

Introduction to Scientific Visualization

Scientific Visualization
Professor Eric Shaffer

A Visualization Lexicon

Many different activities use the terms “visual” or “visualization”

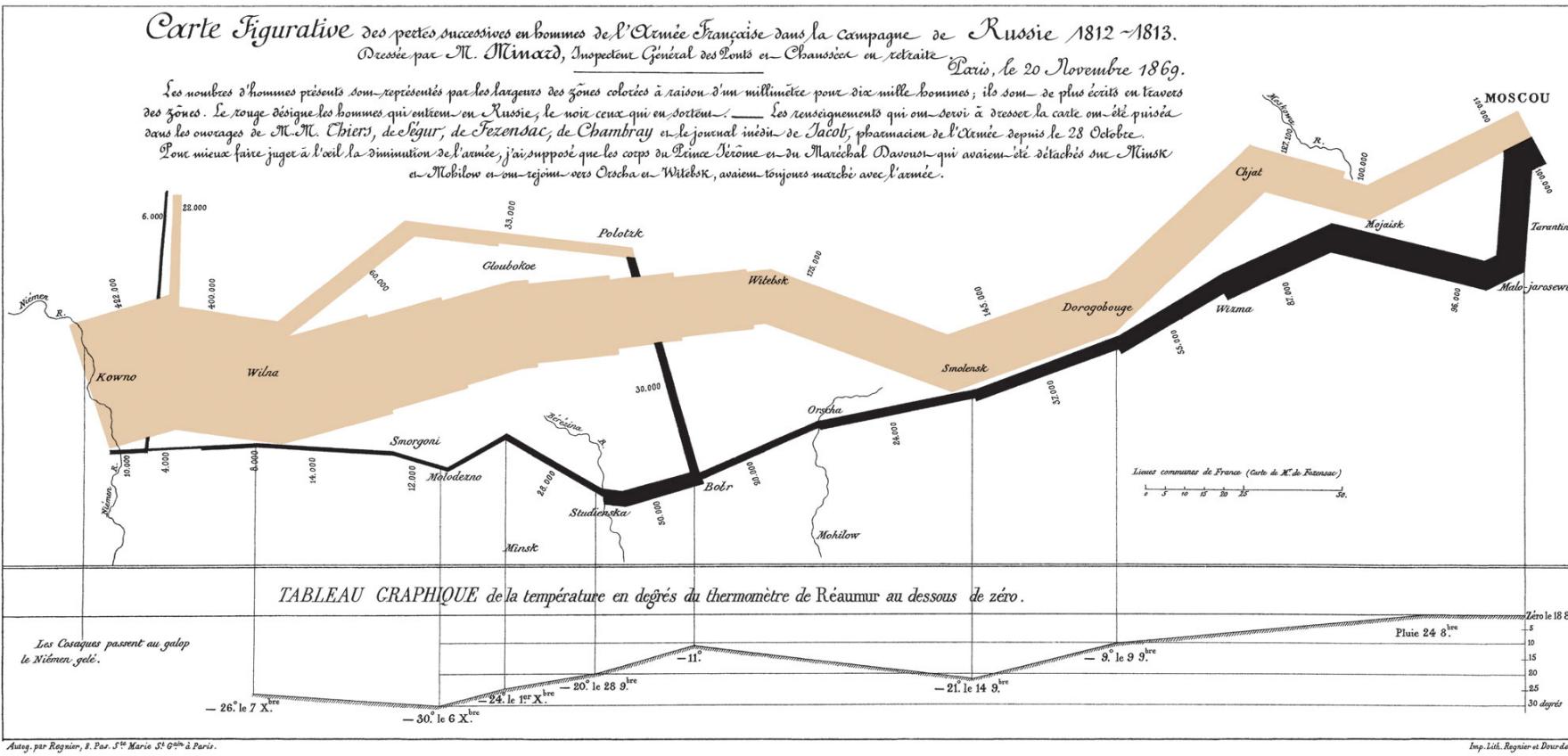
- Data Visualization
- Information Visualization
- Visual Analytics
- Scientific Visualization

What are the differences?

Data Visualization

Data visualization refers to the techniques used to communicate data or information by encoding it as visual objects

- https://en.wikipedia.org/wiki/Data_visualization

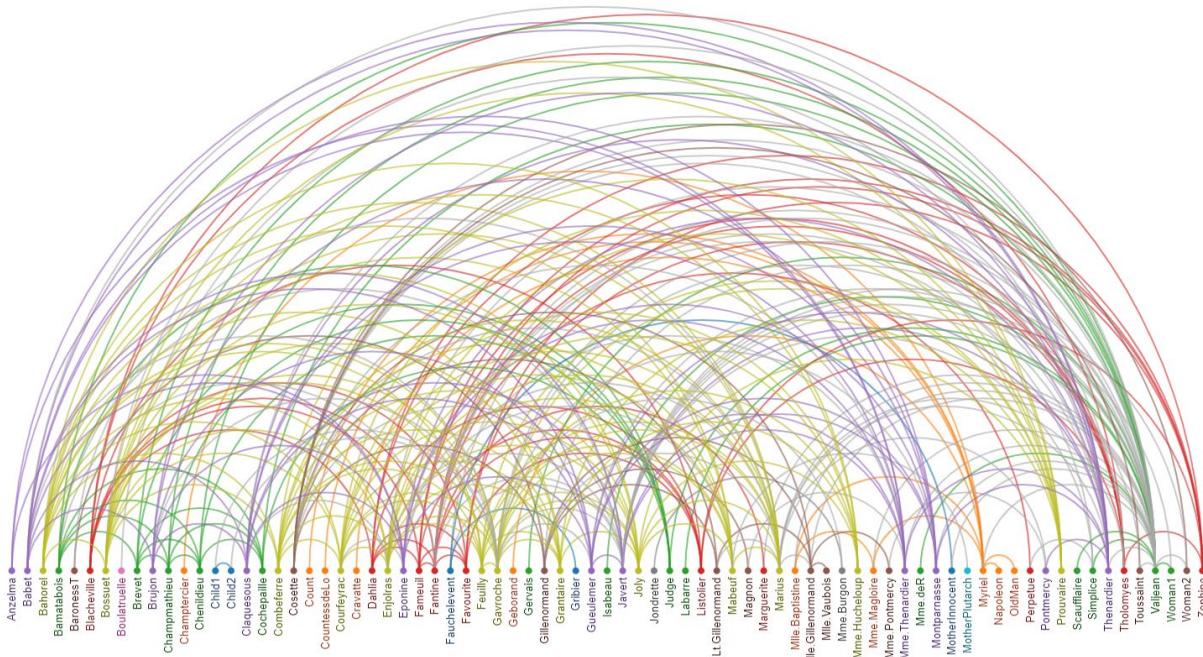


Charles Joseph
Minard's 1869
diagram of
Napoleonic
France's invasion
of Russia

Information Visualization

Information visualization is a process of transforming data and information that are not inherently spatial into a visual form, allowing the user to observe and understand the information.

- Nahum Gershon and Steve Eick, proceedings of the first IEEE Symposium on Information Visualization

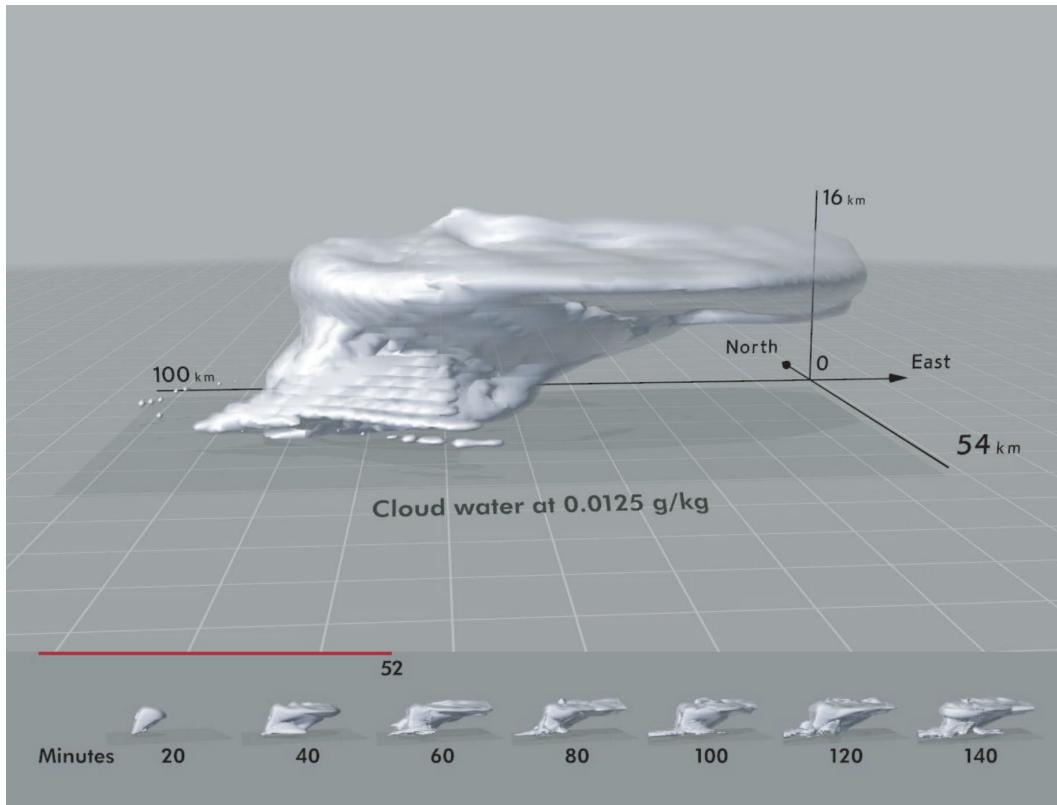




Scientific Visualization

...scientific visualization, which frequently focuses on spatial data generated by scientific processes..

- Nahum Gershon and Steve Eick, proceedings of the first IEEE Symposium on Information Visualization



Scientific Visualization versus Information Visualization

“the subfield names grew out of an accident of history and have some unfortunate connotations when juxtaposed: information visualization isn’t unscientific, and scientific visualization isn’t uninformative.”

- Professor Tamara Munzner

- Fields are more similar than different
- Sci Vis typically uses data from a physical or metric space
- Info Vis typically uses more abstract data with no natural location

Visual Analytics



“the science of analytical reasoning facilitated by interactive visual interfaces.”

James J. Thomas and Kristin A. Cook (Ed.) (2005). Illuminating the Path: The R&D Agenda for Visual Analytics National

- Visual Analytics focuses on how to create computational tools for data analysis that use visualization.
- Often the tools include other capabilities such as data mining to assist the analyst.



A Sampling of Topics



Terrain Visualization

Volume Visualization

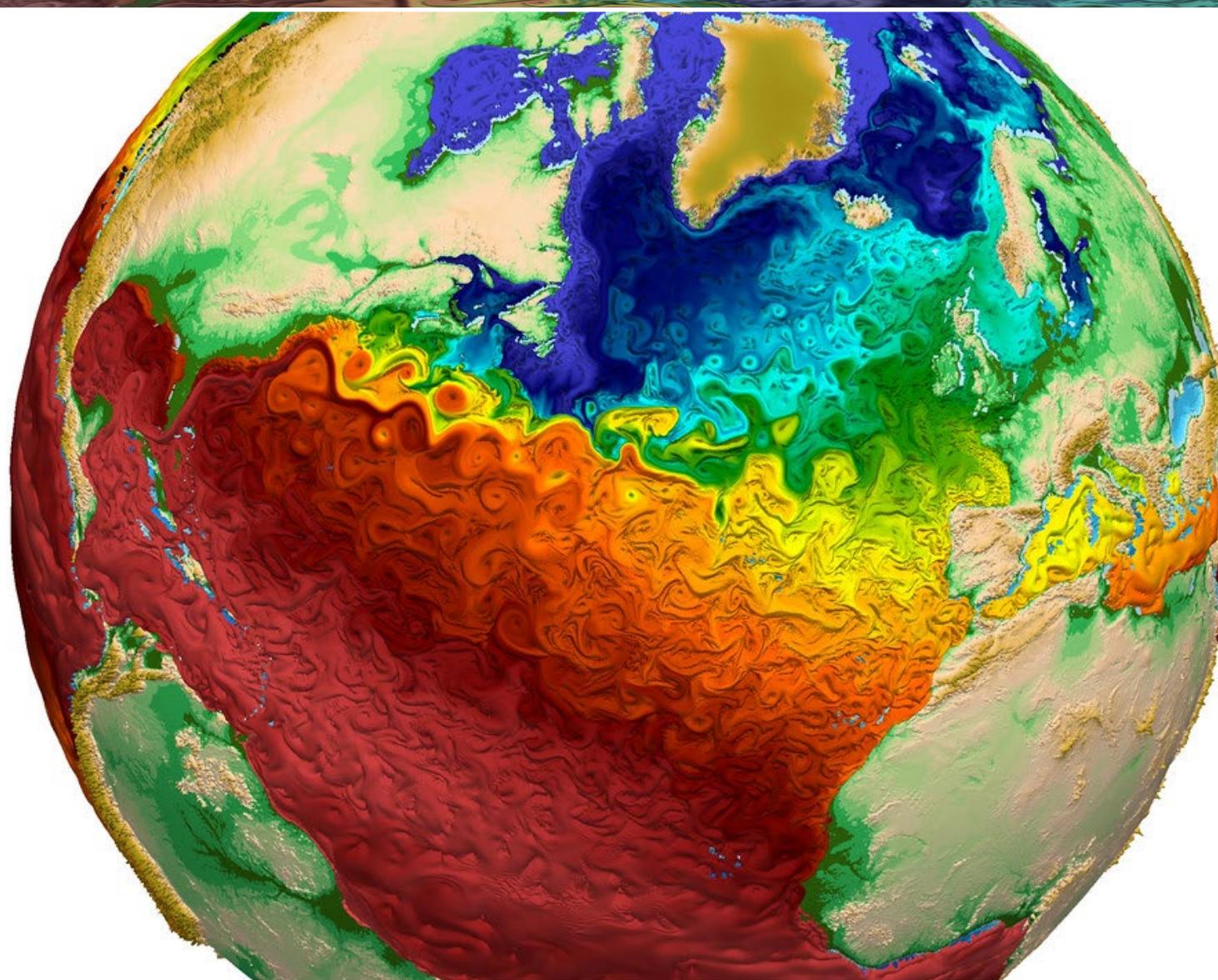
Isosurfaces

Flow Visualization

Tensor Field Visualization

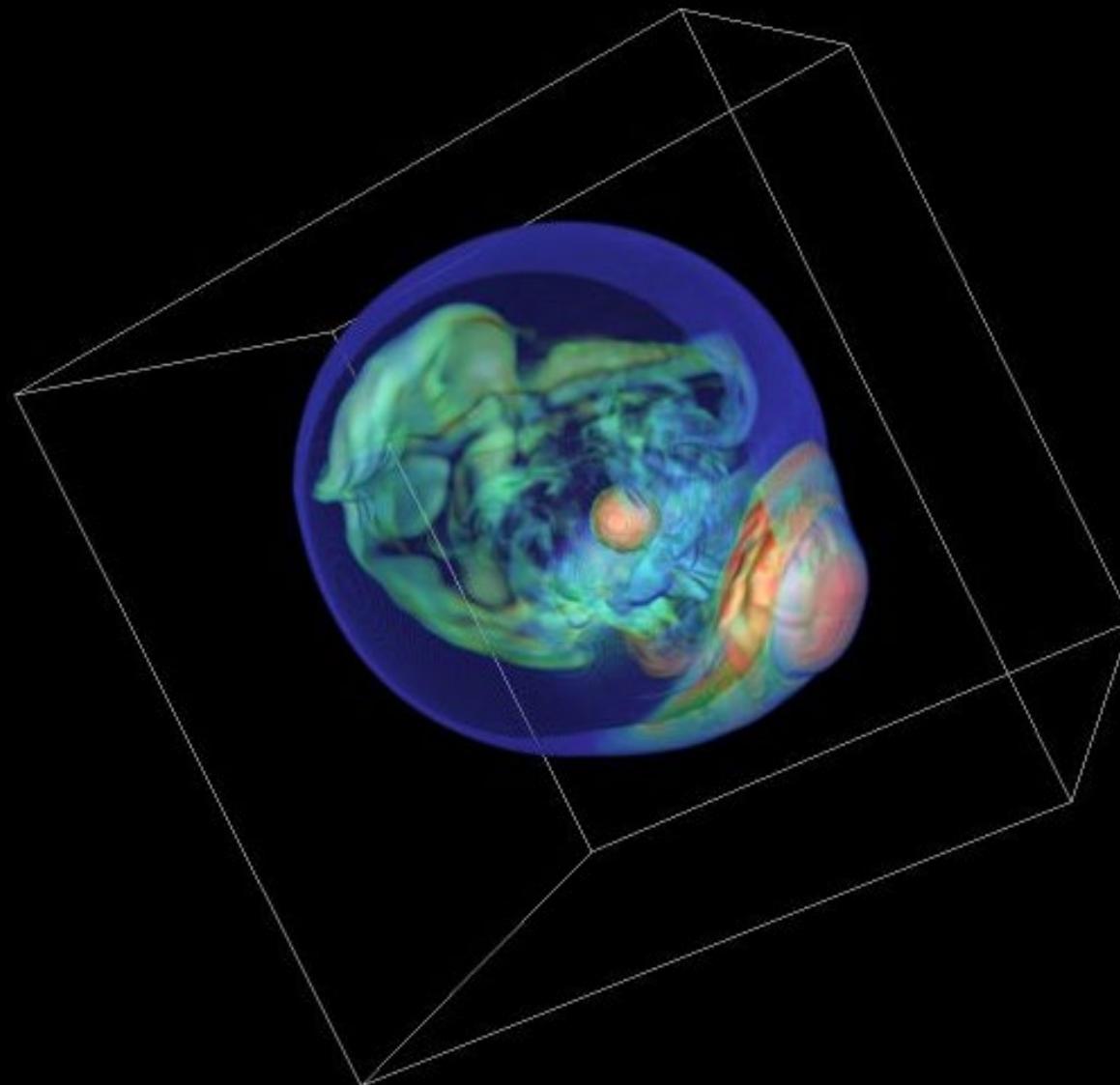
Bioinformatics Visualization

Terrain Visualization



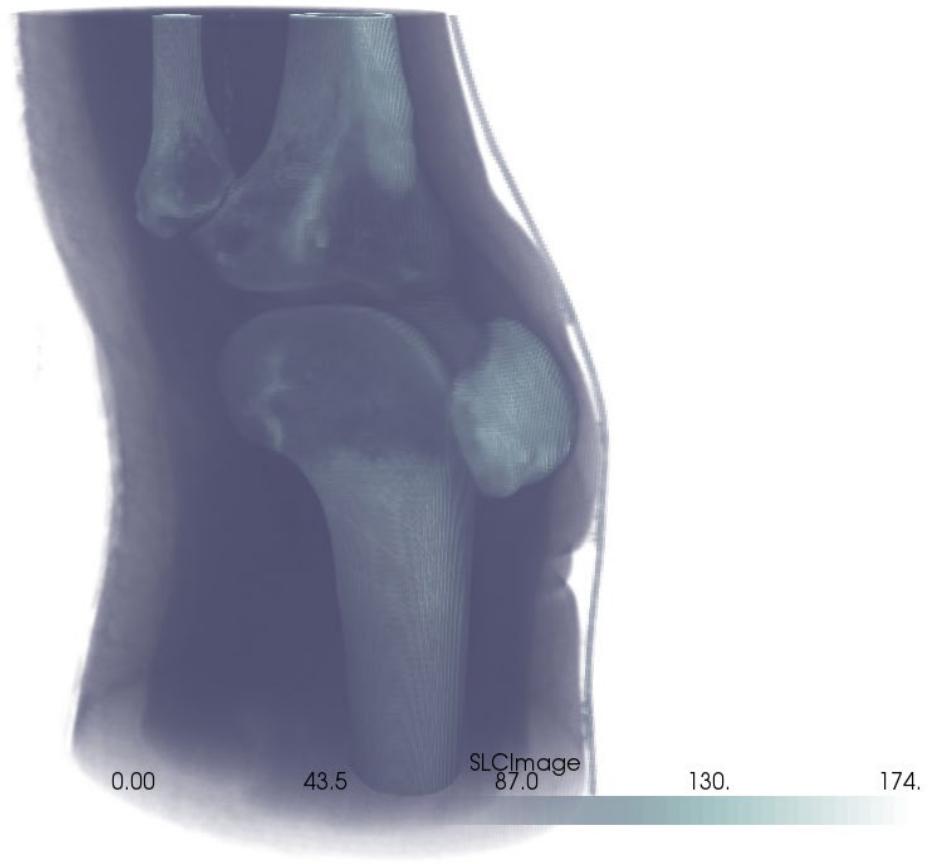
Visualization of
global surface
water
temperature
from Los
Alamos
National
Laboratory

Volume Visualization



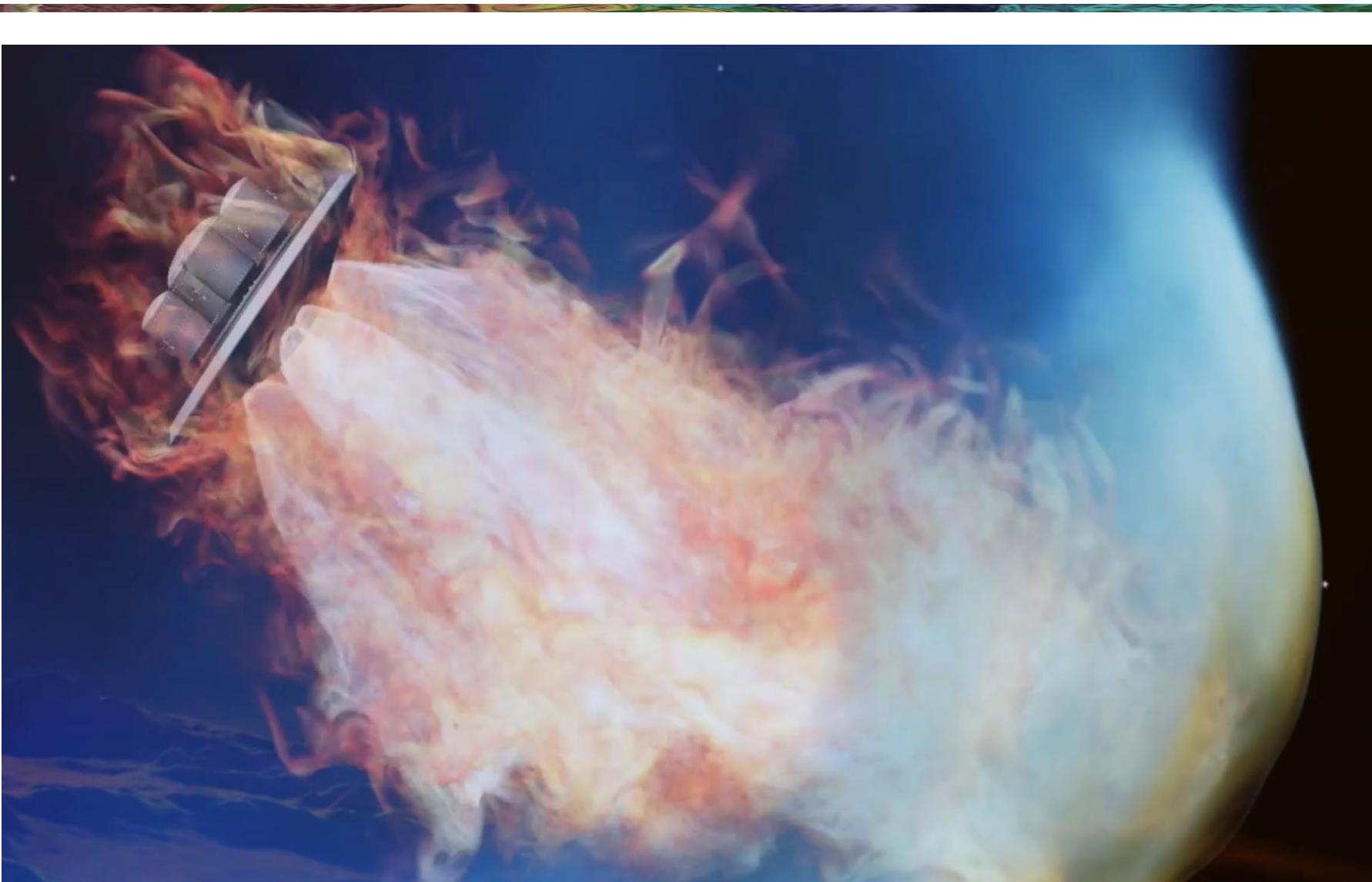
Simulation of a supernova

Isosurfaces



MRI data of a human knee

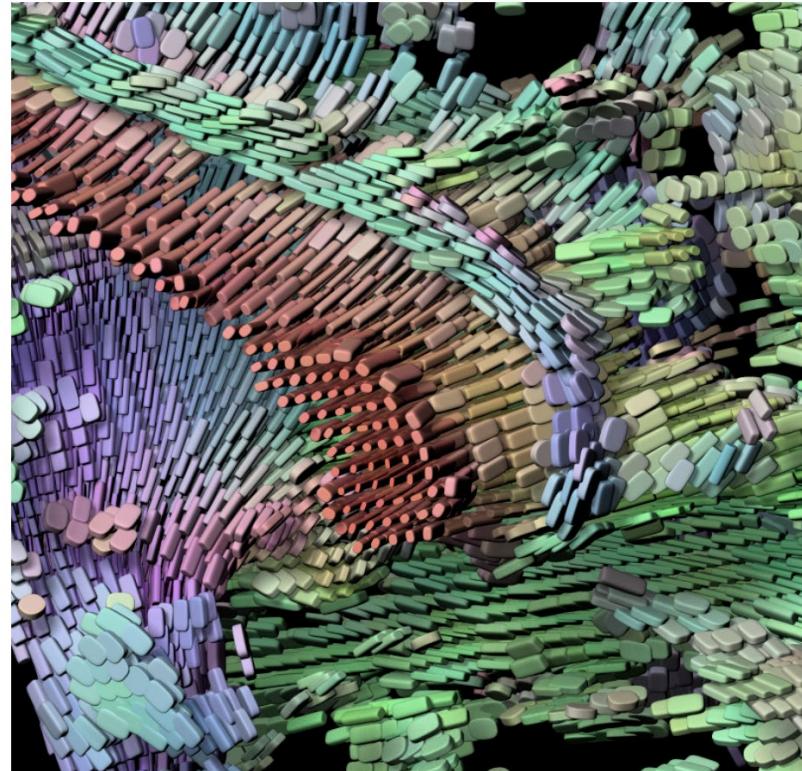
Flow Visualization



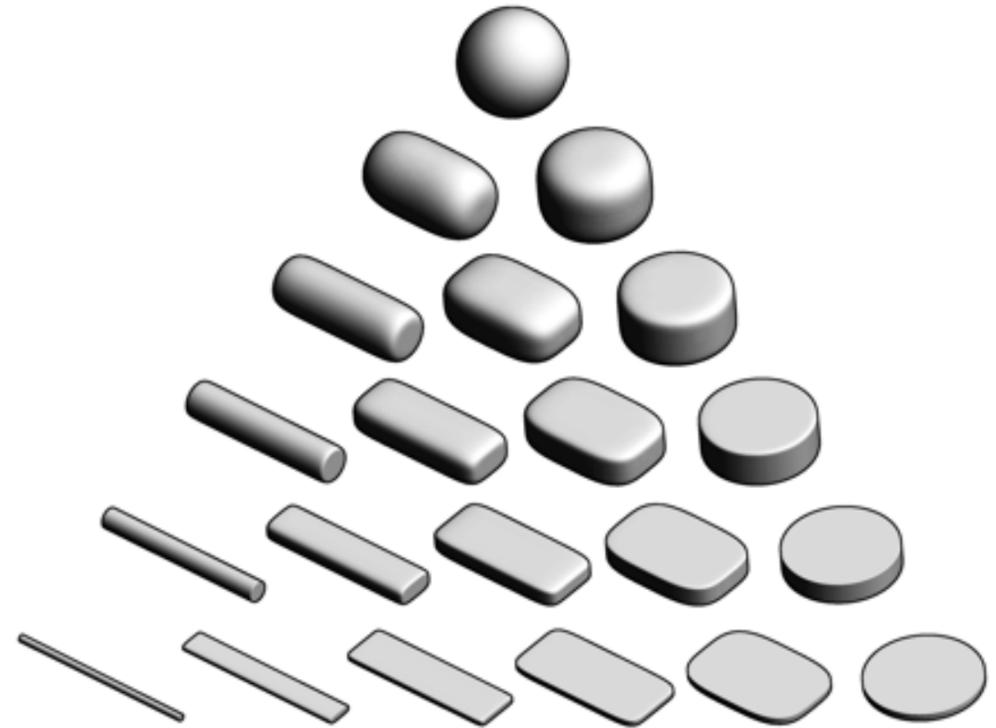
NASA Mars lander
visualization from
2019.

Domain consists
of 6-billion cells
making it the
world's largest
interactive fluid
visualization at
the time.

Tensor Field Visualization

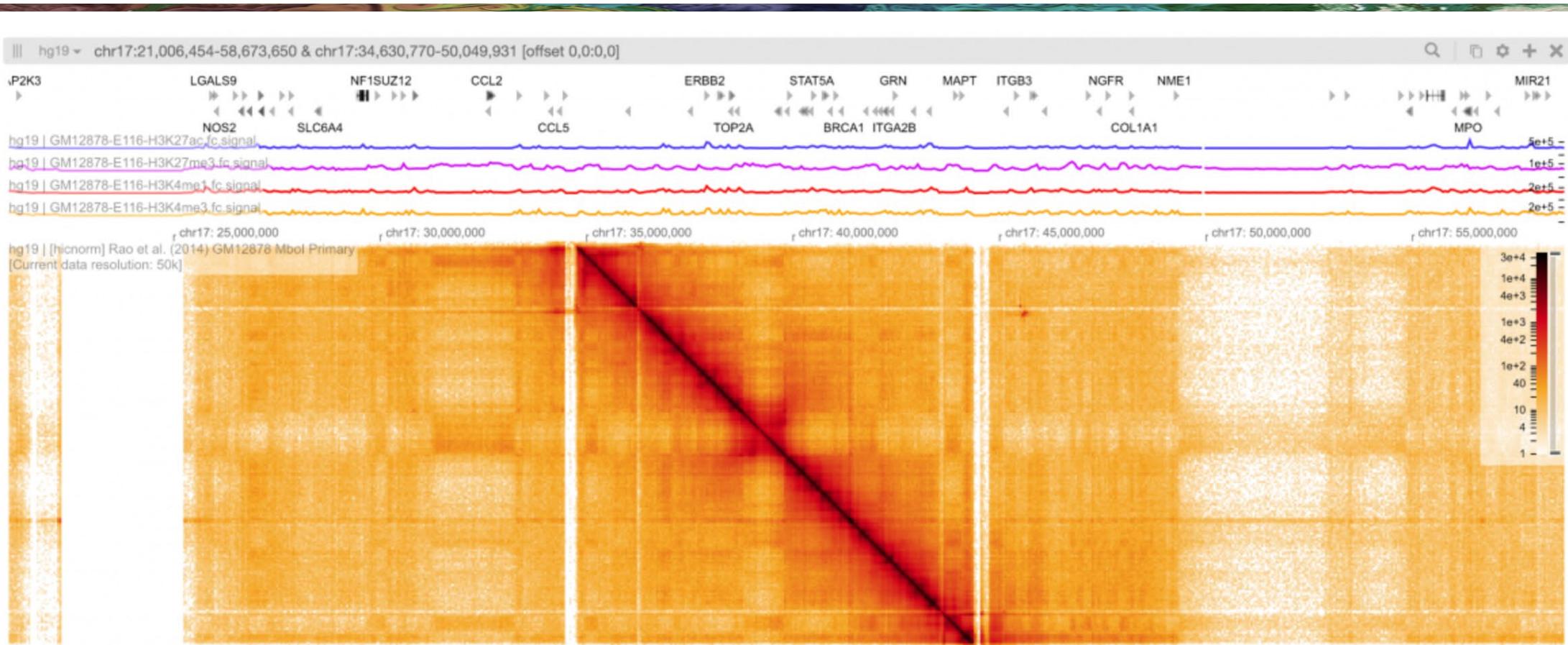


Tensor field generated from Diffusion-Tensor MRI

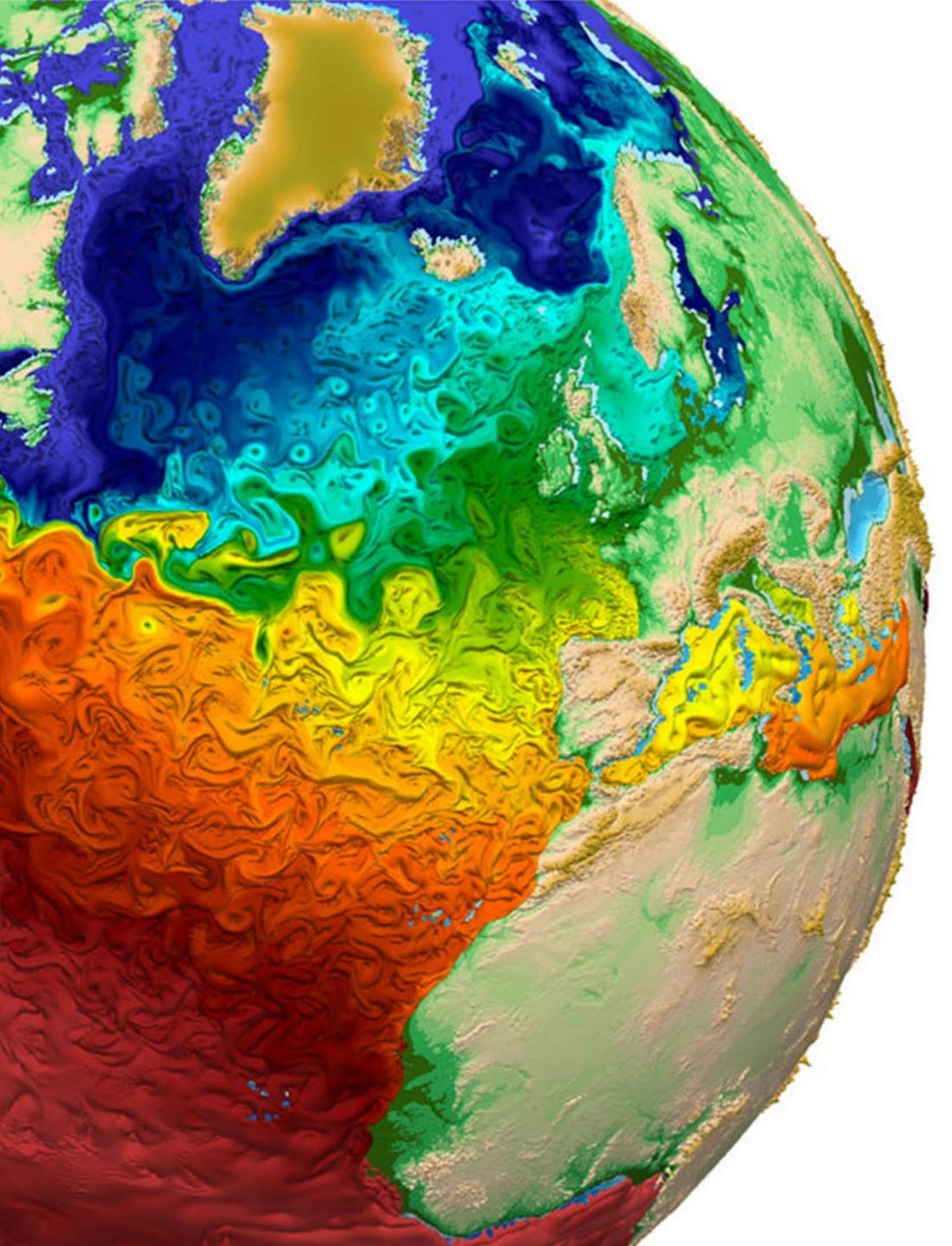


Tensor glyphs

Bioinformatics Visualization



HiGlass is a web-based tool for visually exploring and comparing 2D genomic contact matrices



Visualization and the COVID-19 Pandemic

Scientific Visualization
Professor Eric Shaffer

What is Visualization? Why do it?

Computer-based visualization systems provide visual representations of datasets designed to help people carry out tasks more effectively.

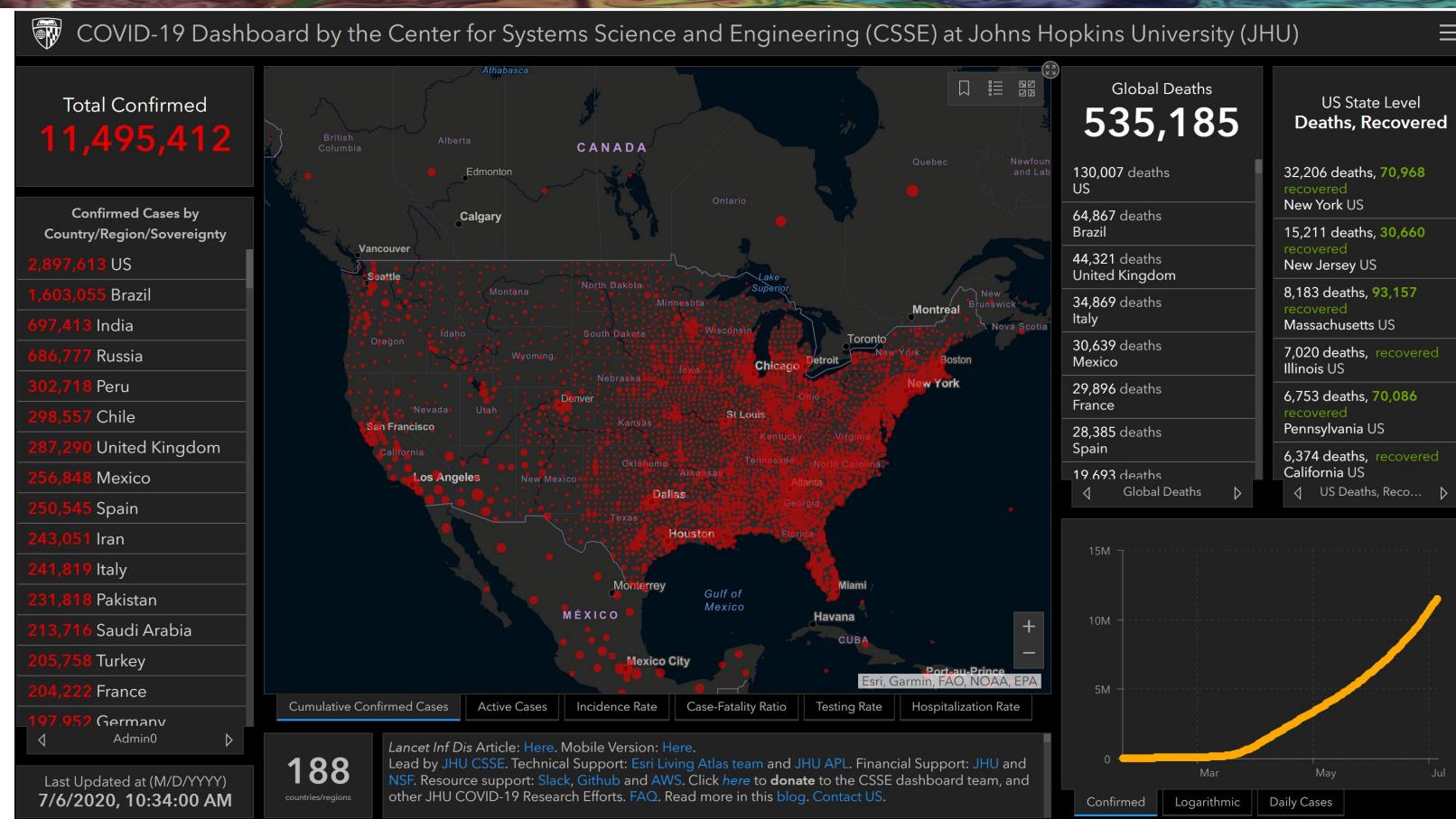
- Professor Tamara Munzner

What tasks is visualization used for?

A good question...we need to understand how this tool is best used.

Let's look at some examples and develop some ideas

Geography of COVID-19



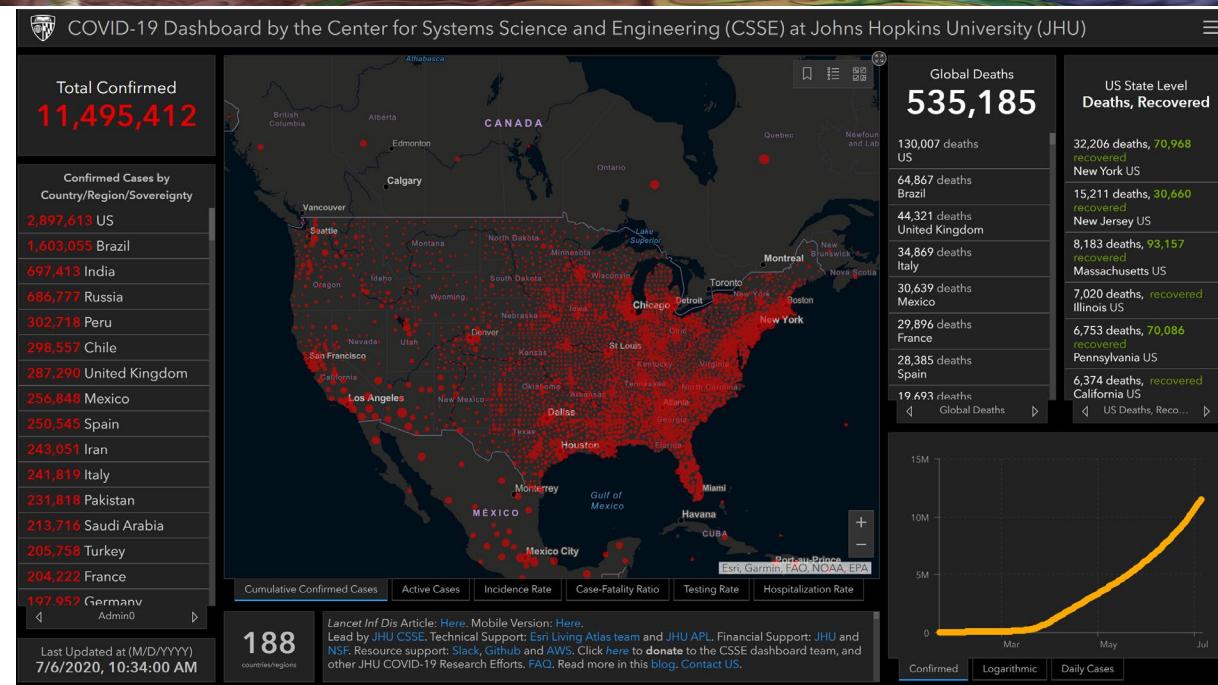
This type of display is called a dashboard

Shows case incidence in geographic regions

You can see change over time

COVID-19 Dashboard by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University (JHU)

Geography of COVID-19

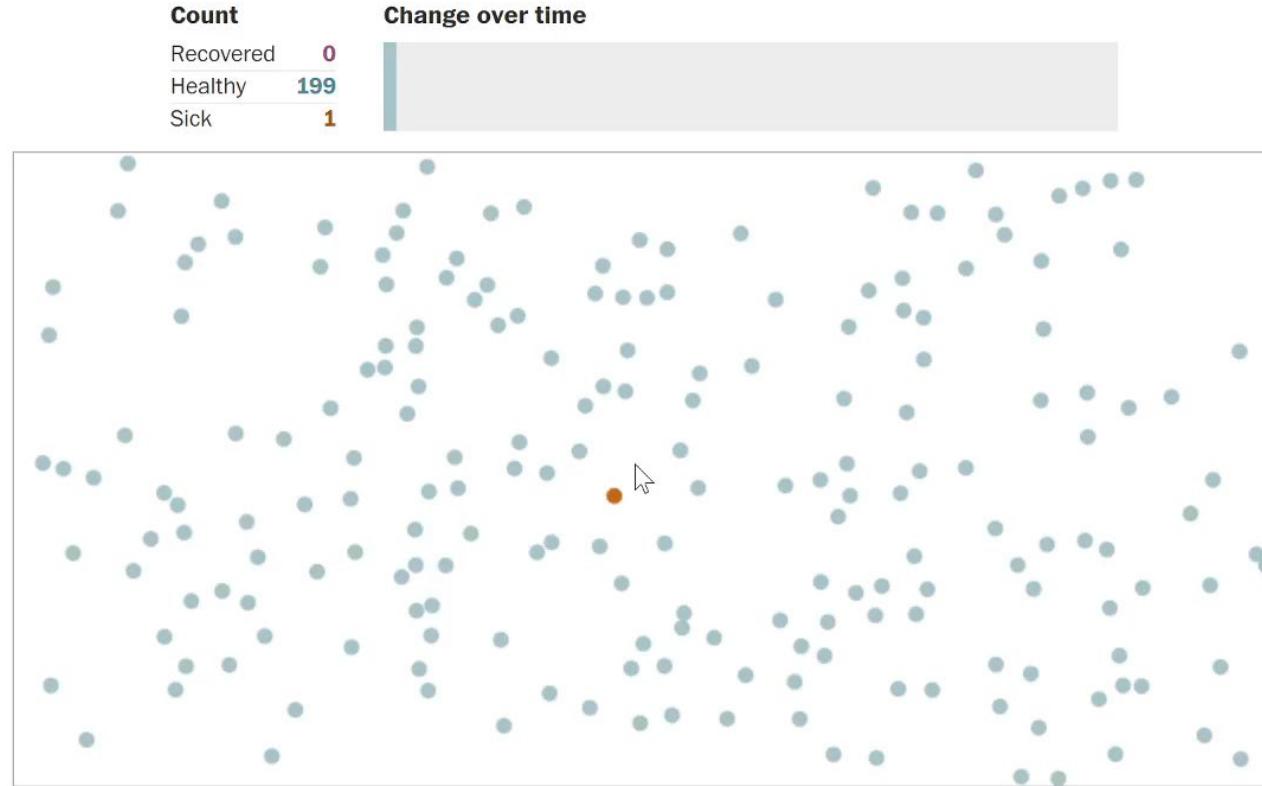


What is the task?

- Reveal patterns
- Explore data and make decisions
 - e.g. resource allocation to different geographic areas
 - e.g. adapting policies to different regions

COVID-19 Dashboard by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University (JHU)

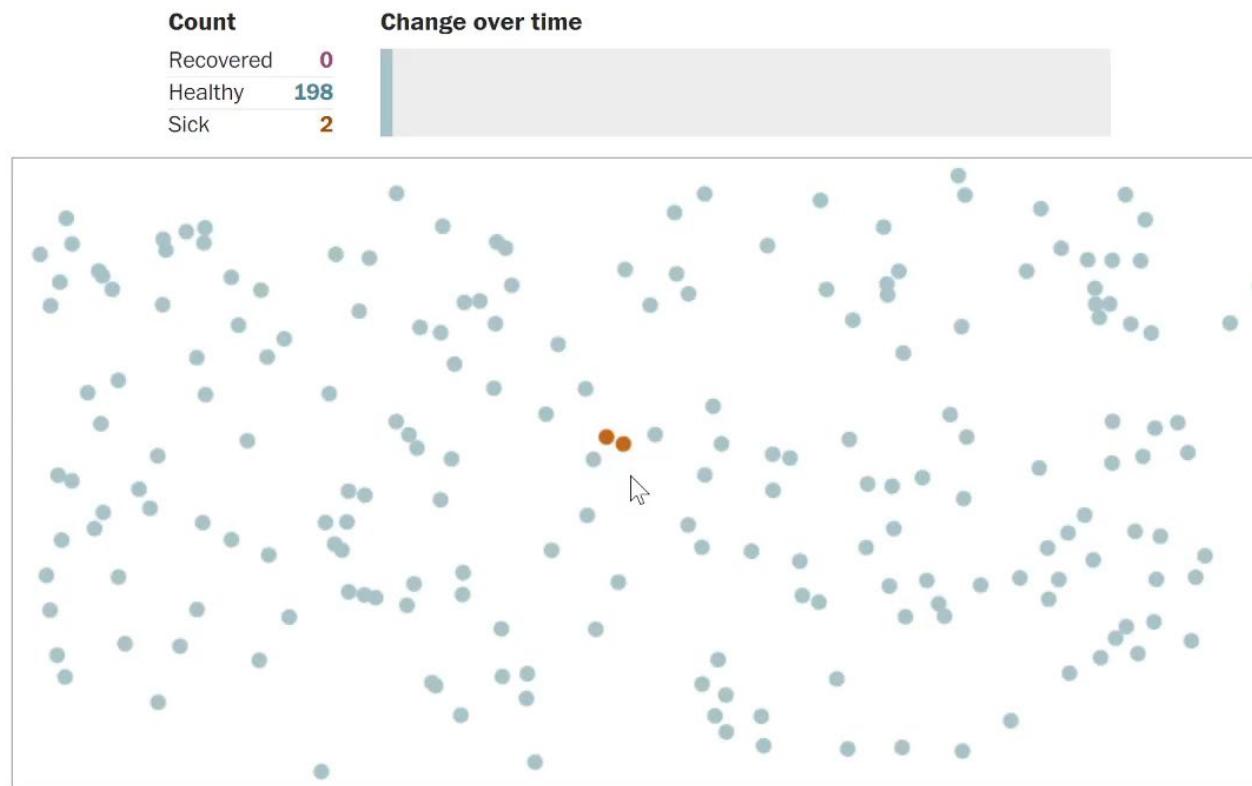
Understanding Exponential Growth



Simulation of disease spread

Why outbreaks like coronavirus spread exponentially, and how to “flatten the curve”
By Harry Stevens, Washington Post, March 14, 2020

Understanding Exponential Growth

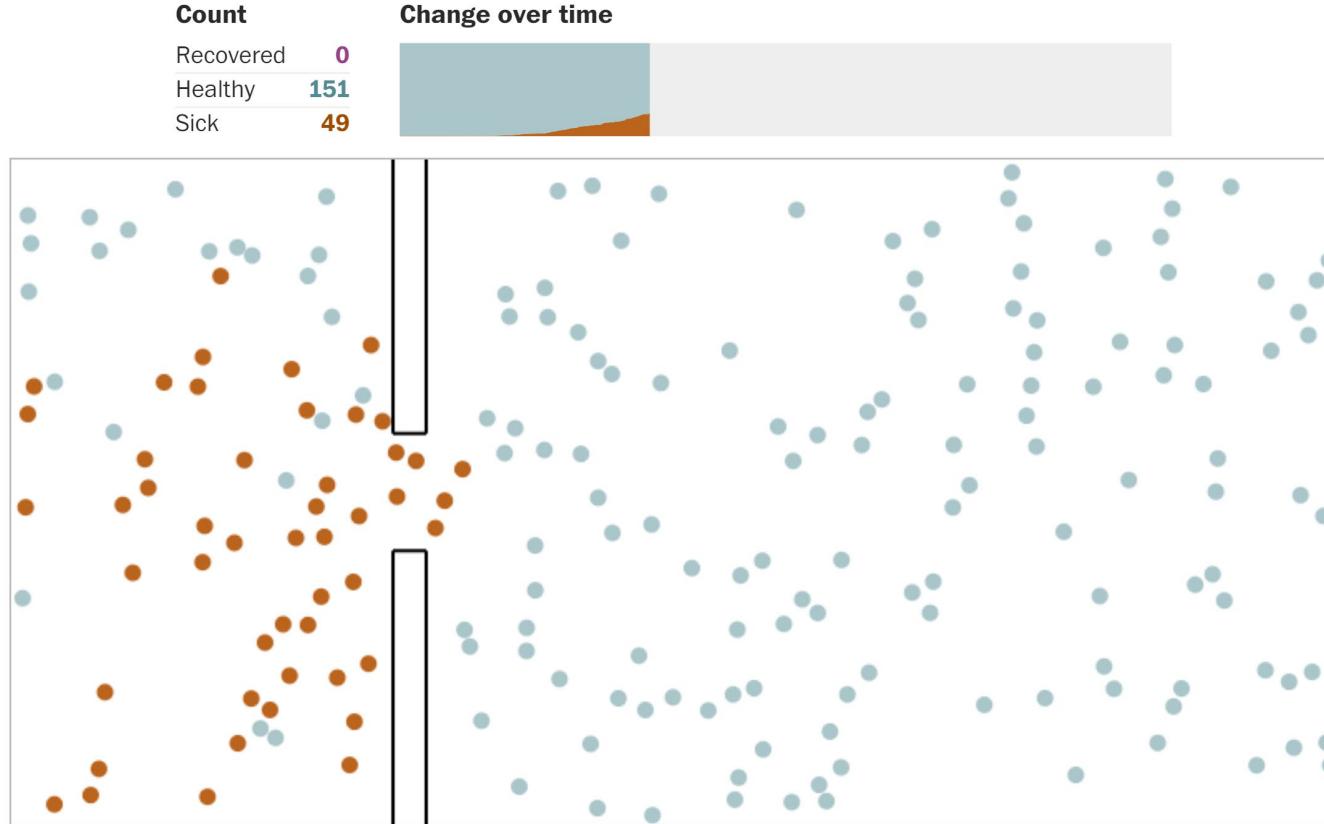


Simulation of disease spread

75% of population uses social distancing

Why outbreaks like coronavirus spread exponentially, and how to “flatten the curve”
By Harry Stevens, Washington Post, March 14, 2020

Understanding Exponential Growth



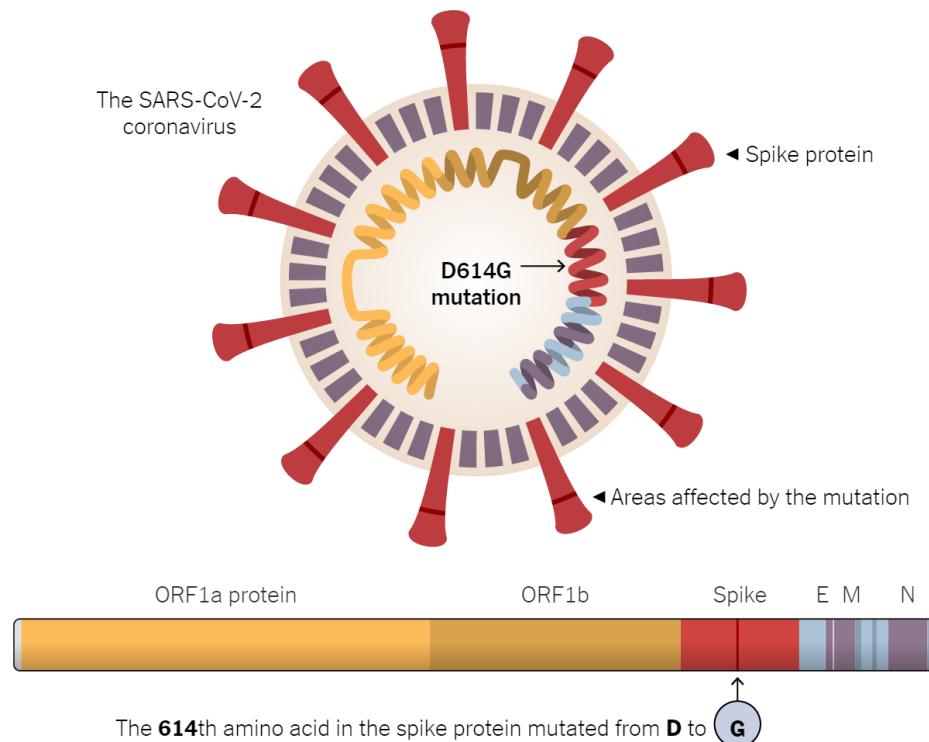
- These visualizations can be used to:
- to explore data and answer questions
 - to communicate ideas
 - generate hypotheses
 - to persuade or inspire

Why outbreaks like coronavirus spread exponentially, and how to “flatten the curve”
By Harry Stevens, Washington Post, March 14, 2020

COVID-19 Protein Structure

The D614G Mutation

A tiny mutation in the coronavirus genome may stabilize the spike proteins that protrude from the virus and allow it to infect more cells — at least in laboratory experiments.



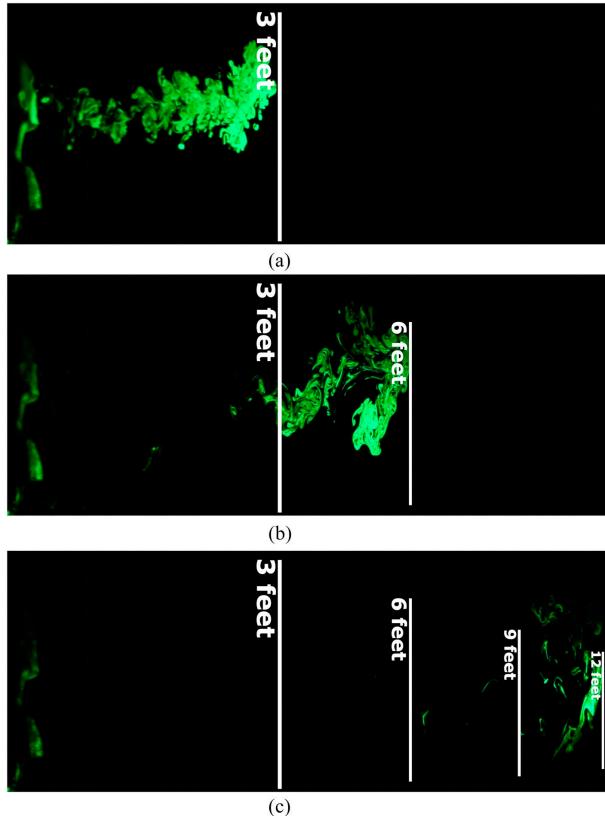
By Jonathan Corum | Source: Lizhou Zhang et al., Scripps Research

The D614G mutation in the SARS-CoV-2 spike protein reduces S1 shedding and increases infectivity

Lizhou Zhang, Cody B Jackson, Huihui Mou, Amrita Ojha, Erumbi S Rangarajan, Tina Izard, Michael Farzan, Hyeryun Choe1

Visualization used to communicate ideas

Mask Usage and Fluid Dynamics

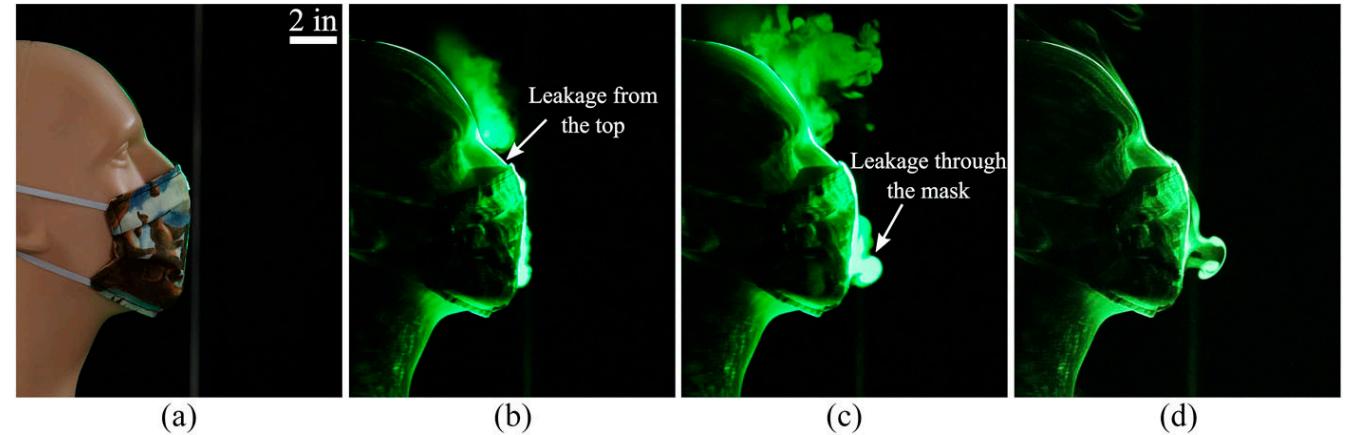


No mask

Visualizing the effectiveness of face masks in obstructing respiratory jets

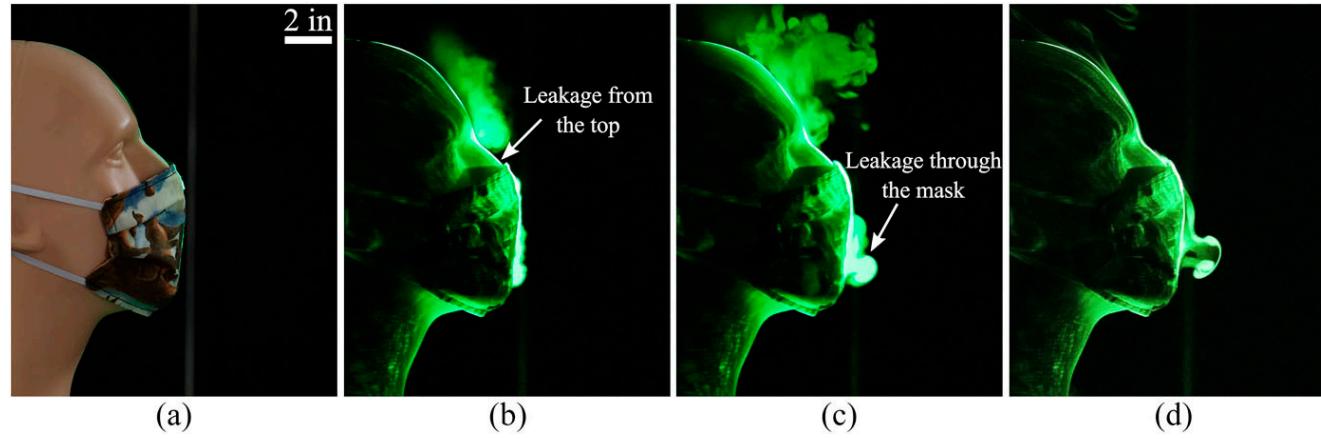
Siddhartha Verma, Manhar Dhanak, and John Frankenfield, Physics of Fluids 32:6

High speed photography of a physical experiment



A homemade face mask stitched using two-layers of cotton quilting fabric. Images taken at (b) 0.2 s, (c) 0.47 s, and (d) 1.68 s after the initiation of the emulated cough.

Mask Usage and Fluid Dynamics



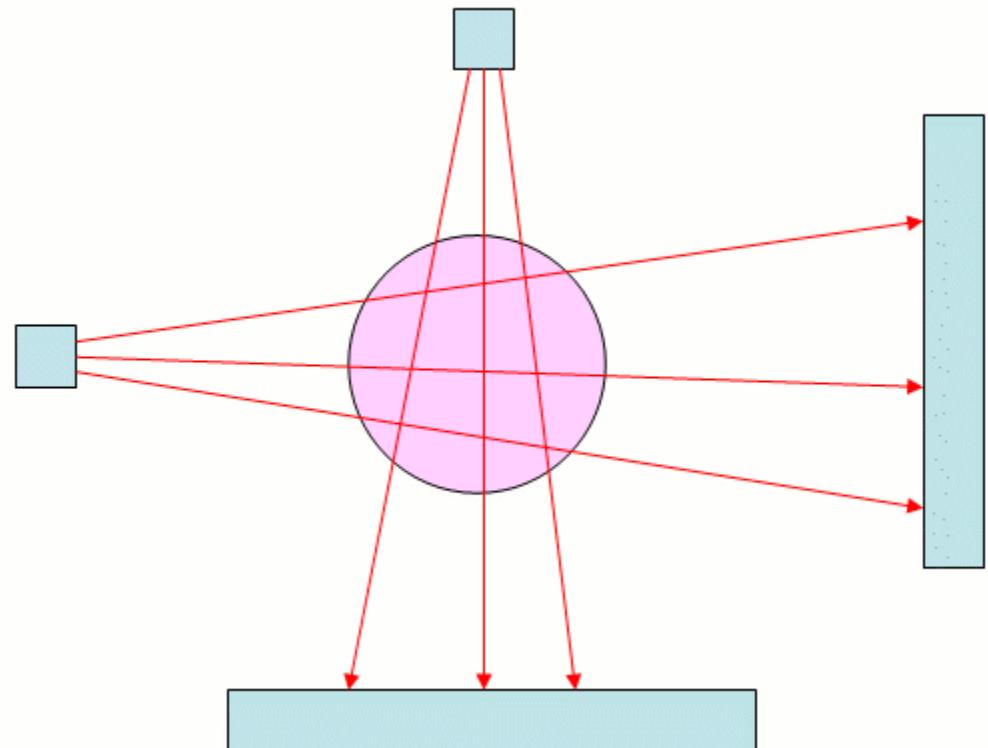
Visualization used to answer questions...which if any masks effectively contain respiratory jets?

Visualization used to communicate and persuade... homemade masks can significantly reduce the spread of a respiratory jet.

Visualizing the effectiveness of face masks in obstructing respiratory jets

Siddhartha Verma, Manhar Dhanak, and John Frankenfield, Physics of Fluids 32:6

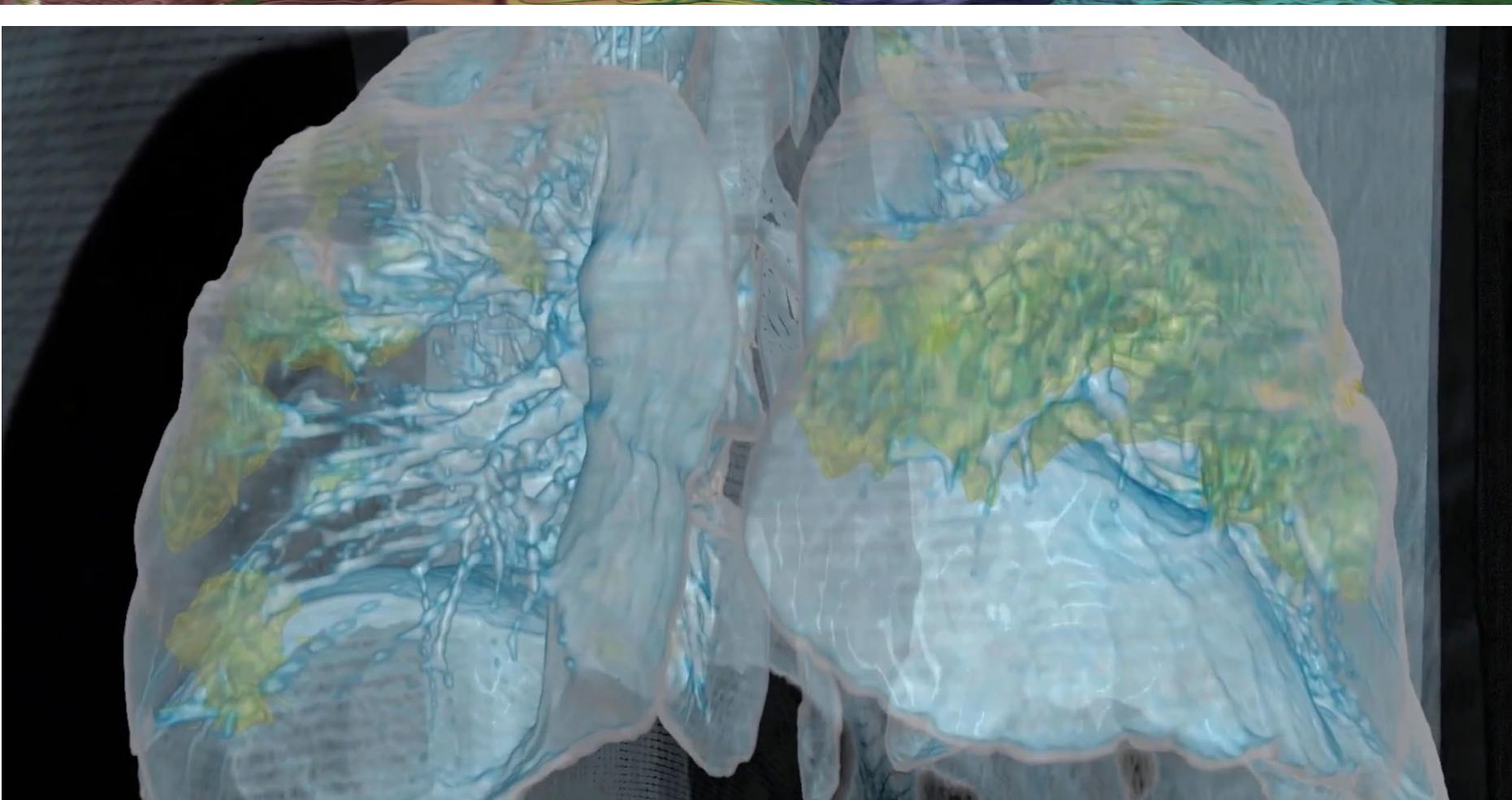
CT Scans



A computerized tomography (**CT**) **scan** uses X-rays to generate a series of 2D images that record the observed tissue density.

These images are then combined to create a 3D data set detailing the observed internal structures.

CT Scans



Used to generate hypotheses
...how does Covid-19 affect
the respiratory system.

Used to answer
questions...to develop a
diagnosis.

Visualization: Purpose

Some tasks for which visualization is used:

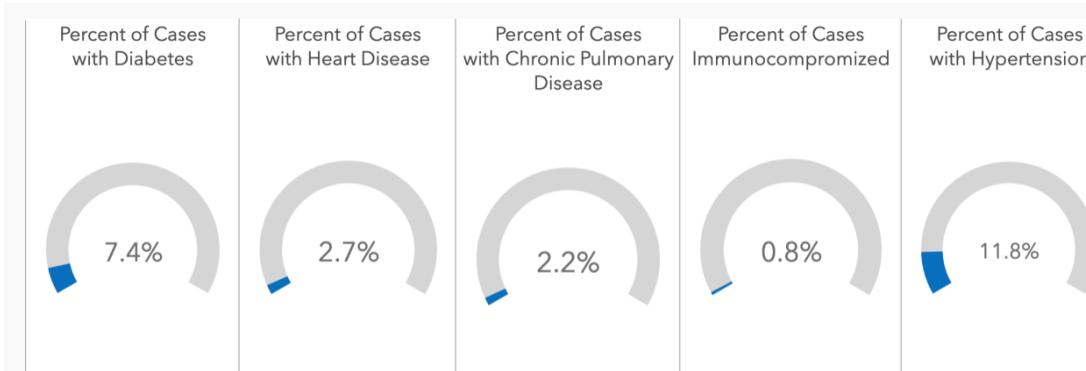
- to explore data and answer questions
 - to communicate ideas
 - find/reveal patterns and generate hypotheses
 - to persuade or inspire
- ...surely other reasons as well

The ubiquity of visualizations during the Covid-19 pandemic shows their value.

Misleading Visualizations

Visualization is notorious in its ability to mislead

- Sometimes intentionally...a sub-category of the “persuade” task



The graphic above, [from this page](#), shows the rate of different preexisting health conditions in patients confirmed to have Covid-19. Because all of these percentages are so low—and they’re depicted on a scale that goes up to 100—it seems like having another condition like hypertension isn’t a big deal.

Using percentages and unordered arcs gives the impression that few people are at risk...may not be true.

Images impart a great deal of information

Great care is required to ensure the information conveyed is accurate

Insight and Numbers and Pictures

“The purpose of computation is insight, not numbers.”

- Professor R.W. Hamming

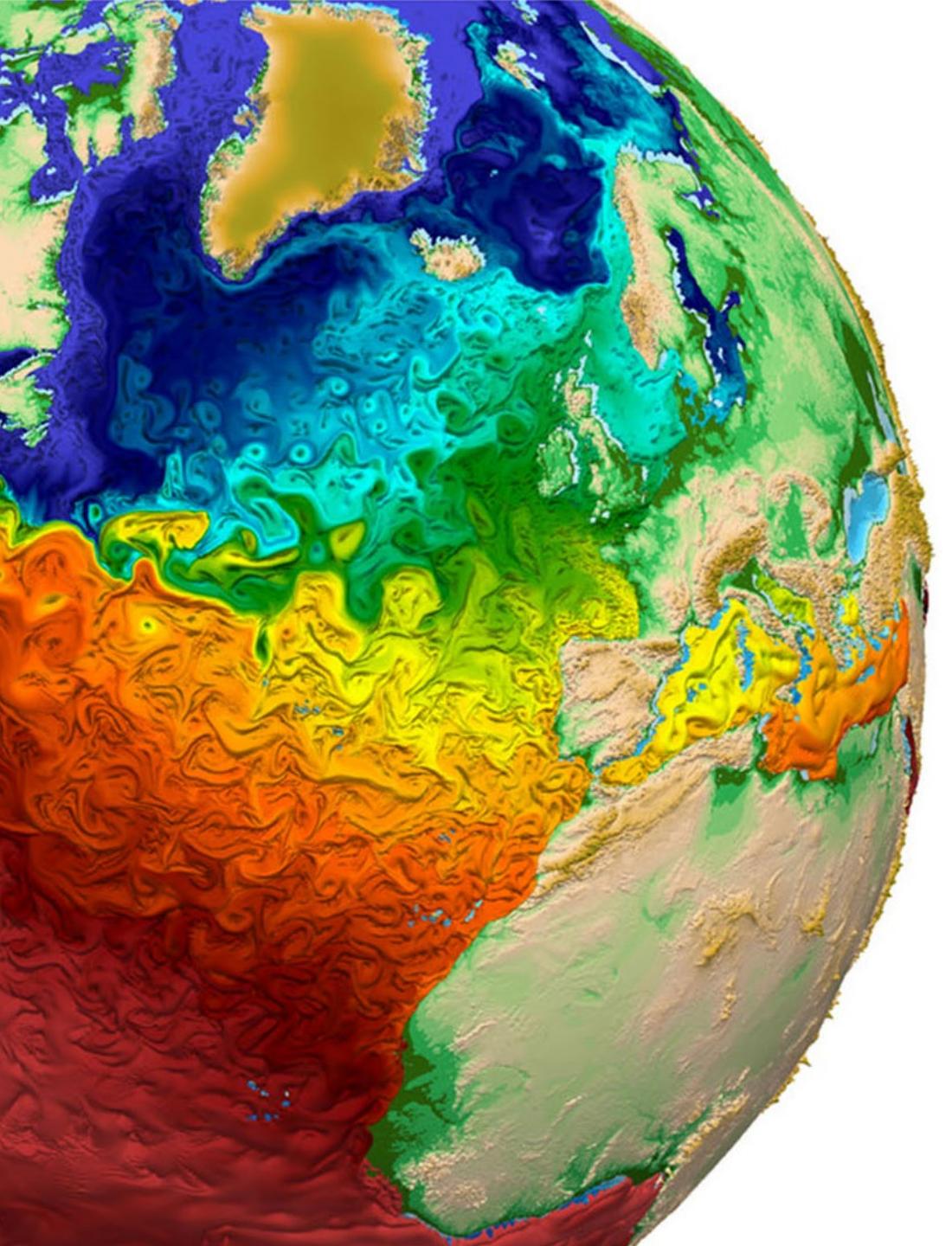
Hamming received his Ph.D. from the University of Illinois at Urbana-Champaign in 1942

“The purpose of visualization is insight, not pictures.”

- Professor Tamara Munzner

Corollary: Do not use visualization to accomplish a task better done without it.

For example...finding if the number 519 occurs in a list of numbers



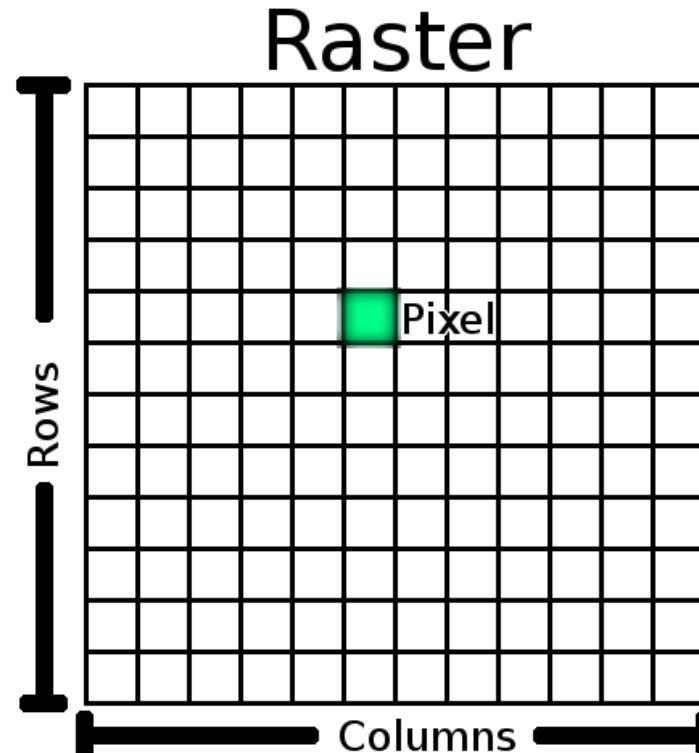
Digital Images

Display-Derived Color Spaces

Scientific Visualization
Professor Eric Shaffer

Digital Images

- Creating a digital image is the fundamental task in visualization
- Most modern digital displays rely on a *raster of pixels*



A *raster* is grid of addressable image elements called pixels

A *pixel* is the smallest controllable element of a digital image

"pixel" is short for
"picture element"

Digital Display Technology

Most modern digital displays are either LCD or OLED displays

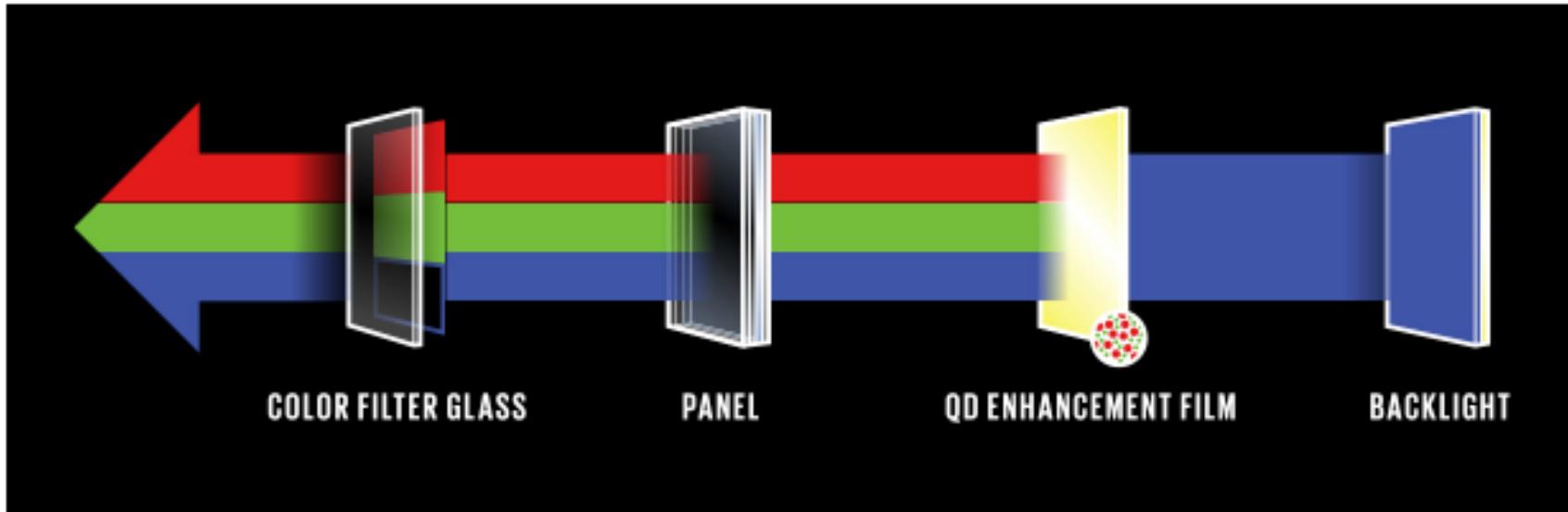
There are a lot of technologies and variations...we won't go in depth

First: what are those acronyms?

LCD: Liquid Crystal Display, transmissive using a light-emitting diode (LED) backlight

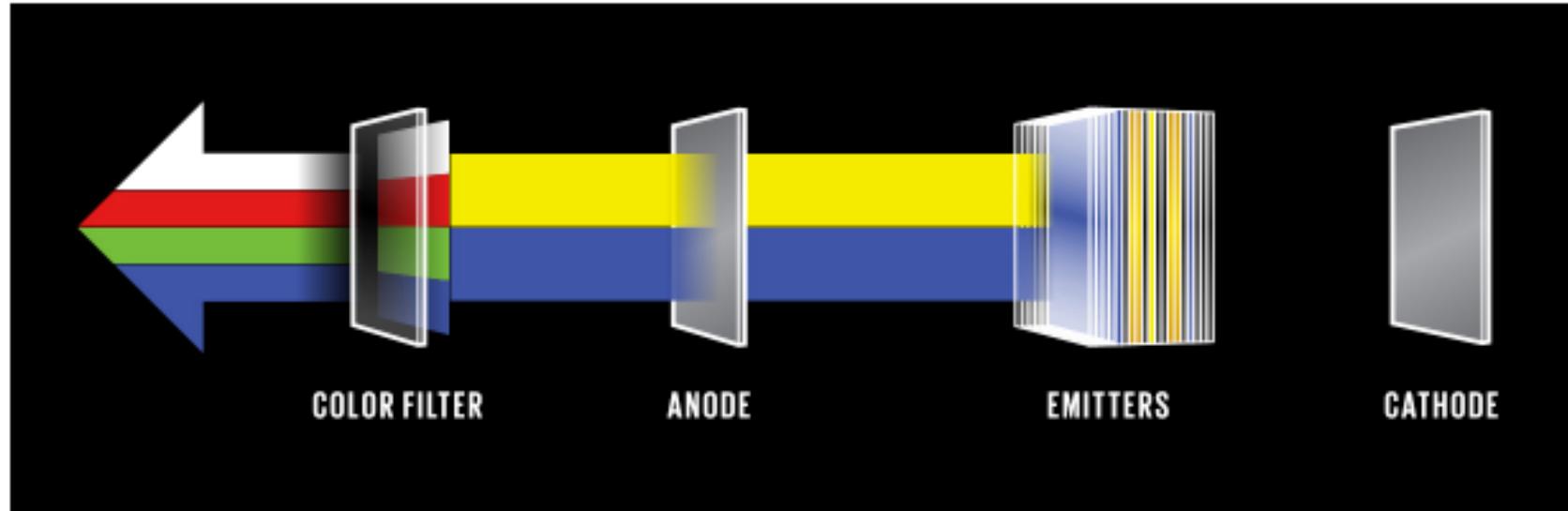
OLED: Organic Light Emitting Diode, uses emissive organic film to generate light

LCD-LED Displays



- A backing array of LEDs generates blue light
- Light is transmitted through a photo-emissive film to generate desired wavelengths of light
- Here, we see that each pixel can be thought of as having Red, Green, and Blue subpixels

OLED Displays



- Uses mix of blue and orange-yellow OLED emitters to create white light
- Light passes through filters to create red, green and blue sub-pixels
- A fourth subpixel lets white light through

OLED and LCD-LED Comparison

OLED Advantages

- Deep black levels
- Excellent viewing angle
- Fast refresh
- Can potentially be manufactured on flexible substrates

LCD-LED Advantages

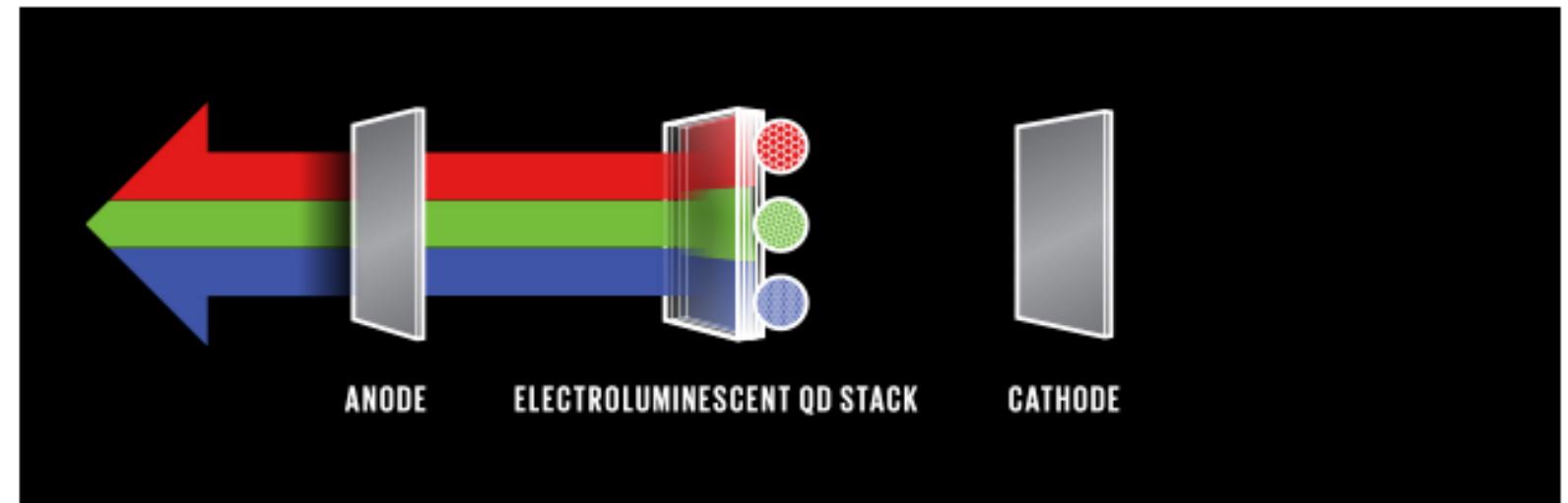
- Cheaper!
- More energy efficient



The Future?

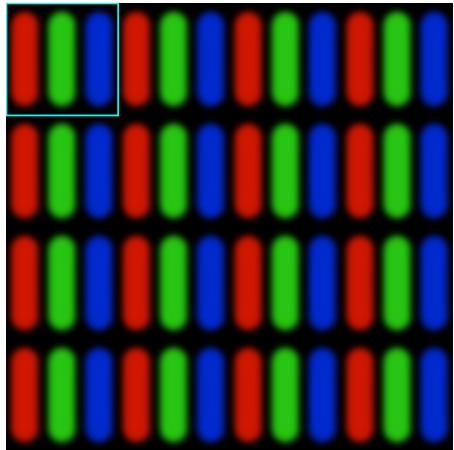
Electro-Emissive Quantum Dot Display

- No viewing angle issue
- Perfect black level
- Low cost
- Flexible substrate
- Fast refresh rate

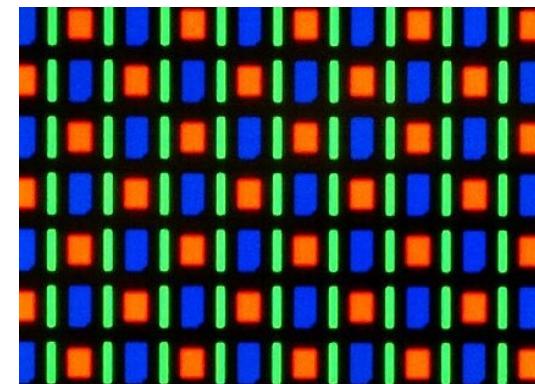


Subpixels

We can generalize that each pixel consists of red, green, and blue subpixels



In practice, different subpixel geometries are used by different displays
PenTile displays mimic the sensitivity of the human eye



"PenTile was invented by Candice H. Brown Elliott, for which she was awarded the Society for Information Display's Otto Schade Prize in 2014."
- Wikipedia

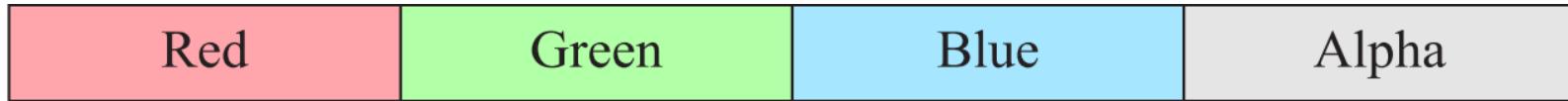


RGB Color Space

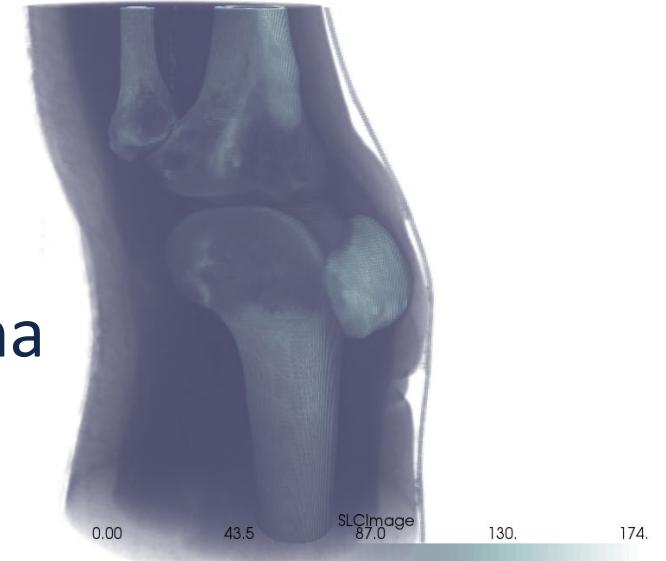


- Emissive displays typically use an RGB space to specify colors
 - R, G, and B are called ***primaries***
 - Each corresponds to a specific wavelength of light
- This means a pixel color is usually specified by a tuple of 3 numbers (R, G, B)
- Each of the R, G and B values are referred to as a ***color channel***
- The value of a channel is referred as the ***intensity***
- Typically, each channel value will be a floating point number in [0.0, 1.0]
 - 0.0 means no light
 - 1.0 means full intensity

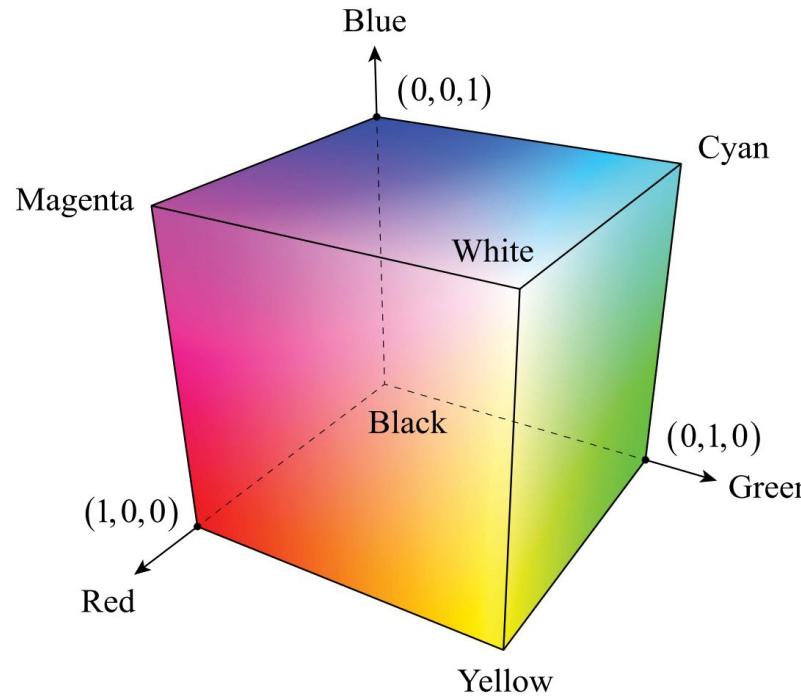
Alpha Channel



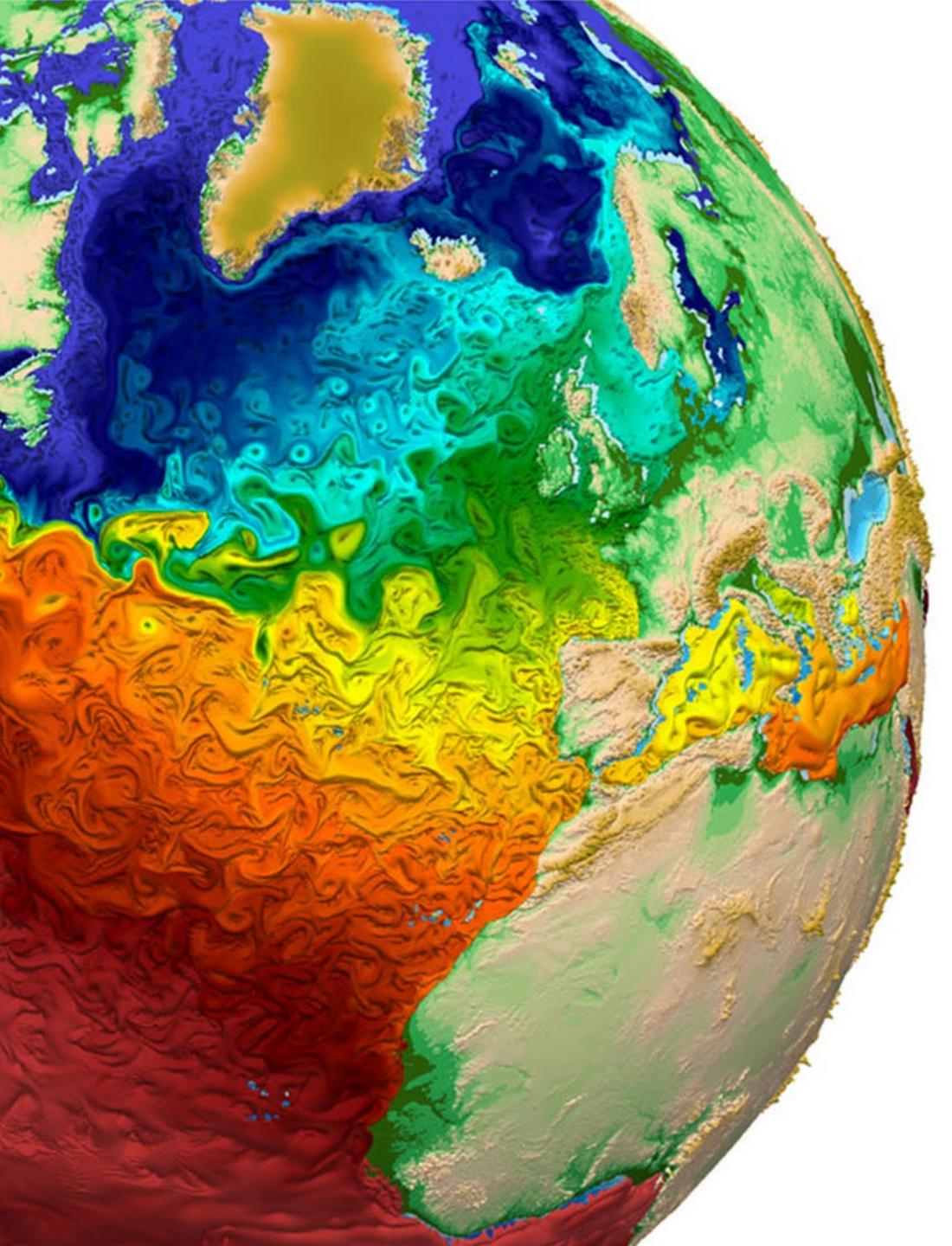
- Some color representations include a fourth channel: alpha
 - RGBA representation
- Alpha represents opacity
 - 1.0 is opaque
 - 0.0 is transparent
- These are referred to as RGBA colors
- Useful for simulating semi-transparent surfaces and compositing
 - e.g. visualizing MRI data



Operations on Colors



- We can add and subtract RGB triples component-wise
 - Like turning on multiple lights and turning off lights
 - Values add/subtract linearly, results clamped to $[0,1]$
- The set of all linear combinations of forms the space of all colors we can create
- To simulate reflection, we can multiply two RGB triples
 - e.g. reflecting white light off a blue surface $(1,1,1) \times (0,0,1) = (0,0,1)$
 - The surface absorbs non-blue wavelengths



Digital Images

Representing Intensity

Scientific Visualization
Professor Eric Shaffer

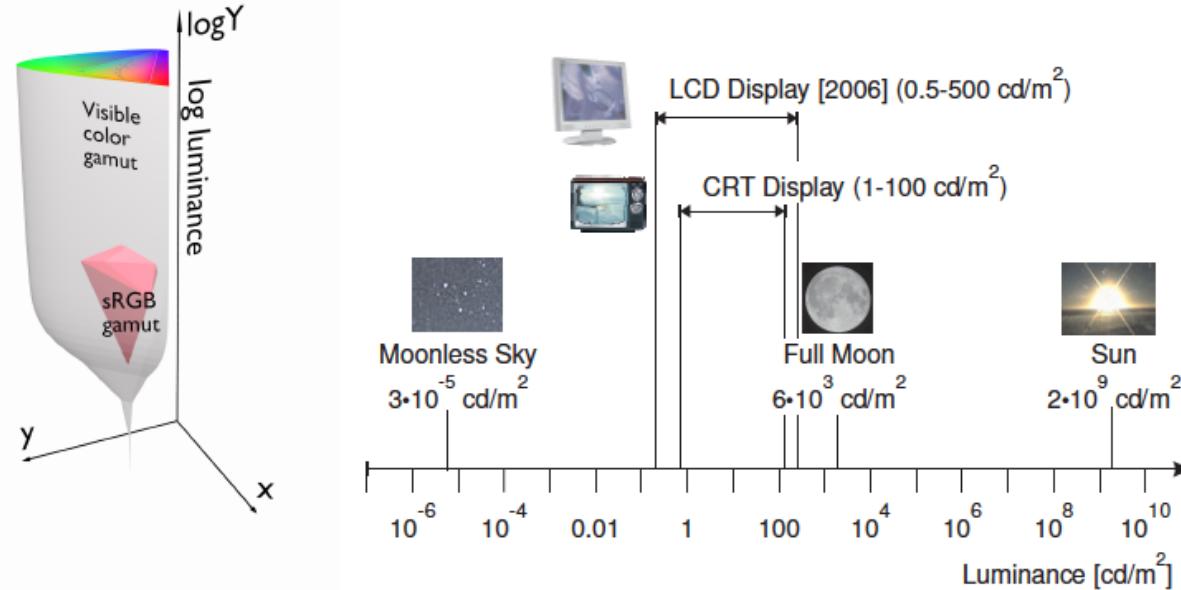
Intensity

- At display time intensity values typically are in 8-bit representations
- 8 bits per color channel (24-bit color)

Byte 0	Byte 1	Byte 2	Byte 3
Red	Green	Blue	Alpha

- Each channel is an 8-bit unsigned int
- Will sometimes see hexadecimal expressions for color range [0x00, 0xFF]
- Intensity levels in range [0,255]...can convert to float by dividing by 255

Luminance and Human Perception



Dictionary

Search for a word

 lu·mi·nance
/lōōmənəns/

noun PHYSICS

the intensity of light emitted from a surface per unit area in a given direction.

- the component of a television signal which carries information on the brightness of the image.

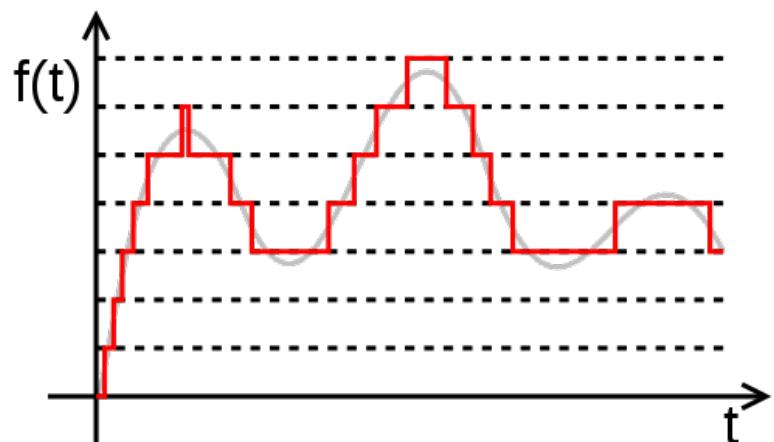
 Translations, word origin, and more definitions

We can't see the entire range from moonless sky to sun in one scene...but much more than the LCD range

- Many technologies do not record/display full range of luminance perceivable by humans
- HDR technologies offer higher contrast capabilities
- 256 gray levels is insufficient to look continuous

Intensity

- Some high-dynamic range (HDR) displays use a different representation
 - For HDR TV 10 bits or 12 bits per color channel
- OpenEXR file format can use 16-bit floating point for each channel
- In application level code, keep color values as at least 32-bit floats
 - Reduces impact of quantization on the colors
 - Reduction of precision will happen on the graphics processing unit (GPU)
- Quantization (in mathematics)



There are 1,056,964,609 single precision floating point numbers between 0 and 1meaning $\frac{1}{2}$ of all the possible 32 bit words are used to represent numbers in $[-1,1]$

The PNG Image File Format

Let's quickly look at a popular storage format for digital images

Portable Network Graphics



File format

Portable Network Graphics is a raster-graphics file format that supports lossless data compression. PNG was developed as an improved, non-patented replacement for Graphics Interchange Format. PNG supports palette-based images, grayscale images, and full-color non-palette-based RGB or RGBA images. [Wikipedia](#)

Hex	As characters
89 50 4E 47 0D 0A 1A 0A 00 00 00 00 0D 49 48 44 52	.PNG.....IHDR
00 00 00 01 00 00 00 01 08 02 00 00 00 90 77 53wS
DE 00 00 00 0C 49 44 41 54 08 D7 63 F8 CF C0 00IDAT..c....
00 03 01 01 00 18 DD 8D B0 00 00 00 00 49 45 4EIEN
44 AE 42 60 82	D.B`.

The PNG Image File Format

Header

Values (hex)	Purpose
89	Has the high bit set to detect transmission systems that do not support 8-bit data and to reduce the chance that a text file is mistakenly interpreted as a PNG, or vice versa.
50 4E 47	In ASCII , the letters <i>PNG</i> , allowing a person to identify the format easily if it is viewed in a text editor.
0D 0A	A DOS-style line ending (CRLF) to detect DOS-Unix line ending conversion of the data.
1A	A byte that stops display of the file under DOS when the command type has been used—the end-of-file character.
0A	A Unix-style line ending (LF) to detect Unix-DOS line ending conversion.

Chunks

Length	Chunk type	Chunk data	CRC
4 bytes	4 bytes	<i>Length</i> bytes	4 bytes

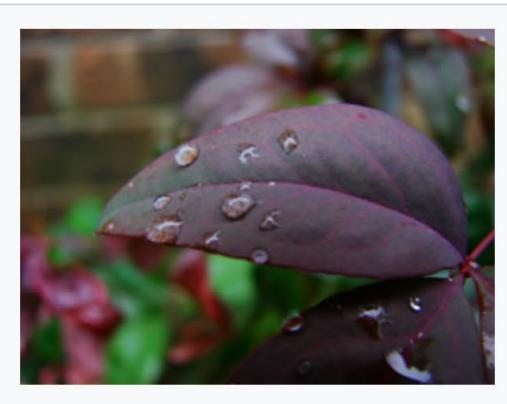
Pixel formats

PNG color types						
Color type	Name	Binary			Masks	
		A	C	P		
0	Grayscale	0	0	0	0	
2	Truecolor	0	0	1	0	color
3	Indexed	0	0	1	1	color, palette
4	Grayscale and alpha	0	1	0	0	alpha
6	Truecolor and alpha	0	1	1	0	alpha, color

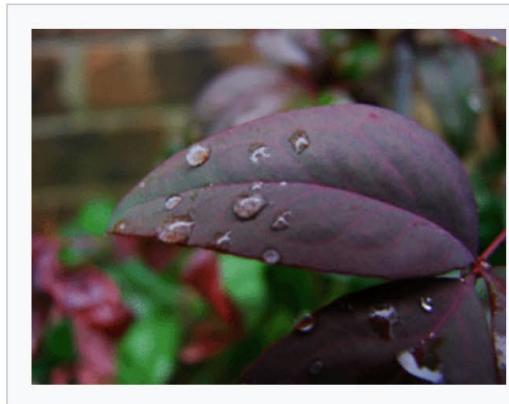
Pay attention to
how you save
important images

Implications for Visualization

- Cannot be sure viewers will see the same colors that you compute
 - Display differences
 - Quantization in storage and/or transmission to display
- If you map values to color, allow users to query for the original numerical value



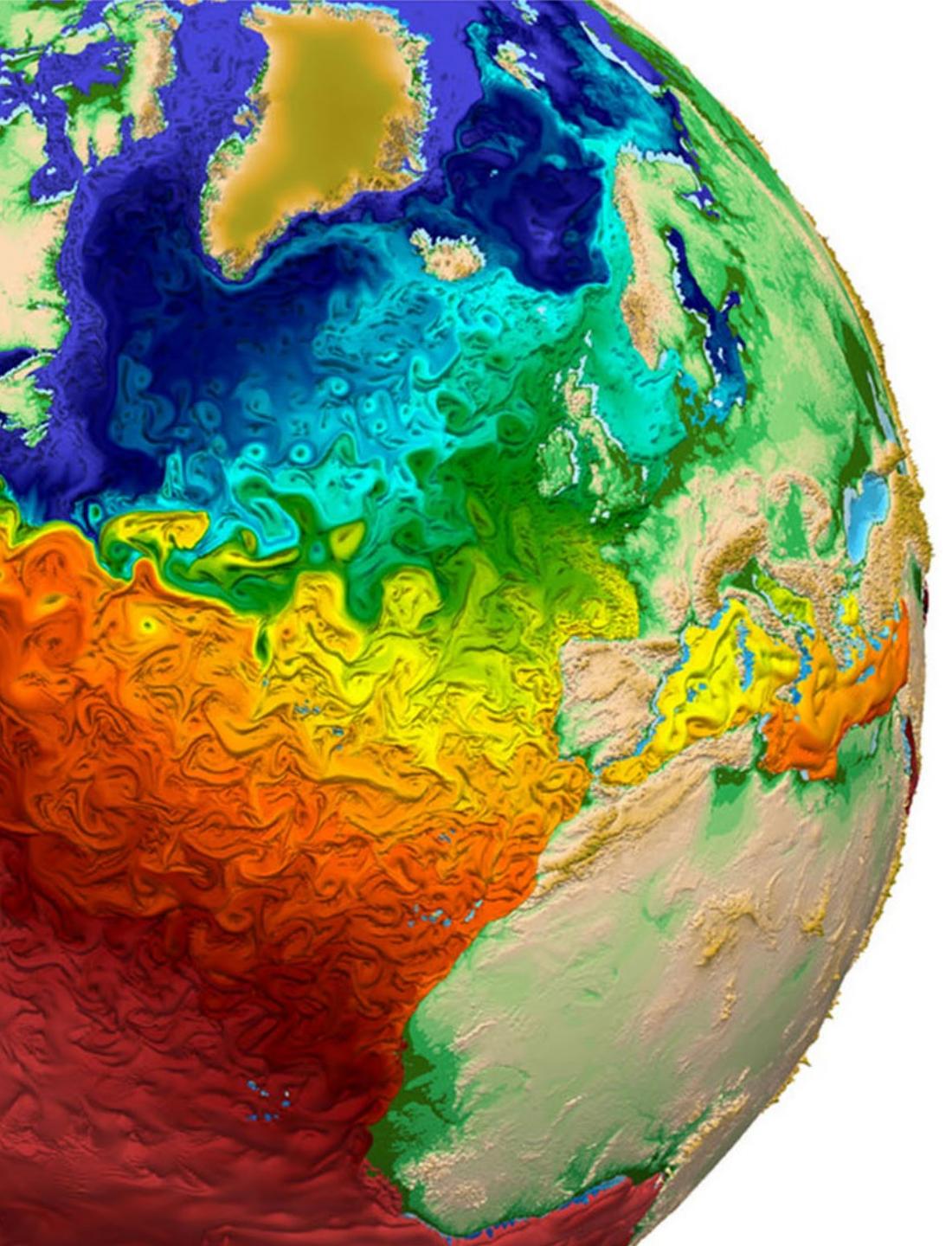
24 bit.png
16,777,216 colors
98 KB



8 bit.png
256 colors
37 KB (~62%)



4 bit.png
16 colors
13 KB (~87%)



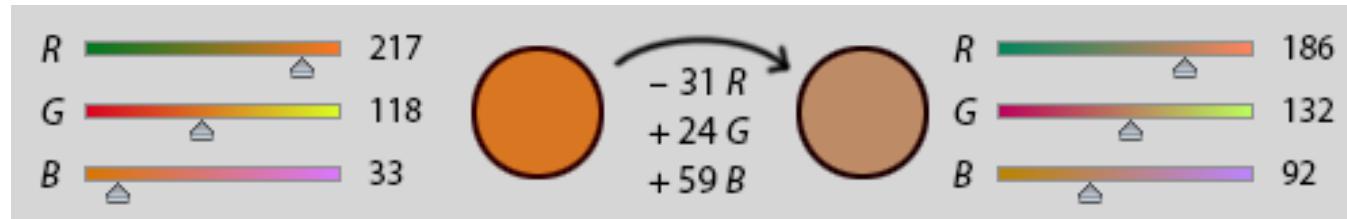
Digital Images

HSV Color Space

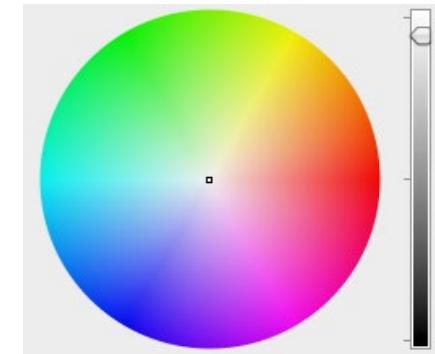
Scientific Visualization
Professor Eric Shaffer

HSV Color Space

For artists...or anyone...color picking in RGB is difficult



RGB Color Picker



HSV Color Picker

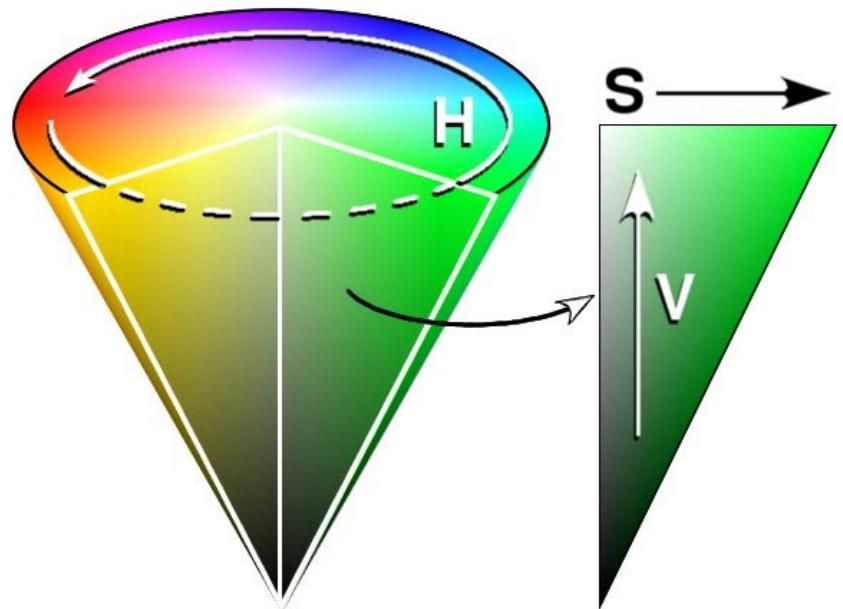
Hue, Saturation and Value (HSV) is an alternative color space

Equivalent to RGB in the colors it can represent

Easier to design color picking interfaces

HSV Color Space

- Hue [0,360] is angle about color wheel
 0° = red, 60° = yellow, 120° = green,
 180° = cyan, 240° = blue, 300° = magenta
- Saturation [0,1] is distance from gray
- Value [0,1] is distance from black

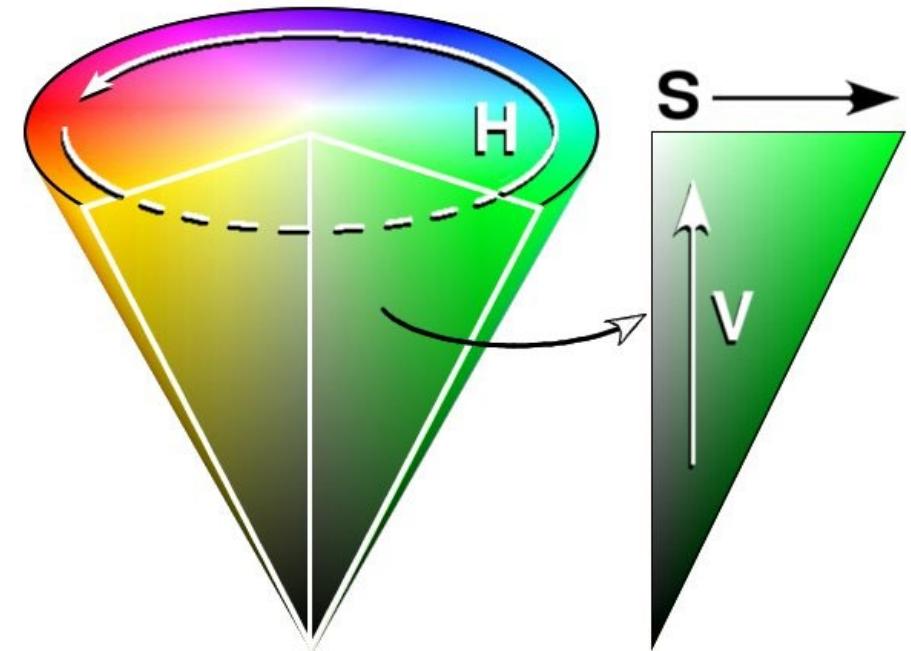


RGB to HSV Conversion

We have 3 channel values R, G, and B

- All in [0.0, 1.0]
- Let $\text{maxRGB} = \max(R, G, B)$
- Let $\text{minRGB} = \min(R, G, B)$
- $S = (\text{maxRGB} - \text{minRGB})/\text{maxRGB}$
- $V = \text{maxRGB}$
- To compute H

```
 $\Delta = \text{maxRGB} - \text{minRGB}$ 
if  $\text{maxRGB} == R \rightarrow H = (G - B)/\Delta$ 
if  $\text{maxRGB} == G \rightarrow H = 2 + (B - R)/\Delta$ 
If  $\text{maxRGB} == B \rightarrow H = 4 + (R - G)/\Delta$ 
 $H = (60 * H) \bmod 360$ 
```



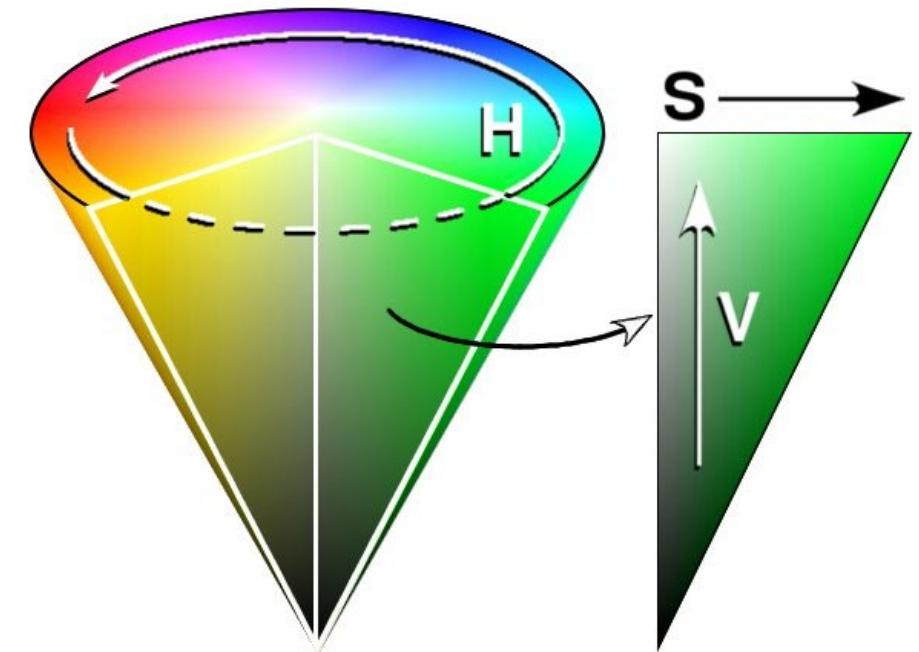
HSV to RGB Conversion

We have H in $[0,360]$, S in $[0,1]$, and V $[0,1]$

$$f(n) = V - VS \max(0, \min(k, 4 - k, 1))$$

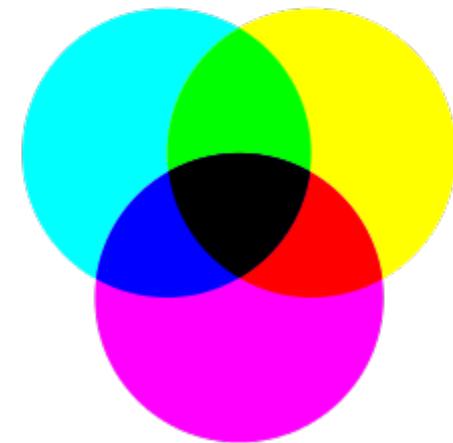
$$k = (n + \frac{H}{60^\circ}) \bmod 6$$

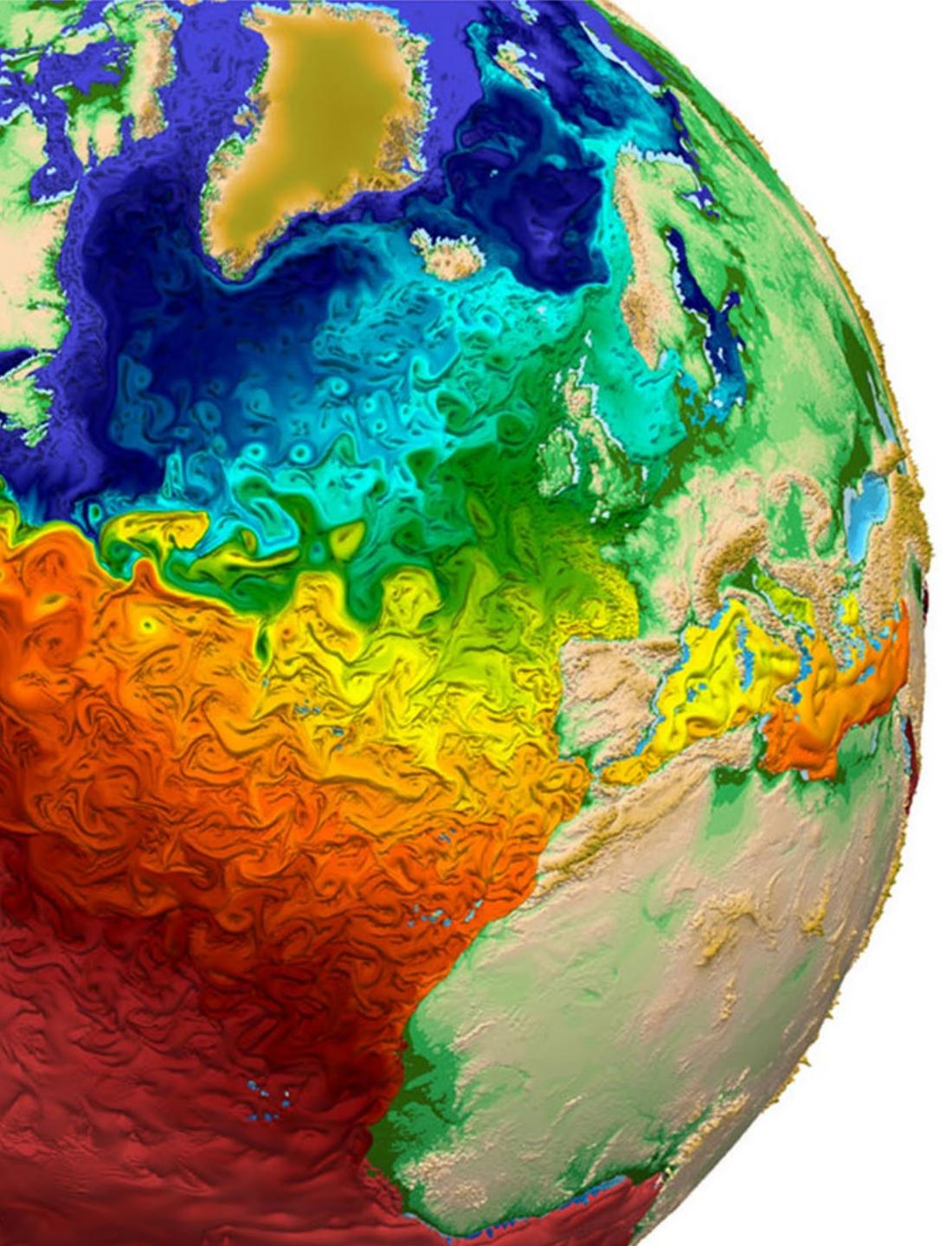
$$(R, G, B) = (f(5), f(3), f(1))$$



Color Spaces: Lies Your Kindergarten Teacher Told You

- Red, Yellow and Blue are not *The Primary Colors*
 - Taken from Cyan, Magenta, Yellow color space used for reflective displays
 - Printing...or finger painting
- Any set of wavelengths can serve as primaries
 - Defines a set of colors you can create by mixing
 - Some let you generate more colors...some fewer
- A 3 wavelength color space **cannot** produce all the colors a person can see
 - To understand why, we need to look at perceptually-defined color spaces

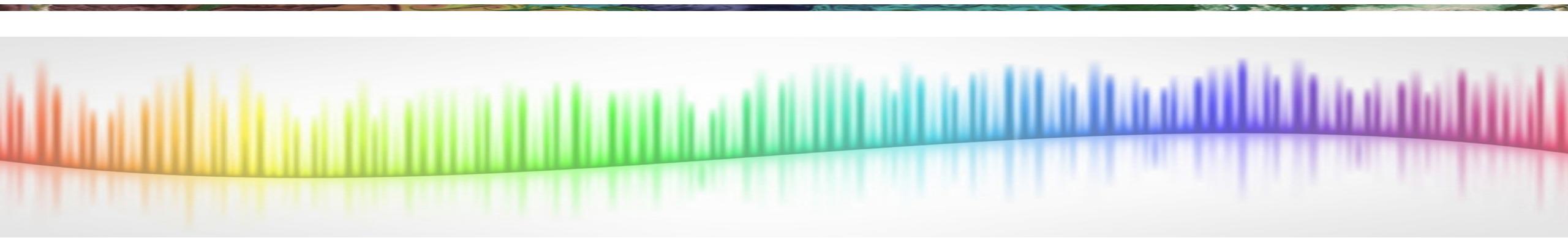




Perceptual Color Spaces

Scientific Visualization
Professor Eric Shaffer

The Most Important Thing to Know About Color



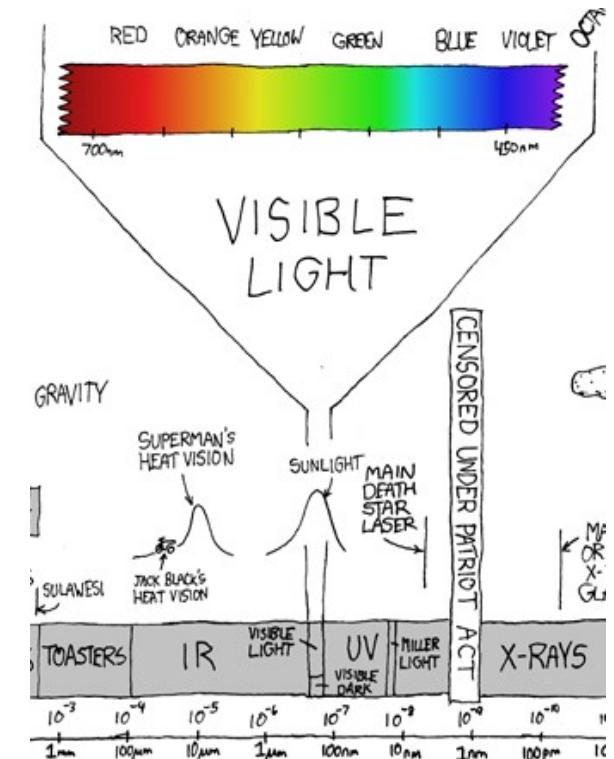
Color is a perceptual phenomenon

It is not a physical property of a material or of light

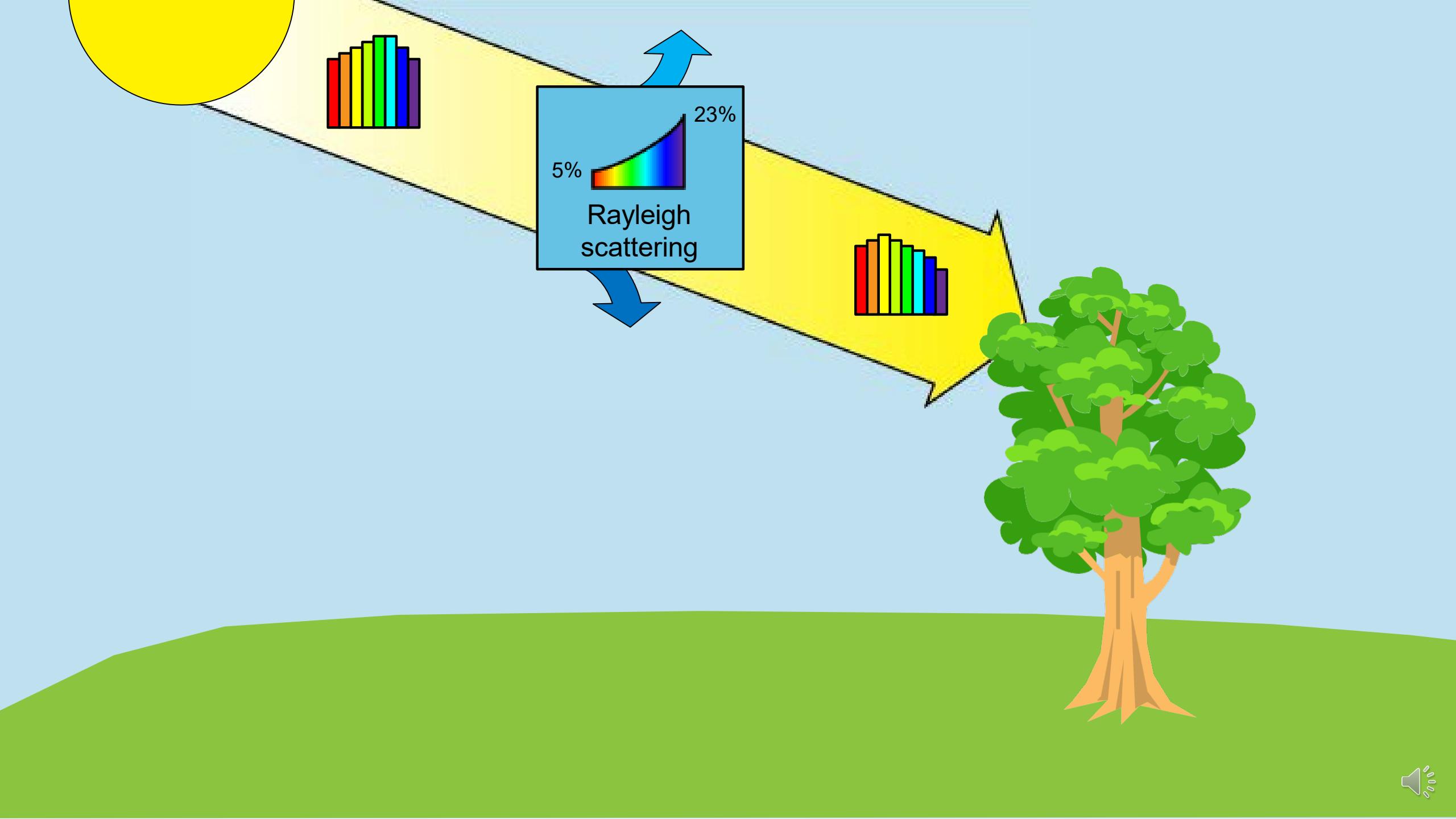
So, how does color perception work?

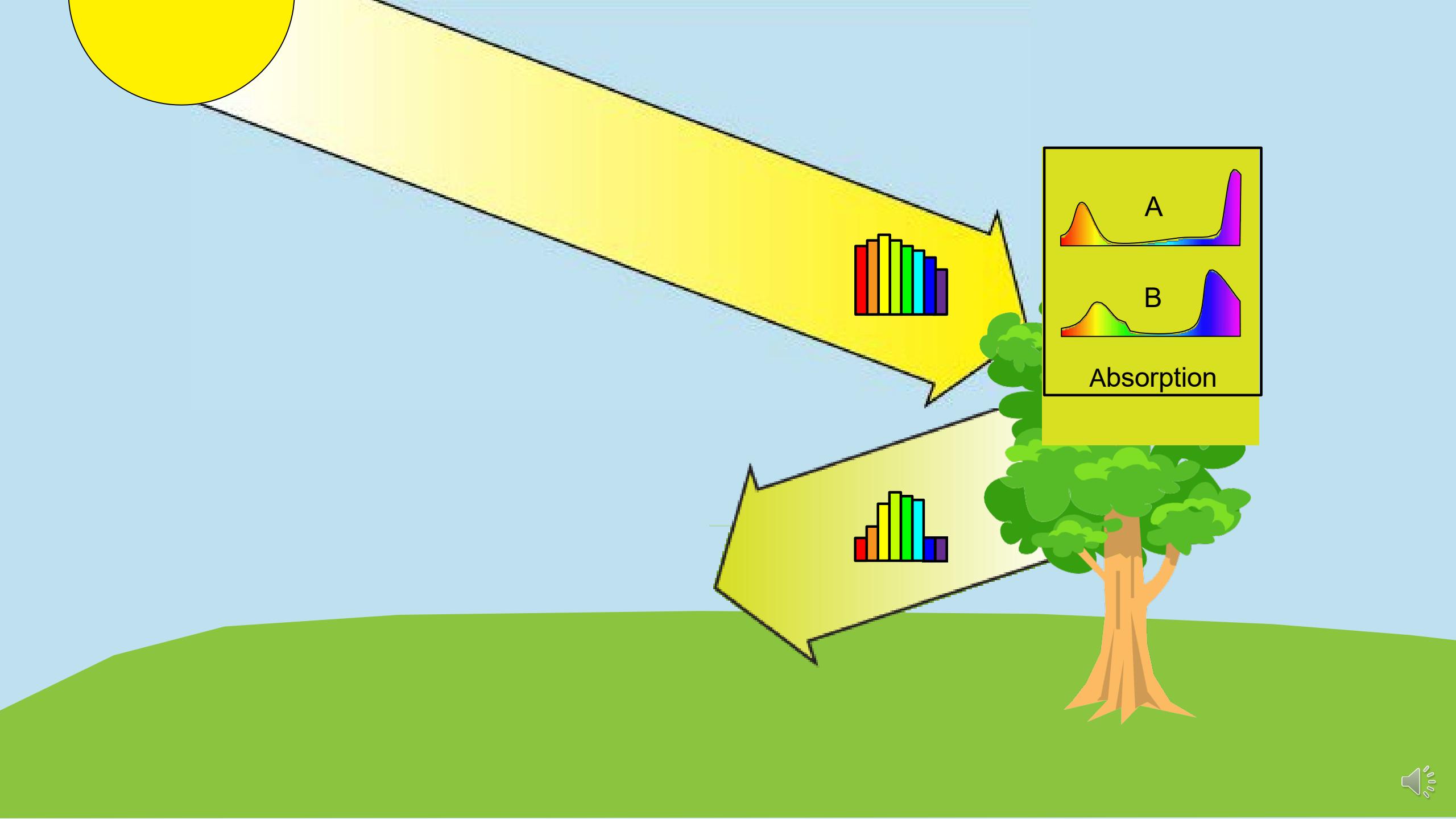
Light

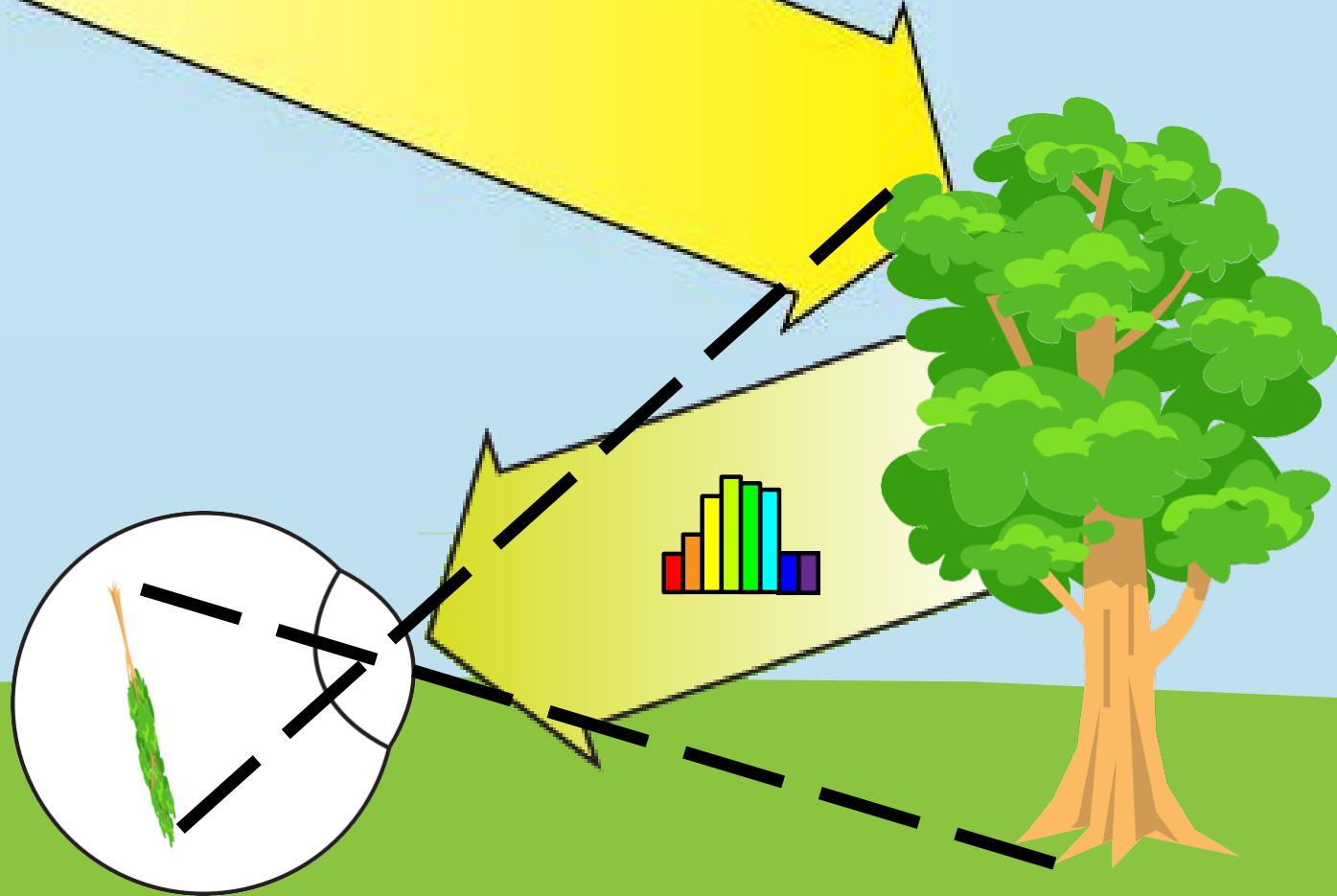
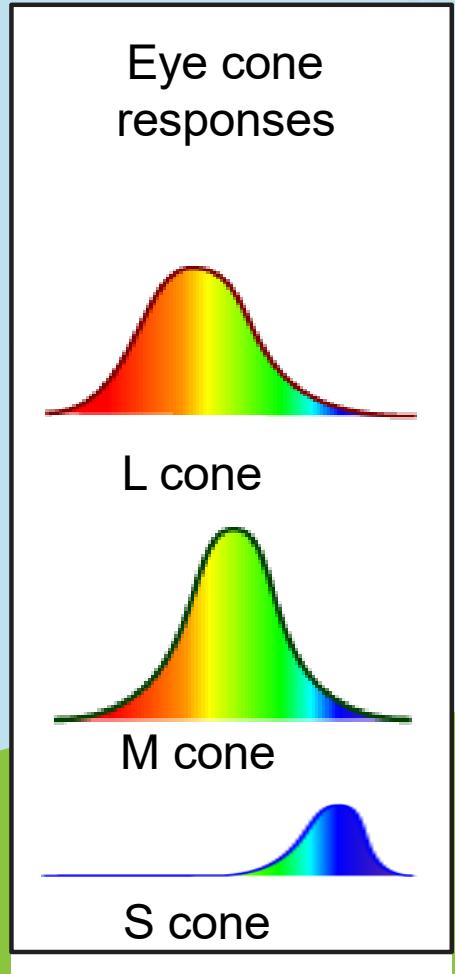
- Human vision senses energy in a portion of the electromagnetic spectrum
- Energy carried by photons
- The energy of each photon is proportional to its frequency
 - Frequency = inverse of wavelength
- The intensity of light is related to the number of photons received



xkcd.com/273

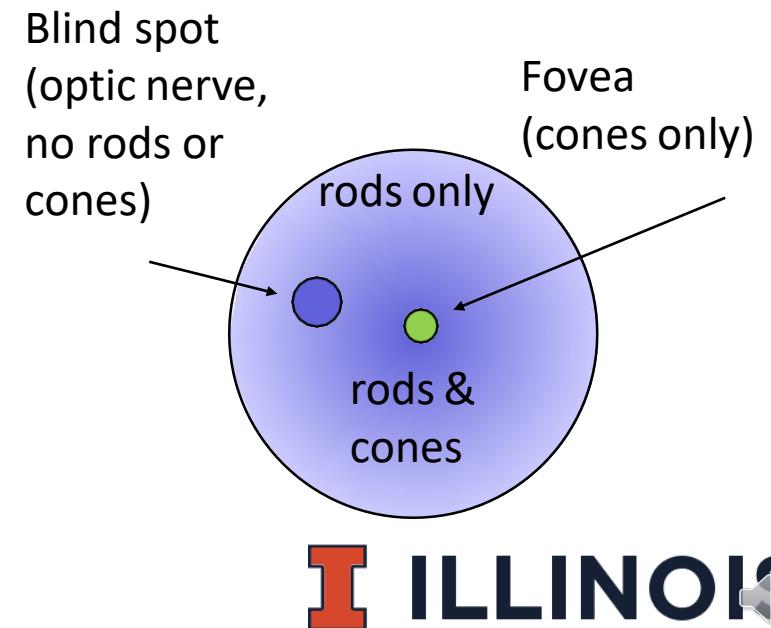






Human Visual System: Rod and Cone Cells

- Rods measure intensity
 - 80 million
 - denser away from fovea
 - astronomers learn to glance off to the side of what they are studying
 - sensitive, shut down in daylight
- L,M and S cones
 - 5 million total
 - 100K – 325K cones/mm² in fovea
 - 150 hues
- Combined
 - 7 million shades



Cone Cell Response

$V(\lambda)$ is the luminosity function

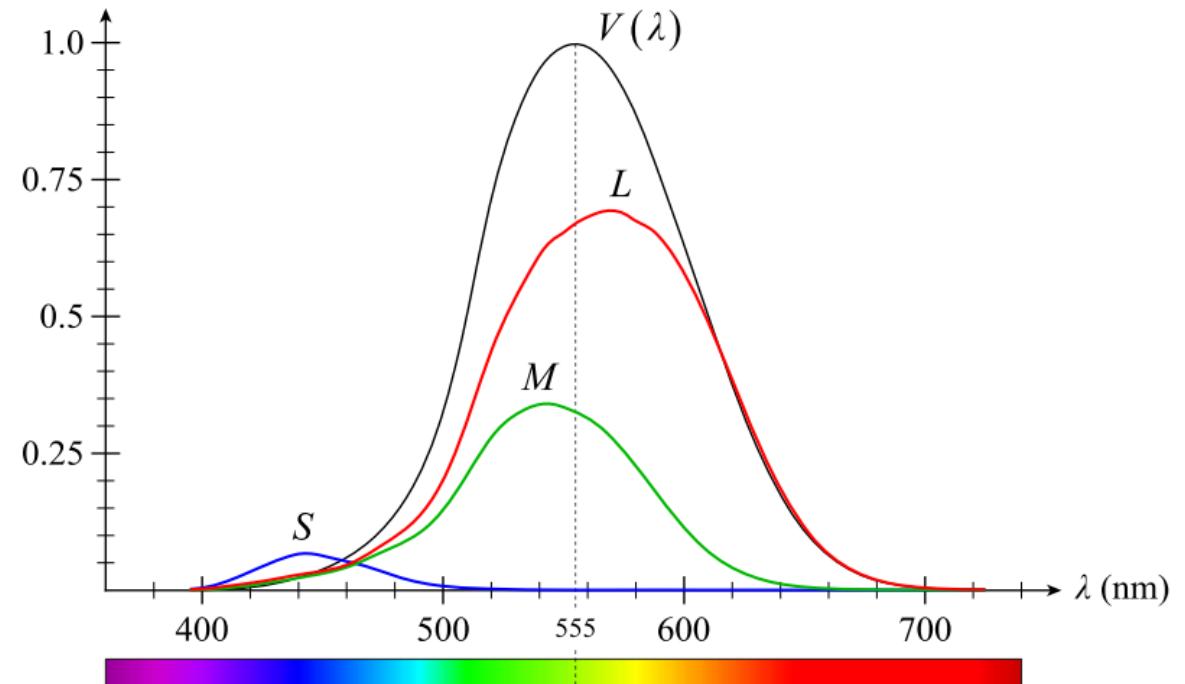
- It indicates perceived brightness

Of total number of cones:

- 63% are L cones
- 31% are M cones
- 6% are S Cones

Luminosity peaks at 555nm

- Green-yellows are perceived as brightest
- Blues are perceived as dark



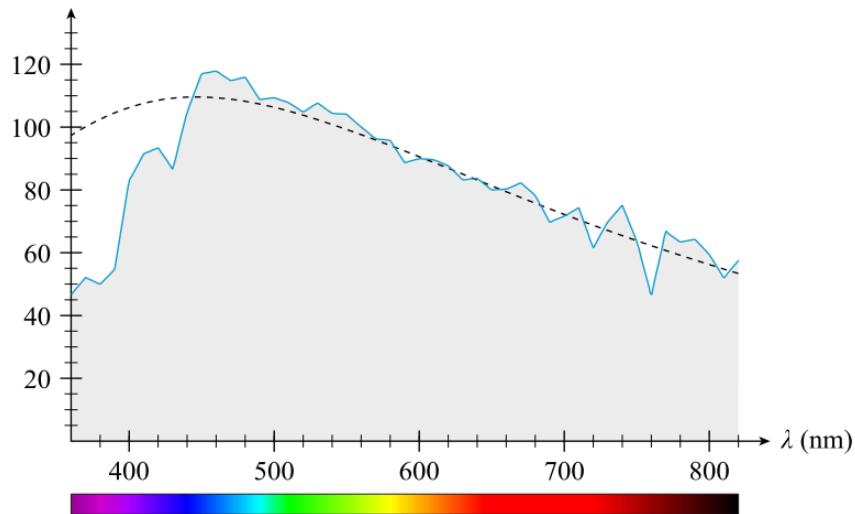
Graph shows relative sensitivity of each type of cone to different wavelengths of light

The Human Visual System

A color corresponds to some amount of stimulus of the cones

Light is usually a mix of wavelengths

- A spectral power distribution
- This distribution is for a “white” light

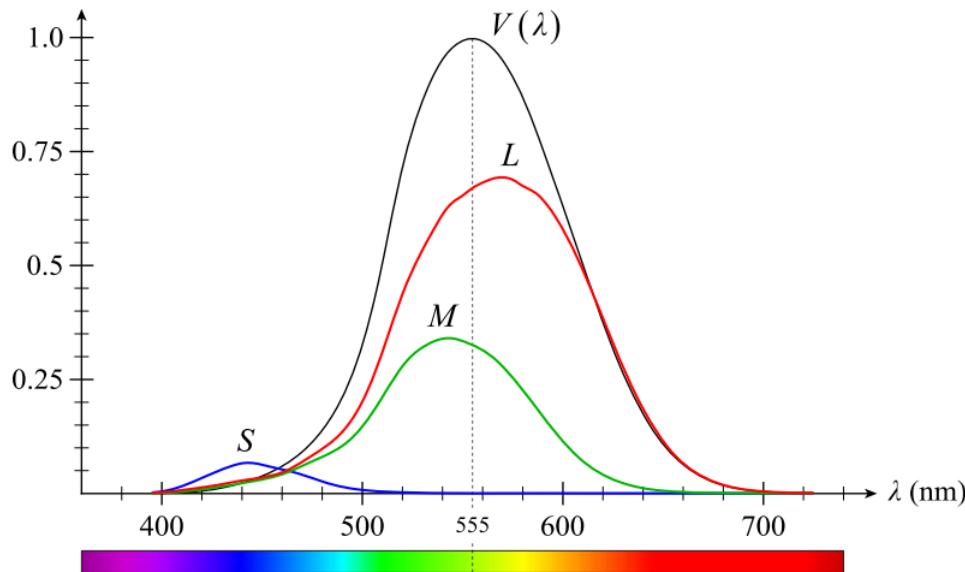


Two different distributions can produce the same stimulus

- Thus they produce the same perceived color
- Different distributions that produce the same color are *metamers*

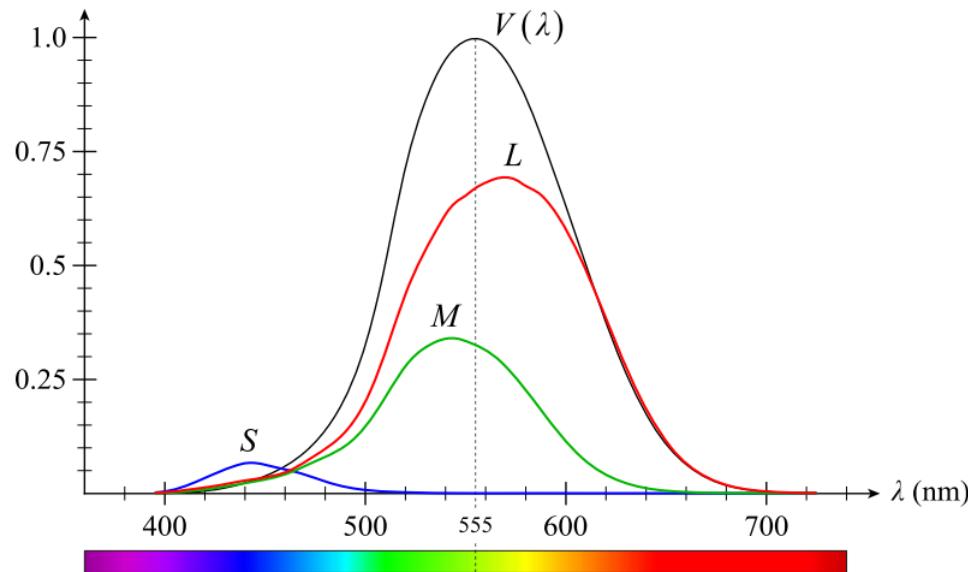
Color Spaces

- To create a color image we need a way to specify colors
- A color corresponds to some level of stimulation of the L, M, S cones
 - The set of possible tristimulus values forms a 3D vector space
- The basis vectors for the space are not physical
 - No wavelength of light stimulates only one kind of cone



Color Spaces

- To create a color space with a physical meaning
 - We can choose 3 discrete wavelengths which we will call primaries.
- The wavelengths can be mixed at different intensities
 - These spectral distributions correspond to colors defined for some average viewer





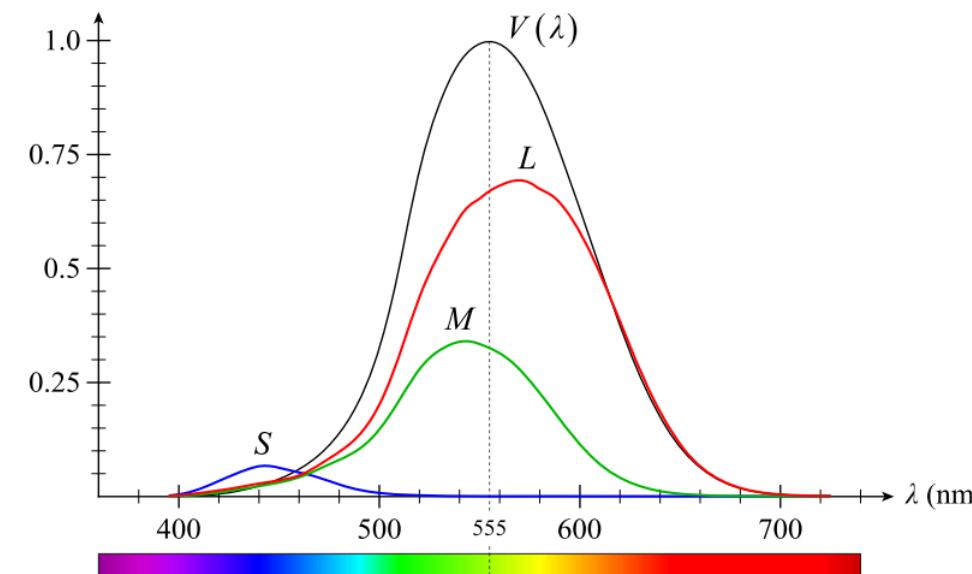
CIE RGB Color Space

CIE RGB Color Space

- Defined in 1931 by the International Commission on Illumination
- CIE is from the French name Commission Internationale de L'éclairage

Used primary wavelengths

- 435.8 nm (“blue”)
- 546.1 nm (“green”)
- 700 nm (“red”)



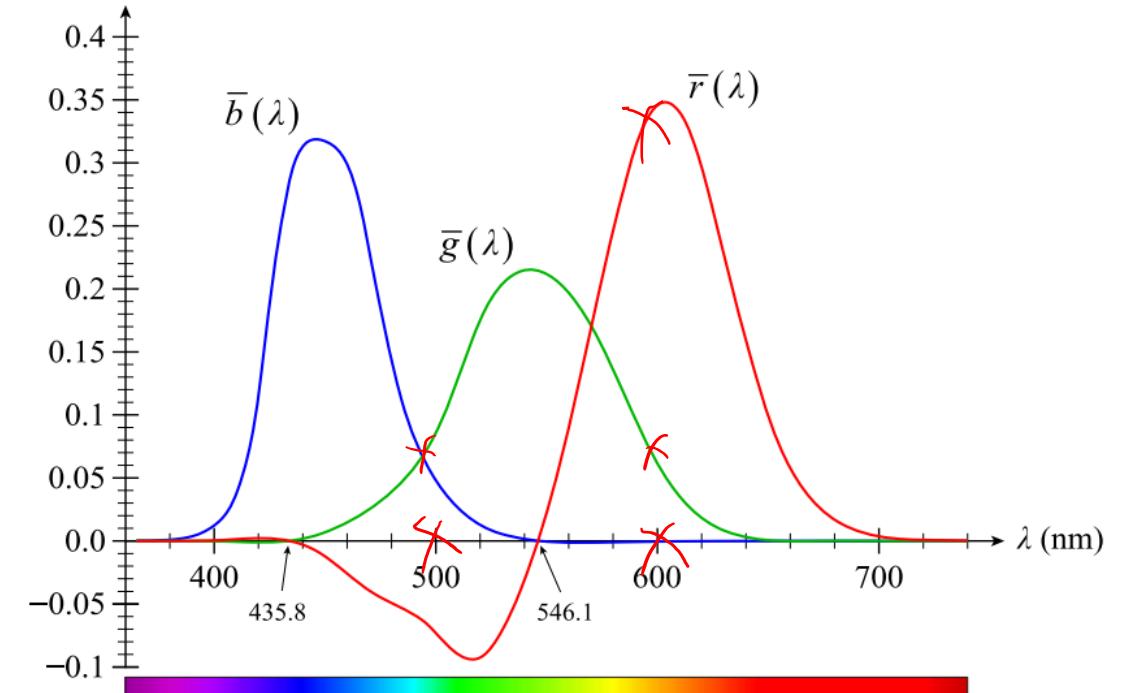
Color Matching Experiments

The Experiment

- A set of viewers were shown a mono-spectral light
- The viewers tried to match the color by mixing the 3 primaries
- This was sometimes impossible
- ...but a match could be made by mixing one of primaries with the mono-spectral light
- You can see these events in the graph as negative values

The plot of the functions was created by

- Repeating the experiment for different wavelengths of mono-spectral light
- Averaging/filtering the data from the test subjects





CIE RGB Color Space

The intensity of the primaries for the CIE RGB color space are computed as follows:
Given a spectral power distribution $P(\lambda)$

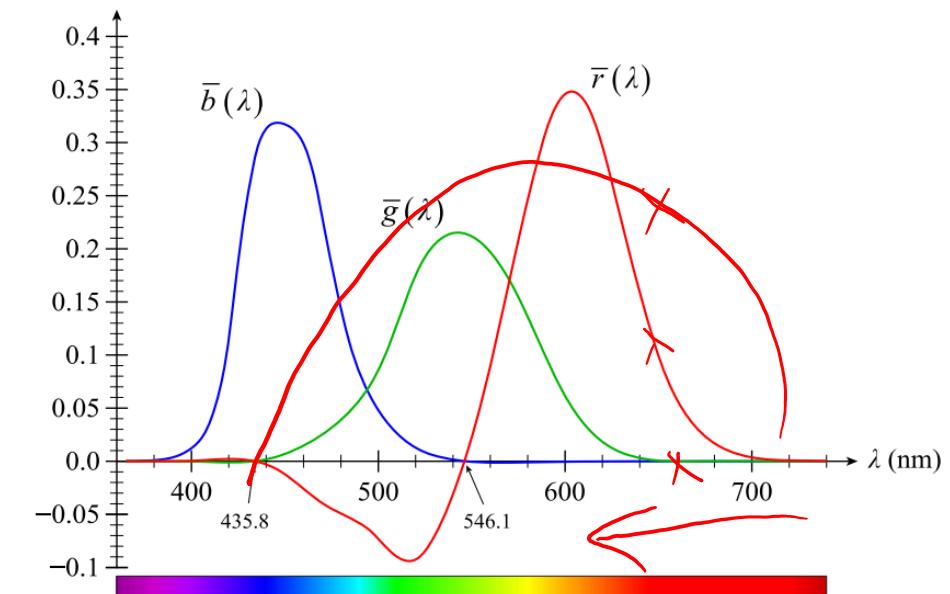
Compute

$$R = \int_{\lambda} \bar{r}(\lambda) P(\lambda) d\lambda$$

$$G = \int_{\lambda} \bar{g}(\lambda) P(\lambda) d\lambda$$

$$B = \int_{\lambda} \bar{b}(\lambda) P(\lambda) d\lambda$$

This is not the RGB color space you know
It is the basis for it...as we'll see



CIE XYZ Color Space

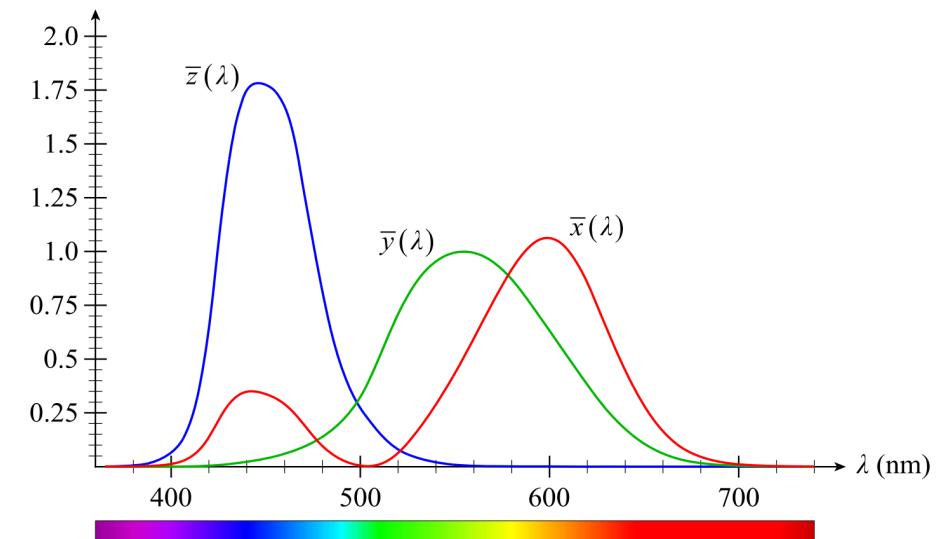
CIE RGB had two problems

- Negative values...people don't like negative values
- It would be useful to separate perceived luminance from chromaticity
 - i.e. separate "brightness" and "hue"

CIE XYZ solves those problems

- Convert from RGB to XYZ

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 2.768892 & 1.751748 & 1.130160 \\ 1 & 4.590700 & 0.060100 \\ 0 & 0.056508 & 5.594292 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



CIE XYZ Color Space

CIE XYZ

- Primaries only take on positive values
- $\bar{y}(\lambda)$ corresponds closely to the values of the luminosity function
 - So Y corresponds to perceived brightness

CIE XYZ

- Primaries only take on positive values
- $\bar{y}(\lambda)$ corresponds closely to the values of the luminosity function
 - So Y corresponds to perceived brightness

xyY Color Space

It can be useful to have normalized chromaticity values

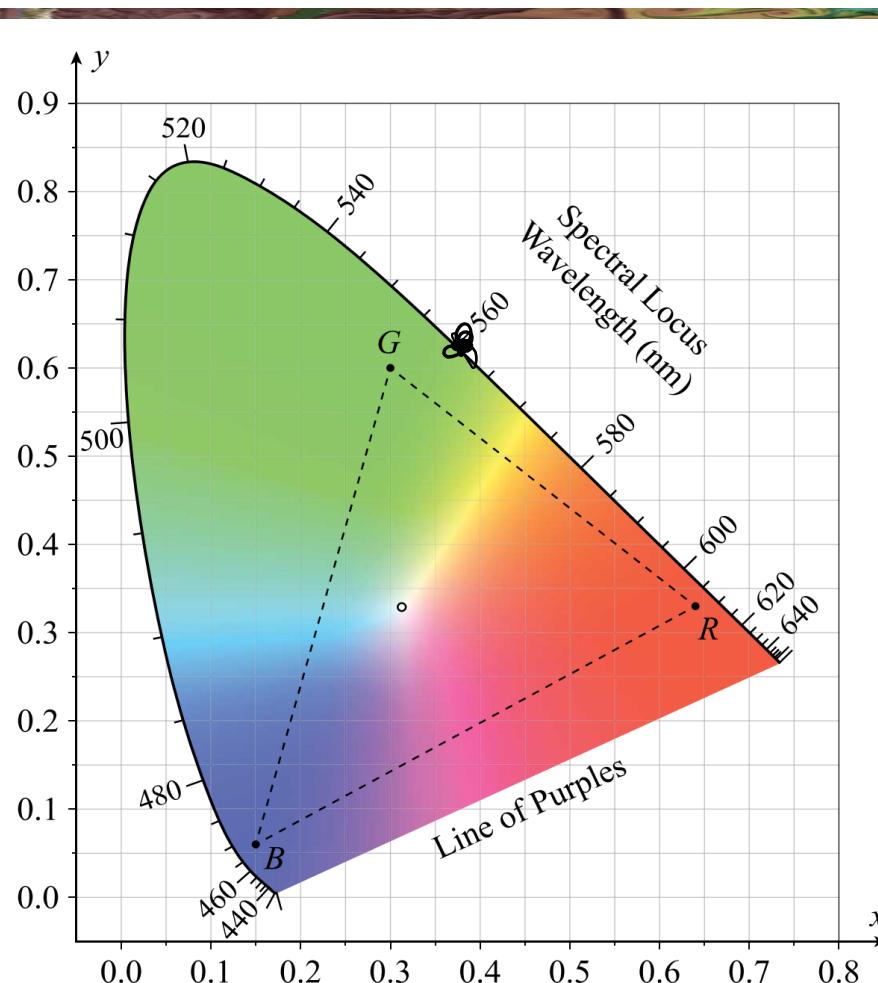
- Fall in range [0,1]
- Do not change with luminance

Standard way to do this is to take

$$x = \frac{X}{X + Y + Z}, \quad y = \frac{Y}{X + Y + Z}, \quad \text{and} \quad z = \frac{Z}{X + Y + Z}$$

- Convention is to use x and y to indicate chromaticity
- Combined with the original Y value gives a color in the xyY colorspace

CIE xy Chromaticity Diagram



The diagram depicts all colors visible to an average human

The curved boundary is the spectral locus

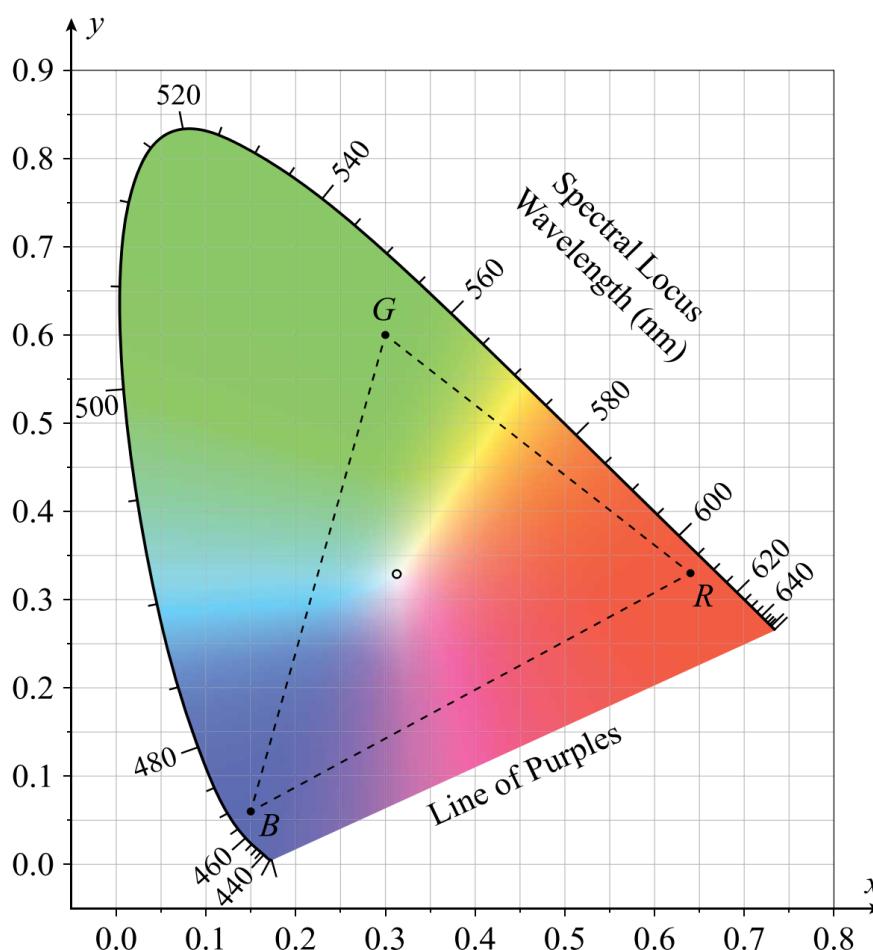
- It consists of all the colors associated with a single wavelength
- The bottom line of purples are not single wavelength colors

Points not on the curved boundary are mixture of multiple wavelengths

The standard sRGB color space is defined by 3 points labeled R, G, and B

- All the colors possible in that space lie in the RGB triangle
- The set of producible colors in a space is called the **gamut**

Three Primary Colors Cannot Produce All Possible Colors



The CIE RGB color space includes all possible colors
...at least for most people

No space defined by 3 primary colors can include all colors
....no triangle can cover the horseshoe in the diagram

What about CIE RGB space? It has 3 primaries

The CIE RGB primaries are not colors
The CIE RGB are not physical lights
The CIE RGB primaries are a function of the color-matching curves



Standard Illuminants

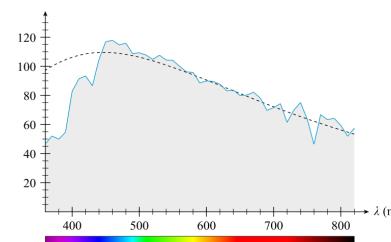
So...what point on the diagram corresponds to white light?

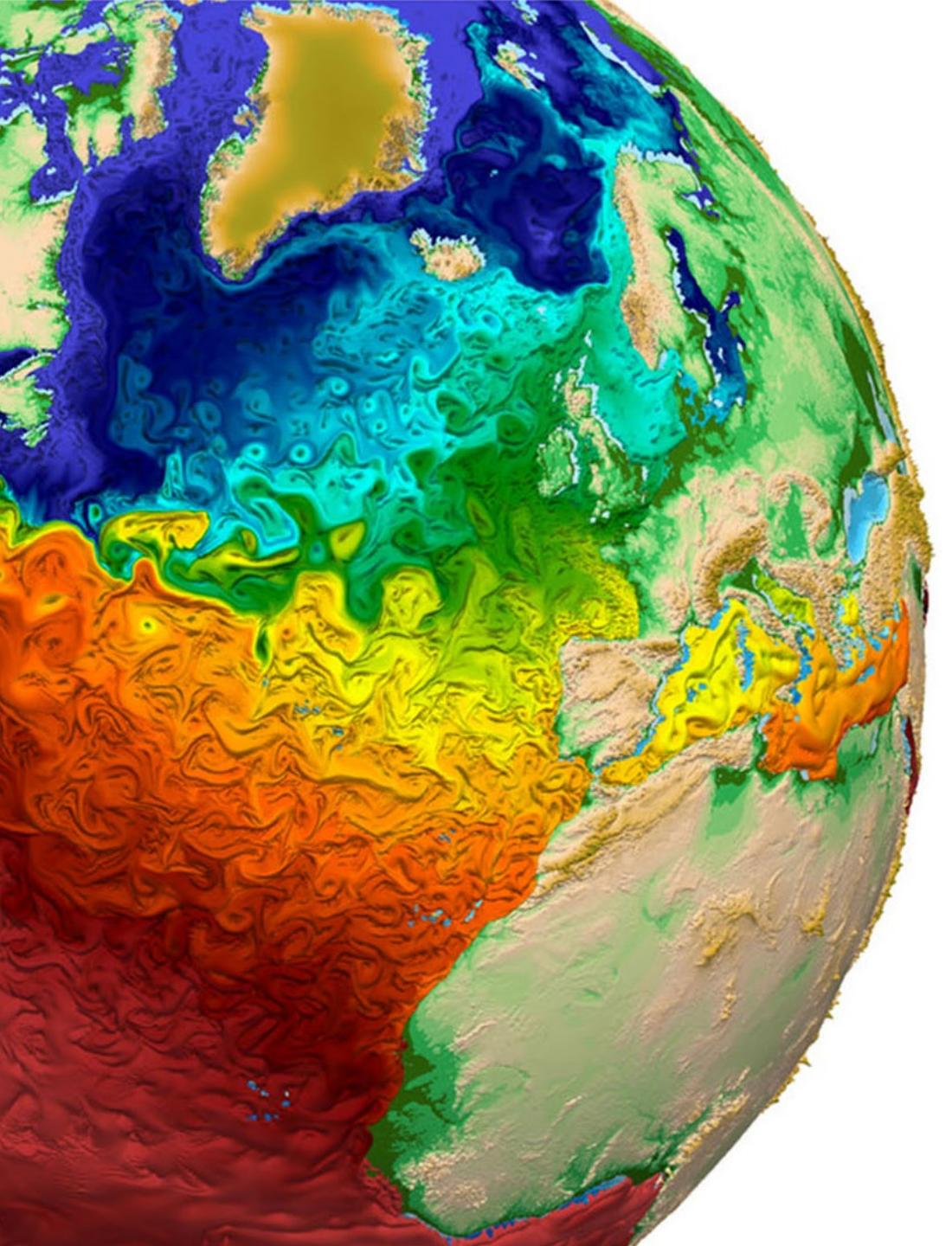
- Perfectly white light occurs at $\left(\frac{1}{3}, \frac{1}{3}\right)$
- This is called standard illuminant E

Most light sources are not perfectly white.

Illuminant D65 is at (0.3127,0.2390)

- Approximates average daylight in most geographic locations
- Defines white light for the sRGB color space
- Has the spectral power distribution shown below





The sRGB Color Space

Scientific Visualization
Professor Eric Shaffer

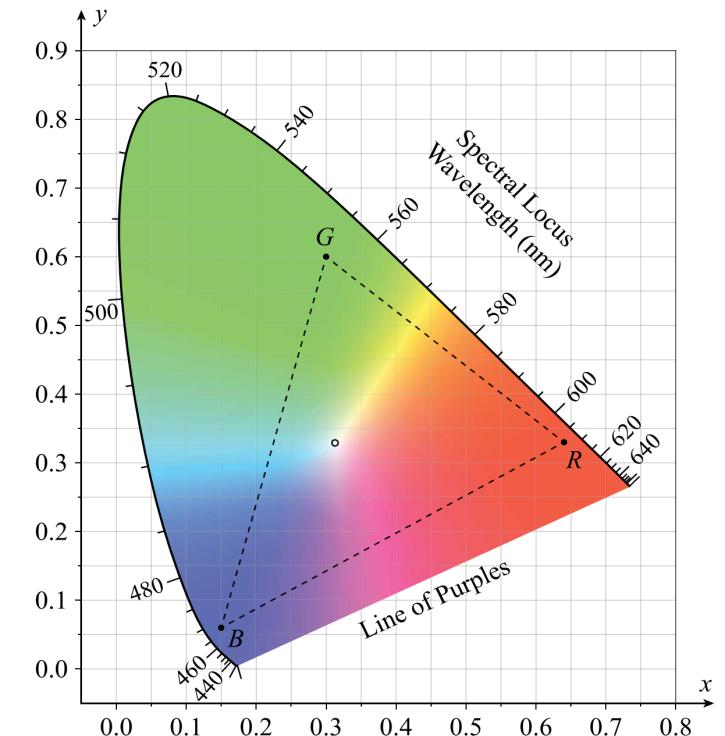
sRGB Color Space

- The standard RGB color space is defined in terms of the CIE XYZ space
- The primaries are

$$(x_R, y_R) = (0.64, 0.33)$$

$$(x_G, y_G) = (0.30, 0.60)$$

$$(x_B, y_B) = (0.15, 0.06)$$



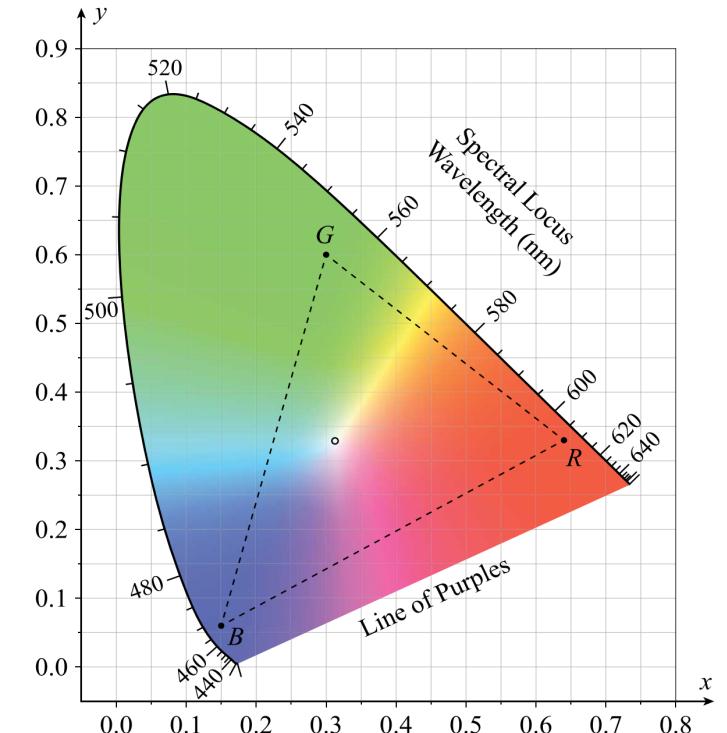
sRGB Color Space

- The standard RGB color space is defined in terms of the CIE XYZ space
- The primaries are
 - $(x_R, y_R) = (0.64, 0.33)$
 - $(x_G, y_G) = (0.30, 0.60)$
 - $(x_B, y_B) = (0.15, 0.06)$
- You can compute the Y coordinates....
 - Require that the primaries sum to D65 with $Y_{D65} = 1$
 - Convert to XYZ space and solve for luminance there

$$Y_R = 0.212639$$

$$Y_G = 0.715169$$

$$Y_B = 0.072192$$



Conversion from XYZ to sRGB

We want to find a matrix M_{sRGB} that converts a XYZ color to sRGB

Each sRGB primary should convert to its defined XYZ coordinates

That gives us

$$M_{sRGB} \begin{bmatrix} \frac{x_R}{y_R} Y_R & \frac{x_G}{y_G} Y_G & \frac{x_B}{y_B} Y_B \\ y_R & Y_G & Y_B \\ Y_R & Y_B & Y_B \\ \frac{z_R}{y_R} Y_R & \frac{z_G}{y_G} Y_G & \frac{z_B}{y_B} Y_B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The columns of the RHS matrix are the sRGB coordinates of the primaries

Conversion from XYZ to sRGB

We can substitute in the known xyY values for variables in middle matrix

e.g. $(x_g, y_g, Y_g) = (0.3, 0.6, 0.715169)$

$$\mathbf{M}_{sRGB} \begin{bmatrix} \frac{x_R}{y_R} Y_R & \frac{x_G}{y_G} Y_G & \frac{x_B}{y_B} Y_B \\ Y_R & Y_G & Y_B \\ \frac{z_R}{y_R} Y_R & \frac{z_G}{y_G} Y_G & \frac{z_B}{y_B} Y_B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We can then solve to get $\mathbf{M}_{sRGB} = \begin{bmatrix} 3.240970 & -1.537383 & -0.498611 \\ -0.969244 & 1.875968 & 0.041555 \\ 0.055630 & -0.203977 & 1.056972 \end{bmatrix}$

Conversion from sRGB to XYZ

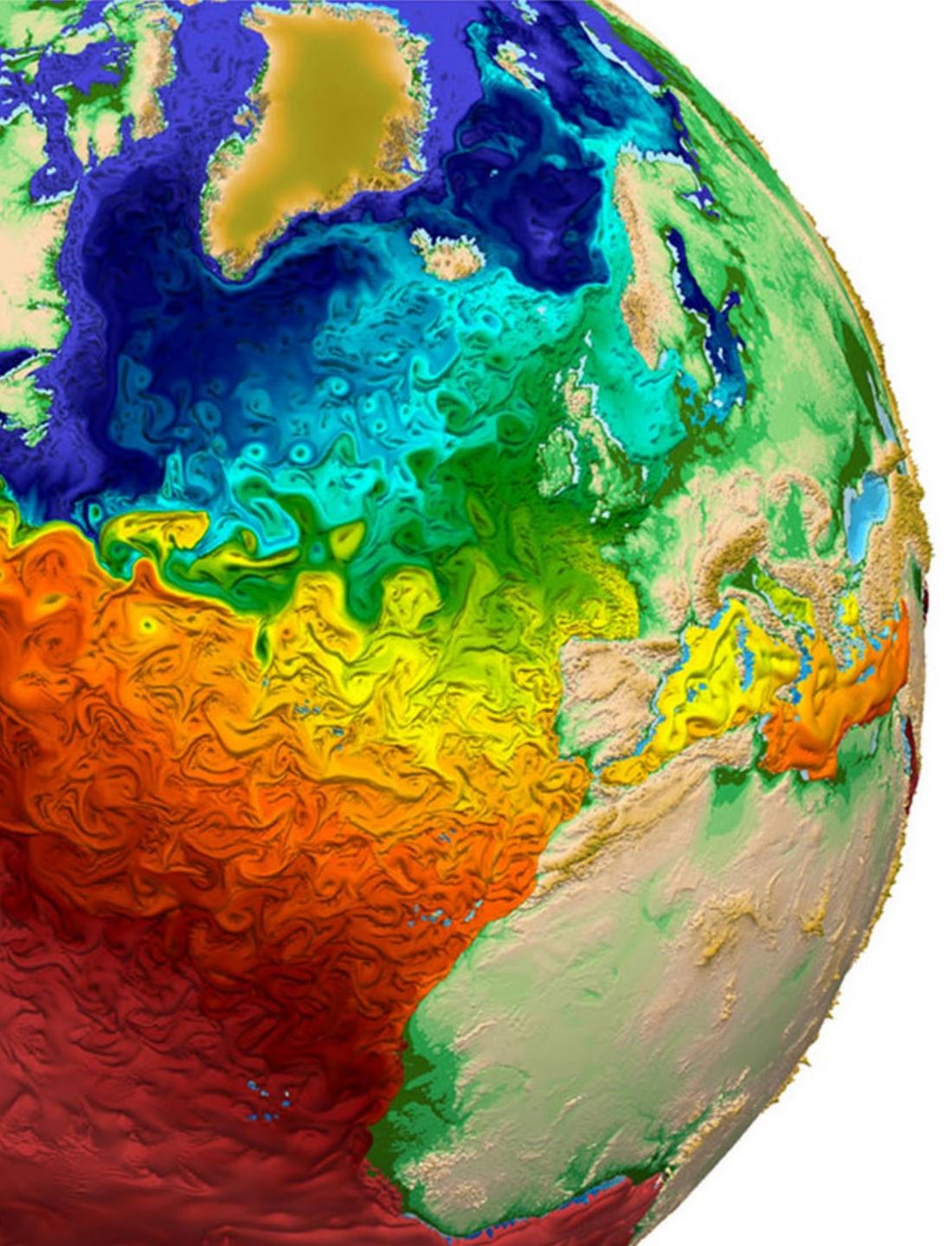
To convert from sRGB to XYZ, we just need to use the inverse of M_{sRGB}

$$M_{sRGB}^{-1} = \begin{bmatrix} 0.412391 & 0.357584 & 0.180481 \\ 0.212639 & 0.715169 & 0.072192 \\ 0.019331 & 0.119195 & 0.950532 \end{bmatrix}$$

sRGB: The Big Picture

sRGB is the standard color space used in modern computing

- Given an RGB triple to display, a web browser assumed it is an sRGB value
- Gives displays the opportunity to exhibit color uniformity
- In practice, this is still difficult to do
 - e.g. LCD-LED black isn't the same as OLED black



Gamma Correction

Scientific Visualization
Professor Eric Shaffer



Gamma Correction

In the days of CRT displays, due to the nature of the technology,

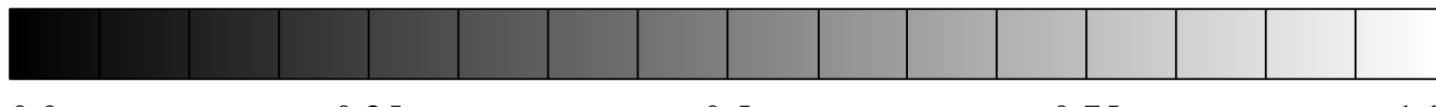
- the brightness of a subpixel was non-linear function of the input brightness
- $V_{display} = V_{signal}^\gamma$
- γ varied by display, but 2.2 was a typical value

Color channels have intensity values in the range [0,1]

This meant displayed colors were darker than the input color

This can be adjusted by ***gamma correction***...using inputs of $V_{signal}^{1/\gamma}$

Linear values



0.0 0.25 0.5 0.75 1.0



Gamma corrected values

Gamma Correction

- It's not clear if the CRT gamma was an unavoidable feature
 - Possibly a design choice
 - Human vision is more sensitive to lower intensity light
 - Human vision is more able to differentiate darker shades
- LCD-LEDs do not have to use gamma
 - Most do...most use a gamma of 2.2...but not all
 - Gamma values for different displays will vary
- The sRGB standard uses gamma
 - Meaning pixels in a sRGB image file have had gamma correction applied

Gamma Correction

- 1) When we create an image using software, the pixels start as *linear values*
- 2) Before pixels are stored in a file or displayed we can gamma correct them

Store inputs of $V_{signal}^{1/\gamma}$

- 3) When displayed on a screen, the pixels should then appear as linear again

Linear values



0.0

0.25

0.5

0.75

1.0



Gamma corrected values

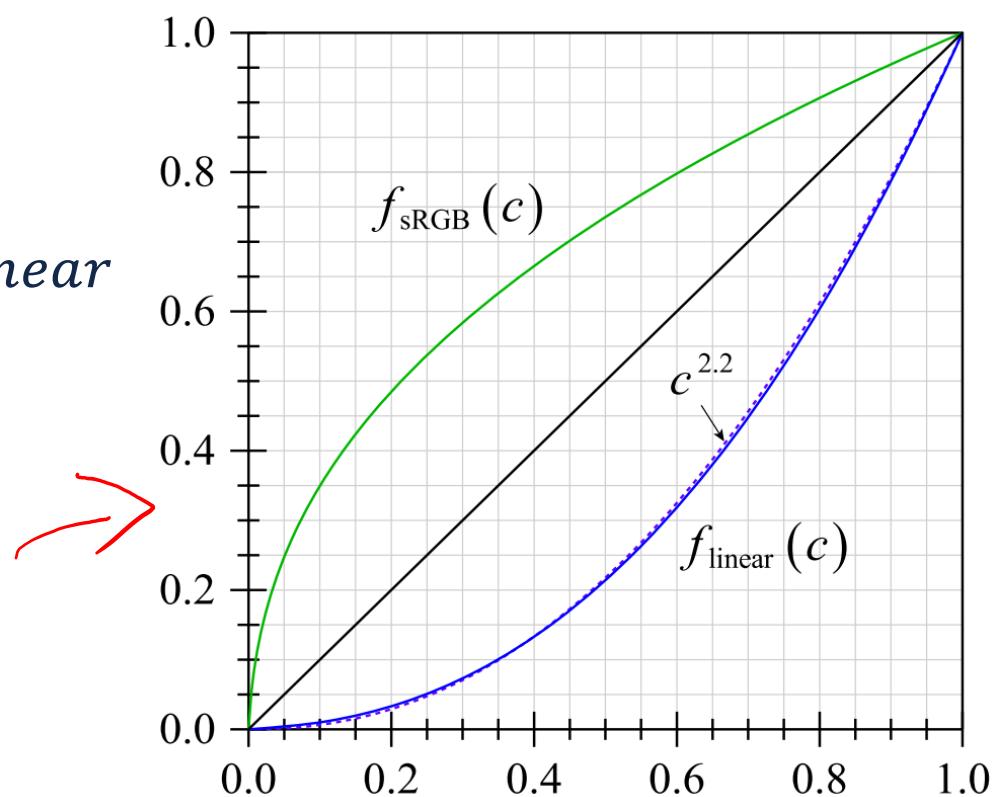
Gamma Correction in sRGB

An image stored in the sRGB format has gamma-corrected pixels

$$f_{\text{sRGB}}(c) = \begin{cases} 12.92c, & \text{if } c \leq 0.0031308; \\ 1.055c^{1/2.4} - 0.055, & \text{if } c > 0.0031308. \end{cases}$$

To decode an sRGB color you need to apply f_{linear}

$$f_{\text{linear}}(c) = \begin{cases} \frac{c}{12.92}, & \text{if } c \leq 0.04045; \\ \left(\frac{c + 0.055}{1.055}\right)^{2.4}, & \text{if } c > 0.04045. \end{cases}$$



Working with Gamma

You usually do not need to gamma correct an image you create

When saving as an image, the library code will apply gamma correction

- e.g. libpng will encode the image data according to the sRGB standard
- ...although you can specify an alternate gamma if you wish

When displaying an image you create in a browser...it's not clear what to do

- e.g. WebGL standard does not specify that colors should be gamma-corrected
- Gamma behavior could vary by browser and OS and GPU....
- Could create a control on the app to enable/adjust gamma manually

Gamma and Visualization

If you are using stored images that you will process computationally

- Need to remove the gamma correction before working with the pixels
- Most image processing operations work with linear colors
- Again, library code can likely be used to read image and linearize colors

If your visualization requires precise understanding of pixel intensities, be careful

- Make sure your color production is compatible with the intended display gamma
- Also allow display of actual numerical values or underlying data that the color represents

Gamma Compression

So...why is gamma still used by modern displays?

- Legacy images encoded with gamma correction...maybe?

Also useful when downsampling images from higher to lower bit-depth

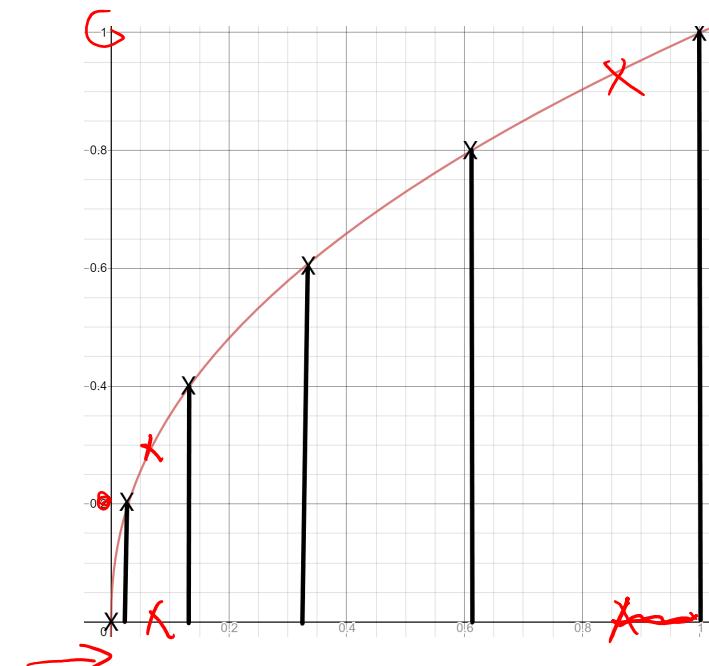
- Human vision has trouble differentiating bright intensities
- Implies that we should allocate more precision to lower intensities when downsampling

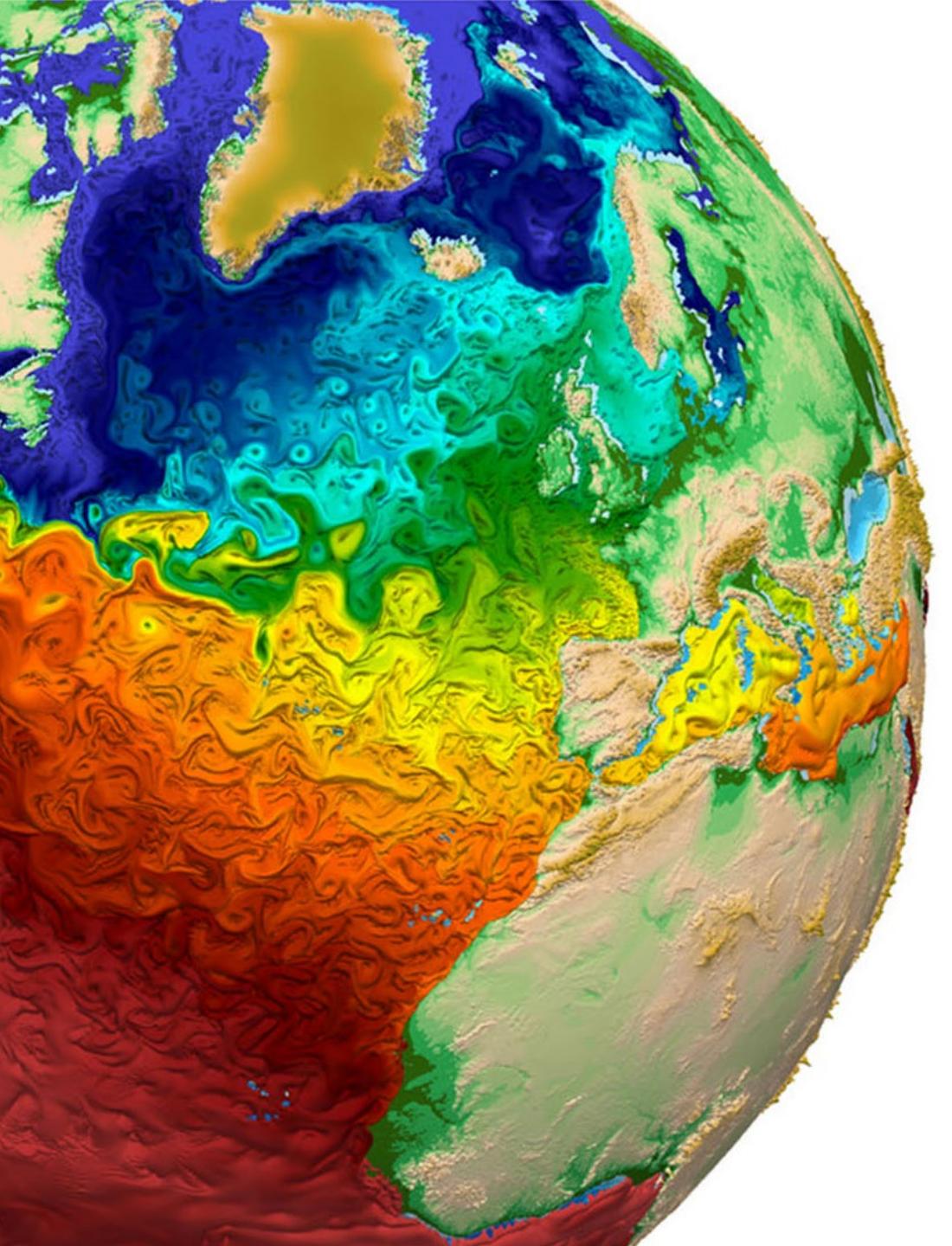
Gamma Compression: Example

Suppose we are downsampling

- The target space can represent only 6 values: 0, 0.2, 0.4, 0.6, 0.8, 1.0
- Procedure is
 - gamma correct raw value x
 - then round to closest value of the 6 we can represent
- in the graph on the right, raw values are on the x axis
- the y axis shows the gamma corrected values
- gamma corrected values get rounded to a representative

...we can drop lines from representatives down to the x axis and see that we are more densely sampling darker values





3D Computer Graphics for People in a Hurry

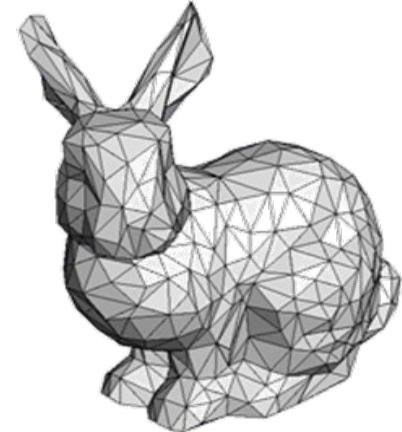
Rendering

Scientific Visualization
Professor Eric Shaffer

Visualization and Computer Graphics

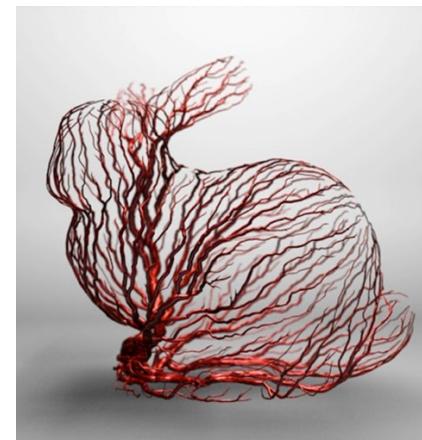
Visualization and Computer Graphics are not the same thing

- Visualization uses computer graphics technology as a tool
- You can create great visualizations without a deep knowledge of CG



Need understand what elements of CG that impact visualization

- What impacts application performance?
 - Important for interactivity
- What impacts the visual quality of the rendered image?
 - Avoid distorting the image in misleading ways



3D CG is important for sci vis because we often want to see 3D physical domains

Rendering

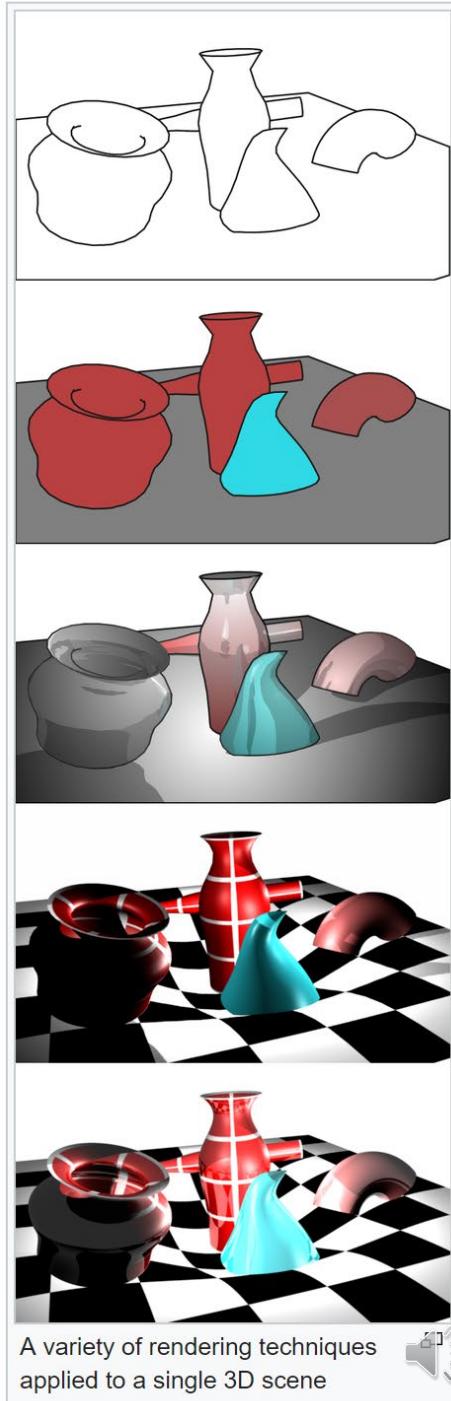
Rendering or image synthesis is the automatic process of generating a photorealistic or non-photorealistic image from a 2D or 3D model (or models in what collectively could be called a scene file) by means of computer programs.

Wikipedia

What is the same about each image at the right?

What is different?

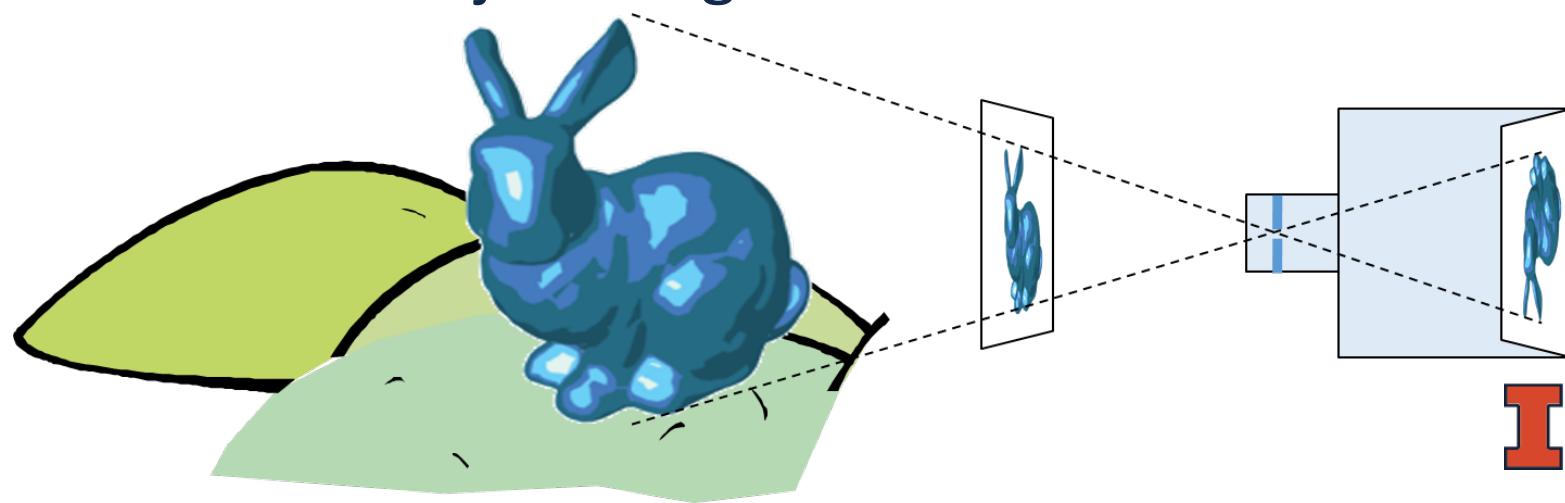
What technology enables this change in modern real-time graphics?



A variety of rendering techniques applied to a single 3D scene

3D Graphics: Image Formation

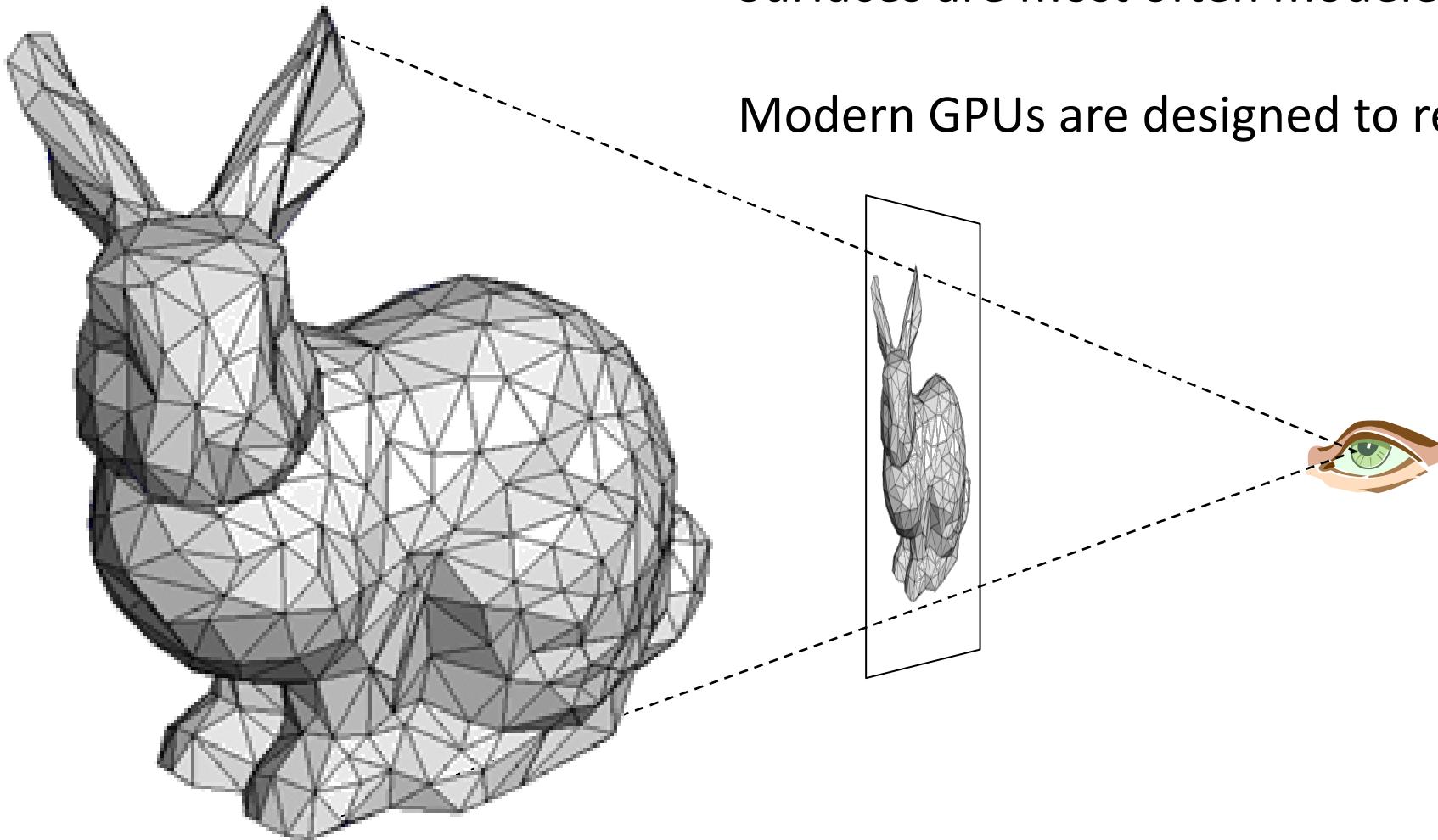
- Goal in CG (usually) is to generate a 2D image of a 3D scene...
 - The input data is a scene description
 - Output is an image
- To achieve this we computationally mimic a camera or human eye
- In the scene...there are objects...lights...and a viewer



Polygonal Models

Surfaces are most often modeled using triangles

Modern GPUs are designed to render triangles



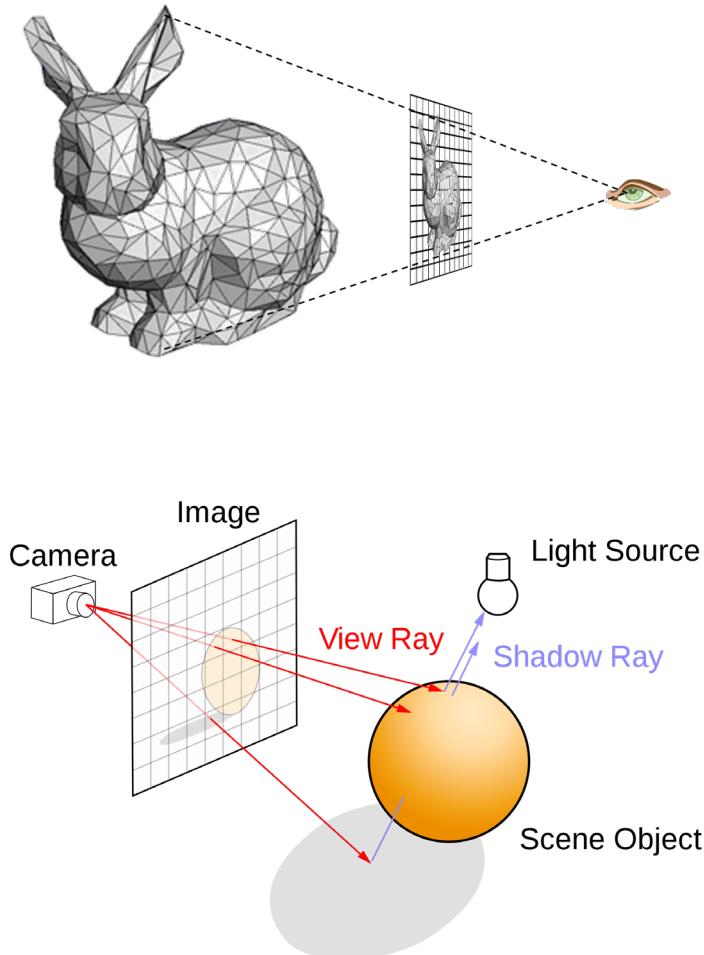


Rendering generally uses one of two approaches

- Rasterization
- Ray tracing
- Sometimes both....
- ...and there are other methods like radiosity

Rasterization versus Ray Tracing

- To oversimplify....
- In rasterization, geometric primitives are projected onto an image plane and the rasterizer figures out which pixels get filled.
- In ray-tracing, we model the physical transport of light by shooting a sampling ray through each pixel in an image plane and seeing what the ray hits in the scene



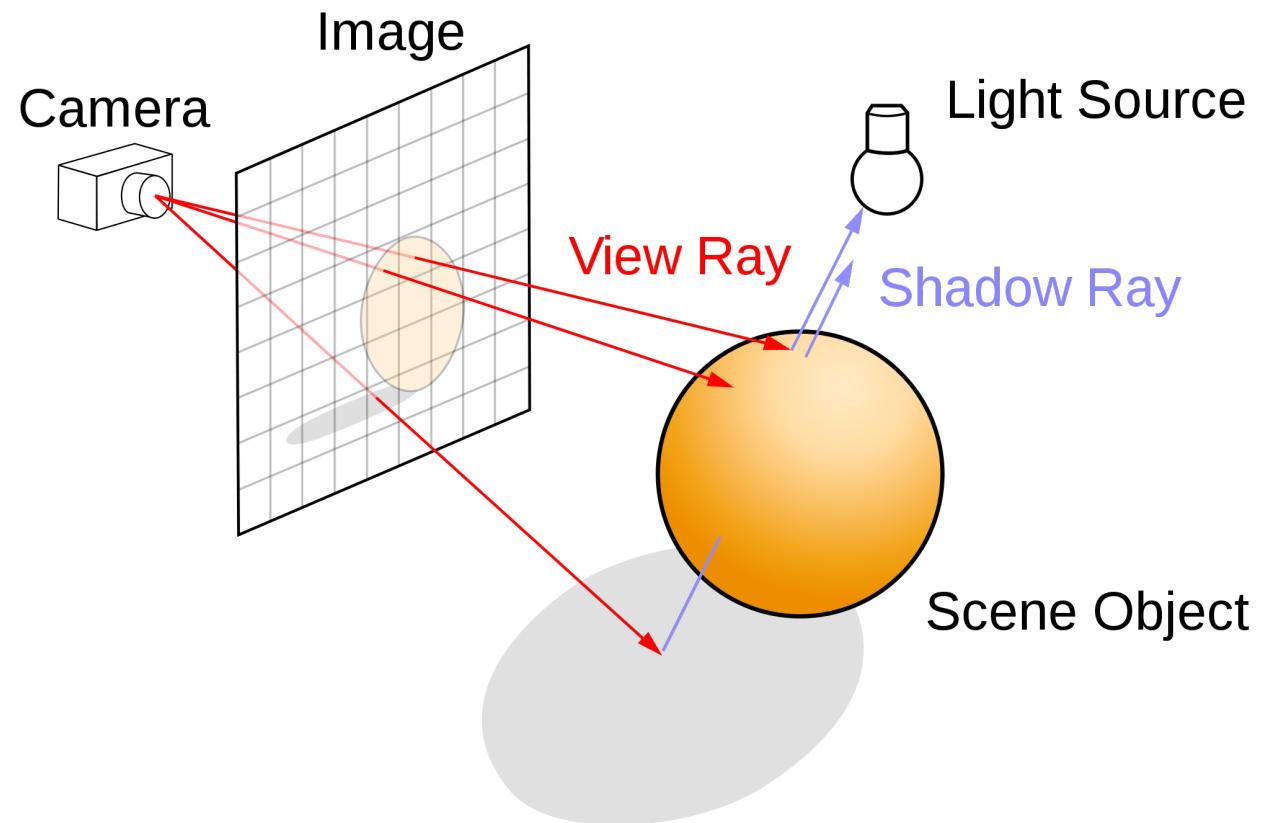
Ray Tracing

Follow ray of light....

Can trace from an eyepoint through a pixel

See what object the ray hits...

How would you check to see if the object is lit or in shadow?



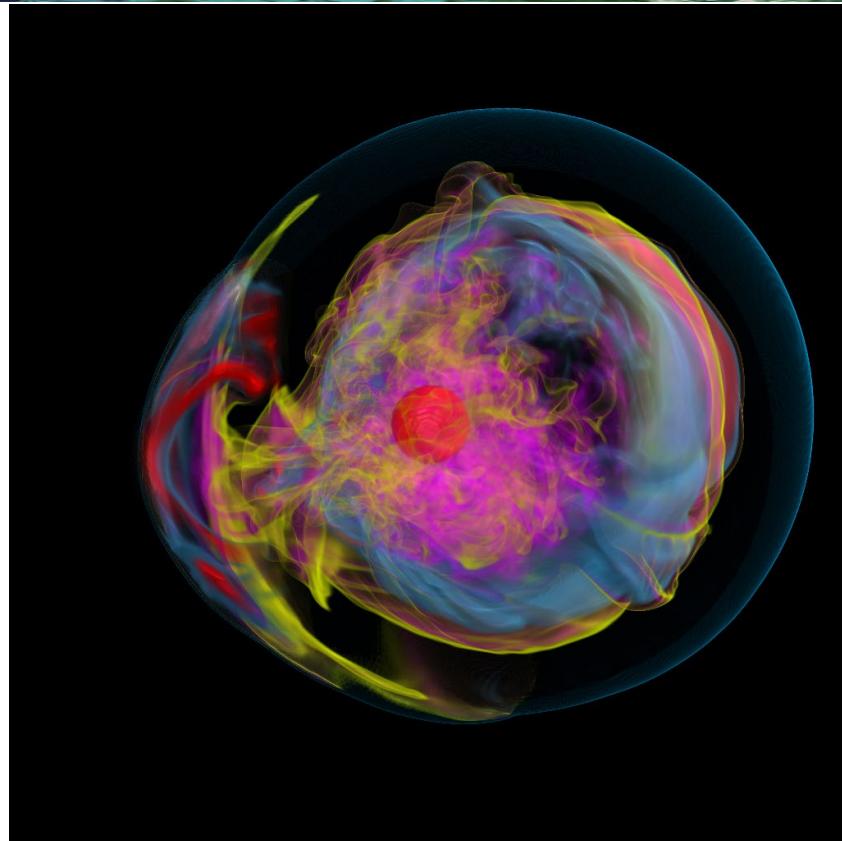
Ray Tracing in Visualization

Ray Tracing is used in visualization to generate semi-transparent views of volumes

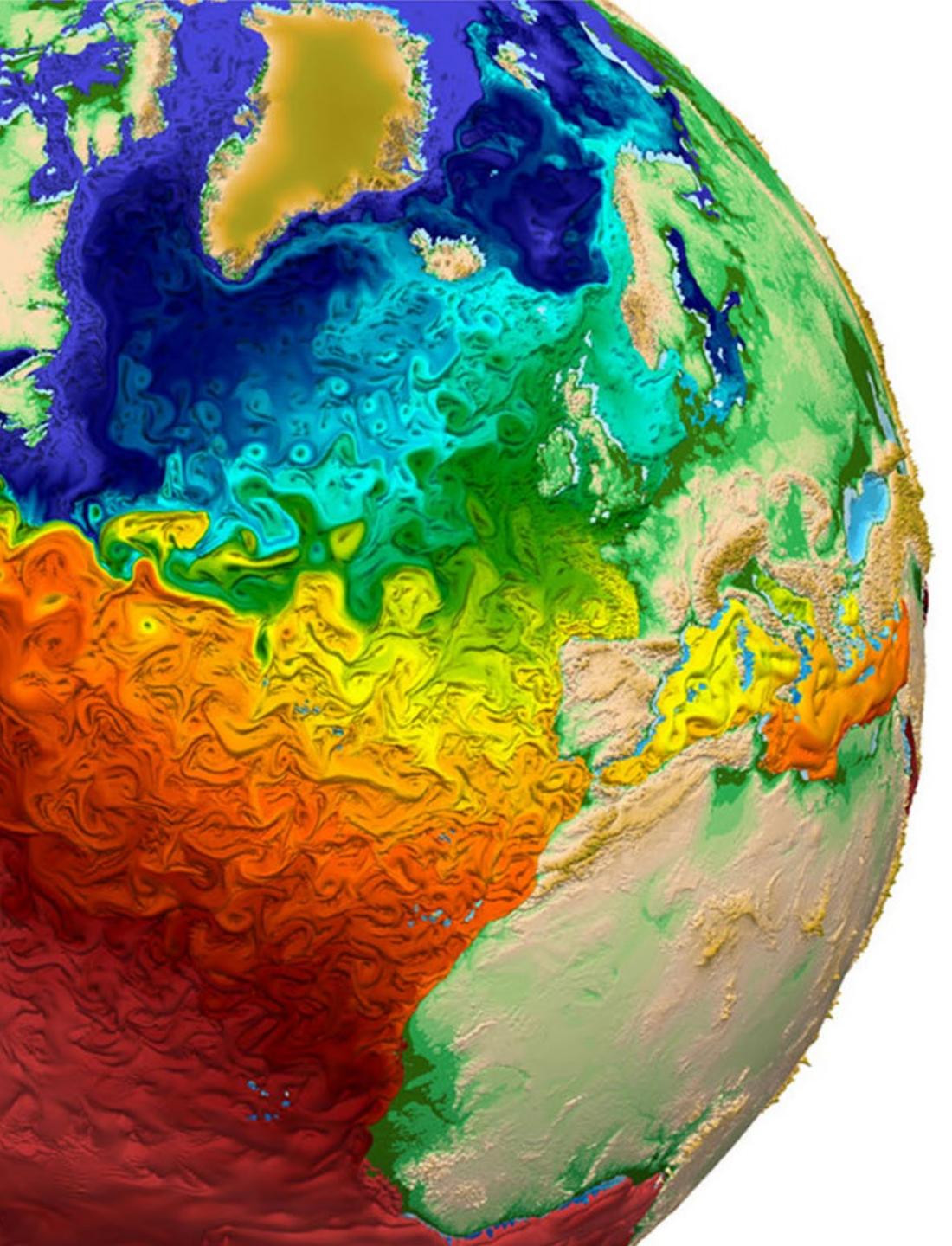
- Rays can sample volumes

Rasterization has difficulty doing this

- Designed for surface rendering



By Kwan-Liu Ma - <https://web.cs.ucdavis.edu/~ma/>, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=88523656>

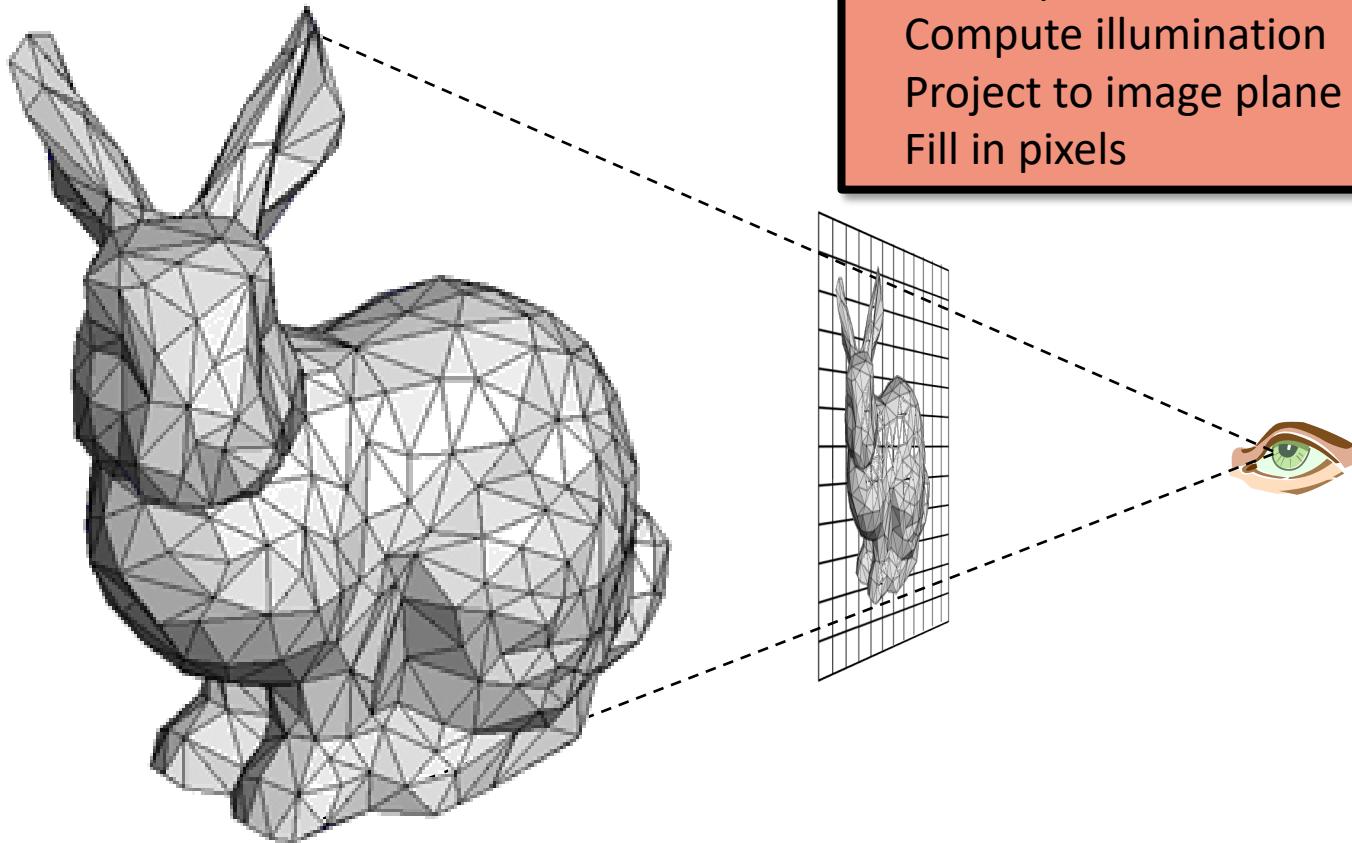


3D Computer Graphics for People in a Hurry

Rasterization Pipeline

Professor Eric Shaffer

Rasterization



Definitions: Pixel and Raster

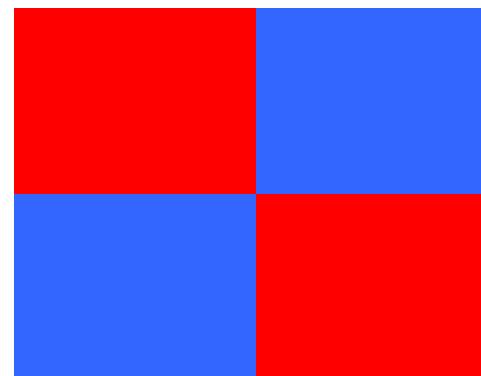
A *pixel* is the smallest controllable picture element in an image

A *raster* is a grid of pixel values

Typically rectangular grid of color values

(1.0, 0.0, 0.0), (0.0, 0.0, 1.0)

(0.0, 0.0, 1.0), (1.0, 0.0, 0.0)



RGB Color Representation

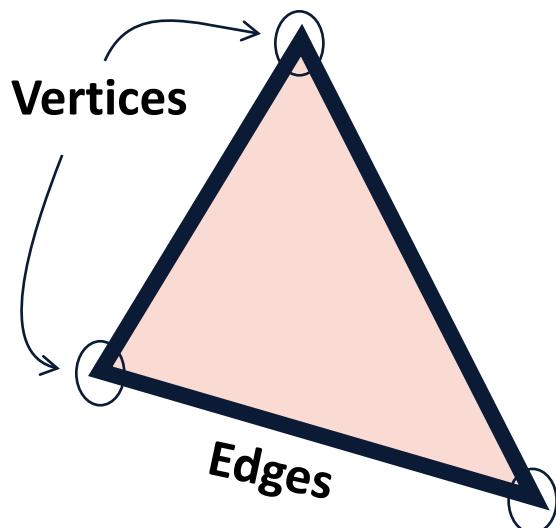
A color is a triple (R,G,B) representing a mix of red, green, and blue light.

Each color channel has a value in [0, 1] indicating how much light is emitted.



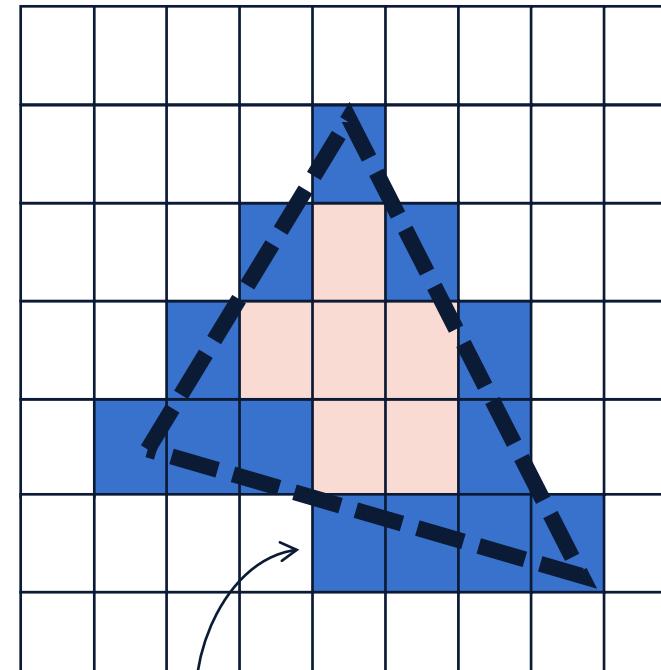
Rasterization

Primitives



Generate a raster image
from a vector description

Pixels



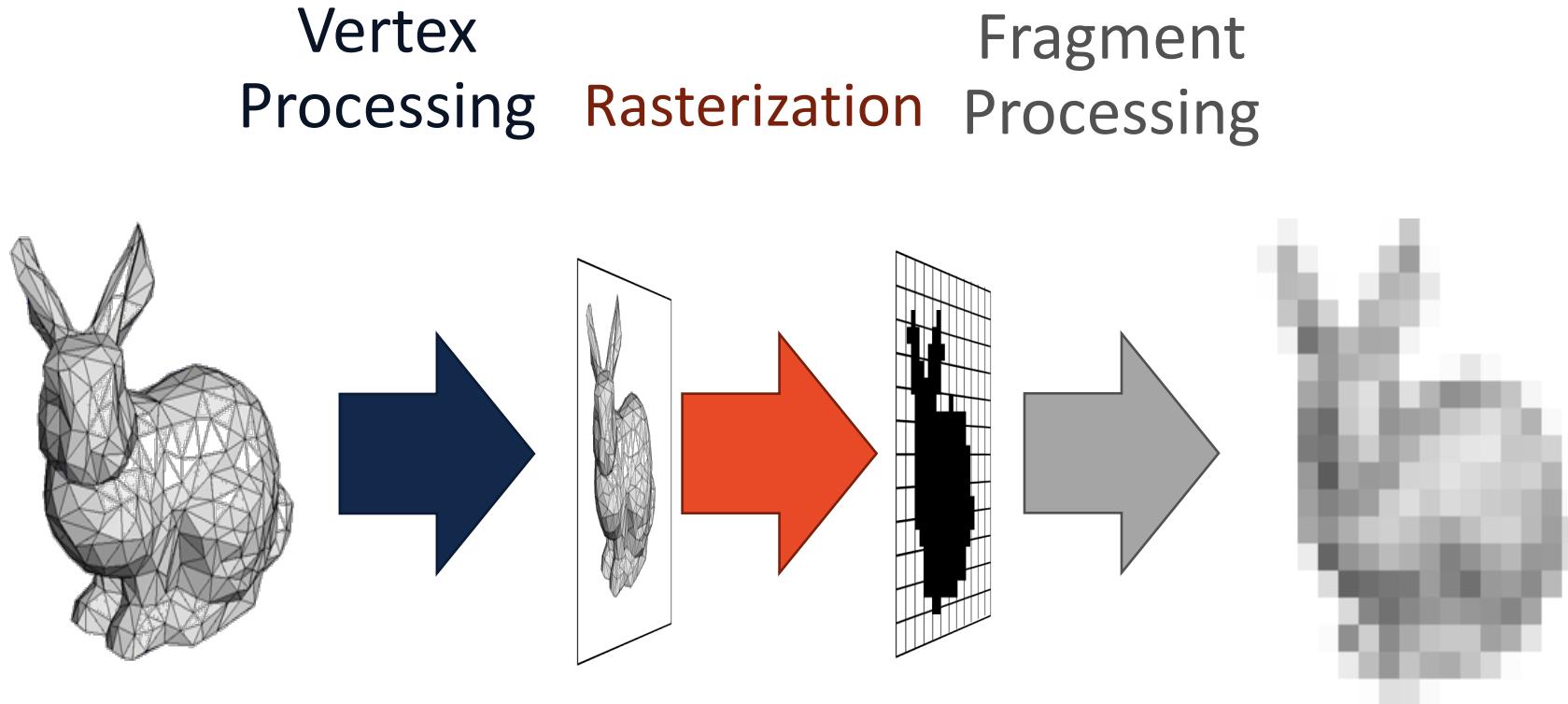
Aliasing

Vector Graphics Representation

Is a purely mathematical representation of shape. For example, a line is $y=mx+b$. Typically, **vector graphics** refers to 2D shapes, but the idea applies to 3D as well.



3D Graphics Pipeline



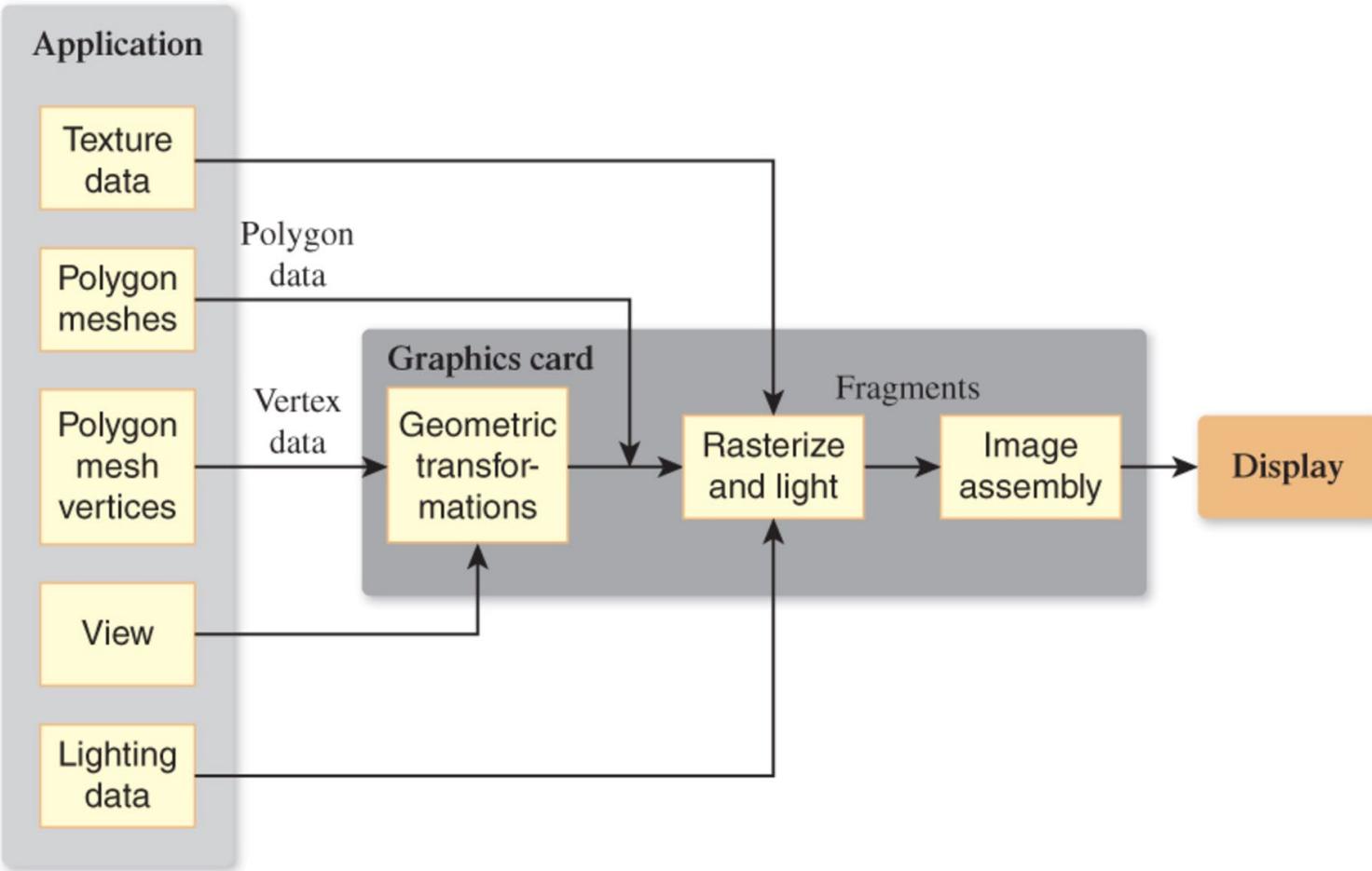
Fragments

Are like pixels...but they aren't necessarily the finalized pixels you see in an image. Each fragment has a 2D location in a raster and a color.

Final pixel value is typically found by applying *hidden surface removal* and possibly *compositing* to a set of fragments.



Rasterization is a Pipeline

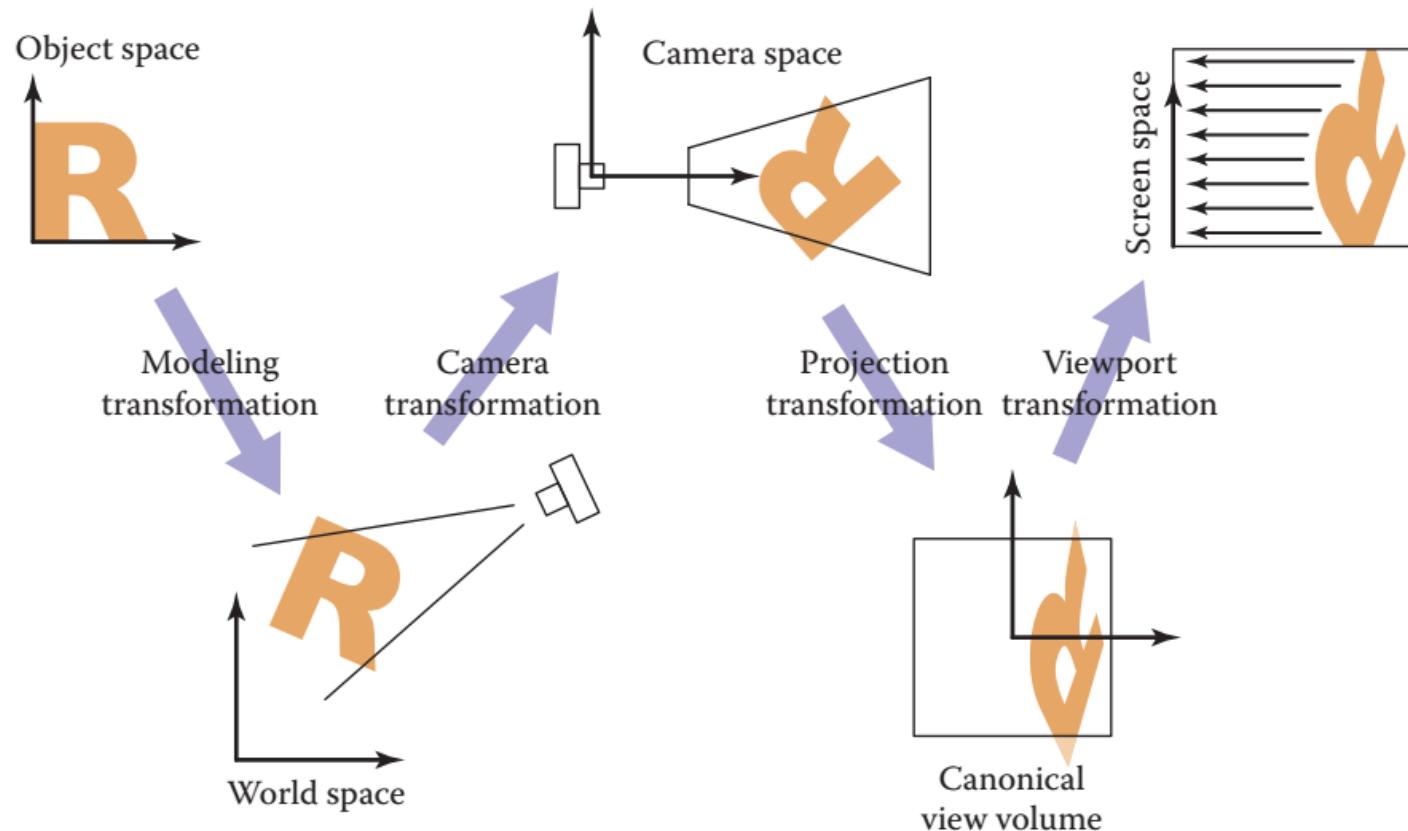


- Data for objects in the scene usually in the form of polygonal meshes
- Most of the work to render an image is done on the Graphics Processing Unit (GPU)
- GPU code will have at least two parts
 - **Vertex Shader**
 - **Fragment Shader**

Vertex Shader

- Program that runs on the GPU
- Typically transforms vertex locations from one coordinate system to another
 - Transformations can be useful for placing objects in your scene
 - Also, some operations on the geometry are easier when done in specific coordinate system
- Change of coordinates usually equivalent to a matrix transformation
- Vertex shader can also compute vertex colors

Changing Coordinate Systems



Model Transformation:

Move a model from a local coordinate system to a position in the “world”

Camera Transformation:

Places camera at the origin and moves the objects in the world using the same transformation

Projection Transformation:

Change coordinates so that a 3D to 2D projection of the geometry is done correctly

Viewport Transformation:

Change from 2D coordinates in $[-1,1]$ to pixel coordinates

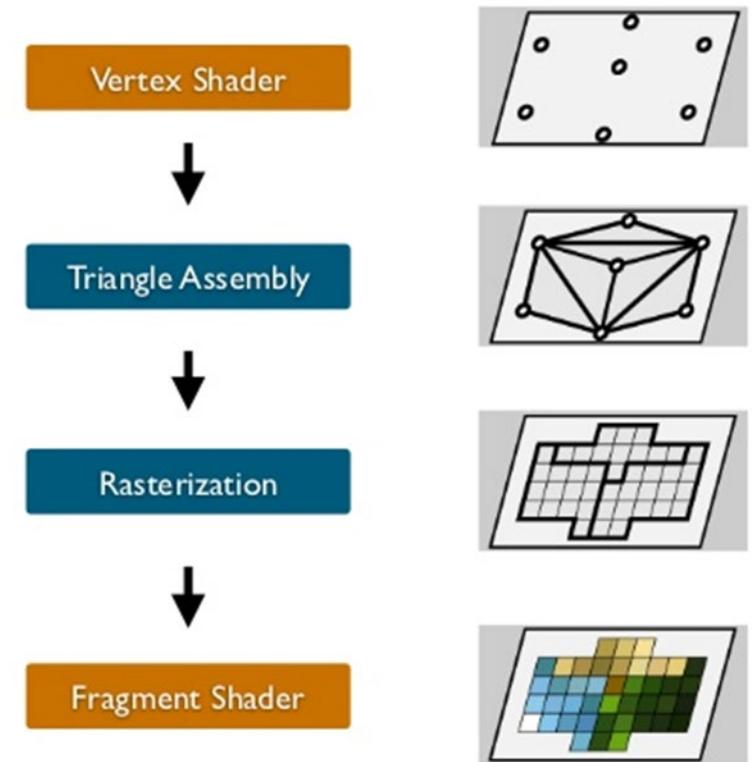
Rasterization

Rasterizer produces a set of fragments for each triangle

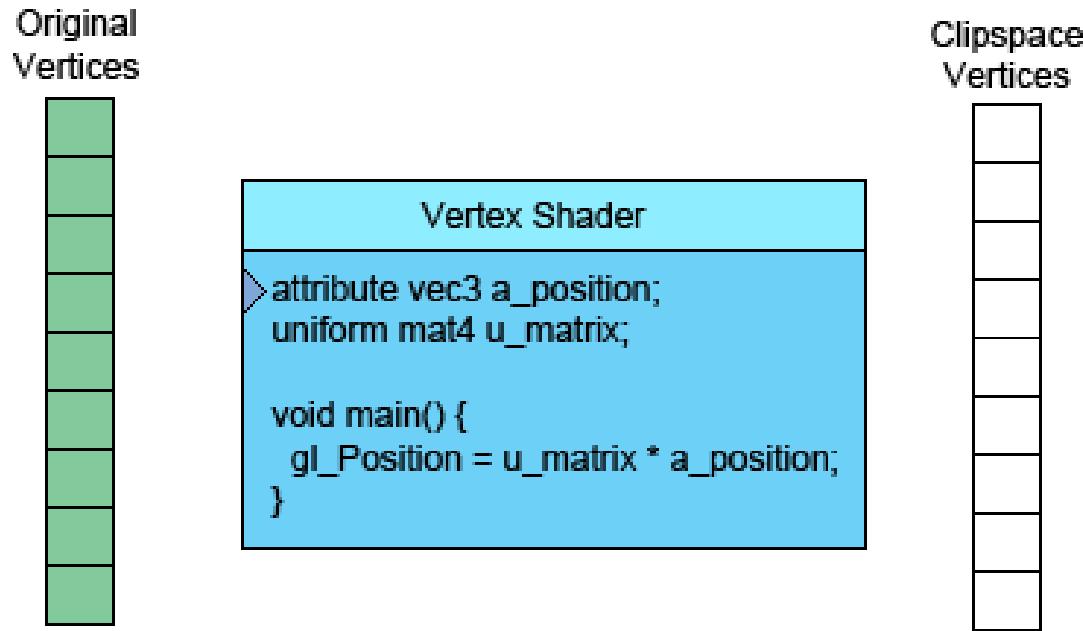
Fragments are “potential pixels”

- Have a location in frame buffer
- Color and depth attributes

Vertex attributes are interpolated across fragments



What a Vertex Shader Does...

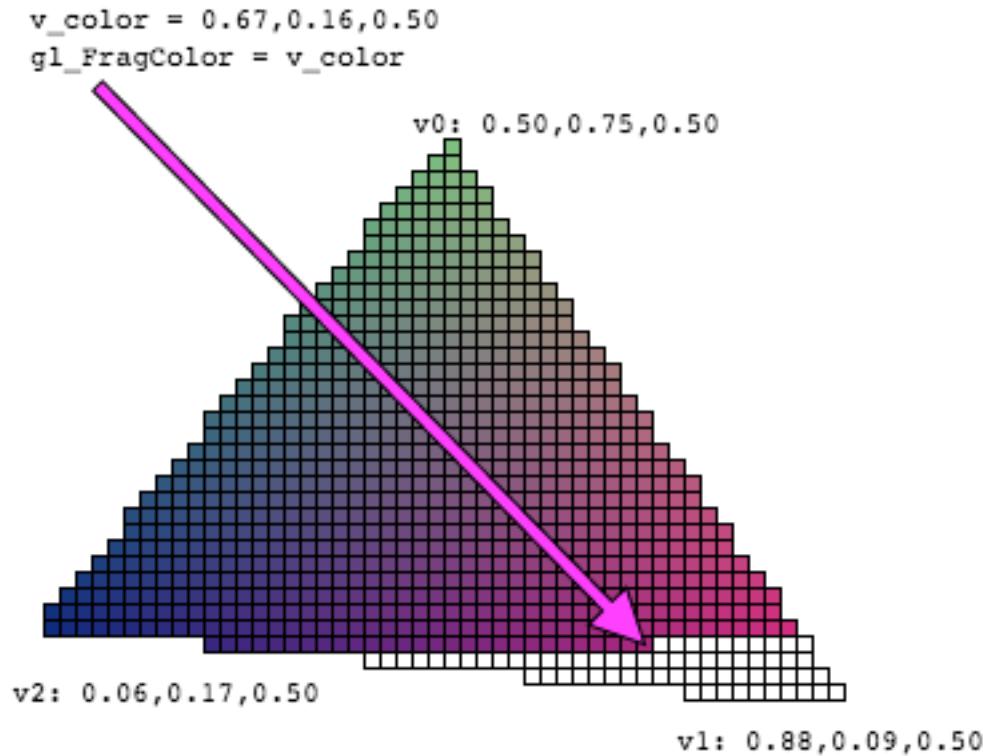


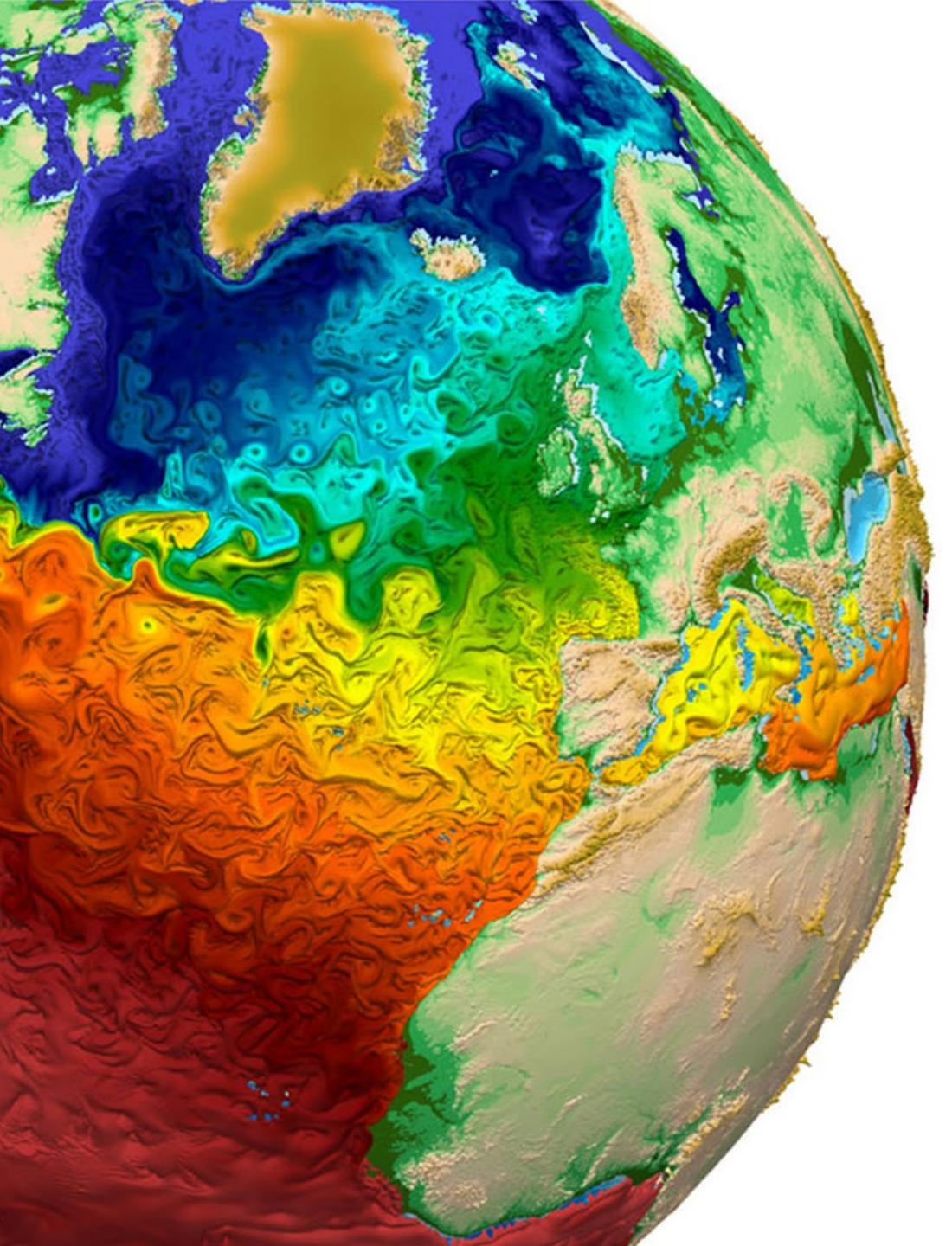
Taken from webglfundamentals.org

Can you guess what is slightly incorrect about this animation?



What a Fragment Shader Does...





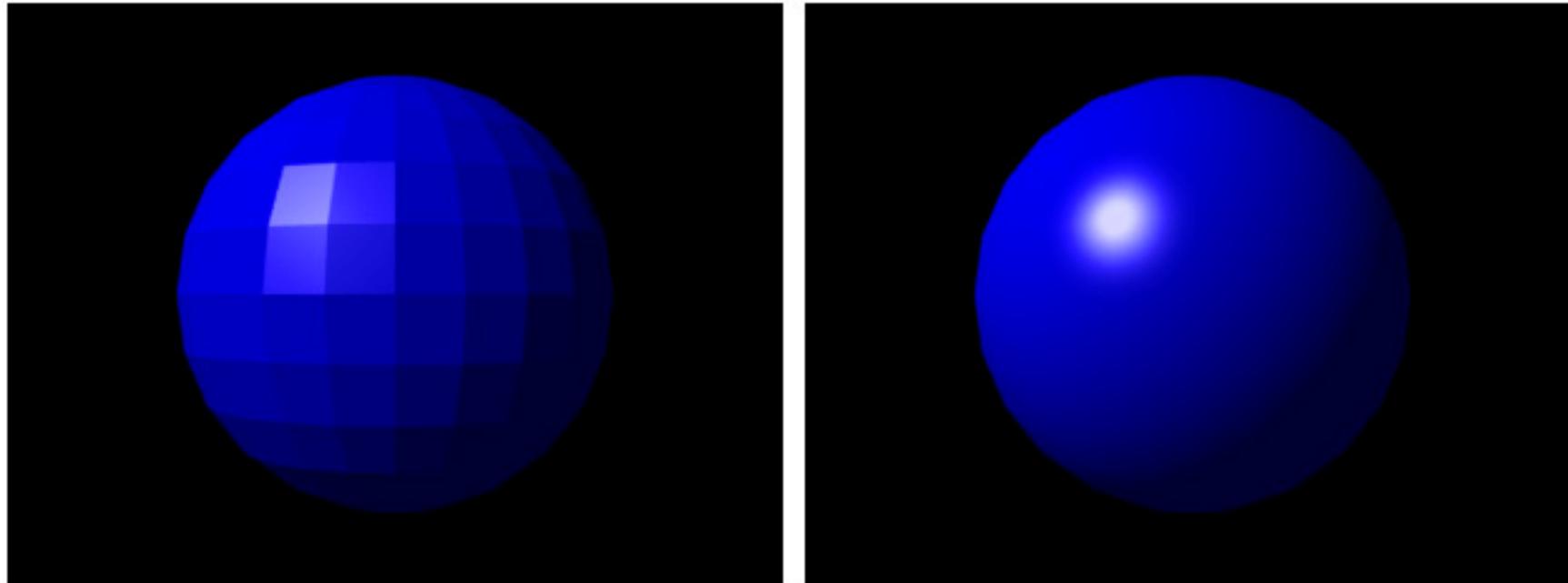
3D Computer Graphics for People in a Hurry

Shading

Professor Eric Shaffer

Shading

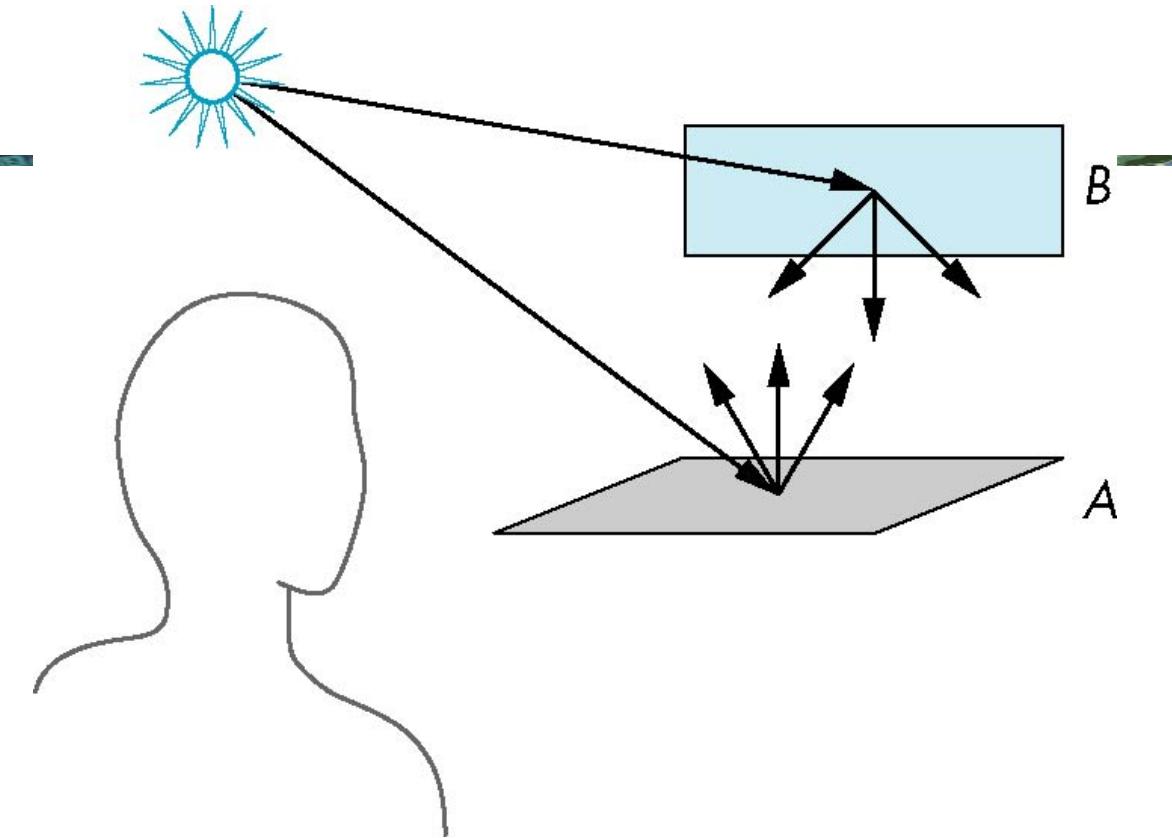
Shading refers to the process of determining the color for a pixel (or vertex...or polygon) during the rendering process



What is the difference between the two images?

Scattering

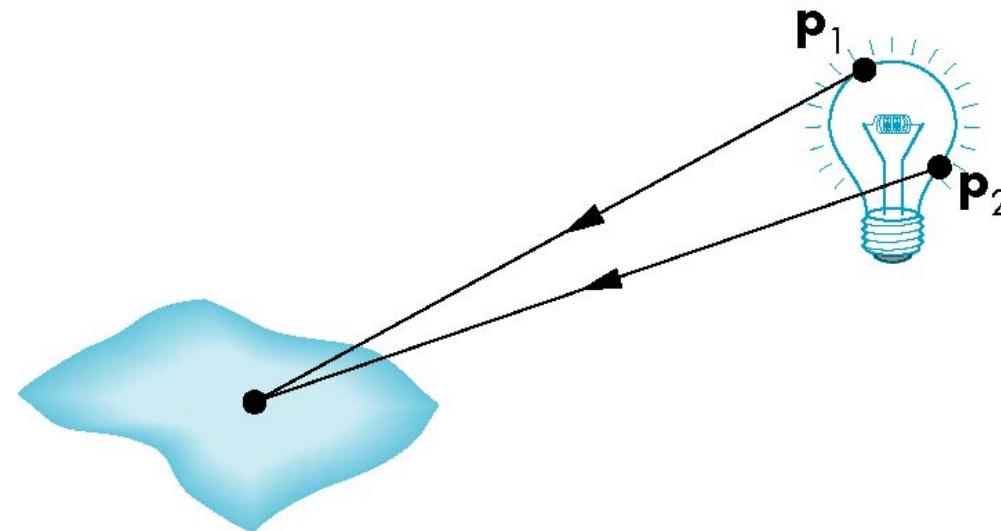
- Light strikes A
 - Some scattered
 - Some absorbed
- Some of scattered light strikes B
 - Some scattered
 - Some absorbed
- Some of this scattered light strikes A and so on



Light Sources

General light sources are complex to model

Would need to integrate light coming from all points on the source

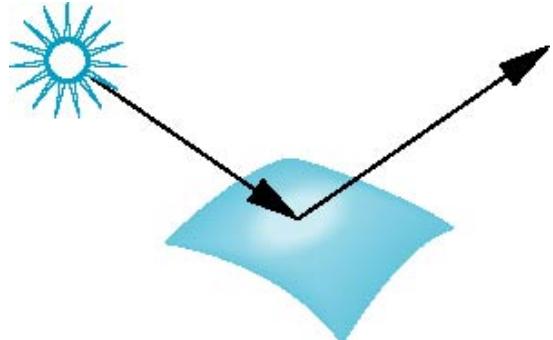


Simple Light Source Models

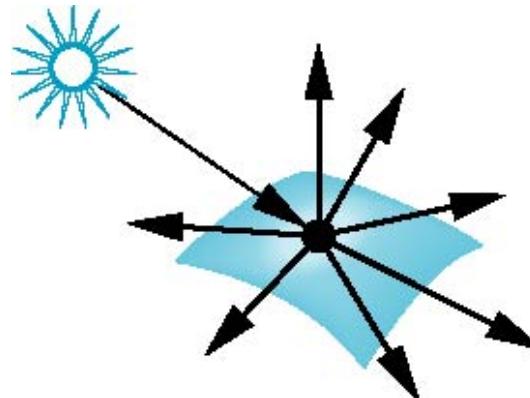
- Point source
 - Model with position and color
- Directional source
 - Distant source = infinite distance away (parallel)
- Ambient light
 - Same amount of light everywhere in scene
 - Can model contribution of many sources and reflecting surfaces

Surface Types

- Consider light traveling along a specific ray
- The smoother a surface, the more reflected light is concentrated in a single direction
 - Perfect mirror reflects perfectly in a single direction
- A very rough surface scatters light in all directions



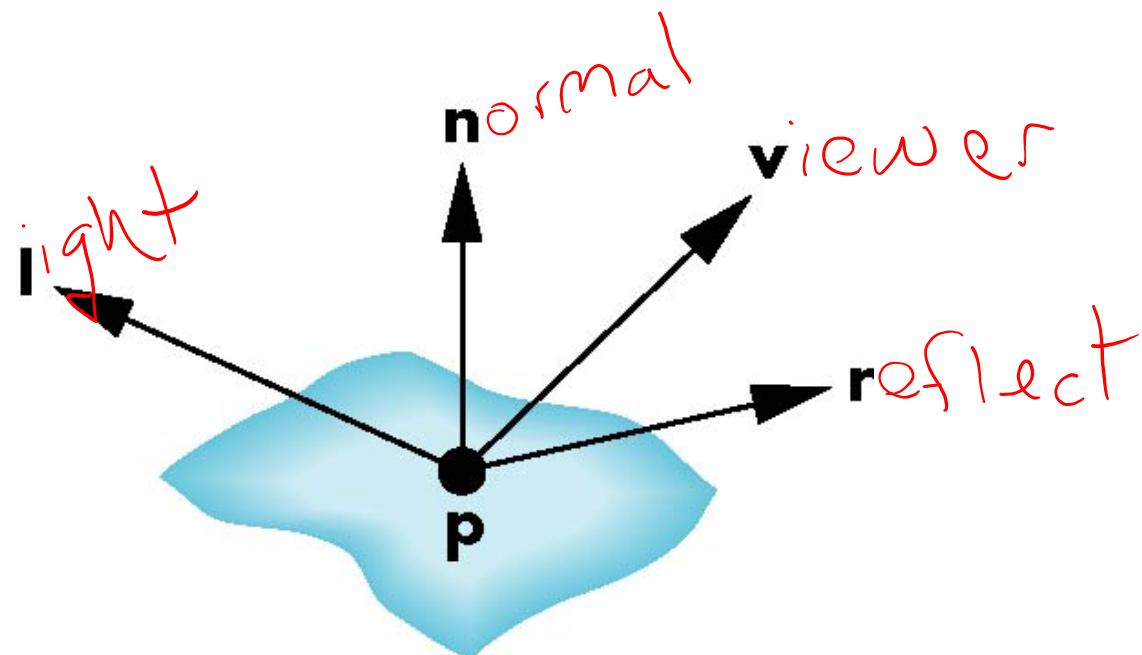
smooth surface



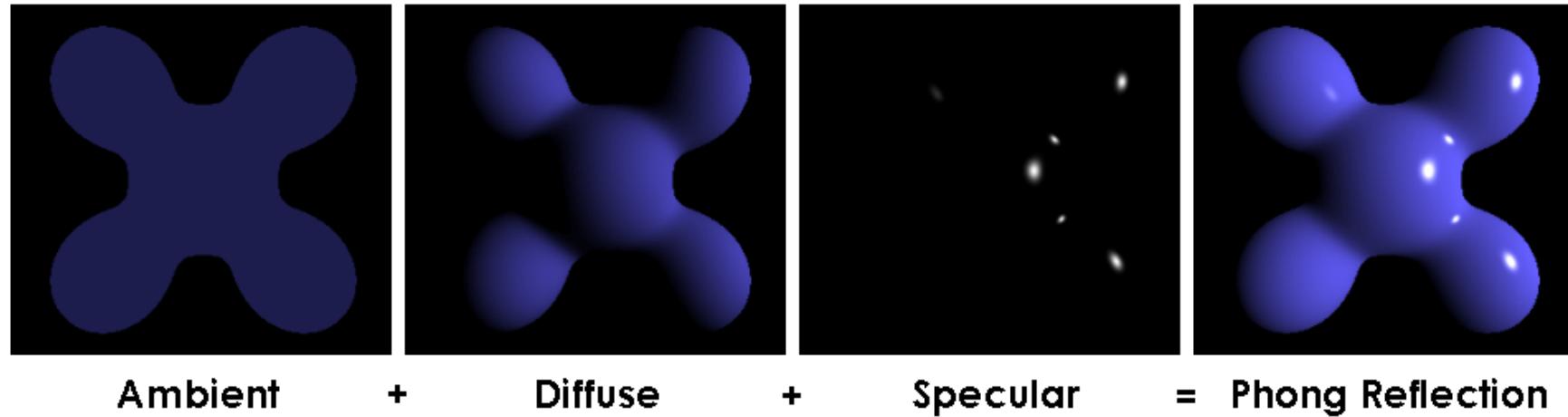
rough surface

The Phong Reflection Model

- A simple model that can be computed rapidly
- Has three components
 - Diffuse
 - Specular
 - Ambient
- Uses four vectors
 - To light
 - To viewer
 - Normal
 - Perfect reflector



Phong Reflectance Model

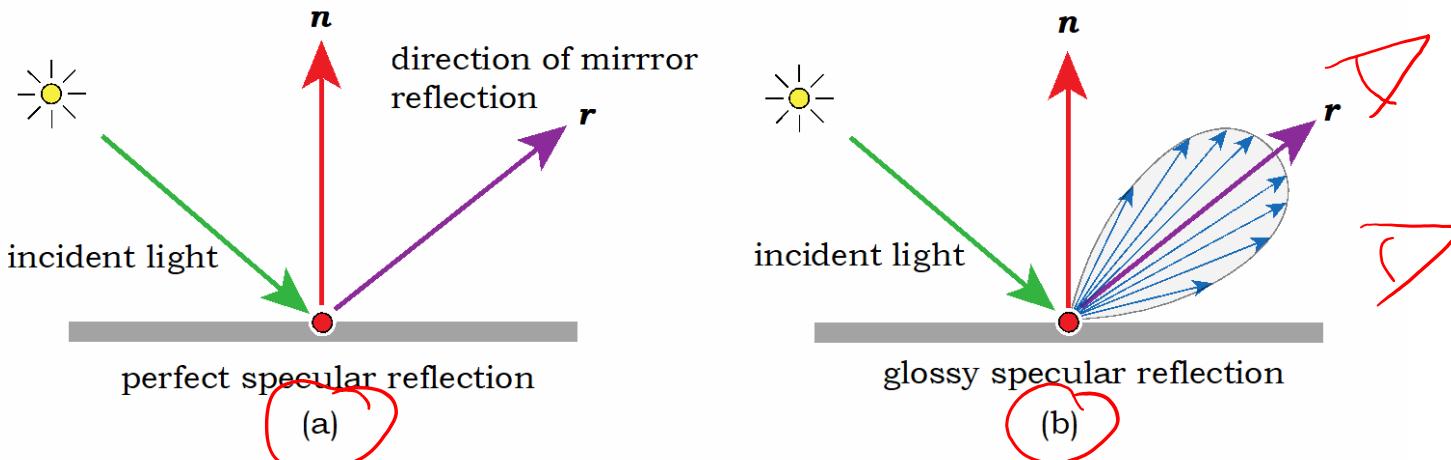
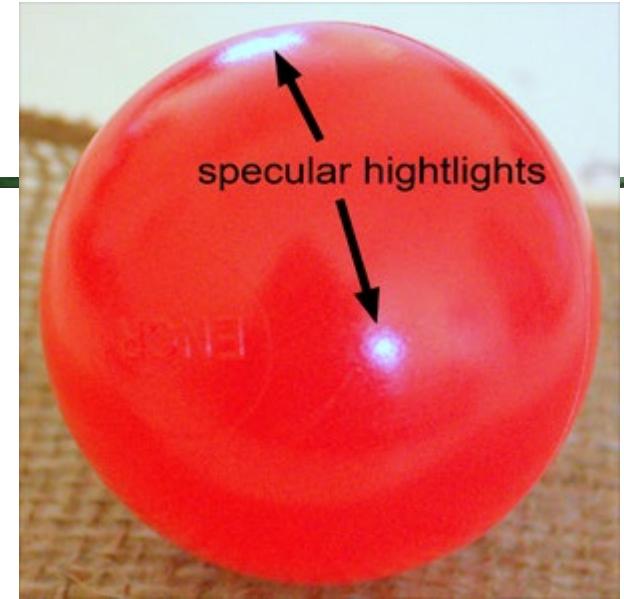


$$I_p = k_a i_a + \sum_{m \in \text{lights}} (k_d (\hat{L}_m \cdot \hat{N}) i_{m,d} + k_s (\hat{R}_m \cdot \hat{V})^\alpha i_{m,s})$$

$(1, 0, 0) \cdot (1/4, 1/4, 1/4) = (1/4, 0, 0)$

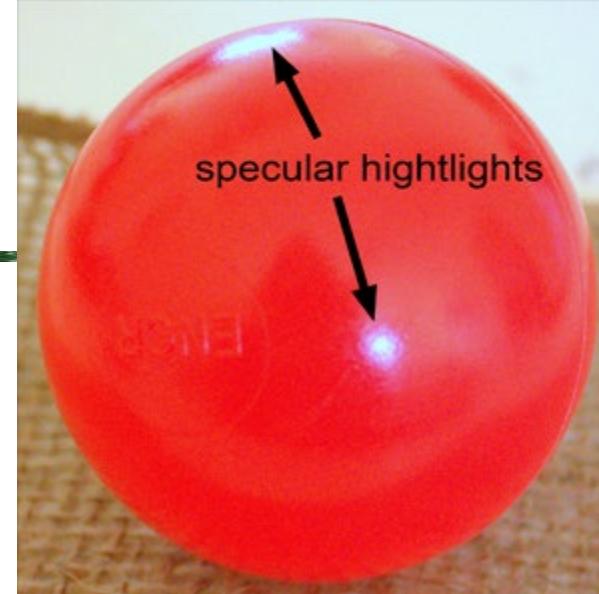
Specular Reflection

- Perfect specular reflection
 - Light is reflected in the single direction r
 - ...the mirror reflection direction
- Glossy specular reflection
 - Scattering clustered around mirror reflection direction



Specular Reflection

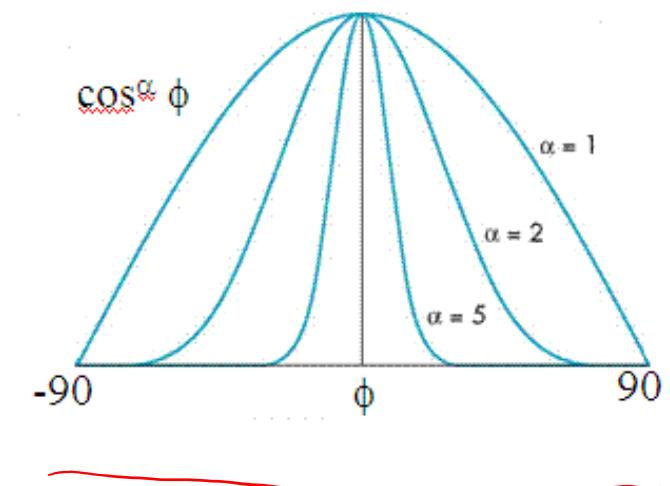
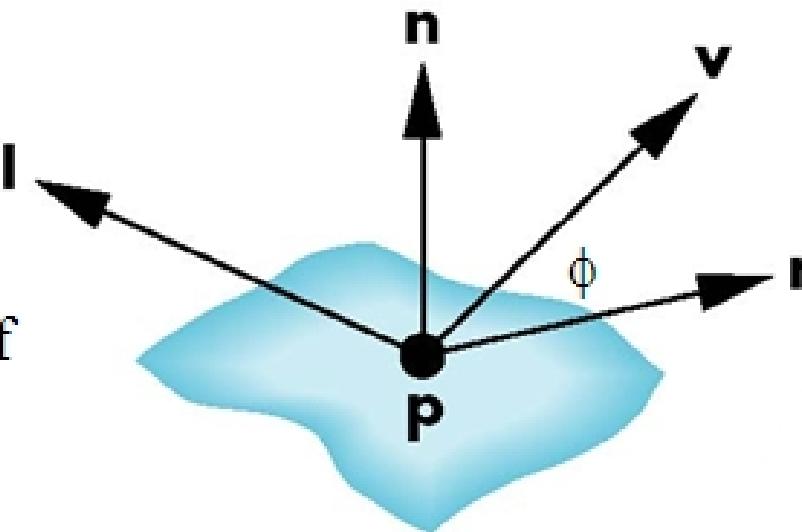
- Reflectance determined by
 - Alignment of view vector with mirror reflection vector
 - Shininess coefficient
- High coefficient means smoother look
 - Maybe 100 for metal
 - Maybe 10 for plastic



$$I_r \sim k_s I \cos^\alpha \phi$$

reflected intensity incoming intensity shininess coef
absorption coef

A hand-drawn diagram illustrating the formula for specular reflection. It shows three red arrows originating from a point labeled I . One arrow points upwards and is labeled "reflected intensity". Another arrow points downwards and is labeled "incoming intensity". A third arrow points to the right and is labeled "shininess coef". The formula $I_r \sim k_s I \cos^\alpha \phi$ is written above the arrows, with a red circle highlighting the term $\cos^\alpha \phi$.

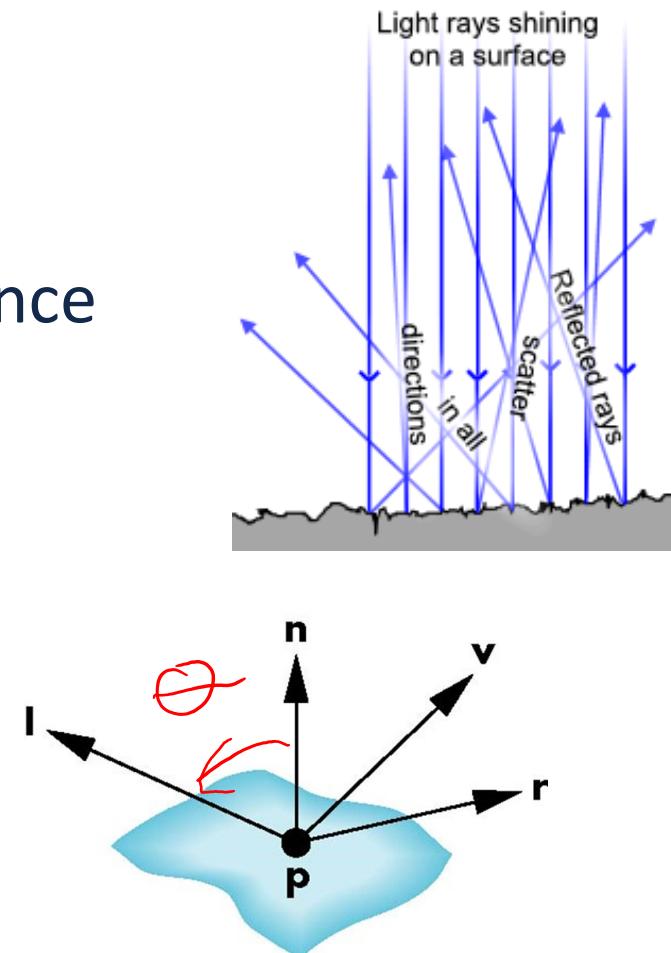


Modeling a Lambertian Surface – Diffuse Reflection

- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is affected by the angle of incidence
 - reflected light proportional to ***cosine of angle between l and n***
 - if vectors normalized

$$\cos(\theta) = \mathbf{n} \cdot \mathbf{l}$$

- Amount of reflected light also affected by k_d and i_d
 - Each is an rgb value with each channel in [0,1]

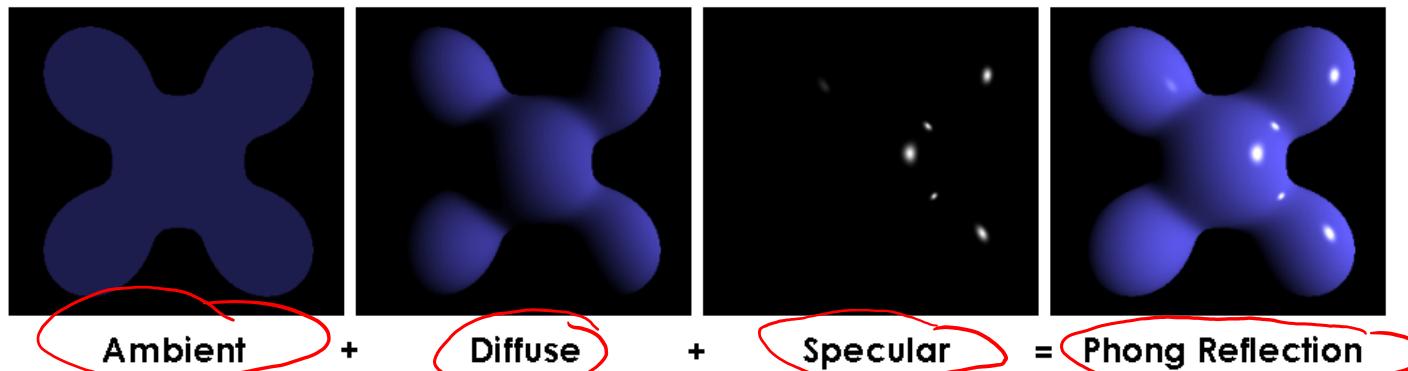


Ambient Light

- Result of multiple interactions between light sources and surfaces
- Amount and color depend on the color of the light(s) and the material properties
- Add $k_a I_a$ to diffuse and specular terms

reflection intensity of ambient light

Remember that k_i multiplications are component-wise multiplications of rgb values
 $(k_r, k_g, k_b)(i_r, i_g, i_b) = (k_r i_r, k_g i_g, k_b i_b)$

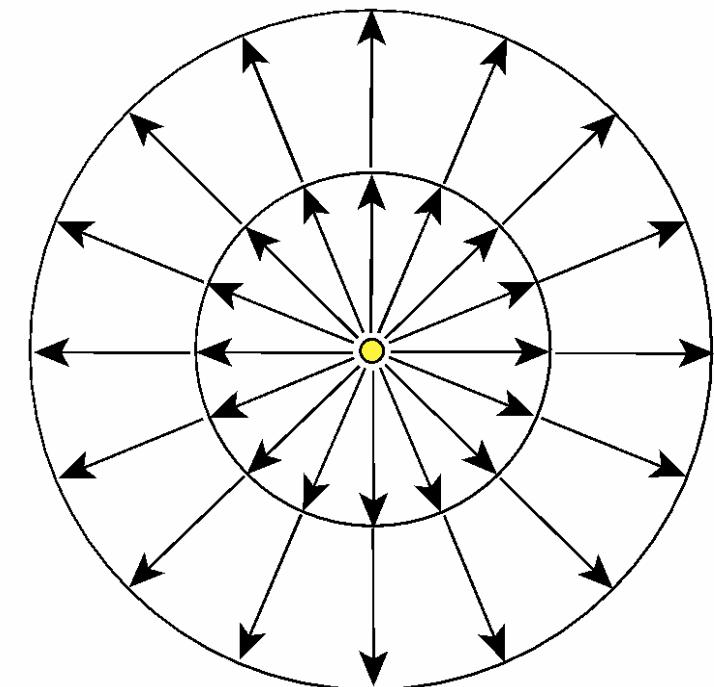


Distance Terms

- The light from a point source that reaches a surface is *attenuated*
 - Intensity falls off with the square of the distance
- We can apply a factor to the diffuse and specular terms

$$\frac{1}{ad^2 + bd + c}$$

- **d** is the distance from the light to surface
- **a,b,c** are constants you choose to get different effects



Blinn-Phong Reflectance Model

- Jim Blinn suggested an approximating changing specular term
 - Replace $(V \cdot R)^a$ by $(N \cdot H)^b$ where
 - “Halfway vector”
- More efficient in terms of the operations used
- Closer to physically correct lighting
- Pick exponent b to match what you want
 - Using higher $b > a$ will make output similar to Phong with a

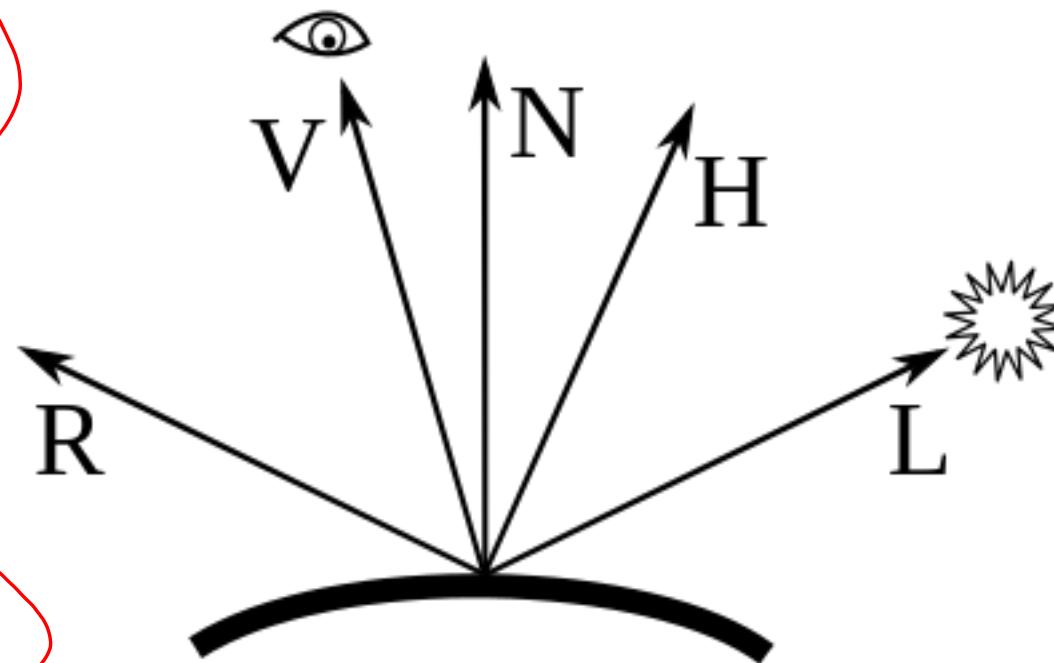
$$H = \frac{L + V}{\|L + V\|}$$

The Halfway Vector

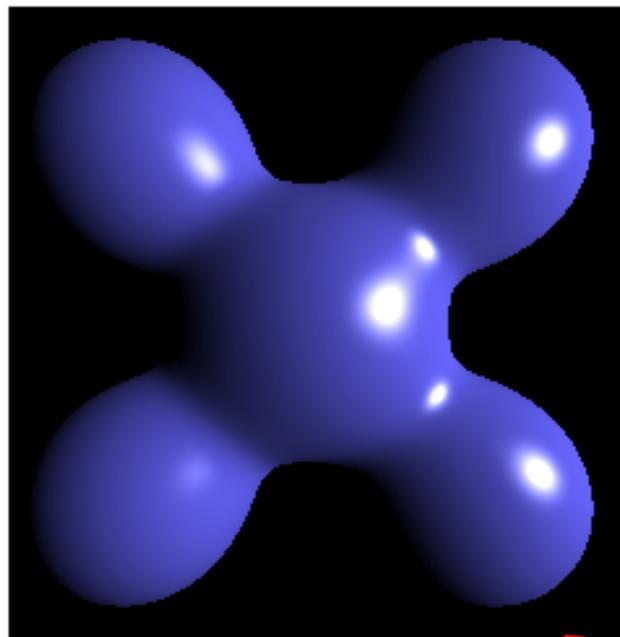
H is normalized vector halfway between L and V

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

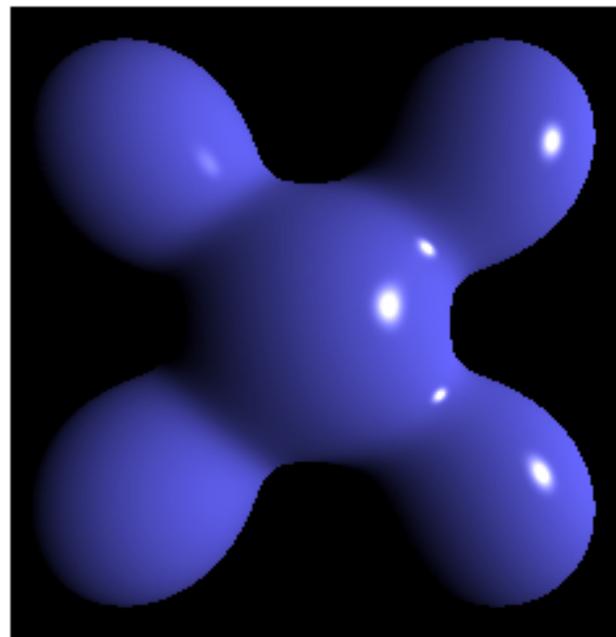
$$r = 2(l \cdot n)n - l$$



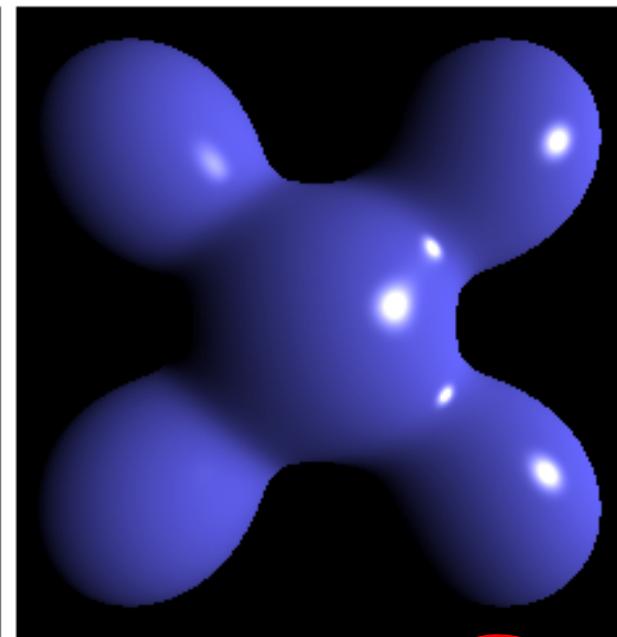
Phong versus Blinn-Phong



Blinn-Phong



Phong



Blinn-Phong
(higher exponent)

Gouraud and Phong Shading

- **Gouraud Shading**
 - Find average normal at each vertex
 - Compute shade at each vertex
 - Interpolate vertex shades across each polygon
- **Phong shading**
 - Find average normal at each vertex
 - Interpolate vertex normals across edges
 - Interpolate edge normals across polygon
 - Compute shade at each fragment



X

Phong Shading is NOT THE
SAME as the Phong
reflectance model

Bui Tuong Phong

- December 14, 1942 – July 1975
- Born in Hanoi
- Earned his PhD in 2 years at the University of Utah (1973)
 - Worked with Professor Ivan Sutherland
 - Dissertation work was the Phong reflectance model
 - Also produced model and realistic image of a VW bug

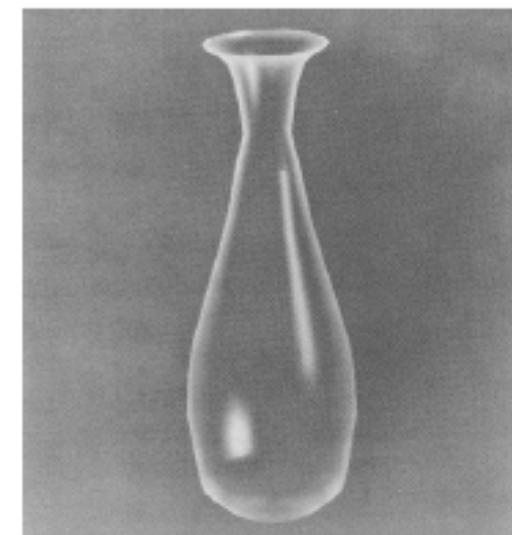
Graphics and
Image Processing

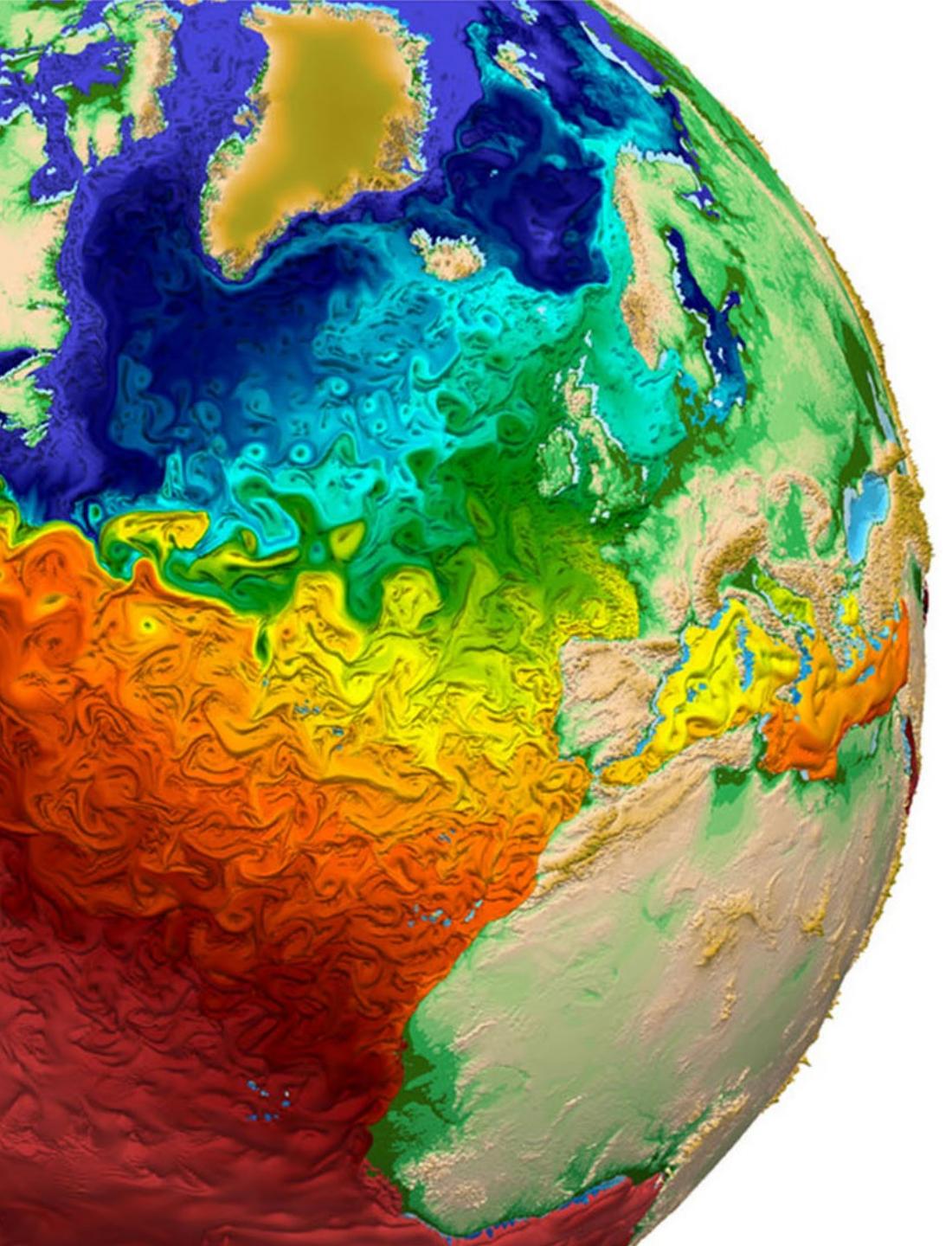
W. Newman
Editor

Illumination for Computer Generated Pictures

Bui Tuong Phong
University of Utah

Fig. 9. Improved shading, applied to the example of Figure 2.

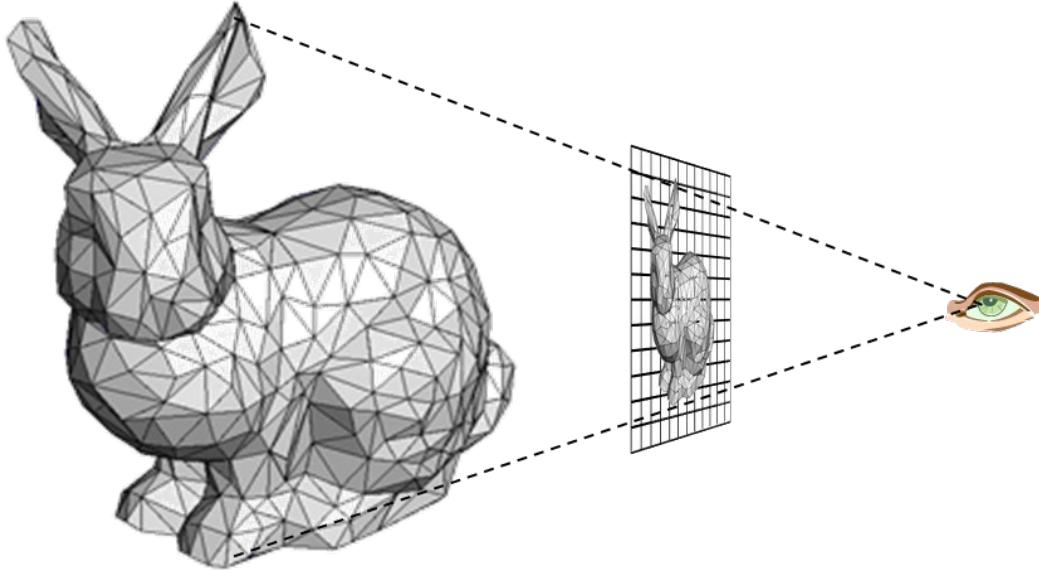




3D Graphics and Visualization

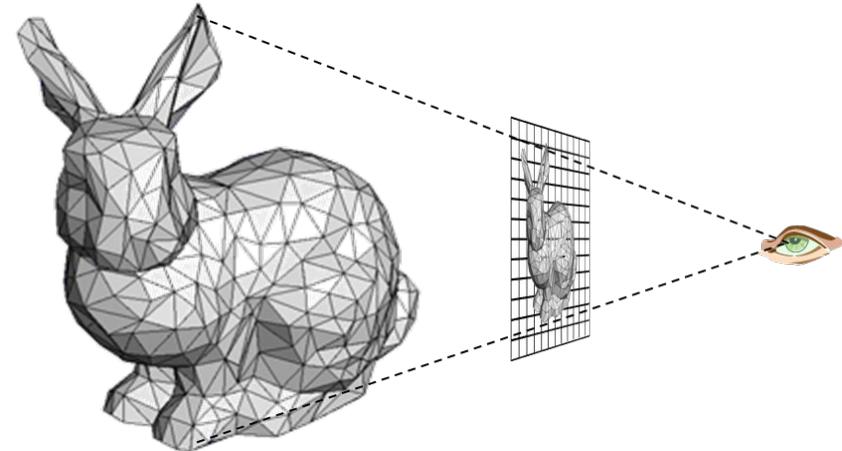
Professor Eric Shaffer

Understanding Real-time Rendering



- 3D surfaces are modeled with triangle meshes
- Each triangle is projected to 2D
- Each triangle is rasterized into pixels
- Each pixel is shaded according to some model

What Questions to Ask?



- What determines how performant an application is?
- What visual artifacts of the rendering process impact visualization?
 - Projection
 - Shading
 - Hidden surface removal

Performance

- Complicated issue
- Number of triangles in surface model often the key factor
 - Fewer triangles = faster rendering



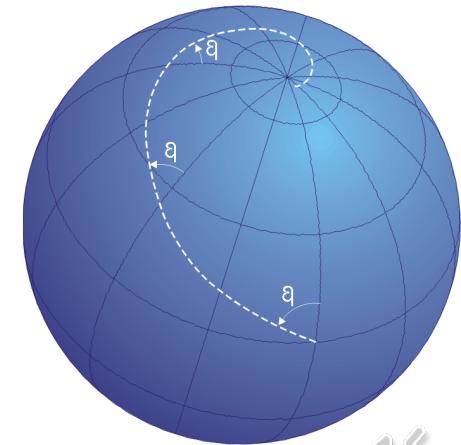
Projections and Distortion



Mercator Projection

- Developed by Gerardus Mercator in 1569
- Lines of constant bearing are straight lines on map.
- Revolutionary for naval navigation
 - Easily plot a course of constant bearing between 2 locations
 - Just maintain a heading using a compass
 - Avoids repeated course corrections to new heading

A constant bearing means maintaining a constant angle between the direction of navigation and true north. A **rhumb line** is a course of constant bearing...it will cross lines of longitude at the same angle and spiral into the pole



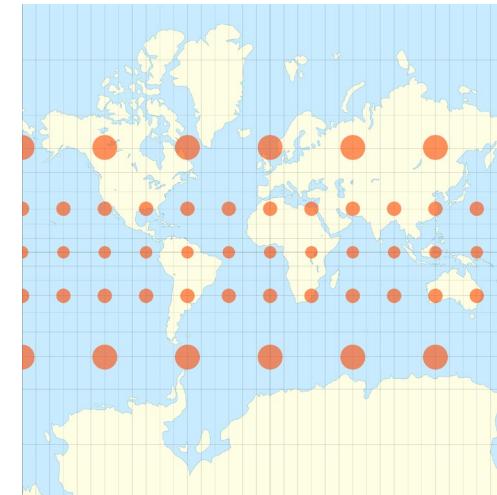
Mercator Projection



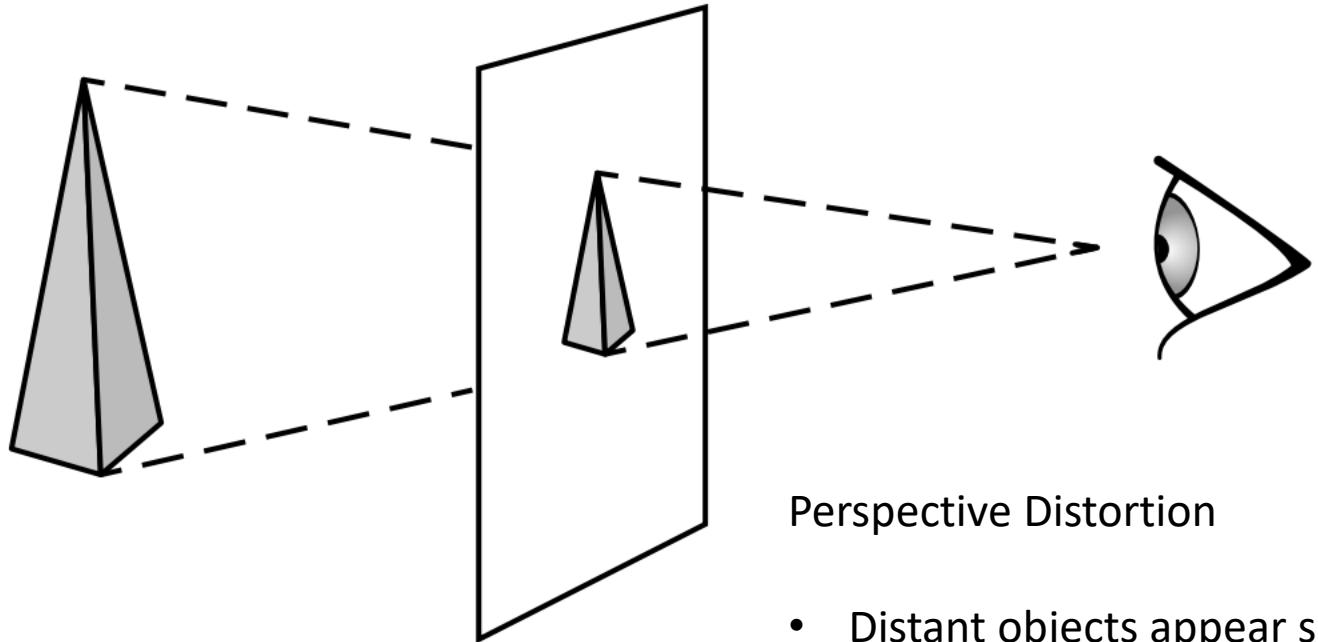
Excellent for the task of navigation

Poor for the task of comparing relative areas

- Africa is actually 14 times larger than Greenland
- Areas farther from the equator appear larger
- Circles on the indicatrix show relative distortion

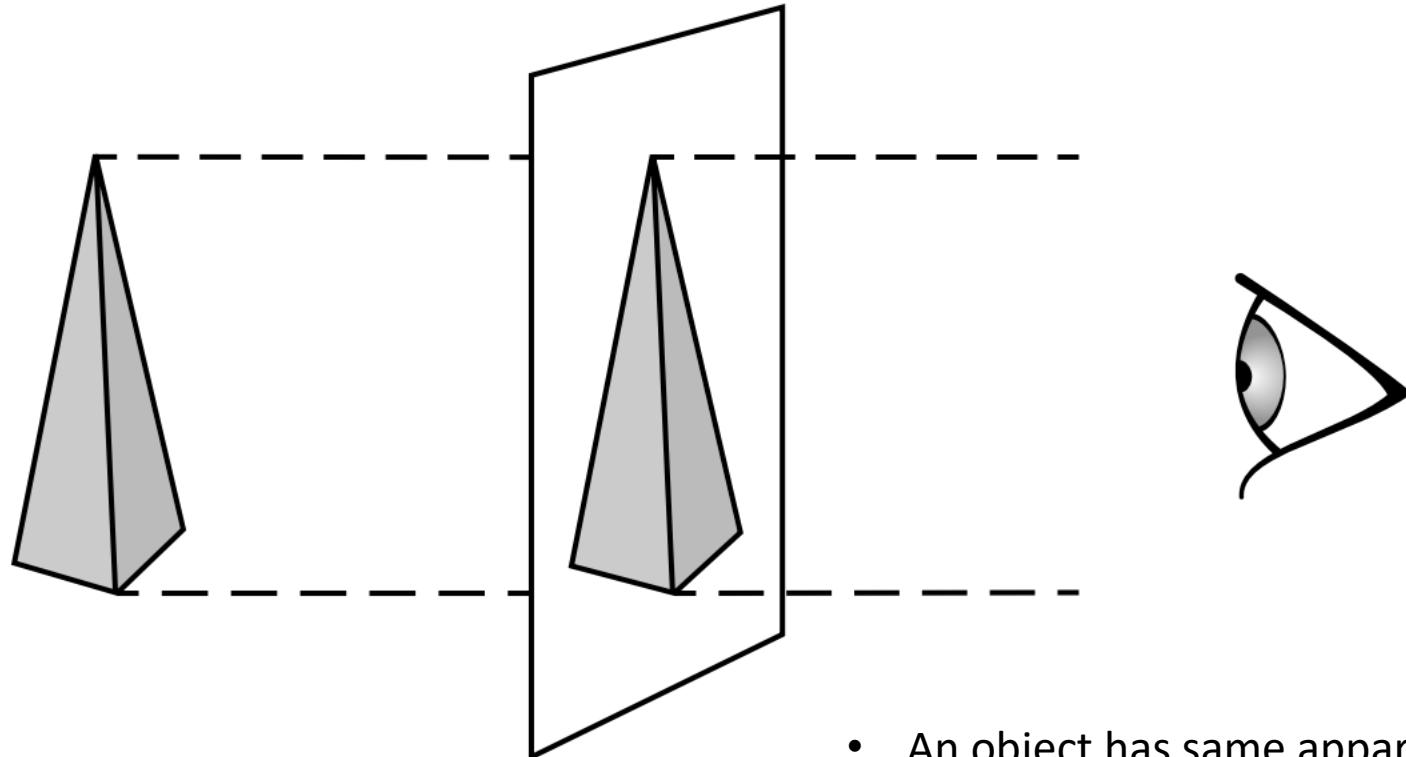


Perspective Projection



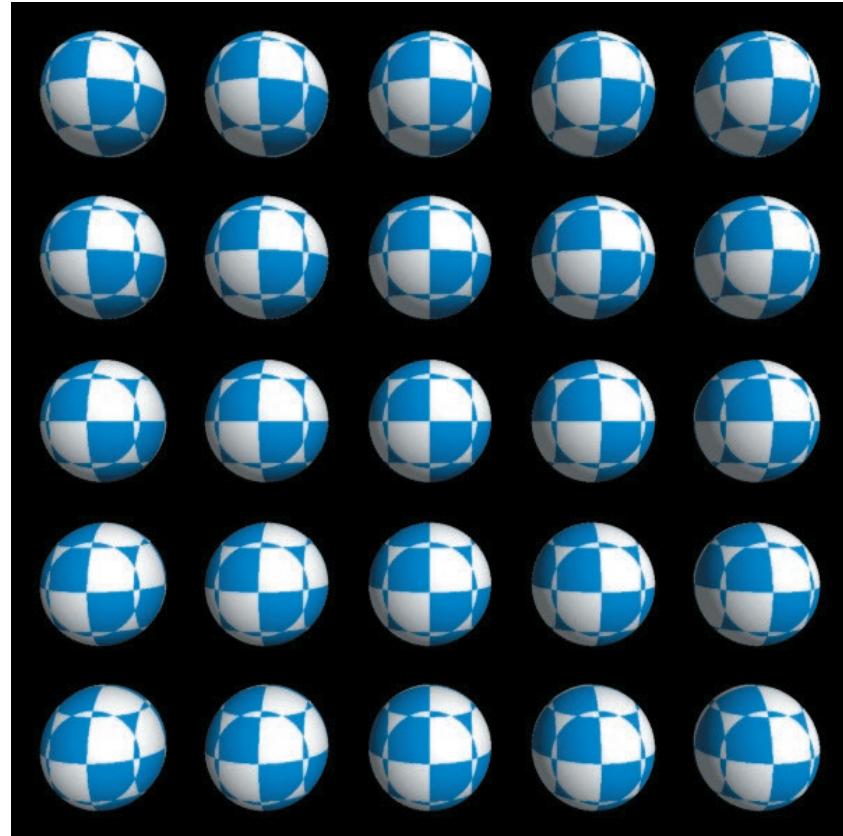
- Distant objects appear smaller than the same object close up
- Objects close to the image plane and away from CoP will look elongated
- CoP = center of projection

Orthographic Projection

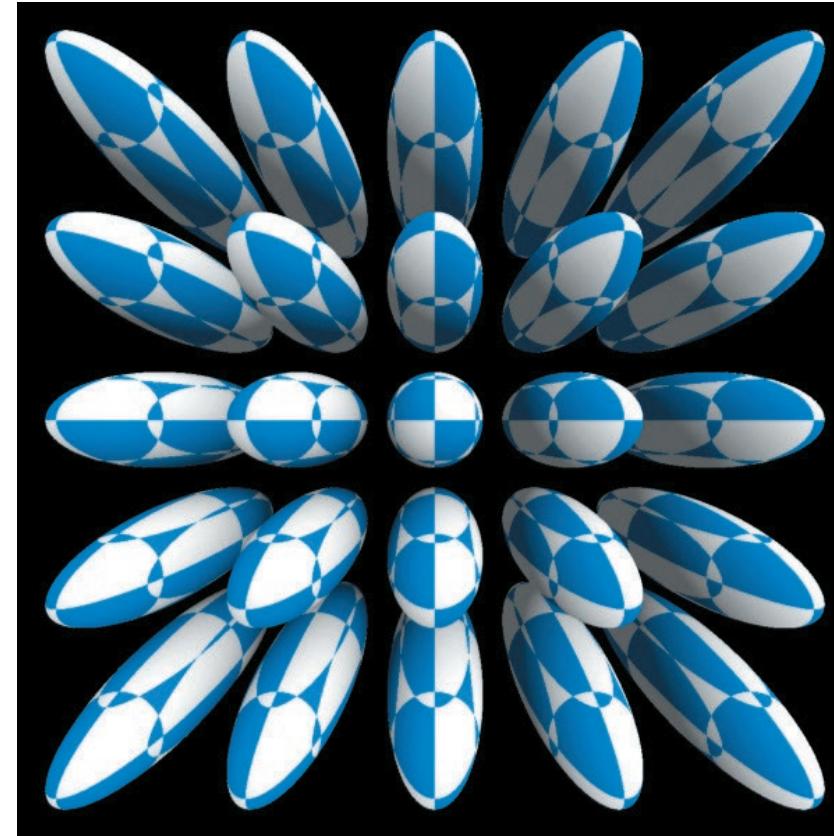


- An object has same apparent size regardless of distance from the eye.
- Foreshortening can still occur if object is angled away from eye

Projections and Distortion



Orthographic Projection

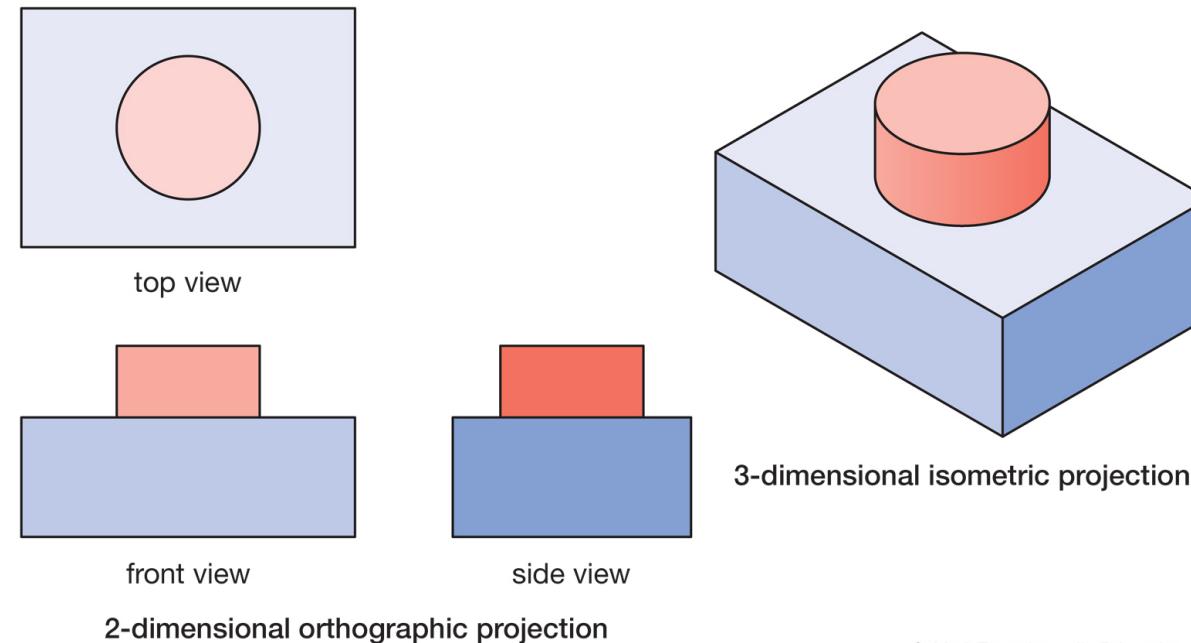


Perspective Projection

Orthographic Projection for Engineering

If comparing lengths is important for application, consider orthographic

Orthographic and isometric projections of an object



© 2012 Encyclopædia Britannica, Inc.

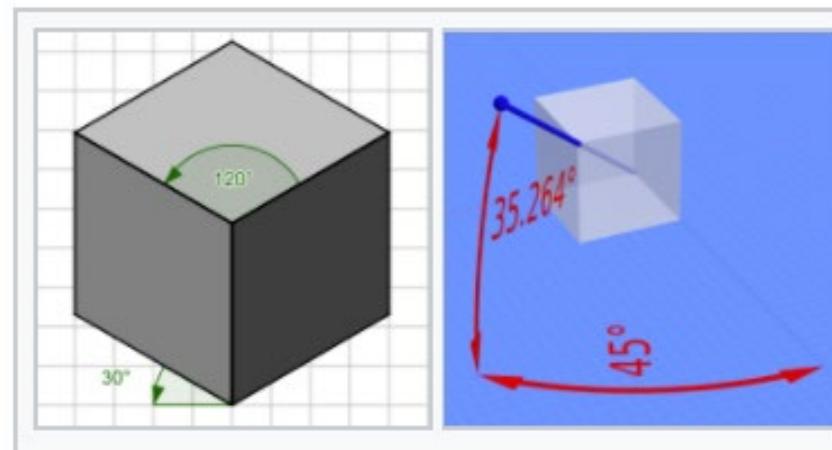
Isometric Projection

Isometric projections are commonly used in technical drawings and some computer game graphics.

In an isometric projection the three axes appear 120° from each other and are equally foreshortened.

It can be achieved by:

1. rotating an object 45° around the vertical axis (Y)
2. rotating $\sim 35.3^\circ$ () through the horizontal axis (X)
3. projecting orthographically onto XY plane



There are actually 8 different orientations that could be used to achieve an isometric projection.

Gives the impression of 3D but relative lengths are preserved.

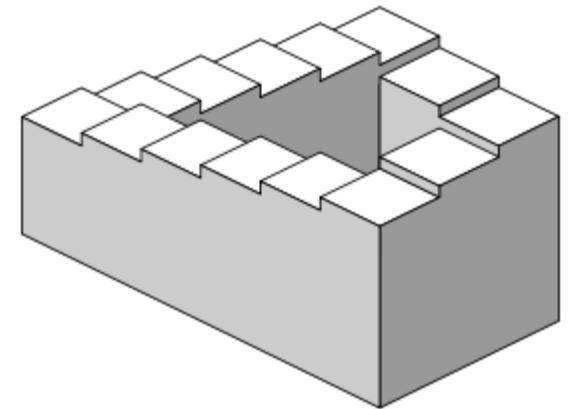
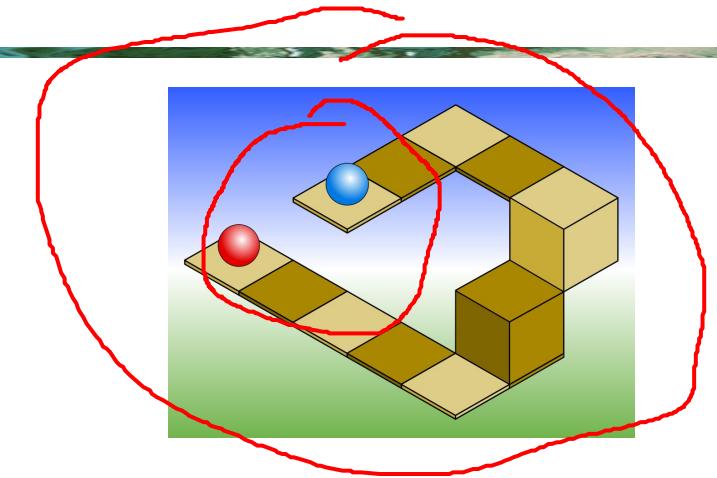
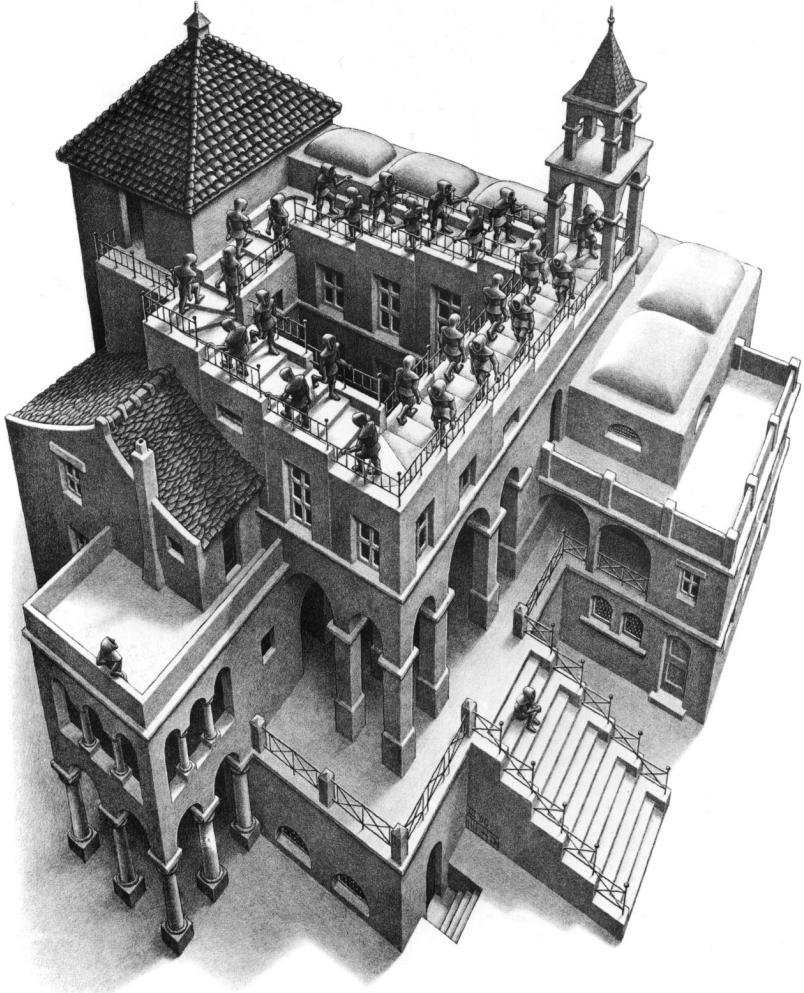
Isometric Projection in Art

Used to generate 3D perspective in Chinese and Japanese art



The "Qing Court Version" of *Along the River During the Qingming Festival* (清院本清明上河圖)
an 18th-century remake by Chen Mei, Sun Hu, Jin Kun, Dai Hong, and Cheng Zhidao

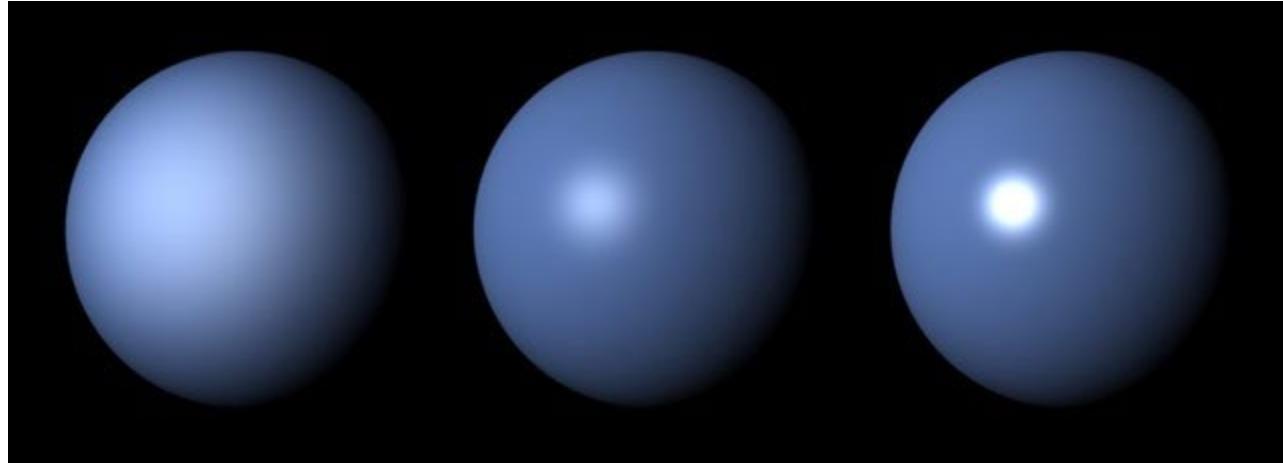
Isometric Projection in Art



It can easily generate optical illusions

Since objects different depths project to the same size you cannot judge distance effectively

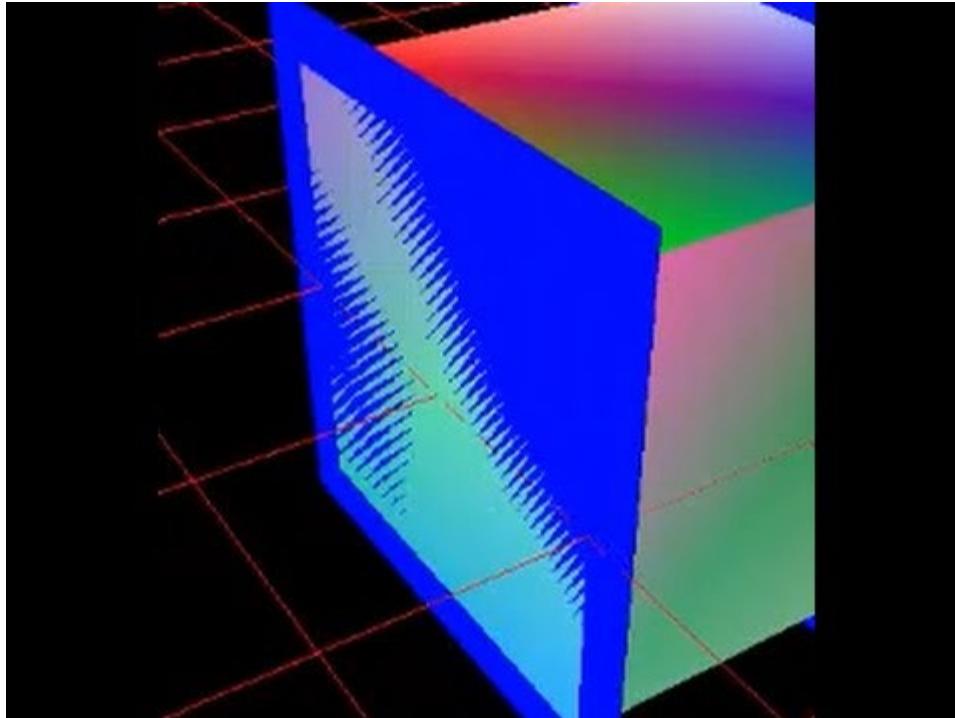
Shading and Visualization



Be aware of the impact of shading on visualization

- Important 3D visual cue
 - Especially diffuse shading in Blinn-Phong
- Was non-white light used?
 - Can change the rendered color of the surface
- Too much ambient light can wash away details
 - Too little can leave structures too dark

Hidden Surface Removal and Z-Fighting



Can occur when 2 surfaces are co-planar or close to co-planar

The “Z” refers to depth...distance from the camera

The rendering engine inconsistently determines which surface is closest

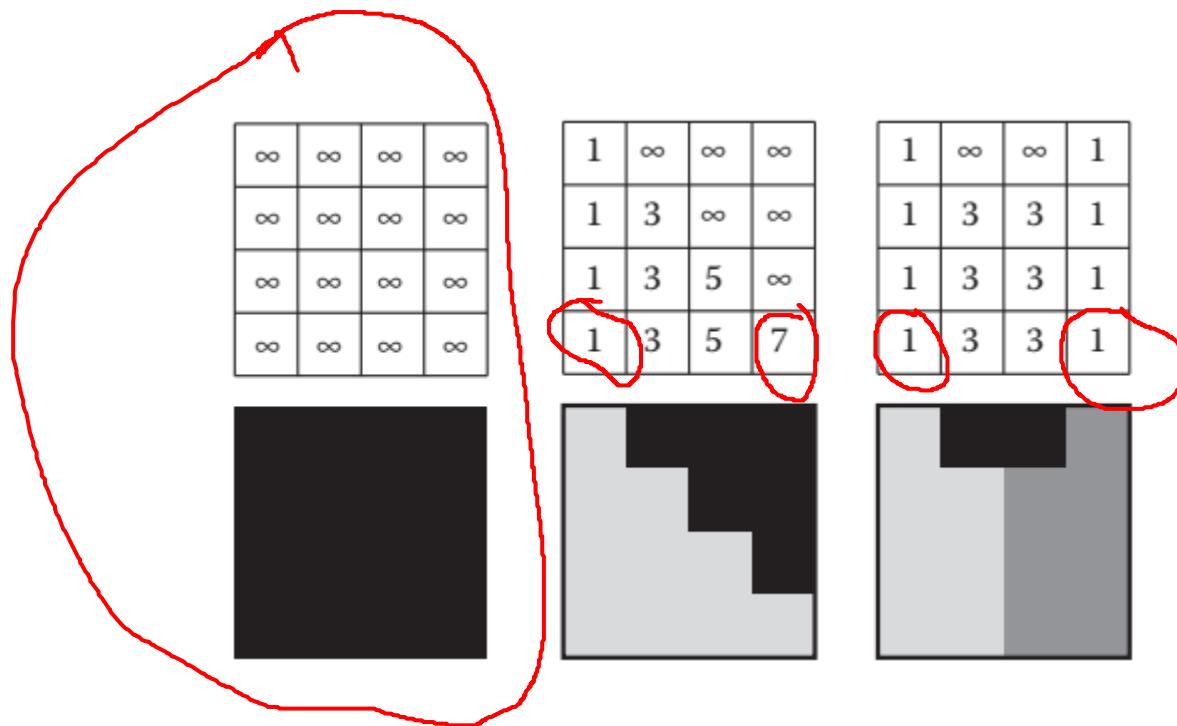
Why?

Hidden Surface Removal and Z-Fighting

Each fragment has a z-value (positive depth from camera)

Hidden surface removal compares the z-values of fragments at same screen location

Fragment with least z-value is retained

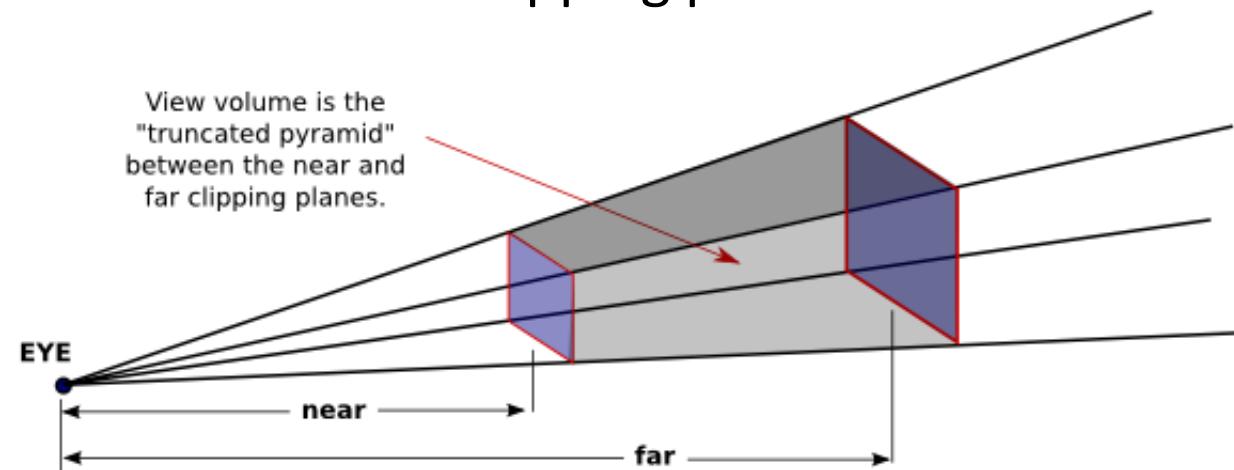


Hidden Surface Removal and Z-Fighting

Depths from the camera lie in the range $[n, f]$

n is the positive distances to the near clipping plane

f is the positive distance to the far clipping plane



To simplify things, assume depths are positive integers $\{0, 1, \dots, B-1\}$

Map n to 0 and f to $B-1 \rightarrow$ each integer in our range corresponds to a bucket of depth $\Delta z = \frac{f-n}{B}$

Hidden Surface Removal and Z-Fighting

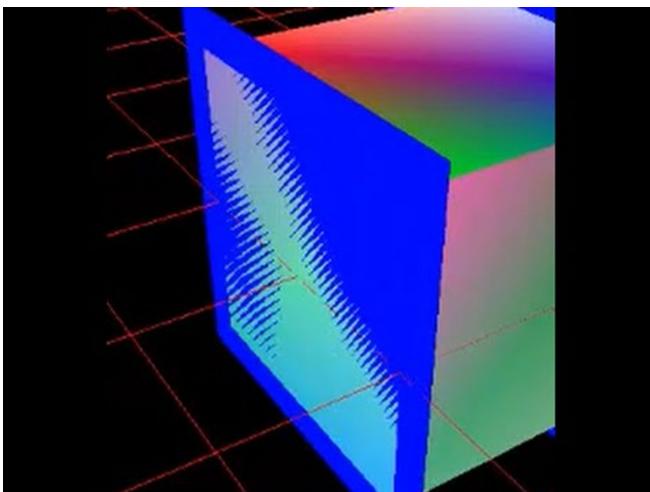
To simplify things, assume stored depths are positive integers $\{0, 1, \dots, B-1\}$

Map n to 0 and f to $B-1 \rightarrow$ each integer in our range corresponds to a bucket of depth $\Delta z = \frac{f-n}{B}$

If you render a scene in which surfaces have a separation of 1 m, if $\Delta z < 1$ then there should be no z-fighting

If the separation is less than the bucket depth...you can have z-fighting

- Cannot determine which surface is closest
- Rounding errors may switch which surface is chosen as closest in different parts of the scene

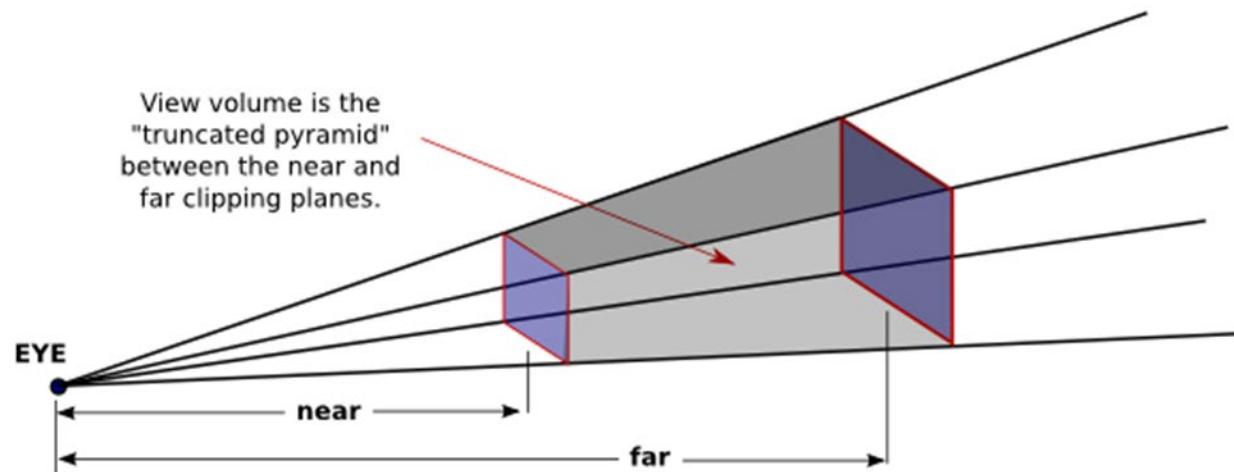


Hidden Surface Removal and Z-Fighting

Some fixes for z-fighting

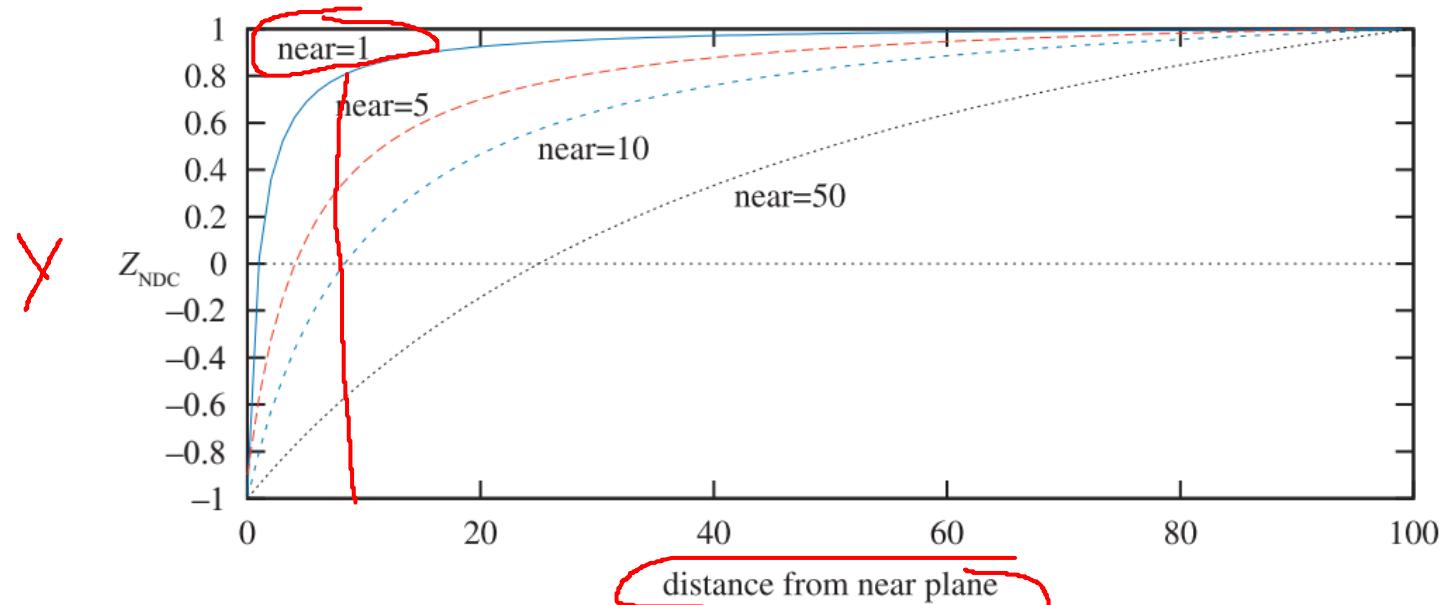
- Move the near and far planes closer together
- Move surfaces apart

$$\Delta z = \frac{f - n}{B}$$



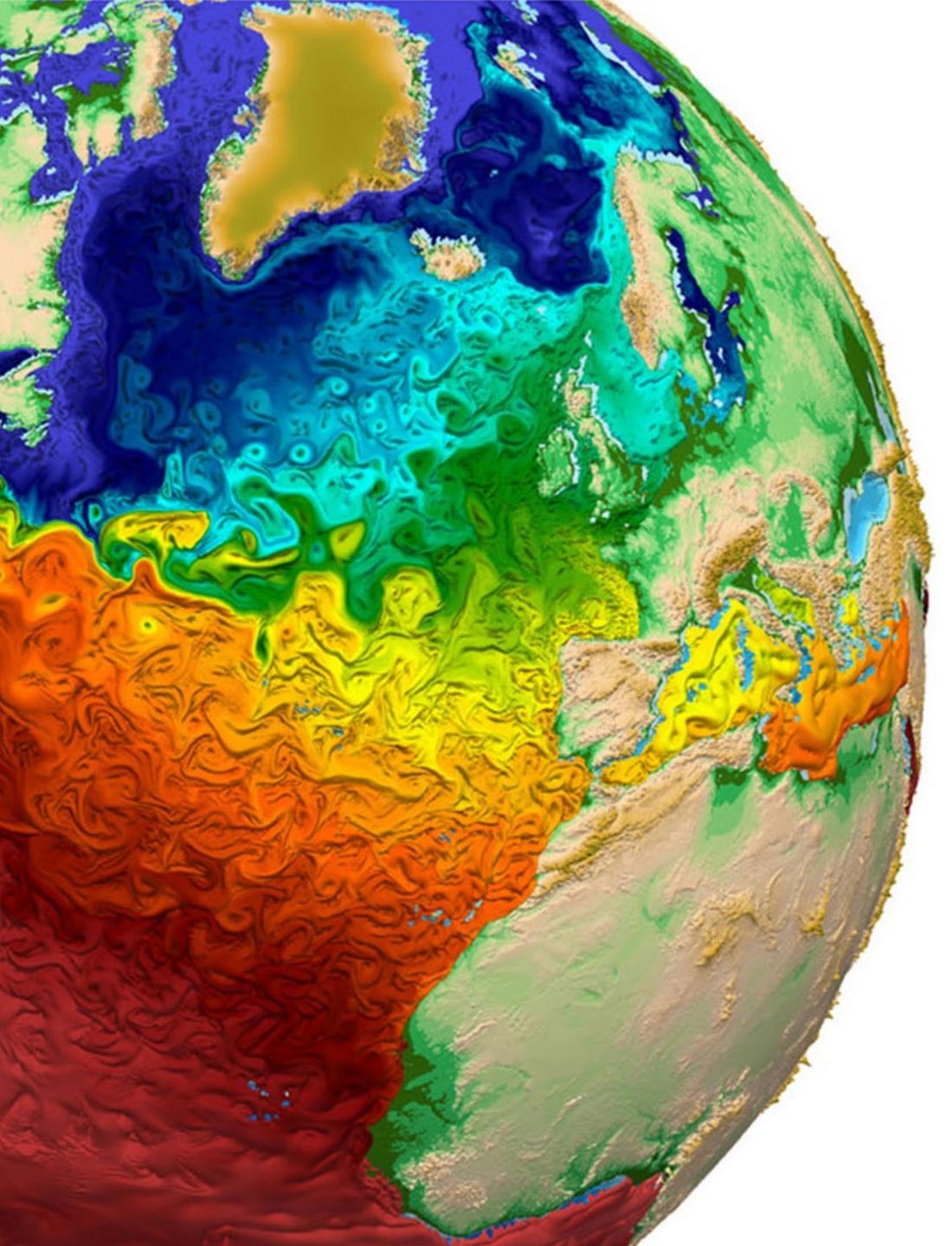
Hidden Surface Removal and Z-Fighting

In actuality, bucket sizes will vary by depth due to perspective projection



Here, $f-n = 100$ and each distance in the range is mapped into $[-1,1]$

- Cannot choose $n=0$ as that results in an infinitely large bucket
- Larger bins at greater depths
 - Ability to do hidden surface removal degrades with distance



Scalar Field Visualization

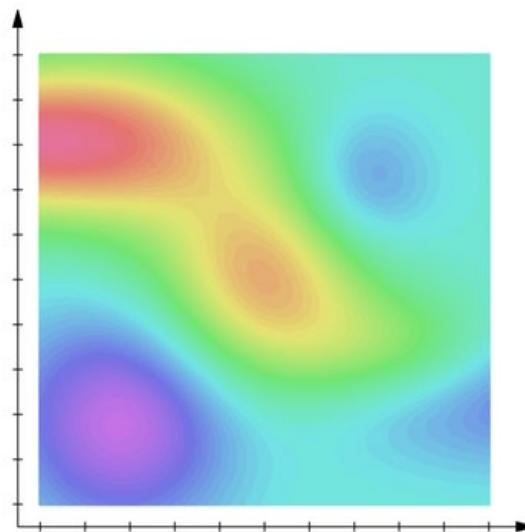
Colormaps

Scientific Visualization
Professor Eric Shaffer

What is a Scalar Field?

A scalar is a single quantity...a number

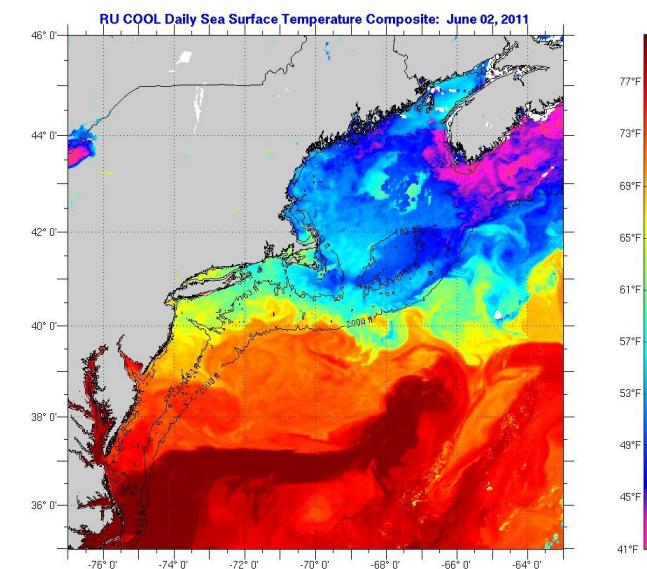
A scalar field assigns a scalar to every point in a given space



For a 2D space, a scalar field is often visualized using color

Render a representation of domain and assign a color to each pixel

Seemingly simple task...just need to construct a colormap



-0.382 +0.459

Coloring Continuous Data

Coloring to denote continuous data is called *pseudo-coloring*

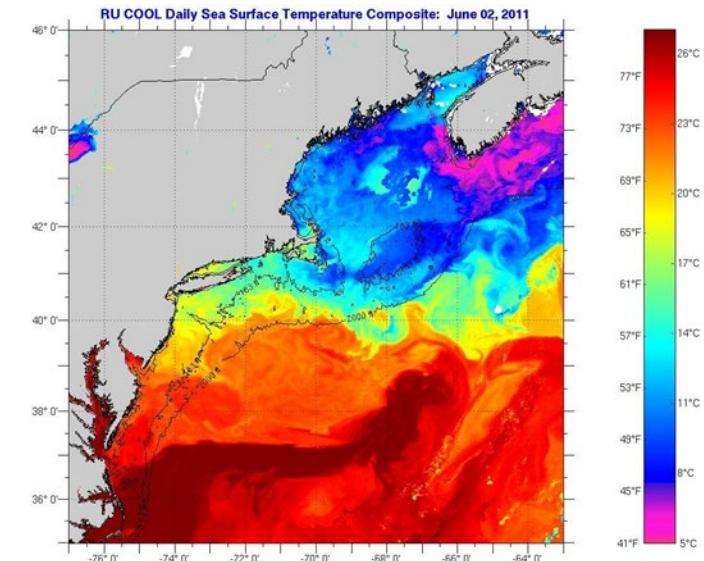
Such a mapping from value to color is called a *choropleth*

Anyone know the standard weather map colors?

How about elevation in geography?

What is the most used colormap?

Is it any good?



Coloring Continuous Data

Coloring to denote continuous data is called *pseudo-coloring*

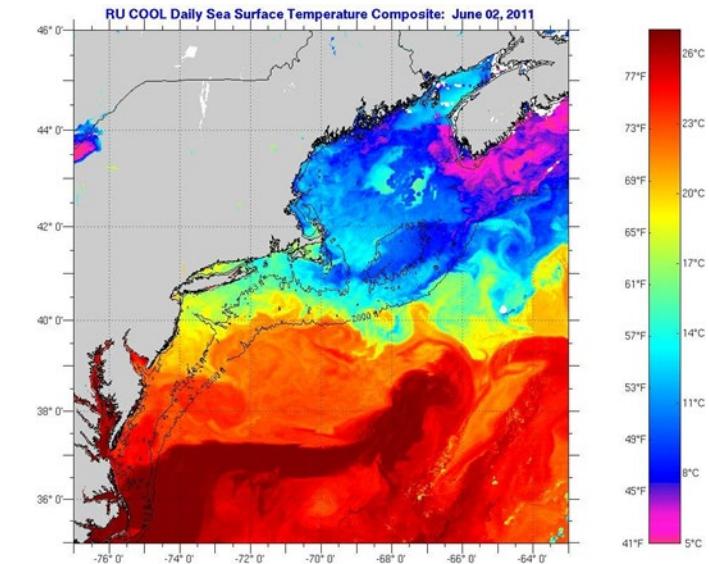
Such a mapping from value to color is called a *choropleth*

Anyone know the standard weather map colors?
blue-cyan-green-yellow-orange-red

How about elevation in geography?
blue-green-brown-white

What is the most used colormap?
rainbow

Is it any good?
probably not

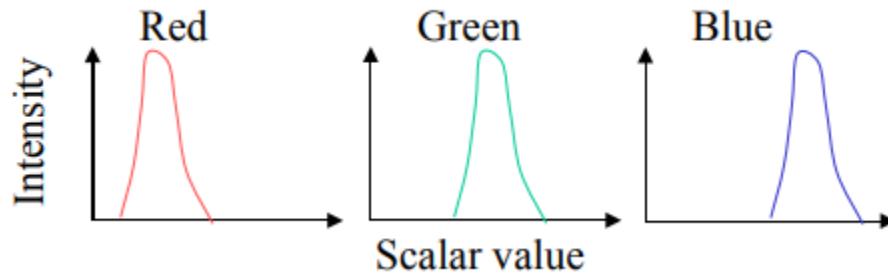


Designing a Color Map: Two Options

- Color Table
 - Pre-compute colors and store colors



- Transfer Function



Designing a Colormap : Generate a Color Table

Map each scalar value $x \in R$ at a point to a color via a table lookup

Assume that we know $x \in [x_{min}, x_{max}]$

Color Tables

- precompute colors and save results into a table of colors $\{c_1, \dots, c_N\}$
- index table by mapping ranges to integers

Suppose we have N colors in a table and we index them $[0, N-1]$

Typically a color mapping function might generate an index i:

$$i = \min \left(\left\lfloor \frac{x - x_{min}}{\frac{x_{max} - x_{min}}{N}} \right\rfloor, N - 1 \right)$$

Transfer Functions

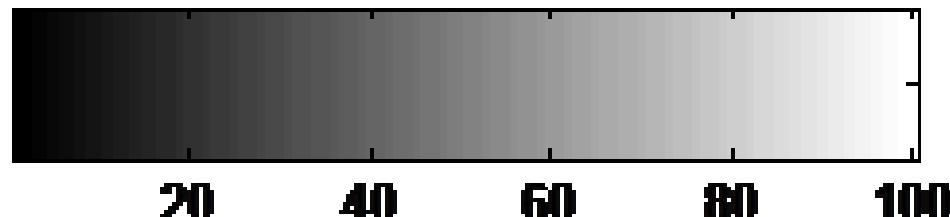
A transfer function defines colors at certain scalar values

- These points are sometimes called knots

Interpolation is then used to define colors for values in between the knots

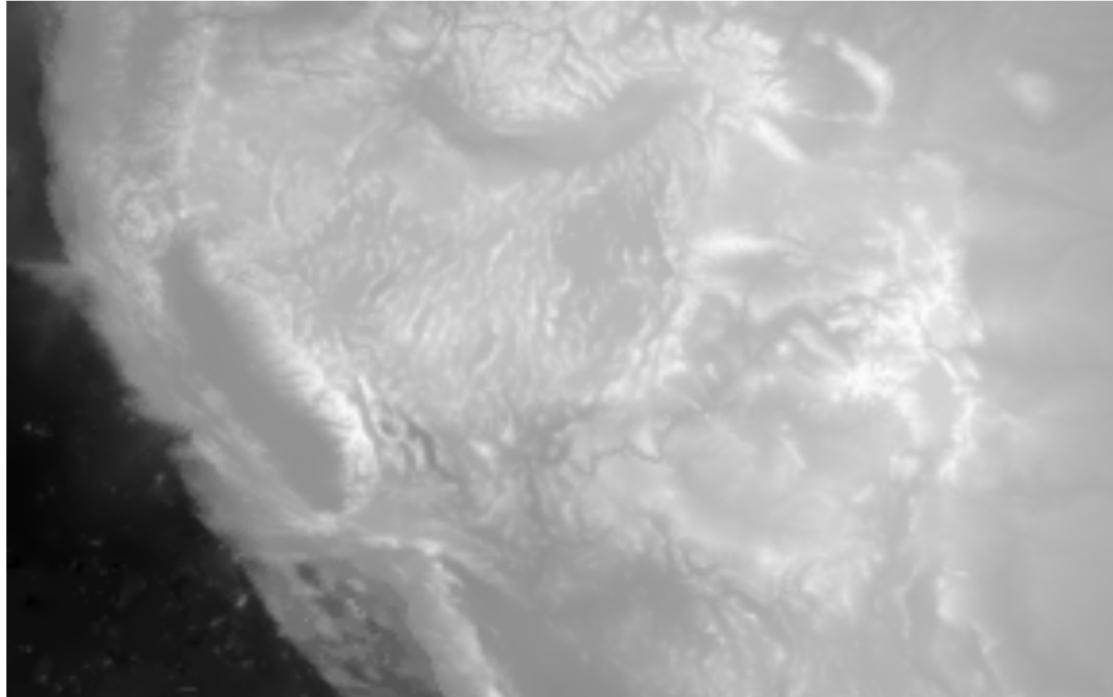
Example:

- Consider a function with a range of [0,100]
- $c(0) = (0,0,0)$ and $c(100)=(1,1,1)$ and use linear interpolation in between

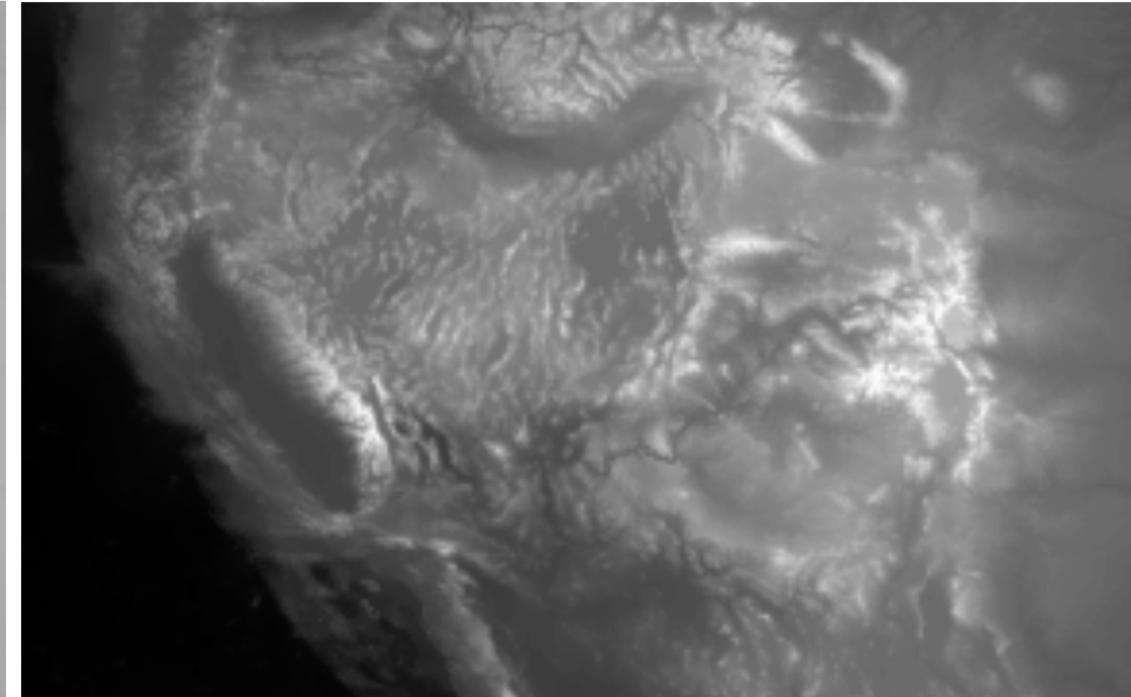


This is a simple but super effective colormap!

Perceptually Linearized Grayscale



Regular Grayscale



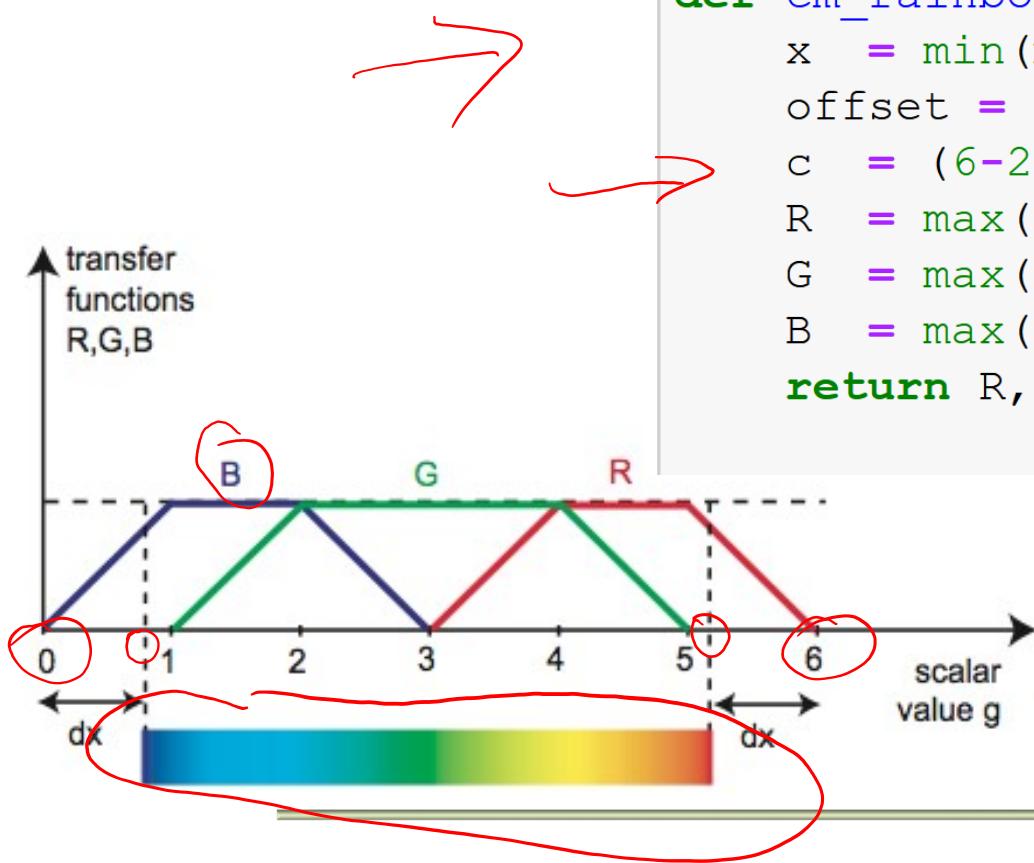
Perceptually Linearized Grayscale

Rainbow Colormap



- Probably most (in)famous colormap in visualization
 - “Cold colors” → Low values
 - “Warm colors” → High values
- People like it...think they can use it well...studies say they do not

Example: Implementing the Rainbow Colormap



```
import numpy as np
import math

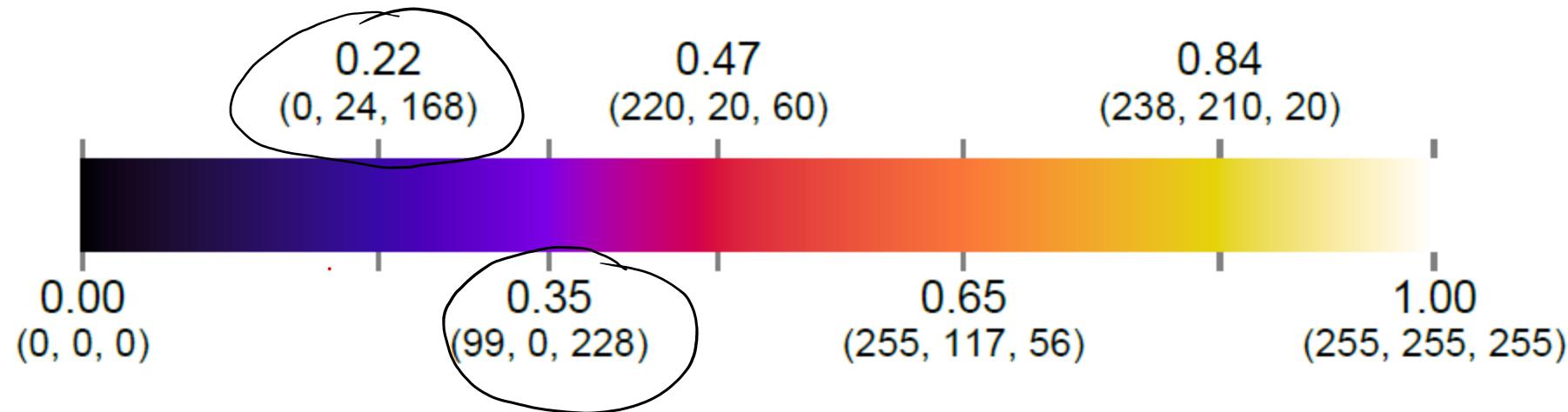
def cm_rainbow(x):
    x = min(max(x,0),1)          #clamp x to expected range [0,1]
    offset = 0.8                  #magic number...like a step size
    c = (6-2*offset)*x + offset #scale x to [offset,6-offset]
    R = max(0, (3-math.fabs(c-4)-math.fabs(c-5))/2.0)
    G = max(0, (4-math.fabs(c-2)-math.fabs(c-4))/2.0)
    B = max(0, (3-math.fabs(c-1)-math.fabs(c-2))/2.0)
    return R, G, B
```

Criticism

- The user is conceptually mapping a linear scale in hue onto a scalar variable
 - Perceptually, however, this scale does not appear linear
 - Equal steps in the scale do not correspond to equal steps in color.
 - The colors appear to change much faster in yellow region than green region.
- Gives impression that the data are organized into discrete regions
 - This can lead the user to infer structure which is not present in the data ...and to miss details that lie completely within a single color region
- Rainbow color map is sensitive to deficiencies in vision
 - Roughly 5% of the population has deficiencies in distinguishing these colors



Alternatives to the Rainbow

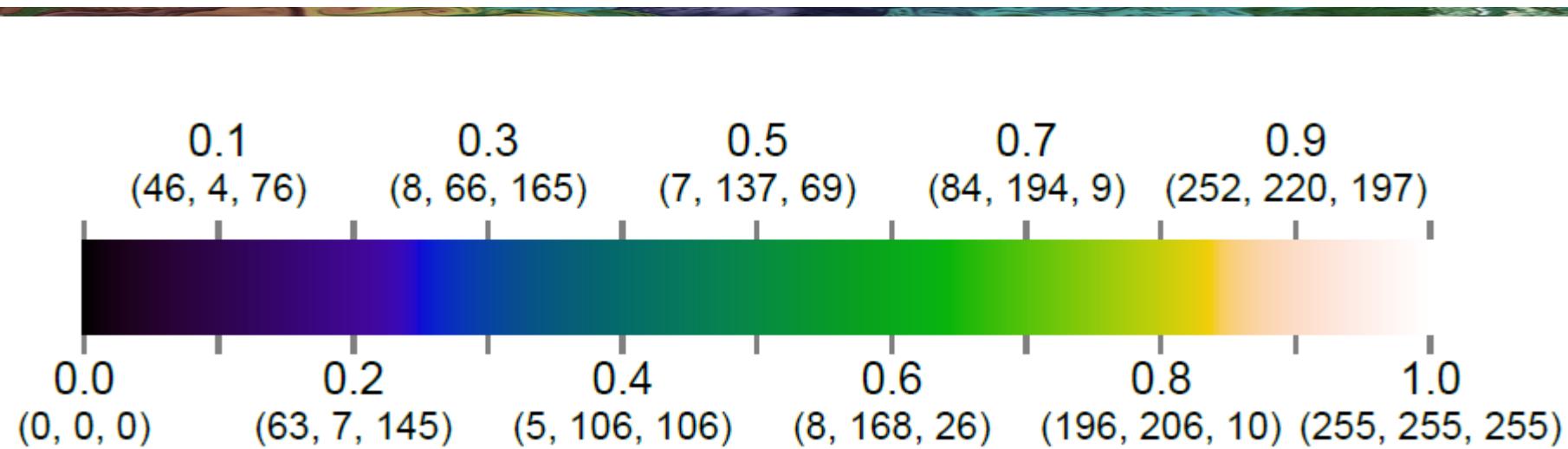


Perceptually linear color map

A change in the underlying metric is matched by similar perceptual change in color

Python code, etc.: <https://www.kennethmoreland.com/color-advice/>

Kindlmann Colormap

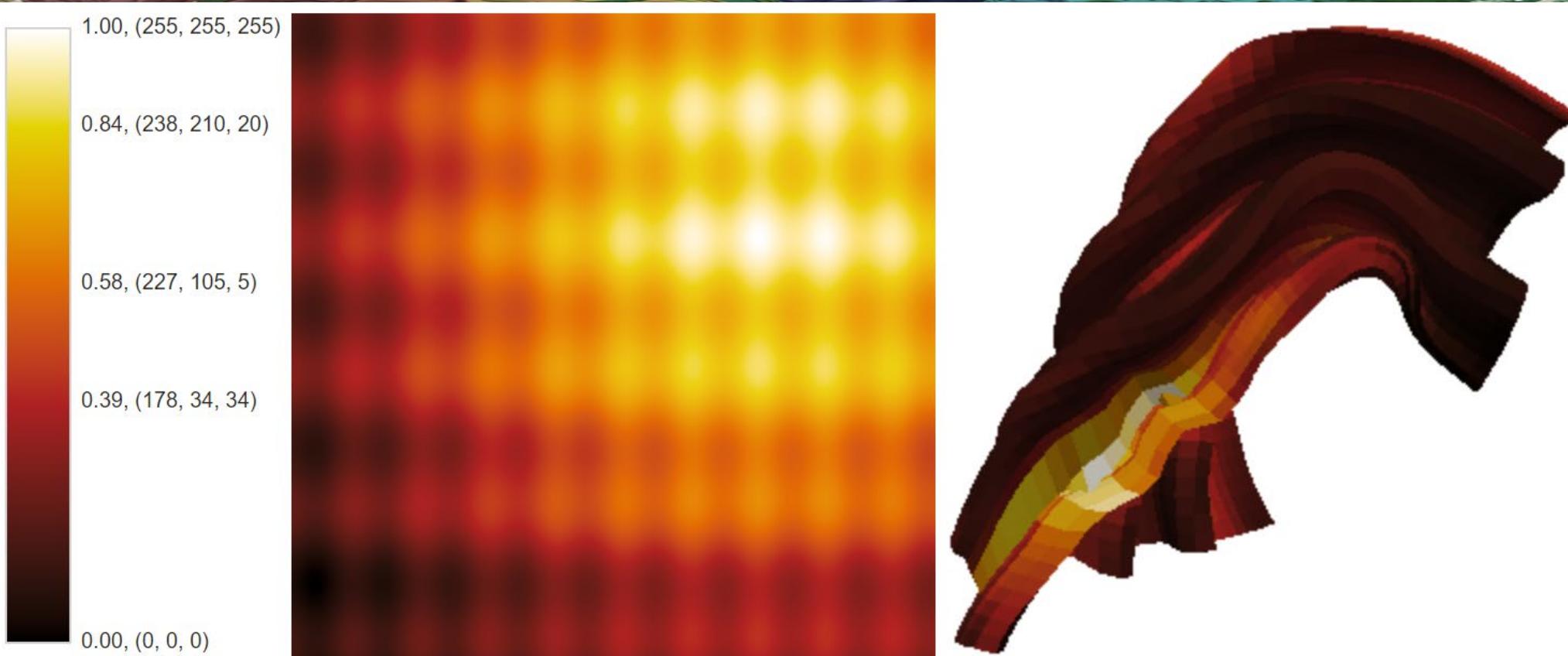


Also perceptually linear....or at least close

Python code, etc.: <https://www.kennethmoreland.com/color-advice/>

Gordon Kindlmann, Erik Reinhard, and Sarah Creem. Face-based luminance matching for perceptual colormap generation. In *Proceedings of IEEE Visualization*, pages 299–306, October 2002.
DOI 10.1109/VISUAL.2002.1183788.

Black Body Colormap



Based on colors from black body radiation.
Designed to have a constant increase in brightness throughout.

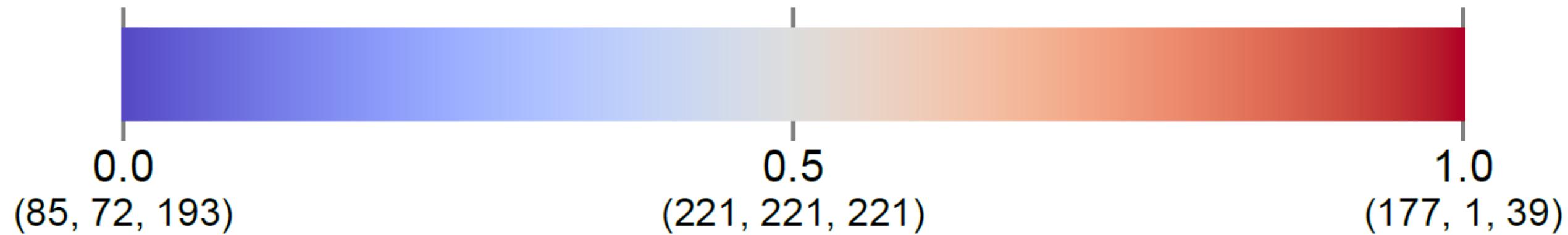
Python code, etc.: <https://www.kennethmoreland.com/color-advice/>



Diverging Colormaps

The underlying data can inform your choice...

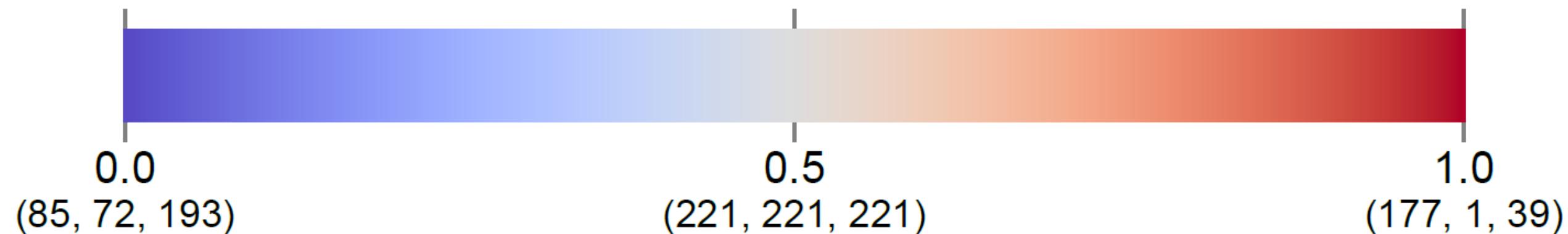
- Is there critical value the viewer should be aware of?
- A *diverging colormap* would be appropriate



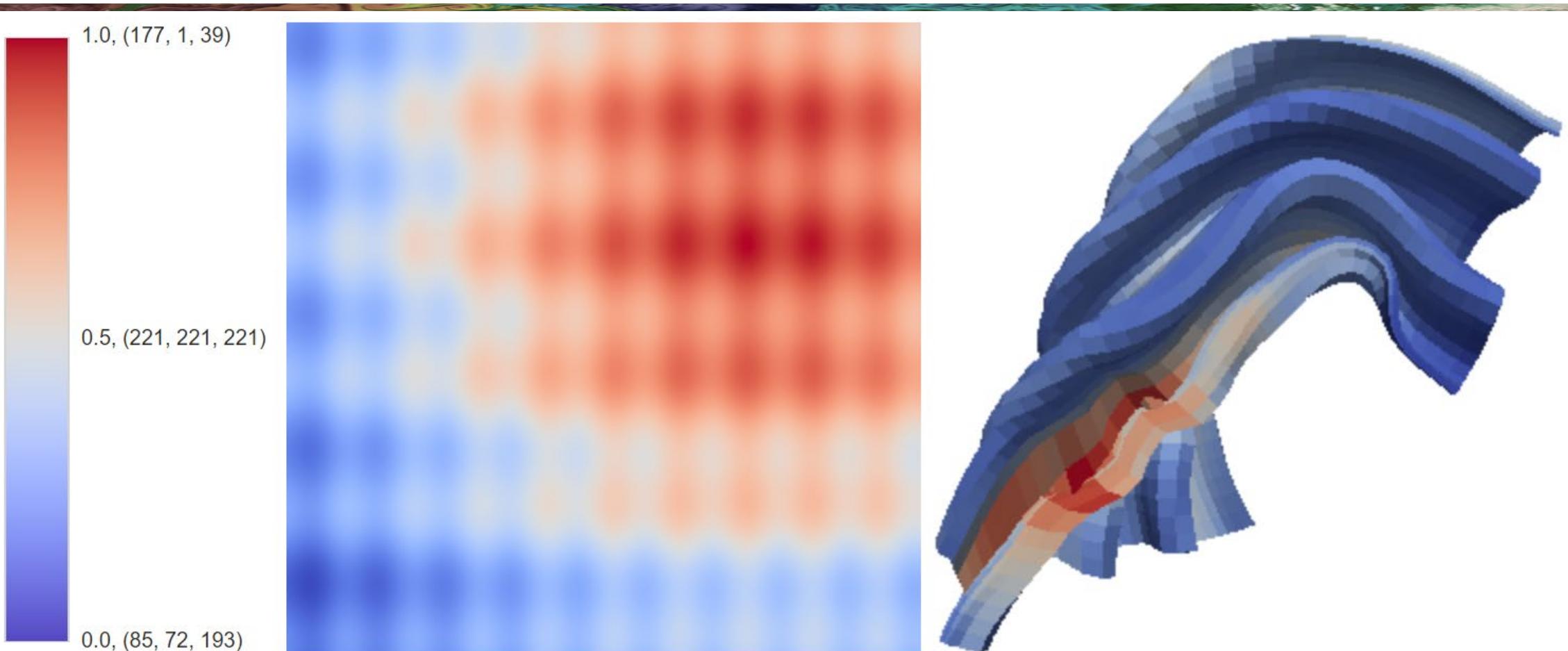
Diverging: change in lightness and possibly saturation of two different colors that meet in the middle at an unsaturated color; should be used when the information being plotted has a critical middle value, such as topography or when the data deviates around zero. -
<https://matplotlib.org/tutorials/colors/colormaps.html>

Colormaps for 3D Surfaces

- Ideally, colormap should use change in luminance to display changes in value.
- However, in 3D scene, shading cues are vital to understanding shapes.
- Need to avoid colormap and shading from interfering
- Achieve this by limiting the color map to bright colors.
 - Reduces the total range of brightness in the color map

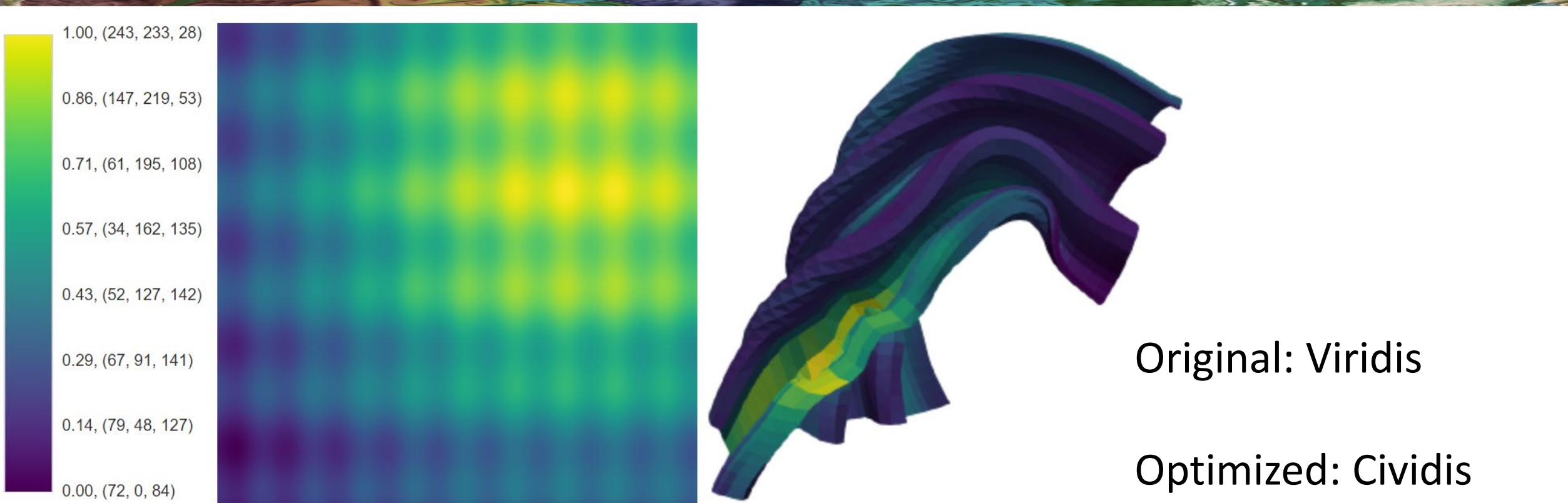


Colormaps for 3D Surfaces



Python code, etc.: <https://www.kennethmoreland.com/color-advice/>

Colormaps for 3D Surfaces



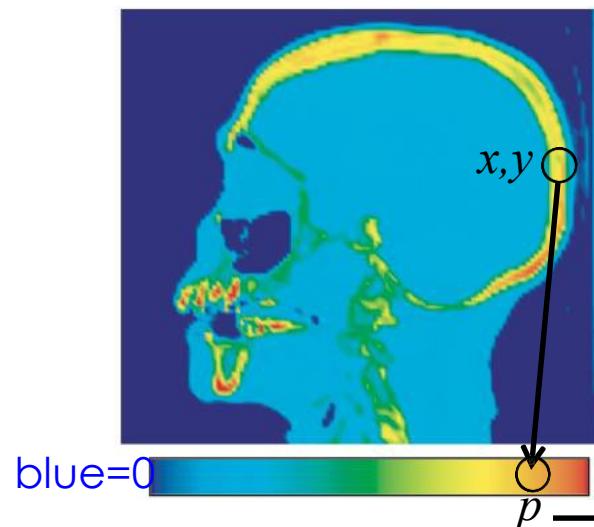
Optimizing colormaps with consideration for color vision deficiency to enable accurate interpretation of scientific data

Keyed Lookup Tasks

Keyed Lookup: User wants to estimate specific data value from color

Some colormaps better than others for this task

Banded maps like rainbow (but don't use it) better than linear maps such as grayscale



Data values mapped to RGB colors via a [colormap](#)

Invert mapping:

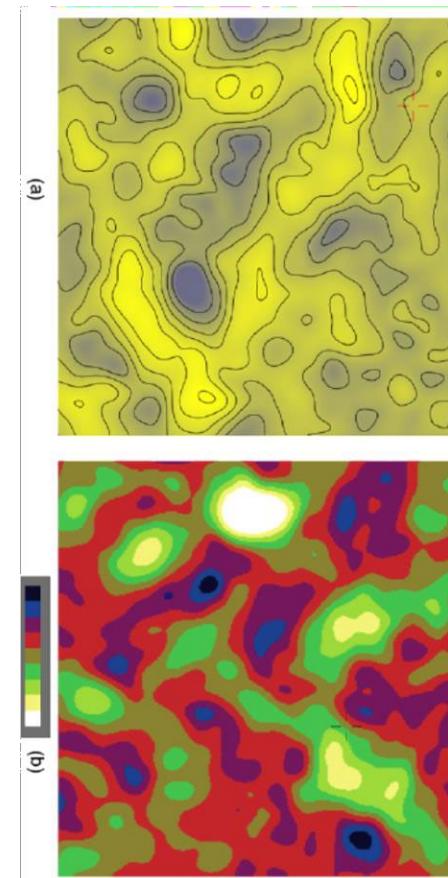
1. look at some point (x,y) in the image \rightarrow color c
2. locate c in colormap at some position p
3. use the colormap legend to derive data value s from p

answer: $s = 90$

Keyed Lookup Tasks

Keyed Lookup: User wants to estimate specific data value from color

Contours help as well.....



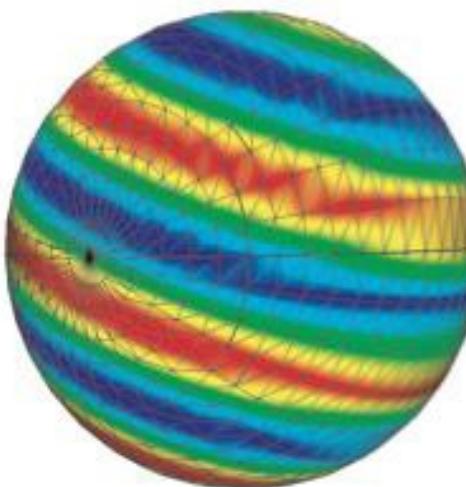
Prefer Interpolating Values to Interpolating Colors

Where to apply the colormap?

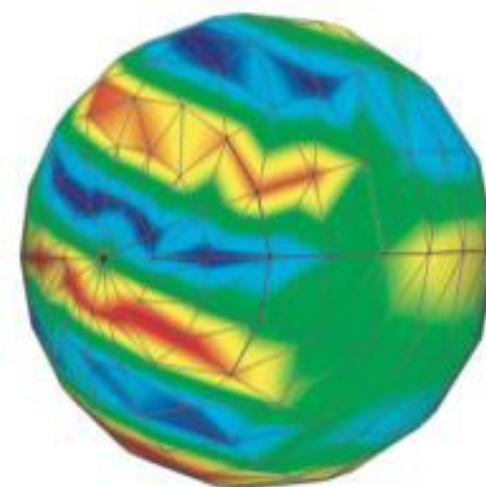
- per pixel – better results than per-vertex colormapping



64x64 points



32x32 points



16x16 points

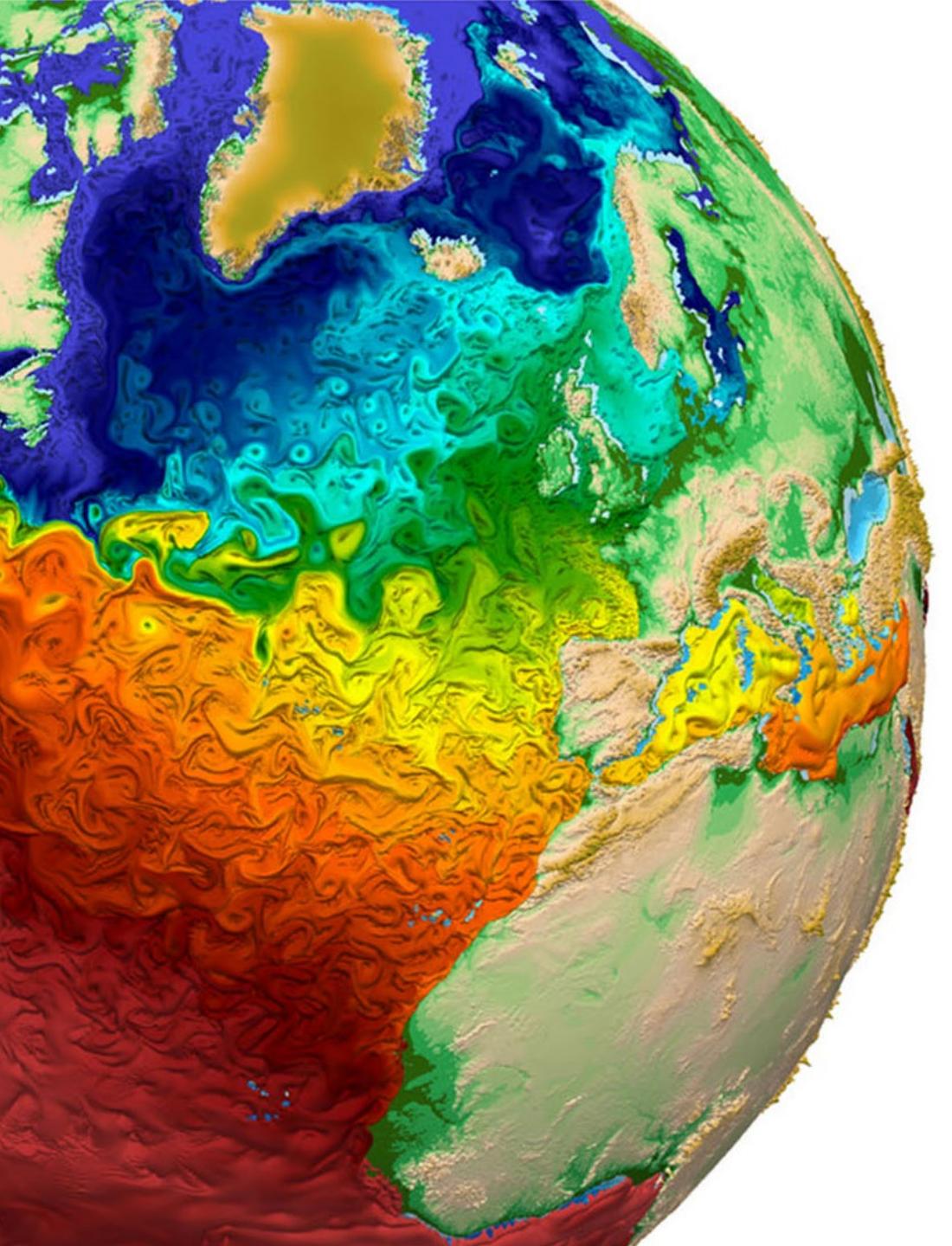
Explanation

color interpolation can fall outside the colormap!

- per-vertex: $f \rightarrow c(f) \rightarrow \text{interpolation}(c(f))$
- per-pixel: $f \rightarrow \text{interpolation}(f) \rightarrow c(\text{interpolation}(f))$ colors always stay in colormap

Colormap Design Advice

- Design for accessibility
 - minimally, don't depend on red-green differentiation
- Use your knowledge of the data set (e.g., is there a critical value?)
- If there is a standard in the field the audience may be expecting?
- Often a perceptually uniform colormap is the best choice
 - equal steps in data are perceived as equal steps in the color space
- We perceive change in lightness as changes in the data pretty well
 - better than changes in hue.
- Use colormaps with monotonically increasing lightness



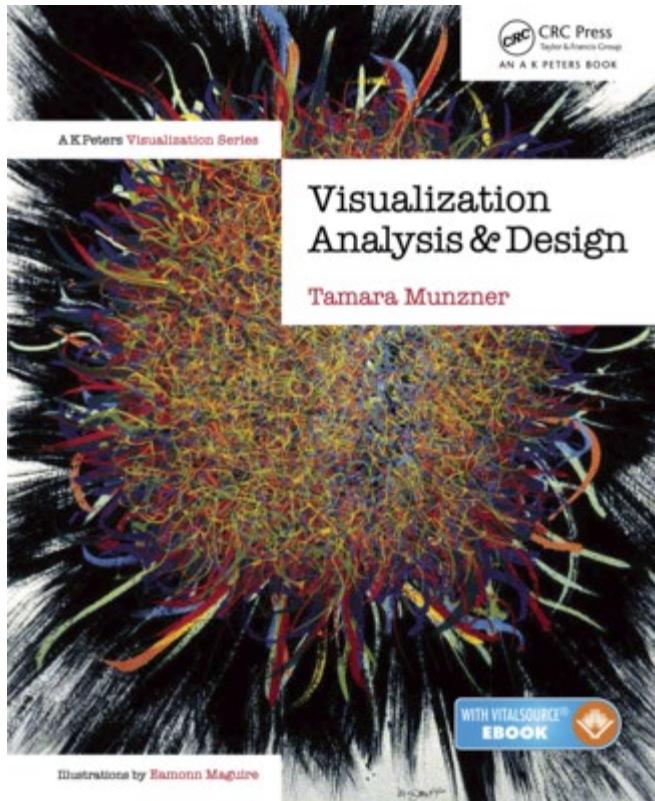
Data Science for People in a Hurry

A Data Taxonomy

Scientific Visualization
Professor Eric Shaffer

Acknowledgment

Material for this lecture from Professor Tamara Munzner



<https://www.cs.ubc.ca/~tmm/>

Some Definitions

Semantics

- real-world meaning

Data Types:

- structural or mathematical interpretation of data
 - different from data types in programming!

```
shaffer1@BOROS:/mnt/c/Users/shaff/Downloads$ tail ClusterDataOriginal.txt
2015-04-30T23:50:00    6.3200e+01    0.0000e+00    0.0000e+00    0.0000e+00
nan
2015-04-30T23:51:00    6.7714e+01    0.0000e+00    0.0000e+00    0.0000e+00
nan
2015-04-30T23:52:00    1.2640e+02    0.0000e+00    0.0000e+00    1.0333e+00
nan
2015-04-30T23:53:00    8.4643e+01    0.0000e+00    0.0000e+00    0.0000e+00
nan
2015-04-30T23:54:00    1.1060e+02    0.0000e+00    0.0000e+00    0.0000e+00
nan
2015-04-30T23:55:00    8.4643e+01    0.0000e+00    0.0000e+00    0.0000e+00
nan
2015-04-30T23:56:00    7.9000e+01    0.0000e+00    0.0000e+00    0.0000e+00
nan
2015-04-30T23:57:00    8.4643e+01    0.0000e+00    0.0000e+00    0.0000e+00
nan
2015-04-30T23:58:00    1.7380e+02    0.0000e+00    0.0000e+00    0.0000e+00
nan
2015-04-30T23:59:00    1.5082e+02    0.0000e+00    0.0000e+00    0.0000e+00
```

Items and Attributes

Item

- Individual entity

Urbana, IL Hourly Weather Forecast ★ 🏠

☀️ 91° AUGERVILLE STATION | CHANGE ▾

TODAY **HOURLY** 10-DAY CALENDAR HISTORY WUNDERMAP

Attribute

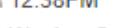
- Property of an item
 - Measurement
 - Observation

< Hourly Forecast for Today, Monday 08/24 >

 Tonight 08/24 ⚡ 10% / 0 in
Clear skies. Low near 70F. Winds SW at 5 to 10 mph.

 Tomorrow 08/25 ⚡ 10% / 0 in
Sunny. High 93F. Winds WSW at 5 to 10 mph.

 Sun  6:14AM  7:38PM

 Moon  12:38PM  11:12PM
Waxing Crescent, 41% visible

Time	Conditions	Temp.	Feels Like	Precip	Amount	Cloud Cover	Dew Point	Humidity	Wind	Pressure
5:00 pm	 Sunny	90 °F	99 °F	<u>2 %</u>	<u>0 in</u>	17 %	72 °F	55 %	8 mph WSW	29.96 in

Other Data Types

- Links
 - express relationship between two items
 - eg friendship on facebook, interaction between proteins
- Positions
 - spatial data: location in 2D or 3D
 - pixels in photo, voxels in MRI scan, latitude/longitude
- Grids
 - sampling strategy for continuous data

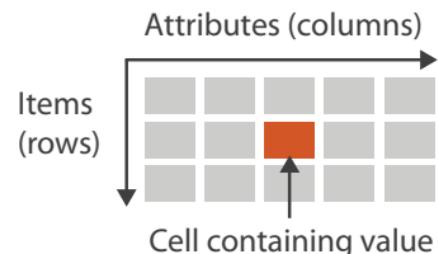
Dataset Types: Tables

Tables

Items

Attributes

→ Tables



flat table

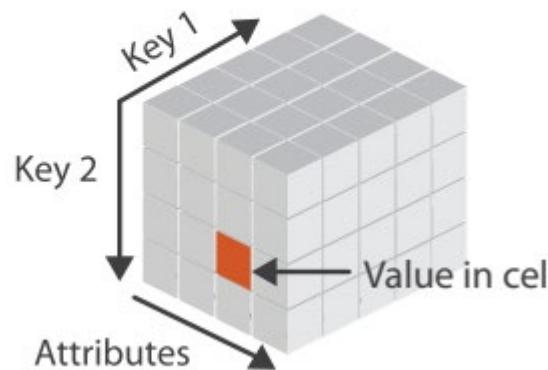
- one item per row
- each column is attribute
- cell holds value

attributes: name, age, shirt size, fave fruit

Name	Age	Shirt Size	Favorite Fruit
Amy	8	S	Apple
Basil	7	S	Pear
Clara	9	M	Durian
Desmond	13	L	Elderberry
Ernest	12	L	Peach
Fanny	10	S	Lychee
George	9	M	Orange
Hector	8	L	Loquat
Ida	10	M	Pear
Amy	12	M	Orange

Dataset Types: Tables

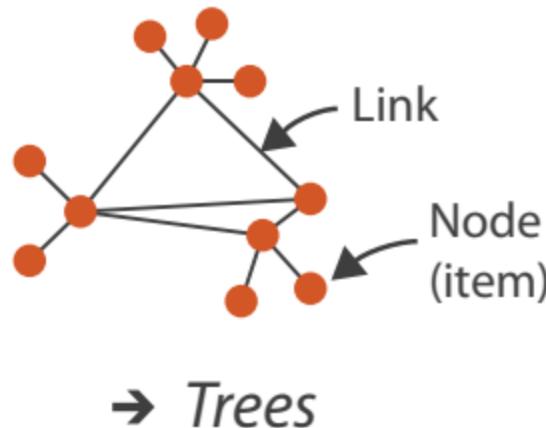
→ *Multidimensional Table*



- multidimensional tables
 - indexing based on multiple keys

Dataset Types: Networks and Graphs

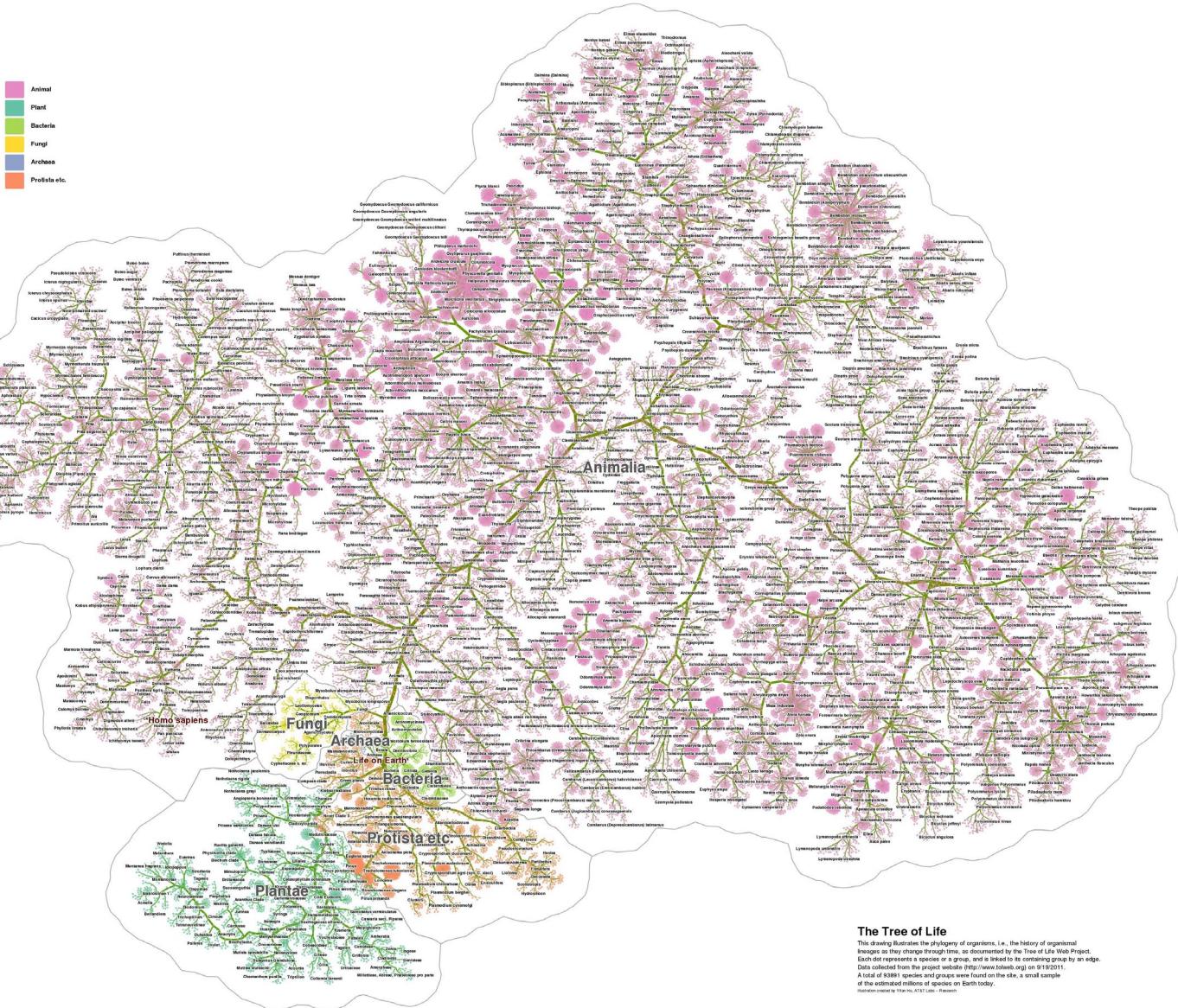
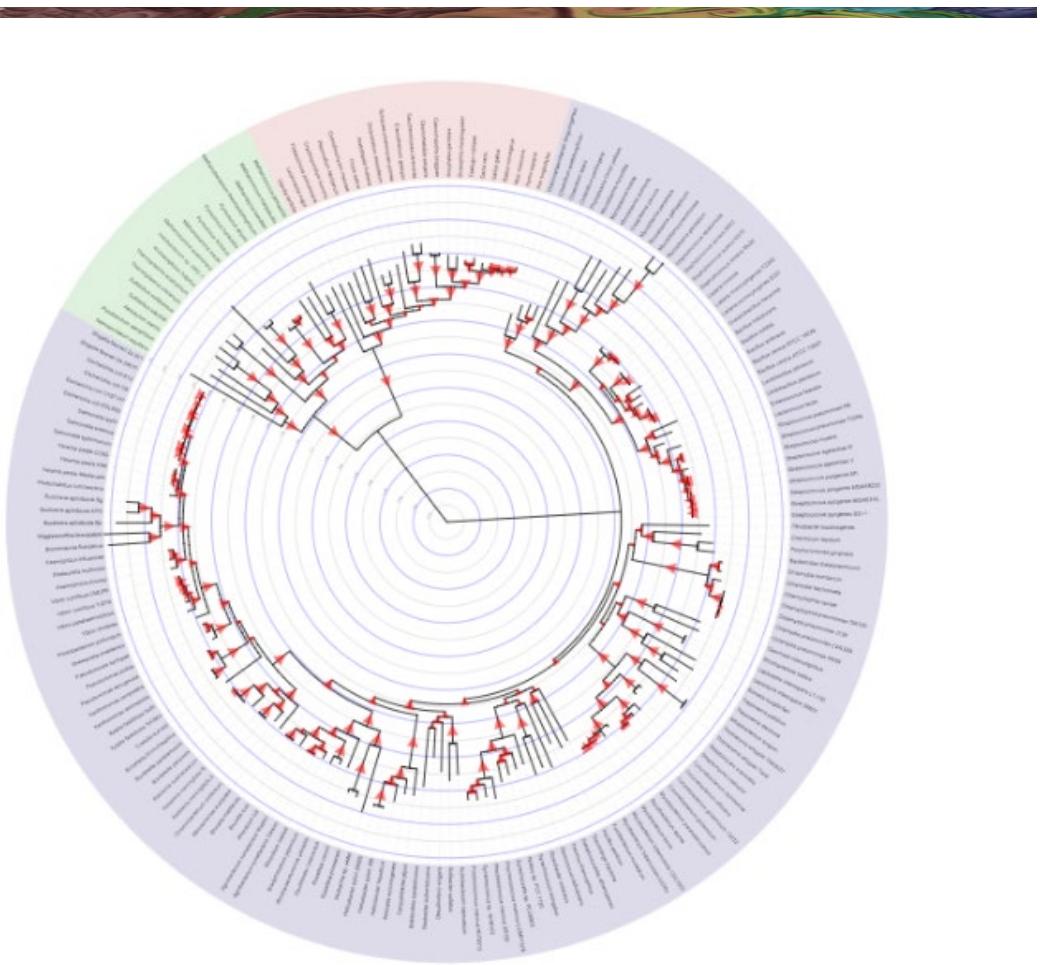
→ Networks



Networks &
Trees
Items (nodes)
Links
Attributes

- **network/graph**
 - nodes (vertices) connected by links (edges)
 - tree is special case: no cycles
 - often have roots and are directed

Visualizing Networks



The Tree of Life
This diagram illustrates the history of organisms, i.e., the history of organismal lineages as they change through time, as documented by the Tree of Life Web Project. Each dot represents a species or a group, and is linked to its containing group by an edge. Data is current as of the project's date (<http://www.tolweb.org>) on 9/19/2011. A total of 50891 species and groups are found on the site, a small sample of the estimated millions of species on Earth today.
Illustration created by Yiran Yu, AT&T Labs - Research

<https://itol.embl.de/>

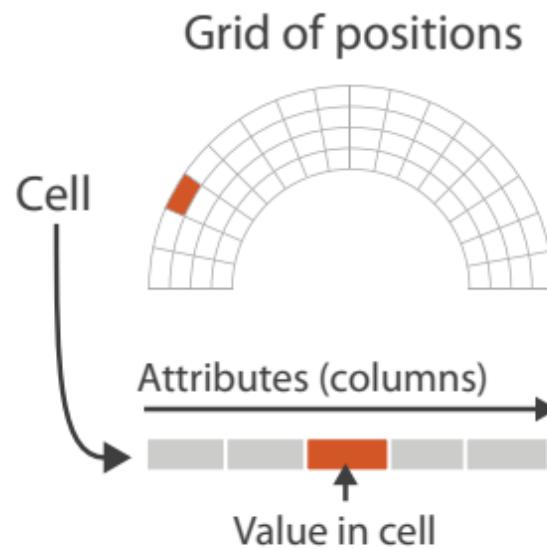
A phylogenetic tree is a diagram that represents evolutionary relationships among organisms.



Dataset Types: Fields

→ Spatial

→ Fields (Continuous)



Fields

Grids

Positions

Attributes

- attribute values associated with cells
- cell contains value from continuous domain
 - eg temperature, pressure, wind velocity
- measured or simulated

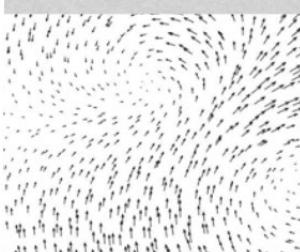
Spatial Fields

Field data

scalar



vector



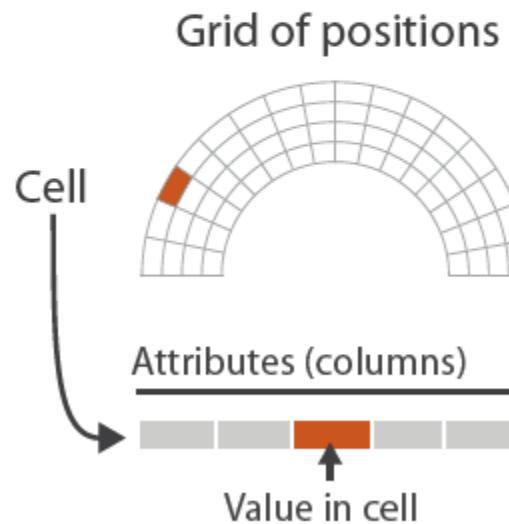
tensor



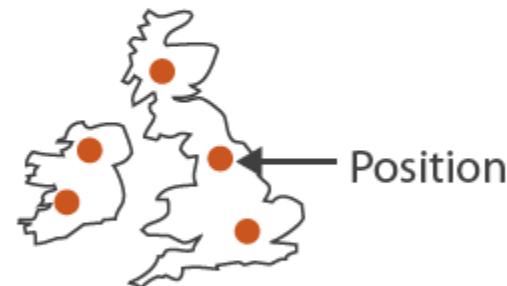
Dataset Types: Geometry

→ Spatial

→ Fields (Continuous)



→ Geometry (Spatial)



Geometry

Items

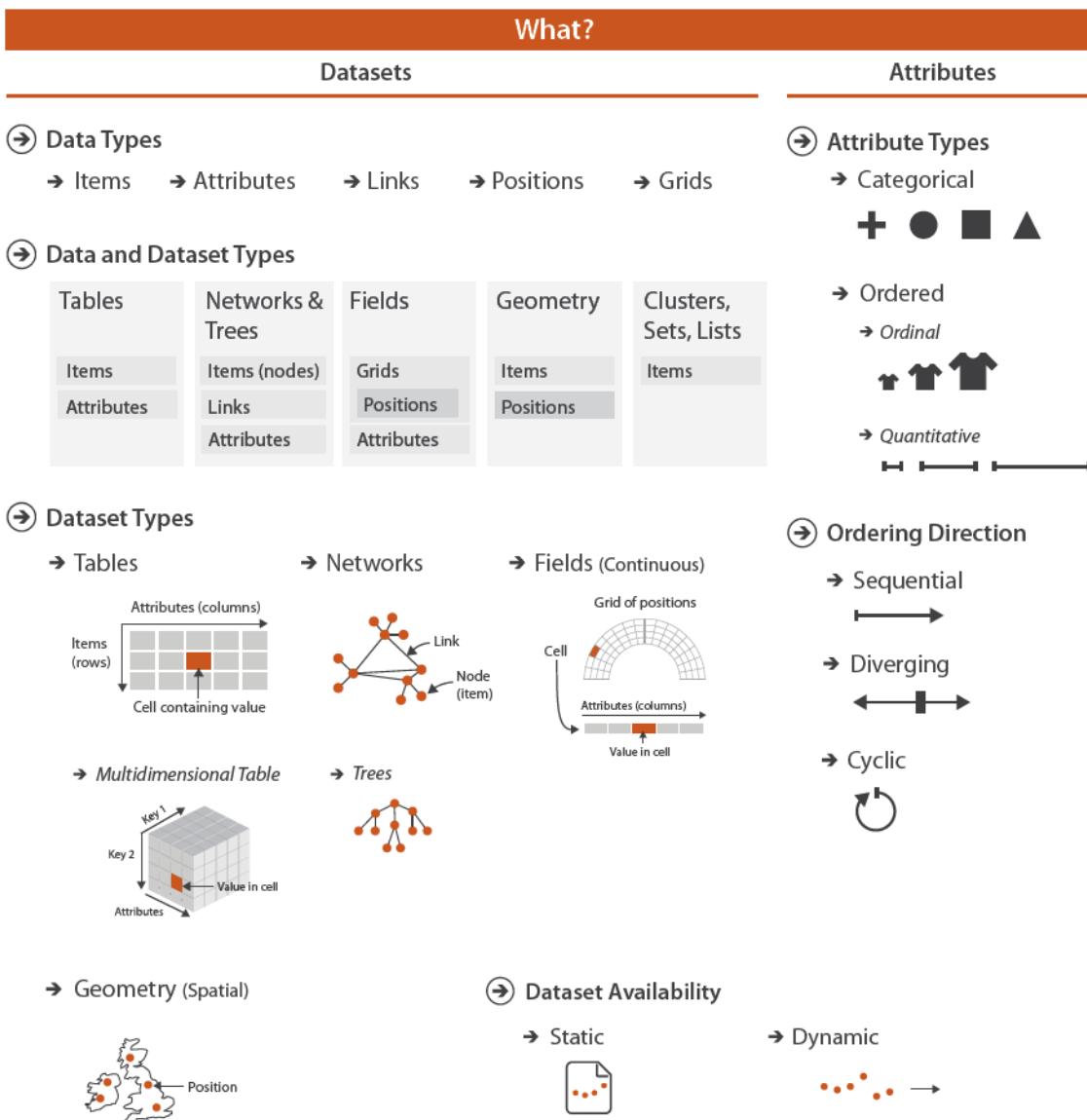
Positions

Attribute Types

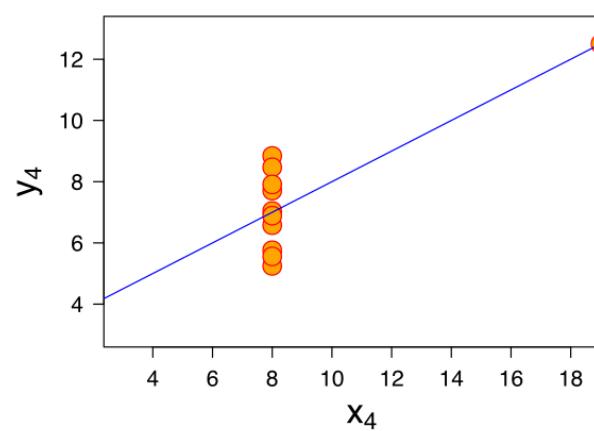
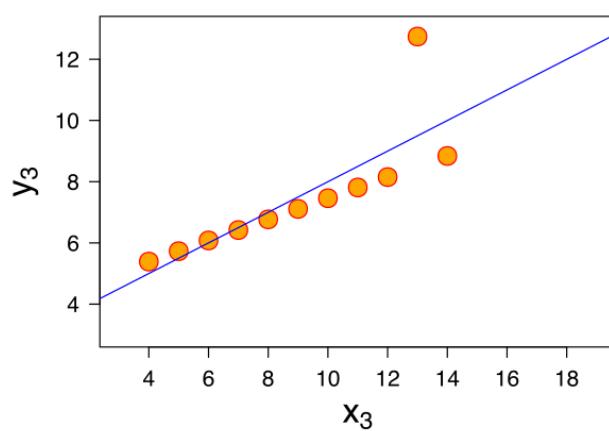
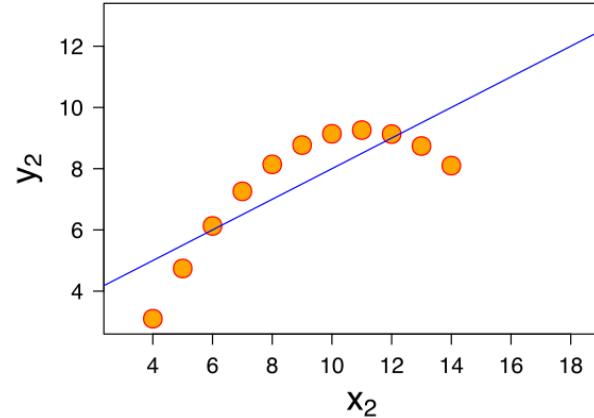
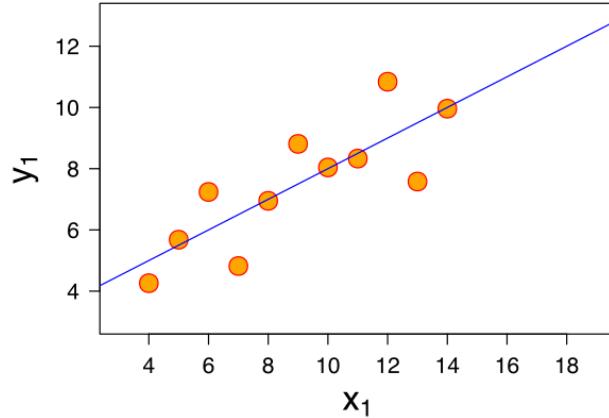
- which classes of values & measurements?
- categorical (nominal)
 - compare equality
 - no implicit ordering
- ordered
 - ordinal
 - less/greater than defined
 - quantitative
 - meaningful magnitude
 - arithmetic possible



A Taxonomy of Data



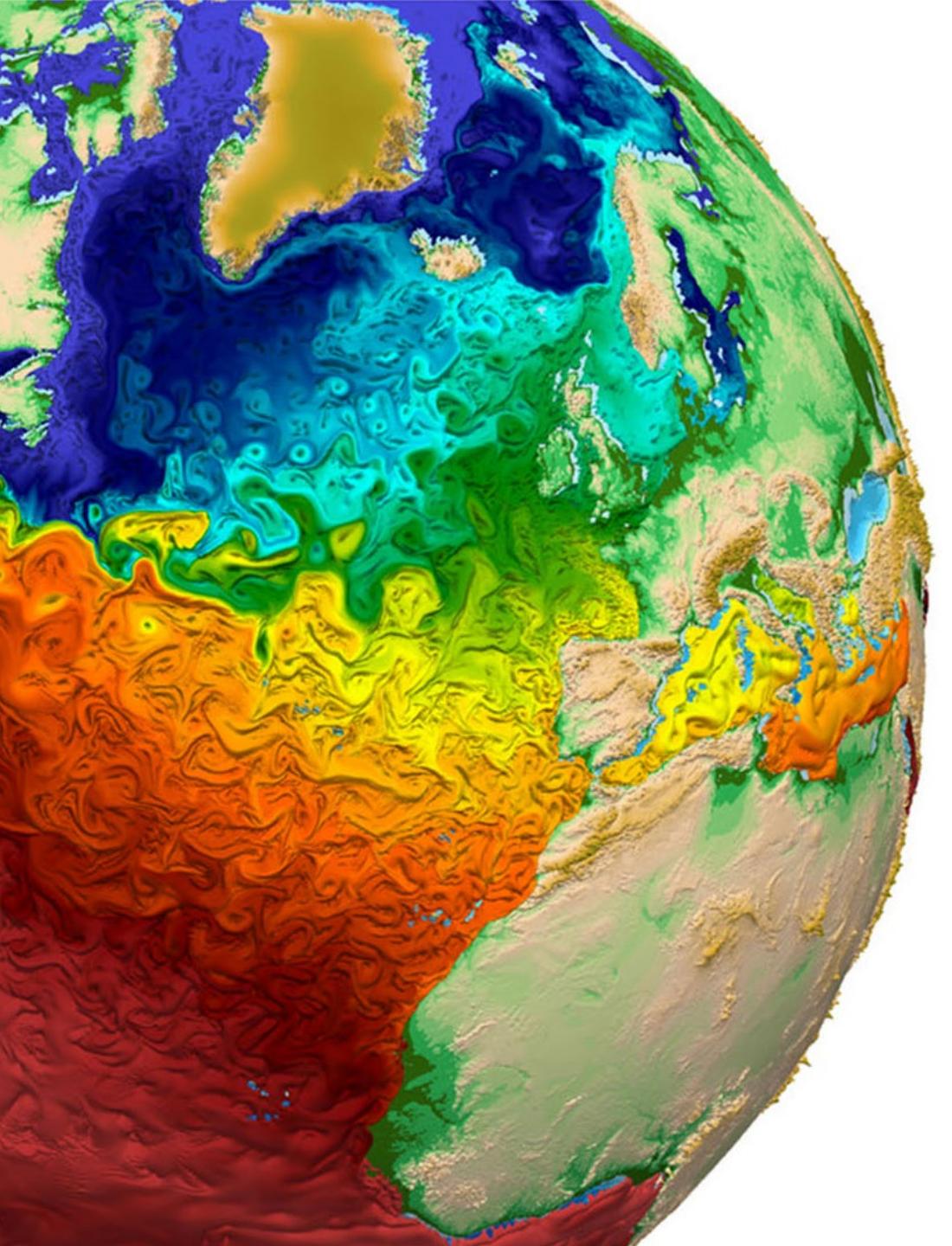
Value of Visualization: Anscombe's Quartet



They were constructed in 1973 by the [statistician Francis Anscombe](#) to demonstrate both the importance of graphing data before analyzing it and the effect of [outliers](#) and other [influential observations](#) on statistical properties. He described the article as being intended to counter the impression among statisticians that "numerical calculations are exact, but graphs are rough."

-- Wikipedia

Mean and variance are the same for all 4 data sets



Data Science for People in a Hurry

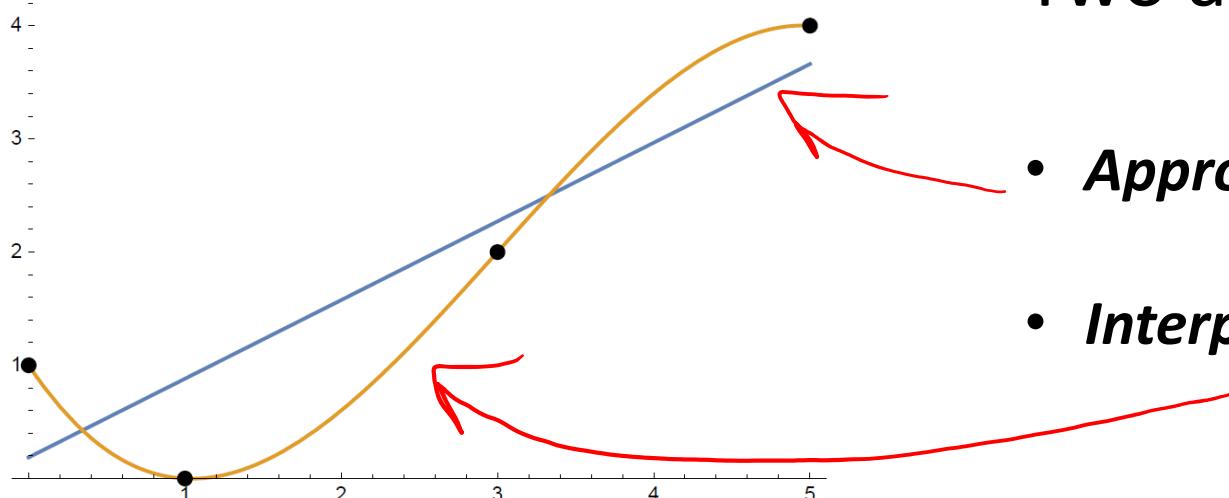
Linear Interpolation

Scientific Visualization
Professor Eric Shaffer

What is Interpolation?

Two approaches to data fitting

- **Approximation**...captures the behavior of the data
- **Interpolation**...matches the observed data exactly

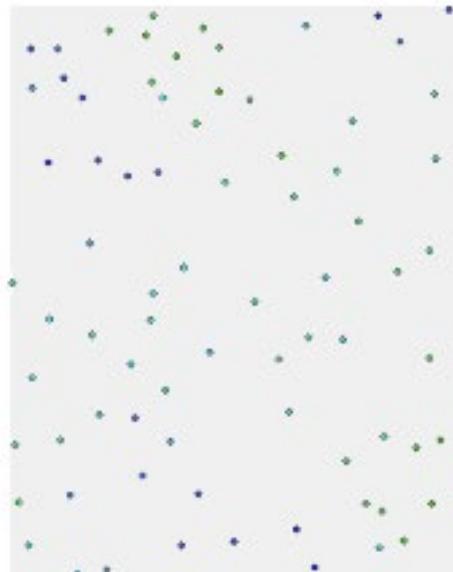


“Interpolation simply means fitting some function to given data so that the function has the same values as the given data.” – Professor Michael T. Heath.

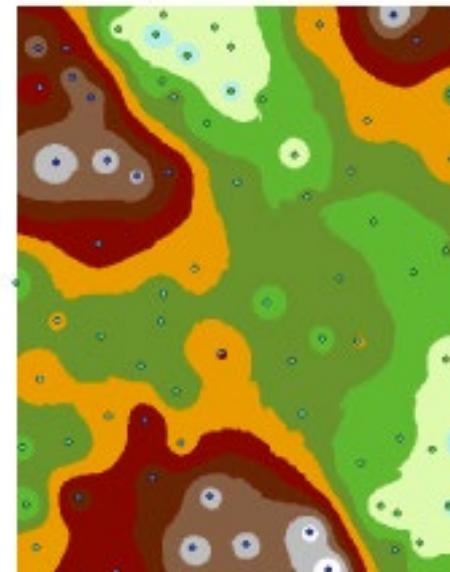
And Why Are We Data Fitting?

- Often have empirical data...sampled values in some domain
- Would like to fill in unknown values in the domain
- Interpolation constructs a function that can fill in these unknown values

Sampled Elevations



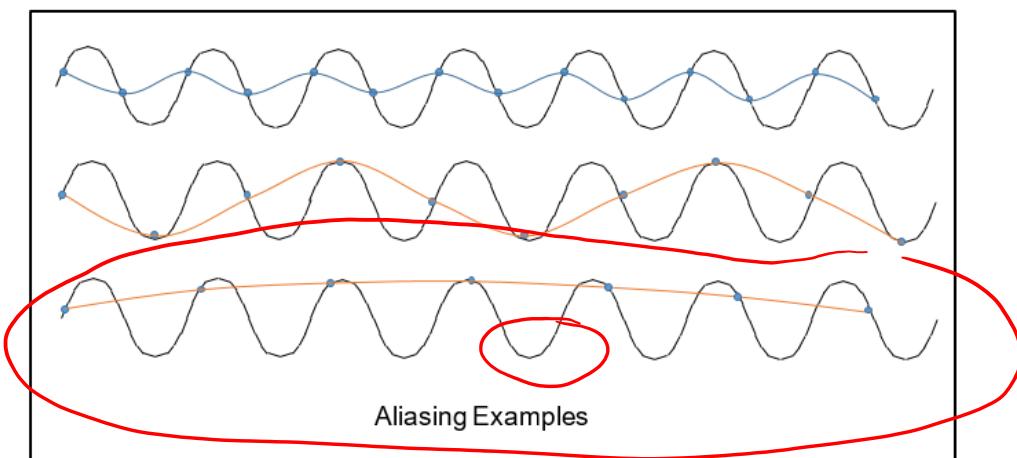
Interpolated Elevations



Why Linear Interpolation

- Simple....conceptually and computationally
- With no other information about underlying function...linear is fine

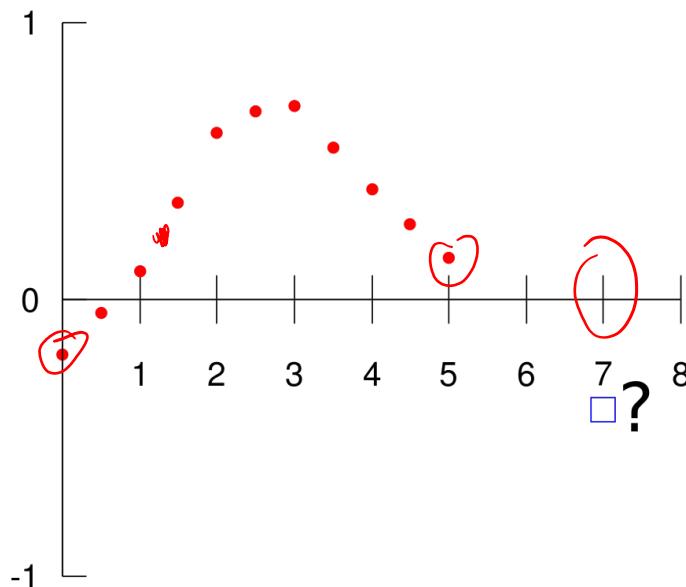
...but if you know the underlying function is not linear...fit with a non-linear function



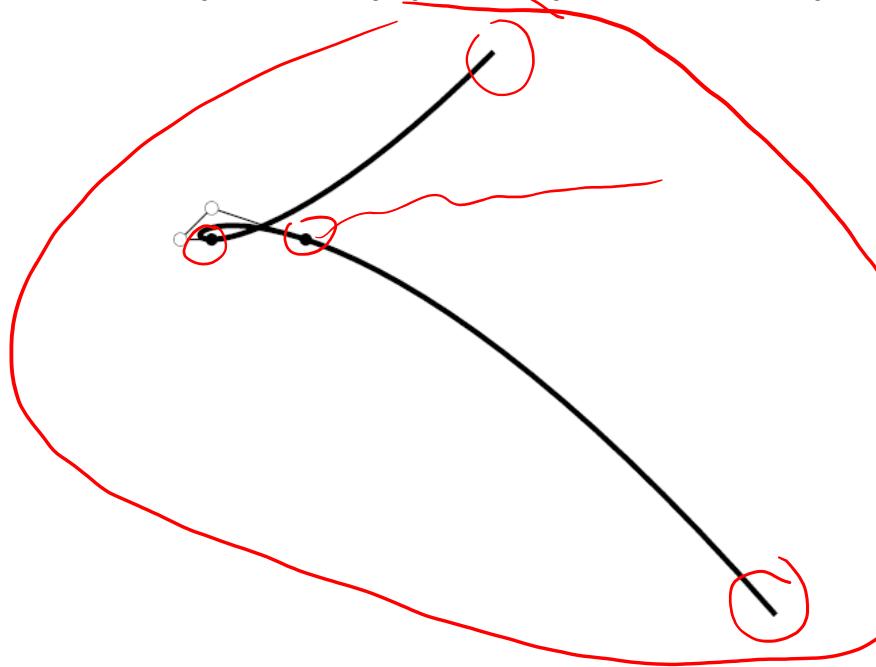
In signal processing and related disciplines, aliasing is an effect that causes different signals to become indistinguishable when sampled. It also often refers to the distortion or artifact that results when a signal reconstructed from samples is different from the original continuous signal. [Wikipedia](#)

Extrapolation

Inferring unknown values beyond the range of known values



Interpolative methods may perform especially poorly for extrapolation



Linear Interpolation of Position

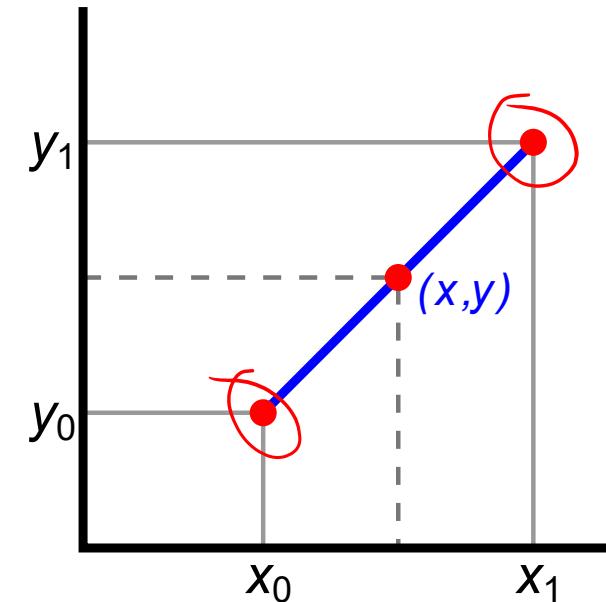
We have initial position of $p_0 = (x_0, y_0)$ and a final position of $p_1 = (x_1, y_1)$

Can generate intermediate positions using a parameterized linear function

$$P(t) = (1-t) p_0 + t p_1$$

$$x(t)$$

$$x_0 (1-t) + x_1 t$$



Linear Interpolation of Function Values

If we have function values sampled at points p_0 and p_1

$$f(p_0) = v_0 \quad \text{and} \quad f(p_1) = v_1$$

We can find $f(t) = (1-t)v_0 + t v_1$

What if we are given a point p_i on the line and don't know t ?

$$t = \frac{\text{dist}(p_i, p_0)}{\text{dist}(p_1, p_0)}$$

Linear Interpolation of Function Values: Example

$$f(x) = 4$$

(0, 0)

$$f(t) 20$$

(7.5, 0)

(10, 0)

$$t = \frac{(7.5 - 0)}{(10 - 0)} = 0.75$$

$$\begin{aligned} f(7.5) &\approx (1 - 0.75)4 + (0.75)20 \\ &\approx 1 + 15 = 116 \end{aligned}$$

Bilinear Interpolation

Assume we know a function value at the four points

$$Q_{11} = (x_1, y_1), Q_{12} = (x_1, y_2),$$

$$Q_{21} = (x_2, y_1), Q_{22} = (x_2, y_2)$$

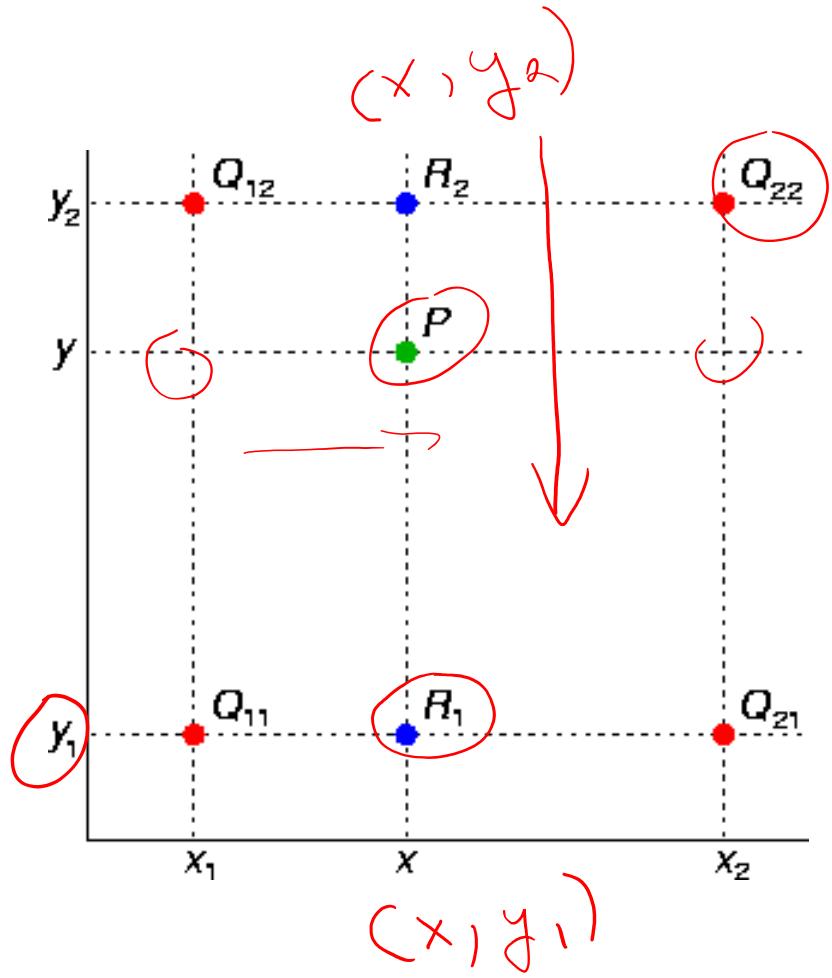
We first do linear interpolation in the x-direction

- Find function values at R_1 and R_2

Then in the y direction

- Interpolate between R_1 and R_2 to find value at P

Order in which you interpolate (which axis goes 1st) does not matter.



Bilinear Interpolation

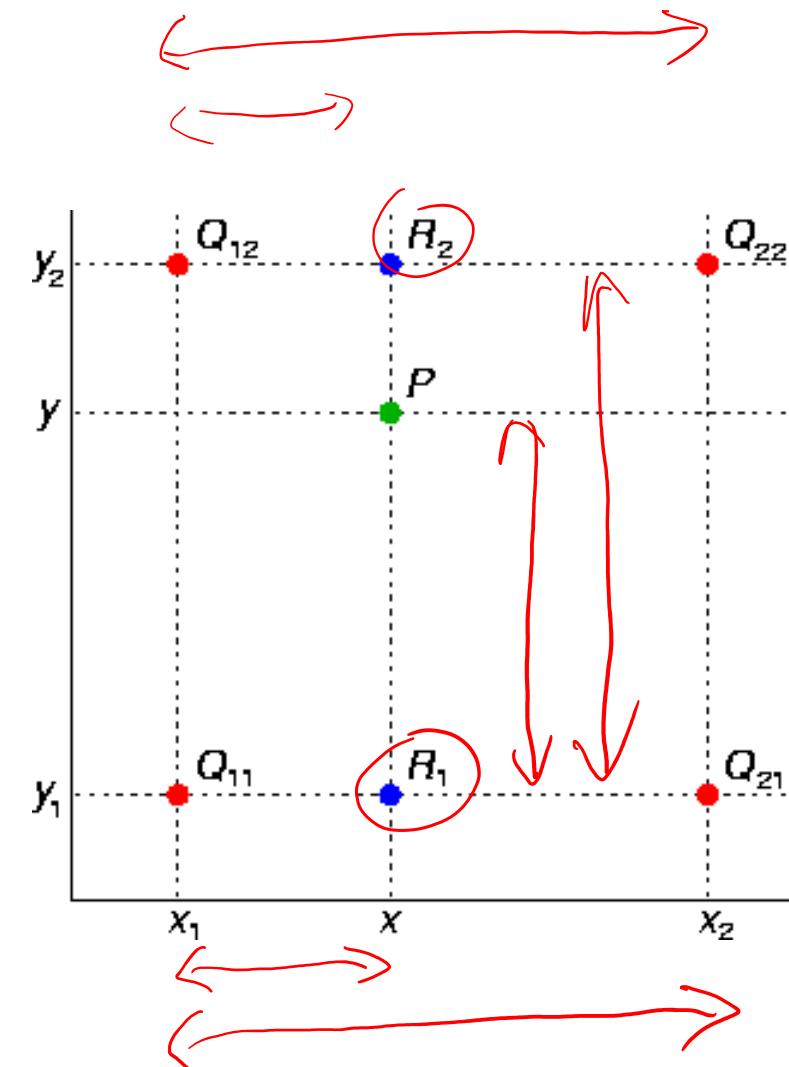
(1 - t)

$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}),$$

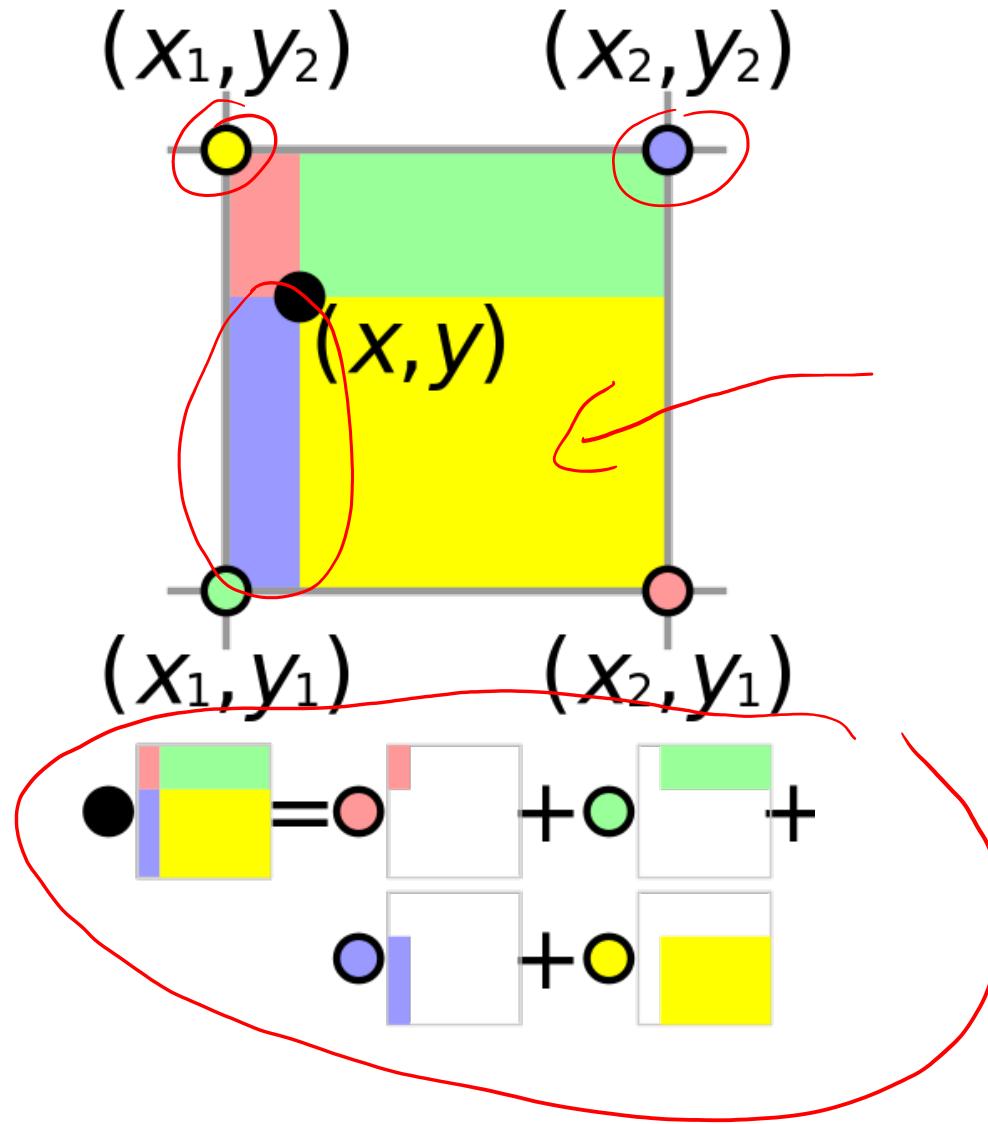
$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).$$

$$f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2)$$

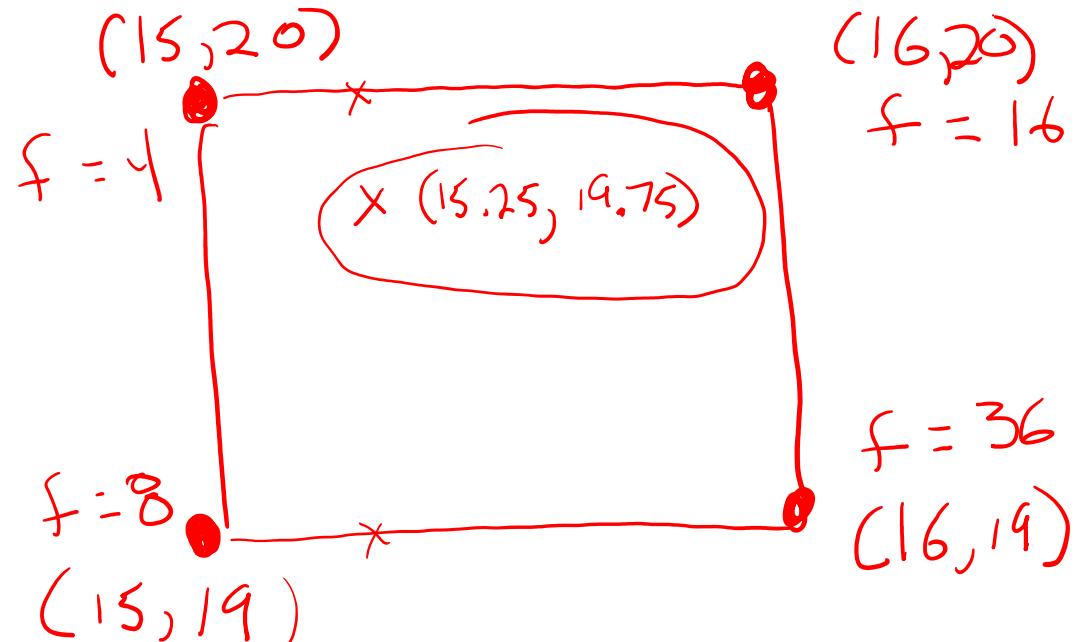
from Wikipedia



Bilinear Interpolation



Bilinear Interpolation: Example

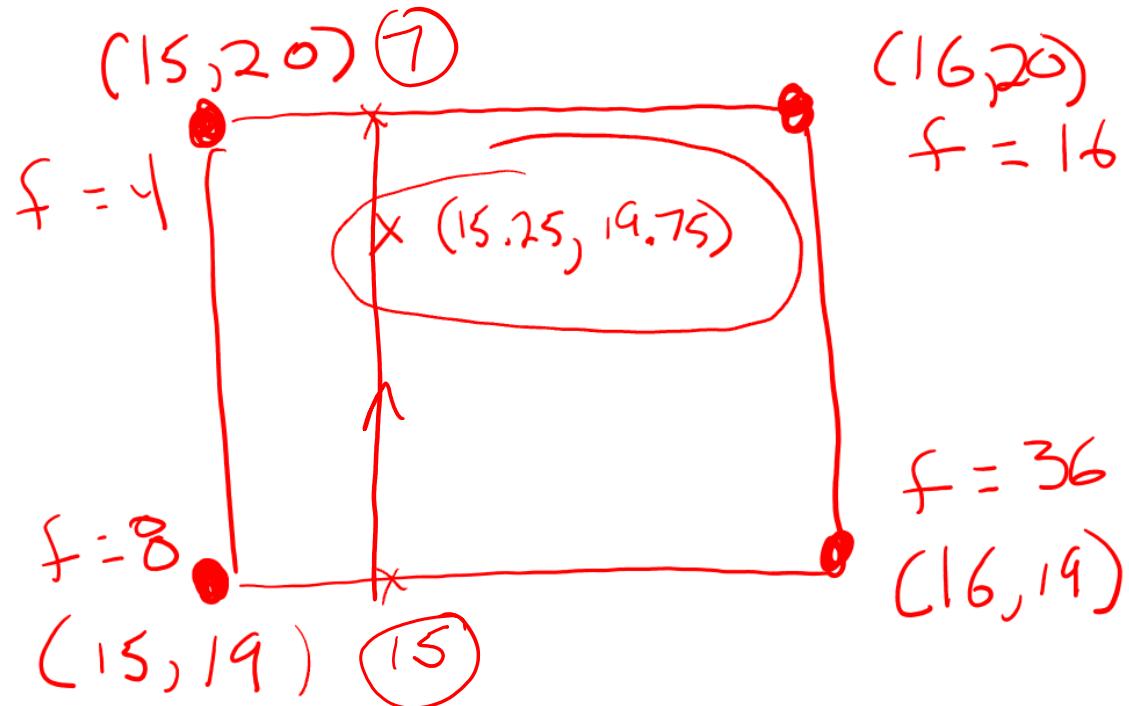


$$t = \frac{15.25 - 15}{16 - 15} = 0.25$$

$$\begin{aligned}f(15.25, 20) &\approx (1 - 0.25)4 + .25(16) \\&\approx 3 + 4 = 7\end{aligned}$$

$$\begin{aligned}f(15.25, 19) &\approx (1 - 0.25)8 + .25(36) \\&= 6 + 9 = 15\end{aligned}$$

Bilinear Interpolation: Example



$$\begin{aligned}f(15.25, 19.75) &\approx \\(1 - 0.75) \cdot 15 + (.75) \cdot 7 \\&= \frac{15}{4} + \frac{21}{4} = \frac{36}{4} \\&= \boxed{9}\end{aligned}$$

$$t = \frac{(19.75 - 19)}{(20 - 19)} = 0.75$$

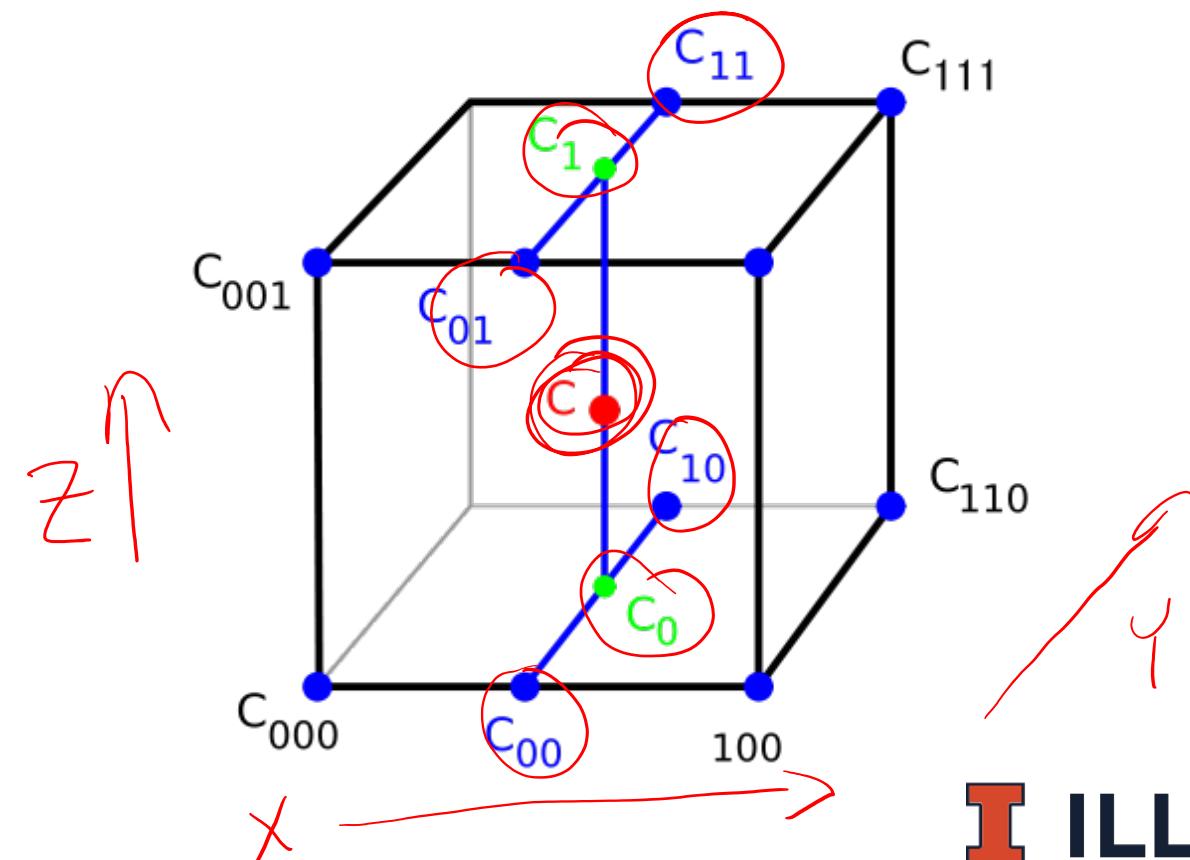
Trilinear Interpolation

First interpolate in x to find c_{00} , c_{01} , c_{10} , and c_{11}

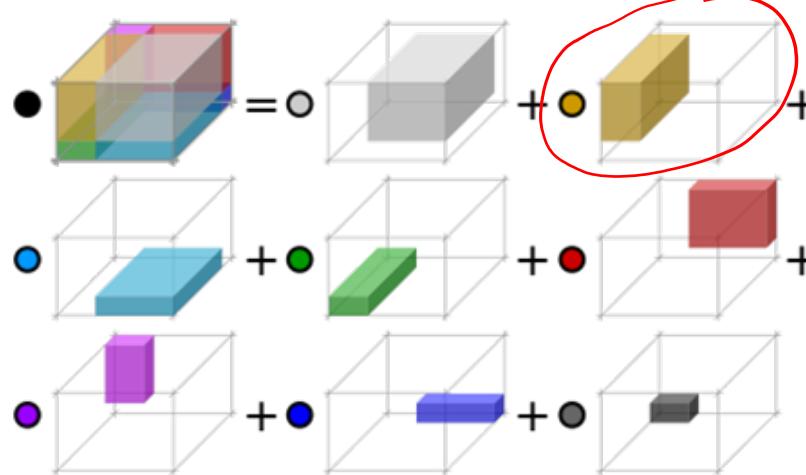
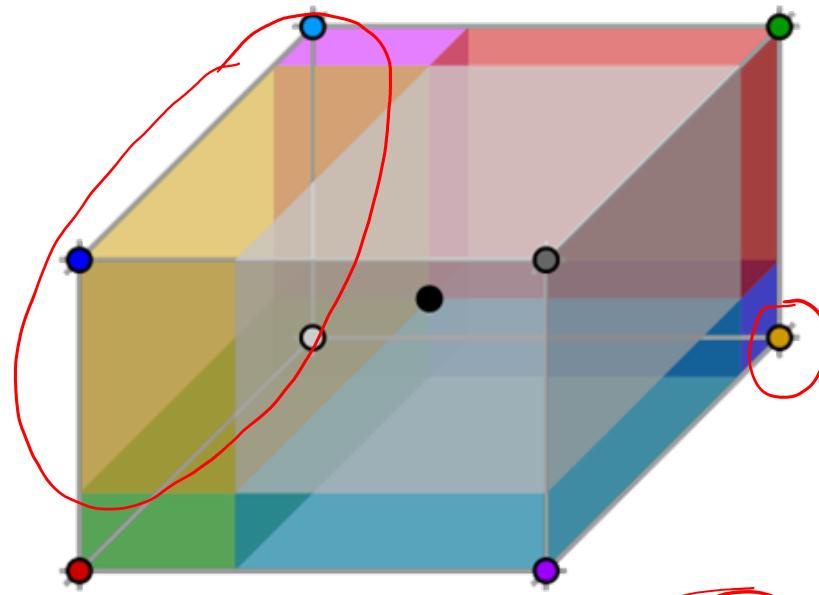
Then in y to find C_0 and C_1

And then in z to find C

Order in which you interpolate (which axis goes 1st, 2nd, 3rd) does not matter.



Trilinear Interpolation



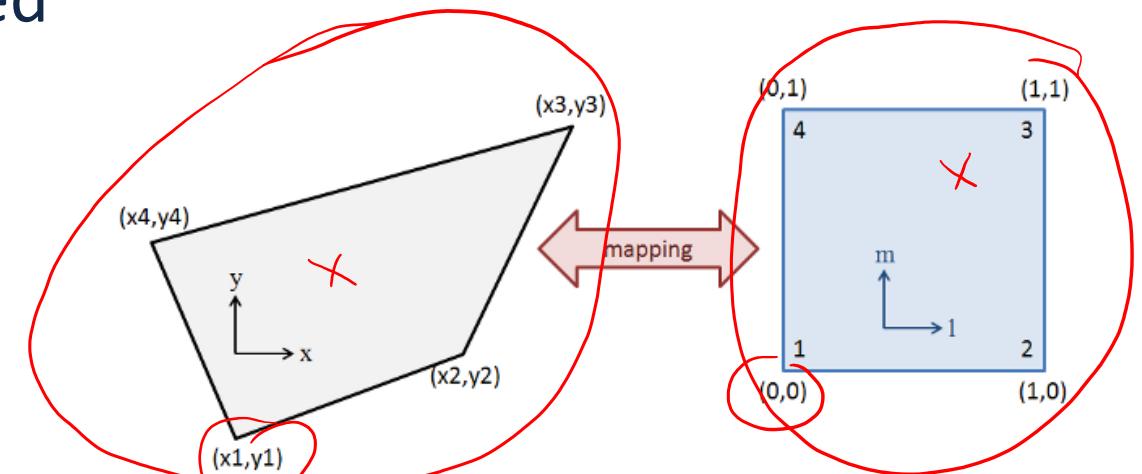
Interpolation and Cell Shape

Can you use bilinear interpolation on an arbitrary quadrilateral?

- Yes, but not the formulas we have used
- Those only apply to rectangles

We can apply the formula we know

...if we can map from an arbitrary quad to a reference quad



Bilinear Interpolation over Arbitrary Quadrilaterals

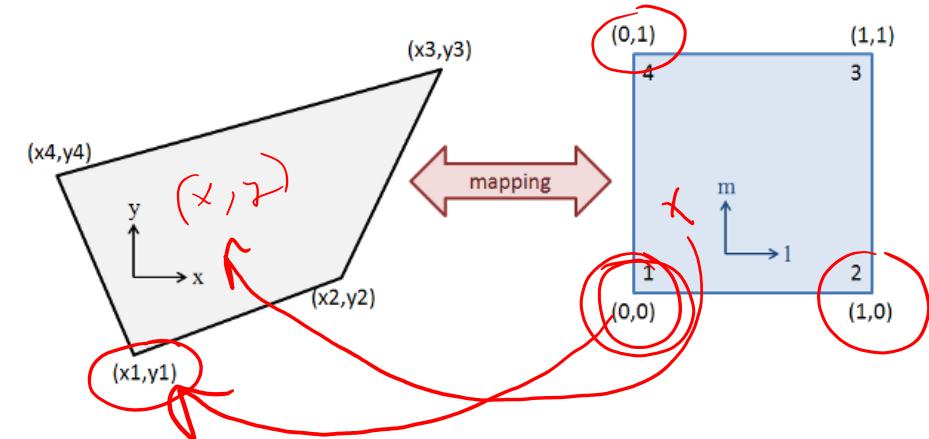
Find a mapping from reference to arbitrary quad
Given (l, m) coordinates we can find (x, y)

$$x = \alpha_1 + \alpha_2 l + \alpha_3 m + \alpha_4 lm$$

$$y = \beta_1 + \beta_2 l + \beta_3 m + \beta_4 lm$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix}$$

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$



Similarly solve for $y = \beta_1 + \beta_2 l + \beta_3 m + \beta_4 lm$

Bilinear Interpolation over Arbitrary Quadrilaterals

To interpolate, we need to go from (x,y) to (l,m)

$$x = \alpha_1 + \alpha_2 l + \alpha_3 m + \alpha_4 lm$$

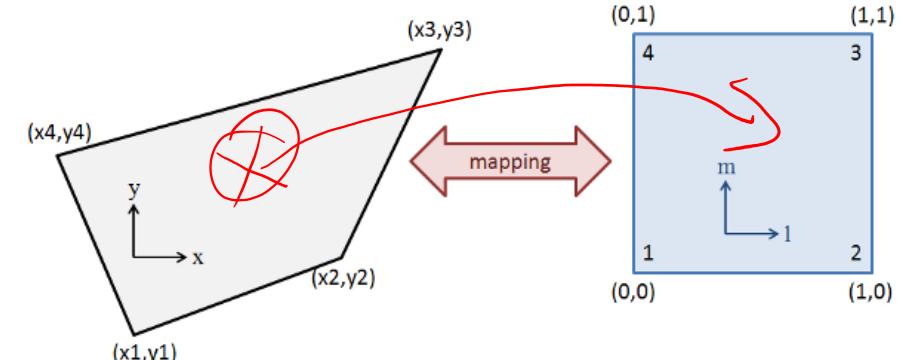
$$l = \left(\frac{x - \alpha_1 - \alpha_3 m}{\alpha_2 + \alpha_4 m} \right)$$

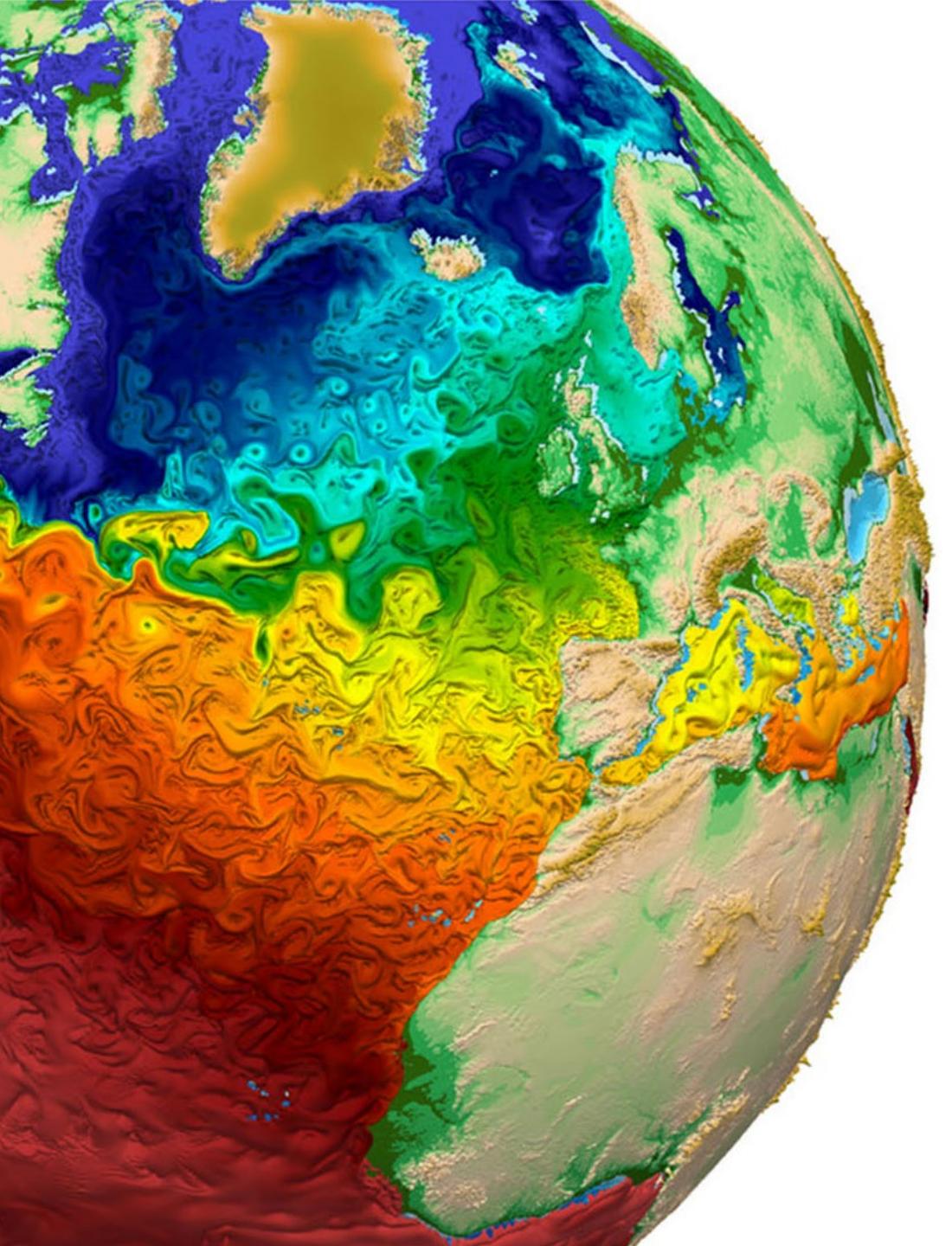
Substitute into $y = \beta_1 + \beta_2 l + \beta_3 m + \beta_4 lm$

$$(\alpha_4 \beta_3 - \alpha_3 \beta_4) m^2 + (\alpha_4 \beta_1 - \alpha_1 \beta_4 + \alpha_2 \beta_3 - \alpha_3 \beta_2 + x \beta_4 - y \alpha_4) m + (\alpha_2 \beta_1 - \alpha_1 \beta_2 + x \beta_2 - y \alpha_2) = 0$$

Which can solved with

$$m = \frac{(-b + \sqrt{b^2 - 4ac})}{2a}$$





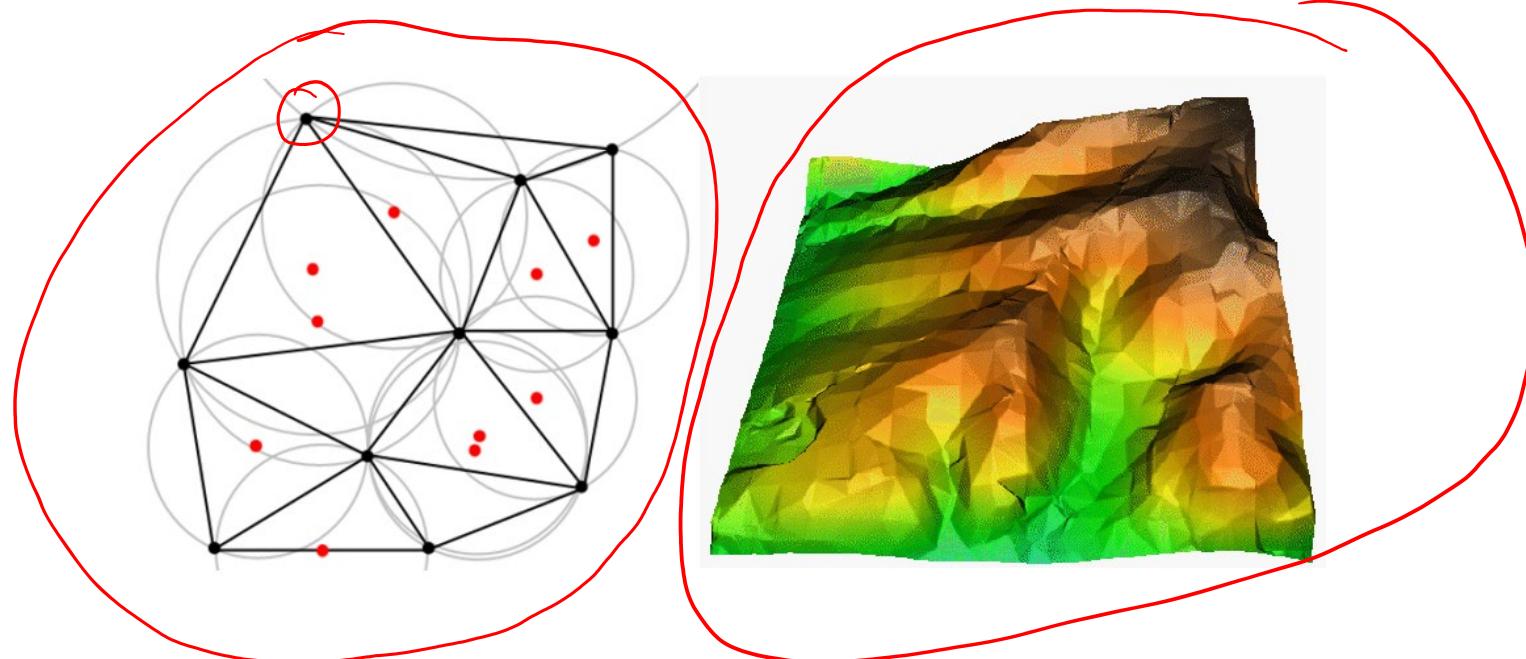
Data Science for People in a Hurry

Barycentric Coordinates and Interpolation

Scientific Visualization
Professor Eric Shaffer

Barycentric Interpolation

How can we linearly interpolate a function over triangles?

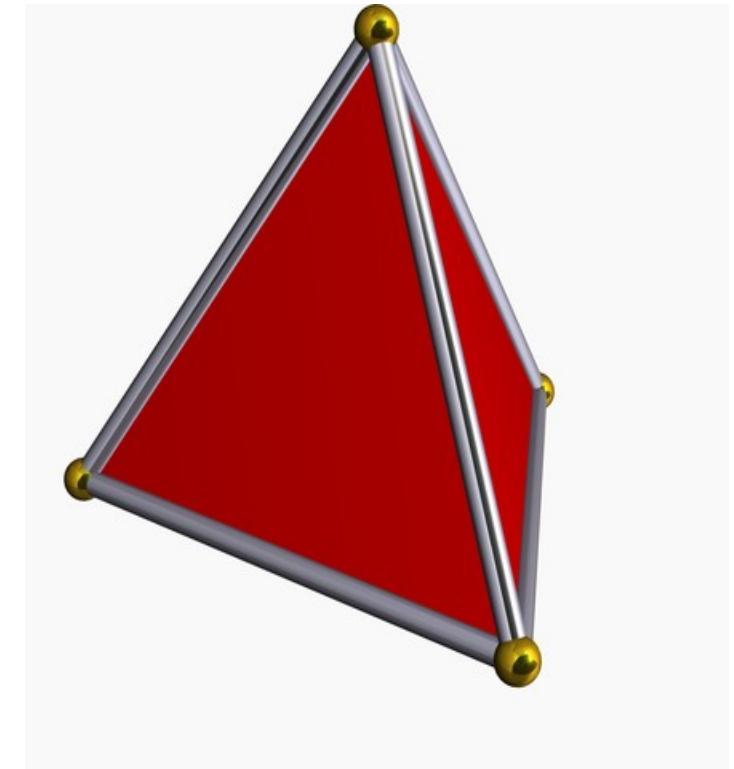


Can be useful:

- Scattered data can be triangulated and then the function interpolated
- Many datasets involve triangulated domains already

Barycentric Interpolation

- Barycentric coordinates apply to more than just triangles
- Used on any simplex
- A simplex is a convex hull of $k+1$ points in a k -dimensional space
 - Simplest convex “polygon” in a k -dimensional space
 - A 3-simplex is a triangle
- Barycentric coordinates provide a way to interpolate over simplices



Barycentric Coordinates for Triangles

Describe location of a point in relation to the vertices of a given triangle

Express point p in barycentric coordinates $p=(\lambda_1, \lambda_2, \lambda_3)$

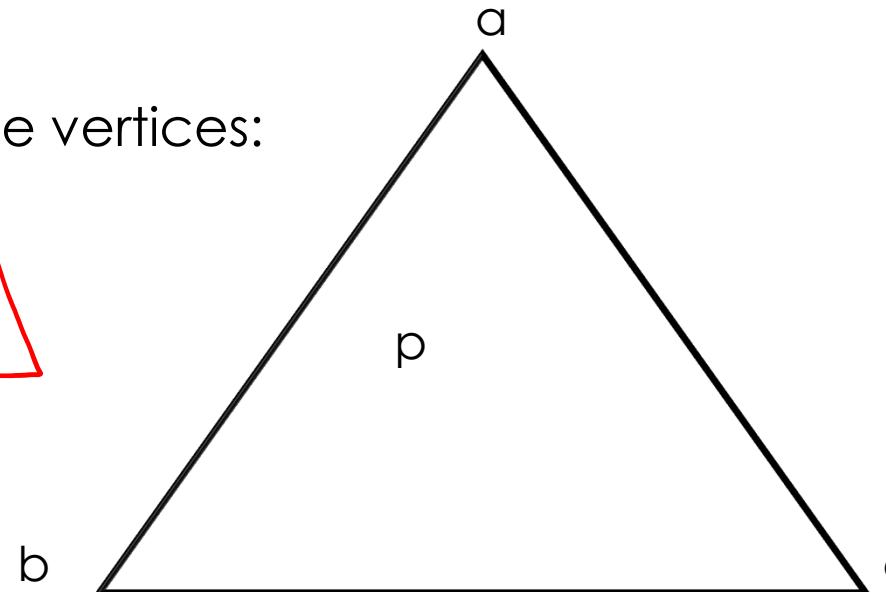
The following must be true

$$p = \lambda_1 a + \lambda_2 b + \lambda_3 c$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

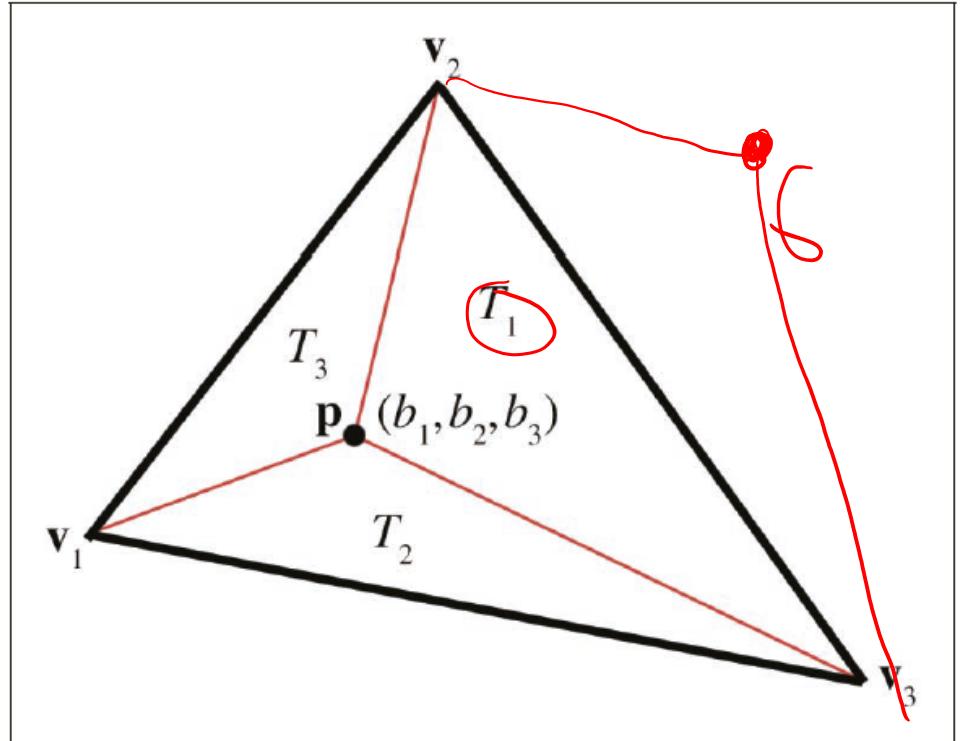
To interpolate a function sampled at the vertices:

$$f(p) = \lambda_1 f(a) + \lambda_2 f(b) + \lambda_3 f(c)$$



Computing Barycentric Coordinates for Triangles

Coordinates are the signed area of the opposite subtriangle divided by area of the triangle



$$b_1 = A(T_1)/A(T),$$

$$b_2 = A(T_2)/A(T),$$

$$b_3 = A(T_3)/A(T)$$

$$b_1 x_1 + b_2 x_2 + b_3 x_3 = p_x,$$

$$b_1 y_1 + b_2 y_2 + b_3 y_3 = p_y,$$

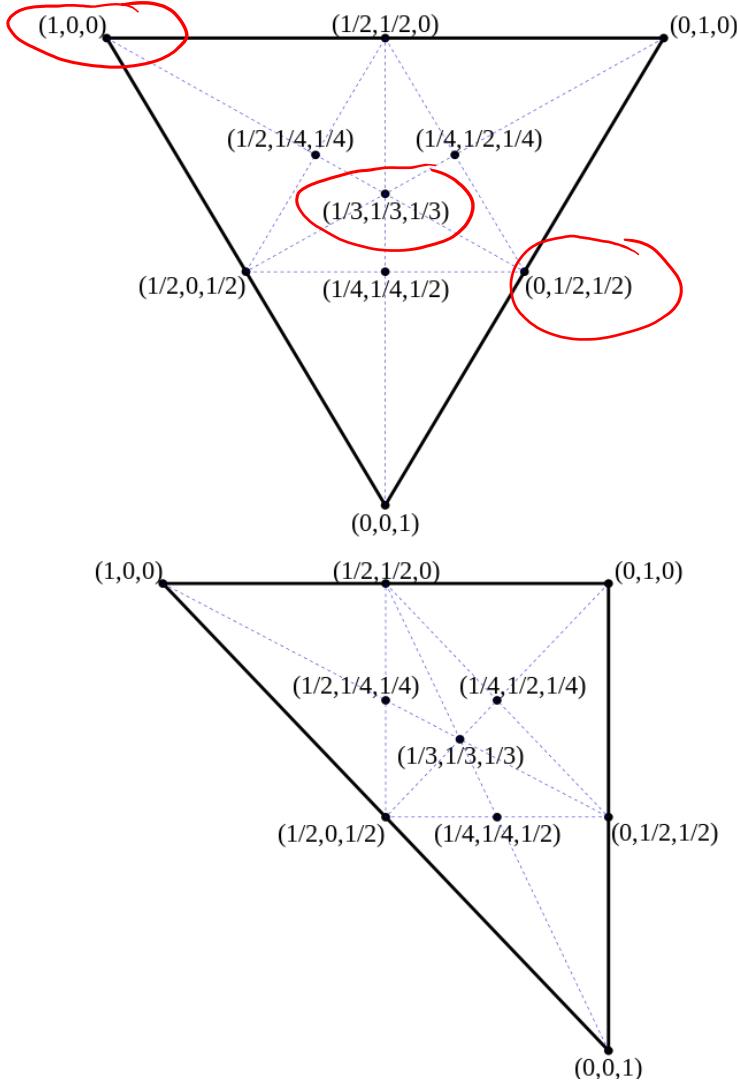
$$b_1 + b_2 + b_3 = 1.$$

$$b_1 = \frac{(p_y - y_3)(x_2 - x_3) + (y_2 - y_3)(x_3 - p_x)}{(y_1 - y_3)(x_2 - x_3) + (y_2 - y_3)(x_3 - x_1)},$$

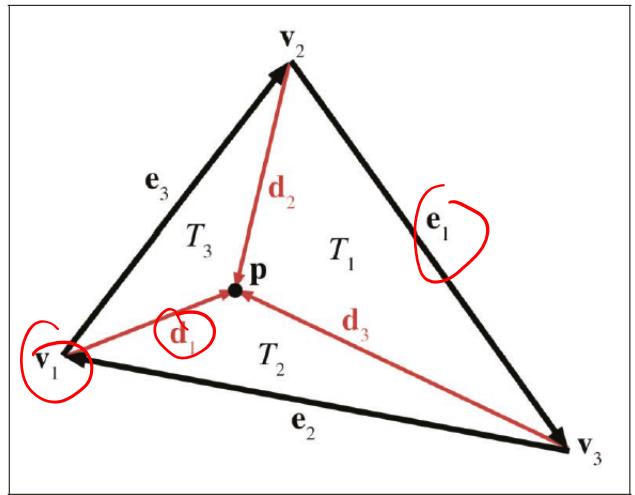
$$b_2 = \frac{(p_y - y_1)(x_3 - x_1) + (y_3 - y_1)(x_1 - p_x)}{(y_1 - y_3)(x_2 - x_3) + (y_2 - y_3)(x_3 - x_1)},$$

$$b_3 = \frac{(p_y - y_2)(x_1 - x_2) + (y_1 - y_2)(x_2 - p_x)}{(y_1 - y_3)(x_2 - x_3) + (y_2 - y_3)(x_3 - x_1)}.$$

Some Important Points



Computing Coordinates for 3D Triangles



$$\mathbf{e}_1 = \mathbf{v}_3 - \mathbf{v}_2,$$

$$\mathbf{d}_1 = \mathbf{p} - \mathbf{v}_1,$$

$$\mathbf{e}_2 = \mathbf{v}_1 - \mathbf{v}_3,$$

$$\mathbf{d}_2 = \mathbf{p} - \mathbf{v}_2,$$

$$\mathbf{e}_3 = \mathbf{v}_2 - \mathbf{v}_1,$$

$$\mathbf{d}_3 = \mathbf{p} - \mathbf{v}_3.$$

$$\hat{\mathbf{n}} = \frac{\mathbf{e}_1 \times \mathbf{e}_2}{\|\mathbf{e}_1 \times \mathbf{e}_2\|}.$$

$$A(T) = ((\mathbf{e}_1 \times \mathbf{e}_2) \cdot \hat{\mathbf{n}})/2,$$

$$A(T_1) = ((\mathbf{e}_1 \times \mathbf{d}_3) \cdot \hat{\mathbf{n}})/2,$$

$$A(T_2) = ((\mathbf{e}_2 \times \mathbf{d}_1) \cdot \hat{\mathbf{n}})/2,$$

$$A(T_3) = ((\mathbf{e}_3 \times \mathbf{d}_2) \cdot \hat{\mathbf{n}})/2.$$

$$b_1 = A(T_1)/A(T) = \frac{(\mathbf{e}_1 \times \mathbf{d}_3) \cdot \hat{\mathbf{n}}}{(\mathbf{e}_1 \times \mathbf{e}_2) \cdot \hat{\mathbf{n}}},$$

$$b_2 = A(T_2)/A(T) = \frac{(\mathbf{e}_2 \times \mathbf{d}_1) \cdot \hat{\mathbf{n}}}{(\mathbf{e}_1 \times \mathbf{e}_2) \cdot \hat{\mathbf{n}}},$$

$$b_3 = A(T_3)/A(T) = \frac{(\mathbf{e}_3 \times \mathbf{d}_2) \cdot \hat{\mathbf{n}}}{(\mathbf{e}_1 \times \mathbf{e}_2) \cdot \hat{\mathbf{n}}}.$$

Barycentric Coordinates for Tetrahedra

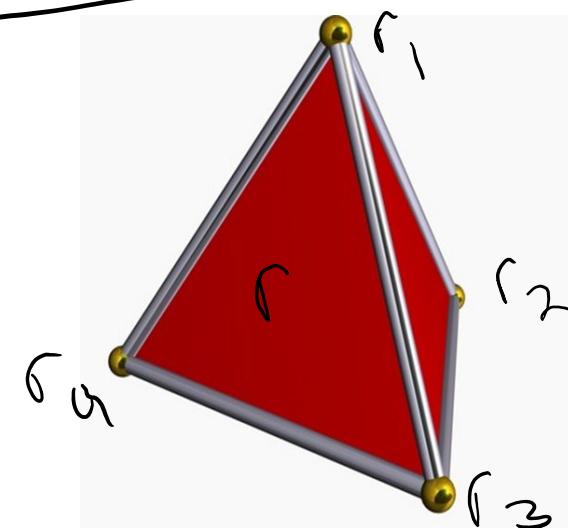
We have 4 vertices of a tetrahedron r_1, r_2, r_3 , and r_4

To find the coordinates of a point r we can compute

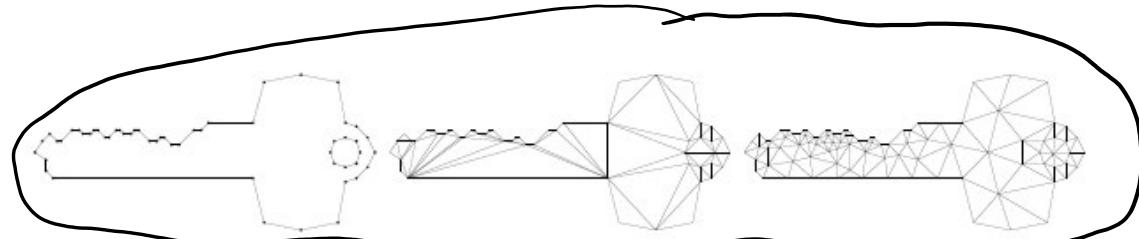
$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \mathbf{T}^{-1}(\mathbf{r} - \mathbf{r}_4)$$

$$\mathbf{T} = \begin{pmatrix} x_1 - x_4 & x_2 - x_4 & x_3 - x_4 \\ y_1 - y_4 & y_2 - y_4 & y_3 - y_4 \\ z_1 - z_4 & z_2 - z_4 & z_3 - z_4 \end{pmatrix}$$

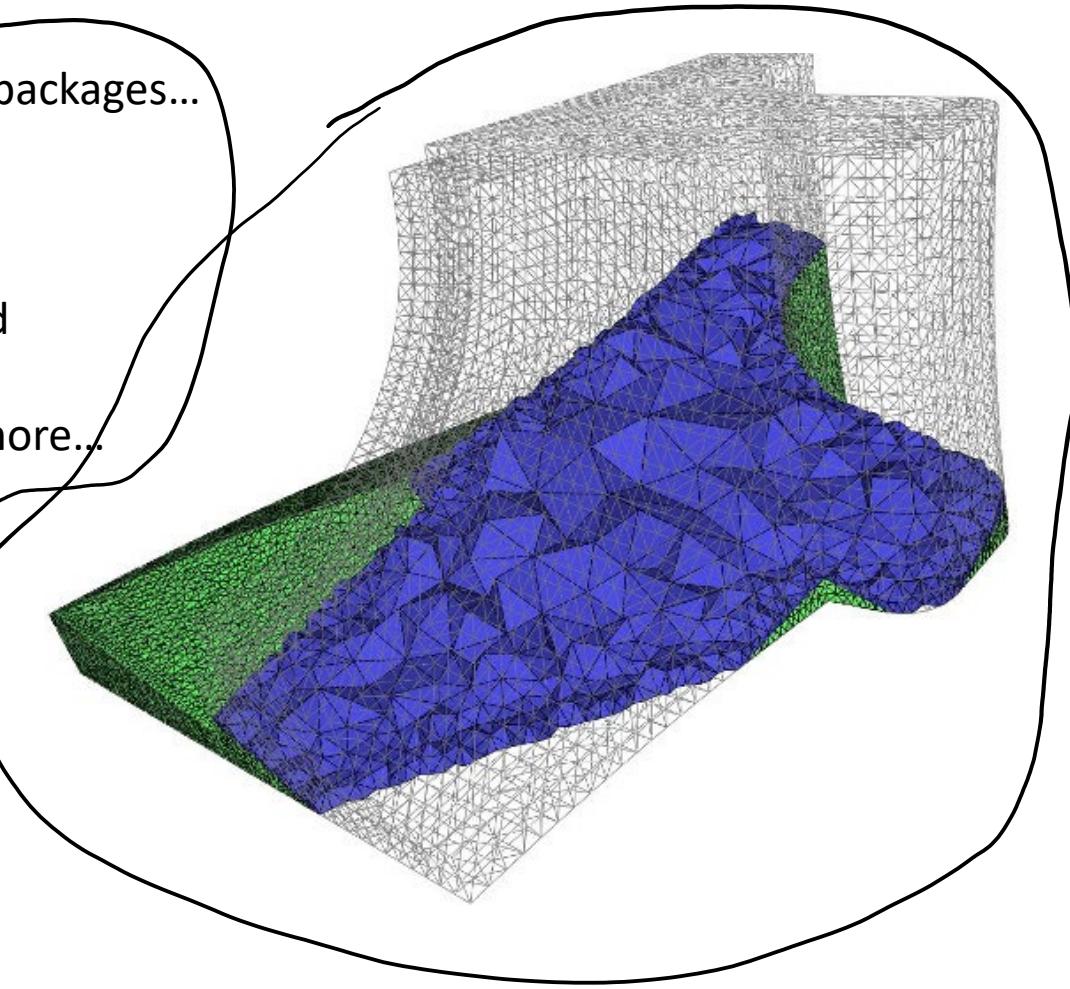
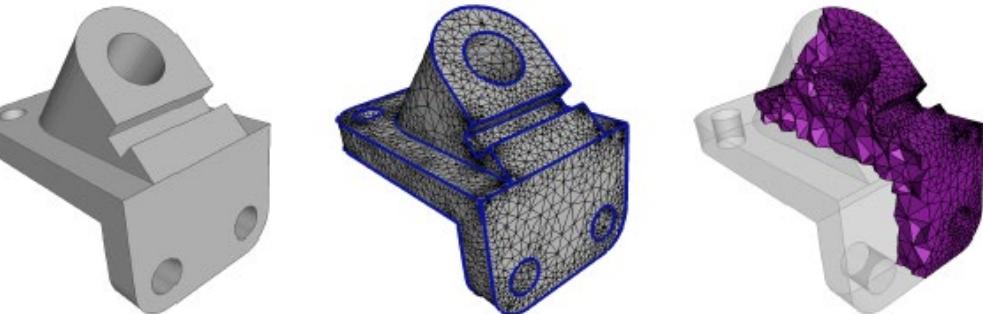
and $\lambda_4 = 1 - \lambda_1 - \lambda_2 - \lambda_3$

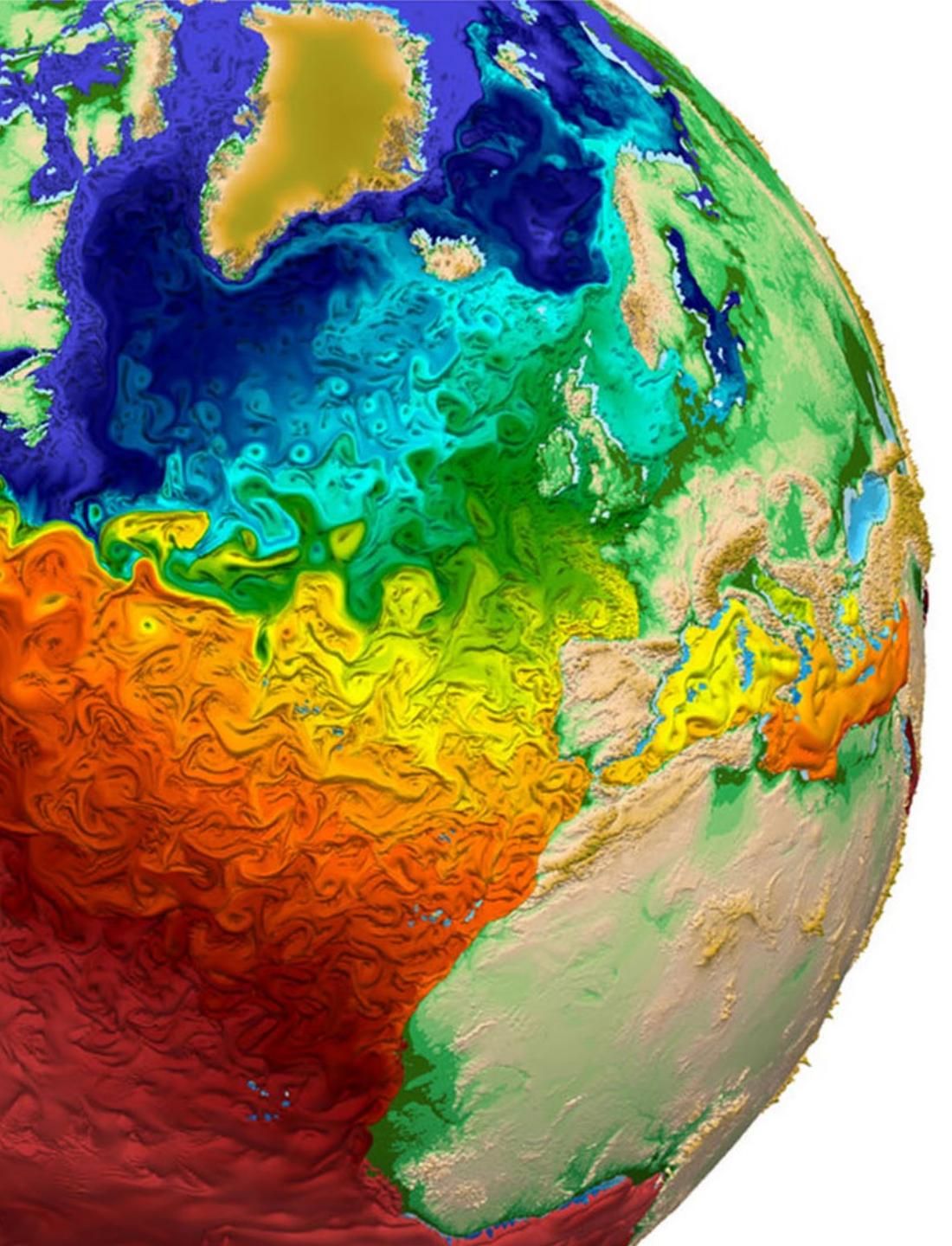


Triangle and Tetrahedral Mesh Generation



Lots of packages...
Ansys
Gridgen
Maya
Autocad
CGAL
Many more...





Data Science for People in a Hurry

Scattered Data Interpolation

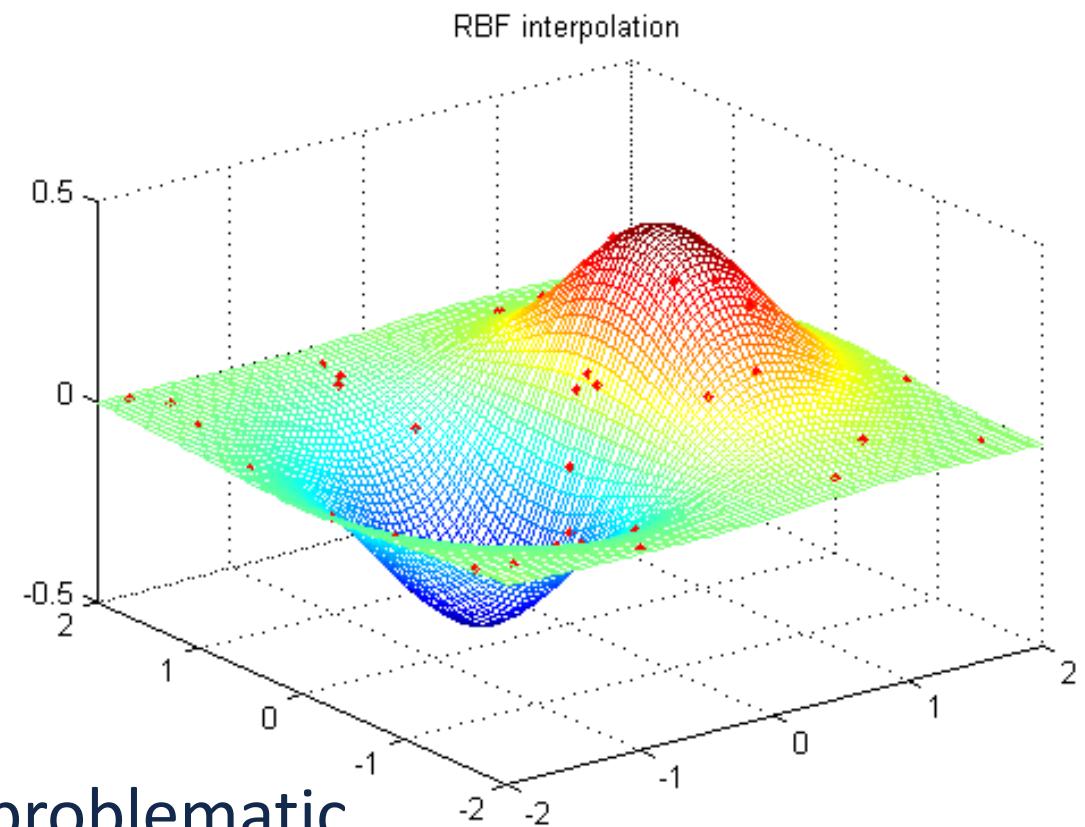
Scientific Visualization
Professor Eric Shaffer

Scattered Data

Scattered data is irregularly sampled

No spatial structure

Using bilinear or trilinear interpolation problematic



Shepard's Method

- Simplest scattered data interpolation method

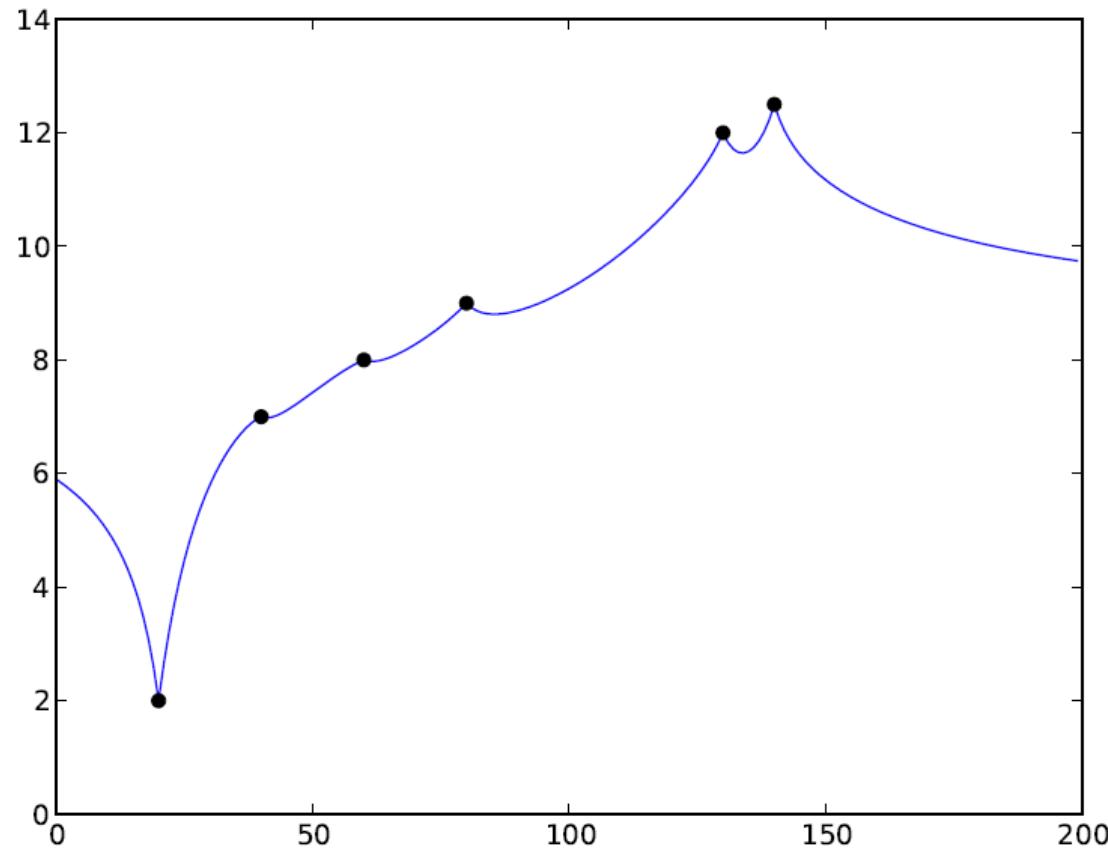
$$\tilde{f}(\mathbf{x}) = \sum_k^N \frac{w_k(\mathbf{x})}{\sum_j w_j(\mathbf{x})} f(\mathbf{x}_k)$$

$$w_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_i\|^{-p}$$

- \mathbf{x}_k are the locations in space with known function values
- \mathbf{x} is the query point
- w is a weight function inversely dependent on distance to \mathbf{x}
- p is a positive real number
 - larger p is, the greater influence points close to \mathbf{x} will have

Shepard's Method Issues

For $p \leq 1$ the interpolant has peaks...not ideal for smooth interpolation

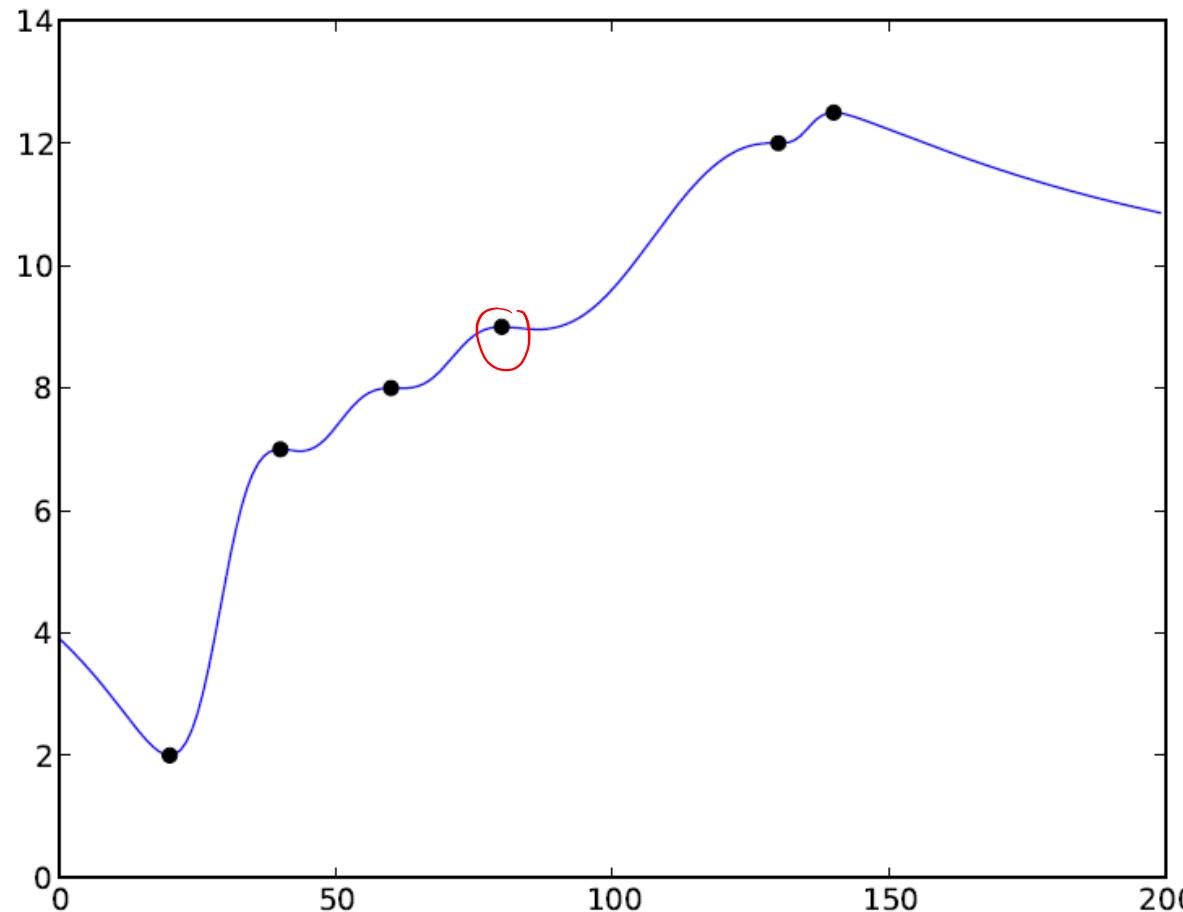


Example with $p=1$

Shepard's Method Issues

For $p > 1$ the interpolant is smooth...

But, first derivative is equal to 0 at data points...again not usually a desired behavior



Example with $p=2$

Modified Shepard's Method

One other issue is lack of scalability...ALL points in a data set used at each query x

Modified Shepard's Method uses only points within a radius of r around x

For those points, the weight function is

$$w_j(\mathbf{x}) = \left[\frac{r - d(\mathbf{x}, \mathbf{x}_i)}{rd(\mathbf{x}, \mathbf{x}_i)} \right]^2$$

Requires use of a spatial data structure such as kd-tree or quadtree/octree

Radial Basis Functions

- Any function dependent on distance from a center is *radial*
- We can compute an interpolating function as a weighted sum...

$$\phi(x, p) = \phi(\|x - p\|)$$

$$f(x) \approx \sum_{i=1}^N w_i \phi(x, p_i)$$

- Some popular kernel functions

$$\phi(r) = e^{-\lambda r^2}$$

Gaussian

$$r = \|x - p\|$$

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

Inverse distance

RBFs Computing Weights

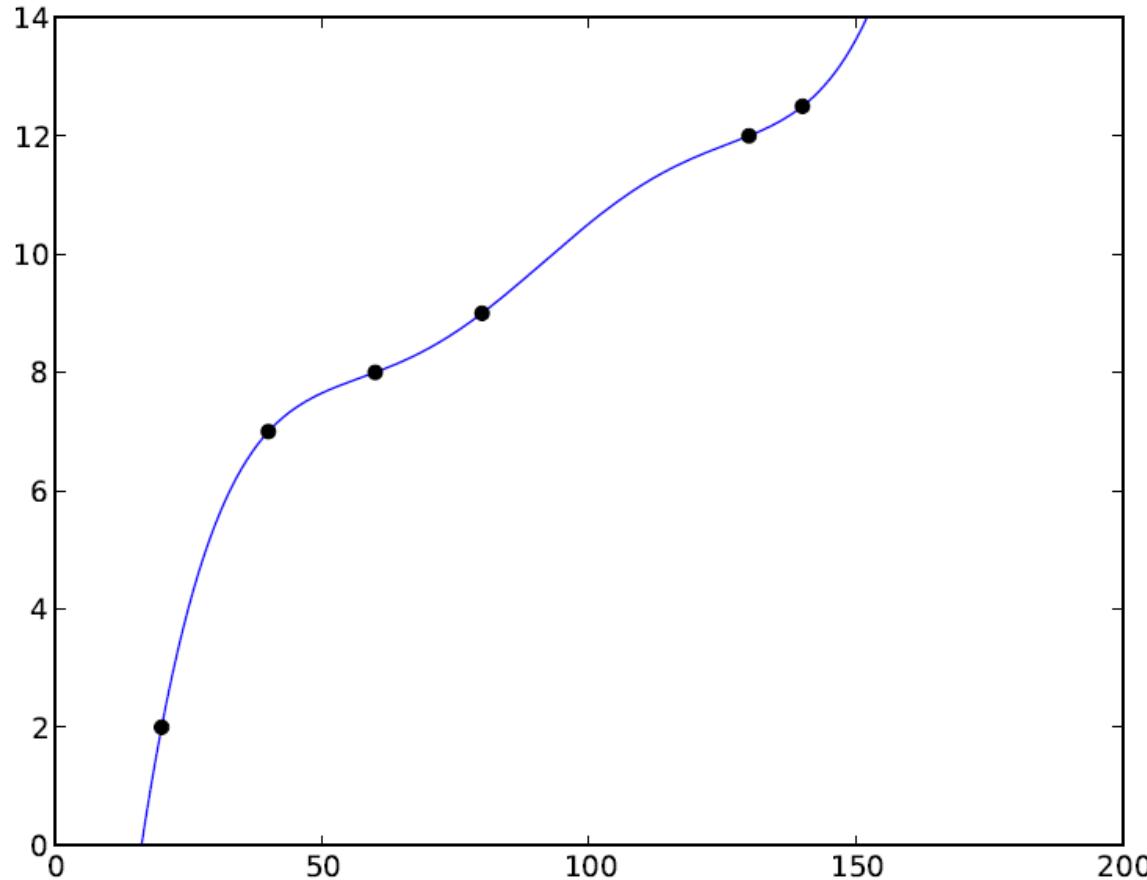
- Need to compute weights
- Constraint is function interpolates data points
- For scalability use a kernel function with width
 - Can lead to non-smooth interpolant

$$f(p_j) = \sum_{i=1}^N w_i \phi(p_j, p_i)$$
$$\underbrace{Aw}_{\text{constraint}} = p$$
$$A = \begin{bmatrix} \phi(p_1, p_1) & \dots & \phi(p_1, p_N) \\ \dots & \dots & \dots \\ \phi(p_N, p_1) & \dots & \phi(p_N, p_N) \end{bmatrix}$$
$$w = \begin{bmatrix} w_1 \\ \dots \\ w_N \end{bmatrix}$$
$$p = \begin{bmatrix} f(p_1) \\ \dots \\ f(p_N) \end{bmatrix}$$

A red circle highlights the first row of matrix A . A red oval highlights the second column of matrix A . A red oval highlights the first element of vector w . A red arrow points from the highlighted element in w to the corresponding element in vector p .



RBF Interpolation Example



Example radial basis interpolation with a Gaussian kernel

Kernel Function Choice

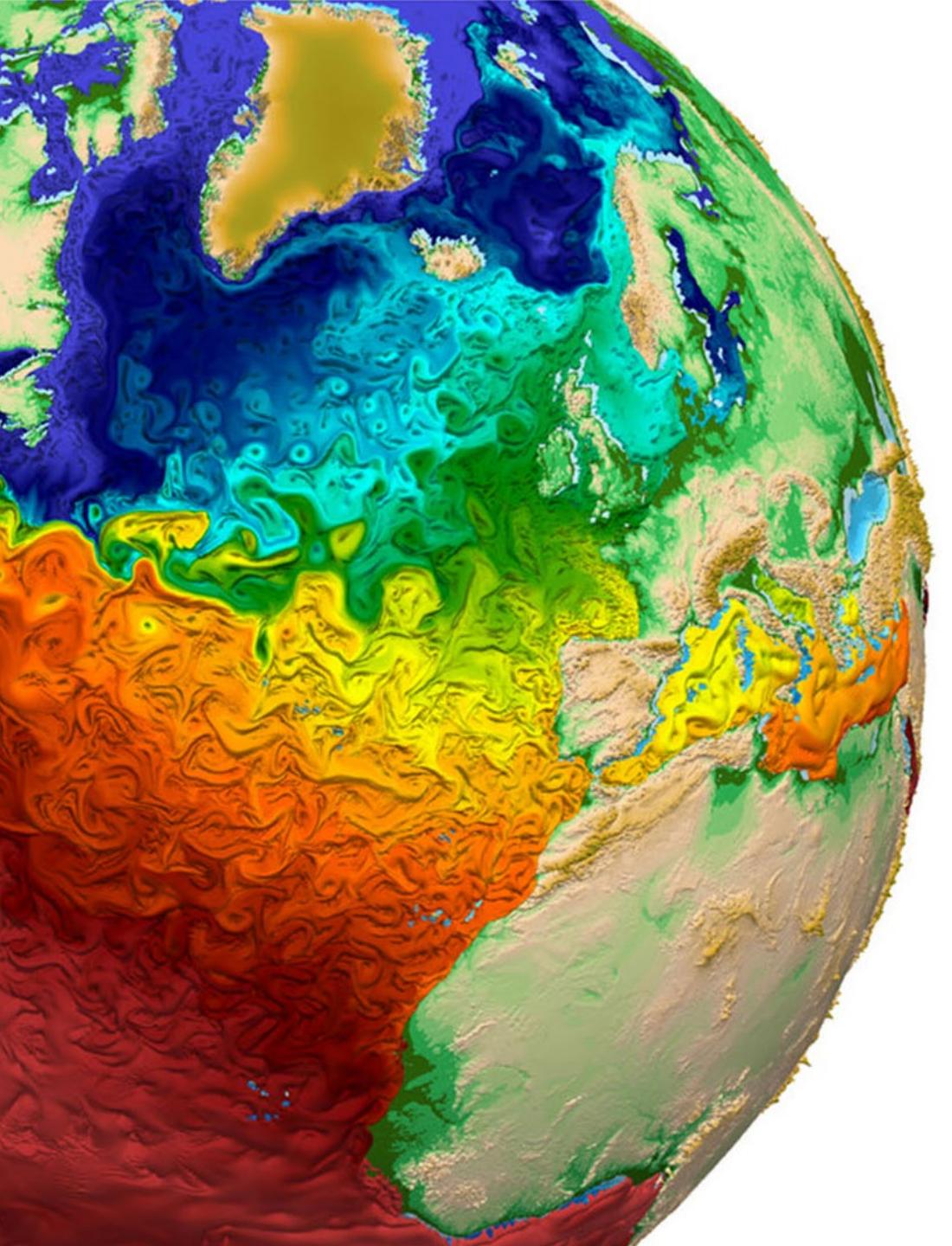
Positive definite functions will result in non-singular A for any data points

One definition of positive definite function: matrix A has all positive eigenvalues

Some useful kernels are not positive definite

Other Interpolation Method Options

- Moving Least Squares
- Natural Neighbor Interpolation
- ...many more



Contouring

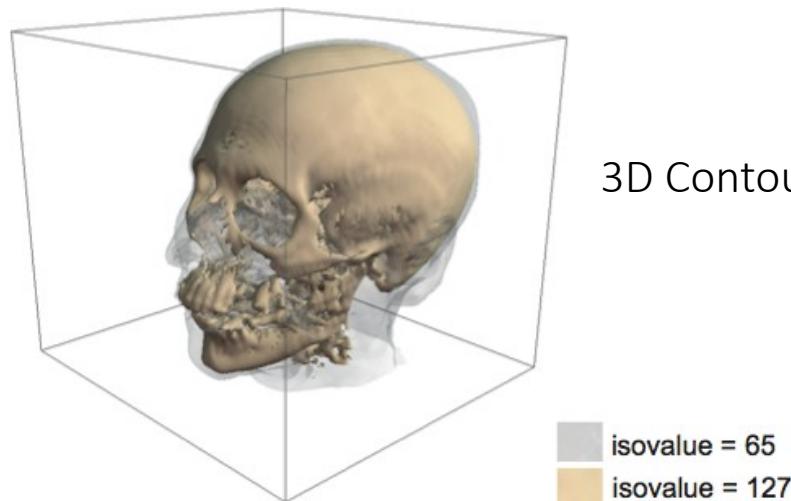
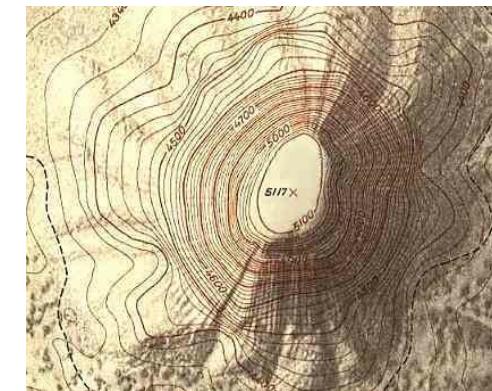
Marching Squares

Scientific Visualization
Professor Eric Shaffer

Contouring

Contours have been used for hundreds of years in cartography

- also called *isolines* ('lines of equal value')



3D Contouring: Marching Cubes:

"Marching cubes: A high resolution 3D surface construction algorithm",
by Lorensen and Cline (1987)

16,000 citations on Google Scholar

Contour Properties

Definition $I(f_0) = \{x \in D \mid f(x) = f_0\}$

Contours are always closed curves (except when they exit D)

- why? Recall that f is C^0

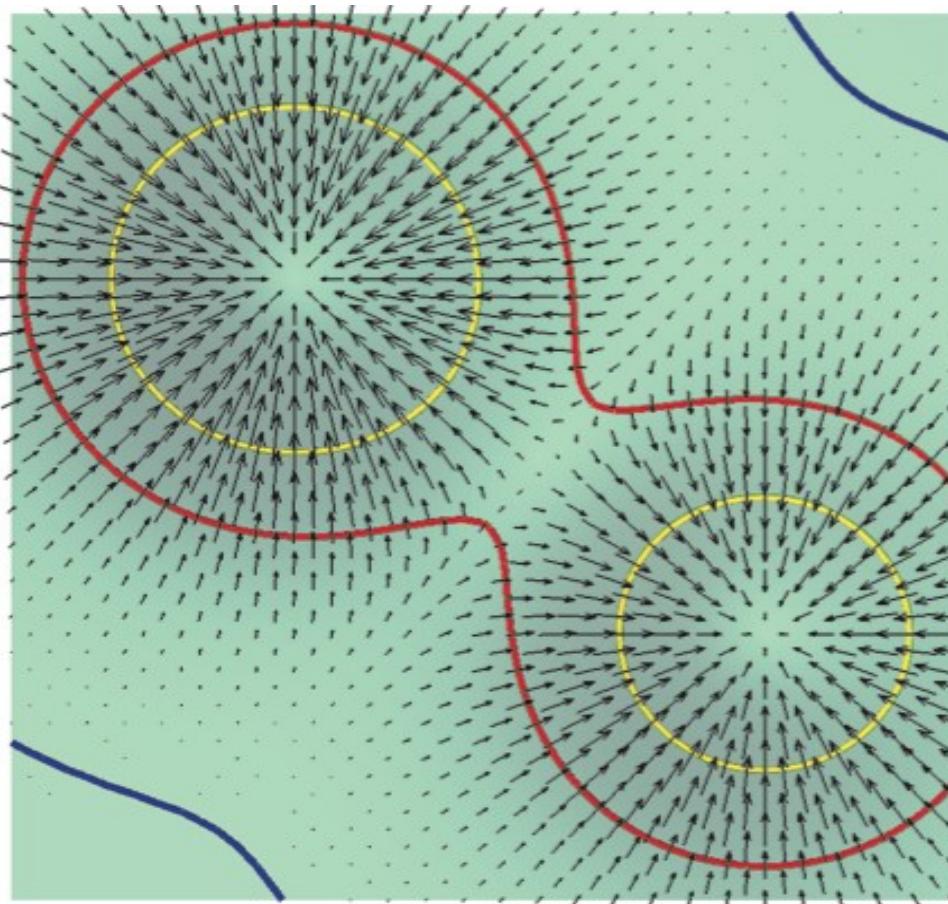
Two different contour lines never intersect, thus are nested

- why? What would it mean if a point belonged to two different contours

Contour Properties

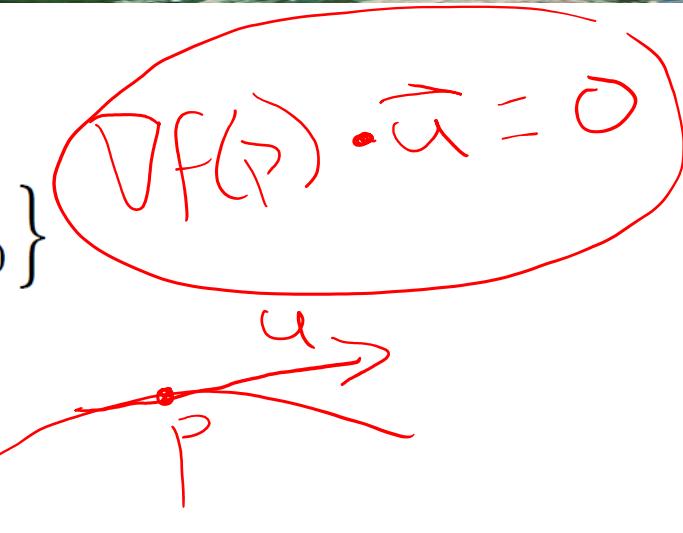
Contours are always orthogonal to the scalar value's gradient

- why?



$$I(f_0) = \{x \in D \mid f(x) = f_0\}$$

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$



contour: $\frac{\partial f}{\partial I} = 0$ since f constant along I

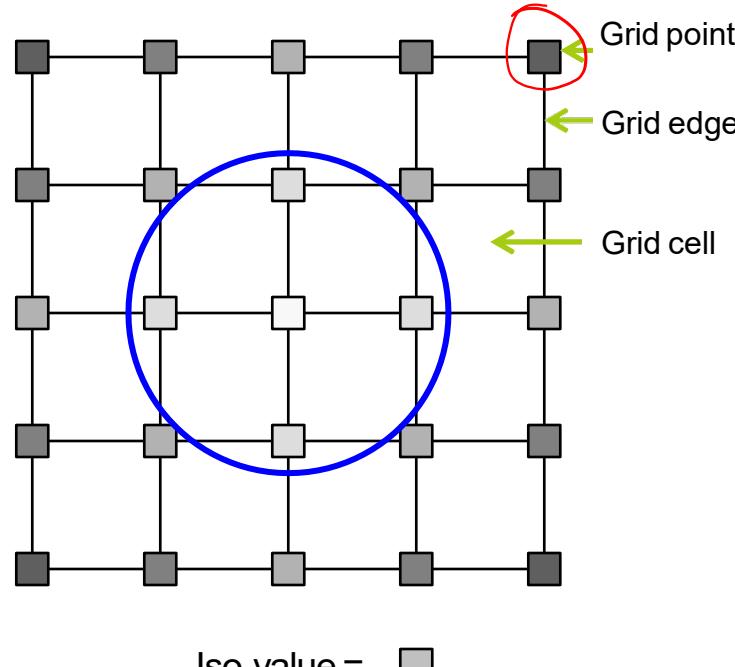
gradient: $\frac{\partial f}{\partial (\nabla f)} = \max$ by definition of gradient
direction of greatest increase in f

gradient of a scalar field (drawn with arrows) is orthogonal to contours

Contouring on a Grid of Sampled Data

Input

- A grid where each grid point has a value
- An iso-value (threshold)



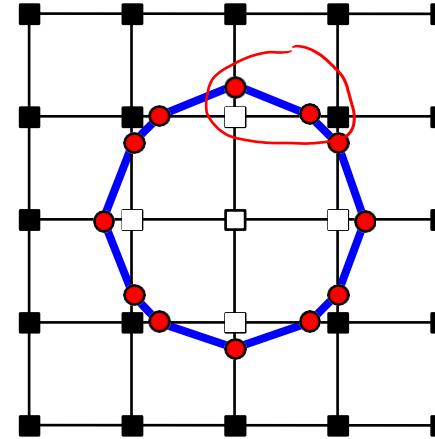
Output

- A closed polyline (2D) or mesh (3D) that separates grid points **above** or **below** the iso-value

Algorithms

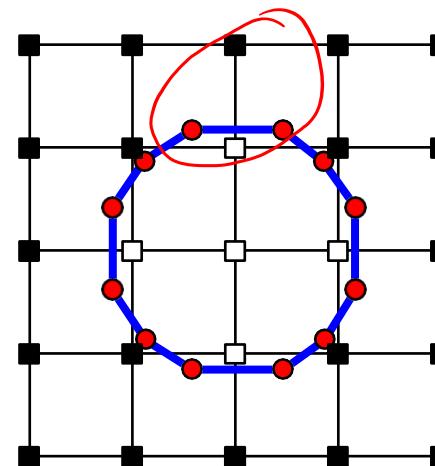
Primal methods

- Marching Squares (2D), Marching Cubes (3D)
- Placing vertices on grid edges

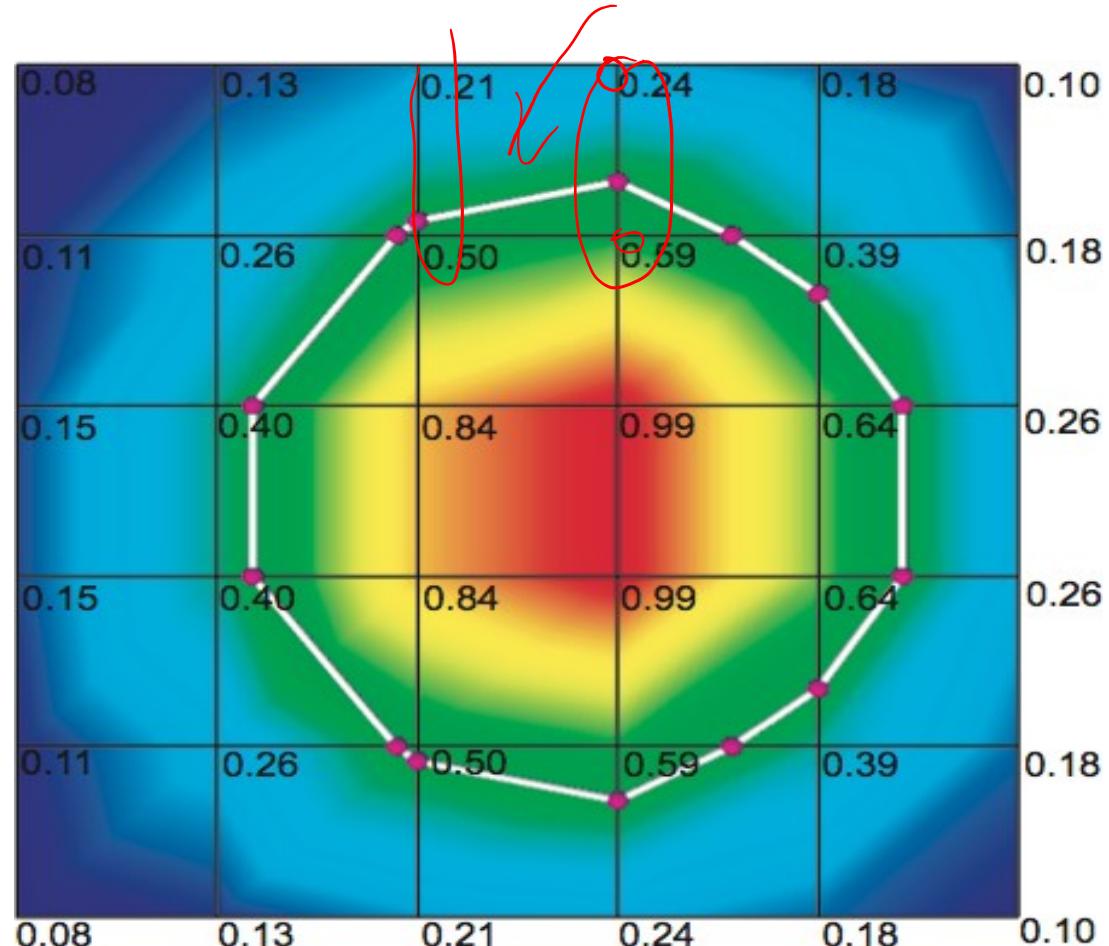


Dual methods

- Dual Contouring (2D,3D)
- Places vertices in grid cells



Contouring in 2D

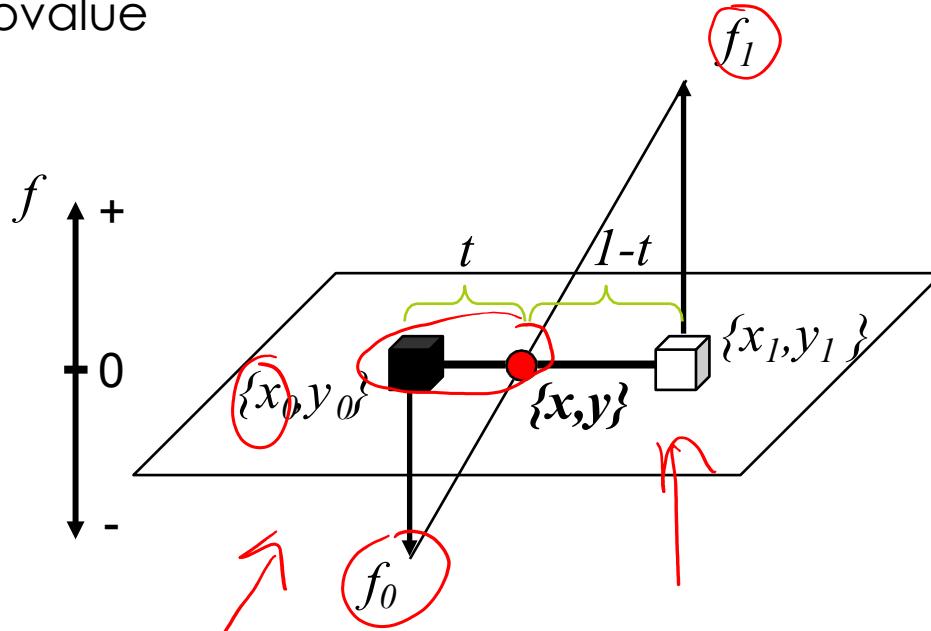


```
S = ∅  
for(each cell c in D)  
{  
    for(each edge e=(pi,pj) of c)  
    {  
        if(fi<ν<fj)  
        {  
            Compute the intersection point q  
            S = S ∪ q  
        }  
    }  
}  
connect points in S with lines to build contour;  
}
```

Marching Squares

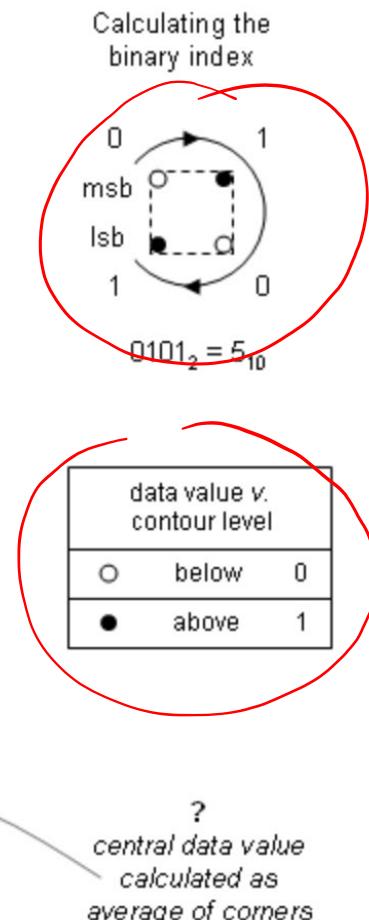
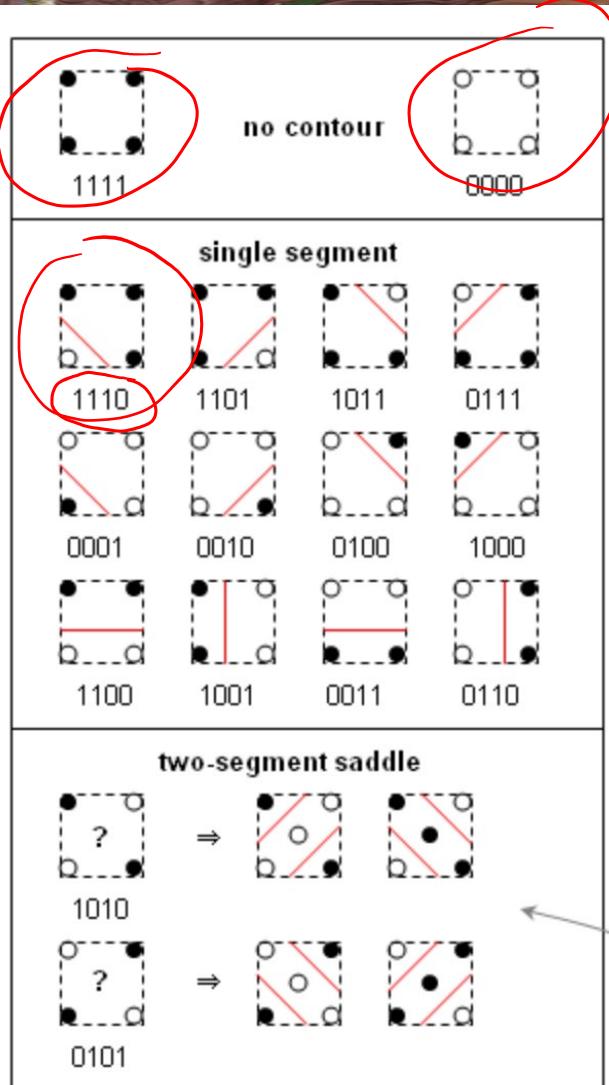
Creating contour line vertices (x, y)

- Assume the underlying, continuous function is linear on the grid edge
- Linearly interpolate the positions of the two grid points
- v is the isovalue



$$t = \frac{v - f_0}{f_1 - f_0}$$
$$x = x_0 + t(x_1 - x_0)$$
$$y = y_0 + t(y_1 - y_0)$$

Marching Squares



2D contouring on quad-cell grids

1. Encode inside/outside state of each vertex in a 4-bit id
2. Process all dataset cells
 - for each cell,
use ids as pointers into a table with 16 cases
 - each case has associated code to
 - compute the edge-contour intersection positions
 - connect to already-computed contour vertices from previous cells

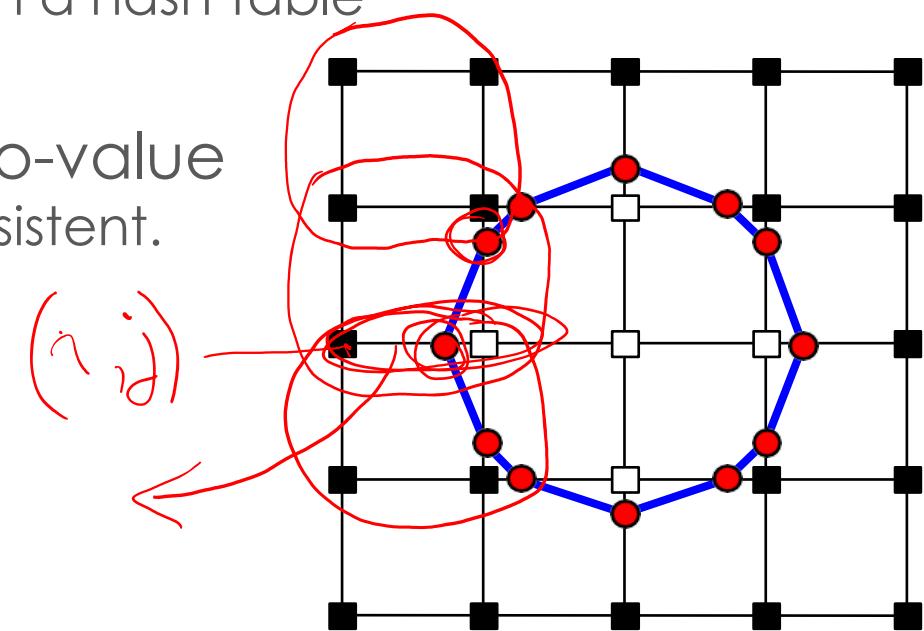
Marching Squares: Implementation

Avoid computing one vertex multiple times

- Compute the vertex location once, and store it in a hash table

When the grid point value is same as the iso-value

- Treat it either as “above” or “below”, but be consistent.

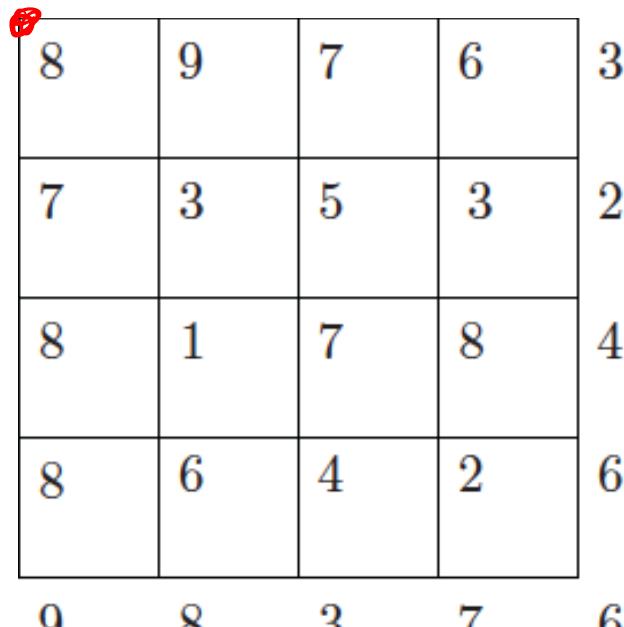


Marching Squares: Example

Use an isovalue of 5

Scalars associated with point to the upper left

Classify points with value 5 as positive

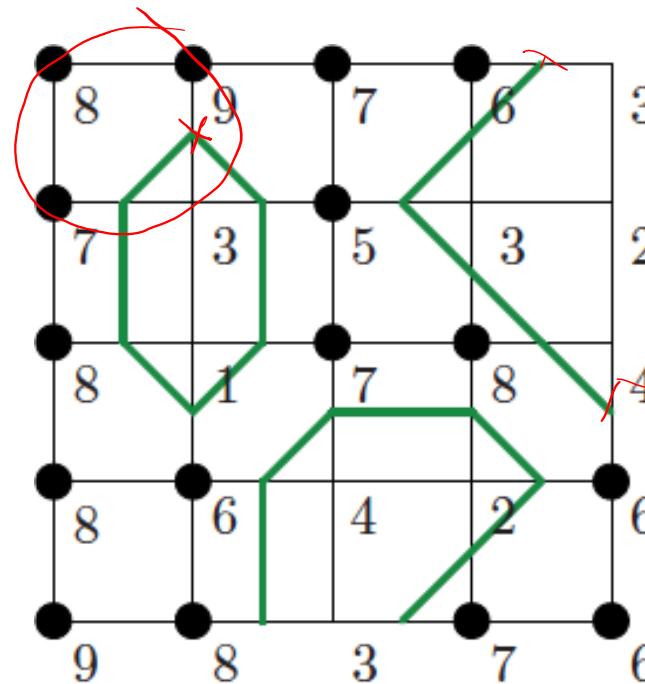


(a) Scalar grid.

Marching Squares: Example Using Midpoints

Scalars associated with point to the upper left

Classify points with value 5 as positive

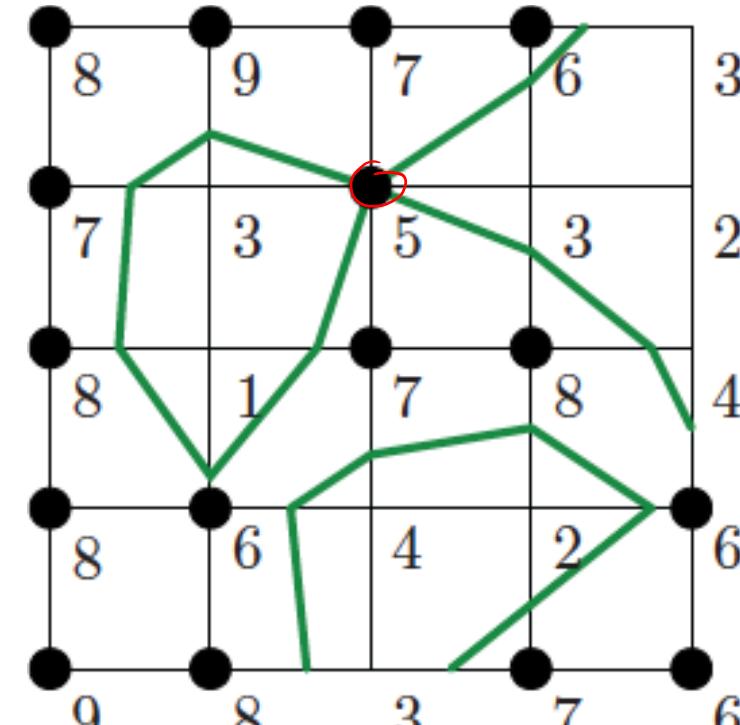


(c) Midpoint vertices.

Marching Squares: Example Using Interpolation

Scalars associated with point to the upper left

Classify points with value 5 as positive

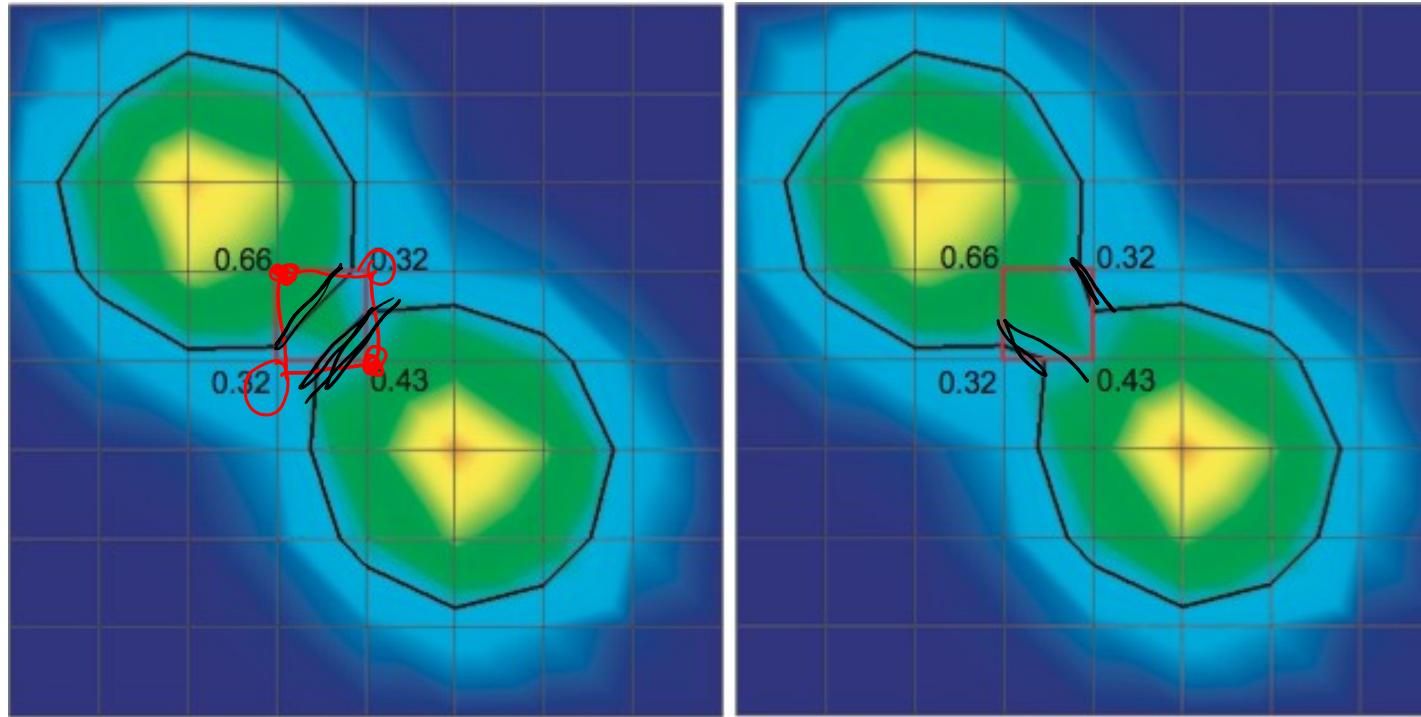


(d) Isocontour.

Contouring: Ambiguity

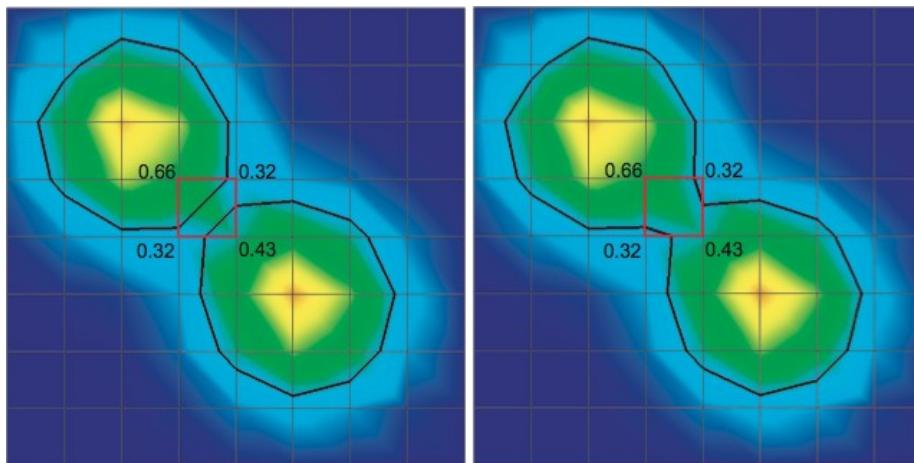
Each edge of the red cell intersects the contour

- which is the right contour result?



Contouring: Ambiguity

Each edge of the red cell intersects the contour
• which is the right contour result?



Both answers are equally correct!

- we could discriminate only if we had higher-level information (e.g. topology)
- at cell level, we cannot determine more unless we increase sampling rate

Contouring: Ambiguity

Some cell corner value configurations yield more than one consistent polygon

- In 3-D can yield holes in surface!

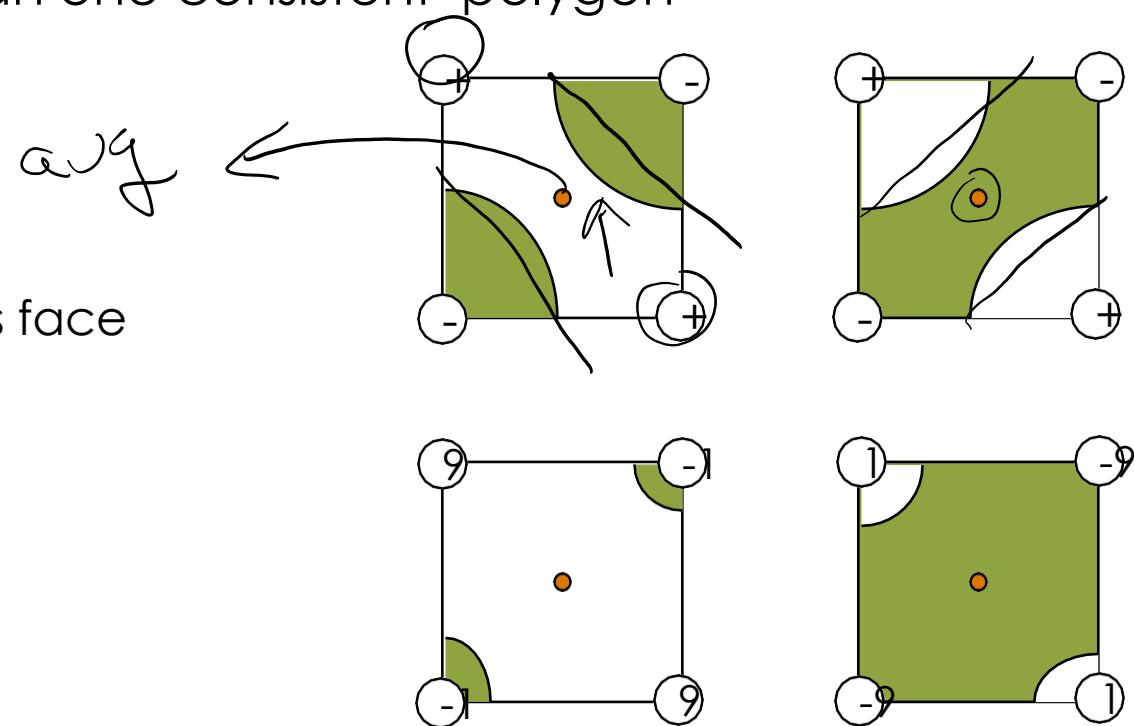
How can we resolve these ambiguities?

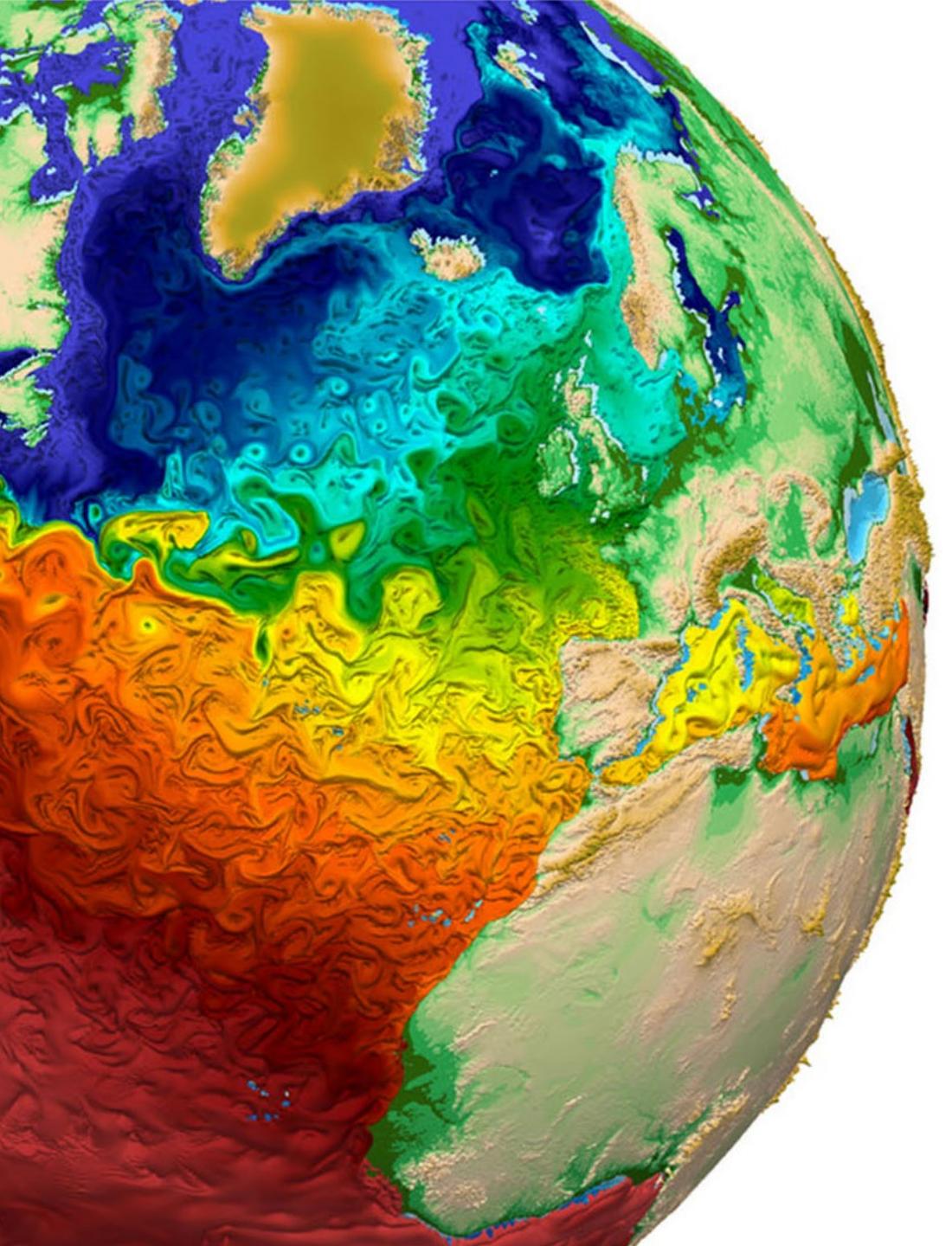
- Topological Inference
 - Sample a point in the center of the ambiguous face

If data is discretely sampled,
bilinearly interpolate to sample

$$p(s,t) = (1-s)(1-t) a + (1-t) b + (1-s) t c + s t d$$

*a,b,c, and d are the function values at the 4 corners
s and t are parametric location inside the grid cell
...for midpoint s = $\frac{1}{2}$ t= $\frac{1}{2}$*





Domain Modeling

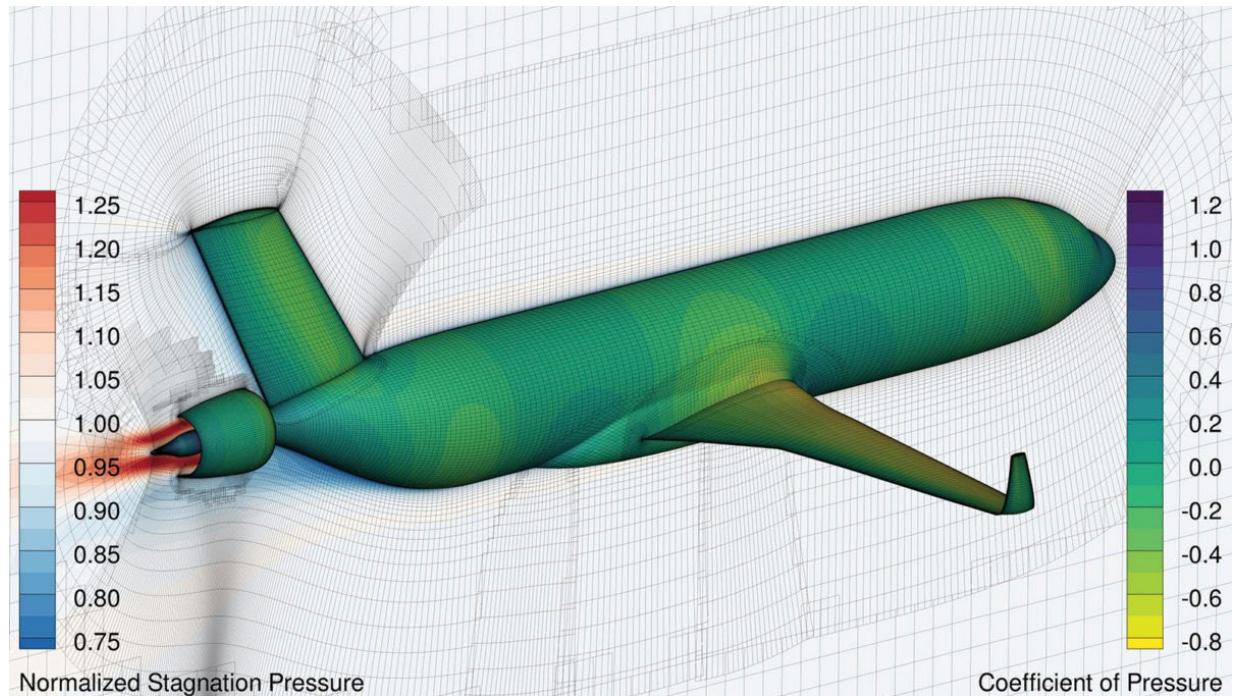
Meshes and Elements

Scientific Visualization
Professor Eric Shaffer

Domain Discretization

Scientific data often presented in a discretized domain

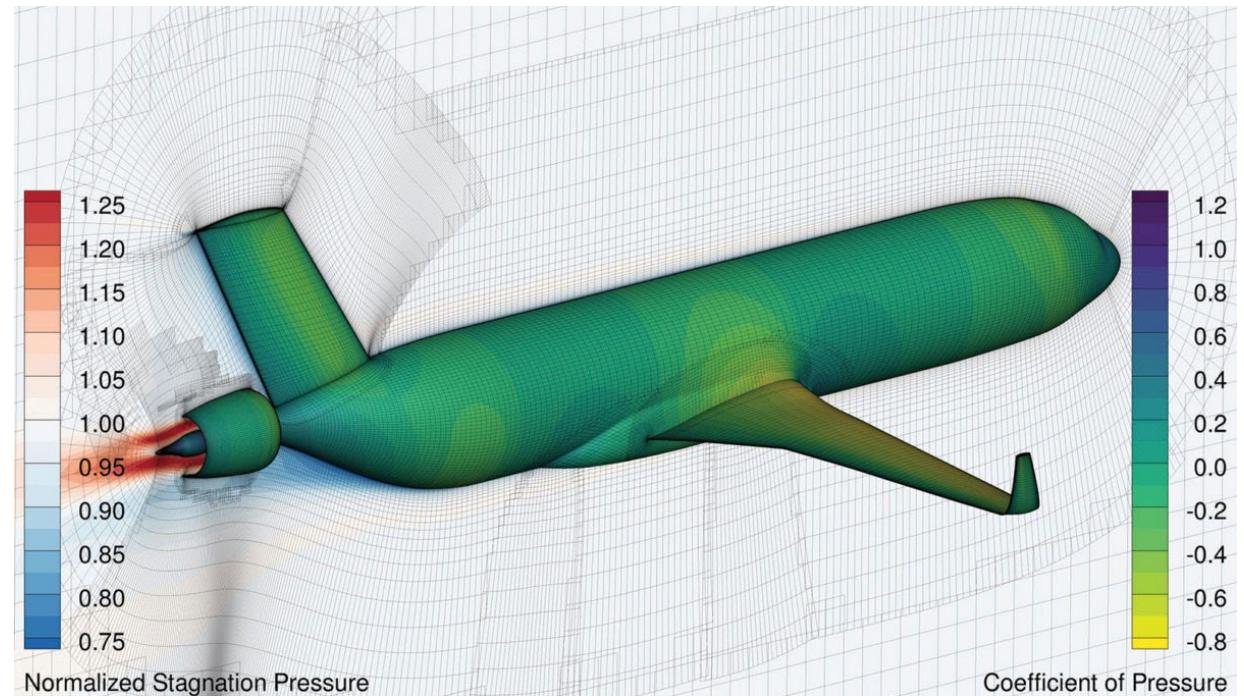
- Partitioned into discrete cells
- Each cell associated with some data
- Simulations often use domain discretization
- Data is easier to analyze, both visually and numerically



Domain Discretization

Many choices for how one can discretize

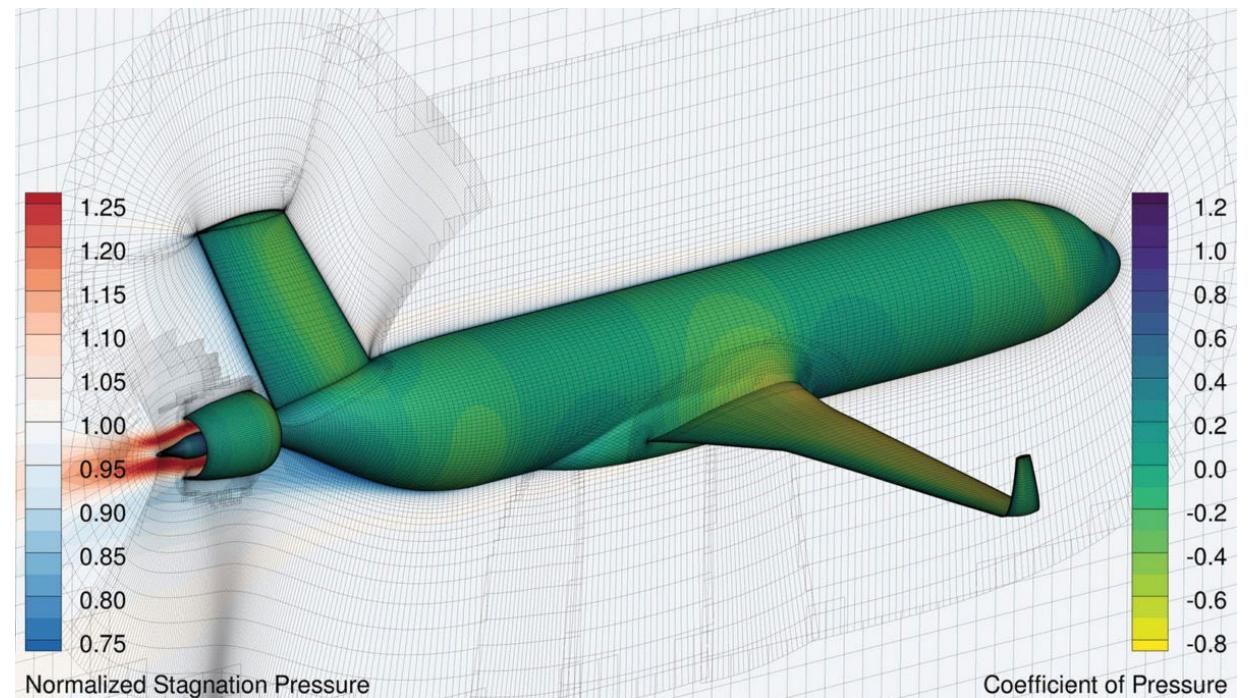
- Impacts space required for data storage
- Efficiency of operations on the data



Domain Discretization

Discretization May Need to Change for Rendering

- Rendering may require processing the data mesh
- e.g. Given a hexahedral mesh may need to
 - Identify surface faces of the hexahedra
 - Triangulate them
 - Render the triangles



Terminology

Geometry

Positions of the vertices in space

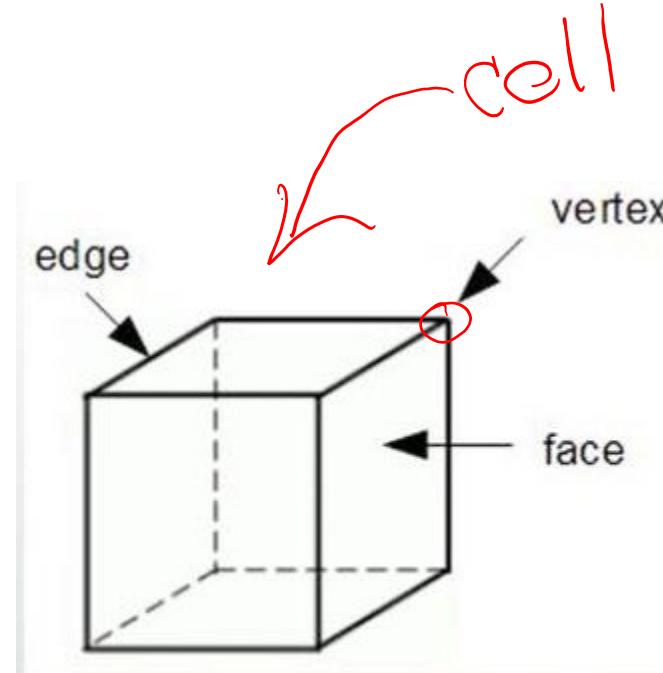
Can be structured or unstructured

Topology

Cells

Connectivity information

Can be structured or unstructured



Equivalent Terminology

- Grid = Mesh
- Nodes = Vertices
- Elements = Cells
- ...lots of discipline specific terminology
 - e.g. in computational fluid dynamics (CFD) a zone is a group of cells
 - e.g. in geographic information systems (GIS) a triangulated irregular network (TIN) is a surface mesh of triangles

Cells and Grids

Cells

- provide interpolation over a small, simple-shaped spatial region

Grids (or meshes)

- partition our complex data domain D into cells
- allow applying per-cell interpolation (as described so far)

Data

- per vertex
- per cell

Given a domain D ...

A grid $G = \{c_i\}$ is a set of cells such that

$$c_i \cap c_j = \emptyset, \forall i \neq j \quad \text{no two cells overlap}$$

$$\bigcup_i c_i = D \quad \text{the cells cover all our domain}$$

The dimension of the domain D constrains which cell types we can use



Cell Types

0D

- point

1D

- line

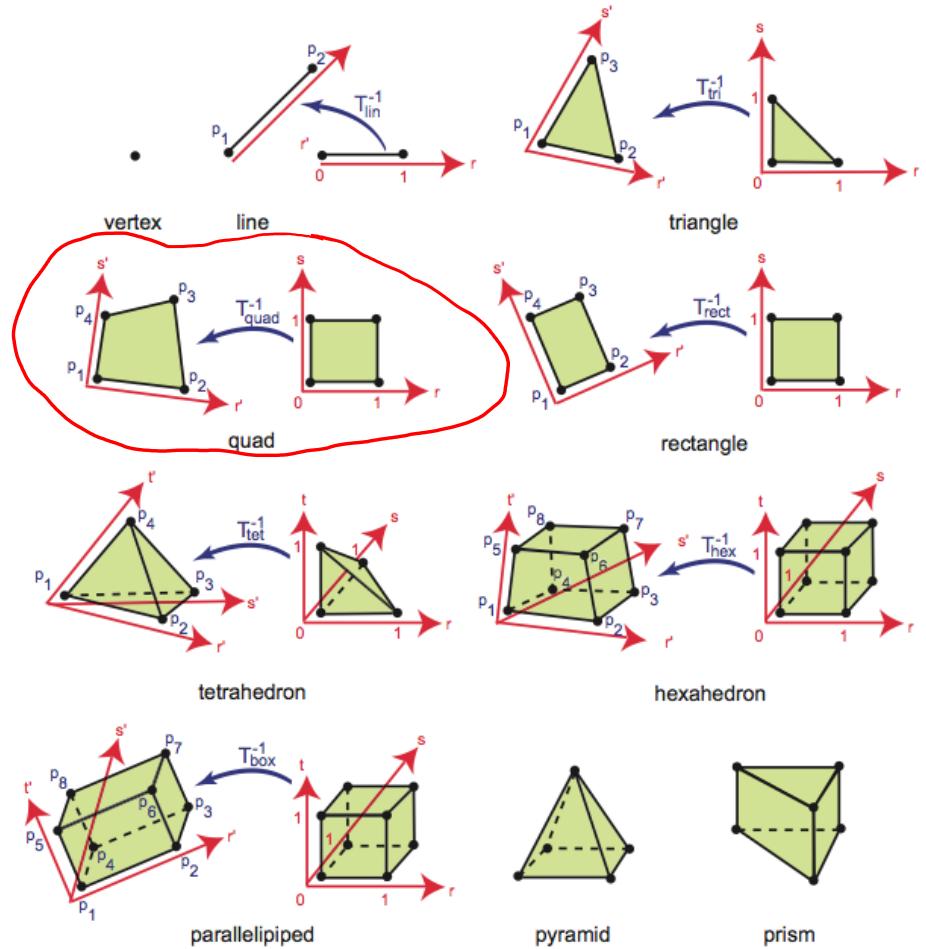
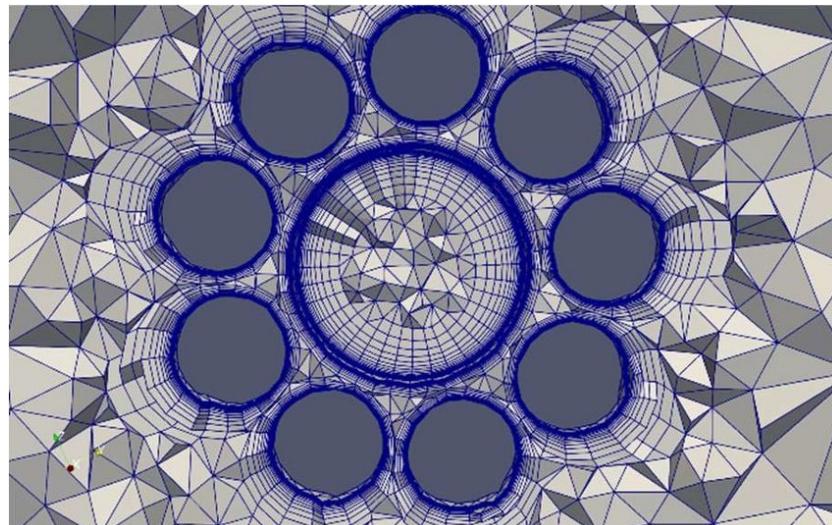
2D

- triangle, quad, rectangle

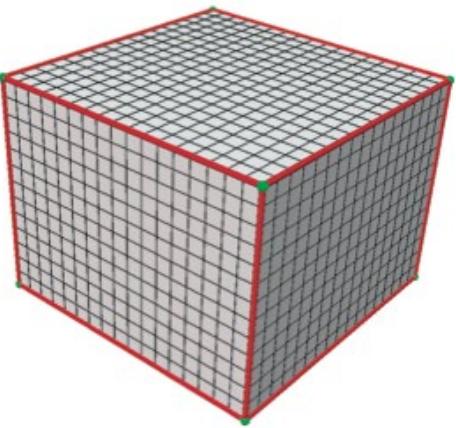
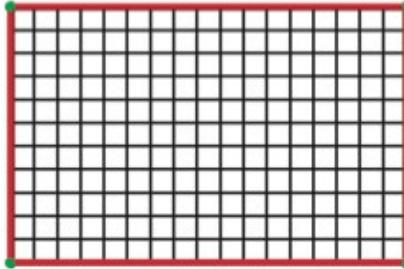
3D

- tetrahedron, parallelepiped, box, pyramid, prism, ...

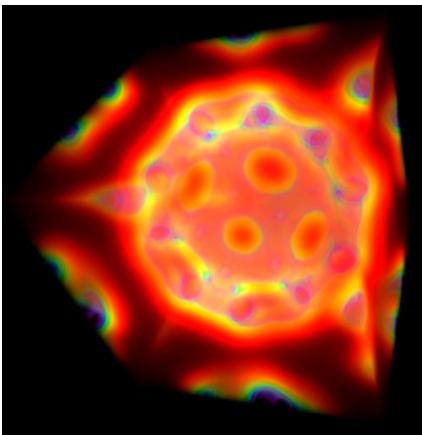
Hybrid Meshes contain more than 1 type of cell



Uniform Grids (Images)



image

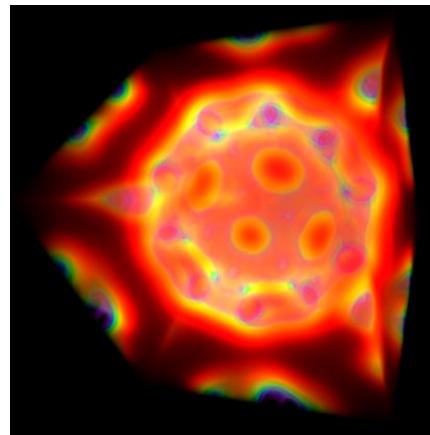
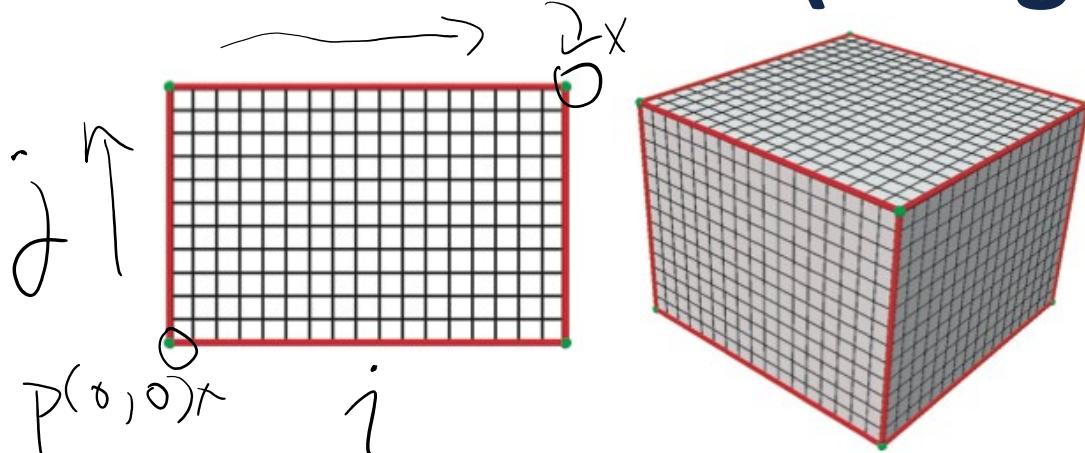


volume

- all cells have identical size and type (typically, square or cubic)
- cannot model non-axis-aligned domains



Uniform Grids (Images)



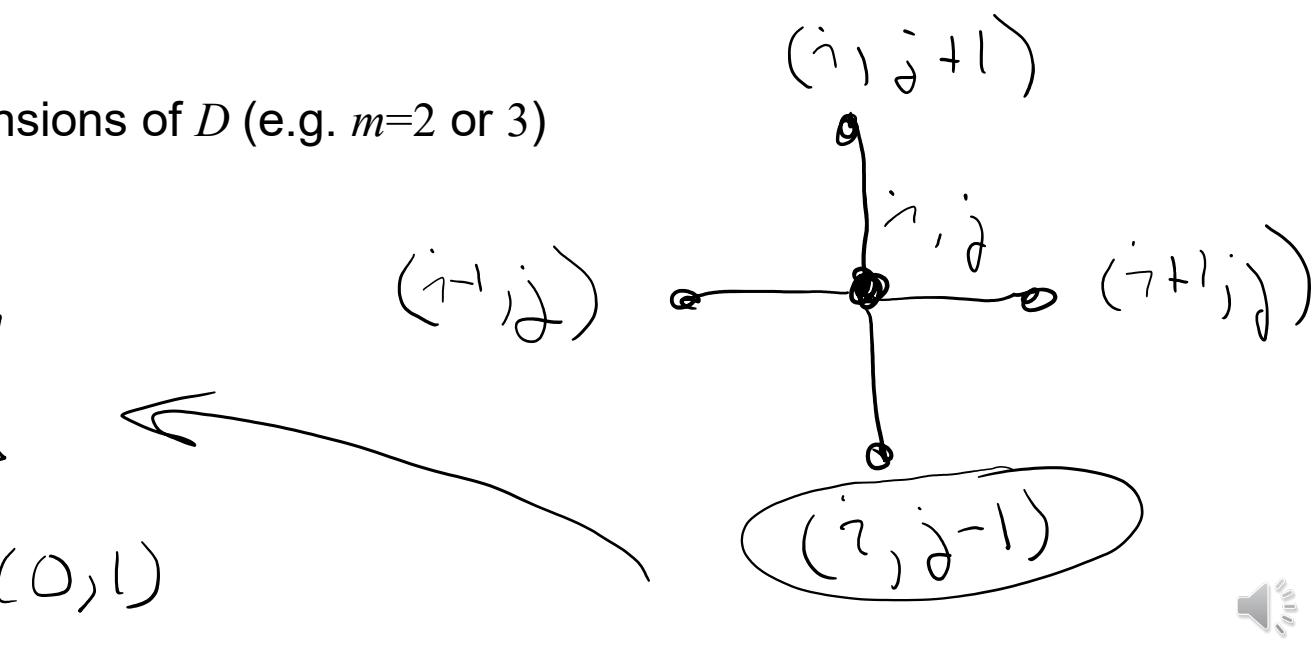
- all cells have identical size and type (typically, square or cubic)
- cannot model non-axis-aligned domains

Storage requirements for the structure

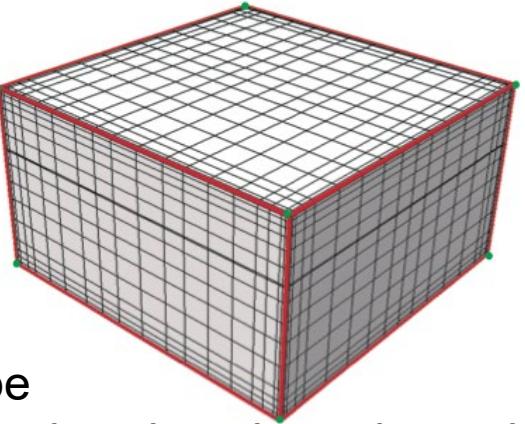
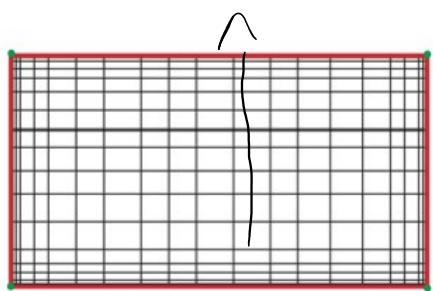
- m integers for the #vertices along each of the m dimensions of D (e.g. $m=2$ or 3)
- two corner points

Node positions can be computed instead of stored

$$P_{i,j} = P_{0,0} + i\Delta_x \vec{e}_x + j\Delta_y \vec{e}_y$$



Rectilinear Grids

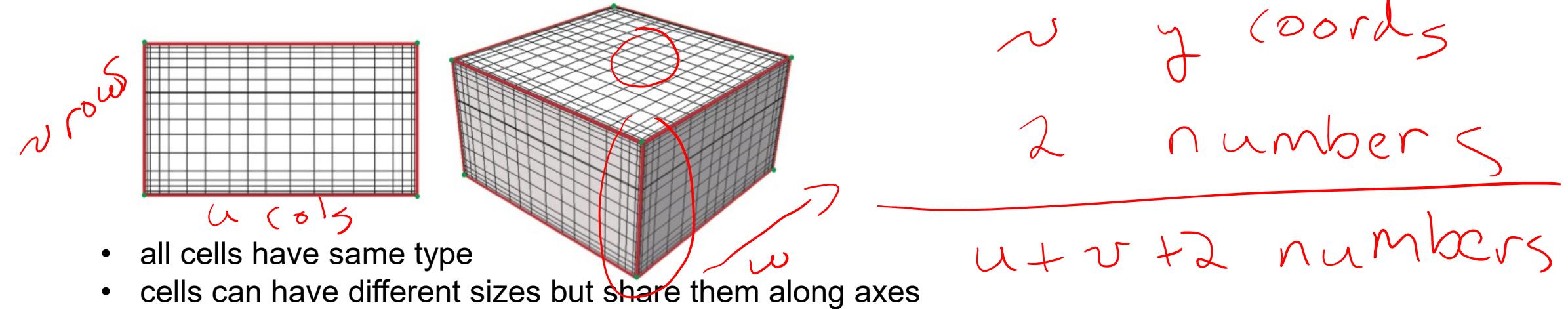


- all cells have same type
- cells can have different sizes but share them along axes

$$[\Delta x_0, \Delta x_1, \dots, \Delta x_n]$$



Rectilinear Grids



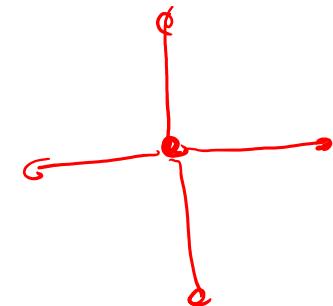
Storage requirements

$$\sum_{i=1}^m d_i \text{ floats (coordinates of vertices along each of the } m \text{ axes of } D)$$

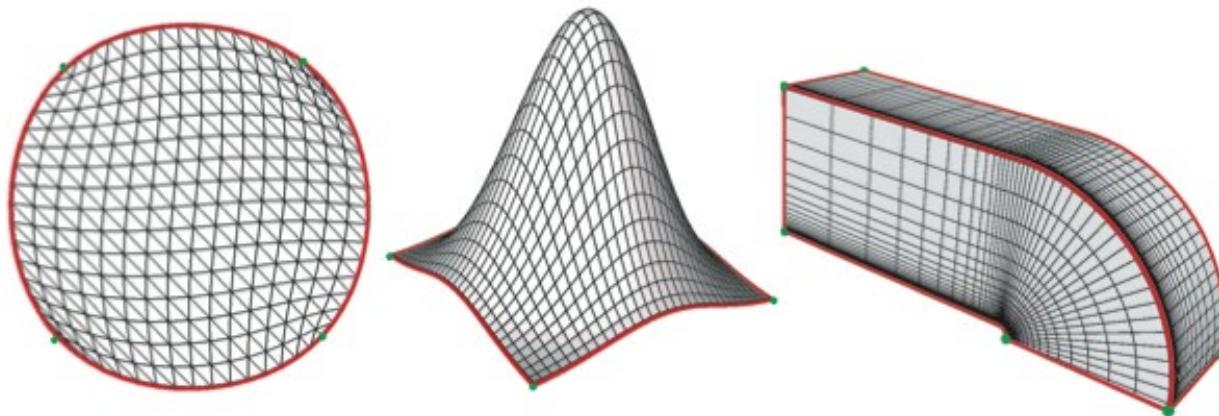
The number of vertices along each axis

Can compute node positions from stored coordinate information

$$P(i, j) = (P_x[i], P_y[j])$$



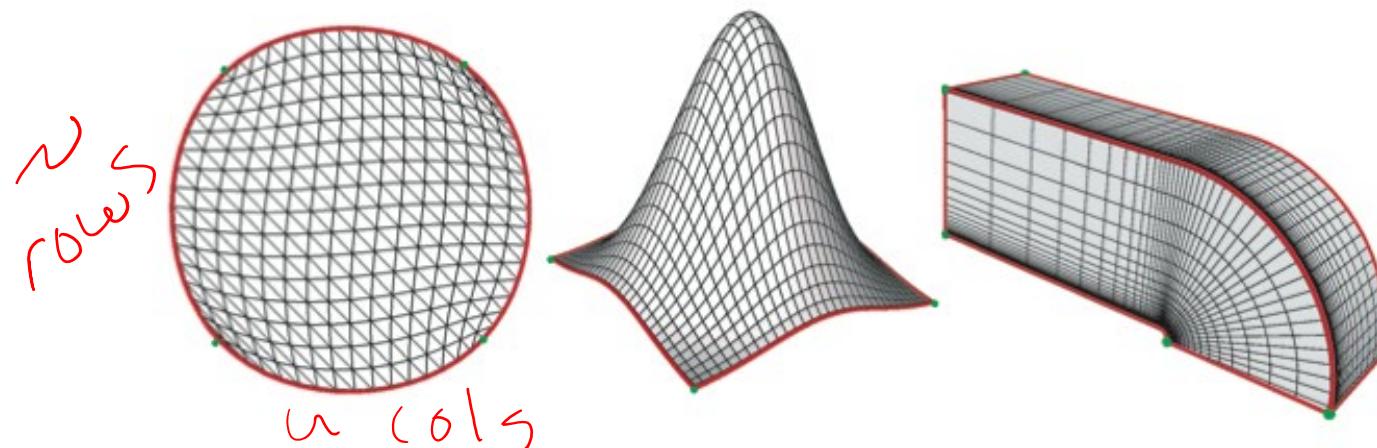
Curvilinear Grids



- all cells have same type
- cell vertex coordinates are freely (explicitly) specifiable...
- ...as long as cells assemble in a matrix-like structure
- can approximate more complex shapes than rectilinear/uniform grids



Curvilinear Grids



- all cells have same type
- cell vertex coordinates are freely (explicitly) specifiable...
- ...as long as cells assemble in a matrix-like structure
- can approximate more complex shapes than rectilinear/uniform grids

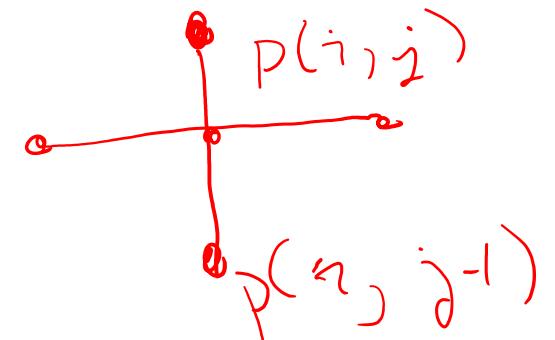
Storage requirements

$$\prod_{i=1}^m d_i \text{ floats (coordinates of all vertices)}$$

Also 1 number for each axis (the number of vertices along each axis)

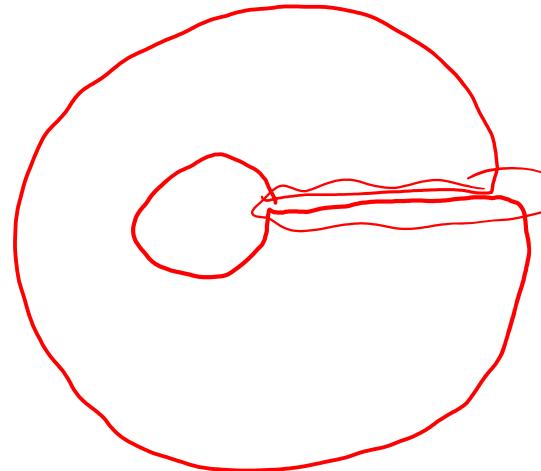
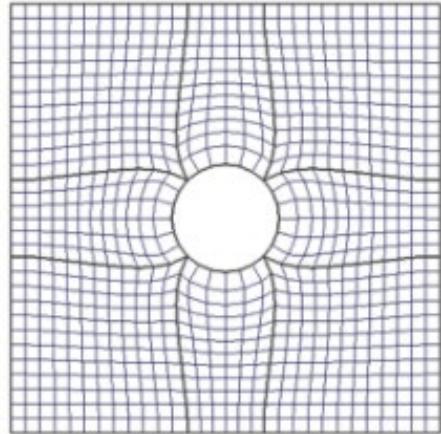
Two numbers

$$p[i][j][k] = (x_j, y_j)$$



Unstructured Grids

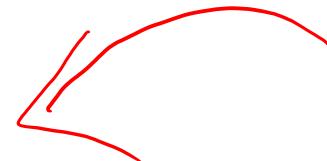
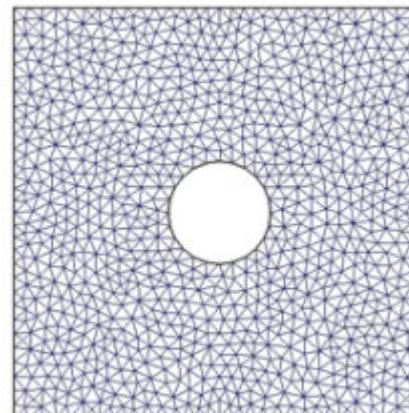
Consider the domain D : a square with a hole in the middle



We cannot cover such a domain with a single structured grid (why?)

- it's not of genus 0, so cannot be covered with a matrix-like distribution of cells

For this, we need unstructured grids



Unstructured Grids

- most flexible grid type for modeling complex geometry
- both vertex coordinates and cell themselves are freely (explicitly) specifiable
- one storage implementation
 - vertex set
 - cell set

Storage requirements

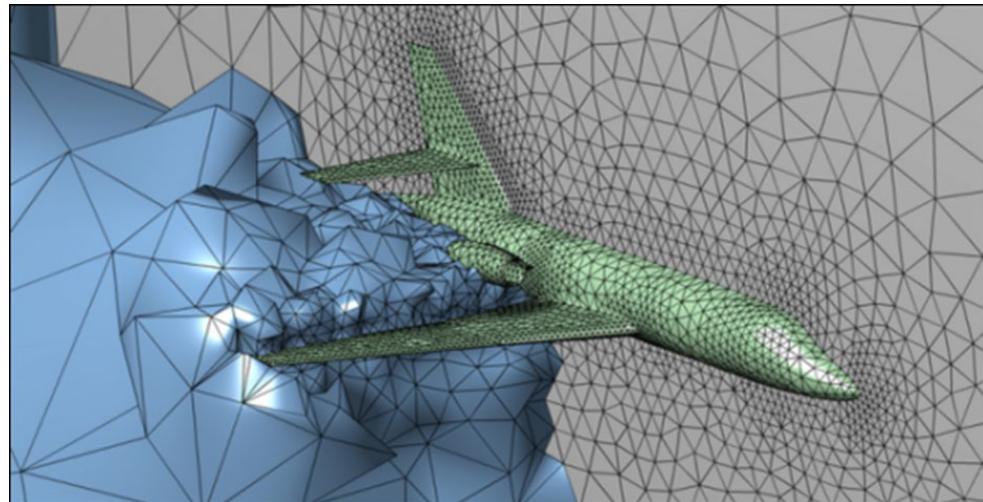
$$V = \{v_i\}$$

$$C = \{c_i = (\text{indices of vertices in } V)\}$$

$$m\|V\| + s\|C\|$$

for a m -dimensional grid with cells having s vertices each

What operation is hard to do with just this information?



$$c_1 = 1, 10, 15$$

Point in cell
cell neighbors

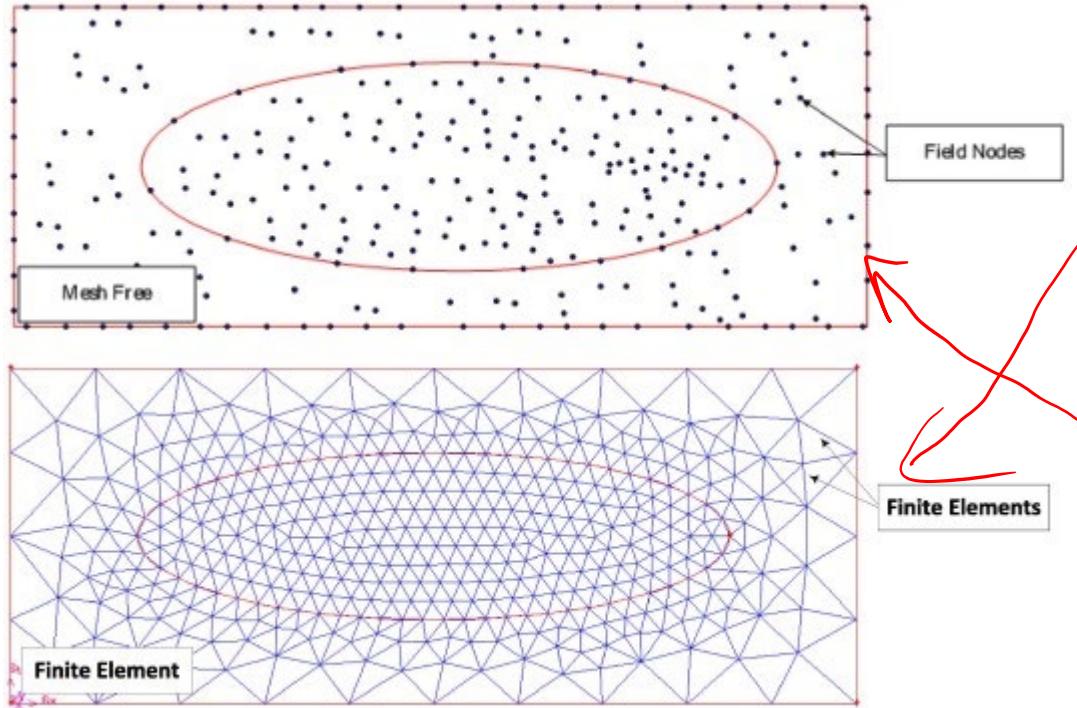


Grid Type Summary

Grid Type	Topology	Geometry
Uniform	Structured	Structured
Rectilinear	Structured	Semi-structured
Curvilinear	Structured	Unstructured
Unstructured	Unstructured	Unstructured

- Geometry can be structured or unstructured
- Topology can be structured or unstructured

Simulation Data: Mesh-Free Methods



Finite Element Method

mesh-based method for solving partial differential equations

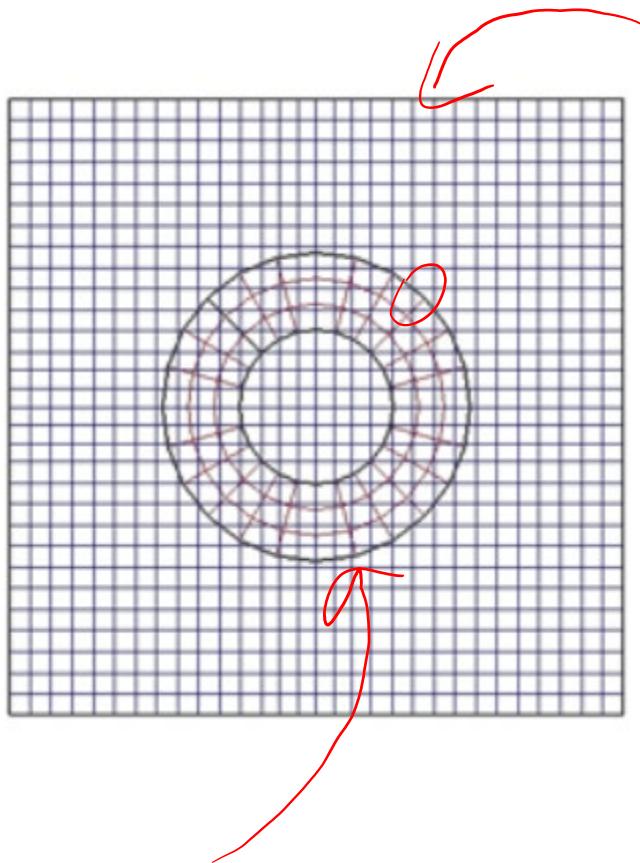
Mesh-free methods exist as well

Uses nodes and neighbor computations

Rendering solution may require meshing and interpolation

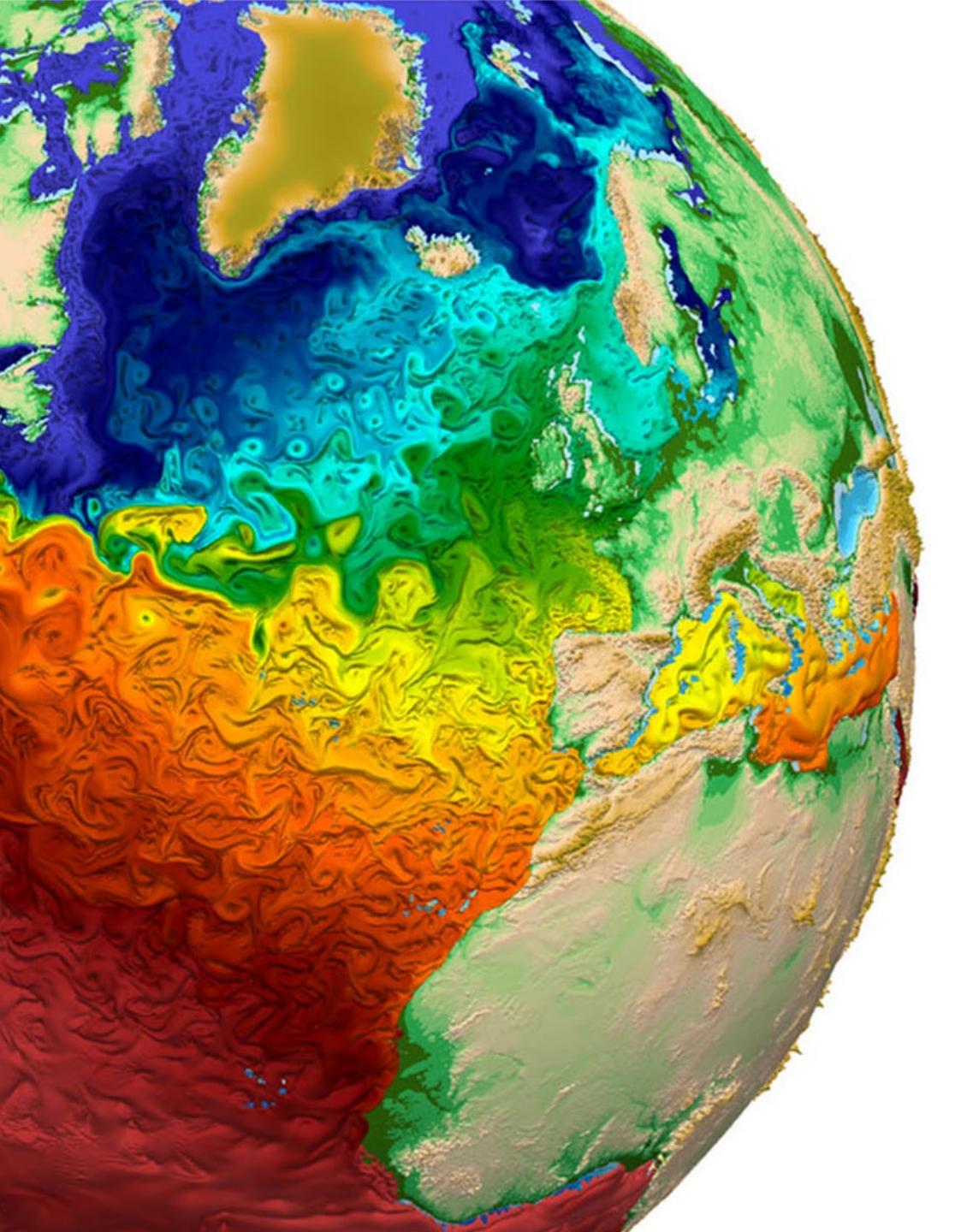


Simulation Data: Overset Meshes



- Used to solve partial differential equations
- Use two or more overlapping methods
- Data transfer between meshes via interpolation
- Challenging to visualize effectively
 - e.g. render multiple views





Domain Modeling

Triangulated Surface Meshes

Scientific Visualization
Professor Eric Shaffer

Polygonal Meshes

In rendering, we typically will generate an image by simulating the reflection of light off surfaces

Rasterization engines most often use polygonal meshes to represent surfaces

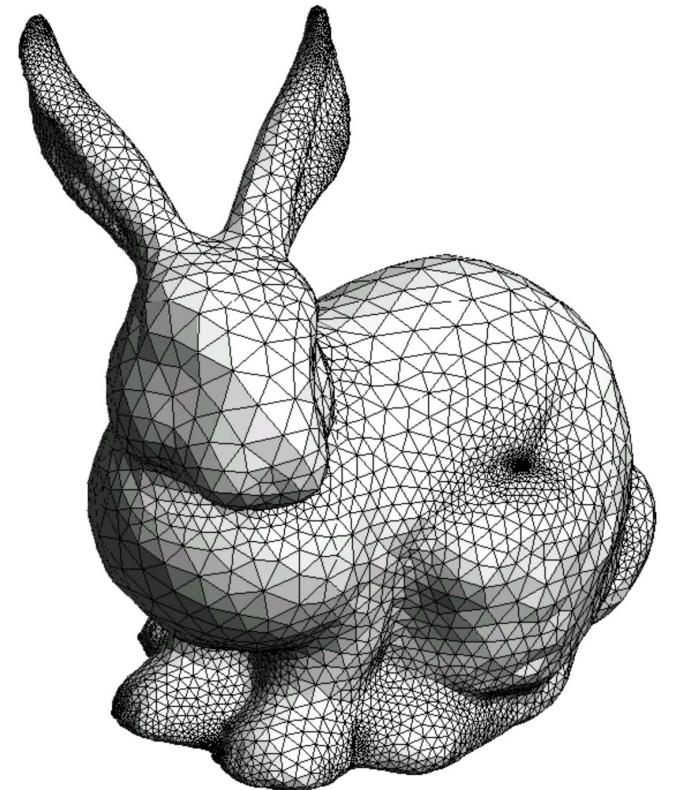
Modern GPUs are designed specifically to rasterize triangles

Many advantages to using triangles:

- Simplest 2D primitive...
- Any 2D polygon can be triangulated
- Can easily represent sharp surface features

Any disadvantages you can think of?

Why do we say 2D when
we are rendering in 3D?

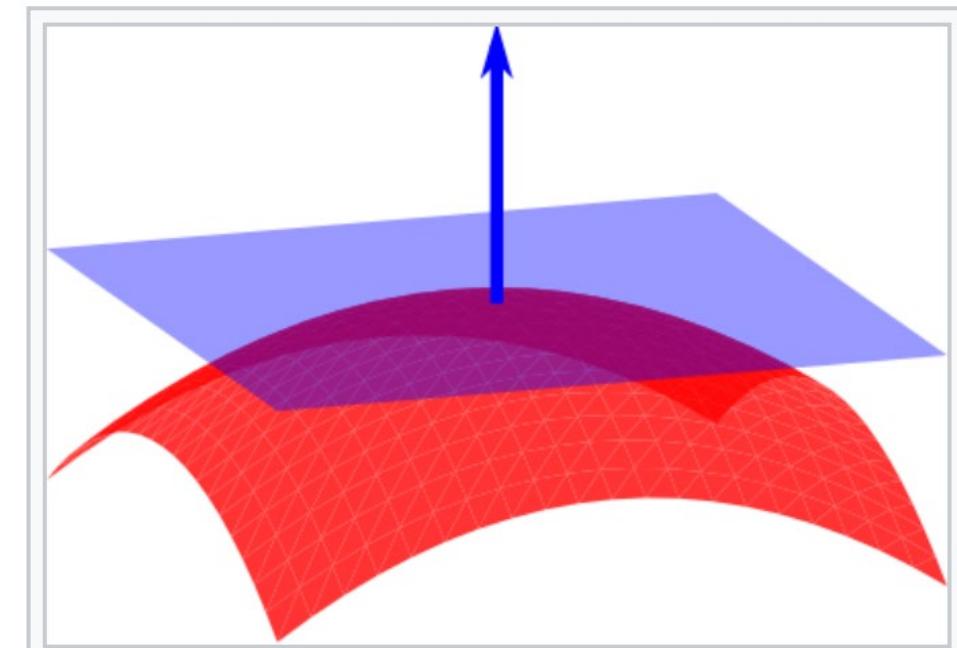
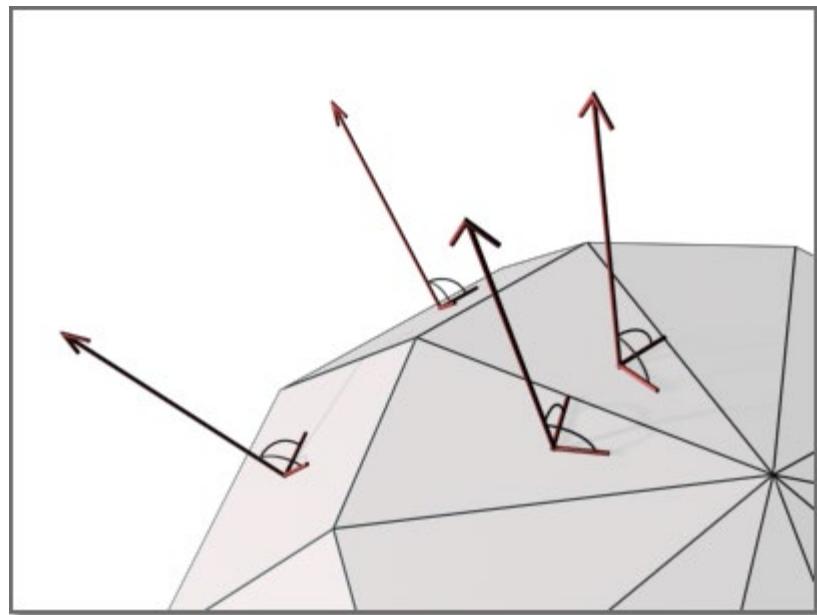


Vocabulary: Surface Normals

A normal is a vector that is perpendicular to object

Each triangle in a surface mesh has outward facing normal

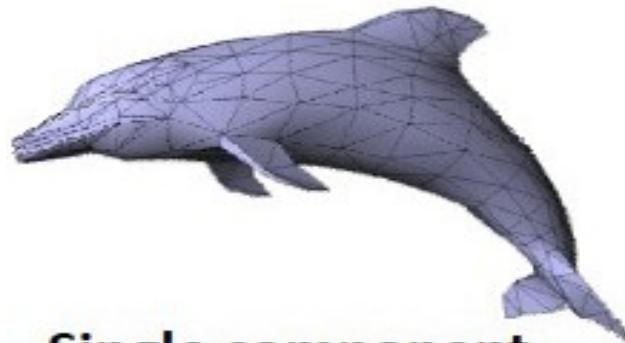
The normal is just the vector perpendicular to the triangle



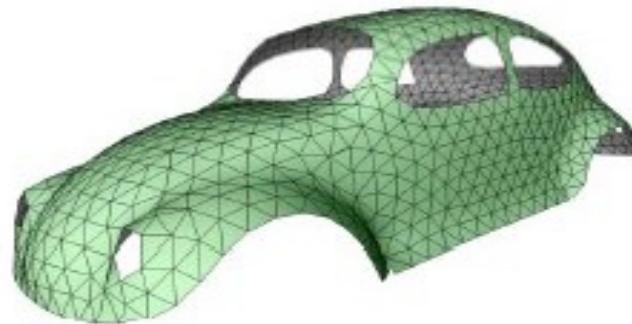
A normal to a surface at a point is
the same as a normal to the tangent
plane to the surface at the same point.

Courtesy Wikipedia

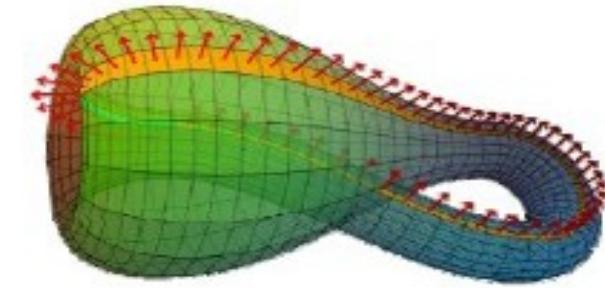
Vocabulary: Surface Mesh Properties



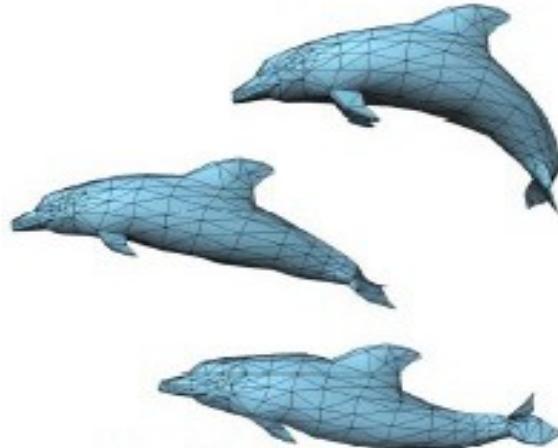
**Single component,
closed, triangular,
orientable manifold**



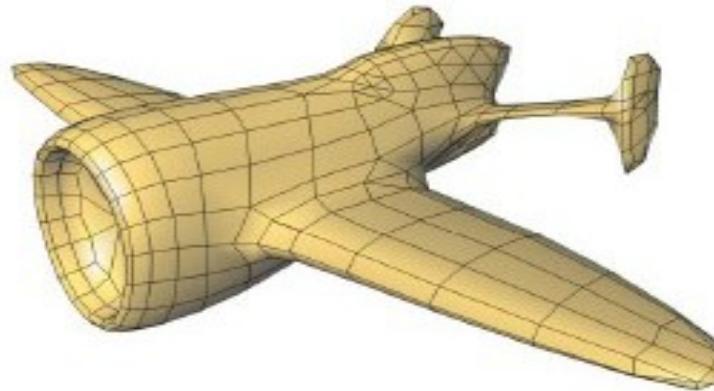
With boundaries



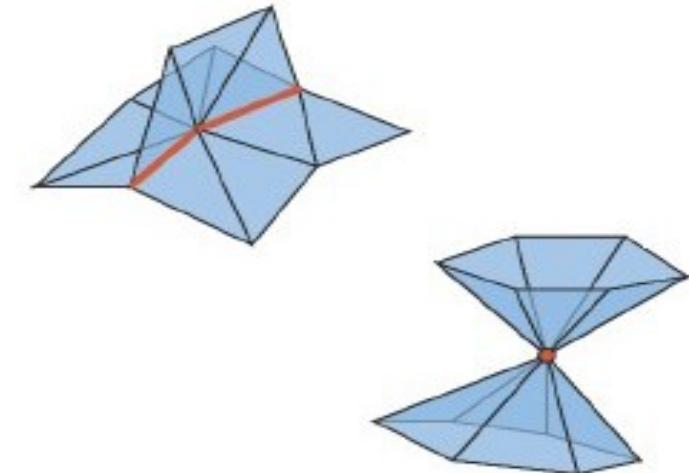
Not orientable



Multiple components



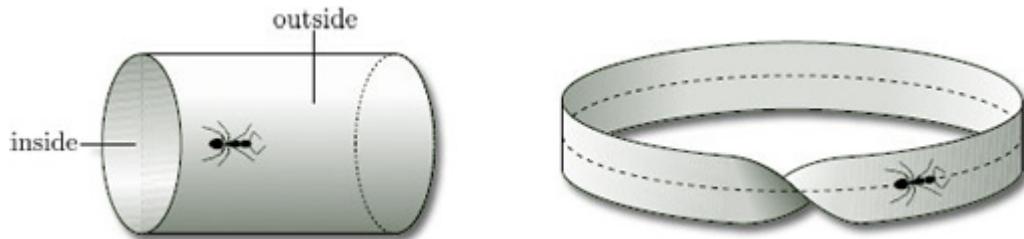
Not only triangles



Non manifold



Orientability



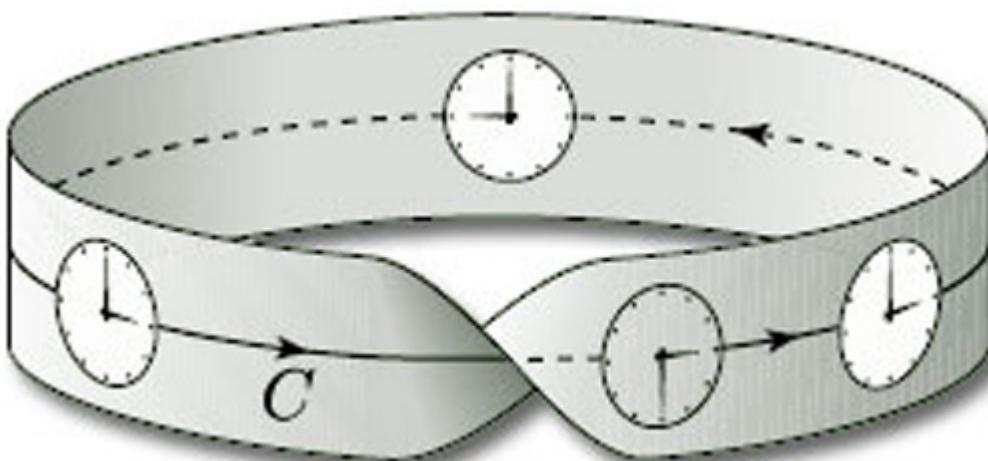
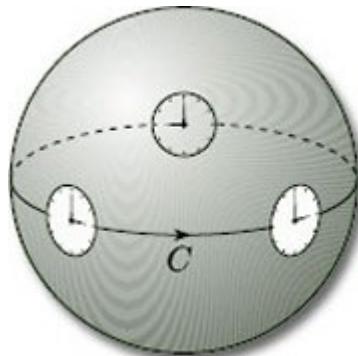
Not that relevant to this course....but it is interesting....

Imagine sliding a clock face around a surface.

On an orientable surface the clock face will return to starting point and appear the same. (Sphere)

On a non-orientable surface it will be a mirror image of the original clock face (Möbius Strip)

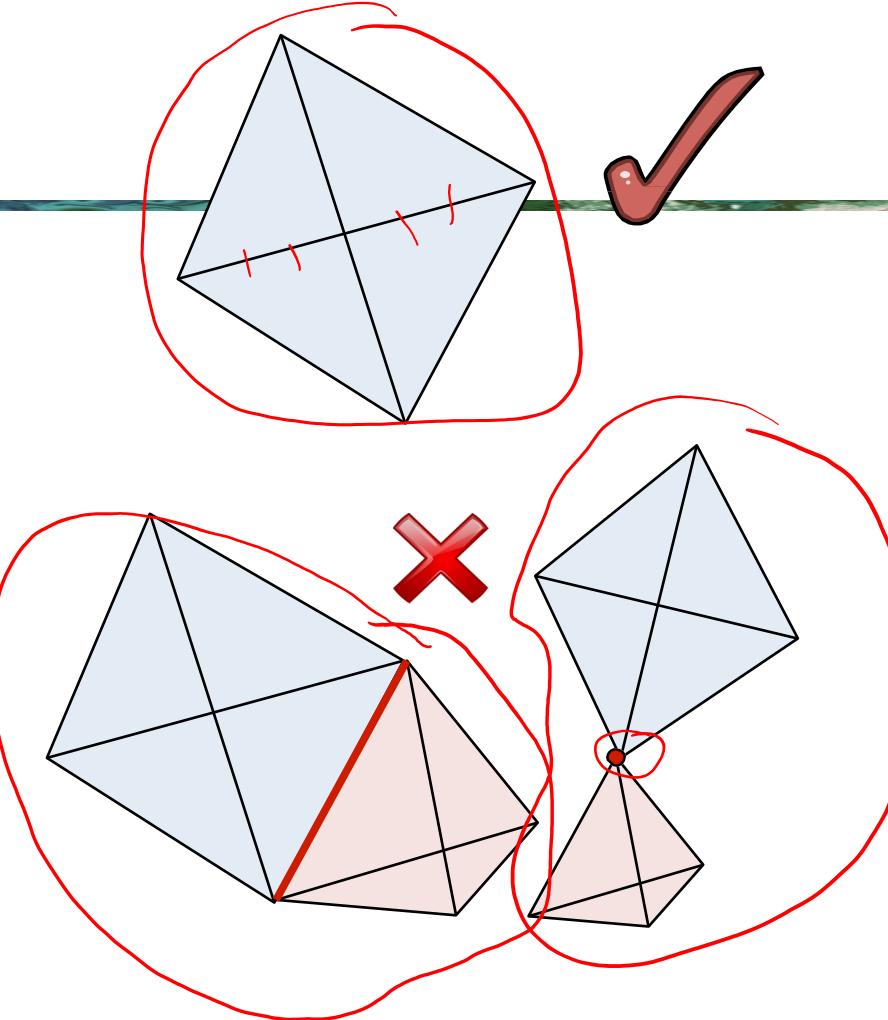
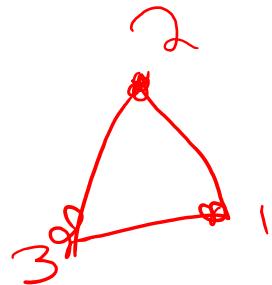
...there is no consistent definition of clockwise on a non-orientable surface.



Surface Mesh Properties

Manifold:

1. Every edge connects exactly two faces
2. Vertex neighborhood is “disk-like”



Orientable: Consistent normals

Watertight: Orientable + Manifold

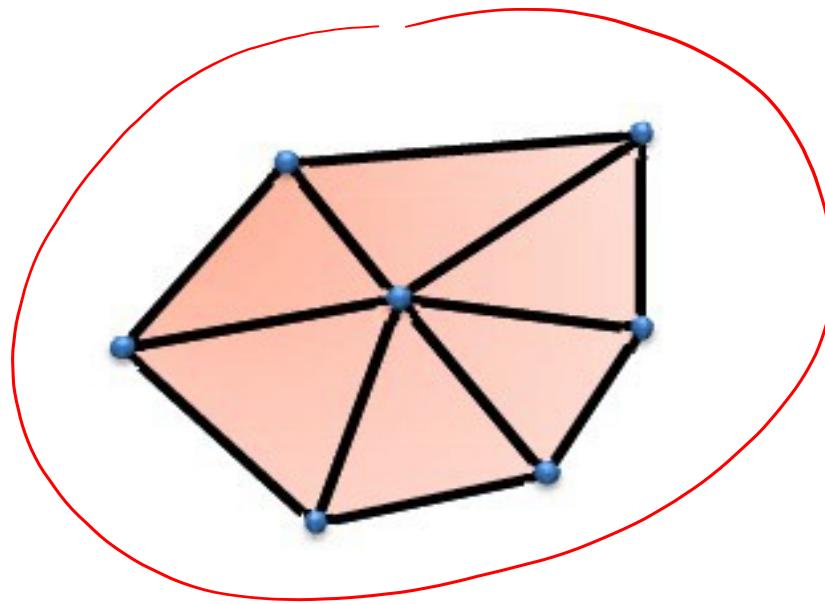
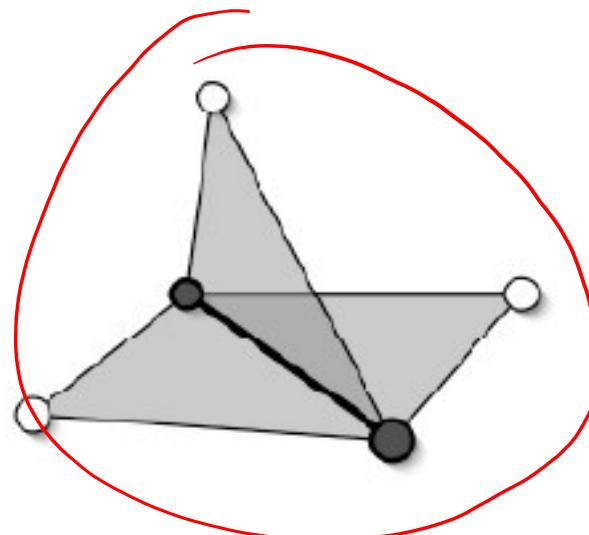
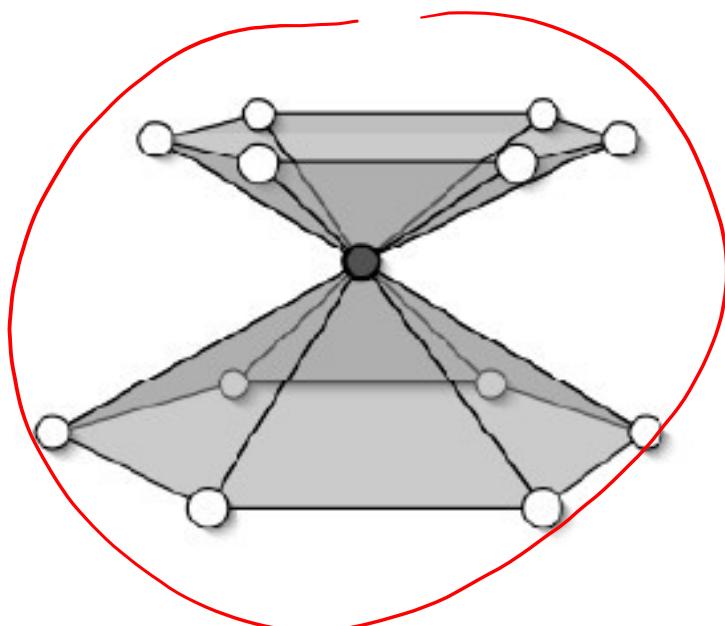
Boundary: Some edges bound only one face

Ordered: Vertices in CCW order when viewed from normal

The blue and pink meshes above are tetrahedra, like 4-sided pyramids

2-Manifold Mesh Examples

Disk-shaped neighborhoods



non-manifolds



Genus

Genus of orientable surfaces



genus 0



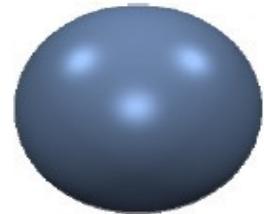
genus 1



genus 2



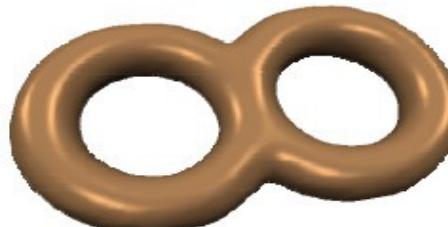
genus 3



Genus 0



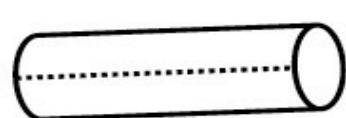
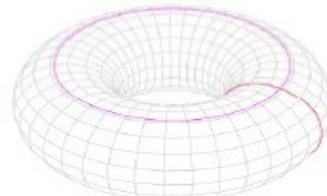
Genus 1



Genus 2



Genus ?



Euler Characteristic

For a closed (no boundary), manifold, connected surface mesh:

$$V - E + F = 2(1 - G)$$

$$V - E + F = \chi$$



V = number of vertices

E = number of edges

F = number of faces

G = genus (number of holes in the surface)

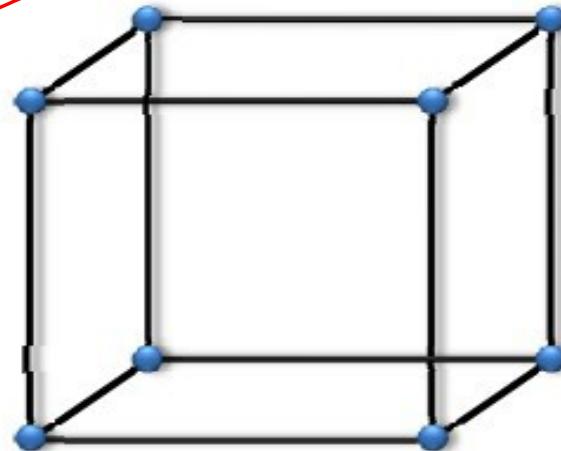
Leonhard Euler
1707 - 1783

Can you think of
anything else named
for him?

A **2-manifold** is a surface (locally like a plane)

Euler Characteristic for Closed 2-Manifold Polygonal Meshes

$$V + F - E = \chi$$



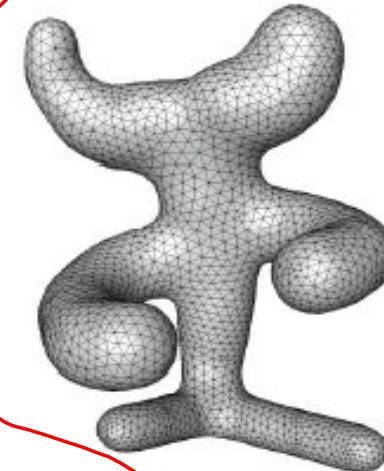
$$V = 8$$

$$E = 12$$

$$F = 6$$

$$\chi = 8 + 6 - 12 = 2$$

Euler characteristic



$$V = 3890$$

$$E = 11664$$

$$F = 7776$$

$$\chi = 2$$



...and if they are triangle meshes

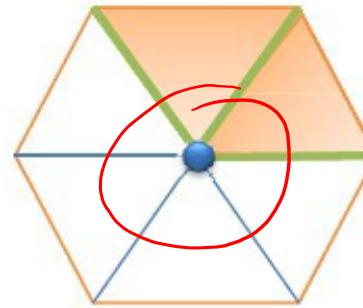
- *Triangle* mesh statistics

$$E \approx 3V$$

$$F \approx 2V$$

- Avg. valence ≈ 6

Show using Euler Formula



Aside from being totally interesting on their own, these formulas are useful for computing memory usage

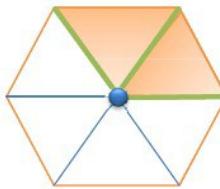


...and if they are triangle meshes

- Triangle mesh statistics

$$E \approx 3V$$

$$F \approx 2V$$



- Avg. valence ≈ 6

Show using Euler Formula



$$V - E + F = 2$$

$$V - \frac{3F}{2} + F = 2$$

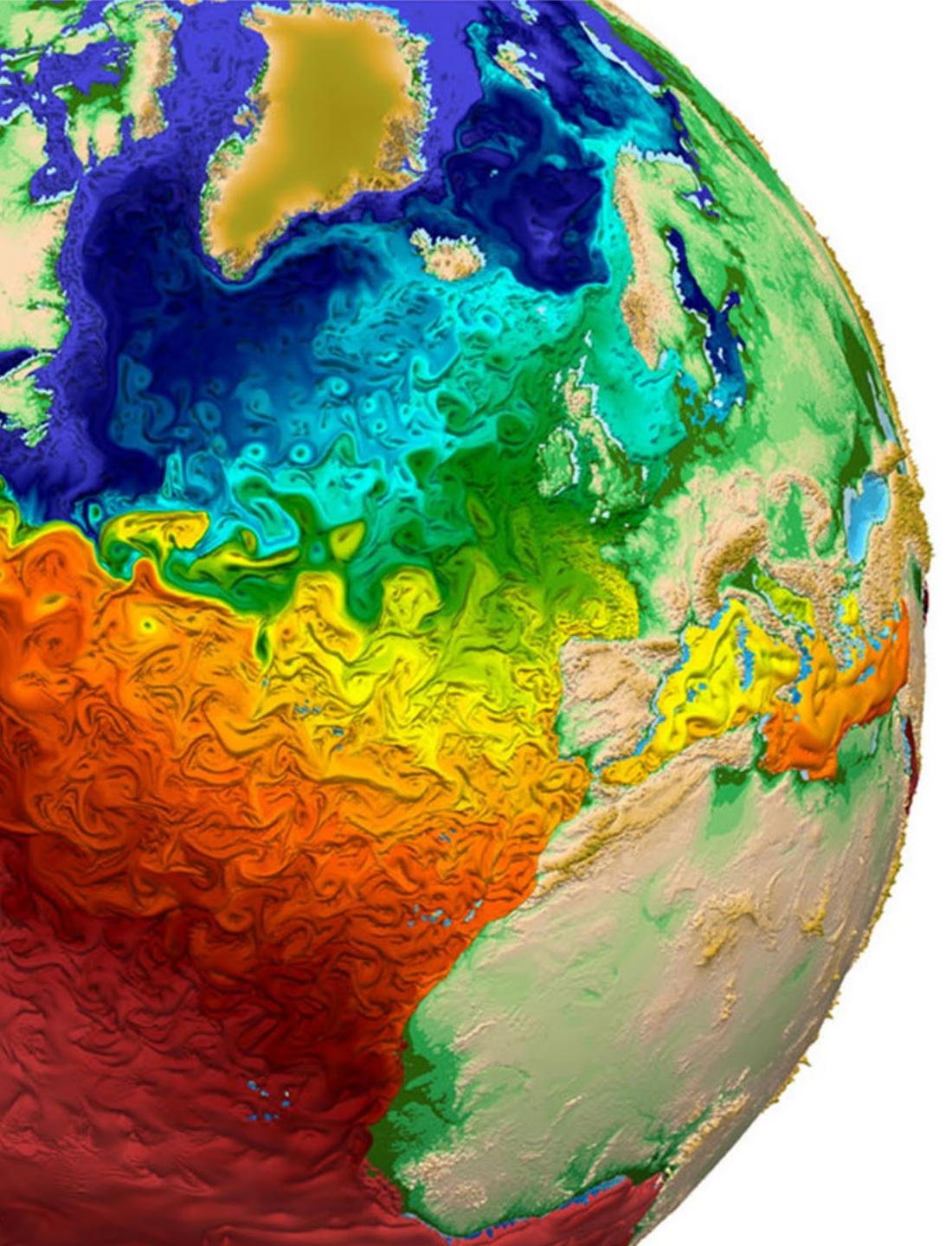
$$V - \frac{F}{2} = 2$$

$$V = \frac{F}{2} + 2$$

$$2V = F + 4$$

$$\boxed{2V \approx F}$$





Domain Modeling

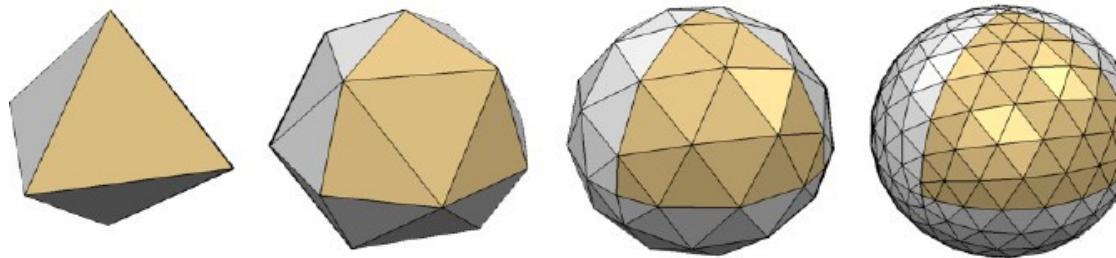
Data Structures for Polygonal Meshes

Scientific Visualization
Professor Eric Shaffer

Mesh Data Structures

Need to store

- Geometry
- Connectivity



Can be used as file formats or internal formats

Considerations

- Space
- Efficient operations

Mesh processing has different requirements than rendering

- Example: Deforming a mesh when simulating physics...

Mesh Data Structure: Face Set (STL)

- face:
 - 3 positions

Triangles								
Δ_1	$x_{11} \ y_{11} \ z_{11}$	$x_{12} \ y_{12} \ z_{12}$	$x_{13} \ y_{13} \ z_{13}$					
Δ_2	$x_{21} \ y_{21} \ z_{21}$	$x_{22} \ y_{22} \ z_{22}$	$x_{23} \ y_{23} \ z_{23}$					
					
	$x_{F1} \ y_{F1} \ z_{F1}$	$x_{F2} \ y_{F2} \ z_{F2}$	$x_{F3} \ y_{F3} \ z_{F3}$					

$2v \approx f$

$36 \text{ B/f} \neq 72 \text{ B/v}$
no connectivity!

Designed in 1987 for stereolithography (3D printing technology)

No explicit information about which vertices are shared by which triangles
Consider how efficiently we could:

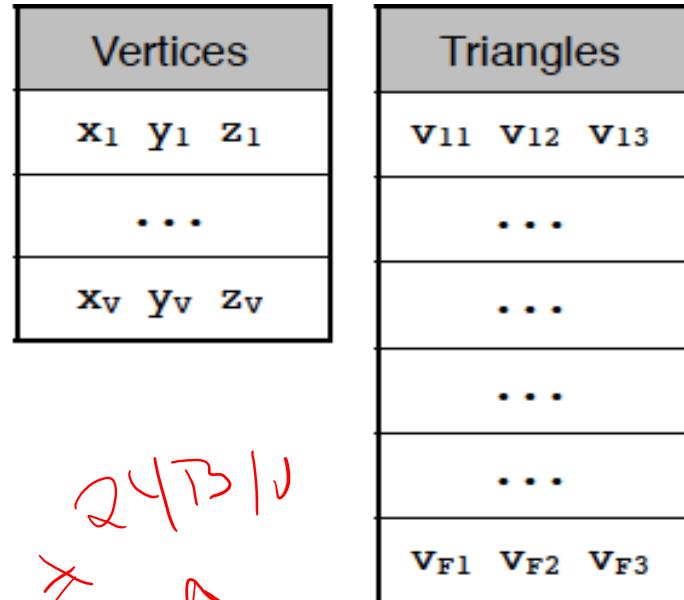
- Deform mesh by moving vertices
- Lookup vertex neighbor information

We will be focusing on triangle meshes...but all of these structures generalize to other polygons...storage requirements will change.



Mesh Data Structure: Indexed Face Set (OBJ)

- vertex:
 - position
- face:
 - vertex indices



24 B/v

12 B/v + 12 B/f = 36 B/v

no neighborhood info

$$2v \approx F$$

The OBJ file format is a popular storage format for meshes

Developed by Wavefront Technologies...now part of AutoDesk

Text files with .obj extension

```
# List of geometric vertices  
v 0.123 0.234 0.345  
v ...  
v ...  
# List of triangles  
f 1 3 4  
f 2 4 5  
...
```



Indexed Face Set

One block of data are the vertices

- Each vertex is a set of 3 coordinates
- Often referred to as the geometry of the mesh

Another block of data is the set of triangles

- Each triangle is set of 3 integers vertex IDs
- The vertex IDs are indices into the vertex block

Vertices	Triangles
$x_1 \ y_1 \ z_1$	$v_{11} \ v_{12} \ v_{13}$
...	...
$x_v \ y_v \ z_v$	$v_{F1} \ v_{F2} \ v_{F3}$
...	...
...	...
...	...
...	...

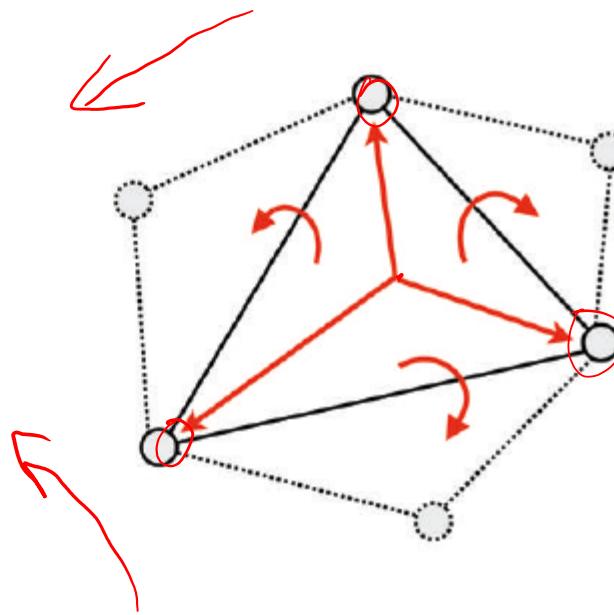
What are some advantages of this representation?

Disadvantages?

Face-Based Connectivity

Vertex	
Point	position
FaceRef	face

Face	
VertexRef	vertex[3]
FaceRef	neighbor[3]



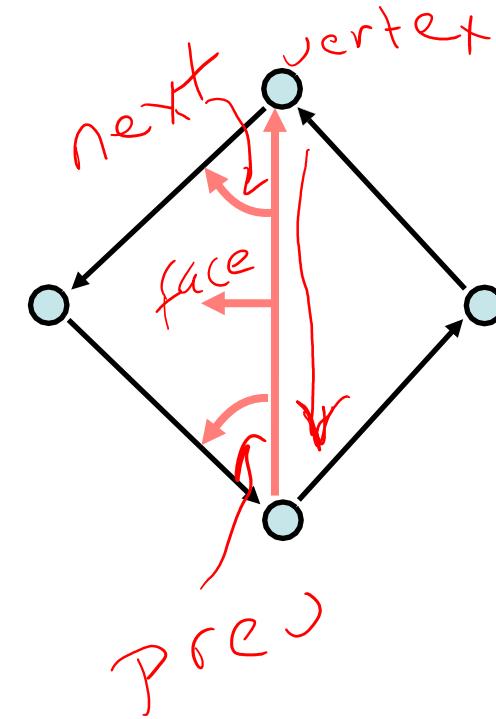
Storage: 64 B/v

No edges: gathering vertex neighbors is not straight-forward...multiple cases

Halfedge Data Structure

Vertex	
Point position	
HalfedgeRef halfedge	
Face	
	HalfedgeRef halfedge

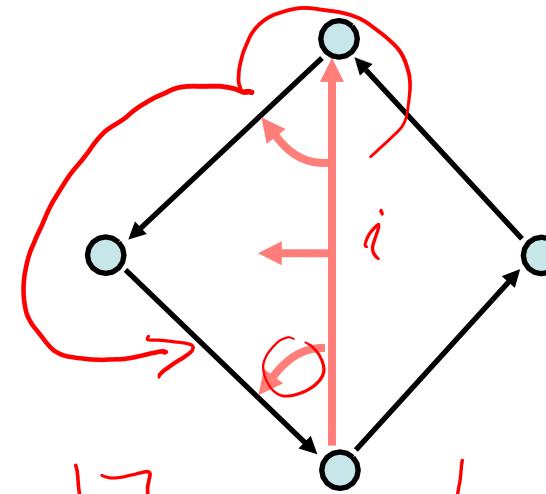
Halfedge	
VertexRef vertex	
FaceRef face	
HalfedgeRef next	
HalfedgeRef prev	
HalfedgeRef opposite	



Halfedge Data Structure

H	
Vertex	
Point	position
HalfedgeRef halfedge	
Face	
HalfedgeRef halfedge	

Halfedge	
VertexRef	vertex
FaceRef	face
HalfedgeRef	next
HalfedgeRef	prev
HalfedgeRef	opposite



$$H[i].prev = H[H[i].next].next$$

$H[0] \xrightarrow{opp}$
 $H[1] \xrightarrow{opp}$

if i is even $\rightarrow H[i].opp = i + 1$
 if i is odd $\rightarrow H[i].opp = i - 1$

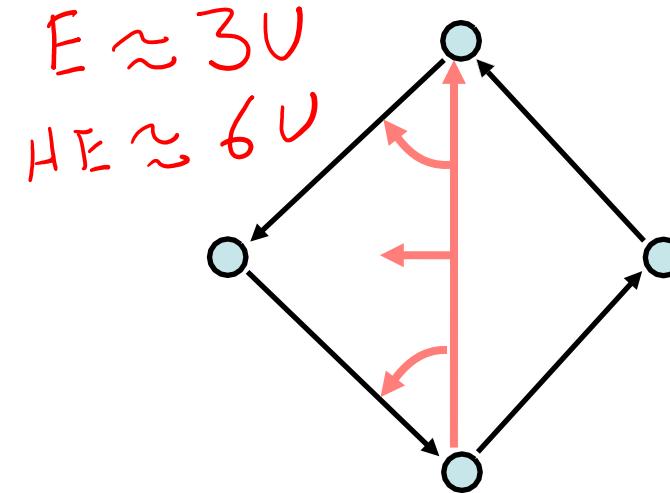
Halfedge Data Structure

Vertex	
Point	position
HalfedgeRef	halfedge

Face	
HalfedgeRef	halfedge

Halfedge	
VertexRef	vertex
FaceRef	face
HalfedgeRef	next
HalfedgeRef	prev
HalfedgeRef	opposite

$$\begin{aligned}
 & \text{vertex} + \text{Face} + \text{Halfedge} \\
 & 16 \text{ B/v} + 4 \text{ B/F} + 12 \text{ B/H} \\
 & 16 \text{ B/v} + 8/\text{v} + 72 \text{ B/H} = \boxed{96 \text{ B/v}}
 \end{aligned}$$



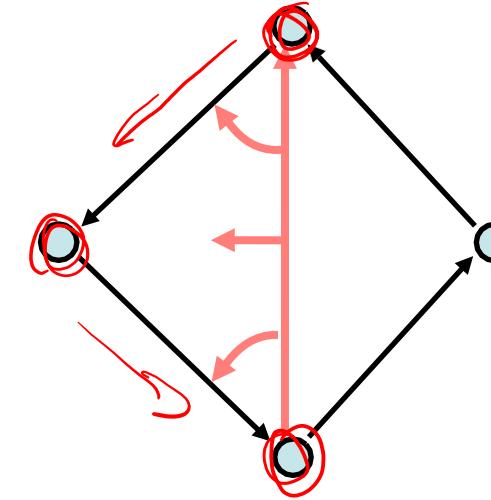
$$\begin{aligned}
 E &\approx 3V \\
 HE &\approx 6V
 \end{aligned}$$

96 to 144 B/v
 no case distinctions
 during traversal

Halfedge Data Structure : Gathering Vertices

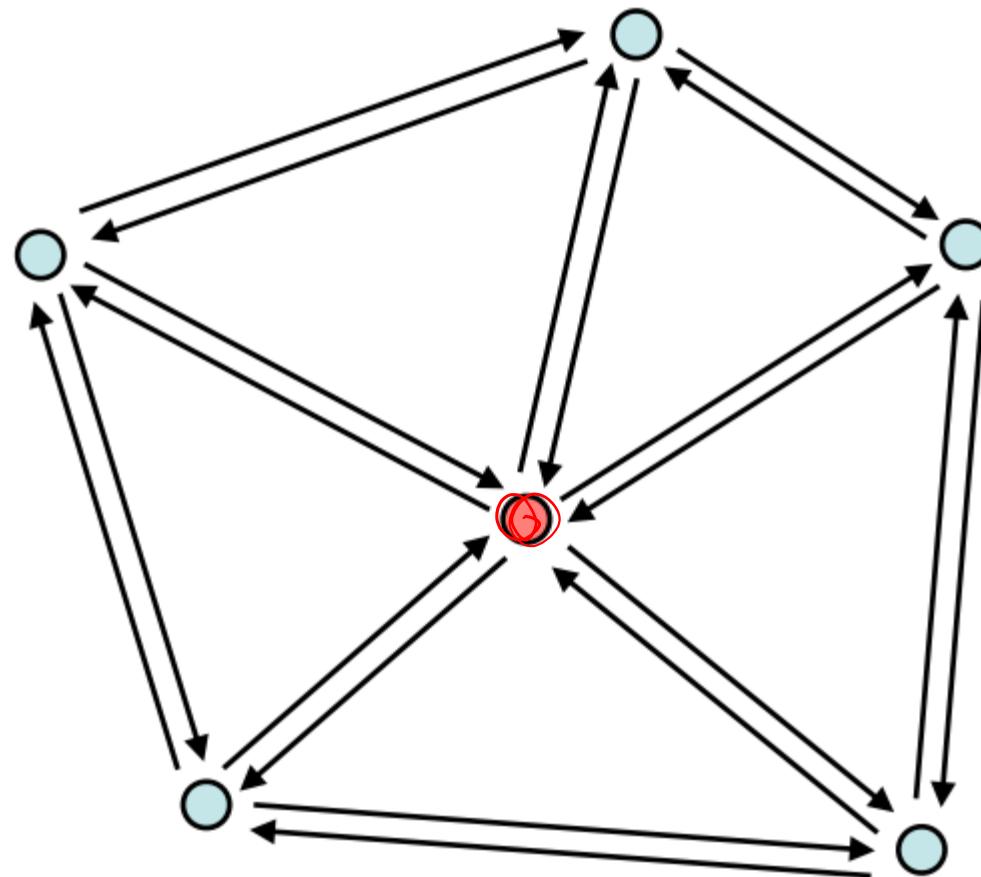
Vertex	
Point position	
HalfedgeRef halfedge	
Face	
	HalfedgeRef halfedge

Halfedge	
VertexRef vertex	
FaceRef face	
HalfedgeRef next	
HalfedgeRef prev	
HalfedgeRef opposite	



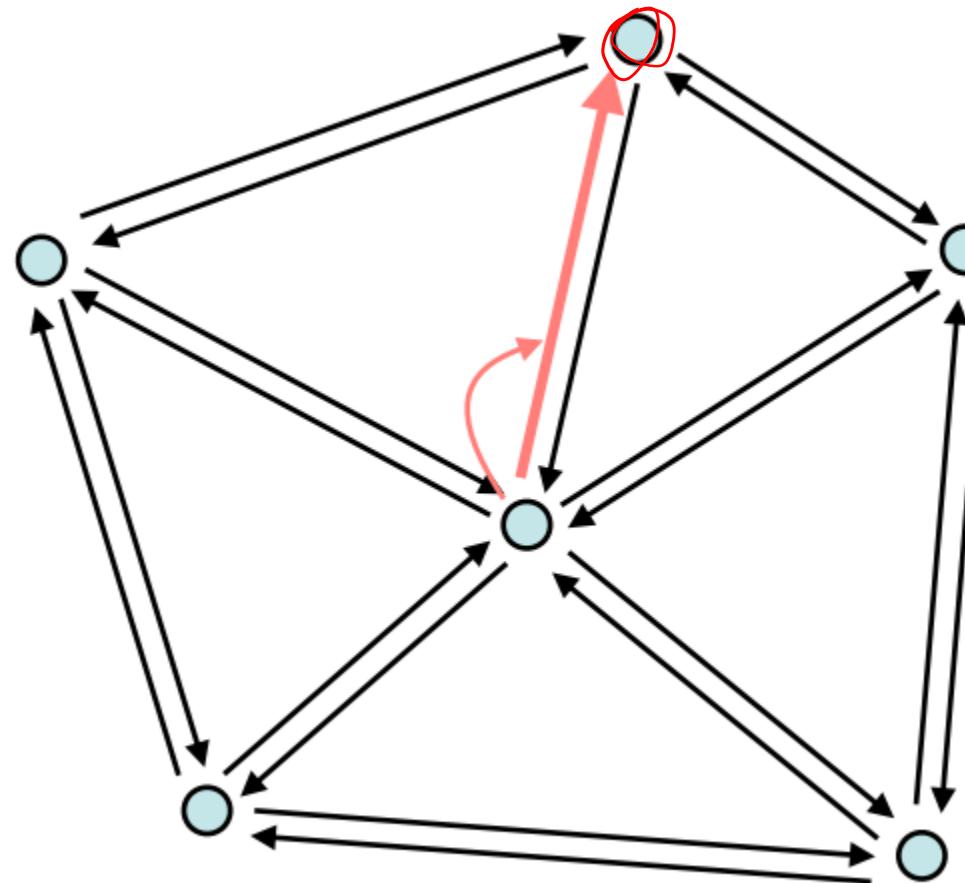
One-Ring Traversal

1. Start at vertex



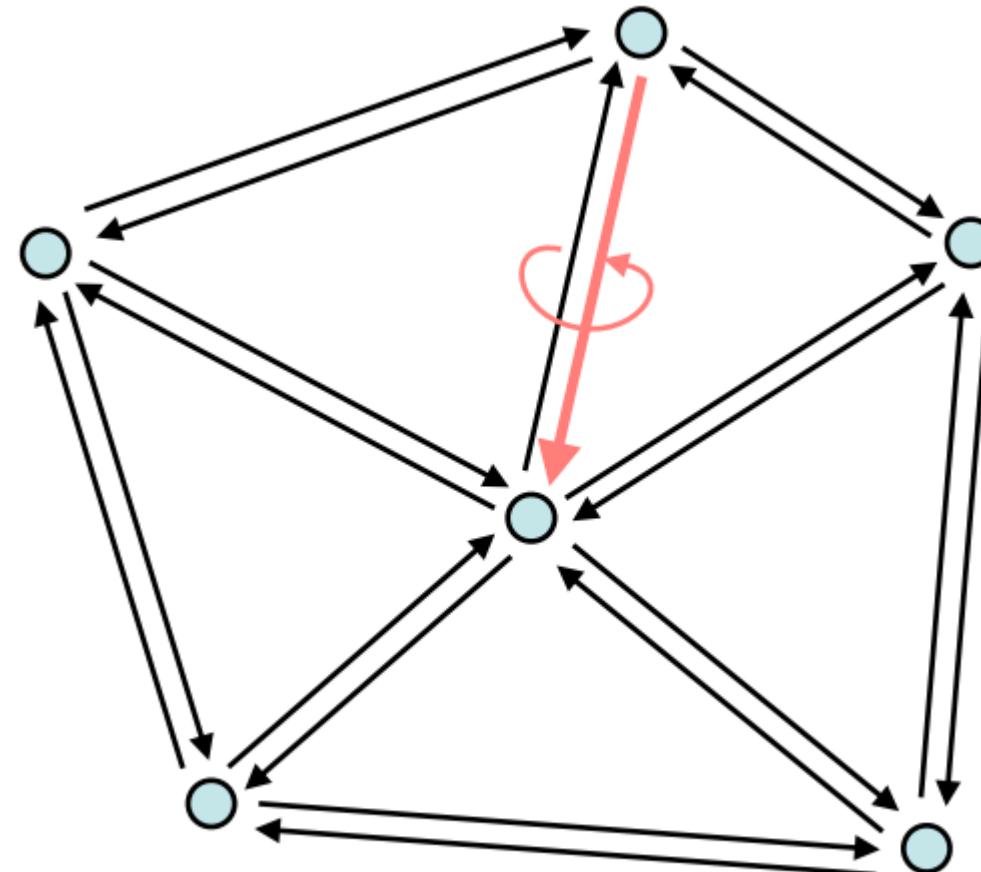
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge



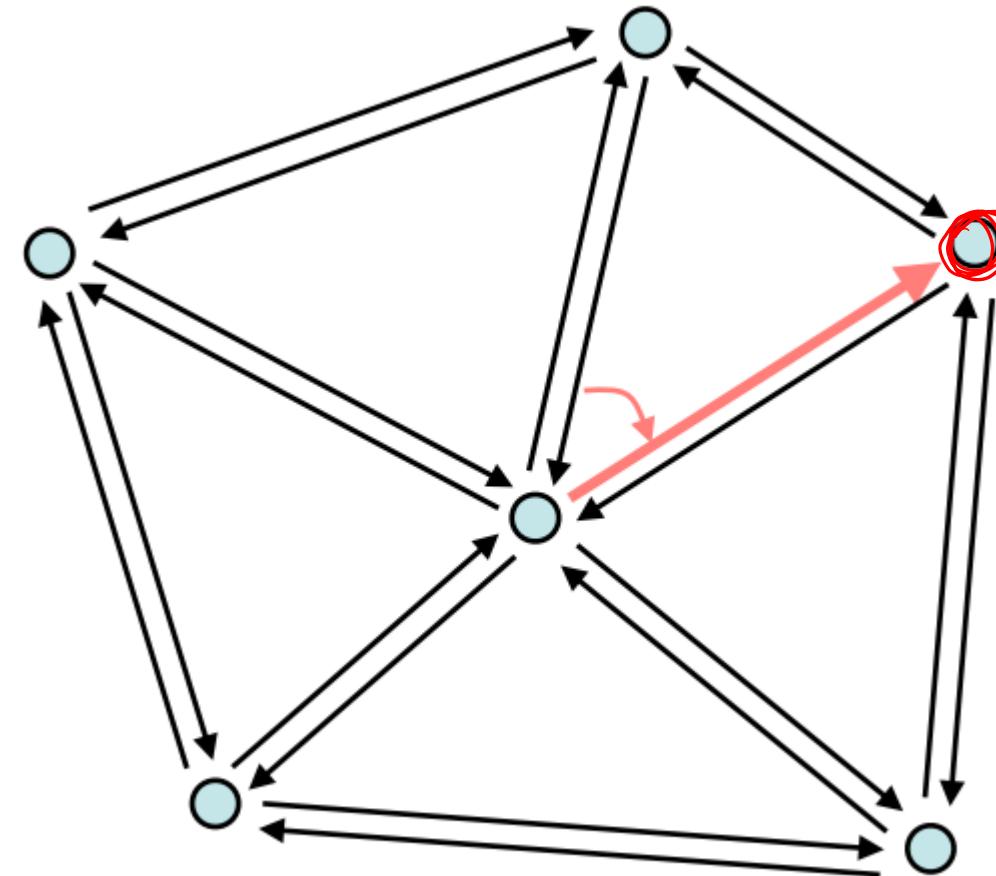
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge



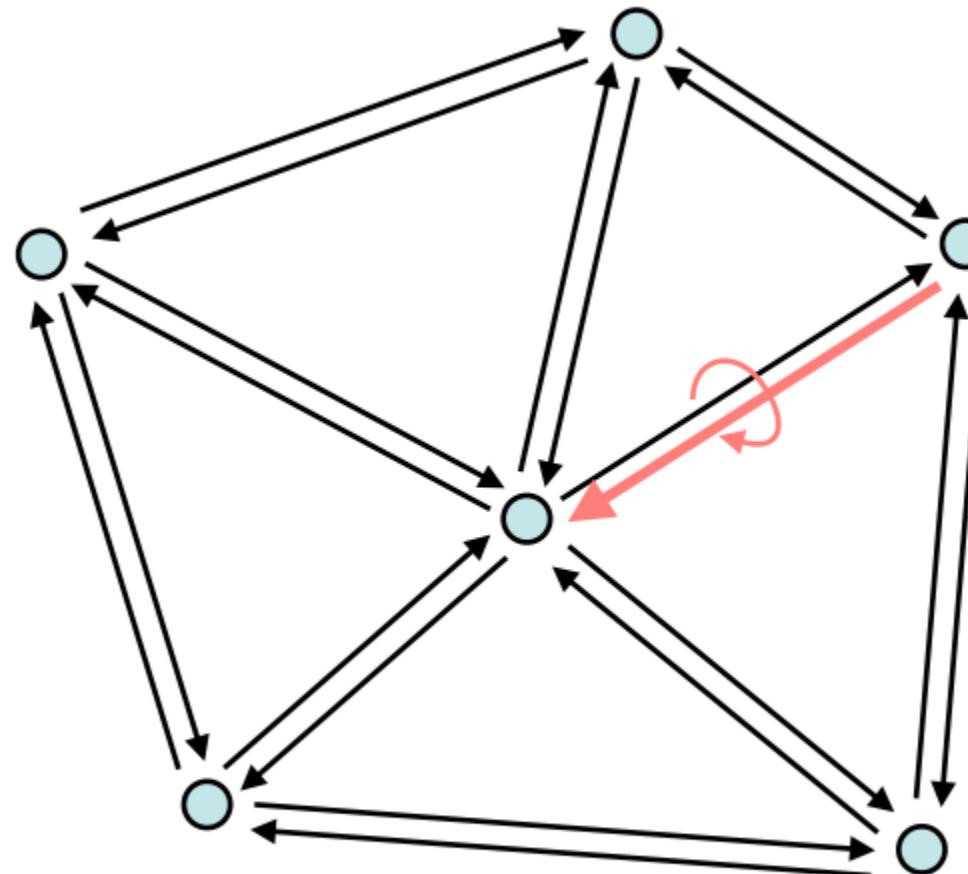
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge



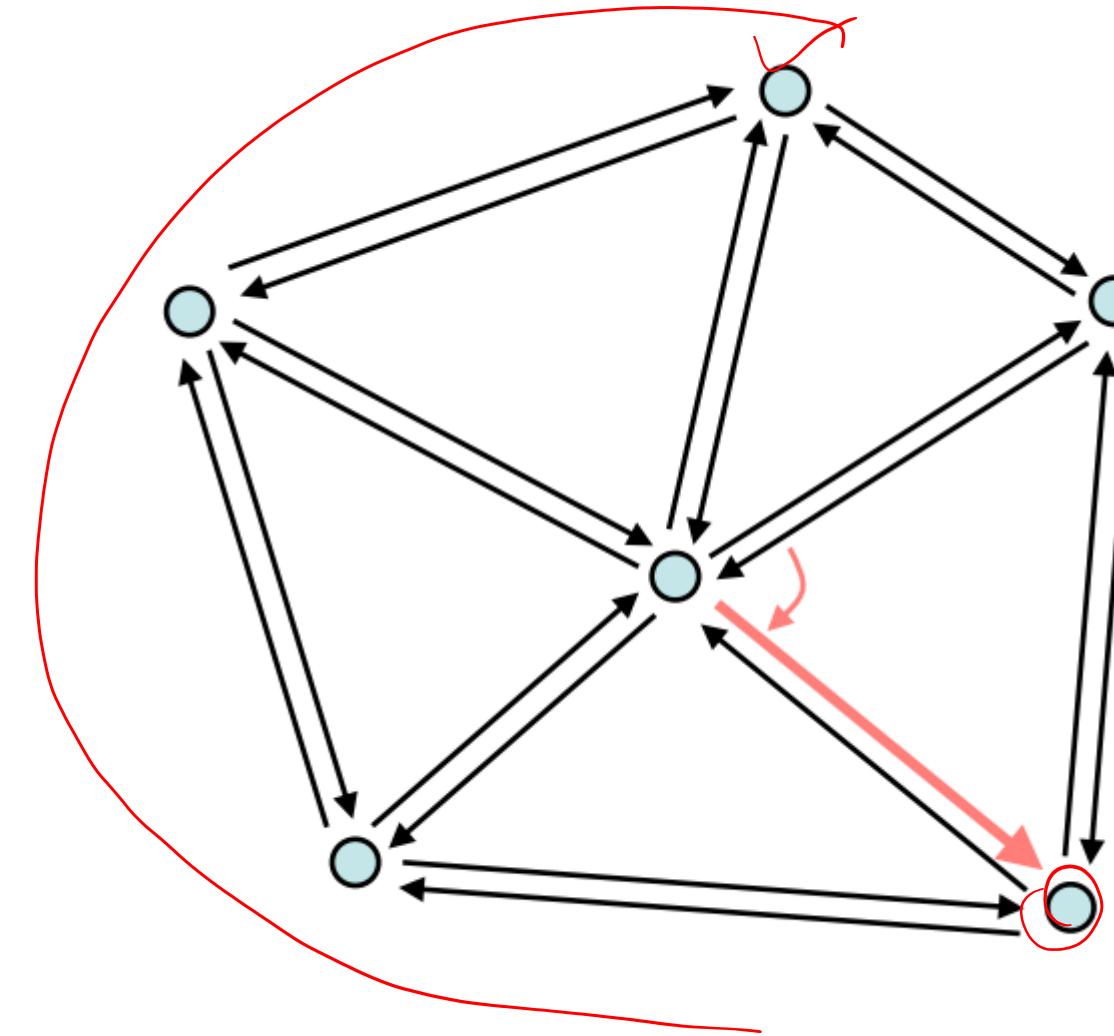
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite



One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite
6. Next
7. ...

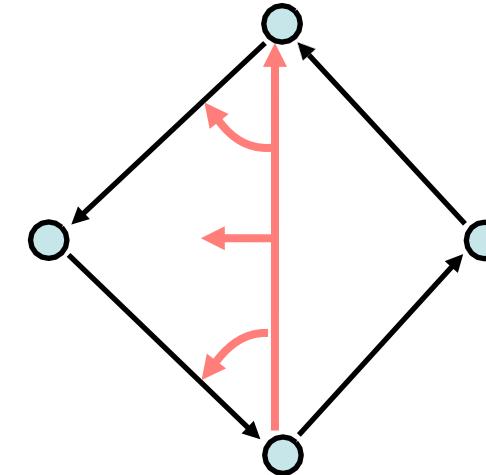


Halfedge Data Structure

Vertex	
Point	position
HalfedgeRef	halfedge

Face	
HalfedgeRef	halfedge

Halfedge	
VertexRef	vertex
FaceRef	face
HalfedgeRef	next
HalfedgeRef	prev
HalfedgeRef	opposite



Advantages:

- simple and efficient traversals of vertex neighborhoods ~~*~~
- can be applied to any polygonal mesh

Where to See Them in the Wild....

Computational Geometry Libraries

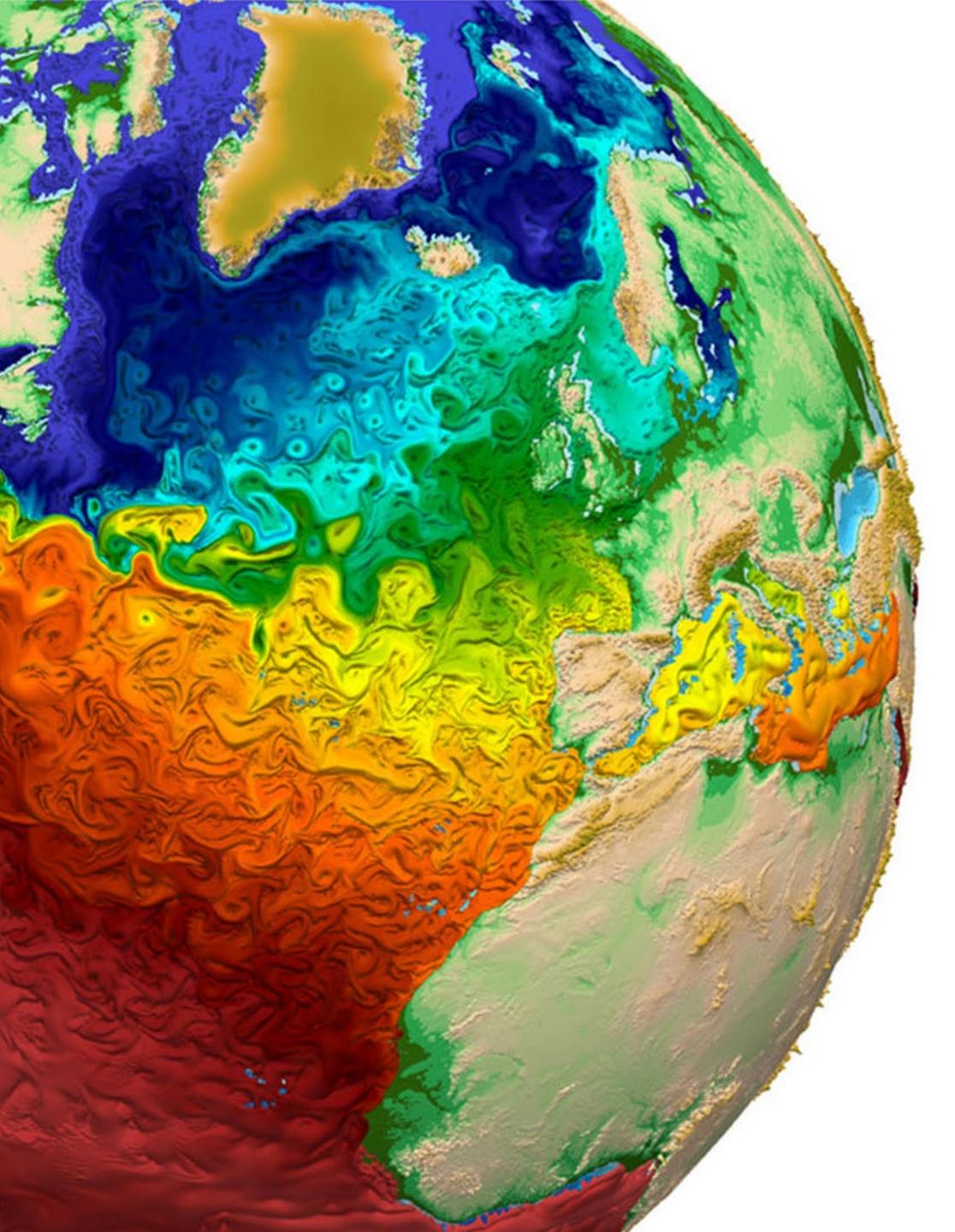
- CGAL
- OpenMesh

File Formats

- VTK (Visualization Toolkit)
- OBJ

More advanced formats as well...

- gltf for streaming (graphics file format...)
- cgns (computational fluid dynamics)
- ...lots of discipline specific formats



Domain Modeling

Terrain Generation for GIS

Scientific Visualization
Professor Eric Shaffer

Acknowledgment

We will look at a proposed method for generating

- Raster DEMs (Digital Elevation Models)
- TINs (Triangulated Irregular Networks)

From LiDAR (laser imaging, detection, and ranging) data

The proposed method is from

Generating Raster DEM from Mass Points Via TIN Streaming. Isenburg M., Liu Y., Shewchuk J., Snoeyink J., Thirion T. (2006) In: Raubal M., Miller H.J., Frank A.U., Goodchild M.F. (eds) Geographic Information Science. GIScience 2006. Lecture Notes in Computer Science, vol 4197. Springer, Berlin, Heidelberg.

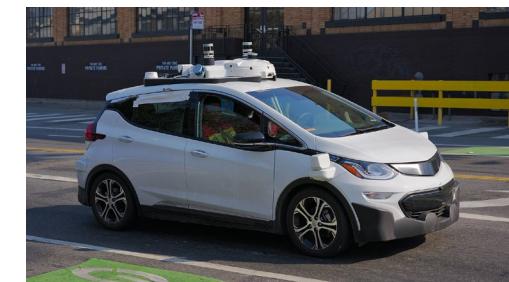
https://doi.org/10.1007/11863939_13

Content in this presentation comes courtesy of Martin Isenburg's publicly available slides

LiDAR

Lidar is a method for measuring distances (ranging) by illuminating the target with laser light and measuring the reflection with a sensor. Differences in laser return times and wavelengths can then be used to make digital 3-D representations of the target. It has terrestrial, airborne, and mobile applications.

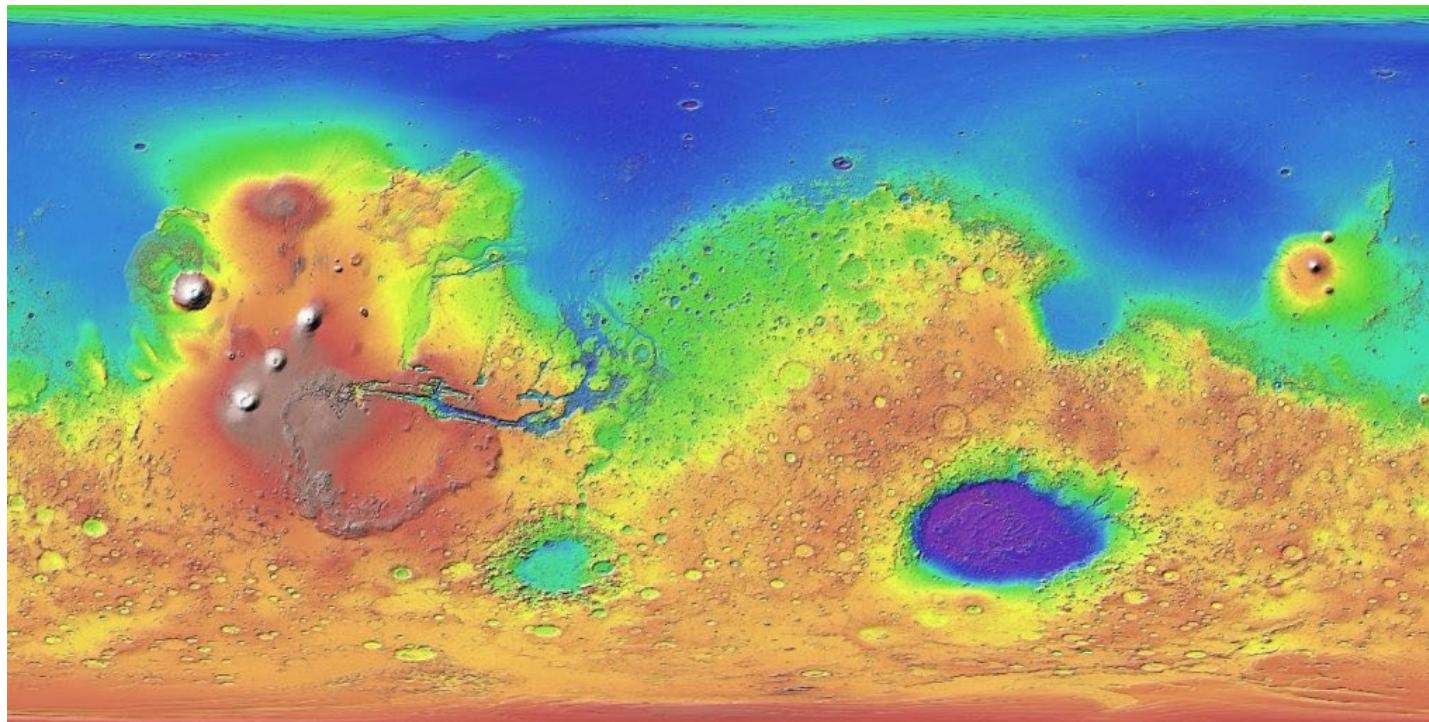
- Wikipedia



DEMs

A digital elevation model (DEM) is a raster data set

A regular grid of elevations arranged by column and row



Mars MGS MOLA Global Color Shaded Relief 463m v1

Visualization and DEMs:

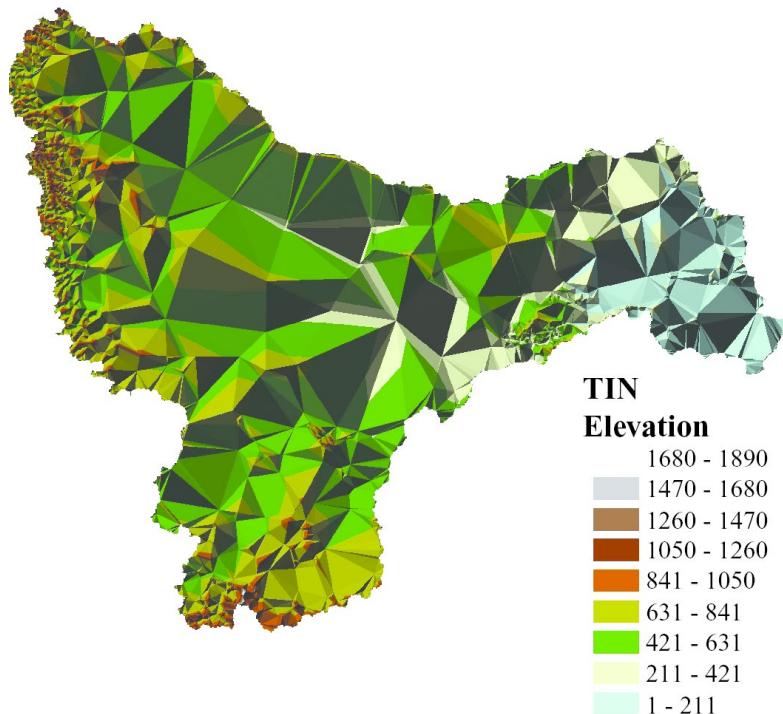
- estimate normal vectors
 - generate shading
- visualize slope
- contour lines

and more...

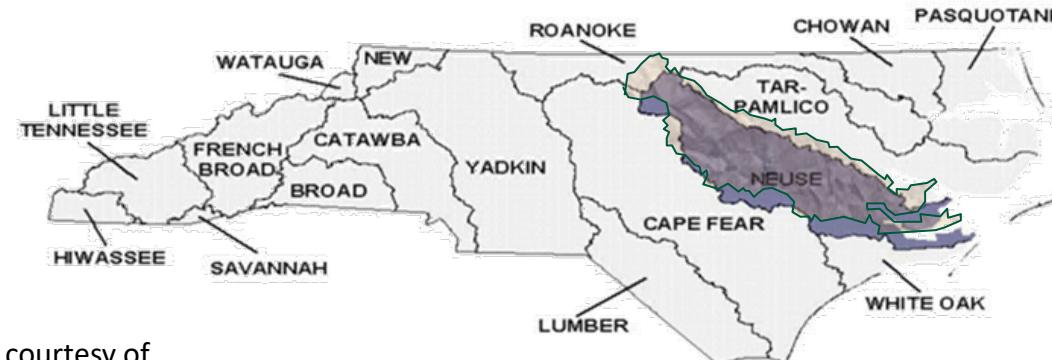
TINs

Triangulated Irregular Networks

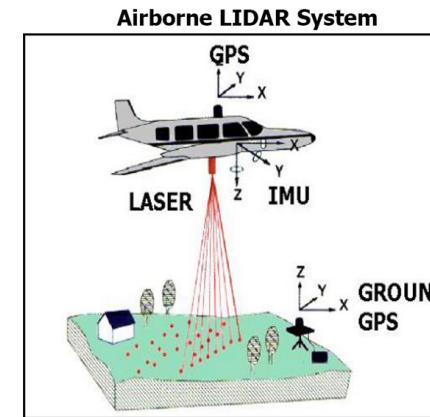
- Triangulated surface mesh of a terrain
- Irregular = unstructured, allows adaptive sampling of areas with rapidly changing elevations



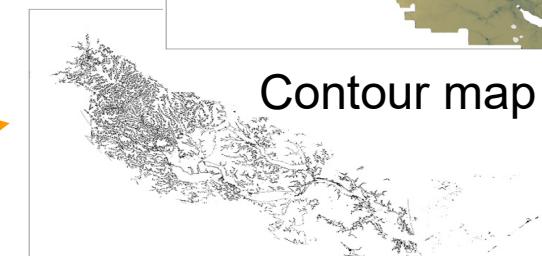
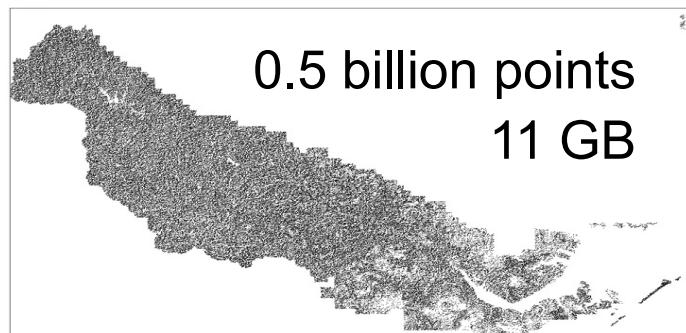
Aerial LiDAR Acquisition (~2005)



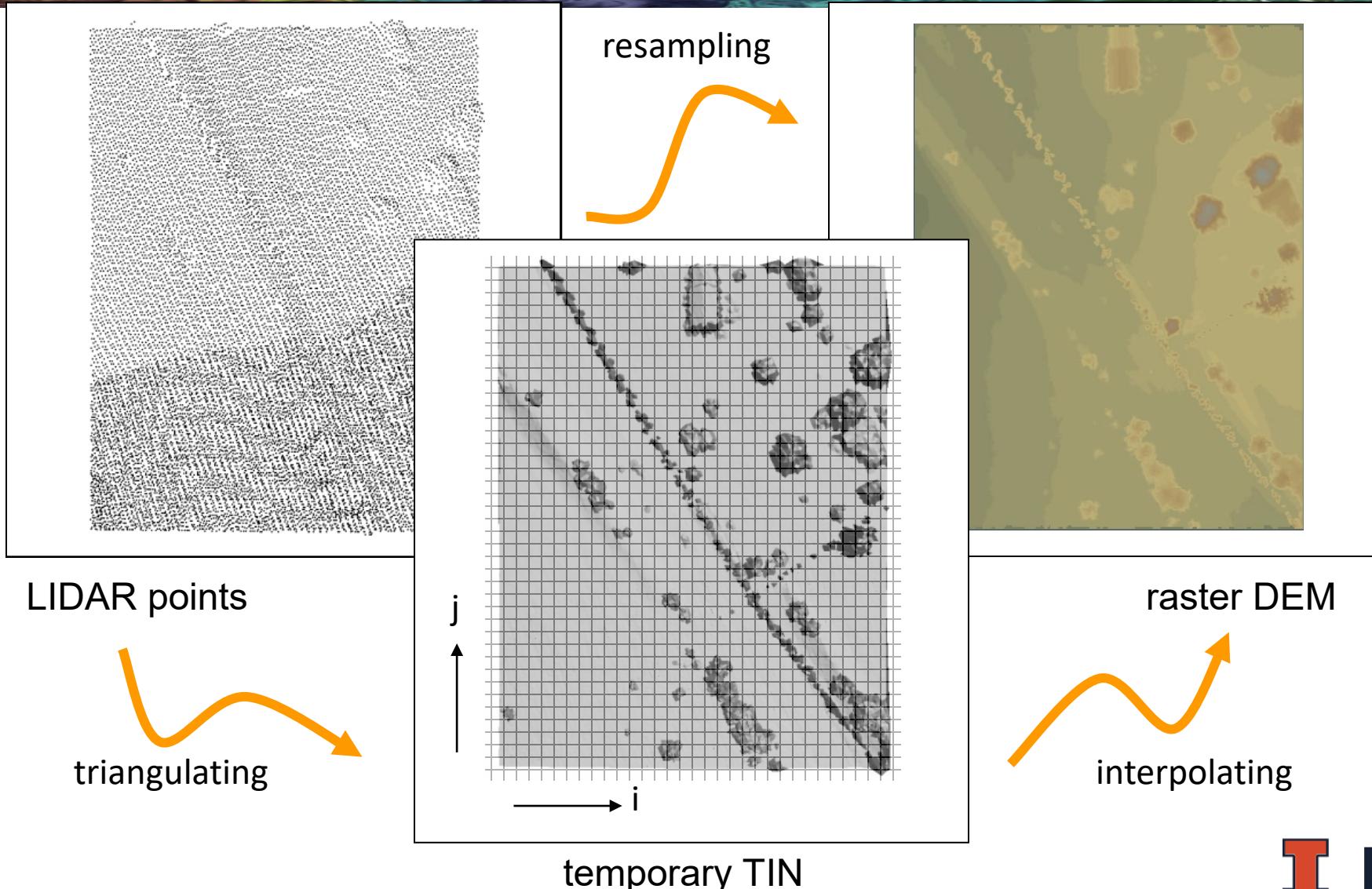
courtesy of
www.ncfloodmaps.com



Neuse-River
Basin



Generating DEMs from LIDAR

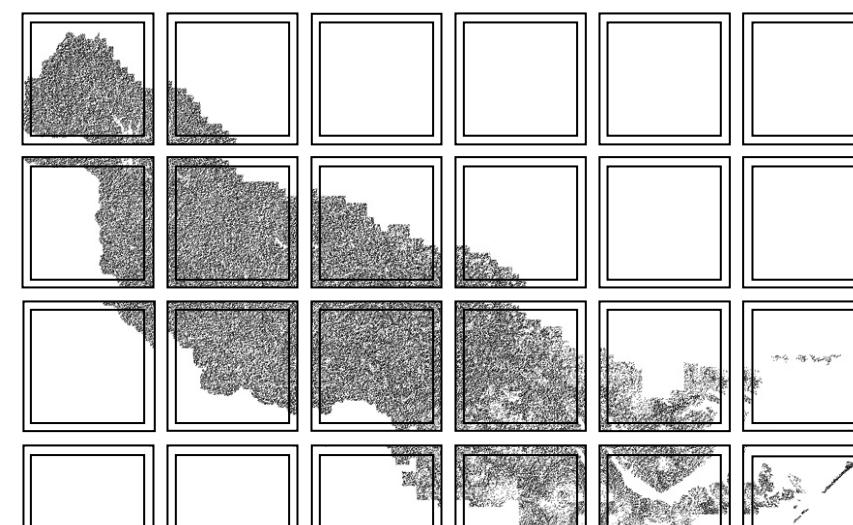
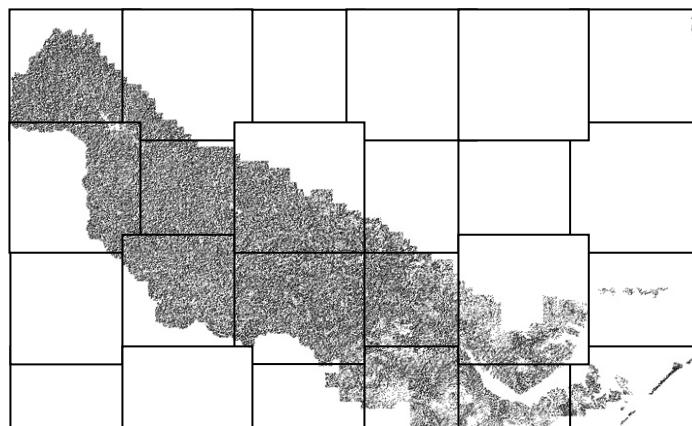


Goal: Make DEM Production from LiDAR Scalable

In 2005 popular GIS packages could process

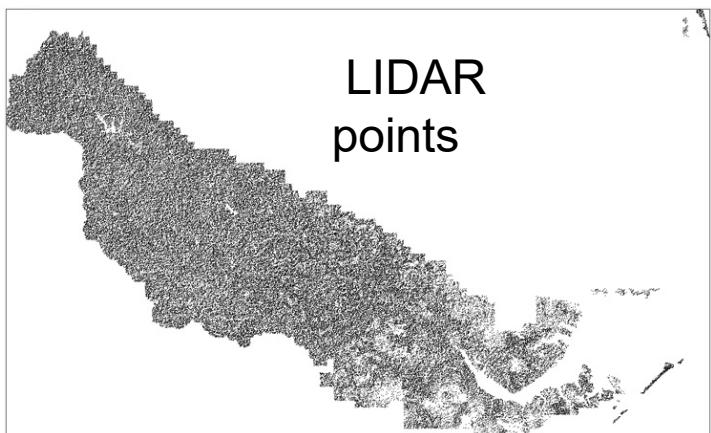
- ArcGIS 9.1 (limit: ~ 25 million points)
- QTModeller 4 (limit: ~ 50 million points)
- GRASS (limit: ~ 25 million points)

Popular strategy: break data into (overlapping) tiles

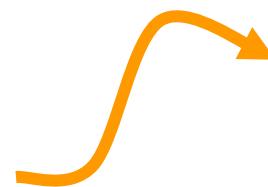
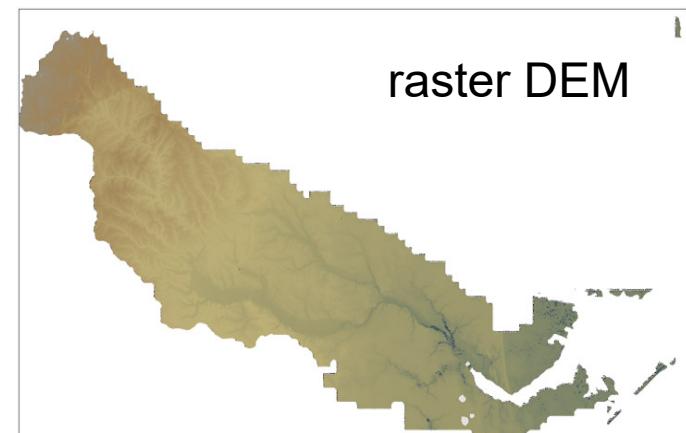


Some Results

500,141,313 Points
11 GB
(binary, xyz, doubles)

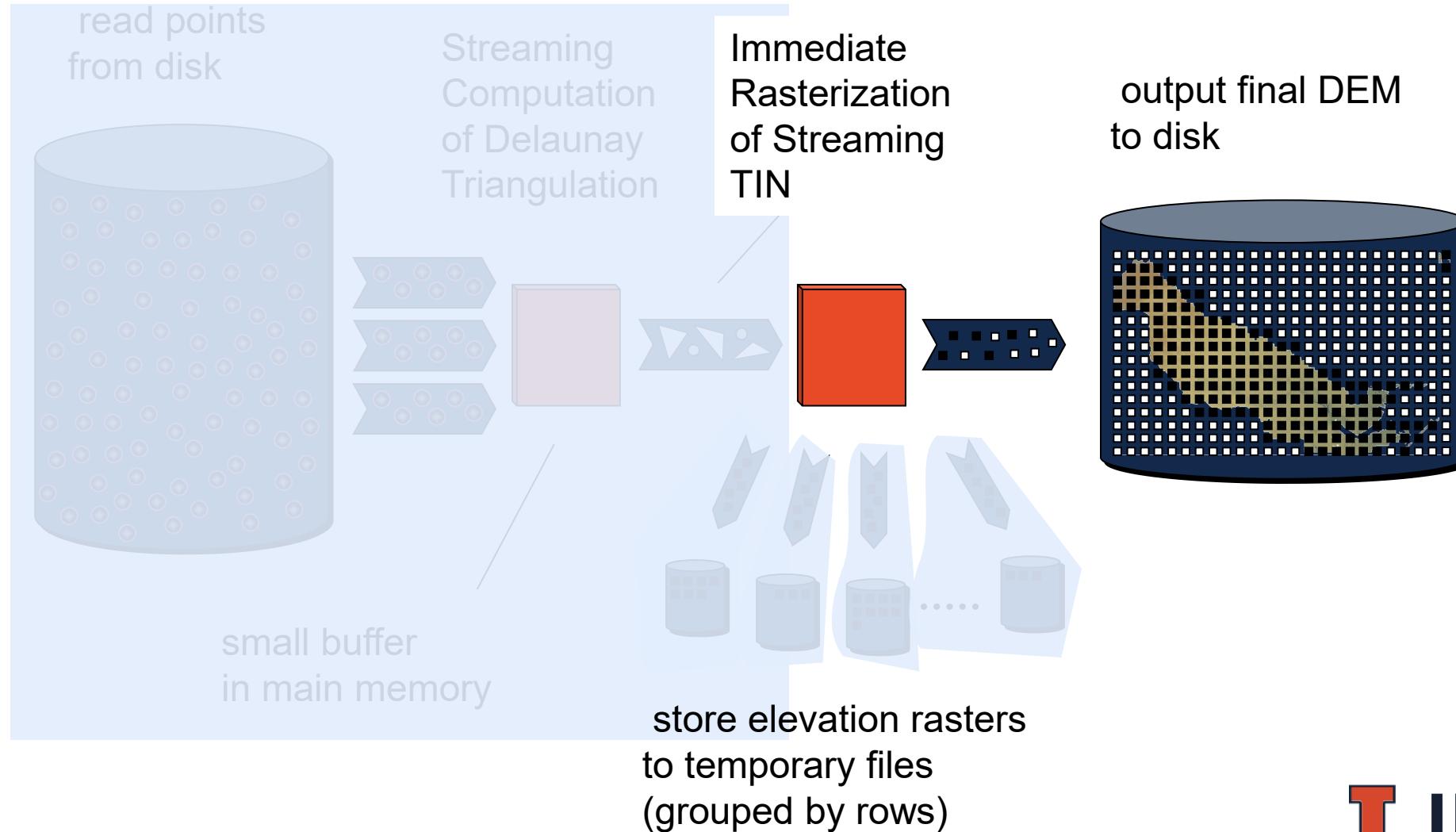


$50,394 \times 30,500$ DEM
3 GB
(binary, BIL, 16 bit, 20 ft)



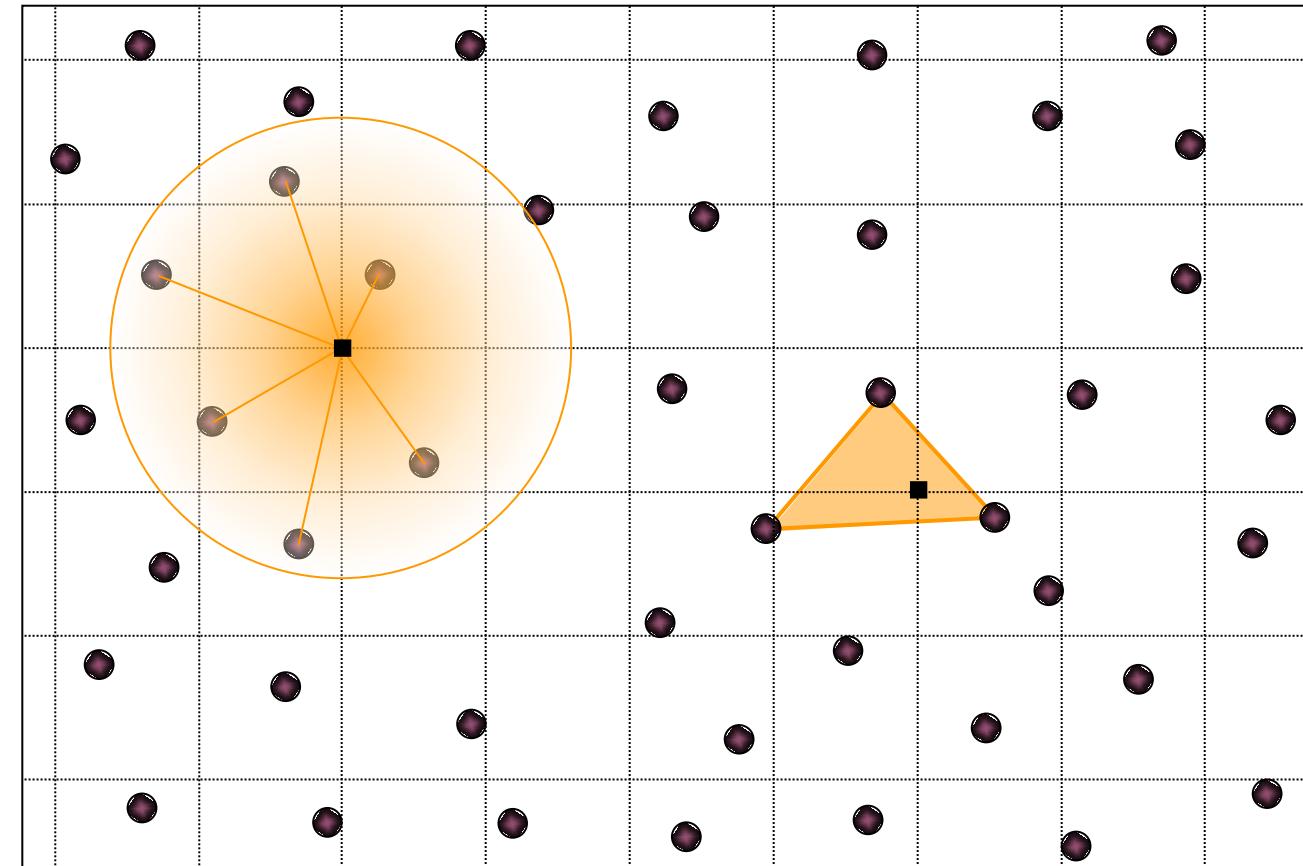
on a household laptop with two harddisks
in 67 minutes
64 MB of main memory

DEM Generation via TIN Streaming



TIN to DEM: Interpolation Methods

- Shepard
- RBFs
- Kriging
- TIN
 - linear
 - higher order

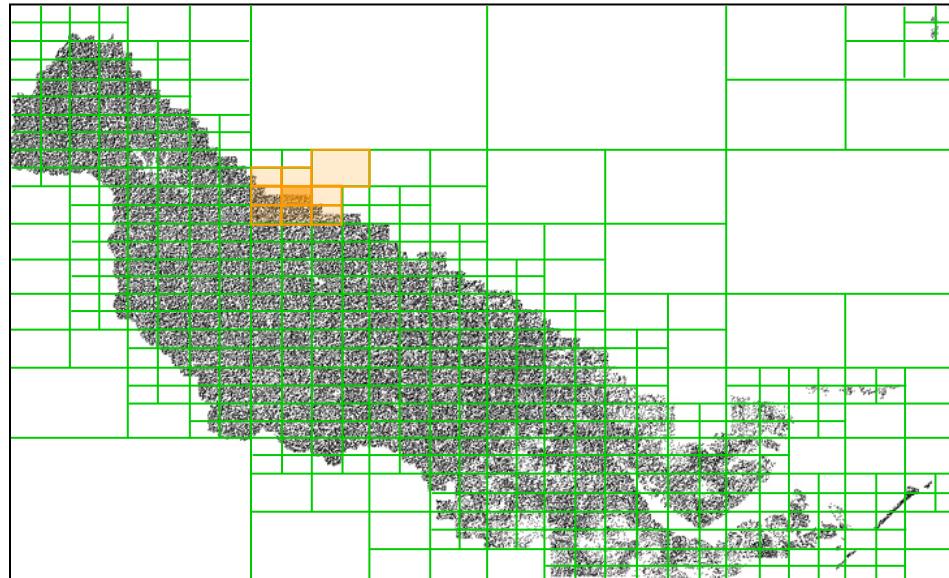


Previous Work: External Memory Quadtree

From Point Cloud to Grid DEM: A Scalable Approach
[Agarwal et al. '06]

- rearrange points on disk into quadtree

- process cell
 - find its neighbors
 - interpolate all (i,j)
 - write rasters to temporary file



- sort temporary file on (i,j) & write DEM

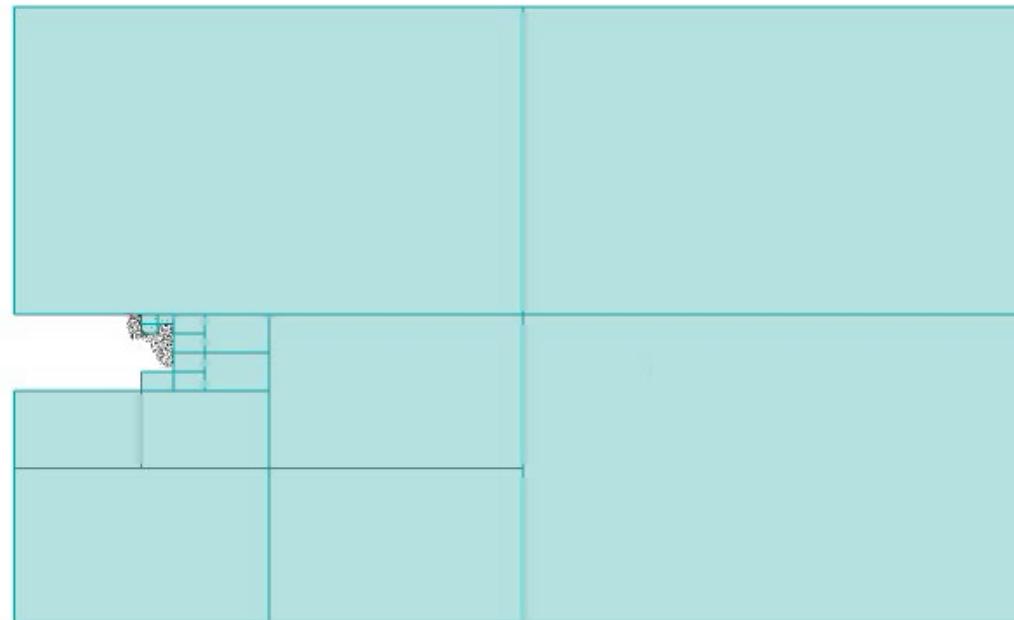


Our Approach: TIN Streaming

- reorganize computations ... not points
 - compute TIN in places where all points have already arrived
 - raster, output & deallocate
 - keep parts that miss neighbors
- enhance points with spatial finalization

Our Approach: TIN Streaming

- reorganize computations ... not points

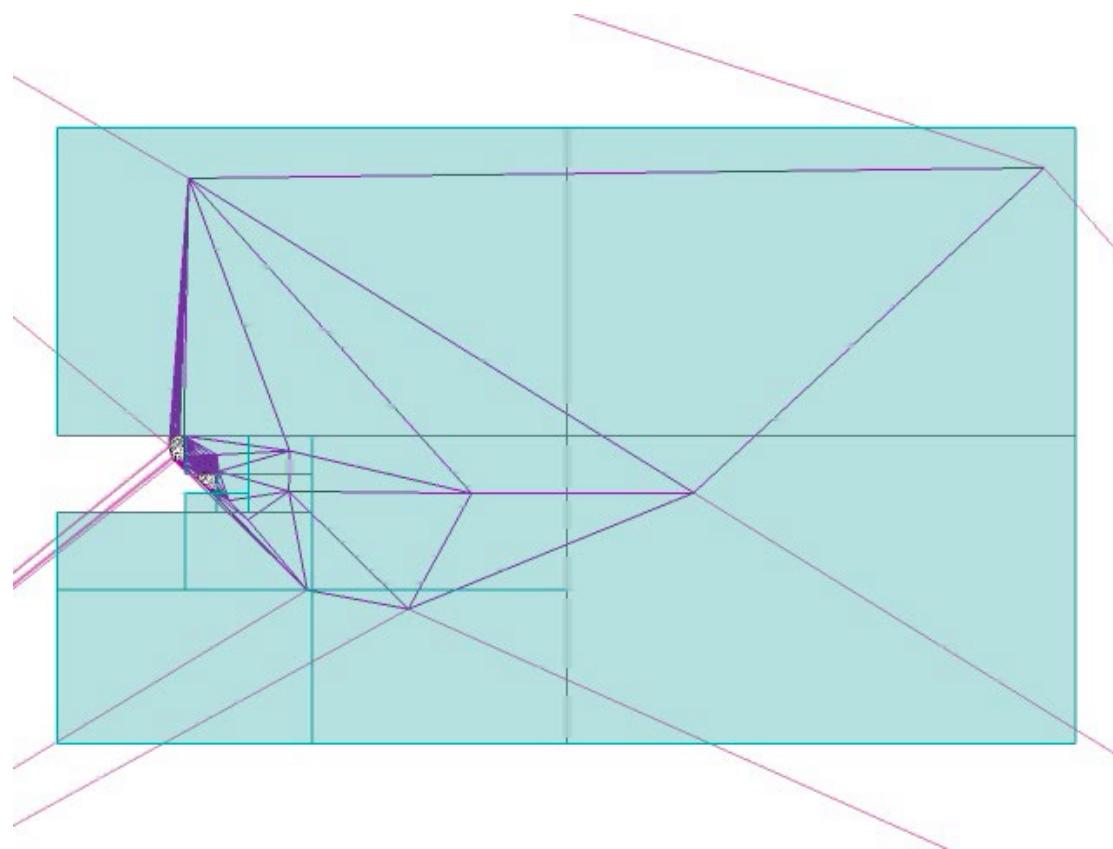


- compute TIN in places where all points have already arrived
- raster, output & deallocate
- keep parts that miss neighbors

- enhance points with spatial finalization

Our Approach: TIN Streaming

- reorganize computations ... not points

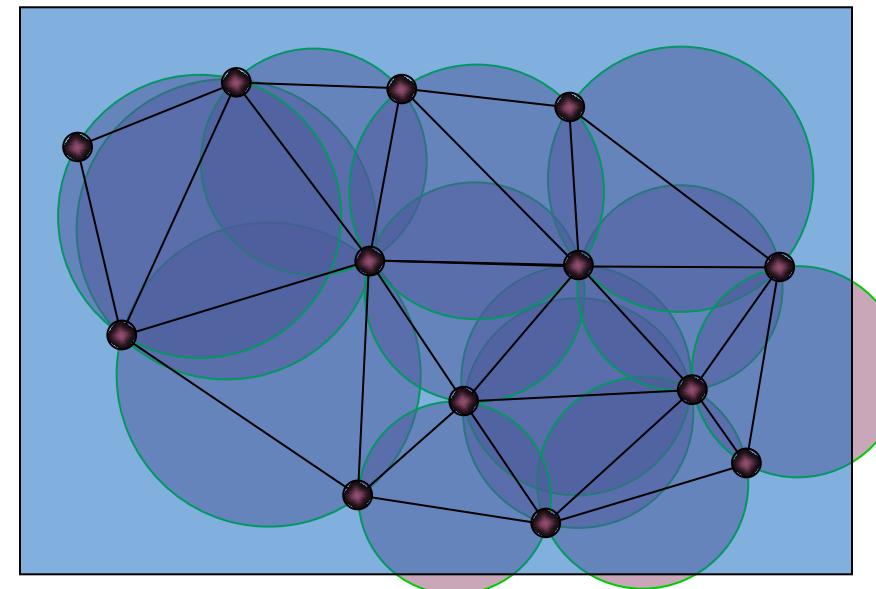


- compute TIN in places where all points have already arrived
- raster, output & deallocate
- keep parts that miss neighbors

- enhance points with spatial finalization

Delaunay Triangulation

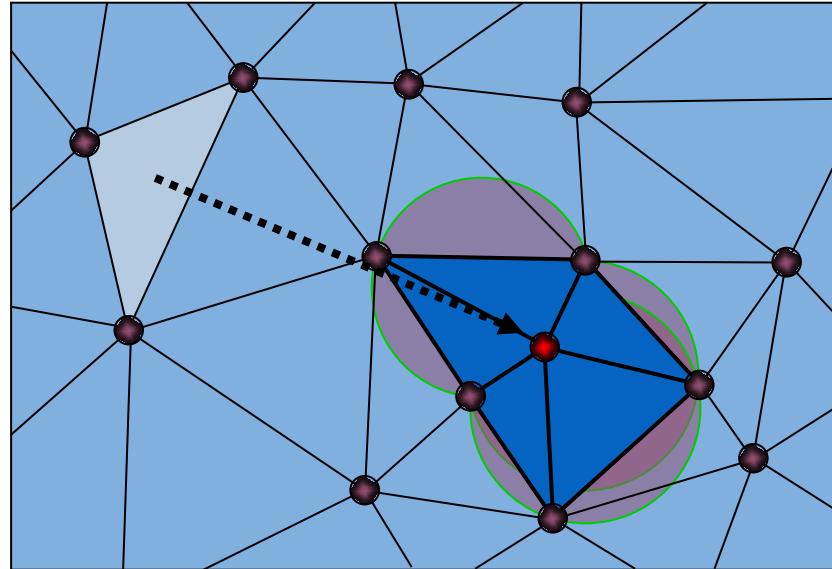
- standard algorithm for TIN construction
- good properties for interpolating points



- a triangulation in which every triangle has an empty circumscribing circle



Incremental Point Insertion



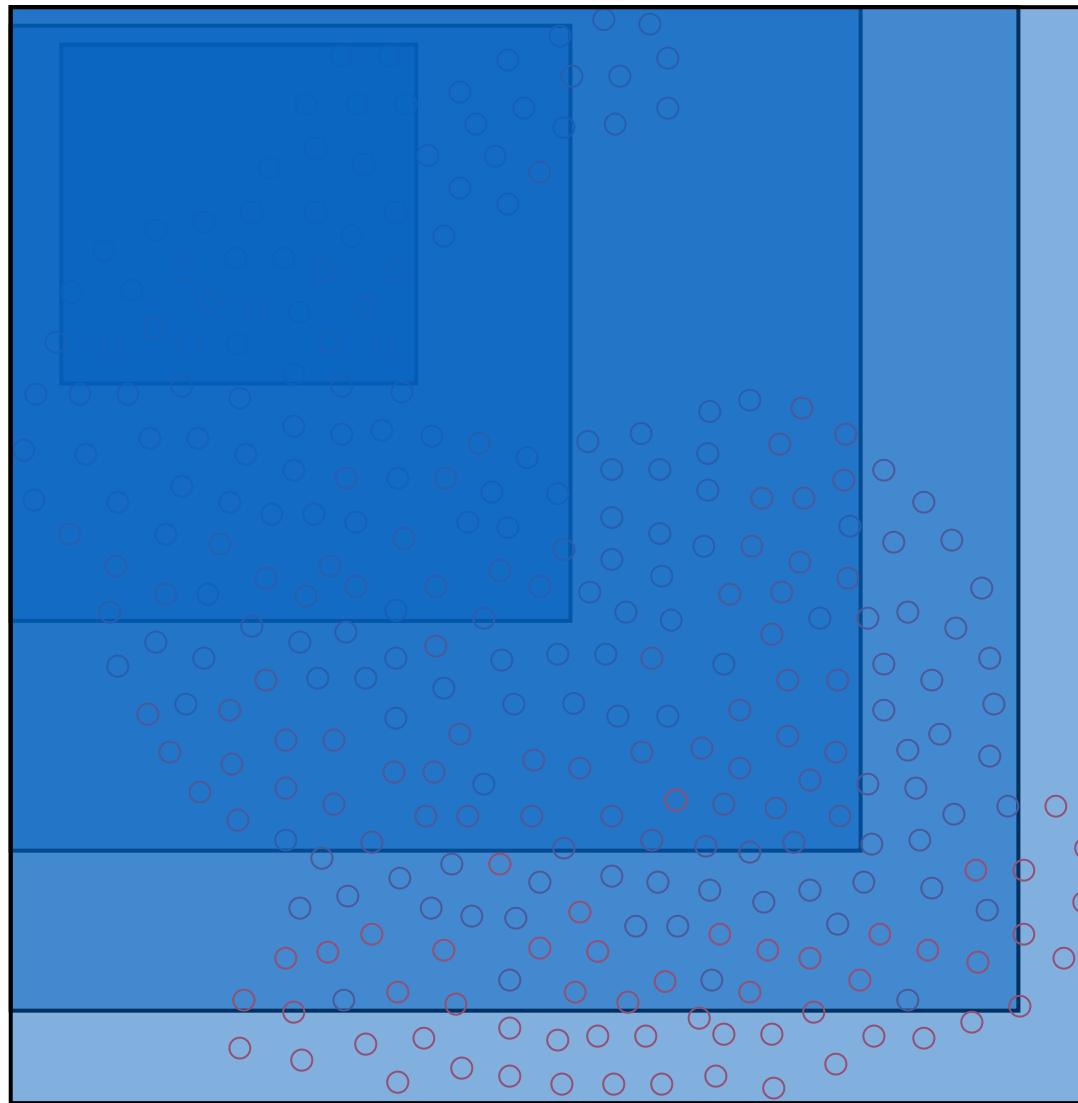
[Lawson '77]
[Bowyer '81]
[Watson '81]

- locate triangle enclosing the point
- find and remove all triangles with non-empty circumcircles
- triangulate by connecting new point

The Finalizer

(spfinalize)

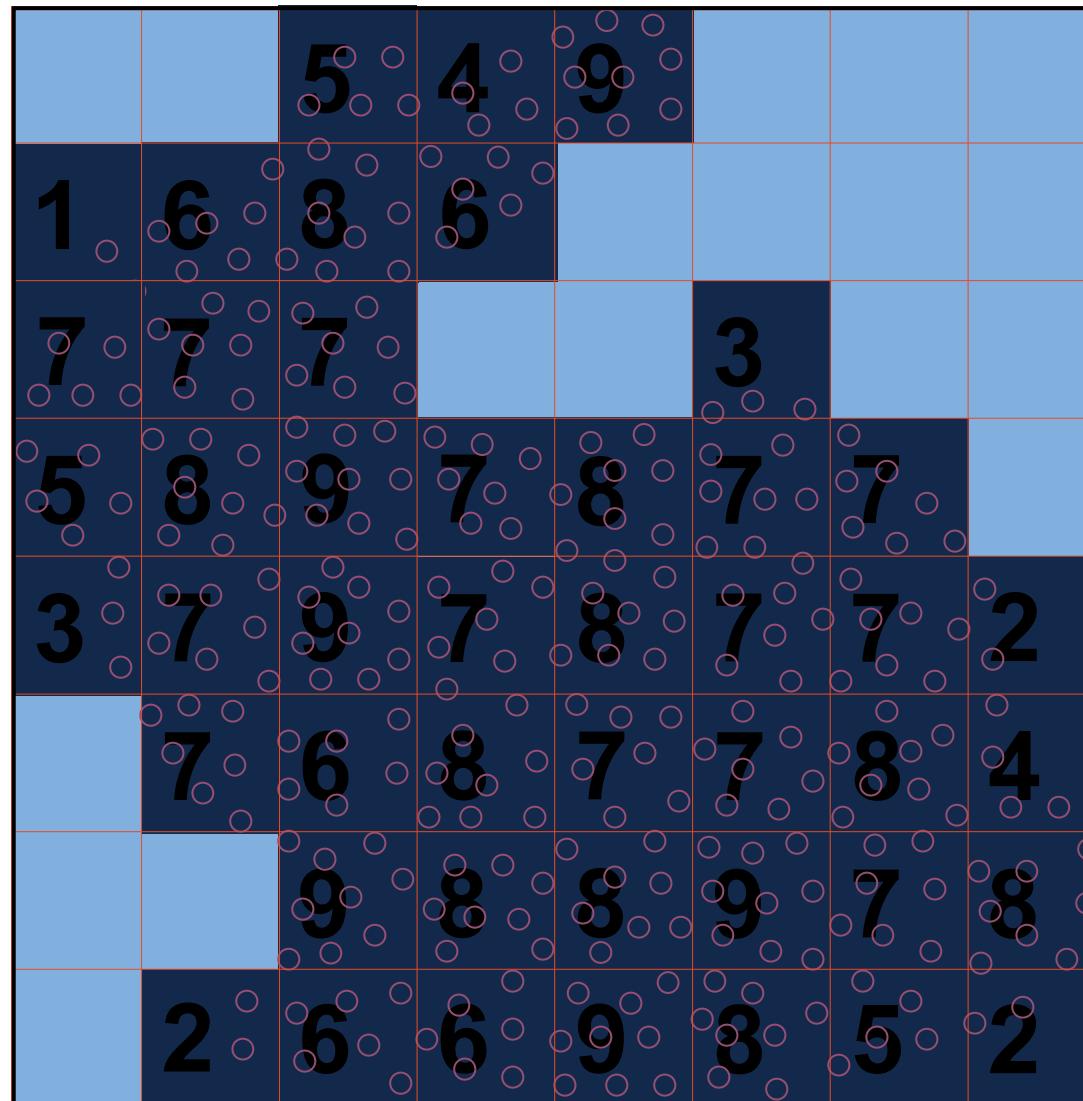
Spatial Finalization of Points



- ➊ compute bounding box



Spatial Finalization of Points



- ① compute bounding box
- ② create finalization grid
 - count number of points per cell



Spatial Finalization of Points

		5	4	9			
1	6	8	6				
7	7	7			3		
5	8	9	7	8	7	7	
3	7	9	7	8	7	7	2
	7	6	8	7	7	8	4
		9	8	8	9	7	8
2	6	6	9	8	5		2

- ① compute bounding box
- ② create finalization grid
 - count number of points per cell
- ③ output finalized points
 - buffer per grid cell
 - if full, output points in one randomized chunk followed by finalization tag



Spatial Finalization of Points



- ① compute bounding box
- ② create finalization grid
 - count number of points per cell
- ③ output finalized points
 - buffer per grid cell
 - if full, output points in one randomized chunk followed by finalization tag



Spatial Finalization of Points



- ① compute bounding box
- ② create finalization grid
 - count number of points per cell
- ③ output finalized points
 - buffer per grid cell
 - if full, output points in one randomized chunk followed by finalization tag



Spatial Finalization of Points



- ① compute bounding box
- ② create finalization grid
 - count number of points per cell
- ③ output finalized points
 - buffer per grid cell
 - if full, output points in one randomized chunk followed by finalization tag



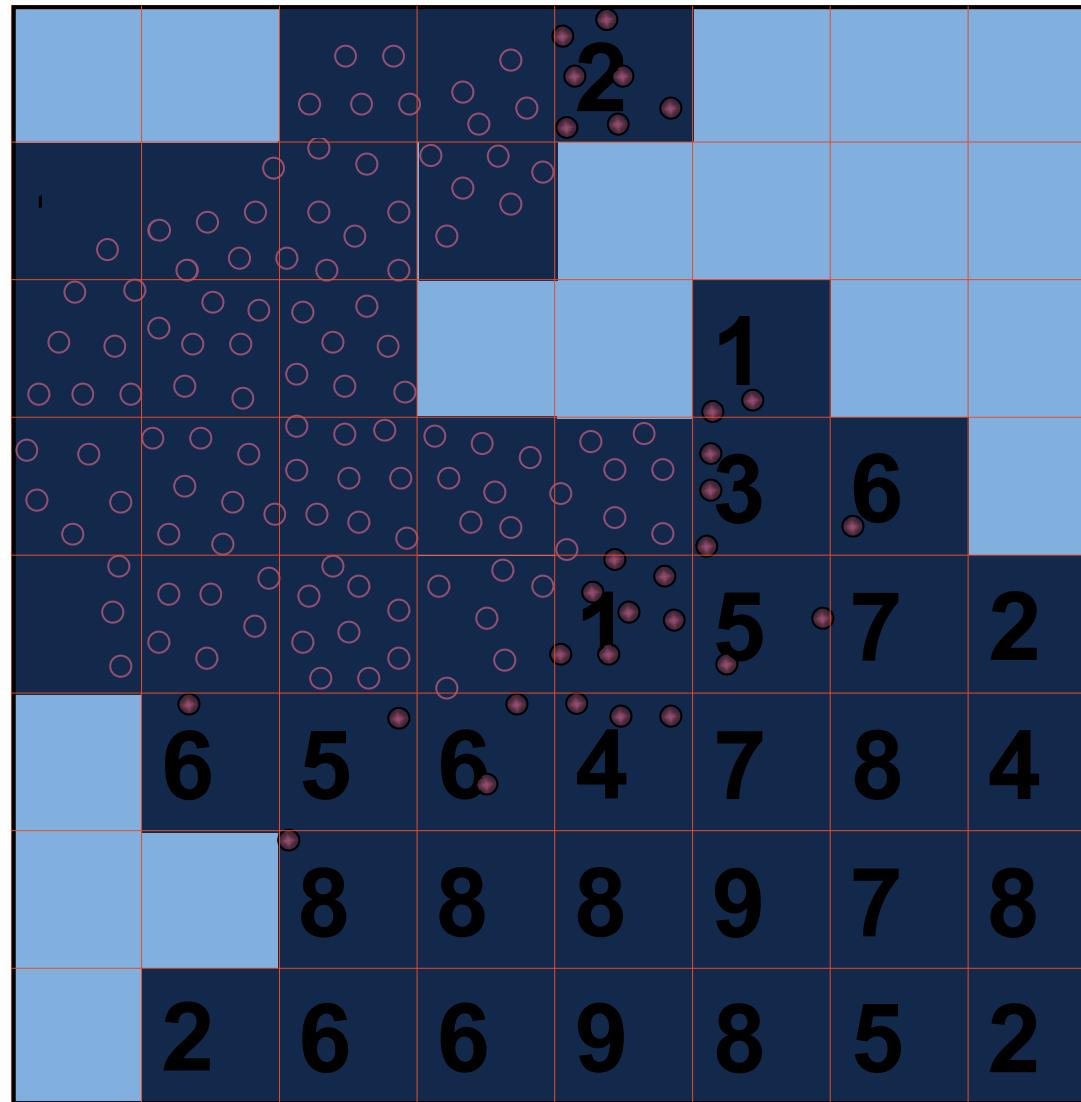
Spatial Finalization of Points



- ① compute bounding box
- ② create finalization grid
 - count number of points per cell
- ③ output finalized points
 - buffer per grid cell
 - if full, output points in one randomized chunk followed by finalization tag



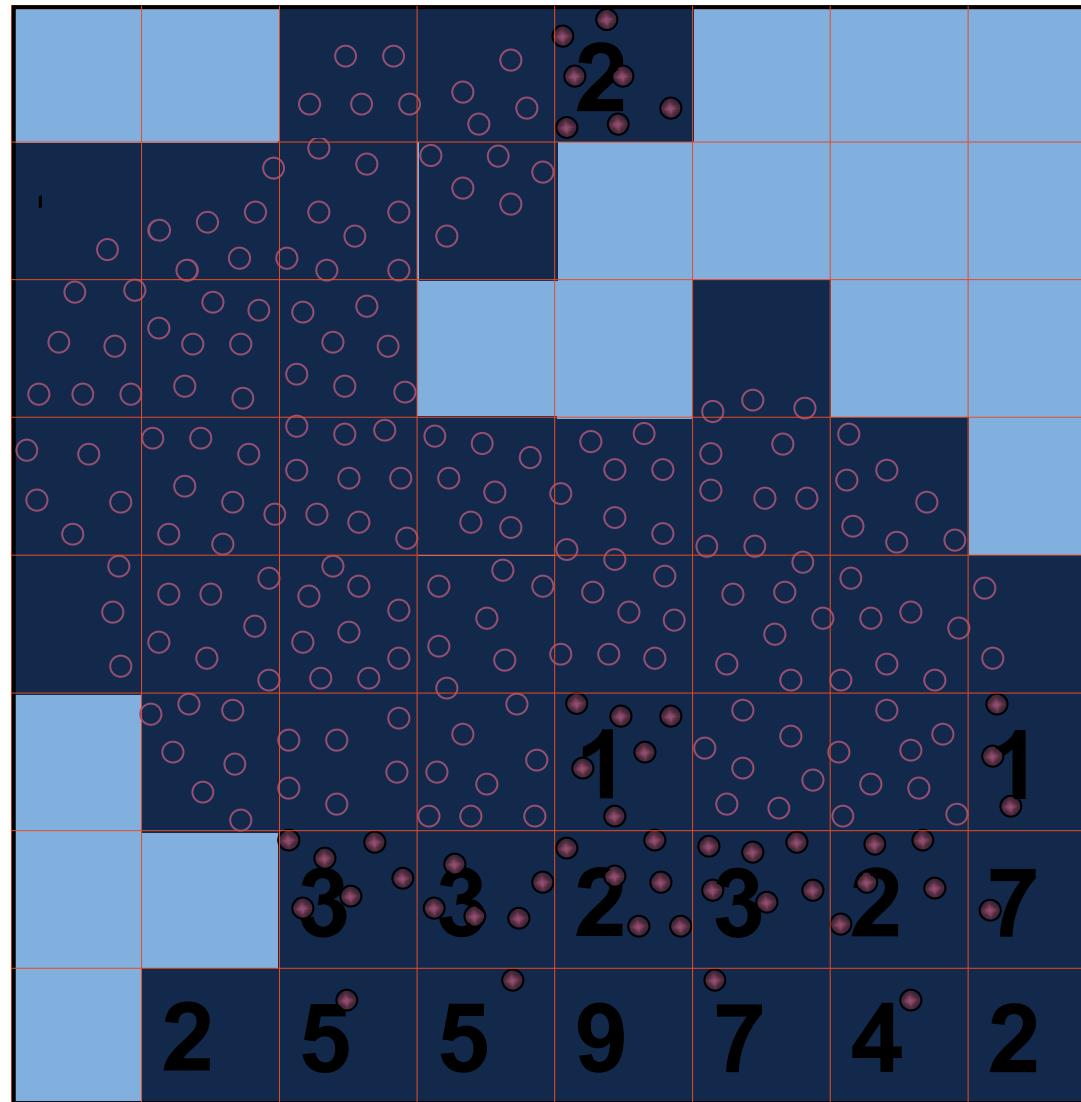
Spatial Finalization of Points



- ① compute bounding box
- ② create finalization grid
 - count number of points per cell
- ③ output finalized points
 - buffer per grid cell
 - if full, output points in one randomized chunk followed by finalization tag



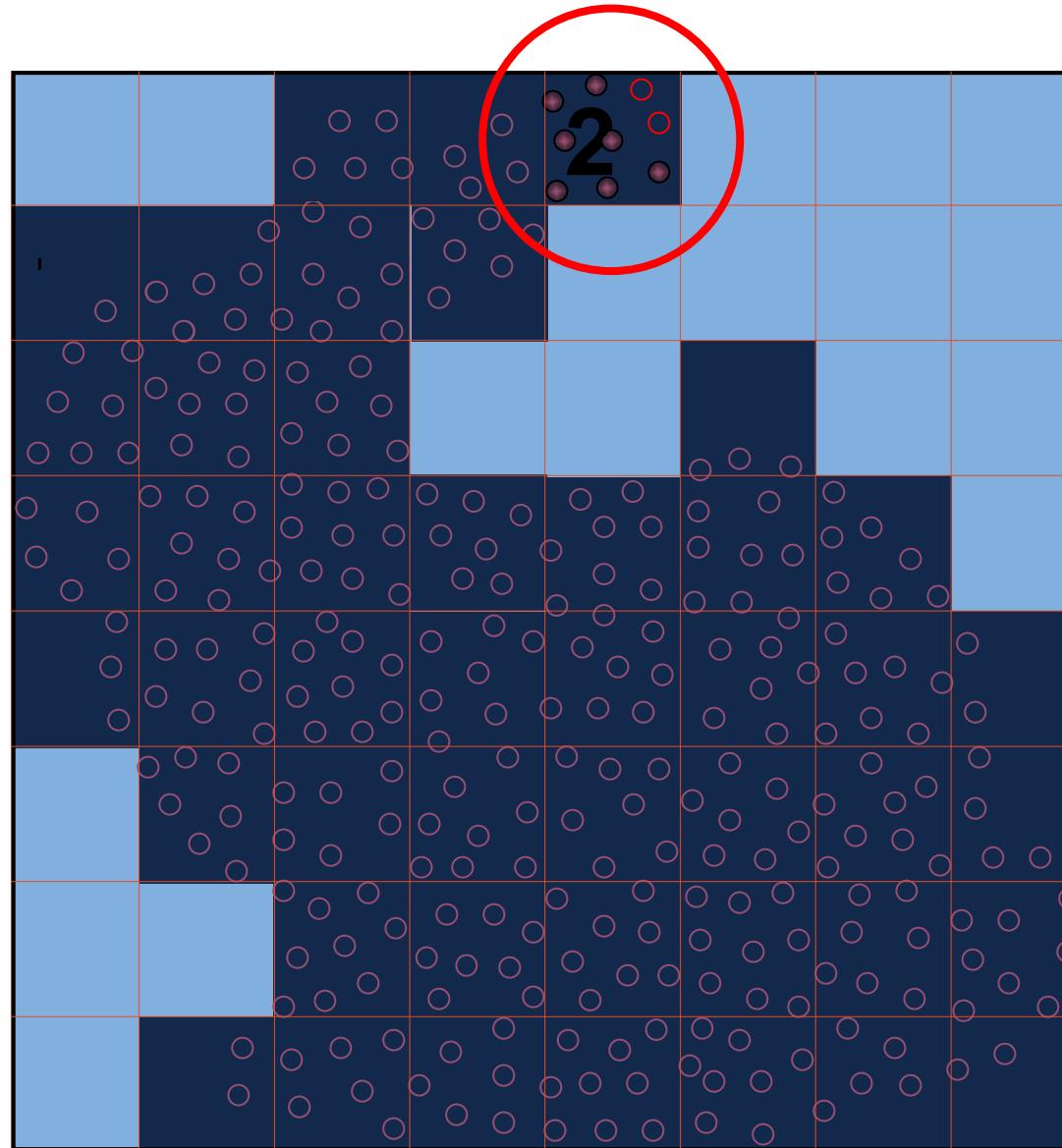
Spatial Finalization of Points



- ① compute bounding box
- ② create finalization grid
 - count number of points per cell
- ③ output finalized points
 - buffer per grid cell
 - if full, output points in one randomized chunk followed by finalization tag



Spatial Finalization of Points



- ① compute bounding box
- ② create finalization grid
 - count number of points per cell
- ③ output finalized points
 - buffer per grid cell
 - if full, output points in one randomized chunk followed by finalization tag



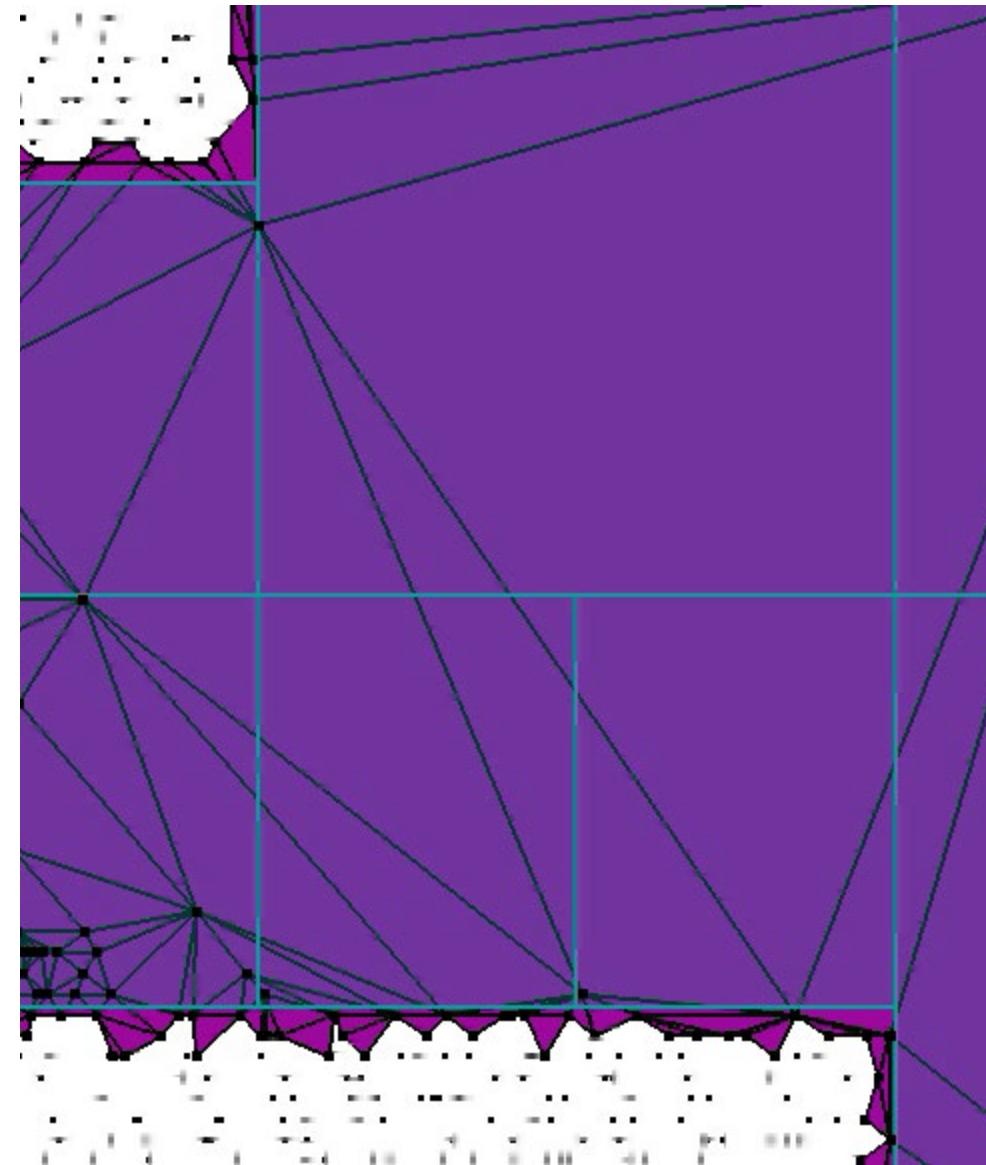
The Triangulator

(sp delaunay)



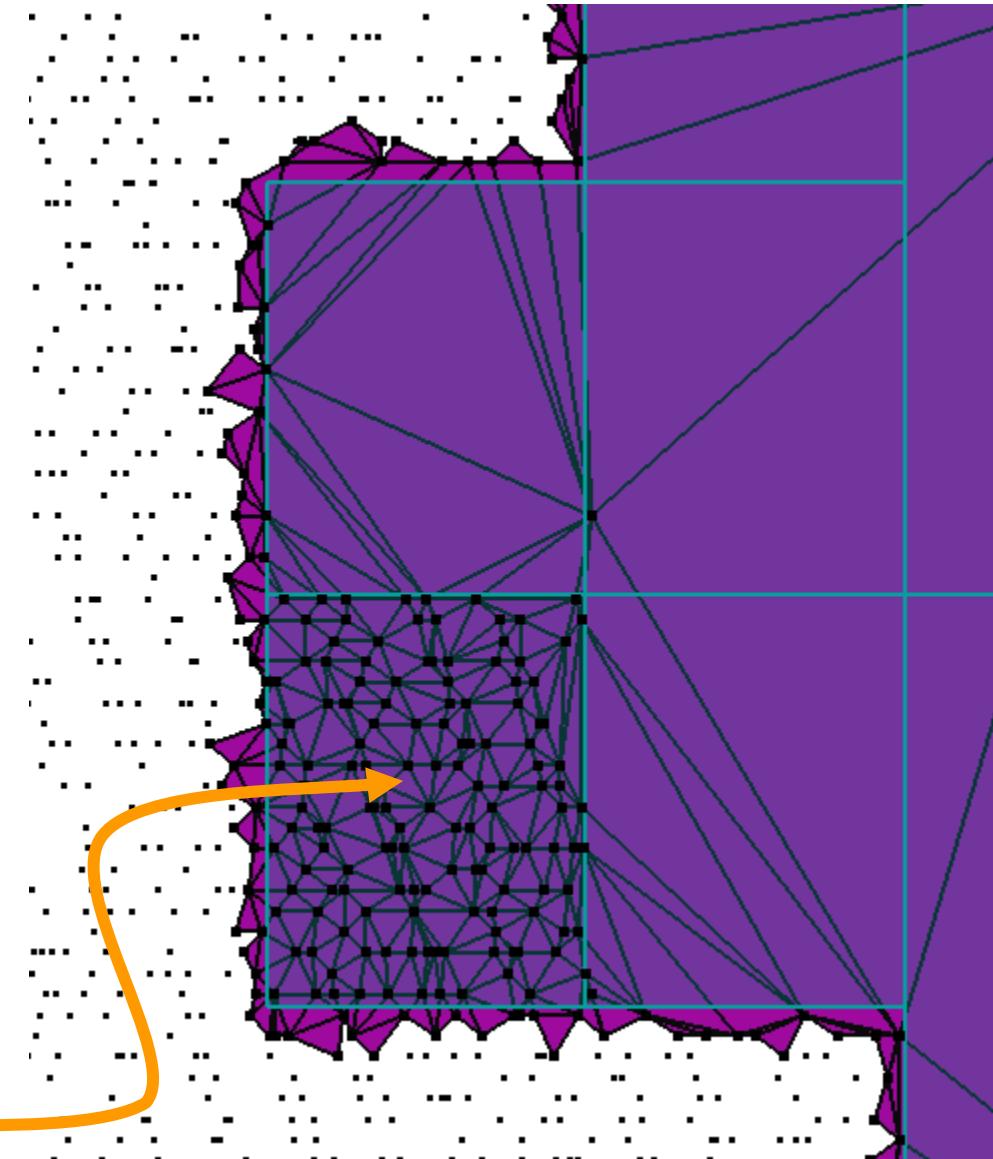
Modified Incremental Delaunay

- insert points
- when reading a finalization tag
 - find certified triangles
 - output them to disk or pipe
 - deallocate data structure



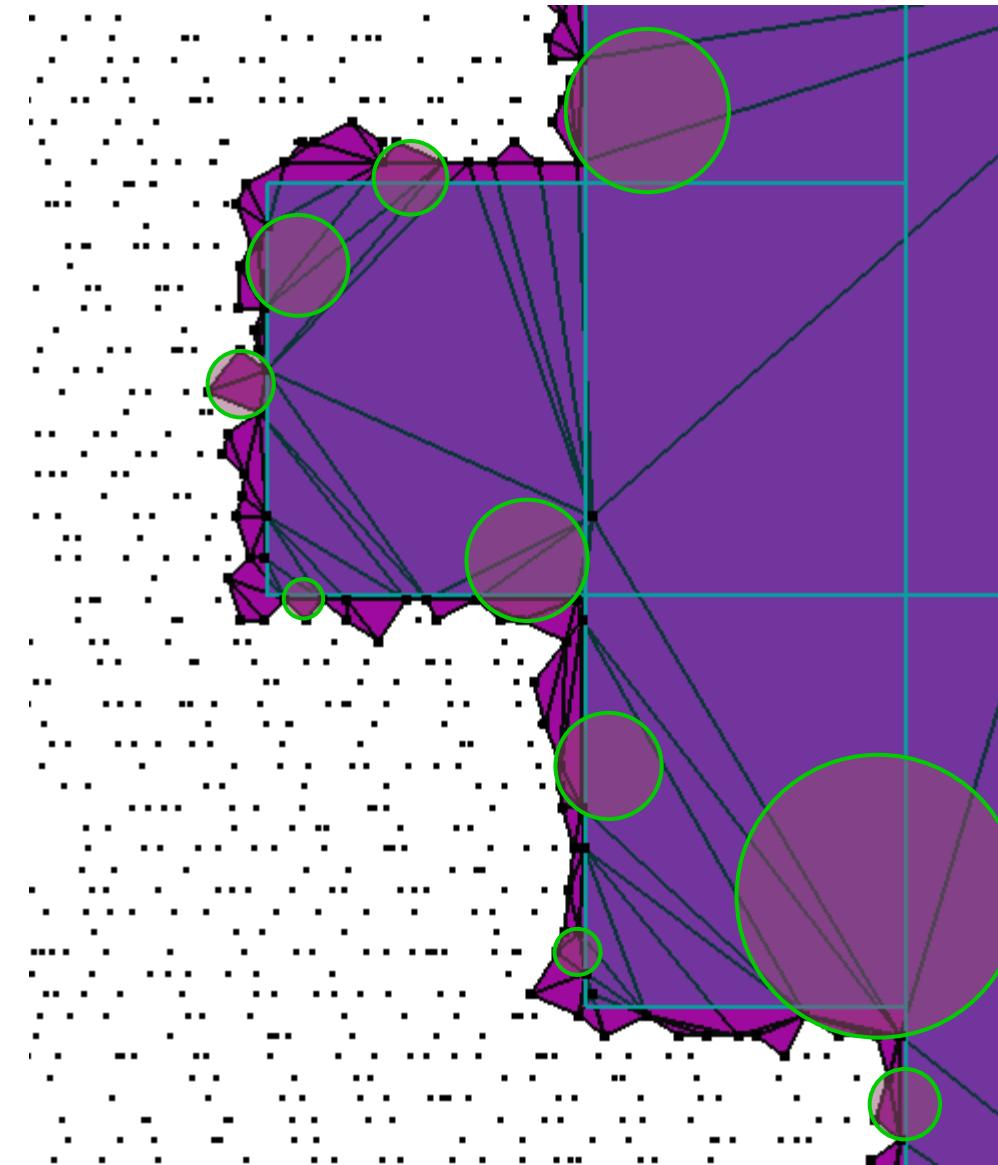
Modified Incremental Delaunay

- insert points
- when reading a finalization tag
 - find certified triangles
 - output them to disk or pipe
 - deallocate data structure



Modified Incremental Delaunay

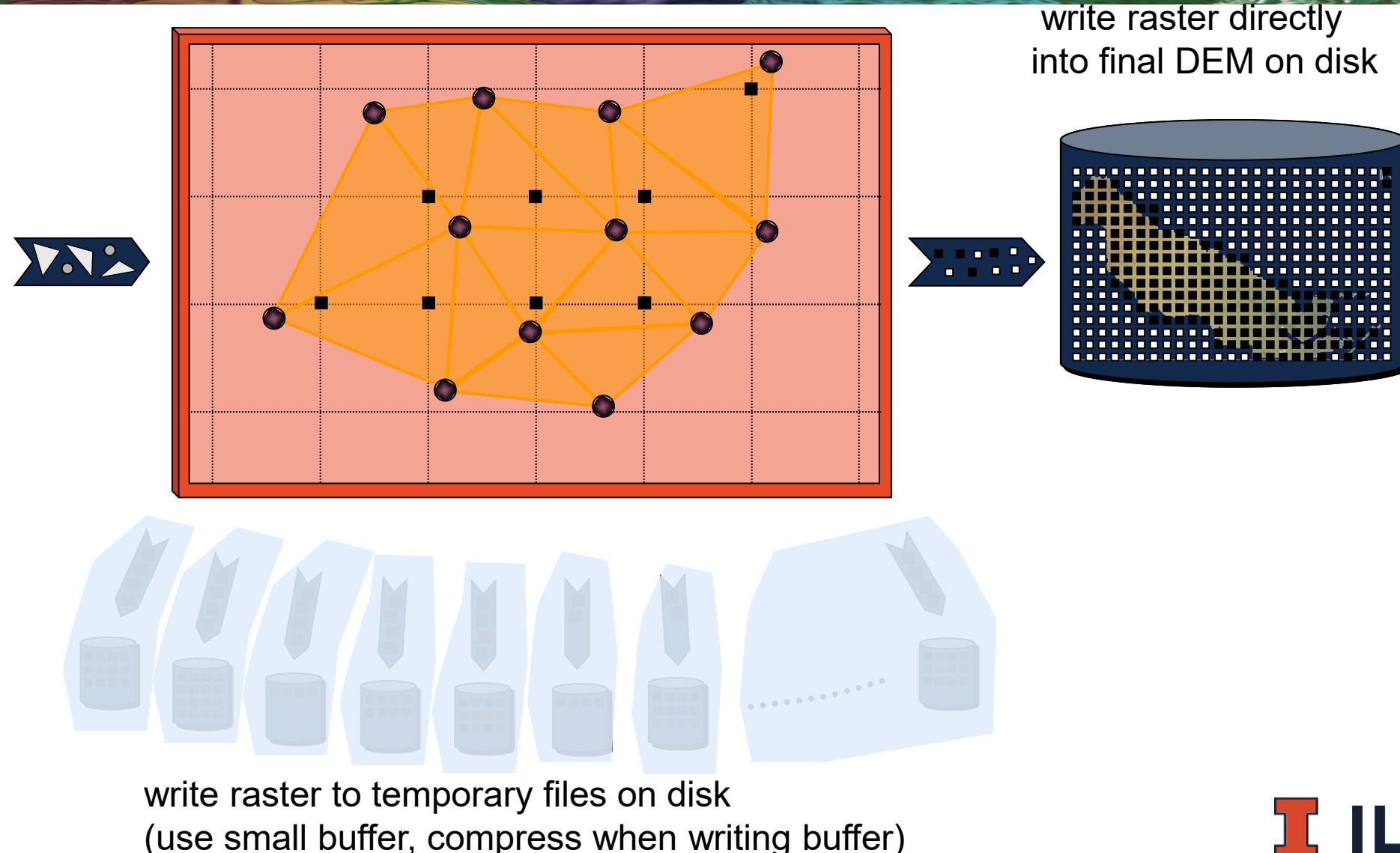
- insert points
- when reading a finalization tag
 - find certified triangles
 - output them to disk or pipe
 - deallocate data structure



The Rasterizer

(tin2dem)

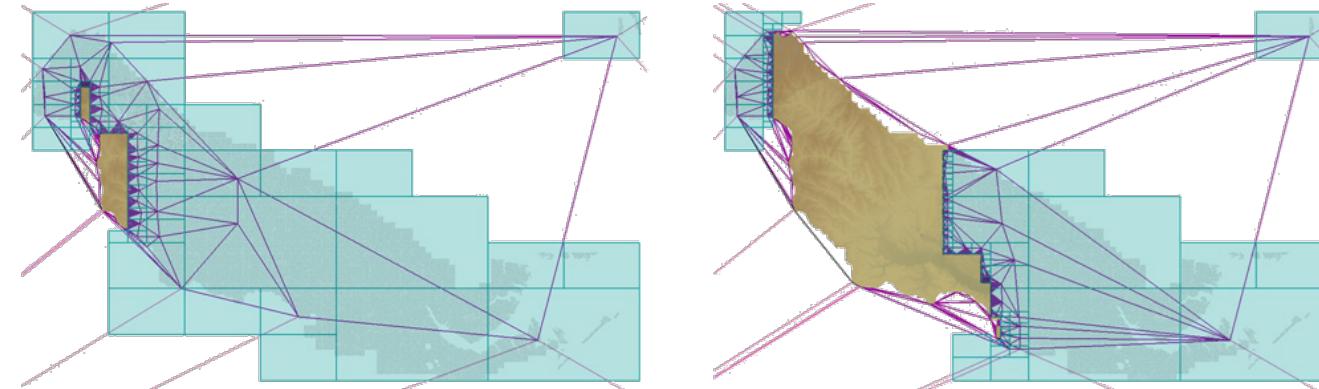
Rasterizing the Streaming TIN





Results

Neuse- River Basin

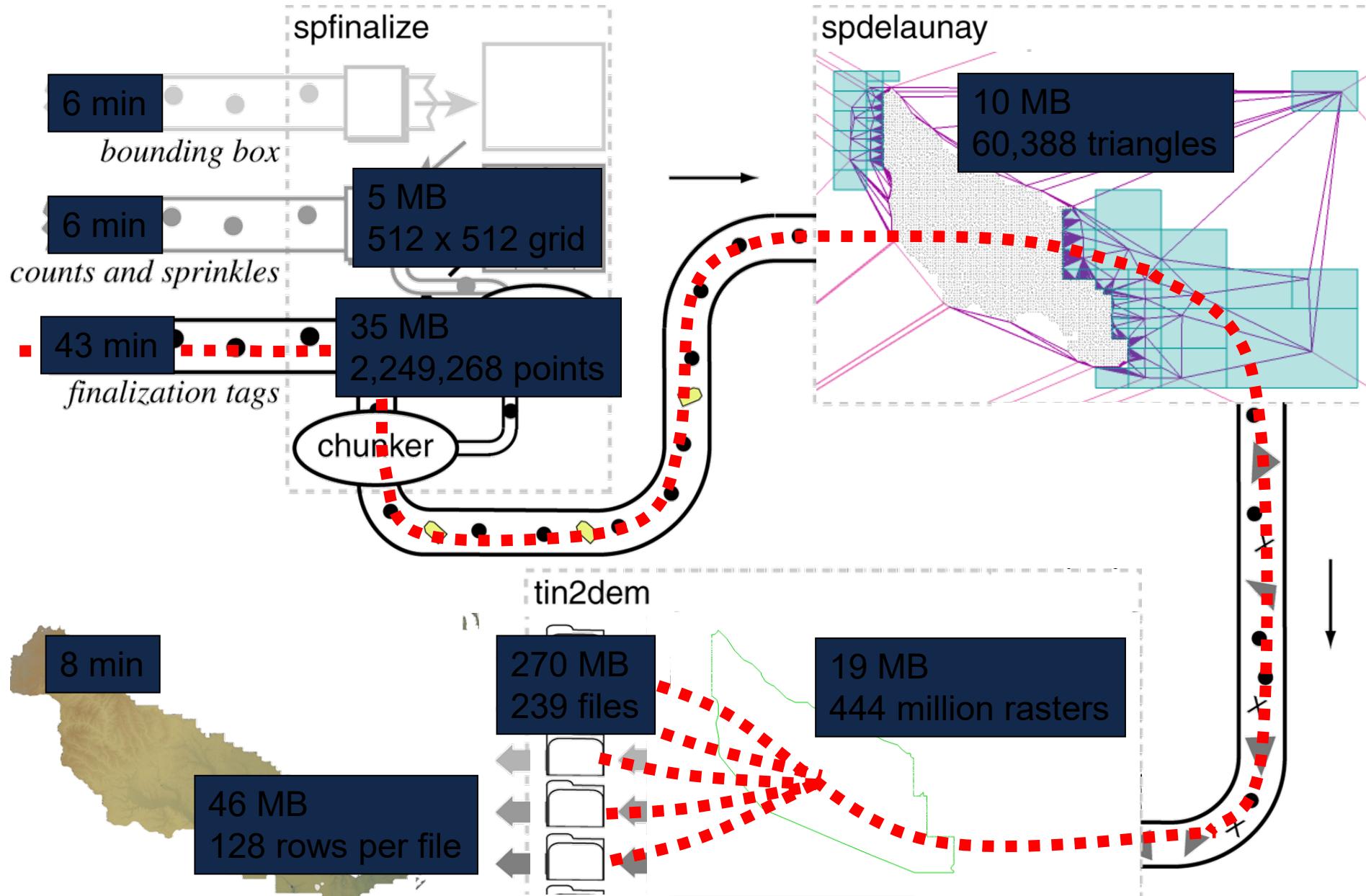


LIDAR	number of points, file size	500,141,313	11.2 GB
	[min _x ; max _x]	[1,930,000; 2,937,860]	
input mass points	[min _y ; max _y]	[390,000; 1,000,000]	
	[min _z ; max _z]	[-129.792; +886.443]	

DEM	grid spacing	40 ft	20 ft	10 ft
	cols	25,198	50,394	100,788
	rows	15,250	30,500	61,000
output grid	output file size	750 MB	3.0 GB	12.0 GB
	Time (hours:minutes)	total	0:53	1:07
	Memory (MB)	maximum	55	64
Scratch Files (MB)	total size	82	270	868



Pipeline Details for the 20 ft DEM





Comparison to

[Agarwal et al. '06]

(20 ft)

- 53 hours
 - 3.4 GHz desktop
 - 10,000 RPM disks
 - scratch space:
 - quadtree: ~ 8 GB
 - rasters: ~ 2.5 GB
- 67 minutes
 - 2.1 GHz laptop
 - 5,400 RPM disks
 - scratch space:
 - compressed rasters: 270 MB
- **86 % of CPU time for expensive RST**
 - remaining 7.5 hours
 - constructing quadtree: 1.1 hours
 - finding cell neighbors: 5.6 hours

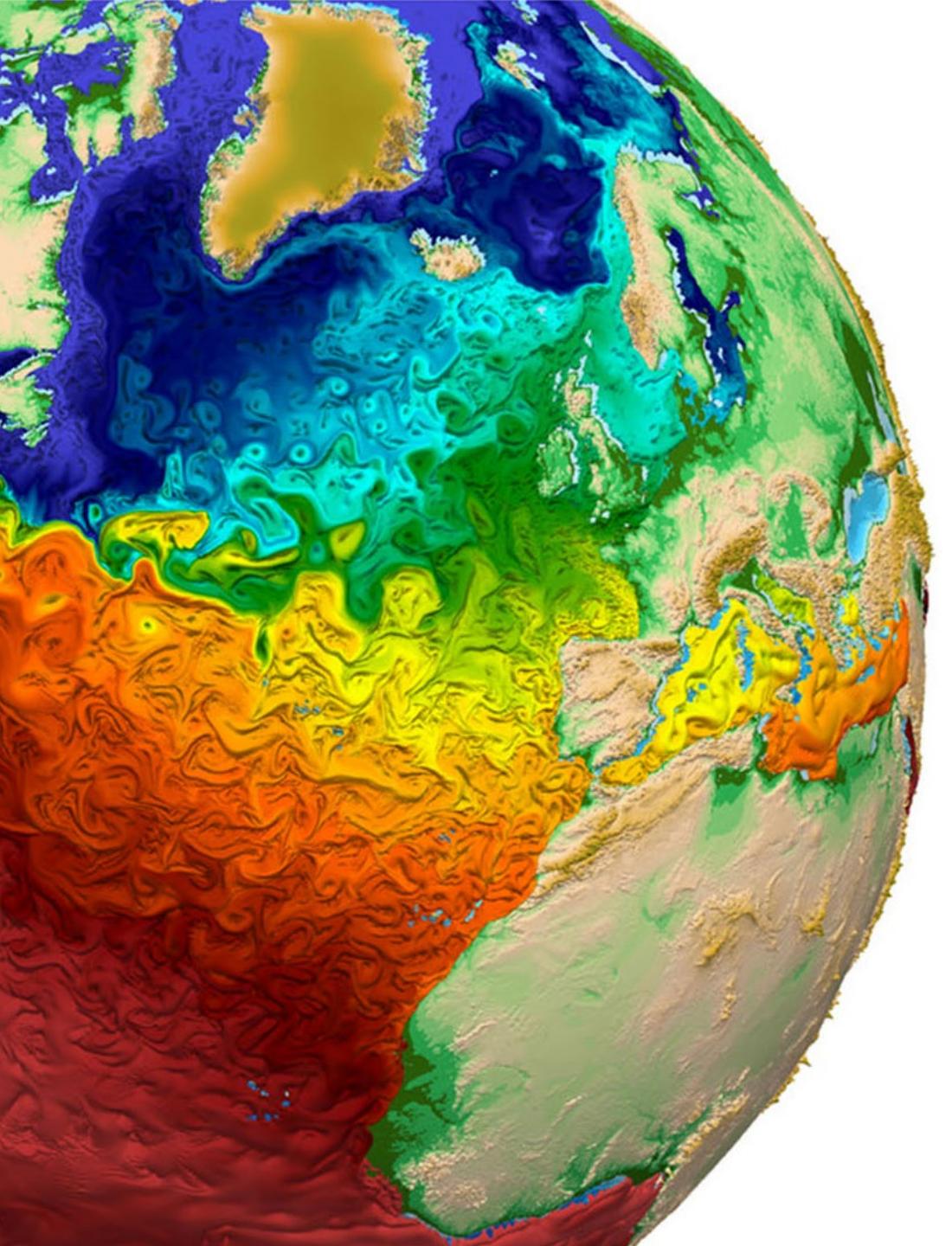


Acknowledgements

- data
 - Kevin Yi, Duke
- support
 - NSF grants 0429901 + 0430065
"Collaborative Research: Fundamentals
and Algorithms for Streaming Meshes."
 - NGA award HM1582-05-2-0003
 - Alfred P. Sloan Research Fellowship

Paper Analysis

- Paper was chosen as a good introduction to tasks related to GIS visualizations
- But if we wanted to discuss it as a research paper:
 - Paper relies on empirical evidence rather than theory
 - Which is not an issue...
 - ...but some reviewers would likely object to drawing general conclusions from 1 data set
 - System pieces together existing ideas...not innovative in that sense
 - Streaming Delaunay was published by authors previously...more ground-breaking
 - For some research area cultures this is a fine approach, less so for others
 - Performance numbers are excellent
 - Addresses a real problem...large data and people working with limited resources
 - Excellent, pragmatic work IMO

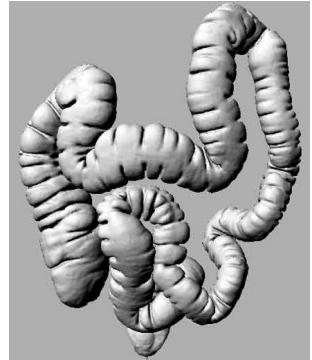


Contouring

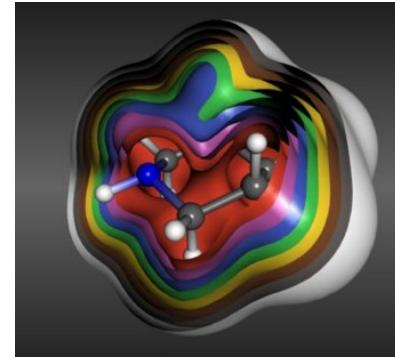
Marching Cubes

Scientific Visualization
Professor Eric Shaffer

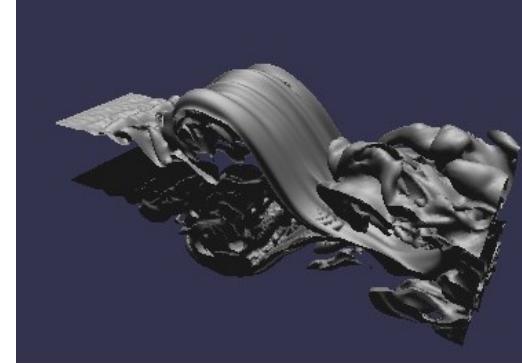
Contouring in 3D



colon (CT dataset)

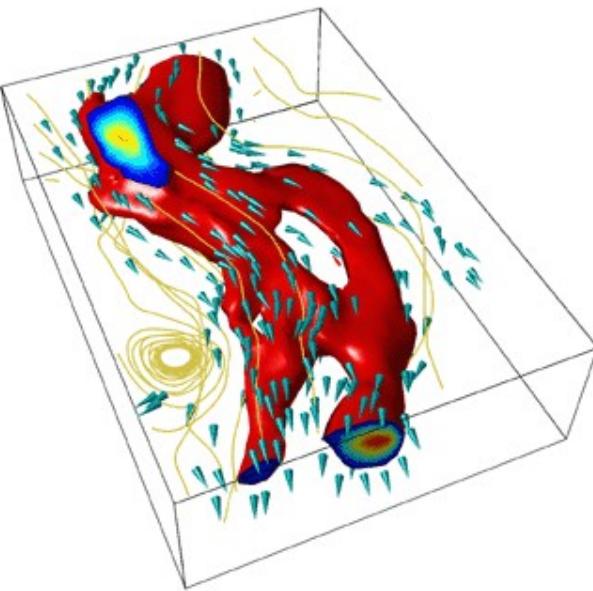


electron density in molecule

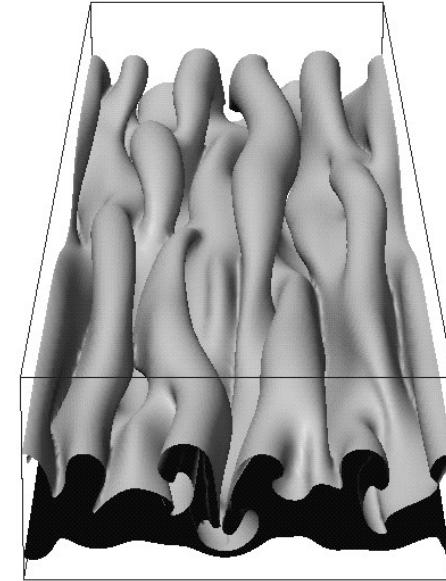


velocity in 3D fluid flow

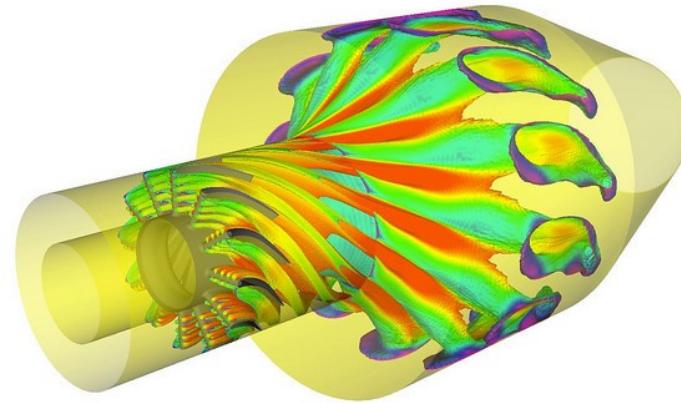
Scalar fields in 3D space are volumes



velocity in 3D fluid flow



magnetic field in sunspots

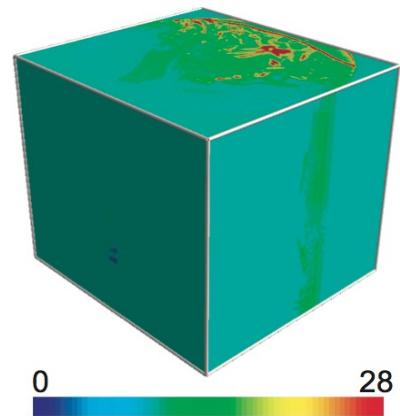


fuel concentration, colored
by temperature in jet engine

Contouring a volume → isosurface

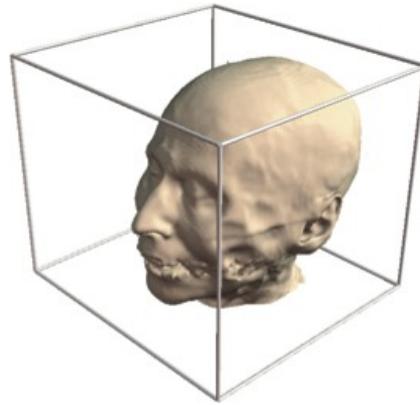
Many uses...
e.g. finding boundary between materials

Isosurfaces

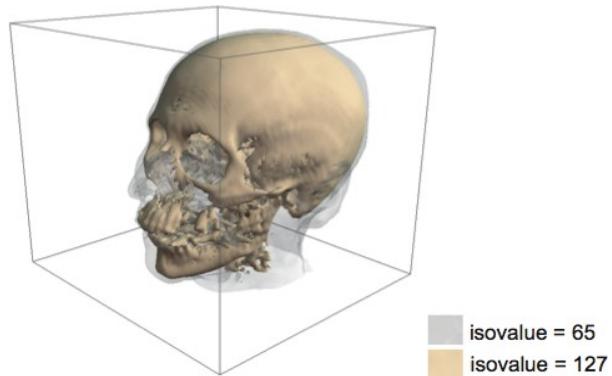


Isosurfaces

scalar CT volume (tissue density)



isosurface for scalar value corresponding to skin



isosurfaces for skin and bone

Marching Cubes

TITLE

Marching cubes: A high resolution 3D surface construction algorithm
WE Lorensen, HE Cline
ACM siggraph computer graphics 21 (4), 163-169

CITED BY

16213

1987

YEAR

Developed by William E. Lorensen and Harvey E. Cline at General Electric Medical.

Designed to efficiently visualize data from CT and MRI devices

Famously patented algorithm (application in 1985) with patent now expired

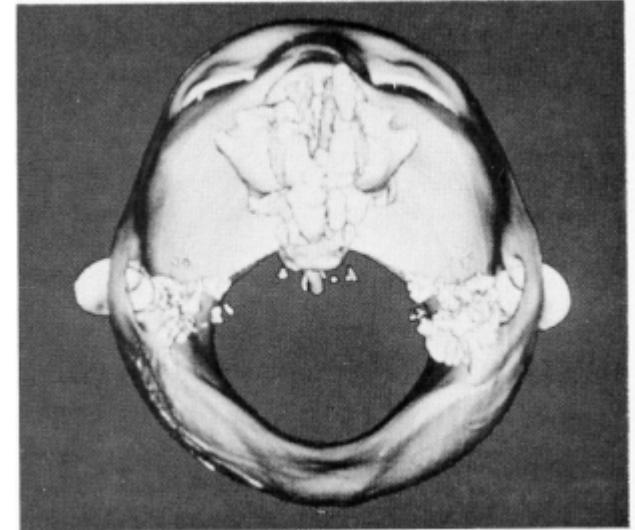
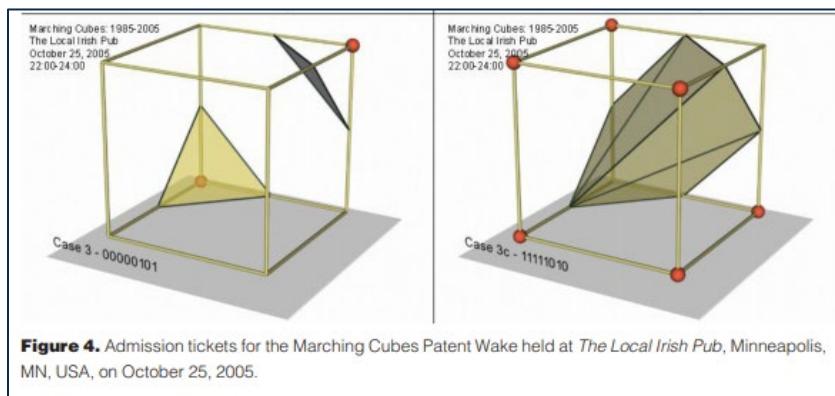


Figure 10. Soft Tissue, Top View.

[Home](#) / [Magazines](#) / [IEEE Computer Graphics and Applications](#) / [2020.02](#)

Remembering Bill Lorensen: The Man, the Myth, and Marching Cubes

March-April 2020, pp. 112-118, vol. 40

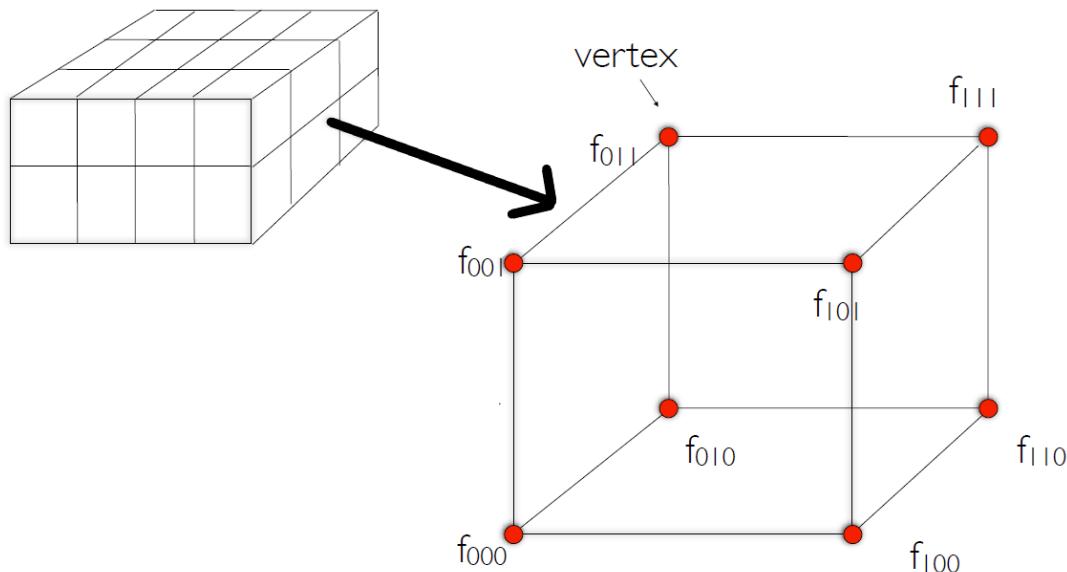
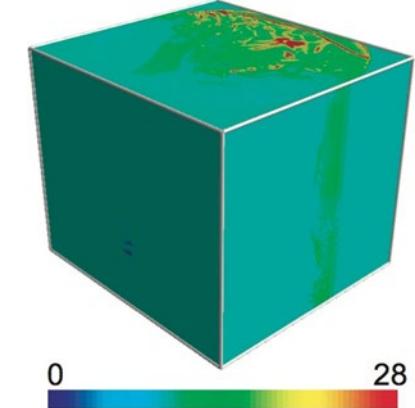
DOI Bookmark: [10.1109/MCG.2020.2971168](https://doi.org/10.1109/MCG.2020.2971168)

Bill Lorensen died on December 12, 2019.

Marching Cubes

Scalar volume: $f : D \subset \mathbb{R}^3 \rightarrow \mathbb{R}$
 $(x, y, z) \mapsto f(x, y, z)$

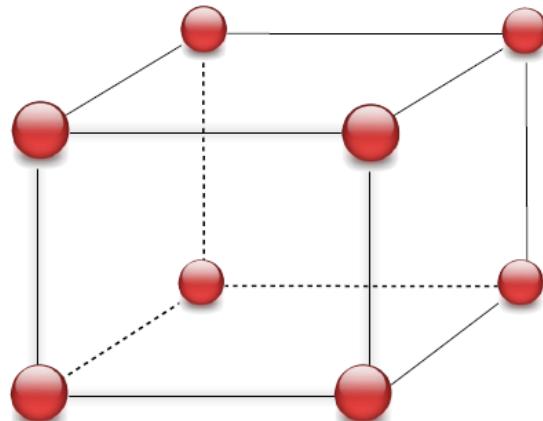
Want to find $S_v = \{(x, y, z) | f(x, y, z) = v\}$



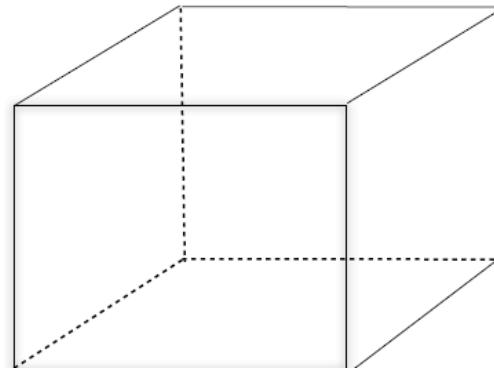
- We seek to construct a polygonal approximation to S_v
- Function is sampled at vertices of a regular cuboid grid
- Generate isosurface cell-by-cell
- Polygons generated across the cells

Labeling Cubes

Label each vertex as greater than or less than the isovalue
For example:



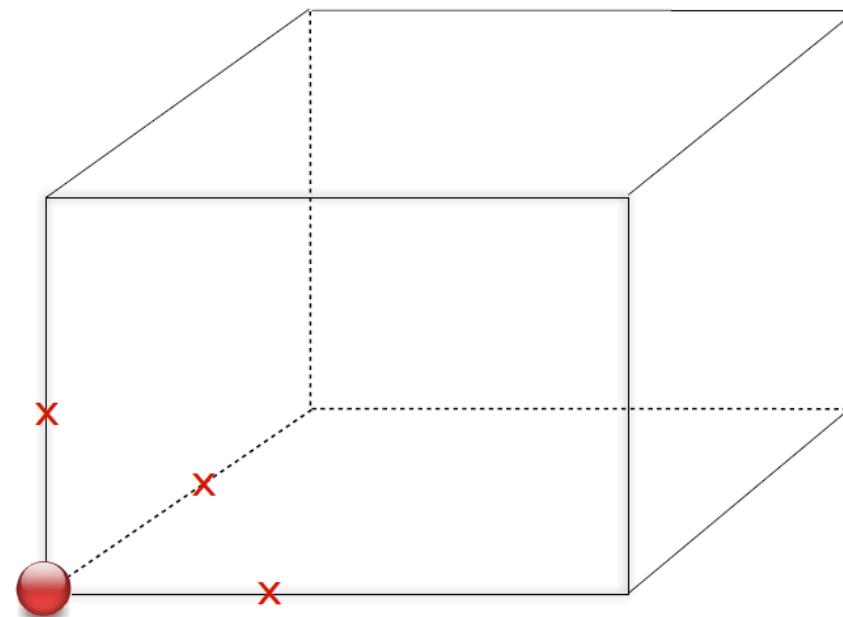
All inside $f > f_0$



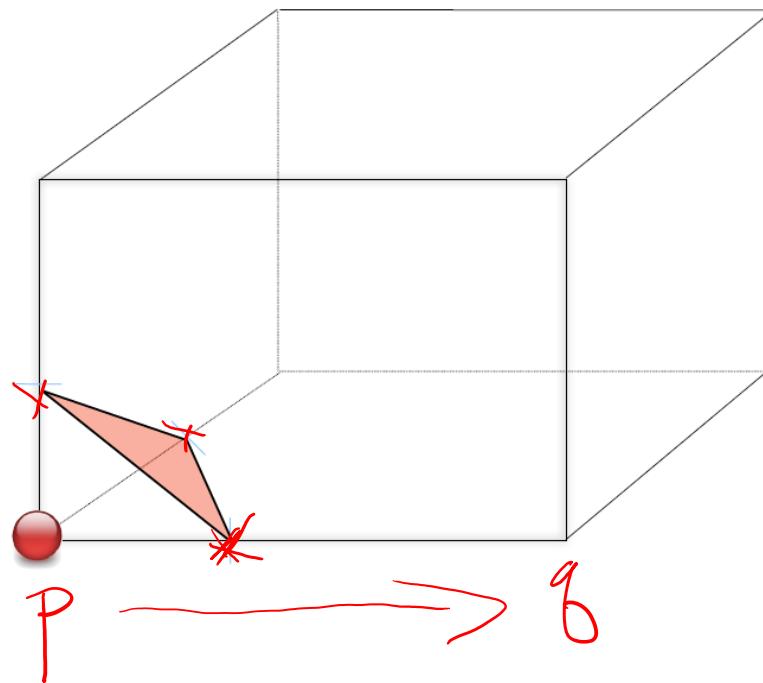
All outside $f \leq f_0$

Bipolar Edges

Edges with two differently classified endpoints are bipolar
The isosurface will cut the edge

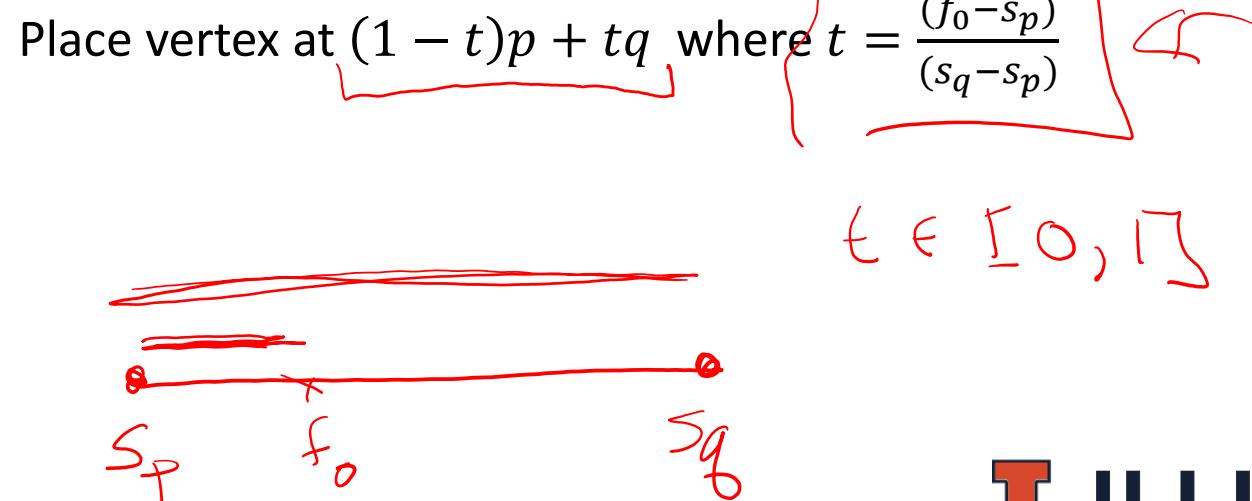


Generating a Polygon

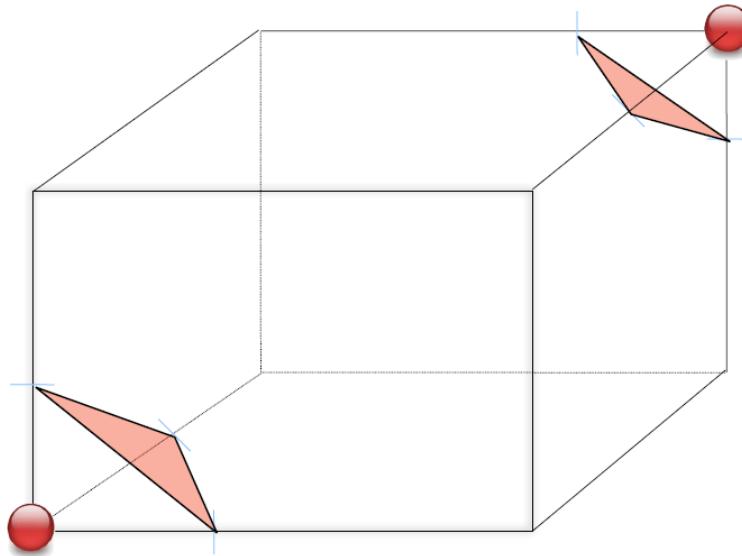


Use linear interpolation to place the polygon vertices on the edges

Grid edge $[p, q]$ with $f(p) = s_p$ and $f(q) = s_q$ and isovalue f_0



Generating a Polygon



Use linear interpolation to place the polygon vertices on the edges

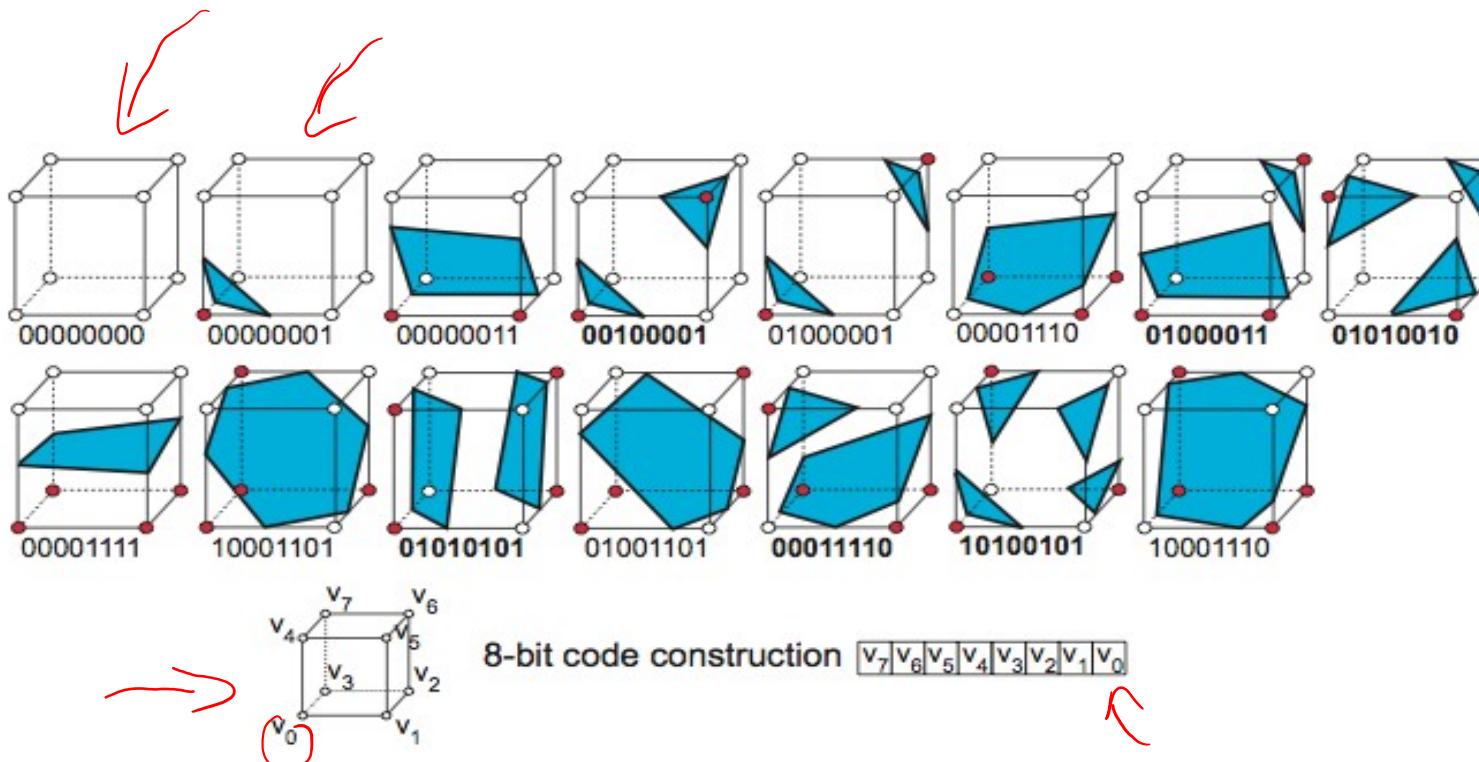
Grid edge $[p, q]$ with $f(p) = s_p$ and $f(q) = s_q$ and isovalue f_0

Place vertex at $(1 - t)p + q$ where $t = \frac{(t-s_p)}{(s_q-s_p)}$

Marching Cubes: Cases

Encode inside/outside state of each vertex w.r.t. contour in an 8-bit code

256 cases ($2^8=256$)

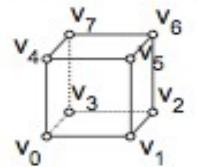
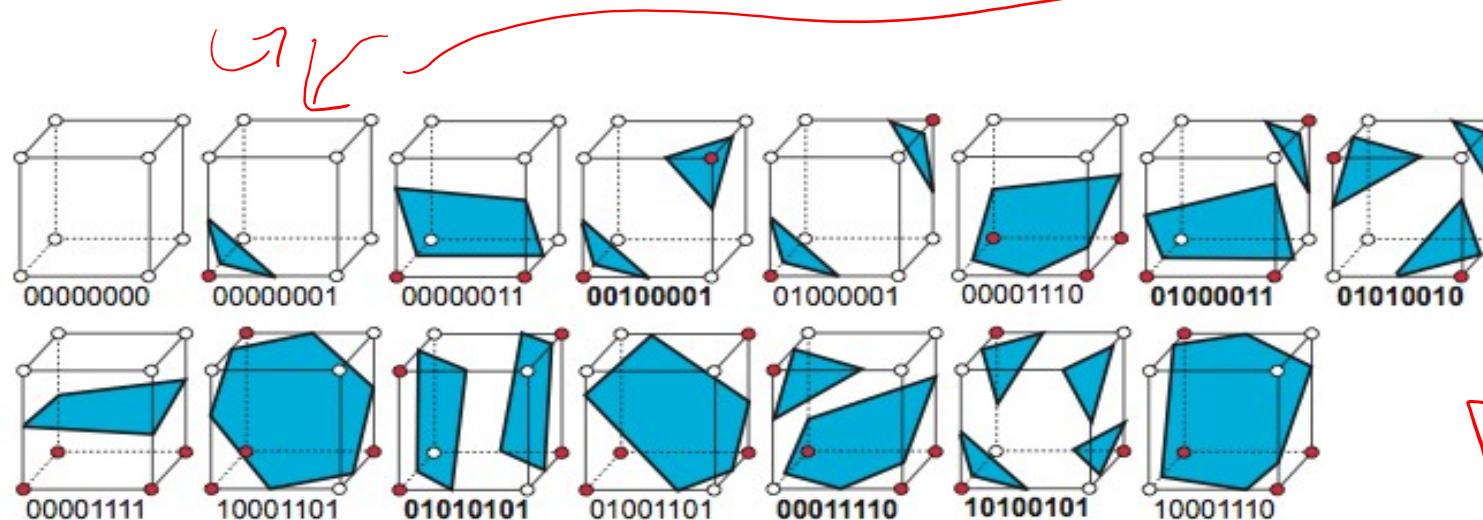


Marching Cubes: Cases

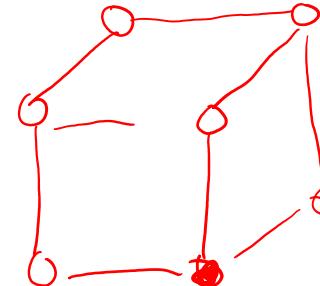
The 256 possible configurations can be grouped into these 15 cases

- complementarity (swapping positive and negative)
- rotational symmetry

Need to code 15 cases not 256 !



8-bit code construction $v_7 \mid v_6 \mid v_5 \mid v_4 \mid v_3 \mid v_2 \mid v_1 \mid v_0$

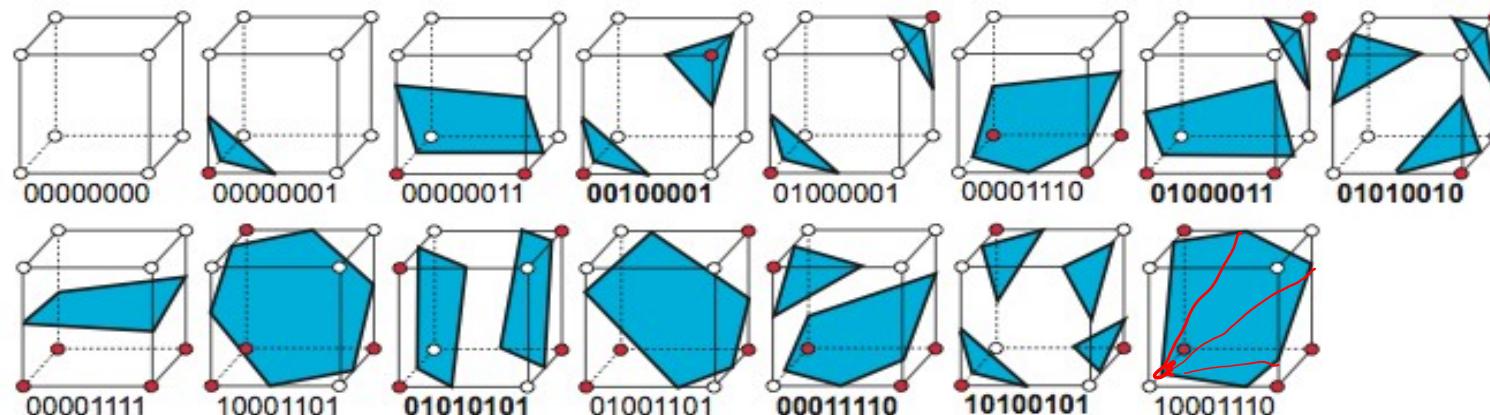


Marching Cubes: Cases

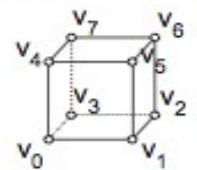
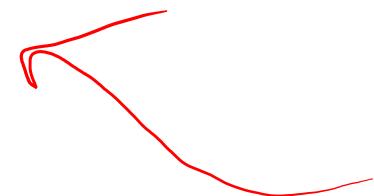
The 256 possible configurations can be grouped into these 15 cases

- complementarity (swapping positive and negative)
- rotational symmetry

Need to code 15 cases not 256 !



Non-triangle polygons can be triangulated



8-bit code construction

v ₇	v ₆	v ₅	v ₄	v ₃	v ₂	v ₁	v ₀
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

Marching Cubes Algorithm

Conceptually simple algorithm

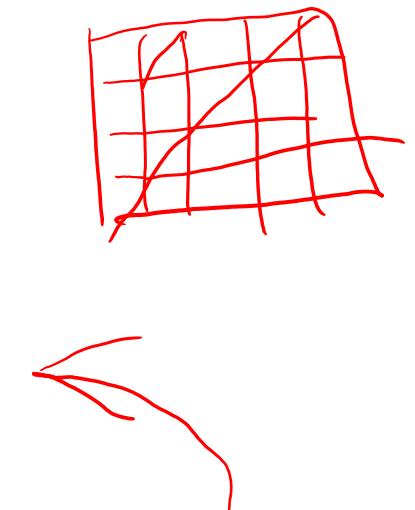
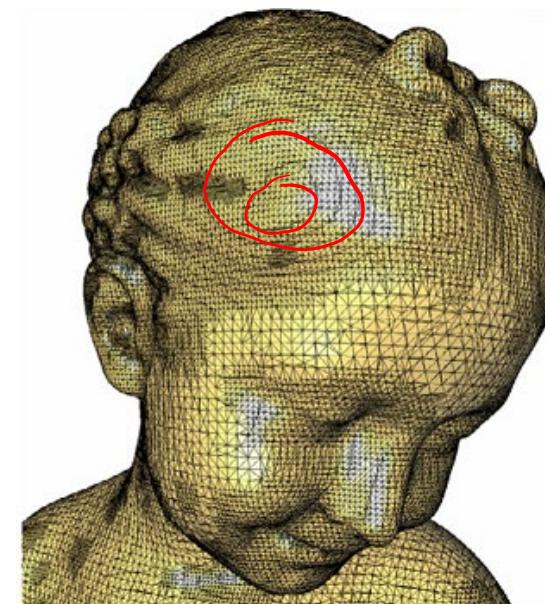
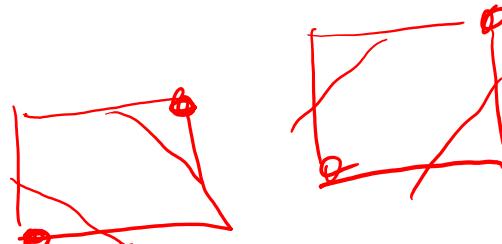
1. Classify vertices of a cube and generate bitcode
2. Read isosurface lookup table using bitcode
3. Retrieve triangles
4. Compute vertex coordinates using linear interpolation
5. Store the triangles....

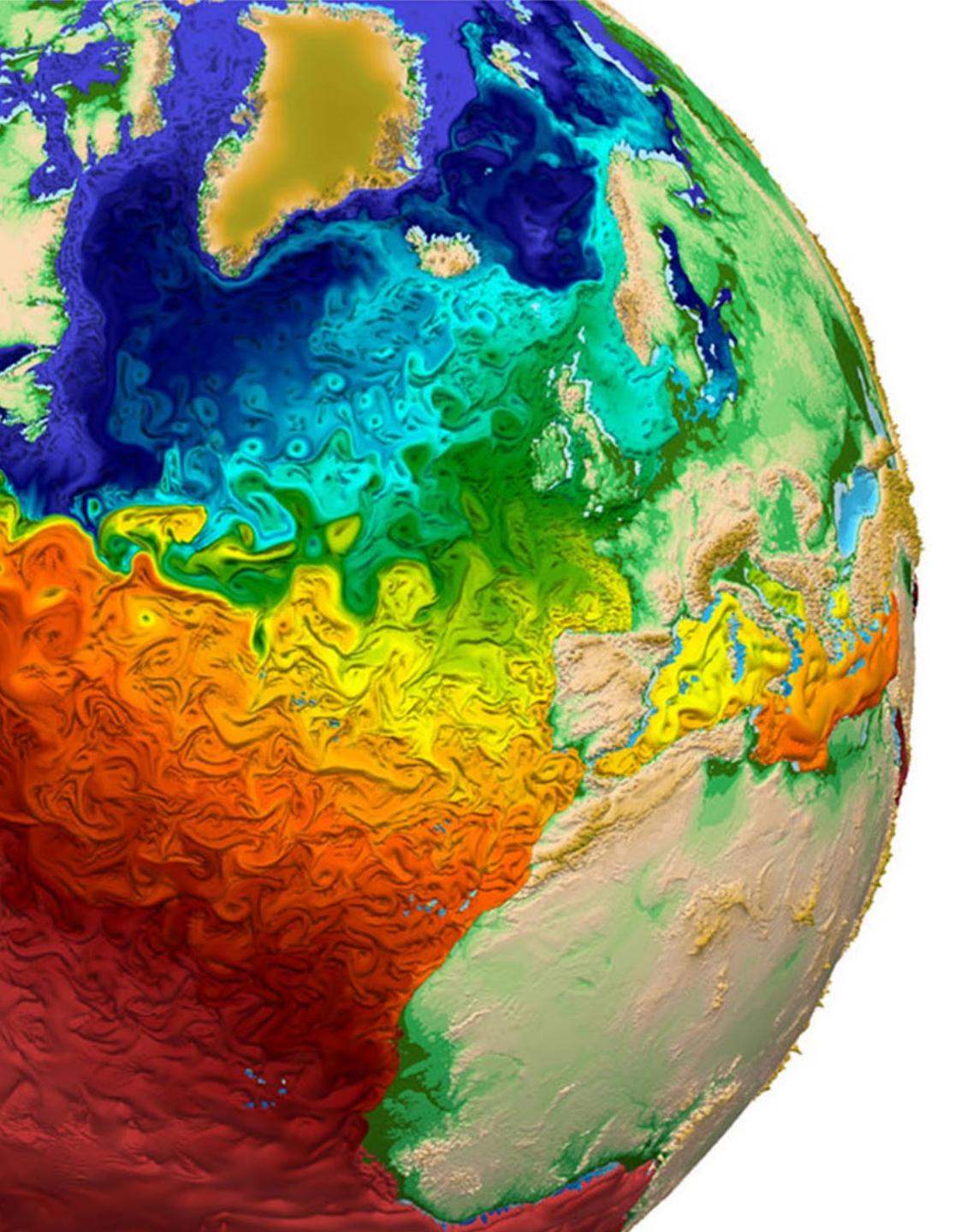
Process cubes in 2D sheets...marching by row and column within a sheet

Only need 2 sheets in memory at a time....

Tradeoffs

- Can be relatively memory efficient...scalable
 - only 2 sheets in memory at once
 - Write out triangles of a sheet to storage after neighbor sheet is done
- Parallelizable
- Relatively simple implementation
- Not adaptive...uses too many triangles
- Ringing artifacts
- Ambiguity





Contouring

Marching Cubes: Consistency and Correctness

Scientific Visualization
Professor Eric Shaffer

Correctness and Consistency

The utility of the visualization is impacted by

Correctness

An isosurface is correct if it matches the behavior of a known function that describes the samples in the data set.

Consistency

If each component of an isosurface is continuous then it is topologically consistent.

- Also topologically consistent if the isosurface's only holes are on the data set's boundary.

Standard MC provides neither of these...leading to many extension

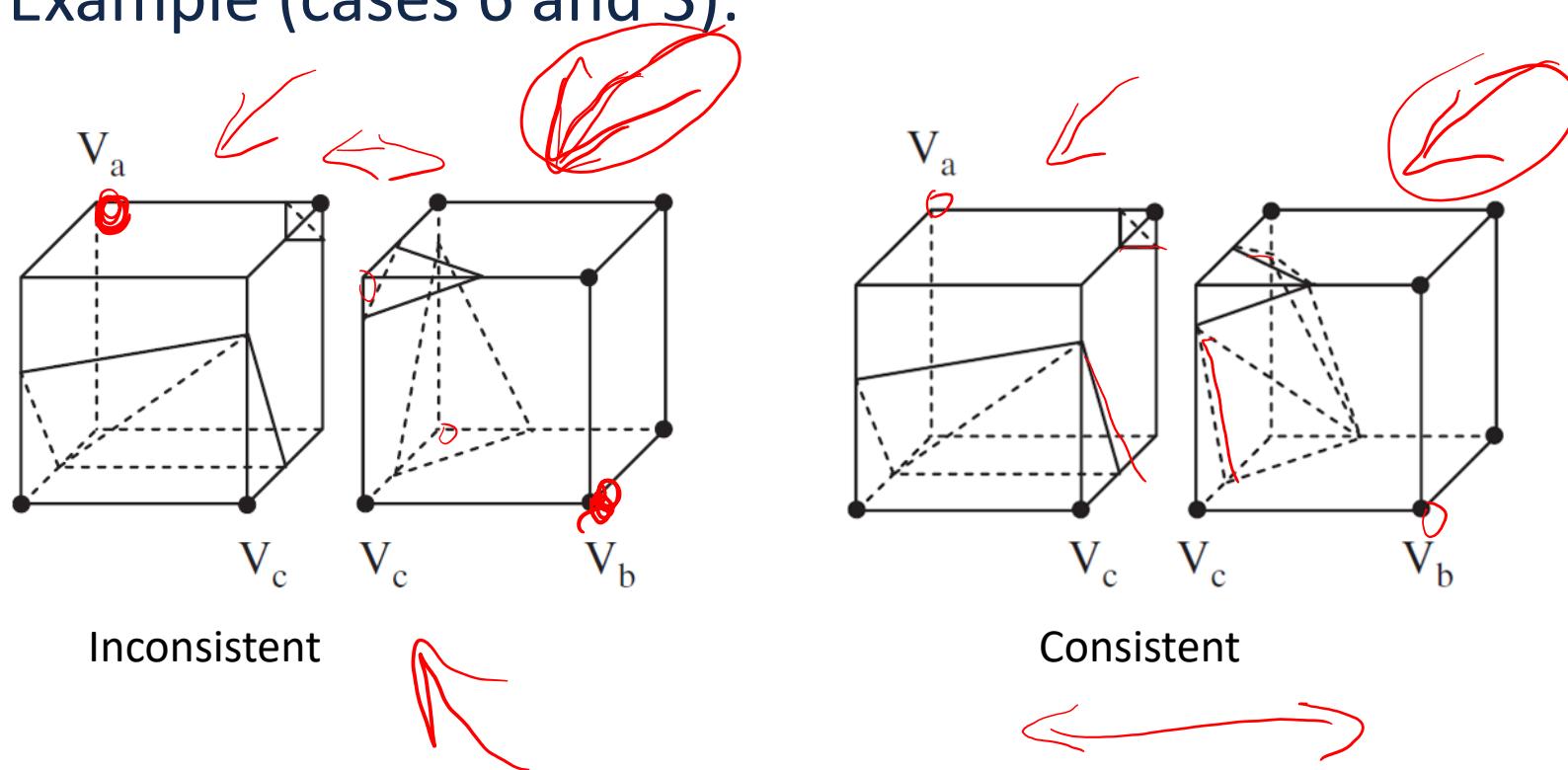
CFD data in particular is often generated by non-linear functions and so will not be strictly correctly represented by a standard MC surface

Face Ambiguity: Example

Several cell cases can be facetized in multiple ways

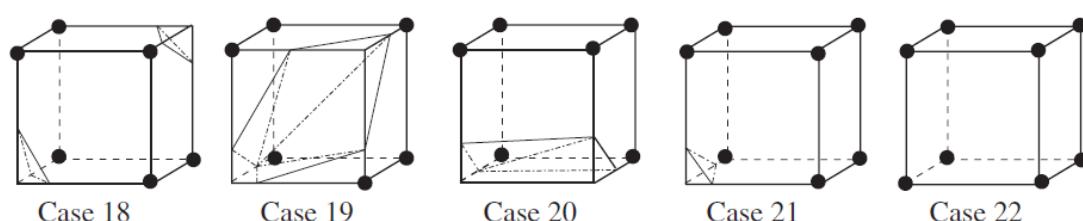
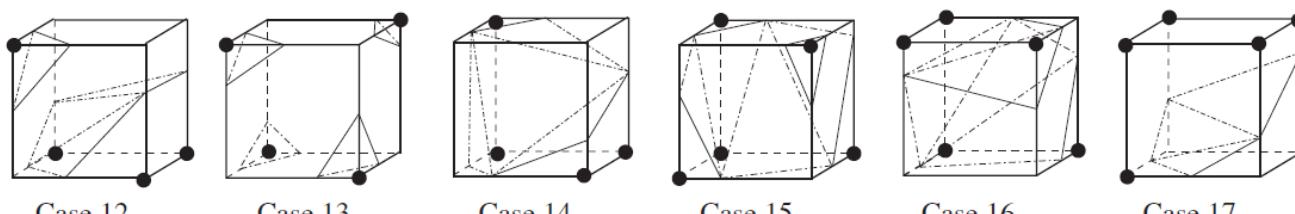
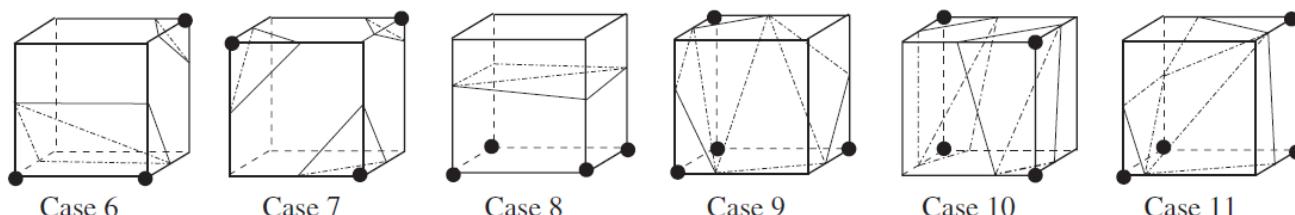
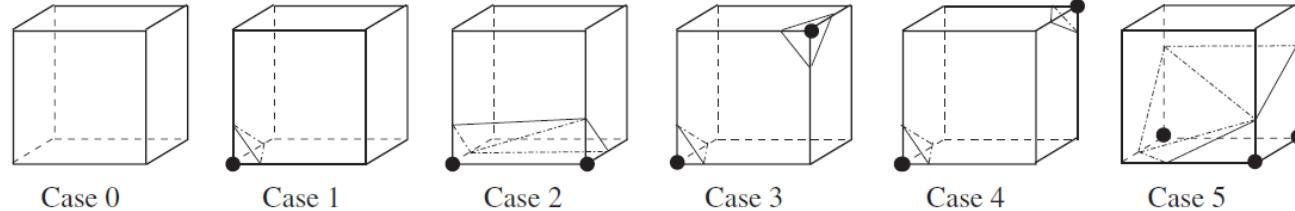
- Cases 3, 6, 7, 10, 12, and 13

Example (cases 6 and 3):



Face Ambiguity: One Solution

Build set of cases only considering rotational symmetry



Will produce a topologically consistent but not necessarily correct isosurface

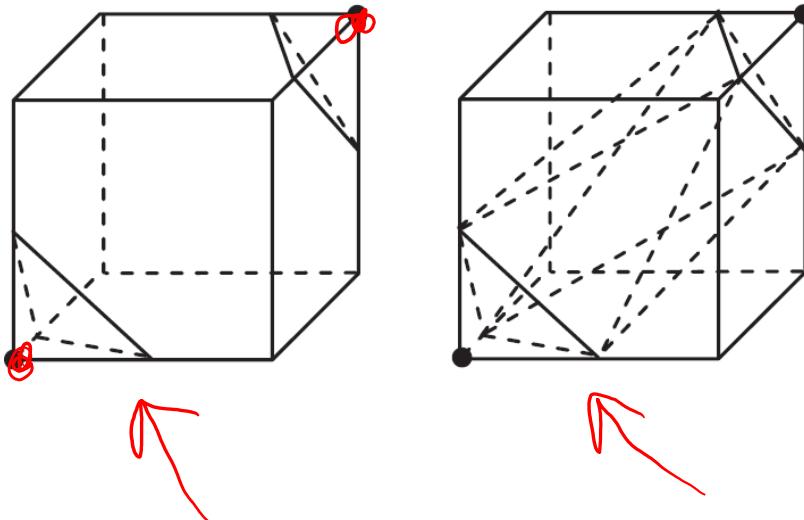
G. M. Nielson, "On marching cubes," in *IEEE Transactions on Visualization and Computer Graphics*, July-Sept. 2003

Internal Ambiguity

The facetization of a cube that has no ambiguous faces can still have internal ambiguity

Internal ambiguity does not cause any topological inconsistency but it can yield an incorrect isosurface

Internal ambiguity can arise in cases 4, 6, 7, 10, 12, and 13.



There are many, many ways people have addressed both kinds of ambiguity...we'll just discuss one

Trilinear Interpolant

Often don't know the underlying function of the field

How can we determine correctness?

One approach: assume trilinear interpolant of the samples is correct function

Within a unit cube the interpolant is

$$\begin{aligned} T(x, y, z) = & (1 - x)(1 - y)(1 - z)T(0, 0, 0) \\ & + \cancel{(1 - x)}(1 - z)zT(0, 0, 1) \\ & + (1 - x)y(1 - z)T(0, 1, 0) + (1 - x)yzT(0, 1, 1) \\ & + x(1 - y)(1 - z)T(1, 0, 0) + x(1 - y)zT(1, 0, 1) \\ & + xy(1 - z)T(1, 1, 0) + xyzT(1, 1, 1). \end{aligned}$$



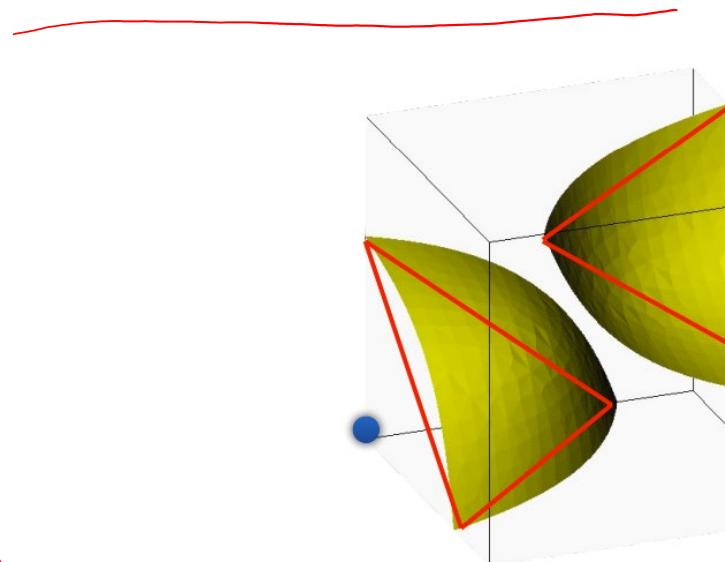
Trilinear Interpolant

$$\begin{aligned} T(x, y, z) &= (1 - x)(1 - y)(1 - z)T(0, 0, 0) \\ &\quad + (1 - x)(1 - y)zT(0, 0, 1) \\ &\quad + (1 - x)y(1 - z)T(0, 1, 0) + (1 - x)yzT(0, 1, 1) \\ &\quad + x(1 - y)(1 - z)T(1, 0, 0) + x(1 - y)zT(1, 0, 1) \\ &\quad + xy(1 - z)T(1, 1, 0) + xyzT(1, 1, 1) \end{aligned}$$
$$\left\{ \begin{array}{l} = \left(\frac{x_1 - x}{x_1 - x_0} \right) \left(\frac{y_1 - y}{y_1 - y_0} \right) \left(\frac{z_1 - z}{z_1 - z_0} \right) T(x_0, y_0, z_0) \\ + \left(\frac{x_1 - x}{x_1 - x_0} \right) \left(\frac{y_1 - y}{y_1 - y_0} \right) \left(\frac{z - z_0}{z_1 - z_0} \right) T(x_0, y_0, z_1) \\ + \left(\frac{x_1 - x}{x_1 - x_0} \right) \left(\frac{y - y_0}{y_1 - y_0} \right) \left(\frac{z_1 - z}{z_1 - z_0} \right) T(x_0, y_1, z_0) \\ \vdots \\ + \left(\frac{x - x_0}{x_1 - x_0} \right) \left(\frac{y - y_0}{y_1 - y_0} \right) \left(\frac{z - z_0}{z_1 - z_0} \right) T(x_1, y_1, z_1). \end{array} \right.$$

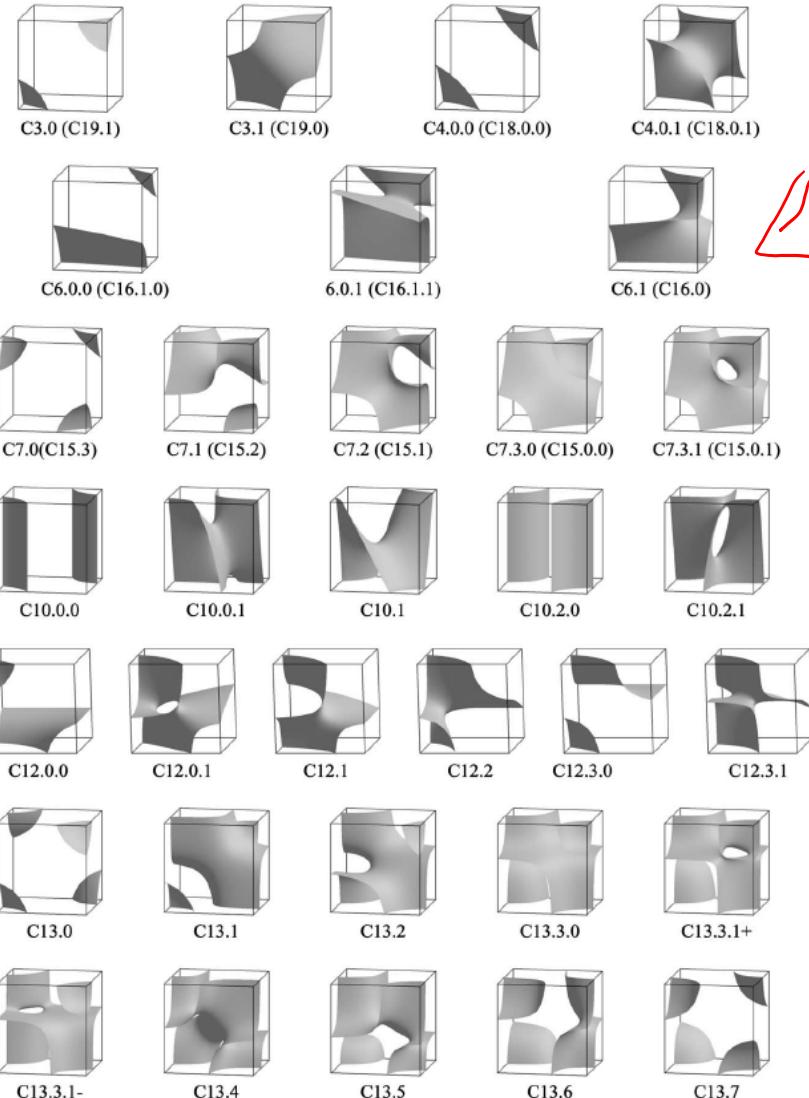
True isosurface of a trilinear function is a cubic curved surface

Contours are hyperbola

MC approximates with triangles



Trilinear Interpolant



Can make isosurface consistent with trilinear interpolant

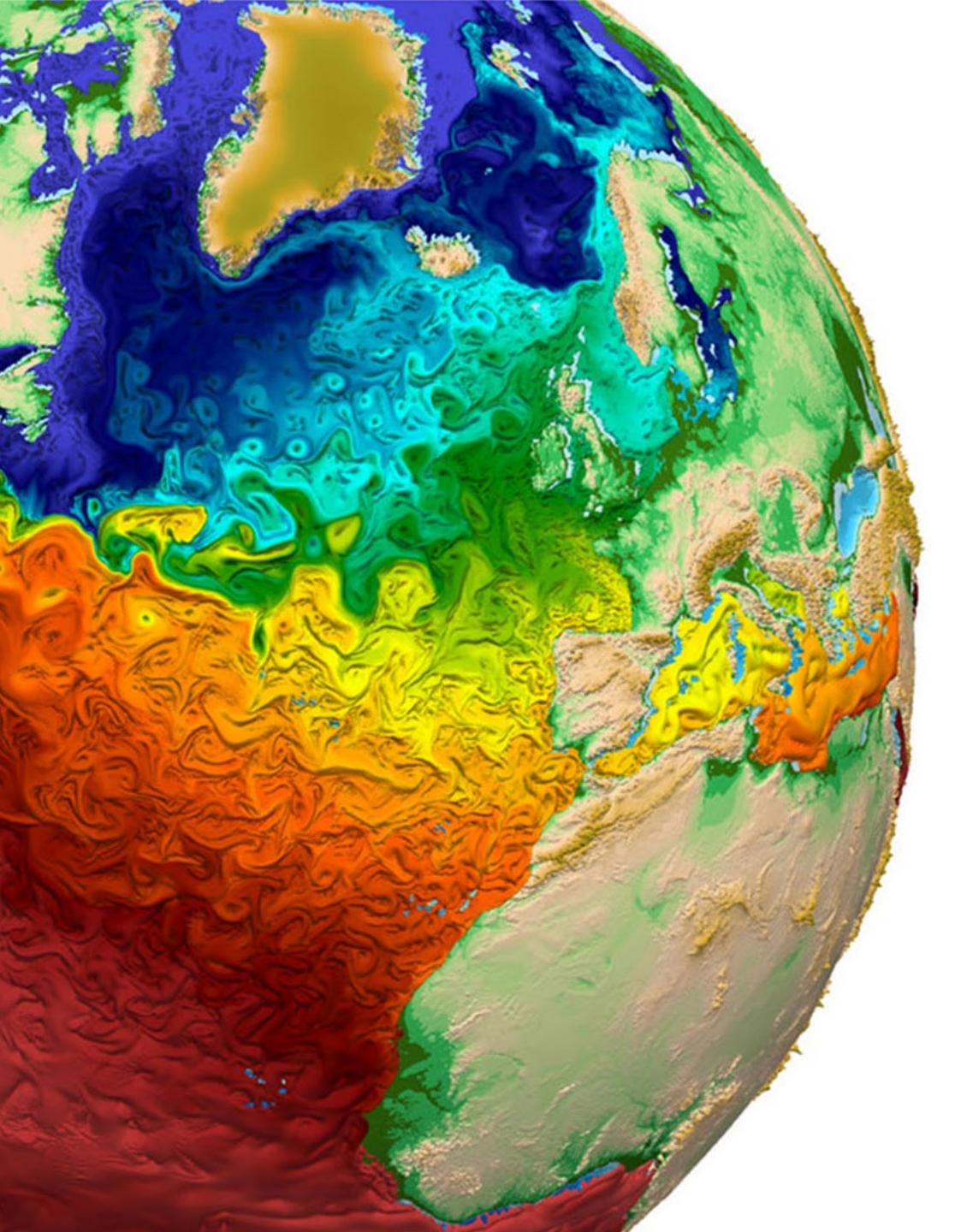


Resolve ambiguities as trilinear interpolant would

G. M. Nielson, "On marching cubes," in IEEE Transactions on Visualization and Computer Graphics, July-Sept. 2003

...3 step process to assure correctness assuming trilinear interpolant is underlying function





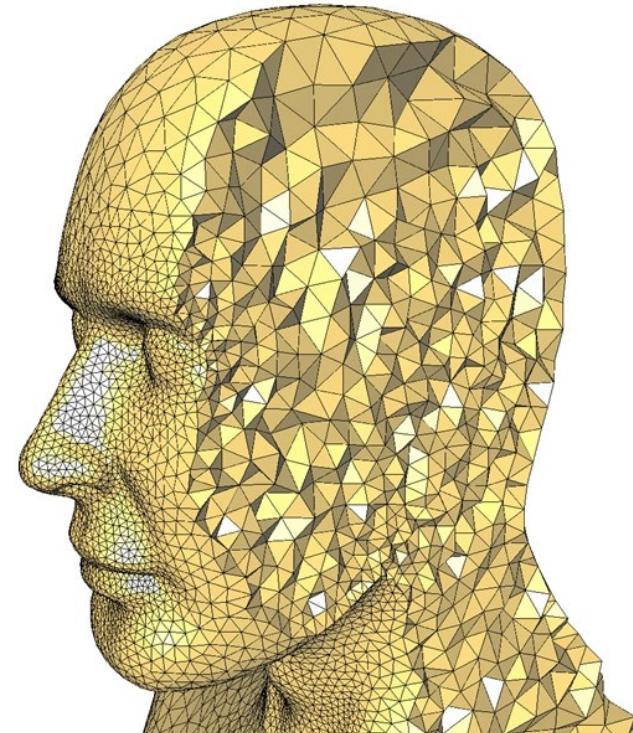
Contouring

Marching Tetrahedra

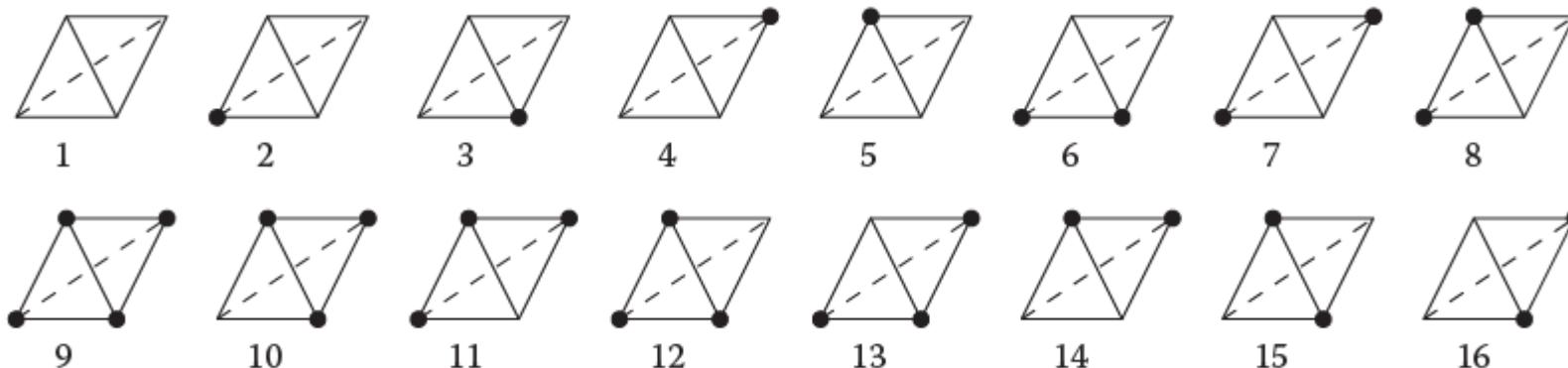
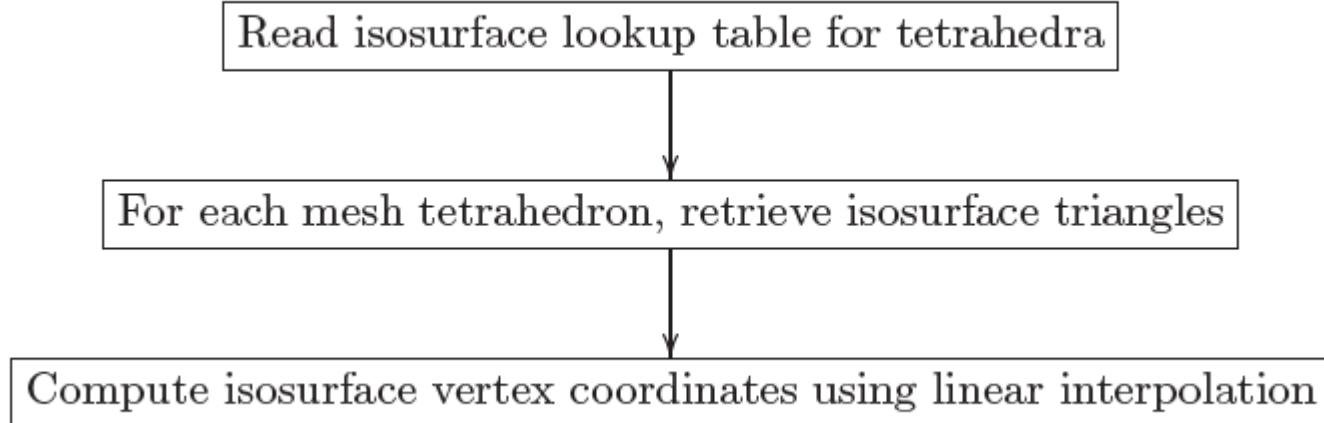
Scientific Visualization
Professor Eric Shaffer

Motivation

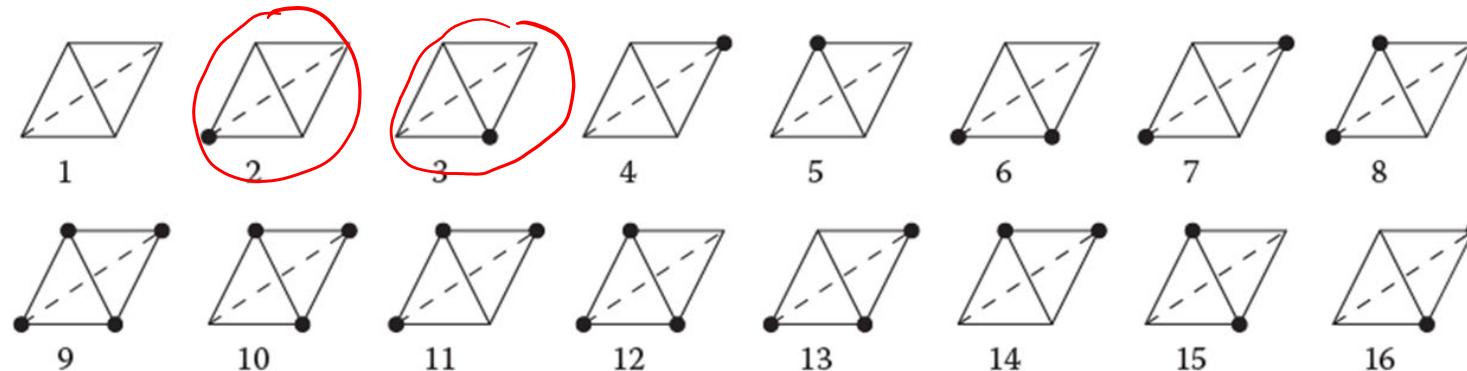
- Marching Cubes was patented
- MC applies to structured grids...not unstructured tetrahedral meshes
 - ..tet meshes are popular
- Handling ambiguities in MC is not easy



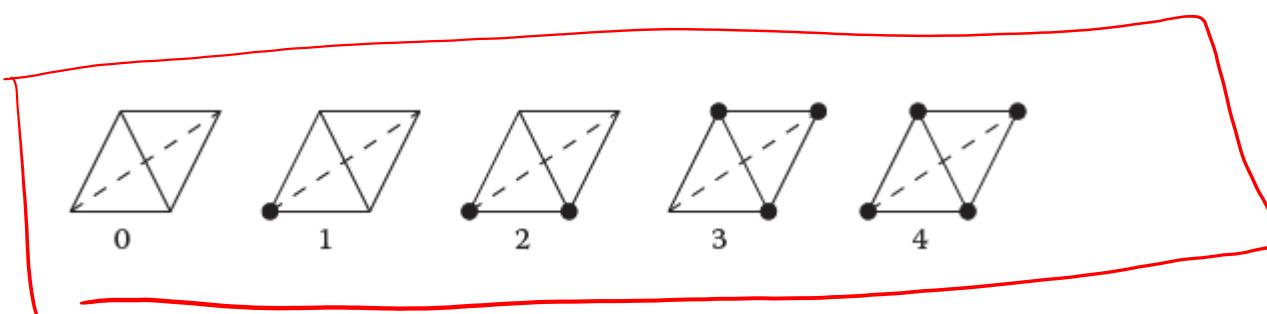
Marching Tetrahedra: Algorithm



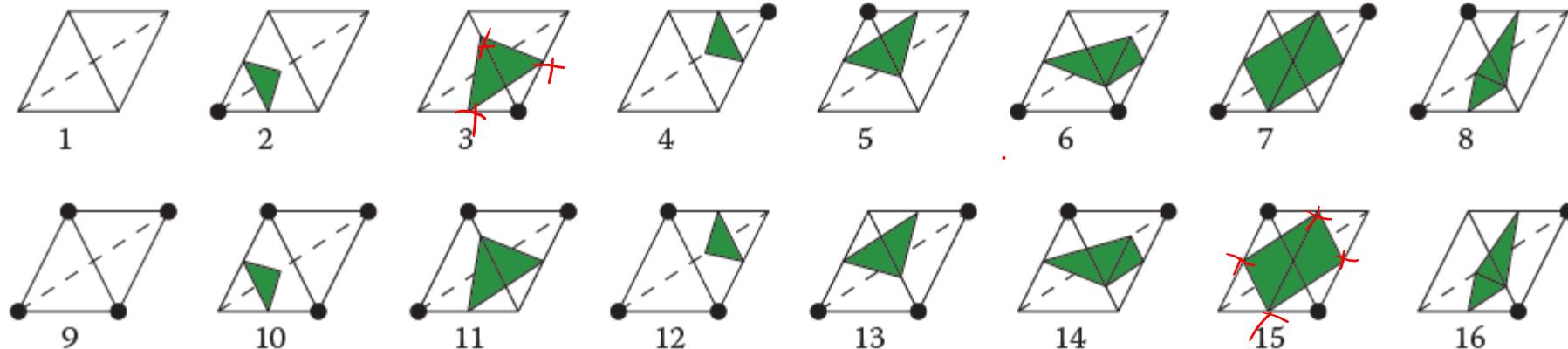
Cell Configurations



Only 5 equivalence classes of configurations if we allow rotation



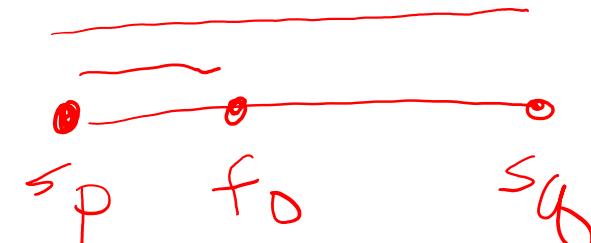
Generating the Isosurface



Generate a vertex along each bipolar edge

For mesh edge $[p, q]$ with $f(p) = s_p$ and $f(q) = s_q$ and isovalue f_0

Place vertex at $(1 - t)p + tq$ where $t = \frac{(f_0 - s_p)}{(s_q - s_p)}$ $\in [0, 1]$



Using these vertices, generate one or two triangles inside the tetrahedron for the isosurface patch

Properties of the Isosurface

Assume:

- The isovalue does not equal the scalar value of any mesh vertex.
- The tetrahedral mesh is a partition of a 3-manifold with boundary.

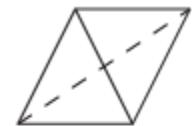
Resulting Properties

- The isosurface is a piecewise linear, orientable 2-manifold with boundary.
- The boundary of the isosurface lies on the boundary of the grid.
- The isosurface does not contain any zero-area triangles or duplicate triangles

Ambiguity

Marching Tetrahedra has no ambiguous configurations.

- The four vertices of each tetrahedron are connected by tetrahedron edges
- Each such edge is intersected at most once by the isosurface
- There is no possible ambiguity in the isosurface construction



1



2



3



4



5



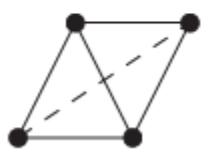
6



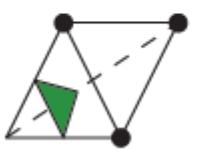
7



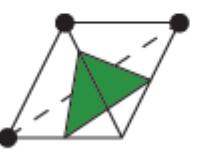
8



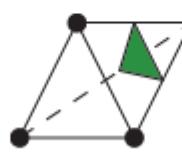
9



10



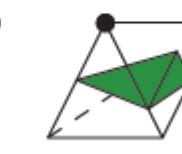
11



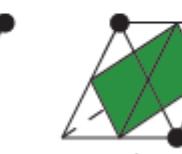
12



13



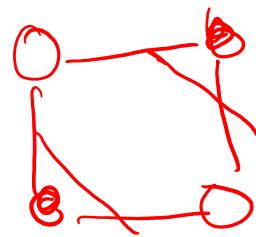
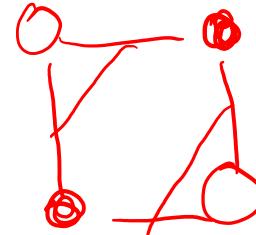
14



15



16



Disadvantages

- Generates a large number of triangles
 - ...more than Marching Cubes
- Unstructured nature of the tetrahedral mesh impacts scalability
 - Harder to manage storage
 - Partitioning for parallel processing

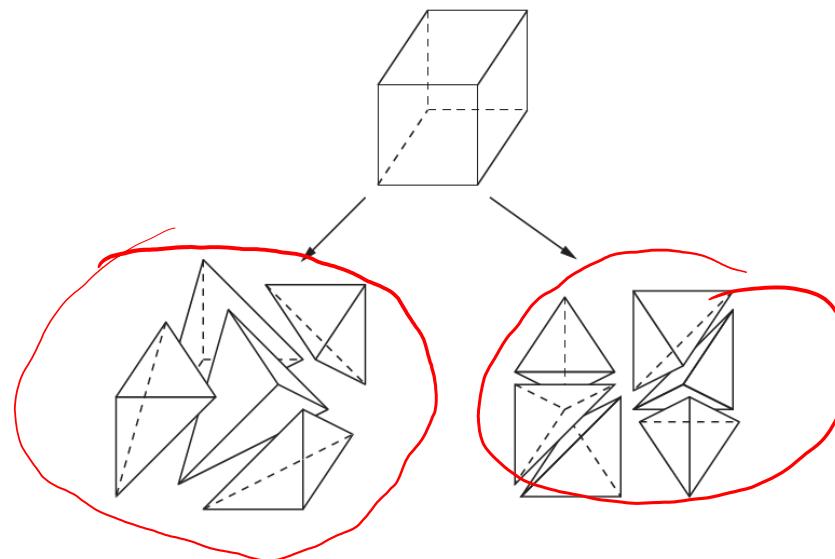
All these issues can be overcome...

Cube Tetrahedralization

Can apply marching tetrahedra to regular grids

Decompose cubes into tetrahedra

Can choose to use either 5 or 6 tetrahedra per cube

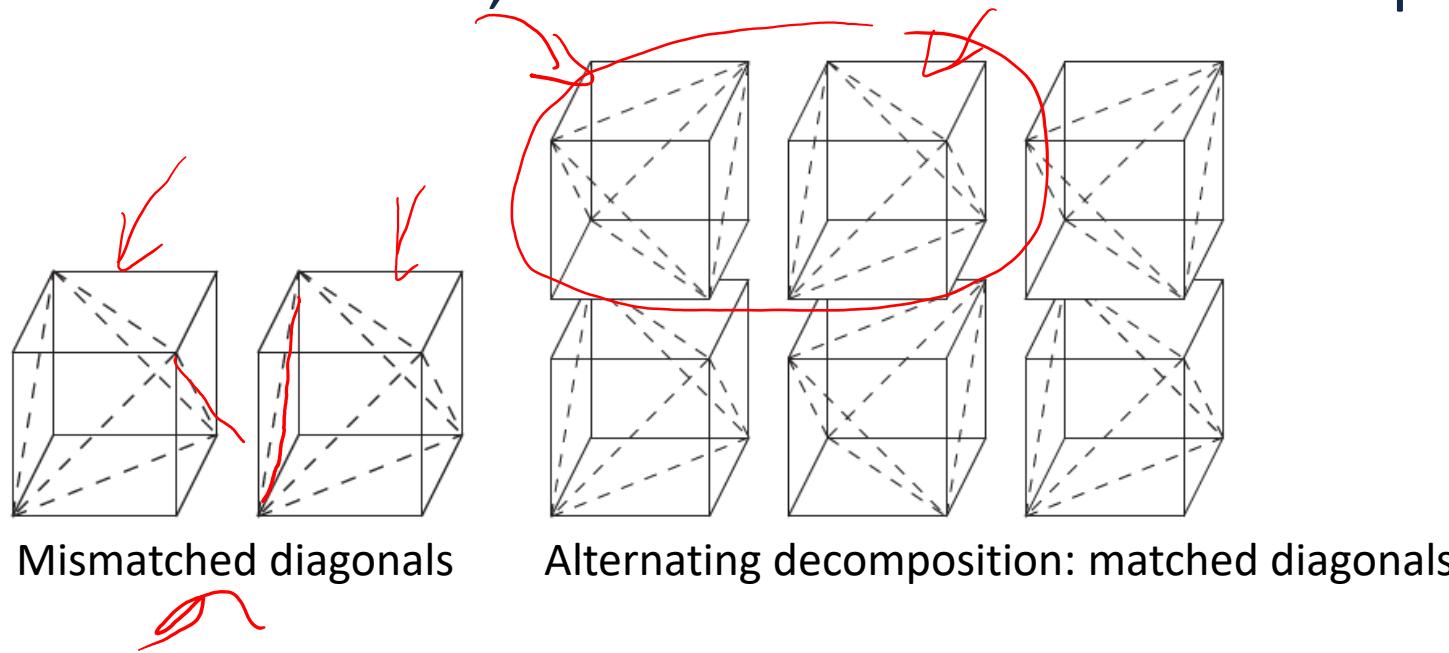


Cube Tetrahedralization

This creates diagonal facets on the square faces of the cube

Diagonals of adjacent cube faces must match

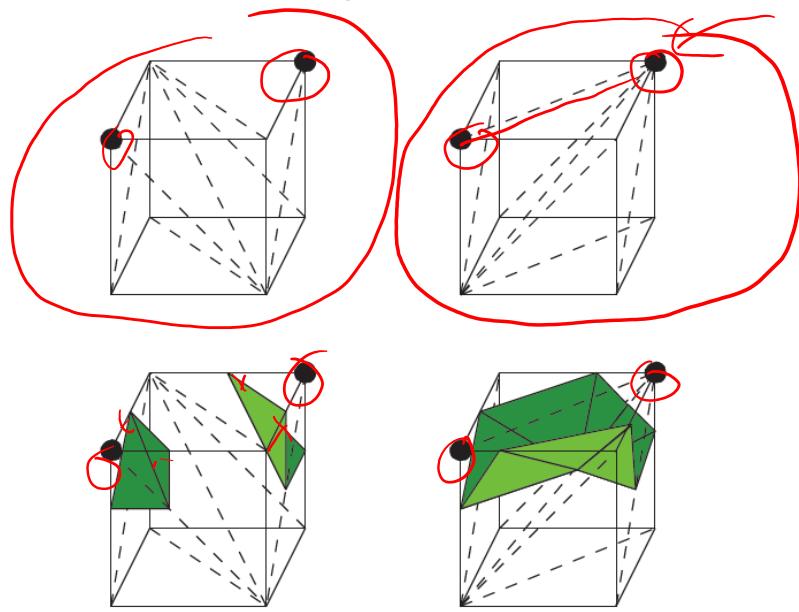
- Using 6 tetrahedra per cube results in matching diagonals
- If 5 tetrahedra are used, need to alternate the decomposition



Where did the Ambiguity Go?

- MC has ambiguous cases
- MT does not

Choice of tetrahedral decomposition orientation resolves the ambiguity



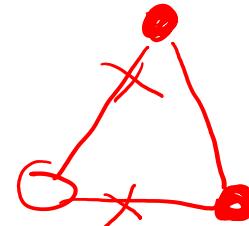
Isocontouring a Triangle Mesh

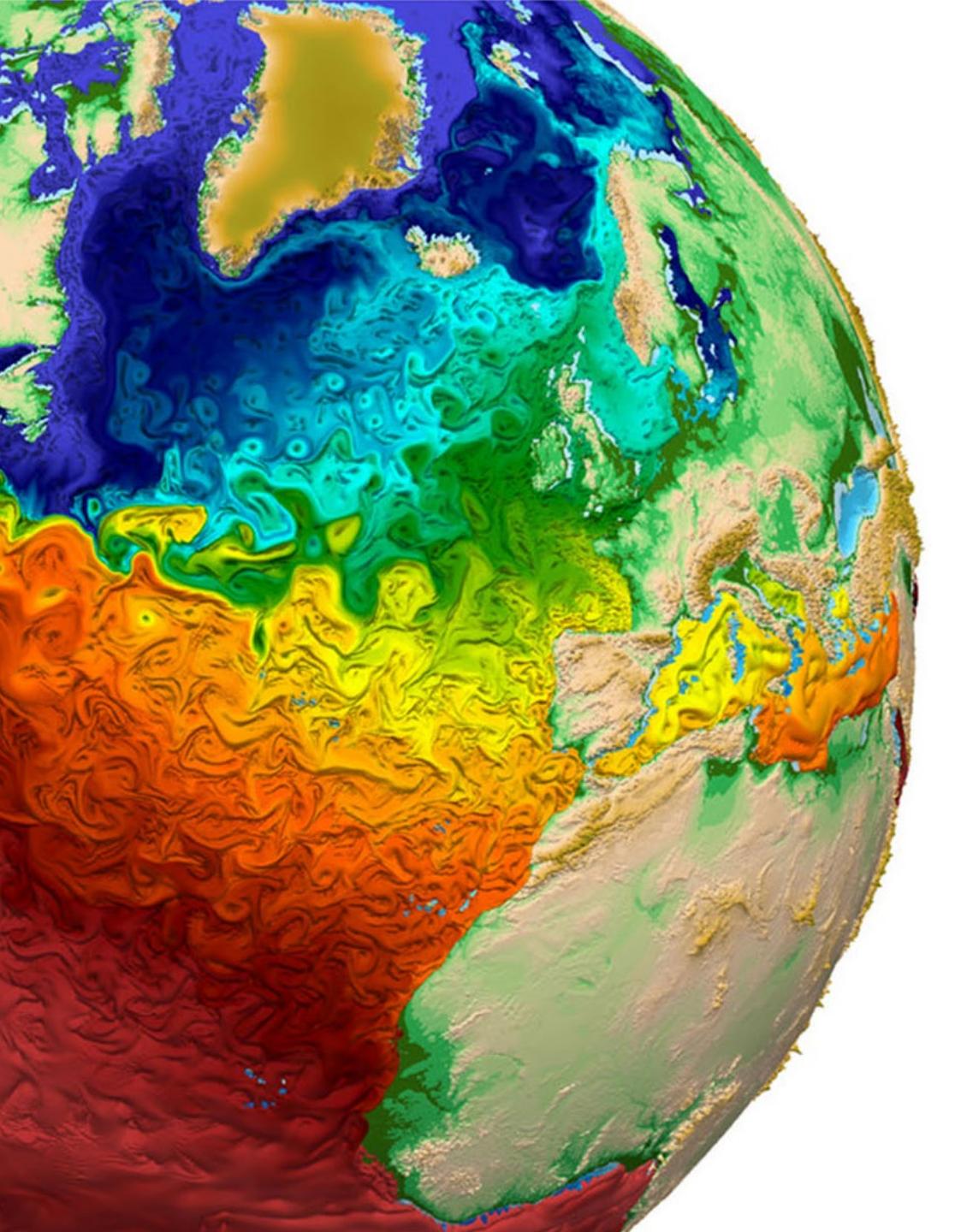
Extremely simple algorithm

...no table

...each triangle has at most one contour edge

...place the vertices of contour edge using linear interpolation.





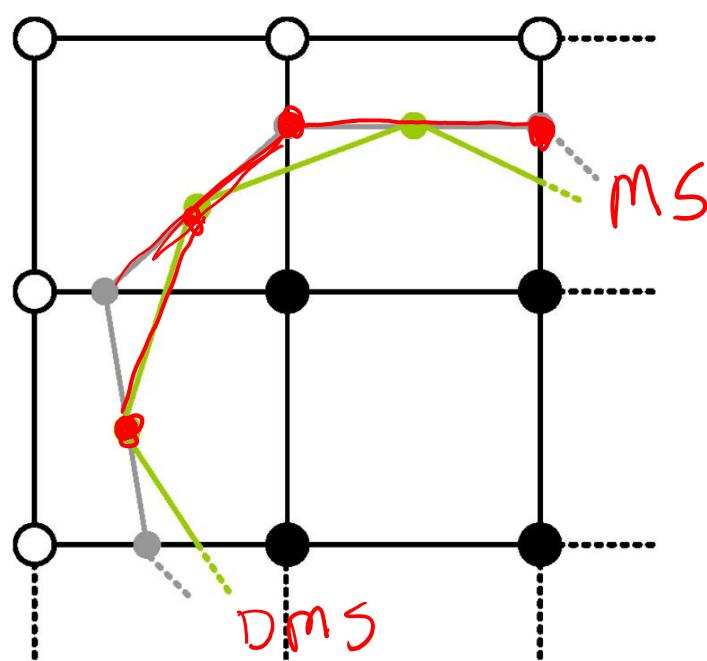
Contouring

Dual Marching Squares

Scientific Visualization
Professor Eric Shaffer

Dual Methods

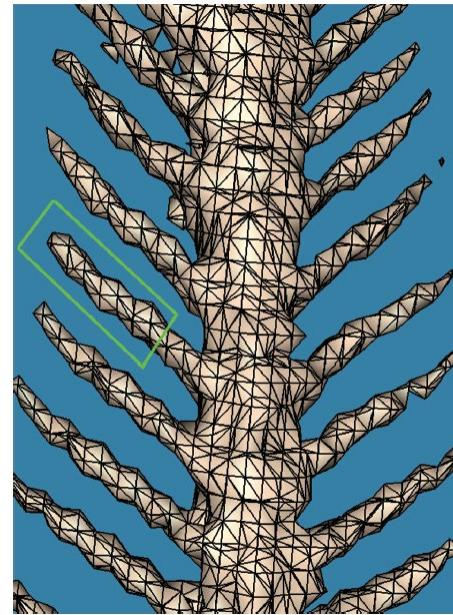
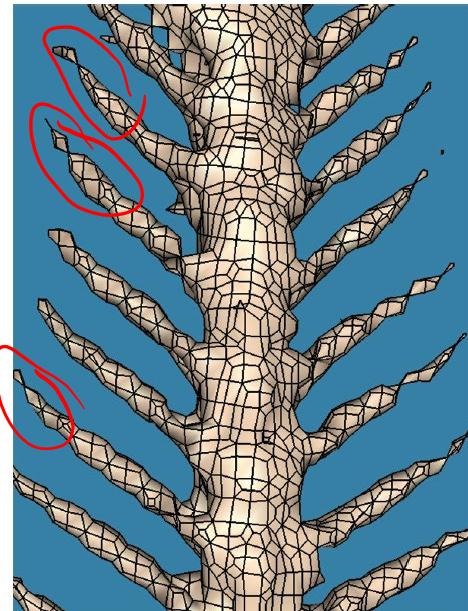
- Dual contouring places isosurface vertices inside mesh elements
- Isosurface vertices in adjacent elements are connected by edges
 - In 3D these edges connect to form facets (faces of a mesh)



MS
Edge ← → Vertex

Dual Marching Cubes and Dual Marching Squares

- Allow more than one vertex per cube (squares)
- Dual MC generates quadrilaterals instead of triangles
 - ...obviously you can triangulate the quadrilaterals

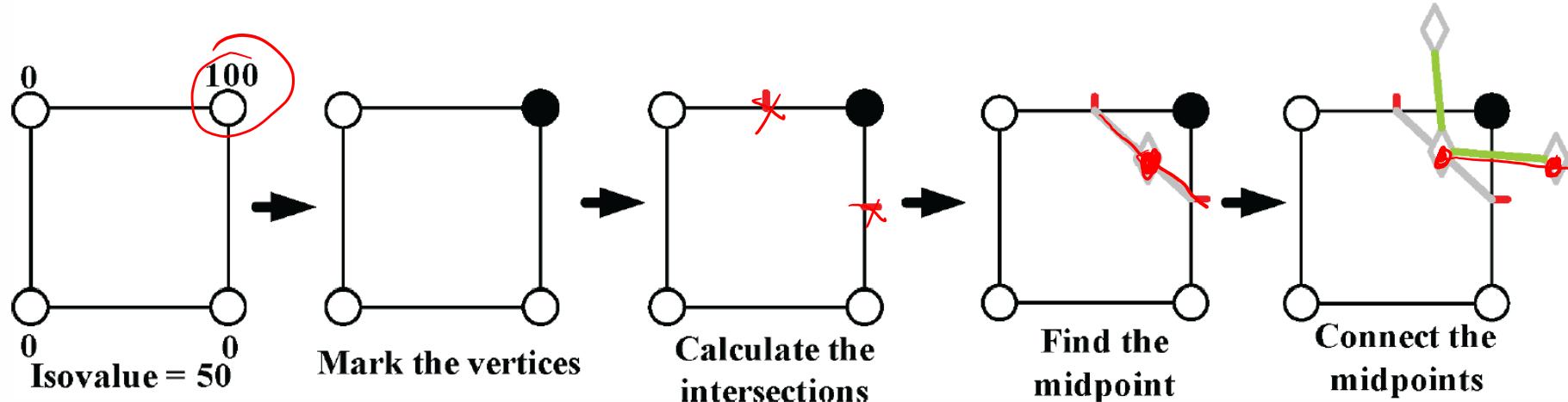


Primal
Marching Ts
dual



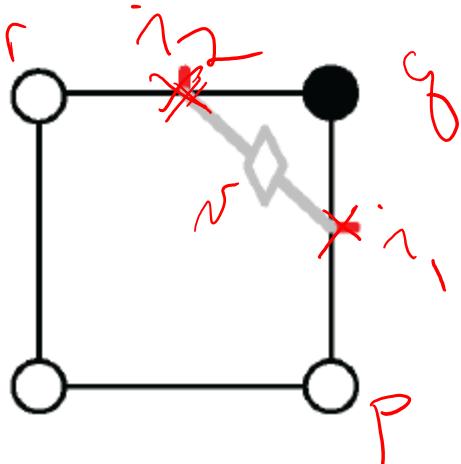
MC surface will be manifold
DMC surface unlikely to be manifold

Dual Marching Squares: Algorithm



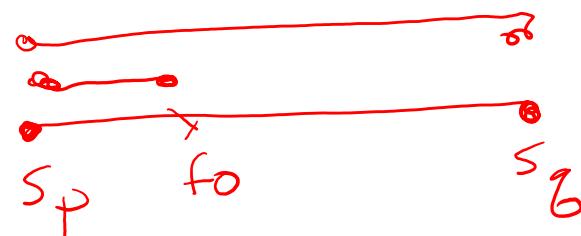
Positioning Vertices

- Compute an intersection point along each bipolar edge
 - Use linear interpolation to estimate where isoline crosses the edge
- Find the midpoint between the intersection points
- Position vertex for isoline at the midpoint



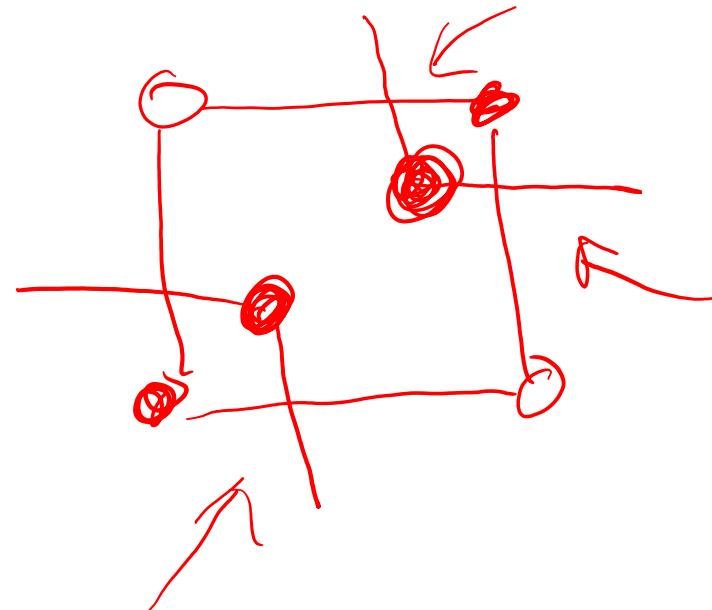
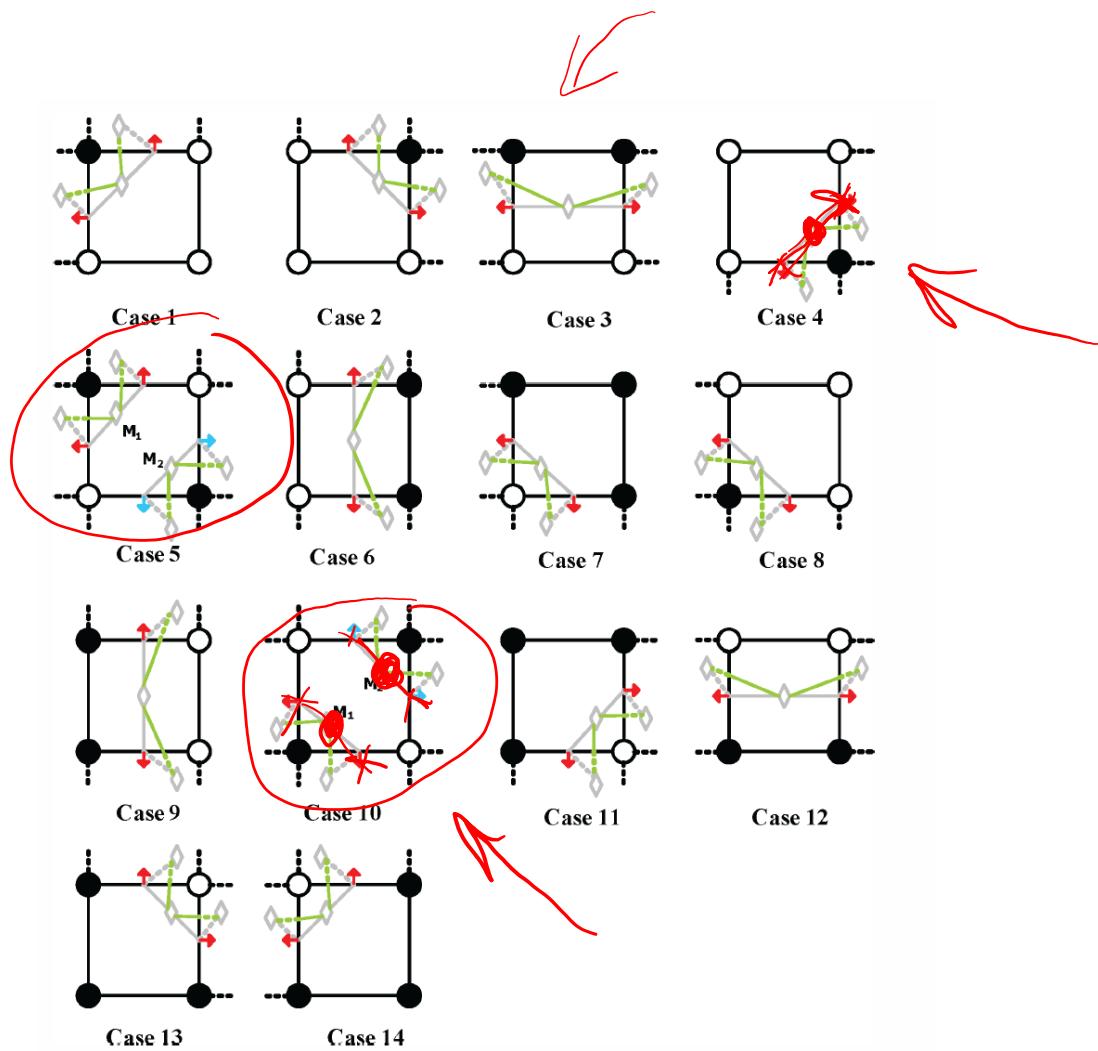
For mesh edge $[p, q]$ with $f(p) = s_p$ and $f(q) = s_q$ and isoline value f_0

Place intersection point at $(1 - t)p + tq$ where $t = \frac{(f_0 - s_p)}{(s_q - s_p)}$ $\in [0, 1]$



$$r = \frac{j_1 + j_2}{2}$$

Cell Configurations

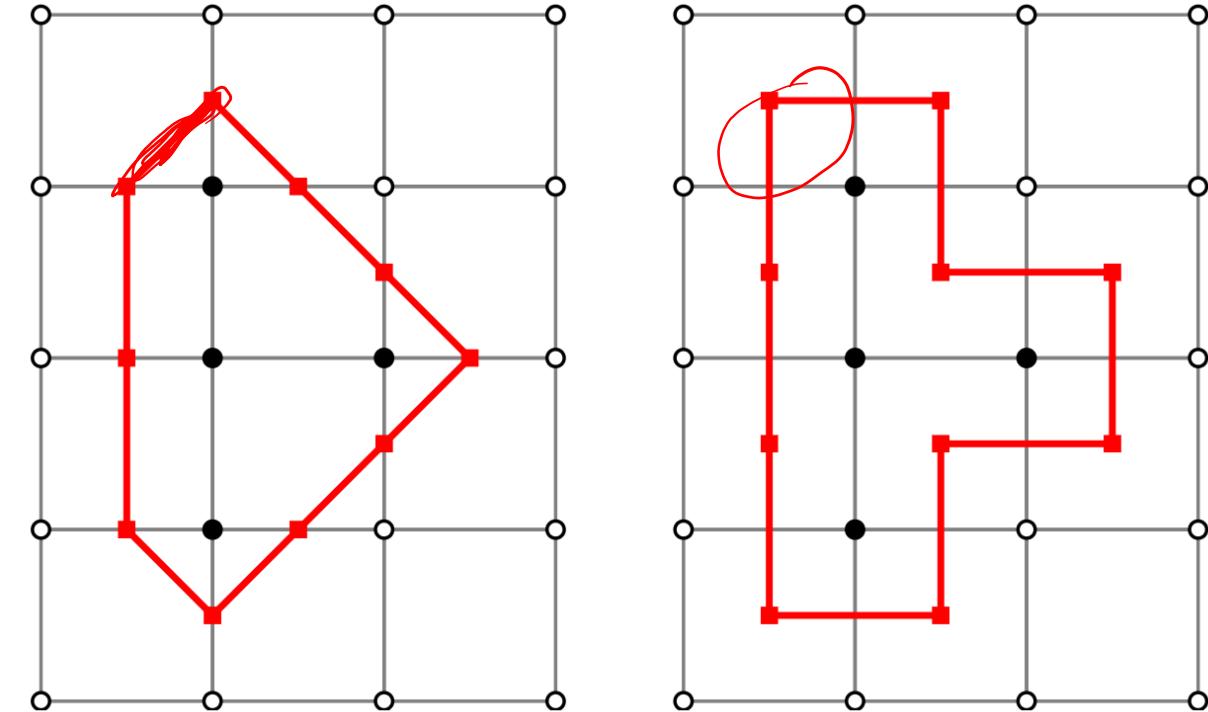


So Why Do This?

So...why do this when MS already exists?

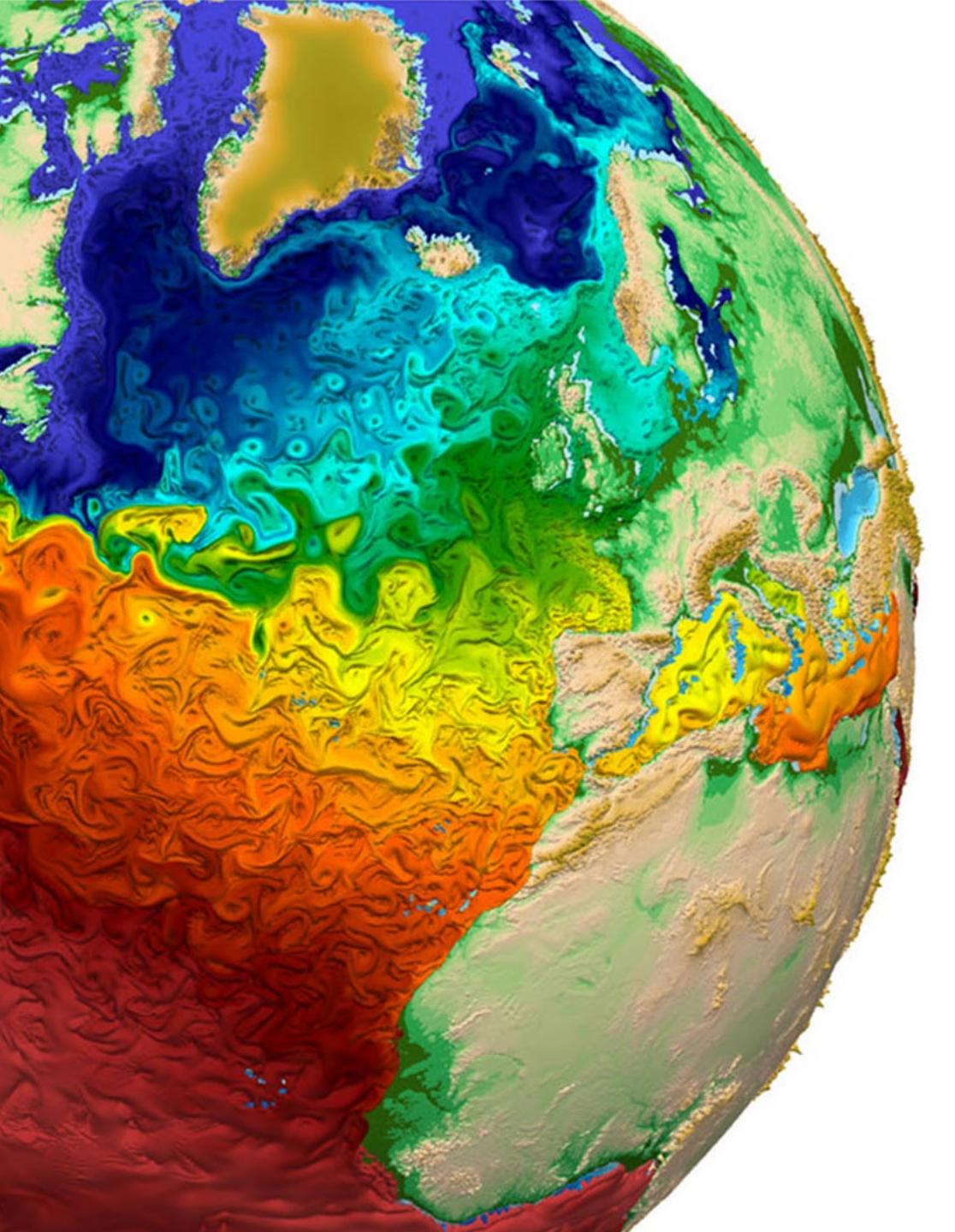
Empirical evidence that DMS generates better approximations to curved isolines.

- Better at reproducing sharp features
- Simpler to implement



Primal

Dual



Contouring

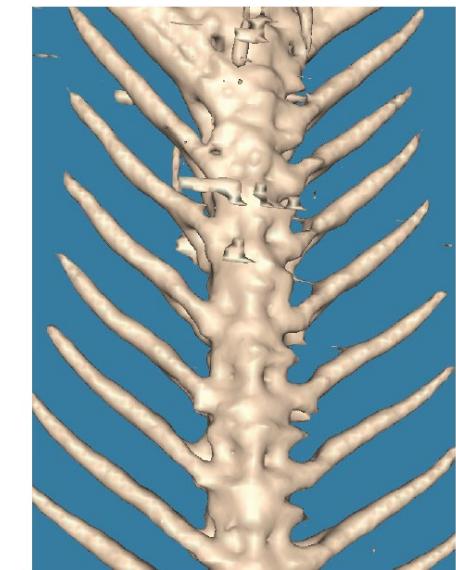
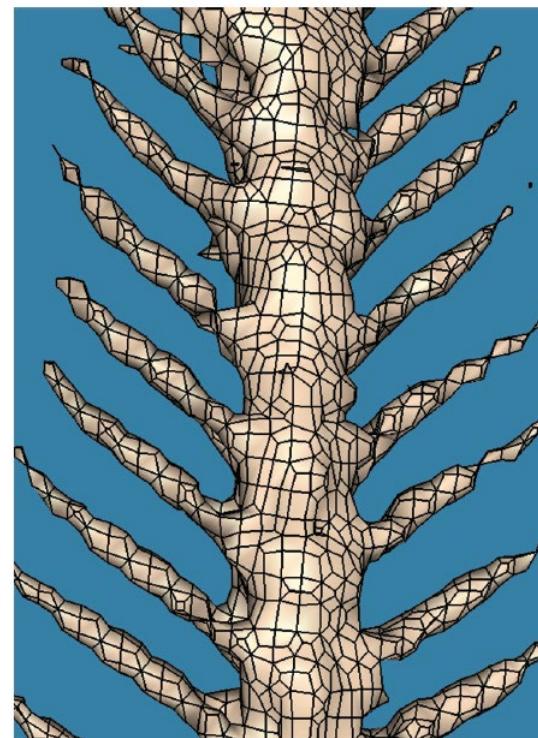
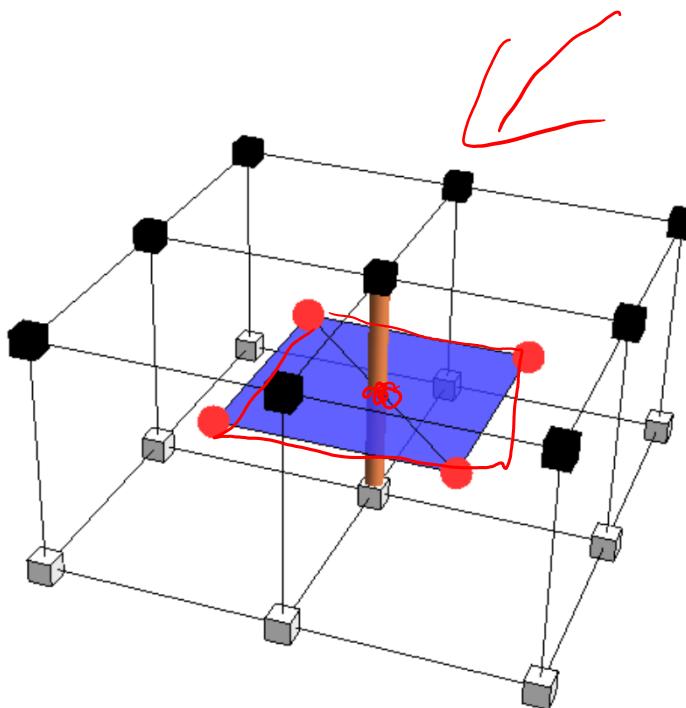
Dual Marching Cubes

Scientific Visualization
Professor Eric Shaffer

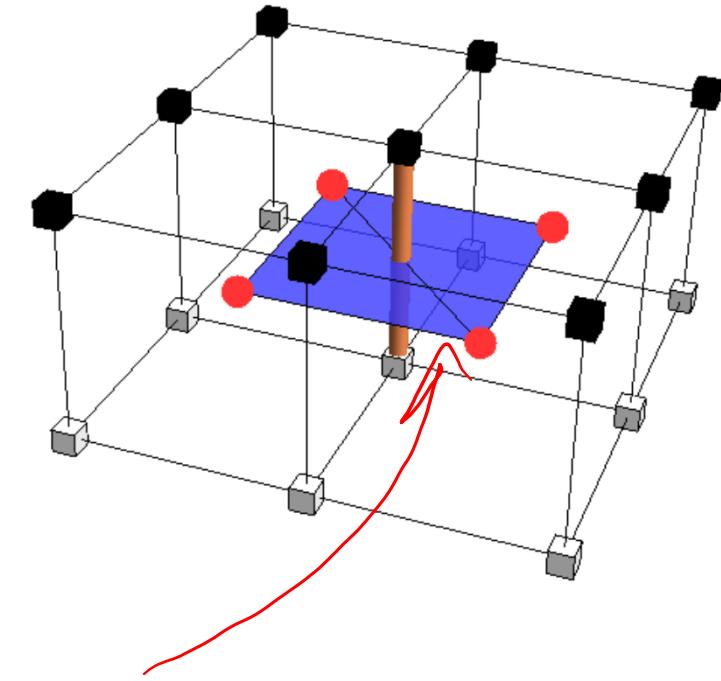
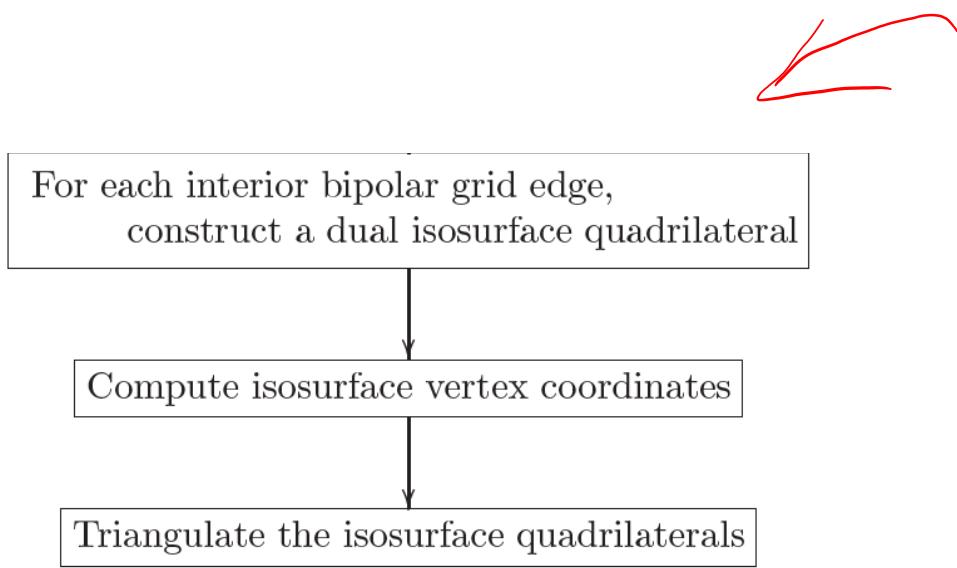
Dual Marching Cubes

Operates on structured grids...like Marching Cubes

Generates quadrilaterals...dual to the vertices generated by Marching Cubes



Dual Marching Cubes: Algorithm

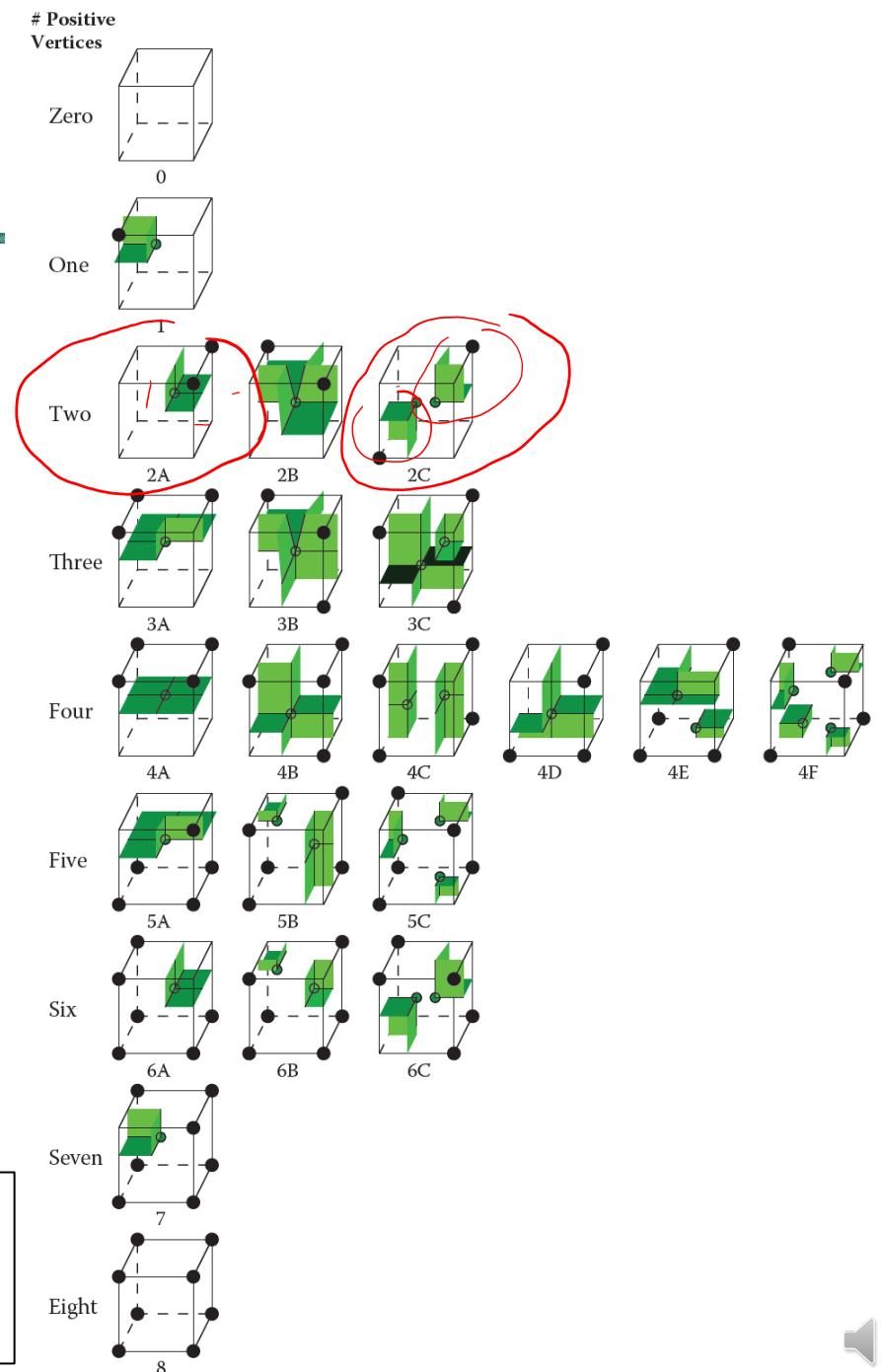


How Many Vertices in a Cell?

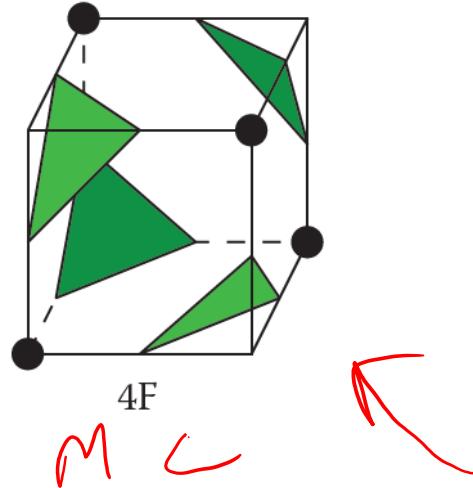
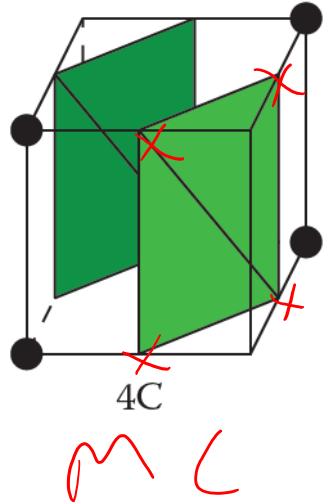
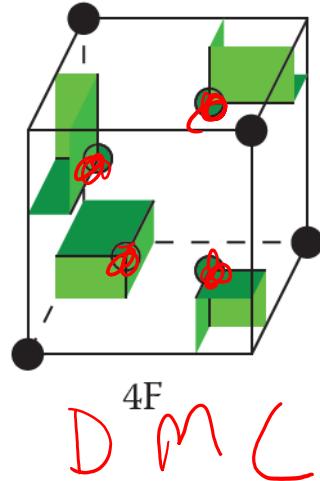
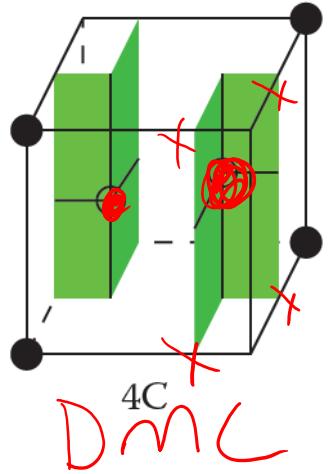
- Each bipolar edge generates a vertex in cells around it
 - Corner of a quadrilateral of the isosurface
 - Quad will intersect four cells
- Some vertices are shared....
- Cases are based on MC cases
 - Each polygonal patch in a cell will generate a vertex in DMC
- Can create a table of cases....

Table-based algorithm from:

Nielson, G. M. (2004). Dual Marching Cubes. In *Proceedings of IEEE Visualization 2004*, pages 489–496, Los Alamitos, CA. IEEE Computer Society.



Duality



Each polygonal patch in a MC cell → vertex in the dual

We can associate a set of bipolar edges with the dual vertex

Each bipolar edge creating vertex for a MC patch,
will be in the set for the associated dual vertex

These edges are used to position the dual vertex

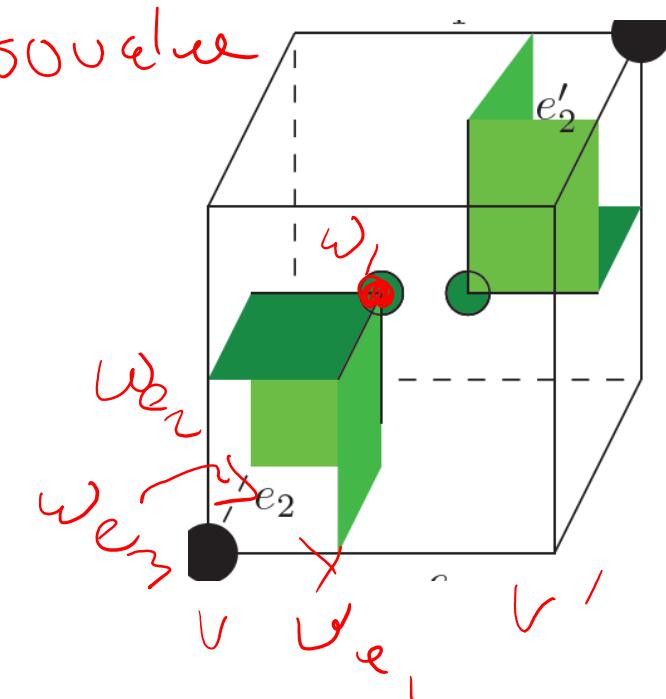
Positioning Vertices

For each bipolar grid edge $e = (v, v')$,
 $w_e \leftarrow \text{LinearInterpolation}(v, F(v), v', F(v'), \sigma)$

For each grid cube c ,
For each isosurface vertex w_c^m in c ,
 $\text{coordinates}(w_c^m) \leftarrow$
centroid of $\{w_e : \text{edge } e \text{ is associated with } w_c^m\}$

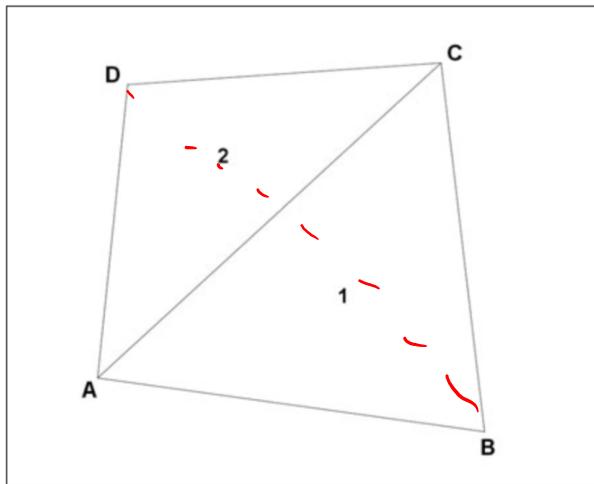
$$t = \frac{\sigma - F(\omega)}{F(\omega') - F(\omega)}$$

$$\omega_o = (1-t)v + t v'$$



Triangulation

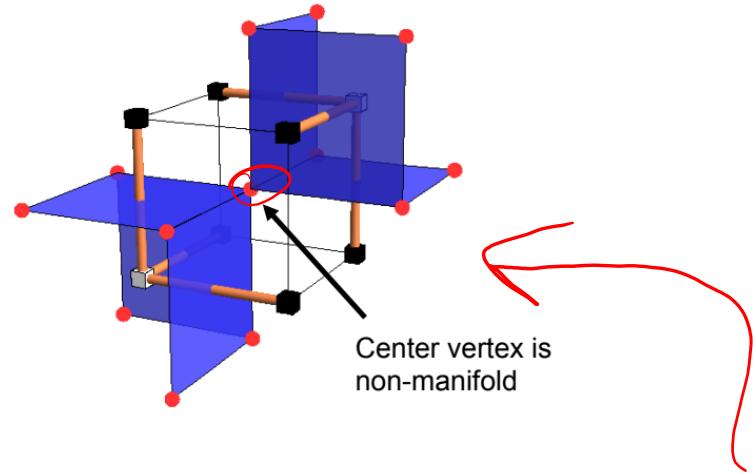
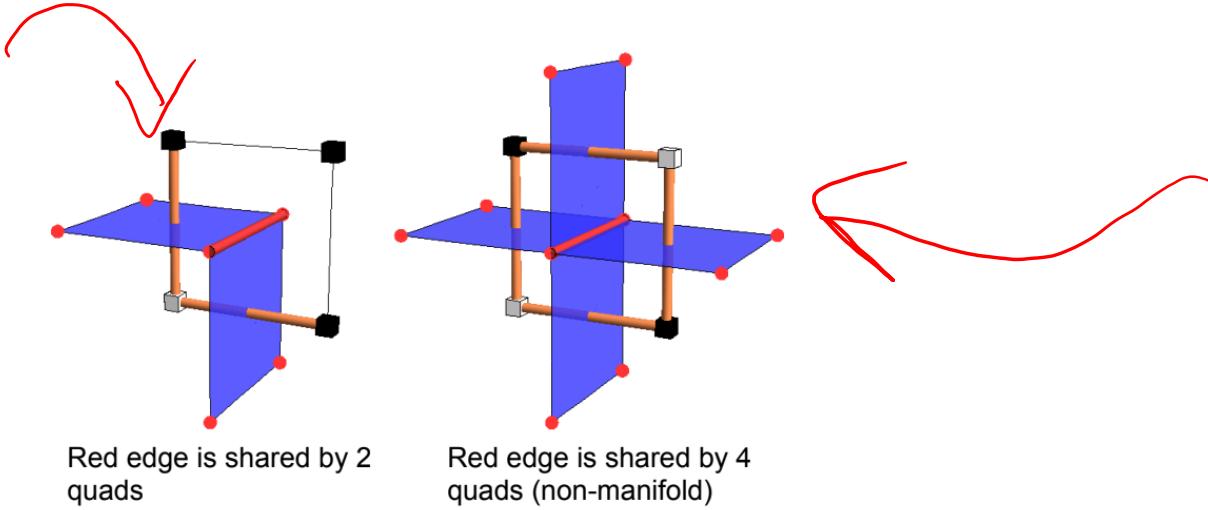
Can replace each quadrilateral around bipolar edge by two triangles.



To construct the isosurface patch with two triangles, use the diagonal that minimizes the maximum triangle angle.



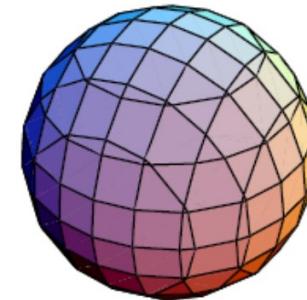
Non-manifold Cases



Comparison

Marching Cubes

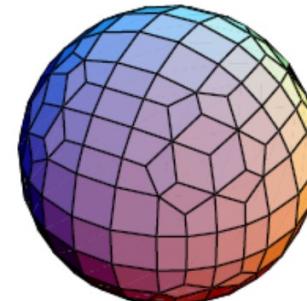
- Closed, manifold, and intersection-free
- Often generates thin and tiny polygons



Marching Cubes

Dual Contouring

- Closed and intersection-free
- Generates better-shaped polygons
- Can be non-manifold



Dual Contouring

Manifold Dual Marching Cubes

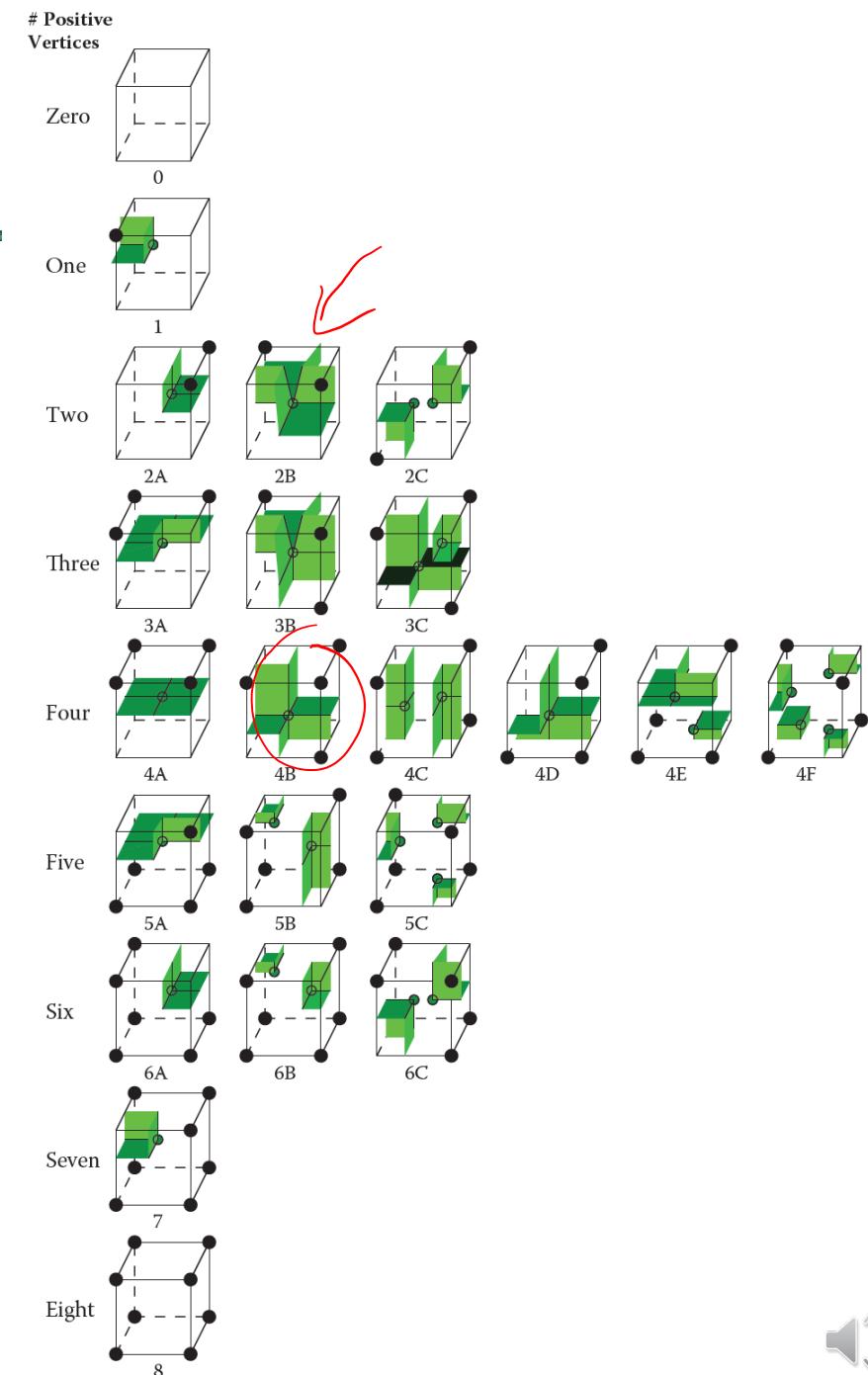
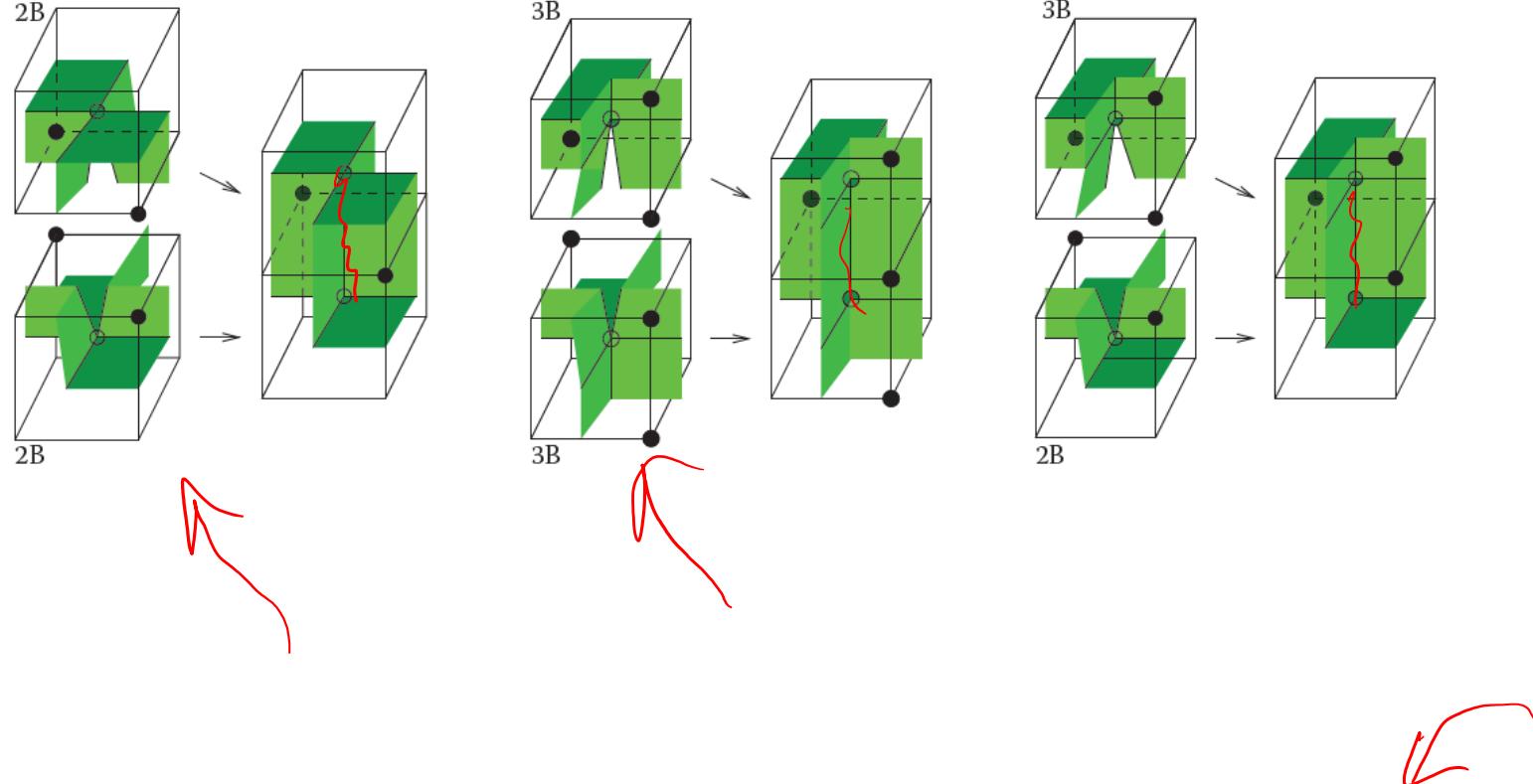
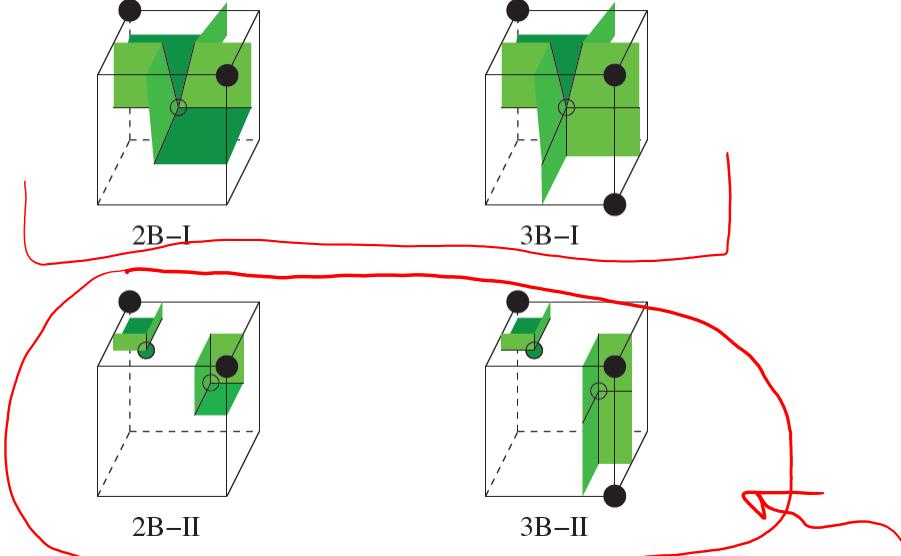


Table-based algorithm from:

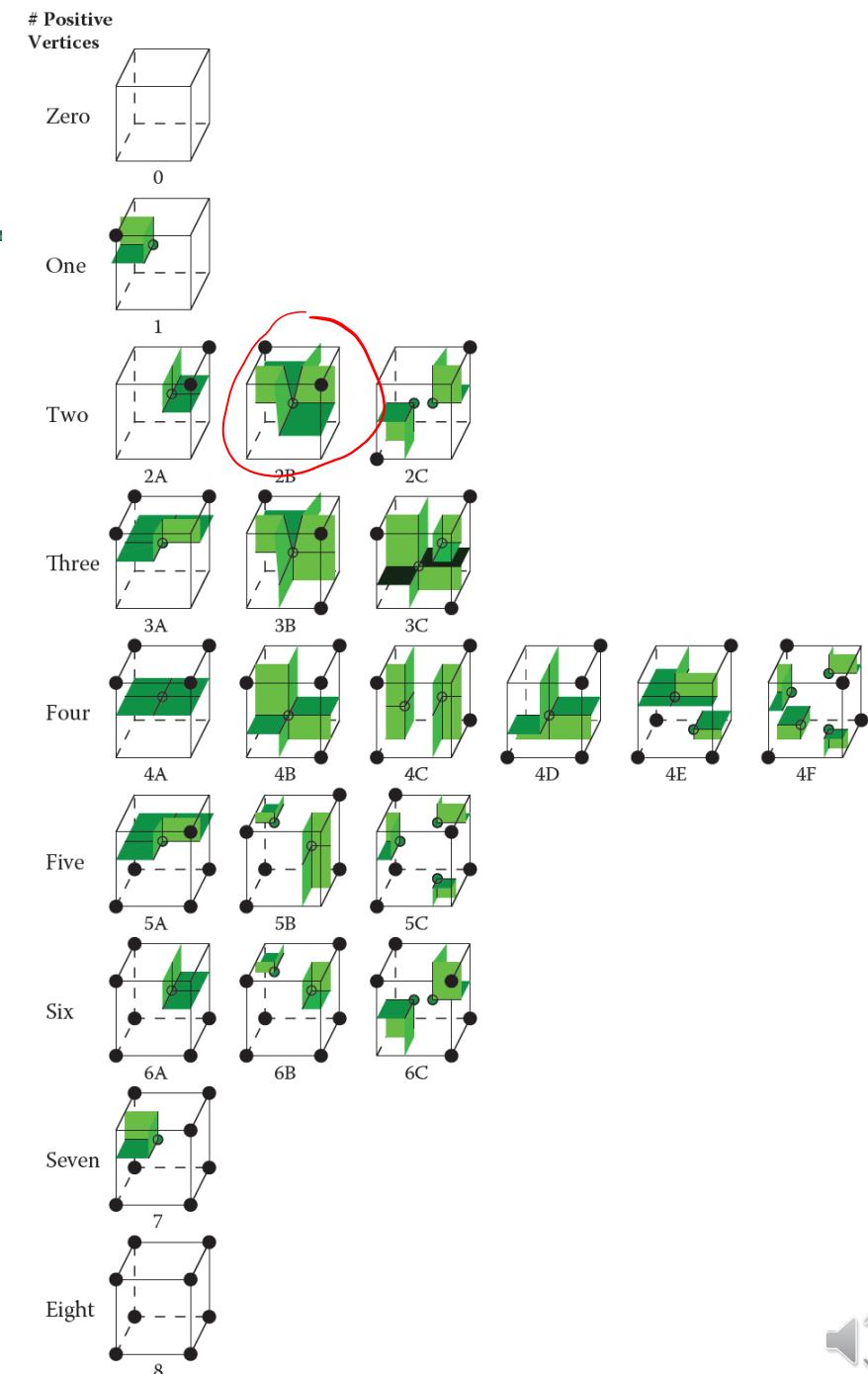
Nielson, G. M. (2004). Dual Marching Cubes. In *Proceedings of IEEE Visualization 2004*, pages 489–496, Los Alamitos, CA. IEEE Computer Society.



Manifold Dual Marching Cubes



Manifold Dual Marching Cubes avoids creating non-manifold edges by using isosurface patches 2B-II and 3B-II instead of 2B-I and 3B-I whenever two cubes with configurations 2B and 3B share their ambiguous facets.

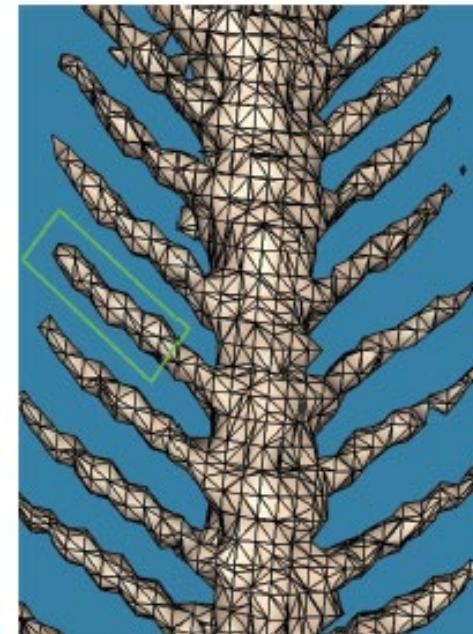


Manifold Dual Marching Cubes

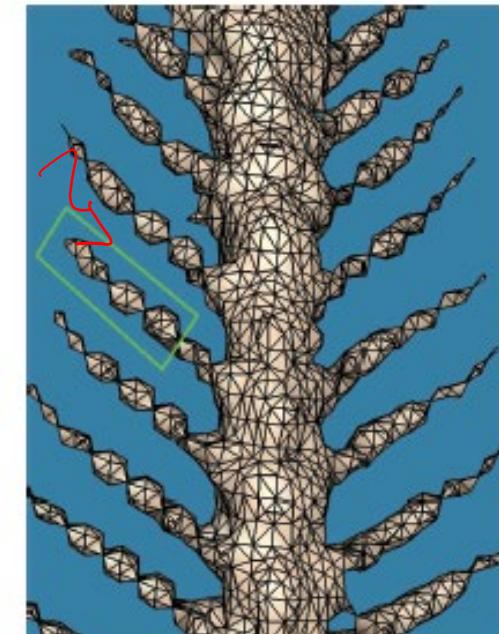
For other approaches....

S. Schaefer, T. Ju and J. Warren, "Manifold Dual Contouring," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 3, pp. 610-619, May-June 2007, doi: 10.1109/TVCG.2007.1012.

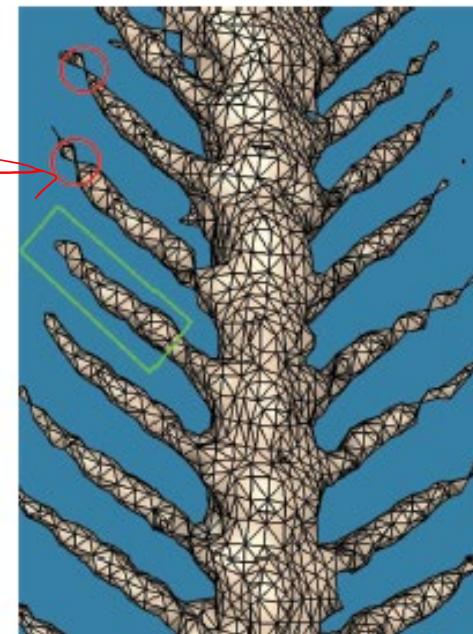
Tanweer Rashid, Sharmin Sultana, Michel A. Audette, "Watertight and 2-manifold Surface Meshes Using Dual Contouring with Tetrahedral Decomposition of Grid Cubes"



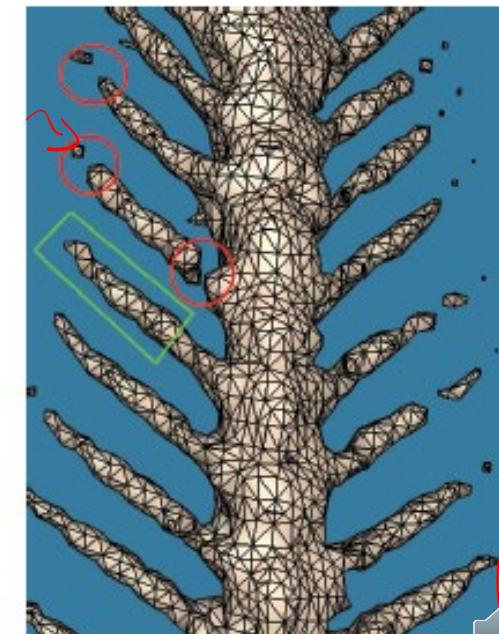
(a) MARCHING CUBES.



(b) SURFACE NETS.

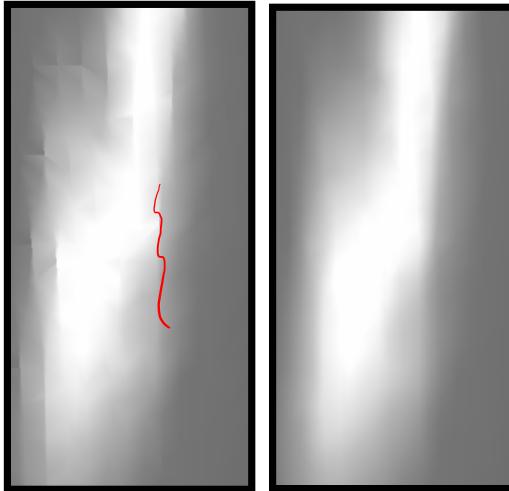


(c) DUAL MARCHING CUBES.

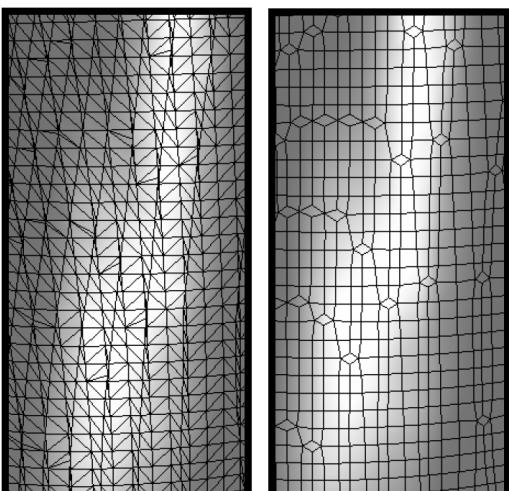


(d) MANIFOLD DUAL MC.

Comaparison with Marching Cubes



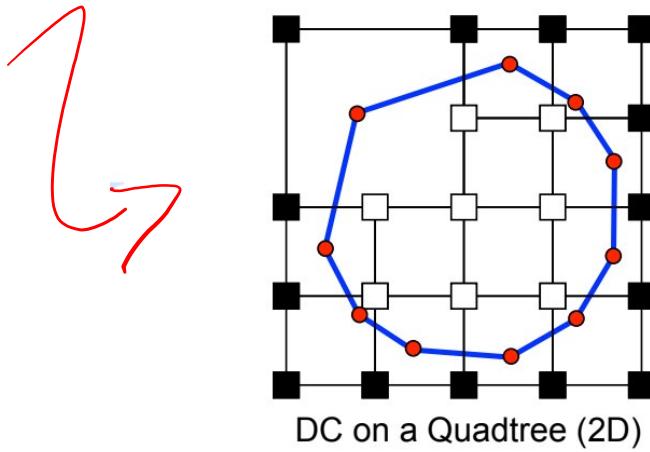
Can see fewer thin polygons in DMC surface



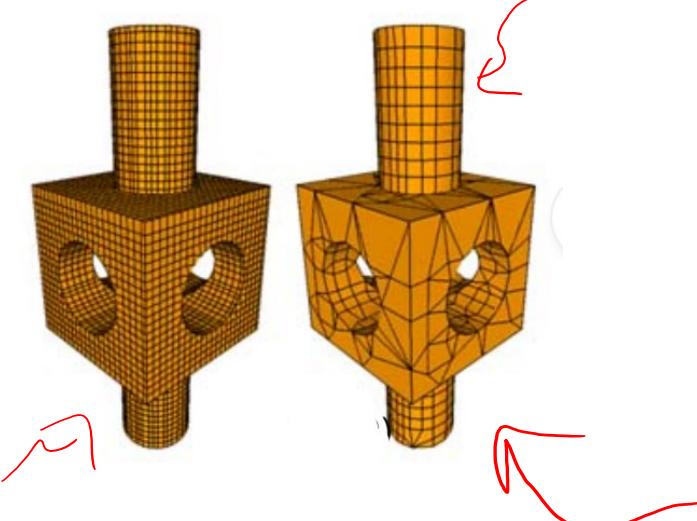
DMC generates smoother surface

- better able to match curvature

Extensions: Contouring on Adaptive Grids



DC on a Quadtree (2D)



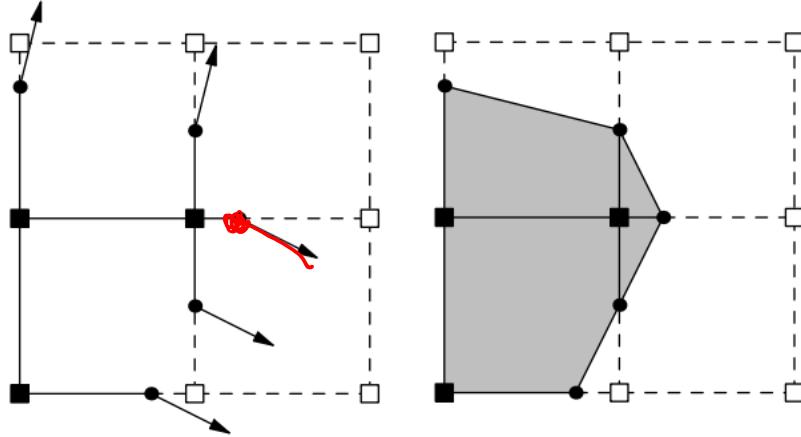
Benefits

Better use of polygon budget...fewer polygons in low curvature areas

Waste less time processing lots of small empty cubes

S. Schaefer and J. Warren, "Dual marching cubes: primal contouring of dual grids," *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.*, Seoul, South Korea, 2004

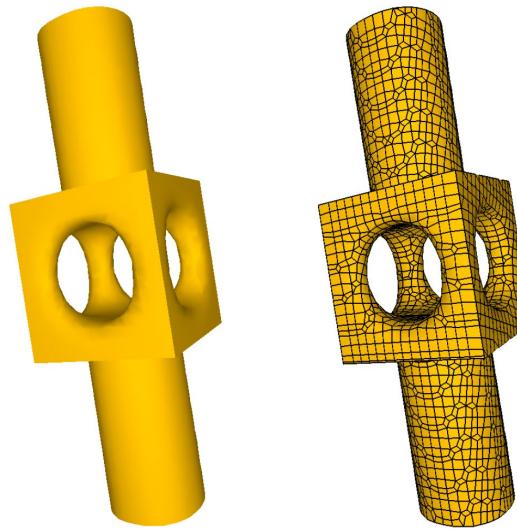
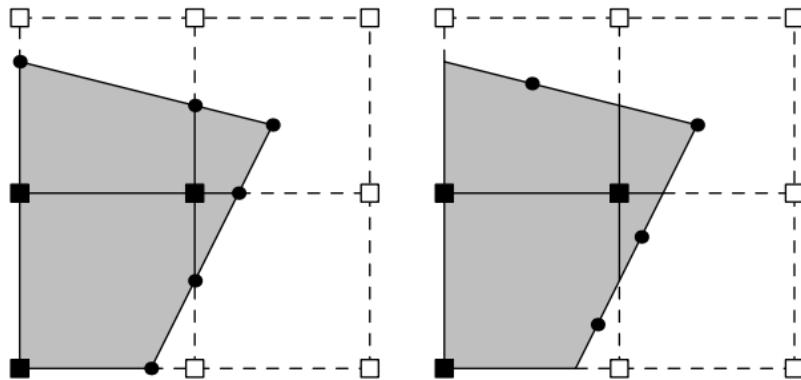
Dual Contouring on Hermite Data



Hermite data → Exact intersections and normal

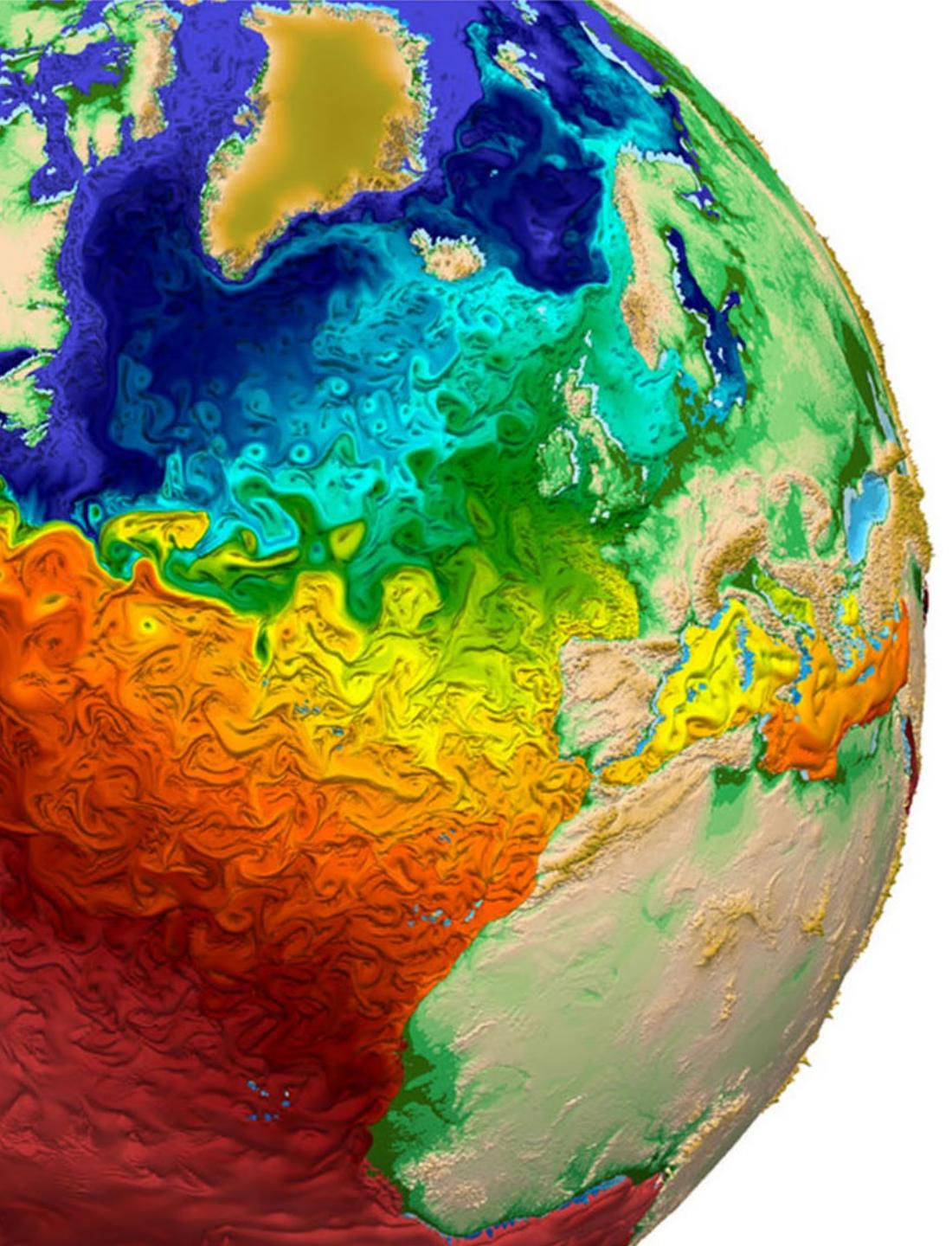
Used on data from implicit surfaces

Retains sharp features



Ju, T., Losasso, F., Schaefer, S., Warren, J.: Dual contouring on hermite data. In: ACM SIGGRAPH (2002)



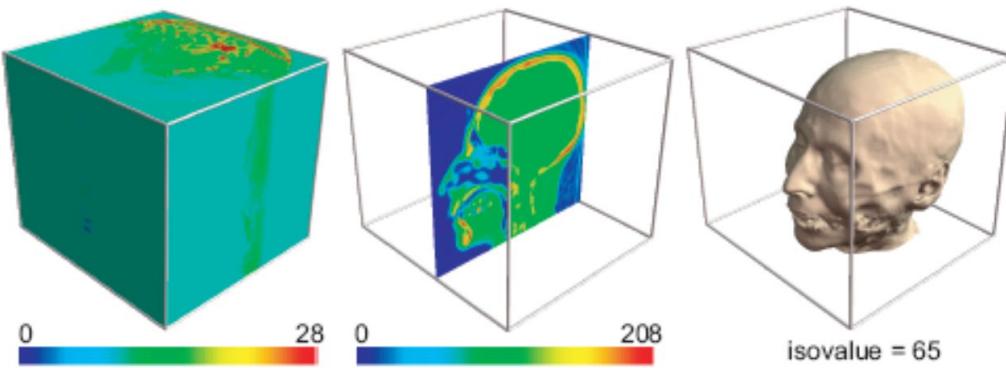


Volume Rendering

Fundamentals

Scientific Visualization
Professor Eric Shaffer

Scalar Field Visualization



Tools we have for investigating 3D volumes are limited...

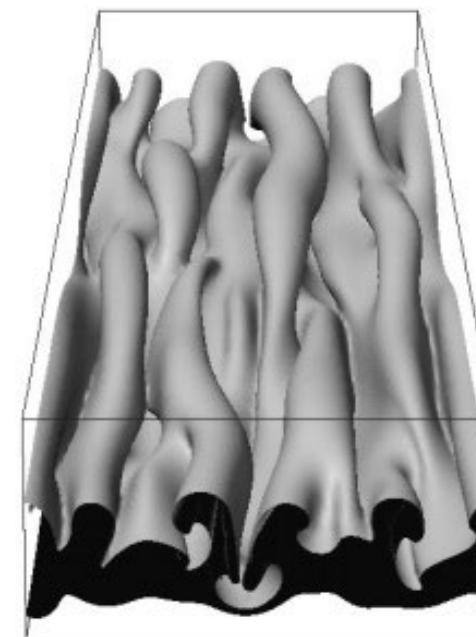
How can we better understand internal structure?

- 2D slices with pseudo-color
- Adjusting isosurface values

Scalar volume: $f : D \subset \mathbb{R}^3 \rightarrow \mathbb{R}$
 $(x, y, z) \mapsto f(x, y, z)$

The Problem With Isosurfaces

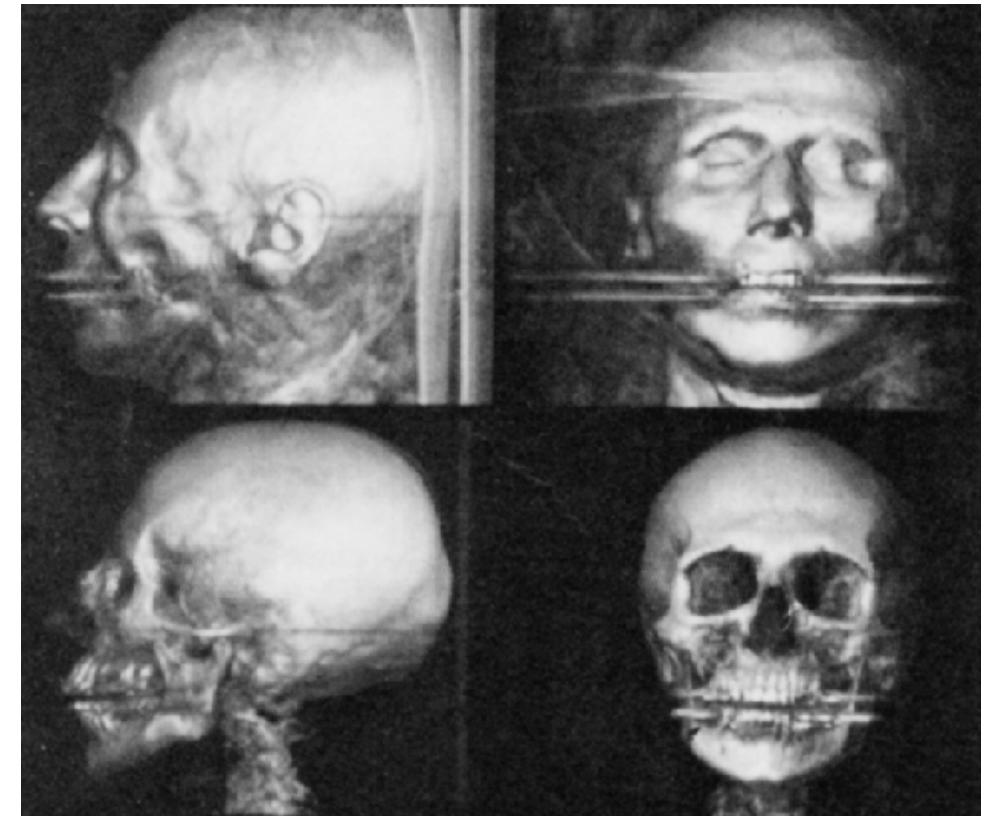
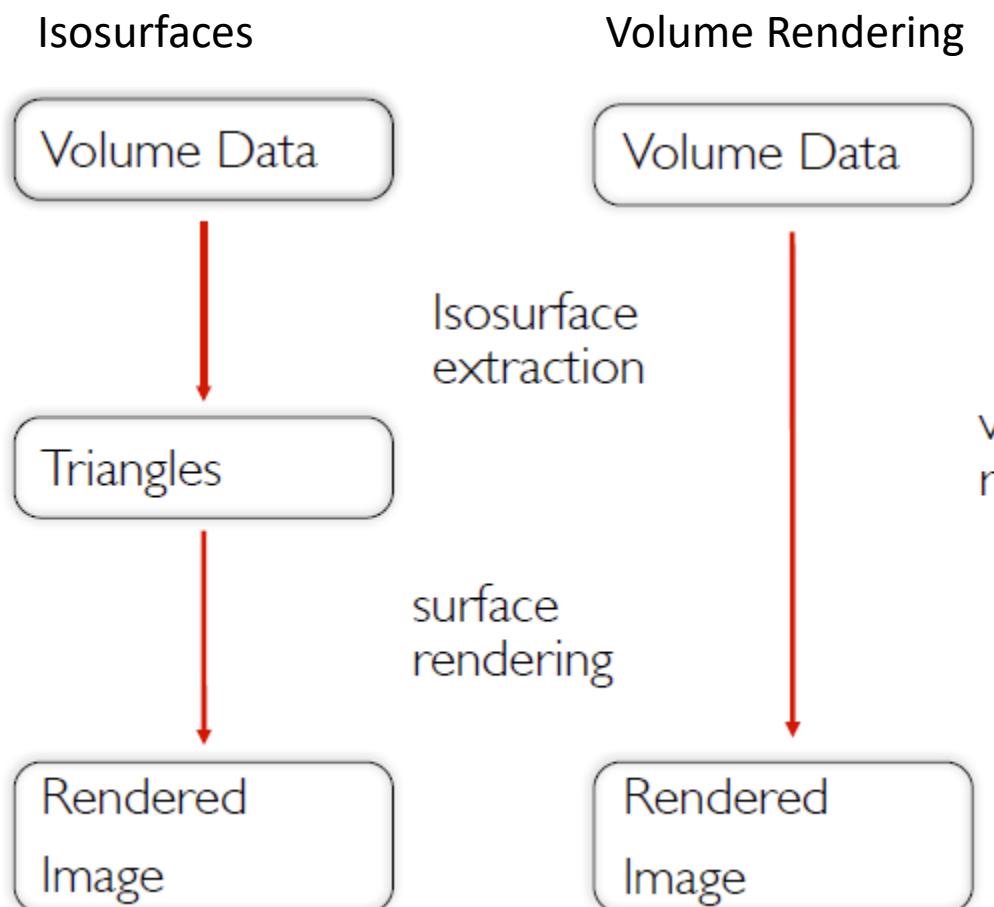
- Delineate hard boundaries which may not accurately represent data
 - Possibly transitions between values are much smoother
- Using multiple isosurface values to investigate the volume problematic
 - Time consuming
 - Can't see everything all at once
 - Sampling may miss important features



Volume Rendering

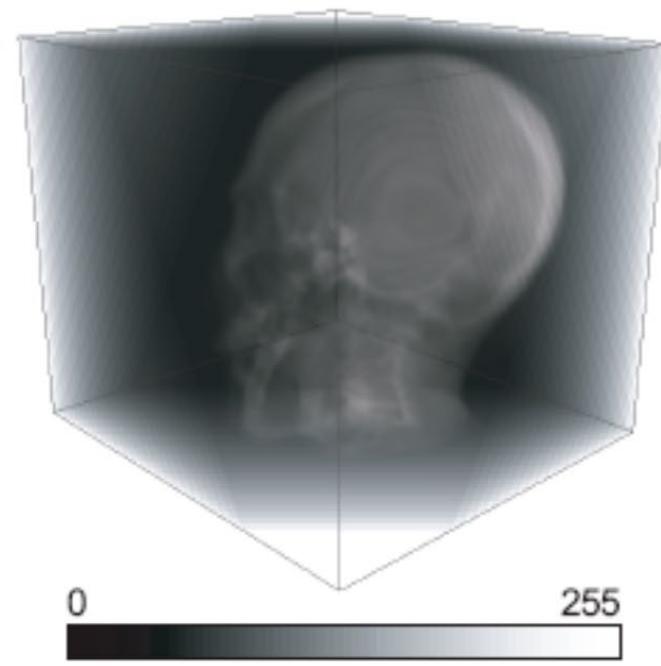
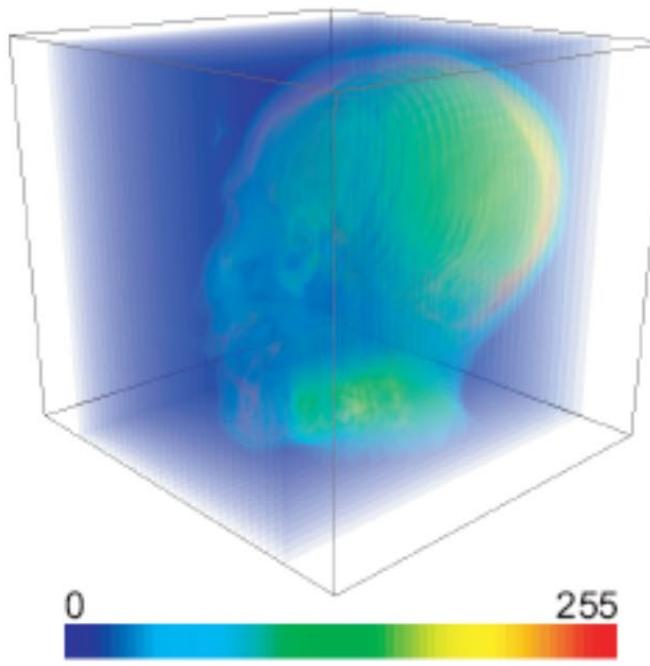
Voxel = volume element...like a grid cell

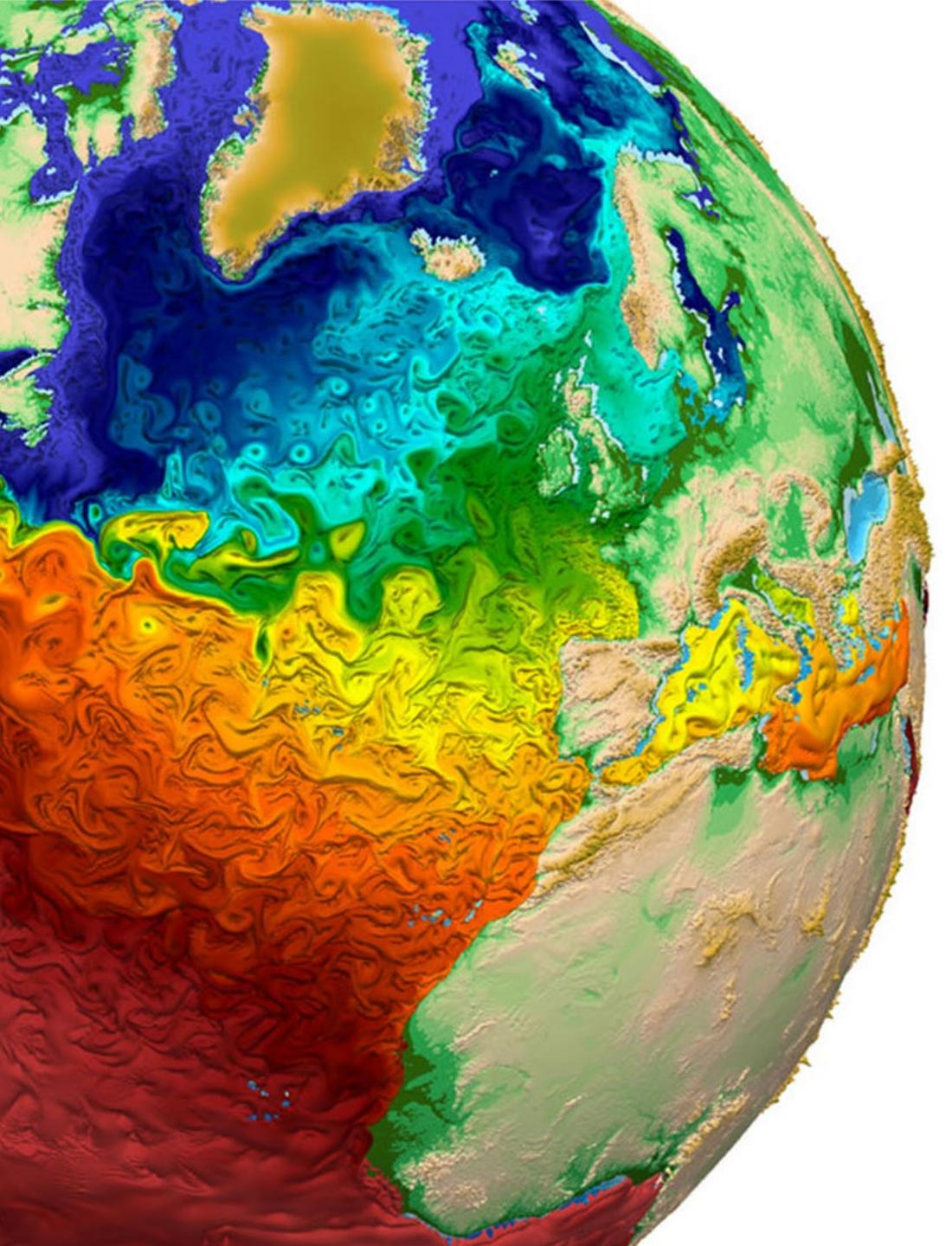
- "Every voxel contributes to image" Marc Levoy, 1988 *"Display of Surfaces from Volume Data"*



What is Volume Rendering

Any rendering process which maps from volume data to an image without introducing **binary distinctions** / intermediate geometry - Xavier Tricoche





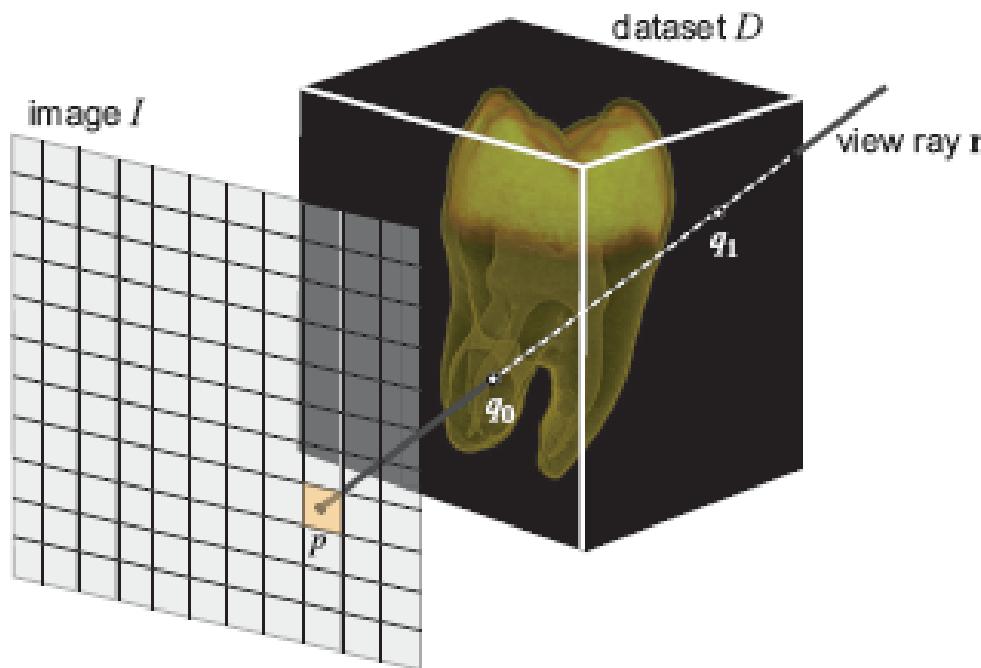
Volume Rendering

Ray Casting

Scientific Visualization
Professor Eric Shaffer

Basic Idea

- The data is considered to represent a semi-transparent light-emitting medium
- Approaches are based on the laws of physics (light emission, absorption, scattering)
- Model transport of light along rays through an image plane



Typically achieved through
ray-casting or similar
technique

Uses color and opacity to
visualize data

Volume Rendering Integral

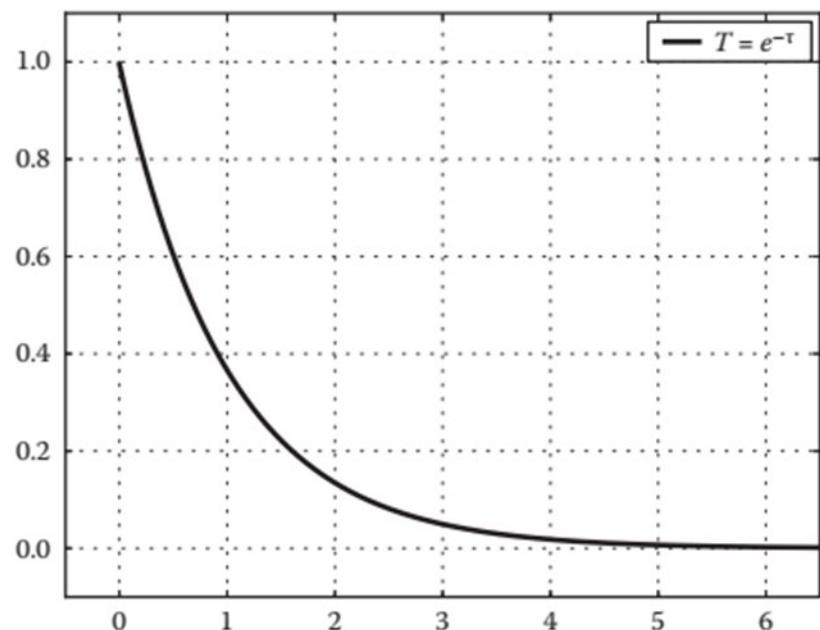
A diagram showing a horizontal black line segment representing a ray, starting at point s_0 and ending at point s . A red arrow points to the right above the ray. Below the ray is a wavy red line representing a color gradient. Handwritten text in red points to the wavy line: "s: scalar value at x" points to the peak of the wavy line; "c: color associated with value s" points to the left end of the wavy line; "x: position along ray R" points to the right end of the wavy line; and " μ : density-opacity associated with that value" points to the middle of the wavy line.

$$c(\mathbf{R}) = \int_0^D c(s(x(t))) \mu(s(x(t))) e^{-\int_0^t \mu(s(u)) du} dt$$

Physics – Beer's Law

$$c(\mathbf{R}) = \int_0^D c(s(x(t)))\mu(s(x(t))) e^{-\int_0^t \mu(s(u))du} dt$$

cumulative
absorption

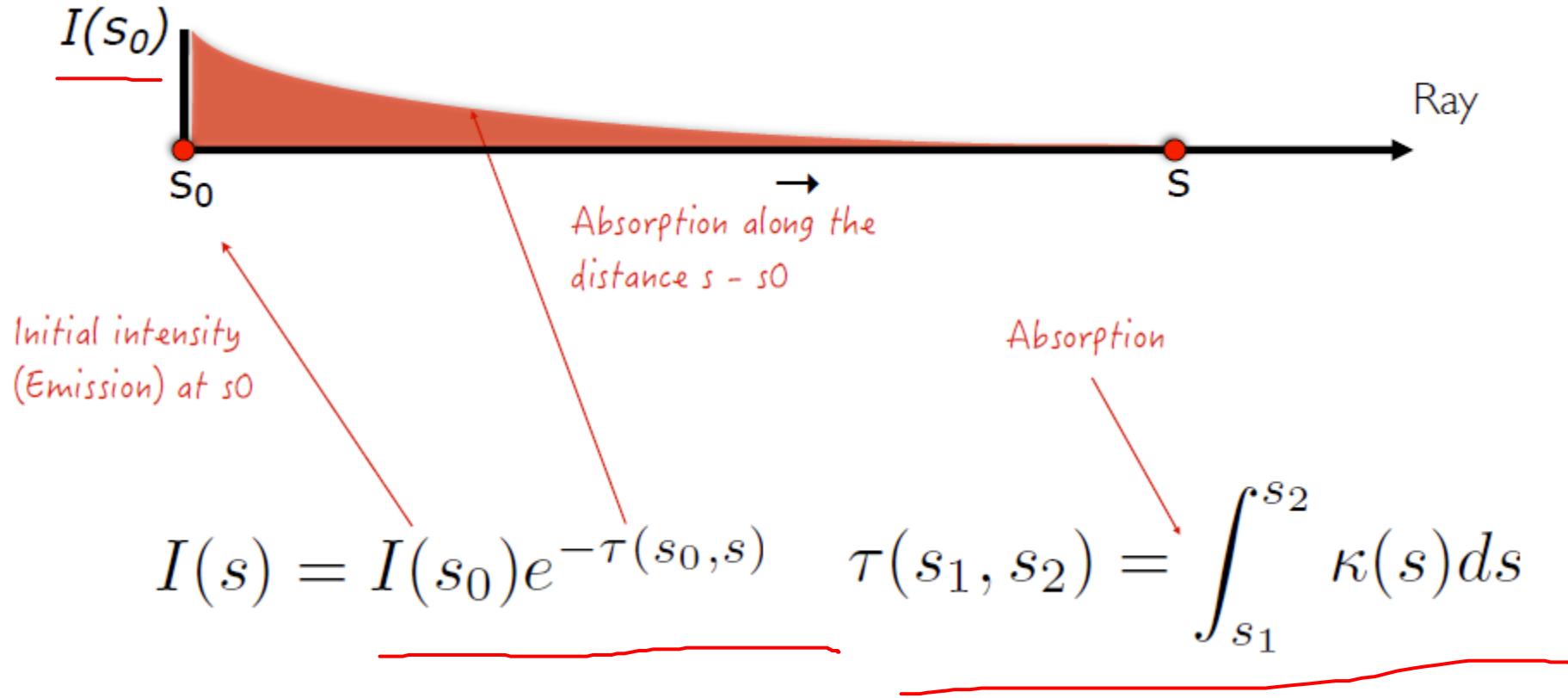


Transmittance T :
How much light can pass between two points in a medium?

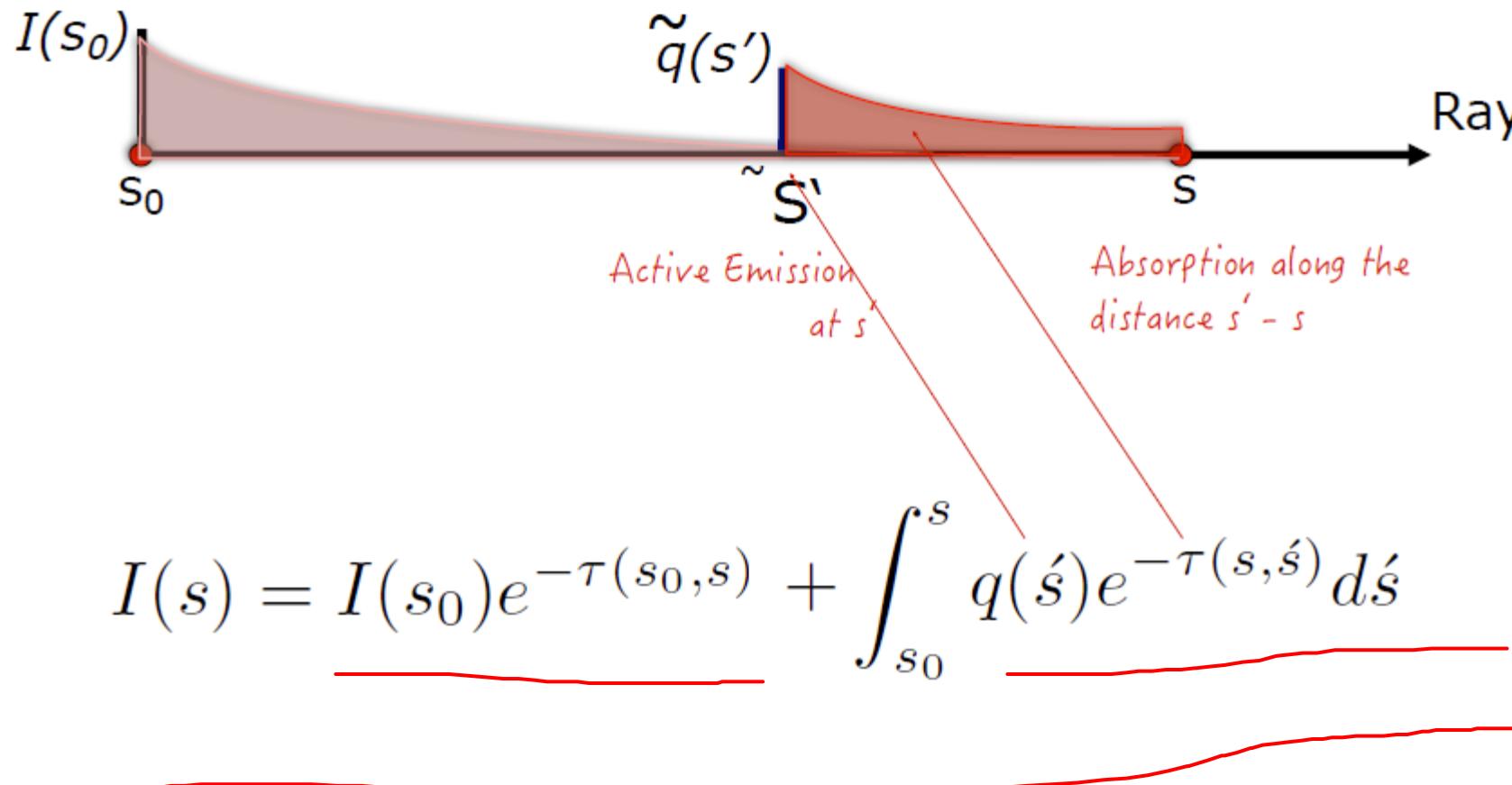
$$T = e^{-\tau}$$

τ Optical thickness...a measure of transparency

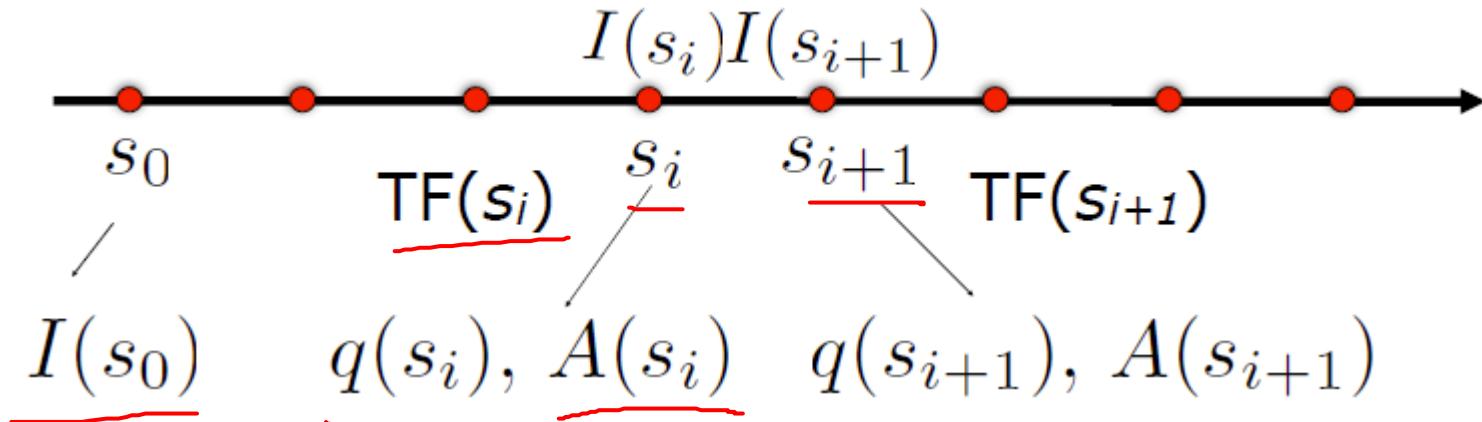
Emission and Absorption Along the Ray



Emission and Absorption Along the Ray

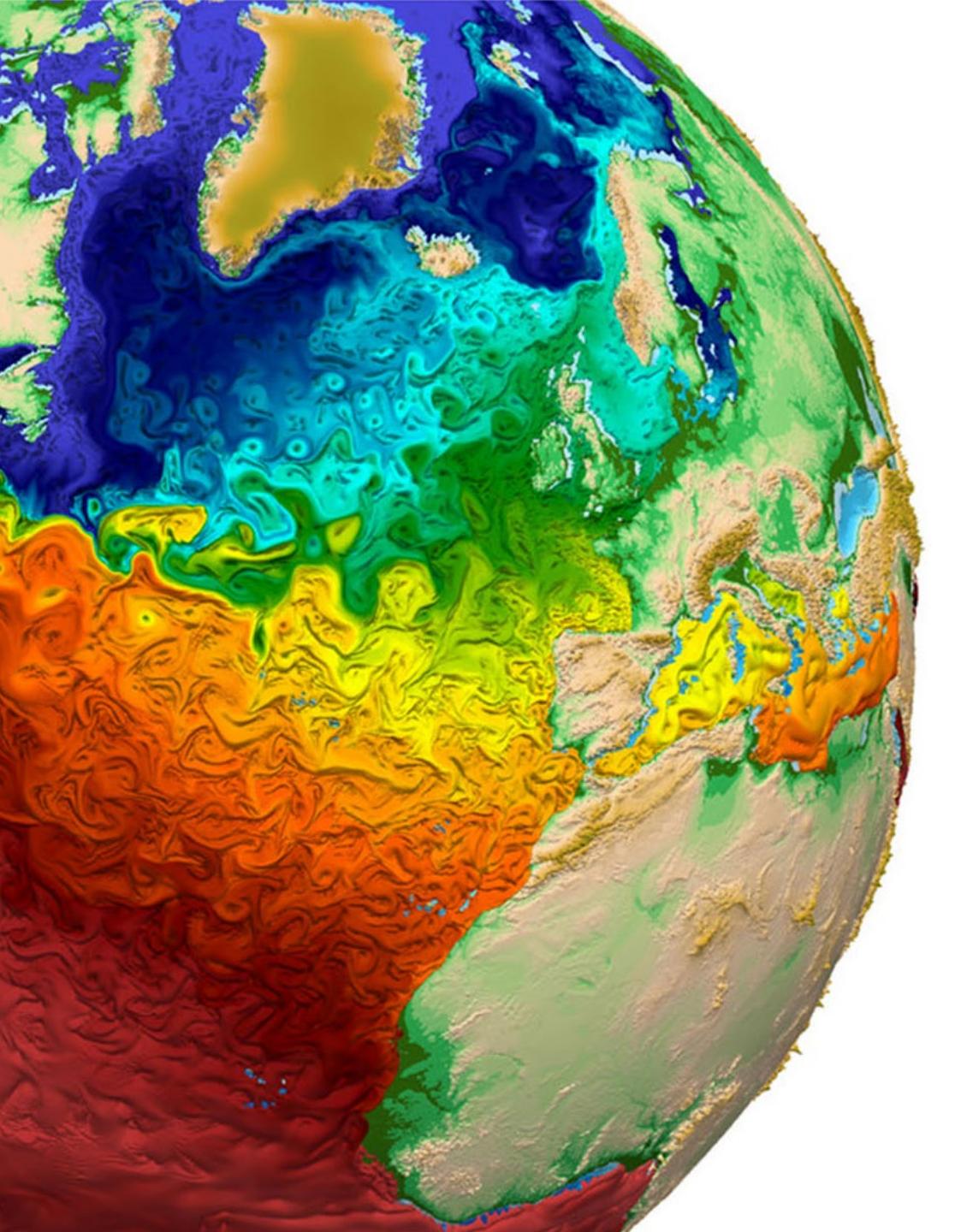


Discrete Approximation



Back-to-front Compositing with $\alpha = \underline{A(s_{i+1})}$

$$\begin{aligned} I(s_{i+1}) &= \underline{\alpha q(s_{i+1})} + \underline{(1 - \alpha)I(s_i)} \\ &= \underline{q(s_{i+1})} \text{ OVER } \underline{I(s_i)} \end{aligned}$$



Volume Rendering

Volumetric Shading

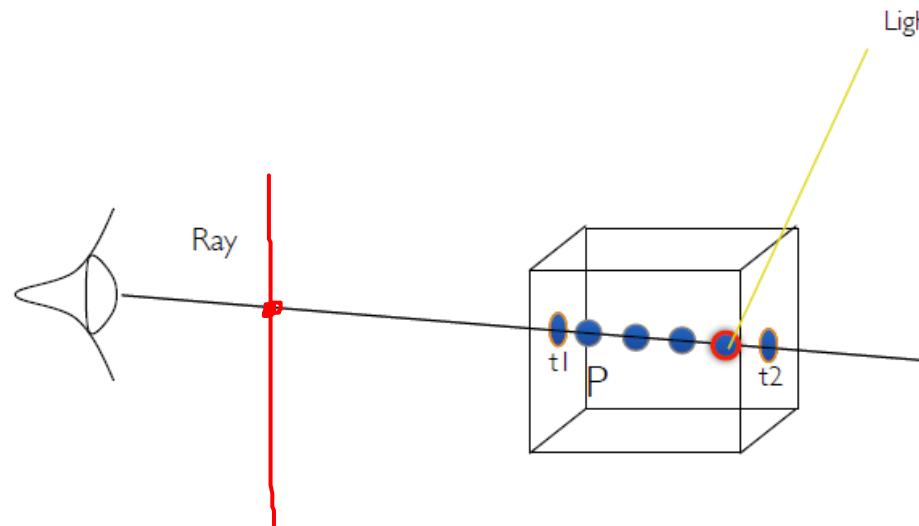
Scientific Visualization
Professor Eric Shaffer

Basic Idea

Shading allows viewers to perceive 3D shape and structure

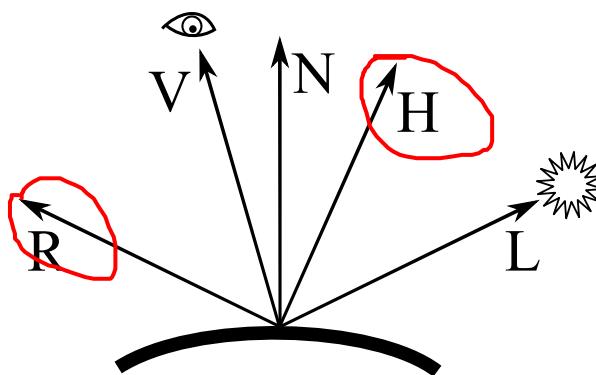
Can apply shading to the volumetric data

Use it to modify light emitted at a sample point



The Blinn-Phong Reflection Model

$$I_p = \underbrace{k_a i_a}_{\text{diffuse}} + \sum_{m \in \text{lights}} \left(k_d (\hat{L}_m \cdot \hat{N}) i_{m,d} + k_s (\cancel{\hat{R}_m} \cdot \hat{V})^\alpha i_{m,s} \right)$$



The material values are generated by the transfer function

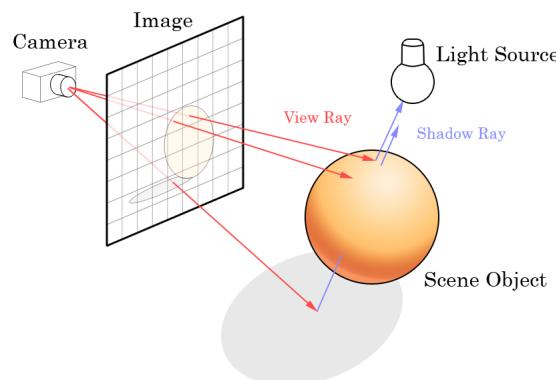
- k_a and $k_{m,s}$ and $k_{m,d}$
- Customary to use $(1,1,1)$ for i_a and $i_{m,s}$ and $i_{m,d}$

L is easy to compute given L_{pos} the light position: $L = \frac{L_{pos}-P}{\|L_{pos}-P\|}$

V can be computed using eyepoint E behind view plane: $V = \frac{E-P}{\|E-P\|}$

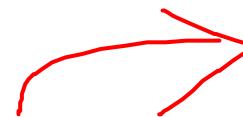
Can substitute H for R in the equation...with $H = \frac{V+L}{2}$

Need to compute the normal N.....

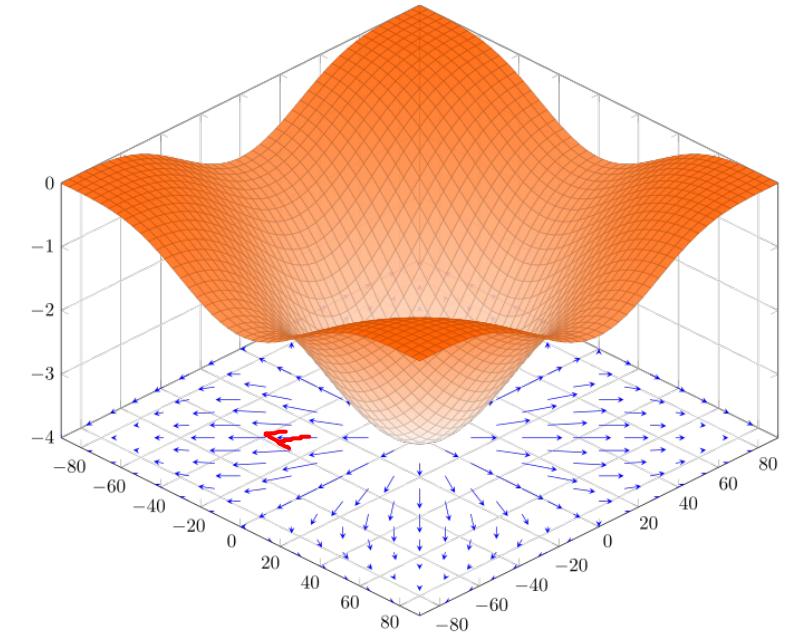


The Gradient

$$\nabla f(p) = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right\rangle$$



Points in the direction of most rapid ascent in function values



Gradient of a function is normal to a contour of a function

Can use gradient as a normal...imagine point on the ray has been sampled from an isosurface

Approximating the Gradient

If the gradient cannot be determined analytically, can approximate it numerically
e.g when function f is unknown

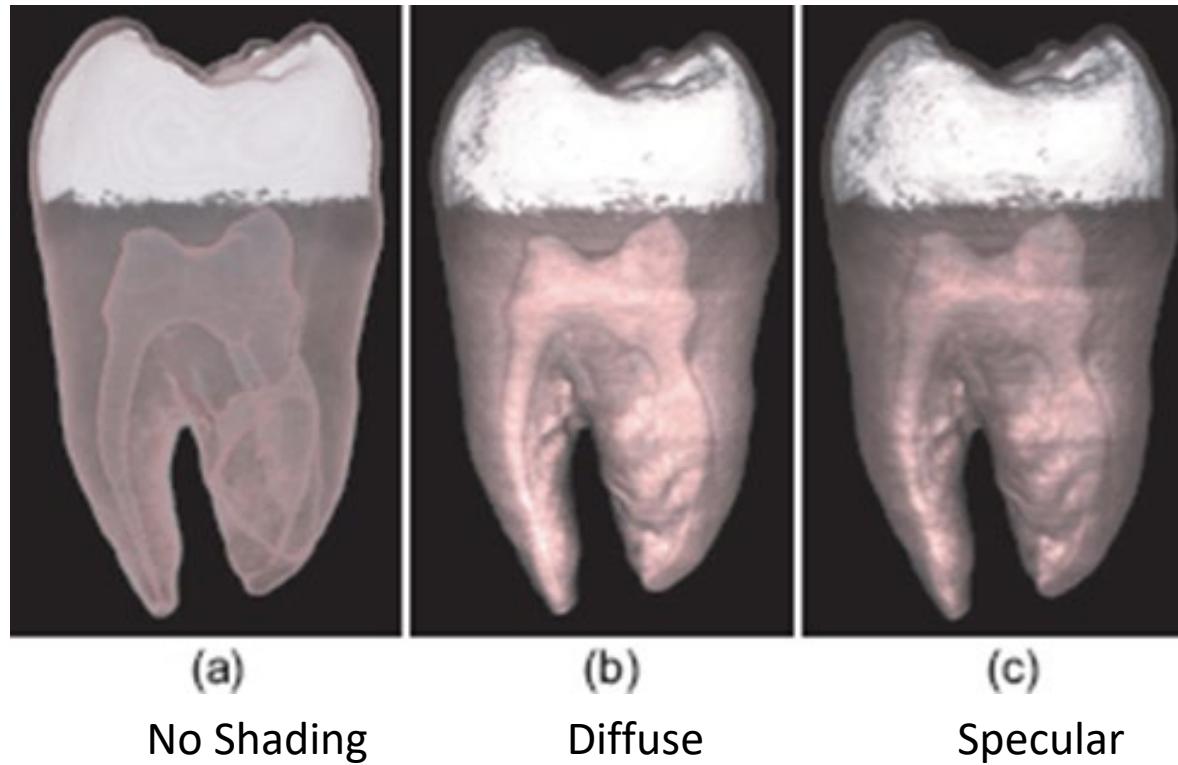
central difference formula

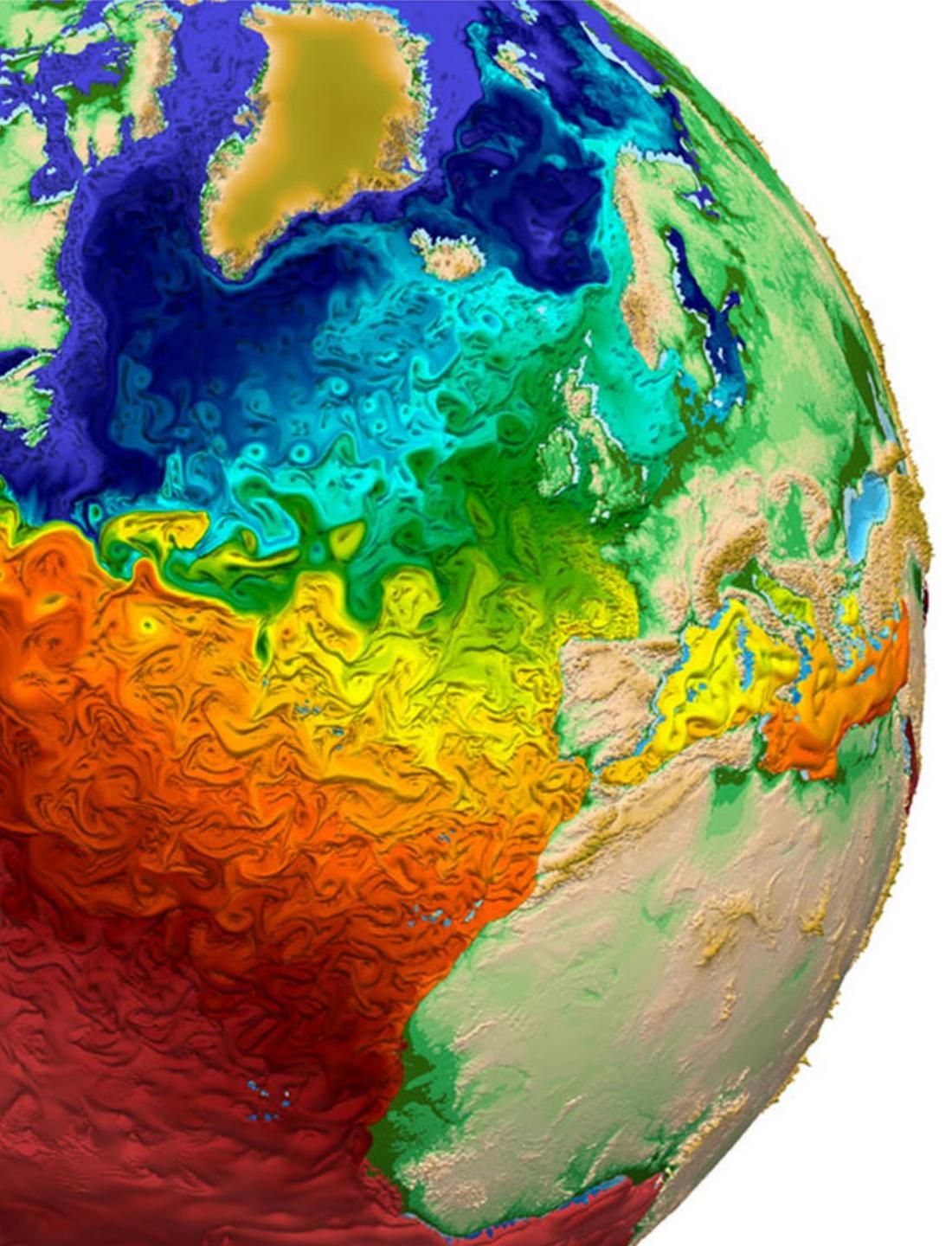
$$\nabla f(p) \approx \left\langle \frac{f(x + h, y, z) - f(x - h, y, z)}{2h}, \frac{f(x, y + h, z) - f(x, y - h, z)}{2h}, \frac{f(x, y, z + h) - f(x, y, z - h)}{2h} \right\rangle$$



Use trilinear interpolation
to sample scalar field
values

Volumetric Shading



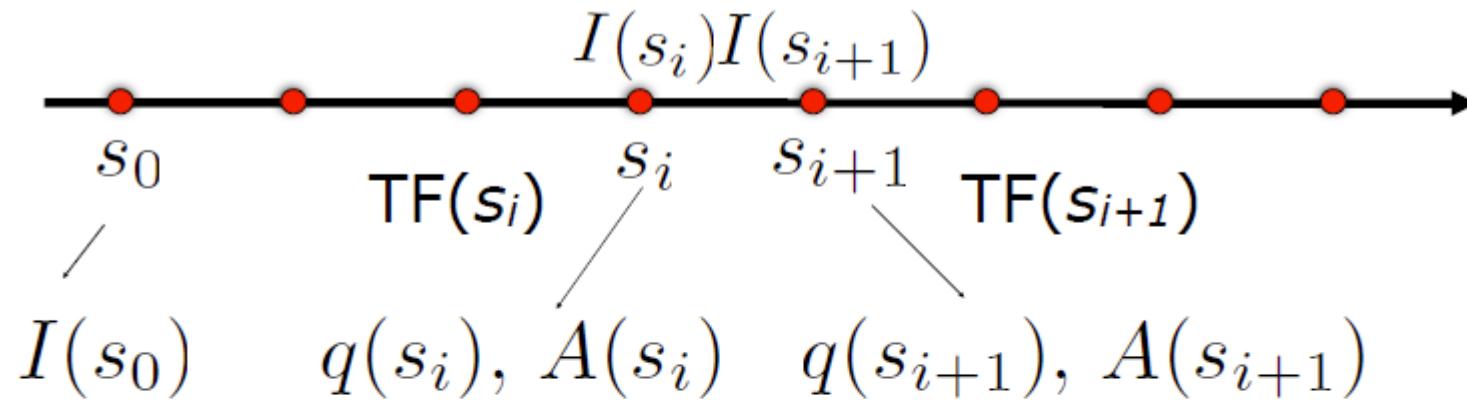


Volume Rendering

Compositing

Scientific Visualization
Professor Eric Shaffer

Approximating the Volume Rendering Integral

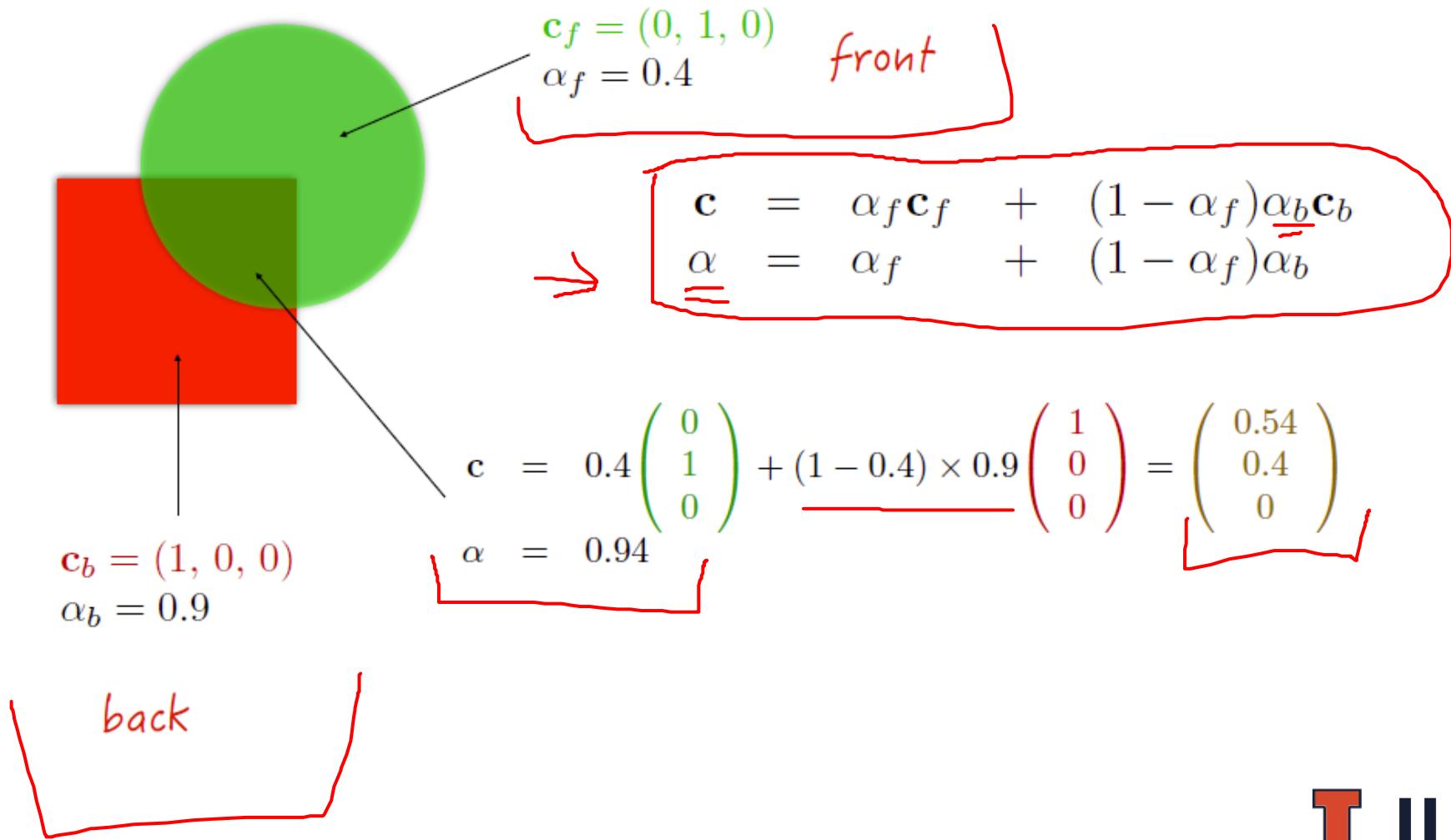


Back-to-front Compositing with $\alpha = A(s_{i+1})$

$$\begin{aligned} \underline{I(s_{i+1})} &= \underline{\alpha q(s_{i+1})} + \underline{(1 - \alpha)I(s_i)} \\ &= \underline{q(s_{i+1})} \text{ OVER } \underline{I(s_i)} \end{aligned}$$

The equation shows the back-to-front compositing formula. The terms $\alpha q(s_{i+1})$ and $(1 - \alpha)I(s_i)$ are underlined in red. The final result $q(s_{i+1})$ OVER $I(s_i)$ is also underlined in red.

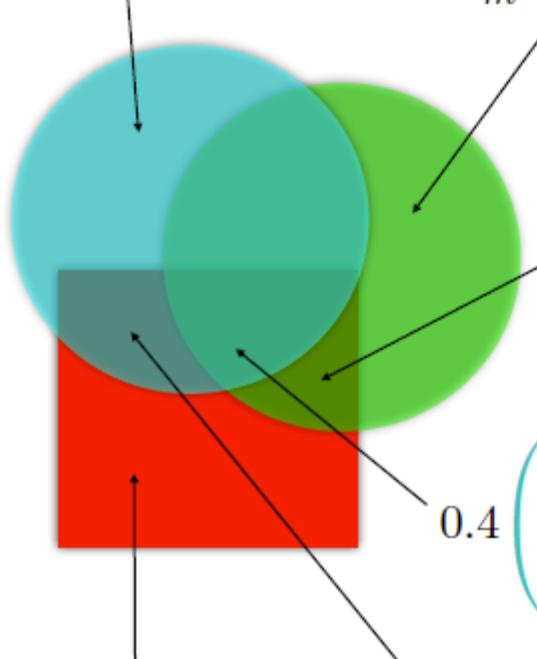
The Over Operator



Order of Computation

$$\mathbf{c}_f = (0, 1, 1)$$

$$\alpha_f = 0.4$$



$$\mathbf{c}_m = (0, 1, 0)$$

$$\alpha_m = 0.4$$

$$\mathbf{c} = \alpha_f \mathbf{c}_f + (1 - \alpha_f) \alpha_b \mathbf{c}_b$$

$$\alpha = \alpha_f + (1 - \alpha_f) \alpha_b$$

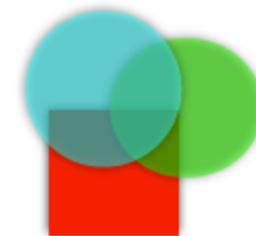
$$0.4 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + (1 - 0.4) \times 0.9 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.54 \\ 0.4 \\ 0 \end{pmatrix}$$



$$\mathbf{c}_b = (1, 0, 0)$$

$$\alpha_b = 0.9$$

$$0.4 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + (1 - 0.4) \begin{pmatrix} 0.54 \\ 0.4 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.324 \\ 0.64 \\ 0.4 \end{pmatrix}$$

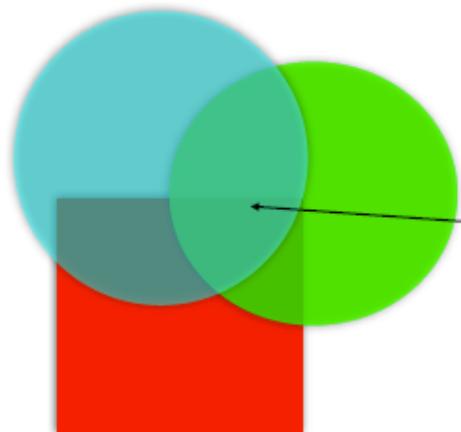


$$0.4 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + (1 - 0.4) \times 0.9 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.54 \\ 0.4 \\ 0.4 \end{pmatrix}$$

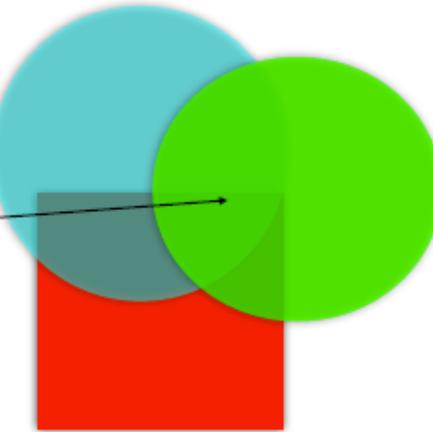
Order of Computation

$$\begin{aligned}\mathbf{c} &= \alpha_f \mathbf{c}_f + (1 - \alpha_f) \alpha_b \mathbf{c}_b \\ \alpha &= \alpha_f + (1 - \alpha_f) \alpha_b\end{aligned}$$

Order Matters!

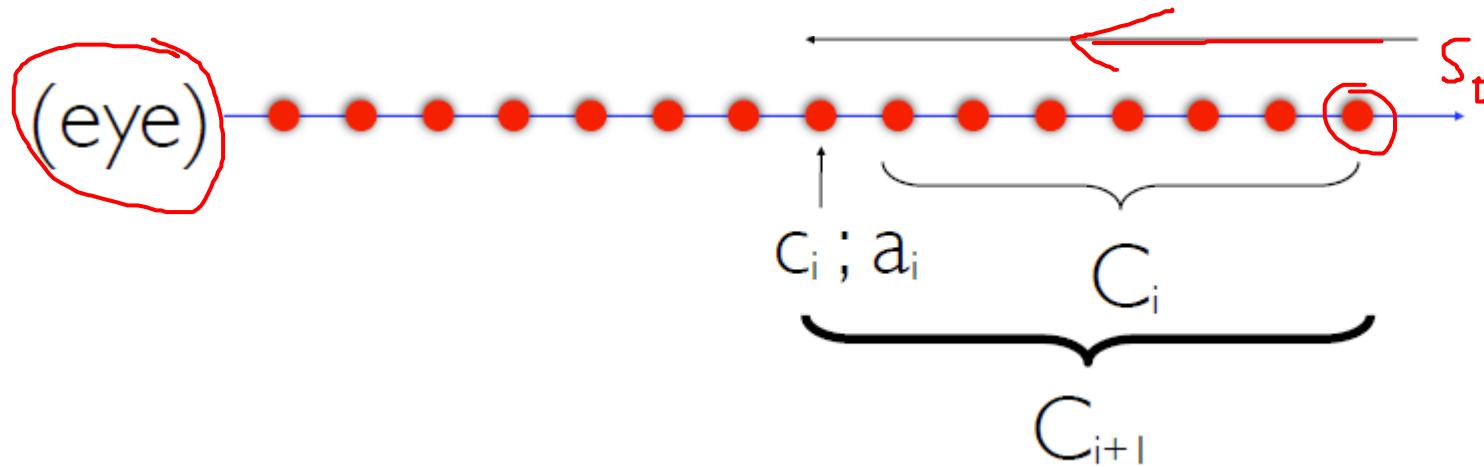


$$\begin{aligned}\mathbf{c} &= (0.324, 0.64, 0.4) \\ \alpha &= 0.964\end{aligned}$$



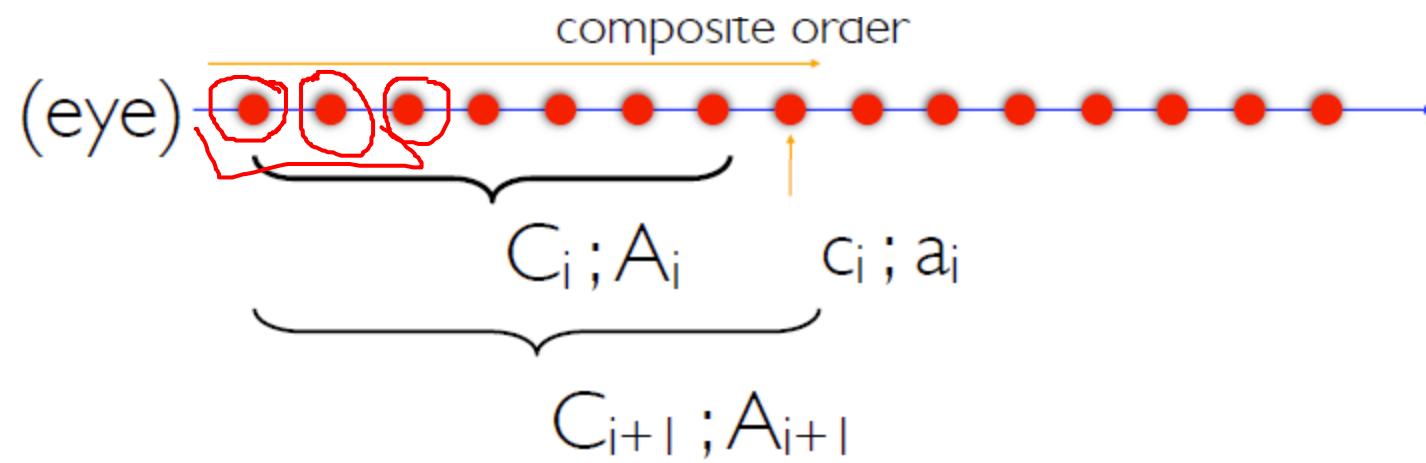
$$\begin{aligned}\mathbf{c} &= (0.324, 0.64, 0.24) \\ \alpha &= 0.964\end{aligned}$$

Back to Front



$$C_{i+1} = a_i c_i + (1-a_i) C_i$$

Front to Back



$$C_{i+1} = C_i + (1 - A_i)a_i c_i$$
$$A_{i+1} = A_i + (1 - A_i)a_i$$

Order of Composition

Back to Front

straightforward use of over operator

intuitively backwards?

Front to Back

intuitively right?

not simple over operator

facilitates early ray termination

Pre-multiplied Alpha

With pre-multiplied alpha, a color value is given by $c = (\alpha r, \alpha g, \alpha b, \alpha)$

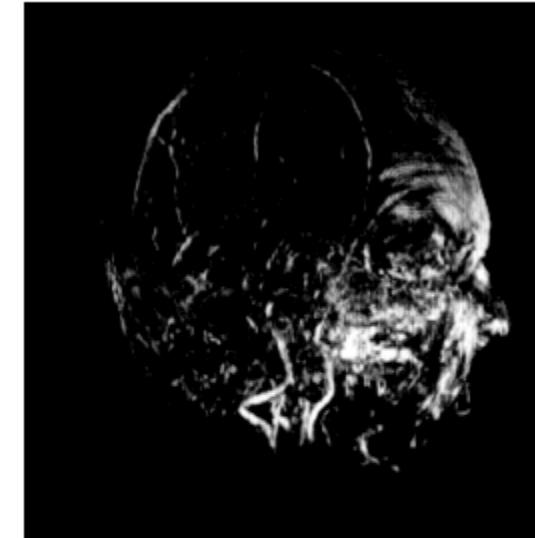
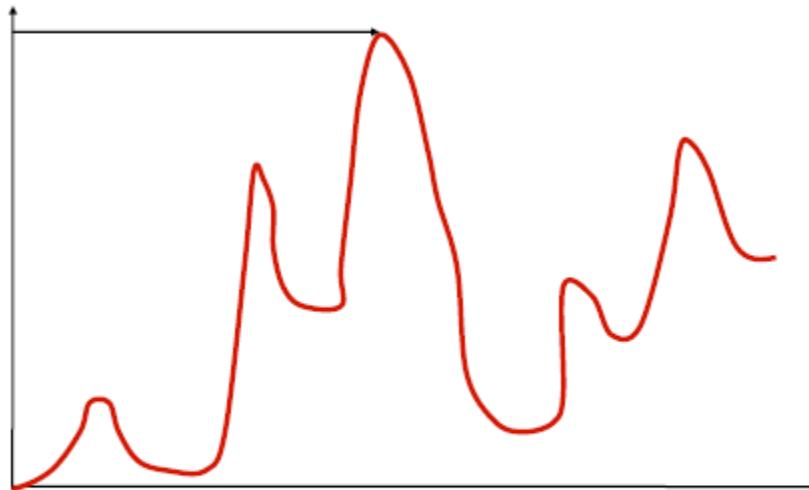
- Sometimes called *associated alpha*
- *Unassociated alpha* = non-pre-multiplied alpha
- **These two versions of alpha will not give the same results for all operations!**
- Can blend using the over operator and pre-multiplied alpha

$$\underline{\mathbf{c}_o} = \underline{\mathbf{c}'_s} + (1 - \alpha_s) \underline{\mathbf{c}_d}$$

- Here $\mathbf{c}' = (\alpha r, \alpha g, \alpha b)$
- Again, \mathbf{c}_d is assumed to be opaque

Alternative to Compositing: Maximum Intensity Projection

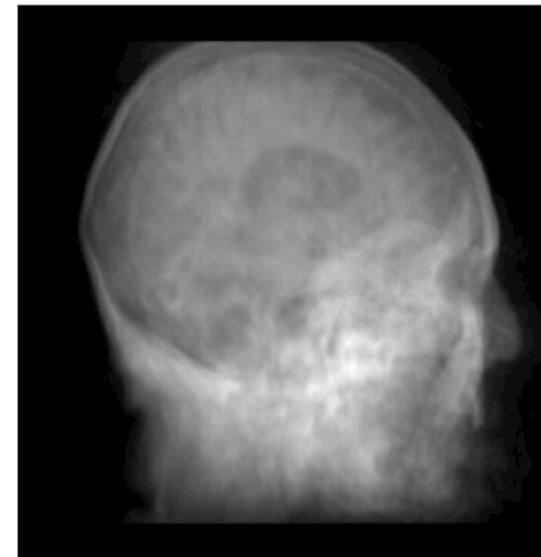
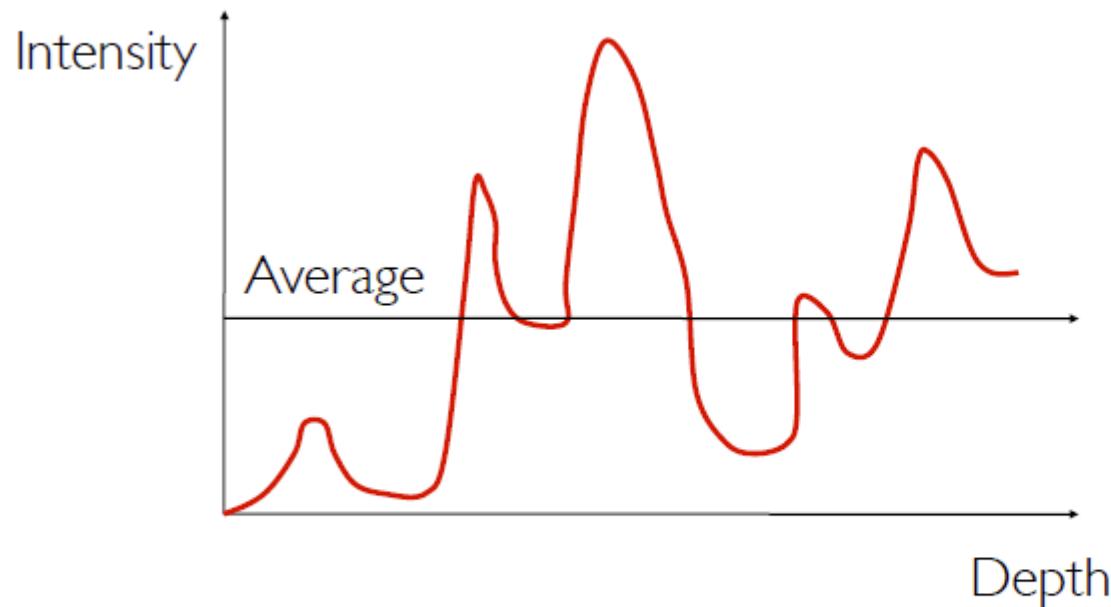
Max Intensity



Maximum Intensity Projection
Magnetic Resonance Angiogram

$$I(p) = f(\max(s(t)))$$

Alternative to Compositing: Average Intensity Projection



$$I(p) = f \left(\frac{\int_{t=0}^T s(t) dt}{T} \right)$$

Analogous to an x-ray

Synthetic Reprojection

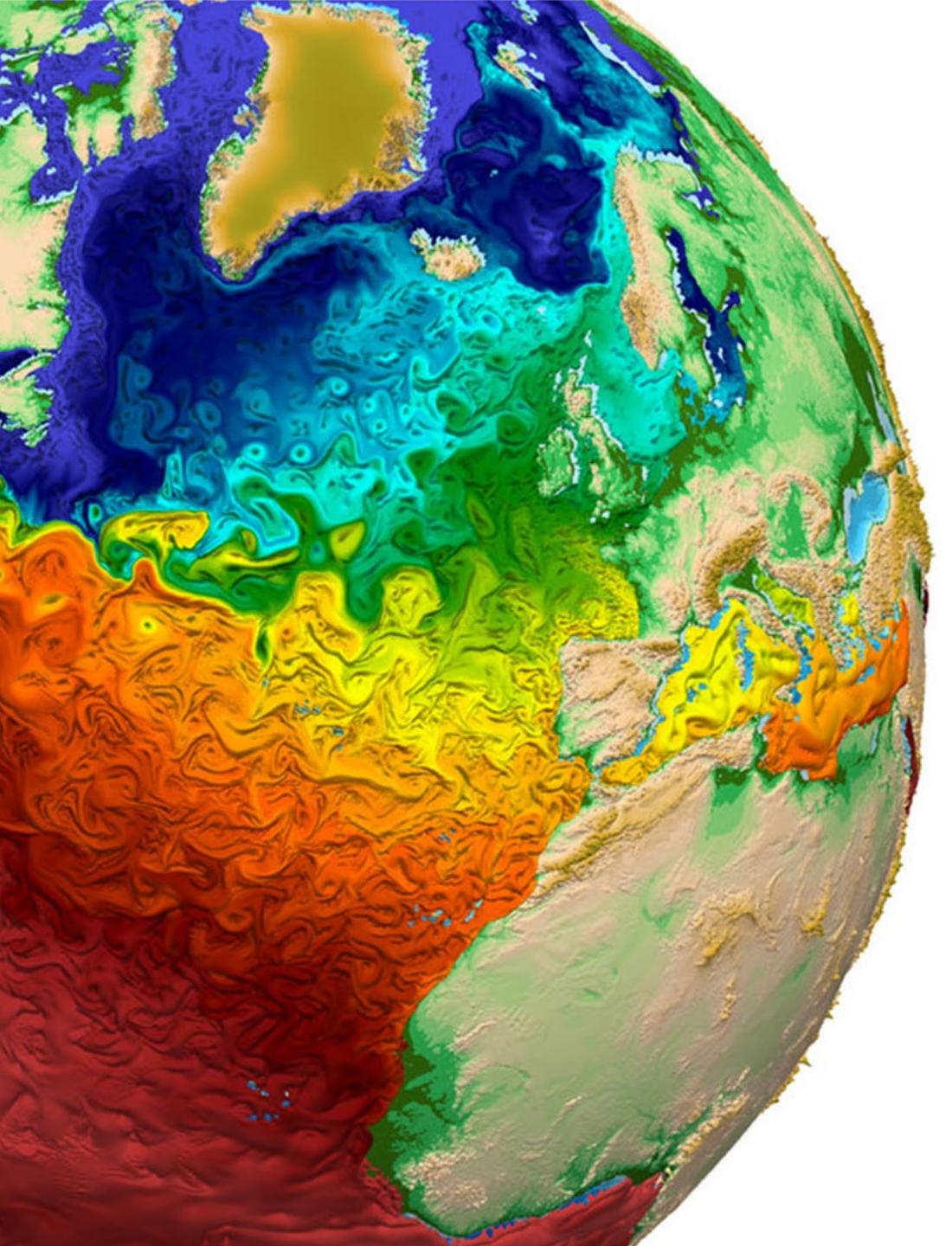


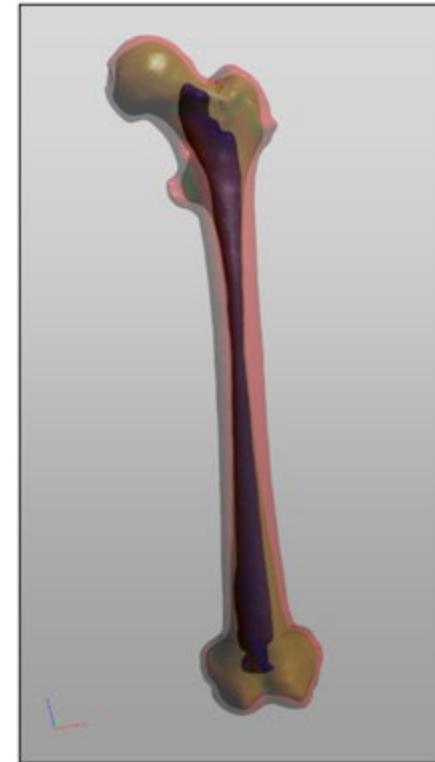
Image Segmentation

Introduction

Scientific Visualization
Professor Eric Shaffer

Segmentation

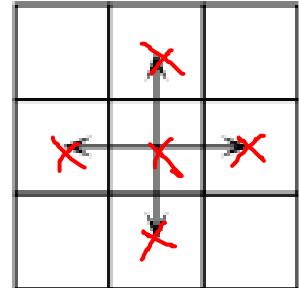
- A partitioning image (or data)
- Each partition is
 - Connected
 - Homogeneous
 - Identified by a unique label
- There are different interpretations of
 - Connected
 - Homogeneous



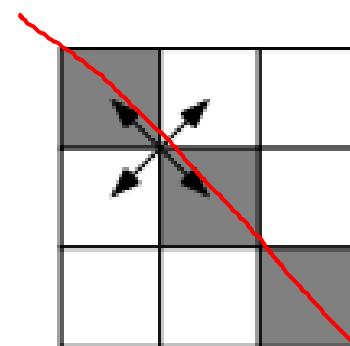
Model of a segmented left human [femur](#). It shows the outer surface (red), the surface between compact bone and spongy bone (green) and the surface of the bone marrow (blue). – Wikipedia

Connected Regions

- Many possible definitions...correctness depends on context
- One option: pixel's neighbors = four pixels that share an edge
 - For $(x,y) \rightarrow (x+1,y), (x-1,y), (x,y+1),$ and $(x,y-1)$
 - Called **4-connected** neighbors

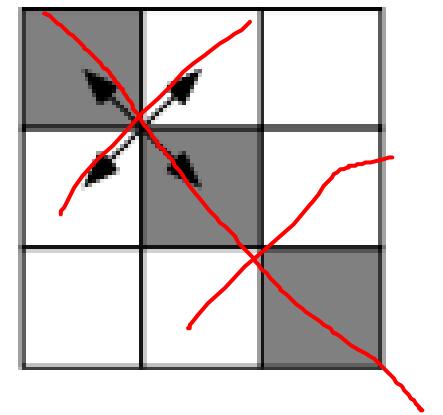
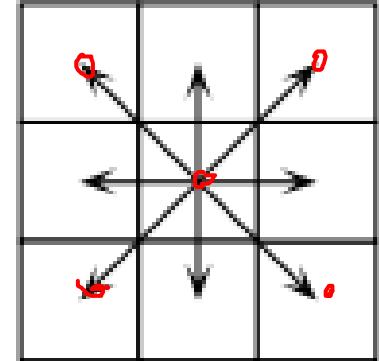


- But...the black pixels are not 4-connected
 - Yet they partition white regions...seem connected intuitively
 - White regions are also not 4-connected across the black pixels.
 - This creates undesirable topological anomalies.



8-Connected Regions

- Alternative: consider diagonal pixels neighbors as well
 - 8-connected neighbors
- Topological anomaly occurs in the case shown
 - Black pixels on the diagonal are connected
 - But so are the white pixels.
 - Some pixels are connected across the links between other connected pixels!

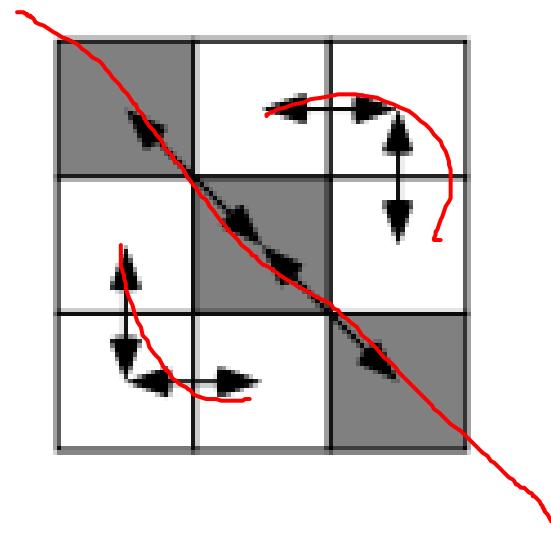


Solution: Different Rules for Different Classifications

- Usual solution:
 - Use 4-connectivity for the foreground
 - Use 8-connectivity for the background

or

- Use 8-connectivity for the foreground
 - Use 4-connectivity for the background



Homogeneous Regions

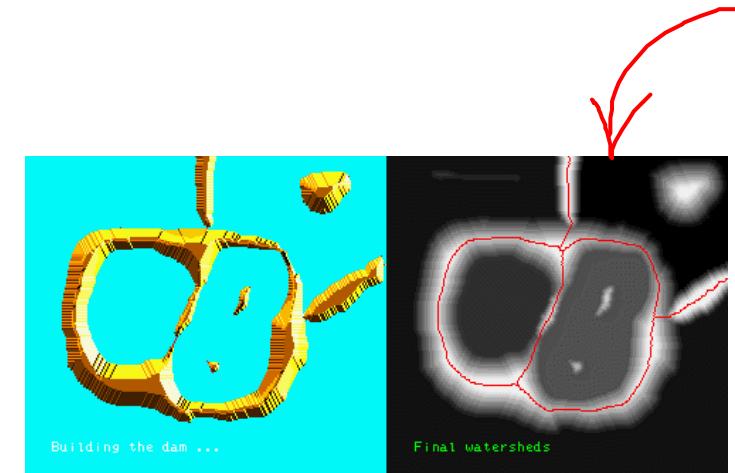
Possible meanings:

- All pixels are the “same” brightness
- Color
- Motion
- Texture
- Generically
 - The values of each pixel are consistent with having been generated by a mechanism



Segmentation Methods

- There are *many* methods
- Here are a few examples:
 - Threshold-based
 - Guaranteed to form closed regions (why?)
 - Region-based
 - Start with elemental homogeneous regions
 - Merge & split them
 - Hybrid methods
 - e.g., watershed



Flood surface from its minima
Prevent the merging of the waters from different sources
→ partitions the image into two different sets:
the catchment basins and the watershed lines.

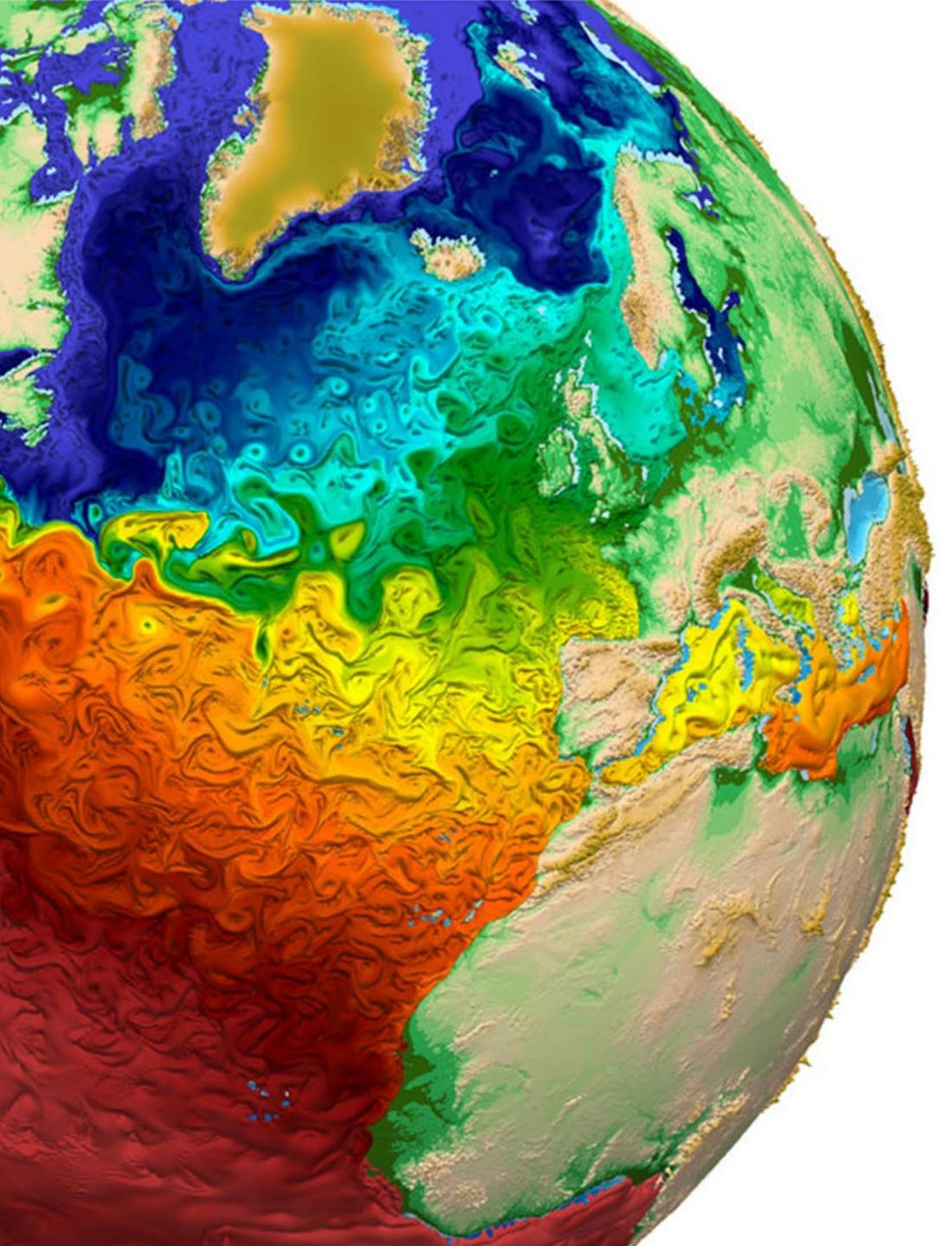


Image Segmentation

Thresholding

Scientific Visualization
Professor Eric Shaffer

Segmentation by Thresholding

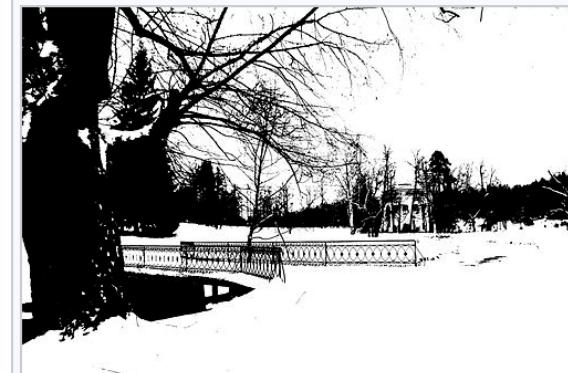
The simplest thresholding methods replace each pixel in an image with a black pixel if the pixel intensity is less than some fixed constant or a white pixel if the image intensity is greater than that constant.

- Wikipedia

Intensity of an (r,g,b) can be computed as $(r+g+b)/3.0$



Original image



Example of a threshold effect used on an image

- Results in a binary labeling

- Good approach for segmenting single object or all objects of given type
- Supports notion of object(s)/foreground vs. background
- Will not be effective distinguishing multiple objects

Segmentation by Thresholding

Problems in real-world conditions

- Images are not taken under perfectly uniform illumination (or radiation, contrast agent, etc.)
- Optical imaging devices are typically not equally sensitive across their field of view

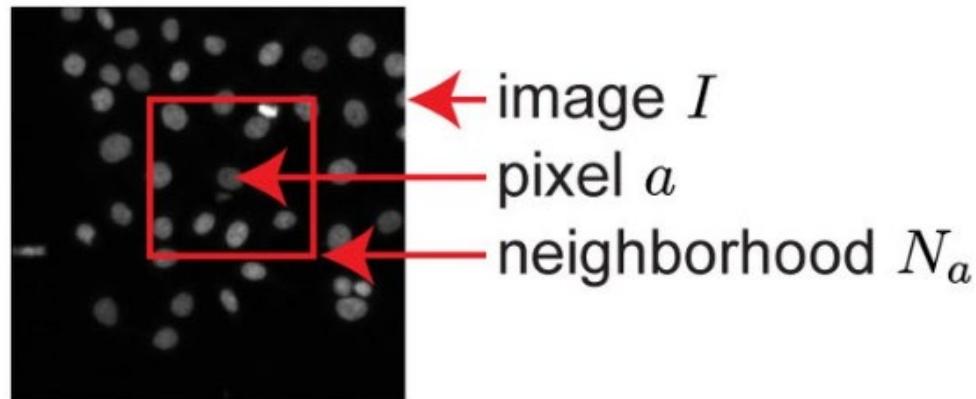
The same object has different intensities at different locations in an image

- Can try to handle this by local thresholding

Local Thresholding

Some options

- Tessellate the image into rectangular blocks
 - Use different threshold parameter(s) for each block
- Use a moving window



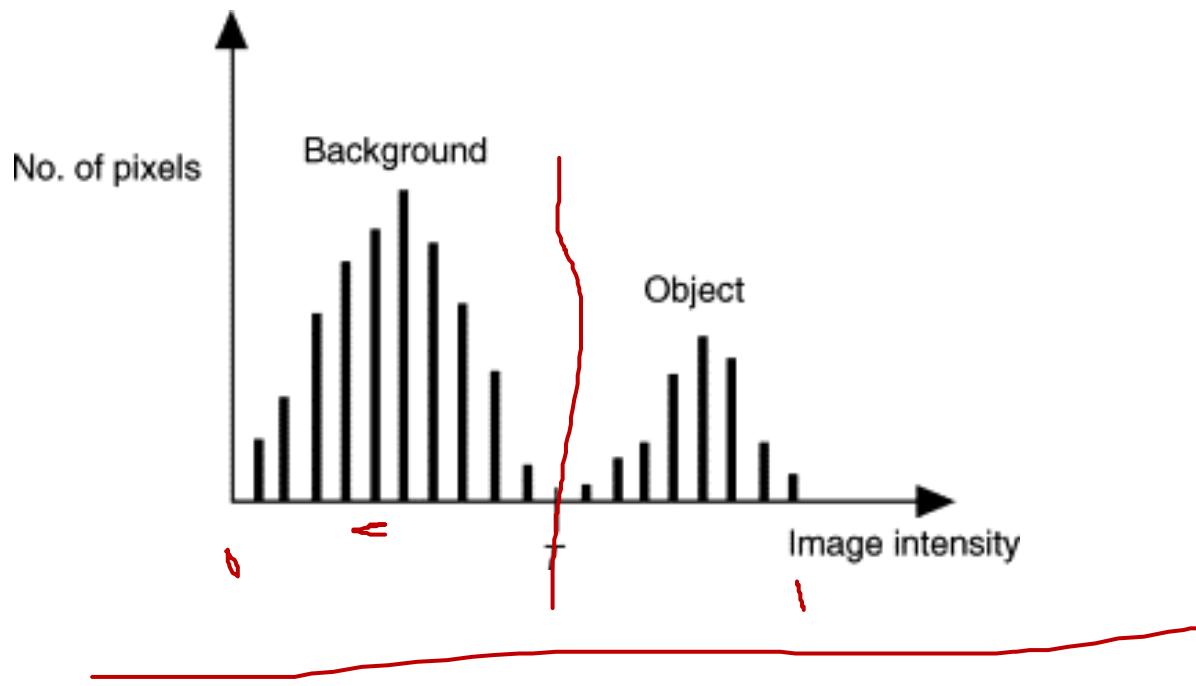
Global threshold: $\tau = f(\{i \in I\})$
Local threshold: $\tau_a = f(\{i \in N_a\})$

Computing Thresholds

- In very limited cases:
 - Multiple, similar objects imaged under virtually identical conditions
 - Choose a good threshold manually for the first object
 - Use that threshold for all future objects
- Most of the time:
 - Different objects under different conditions
 - Need to automate the selection of thresholds
 - Many ways to do this

Computing Thresholds

- If we expect lots of contrast, then use:
 - $T = i_{avg} + \varepsilon$
- More typically, use histogram analysis

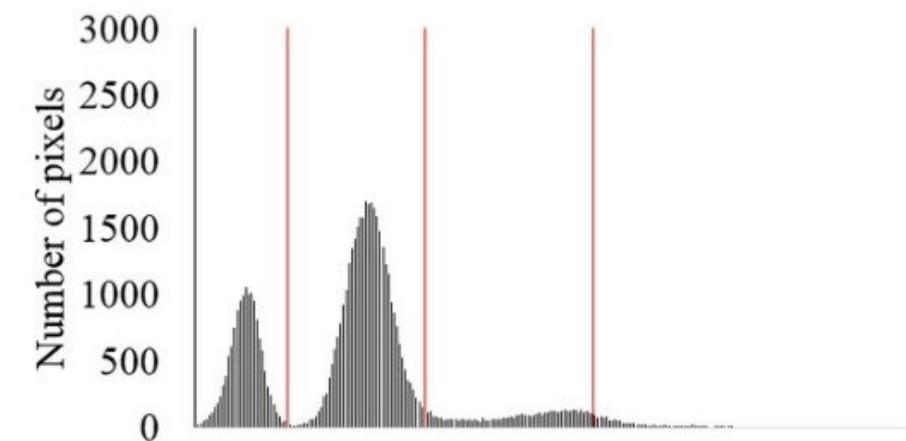
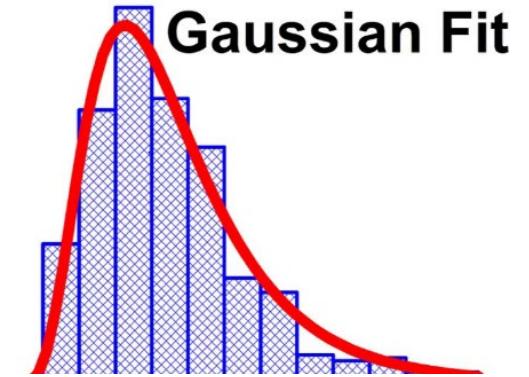


Intensity of an (r,g,b) can be computed as $(r+g+b)/3.0$



Histogram Analysis

- Idea: Choose a threshold between the peaks
- Histograms often require pre-processing
- How to find the peaks?
 - Try fitting Gaussians to the histogram
 - Use their intersection(s) as the threshold(s)
 - How many Gaussians to try to fit?
 - Requires prior knowledge...or maybe Machine Learning



Otsu's Method

In the simplest form, the algorithm returns a single intensity threshold that separate pixels into two classes, foreground and background. This threshold is determined by minimizing intra-class intensity variance, or equivalently, by maximizing inter-class variance.

- Wikipedia



An example image thresholded using Otsu's algorithm

The algorithm searches for the threshold that minimizes

$$\underline{\sigma_w^2(t)} = \underline{\omega_0(t)\sigma_0^2(t)} + \underline{\omega_1(t)\sigma_1^2(t)}$$

the intra-class variance, defined as a weighted sum of variances of the two classes

Weights ω_0 and ω_1 are the probabilities of the two classes separated by a threshold t , and σ_0^2 and σ_1^2 are variances of these two classes.



Original image

And...what is variance again?

For a discrete random variable X

$$\text{Var}(X) = \sum_{i=1}^n p_i \cdot (x_i - \mu)^2$$

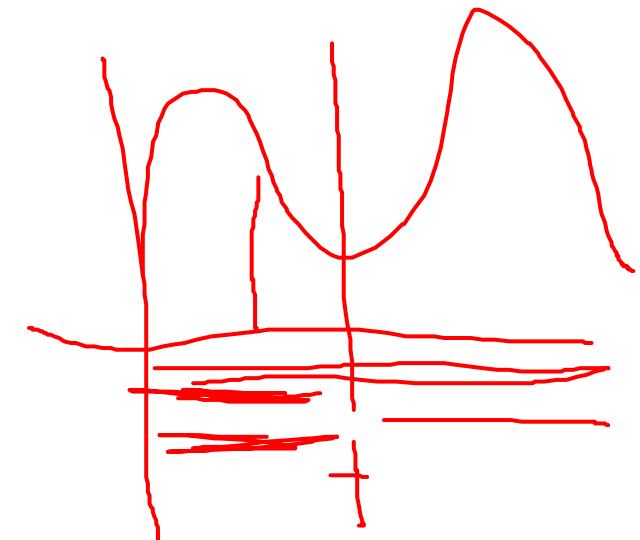
μ is the average

p_i is the Probability the $X \rightarrow x_i$

Informally, it measures how far a set of numbers is spread out from their average value.

Computing the Weights

Weights ω_0 and ω_1 are the probabilities of the two classes separated by a threshold t , and σ_0^2 and σ_1^2 are variances of these two classes.



$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

The class probability is computed from the L bins of the histogram

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

Maximizing the Gap

For 2 classes, minimizing the intra-class variance is equivalent to maximizing inter-class variance

..search for best gap...find t maxing:

$$\omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

$$\mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

Algorithm

1. Compute histogram and probabilities of each intensity level
2. Set up initial $\omega_i(0)$ and $\mu_i(0)$
3. Step through all possible thresholds $t = 1, \dots$ maximum intensity
 1. Update ω_i and μ_i
 2. Compute $\sigma_b^2(t)$
4. Desired threshold corresponds to the maximum $\sigma_b^2(t) = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$

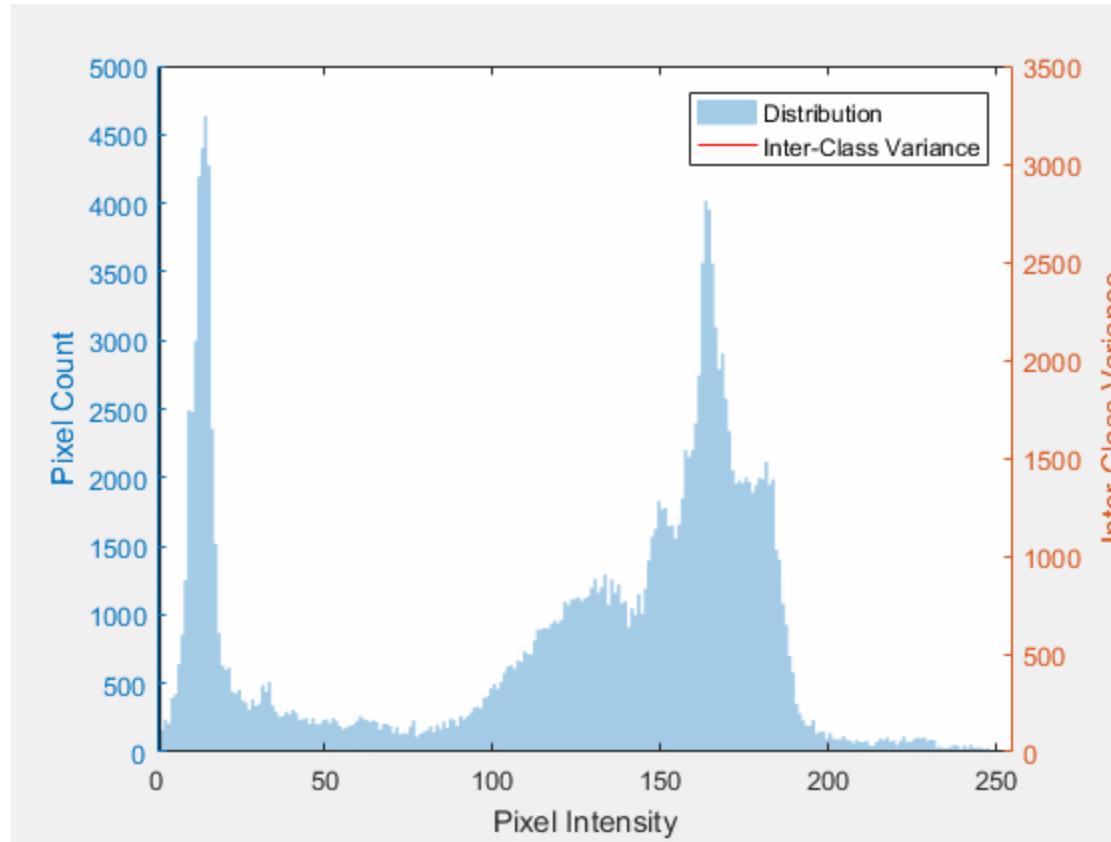
$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

$$\mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

Otsu's Method



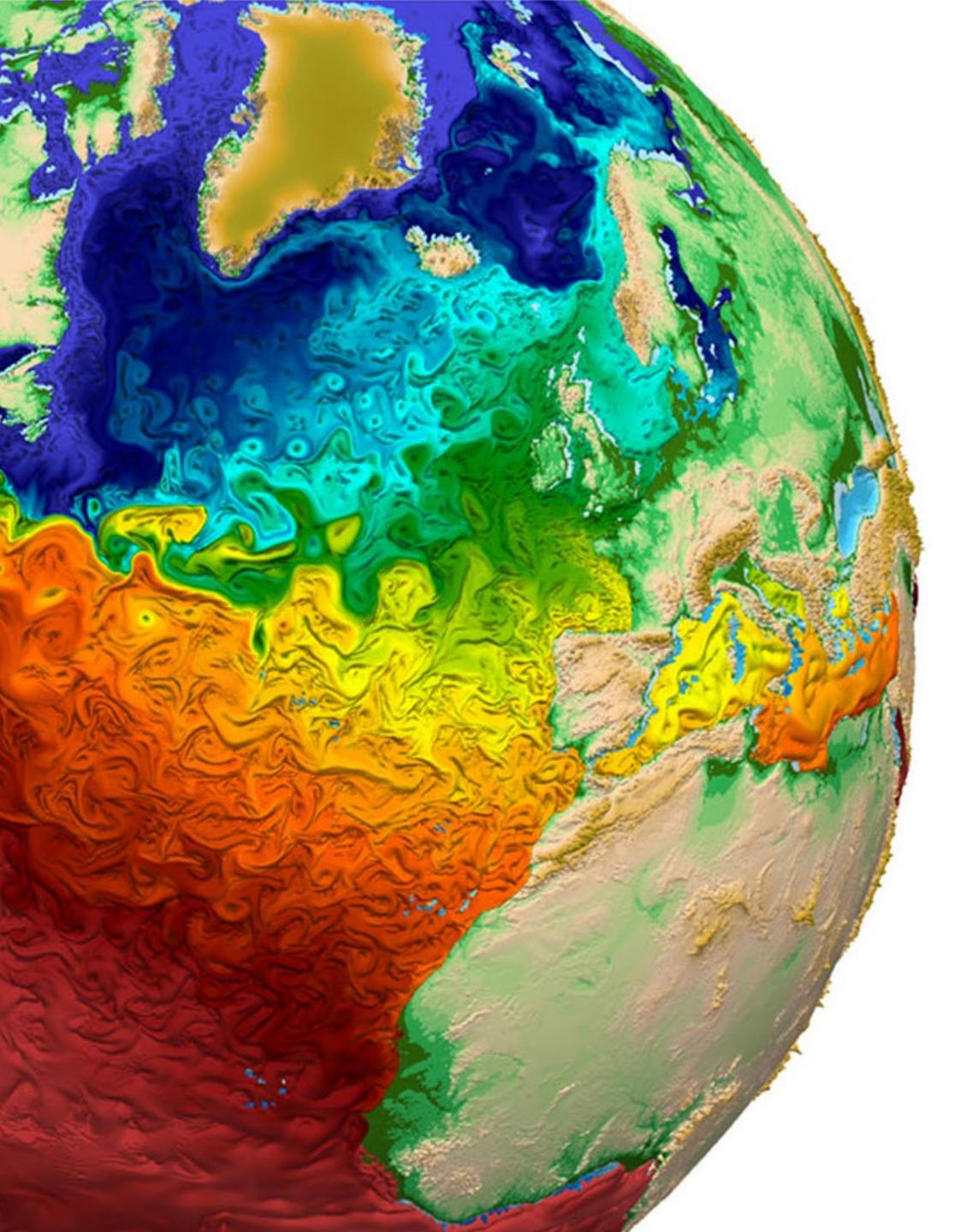


Image Segmentation

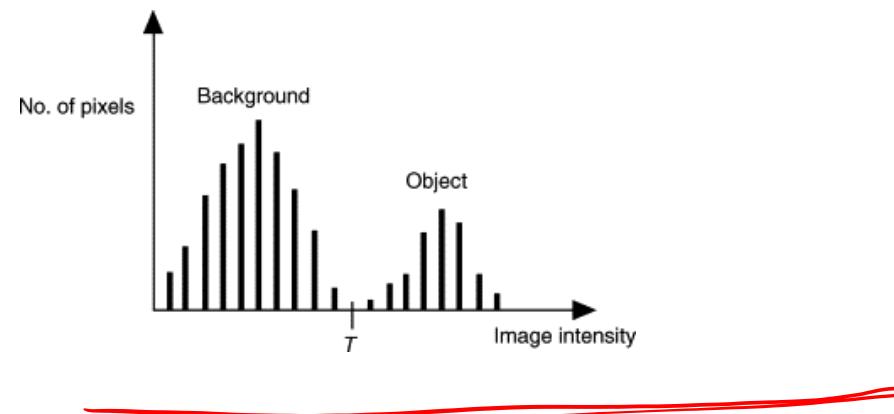
Segmenting Multiple Objects

Scientific Visualization
Professor Eric Shaffer

Review: Segmentation by Thresholding

Classical thresholding results in a binary labeling

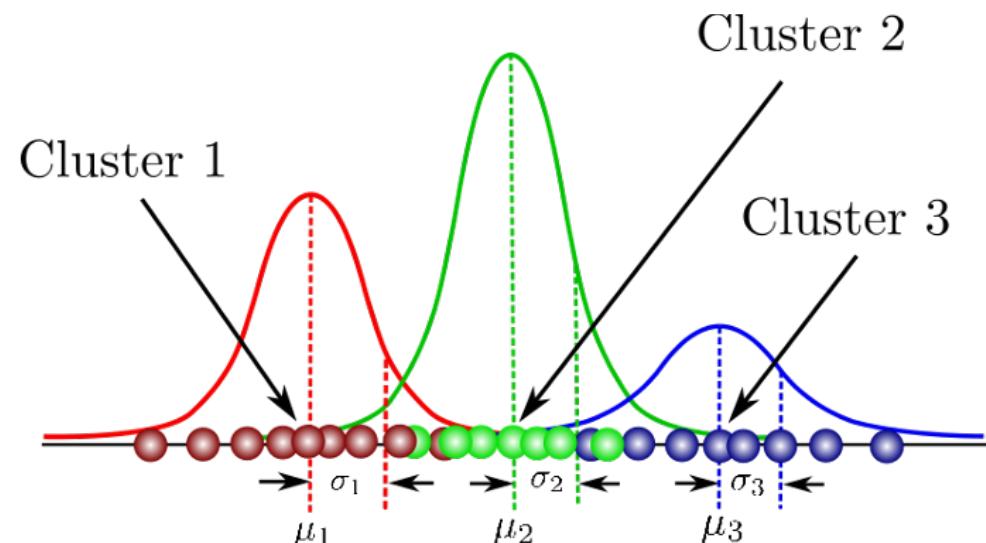
- Otsu's method
- Good approach for segmenting single object or all objects of given type
- Supports notion of object(s)/foreground vs. background
- Will not be effective distinguishing multiple objects



Other Approaches to Thresholding

Expectation Maximization (EM) to fit Gaussians

- More powerful and flexible than Otsu's method
- EM can infer the parameters of multiple Gaussian distributions
- Usually more computationally expensive
- You must separately compute the thresholds after the GMM has been inferred
 - GMM is Gaussian mixture model
- General EM of GMM can easily model multiple classes of pixels



Thresholding with Non-Binary Labels

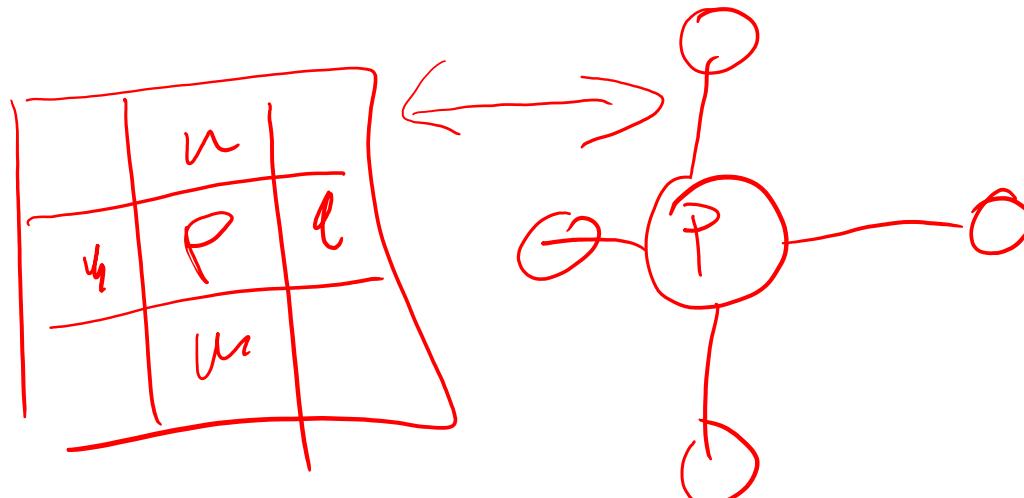
- Recall that thresholding produces a binary segmentation
- What if thresholding detects multiple objects
...but we need to analyze each of them separately?
 - e.g. multiple bones in a CT scan
 - Multiple fiducial markers.
- We need to give a different label to each detected object

Graph-Theoretic Image Segmentation

Can use a graph-theoretic approach to extend binary thresholding

Connected component analysis lets us:

- Assign a different label to each (disconnected) object...
- from the (binary) set of segmented objects



1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	2	2	1	1	1	2	2	2	2	1
1	1	2	2	1	1	1	2	2	2	2	1
1	1	2	2	1	1	1	2	2	2	2	1
1	1	2	2	1	1	1	2	2	2	2	1
1	1	2	2	1	1	1	2	2	2	2	1
1	1	2	2	2	1	1	1	2	2	2	1
1	1	2	2	2	1	1	1	2	2	2	1
1	1	2	2	2	1	1	1	2	2	2	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1

1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	2	2	1	1	1	3	3	3	3	1
1	1	2	2	1	1	1	3	3	3	3	1
1	1	2	2	1	1	1	3	3	3	3	1
1	1	2	2	1	1	1	3	3	3	3	1
1	2	2	2	2	1	1	1	3	3	3	1
1	2	1	1	2	1	1	1	3	3	3	1
1	2	1	1	2	1	1	1	3	3	3	1
1	2	2	2	2	1	1	1	3	3	3	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1

Recursive Region Growing

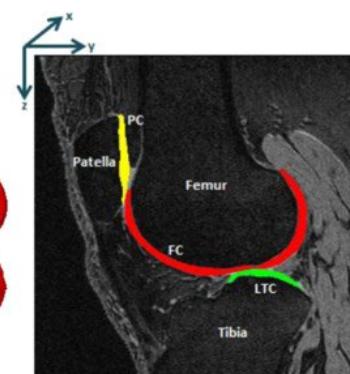
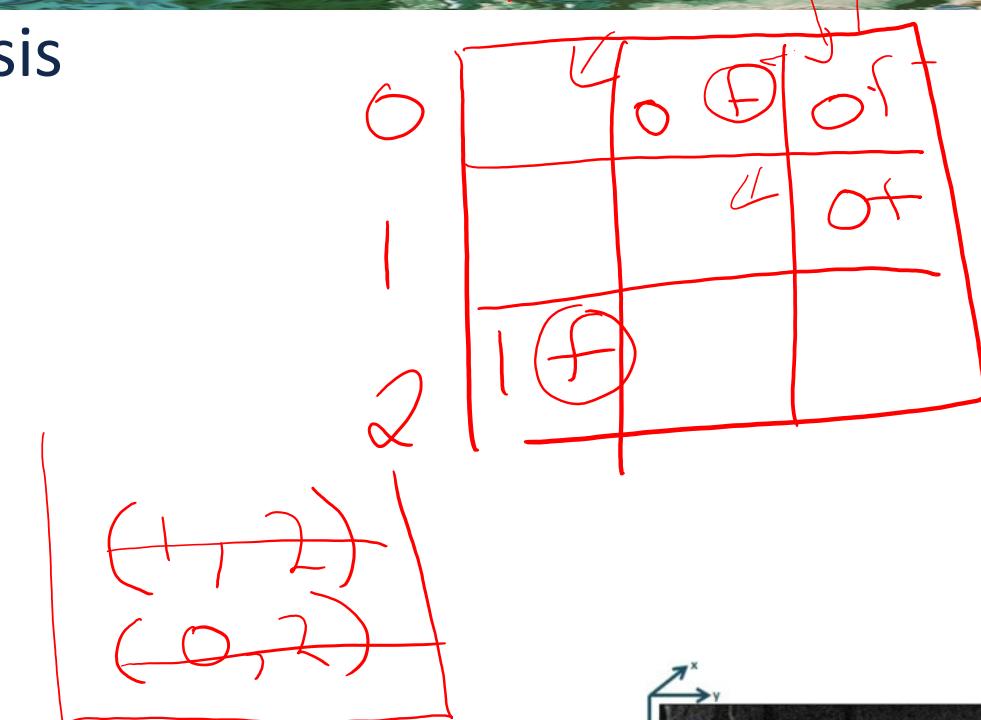
$i = \text{?}$ 0 1 2

One method of doing connected component analysis

Algorithm

Input: A threshold segmentation with object pixels as black in f

1. Search for an unlabeled black pixel; that is, $L(x, y) = 0$
If you find one, choose a new label number for this region, call it N
2. If all pixels have been labeled, stop
3. $L(x, y) \leftarrow N$
4. Push unlabeled neighboring object pixels onto the stack
 - If $f(x-1, y)$ is black push $(x-1, y)$ onto the stack.
 - If $f(x+1, y)$ is black push $(x+1, y)$ onto the stack.
 - If $f(x, y-1)$ is black push $(x, y-1)$ onto the stack.
 - If $f(x, y+1)$ is black push $(x, y+1)$ onto the stack.
5. If the stack is empty, go to 1
6. Else choose a new (x, y) by popping the stack and go to 2.



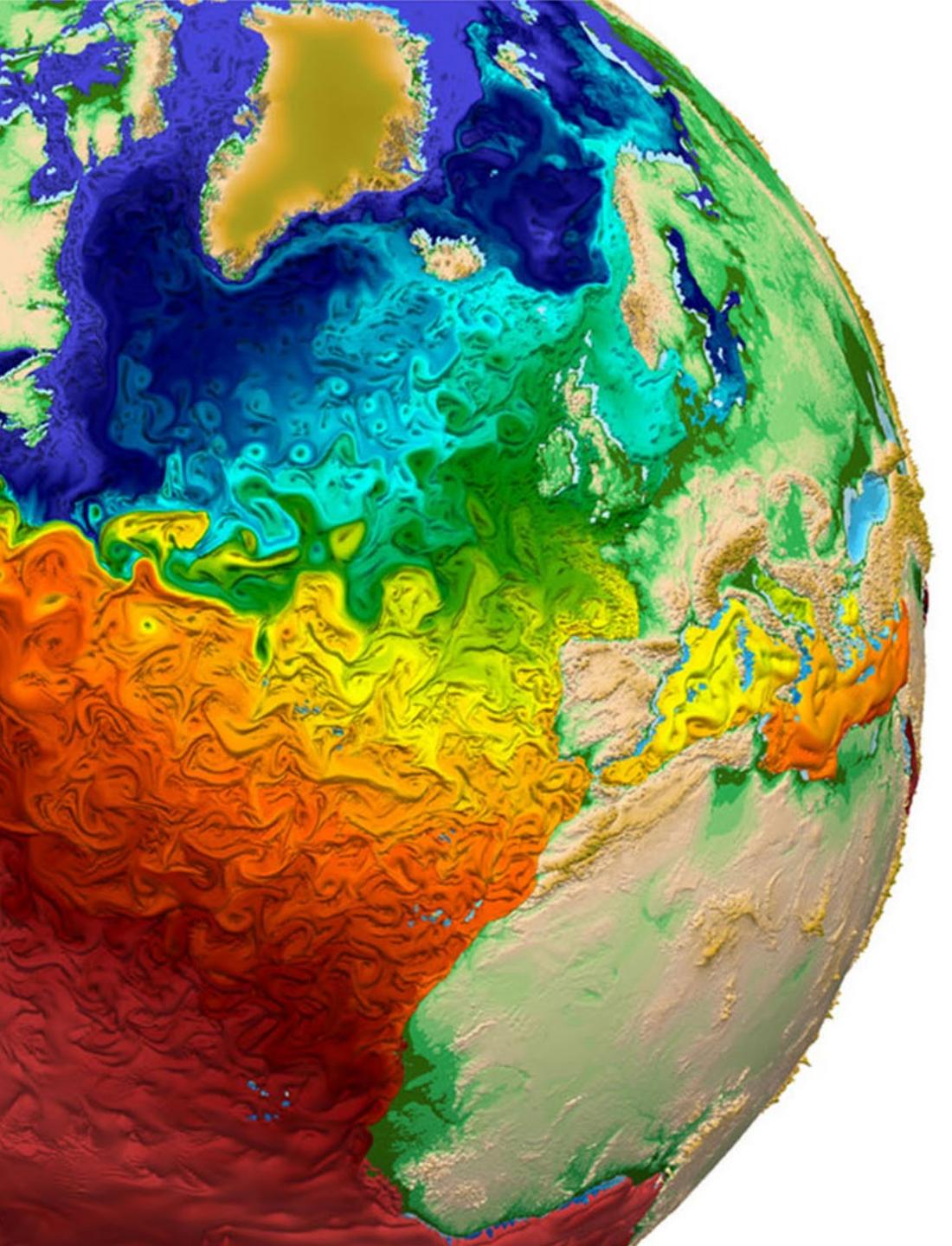


Image Segmentation

Graph Cuts

Scientific Visualization
Professor Eric Shaffer

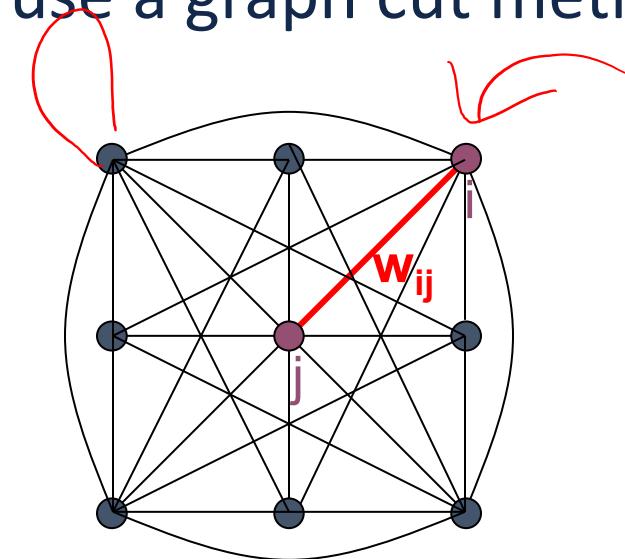
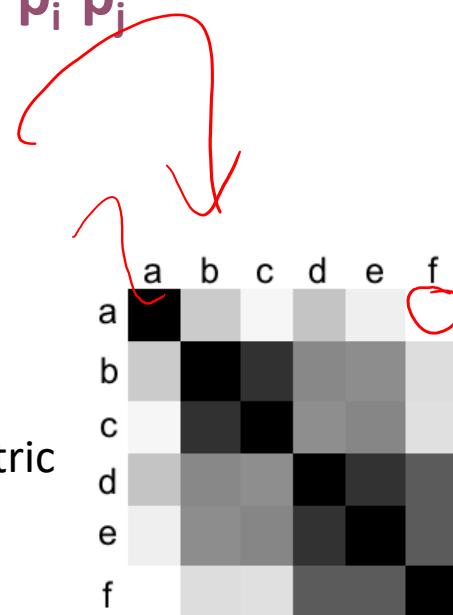
Segmentation with Graph Cuts

Another approach to segmentation in general is to use a graph cut method

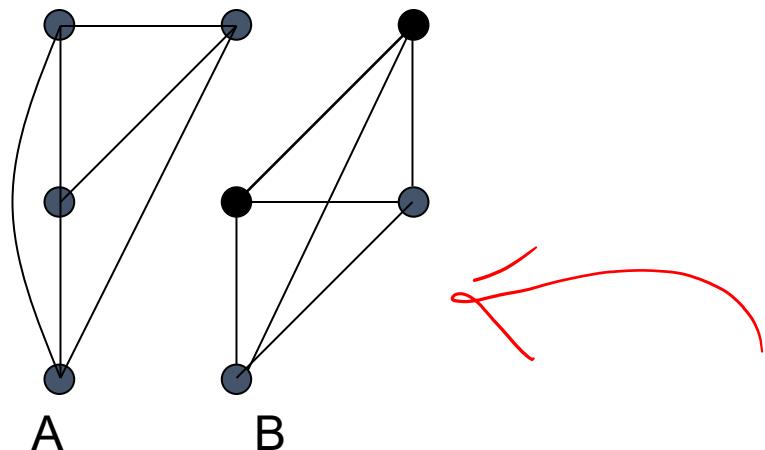
Represent image as a fully-connected graph

- node for every pixel
- link between *every* pair of pixels, p_i , p_j
- similarity w_{ij} for each link

Similarity matrix using distance metric

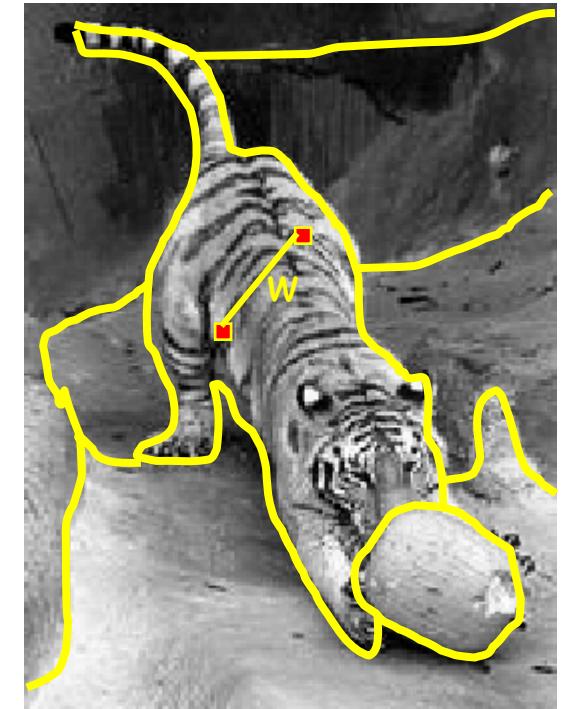


Segmentation by Graph Cuts



Idea: break graph into segments by deleting links

- Break links that have low cost (low similarity)
 - similar pixels should be in the same segment
 - dissimilar pixels should be in different segment



Do We Need All those Edges?

Fully connected:

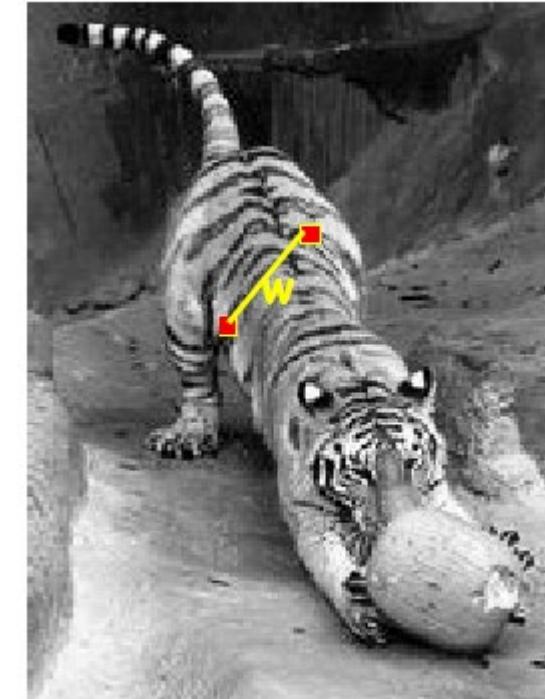
- Captures all pairwise similarities
- Infeasible for most images

Neighboring pixels (4 or 8 connected)

- Very fast to compute
- Only captures very local interactions

Local neighborhood (...more than 8)

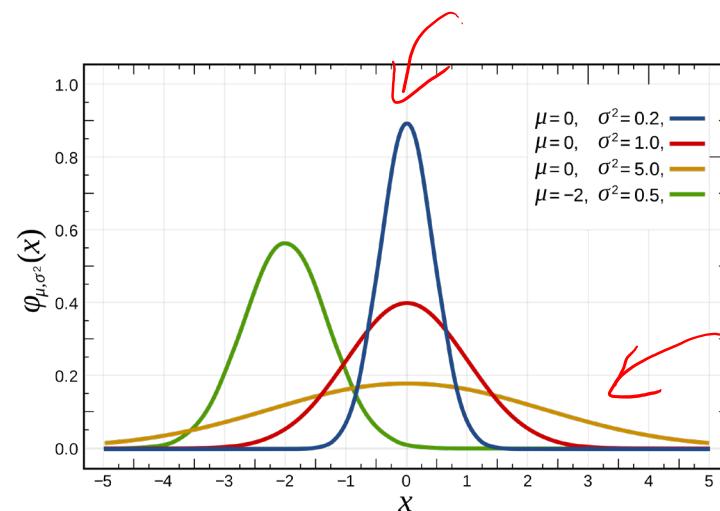
- Reasonably fast, graph still very sparse



How to Calculate Weights?

$$\text{affinity}(p_i, p_j) = e^{-\frac{\|f(p_i) - f(p_j)\|^2}{2\sigma^2}}$$

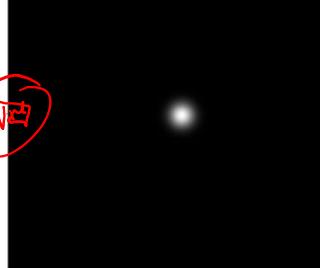
- p_i, p_j are pixels
- $f(p_i)$ metric for a pixel we can use in a distance function
- $2\sigma^2$ is “normalization” factor...a tweakable parameter



Metrics for Distance Measurement

Distance:

$$f(x) = \text{location}(x)$$



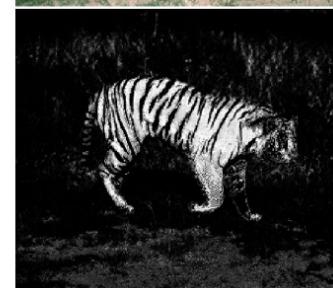
Intensity:

$$f(x) = \text{intensity}(x)$$



Color:

$$\underline{f(x) = \text{color}(x)}$$



Texture:

$$\underline{f(x) = \text{filterbank}(x)}$$

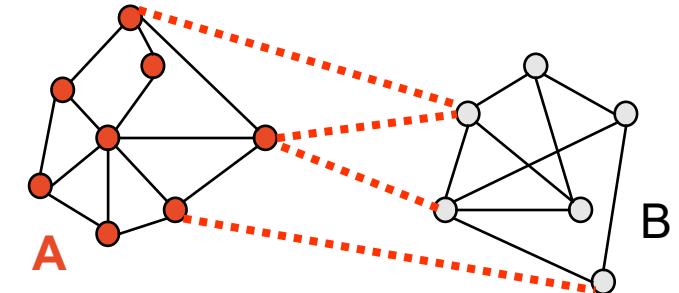


Cuts in a Graph

Definition: Cut

- set of links whose removal makes a graph disconnected

- cost of a cut: $cut(A, B) = \sum_{p \in A, q \in B} c_{p,q}$



Many approaches to finding a cut:

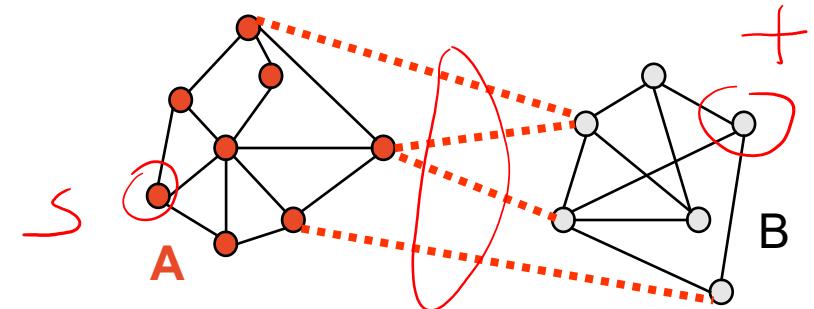
- Spectral clustering (use eigenvalues/vectors to find a cut)
- Optimization problem (minimize an energy function related to affinities)
- Minimum cut using combinatorial optimization algorithm
- Normalized cuts

With one cut we are segmenting out a single object...this approach can be extended to multiple objects as well

Cuts in a Graph

Definition: Cut

- set of links whose removal makes a graph disconnected
- cost of a cut: $cut(A, B) = \sum_{p \in A, q \in B} c_{p,q}$



One idea: find minimum cut

- gives you a segmentation
- fast algorithms exist for doing this
- max flow - min cut algorithms
 - Ford-Fulkerson
 - Push-Relabel
 - Boykov-Komolgorov

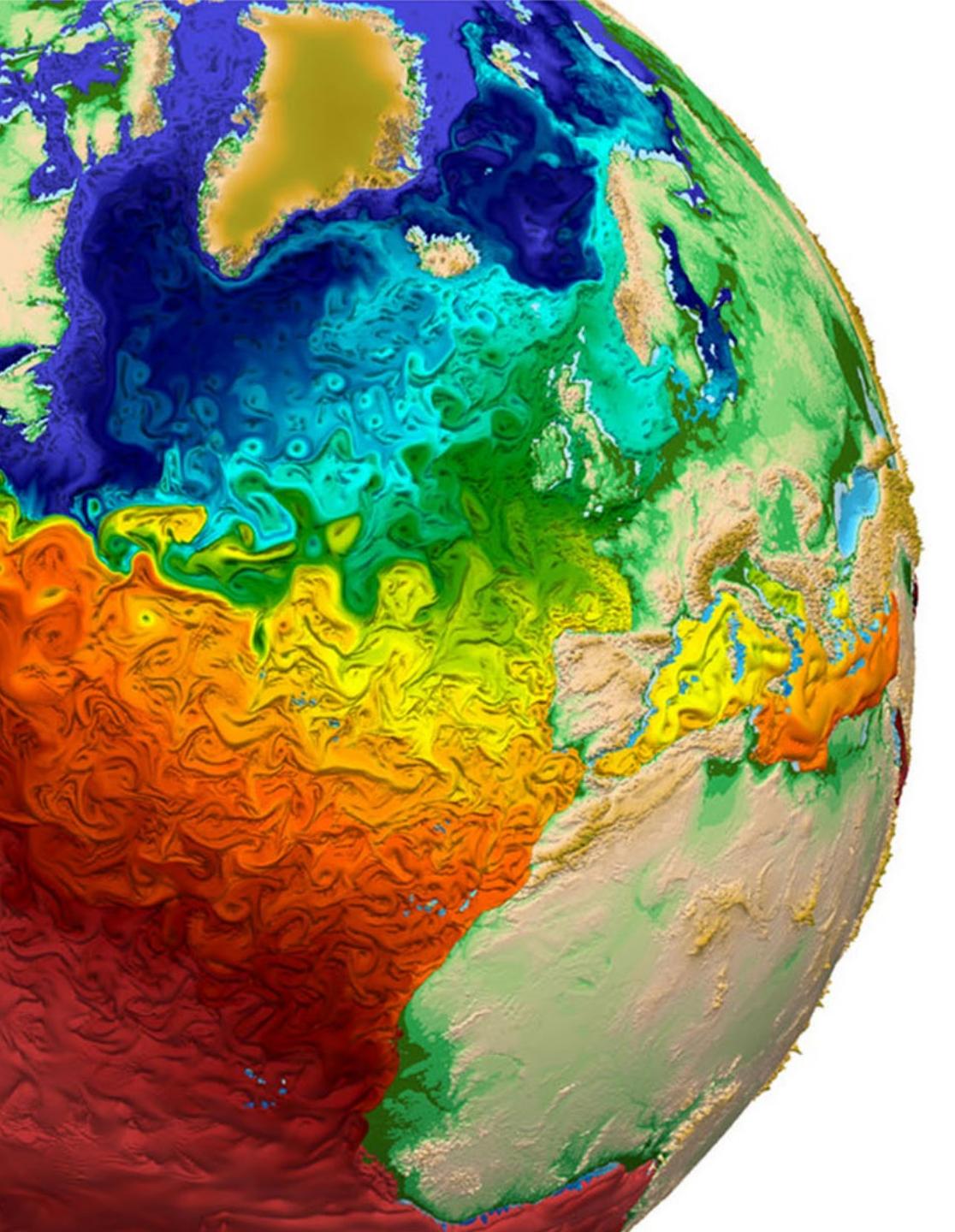


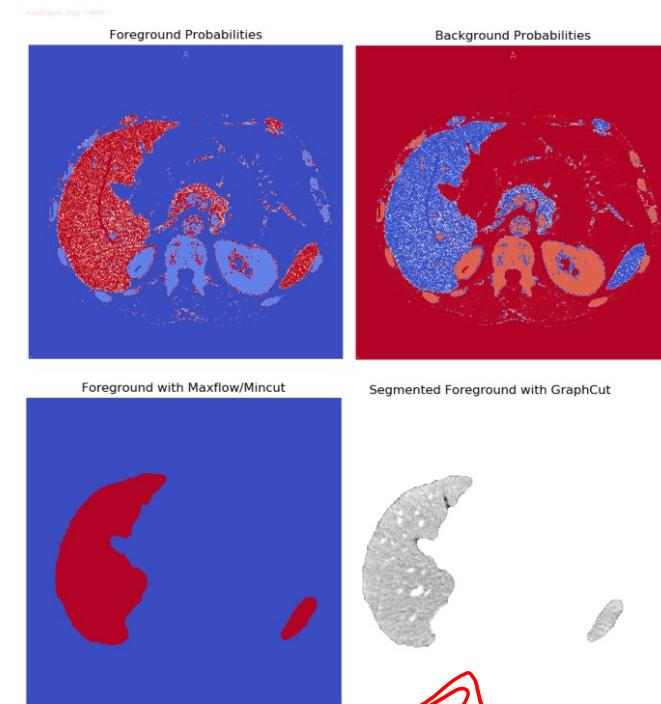
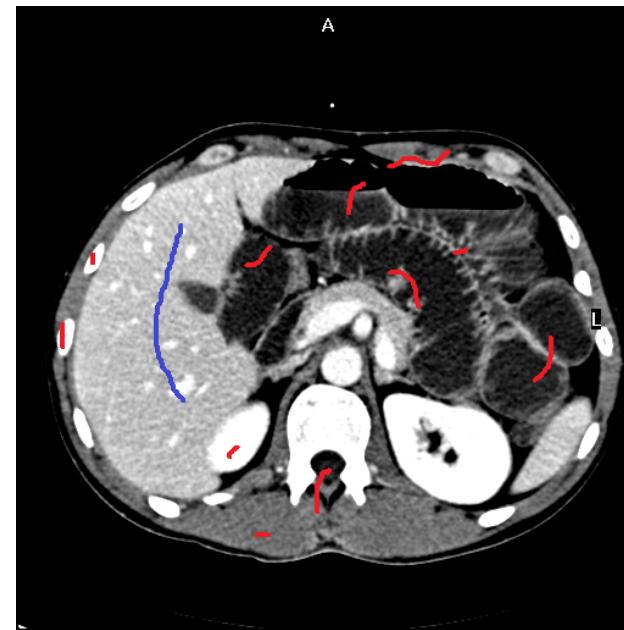
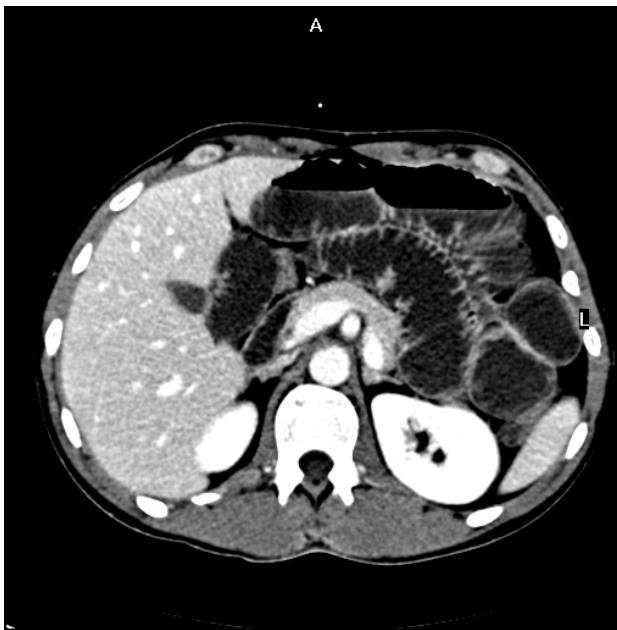
Image Segmentation

Interactive Graph Cuts

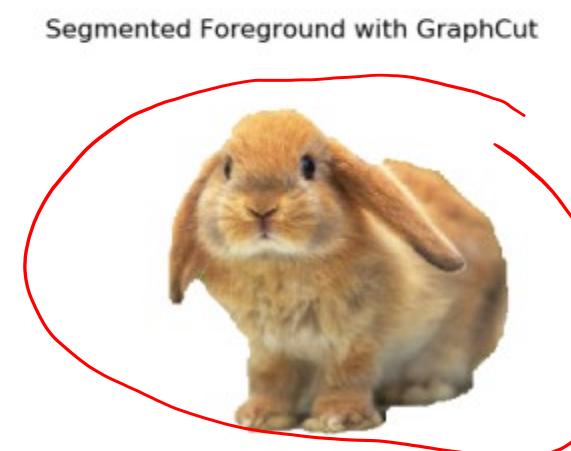
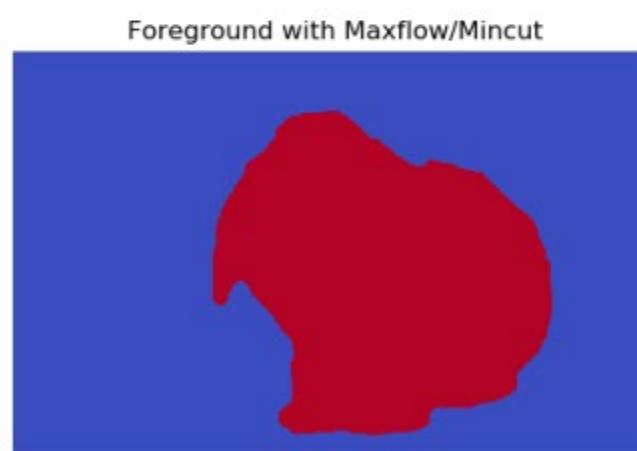
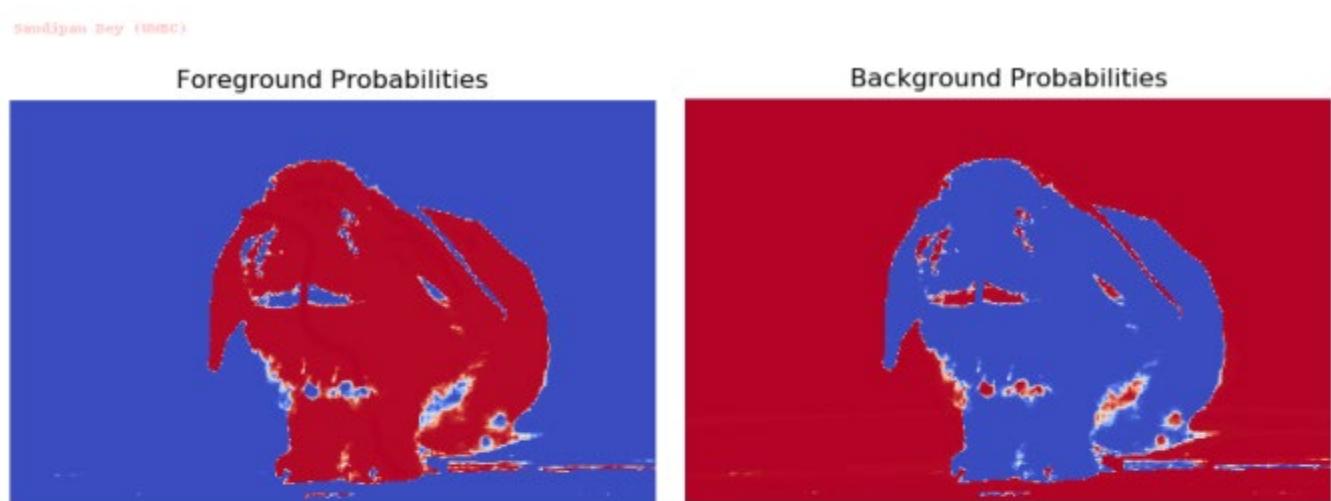
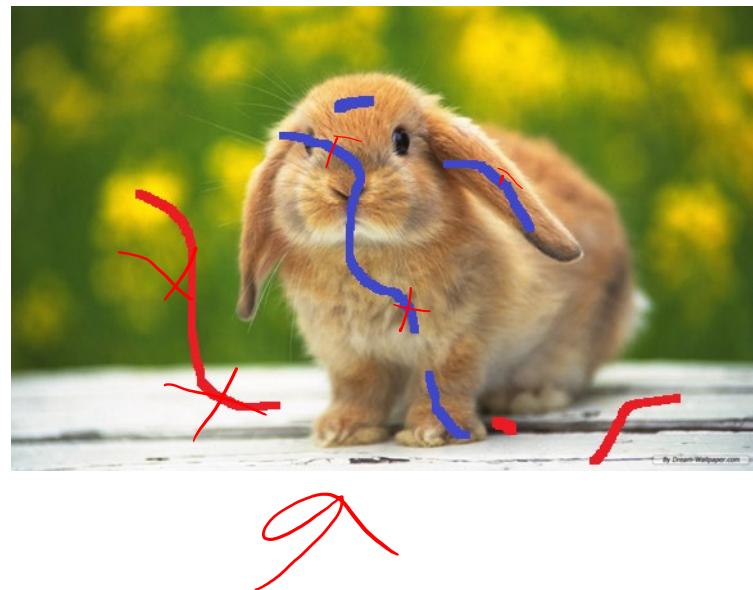
Scientific Visualization
Professor Eric Shaffer

Example: Interactive Graph Cuts

Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images Y. Y. Boykov and M. Jolly, Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images, Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001,

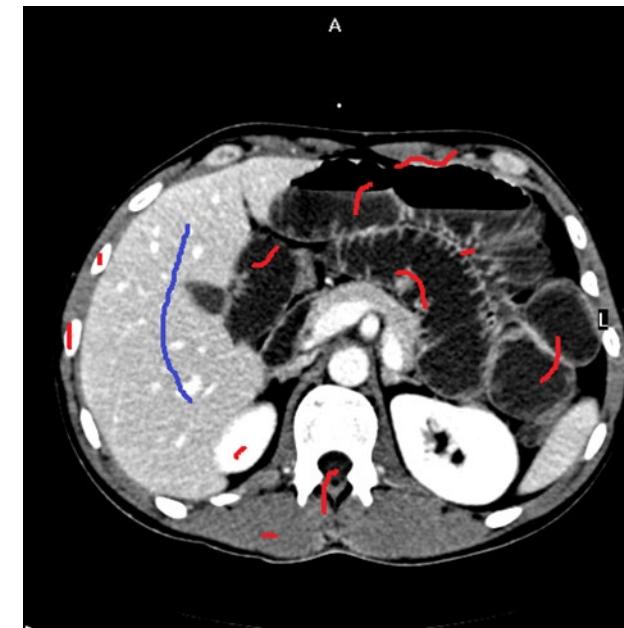
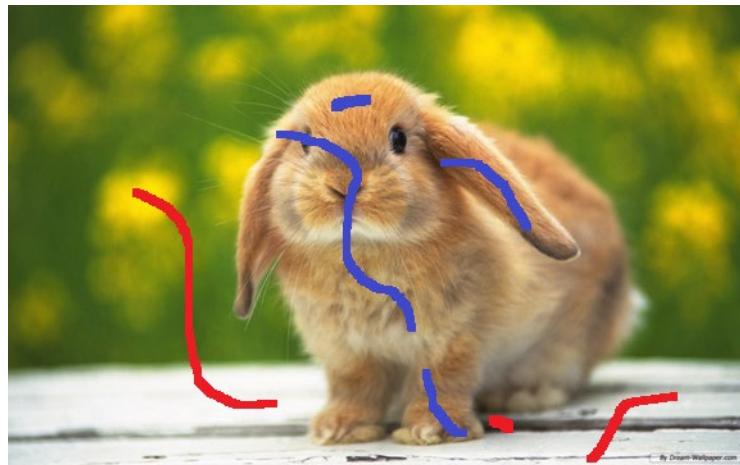


...another example

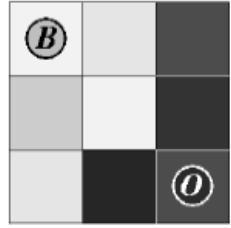


User Input

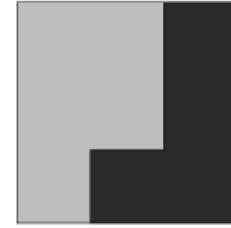
user scribbles provide indication of foreground versus background



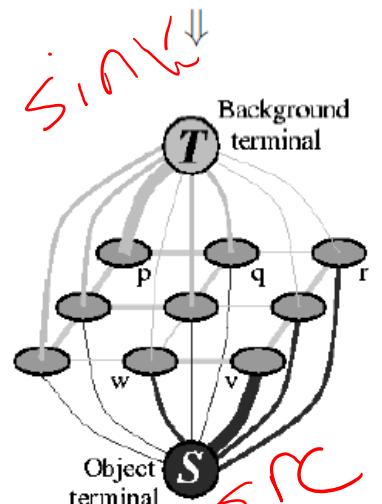
Graph Construction



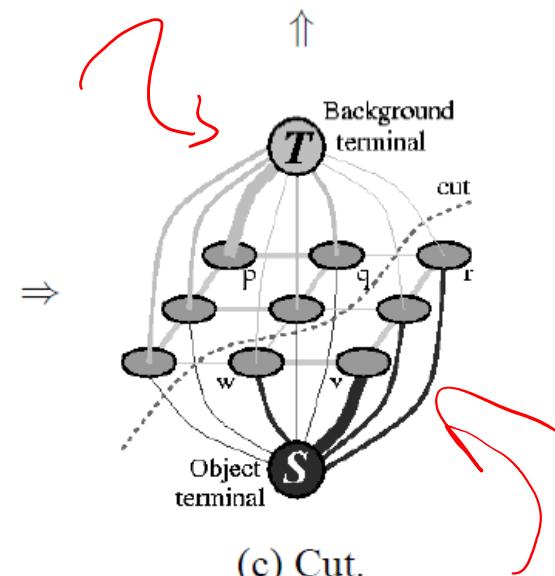
(a) Image with seeds.



(d) Segmentation results.



(b) Graph.



(c) Cut.

Add source s vertex and connect to all pixels

Add sink t vertex connected to all pixels

...doing this in order to use a max flow algorithm

Source and Sink Edge Weights

Connect source to all pixels

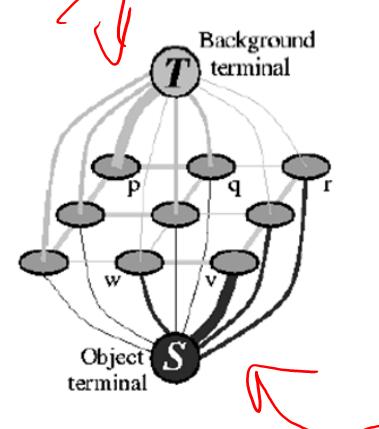
Foreground scribble pixels to foreground(src) vertex = infinity (max float)

Foreground scribble pixels to background(sink) vertex = 0

Connect the sink to all pixels

Background scribble pixels to background vertex(sink) = infinity (max float)

Background scribble pixels to foreground vertex(src) = 0



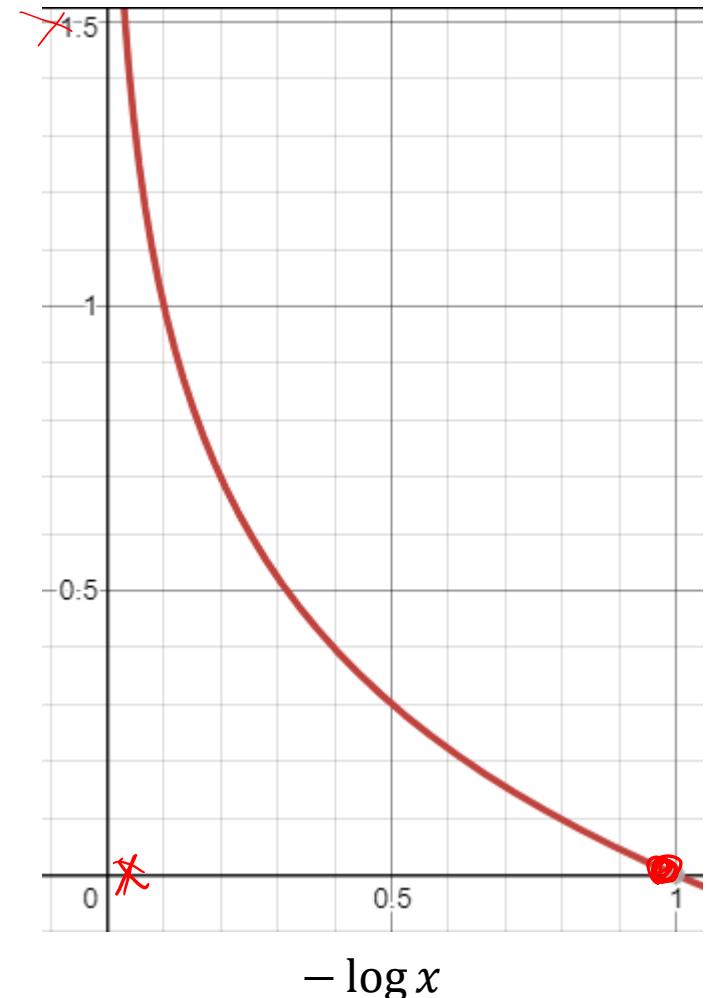
Edge Weights

Source and Sink to Non-Scribble Vertices p

$[0, 1]$

Weight to foreground $\text{affinity}_{foreground}(p) = -\lambda \log P_B(p)$

Weight to background $\text{affinity}_{background}(p) = -\lambda \log P_F(p)$



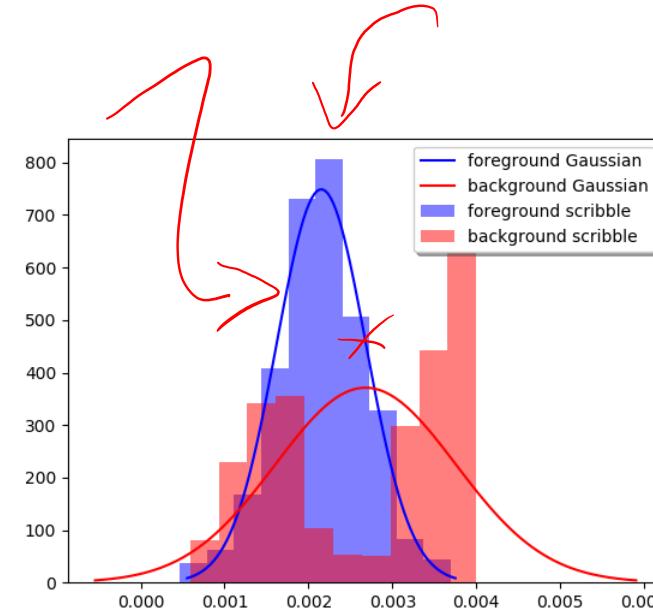
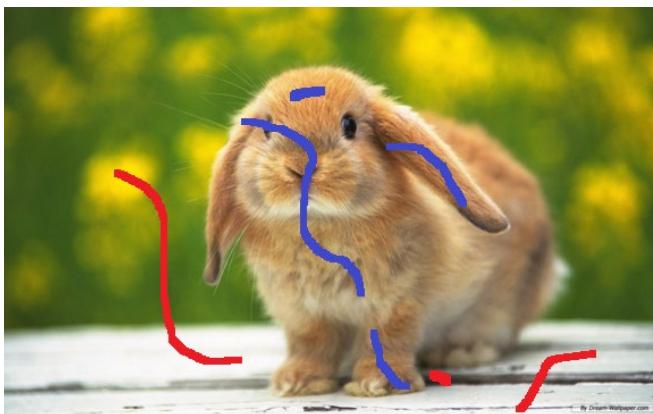
Foreground and Background Probability Distribution

Can use intensity of foreground pixels and compute histogram

Fit a Gaussian to that histogram

Use that distribution to compute $P_F(p) = P_F(I_p)$

I_p is intensity of pixel p



Edge Weights

Connect neighboring image vertices with edges (4 or 8 connected)

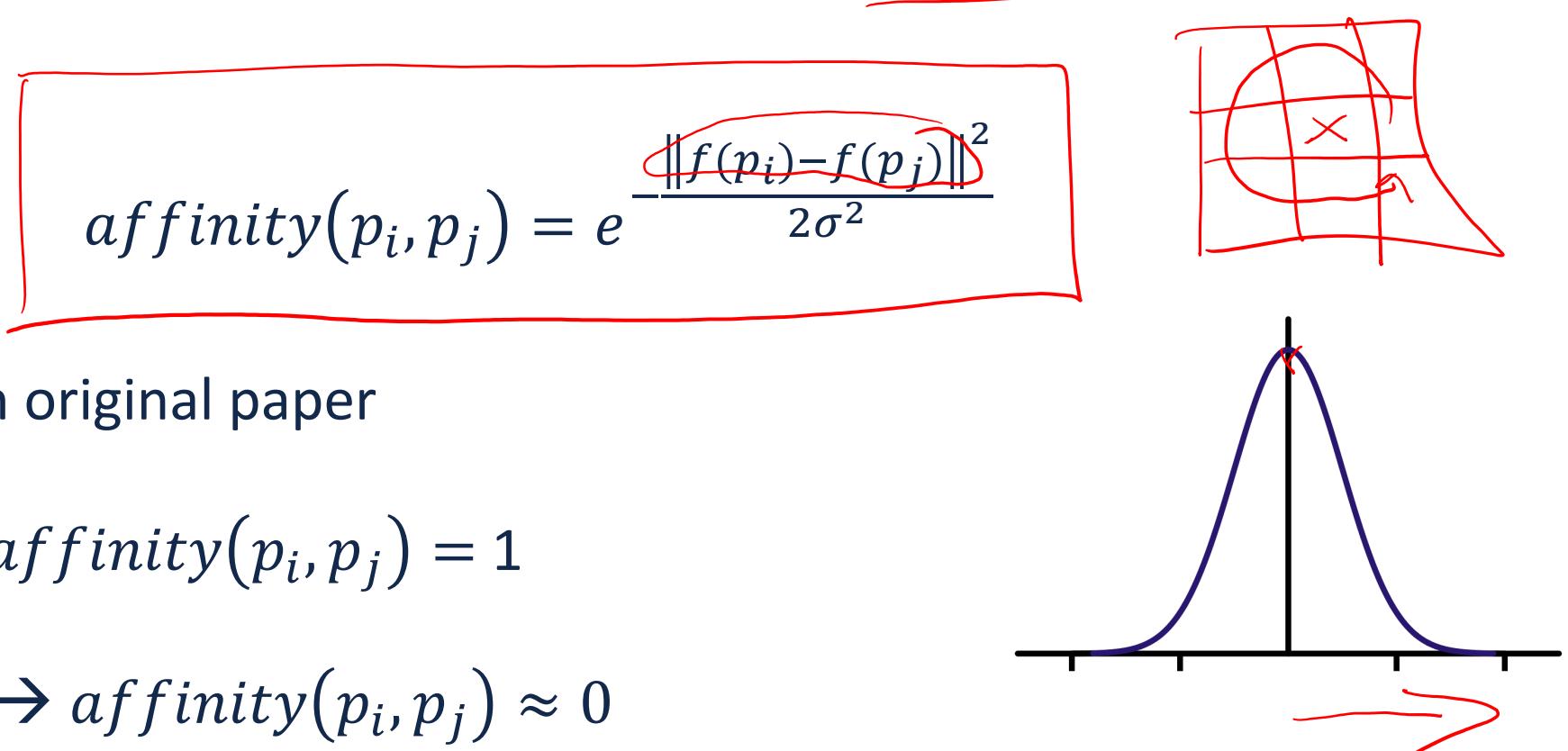
Edge weights:

$$\text{affinity}(p_i, p_j) = e^{-\frac{\|f(p_i) - f(p_j)\|^2}{2\sigma^2}}$$

$f(p_i)$ is intensity in original paper

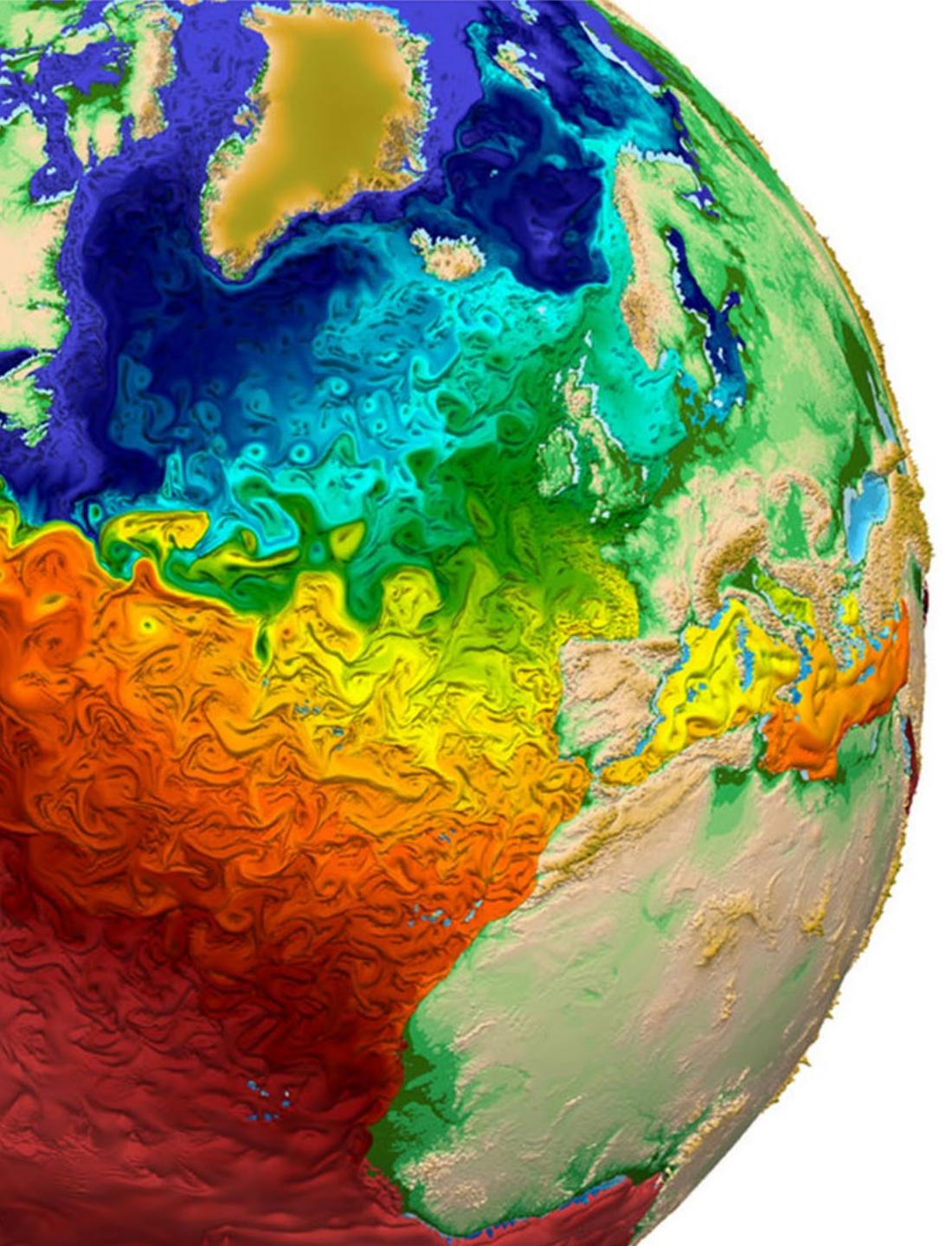
Same intensity $\rightarrow \text{affinity}(p_i, p_j) = 1$

Different intensity $\rightarrow \text{affinity}(p_i, p_j) \approx 0$



Generalizes to nD Volumes

- Can be applied to video data (each frame is a 2D slice)
- Or a volume (e.g. MRI)
 - Uses a 26-connected neighborhood
- In general, fastest max-flow requires $O(VE)$ time
 - In practice....more complicated
 - Algorithms with a higher time-bound can perform better for image data
- Boykov max flow algorithm allows fast updating
 - Add scribbles and adjust existing flow quickly
- Algorithm will generally segment 1920×1080 image in a few seconds



Classification

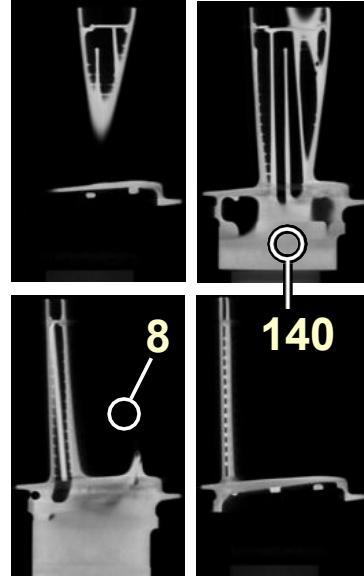
Transfer Functions

Professor Eric Shaffer

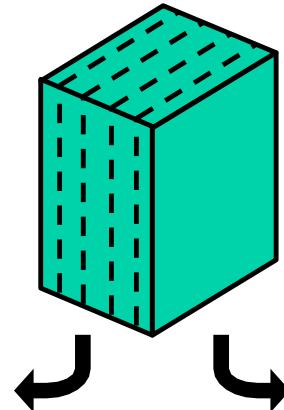
What are Transfer Functions?

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

slices:



volume data:

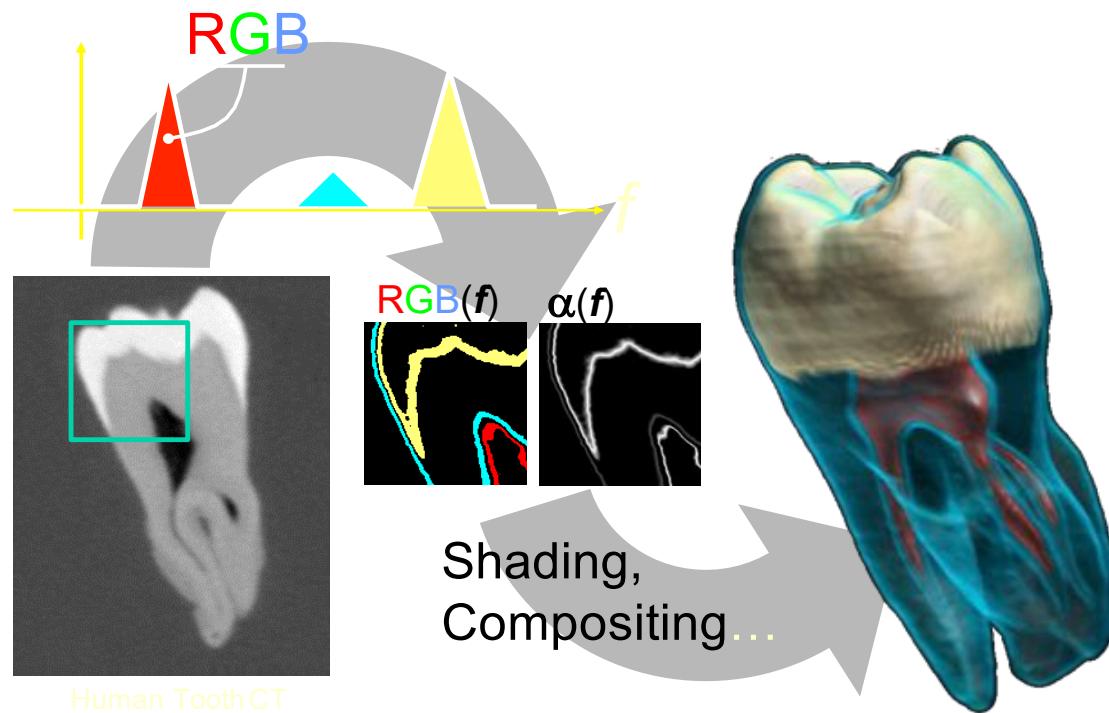


volume rendering:



Transfer Functions

Simple and usual case: Function maps data value to color and opacity



Optical Properties

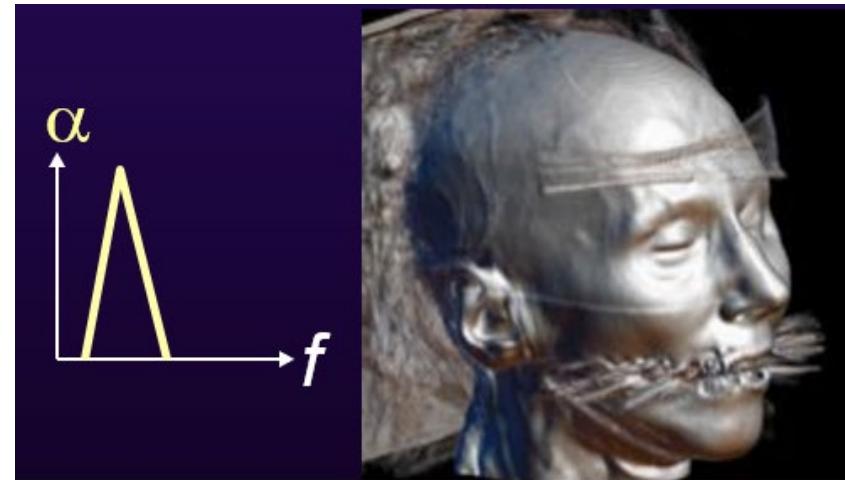
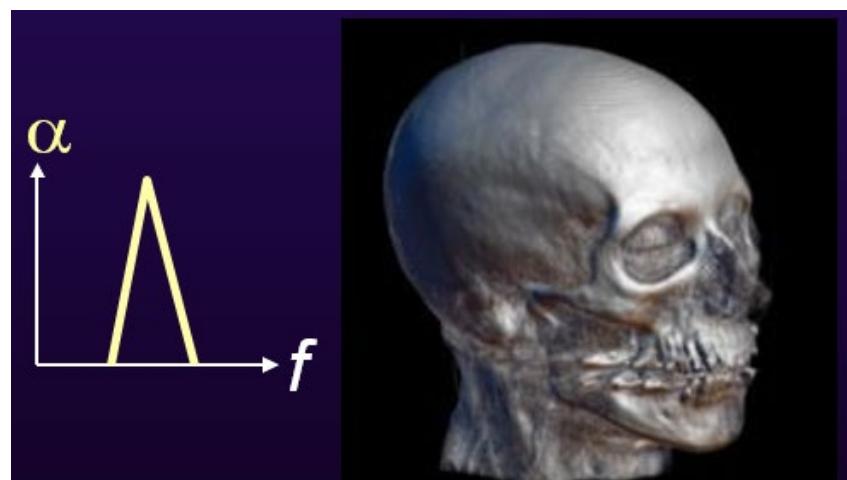
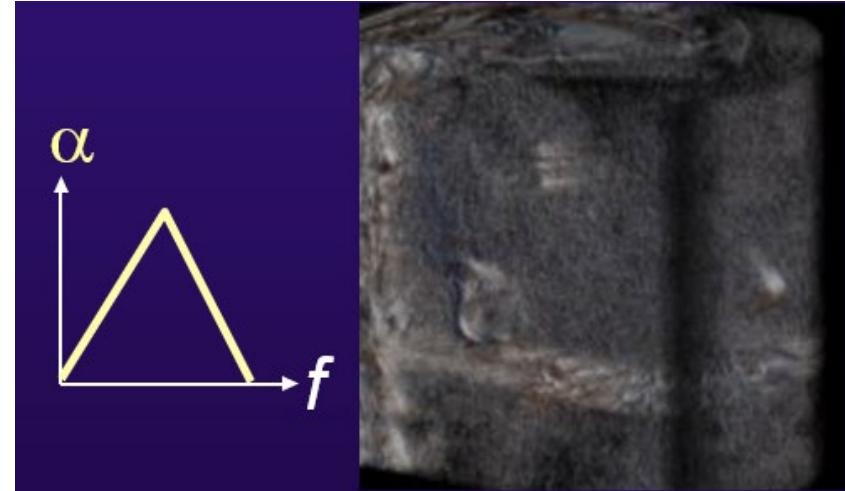
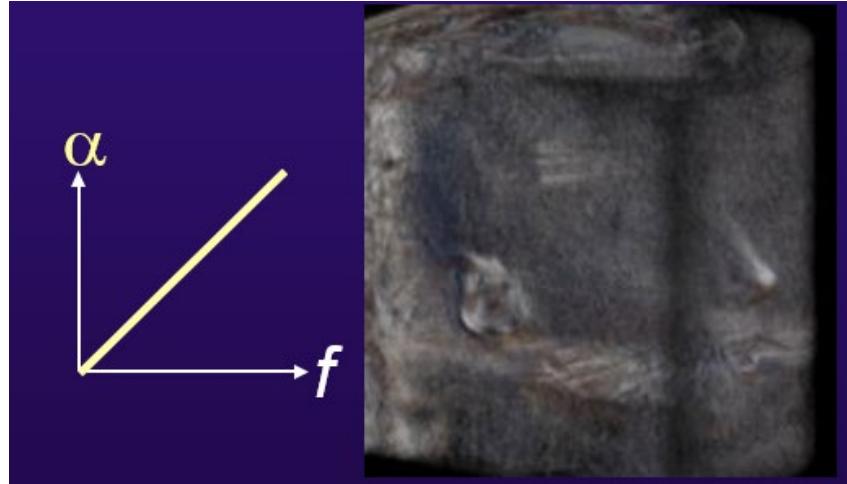
Anything that can be composited with the standard graphics operator “over“

- Opacity: “opacity functions”
 - Most important
- Color
 - Can help distinguish features
- Emittance
- Phong parameters (k_a , k_d , k_s)
- Index of refraction

$$\mathbf{I} = \sum_{i=1}^n \mathbf{C}_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$

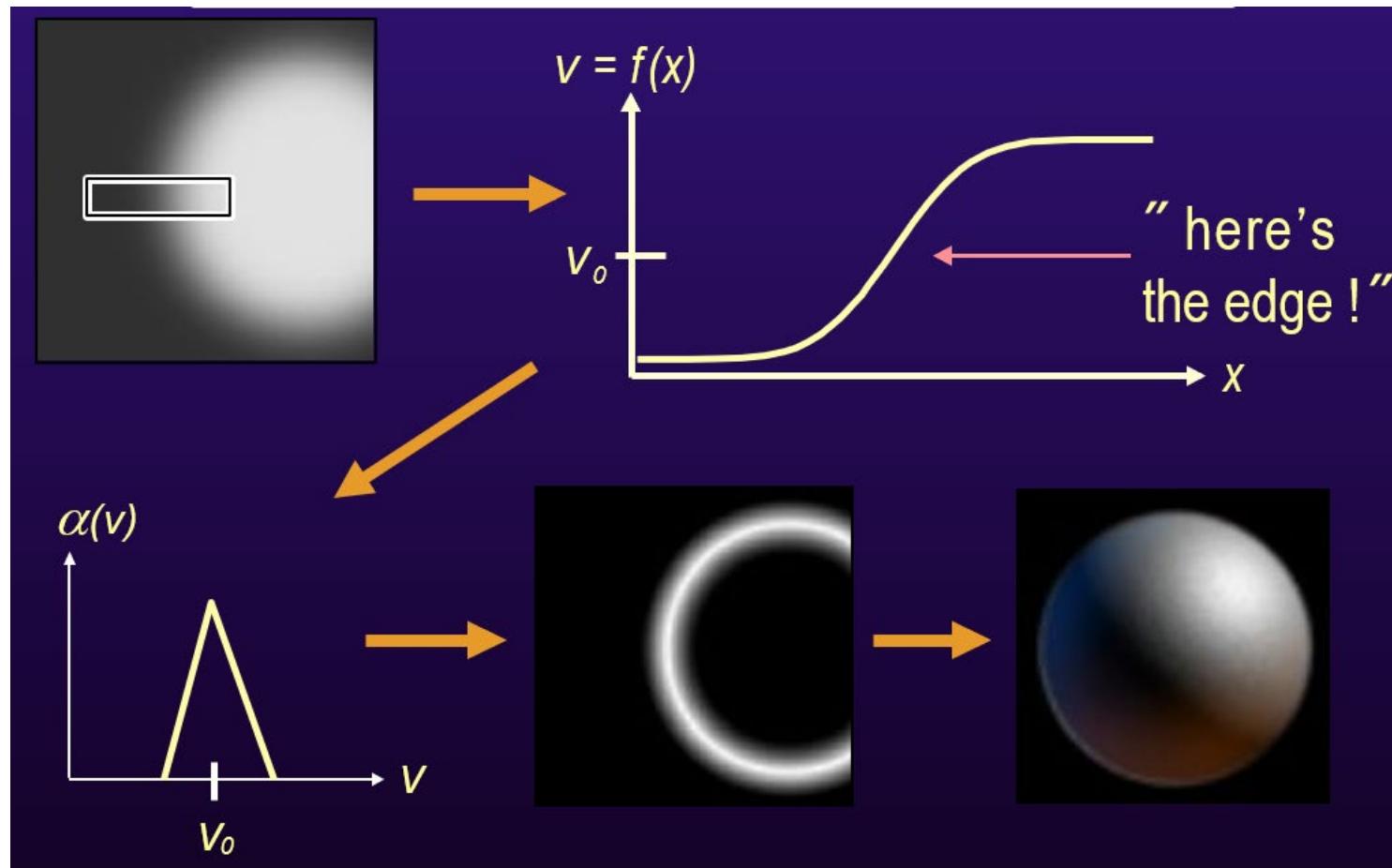
$$\begin{aligned}\mathbf{C}'_{i+1} &= \mathbf{C}'_i + (1 - \alpha'_i) \mathbf{C}_i \alpha_i \\ \alpha'_{i+1} &= \alpha'_i + (1 - \alpha'_i) \alpha_i\end{aligned}$$

Setting Transfer Functions is difficult...unintuitive and slow

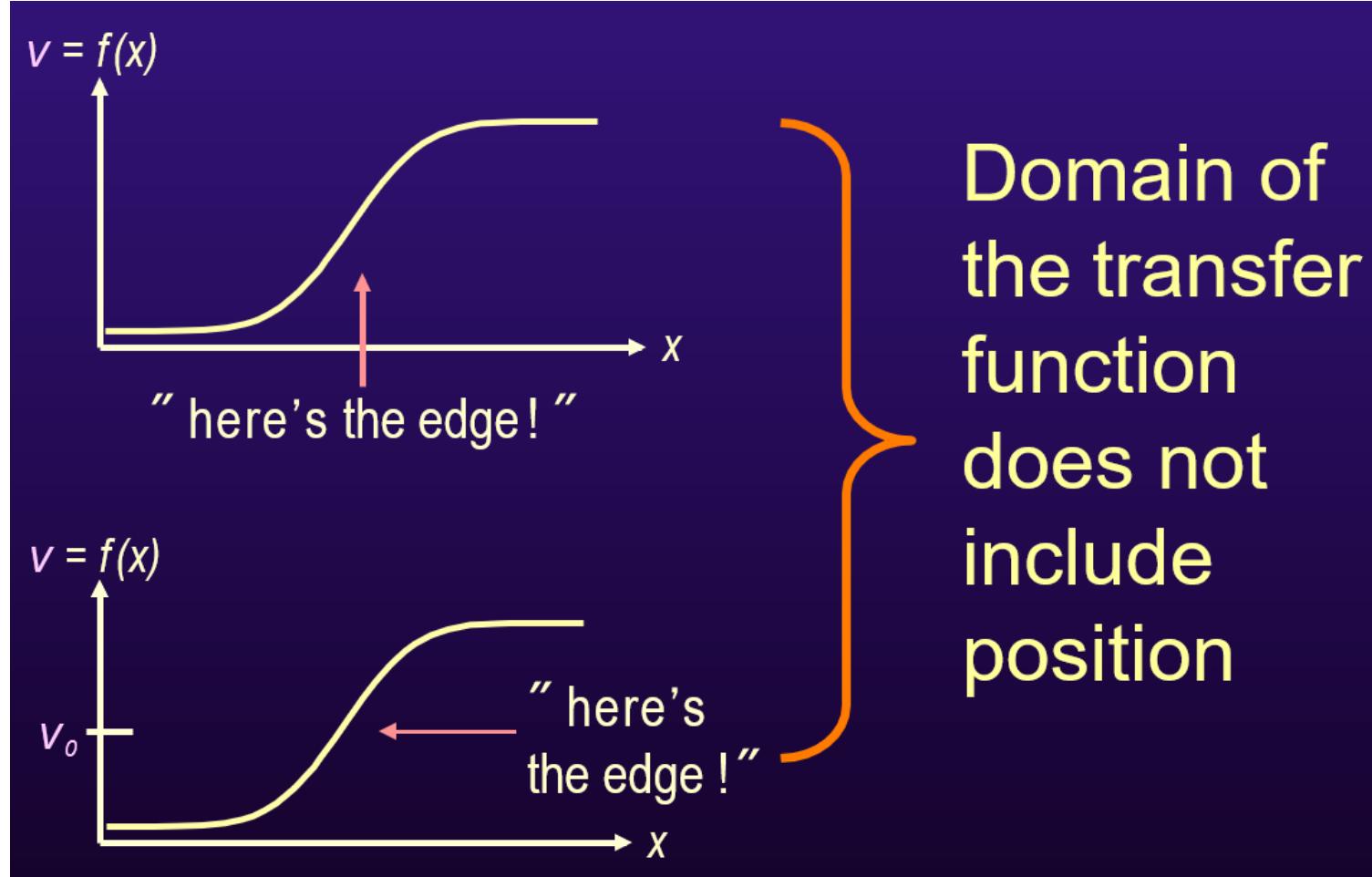


Transfer Functions as Feature Detection

Where's the edge or surface?



Why are Transfer Functions Difficult to Create?



Consider creating a function to segment multiple features

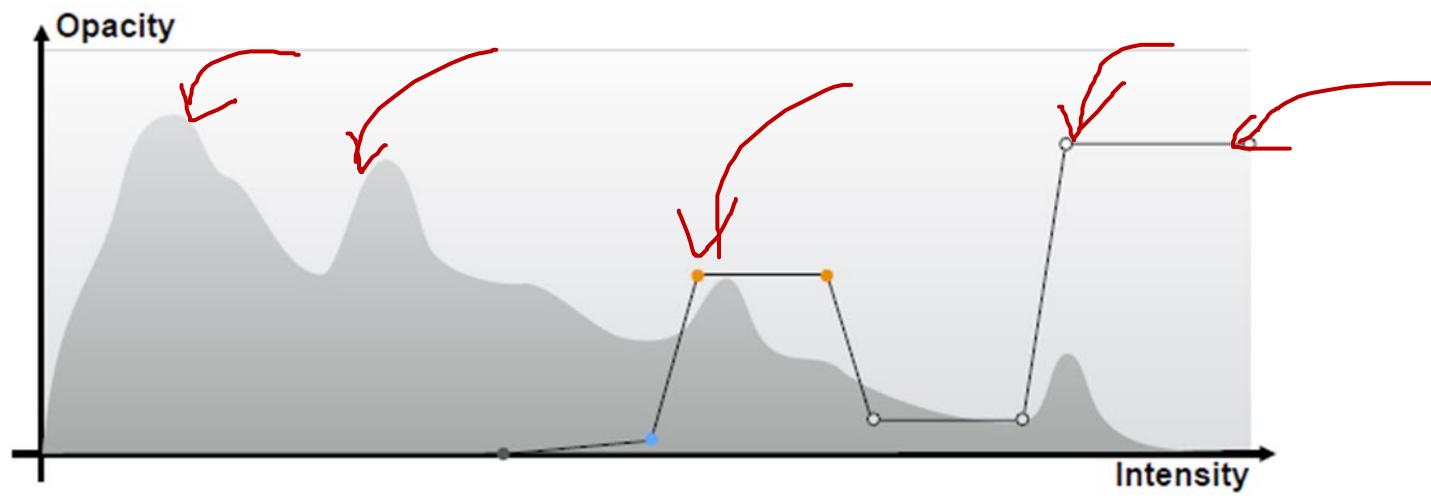
Consider having to create a multi-dimensional transfer function

Transfer Function Taxonomy

There are many ways to categorize transfer functions.

One common approach is data-centric.

What is the dimensionality of the data the function operates on?



A standard user interface for 1D TF settings. Color and opacity are assigned to data ranges using piecewise linear TF widgets. The background represents the histogram of binned scalar attribute data.

1D Transfer Functions

1D TFs map a scalar value to a visual representation...usually color and opacity

Some suitable applications for 1D TFS include:

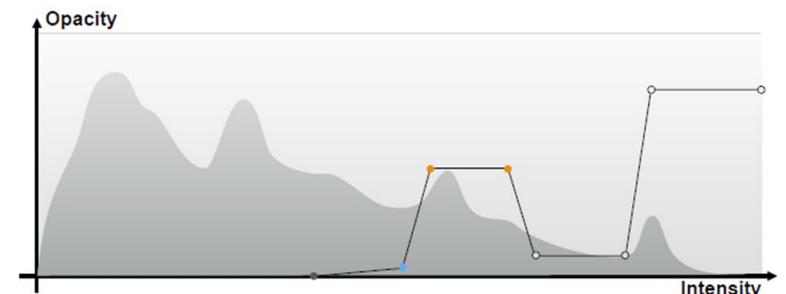
- Industrial CT scan: density data with few overlapping intensity ranges
- Simulation data which is typically low-noise or noise-free

For most medical image data, 1D TF is inadequate:

- tissues have significant overlap in the intensity range, as described
- medical data is measured and relatively noisy

These characteristics negatively impact ability of 1D TFs to correctly classify tissue types.

1D TFs still often used in these applications because higher dimensional work is so hard



MD Transfer Functions

TF operating on an input with more than one dimension is:

- a 2D TF (for a bi-variate input)
- An MD TF (for an input of multiple dimensions)

We can distinguish between TFs are that separable or are intrinsically high-dimensional.

A separable 2D TF is defined as:

two separate 1D functions that are combined only after both 1D functions have been applied separately

Separable 2D TF

$$\mathbf{q}_{\text{separable}}(d_1, d_2) = \mathbf{V}(\mathbf{M}(d_1), d_2)$$

\mathbf{M} classifies material and \mathbf{V} generates a visual mapping (e.g. rgba value)

Example:

Gradient-based opacity modulation in which the second dimension is used to improve visual appearance.

This suppresses interior homogeneous material regions and enhances boundaries.

Essentially, a 1D TF is applied first, followed by multiplying the opacity with a 1D function of gradient magnitude.

Non-Separable 2D TF

$$\mathbf{q}_{\text{non-separable}}(d_1, d_2) = \mathbf{V}(\mathbf{M}(d_1, d_2))$$

M classifies material and V generates a visual mapping (e.g. rgba value)

Example:

Simultaneous value and gradient magnitude mapping presented in

KNISS J., KINDELMANN G., HANSEN C.: Multidimensional transfer functions for interactive volume rendering. IEEE TVCG 8, 3 (July 2002)

9 Multi-Dimensional Transfer Functions for Volume Rendering

JOE KNISS, GORDON KINDELMANN, and CHARLES HANSEN

Scientific Computing and Imaging Institute
School of Computing, University of Utah

9.1 Introduction

Direct volume-rendering has proven to be an effective and flexible visualization method for 3D scalar fields. Transfer functions are fundamental to direct volume-rendering because their role essentially to make the data visible: by assigning optical properties like color and opacity to the voxel data, the volume can be rendered with traditional computer graphics methods. Good transfer functions reveal the important structures in the data without obscuring them with unimportant regions. To date, transfer functions have generally been limited to 1D domains, meaning that the 1D space of scalar data value has been the only variable to which opacity and color are assigned. One aspect of direct volume-rendering that has received little attention is the use of multidimensional transfer functions.

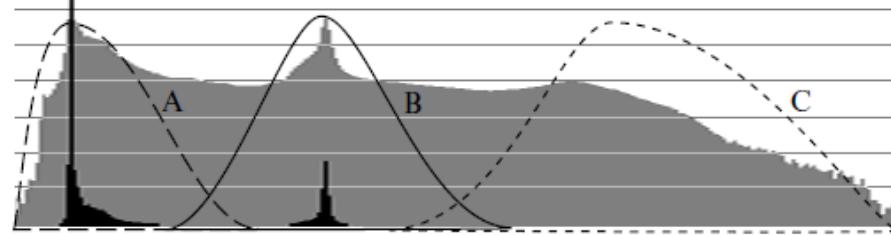
Often, there are features of interest in volume data that are difficult to extract and visualize with 1D transfer functions. Many medical datasets created from CT or MRI scans contain a complex combination of boundaries between multiple materials. This situation is problematic for 1D transfer functions because of the potential for overlap between the data-value intervals spanned by the different boundaries. When one data value is associated with multiple boundaries, a 1D transfer function is unable to render them in isolation. Another benefit of higher dimensional transfer functions is their ability to portray subtle variations in properties of a

single boundary, such as thickness. When working with multivariate data, a similar difficulty arises with features that can be identified only by their unique combination of multiple data values. A 1D transfer function is simply not capable of capturing this relationship.

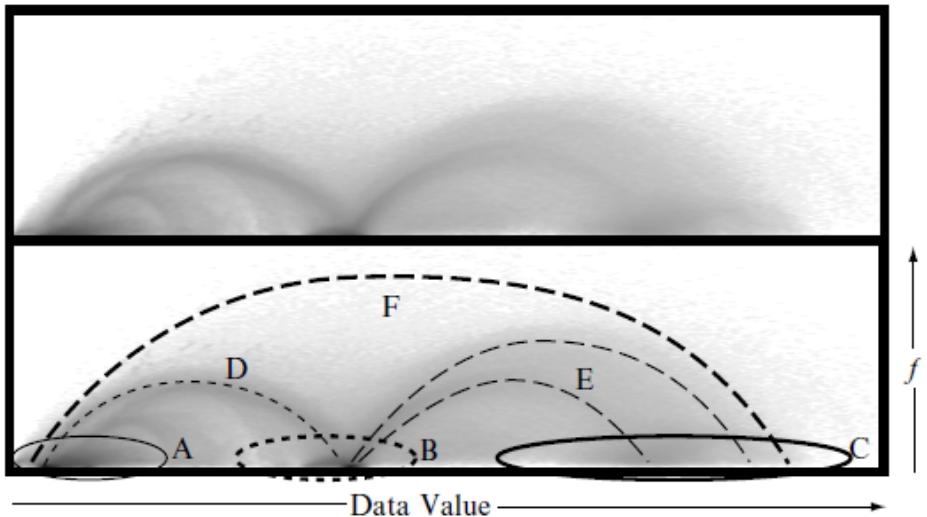
Unfortunately, using multidimensional transfer functions in volume rendering is complicated. Even when the transfer function is only 1D, finding an appropriate transfer function is generally accomplished by trial and error. This is one of the main challenges in making direct volume-rendering an effective visualization tool. Adding dimensions to the transfer-function domain only compounds the problem. While this is an ongoing research area, many of the proposed methods for transfer-function generation and manipulation are not easily extended to higher-dimensional transfer functions. In addition, fast volume-rendering algorithms that assume the transfer function can be implemented as a linear lookup table (LUT) can be difficult to adapt to multidimensional transfer functions due to the linear interpolation imposed on such LUTs.

This chapter provides a detailed exposition of the multidimensional transfer function concept, a generalization of multidimensional transfer functions for both scalar and multivariate data, as well as a novel technique for the interactive generation of volumetric shadows. To resolve the potential complexities in a user interface for multidimensional transfer functions, we introduce a set of direct manipulation widgets

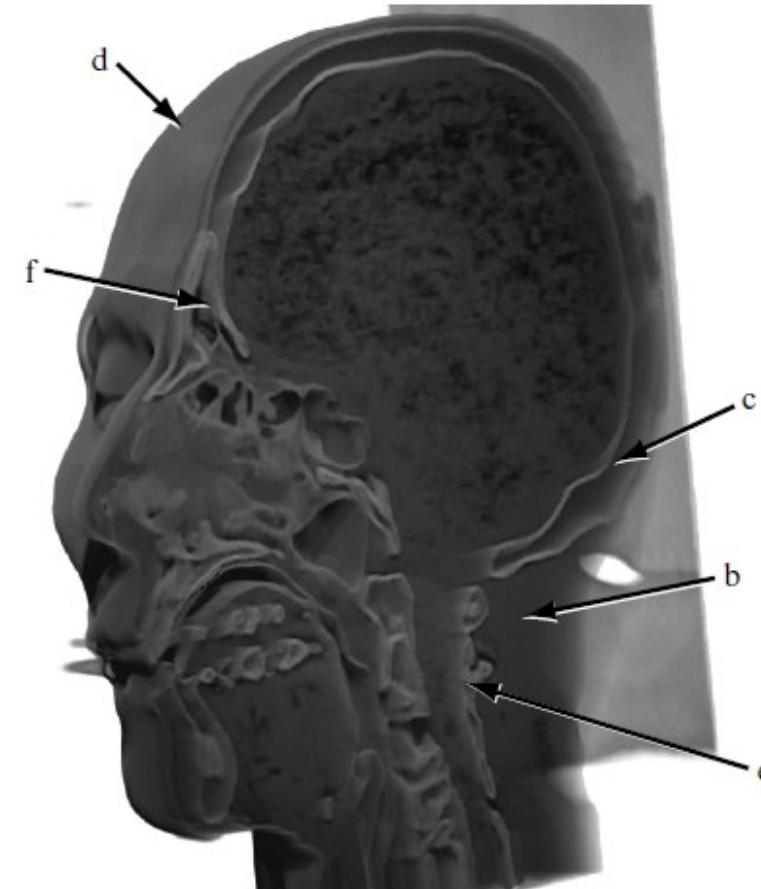
2D Transfer Function Example



(a) A 1D histogram. The black region represents the number of data value occurrences on a linear scale; the grey is on a log scale. The colored regions (A,B,C) identify basic materials.

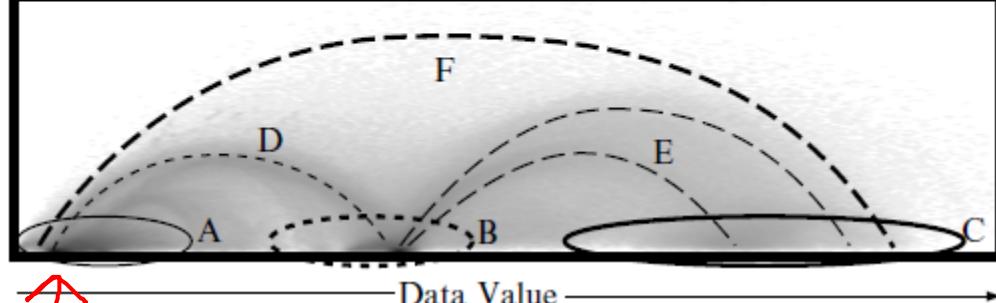
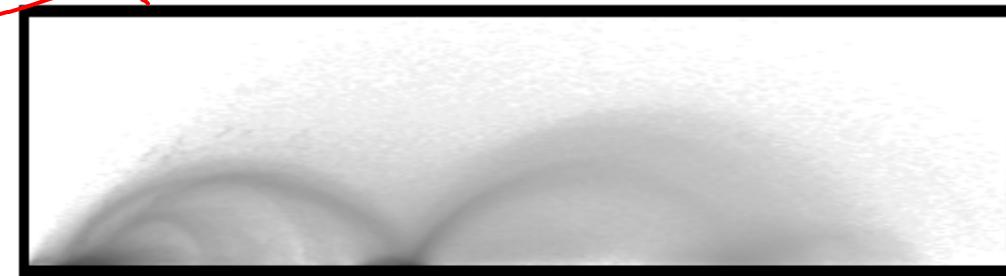
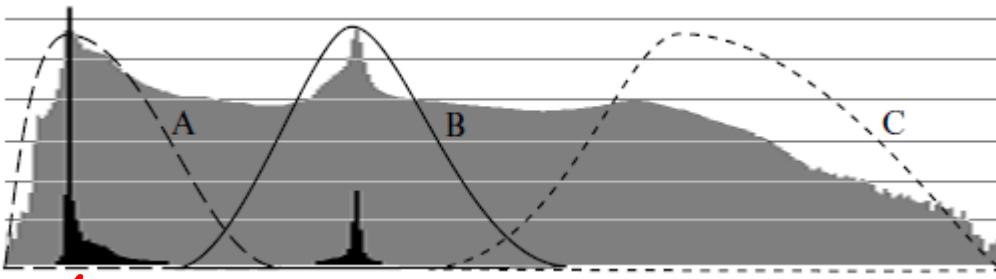


(b) A log-scale 2D joint histogram. The lower image shows the location of materials (A,B,C), and material boundaries (D,E,F).



(c) A volume-rendering showing all of the materials and boundaries identified above, except air (A), using a 2D transfer function.

Materials in CT Scan



The air–bone boundary, example of a surface that cannot be isolated using a simple 1D transfer function.
This type of boundary appears in CT datasets as the sinuses and mastoid cells.

A is air
B is soft tissue
C is bone

Density vs gradient magnitude plot

The boundaries between the materials are shown as the arches; air and soft tissue boundary (D), soft tissue and bone boundary (E), and air and bone boundary (F).

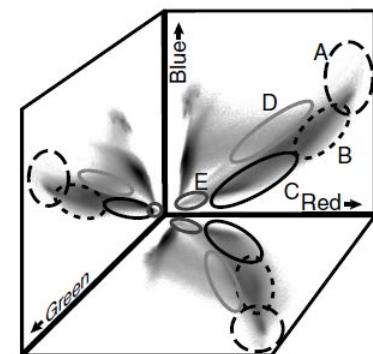
Multidimensional Transfer Functions

Moving beyond 2D TFs poses significant challenges in terms of user interfaces and cognitive comprehension.

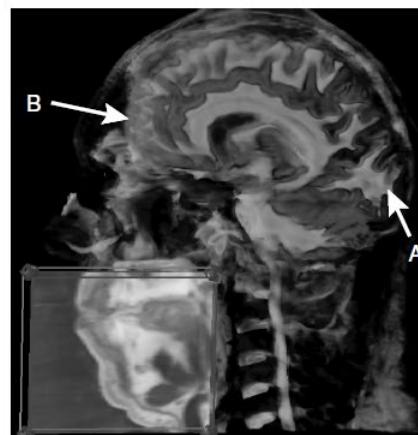
Much research is, therefore, related to various forms of automation

Typical approaches include

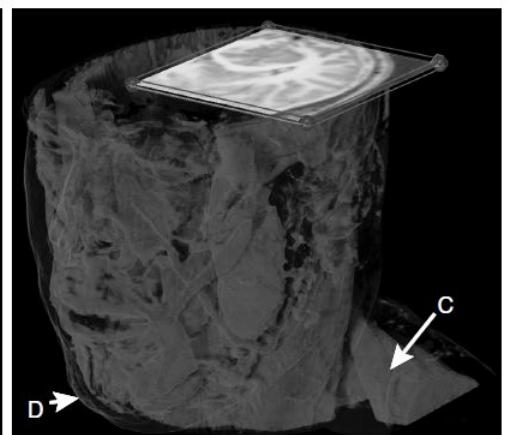
- dimensional reduction,
- clustering and grouping
- machine learning,
- user interfaces such as parallel coordinates



(a) Histograms of the Visible Male RGB dataset



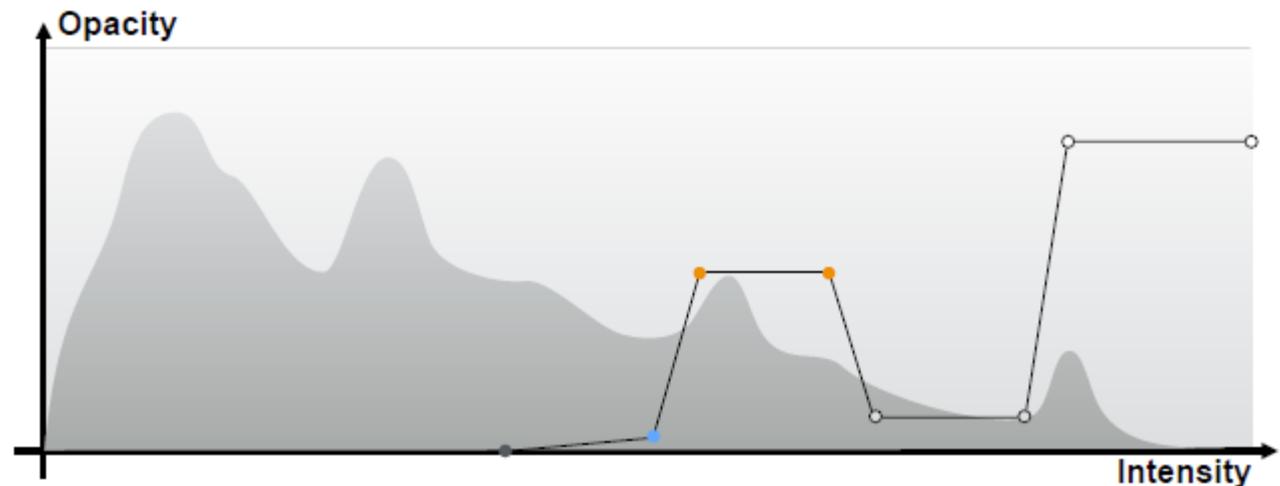
(b) The white (A) and gray (B) matter of the brain

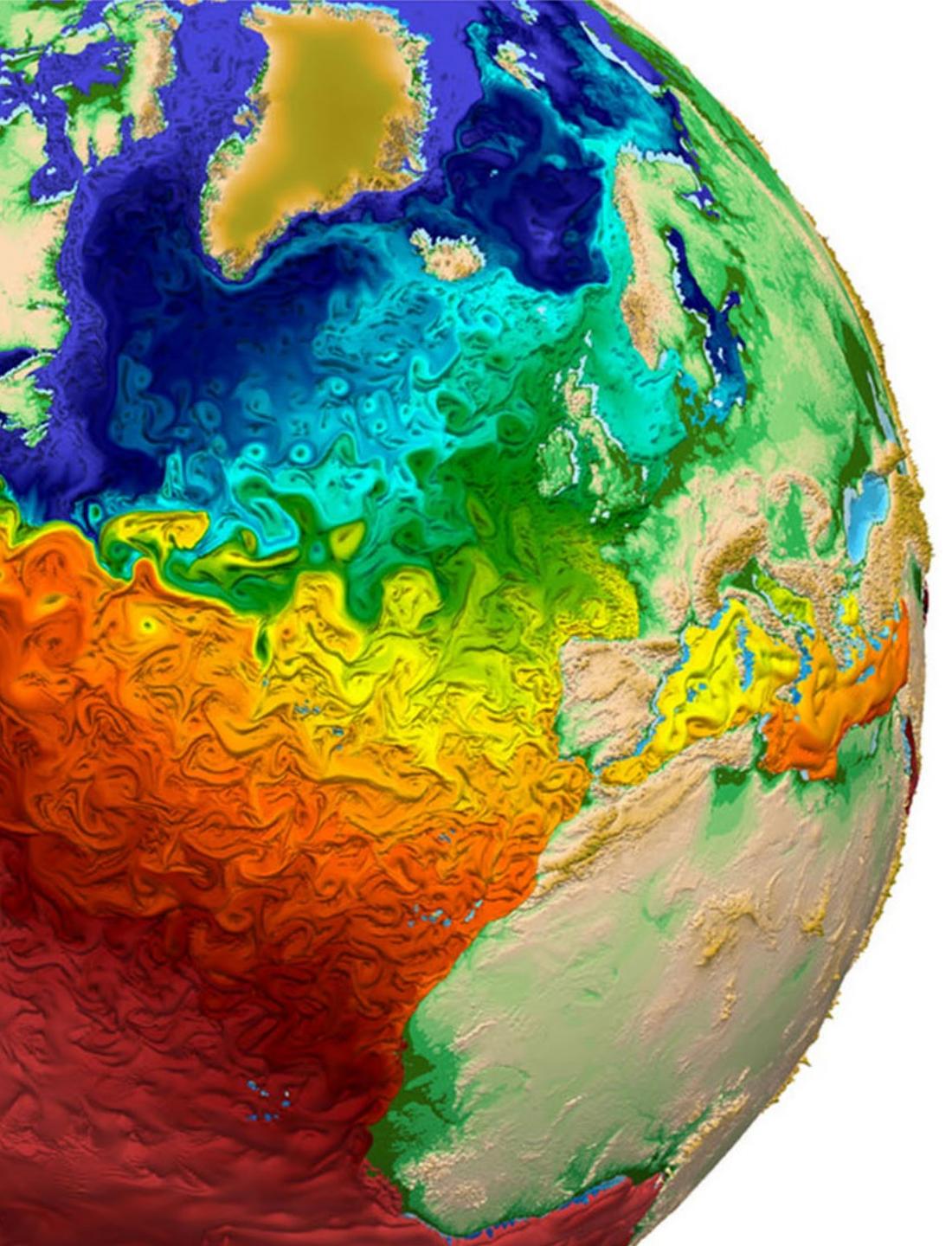


(c) The muscle and connective tissues (C) of the head and neck, showing skin (D) for reference

Transfer Function Research

- 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





Vector Visualization

Introduction Glyphs

Scientific Visualization
Professor Eric Shaffer

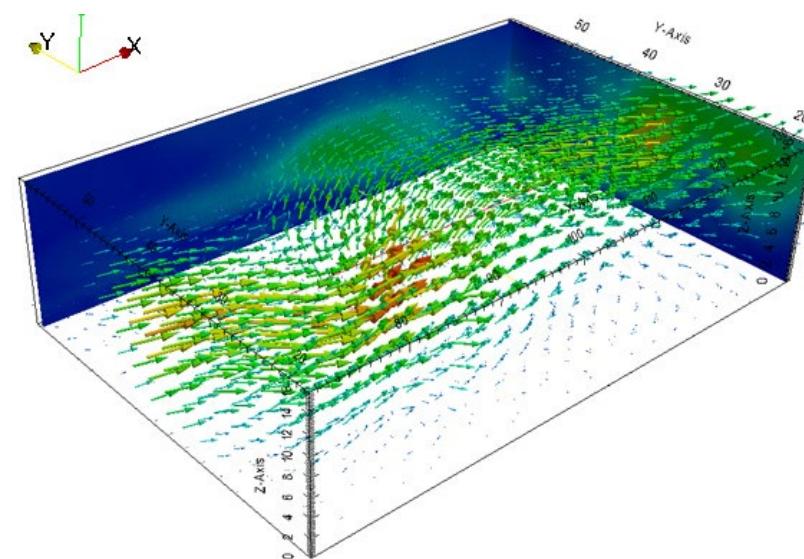
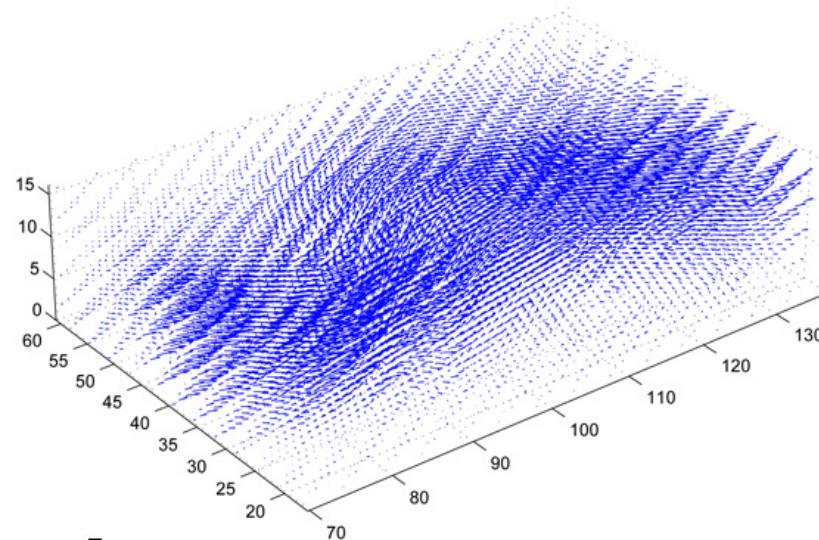
Vector Fields

For each point in a domain we have a vector component and magnitude

- May be discretely sampled over domain
- Often results from study of fluid flow or
- Or by looking at derivatives (rate of change) of some scalar quantity

No visual intuition for what a vector field should look like

Many visualization techniques proposed

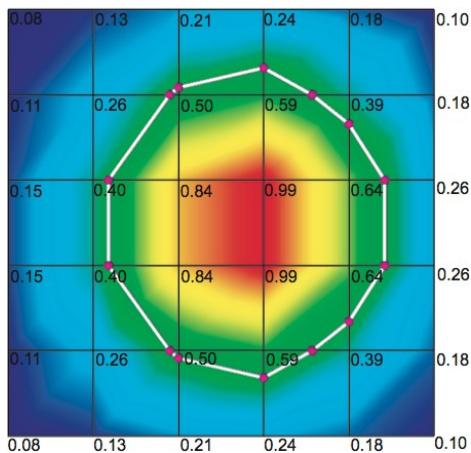


Vector Fields

Input data

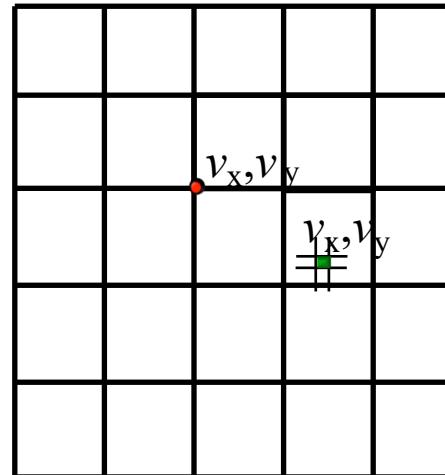
- vector field $v : D \rightarrow \mathbf{R}^n$
 - domain D 2D planar surfaces, 2D surfaces embedded in 3D, 3D volumes
 - for typical scientific application: $n=2$ (fields tangent to 2D surfaces) or $n=3$ (volumetric fields)

Challenging compared with scalar visualization



Scalar visualization

- challenge is to map D to 2D screen
 - after that, we have 1 pixel per scalar value



Vector visualization

- challenge is to map D to 2D screen
 - after that, we have 1 pixel for 2 or 3 scalar values!

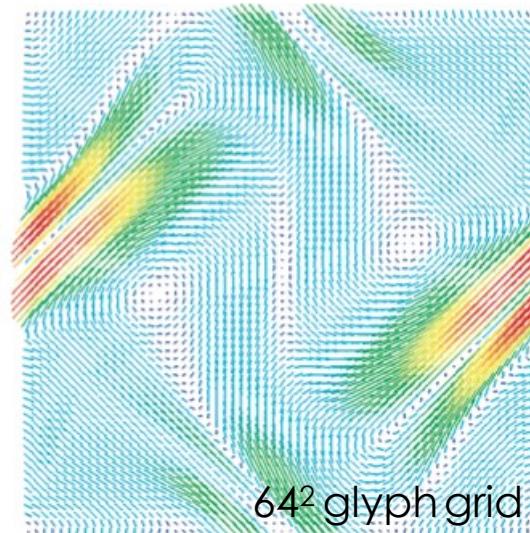
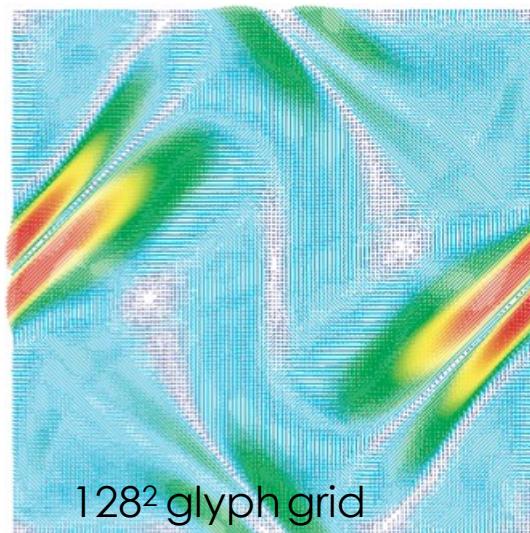
Glyphs

Icons, or signs, for visualizing vector fields

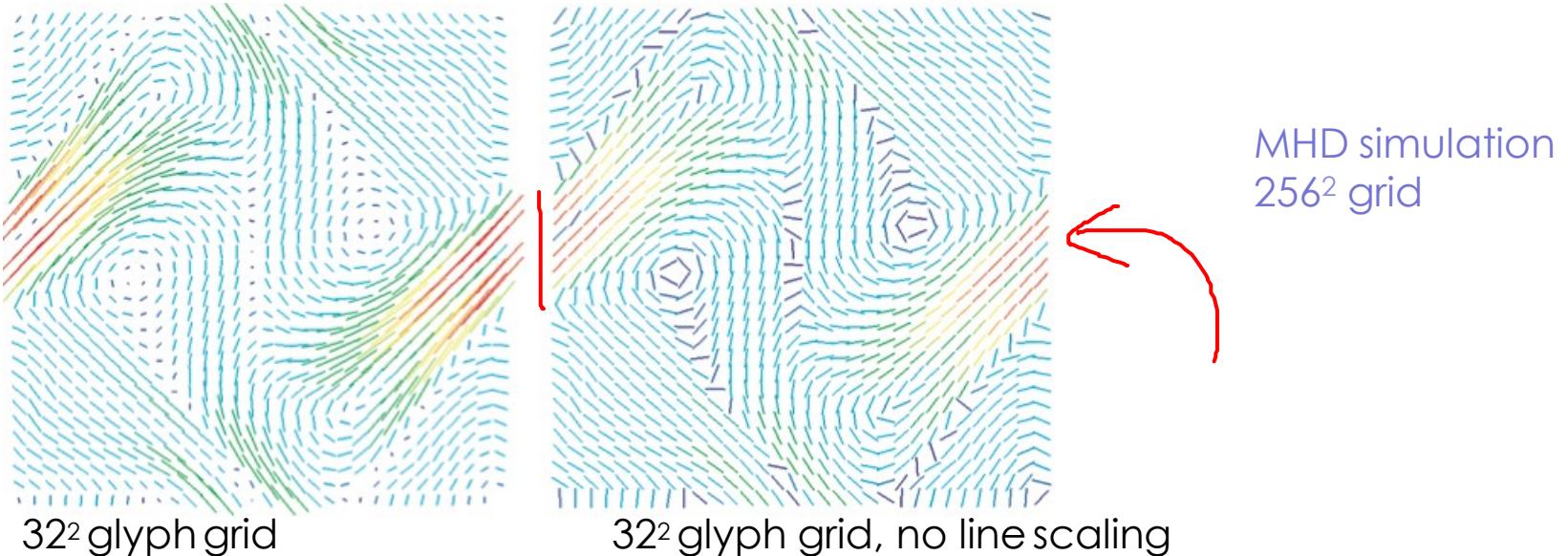
- placed by (sub)sampling the dataset domain
- attributes (scale, color, orientation) map vector data at sample points

Simplest glyph: Line segment (hedgehog plots)

- for every sample point $x \in D$
 - draw line $(x, x + k\mathbf{v}(x))$
 - optionally color map $\|\mathbf{v}\|$ onto it



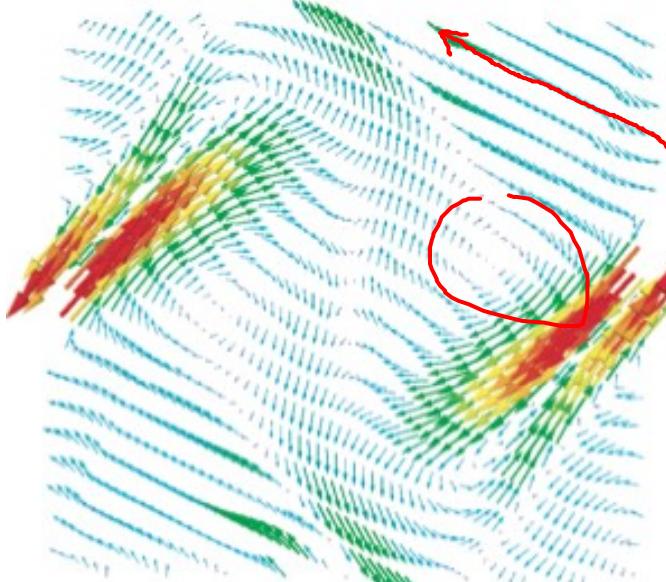
Glyphs



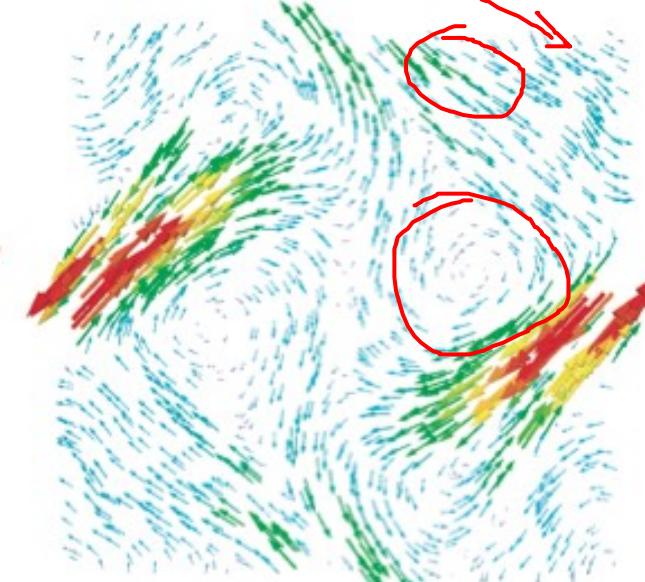
Observations:

- more samples: more data points depicted, but more potential clutter
- fewer samples: fewer data points depicted, but higher clarity
- more line scaling: easier to see high-speed areas, but more clutter
- less line scaling: less clutter, but harder to perceive directions

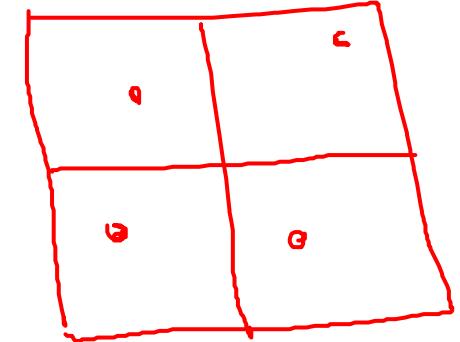
Glyphs



samples on a rotated grid



random samples, quasi-uniform density

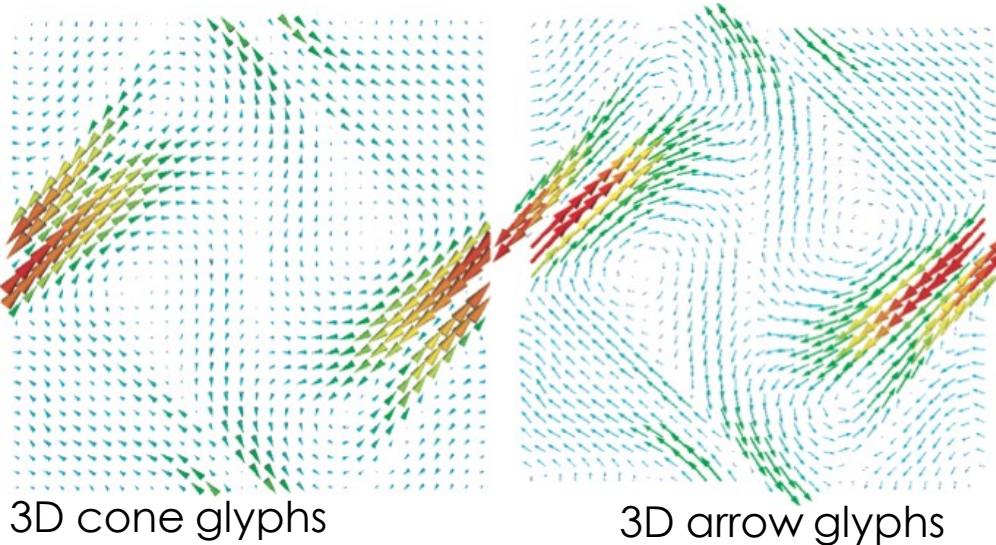


How to choose sample points

- avoid uniform grids!
- random sampling: generally OK

What false impressions does the left plot convey w.r.t. the right plot?

Glyphs



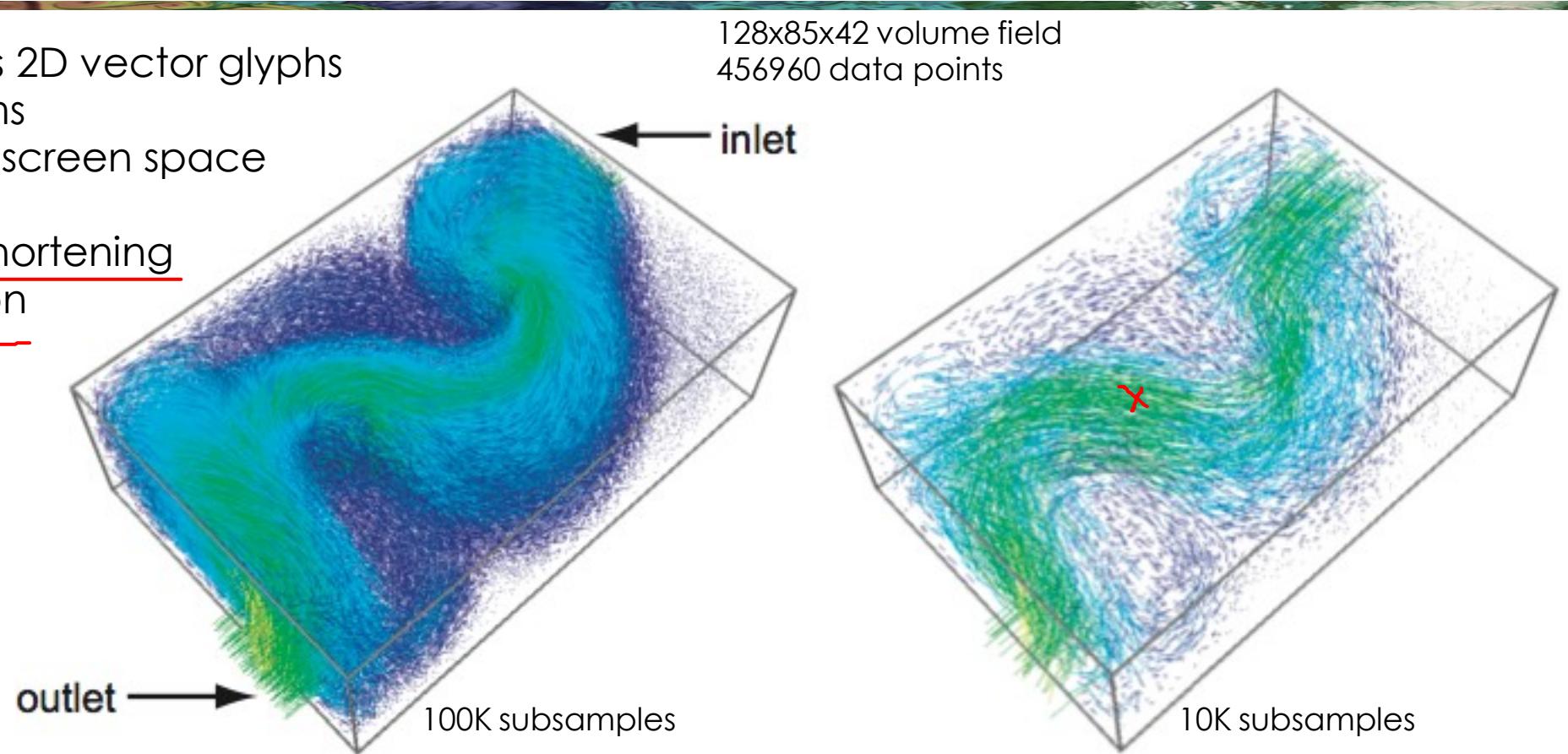
Variants

- cones, arrows, ...
 - show *orientation* better than lines
 - but take more space to render
 - shading: good visual cue to separate (overlapping) glyphs

3D Glyphs

Same idea/technique as 2D vector glyphs

- 3D additional problems
 - more data, same screen space
 - occlusion
 - perspective foreshortening
 - viewpoint selection

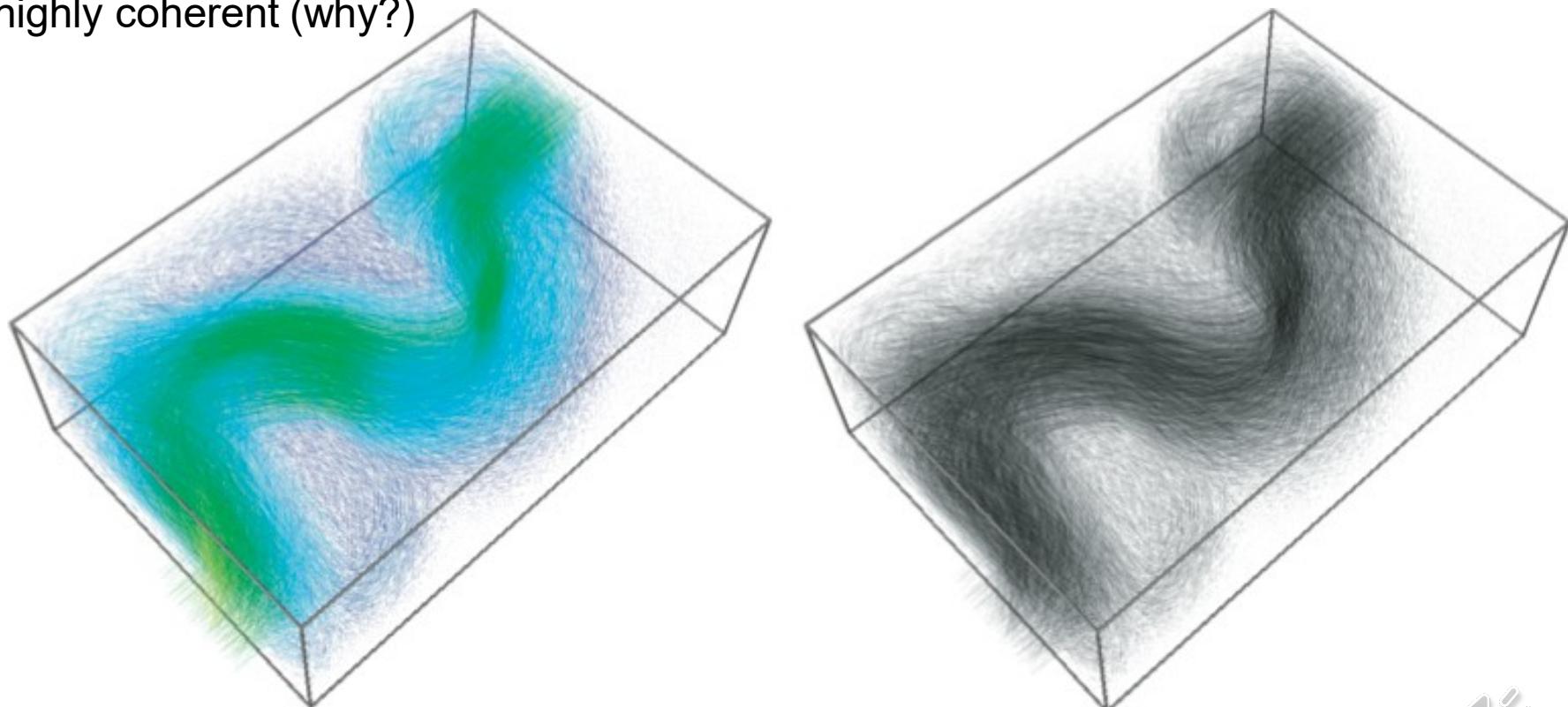


Glyphs in 3D

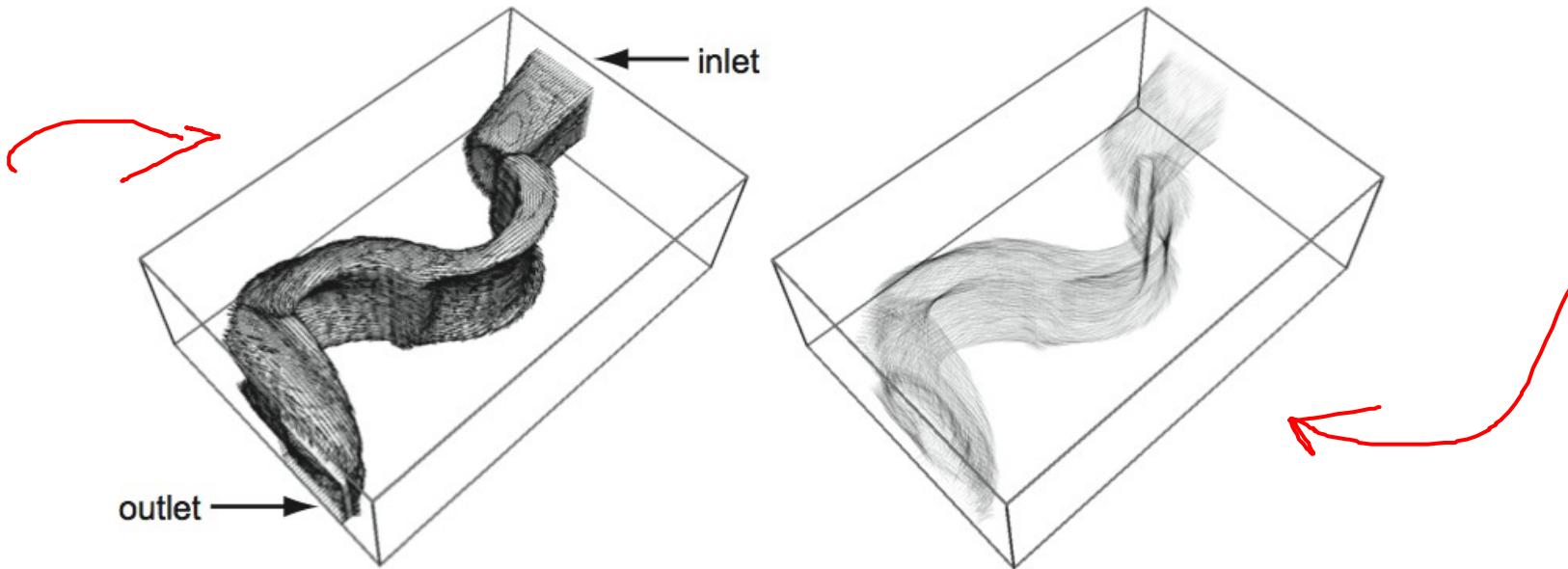
Alpha blending

- extremely simple and powerful tool
- reduce *perceived* occlusion
- low-speed zones: highly transparent
- high-speed zones: opaque and highly coherent (why?)

128x85x42 volume field
456960 data points



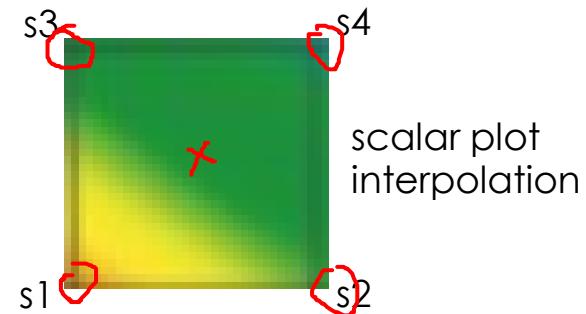
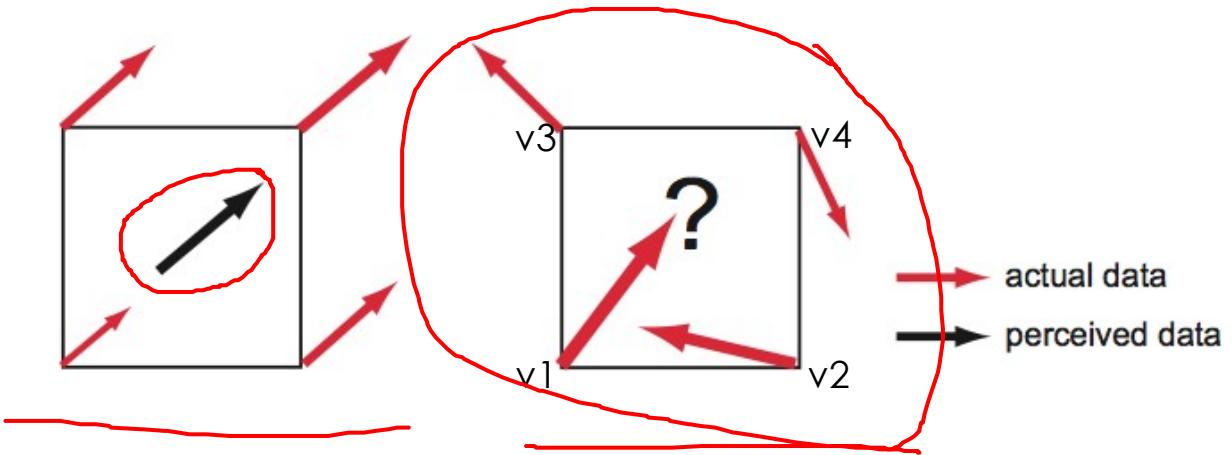
Glyphs on Surfaces



Trade-off between vector glyphs in 2D planes and in full 3D

- find interesting surface
 - e.g. **isosurface** of flow velocity
- plot 3D vector glyphs on it

Problems with Glyphs

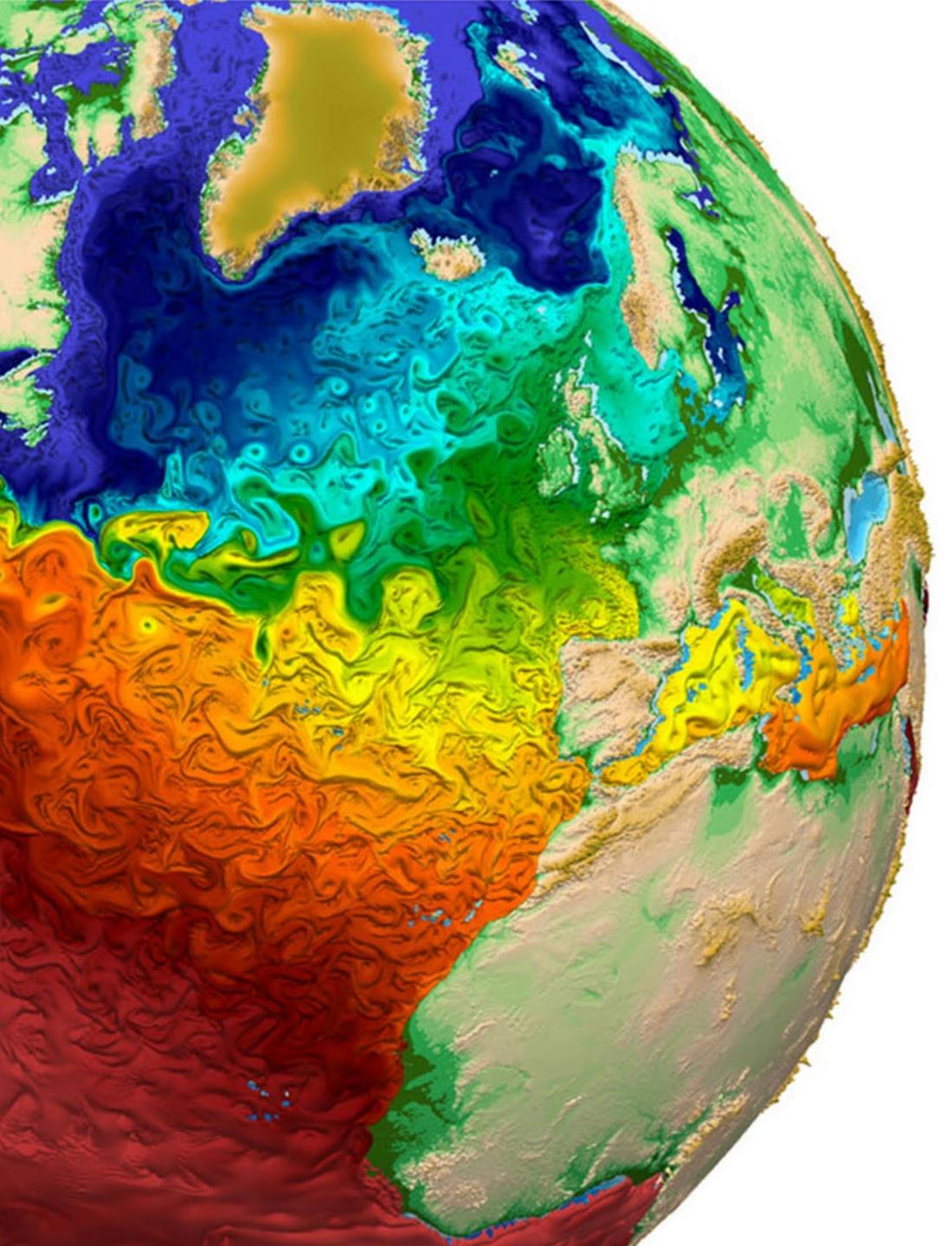


The 'inverse mapping' proposal

- we render something...
- ...so we can visually map it to some data/phenomenon

Glyph problems

- no interpolation in glyph space (unlike for scalar plots with color mapping!)
- a glyph takes more space than a pixel
- we (humans) aren't good at visually interpolating arrows...
- scalar plots are **dense**; glyph plots are **sparse**
 - this is why glyph positioning (sampling) is extra important



Vector Visualization

Derived Quantities

Scientific Visualization
Professor Eric Shaffer

Visualizing Derived Scalar Quantities

Compute derived scalar quantities from vector fields $\underline{\mathbf{v}}(x, y, z) = \langle v_x, v_y, v_z \rangle$

Use known scalar visualization methods for these quantities

Divergence $\text{div } \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}$ equivalent to $\text{div } \mathbf{v} = \lim_{\Gamma \rightarrow 0} \frac{1}{|\Gamma|} \int_{\Gamma} (\mathbf{v} \cdot \mathbf{n}_{\Gamma}) ds$

Example:

$$\begin{aligned}\mathbf{v}(x, y, z) &= \langle xy^2, xy^2, zy \rangle \\ \mathbf{v}(1, 2, 3) &= \langle 4, 4, 6 \rangle\end{aligned}$$

$$\begin{aligned}\text{div } \mathbf{v}(x, y, z) &= y^2 + 2xy + y \\ \text{div } \mathbf{v}(1, 2, 3) &= 4 + 4 + 2 = 10\end{aligned}$$

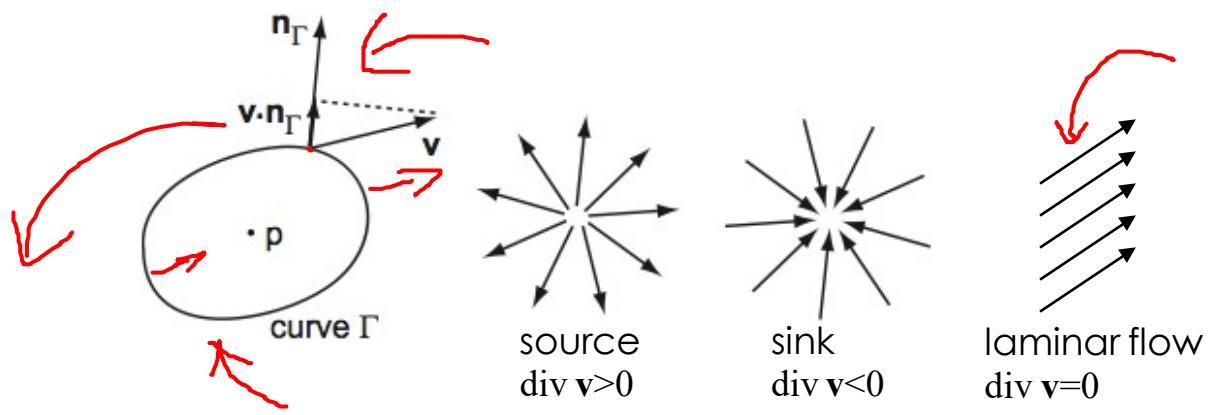
Visualizing Derived Scalar Quantities

Compute derived scalar quantities from vector fields $\mathbf{v}(x, y, z) = \langle v_x, v_y, v_z \rangle$

Use known scalar visualization methods for these quantities

Divergence $\operatorname{div} \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}$ equivalent to $\operatorname{div} \mathbf{v} = \lim_{\Gamma \rightarrow 0} \frac{1}{|\Gamma|} \int_{\Gamma} (\mathbf{v} \cdot \mathbf{n}_{\Gamma}) ds$

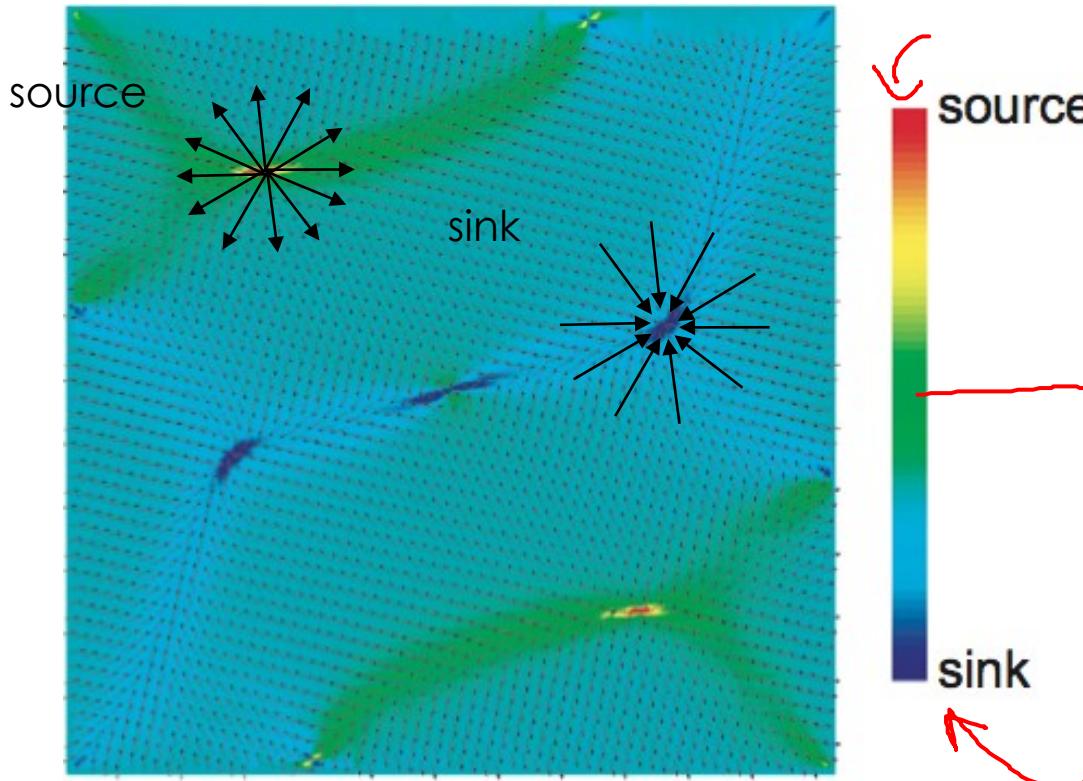
- think of vector field as encoding a fluid flow
- intuition: degree to which the field converges or diverges at a point
- given a field $\mathbf{v} : \mathbf{R}^3 \rightarrow \mathbf{R}^3$, $\operatorname{div} \mathbf{v} : \mathbf{R}^3 \rightarrow \mathbf{R}$ is



$\operatorname{div} \mathbf{v}$ is sometimes denoted as $\nabla \cdot \mathbf{v}$

Example: Divergence

- compute using definition with partial derivatives
- visualize using color mapping



- gives a good impression of where the flow ‘enters’ and ‘exits’ some domain

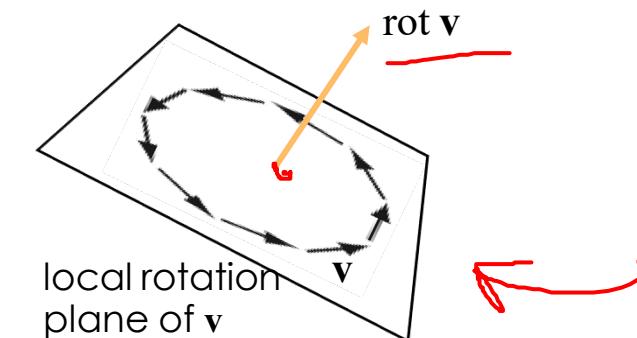
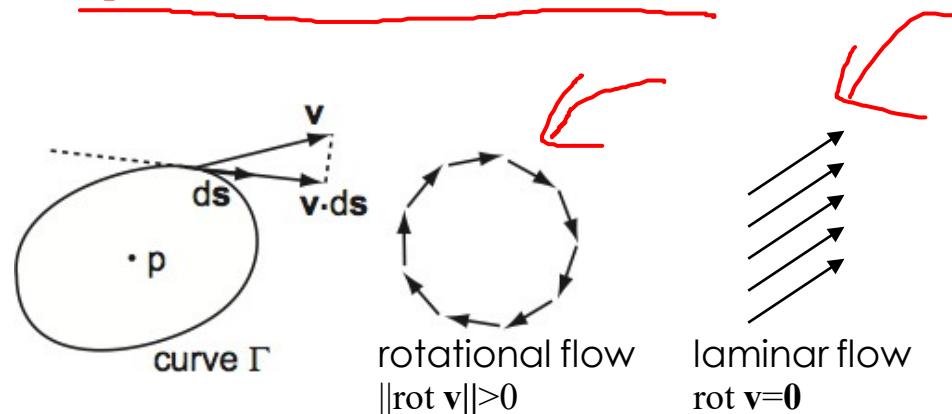
Vorticity

Vorticity or Rotor

- vector quantity...a field locally orthogonal to the plane of rotation
- consider again a vector field as encoding a fluid flow
- intuition: magnitude is how quickly the flow 'rotates' around each point?
- given a field $\mathbf{v} : \mathbf{R}^3 \rightarrow \mathbf{R}^3$, $\text{rot } \mathbf{v} : \mathbf{R}^3 \rightarrow \mathbf{R}^3$ is

produced by taking the curl of the flow field

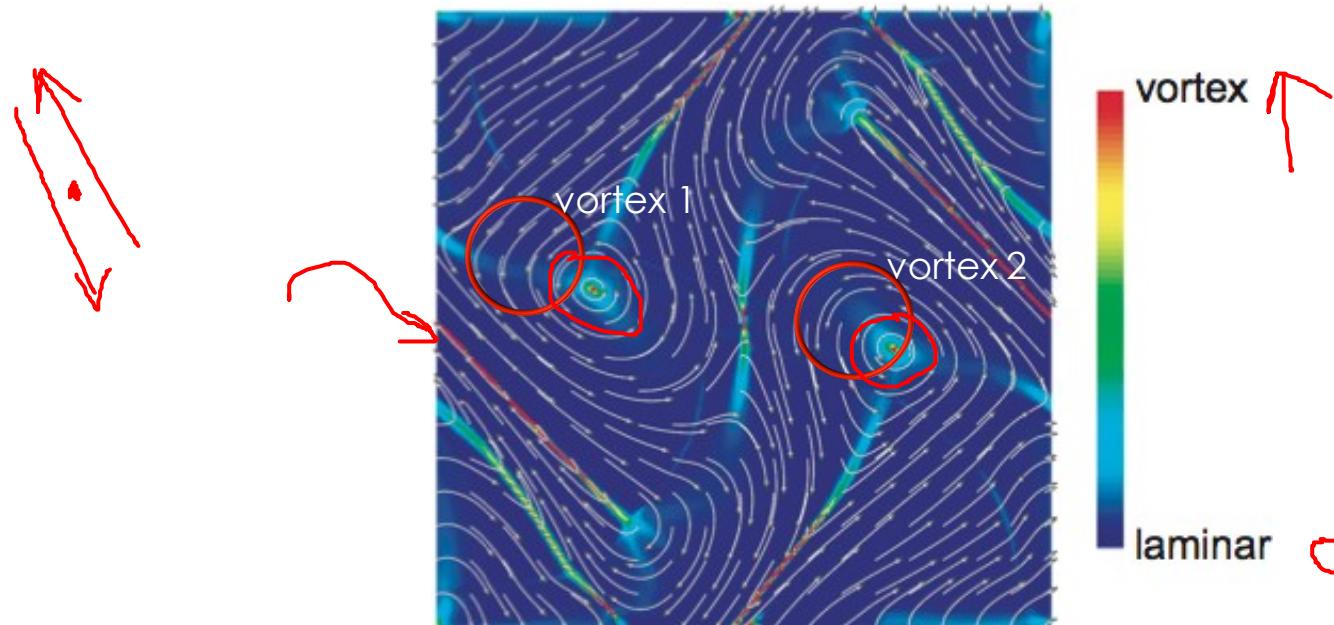
$$\text{rot } \mathbf{v} = \left(\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z}, \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x}, \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right)$$



$\text{rot } \mathbf{v}$ is sometimes denoted as $\nabla \times \mathbf{v}$

Visualizing Vorticity

- compute using definition with partial derivatives
- visualize magnitude $\|\text{rot } \mathbf{v}\|$ using e.g. color mapping



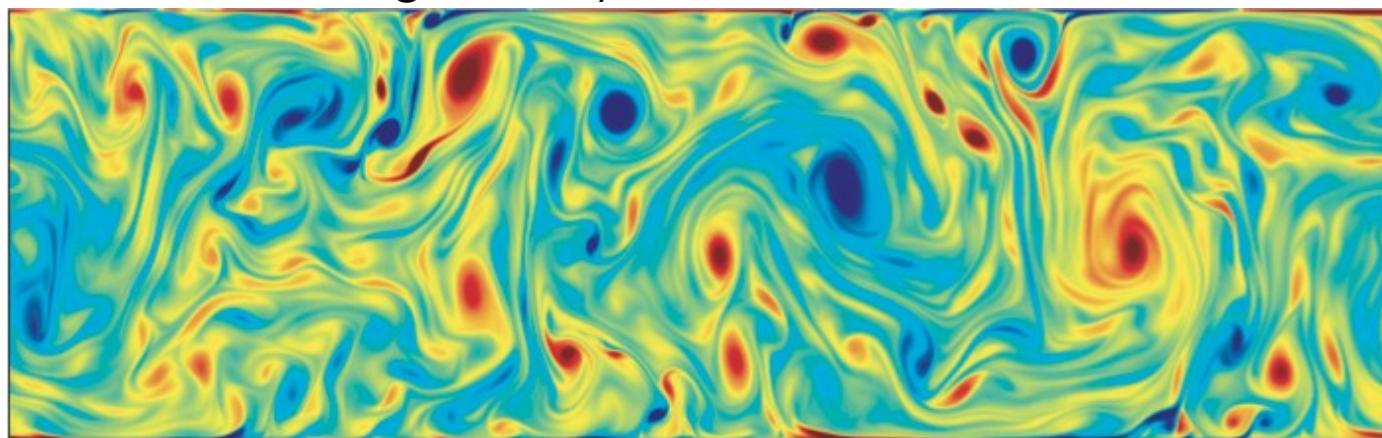
- very useful in practice to find **vortices** = regions of high vorticity
- these are highly important in flow simulations (aerodynamics, hydrodynamics)

Visualizing Vorticity

Example of vorticity

- 2D fluid flow
- simulated by solving Navier-Stokes equations
- visualized using vorticity

Express $\mathbf{v}(x, y) = \langle v_x, v_y, 0 \rangle \rightarrow \text{rot } \mathbf{v} = \left\langle 0, 0, \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right\rangle$



counterclockwise

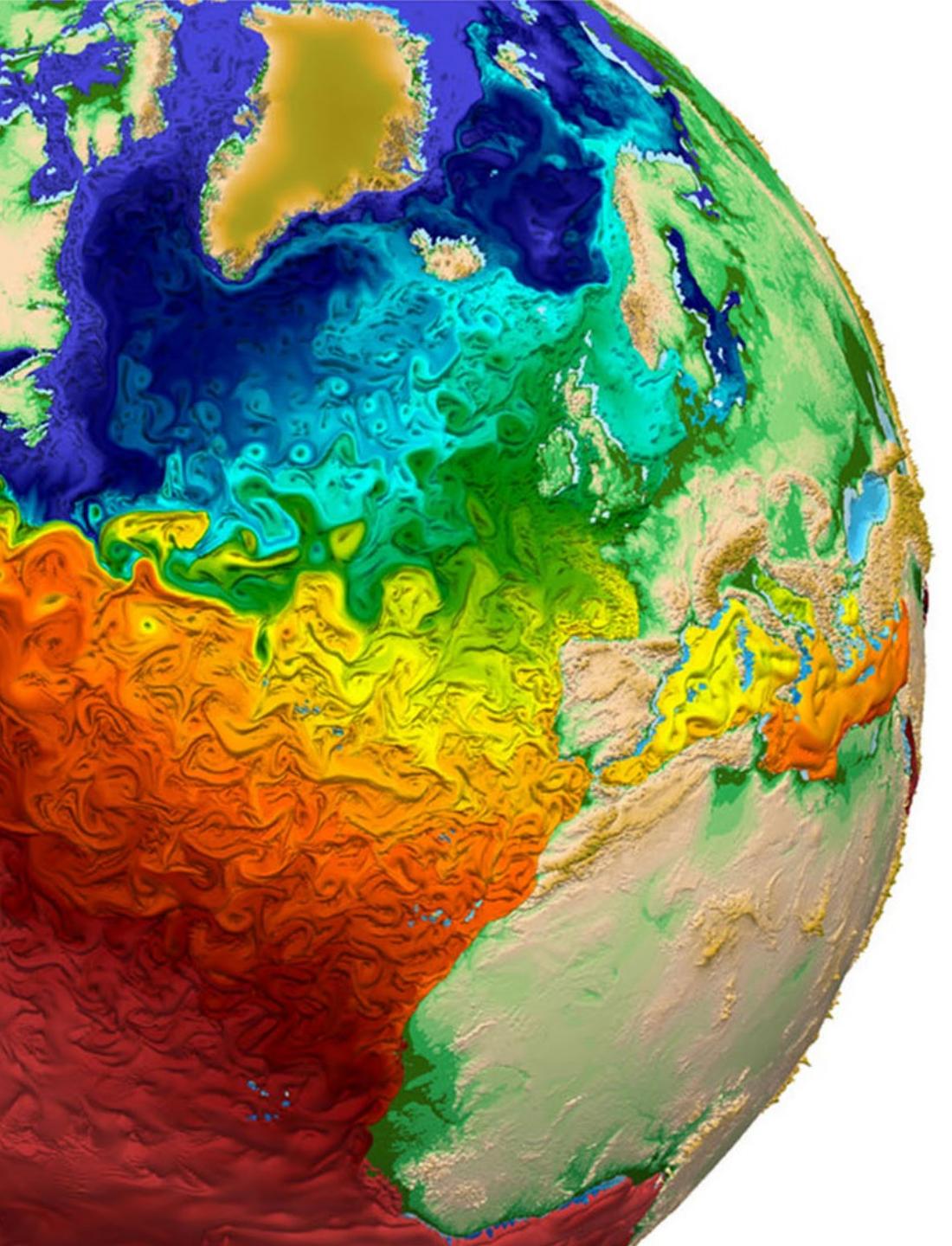
laminar

clockwise

-



+



Vector Visualization

Euler's Method

Scientific Visualization
Professor Eric Shaffer

Flow Fields and Differential Equations

A vector field can be thought of as a field of velocities

Each vector describes velocity at a point...velocity is first derivative of position

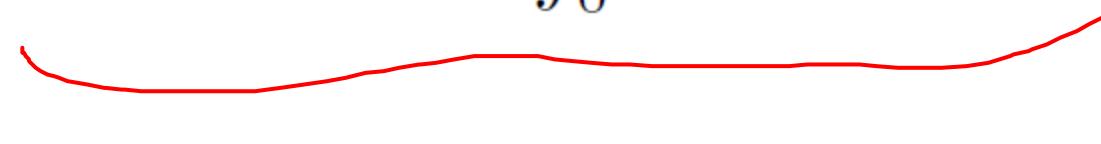
Flow of a vector field tells us where a particle ends up

- after a certain time
- given a particular starting point

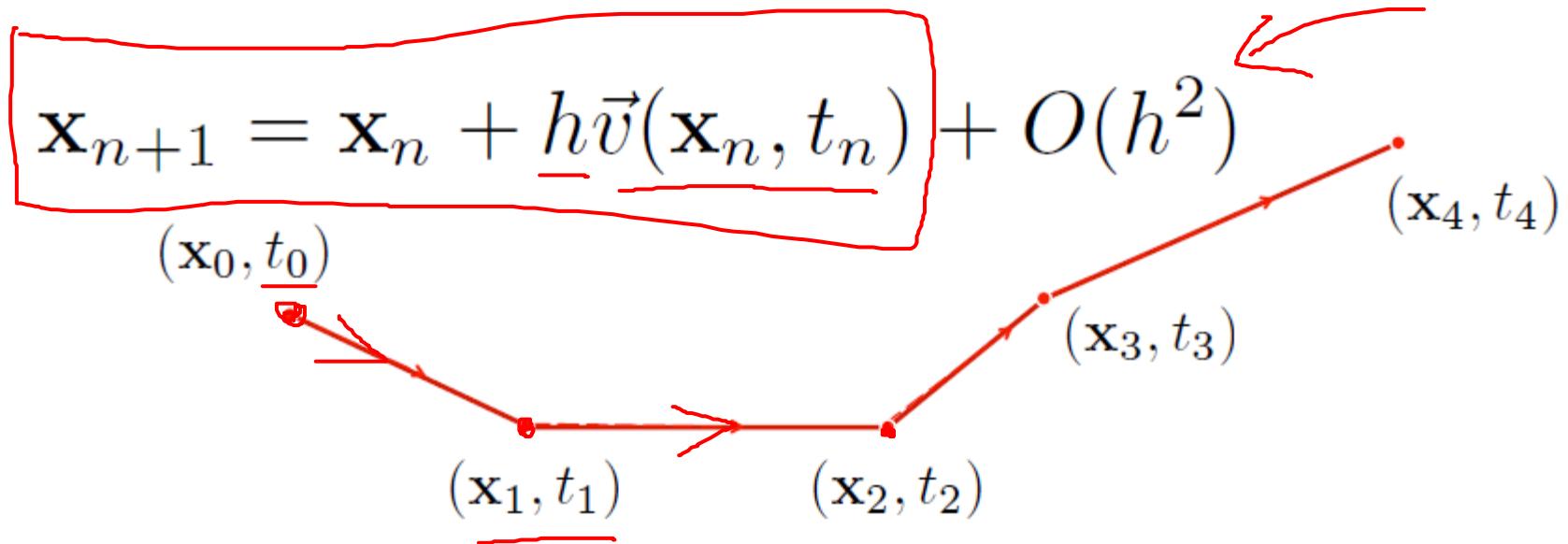
Requires the solution of an ordinary differential equation (ODE)

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_0^t \vec{v}(\mathbf{x}(u)) du$$

Analytical solution generally does not exist...need to use a numerical method



Euler's Method



- Simple
- Fast
- Inaccurate
- Unstable

But beloved in computer
graphics where things just
need to look good...in
visualization they should
probably be accurate as
well.

Understanding Error

Two sources of error

1. Rounding error

- due to the finite precision of floating-point arithmetic

2. Truncation error (or discretization error)

- e.g. approximating an infinite process with a finite number of steps
- For ODEs truncation error is usually dominant
- The two types error are not independent
- Reducing the step-size will typically reduce truncation error but may increase rounding error

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\vec{v}(\mathbf{x}_n, t_n) + O(h^2)$$

Truncation Error

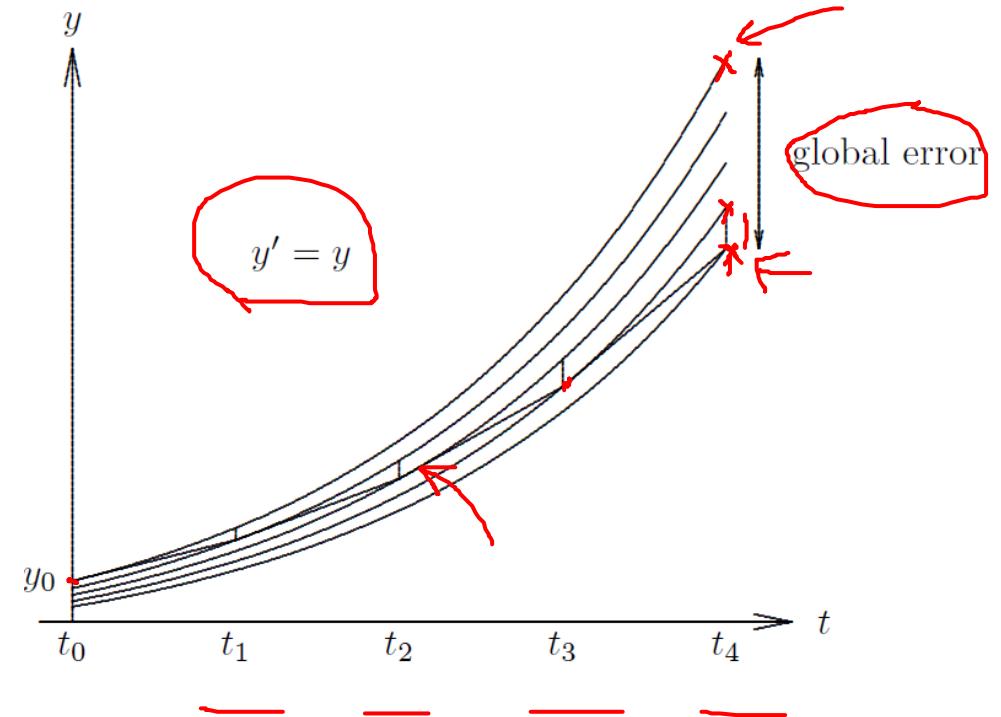
At kth step

- Global error is the cumulative overall error

$$e_k = y_k - y(t_k)$$

- Local error is the error in one step

$$\ell_k = \underline{y_k} - u_{k-1}(t_k)$$



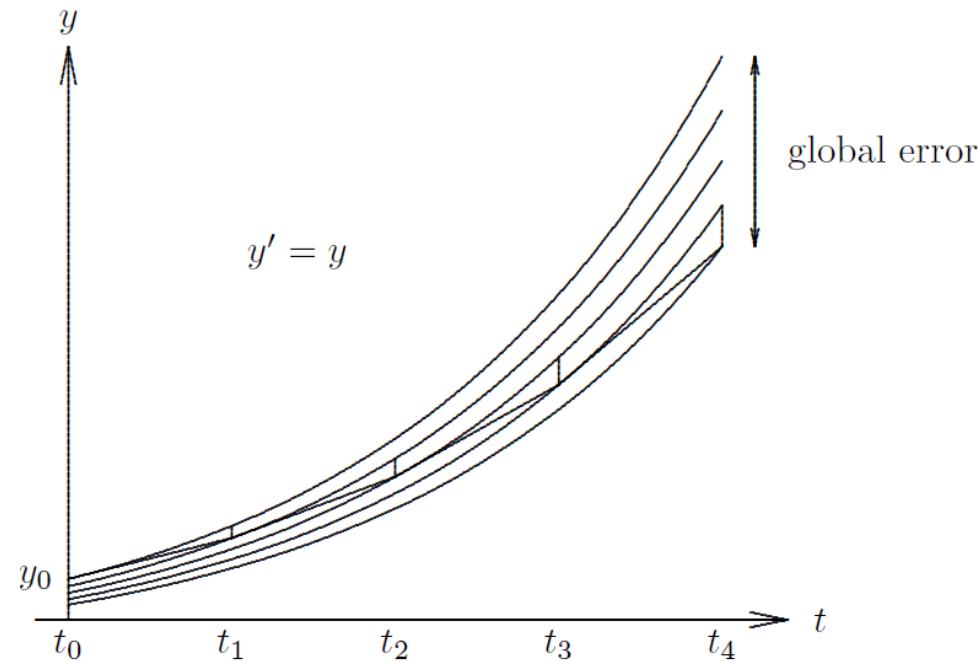
Truncation Error for Euler's method

- Global error $O(h)$

$O(h)$

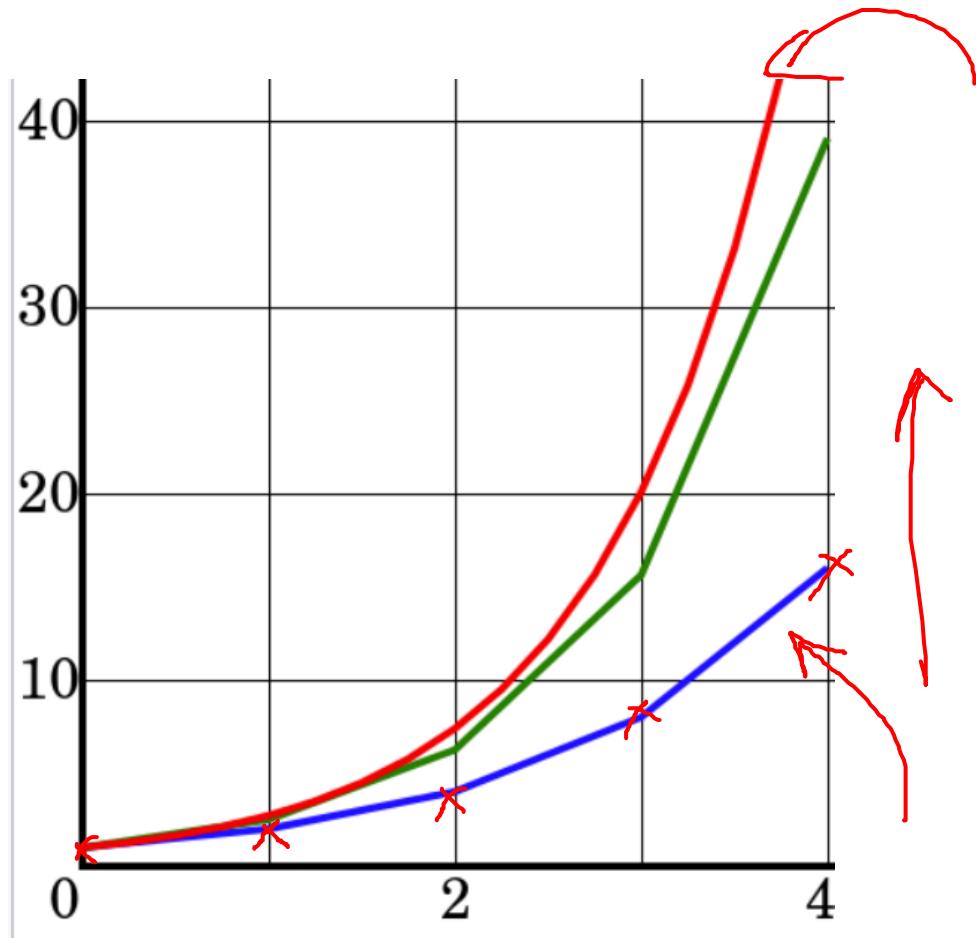
- Local error is $O(h^2)$

$O(h^2)$



This is called *first order accurate* $O(h^2)$
pth order accurate when $\ell_k = \mathcal{O}(h_k^{p+1})$

1D Example: Euler's Method



$$y_{n+1} = y_n + h f(t_n, y_n)$$

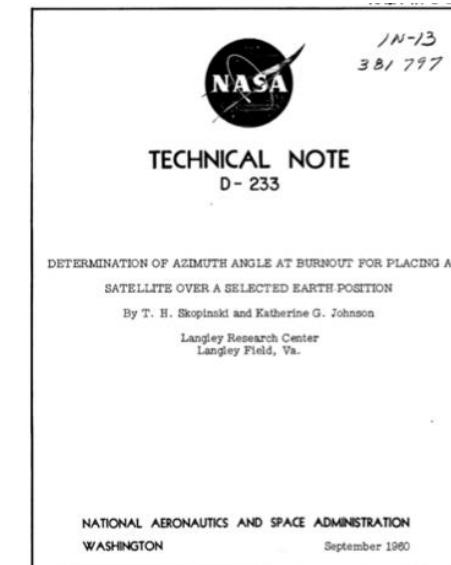
n	y_n	t_n	$f(t_n, y_n)$	h	Δy	y_{n+1}
0	1	0	1	1	1	2
1	2	1	2	1	2	4
2	4	2	4	1	4	8
3	8	3	8	1	8	16

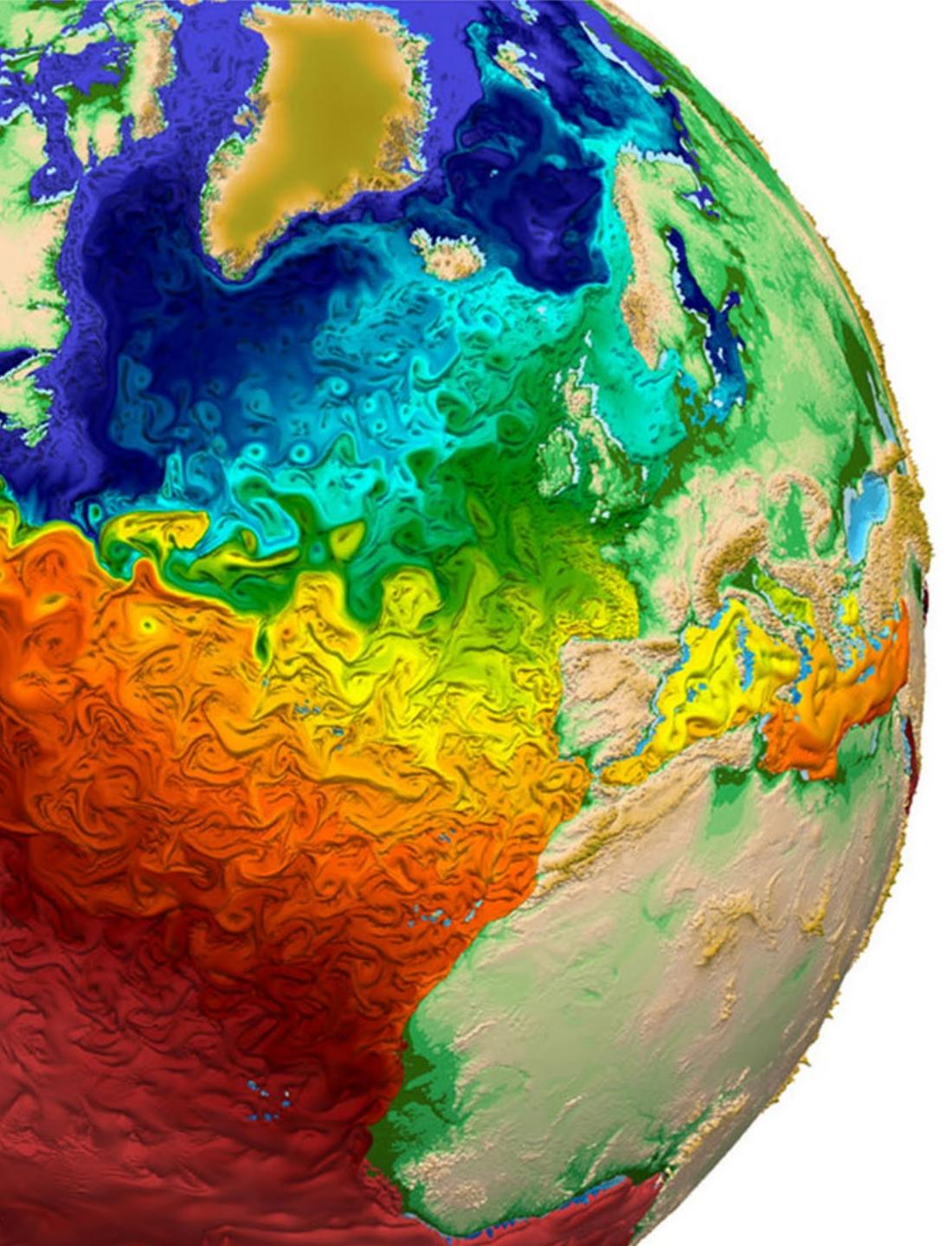
Euler's Method in History



Euler's Method scene in Hidden Figures (2016)
https://youtu.be/v-pbGAts_Fg

Used to calculate trajectories by NASA for Project Mercury...apparently





Numerical Differentiation

Finite Difference Methods

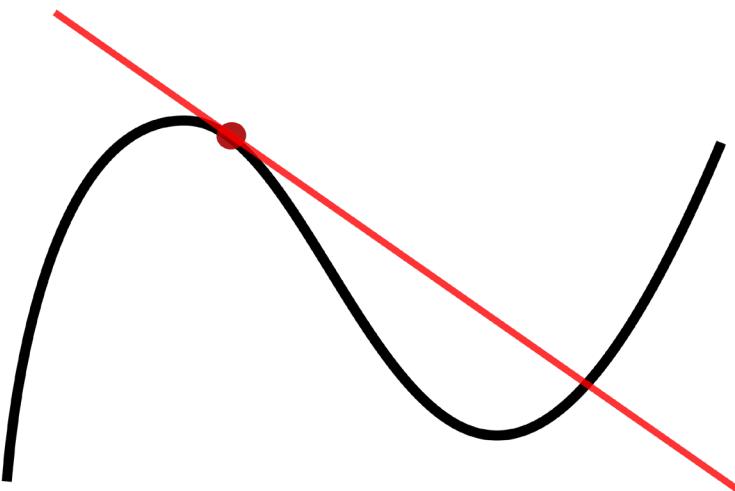
Scientific Visualization
Professor Eric Shaffer

Differentiation

Differentiation is the process of finding the derivative of a function

The derivative expresses the rate of change of an output with respect to a change in the input

“the ratio of the instantaneous change in the dependent variable to that of the independent variable”



Really it's a fundamental element
of data science...helps analyze change...

Numerical Differentiation

Analytical differentiation finds a derivative through symbolic manipulation

This is often ideal...no error and sometimes fast evaluation of the derivative

This is not always possible

- We may only have sampled data
- Derivative can be difficult or impossible to compute analytically
- Derivative may be prohibitively expensive to evaluate

"Evaluate" means to compute the value of a function (or derivative...which is also a function) for a specific input.

Numerical methods are an alternative...approximate the analytical solution

Centered Difference Formula

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

h is called the step-size

- smaller step-size will result in less error
- error is on the order of $O(h^2)$

x is the location in the domain at which we evaluate the derivative

this is a *finite difference method*

Understanding Error



- Differentiation is an inherently sensitive operation
 - Derivative of some functions changes rapidly..small input error can have great impact
- The centered difference formula can be derived from Taylor Series

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x-h)}{2h} - \frac{f'''(x)}{6} h^2 + \dots \\ &\approx \frac{f(x+h) - f(x-h)}{2h}, \end{aligned}$$

Reducing step-size by $\frac{1}{2}$ reduces errors by $\frac{1}{4}$

- Can see that truncation error is $O(h^2)$

Derivatives of Sampled Data

People frequently use the centered difference formula...it's fast and easy

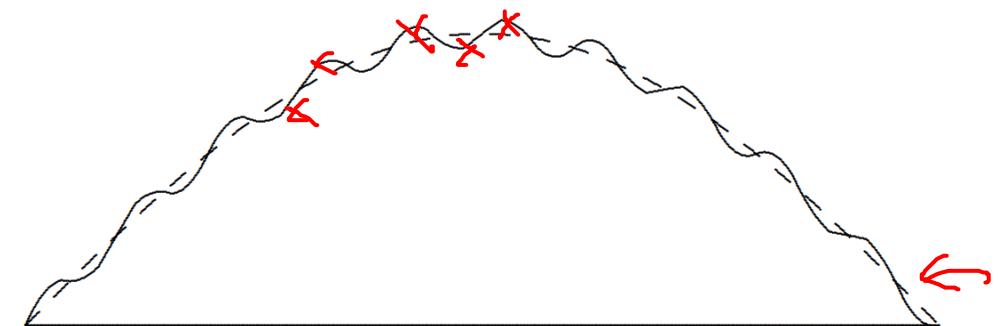
- Appropriate when function is known and we want fast approximation
- Not really appropriate for sampled or noisy data

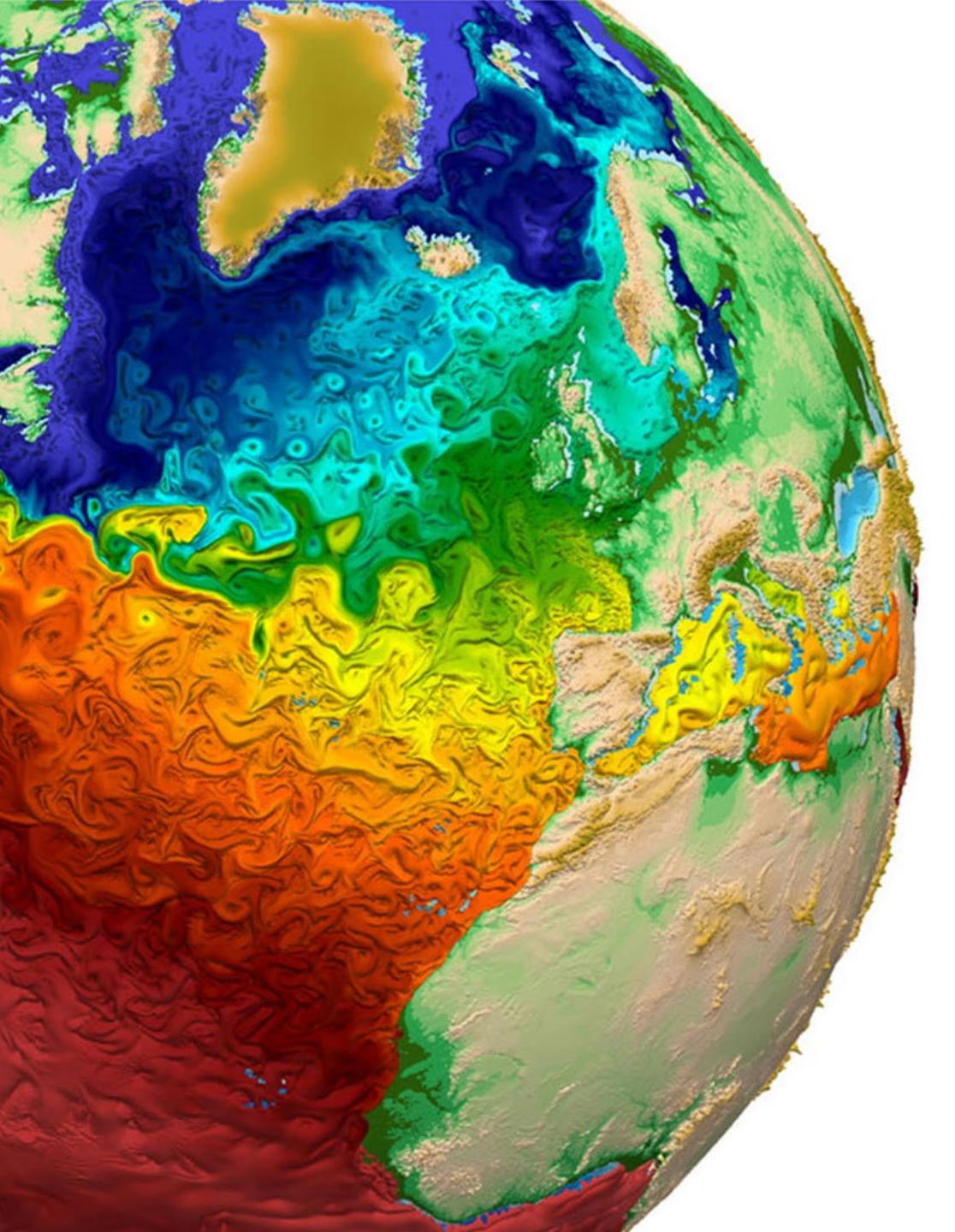
Better approach for sampled data

- Fit an approximate function to data
- Take the derivative of the approximate function

Example:

- Least-squares best fit of quadratic function
- Then differentiate quadratic





Numerical Methods

Runge-Kutta Methods

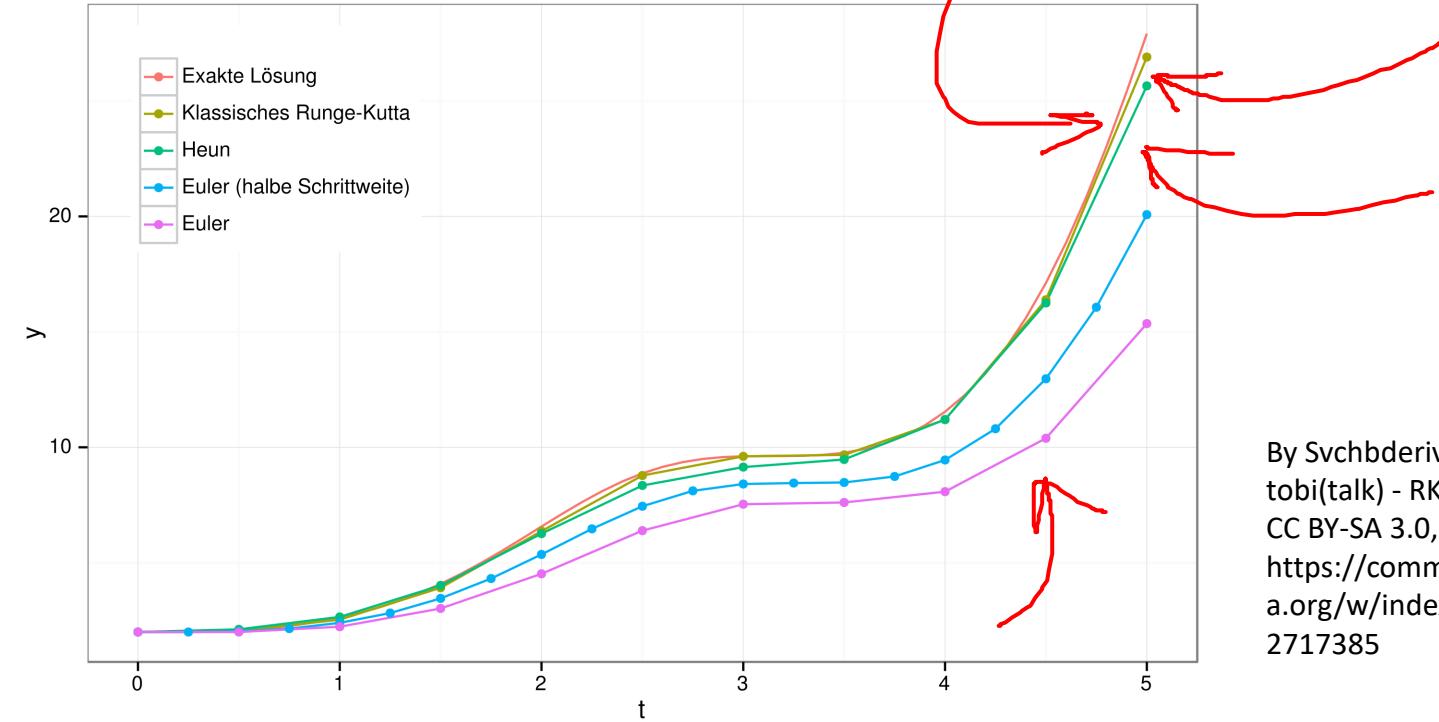
Scientific Visualization
Professor Eric Shaffer

Runge-Kutta Methods

RK methods numerically solve ODEs

Developed around 1900 by Carl Runge and Wilhelm Kutta

Similar to Euler's Method
...offer higher accuracy



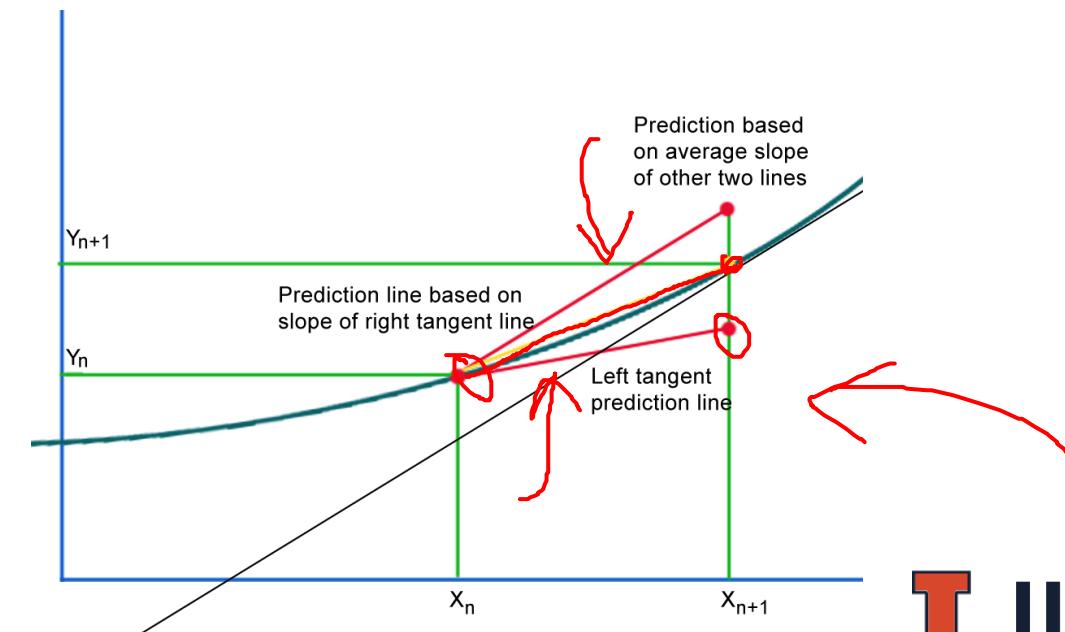
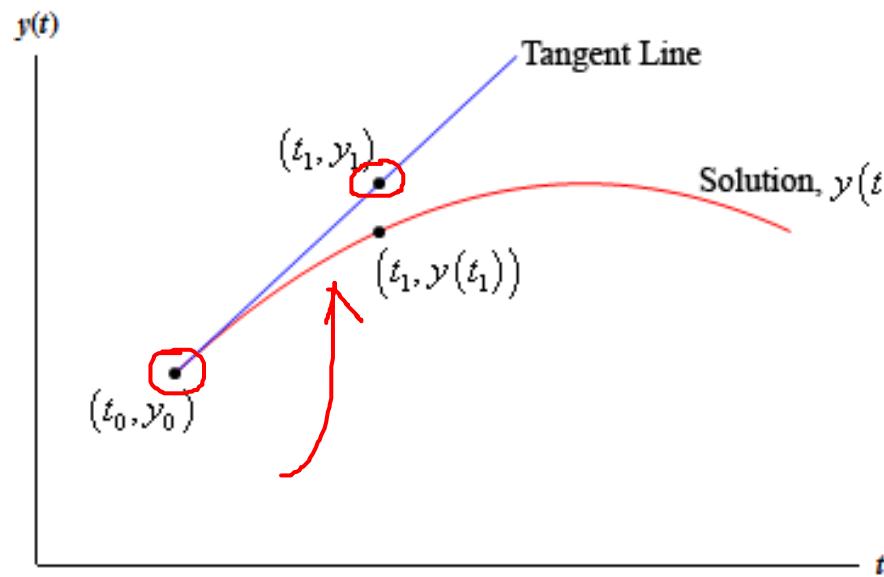
By Svchbderivative work:
tobi(talk) - RK Verfahren,
CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=32717385>

...Euler's Method can be thought of as the first order RK method

Euler's Method

Samples the rate of change at a single point each iteration

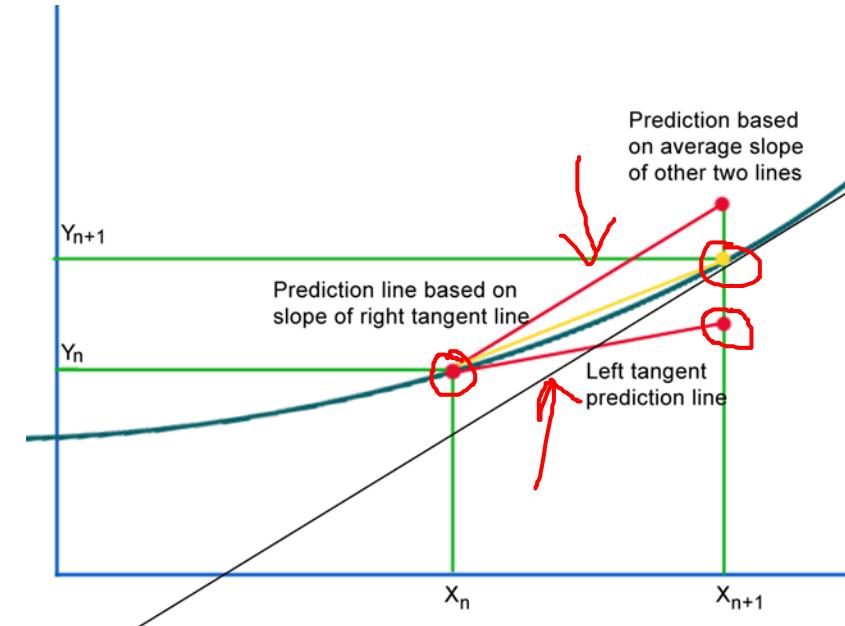
Intuitively...if we sampled more points maybe we could do better



Heun's 2nd Order Method

Second order RK method

1. $\vec{k}_1 = \vec{v}(x_n, t_n)$
2. $\vec{k}_2 = \vec{v}(x_n + h_n k_1, t_n + h_n)$
3. $x_{n+1} = x_n + \frac{h_n}{2} (\vec{k}_1 + \vec{k}_2)$



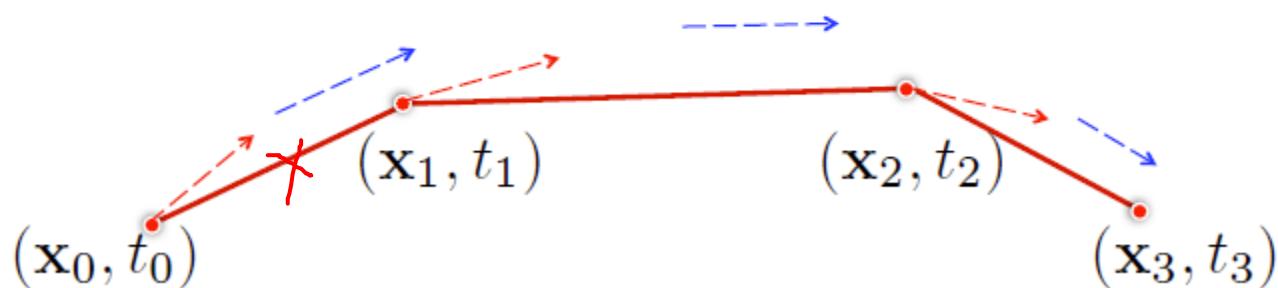
Local truncation error is $O(h^3)$

RK2 Variation....

Sometimes you will see

$$\vec{k}_1 = h\vec{v}(\mathbf{x}_n, t_n)$$
$$\vec{k}_2 = h\vec{v}\left(\mathbf{x}_n + \frac{1}{2}\vec{k}_1, t_n + \frac{1}{2}h\right)$$

$$\boxed{\mathbf{x}_{n+1} = \mathbf{x}_n + \vec{k}_2 + O(h^3)}$$



RK4

- Fourth order Runge-Kutta

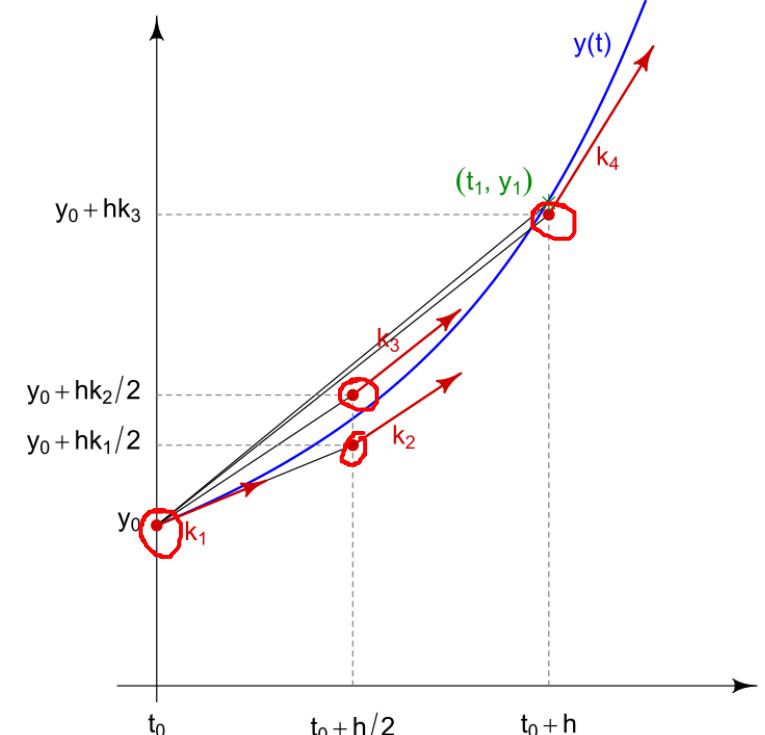
$$\vec{k}_1 = h \vec{v}(\mathbf{x}_n, t_n)$$

$$\vec{k}_2 = h \vec{v}\left(\mathbf{x}_n + \frac{1}{2} \vec{k}_1, t_n + \frac{1}{2} h\right)$$

$$\vec{k}_3 = h \vec{v}\left(\mathbf{x}_n + \frac{1}{2} \vec{k}_2, t_n + \frac{1}{2} h\right)$$

$$\vec{k}_4 = h \vec{v}(\mathbf{x}_n + \vec{k}_3, t_n + h)$$

$$\boxed{\vec{x}_{n+1} = \vec{x}_n + \frac{1}{6} \vec{k}_1 + \frac{1}{3} \vec{k}_2 + \frac{1}{3} \vec{k}_3 + \frac{1}{6} \vec{k}_4 + O(h^5)}$$



Accuracy

- RK4 requires 4 evaluations of the derivative (velocity) per step
- RK2 requires 2 evaluations per step
- Euler requires 1 evaluation per step

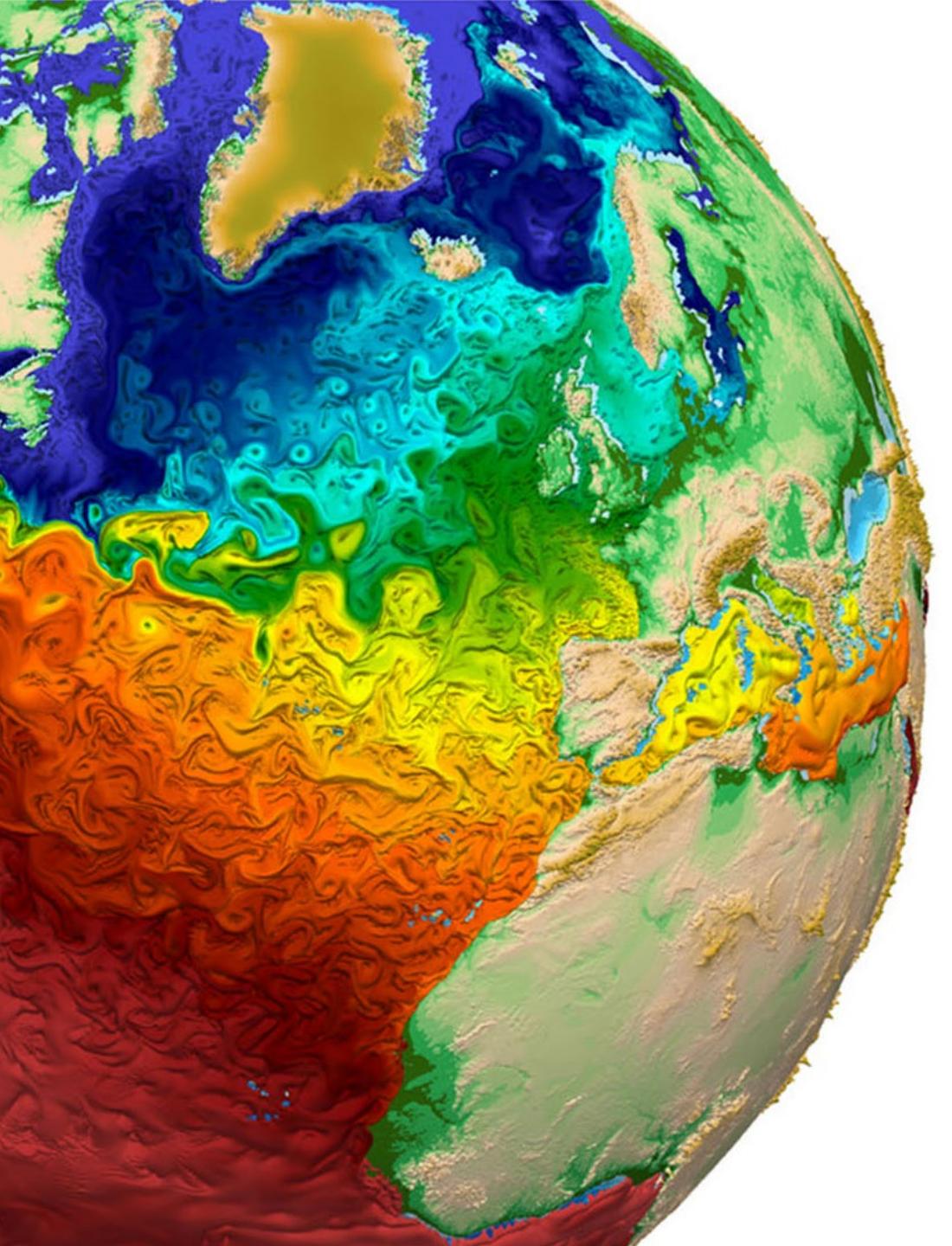
For RK4 to be superior it would need to be more accurate than

- RK2 using $\frac{1}{2}$ the stepsize
- Euler using $\frac{1}{4}$ the stepsize

This is usually the case...but can vary by problem

Properties of Runge-Kutta Methods

- Easy to implement
- Each step requires only one previous step
- Can adjust step-size at each step...make method adaptive
 - Unfortunately no error estimate available to control step-size
 - Embedded RK methods use multiple RK methods to remedy this

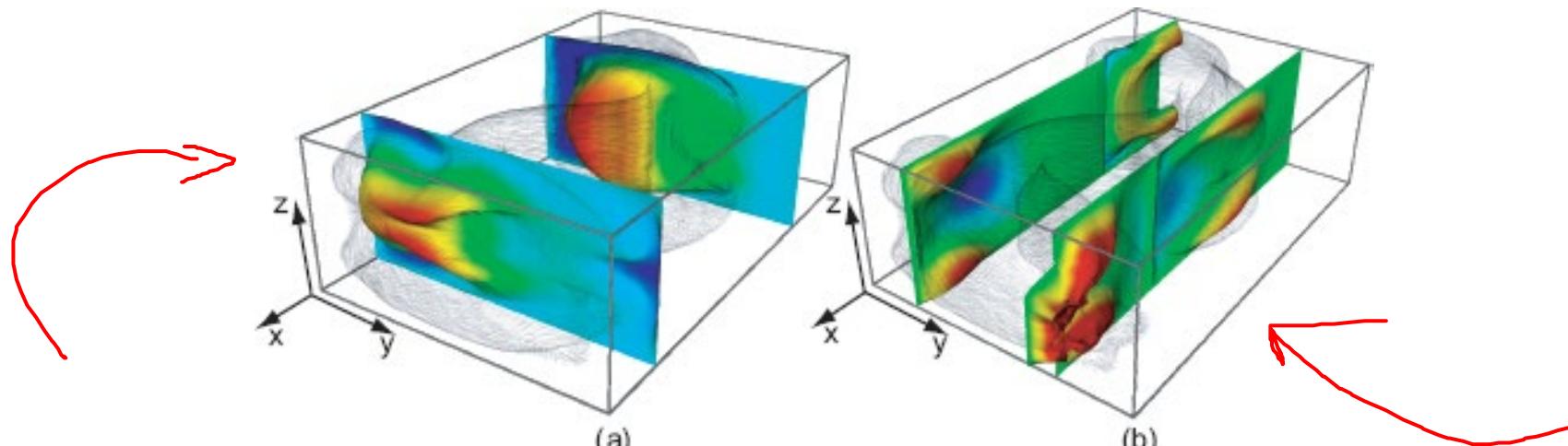


Vector Visualization

Displacement Plots

Scientific Visualization
Professor Eric Shaffer

Displacement Plots



We create a surface $S \in D$ inside the domain D of a vector field

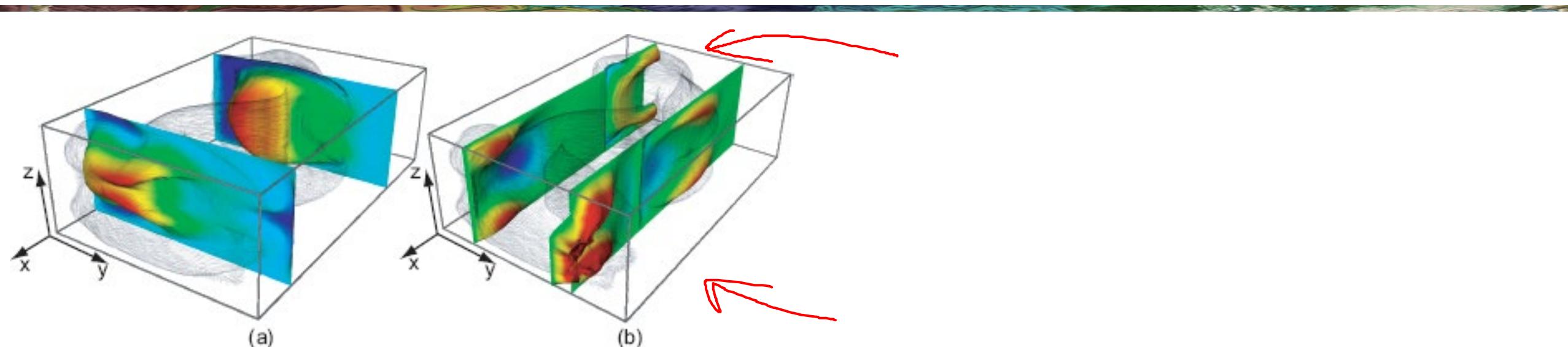
S is discretized as a set of sample points p_i

A displacement plot of S is a new surface S'' given by the set of sample points

$$p'_i = p_i + k\mathbf{v}'(p_i)$$

- k controls the scale of the displacement
- \mathbf{v}' is a vector field that controls the displacement of the surface S
- in the simplest case we can use $\mathbf{v}' = \mathbf{v}$

Displacement Plots



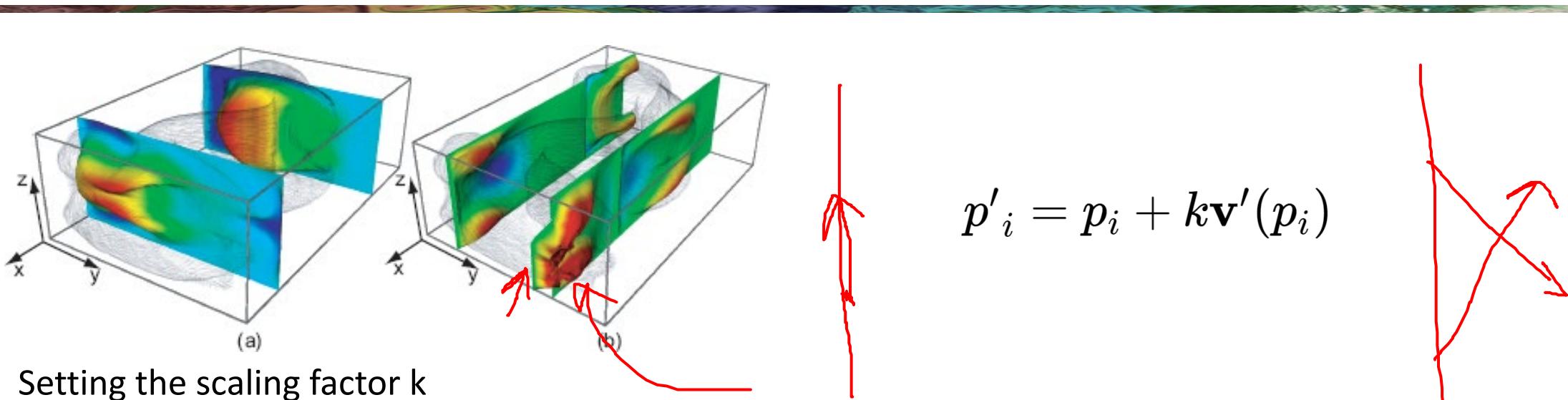
Visualizing flows using two planar surfaces in each visualization

- (a) orthogonal to the x-axis
- (b) orthogonal to the y-axis

Colored by the vector field component on which the input surface is perpendicular

- (a) $|\mathbf{v}_x|$
- (b) $|\mathbf{v}_y|$

Parameter Settings



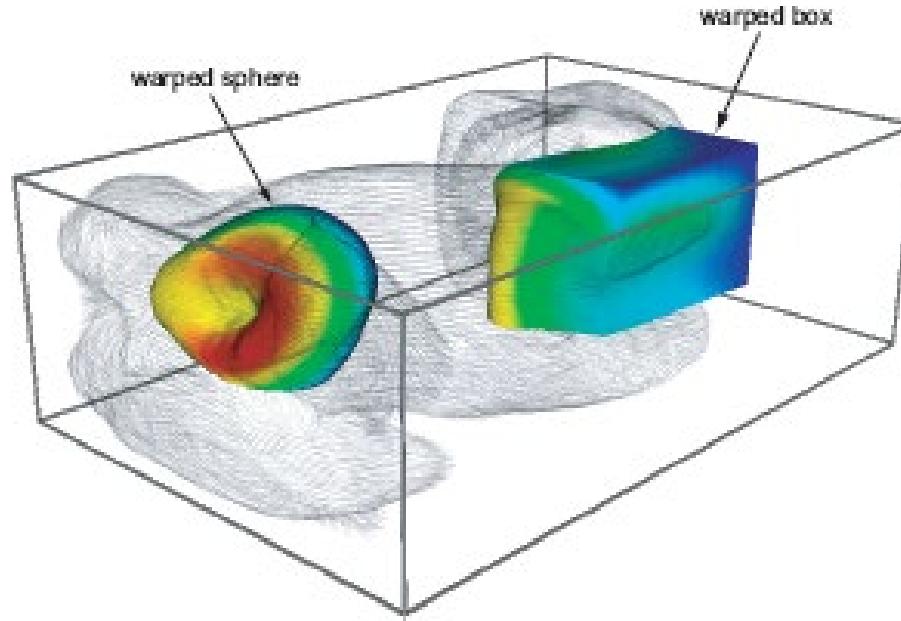
Setting the scaling factor k

- too large can easily lead to self-intersecting surfaces.
- too large warp factors shift the displaced surface far away from its actual location
- too small a factor will not show the warping effect enough so that it is recognizable

Setting the shape and position of the surface to be warped

- planar surfaces are often a popular choice for displacement plots
- since they are flat, displacement values are easy to distinguish on them
- problematic when surface is (almost) tangent to the vector field to be visualized
- warped surface stays in the same plane as the original surface

Alternatives

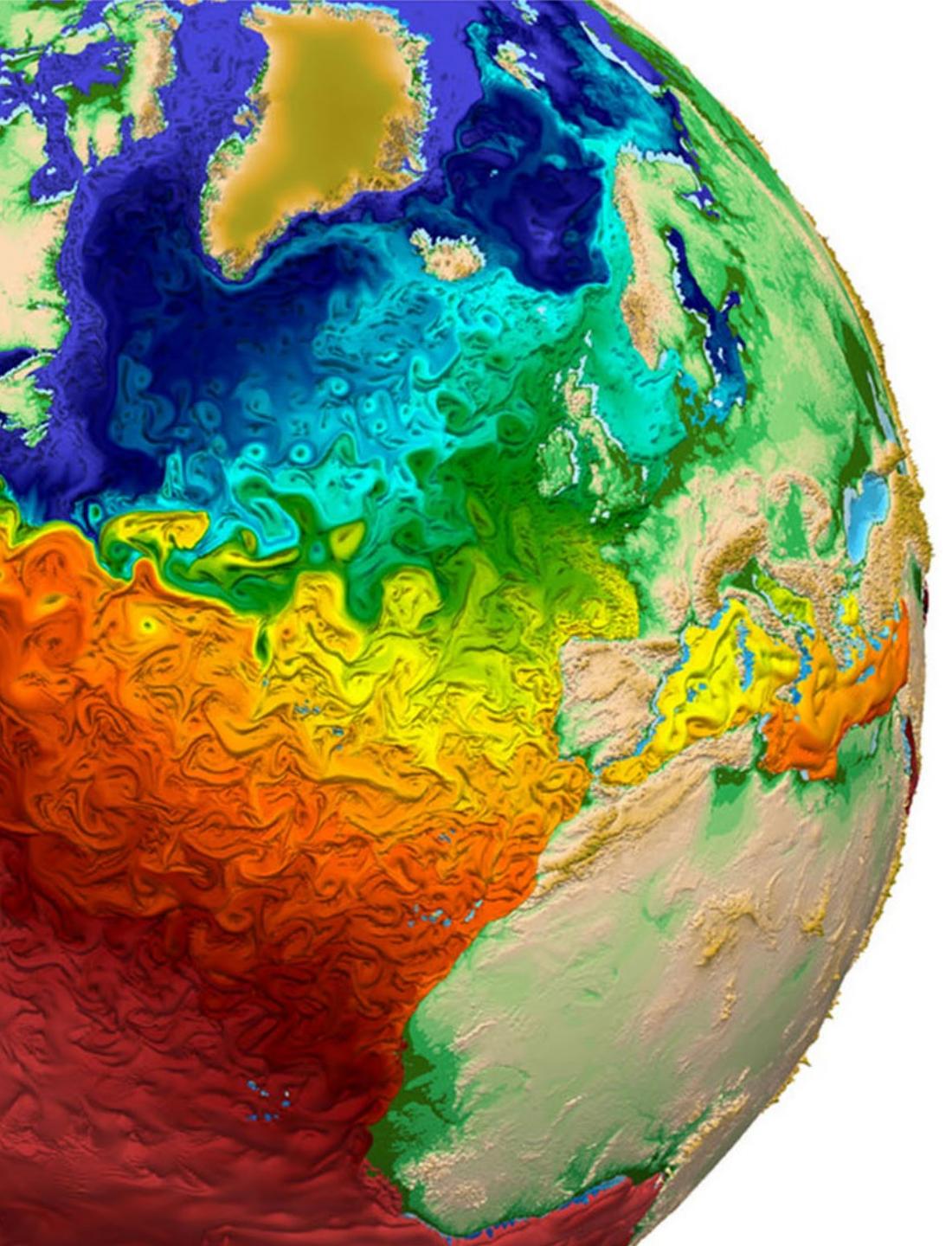


Non-planar geometric objects can be used to create displacement plots

In addition to surfaces, curves can be used too

Choice of the object to deform should be correlated with the vector field behavior and meaning

- example: material deformation a mechanical assembly can be shown using the assembly's own surface.



Vector Visualization

Line Integral Convolution

Scientific Visualization
Professor Eric Shaffer

Line Integral Convolution (LIC) Principle

So Far

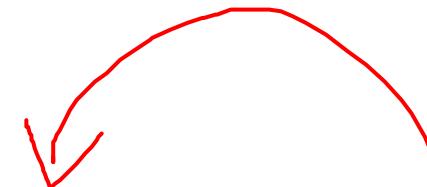
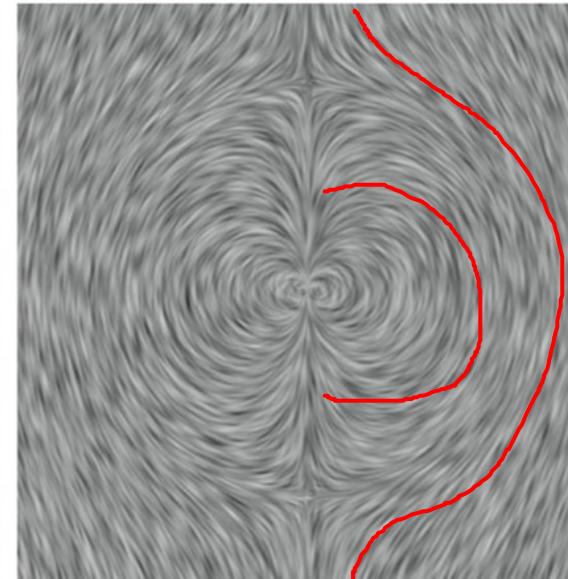
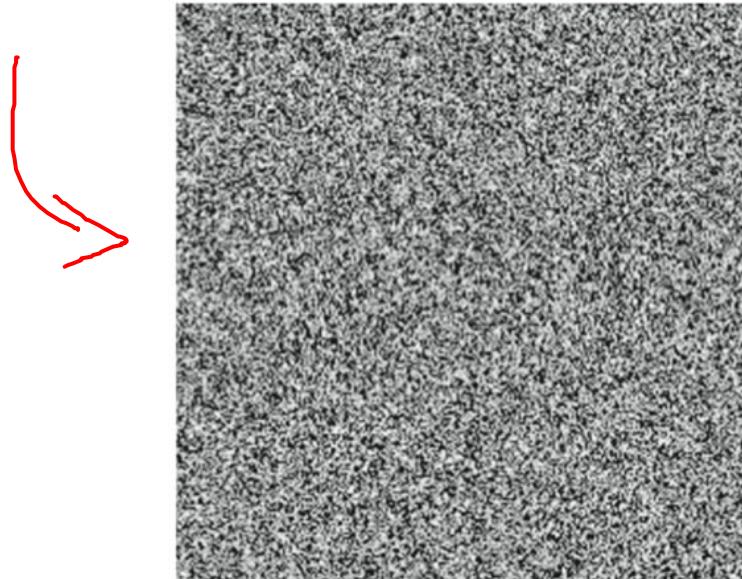
- mostly discrete visualizations (glyphs, streamlines, stream ribbons)

Goal

- a dense, pixel-filling, continuous, vector field visualization

Line integral convolution

- highly coherent images *along* streamlines
- highly contrasting images *across* streamlines
by blurring noise along the streamlines of v



Line Integral Convolution (LIC) Principle

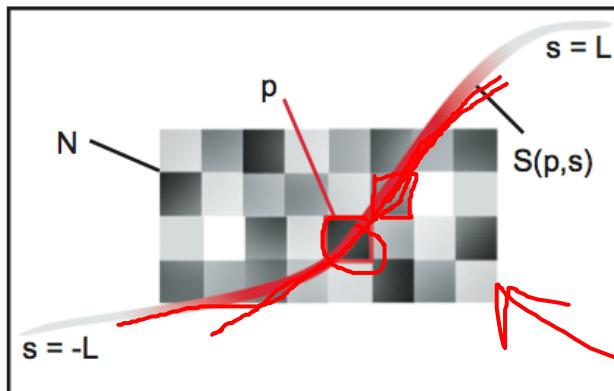
So Far

- mostly discrete visualizations (glyphs, streamlines, stream ribbons)

Goal

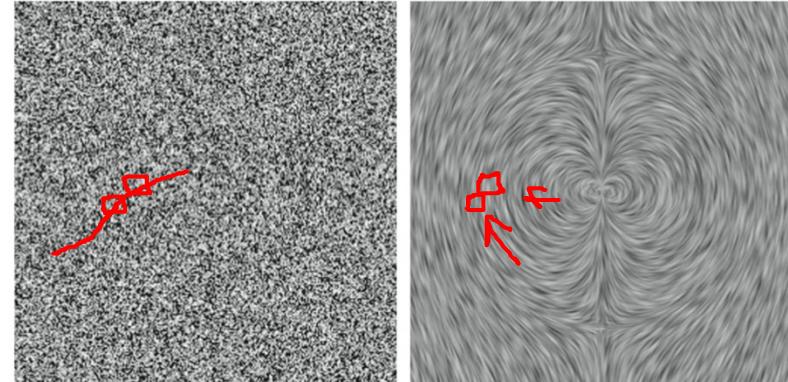
- a dense, pixel-filling, continuous, vector field visualization

Principle



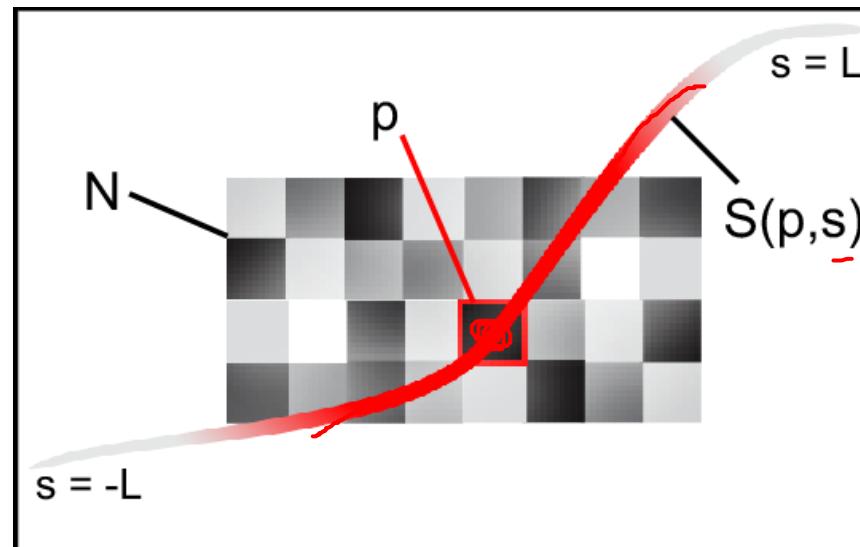
$$T(p) = \frac{\int_{-L}^L N(S(p, s))k(s)ds}{\int_{-L}^L k(s)ds}$$

gray value at pixel p
 N = noise texture



- take each pixel p of the screen image
- trace a streamline from p upstream and downstream (as usual)
- blend all streamlines, pixel-wise
 - multiplied by a random-grayscale value at p
 - with opacity decreasing (exponentially) on distance-along-streamline from p
- identical to *blurring* noise along the streamlines of v

Understanding Line Integral Convolution (LIC)



LIC: Line Integral Convolution

$$T(p) = \frac{\int_{-L}^L N(S(p,s))k(s)ds}{\int_{-L}^L k(s)ds}$$

$k(s) = e^{-s^2}$

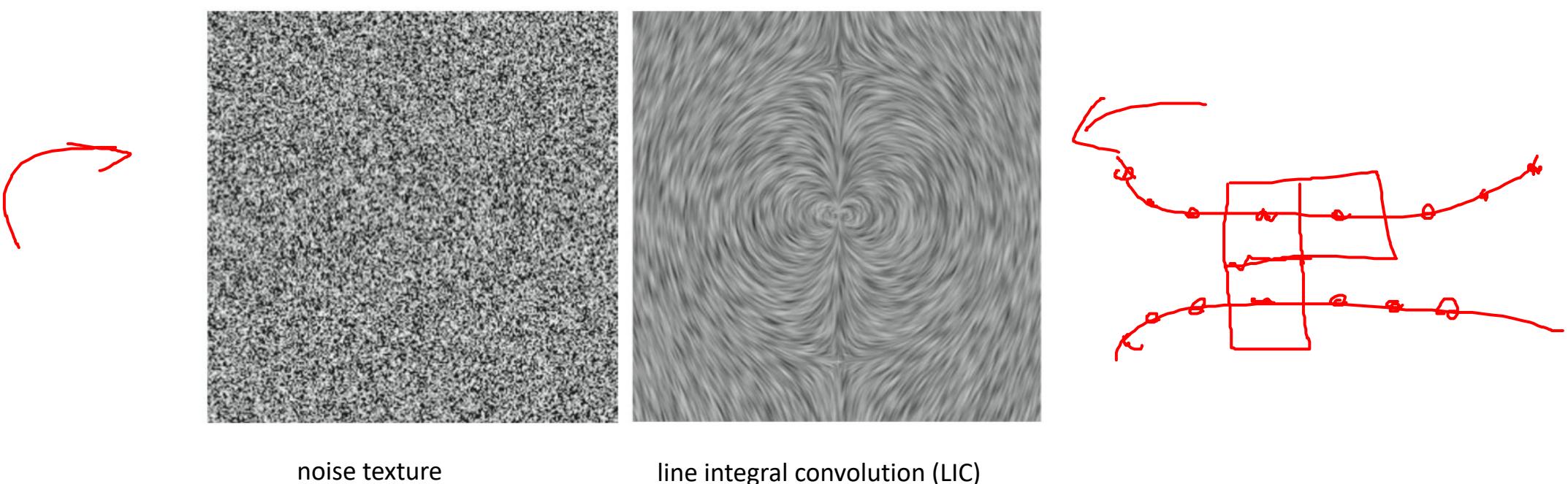
N : noise texture

$S(p,s)$: streamline of seed point P

$k(s)$: weighting or blurring function

L : width of blurring function

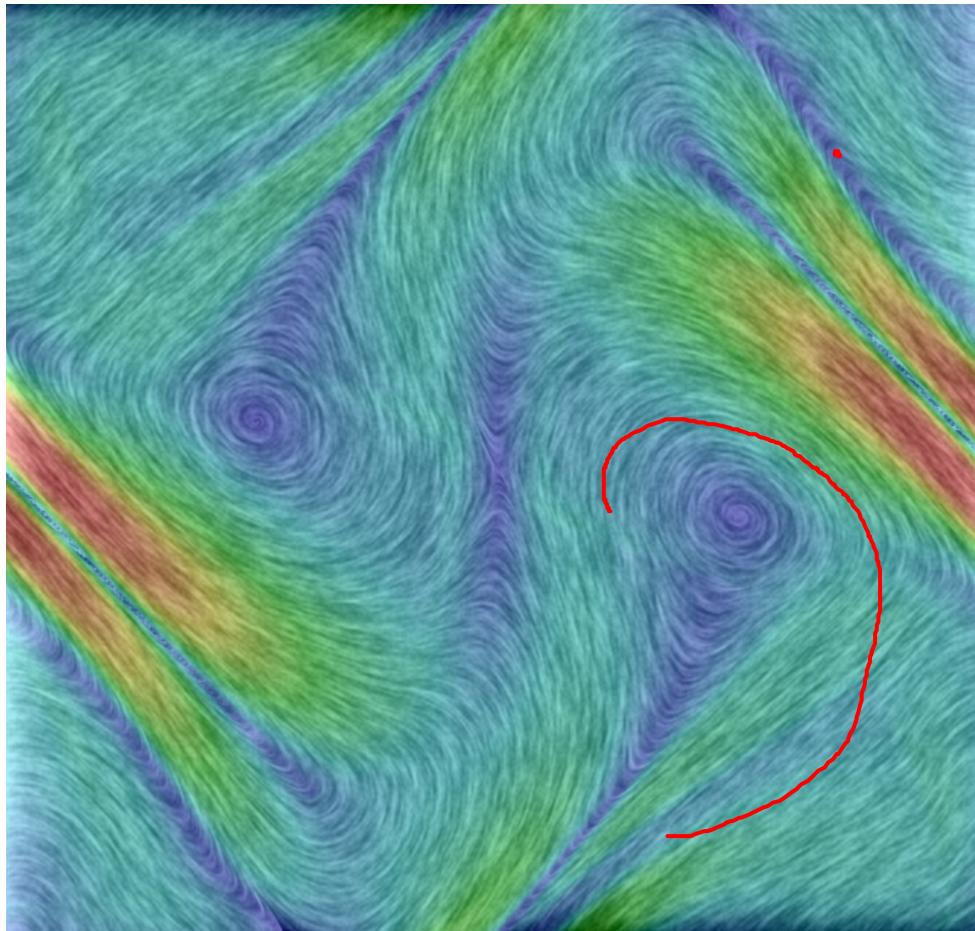
Texture-based Methods



Line integral convolution

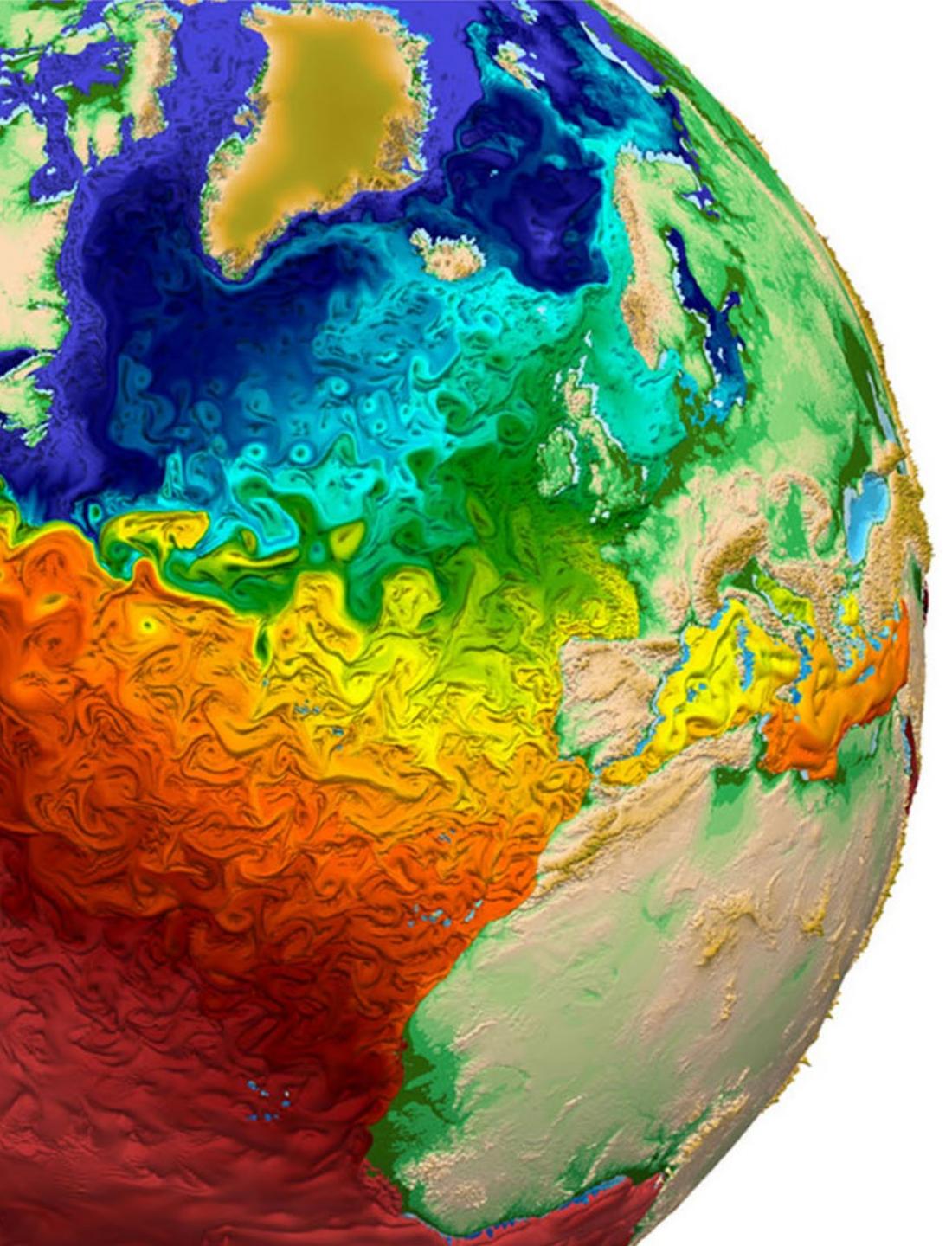
- highly coherent images *along* streamlines
- highly contrasting images *across* streamlines
- Fast implementation possible using texture-mapping capabilities of modern graphics processing units (GPUs)

Example: LIC with a Colormap



Vector magnitude:
Color

Vector direction:
Graininess



Vector Visualization

Stream Objects

Scientific Visualization
Professor Eric Shaffer

Stream Objects

Main Idea

- think of the vector field $\mathbf{v} : D$ as a flow field
- choose some ‘seed’ points $s \in D$
- move the seed points s in \mathbf{v}
- show the trajectories

Streamlines

- assume that \mathbf{v} is not changing in time (steady-states)
- for each seed $p_0 \in D$
 - the streamline S seeded at p_0 is given by

$$S = \{p(\tau), \tau \in [0, T]\}, p(\tau) = \int_{t=0}^{\tau} \mathbf{v}(p) dt, \quad \text{where } p(0) = p_0$$



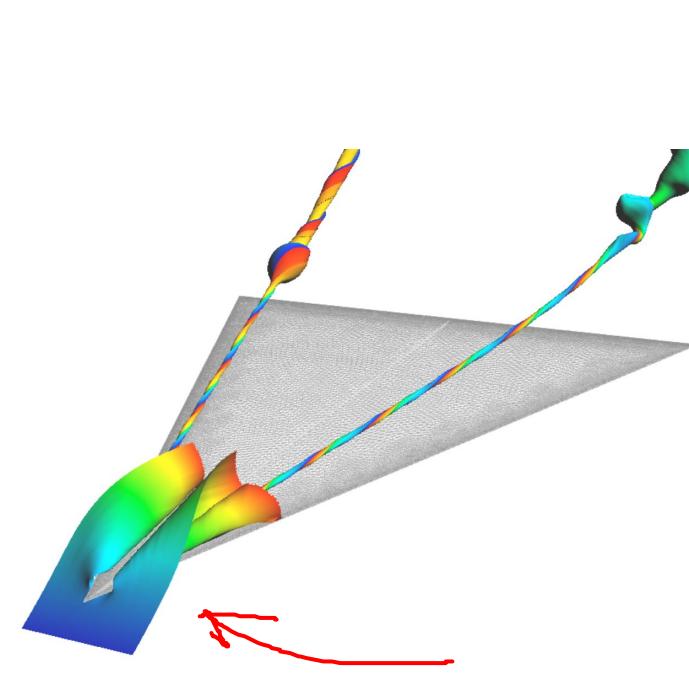
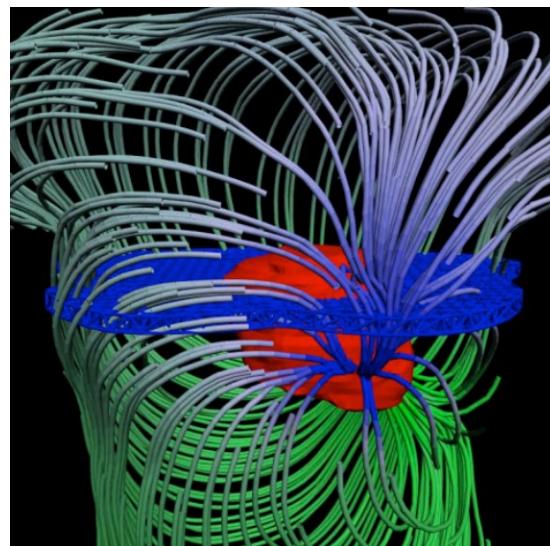
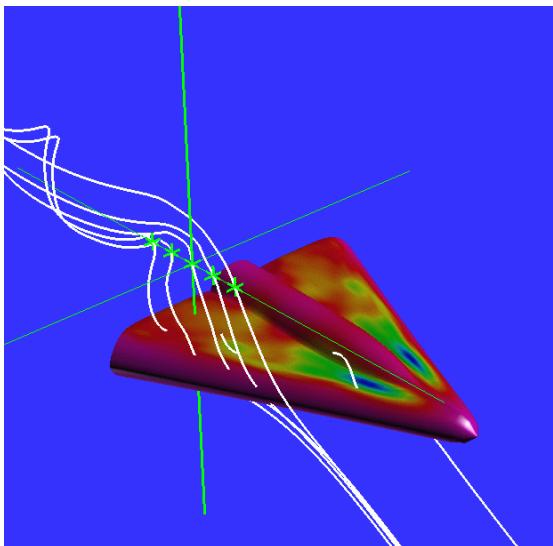
integrate p_0 in vector field \mathbf{v} for time T

- if \mathbf{v} is time dependent $\mathbf{v}=\mathbf{v}(t)$, streamlines are called **particle traces**

Stream Objects

Stream-{lines | tubes | ribbons | polygons | ...}

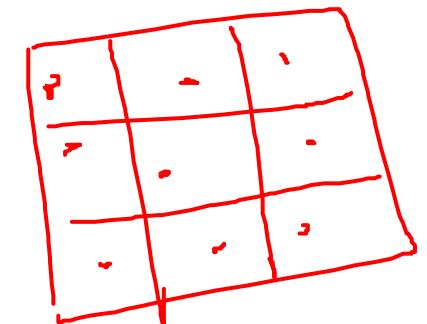
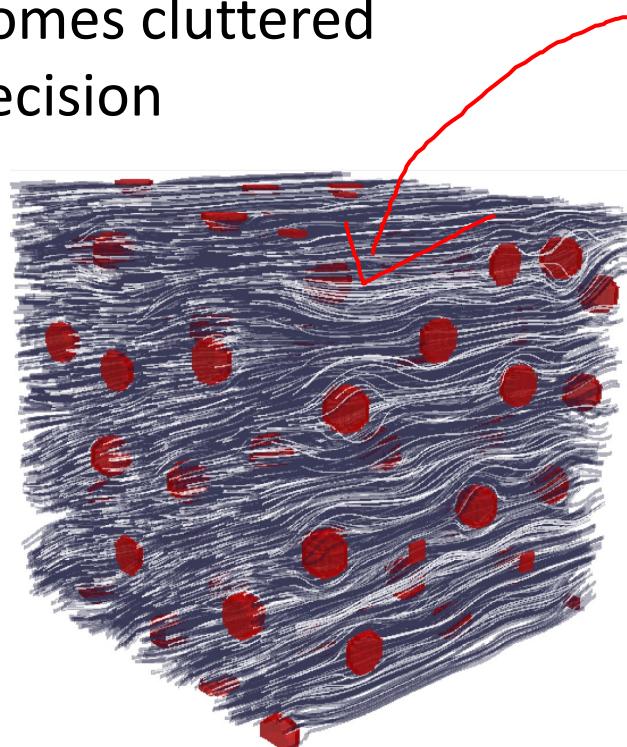
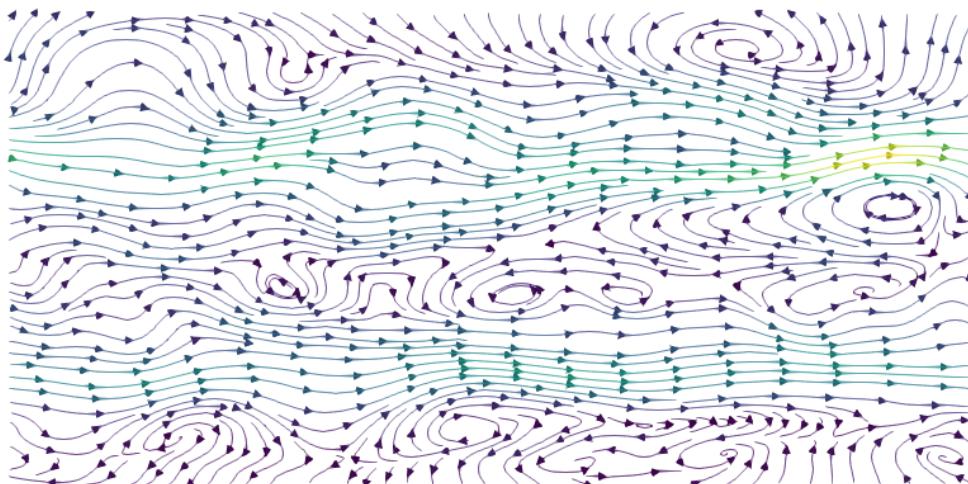
- Vector glyph plots show the **trajectories over a short time** of trace particles released in the vector fields
- Stream objects show the **trajectories for longer time intervals** for a given vector field



Stream Objects

Stream-{lines | tubes | ribbons | polygons | ...}

- Displaying streamlines is a local technique
- You can only visualize the flow directions initiated from a few particles
- Too many streamlines and the scene becomes cluttered
- Location of the seed particles is crucial decision



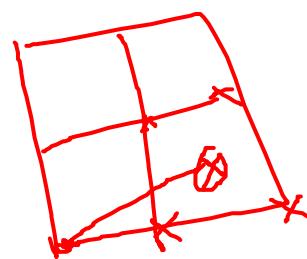
Stream Objects

Construction

- numerically integrate

$$S = \{p(\tau), \tau \in [0, T]\}, p(\tau) = \int_{t=0}^{\tau} \mathbf{v}(p) dt, \quad \text{where } p(0) = p_0$$

- discretizing time yields

$$\int_{t=0}^{\tau} \mathbf{v}(p) dt = \sum_{i=0}^{\tau/\Delta t} \mathbf{v}(p_i) \Delta t \quad \text{where } p_i = p_{i-1} + \mathbf{v}_{i-1} \Delta t \quad (\text{simple Euler integration})$$


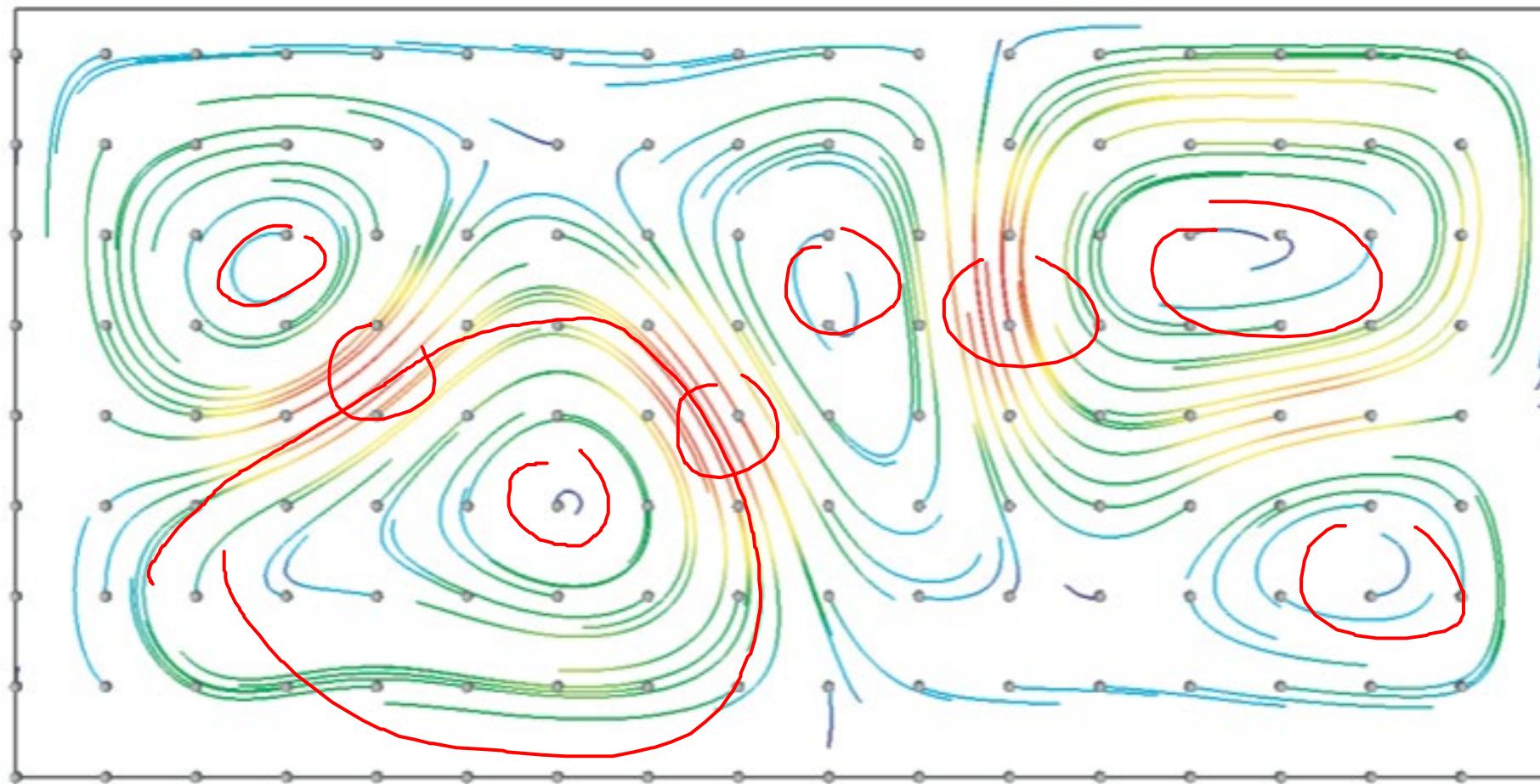
- Euler integration

- we consider \mathbf{v} constant between two sample points p_i and p_{i+1}
 - we compute $\mathbf{v}(p)$ by linear interpolation within the cell containing p
 - variant: use $\mathbf{v}(p)/\|\mathbf{v}(p)\|$ instead of $\mathbf{v}(p)$ in integral
 - S will be a polyline, $S = \{p_i\}$
- stop when $\tau=T$ or $\mathbf{v}(p)=0$ or $p \notin D$
 - what does $\tau=T$ mean when we use $\mathbf{v}(p)/\|\mathbf{v}(p)\|$?

RK4 would be better



Example: Streamlines

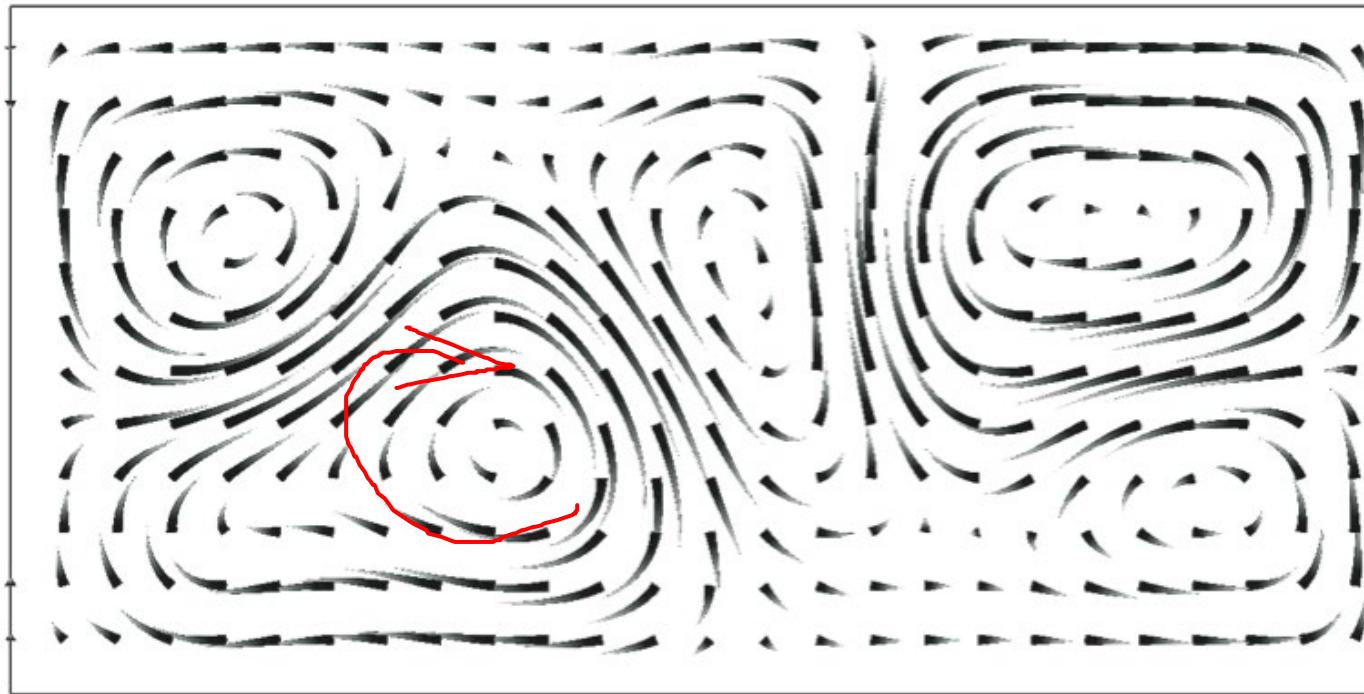


streamlines: seeds from regular grid; use un-normalized \mathbf{v} for integration; color by $\|\mathbf{v}\|$

Stream Tubes

Can modulate tube thickness by

- data (hyperstreamlines)
- integration time – we obtain nice tapered arrows

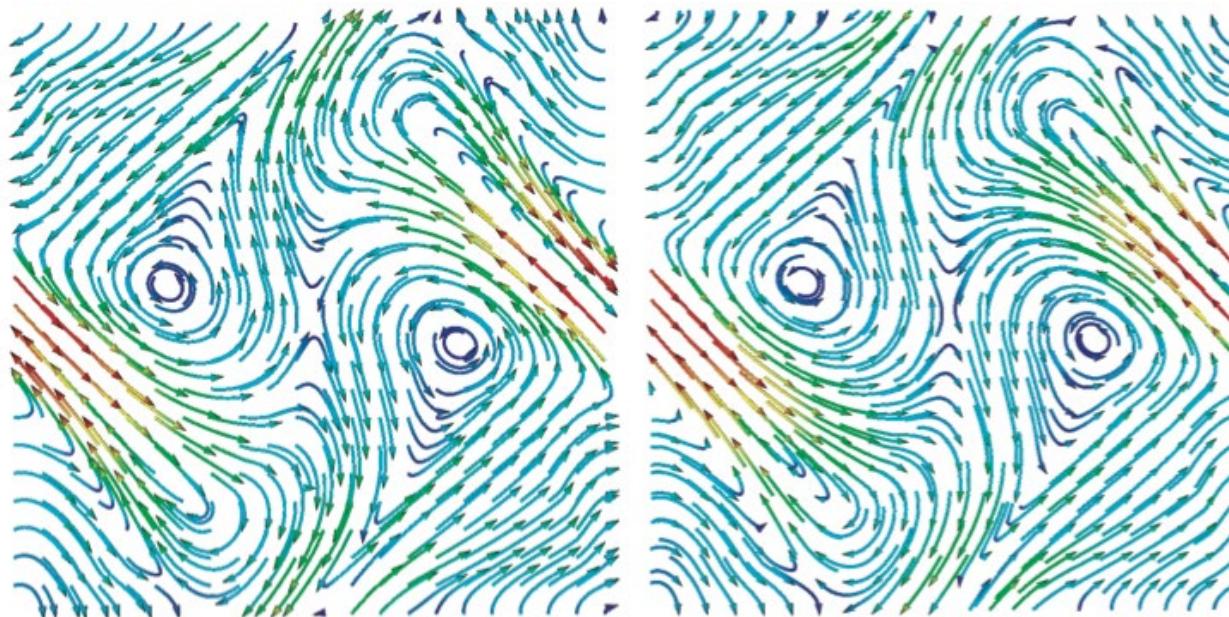


stream tubes – radius *and* opacity decrease with integration time

Stream Tubes

Like stream objects, but 3D

- compute 1D stream objects (e.g. streamlines)
- sweep (circular) cross-section along these
- visualize result with shading

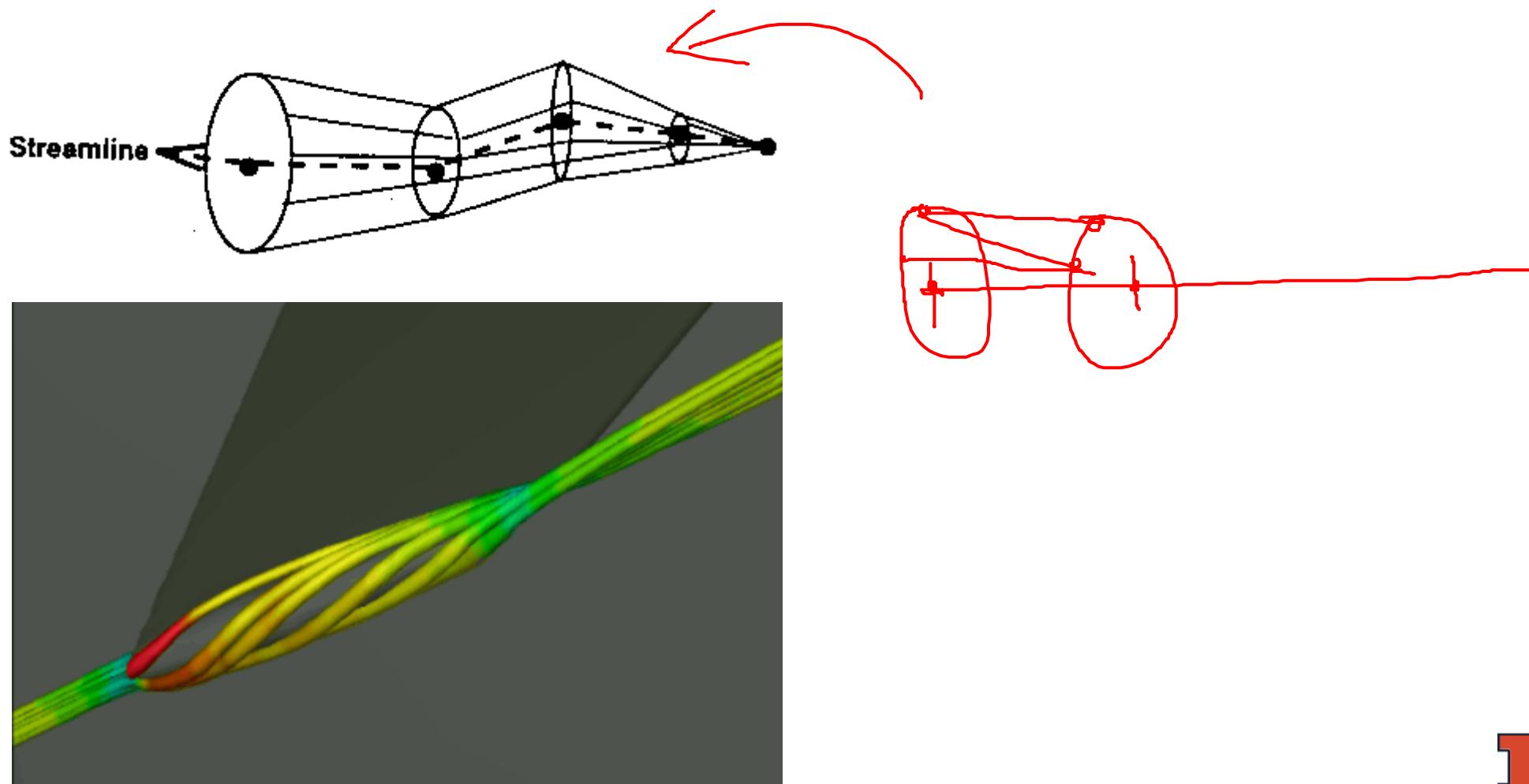


stream tubes, backward integration\

- in 2D they are a nicer option than hedgehog/glyph plots

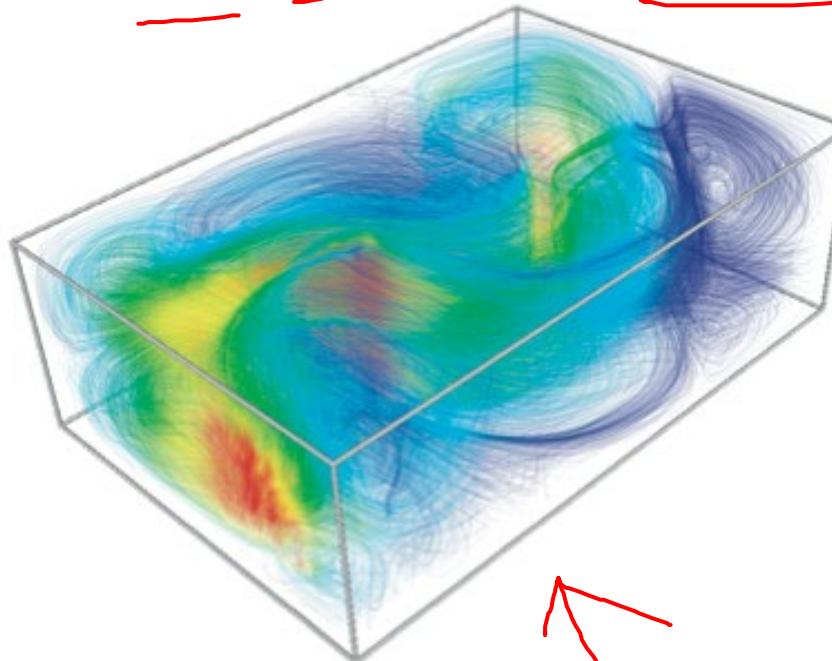
Stream Tube

Generate a stream-line and connect circular crossflow sections along the stream-line



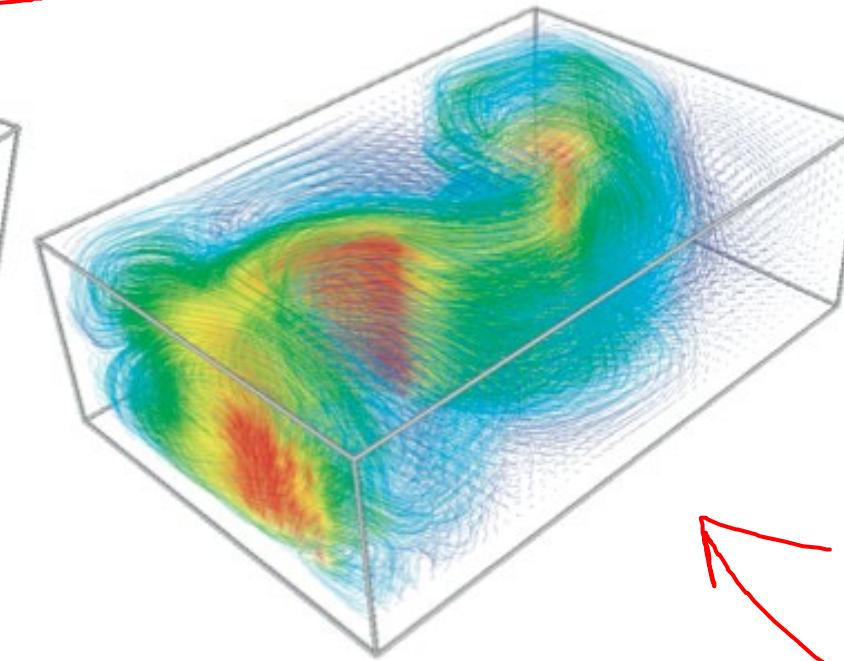
Streamlines in 3D

Can vary opacity, seeding density, integration time



undersampling, opacity

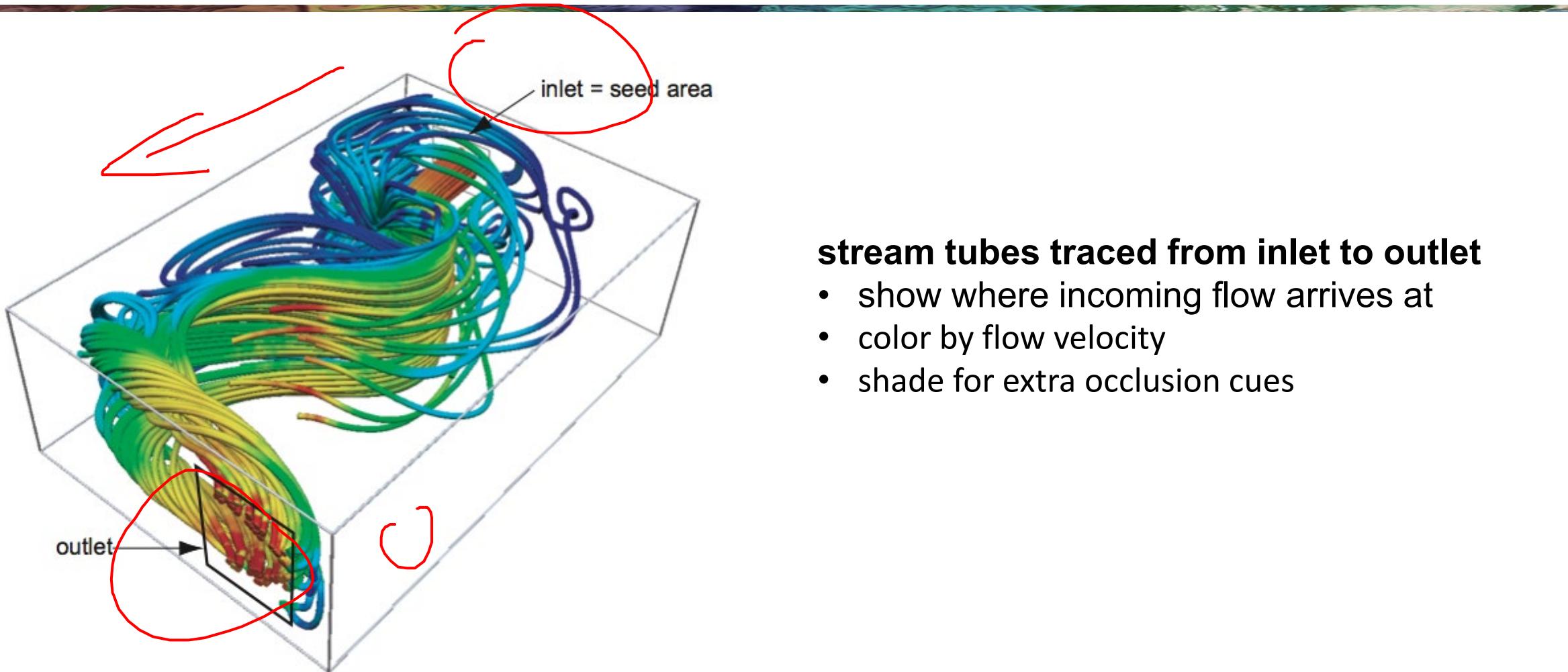
- less occlusion
- good coverage



undersampling, shorter time

- even less occlusion
- but less continuity

Stream Tubes in 3D



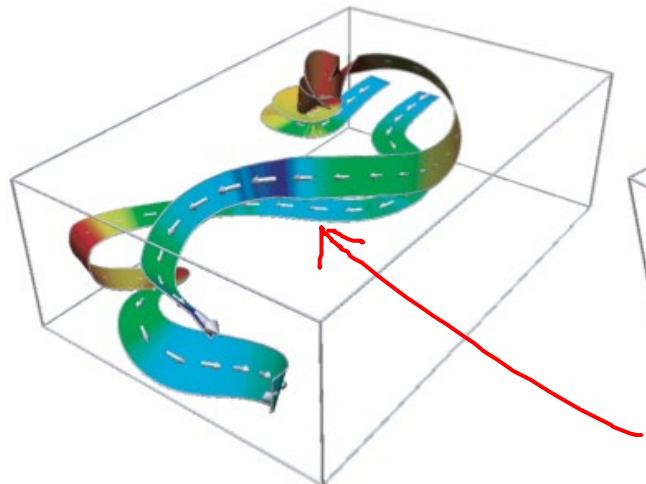
stream tubes traced from inlet to outlet

- show where incoming flow arrives at
- color by flow velocity
- shade for extra occlusion cues

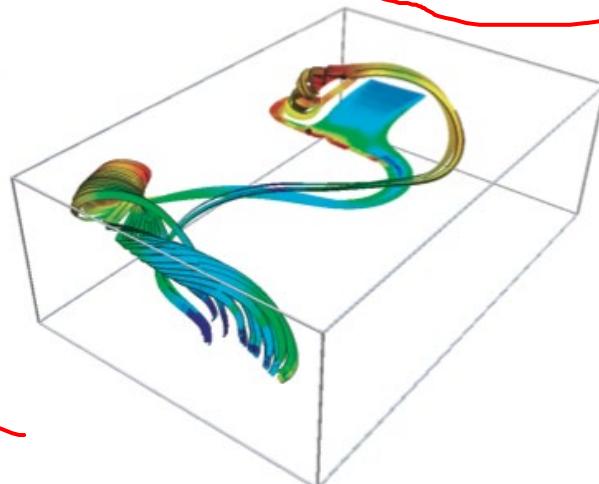
- even higher occlusion problem than for 3D streamlines
- must reduce number of seeds

Stream Ribbons

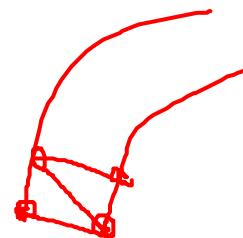
- Visualize how the vector field ‘twists’ around itself as it advances in space
- Visualizes the *helicity* of a vector field
$$h = \frac{1}{2} \mathbf{v} \cdot \text{rot } \mathbf{v}$$



stream ribbons: two thick ribbons



stream ribbons: 20 thin ribbons



Algorithm

- define pairs of close seeds (p_a, p_b)
- trace streamlines S_a, S_b from (p_a, p_b)
- construct strip surface connecting closest points on S_a, S_b

Stream Objects for Unsteady Flows

- Streamline
particle trajectory in steady (unchanging) vector field
- Pathline
trajectory of particle in an unsteady flow
- Timeline
connect particles released simultaneously at discrete time-steps
- Streakline
continuously inject particles at a point, connect consecutive particles

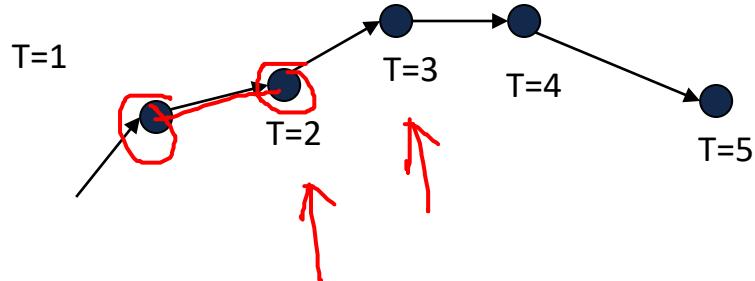
Pathlines

Extension of streamlines for time-varying data (unsteady flow)

Insert a particle into the flow

Connect positions over time

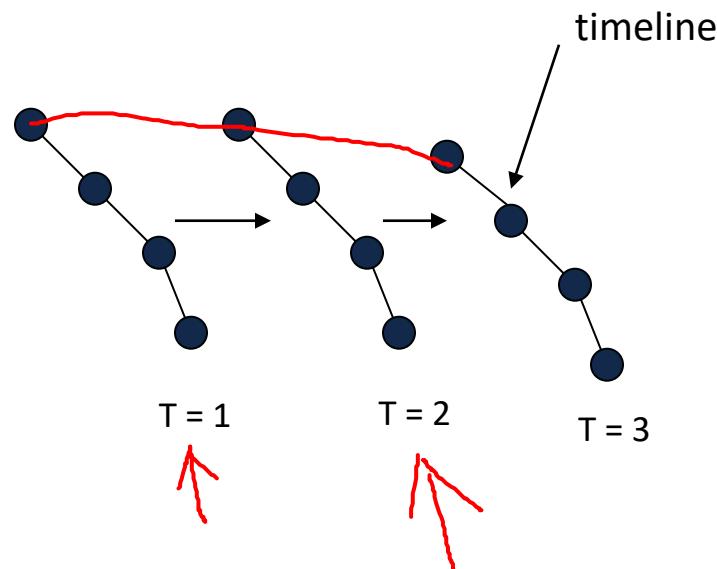
Difference from streamline is that the vector field is changing each time step



Timelines

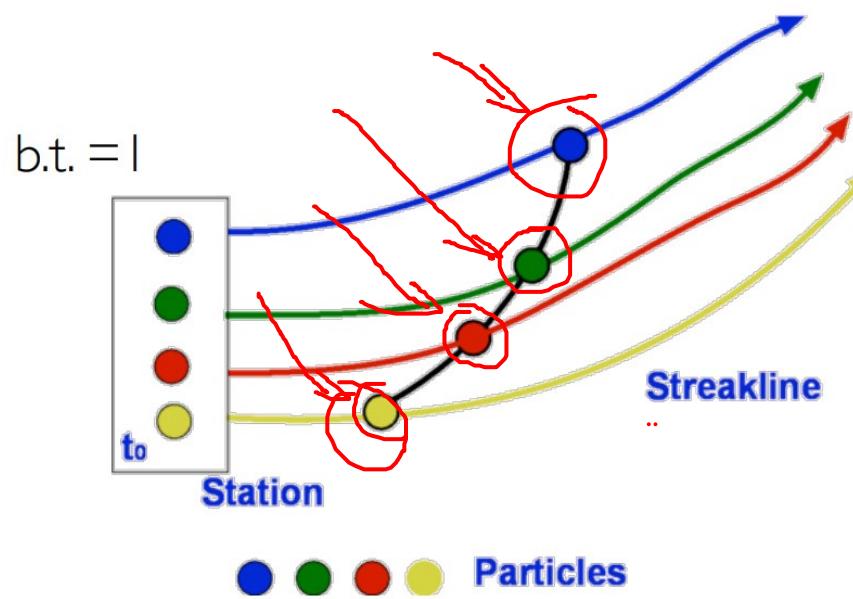
Extension of streamlines for time-varying data (unsteady flows)

Timeline draws a line through adjacent particles in flow at any instant of time

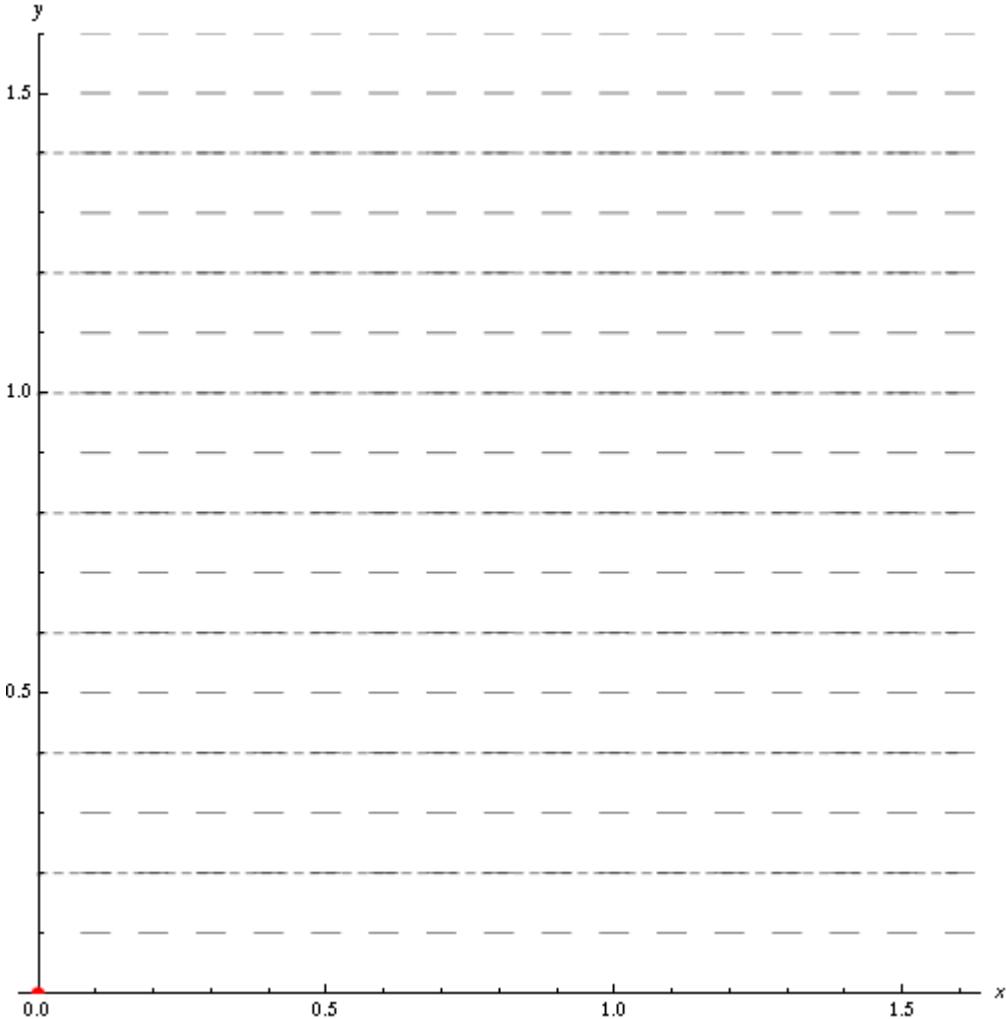


Streaklines

- For unsteady flows
- Continuously injecting a new particle at each time step
- Advectiong all the existing particles and connect them together into a *streakline*
- i.e. connecting particles that have gone through a fixed point in the domain

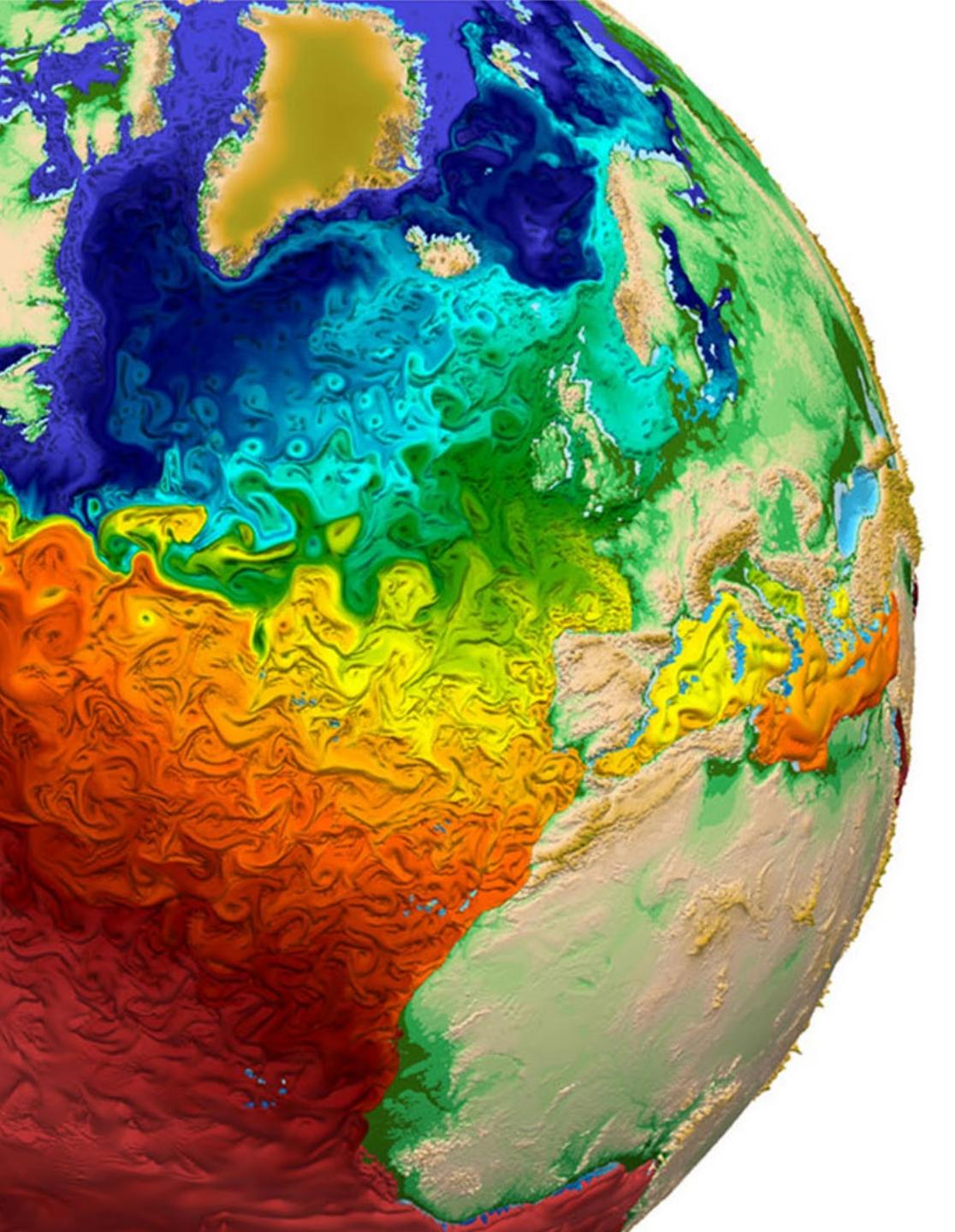


Pathlines and Streaklines



The red particle moves in a flowing fluid; its pathline is traced in red; the tip of the trail of blue ink released from the origin follows the particle, but unlike the static pathline (which records the earlier motion of the dot), ink released after the red dot departs continues to move up with the flow. (This is a streakline.) The dashed lines represent contours of the velocity field (streamlines), showing the motion of the whole field at the same time.

- Wikipedia



Numerical Methods

Sampling Domains

Scientific Visualization
Professor Eric Shaffer

2D Sampling

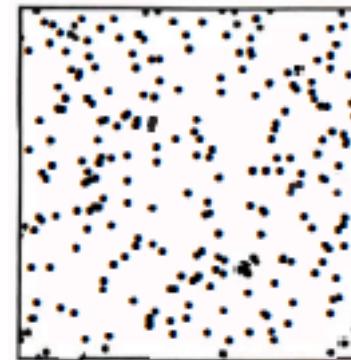
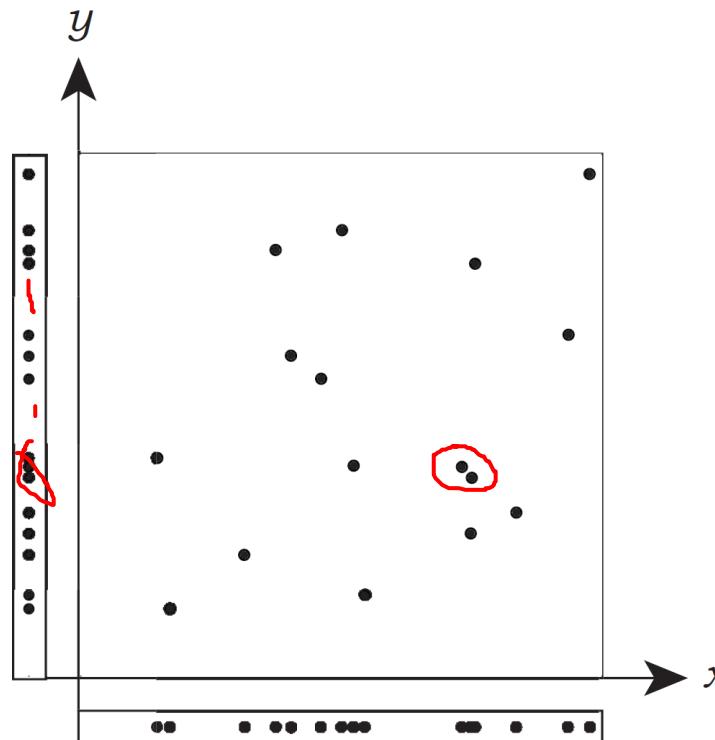
- Assume we are sampling a function on a unit square
- Good sampling
 - Not structured...involves some amount of randomness
 - Uniform(ish) distribution...avoid gaps and clumps
 - Projections into 1D along x and y are also uniform(ish)
 - There is a non-trivial minimum distance between all sample points
- Such a sample pattern is called *Well-Distributed*

Random

Too irregular

Oversamples some areas...

Undersamples others



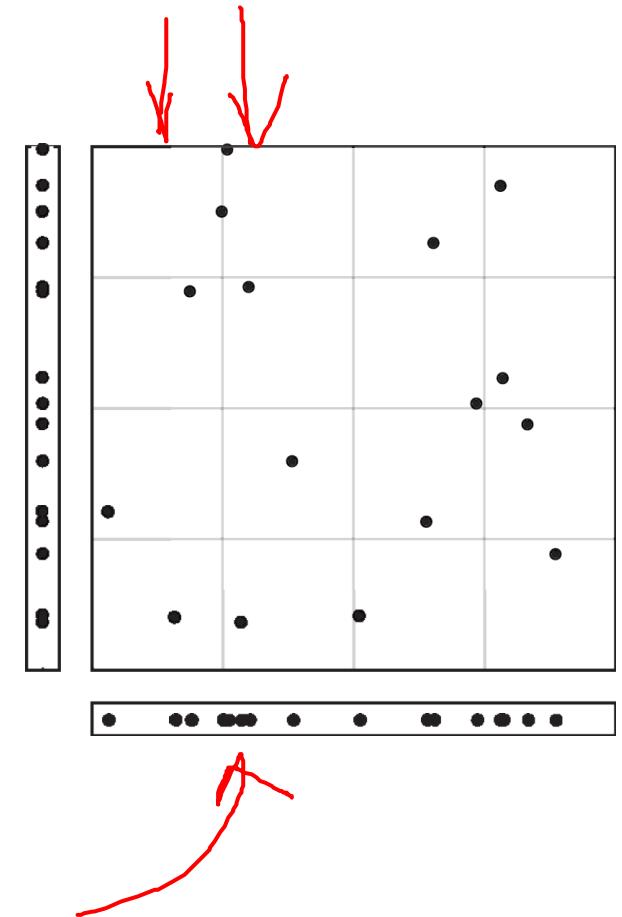
By random we mean a floating point values generated by a pseudo-random number generator of reasonable quality like Python's:

`random.random()`

Return the next random floating point number in the range [0.0, 1.0).

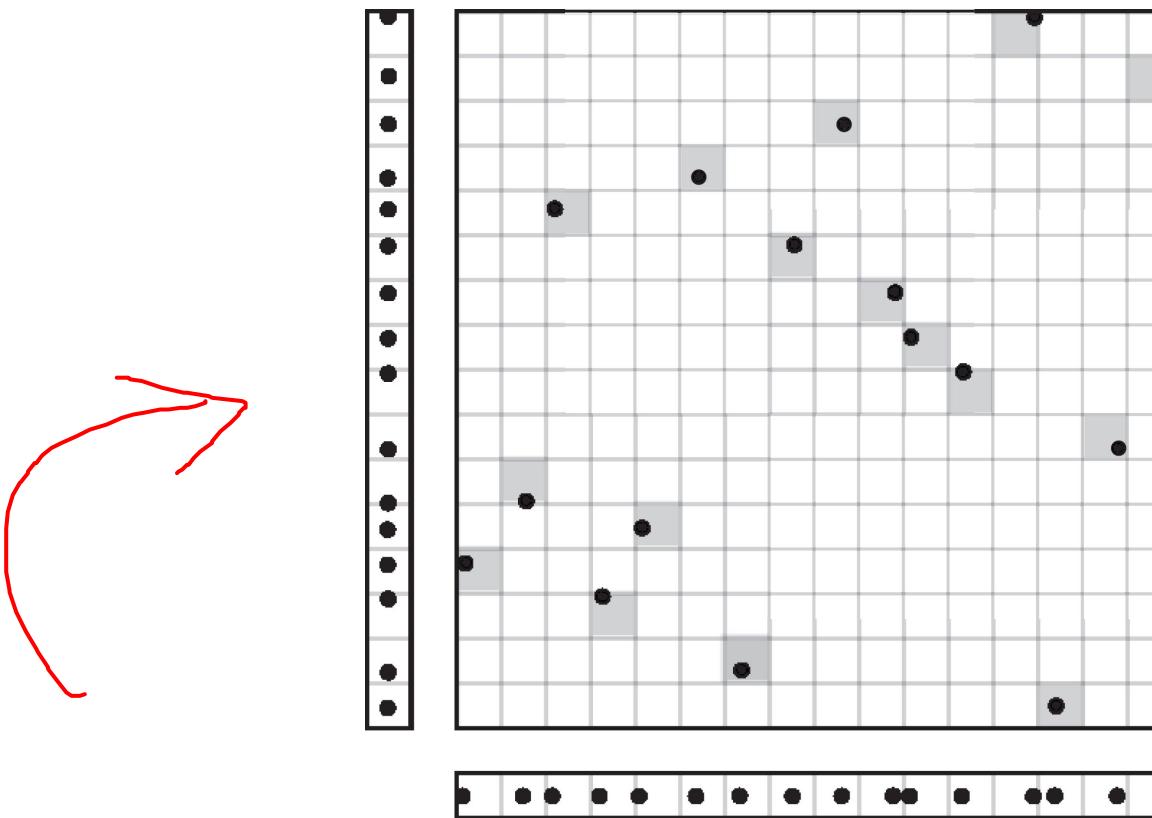
Jittered

- Create a $n \times n$ grid covering the domain
 - Or a n^d uniform grid in d dimensional space
- Randomly generate a sample in each cell
- Example of *stratified* sampling
 - Each cell is a strata
- Significantly better than random
- x-y projections can still be poorly distributed



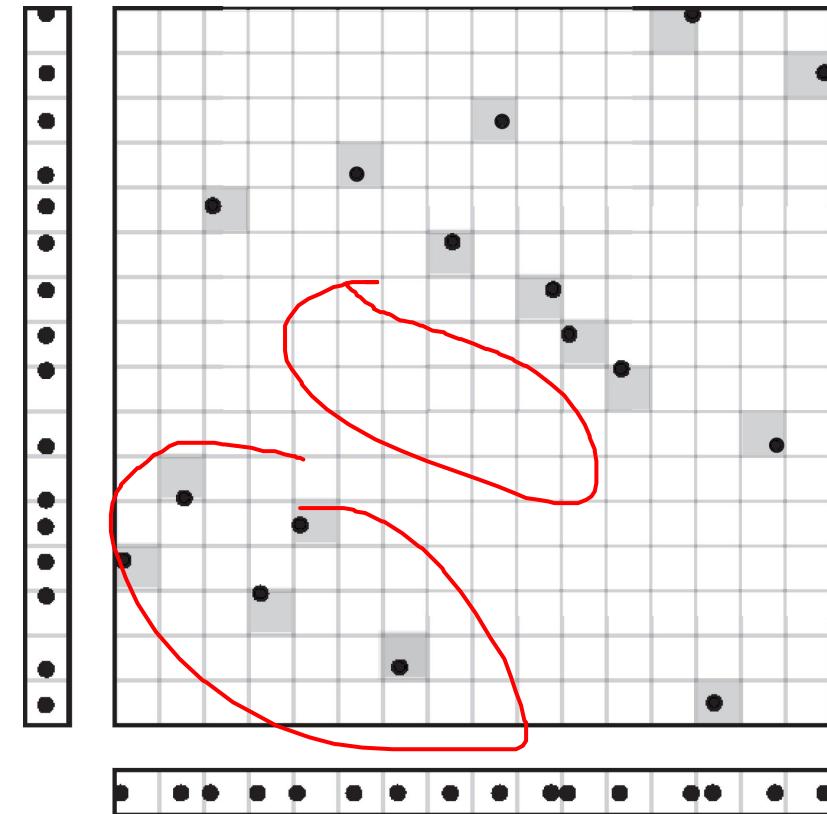
n-rooks

- Also called Latin hypercube sampling
- Use an $n \times n$ grid
- One sample exactly in each row and column
 - Again, randomly position a sample within the cell containing it
 - If samples were rooks in chess, no captures can occur



n-rooks

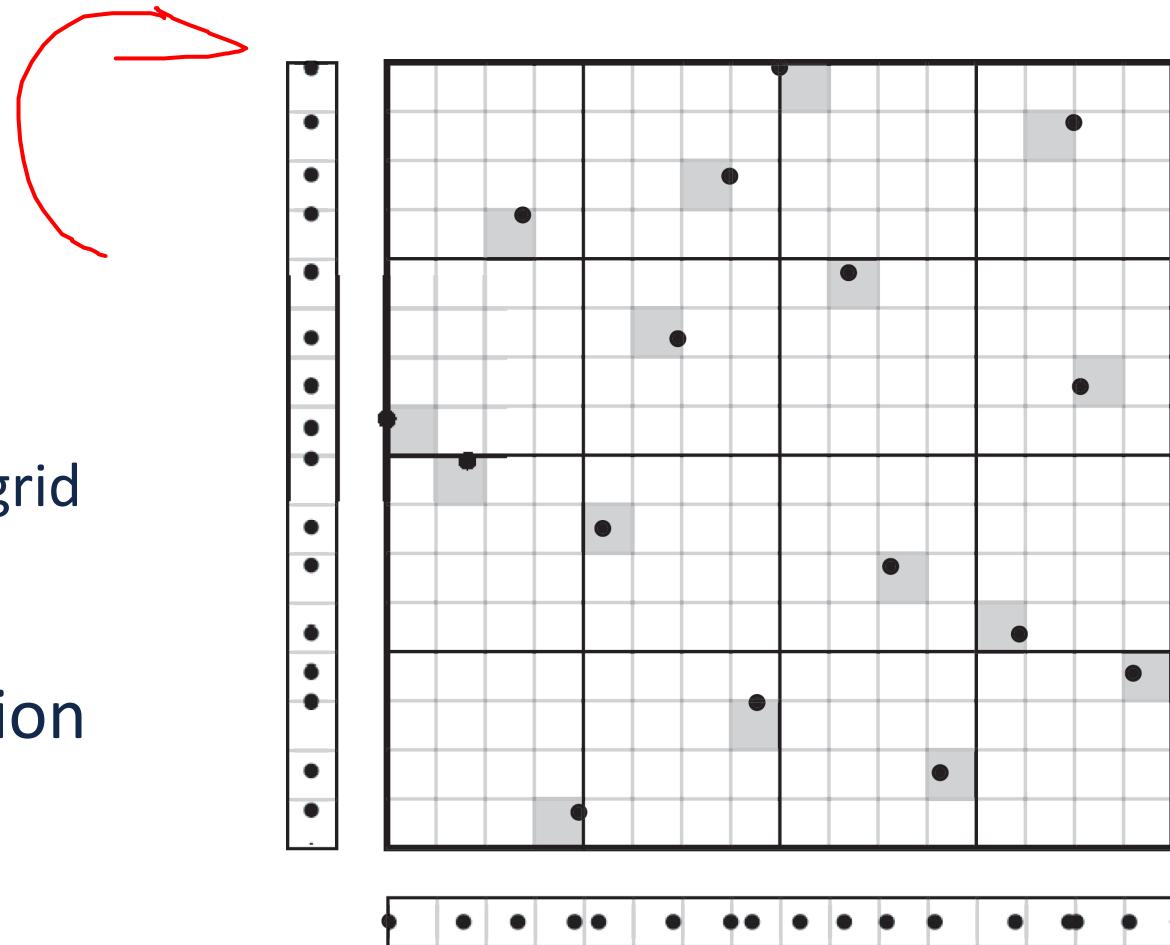
- Produced by random shuffle of diagonal samples
 - Must maintain the rook condition
- Use n samples instead of n^2 as in jittered
- 1D distributions are good...better than jittered
- 2D barely better than random...worse than jittered



Developed by UIUC
alum Pete Shirley
in 1991

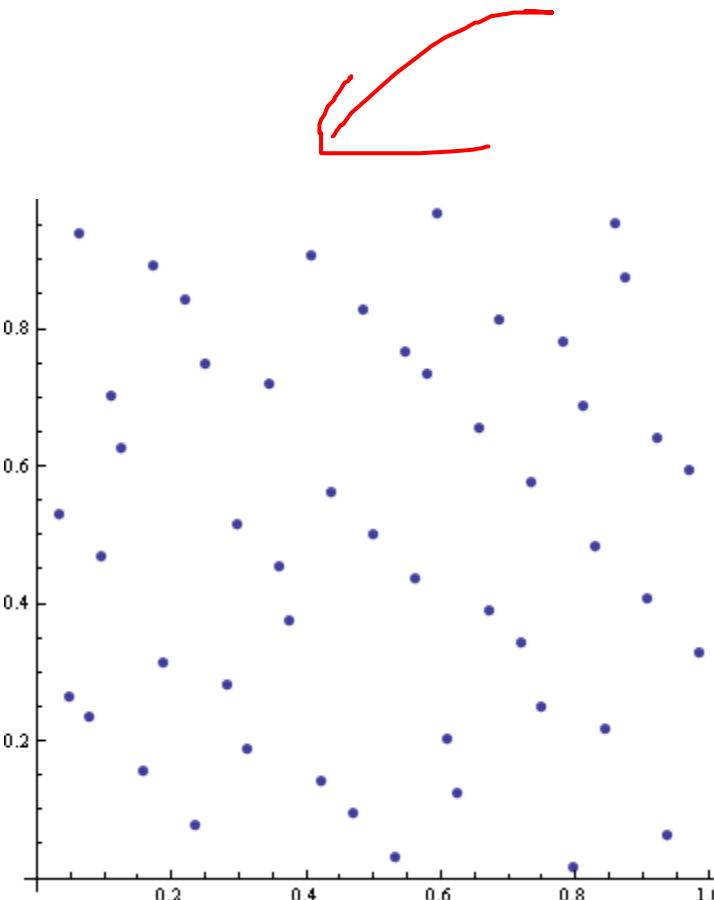
Multi-Jittered Sampling

- We use two grids
- For n samples with n a perfect square
 - Coarse grid is $\sqrt{n} \times \sqrt{n}$
 - Fine grid is $n \times n$
 - 1 sample per coarse grid cell
 - For each select unique row & column of fine grid
 - Randomly position sample in fine grid cell
- Good 1D projections from the rook condition
- Good 2D distribution from stratification
- Very good sampling technique

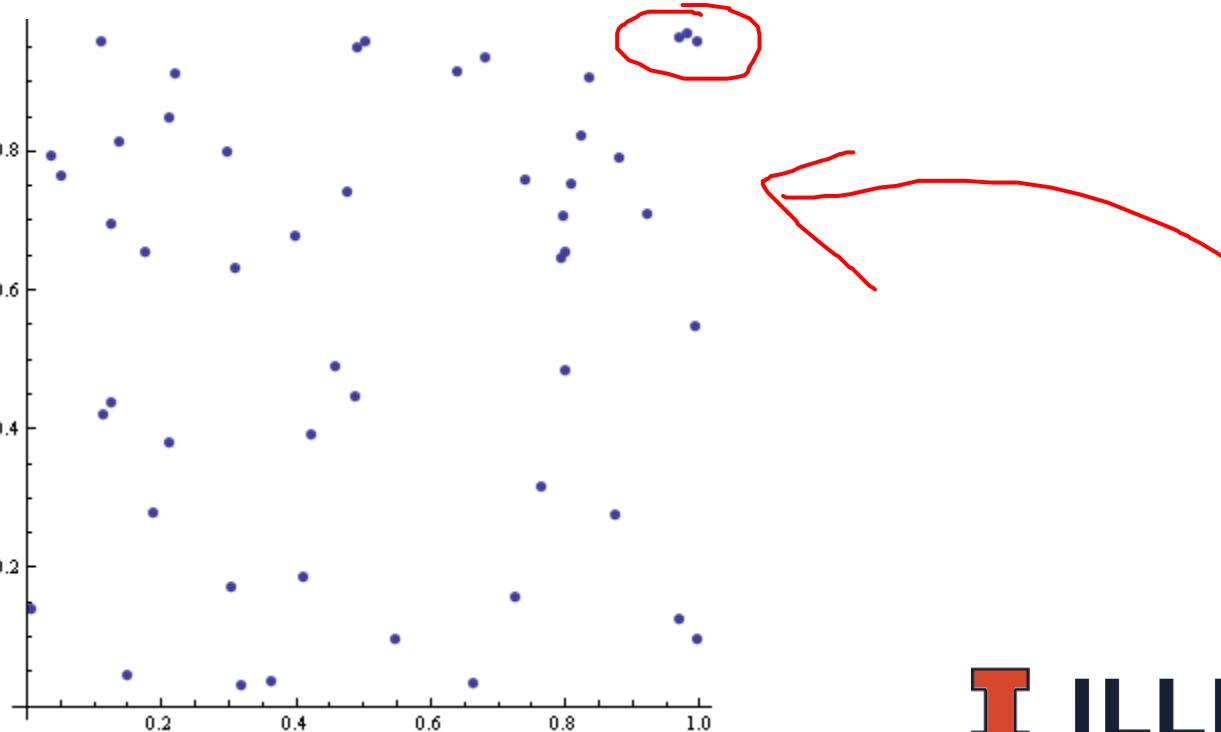


Hammersley Sampling

Deterministic quasi-random sequence



Low-discrepancy sequences are also called quasirandom sequences, due to their common use as a replacement of uniformly distributed random numbers. The "quasi" modifier is used to denote more clearly that the values of a low-discrepancy sequence are neither random nor pseudorandom.
- Wikipedia



Hammersley Sampling

$$\Phi_2(i) = \sum_{j=0}^n a_j(i) 2^{-j-1} = a_0(i) \frac{1}{2} + a_1(i) \frac{1}{4} + a_2(i) \frac{1}{8} \dots$$

Radical inverse function of integer i to base 2

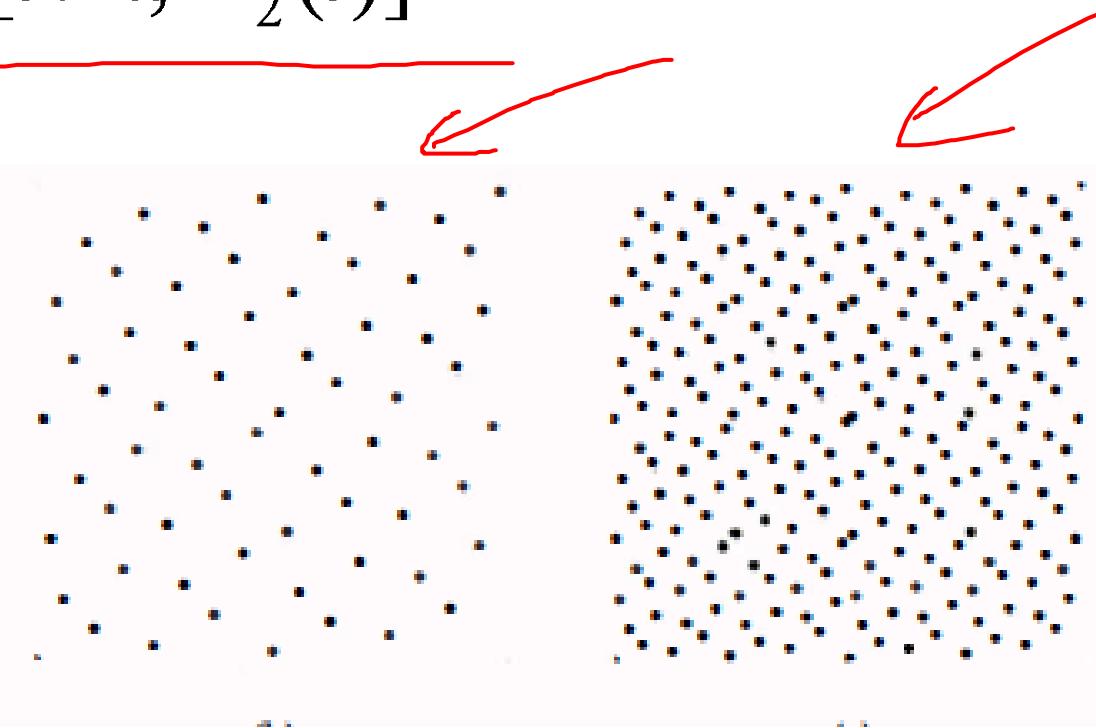
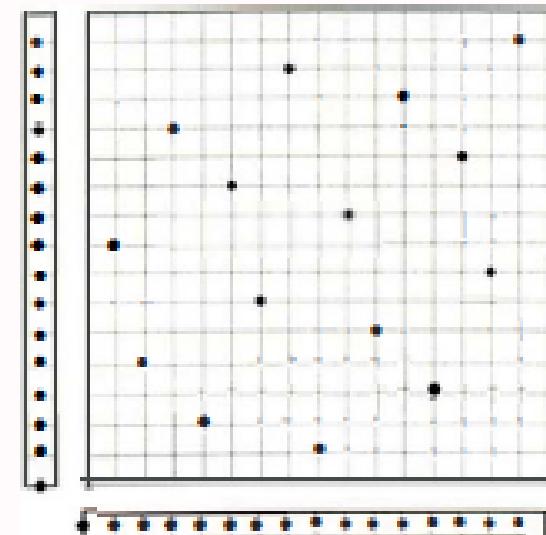
- reflect binary digits of i across decimal point
- evaluate this new number now in $[0,1)$

i	Reflection around the Decimal Point	$\Phi_2(i)$ (base 2)
1	$.1_2 = 1/2$	0.5
2	$.01_2 = 1/4$	0.25
3	$.11_2 = 1/2 + 1/4$	0.75
4	$.001_2 = 1/8$	0.125
5	$.101_2 = 1/2 + 1/8$	0.635
6	$.011_2 = 1/4 + 1/8$	0.325
7	$.111_2 = 1/2 + 1/4 + 1/8$	0.875
8	$.0001_2 = 1/16$	0.0625

Hammersley Sequence in 2D

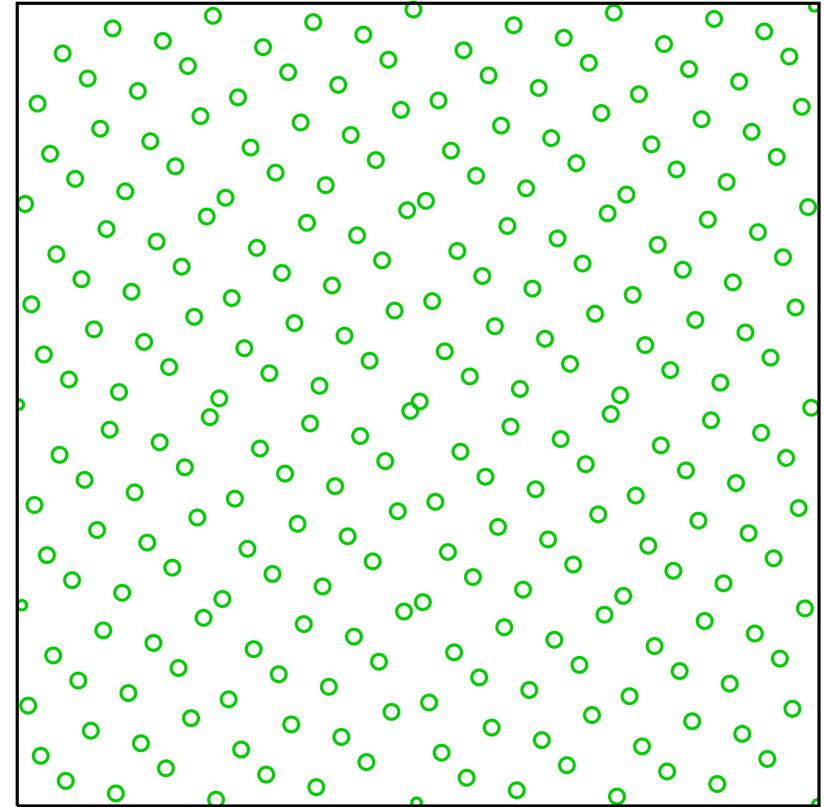
Set of n 2D samples in unit square:

$$\underline{p_i = (x_i, y_i) = [i / n, \Phi_2(i)]}$$



Hammersley Issues

- Pattern is well-distributed but...
- 1D projections are regular
 - Too much structure in the pattern
- For a given n only one sequence exists



Halton Sequence

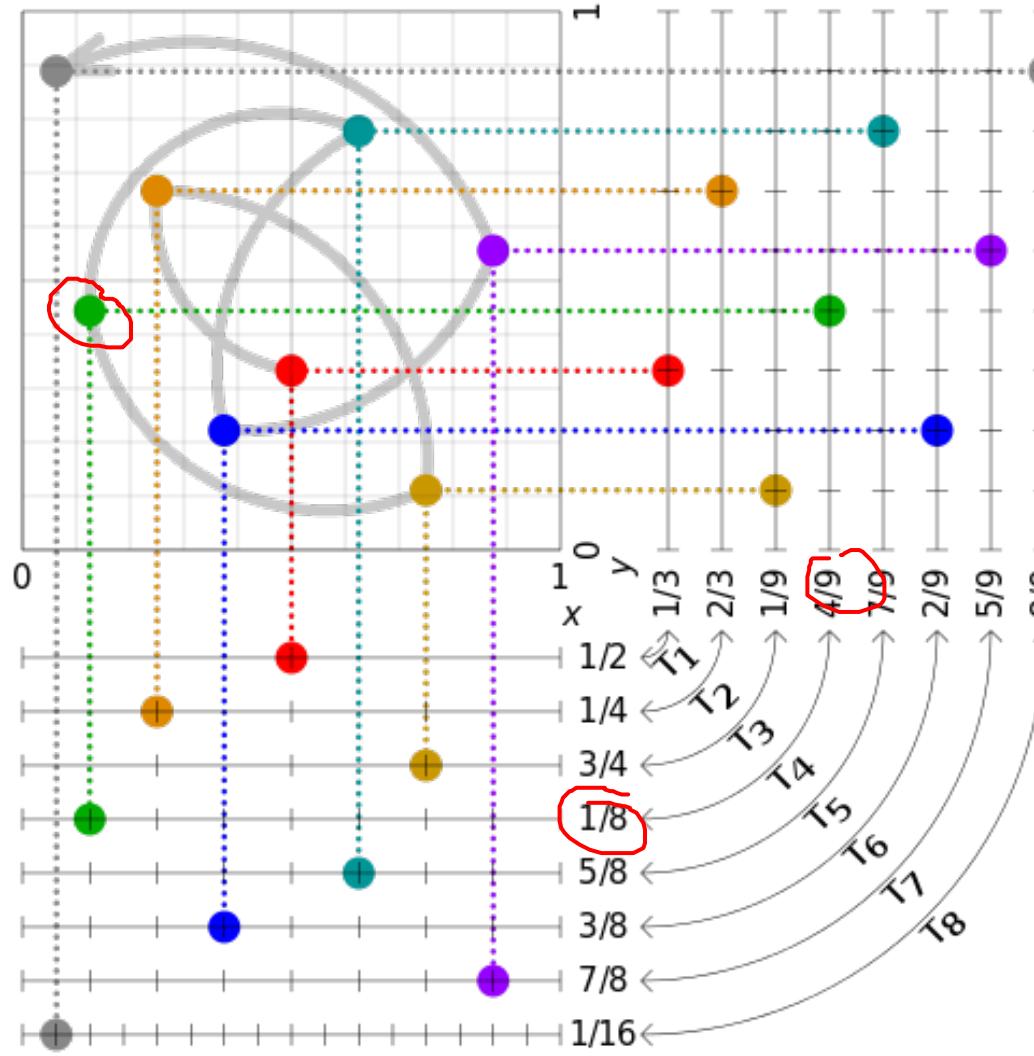
$$p_i = (\Phi_2(i), \Phi_3(i), \Phi_5(i), \dots)$$

- Better low discrepancy sequence
- Generate n-dimensional points
 - though Hammersley can be generalized as well....
- Number of samples need not be known in advance

Halton Sequence Example

The 2,3 Halton Sequence

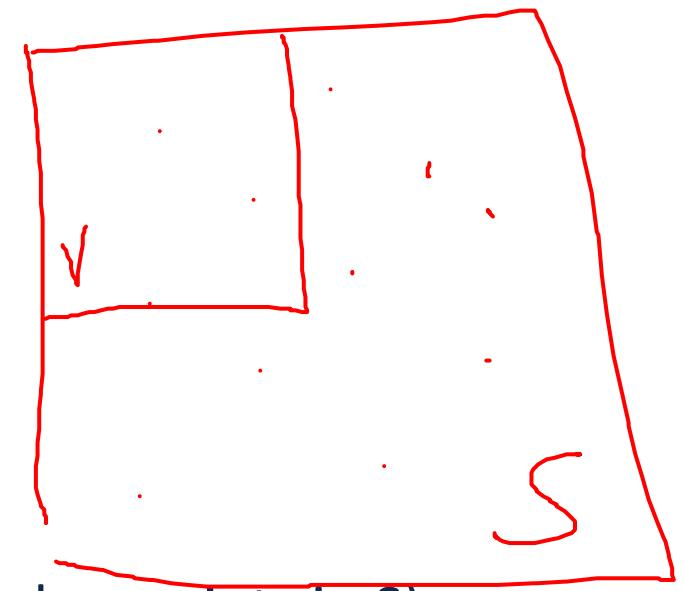
$\frac{1}{2} \left(\frac{1}{3}\right) =$
.001



$\frac{1}{2} \left(\frac{1}{3}\right) =$
0.11

Definition: Discrepancy

- We often want a low discrepancy sequence
 - e.g. Monte Carlo methods
- Imagine points in some space $S = [0,1]^n$
- Suppose we sample using K points...
- We can evaluate the quality by
 - take portion V of S
 - volume V /volume S should equal (number points in V)/(number points in S)
 - ...but it generally won't
 - the difference is the *discrepancy*
- Different formal ways to measure discrepancy...



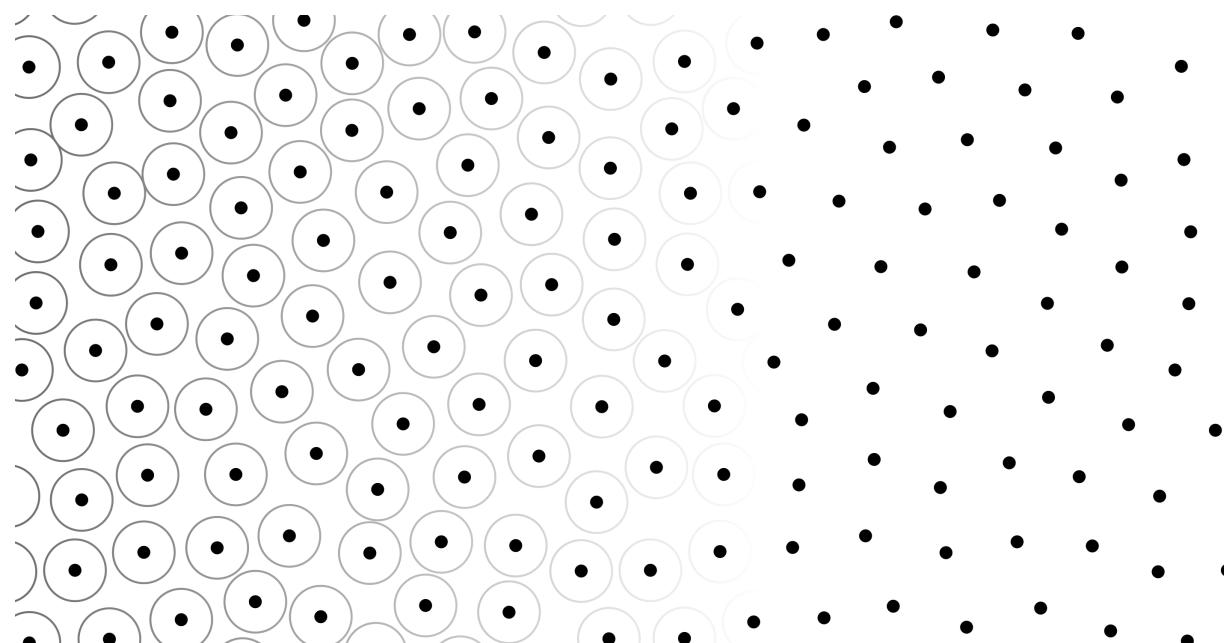
Poisson Disk Sampling

Not actually a low-discrepancy sequence

Does assure a minimum distance between points

Fast and generalizes well to higher dimensions

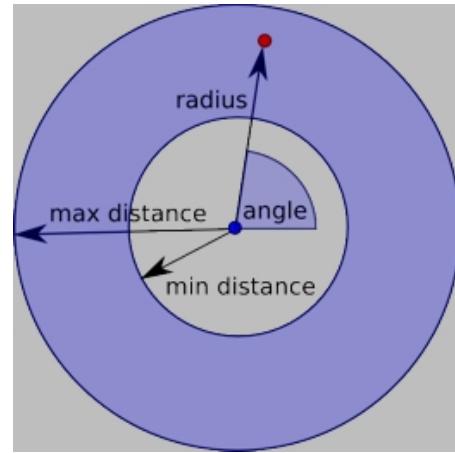
Popular in game industry



Poisson Disk Sampling

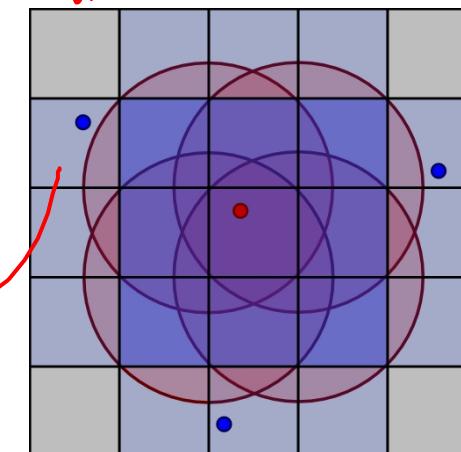
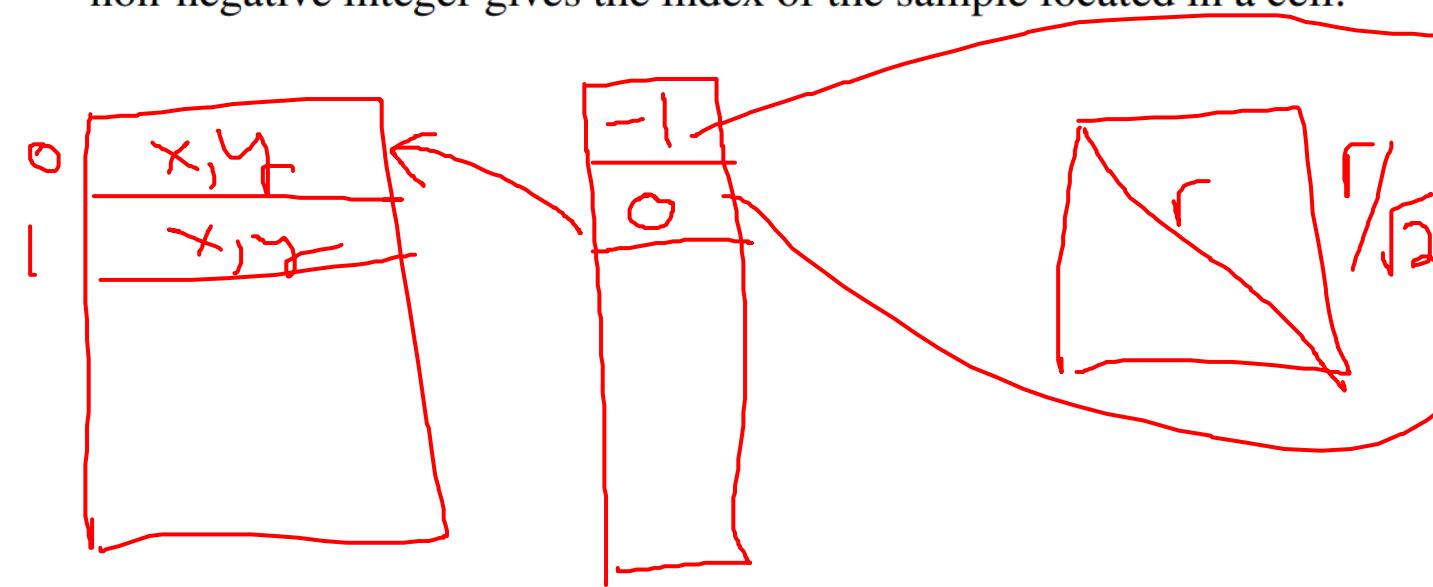
Algorithm

Step 0. Initialize an n -dimensional background grid for storing samples and accelerating spatial searches. We pick the cell size to be bounded by r/\sqrt{n} , so that each grid cell will contain at most one sample, and thus the grid can be implemented as a simple n -dimensional array of integers: the default -1 indicates no sample, a non-negative integer gives the index of the sample located in a cell.



r is minimum distance between two sample points

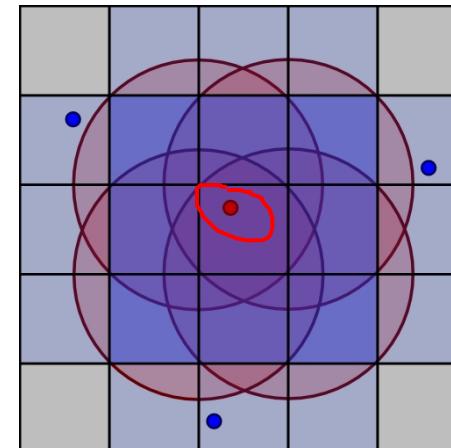
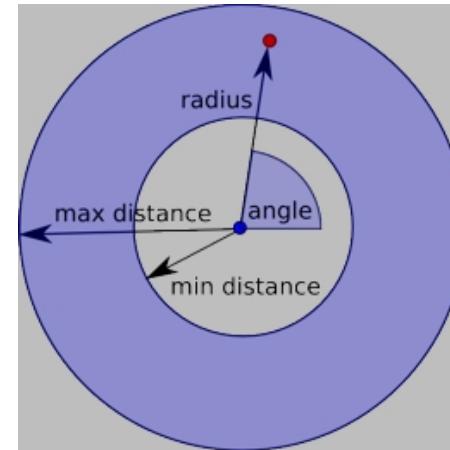
r will be the length of a cell diagonal



Poisson Disk Sampling

Algorithm

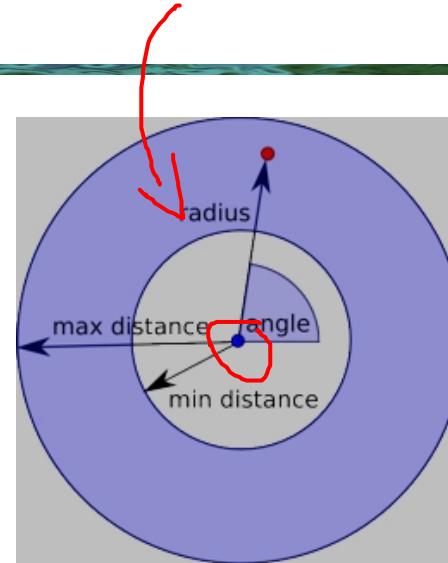
Step 1. Select the initial sample, x_0 , randomly chosen uniformly from the domain. Insert it into the background grid, and initialize the “active list” (an array of sample indices) with this index (zero).



Poisson Disk Sampling

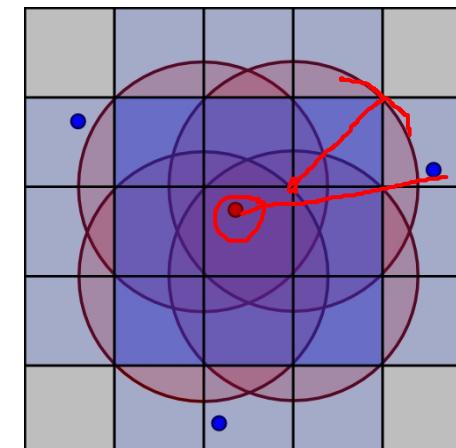
Algorithm

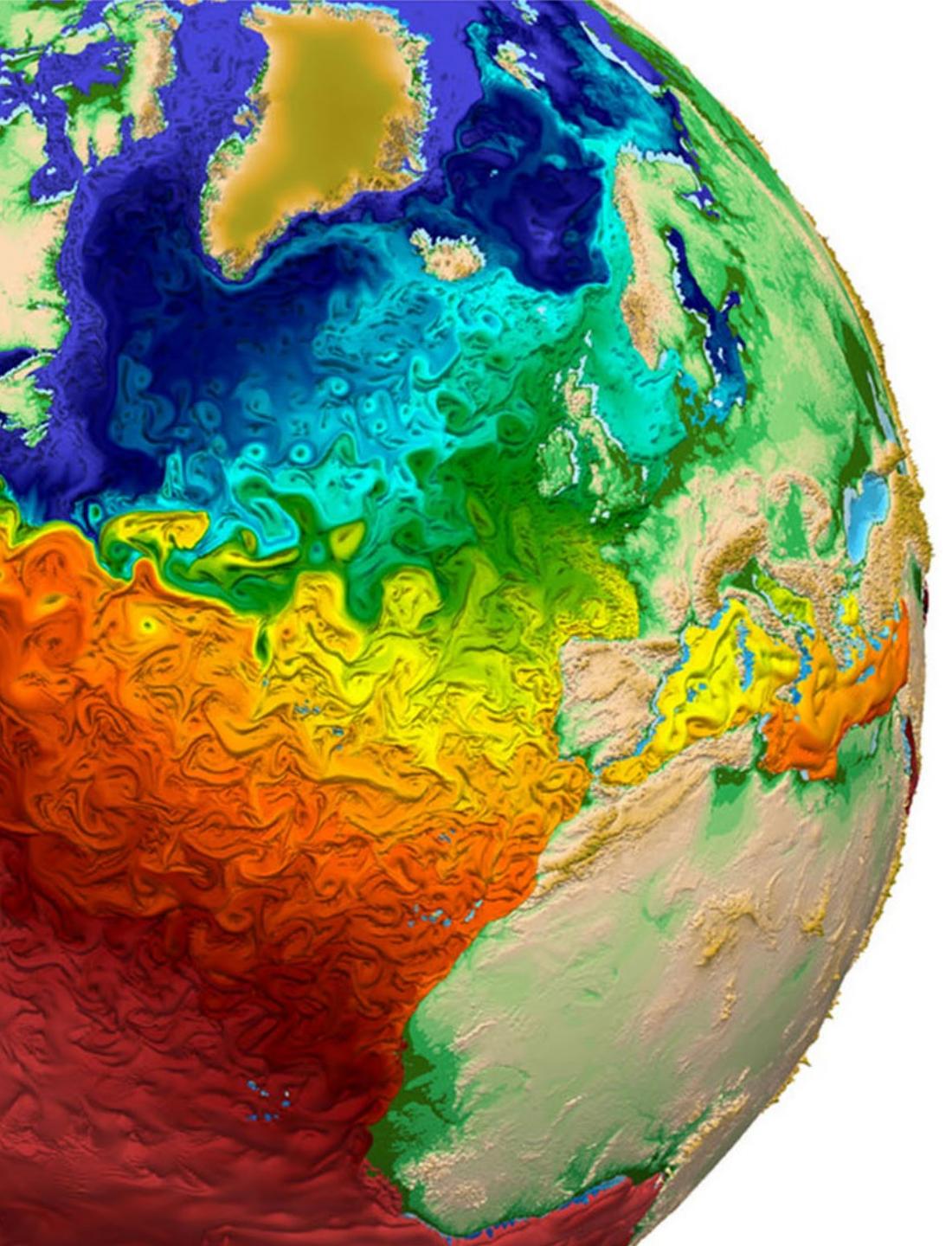
Step 2. While the active list is not empty, choose a random index from it (say i). Generate up to k points chosen uniformly from the spherical annulus between radius r and $2r$ around x_i . For each point in turn, check if it is within distance r of existing samples (using the background grid to only test nearby samples). If a point is adequately far from existing samples, emit it as the next sample and add it to the active list. If after k attempts no such point is found, instead remove i from the active list.



r will be the length of a cell diagonal

$k=30$ is a popular choice





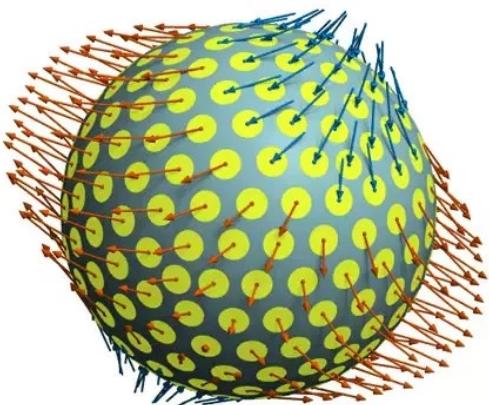
Tensor Visualization

What are Tensors?

Scientific Visualization
Professor Eric Shaffer

Tensors

“A tensor is just a machine that takes in some number of vectors and spits out some other vectors in a linear fashion. For example, the dot product can be viewed as a tensor that takes two vectors in and spits out a number.”

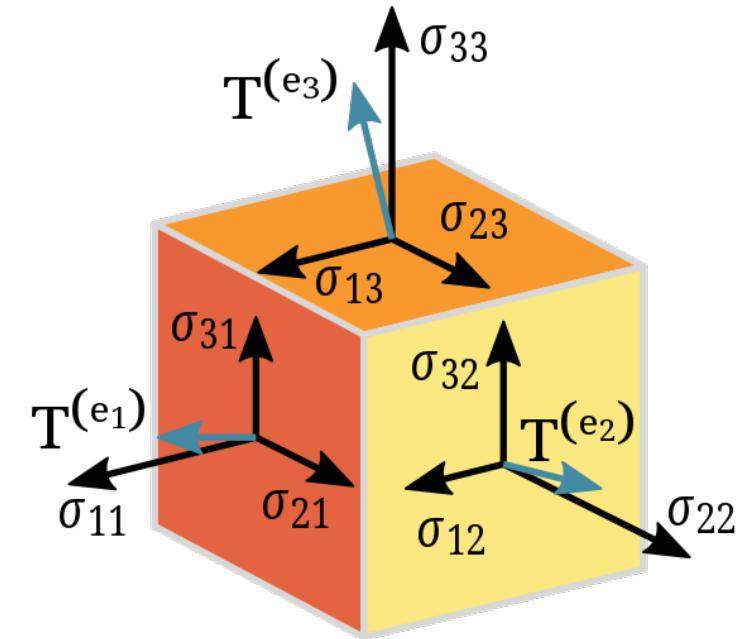


Stress forces acting on a particle in a homogeneous continuous medium under a uniform linear stress, as a function of the orientation of a surface element on the particle's boundary.

What is a Tensor

In [mathematics](#), a **tensor** is an algebraic object that describes a ([multilinear](#)) relationship between sets of algebraic objects related to a [vector space](#). Objects that tensors may map between include [vectors](#) and [scalars](#), and even other tensors.

-wikipedia



Is Tensor a Matrix?

A tensor is often thought of as a generalized matrix.

- A multi-dimensional array of numbers
- A rank 2 tensor is a matrix
- Not all matrices are tensors

A tensor is an N-dimensional array of data



Rank 0
Tensor
scalar

Rank 1
Tensor
vector

Rank 2
Tensor
matrix

Rank 3
Tensor

Rank 4
Tensor

The TensorFlow machine learning platform name derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors. ...

What is a Tensor?

Explanation 1: Dimensionality

- scalar: a **0D** array of values e.g. 1 value
- vector: a **1D** array of values e.g. 3 values
- tensor: a **2D** matrix of values e.g. $3 \times 3 = 9$ values

A tensor is an N-dimensional array of data

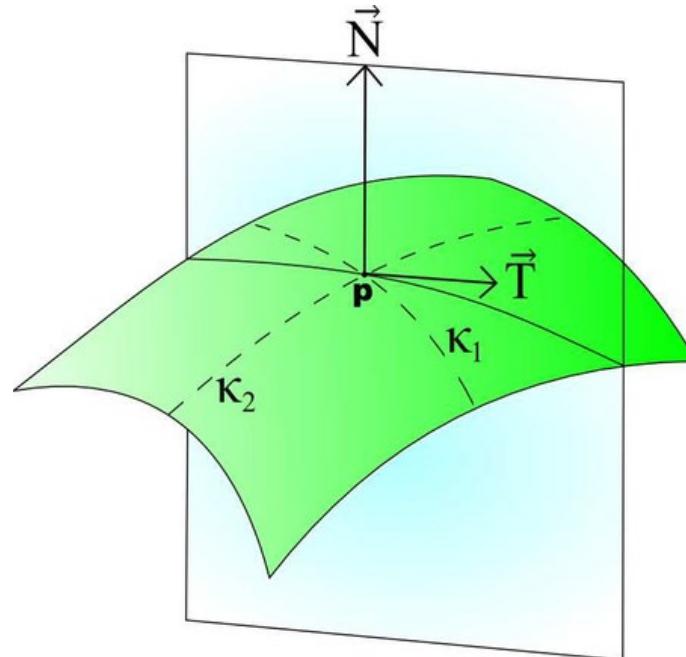


We will focus only on rank-2 tensors

What is a Tensor?

Explanation 2: Analysis

- scalar: **magnitude** (of some signal at a point in space)
- vector: **magnitude** and **direction** (of some signal at some point in space)
- tensor: **variation of magnitude** (of some signal at some point in space)



What is a Tensor?

Explanation 3: As a function

- **scalar:** at $\mathbf{x} \in \mathbf{R}^3$, measure some value $s \in \mathbf{R}$
- **vector:** at $\mathbf{x} \in \mathbf{R}^3$, measure some magnitude and direction $\mathbf{v} \in \mathbf{R}^3$
- **tensor:** at $\mathbf{x} \in \mathbf{R}^3$ and in a direction $\mathbf{v} \in \mathbf{R}^3$, measure some magnitude $s \in \mathbf{R}$

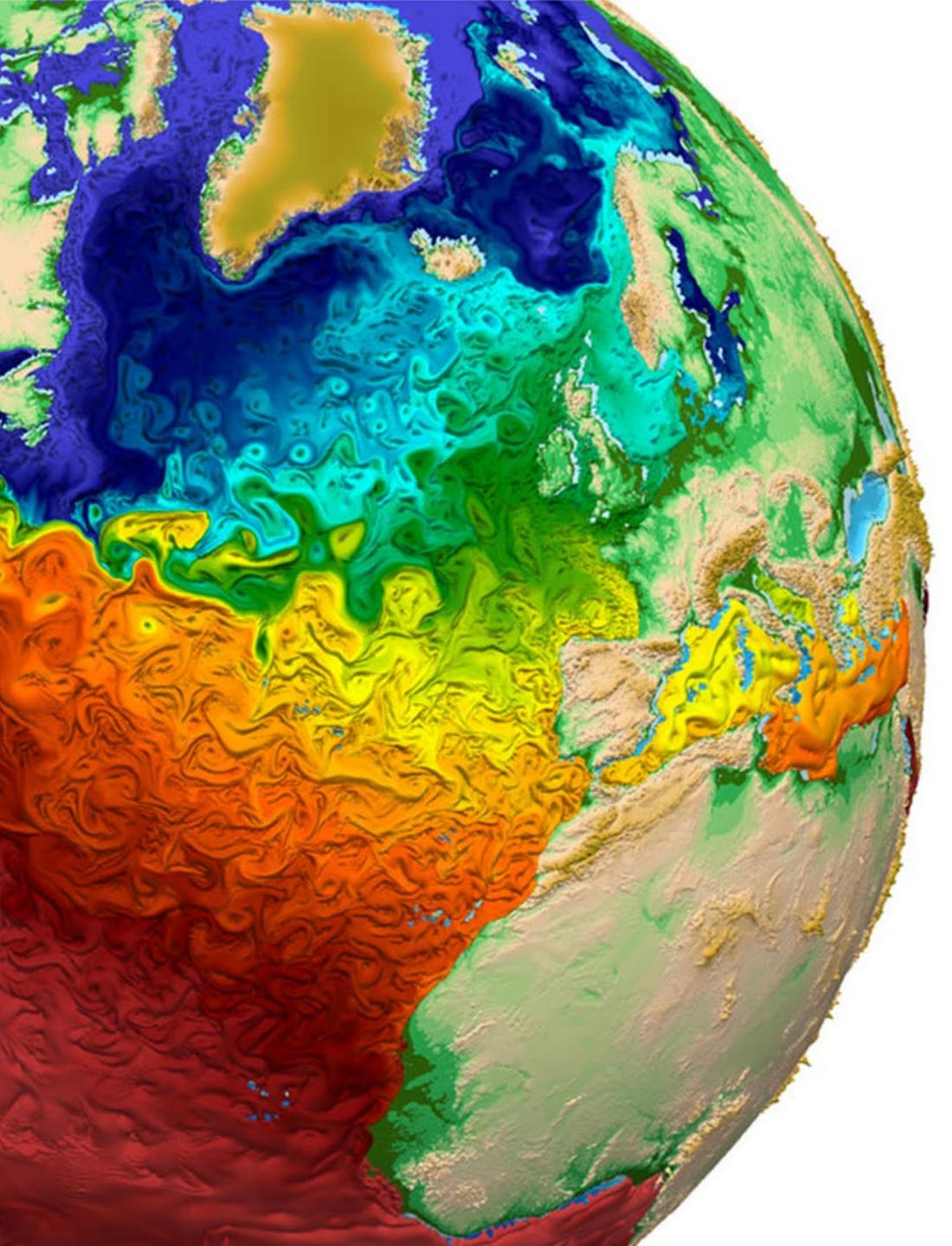
Fields

So we have different kinds of fields (i.e. **functions** of a variable $\mathbf{x} \in \mathbf{R}^3$):

Scalar fields $s : \mathbf{R}^3 \rightarrow \mathbf{R}$

Vector fields $\mathbf{v} : \mathbf{R}^3 \rightarrow \mathbf{R}^3$

Tensor fields $\mathbf{T} : \mathbf{R}^3 \times \mathbf{R}^3 \rightarrow \mathbf{R}$



Tensor Visualization

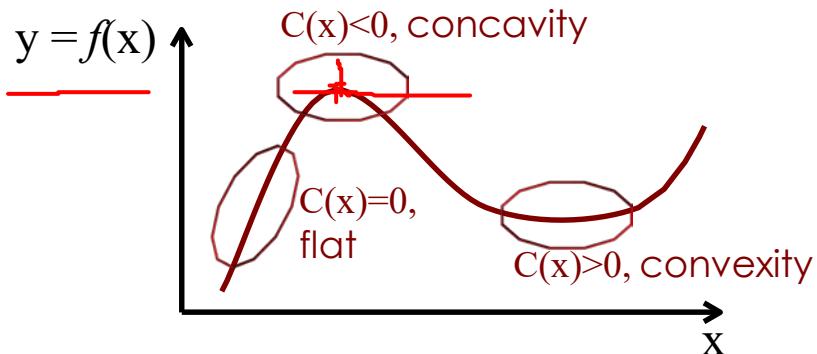
Curvature Tensor

Scientific Visualization
Professor Eric Shaffer

Curvature

Curvature in 1D

- take a curve c
- locally, c can be described as a function $y = f(x)$



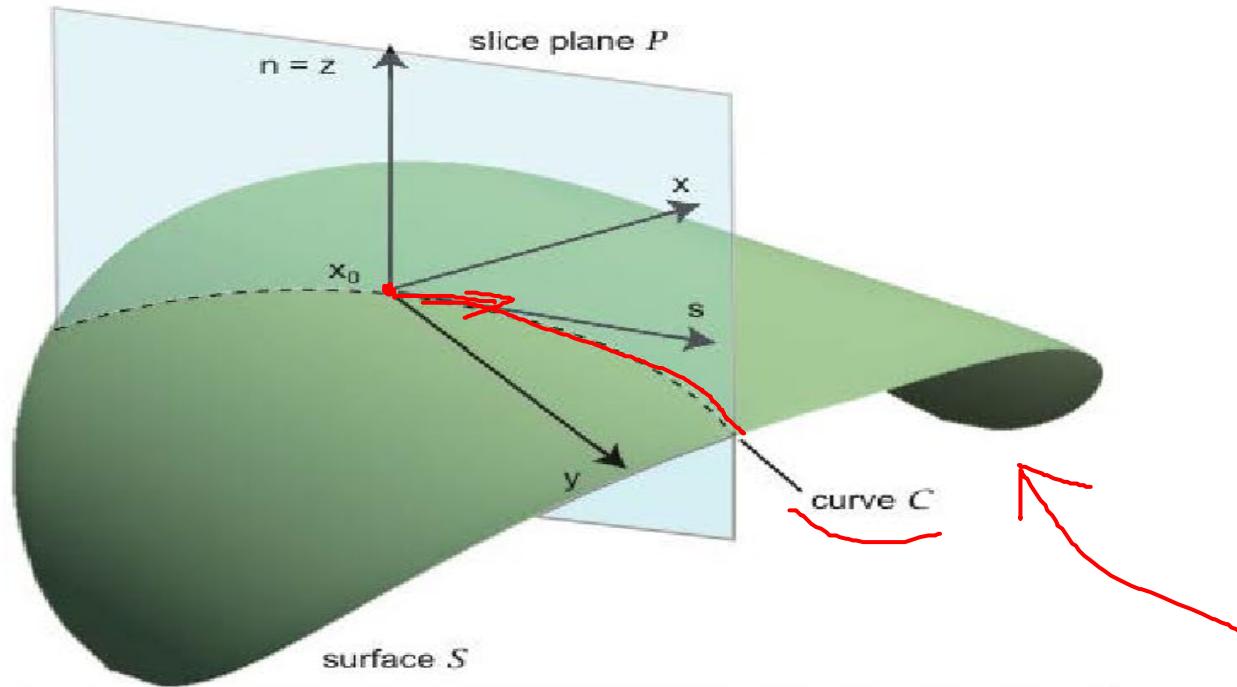
- curvature of f $C(x) = \frac{\partial^2 f}{\partial x^2}$ (2nd derivative of f)
- analytically: $C(x) =$ how quickly the normal \mathbf{n}_c changes around x
(why? Because the tangent to c is $\partial f / \partial x$ and its change is $\partial^2 f / \partial x^2$)

Curvature in 2D

- take a surface $S \subset \mathbf{R}^3$
- at each $x_0 \in S$
 - take a coordinate system xyz with x,y tangent to S and z along \mathbf{n}_S
 - locally, S can be described as a function $z = f(x,y)$

How to describe 2D curvature?

- 1D analogy: how quickly the normal \mathbf{n}_S changes around x_0
- problem: we have a surface – in which direction to look for change?



We must compute
$$C(x,s) = \frac{\partial^2 f(x)}{\partial s^2}$$
for any direction s



Curvature Tensor

$$C(x, s) = \frac{\partial^2 f(x)}{\partial s^2}$$

- recall our definition of a tensor $\mathbf{T} : \mathbf{R}^3 \times \mathbf{R}^3 \rightarrow \mathbf{R}$? The above is precisely that

Also note that

$$\frac{\partial^2 f}{\partial s^2}(x_0) = \mathbf{s}^T H \mathbf{s}$$

where H is the so-called **Hessian** of f

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}$$

In other words, if we have H , we can compute the curvature tensor

- at any point x_0
- in any direction s

The Curvature Tensor

However, there's a problem with the previous definition

- we need to construct local coordinate systems at every point on S
- not obvious how to do that....

General solution:

Describe S as an implicit function (i.e. the zero-level isosurface of a function)

$$S = \{x \in \mathbf{R}^3 \mid f(x) = 0\} \text{ for a given } f: \mathbf{R}^3 \rightarrow \mathbf{R}$$

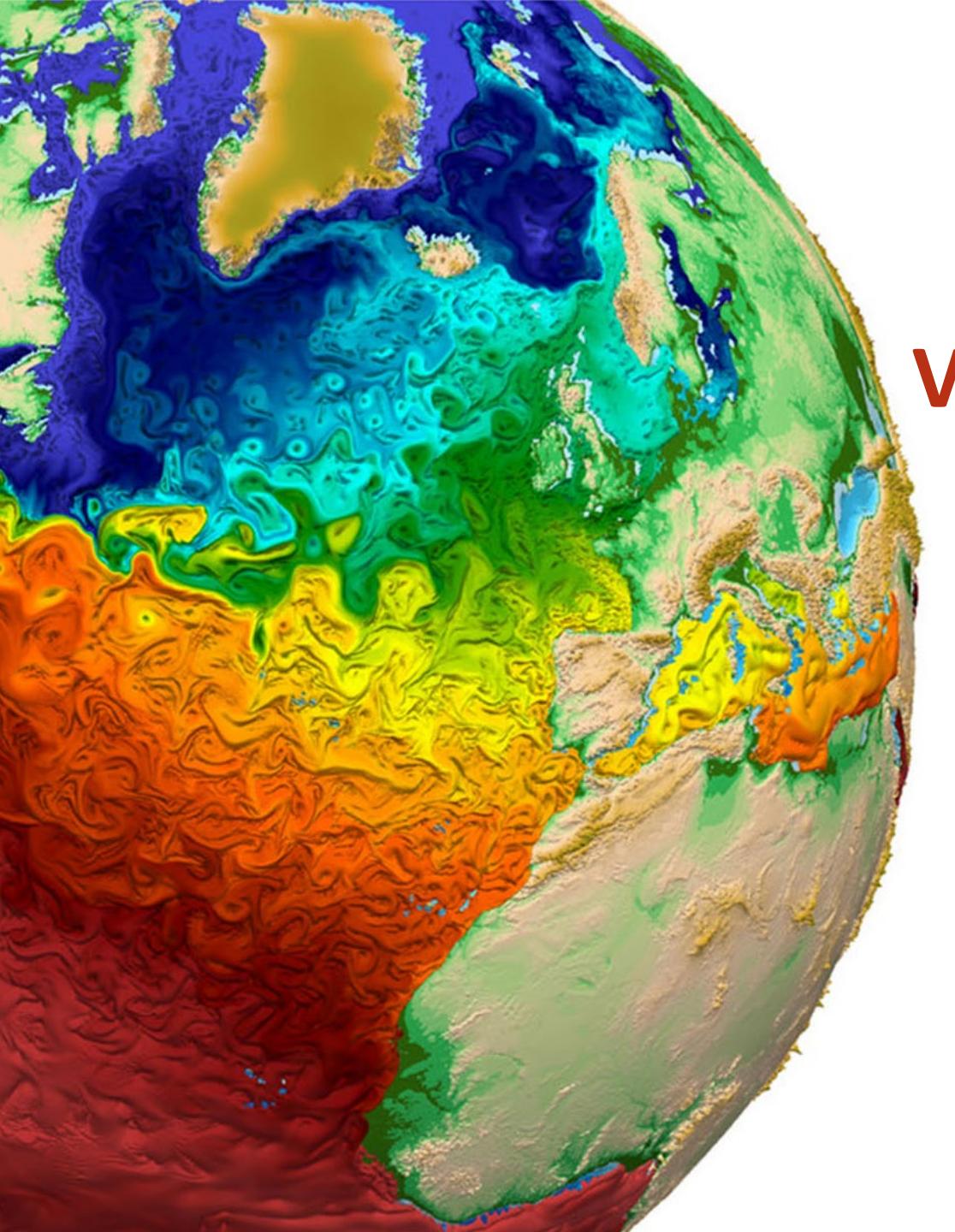
Then, we still have

$$\frac{\partial^2 f}{\partial s^2}(x_0) = \underline{s^T H s} \quad \text{where } H \text{ is the } 3 \times 3 \text{ Hessian matrix}$$

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} \end{pmatrix}$$

Conclusion

- A curvature tensor is fully described by a 3×3 matrix of 2nd order derivatives



Tensor Visualization

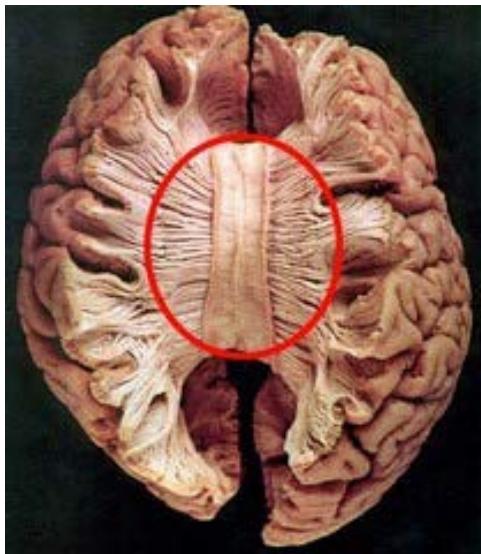
Visualizing the Diffusion Tensor

Scientific Visualization
Professor Eric Shaffer

The Diffusion Tensor

- consider an anisotropic material (e.g. tissue in the human brain)
- water diffuses in this tissue
 - **strongly** along neural fibers
 - **weakly** across fibers

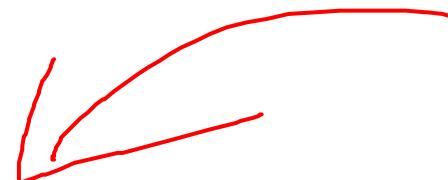
Actual image of a dissected human brain



Diffusion tensor

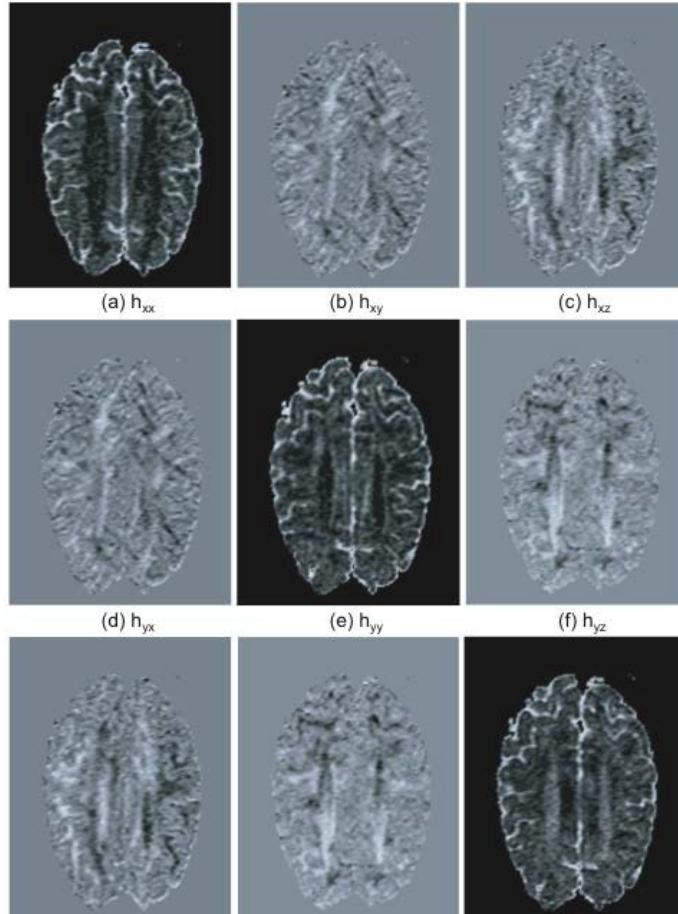
$$D(x, s) = \frac{\partial^2 f(x)}{\partial s^2}$$

diffusivity at a point x in a direction s



speed of water motion in tissue

The Diffusion Tensor



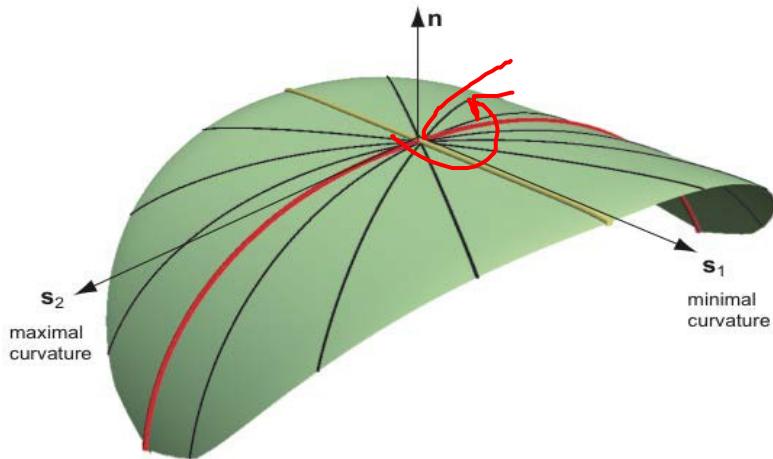
First visualization try

- compute hessian $H = \{h_{ij}\}$ in \mathbf{R}^3
- select some slice of interest
- visualize all components h_{ij} using e.g. color mapping

Simple, but not very useful

- we get a lot of images (9)...
- we see the tensor is symmetric...
- ...but we don't really care about diffusion along x, y, z axes!

Principal Component Analysis



$$C(x,s) = \frac{\partial^2 f(x)}{\partial s^2}$$

- fix some point x_0 on the surface
- compute $C(x_0,s)$ for all possible tangent directions s at x_0
- denote $\alpha = \text{angle of } s \text{ with local coordinate axis } x_0$

So we have

$$\frac{\partial^2 f}{\partial s^2} = \underbrace{s^T H s}_{= h_{11} \cos^2 \alpha + (h_{12} + h_{21}) \sin \alpha \cos \alpha + h_{22} \cos^2 \alpha}$$

Now, let's look for the values of α for which this function is extremal!

Principal Component Analysis

Our curvature (as function of α) is extremal when

$$\frac{\partial C}{\partial \alpha} = 0$$

This is equivalent to a system of equations

$$\begin{cases} h_{11} \cos \alpha + h_{12} \sin \alpha &= \lambda \cos \alpha \\ h_{21} \cos \alpha + h_{22} \sin \alpha &= \lambda \sin \alpha, \end{cases}$$
 which in matrix form is $H\mathbf{s} = \lambda \mathbf{s}$ or $(H - \lambda I)\mathbf{s} = 0$

Since we're looking for the non-trivial solution $\mathbf{s} \neq \mathbf{0}$ this means

$$\det(H - \lambda I) = (h_{11} - \lambda)(h_{22} - \lambda) - h_{12}h_{21} = 0$$

Solving the above 2nd order equation in λ yields

- two real values λ_1, λ_2 eigenvalues (principal values) of tensor

Plugging λ_1, λ_2 into $H\mathbf{s} = \lambda \mathbf{s}$ yields

- two direction vectors $\mathbf{s}_1, \mathbf{s}_2$ eigenvectors (principal directions) of tensor

Summarizing

- Given a 2x2 tensor, we can compute its principal directions and values
- directions: those in which tensor has extremal (minimal, maximal) values
- can be shown that eigendirections are orthogonal to each other
- eigenvalues: the actual minimal and maximal values

Principal Component Analysis

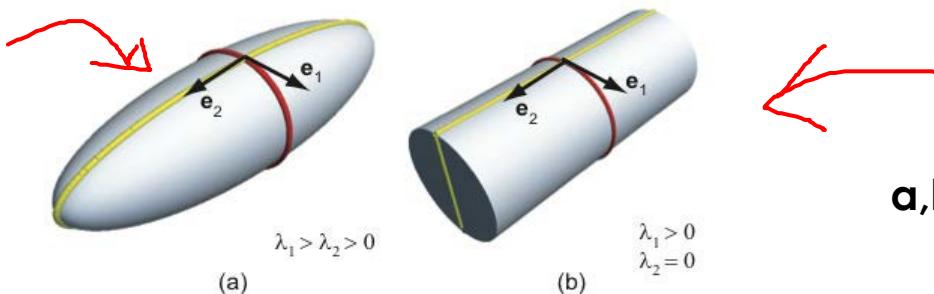
How about a 3×3 tensor, like the diffusion tensor?

- 3 eigenvalues, 3 eigenvectors (computed similarly, see Sec. 7.1)

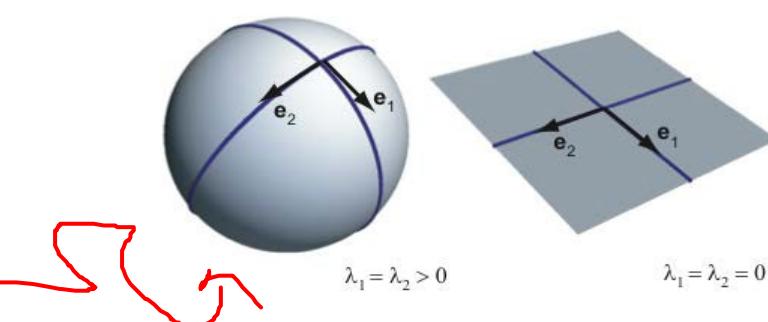
Say we order eigenvalues (and their vectors) as $\lambda_1 > \lambda_2 > \lambda_3$

- | | |
|------------------------------|---|
| $\underline{\lambda_1, s_1}$ | major eigenvector i.e. <u>direction of strongest diffusion</u> |
| λ_2, s_2 | medium eigenvector (no particular meaning) |
| $\underline{\lambda_3, s_3}$ | minor eigenvector i.e. <u>direction of weakest diffusion</u> |

What if two or more eigenvalues are equal (so we cannot fully order them all)?



a,b) all values ordered: unique eigendirections

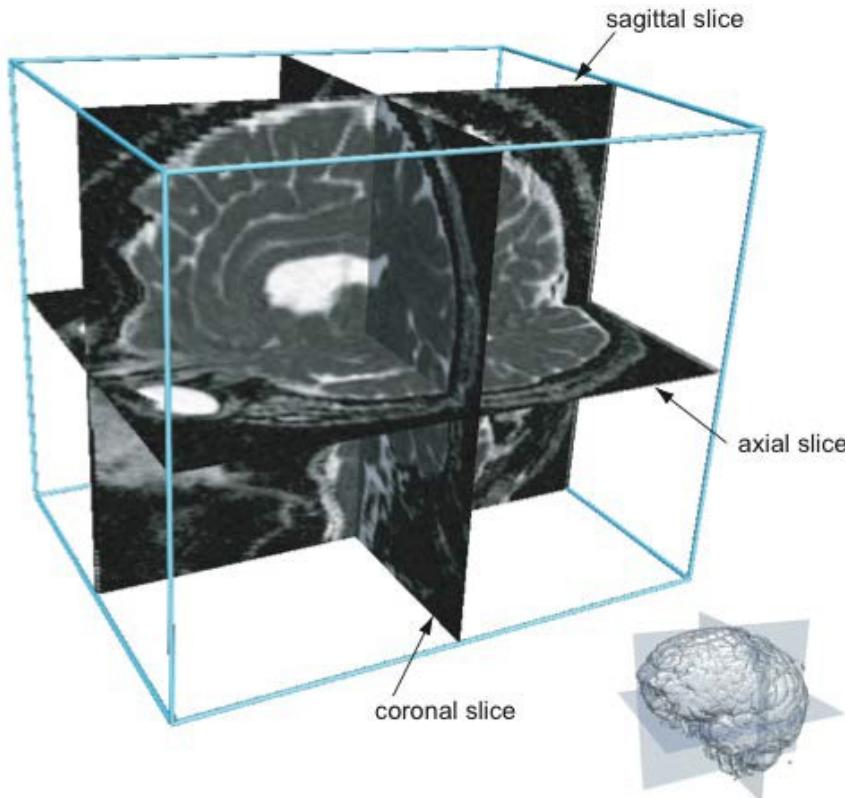


c,d) equal eigenvalues: eigendirections not determined (any two orthogonal vectors tangent to surface are valid eigendirections)

Principal Component Analysis

How to use PCA for visualization?

Visualize mean diffusivity $\mu = \frac{1}{3}(\lambda_1 + \lambda_2 + \lambda_3)$



white: strong mean diffusivity
black: weak mean diffusivity

Principal Component Analysis

Linear diffusivity

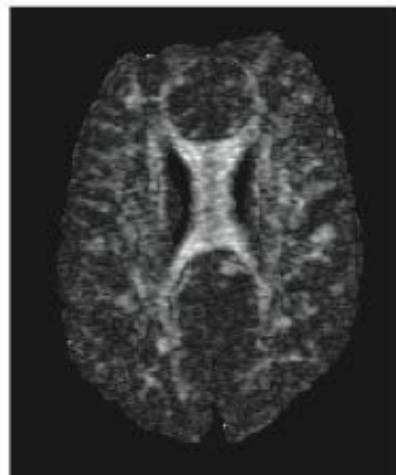
$$c_l = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3}$$

Fractional anisotropy $FA = \sqrt{\frac{3}{2} \frac{\sqrt{\sum_{i=1}^3 (\lambda_i - \mu)^2}}{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}}$

where $\mu = \frac{1}{3}(\lambda_1 + \lambda_2 + \lambda_3)$

Relative anisotropy $RA = \sqrt{\frac{3}{2} \frac{\sqrt{\sum_{i=1}^3 (\lambda_i - \mu)^2}}{\lambda_1 + \lambda_2 + \lambda_3}}$

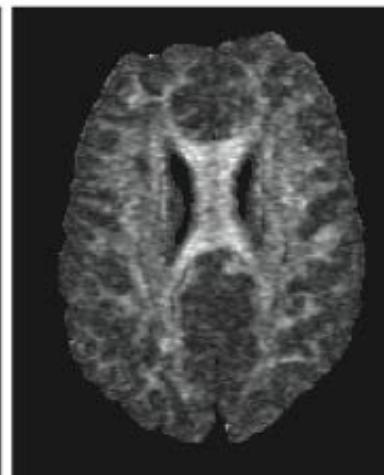
All above measures estimate how much 'fiber-like' is the current point



(a) c_l linear estimator



(b) fractional anisotropy



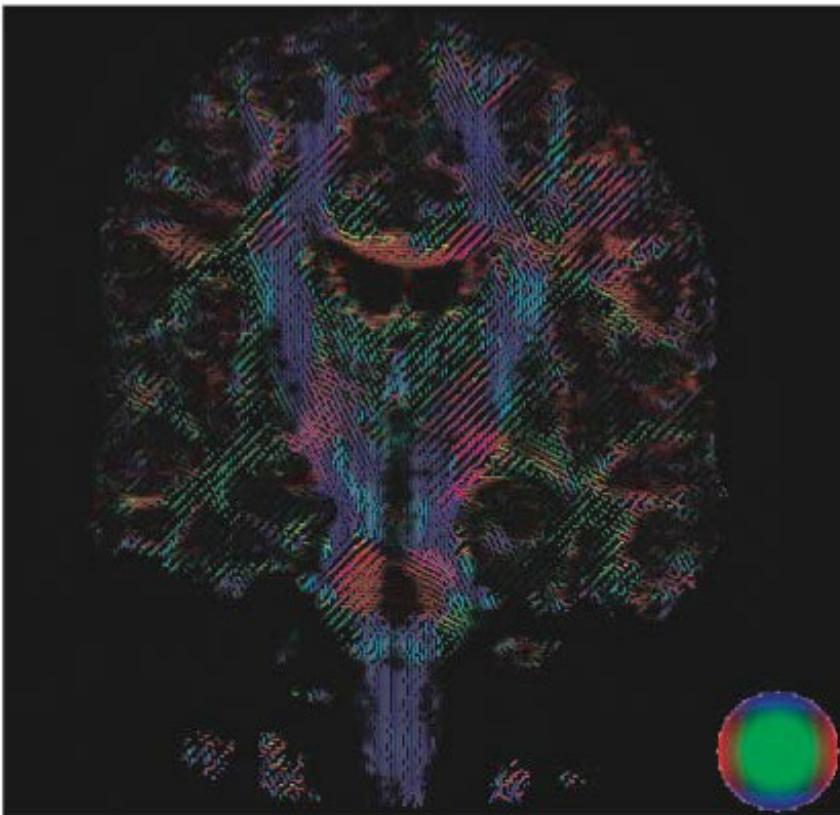
(c) relative anisotropy

white: strong fibers

Principal Component Analysis

Exploit the directional information in the eigenvectors

- major eigenvector e_1 : along the **strongest** diffusion direction
- for DTI tensors, it thus indicates fiber directions



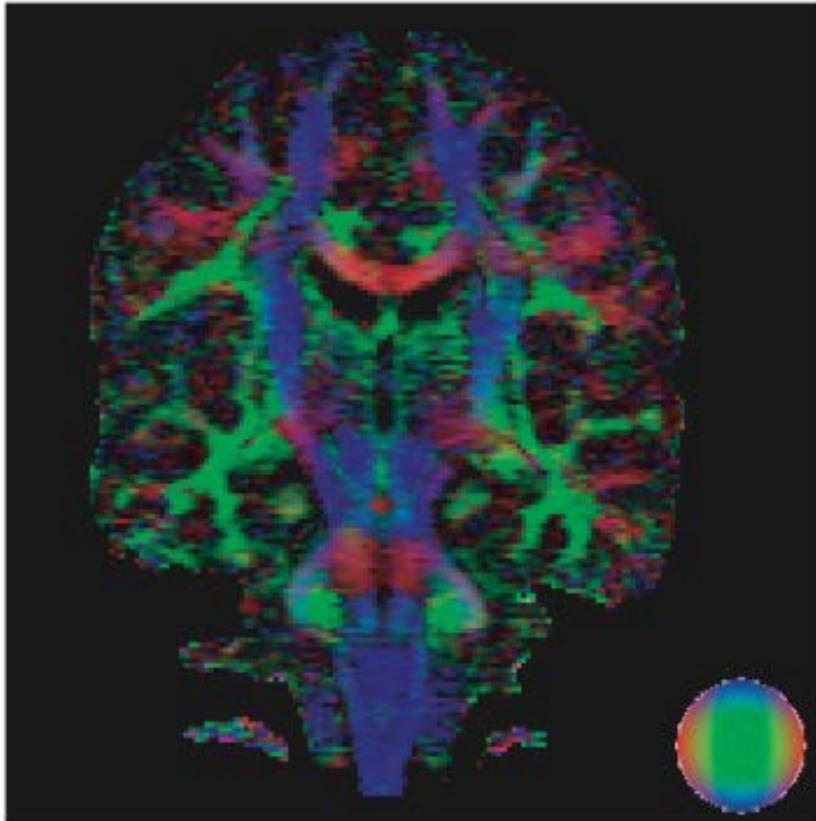
Directional color coding

- like for vectors (see Module 4)
- use simple colormap

$$\begin{aligned} R &= |e_1 \cdot x|, \\ G &= |e_1 \cdot y|, \\ B &= |e_1 \cdot z|. \end{aligned}$$

- use vector glyphs / hedgehogs
- seed only points where c_l, FA or RA are large enough
(other points don't cover fibers)
- OK, but takes training to grasp

Vector PCA



- Directional color coding
 - like before, but simply color points by direction
 - no glyphs drawn
 - no occlusion/clutter
 - direction coded only by color – less intuitive images

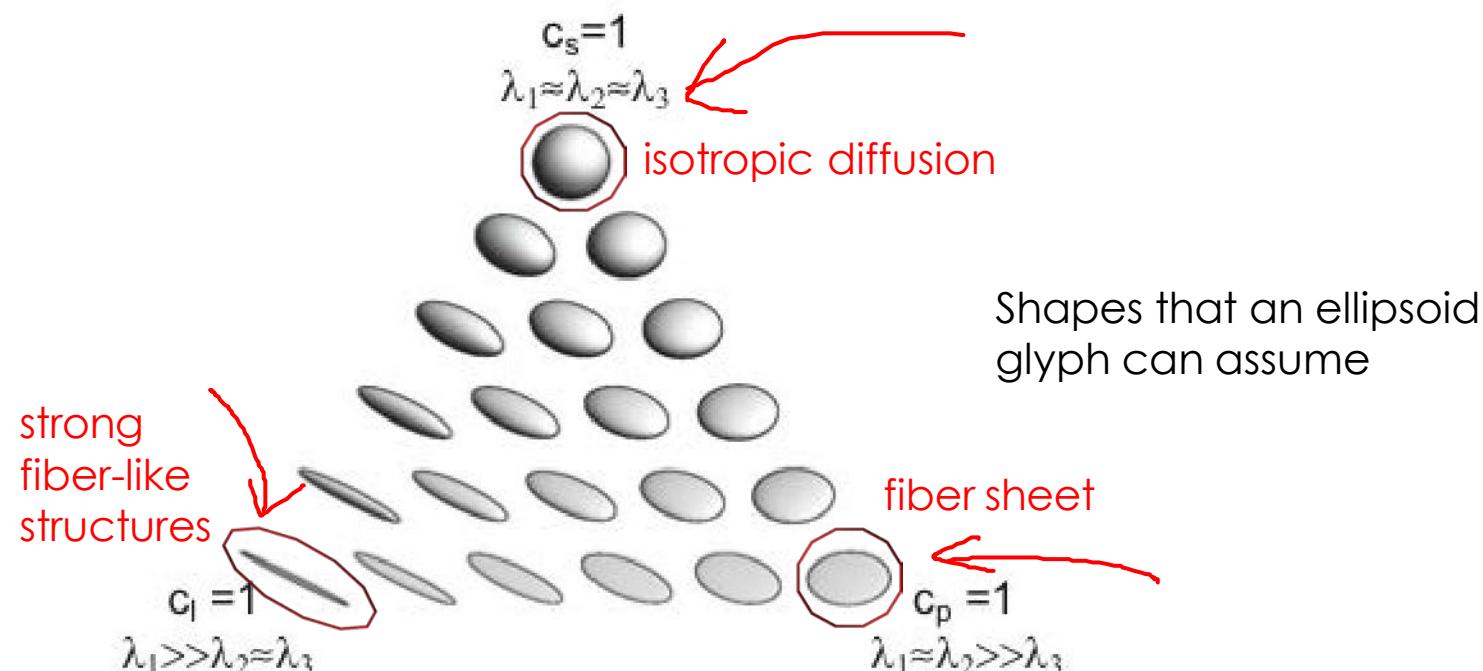
Tensor Glyphs

So far, we only visualized the major eigenvector e_1

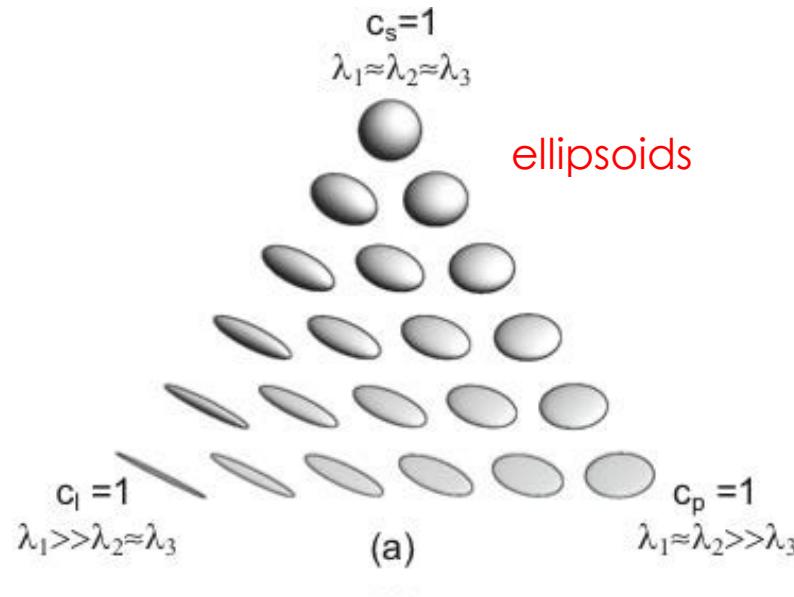
- so we reduced a tensor field to a vector field
- we **threw away** existing information (medium+minor eigenvectors e_2, e_3)

Ellipsoid glyph: Use all eigenvalues + eigenvectors

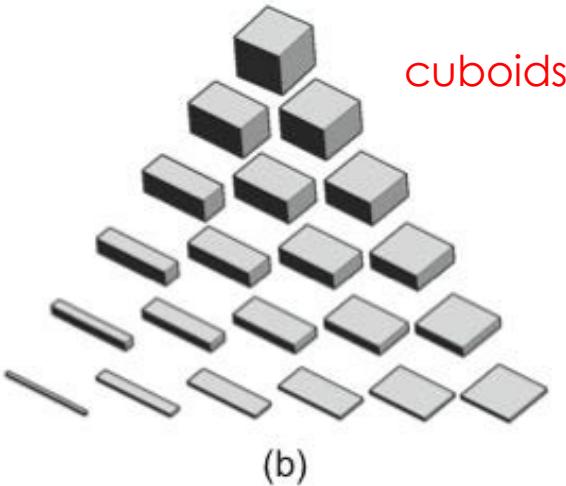
- orient glyph along eigensystem (e_1, e_2, e_3)
- scale it by eigenvalues ($\lambda_1, \lambda_2, \lambda_3$)



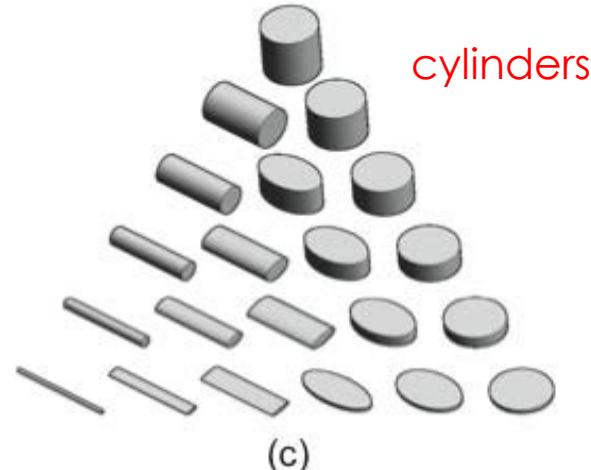
Tensor Glyphs



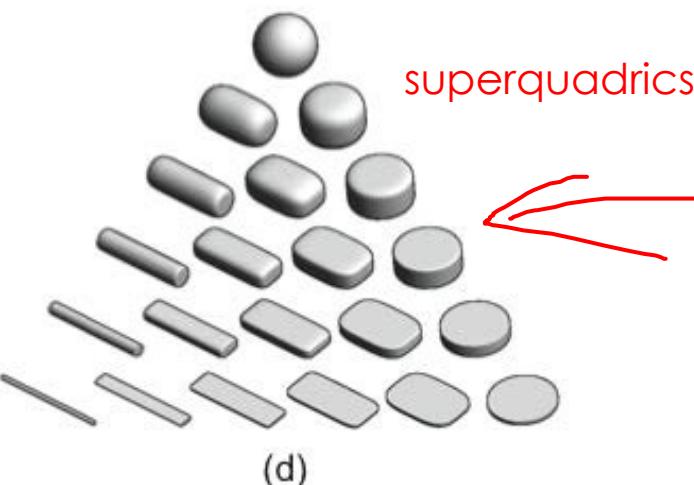
ellipsoids



cuboids



cylinders

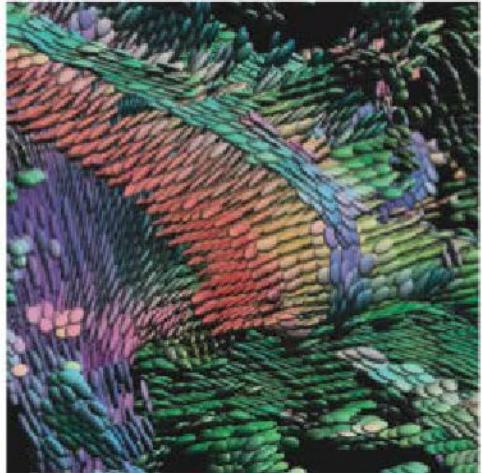


superquadrics

Can use other glyph shapes besides ellipsoids

Tensor Glyphs

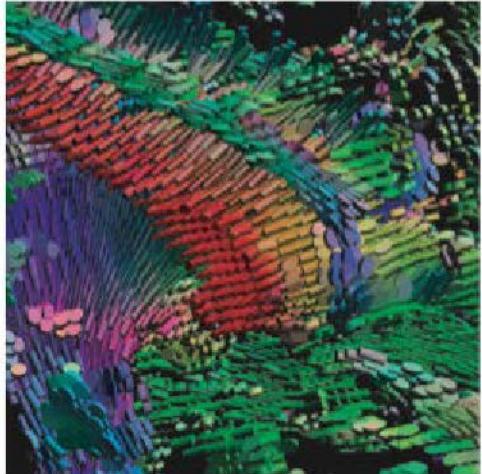
Zoom-in on brain DT-MRI dataset



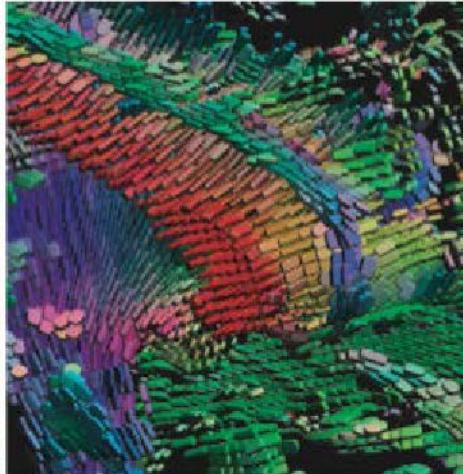
(a)



(b)



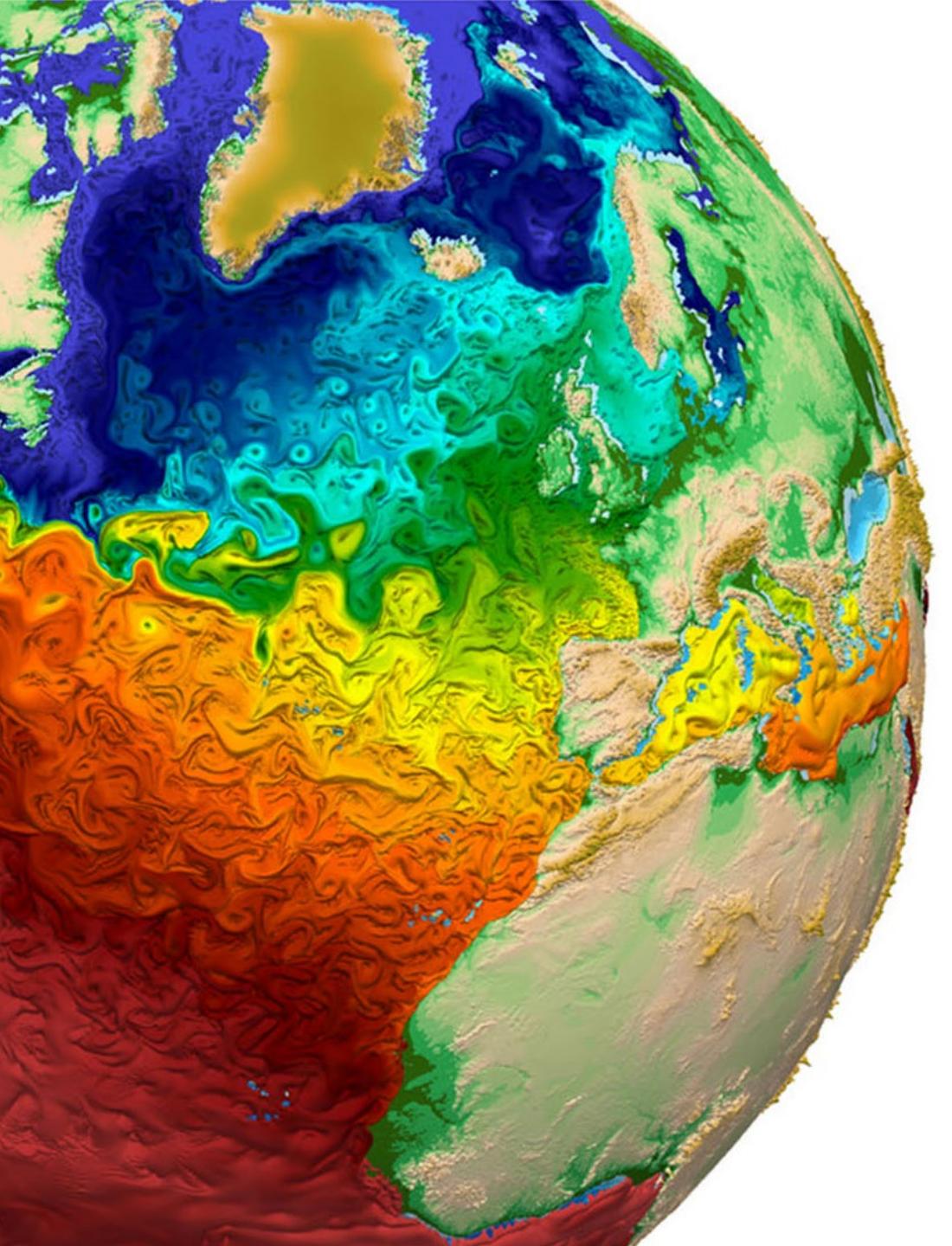
(c)



(d)

- a) ellipsoids
- b) cuboids
- c) cylinders
- d) superquadrics

Superquadrics look arguably most 'natural'

A 3D rendering of a human brain hemisphere, showing a complex network of colored fibers. The fibers are primarily colored in shades of red, orange, yellow, green, blue, and purple, representing different neural pathways. The brain is set against a white background.

Fiber Tracking

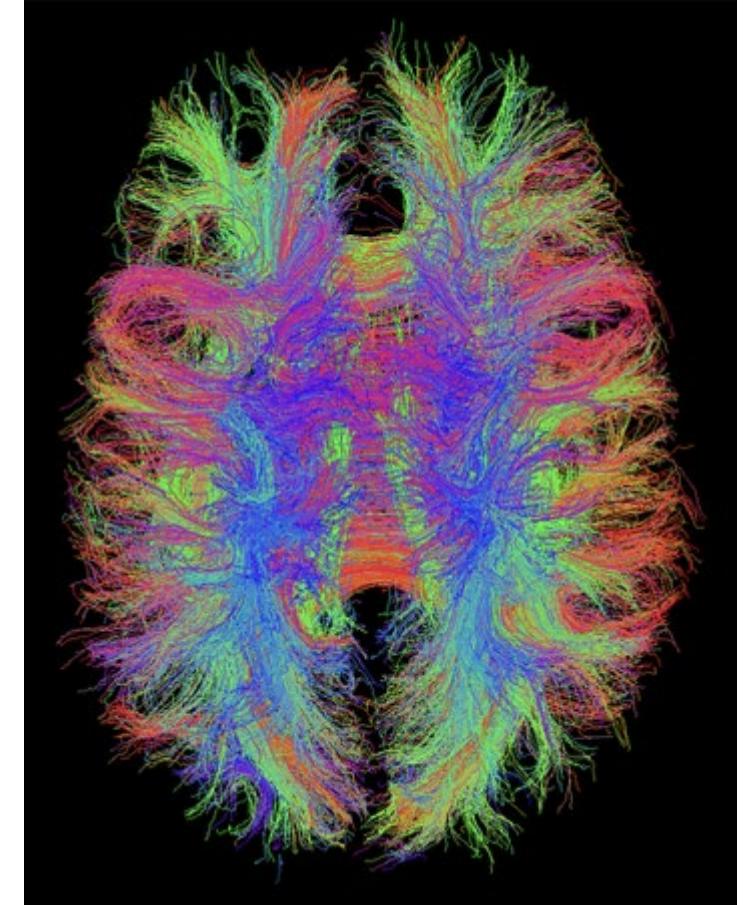
Scientific Visualization
Professor Eric Shaffer

Tractography: Fiber Tracking

“Recently, attention has been given to the visualization of 2D and 3D diffusion tensor fields from DT-MRI data. Although these methods provide nice visual cues, they do not attempt to recover the underlying anatomical structures, which are the white matter fiber tracts (bundles of axons) found within the brain.”

Oriented tensor reconstruction: tracing neural pathways from diffusion tensor MRI

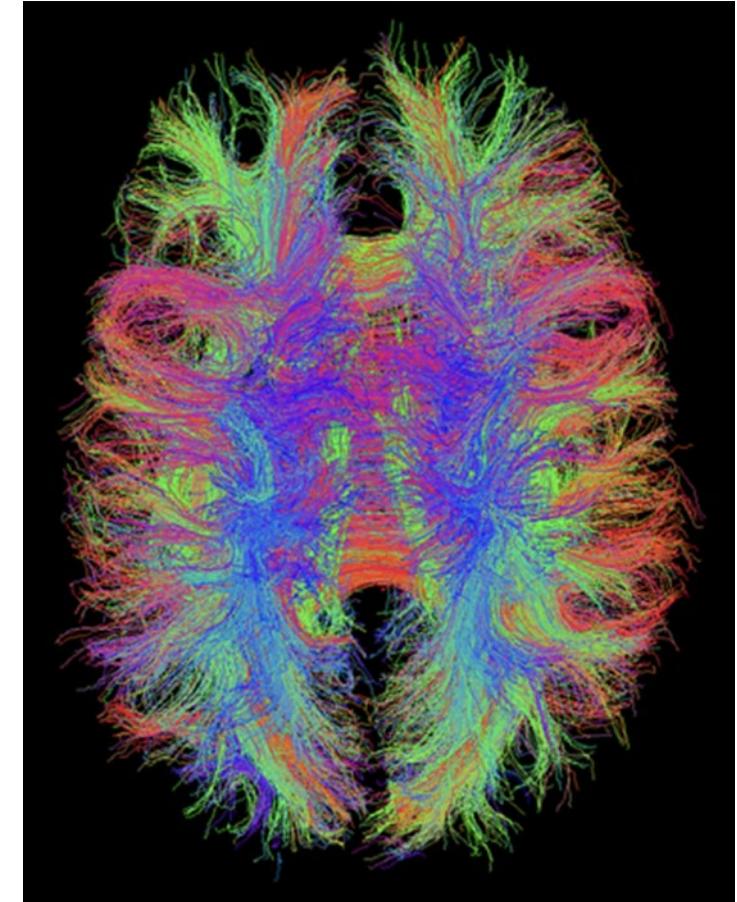
L. Zhukov, A. H. Barr



Applications of Fiber Tracking

“To illustrate the potential for tractography to provide new information in clinical neuroscience we review recent studies in three broad areas: First, use of tractography for quantitative comparisons of specific white matter pathways in disease; second, evidence from tractography for the presence of qualitatively different pathways in congenital disorders or following recovery; third, use of tractography to gain insights into normal brain anatomy that can aid our understanding of the consequences of localised pathology, or guide interventions.”

Just pretty pictures? What diffusion tractography can add in clinical neuroscience. Johansen-Berg, H., & Behrens, T. E. (2006).



DT-MRI Basics

Neural fibers are comprised mostly of bundles of long cylindrical cells
Filled with fluid and are bounded by less-water-permeable cell membranes

The diffusion properties can be represented with a symmetric second order tensor

$$\mathbf{T} = \begin{pmatrix} T^{xx} & T^{xy} & T^{xz} \\ T^{yx} & T^{yy} & T^{yz} \\ T^{zx} & T^{zy} & T^{zz} \end{pmatrix}$$

The 6 independent values (the tensor is symmetric)
The tensor elements vary continuously with spatial location

Principal Component Analysis

$$\mathbf{T} = \begin{pmatrix} T^{xx} & T^{xy} & T^{xz} \\ T^{yx} & T^{yy} & T^{yz} \\ T^{zx} & T^{zy} & T^{zz} \end{pmatrix}$$

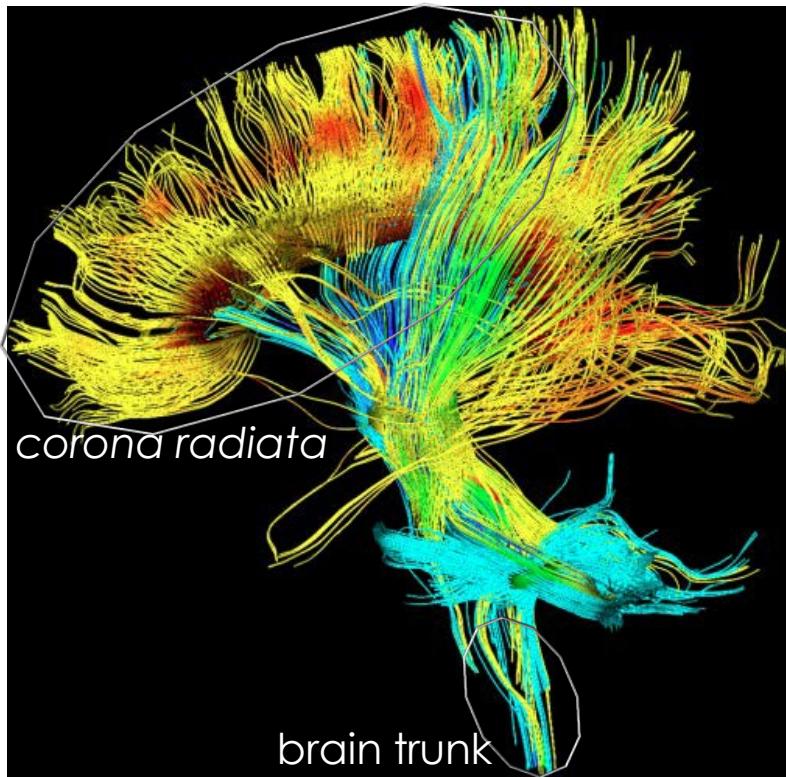
$$\underline{\mathbf{T}\mathbf{e}_i} = \underline{\lambda_i} \mathbf{e}_i$$

The 3-D local axis direction of the neuron fibers will be the dominant eigenvector of the tensor
There should be one large eigenvalue, and two small eigenvalues

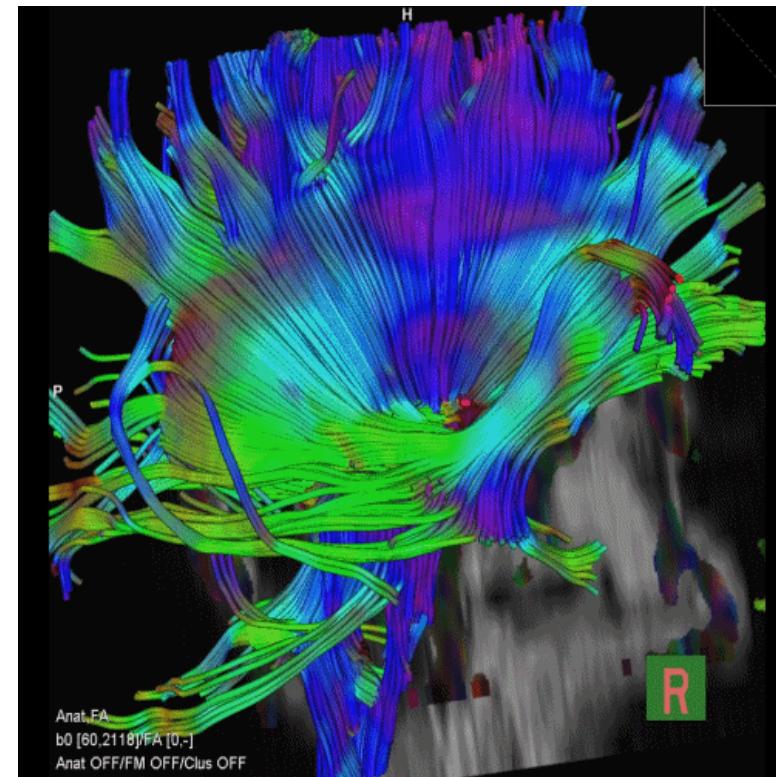
$$\mathbf{T} = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\} \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}^T$$

Fiber Tracking

- consider major eigenvector field
- trace streamlines
 - **seed:** in regions with high anisotropy (i.e. where fibers are)
 - **stop:** when anisotropy gets too low (i.e. when we leave fibers)



streamlines, brain overview



stream tubes, brain detail

Seeding

Choose to seed points exhibiting strong anisotropy

These likely lie on fibers

Measure

$$c_\ell = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3}$$

Values close to 1 indicate strong linear diffusion $\lambda_1 \gg \lambda_2 \approx \lambda_3$

Tensor Interpolation

Reconstruct the continuous tensor field using linear interpolation

Value of a tensor inside a voxel is a linear combination of 8 corner values

Interpolation is tri-linear

Interpolation is component-wise

$$\begin{aligned}\mathbf{T}(x, y, z) = & \mathbf{T}_{ijk} (1 - x)(1 - y)(1 - z) + \\ & + \mathbf{T}_{i+1,jk} x(1 - y)(1 - z) + \mathbf{T}_{i,j+1,k} (1 - x)y(1 - z) \\ & + \mathbf{T}_{ij,k+1} (1 - x)(1 - y)z + \mathbf{T}_{i+1,j,k+1} x(1 - y)z \\ & + \mathbf{T}_{i,j+1,k+1} (1 - x)yz + \mathbf{T}_{i+1,j+1,k} xy(1 - z) \\ & + \mathbf{T}_{i+1,j+1,k+1} xyz\end{aligned}$$

For general position
replace x with $\frac{x - x_{min}}{x_{max} - x_{min}}$
Etc.

Tensor Interpolation

We can use trilinear component-wise interpolation because:

Any linear combination of symmetric tensors remains a symmetric tensor

Component-wise interpolation of eigenvectors would not lead to correct results:
A linear interpolation between two unit vectors is not a unit vector anymore

$$\begin{aligned}\mathbf{T}(x, y, z) = & \mathbf{T}_{ijk} (1-x)(1-y)(1-z) + \\ & + \mathbf{T}_{i+1,jk} x(1-y)(1-z) + \mathbf{T}_{i,j+1,k} (1-x)y(1-z) \\ & + \mathbf{T}_{ij,k+1} (1-x)(1-y)z + \mathbf{T}_{i+1,j,k+1} x(1-y)z \\ & + \mathbf{T}_{i,j+1,k+1} (1-x)yz + \mathbf{T}_{i+1,j+1,k} xy(1-z) \\ & + \mathbf{T}_{i+1,j+1,k+1} xyz\end{aligned}$$

Moving Least Squares (MLS)

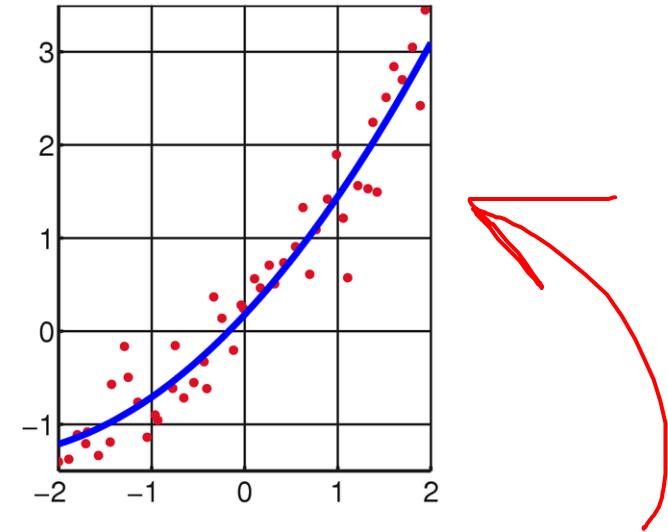
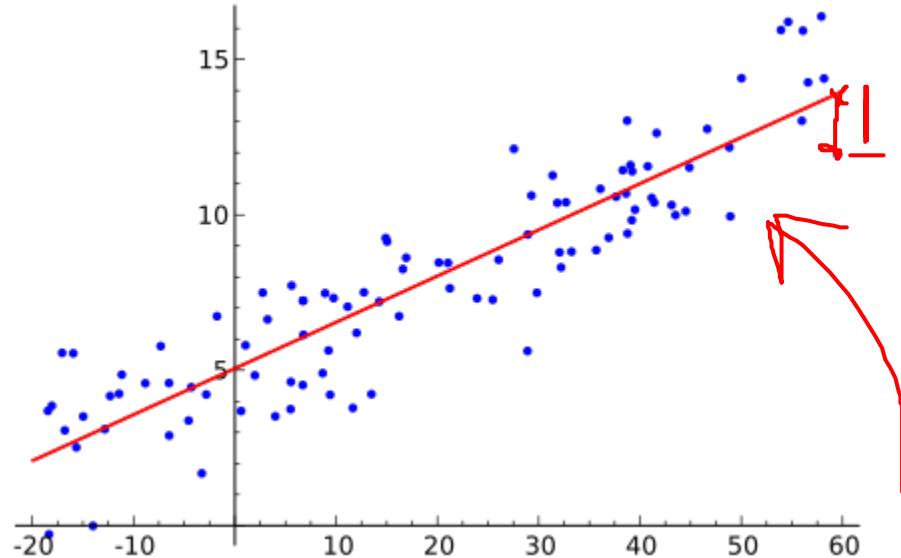
Experimental data is noisy

- Needs to be filtered for best results
- Gaussian filter will not work well, will blur the directional information

MLS - find a low degree polynomial:

- which best fits the data, in the least squares sense
- in the small region around the point of interest
- replace the data value at that point value of the polynomial at that point
 - Here data = tensor

Least Squares Best Fit



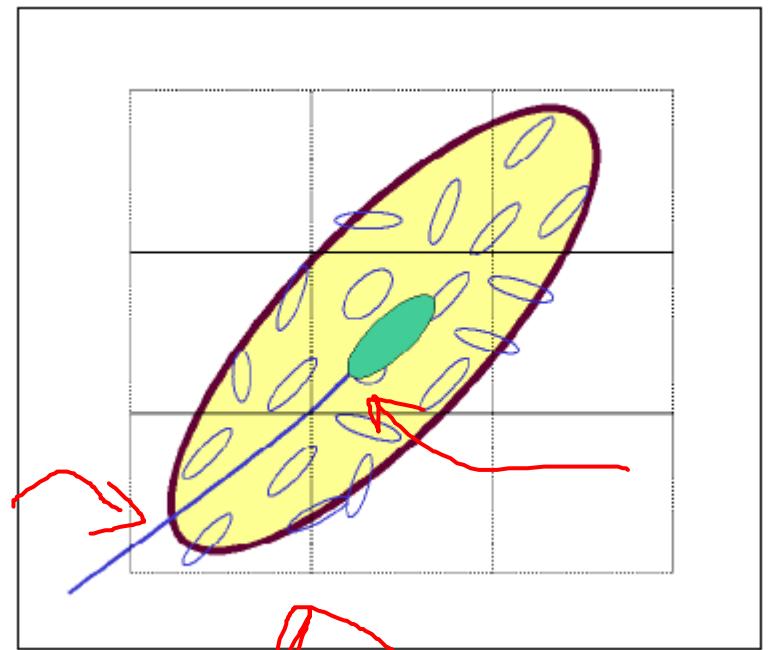
Least squares best fit for data (x_i, y_i)

- Finds coefficients of a polynomial $F(X)$
- Minimizes sum of the squared error between $F(x_i)$ and y_i
- Moving Least Squares considers a subset of the data in a sliding window

Algorithm

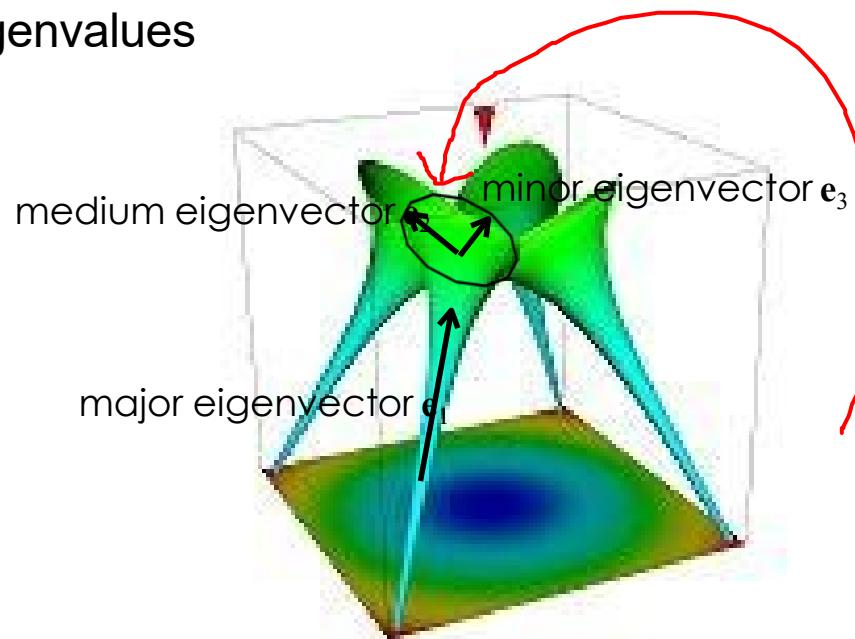
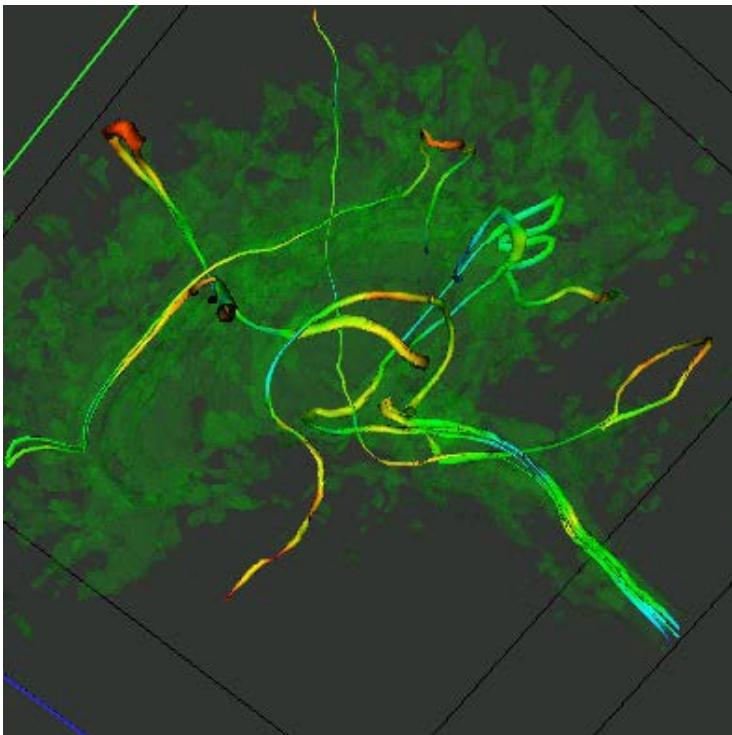
```
1. User inputs starting region  
2. For every starting point P in the region  
{  
    Tp = filter(T, P, sphere) ;  
    cl = anisotropy(Tp) ;  
    if (cl > eps){  
        e1 = direction(Tp) ;  
        trace1 = fibertrace(P, e1) ;  
        trace2 = fibertrace(P, -e1) ;  
        trace =| trace1 + trace2 ;  
    }  
}
```

```
trace = fibertrace(P, e){  
    trace->add(P) ;  
    do {  
        Pn = integrate_forward(P, e1, dt) ;  
        Tp = filter(T, Pn, ellipsoid, e1) ;  
        cl = anisotropy(Tp)  
        if (cl>eps){  
            trace->add(Pn) ;  
            P = Pn;  
            e1 = direction(Tp) ;  
        }  
    } while (cl >eps)  
    return(trace) ;  
}
```



Hyperstreamlines

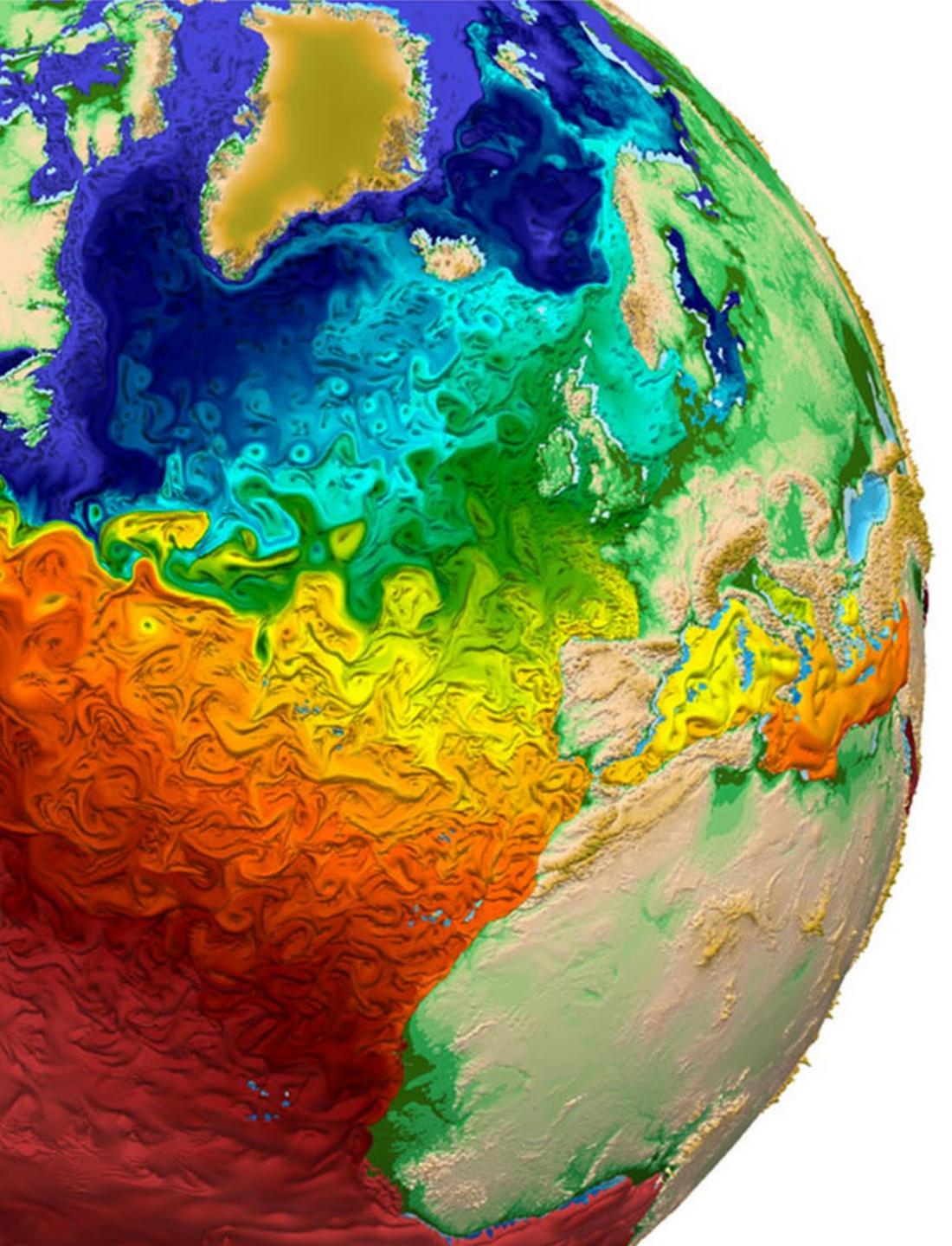
- trace stream tubes in major eigenvector field (like so far)
- use an **elliptic** cross-section
 - oriented along medium + minor eigenvectors
 - scaled with medium + minor eigenvalues



Tube cross-section shows diffusion across fibers

- Thin, round tubes: we're in a fiber **bundle**
- Thick, flat tubes: we're in a fiber **sheet**
- Thick, round tubes: we're **exiting** a fiber

Tube color is often mapped to direction...giving viewers another cue to help understand the fiber path through 3D space.



Visualizing Trees

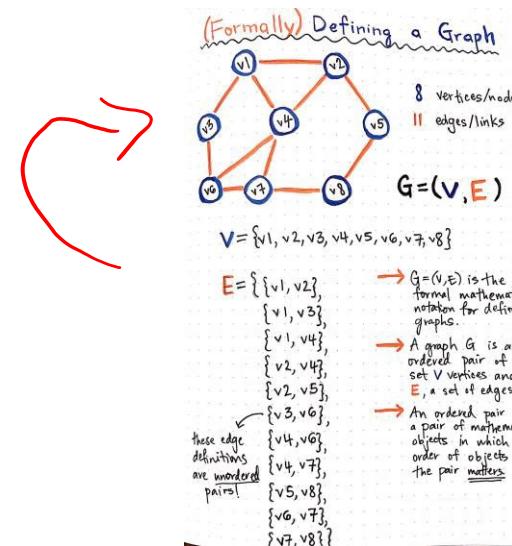
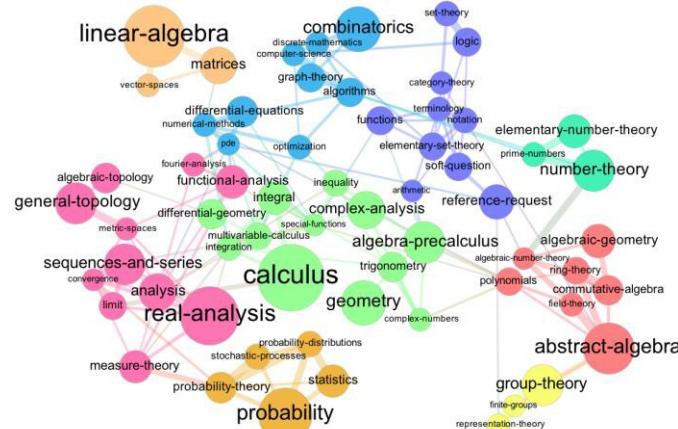
Scientific Visualization
Professor Eric Shaffer

Network Visualization = Graph Visualization

Graph drawing is an area of mathematics and computer science combining methods from geometric graph theory and information visualization to derive two-dimensional depictions of graphs arising from applications such as social network analysis, cartography, linguistics, and bioinformatics.^[1]

A drawing of a graph or **network diagram** is a pictorial representation of the vertices and edges of a graph.

- Wikipedia



Node = vertex
link = edge

Network Visualization

Arrange networks and trees

→ Node–Link Diagrams

Connection Marks

✓ NETWORKS

✓ TREES

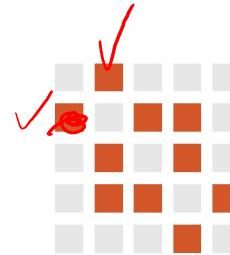


→ Adjacency Matrix

Derived Table

✓ NETWORKS

✓ TREES



→ Enclosure

Containment Marks

✗ NETWORKS

✓ TREES



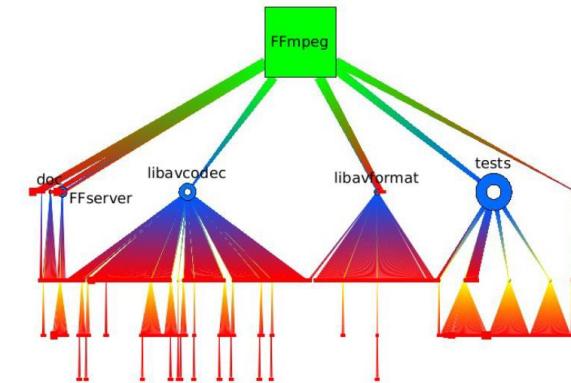
Trees

Trees are acyclic graphs

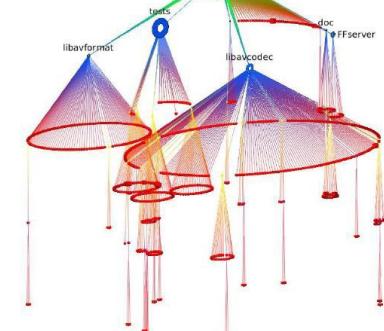
Examples

- icon size: folder size
- icon shape+color: level in tree

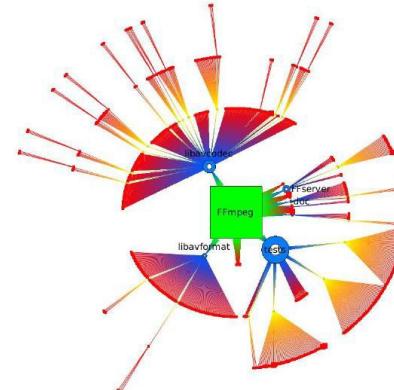
rooted tree



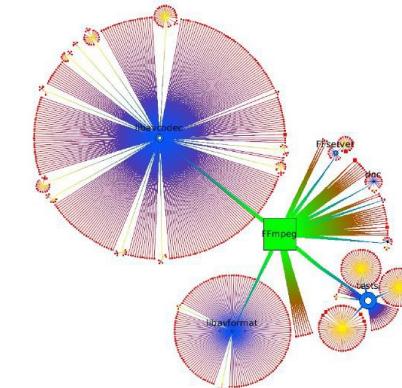
cone tree



ffmpeg video code C library: 785 files, 42 folders

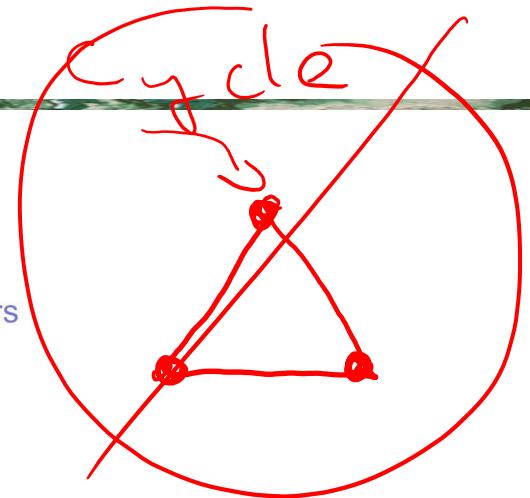


radial tree

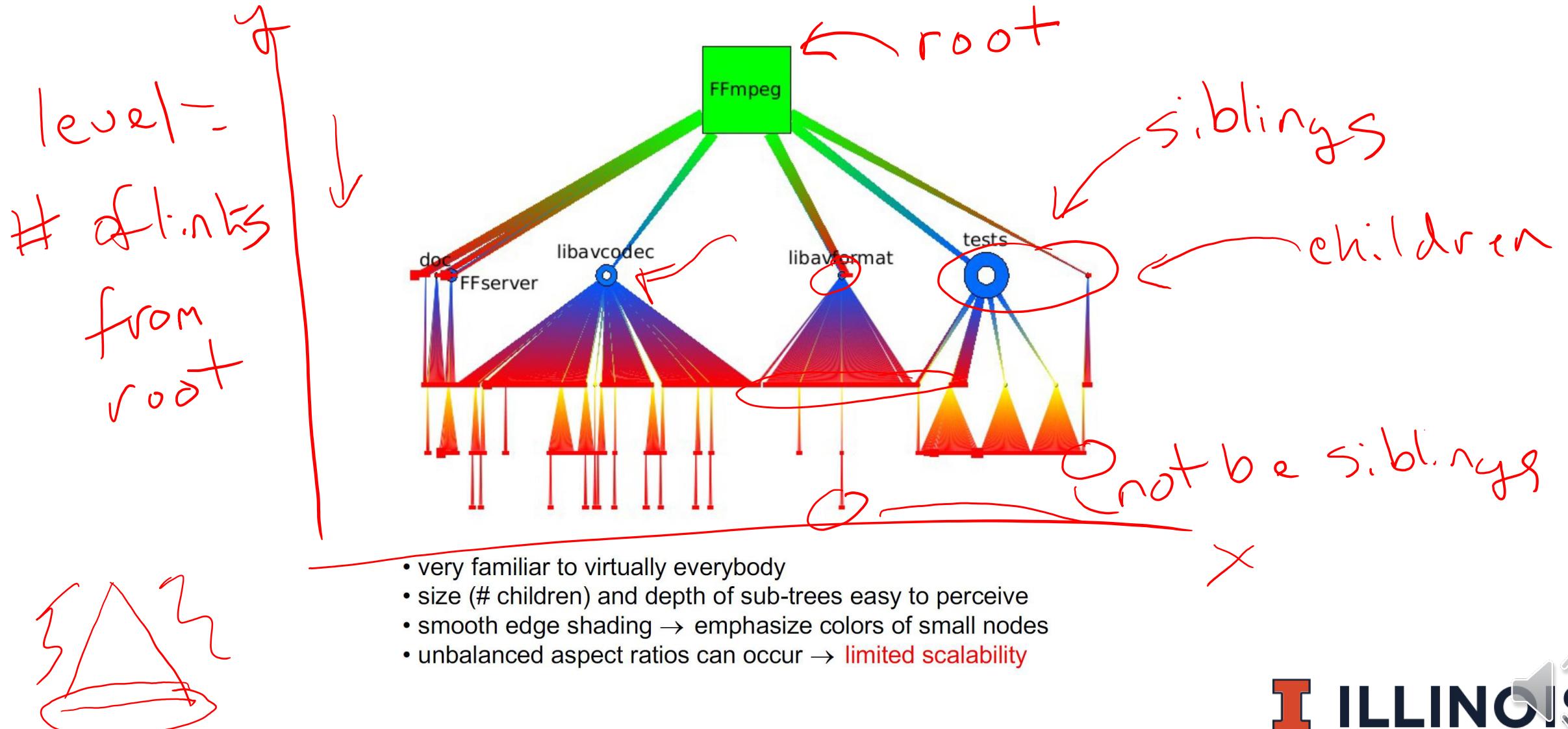


bubble tree

- root
- level 2
- level 3
- others



Tree Layout: Rooted Layout



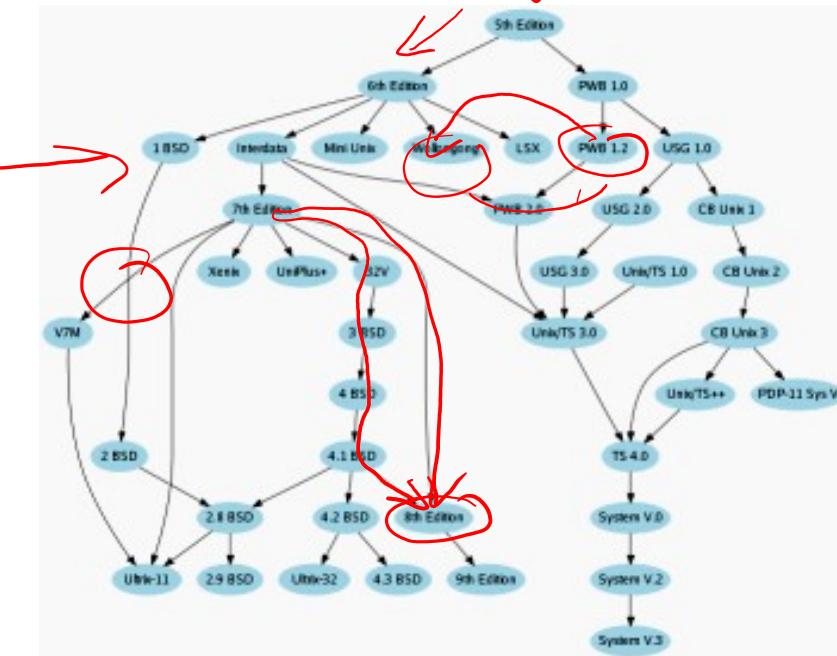
Directed Acyclic Graphs (DAGs)

- class hierarchies (multiple inheritance)
- organization structure (multiple bosses)
- also created from general graphs by removing cycles

Hierarchical layout [Sugiyama et al '81]

Algorithm:

- swap edges to eliminate cycles and get a directed (rooted) graph
- for every level, starting from root:
 - assign y coordinate as function of level
 - permute nodes on level to minimize edge lengths/crossings
- edges drawn as curves (splines) to minimize crossings



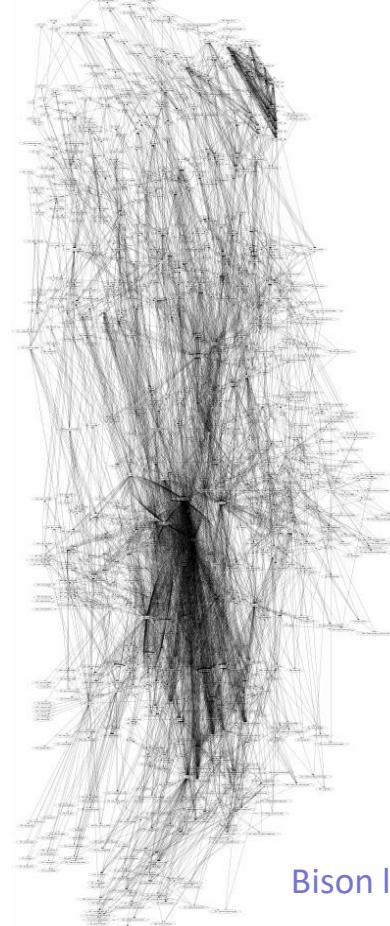
UNIX system history drawn with
the GraphViz package
(www.graphviz.org)

DAG Layout

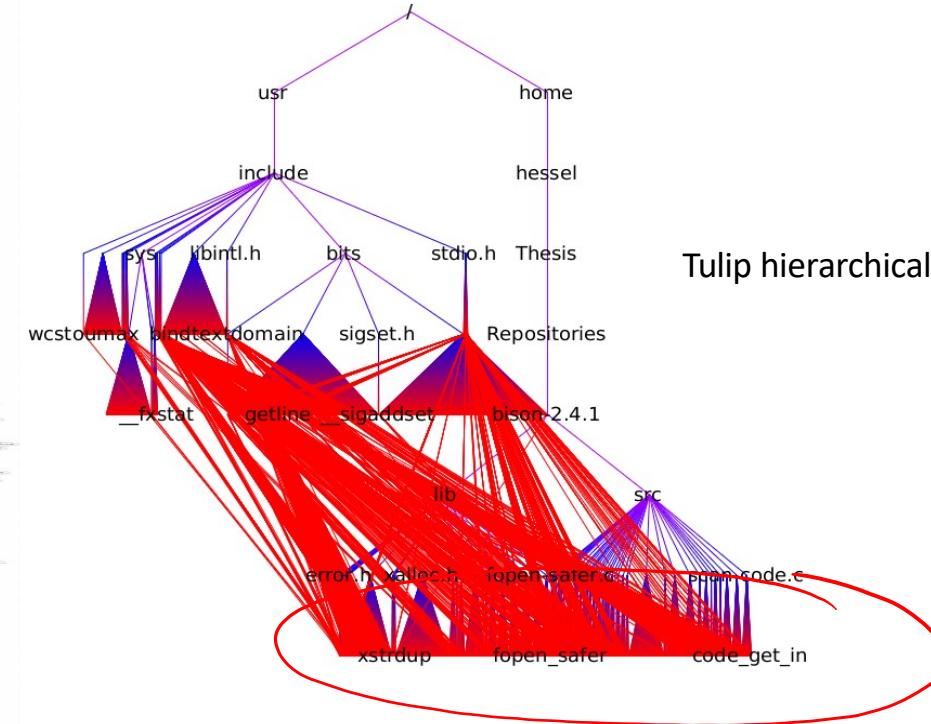
Hierarchical layout scalability

- OK for < 1000 nodes or edges
- too many crossings and/or bad aspect ratios for large graphs
- we shall see later how to handle large graphs

Hierarchical DAG Layout



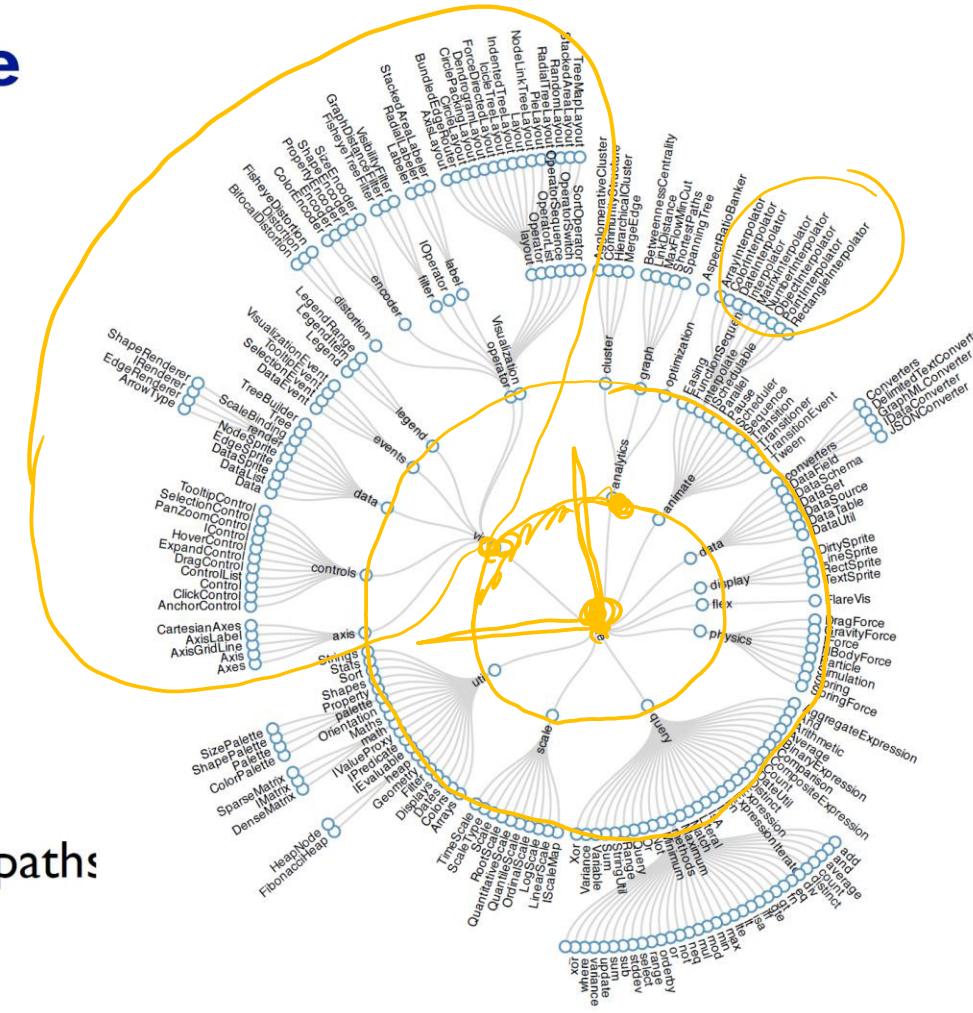
Bison library call graph (3.214 nodes, 14.382 edges)



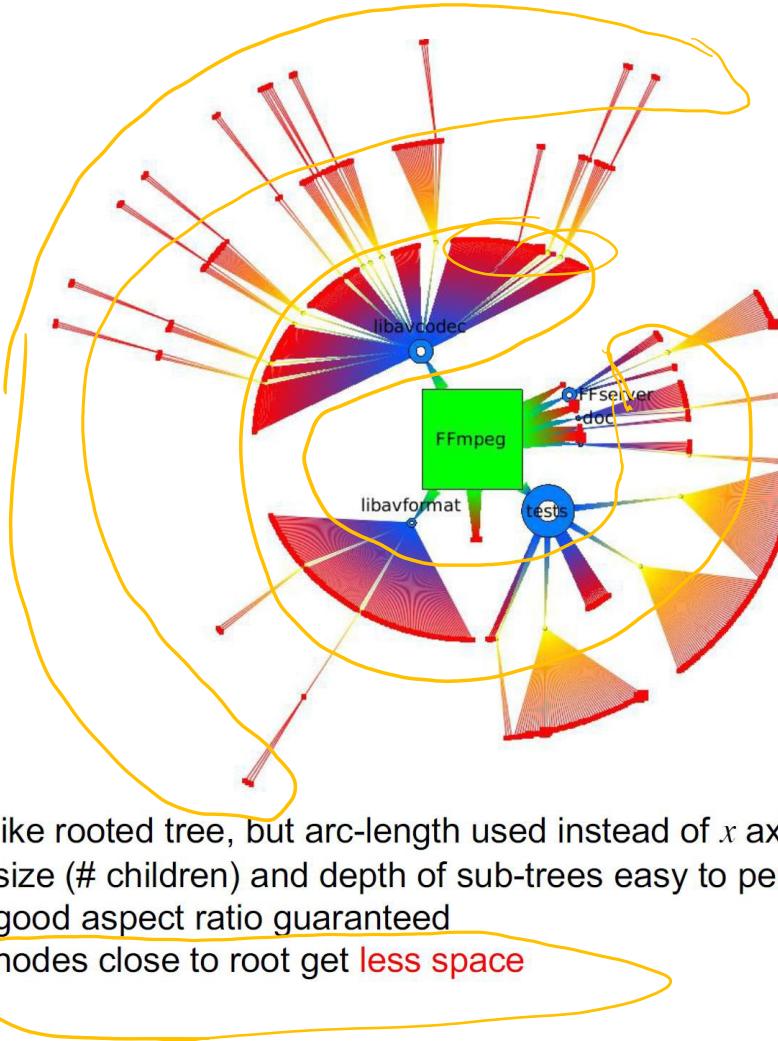
Radial Tree Layout

Idiom: radial node-link tree

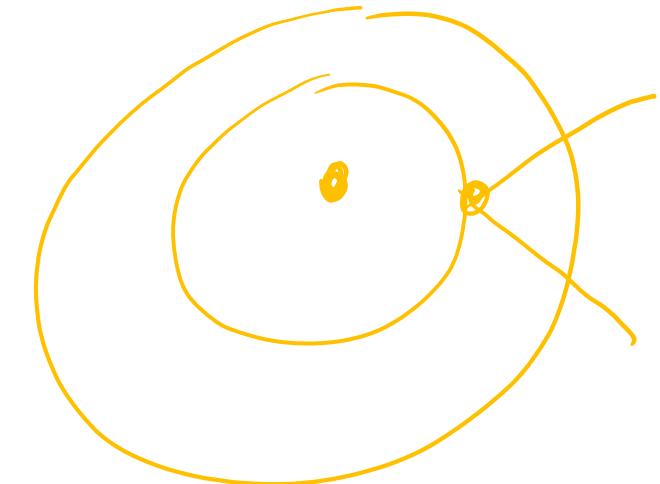
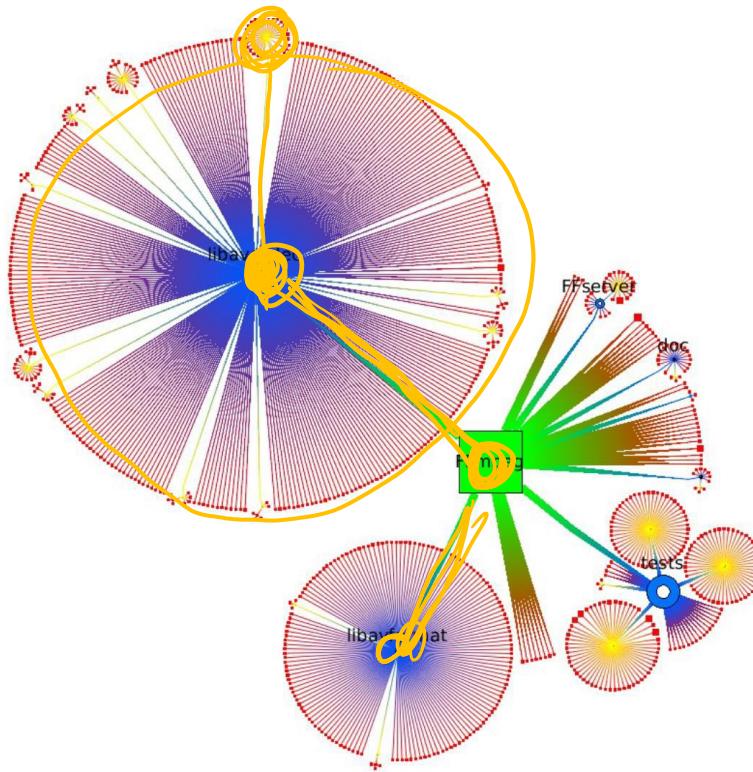
- data
 - tree
 - encoding
 - link connection marks
 - point node marks
 - radial axis orientation
 - angular proximity: siblings
 - distance from center: depth in tree
 - tasks
 - understanding topology, following paths
 - scalability
 - 1K - 10K nodes



Tree Layout: Radial Layout

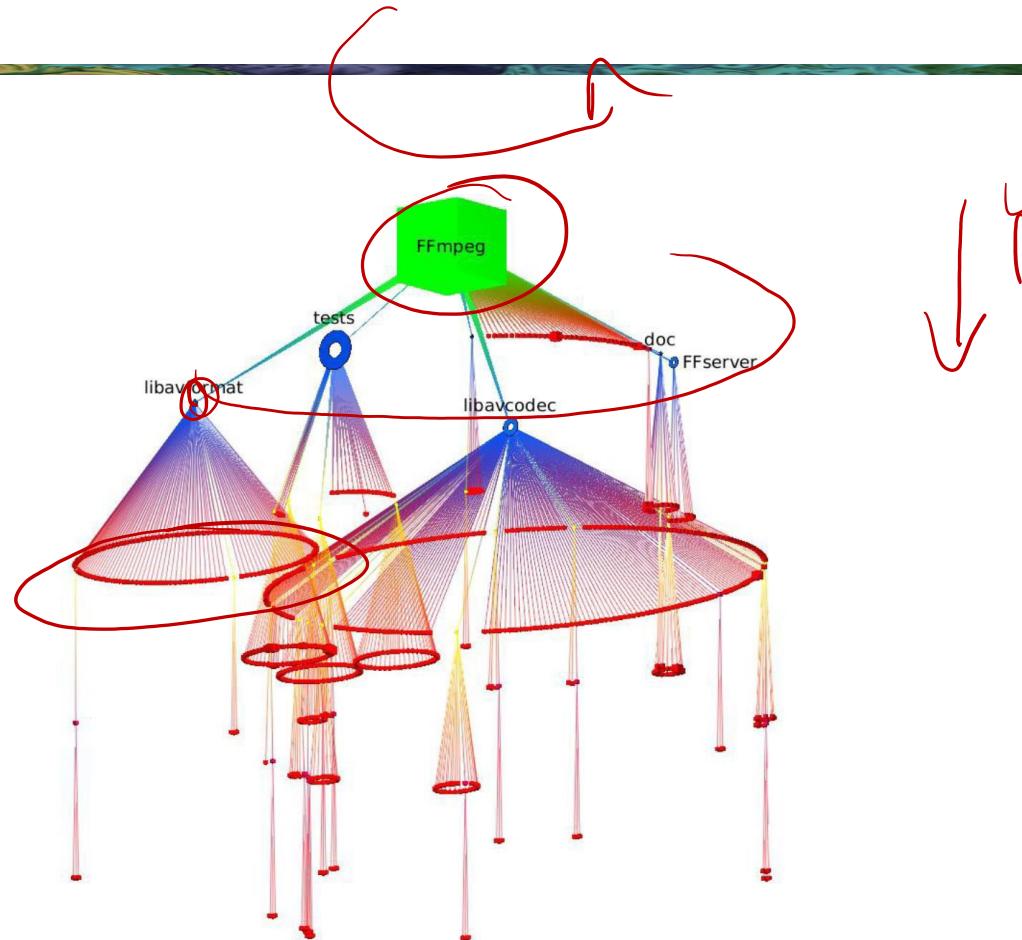


Tree Layout: Bubble Layout

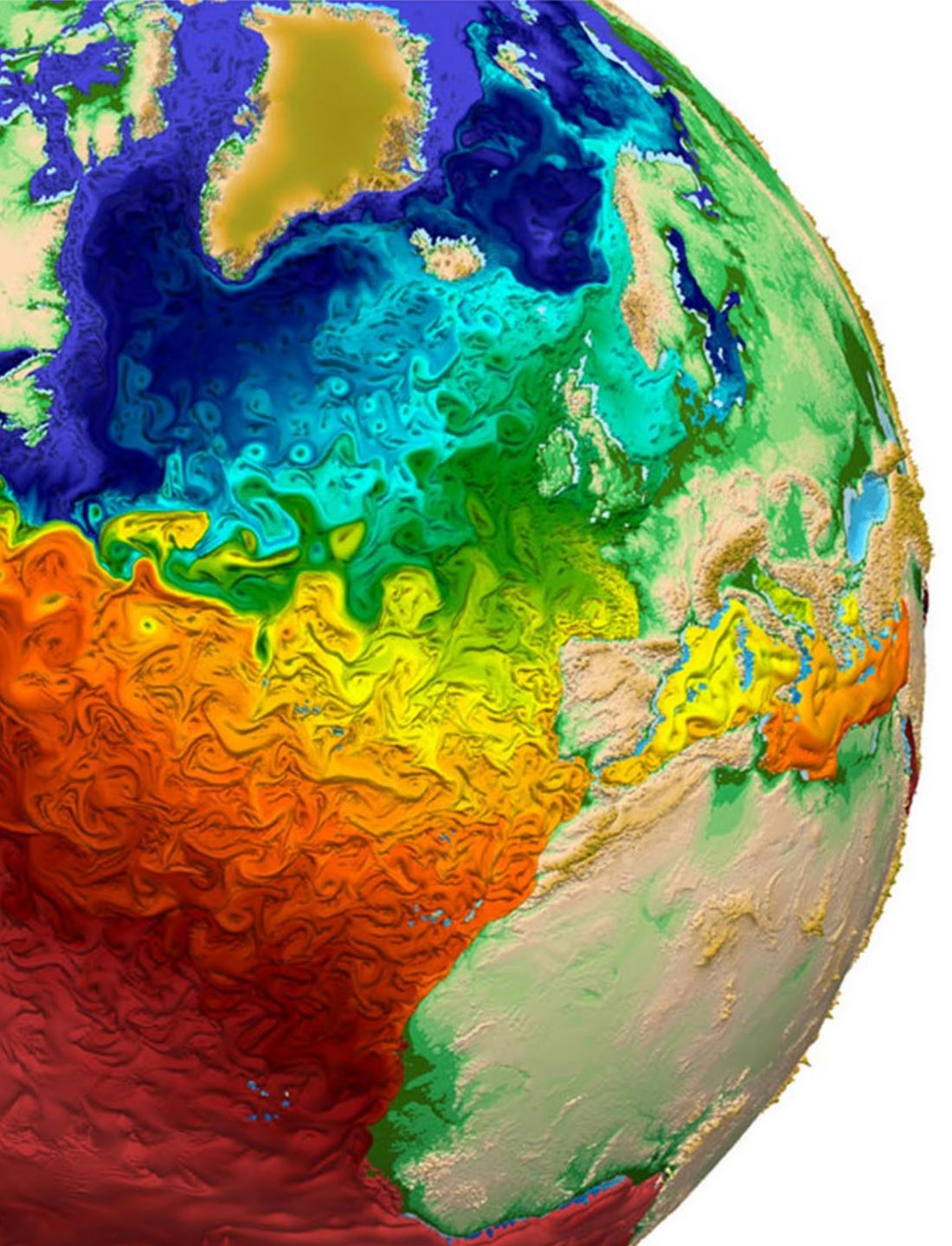


- a subtree gets a full circle instead of a circle sector
- better spreading of the nodes for large trees
- variable edge lengths
- hard to distinguish node depth in the tree

Cone Tree



- a subtree gets a full cone instead of a sector / circle / line
 - 3D effectively shows the tree depth
 - combines bubble tree (seen from above) with rooted tree (seen from profile)
- 3D is tricky:** occlusions, perspective shortening, navigation



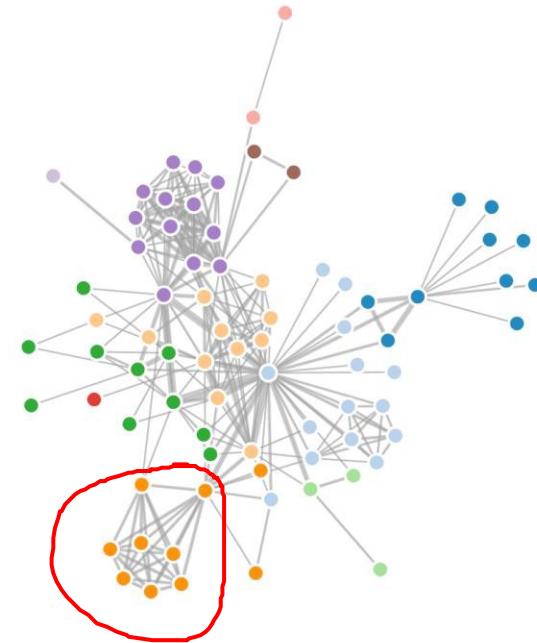
Forces-Directed Graph Layout

Scientific Visualization
Professor Eric Shaffer

Force-Directed Graph Layout

Idiom: **force-directed placement**

- visual encoding
 - link connection marks, node point marks
- considerations
 - spatial position: no meaning directly encoded
 - left free to minimize crossings
 - proximity semantics?
 - sometimes meaningful
 - sometimes arbitrary, artifact of layout algorithm
 - tension with length
 - long edges more visually salient than short
- tasks
 - explore topology; locate paths, clusters
- scalability
 - node/edge density $E < 4N$



<http://mbostock.github.com/d3/ex/force.html>

Force-directed Layout Intuition

Goals

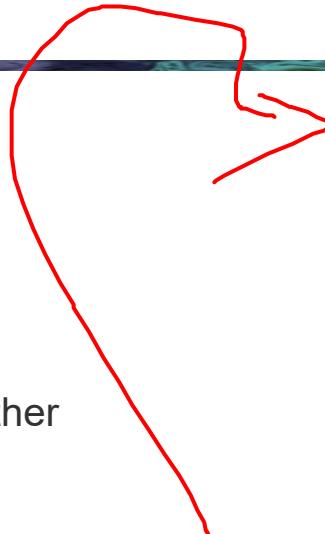
- Vertices well-distributed on the display
- Edges cross each other as little as possible.

Approach

- Simulate the graph as a physical system.
- Nodes are electrically charged particles and repulse each other
- Edges act as springs that attract connected nodes

Result

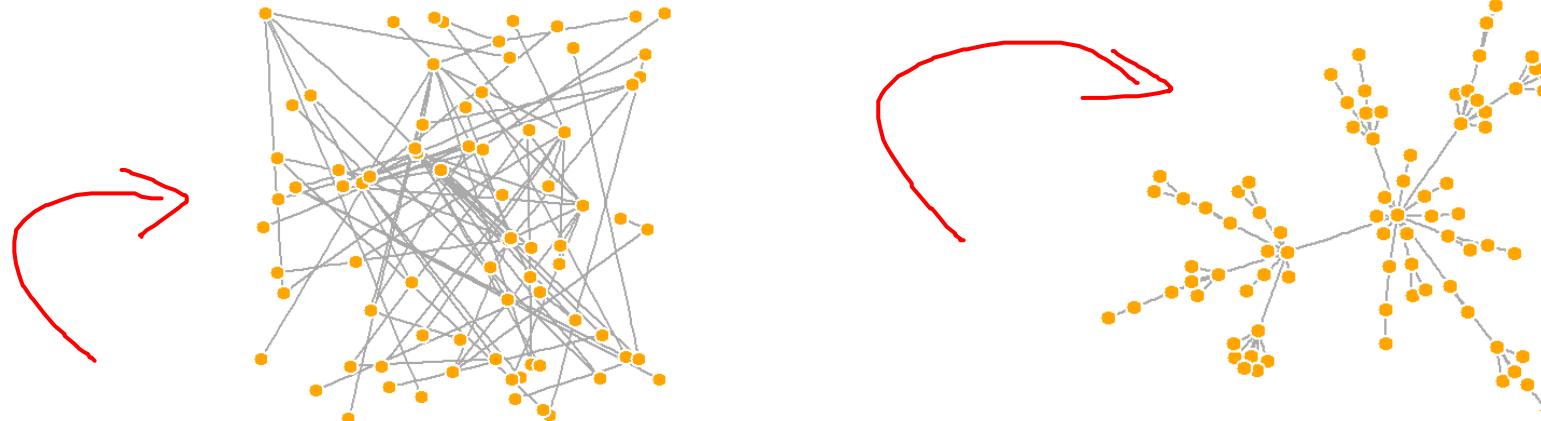
- Nodes are evenly distributed through the chart area
- Nodes which share more connections are closer to each other



"Fruchterman-Reingold is one of the most used force-directed layout algorithms out there." - The Internet

Graph Drawing by Force-Directed Placement Fruchterman, Thomas M. J.; Reingold, Edward M. (1991)

Developed at University of Illinois



Fruchterman-Reingold Details

$$\underline{F_a(n_i, n_j) = \frac{|p_i - p_j|^2}{k}}$$

$$\underline{F_r(n_i, n_j) = -\frac{k^2}{|p_i - p_j|}}$$

$$k = C \sqrt{\frac{\text{area}}{\text{number of vertices}}}$$

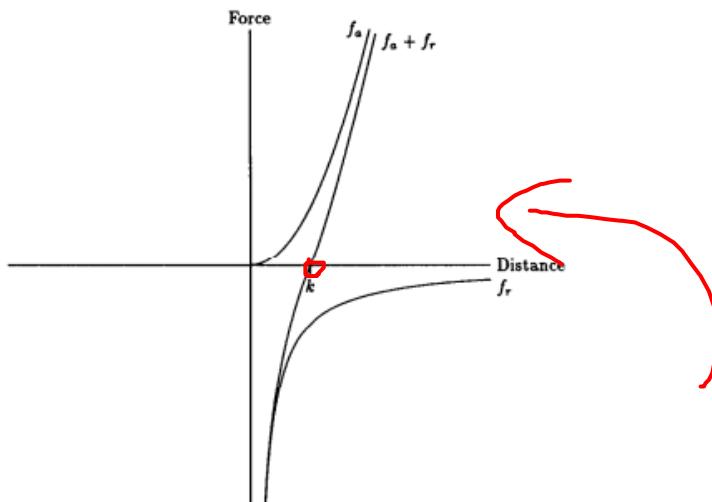
n_i is a vertex

p_i is the position of that vertex

C is a constant found experimentally to yield good result

k if vertices are uniformly distributed,
would be radius of empty circle around vertex

the distance at which the forces will balance

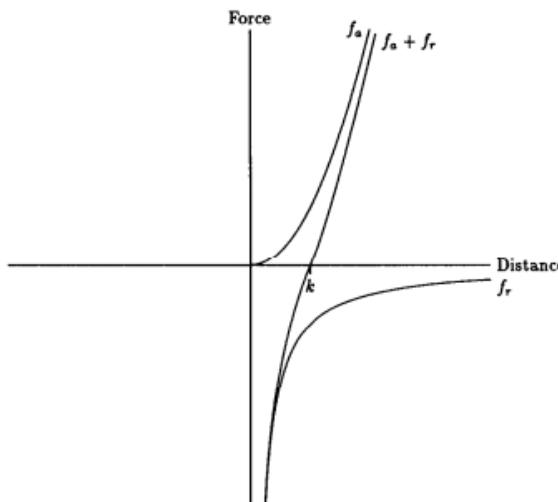


Fruchterman-Reingold Details

$$F_a(n_i, n_j) = \frac{|p_i - p_j|^2}{k}$$

$$F_r(n_i, n_j) = -\frac{k^2}{|p_i - p_j|}$$

$$k = C \sqrt{\frac{\text{area}}{\text{number of vertices}}}$$



For each vertex $\underline{n_i}$ calculate a sum of forces:

- $F_a(n_i, n_j)$ between n_i and all neighbors n_j
- $\underline{F_r(n_i, n_j)}$ between n_i and all vertices n_j
- Each force has a magnitude and direction $\overleftarrow{p_j - p_i}$
- For repulsion, direction is negated
- Move p_i accordingly

Do this until change in positions below a threshold
...or you hit your limit on the number of iterations

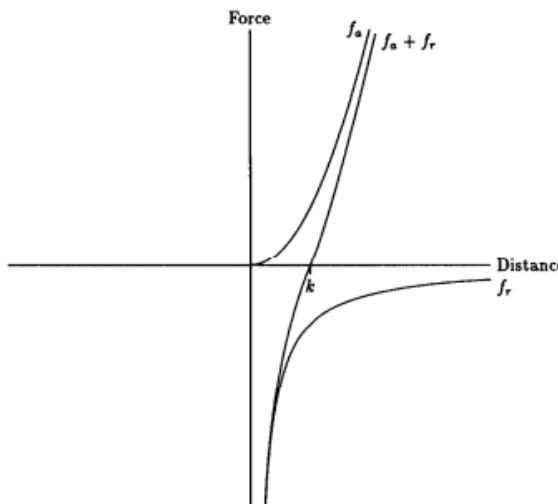
Fruchterman-Reingold...More Details

$$F_a(n_i, n_j) = \frac{|p_i - p_j|^2}{k}$$

$$F_r(n_i, n_j) = -\frac{k^2}{|p_i - p_j|}$$

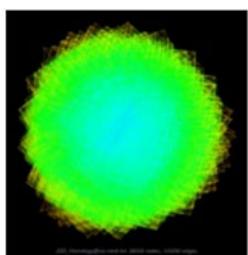
$$k = C \sqrt{\frac{\text{area}}{\text{number of vertices}}}$$

- Direction $\overrightarrow{p_j - p_i}$ should be unit length vector
- Each iteration, stop vertices from moving outside display
- Apply some scale factor to the amount of movement of each vertex
 - Reduce this scale factor each iteration

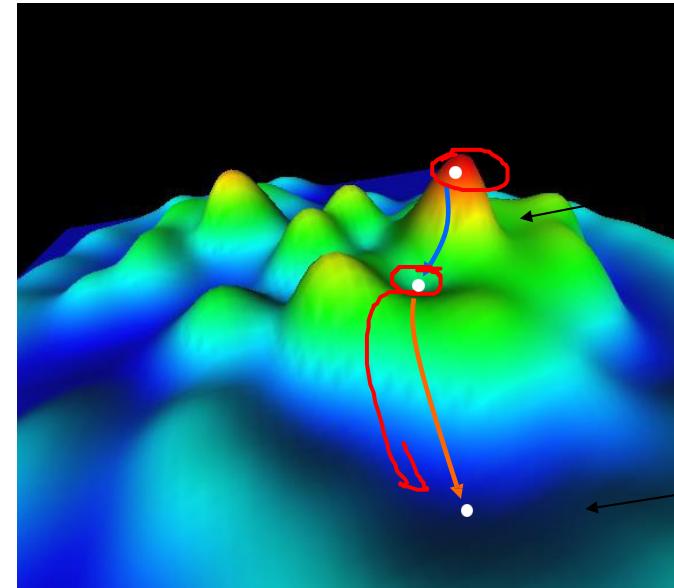


Disadvantages

- Computationally slow... $O(n^3)$
 - Can be sped up using spatial partitioning and calculating repulsion for only near-by nodes
 - Also can employ multi-level computation like Barnes-Hut and handle 1M node graph in seconds
- Visual limit is probably around 10K vertices/edges...hairball problem
- Layout can get trapped in locally optimal rather than globally optimal state



<http://www.research.att.com/yifanhu/GALLERY/GRAPHS/index1.html>

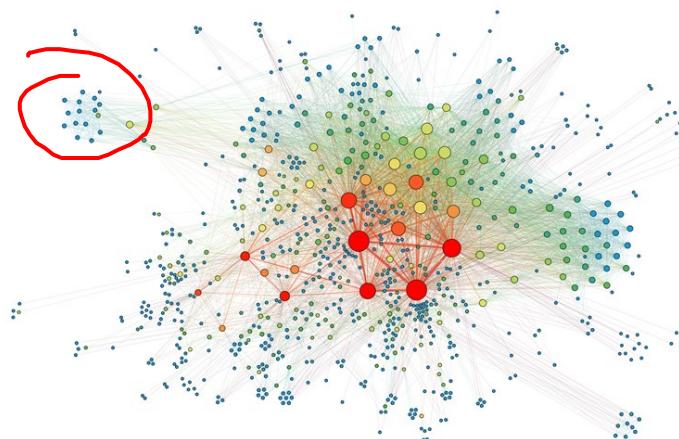


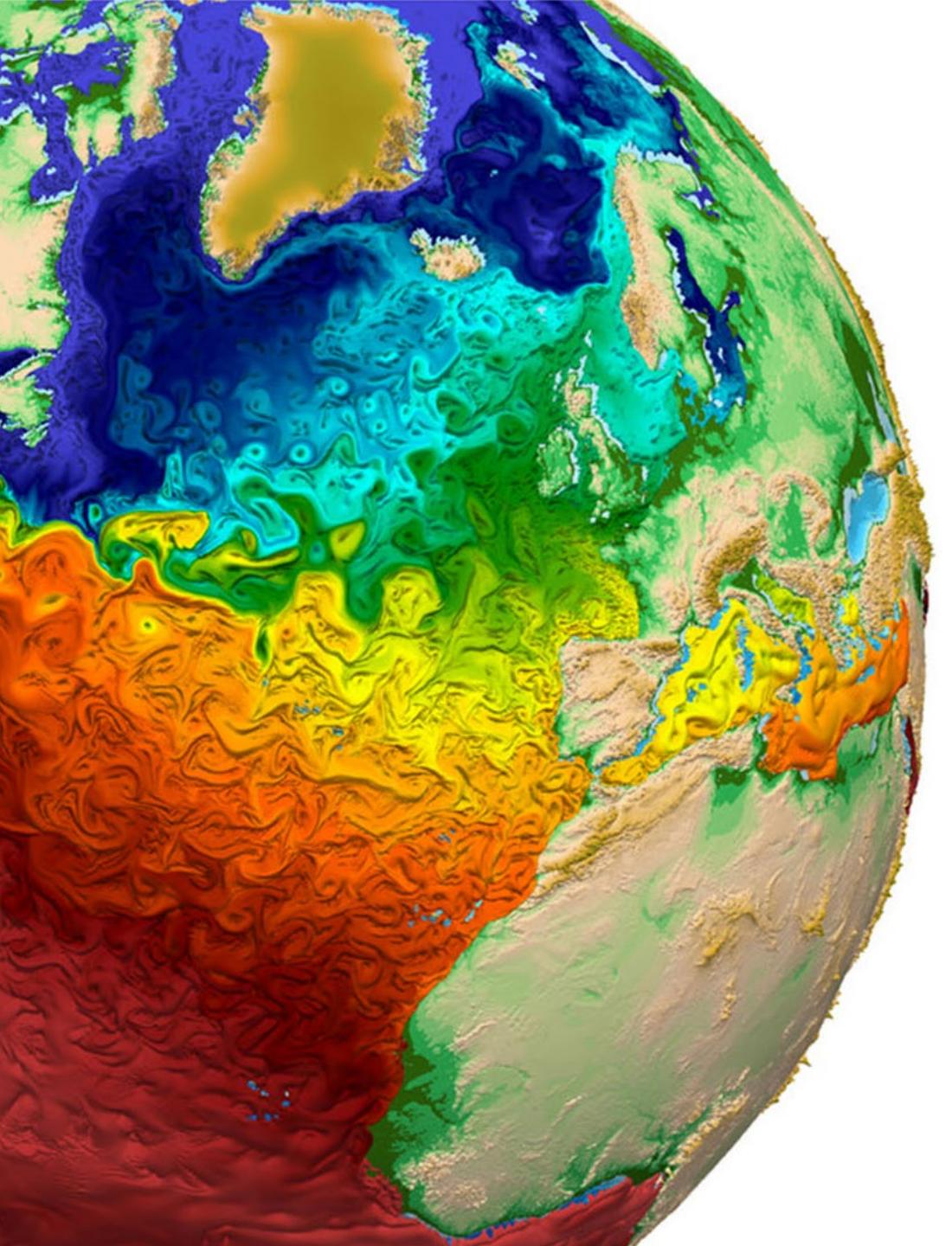
...minimizer may get stuck here (*local* minimum)

...instead of arriving here (*global* minimum)

Force-Directed Layout Advantages

- Simple to implement
- Can work on any graph (e.g. not just trees)
- Can change force functions to use other data for specific applications
 - e.g. directed edges, edge weights, different classes of nodes, etc.
- Can be interactive
- Works pretty well



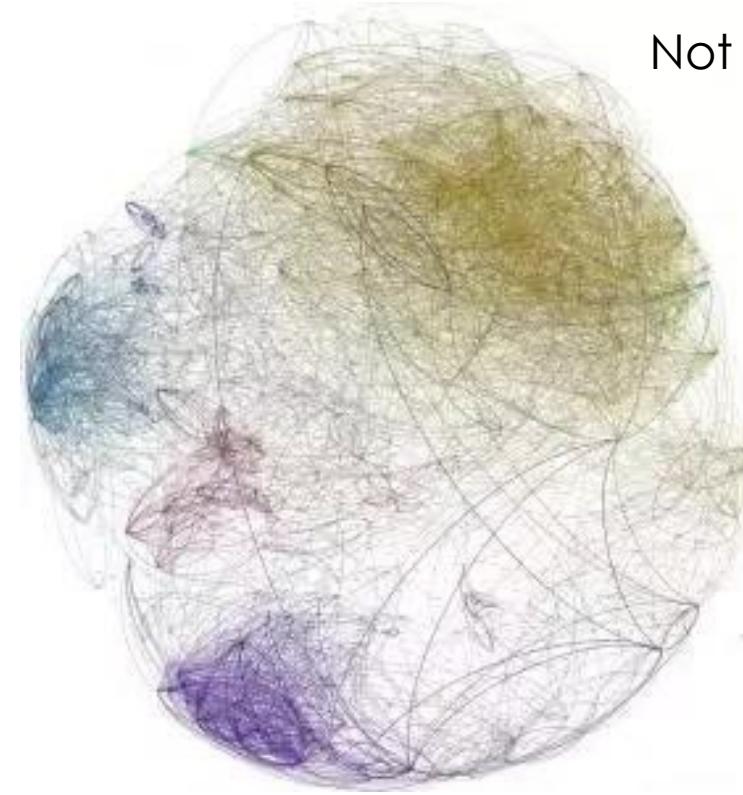


Large Graph Visualization Edge Filtering

Scientific Visualization
Professor Eric Shaffer

Large Networks Are Problematic

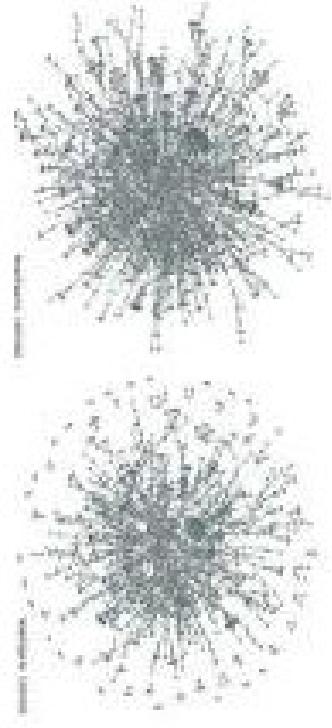
- 2012 webgraph:
- 3.5 billion pages 128 billion links
- Probably don't have enough pixels
- Even if we did, probably don't have enough cognitive capacity



Graph Preprocessing

Idea: We can visualize smaller graphs

Let's make the big graph into a small graph
...try to keep the most important parts



Two approaches:

- Graph filtering: remove unimportant parts
- Graph aggregation: merge similar graph elements together

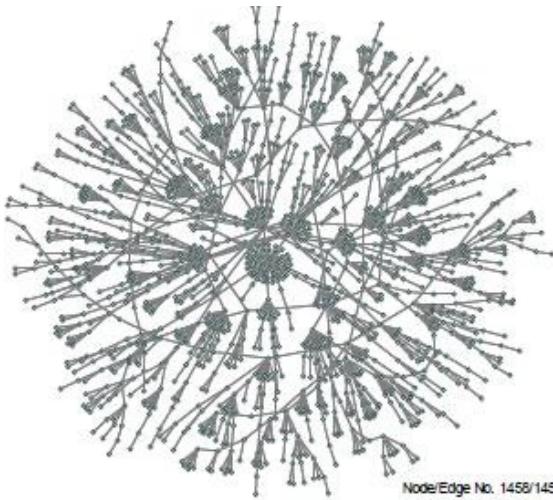
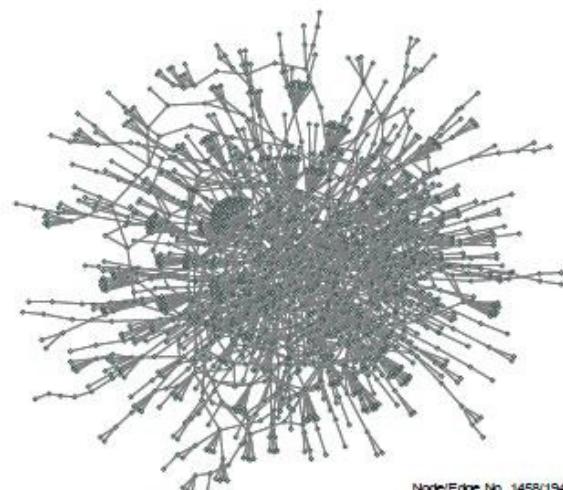
Graph Filtering

JIA Y., HOBEROCK J., GARLAND M., HART J.:

On the visualization of social and other scale-free networks.

IEEE Transactions on Visualization and Computer Graphics (2008)

- Removes edges in order of increasing betweenness centrality
- Preserves connectivity
- Preserves graph features (e.g. cliques)

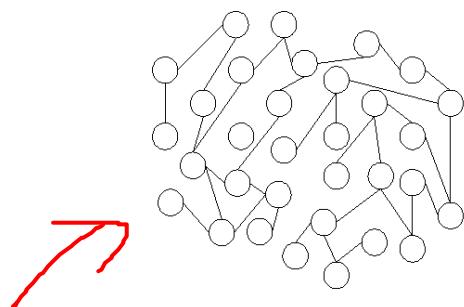


Scale-Free Networks

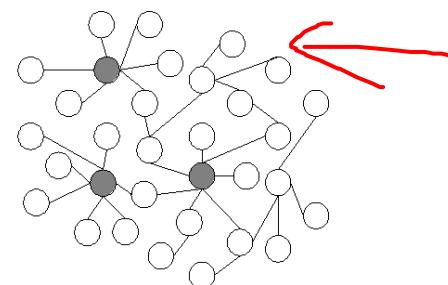
Real-world networks are often claimed to be scale free

Meaning that the fraction of nodes with degree k follows a power law $k^{-\alpha}$

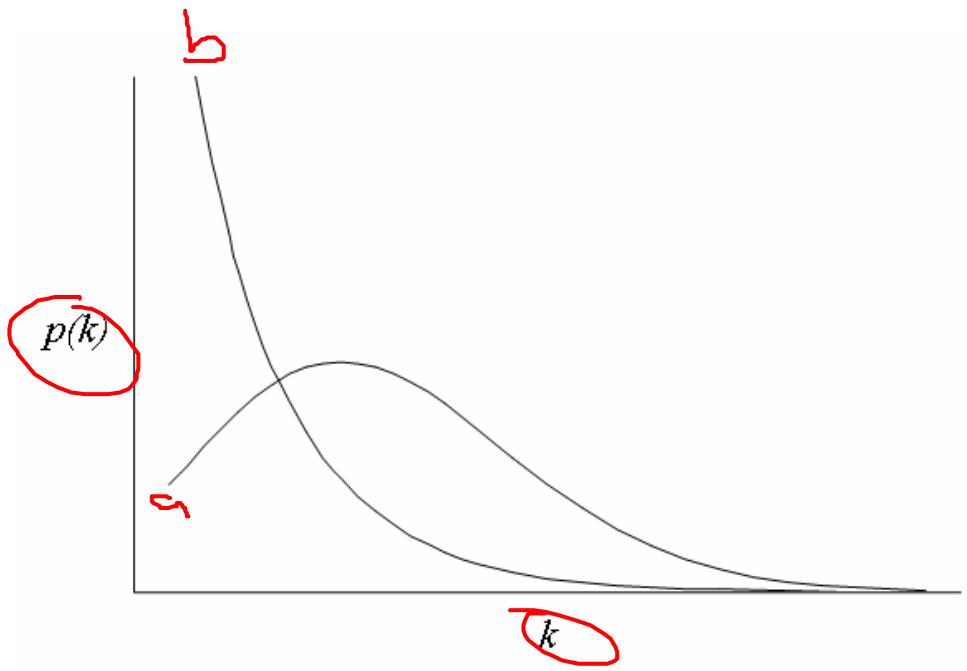
- A few nodes are hubs with many incident edges
- Many nodes have few incident edges
- Social networks were thought to be scale-free



(a) Random network



(b) Scale-free network



Current Research in Networks

Article | Open Access | Published: 04 March 2019

Scale-free networks are rare

Anna D. Broido  & Aaron Clauset 

Nature Communications **10**, Article number: 1017 (2019) | [Cite this article](#)

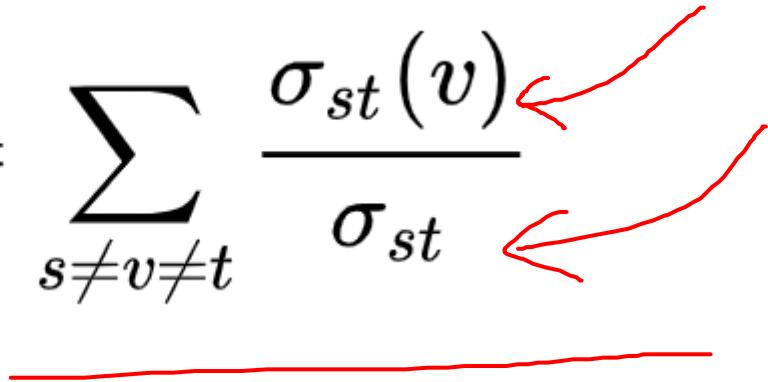
36k Accesses | **119** Citations | **631** Altmetric | [Metrics](#)

- Statistical analysis has shown that few empirical data sets are truly scale-free
- Social networks currently thought to be weakly scale-free...

Centrality-based graph filtering can be applied to non-scale-free networks...just won't work as well



Betweenness Centrality for Vertices

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$


σ_{st} is the total number of shortest paths from node s to node t

$\sigma_{st}(v)$ is the number of those paths that pass through node v

Betweenness Centrality for Edges

$$g(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}}$$

σ_{st} is the total number of shortest paths from node s to node t

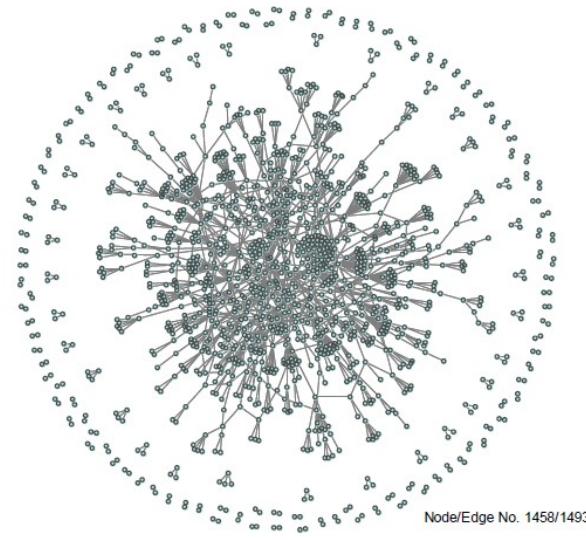
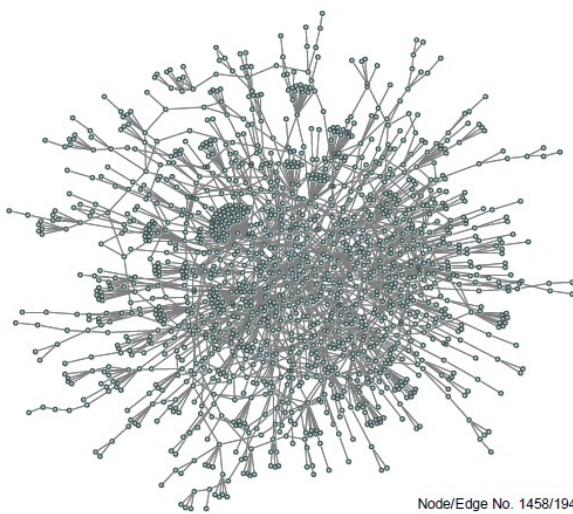
$\sigma_{st}(e)$ is the number of those paths that pass through edge e

Betweenness Centrality

Betweenness Centrality (BC) ranks edges (or vertices)

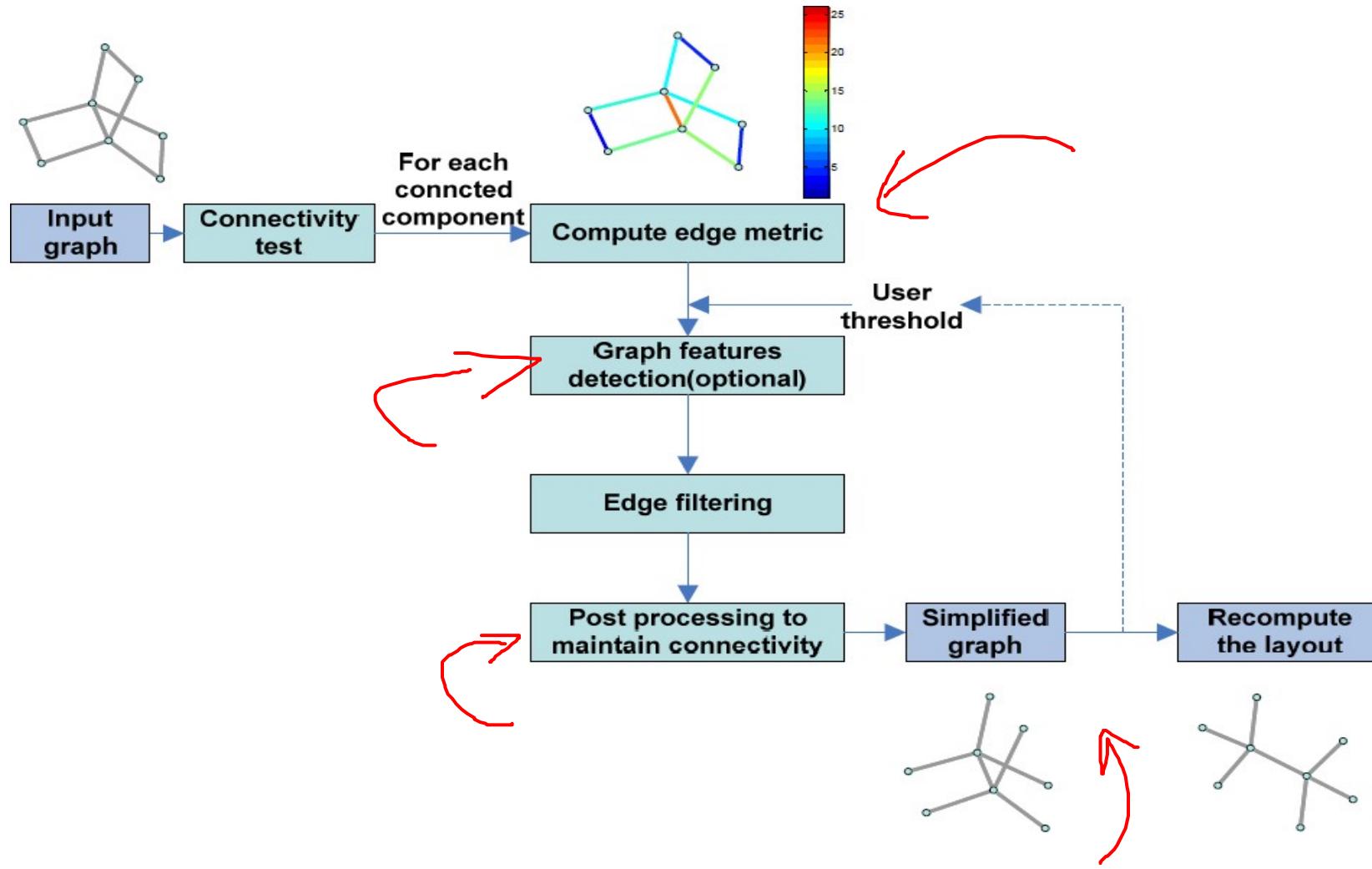
- How often they appear on shortest paths
- High BC → important communication tunnels
- Low BC → less important
- Remove low BC edges
- Keeps “back bone” of the graph

Simple Edge Filtering is Insufficient



Need to maintain connectivity...possibly other important features

Workflow



Betweenness Centrality is Expensive

Graph $G = (V, E)$, $|V| = n$, $|E| = m$

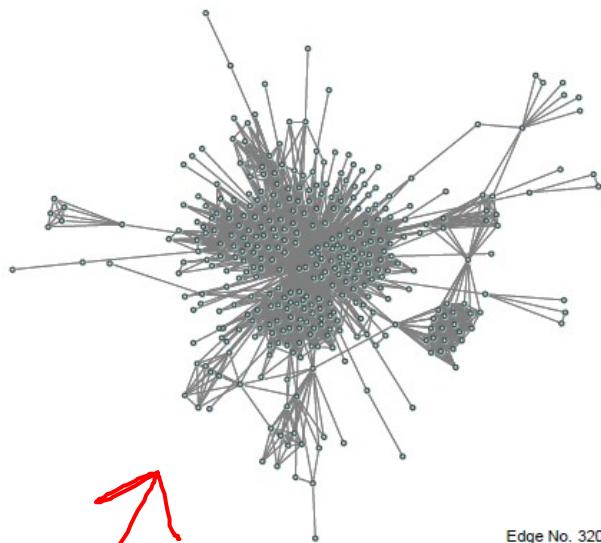
- Betweenness centrality [Freeman 1977]
- Relies on computing All-Pairs Shortest Paths
- Complexity $O(m^*n)$ for unweighted graph [Brandes 01]

For huge graphs

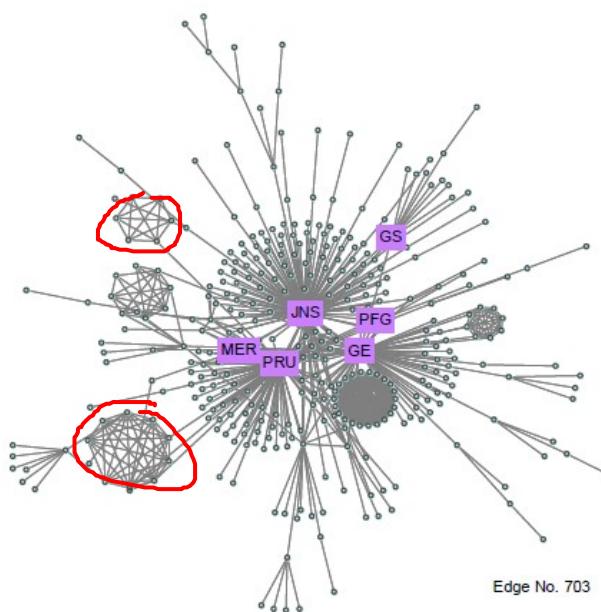
- Approximated with random sampling [Jacob et al. 05]
 - $O((m+n)^*/\log(n))$ with $C^*/\log(n)$ samples where C is a constant
- For our edge filtering purpose
 - Only relative orders of BC are needed
 - Select $C^*/\log(n)$ highest degree hub nodes

Graph Feature Detection

- Graph features
 - Cliques
 - NP-Complete problem
 - Fast approximation $O(m*n)$ [Chircota et al. 03]
- User defined features



Edge No. 3206



Edge No. 703



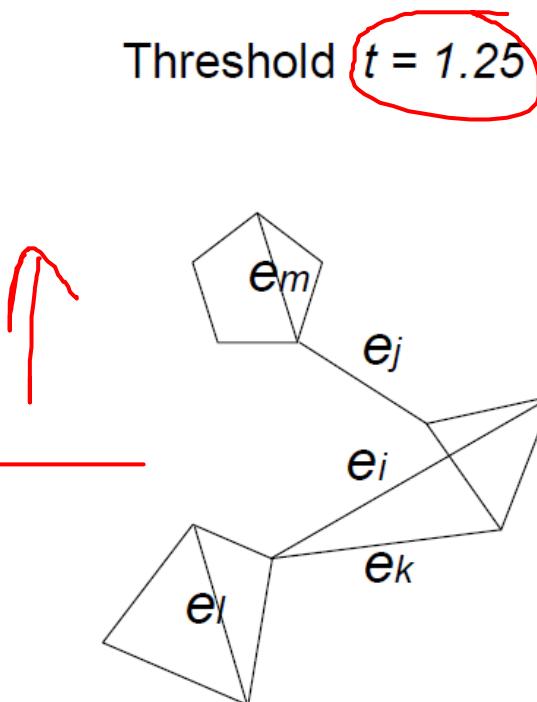
Edge Filtering

Edges	BC Metric
...	...
e_h	1.3
e_i	1.1
e_j	1.2
e_k	1.15
e_l	1.21
e_m	1.09
...	...

Sort

Edges	BC Metric
e_m	1.09
e_i	1.1
e_k	1.15
e_j	1.2
e_l	1.21
e_h	1.3
...	...

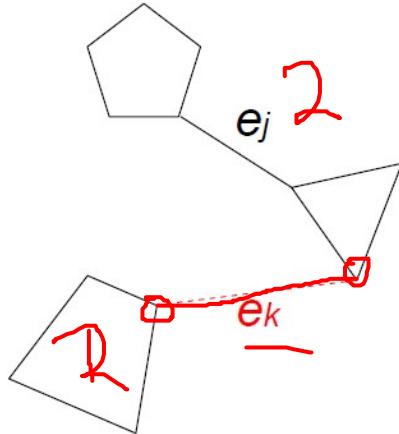
Threshold $t = 1.25$



Recover Connectivity

e_l	1.21
e_j	1.2
e_k	1.15
e_i	1.1
e_m	1.09
Removed Edges	BC Metric

stack

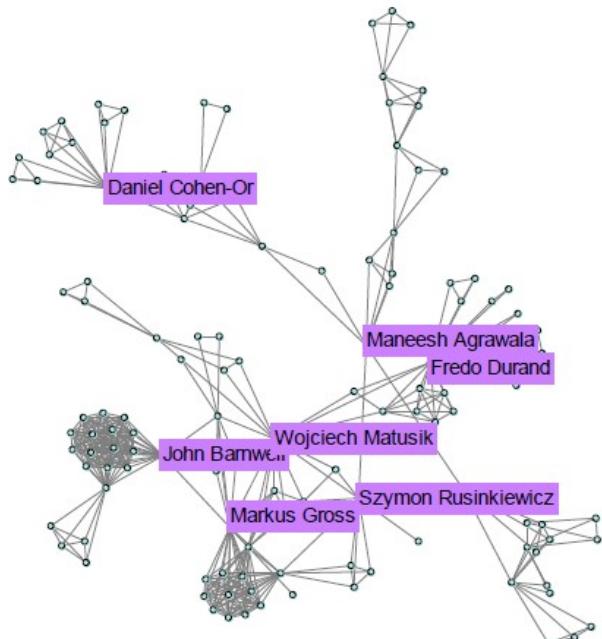


- Compute connected components of graph and label vertices by component
- Iterate through the removed edges in reverse order of removal.
- If an edge links a pair of nodes belonging to different components
 - restore that edge and unify components and labels
- The iteration continues until graph contains a single component

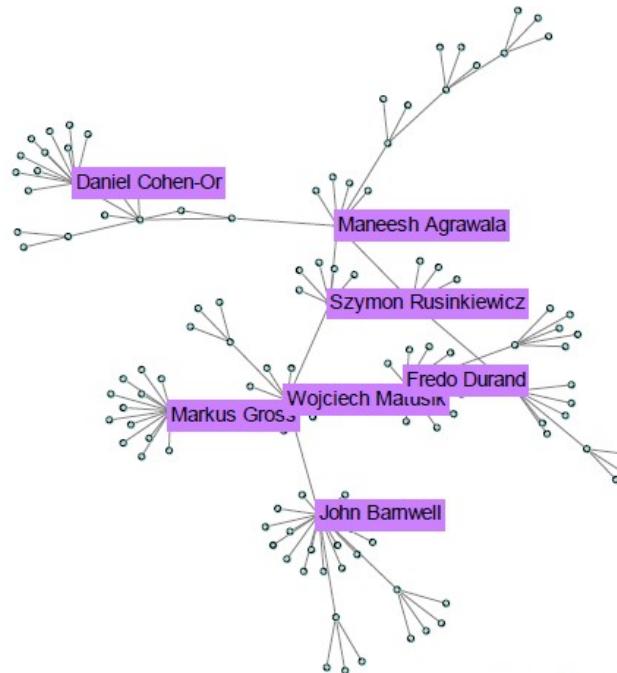
Recompute the Layout

After filtering, apply a force-directed layout

Fewer edges...should be more efficient and less visually cluttered

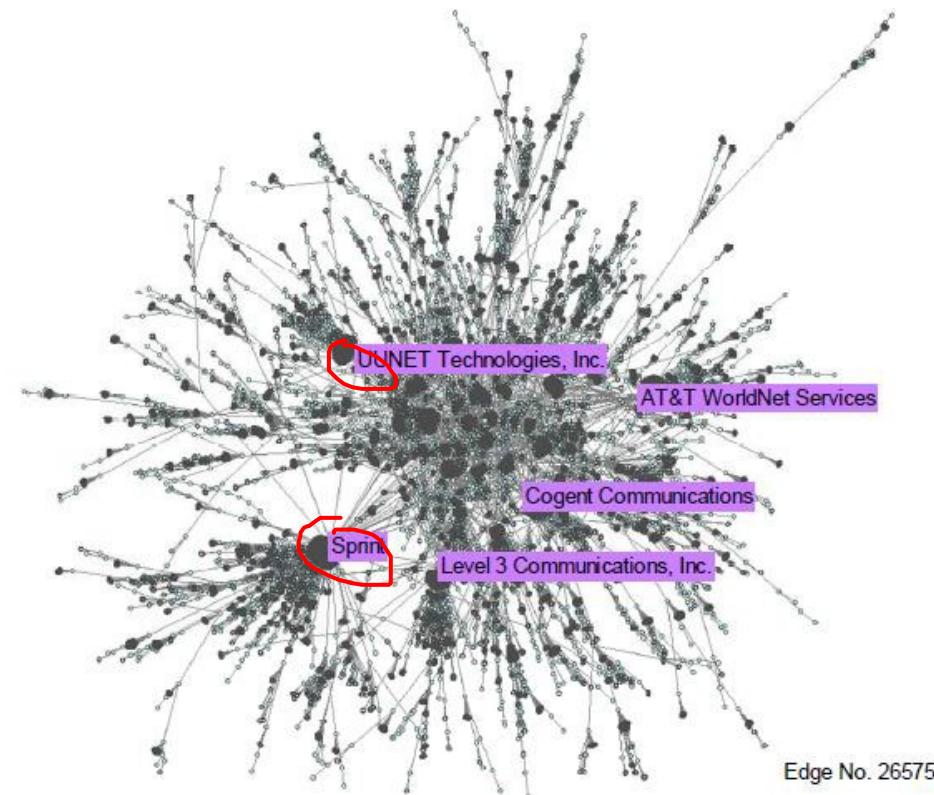
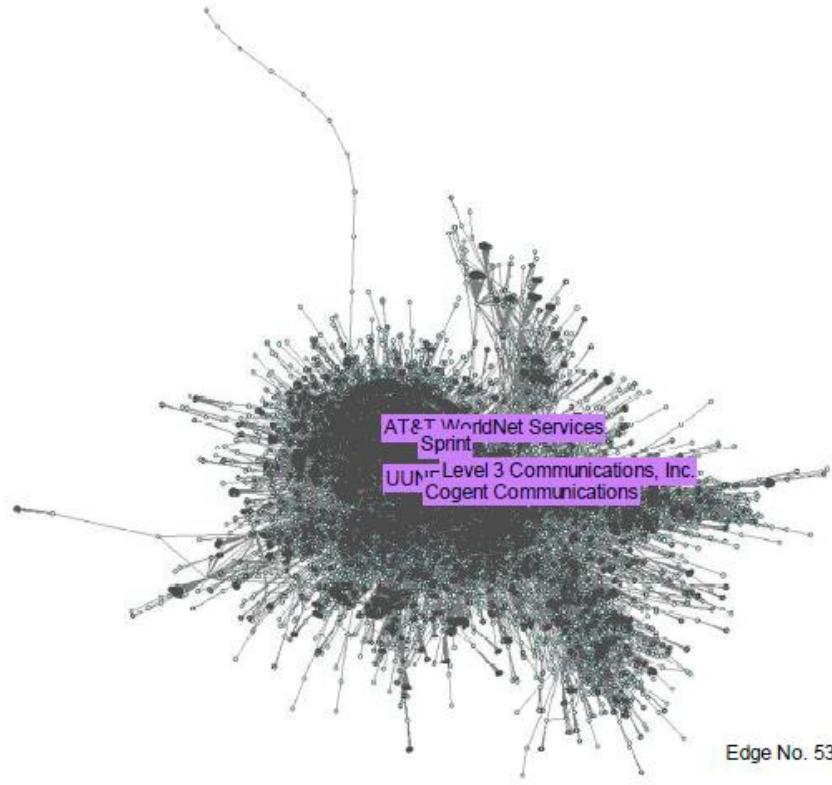


Edge No. 441

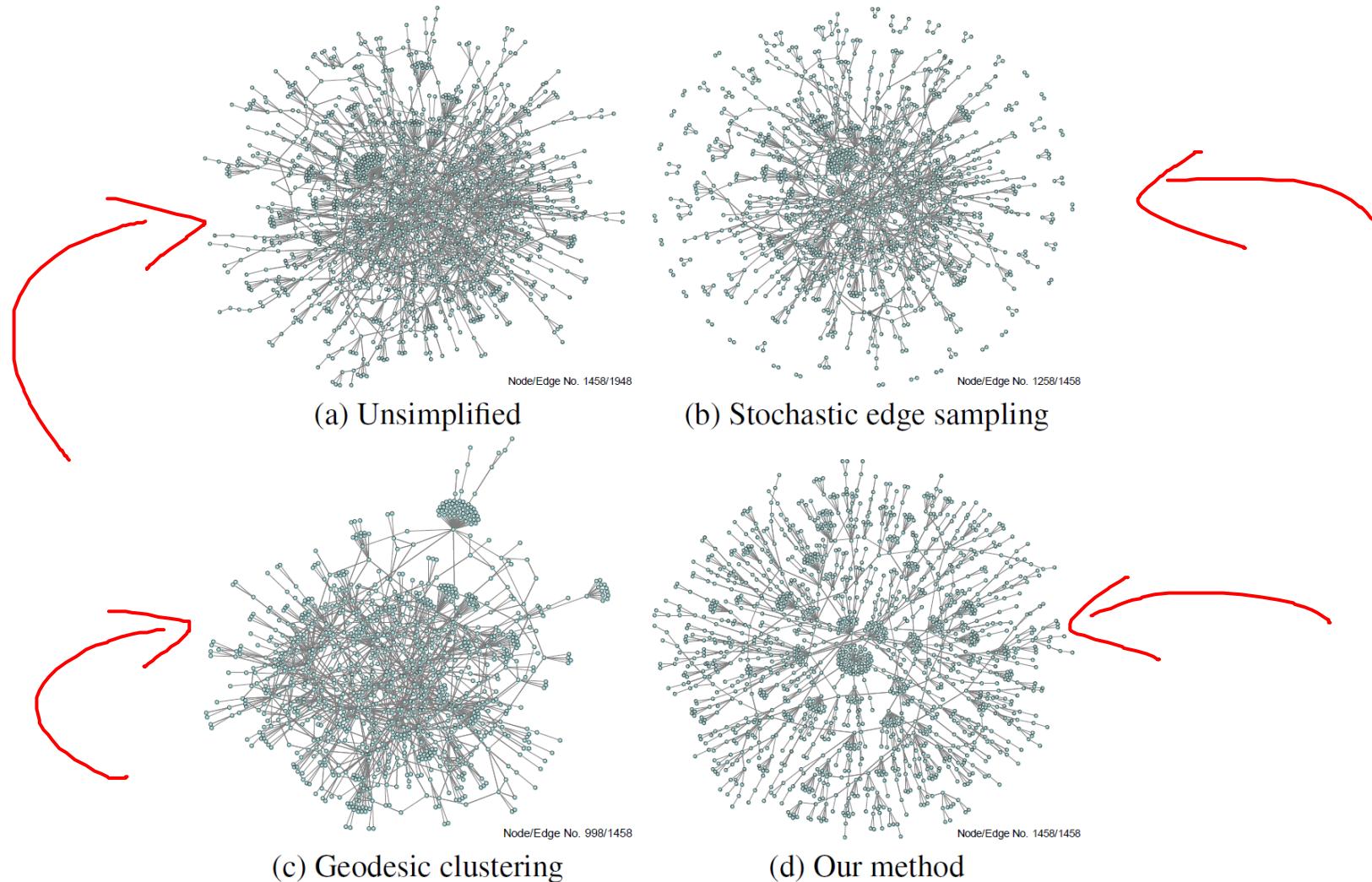


Edge No. 129

Fixing the Hairball....

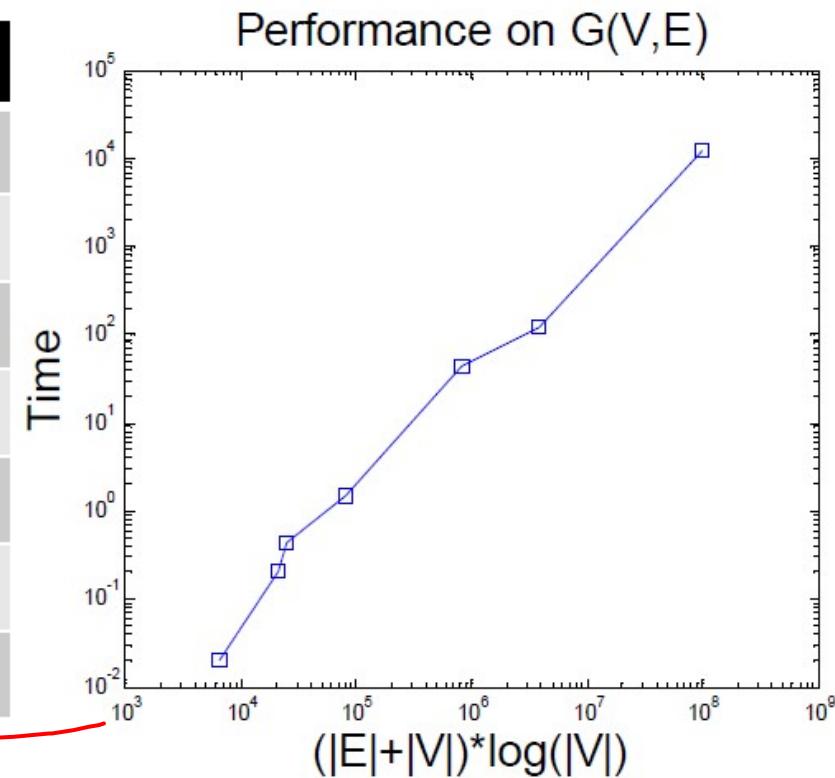


Comparison



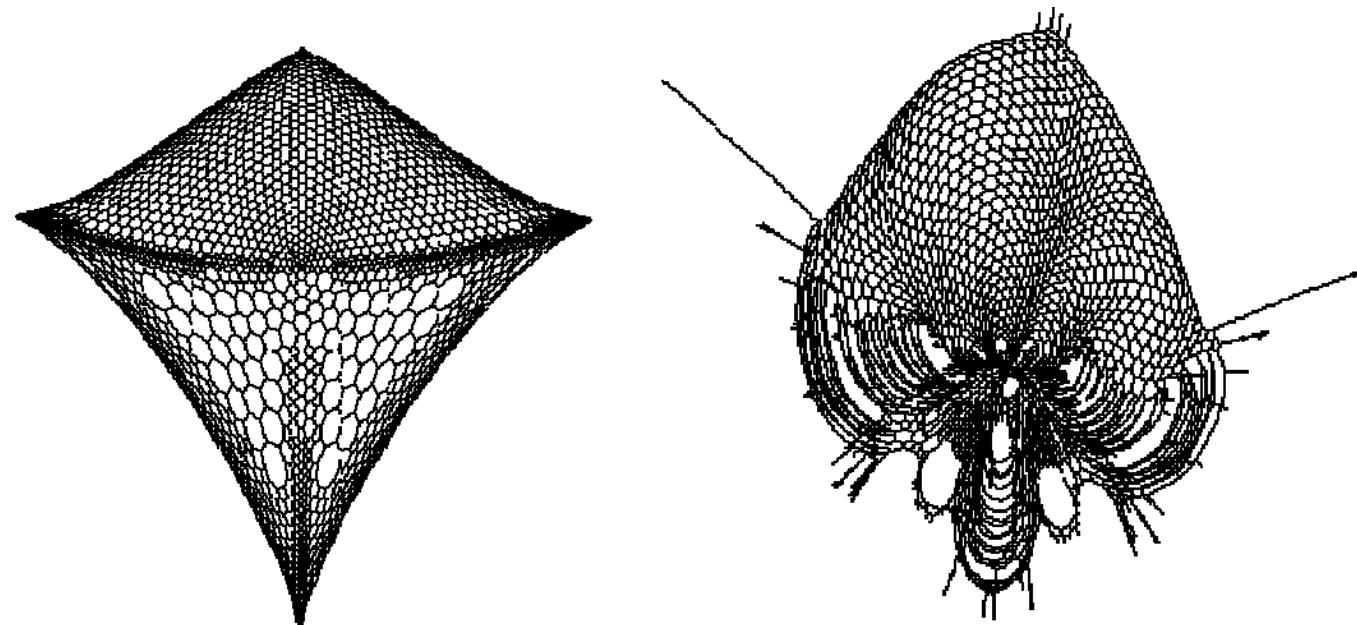
Performance

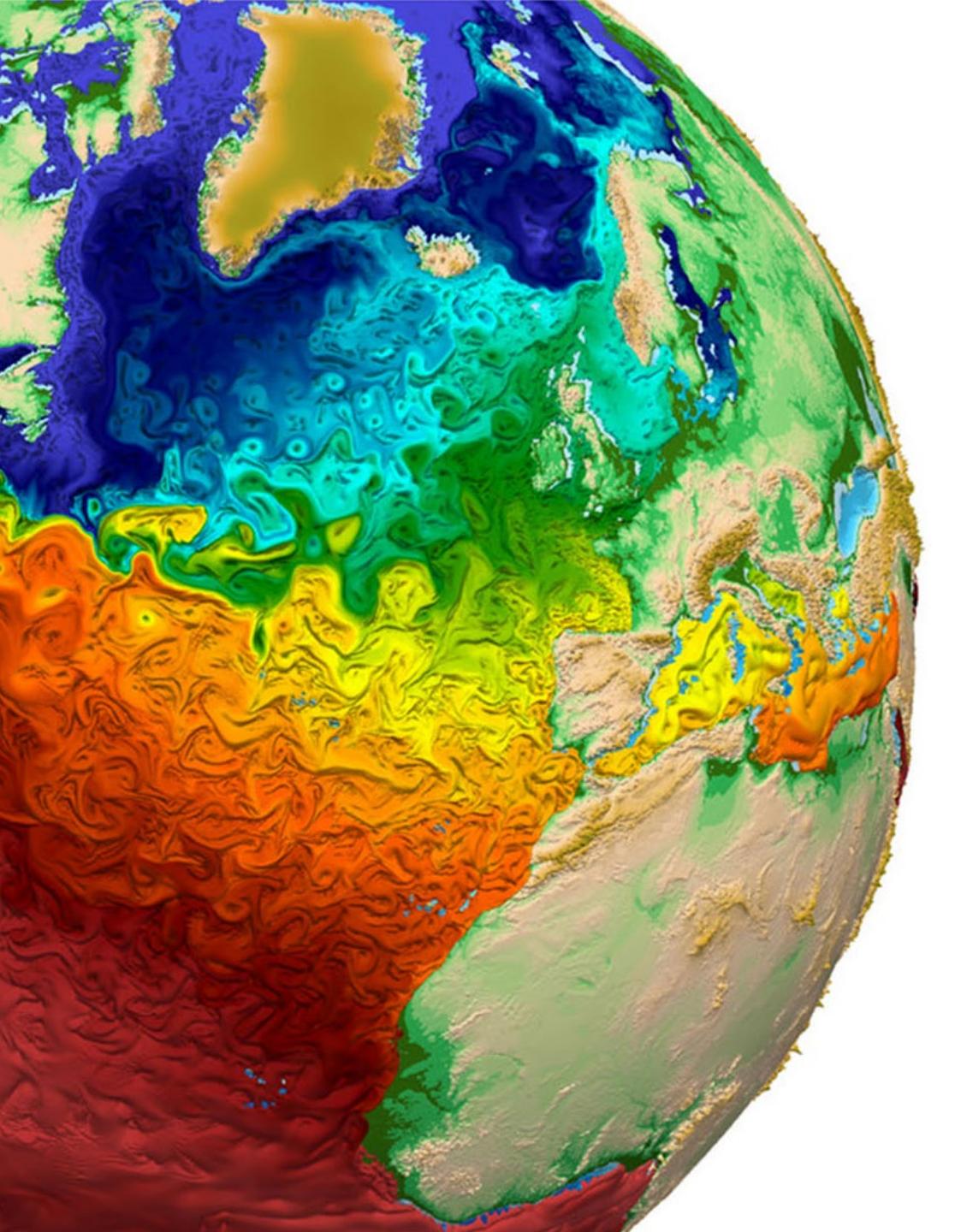
Graph	Nodes	Edges	Timing
siggraph07	328	773	0.02s
sp500-038	365	3206	0.20s
bo	1458	1948	0.44s
cg_web	2269	8131	1.50s
as-rel.071008	26242	53174	43.66s
hep-th	27400	352021	120.72s
flickr	820878	6625280	12442.70s



Limitations

- Doesn't work well for non-power law graphs
 - Including planar graphs
 - Clustering may be a better choice than filtering
- Obviously doesn't show entire data set



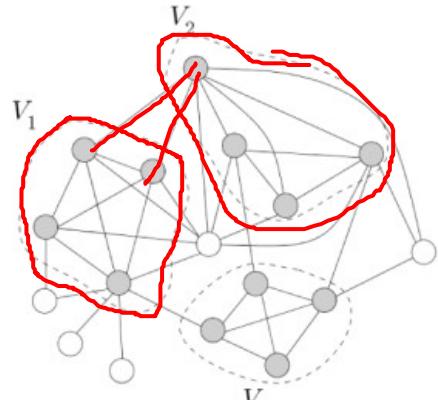


Large Graph Visualization

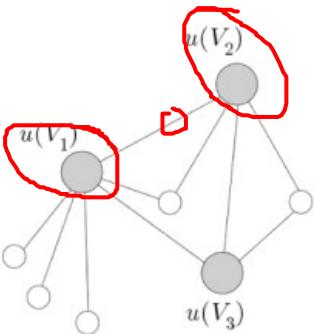
Edge Bundling

Scientific Visualization
Professor Eric Shaffer

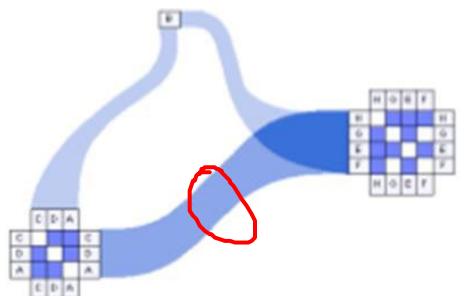
Graph Aggregation



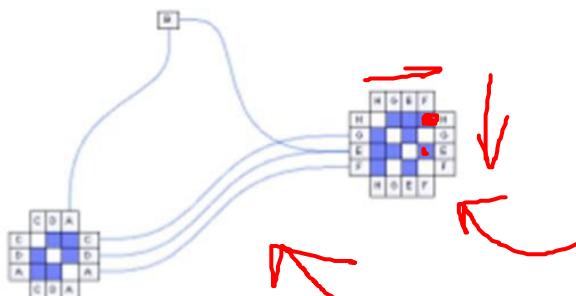
(a)



(b)



aggregated edges



original edges

Produces a simpler/smaller ‘cluster graph’ from a large one

Vertices: partitioned between disjoint clusters

Edges: often aggregated between clusters

Many clustering methods (strongly-connected components, data based, ...)

Visualize the cluster graph

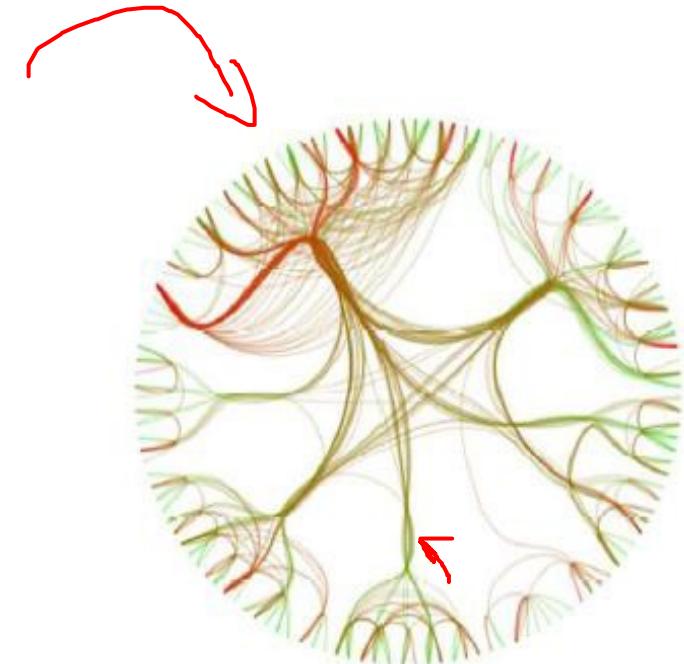
Cluster internals shown using a cluster icon (e.g. a matrix plot)

Graph Aggregation: Edge Bundling

- Edge bundles are clusters of similar edges
- Many approaches...usually cluster vertices
- Edges between clusters follow similar paths

Some metrics

- Shortest path distance to a “hub-node”
- Remove high-BC edges to discover clusters
- Lots of others.....



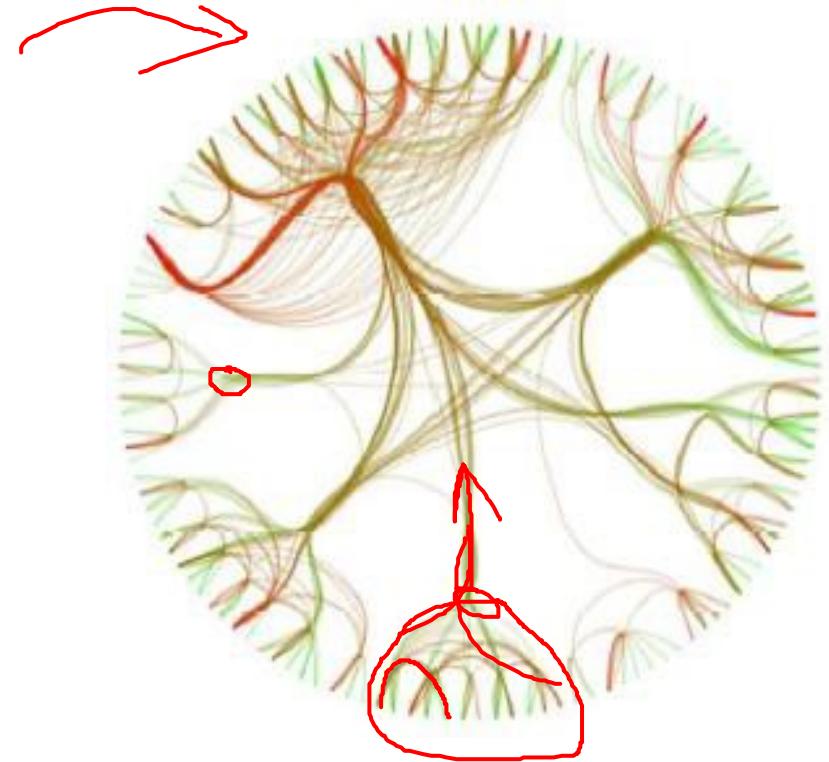
Hierarchical Edge Bundling: Example

Yuntao Jia, Michael Garland, John C. Hart:

Social Network Clustering and Visualization using Hierarchical Edge Bundles.

Computer Graphics Forum (2011)

1. Generate a hierarchical structure of vertex clusters
2. Vertices are placed radially around circle
 1. Positions from in-order traversal of hierarchy
 2. Root nodes of clusters in interior, leaves on the perimeter
3. Edges are B-Spline curves
 1. Control points are hierarchy node layout positions
 2. Positions along shortest tree path between the two nodes



Community Discovery

- Edge betweenness centrality

$$\text{BC}(u, v) = \sum_{s,t,u,v \in V} \frac{\sigma_{s,t}(u, v)}{\sigma_{s,t}}$$

- σ_{st} is the total number of shortest paths from node s to node t
- $\sigma_{st}(u, v)$ is the number of those paths that pass through edge u,v
- Low BC edges connect nodes within a community
- High BC edges connect communities

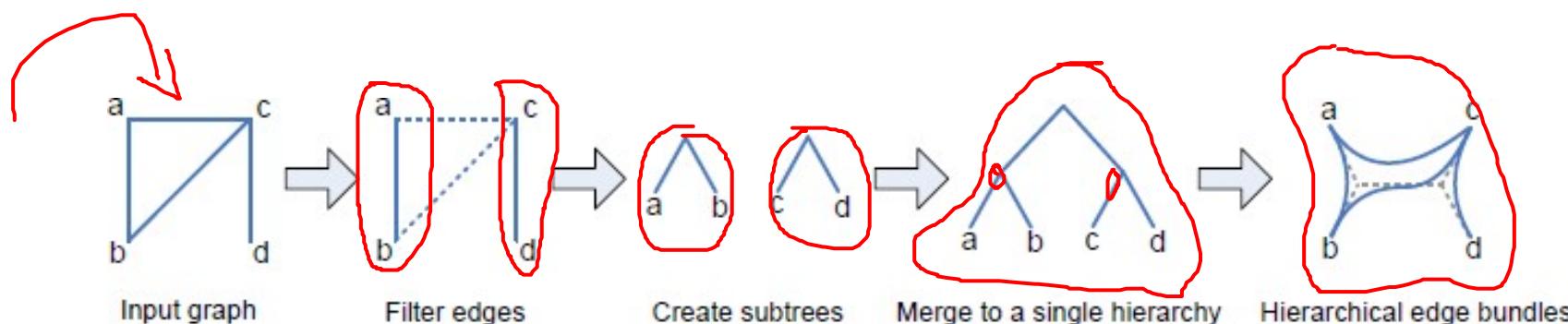
Balanced Hierarchy Construction

Filter edges by removing highest-bc edges first

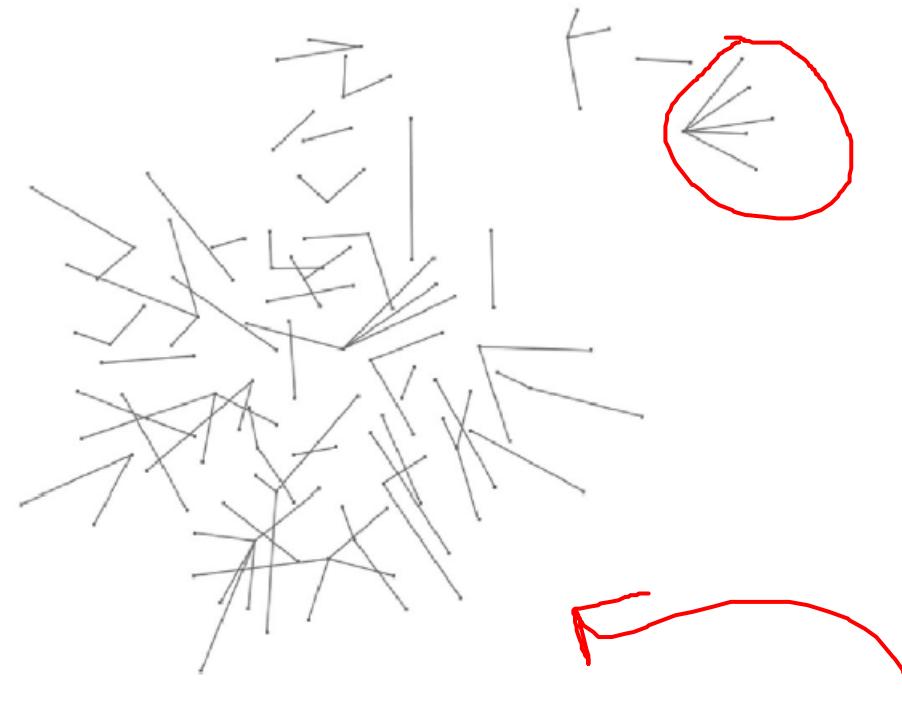
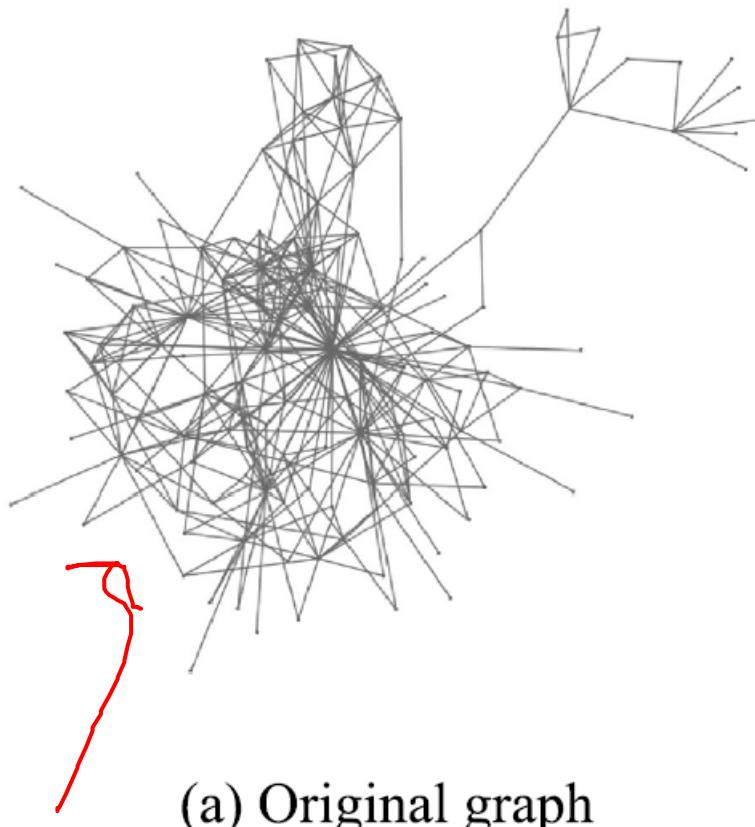
- An edge a,b is removed only if $\min(\deg(a), \deg(b)) > 1$ and $BC(a,b) > 1$



Construct communities by merging in increasing BC order of removed edges



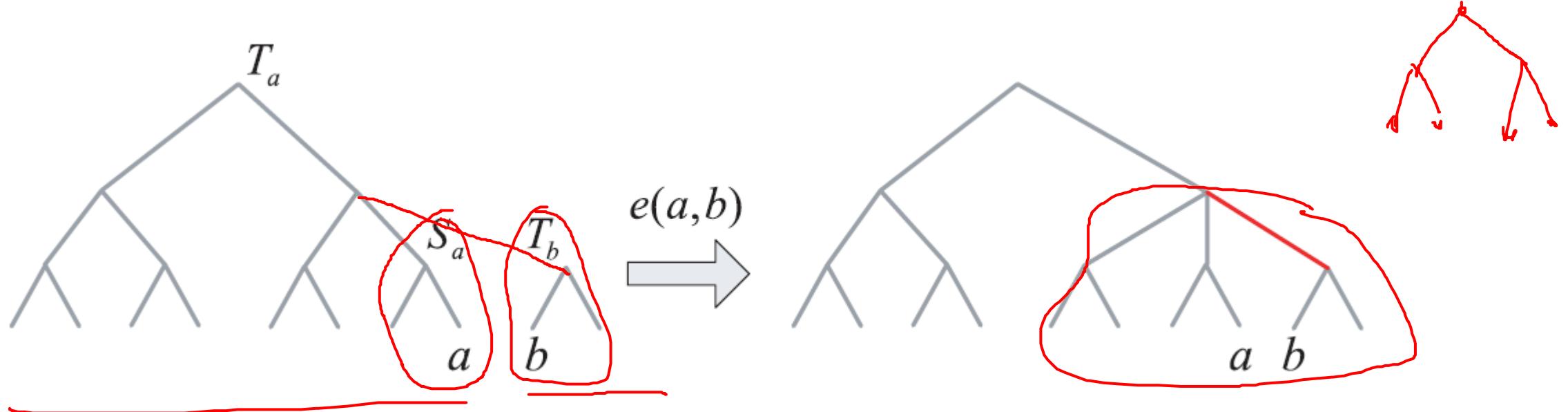
Edge Removal



(a) Original graph

(b) After edge removal

Merging Communities into a Hierarchy



We scan the list of removed edges in order of increasing BC and merge subtrees connected by those edges.

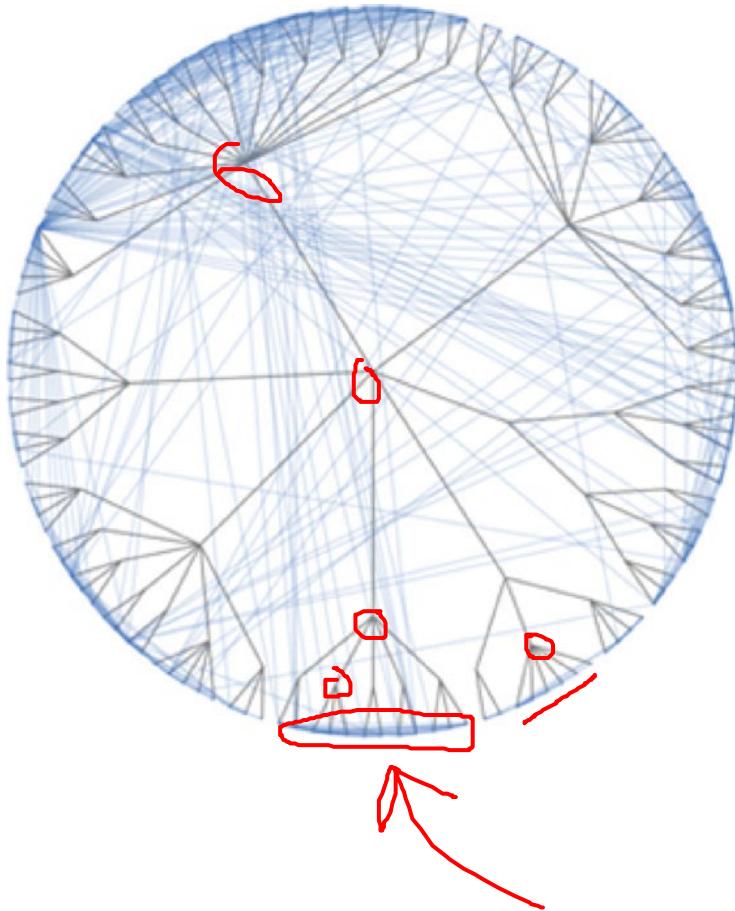
If T_a and T_b share the same height, then they can be merged as children of the same new parent tree node.

Otherwise assume without loss of generality that T_a is taller than T_b .

Let S_a be the unique (lowest) subtree of T_a that contains a and shares the same height as T_b .

Then the communities are merged by assigning the parent of S_a as the parent of T_b .

Using the Hierarchy to Bundle Edges



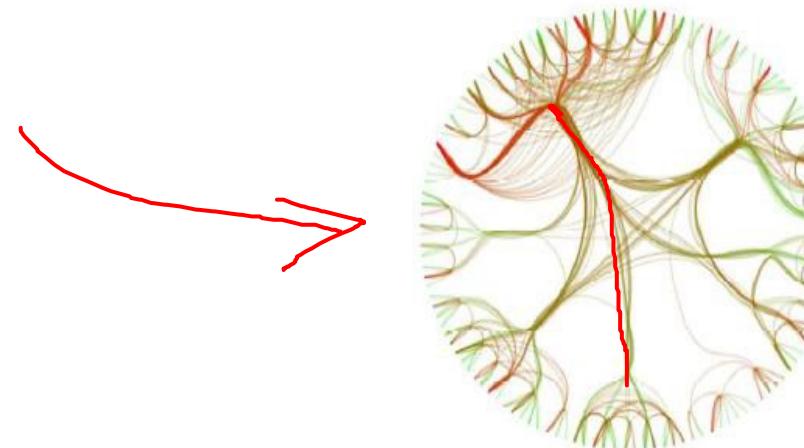
Gray lines show the computed hierarchy used to layout the graph

Blue lines show edges drawn linearly between nodes

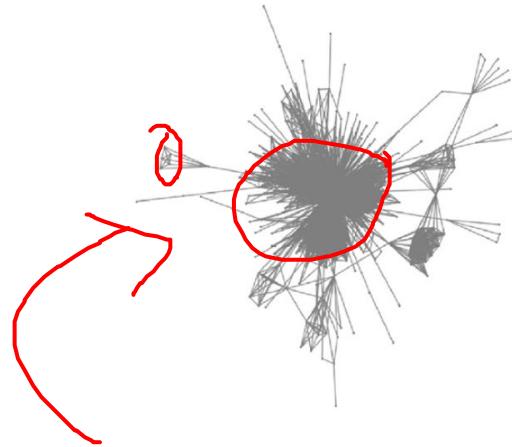
Edge bundling is accomplished by drawing them as curves

Curves with control points defined by the internal nodes

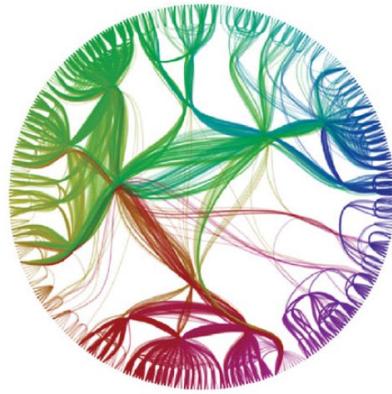
Edges between 2 communities will be drawn with similar curves



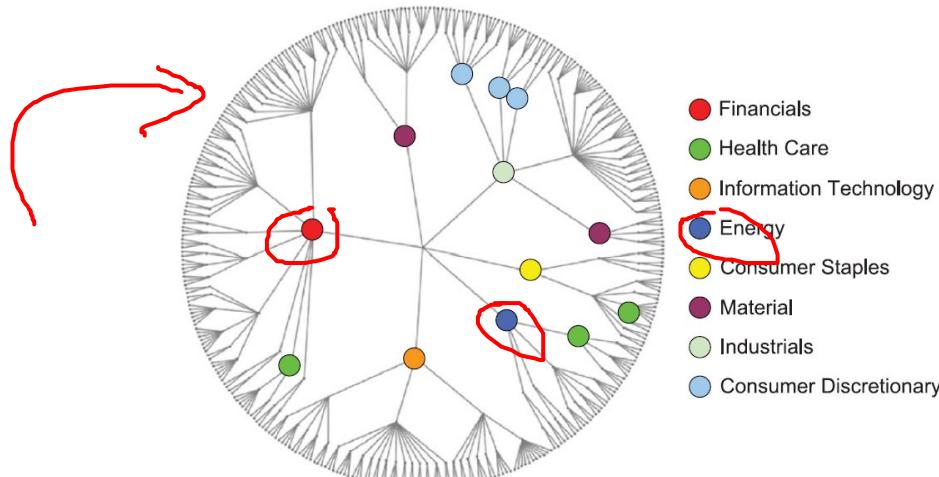
Example: E-mail in the Enron Scandal



(a) Original graph

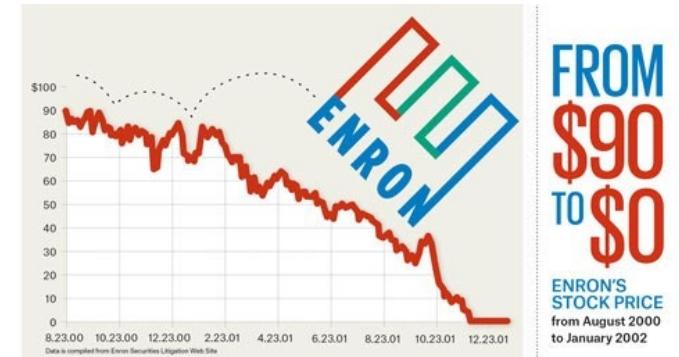


(b) Edge Bundles

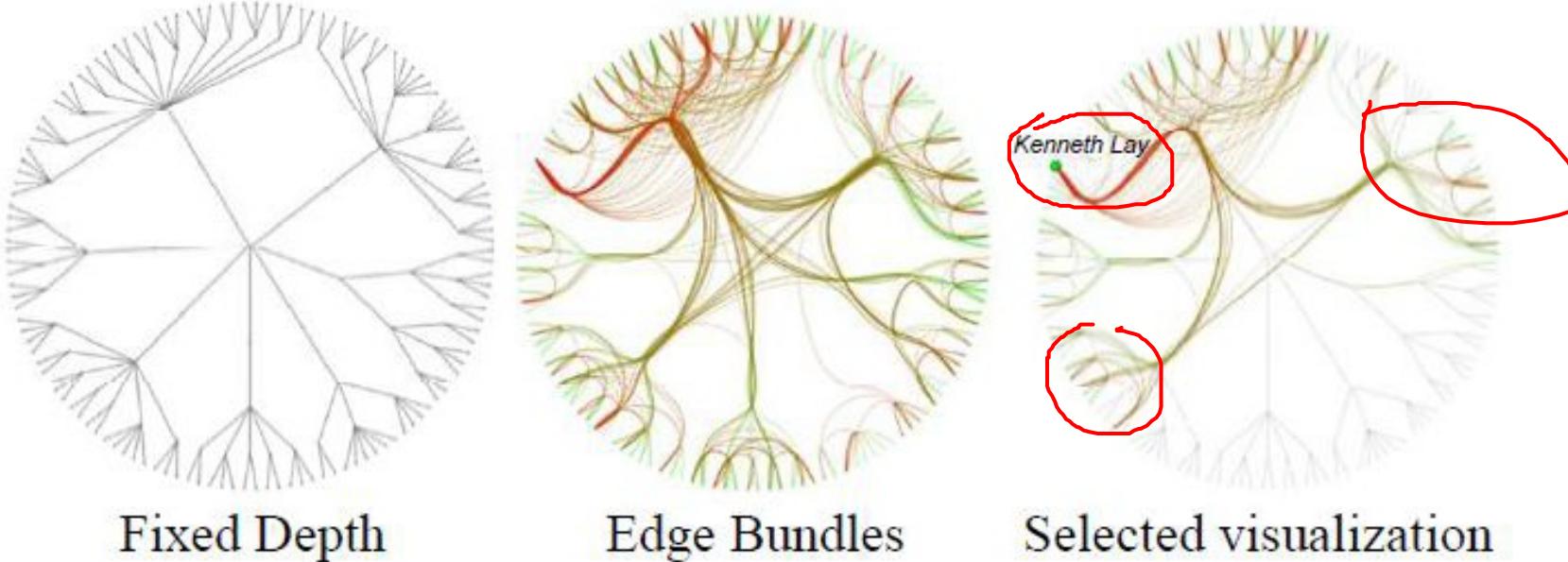


(c) Generated Hierarchy (with user supplied labels)

The Enron scandal was an accounting scandal of Enron Corporation, an American energy company based in Houston, Texas. It was publicized in October 2001, and led to the bankruptcy of the company, and the de facto dissolution of Arthur Andersen, which was one of the five largest audit and accountancy partnerships in the world. In addition to being the largest bankruptcy reorganization in American history at that time, Enron was cited as the biggest audit failure.
- Wikipedia



Enron E-Mail Graph



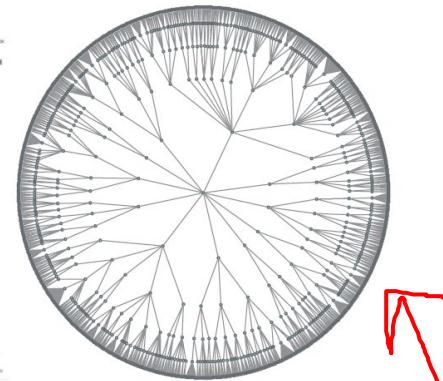
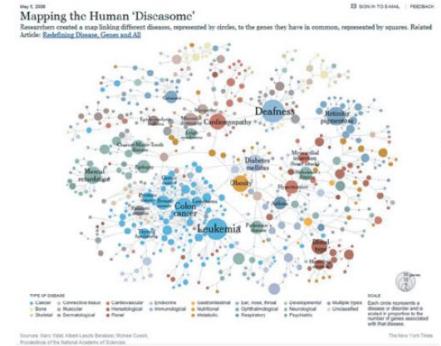
Enron scandal 2001

389 e-mails, 132 employees

Red = sender, Green = recipient

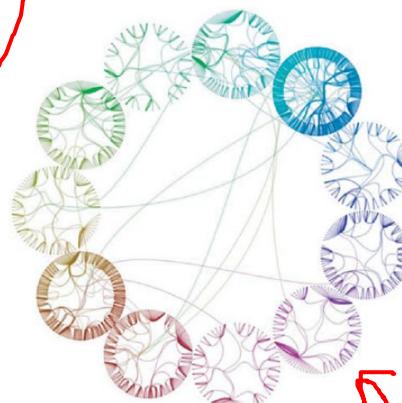
Can select node to see which communities that person contacted

Example: Diseasome

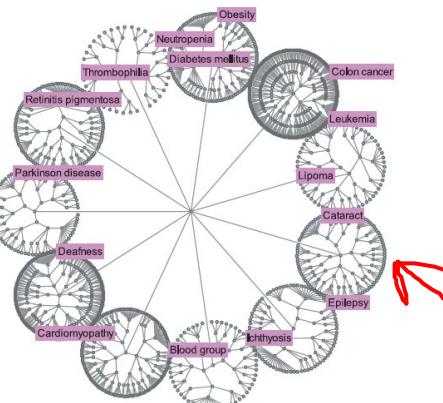


(a) "diseasome"

(b) Generated hierarchy



(c) Edge bundles



(d) Adaptive hierarchy drawing

A bipartite human disorder-gene network

1550 associations between 1419 disorders and genes

A disorder and gene connect if disorder arises from mutation of the gene

(a) shows a visualization of the network published in New York Times

- squares correspond to genes and circles correspond to disorders
- coloured by disorder types
- sized by the number of gene links

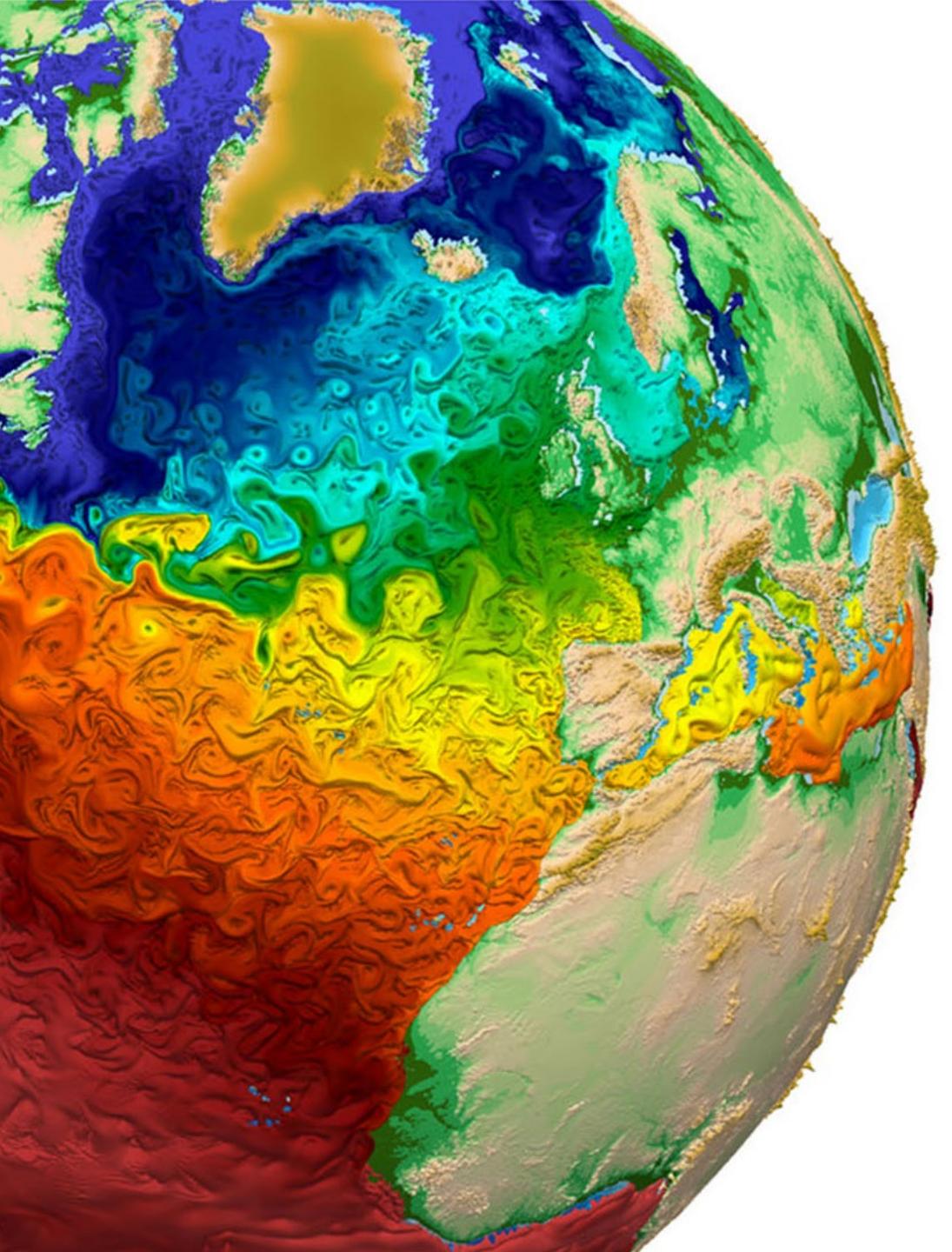
(b) HEB clusters form a community hierarchy

(d) Hierarchy laid out using adaptive tree drawing

- all first level internal nodes are drawn on different layout circles
- disorders are also labelled
- same type of disorders is likely clustered into the same community

(c) disorder–gene edge bundles are drawn on top of hierarchy

- seldom connections between communities

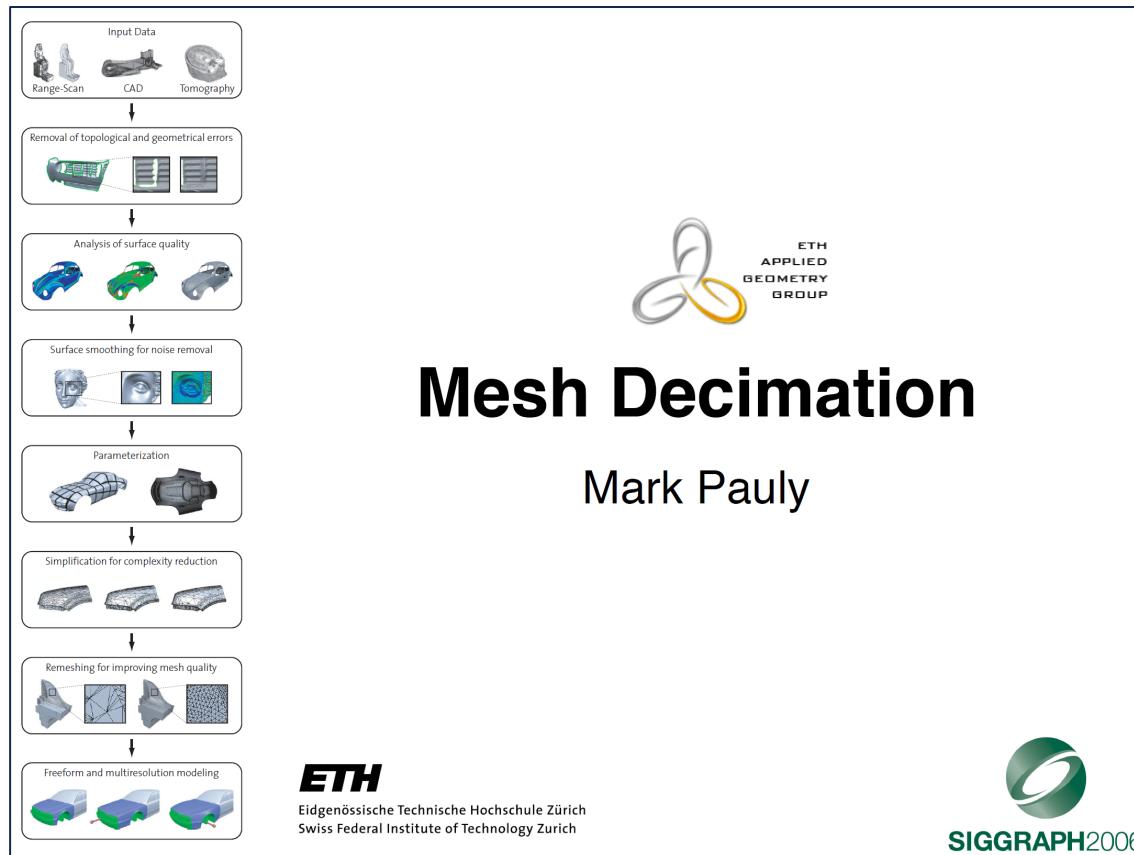


Mesh Simplification

Scientific Visualization
Professor Eric Shaffer

Acknowledgements

Slides based on presentation by Professor Mark Pauly of ETH Zurich

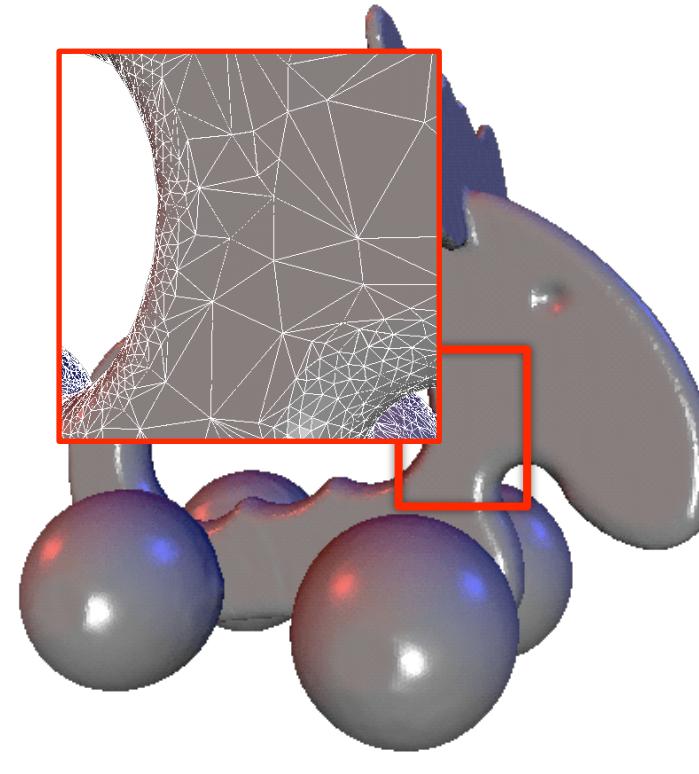
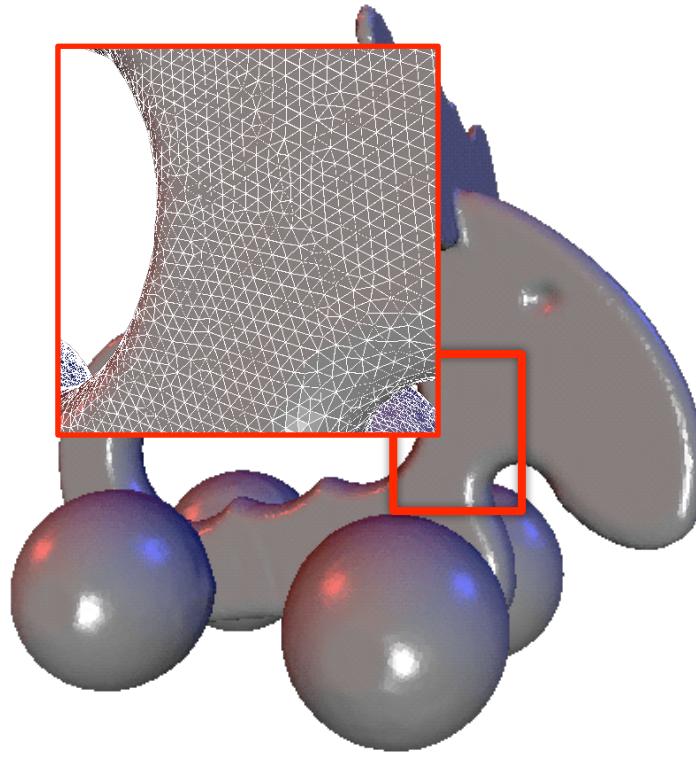


<http://www.pmp-book.org/>



Surface Meshes often Overtesselated

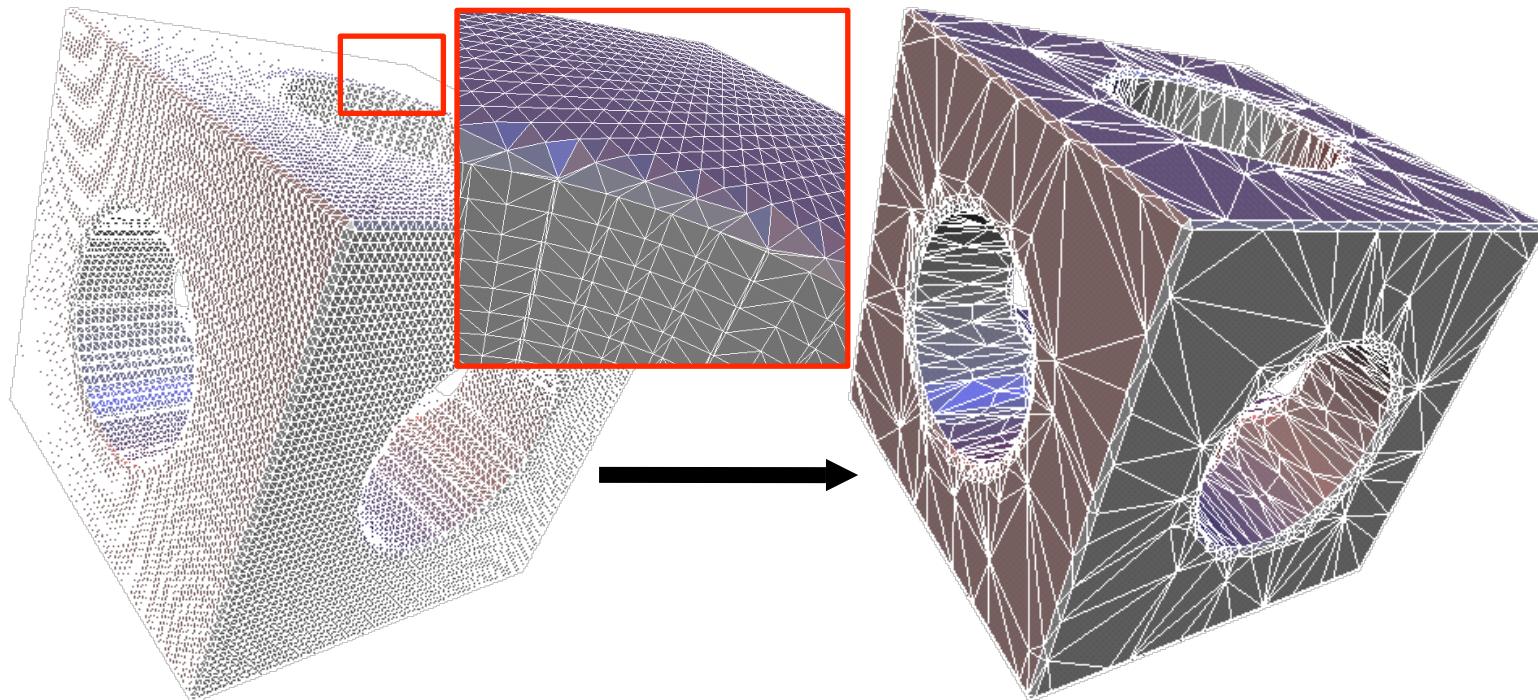
- Oversampled 3D scan data



Same shape can be well-approximated by many fewer triangles

Surface Meshes often Overtessellated

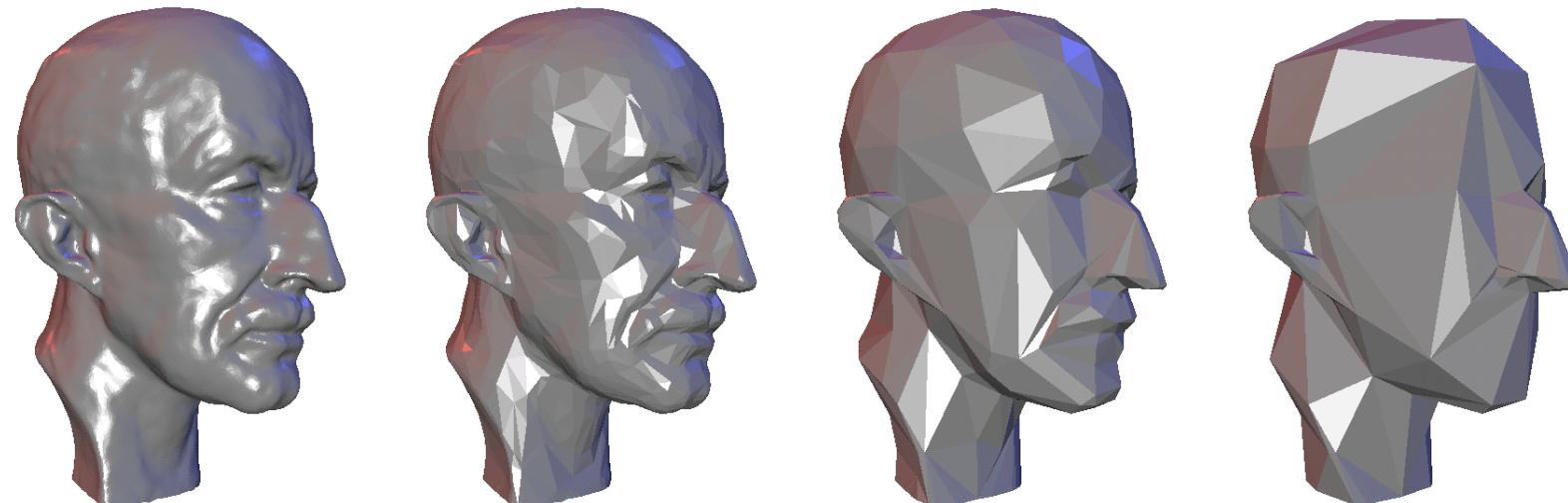
- Overtessellation: E.g. iso-surface extraction



- Large polygon counts can make applications non-performant
- Problematic for interactive visualization

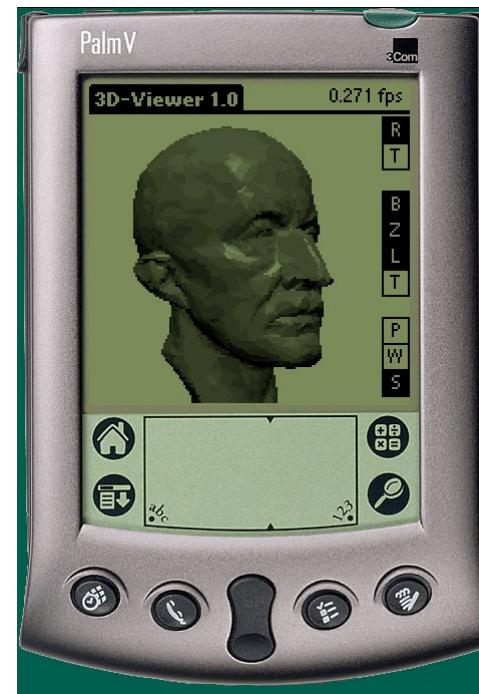
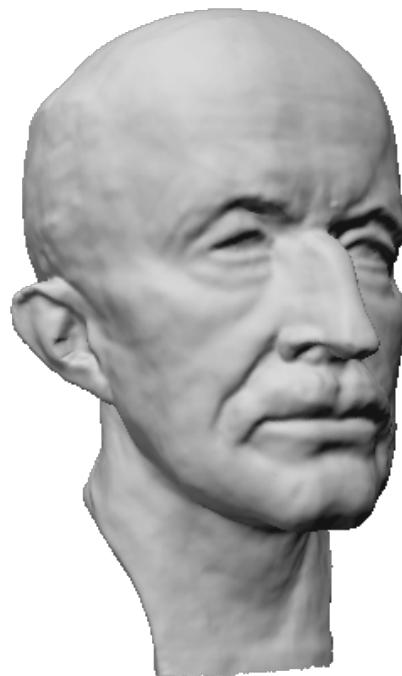
Multi-resolution Hierarchies

- Construct multiple versions of mesh
 - Varying polygon count
- Multi-resolution hierarchies enable
 - efficient geometry processing
 - level-of-detail (LOD) rendering

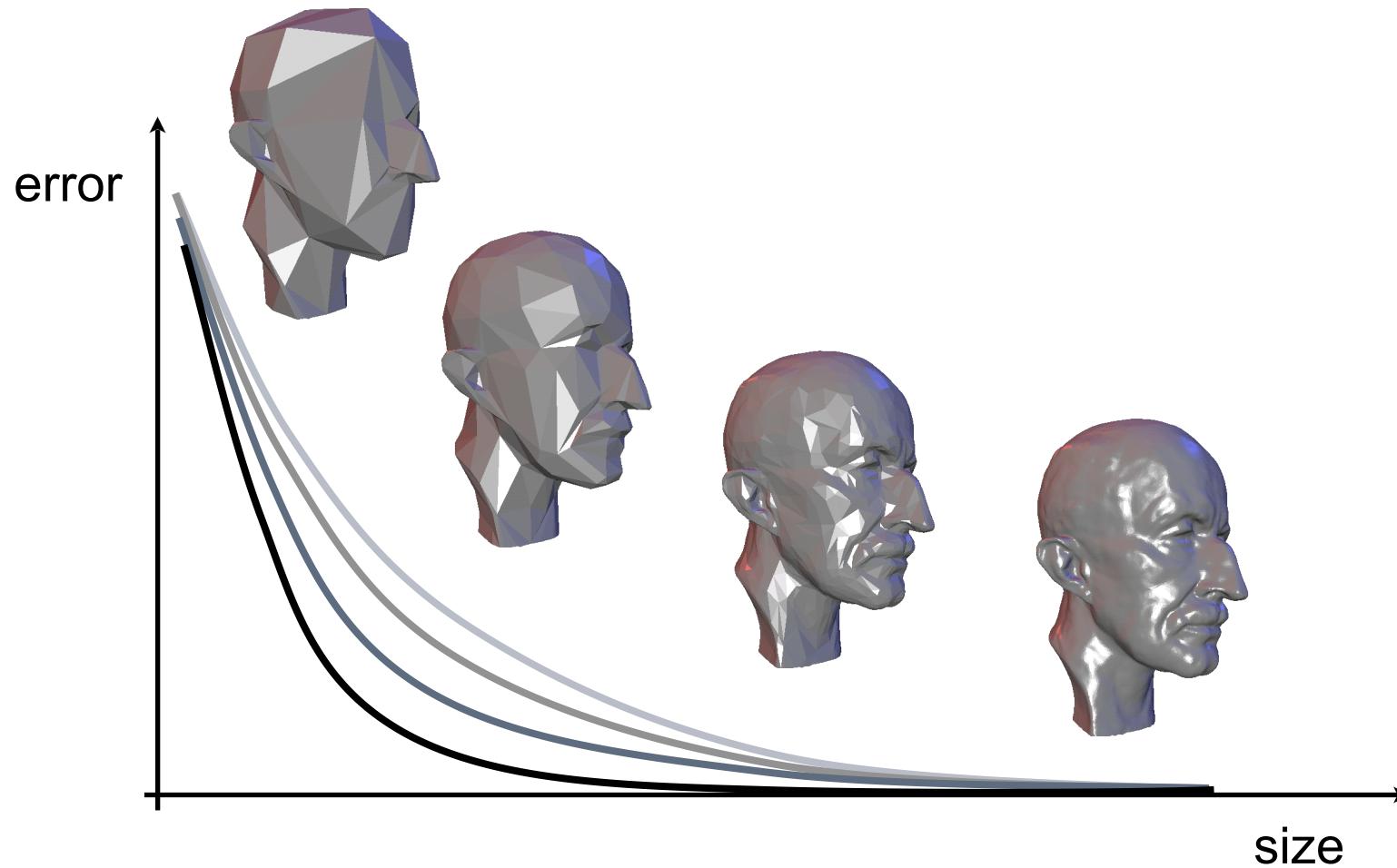


Applications

- Adaptation to hardware capabilities

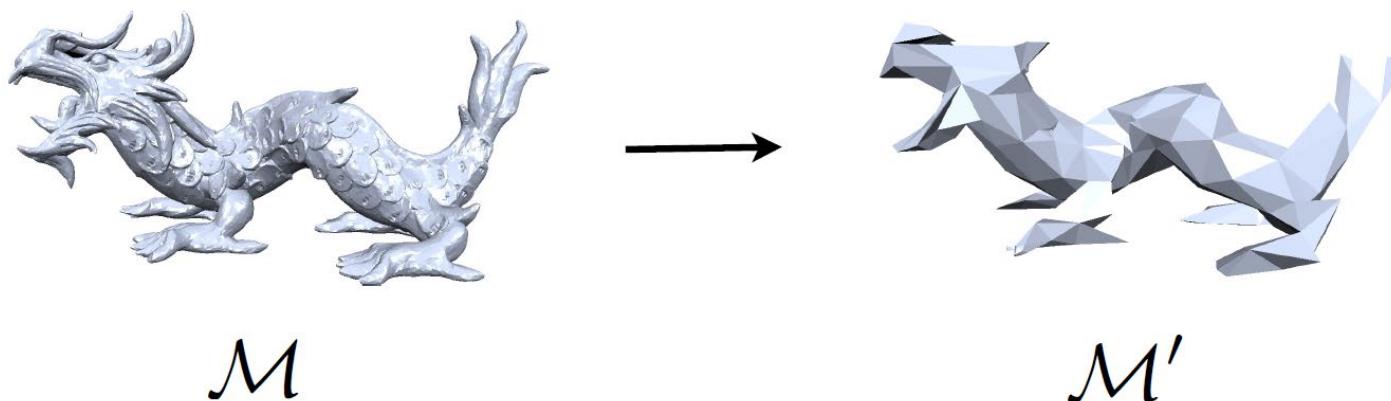


Size-Quality Tradeoff



Problem Statement

- Given: $\mathcal{M} = (\mathcal{V}, \mathcal{F})$
- Find: $\mathcal{M}' = (\mathcal{V}', \mathcal{F}')$ such that
 - 1. $|\mathcal{V}'| = n < |\mathcal{V}|$ and $\|\mathcal{M} - \mathcal{M}'\|$ is minimal, or
 - 2. $\|\mathcal{M} - \mathcal{M}'\| < \epsilon$ and $|\mathcal{V}'|$ is minimal



Problem Statement

- Given: $\mathcal{M} = (\mathcal{V}, \mathcal{F})$
- Find: $\mathcal{M}' = (\mathcal{V}', \mathcal{F}')$ such that
 1. $|\mathcal{V}'| = n < |\mathcal{V}|$ and $\|\mathcal{M} - \mathcal{M}'\|$ is minimal, or
 2. $\|\mathcal{M} - \mathcal{M}'\| < \epsilon$ and $|\mathcal{V}'|$ is minimal

hard!

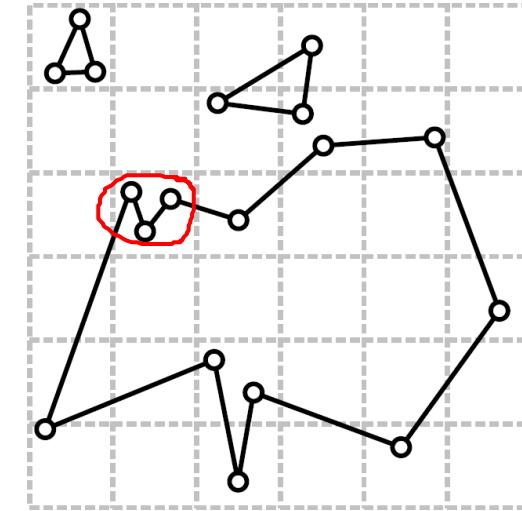
→ look for sub-optimal solution

Mesh Simplification: Vertex Clustering

- Cluster Generation
- Computing a representative
- Mesh generation
- Topology changes

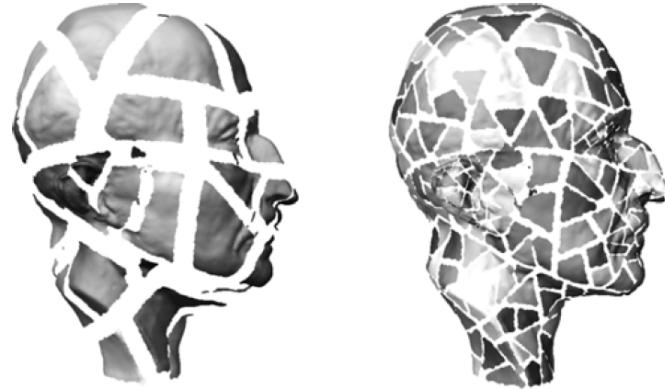
Vertex Clustering

- Cluster Generation
 - Uniform 3D grid
 - Map vertices to cluster cells
- Computing a representative
- Mesh generation
- Topology changes



Vertex Clustering

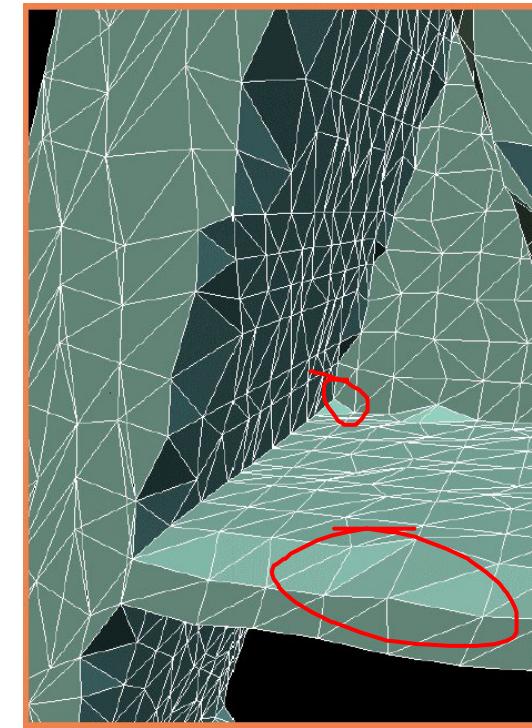
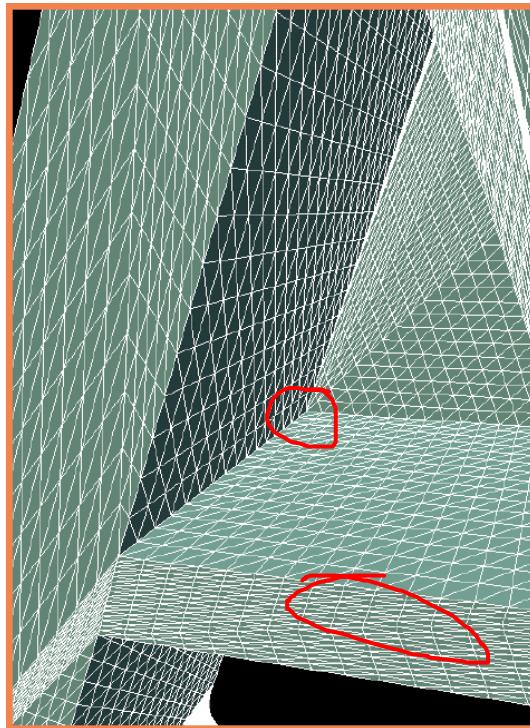
- Cluster Generation
 - Hierarchical approach
 - Top-down or bottom-up
- Computing a representative
- Mesh generation
- Topology changes



Vertex Clustering

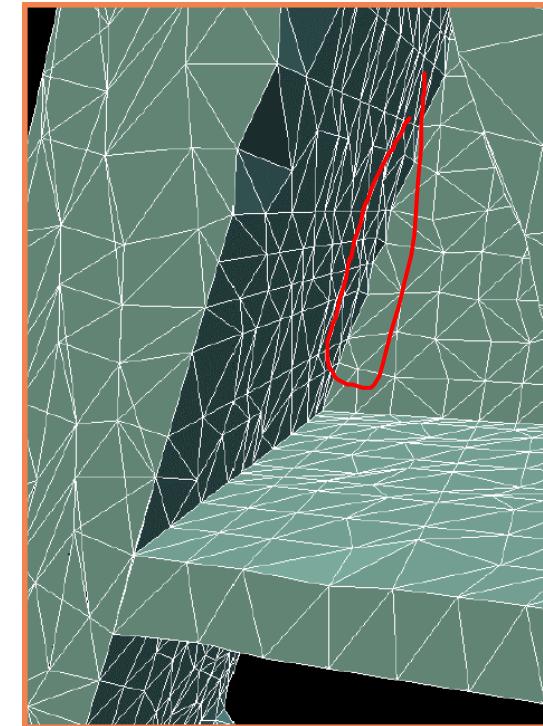
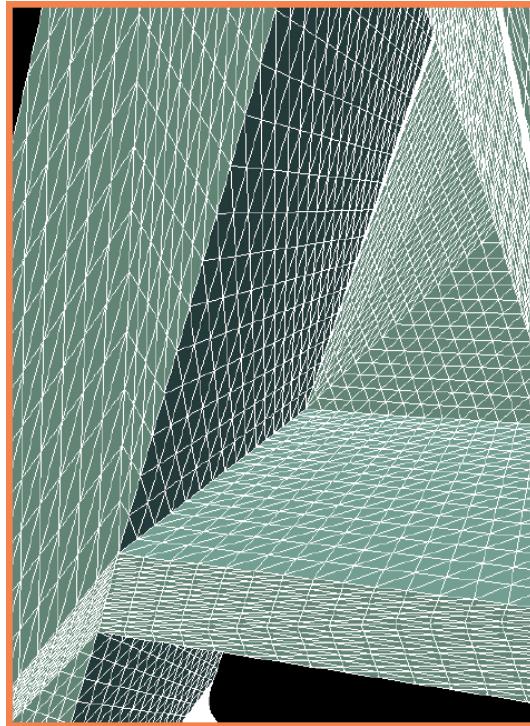
- Cluster Generation
- Computing a representative
 - Average/median vertex position
 - Error quadrics
- Mesh generation
- Topology changes

Computing a Representative



Average vertex position → Low-pass filter

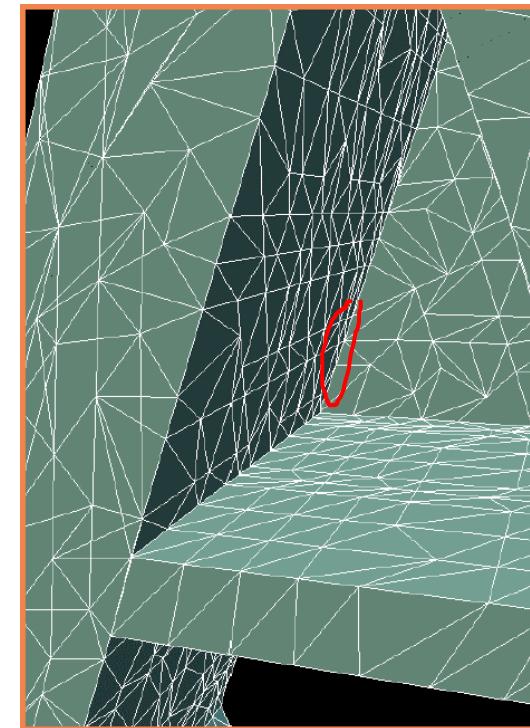
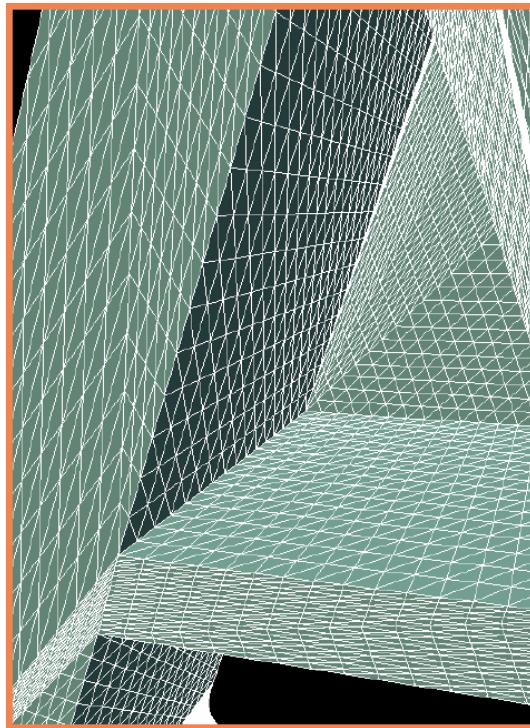
Computing a Representative



Median vertex will be a vertex of the original mesh closest to the average position of the vertices in the cluster.

Median vertex position → Sub-sampling

Computing a Representative



Error quadrics

Error Quadrics

- Squared distance to plane

$$p = (x, y, z, 1)^T, \quad q = (a, b, c, d)^T$$

$$\text{dist}(q, p)^2 = (q^T p)^2 = p^T (q q^T) p =: p^T Q_q p$$

$$Q_q = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & b^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

Using implicit
form of a
plane
equation
 $ax+by+cz+d=0$

Error Quadrics

- Sum distances to vertex' planes

$$\sum_i \text{dist}(q_i, p)^2 = \sum_i p^T Q_{q_i} p = p^T \left(\sum_i Q_{q_i} \right) p =: p^T Q_p p$$

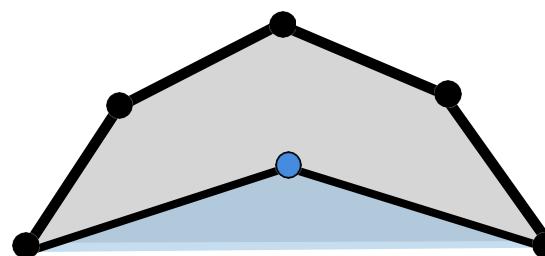
You can compute the sum of squared distances from p to N planes using a single 4×4 matrix

- Point that minimizes the error

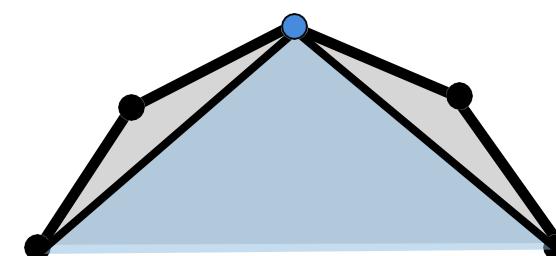
$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} p^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

You simply sum up the N matrices Q_{q_i} component-wise and use it as shown here.

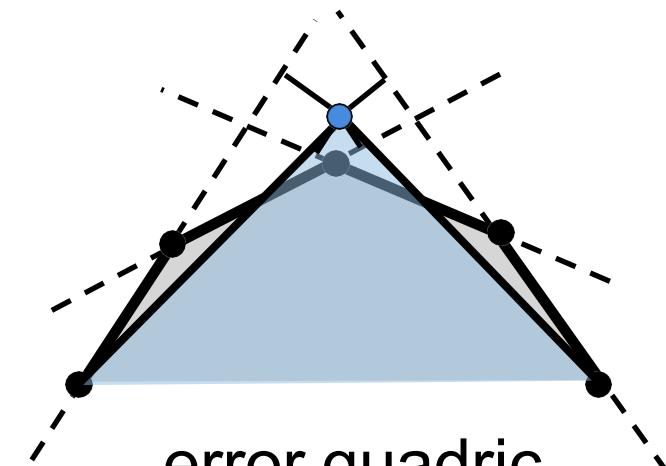
Comparison



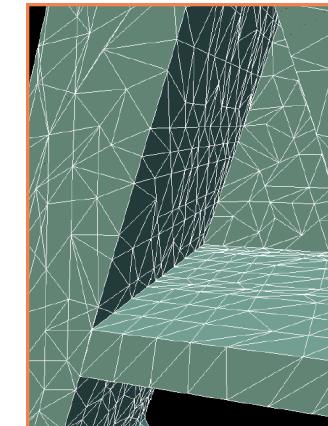
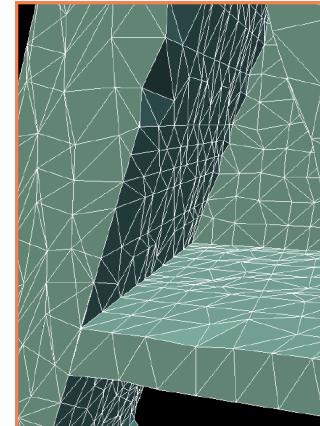
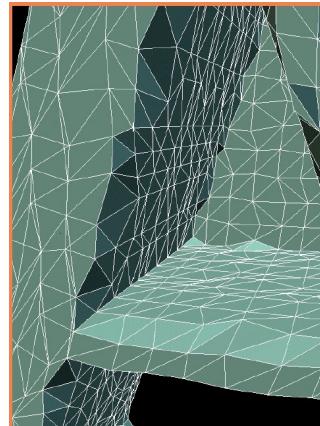
average



median



error quadric

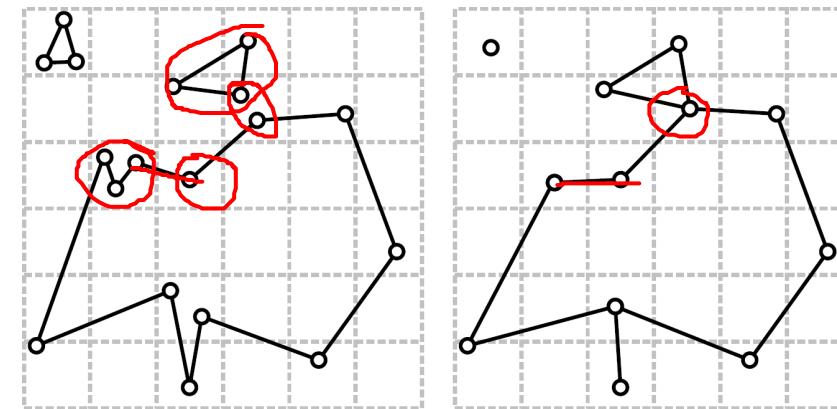
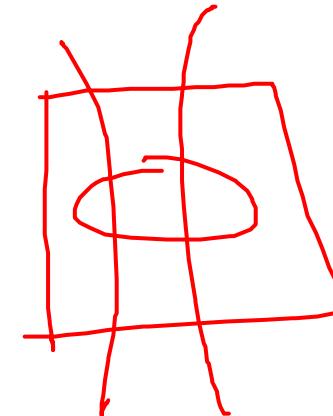


Vertex Clustering

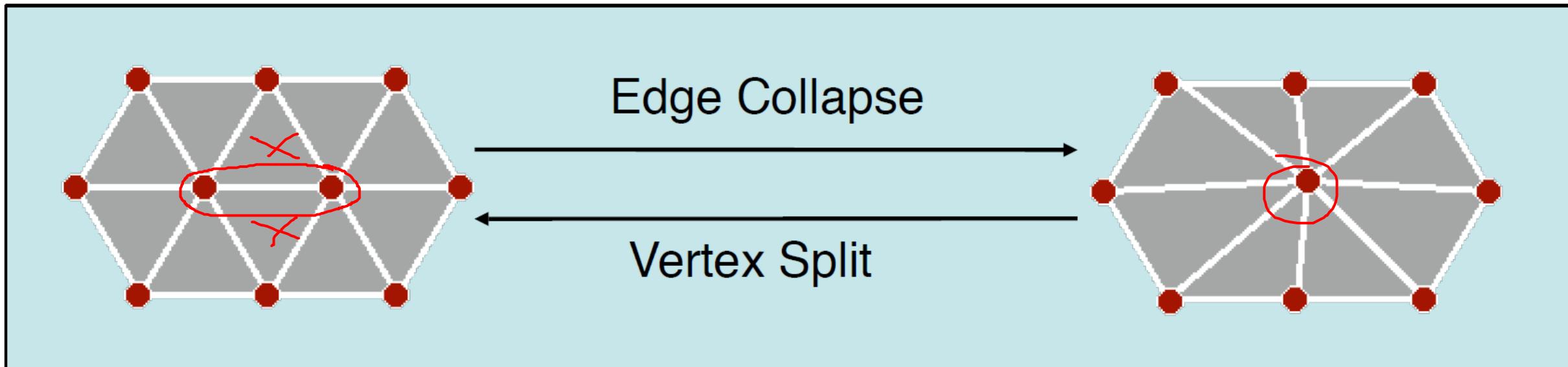
- Cluster Generation
- Computing a representative
- Mesh generation
 - Clusters $p \Leftrightarrow \{p_0, \dots, p_n\}$, $q \Leftrightarrow \{q_0, \dots, q_m\}$
 - Connect (p, q) if there was an edge (p_i, q_j)
- Topology changes

Vertex Clustering

- Cluster Generation
- Computing a representative
- Mesh generation
- Topology changes
 - If different sheets pass through one cell
 - Not manifold



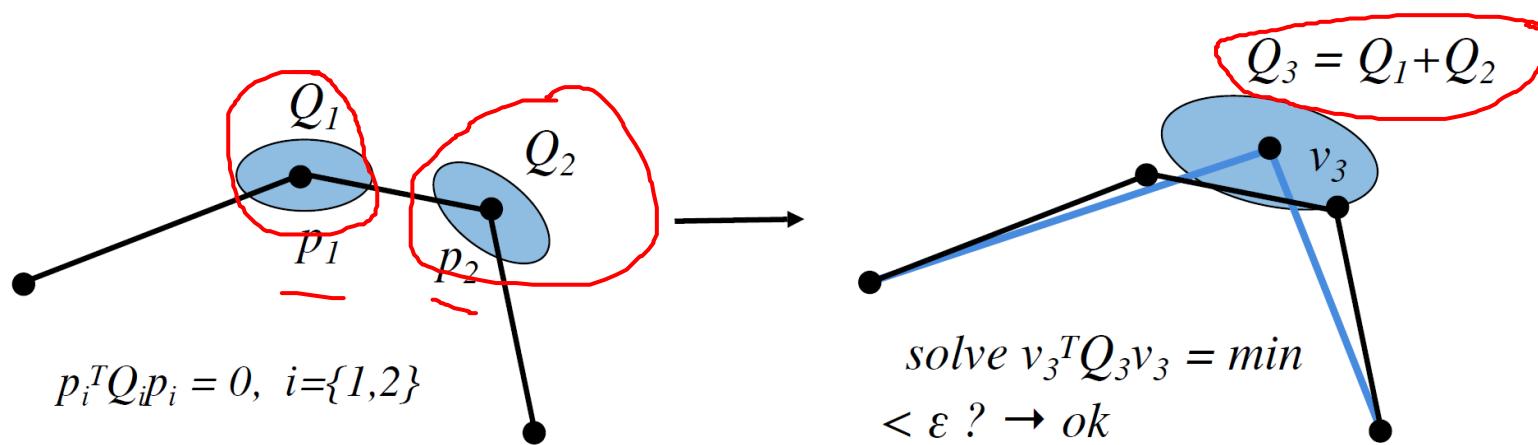
Incremental Simplification: Edge Collapse



- Merge two adjacent triangles
- Define new vertex position

Error Metrics

- Error quadrics [Garland, Heckbert 97]
 - Squared distance to planes at vertex
 - Can iteratively pick edge collapse that induces least error
 - Update error quadrics at each iteration



Comparison

- Vertex clustering
 - fast, but difficult to control simplified mesh
 - topology changes, non-manifold meshes
 - global error bound, but often not close to optimum
- ✓ • Iterative simplification with quadric error metrics
 - good trade-off between mesh quality and speed
 - explicit control over mesh topology
 - restricting normal deviation improves mesh quality

Out-of-core Decimation

- Handle very large data sets that do not fit into main memory
- Key: Avoid random access to mesh data structure during simplification
- Examples
 - Garland, Shaffer: *A Multiphase Approach to Efficient Surface Simplification*, IEEE Visualization 2002
 - Wu, Kobbelt: *A Stream Algorithm for the Decimation of Massive Meshes*, Graphics Interface 2003

Multiphase Simplification

1. Phase: Out-of-core clustering

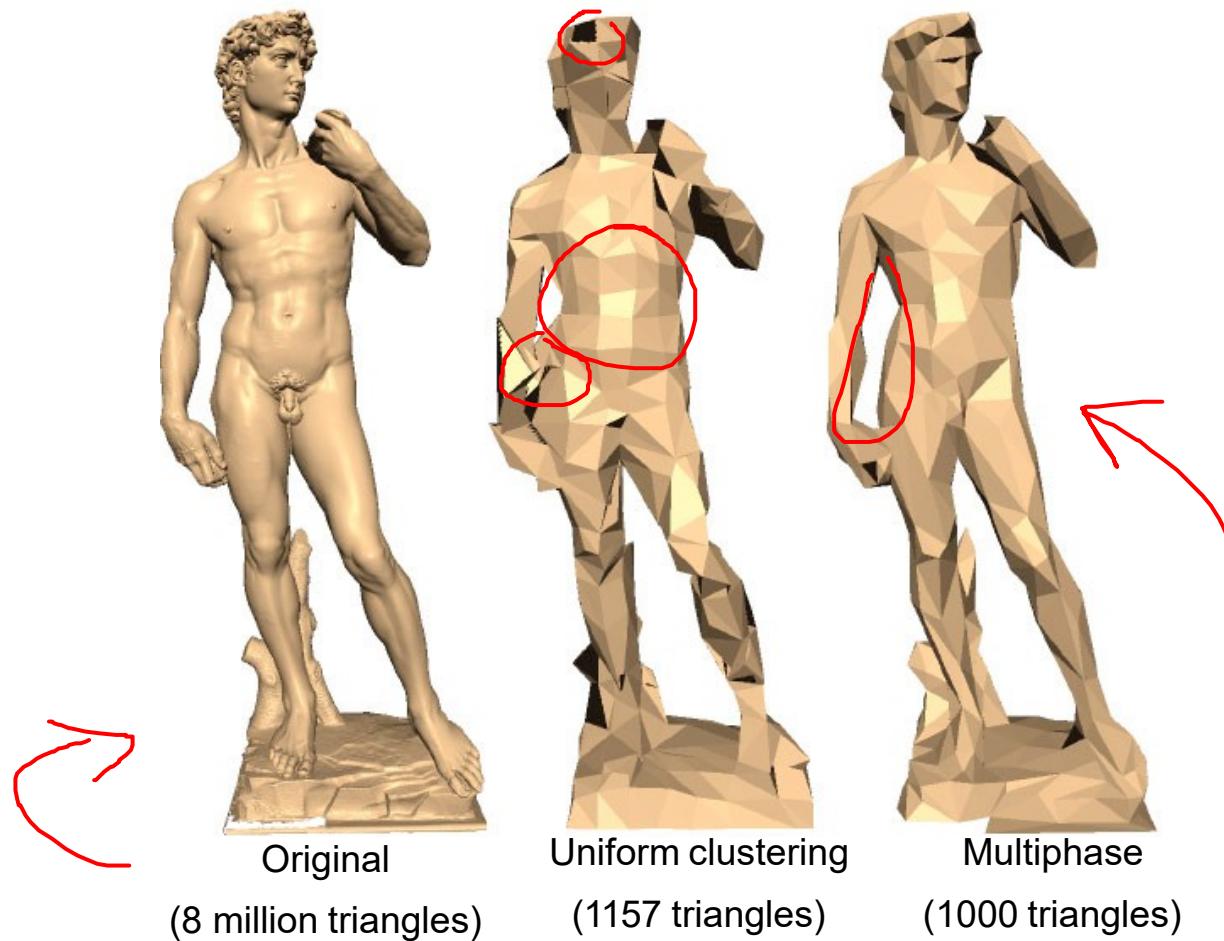
- compute accumulated error quadrics and vertex representative for each cell of uniform voxel grid

2. Phase: In-core iterative simplification

- use accumulated quadrics from clustering phase
- iteratively contract edge of smallest cost

→ achieves a coupling between the two phases

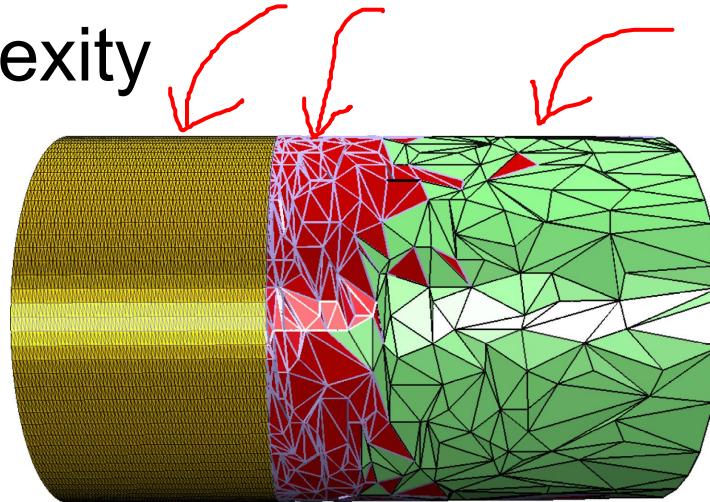
Multiphase Simplification



Garland, Shaffer: *A Multiphase Approach to Efficient Surface Simplification*, IEEE Visualization 2002

Out-of-core Decimation

- Streaming approach based on edge collapse operations using QEM
- Pre-sorted input stream allows fixed-sized active working set independent of input and output model complexity



See also
Martin Isenburg, Peter Lindstrom:
Streaming Meshes.
IEEE Visualization 2005