Oculean

OCL-001 / Optical codes

How much of a QR code can be given to an image before it stops scanning

Author
Stone Baldwin

Published
2026-11-28

Version
0.1

Type
Paper

Program
Optical codes

Data and code
Not yet public

This page, v3-M

Abstract

A halftone QR code replaces most of each module with a picture and keeps only a small dot in the middle to carry the data. This study measures how small that dot can get. In simulation, 360,000 decode attempts covered four error-correction levels, ten dot sizes, six source images, five capture conditions and three open-source decoders. Scan rates did not decline gradually. They fell from about 100% to near zero within a few steps of dot size, and where that cliff sat depended far more on the decoder and on capture quality than on the error-correction level. With a pristine image one decoder read codes that were 97% picture, while another failed past 44%. Under moderate simulated blur and noise the best decoder was reliable up to 66% to 75% picture. Raising error correction from L to H moved the cliff by at most two dot steps. All results here are simulated; no printed codes or phone cameras were tested.

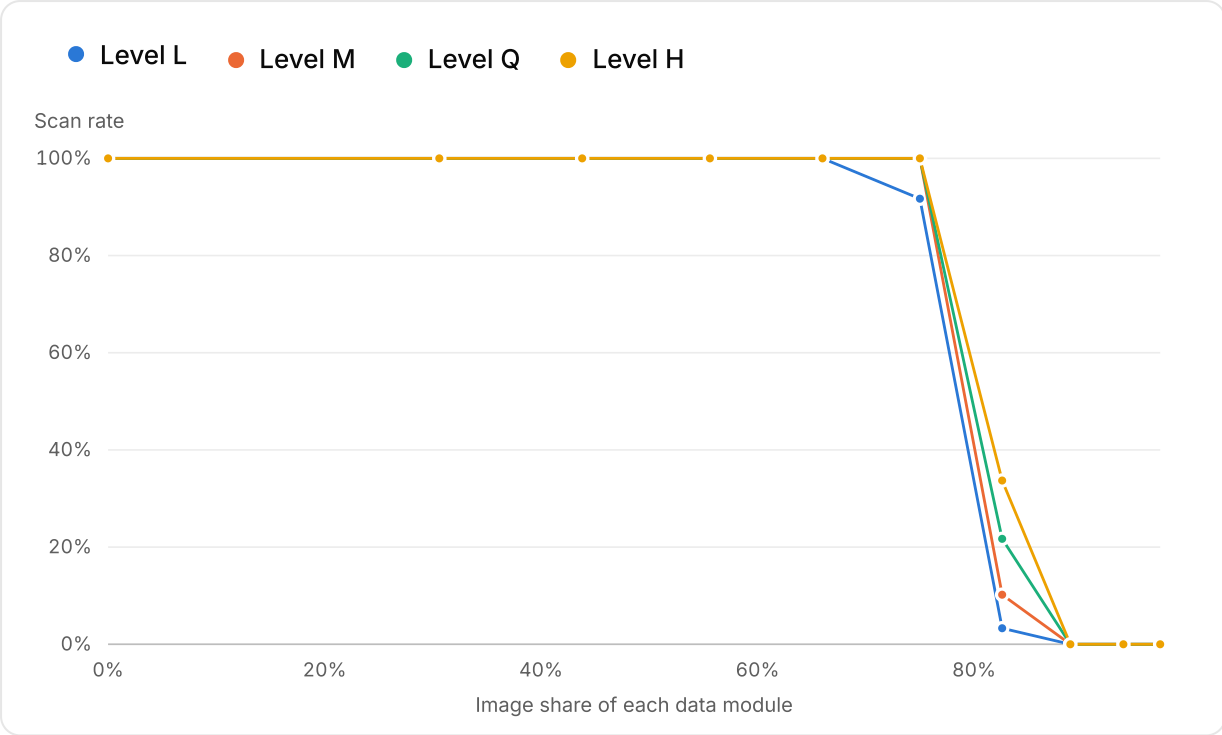


Figure 1 Scan rate against the share of each data module given to the image, for the ZXing-C++ decoder under the moderate capture condition. Each point is 600 codes. Reliable decoding ends at 66% image share for level L and at 75% for the other levels; higher levels only soften the drop. Simulated decoding.

Findings

Scan rates fall off a cliff. In almost every combination tested, the scan rate stayed at or near 100% as the data dot shrank, then collapsed within one to three dot steps. Figure 1 shows the typical shape. For ZXing-C++ under the moderate condition, level M decoded 600 of 600 codes at 75% image share and 61 of 600 (10.2%) at 82.6%. One step of dot size is one twelfth of a module.

The decoder matters more than anything else tested. Figure 3 shows the three decoders on pristine, undistorted renderings at level M. ZXing-C++ decoded every code at every dot size, including codes that were 97.2% image. ZBar was reliable up to 66% and failed completely by 93.8%. OpenCV was reliable up to 43.8% and failed completely by 75%. The same code can be unreadable to one decoder and trivial for another.

Capture quality moves the cliff a long way. Figure 4 shows ZXing-C++ at level M under four capture conditions. The largest image share that still decoded at least 95% of the time was 97.2% clean, 88.9% light, 75% moderate and 55.6% hard. Under the severe condition it was 30.6% (Table 2).

Error correction helps, but only by a step or two. Across all 15 combinations of decoder and capture condition, moving from level L to level H never raised the 95% threshold by more than two dot steps, and in six combinations it did not move it at all (Table 2). Where it helped, the effect was to soften the cliff: for ZXing-C++ under the moderate condition at 82.6% image

share, the scan rate was 3.3% at level L, 10.2% at M, 21.7% at Q and 33.7% at H. None of those is usable.

The picture itself changes the result. At that same point (ZXing-C++, moderate, 82.6% image share, all levels pooled, 400 codes per image), the scan rate was 69.5% for the moon image, 19.5% for cat, 12.3% for camera, 2.0% for coffee and 0% for astronaut and coins. Six images are too few to say which image properties drive this.

Most of the visual gain arrives early. Table 3 gives a simple measure of how closely a rendering resembles its source image. The correlation with the source rises from 0.62 at 55.6% image share to 0.75 at 75%, then only to 0.83 at 97.2%. Shrinking the dot below half a module buys little extra resemblance by this measure.

One wrong answer in 360,000. A decoder returned text that did not match the payload exactly once (ZXing-C++, level H, 66% image share, severe condition). Every other failure was a refusal to decode.

OpenCV was unreliable on ordinary codes. Under the moderate condition OpenCV decoded 62.7% of plain, non-halftone level M codes, and did slightly better (74.2%) on codes that were 30.6% image. This looks like a property of that detector under these conditions and is reported as observed, without an explanation.

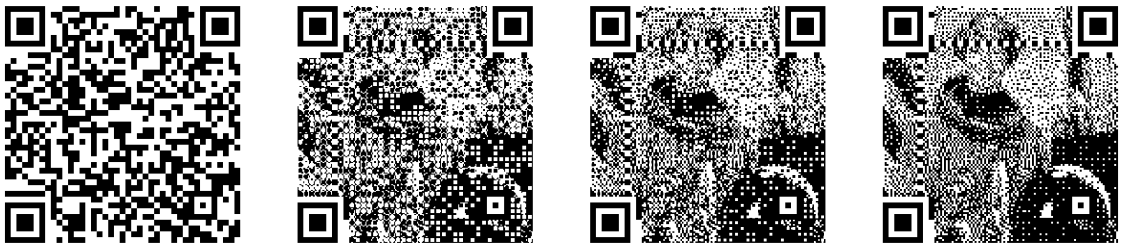


Figure 2 The same level H code rendered with data dots of 12, 8, 6 and 4 pixels in a 12-pixel module: 0%, 55.6%, 75% and 88.9% image share. Finder, timing, alignment and format patterns are always drawn solid. Simulated decoding.

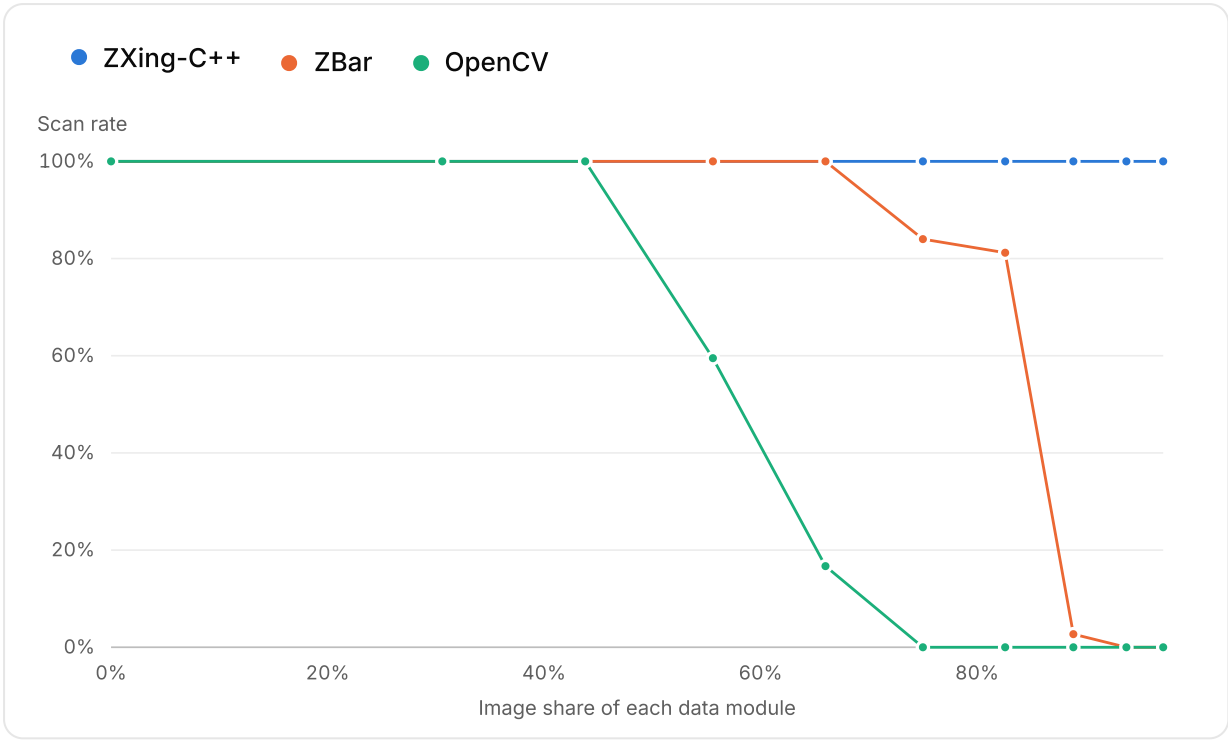


Figure 3 Three decoders on pristine renderings at level M. Each point is 600 codes. Simulated decoding.

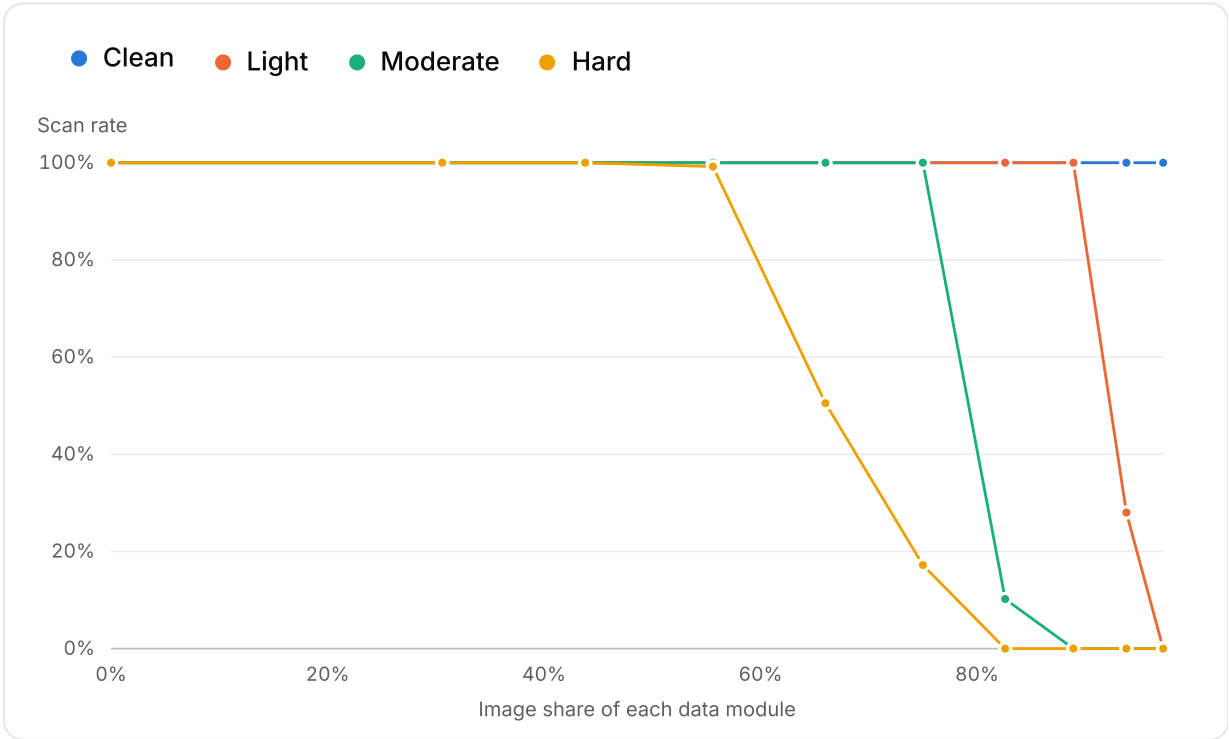


Figure 4 ZXing-C++ at level M under four capture conditions. Each point is 600 codes. Simulated decoding.

Table 2 Largest image share at which at least 95% of codes decoded, with every smaller share also at 95% or better. 'None' means even ordinary codes fell below 95%. Shares are limited to the ten dot sizes tested. Simulated decoding.

Condition	Decoder	Level L	Level M	Level Q	Level H
Clean	ZXing-C++	97.2%	97.2%	97.2%	97.2%
Clean	ZBar	66.0%	66.0%	82.6%	82.6%
Clean	OpenCV	43.8%	43.8%	43.8%	43.8%
Light	ZXing-C++	88.9%	88.9%	88.9%	88.9%
Light	ZBar	66.0%	66.0%	75.0%	75.0%
Light	OpenCV	43.8%	43.8%	43.8%	55.6%
Moderate	ZXing-C++	66.0%	75.0%	75.0%	75.0%
Moderate	ZBar	55.6%	55.6%	66.0%	66.0%
Moderate	OpenCV	none	none	none	none
Hard	ZXing-C++	43.8%	55.6%	55.6%	55.6%
Hard	ZBar	30.6%	30.6%	55.6%	55.6%
Hard	OpenCV	none	none	none	none
Severe	ZXing-C++	0%	30.6%	30.6%	30.6%
Severe	ZBar	none	0%	0%	0%
Severe	OpenCV	none	none	none	none

Table 3 Resemblance between a rendering and its source image, after both are blurred by one module. Means over 60 codes per dot size (six images, ten payloads, level M). Simulated decoding.

Dot side (px of 12)	Image share	Correlation with source	Structural similarity
12	0.0%	-0.01	0.56
10	30.6%	0.36	0.69
9	43.8%	0.51	0.75
8	55.6%	0.62	0.80
7	66.0%	0.70	0.84
6	75.0%	0.75	0.87
5	82.6%	0.78	0.90
4	88.9%	0.80	0.91
3	93.8%	0.82	0.92
2	97.2%	0.83	0.93

Method

Codes. Every code was a version 6 QR code (41 by 41 modules) in byte mode, generated with segno 1.6.6, at error-correction level L, M, Q or H. The payload was a 42-character URL: a fixed prefix followed by 20 random letters and digits. One hundred payloads were drawn from a fixed seed and reused in every cell, so cells are paired. The version was held at 6 for all levels, which means lower levels carry more padding.

Halftone rendering. Codes were drawn at 12 pixels per module with a four-module quiet zone. Function patterns (finders, separators, timing, alignment and format information) were drawn as ordinary solid modules. Every other module was filled with a one-bit halftone of a source image, and a centred square data dot in the module's own colour was painted on top. The halftone used Floyd-Steinberg error diffusion at three cells per module side. This is the simple centre-dot construction, not the optimised method of the paper that introduced halftone QR codes.[1]

The independent variable is the image share of a data module:

$$\text{image share} = 1 - \left(\frac{d}{12}\right)^2$$

where d is the side of the data dot in pixels. Ten values of d were tested: 12 (an ordinary code), 10, 9, 8, 7, 6, 5, 4, 3 and 2. Odd dot sizes sit half a pixel off centre.

Images. Six test images bundled with scikit-image 0.26.0 were used, converted to greyscale and centre-cropped square: camera, astronaut, coins, moon, cat and coffee.

Simulated capture. Each rendering was passed through five capture conditions (Table 1). A condition resamples the image to a number of pixels per module, rotates it, blurs it with a Gaussian, and adds Gaussian pixel noise seeded per trial. Figure 5 shows one code under four of them.

Table 1 Capture conditions. Blur is the Gaussian standard deviation in modules. Noise is the standard deviation in grey levels out of 255.

Condition	Pixels per module	Rotation	Blur	Noise
Clean	12	0°	0	0
Light	8	2°	0.10	4
Moderate	5	3°	0.20	8
Hard	4	5°	0.25	12
Severe	3	7°	0.30	16

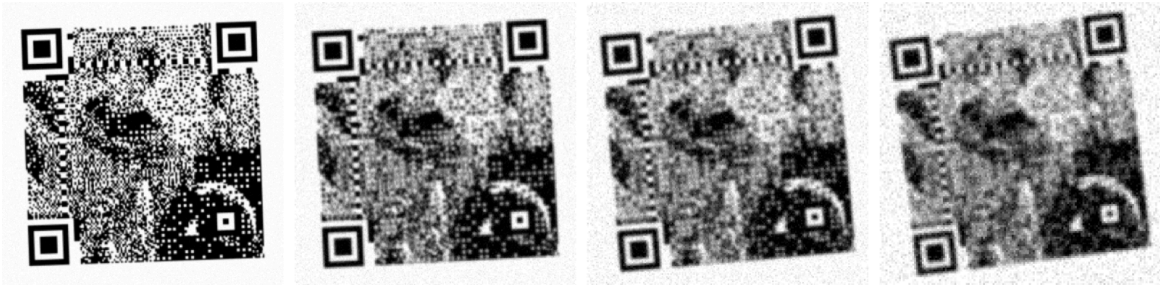


Figure 5 One level H code at 75% image share after the light, moderate, hard and severe capture conditions, enlarged to the same size. Simulated decoding.

Decoders. Three open-source decoders were run with default settings on the greyscale image: ZXing-C++ through its Python package zxing-cpp 3.2.0.post1, ZBar 0.23.95 through pyzbar 0.1.9, and OpenCV 5.1.0 using `QRCodeDetectorAruco`. OpenCV's older `QRCodeDetector` class was not used because it failed to detect ordinary undistorted codes at 8 and 12 pixels per module during setup.

Outcome. A trial is a success only if the decoder returns text equal to the payload. Each cell has 600 trials (100 payloads by 6 images). The full design is 4 levels by 10 dot sizes by 600 codes by 5 conditions by 3 decoders, which is 360,000 decode attempts. The result file `summary.csv` gives the count and a Wilson 95% interval for every cell.

Resemblance. For Table 3, each rendering (without its quiet zone) and its source image were blurred with a Gaussian of one module, then compared by Pearson correlation and by structural similarity.

```
python run_experiment.py   # writes results/trials.csv, about 7 minutes
on 14 cores
python analyze.py          # writes every table and number used here
```

Limitations

- **Data and code are not public yet.** The repository this paper was produced from has not been published, so readers cannot yet rerun the study or inspect the raw trials. It will be linked from this page.
- **Simulated only.** No code was printed and no camera was used. Real capture adds lens blur, uneven lighting, perspective, print gain, moiré and autofocus behaviour that this model does not have. The lab's own rule is that a study includes physical testing, and this version does not yet.
- **Not phone decoders.** The decoders built into iOS and Android cameras were not tested. They may behave differently from all three decoders here.
- **Default settings.** Each decoder ran once per image with default options. Phone scanners try many frames.
- **Capture conditions are bundles.** Resolution, rotation, blur and noise change together, so the study cannot say which of them does the damage. The five conditions were fixed after a pilot run of five payloads and are not calibrated against real cameras.
- **Coarse dot steps.** Dot size moves in steps of one pixel in twelve. A threshold reported as 75% means the cliff lies somewhere between 75% and 82.6%.
- **One code size and payload length.** Only version 6 with 42-character payloads was tested.
- **Simple halftone.** Published halftone QR methods choose the picture pixels to help the decoder. The centre-dot construction here does not, so better methods may reach higher image shares.
- **Six images, one resemblance measure.** The image effect is large but rests on six pictures. The resemblance measure is crude and is not a study of what people find acceptable.
- **Black and white.** Colour and grey halftones were not tested.

What AI did and what the author verified

An AI agent (Claude, made by Anthropic) did the following, working from the question set in the lab's research plan: designed the experiment, wrote the rendering, capture-simulation and analysis code, chose the capture conditions after a pilot run, ran the full experiment, produced

the tables and figures, and drafted this text. Every number in this paper was written by a script from the repository's result files, not typed by hand.

The agent also reran the whole study from a fresh clone of the repository in a new Python environment on the same machine. All 360,000 trial outcomes and every derived result file came out byte for byte identical. That shows the code is deterministic and the results follow from it. It does not show that the design is sound, and it is not a check by a person.

At version 0.1 the author has not yet rerun the repository or inspected the rendered codes independently. This section will state what was verified, and how, before the paper is published as version 1.0.

---

1. Hung-Kuo Chu, Chia-Sheng Chang, Ruen-Rone Lee and Niloy J. Mitra, "Halftone QR Codes", ACM Transactions on Graphics 32(6), article 217, 2013. doi:10.1145/2508363.2508408. ↩

Disclosure

The author operates Tilement, a QR code product.

Data and code

Not yet public. The repository for this paper will be linked here when it is available.

How to cite

Baldwin, S. (2026). How much of a QR code can be given to an image before it stops scanning. Oculean, OCL-001, version 0.1. https://oculean.com/research/ocl-001/

```
@techreport{ocl001,
  author      = {Baldwin, Stone},
  title       = {How much of a QR code can be given to an image be
  institution = {Oculean},
  number      = {OCL-001},
  year        = {2026},
  note        = {Version 0.1},
  url         = {https://oculean.com/research/ocl-001/}
}
```

Version history

0.1 / 2026-11-28

	First published version. Simulated results only. Data and code not yet public.