Video Industry

H.264 (AVC) Video

H.264, также известный как `AVC` (Advanced Video Coding), является стандартом видеокодирования, который был разработан совместной группой специалистов `ITU-T Video Coding Experts Group (VCEG)` и `ISO/IEC Moving Picture Experts Group (MPEG)`. Он предназначен для высокоэффективного кодирования видео с различным качеством и разрешением. H.264 использует множество сложных алгоритмов и методов для сжатия видео. Ниже приведены основные компоненты и методы, которые используются в H.264:

Основные алгоритмы и методы в H.264

1. **Блочное кодирование с переменным размером блока (Variable Block Size Coding)**
   - В H.264 видео разделяется на макроблоки размером 16x16 пикселей.
   - Эти макроблоки могут быть дополнительно разделены на меньшие блоки различных размеров (например, `16x8`, `8x16`, `8x8`, `4x4`) для лучшего соответствия локальных особенностей изображения.
2. **Интра- и интерпредсказание (Intra- and Inter-prediction)**
   - **Интрапредсказание:** Используется для кодирования кадров без ссылки на другие кадры (I-кадры). Интрапредсказание использует соседние пиксели в пределах того же кадра для предсказания текущего блока.
   - **Интерпредсказание:** Используется для кодирования кадров с ссылкой на предыдущие или будущие кадры (P- и B-кадры). Интерпредсказание использует блочное движение для предсказания текущего блока на основе информации из других кадров.
3. **Предсказание движения (Motion Estimation and Compensation)**
   - Блочное движение используется для предсказания изменения положения объектов между кадрами.
   - Каждый блок в текущем кадре сравнивается с блоками в одном или нескольких опорных кадрах для нахождения наилучшего соответствия.
   - Векторы движения (указывающие смещение блоков) кодируются и используются для восстановления блоков на стороне декодера.
4. **Трансформация и квантование (Transformation and Quantization)**

- Используется дискретное косинусное преобразование (DCT) для преобразования пространственной области (пикселей) в частотную область.
- Преобразованные коэффициенты затем квантируются для уменьшения точности и, соответственно, объема данных.

5. **Энтропийное кодирование (Entropy Coding)**
   - H.264 поддерживает два основных метода энтропийного кодирования:
     - **Кодирование с изменяющейся длиной (Variable Length Coding, `VLC`):** Использует кодирование с изменяющейся длиной для часто встречающихся символов.
     - **Контекстно-адаптивное побитовое арифметическое кодирование (Context-Adaptive Binary Arithmetic Coding, `CABAC`):** Более сложный и эффективный метод, который учитывает контекст символов для повышения эффективности сжатия.
6. **Адаптивная фильтрация (Deblocking Filter)**
   - H.264 включает в себя встроенный фильтр для уменьшения артефактов на границах блоков (blocking artifacts).
   - Фильтр используется для сглаживания границ между блоками, что улучшает визуальное качество видео.

Типы кадров в H.264

- **I-кадры (Intra-coded frames):** Кадры, которые кодируются без ссылки на другие кадры. Они служат точками восстановления, что важно для случайного доступа и восстановления после ошибок.
- **P-кадры (Predicted frames):** Кадры, которые кодируются с использованием предсказания движения из предыдущих кадров (опорные кадры).
- **B-кадры (Bi-predictive frames):** Кадры, которые кодируются с использованием предсказания движения как из предыдущих, так и из будущих кадров. Они обеспечивают наибольшую степень сжатия, но также требуют большей вычислительной мощности для кодирования и декодирования.

Пример работы H.264

1. **Разделение на макроблоки:** Входной видеопоток разделяется на макроблоки.
2. **Предсказание:** Каждый макроблок предсказывается с использованием интрапредсказания или интерпредсказания.

3. **Ошибка предсказания:** Вычисляется разница между предсказанным и реальным макроблоком.
4. **Трансформация и квантование:** Ошибка предсказания подвергается DCT и квантованию.
5. **Энтропийное кодирование:** Квантованные коэффициенты и векторы движения кодируются с использованием `VLC` или `CABAC`.
6. **Адаптивная фильтрация:** Декодированный макроблок подвергается фильтрации для удаления артефактов.
7. **Сборка кадра:** Кадры собираются из декодированных макроблоков и выводятся на дисплей.

H.265 (HEVC) Video

H.265, также известный как HEVC (High Efficiency Video Coding), является современным стандартом видеокодирования, разработанным для преодоления ограничений предыдущих стандартов, таких как `H.264/AVC. HEVC` значительно улучшает эффективность сжатия, что позволяет уменьшить битрейт на 50% при сохранении того же уровня качества видео по сравнению с H.264. Это достигается за счет использования множества сложных алгоритмов и методов. Ниже приведены основные компоненты и методы, используемые в H.265:

Основные алгоритмы и методы в H.265

1. **Кодирование с использованием блоков переменного размера (Coding Tree Units, `CTU`)**
   - В H.265 введены Coding Tree Units (`CTUs`), которые могут иметь размеры от `16x16` до `64x64` пикселей.
   - `CTU` могут быть дополнительно разделены на более мелкие блоки, называемые Coding Units (`CUs`), для лучшего соответствия локальных особенностей изображения.
2. **Интра- и интерпредсказание (Intra- and Inter-prediction)**
   - **Интрапредсказание:** H.265 использует 33 различных режима предсказания для блоков, что значительно больше, чем в H.264, обеспечивая более точное предсказание.
   - **Интерпредсказание:** Включает использование векторов движения с более высокой точностью и дополнительные опорные кадры для повышения эффективности предсказания.
3. **Предсказание движения (Motion Estimation and Compensation)**

- Векторы движения могут иметь субпиксельную точность (до 1/4 или 1/8 пикселя), что обеспечивает более точное предсказание движения.
- Вводится использование амплитудных компенсирующих векторов движения (Motion Vector Merging), что уменьшает количество необходимых данных для кодирования векторов движения.

4. **Трансформация и квантование (Transformation and Quantization)**
   - H.265 использует больше размеров блоков для трансформации (`4x4`, `8x8`, `16x16` и `32x32`), что позволяет лучше адаптироваться к различным структурам изображения.
   - Используется Discrete Cosine Transform (DCT) и Discrete Sine Transform (DST) для повышения эффективности сжатия.
5. **Энтропийное кодирование (Entropy Coding)**
   - H.265 использует усовершенствованное контекстно-адаптивное бинарное арифметическое кодирование (`CABAC`), которое улучшает эффективность сжатия по сравнению с H.264.
6. **Адаптивная фильтрация (In-loop Filtering)**
   - Включает два типа фильтров:
     - **`Deblocking Filter`:** Удаляет артефакты на границах блоков.
     - **Sample Adaptive Offset (`SAO`):** Корректирует ошибки квантования для улучшения качества изображения.
7. **Методы предсказания текстур (Intra Prediction)**
   - 33 различных направления предсказания для блоков, что позволяет более точно предсказывать текстурные особенности.

Типы кадров в H.265

- I-кадры (Intra-coded frames): Кадры, которые кодируются без ссылки на другие кадры. Они служат точками восстановления, что важно для случайного доступа и восстановления после ошибок.
- P-кадры (Predicted frames): Кадры, которые кодируются с использованием предсказания движения из предыдущих кадров (опорные кадры).
- B-кадры (Bi-predictive frames): Кадры, которые кодируются с использованием предсказания движения как из предыдущих, так и из будущих кадров. Они обеспечивают наибольшую степень сжатия, но также требуют большей вычислительной мощности для кодирования и декодирования.

Пример работы H.265

1. **Разделение на CTU:** Входной видеопоток разделяется на Coding Tree Units (CTUs).
2. **Предсказание:** Каждый CTU предсказывается с использованием интрапредсказания или интерпредсказания.
3. **Ошибка предсказания:** Вычисляется разница между предсказанным и реальным блоком.
4. **Трансформация и квантование:** Ошибка предсказания подвергается DCT или DST и квантованию.
5. **Энтропийное кодирование:** Квантованные коэффициенты и векторы движения кодируются с использованием CABAC.
6. **Адаптивная фильтрация:** Декодированный блок подвергается фильтрации для удаления артефактов.
7. **Сборка кадра:** Кадры собираются из декодированных блоков и выводятся на дисплей.

VP9 Video

`VP9` — это видеокодек, разработанный Google для замены `VP8` и конкуренции с `H.265/HEVC`. `VP9` обеспечивает высокую эффективность сжатия, улучшая качество видео при меньшем битрейте. Ниже приведены основные алгоритмы, методы и функции, которые используются в `VP9`:

Основные алгоритмы и методы в `VP9`

1. **Блочное кодирование с переменным размером блока (Variable Block Size Coding)**
   - В `VP9` видео разделяется на суперблоки размером `64x64` пикселей.
   - Эти суперблоки могут быть дополнительно разделены на более мелкие блоки (например, `32x32`, `16x16`, `8x8`) для лучшего соответствия локальных особенностей изображения.
   - Блоки могут иметь размеры вплоть до `4x4` пикселей, обеспечивая гибкость в кодировании сложных деталей изображения.
2. **Интра- и интерпредсказание (Intra- and Inter-prediction)**
   - **Интрапредсказание:** `VP9` использует 10 направлений для предсказания блоков внутри кадра (интрапредсказание), что позволяет лучше кодировать статичные сцены и детали.
   - **Интерпредсказание:** Используется для предсказания движения между кадрами. `VP9` поддерживает до 3 опорных кадров для интерпредсказания,

что позволяет улучшить качество предсказания движения.

3. **Предсказание движения (Motion Estimation and Compensation)**
    - Векторы движения используются для предсказания блоков в текущем кадре на основе информации из предыдущих и будущих кадров.
    - `VP9` поддерживает векторы движения с точностью до 1/8 пикселя, что позволяет более точно предсказывать движение.
    - Вводится использование сегментированных векторов движения и предсказания движения по нескольким направлениям для улучшения качества.
4. **Трансформация и квантование (Transformation and Quantization)**
    - `VP9` использует дискретное косинусное преобразование (DCT) и дискретное синусное преобразование (DST) для преобразования пространственной области в частотную.
    - Размеры блоков для трансформации могут быть `4x4`, `8x8`, `16x16` и `32x32`, что позволяет адаптироваться к различным структурам изображения.
    - Преобразованные коэффициенты затем квантируются для уменьшения объема данных.
5. **Энтропийное кодирование (Entropy Coding)**
    - `VP9` использует контекстно-адаптивное двоичное арифметическое кодирование (Context-Adaptive Binary Arithmetic Coding, `CABAC`) для повышения эффективности сжатия.
    - Также используется специальное кодирование с изменяющейся длиной (Variable Length Coding, `VLC`) для кодирования оставшихся символов.
6. **Адаптивная фильтрация (Loop Filtering)**
    - `VP9` включает несколько этапов фильтрации для улучшения качества изображения:
        - **`Deblocking Filter`:** Уменьшает артефакты на границах блоков.
        - **Adaptive Loop Filter (`ALF`):** Применяется для коррекции ошибок квантования и улучшения визуального качества.
        - **Sharpening Filter:** Применяется для повышения резкости изображения.
7. **Поддержка многопоточного кодирования (`Multithreaded Encoding`)**
    - `VP9` оптимизирован для многопоточного кодирования, что позволяет эффективно использовать многоядерные процессоры для ускорения кодирования видео.
8. **Интеллектуальная сегментация (Segmentation)**

- `VP9` позволяет сегментировать кадр на различные регионы, к которым могут применяться разные параметры квантования и фильтрации. Это позволяет более гибко управлять качеством и битрейтом.

Типы кадров в `VP9`

- Key Frames (Ключевые кадры): Кадры, которые кодируются независимо от других кадров и служат точками восстановления.
- Inter Frames (Промежуточные кадры): Кадры, которые кодируются с использованием информации из предыдущих или будущих кадров (опорные кадры).

Пример работы VP9

1. **Разделение на суперблоки:** Входной видеопоток разделяется на суперблоки размером 64x64 пикселей.
2. **Предсказание:** Каждый суперблок предсказывается с использованием интрапредсказания или интерпредсказания.
3. **Ошибка предсказания:** Вычисляется разница между предсказанным и реальным суперблоком.
4. **Трансформация и квантование:** Ошибка предсказания подвергается DCT или DST и квантованию.
5. **Энтропийное кодирование:** Квантованные коэффициенты и векторы движения кодируются с использованием CABAC или VLC.
6. **Адаптивная фильтрация:** Декодированный суперблок подвергается фильтрации для удаления артефактов.
7. **Сборка кадра:** Кадры собираются из декодированных суперблоков и выводятся на дисплей.

Заключение

`VP9` является современным и высокоэффективным видеокодеком, который значительно улучшает качество сжатия по сравнению с предыдущими стандартами. Благодаря множеству усовершенствованных методов и алгоритмов, VP9 достигает высокой степени сжатия без значительных потерь качества, что делает его идеальным для современных приложений, таких как потоковое видео, видеоконференции и записи видео.

AV1 Video

`AV1` (`AOMedia` Video 1) — это современный видеокодек с открытым исходным кодом, разработанный Альянсом за открытые медиа (`AOMedia`). Он предназначен для обеспечения высокой эффективности сжатия видео, улучшения качества изображения и уменьшения битрейта по сравнению с существующими стандартами, такими как `VP9` и H.265/HEVC. `AV1` включает множество инновационных алгоритмов и методов для достижения этих целей. Ниже приведено подробное описание основных алгоритмов, методов и функций, которые используются в `AV1`.

Основные алгоритмы и методы в `AV1`

1. **Адаптивное деление на блоки (Adaptive Block Partitioning)**
   - `AV1` поддерживает сложное деление блоков, начиная с суперблоков размером `128x128` пикселей.
   - Суперблоки могут быть рекурсивно разделены на более мелкие блоки, вплоть до `4x4` пикселей, что позволяет эффективно кодировать как однородные области, так и области с мелкими деталями.
   - Поддерживаются различные формы разделения, включая горизонтальное и вертикальное деление, а также нестандартные формы, такие как T-образные и другие.
2. **Интрапредсказание (Intra Prediction)**
   - `AV1` поддерживает 56 режимов интрапредсказания, что значительно больше, чем в предыдущих кодеках, таких как H.265.
   - Эти режимы включают традиционные направления (горизонтальное, вертикальное и диагональное), а также более сложные предсказания на основе окружающих пикселей.
3. **Интерпредсказание (Inter Prediction)**
   - Поддерживает до 7 опорных кадров, что позволяет улучшить точность предсказания и уменьшить количество бит.
   - Векторы движения могут иметь точность до 1/8 пикселя, что позволяет более точно предсказывать движение.
   - Поддерживает многофреймовое предсказание, bi-prediction и использование глобального движения.
4. **Смешанные режимы предсказания (Compound Prediction Modes)**
   - `AV1` включает различные комбинированные режимы предсказания, такие как среднее комбинированное предсказание (Average Compound Prediction) и взвешенное комбинированное предсказание (Weighted Compound Prediction).

5. **Трансформации и квантование (Transformations and Quantization)**
    - Поддерживает различные размеры трансформационных блоков, включая `4x4`, `8x8`, `16x16`, `32x32` и `64x64`.
    - Использует дискретное косинусное преобразование (DCT) и дискретное синусное преобразование (DST), а также новые методы, такие как flip-adaptive transform (`FLIPADST`) и состоящие из двух частей трансформации (EXT_TX).
    - Квантование адаптивно настраивается для каждого блока, учитывая его важность для качества изображения.
6. **Энтропийное кодирование (Entropy Coding)**
    - Использует контекстно-адаптивное бинарное арифметическое кодирование (Context-Adaptive Binary Arithmetic Coding, `CABAC`), что обеспечивает высокую степень сжатия.
    - Улучшенные алгоритмы кодирования, такие как Multi-symbol Entropy Coding, повышают эффективность энтропийного кодирования.
7. **Адаптивная фильтрация (In-loop Filtering)**
    - Включает несколько этапов фильтрации для улучшения качества изображения:
        - **`Deblocking Filter`:** Уменьшает артефакты на границах блоков.
        - **`Constrained Directional Enhancement Filter (CDEF)`:** Применяется для сглаживания и улучшения качества изображения.
        - **`Loop Restoration Filter`:** Использует алгоритмы восстановления, такие как Wiener Filter и Self-Guided Restoration, для уменьшения шума и улучшения визуального качества.
8. **Поддержка высокого динамического диапазона (High Dynamic Range, `HDR`)**
    - `AV1` поддерживает кодирование видео с высоким динамическим диапазоном, обеспечивая лучшее представление светлых и темных областей изображения.
    - Поддержка различных цветовых пространств и глубины цвета до 12 бит на канал.
9. **Многопоточное кодирование (`Multithreaded Encoding`)**
    - `AV1` оптимизирован для многопоточного кодирования, что позволяет эффективно использовать многоядерные процессоры и ускорять процесс кодирования видео.
10. **Субпиксельное предсказание (`Subpixel Prediction`)**

- Использует различные методы интерполяции для предсказания субпиксельного движения, что позволяет улучшить точность и качество предсказания движения.

Пример работы `AV1`

1. **Разделение на суперблоки:** Входной видеопоток разделяется на суперблоки размером до `128x128` пикселей.
2. **Адаптивное деление блоков:** Каждый суперблок рекурсивно делится на более мелкие блоки с учетом сложности изображения.
3. **Предсказание:** Выполняется интрапредсказание для статичных блоков и интерпредсказание для блоков с движением.
4. **Трансформация и квантование:** Ошибки предсказания подвергаются DCT, DST или другим трансформациям и квантованию.
5. **Энтропийное кодирование:** Квантованные коэффициенты и векторы движения кодируются с использованием CABAC и других методов.
6. **Адаптивная фильтрация:** Применяются deblocking filter, CDEF и loop restoration filter для улучшения качества изображения.
7. **Сборка кадра:** Кадры собираются из декодированных блоков и выводятся на дисплей.

Заключение
`AV1` представляет собой современный и высокоэффективный видеокодек, который значительно улучшает эффективность сжатия и качество изображения по сравнению с предыдущими стандартами. Это достигается за счет использования множества усовершенствованных алгоритмов и методов, таких как адаптивное деление блоков, улучшенное предсказание движения, сложные трансформации и квантование, а также мощные методы энтропийного кодирования и адаптивной фильтрации. `AV1` подходит для широкого спектра приложений, от потокового видео и видеоконференций до записи видео с высоким разрешением и высоким динамическим диапазоном.

H.264 VS H.265

`H.264 (AVC)` и `H.265 (HEVC)` – это два широко используемых стандарта видеокодирования, которые разработаны для сжатия видеофайлов, уменьшая их размер, сохраняя при этом качество изображения. H.265 был разработан как улучшение H.264, и он включает множество усовершенствований и новых функций

для повышения эффективности сжатия. Вот подробное сравнение и отличия между H.264 и H.265:

Основные отличия между H.264 и H.265

1. **Эффективность сжатия**

- `H.264 (AVC)`:
  - Кодек H.264 использует методы блочного кодирования, предсказания движения и трансформации для сжатия видео.
  - Средняя степень сжатия H.264 позволяет достичь хорошего качества изображения при разумных размерах файлов, но для высоких разрешений требуется значительный битрейт.
- `H.265 (HEVC)`:
  - H.265 улучшает методы блочного кодирования и предсказания движения, позволяя достичь того же уровня качества изображения при примерно половинном битрейте по сравнению с H.264.
  - Это достигается за счет более сложных алгоритмов и улучшенной архитектуры кодирования.

2. **Структура блоков**

- `H.264`:
  - Использует макроблоки размером `16x16` пикселей, которые могут быть разделены на более мелкие блоки.
  - Поддерживает блоки размера `4x4`, `8x8` и `16x16` для адаптации к различным деталям изображения.
- `H.265`:
  - Вводит Coding Tree Units (`CTU`), которые могут иметь размеры до `64x64` пикселей, обеспечивая большую гибкость и эффективность.
  - `CTU` могут быть разделены на Coding Units (CU), Prediction Units (PU) и Transform Units (TU) различных размеров (например, `64x64`, `32x32`, `16x16` и `8x8`).
  - Это позволяет более точно адаптироваться к сложным деталям изображения и снижать избыточность данных.

3. **Предсказание и компенсация движения**

- `H.264`:
  - Поддерживает до 16 опорных кадров для предсказания движения.
  - Предсказание движения основано на векторах движения с точностью до 1/4 пикселя.
  - Использует прямое и двустороннее предсказание для улучшения качества.
- `H.265`:
  - Поддерживает до 32 опорных кадров для предсказания движения, что позволяет улучшить точность и качество предсказания.
  - Векторы движения имеют точность до 1/4 и 1/8 пикселя.
  - Вводит методы амплитудной компенсации движения (AMP) и временного фильтра (Temporal Motion Vector Prediction) для повышения эффективности.

4. **Трансформация и квантование**

- `H.264`:
  - Использует дискретное косинусное преобразование (DCT) для трансформации блоков.
  - Размеры блоков для трансформации фиксированы: 4x4 и 8x8.
- `H.265`:
  - Поддерживает как DCT, так и дискретное синусное преобразование (DST) для лучшего соответствия различным типам данных.
  - Размеры блоков для трансформации могут быть 4x4, 8x8, 16x16 и 32x32, что обеспечивает большую гибкость и эффективность.

5. **Энтропийное кодирование**

- `H.264`:
  - Использует контекстно-адаптивное двоичное арифметическое кодирование (`CABAC`) и контекстно-адаптивное кодирование с изменяющейся длиной (`CAVLC`) для энтропийного кодирования.
- `H.265`:
  - Улучшает `CABAC`, делая его более эффективным за счет увеличения числа контекстных моделей и улучшенного кодирования.

6. **Адаптивная фильтрация**

- `H.264`:

    - Включает фильтр для уменьшения артефактов на границах блоков (`deblocking filter`).
- `H.265`:
    - Включает улучшенный `deblocking filter` и добавляет новый фильтр Sample Adaptive Offset (`SAO`), который корректирует ошибки квантования и улучшает качество изображения.

7. **Поддержка разрешений и частоты кадров**

- `H.264`:
    - Поддерживает разрешения до `4K UHD 3840x2160` и частоты кадров до 60 кадров в секунду.
- `H.265`:
    - Поддерживает разрешения до `8K UHD (8192x4320`) и частоты кадров до 120 кадров в секунду, что делает его более подходящим для будущих применений с высоким разрешением и частотой кадров.

8. **Применение и совместимость**

- `H.264`:
    - Широко используется в потоковом видео, видеоконференциях, видеонаблюдении и других областях.
    - Обеспечивает высокую совместимость с различными устройствами и платформами.
- `H.265`:
    - Используется в тех же областях, что и H.264, но предоставляет улучшенные возможности для приложений с высоким разрешением и эффективностью сжатия.
    - Совместимость с устройствами постепенно улучшается, но требует поддержки аппаратного декодирования для достижения наилучших результатов.

Заключение
H.265 представляет собой значительное улучшение по сравнению с H.264, предлагая более высокую степень сжатия при сохранении или даже улучшении качества изображения. Это достигается за счет использования более сложных и эффективных алгоритмов и методов кодирования, включая более гибкую структуру блоков, улучшенное предсказание движения, расширенные методы трансформации

и квантования, а также более эффективное энтропийное кодирование и адаптивную фильтрацию. Эти усовершенствования делают H.265 более подходящим для современных и будущих приложений, требующих высокого разрешения и высокой эффективности сжатия.

H.265 VS VP9

`VP9` и `H.265 (HEVC)` – это два современных стандарта видеокодирования, разработанных для повышения эффективности сжатия видео и уменьшения битрейта при сохранении высокого качества. `VP9` был разработан Google как замена `VP8` и конкурент `H.265`, который был разработан совместными усилиями `ISO/IEC Moving Picture Experts Group (MPEG)` и `ITU-T Video Coding Experts Group` (`VCEG`). Ниже приведено подробное сравнение и отличия между `VP9` и `H.265`.

Основные отличия между `VP9` и `H.265`

1. **Эффективность сжатия**

- `H.265 (HEVC)`:
  - HEVC предназначен для сжатия видео с высоким качеством при меньшем битрейте по сравнению с H.264. Он достигает до 50% более высокой эффективности сжатия по сравнению с H.264.
  - HEVC использует сложные алгоритмы предсказания, трансформации и энтропийного кодирования для повышения эффективности.
- `VP9`:
  - `VP9` также улучшает эффективность сжатия по сравнению с H.264, достигая до 50% более высокой эффективности.
  - `VP9` использует аналогичные, но несколько различающиеся подходы к предсказанию, трансформации и энтропийному кодированию.

2. **Структура блоков**

- `H.265`:
  - Вводит Coding Tree Units (`CTU`) размером до `64x64` пикселей.
  - `CTU` могут быть разделены на Coding Units (CU), Prediction Units (PU) и Transform Units (TU) различных размеров (например, `64x64`, `32x32`, `16x16` и `8x8`).
- `VP9`:
  - Использует суперблоки размером до `64x64` пикселей.

- Суперблоки могут быть разделены на меньшие блоки различных размеров (например, `32x32`, `16x16`, `8x8` и вплоть до `4x4`), что позволяет более точно адаптироваться к сложным деталям изображения.

3. **Предсказание и компенсация движения**

- `H.265`:
  - Поддерживает до 32 опорных кадров для предсказания движения.
  - Векторы движения имеют точность до 1/4 и 1/8 пикселя.
  - Вводит методы амплитудной компенсации движения (AMP) и временного фильтра (Temporal Motion Vector Prediction).
- `VP9`:
  - Поддерживает до 3 опорных кадров для предсказания движения.
  - Векторы движения имеют точность до 1/4 пикселя.
  - Использует многопоточное предсказание движения и сегментацию векторов движения для улучшения качества.

4. **Трансформация и квантование**

- `H.265`:
  - Поддерживает размеры блоков для трансформации `4x4`, `8x8`, `16x16` и `32x32`.
  - Использует как дискретное косинусное преобразование (DCT), так и дискретное синусное преобразование (DST) для повышения эффективности.
- `VP9`:
  - Поддерживает размеры блоков для трансформации `4x4`, `8x8`, `16x16` и `32x32`.
  - Использует только DCT для трансформации.

5. **Энтропийное кодирование**

- `H.265`:
  - Использует улучшенное контекстно-адаптивное двоичное арифметическое кодирование (`CABAC`), что обеспечивает высокую эффективность сжатия.
- `VP9`:
  - Использует контекстно-адаптивное бинарное арифметическое кодирование (Context-Adaptive Binary Arithmetic Coding, `CABAC`) и

контекстно-адаптивное кодирование с изменяющейся длиной (Variable Length Coding, `VLC`).

6. **Адаптивная фильтрация**

- `H.265`:
  - Включает улучшенный фильтр для уменьшения артефактов на границах блоков (`deblocking filter`).
  - Включает фильтр Sample Adaptive Offset (`SAO`) для улучшения качества изображения.
- `VP9`:
  - Включает многоступенчатую фильтрацию, включая `deblocking filter` и восстановление границ (Loop Restoration).

7. **Поддержка разрешений и частоты кадров**

- `H.265:`
  - Поддерживает разрешения до `8K UHD (8192x4320)` и частоты кадров до 120 кадров в секунду.
- `VP9`:
  - Поддерживает разрешения до `8K UHD (8192x4320)` и частоты кадров до 120 кадров в секунду.

8. **Лицензирование и роялти**

- `H.265`:
  - Лицензирование H.265 требует уплаты роялти, что может быть препятствием для его широкого принятия.
  - Управляется организациями MPEG LA и HEVC Advance.
- `VP9`:
  - VP9 является открытым и бесплатным для использования, не требует уплаты роялти.
  - Продвигается и поддерживается Google.

Заключение
H.265 и **VP9** оба представляют собой современные и эффективные стандарты видеокодирования, которые значительно превосходят по эффективности сжатия своих предшественников. Однако они отличаются рядом ключевых аспектов:

- **Эффективность сжатия:** Оба кодека предлагают схожую степень сжатия, но используют разные алгоритмы и методы для достижения этой цели.
- **Структура блоков:** Оба кодека используют гибкие структуры блоков для повышения эффективности, но реализация этих структур различается.
- **Предсказание движения:** `H.265` предлагает более сложные методы предсказания движения и поддержку большего числа опорных кадров, в то время как `VP9` использует менее сложные, но всё ещё эффективные методы.
- **Энтропийное кодирование:** `H.265` использует усовершенствованное `CABAC`, тогда как `VP9` использует смесь `CABAC` и `VLC`.
- **Лицензирование:** `VP9` имеет преимущество в плане отсутствия роялти, что делает его привлекательным для разработчиков и компаний, желающих избежать дополнительных затрат.

Выбор между `H.265` и `VP9` будет зависеть от конкретных требований проекта, включая эффективность сжатия, совместимость с устройствами, лицензирование и роялти.

H.265 VS AV1

Сравнение видео-кодеков `H.265` и `AV1`

1. Основные технологии и методы, используемые в кодеках

`H.265 (HEVC - High Efficiency Video Coding)`:

- **Intra Prediction**: Улучшенное предсказание в пределах кадра для снижения избыточности.
- **Inter Prediction**: Использование сложных методов предсказания между кадрами для повышения эффективности сжатия.
- **Transform and Quantization**: Применение DCT (Discrete Cosine Transform) и DST (Discrete Sine Transform) для преобразования пространственной информации в частотную область, с последующей квантованием для сжатия данных.
- **Loop Filters**: Использование фильтров для улучшения качества видео после декодирования.
- **Entropy Coding**: Использование методов CABAC (Context-Adaptive Binary Arithmetic Coding) для эффективного кодирования битов.

`AV1 (AOMedia Video 1)`:

- **Intra Prediction**: Более гибкие и сложные методы предсказания в пределах кадра.
- **Inter Prediction**: Улучшенные методы межкадрового предсказания, включая использование специальных векторов движения и сложных моделей предсказания.
- **Transform and Quantization**: Применение множества различных трансформаций и квантовок, включая KLT (Karhunen-Loeve Transform) для более эффективного сжатия.
- **Loop Filters**: Более сложные и адаптивные методы фильтрации для улучшения качества видео.
- **Entropy Coding**: Использование методов ANS (Asymmetric Numeral Systems) для более эффективного сжатия данных.

2. Статистика по скорости, производительности и иным метрикам

Скорость кодирования и декодирования:

- **H.265**: Обычно имеет более высокую скорость кодирования по сравнению с AV1, но требует значительных вычислительных ресурсов для декодирования.
- **AV1**: Более медленная скорость кодирования из-за сложных алгоритмов, но также требует значительных вычислительных ресурсов для декодирования. Однако, с развитием аппаратных декодеров, скорость воспроизведения AV1 улучшается.

Эффективность сжатия:

- **H.265**: Эффективнее сжимает видео по сравнению с предыдущими стандартами (H.264), обеспечивая примерно 50% экономии битрейта при сохранении того же качества.
- **AV1**: Обеспечивает ещё более высокую эффективность сжатия по сравнению с H.265, обеспечивая примерно на 30% меньший битрейт при том же уровне качества.

Качество видео:

- **H.265**: Обеспечивает высокое качество видео, особенно при высоких разрешениях (4K и выше).
- **AV1**: Обеспечивает ещё более высокое качество видео при более низких битрейтах, что особенно важно для потоковой передачи видео через интернет.

Анализ метрик:

- **VMAF (Video Multi-Method Assessment Fusion)**: Метрика, используемая для оценки восприятия качества видео.
  - **H.265**: Получает высокие оценки VMAF при низких и средних битрейтах.
  - **AV1**: Получает ещё более высокие оценки VMAF, особенно при низких битрейтах.

3. Применение кодеков в различных сценариях

H.265 (HEVC):

- **Трансляция и хранение видео**: Используется для трансляции видео в высоком разрешении, включая 4K и 8K, благодаря своей эффективности сжатия.
- **Съемка и производство видео**: Часто используется в профессиональном оборудовании для съемки видео высокого качества.
- **Потоковые сервисы**: Широко поддерживается на большинстве современных устройств и платформ.

AV1:

- **Потоковая передача через интернет**: Идеально подходит для потоковой передачи видео благодаря высокой эффективности сжатия, что позволяет экономить пропускную способность.
- **Мобильные устройства**: Сниженный битрейт при высоком качестве делает его отличным выбором для мобильных сетей с ограниченной пропускной способностью.
- **Поддержка открытых стандартов**: AV1 является бесплатным и открытым стандартом, что делает его привлекательным для внедрения на широком спектре устройств и платформ.

Заключение
Выбор между H.265 и AV1 зависит от конкретных требований и условий использования. H.265 лучше подходит для приложений, где важна поддержка на широком спектре устройств и высокая скорость кодирования. AV1 предпочтительнее для потоковой передачи видео в интернете, где важна высокая эффективность сжатия и минимальные затраты на пропускную способность.

Quantization

Что такое Quantization (Квантование) в кодировании аудио и видео?
Квантование — это процесс, используемый в кодировании аудио и видео для уменьшения объема данных за счет снижения точности представления сигналов. Основная цель квантования — преобразовать непрерывные (аналоговые) сигналы в дискретные (цифровые) с ограниченным числом уровней, что позволяет уменьшить объем данных и упростить дальнейшую обработку и хранение.

Основная идея
Основная идея квантования заключается в округлении значений сигнала до ближайших дискретных уровней. Это приводит к потере некоторой точности, но позволяет значительно сократить объем данных. Видеокодирование и аудиокодирование используют квантование для уменьшения количества информации, передаваемой или сохраняемой, при сохранении приемлемого уровня качества.

Как работает квантование?

1. **Разделение сигнала:** Сначала исходный непрерывный сигнал делится на небольшие сегменты или блоки (например, блоки пикселей в видео или сегменты звуковых волн в аудио).
2. **Применение преобразования:** Применяется математическое преобразование, например, дискретное косинусное преобразование (DCT) для видео или дискретное преобразование Фурье (`DFT`) для аудио, чтобы преобразовать данные в частотный домен.
3. **Округление значений:** Полученные значения округляются до ближайших дискретных уровней, что снижает точность, но уменьшает объем данных.
4. **Кодирование:** Квантованные значения кодируются с использованием методов энтропийного кодирования для дальнейшего уменьшения объема данных.

Пример на практике
Видео
Рассмотрим процесс квантования на примере видеокодирования:

1. **Разделение на блоки:** Изображение делится на блоки, например, размером 8x8 пикселей.
2. **Применение DCT:** Для каждого блока применяется дискретное косинусное преобразование (DCT), которое преобразует значения пикселей в частотный домен.

3. **Квантование:** Полученные частотные коэффициенты округляются до ближайших дискретных значений. Часто используются матрицы квантования, чтобы уменьшить точность высокочастотных компонентов, которые менее заметны для глаза.
4. **Кодирование:** Квантованные коэффициенты кодируются с использованием методов энтропийного кодирования (например, кодирования Хаффмана).

Пример матрицы квантования для блока `8x8`:

```
[16, 11, 10, 16, 24, 40, 51, 61]
[12, 12, 14, 19, 26, 58, 60, 55]
[14, 13, 16, 24, 40, 57, 69, 56]
[14, 17, 22, 29, 51, 87, 80, 62]
[18, 22, 37, 56, 68, 109, 103, 77]
[24, 35, 55, 64, 81, 104, 113, 92]
[49, 64, 78, 87, 103, 121, 120, 101]
[72, 92, 95, 98, 112, 100, 103, 99]
```

Каждый элемент матрицы квантования используется для деления соответствующего коэффициента DCT, уменьшая тем самым точность высокочастотных коэффициентов больше, чем низкочастотных.

Аудио
Рассмотрим процесс квантования на примере аудиокодирования:

1. **Разделение на сегменты:** Звуковой сигнал делится на небольшие временные сегменты.
2. **Применение `DFT`:** Для каждого сегмента применяется дискретное преобразование Фурье (DFT), которое преобразует сигнал в частотный домен.
3. **Квантование:** Полученные частотные коэффициенты округляются до ближайших дискретных значений. Часто используется неравномерное квантование, чтобы уменьшить влияние на слух.
4. **Кодирование:** Квантованные коэффициенты кодируются с использованием методов энтропийного кодирования (например, кодирования Хаффмана).

Преимущества квантования

1. **Сжатие данных:** Квантование позволяет значительно уменьшить объем данных, что особенно важно для хранения и передачи аудио- и видеоконтента.
2. **Простота реализации:** Процесс квантования прост в реализации и легко интегрируется в различные схемы сжатия.
3. **Эффективность:** Квантование, особенно с использованием матриц квантования, позволяет оптимизировать баланс между качеством и степенью сжатия.

Недостатки квантования

1. **Потери качества:** Поскольку квантование приводит к потере точности, это может повлиять на качество изображения или звука. Визуальные артефакты или искажения звука могут быть заметны при сильном сжатии.
2. **Чувствительность к параметрам:** Эффективность и качество квантования сильно зависят от выбранных параметров, таких как матрицы квантования или шаг квантования.

Применение квантования
Квантование используется в различных форматах аудио- и видеокодирования, таких как:

- **JPEG:** Для сжатия изображений.
- **MPEG, H.264, HEVC:** Для сжатия видео.
- **MP3, AAC:** Для сжатия аудио.

Пример
Рассмотрим пример квантования для блока 8x8 пикселей в изображении. Допустим, после применения DCT мы получили следующие коэффициенты:

```
[231, -32,  8, -4, -2, -1,  0,  0]
[-30,  15, -3,  2,  1,  1,  0,  0]
[ 7,  -3,  1, -1, -1,  0,  0,  0]
[-4,   2, -1,  0,  0,  0,  0,  0]
[-2,   1, -1,  0,  0,  0,  0,  0]
[-1,   1,  0,  0,  0,  0,  0,  0]
[ 0,   0,  0,  0,  0,  0,  0,  0]
[ 0,   0,  0,  0,  0,  0,  0,  0]
```

Используя матрицу квантования, делим каждый элемент на соответствующее значение матрицы и округляем результат:

```
[231/16, -32/11,  8/10, -4/16, -2/24, -1/40,  0/51,  0/61]
[-30/12,  15/12, -3/14,  2/19,  1/26,  1/58,  0/60,  0/55]
[ 7/14,  -3/13,  1/16, -1/24, -1/40,  0/57,  0/69,  0/56]
[ -4/14,   2/17, -1/22,  0/29,  0/51,  0/87,  0/80,  0/62]
[ -2/18,   1/22, -1/37,  0/56,  0/68,  0/109,  0/103,  0/77]
[ -1/24,   1/35,  0/55,  0/64,  0/81,  0/104,  0/113,  0/92]
[  0/49,   0/64,  0/78,  0/87,  0/103,  0/121,  0/120,  0/101]
[  0/72,   0/92,  0/95,  0/98,  0/112,  0/100,  0/103,  0/99]
```

Округленные значения:

```
[14, -3,  1,  0,  0,  0,  0,  0]
[-2,  1,  0,  0,  0,  0,  0,  0]
[ 1,  0,  0,  0,  0,  0,  0,  0]
[ 0,  0,  0,  0,  0,  0,  0,  0]
[ 0,  0,  0,  0,  0,  0,  0,  0]
[ 0,  0,  0,  0,  0,  0,  0,  0]
[ 0,  0,  0,  0,  0,  0,  0,  0]
[ 0,  0,  0,  0,  0,  0,  0,  0]
```

Run Length Coding

Что такое Run-Length Encoding (RLE)?
Run-Length Encoding (RLE) — это простой метод сжатия данных, который используется для сокращения объема данных, представляющих последовательности повторяющихся символов. Этот метод особенно эффективен для данных, содержащих длинные серии одинаковых символов, таких как изображения с большими областями одного цвета.

Основная идея
Основная идея RLE заключается в замене последовательностей повторяющихся символов (называемых "runs") одним символом и числом, которое указывает, сколько раз этот символ повторяется.

Как работает RLE?
RLE работает путем последовательного чтения данных и записи каждого символа вместе с количеством его повторений. Давайте рассмотрим пример для наглядности.

Пример
Представьте, что у вас есть строка данных:

```
AAAAABBBBCCCCCCCCDDDDDDDDDD
```

Вместо того чтобы хранить каждый символ отдельно, мы можем закодировать эту строку с использованием RLE:

```
A5B4C8D10
```

Это означает:

- A встречается 5 раз подряд
- B встречается 4 раза подряд
- C встречается 8 раз подряд
- D встречается 10 раз подряд

Преимущества RLE

1. **Простота:** RLE очень легко реализовать и понять.
2. **Эффективность для данных с длинными последовательностями:** Если данные содержат много длинных последовательностей одинаковых символов, RLE может значительно сократить объем данных.

Недостатки RLE

1. **Неэффективность для данных без длинных последовательностей:** Если данные содержат мало повторяющихся символов, RLE может даже увеличить размер данных.
2. **Ограниченность применения:** RLE не подходит для данных с высокой энтропией, где символы часто меняются.

Применение RLE
RLE широко используется в различных областях, таких как:

- **Графика и изображения:** Форматы изображений, такие как `BMP` и `TIFF`, часто используют RLE для сжатия данных.
- **Телевидение:** Аналоговые и цифровые телевизионные сигналы могут использовать RLE для уменьшения объема данных.
- **Данные в текстовом формате:** RLE может быть использовано для сжатия текстов, особенно если они содержат много повторяющихся символов или пробелов.

Пример на практике
Рассмотрим простое черно-белое изображение размером `5x5` пикселей, где "1" — это белый пиксель, а "0" — черный:

```
11111
10001
10001
10001
11111
```

Закодируем это изображение с использованием RLE

- Первая строка: 1 встречается 5 раз → 15
- Вторая строка: 1, 0 встречается 3 раза, 1 → 1031
- Третья строка: 1, 0 встречается 3 раза, 1 → 1031
- Четвертая строка: 1, 0 встречается 3 раза, 1 → 1031
- Пятая строка: 1 встречается 5 раз → 15

Закодированное изображение:

```
15, 1031, 1031, 1031, 15
```

Декодирование RLE
Для декодирования строки, закодированной с использованием RLE, необходимо просто распаковать каждую пару символов и чисел.

Пример
Закодированная строка:

```
A5B4C8D10
```

Процесс декодирования:

- A5 → AAAAA
- B4 → BBBB
- C8 → CCCCCCCC
- D10 → DDDDDDDDDD

Декодированная строка:

```
AAAAABBBBCCCCCCCCDDDDDDDDDD
```

Lempel-Ziv-Welch (LZW)

Что такое `Lempel-Ziv-Welch (LZW)`?
`Lempel-Ziv-Welch (LZW)` — это алгоритм сжатия данных без потерь, который используется для уменьшения размера данных путем замены повторяющихся последовательностей символов на более короткие коды. LZW был разработан Абрахамом Лемпелем, Джейкобом Зивом и Терри Уэлчем в 1984 году и стал основой для многих популярных форматов сжатия данных, таких как GIF и TIFF.

Основная идея
Основная идея LZW заключается в использовании динамически обновляемого словаря для замены последовательностей символов на коды фиксированной длины. По мере обработки данных алгоритм строит словарь, который содержит пары "последовательность символов — код". По мере того как алгоритм встречает новые последовательности, он добавляет их в словарь и использует коды для замены этих последовательностей.

Пошаговый процесс LZW

1. Инициализация словаря
   Сначала инициализируется словарь, содержащий все возможные символы входных данных. Например, для текстовых данных словарь может содержать все символы ASCII (256 символов).
2. Чтение входных данных
   Алгоритм последовательно читает входные данные и строит словарь по мере нахождения новых последовательностей символов.

3. Обработка последовательностей

- **Поиск:** Алгоритм ищет самую длинную последовательность символов, которая уже существует в словаре.
- **Кодирование:** Когда такая последовательность найдена, она заменяется соответствующим кодом из словаря.
- **Добавление:** Алгоритм добавляет новую последовательность (текущая последовательность + следующий символ) в словарь с новым кодом.

Пример

Рассмотрим пример сжатия строки `ABABABA`:

1. **Инициализация словаря:**

```
A -> 1
B -> 2
```

2. **Чтение входных данных:**
   - Текущая последовательность: "A"
   - Следующий символ: "B"
3. **Обработка последовательностей:**
   - Найдено "A" в словаре -> код "1"
   - Добавляем "AB" в словарь с новым кодом:

```
A -> 1
B -> 2
AB -> 3
```

4. **Продолжение:**
   - Следующая последовательность: "B"
   - Следующий символ: "A"
   - Найдено "B" в словаре -> код "2"
   - Добавляем "BA" в словарь:

```
A -> 1
B -> 2
```

```
AB -> 3
BA -> 4
```

- Следующая последовательность: "AB"
- Найдено "AB" в словаре -> код "3"
- Добавляем "ABA" в словарь:

```
A -> 1
B -> 2
AB -> 3
BA -> 4
ABA -> 5
```

- Следующая последовательность: "A"
- Найдено "A" в словаре -> код "1"
- Добавляем "AA" в словарь:
  `A -> 1 B -> 2 AB -> 3 BA -> 4 ABA -> 5 AA -> 6`
  Закодированная строка: "1, 2, 3, 1"

Преимущества LZW

1. **Эффективность сжатия:** LZW может значительно уменьшить объем данных, особенно для данных с множеством повторяющихся последовательностей.
2. **Универсальность:** Алгоритм может применяться к различным типам данных, включая текст, изображения и звуковые файлы.
3. **Отсутствие потерь:** Сжатие и восстановление данных происходят без потерь информации.

Недостатки LZW

1. **Ограниченность при малых объемах данных:** Для очень коротких данных или данных с высокой энтропией алгоритм может быть неэффективен.
2. **Зависимость от размера словаря:** По мере роста словаря алгоритм может требовать больше памяти для хранения всех последовательностей.

Применение LZW

- **Форматы изображений:** Один из наиболее известных форматов, использующих LZW, — это формат GIF, который широко применяется для сжатия растровых изображений.
- **Форматы файлов:** LZW также используется в форматах файлов TIFF и PDF для сжатия текстовых и графических данных.

Variable Block Size Coding

Что такое Variable Block Size Coding (Кодирование с переменным размером блока)?
Variable Block Size Coding (VBSC), или кодирование с переменным размером блока, — это метод, используемый в видеокодировании для улучшения эффективности сжатия и качества видео. В отличие от методов, использующих фиксированный размер блока, VBSC позволяет динамически изменять размер блоков пикселей в зависимости от содержания изображения.

Основная идея
Основная идея VBSC заключается в том, чтобы использовать небольшие блоки для сложных, детализированных областей изображения и более крупные блоки для однородных, простых областей. Это позволяет более точно моделировать движение и текстуры, улучшая качество сжатия.

Как работает VBSC?

1. **Разбиение изображения на блоки:** Исходное изображение делится на блоки пикселей. Размеры блоков могут варьироваться, например, `4x4`, `8x8`, `16x16` и так далее.
2. **Анализ содержания блока:** Каждому блоку присваивается размер в зависимости от его сложности. Сложные области, содержащие много деталей или движения, получают более мелкие блоки. Однородные области получают более крупные блоки.
3. **Кодирование блоков:** Каждый блок кодируется отдельно с использованием методов предсказания движения и энтропийного кодирования.

Преимущества VBSC

1. **Улучшенное качество изображения:** Более точное моделирование движений и текстур в сложных областях улучшает визуальное восприятие.

2. **Эффективное сжатие:** Использование более крупных блоков в однородных областях позволяет снизить объем данных, подлежащих сжатию.
3. **Гибкость:** Возможность выбора размера блока для каждой области изображения позволяет адаптироваться к различным типам видеоконтента.

Технические детали
VBSC реализуется в нескольких этапах:

1. **Анализ содержания изображения:**
   - Вычисляется вариативность и сложность каждой области изображения.
   - Определяется оптимальный размер блока для каждой области.
2. **Предсказание движения:**
   - Выполняется поиск движения для каждого блока, используя предыдущие и последующие кадры.
   - Для каждого блока вычисляются векторы движения.
3. **Кодирование:**
   - Каждому блоку присваивается уникальный код, который включает информацию о его размере, векторах движения и содержимом.
   - Используются методы энтропийного кодирования для дальнейшего сжатия данных.

Пример на практике
Рассмотрим пример видеокадра, содержащего две области:

1. **Область с высоким уровнем детализации:** Например, лицо человека с множеством мелких деталей.
2. **Однородная область:** Например, небо или стена с однородным цветом.

Шаги VBSC:

1. **Анализ:**
   - Область с высоким уровнем детализации будет разбита на мелкие блоки (например, 4x4 пикселя), чтобы более точно моделировать детали лица.
   - Однородная область будет разбита на крупные блоки (например, 16x16 пикселей), поскольку в этой области меньше изменений и деталей.
2. **Предсказание движения:**
   - Для мелких блоков лица будет выполнен точный поиск движения, чтобы точно отследить изменения в выражении лица.

- Для крупных блоков неба будет выполнен более общий поиск движения, поскольку изменения в этой области менее значительны.

3. **Кодирование:**
   - Мелкие блоки лица будут кодироваться с высокой точностью, включая информацию о каждом пикселе и его движении.
   - Крупные блоки неба будут кодироваться с низкой точностью, что позволяет значительно уменьшить объем данных.

Применение VBSC
VBSC используется в различных форматах видеокодирования и сжатия, таких как:

- **H.264/AVC:** Один из наиболее распространенных стандартов видеокодирования, который использует VBSC для улучшения качества видео и эффективности сжатия.
- **HEVC (H.265):** Более новый стандарт, который также использует VBSC для достижения еще большей эффективности сжатия и улучшения качества видео.

Intra-coded Frames

Что такое Intra-coded frames (I-кадры)?
Intra-coded frames (I-кадры), также известные как ключевые кадры, — это тип кадра в видео, который кодируется независимо от других кадров. Они содержат полную информацию о изображении и не зависят от данных других кадров для своего декодирования.

Зачем нужны I-кадры?
I-кадры играют ключевую роль в видеокодировании и сжатии, обеспечивая точки восстановления, где декодер может начать декодирование без необходимости смотреть предыдущие кадры. Это важно для:

1. **Редактирования:** Позвляет легко нарезать и монтировать видео, поскольку I-кадры содержат всю информацию.
2. **Потоковой передачи:** Обеспечивает возможность начать воспроизведение видео с любого места.
3. **Стабильности:** Помогает восстановиться после ошибок передачи данных.

Как работают I-кадры?
I-кадры кодируются без учета информации из других кадров. Это похоже на то, как бы вы сохранили отдельное изображение в формате JPEG. Каждый пиксель

кодируется, основываясь только на его собственных данных и данных окружающих пикселей в этом же кадре.

Технические детали

1. **Компрессия:** I-кадры используют методы компрессии без потерь или с минимальными потерями, такие как дискретное косинусное преобразование (DCT), чтобы уменьшить объем данных, сохраняя высокое качество изображения.
2. **Кодирование:** В кодировании I-кадров используются методы энтропийного кодирования (например, Хаффман или арифметическое кодирование) для сжатия данных, основываясь на частоте появления символов.
3. **Размер:** I-кадры обычно занимают больше места по сравнению с другими типами кадров (например, P-кадрами и B-кадрами), потому что они содержат полную информацию о кадре.

Сравнение с другими типами кадров

- **P-кадры (Predictive-coded frames):** Эти кадры кодируются на основе предыдущих I-кадров или других P-кадров, используя информацию о движении. Они содержат разницу (остатки) между текущим кадром и предсказанным изображением.
- **B-кадры (Bi-directional predictive-coded frames):** Эти кадры кодируются на основе как предыдущих, так и последующих кадров (I или P), что позволяет достичь более высокой степени сжатия.

Пример на практике
Представьте, что у вас есть видео, состоящее из последовательности кадров:

1. **Кадр 1:** I-кадр, содержит полную информацию о сцене.
2. **Кадр 2:** P-кадр, содержит информацию о разнице между кадром 1 и кадром 2.
3. **Кадр 3:** P-кадр, содержит информацию о разнице между кадром 2 и кадром 3.
4. **Кадр 4:** I-кадр, содержит полную информацию о новой сцене.
5. **Кадр 5:** P-кадр, содержит информацию о разнице между кадром 4 и кадром 5.

Если мы начнем воспроизведение с кадра 4, нам не нужно знать, что было в кадре 1, 2 или 3, потому что кадр 4 (I-кадр) содержит полную информацию для своего декодирования.

Почему I-кадры важны?

1. **Редактирование и нарезка:** Видео можно легко резать и монтировать, начиная с I-кадров, так как они содержат полную информацию.
2. **Начало воспроизведения:** При потоковой передаче видео пользователь может начать просмотр с любого I-кадра, не дожидаясь загрузки всех предыдущих кадров.
3. **Ошибка восстановления:** В случае ошибки в передаче данных I-кадры позволяют восстановить изображение без необходимости перезапускать все предыдущее видео.

Predicate Frames

Что такое Predicted Frames (P-кадры)?
Predicted frames, или P-кадры, — это тип кадров в видеокодировании, которые кодируются на основе предыдущих кадров. Они содержат информацию о разнице между текущим кадром и предсказанным изображением, что позволяет существенно уменьшить объем данных по сравнению с I-кадрами.

Основная идея
P-кадры используют информацию о движении и изменении сцены между кадрами, чтобы предсказать текущий кадр. Вместо того чтобы хранить полную информацию о каждом кадре, как это делается в I-кадрах, P-кадры хранят только разницу (остатки) между текущим кадром и предсказанным изображением на основе предыдущих кадров.

Как работают P-кадры?

1. **Поиск движения:** Видеокодер анализирует текущий кадр и предыдущий кадр, чтобы определить, как объекты перемещаются между кадрами. Это называется поиском движения.
2. **Векторы движения:** Результаты поиска движения записываются в виде векторов движения, которые указывают, как каждый блок пикселей в текущем кадре сдвигается по сравнению с предыдущим кадром.
3. **Предсказание кадра:** На основе векторов движения создается предсказанное изображение текущего кадра.
4. **Вычисление остатков:** Остатки (разница между предсказанным и фактическим кадром) также кодируются и сохраняются.

Технические детали

1. **Компрессия:** P-кадры обычно занимают меньше места, чем I-кадры, потому что они кодируют только разницу между кадрами, а не всю информацию о кадре.
2. **Кодирование:** Остатки и векторы движения могут быть дополнительно сжаты с использованием методов энтропийного кодирования (например, Хаффман или арифметическое кодирование).

Пример на практике
Представьте, что у вас есть видео, где мяч движется справа налево по экрану:

1. **Кадр 1 (I-кадр):** Полная информация о сцене, мяч в правом углу.
2. **Кадр 2 (P-кадр):** Предсказывается на основе кадра 1, мяч сдвигается немного влево. Вектор движения указывает, что мяч сдвинулся на 10 пикселей влево.
3. **Кадр 3 (P-кадр):** Предсказывается на основе кадра 2, мяч продолжает двигаться влево. Вектор движения указывает еще на 10 пикселей влево.

Вместо того чтобы хранить всю информацию о каждом кадре, P-кадры хранят только векторы движения и разницу между предсказанным и фактическим положением мяча.

Bi Predicate Frames

Что такое Bi-Predicted Frames (B-кадры)?
Bi-Predicted frames, или B-кадры, — это тип кадров в видеокодировании, которые используют информацию как из предыдущих, так и из последующих кадров для кодирования. Это позволяет достичь более высокой степени сжатия по сравнению с I- и P-кадрами.

Основная идея
B-кадры кодируются на основе двух или более кадров: одного или нескольких предыдущих и одного или нескольких последующих. Они используют информацию о движении объектов между этими кадрами для предсказания текущего кадра.

Как работают B-кадры?

1. **Поиск движения:** Видеокодер анализирует текущий кадр, предыдущие кадры и последующие кадры, чтобы определить движение объектов между кадрами.

2. **Векторы движения:** Результаты поиска движения записываются в виде векторов движения, которые указывают, как каждый блок пикселей в текущем кадре сдвигается по сравнению с предыдущими и последующими кадрами.
3. **Предсказание кадра:** На основе векторов движения создается предсказанное изображение текущего кадра, используя информацию из предыдущих и последующих кадров.
4. **Вычисление остатков:** Остатки (разница между предсказанным и фактическим кадром) также кодируются и сохраняются.

Пример на практике
Представьте, что у вас есть видео, где мяч движется слева направо по экрану:

1. **Кадр 1 (I-кадр):** Полная информация о сцене, мяч в левом углу.
2. **Кадр 2 (B-кадр):** Предсказывается на основе кадров 1 и 3, мяч немного сдвинулся вправо.
3. **Кадр 3 (P-кадр):** Предсказывается на основе кадра 1, мяч еще дальше вправо.

B-кадр 2 будет использовать информацию из обоих I-кадра 1 и P-кадра 3 для точного предсказания положения мяча.

Технические детали

1. **Компрессия:** B-кадры обычно обеспечивают более высокую степень сжатия, чем P-кадры, так как используют информацию из большего количества кадров.
2. **Кодирование:** Остатки и векторы движения могут быть дополнительно сжаты с использованием методов энтропийного кодирования (например, Хаффман или арифметическое кодирование).
3. **Размещение:** B-кадры не могут быть начальной точкой для декодирования, так как они зависят от других кадров. Поэтому они используются между I- и P-кадрами для повышения эффективности сжатия.

Discrete Cosine Transform (DCT)

Discrete Cosine Transform (DCT) — это важная математическая техника, широко используемая в обработке изображений и видео, а также в сжатии данных. Чтобы понять, что это такое и как она работает, давайте рассмотрим её поэтапно с использованием аналогий.

Что такое DCT?
Представьте, что у вас есть сложный рисунок на листе бумаги. Если вы хотите сохранить его в компьютере, вам нужно найти способ представить этот рисунок в виде чисел. Одним из способов сделать это является использование DCT.

DCT преобразует ваши данные (например, пиксели изображения) из пространственного домена (где у вас есть значения яркости пикселей) в частотный домен (где данные представлены как сумма косинусоидальных функций с различными частотами). Проще говоря, DCT разбивает изображение на базовые элементы, которые можно комбинировать для восстановления исходного изображения.

Аналогия с музыкой
Подумайте о DCT как о чем-то вроде музыкального преобразования. Представьте, что у вас есть музыкальное произведение, и вы хотите записать его на бумаге. Вы можете записать ноты каждой отдельной ноты, но это будет длинный и громоздкий список. Вместо этого, вы можете записать аккорды, которые состоят из нескольких нот, играемых одновременно. Аккорды — это нечто вроде частотных компонентов в музыке. Они представляют более сложные звуки с помощью простых комбинаций.

Точно так же DCT берет ваше изображение и разбивает его на «аккорды» — базовые косинусные функции с разными частотами. Эти «аккорды» легче хранить и обрабатывать.

Технические детали
DCT можно описать математически. Формула для одного измерения DCT для последовательности длиной (N) выглядит так:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right]$$

где:

- (X_k) — это коэффициент DCT для частоты (k).
- (x_n) — это исходное значение на позиции (n) (например, яркость пикселя).
- (N) — общее количество значений в последовательности.
  Этот процесс преобразует данные из временного (или пространственного) домена в частотный домен.

DCT в 2D

Когда мы говорим об изображениях, нам нужно работать с двумя измерениями, поэтому мы используем двухмерную версию DCT (2D DCT). Она просто выполняет DCT по каждому измерению (строке и столбцу изображения):

$$X_{k1,k2} = \sum_{n1=0}^{N1-1} \sum_{n2=0}^{N2-1} x_{n1,n2} \cdot \cos\left[\frac{\pi}{N1}\left(n1 + \frac{1}{2}\right)k1\right] \cdot \cos\left[\frac{\pi}{N2}\left(n2 + \frac{1}{2}\right)k2\right]$$

где:

- $(X_{k1,k2})$ — это коэффициент DCT для частот $(k1)$ и $(k2)$.
- $(x_{n1,n2})$ — это исходное значение (например, яркость пикселя) в позиции $((n1, n2))$.
- $(N1)$ и $(N2)$ — размеры изображения по каждому из измерений.

Применение DCT

Наиболее известное применение DCT — это сжатие изображений и видео, например, в формате JPEG. В JPEG изображение делится на маленькие блоки (обычно 8x8 пикселей), и для каждого блока вычисляется 2D DCT. Большинство коэффициентов DCT оказываются близкими к нулю и могут быть отброшены без значительной потери качества изображения, что приводит к сжатию данных.

Преимущества DCT

1. **Эффективное сжатие:** DCT позволяет сильно сжимать данные, сохраняя при этом достаточное качество изображения.
2. **Уменьшение объема данных:** Многие коэффициенты DCT оказываются малы и могут быть отброшены, что уменьшает объем хранимых данных.
3. **Быстрое вычисление:** Существуют эффективные алгоритмы для вычисления DCT, что делает её практически применимой для обработки изображений и видео.

Пример на практике

Допустим, у вас есть чёрно-белое изображение размером 8x8 пикселей. Вы можете представить это изображение как матрицу чисел, где каждое число — это яркость пикселя. DCT преобразует эту матрицу в другую матрицу, где каждый элемент представляет вклад определенной частоты в исходное изображение. Большие числа в этой новой матрице показывают, что соответствующая частота играет важную роль в формировании исходного изображения, а маленькие числа можно игнорировать для сжатия.

Пространственный и частотный домен

Пространственный домен

Пространственный домен — это способ представления данных (например, изображения) в том виде, как мы их обычно видим. Представьте себе обычное изображение. Оно состоит из пикселей, каждый из которых имеет свою яркость и цвет. Если мы говорим о черно-белом изображении, каждый пиксель имеет значение яркости, которое может быть представлено числом.

Изображение как матрица чисел

Черно-белое изображение можно представить как матрицу чисел. Давайте рассмотрим небольшой пример:

```
[  52,  55,  61,  66,  70,  61,  64,  73 ]
[  63,  59,  55,  90, 109,  85,  69,  72 ]
[  62,  59,  68, 113, 144, 104,  66,  73 ]
[  63,  58,  71, 122, 154, 106,  70,  69 ]
[  67,  61,  68, 104, 126,  88,  68,  70 ]
[  79,  65,  60,  70,  77,  68,  58,  75 ]
[  85,  71,  64,  59,  55,  61,  65,  83 ]
[  87,  79,  69,  68,  65,  76,  78,  94 ]
```

Каждое число в этой матрице представляет яркость одного пикселя. Например, значение 52 означает темный пиксель, а 144 — светлый пиксель.

Частотный домен

Частотный домен — это другой способ представления данных. Вместо того чтобы смотреть на пиксели по отдельности, мы смотрим на их общие колебания и изменения. Чтобы понять это, давайте рассмотрим аналогию с музыкой.

Аналогия с музыкой

Представьте себе музыкальную ноту. Вы можете записать ее как амплитуду звуковой волны во времени (пространственный домен). Но вы также можете представить ее как частоту (количество колебаний в секунду). Так, например, нота "ля" (A) имеет частоту 440 Гц.

Когда мы преобразуем изображение из пространственного домена в частотный домен с помощью DCT, мы представляем его как сумму косинусоидальных

функций (волн) с разными частотами. Это позволяет нам увидеть, какие частоты (или детали) важны для изображения.

Простая иллюстрация
Представьте себе черно-белую шахматную доску. В пространственном домене она представлена чередующимися черными и белыми квадратами. В частотном домене она будет представлена высокочастотными компонентами, так как у нее много резких изменений от черного к белому.

Зачем это нужно?
Преобразование в частотный домен полезно, потому что часто оказывается, что большинство важной информации сосредоточено в нескольких низкочастотных компонентах, а высокочастотные компоненты можно отбросить без значительной потери качества. Это позволяет сжимать данные, уменьшая объем, но сохраняя важную информацию.

Пример
Представьте, что у вас есть изображение, которое вы хотите сжать. В пространственном домене это изображение выглядит как множество чисел, каждое из которых представляет яркость пикселя. При преобразовании в частотный домен вы получаете другую матрицу, где значения показывают вклад различных частот. Большинство этих значений могут быть близки к нулю, и их можно отбросить, уменьшив размер данных.

Подведем итоги

1. **Пространственный домен:** Представление изображения в виде пикселей и их яркости.
2. **Частотный домен:** Представление изображения как суммы косинусоидальных функций разной частоты, что позволяет видеть, какие частоты (детали) важны.

Motion Compensation (MC)

Зачем нужна компенсация движения?
Когда вы смотрите видео, вы видите последовательность изображений (кадров), которые быстро сменяют друг друга, создавая иллюзию движения. Однако, в видео часто много повторяющейся информации. Например, если камера медленно панорамирует по сцене, большая часть изображения остаётся почти такой же из кадра в кадр, только немного сдвигается.

Хранить каждую такую картинку целиком — это расточительно, поскольку можно было бы использовать информацию из предыдущих кадров. Здесь и приходит на помощь компенсация движения.

Основная идея компенсации движения
Компенсация движения позволяет нам использовать информацию из предыдущих кадров для предсказания текущего кадра, уменьшая объем данных, который нужно хранить. Это делается путём отслеживания движущихся объектов между кадрами и использования этой информации для прогнозирования.

Аналогия с комиксом
Представьте себе, что вы создаете комикс, где один персонаж медленно ходит по комнате. Вместо того чтобы рисовать каждую картинку с нуля, вы можете просто копировать предыдущее изображение и немного перемещать персонажа. Это позволяет вам экономить время и усилия, а также уменьшает объем работы. Точно так же в видео компрессии мы можем копировать блоки пикселей из предыдущих кадров и слегка сдвигать их, чтобы они соответствовали текущему кадру.

Как это работает?

1. **Разбиение на блоки:** Текущий кадр разбивается на маленькие блоки (обычно 16x16 пикселей).
2. **Поиск движения:** Для каждого блока в текущем кадре ищется наиболее похожий блок в предыдущем кадре. Это называется **поиском движения**.
3. **Вектор движения:** Если найдено соответствие, мы фиксируем сдвиг (или вектор движения), который показывает, как блок переместился.
4. **Предсказание:** Используя векторы движения, мы можем предсказать текущий кадр, копируя блоки из предыдущих кадров и перемещая их согласно вектору движения.
5. **Остатки:** Иногда предсказание не идеально, и остаются небольшие ошибки. Эти ошибки называются остатками, и они также сохраняются для восстановления точного изображения.

Пример на практике
Давайте рассмотрим простой пример. Представьте, что у вас есть видео, где мяч движется слева направо по экрану.

1. **Первый кадр:** Вы видите мяч в левом углу.
2. **Второй кадр:** Мяч немного сдвинулся вправо.

3. **Третий кадр:** Мяч еще дальше вправо.

Вместо того чтобы хранить три отдельных изображения, мы можем хранить первый кадр полностью, а затем использовать вектор движения, чтобы описать, как мяч переместился из кадра в кадр. Например:

- Вектор движения для второго кадра: мяч сдвинулся на 10 пикселей вправо.
- Вектор движения для третьего кадра: мяч сдвинулся еще на 10 пикселей вправо.

Преимущества компенсации движения

1. **Сжатие:** Значительно уменьшает объем данных, которые нужно хранить и передавать.
2. **Эффективность:** Улучшает качество видео при той же степени сжатия, так как позволяет лучше использовать информацию из предыдущих кадров.
3. **Реализация:** Широко используется в современных форматах видео, таких как MPEG и H.264.

Остатки в Motion Compensation

Что такое ошибки предсказания (остатки)?
Когда мы используем компенсацию движения, мы предсказываем текущий кадр на основе предыдущих кадров, перемещая блоки пикселей в соответствии с векторами движения. Однако, это предсказание не всегда идеально. Разница между реальным значением пикселя и предсказанным значением называется ошибкой предсказания или остатком.

Почему возникают остатки?
Остатки возникают по нескольким причинам:

1. **Сложное движение:** Иногда движение объектов в кадре может быть слишком сложным или нестандартным для точного предсказания.
2. **Освещение и текстуры:** Изменения в освещении или текстуре объекта могут сделать предсказание менее точным.
3. **Неточность поиска движения:** Алгоритмы поиска движения не всегда могут найти идеальное соответствие блоков между кадрами.

Влияние остатков на качество видео
Аналогия с рисованием

Представьте, что вы рисуете комикс, где персонаж двигается из кадра в кадр. Вы используете предыдущие рисунки, чтобы ускорить процесс, копируя и перемещая части рисунка. Но иногда вы замечаете, что части рисунка не совпадают идеально — например, рука персонажа немного сместилась или изменилось выражение лица. Чтобы исправить это, вы вручную дорисовываете нужные детали. Остатки в видео — это как эти доработки, которые исправляют неточности.

Как остатки учитываются в сжатии видео?

1. **Сохранение остатков:** В процессе сжатия видео, вместе с векторами движения, мы также сохраняем остатки. Эти остатки позволяют точно восстановить изображение при декодировании.
2. **Использование остатков:** Во время декодирования, предсказанный кадр комбинируется с остатками, чтобы получить точное изображение. Это гарантирует, что все мелкие детали и изменения, которые не были учтены в предсказании, будут восстановлены.

Пример
Давайте рассмотрим пример с движущимся мячом:

1. **Первый кадр:** Мяч в левом углу.
2. **Второй кадр:** Мяч сдвинулся вправо, но освещение изменилось, и тень появилась под другим углом.
   При предсказании второго кадра на основе первого мы можем точно определить положение мяча, но из-за изменения освещения предсказание будет немного неточным. Разница в яркости и тенях между предсказанным и реальным кадром будет сохранена как остатки.

Важность остатков

1. **Точность:** Остатки обеспечивают точное восстановление изображения, сохраняя все мелкие детали, которые важны для визуального восприятия.
2. **Качество:** Без остатков предсказание было бы менее точным, что привело бы к ухудшению качества видео, появлению артефактов и искажений.
3. **Эффективность:** Хранение остатков вместе с векторами движения позволяет значительно сжать данные, но при этом сохранить высокое качество изображения.

Entropy Coding

Что такое Entropy Coding (Кодирование по энтропии)?
Entropy Coding, или кодирование по энтропии, — это метод сжатия данных, который используется для уменьшения объема информации без потерь. Это означает, что данные могут быть восстановлены точно в исходном виде после сжатия. Энтропийное кодирование основано на вероятности появления данных и кодировании более вероятных данных более короткими кодами.

Почему это важно?
Давайте рассмотрим простую аналогию. Представьте, что вы работаете в офисе и часто используете одни и те же фразы в своих письмах. Вместо того чтобы каждый раз писать эти фразы полностью, вы можете создать сокращения для них. Например, "Best regards" можно заменить на "BR". Так, наиболее часто используемые фразы занимают меньше места. В энтропийном кодировании мы делаем что-то похожее, но с числами и символами.

Основные типы энтропийного кодирования

1. **Кодирование Хаффмана (Huffman Coding)**
2. **Арифметическое кодирование (Arithmetic Coding)**

Huffman Coding

Что такое Huffman Coding (Кодирование Хаффмана)?
Кодирование Хаффмана — это метод сжатия данных без потерь, основанный на вероятности появления символов. Этот метод позволяет кодировать символы более короткими кодами, если они встречаются чаще, и более длинными кодами, если они встречаются реже.

Основная идея
Идея заключается в том, чтобы минимизировать среднюю длину кодируемого сообщения. Для этого используется дерево Хаффмана, которое строится на основе частот символов в исходных данных.

Пошаговый процесс кодирования Хаффмана

1. Подсчет частот символов
   Сначала необходимо подсчитать, сколько раз каждый символ встречается в данных. Это поможет определить, какие символы более часты и должны получать более короткие коды.

Пример:
Представим строку `AAABBC`. Подсчитаем частоты:

- A: 3 раза
- B: 2 раза
- C: 1 раз

2. Создание узлов дерева
   Каждый символ и его частота представляются в виде узлов дерева. В начале у нас есть отдельные узлы для каждого символа.

Начальные узлы:

- A (частота 3)
- B (частота 2)
- C (частота 1)

3. Построение дерева Хаффмана
   Следующим шагом является построение дерева Хаффмана. Это делается путем последовательного слияния двух узлов с наименьшей частотой до тех пор, пока не останется один узел — корень дерева.
   Шаги:
4. **Слияние B и C:** Создаем новый узел с частотой 3 (2 + 1).

```
B (2)
 \
  (3)
 /
C (1)
```

5. **Слияние нового узла с A:** Создаем новый узел с частотой 6 (3 + 3).

```
    (6)
   /   \
  A (3) (3)
       / \
      B   C
```

Теперь у нас есть дерево Хаффмана.

6. Назначение кодов
   На основе дерева Хаффмана мы можем назначить бинарные коды каждому символу. Начинаем с корня и движемся вниз по дереву:

- Каждое левое ребро обозначаем "0".
- Каждое правое ребро обозначаем "1".

Назначение кодов:

- A: 0
- B: 10
- C: 11

Пример кодирования
Теперь мы можем закодировать нашу строку `AAABBC` используя полученные коды:

- A: 0
- B: 10
- C: 11
  Строка `AAABBC` закодируется как "00010110".

Преимущества кодирования Хаффмана

1. **Эффективность:** Позволяет значительно сократить объем данных, особенно если некоторые символы встречаются чаще других.
2. **Оптимальность:** Гарантирует минимальную среднюю длину кодов для данных с известными частотами символов.
3. **Без потерь:** Кодирование и декодирование происходят без потерь информации.

Декодирование
Декодирование строки, закодированной по Хаффману, происходит путем чтения битов и движения по дереву от корня до листа. Когда мы достигаем листа, это означает, что мы декодировали один символ.

Пример:
Закодированная строка "00010110":

- 0 -> A
- 0 -> A
- 0 -> A
- 10 -> B
- 11 -> C

Изначальная строка: `AAABBC`.

Arithmetic Coding

Что такое Arithmetic Coding (Арифметическое кодирование)?
Арифметическое кодирование — это метод сжатия данных без потерь, который представляет всю строку символов как одно число в интервале от 0 до 1. В отличие от других методов, таких как кодирование Хаффмана, где каждому символу присваивается фиксированный бинарный код, арифметическое кодирование использует дробные числа для представления всей строки.

Основная идея
Идея арифметического кодирования заключается в последовательном сужении интервала $[0, 1)$ по мере обработки каждого символа. Чем чаще встречается символ, тем меньше сужается интервал для него. В итоге весь набор данных представляется одним числом из этого интервала.

Пошаговый процесс арифметического кодирования

1. Подсчет частот символов
   Для начала подсчитываем частоту появления каждого символа в данных. Эти частоты затем используются для построения интервалов.

Пример:
Строка `AAABBC`. Подсчитаем частоты:

- A: 3 раза
- B: 2 раза
- C: 1 раз

Общая длина строки — 6 символов, поэтому вероятности:

- A: 3/6 = 0.5
- B: 2/6 = 0.333

- C: 1/6 = 0.167

2. Построение интервалов
   На основе вероятностей строим интервалы для каждого символа:

- A: $[0, 0.5)$
- B: $[0.5, 0.833)$
- C: $[0.833, 1)$

3. Сужение интервала
   Начинаем с интервала $[0, 1)$ и последовательно сужаем его для каждого символа в строке.

Пример:
Строка `AAABBC`. Начальный интервал — $[0, 1)$.

1. **Первый символ "A":** Сужаем интервал до $[0, 0.5)$.
2. **Второй символ "A":** Сужаем текущий интервал $[0, 0.5)$ до его подинтервала для "A": $[0, 0.25)$.
3. **Третий символ "A":** Сужаем интервал $[0, 0.25)$ до $[0, 0.125)$.
4. **Четвертый символ "B":** Сужаем интервал $[0, 0.125)$ до его подинтервала для "B": $[0.0625, 0.09375)$.
5. **Пятый символ "B":** Сужаем интервал $[0.0625, 0.09375)$ до $[0.078125, 0.0859375)$.
6. **Шестой символ "C":** Сужаем интервал $[0.078125, 0.0859375)$ до $[0.08203125, 0.0859375)$.

В итоге получаем интервал $[0.08203125, 0.0859375)$. Любое число в этом интервале может представлять исходную строку.

4. Представление результата
   Для кодирования выбираем любое число в полученном интервале. Например, можно выбрать середину интервала:

$$\text{Закодированное число} = \frac{0.08203125 + 0.0859375}{2} = 0.083984375$$

Декодирование
Для декодирования числа 0.083984375, которое находится в интервале

$[0.08203125, 0.0859375)$:

1. **Первый символ:** Определяем, что число попадает в интервал $[0, 0.5)$ -> "A".
2. **Второй символ:** Уточняем интервал для второго символа, учитывая, что первый символ "A" -> $[0, 0.25)$.
3. **Третий символ:** Продолжаем сужать интервал, пока не восстановим всю строку.

Преимущества арифметического кодирования

1. **Высокая эффективность:** Позволяет достичь близкой к теоретическому пределу сжатия, особенно для данных с высоко неоднородными частотами символов.
2. **Гибкость:** Не ограничено целым числом битов для каждого символа, как в кодировании Хаффмана, что делает его более адаптивным.

Multi-symbol Entropy Coding

Что такое Multi-symbol Entropy Coding (Кодирование энтропии с несколькими символами)?
Multi-symbol Entropy Coding (MSEC), или кодирование энтропии с несколькими символами, — это метод сжатия данных, который использует вероятности появления различных последовательностей символов (не отдельных символов) для более эффективного кодирования. В отличие от традиционного кодирования энтропии, где каждый символ кодируется отдельно, MSEC кодирует целые последовательности символов, что позволяет достичь более высокой степени сжатия за счет использования более длинных и, следовательно, более информативных символов.

Основная идея
Основная идея MSEC заключается в использовании длинных последовательностей символов вместо отдельных символов для кодирования. Это позволяет более эффективно использовать статистические свойства данных, поскольку последовательности символов часто имеют более предсказуемые и повторяющиеся паттерны, чем отдельные символы.

Как работает MSEC?

1. **Анализ частот последовательностей:** Определяются частоты появления различных последовательностей символов в исходных данных.

2. **Построение кодов:** На основе частот построенных последовательностей создаются коды переменной длины. Более часто встречающиеся последовательности получают более короткие коды.
3. **Кодирование данных:** Исходные данные заменяются соответствующими кодами последовательностей.

Пример на практике
Представим, что у нас есть текстовая строка:

```
ABABABAB
```

Для кодирования отдельных символов нам потребуются следующие коды (например, используя кодирование Хаффмана):

- A: 0
- B: 1

Закодированная строка: `01010101`
Однако, если мы используем `MSEC` и кодируем пары символов, мы можем достичь лучшего сжатия.

Анализ частот пар символов
Рассмотрим пары символов:

- AB: 4 раза

На основе частот мы можем создать код для пары AB:

- AB: 0

Закодированная строка будет выглядеть так:

```
ABABABAB -> 0000
```

Мы видим, что вместо 8 битов (при кодировании каждого символа отдельно) нам потребовалось всего 4 бита для кодирования всей строки.

Преимущества `MSEC`

1. Более высокая степень сжатия: Использование последовательностей символов позволяет более эффективно сжимать данные за счет использования более длинных символов.
2. Учет контекста: Кодирование последовательностей учитывает контекст, что позволяет лучше использовать статистические свойства данных.
3. Эффективность для повторяющихся данных: `MSEC` особенно эффективен для данных с высоко повторяющимися последовательностями.

Недостатки `MSEC`

1. **Сложность реализации:** Анализ и кодирование последовательностей символов более сложны, чем кодирование отдельных символов.
2. **Увеличение вычислительной нагрузки:** Требуется больше вычислительных ресурсов для анализа и кодирования данных.
3. **Размер словаря:** Для эффективного кодирования требуется хранить более крупные словари последовательностей, что может увеличить объем вспомогательной информации.

Применение MSEC
`MSEC` находит применение в различных областях, включая:

- Сжатие текстов: Кодирование последовательностей слов или фраз в текстах.
- *Сжатие изображений: Кодирование последовательностей пикселей или блоков пикселей.
- Сжатие видео: Кодирование последовательностей кадров или макроблоков в видеопотоке.

Пример декодирования
Для декодирования строки, закодированной с использованием `MSEC`, необходимо иметь словарь последовательностей и соответствующих кодов.

Закодированная строка: `0000`

Словарь последовательностей:

- 0: AB

Процесс декодирования:

- Читаем `0` -> AB

- Читаем `0` -> AB
- Читаем `0` -> AB
- Читаем `0` -> AB

Декодированная строка: `ABABABAB`

Context Modeling

Контекстное моделирование — это метод, который используется для улучшения точности сжатия данных, путем учета предыдущих символов или данных при предсказании следующего символа. Представьте себе, что у вас есть серия событий, и вы пытаетесь предсказать, что произойдет дальше, основываясь на том, что уже произошло. Контекстное моделирование помогает вам делать это предсказание более точно.

Пример с текстом
Представим себе, что вы пишете текст и у вас есть следующие слова: "кошка", "кот", "собака". Вы заметите, что слово "кошка" часто следует за словом "кот", но редко за словом "собака". Это и есть контекстное моделирование: использовать предыдущие слова, чтобы предсказать, какое слово будет следующим.

Как это работает в CABAC

1. Сбор статистики
2. Определение контекста
3. Прогнозирование вероятностей
4. Сбор статистики
   На этом этапе система собирает данные о том, какие символы чаще всего встречаются после других символов. Например, если в видео кадры с зелеными пикселями часто следуют за кадрами с синими пикселями, то это будет учтено при сборе статистики.
5. Определение контекста
   Контекст — это набор условий, который используется для определения вероятностей. В видео это может быть цвет предыдущих пикселей или положение кадра. Например, если мы знаем, что предыдущий пиксель был зеленым, это будет нашим контекстом.
6. Прогнозирование вероятностей
   Используя собранную статистику и текущий контекст, система предсказывает вероятность появления каждого следующего символа. Например, если

предыдущий пиксель был зеленым, то вероятность появления синего пикселя будет выше, чем вероятность появления красного.

Аналогия с написанием писем
Представьте, что вы пишете письма на работу и используете шаблоны. Если ваше предыдущее письмо было жалобой, скорее всего, следующее письмо тоже будет жалобой, потому что это логично в данном контексте. Контекстное моделирование работает схожим образом, предсказывая, что будет дальше, на основе того, что было ранее.

Пример с видеокадрами
Допустим, у нас есть видеоряд, где один кадр показывает зеленую траву, а следующий кадр — синее небо. Если система замечает, что после каждого кадра с зеленой травой часто следует кадр с синим небом, она будет использовать эту информацию для прогнозирования следующих кадров.

Технические детали

1. **Инициализация контекста**: при начале кодирования система устанавливает начальный контекст на основе первых нескольких символов или пикселей.
2. **Обновление контекста**: при каждом новом символе или пикселе система обновляет контекст на основе новой информации.
3. **Использование вероятностей**: система использует текущий контекст для определения вероятностей следующих символов или пикселей.

Почему это важно?
Контекстное моделирование позволяет значительно улучшить эффективность сжатия данных. Без учета контекста система будет предсказывать вероятности на основе общего распределения символов, что менее точно и приводит к большему количеству данных для хранения. Используя контекст, система может более точно предсказывать следующие символы, уменьшая объем сжатых данных.

Context Modeling (Scientific campaign)

Контекстное моделирование в теории кодирования: Научный подход
Введение
Контекстное моделирование является фундаментальной концепцией в теории кодирования, особенно в области сжатия данных. Оно основано на использовании контекста (предшествующих символов или событий) для предсказания следующего символа с целью улучшения эффективности кодирования. Основной идеей является

учет зависимости вероятностей появления символов от их окружения, что позволяет более точно моделировать источники данных и, следовательно, более эффективно их сжимать.

Основные концепции

1. **Марковские модели**

Контекстное моделирование часто основано на марковских моделях. В таких моделях вероятность появления символа зависит от состояния системы, которое определяется предыдущими символами.

Пусть $(X = \{X_1, X_2, \ldots, X_n\})$ — последовательность символов, тогда вероятность появления символа (X_{i+1}) зависит от (k) предыдущих символов:

$$P(X_{i+1}|X_1, X_2, \ldots, X_i) \approx P(X_{i+1}|X_{i-k+1}, \ldots, X_i)$$

где (k) — порядок марковской модели.

2. **Вероятностные модели**

Контекстное моделирование требует построения вероятностной модели для символов исходя из их контекста. Например, в случае первого порядка (`bigram`) вероятность символа (X_{i+1}) будет:

$$P(X_{i+1}|X_i)$$

Для более высоких порядков вероятности будут учитывать больший контекст:

$$P(X_{i+1}|X_{i-k+1}, \ldots, X_i)$$

Применение в кодировании

1. **Арифметическое кодирование с контекстным моделированием**

Вместе с контекстным моделированием арифметическое кодирование может существенно улучшить эффективность сжатия.
Процесс кодирования начинается с создания таблицы вероятностей для всех возможных символов в контексте:

$$P(X_i|\textit{контекст})$$

На каждом шаге текущий интервал кодирования уточняется на основе вероятностей символов в текущем контексте. Например, если (X_i) и (X_{i-1}) — предыдущие

символы, то следующий символ (X_{i+1}) будет кодироваться с учетом вероятности $(P(X_{i+1}|X_i, X_{i-1}))$.

2. **PPM (Prediction by Partial Matching)**

PPM является одной из наиболее известных и эффективных методик контекстного моделирования. В этой модели используются различные длины контекстов, и выбор делается на основе наиболее подходящего контекста для предсказания следующего символа.

Формально, вероятность появления символа (X_{i+1}) рассчитывается как взвешенная сумма вероятностей по различным длинам контекстов:

$$P(X_{i+1}) = \sum_{j=0}^{k} \alpha_j P(X_{i+1}|\textit{контекст}_j)$$

где (α_j) — вес для контекста длины (j), $(\textit{контекст}_j)$ — контекст длины (j).

Математические модели и формулы

1. **Модель энтропии**

Основная цель контекстного моделирования — минимизация энтропии источника, что позволяет достичь более высокой степени сжатия. Энтропия (H) для источника с контекстным моделированием может быть представлена как:

$$H(X) = -\sum_{x \in X} P(x) \log P(x|\textit{контекст})$$

где $(P(x|\textit{контекст}))$ — условная вероятность символа (x) в заданном контексте.

2. **Смешанные модели (Mixture Models)**

Иногда используются смешанные модели для комбинирования нескольких источников вероятностей:

$$P(x|\textit{контекст}) = \sum_{i=1}^{n} \lambda_i P_i(x|\textit{контекст})$$

где (λ_i) — весовые коэффициенты, а (P_i) — отдельные модели вероятностей.

Заключение

Контекстное моделирование играет ключевую роль в современных методах сжатия данных, таких как PPM и другие адаптивные алгоритмы. Оно позволяет

значительно повысить эффективность кодирования за счет учета зависимостей между символами. Научное изучение и применение контекстного моделирования включает разработку сложных вероятностных моделей и алгоритмов, способных адаптироваться к статистике обрабатываемых данных, что обеспечивает максимальную степень сжатия при минимальных потерях информации.

Deblocking Filter

Что такое `Deblocking Filter` (Фильтр деблокинга) в видеокодеках?
`Deblocking Filter` (фильтр деблокинга) — это технология, используемая в видеокодеках для уменьшения или устранения визуальных артефактов, известных как "блокинг-артефакты". Эти артефакты возникают в результате сжатия видео и проявляются в виде видимых блоков или границ между блоками пикселей, что ухудшает визуальное качество изображения.

Основная идея
Основная цель фильтра деблокинга — сгладить границы между блоками пикселей, чтобы они выглядели более естественно и менее заметно. Это достигается путем применения специальных алгоритмов, которые анализируют и изменяют значения пикселей на границах блоков.

Почему возникают блокинг-артефакты?
Блокинг-артефакты возникают из-за того, что многие видеокодеки, такие как H.264, H.265 и другие, используют блочное преобразование для сжатия данных. Видео делится на небольшие блоки (обычно `8x8` или `16x16` пикселей), и каждый блок обрабатывается отдельно. При сильном сжатии разница в значениях пикселей между соседними блоками становится заметной, что и приводит к появлению артефактов.

Как работает фильтр деблокинга?
Фильтр деблокинга применяет различные методы сглаживания и коррекции на границах блоков, чтобы уменьшить видимость артефактов. Вот основные этапы работы фильтра деблокинга:

1. **Определение границ блоков:** Фильтр идентифицирует границы между блоками пикселей в кадре.
2. **Анализ границ:** Фильтр анализирует значения пикселей на границах, чтобы определить, есть ли значительные различия, которые могут быть восприняты как артефакты.

3. **Применение сглаживания:** Если разница значительна, фильтр применяет сглаживание, изменяя значения пикселей на границах, чтобы уменьшить резкость переходов.

Технические детали

1. Адаптивное сглаживание: Фильтр деблокинга адаптивен, то есть он применяет различную степень сглаживания в зависимости от уровня разницы между блоками. Если разница небольшая, фильтр может оставить границу неизменной. Если разница большая, фильтр применяет более агрессивное сглаживание.
2. Пространственное и временное сглаживание: Фильтр деблокинга может применять пространственное сглаживание (в пределах одного кадра) и временное сглаживание (между последовательными кадрами), чтобы улучшить визуальное качество видео.
3. Краевое сглаживание: Особое внимание уделяется краям объектов и тонким деталям, чтобы избежать размытости и сохранить резкость изображения.

Пример на практике
Рассмотрим, как фильтр деблокинга работает на примере видеокадра, содержащего блоки 8x8 пикселей.

Исходное состояние

- Без фильтра: Видны четкие границы между блоками, особенно в однородных областях, таких как небо или стены.
- С фильтром: Границы сглажены, блоки менее заметны, изображение выглядит более естественно.

Применение фильтра

1. Идентификация границ: Фильтр определяет границы между блоками 8x8 пикселей.
2. Анализ: Фильтр анализирует разницу в значениях пикселей на границах. Например, если граница между двумя блоками выглядит так:

```
Блок 1:  50, 50, 50, 50, 50, 50, 50, 50
Граница: 50, 50, 50, 50 | 60, 60, 60, 60
Блок 2:  60, 60, 60, 60, 60, 60, 60, 60
```

3. Сглаживание: Если разница между блоками значительна, фильтр изменяет значения пикселей на границе, чтобы уменьшить видимость перехода:

```
Новая граница: 50, 50, 55, 55 | 55, 55, 60, 60
```

Преимущества фильтра деблокинга

1. Улучшение качества видео: Сглаживание границ между блоками делает изображение более плавным и естественным.
2. Сохранение деталей: Адаптивный подход позволяет сохранять важные детали изображения, такие как края и тонкие линии, без значительного размытия.
3. Снижение артефактов: Уменьшение блокинг-артефактов улучшает общее восприятие видео, особенно при сильном сжатии.

Недостатки фильтра деблокинга

1. Увеличение вычислительной нагрузки: Применение фильтра требует дополнительных вычислительных ресурсов, что может замедлить обработку видео, особенно на устройствах с ограниченной мощностью.
2. Потенциальное размытие: В некоторых случаях фильтр может слишком агрессивно сглаживать изображение, что приводит к потере резкости и размытию деталей.

Применение в видеокодеках
Фильтр деблокинга используется в различных видеокодеках для улучшения качества видео:

- **`H.264/AVC`:** Включает встроенный фильтр деблокинга для сглаживания блокинг-артефактов.
- **`HEVC/H.265`:** Использует улучшенный фильтр деблокинга для достижения более высокого качества видео при сильном сжатии.
- **`VP9`:** Видеокодек от Google, также включает фильтр деблокинга для улучшения качества изображения.

Context-based Adaptive Binary Arithmetic Coding (CABAC)

Context-based Adaptive Binary Arithmetic Coding (`CABAC`) — это метод сжатия данных, используемый в кодировании видео, который помогает уменьшить размер видеофайлов без потери качества. `CABAC` используется в стандарте видео-кодирования H.264/MPEG-4 AVC, который широко применяется для сжатия видео. Чтобы понять, как работает `CABAC`, давайте разберем его на части и объясним с помощью аналогий.

Основные компоненты `CABAC`

1. Бинаризация (Binarization)
2. Контекстное моделирование (Context Modeling)
3. Адаптивное арифметическое кодирование (Adaptive Arithmetic Coding)
4. Бинаризация (Binarization)
   Представьте, что у вас есть книга, написанная на русском языке. Если бы мы захотели передать эту книгу по телеграфу, который понимает только сигналы «точка» и «тире», нам нужно было бы перевести каждую букву в последовательность точек и тире (аналог азбуки Морзе). Бинаризация в CABAC делает нечто подобное: она переводит сложные данные (например, цвета пикселей) в простую бинарную форму (последовательность 0 и 1).
5. Контекстное моделирование (Context Modeling)
   Теперь представьте, что у вас есть книга с загадками. Если вы уже знаете, что следующая загадка будет про животных, вам будет легче угадать ответ. Контекстное моделирование в CABAC использует похожую идею: оно смотрит на предыдущие данные, чтобы предсказать, что будет дальше. Например, если предыдущий бит был 1, то вероятность того, что следующий бит тоже будет 1, может быть высокой. Таким образом, система адаптируется к контексту, делая предсказания более точными.
6. Адаптивное арифметическое кодирование (Adaptive Arithmetic Coding)
   Представьте себе весы с двумя чашами. Одна чаша представляет собой вероятность появления 0, а другая — вероятность появления 1. Адаптивное арифметическое кодирование берет эти вероятности и сжимает данные в очень компактный формат, как если бы вы пытались сбалансировать весы с высокой точностью. Этот метод позволяет закодировать больше информации в меньшем количестве битов, чем стандартные методы, такие как кодирование фиксированной длины.

Как это работает вместе?

1. Бинаризация переводит данные в последовательность битов (0 и 1).
2. Контекстное моделирование предсказывает вероятности появления каждого бита на основе предыдущих данных.
3. Адаптивное арифметическое кодирование использует эти вероятности, чтобы сжать данные в очень компактную форму.

Аналогия с библиотекой
Представьте себе большую библиотеку с множеством книг:

- **Бинаризация** — это как если бы вы перевели каждую книгу в библиотеке в последовательность точек и тире.
- **Контекстное моделирование** — это как если бы вы использовали знания о предыдущих книгах, чтобы предсказать содержание следующих книг, делая процесс предсказания быстрее и точнее.
- **Адаптивное арифметическое кодирование** — это как если бы вы упаковали все эти книги в коробки так плотно и эффективно, что они занимают как можно меньше места.

Flip-Adaptive Transform

Что такое Flip-Adaptive Transform (FAT)?
Flip-Adaptive Transform (FAT), или адаптивное преобразование с переворотом, — это метод, используемый в видеокодировании для улучшения эффективности сжатия и качества изображения. FAT динамически определяет, следует ли переворачивать блоки пикселей перед применением дискретного косинусного преобразования (DCT) или другого аналогичного преобразования, чтобы минимизировать высокочастотные компоненты и, следовательно, улучшить сжатие.

Основная идея
Основная идея FAT заключается в адаптивном изменении ориентации блоков пикселей в зависимости от их содержимого перед применением преобразования. Это помогает лучше приспособить блоки к природе DCT, уменьшая количество высокочастотных компонентов, которые труднее сжать.

**Как работает FAT?

1. Разделение изображения на блоки: Исходное изображение делится на небольшие блоки пикселей, например, размером 8x8 или 16x16 пикселей.

2. Анализ содержимого блока: Для каждого блока анализируется его содержимое, чтобы определить оптимальную ориентацию перед применением преобразования.
3. Переворот блока (если необходимо): В зависимости от анализа блок может быть перевернут (по горизонтали, вертикали или диагонали) для минимизации высокочастотных компонентов.
4. Применение преобразования: Применяется DCT или другое преобразование к адаптированному блоку для дальнейшего сжатия.

Преимущества FAT

1. Улучшенное сжатие: Оптимизация ориентации блоков помогает снизить количество высокочастотных компонентов, что улучшает эффективность сжатия.
2. Повышенное качество изображения: Уменьшение высокочастотных компонентов приводит к меньшему количеству артефактов при декодировании, что улучшает визуальное качество.
3. Адаптивность: Метод адаптивен и может быть применен к любым блокам в изображении, что делает его гибким для различных типов контента.

Пример на практике
Рассмотрим пример использования FAT на блоке `8x8` пикселей.

1. Исходный блок:

```
[52, 55, 61, 66, 70, 61, 64, 73]
[63, 59, 55, 90, 109, 85, 69, 72]
[62, 59, 68, 113, 144, 104, 66, 73]
[63, 58, 71, 122, 154, 106, 70, 69]
[67, 61, 68, 104, 126, 88, 68, 70]
[79, 65, 60, 70, 77, 68, 58, 75]
[85, 71, 64, 59, 55, 61, 65, 83]
[87, 79, 69, 68, 65, 76, 78, 94]
```

2. Анализ содержимого: Алгоритм анализирует блок, чтобы определить, следует ли его перевернуть для минимизации высокочастотных компонентов.
3. Переворот блока (если необходимо):
    - Горизонтальный переворот:

```
[73, 64, 61, 70, 66, 61, 55, 52]
[72, 69, 85, 109, 90, 55, 59, 63]
[73, 66, 104, 144, 113, 68, 59, 62]
[69, 70, 106, 154, 122, 71, 58, 63]
[70, 68, 88, 126, 104, 68, 61, 67]
[75, 58, 68, 77, 70, 60, 65, 79]
[83, 65, 61, 55, 59, 64, 71, 85]
[94, 78, 76, 65, 68, 69, 79, 87]
```

- Вертикальный переворот:

```
[87, 79, 69, 68, 65, 76, 78, 94]
[85, 71, 64, 59, 55, 61, 65, 83]
[79, 65, 60, 70, 77, 68, 58, 75]
[67, 61, 68, 104, 126, 88, 68, 70]
[63, 58, 71, 122, 154, 106, 70, 69]
[62, 59, 68, 113, 144, 104, 66, 73]
[63, 59, 55, 90, 109, 85, 69, 72]
[52, 55, 61, 66, 70, 61, 64, 73]
```

4. Применение DCT: После переворота (если он необходим), к блоку применяется DCT для преобразования данных в частотный домен.

Применение FAT в видеокодеках
FAT используется в современных видеокодеках для улучшения эффективности сжатия и качества изображения, включая:

- `HEVC (H.265)` : Включает механизмы адаптивного изменения ориентации блоков для оптимизации сжатия.
- `VP9` : Видеокодек от Google, также использует адаптивные методы для улучшения качества сжатия.

Преимущества и недостатки
Преимущества

1. Улучшенное сжатие: FAT помогает уменьшить высокочастотные компоненты, улучшая сжатие.

2. Повышенное качество: Сглаживание высокочастотных компонентов уменьшает артефакты, улучшая визуальное восприятие.
3. Гибкость: Метод адаптивен и может быть применен к различным блокам изображения, обеспечивая гибкость.

Недостатки

1. Увеличение вычислительной нагрузки: Анализ и переворот блоков требуют дополнительных вычислительных ресурсов.
2. Сложность реализации: Реализация FAT может быть сложной, особенно при обработке больших объемов данных в реальном времени.

Constrained Directional Enhancement Filter

Зачем нужен `CDEF`?
Представьте себе, что вы нарисовали красивую картину, но кто-то пролил на нее воду. Некоторые детали размазались, и картина потеряла свою четкость. `CDEF` работает как особая «восстанавливающая кисть», которая помогает восстановить детали и уменьшить размытость.

Артефакты и шумы
В контексте цифрового видео, артефакты и шумы — это нежелательные искажения. Артефакты могут появляться в результате сжатия видео (например, когда мы пытаемся уменьшить размер файла), а шумы могут быть вызваны недостаточным освещением или помехами в процессе съемки. Это похоже на то, как пятна и размытые линии могут появиться на вашей картине.

Направления в изображении
`CDEF` учитывает направление линий и границ в изображении. Например, если в вашей картине есть четкие горизонтальные или вертикальные линии, фильтр будет стараться сохранить их четкость, удаляя только те искажения, которые не следуют этим направлениям. Это можно сравнить с тем, как реставратор картина использует направление мазков, чтобы аккуратно восстановить поврежденные участки.

Констрейнты (ограничения)
`CDEF` применяет улучшения, соблюдая определенные ограничения, чтобы не испортить оригинальное изображение. Это значит, что фильтр старается найти баланс между удалением артефактов и сохранением исходных деталей. Это похоже на то, как реставратор картины аккуратно удаляет грязь, не повреждая оригинальные мазки художника.

Процесс работы `CDEF`

Теперь давайте представим, как именно работает `CDEF`:

1. **Разбиение изображения на блоки**: Фильтр разделяет изображение на небольшие блоки, чтобы работать с каждым из них отдельно. Это как если бы вы разделили свою картину на множество маленьких квадратиков для более точного восстановления.
2. **Определение направлений**: В каждом блоке фильтр определяет основные направления (горизонтальные, вертикальные или диагональные линии). Это как если бы вы сначала определили направление мазков в каждом квадратике.
3. **Применение фильтрации**: Затем фильтр применяет специальный алгоритм для сглаживания шумов и артефактов, учитывая направления. Это как если бы вы аккуратно восстанавливали каждый квадратик, следуя направлению мазков, чтобы сохранить оригинальные детали.
4. **Собирание блоков**: После обработки каждого блока фильтр собирает их вместе, чтобы получить улучшенное изображение. Это как если бы вы снова собрали все квадратики в единую картину, теперь уже без пятен и размытых линий.

Технические детали

Теперь давайте рассмотрим некоторые технические аспекты работы CDEF:

- **Разделение на блоки**: Изображение разбивается на блоки фиксированного размера, например, 8x8 пикселей.
- **Определение направлений**: В каждом блоке определяется преобладающее направление границ, чтобы фильтрация была наиболее эффективной.
- **Параметры фильтрации**: Для каждого направления подбираются параметры фильтрации, которые зависят от интенсивности артефактов и шума.
- **Адаптивность**: Фильтр адаптируется к различным типам артефактов и уровню шума, что позволяет эффективно улучшать изображение при различных условиях.

Loop Restoration Filter

Loop Restoration Filter (`LRF`) — это фильтр, применяемый в области обработки видео, особенно в современных видеокодеках, таких как `AV1`. Он предназначен для улучшения качества изображения, выполняя очистку от шумов и артефактов, возникающих при сжатии видео. Чтобы объяснить, как работает `LRF`, давайте разберем его компоненты и принципы работы с помощью аналогий.

Зачем нужен Loop Restoration Filter?
Представьте, что вы рисуете картину и по завершении обнаруживаете мелкие пятна и дефекты. Loop Restoration Filter работает как специальный инструмент для их удаления, улучшая общее качество изображения.

Шумы и артефакты
В цифровом видео шумы могут появляться из-за плохого освещения или помех при съемке, а артефакты — из-за сжатия видеофайлов. Это аналогично тому, как на картине могут появиться пятна от грязных кистей или от неправильного смешивания красок.

Принцип работы Loop Restoration Filter
Теперь давайте рассмотрим, как именно работает `LRF`, используя понятные аналогии.

1. Декомпозиция на блоки
   Изображение делится на небольшие блоки, обычно размером `64x64` пикселей. Это можно представить как деление большой картины на множество маленьких квадратиков, чтобы удобнее работать с каждым из них.
2. Анализ и очистка
   Каждый блок анализируется на наличие шумов и артефактов. `LRF` применяет несколько техник для их устранения:

- **Wiener Filter**: Представьте себе реставратора, который использует мелкие кисти и растворители, чтобы аккуратно убрать пятна, не повреждая оригинальные мазки. Wiener фильтр работает подобным образом, устраняя шумы и восстанавливая исходное изображение.
- **Self-Guided Filter**: Это как если бы реставратор смотрел на похожие участки картины и использовал их для восстановления поврежденного участка, подбирая цвета и текстуры, чтобы они соответствовали окружающим областям.

Примеры фильтров

1. Wiener Filter
   Этот фильтр используется для уменьшения шума в изображении. Он работает, находя оптимальный баланс между сглаживанием шума и сохранением деталей изображения. Это как если бы реставратор тщательно выбирал растворитель, который удаляет грязь, но не трогает краску.

2. Self-Guided Filter

   Этот фильтр основывается на анализе соседних пикселей и их значений для восстановления поврежденных областей. Он использует информацию о соседних пикселях, чтобы восстановить детали в каждой точке изображения. Это похоже на реставратора, который смотрит на похожие участки картины, чтобы понять, как восстановить поврежденный участок.

Комбинация фильтров

`LRF` применяет несколько фильтров последовательно или параллельно для достижения наилучшего результата. Это как если бы реставратор сначала использовал один метод для удаления грязи, а затем другой для восстановления деталей, и затем проверял, нужно ли еще что-то доработать.

Итеративный процесс

Важной особенностью `LRF` является итеративный процесс улучшения. После первичной обработки фильтр может повторно проанализировать изображение и применить дополнительные улучшения. Это аналогично тому, как реставратор может несколько раз проходить по картине, уточняя и улучшая результат после каждой итерации.

Multithreaded Encoding

Многопоточное кодирование (`Multithreaded Encoding`) — это метод увеличения скорости обработки данных за счет одновременного использования нескольких потоков выполнения. Этот подход широко используется в видеокодировании для ускорения процесса сжатия видео, что особенно важно при работе с высококачественным видеоформатом.

Зачем нужно многопоточное кодирование?

Представьте, что вы решили написать книгу и хотите сделать это как можно быстрее. Если вы будете писать книгу в одиночку, это займет много времени. Но что если вы сможете разделить свою книгу на главы и раздать их нескольким писателям, чтобы они писали одновременно? В итоге, книга будет готова значительно быстрее. Многопоточное кодирование работает по тому же принципу: процесс кодирования видео разбивается на части, каждая из которых обрабатывается отдельным потоком.

Потоки и ядра процессора

Компьютерные процессоры (ЦП) имеют несколько ядер, и каждое ядро может выполнять отдельный поток. Поток — это последовательность инструкций,

которые выполняются процессором. В многопоточном кодировании несколько потоков одновременно обрабатывают различные части видео, используя разные ядра процессора.

Разделение задачи
Многопоточное кодирование видео подразумевает разделение видео на меньшие части, которые могут быть обработаны параллельно. Это похоже на разделение вашей книги на главы для одновременного написания разными авторами.

1. Разделение на кадры
   Видеофайл состоит из множества кадров (изображений, отображаемых последовательно). Один из способов многопоточного кодирования — разделение видео на кадры, каждый из которых обрабатывается отдельным потоком. Это как если бы каждый писатель работал над отдельной главой книги.
2. Разделение на блоки

Внутри каждого кадра изображение можно разделить на блоки фиксированного размера. Каждый блок может быть обработан отдельным потоком. Это как если бы каждый писатель работал над отдельным абзацем главы.

Координация потоков
Для эффективной работы многопоточное кодирование требует координации потоков. Это включает:

- **Управление доступом к данным**: Потоки должны координировать доступ к общей памяти, чтобы избежать конфликтов и гарантировать согласованность данных. Это похоже на то, как писатели должны следить за тем, чтобы их главы и абзацы логически согласовывались друг с другом.
- **Синхронизация**: Потоки должны синхронизироваться, чтобы завершить свои задачи одновременно. Например, все главы книги должны быть готовы к печати одновременно.

Преимущества многопоточного кодирования

- **Увеличение скорости**: Многопоточное кодирование позволяет значительно сократить время обработки видео, что особенно важно для больших видеороликов или потоковой передачи видео в реальном времени.

- **Эффективное использование ресурсов**: Современные процессоры имеют несколько ядер, и многопоточное кодирование позволяет использовать все ядра, увеличивая общую производительность системы.

Вызовы и сложности

Несмотря на преимущества, многопоточное кодирование имеет свои вызовы:

- **Управление зависимостями**: Некоторые части видео могут зависеть друг от друга. Например, один кадр может зависеть от предыдущего. Это требует сложных механизмов синхронизации.
- **Проблемы с согласованностью данных**: Несогласованные изменения данных могут привести к ошибкам в конечном видеофайле. Это требует тщательного управления доступом к общей памяти.

Технические детали

Теперь давайте рассмотрим технические аспекты многопоточного кодирования:

1. Кодирование кадров
   Процессор может одновременно кодировать несколько кадров, назначая каждому ядру обработку определенного кадра. Это позволяет сократить время, затрачиваемое на обработку всего видеофайла.
2. Кодирование блоков
   Внутри каждого кадра изображение делится на блоки, которые обрабатываются параллельно. Это позволяет оптимизировать использование процессорного времени и памяти.
3. Синхронизация потоков
   Для эффективного многопоточного кодирования используются различные методы синхронизации, такие как мьютексы и семафоры, чтобы управлять доступом к общей памяти и координировать завершение задач.