Lecture 16:

Video Compression + Basic Video Conferencing Systems

Visual Computing Systems
Stanford CS348K, Spring 2024
Estimate: 82% of internet traffic will be video

Global IP Traffic by Application Type
By 2022, video will account for 82% of global IP traffic

26% CAGR 2017–2022

* Figures (n) refer to 2017, 2022 traffic share

Source: Cisco VNI Global IP Traffic Forecast, 2017–2022
Basically, we’re watching TV and movies…

Global Internet Video Traffic by Type
By 2022, live video will increase 15-fold and reach 17% of Internet video traffic

33% CAGR
2017–2022

Exabytes per Month

2017 2018 2019 2020 2021 2022

- Video Surveillance (2%, 3%)
- Live Internet Video (5%, 17%)
- Long-Form Internet VoD (61%, 62%)
- Short-Form Internet VoD (32%, 18%)

* Figures (n) refer to 2017, 2022 traffic share
Source: Cisco VNI Global IP Traffic Forecast, 2017–2022
20 second video: 1920 x 1080, @ 30fps
After decode: 8-bits per channel RGB → 24 bits/pixel → 6.2 MB/frame
(6.2 MB/frame x 20 sec x 30 fps = 3.5 GB)
Size of data when each frames stored as JPG: 404 MB
Video file size when compressed using H.264: 26.6 MB (133-to-1 compression ratio compared to uncompressed, 8-to-1 compared to JPG)
H.264 Video Compression
H.264/AVC video compression

- AVC = advanced video coding
- Also called MPEG4 Part 10
- Common format in many modern HD video applications:
  - HD streaming video on internet (YouTube, Vimeo, iTunes store, etc.)
  - HD video recorded by your smartphone
  - European broadcast HDTV (U.S. broadcast HDTV uses MPEG 2)
  - Some satellite TV broadcasts (e.g., DirecTV)
- Benefit: higher compression ratios than MPEG2 or MPEG4
  - Alternatively, higher quality video for fixed bit rate
- Costs: higher decoding complexity, substantially higher encoding cost
  - Idea: trades off more compute for requiring less bandwidth/storage
Hardware implementations

- Support for H.264 video encode/decode is provided by fixed-function hardware on most modern processors (not just mobile devices).

- Hardware encoding/decoding support existed in modern Intel CPUs since Sandy Bridge (Intel “Quick Sync”).

- Modern operating systems expose hardware encode decode support through hardware-accelerated APIs
  - e.g., DirectShow/DirectX (Windows), AVFoundation (iOS)
Terminology: video container format versus video codec

- Video container (MOV, AVI) bundles media assets

- Video codec: H.264/AVC (MPEG 4 Part 10)
  - H.264 standard defines how to represent and decode video
  - H.264 does not define how to encode video (this is left up to implementations)
Video compression: main ideas

Compression is about exploiting redundancy in a signal

- Intra-frame redundancy: value of pixels in neighboring regions of a frame are good predictor of values for other pixels in the frame (spatial redundancy)

- Inter-frame redundancy: pixels from nearby frames in time are a good predictor for the current frame’s pixels (temporal redundancy)
Residual: difference between compressed image and original image

In video compression schemes, the residual image is compressed using lossy compression techniques. Better predictions lead to smaller and more compressible residuals!
Residual: difference between predicted pixel values and input video pixel values

In other words: The main idea today: use an algorithm to predict what a future pixel should be, then store a description of the algorithm and the residual of the prediction.
16 x 16 macroblocks

Video frame is partitioned into 16 x 16 pixel macroblocks

Due to 4:2:0 chroma subsampling, macroblocks correspond to 16 x 16 luma samples and 8 x 8 chroma samples
Macroblocks in an image are organized into slices

Figure to left shows the macroblocks in a frame (boxes are macroblocks not pixels)

Macroblocks are grouped into “slices”

Can think of a slice as a sequence of macroblocks in raster scan order *

Slices can be decoded independently **
(Facilitates parallel decode + robustness to transmission failure)

* H.264 also has non-raster-scan order modes (FM0), will not discuss today.

** Final “deblocking” pass is often applied to post-decode pixel data, so technically slices are not fully independent.
Decoding via prediction + correction

During decode, samples in a macroblock are generated by:

1. Making a prediction based on already decoded samples in macroblocks from the same frame (intra-frame prediction) or from other frames (inter-frame prediction)
2. Correcting the prediction with a residual stored in the video stream

Three forms of prediction:

- I-macroblock: (“intra-picture predictive only”) macroblock samples predicted from samples in previous macroblocks in the same slice of the current frame
- P-macroblock: (“predictive”) macroblock pixel samples can be predicted from samples from one other frame (one prediction per macroblock)
- B-macroblock: (“bipredictive”) macroblock pixel samples can be predicted by a weighted combination of multiple predictions from samples from other frames
Intra-frame prediction (I-macroblock)

- Prediction of sample values is performed in spatial domain, not transform domain
  - Predict pixel values, not basis coefficients

- Modes for predicting the 16x16 luma (Y) values:
  - Intra_4x4 mode: predict 4x4 block of samples from adjacent row/col of pixels
  - Intra_16x16 mode: predict entire 16x16 block of pixels from adjacent row/col
  - I_PCM: actual sample values provided

* An additional 8x8 mode exists in the H.264 High Profile

Yellow pixels: already reconstructed (values known)
White pixels: 4x4 block to be reconstructed
Intra_4x4 prediction modes

- Nine prediction modes (6 shown below)
  - Other modes: horiz-down, vertical-left, horiz-up

Mode 0: vertical
(4x4 block is copy of above row of pixels)

Mode 1: horizontal
(4x4 block is copy of left col of pixels)

Mode 2: DC
(4x4 block is average of above row and left col of pixels)

Mode 3: diagonal down-left (45°)

Mode 4: diagonal down-right (45°)

Mode 5: vertical-right (26.6°)
Intra_4x4 prediction modes (another look)

AVC/H.264 intra prediction modes

Intra_16x16 prediction modes

4 prediction modes: vertical, horizontal, DC, plane

Mode 0: vertical
Mode 1: horizontal
Mode 2: DC
Mode 4: plane

\[ P[i,j] = A_i \times B_j + C \]
A derived from top row, B derived from left col, C from both
Intra-prediction of chroma (8x8 block) is performed using four modes similar to those of intra_16x16 (except they are reordered as: DC, vertical, horizontal, plane). Each mode is a different prediction algorithm, so we have to store which algorithm we chose in the video stream in order to decode it.

Intra-prediction scheme for each 4x4 block within macroblock encoded as follows:

- One bit per 4x4 block:
  - if 1, use most probable mode
    - Most probable = lower of modes used for 4x4 block to left or above current block
  - if 0, use additional 3-bit value rem_intra4x4_pred_mode to encode one of nine modes
    - if intra4x4_pred_mode is smaller than most probable mode, then actual mode is given by intra4x4_pred_mode
    - else, actual mode is intra4x4_pred_mode + 1
Inter-frame prediction (P-macroblock)

- Predict sample values using values from a block of a previously decoded frame *

- Basic idea: pixels in current frame are given by some translation of pixels from temporally nearby frames (e.g., consider an object that moved slightly on screen between frames)
  - “Motion compensation”: use of spatial displacement to make prediction about pixel values

* Note: “previously decoded” does not imply source frame must come before current frame in the video sequence. (H.264 supports decoding out of order.)
P-macroblock prediction

- Prediction can be performed at macroblock or sub-macroblock granularity
  - Macroblock can be divided into 16x16, 8x16, 16x8, 8x8 "partitions"
  - 8x8 partitions can be further subdivided into 4x8, 8x4, 4x4 sub-macroblock partitions
- Each partition predicted by sample values defined by:
  (reference frame id, motion vector)

Block A: predicted from (frame 0, motion-vector = [-3, -1])
Block B: predicted from (frame 1, motion-vector = [-2.5, -0.5])

Note: non-integer motion vector
Motion vector visualization

Image credit: Keyi Zhang
Non-integer motion vectors require resampling

H.264 supports both 1/2 pixel and 1/4 pixel resolution motion vectors
1/4 resolution resampling performed by bilinear interpolation of 1/2 pixel samples
1/8 resolution (chroma only) by bilinear interpolation of 1/4 pixel samples

Example: motion vector with 1/2 pixel values.
Must resample reference block at positions given by red dots.

Interpolation to 1/2 pixel sample points via 6-tap filter:
half_integer_value = clamp((A - 5B + 20C + 20D - 5E + F) / 32)
Motion vector prediction

- Problem: per-partition motion vectors require significant amount of storage
- Solution: predict motion vectors from neighboring partitions and encode residual in compressed video stream
  - Example below: predict block D’s motion vector as average of motion vectors from block A, B, C
  - Prediction logic becomes more complex when partitions of neighboring blocks are of different size
Question: what partition size is best?

- Smaller partitions likely yield more accurate prediction
  - Fewer bits needed for residuals

- Smaller partitions require more bits to store partition information (diminish benefits of prediction)
  - Must store:
    - Source picture id
    - Motion vectors (note that motion vectors are more “coherent” in adjacent blocks with finer sampling, so they likely compress well)
### Inter-frame prediction (B-macroblock)

- Each partition predicted by up to two source blocks
  - Prediction is the average of the two reference blocks
  - Each B-macroblock partition stores two frame references and two motion vectors (recall P-macroblock partitions only stored one)
Optional weighting to prediction:

- Per-slice explicit weighting (reference samples multiplied by weight)
- Per-B-slice implicit weights (reference samples weights by temporal distance of reference frame from current frame in video)
  - Idea: weight samples from reference frames nearby in time more
Post-process filtering

- **Deblocking**
  - Blocking artifacts may result as a result of macroblock granularity encoding.
  - After macroblock decoding is complete, optionally perform smoothing filter across block edges.

[Image credit: Averbuch et al. 2005]
Putting it all together: encoding an inter-predicted macroblock

- **Inputs:**
  - Current state of decoded picture buffer (state of the video decoder)
  - 16x16 block of input video that the encoder needs to encode

- **General steps:** (need not be performed in this order)
  - Resample images in decoded picture buffer to obtain 1/2, and 1/4, 1/8 pixel resampling
  - Choose prediction type (P-type or B-type)
  - Choose reference pictures for prediction
  - Choose motion vectors for each partition (or sub-partition) of macroblock
  - Predict motion vectors and compute motion vector difference
  - Encode choice of prediction type, reference pictures, and motion vector differences
  - Encode residual for macroblock prediction
  - Store reconstructed macroblock (post deblocking) in decoded picture buffer to use as reference picture for future macroblocks
H.264/AVC video encoding

$MB = \text{macroblock}$

$MV = \text{motion vector}$

Intra-frame Prediction → Predicted MB → Compute Residual → Transform/Quantize Residual → Entropy Encoder → Compressed Video Stream

Inter-frame Prediction → Motion Vector Pred. → Compute MV Diffs →

Actual MB pixels → Basis coefficients

Decoded picture buffer → Deblock → Inverse transform/quantize
Motion estimation algorithms

- Encoder must **find** reference block that predicts current frame’s pixels well.
  - Can search over multiple pictures in decoded picture buffer + motion vectors can be non-integer (huge search space)
  - Must also choose block size (macroblock partition size)
  - And whether to predict using combination of two blocks
  - Literature is full of heuristics to accelerate this process
    - Remember, must execute motion estimation in real-time for HD video (1920x1080) on a low-power smartphone

Limit search window:

gray area: search region

Decoded picture buffer: frame 0

Current frame
Motion estimation algorithm optimizations

- **Coarser search:**
  - Limit search window to small region
  - First compute block differences at coarse scale (save partial sums from previous searches)

- **Smarter search:**
  - Guess motion vectors similar to motion vectors used for neighboring blocks
  - Diamond search: start by test large diamond pattern centered around block
    - If best match is interior, refine to finer scale
    - Else, recenter around best match

- **Early termination:** don’t find optimal reference patch, just find one that’s “good enough”: e.g., compressed representation is lower than threshold
  - Test zero-motion vector first (optimize for non-moving background)

- **Optimizations for subpixel motion vectors:**
  - Refinement: find best reference block given only pixel offsets, then try 1/2, 1/4-subpixel offsets around this match
H.265 (HVEC)

- Standard ratified in 2013
- Goal: ~2x better compression than H.264
- Main ideas: (more options, but similar in spirit to what we’ve discussed so far)
  - Macroblock sizes up to 64x64
  - Prediction block size and residual block sizes can be different
  - 35 intra-frame prediction modes (recall H.264 had 9)
  - ...

AV1

Main appeal may not be technical: royalty free codec, but many new options for encoders

AV1 Superblock Partitionings

- 56 angles for intraframe block prediction! (recall H.264 had nine!)
- Global transforms to geometrically warp previous frames to new frames
- Prediction of chroma channels from luma
- Synthetic generation of film-grain texture so that high-frequency film grain does not need to be compressed
Example: searching for best intra angles

Compute image gradients in block
Bin gradients to find most likely to be useful angles.
Only try the most likely angles.

https://www.slideshare.net/luctrudeau/i-dont-care-if-you-have-360-intra-directional-predictors
High cost of software encoders

- Statistic from Google: [Ranganathan 2021]
  - About 8-10 CPU minutes to compress 150 frames of 2160p H.264 video (4K video)
  - About 1 CPU hour for more expensive VP9 codec
Coarse-grained parallel video encoding

- Parallelized across segments (I-frame inserted at start of segment)
- Concatenate independently encoded bitstreams

Example: encoding an eight minute video

Task 1
(encode 0-2 min)

Task 2
(encode 2-4 min)

Task 3
(encode 4-6 min)

Task 4
(encode 6-8 min)

Task 5
concat

Smaller segments = more potential parallelism, worse video compression
Three types of encoding from Facebook Meta

Facebook / Instagram Posts

Messenger

Facebook Live / Messenger Live Video

Consider different tradeoffs: compression quality vs. latency in each of these cases
Fraction of energy consumed by different parts of instruction pipeline
(H.264 video encoding)

[Figure showing energy consumption by different parts of the instruction pipeline for H.264.]

- IME: Integer motion estimation
- FME: Fractional (subpixel) motion estimation
- IP: Intraframe prediction, DCT, quantization
- CABAC: Arithmetic encoding

Table 5 presents the number of fused operations created for each subgraphs to get performance and energy gains. FME required fusion of large subgraphs to get significant performance improvement.

Employing fused operations in combination with SIMD/VLIW doubling the energy/performance advantage of SIMD/VLIW.

Employing fused operations provides results in 3 instructions instead of 5 multiplications and two shits and adds. Operation fusion of multiplicand and adds the result. Multiplication is performed using additions. AddShft takes two inputs, multiplies both with the multiplicand and adds the result. Multiplication is performed using additions.

Hadamard transform. DCT transform also implement instructi
ons, though Intra and FME share instructions for the same parts of instruction pipeline.

Intra and FME share instructions for the same parts of instruction pipeline.

The required communication for the SIMD/VLIW algorithmic methods being optimized. These storage elements can be directly wired to custom designed multiple input and output functional units, directly implementing the required communication for the SIMD machine. To achieve this requires creating instructions that are tightly connected to supply the large amounts of data very closely tied to the specific algorithm.

Once this hardware is in place, the machine can issue “magic” instructions that can accomplish large amounts of computation at very low costs. This type of structure eliminates almost all the pipeline registers, busses, and clocking. Ctl is random control. RF = register file. FU = functional elements. Onl = on-line.

Stanford CS348K, Spring 2024
ASIC acceleration of video encode/decode
NVIDIA GPUs have video encode/decode ASICs

- Example: GeForce NOW game streaming service
- Rendered images compressed by GPU and directly streamed over network to remote player

- Another example: consumers at home streaming to Twitch
  - Do not want compression to take processing capability away from running the game itself.
Why do you think Google’s video sharing services (Youtube, Google photos, etc.) are willing to pay a high compute cost for compression?

- Reminder: statistic from Google:  [Ranganathan 2021]
  - About 8-10 CPU minutes to compress 150 frames of 2160p H.264 video (4K video)
  - About 1 CPU hour for more expensive VP9 codec

Video Usage Patterns at Scale: As with other internet media content [25], video popularity follows a stretched power law distribution, with three broad buckets. The first bucket – the very popular videos that make up the majority of watch time – represents a small fraction of transcoding and storage costs, worth spending extra processing time to reduce bandwidth to the user. The second bucket includes modestly watched videos which are served enough times to motivate a moderate amount of resources. And finally, the third bucket includes the long tail, the majority of videos that are watched infrequently enough that it makes sense to minimize storage and transcoding costs while maintaining playability. Note that old videos can increase in popularity and may need to be reprocessed with a higher popularity treatment well after upload.
When you upload a video it gets processed into many different output videos for serving

- **Different resolutions:**
  - For different viewing device types
  - For different network conditions

- **Different formats:**
  - Different devices might have video decode hardware that supports different formats (older devices might only support H.264, newer devices H.265, AV1 etc)

Note: it makes sense to amortize data loading (from storage) and data decoding costs over many output resolutions/formats
Google’s Video (Trans)coding Unit (VCU)

- ASIC hardware for decoding/encoding video in Google datacenter for Youtube/Youtube Live/etc.
- Consider load:
  - 500 hours of video uploaded to Youtube per minute (2019)
  - Must generate encoded versions assets at many resolutions and using different codecs to support streaming to consumers with many different devices and networks

<table>
<thead>
<tr>
<th>System</th>
<th>Throughput [Mpix/s]</th>
<th>Perf/TCO&lt;sup&gt;8&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H.264</td>
<td>VP9</td>
</tr>
<tr>
<td>Skylake</td>
<td>714</td>
<td>154</td>
</tr>
<tr>
<td>4xNvidia T4</td>
<td>2,484</td>
<td>—</td>
</tr>
<tr>
<td>8xVCU</td>
<td>5,973</td>
<td>6,122</td>
</tr>
<tr>
<td>20xVCU</td>
<td>14,932</td>
<td>15,306</td>
</tr>
</tbody>
</table>

[Ranganathan 2021]
Machine Learned Compression Schemes
Learned compression schemes

- H.264/265/AV1 video compression are “lossy” compression techniques that discard information that is present in the visual signal, but less likely to be noticed by the human eye
  - Key principle: “Lossy, but still looks good enough to humans!”

- Compression schemes described in this lecture so far involve manual choice / engineering of good representations (features)

- But machine learning is all about learning good representations from data.
  - Interest in learning highly compressed representations for a specific class of images/videos, or for a specific task to perform on images/videos
DNN autoencoder

If this latent representation is compact, then it is a compressed representation of the input image.

https://medium.com/@birla.deepak26/autoencoders-series-daad78df9350
Learned compression schemes

Many recent DNN-based approaches to compressing video learn to *compress the residual*

![Diagram of video compression pipeline](image)

[Tsai et al. 2018]

Use standard video compression at low quality, then use an autoencoder to compress the residual. *(Learn to compress the residual)*
Super-resolution-based reconstruction

Single image superresolution task: given a low-resolution image, predict the corresponding high-resolution image

bicubic (21.59dB/0.6423)  SRResNet (23.53dB/0.7832)  SRGAN (21.15dB/0.6868)  original

[SRGAN, Ledig et al. CVPR 2017]
Super resolution-based reconstruction

- Encode low-resolution video using standard video compression techniques
- Also transfer (as part of the video stream) a video-specific super-resolution DNN to upsample the low resolution video to high res video.
  - Assumption: training costs are amortized over many video downloads

[Yeo et al. OSDI 2018]
Neural volumes

- Learn to encode multiple views of a person into a latent code (z) that is decoded into a volume than can be rendered with conventional graphics techniques from any viewpoint.

- Motivated by VR applications.
Person specific compression

Input: video of professional ballerina performing a motion

Output: video of graduate student performing the same motion

Input video
Frame → Pose Detector DNN → 2D Pose → Pose-2-image DNN → Output video frame

Encode video as just a set of 14 pose joints.

[Chan et al. 2019]
NVIDIA Maxine

GPU-accelerated video processing for video conferencing applications

Examples: avatar control, video superresolution, advanced background segmentation
Where is the line between transmission of what happened and “making something up”?
The best camera is the one that's off?

Yes, you can make a Zoom background of yourself pretending to pay attention

And it's surprisingly easy to do, too.

Best funny Zoom background trick: Put yourself in a looping video so you can skip the meeting

Now you can duck out on those hourlong conference calls.

By Gordon Ung
Executive Editor, PCWorld | APR 13, 2020 3:30 AM PDT

We've all been in Zoom video conference meetings that drag on longer than a bad
Synthesizing reactions?

Input: audio of speaker
Output: video of listener’s reaction
User-triggered effects (examples: audio clips, “reactions”)
Project Starline: high fidelity

[Lawrence et al. 2021]

Fig. 4. Side-elevation view of our prototype system, illustrating the relative placement of the user, cameras, display, and virtual remote participant.
Summary

- H.264/265/AV1 video compression are “lossy” compression techniques that discard information that is present in the visual signal, but less likely to be noticed by the human eye
  - Key principle: “Lossy, but still looks good enough to humans!”

- Key idea of video encoding is “searching” for a compact encoding of the visual signal in a large space of possibilities
  - Video encoder ASIC used to accelerate this search

- Growing interest in learning these encodings, but it remains hard to beat well-engineered features
  - But promising if learned features are specialized to video stream contents
  - Or to specific tasks (remember, increasing amount of video is not meant to be consumed by human eyes)
Videoconferencing systems
As you can imagine, a lot of interest in video conferencing
Let’s design a video conferencing system
We want to deliver a visually rich experience similar to features of modern platforms
Deliver to wide range of clients and network settings
Let’s design a video conferencing system

Large gallery views: companies raced to provide 7x7 gallery in 2020 *

* it was quickly determined this was not particularly great
Setup...

Consider issues like latency...

Cloud

West Coast Servers

East Coast Servers

Personal computer

Icon credits: person by mim studio from the Noun Project, avatar by Soremba from the Noun Project
Q. Should we transcode/process video on our cloud servers?

■ What are advantages (to users? To the service provider)?
■ What are disadvantages?
Implementing gallery view

Cloud routes compressed video bitstreams to users (Does not manipulate bits)

Clients transmit individually compressed bitstreams

Receiving client “renders” all streams into appropriate display

Zoom calls this “multimedia routing”
One drawback of this design

- If each client is providing a single compressed video stream, that means each person on the video call must receive the same bits right? (What if they are on different network connections?)
Scalable video codec (SVC)

- "Scalable" compressed video bitstream: subsets of the bitstream encode valid video streams for a decoder
  - Implication: if packets get lost, the remaining packets form a valid H.264 bitstream, albeit at lower resolution or quality

Example: temporal scalability

Layer 0: \((T_0)\) defines valid video at frame rate \(R\)
Layer 1 \((T_1)\) defines bumps frame rate to \(2R\)

Note how layer 0 information is used to predict higher layer information
Scalable video codec (SVC)

Example: spatial scalability

Layer 0: defines valid video at low resolution (and low frame rate)
Layer 1: provides additional information for higher resolution (and higher frame rate) video
Scalable video codec (SVC) encoder

Costs: higher encoding/decoding costs
(But possible on modern clients as SVC is supported in hardware)