Фрактальная графика

Понятия фрактал и фрактальная геометрия, появившиеся в конце 70-х, с середины 80-х прочно вошли в обиход математиков и программистов. Слово фрактал, образованого от латинского fractus и в переводе означает, состоящий из фрагментов. В математике фракталом называют множество точек в Евклидовом пространстве с дробной метрической размерностью (например, хаусдорфовой), превышающей топологическую. Первые известные примеры фрактальных множеств были изучены уже в XIX веке (например, *канторово множество*, *кривая Коха*, *треугольник Серпинского*). Однако термин «фрактал» ввёл Б. Мандельброт только в 1975 г., а широкую известность они получили после публикации его книги «Fractal Geometry of Nature» (перевод «Фрактальная геометрия природы», 1982). Мандельброт первым применил компьютерные расчёты для визуализации фракталов, в частности множества Мандельброта и ассоциированных множеств Жюлиа.

Фракталы оказались чрезвычайно полезны в **компьютерной графике** и моделировании природных форм. Многие явления природы (облака, горы, береговые линии, разветвления деревьев) оказываются приближенными фракталами: их структура повторяется на разных масштабах. Это привело к развитию фрактального искусства (генерация абстрактных картин на основе простейших итеративных правил) и активному использованию фракталов в фильмах и играх для создания реалистичных пейзажей и спецэффектов. Влияние фракталов распространилось и на другие области: физику (моделирование турбулентности), биологию (структуры ветвящихся сосудов), сигнализацию и даже финансы (капитал, где Мандельброт изучал «хаотичность» цен).

Одним из основных свойств фракталов является самоподобие. В самом простом случае небольшая часть фрактала содержит информацию о всем фрактале.

Определение фрактала, данное Мандельбротом, звучит так: “Фракталом называется структура, состоящая из частей, которые в каком-то смысле подобны целому”.

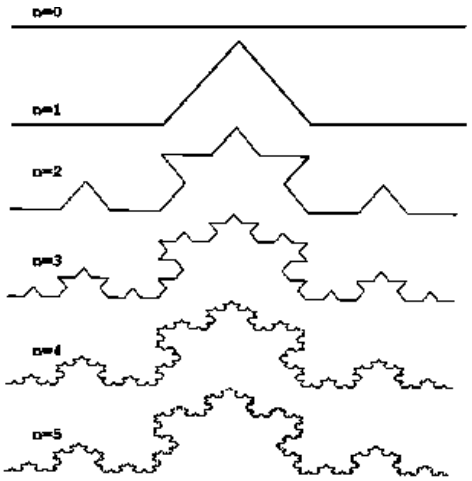
Более научное определение: фрактал — множество (или фигура), обладающее рекурсивной структурой, при которой локальные части подобны глобальной форме. Это означает, что независимо от масштаба мелкие детали сохраняют свойства целого. Классическим примером самоподобной кривой является снежинка Коха, части которой (четыре отрезка) подобны целому крылу.

Классификация фракталов

Для чтобы представить все многообразие фракталов удобно прибегнуть к их общепринятой классификации.

Геометрические фракталы. Фракталы этого класса самые наглядные. В двумерном случае их получают с помощью некоторой ломаной (или поверхности в трехмерном случае), называемой *генератором*. За один шаг алгоритма каждый из отрезков, составляющих ломаную, заменяется на ломаную-генератор, в соответствующем масштабе. В результате бесконечного повторения этой процедуры, получается геометрический фрактал.

Построение триадной кривой Кох начинается с отрезка единичной длины, который является нулевым поколением кривой. Затем отрезок делится на три равные части, и средняя часть заменяется на образующий элемент, который представляет собой равносторонний треугольник. Этот процесс повторяется для каждого из четырёх звеньев, что приводит к образованию кривой Коха. Каждое поколение кривой становится более сложной и детализированной, но не пересекает себя.

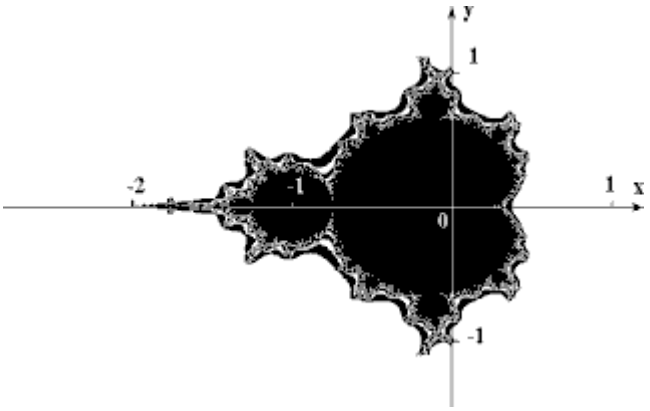


Построение триадной кривой Кох.

В компьютерной графике использование геометрических фракталов необходимо при получении изображений деревьев, кустов, береговой линии. Двумерные геометрические фракталы используются для создания объемных текстур (рисунка на поверхности объекта).

Алгебраические фракталы. Это самая крупная группа фракталов. Получают их с помощью нелинейных процессов в n-мерных пространствах. Наиболее изучены двумерные процессы. Интерпретируя нелинейный итерационный процесс, как дискретную динамическую систему, можно пользоваться терминологией теории этих систем: фазовый портрет, установившийся процесс, аттрактор и т.д.

Известно, что нелинейные динамические системы обладают несколькими устойчивыми состояниями. То состояние, в котором оказалась динамическая система после некоторого числа итераций, зависит от ее начального состояния. Поэтому каждое устойчивое состояние (или как говорят - аттрактор) обладает некоторой областью начальных состояний, из которых система обязательно попадет в рассматриваемые конечные состояния. Таким образом, фазовое пространство системы разбивается на области притяжения аттракторов. Если фазовым является двухмерное пространство, то окрашивая области притяжения различными цветами, можно получить цветовой фазовый портрет этой системы (итерационного процесса). Меняя алгоритм выбора цвета, можно получить сложные фрактальные картины с причудливыми многоцветными узорами. Неожиданностью для математиков стала возможность с помощью примитивных алгоритмов порождать очень сложные нетривиальные структуры.



Множество Мандельброта.

Стохастические фракталы. Еще одним известным классом фракталов являются стохастические фракталы, которые получаются в том случае, если в итерационном процессе случайным образом менять какие-либо его параметры. При этом получаются объекты очень похожие на природные – несимметричные деревья, изрезанные береговые линии и т.д. Двумерные стохастические фракталы используются при моделировании рельефа местности и поверхности моря.

Математические основы

Фрактальное измерение

Ключевая характеристика фрактала — его **фрактальная размерность** (по Хаусдорфу). В отличие от целочисленной топологической размерности, фрактальная размерность может быть дробной. Фрактальная размерность (D) фактически определяется масштабным законом: при увеличении масштаба объект «раздувается» в (b) раз по размерам, а во столько же раз количество самоподобных копий изменяется как (a), тогда (a = b^D). Например, треугольник Серпинского состоит из 3 копий в масштабировании 2 раза, поэтому (3 = 2^D) и (D=\log_2 3 \approx 1{,}585). Кривая Коха (запоминаемая как снежинка) состоит из 4 копий в масштабе 3, давая (4 = 3^D) и (D=\log_3 4 \approx 1{,}26). Такие примеры показывают: фракталы не вписываются в привычные целочисленные измерения «линия (1D), плоскость (2D)», а имеют дробную «степень сложности».

Практически фрактальную размерность вычисляют как *предел отношения* логарифмов числа частей к логарифму коэффициента масштабирования. Более строго — размерность Хаусдорфа. Важно, что фрактальное измерение чувствительно к «шероховатости» объектов: гладкая кривуля всегда имеет размерность 1, а фрактальная кривая чуть «извитее» уже даст >1. Так, площадь снежинки Коха равна конечному числу, хотя ее периметр бесконечен, и размерность ее контура — 1,26. Для множество Мандельброта известна результативность: установлено, что его граница имеет размерность 2 (кстати, доказывал это Шишикура, 1998).

Таким образом, фракталы введены для количественного описания сложных фигур природы. Факт: **любой контур или поверхность, существенно отличающиеся от идеально гладкой**, можно характеризовать фрактальной размерностью Хаусдорфа. Создание правил самоподобия или вычисление размерности позволяет переносить аналитические задачи в область геометрии, где они легко интерпретируются.

Фрактальные множества и самоподобие

Типичными примерами фрактальных множеств являются *канторово множество*, *треугольник Серпинского*, *снежинка Коха*, а также *множество Жюлиа* и *множество Мандельброта*. Канторово множество строится отрезком, из которого повторно вырезаются средние трети, получая бесконечно раздробленную дискретную структуру. Треугольник Серпинского и снежинка Коха уже были рассмотрены выше. Все они образуются детерминированным применением правил редукции (преобразования частей).

Множества Жюлиа и Мандельброта возникают из итераций комплексного квадратного отображения. Множество Мандельброта определяется так: точка (c) комплексной плоскости принадлежит множеству, если последовательность (z_{n+1}=z_n^2+c) (начиная с (z_0=0)) остаётся ограниченной. Множество Жюлиа

для заданного параметра (c) — это множество начальных (z_0), для которых итерация не убегает на бесконечность. Для них применяется *escape-time* (алгоритм “время выхода”), где число шагов до выхода или достижение максимального числа итераций служит значением функции, определяющей окраску точки. Таким образом возникают «классические» фрактальные изображения (со сложной границей, самоподобной структурой).

Отметим: фракталы бывают *точно* самоподобными (как Кох, Серпинский), а могут быть и *стохастически* самоподобными. Последние, например, моделируют природные объекты, где копии похожи в среднем, а не точь-в-точь. Разбиения отрезка по среднему (кантор), графическое повторение ломаных (Кох) — примеры геометрических замен. Множества Жюлиа/Мандельброта демонстрируют *аналитическое* самоподобие, вытекающее из динамики итераций. Все эти примеры иллюстрируют, что локальная структура фрактала повторяет глобальную форму при разных масштабах.

Алгоритмы генерации фракталов

Существует множество подходов к генерации фрактальных изображений. Общая идея – многократно применять простые операции. Далее перечислены основные группы алгоритмов:

- **Итерационные функции (IFS)**: семейство сокращающих аффинных преобразований, применяемых итеративно к набору точек или целой фигуре. Теорема Хатчинсона гарантирует, что последовательное применение каждого преобразования с некоторой вероятностью даёт единственный инвариантный аттрактор — фрактал. Известный пример — папоротник Барнсли, получаемый из набора линейных преобразований. В общем случае IFS-алгоритмы случайно выбирают на каждом шаге одно из преобразований и перемещают текущую точку, постепенно заполняя фрактальный аттрактор.

- **Кривые Коха, Дракон и другие**: генерируются рекурсивным разбиением отрезков или полилиний. Например, кривая Коха строится из начального отрезка и на каждом шаге заменяет каждый отрезок 4-мя короткими (изгибая средний отрезок). Дракон Хартера–Хайтуитта и другие «линиеподобные» фракталы строятся по подобному принципу (L-системы или рекурсивное деление отрезков).

- **Escape-time алгоритмы (Мандельброт/Жюлиа)**: для каждой точки на экране определяется, улетит ли последовательность (z_{n+1}=z_n^2+c) в бесконечность. Количество итераций до «выхода» или достижение максимум-итераций используется для раскраски. Эти алгоритмы требуют двойной вложенной петли по пикселям и итерациям, сложность ~O(N_pixels × N_maxIter). Их преимущество — крайне детализованные структуры, но негатив — «плоские» зоны (когда точки остаются в множестве).

- **L-системы**: формальные грамматики, введённые А. Линденмайером в 1968 году для моделирования роста растений. L-система описывается аксиомой и правилами переписывания символов. После n итераций полученный «алфавит» интерпретируется через трафарет (обычно «черепашью графику»), что даёт дерево или фрактал. Пример: снежинка Коха можно задать L-системой вида (F \to F+F--F+F). Этот подход удобен для «органических» форм и легок в реализации. Обобщённые (стохастические, параметрические) L-системы позволяют создавать вариативность. Сложность генерации экспоненциальна по числу итераций (рост числа символов). (Подтверждение: введено Линденмайером, 1968.)

- **Алгоритм среднего смещения (Midpoint displacement)**: рекурсивно разбивает отрезки (или треугольники) и добавляет случайный дрейф к средней точке. Чаще всего применяется для генерации высотных карт. Алгоритм генерирует две новые точки на середине отрезка с добавленным рандомом, затем рекурсивно повторяется для каждого подпроцесса. На двумерной решётке это может выглядеть как «алгоритм Ричардсона». Сложность на сетку размера (n\times n) примерно O(n^2). Результаты выглядят как природные горы с параметром «шероховатость» (постоянная смещения).

- **Алгоритм «алмаз-квадрат»**: усовершенствование midpoint для генерации плавающих объектов (ландшафтов) на квадратной сетке. Изначально задаются 4 угловые высоты. Затем шаг «алмаз»: вычисление центральной точки квадрата как среднее углов плюс шум; шаг «квадрат»: вычисление точек по серединам сторон квадратов в аналогичном стиле. Эти шаги повторяются, пока не заполнится сетка. Алгоритм обладает рекурсивной структурой и даёт равномерно шероховатый рельеф. Он был впервые описан Альеном Фурнье (1982). Сложность — O(n^2) для сетки n×n.

- **Стохастические фракталы и шум**: вместо детерминированной рекурсии можно использовать случайные процессы. Например, фрактальный шум (шум Перлина, симплекс-шума) генерируется наложением шумовых функций с разными частотами (фрактализация шума). Перлин (1985) предложил алгоритм *gradient noise*: на фиксированной решётке точек задаются псевдослучайные градиенты, а между ними интерполируется плавная функция. В результате образуется натурально выглядящая текстура. Симплекс-шум (Перлин, 2001) — модификация для выше размерностей без артефактов. По словам Перлина, классический алгоритм имеет сложность O(d^2) в измерении d, а симплекс-шума — O(d). Это делает симплекс-шума предпочтительным для 3D/4D эффектов. Стохастические методы легко генерируют естественные текстуры и рельефы (с помощью *фрактализации*: многократное наложение шума с уменьшенной амплитудой/частотой). Они обычно имеют меньше артефактов регулярной структуры, но труднее точно контролируются. В целом, выбор алгоритма зависит от требуемого результата: IFS/крючья подходят для рекурсивно-детализированных объектов; escape-time для «математических» узоров; L-системы — для биомоделирования; шум и midpoint/diamond — для природных пейзажей.

Сравнение алгоритмов генерации

Ниже приведена сводная таблица основных методов генерации фракталов. Она иллюстрирует относительные затраты времени (большая сложность = дольше считать), качество получаемой фрактальной текстуры и области применения.

Алгоритм / метод	Сложность	Качество результата	Область применения
Escape-time (Мандельброт)	Высокая (итерации на пиксель)	Очень детализированные «спирали» множества	Классическая фрактальная визуализация; демонстрация; наука
IFS (случайный)	Средняя (много точек)	Симметричные геометрические узоры (деревья, папоротники)	Генерация растений, фрактальное сжатие изображений
IFS (детермин.)	Средняя–высокая	Чёткие симметричные структуры	Сжатие изображений, художественная графика
L-системы	Средняя	Натуральные разветвления; «растения»	Моделирование растительности, интерактивные модели
Fractal noise (fBM)	Низкая (O(N))	Органичные текстуры (облака, горы)	Текстуры, ландшафты, спецэффекты
Diamond-Square	Средняя (O(n^2))	Неровный ландшафт, земная топография	Генерация карт высот, игра/VR-технологии
Другие стох. методы (DLA)	Высокая (моделирование частиц)	Сложные натуральные узоры (кристаллы, снежинки)	Специальные задачи (напр., моделирование волн кристаллов)

Эта таблица упрощённо отражает, что *escape-time* даёт чистейший вид математических фракталов (качественно красивых, но тяжёлых в расчёте); *IFS/L-системы* хорошо работают для «похожих на природу» форм; *шумовые алгоритмы* дают быстро грубые структуры, удобные для эффектов и

игрушек. Выбор метода зависит от цели: точная геометрия (IFS/L) или реалистичная имитация (шумы).

Визуализация и рендеринг фракталов

Фракталы, как объекты с бесконечной детализацией, требуют специальных приёмов отображения. Ключевые моменты:

- **Антиалиасинг**: из-за «кусочков» и сильных контуров на высоких масштабах возникают эффекты дискретизации. Практики рекомендуют многократную выборку пикселя или фильтрацию изображения для подавления «лесенок» . В случае фракталов с гладкими кривыми можно использовать аналитический подход: строить кривую до пиксельного разрешения, но это тяжеловато. Алгоритмы сглаживания частично решают проблему, но не устраняют её полностью.

- **Цветовые палитры и нормализация**: в алгоритмах escape-time цвет пикселя обычно зависит от числа итераций. Стандартные техники (линейная или полосная палитра, градиент по нормированному числу итераций) применяются для художественной окраски. Выбор палитры и методы нормализации уровня («плавная» или дискретная окраска) существенно влияют на восприятие картины.

- **Освещение и текстуры**: при отображении 3D-террейнов, полученных фракталами (midpoint, diamond), можно применять традиционное освещение: вычислять нормали поверхности и фоново-шедевать. «Гипер-фрактальные» объекты (например, линейные фракталы в 3D) требуют вычисления «касательных» или «нормалей» к фрактальной поверхности для освещения. Для текстурирования фрактальных поверхностей обычно используют процедурные текстуры (шум, слойность оттенков) или обычные изображения. Во многих CG-приложениях фрактал задаёт только форму (рельеф), а визуальные детали добиваются сочетанием diffuse/specular текстур.

Ключевое замечание: фракталы не фотопараметрические (не основаны на реальных физических моделях), поэтому многие методы рендеринга (освещение, тени) применимы к ним так же, как и к любой геометрии, но артефакты могут появляться при интенсификации деталей. Важно нормализовать координаты фрактала к единичному пространству визуализации (обычно [0,1] или экранные координаты) и корректировать амплитуду функций шума, чтобы не было «выпрыгивающих» значений (***не подтверждено***).

Применение фрактальной графики в CG

Фракталы нашли широкое применение в компьютерной графике благодаря своей способности моделировать сложные природные структуры. Известные применения:

- **Процедурная генерация ландшафтов**: один из классических примеров — генерация гор и планетарных пейзажей с помощью алгоритмов среднего смещения, diamond-square или многослойного шума Перлина. Такие алгоритмы позволяют быстро создать большие равнины, горные хребты, каньоны, озёра и т.д. При должной постобработке (сглаживание, эрозия) достигается фотореалистичный эффект. Fractal Foundation отмечает, что фракталы особенно полезны в компьютерной графике для **создания реалистичных ландшафтов и текстур** в играх и фильмах.

- **Процедурные текстуры**: облака, мрамор, дерево, кора – часто генерируются фрактальным шумом. Перлин и Симплекс создают «органические» пятнистые узоры, похожие на фактуру камня или листву. Кроме того, с помощью IFS или L-систем можно имитировать структуры типа папоротников, ветвей, геометрических узоров ткани.

- **Растительность и био-структуры**: L-системы традиционно используют для моделирования деревьев и растений (каждое ветвление повторяется по алгоритму). Сочетание L-систем и фрактальных преобразований позволяет генерировать леса, кустарники и даже сети кровеносных сосудов (они фрактальны по природе). С помощью шаровидного фрактализма или random fractal можно «рассыпать» листья/ветки случайно, создавая настраиваемый фотореалистичный эффект.

- **LOD (уровни детализации)**: для рендеринга больших сцен часто применяют разные уровни детализации фракталов. При удалении от камеры используется «упрощённый» фрактальный рельеф

(меньший уровень итераций), а при приближении – более детализированный. Так достигается баланс качества и скорости отрисовки. В GPU-шейдерах фрактальный шум вычисляется на лету для каждой вершины или пикселя, что позволяет генерировать ландшафт «по запросу» без хранения массивов высот.

- **Оптимизация (GPU)**: классические алгоритмы фрактальной генерации легко параллелятся. Например, расчёт шумовых функций (Perlin, Simplex) эффективно реализуется в GLSL/GLSL ES и используется в шейдерах процедурной воды, огня, облаков и т.д. Многие графические процессоры имеют функции «воксельного шума» или доступ к текстурам градиентов. Более того, фрактальный рельеф можно генерировать с помощью геометрических шейдеров или вычислительных шейдеров (compute shaders) прямо на GPU, что разгружает CPU.

В целом, фрактальные методы позволяют **автоматически создавать большие объёмы «игрового контента»** и обеспечивать вариативность без ручного моделирования. Так, в играх и симуляциях фрактальные алгоритмы применяют для генерации диких ландшафтов, облачных покровов, коралловых рифов, городских пятен (вида «размытый фрактал» строения улиц) и т.д.

Ограничения и артефакты

Фрактальные алгоритмы имеют и ограничения:

- **Артефакты дискретизации**: из-за итеративной природы на экране иногда проявляются повторяющиеся узоры или «шаги», особенно при недостатке итераций. Это может приводить к видимым границам уровней масштабирования (особенно у escape-time фракталов, если палитра плохо сглажена).

- **Битовые разрешения**: при больших масштабах (zoom) многие алгоритмы требуют высокоточной арифметики (особенно для множеств Мандельброта) — иначе мелкие детали «теряются» или искажаются.

- **Производительность**: генерация фракталов может быть дорогой. Escape-time фракталы трудно ускорить без потери качества (симметрии Мандельброта). Итерационные алгоритмы и L-системы экспоненциально растут по сложности с глубиной рекурсии. Шум Перлина потребляет много операций на каждом пикселе (но хорошо оптимизируем на GPU).

- **Контроль и натуральность**: фрактальные методы редко обеспечивают полный контроль над результатом. Они задают «статистику» формы, но отдельные детали носят случайный характер. Для «идеальных» текстур или объектов с конкретной формой фракталы могут выглядеть «слишком случайными» или «повторяющимися».

Несмотря на это, грамотный подбор алгоритмов и параметров (число итераций, коэффициенты случайности, тип интерполяции) позволяет минимизировать артефакты. Обычно фрактальную генерацию сочетают с другими методами: шумные фракталы фильтруются и комбинируются, рельефы «разрушаются» алгоритмами эрозии, цветовые схемы накладываются постфильтрами.

Сравнение с другими методами

Фрактальная генерация — особый класс процедурных методов, отличающийся рекурсивной природой. Её можно сравнить с:

- **Классическими моделями** (сплайны, NURBS): фракталы лучше передают бесконечную сложность, но менее интуитивны в управлении. Параметрические поверхности нужны для точных форм (детали автомобилей, архитектура), а фракталы — для «хаотичных» природных структур.

- **Стохастические симуляции** (OpenGL FX, физически базированные модели): фракталы проще в реализации и гораздо быстрее, но не учитывают физику (например, стохастическая модель облаков даёт похожий вид, но не метеорологически «правильный»).

- **Зернистые алгоритмы и клеточные автоматы** (например, шум Уорли для текстур, алгоритм Диффузионной дендритной кристаллизации для снега): эти методы хорошо моделируют специфические эффекты. Фракталы же более универсальны для общего сглаживания форм и текстур.

В каждом случае выбор зависит от требований сцены: реализма, управляемости, производительности. Фрактальные алгоритмы часто используются *в комплексе* с другими подходами: например, генерация базового рельефа фракталом, а детализация — ситуативным шумом и процедурными текстурами.

Обзор программ фрактальной графики

Компания Fractal Design впервые ввела в компьютерный обиход понятие Natural Media, представляющая возможность имитации «естественных инструментов художника». Например: вы рисуете на виртуальном холсте линию зеленного цвета используя инструмент «кисть с масляной краской». Затем рядом рисуется линия крестного цвета. В результате на экране происходит смешивание красок как на бумаге или холсте.

Лидером на рынке фрактальной графики до недавнего времени являлась компания Meta Creations. Спектр ее продуктов охватывает многие области компьютерной графики.

Fractal Design Painter – программа для создания и обработки художественных растровых иллюстраций. Поддерживает многослойность изображений и возможность использования фильтров. Позволяет эмулировать большое число художественных инструментов: карандаши, кисти, разнообразные типы красок.

Fractal Design Expression комбинирует в себе растровую и векторную технику компьютерной графики. То есть рисуются векторные объекты, которые можно редактировать по опорным узлам, но каждой линии или фигуре можно назначить любой растровый тип кисти.

Fractal Design Detailer позволяет раскрашивать поверхности трехмерных моделей.

Fractal Design Poser позволяет интегрировать 2D-изображения, 3D-сцены, web-графику и анимацию.

Art Dabbler является средством обучения компьютерной графике и азам рисования.

Painter 3D используется для наложения иллюстраций и текстур на 3D-моделей последующей визуализацией.

Вот обзор программ фрактальной графики:

Fractal Design: Программа для создания и обработки высокохудожественных растровых иллюстраций, поддерживает многослойность и фильтры от Photoshop.

Ultra Fractal: Лучшее решение для создания уникальных фрактальных изображений профессионального качества, с подробной документацией и интерфейсом, напоминающим Photoshop.

Fractal Explorer: Программа для создания изображений фракталов и трехмерных аттракторов, позволяет генерировать фрактальные изображения на основе формул или с нуля.

ChaosPro: Бесплатный генератор двумерных фрактальных изображений, который позволяет создавать трехмерные представления и анимации.

Эти программы предлагают различные возможности для создания и обработки фрактальных изображений, что делает их популярными среди художников и исследователей.

8 Сжатие данных

Итак для получения растровых изображений фотографического качества трбуется высокое разрешение. Это, в свою очередь, сказывается на размерах графических файлов,

требующих (если не предпринимать специальных мер) для своего хранения от нескольких единиц до нескольких десятков мегабайтов памяти. Единственный выход из положения — сжатие информации.

Алгоритмы сжатия изображений

Поскольку любой используемой нами информации (в том числе и графической) свойственна избыточность (особенно огромным видеофайлам), сжатие позволяет значительно уменьшить ее объем. Степень сжатая может колебаться от 4:1 до 200:1 — это зависит от типа данных и применяемого алгоритма. Более чем пятидесятикратное сжатие можно применять для звуковых или видеофайлов, но оно связано с потерей качества. Существует множество разных алгоритмов сжатия, учитывающих те или иные особенности сжимаемой информации. Однако алгоритма, одинаково хорошо сжимающего файлы любых форматов, пока не создано.

С самых общих позиций все существующие алгоритмы сжатия можно разбить на два больших класса:

- сжатие с потерями;

- сжатие без потерь.

Сжатие без потерь

Большинство схем сжатия без потерь основано на поиске в растровом изображении повторяющихся пиксельных узоров. Такой узор можно запомнить один раз и впоследствии повторить его необходимое количество раз. Подобные схемы сжатия полностью — пиксел за пикселом — восстанавливают исходное изображение. При этом в исходных данных ничего не отбрасывается и не теряется. Метод сжатия без потерь (например, используемый в форматах GIF или TIFF) очень эффективен для растровых рисунков, содержащих большие области однотонной закраски, или повторяющихся растровых узоров. В таких случаях чаще всего достигается коэффициент сжатия 10:1.

В основе алгоритмов сжатия без потерь лежат несколько методов. Рассмотрим наиболее распространенные из них.

Метод сжатия RLE (run length encoding) — кодирование с переменной длиной с троки. Этот алгоритм является одним из простейших. В основе его принципа действия заложен механизм поиска одинаковых пикселов в одной строке. Алгоритм RLE хорошо работает с искусственными картинками и плохо — с фотографиями. В действительности, если фотография детализирована, RLE может даже увеличить размер файла. В настоящее время этот алгоритм используется для сжатия информации в PSD-формате.

Метод сжатия LZW (Lempel-Ziv-Welch) разработан в 1978 году израильтянами Лемпелом и Зивом и доработан позднее в США: Сжимает данные путем поиска одинаковых последовательностей (называемых фразами) во всем файле. Затем выявленные последовательности сохраняются в таблице, где им присваиваются более короткие маркеры (ключи). Например, если в изображении имеются наборы из пурпурного, оранжевого и зеленого пикселов, повторяющиеся 50 раз, LZW выявляет это, присваивает данному набору отдельное число (например. 7) и затем сохраняет эти данные 50 раз в виде числа 7. Метод LZW так же как и RLE. лучше работает на однородных участках, свободных от цветового шума. Однако по сравнению с RLE-алгоритмом он лом случае происходит медленнее. Механизм LZW -компрессии используется в формате TIFF? а также в одном из основных форматов сети Интернет — GIF.

Сжатие с потерями

Как уже отмечалось, использование алгоритмов сжатия без потерь оказывается неэффективным для растровых изображений фотографического качества, в которых каждый пиксел отличается от соседних. Применение механизма сжатия узоров к изображениям, на которых повторяющихся узоров нет. часто приводит к ничтожным результатам при больших затратах времени.

Сжатие с потерями, наоборот, лучше всего работает с теми изображениями, на которых нет повторяющихся узоров или больших областей однотонной закраски. В растровом рисунке, который содержит множество слегка отличающихся друг от друга пикселов (например, 100 немного отличающихся оттенков голубою цвета неба), большие области могут заполняться пикселами одного цвета или пиксельным узором, имитирующим вид исходной области.

Ключевым моментом в применении сжатия с потерями является определение «приемлемою уровня» потерь. Уровень этот субъективен и зависит от изображения-оригинала и от того, как он будет использоваться. Если ваше оригинальное изображение — фотография музейного качества, предназначенная для публикации в ысокохудожественном издании, то ни о каких «приемлемых потерях» не может рыть и речи. Рисунок должен быть воспроизведен как можно точнее. Другое дело электронная публикация на Web-странице, где одним из главных критериев является малый размер файла.

В настоящее время создано несколько алгоритмов сжатия с потерями, самым известным и; которых является JPEG.

В последнее время создан новый алгоритм сжатия — WI (Wavelet Compressed Bitmap), в котором применяется схема сжатия, близкая к схеме сжатия JPEG. Однако в основу его работы заложено использование нового и очень перспективного математического аппарата — вейвлетов. Пока «на официальном» уровне этот алгоритм поддерживается только продуктами фирмы Corel.

JPEG

На сегодняшний день формат JPEG (Joint Photographic Experts Group) является одним из наиболее распространенных графических форматов для сжатия файлов. Как уже отмечалось, в нем реализован алгоритм сжатия с потерями. Это означает, что в процессе сжатия изображения происходит частичная потеря хранящейся в файле информации. Поэтому в процессе применения этой процедуры сжатия приходится искать, компромисс между степенью сжатия и качеством сохраняемого изображения. Чем больше сжатие, тем ниже качество, и наоборот.

Строго говоря, JPEG не формат, а алгоритм сжатия, в основе которого лежит не поиск одинаковых элементов, как в случае RLE и LZW, а поиск разницы между пикселами.

Кодирование данных с помощью используемого в JPEG алгоритма сжатия осуществляется в несколько этапов.

1. Сначала графические данные конвертируются в цветовое пространство типа LAB.
2. Затем отбрасывается половина или три четверти информации о цвете (в зависимости от реализации алгоритма).
3. Далее анализируются блоки размером 8x8 пикселов. Для каждого блока формируется набор чисел. Первые несколько чисел представляют цвет блока в целом, в то время как последующие числа отражают тонкие делали. Поскольку спектр деталей базируется на зрительном восприятии человека, то крупные детали более заметны.

4. На следующем этапе в зависимости от выбранного вами уровня качества, отбрасывается определенная часть чисел, характеризующих тонкие детали.
5. На последнем этапе используется кодирование методом Хаффмана для более эффективного сжатия конечных данных.

Алгоритм сжатия Хаффмана (Huffman) разработан еще в 1952 году. В нем осуществляется последовательный перебор наборов символов, которые анализируются с целью определения частоты появления каждого символа. Затем наиболее часто встречающиеся символы кодируются с помощью минимально возможного количества битов. Например, в английских текстах чаще всего встречается буква «е». Используя кодировку Хаффмана, вы можете представить «е» всего лишь двумя битами (1 и 0) вместо восьми битов, необходимых для представления буквы «е» в кодировке ASCII

Восстановление данных происходит в обратном порядке. Таким образом, чем выше уровень компрессии, тем больше данных отбрасывается и тем ниже качество. Используя JPEG, можно получить файл в 1-500 раз меньше, чем BMP. Этот формат аппаратно независим, полностью поддерживается и PC и Macintosh, однако он относительно новый и не понимается старыми программами (до 1995 года).

JPEG не поддерживает индексированные палитры цветов. Первоначально в спецификациях формата не было поддержки цветовой модели CMYK и только в последние годы фирмой Adobe добавлена по,тдержка цветоделения.

Наряду со стандартным вариантом существуют еще два подтипа формата JPEG , ориентированных на применение в Интернете.

Baseline Optimized — файлы этого подтипа формата несколько лучше сжимаются, но не читаются некоторыми программами. Однако все основные браузеры его поддерживают.

Progressive JPEG также разработан специально для сети, его файлы меньше стандартных, но чуть больше подтипа формата Baseline Optimized. Главная особенность Progressive JPEG состоит в поддержке чересстрочного вывода изображения (использование этого свойства намного сокращает время передачи и вывода на экран насыщенных графикой web-страниц).

При сохранении графических изображений в формате JPEG- следует учитывать следующее.

JPEG лучше подходит для сжатия растровых картинок фотографического качества, чем для логотипов или схем. Это связано с тем, что в них больше полутоновых переходов, в то время как при сжатии однотонных заливок появляются нежелательные помехи.

Лучше сжимаются (и с меньшими потерями) большие изображения для web и изображения с высоким разрешением для печати (200-300 dpi и более), так как в каждом квадрате (8x8 пикселов) переходы получаются более мягкими за счет большего числа квадратов в таких файлах.

Нежелательно сохранять в JPEG -формате любые изображения, в которых важны тонкие нюансы цветопередачи (репродукции), так как во время сжатия происходит отбрасывание цветовой информации.

Этот формат следует использовать только для сохранения конечного варианта работы, потому что каждое последующее сохранение приводит к новым потерям (отбрасыванию) данных.

Алгоритм фрактального сжатия изображений

Фрактальная архивация основана на том, что с помощью коэффициентов системы итерируемых функций изображение представляется в более компактной форме. Прежде чем рассматривать процесс архивации, разберем, как IFS строит изображение.

Строго говоря, IFS - это набор трехмерных аффинных преобразований, переводящих одно изображение в другое. Преобразованию подвергаются точки в трехмерном пространстве (x координата, y координата, яркость).

Наиболее наглядно этот процесс продемонстрировал сам Барнсли в своей книге “Фрактальное сжатие изображения”. В ней введено понятие Фотокопировальной Машины, состоящей из экрана, на котором изображена исходная картинка, и системы линз, проецирующих изображение на другой экран. Каждая линза проецирует часть исходного изображения. Расставляя линзы и меняя их характеристики, можно управлять получаемым изображением. На линзы накладывается требование они должны уменьшать в размерах проектируемую часть изображения. Кроме того, они могут менять яркость фрагмента и проецируют не круги, а области с произвольной границей.

Одна шаг Машины состоит в построении с помощью проецирования по исходному изображению нового. Утверждается, что на некотором шаге изображение перестанет изменяться. Оно будет зависеть только от расположения и характеристик линз и не будет зависеть от исходной картинки. Это изображение называется неподвижной точкой или аттрактором данной IFS. Collage Theorem гарантирует наличие ровно одной неподвижной точки для каждой IFS. Поскольку отображение линз является сжимающим, каждая линза в явном виде задает самоподобные области в нашем изображении. Благодаря самоподобию мы получаем сложную структуру изображения при любом увеличении.

Наиболее известны два изображения, полученных с помощью IFS треугольник Серпинского и папоротник Барнсли. Первое задается тремя, а второе - пятью аффинными преобразованиями (или, в нашей терминологии, линзами). Каждое преобразование задается буквально считанными байтами, в то время, как изображение, построенное с их помощью, может занимать и несколько мегабайт.

Становится понятно, как работает архиватор, и почему ему требуется так много времени. Фактически, фрактальная компрессия - это поиск самоподобных областей в изображении и определение для них параметров аффинных преобразований.

В худшем случае, если не будет применяться оптимизирующий алгоритм, потребуется перебор и сравнение всех возможных фрагментов изображения разного размера. Даже для небольших изображений при учете дискретности мы получим астрономическое число перебираемых вариантов. Даже резкое сужение классов преобразований, например, за счет масштабирования только в определенное число раз, не позволит добиться приемлемого времени. Кроме того, при этом теряется качество изображения. Подавляющее большинство исследований в области фрактальной компрессии сейчас направлены на уменьшение времени архивации, необходимого для получения качественного изображения.

До сих пор мы не затронули несколько важных вопросов. Например, что делать, если алгоритм не может подобрать для какого-либо фрагмента изображения подобный ему? Достаточно очевидное решение - разбить этот фрагмент на более мелкие и попытаться поискать для них. Однако понятно, что процедуру эту нельзя повторять до бесконечности, иначе количество необходимых преобразований станет так велико, что алгоритм перестанет быть алгоритмом компрессии. Следовательно, допускаются потери в какой-то части изображения.

Для фрактального алгоритма компрессии, как и для других алгоритмов сжатия с потерями, очень важны механизмы, с помощью которых можно будет регулировать степень сжатия и степень потерь. К настоящему времени разработан достаточно большой набор

таких методов. Во-первых, можно ограничить количество преобразований, заведомо обеспечив степень сжатия не ниже фиксированной величины. Во-вторых, можно потребовать, чтобы в ситуации, когда разница между обрабатываемым фрагментом и наилучшим его приближением будет выше определенного порогового значения, этот фрагмент дробился обязательно (для него обязательно заводится несколько линз). В третьих, можно запретить дробить фрагменты размером меньше, допустим, четырех точек. Изменяя пороговые значения и приоритет этих условий, можно очень гибко управлять коэффициентом компрессии изображения: от побитного соответствия, до любой степени сжатия.

Возможности масштабирования

Итак, мы выяснили, что IFS задает фрактальную структуру, сколь угодно близкую к нашему изображению. При внимательном рассмотрении процесса построения изображения с ее помощью становится понятно, что восстанавливаемое изображение может иметь любое (!) разрешение. В самом деле, возвращаясь к аналогии с Фотокопировальной Машиной, можно сказать, что нам не важно до какой сетки растра будет огрубляться установившееся неподвижное изображение. Ведь Машина работает вообще с непрерывными экранами.

На этапе архивации проводится распознавание изображения, и в виде коэффициентов хранится уже не растровая информация, а информация о структуре самого изображения. Именно это и позволяет при развертывании увеличивать его в несколько раз. Особенно впечатляют примеры, в которых при увеличении изображений природных объектов проявляются новые детали, действительно этим объектам присущие (например, когда при увеличении фотографии скалы она приобретает новые, более мелкие неровности).

Но не все так гладко, как может показаться. Если изображение однородно (на фотографии только скала), то при увеличении получаются отличные результаты, однако, если сжимать изображение натюрморта, то предсказать, какие новые фрактальные структуры возникнут, очень сложно. Впрочем, вдвое-втрое можно увеличить практически любое изображение, при архивации которого задавалась небольшая степень потерь.

Масштабирование - уникальная особенность, присущая фрактальной компрессии. Со временем ее, видимо, будут активно использовать как в специальных алгоритмах масштабирования, так и во многих приложениях. Действительно, этого требует концепция "приложение в окне". Было бы неплохо, если бы изображение, показываемое в окне 100x100, хорошо смотрелось при увеличении на полный экран - 1024x768.

Сравнение JPEG и алгоритма фрактального сжатия

Сегодня наиболее распространенным алгоритмом архивации графики является JPEG. Сравним его с фрактальной компрессией.

Во-первых, заметим, что и тот, и другой алгоритм оперируют 8-битными (в градациях серого) и 24-битными полноцветными изображениями. Оба являются алгоритмами сжатия с потерями и обеспечивают близкие коэффициенты архивации. И у фрактального алгоритма, и у JPEG существует возможность увеличить степень сжатия за счет увеличения потерь. Кроме того, оба алгоритма очень хорошо распараллеливаются.

Различия начинаются, если мы рассмотрим время, необходимое алгоритмам для архивации/разархивации. Так, фрактальный алгоритм сжимает в сотни и даже в тысячи раз дольше, чем JPEG. Распаковка изображения, наоборот, произойдет в 5-10 раз быстрее. Поэтому, если изображение будет сжато только один раз, а передано по сети и распаковано множество раз, то выгодней использовать фрактальный алгоритм.

JPEG использует разложение изображения по косинусоидальным функциям, поэтому потери в нем (даже при заданных минимальных потерях) проявляются в волнах и ореолах на границе резких переходов цветов. Именно за этот эффект его не любят использовать при

сжатии изображений, которые готовят для качественной печати: там этот эффект может стать очень заметен.

Фрактальный алгоритм избавлен от этого недостатка. Более того, при печати изображения каждый раз приходится выполнять операцию масштабирования, поскольку растр (или линиатура) печатающего устройства не совпадает с растром изображения. При преобразовании также может возникнуть несколько неприятных эффектов, с которыми можно бороться либо масштабируя изображение программно (для дешевых устройств печати типа обычных лазерных и струйных принтеров), либо снабжая устройство печати своим процессором, винчестером и набором программ обработки изображений (для дорогих фотонаборных автоматов). Как можно догадаться, при использовании фрактального алгоритма таких проблем практически не возникает.

Вытеснение JPEG фрактальным алгоритмом в повсеместном использовании произойдет еще не скоро (хотя бы в силу низкой скорости архивации последнего), однако в области приложений мультимедиа, в компьютерных играх его использование вполне оправдано.

Алгоритмы сжатия изображений: с потерями, без потерь, JPEG, фрактальное сжатие изображений

Введение

Объём графической информации постоянно растёт. Современные цифровые камеры, смартфоны, системы видеонаблюдения и графические редакторы создают изображения большого размера. Хранение и передача таких данных требуют значительных ресурсов памяти и пропускной способности каналов связи.

Для решения этой проблемы применяются методы сжатия данных. Сжатие позволяет уменьшить объём файла без существенного ухудшения качества изображения или вовсе без потери информации.

Классификация алгоритмов

По природе восстановления данных алгоритмы сжатия делят на **без потерь** (lossless) и **с потерями** (lossy). К безпотерьным относятся методы, при которых исходные пиксели полностью восстанавливаются (например, кодирование повторов **RLE**, **Хаффман**, **LZW**, DEFLATE/PNG, арифметическое кодирование и др.). Лоссовые алгоритмы в процессе кодирования «отбрасывают» малозаметную информацию, что даёт более сильное сжатие (например, JPEG, JPEG2000, современные кодеки HEVC/AV1). Также существуют гибридные схемы, комбинирующие эти подходы (большинство современных кодеков проводят сначала лоссовую «трансформацию и квантование», а потом – без потерьное энтропийное кодирование результатов).

- **Без потерь:** RLE (Run-Length Encoding), код Хаффмана, алгоритмы Лемпеля-Зива (LZ77, LZ78, LZW), PNG/DEFLATE, арифметическое кодирование. Эти методы подходят для документов, сканов, иллюстраций и научных изображений, где важна точная реконструкция.
- **С потерями:** JPEG (DCT-кодирование), JPEG2000 (волновой анализ), форматы WebP/AVIF/HEIF (используют видео-кодеки VP8/VP9, AV1, HEVC), BPG, JPEG XL и т.д. Применяются для фотографий, видео, где допустимы искажения ради большой экономии места.
- **Гибридные:** Алгоритмы, объединяющие прогнозирование/разложение сигнала и эффективное кодирование. Например, JPEG сначала цветовым преобразованием и

DCT, затем квантует и кодирует результаты Хаффманом (либо арифметикой) – таким образом сочетая lossy (DCT+квантование) и lossless (энропийный код) шаги.

Алгоритмы

RLE (Run-Length Encoding): самый простой безпотерьный метод. Последовательности одинаковых пикселей («прогоны») заменяются парой «значение + длина». Эффективен для изображений с большими однотонными участками (например, скриншоты), но не работает на сложных фотографиях.

Кодирование Хаффмана: строится на частотном анализе. Алгоритм вычисляет частоту каждого символа (например, байта интенсивности) и строит оптимальное дерево префиксного кода, где более частым символам соответствуют более короткие коды. Хаффман-код – префиксный оптимальный безпотерьный код, широко используется в PNG, JPEG (после DCT) и др. Его сложность построения – порядка O(n log n) от числа символов.

LZW (Lempel–Ziv–Welch): универсальный безпотерьный алгоритм, появившийся в 1984 г. как усовершенствование LZ78. LZW динамически строит словарь повторяющихся фрагментов входной последовательности. При кодировании он заменяет найденные фразы на индексы в словаре. LZW прост в реализации и обеспечивает хорошее сжатие текстов и графики (GIF использует LZW). Часто удаётся достичь сжатия ~2:1 и выше при умеренной скорости кодирования.

PNG/DEFLATE (LZ77+Хаффман): PNG – графический формат сжатия без потерь. Он использует фильтрацию строк и алгоритм *DEFLATE* (обобщение LZ77 с последующим Хаффман-кодированием). DEFLATE сначала ищет повторяющиеся подстроки с использованием скользящего окна (LZ77-подобно), затем строит оптимальный Хаффман-код для сжатия коэффициентов матчей и литералов. PNG обеспечивает хорошее сжатие для фотографий и сканов без артефактов.

Арифметическое кодирование: энтропийный метод безпотерьного кодирования, который в отличие от Хаффмана кодирует не отдельные символы, а всю последовательность как единый дробный код. Каждому символу соответствует отрезок [p; q) исходящего интервала [0,1), который последовательно уточняется при чтении символов. Арифметическое кодирование позволяет приближаться к предельному (энтропийному) уровню сжатия и часто превосходит Хаффман, особенно на данных с нецелочисленными вероятностями. Недостаток – большая вычислительная сложность и проблем с побитовыми погрешностями, но современные реализации эффективны и использованы во многих современных кодеках (например, JPEG2000, HEVC, AV1).

JPEG (DCT + квантование): стандарт 1992 г. для фотосжатия с потерями. Изображение преобразуется в цветовое пространство Y′CbCr, где яркостной компоненте Y′ обычно отводится больше разрешения (человеческий глаз менее чувствителен к оттенкам цвета). Затем картинка делится на блоки 8×8 пикселей, и к каждому блоку применяется двумерное дискретное косинусное преобразование (DCT).

После DCT происходит **квантование**: каждый коэффициент делится на соответствующий элемент матрицы квантования и округляется.

Высокие частоты при этом сильно обрезаются, поскольку человеческий глаз менее чувствителен к тонким деталям. Низкое значение «качества» означает большие значения матрицы квантования , больше искажений и сильное сжатие. После квантования получает 8×8 блок целых чисел, который дальше кодируется без потерь (обычно Хаффманом или арфметическим методом). Декодирование выполняет обратные операции (IDCT) с потерей информации из-за округления. В результате JPEG обеспечивает высокую степень сжатия (типично 10:1…20:1) с заметными блоковыми артефактами при сильном сжатии.

- **JPEG2000 (волновое преобразование):** в конце 1990-х JPEG-2000 задумали как преемника JPEG, использующего *дискретное вейвлет-преобразование (DWT)* вместо DCT. Волновые фильтры (обычно Кофен-Дауич-Фёво 9/7 для лосс и 5/3 для лесс) применяются к изображению рекурсивно, создавая представление с частотной и разностной разреженностью. Затем коэффициенты вейвлет-преобразования квантируются и кодируются (в стандарте используется арифметическое кодирование по алгоритму JPEG2000). JPEG2000 позволяет добиться более гибкого кодирования (скальируемость по качеству и разрешению) и чуть лучшего качества при равном битрейте по

сравнению с JPEG. Однако кодеки JPEG2000 сложны и медленны. JPEG2000 артефакты – это главным образом «рябь» и размытость по краям, но **без** характерной блоковой решётки JPEG.

- **WebP:** формат от Google (2010), базируется на видеокодеке VP8. В **lossy-режиме** WebP использует подобие JPEG: цветовая модель YUV 4:2:0, предсказание и 4×4 DCT-блоки, квантование, затем код VP8 (аналог DCT+энропийный код). **Lossless-WebP** – собственный алгоритм (создан Алокуялайла): он использует предсказание пикселей и энтропийные коды для каждого канала, а также «цветовой кэш» часто встречающихся оттенков. WebP поддерживает прозрачность (алфа-канал) и анимацию. Исследования показали, что lossy-WebP при равном SSIM даёт файлы на 25–34 % меньше, чем JPEG. Lossless-WebP обычно на 26 % меньше PNG того же изображения.

- **HEIF/HEIC (HEVC):** HEIF – контейнер MPEG (стандарт ISO/IEC 23008-12) для изображений. В нём часто хранят кодированные изображения в формате **HEVC (H.265)**, называемом HEIC. HEVC использует 4×4 и 8×8 транспозиции и продвинутую предсказательную кодировку. HEIC поддерживает HDR, 10–16 бит цветности, несколько кадров (анимация), тайлы, метаданные. На практике HEIC (чаще Apple) даёт файл примерно вдвое меньше JPEG при равном качестве. Недостаток – сложность кодирования и лицензионные ограничения (патенты HEVC).

- **AVIF (AV1):** формат образа AV1 (AOMedia) – аналог HEIC, но основан на кодеке **AV1** (разработка Alliance for Open Media, 2018). AV1 – более новая разработка, бесплатная от патентных отчислений. AVIF поддерживает lossless и lossy, до 12 бит цвета, HDR, анимацию, прозрачность. AVIF превосходит WebP: при том же качестве файлы могут быть примерно на 20–30 % меньше. Недостаток AVIF – очень высокая вычислительная сложность (кодирование медленное без аппаратной поддержки).

1. Сжатие изображений без потерь

Сжатие без потерь представляет собой метод кодирования данных, при котором после восстановления получается изображение, полностью совпадающее с исходным.

Ни один пиксель не изменяется и не теряется.

Такие методы используются в случаях, когда важна абсолютная точность данных:

- медицинские изображения;
- архивное хранение;
- технические чертежи;
- схемы;
- документы;
- изображения для последующей обработки.

Основная идея

Во многих изображениях присутствует избыточность данных.

Например, длинные последовательности одинаковых пикселей могут быть записаны более компактно.

Вместо хранения каждого элемента отдельно сохраняется информация о повторениях или закономерностях.

Метод RLE

RLE расшифровывается как Run Length Encoding.

Метод основан на кодировании серий одинаковых значений.

Рассмотрим последовательность:

AAAAAAABBBBBCCCC

Вместо записи каждого символа отдельно можно записать:

7A 5B 4C

Количество данных уменьшается.

Метод особенно эффективен для:

- чертежей;
- схем;
- простых рисунков;
- изображений с большими одноцветными областями.

Для фотографий эффективность обычно невысока.

Кодирование Хаффмана

Метод предложен американским учёным Дэвидом Хаффманом в 1952 году.

Идея заключается в том, что часто встречающиеся символы кодируются короткими битовыми последовательностями, а редкие символы получают более длинные коды.

Например:

Символ	Частота	Код
A	высокая	0
B	средняя	10
C	низкая	110
D	низкая	111

В результате средняя длина кодового слова уменьшается.

Алгоритм Хаффмана широко применяется в современных форматах хранения данных и является частью стандарта JPEG.

LZW

Алгоритм LZW был разработан Абрахамом Лемпелем, Якобом Зивом и Терри Уэлчем.

Метод строит словарь повторяющихся последовательностей.

Вместо многократного хранения одинаковых фрагментов сохраняются ссылки на словарь.

Алгоритм используется в форматах:

- GIF;
- TIFF;
- некоторых архиваторах.

Преимущество метода заключается в высокой скорости работы и отсутствии потерь информации.

Формат PNG

Одним из наиболее известных форматов сжатия без потерь является PNG.

PNG использует:

- предсказание значений пикселей;
- алгоритм DEFLATE.

DEFLATE объединяет:

- LZ77;
- кодирование Хаффмана.

Преимущества PNG:

- отсутствие потерь качества;
- поддержка прозрачности;
- хорошее сжатие графики и интерфейсов.

Недостаток заключается в том, что фотографии обычно занимают значительно больше места по сравнению с JPEG.

---

2. Сжатие изображений с потерями

Сжатие с потерями допускает частичное удаление информации.

Предполагается, что человеческий глаз не способен заметить некоторые изменения изображения.

За счёт удаления малозаметных деталей достигается гораздо более высокая степень сжатия.

После восстановления изображение отличается от оригинала.

Однако эти различия могут быть практически незаметны.

Причины использования

Современные фотографии содержат огромное количество информации.

Например, фотография размером 4000 × 3000 пикселей содержит:

12 миллионов пикселей.

При глубине цвета 24 бита объём несжатого изображения составляет примерно 36 мегабайт.

Передавать и хранить такие файлы неудобно.

Поэтому используются методы сжатия с потерями.

Особенности человеческого зрения

Многие алгоритмы с потерями основаны на особенностях восприятия человека.

Человеческий глаз:

- лучше различает яркость, чем цвет;
- хуже замечает мелкие изменения цветовых оттенков;
- менее чувствителен к высокочастотным деталям.

Эти особенности используются при удалении части информации.

---

3. Формат JPEG

JPEG является самым распространённым форматом хранения фотографий.

Название происходит от организации:

Joint Photographic Experts Group

Первый международный стандарт JPEG был утверждён в 1992 году.

Общая схема работы JPEG

Алгоритм состоит из нескольких этапов:

1. Преобразование цветового пространства.
2. Субдискретизация цвета.
3. Разбиение изображения на блоки.
4. Дискретное косинусное преобразование.
5. Квантование.
6. Кодирование.

Преобразование RGB в YCbCr

Большинство изображений хранится в модели RGB.

JPEG преобразует изображение в пространство YCbCr.

Здесь:

- Y — яркость;
- Cb — синяя цветовая компонента;
- Cr — красная цветовая компонента.

Такое представление позволяет отдельно обрабатывать яркость и цвет.

Субдискретизация цвета

Поскольку глаз лучше воспринимает яркость, цветовую информацию можно хранить менее подробно.

Наиболее распространённый вариант:

4:2:0

При этом количество цветовых данных уменьшается почти в четыре раза.

Визуально различия часто остаются незаметными.

Разбиение на блоки

Изображение делится на блоки размером:
8 × 8 пикселей.
Каждый блок обрабатывается независимо.

Дискретное косинусное преобразование

Следующий этап называется DCT.

Для каждого блока вычисляется набор коэффициентов, описывающих пространственные частоты изображения.

Низкие частоты содержат основную информацию о структуре изображения.

Высокие частоты описывают мелкие детали и шум.

После преобразования большая часть энергии изображения сосредотачивается в нескольких коэффициентах.

Квантование

Квантование является главным источником потерь в JPEG.

Коэффициенты DCT делятся на значения специальной таблицы квантования и округляются.

Многие высокочастотные коэффициенты становятся равными нулю.

Именно благодаря этому достигается значительное уменьшение объёма данных.

Чем сильнее квантование:

- меньше размер файла;
- ниже качество изображения.

Кодирование

После квантования данные дополнительно сжимаются.

Для этого применяются:

- RLE;
- кодирование Хаффмана.

Таким образом достигается ещё большее уменьшение размера файла.

Преимущества JPEG

Преимущества:

- высокая степень сжатия;
- широкая поддержка;
- удобство хранения фотографий;
- высокая скорость обработки.

Недостатки JPEG

Недостатки:

- потери качества;
- невозможность полного восстановления оригинала;
- появление артефактов при сильном сжатии;
- ухудшение качества при многократном пересохранении.

Типичными артефактами являются:

- блочность;
- размытость деталей;
- ложные контуры.

---

4. Фрактальное сжатие изображений

Фрактальное сжатие является отдельным направлением исследований в области обработки изображений.

Метод появился в конце 1980-х годов.

Основной вклад в развитие технологии внёс американский математик Michael Barnsley.

Понятие фрактала

Фракталом называется объект, обладающий свойством самоподобия.

Это означает, что отдельные части объекта напоминают его целиком.

Примеры:

- снежинки;
- береговые линии;
- облака;
- листья растений.

Во многих изображениях также можно обнаружить элементы самоподобия.

Основная идея метода

Вместо хранения всех пикселей сохраняется набор математических преобразований.

Алгоритм ищет участки изображения, похожие друг на друга.

Если один фрагмент можно получить из другого при помощи:

- масштабирования;
- поворота;
- отражения;
- изменения яркости,

то сохраняются параметры преобразования, а не сами пиксели.

Процесс кодирования

Сначала изображение делится на блоки.

Затем выполняется поиск похожих областей.

Для каждого блока определяется преобразование, которое наиболее точно воспроизводит данный участок.

В файл записываются параметры найденных преобразований.

Восстановление изображения

При декодировании запускается итерационный процесс.

Преобразования многократно применяются к начальному изображению.

Через несколько итераций формируется итоговое изображение.

Этот процесс основан на теории итерационных функциональных систем.

Преимущества фрактального сжатия

Преимущества метода:

- высокая степень сжатия для некоторых типов изображений;
- возможность масштабирования изображения с меньшими визуальными потерями;
- компактное хранение данных.

Недостатки фрактального сжатия

Недостатки существенно ограничили распространение технологии.

Основные проблемы:

- чрезвычайно длительное кодирование;
- высокая вычислительная сложность поиска похожих областей;
- отсутствие существенных преимуществ перед современными методами;
- ограниченная поддержка программным обеспечением.

По этой причине фрактальное сжатие не получило такого распространения, как JPEG.

Сравнение методов

Характеристика	Без потерь	JPEG	Фрактальное
Потеря данных	Нет	Да	Да
Восстановление оригинала	Полное	Неполное	Неполное
Скорость кодирования	Высокая	Высокая	Низкая
Размер файла	Больше	Меньше	Может быть очень малым
Использование	Документы, схемы	Фотографии	Специальные задачи

Заключение

Сжатие изображений является важной частью компьютерной графики. Методы без потерь обеспечивают полное сохранение информации и применяются там, где недопустимо изменение данных. Методы с потерями позволяют значительно уменьшить объём файлов за счёт удаления малозаметной информации.

Наиболее распространённым форматом остаётся JPEG. Его популярность обусловлена хорошим соотношением качества изображения и размера файла. Фрактальное сжатие представляет значительный научный интерес и основано на принципах самоподобия, однако высокая вычислительная сложность ограничила его практическое применение.

Таким образом, выбор метода сжатия зависит от требований к качеству изображения, скорости обработки и объёму хранимых данных.

Практические рекомендации

- **Выбор формата зависит от задачи:**
- Если требуется **максимальное качество без потерь** (например, архивирование, научные данные, логотипы), используйте PNG, TIFF или безпотерьные режимы WebP/AVIF.
- Для **фотографий** с хорошим качеством и умеренным размером: JPEG (совместимость) либо современные форматы. Если требуется альфа-канал – PNG (без потерь) или lossy WebP/AVIF (с поддержкой прозрачности).
- Для **веб-изображений** лучше WebP или AVIF: они обеспечивают меньший размер при сопоставимом качестве. По данным Google, **WebP** даёт файлы на 25–34 % меньше, чем JPEG при равном SSIM[1]; **AVIF** ещё на ~20 % меньше, чем WebP[21]. Однако поддержка AVIF моложе, и кодирование медленнее. Если важна совместимость – WebP или PNG.
- **Мобильные устройства:** на iOS обычно стандарт HEIC (HEVC), на Android – HEIF/AVIF (нужны библиотеки поддержки). Однако для широкой совместимости (со старыми ОС/браузерами) часто вручную конвертируют в JPEG/PNG.
- **Видео и анимация:** неподвижные кадры можно кодировать теми же методами (например, Motion-JPEG2000 для видео).
- **Качество против скорости:** если нужно быстрое кодирование/декодирование (стриминг, реальное время), часто выбирают JPEG или JPEG XS. Если кодируетесь разово и важен размер – AVIF/HEIC/JPEG XL.
- **Используемые инструменты:** для практики удобно применять стандартные библиотеки и утилиты:
- **libjpeg (IJG)** – классика для JPEG (`cjpeg`, `djpeg`).

- **libpng**, **pngcrush**/`optipng` – для оптимизации PNG.
- **libwebp** (`cwebp`, `dwebp`) – WebP.
- **libavif** (`avifenc`, `avifdec`) – AVIF/HEIF.
- **FFmpeg** – универсальный конвертер, поддерживает многие кодеки (например, `ffmpeg -i in.png -c:v libwebp out.webp` или кодек x265/x264 для HEIC/AV1).

Эксперименты Google использовали командные утилиты `cwebp`, `dwebp`, `cjpeg`, `djpeg`, `ffmpeg` и др. для конвертации изображений и оценки качества[29].

История и перспективы

Хронология: первые идеи связаны с теориями информации 1950-х (коды Хаффмана, 1952), алгоритмов Лемпеля-Зива 1977–1984 (LZ77, LZW). С появлением цифровых камер и интернета в 1990-е возник JPEG (1992) и PNG (1996) – стандарты для фото и графики. В 2000-х появились JPEG2000 (2000, стандарт ISO/IEC 15444) и JPEG LS (монохромный безпотерьный JPEG). В 2010-е Google внедрил WebP (2010), затем MP4/HEVC стали расширяться на изображения (HEIC, 2017). Alliance for Open Media выпустила AVIF (2018) на базе AV1. 2020-е – эра новых стандартов: JPEG XL (2021) и первые *нейрокодеки* – в 2025 году опубликован стандарт **JPEG AI** (ITU-T T.840.1/ISO IEC 6048-1) для обученных нейросетевых кодеков. Ниже – условная временная шкала:

Перспективы: сегодня активно исследуются **нейросетевые кодеки**, которые обучаются совместно на сжатие и восстановление. Такие подходы (например, сжатие на базе вариационных автоэнкодеров или GAN) уже показывают выигрыши в качестве при высоких степенях сжатия. JPEG AI – первый официальный стандарт, предусматривающий использование нейросетей для изображений, и он обещает около 30 % экономии битрейта при том же субъективном качестве по сравнению с традиционными кодеками. Будущее – за гибридными решениями и ML-моделями, которые могут адаптивно выбирать оптимальный кодек для фрагмента изображения. Однако эти методы пока очень тяжёлые по вычислениям и требуют тренировки на больших базах. Широкую индустриализацию пока сдерживает лишь цена вычислений и стандартизация, но тренды явно движутся в сторону "умных" кодировщиков.