Documentation: Visualize My Picture

Bauhaus-Universität Weimar
Prof. Dr. Charles Wüthrich
Banafsheh Azari

Phil Jungschlaeger
Artur Solomonik
Muhammad Muksitur Rahman

1 Introduction

This documentation covers the work of the students project "Visualize my Picture"[WS17/18], supervised by Charles Wüthrich and Banafsheh Azari at the Computer Graphics department. We explored the possibilities of sampling pictures not with a grid pixel distribution but with a random pixel distribution. Our goal is, to somehow prove that we gain a significantly better image quality by arranging pixels of a photo sensor not in a grid, but random. To do so, we want to compare the outputs of both opportunities.
The idea of sampling images random has its origin in the distribution of photo-sensitive crystals on analogue camera film. The common grid pixel distribution has bad characteristics in sampling certain frequencies as perceived in the Moiré-effect.
We simulated the process of sampling images randomly by re-sampling big grid pixel distribution reference images. We compared three sample distributions, the common grid distribution, a simple random distribution and a Monte Carlo method using the Halton sequence. In order to reproduce samplings, we defined a simple parsable text-file-format.
For the time being, there is no screen to display the samples with a random distribution; therefore, we came up with a selection of interpolation techniques to view our randomly sampled pictures on a common grid display. We worked with numerous techniques from several fundamental algorithms like Delaunay triangulation and Splatting up to a number of more in-depth implementations like our so-called proximity interpolation. Consequently, we evaluated our results using a naive error measuring approach, the complex-wavelet-structural-similarity-index(cw-ssim) and our eyes.
Furthermore we started sampling videos. We compared using a constant random pattern to always generating a new random pattern for each frame.
We prototyped a first random display. A micro-controller draws our random samples on a oscilloscope, which is capable of drawing a pixel at more or less any position.

2 Research

In order to get a first impression on the main issue of our work, we worked ourselves through numerous papers concerning random sampling, image interpolation and quality assessment. Moreover, our first topics to begin with were random ray tracing, Delaunay triangulation and salt-and-pepper noise removal.
We found out that sending out rays in random pattern occasionally perform better than common raster approaches. The fact that the human prefers noisy images over aliased ones is an important fact that is fundamental for numerous methods that we used throughout the project. The trade-off between using noise in order to eliminate aliasing is base of random ray tracing; moreover, such randomness comes in many ways. Rendering a scene randomly might result in point clusters that will effect the ray tracing negatively; therefore, techniques like jitter sampling provide us with a more balanced distribution of random image points. The idea is to move points of a raster by a specific offset in x- and y-direction. As a result, we achieve a distribution in which the samples can't possibly form unwanted clustering. That approach was patented by Pixar over many years. [1]

3 Techniques

This section explains the different sampling approaches, possible interpretations and the problem of quality assessment.

3.1 Sampling

Common cameras have a grid/raster light sensor. Reading, writing and viewing of raster-pictures is really easy and fast. It lays in computer scientist's nature to keep things in a matrix. Software and hardware is designed to work that way. But a raster might not be the best structure to sample and view images. Someone said, looking at a raster image is like looking at the world through prison bars.When you view the raster image in the complex wavelet domain, the influence of it's structure gets immensely visible. You only sample in two directions(up-down and left-right). When you use hexagonal sampling, you add another direction(diagonal). When you look at the complex wavelet interpretation of analogue images, you see a lot better result.
Light sensitive crystals on an analogue film are distributed random. This sample-pattern is not biased to certain directions. In average it samples all directions equally good.
At the moment we do not have such a digital camera sensor with a random pixel distribution. So we simulate the process of sampling, by re-sampling big reference images. These high-resolution reference images represent our real-world. We could not yet answer, how big a reference image is supposed to be. Is a 4K image big enough when you re-sample it with a sample amount of a Full-HD picture? Is the influence of the reference image's raster structure significant in the re-sampled images? Another question is, do we re-sample only the original reference pixels, like integers? Or do we sample in-between the original reference pixels, like floats? We only implemented integer re-sampling, to keep our algorithms easier

[1] https://graphics.pixar.com/library/MultiJitteredSampling/paper.pdf

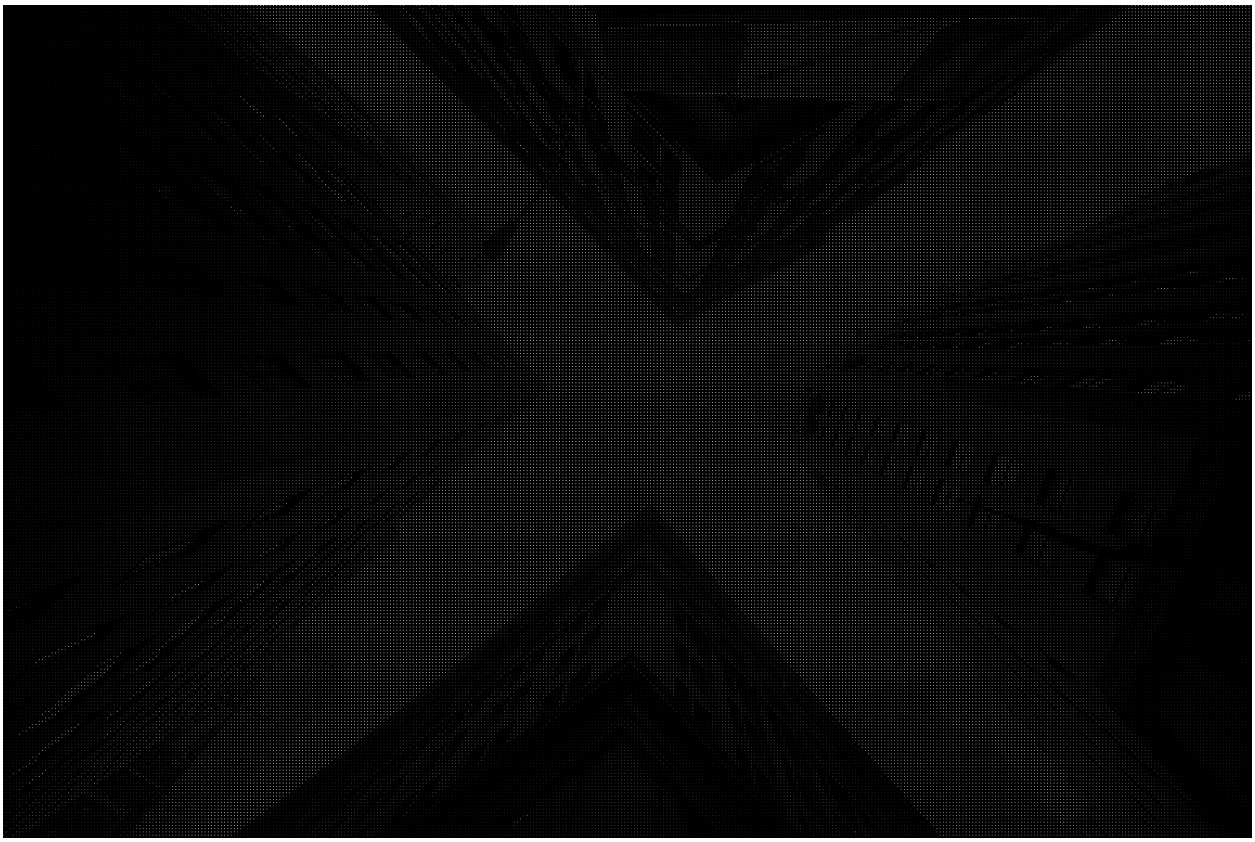
at the start. It is an advantage, that the re-sampled pixels are somehow "real", so we do not need to interpolate between pixels. The integer-randomness is really limited in contrast to float-randomness.
We started implementing our functions in python and c++. For our final comparison, we came up with a small framework to easily loop functions. You can find the different sampling methods in the Sampler-Class [2]

To shrink computational errors in later processing of the re-sampled images, we decided to store the pixel's colour not as integers, but as double values.

When you view the pictures in our documentation, we recommend to view them in a pdf-reader and probably zoomed in, to avoid down-sampling influences. You could also open the Latex document, to find the image references. Than you can view the raw-image from the img-folder, in a proper image-viewer.

3.1.1 Grid Sampling

Re-sampling images with a grid structure is really straight-forward.

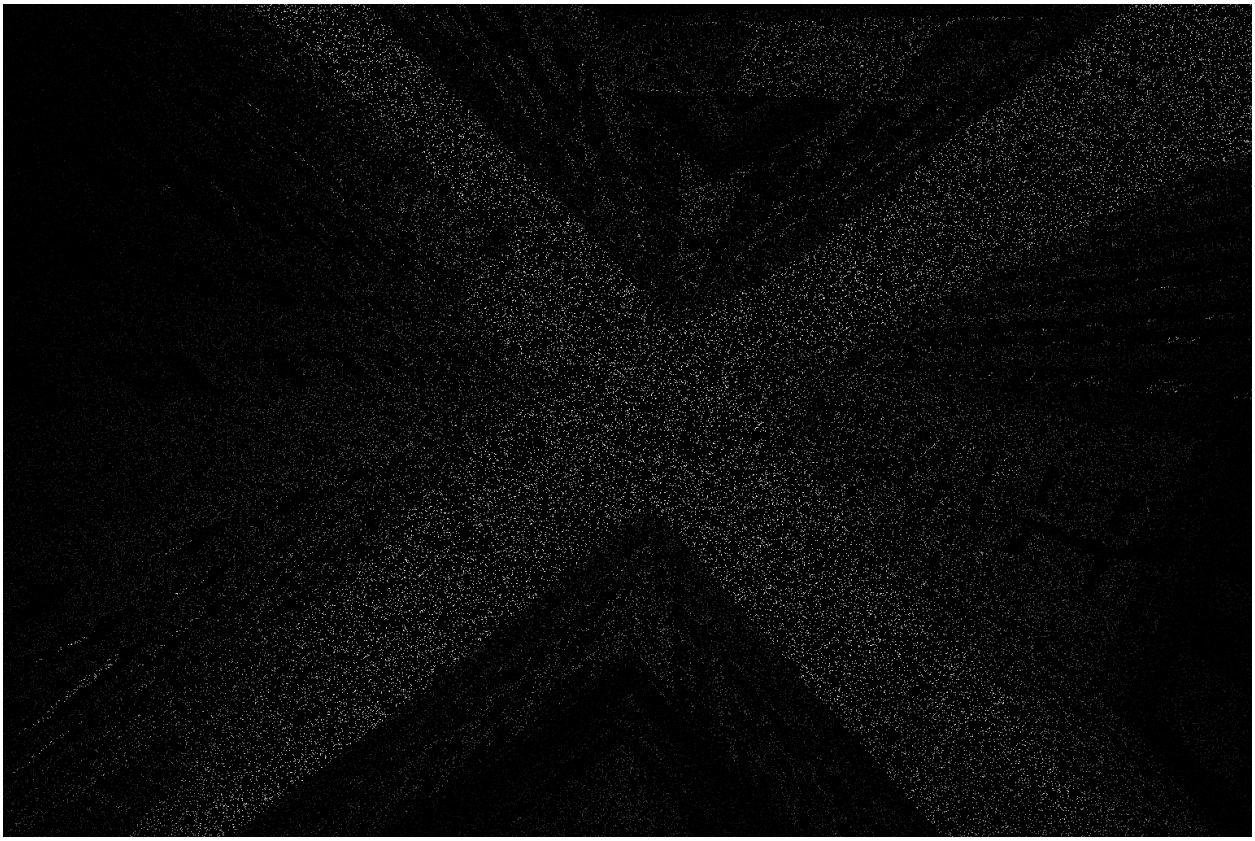


(a) Grid-Sampling: 393216 Samples

[2] github.com/PhilJungschlaeger/rand_pixel_sampling/blob/master/source/sampling/sample.cpp

3.1.2 Random Sampling

At first we implemented a naive random sampling method. In general we simply choose random (x,y)-coordinates with the c++ rand() function. To avoid duplicates and still sample real fast, we use a vector with a small trick:
The random re-sampling method creates a vector with all reference pixels as vector elements. It always picks one vector element randomly. Than it switches the content of the last vector element with the just picked element. Than we delete the last element, and we can pick the next sample.
This makes the re-sampling really fast. We use the speed of the vector. By switching the contents, we do not have to move too many vector elements.

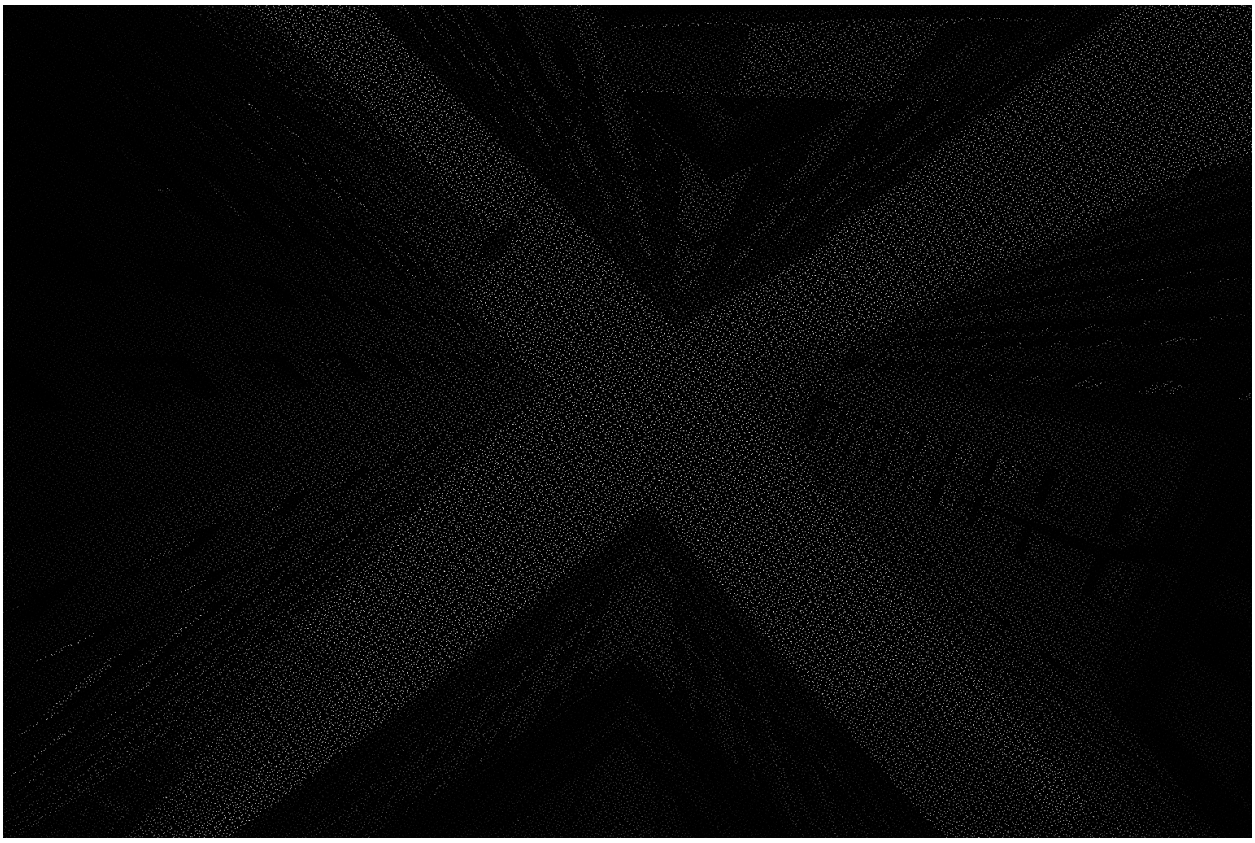


(a) Rand-Sampling: 393216 Samples

3.1.3 Halton Sampling

Our final approach to sample our original images was a Monte Carlo method that makes use of the Halton sequence. Random samples tend to cluster occasionally and lead to problems when it comes to details and surface structures. Therefore we decided to work with an algorithm that promises a quasi-random, well-distributed sampling.
A Halton sequence can be created by taking two numbers with the greatest common divider being 1 and an index determining the number of iterations.

The resulting image points can be rounded to content, hence the implementation could be adapted to either float or integer inputs. Moreover, our base for most of the evaluation was the $(2, 3)$ Halton sequence which lead to images that truly looked like they were randomly sampled.



(a) Halton-Sampling: 393216 Samples

3.2 File-Format

In order to store our re-sampled pixels, we came up with an easily parsable text file format that suited our implementation.
Each line of the document represents a pixel which is defined by its coordinates and the corresponding RGB values; additionally, we separated each value using spaces.

```
x y r g b
```

The first line usually represents the image resolution.
The sampling file of a re-sampled HD Picture with 4 samples might look like this:

```
1280 720
871 76 34 46 50
186 490 173 174 178
98 615 7 13 8
766 89 48 59 63
```

We implemented the RsfWriter[3] (random-sample-file-writer) in order to easily store samples under the pre-defined convention.

3.3 Interpolation

At the moment you only find raster screens, but we somehow want to view our randomly sampled pictures. So we came up with some interpolations to interpret the random data on a raster screen. For later computations, we crafted a Interpreter-Class. All the interpretations can be found there[4].
The interpreter is initialised with the desired output resolution and expects a Pixel-/Sample-Pattern to be set. Furthermore the interpolation is always calculated with double value colours to minimize rounding errors.

3.3.1 No-Interpolation

In order to just see the sampling as it is, we can simply draw them using the sampled colour codes. You can find example pictures in the sampling section. Every pixel that was not sampled will be black. This basic idea turned out to be the fundamental idea of the Splatting algorithm.

3.3.2 Splatting

The Splatting algorithm was an initial technique to be worked with. Given a set of points with the corresponding RGB values, colour a black or transparent canvas using the so-called splats at the sampled coordinates of the image. A splat is a circle with a transparent gradient; accordingly, a splat can vary in size, gradient, colour and transparency.

[3]

[4] github.com/PhilJungschlaeger/rand_pixel_sampling/blob/master/source/interpretation/interpret.cpp

(a) Reference image

(b) 50% image information

(c) 25% image information

(d) 12.5% image information

Figure 4: Image interpolation using with a splat size of 10px

The splats overlap and create a very detailed interpolation; though, there is a significant error concerning black holes resulting from pixels that were not influenced by any splat. We were trying to find some ways to prohibit such black holes.
In order to use this technique efficiently, we had to find relations between such parameters and the image resolution, yet a suitable equation could not be determined. Nevertheless, you can say that a the bigger the splats, the more inaccurate the result. On the other hand, the smaller the samples, the better the interpolated image; even though, the occurrence of said black holes is a lot more frequent.

(a) 15px-sized splats

(b) 20px-sized splats

Figure 5: Varying splat sizes with 25% image information

Our first approach to bypass the black holes was finding a suitable interpolation method to combine it with splats to enforce the high accuracy using smaller splats. Therefore, even more relations have to be determined concerning all the different methods.

3.3.3 Delaunay Triangulation

In order to process a given set of points into a two-dimensional triangle mesh, a common triangulation approach is used. A Delaunay triangulation creates such a mesh enforcing a circumcircle constraint, the Delaunay condition; thus, no other point can be found in said circle except for the three points forming the desired triangle. Moreover, the occurrence of thin silver triangles with two acute angles is minimal, which is a desired property to colour the interpolation appropriately.
As for the colouring, we computed the average RGB values of the processed image points, filling the triangle with said colour.
In combination with out splatting, we, similar to our Voronoi diagram combination, put splats on top of the triangulation itself. Here, on the other hand, we have to try to achieve that every splat has to reach every other splat connecting to each other. In general, the triangle edges of the triangulation should not be visible. As a matter of fact, we want to take advantage of the inner parts of the triangle and ignore its straight surroundings. A formal computation still has to developed to consider not only the distance between one splat to another but also the resolution of the whole image. Such an approach can be facilitated by sampling the image points randomly but in a more controlled distribution.

(a) Reference image

(b) 1% image information

(c) 12.5% image information

(d) 25% image information

Figure 6: Delaunay triangulation with varying sample amounts

After working exclusively in additive colour space, a more perceptional approach including the Lab colour space seemed tempting. The human eye perceives certain colours differently, hence we computed the average of the converted RGB values.
The perceivable result of the interpolation depends on the amount of random samples. While lower resolutions lead to straight edges, higher sample sizes tend to be much more blurry. Moreover, triangulating Raster images did not bear any benefits.

(a) Reference image

(b) Delaunay triangulation using a random sampling

(c) Delaunay triangulation using a Halton sequence

(d) Delaunay triangulation using a grid sampling

Figure 7: Delaunay triangulation using different samplings of 5.5% percent image information

3.3.4 Voronoi Diagrams

A Voronoi diagram divides a given region into multiple smaller regions called Voronoi cells. Such a cell is defined by its centroid and every other point of which the distance to said centroid is smaller than any other. Specifically, we create a diagram that visualizes the maximum impact of one pixel on the resulting image given a set of points.

(a) Reference image

(b) 1% image information

(c) 12.5% image information

(d) 25% image information

Figure 8: Voronoi diagram with varying sample amounts

As soon as we have a Voronoi diagram, we can easily determine the corresponding Delaunay triangulation, hence we can say that a duality exists between the two interpolation methods. Accordingly, a comparison of the two is obligatory.
Our sampled pixels turned out to be the centroids of the individual cells, and their RGB values determined the colouring of the corresponding regions.

(a) Reference image

(b) Voronoi diagram using a random sampling

(c) Voronoi diagram using a Halton sequence

(d) Voronoi diagram using a grid sampling

Figure 9: Voronoi diagram using different samplings of 10% percent image information

In order to enhance our results, we combined the algorithm with our splatting implementation. In that case every splat is put on top of every centroid; although, it has to be noted that certain conditions have to be fulfilled so that there will be any significant changes. The Voronoi diagram itself has a supportive role, filling the black holes that tend to be a side effect of the approach. We put the splats on the diagram itself, so we have to keep in mind that the area of every splat is ought to be bigger than its surrounding cell. Obviously, the splats will not be visible if they don't cross the cell's boundaries. This and the relation to the resolution of the image have to be taken to consideration when determining the correct parameters for the implementation.

3.3.5 Proximity

The basic-idea of the proximity interpolation is to draw all not sampled pixels using the colour data of the k-closest-samples according to their proximity.
At first we simply draw all sampled pixels to the output image. All the other pixels of our output image have to be interpolated. For each of them we have to find the k-closest-samples. To find the k-closest-samples is ambiguous. Sometimes there are two samples with the same distance.

Anyway, we decided to simply sort these according to their colour.
The k-closest-samples will influence the colour of our to be interpolated pixel. Each of them should have an influence.

- The closest sample should have the biggest influence
- The farrest sample should have the smallest influence
- Two samples with the same distance should have the same influence.

We sum up all k colour-values with a factor that represents their proximity. You can use linear, squared or n-th proximity.
We need a reference distance or radius to turn a small proximity value into a big influence value. At first, we used the sum over all k distances. Simply taking the smallest distance of all k distances plus the biggest distance of all k distances is the smallest possible reference distance! That makes the output less blurry:

$$radius = distance_{min} + distance_{max}$$

We can than calculate the pixel colour using this formular:

$$pixel.colour = \frac{\sum_{i=1}^{k}((radius - sample_i.distance)^n \cdot sample_i.colour)}{\sum_{i=1}^{k}((radius - sample_i.distance)^n)}$$

With the fraction we divide the sum by all factors to have the correcting scaling again.
When you choose $k = 1$, proximity-interpolation is just like Voronoi. When n goes to infinity we also tend to get close to Voronoi.
The more influencing points you choose, the more smoothed gets the output.

(a) Proximity: k=1 n=any

(b) Proximity: k=2 n=5

(c) Proximity: k=3 n=5

(d) Proximity: k=4 n=5

(e) Proximity: k=5 n=5

3.3.6 Bucket-System

The search for the k-closest samples takes really long, because you need to compute the distance to all samples for each to be interpolated pixel. To minimize this we made a bucket system:
When we read the samples, we simply store them in buckets like bounding boxes. In other words, we per-sort the samples according to their location. If we had two buckets for an image. We could simply store a sample in the bucket, which represents the left side of the image, or in the other bucket, which represents the right side of the image. With only two buckets you nearly half the work. When we subdivide the image in more buckets, everything gets real fast.
Sorting takes nearly no time. We made this function to find the corresponding bounding

box/bucket for each sample.

```
int xy_to_N(int x, int y)
{
        //_X := image resolution-> width
        //_Y := image resolution-> height
        //_X_Buckets := amount of bucket-subdivisions in X-direction
        //_Y_Buckets := amount of bucket-subdivisions in Y-direction
        int N;
        N=std::floor((double)x*((double)_X_Buckets/((double)_X)))
        +_X_Buckets*std::floor(((double)y*((double)_Y_Buckets/((double)_Y))));
        return N;
}
```

When we have our to be interpolated pixel, we can start searching for the closest-k-samples in the corresponding bucket. Things get a bit difficult, when we need to visit other buckets. We need to know the closest possible point in a bucket to our pixel. It is always a point on the bucket border. In the interpreter[5] is a function, which computes the distance of the closest point in a box n_new to our to be interpolated pixel, with the coordinates x,y from the box n.

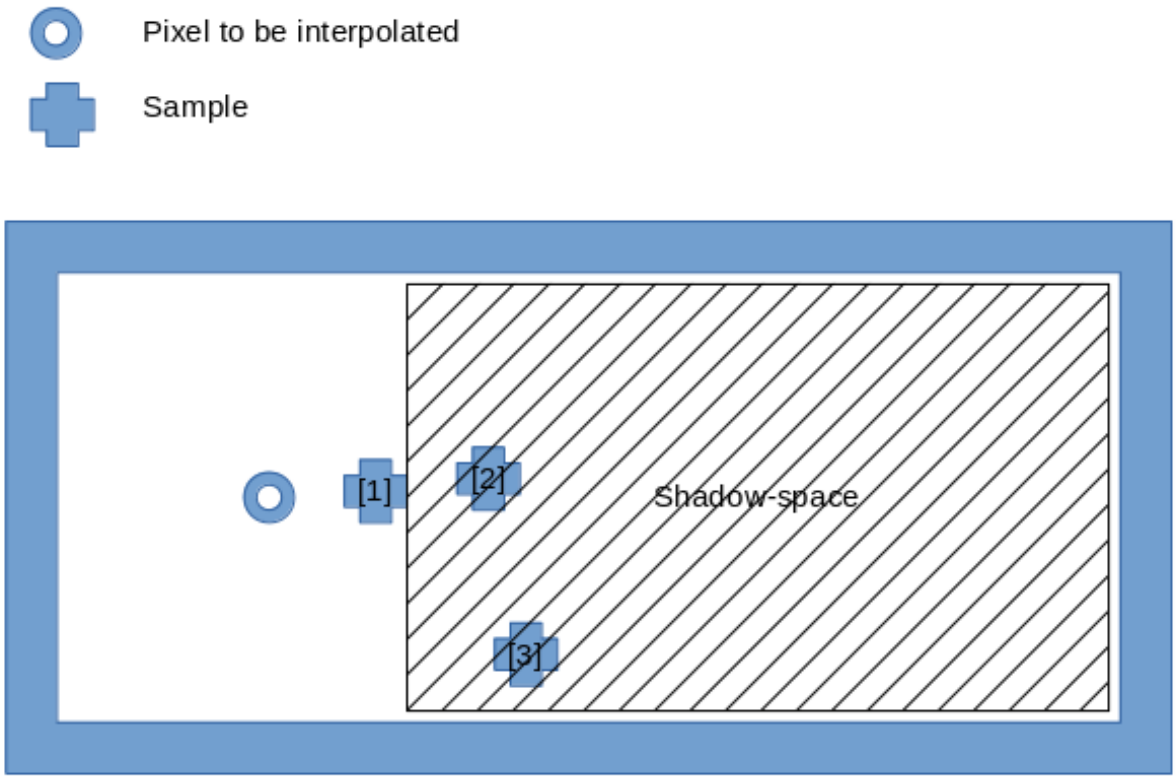
```
double closest_dist(int x, int y, int n, int n_new){...}
```

There are basically 8 cases. When the new bucket is in the same row or column, we simply take the bucket border, that faces to our pixel. This can happen in 4 directions, so 4 cases. When the new bucket is not in row or column with our pixel, we need to calculate the corner point of the bucket, that faces our pixel. At the moment, the bucket system only works with integer dividers of the reference-image-resolution. With a Full-HD reference image(1920x1080), you could subdivide it into 192 columns and 108 rows, so we get 20736 buckets. A certain amount of buckets might be the best to find the k-closest-samples as fast as possible. We don't have a formula yet, to compute the best bucket-amount.

3.3.7 Shadowed-Proximity

To enhance the Proximity-Interpolation, we use the idea of shadowing. When there are to influencing points and the second one is "behind" the first one, the second is somehow in the shadow of the first one.

[5] github.com/PhilJungschlaeger/rand_pixel_sampling/blob/master/source/interpretation/interpret.cpp



(a) Shadow-Space-Sketch

The second sample might only smooth or distort the resulting colour with it's influence. So we delete all the closest samples from our k-closest-samples, that are in a shadow space of a closer sample. We take a vector from the pixel to the sample and construct a line that is orthogonal to that vector and goes through the sample. To determine whether another sample is behind that shadow line can be determined with geometrical-set-theory.

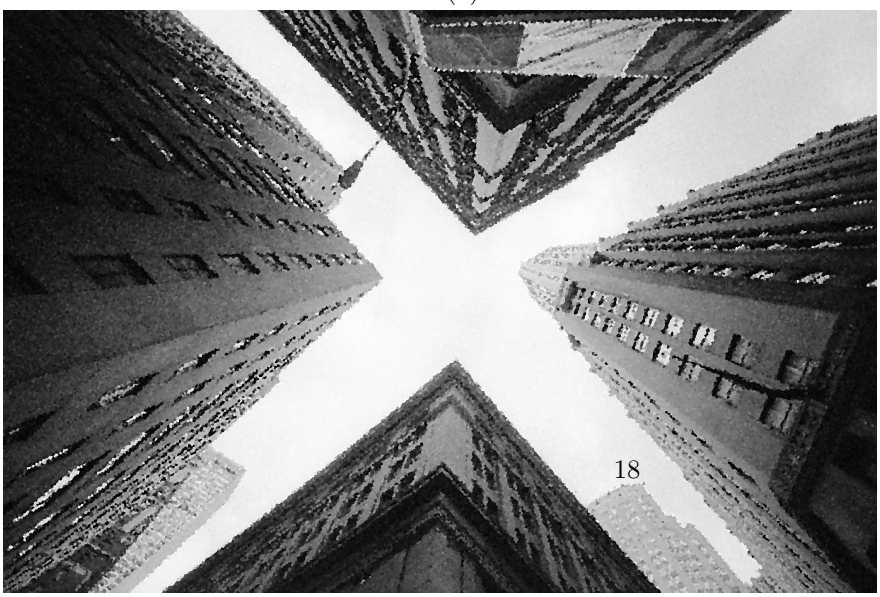
3.3.8 Angle-Proximity

Another enhancement idea, was to let the positional relation of the k-closest-samples to each other have an influence to the computation. So we came up with a directional or angle interpolation.
Around our pixel are 360 degrees. When we have only one sample with influence, it gets all the 360 degrees. Two samples simply split the degrees in half. It starts to get interesting with more than two samples. Imagine 3 samples and two of them were more or less on the same coordinate. With simple proximity interpolation, they all would get their influence according to their proximity. With Angle-Proximity we say, that the two samples, that are next to each other share more or less the same direction and therefore probably also the same information. So we halve their influence.
When our reference image is an gradient, it seems promising to let the angle compute the influence. When we have a detail-/variety-rich reference image, it might be not too promising to let the angle have an influence to the computation. Not every interpolation

fits every reference image?
We used the shadow and angle ideas as stand alone interpolations and in combination.

(a) Voronoi and Grid-Pattern

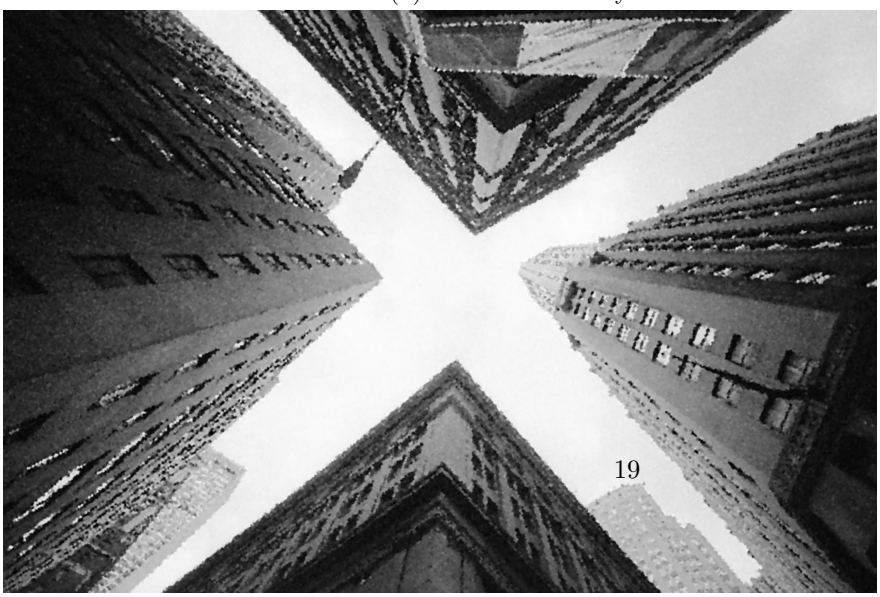
(b) Voronoi and Random-Pattern



(c) Voronoi and Halton-Pattern

(a) Shadow-Proximity and Grid-Pattern

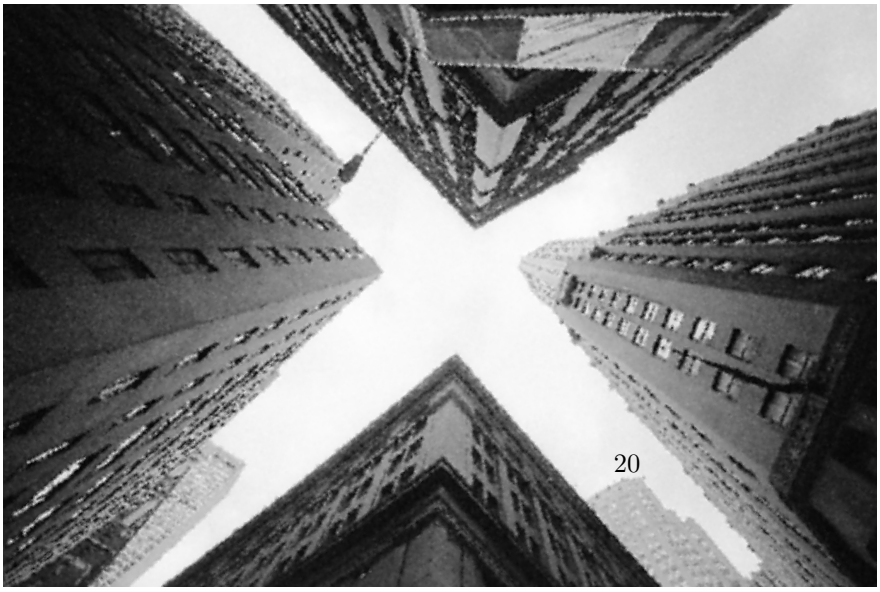
(b) Shadow-Proximity and Random-Pattern



(c) Shadow-Proximity and Halton-Pattern

(a) Area-Proximity and Grid-Pattern

(b) Area-Proximity and Random-Pattern



(c) Area-Proximity and Halton-Pattern

There are some parameters, that could be explored with a bit more dynamic. According to the diversities of influencing colours, we could control the power n of influence or we could decide to take more or less influencing points k. The diversities of distances or colour could also have an influence on n and k.
Instead of using polynomial functions(linear, squared, nth-power) we could think of a better function to compute proximity? How about a Gauß-curve? The shadow space could also look different and therefore have another function.

3.3.9 Other Interpolations

There are also other interpolations we did not use yet. One could imagine to use **neural networks** with pattern recognition to interpolate. We decided to not use this technique, because it might be more like guessing what an image is about, than actually interpolating what is in between two samples.
You can also see an black and white image like a **surface** with values between 0 and 255. Maybe there is mathematical solution to interpolate that surface with random samples using splines or another surface interpolation?

3.4 Evaluation

To evaluate the quality of an image is a surprisingly difficult task. In our case, luckily we have a reference image to compare our output images too(Full-Reference). Even though it is challenging.

3.4.1 The Eye

In general our eye and our perception of an output image can tell us a lot about the quality and the characteristics of an image. To tell that one output image is better than the other is sometimes an arbitrary decision, especially when they look more or less the same.It happens, that one image is good in showing fine details, while the other image is better in representing the correct smooth contours.

3.4.2 Loop-Analysis

When the sample resolution is too high to easily find differences it might be useful to somehow amplify the sample- or interpolation-error. Accordingly, we had the idea of simply looping the procedure.
First of all, we re-sample our reference image with a certain amount of samples and a certain random distribution. We interpolate the samples to get an output image with the same size as our reference image. Then we re-sample this output image with the initial sample-amount and the initial sample-method, but with a new distribution. We interpolate it with the same method and than we take that output image to do it again. When we loop the process of re-sampling and re-interpolating, the overall error increases.
Here is an example run, looping an always new random pattern with 50 percent of the original samples, to compare Voronoi to, simple Proximity, to Shadowed Proximity:

(a) Voronoi: 1 loop

(b) Proximity: 1 loop

(c) Shadowed-Proximity: 1 loop

Figure 15: It is nearly impossible to name one output as the "best"

(a) Voronoi: 15 loops

(b) Proximity: 15 loops

(c) Shadowed-Proximity: 15 loops

Figure 16: While sharp errors appear in Voronoi, the other two interpolations start to blur.

(a) Voronoi: 99 loops

(b) Proximity: 99 loops

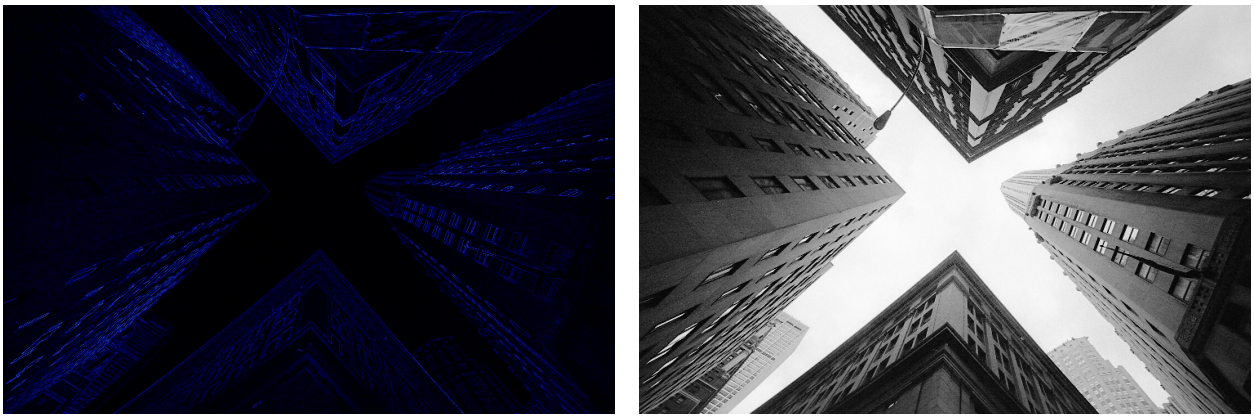
(c) Shadowed-Proximity: 99 loops

Figure 17: Shadowed Proximity seems to uphold most of the original information.

This example loop-run shows that there are completely different errors and characteristics for different interpolation techniques. While Voronoi-Interpolation has really hard errors, Proximity-Interpolation has smooth errors.
For now, this idea of looping the re-sampling and re-interpolating stays an idea. We are not sure whether this loop-technique tells us anything about the actual error of the first loop-cycle. Furthermore we can use this technique only with alternating sample patterns. We can compare different random distributions, but we might not be able to apply it to grid sampling. At least it is quite nice to use the loop-analysis to find the characteristics of an interpolation/sampling method.

3.4.3 Absolute Pixel Error

In order to have a simple analysis and feedback to our interpolated images, we looked at the absolute pixel errors. We simply compare each pixel-colour of the interpolated picture to the equivalent pixel-colour in the reference image. We sum up the errors in red, green and blue to a absolute pixel error. Of course this is not an acceptable quality metric, a picture is more than the sum of it's pixel! But the absolute pixel error is helpful to get an impression were errors occur, by drawing an error map.
This is a Error Map. The whiter, the more wrong:



(a) Error-Map (b) Interpolated Image

Besides counting the overall image error, it is interesting to count how many pixel's are correct or completely wrong. This can mirror the characteristics of an sample-pattern or interpolation technique quite well. The overall image error sum for the just shown error map is 1.42703e+08 and 1003131 Pixel are completely correct. For example Voronoi usually hast the most completely correct pixels in contrast to Delaunay and proximity-approaches, but the overall error sum is a lot higher.

3.4.4 HDR-VDP-2

HDR-VDP-2 is a visual metric determining visibility and quality predictions.[6] Based on a tool developed by the Max-Planck-Institute[7] we evaluated numerous interpolations to see

[6] http://hdrvdp.sourceforge.net/hdrvdp.pdf
[7] http://driiqm.mpi-inf.mpg.de/

exactly which areas are perceived as clearly different from our reference image. A resulting map that was generated by the tool can be a convenient source of information when we want to see where interpolations tend to struggle. Nonetheless, the metric is very strict and lead us to maps that were completely coloured in red. As such, we looked for another metric that can possibly take rather noisy images in account.

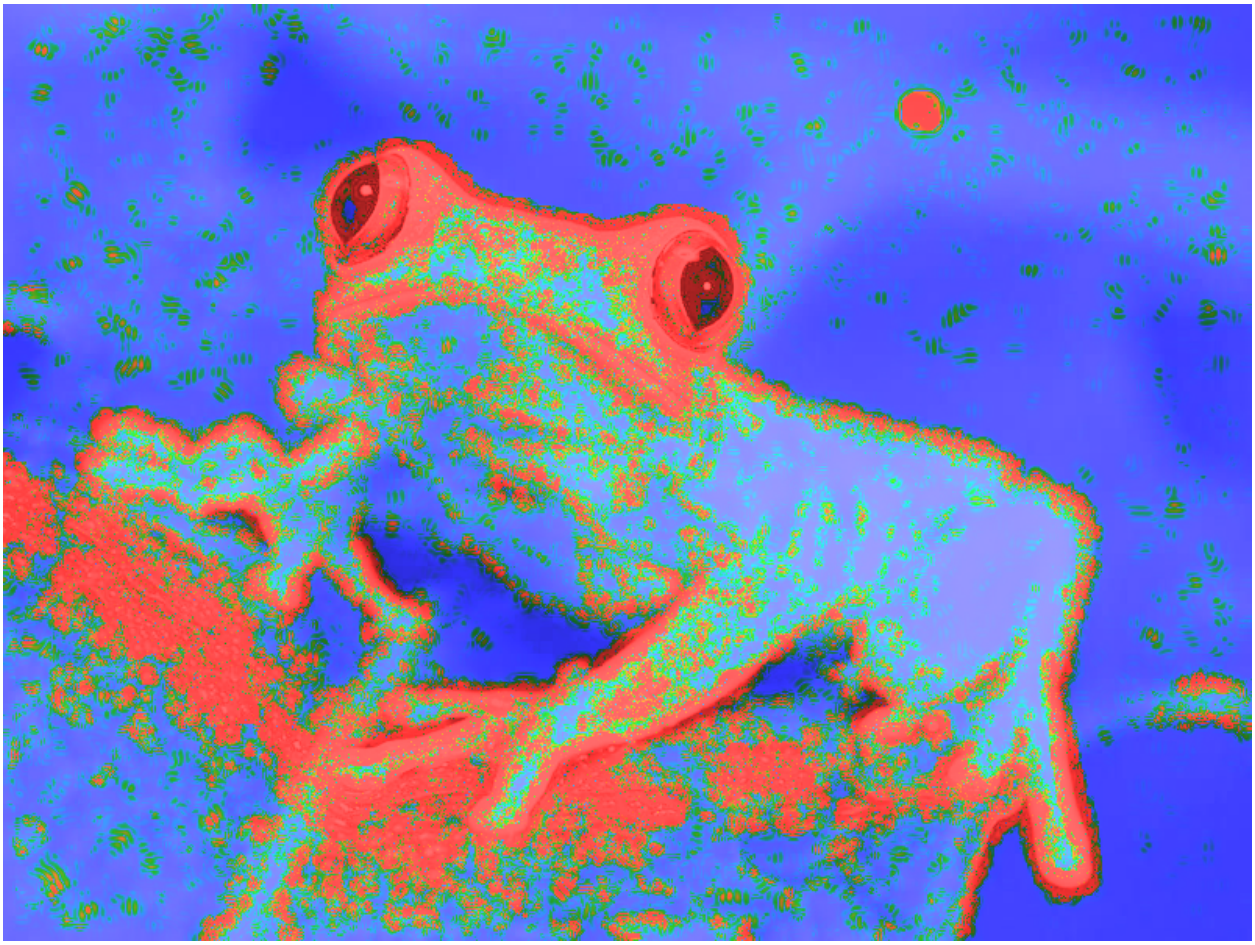
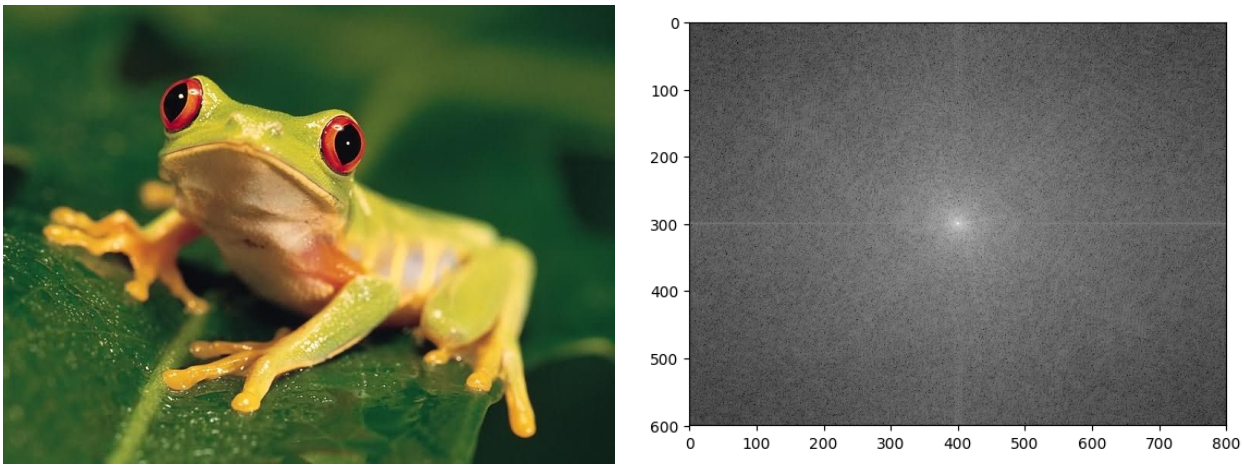


Figure 19: HDR-VDP-2 of a Delaunay triangulation

3.4.5 Fourier Transformation

In order to visually assess the quality of our interpolation we decided to make use of Fourier Transformations and compare the results to our reference. According to possible similarities and preserved features we can tell whether the results are satisfactory or not.



(a) Original image

(b) Corresponding transformation

Figure 20: Fourier transformation

3.4.6 other

There are also other quality-metrics, we did not implement this semester. With the Complex-Wavelet-Analysis,you can calculate the Energy-Efficiency, to see how good the sample distribution is.

4 Implementation

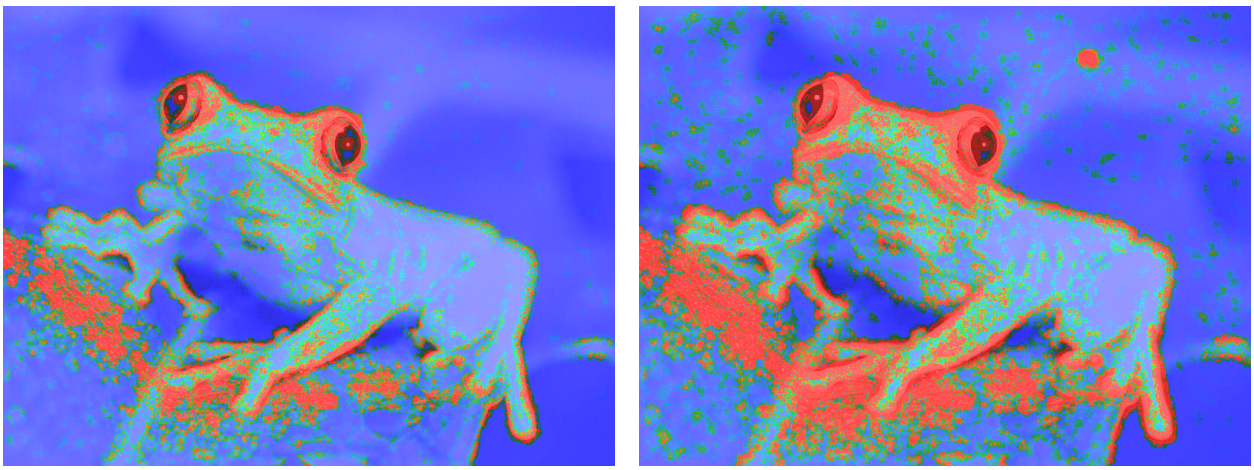
In order to compare the different sample techniques, sample amounts and reference images while using various interpolations, we created a framework [8]. The tool is written in C++ to achieve a faster computation and depends on OpenCV so that we were able to make use of cv2::Mat class and numerous functions including cv2::Subdiv functionalities.
The evaluate_picture.cpp loops the different sample techniques with different sample amounts over different images and interpolates the samples with the different interpolations. It outputs the resulting images and stores some naive error analysis data in a .txt file.
This big loop simply uses the sampler class in the sampling.cpp, the interpreter class of the interpret.cpp and the evaluator class of the evaluate.cpp.
Analogously, we implemented a similar Python framework called DeVoLa [9] with our first implementations of Voronoi interpolation Delaunay triangulation and splatting. On top of that, there is a tool for creating corresponding Fourier transformations and a selection of bash scripts for batch processing. Both frameworks deal with similar text files and generate convenient sampling, evaluation and interpolation images.

[8] https://github.com/PhilJungschlaeger/rand_pixel_sampling
[9] https://github.com/Arduqq/DeVola

5 Comparison

5.1 Perceivable Differences

As soon as we developed our implementation of our two first interpolation methods, a comparison of the two was obligatory two see which benefits are gained from which approach. Our first idea was to assess whether the human eye can perceive possible differences between the reference and the resulting image; therefore, we tried out a perception assessment tool developed by the Max-Planck-Institute. We compared either the image after the Delaunay triangulation and the Voronoi interpolation. As such, the tool created map of a possible image difference prediction using a metric called HDR-VDP-2. The red areas mark spots where the difference is greatly visible determining which interpolation turns out to be more similar to our reference image.



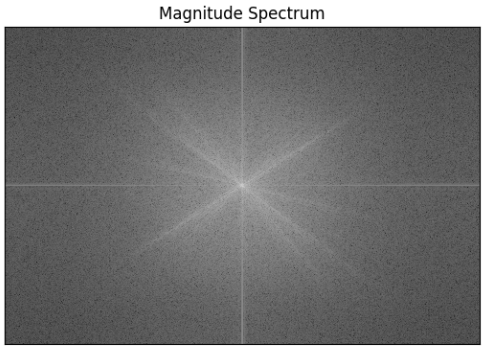
(a) Voronoi diagram HDR-VDP-2 (b) Delaunay triangulation HDR-VDP-2

Figure 21: HDR-VDP-2 comparison between Voronoi diagrams and Delaunay triangulation

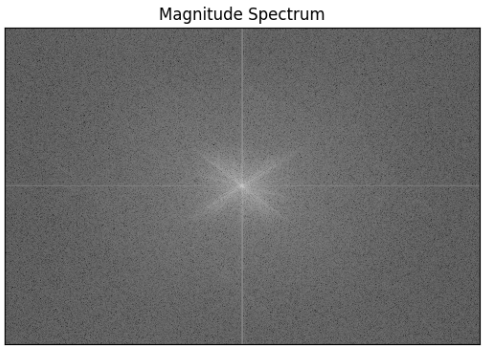
The resulting maps showed that both methods tend to be more problematic when it comes to contrasts and straight edges. Such a behaviour surely is understandable; moreover, we manage to show in that way that especially textures seem to look nearly identical as far as we are looking at the Voronoi interpolation. We can tell that our main issue with such interpolations might be the representation of edges and sharp contrasts turning the splatting algorithm into a valuable addition to our techniques.
Nonetheless, Delaunay triangulation is not obsolete for future research as it might be a valuable method concerning its duality to the Voronoi computation. Moreover, the HDR-VDP-2 prediction tends to overamplify certain errors that are not as bad as they seem to be portrayed as. That lead us to the question whether we want to achieve pixel perfection or a representation of pixels that emulates such a perfection while there are clear errors when you take a much closer look.

5.2 Fourier Transformation

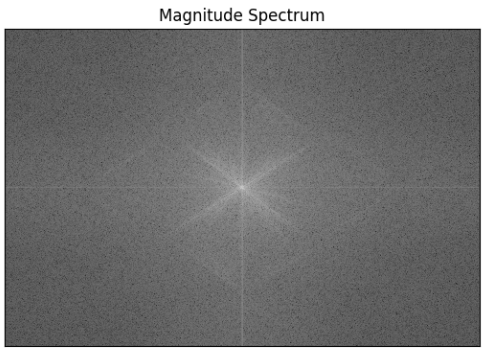
We created a corresponding Fourier transformation in terms of evaluating which sampling method might turn out for the best.
Here we clearly see that a raster image is not always a perfect solution, since there are different features compared to the original Fourier transformations. Random sampling including our Halton sampling on the other hand do have reoccurring lines from our reference; even though, they are not as long as they are in the reference. Furthermore, with lower sample amounts we get better results when using random sampling.



(a) Original image



(b) Voronoi diagram using Halton sampling



(c) Voronoi diagram using grid sampling

Figure 22: Fourier transformation of Voronoi diagrams using different sample techniques

5.3 Splatting Combinations

Certainly, splatting turned out to be a great addition to our implemented algorithms; furthermore, we can take advantage of the accuracy of the approach while filling the black

holes with our existing methods. As such, we had to evaluate on the quality of the different interpolations using reference images that will make every interpolation reach its limits. Our first image had to have a lot of small details that might be overseen by our interpolation. The goal is to have as many details as possible while filling any possible salt-and-pepper noise.

(a) Reference image

(b) Voronoi diagram

(c) Splatted canvas

(d) Combination of Voronoi and splatting

Figure 23: Combining Voronoi diagrams and splatting

We can see that our Voronoi diagram itself does represent the original image quite well but when it comes to the smaller details like the snowflakes and small lens flares we are unable to retrieve the whole information; furthermore, thin lines look inconsistent and aliased. The splatting by itself has a lot of noise when we are using rather small splats and sampling sizes. As a result, we simply splat on top of the diagram and get a new interpolation that seems promising regarding the more detailed representation of the snowflakes and the consistence of thin lines. Not only do we visualize more key informations of the reference image, but we achieve an all-over improvement of our interpolated pictures.
Combining Delaunay triangulations on the other hand seemed to be more troublesome. Simply putting splats on top of the sampled pixels will result in an image where you can clearly identify the exact same pixels. The triangles are coloured according to an average

RGB value; accordingly, the splat's colour stands out compared to its surroundings. A future approach could be putting a splat not on the sampled pixel but on the centroid of the triangle itself. Here, again, we will have to enforce a proper relation between splat size, triangle area and image resolution. With high resolutions and big amounts of samples such a problem is hardly perceivable as seen in the figure below.

(a) Reference image (b) Delaunay triangulation combined with splatting

Figure 24: Combining Delaunay triangulation and splatting

5.4 Post-Processing

In the ongoing process of combining splatting with different interpolation methods we tried to work with basic post-processing techniques; specifically, we tried to smooth, de-noise and median filter the pictures in order to remove the predominant black holes. The issue with such an approach is the loss of information especially when it comes to the median filter. After the process one can barely say which interpolation was actually used and the images themselves don't look too good either.

(a) Splatted canvas

(b) CV2 smoothing

(c) CV2 denoising

(d) CV2 median filtering

Figure 25: Post-processing applied on a splatted canvas

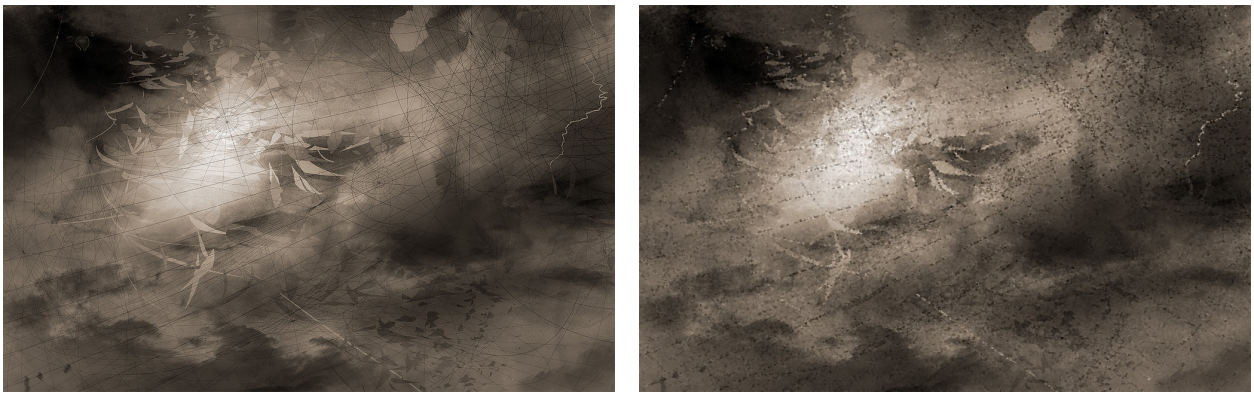
5.5 Source Image Preferences

In order to modify our implementations even further we wanted to work out which image features are the hardest to interpolate with which method. Therefore, we worked with numerous reference images featuring possible deficiencies for our process. We took rather big images that excess in a certain manner. We used the combination of Voronoi diagrams and splatting throughout the comparison.

(a) Reference image (b) Voronoi-splatted interpolation

Figure 26: Interpolation using 50% image information

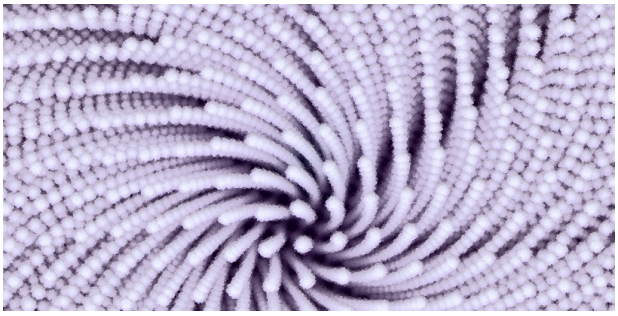
Figure 26 shows that interpolating soft edges and light rays is not a problem. The silhouettes are a common artefact of our interpolations; although, the lines get a lot more consistent when we use splatting on top of our diagram.



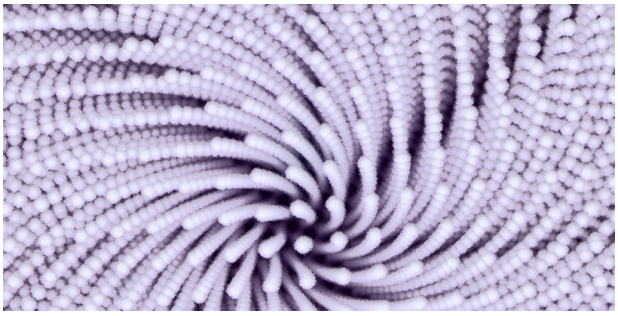
(a) Reference image (b) Voronoi-splatted interpolation

Figure 27: Interpolation using 50% image information

Figure 27 features many thin lines crossing each other. Even though we have very smooth textures in our images, thin lines are really hard to interpolate when we have a small amount of samples. As a result, the image looks very inconsistent and noisy. Nevertheless, most of the features still are preserved.



(a) Voronoi Diagram



(b) Voronoi Diagram with the splatting addition

(c) Voronoi Diagram close-up

(d) Voronoi Diagram with the splatting addition close-up

Figure 28: Adding splats on top of a diagram may result in an overall improvement

Figure 28 shows especially how well splatting effects our Voronoi diagrams. Not only is the texture a lot smoother, but the contours of fibre are really smooth. On top of such interpolations we reach further to a proper relation between splat size and image resolution.

5.6 Final Observations

5.6.1 Interpolation Techniques

After viewing the results we can describe or characterise the different interpolations better. The following 3 pictures are sampled with a halton-pattern of 93750 samples:

(a) Delaunay: err-sum:2.79879e+08, correct-pix: 0

(b) Voronoi: err-sum:2.28788e+08, correct-pix: 589122

(c) Shadowed-Proximity: err-sum:2.04348e+08, correct-pix: 100969

It is probably helpful to view the full folder of results, with different images and different sample amounts, to get a deeper insight.
Delaunay's images look unnecessary blurred in contrast to proximity. The reason might be unweighed smoothing. For interpolation purposes it does not seem to be too useful. Also the error map and the naive error analysis show, that there are nearly no pixels completely correct and the overall error is still quite high. Delaunay somehow has an enjoyable artistic visual appearance.
Voronoi is the common way to show grid images on a screen. That makes it a good interpolation to compare random and grid samples to each other. Voronoi looks mostly really sharp and on spot. It has usually the most completely correct pixel, but also the most really wrong pixel. It has a not to good overall error sum. Voronoi shows all sampled details good, but has problems to show smooth edges and straight lines.
Shadowed-Proximity is the best interpolation of the proximity-family. It reduces the overall error sum better than the other interpolations, but the amount of completely correct pixels could be better. The result looks in general a bit smoothed. The clever smoothing makes edges look more correct, but it tends to loose fine details.

5.6.2 Sampling Techniques

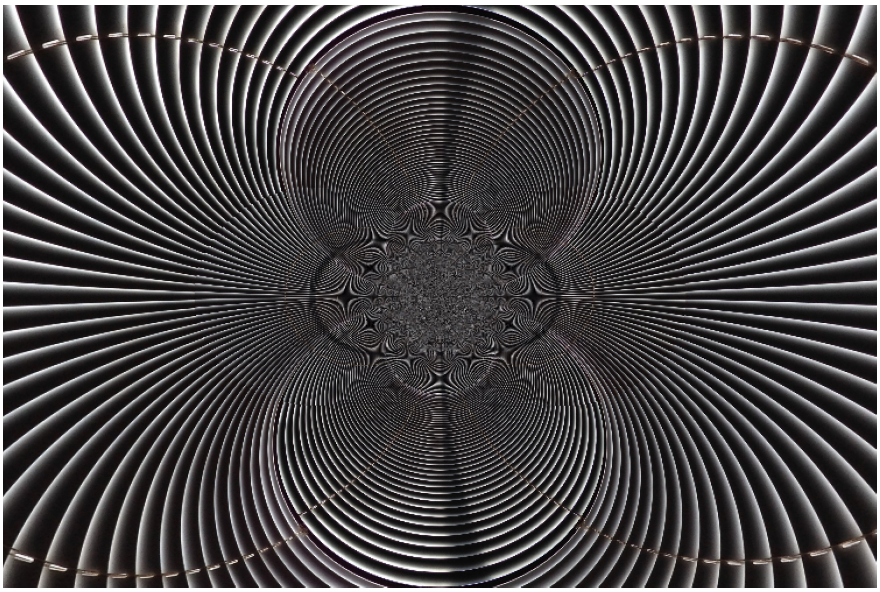
When we compare the naive random sampling to the pseudo random Halton-Sequence, the clustering of samples is reduced with Halton.

(a) Grid-Voronoi: err-sum:1.42703e+08, correct-pix:1003131

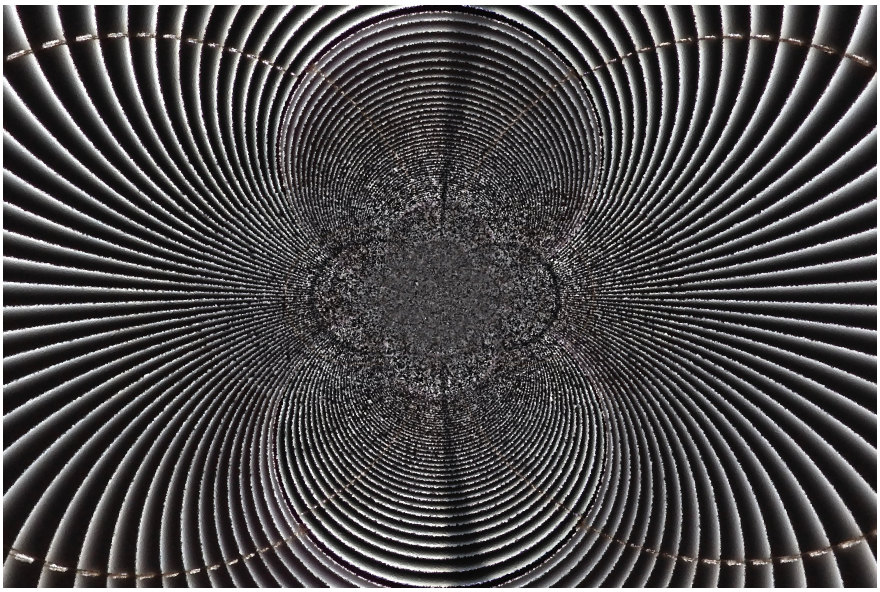
(b) Rand-Voronoi: err-sum:1.59882e+08, correct-pix:961732

(c) Halton-Voronoi: err-sum:1.46286e+08, correct-pix: 987268

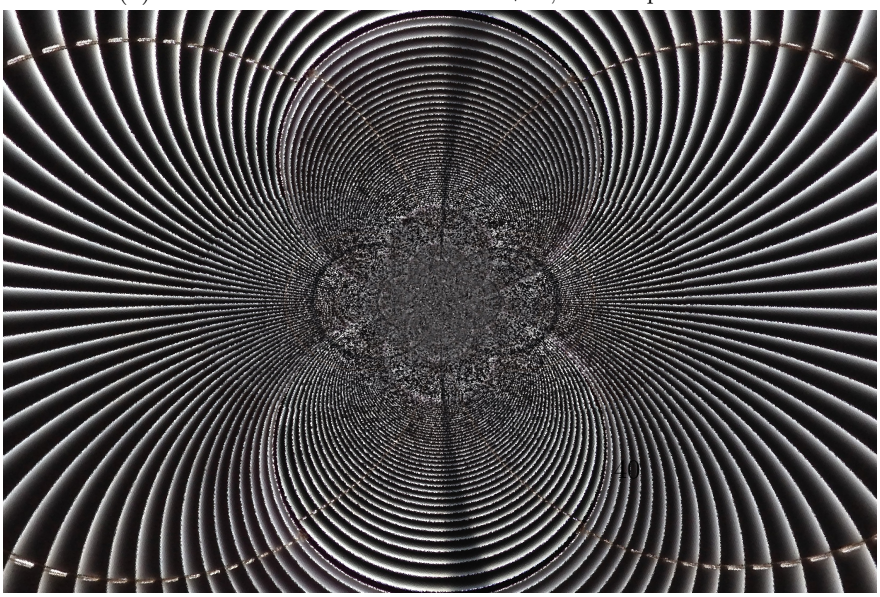
This effects the interpolated images in a positive way, when you look at them with the eye and when you look at the naive error calculations. In general we should look for more well distributed random patterns! We could try to simulate analogue camera crystals and see them like entities with a minimum entity-size. They could distribute themselves on the image and could avoid clustering.
It is really surprising that the grid sampled results look real good, when you look at them with the eye and when you look at the naive error calculations. The grid is probably best in making things understandable or recognizable, because we are used to it. Remarkable is, that we see errors in random- or halton-sampled images earlier than in grid. When you look at the black-white frequency results of halton, you see first errors at the edges really early. On the way to the middle of the image, the noise gets bigger and bigger. The grid-sampled result seems to have nearly no errors in the outer parts of the image, but at some point suddenly aliasing and Moirè happen:



(a) Grid-Voronoi: err-sum:2.85692e+08, correct-pix:820055



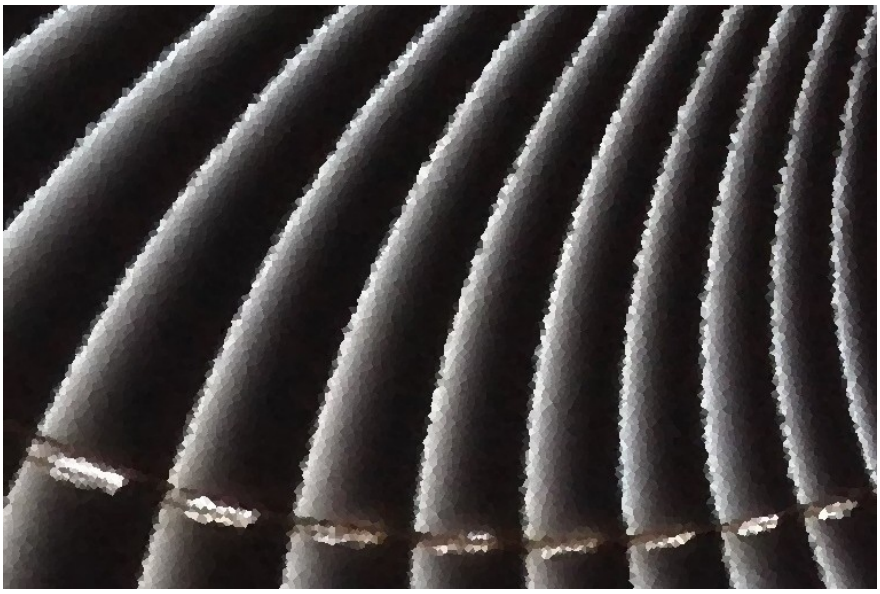
(b) Rand-Voronoi: err-sum:3.35152e+08, correct-pix:771737



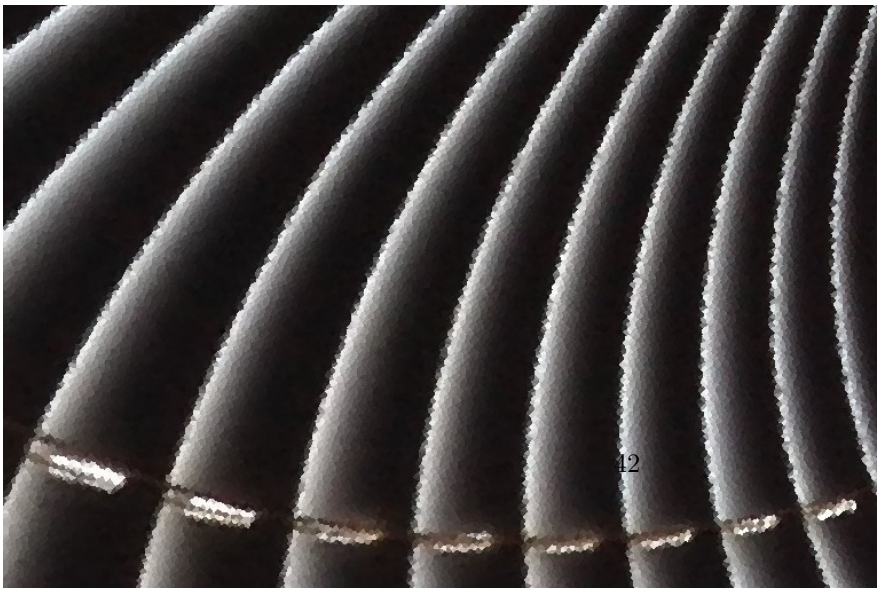
(c) Halton-Voronoi: err-sum:3.05777e+08, correct-pix: 803376

In a way, we might say that a randomly sampled image is more trust-worthy. Whenever there are errors, you see them. On the other hand, one could say, it is better to have a common pattern, so we know where errors appear.
We think, a random noise might be less annoying, than a grid-structured noise. When we look closer, at the images, it's structure gets visible. The random sampled images might look more natural. They remind us of analogue images:

(a) Grid-Voronoi: closer



(b) Rand-Voronoi: closer



(c) Halton-Voronoi: closer

5.7 Different sampling technique plotting on our own first random screen

Figure 33 shows the comparison among 3 different sampling techniques of the same image. These example images are seemingly showing that grid image is best for understanding the image. The inefficiency of Halton and Random pattern plotting might be due to the inefficiency of the plotting device, which is the oscilloscope in this case. There is another significant error pattern during the grid plotting. In grid, the plotting goes from left to right. It seems that, with the increase of pixels, the plotting is faster, thus faster plotting in the right. So in the right, the pixels have less space in between. But in Halton or Random, the pixels are more or less distributed equally, thus the pixels are equally spaced. Other than that, the presence of the pencil in the image is best visualized in grid pattern, a little worse in Halton pattern and the worst in Random pattern.
There is another minor error pattern which is advised to be ignored. At the position, where the beamer stops plotting, seems to be very intense. This explains the extremely bright singular point in an oscilloscope image.
The oscilloscope image also flickers, when the image takes up a wide portion of the screen. This effect is reduced, when the plotting area is reduced. For showing wide screen images, we have used long time exposure camera.

6 Displaying Random Pixels

We had to get familiar with the oscilloscope drawing before getting into our objective. We started with playing with oscilloscope music. Then we used VST board for oscilloscope drawing. Oscilloscope can also be used for vector drawing. The oscilloscope window size is kept 4095x4095. In other words each x or y value is a 12 bit long word. VST board is constructed by Teensy 3.2 which has 64 KBytes RAM, 2 KBytes EEPROM and 256 KBytes Flash memory. The both X and Y channel was calibrated 0.5 volts/div. Figure 34 shows our own VST board.

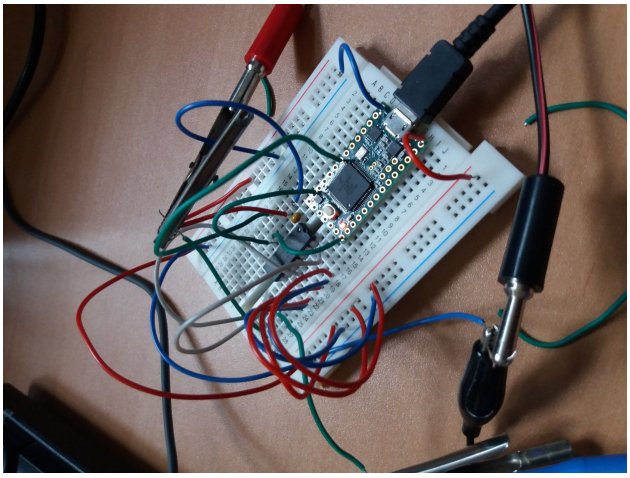
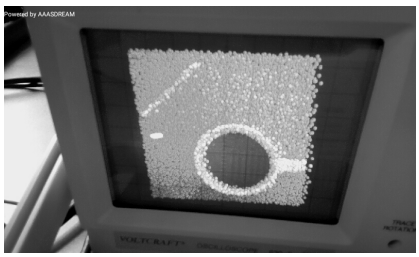


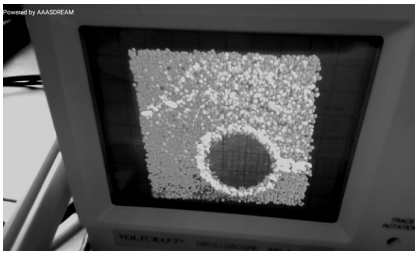
Figure 34: VST board

(a) Reference image

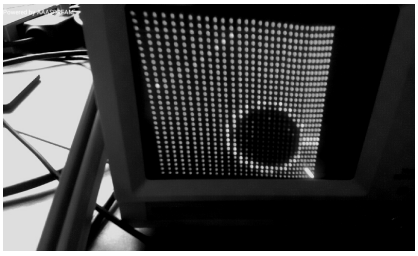
(b) Grid or Raster 8464 pixels, max 32 intensity



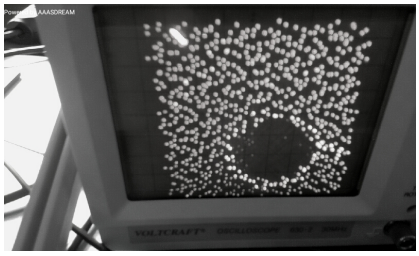
(c) Halton pattern 8464 pixels, max 32 intensity



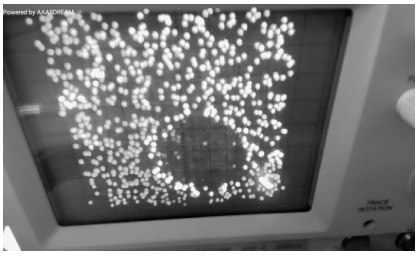
(d) Random 8464 pixels, max 32 intensity



(e) Grid or Raster 900 pixels, max 64 intensity



(f) Halton pattern 900 pixels, max 64 intensity

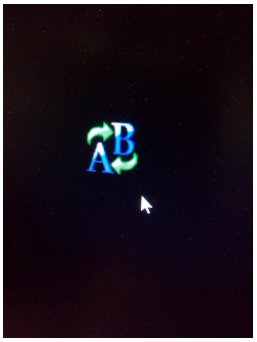


(g) Random 900 pixels, max 64 intensity

Figure 33: Comparison, subject: coffee mug

6.1 Grid Images

Initially we implemented raster image drawing. The challenge was the different intensities of grey-scale. The technique of varying the intensity was plotting the same point varied times. Hence, more times more intensity, less times less intensity. These are the same image in different intensities in figure 35. Since for these grid images, we stored the pixels in the RAM, we kept the resolution 30x30.



(a) Reference image

(b) max 4 intensities

(c) max 8 intensities

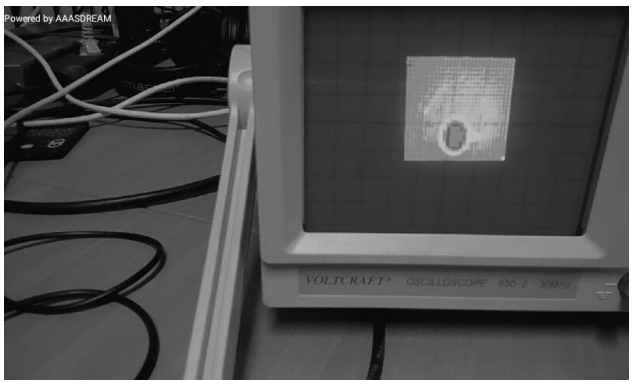
(d) max 16 intensities

(e) max 32 intensities

Figure 35: Grid images in different intensities; 900 pixels each

Two more examples of 30x30 resolution grid images are provided in figures 36 and 37.

(a) Reference image



(b) max 32 intensities, 900 pixels

Figure 36: Grid image: Coffee mug

(a) Reference image

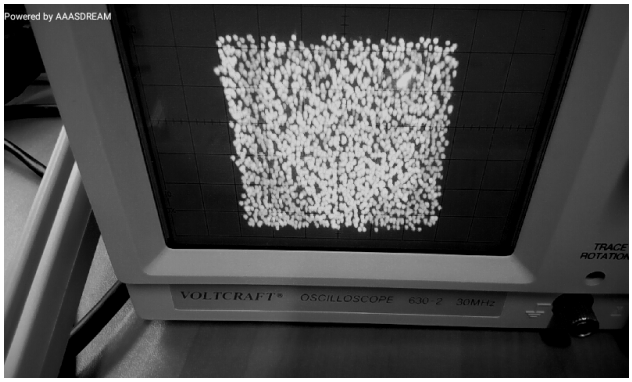
(b) max 32 intensities, 900 pixels

Figure 37: Grid image: Flower

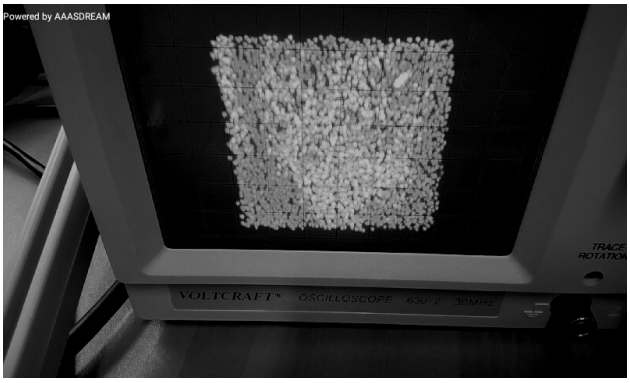
6.2 Random pixel Images

Initially the points were stored in the RAM which is of size 64 KBytes. So, we could not store more than 900 pixels. Then we stored the points in Flash memory of size 256 KBytes, which allowed us to store up to 8800 pixels. This time we did not use grid any more. So the intensities are no longer in a matrix. We took a text file, where each of the random pixels are in following format in each line:-
X Y R G B
X, Y is the coordinate of the pixel and R, G and B are respectively red, green and blue intensity values. RGB was converted to Gray level. Eventually the file is converted into a string of logical points, where each logical point symbolizes a singular plotting of a pixel. Codes and resources associated with oscilloscope drawing in our project are also available in the oscilloscope folder of this document.
Figure 38, 39, 40 and 41 show us how varied number of random pixels look like in oscilloscope.

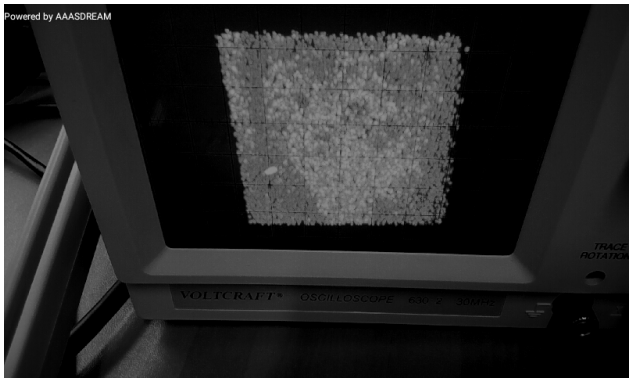
(a) Reference image



(b) Random 2200 pixels, max 32 intensity



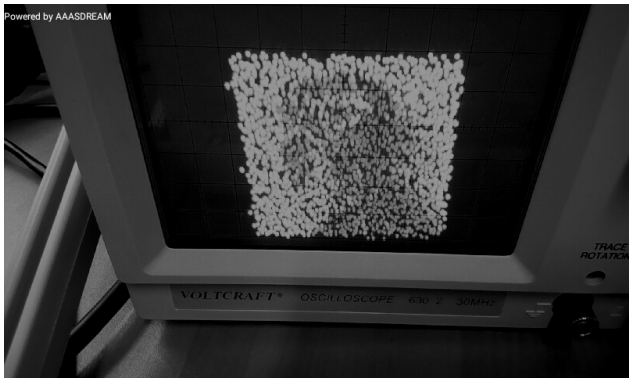
(c) Random 4400 pixels, max 16 intensity



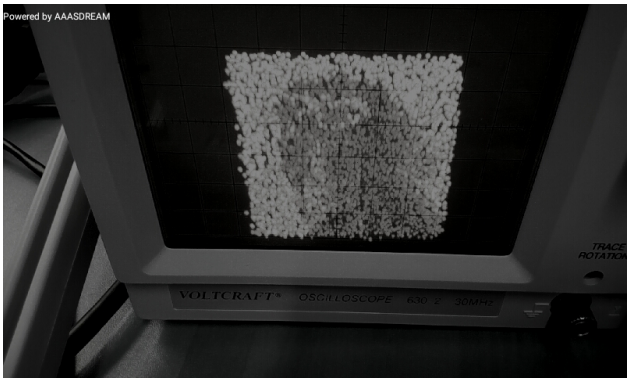
(d) Random 8800 pixels, max 13 intensity

Figure 38: Random pixel image, subject: cat

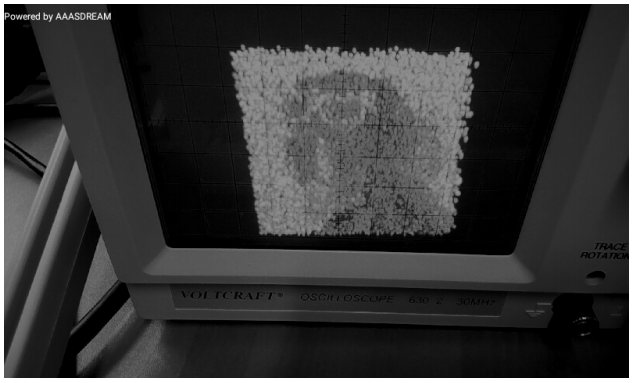
(a) Reference image



(b) Random 2200 pixels, max 32 intensity



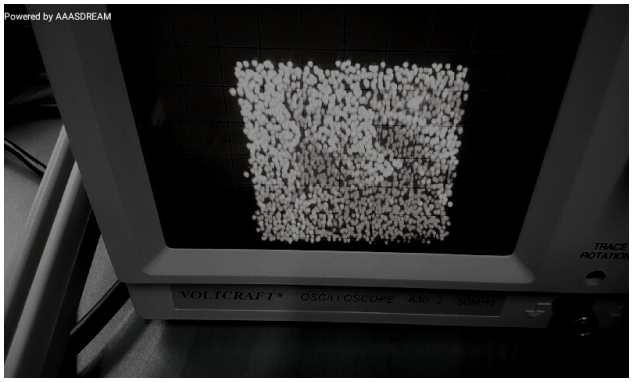
(c) Random 4400 pixels, max 16 intensity



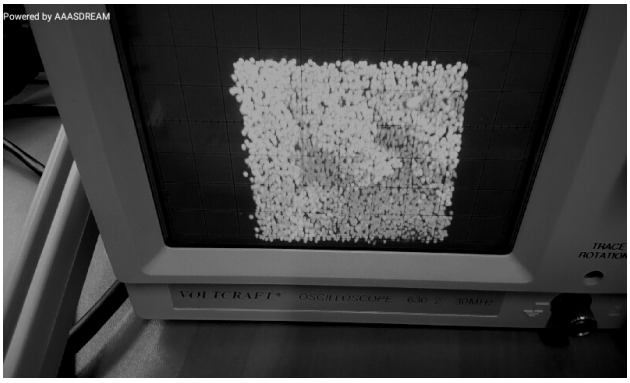
(d) Random 8800 pixels, max 11 intensity

Figure 39: Random pixel image, subject: dog

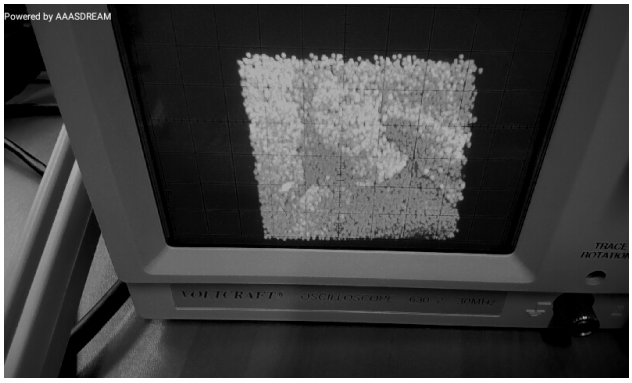
(a) Reference image



(b) Random 2200 pixels, max 32 intensity



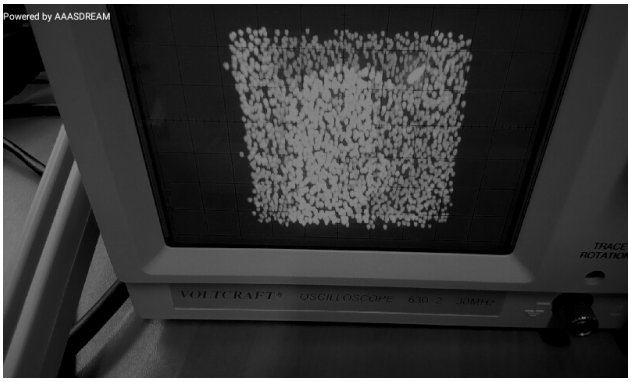
(c) Random 4400 pixels, max 32 intensity



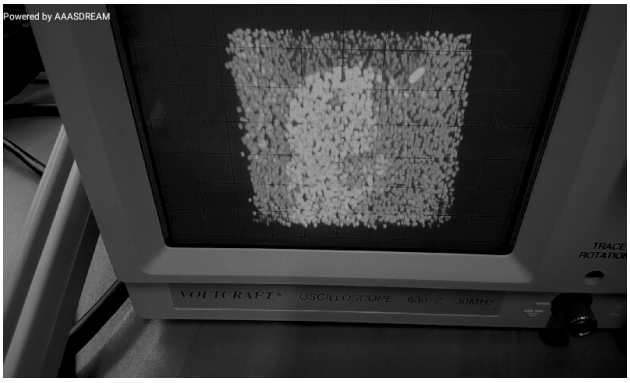
(d) Random 8800 pixels, max 16 intensity

Figure 40: Random pixel image, subject: girl

(a) Reference image



(b) Random 2200 pixels, max 64 intensity



(c) Random 4400 pixels, max 32 intensity

(d) Random 8800 pixels, max 25 intensity

Figure 41: Random pixel image, subject: owl

7 Random Sampled Videos

We started to re-sample videos with random patterns. We simply loop our sampling and interpolation techniques. For each video-frame, we could use the same/a constant random sample pattern.

7.1 Constant Pattern

A constant pattern works fine when you look at the output video of a static scene. When the scene is not static, for example a moving object, one can observe the underlying sample pattern as the object moves through it. When you even have a dynamic camera, you can notice the whole pattern. To suppress this effect, you have to scale up the sample resolution.
In the video-folder are example videos.

7.2 Always New Random Pattern

When each frame gets a new random sample pattern, a overall flickering comes into the video. Charles said, it is like looking at a frying pan. In simple or information-less areas of

the video, the effect is less visible compared to detail-rich areas.
At the moment, the constant video is nicer to look at, but I think a always new random pattern might be more promising later. In a static scene, we always get new information, when we always sample with a new pattern. Because most of the video-picture might not change.
I sampled a video with a always new random pattern and only updated the just sampled pixels. You see, how the picture develops. We achieve a real high resolution in static areas of the video-picture. There are only errors in dynamic regions.

(a) First Frame.

(b) After a few frames

(c) After a few more frames

I re-sampled the video-fames with simply new random patterns. With a clever sequence of tuned sample patterns, we could cover the scene a lot better! If we were able to detect movement in the live-view of the camera, we could focus the resolution of our random patterns to dynamic regions.

7.3 Interpolation In Time

Furthermore we could think about interpolating in time. Just imagine Voronoi not as a cell, but as a volume with the new dimension: time. This might also work with Delaunay and definitely with the proximity-idea. I think the overall resolution would increase a lot!

8 Resume

Our goal was, to somehow prove that we gain a significantly better image quality by arranging pixels of a photo sensor not in a grid, but random. We realised it is actually stubborn, to break things down to a simple word like "quality". It needs a bit of moony openness to look at randomly sampled images. At the moment, everything is grid. Our cameras use grids, our computers use grids, our screens use grids. We are used to grids. After working with randomly sampled images, one starts to prefer rocky-edges to stairway-edges. The random noise looks more natural and less annoying, in contrast to the structured noise. The random pattern is more truthful in showing its errors, while the grid starts aliasing. Grid images win the comparison if we just consider the naive error-analysis(amount of correct pixels, overall error sum). Images tend to look more understandable when they are sampled with a grid-pattern. But for now it is literally more beautiful to break up with the grid. We just started with random sampling, so more performance, more insights might be to come.
Anyway this semesters work shows, it is possible to sample images random. At the moment, the Halton-Sampling seems to be the best random-distribution. The Naive-Random-Sampling lacks because of clustering. We are capable to save random images with our simple .txt file format.
The oscilloscope is our **first random-screen** to show any integer-pixel-distributions. At the moment it is capable of showing maximum 8900 Pixels, black and white, with maximum 32 different intensities. With the help of some interpolation techniques, we are capable to convert any integer-pixel-distribution to a grid-format in order to view it on a common grid screen. The Voronoi-Interpolation is probably the best for simple comparisons between pixel-distributions. The Shadow-Proximity-Interpolation is a first approach to reduce the overall error and the rocky-edges of random-sampled-images.

9 Future Work

Our project shows a lot of points to work at. At the moment everything starts and ends as a grid. Our reference images were sampled by a camera with a grid pattern. We re-sample this images random, but than interpolate them to fit a common screen. Depending on the resolution of our screen, the image-viewer up- or down-samples the output, of course in a grid. To start right at the start, a first random-sample-pattern-camera or -*let's call it*- **any-sample-pattern-camera** would be interesting. It could be slow and just capable of sampling static scenes. The camera could be connected to a processing unit to sample clever. While sampling, the computer could recognise detail-rich areas in order to raise the sample resolution there.
To view the pictures, another **screen**, capable of higher resolutions, would be nice. The comparison of grid-, random- and any-distribution could go further.
Later a storage saving and faster **file-format** could be crafted. We could think of a hexa coding. With pseudo random functions, we could simply use the pattern-seed to not have to store all the positions? We don't think it is necessary to pre sort the samples, because it takes only linear time do it while reading(bucket system).
As for our interpolation techniques and the combinations with the splatting approach a formal analysis on determining a proper relation between splat size, colour values, image resolution and the amount of samples is obligatory. An equation might be effective to make the most out of all of our implementations. The bucket-system needs a mathematical analysis to find the most efficient amount of buckets. Furthermore, a **stable framework** is necessary for future researches and might have to be developed from scratch. The framework should include jittered-grid-sampling and hexagonal-sampling. It might be capable of switching between integer and float randomness, to show it's (dis)advantages. It could be extended by some new (pseudo) random distributions.
Evaluating interpolation and sampling methods empirically can be linked to multiple image quality metrics, and working out a viable method for assessing such images specifically can be an important matter. It might be really useful to make experiments with humans. How do people perceive randomly sampled images. Do they also prefer rocky-edges to staircase-edges?A long term study could be interesting to see whether their perception changes, when they get familiar with random images? The recognisability of image features and objects for a human or a computer might be interesting to analyse.
Randomly sampled videos offer a lot of opportunities. A clever **interpolation in time** could boost frame rate and resolution. One could think about using Canon's Cameras and Magic Lantern to edit the video sample pattern. We could read randomly $\frac{1}{8}$ of a 4k image-sensor's pixel. This might allow a 8 times higher frame-rate. With a clever pattern generator, we could interpolate the random-image-data over time to a 4k output video.
There are also some **general questions**. How does "sampling in more than 2 directions" affect the humans perception of an image? To understand an image better, it might be better to sample 2 directions in high resolution instead of spreading the sampling amount in more directions? Do we prefer a sharp detail rich image to a smooth well-contour-representing image?