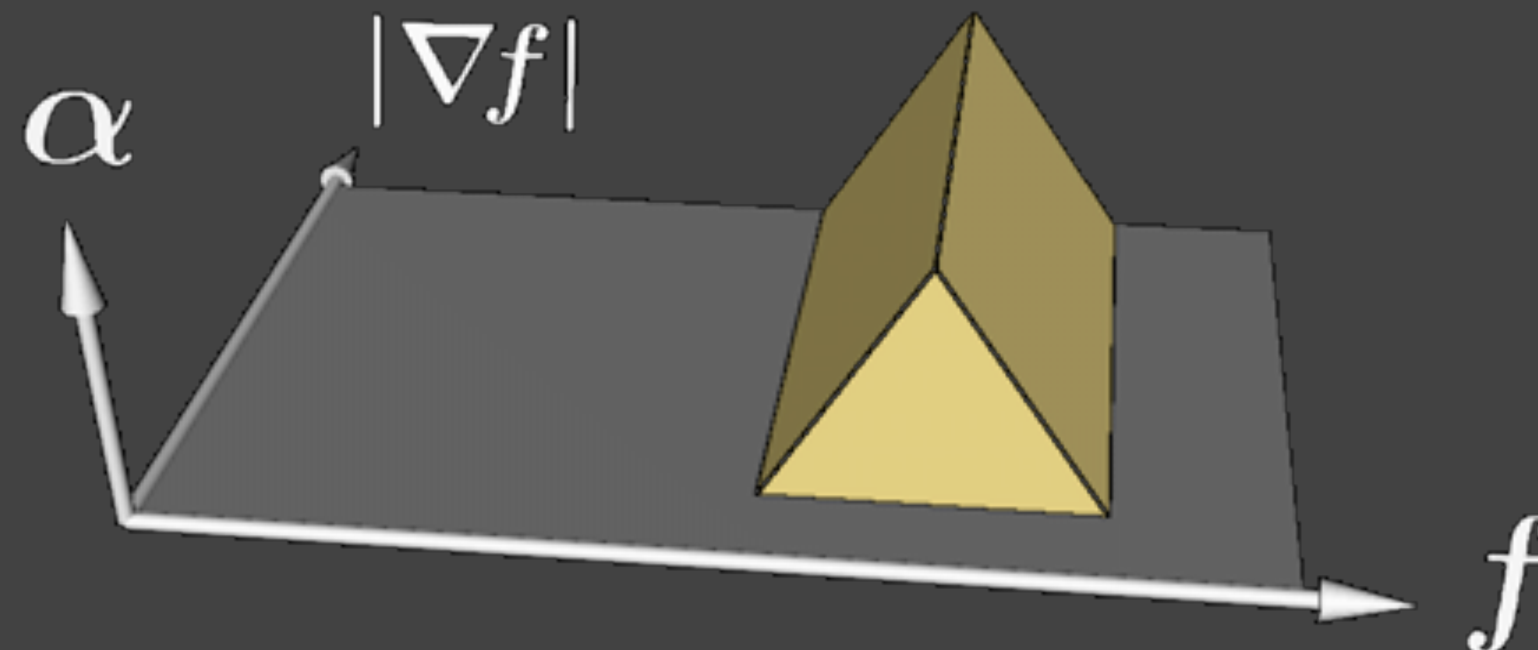
2D Transfer Function

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



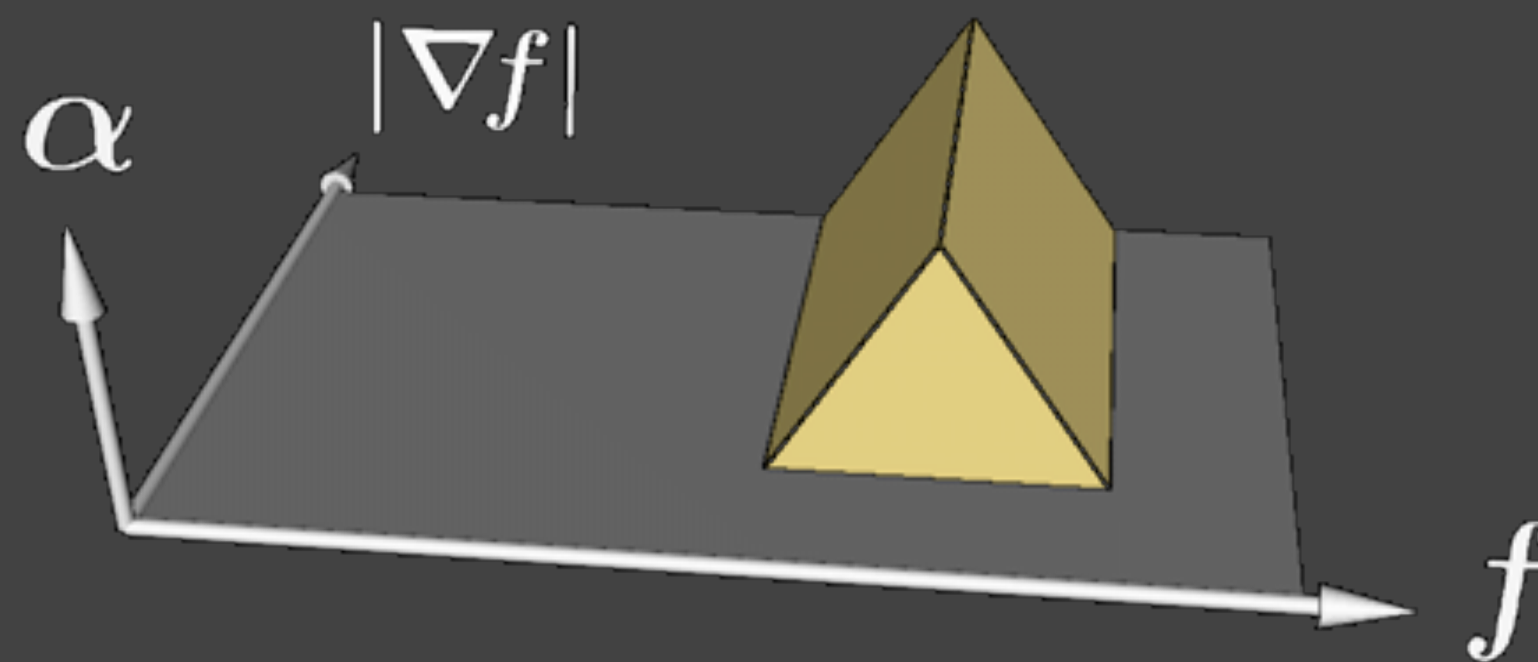
2D Transfer Function

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



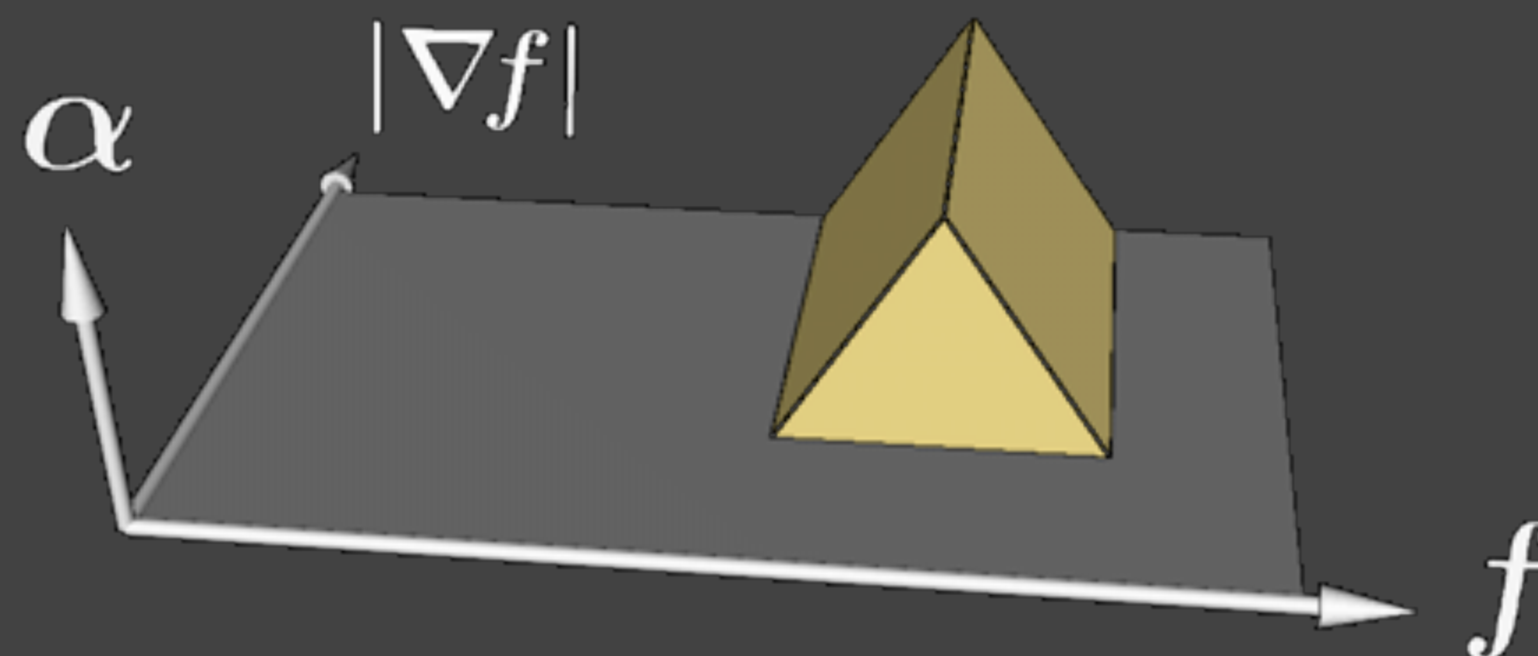
2D Transfer Function

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



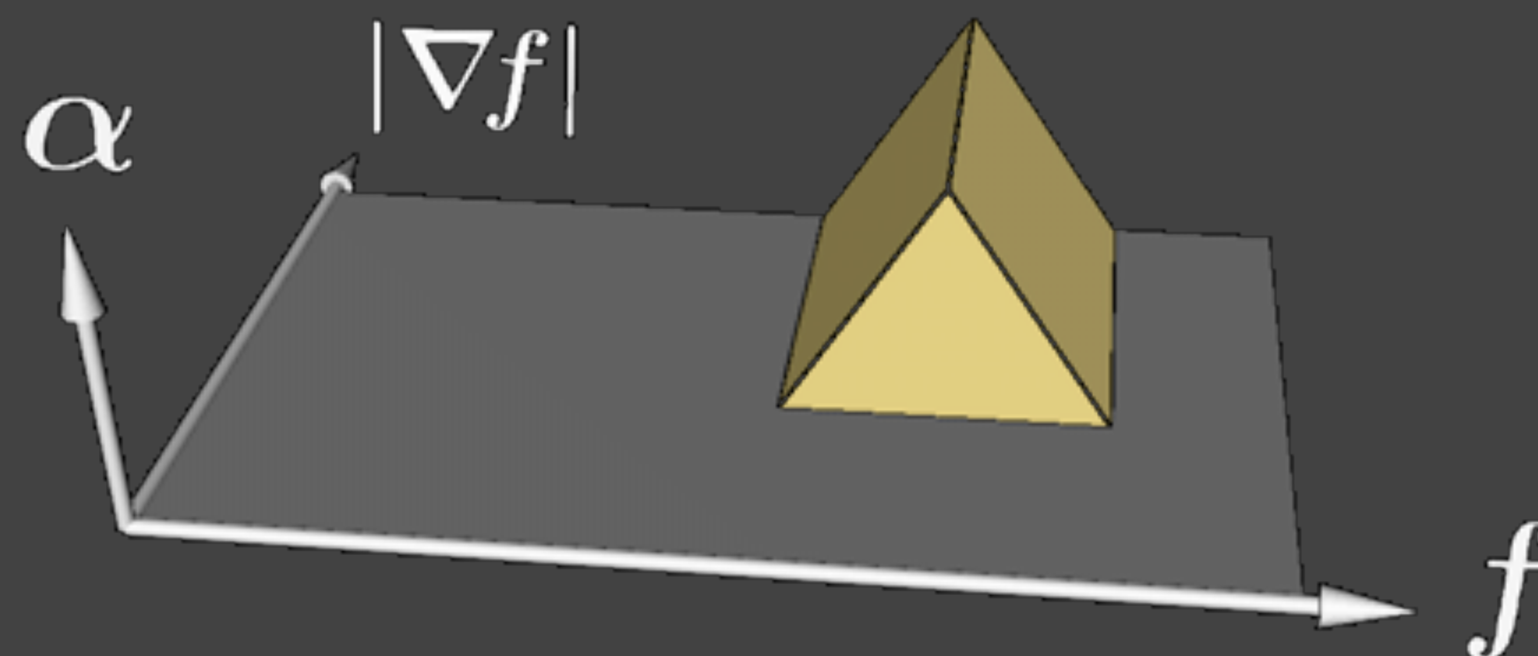
2D Transfer Function

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



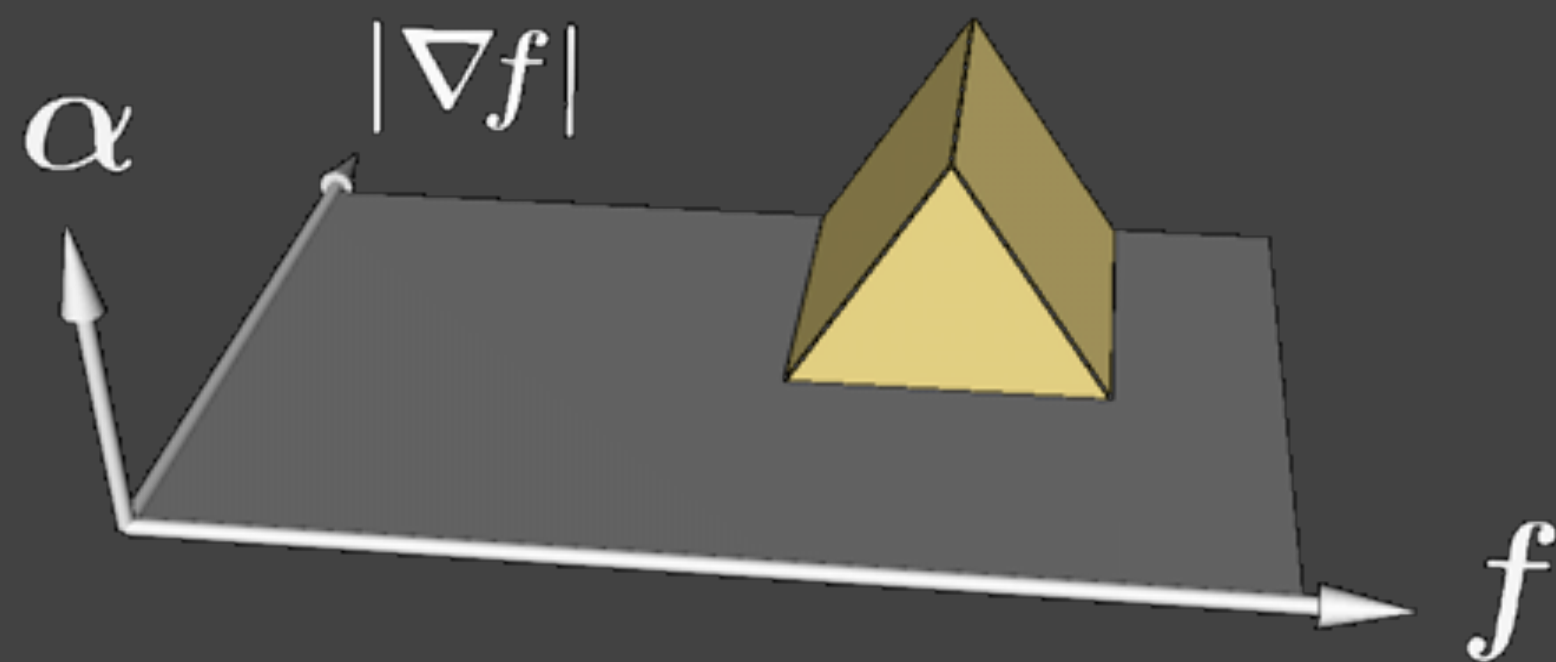
2D Transfer Function

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

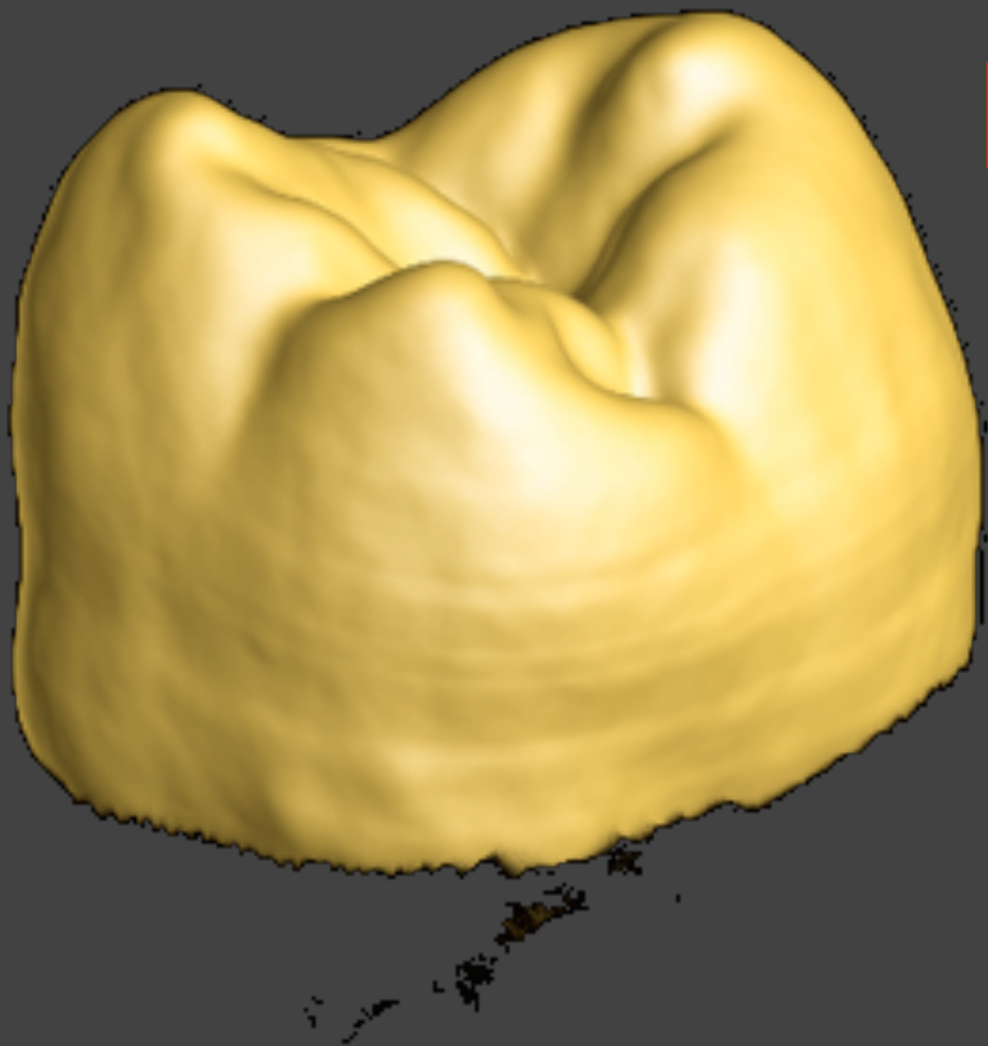


2D Transfer Function

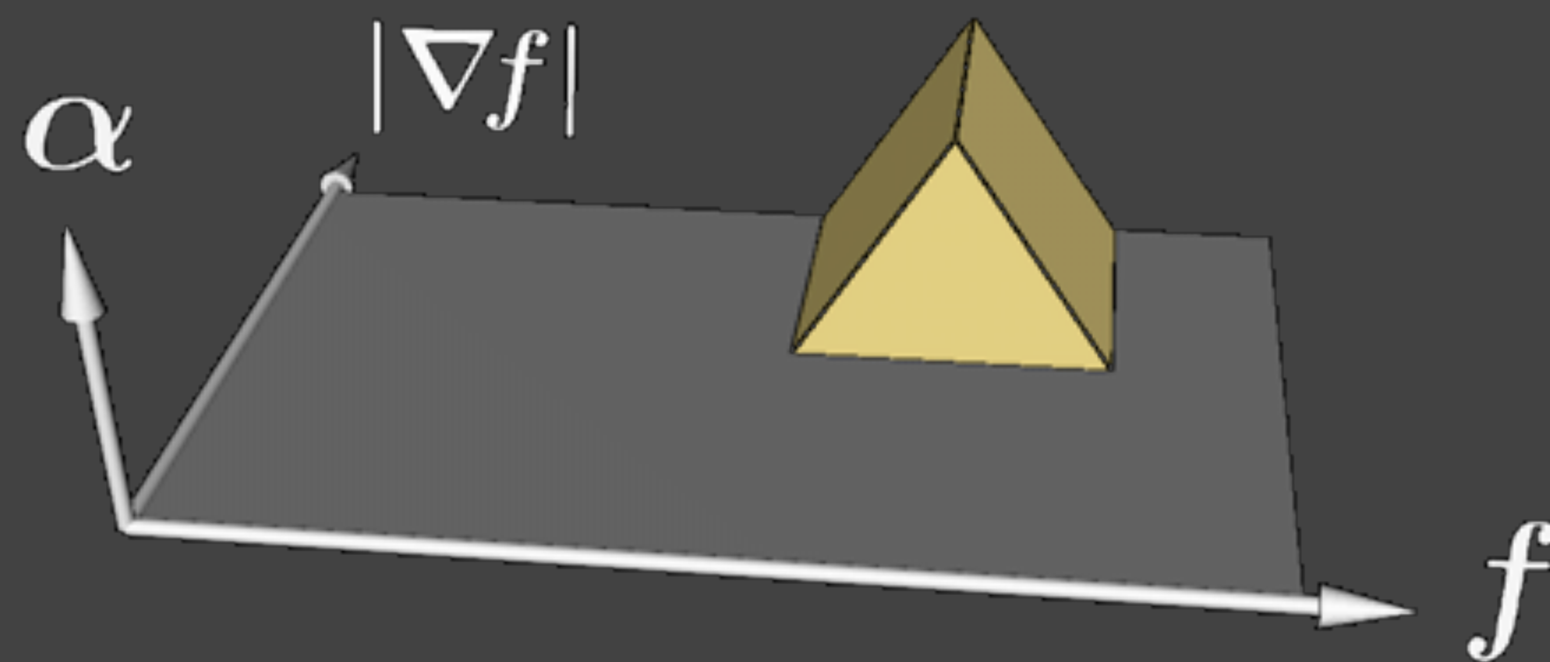
$$\left.\begin{array}{r}\mathrm{RGB}(f,|\nabla f|)\\ \alpha(f,|\nabla f|)\end{array}\right\}\text{Modify...}$$



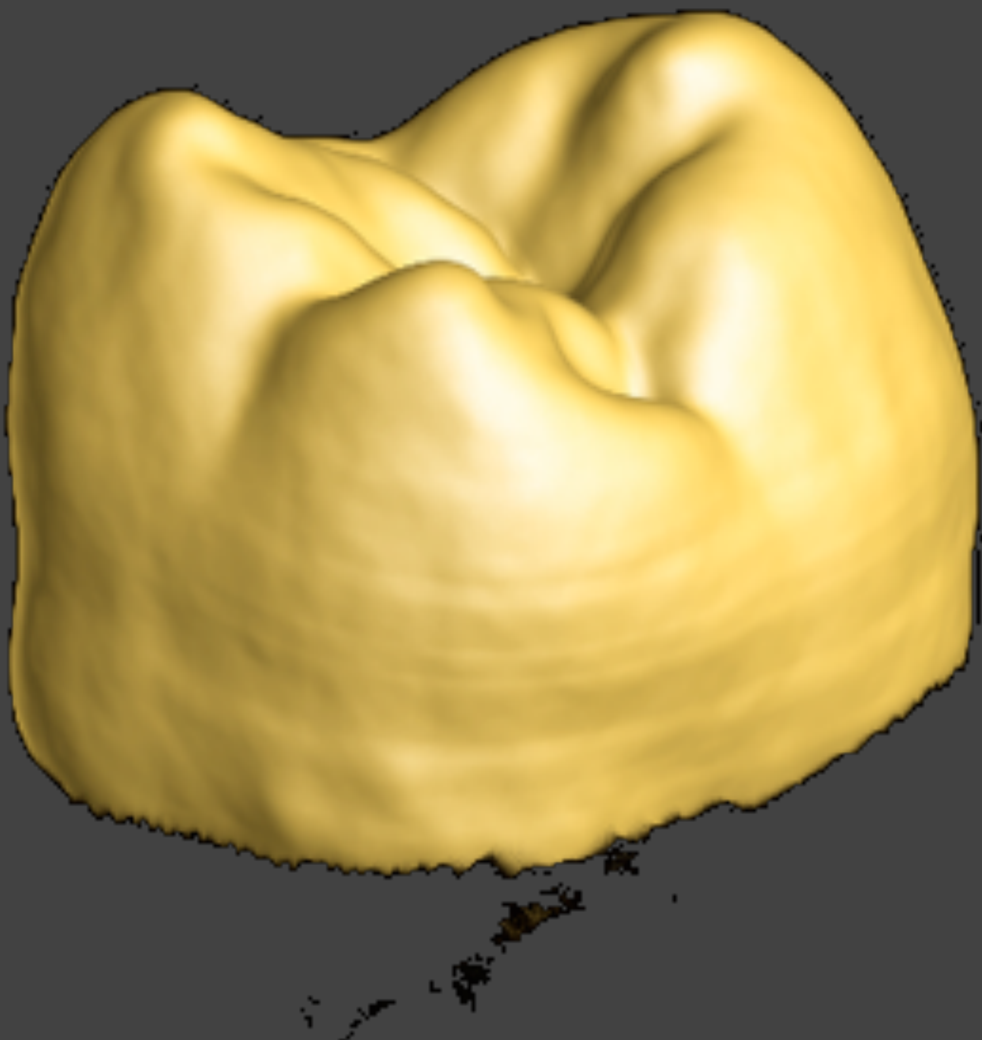
2D Transfer Function



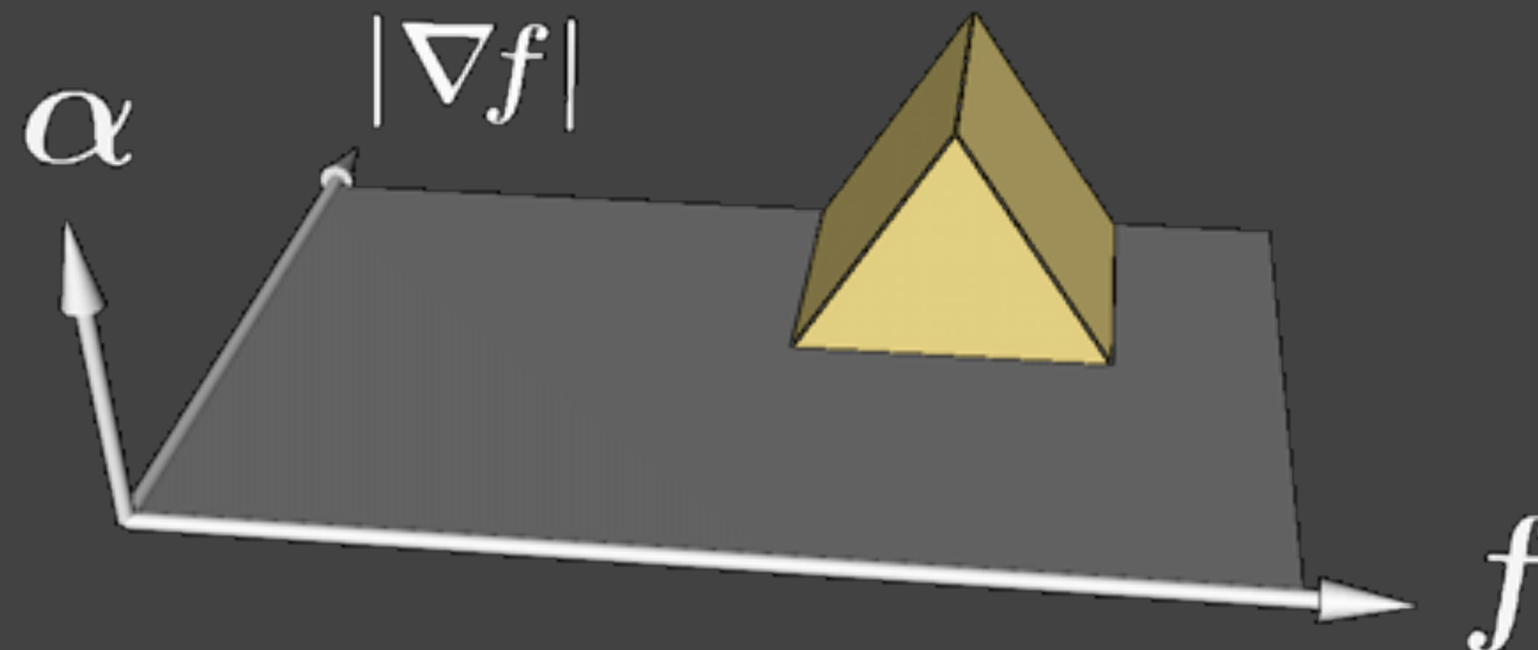
$$\left.\begin{array}{r}\mathrm{RGB}(f,|\nabla f|)\\ \alpha(f,|\nabla f|)\end{array}\right\}\text{Modify…}$$



2D Transfer Function



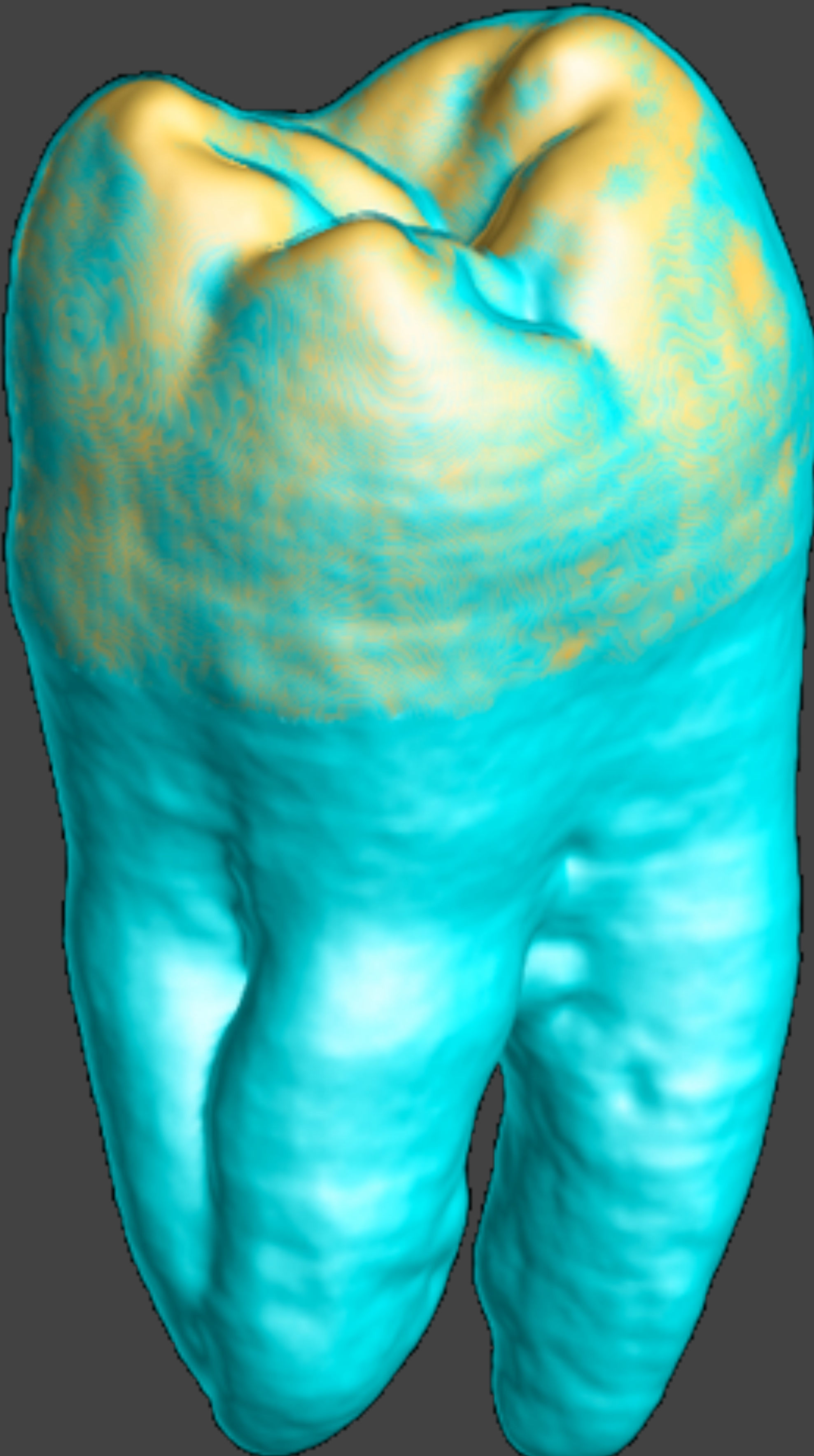
$$\left.\begin{array}{r}\mathrm{RGB}(f, |\nabla f|)\\ \alpha(f, |\nabla f|)\end{array}\right\}\text{Modify…}$$



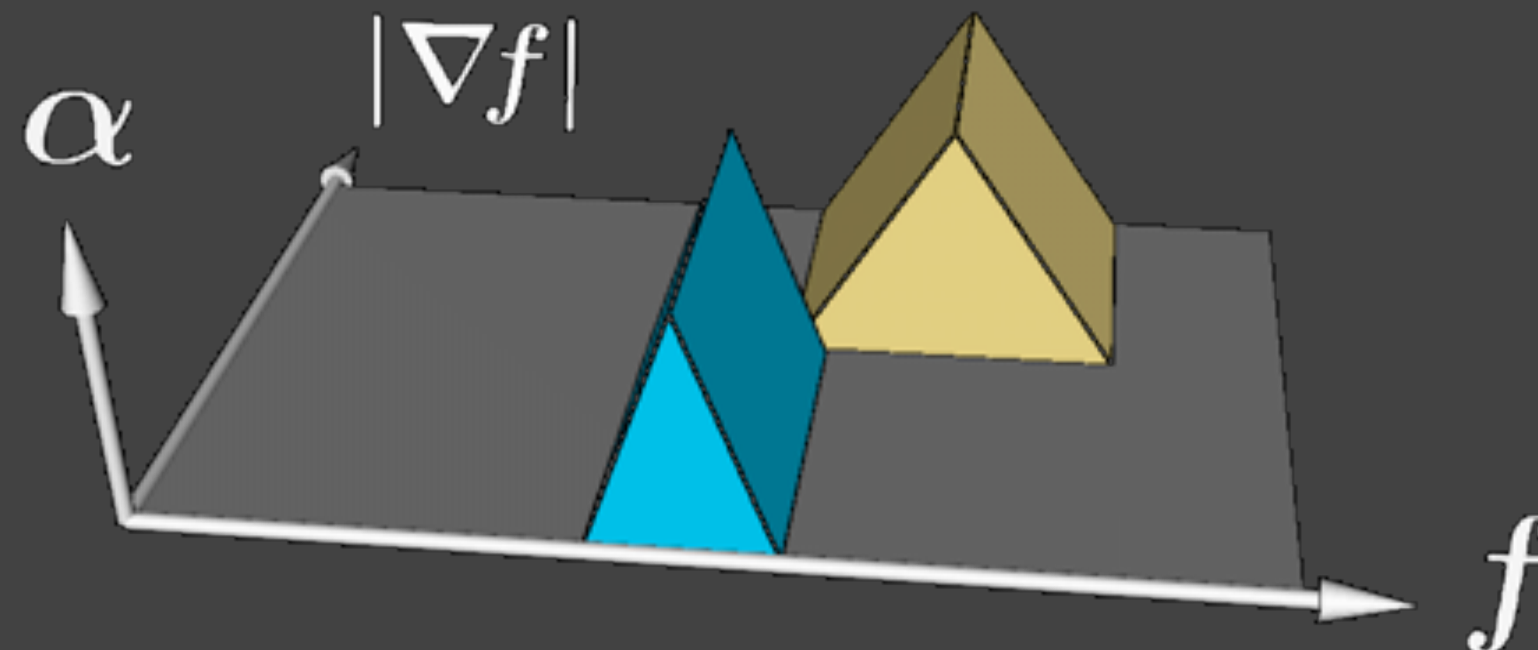
2D transfer functions give greater flexibility in boundary visualization

Display of Surfaces from Volume Data, Levoy 1988

2D Transfer Function

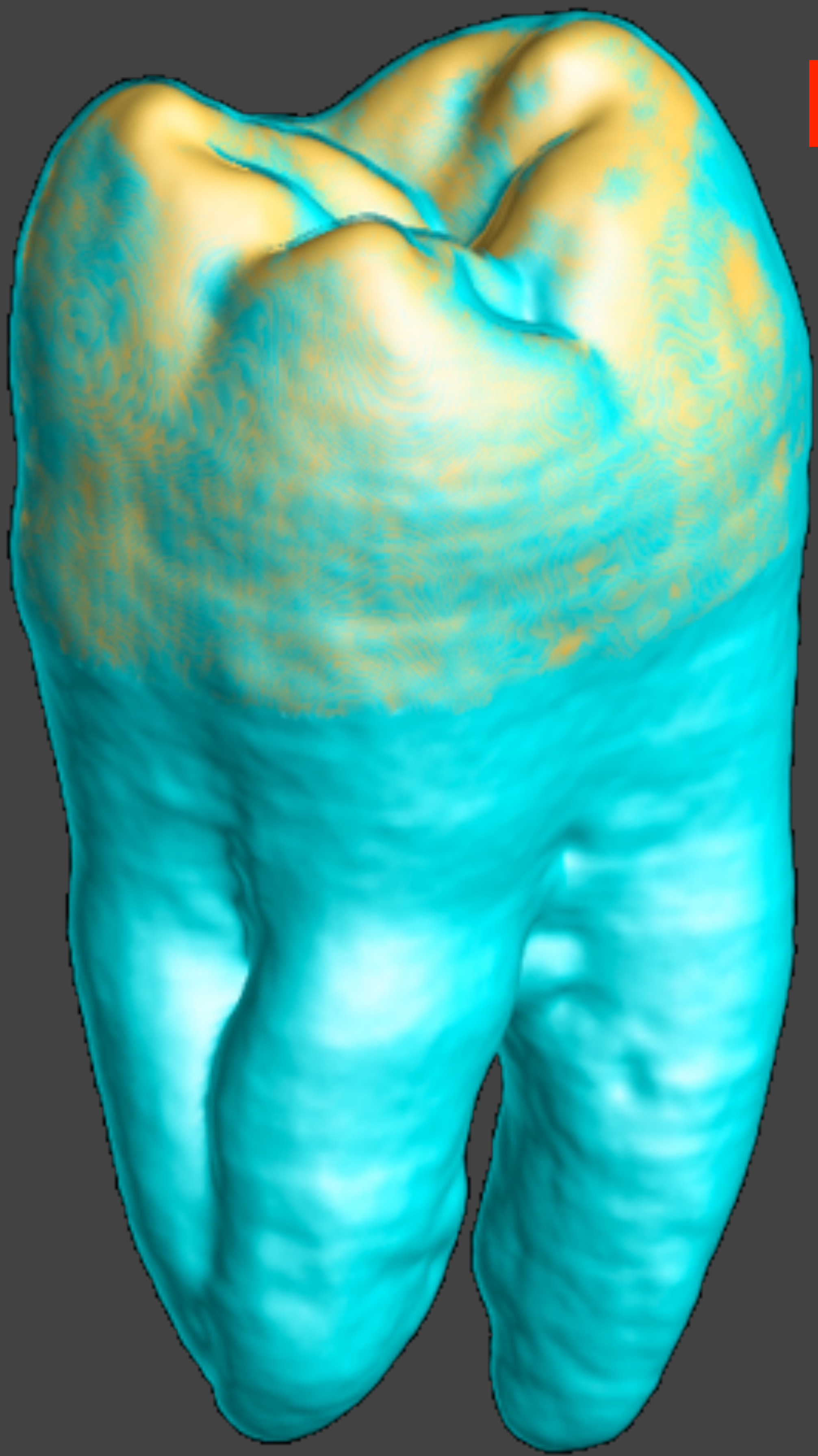


$$\left.\begin{array}{r}\mathrm{RGB}(f,|\nabla f|)\\ \alpha(f,|\nabla f|)\end{array}\right\}\text{Modify...}$$

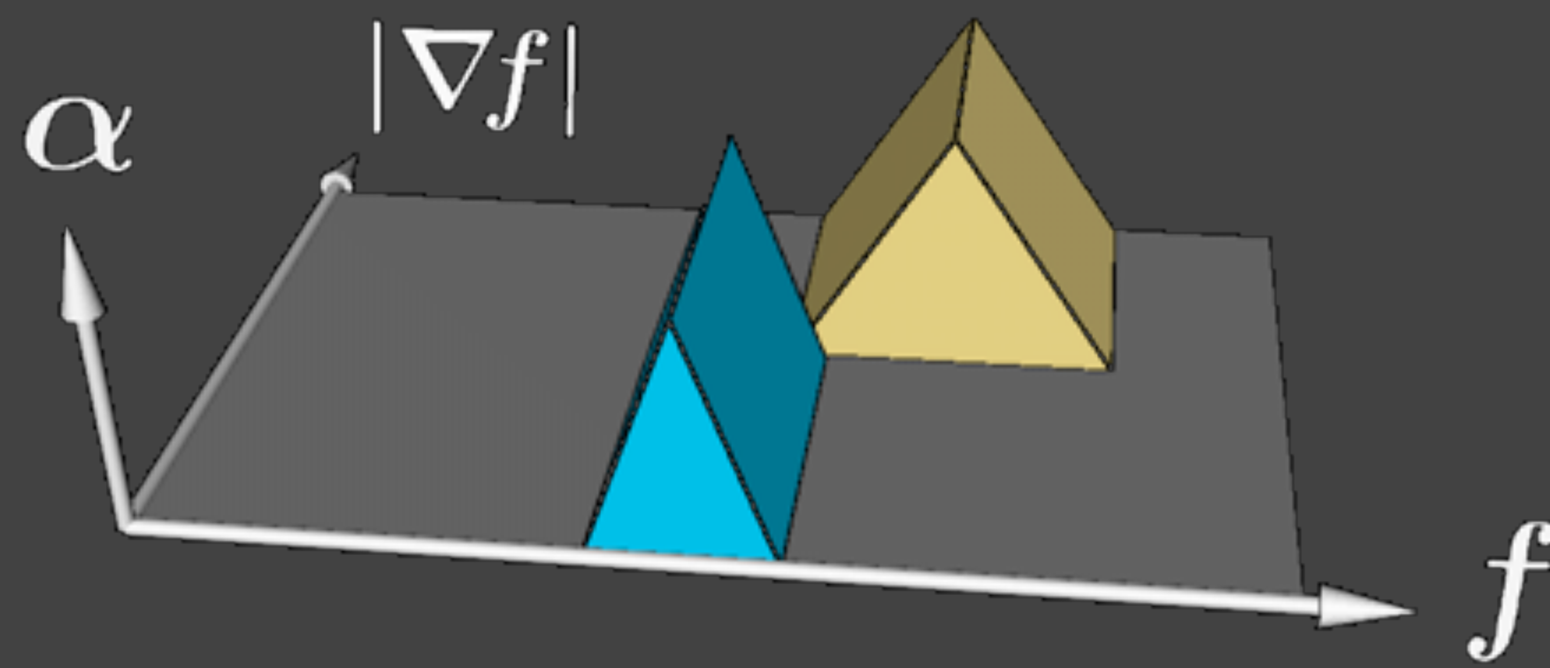


Trying to reintroduce dentin / background boundary …

2D Transfer Function

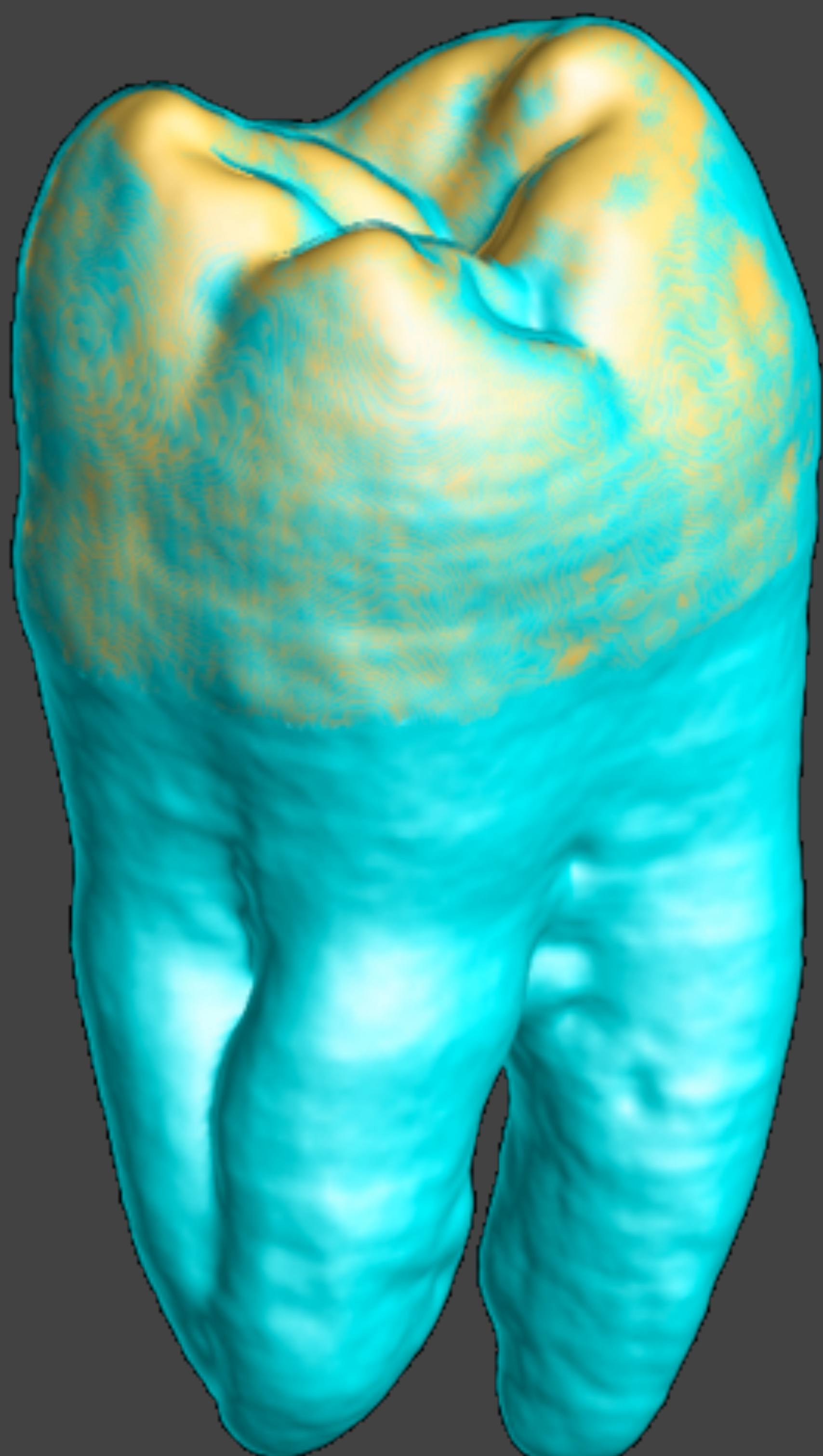


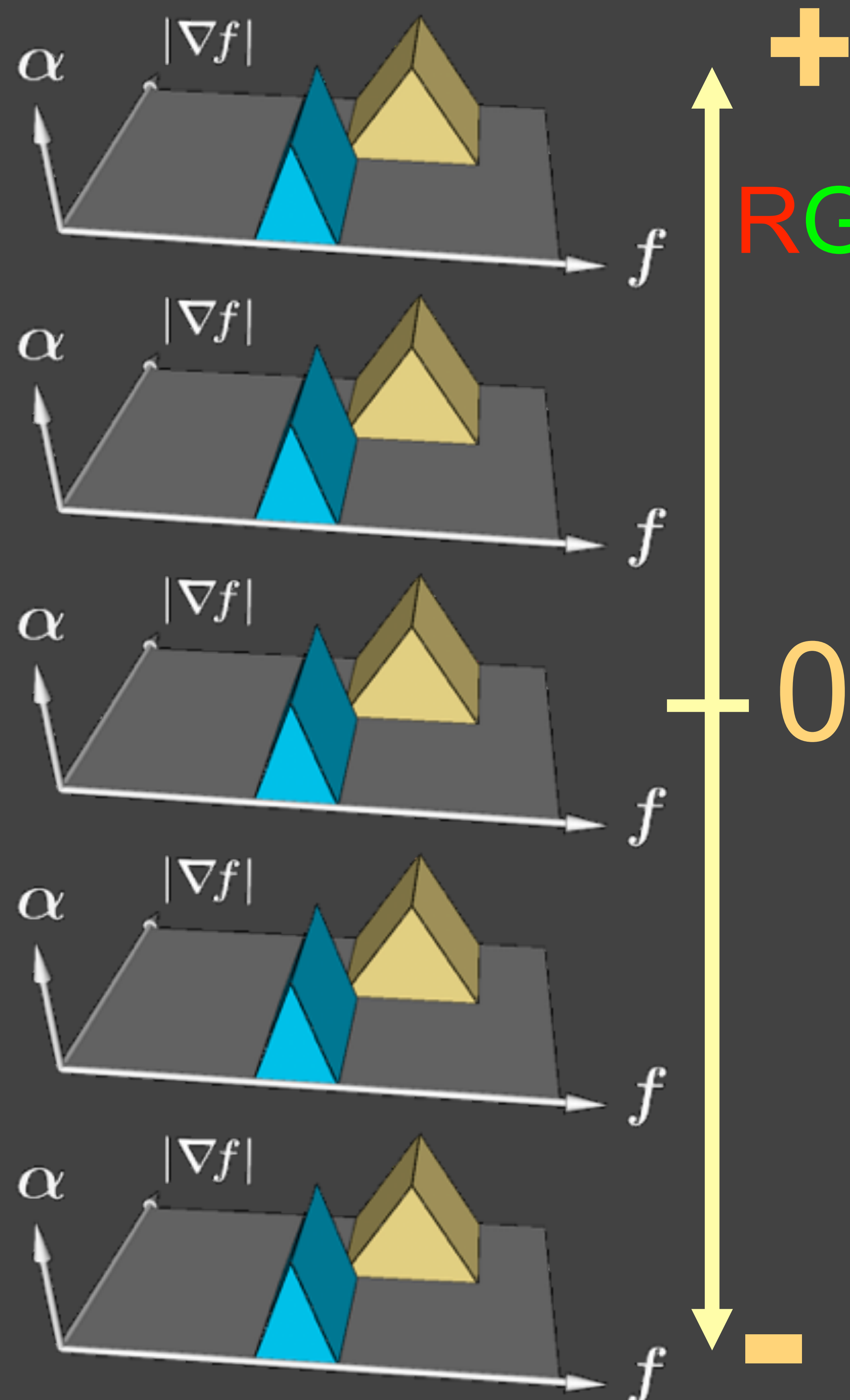
$$\left.\begin{array}{r} RGB(f, |\nabla f|) \\ \alpha(f, |\nabla f|) \end{array}\right\}$$ Modify…



Trying to reintroduce dentin / background boundary …

2D → 3D Transfer Function





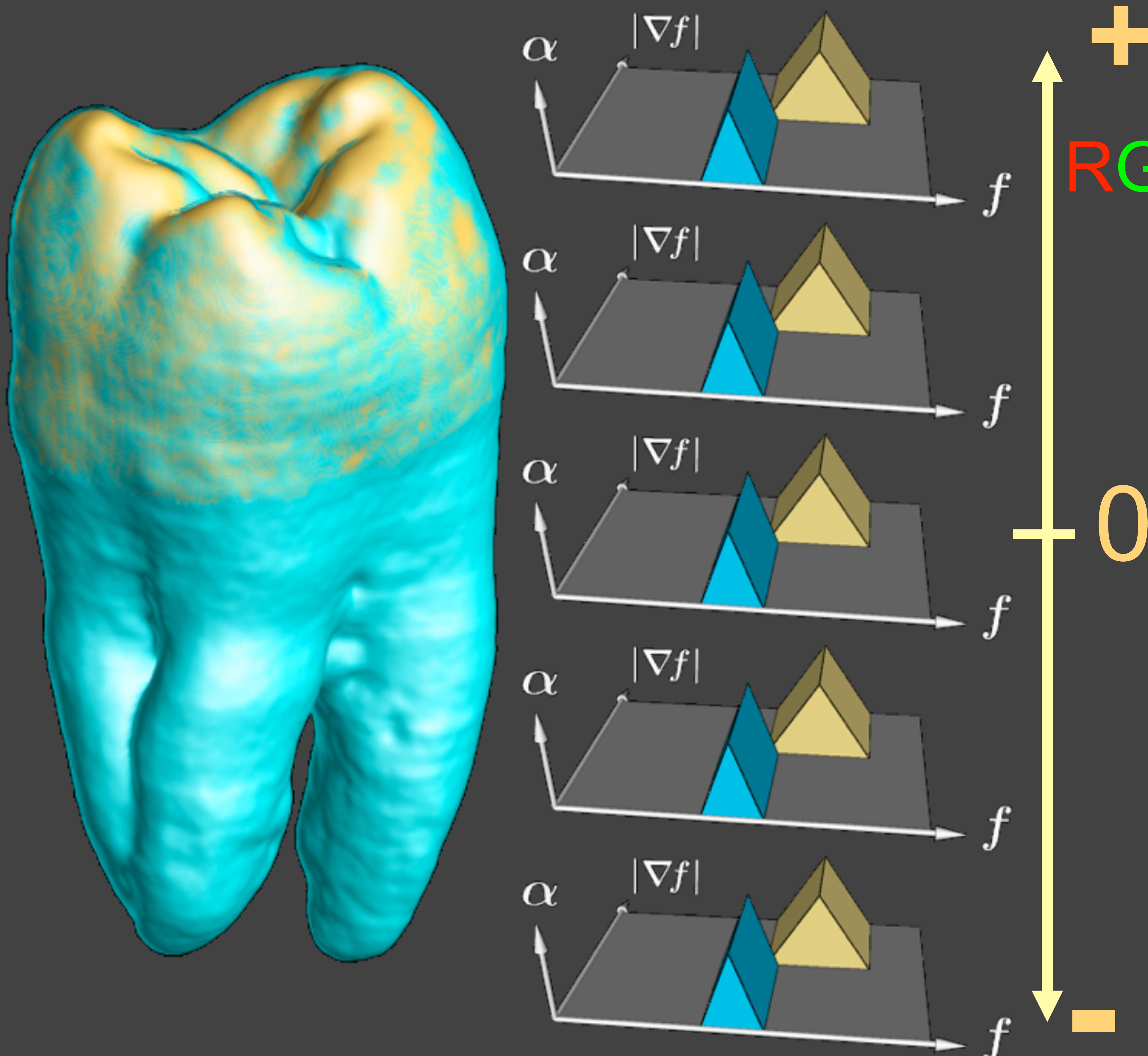
RGB $\alpha(f, |\nabla f|, \mathbf{D}^2_{\widehat{\nabla f}} f)$

Second directional derivative

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

measured with Hessian

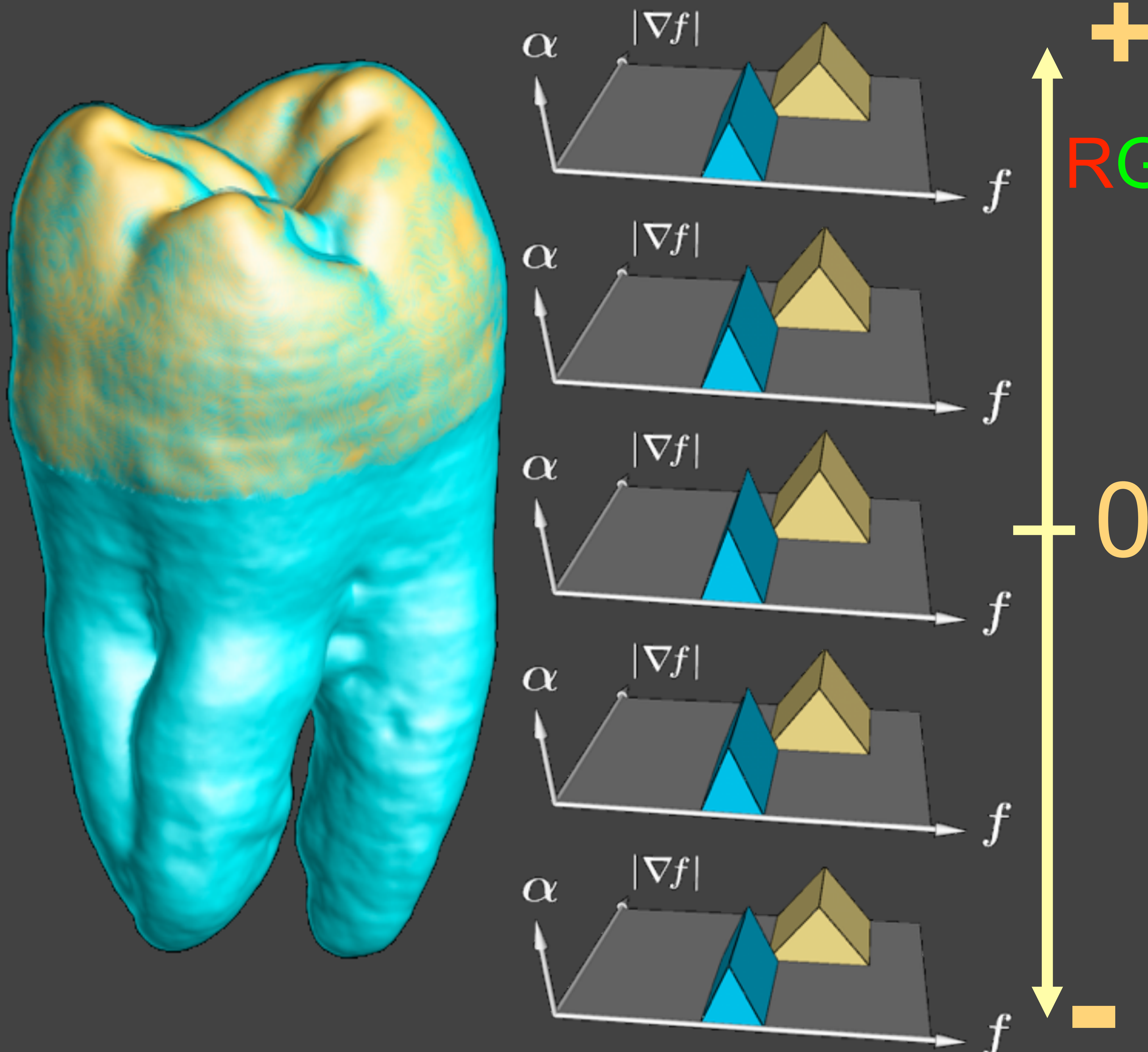
3D Transfer Function



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

Modify…

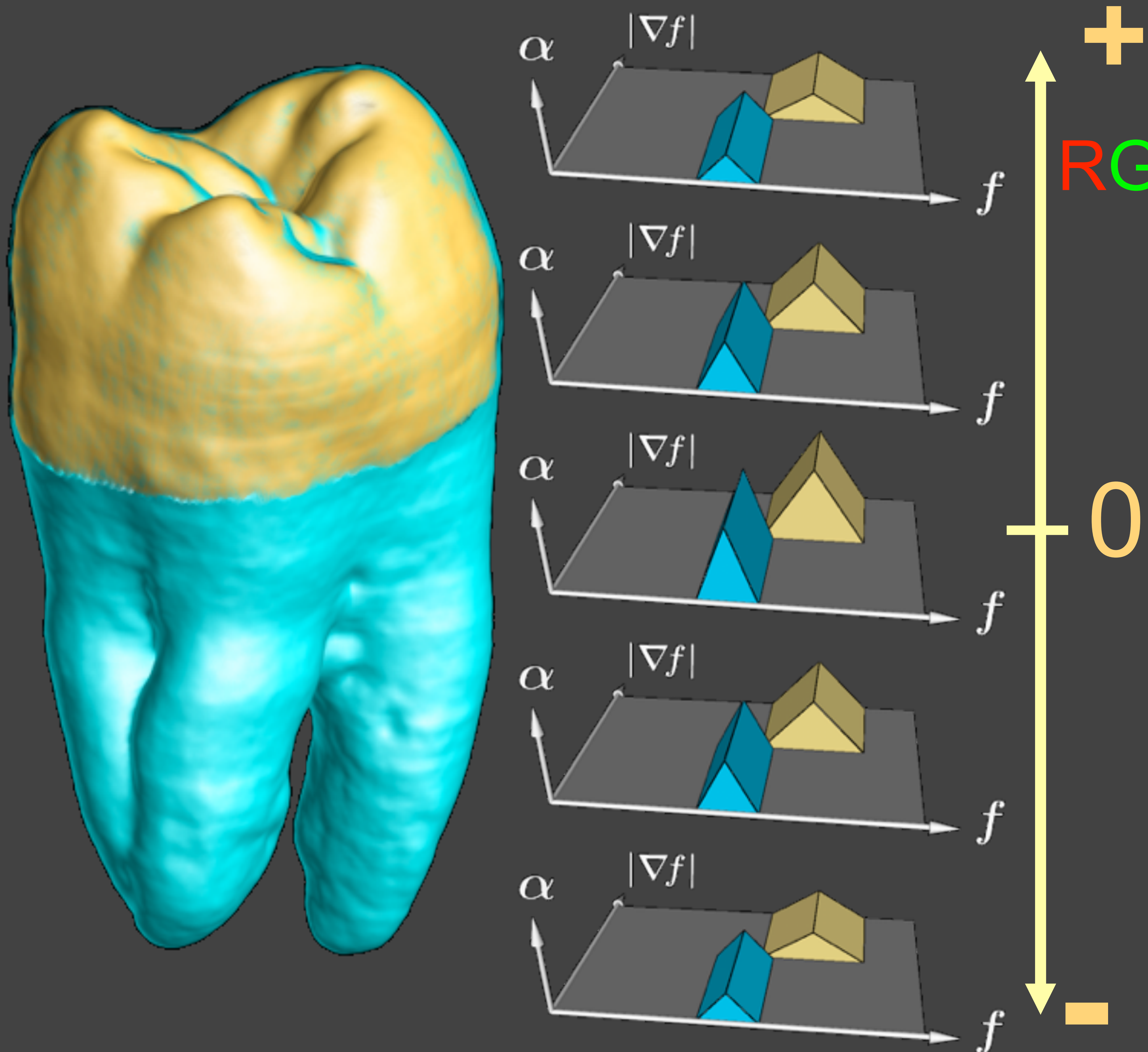
3D Transfer Function



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

Modify…

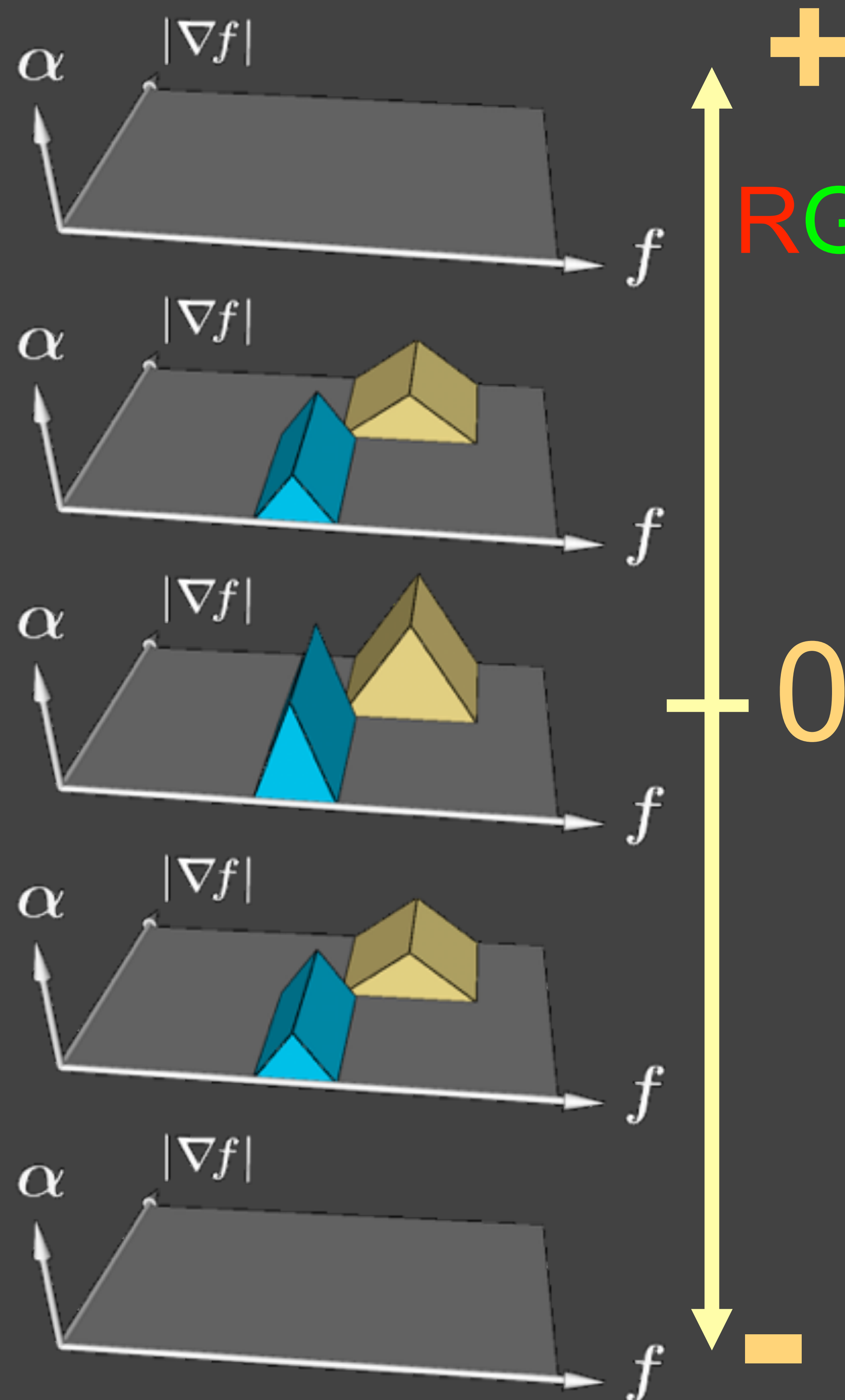
3D Transfer Function



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

Modify…

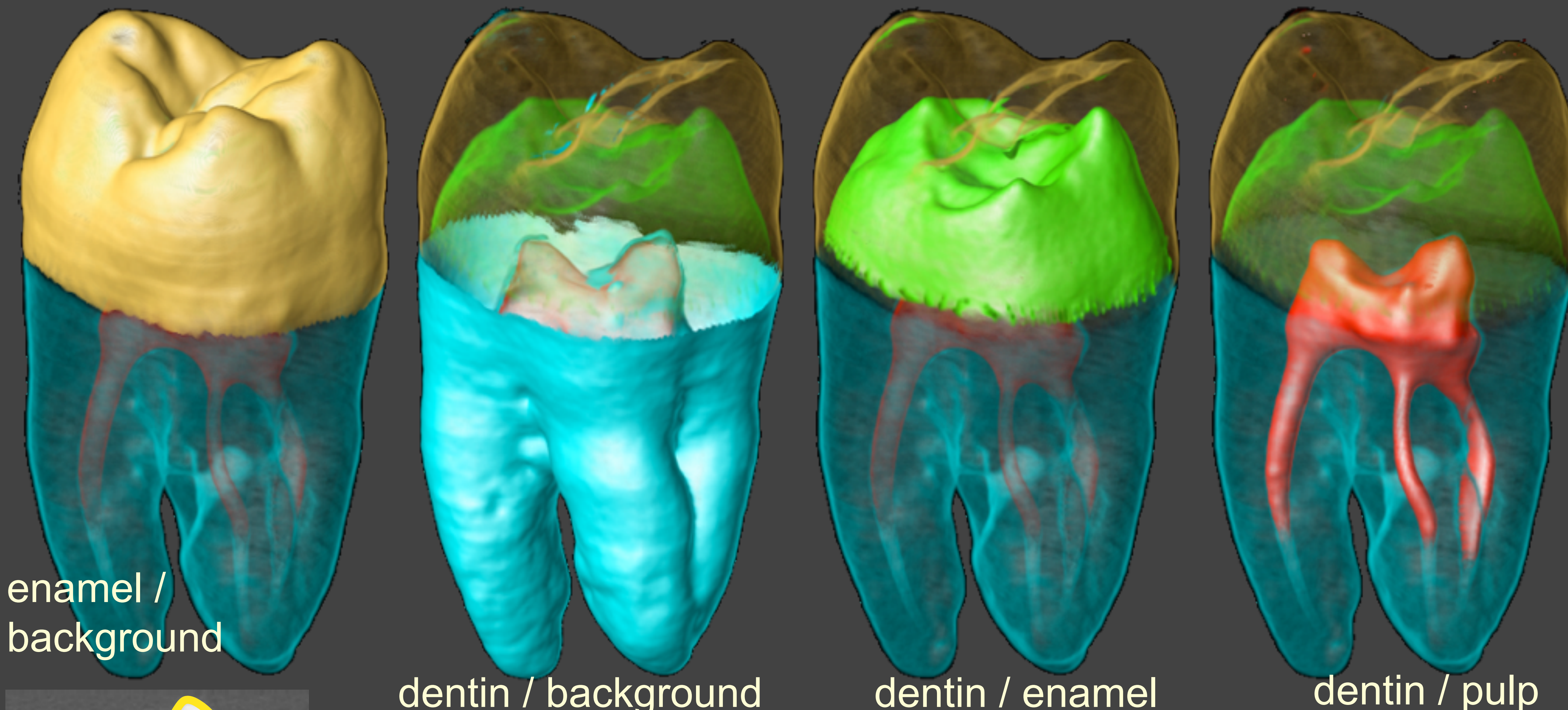
3D Transfer Function



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

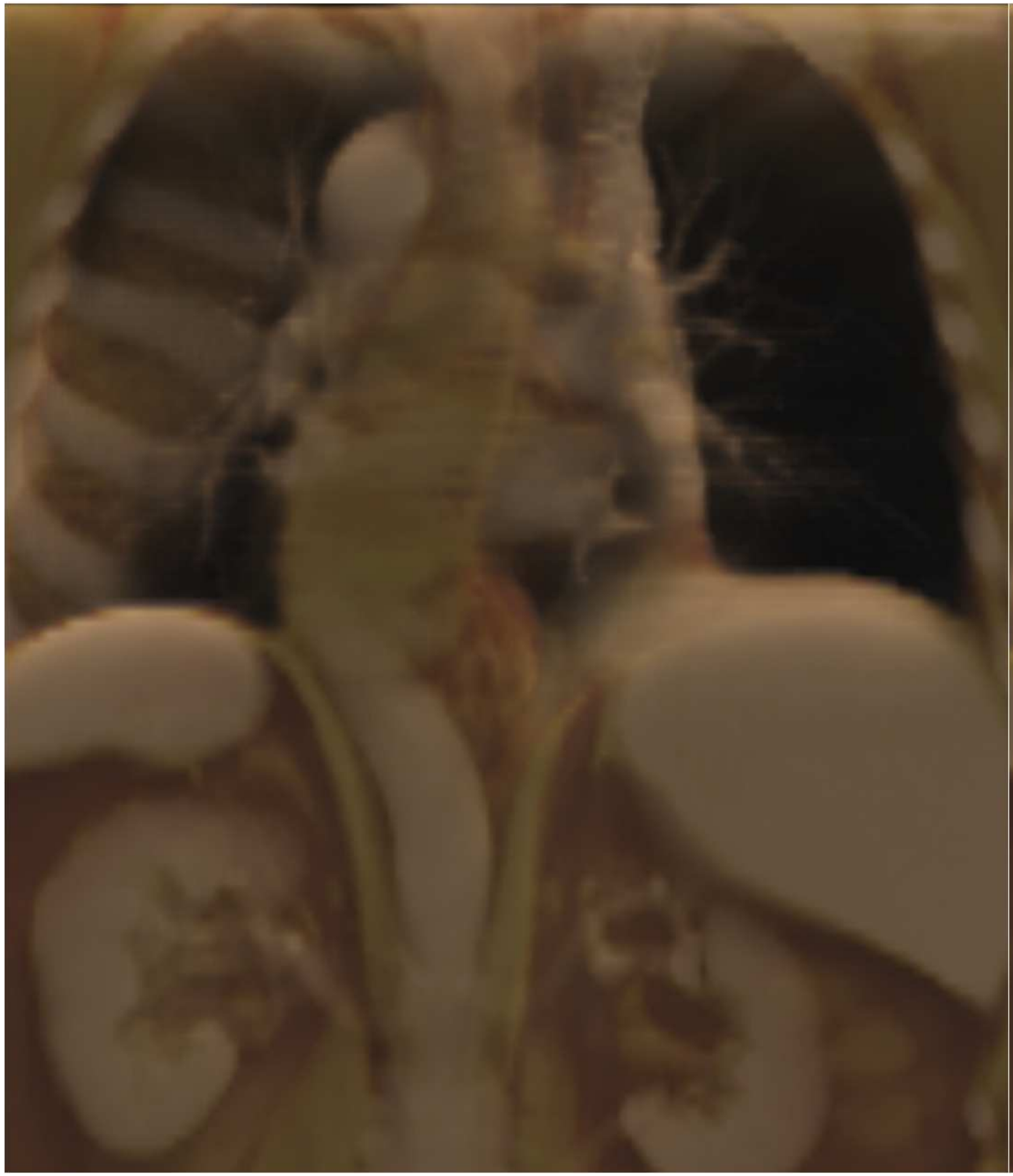
Done

3D Transfer Function



1D: not possible

2D: specificity not as good

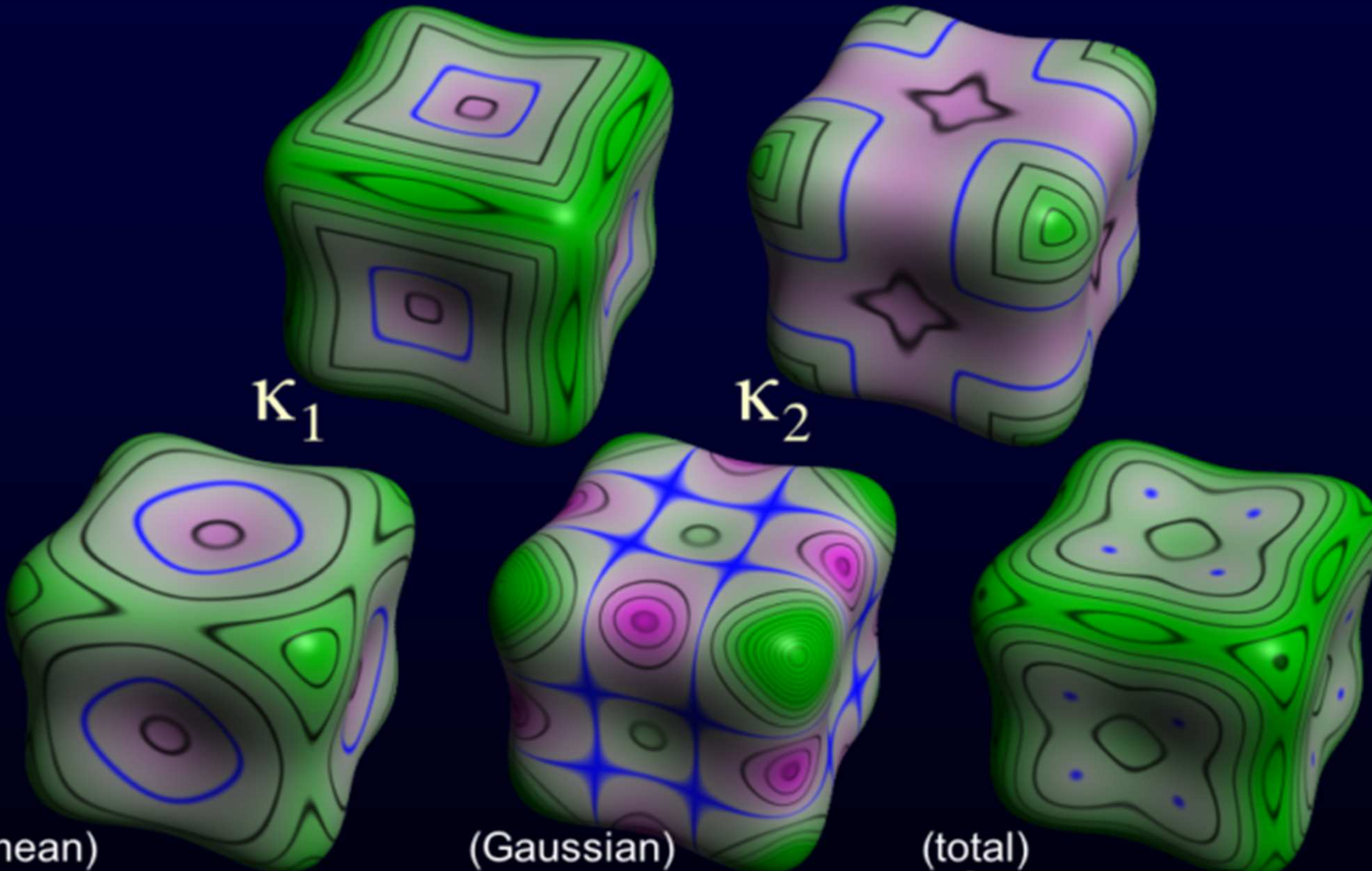



Original TF | Boundaries (gradient)

Multi-Dimensional TFs

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

Curvature measures



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

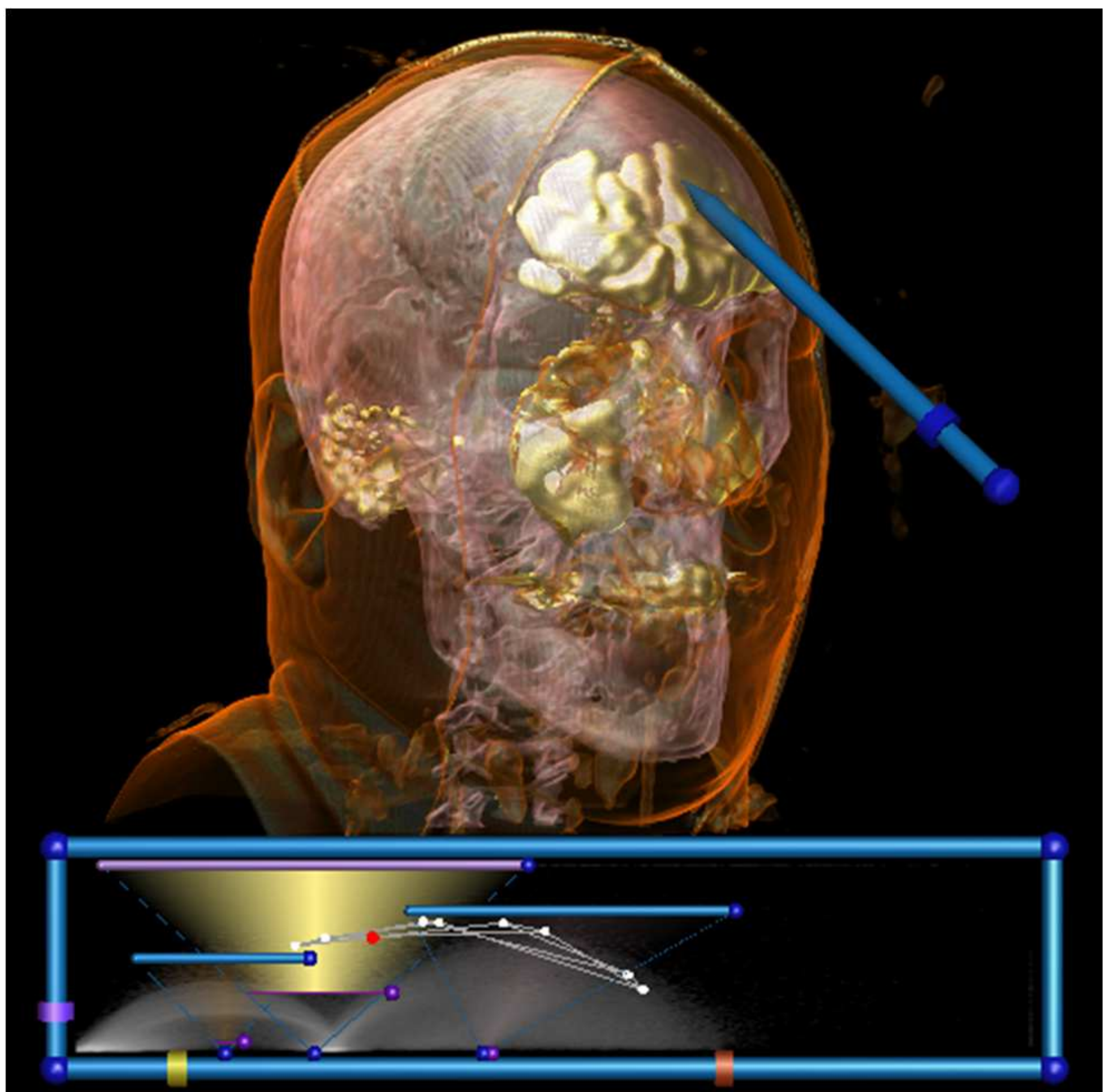
(Gaussian) $\kappa_1 \kappa_2$

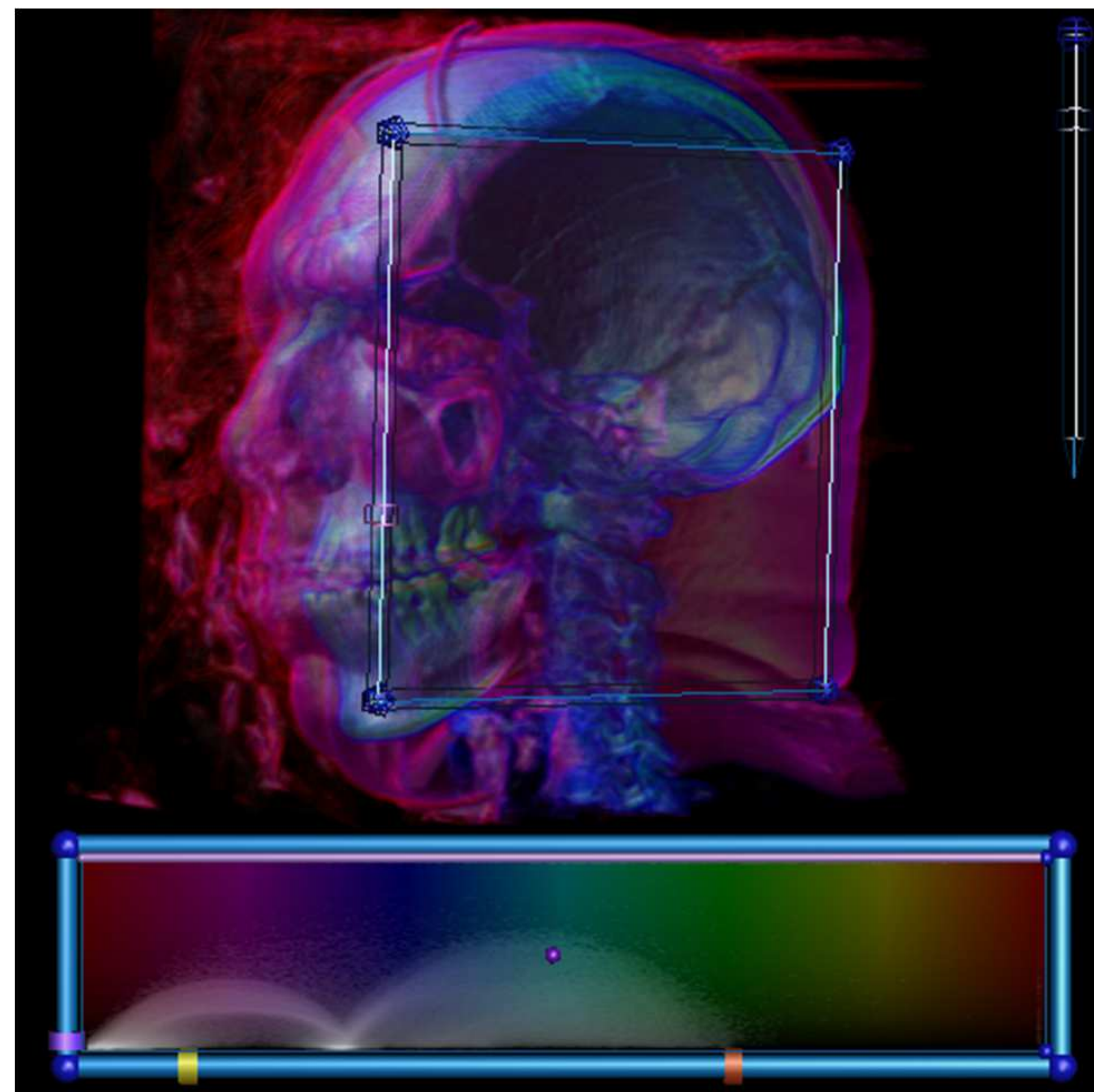
(total) $\sqrt{\kappa_1^2 + \kappa_2^2}$

Different Interaction

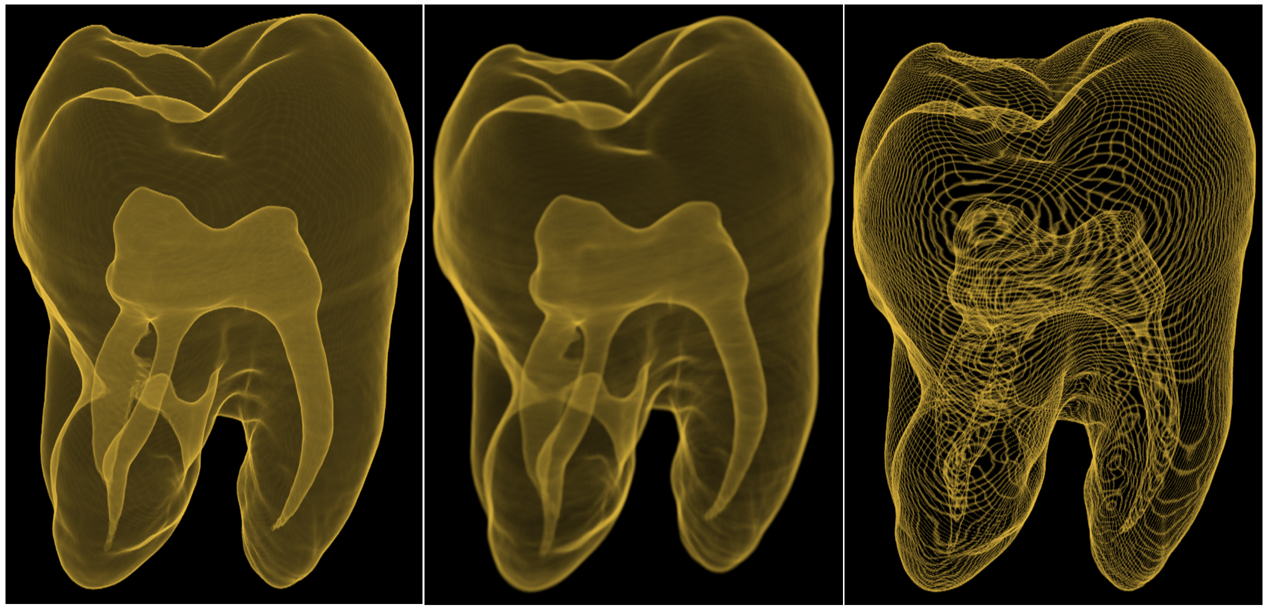
"Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets" Kniss, Kindlmann, Hansen: Vis '01

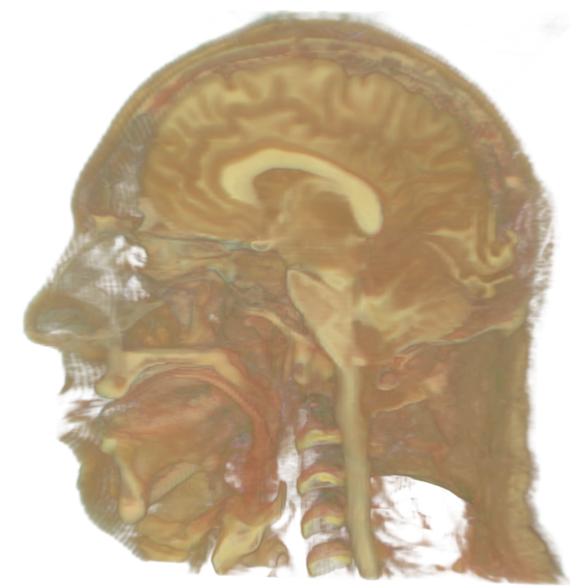
- Make things opaque by pointing at them
- Uses 3D transfer functions (value, 1st, 2nd derivative)
- "Paint" into the transfer function domain





Multidimensional gaussian transfer functions





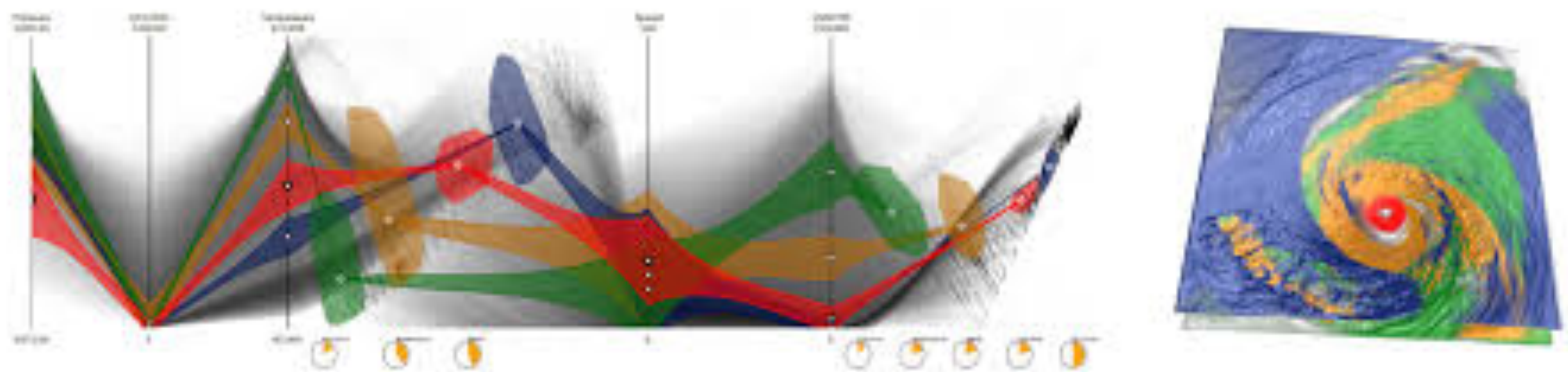
$$\mathrm{GTF}(\vec{v},\vec{c},\mathbf{K}) = e^{-(\vec{v}-\vec{c})^T\mathbf{K}^T\mathbf{K}(\vec{v}-\vec{c})}$$

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

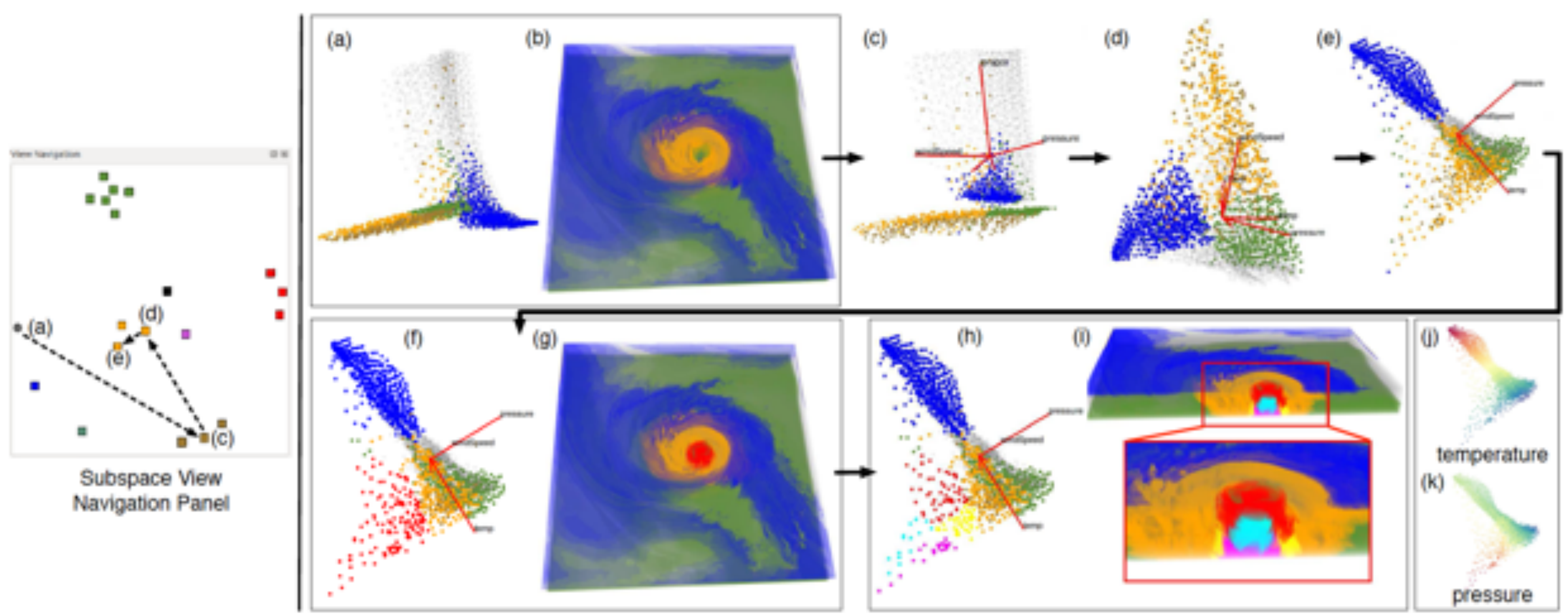
- Analytical integration of any-dimensional transfer functions, summed together as a multivariate Gaussian.
- For data with 2, 3, 4, etc. fields
- Piecewise-linear integration along the ray using compositing

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.

Course Notes 28

Real-Time Volume Graphics

Klaus Engel
Siemens Corporate Research, Princeton, USA

Markus Hadwiger
VR VIS Research Center, Vienna, Austria

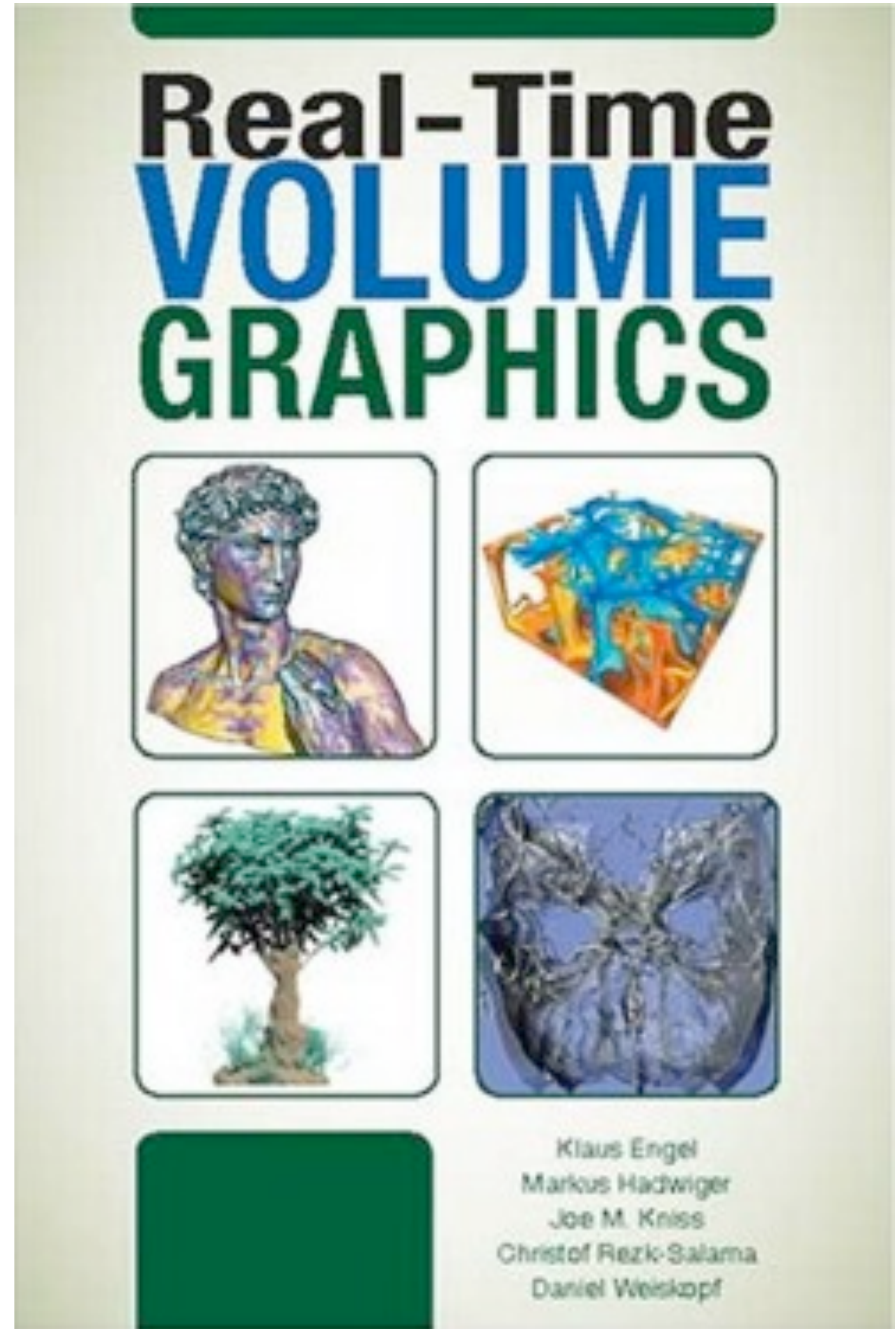
Joe M. Kniss
SCI, University of Utah, USA

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

Next lectures

- 10-27: Project feedback day
- 10-29: Janet Iwasa: molecular visualization & animation
- 11-3: Visualizing tabular data
- 11-5: Visualizing graphs and trees
- 11-10: Isosurfaces
- 11-12: Vector and Tensor Fields