THE UNIVERSITY of TEXAS
HEALTH SCIENCE CENTER AT HOUSTON
SCHOOL of HEALTH INFORMATION SCIENCES

Interpolation and Morphing

For students of HI 5323
"Image Processing"

Willy Wriggers, Ph.D.
School of Health Information Sciences

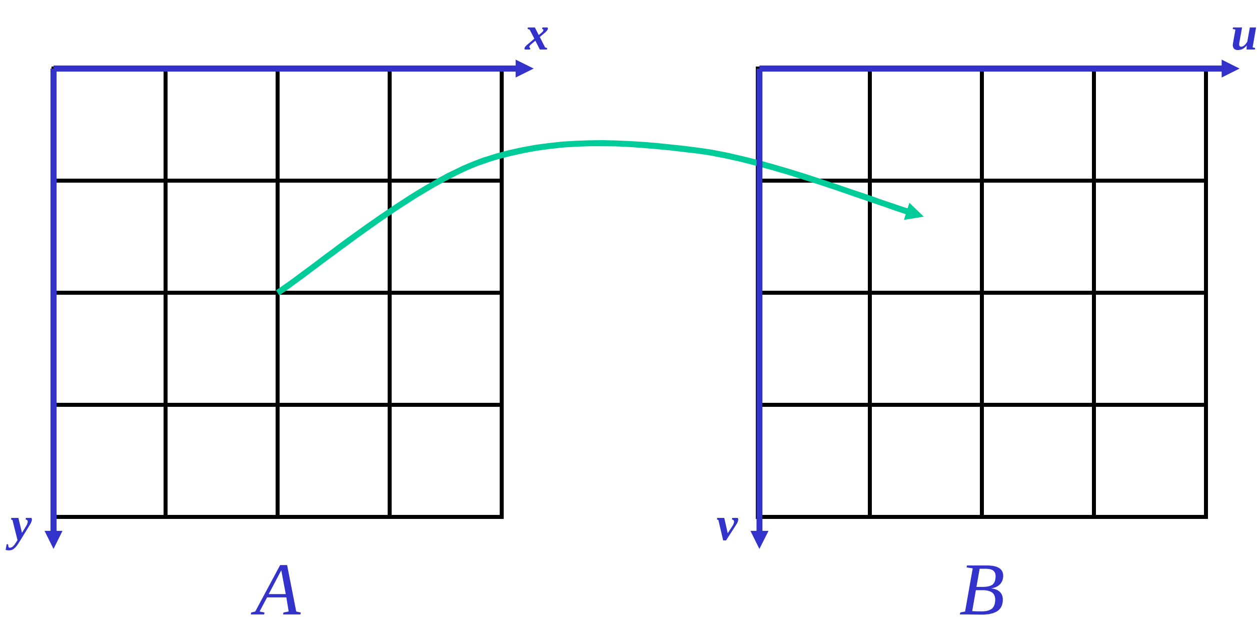
http://biomachina.org/courses/processing/05.html

Interpolation

Forward Mapping

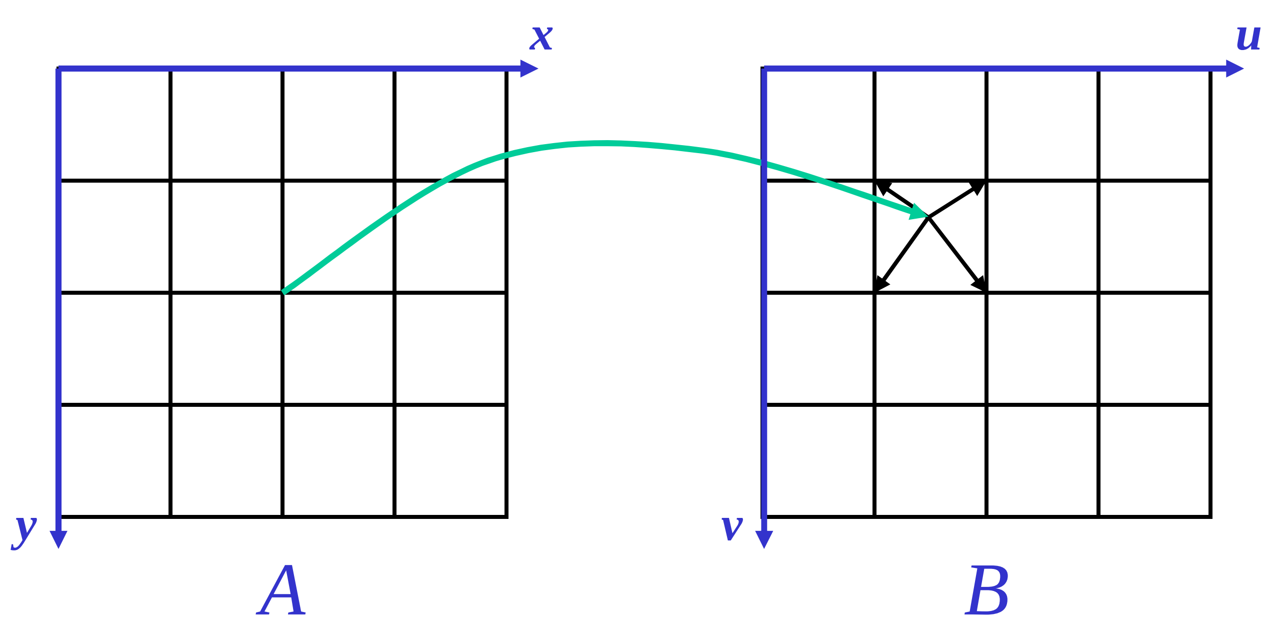
Let $u(x, y)$ and $v(x, y)$ be a mapping from location (x, y) to (u, v):

$$B[u(x, y), v(x, y)] = A[x, y]$$



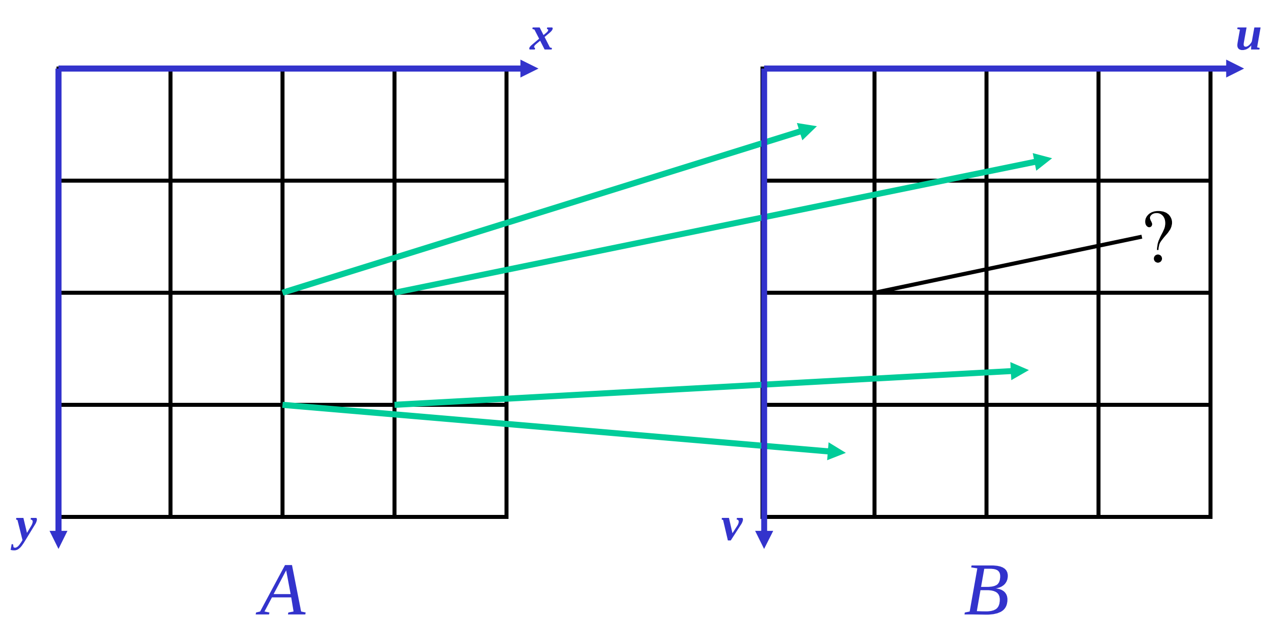
Forward Mapping: Problems

- Doesn't always map *to* pixel locations
- Solution: spread out effect of each pixel, e.g. by bilinear interpolation



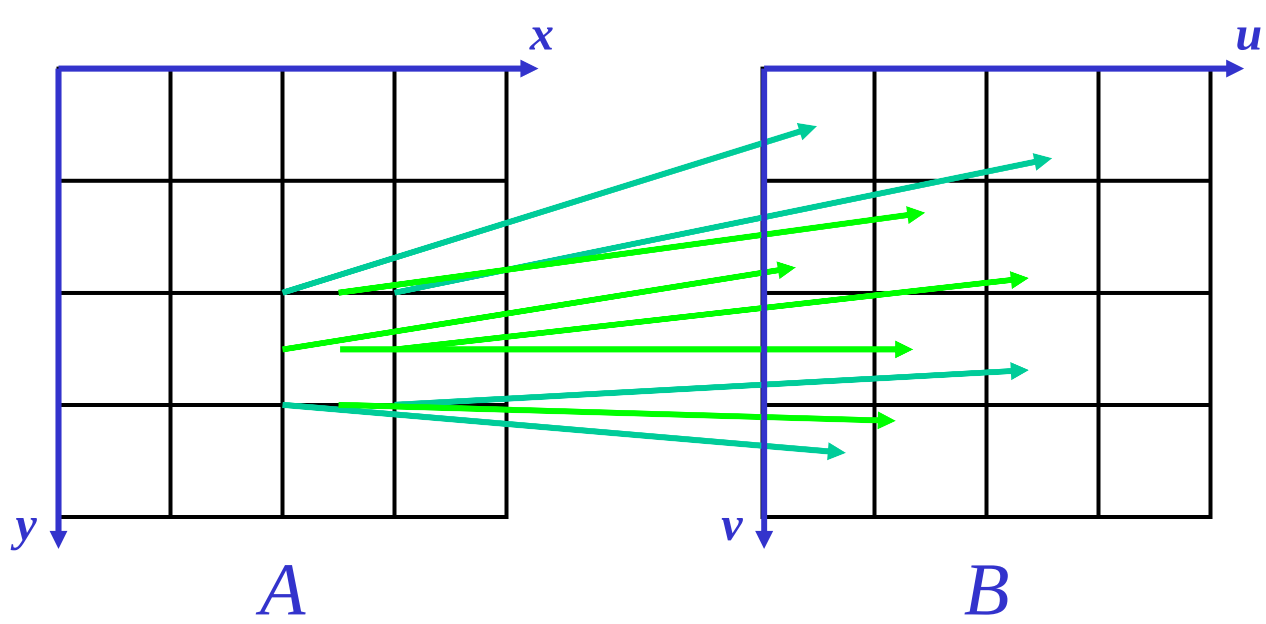
Forward Mapping: Problems

- May produce holes in the output



Forward Mapping: Problems

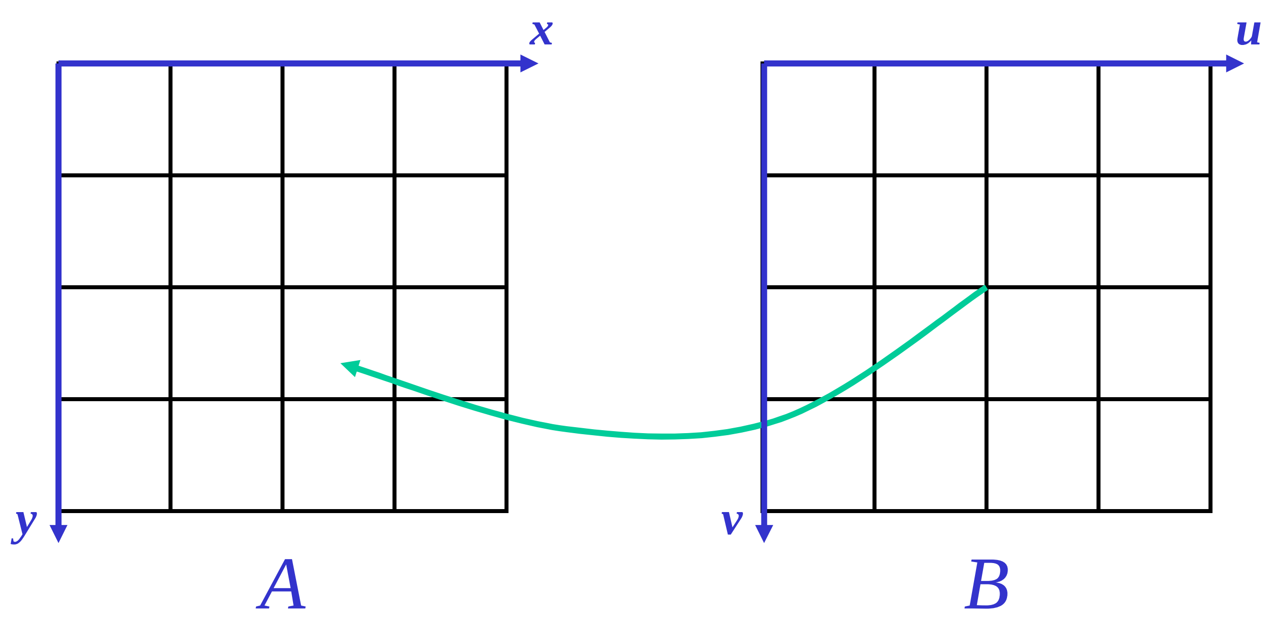
- May produce holes in the output
- Solution: sample source image (*A*) more often
  - Still can leave holes



Backward Mapping

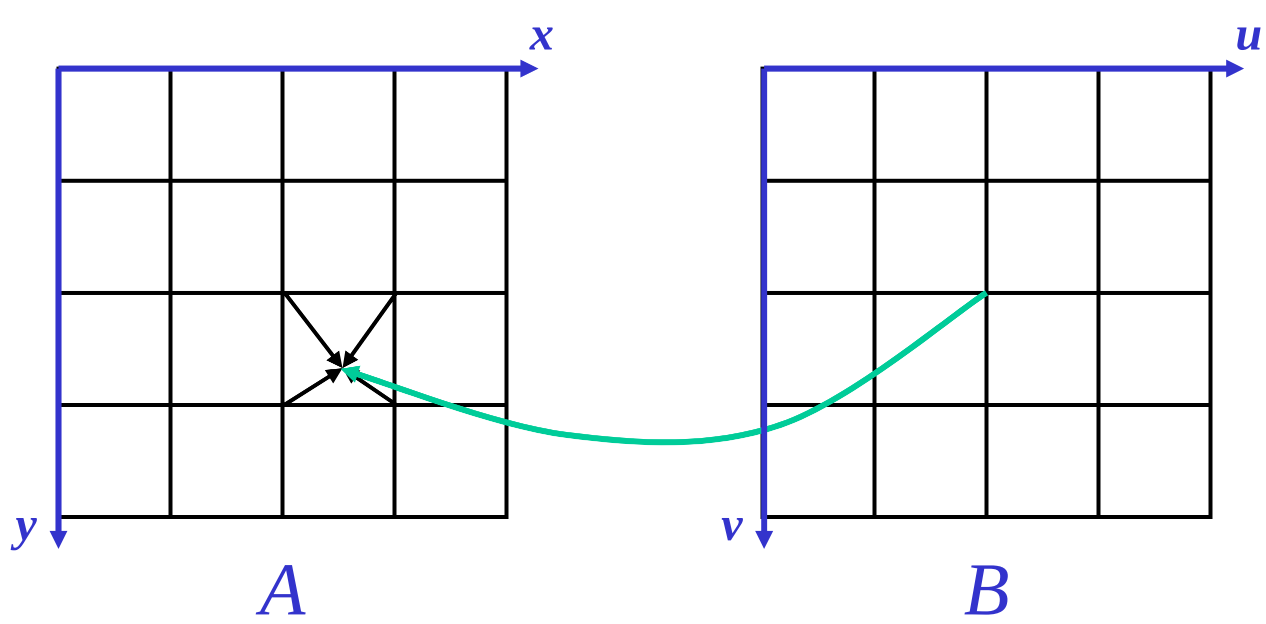
Let $x(u, v)$ and $y(u, v)$ be an inverse mapping from location (x, y) to (u, v):

$$B[u, v] = A[x(u, v), y(u, v)]$$



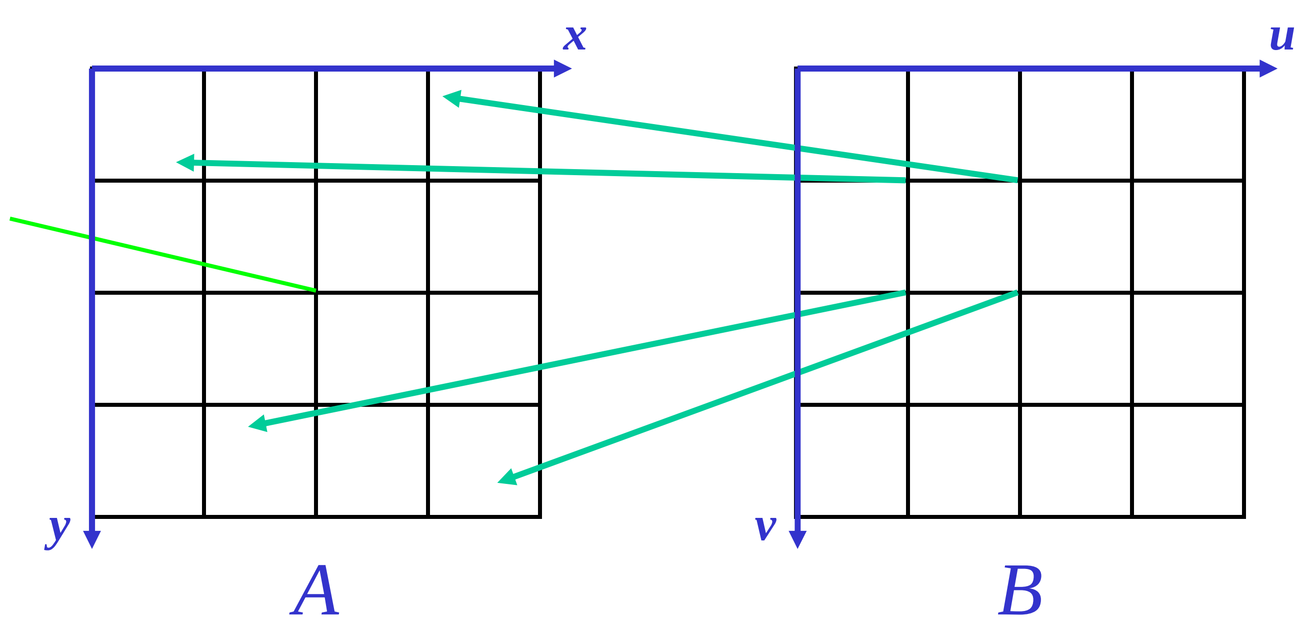
Backward Mapping: Problems

- Doesn't always map *from* a pixel
- Solution: Interpolate between pixels



Backward Mapping: Problems

- May produce holes in the input
- Solution: reduce input image (by averaging pixels) and sample reduced/averaged image → MIP-maps

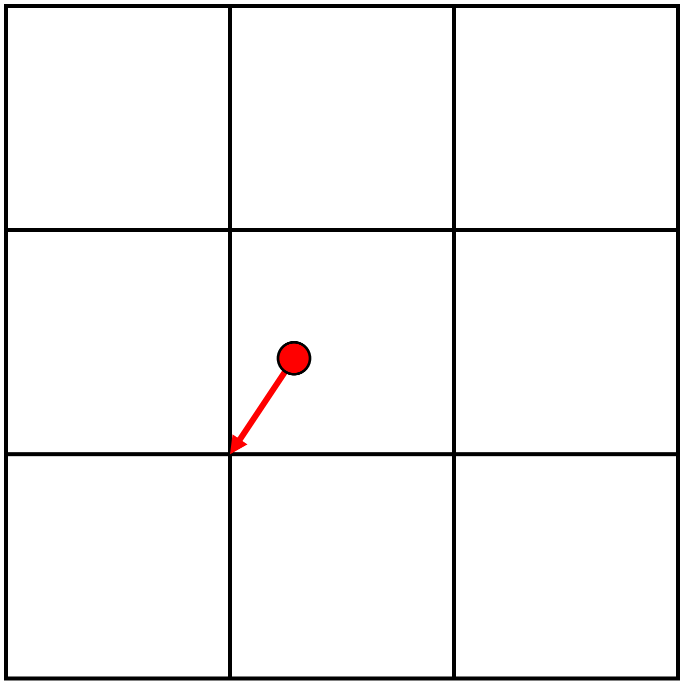


Interpolation

- “Filling In” between the pixels
- A function of the neighbors or a larger neighborhood
- Methods:
  - Nearest neighbor
  - Bilinear
  - Bicubic or other higher-order

Interpolation: Nearest-Neighbor

- Simplest to implement: the output pixel is assigned the value of the pixel that the point falls within
- Round off *x* and *y* values to nearest pixel
- Result is not continuous (blocky)

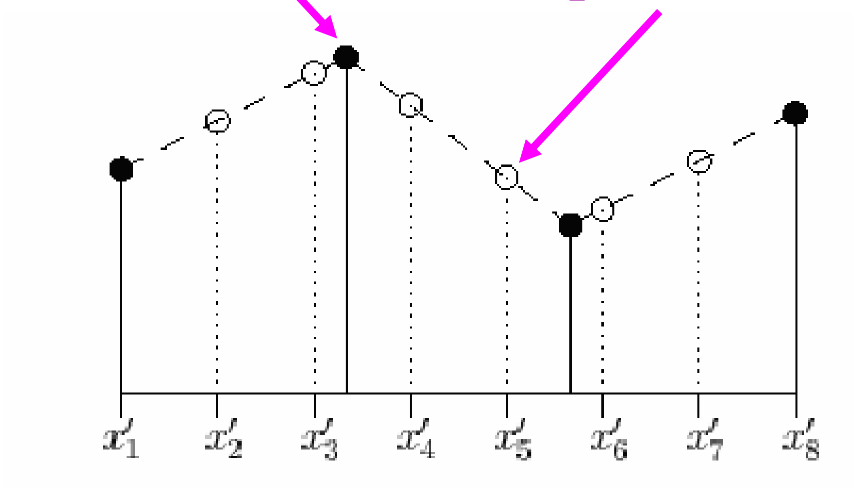


Interpolation: Linear (1D)

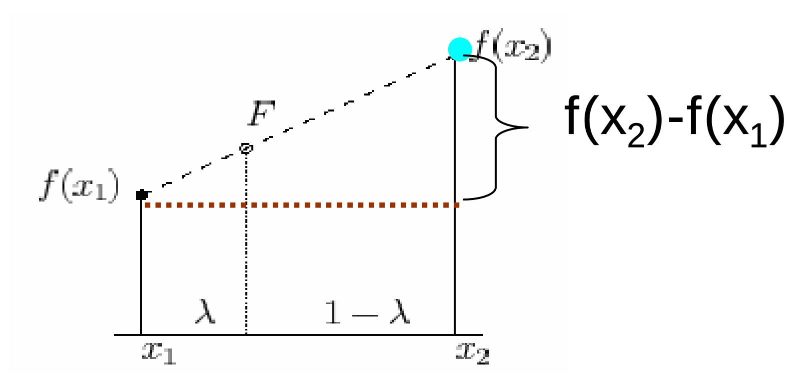
- General idea:

original function values

interpolated values



To calculate the interpolated values



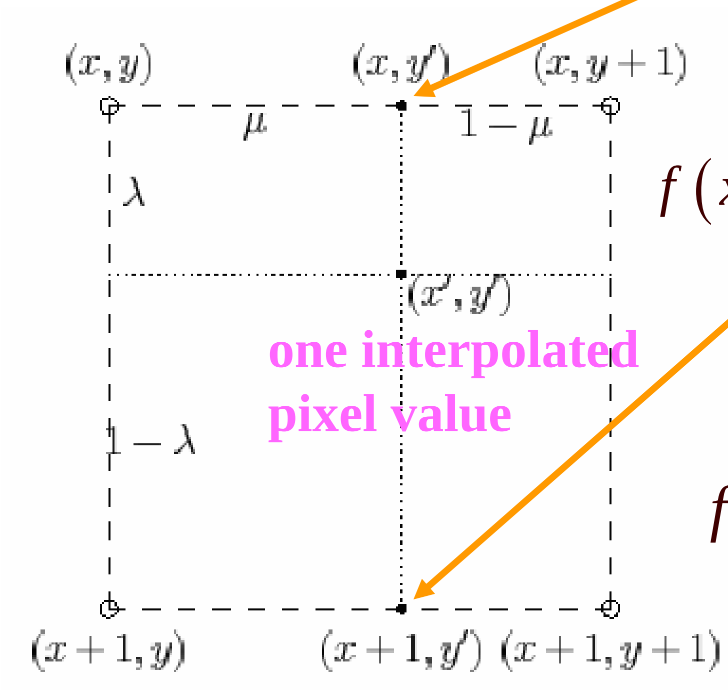
$$\frac{F - f(x_1)}{\lambda} = \frac{f(x_2) - f(x_1)}{1}.$$

Interpolation: Linear (2D)

- How a 4x4 image would be interpolated to produce an 8x8 image?

$$f\left(x, y'\right) = \mu f\left(x, y+1\right) + \left(1-\mu\right) f\left(x, y\right)$$

4 original pixel values



$$f\left(x+1, y'\right) = \mu f\left(x+1, y+1\right) + \left(1-\mu\right) f\left(x+1, y\right)$$

Along the y' column we have

$$f\left(x', y'\right) = \lambda f\left(x+1, y'\right) + (1-\lambda) f(x, y')$$

Bilinear Interpolation

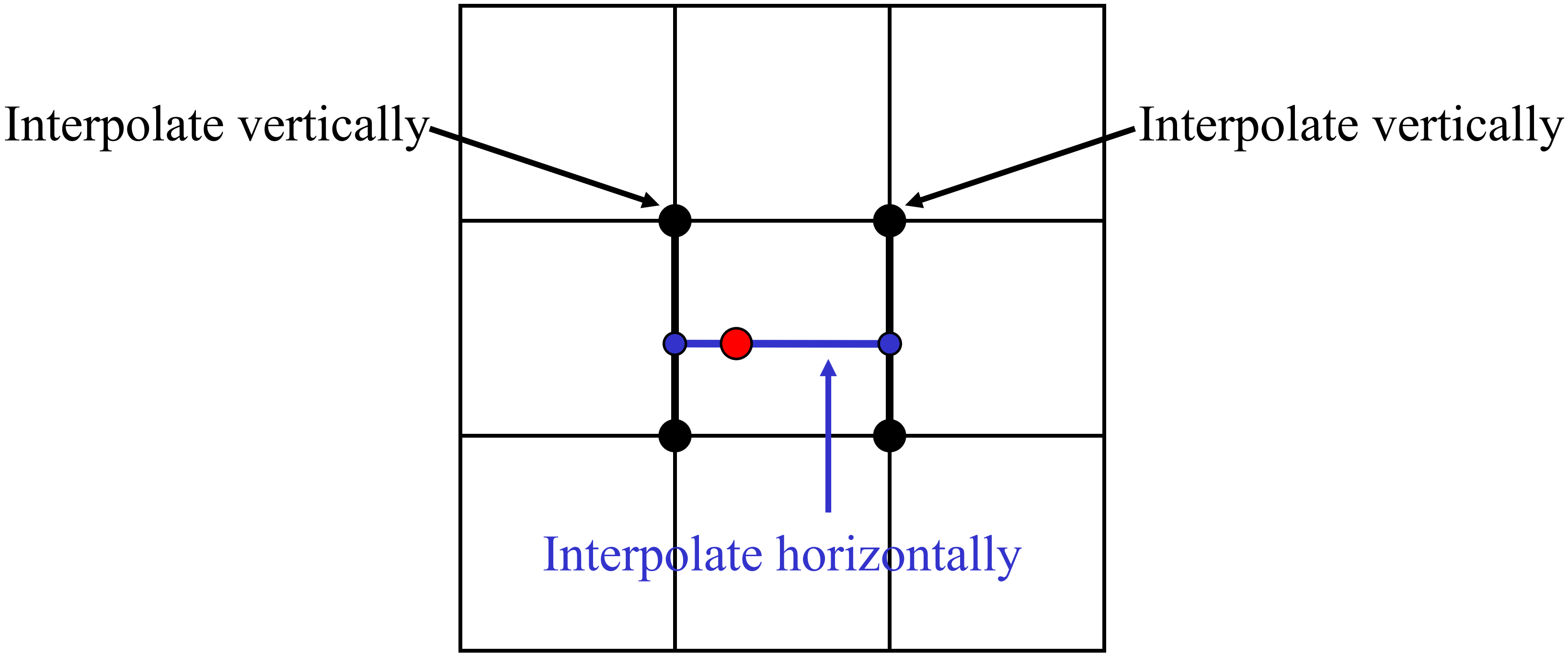
- Substituting with the values just obtained:

$$f(x',y') = \lambda\big(\mu f(x+1,y+1) + (1-\mu) f(x+1,y)\big) \\ + (1-\lambda)\big(\mu f(x,y+1) + (1-\mu) f(x,y)\big)$$

- You can do the expansion as an exercise.
- This is the formulation for bilinear interpolation

Bilinear Interpolation

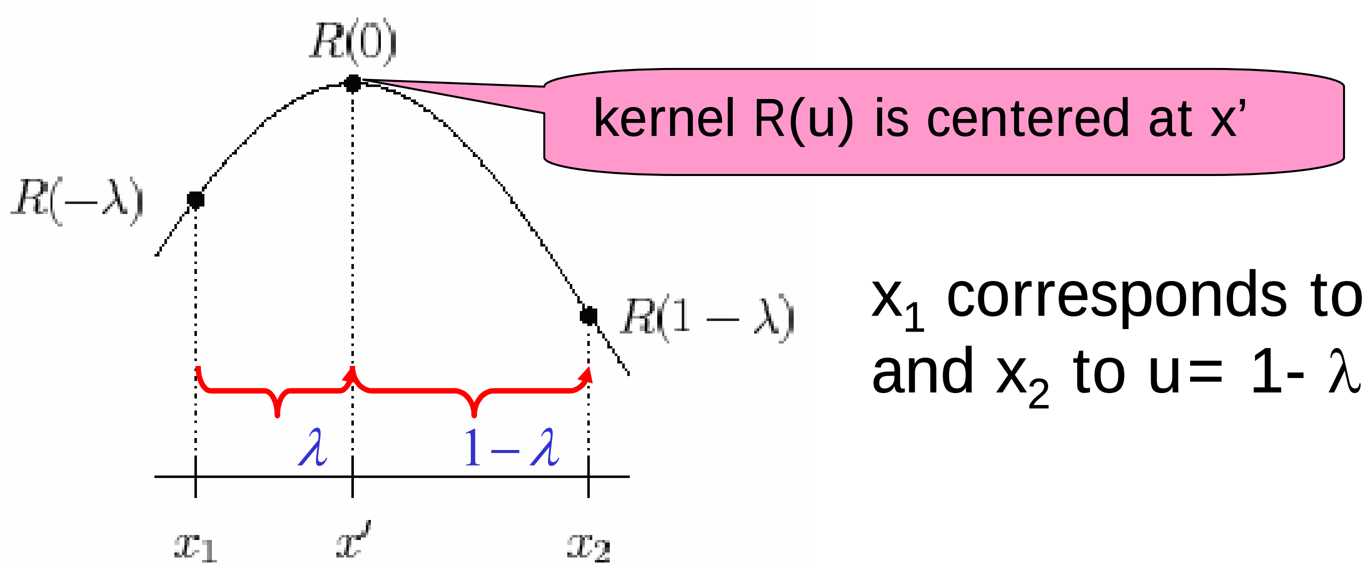
- The output pixel value is a weighted average of pixels in the nearest 2-by-2 neighborhood
- Linearly interpolate in one direction (e.g., vertically)
- Linearly interpolate results in the other direction (horizontally)



General Interpolation

- We wish to interpolate a value f(x') for $x_1 \le x' \le x_2$ and suppose $x' - x_1 = \lambda$
- We define an interpolation kernel R(u) and set

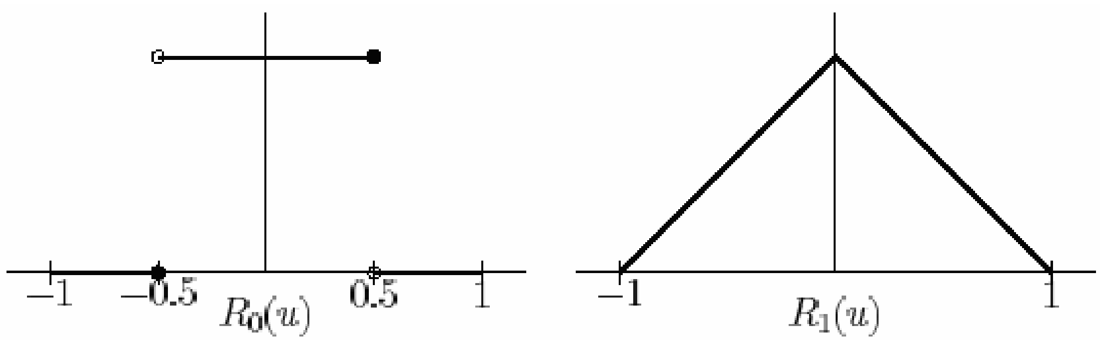
$$f(x') = R(-\lambda) f(x_1) + R(1-\lambda) f(x_2)$$



x_1 corresponds to u= - λ, and x_2 to u= 1- λ

General Interpolation: 0[th] and 1[st] orders

- Consider 2 functions $R_0(u)$ and $R_1(u)$



$$R_0(u) = \begin{cases} 0 & \text{if } u \le -0.5 \\ 1 & \text{if } -0.5 < u \le 0.5 \\ 0 & \text{if } u > 0.5 \end{cases}$$

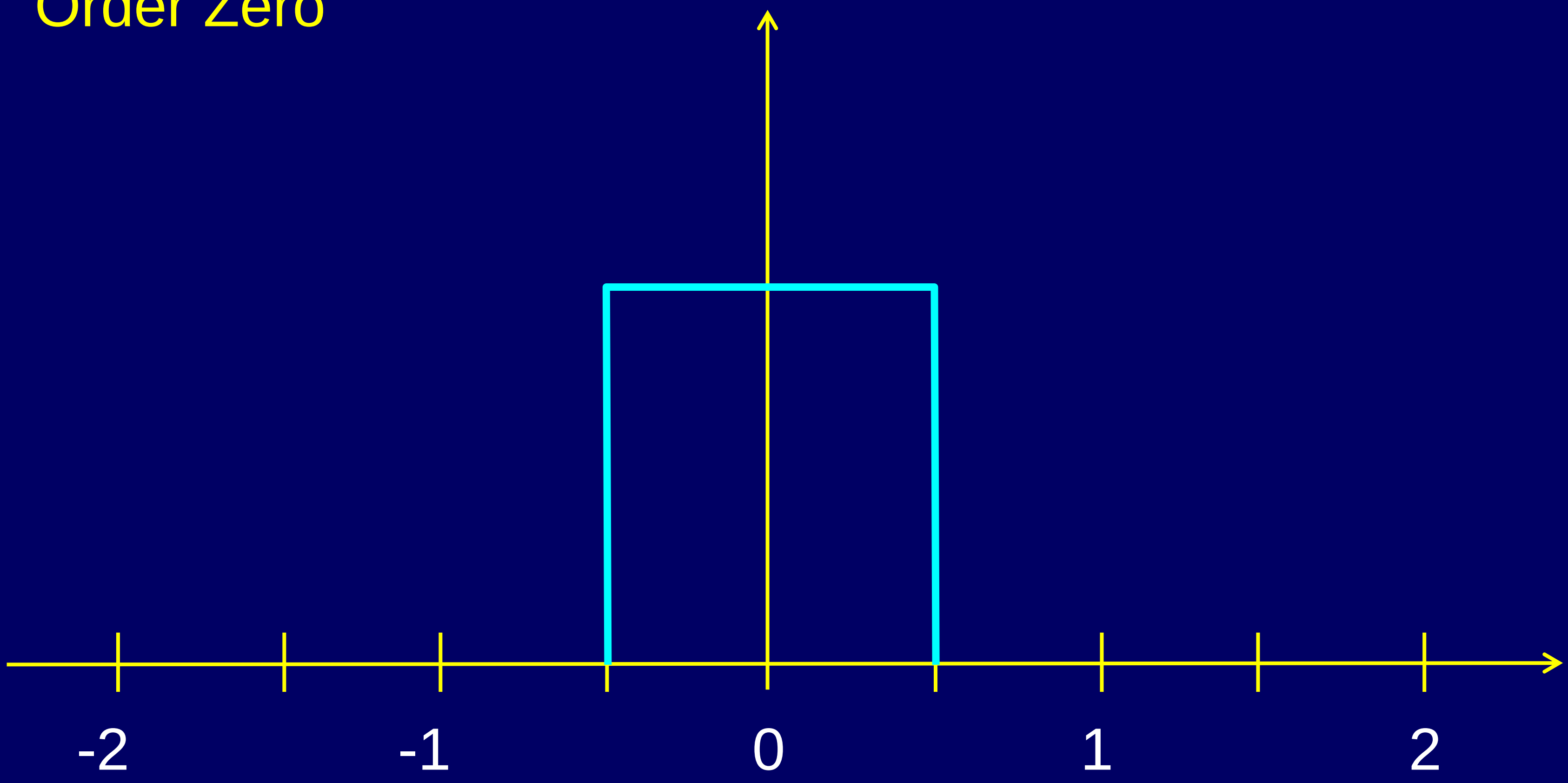
$$R_1(u) = \begin{cases} 1+u & \text{if } u \le 0 \\ 1-u & \text{if } u \ge 0 \end{cases}$$

Substitute $R_0(u)$ for $R(u)$ ⇨ nearest-neighbor interpolation.

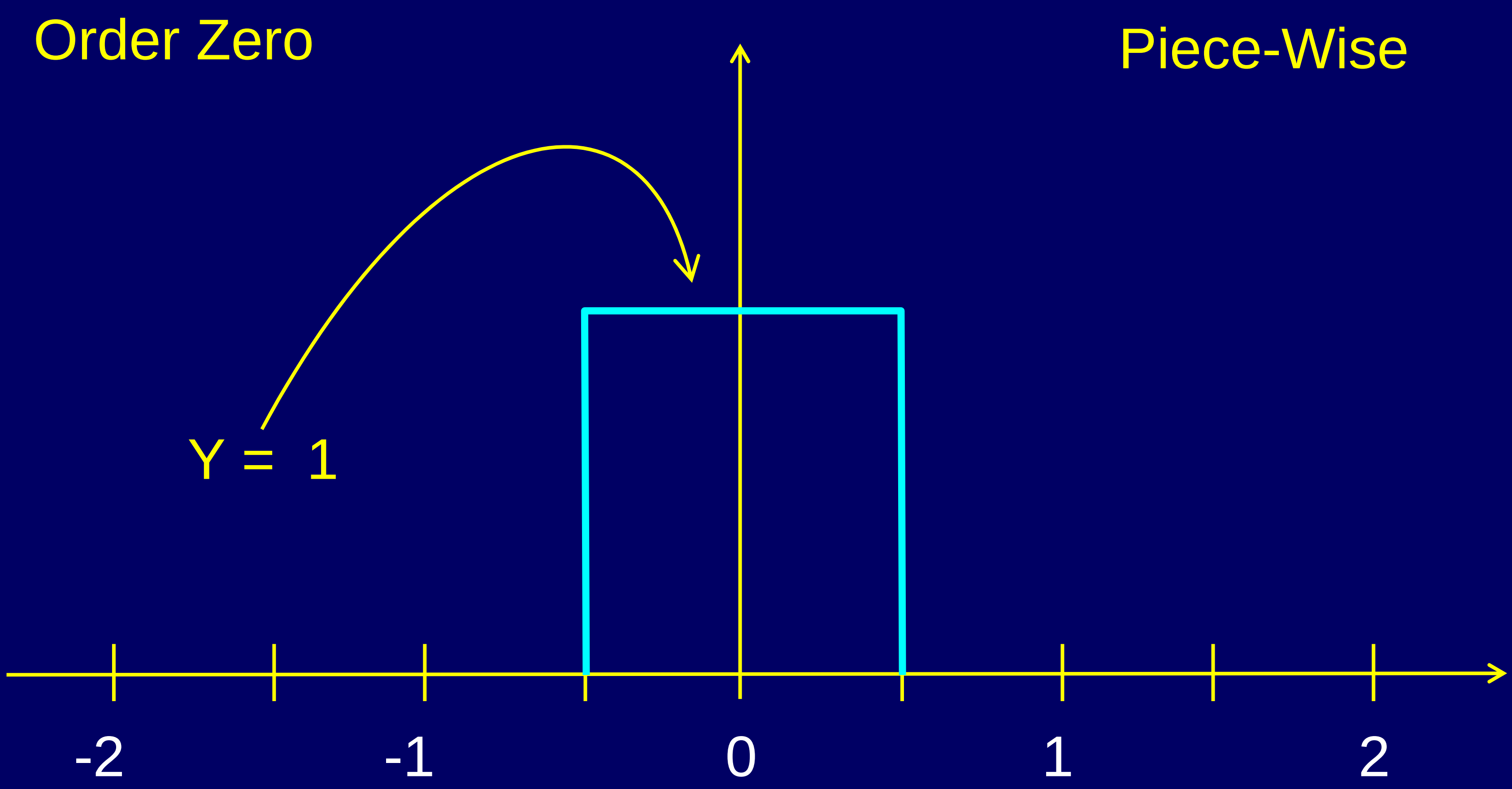
Substitute $R_1(u)$ for $R(u)$ ⇨ linear interpolation.

Interpolation Kernel

Order Zero

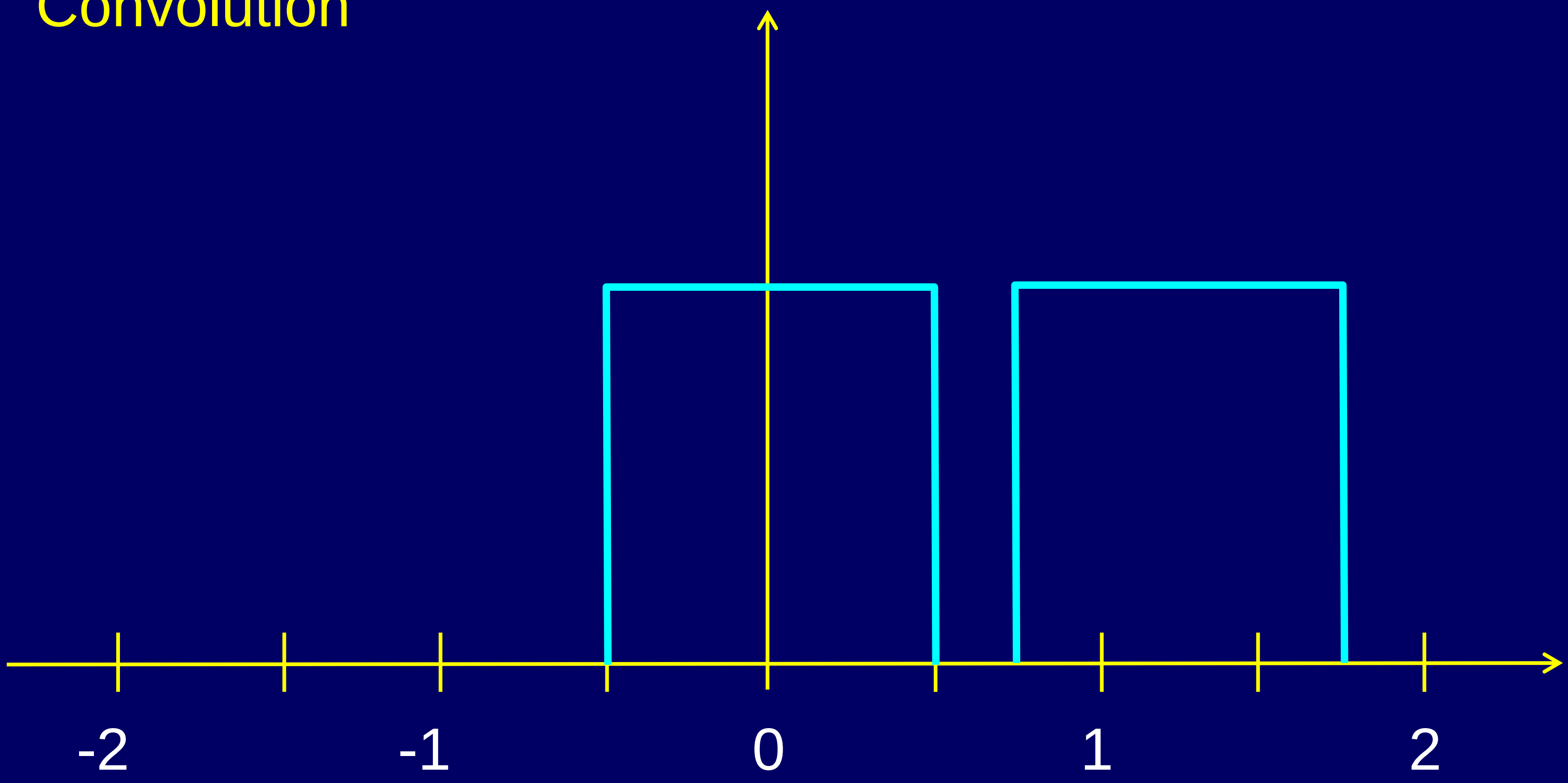


Interpolation Kernel



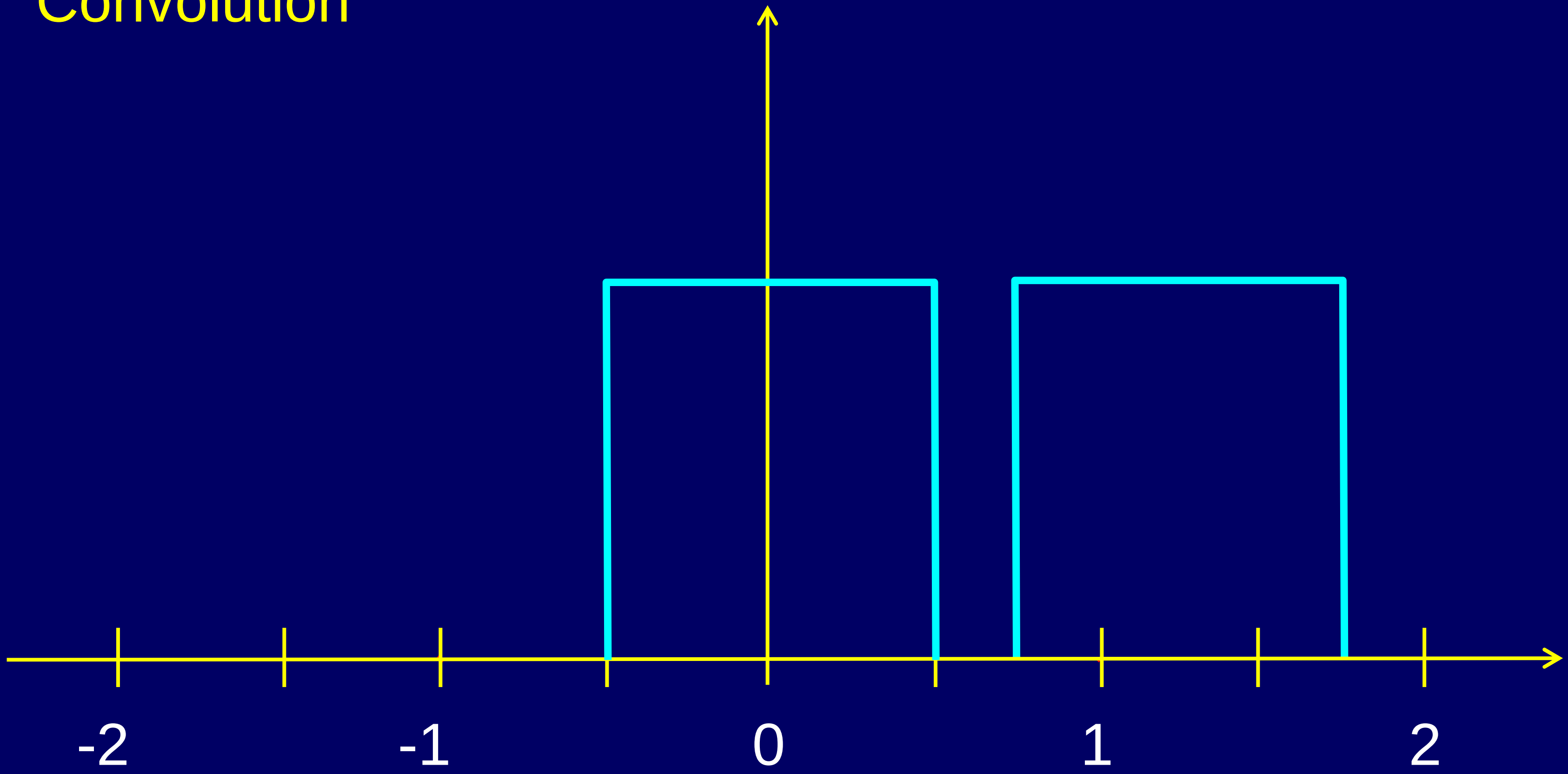
Interpolation Kernel

Convolution

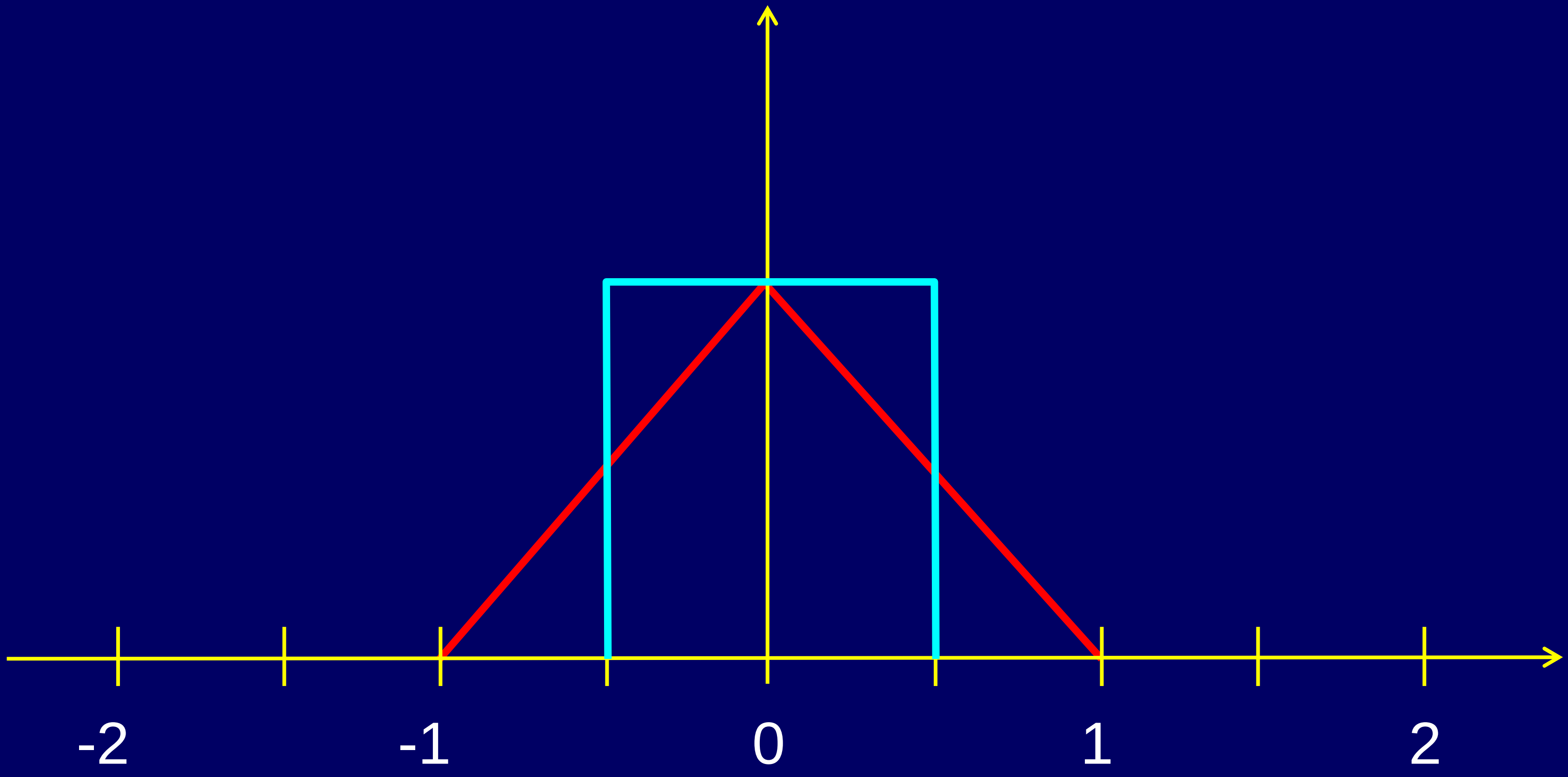


Interpolation Kernel

Convolution

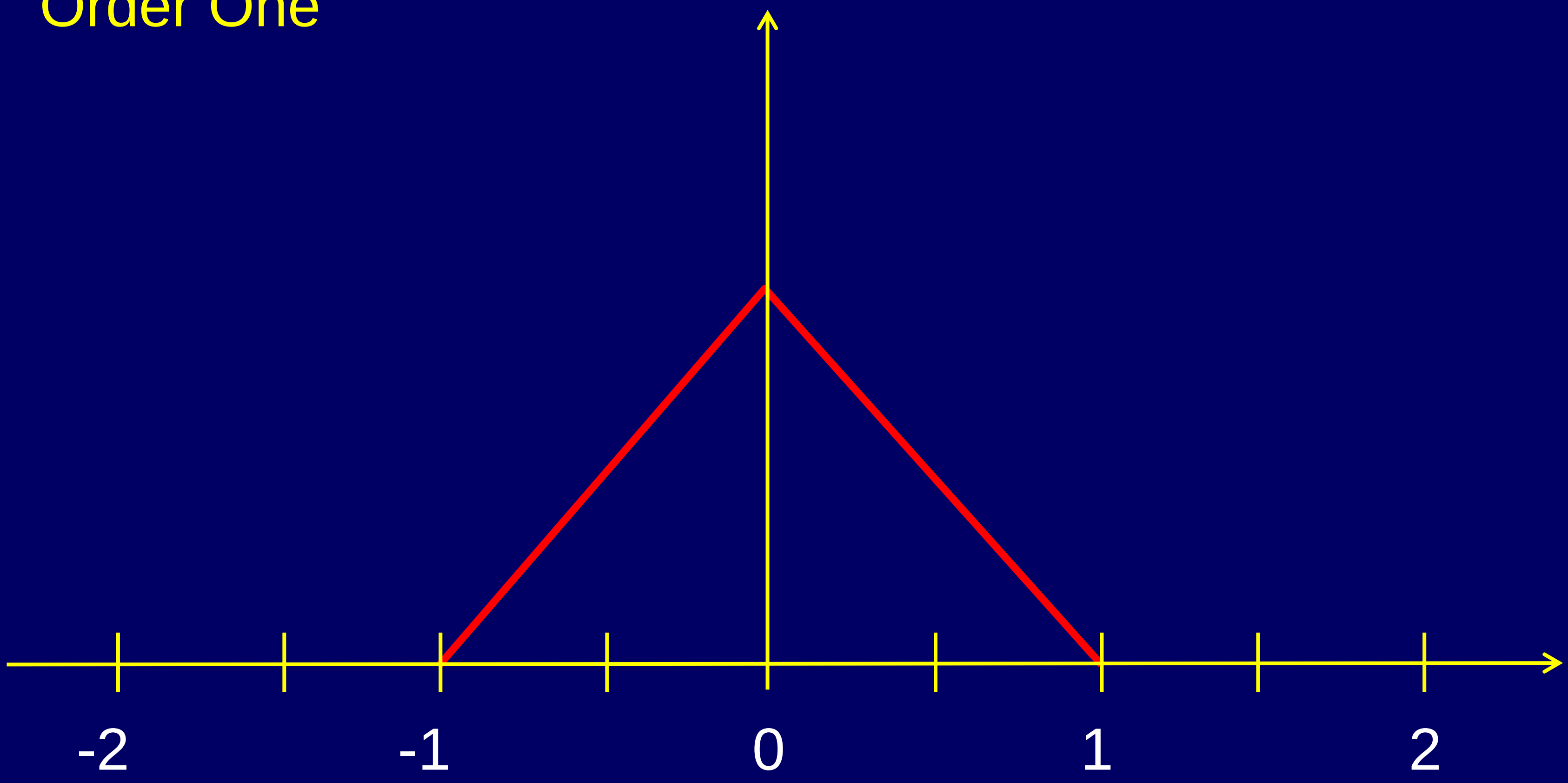


Interpolation Kernel

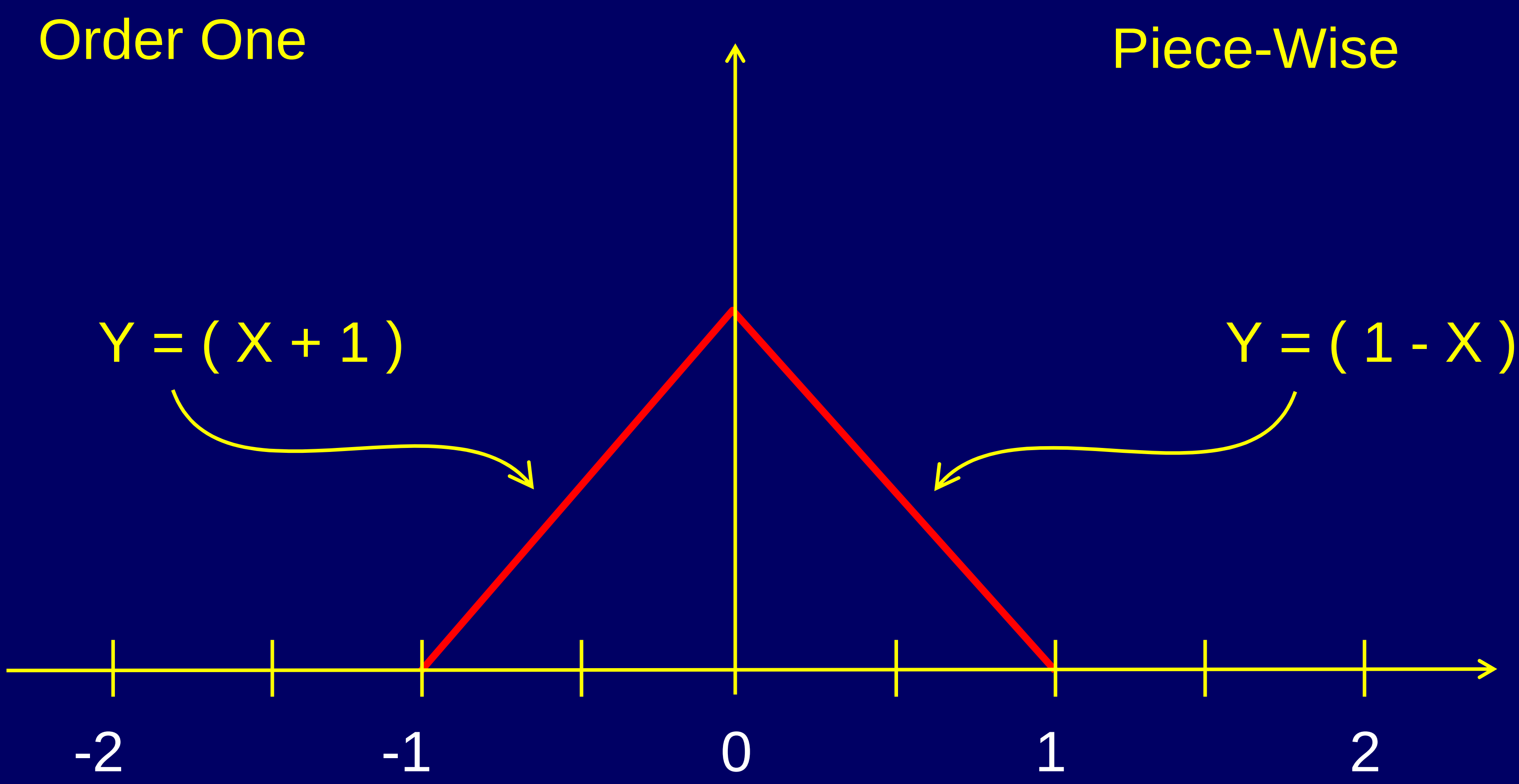


Interpolation Kernel

Order One

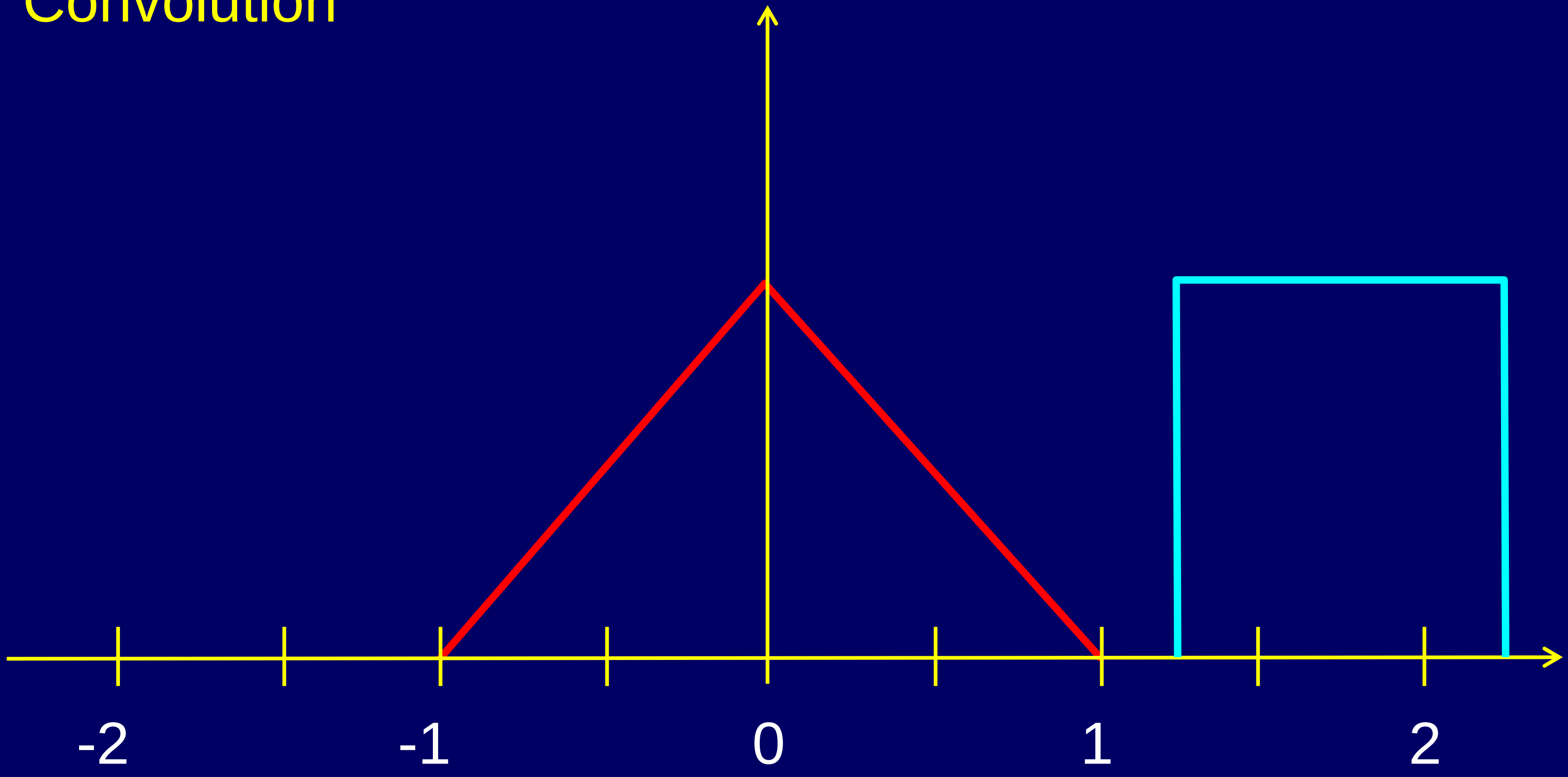


Interpolation Kernel



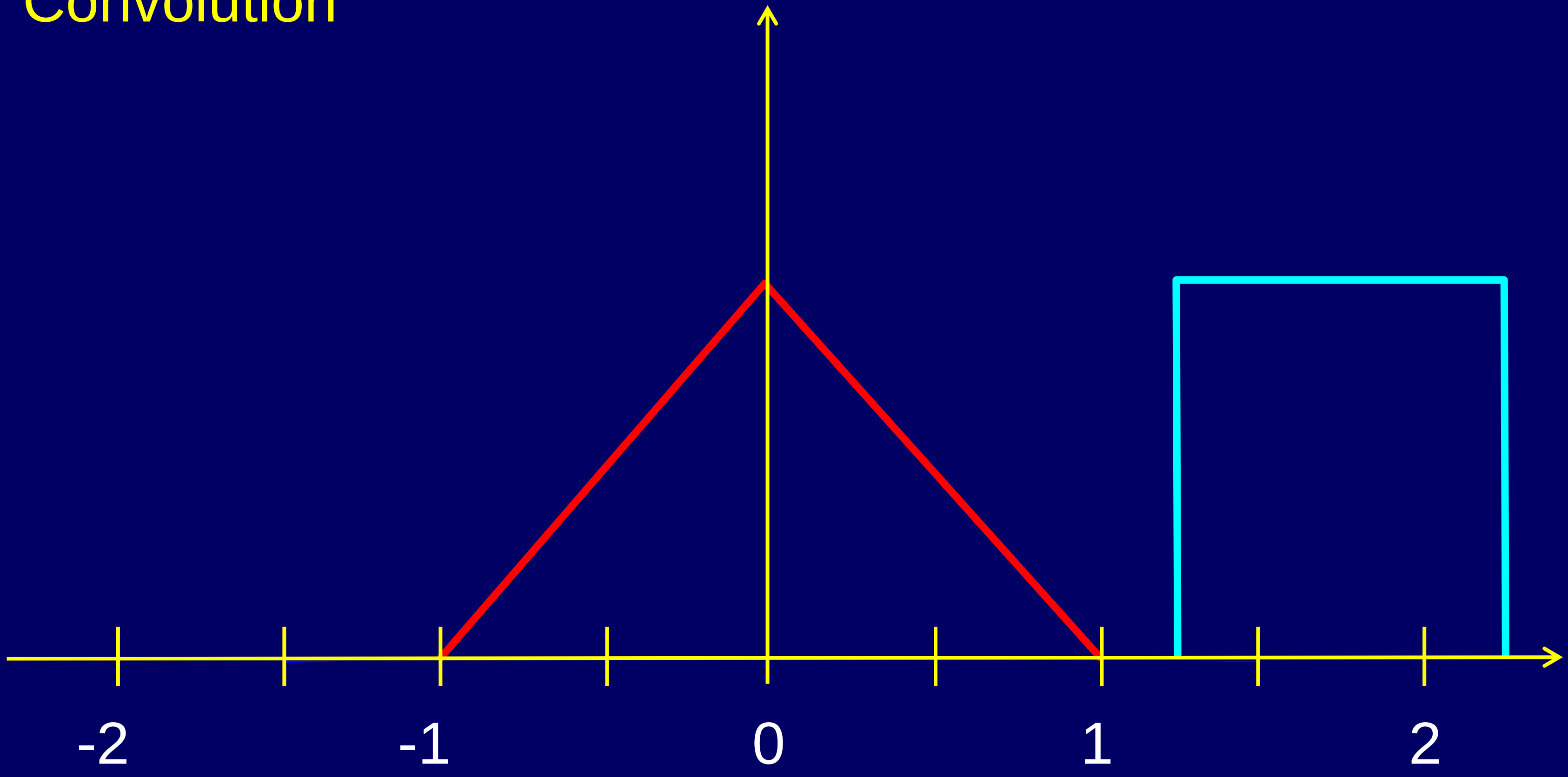
Interpolation Kernel

Convolution



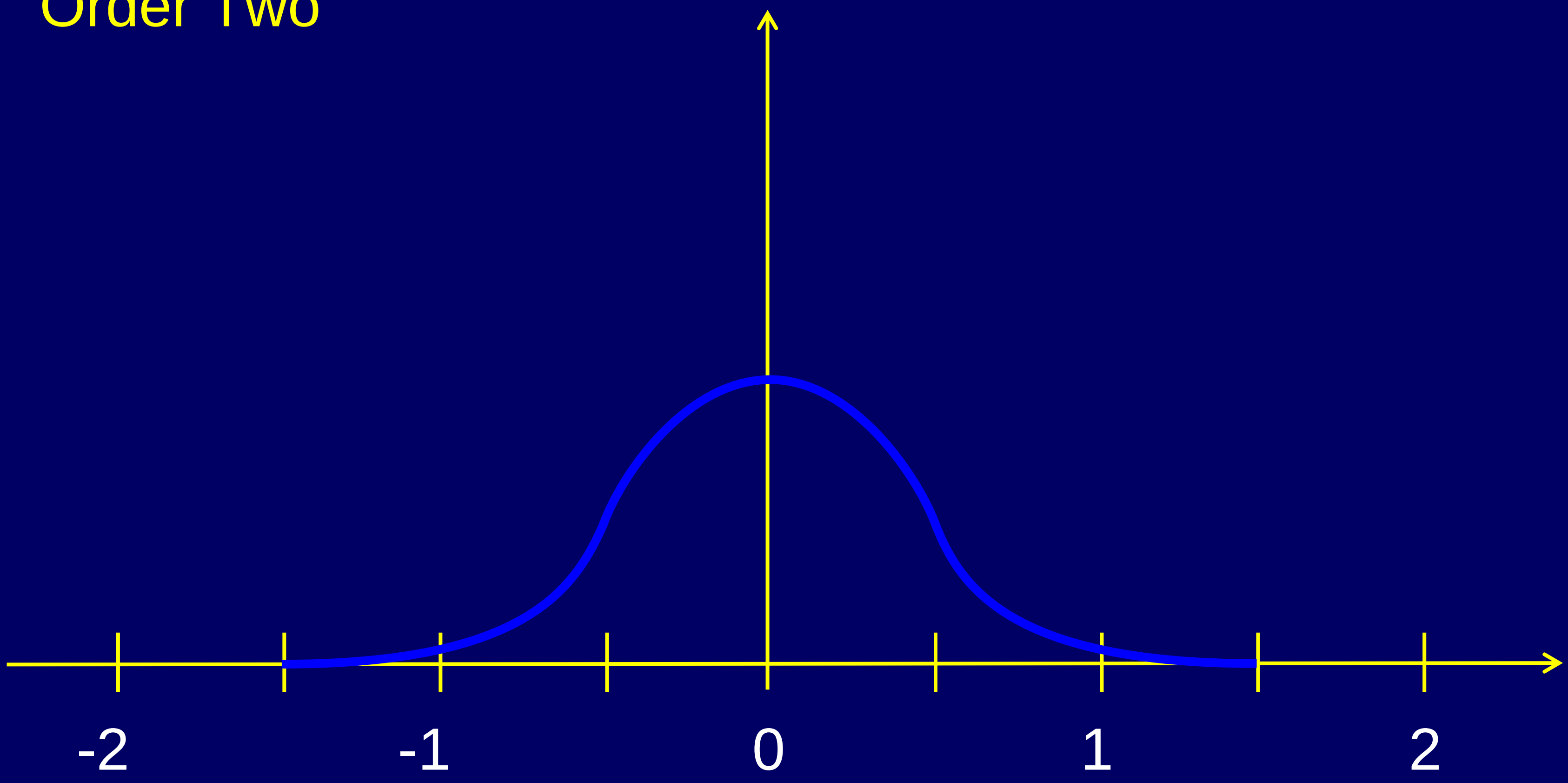
Interpolation Kernel

Convolution



Interpolation Kernel

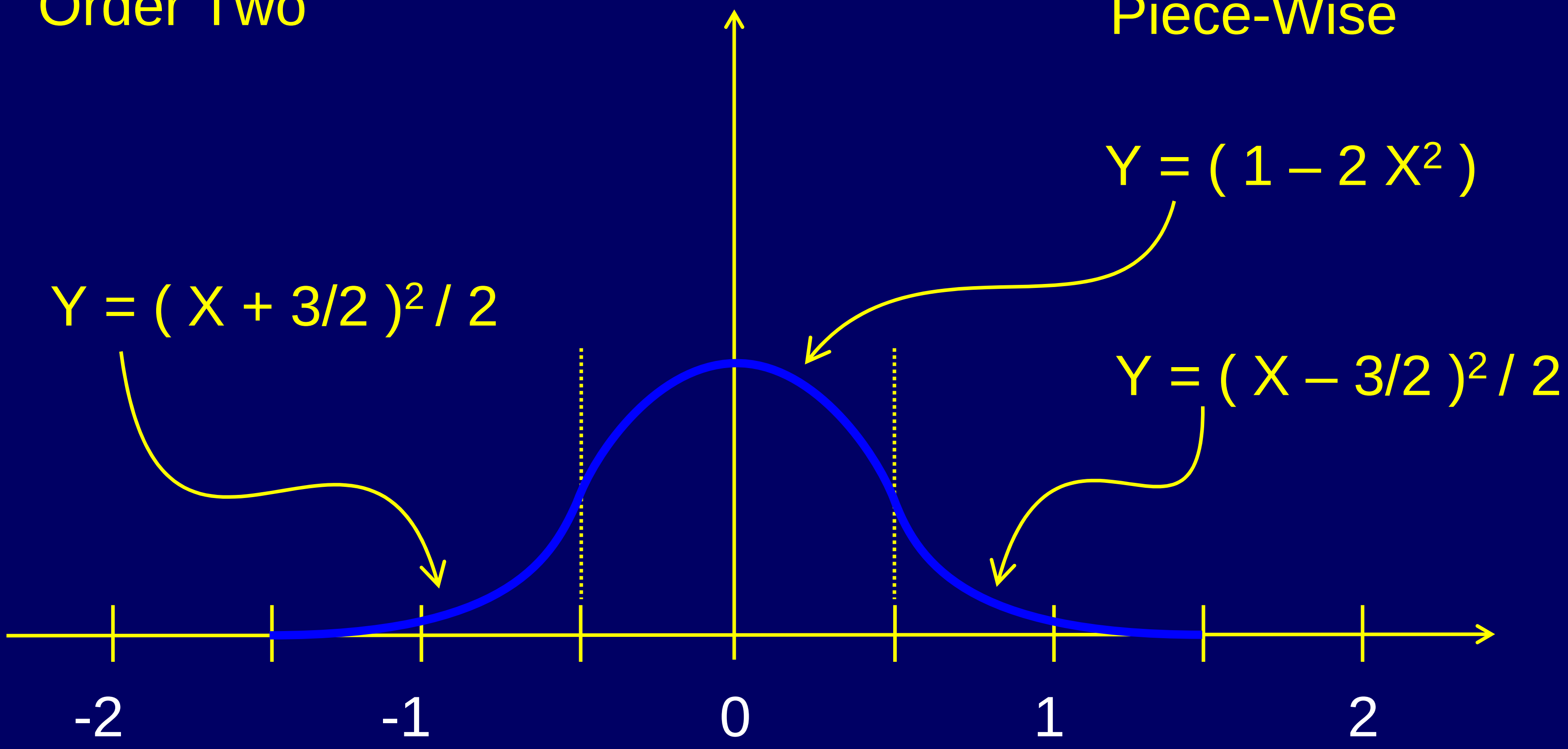
Order Two



Interpolation Kernel

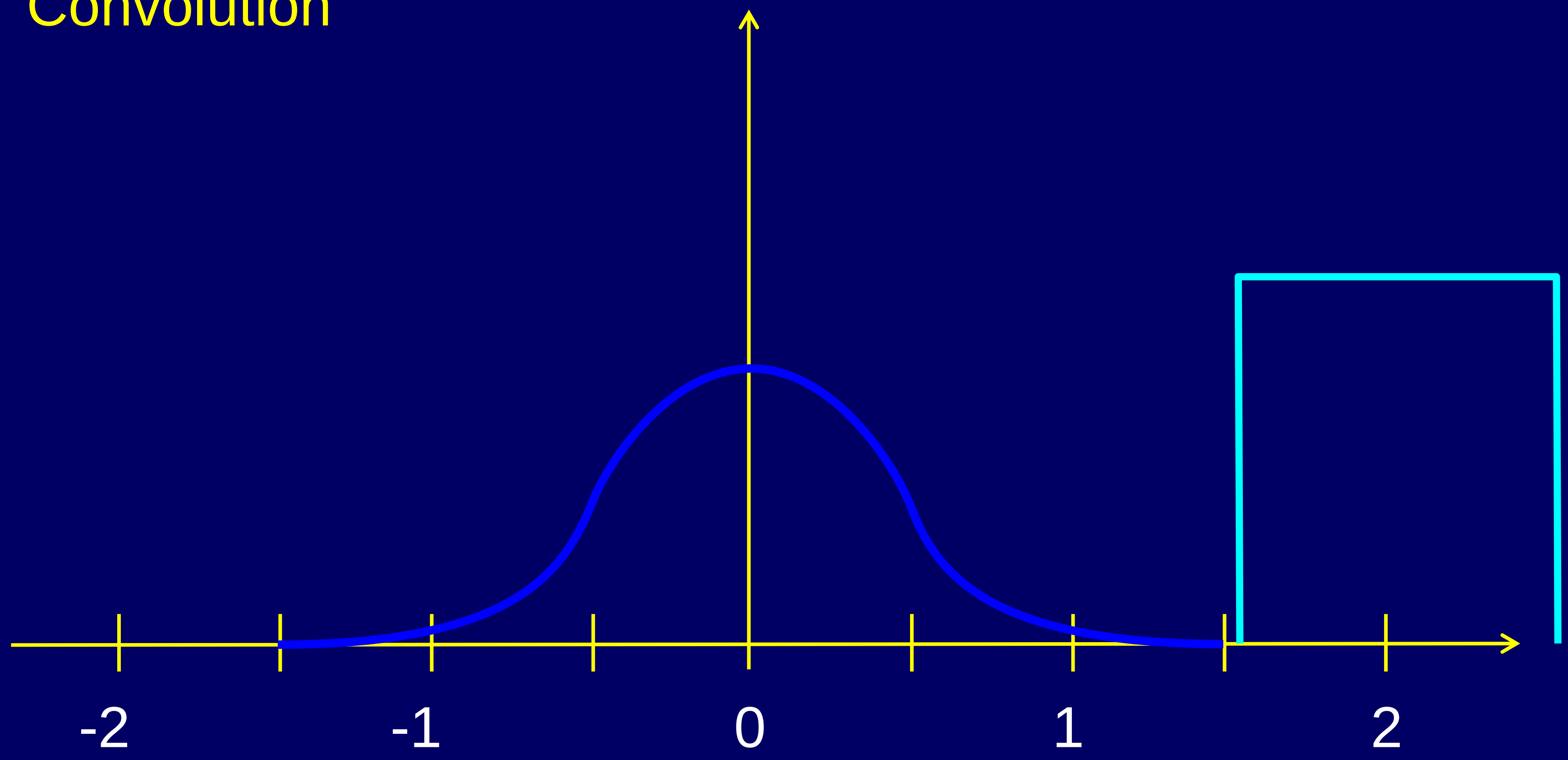
Order Two

Piece-Wise



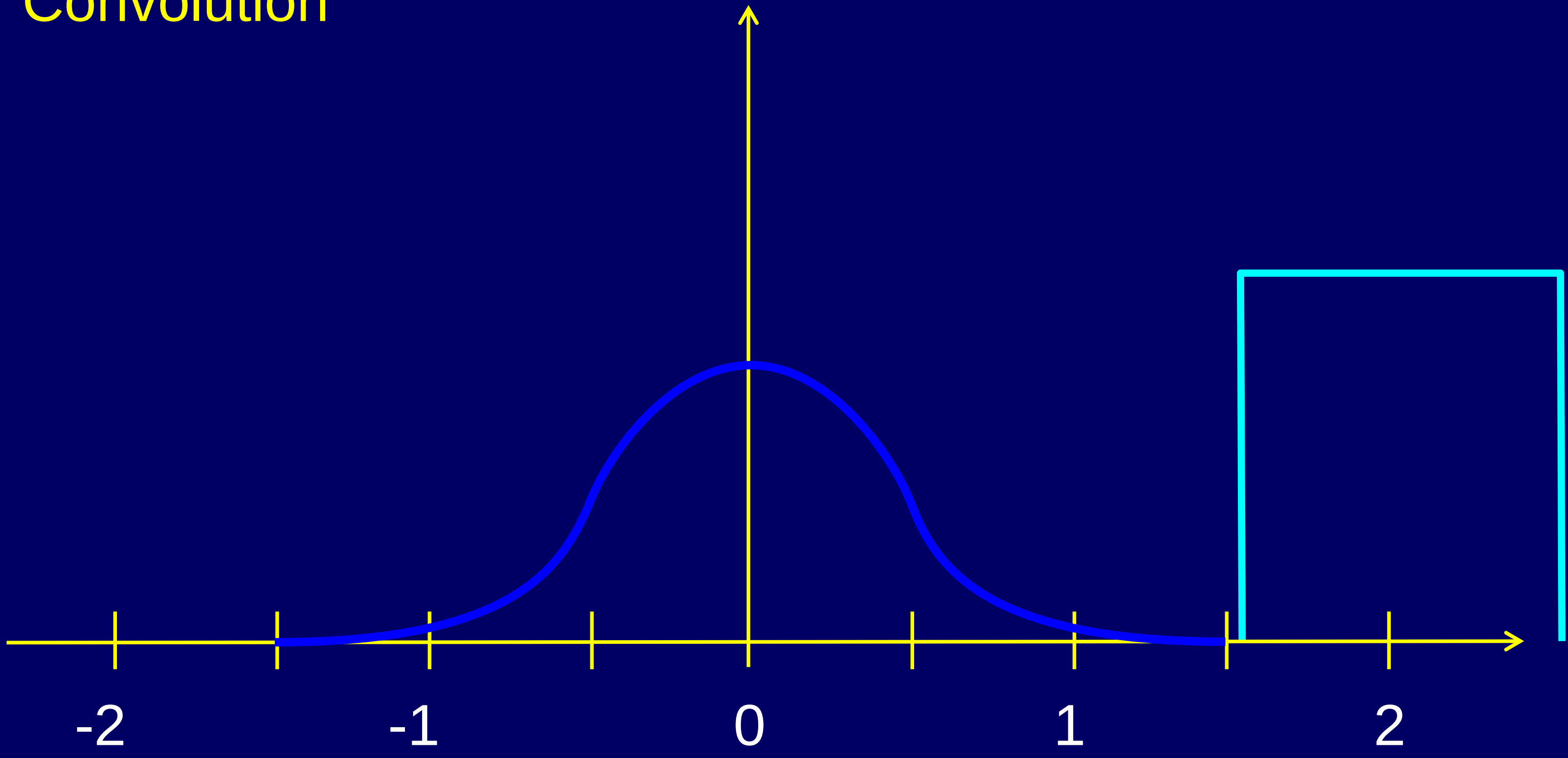
Interpolation Kernel

Convolution



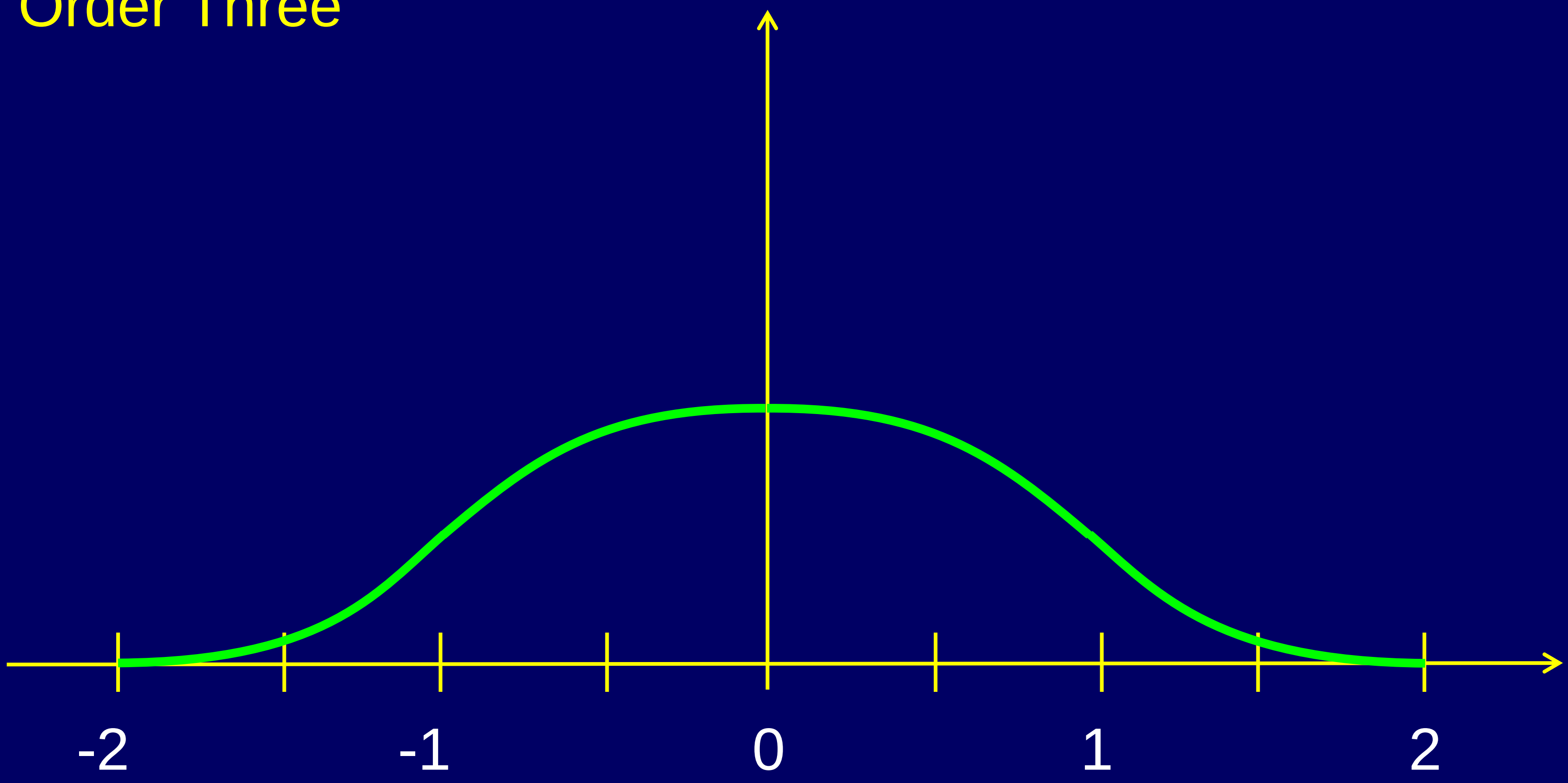
Interpolation Kernel

Convolution

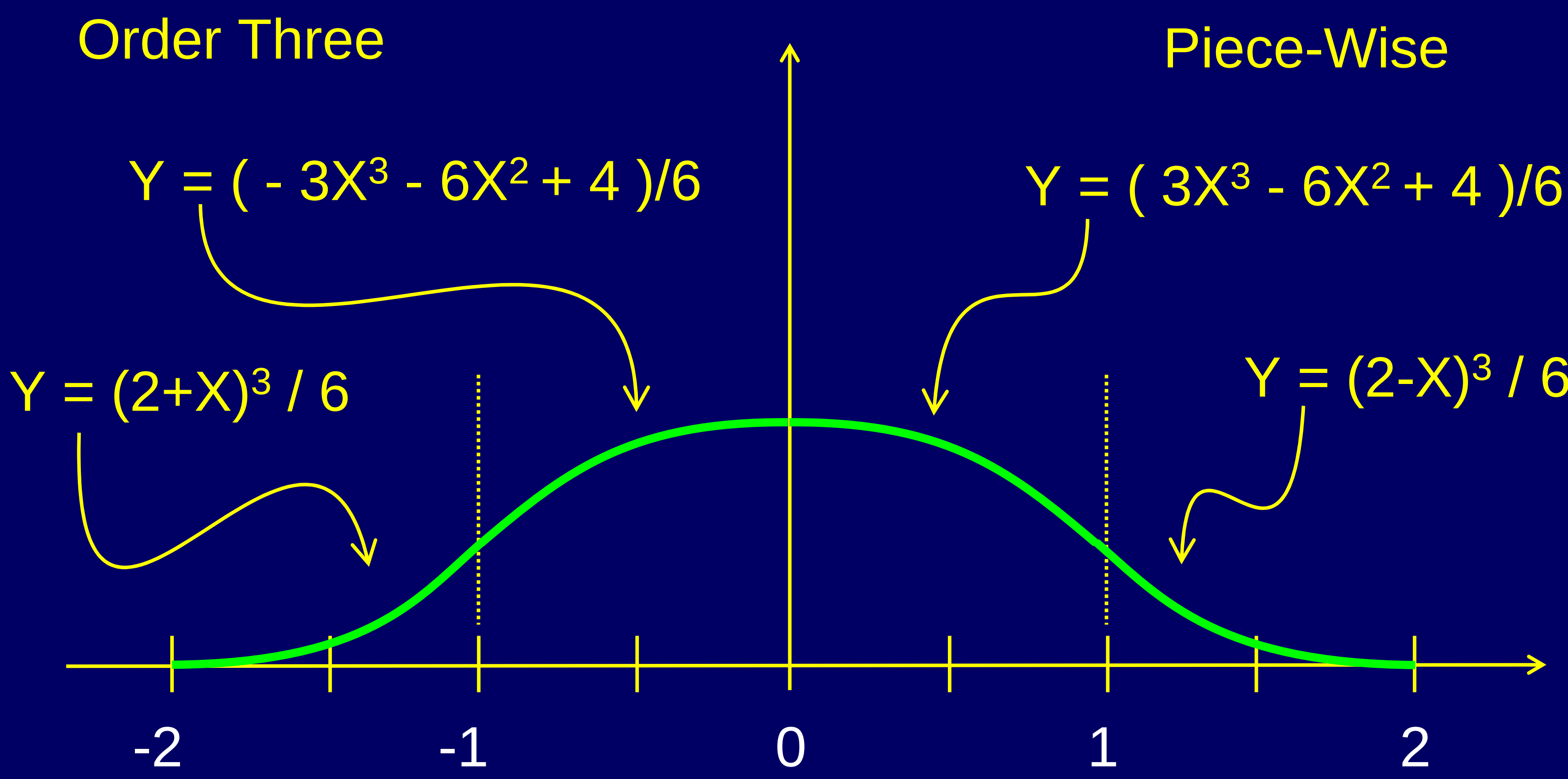


Interpolation Kernel

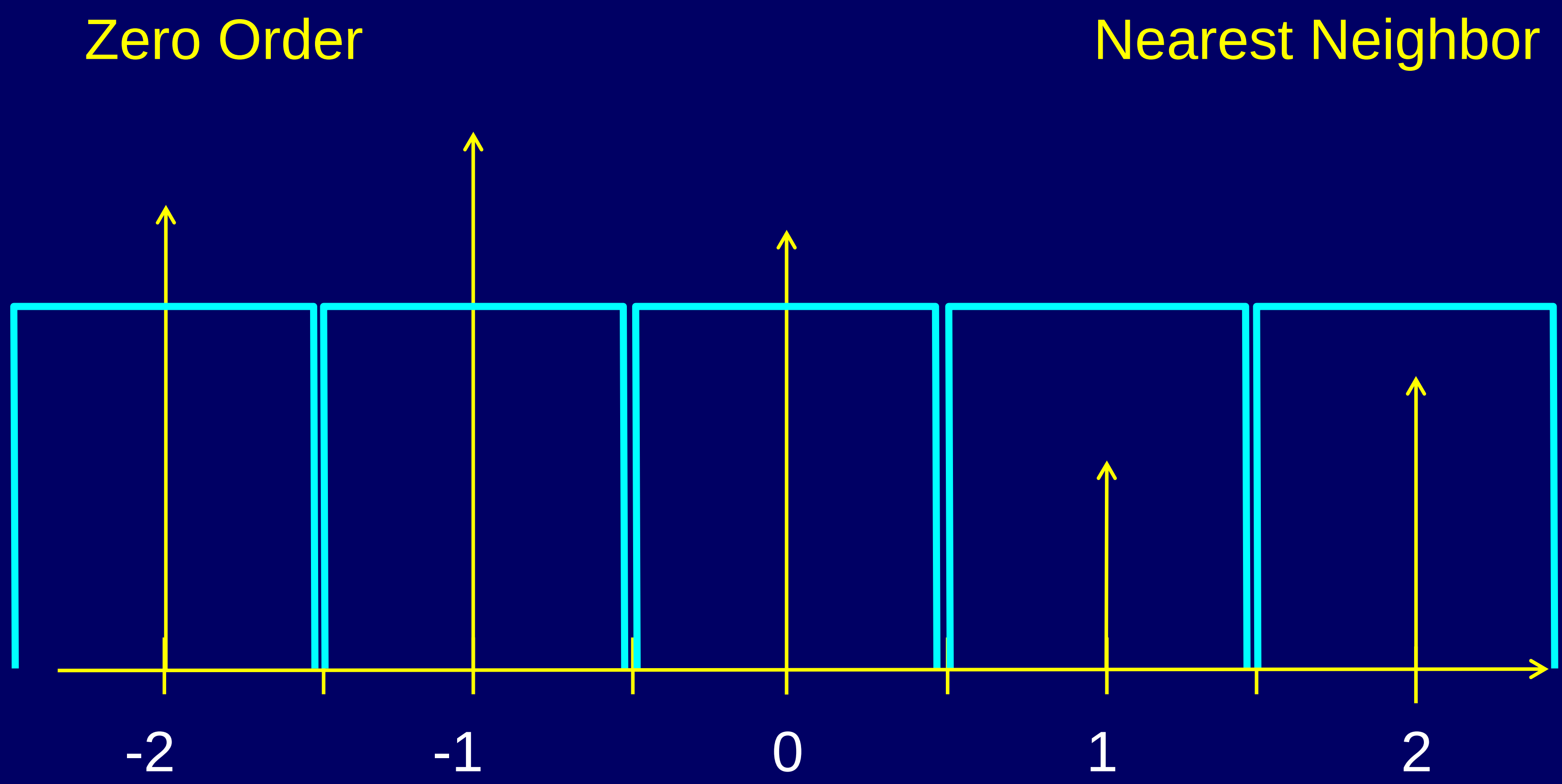
Order Three



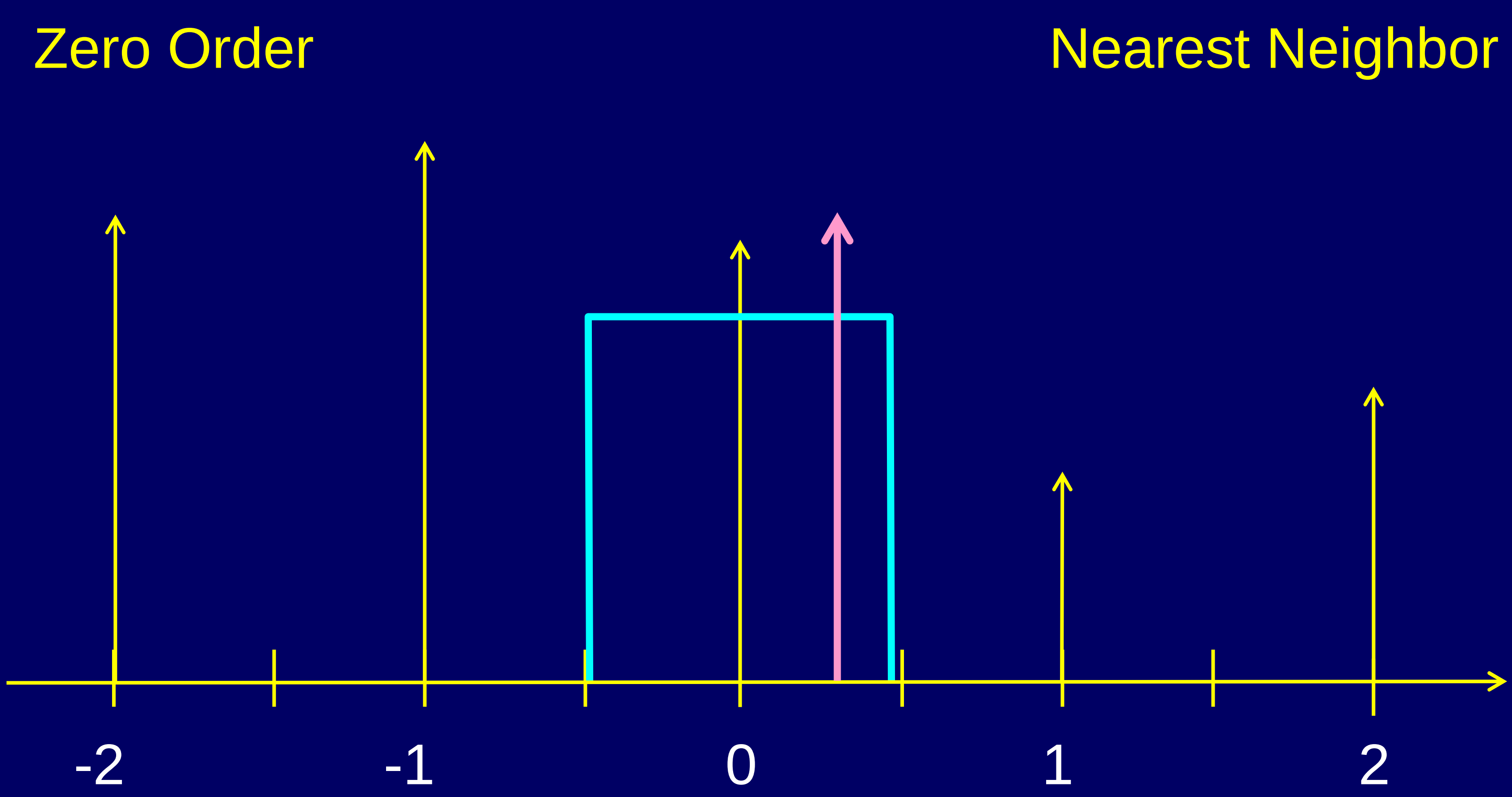
Interpolation Kernel



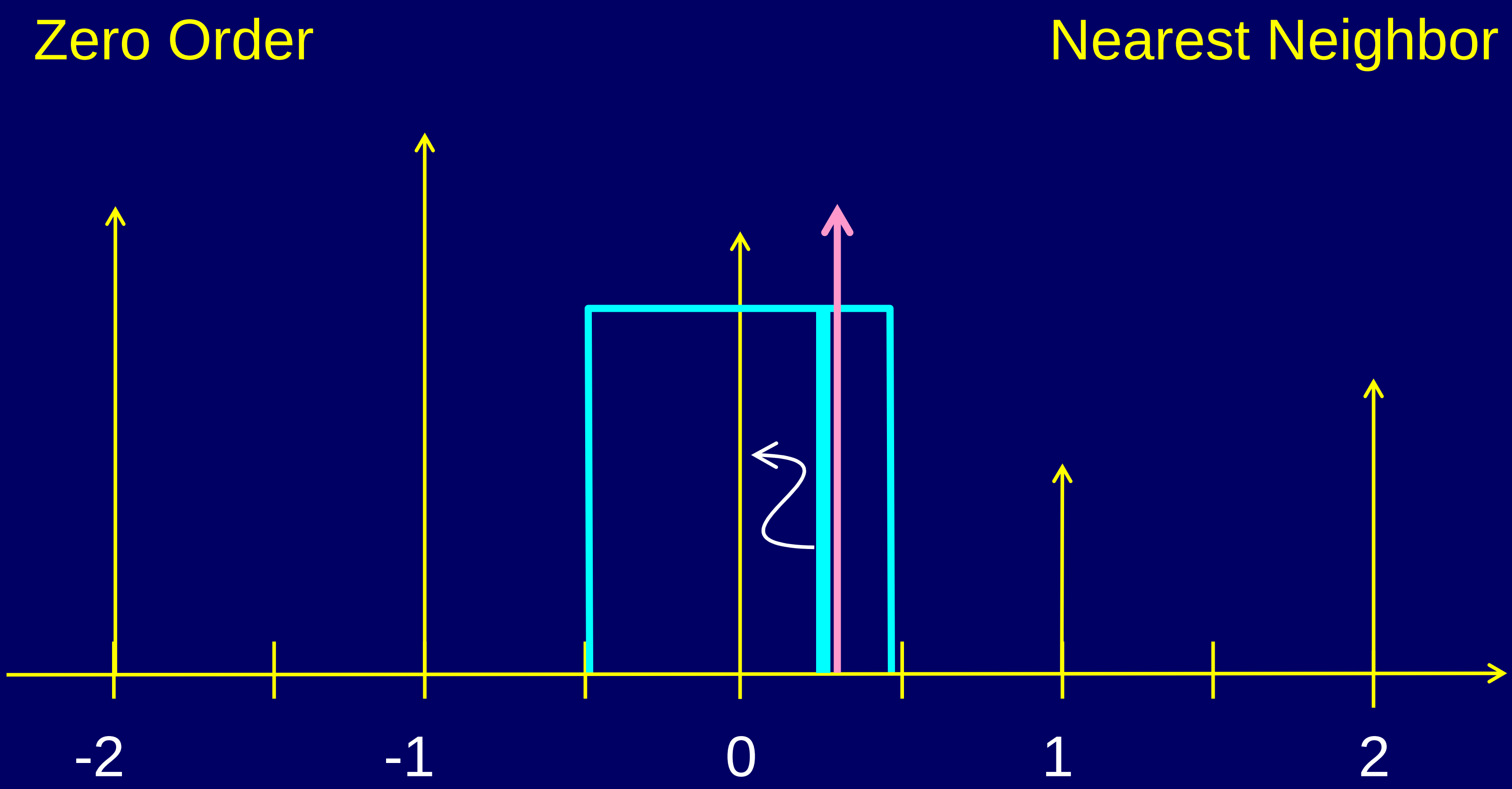
1D Interpolation



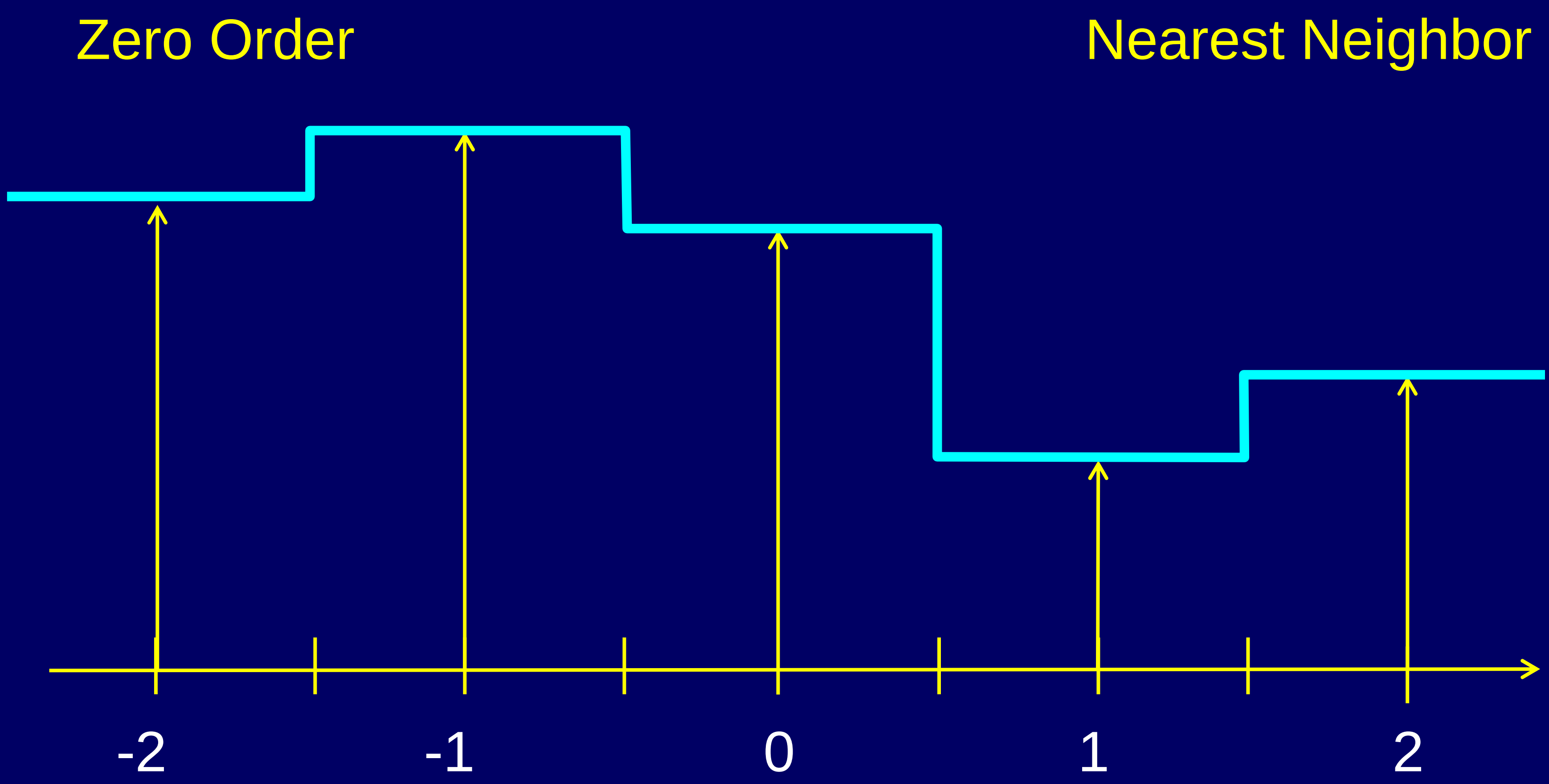
1D Interpolation



1D Interpolation



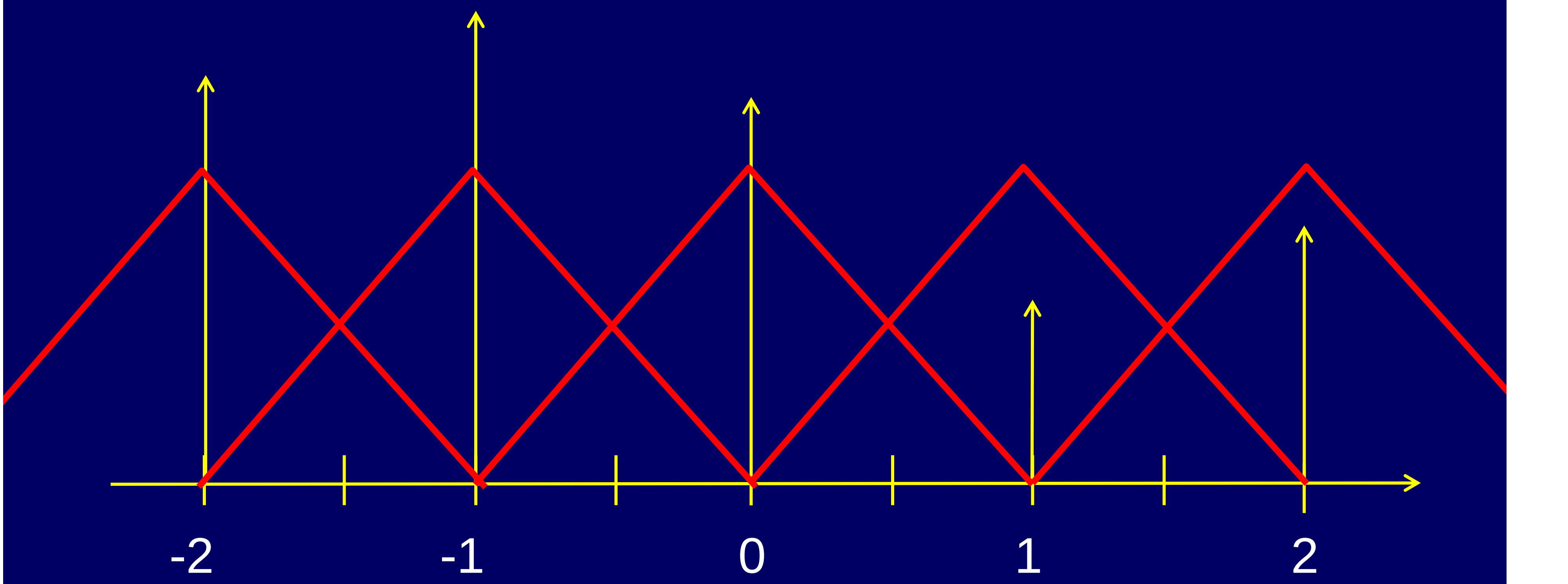
1D Interpolation



1D Interpolation

First Order

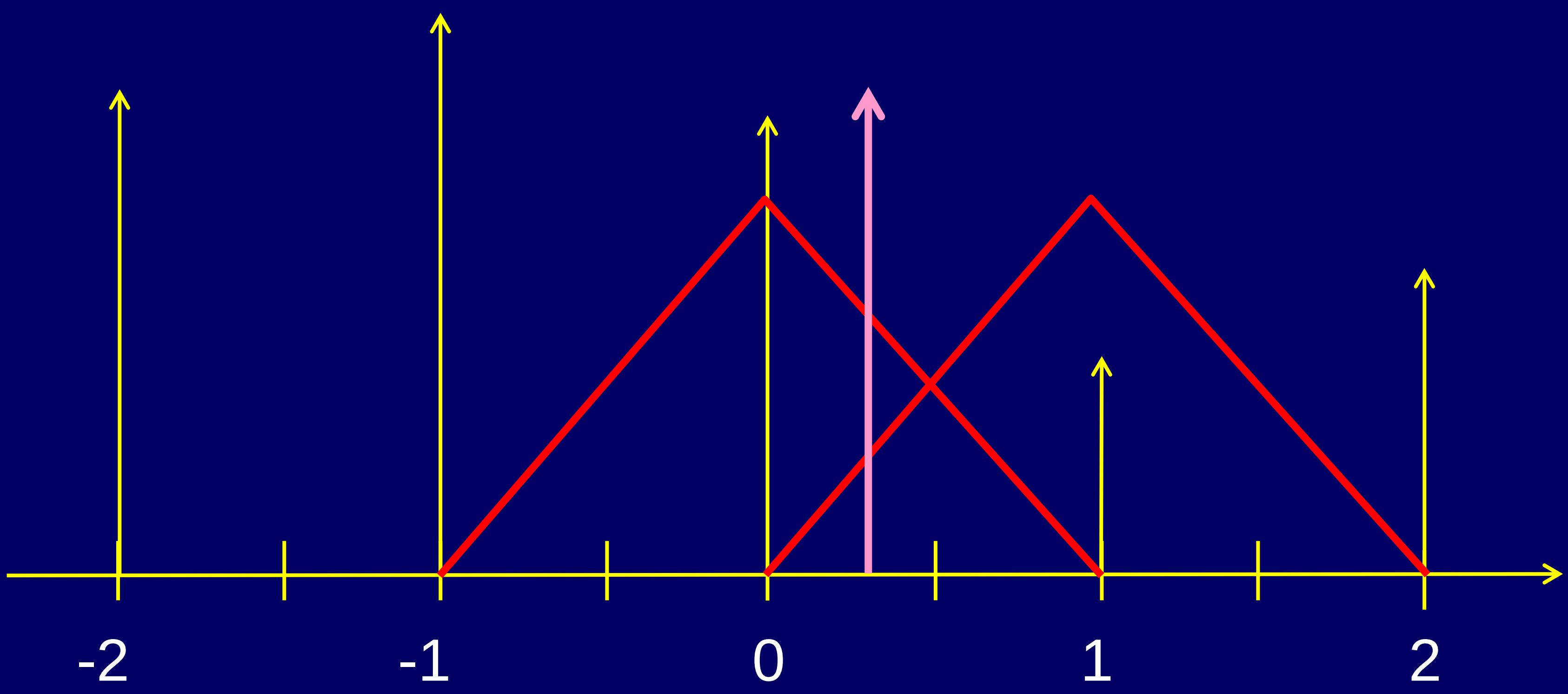
Linear Interpolation



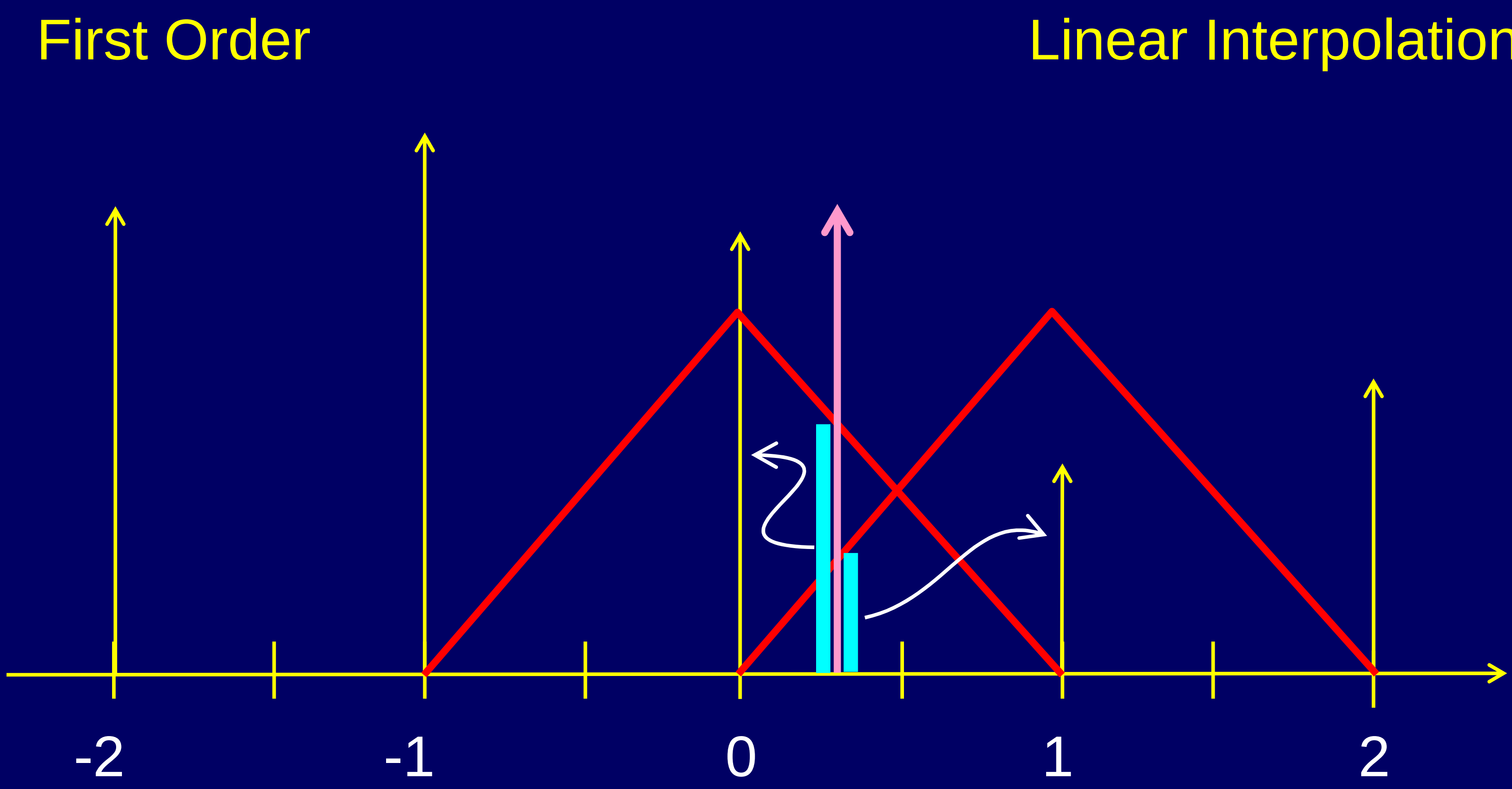
1D Interpolation

First Order

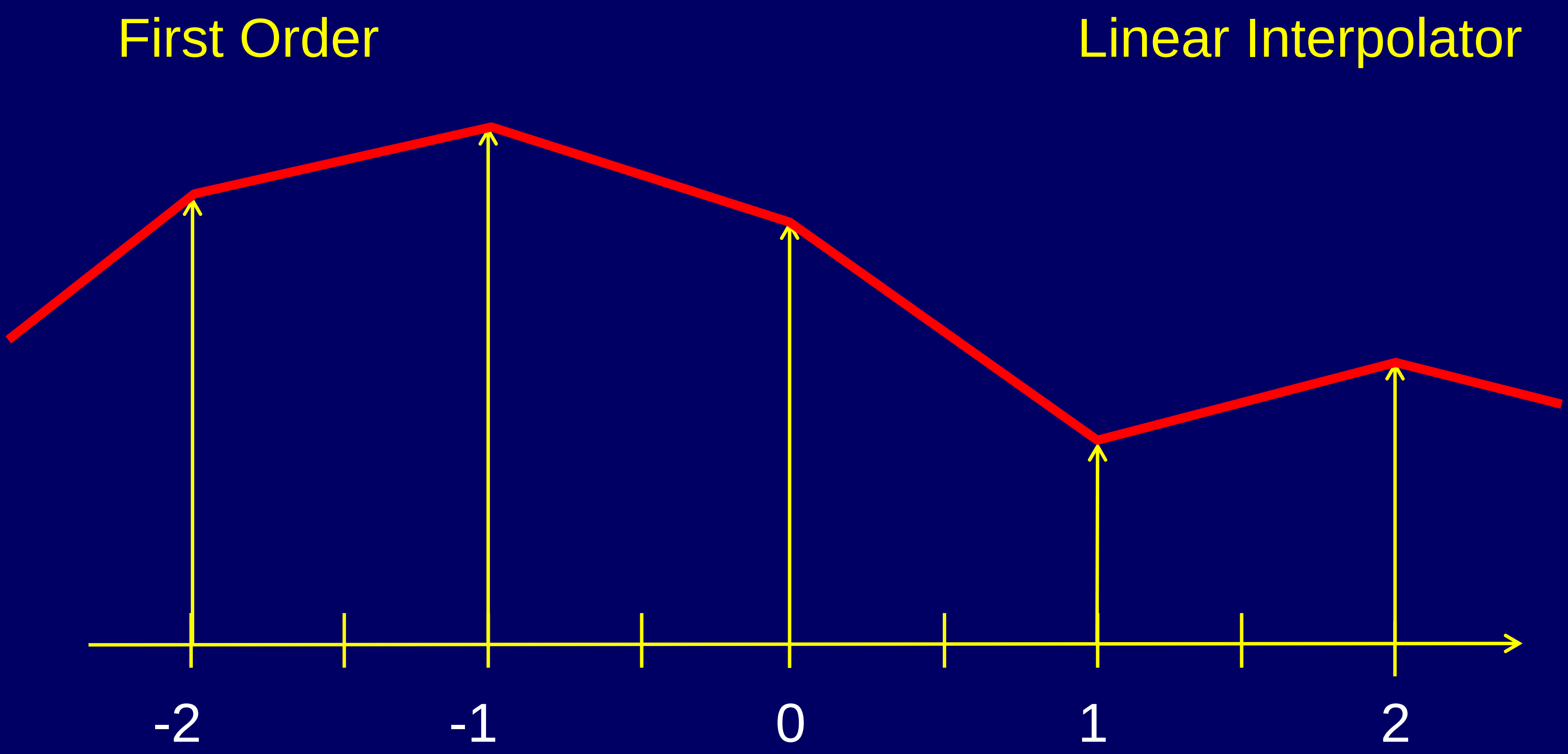
Linear Interpolation



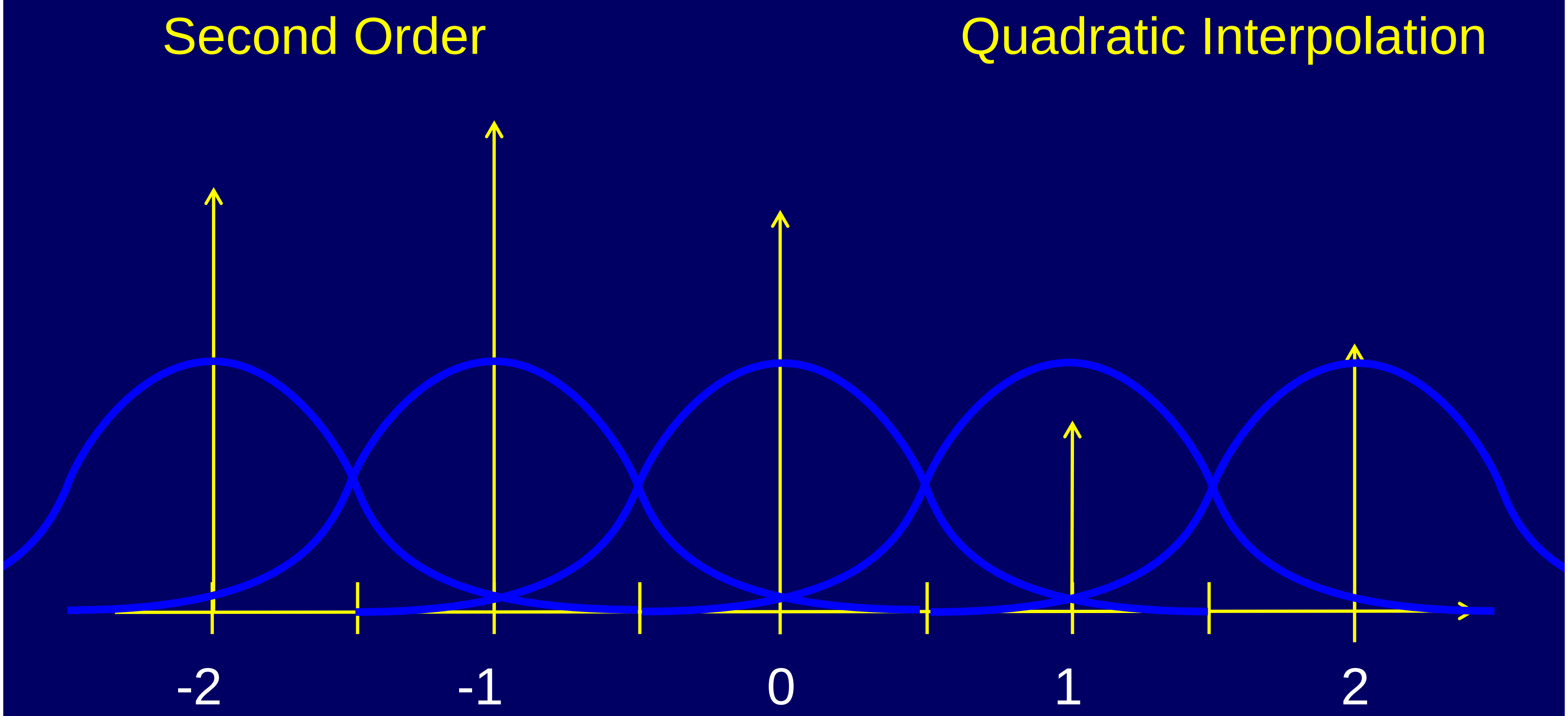
1D Interpolation



1D Interpolation



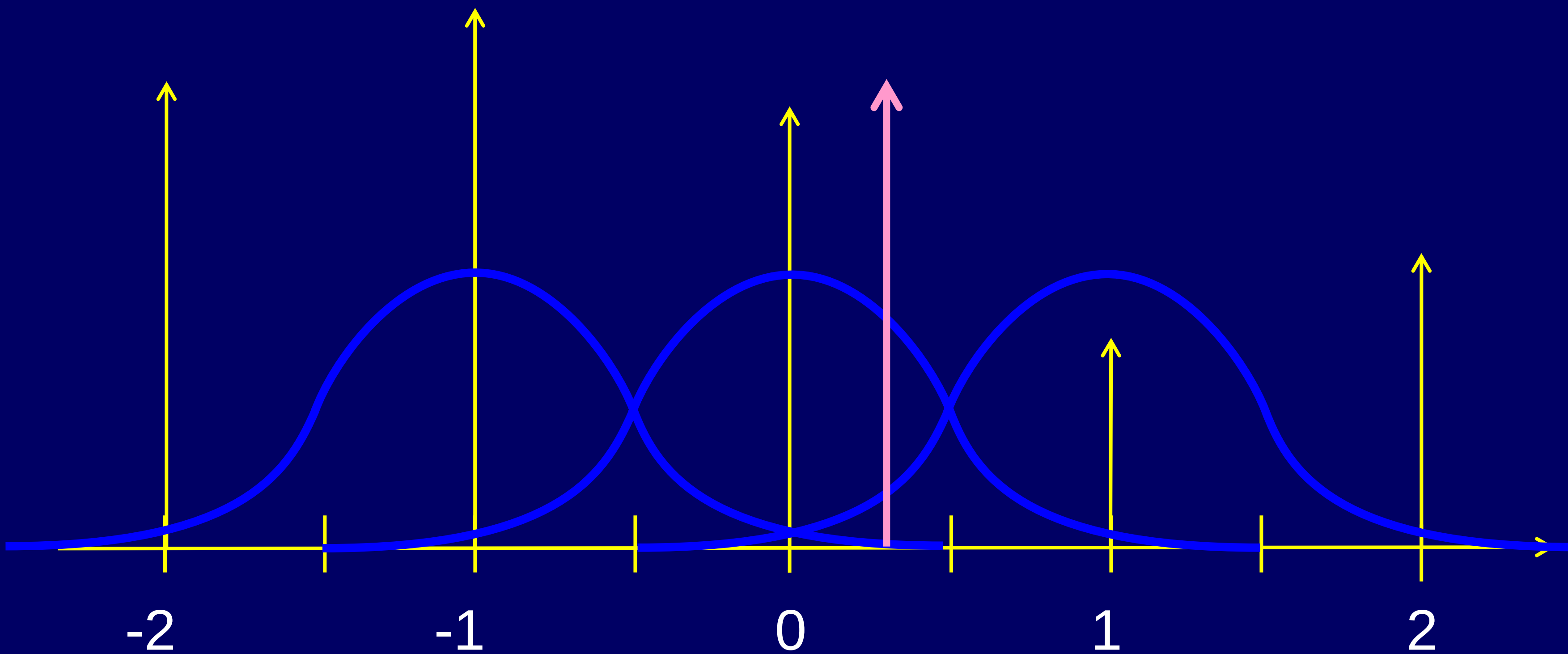
1D Interpolation



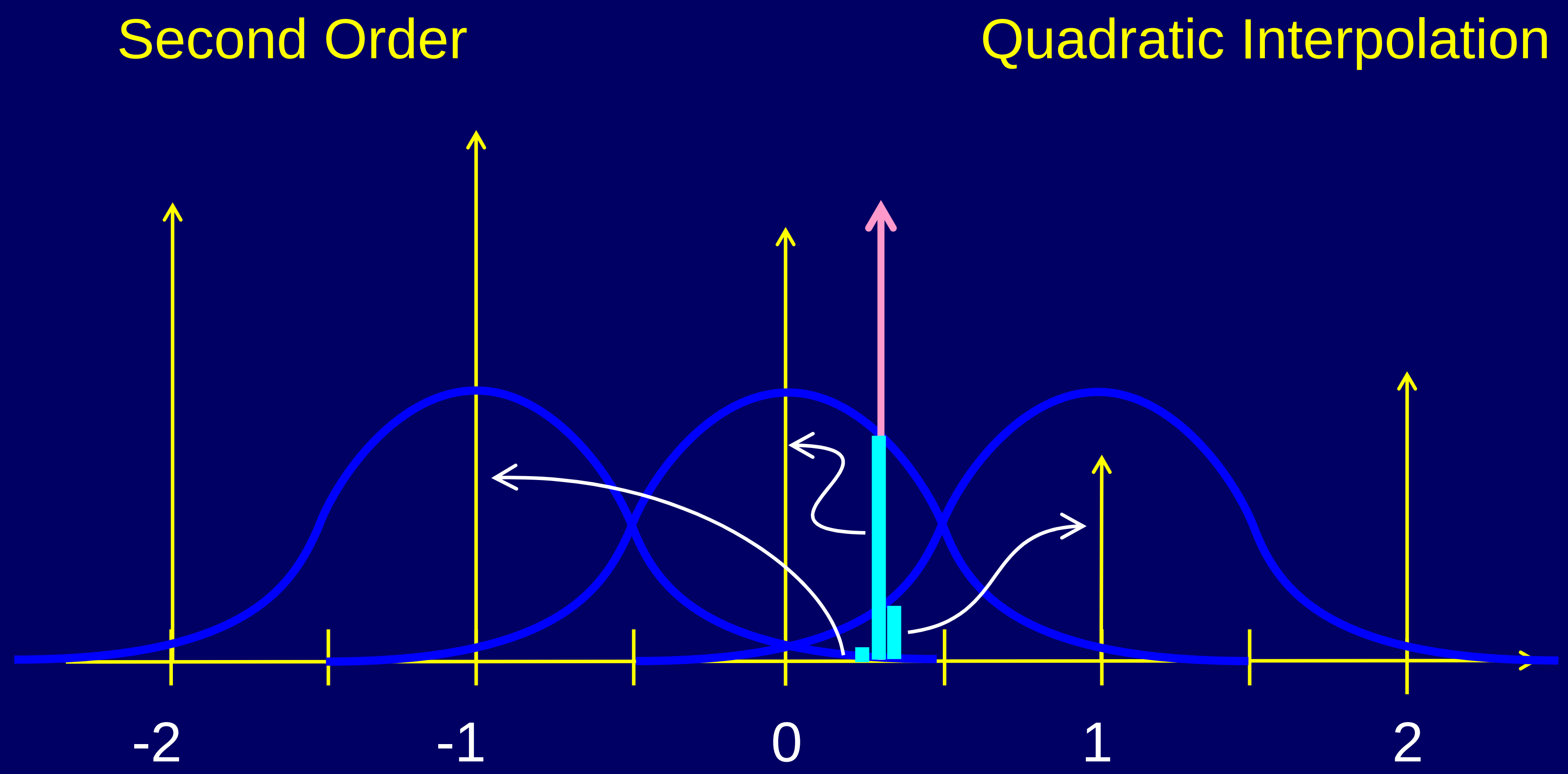
1D Interpolation

Second Order

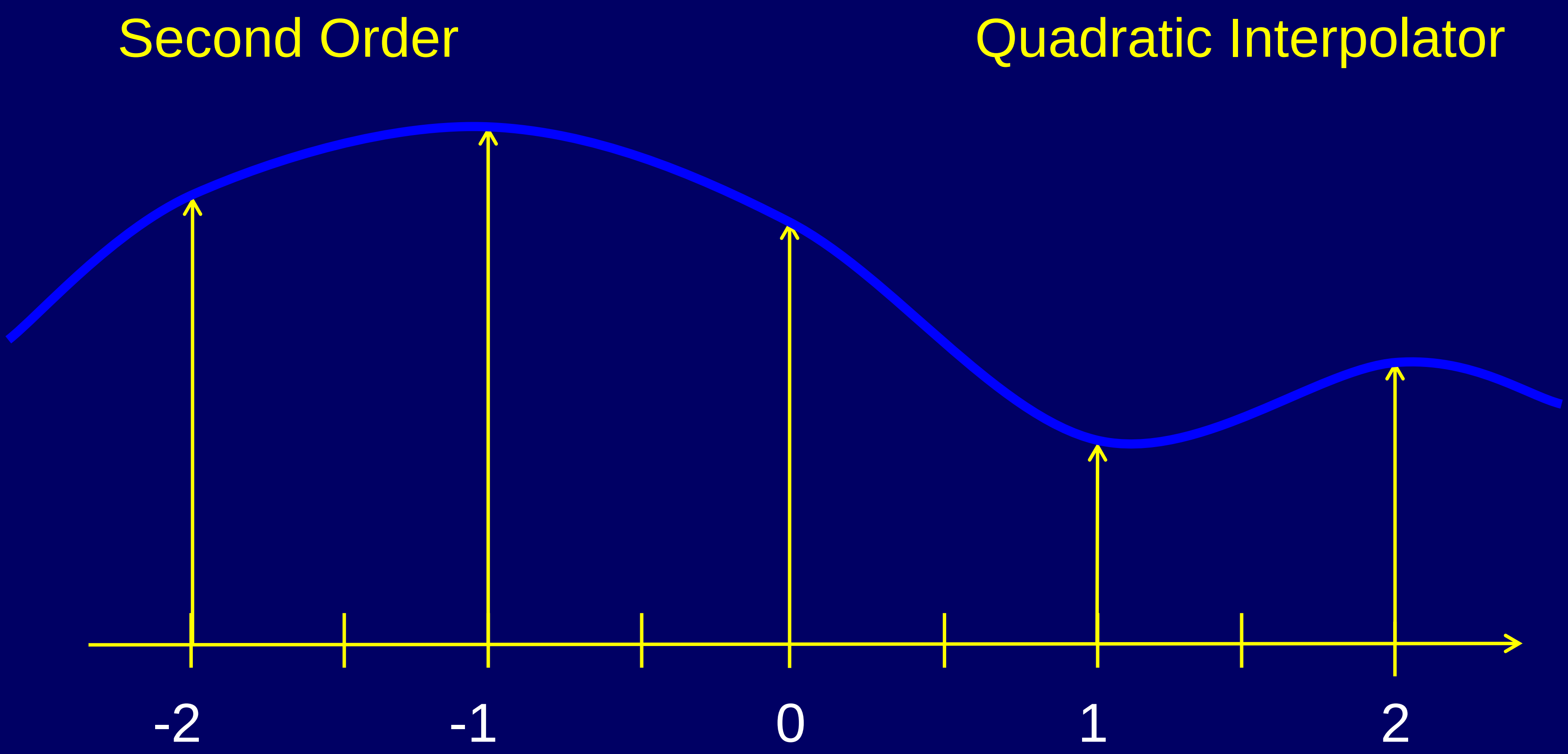
Quadratic Interpolation



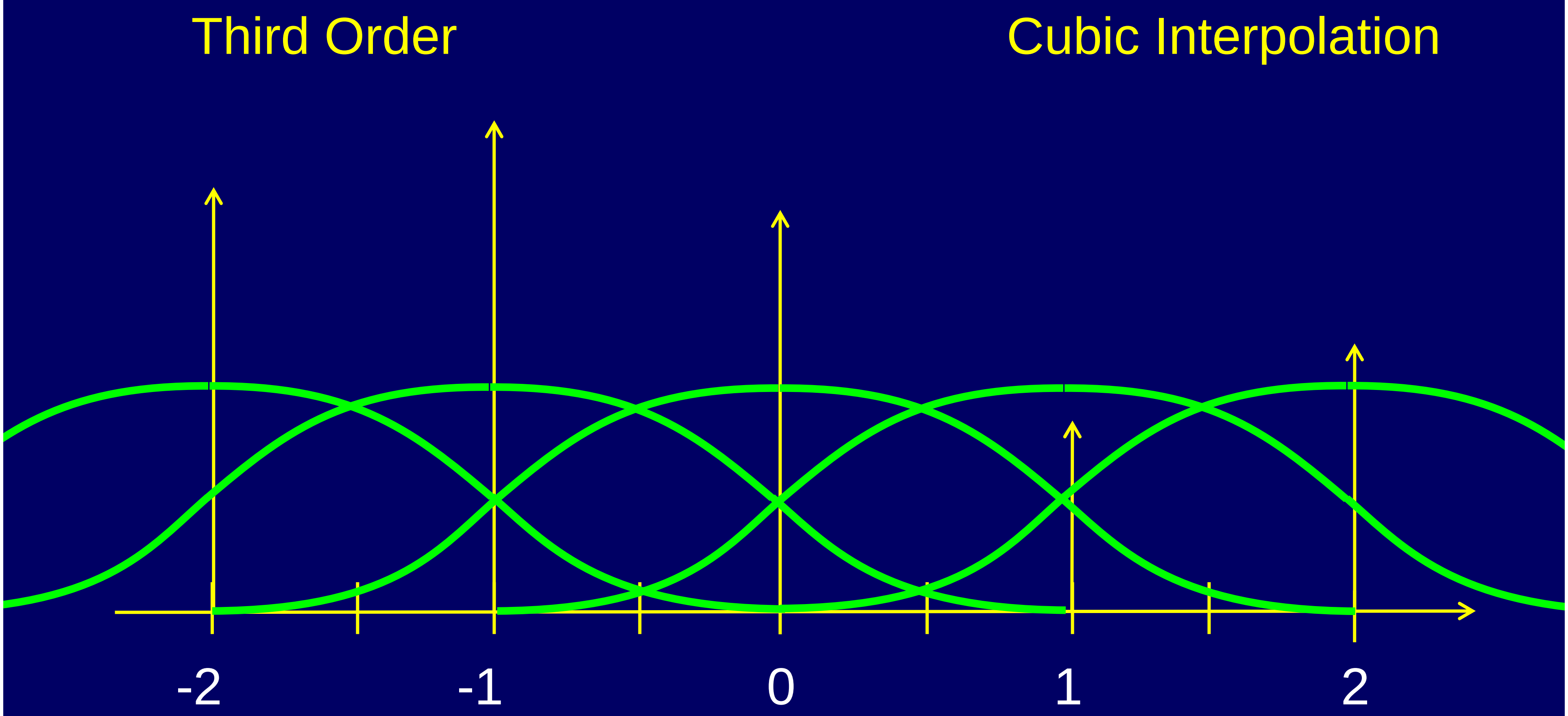
1D Interpolation



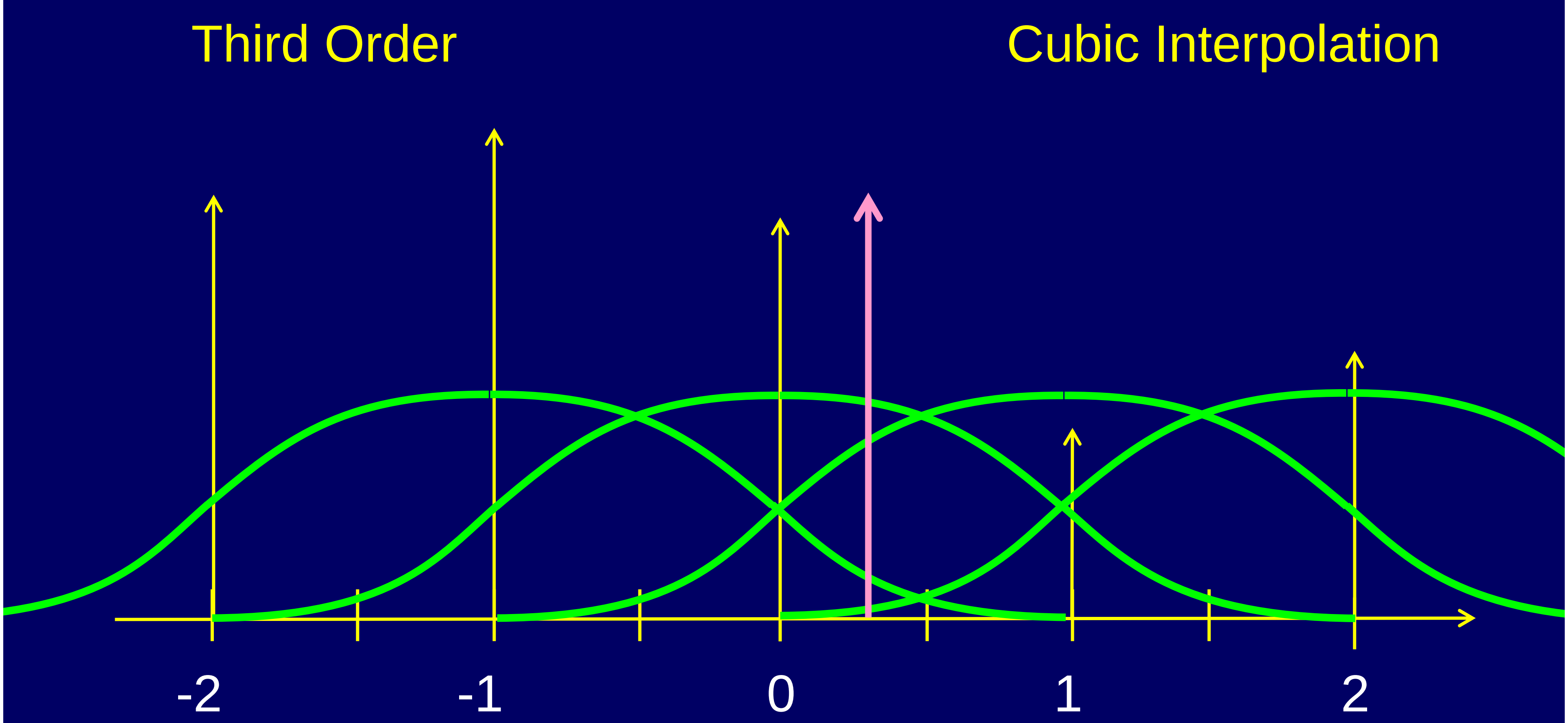
1D Interpolation



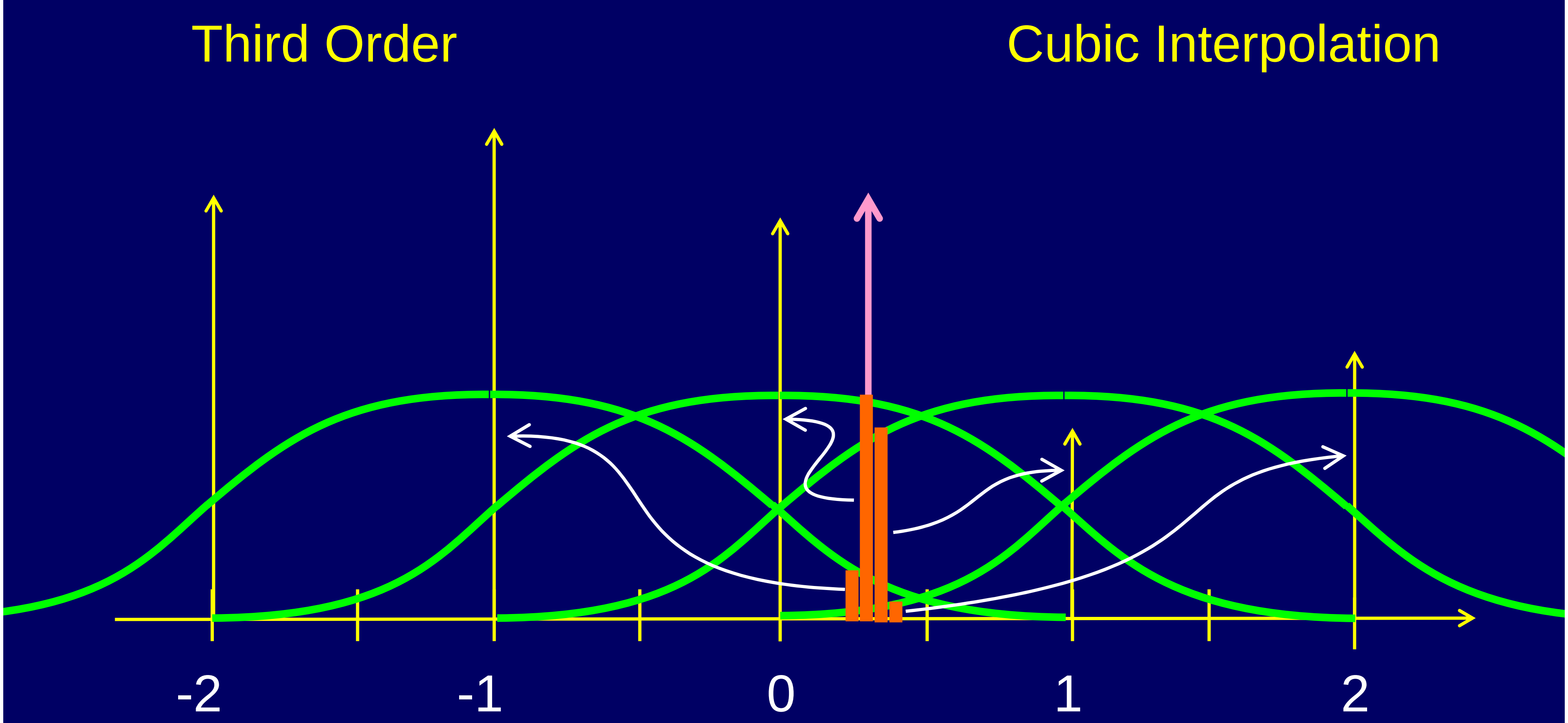
1D Interpolation



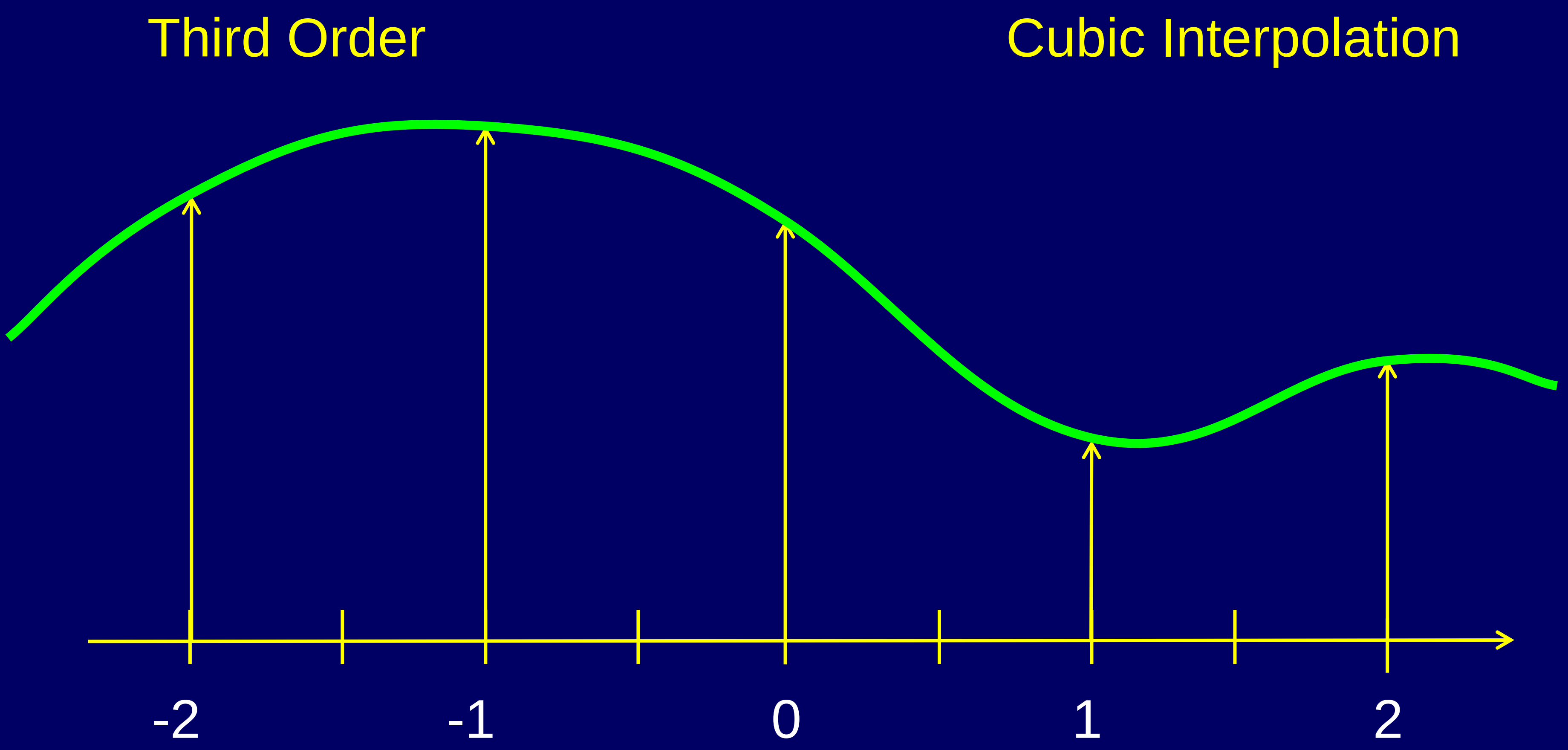
1D Interpolation



1D Interpolation



1D Interpolation

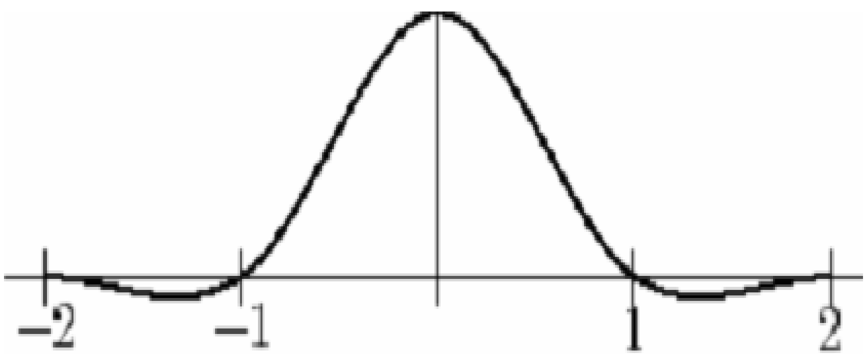


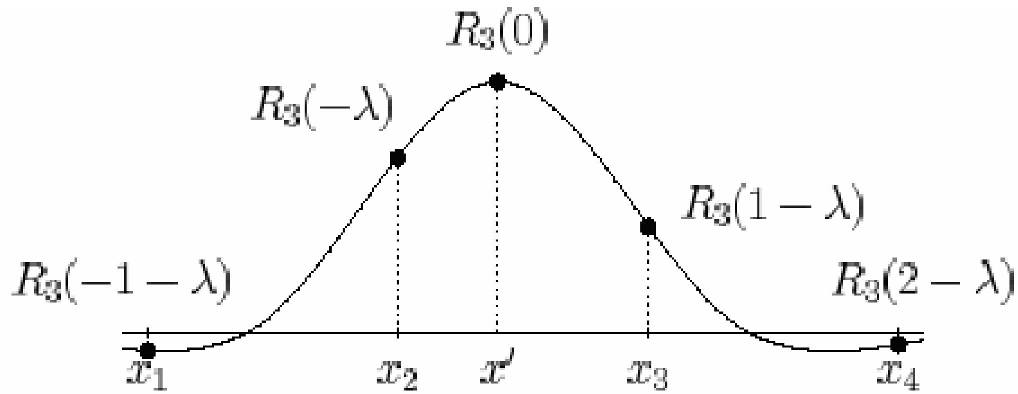
Remarks About Higher-Order Interpolation

- Higher-degree polynomials:
  - e.g., cubic
- Sometimes other interpolating functions
- Requires a larger neighborhood:
  - e.g., bicubic requires a 4×4 neighborhood
- More expensive

Another 3rd order (Cubic) Example

$$R_3(u) = \begin{cases} 1.5|u|^3 - 2.5|u|^2 + 1 & if\ |u| \le 1 \\ -0.5|u|^3 + 2.5|u|^2 - 4|u| + 2 & if\ 1 < |u| \le 2 \end{cases}$$



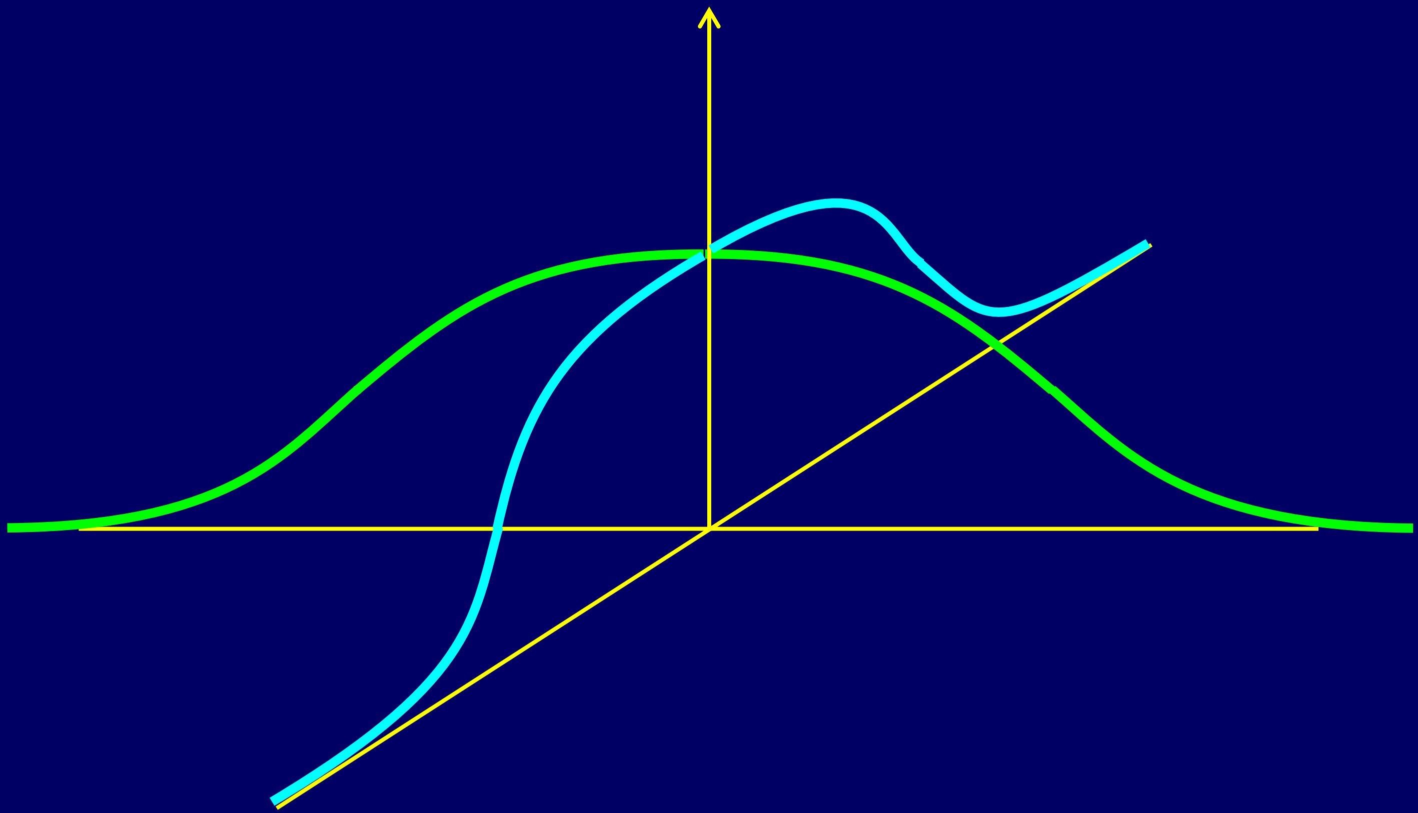


Now have 4 support points:

$$f(x') = R_3(-1-\lambda)\,f(x_1) + R_3(-\lambda)\,f(x_2) + R_3(1-\lambda)\,f(x_3) + R_3(2-\lambda)\,f(x_4)$$

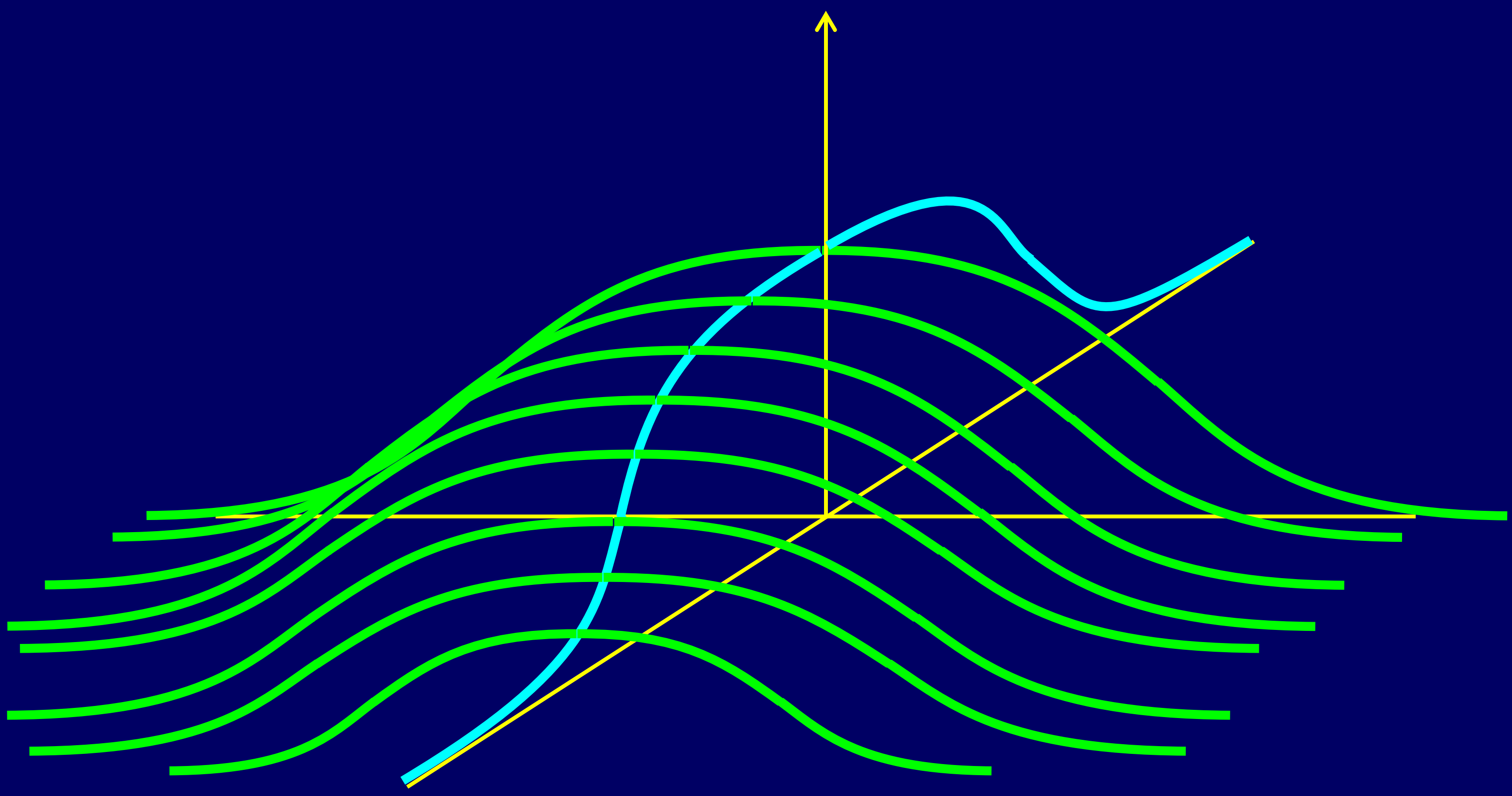
2D Interpolation

Kernel Product



2D Interpolation

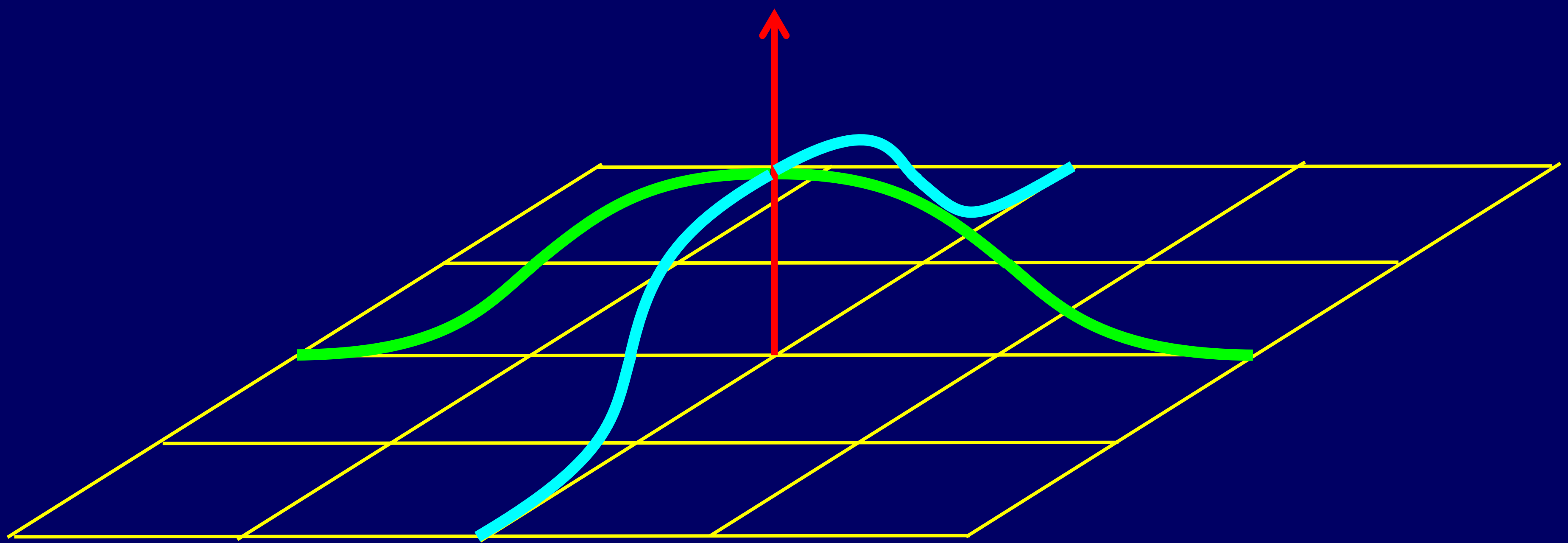
Kernel Product



2D Interpolation

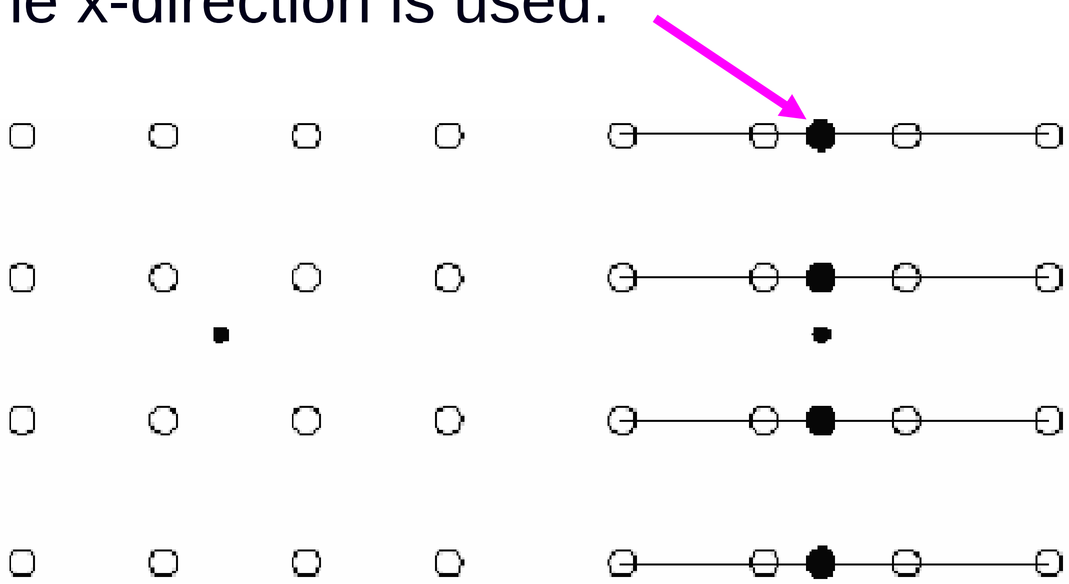
Kernel Product

x, y separable variables



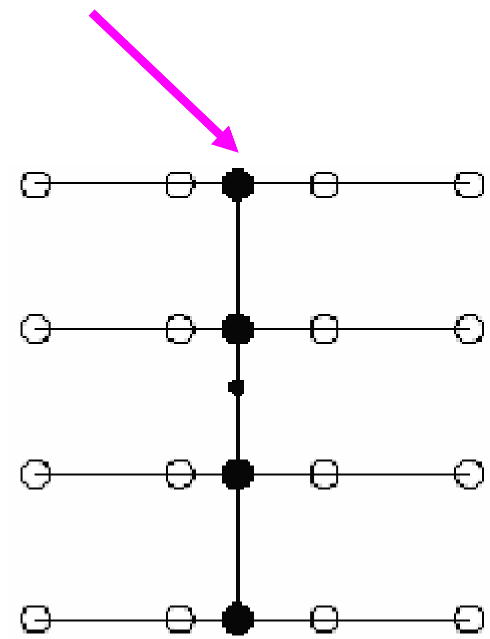
Bicubic (2D)

- Bicubic interpolation fits a series of cubic polynomials to the brightness values contained in the 4 x 4 array of pixels surrounding the calculated address.
  - Step 1: four cubic polynomials F(i), i = 0, 1, 2, 3 are fit to the control points along the rows. The fractional part of the calculated pixel's address in the x-direction is used.



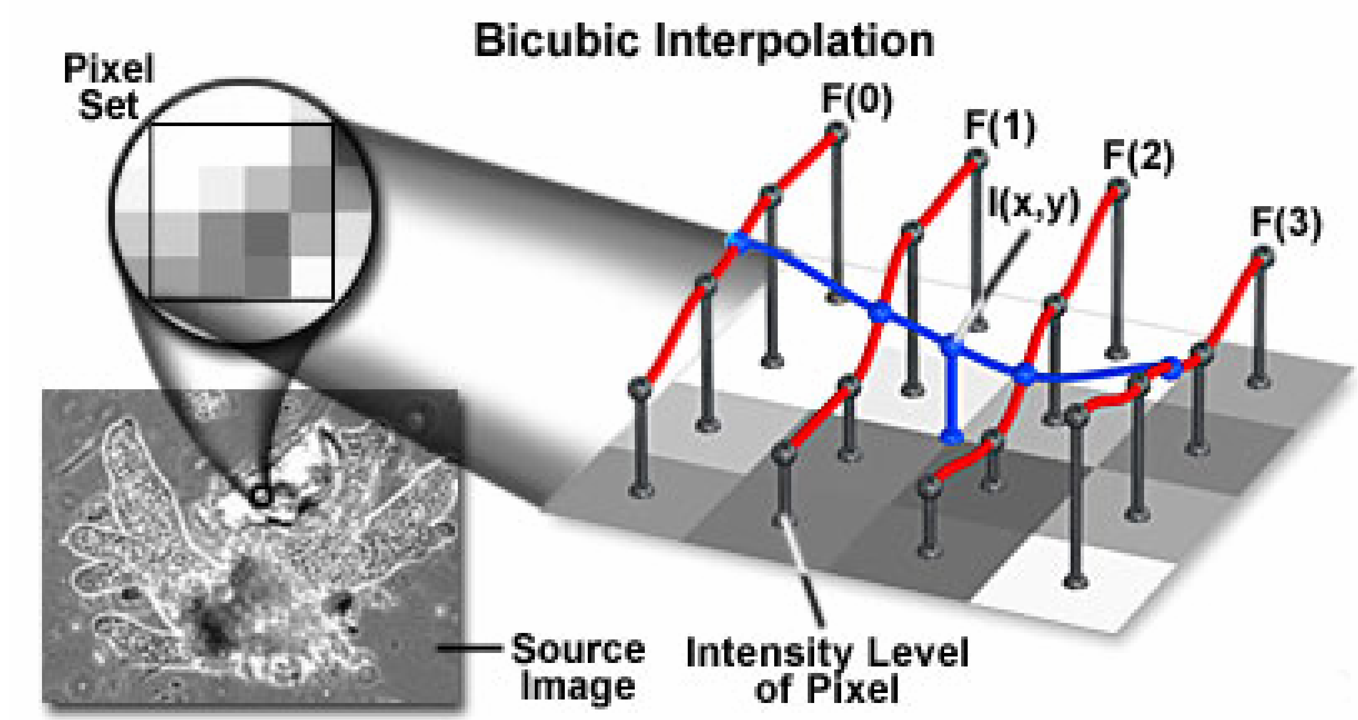
Bicubic (2D)

- Step 2: the fractional part of the calculated pixel's address in the y-direction is used to fit another cubic polynomial down the column, based on the interpolated brightness values that lie on the curves F(i), i = 0, ..., 3.



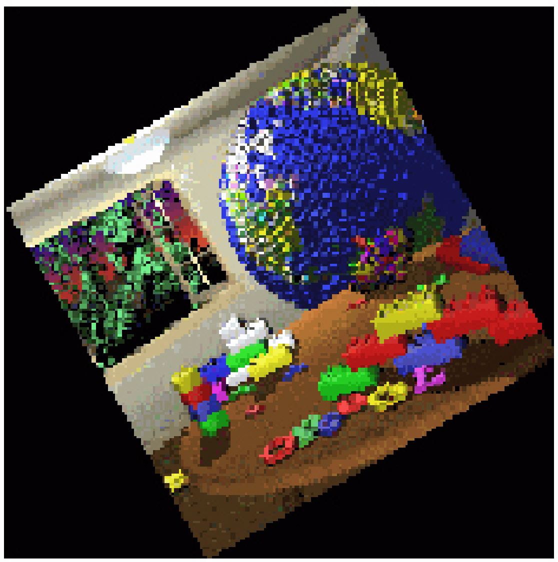
Bicubic (2D)

- Substituting the fractional part of the calculated pixel's address in the x-direction into the resulting cubic polynomial then yields the interpolated pixel's brightness value.

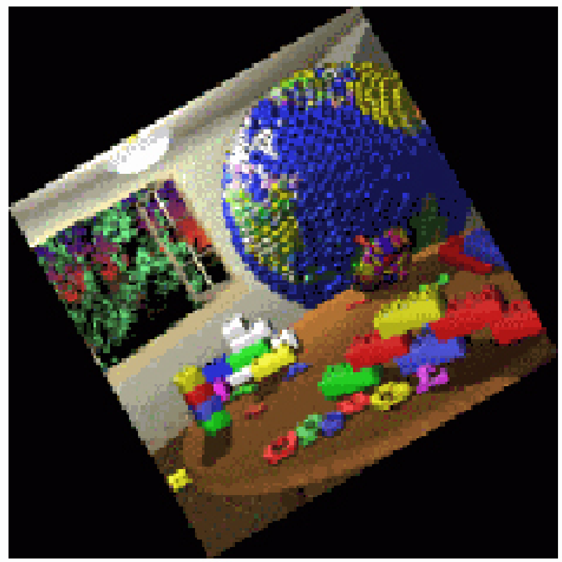


Three Interpolations Comparison

- Trade offs:
  - Aliasing versus blurring
  - Computation speed



nearest neighbor



bilinear

bicubic

General Interpolation: Summary

- For NN interpolation, the output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered.
- For bilinear interpolation, the output pixel value is a weighted average of pixels in the nearest 2-by-2 neighborhood.
- For bicubic interpolation, the output pixel value is a weighted average of pixels in the nearest 4-by-4 neighborhood.
- Bilinear method takes longer than nearest neighbor interpolation, and the bicubic method takes longer than bilinear.
- The greater the number of pixels considered, the more accurate the computation is, so there is a trade-off between processing time and quality.
- Only trade-off of higher order methods is edge-preservation.
- Sometimes hybrid methods are used.

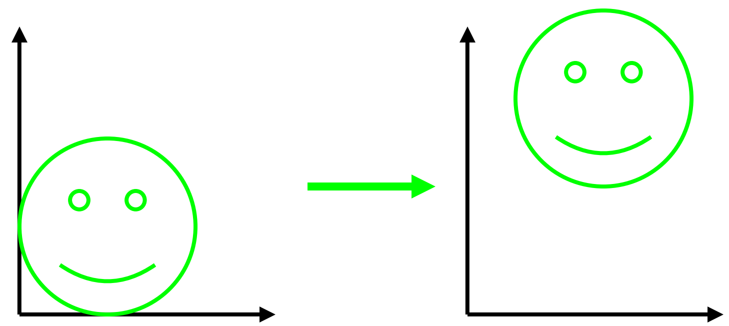
2D Geometric Operations

2D Geometric Operations: Translation

Shifting left-right and/or up-down:

$$x' = x + x_0$$

$$y' = y + y_0$$



Matrix form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + x_0 \\ y + y_0 \\ 1 \end{bmatrix}$$

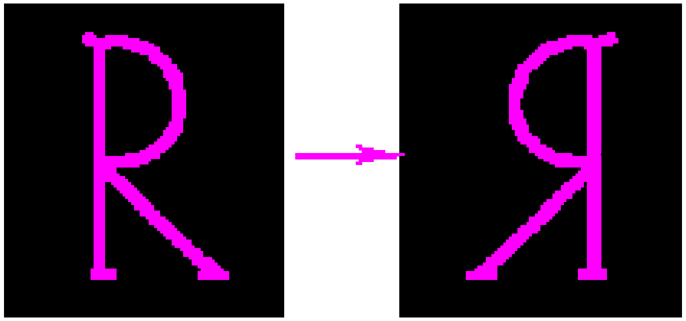
Convenient Notation: *Homogeneous Coordinates*

- Add one dimension, treat transformations as matrix multiplication
- Can be generalized to 3D

2D Geometric Operations: Reflection

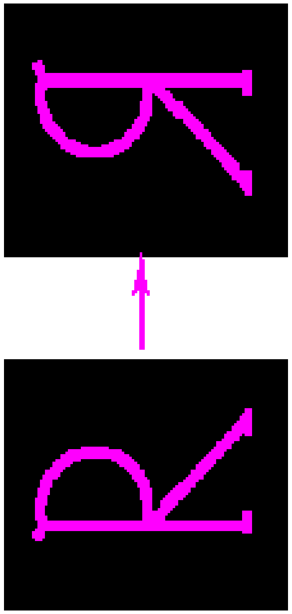
Reflection Y

$$\begin{bmatrix} t'_x \\ t'_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix}$$



Reflection X

$$\begin{bmatrix} t'_x \\ t'_y \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix}$$

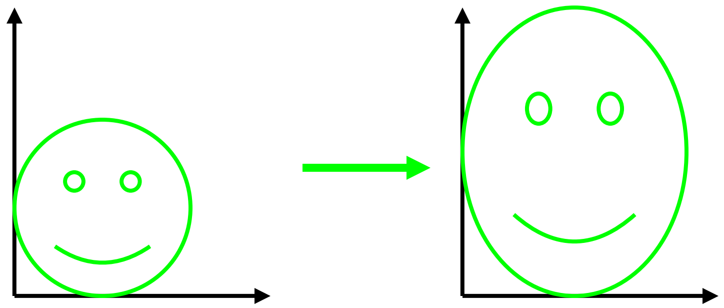


2D Geometric Operations: Scaling

Enlarging or reducing horizontally and/or vertically:

$$x' = S_x x$$

$$y' = S_y y$$



Matrix form:

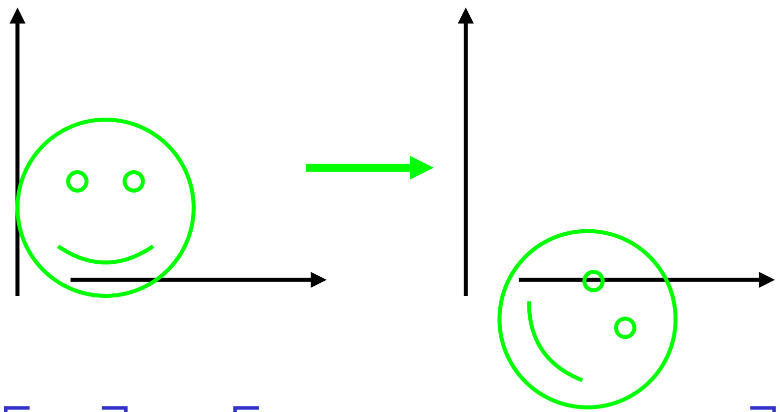
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S_x x \\ S_y y \\ 1 \end{bmatrix}$$

2D Geometric Operations: Rotation

Result components dependent on *both* x & y:

$$x' = \cos(\theta)\,x - \sin(\theta)\,y$$

$$y' = \sin(\theta)\,x + \cos(\theta)\,y$$

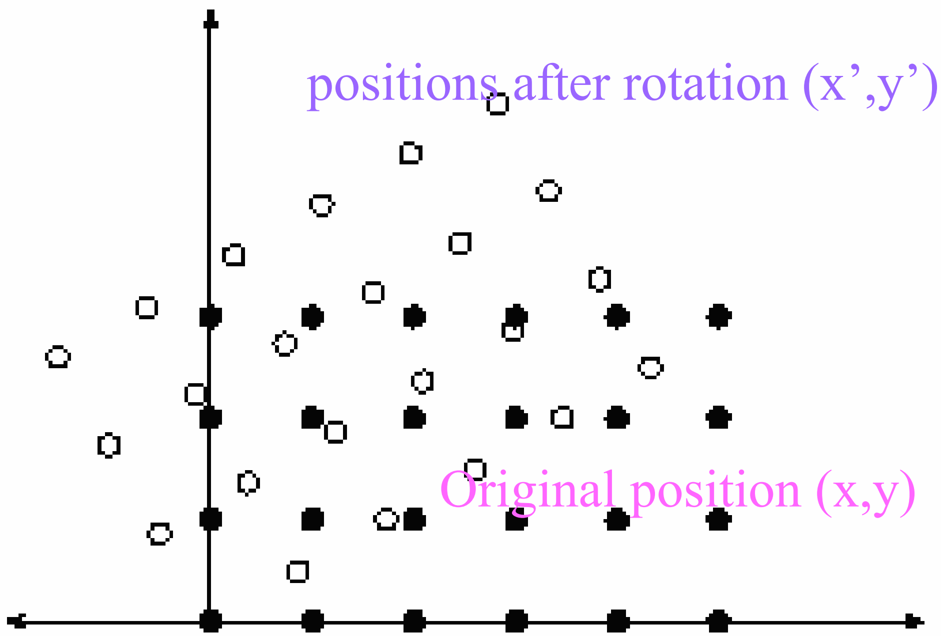


Matrix form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta)\,x - \sin(\theta)\,y \\ \sin(\theta)\,x + \cos(\theta)\,y \\ 1 \end{bmatrix}$$

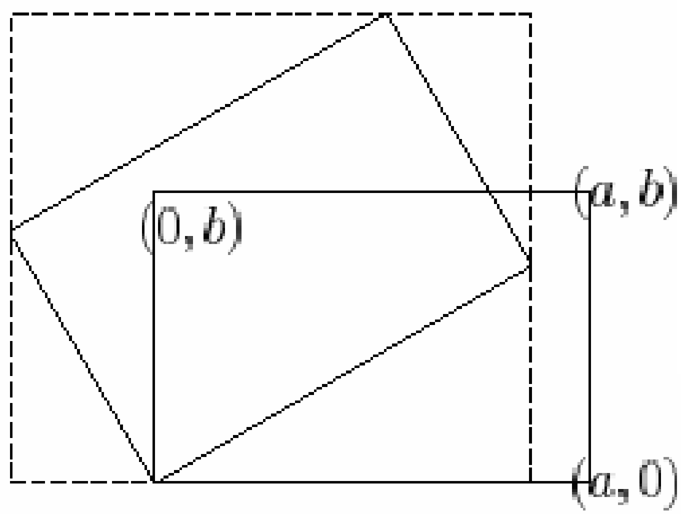
Rotation Operation: Problems

- In image space, when rotating a collection of points, what could go wrong?



Rotation Operation: Problems

- Problem1: part of rotated image might fall out of valid image range.
- Problem2: how to obtain the intensity values in the rotated image?



A rectangle surrounding a rotated image

Consider all integer-valued points (x',y') in the dashed rectangle.

A point will be in the image if, when rotated back, it lies within the original image limits.

$$0 \leq x'\cos\theta + y'\sin\theta \leq a$$

$$0 \leq -x'\sin\theta + y'\cos\theta \leq b$$

See homework assignment 2!

2D Geometric Operations: Affine Transforms

Linear combinations of x, y, and 1: encompasses all translation, scaling, & rotation (also skew and shear):

$$x' = ax + by + c$$

$$y' = dx + ey + f$$

Matrix form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

Affine Transformations (cont.)

- Translations and rotations are *rigid body* transformations
- General affine transformations also include non-rigid transformations (e.g., skew or shear)
  - Affine means that parallel lines transform to parallel lines

Compound Transformations

Example: rotation around the point (x_0, y_0)

$$\begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Matrix multiplication is associative (but not commutative):

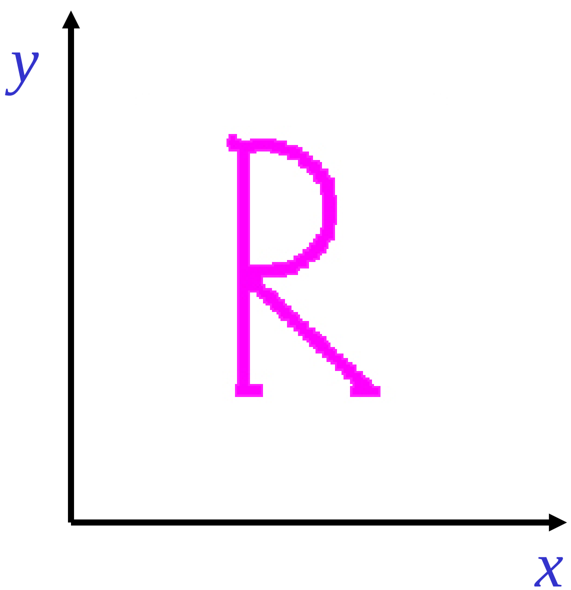
$$\boldsymbol{B}(\boldsymbol{A}\boldsymbol{v}) = (\boldsymbol{B}\boldsymbol{A})\boldsymbol{v} = \boldsymbol{C}\boldsymbol{v}$$

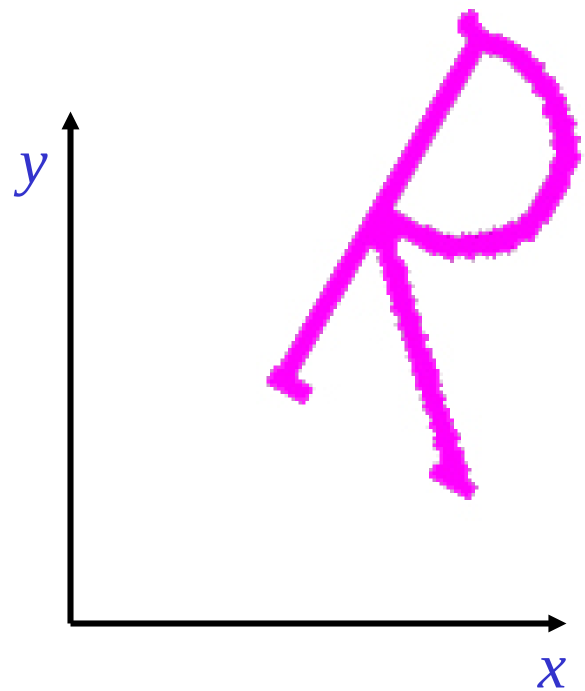
where $C = BA$

- Can compose multiple transformations into a single matrix
- Much faster when applying same transform to many pixels

Compound Transformations

Example:





Inverting Matrix Transformations

If

$$v' = M v$$

then

$$v = M^{-1} v'$$

Thus, to invert the transformation, invert the matrix

Useful for computing the backward mapping given the forward transform

For more info see e.g. http://home.earthlink.net/~jimlux/radio/math/matinv.htm

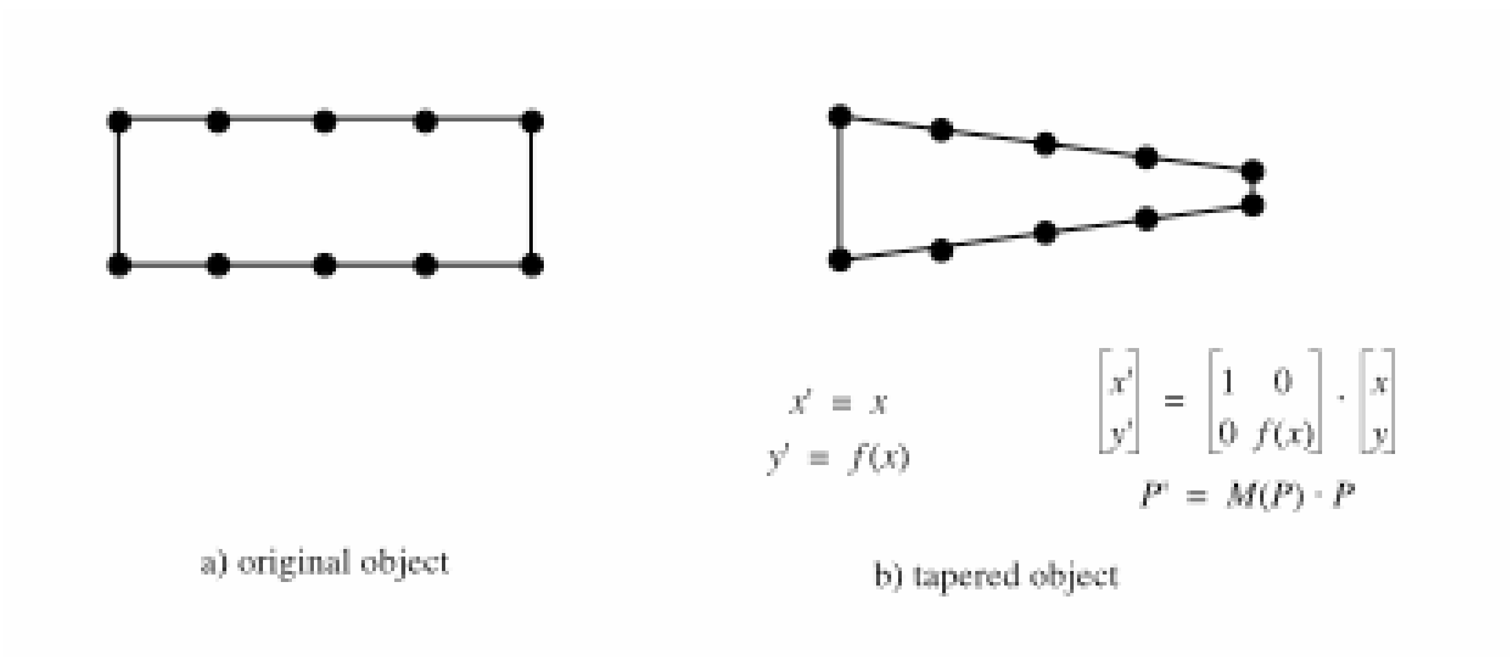
Morphing: Deformations in 2D and 3D

Morphing: Deformations in 2D and 3D

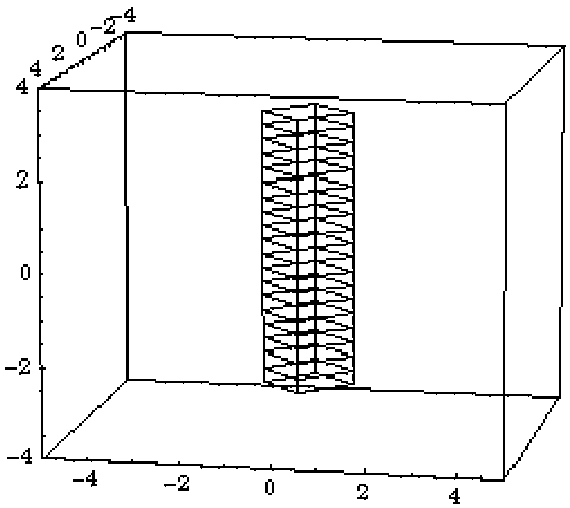
- Parametric Deformations
- Cross-Dissolve
- Mesh Warping
- Control Points

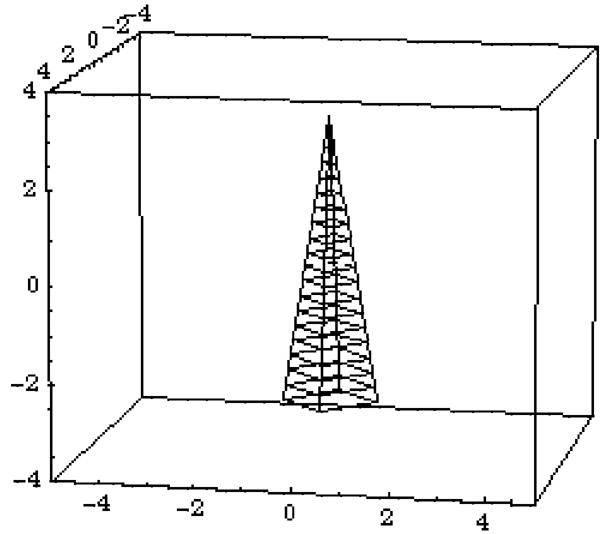
Parametric Deformations

Parametric Deformations - Taper



Parametric Deformations - Taper



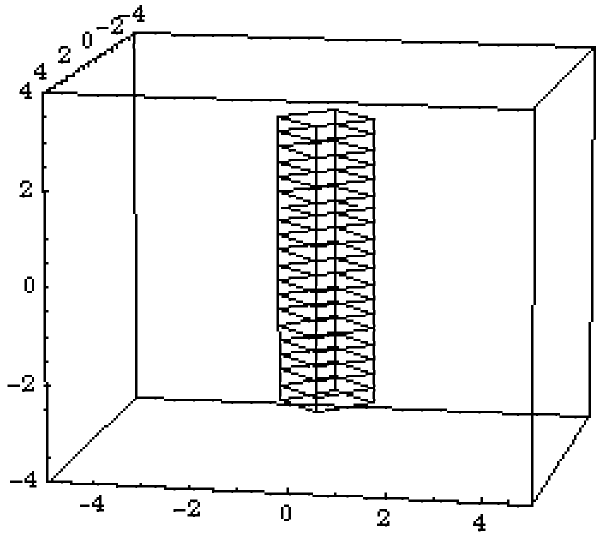


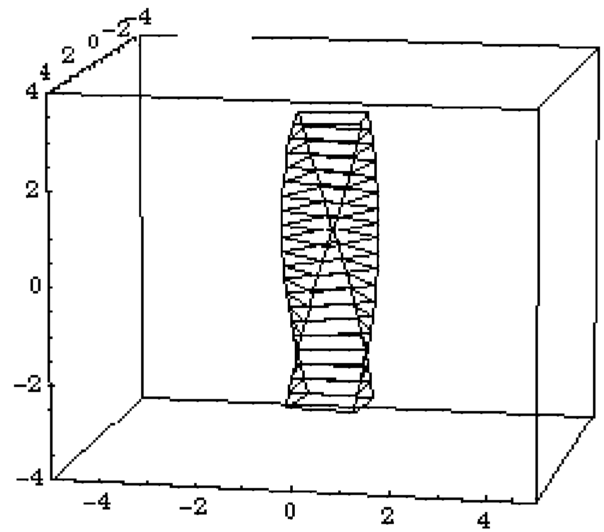
Parametric Deformations - Twist

$$x' = s(z) \cdot x$$
$$y' = s(z) \cdot y$$
$$z' = z$$

Where $s(z) = \dfrac{(maxz - z)}{(maxz - minz)}$

Parametric Deformations - Twist





Parametric Deformations - Bend

y_0 - center of bend
1/k - radius of bend
y_{min}:y_{max} - bend region

$$\hat{y} = \begin{cases} y_{min} & y \le y_{min} \\ y & y_{min} < y < y_{max} \\ y_{max} & y \ge y_{max} \end{cases}$$

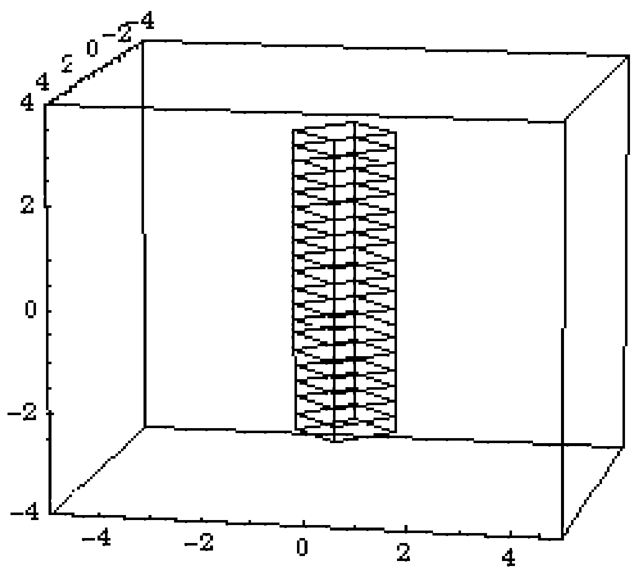
$$\theta = k \cdot (\hat{y} - y_0)$$
$$C_\theta = \cos\theta$$
$$S_\theta = \sin\theta$$

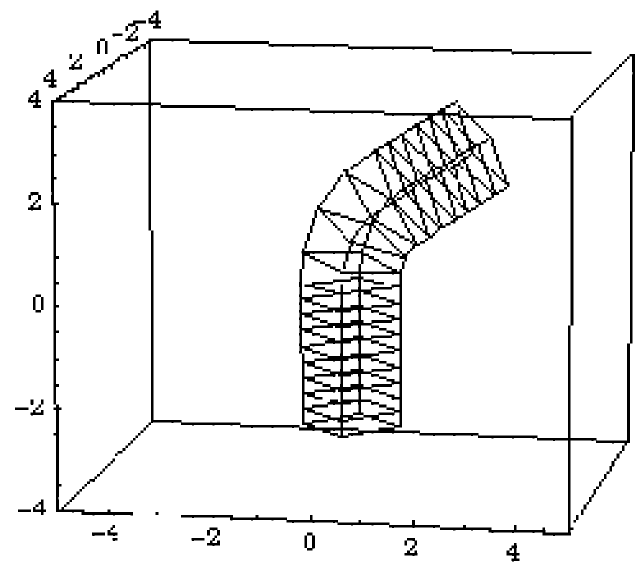
$$x' = x$$

$$y' = \begin{cases} -S_\theta \cdot z - \frac{1}{k} + y_0 & y_{min} \le y \le y_{max} \\ -\left(S_\theta \cdot \left(z - \frac{1}{k}\right)\right) + y_0 + C_\theta \cdot (y - y_{min}) & y < y_{min} \\ \left(-\left(S_\theta \cdot \left(z - \frac{1}{k}\right)\right) + y_0 + C_\theta \cdot (y - y_{max})\right) & y > y_{max} \end{cases}$$

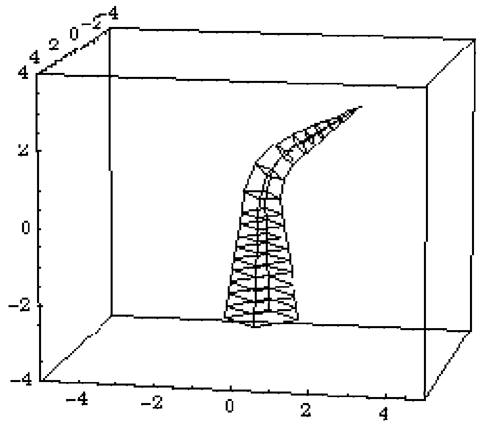
$$z' = \begin{cases} -C_\theta \cdot z - \frac{1}{k} + \frac{1}{k} & y_{min} \le y \le y_{max} \\ -\left(C_\theta \cdot \left(z - \frac{1}{k}\right)\right) + \frac{1}{k} + S_\theta \cdot (y - y_{min}) & y < y_{min} \\ \left(-\left(C_\theta \cdot \left(z - \frac{1}{k}\right)\right) + \frac{1}{k} + S_\theta \cdot (y - y_{max})\right) & y > y_{max} \end{cases}$$

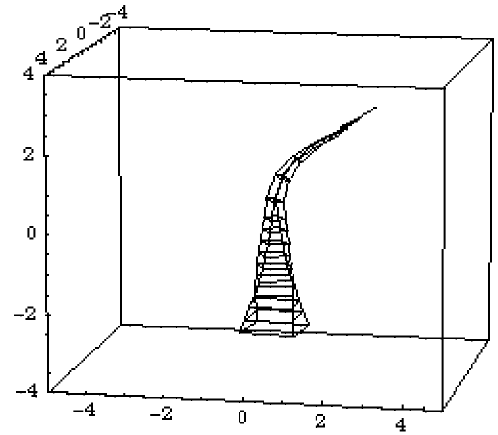
Parametric Deformations - Bend

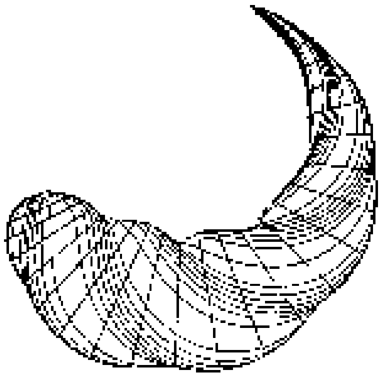




Parametric Deformations - Compound







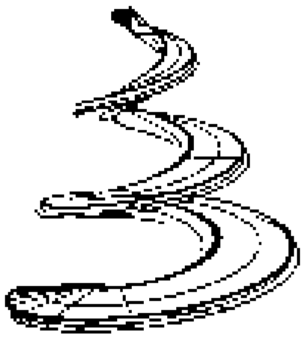


Image Blending

Image Blending

- Goal is smooth transformation between image of one object and another
- The idea is to get a sequence of intermediate images which when put together with the original images would represent the change from one image to the other
- Realized by
  - Image warping
  - Color blending
- Image blending has been widely used in creating movies, music videos and television commercials
  - Terminator 2

Cross-Dissolve (Cross-Fading)

- Simplest approach is **cross-dissolve**:
  - linear interpolation to fade from one image (or volume) to another
- No geometrical alignment between images (or volumes)
- Pixel-by-pixel (or voxel by voxel) interpolation
- No smooth transitions, intermediate states not realistic

from G. Wolberg, CGI '96

Problems

- Problem with cross-dissolve is that if features don't line up exactly, we get a double image
- Can try shifting/scaling/etc. one **entire** image to get better alignment, but this doesn't always fix problem
- Can handle more situations by applying different warps to different **pieces** of image
  - Manually chosen
  - Takes care of feature correspondences

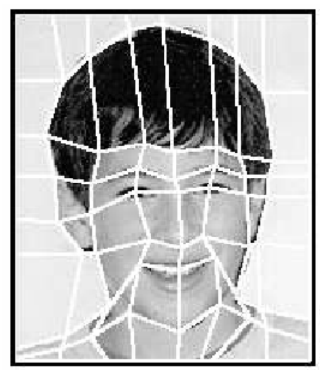


Image $\mathbf{I}_S$ with mesh $\mathbf{M}_S$ defining pieces

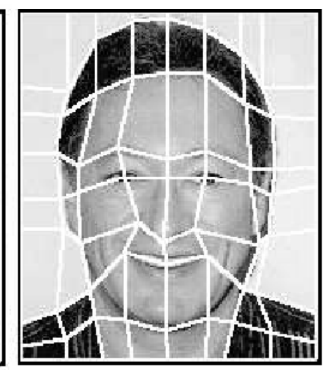


Image $\mathbf{I}_T$, mesh $\mathbf{M}_T$

from G. Wolberg, CGI '96

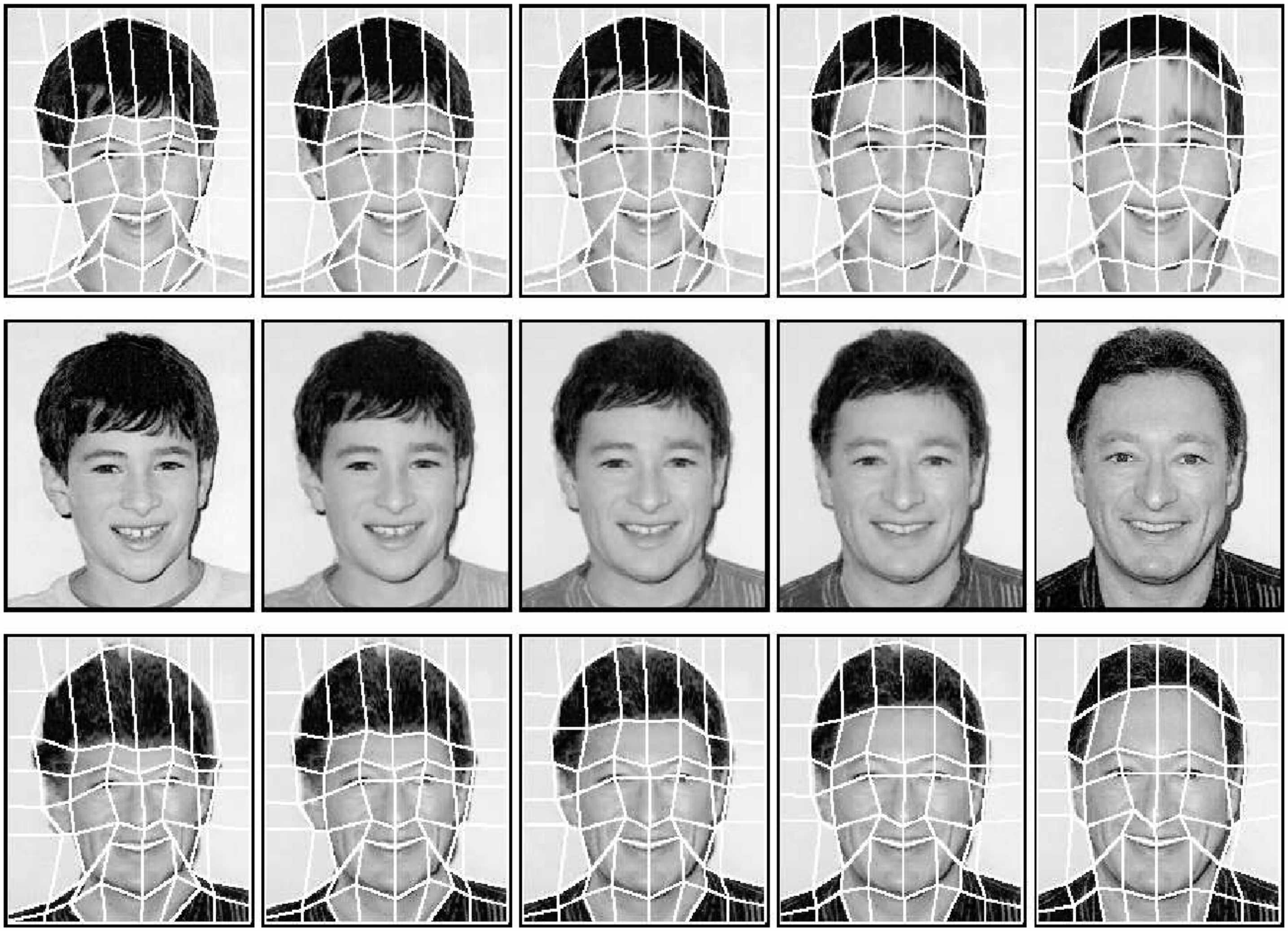
Mesh Warping

Mesh Warping Application

Images $\mathbf{I}_S$
& meshes $\mathbf{M}_S$

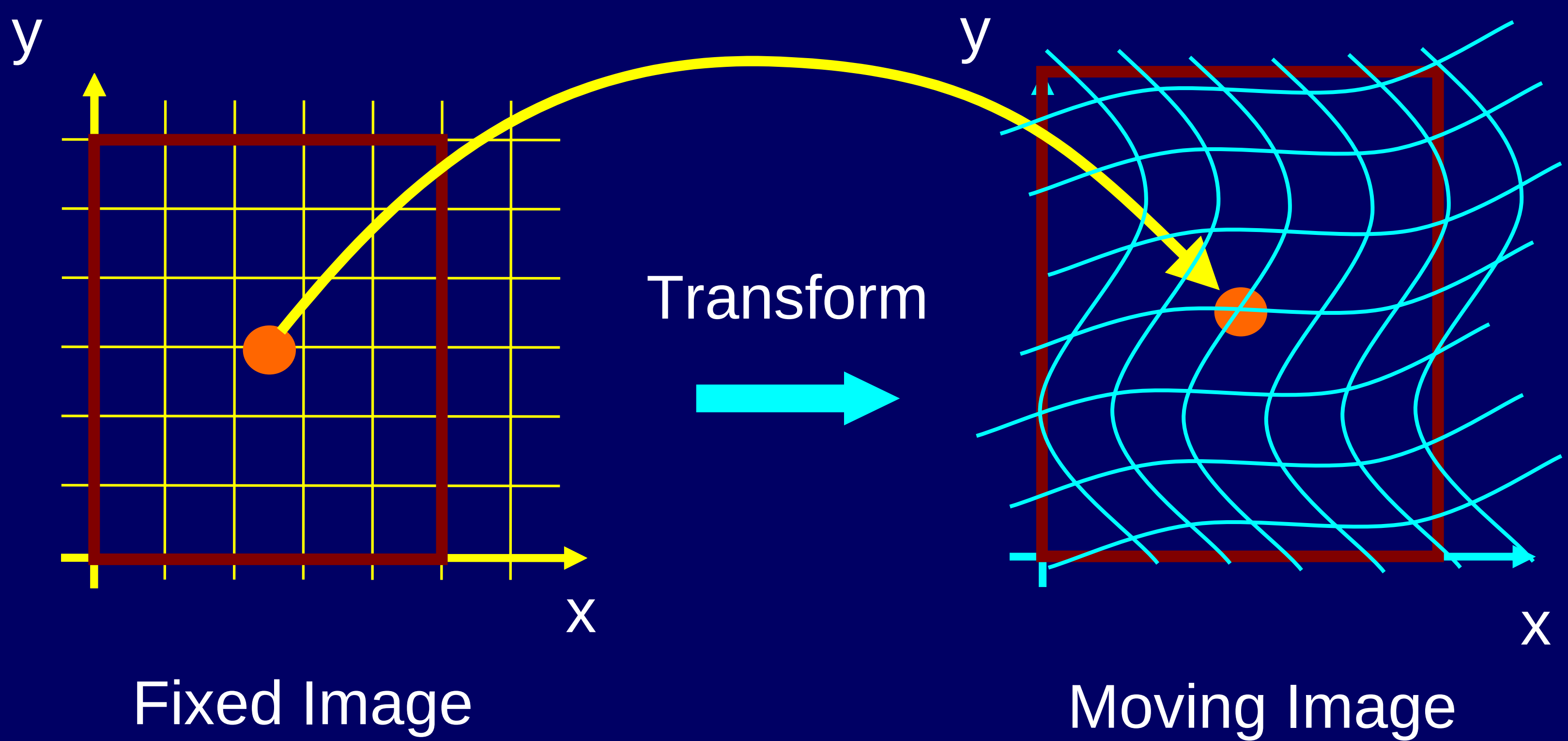
Morphed
images $\mathbf{I}^f$

Images $\mathbf{I}_T$
& meshes $\mathbf{M}_S$

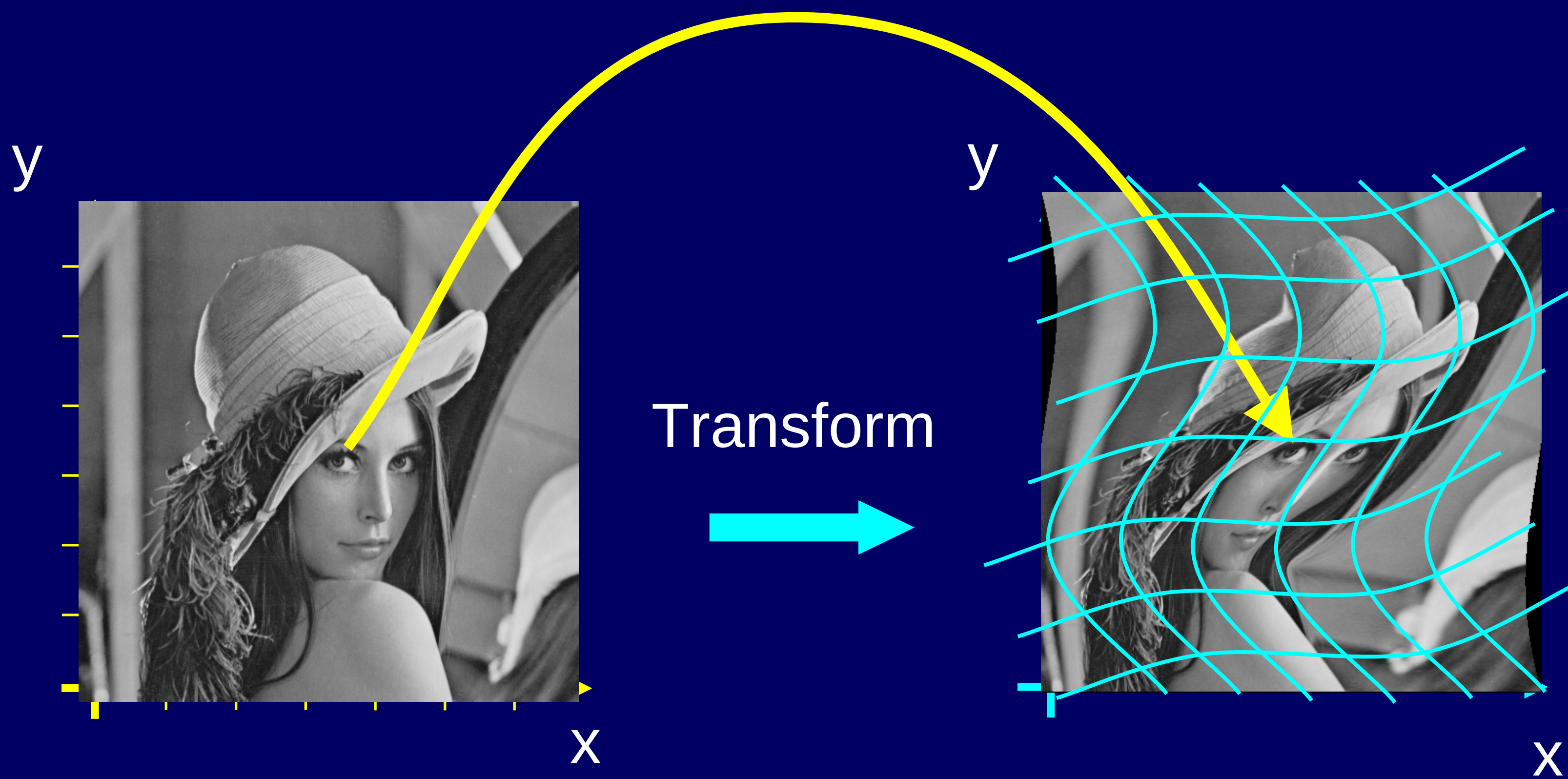


from G. Wolberg, CGI '96

Mesh Warping



Mesh Warping



Fixed Image

Moving Image

Mesh Warping

- Source and target images are meshed
- The meshes for both images are interpolated
- The intermediate images are cross-dissolved
- Here, we look at 2D example

Mesh Warping Algorithm

- Algorithm

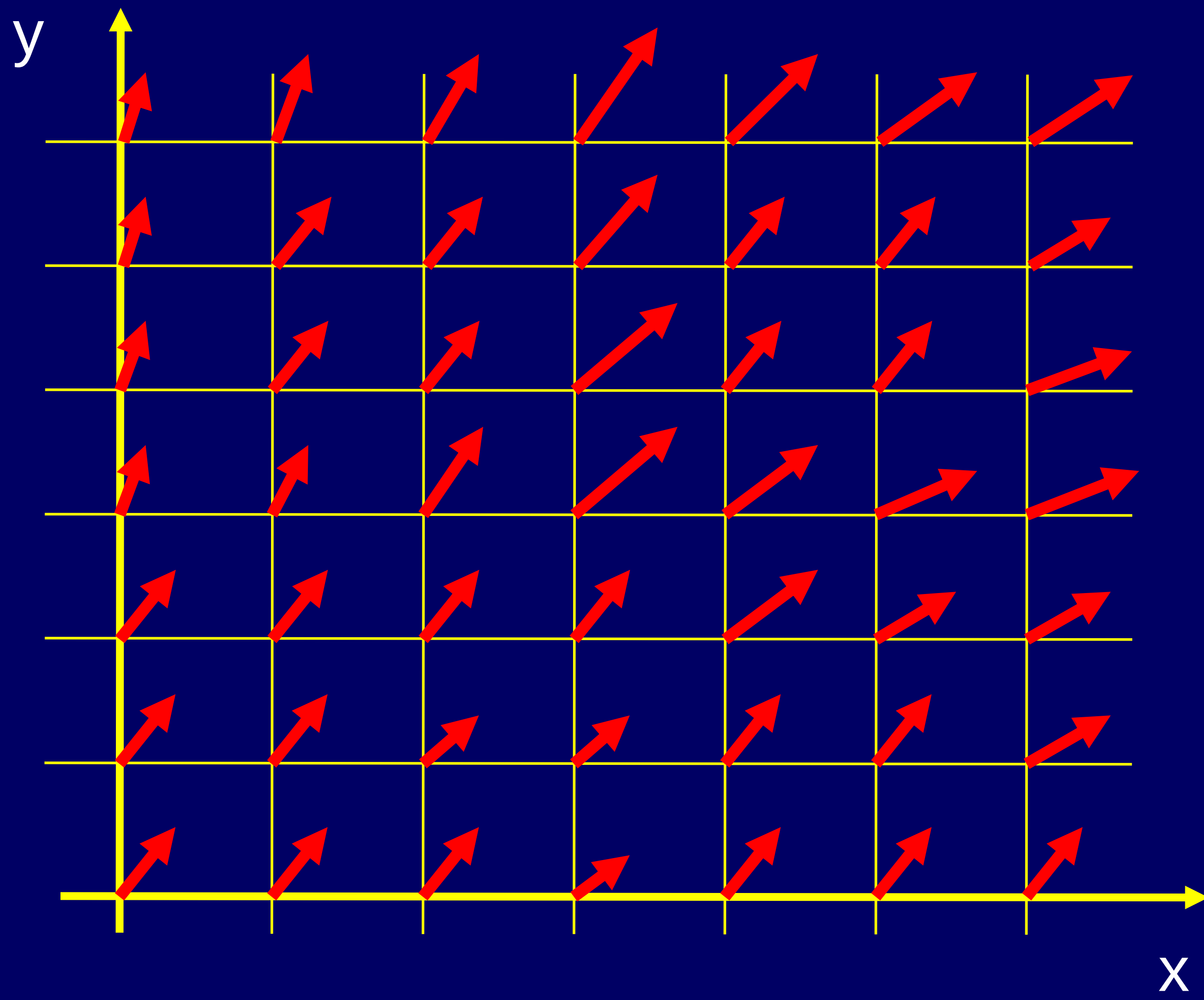
for each frame *f* **do**

- interpolate mesh M, between M_s and M_t
- warp Image I_s to I_1, using meshes M_s and M
- warp Image I_t to I_2, using meshes M_t and M
- interpolate image I_1 and I_2

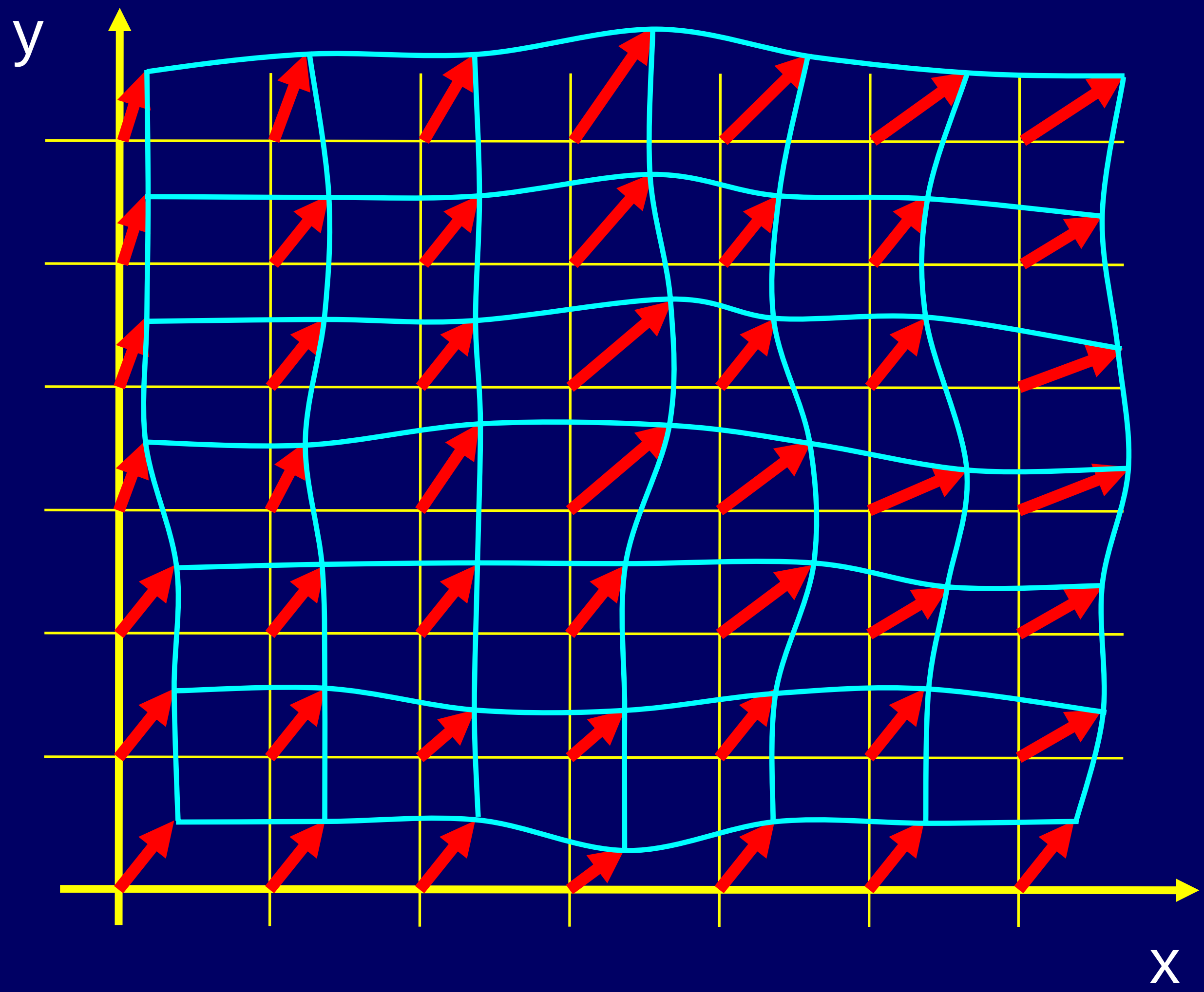
end

- I_s : source image, I_t : target image
- source image has mesh M_s, target image has mesh M_t

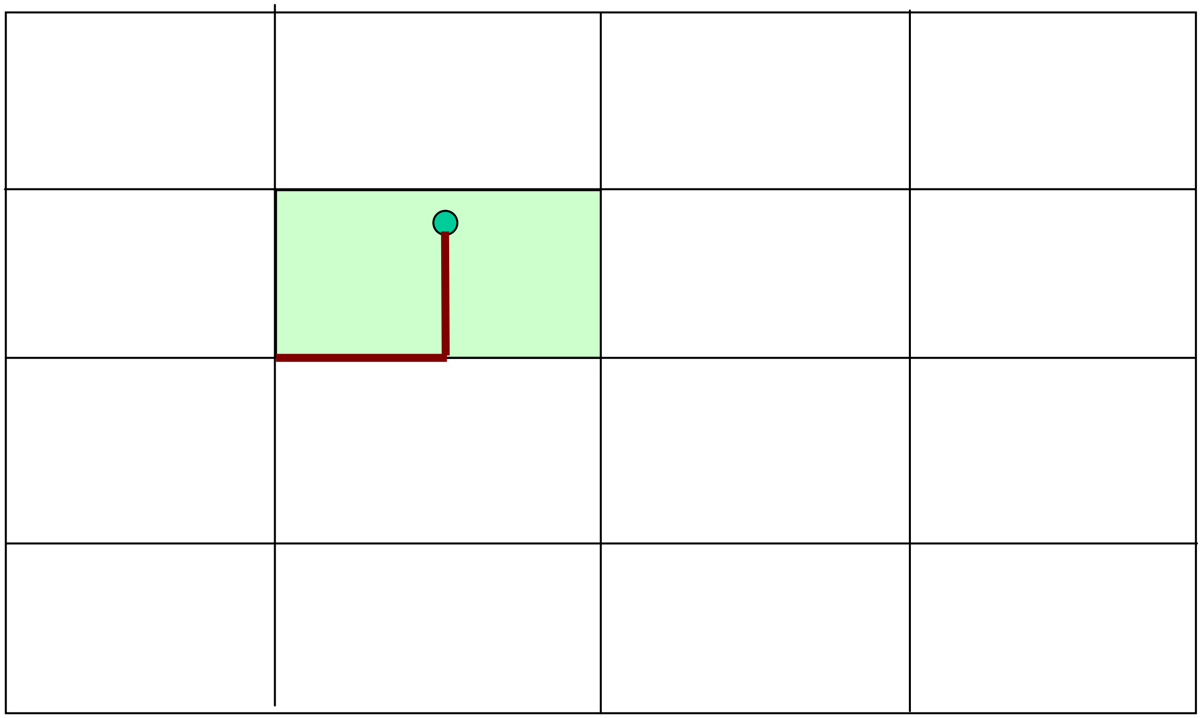
Mesh Deformation



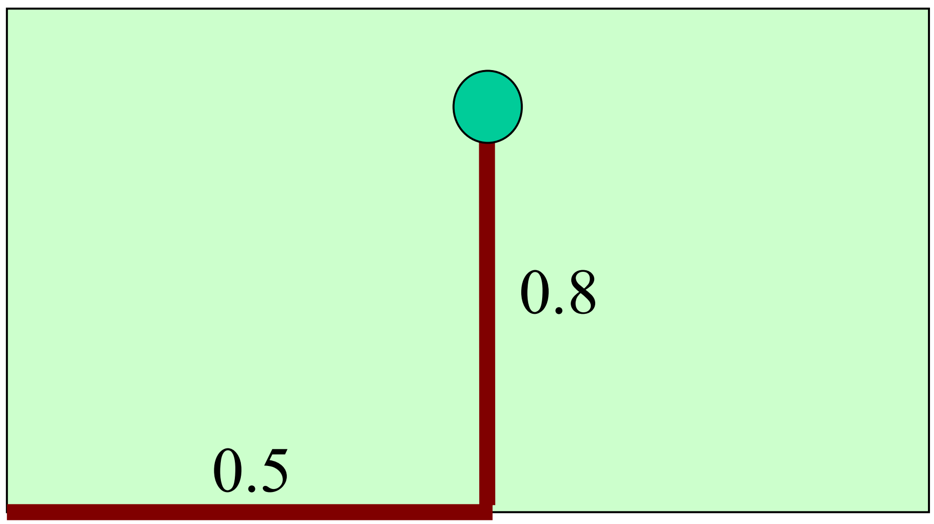
Mesh Deformation



Mesh Deformation

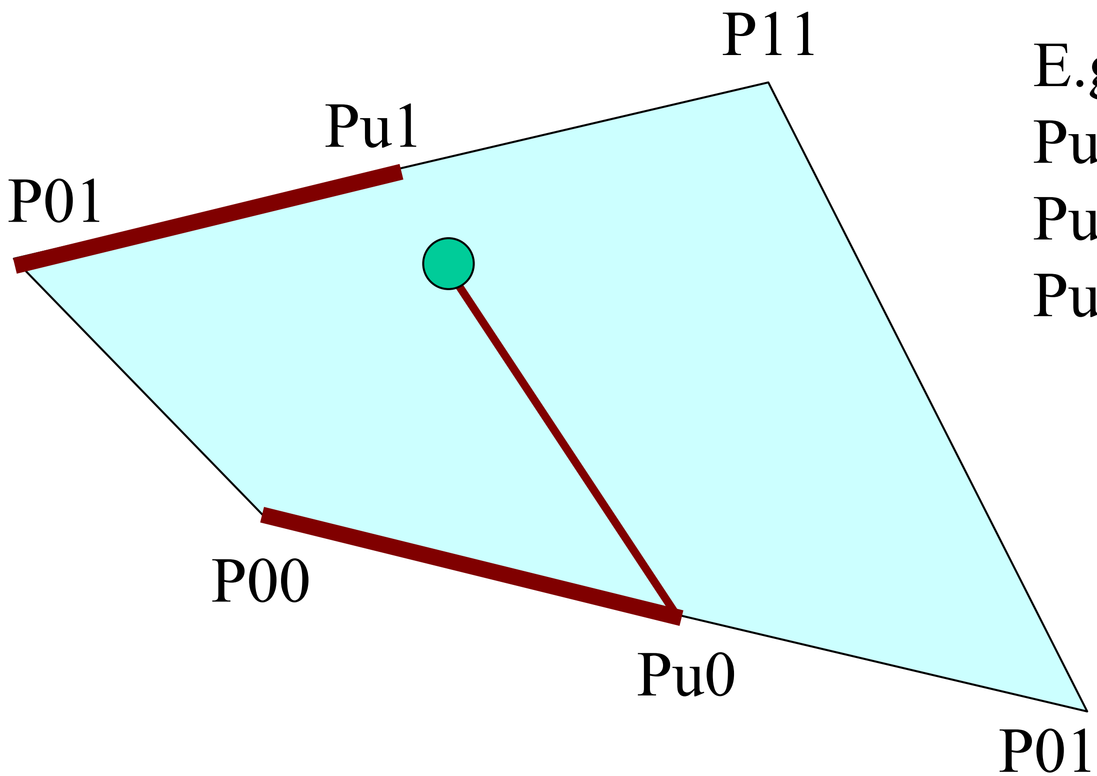


Mesh Deformation



For each vertex
identify cell,
fractional u,v
coordinate
in unit cell

Mesh Deformation



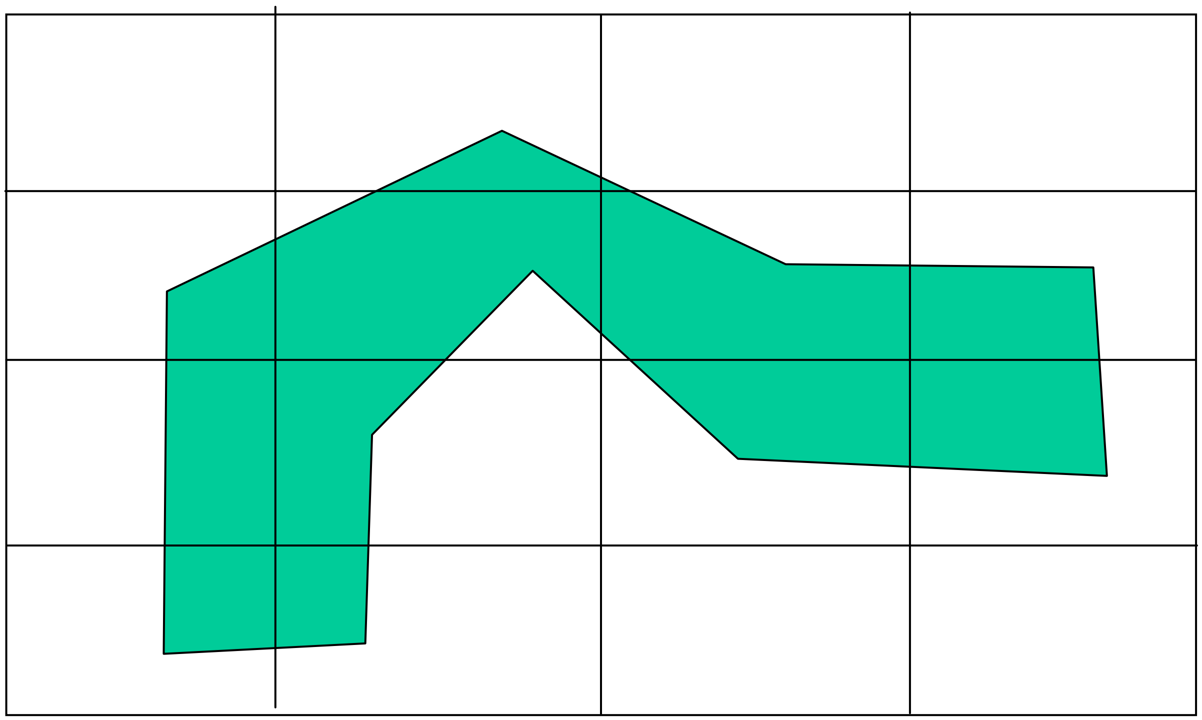
E.g. bilinear interpolation

Pu0 = (1-u)*P00 + u*P10

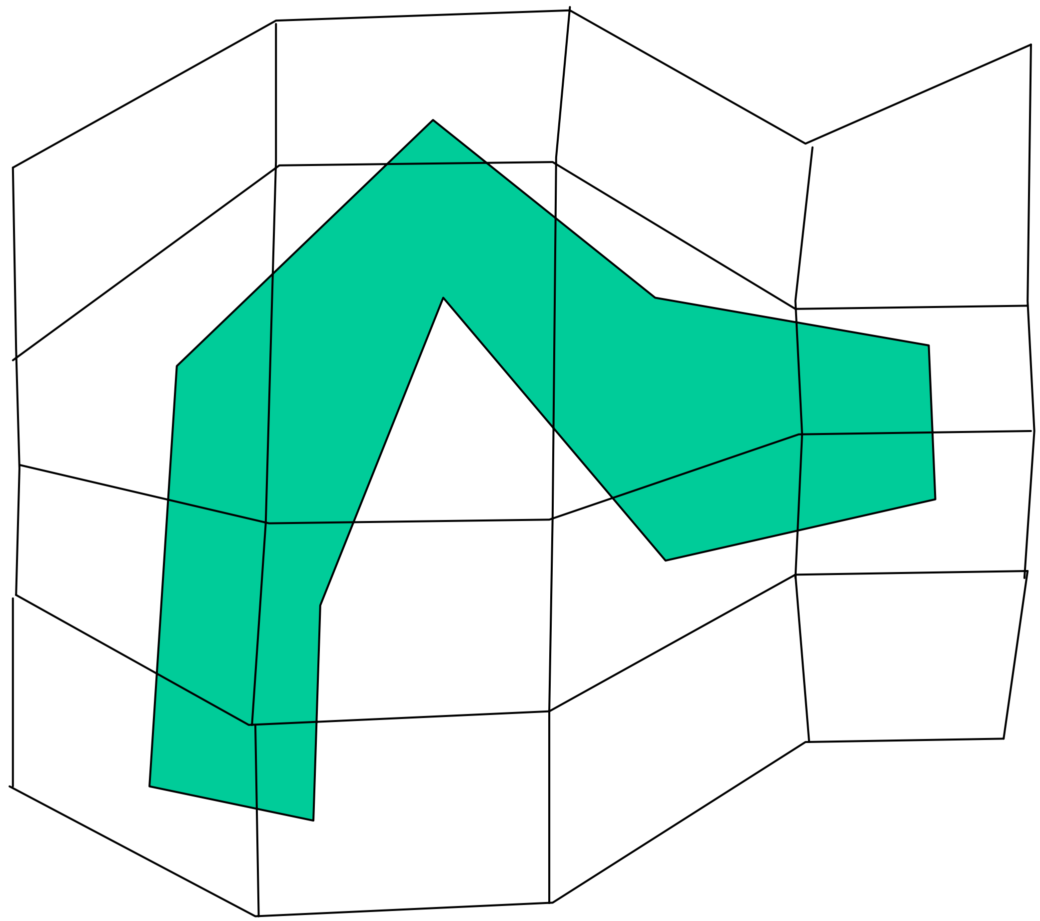
Pu1 = (1-u)*P01 + u*P11

Puv = (1-v)*P0u + v*P1u

Mesh Deformation

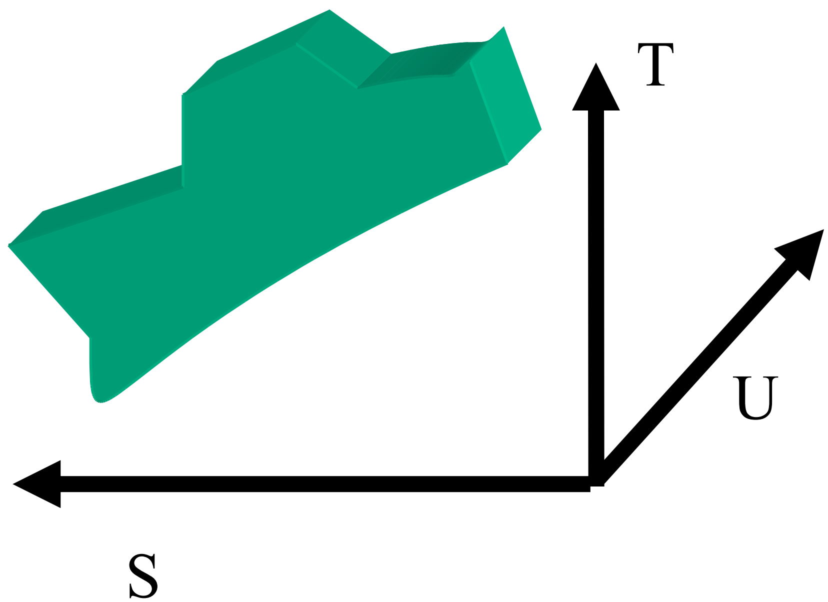


Mesh Deformation



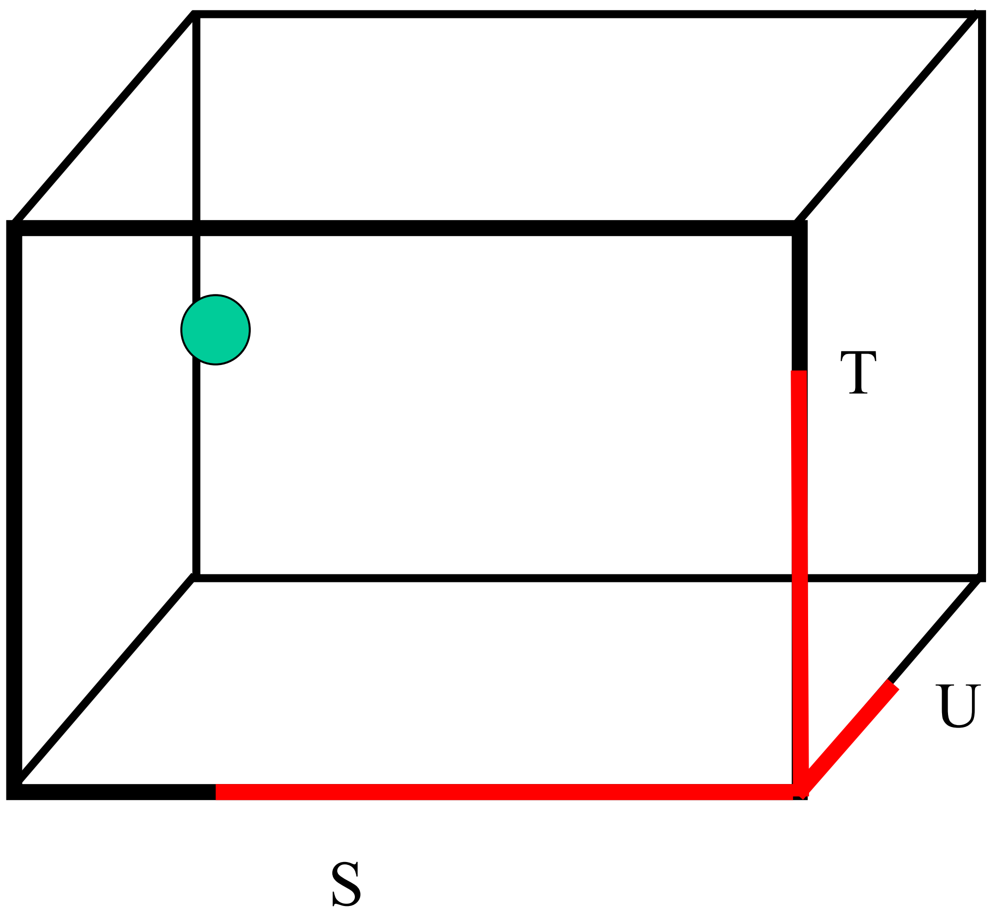
Free-Form Deformations

Define local coordinate system for deformation

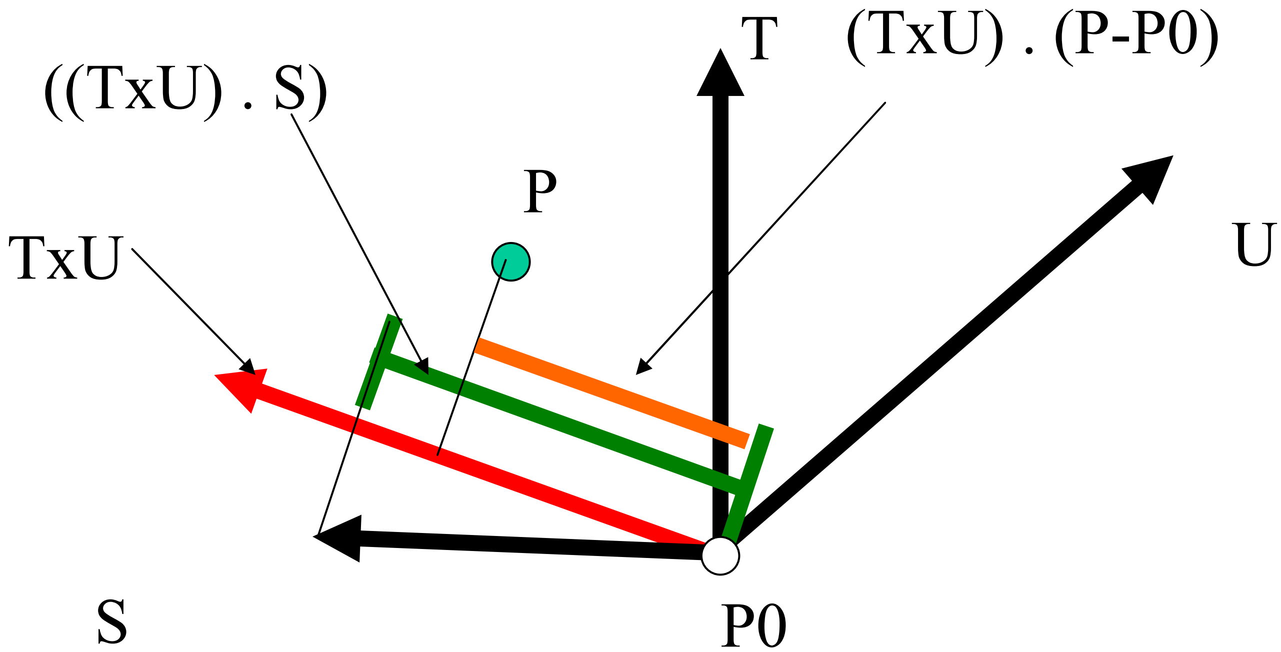


(not necessarily orthogonal)

FFD – Register Point in Cell



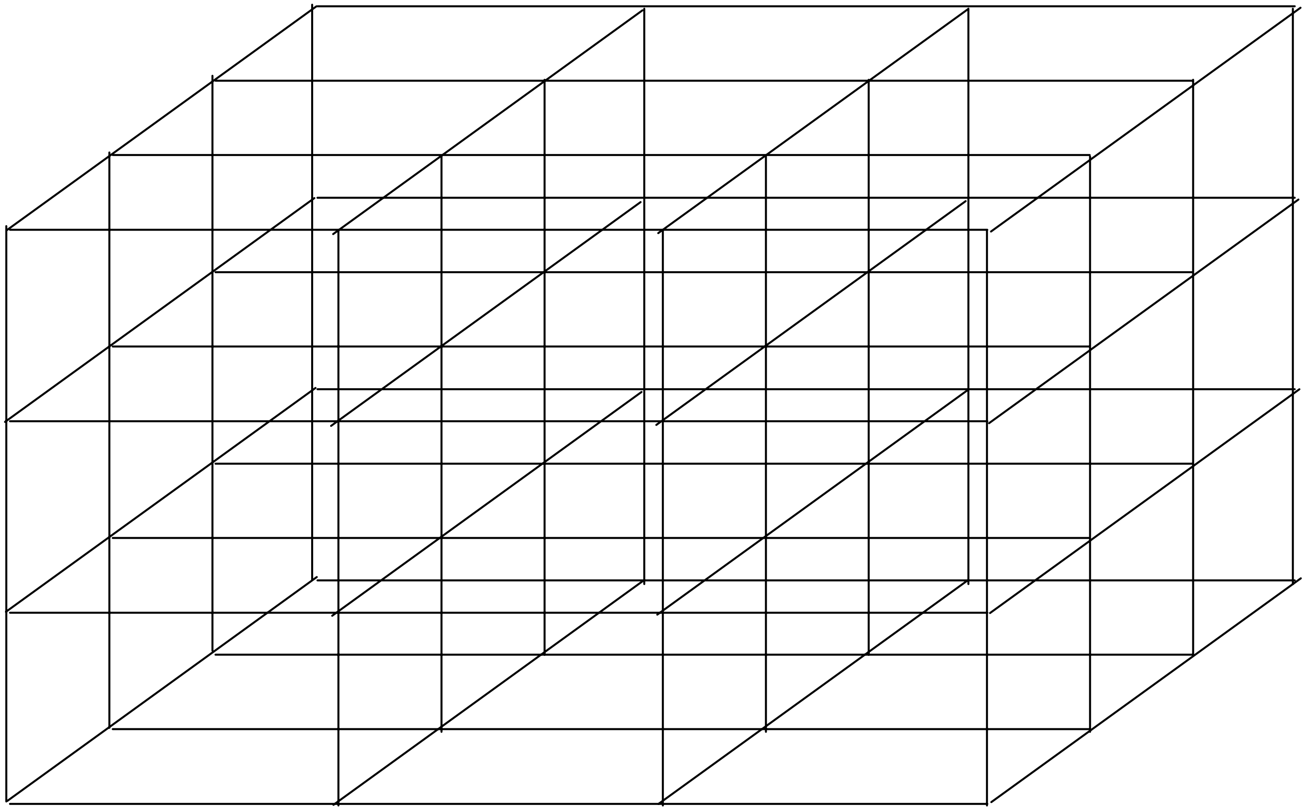
FFD – Register Point in Cell



s = (TxU) . (P-P0) / ((TxU) . S)

P = P0 + sS + tT + uU

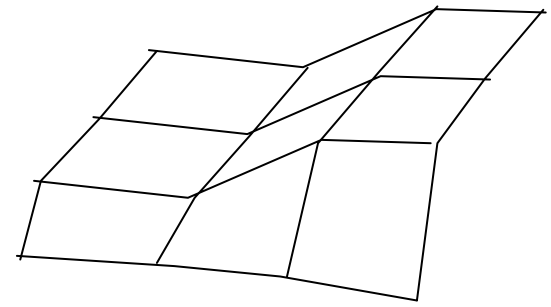
FFD – Create Control Grid



(not necessarily orthogonal)

FFD – Move and Reposition

Move control grid points



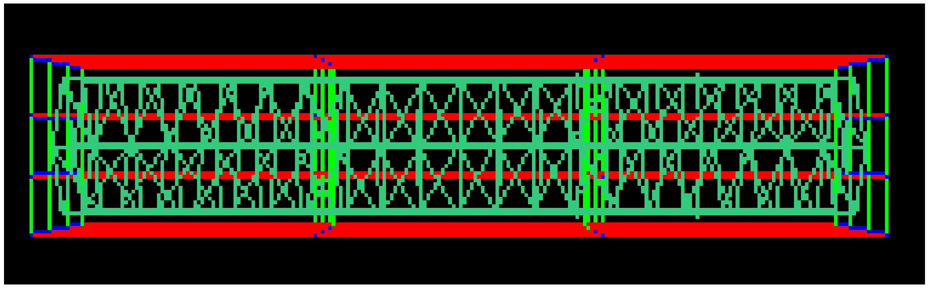
Usually tri-cubic interpolation is used with FFDs

Originally, Bezier interpolation was used.

B-spline and Catmull-Romm interpolation have also been used (as well as tri-linear interpolation)

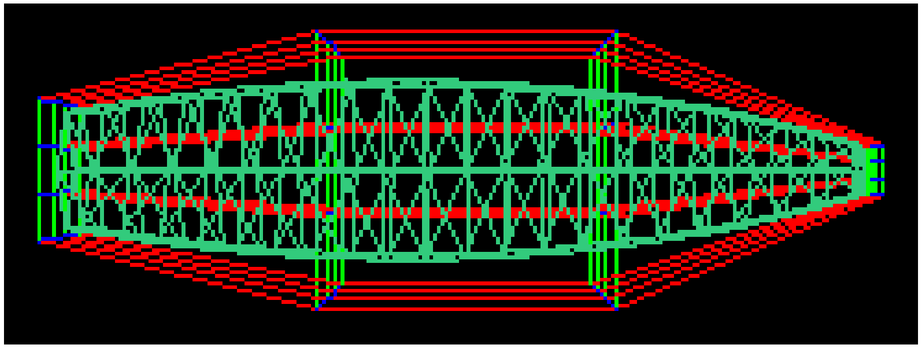
FFD Example

Step1



It is originally a cylinder.
Red boundary is FFD block embedded with that cylinder.

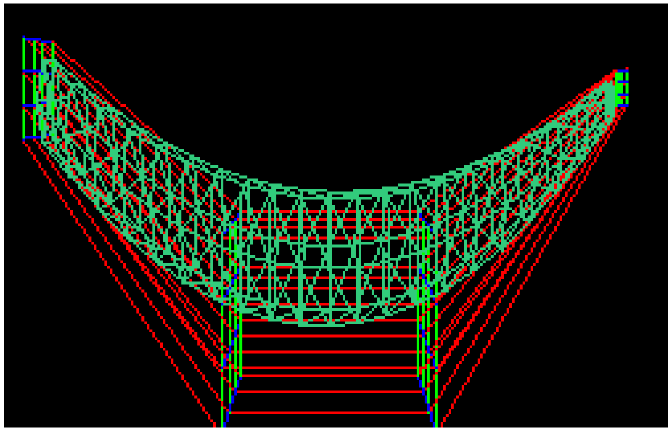
Step2



Move control points of each end, and you can see cylinder inside also changes.

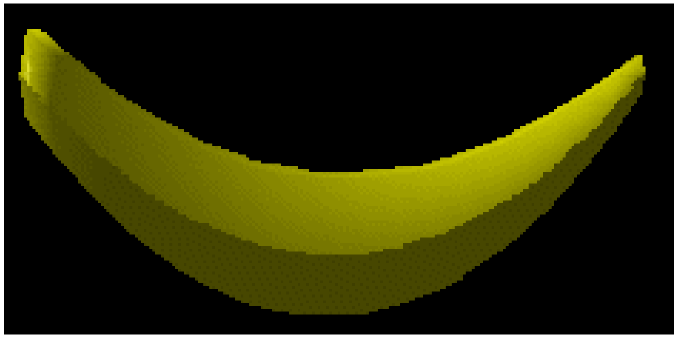
FFD Example

step3



Move inner control points downwards.

step4



Finally, get a shaded version of banana!

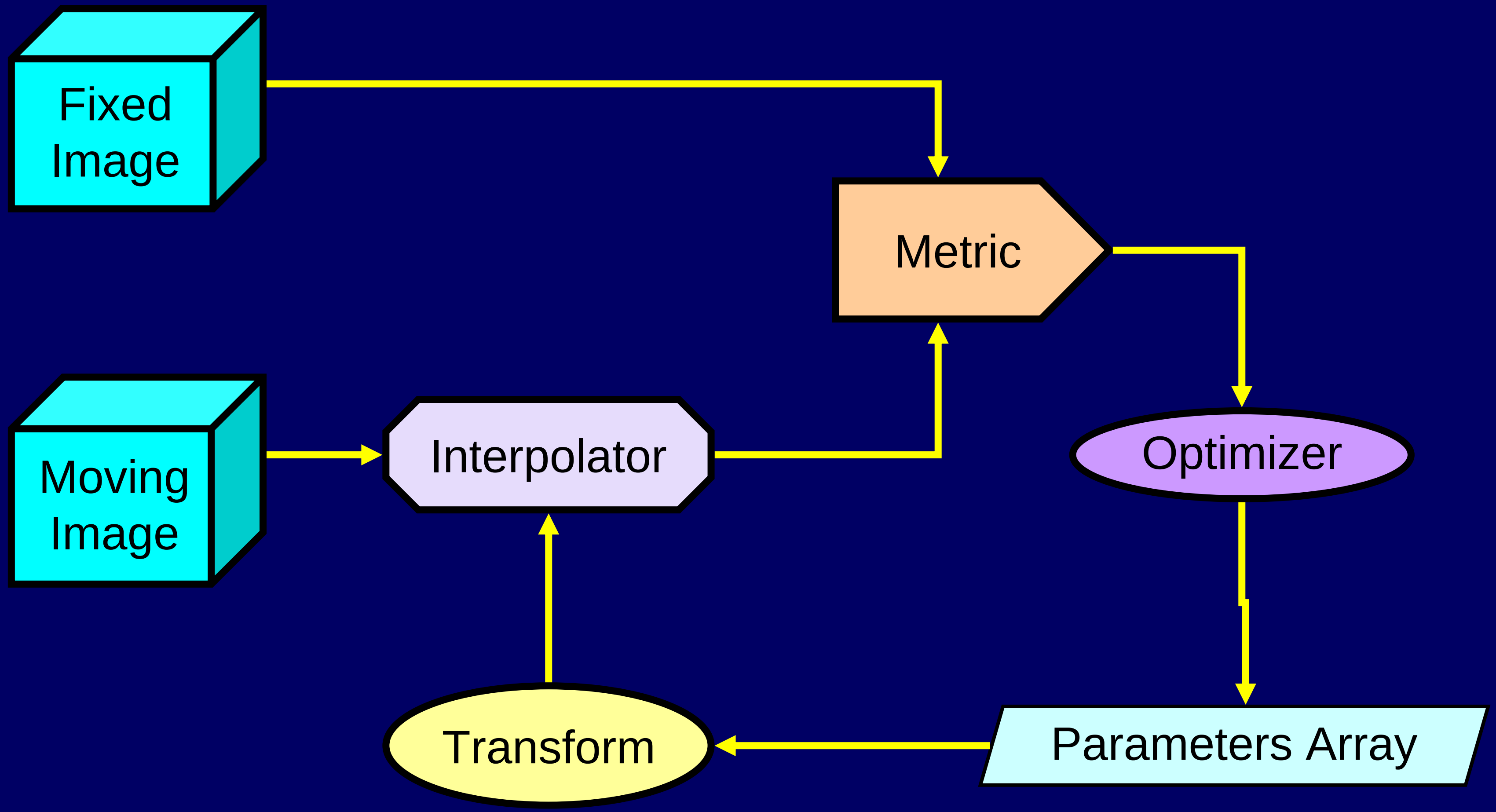
BSplines (Cubic) Interpolation

Original Lena

BSplines (Cubic) Interpolation

Deformed with BSpline Transform

Deformable Registration Framework



Deformable Registration

Deformed with BSpline Transform

Deformable Registration

Registered with BSpline Transform

Deformable Registration

Original Lena

Deformable Registration

Difference Before Registration

Difference After Registration

Control Point Warping

Control Point Warping

Instead of a warping mesh, use arbitrary correspondence points:

- Tip of one person's nose to the tip of another, eyes to eyes, etc.

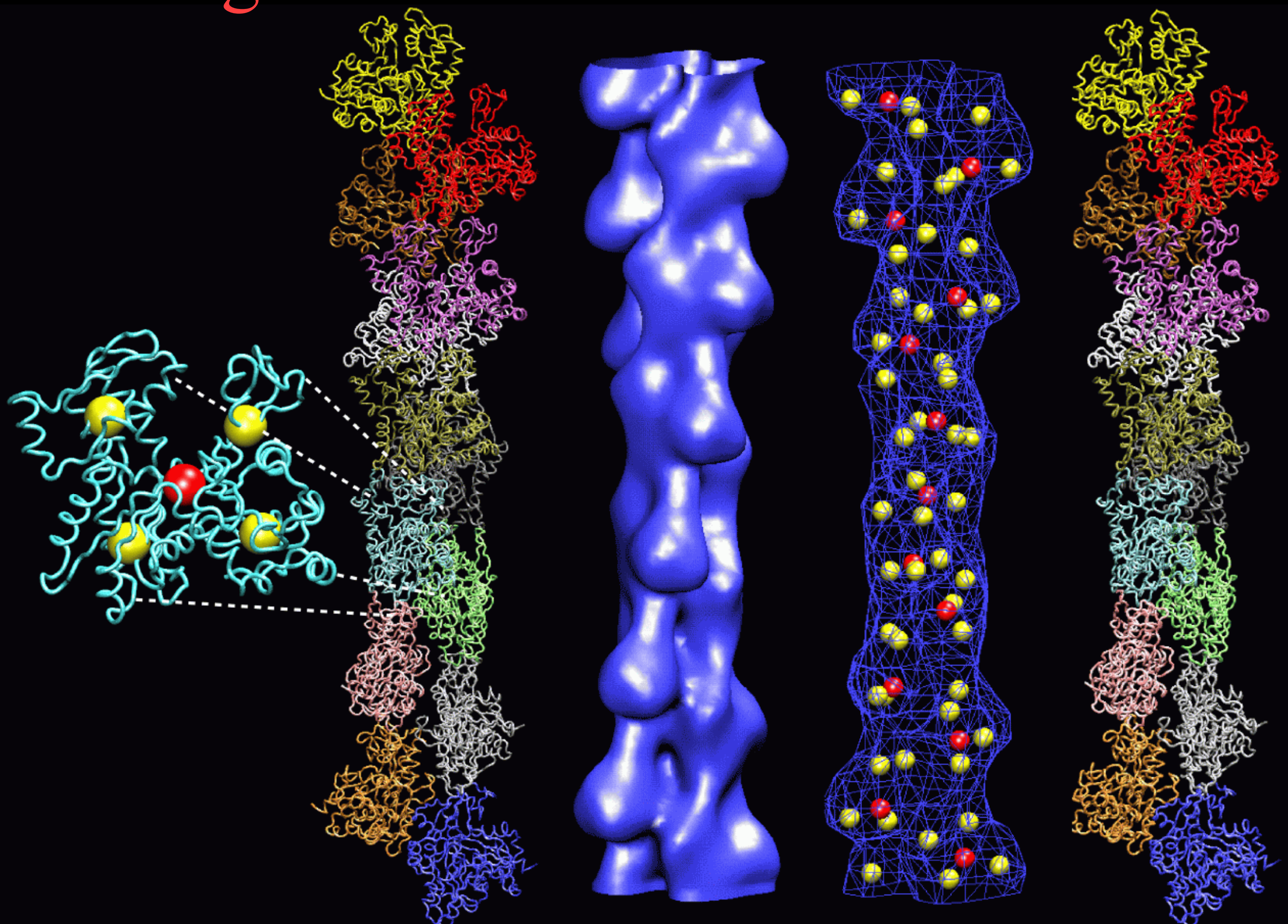
Interpolate between correspondence points to determine how points move

Apply standard warping (forward or backward):

- In-between image is a weighted average of the source and destination corresponding pixels

Here we look at 3D example…

Finding Control Points in 3D Structures

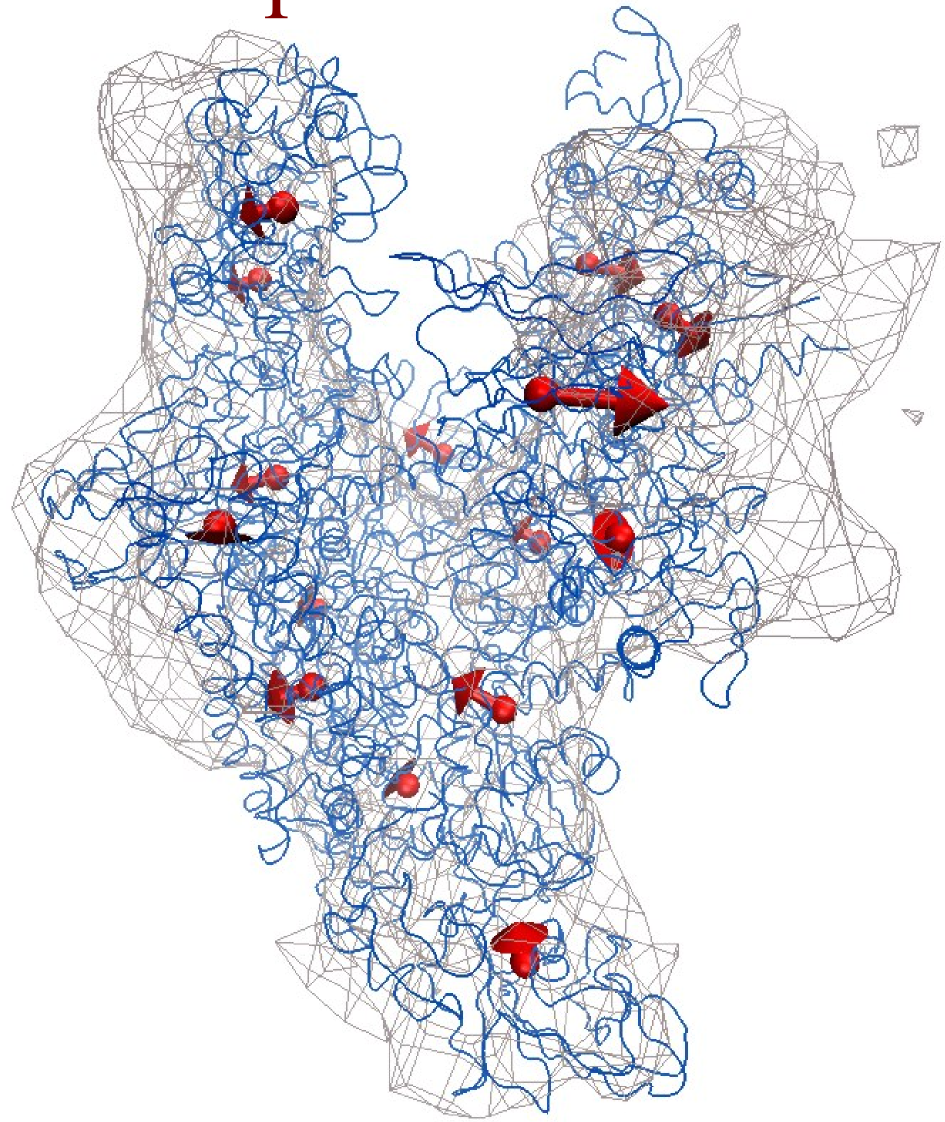


Actin filament: Reconstruction from EM data at 20Å resolution rmsd: 1.1Å

Wriggers et al., J. Molecular Biology, 1998, 284: 1247-1254

Control Point Displacements

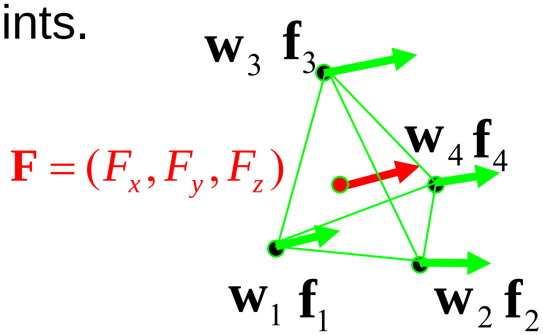
Have 2 conformations, both source and target characterized by control points



RNA Polymerase, Wriggers, Structure, 2004, Vol. 12, pp. 1-2.

Piecewise-Linear Inter- / Extrapolation

For each probe position find 4 closest control points.



Ansatz:
$$F_x(x,y,z) = ax + by + cz + d$$
$$F_x(\mathbf{w}_1) = f_{1,x},$$
$$F_x(\mathbf{w}_2) = f_{2,x},$$
$$F_x(\mathbf{w}_3) = f_{3,x},$$
$$F_x(\mathbf{w}_4) = f_{4,x} \quad \text{(similar for } F_y, F_z\text{)}.$$

Cramer's rule:

$$a = \frac{\begin{vmatrix} f_{1,x} & w_{1,y} & w_{1,z} & 1 \\ f_{2,x} & w_{2,y} & w_{2,z} & 1 \\ f_{3,x} & w_{3,y} & w_{3,z} & 1 \\ f_{4,x} & w_{4,y} & w_{4,z} & 1 \end{vmatrix}}{D}, \quad b = \frac{\begin{vmatrix} w_{1,x} & f_{1,y} & w_{1,z} & 1 \\ w_{2,x} & f_{2,y} & w_{2,z} & 1 \\ w_{3,x} & f_{3,y} & w_{3,z} & 1 \\ w_{4,x} & f_{4,y} & w_{4,z} & 1 \end{vmatrix}}{D}, \quad \cdots, \quad D = \begin{vmatrix} w_{1,x} & w_{1,y} & w_{1,z} & 1 \\ w_{2,x} & w_{2,y} & w_{2,z} & 1 \\ w_{3,x} & w_{3,y} & w_{3,z} & 1 \\ w_{4,x} & w_{4,y} & w_{4,z} & 1 \end{vmatrix}$$

See e.g. http://mathworld.wolfram.com/CramersRule.html

Non-Linear Kernel Interpolation

Consider all k control points and interpolation kernel function $U(r)$.

Ansatz:

$$F_x(x,y,z) = a_1 + a_x x + a_y y + a_z z + \sum_{k=1}^{k} b_i \cdot U\left(\left|\mathbf{w}_i - (x,y,z)\right|\right)$$

$$F_x(\mathbf{w}_i) = f_{i,x}, \ \forall i \quad \text{(similar for } F_y, F_z\text{)}.$$

Solve :

$$\mathrm{L}^{-1}(f_{1,x},\cdots,f_{k,x},0,0,0,0) = (b_1,\cdots,b_k,a_1,a_x,a_y,a_z)^{\mathbf{T}},$$

$$\text{where} \quad \mathbf{L} = \left(\begin{array}{c|c} \mathbf{P} & \mathbf{Q} \\ \hline \mathbf{Q}^{\mathbf{T}} & \mathbf{0} \end{array}\right), \quad \mathbf{Q} = \begin{pmatrix} 1 & w_{1,x} & w_{1,y} & w_{1,z} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & w_{k,x} & w_{k,y} & w_{k,z} \end{pmatrix}, \ k \times 4,$$

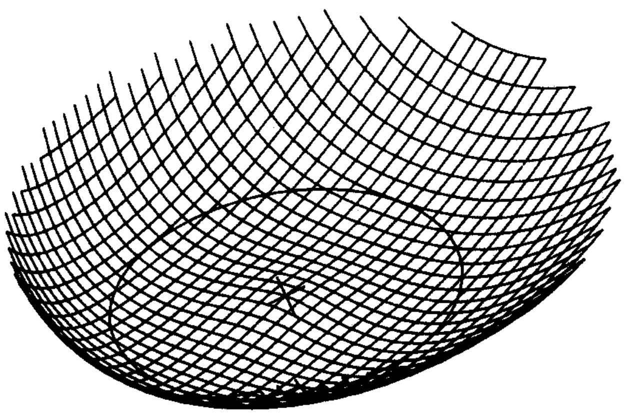
$$\mathbf{P} = \begin{pmatrix} 0 & U(w_{12}) & \cdots & U(w_{1k}) \\ U(w_{21}) & 0 & \cdots & U(w_{2k}) \\ \cdots & \cdots & \cdots & \cdots \\ U(w_{k1}) & U(w_{k2}) & \cdots & 0 \end{pmatrix}, \ k \times k.$$

Bookstein "Thin-Plate" Splines

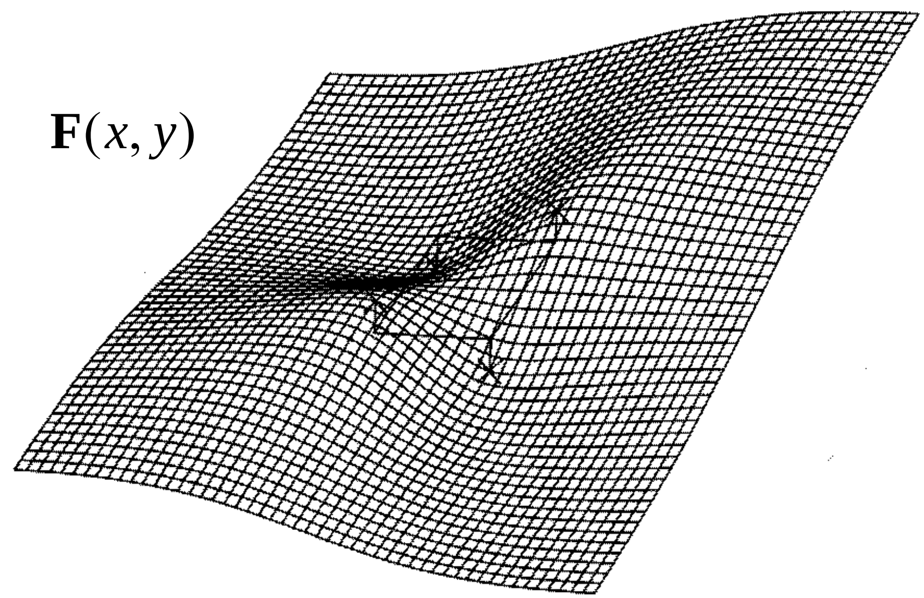
- kernel function $U(r)$ is principal solution of biharmonic equation that arises in elasticity theory of thin plates: $\Delta^2 U(r) = \nabla^4 U(r) = \delta(r).$

- variational principle: $U(r)$ minimizes the bending energy (not shown).
- 1D: $U(r) = |r^3|$ (cubic spline)
- 2D: $U(r) = r^2 \log r^2$
- 3D: $U(r) = |r|$

2D: $U(r)$

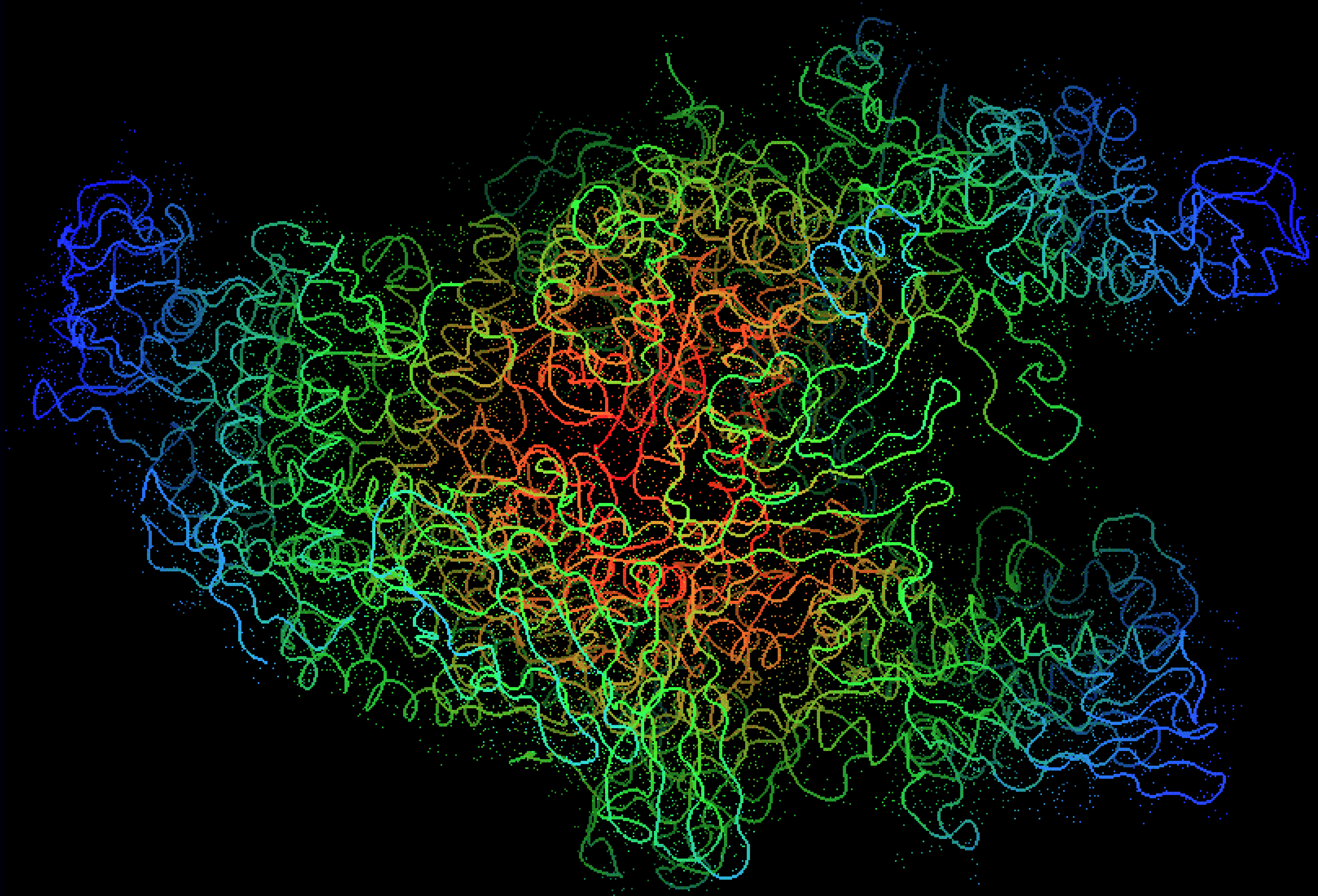


$\mathbf{F}(x, y)$

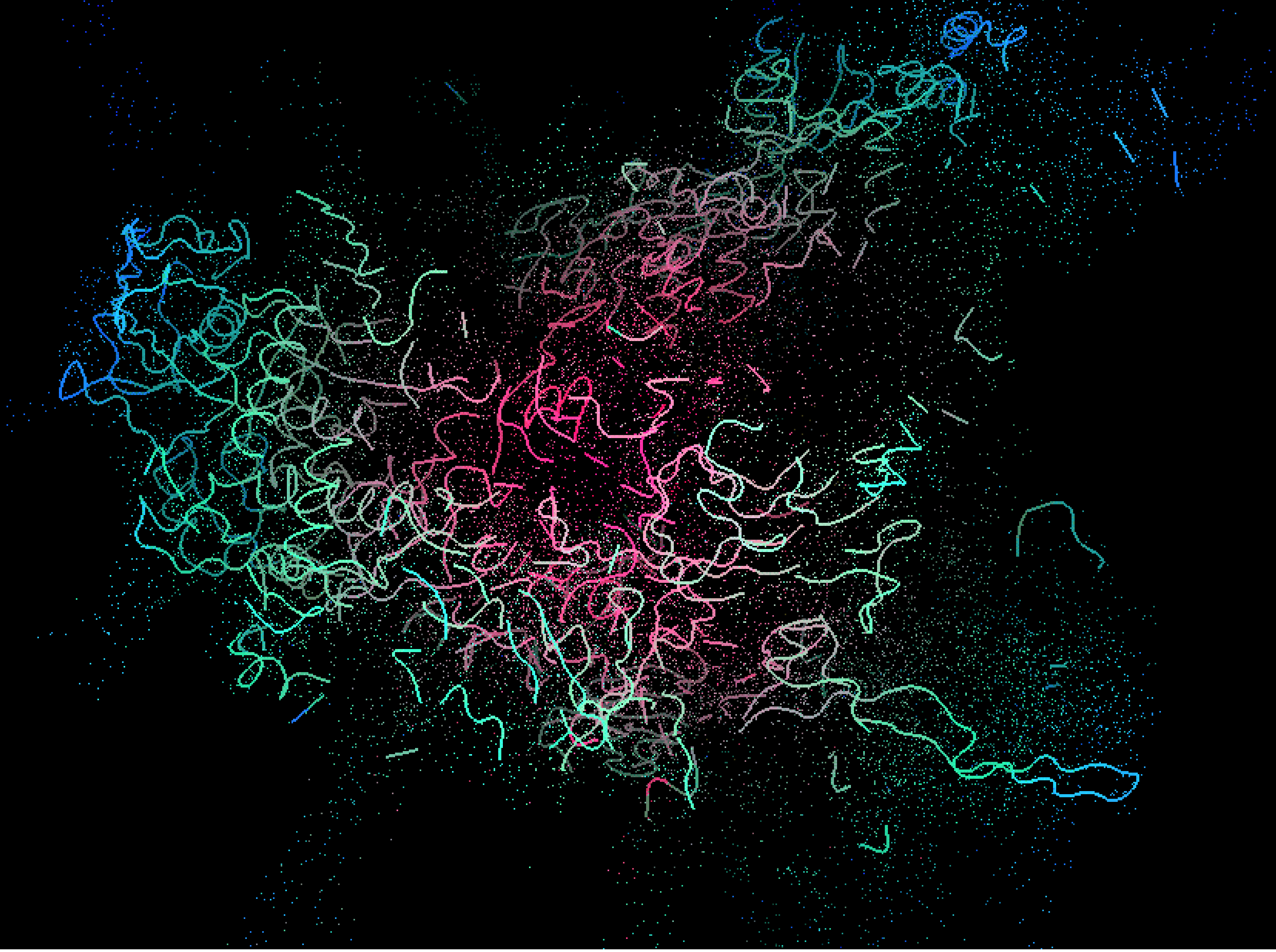


See Bookstein, Morphometric Tools for Landmark Data, Cambridge U Press, 1997

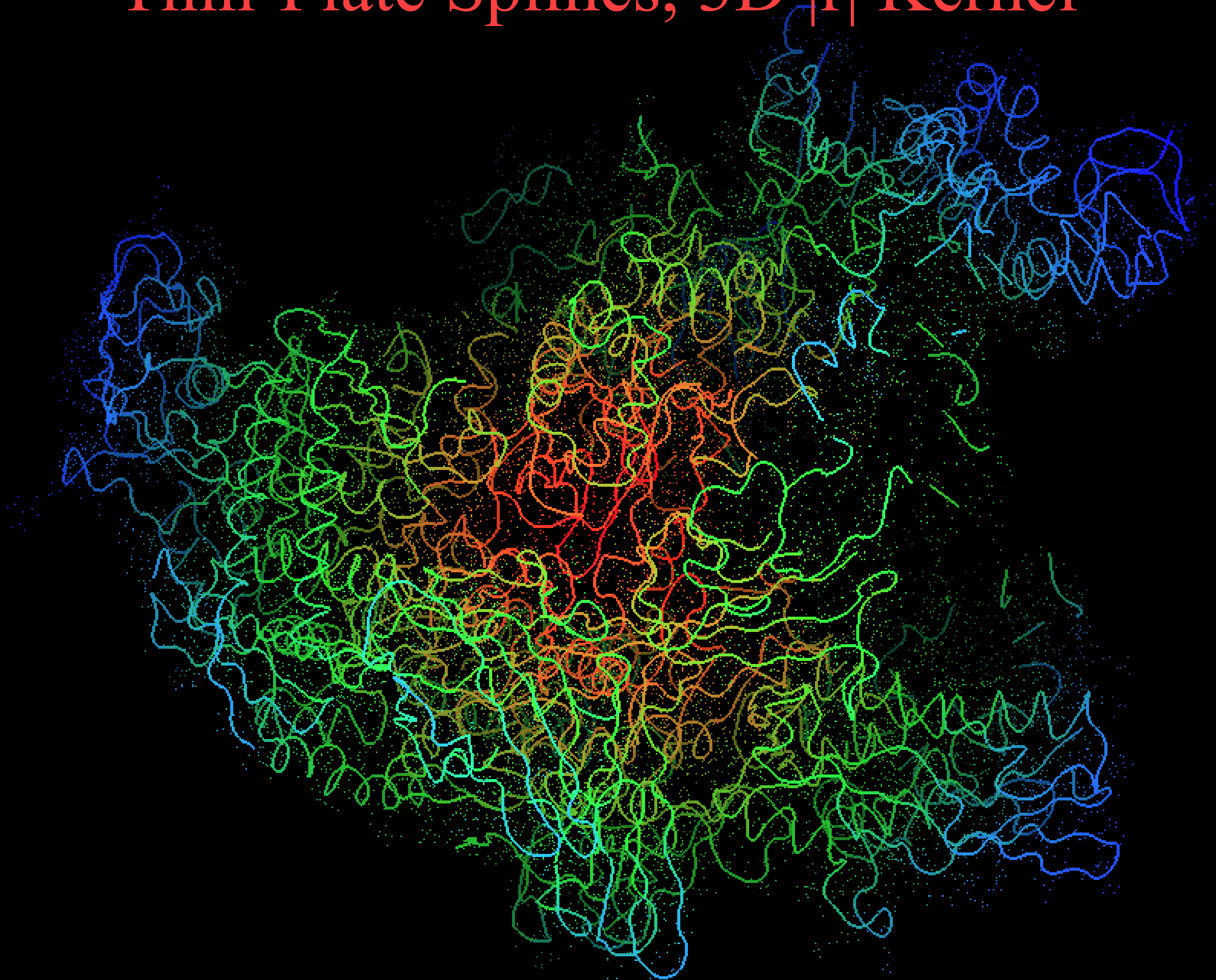
RNAP Example: Source Structure



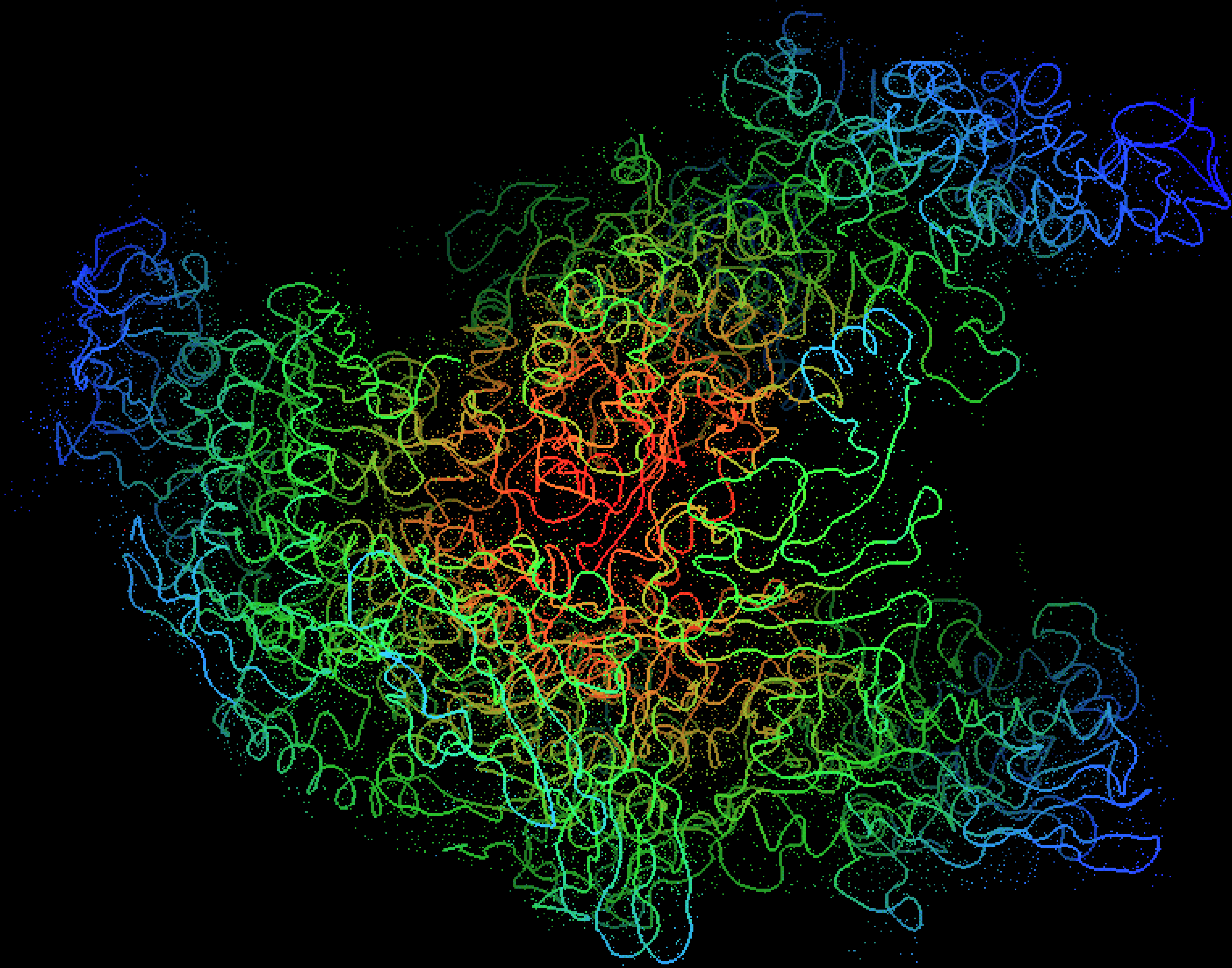
Piecewise-Linear Inter- / Extrapolation



Thin-Plate Splines, 3D |r| Kernel



Control: Molecular Dynamics



Resources

Textbooks:
Kenneth R. Castleman, Digital Image Processing, Chapter 8
John C. Russ, The Image Processing Handbook, Chapter 3

Online Graphics Animations:
http://nis-lab.is.s.u-tokyo.ac.jp/~nis/animation.html