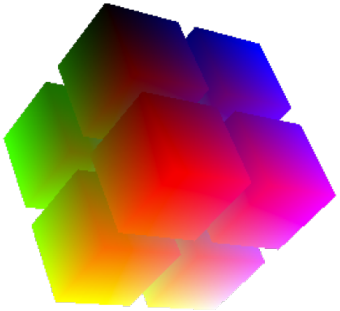


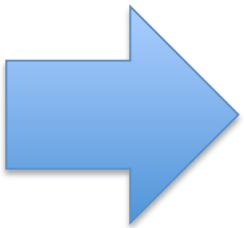
“超解像”の前に

2017年1月27日(金)

“よや” yoya@awm.jp

超解像とは？

- いわゆる解像度を上げる処理の事

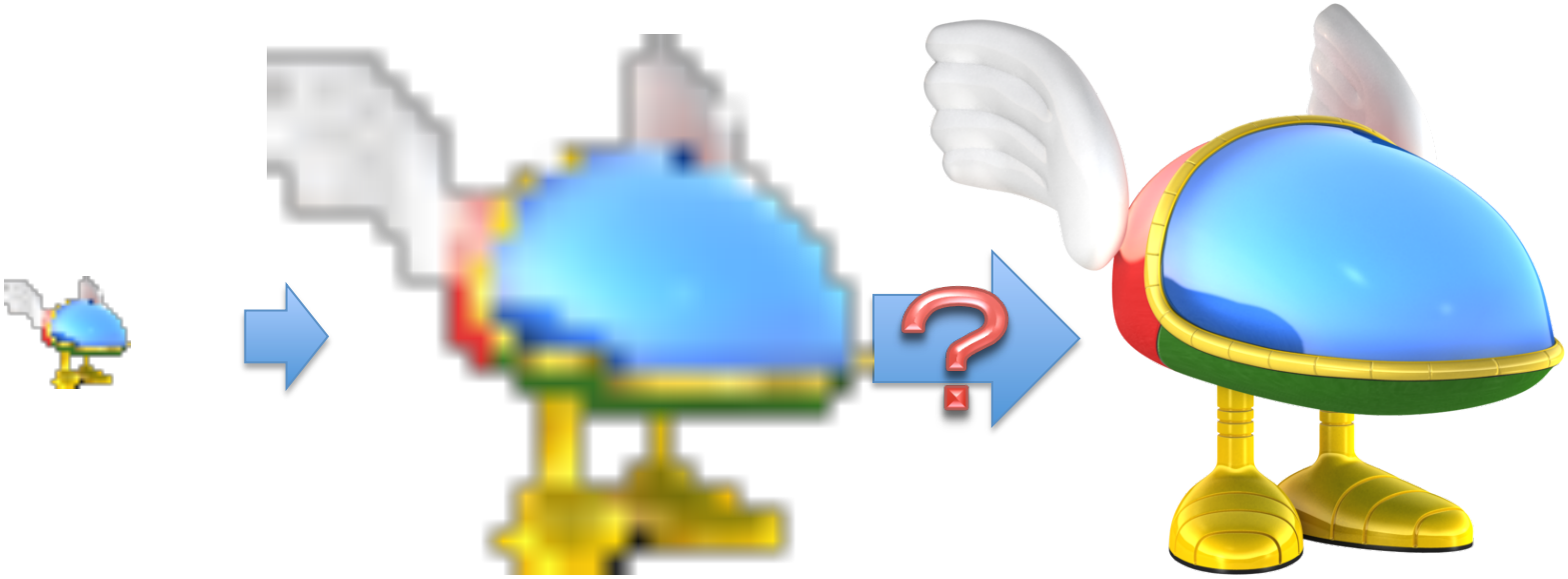


© http://dic.nicovideo.jp/a/ファンタジーゾーン

© http://sega.wikia.com/wiki/Opa-Opa

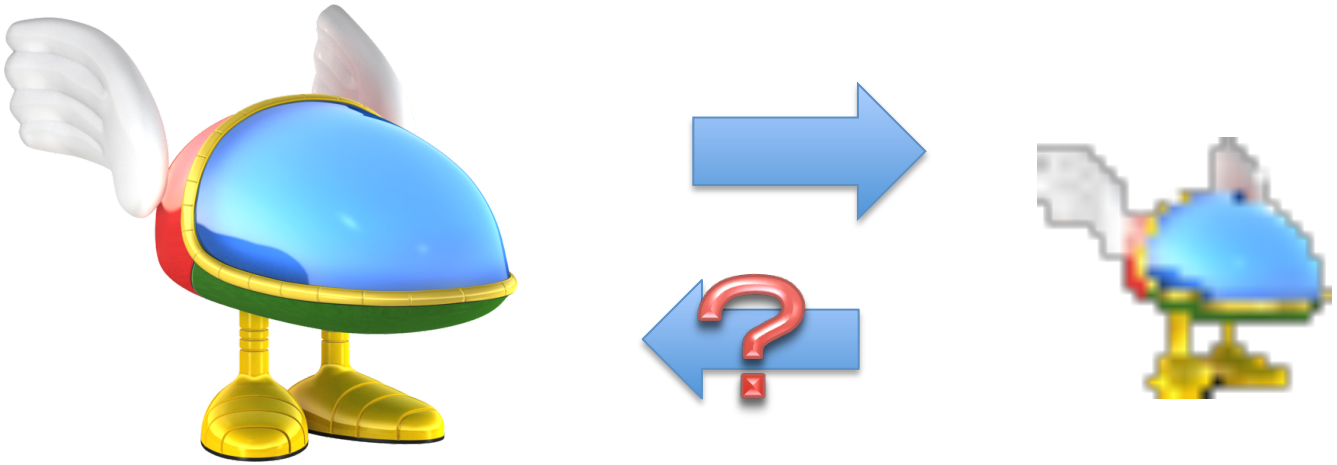
リサンプル処理

- 画像の拡大とは、ドットの細分化
  - “大きくするのは小さくする事”



暗黙の前提

- 元々、高画質な画像があって、縮小リサイズされたものを元に戻すという暗黙の前提



いわゆる逆問題 (しかも情報が減ってる)

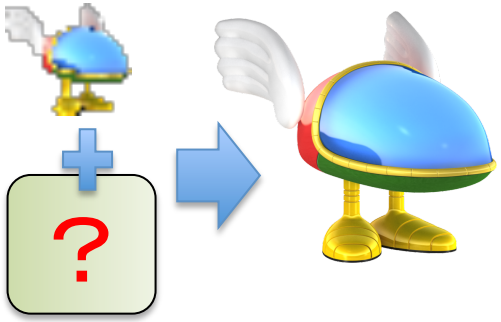
超解像は主に2つ

- 複数画像から生成
  - 情報量を数でカバー
  - サブピクセルでずらした画像を大量に用意
    → 次ページで説明

- 単体画像から生成
  - 情報不足のまま何かを仮定して補完
    - ニアレストネイバー、バイリニア等々

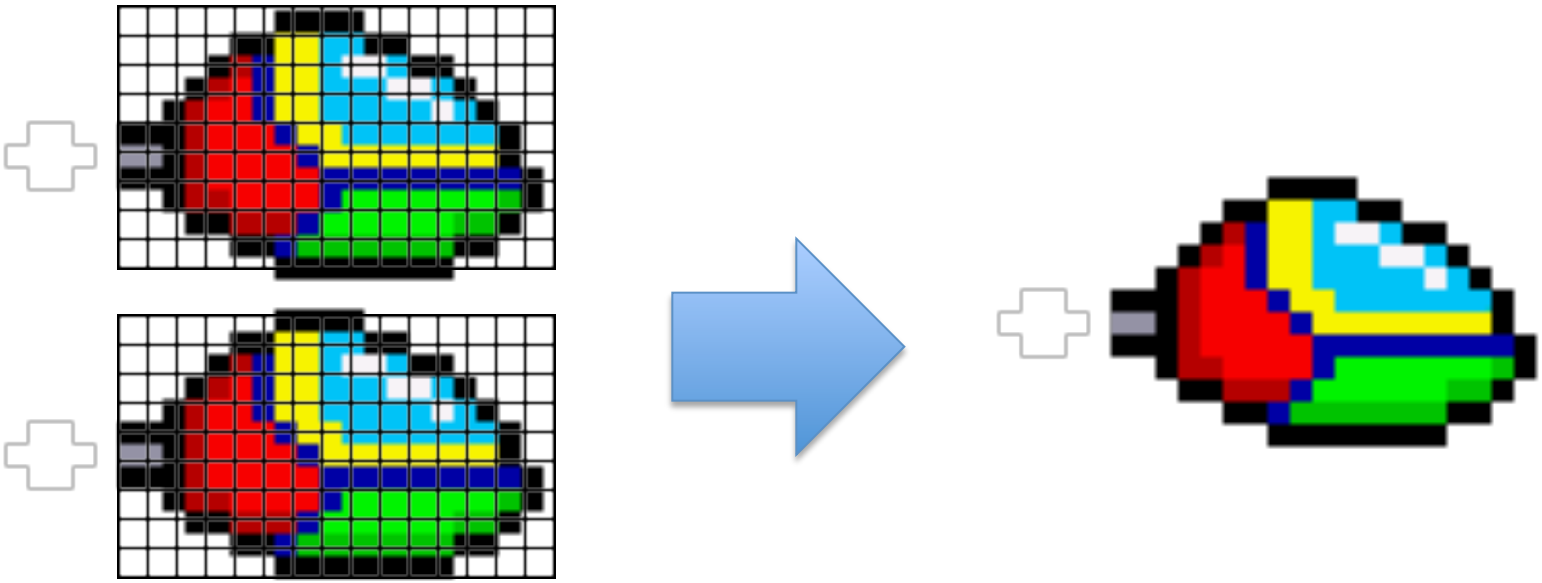
      ～ 今日はここまで ～
    - エッジ抽出
    - 機械学習系

      ～ この辺りは次回 ～



複数画像から生成

- サブピクセルでずらして沢山写真をとる
  - 連写する、複数レンズで同時に撮る等



(参考)複数レンズのカメラ

- L16

© http://gigazine.net/news/20151008-16-lens-camera/

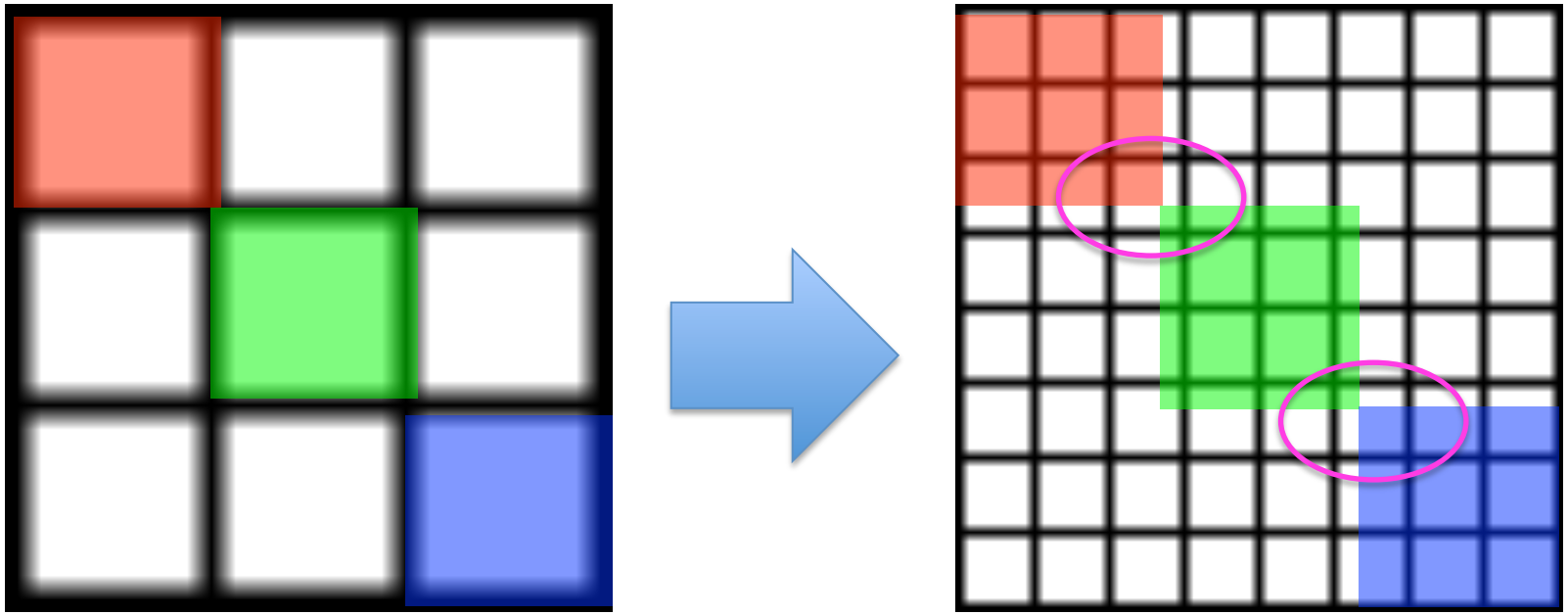
一枚の画像から高解像

- 情報量が減っているので原理的に元に戻せない

- 足りない分、何かしらの仮定を以って補う

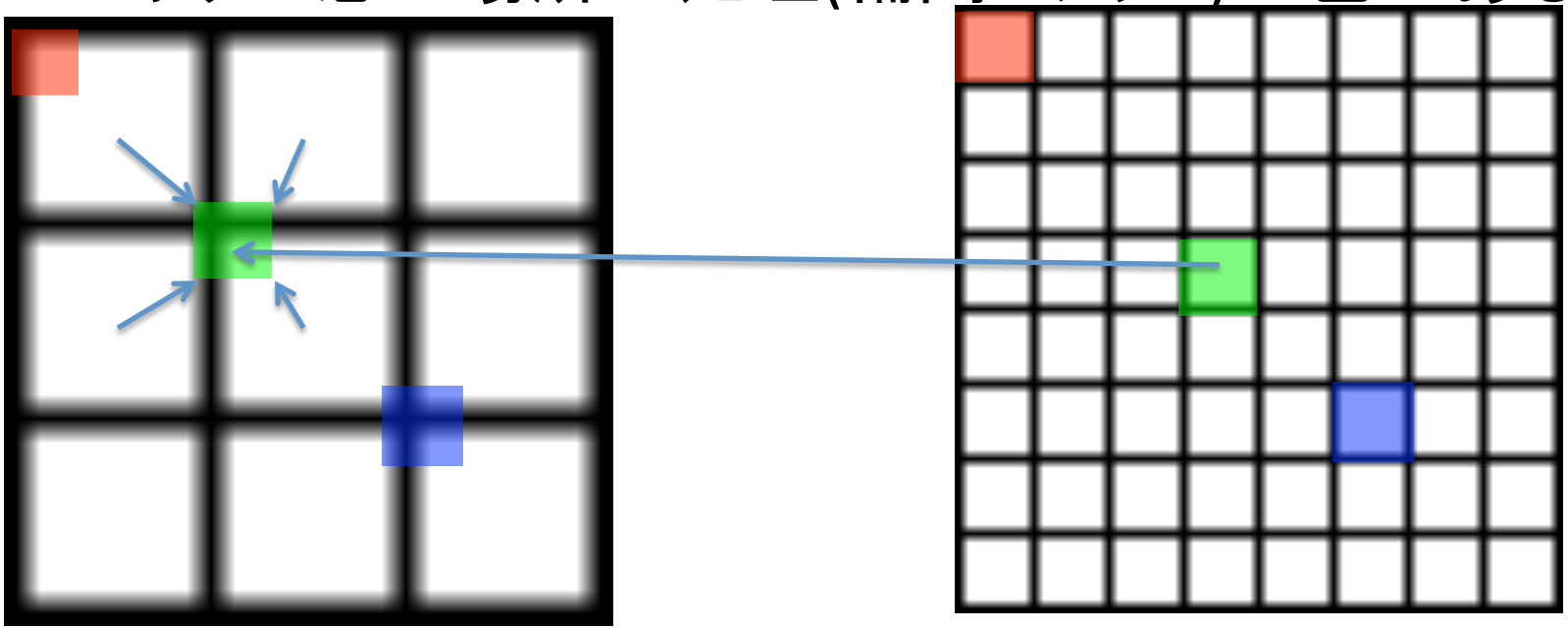
画像拡大のドット処理

- 概念的にはこれ
  - キリの悪い場所をどうするかがキモになる



リサイズのドット処理

- リサイズ後のピクセル座標で for ループ
  - 各々に対応するリサイズ前のドットを探す。
  - キリの悪い場所の処理(補間メソッド)が色々ある

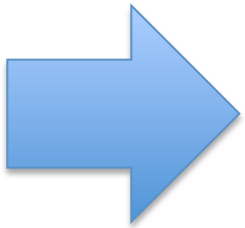


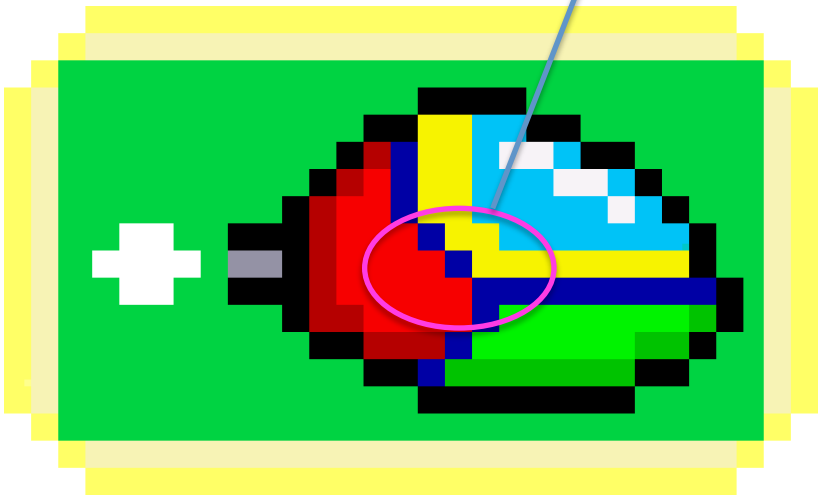
Nearest Neighbor

- 元々ジャギーな画像だと決めつける
  - 色をそのままで、ドットを広げる

```
$ convert in.png −filter point −resize 800%x800% out.png
```

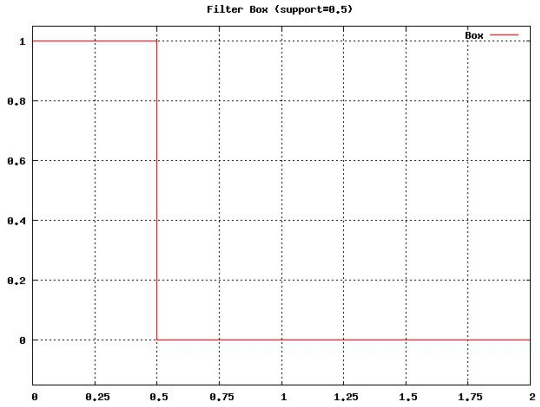
- 混ぜない！

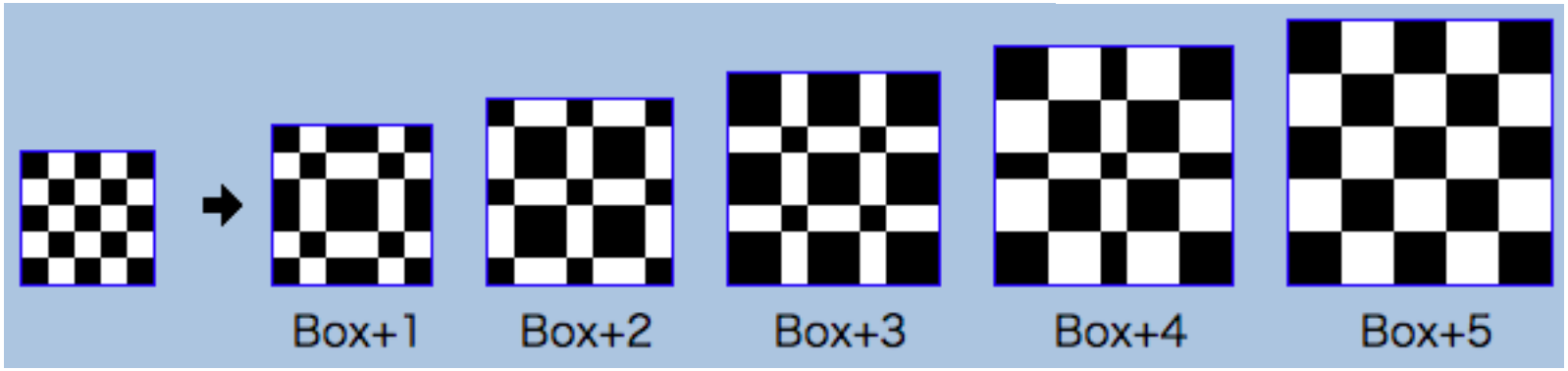




Nearest Neighbor

- 隣のピクセルをそのまま
  コピーして隙間を埋める
- とにかく処理が軽い！

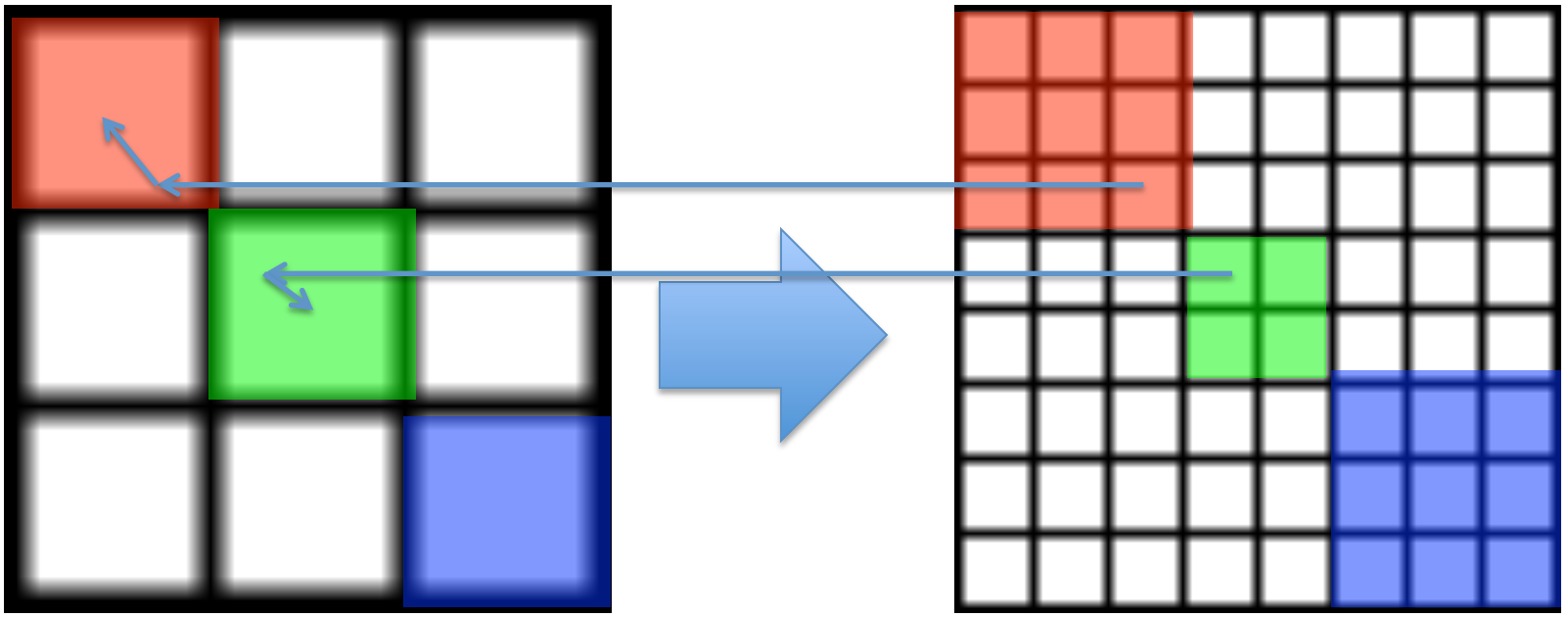




http://www.imagemagick.org/Usage/filter/#box

Nearest Neighbor のドット処理

- 混ぜない！

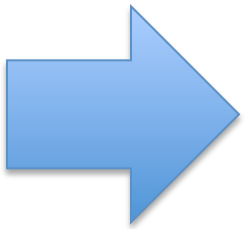


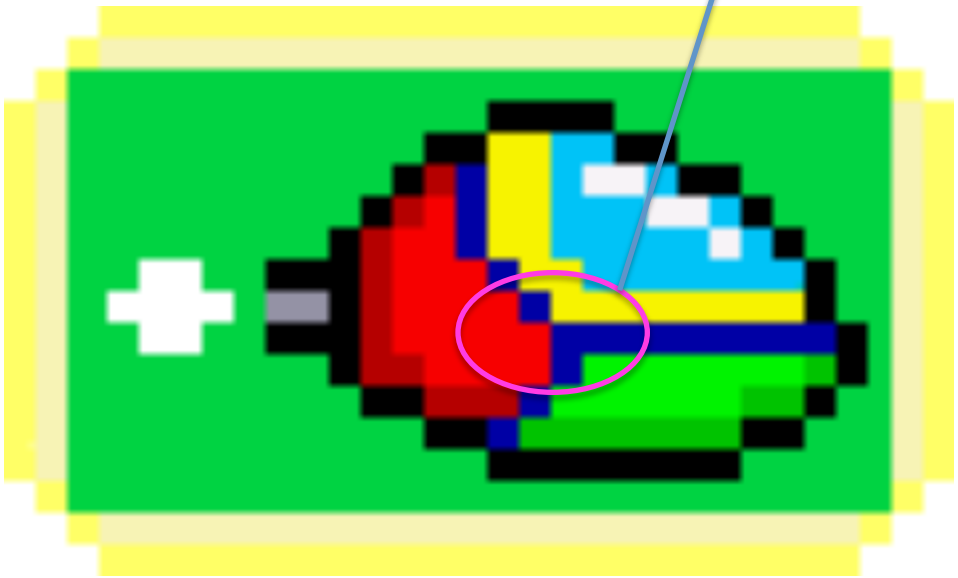
- プログラム的には座標を四捨五入するだけ

Bi-Linear

- ジャギーを増やしたくない

```
$ convert in.png −filter triangle −resize 800%x800% out.png
```



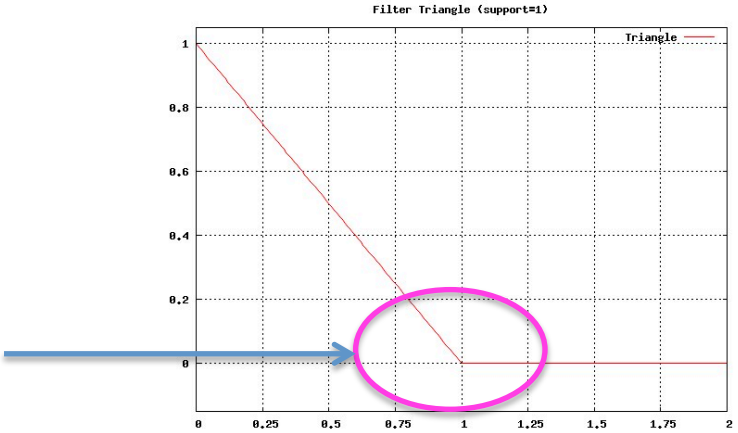


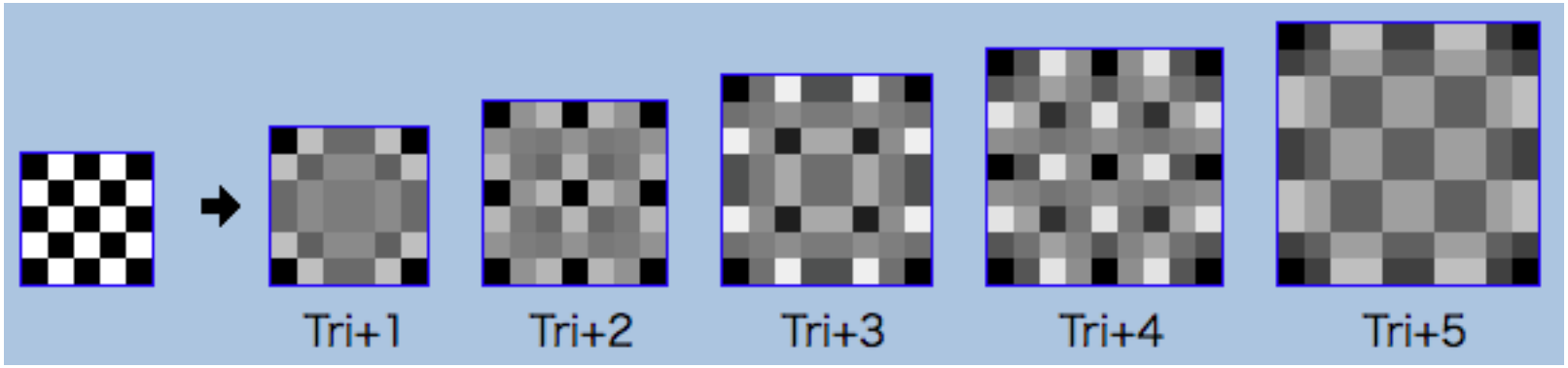
Bi-Linear

- さらに拡大。徐々に変化する様が見える

Bi-Linear

- 線形補完
  - ジャギーは増えないけど
    折り目のところが不自然

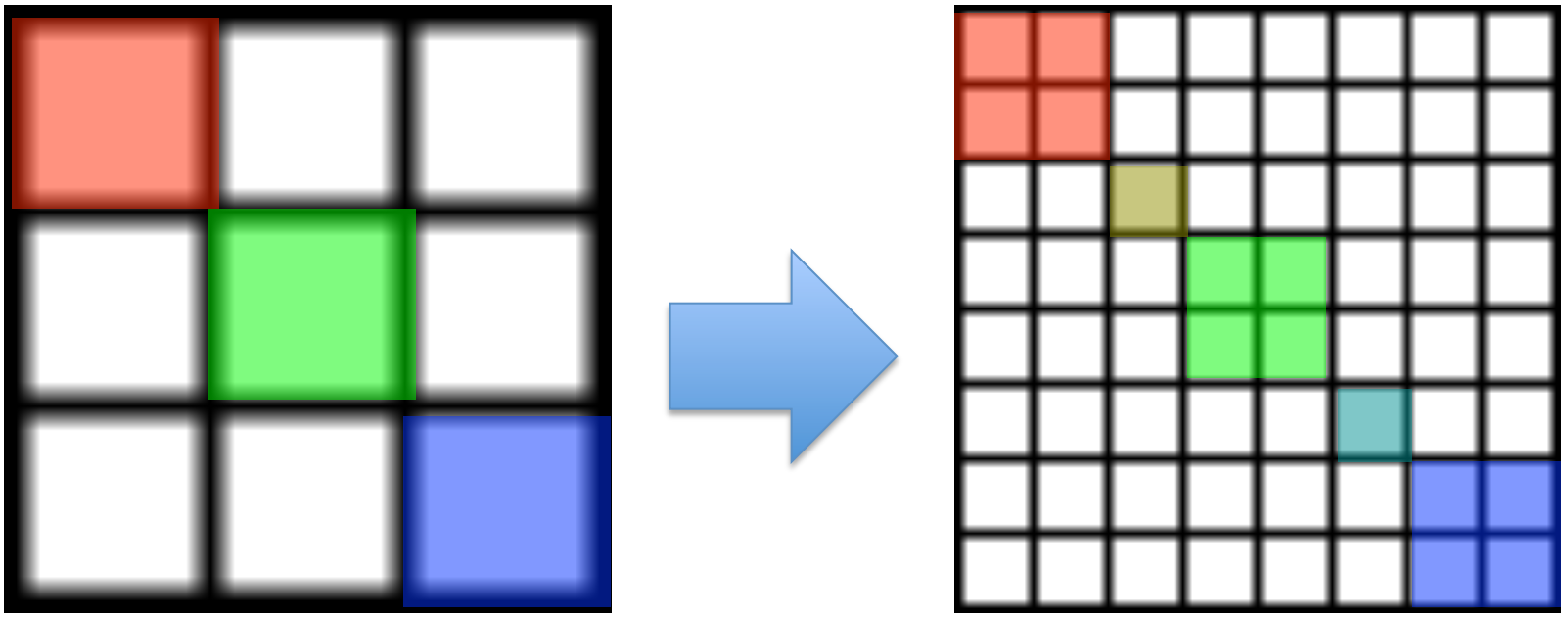




http://www.imagemagick.org/Usage/filter/#box

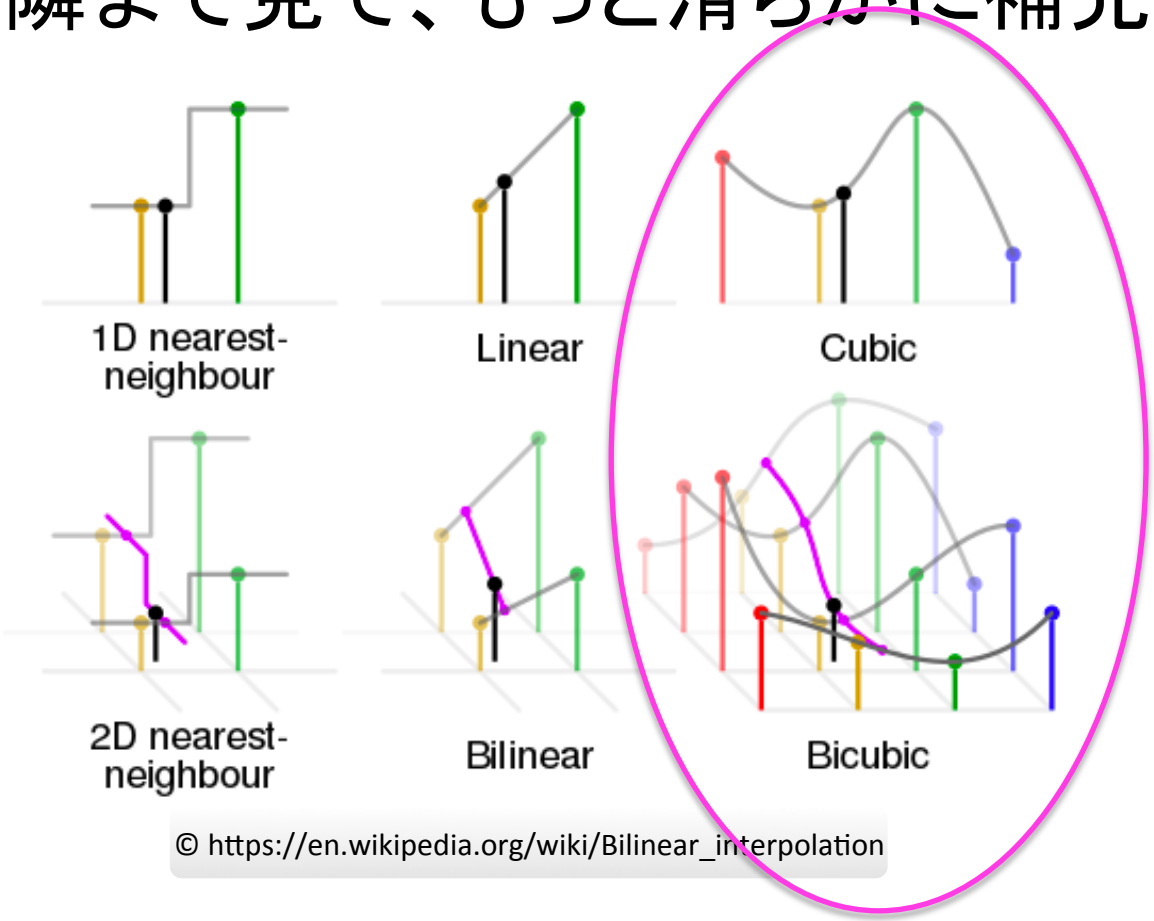
Bi-Linear のドット処理

- 単純に混ぜる
  - 距離の割合だと計算重たいので、x、y の割合



Bi-Cubic

- 隣の隣まで見て、もっと滑らかに補完する



© https://en.wikipedia.org/wiki/Bilinear_interpolation

Cubic family の計算式

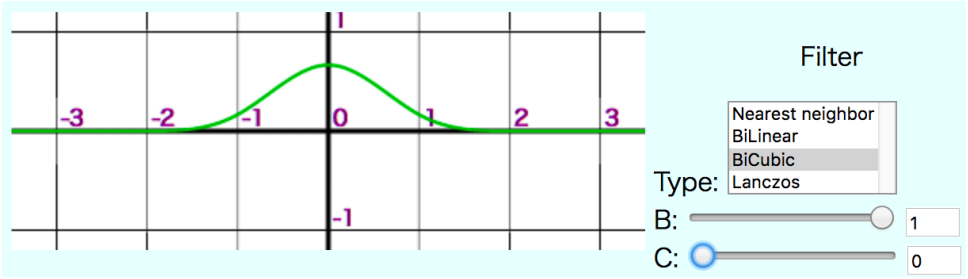
- B(blur) ,C(cardinal) で色んなフィルタを作れる

```
Coefficents are determined from B,C values:

   P0 = (  6 - 2*B       )/6 = coeff[0]
   P1 =         0
   P2 = (-18 +12*B + 6*C )/6 = coeff[1]
   P3 = ( 12 - 9*B - 6*C )/6 = coeff[2]
   Q0 = (      8*B +24*C )/6 = coeff[3]
   Q1 = (    -12*B -48*C )/6 = coeff[4]
   Q2 = (      6*B +30*C )/6 = coeff[5]
   Q3 = (    - 1*B - 6*C )/6 = coeff[6]

which are used to define the filter:

   P0 + P1*x + P2*x^2 + P3*x^3      0 <= x < 1
   Q0 + Q1*x + Q2*x^2 + Q3*x^3      1 <= x < 2
```

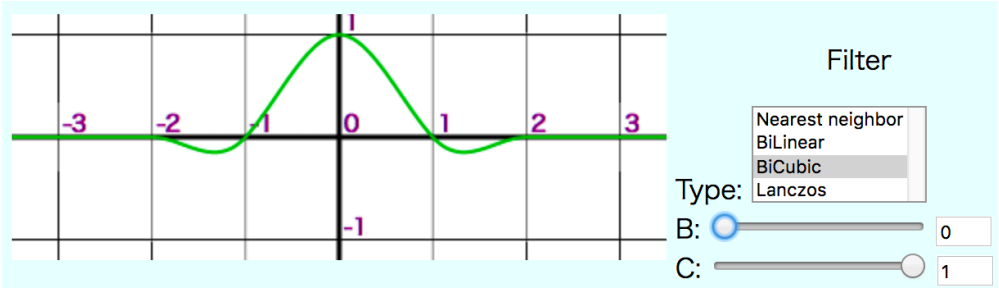
© ImageMagick-6 magick/resize.c

Cubic family の生成

- B(blur) を上げすぎるとボヤける



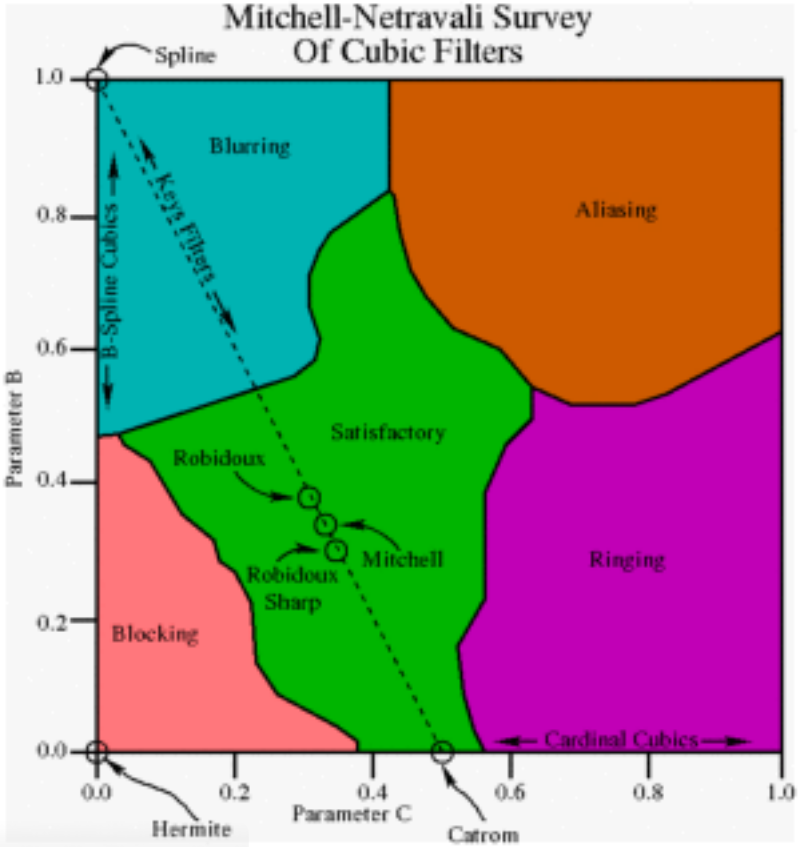
- C(cardinal) でリンギングがかかる



Cubic BC のバランスをとる

- (B,C)=(1/3,1/3) が良さそう

→ Mitchell フィルタ

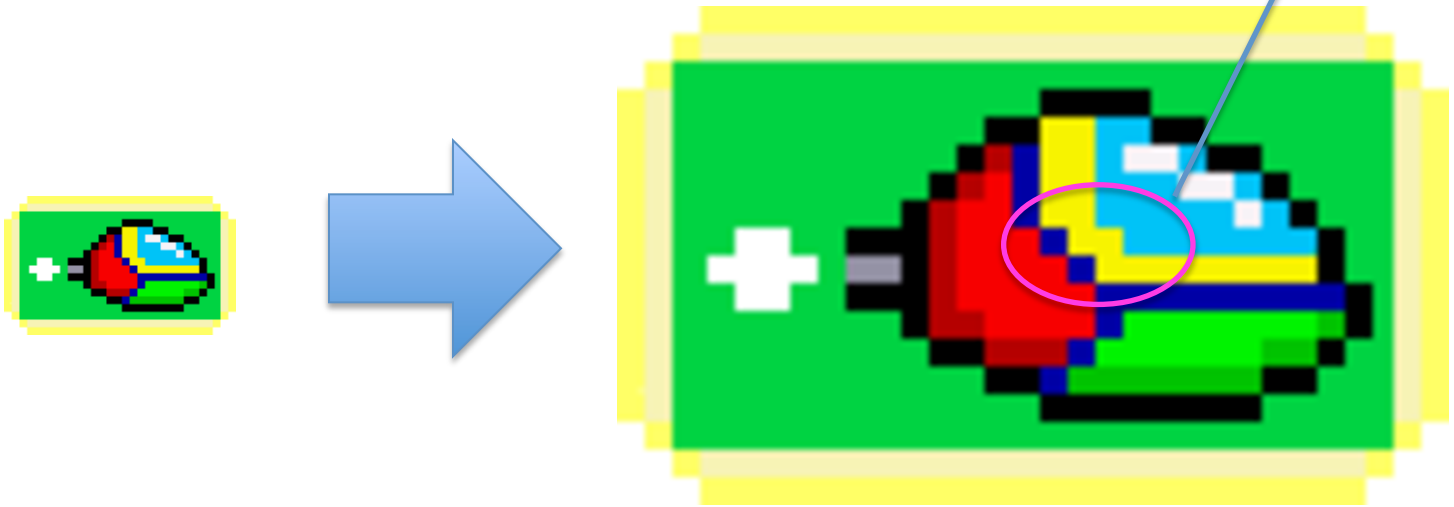


© http://www.imagemagick.org/Usage/filter/#mitchell

Mitchell-Netravali

- ImageMagick の拡大のデフォルト

```
$ convert in.png −filter mitchell −resize 800%x800% out.png
```



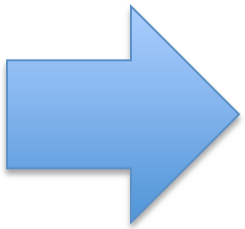
少し蛇足 (縮小について)

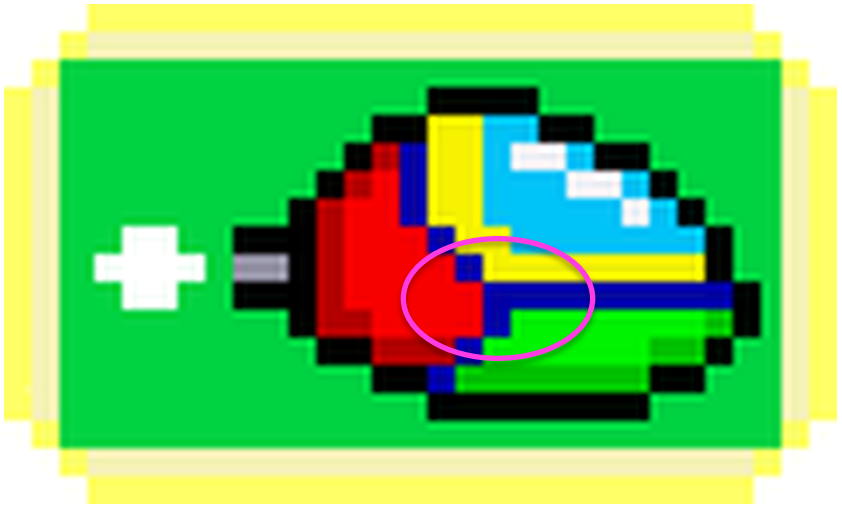
- みんな大好き Lanczos フィルタ
  - 縮小はこれを使えば大体OK

Lanczos

- ImageMagick の縮小のデフォルト
  – (見ての通り、拡大は微妙)

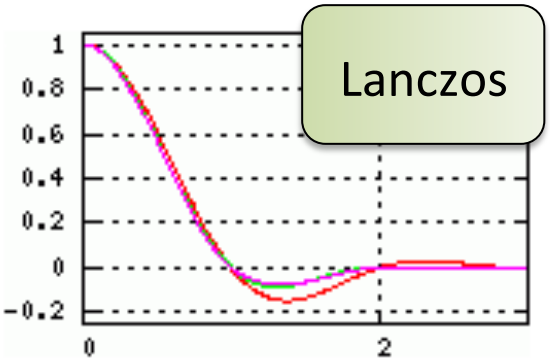
```
$ convert in.png −filter lanczos −resize 800%x800% out.png
```



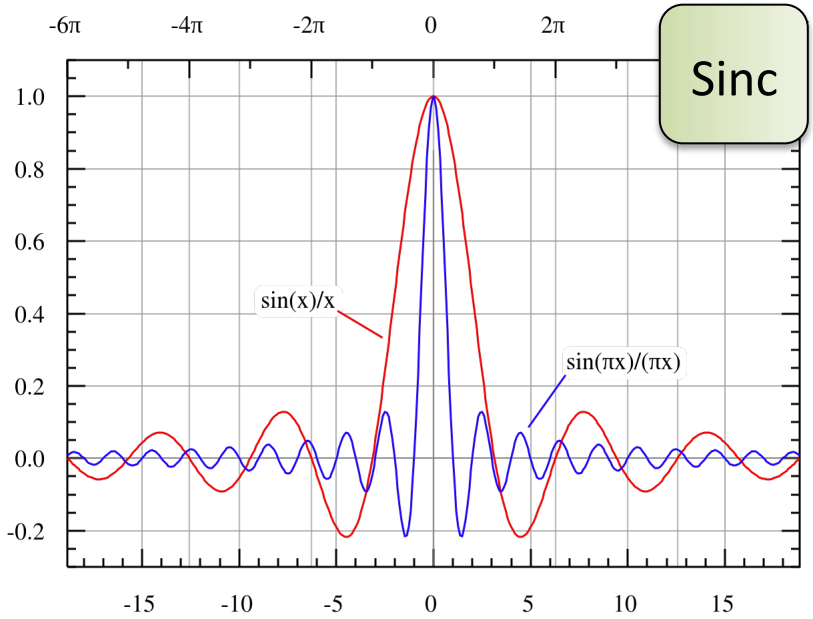


Lanczos の縮小がなぜ良いか？

- Lanczos



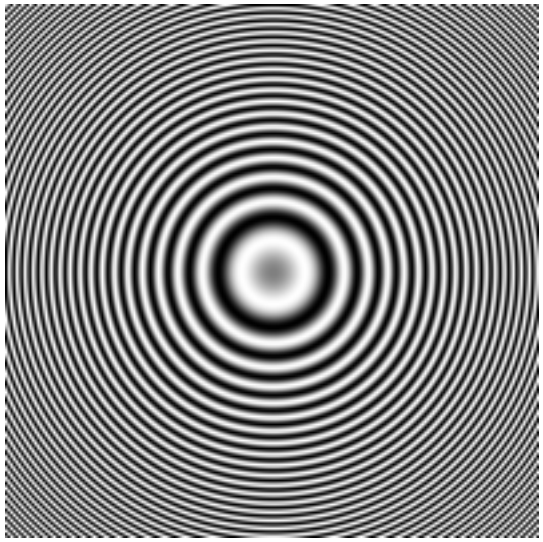
- Sinc 関数を元にしてる
- それの
  コンパクトサポートが
  Lanczos

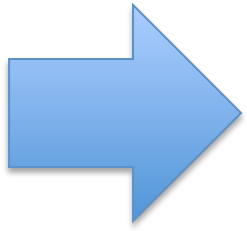


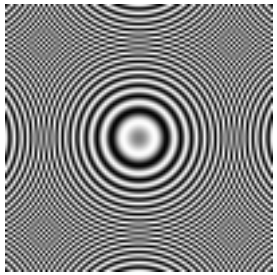
© https://en.wikipedia.org/wiki/Sinc_function

Sinc 関数って？

- 詳しくは Wikipedia で
- フーリエ変換すると矩形波になる
  - つまり、周波数フィルタになる > LPF
- エイリアシングとおさらば

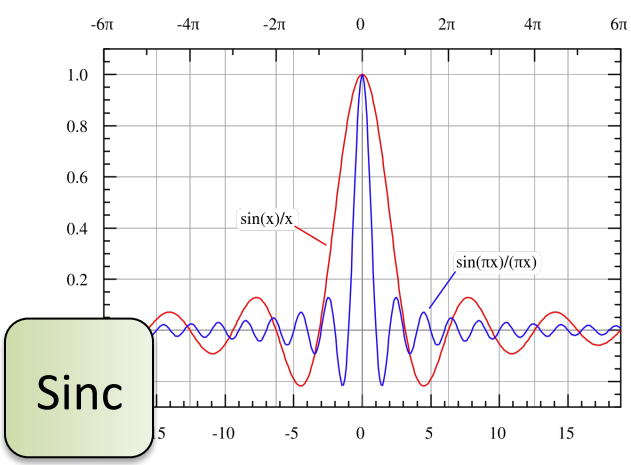






コンパクトサポートって？

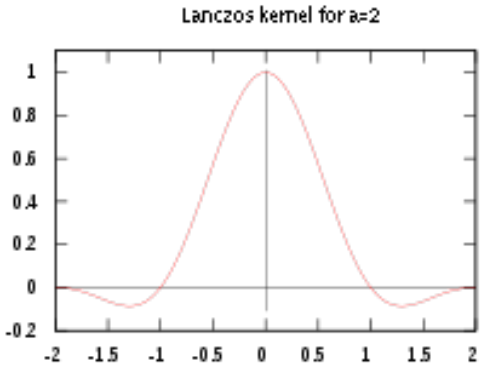
- Sinc 関数は無限の範囲で波打つ
  - 計算量が辛い。理屈では無限
- 一定範囲の外を0に抑える
  ↑これがコンパクトサポート

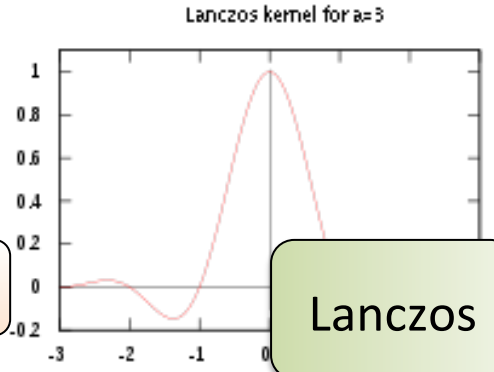


Sinc

sin(x)/x

sinc(x)/sinc(x/a)

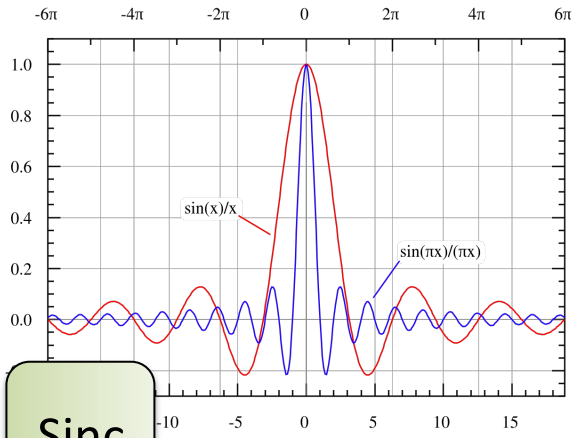




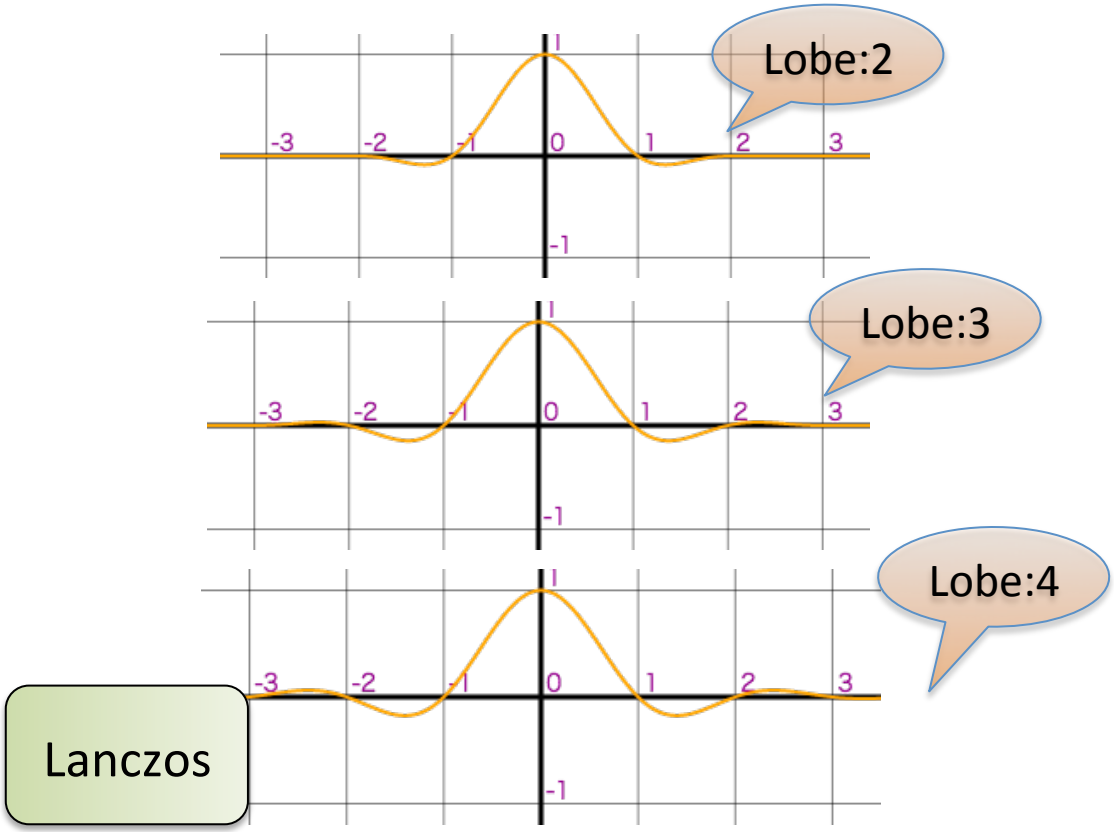
Lanczos

Lanczos2,3,4

- 2,3,4 に区間に抑えたのが Lanczos



Sinc



Lanczos

まとめ

- これらのフィルタを使い分けると良いかも？
  - リアルタイム表示 (もしくはドット絵の場合も)
    - Nearest Neighbor ← 処理が超軽い
  - 拡大は
    - Mitchell フィルタ
  - 縮小は
    - Lanczos フィルタ

次回予告

- 超解像
  - ~~単純な補間フィルタ (今回説明したこと)~~
  - 勾配から輪郭抽出 (いまどきの超解像)
  - 機械学習系 (先端的な？)