Leistungsstarke und effiziente Bildinterpolation

Bastian Erdnüß[1,2] und Thomas Müller[1]

[1] Fraunhofer IOSB (Institute of Optronics,
System Technologies and Image Exploitation),
Fraunhoferstr. 1, 76131 Karlsruhe, Germany
[2] Institut of Photogrammetry and Remote Sensing (IPF),
Karlsruher Institut of Technology (KIT), 76128 Karlsruhe, Germany

Zusammenfassung Bildinterpolation ist Teil vieler Algorithmen zur Bildverarbeitung. Bei der Wahl der Methode wird meist ein Kompromiss zwischen Laufzeit und Qualität getroffen, der oft zu ungünstig für die Bildqualität ist. Ziel ist es, ein schnelles Verfahren mit hoher Qualität auf aktueller Hardware zu finden.

Keywords Bikubische Bildinterpolation, Lánczos-Interpolation

1 Einleitung

In dieser Arbeit werden verschiedene Interpolationsverfahren hinsichtlich ihrer Laufzeit und Ergebnisqualität untersucht. Die Interpolation ist wichtiger Bestandteil zahlreicher Bildverarbeitungsverfahren wie Bildreferenzierung und -mosaikierung. Bei der Verarbeitung von Live-Videodaten können dabei hohe Anforderungen an die Laufzeit der Interpolation bestehen. Grafikkarten etwa bieten häufig eine hochoptimierte bilinare Interpolation in ihren Texture Units an. Wenn die Qualität jedoch für eine Aufgabenstellung nicht ausreicht, muss auf rechenzeitintensivere Verfahren zurückgegriffen werden. Interpolationsverfahren werden zwar seit Jahrzehnten ausführlich untersucht, jedoch ändert sich mit der Weiterentwicklung der Hardware und der Verbreitung neuer Videostan-

dards (hinsichtlich Auflösung und Framerate) fortlaufend der Kompromiss, der zwischen Qualität und Laufzeit der Interpolationsverfahren getroffen werden kann.

Im Folgenden werden Interpolationsverfahren untersucht, die prinzipiell für die Echtzeitanwendung geeignet sind. Aus diesem Grund beschränkt sich dieser Artikel auf die Bildinterpolation mittels linearer separabler Filter mit beschränkten Trägern. Die Laufzeiten der Verfahren werden auf CPU und GPU ermittelt. Die Interpolationen werden auf verschiedene Testbilder mit häufig anzutreffenden Störungen angewandt wie Aliasing, Kompressionsartefakte und Farbrauschen. Um die Qualität der Verfahren zu vergleichen, wird die Interpolation iterativ mehrfach angewandt und die Degradierung der Bildqualität gegenüber dem Originalbild betrachtet.

2 Grundlagen

Ein Computerbild ist ein zweidimensionales $M \times N$-Tableau $I_{i,j} \in [0,1], i \in \{1,2,\dots,M\}, j \in \{1,2,\dots,N\}$ von Grauwerten. Die Grauwerte sind hier auf das Intervall $[0,1]$ normiert, wobei 0 schwarz und 1 weiß darstellt. Bei Farbbildern werden die drei Farbkanäle unabhängig voneinander behandelt und das Bild wird interpoliert, als ob es sich um drei Grauwertbilder handeln würde. Daher wird zur Vereinfachung im Folgenden von Grauwertbildern ausgegangen. Ziel der Bildinterpolation ist es, eine Funktion $f : \mathbb{R}^2 \to \mathbb{R}$ anzugeben, die $I_{i,j}$ interpoliert, für die also

$$f(i,j) = I_{i,j} \tag{2.1}$$

für $i \in \{1,\dots,M\}$ und $j \in \{1,\dots,N\}$ gilt. Dadurch ist es möglich, die Grauwerte für das Bild nicht nur im Pixelraster $(i,j) \in \mathbb{N}^2$ sondern an beliebigen Zwischenstellen $(x,y) \in \mathbb{R}^2$ anzugeben.

Es gibt einige weitere Eigenschaften neben der Interpolationsbedingung (2.1), die ein Interpolationsverfahren haben kann oder die wünschenswert sind. Hier beschränken wir uns auf lineare Interpolationsverfahren, die separabel und lokal sind. Lokal bedeutet, dass zur Interpolation nur die Bildwerte in einer kleinen Umgebung des zu interpolierenden Punktes benötigt werden, und separabel bedeutet, dass die 2D-Bildinterpolation durch mehrmalige Anwendung

von 1D-Interpolationen auf den Bildzeilen und deren Ergebnissen entlang der Spalten gefunden werden kann (siehe Abb. 2.1).

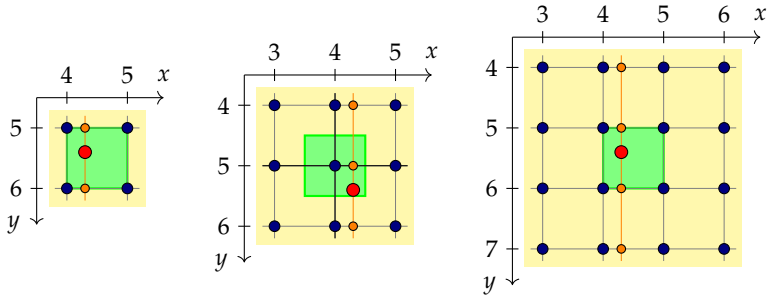

Abbildung 2.1: Separierbare Interpolation auf (4.3, 5.4) am Beispiel einer 2×2-, einer 3×3- und einer 4×4-Umgebung. Zunächst wird entlang der Zeilen auf die x-Koordinate 4.3 interpoliert (orangene Punkte). Anschließend werden diese entlang der Spalte (orangene Linie) auf die y-Koordinate 5.4 interpoliert (roter Punkt, Ergebnis der Interpolation). Alle interpolierten Punkte im grünen Quadrat hängen jeweils von denselben Bildpunkten $I_{i,j}$ der Umgebung ab, lediglich in unterschiedlichen Gewichtungen.

3 Methode

Folgende Interpolationstypen werden betrachtet:

- Nearest-Neighbour-Interpolation [1],
- bilineare Interpolation [1, 2],
- bikubische Interpolation [1, 2] (in verschiedenen Varianten),
- Lánczos-Interpolation [1, 3] (mit verschiedenen Größen).

Die ersten beiden Varianten dienen als Referenz für die schnellsten und die letzte Variante als Referenz für die qualitativ besten Interpolationsmethoden. Durch Wahl geeigneter Bildgradienten zur Berechnung der bikubischen Interpolation wird versucht eine Methode zu finden, die ähnlich gute Ergebnisse wie die Lánczos-Interpolation liefert bei gleichzeitig geringerer Rechenzeit.

3.1 Interpolationsverfahren

Beim Nearest-Neighbour-Verfahren wird der Wert $f(x,y)$ durch den Farbwert $I_{i,j}$ des nähestgelegenen Pixels (i,j) mit $i = [x]$ und $j = [y]$ interpoliert, wobei $[\cdot]$ der Rundungsoperator ist. Das Bild wird gewissermaßen mittels Treppenstufen durch die Pixel interpoliert.

Bei der bilinearen Interpolation werden in den beiden Zeilen $j = \lfloor y \rfloor$ und $j = \lfloor y \rfloor + 1$ Zwischenwerte $f_j(x)$ berechnet und aus diesen $f(x,y)$ gebildet. Die Formeln dazu sind

$$f_j(x) = (i+1-x)I_{i,j} + (x-i)I_{i+1,j} \ , \tag{3.1}$$

$$f(x,y) = (j+1-y)f_j(x) + (y-j)f_{j+1}(x) \tag{3.2}$$

mit $i = \lfloor x \rfloor$ und dem Abrundungsoperator $\lfloor \cdot \rfloor$. Zwischen benachbarten Punkten wird dabei auf deren Verbindungsgeraden interpoliert.

Bei der Lánczos-Interpolation werden die Bildzeilen mit dem Lánczos-Kernel

$$l_a(x) = \begin{cases} \operatorname{sinc}(\pi x)\operatorname{sinc}(\frac{\pi x}{a}) & \text{für } |x| \leq a \\ 0 & \text{sonst} \end{cases} \tag{3.3}$$

gefaltet, wobei $a \in \mathbb{R}$ die Größe der betrachteten Umgebung bestimmt und $\operatorname{sinc}(x) = \frac{\sin x}{x}$ ist. D. h. es werden erst Zwischenwerte $f_j(x)$ und daraus $f(x,y)$ nach den Formeln

$$f_j(x) = \sum_i l_a(x-i)I_{i,j} \ , \tag{3.4}$$

$$f(x,y) = \sum_j l_a(y-j)f_j(x) \tag{3.5}$$

berechnet, wobei nur Summanden mit $i \in [x-a, x+a]$ und $j \in [y-a, y+a]$ einen Beitrag zur Summe liefern und $f_j(x)$ auch nur für entsprechende j berechnet werden muss.

Bei der bikubischen Interpolation wird zwischen zwei benachbarten Punkten mittels eines kubischen Polynoms interpoliert. Neben den beiden Randpunkten werden auch die Steigungen des Polynoms an den Rändern vorgegeben, wodurch es eindeutig bestimmt

ist. Sind G_k die Funktionswerte in einer Zeile und H_k deren Steigungen, so wird zwischen k und $k+1$ mit

$$g(x) = s^2(1+2t)G_k + t^2(1+2s)G_{k+1} + s^2tH_k - st^2H_{k+1} \quad (3.6)$$

interpoliert, wobei $s = k+1-x$ und $t = x-k$ die beiden Abstände von $x \in [k, k+1]$ zu den umliegenden beiden Punkten sind. Um die Abhängigkeiten von den Daten explizit zu machen, wird $g(x)$ auch ausführlicher als $g(x, G_k, G_{k+1}, H_k, H_{k+1})$ notiert.

Es bezeichne I^x das Gradientenbild von I in x-Richtung, I^y das Gradientenbild in y-Richtung und I^{xy} dessen gemischte zweite Ableitung, jeweils auf den ganzzahligen Pixeln (i, j) definiert. Analog zur bilinearen Interpolation werden zunächst in den beiden Zeilen $j = \lfloor y \rfloor$ und $j = \lfloor y \rfloor + 1$ Zwischenwerte $f_j(x)$ aus I und I^x berechnet. Um jedoch die Interpolation in y-Richtung durchführen zu können, müssen auch die y-Gradienten $f_j^y(x)$ aus I^y und I^{xy} nach x interpoliert werden. Anschließend kann $f(x, y)$ aus $f_j(x)$ und $f_j^y(x)$ in y-Richtung interpoliert werden. Die Formeln dazu sind

$$f_j(x) = g(x, I_{i,j}, I_{i+1,j}, I_{i,j}^x, I_{i+1,j}^x) \ , \quad (3.7)$$

$$f_{j+1}(x) = g(x, I_{i,j+1}, I_{i+1,j+1}, I_{i,j+1}^x, I_{i+1,j+1}^x) \ , \quad (3.8)$$

$$f_j^y(x) = g(x, I_{i,j}^y, I_{i+1,j}^y, I_{i,j}^{xy}, I_{i+1,j}^{xy}) \ , \quad (3.9)$$

$$f_{j+1}^y(x) = g(x, I_{i,j+1}^y, I_{i+1,j+1}^y, I_{i,j+1}^{xy}, I_{i+1,j+1}^{xy}) \ , \quad (3.10)$$

$$f(x, y) = g(y, f_j(x), f_{j+1}(x), f_j^y(x), f_{j+1}^y(x)) \ . \quad (3.11)$$

Die eigentliche Interpolation wird mit einer festen Anzahl an Rechenschritten und Speicherzugriffen ausgeführt. Die Qualität der Interpolation hängt jedoch von der Qualität der Gradientenbilder I^x, I^y und I^{xy} ab. Diese können mittels disketer Faltung erzeugt werden. Von der Kernelgröße dieser Faltung hängt die Gesamtlaufzeit der Interpolation ab. Da die Berechnung der Gradientenbilder jedoch im Pixelraster erfolgt, kann diese effizienter implementiert werden, als es bei der Interpolation der Fall ist. Denn Zwischenergebnisse können wiederverwendet werden, Vektorsierung kann besser genutzt werden und der Prozessor-Cache wird weniger belastet.

3.2 Diskrete Ableitungsoperatoren

Die Gradientenbilder I^x, I^y und I^{xy} werden wiefolgt aus dem Eingangsbild I berechnet. Im ersten Schritt wird I^x zeilenweise aus I berechnet und (ggf. parallel dazu) mit dem gleichen Verfahren I^y spaltenweise aus I. Anschließend wird ebenfalls mit dem gleichen Verfahren I^{xy} zeilenweise aus I^y berechnet. Daher ist es hier ausreichend, die Berechnung von I^x aus I zu behandeln.

Im einfachsten Fall wird I^x durch den Differenzenquozienten

$$I^x_{i,j} = \frac{I_{i+1,j} - I_{i-1,j}}{2} \tag{3.12}$$

approximiert. Das entspricht der Faltung mit dem schiefsymmetrischen Kernel $[1, 0, -1]/2 = [\frac{1}{2}, 0, -\frac{1}{2}]$ in x-Richtung und führt zur Standardvariante der bikubischen Interpolation, wie sie in den meisten Softwarebibliotheken implementiert ist. Andere Ableitungsoperatoren werden gebildet, indem mit größeren schiefsymmetrischen Kerneln gefaltet wird. Diese haben die Form $[A_n, \ldots, A_1, 0, -A_1, \ldots, -A_n]$ und ihre Anwendung kann effizient als

$$I^x_{i,j} = \sum_{k=1}^{n} A_k (I_{i+k,j} - I_{i-k,j}) \tag{3.13}$$

implementiert werden. Daher wird bei Ableitungskerneln hier nur die Hälfte der Koeffizienten $[A_1, \ldots, A_n]$ aufgelistet und z. B. der Ableitungskernel zu Gleichung (3.12) verkürzt als $[1]/2 = [0.5]$ geschrieben.

Zusätzlich kann in y-Richtung auch noch geglättet werden. Wird z. B. beim Ableitungskernel $[0.5]$ in y-Richtung mit dem Glättungskernel $[1, 2, 1]/4$ gefaltet, erhält man den Sobel-Operator [1]. Zu allen Ableitungskerneln wurde auch eine Glättung in Querrichtung erprobt. Diese Varianten haben jedoch durchweg zu schlechteren Ergebnissen geführt (deutliche Tiefpassfilterwirkung im Ergebnisbild), sodass sie hier nicht weiter verfolgt werden.

Drei Klassen von Ableitungskerneln werden betrachtet, sie werden als Diff, OptDiff und LanczosDiff bezeichnet. Diff sind die klassischen Ableitungskernel, die die größte Konsistenz im Fourier-Raum

haben, wenn der Pixelabstand gegen 0 geht. OptDiff sind dagegen unter Berücksichtigung der diskreten Pixel über das gesamte Fourier-Spektrum optimiert. Die Koeffizienten für beide stammen aus den Tabellen B.1 und B.2 in [4], siehe Tabelle 1. Das rekursive Filter in Tabelle B.3 aus [4] wurde ebenfalls getestet, allerdings konnten damit keine guten Ergebnisse produziert werden.

Tabelle 1: Koeffizienten für Diff und OptDiff für verschiedene n gemäß [4]

n	**Diff**	**OptDiff**
2	$[8, -1] / 12$	$[0.758, -0.129]$
3	$[45, -9, 1] / 60$	$[0.848, -0.246, 0.048]$
4	$[672, -168, 32, -3] / 840$	$[0.896, -0.315, 0.107, -0.0215]$
5	$[2100, -600, 150, -25, 2] / 2520$	$[0.924, -0.360, 0.152, -0.0533, 0.0109]$

Das Ziel von LanczosDiff ist es, möglichst ähnlich zur Lánczos-Interpolation zu sein. Wird eine Bildzeile j mittels Lánczos-Interpolation zu $f_j(x) = \sum_i l_a(x-i) I_{i,j}$ interpoliert, kann daraus deren Ableitung $I^x_{k,j} = f'_j(k) = \sum_i l'_a(k-i) I_{i,j}$ an den Pixelpositionen berechnet werden, um das Gradientenbild I^x zu bilden. Das Ergebnis entspricht der diskreten Faltung mit dem schiefsymmetrisch fortgesetzten Kernel $[l'_a(1), l'_a(2), \ldots, l'_a(n)]$ mit $n = \lfloor a \rfloor$ und

$$l'_a(k) = \frac{(-1)^k}{k} \operatorname{sinc}\left(\frac{\pi k}{a}\right) . \tag{3.14}$$

4 Ergebnisse

Zur genauen Analyse und Verdeutlichung der Wirkung der unterschiedlichen Verfahren wird ein Bild mehrmals interpoliert, sodass Interpolationsfehler verstärkt und die Ergebnisse qualitativ gut unterscheidbar werden. Hierzu wird ein Eingabebild m Mal um das Bildzentrum mit dem Winkel $360^\circ / m$ rotiert. Die sich ergebende Volldrehung um 360° wird anschließend mit dem Originalbild visuell verglichen. Dabei können zwar subjektive Eindrücke in die Bewertung einfließen, jedoch hat sich eine automatische Auswertung über Differenzbilder als nicht zielführend erwiesen, da diese vor allem einen Tiefpassfiltereffekt der Interpolation nicht angemessen

Abbildung 4.1: Testbild (links) und mittels Lánczos-Interpolation mit $a = 6$ berechnete, um 15° rotierte Variante (erste Rotation zu $m = 24$; rechts).

gewichtet. Ebenso ist eine Auswertung über das Fourierspektrum nicht zweckmäßig, da Realdaten etwa bei Konsumer-Sensoren Aliasing enthalten können.

Abb. 4.1 zeigt ein Eingabebild und veranschaulicht das Vorgehen. In Abb. 4.2 sind die Ergebnisse einiger Interpolationsverfahren an einem Ausschnitt dieses kontrastscharfen Eingabebildes für $m = 24$ dargestellt. Als Umgebungsgröße wird 12×12 verwendet (d. h. $a = 6$ bei der Lánczos-Interpolation und $n = 5$ bei allen Varianten der bikubischen Interpolation), was sich als günstig erwiesen hat. Wie man erkennen kann, erzeugt die Nearest-Neighbour-Interpolation zwar ein scharfes Ergebnis, ist aber bei feinen Strukturen nicht ortserhaltend. Wie erwartet ist die bilineare Interpolation am unschärfsten und die standard bikubische auch deutlich unscharf. OptDiff bietet diesbezüglich eine deutliche Verbesserung. Das schärfste Ergebnis mit der besten Detailerhaltung liefert die Lánczos-Interpolation. Bei genauer Betrachtung von Abb. 4.2 fällt jedoch auf, dass diese auch den stärksten Klingeleffekt an scharfen Farbkanten erzeugt, d. h. diese vervielfacht.

Das Diff-Ergebnis (ohne Abbildung) liegt bzgl. Detailtreue zwischen standard bikubisch und OptDiff. Das Ergebnis mit LanczosDiff (ohne Abbildung) ist ebenso gut wie OptDiff, nur geringfügig anders.

Wie die Rechenzeiten der Verfahren in Tabelle 2 zeigen, benötigen OptDiff und LanczosDiff knapp das Doppelte der Rechenzeit der standard bikubischen Interpolation, wenn man nur eine CPU zur Verfügung hat. In Anbetracht des signifikanten Bildqualitätgewinns

ist das durchaus akzeptabel. Lediglich Lánczos liefert noch bessere Ergebnisse, ist aber hinsichtlich Rechenzeit zumeist in der Praxis ungeeignet.

Ganz anders sieht es auf der GPU aus. Hier lässt sich Lánczos erstaunlich effizient realisieren. Die Berechnungen dauern nur unbedeutend länger als standard bikubisch bei wesentlich besseren Ergebnissen. OptDiff und LanczosDiff rechnen sogar etwas länger als Lánczos und sind weder hinsichtlich Ergebnis noch Rechenzeit eine echte Alternative dazu. Sie wären aber für die meisten Anwendungen schnell genug und erzielen auch bessere Ergebnisse als standard bikubisch.

Tabelle 2: Rechenzeiten der Interpolationsverfahren für ein Bild der Größe 960×640 Pixel auf einer Intel i7-4770 CPU mit 3.40 GHz bzw. einer NVidia GeForce 940MX GPU. Das Pluszeichen verdeutlicht die Verteilung auf Ableitungsberechnung und eigentliche Interpolation.

Interpolation	**CPU**	**GPU**
Nearest-Neighbour	24 ms	3.7 ms
bilinear	31 ms	4.1 ms
standard bikubisch	63 ms	11.9 ms
Diff	33 ms + 82 ms = 115 ms	20.0 ms + 4.4 ms = 24.4 ms
OptDiff	33 ms + 82 ms = 115 ms	19.9 ms + 4.4 ms = 24.3 ms
LanczosDiff	33 ms + 82 ms = 115 ms	18.1 ms + 4.4 ms = 22.5 ms
Lánczos	21 s	17.5 ms

Abschließend werden in Abb. 4.3 noch ein Bild mit starken MPG-Artefakten aus einem hochgradig komprimierten Video und in Abb. 4.4 ein Bildbeispiel mit starkem Farbrauschen betrachtet, um die Auswirkung von Störeinflüssen zu untersuchen. Wie man in beiden Abbildungen erkennen kann, ergeben sich die gleichen Aussagen hinsichtlich Bildqualität wie beim obigen Eingabebild, das bedeutet, die Verfahren erweisen sich als robust gegenüber solchen Störungen.

5 Zusammenfassung

Heutige GPUs erlauben eine hocheffiziente Anwendung der Lánczos-Interpolation, welche qualitativ bestmögliche Resultate mit

hoher Detailtreue liefert, und das bei einer ähnlichen Rechenzeit wie die standard bikubische Interpolation.

Auf der CPU ist die Lánczos-Interpolation in der Praxis meist zu langsam. Hier erzielen die OptDiff- und LanczosDiff-Interpolation hochqualitative Ergebnisse bei lediglich einer knapp doppelt so hohen Rechenzeit wie die standard bikubische Interpolation.

Literatur

1. *Wikipedia: The Free Encyclopedia*, (abgerufen: 30. September 2020). [Online]. Available: https://www.wikipedia.org
2. W. H. Press et al., *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. USA: Cambridge University Press, 1992.
3. W. Burger and M. J. Burge, *Principles of Digital Image Processing: Core Algorithms*, ser. Undergraduate Topics in Computer Science. Springer London, 2009.
4. H. Scharr, "Optimale Operatoren in der Digitalen Bildverarbeitung," *Dissertation, Universitätsbibliothek Heidelberg*, 2000.

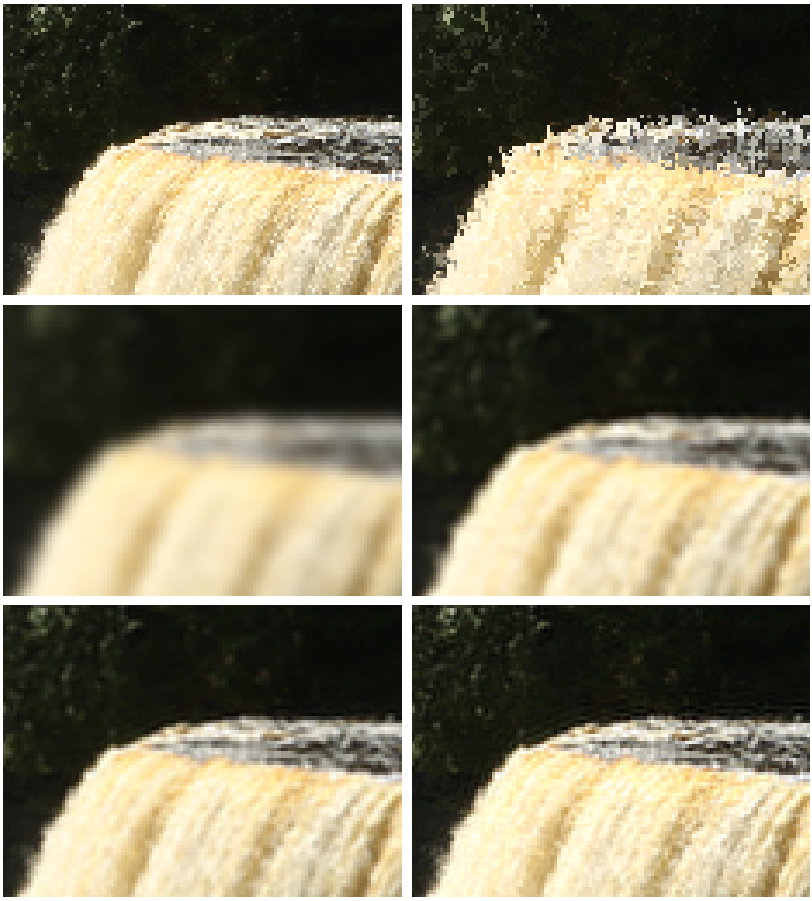

Abbildung 4.2: Ausschnitt des Testbildes aus Abb. 4.1 (oben links) sowie Interpolationen mit Nearest-Neighbour (oben rechts), bilinear (Mitte links), standard bikubisch (Mitte rechts), OptDiff (unten links) und Lánczos (unten rechts). Das Bild wurde jeweils $m = 24$ Mal rotiert.

Abbildung 4.3: Testbild (oben links) mit Ausschnittsvergrößerung (oben rechts) sowie Interpolationen bilinear (Mitte links), bikubisch (Mitte rechts), Opt-Diff (unten links) und Lánczos (unten rechts). Das Bild wurde jeweils $m = 24$ Mal rotiert.

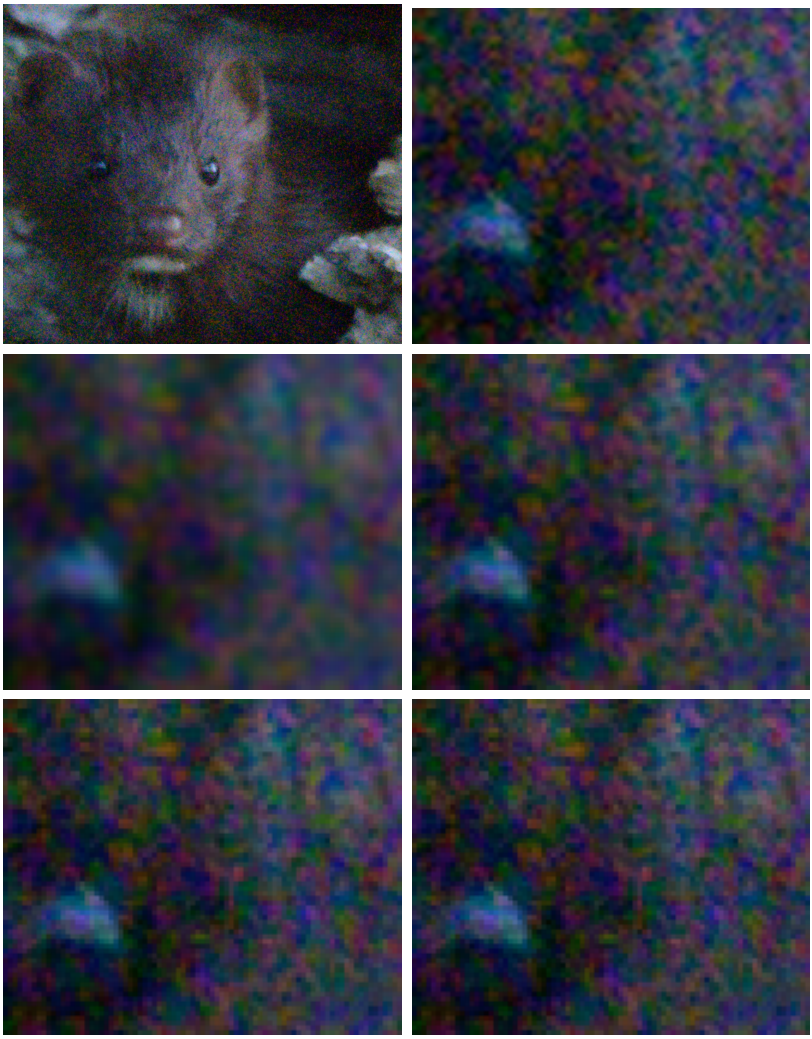

Abbildung 4.4: Testbild (oben links) mit Ausschnittsvergrößerung (oben rechts) sowie Interpolationen bilinear (Mitte links), bikubisch (Mitte rechts), Opt-Diff (unten links) und Lánczos (unten rechts). Das Bild wurde jeweils $m = 24$ Mal rotiert. [Quelle: Foto von Bill Maynard]