CS4365 Applied Image Processing - Final Project
Pixel-art Upscaling Algorithms

William Narchi (5046122)

Delft University of Technology, The Netherlands

1. Introduction

Pixel-art denotes a form of digital art in which pixels are the core building block and aesthetic focal point of the art piece. This style is especially pervasive in videogames developed before the mid 1990s, primarily due to the limitations of consumer-grade graphics processing hardware at the time. Many of the most immediately recognizable cultural icons in videogame communities originate from the work of artists who - due to these technical constraints - meticulously drew out artwork pixel by pixel.

Hardware capable of displaying higher resolution imagery has proliferated since then, though appreciation for the medium has persisted, birthing a need for a form of upscaling to present pixel-art at a higher fidelity to accommodate modern standards. As such, a number of upscaling algorithms have been developed which specifically target pixel-art as their problem space. Most of the work done in this field is conducted by hobbyists in their spare time for enhancing retro home console emulators and as such most of the algorithms presented in this report originate from personal websites, blogs and forum posts, though two approaches with roots in formal academia are also presented.

This project submission is specifically concerned with upscaling by a factor of two, i.e. producing an image of dimensions $2X \cdot 2Y$ given an image of dimensions $X \cdot Y$. Additional upscaling can be performed by repeated application of the upscaling algorithm to produce images of scales that are powers of two, though in cases where there are specific variants of a particular algorithm targeted towards >2x scaling, they have not been implemented as this project submission aims to only present the core idea inherent in each algorithm.

2. Literature Survey

As there is very little formal academic research into the field of pixel-art upscaling algorithms, the literature survey table presented consists of a brief collection of pros and cons of the implemented algorithms.

Algorithm	Pros	Cons
Rule-based Interpolation		
EPX / AdvMAME2x	- Simple implementation - Low performance cost	- Overtly sharp output
Eagle	- Simple implementation - Low performance cost	- Loss of small, isolated details
2xSaI	- Low performance cost	- Blurry blocky artifacts on repeated application
hqx	- Specialized variants for 2x, 3x, and 4x scaling - Balanced smoothing and blocky style preservation	- Scaling criteria unexplained - Clunky implementation
xBR	- Specialized variants for 2x, 3x, and 4x scaling - Smoother 'anti-aliased' output	- Strong softening when low scaling factors used
Edge-directed Interpolation		
NEDI	- Preservation of edge detail - Strong theoretical basis	- Potential artifacting - No publicly available implementation by author(s)
ANEDI	- Less artifacting than NEDI - Potentially lower computation costs than NEDI	- Potentially higher computation costs than NEDI if matrices highly ill-fitted - No publicly available implementation by author(s)
Vectorization		
Kopf-Lischinski	- Arbitrary scaling factors - Relatively computationally expensive - Complex implementation	- 'Toonification' and loss of intended blocky aesthetic - Ineffective with realistic sprites - No publicly available implementation by author(s)

3. Algorithms

This section explains the main workings of the implemented algorithms. Section 3.1 discusses techniques which rely on a fixed set of rules to compute the values of the interpolated pixels. Section 3.2 discusses a variant of the **New Edge-Directed Interpolation (NEDI)** algorithm, which relies on computing pixel covariances in order to augment interpolation with information about perceived edges [LO01a]. Finally, section 3.3 discusses an approach which relies on computing a resolution-independent vector representation of the given image which can then be used to scale the image to an arbitrary degree.

3.1. Rule-based Interpolation

3.1.1. EPX & AdvMAME2x

Eric's Pixel Expansion (EPX) was developed by Eric Johnston in 1992 while porting SCUMM engine games from the IBM PC to early Macintosh computers while working at LucasArts [Tho99]. It is a simple algorithm that expands a pixel P with axial neighbours A, B, C, and D (see fig. 1) according to the following rules:

```
1 = P; 2 = P; 3 = P; 4 = P;
if (C == A) { 1 = A; }  if (A == B) { 2 = B; }
if (D == C) { 3 = C; }  if (B == D) { 4 = D; }
if (threeOrMoreIdentical(A, B, C, D)) { 1 = 2 = 3 = 4 = original_pixel; }
```

A variant of this algorithm known as AdvMAME2x also exists and produces identical results though using the following - more efficient - rule set [Wik22]:

```
1 = P; 2 = P; 3 = P; 4 = P;
if (C == A && C != D && A != B) { 1 = A; }
if (A == B && A != C && B != D) { 2 = B; }
if (D == C && D != B && C != A) { 3 = C; }
if (B == D && B != A && D != C) { 4 = D; }
```

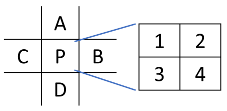


Figure 1: The set of pixels considered by the EPX and AdvMAME2x algorithms [Wik22]

3.1.2. Eagle

Developed around 1999, Eagle is similar to EPX and AdvMAME2x in that it applies a simple set of interpolation rules based on pixel equalities [zin07]. The rule set is as follows, where pixel *C* is to be expanded to the four pixels 1, 2, 3, and 4:

```
// S T U
// V C W
// X Y Z
if (V == S == T) { 1 = S; } if (T == U == W) { 2 = U; }
if (V == X == Y) { 3 = X; } if (W == Z == Y) { 4 = Z; }
```

3.1.3. 2x Scale and Interpolation

2x Scale and Interpolation (2xSaI) is a slightly more complex approach, first developed by Derek Liauw Kie Fa in 1999. It operates on two levels: first, it checks for the existence of diagonal edges (i.e. diagonal pixels with the same values) in a 2x2 pixel matrix in the original image, where the pixel to be scaled is in the top-left corner. Second, based on which edges were detected, the resulting pixel values are computed as interpolations of the pixels in the 2x2 matrix depending on the similarity of pixels in a 4x4 matrix in the original image, where the pixel to be scaled is in the second row, second column. [Wik22, Fa02]

3.1.4. hq2x

hqx is a family of pixel-art upscaling algorithms developed by Maxim Stepin and constitutes one of the most prolific approaches in the retro console emulation community, in addition to having inspired numerous more complex approaches [Wik22]. There are three variants which allow for upscaling by factors of 2x, 3x, and 4x, though the 2x variant is the one being analysed in this report.

The approach works by considering the 3x3 pixel matrix centered around the pixel currently being expanded. Each of the expansion pixel's eight neighbours are marked as 'similar' or 'different' to the expansion pixel depending on the difference in YUV component values. Based on the 256 possible combinations of 'similar' and 'different' neighbours the resulting expanded pixels are computed as interpolations of the pixels in the 3x3 matrix. The original algorithm does not specify a particular line of logic behind the chosen interpolation combinations, only citing that "for each combination the most probable vector representation of the area has to be determined, with the idea of edges between the different colored areas of the image to be preserved, with the edge direction to be as close to a correct one as possible."

A streamlined version of the algorithm was implemented using the findings of [ins14], which allow for the compression of the multi thousand line switch statement of the original implementation to only a few hundred lines by exploiting redundancies and making using of preprocessor macros.

3.1.5. xBR

Scale by Rules (xBR) is a family of upscaling algorithms that relies on a multi-stage process that varies depending on the exact member of the family [Wik22]. The prototypical filter - and the particular variant implemented for this project - relies on an edge detection stage followed by an interpolation stage and was developed by LibRetro forum poster Hylian in 2012 [Hyl12].

The first stage relies on detecting diagonal edges by utilising a distance function that relies on YUV components in a manner very similar to hqx. If an edge has been found in either of the four diagonal directions, a test is conducted to attempt to estimate the exact orientation of the found edge by checking neighbouring pixels for equality. Once an exact orientation has been computed, the value of each resultant pixel is computed as an interpolation between the pixel being expanded and the axial neighbours relevant with respect to the orientation of the edge that was detected. By varying the slope of the exact orientations being tested for, the algorithm can be extended to 2x, 3x, and 4x scale factors, though the provided implementation only performs 2x scaling.

3.2. Edge-directed Interpolation (NEDI)

Edge-directed interpolation is a class of statistically-informed image interpolation techniques that attempts to ensure the smoothness of colour gradients along edges and the lack of smoothness of colour gradients across edges. The specific approach implemented for this project submission is a slightly modified version of **Adaptive New Edge-Directed Interpolation (ANEDI)** [Tze04], which is based on **New Edge-Directed Interpolation (NEDI)** [LO01b] and relies on modelling edges as resolution-invariant curves.

The NEDI approach relies on computing covariances between pixels in order to model edges. This is done by sampling a local window of pixels, once for diagonal neighbours and once for axial neighbours, in order to compute the weights used to interpolate the missing pixels. ANEDI further introduces variable sizes for the window in order to reduce processing time and generate more visually consistent results by attempting to reduce the occurrence of ill-conditioned covariance matrices.

3.3. Vectorization (Kopf-Lischinski)

Vector graphics are a form of graphics representation that relies on encoding images as a collection of geometric shapes, as opposed to encoding values of singular pixels as is commonly done with most images. The key advantage of this approach is that images can be scaled to any arbitrary resolution without any loss in quality. The technique presented in [KL11] attempts to 'vectorize' pixel-art in order to leverage this benefit and achieve arbitrary scaling.

First, a square lattice graph representing the original bitmap image is constructed and then reshaped so that neighbouring pixel cells that make up an edge or that are part of the same visual feature share an edge. This is done by constructing a similarity graph where each node represents a pixel and is initially connected to all 8 of its neighbours, axial and diagonal. A number of heuristics are used to eliminate diagonal crossing edges and produce a planar graph whose dual has the desired property of visually connected nodes sharing an edge and which is then used to reshape the square lattice graph.

Following this, B-spline curves are then fitted onto the edges of the square lattice graph; special attention is given to the case where three of such curves meet at a common point, in which case two of the curves are collapsed into one, creating a T-junction. Lastly, the curves are optimized to better enhance their smoothness while attempting to preserve intentionally placed sharp edges around relevant features. This is done by varying the location of each curve's control points in order to minimize the sum of a per-node energy function; sharp edges are preserved by excluding nodes following certain patterns from the summation. The resultant optimized B-spline curves then encode a vector representation of the original image.

4. Results and Analysis

This section begins with an evaluation of the visual fidelity of the implemented algorithms' results in section 4.1. This evaluation is based on and followed by a collection of images upsampled using the implemented algorithms located in section 4.2.

4.1. Evaluation

EPX and AdvMAME2x produce extremely sharp results which preserve a significant amount of detail, though their output is occasionally jarringly sharp and is generally quite blocky. Eagle also produces quite sharp output, but introduces blocky artifacting and jagged 'spikes' of colour around edges. Further, sufficiently small details disappear entirely, such as the eyes of the dolphin in the 'Dolphin' sprite. 2xSaI remedies the perceived blockiness of the prior filters, providing a smoother output, though this often results in a seemingly more blurry final result. Edges in particular are characterised by a more gradual transition between colours which lends additional coherence at the cost of sharpness and the intended blocky aesthetic of the original image. hq2x offers a balance between the extremes of 2xSaI, Eagle, and EPX/AdvMAME2x. Images produced by it are smoother than those produced by the latter two, though do not come off as blurry as those produced by 2xSaI. xBR provides a more homogeneous aesthetic compared to hq2x; colours are grouped into regions of smoother bands which have a toon-like aesthetic. The final output comes off as smooth, though still retaining a significant degree of sharpness.

The images produced by the modified version of NEDI suffer from significant artifacts as a result of the occurrence of ill-fitted matrices. Additionally, an effect similar to the sense of motion imparted by a radial blur filter is also evident.

Lastly, the Kopf-Lischinski approach takes the visual style demonstrated by xBR to an extreme, producing a very toon-like aesthetic with smooth regions of relatively homogeneous colours. However, it retains a significantly higher degree of sharpness at the cost of smaller details, such as the curvature of certain shapes, being altered. Further, some of the examples produce garbled results using this method, though a significant amount of this garbling is likely due to faults in the provided implementation, as the original paper boasts much more satisfactory results when using the same inputs as those provided in this report.

4.2. Upscaled Images

This section contains a collection of low resolution sprites that have been upscaled by the implemented algorithms. Each set contains the original image and versions that have been scaled by a factor of 2 using each technique.

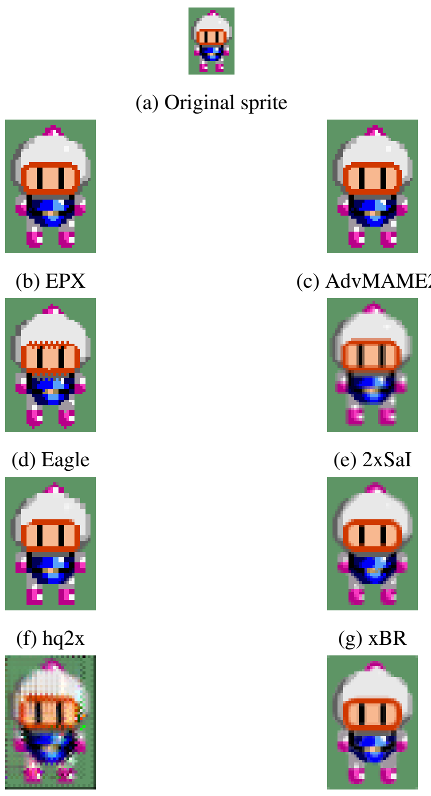
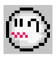
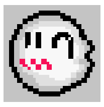
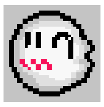
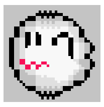

(a) Original sprite
(b) EPX
(c) AdvMAME2x
(d) Eagle
(e) 2xSaI
(f) hq2x
(g) xBR
(h) NEDI
(i) Kopf-Lischinski

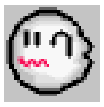
Figure 2: 2x upscalings of the Bomber sprite compared to the original (Super Bomberman, © Hudson Soft)

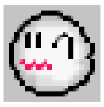

(a) Original sprite

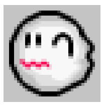

(b) EPX

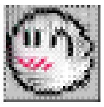

(c) AdvMAME2x


(d) Eagle


(e) 2xSaI


(f) hq2x


(g) xBR


(h) NEDI

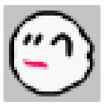
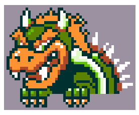
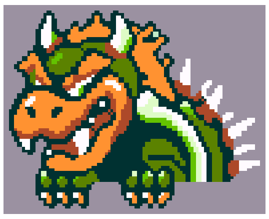
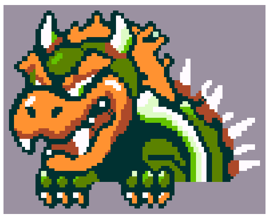
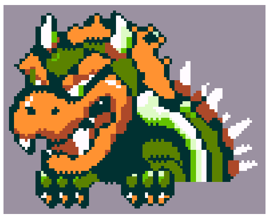

(i) Kopf-Lischinski

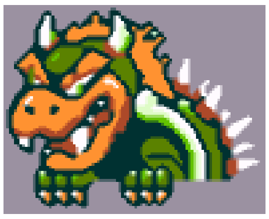
Figure 3: 2x upscaling of the Boo sprite compared to the original (Super Mario World, © Nintendo)

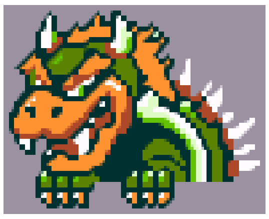

(a) Original sprite

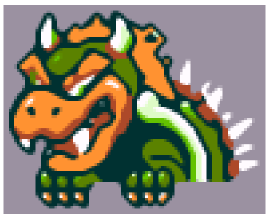

(b) EPX

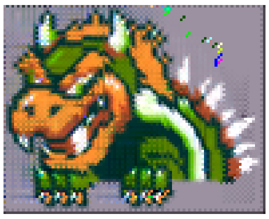

(c) AdvMAME2x


(d) Eagle


(e) 2xSaI


(f) hq2x


(g) xBR


(h) NEDI

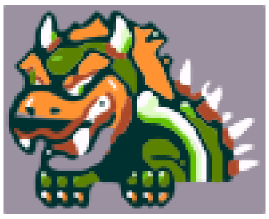

(i) Kopf-Lischinski

Figure 4: 2x upscaling of the Bowser sprite compared to the original (Super Mario World, © Nintendo)

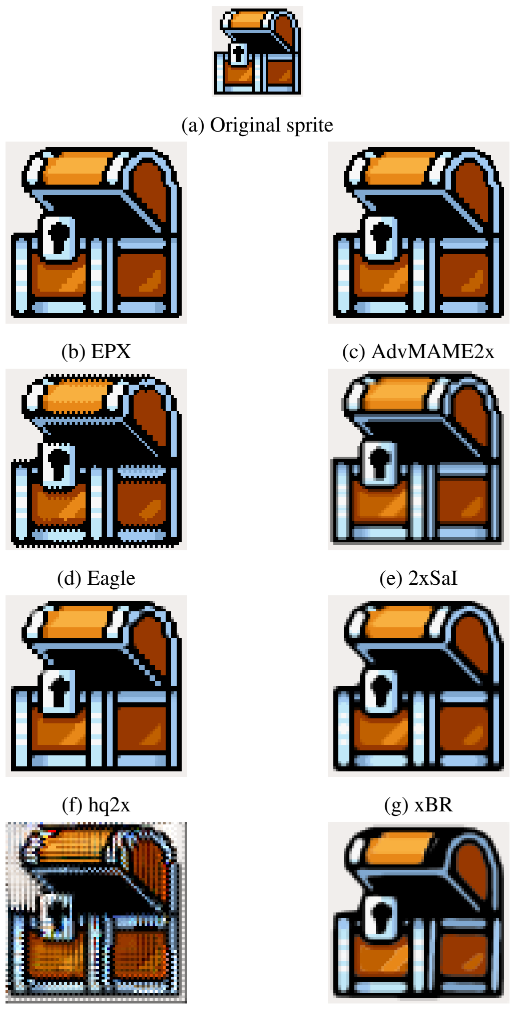


Figure 5: 2x upscaling of the Chest sprite compared to the original (Super Mario World, © Nintendo)

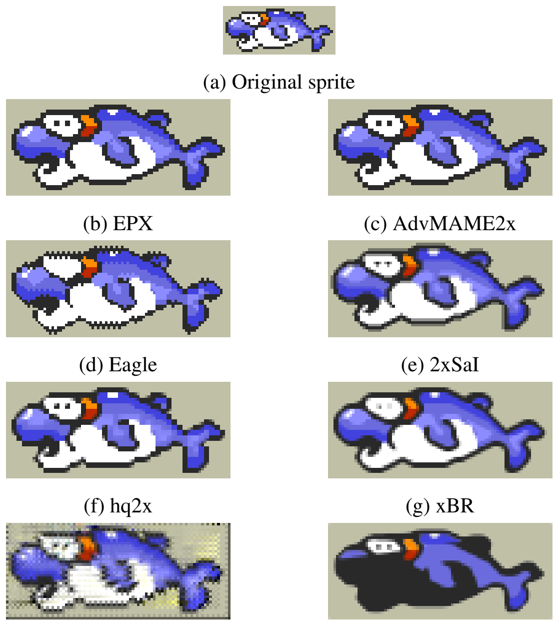


Figure 6: 2x upscaling of the Dolphin sprite compared to the original (Super Mario World, © Nintendo)

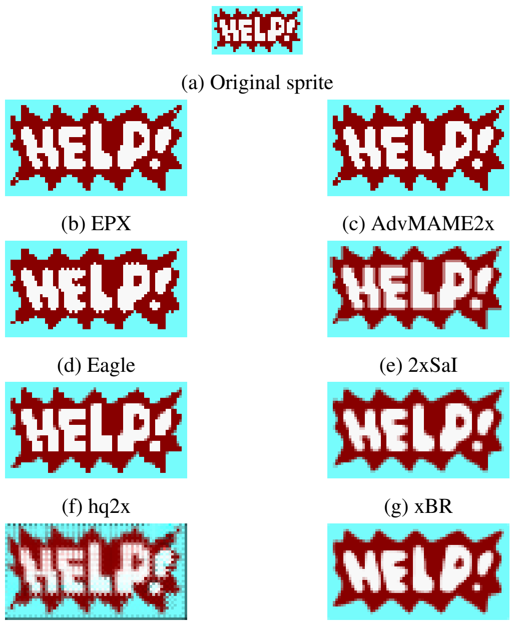
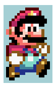
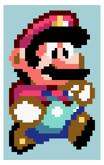
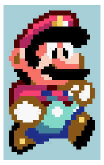
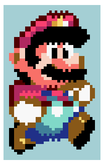


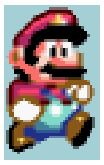
Figure 7: 2x upscaling of the Help sprite compared to the original (Super Mario World, © Nintendo)

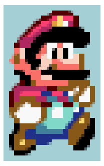

(a) Original sprite

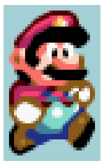

(b) EPX

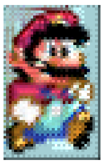

(c) AdvMAME2x


(d) Eagle


(e) 2xSaI


(f) hq2x


(g) xBR


(h) NEDI

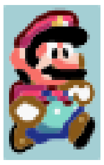
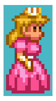
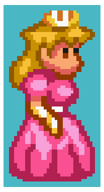
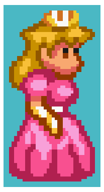
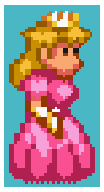

(i) Kopf-Lischinski

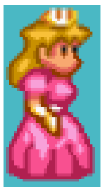
Figure 8: 2x upscaling of the Mario sprite compared to the original (Super Mario World, © Nintendo)

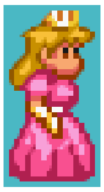

(a) Original sprite

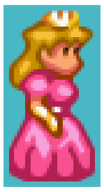

(b) EPX

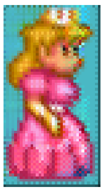

(c) AdvMAME2x


(d) Eagle


(e) 2xSaI


(f) hq2x


(g) xBR


(h) NEDI

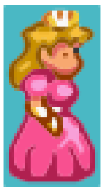
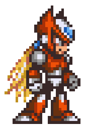
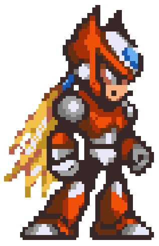
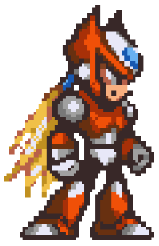
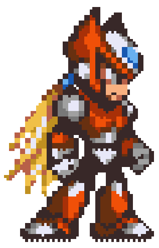

(i) Kopf-Lischinski

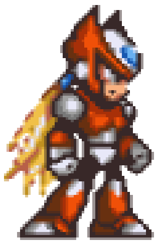
Figure 9: 2x upscaling of the Princess Peach sprite compared to the original (Super Mario All Stars, © Nintendo)

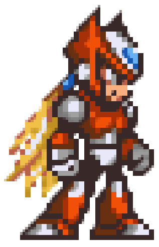

(a) Original sprite

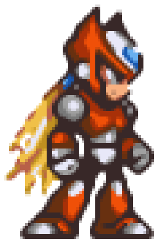

(b) EPX

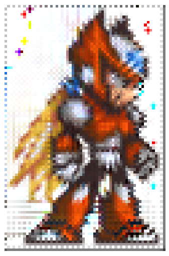

(c) AdvMAME2x


(d) Eagle


(e) 2xSaI


(f) hq2x


(g) xBR


(h) NEDI

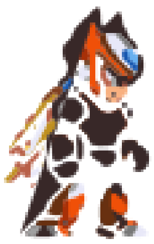
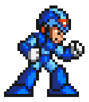
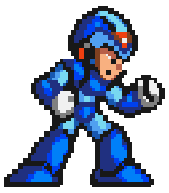
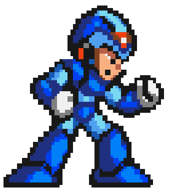
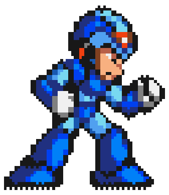

(i) Kopf-Lischinski

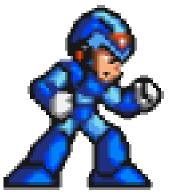
Figure 10: 2x upscaling of the Zero sprite compared to the original (Mega Man X, © Capcom)

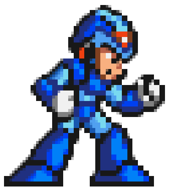

(a) Original sprite

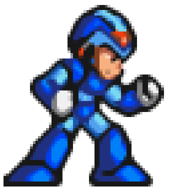

(b) EPX

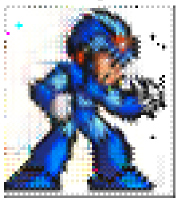

(c) AdvMAME2x


(d) Eagle


(e) 2xSaI


(f) hq2x


(g) xBR


(h) NEDI

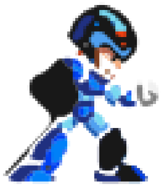
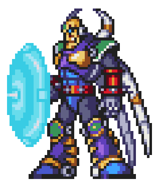
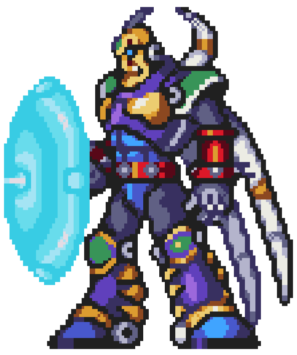
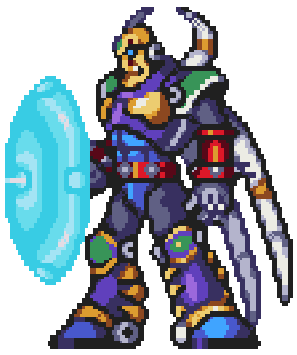
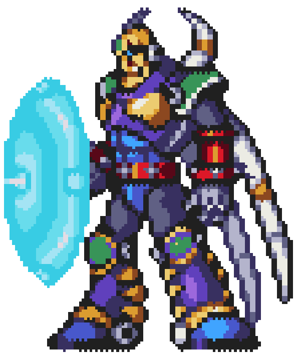

(i) Kopf-Lischinski

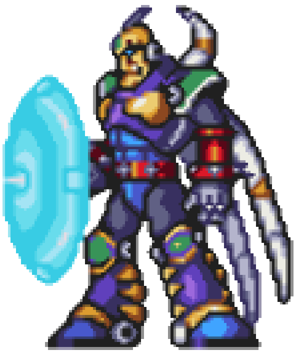
Figure 11: 2x upscaling of the X sprite compared to the original (Mega Man X, © Capcom)

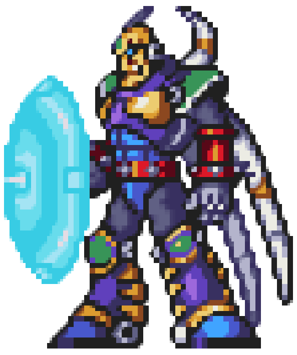

(a) Original sprite

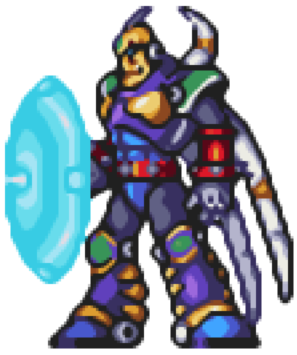

(b) EPX

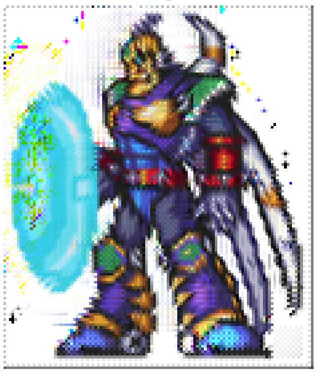

(c) AdvMAME2x


(d) Eagle


(e) 2xSaI


(f) hq2x


(g) xBR


(h) NEDI

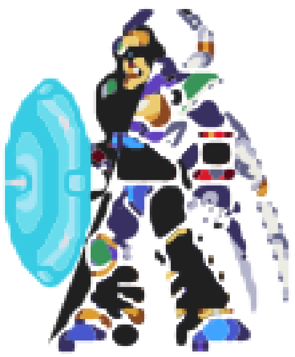
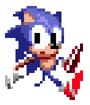
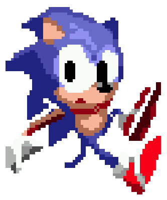
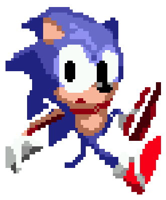
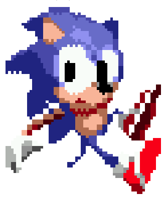

(i) Kopf-Lischinski

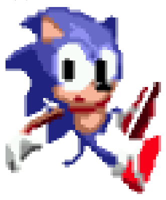
Figure 12: 2x upscaling of the Sigma sprite compared to the original (Mega Man X3, © Capcom)

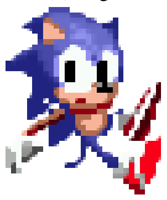

(a) Original sprite

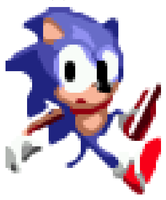

(b) EPX

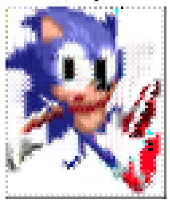

(c) AdvMAME2x


(d) Eagle


(e) 2xSaI


(f) hq2x


(g) xBR


(h) NEDI

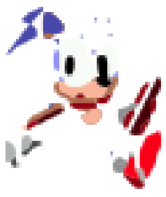

(i) Kopf-Lischinski

Figure 13: 2x upscaling of the Sonic sprite compared to the original (Sonic The Hedgehog, © SEGA)

References

[Fa02] FA D. L. K.: 2xsai : The advanced 2x scale and interpolation engine, Feb 2002. URL: https://vdnoort.home.xs4all.nl/emulation/2xsai/. 2

[Hyl12] HYLLIAN: Xbr algorithm tutorial, Sep 2012. URL: https://forums.libretro.com/t/xbr-algorithm-tutorial/123. 2

[ins14] INSOURIS:, Jun 2014. URL: http://blog.pkh.me/p/19-butchering-hqx-scaling-filters.html. 2

[KL11] KOPF J., LISCHINSKI D.: Depixelizing pixel art. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011) 30*, 4 (2011), 99:1 – 99:8. 3

[LO01a] LI X., ORCHARD M. T.: New edge-directed interpolation. *IEEE transactions on image processing 10*, 10 (2001), 1521–1527. 1

[LO01b] LI X., ORCHARD M. T.: New edge-directed interpolation. *IEEE transactions on image processing 10*, 10 (2001), 1521–1527. 2

[Tho99] THOMAS K.: Fast blit strategies: A mac programmer's guide, 1999. URL: http://preserve.mactech.com/articles/mactech/Vol.15/15.06/FastBlitStrategies/index.html. 1

[Tze04] TZENG F.-Y.: Adaptive new edge-directed interpolation 200, Apr 2004. URL: https://cs.ucdavis.edu/. 2

[Wik22] WIKIPEDIA CONTRIBUTORS: Pixel-art scaling algorithms — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Pixel-art_scaling_algorithms&oldid=1115548684, 2022. [Online; accessed 21-October-2022]. 2

[zin07] ZINOB:, Jan 2007. URL: https://everything2.com/index.pl?node_id=1859453. 2