IMAGE INTERPOLATION ON THE CPU AND GPU USING LINE RUN SEQUENCES

Dirk Frommholz

DLR Institute of Optical Sensor Systems, Berlin, Germany - dirk.frommholz@dlr.de

Commission II, WG II/2

KEY WORDS: Interpolation, Inverse Distance Weighting, Rasterization, Line Runs, Multithreading, OpenCL.

ABSTRACT:

This paper describes an efficient implementation of an image interpolation algorithm based on inverse distance weighting (IDW). The time-consuming search for support pixels bordering the voids to be filled is facilitated through gapless sweeps of different directions over the image. The scanlines needed for the sweeps are constructed from a path prototype per orientation whose regular substructures get reused and shifted to produce aligned duplicates covering the entire input bitmap. The line set is followed concurrently to detect existing samples around nodata patches and compute the distance to the pixels to be newly set. Since the algorithm relies on integer line rasterization only and does not need auxiliary data structures beyond the output image and weight aggregation bitmap for intensity normalization, it will run on multi-core central and graphics processing units (CPUs and GPUs). Also, occluded support pixels of non-convex void patches are ignored, and over- or undersampling close-by and distant valid neighbors is compensated. Runtime and accuracy compared to generated IDW ground truth get evaluated for the CPU and GPU implementation of the algorithm on single-channel and multispectral bitmaps of various filling degrees.

1. INTRODUCTION

During the generation of digital surface models (DSMs), digital terrain models (DTMs), true-ortho mosaics (TOMs), 3D model textures and other two-dimensional remote sensing data products, areas where no information is available frequently occur. The appearance of void patches may have a variety of causes, for instance, the lack of available data when the acquisition campaign on the targeted scene is subject to economic constraints that prohibit a comprehensive coverage. Invalid samples may also result from occlusions which cannot be worked around by adapting the sensor pose or from applied processing tools when the underlying algorithms run into ambiguities that they can possibly detect but not adequately resolve. An example for this behavior in photogrammetry is stereo image matching as a prerequisite for 3D object reconstruction. While searching for corresponding content in overlapping pairs of oriented bitmaps, homogeneously or periodically textured surfaces translate into sound but highly divergent disparity values. Such disparities will fail a subsequent consistency check and get flagged as invalid. Further, when a scene containing sensitive objects is acquired, these parts may have to be intentionally replaced by dedicated void samples before the recorded data can be passed to parties that do not have the appropriate security clearance.

In any case, many applications that operate on raster data derived through remote sensing require their inputs to be continuous and cannot handle nodata areas. This applies particularly to simulation and visualization tasks. For instance, when the impact of flooding or the propagation of sound is to be physically modeled based on digital elevation information, nodata areas may influence the dynamics of the waves and distort the output. In 3D rendering, the display of terrain data or color imagery like for mesh textures containing invalid samples might come along with visible glitches that negatively affect the immersion experience. Therefore, when raster bitmaps contain patches without information and there are no external sources at hand to cover the voids, the gaps will have to be filled up from the available valid image pixels by interpolation.

2. RELATED WORK AND MOTIVATION

There are several well-studied generic and application-specific approaches that address the interpolation problem, and some of them have been implemented as software modules for commercial and open-source geographic information systems (GIS). Among the deterministic methods solely operating on the available image content, inverse distance weighting (IDW) (Shepard, 1968) calculates the intensities of missing samples as a linear combination of existing near-by image pixels. The individual contribution of these neighbors is inversely proportional to their (Euclidean) distance to the location of the nodata sample to be filled. Comprehensive IDW implementations run with quadratic time complexity regarding the pixel count of the input data due to the necessary search for support points which can be located anywhere within the image frame. The computational effort can be reduced to linearithmic and even linear time when spatial partitioning using e.g. kd-trees or subsampling techniques are deployed on the valid near-by samples. However, this involves external non-image data structures, and any reduction in supporting pixels on which the linear combination of IDW is hinged on presumably will introduce artifacts like intensity discontinuities and star-shaped patterns.

Another deterministic approach for filling voids in raster data is spline interpolation, that is, constructing two-dimensional piecewise polynomial differentiable functions going exactly through the valid pixels and sampling them at the nodata positions (Franke, 1984). Similar to IDW, this technique known from image resizing will preserve the intensity values in the support points whose number directly affects the processing speed. However, the use of polynomial curves limits the shape of the interpolants depending on their degree and may introduce strong over- and undershots on high-frequency image content near the boundaries of void patches. Instead of using polynomials, natural

neighbor interpolation (Sibson, 1981) derives the contribution of valid intensity values to set void pixels geometrically from the Voronoi tessellation of the input bitmap. Originally, the contribution of neighbor pixels surrounding a particular nodata area has been obtained as the share a fictively constructed Voronoi cell would consume from the existing regions of the initial partition. Since this approach requires multiple calculations of convex polygon areas, computationally more efficient weighting formulas have also been proposed using edge ratios (Kotulak et al., 2017). In any case, natural neighbor interpolation requires additional storage for the tessellation which however can be run in linearithmic time regarding the image pixel count. Because the Voronoi diagram will be accessed in read-only mode after its creation, lock-free multithreading can be deployed to fill multiple void areas concurrently.

A statistical method for image interpolation which can be considered a generalization of inverse distance weighting is Kriging (Krige, 1951). Kriging treats the available image samples as realizations of random variables. The method attempts to determine statistical measures on the spatial dependence of the input data assuming local second-order stationarity and orientation uniformity instead of immediately utilizing the available image samples. This helps to suppress clustering effects deterministic interpolation is prone to. After model fitting, the obtained measures named variograms are used to set up a linear regression equation to be solved for the weights that control the contribution of the support points when a particular void pixel is to be predicted. Due to the construction of the variograms and the involved matrix inversions which have to be performed for each nodata element, naive Kriging not using approximating acceleration techniques is computationally more expensive than IDW. However, the method will provide an estimate for the uncertainty of the interpolated samples with respect to the underlying covariance model, and it will minimize the deviation as long as the stationarity and isotropy requirements on the existing image samples are met.

Lately, with recent advances in machine learning, inpainting methods have gained attention for image interpolation due to their visually appealing and semantically sound results (Liu et al., 2018). These approaches, which constitute a separate group of algorithms, are commonly built on convolutional neural networks (CNNs) and related composite frameworks like generative adversarial networks (GANs) (Elharrouss et al., 2019). The basic idea behind GAN-based inpainting is to obtain a generator neural network that learns to produce contextually plausible samples filling nodata bitmap areas. For this purpose, the generator is coupled to a discriminator network which has been trained on the synthesized output and true void-free imagery and attempts to correctly distinguish the interpolated from the complete bitmaps. The discriminator's decision gets backpropagated into the generator component to adjust its model weights with respect to a predefined loss function and iteratively improve its capabilities to create more coherent interpolations. When the discriminator is no longer able to differentiate between the synthesized and true samples, the generator has been successfully enabled to perform void interpolation adequately. Inpainting techniques based on CNNs tend to require a considerable amount of computational effort for the initial learning phase even on subsampled bitmaps (Chen and Haifeng, 2019). Depending on the network architecture and the availability of suitable training samples, interpolation results will show global consistency regarding the existing image content. However, there may be robustness issues on slightly varied input data and a lack of explainability which could disqualify inpainting for security-related applications (Došilović et al., 2018).

This paper will revisit the idea of deterministic image interpolation and outline an efficient approximation of the inverse distance weighting algorithm. The search for support pixels contouring the nodata areas to be completed is performed along densely packed rasterized scanlines of arbitrary directions over the entire input image. It hence exposes linear worst-case time complexity. The interpolation result depends on two intuitive parameters, i.e., the direction count effectively being a sampling factor on the boundaries of the nodata areas and a smoothness exponent. This allows to trade processing speed for coverage depending on the focus of the actual application. Using incremental integer line rasterization only, the presented algorithm operates exclusively on pixel matrices without the need for auxiliary non-bitmap storage. It therefore can be run concurrently on both multi-core general-purpose central processing units (CPUs) as well as dedicated graphics processors (GPUs). In contrast to window-based IDW implementations, occluded support pixels of non-convex nodata regions will be ignored. Also, to converge towards an optimal interpolation result, over- and undersampling of near and far valid samples is compensated. To assess the interpolation performance, runtime and accuracy compared to generated IDW ground truth will be evaluated for proof-of-concept CPU and GPU implementations of the described algorithm on single-channel and multispectral bitmaps of various filling percentages.

3. INTERPOLATION ALGORITHM DESCRIPTION

The proposed inverse distance weighting algorithm picks up an idea originally posted for semi-global stereo matching with arbitrary path orientations (Frommholz, 2020). Instead of minimizing the matching cost for each pixel and disparity value using dynamic programming, the code executed during scanline traversal will backup the position and intensities when a valid input image pixel is encountered and remember this information for the interpolation of any subsequent nodata values along the path.

More specifically, given the input image containing void areas to be completed from surrounding pixels, IDW interpolation is performed along k parallel paths $\mathbf{r_{ij}}$, $1 \leq i \leq n, 1 \leq j \leq k$ of n equiangular orientations $\mathbf{d_i} = \mathbf{d_1}, ..., \mathbf{d_n}$ to eliminate a directional bias. The paths are constructed using the four-connected component (4-cc) Bresenham algorithm known from computer graphics (Bresenham, 1965). Pixels surrounding the nodata areas will be linearly combined to replace missing samples in at most two coordinated sweeps per direction over the input image as shown by figure 1.

3.1 Primary sweep

During the primary sweep performed first, the common shape $\mathbf{r_i}$ of the lines $\mathbf{r_{ij}}$ of direction $\mathbf{d_i}$ is rasterized by running the Bresenham algorithm exactly once over the full horizontal or vertical extent of the image plane depending on the slope. Path sampling therefore must begin in one of the bitmap corners. As a result, the longest sequence of linear segments, or runs $\tilde{\mathbf{r}}_{\mathbf{il}}$, in which the horizontal or vertical coordinate remains constant is obtained for $\mathbf{r_i}$ (figure 2). The runs characterize the fast-changing major direction of the line to be discretized and come in two lengths except for the possibly truncated first and last segments (Stephenson and Litow, 2001).

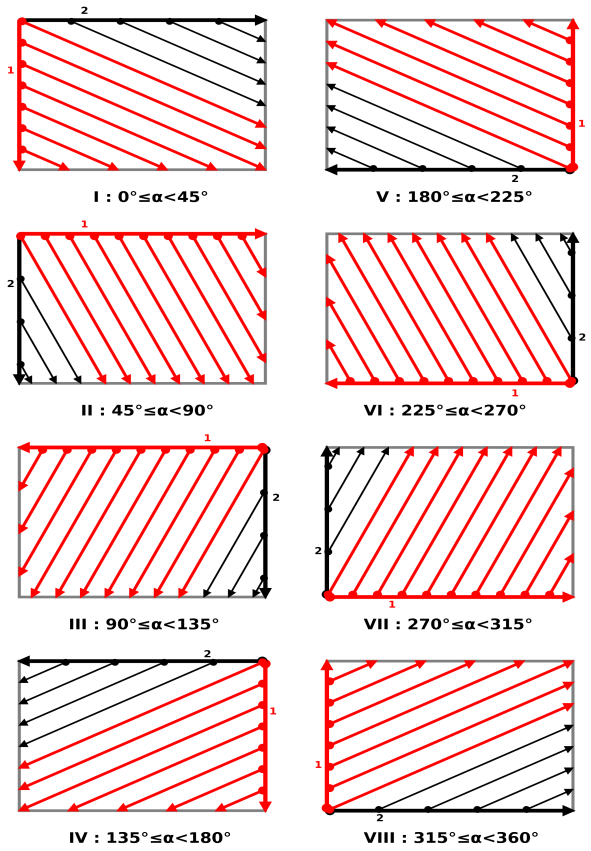

Figure 1. Primary (red) and secondary (black) sweeps over the image to be interpolated for paths with an arbitrary slope α

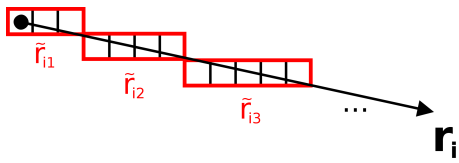

Figure 2. Runs $\tilde{\mathbf{r}}_{\mathbf{i1}}, \tilde{\mathbf{r}}_{\mathbf{i2}}, \tilde{\mathbf{r}}_{\mathbf{i3}}$ of the line prototype of shape $\mathbf{r_i}$ with a length of three (truncated start segment), four and five pixels

For IDW interpolation along a specific path over the input, the run sequence archetype is followed starting at the respective image boundary according to the sweep scheme. Each run pixel at image position $\mathbf{x}'$ gets tested for the nodata value assigned to the bitmap. If the pixel content is valid, its position $\mathbf{x} = \mathbf{x}'$ is backed up forming the last support pixel $\mathbf{p_m}(\mathbf{x})$ on the path. Also, the intensities of $\mathbf{p_m}(\mathbf{x})$ are verbatimly copied once to the output image of the same dimensions and channel count as the input bitmap. When a void pixel is hit after $\mathbf{p_m}(\mathbf{x})$, its intensities which initially have been reset to zero get incrementally updated from the position and color information of the last support pixel according to a modified version of Shepard's formula as shown in equation 1.

$$\begin{aligned} \mathbf{u}(\mathbf{x}') &= \frac{1}{\sum_{m\in M} w_{md} w_{mo}} \sum_{m\in M} w_{md} w_{mo} \mathbf{p_m}(\mathbf{x}) \\ w_{md} &= \frac{1}{d_E(\mathbf{x},\mathbf{x}')^s} = \frac{1}{\|\mathbf{x}-\mathbf{x}'\|_2^s} \\ w_{mo} &= \frac{8\, d_I(\mathbf{x},\mathbf{x}')}{n} = \frac{8\, max(|x-x'|,|y-y'|)}{n} \end{aligned} \tag{1}$$

In the equation, the pixel to be interpolated $\mathbf{u}(\mathbf{x}')$ is obtained as the linear combination of its set M of support pixels. Their contribution depends on the inverse of the Euclidean distance d_E to $\mathbf{u}(\mathbf{x}')$ raised to the power of the smoothness s and a coefficient w_{mo} for oversampling compensation. The values w_{mo} are derived from the number of support pixels for $\mathbf{u}(\mathbf{x}')$ within a discrete radius of d_I on the image raster (figure 3) divided by the direction count. Therefore, multiple contributions of closeby support pixels to the interpolation result along the n scanned paths will get neutralized by a lower share of their intensities, and valid pixels in the distance virtually will be oversampled. This effectively emulates a closed contour around $\mathbf{u}(\mathbf{x}')$ like in the ideal IDW algorithm.

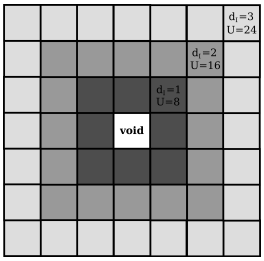

Figure 3. The $8\,d_I$ support pixels around a void pixel

If there is no prior valid pixel $\mathbf{p_m}(\mathbf{x})$ available, interpolation will not be performed for the current path instance. Because the evaluation of equation 1 happens incrementally during the sweep, the output image storing the $\mathbf{u}(\mathbf{x}')$ must feature a pixel data type that is large enough to hold the aggregated weighted intensities for all n orientations $\mathbf{d_i}$. Similarly, the set of support M is unknown in its entirety when the current line run is processed. The total weights $w_{md}w_{mo}$ hence are accumulated in a separate bitmap congruent to the input image. Normalization to preserve the average image intensity will be deferred to a final step once all directions have been dealt with.

After completing the pixel at the current image location, the procedure is repeated for the next position on the run which can be obtained through an increment or decrement of $\mathbf{x}'$ by one in the major path direction. When the current segment of the precomputed sequence is exhausted, the position of the pixel to be tested will be incremented or decremented with respect to the minor path direction, and calculation will proceed with the next line run. If the image frame is eventually left, a new path $\mathbf{r_{ij+1}}$ of the same orientation next to the current one will be processed. For this purpose, the start position inside the bitmap is reset to the original boundary and altered by one pixel in the minor line direction, and the run sequence gets rewound to its start.

The primary sweep terminates as soon as a new path $\mathbf{r_{ij+1}}$ of direction $\mathbf{d_i}$ begins outside the image. When this happens, at least 50% (diagonal paths, square image) and at most 100% (strictly horizontal or vertical paths) of the input samples will have been processed. Because adjacent rasterized paths are guaranteed to be piecewise parallel and tightly aligned when the same run sequence gets repeatedly replayed as described, each image location visited during the primary sweep is accessed exactly once. Therefore, no intensity or weight update will be omitted or performed multiple times for a particular position $\mathbf{x}'$ removing any chance for over- or underrepresented support pixels in the interpolation result.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-V-2-2022-53-2022 | © Author(s) 2022. CC BY 4.0 License.

3.2 Secondary sweep

Except for perfectly horizontal and vertical path directions, a secondary sweep will be needed to process those pixels of the input image not visited during the primary sweep. Like in the first pass, the run sequence is followed to obtain the position $\mathbf{x}'$ in order to either copy valid intensities as they are to the output bitmap or run the interpolation formula. However, the start position of the l-th secondary path along the respective image boundary is computed as the accumulated length of the runs $\tilde{\mathbf{r}}_{\mathbf{i1}}, ..., \tilde{\mathbf{r}}_{\mathbf{il}}$ of the rasterized line prototype. Also, the path gets stripped off its first l runs to be piecewise adjacent to the paths of the primary sweep (see figure 4). Thus, overlaps and gaps are avoided during pixel access between scanlines of the secondary sweep. The sweep scheme prevents collisions with the positions visited in the first pass over the input, weight and output images involved in interpolation.

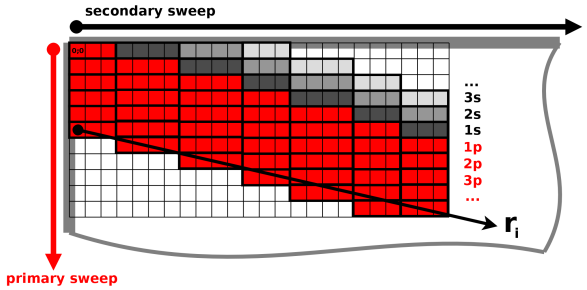

Figure 4. Truncation and shifted start of paths of shape $\mathbf{r_i}$ during the secondary sweep (shades of gray) ensure the alignment to paths of the primary sweep (red) and second pass itself

3.3 Quality and time complexity

Because the primary and secondary sweep combined cover the entire image, the incremental interpolation approach will fill void areas of arbitrary size. The quality of the approximation compared to an ideal all-neighbor IDW implementation depends on the number of path directions n which defines how many support samples on the void contour will be taken for the linear combination. In the worst case of a single valid pixel surrounded completely by void samples, interpolation will occur along one path per image pass only and hence be degenerate. Since nodata values always get interpolated from the last support pixel along a particular scanline, preceding valid samples contouring non-convex voids will be ignored. Compared to window-based IDW techniques which incorporate any valid sample within the support frame, this built-in 2D visibility check helps to suppress intensity edges near the boundaries to void areas that may arise from distant but yet contributing intensity spikes.

During the sweeps, each position of the input image will be visited once per path direction. For a rectangular bitmap of w by h pixels and c color bands, time complexity of the proposed method yields $O(whc)$ for an arbitrarily chosen but constant orientation count n. In practice, executing the algorithm with different direction counts should expose a proportional runtime dependency. On the other hand, concurrently processing scanlines of the same sweep becomes possible without synchronization. Due to the lack of path overlaps, data races are impossible when the incrementally accumulated weights and output intensities are written to the respective bitmaps. Therefore, a speedup linear to the number of worker threads could be ideally expected for a parallelized version of the interpolator.

4. IMPLEMENTATION

As a proof-of-concept, the proposed interpolation algorithm was implemented in compatible C++ to run on both general-purpose central processors (CPUs) and modern graphics cards (GPUs). The prototype takes n-channel TIFF bitmaps comprising samples up to 32-bit floats commonly used to encode digital terrain. Consistently to the input data, the created output and weight images likewise will be internally represented as interleaved 32-bit floating-point matrices. As main parameters, the software can be passed the nodata value, the number of scan directions controlling the degree of IDW approximation and the smoothness exponent. Further command-line options include the angular offset of the path set along which the interpolation is to be conducted, settings on oversampling compensation and postprocessing, and hardware-related information like the thread count and GPU configuration. When the tool successfully finishes, it will save a TIFF bitmap of equal characteristics as the input.

4.1 CPU implementation

The CPU implementation[1] almost directly reproduces line prototype rasterization, the sweep scheme and intensity normalization. Navigation inside the images and pixel access is accomplished using pointer arithmetic for fast relative positioning. All processing stages utilize multithreading with dynamic scheduling based on OpenMP (OpenMP Architecture Review Board, 2015). This accounts for varying line run lengths during interpolation along the traces and the unknown distribution of nodata pixels that locally induce volatile execution times. Calculation of the Euclidean distance to the last support pixel is performed incrementally for each path orientation. This removes the square root function call from the line run loop at the cost of small round-off errors from repeated additions leaving a single non-elementary operation, i.e., a pow call when the smoothness exponent gets applied to the intensity weights w_{md} of equation 1. Since in practice the difference between raster and Euclidean distances is insignificant, the latter get recycled to obtain the coefficients w_{mo} for oversampling compensation.

4.2 GPU implementation

For the GPU implementation of the sketched algorithm, the loop over the line runs in which the IDW equation gets incrementally evaluated was implemented as an OpenCL kernel (Khronos OpenCL Working Group, 2021). Before the interpolation gets started, the involved images will be buffered in graphics memory as 32-bit floating-point arrays. Also, the kernel is passed the necessary image metadata to access pixels using pointer arithmetic, the IDW smoothness and several precomputed quantities like the sweep start positions, pixel offsets for the major and minor line directions and the Euclidean distance increment for the current path orientation. Unlike the CPU version performing monolithic sweeps on the entire image array, the configured OpenCL kernel gets enqueued to work on blocks of scanlines. This prevents GPU resets triggered on high computational loads and ensures that even extensive interpolation jobs do not block the graphics card from displaying normal content on the attached screens. When processing is finished, the interpolation output will be read back from video into system RAM before it gets concurrently normalized and persistently stored.

[1] source code available for download from `https://github.com/DLR-OS/myInterpolator`

In practice, the GPU-based interpolator mostly will be limited by the available amount of graphics memory which must keep three 32-bit image arrays involved in intensity weighting. To lower the storage requirements, on decent graphics cards, at least the weight bitmap could be reconfigured to store bfloat16 samples as they are already used for machine learning (Kalamkar et al., 2019). This cuts the space needed for the normalization coefficients by half. However, quantization errors due to the lack of precision will accumulate potentially yielding rough interpolation results. Storage requirements can further be reduced to those of single-channel bitmaps when the IDW code gets executed separately for each color band of multispectral images for both the CPU and GPU implementation. Runtime then scales with the channel count, and advantages resulting from data caching cannot materialize. Also, there is a substantial overhead for splitting up the images into color planes and merging them afterwards which altogether will qualify this approach for special applications only.

4.3 Postprocessing

For the CPU and GPU version of the software, a postprocessing filter is optionally available to smoothen sampling artifacts from path-wise IDW approximation. It constitutes a 2D Gaussian of a user-defined kernel size. Convolution for now is performed separately for the horizontal and vertical image direction and parallelized using OpenMP in both implementations due to the small computational overhead for realistic window dimensions compared to interpolation.

5. PERFORMANCE ANALYSIS

Performance analysis of the proposed algorithm focuses on runtime and the degree of approximation of comprehensive inverse distance weighting. There will be no discussion of the method's strengths and weaknesses compared to other interpolation techniques which have already been covered in detail, for instance by (Cățeanu and Ciubotaru, 2020) for the DTM case.

5.1 Test sample generation

For the evaluation of the proposed algorithm, the CPU and GPU branches of the software prototype were run on a set of graylevel and multispectral input images with randomly added non-overlapping nodata areas of varying shapes. The voids were constructed by regularly sampling the perimeter of a unit circle between 3 to 20 times and scaling the vectors originating in the circle center by a positive length value between 10 and 2000. The resulting points were translated, rounded to integers and connected by a 4-cc Bresenham rasterizer to form the outer contour of a potentially concave polygon, however, only convex shapes were kept. Any input image pixels located strictly inside the polygon were set to a nodata value of zero for all color bands yielding the test data for the software. Perfect ground truth against which the interpolated test data was compared to resulted from the weighted linear combination of all valid contour pixels of each nodata polygon utilizing precisely the same formula as for the proof-of-concept implementation. To reduce the runtime of this calculation, per-pixel visibility checks were omitted. However, the convexity of the void polygons guarantees that there will be no systematic bias due to shadowing.

To assess the IDW algorithm on real-world image encodings, a 15k by 15k (225 megapixel) 32-bit floating-point height map of Heligoland/Germany photogrammetrically derived from MACS HALE imagery (Brauchle et al., 2015) was altered as described to keep 202240954 (∼90%), 134843025 (∼60%) and 68280296 (∼30%) of its valid pixels. This simulates a DSM that has been stripped elevated objects to eventually become a DTM. Similarly, 16-bit RGB and panchromatic (PAN) true-ortho mosaics congruent to the perforated DSM got prepared to assess the influence of the pixel data type and channel count. Figure 5 depicts the DSM data as a representative for the input set.

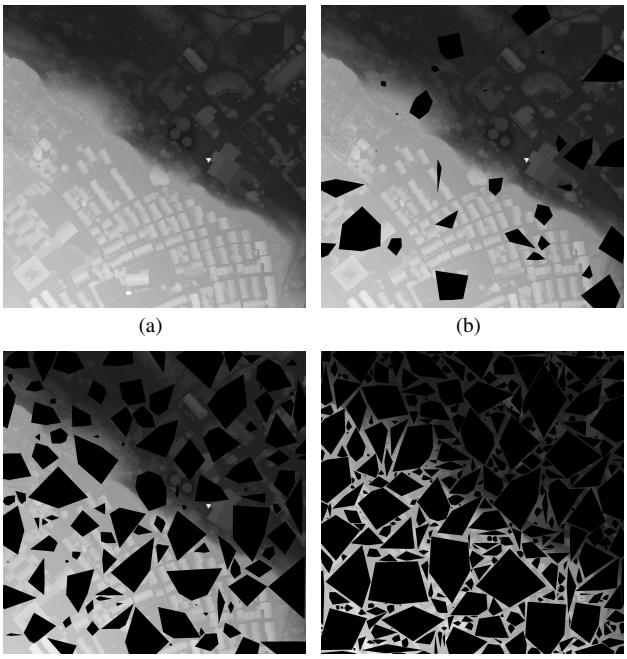

Figure 5. DSM test data (a) full image, (b) 90% valid pixels, (c) 60% valid pixels, (d) 30% valid pixels

5.2 Test results on the CPU

Tests for the CPU implementation of the prototype were conducted on a workstation equipped with an AMD EPYC 7402p 2.8 GHz 24-core general-purpose processor and 256 GiB of DDR4 RAM from 2019 that was running the Windows 10 64-bit operating system. Table 1 summarizes the results for the prepared images for a smoothness exponent $s = 2$ and orientation counts $n = 64$, 256 and 1024 without postprocessing. The values for n have been set empirically to represent low, medium and high-quality approximations of the ideal IDW algorithm. Runtime is in seconds utilizing one thread per physical CPU core excluding I/O. Also, the durations for image initialization and normalization have been stripped from the results since both steps altogether just took milliseconds. Throughput is given in megapixels per second, and approximation quality is obtained from the pixel-wise comparison of the interpolation output to the ground truth as the mean and standard deviation for the set of void areas, i.e., not counting valid image pixels. Calculation of the statistical quantities is performed on the Euclidean distance of the intensity values treated as coordinate tuples in the underlying image color space. For single-channel bitmaps, this is equivalent to the absolute difference between every two samples compared.

Numbers indicate that the proposed IDW algorithm approximates ideal inverse distance weighting fairly well. The mean difference for the DSM is close to its vertical resolution of about

Image (valid %)	n	time s	mpix/s	mean	stddev
Heligoland DSM (90%)	64	36.0	6.257	0.165	0.165
	256	134.7	1.670	0.162	0.163
	1024	559.4	0.402	0.162	0.163
Heligoland DSM (60%)	64	36.0	6.245	0.166	0.199
	256	141.2	1.594	0.163	0.197
	1024	595.1	0.378	0.163	0.197
Heligoland DSM (30%)	64	44.3	5.077	0.157	0.178
	256	174.7	1.288	0.153	0.176
	1024	698.8	0.322	0.152	0.176
Heligoland PAN (90%)	64	34.7	6.493	330.4	384.1
	256	140.9	1.597	277.7	348.7
	1024	555.8	0.405	273.6	346.8
Heligoland PAN (60%)	64	36.3	6.192	311.4	347.6
	256	144.2	1.560	255.1	317.7
	1024	621.9	0.362	250.2	316.7
Heligoland PAN (30%)	64	44.2	5.087	309.4	343.2
	256	183.0	1.229	253.1	316.0
	1024	704.5	0.319	247.8	315.3
Heligoland RGB (90%)	64	38.9	5.780	597.9	658.8
	256	136.6	1.648	503.9	601.6
	1024	573.6	0.392	496.4	598.8
Heligoland RGB (60%)	64	49.8	4.515	555.1	597.4
	256	179.8	1.251	454.8	549.4
	1024	754.7	0.298	445.8	548.0
Heligoland RGB (30%)	64	53.5	4.202	549.4	586.5
	256	216.7	1.038	449.0	542.6
	1024	877.4	0.256	439.3	541.8

Table 1. Runtime, throughput, mean difference and standard deviation to the reference interpolation for n path orientations for the CPU implementation of the proposed algorithm

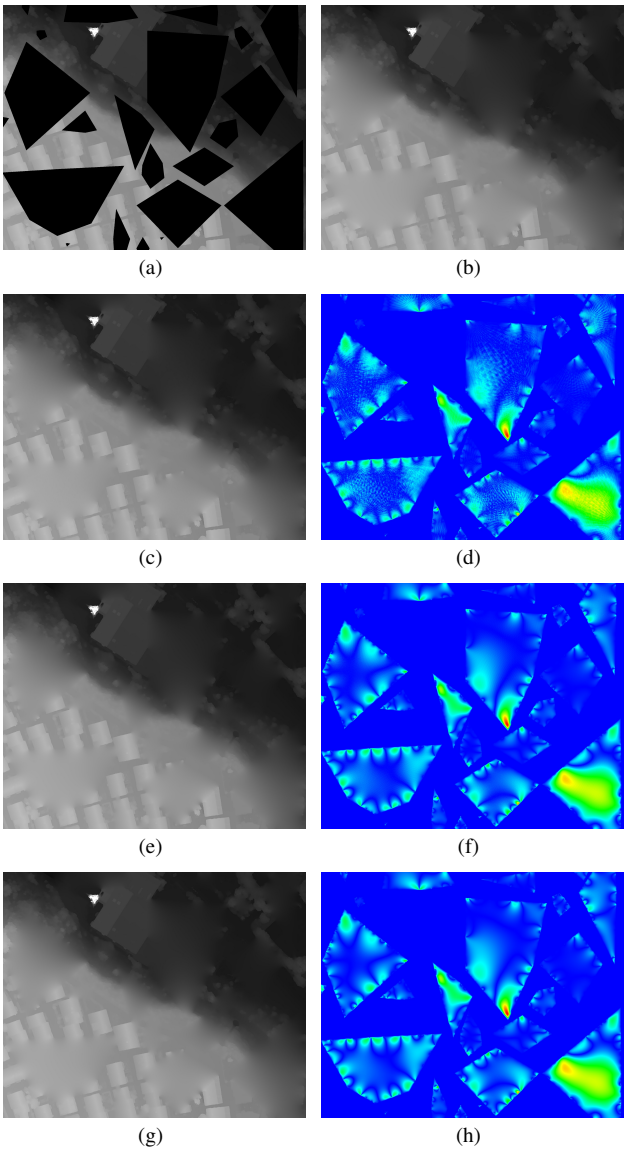

Figure 6. DSM interpolation details (a) prepared image, 60% valid pixels, (b) reference IDW interpolation, (c) interpolation for $n = 64$, (d) heat map for $n = 64$, (e) interpolation for $n = 256$, (f) heat map for $n = 256$, (g) interpolation for $n = 1024$, (h) heat map for $n = 1024$ path orientations

0.1 m, and nearly all interpolated void heights will differ from the average deviation to the ground truth by no more than six times that value. Regarding the true-orthos, the mean difference to the optimal filling is about 0.7% to 1.8% of the effective dynamic range (mean intensity $\pm 3\sigma$) of 15 bits per color channel. For all images, the error decreases when the path direction count goes up, and there seems to be little gain in interpolation quality beyond 256 orientations for the test data. This observation is consistent with a visual inspection of the obtained outputs (figure 6) where no prominent artifacts can be identified at first sight for all three values n. However, the corresponding heat maps which dye the per-pixel differences to the reference according to their magnitude from blue to green, yellow and red reveal streaking due to contour undersampling particularly for the low and medium direction counts. This mostly affects the interior parts of the void polygons since the distance to the support pixels reaches its maximum while path coverage drops. Postprocessing will diffuse the artifacts locally reducing the mean difference and standard deviation only marginally (figure 7). On the other hand, in these regions, the overall divergence to the ground truth diminishes as there is an averaging effect in both the reference and calculated results when the almost equal distance weights approach zero. The red spots in the heat map are situated near local intensity discontinuities in the DSM. In these areas, the vast number of precise samples in the reference data attenuates the influence of close elevated (bright) objects in contrast to the linear oversampling compensation applied by the line-based IDW implementation.

Runtime of the CPU version on the test data grows proportionally by 3.51 to 4.31 as predicted when the number of path orientations quadruples independently of the image type. It increases by up to 59% when the fill ratio of the images drops from 90% to 30%, and going from 90% to 60% valid pixels is less expensive than the switch from 60% to 30% at least for the DSM and PAN data. Also, the slowdown is greater for the RGB than the graylevel bitmaps, particularly when switching from 64 to 256 orientations. Since all input gets converted to floating-point samples, there is only marginal deviations between the single-channel DSM and PAN bitmaps. When comparing the execution speed for the RGB and PAN images, an average de-

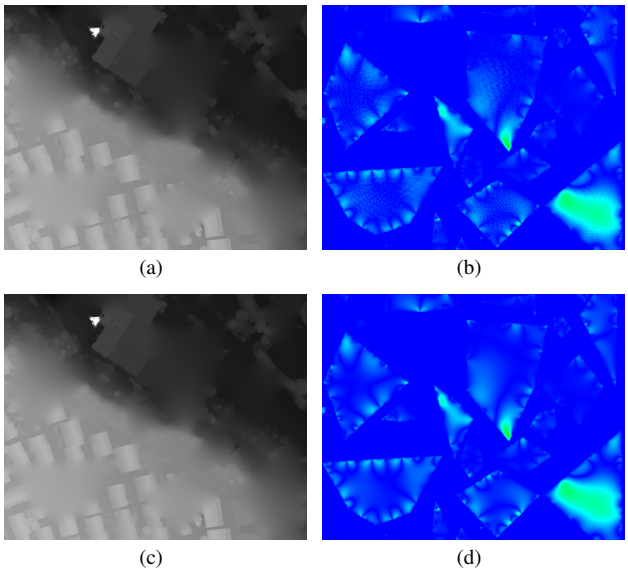

Figure 7. DSM interpolation details after postprocessing with a 31 x 31 Gaussian kernel, 60% valid pixels (a) interpolation for $n = 64$, (b) heat map for $n = 64$, (c) interpolation for $n = 256$, (d) heat map for $n = 256$ path directions

celeration of 18% occurs during the measurements with a peak value of 37% although three times the intensities need to be processed for the multispectral bitmap. The disparity most likely can be attributed to data locality of the interleaved matrix storage. The design choice enables fast inner loops to traverse the color channels and boosts the CPU cache hit rate. It also provides an explanation for the contrasting slowdown behavior on falling fill ratios between the graylevel and color images.

Scalability for the CPU implementation is almost linear with speedups between 1.82 and 1.91 when the OpenMP thread count gets doubled as indicated by table 2 for 256 path orientations on the DSM bitmap with 60% valid pixels. A reproducible outlier occurs for four workers which are only 1.41 times as fast as running two threads concurrently. No explanation can be given for this behavior at the moment. Due to two-way hardware-level parallelization, thread counts greater than the number of physical CPU cores further accelerate program execution on the test system. Minimum runtime of 117 seconds occurs at around 40 threads which is a 17% decrease compared to using 24 workers.

threads	1	2	4	8	16	24
time s	2047	1072	758	411	226	141

threads	28	32	36	40	44	48
time s	134	124	119	117	119	125

Table 2. Scalability of the CPU version of the interpolator for the Heligoland DSM with 60% valid pixels, $n = 256$ path directions

5.3 Test results on the GPU

Measurements for the GPU branch of the software were taken on the DSM and RGB images of Heligoland/Germany using an OpenCL 2.1-capable AMD Radeon RX580 consumer graphics card with 8 GiB of GDDR5 video memory from 2018 that was installed into the EPYC workstation. The amount of available VRAM is just enough to accommodate the 7.54 GiB in total for the multispectral true-ortho, the corresponding weight image and output bitmap. Kernel block size was empirically chosen as 250 million 32-bit float samples not causing any GPU stalls on the test data. This setting means that the graylevel images could be processed as a whole. Table 3 contains the results.

Image (valid %)	n	time s	mpix/s	mean	stddev
Heligoland DSM (90%)	64	10.5	21.504	0.165	0.165
	256	36.2	6.212	0.162	0.163
	1024	139.4	1.614	0.162	0.163
Heligoland DSM (60%)	64	12.1	18.629	0.166	0.199
	256	42.9	5.242	0.163	0.197
	1024	164.5	1.368	0.163	0.197
Heligoland DSM (30%)	64	13.3	16.981	0.157	0.178
	256	47.3	4.760	0.153	0.176
	1024	183.6	1.225	0.152	0.176
Heligoland RGB (90%)	64	16.0	14.104	597.9	658.8
	256	51.4	4.376	503.9	601.6
	1024	193.3	1.164	496.4	598.8
Heligoland RGB (60%)	64	18.9	11.929	555.1	597.4
	256	63.6	3.536	454.9	549.4
	1024	242.3	0.929	445.8	548.0
Heligoland RGB (30%)	64	21.6	10.428	549.4	586.5
	256	73.6	3.058	449.0	542.6
	1024	282.2	0.797	439.3	541.9

Table 3. Runtime, throughput, mean difference and standard deviation to the reference interpolation for n path orientations for the GPU implementation of the proposed algorithm

The numbers indicate that interpolating the DSM on the graphics card is between 2.43 to 4 times faster than on the CPU. Runtime shows slightly sub-linear growth when the number of path directions quadruples. It increases by 15% to 25% as the valid pixel count shrinks from 90% to 60%, however, the slowdown on the transition from a fill ratio of 60% to 30% lies only between 10% and 16%. This indicates that completing smaller voids like they appear in the heavily perforated images better suits the used GPU than the CPU hardware independently of the color bands. For the RGB TOM, there is a speed penalty of 1.39 to 1.62 in contrast to the single-channel DSM. Interpolation quality in terms of the mean difference and standard deviation for the void areas is on par with the CPU implementation showing only insignificant fluctuations. This is probably caused by rounding discrepancies between the GPU and central processor and the native power function used as an optimization in the OpenCL kernel code.

6. CONCLUSION

This paper has described a deterministic approximation of inverse distance weighting interpolation that can fill void areas of arbitrary size in single- and multichannel images. Aside from a temporary bitmap and the output image, the proposed algorithm does not require any auxiliary data structures dissimilar to the input. Interpolation quality and speed are controlled via two key parameters, that is, the path orientation count as the sampling rate on the boundaries of nodata areas and a smoothness exponent. Multithreaded proof-of-concept implementations for both the CPU and GPU have been discussed and evaluated on data sets that are representative for remote sensing.

As future work, to further simplify the use of the presented solution, the choice for the orientation count is to be automatically estimated from the maximum size of void patches detected during the first few passes over the input image. The number of directions then could be dynamically adjusted at runtime according to the outcome. Also, memory consumption of the algorithm is to be optimized particularly for its use on GPUs. For this purpose, the weight image involved in the calculation needs to be switched to 16-bit floating-point numbers or similar compact representations. However, to be efficient, this will require native hardware and software support which currently is not widely available.

In the long term, to process terapixel bitmaps, an adaptation of the existing implementation to an out-of-core image subsystem with block or row paging is to be evaluated. Possible solutions to convert the algorithm to an external memory architecture comprise tiling with overlaps and scheduling the concurrent path traversal either breadth-first or depth-first depending on the scanline orientation. Specifically for DSM interpolation, when the height map is not to be converted to a DTM, the IDW method is to be analyzed on whether it also can perform "from lowest" interpolation, i.e., prefer the dark pixels around a no-data area. This would require knowledge about the intensities on the void boundaries to be present during incremental interpolation. It needs to be evaluated how this information can be obtained preferably without affecting the storage requirements unfavorably.

REFERENCES

Brauchle, J., Hein, D., Berger, R., 2015. Detailed and highly accurate 3D models of high mountain areas by the MACS-Himalaya aerial camera platform. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-7/W3, 1129–1136.

Bresenham, J. E., 1965. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1), 25–30.

Cățeanu, M., Ciubotaru, A., 2020. Accuracy of ground surface interpolation from airborne laser scanning (ALS) data in dense forest cover. *ISPRS International Journal of Geo-Information*, 9(4).

Chen, Y., Haifeng, H., 2019. An improved method for semantic image inpainting with GANs: Progressive inpainting. *Neural Processing Letters*, 49(3), 1355–1367.

Došilović, F. K., Brčić, M., Hlupić, N., 2018. Explainable artificial intelligence: A survey. *41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 210–215.

Elharrouss, O., Almaadeed, N., Al-Maadeed, S., Akbari, Y., 2019. Image inpainting: A review. *Neural Processing Letters*, 51(2), 2007–2028.

Franke, R. H., 1984. Scattered data interpolation using thin plate splines with tension. Technical Report NPS-53-58-0005, Naval Postgraduate School, Monterey, California, USA.

Frommholz, D., 2020. Lock-free multithreaded semi-global matching with an arbitrary number of path directions. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, V-2-2020, 143–150.

Kalamkar, D. D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S. et al., 2019. A study of BFLOAT16 for deep learning training. *Computing Research Repository*, abs/1905.12322.

Khronos OpenCL Working Group, 2021. The OpenCL specification. The Khronos Group, Inc., Beaverton, Oregon, USA. https://www.khronos.org/registry/OpenCL (4 December 2021).

Kotulak, K., Froń, A., Krankowski, A., Pulido, G. O., Henrandez-Pajares, M., 2017. Sibsonian and non-Sibsonian natural neighbour interpolation of the total electron content value. *Acta Geophysica*, 65, 1–16.

Krige, D. G., 1951. A statistical approach to some mine valuation and allied problems on the Witwatersrand. Master's thesis, University of the Witwatersrand, Johannesburg, South Africa.

Liu, G., Reda, F. A., Shih, K. J., Wang, T., Tao, A., Catanzaro, B., 2018. Image inpainting for irregular holes using partial convolutions. *Computer Vision – ECCV 2018*, Springer International Publishing, 89–105.

OpenMP Architecture Review Board, 2015. OpenMP application programming interface version 4.5. https://www.openmp.org/specifications (12 December 2019).

Shepard, D., 1968. A two-dimensional interpolation function for irregularly-spaced data. *Proceedings of the 1968 23rd ACM National Conference*, ACM '68, Association for Computing Machinery, New York City, New York, USA, 517–524.

Sibson, R., 1981. A brief description of natural neighbor interpolation. *Interpolating multivariate data*, John Wiley & Sons, New York City, New York, USA, 21–36.

Stephenson, P., Litow, B. E., 2001. Making the DDA run: two-dimensional ray traversal using runs and runs of runs. *24th Australian Computer Science Conference (ACSC)*, 177–183.