VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Data Visualization

460-4120

Fall 2023

Last update 11. 10. 2023

Scattered Point Interpolation

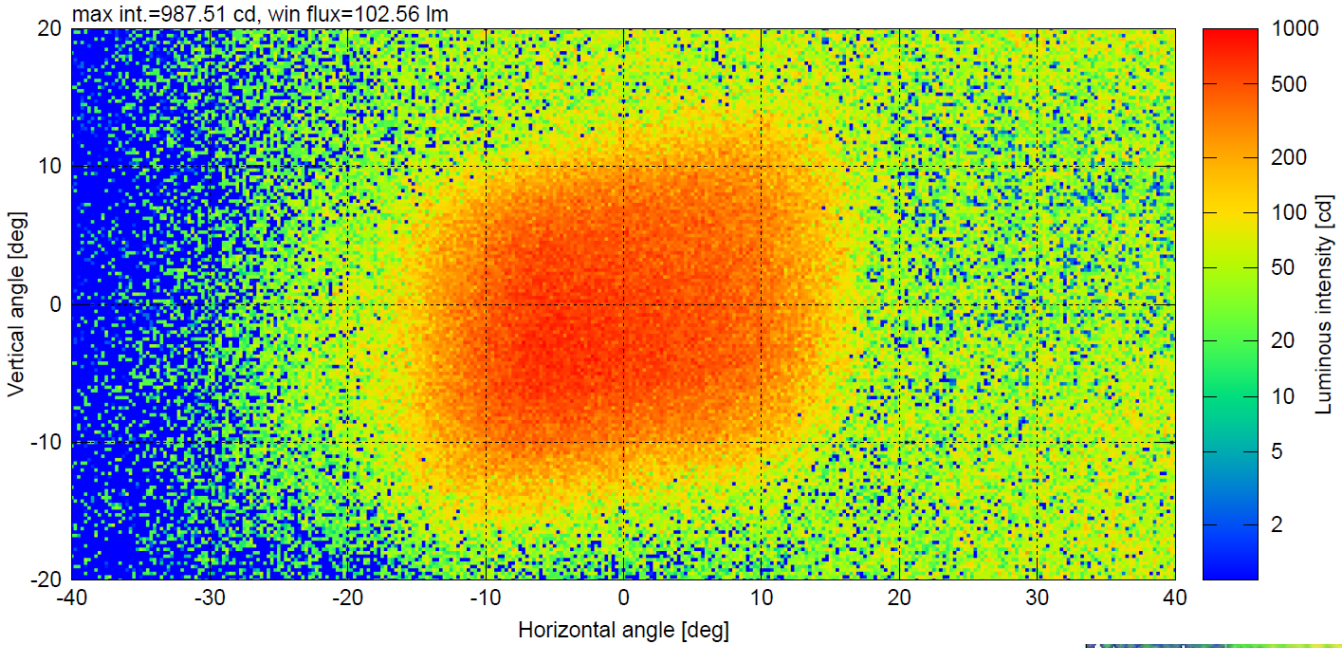
- Sometimes, we have no explicit cell information connecting the sample points with a tiling of some domain
- A scattered (e.g. 3D) point sets – point clouds, i.e. $\{\boldsymbol{p}_i, f_i\}$
- How to reconstruct some continuous signal from these sets

$$\tilde{f} = \sum_{i=1}^{N} f_i \phi_i$$

- There are, basically, two ways to do this:
  - Constructing a grid from scattered points
  - Gridless interpolation

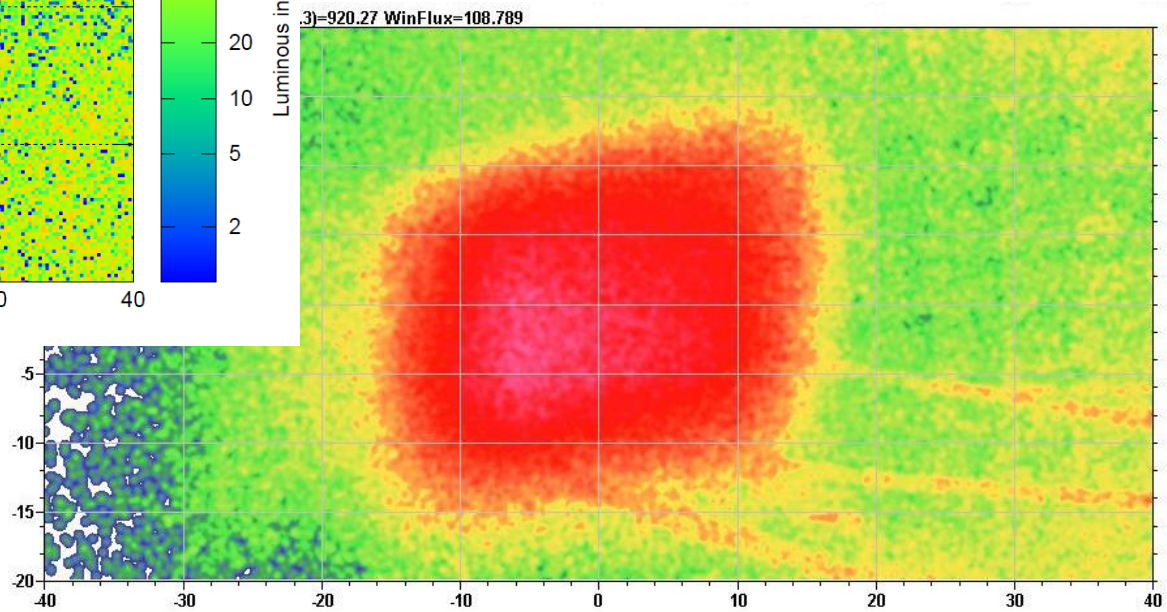
Motivation

„Despite its popularity, the rainbow color map has been shown to be a poor choice for a color map in nearly all problem domains.“ MORELAND, Kenneth. Diverging color maps for scientific visualization. ISVC 2009.



↑ Nearest-neighbor interpolation ↑

↓ Scattered point interpolation ↓



Scattered Point Interpolation

- Constructing a grid from scattered points
  - Quite straightforward, e.g. construct structured grid with 2D cells (triangles) which have $\boldsymbol{p}_i$ as vertices
  - But…
    - Triangulation of large and complex point clouds can be tricky
    - Retriangulation of moving point set can be a costly operation
    - Storing the cell information can double memory demands

Gridless Interpolation

- We need a set of gridless basis functions
- Should respect the properties of the orthogonality and normality as much as possible
- There are several ways to do this, e.g. **radial basis functions** (RBFs)

$$\Phi\colon \mathbb{R}^d \to \mathbb{R} \text{ where } \Phi(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|) = \phi(r)$$

$$\text{or more generally } \Phi_i(\boldsymbol{x}) = \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|) = \phi(r)$$

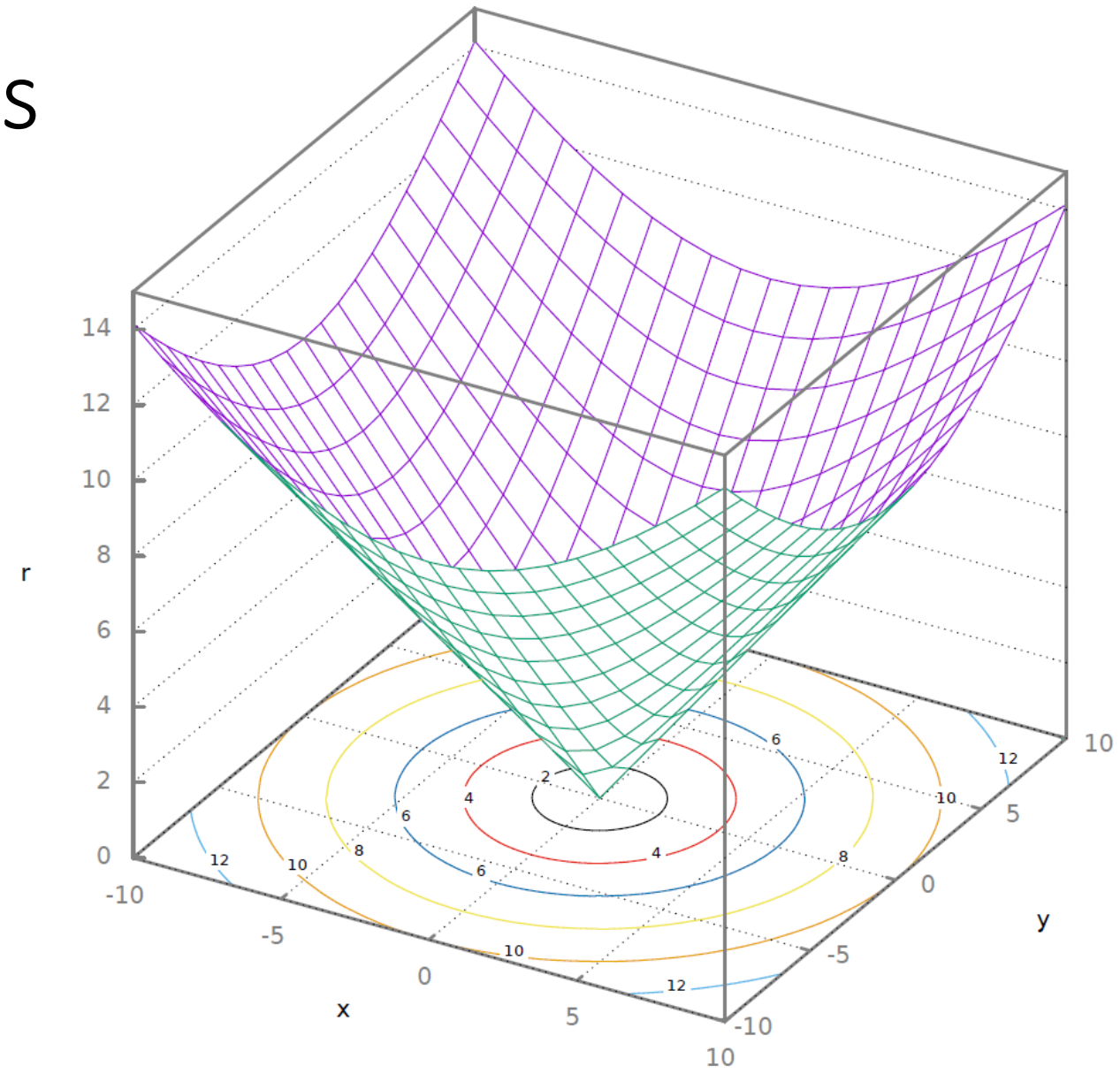
- Φ depends only on the distance of the given point $\boldsymbol{x}$ from the origin (or some other fixed point $\boldsymbol{x}_i$). We usually choose Euclidean distance

$$r = \|x\| = \sqrt{\sum_{i=1}^{d} x_i^2}$$

Radial Basis Functions

- Euclidean distance (L2-norm)

$$r = \|x\| = \sqrt{\sum_{i=1}^{d} x_i^2}$$



Radial Basis Functions

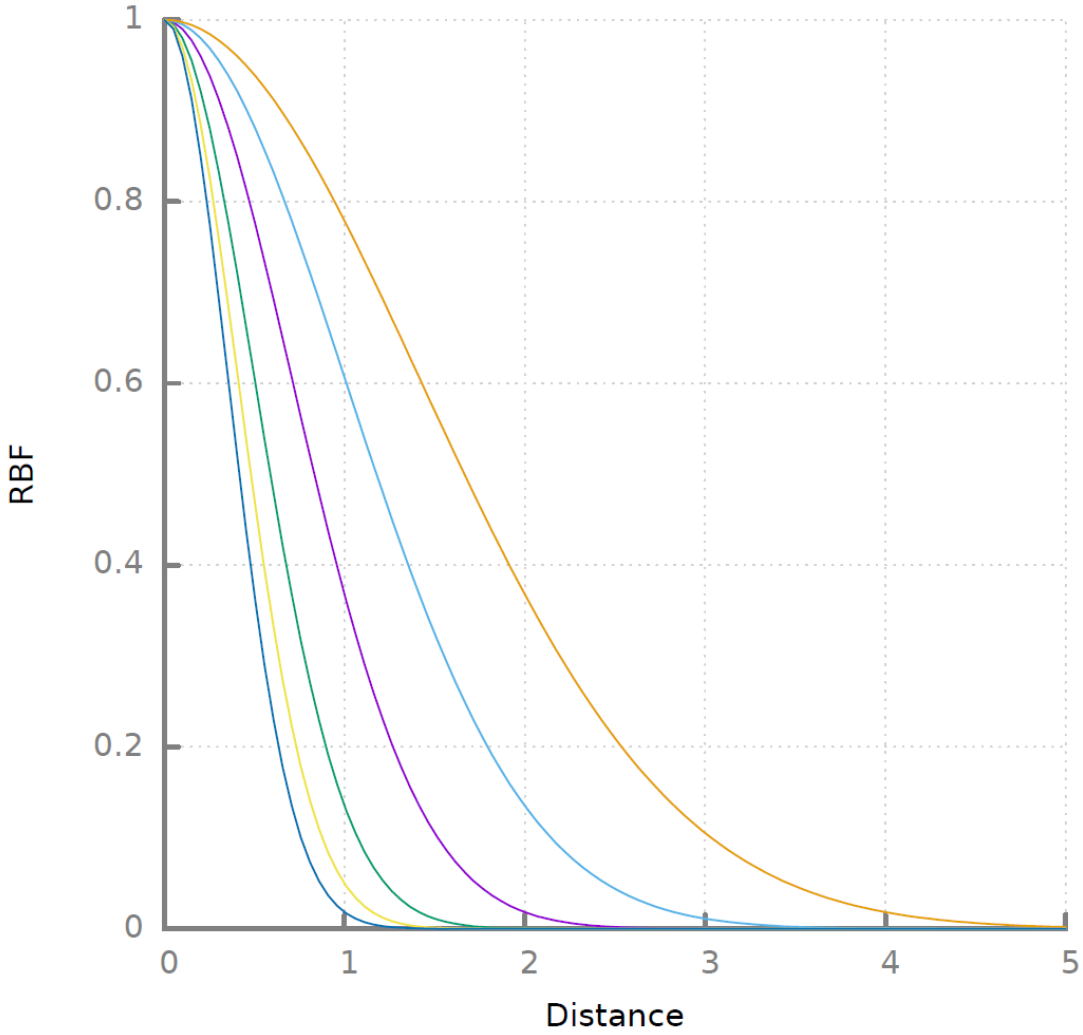
- RBFs smoothly drop from 1 at the origin to 0 at the infinity
- In practice, we limit the range of non zero values, i.e. we limit the range of the basis function effect to its closest neighborhood
  - Computationally efficient
  - Does not smooth out the information present in point cloud
- To do so, we specify a radius of influence R beyond which RBF is 0

Radial Basis Functions

- Gaussian function as a RBF

$$\Phi(r) = \begin{cases} e^{-\lambda r^2}, & r < R \\ 0, & r \geq R \end{cases}$$

- Lambda controls the decay speed

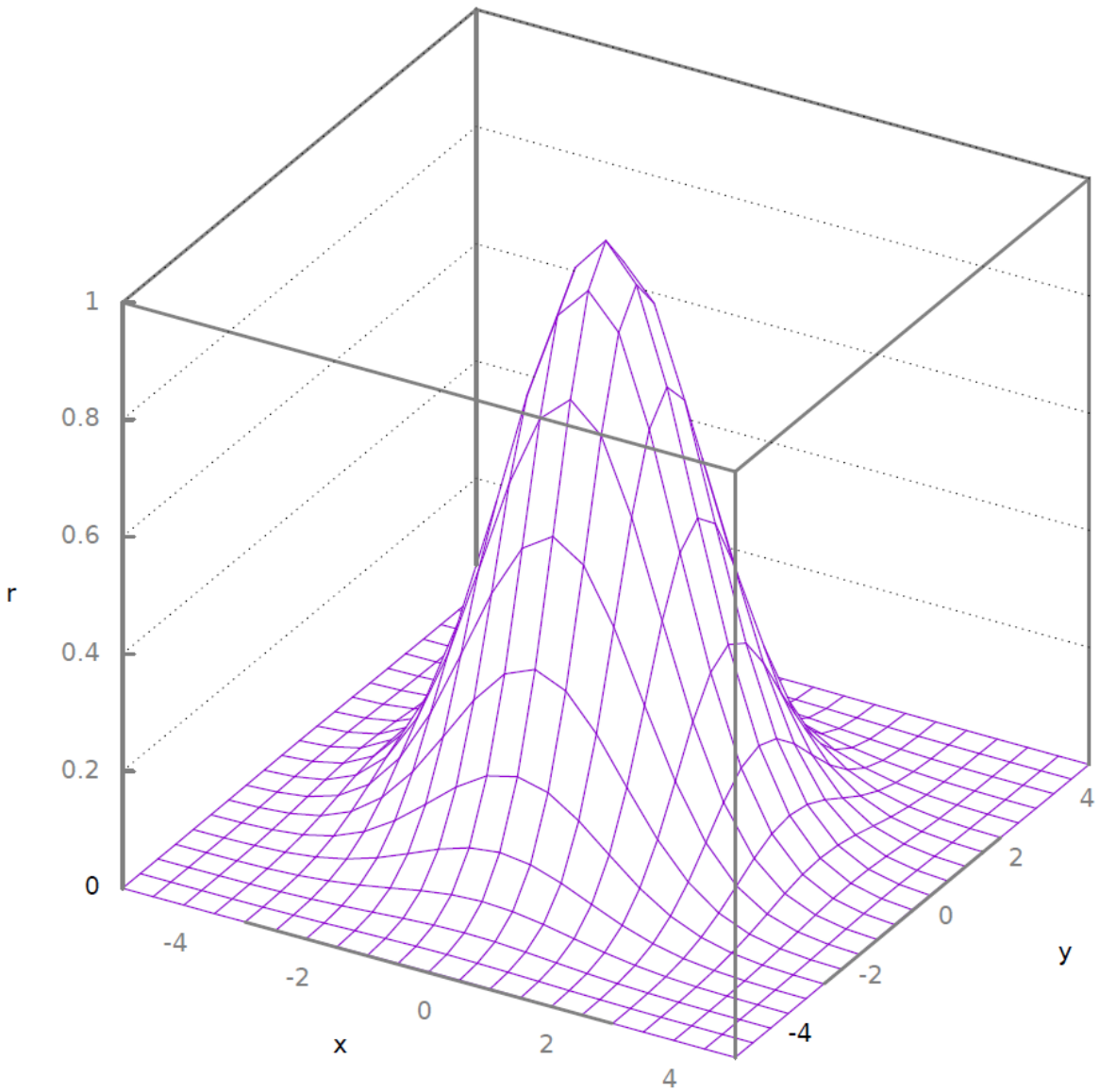


Radial Basis Functions

- Gaussian function as a RBF

$$\Phi(r) = \begin{cases} e^{-\lambda r^2}, & r < R \\ 0, & r \geq R \end{cases}$$

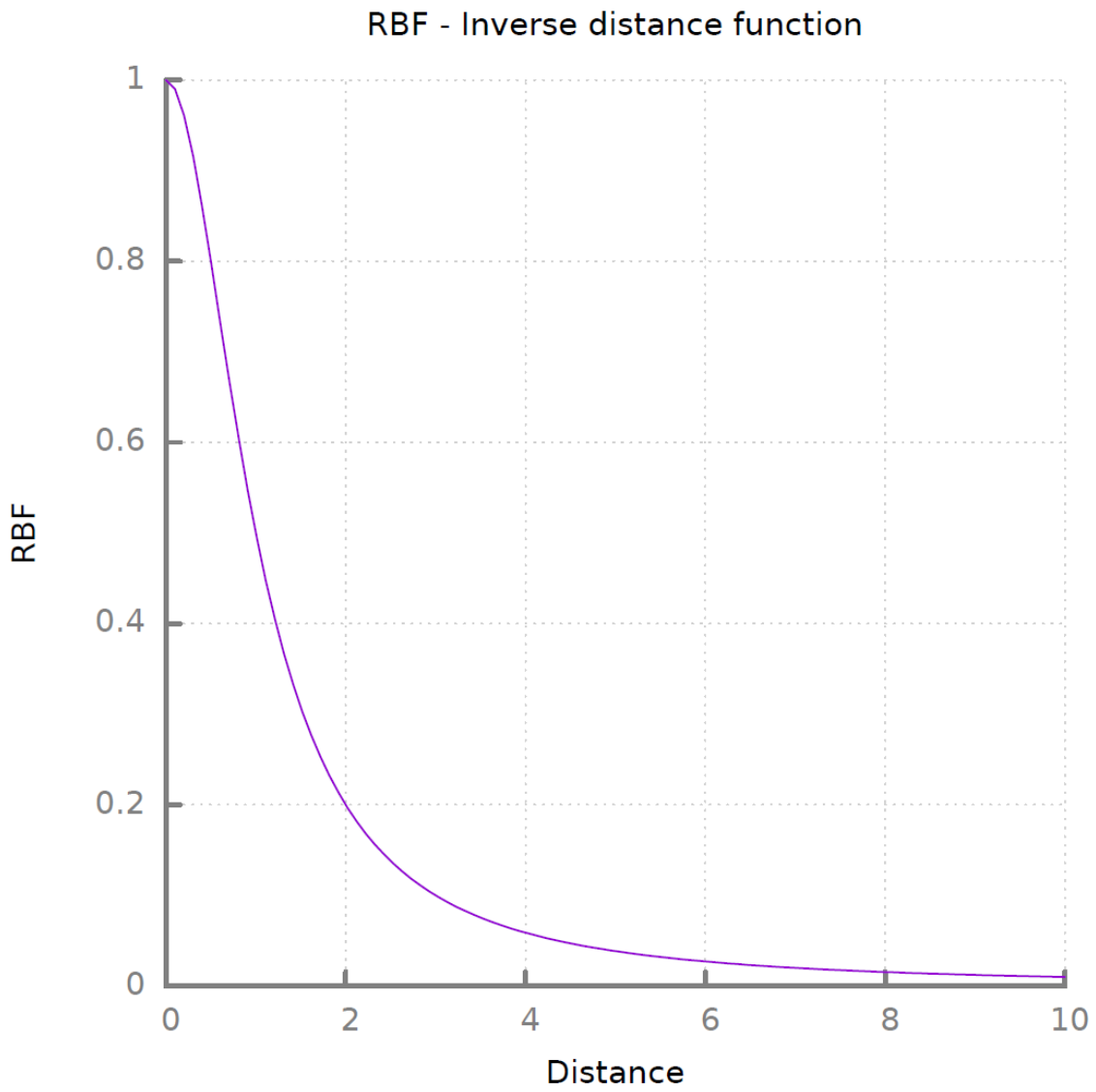
- Lambda controls the decay speed



Radial Basis Functions

- Inverse quadratic function as a RBF

$$\Phi(r) = \begin{cases} \frac{1}{1+r^2}, & r < R \\ 0, & r \geq R \end{cases}$$



Radial Basis Functions

- Other options from C^∞
  - Inverse multiquadric

$$\phi(r) = \frac{1}{\sqrt{1 + (\lambda r)^2}}$$

  - Bump (test) function

$$\phi(r) = \begin{cases} \exp\left(-\frac{1}{1 - (\lambda r)^2}\right) & r < \frac{1}{\lambda} \\ 0 & \text{otherwise} \end{cases}$$

Gridless Interpolation

- The support radius R may be variable for each sample
- Large R leads to smooth reconstruction with high computation cost
- Lower R vice versa
- For uniformly sampled points: $R_i = 1/\sqrt{\rho}$
  - ρ is the surface point density
- For nonuniformly sampled points: the average interpoint distance in the neighborhood of point $\boldsymbol{p}_i$

Gridless Interpolation

- The goal is to construct a continuous surface from irregularly spaced data points
- Application areas include engineering, data mining, computer graphics, and image processing
- There are a number of solutions to the scattered data interpolation
- In 1968, Shepard proposed an interpolation method that creates a surface based on a weighted average of values at sample point

Scattered Point Interpolation

- Shepard's interpolation method (1968) using inverse distance functions (aka Inverse Distance Weighting or IDW)

$$\tilde{f}(p) = \frac{\sum_{i=0}^{N-1} \phi_i(p) f_i}{\sum_{i=0}^{N-1} \phi_i(p)} \qquad \phi_i(p) = \frac{1}{||p - p_i||_2^2}$$

- Results are very similar to Gaussian RBF methods
- Weighting function accords too much influence to data points that are far away from the query point

Scattered Point Interpolation

- Franke and Neilson (1980) modified the weighting function to have local support

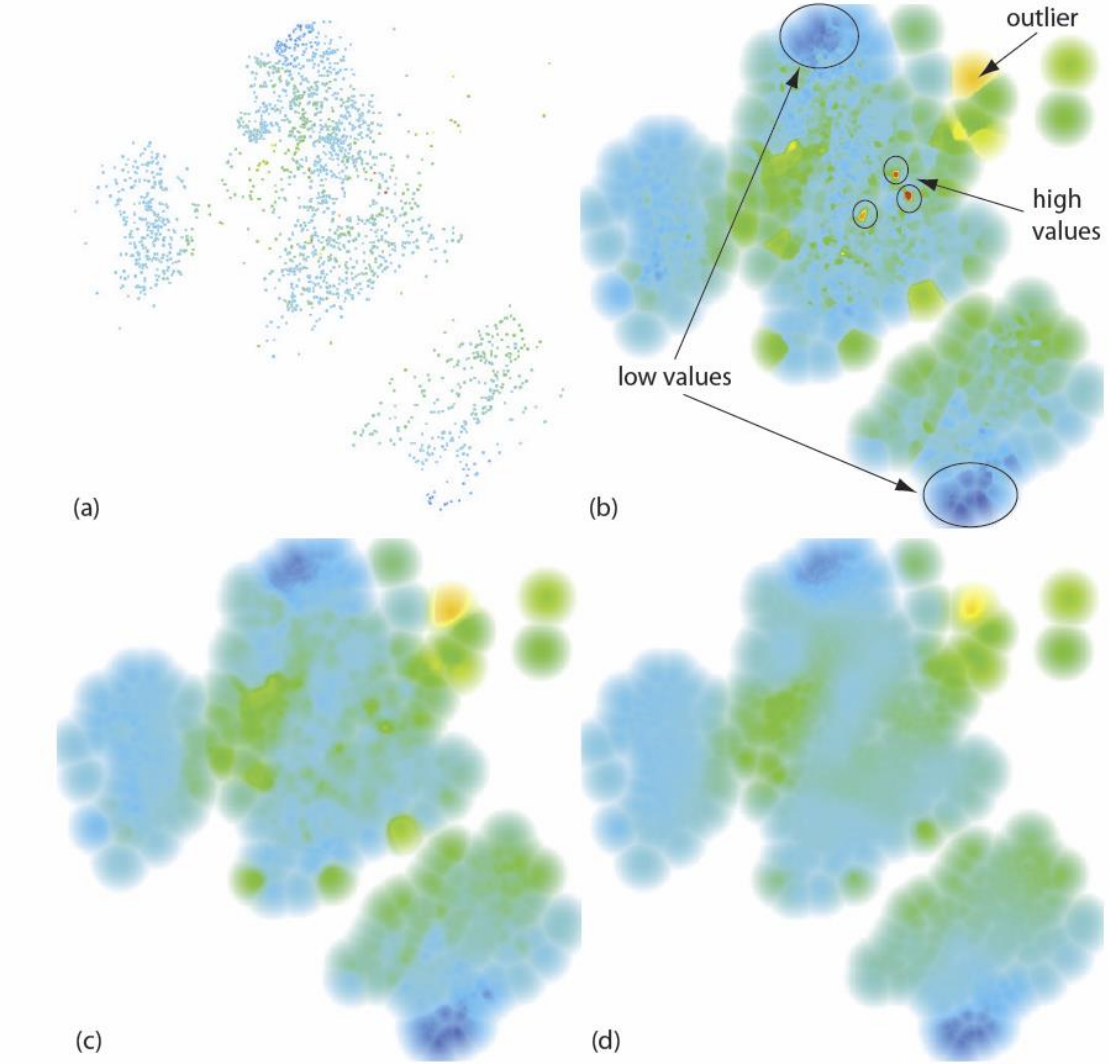
$$\phi_i(p) = \left(\frac{\max\{0, R_p - ||p - p_i||_2\}}{R_p ||p - p_i||_2} \right)^2$$

- The radius of influence around the point *p* is defined as follows

$$R_p = \max_{i \in K} ||p - p_i||_2$$

- K is the set of k nearest neighbors of query point *p*

Scattered Point Interpolation



Shepard interpolation of a scalar signal from a 2D point cloud.

Alexandru C. Telea, Data Visualization: Principles and Practice

Scattered Point Interpolation

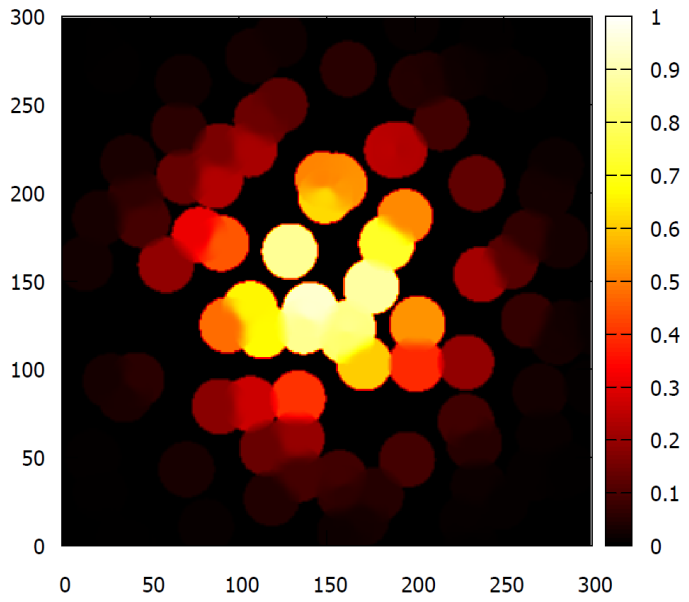
- Previous picture shows the following Shepard's interpolation

$$\tilde{f}(p \in \mathbb{R}^2) = \frac{\sum_{p_i \in N_{R(p)}} f_i e^{-\lambda \left(\frac{\| p - p_i \|}{R(p)} \right)^2}}{\sum_{p_i \in N_{R(p)}} e^{-\lambda \left(\frac{\| p - p_i \|}{R(p)} \right)^2}}$$

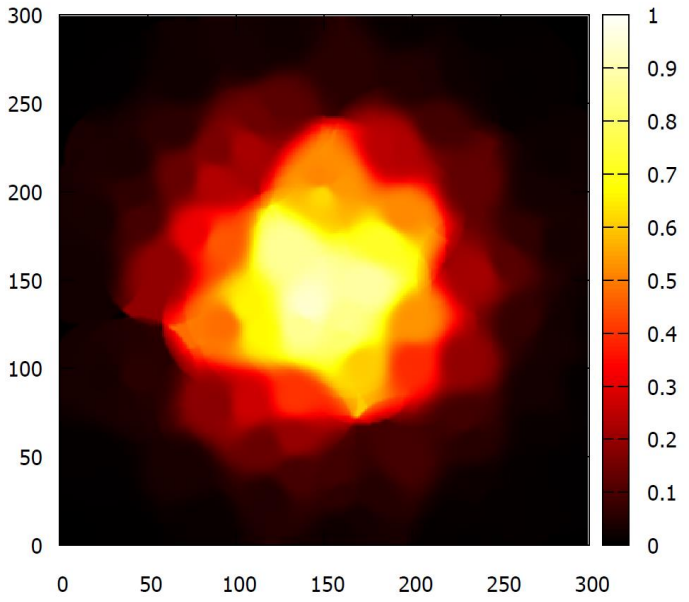
- Gaussian RBF reaches near-zero values at the distance $r = 1$
- The set $N_{R(p)}$ contains all neighbors $\boldsymbol{p}_i$ located within a radius $R(p)$ to the current point *p*
- $R(\boldsymbol{p}) = d(\boldsymbol{p}) + \varepsilon$ is set to the distance $d(\boldsymbol{p})$ between $\boldsymbol{p}$ and its closest data point $\boldsymbol{p}_i$

Scattered Point Interpolation

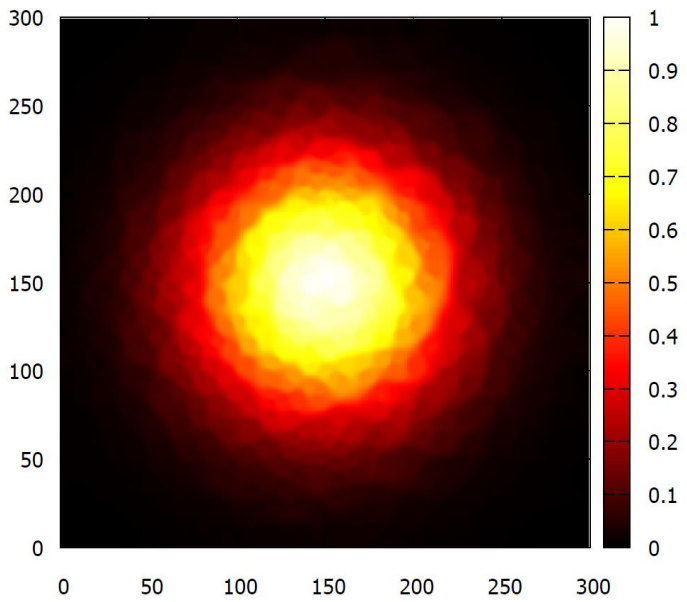
- Modified Shepard's interpolation results



N = 100, r = 0.1



N = 100, r = 0.5



N = 1000, r = 1.0

Scattered Point Interpolation

- K-nearest neighbors (k-NN) libraries - searching for points in space that are close to a given query point:
  - FLANN: Fast Library for Approximate Nearest Neighbors
  - http://www.cs.ubc.ca/research/flann/
  - ANN: A Library for Approximate Nearest Neighbor Searching
  - https://www.cs.umd.edu/~mount/ANN/
  - Annoy: Approximate Nearest Neighbors Oh Yeah
  - https://github.com/spotify/annoy

Another View on Gridless Interpolation

- Let us consider an interpolant in the form

$$s(\boldsymbol{x}) = \sum_{i=1}^{n} \lambda_i \phi(\|\boldsymbol{x}_i - \boldsymbol{x}\|) \ ,$$

providing that $s(\boldsymbol{x}_i) = f_i$ for some dataset $\{\boldsymbol{x}_i, f_i\}$.

- We need to compute λ_i such that

$$f_i = \sum_{j=1}^{n} \lambda_j \phi(\|\boldsymbol{x}_j - \boldsymbol{x}_i\|)$$

yielding set of n linear equations for n unknowns λ_j

$$A = \begin{bmatrix} \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_1\|) & \cdots & \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_1\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_n\|) & \cdots & \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_n\|) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

Color Maps

- Process of mapping scalar values to colors
- The choice of color map is important to allow the viewer easily perform reverse mapping back to scalar values
- The most common color map used in scivis is the rainbow

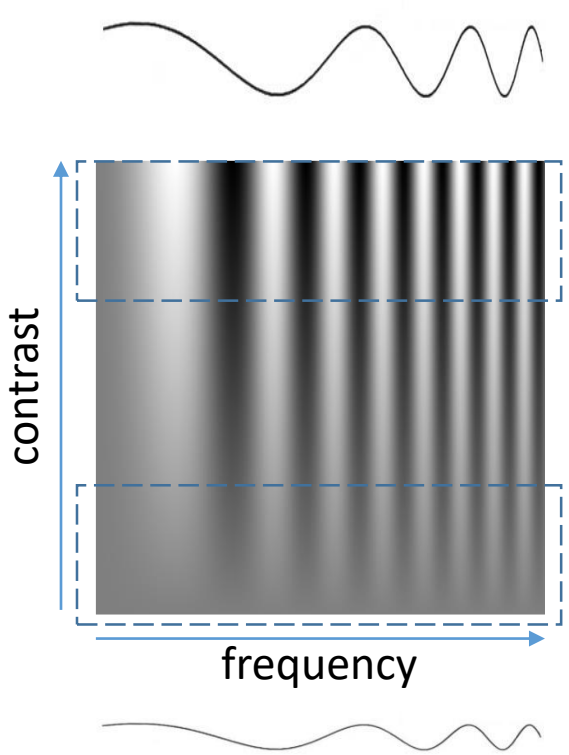
  which cycles through all of the most saturated colors ordered by wavelength
- This color map is widely used in many popular visualization toolkits although:
  - the colors do not follow any natural perceived ordering,
  - the perceptual changes in the colors are not uniform,
  - and it is sensitive to deficiencies in vision

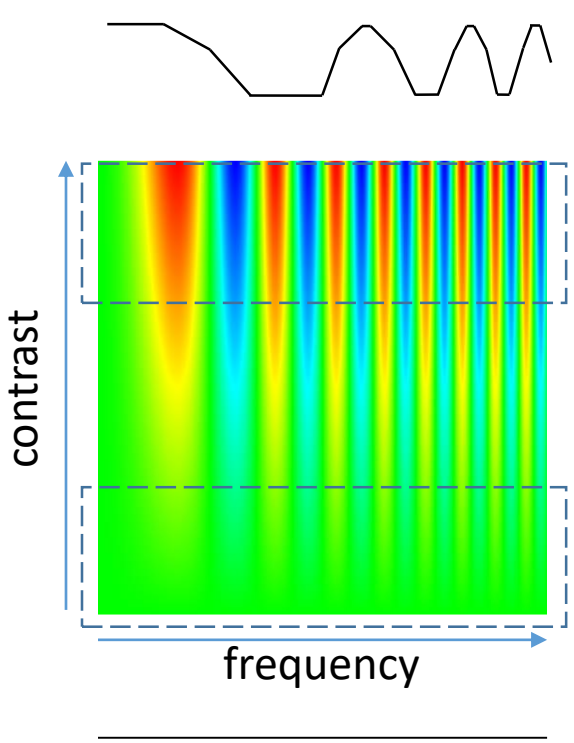
Better Color Map: Grayscale

- A simple grayscale color map completely devoid of any chromaticity

  and relies entirely on luminance
- But it also has disadvantages:
  - Perception of brightness is subject to the brightness of the surroundings
  - Low dynamic range (8 bits produce only 256 unique shades of gray)
  - Inference with surface shading (lighting is important for perceiving 3D shapes)
- Both problems can be reduced by adding a chromaticity shift

Color Maps Comparison





The rainbow appears less smooth in the high-contrast region

The rainbow hides the variation in the low-contrast region

Isoluminant Color Maps

- Problem: maps with big changes in luminance hide shading cues

- Solution: isoluminant maps maintains a constant (perceptual) luminance and relies entirely on chromatic shifts

- The green to red color map uses a pair of opponents colors, but the mauve taupe is much easier to see by individuals with deuteranope or protanopic vision
  (missing green) (missing red)

Isoluminant Color Maps

- But it also has disadvantages:
  - Human perception is less sensitive to changes in saturation or hue than changes in luminance
  - Keeping the luminance constant also restricts the colors that can be represented
  - They also tend to look dull and ugly over a more vibrant color maps

- In general, these color maps are the most commonly used ones

Scalar Visualization and Color Maps 1/2

- Effective color maps meets following criterion:
  - Absolute values – tell the absolute data values at all points
  - Value ordering – tell which of two data values is greater
  - Value difference – tell what is the difference of two given values
  - Selected values – tell which point takes a particular data value
  - Value change – tell the speed of change of data values

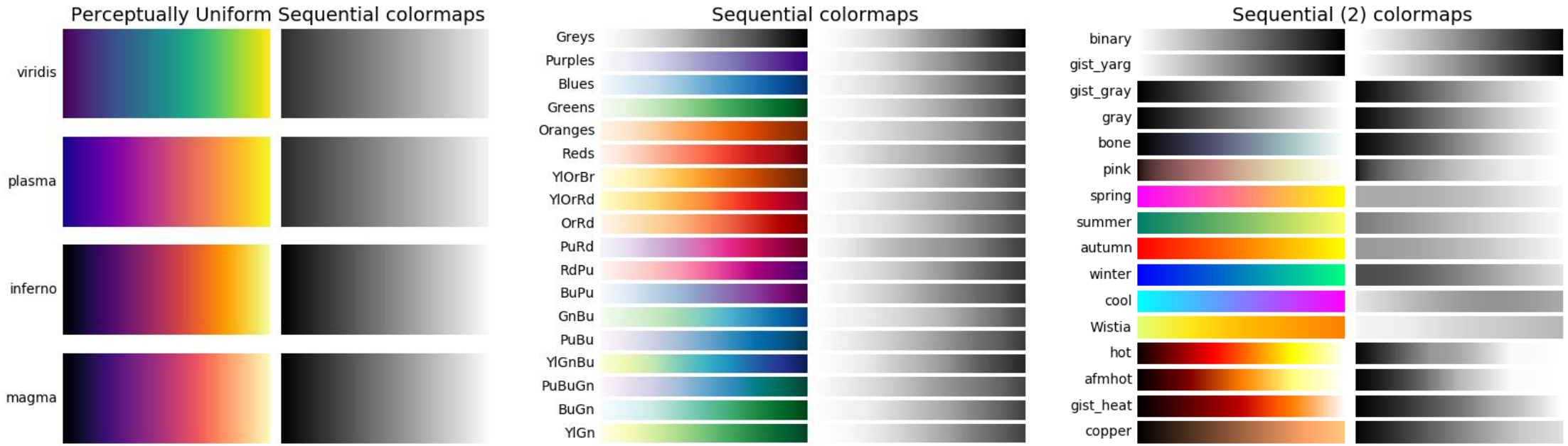
Scalar Visualization and Color Maps 2/2

- Effective color maps meets following criterion:
  - The map yields images that are aesthetically pleasing
  - The map has a maximal perceptual resolution
  - Interference with the shading of 3D surfaces is minimal
  - The map is not sensitive to vision deficiencies
  - The order of the colors should be intuitively the same for all people
  - The perceptual interpolation matches the underlying scalars of the map

Scalar Visualization and Color Maps

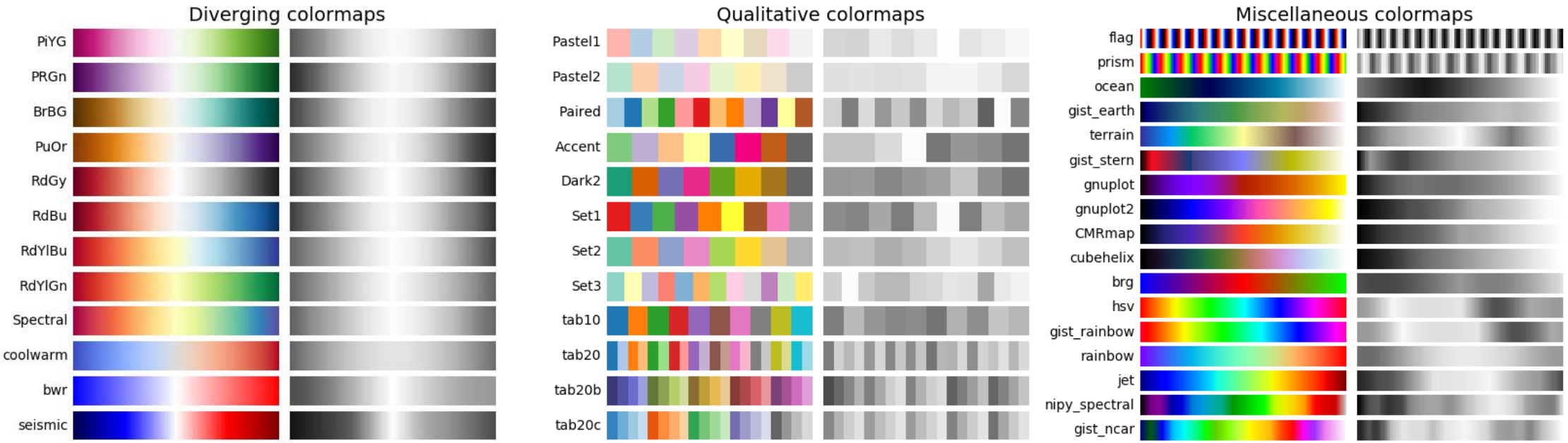
- Grayscale (luminance)
- Rainbow
- Two-color/hue – blue-yellow
  - Different perceived luminance ease color ordering
- Diverging – blue-white-red, green-yellow-red
- Heat map – black-red-yellow-white
- Zebra – emphasizes rapid value variation
- Color banding – isolines
- http://www.zonums.com/online/color_ramp
- https://colorbrewer2.org

Scalar Visualization and Color Maps



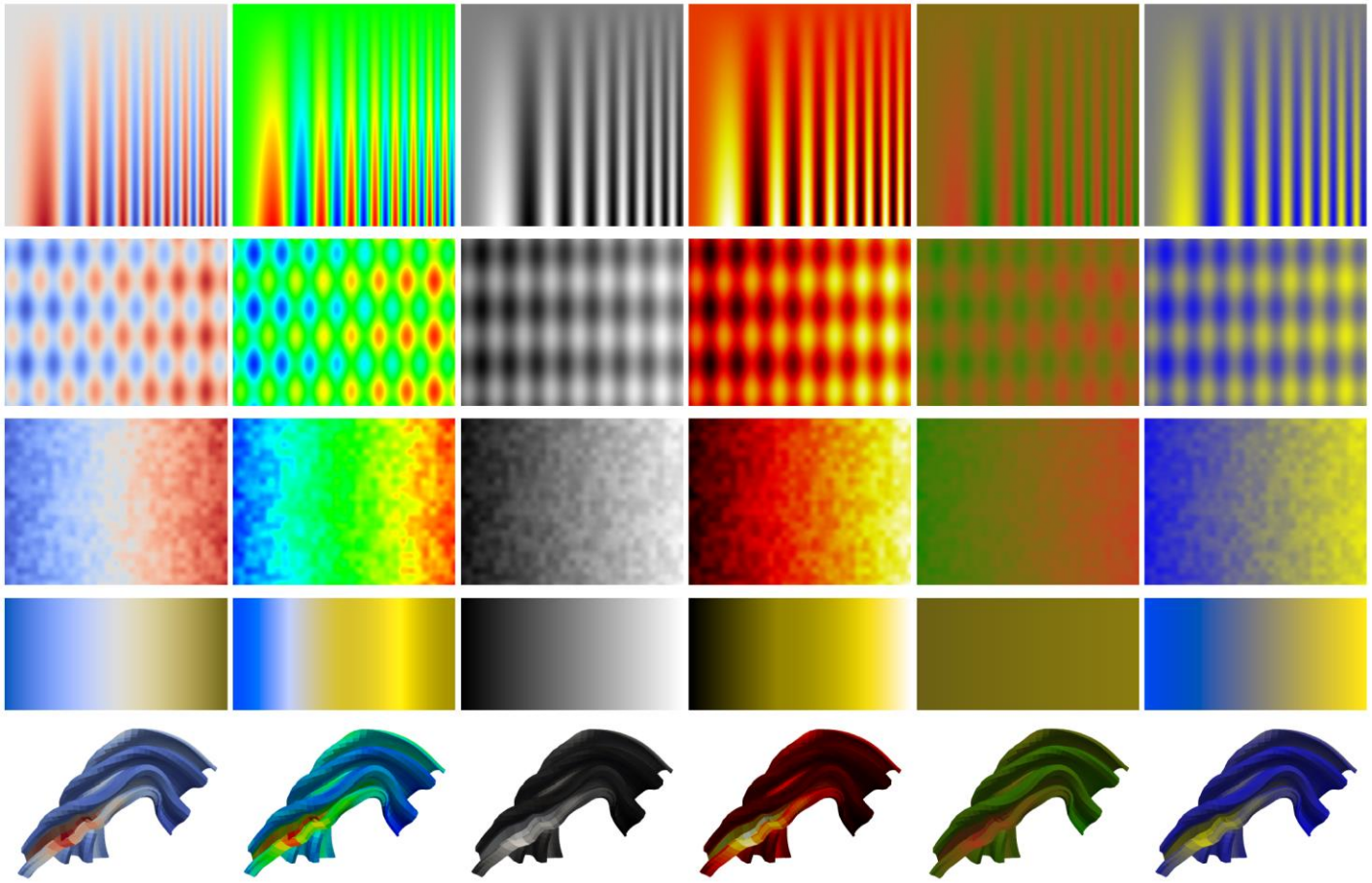
Source: matplotlib.org
https://matplotlib.org/users/colormaps.html

Scalar Visualization and Color Maps



Source: matplotlib.org
https://matplotlib.org/users/colormaps.html

Scalar Visualization and Color Maps



Color Map Suited to Scientific Visualization

- A continuous diverging color map

Scalar	Red	Green	Blue
0.0	59	76	192
0.03125	68	90	204
0.0625	77	104	215
0.09375	87	117	225
0.125	98	130	234
0.15625	108	142	241
0.1875	119	154	247
0.21875	130	165	251
0.25	141	176	254
0.28125	152	185	255
0.3125	163	194	255
0.34375	174	201	253
0.375	184	208	249
0.40625	194	213	244
0.4375	204	217	238
0.46875	213	219	230
0.5	221	221	221

Scalar	Red	Green	Blue
0.53125	229	216	209
0.5625	236	211	197
0.59375	241	204	185
0.625	245	196	173
0.65625	247	187	160
0.6875	247	177	148
0.71875	247	166	135
0.75	244	154	123
0.78125	241	141	111
0.8125	236	127	99
0.84375	229	112	88
0.875	222	96	77
0.90625	213	80	66
0.9375	203	62	56
0.96875	192	40	47
1.0	180	4	38
.			

Scalar Visualization and Color Maps

- Color ramps generator: http://www.zonums.com/online/color_ramp

```
Vector3 ColorRamp( const float value, const float min_value, const float max_value,
Vector3 * ramp, const int ramp_size ) {
if ( value >= max_value )
        return ramp[ramp_size - 1];
else {
        float a = ( value - min_value ) / ( ( max_value - min_value ) / ( ramp_size - 1
) );
        const int band = _cast<int>( floor( a ) );
        a -= band;
        return ramp[band] * ( 1 - a ) + ramp[band + 1] * a;
} }
```

Scalar Visualization and Color Maps

- Vector3 **gray_scale_ramp**[] = { Vector3(0, 0, 0), Vector3(255, 255, 255) };

  int gray_scale_ramp_size = 2;

- Vector3 **pn_ramp**[] = { Vector3(210,210,255), Vector3(0,0,255), Vector3(0,0,0), Vector3(255,0,0), Vector3(255,210,210) };

  int pn_ramp_size = 5;

- Vector3 **rainbow_ramp**[] = { Vector3(0,0,0), Vector3(0,0,255), Vector3(0,255,255), Vector3(0,255,0), Vector3(255,255,0), Vector3(255,0,0), Vector3(255,255,255) };

  int rainbow_ramp_size = 7;

- Vector3 **mathlab_ramp**[] = { Vector3(0,0,144), Vector3(0,15,255), Vector3(0,144,255), Vector3(14,255,238), Vector3(144,255,112), Vector3(255,238,0), Vector3(255,112,0), Vector3(238,0,0), Vector3(127,0,0) };

  int mathlab_ramp_size = 9;

- Vector3 **iron_ramp**[] = { Vector3(0,0,0), Vector3(0,0,255), Vector3(255,0,255), Vector3(255,255,0), Vector3(255,255,255) };

  int iron_ramp_size = 5;

Color Spaces

- All color spaces are based on the tristimulus theory: any perceived color can be uniquely represented by a 3-tuple
- In computer applications, the most commonly used color space is RGB
- But the display may have nonlinearities that interfere with the blending and interpolation of colors. When computing physical light effects, it is best to use a color space defined by physical properties of light
- A widely used color space defined by physical light spectra is XYZ

RGB vs sRGB Color Spaces

- RGB color space is any additive color space based on RGB color model that employs RGB primaries (i.e. red, green, and blue chromacities)
- Primary colors are defined by their CIE 1931 color space chromacity coordinates (x, y)
- Specification of any RGB color space also includes definition of white point and transfer function
- sRGB is one of many (but by far the most commonly used) color spaces for computer displays
- Our renderer will produce sRGB images

Specifications of RGB color spaces

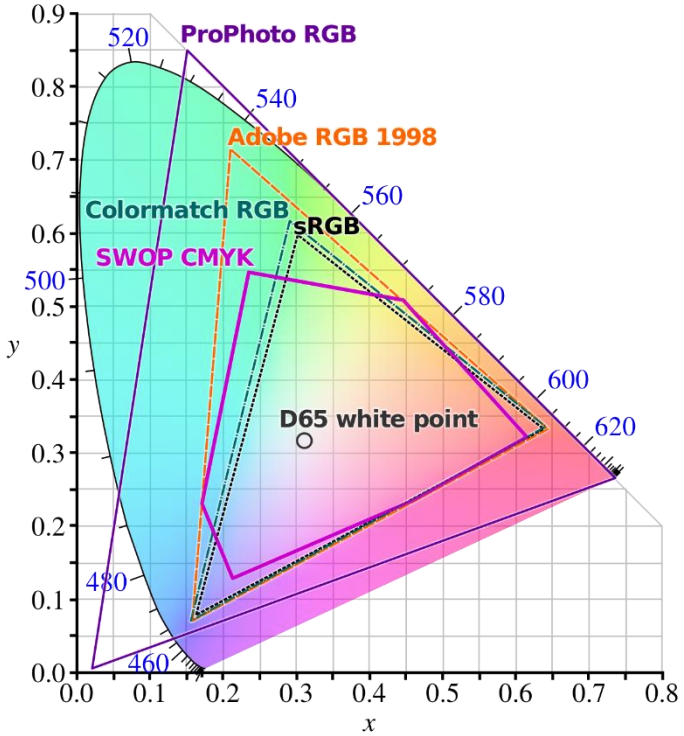
Color space	Standard	Year	Gamut	White point	Primaries						Transfer function parameters				
					Red		Green		Blue		α	β	γ	δ	βδ
					x_R	y_R	x_G	y_G	x_B	y_B	a + 1	$K_0/\varphi = E_t$		φ	K_0
ISO RGB			Limited	floating	floating										
Extended ISO RGB			Unlimited (signed)												
scRGB	IEC 61966-2-2	2003													
sRGB	IEC 61966-2-1	1990, 1996		D65	0.64	0.33	0.30	0.60	0.15	0.06	1.055	0.0031308	$\frac{12}{5}$	12.92	0.04045
HDTV	ITU-R BT.709	1999	CRT	D65	0.64	0.33	0.30	0.60	0.15	0.06	1.099	0.004	$\frac{20}{9}$	4.5	0.018
Adobe RGB 98		1998					0.21	0.71			1	0	$\frac{563}{256}$	1	0
PAL / SECAM	EBU 3213-E, ITU-R BT.470/601 (B/G)	1970					0.29	0.60			1	0	$\frac{14}{5}$	1	0
Apple RGB					0.625	0.34	0.28	0.595	0.155	0.07					
NTSC	SMPTE RP 145 (C), 170M, 240M	1987			0.63		0.31				1.1115	0.0057	$\frac{20}{9}$	4	0.0228
NTSC-J		1987		D93											
NTSC (FCC)	ITU-R BT.470/601 (M)	1953		C	0.67	0.33	0.21	0.71	0.14	0.08	1	0	$\frac{11}{5}$	1	0
eciRGB	ISO 22028-4	1999 (v1), 2007, 2012	Wide	D50							1.16	0.008856	3	9.033	0.08
DCI-P3	SMPTE RP 431-2	2011		Theater	0.68	0.32	0.265	0.69	0.15	0.06	1.055	0.0031308	$\frac{12}{5}$	12.92	0.04045
Display P3	SMPTE EG 432-1	2010		D65											
UHDTV	ITU-R BT.2020, BT.2100	2012, 2016			0.708	0.292	0.170	0.797	0.131	0.046	1.0993	0.018054		4.5	0.081243
Adobe Wide Gamut RGB				D50	0.735	0.265	0.115	0.826	0.157	0.018	1	0	$\frac{563}{256}$	1	0
RIMM	ISO 22028-3	2006, 2012			0.7347	0.2653	0.1596	0.8404	0.0366	0.0001	1.099	0.0018	$\frac{20}{9}$	5.5	0.099
ROMM RGB, ProPhoto RGB	ISO 22028-2	2006, 2013									1	0.001953125	$\frac{9}{5}$	16	0.031248
CIE RGB		1931		E			0.2738	0.7174	0.1666	0.0089					
CIE XYZ		1931	Unlimited		1	0	0	1	0	0	1	0	1	1	0

Source: https://en.wikipedia.org/wiki/RGB_color_space

Color Gamut

- A color gamut is defined as a range of colors that a particular device is capable of displaying or recording
- It usually appears as a closed area of primary colors in a chromaticity diagram. The missing dimension is the brightness, which is perpendicular to the screen or paper
- Color gamut is displayed as a triangular area enclosed by color coordinates corresponding to the red, green, and blue color

CIE 1931 XYZ color space



Color Gamut

- sRGB - by far the most commonly used color space for computer displays
- NTSC – standard for analog television
- Adobe RGB (1998) - encompass most of the colors achievable on CMYK color printers, but by using RGB primary colors on a device such as a computer display
- DCI-P3 – space for digital movie projection from the American film industry
- EBU – European color space surpassing the PAL standard
- Rec. 709 - shares the sRGB primaries, used in HDTVs
- Rec. 2023 – 4K or 8K resolution at 10 or 12 bits per channel
- Remember that 72% NTSC is not sRGB (which is often claimed). Matching the ratios of the color gamut areas does not necessarily guarantee the ability to achieve the same image (the degree of overlap of the triangles is important and not the ratio of their areas).

Linear sRGB and Gamma Compressed sRGB

- Images displayed on monitors are encoded in nonlinear sRGB color space to compensate the transformation of brightness the monitor does
- Our render has to work in linear space as we are using linear operations with colors
- Every texel and material color have to be processed in linear sRGB color model
- Only the final color values stored in framebuffer are converted back to gamma compensated sRGB values
- Issue with blending two sRGB colors in non-linear color space:

$\boldsymbol{C}_{srgb} = \alpha \boldsymbol{A}_{srgb} + (1-\alpha)\boldsymbol{B}_{srgb}$ doesn work well

$\boldsymbol{C}_{rgb} = \alpha \boldsymbol{A}_{rgb} + (1-\alpha)\boldsymbol{B}_{rgb}$ correct

$\boldsymbol{C}_{srgb} = ToSRGB\left(\alpha\, ToRGB\left(\boldsymbol{A}_{srgb}\right) + (1-\alpha)\, ToRGB\left(\boldsymbol{B}_{srgb}\right)\right)$ correct

Note the undesirable dark silhouettes when mixing two colors directly in sRGB space (created in paint.net)

Gamma Correction

- The human visual system response is logarithmic, not linear, resulting in the ability to perceive an incredible brightness range of over 10 decades
- Gamma characterizes the reproduction of tone scale in an imaging system. Gamma summarizes, in a single numerical parameter, the nonlinear relationship between code value (in an 8-bit system, from 0 through 255) and luminance. Nearly all image coding systems are nonlinear, and so involve values of gamma different from unity
- The main purpose of gamma correction is to code luminance into a perceptually-uniform domain, so as optimize perceptual performance of a limited number of bits in each channel

Source: POYNTON, Charles A. The rehabilitation of gamma, 2000.

sRGB Transfer Functions

- Function returns gamma-expanded (or linear) sRGB values from gamma-compressed (or non-linear) sRGB values

```
float c_linear( float c_srgb, float gamma = 2.4f )
{
  if ( c_srgb <= 0.0f ) return 0.0f;
  else if ( c_srgb >= 1.0f ) return 1.0f;

  assert( ( c_srgb >= 0.0f ) && ( c_srgb <= 1.0f ) );

  if ( c_srgb <= 0.04045f )
  {
    return c_srgb / 12.92f;
  }
  else
  {
    const float a = 0.055f;
    return powf( ( c_srgb + a ) / ( 1.0f + a ), gamma );
  }
}
```

- Function returns gamma-compressed (or non-linear) sRGB values from gamma-expanded (or linear) sRGB values

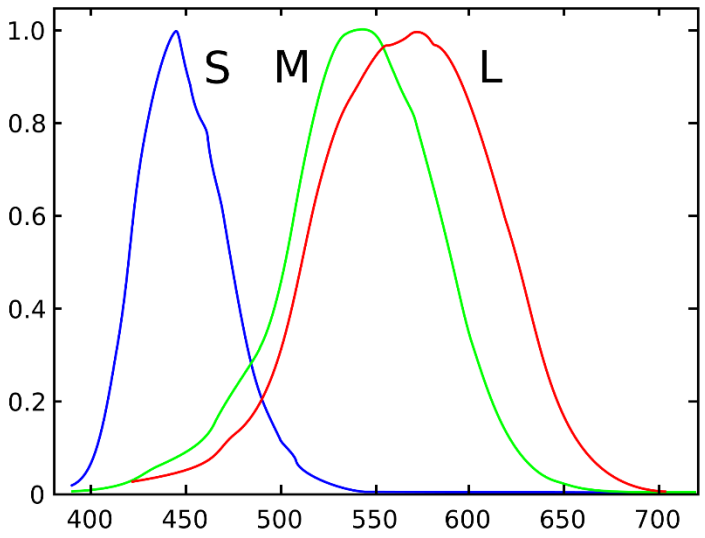
```
float c_srgb( float c_linear, float gamma = 2.4f )
{
  if ( c_linear <= 0.0f ) return 0.0f;
  else if ( c_linear >= 1.0f ) return 1.0f;

  assert( ( c_linear >= 0.0f ) && ( c_linear <= 1.0f ) );

  if ( c_linear <= 0.0031308f )
  {
    return 12.92f * c_linear;
  }
  else
  {
    const float a = 0.055f;
    return ( 1.0f + a )*powf( c_linear, 1.0f / gamma ) - a;
  }
}
```

Spectral Sensitivity of Human Eye

- The human eye with normal vision has three kinds of cone cells (čípek) that sense light (trichromacy vs pentachromacy), having peaks of spectral sensitivity in short ("S", 420 nm – 440 nm), middle ("M", 530 nm – 540 nm), and long ("L", 560 nm – 580 nm) wavelengths
- LMS forms a 3D cone space

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.38971 & 0.68898 & -0.07868 \\ -0.22981 & 1.18340 & 0.04641 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1.91019 & -1.11214 & 0.20235 \\ 0.37095 & 0.62905 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}$$

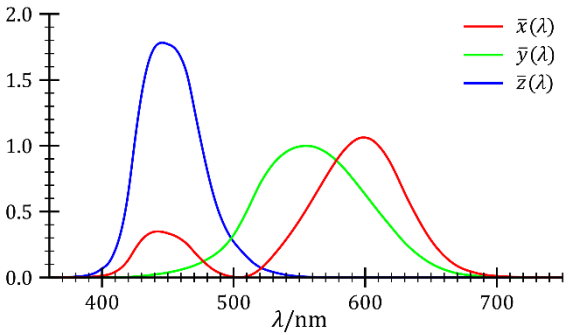


Normalized responsivity spectra of human cone

CIE 1931 Color Space

Can be thought of as the spectral sensitivity curves of three linear light detectors yielding the CIE tristimulus values *X*, *Y* and *Z*. Describe the CIE standard observer.

CIE's Color Matching Functions



- CIE XYZ is a color space deliberately designed so that the Y parameter represents the luminance of a color
- We can compute XYZ values from spectral data $L_{e,\Omega}$ as follows

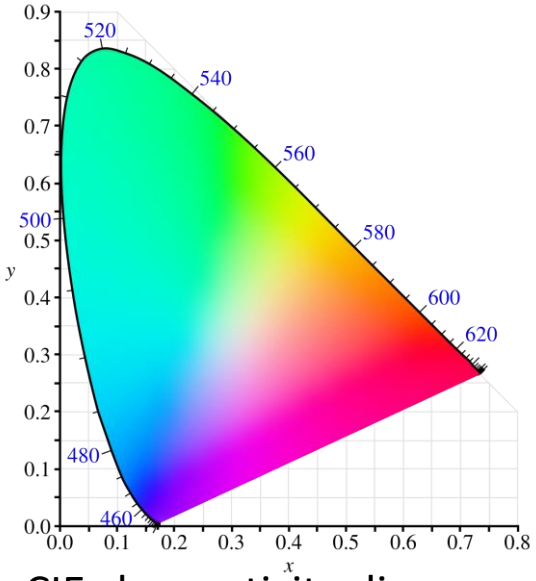
$$\{X, Y, Z\} = \int_{\lambda} L_{e,\Omega}(\lambda)\{\bar{x}, \bar{y}, \bar{z}\}(\lambda)\, \mathrm{d}\lambda \text{ where } \lambda \in \langle 380, 780 \rangle \text{ nm}$$

spectral radiance (W·sr^{-1}·m^{-2}·nm^{-1}) of a reference illuminant

$$X = \frac{Y}{y}x;\ \mathrm{Z} = \frac{Y}{y}(1 - x - y)$$

- CIE xyY is a three dimensional color space defined by (x, y) (chromaticity) and Y (luminance) parameters

$$x = \frac{X}{X+Y+Z};\ \mathrm{y} = \frac{Y}{X+Y+Z};\ z = \frac{Z}{X+Y+Z} = 1 - x - y$$



CIE chromaticity diagram

sRGB Color Space

- Forward

$$\begin{bmatrix} R_{linear} \\ G_{linear} \\ B_{linear} \end{bmatrix} = \begin{bmatrix} 3.240969942 & -1.537383178 & -0.49861076 \\ -0.969243636 & 1.875967502 & 0.041555057 \\ 0.05563008 & -0.203976959 & 1.056971514 \end{bmatrix} \begin{bmatrix} X_{D65} \\ Y_{D65} \\ Z_{D65} \end{bmatrix}$$

- and backward linear transformation follows

$$\begin{bmatrix} X_{D65} \\ Y_{D65} \\ Z_{D65} \end{bmatrix} = \begin{bmatrix} 0.412390799 & 0.357584339 & 0.180480788 \\ 0.212639006 & 0.715168679 & 0.072192316 \\ 0.019330819 & 0.119194780 & 0.950532152 \end{bmatrix} \begin{bmatrix} R_{linear} \\ G_{linear} \\ B_{linear} \end{bmatrix}$$

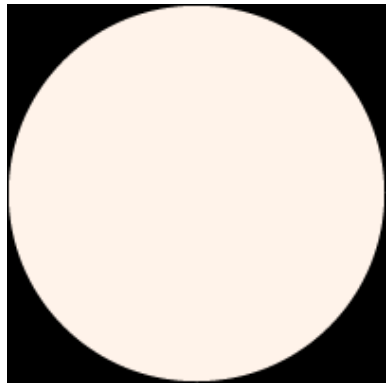
- And finally, gamma-compression (or expansion) takes place here

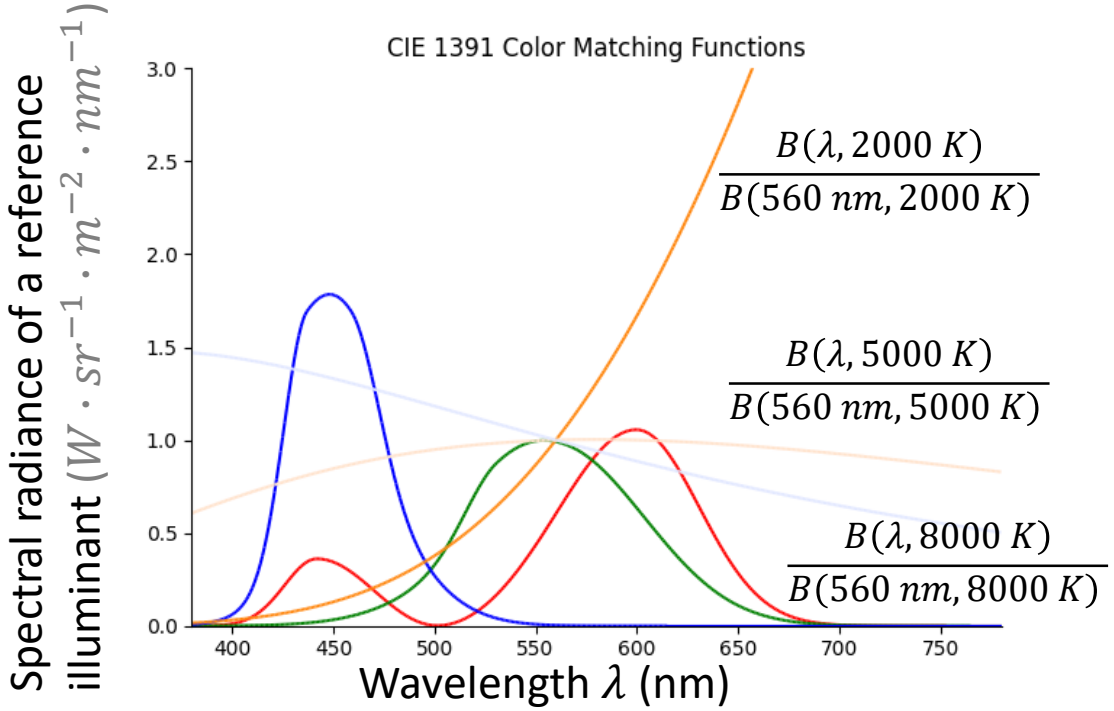
True color of the Sun

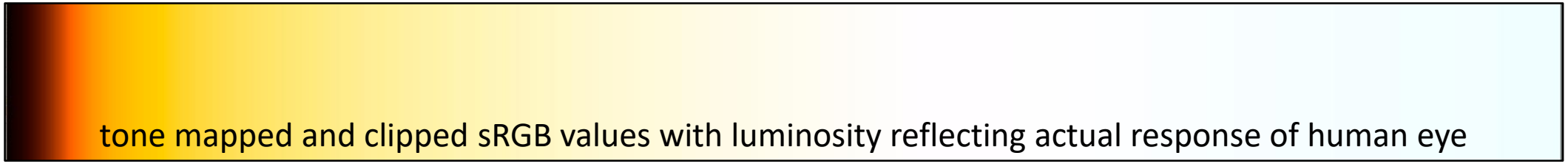
Experiment 1

- Blackbody radiation

$$B(\lambda, T) = \frac{2\hbar c^2}{\lambda^5} \frac{1}{e^{\frac{\hbar c}{\lambda k_B T}} - 1}$$







1000 K 4500 K 10000 K

Experiment 2

Chromaticity coordinates of D65 white point used by sRGB

$x\_d65 := 0.3127 \qquad y\_d65 := 0.3290 \qquad z\_d65 := 1 - x\_d65 - y\_d65 = 0.3583$

CIE XYZ coordinates

$Y\_d65 := 0.8$ reference display white point luminace 80 cd/m2

$$X\_d65 := x\_d65 \cdot \frac{Y\_d65}{y\_d65} = 0.760365$$

$$Z\_d65 := \frac{(1 - x\_d65 - y\_d65) \cdot Y\_d65}{y\_d65} = 0.871246$$

$$M := \begin{bmatrix} \frac{12831}{3959} & -\frac{329}{214} & -\frac{1974}{3959} \\ -\frac{851781}{878810} & \frac{1648619}{878810} & \frac{36519}{878810} \\ \frac{705}{12673} & -\frac{2585}{12673} & \frac{705}{667} \end{bmatrix} = \begin{bmatrix} 3.240969942 & -1.537383178 & -0.49861076 \\ -0.969243636 & 1.875967502 & 0.041555057 \\ 0.05563008 & -0.203976959 & 1.056971514 \end{bmatrix}$$

Linear sRGB values

$$\begin{bmatrix} R\_l \\ G\_l \\ B\_l \end{bmatrix} := M \cdot \begin{bmatrix} X\_d65 \\ Y\_d65 \\ Z\_d65 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.8 \\ 0.8 \end{bmatrix}$$

White points of standard illuminants

Name	CIE 1931 2°		CIE 1964 10°		CCT (K)	Hue	RGB	Note
	x_2	y_2	x_{10}	y_{10}				
A	0.44757	0.40745	0.45117	0.40594	2856			Incandescent / Tungsten
B	0.34842	0.35161	0.34980	0.35270	4874			{obsolete} Direct sunlight at noon
C	0.31006	0.31616	0.31039	0.31905	6774			{obsolete} Average / North sky Daylight
D50	0.34567	0.35850	0.34773	0.35952	5003			Horizon Light. ICC profile PCS
D55	0.33242	0.34743	0.33411	0.34877	5503			Mid-morning / Mid-afternoon Daylight
D65	0.31271	0.32902	0.31382	0.33100	6504			Noon Daylight: Television, sRGB color space
D75	0.29902	0.31485	0.29968	0.31740	7504			North sky Daylight
E	1/3	1/3	1/3	1/3	5454			Equal energy

Source: https://en.wikipedia.org/wiki/Standard_illuminant

Experiment 3

- Direct mixing of two gamma-compressed sRGB colors

- Mixing of two gamma-expanded sRGB colors

Code for Experiment 3

```
def _compress(u):
  if u <= 0:
    return 0
  if u >= 1:
    return 1
  if u <= 0.0031308:
    return 12.92 * u
  else:
    return (1.055 * u**(1 / 2.4) - 0.055)

def _expand(u):
  if u <= 0:
    return 0
  if u >= 1:
    return 1
  if u <= 0.04045:
    return u / 12.92
  else:
    return ( ( u + 0.055 ) / 1.055 )**2.4
```

```
def compress(c_in):
  c_out = []
  for i in range(3):
    c_out.append(_compress(c_in[i]))
  return tuple(c_out)

def expand(c_in):
  c_out = []
  for i in range(3):
    c_out.append(_expand(c_in[i]))
  return tuple(c_out)

def mix_linear(c0, c1, alpha):
  c_out = []
  for i in range(3):
    c_out.append(alpha * c0[i] + (1 - alpha) * c1[i])
  return tuple(c_out)

def mix_srgb(c0, c1, alpha):
  return compress(mix_linear(expand(c0), expand(c1), alpha))
```