A Quick Overview of Video Compression and AV1: Part 1

Nasir Hemed · Follow

14 min read · Jul 17, 2021

Listen Share More

Videos have become very popular in the current era of the Internet with applications ranging from Video-on-Demand to live streaming to virtual reality. As the demand for high quality video continues to grow, it's important to ensure that they're delivered efficiently at a low bandwidth.

AV1 is an open video compression format that was developed by the Alliance of Open Media. In this blog post, I will be going over some of the core tools that are used in AV1. I will be using AOMAnalyzer to show the tools as well as some of the

things that I am currently working on in GSoC this summer. My main work included adding compound types and film grain work.

The Internet and Videos

The Internet that we know and love is filled with videos (of mostly cats probably). There is a significant demand for videos with applications such as Video on Demand, streaming, real-time communication and many others. A recent study by Sandvine showed that over 60% of the internet traffic comes from videos and this will continue to increase going forward.

Raw Videos

Consider this YouTube video I watched the other day.

In a Room 2019: An 8K video uploaded to YouTube (first upload)

It's a 30-second video recorded at 8K. Let's do some math to see the size of the raw video I was watching.

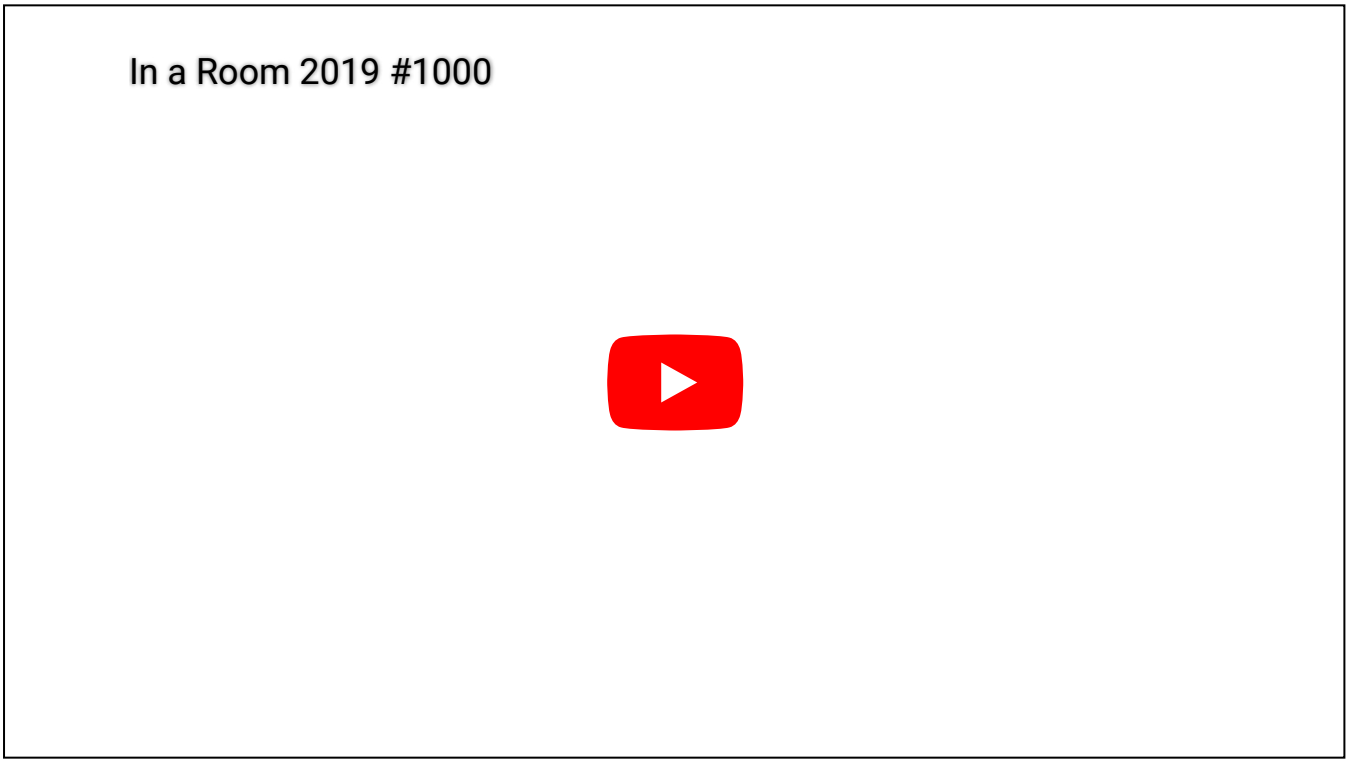
```
Resolution: 7680 x 3840 pixels
FPS (Frames Per Second): 30
Video Length: 30 seconds (900 frames)
Pixel Size (RGB) : 24 bits (8 * 3)

Total Size: 7680 * 3840 * 30 * 30 * 24 ~= 637 gigabits
Roughly 80 gigabytes
```

A 30 second video is bigger than the size of GTA 5! If you were to stream this video, you would need to have a bandwidth of `21 Gbps`. That's crazy and almost impossible for most internet connections! Yet in reality, when I downloaded the 8K video, the size of it was just 35 megabytes. What exactly is going on?

The World of Video Codecs

The answer has to do with video codecs and compression. Essentially, these are tools that compress raw videos into a smaller size (encode) and decompress while viewing them(decode). So most of the videos that we are watching have been compressed and are usually lossy, which means that some information was lost during compression (usually hard for us to notice with the eye). Thus, when we reduce the quality of a video, we ask the encoder to compress even more and thus more information is lost. It's also why when you re-upload a video again, the video gets even more distorted than the previous one.



In a Room 2019: A video that has been reuploaded 100 times

There's a lot of cool things that take place when compressing a video and you can learn more about it here. I want to touch on some of the tools being used and discuss the work that I am currently doing with AV1. It mostly revolves around extracting metadata from the decoder and making use of it. It's very useful for analyzing/debugging for codec researchers and engineers. One main use of extracting metadata is to create a bitstream analyzer.

AV1 and Royalty Free Codecs

Codecs were used for quite some time even way before they existed. They were used during times when software was mostly proprietary. They were used during the days when movies and videos were stored in CDs and you had to buy/rent them. Thus, video codecs were heavily patented and using them involved paying royalty fees. But the internet needs an open standard for video compression now more than ever given the accessibility of videos everywhere.

There have been efforts in making open and royalty-free encoders. There was a ton of work done by Google with VP9, Mozilla with Daala, Cisco with Thor and many others. Eventually, all these companies teamed up and formed a non-profit consortium known as Alliance for Open Media to develop open, royalty-free media tools.

One of the coding formats developed was AV1, which was open and royalty-free. The work of VP9, Daala and Thor were all combined to create this codec. Currently, the widely available open source encoders are rav1e, libaom, and SVT-AV1, while the most popular decoder is dav1d.

As we will see, there are a lot of tools being used in AV1. Choosing the right tool to use for a video is quite complicated. In fact, choosing the best tools may not be feasible at all and can take forever. Thus, there are heuristics and pruning that take place and that depends on the implementation of the encoder. As a video codec engineer, it's important to be able to visualize and see the tools being implemented. One of the ways to do that is by using a codec analyzer.

Bitstream Analyzer

As mentioned, there's a lot of research that goes into video codec, so being able to analyze the components of the encoder is very useful. This is what a bitstream is used for and it allows you to display the different components of the bitstream such as block partitioning, motion vectors and many other details.

The analyzer I am working with is AOMAnalyzer, which is a browser-based analyzer for viewing AV1 bitstreams. It is quite unique because of the fact that it's free to use and all you need is a browser. It mainly consists of two components: The decoder, which is a js/wasm decoder that decodes the video and extracts the relevant metadata, and the UI which displays the metadata.

The decoder currently comes from libaom, which also extracts the relevant metadata for the analyzer. Although it's written in C, there is a way to build it with emscripten. This will generate the necessary js/wasm files to be used for the aomanalyzers. You can find instructions to build it here

The UI is a React/Electron project and the source code can be found here. You can also access the built version here. So if you wanted to analyze a video, you would supply the file as an argument in the url pointing to the AV1 encoded file.

As an example, here's a link to a sample bitstream. The link is

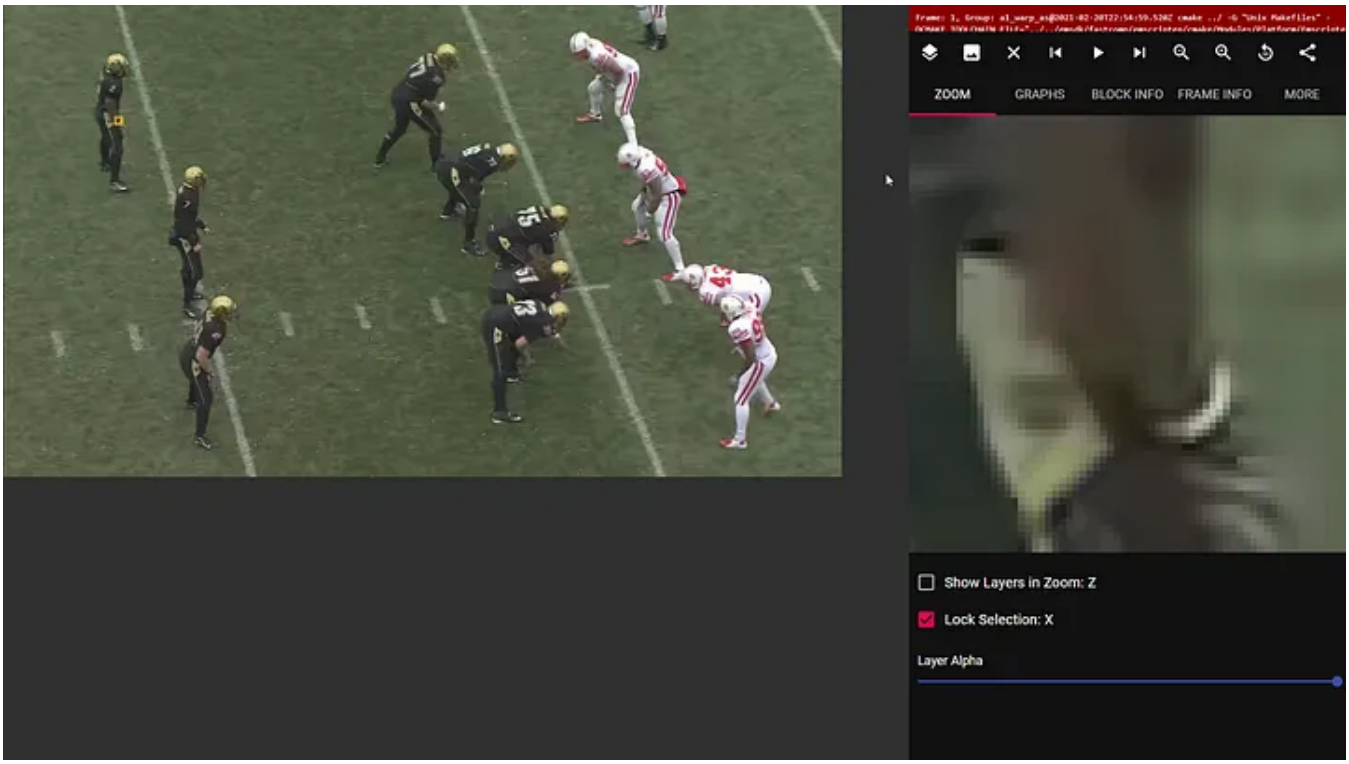
```
https://beta.arewecompressedyet.com/analyzer/?
maxFrames=4&decoder=https://media.xiph.org/analyzer/inspect.js&file=
https://mindfreeze.tk/~mindfreeze/ducks_1.00_23.ivf
```

The arguments are the following:

```
decoder: Link to the inspect decoder that was compiled
file: Link to the AV1 bitstream file to analyzer
maxFrames: Number of frames to decode. We may
    need this since decoding generates a lot
    of data for the browser
```

Looking through the Analyzer

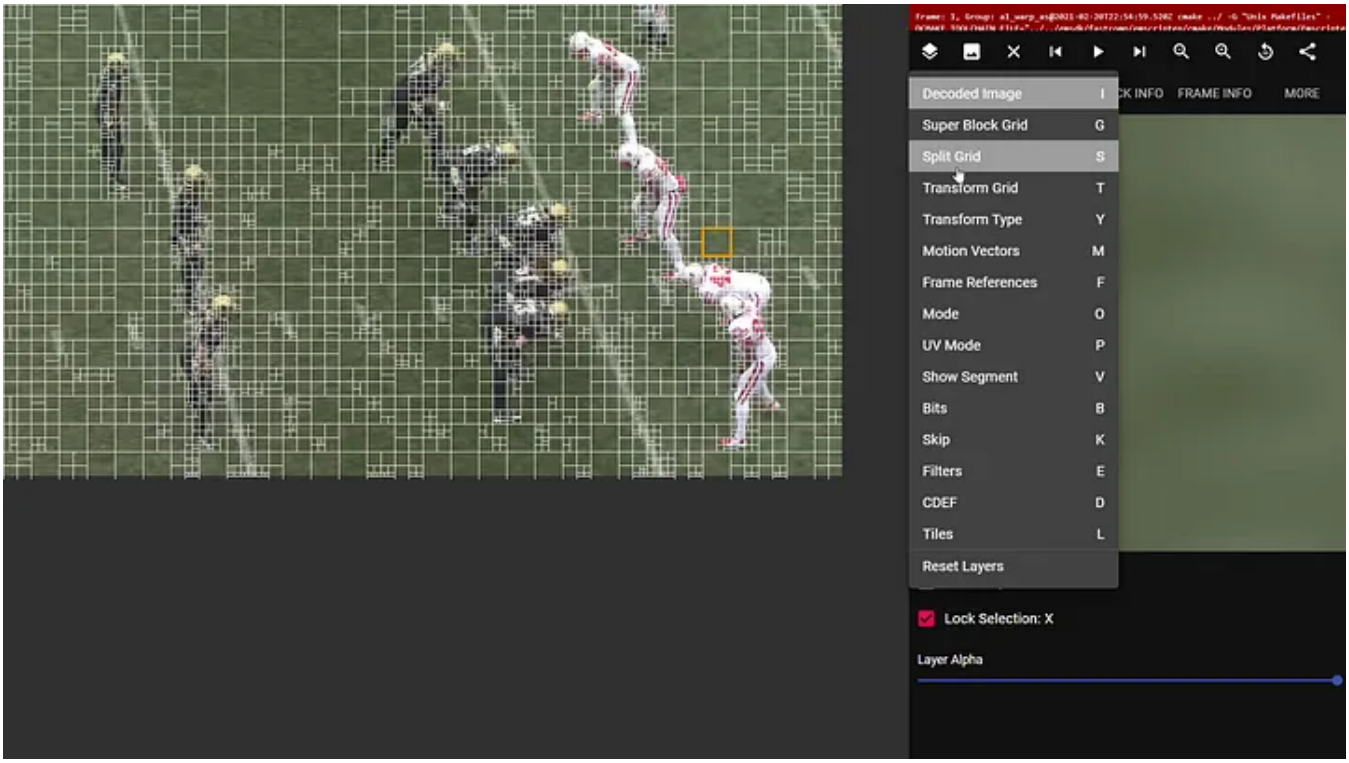
One of the ways you can learn about how encoded videos work is by looking at the videos through the lens of the analyzer. Currently, AOMAnalyzer supports the major coding tools used in AV1 such as block partitioning, motion vectors, transform types, inter/intra prediction, and many more. Let's take a look at a sample sequence:



AOMAnalyzer Football Sequence

Block Partitioning

The first step to compressing a video involves partitioning the frame into different blocks, sub-blocks (sub-sub-blocks, ...). This allows us to work on different parts of the frame and do different things on them. For example, areas that have the same color throughout have bigger partitions since the area is roughly the same, whereas detailed areas such as the player's jerseys have tiny sub-blocks since they have a lot of detail. Let's take a look at the split mode on the analyzer:

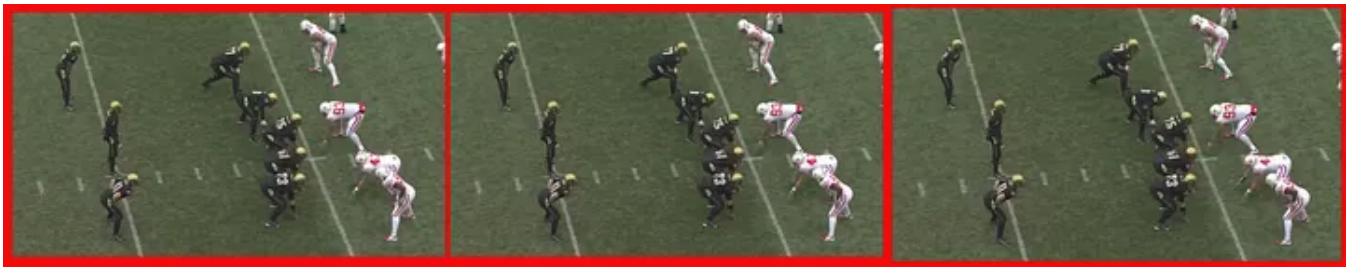


Split Grid enabled

You can see here that there are big blocks around plain areas such as the field, and smaller blocks in the players' jerseys. AV1 supports multiple block sizes ranging from 4x4 all the way up to 128x128. There's support for rectangular blocks as well. A frame can be partitioned in a lot of ways and the way you partition it is super important because that's the level in which different components will be applied. The more partitions you have, the more time and bits you spend encoding, and if you have a deadline to meet (such as real-time video or video size), you may not be able to encode in time. Thus, it's important to choose the right block partitions.

Inter/Intra Prediction

Normally for a video, there's a lot of repetition that takes place. For example, consider the sequence of the first 3 frames in the football video:



Comparison between 3 sequential frames

We can see that there's a lot of repetition here. There is a slight movement in the players preparing to kneel down. They all have the same green grass background. There are few things we can consider doing here:

Instead of storing the whole frame, why don't we have estimations of plain areas instead of all the pixel values there? Moreover, why don't we just store one main frame and predict/keep track of the movements taking place? Afterall, videos usually have some form of continuity that takes place throughout the sequence. This is where inter and intra prediction comes into play.

For the above sequence, suppose the camera doesn't move but the players move. One thing we can do is instead of displaying another copy of the frame, we can just track the movement of the players. This is where inter prediction comes in handy

On the other hand, consider one of those big blocks that's just mostly green. Instead of storing all the pixels for that block, we can estimate the color of the block and just

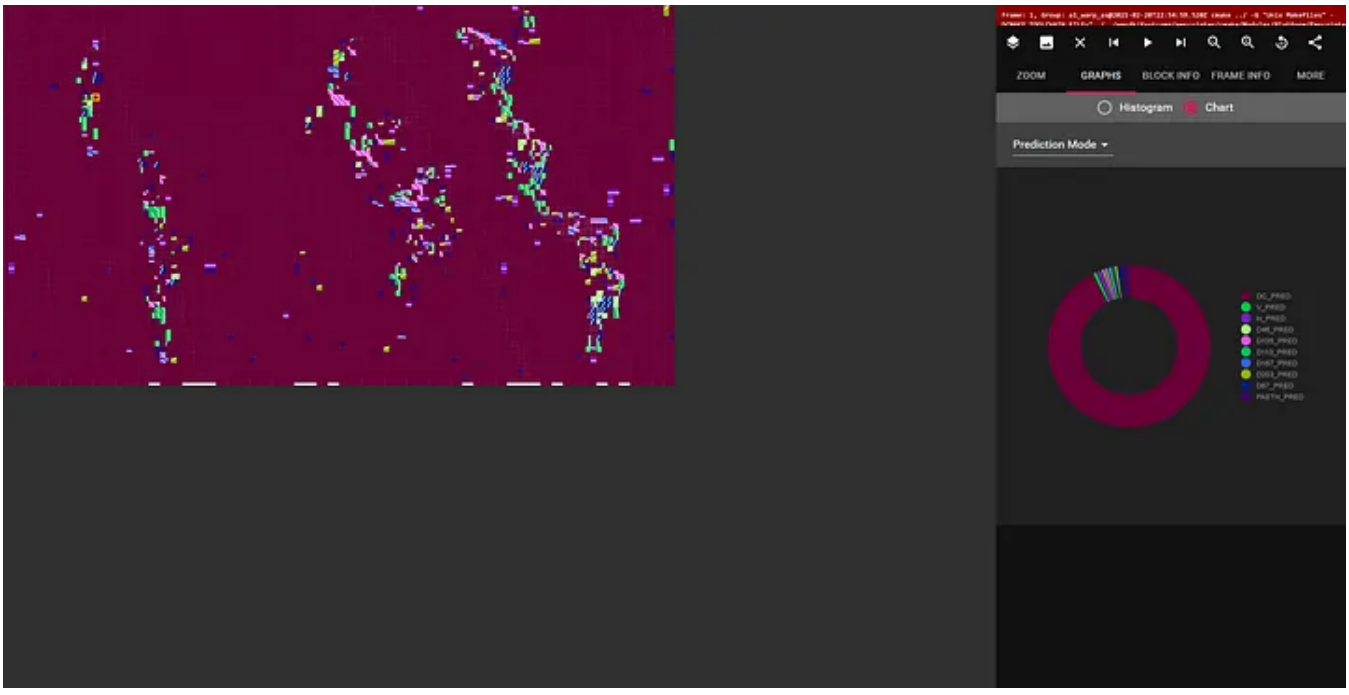
say that the color of that block is just that. This is where intra prediction comes in handy.

There are different types of frames in AV1 but essentially, they boil down to two main types: Intra Frame (keyframe) and Inter Frames (motion estimation).

Intra Prediction: Spatial Redundancy

If we take a look at the football game, there are a lot of areas that are correlated, such as the green field. There are some blocks, where the color of the block is roughly the same (greenish for example). Instead of storing pixel values for each point, we can store some correlation metadata that estimates what the pixel values will be.

AV1 has many modes for intra prediction such as Directional Intra Prediction, True motion, Chroma from Luma, and many others. Essentially, edge pixels of the corner block are used to predict the contents of the block. The mechanism used to predict the blocks depends on the mode chosen. You can see the different types of intra prediction modes used on AOMAnalyzer by choosing the `mode` layer.

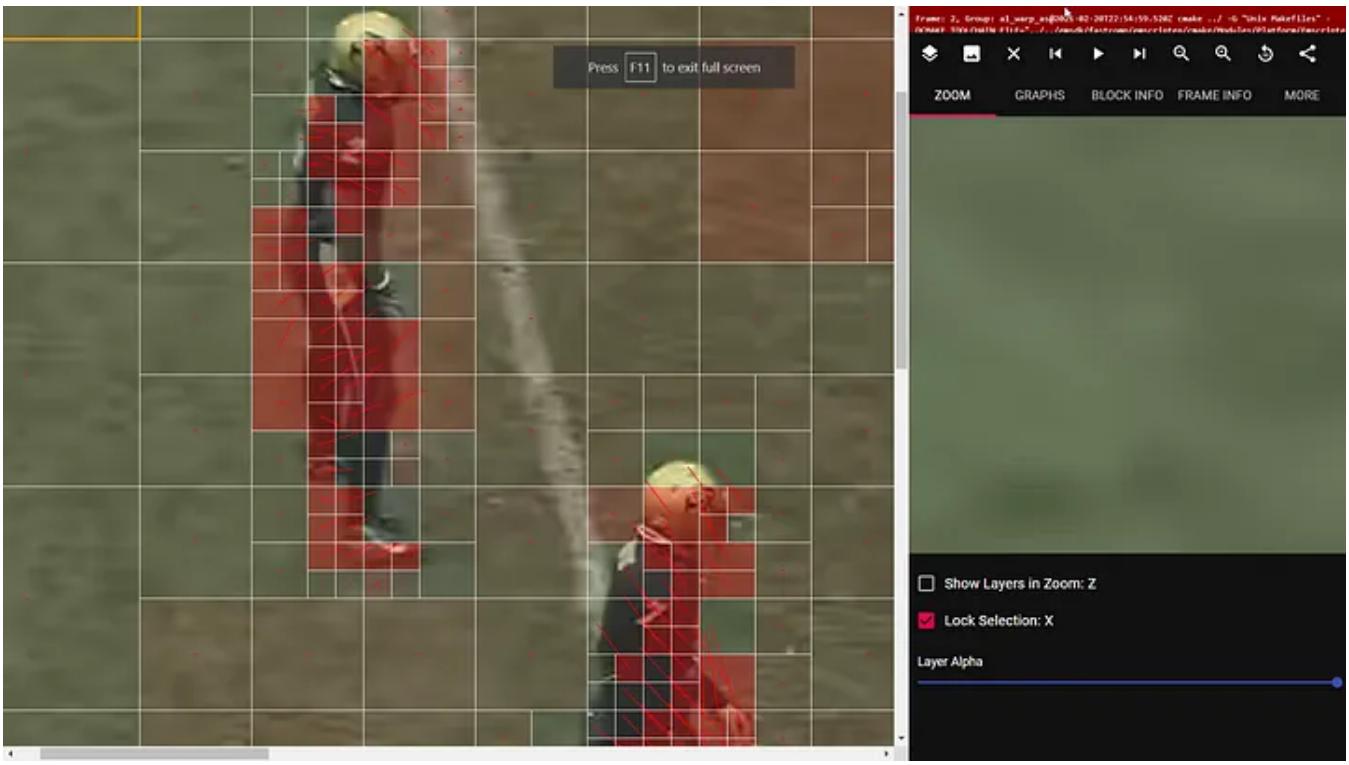


Intra prediction modes used for the first frame

Inter Prediction: Temporal Redundancy

There's a lot of motion that takes place in a video. Inter prediction, tries to predict the motion of moving object by using multiple frames. Motion vectors are used to determine the new position of the block from the reference frame. For example, in

the case of football players, there is little movement in every frame. Thus, the encoder tries to predict the motion of the blocks by comparing frames. The figure below shows what motion was predicted for the second frame on aomanalyzer.



Motion vectors for the second frame

In order to predict the motion of the block, we need a reference frame to extract the motion. In AV1, you can either use one reference frame (from the past to predict the

Compound Prediction

In the case where 2 reference frames are used, the prediction values for each reference frame will be used to generate the final motion vector predictor. The equation for predicting the value in compound prediction is:

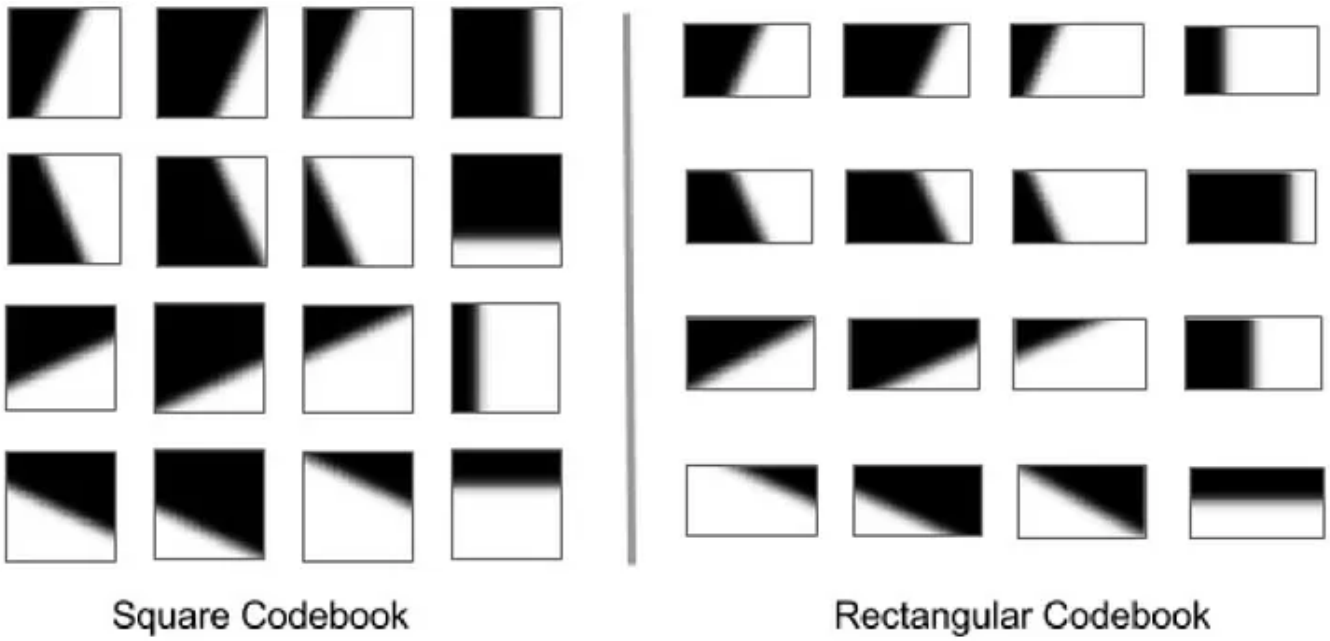
$$P(x, y) = w(x, y) * R_1(x, y) + (1 - w(x, y)) * R_2(x, y)$$

where $0 \leq w(x,y) \leq 1$ is the weight distribution for **R1** and **R2**, **R1(x, y)** and **R2(x, y)** represent the pixels at position **(x, y)** in the two reference blocks, and **P(x, y)** is the final predicted pixel value to be used. Since **w(x, y)** is a float and floats are slow for computation, the actual equation is scaled up by 64 for integer computation.

$$P(x,y) = m(x,y) * R_1(x,y) + (64 - m(x,y)) * R_2(x,y)$$

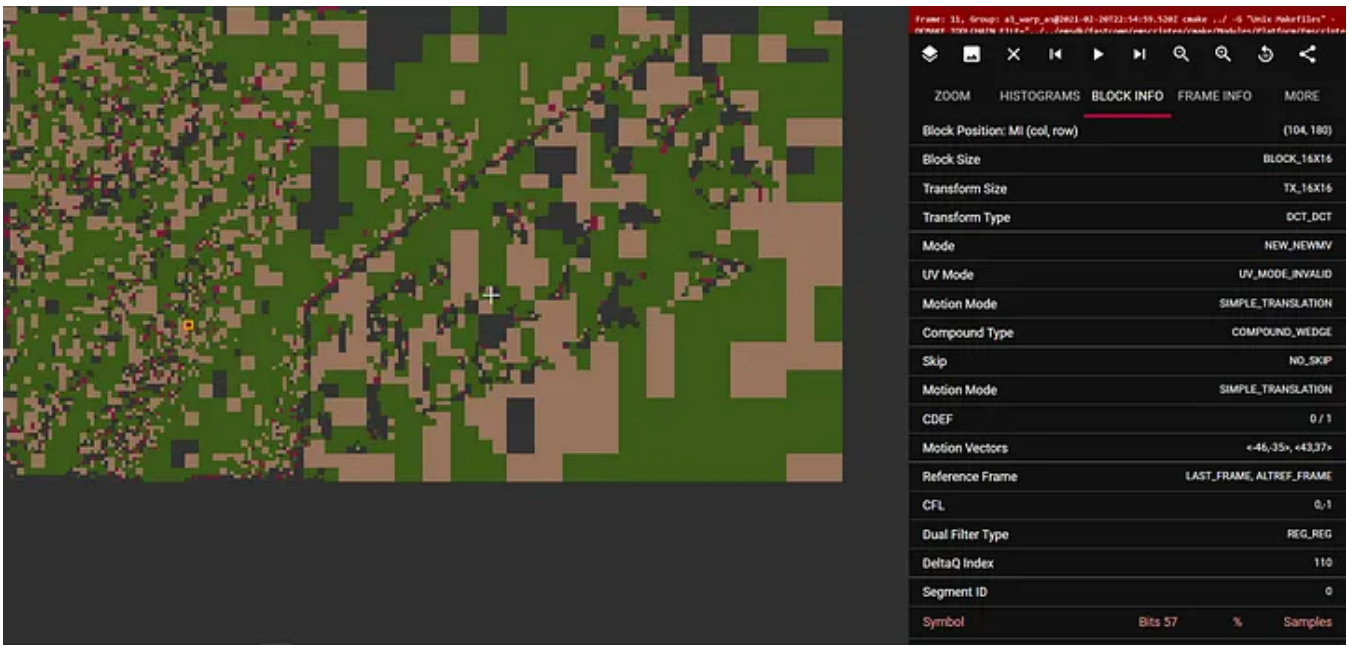
In AV1, there are three modes for choosing the weight distribution:

- COMPOUND_AVERAGE: In this mode, the weight is equally distributed between the two reference frames. Thus, **w(x, y) = 0.5** or **m(x, y) = 32**
- COMPOUND_DIST: In this case, the distance between the current frame and reference frames (**d1** and **d2**) is taken and the weight is determined based on the relative values of the distances. In this case, **m(x, y)** is a piecewise constant function by comparing the distances **d1** and **d2**
- COMPOUND_DIFF: In this case, the pixels are combined when the difference in pixels between the reference frames is very small. In the case, where the difference is large, one of the reference frames is prioritized and the pixel is weighted more towards that side
- COMPOUND_WEDGE: In this mode, the weight is determined based on a set of 16 shapes that have been preset. These shapes split the block into two sections with different angles. In the figure below, you can see the different wedge shapes that can be used. Thus, more weight is given to one of the reference frames in one region, and more weight is given to the other reference frame in the other region. A wedge sign is used to indicate which region corresponds to which reference frame.



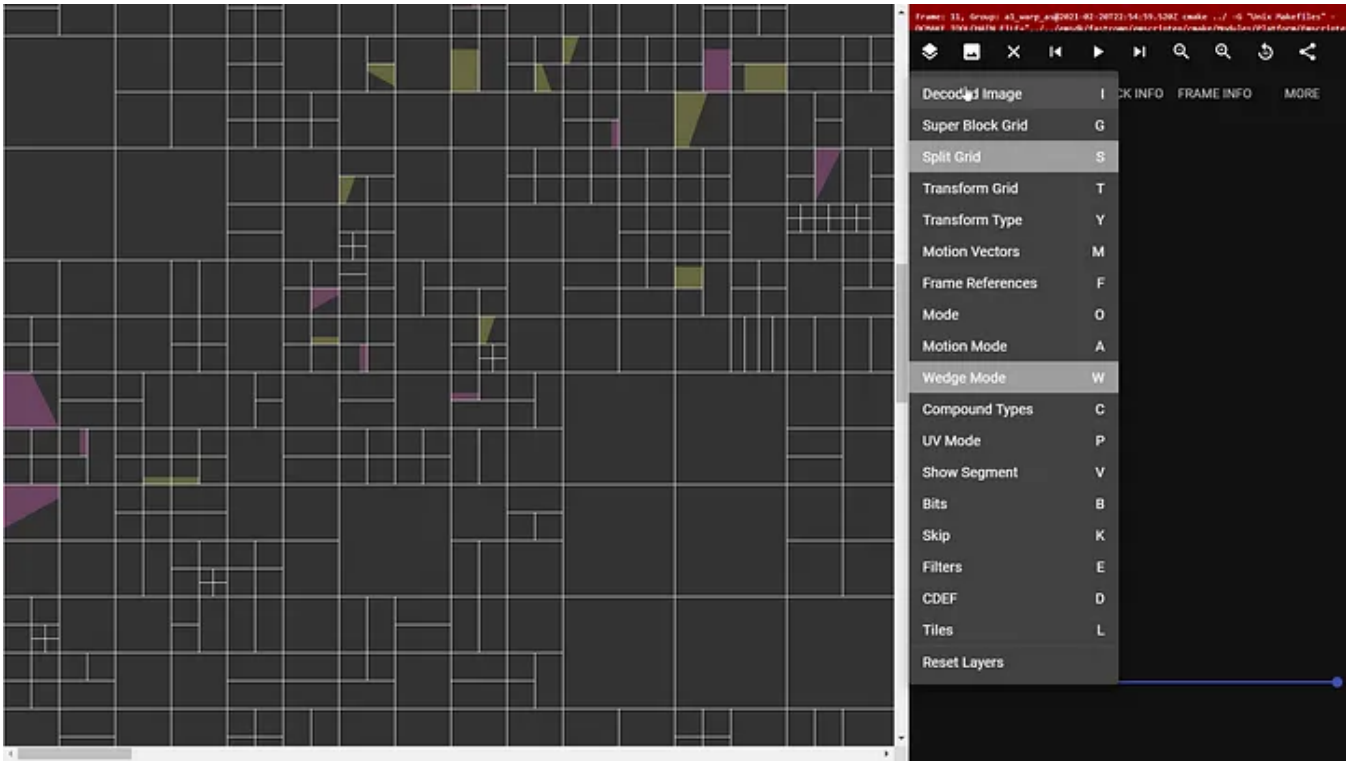
Wedge codebook for compound wedge prediction

In AOMAnalyzer, you can analyze the compound type used for each block by selecting the compound type. The figure below shows the compound types used for a frame.



Compound Types for different blocks in AOMAnalyzer

Furthermore, you can also analyze the wedge shape and sign used when COMPOUND_WEDGE is used.

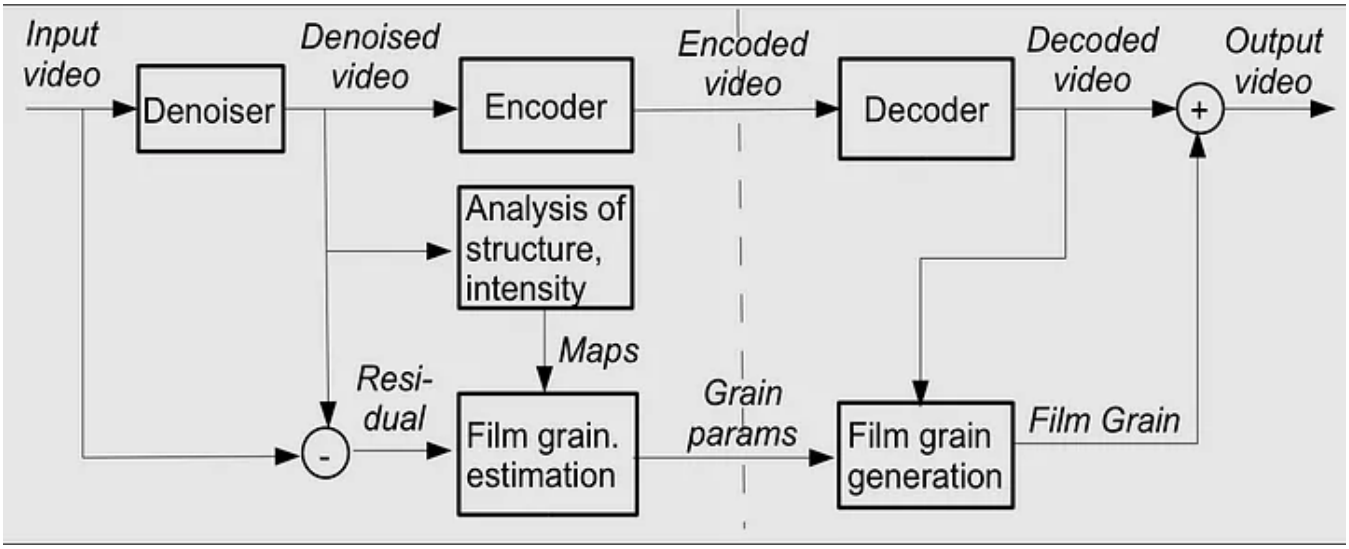


Wedge types for different blocks in AOMAnalyzer

Film Grain Synthesis

Sometimes, when recording in low light, we notice that the video is grainy. While we may want to get rid of the grain, there are cases where we need to preserve it due to its style. As described above, video compression depends a lot on spatial and temporal redundancy. As you can imagine, film grain is quite the opposite due to randomness. It changes drastically with time and is randomly different across the frame. Thus, preserving and compressing grain is quite difficult for encoders. In fact, some of the codec tools used in compression can suppress and remove film grain.

Instead of preserving film grain, another approach would be to generate artificial grain. AV1 has a tool that allows you to generate synthetic grain and add it to the video. Prior to encoding, the user can first detect the grain, denoise the source video, determine the film grain parameters and pass them to the encoder along with the input video. The encoder then generates film grain parameters that the decoder uses to add the grain once the video is decoded. The figure below shows the step in which film grain is generated and added.



Enocder/Decoder model for AV1

Quick Tangent: How to use film grain in AV1

Let's say you have a grainy video and wanted to preserve film grain. Assume it's a YUV file with height h and width w.

Denoise your video: You can use ffmpeg to denoise your video. There are multiple ways of doing that. As an example, here's a command using hqdn3d filter

```
ffmpeg -vcodec rawvideo -video_size wxh -i input.yuv \
    -vf hqdn3d=5:5:5:5 -vcodec rawvideo -an -f rawvideo \
```

```
    denoised.854_480.yuv
```

Once you denoise the video, you will need to estimate film grain parameters. libaom currently has an example noise model that models the noise between the source and the denoised video and generates the film grain parameters. Once built, you can run the following command:

```
./examples/noise_model --fps=fps --width=width --height=height --
i420 \
    --input-denoised=denoised.yuv --input=original.yuv \
    --output-grain-table=film_grain.tbl
```

You can then encode your denoised video using an AV1 encoder and pass the film grain table as a parameter. Here's an example of encoding using libaom:

```
aomenc --cpu-used=4 --input-bit-depth=bit-depth  \
    --i420 -w W -h H --end-usage=q --cq-level=25 --lag-in-frames=25
\
    --auto-alt-ref=2 --bit-depth=bit-depth --film-grain-
table=film_grain.tbl \
    -o denoised_with_grain_params.ivf denoised.yuv
```

How Film Grain Synthesis works

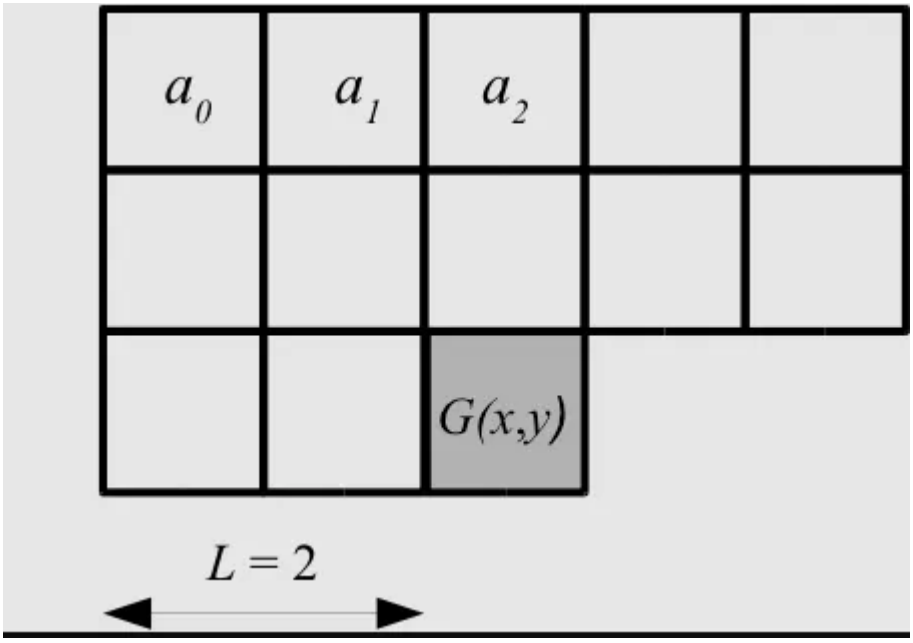
The film grain synthesis is a "pseudorandom" process. This is because one of the film grain parameters is the `random_seed` which ensures that the grain pattern is the same if the process is repeated

The film grain pattern is generated in a raster scan order using an autoregressive (AR) model. The parameters of this model are specified by one of the film grain parameters, `ar_coeff_<luma, cb, cr>`, which is a list of AR coefficient values. The `ar_coeff_lag` parameter determines the neighborhood range of values to pick from the previous blocks. For each pixel at position (x, y), the grain value is the following:

$$G(x,y) = a_0 * G(x-2, y-2) + a_1 * G(x-1, y-2) + \cdots + z*$$

where **a0 , ... , aN** are the AR coefficients and **z** is a unit-variance Gaussian noise. The figure below is an example of how the grain value at **(x, y)** is calculated wtith

the `ar_coeff_lag = 2`:



Film grain pattern value at pixel (x,y)

This process will generate a 64X64 grain block for luma and a 32X32 grain block for chroma blue/red. Since each grain pixel depends on the previous pixels, the blocks are initially padded so that the top-left corner blocks can have proper values. Once the grain samples are generated, the grain is applied to the frame in blocks of size 32X32 for luma by taking a random subblock of 32X32 from the grain block and applying it to the frame block. For chroma blue/red, the size of the subblock depends on chroma subsampling. In the case of 4:2:0, the size would be 16X16. For 4:0:0, the size would be 8X8.

The final pixel position at (x, y) will be:

$$\hat{P}(x,y) = P(x,y) + f(t) * G(x,y)$$

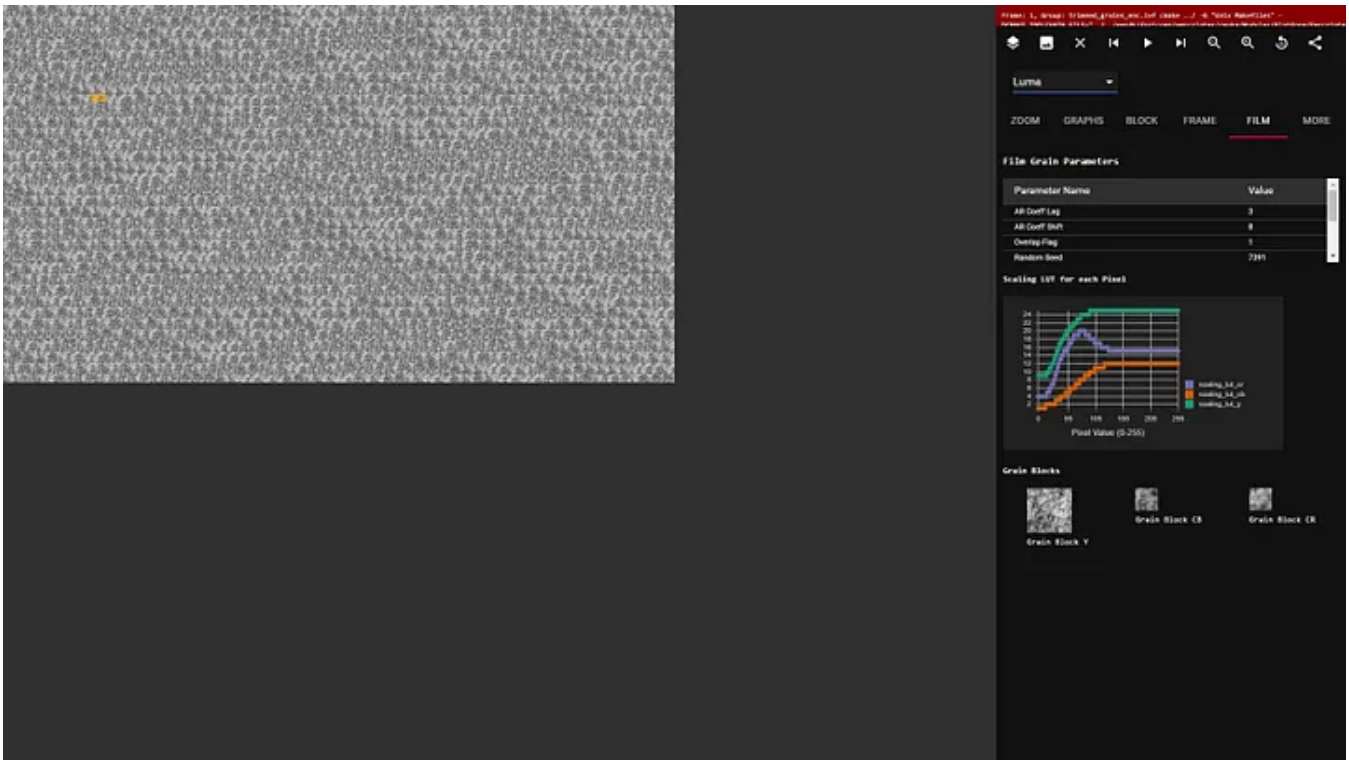
where **P(x, y)** is the denoised pixel value, **G(x, y)** is the grain block value, and **f** is a piecewise-linear function that scales the grain sample according to the pixel intensity. The function **f** is generated from the scaling points provided by the film grain parameters. For luma values, the value **t** is just the pixel value **P(x, y)**. However, for chroma blue/red, the value t is the following:

$$t = b_u * P(x,y) + d_u * avg(P(x,y)) + h_u$$

where **avg(P(x, y))** is the average of the collocated luma pixels and **b_u**, **d_u**, and **h_u** are values specified by the film grain parameters.

Analyzing Film Grain through AOMAnalyzer

I am currently working on adding Film Grain analysis to AOMAnalyzer. Currently, you are able to see the what the film grain parameters are, as well as the scaling function **f** for all 8-bit pixel values. You are also able to see the 64x64 luma grain sample as well as the 32x32 chroma grain samples that were generated from the film grain parameters. Moreover, you can also see what the grain image looks like for luma, and chroma.



Film Grain Info along with the luma grain image in AOM Analyzer

Conclusion

There are still a lot more coding tools being used in AV1 such as Transform Coding, Entropy Coding, In-Loop Filters and much more. There's also a lot of work left to do for GSoC in the analyzer. In the next blog post, I will discuss the tools used in AV1 and visualize them with AOMAnalyzer.

Video Compression | Av1 | Compression | Analyzer

Follow

Written by Nasir Hemed

16 Followers

I study computer science at the University of Toronto. I love math and I'm interested in data science.
https://www.github.com/nasirhemed

More from Nasir Hemed

Nasir Hemed in Towards Data Science

A deeper look at descent algorithms

Overview and comparison of different descent algorithms

· 9 min read · Apr 22, 2019

94

See all from Nasir Hemed

Recommended from Medium

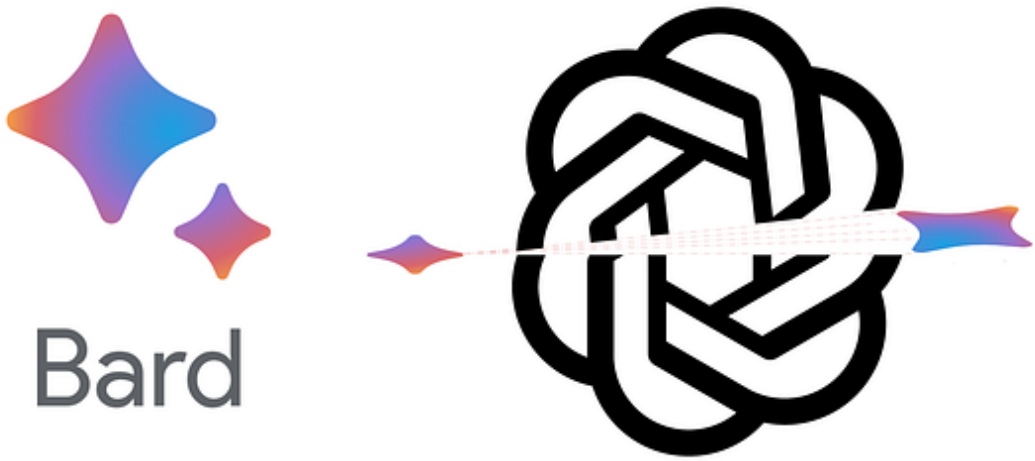
Vaishnav Manoj in DataX Journal

JSON is incredibly slow: Here's What's Faster!

Unlocking the Need for Speed: Optimizing JSON Performance for Lightning-Fast Apps and Finding Alternatives to it!

16 min read · Sep 28

10.7K 125



AL Anany

The ChatGPT Hype Is Over—Now Watch How Google Will Kill ChatGPT.

It never happens instantly. The business game is longer than you know.

· 6 min read · Sep 1

20K 626

Lists

Staff Picks

531 stories · 518 saves

Stories to Help You Level-Up at Work

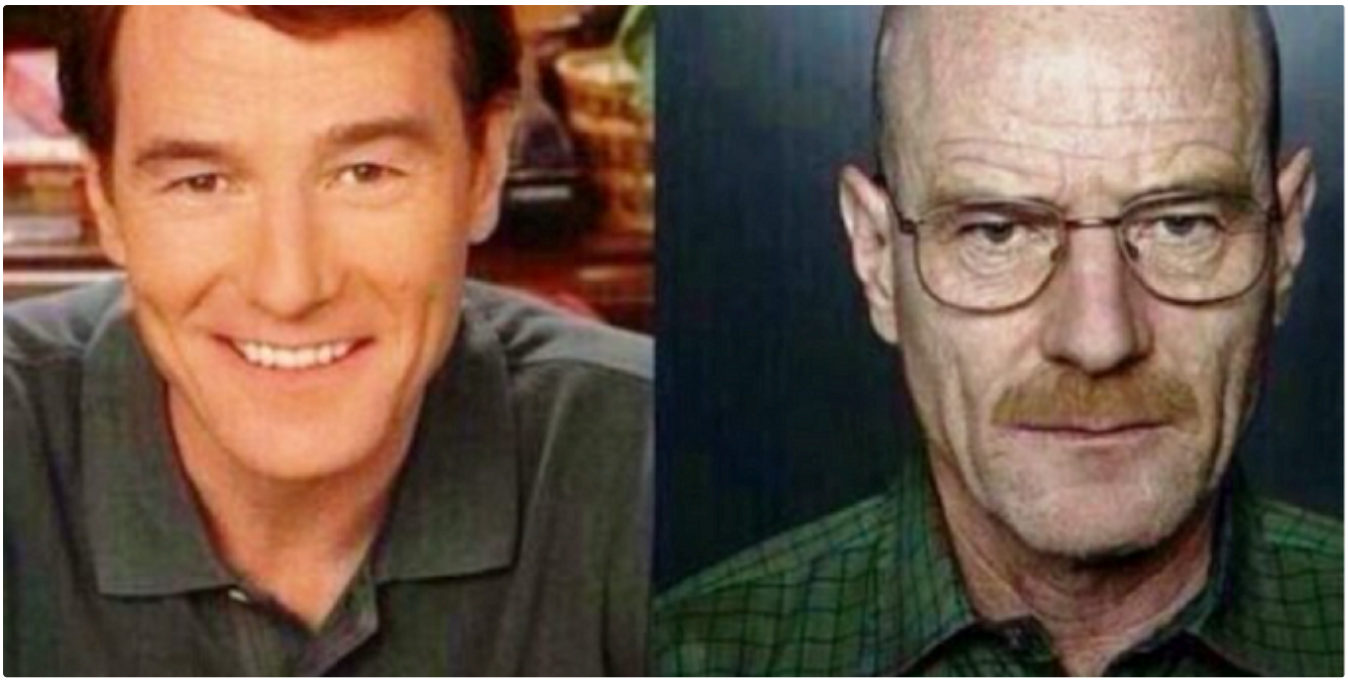
19 stories · 361 saves

Self-Improvement 101

20 stories · 1022 saves

Productivity 101

20 stories · 928 saves



David Goudet

This is Why I Didn't Accept You as a Senior Software Engineer

An Alarming Trend in The Software Industry

· 5 min read · Jul 26

5.7K 59

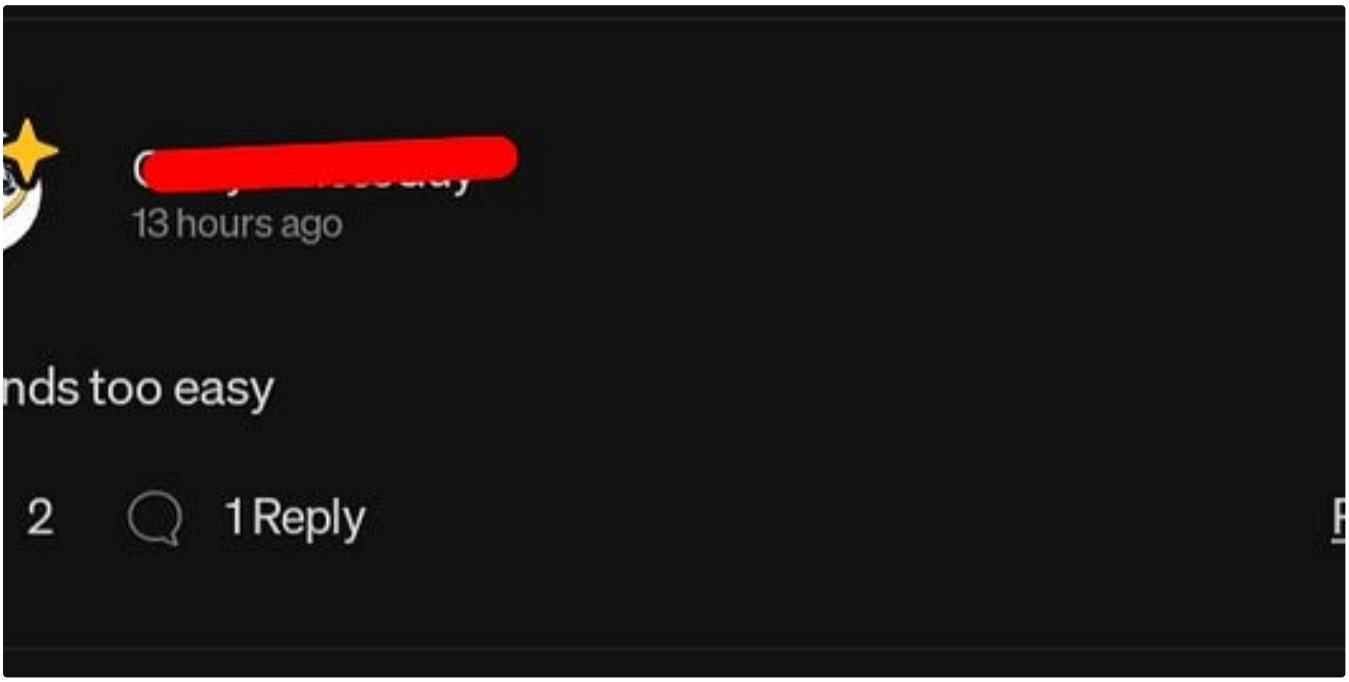
Lessig

ChatGPT, or: How I Learned to Stop Worrying and Love AI

In my first book, Code and Other Laws of Cyberspace (1999), I told the story of why I had become a lawyer. My uncle, Richard Cates, had...

7 min read · Nov 29

5K 123



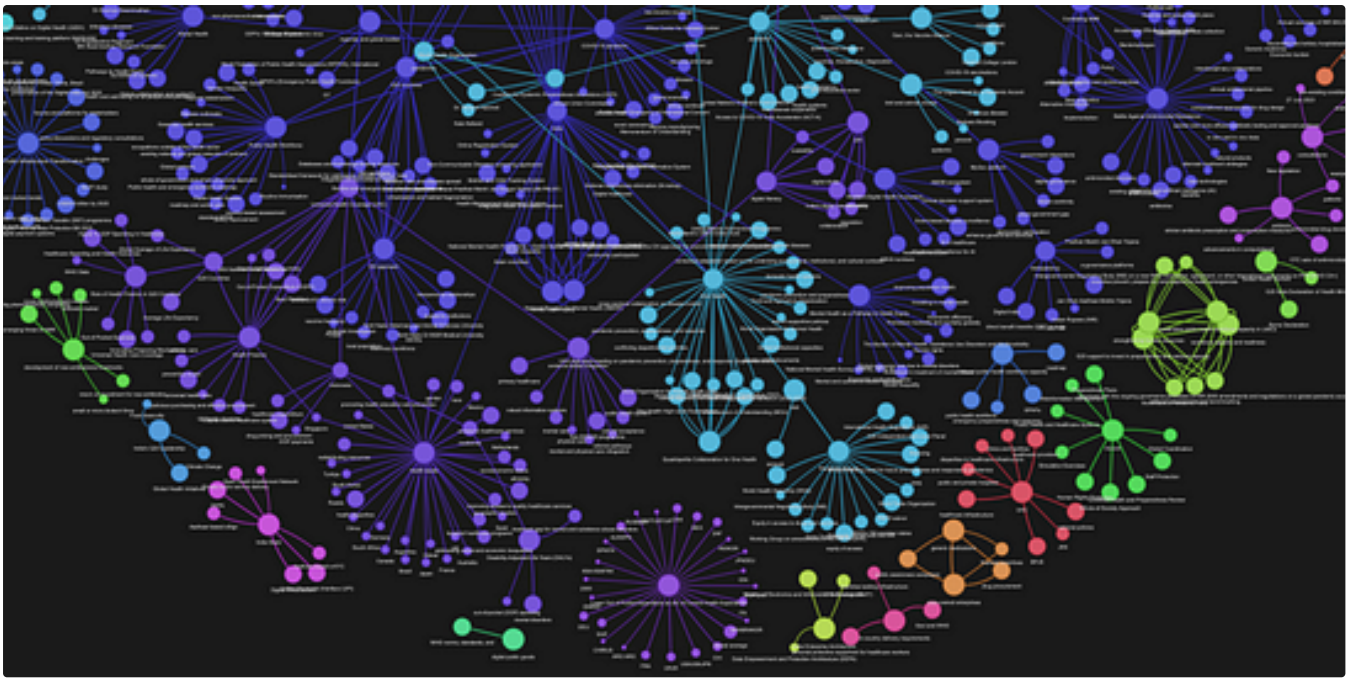
Paul Rose

I Found A Very Profitable AI Side Hustle

And it's perfect for beginners

6 min read · Oct 19

14K 242



 Rahul Nayak in Towards Data Science

How to Convert Any Text Into a Graph of Concepts

A method to convert any text corpus into a Knowledge Graph using Mistral 7B.

12 min read · Nov 10

4.2K 41