ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий
Кафедра компьютерных технологий

И. Г. Таранцев

КОМПЬЮТЕРНАЯ ГРАФИКА

Учебное пособие

Новосибирск
2017

Таранцев И. Г. Компьютерная графика: Метод. пособие / Новосиб. гос. ун-т. Новосибирск, 2017. 60 с.

Данное учебное пособие предназначено для подготовки специалистов в области разработки программных средств современных информационных технологий. Представленный материал позволяет ознакомится с базовыми знаниями и современным технологиям для работы с двумерной и трехмерной компьютерной графикой.

Пособие объединяет знания по двумерной и трехмерной компьютерной графике, позволяя за короткое время получить знания по самым различным областям. Ведь без четкого понимания принципов визуализации графической информации, как двумерной, так и трехмерной, невозможно адекватно использовать весь потенциал современных технических средств, предоставляемых разработчиками операционных систем и аппаратных средств визуализации.

Рецензент
доц., зам. зав. кафедрой АФТИ ФФ НГУ М. Ю. Шадрин

ОГЛАВЛЕНИЕ

1 ДВУМЕРНАЯ ГРАФИКА

1.1 Физиология цветового зрения.

Видимая часть спектра включает излучения с разной длиной волны, воспринимаемые глазом в виде различных цветов. Цветовое зрение обусловлено совместной работой нескольких светоприемников – фоторецепторов с разной спектральной чувствительностью. Фоторецепторы преобразуют энергию излучения в физиологическое возбуждение, которое воспринимается нервной системой как различные цвета. Спектральная чувствительность фоторецепторов разного типа различна и определяется спектром поглощения зрительных пигментов. Разные сочетания излучений с разными спектральными составляющими могут формировать одинаковые сигналы в одном светоприемнике. Но светоприемники разных типов будут формировать разные сигналы. Поэтому при сочетании нескольких типов светоприемников разные излучения не могут быть уравнены никаким подбором их интенсивностей.

Основы современных представлений о цветовом зрении человека разработаны в 19 веке английским физиком Т. Юнгом и немецким учёным Г. Гельмгольцем в виде так называемой трёхкомпонентной или трихроматической теории цветовосприятия. Согласно этой теории, в сетчатке глаза человека имеются три типа фоторецепторов (колбочковых клеток), чувствительных в разной степени к красному, зелёному и синему свету. Однако физиологический механизм цветовосприятия позволяет различать не все излучения. Так, смеси красного и зелёного в определённых соотношениях неотличимы от жёлто-зелёного, жёлтого и оранжевого излучений; смеси синего с оранжевым могут быть уравнены со смесями красного с голубым или с сине-зелёным. У некоторых людей наследственно отсутствует один или два светоприёмника из трёх, в последнем случае цветовое зрение отсутствует, что приводит к дальтонизму.

Цветовое зрение свойственно многим видам животных. У позвоночных (обезьяны, многие виды рыб, земноводные), а из насекомых у пчёл и шмелей цветовое зрение трихроматическое, как и у человека. У сусликов и многих видов насекомых цветовое

зрение дихроматическое, т. е. основано на работе двух типов светоприемников, у птиц и черепах - четырех. Для насекомых видимая область спектра смещена в сторону коротковолновых излучений и включает ультрафиолетовый диапазон. Поэтому мир красок насекомого существенно отличается от человеческого.

Основное биологическое значение цветового зрения для человека и животных, существующих в мире несамосветящихся объектов, – правильное узнавание их окраски, а не просто различение излучений. Спектральный состав отражённого света зависит как от окраски предмета, так и от падающего света и поэтому подвержен значительным изменениям при перемене условий освещения. Способность зрительного аппарата правильно узнавать (идентифицировать) окраску предметов по их отражательным свойствам в меняющихся условиях освещения называются константностью восприятия окраски. Цветовое зрение – важный компонент зрительной ориентации животных. В ходе эволюции многие животные и растения приобрели разнообразные средства сигнализации, рассчитанные на способность животных-«наблюдателей» воспринимать цвета. Таковы ярко окрашенные венчики цветков растений, привлекающие насекомых и птиц-опылителей; яркая окраска плодов и ягод, привлекающая животных – распространителей семян; предупреждающая и отпугивающая окраска ядовитых животных и видов, им подражающих; «плакатная» раскраска многих тропических рыб и ящериц, имеющая сигнальное значение в территориальных взаимоотношениях; яркий брачный наряд, носящий сезонный или постоянный характер, свойственный множеству видов рыб, птиц, пресмыкающихся, насекомых; наконец, специальные средства сигнализации, облегчающие у рыб и птиц взаимоотношения между родителями и потомством.

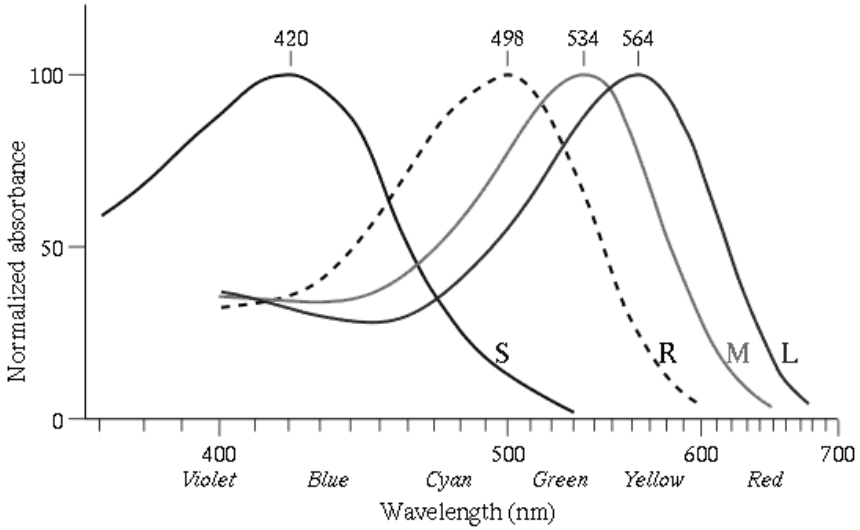
Наличие двух фоторецепторных систем (колбочки и палочки), различающихся по световой чувствительности, обеспечивает подстройку к изменчивому уровню внешнего освещения. В условиях недостаточной освещенности восприятие света обеспечивается палочками, цвета при этом неразличимы (скототопическое зрение). При ярком освещении зрение обеспечивается главным образом колбочками, что позволяет хорошо различать цвета (фототопическое зрение).

Физиологические свойства палочек и колбочек

	Палочки	Колбочки
Светочувствительный пигмент	Родопсин	Йодопсин
Максимум поглощения пигмента	Имеет два максимума – один в видимой части спектра (500 нм), другой – в ультрафиолетовой (350 нм)	Существуют 3 вида йодопсинов, которые имеют различные максимумы поглощения: 440 нм (синий), 520 нм (зеленый) и 580 нм (красный)
Классы клеток	Нет	Каждая колбочка содержит только один пигмент. Соответственно, существуют 3 класса колбочек, чувствительных свету с разной длиной волны
Распределение по сетчатке	В центральной части сетчатки плотность палочек составляет около 150 000 на мм^2, по направлению к периферии она снижается до 50 000 на мм^2. В центральной ямке и слепом пятне палочки отсутствуют.	Плотность колбочек в центральной ямке достигает 150 000 на мм^2, в слепом пятне они отсутствуют, а на всей остальной поверхности сетчатки плотность колбочек не превышает 10 000 на мм^2 .
Чувствительность к свету	У палочек примерно в 500 раз выше, чем у колбочек	
Функция	Обеспечивают черно-белое или серое (скототопическое) зрение	Обеспечивают цветное (фототопическое) зрение

Равномерное раздражение всех трёх типов колбочек, соответствующее средневзвешенному дневному свету, вызывает ощущение белого цвета. Очень сильный свет также возбуждает все три типа колбочек и потому воспринимается как излучение слепяще-белого цвета.

Спектральная чувствительность колбочек и палочек



Нормализованные графики светочувствительности колбочек человеческого глаза S (синяя), M (зеленая), L (красная). Пунктиром показана сумеречная, «чёрно-белая» восприимчивость палочек

Кроме трехкомпонентной теории цветового зрения есть еще теория оппонентных цветов, которая предполагает, что любой цвет можно однозначно описать, указав его положение на двух шкалах - «синий-желтый», «красный-зеленый». Цвета, лежащие на полюсах этих шкал, называют оппонентными. Эта теория подтверждается тем, что в сетчатке, глазном нерве и коре головного мозга существуют нейроны, которые активируются, если их рецептивное поле освещают красным светом и тормозятся, если свет зеленый.

Другие нейроны возбуждаются при действии желтого цвета и тормозятся при действии синего. Предполагается, что сравнивая степень возбуждения нейронов «красно-зеленой» и «желто-синей» системы, зрительная сенсорная система может вычислить цветовые характеристики света. Авторы теории – Мах и Геринг.

Существуют экспериментальные доказательства обеих теорий цветового зрения. В настоящее время считается, что трехкомпонентная теория адекватно описывает механизмы цветовосприятия на уроне фоторецепторов сетчатки, а теория оппонентных цветов – механизмы цветовосприятия на уровне нейронных сетей.

Поскольку размеры палочек гораздо меньше размеров колбочек, то человеческий глаз обладает разным пространственным разрешением для цветного и черно-белого изображений. А именно, угловое разрешение фототопического (цветового) зрения в разы хуже углового разрешения скототопического (черно-белого или серого) зрения.

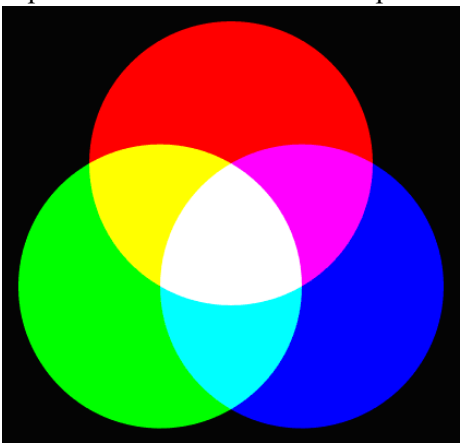
Также, в процессе обработки зрительной информации информация о больших областях с одинаковым цветом объединяется в один средний цвет и границы области. Две соприкасающиеся области с разными цветами будут четко различаться друг от друга по их общей границе. Если же области не имеют общей границы, то цветовое различение резко падает. Аналогично, человек хорошо замечает резкое изменение цвета области. То есть при попеременном отображении на одном и том же месте двух почти одинаковых изображений, человек очень тонко чувствует различия цветов. Таким образом, можно говорить про высокую дифференциальную чувствительность человеческого зрения, как во временном представлении, так и в пространственном.

1.2 Цветовые системы координат.

Аддитивная

RGB (аббревиатура английских слов Red, Green, Blue – красный, зелёный, синий) – аддитивная цветовая модель, описывающая способ синтеза цвета путем добавления (англ. addition) к черному.

Выбор основных цветов обусловлен особенностями физиологии восприятия цвета сетчаткой человеческого глаза. Цветовая модель RGB нашла широкое применение в технике, поскольку в телевизорах и мониторах применяются три электронные пушки (светодиода, светофильтра) для красного, зеленого и синего каналов. А при выключенном телевизоре его экран черный.



Если цвет экрана, освещённого цветным прожектором, обозначается в RGB как (r1, g1, b1), а цвет того же экрана, освещенного другим прожектором, – (r2, g2, b2), то при освещении двумя прожекторами цвет экрана будет обозначаться как (r1+r2, g1+g2, b1+b2).

Изображение в данной цветовой модели состоит из трёх каналов. При смешении основных цветов (основными цветами считаются красный, зелёный и синий) – например, синего (B) и красного (R), мы получаем пурпурный (M magenta), при смешении зеленого (G) и красного (R) – жёлтый (Y yellow), при смешении зеленого (G) и синего (B) – голубой (C cyan). При смешении всех трёх цветовых компонентов мы получаем белый цвет (W white). При отсутствии любого из цветов получаем черный цвет (black).

Субтрактивная

Четырёхцветная автотипия (CMYK: Cyan, Magenta, Yellow, blacK) – субтрактивная схема формирования цвета, используемая прежде всего в полиграфии для стандартной триадной печати. По-русски эти цвета часто называют так: голубой, пурпурный, жёлтый;

но профессионалы подразумевают cyan, magenta и yellow. Печать четырьмя красками, соответствующими CMYK, также называют печатью триадными красками. Каждая краска (C,M,Y) поглощает соответствующую часть спектра. В результате из исходного белого цвета бумаги остается только часть цвета.

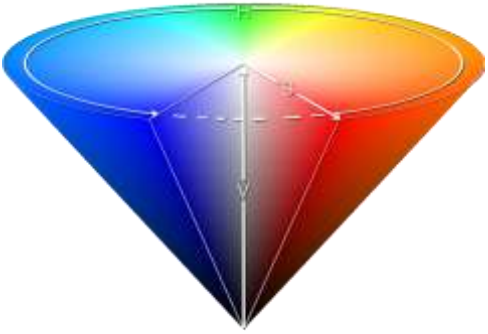
При использовании всех трех красок в равной пропорции цвет должен получиться черным (или серым). Однако точности изготовления красок недостаточно и цвет получается грязно-коричнеым или грязно-зеленым. Черная краска, изготавливаемая, например, из сажи, очень дешевая и дает точный черный цвет. Поэтому вводится дополнительный коэффициент K, указывающий количество черной краски. Соответственно, из всех трех компонент (C,M,Y) вычитается количество черной краски K. В результате в четырехкомпонентной системе цветовых координат CMYK одна из компонент всегда нулевая и остается только три ненулевых независимых компоненты.

На практике один и тот же цвет CMYK будет по разному выглядеть на разных устройствах печати, с разными партиями красителей, на разной бумаге и т.п. В то время как люминофор, использующийся в цветных телевизорах, обладает высокой стабильностью спектральных характеристик. Поэтому нередко создают профиль CMYK, описывающий печатный процесс конкретной печатной машины с конкретной бумагой. Соответственно, дизайнеры используют этот профиль для преобразования исходного изображения в «правильный» CMYK. Профиль CMYK, как правило, обладает сравнительно небольшим цветовым охватом. Цветовая модель RGB имеет по многим тонам цвета более широкий цветовой охват (может представить более насыщенные цвета), чем типичный охват цветов CMYK, поэтому иногда изображения, замечательно выглядящие в RGB, значительно тускнеют и гаснут в CMYK.

«Палитра художника»

Следующие цветовые модели очень удобны для выбора цвета человеком. В системах HSV (англ. Hue, Saturation, Value – цветовой тон, насыщенность, значение) и HSB (англ. Hue, Saturation, Brightness – цветовой тон, насыщенность, яркость) координатами цвета являются:

- Шкала оттенков – Hue – цветовой тон, (например, красный, зелёный или сине-голубой). Варьируется в пределах 0..360°, однако иногда приводится к диапазону 0..100% или 0..1.
- Saturation – насыщенность. Варьируется в пределах 0..100% или 0..1. Чем больше этот параметр, тем «чище» цвет, поэтому этот параметр иногда называют чистотой цвета. А чем ближе этот параметр к нулю, тем ближе цвет к нейтральному серому.
- Value (значение цвета) или Brightness – яркость. Также задаётся в пределах 0..100% и 0..1.



Модель HSV была создана в 1978 году Элви Реем Смитом, одним из основателей компании Pixar. Она является нелинейным преобразованием модели RGB.

Для определения преобразования HSV удобно ввести максимальное и минимальное значение компонент R, G и B – $C_{max} = \max(R, G, B)$ и $C_{min} = \min(R, G, B)$. Тогда:

$$Hue = \begin{cases} 0, & \text{если } C_{max} = C_{min} \\ 60 \cdot \dfrac{G - B}{C_{max} - C_{min}} + 0, & \text{если } C_{max} = R \text{ и } G \geq B \\ 60 \cdot \dfrac{G - B}{C_{max} - C_{min}} + 360, & \text{если } C_{max} = R \text{ и } G < B \\ 60 \cdot \dfrac{B - R}{C_{max} - C_{min}} + 120, & \text{если } C_{max} = G \\ 60 \cdot \dfrac{R - G}{C_{max} - C_{min}} + 240, & \text{если } C_{max} = B \end{cases}$$

$$Saturation = \begin{cases} 0, & \text{если } C_{max} = 0 \\ 1 - \dfrac{C_{min}}{C_{max}}, & \text{если } C_{max} \neq 0 \end{cases}$$

$$Value = C_{max}$$

Телевизионная

В телевизиооной цветовой модели YUV используются три компоненты – яркостная Y и цветоразностные U(Cb) и V(Cr):

Y = 0.299*Red + 0.587*Green + 0.114*Blue

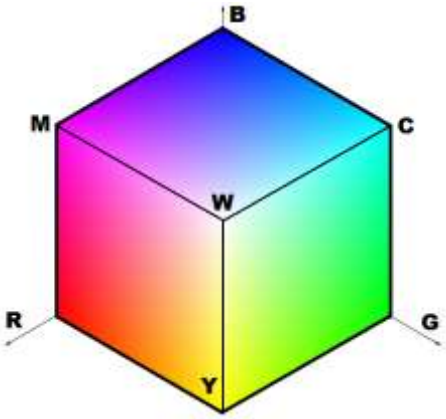
U = Cb = Blue - Y = - 0.299*Red - 0.587*Green + (1- 0.114)*Blue

V = Cr = Red - Y = (1- 0.299)*Red - 0.587*Green - 0.114*Blue

Яркостная компонента Y соответствует скототопическому (черно-белому или серому) зрению. Зеленая компонента вносит наибольший вклад, а синяя – наименьший, что соответствует относительной чувствительности колбочек разных типов. Чистый синий цвет выглядит темнее красного, а чистый красный – темнее зеленого. Если сравнить телевизионную систему координат YUV и аддитивную систему координат RGB, то легко заметить, что ось Y совпадает с главной диагональю цветового куба RGB, поскольку для всех «серых» цветов значения R, G и B совпадают, а значит и компонента Y в точности равна значениям R, G и B. Для всех серых цветов значения U и V равны нулю.

Соответствие систем друг другу

Рассмотрим полный цветовой куб в пространстве RGB. Слева снизу видна вершина R (Red), снизу в центре – Y (Yellow), справа снизу – G (Green), справа сверху – C (Cyan), сверху в центре – B (Blue), слева сверху – M (Magenta). По центру видна вершина W (White), а за ней сзади расположена вершина K (blacK), но она не видна на рисунке.



Основные оси системы RGB соответствуют осям $\overrightarrow{KR}$, $\overrightarrow{KG}$ и $\overrightarrow{KB}$, начало координат системы RGB расположено в точке K (blacK). Основные оси системы CMY соответствуют осям $\overrightarrow{WC}$, $\overrightarrow{WM}$ и $\overrightarrow{WY}$, начало координат системы CMY расположено в точке W (white). Видно, что оси системы CMY параллельны осям системы RGB, но начинаются с другого края главной диагонали цветового куба $\overrightarrow{KW}$.

Если смотреть на цветовой куб вдоль главной диагонали $\overrightarrow{WK}$, то виден правильный шестиугольник, в вершинах которого располагаются чистые цвета R, Y, G, C, B, M, что в точности соответствует цветам радуги. В центре шестиугольника (на главной диагонали $\overrightarrow{KW}$) располагаются серые цвета с нулевой насыщенностью, а по краям шестиугольника – цвета с максимальной насыщенностью. Если шестиугольник плавно растянуть в круг, касающийся его вершин, то получится цветовой круг из системы «палитра художника».

Ось яркости Y телевизионной системы YUV соответствует главной диагонали куба $\overrightarrow{KW}$, а цветоразностные оси лежат в плоскости, перпендикулярной диагонали куба $\overrightarrow{KW}$.

1.3 Растровая графика. Пространственное разрешение, размеры изображения и пикселя.

Первыми появились устройства для отображения векторной графики. Они использовали набор геометрических примитивов, таких как точки, линии, сплайны и многоугольники, для представления изображений. Изначально, для визуализации использовался телевизионный монитор, основой которого является электронно-лучевая трубка. В устройствах векторной графики все примитивы отображались последовательно один за другим. Соответственно, время построения очередного изображения сильно зависело от его сложности.

С развитием элементов памяти появились первые растровые устройства, у которых изображение строилось из отдельных точек – пикселей (pixel = picture element). Изображение разбивалось на строки и на столбцы, образовывая таблицу, в каждой ячейке которой указывается цвет конкретного пикселя. В результате получался прямоугольный растр. В растровых устройствах сфокусированный луч последовательно разворачивал строки растра

слева направо и сверху вниз. По окончании развертки одного изображения луч перемещался вверх и влево и снова разворачивал следующее изображение. Время развертки каждого изображения было строго определено и не зависело от сложности изображения.

Растровое изображение – это файл данных или структура, представляющая собой сетку пикселей или точек цветов (на практике прямоугольную), предназначенную для последующего отображения на бумаге и других отображающих устройствах и материалах. Размерность сетки – количество строк (высота) и столбцов (ширина). Изображение в пикселе задается в каком-либо цветовом представлении: RGB, CMYK, YCbCr или другом. Также задается глубина цвета или разрядность цветовых компонент, обычно определяемая числом бит на пиксель (bit per pixel – BPP).

Следует четко различать физические и логические размеры изображения. Логические размеры – ширина и высота изображения в пикселях. Физические размеры – ширина и высота изображения в стандартных единицах длины (метры, сантиметры и т.д.). Соотношение между физическими и логическими размерами зависят от конкретного устройства отображения растровых изображений и измеряется в точках на дюйм (dot per inch – DPI). Так, характерные значения DPI для мониторов VGA – 96 DPI, а для современных принтеров – до 1200 DPI. Поэтому один и тот же файл (одно и то же растровое изображение) на экране монитора будет выглядеть очень крупным, а на принтере – очень мелким. А если это изображение на принтере «растянуть» до таких же размеров, как и на мониторе, то изображение будет выглядеть нечетким (размытым), либо будет иметь четко видимую «квадратную» структуру.

Кроме того, различные устройства могут иметь разное разрешение (DPI) по горизонтали и вертикали. Например, старые струйные и матричные принтеры имели разрешение по горизонтали 150-300 DPI, а по вертикали – всего 72 DPI. Для таких устройств актуальным становится понятие Aspect Ratio – соотношение сторон пикселя – отношение физической ширины одного пикселя к его физической высоте. Не единичный Aspect Ratio означает, что круг, нарисованный в графическом редакторе, на устройстве будет показываться эллипсом.

1.4 Палитра, цветовое разрешение.

Память в первых растровых устройствах была очень медленной и дорогой, поэтому ее хватало только для отображения однобитовой информации. Изображение было черно-белым. Единичное значение бита включало сканирующий луч и соответствующая точка на экране светилась. Нулевое значение выключало луч и соответствующая точка оставалась темной. Затем появились устройства с четырьмя банками памяти, что позволило задавать по 4 бита на пиксель. С тех пор стандартной считается следующая палитра из 16 цветов: шесть основных цветов (R, G, B, C, M, Y) полной интенсивности, шесть основных цветов половинной интенсивности, черный, белый, темно серый и светло серый.

Затем появились устройства с одним байтом на пиксель – 8 BPP. Это соответствует 256 цветам в палитре. В каждом пикселе изображения хранится индекс в палитре, а сама палитра (256 цветов, заданных 8-битными компонентами R, G и B) хранится рядом с изображением. Такой стандарт используется до сих пор, поскольку в очень многих областях применений не требуется большого многообразия цветовых оттенков.

Эксперименты, проводимые Тепловым и Соколовым, показали, что число замечаемых глазом различий по насыщенности колеблется от 7 до 12 градаций для разных цветовых тонов. Эксперименты, проводимые Райтом и Питтом в 1946 году, показали, что общее число различимых глазом цветовых тонов примерно равно 180. Наиболее чувствителен глаз к различию яркостей – эксперименты Кенига показали, что глаз различает около 600 градаций яркости. Однако при уменьшении насыщенности или при увеличении или уменьшении яркости ухудшается способность различать цветовые тона. При минимальной насыщенности хроматические цвета сводятся к двум различным тонам: желтоватому («теплому») и синеватому («холодному»). В результате в среднем человек способен различать лишь несколько тысяч разных цветовых оттенков.

Для передачи реального изображения палитры в 256 цветов явно недостаточно, однако разница в требуемом количестве не столь велика – всего в несколько раз. Если есть возможность пожертвовать эффективным пространственным разрешением

изображения, то можно пропорционально повысить цветовое разрешение. Алгоритмы дизеринга позволяют расположить рядом цвета из компактной палитры таким образом, чтобы «издалека» получившееся изображение было максимально похоже на исходное изображение.

Таким образом, хотя во всех современных мониторах используется «полноцвет» (24 бита для представления цвета, то есть по 8 бит на каждую из компонент R, G и B), на практике до сих пор активно используются изображения с ограниченной цветовой палитрой, как правило, не превышающей 256 цветов (когда индекс цвета не превышает одного байта).

1.5 Методы уменьшения цветового разрешения (подбор палитры).

Подбор палитры или уменьшение цветового разрешения (числа используемых цветов) означает выбор некоторого подмножества цветов, наиболее точно описывающих исходное полноцветное изображение.

Простейшим способом получения цветов в палитре является «цветовой срез», когда для отображения цвета используются старшие значащие биты каждой из цветовых компонент. Так, при использовании 256-цветной палитры 8 бит индекса в палитре формируются следующим образом: 3 старших бита компоненты R, затем 3 старших бита компоненты G и затем 2 старших бита компоненты B. Человеческий глаз обладает наименьшей спектральной чувствительностью к компоненте B, поэтому ей отводится меньшее число бит. В итоге для любого цвета пикселя быстро и однозначно формируется индекс в палитре. Этот способ хорош для аппаратной реализации, не зависит от изображения, но очень плохо передает информацию о цветах в исходном изображении и практически не применяется.

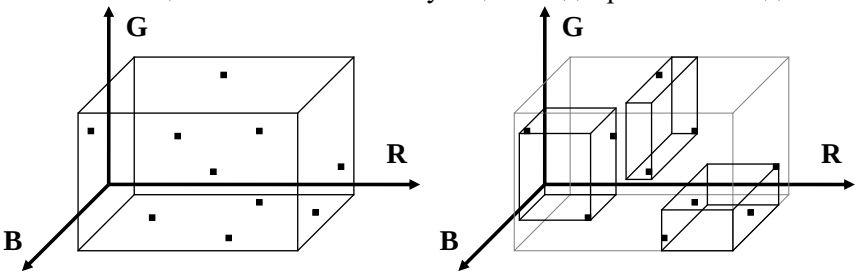
Простейшим способом эффективного подбора палитры является прямой способ построения статистики цветов в исходном изображении. Пусть по горизонтальной оси отложены все цвета, а по вертикальной оси – количество пикселей с данным цветом, присутствующие в изображении. N – полное число пикселей в изображении. K – число цветов в палитре. Тогда N/K – число пикселей, приходящихся в среднем на один цвет в палитре. Найдя

максимум статистики, можно выделить полосу цветов, суммарное количество пикселей в которой больше или равно N/K. Средневзвешенный цвет в полосе заносим в первый цвет в палитре, убираем все цвета полосы из статистики и продолжаем алгоритм. В результате мы разделим все цвета на не более чем K полос, что будет соответствовать K цветам в палитре.

Этот метод очень прост теоретически, но очень плох на практике. Во-первых, при очень высоком цветовом разрешении распределение цветов в статистике может быть слабо выраженным, без заметных максимумов. Можно специально сократить цветовое разрешение, чтобы укрупнить распределение и уменьшить полное число цветов в статистике, тем самым ускорив быстродействие алгоритма и уменьшив требования к памяти.

Метод деления цветового параллелепипеда.

В пространстве RGB строится минимальный параллелепипед, охватывающий все цвета исходной палитры (рисунок слева). Этот параллелепипед итеративно делится на подпараллелепипеды (рисунок справа), по которым строится конечная палитра: число цветов в палитре равно числу подпараллелепипедов; каждый цвет в палитре является среднеарифметическим цветом $\overline{C}$ соответствующего подпараллелепипеда.

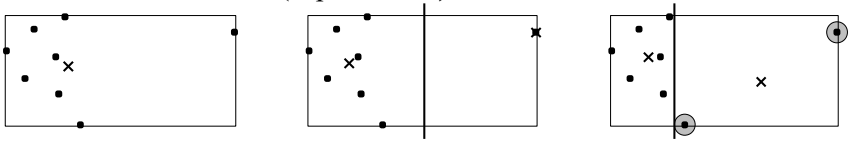


N-ый шаг итерации. Для каждого подпараллелепипеда вычисляется общий вес $P = \sum_i w_i \bullet P_i$, зависящий от нескольких параметров:

1) максимальный размер подпараллелепипеда $L_{\max} = \max(L_R, L_G, L_B)$, где L_R, L_G и L_B являются размерами подпараллелепипеда вдоль осей R, G и B соответственно;
2) число цветов в подпараллелепипеде n;
3) отношение $\frac{L_{\max}}{L_{\min}}$, где $L_{\min} = \min(L_R, L_G, L_B)$.

Первые два параметра вносят прямой вклад в результирующую ошибку, вычисляемую, как суммарная разница ошибки в цвете между исходным и результирующим изображениями, поэтому их веса равноправны и служат только для нормализации: w_1 нормируется к размерности цветовой компоненты, а w_2 – к полному числу пикселей изображения. Третий параметр должен влиять на выбор только в том случае, если нужно выбрать один из двух примерно равноправных подпараллелепипедов, поэтому его вес должен быть в разы меньше. Третий параметр будет тем больше, чем сильнее вытянут подпараллелепипед вдоль какой-либо оси. В среднем, после разделения таких подпараллелепипедов на две части суммарный объем двух новых подпараллелепипедов оказывается существенно меньше объема исходного подпараллелепипеда. Для нормализации необходимо использовать логарифмическую функцию.

Подпараллелепипед с максимальным общим весом делится на две части секущей плоскостью. В качестве секущей плоскости выбирается плоскость, нормаль которой совпадает с той осью, вдоль которой размер подпараллелепипеда максимален. Плоскость может проходить через центр подпараллелепипеда или через $\overline{C}$. При равномерном распределении цветов оба варианты практически эквивалентны. При значительной неравномерности распределения цветов (вариант «а») деление подпараллелепипеда по $\overline{C}$ может давать явные ошибки (вариант «в»).



(а) (б) (в)

Кроме того, деление подпараллелепипеда пополам (вариант «б») проще и быстрее в реализации – не нужно вычислять $\overline{C}$ на каждое деление подпараллелепипеда.

В процессе деления образуется два набора цветов, по которым вместо разделенного подпараллелепипеда строятся два новых подпараллелепипеда, причем те подпараллелепипеды, размеры которых не превышают максимально допустимой величины ошибки приближения цвета $Err_{\max}$, в дальнейшем не рассматриваются.

0-ой шаг итерации. Поскольку вначале имеется только один набор цветов (исходная палитра), то для деления выбирается исходный параллелепипед. В остальном 0-ой шаг итерации подобен остальным шагам.

Итеративный цикл деления цветовых подпараллелепипедов завершается при достижении максимально допустимого числа цветов конечной палитры (ограничение количества подпараллелепипедов) или в том случае, когда $L_{\max}$ всех подпараллелепипедов не будут превышать $Err_{\max}$.

1.6 Дизеринг: метод упорядоченного возбуждения, распространение ошибки.

После построения палитры (или при использовании заранее заданной палитры) необходимо заменить каждый пиксель исходного полноцветного изображения некоторым индексом в палитре. Простейший способ – выбор наиболее близкого цвета в палитре. Однако он обладает рядом недостатков – плохо передает детали изображения, формирует большие одноцветные области, границы между которыми очень сильно бросаются в глаза из-за высокой дифференциальной чувствительности человеческого зрения. Если границы между одноцветными областями «размыть», сделать не такими заметными, то визуально качество результирующего изображения существенно возрастет. Аналогичная проблема встает при печати на устройствах, у которых может быть только два цвета – черный (цвет краски) или белый (цвет бумаги). Используя палитру всего из двух цветов необходимо максимально достоверно отобразить произвольное изображение с

256 градациями серых цветов. Точки в принтере очень маленькие, гораздо меньше углового разрешения человеческого зрения при рассматривании изображения в нормальных условиях. Объединяя несколько черных и белых точек в одну «общую» точку можно получить много градаций серого цвета. Собственно метод увеличения цветового разрешения за счет уменьшения пространственного разрешения и называется дизерингом.

Один из самых простых и достаточно эффективных методов дизеринга – метод упорядоченного возбуждения. В изображение вводится случайная ошибка, которая добавляется к интенсивности каждого пикселя до ее сравнения с выбранной пороговой величиной. Добавление совершенно произвольной ошибки не приводит к оптимальному результату. Тем не менее, существует оптимальная аддитивная матрица ошибки, минимизирующая эффекты появления фактуры на изображении. Матрица ошибки добавляется к изображению таким же способом, как расположены клетки на шахматной доске. Минимальная матрица упорядоченного возбуждения имеет размер 2x2. Оптимальная 2x2–матрица, которую первым предложил Лим, имеет вид:

$$|D_2| = \begin{vmatrix} 0 & 2 \\ 3 & 1 \end{vmatrix}$$

Матрицы 4x4, 8x8 и больших размеров получают с помощью рекуррентных соотношений (n => 4)

$$|D_{2n}| = \begin{vmatrix} 4D_n & 4D_n + 2U_n \\ 4D_n + 3U_n & 4D_n + U_n \end{vmatrix}$$

где n – размер матрицы и Un – единичная матрица nxn
Например, матрица возбуждения размера 4x4 имеет вид:

$$|D_4| = \begin{vmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{vmatrix}$$

Обратите внимание, что независимо от n изображение не теряет пространственного разрешения. Черный цвет по прежнему останется черным, а белый – белым, поэтому контрастный цветовой переход сохранится в результирующем изображении.

Алгоритм упорядоченного возбуждения

```
Xmin, Xmax, Ymin, Ymax — пределы растра
Mod — функция, возвращающая остаток от целого деления
первого аргумента на второй
  for y = Ymax to Ymin step -1
    // для каждого пиксела на строке (слева направо)
    for x = Xmin to Xmax
      // определяем позицию в матрице возбуждения
      i = (x Mod n) + 1
      j = (y Mod n) + 1
      // определяем выводимое значение пиксела
      if I(x, y) < D(i, j) then
        Пиксел(x, y) = Черный
      else
        Пиксел(x, y) = Белый
      end if
      // изображаем пиксел
      Display Пиксел(x, y)
    next x
  next y
finish
```

Метод упорядоченного возбуждения удобен для аппаратной реализации, поскольку легко распараллеливается. К недостаткам можно отнести «заметность» матрицы возбуждения. Например, при обработке последовательности изображений (кадров мультфильма) матрица остается на месте, а изображение меняется. В результате у зрителя возникает ощущение, что он смотрит как бы через рифленое стекло.

Другим очень популярным методом дизеринга является метод «распространения ошибки». В методе, разработанном Флойдом и Стейнбергом, эта ошибка распределяется на окружающие пиксели. Распределение ошибки происходит всегда вниз и вправо. Следовательно, при генерации изображения в порядке сканирования возвращаться обратно не нужно. В алгоритме Флойда-Стейнберга 3/8 ошибки распределяется вправо, 3/8 – вниз и 1/4 – по диагонали вправо-вниз.

Алгоритм распространения ошибки

Для порога, равного среднему между минимальной и максимальной интенсивностями, T = (Белый + Черный)/2, алгоритм формулируется следующим образом.

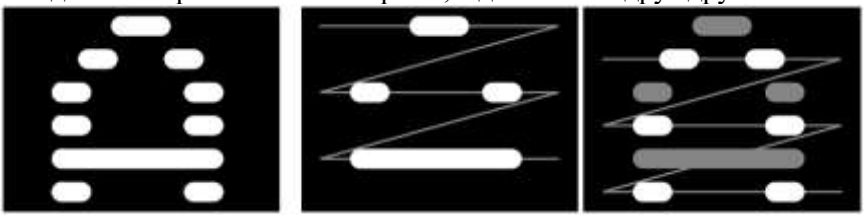
```
Xmin, Xmax, Ymin, Ymax — пределы растра
 T = (Черный + Белый)/2
  for y = Ymax to Ymin step -1
    // для каждого пиксела на строке (слева направо)
    for x = Xmin to Xmax
      // определяем выводимое значение пиксела для
      // пороговой величины T и вычисляем ошибку
      if I(x, y) < T then
        Пиксел(x, y) = Черный
        Ошибка = I(x, y) - Черный
      else
        Пиксел(x, y) = Белый
        Ошибка = I(x, y) - Белый
      end if
      // изображаем пиксел
      Display Пиксел(x, y)
      // распределяем ошибку на соседние пикселы
      I(x + 1, y) = I(x + 1, y) + 3 * 0шибка/8
      I(x, y - 1) = I(x, y - 1) + 3 * Ошибка/8
      I(x + 1, y - 1) = I(x + 1, y - 1) + Ошибка/4
    next x
  next y
finish
```

Распределение ошибки на соседние пиксели улучшает вид деталей изображения, так как информация, заключенная в изображении, не теряется. Однако этот алгоритм невозможно распараллелить. Также, интересной особенностью метода является визуализация какого-либо небольшого предмета на строго одноцветном фоне. Вначале, при обработке верхней части изображения, образуется одна структура черных и белых точек, характер которой зависит от яркости фона. Эта же структура сохранится и слева от изображения. Но справа и снизу от изображения на одноцветном фоне сформируется другая структура черных и белых точек, поскольку на точки фона будет распространяться ошибка, накопившаяся при обработке точек предмета (не фона). Граница между разными структурами достаточно четко определяется зрителем.

1.7 Телевизионная графика. Чересстрочная развертка. Уплотнение спектра (гребенчатый фильтр).

Телевидение начиналось с передачи черно-белого изображения, когда яркостный сигнал Y построчно сканировался и передавался в виде частотно модулированного сигнала в эфир. Телевизионный приемник принимал сигнал, демодулировал из него сигнал яркости и разворачивал его построчно на электронно-лучевой трубке. Модуль развертки в телевизионном приемнике смещал луч по вертикали и по горизонтали с кадровой и строчной частотами соответственно.

При разработке телевизионных стандартов использовалось большое число различных остроумных инженерных решений. Одним из таких решений является чересстрочная развертка, когда в последовательности изображений, передаваемых в эфир, четные изображения смещены вверх на пол строки, а нечетные – вниз на пол строки. В результате при передаче статического изображения телевизионный приемник отображает в два раза больше строк, чем их передается в одном изображении, поскольку люминофор монитора обладает высокой остаточной светимостью и строки соседних изображений не затирают, а дополняют друг друга.



На рисунке слева показано изображение, включающее все строки (исходное изображение), в центре - изображение, передающие его нечетные строки (1, 3, 5), а справа – четные строки (2, 4, 6). Видно, что хотя нечетные строки и имеют меньшую светимость, но успешно воспринимается зрительным аппаратом человека. Два соседних изображения называют двумя полями одного кадра. В телевидении их называют четным и нечетным полями, но в разных телевизионных стандартах первое поле кадра бывает разным – четным или нечетным, поэтому удобнее использовать понятие Upper Field First или Lower Field First («верхнее поле первым» или «нижнее поле первым»). Верхнее поле

– это поле, включающее самую верхнюю видимую строку кадра. Соответственно нижнее поле – это поле, включающее самую нижнюю видимую строку кадра.

С другой стороны, при передаче динамических изображений, когда каждая следующая картинка отличается от предыдущей, половинного разрешения кадра достаточно для грубого (первичного) распознавания зрителем быстроменяющейся картинки. Для того, чтобы зритель мог рассмотреть детали, он должен вглядываться в изображение, а для этого изображение должно быть относительно неподвижным.

В компьютерной технике, как правило, используется прогрессивная развертка, когда все строки кадра воспроизводятся подряд (и четные, и нечетные строки воспроизводятся вместе). При одинаковой частоте передачи изображений и одинаковом максимальном разрешении, при чересстрочной развертке нужно передавать в единицу времени в два раза меньше данных, чем при прогрессивной развертке, поэтому даже в современном телевидении сохранилась чересстрочная развертка.

Другим очень красивым инженерным решением является метод «уплотнения спектра» для передачи цветоразностного сигнала в яркостном сигнале.

Для объяснения этого метода надо рассмотреть особенности спектра телевизионного сигнала. При развертке сигнала по горизонтали сканирующий луч часть времени «бежит» вдоль строки, а затем быстро возвращается назад к началу следующей строки. Время обратного хода луча порядка 10% от времени прямого хода луча. Аналогично, при развертке по вертикали в телевизионном сигнале есть область «обратного хода кадра», когда сканирующий луч перемещается снизу вверх к первой строке следующего поля. Полная структура «обратных ходов луча» повторяется через два поля, то есть ровно через кадр.

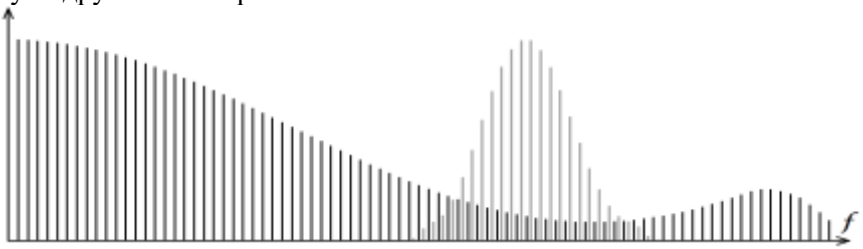
Если изображение не изменяется во времени (статический кадр), то механизм развертки формирует периодический сигнал, полностью повторяющийся с кадровой частотой.

Из Фурье анализа известно, что спектр периодического сигнала состоит только из гармоник основной частоты $F_{\text{кадра}}$.

$$Y(f) = \sum_n Y_n \cdot \delta(f - n \cdot F_{\text{кадра}})$$

Если изображение изменяется во времени, то дельта-функции гармоник «расплываются» в гаусовское распределение. И при очень быстром изменении гаусовские функции перекрываются. Полное перекрытие соответствует «белому шуму», когда каждое следующее изображение абсолютно не похоже на предыдущее. На практике каждый следующий кадр очень похож на предыдущий, поэтому спектр яркостного сигнала имеет четко выраженную гребенчатую структуру, где каждый зубчик «гребенки» соответствует одной из гармоник кадровой частоты.

Свойство гребенчатости спектра активно использовали разработчики цветного телевидения. Сигнал, несущий информацию о цвете обладает таким же гребенчатым спектром, как и сигнал яркости, поскольку он сформирован тем же механизмом развертки. Если сместить спектры яркостного и цветоразостных сигналов на величину N*Fкадра + Fкадра/2, то пики одного спектра попадут в нули другого спектра.



При декодировании такого сигнала можно использовать гребенчатые фильтры для разделения спектров сигналов, а значит и самих сигналов. Разделить сигналы не удастся только тогда, когда спектры исходных сигналов «расплываются» слишком сильно, то есть когда изображение будет изменяться очень быстро. Но при такой резкой смене зритель все равно не успеет что-либо рассмотреть, а значит, внесенные искажения не будут заметны. Для надежного распознавания зрительным аппаратом человеку требуется время, существенно большее длительности нескольких кадров. Поэтому при передаче нормальных изображений никаких искажений при разделении спектра не наблюдается.

1.8 Стандарты цветного телевидения: NTSC, PAL, SECAM. Искажения, возникающие при передаче телевизионных изображений.

Основные искажения, возникающие при отображении синтезированных изображений на цветных телевизорах, связаны с проблемами построения гребенчатых фильтров для разделения спектров яркостного и цветоразностных сигналов. В бытовых телевизионных приемниках вместо сложных и дорогих гребенчатых фильтров используют простые полосовые фильтры, что приводит к неполному разделению спектров и к так называемым перекрестным помехам, когда высокочастотные составляющие сигнала яркости попадают в спектр сигнала цветности и наоборот, высокочастотные составляющие сигнала цветности попадают в спектр сигнала яркости. Поэтому на резких контрастных цветовых переходах возникают помехи – по контрастной границе движутся маленькие точки, похожие на муравьев.

Во всех трех стандартах NTSC, SECAM и PAL цветовая информация передается в виде одного сигнала на поднесущей в спектре яркостного сигнала. Отличие состоит в способе модуляции поднесущей и выборе способа получения композитного цветоразностного сигнала из двух основных цветоразностного сигналов U/Cb и V/Cr.

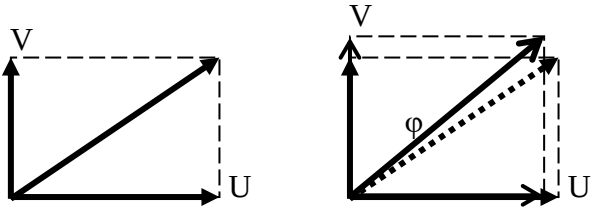
NTSC

Параметры развертки: 486 видимых строк, 29,97 кадров в секунду (59,94 полей в секунду).

Для получения цветоразностного сигнала используется квадратурная модуляция цветовой поднесущей с частотой 3.58 МГц, с использованием двух балансных модуляторов (один для сигнала U, второй для сигнала V), питаемых одной и той же поднесущей с фазами, смещенными на 90°.

Для сигнала NTSC характерной проблемой является синфазная помеха, приводящая к изменению цветового тона. Есть даже расшифровка аббревиатуры NTSC как «Never Twice the Same Color». При декодировании сдвиг фазы приводит к синхронному изменению цветоразностных компонент, что приводит к изменению цветового тона (см. рисунок). Даже небольшие изменения в цвете

кожи (например, на лице диктора) оказываются очень заметными для зрителя.



PAL

Параметры развертки: 576 видимых строк, 25 кадров в секунду (50 полей в секунду).

Как и в системе NTSC, сигналы U и V используются для модуляции цветовых поднесущих, используя два балансных модулятора, работающих со сдвигом по фазе на 90°. Однако для подавления фазовых помех используется смена фазы сигнала V на противоположную от строки к строке. При декодировании складываются сигналы текущей и предыдущей строк. Фазовая помеха меняется от строки очень слабо, поэтому она практически полностью подавляется.

SECAM

Параметры развертки: 576 видимых строк, 25 кадров в секунду (50 полей в секунду).

Система SECAM (Séquentiel couleur à mémoire — последовательная цветная с памятью) исторически является первым стандартом цветного телевидения в Европе. Отличительной чертой стандарта SECAM является поочередная передача двух цветоразностных сигналов через строку на частотно-модулированной поднесущей при непрерывной передаче сигнала яркости. При декодировании для текущей строки используется один цветоразностный сигнал, переданный в текущей строке, и другой цветоразностный сигнал, переданный в предыдущей строке.

У сигнала SECAM характерной проблемой является формирование несуществующих цветов вдоль широкой горизонтальной границы двух областей разных цветов. Например, при полноэкранном отображении российского флага на границе

синего и красного цветов возникают фиолетовый и черный цвета. Эти цвета меняются от поля к полю с периодичностью 4 поля, то есть повторяются с частотой 12,5 Герц, что уже заметно зрителю.

1.9 Простые алгоритмы сжатия изображений: RLE, LZW.

Алгоритм RLE

Алгоритм группового кодирования (RLE – Run Length Encoding) является самым простым алгоритмом сжатия. Он имеет очень много модификаций и часто используется как часть более сложных алгоритмов. Он является самым быстрым, простым и понятным алгоритмом компрессии и при этом иногда оказывается весьма эффективным. Алгоритм оперирует группами данных – последовательностями, в которых один и тот же символ встречается несколько раз подряд. Суть его заключается в том, что при кодировании строка повторяющихся символов, составляющих группу, заменяется парой, содержащей сам повторяющийся символ (значение группы) и количество его повторов (счетчик группы). Как правило, счетчик передается первым.

Легко заметить, что высокий коэффициент сжатия достигается при кодировании данных, содержащих минимальное количество цветов, без плавных переходов (с резкими границами) и с большими участками, заполненными одним цветом. Так как сканирование данных осуществляется построчно слева направо, высокий коэффициент сжатия будет достигаться на изображениях, строки которых содержат достаточно длинные цепочки одинаковых пикселей.

Одна из наиболее известных модификаций алгоритма RLE – RLE с флагом – обеспечивает относительно высокий коэффициент сжатия при кодировании последовательностей с неповторяющимися данными. Старший (знаковый) бит счетчика отводится под флаг. Если флаг не установлен (значение счетчика положительное), то это означает, что далее во входном потоке следует один символ, который нужно повторить в выходной поток указанное число раз. Если же флаг установлен (значение счетчика отрицательное), то далее во входном потоке следует указанное

число разных символов, которые нужно скопировать в выходной поток.

Алгоритм группового кодирования применяется в форматах PCX, TIFF, BMP.

Алгоритм LZW

Алгоритм LZW – это универсальный алгоритм сжатия данных без потерь, созданный Абрахамом Лемпелем (Abraham Lempel), Якобом Зивом (Jacob Ziv) и Терри Велчем (Terry Welch). Он был опубликован Велчем в 1984 году, в качестве улучшенной реализации алгоритма LZ78, опубликованного Лемпелем и Зивом в 1978 году. Алгоритм разработан так, чтобы его можно было быстро реализовать, но он не обязательно оптимален, поскольку он не проводит никакого анализа входных данных.

Данный алгоритм при сжатии (кодировании) динамически создаёт таблицу преобразования строк: каждая ячейка таблицы содержит ссылку на известную (уже переданную) строку. Индекс ячейки является кодом этой строки. Причем в ячейке хранится не строка целиком (в виде массива байт с нулем в конце), а последний символ строки и индекс строки без последнего символа. Таким образом, каждая ячейка таблицы содержит один символ и один индекс. Таблица инициализируется всеми односимвольными строками (в случае 8-битных символов – это 256 записей). По мере кодирования, алгоритм просматривает входной текст символ за символом, и сохраняет каждую новую уникальную строку в таблицу в виде пары код/символ, где код ссылается на последнюю найденную в таблице строку, а символ хранит последний символ новой строки. Одновременно с сохранением новой строки в таблице, на выход передаётся код последней найденной в таблице строки. Соответственно, когда декодер читает из входного потока очередной код, то он находит по этому коду (индексу) строку символов (разворачивая ее из конца в начало), выдает ее в выходной поток и добавляет в таблицу (в очередную свободную ячейку) новую пару из полученного кода и первого символа следующей строки, которая будет декодироваться из следующего кода во входном потоке.

Размерность таблицы строго фиксирована (не может меняться в процессе кодирования). Алгоритму декодирования на

входе требуется только закодированный текст, поскольку он воссоздает таблицу преобразования строк непосредственно по закодированному тексту. Кодер выдает очередной код в выходной поток одновременно с внесением в таблицу очередной записи. Декодер также создает новую запись в таблице одновременно с чтением очередного кода и входного потока. При кодировании длинной последовательности данных наступает момент, когда вносится запись в последнюю ячейку таблицы. У кодера и у декодера этот момент наступает одновременно. Для продолжения работы кодер (и декодер) просто начинают работу заново, инициализируя таблицу всеми односимвольными строками.

Алгоритм

1. Инициализация словаря всеми возможными односимвольными фразами.
2. Инициализация входной фразы *W* первым символом сообщения.
3. Считать очередной символ *C* из кодируемого сообщения.
4. Если данных больше нет, то выдать код для *W*, иначе
5. Если фраза *WC* уже есть в словаре, то присвоить входной фразе значение *WC* и перейти к шагу 3, иначе выдать код *W*, добавить *WC* в словарь, присвоить входной фразе значение *C* и перейти к шагу 3.

1.10 Статистические алгоритмы упаковки данных: алгоритм Хаффмана и арифметическое кодирование.

Алгоритм Хаффмана

Название алгоритм получил в честь разработчика – Дэвида Хаффмана. Метод Хаффмана строит таблицы кодов, базирующихся на частоте повторения символов. Чем чаще встречается тот или иной символ, тем короче будет заменяющий его код.

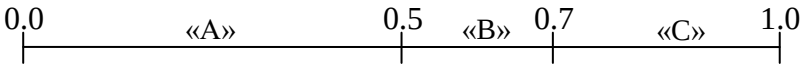
По алгоритму Хаффмана для сжатия файла необходимо прочитать его полностью и подсчитать, сколько раз встречается каждый символ исходного алфавита, подлежащий кодированию, то есть подсчитать вероятность появления каждого символа. Затем нужно построить двоичное дерево, каждый узел которого имеет суммарную вероятность всех узлов, находящихся ниже него, а

каждый лист (конечный узел) содержит один символ. Для построения дерева сначала добавим в корневой узел листья со всеми символами алфавита. Затем итеративно будем повторять следующий алгоритм, пока в корневом узле не останется всего два узла: найдем два узла с наименьшей вероятностью и вынесем их в отельный узел, у которого в качестве вероятности запишем сумму вероятностей найденных узлов. В конечном итоге мы построим дерево, каждый узел которого имеет суммарную вероятность всех узлов, находящихся ниже него. Прослеживаем путь к каждому листу дерева, помечая направление к каждому узлу (например, направо – 1, налево – 0). Выполнив эту процедуру для каждого листа, получим бинарные коды для каждого изначального элемента – таблицу Хаффмана.

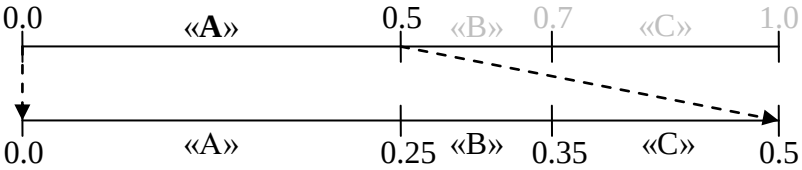
В чистом виде алгоритм Хаффмана для сжатия изображений практически не используется, однако он является ключевым алгоритмом, обеспечивающим сжатие в форматах JPEG и PNG.

Арифметическое кодирование

Из теории информации известно, что для символа с вероятностью появления равной N в идеале следует использовать код длиной $-\log_2(N)$ бит. Код Хаффмана присваивает каждому символу алфавита код с целым числом бит, поэтому если вероятности символов не выражаются числами $1/2^n$, то код Хаффмана не является эффективным. Результатом арифметического кодирования является дробное число в интервале от нуля до единицы, передающее весь файл. Рассмотрим пример кодирования последовательности «ABCABC», если вероятности символов следующие: A – 0.5, B – 0.2, C – 0.3. В арифметическом кодере каждый символ представляется интервалом в диапазоне чисел [0, 1) в соответствии с частотой его появления. В данном примере, для символов нашего алфавита получим следующие наборы интервалов:



Считываем первый символ «A» – ему соответствует интервал [0, 0.5), поэтому результат будет лежать в [0, 0.5). Спроецируем исходный набор вероятностей в этот диапазон:



Следующий символ «B» – ему соответствует интервал [0.25, 0.45). Соответственно проецируем на этот интервал исходный набор вероятностей и повторяем шаги, пока не кончится входная последовательность символов. Затем выбираем число из конечного интервала, содержащее наименьшее число бит – именно это число и будет содержать информацию обо всей последовательности входных символов. В следующей таблице показан результат обработки последовательности «ABCABC»:

№	Символ	L	R	Границы
1	**A**	0	0.5	0+(1-0)*0.0 = 0 0+(1-0)*0.5 = 0.5
2	**B**	0.5	0.7	0+(0.5-0)*0.5 = 0.25 0+(0.5-0)*0.7 = 0.35
3	**C**	0.7	1	0.25+(0.35-0.25)*0.7 = 0.32 0.25+(0.35-0.25)*1.0 = 0.35
4	**A**	0	0.5	0.32+(0.35-0.32)*0.0 = 0.32 0.32+(0.35-0.32)*0.5 = 0.335
5	**B**	0.5	0.7	0.32+(0.335-0.32)*0.5 = 0.3275 0.32+(0.335-0.32)*0.7 = 0.3305
6	**C**	0.7	1	0.3275+(0.3305-0.3275)*0.7 = **0.3296** 0.3275+(0.3305-0.3275)*1.0 = **0.3305**

Итоговый интервал [0.3296, 0.3305), ширина интервала 0,0009. 10 бит достаточно для передачи числа с точностью до ширины интервала. Соответственно, результат: 338/1024 = 0,330078.

Рассмотрим процесс декодирования. Число 0,330078 лежит в диапазоне [0, 0.5), это значит, что первым идет символ «A». Чтобы найти следующий символ необходимо

интервал текущего символа растянуть до полного интервала [0, 1) и соответствующим образом изменить значение ключевого числа. Для преобразования необходимо вычесть левую границу диапазона и разделить на ширину диапазона. Далее цикл повторяется. Результат декодирования показан в следующей таблице:

№	Символ	L	R	
1	**A**	0	0,5	(**0,330078**-0)/(0,5-0) = 0,660156
2	**B**	0,5	0,7	(0,660156-0,5)/(0,7-0,5) = 0,800780
3	**C**	0,7	1	(0,800780-0,7)/(1-0,7) = 0,335933
4	**A**	0	0,5	(0,335933-0)/(0,5-0) = 0,671867
5	**B**	0,5	0,7	(0,671867-0,5)/(0,7-0,5) = 0,859333
6	**C**	0,7	1	(0,859333-0,7)/(1-0,7) = 0,531111

1.11 Сложные алгоритмы сжатия изображений: JPEG, Wavelet.

Алгоритм JPEG

Алгоритм JPEG разрабатывался объединённой группой экспертов в области фотографии (Joint Photographic Experts Group) и в настоящий момент является наиболее распространенным методом сжатия одиночных фотореалистичных изображений.

При сжатии изображение переводится в цветовую систему YCbCr (YUV). Далее каналы изображения Cb и Cr, отвечающие за цвет, уменьшаются в 2 раза (4:2:2), поскольку цветовое разрешение человеческого зрения гораздо хуже яркостного. Реже используется уменьшение цветовой информации в 4 раза (4:1:1 или 4:2:0) или сохранение размеров цветовых каналов как есть (4:4:4). Далее компоненты Y, Cb и Cr сжимаются независимо. Каждая компонента разбивается на блоки 8x8 значений (байт), начиная с верхнего левого угла. Если размеры изображения не кратны 8, то изображение расширяется (крайние пиксели копируются).

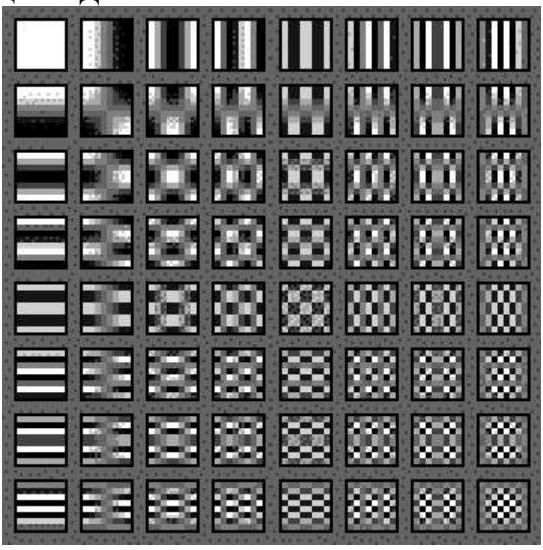
Каждый блок 8x8 подвергается двумерному дискретно-косинусному преобразованию ДКП (DCT). Формула преобразования в общем виде:

$$C(i,j) = \frac{A(i)A(j)}{\sqrt{2N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} V(x,y) \cdot \cos\left(\frac{\pi i(2x+1)}{2N}\right) \cos\left(\frac{\pi j(2y+1)}{2N}\right)$$

где V(x,y) – значение компонент Y, Cb или Cr,

$$A(k) = \begin{cases} \frac{1}{\sqrt{2}}, k = 0 \\ 1\ , k > 0 \end{cases}$$

Следующий рисунок показывает, свертку с каким изображением нужно сделать, чтобы получить соответствующий коэффициент ДКП.



Чем больше значение i и j (то есть чем правее и ниже компонента ДКП), тем выше частота косинусов в преобразовании. Таким образом, слева сверху располагаются компоненты, передающие информацию о наиболее важных «крупных» элементах изображения, а справа снизу – компоненты, передающие информацию о наименее важных «мелких» элементах изображения. Коэффициенты преобразования обходятся зигзагом с левого верхнего угла в правый нижний, в результате чего все низкочастотные компоненты, несущие основную информацию об

изображении оказываются в начале строки, а все высокочастотные компоненты – в конце строки.

Полученные коэффициенты подвергаются квантованию делением на строку такой же длины, у которой делители монотонно возрастают от начала к концу строки. Конкретный набор делителей задается параметром «качество». Чем ниже «качество», тем больше делители, и тем меньше по абсолютному значению оказываются коэффициенты после квантования. Строка с делителями обязательно передаются в выходной поток для декодирования.

Далее строка коэффициентов сжимается вариантом алгоритма RLE – цепочка нулей заменяется счетчиком нулей. Получившиеся пары {число нулей и первое ненулевое значение} упаковываются с помощью кодов Хаффмана.

Таким образом, потеря информации в алгоритме JPEG происходит в двух местах: при уменьшении компонент цветности (Cb и Cr) и при квантовании коэффициентов ДКП.

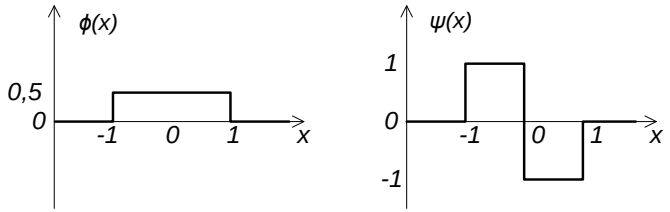
Характерные проблемы JPEG

Базовой функцией ДКП является косинус. В частотном представлении косинус является дельта-функцией, однако в пространственном представлении он бесконечен. Именно из-за неубывания косинуса приходится разбивать компоненты изображения на блоки. Каждый блок сжимается независимо, поэтому даже небольшие искажения в соседних блоках будут заметны вдоль протяженных границ блоков. Собственно «блочность» является самой неприятной проблемой, возникающей при сжатии изображений в JPEG. Другая проблема проявляется при кодировании синтезированных изображений, изобилующих четкими контрастными переходами и линейными градиентами. Из Фурье анализа известен «эффект Гиббса», возникающий из-за ограничения спектра при резком (пороговом) изменении значения функции. Алгоритм JPEG сильнее всего «портит» высокочастотные компоненты, поэтому даже при относительно высоком «качестве» сжатия вдоль резких переходов (контуров) возникают обратные переходы (ложные контура). Аналогичный эффект наблюдается при кодировании больших областей с плавной (линейной) интерполяцией цветов. При понижении «качества», никак не проявляющемся на обычных изображениях, на линейных

градиентах возникают небольшие колебания, воспринимаемые зрителем как текстурное свойство объекта, не существующее на самом деле.

WaveLet

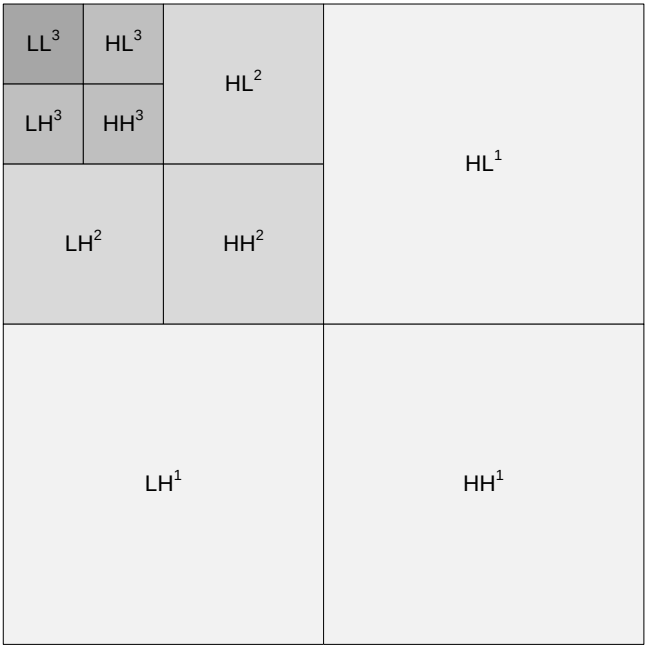
В алгоритме JPEG2000 вместо дискретно-косинусного преобразования используется вейвлет-преобразование (WaveLet). Вейвлеты – это семейство функций, которые локальны во времени и по частоте, и в которых все функции получаются из пары базовых функций посредством их сдвигов и растяжений по оси времени. Локальность функций вейвлет-преобразования порождает ряд кардинальных преимуществ. В первую очередь свертка исходного изображения с конкретной функцией не требует всего изображения – достаточно обрабатывать по несколько пикселей (по ширине окна базовой функции). Вычислительная сложность растет линейно (а не квадратично) от полного числа пикселей в изображении. Соответственно, нет необходимости разбивать изображение на блоки – можно выполнять преобразование над всем изображением сразу. Базовые функция вейвлет-преобразования похожи на функцию $\mathrm{sync}(x) = \frac{\sin(x)}{x}$, которая выглядит одинаково и в пространственном, и в частотном представлении. «Материнская» функция имеет нулевое значение интеграла по времени и играет роль высокочастотного фильтра (H-фильтр). Парная к ней «отцовская» базовая функция должна иметь единичный интеграл, обеспечивая низкочастотную фильтрацию входного сигнала (L-фильтр). Простейшим примером базовых функций вейвлет-преобразования являются функции Хаара – «отцовская» ϕ(x), «материнская» ψ(x):



После преобразования строки из N значений (например, N компонент яркости) получится N «средних» (L) и N «отличий» (H),

то есть всего 2N коэффициентов. Для однозначного восстановления N исходных значений достаточно оставить N коэффициентов, поэтому преобразование делается с шагом 2.

Аналогично выполняется преобразование по вертикали. В результате получается 4 группы коэффициентов: LL (усредненное изображение), LH (отличия по вертикали), HL (отличия по горизонтали) и HH (отличия по вертикали и по горизонтали одновременно). Усредненное изображение LL подвергается преобразованию еще раз, вновь разделяясь на 4 группы: LL^2, LH^2, HL^2 и HH^2. Затем еще несколько раз (обычно 3-4 раза).



В итоге получается несколько групп коэффициентов, из которых самое усредненное (например, LL^4) несет основную информацию и практически не сжимается, а все «отличия» содержат большое количество нулевых или близких к нулевым значениям, и, следовательно, могут сильно сжиматься статистическими алгоритмами типа Хаффмана.

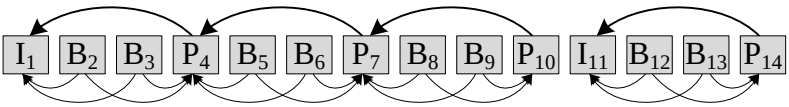
1.12 Алгоритмы сжатия последовательностей изображений: MPEG (1,2,4), AVC, HEVC.

Алгоритм MPEG

Алгоритм MPEG разрабатывается экспертной группой по вопросам движущегося изображения (Motion Picture Experts Group) и в настоящее время является наиболее распространенным алгоритмом сжатия последовательностей изображений. Алгоритм учитывает «схожесть» частей изображений в последовательности, кодируя не само исходное изображение, а разницу между похожими кусочками исходного и уже закодированного (и переданного) изображения.

Так же, как и в JPEG-е, исходное изображение переводится в цветовое пространство YCrCb. Затем цветовые компоненты масштабируются, но уже не в 2 раза, а в 4 раза (рекомендуется формат 4:2:0). Далее каждая из компонент разбивается на блоки 8x8, но дальше идет принципиальное отличие от алгоритма JPEG – для каждого блока указывается некоторый опорный блок 8x8, из которого сжимаемый блок вычитается и дискретно-косинусное преобразование выполняется над полученной разностью. Далее коэффициенты ДКП квантуются, обходятся «зигзагом», подвергаются RLE кодированию в пары {число нулей и ненулевое значение} и упаковываются с помощью кодов Хаффмана, как и в алгоритме JPEG. В стандарте MPEG-1 определено три типа кадров, отличающихся способом выбора опорного блока 8x8:

- I-кадр (Intra) – опорные блоки не задаются и кадр сжимается аналогично алгоритму JPEG.
- P-кадр (Predicted) – опорный блок берется из предыдущего I-кадра или P-кадра.
- B-кадр (Bidirectional) – опорный блок формируется линейной интерполяцией из двух блоков 8x8 в предыдущем и последующем I-кадрах и/или P-кадрах (один предшествующий данному кадру и один следующий за ним):



Порядок изображений на выходе кодера меняется, поскольку декодер при декодировании B-кадра должен сначала получить и декодировать следующий за ним по времени P-кадр. Для представленного выше примера последовательность кадров на выходе кодера будет такой: I_1, P_4, B_2, B_3, P_7, B_5, B_6, P_{10}, B_8, B_9, I_{11}, P_{14}, B_{12}, B_{13}. Последовательность от одного I-кадра до следующего I-кадра называется GOP-ом (Group of Picture). Как правило, длительность GOP-а не должна превышать 12-15 кадров, поскольку декодер должен дождаться начала GOP-а в потоке (или в файле), прежде чем он сможет декодировать и показать зрителю хоть одно изображение.

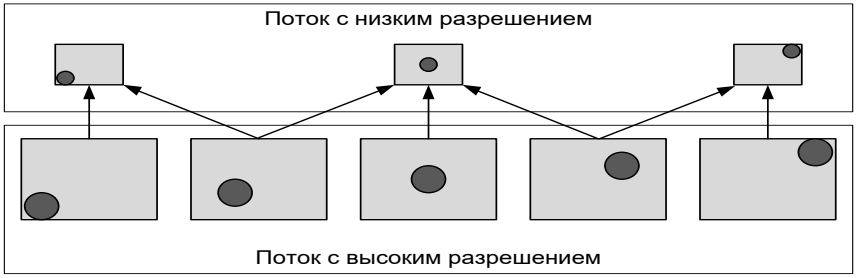
Опорный блок задается в виде вектора смещения с точностью до пикселя или до половины пикселя (при этом опорное изображение интерполируется). Вектор смещения задается для макроблока 16x16 пикселей. При кодировании P-кадра сжимаемое изображение разбивается на макроблоки 16x16 пикселей (начиная с верхнего левого угла) и для каждого макроблока ищется наболее похожий на него квадрат (16x16 пикселей) в предыдущем I-кадре или P-кадре. Очевидно, что поиск занимает очень много времени, поэтому кодирование в MPEG требует гораздо больше вычислительных ресурсов, чем декодирование. При кодировании B-кадра сжимаемое изображение также разбивается на макроблоки 16x16 пикселей и для каждого макроблока ищется наиболее похожие на него квадраты как в предыдущем, так и в последующем за ним I- и P-кадрах. Для пары найденных квадратов указываются два вектора смещения и задается коэффициент интерполяции α (от 0 до 1). Декодер интерполирует с указанным коэффициентом изображения (16x16 пикселей) из восстановленных предыдущего и последующего I- и P-кадров и добавляет его к декодированным блокам 8x8 значений компонент Y, Cr и Cb.

Если кодер не может найти в опорном кадре ни одного квадрата 16x16 пикселей, похожих на сжимаемый макроблок, то все блоки макроблока сжимаются как в алгоритме JPEG (без вычитания опорного блока). Поскольку декодер может использовать в качестве опорных блоков только восстановленные изображения, то и кодер обязан при сжатии формировать опорный блок 8x8 только из восстановленного изображения.

Стандарт MPEG-2

Стандарт MPEG-1 был принят в 1993 году и использовался для записи фильмов на обычном компакт-диске (Video CD). Поток данных с компакт-диска составляет 1150 кбит в секунду (~1 Mb/s). Разрешение кадра – 360x240x15 кадров в секунду, то есть половинное разрешение по горизонтали и по вертикали. Стандарт MPEG-2, принятый в 1995 году, включил доработки, необходимые для поддержки цифрового телевещания. В первую очередь это означает поддержку как стандартных разрешений (720x576x25 кадров в секунду и 720x480x30 кадров в секунду), так и всего спектра разрешений ТВЧ (HDTV) вплоть до разрешения 1920x1080. Также в стандарте описано понятие транспортного потока (Transport Stream), позволяющего передавать в одном потоке данных несколько телевизионных передач, каждая из которых может содержать несколько звуковых и видео потоков.

В стандарте предусмотрена возможность одновременного кодирования нескольких видео-потоков разного разрешения, полученных из одной последовательности изображений. Это актуально при приеме одной и той же спутниковой программы на большой телевизор с разрешением 1920x1080 и на маленький телефон с разрешением 320x240. Телефон, как правило, не имеет вычислительных мощностей, достаточных для декодирования видео с максимальным разрешением. Для него нужно передавать видео с разрешением 320x240 или 720x576 с обычной частотой кадров (25 кадров в секунду). Но тогда при сжатии видео с высоким разрешением (1920x1080) и/или с повышенной частотой кадров (50 кадров в секунду) имеет смысл использовать уже закодированную и переданную информацию, просто вычитая из сжимаемого изображения интерполированное изображение с меньшим разрешением:



Стандарт MPEG-4

Стандарт MPEG-3 не был принят, в том числе потому, что широкое распространение получил формат сжатия звуковых файлом MP3, реально являющийся подмножеством стандарта MPEG-1 (MPEG-1 Layer 3). Стандарт MPEG-4 включает очень широкий набор средств для сжатия видео и звука при передачи по низкоскоростным линиям (в первую очередь через «Интернет» и телефонные каналы).

Широкое распространение получил алгоритм сжатия, описанный в 10-й части стандарта (MPEG-4 Part 10), получивший название AVC или H.264. Его ключевые особенности:

- использование до 16 опорных кадров (вместо 2 у B-кадра);
- использование нескольких (до 16) векторов смещения для макроблока, когда макроблок разбивается на прямоугольники вплоть до 4x4 и каждому из них указывается свой вектор смещения;
- возможность задания вектора смещения внутри кодируемого кадра, что бывает очень удобно при сжатии I-кадра;
- задание вектора смещения с точностью до четверти пикселя;
- использование алгоритмов CABAC (context-adaptive binary arithmetic coding) и CAVLC (context-adaptive variable-length coding) вместо кодов Хаффмана;
- использование специальных фильтров для устранения «блочности» в декодированном изображении;
- полностью целочисленная реализация, позволяющая восстанавливать изображение с абсолютной точностью.

Стандарт HEVC

Стандарт HEVC (High efficiency video coding) или H.265 следующий шаг в развитии кодирования видео. Его ключевые особенности:

- вместо макроблоков используются блоки с иерархической структурой кодирования (очень большие);
- задание вектора смещения с точностью до 1/8 пикселя;
- наличие двумерных неразделимых, разделимых и направленных интерполяционных фильтров;
- сравнительная система кодирования вектора движения;

- адаптивное предсказание ошибок кодирования в пространственной и частотной областях;
- адаптивный выбор матрицы квантования;
- режимо-зависимое внутрикадровое кодирование.

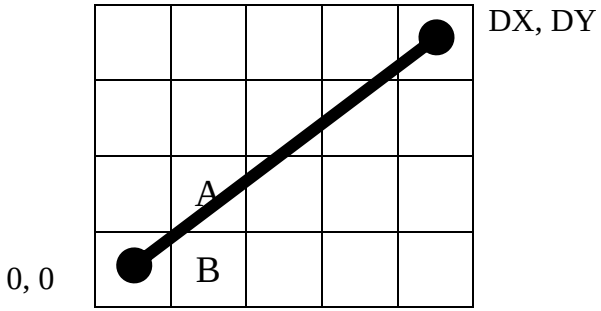
Кодек H.264 имеет на 20%..50% более высокую эффективность, чем кодек MPEG-2 при кодировании наиболее распространенного контента. Аналогично, кодек H.265 имеет на 20%..50% более высокую эффективность, чем кодек H.264, но также на стандартном контенте. Речь идет про кодирование качественно сделанных фильмов, студийных передач и другого контента, для которого характерно наличие четкого объекта и нечеткого или неизменного фона. Объект не выходит за пределы экрана, поэтому его легко повторить из предыдущих кадров. Фон либо неподвижен, либо очень размыт, поэтому его можно очень сильно «портить». Если же передавать сложную динамическую картинку, например, транслировать футбольный матч, то все эти алгоритмы показывают примерно одинаковую степень сжатия. Это связано с тем, что основной объект передачи – футболисты, мяч и футбольное поле содержат меньше деталей, чем фон – трибуны, заполненные яркими разноцветными пятнами. При этом камера перемещается по полю вслед за мячом с очень высокой скоростью. Поэтому алгоритмы сжатия начинаю в первую очередь терять информацию о травяном покрытии – поле начинает выглядеть однородным, что сразу бросается в глаза зрителю.

1.13 Растрирование на плоскости. Алгоритм Брезенхема для растрирования отрезка и дуги. Кривые Безье. Векторизация кривых. Растрирование контуров.

Алгоритм Брезенхема

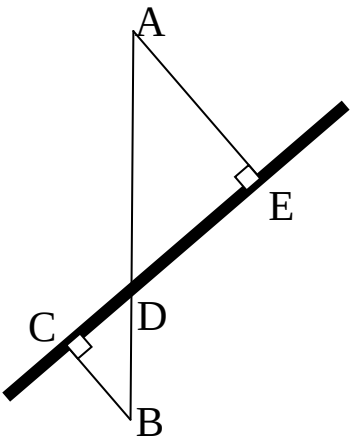
Алгоритм Брезенхема служит для растрирования отрезка на плоскости. Необходимо построить линию единичной толщины, максимально точно приближающую отрезок. Пусть концы отрезка заданы в точках растра, причем начало отрезка совпадает с началом системы координат {0,0}. Необходимо растрировать отрезок из точки {0,0} в точку {DX,DY}. В процессе работы одна из координат – либо x, либо y (в зависимости от углового коэффициента) – всегда изменяется на единицу. Изменение другой координаты (либо на нуль, либо на единицу) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние мы назовем ошибкой. Алгоритм построен так, что на каждой итерации требуется проверять лишь знак этой ошибки.

Приведем алгоритм Брезенхема для первого октанта, то есть для случая 0 <= DY <= DX.



Очевидно, что при работе в первом квадранте на каждом шаге алгоритма необходимо шагать либо вправо, либо вправо вверх. То есть нужно увеличивать значение x на единицу, а значение y оставлять старым или увеличивать на единицу.

Рассмотрим прямую возле двух центров сетки растра (точки A и B на рисунке выше):



Точки A и B – центры пикселей. Отрезки AE и BC соответствуют расстоянию от центра пикселя до прямой. Какое из расстояний (AE или BC) меньше, такой пиксель (A или B) и нужно выбрать для растрирования. Из подобия треугольников ADE и BDC следует, что вместо сравнения расстояний AE и BC можно сравнивать расстояния AD и BD. Уравнение прямой $y = \frac{DY}{DX} \cdot x$, поэтому расстояние BD (текущая ошибка err) легко вычисляется на каждом шаге алгоритма – при очередном шаге вправо к ошибке добавляется величина DY/DX. Если получившаяся ошибка меньше половины единицы, то нужно шагать вправо, иначе надо шагать вверх и из ошибки вычесть единицу.

Получаем следующий алгоритм:

```
// Инициализация переменных (начальный шаг алгоритма):
x = 0,  y = 0,  err = 0
// Очередной шаг алгоритма:
x += 1;
err += DY/DX;
if( err > 0,5 ) { // нужно шагать вправо вверх
   y += 1;
   err -= 1;
} else { // нужно шагать вправо
   // ошибка не меняется
}
```

Для перехода в целые числа умножим ошибку на 2*DX и установим начальное значение ошибки -0.5, чтобы сравнивать ошибку с нулевым значением (т.е. сравнивать только знак ошибки):

```
// Инициализация переменных (начальный шаг алгоритма):
X = 0,  Y = 0,  err = -DX
// Очередной шаг алгоритма:
x += 1;
err += 2*DY;
if( err > 0 ) { // нужно шагать вправо вверх
   y += 1;
   err -= 2*DX;
} else { // нужно шагать вправо
   // ошибка не меняется
}
```

Также существует алгоритм Брезенхема для растрирования дуги, использующий два сравнения на каждом шаге.

Кривая Безье

Кривые Безье являются одним из самых популярных способов моделирования гладких линий в компьютерной графике. Кривые Безье являются параметрическими кривыми (параметр t изменяется от 0 до 1). В общем виде кривая порядка n задается формулой:

$B(t) = \sum_{i=0}^{n} P_i b_{i,n}(t)$

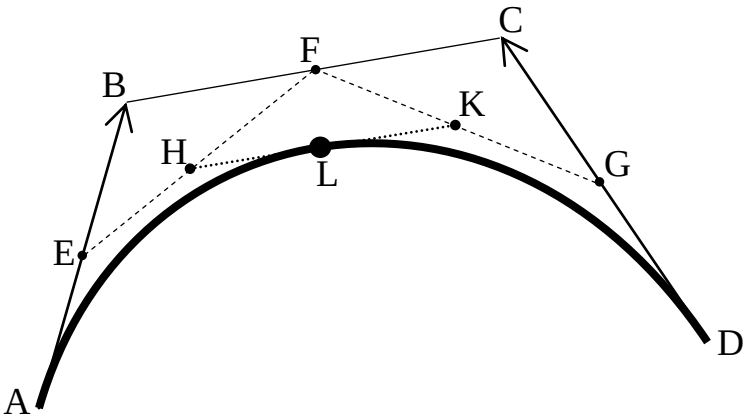
где Pi – контрольные точки кривой,

$$b_{i,n}(t) = \frac{n!}{i!\,(n-i)!} t^i (1-t)^{n-i}$$

Кубическая кривая Безье задается формулой:

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

Векторизация кубической кривой Безье (то есть перевод кривой в набор отрезков) может выполняться простым рекурсивным алгоритмом, опирающемся на простой способ деления кривой Безье на две кривые Безье меньшего размера:



Пусть исходная кривая ABCD задается двумя конечными точками A и D (через них кривая проходит) и двумя контрольными точками B и C (вектора AB и DC являются касательными к кривой в точках A и D соответственно). Разделим пополам отрезки AB, BC и CD в точках E, F и G. Соединим точки в два отрезка EF и FG. Эти отрезки вновь разделим пополам в точках H и K. Отрезок HK вновь разделим пополам в точке L. Точка L будет точкой исходной кривой, а отрезок HK будет касательной к кривой в точке L. Мы получили две кривые: AEHL и LKGD.

Рекурсивный алгоритм делит исходную кривую на две части и для каждой части оценивает, нельзя ли заменить ее отрезком, проходящим через ключевые точки соответствующей кривой. Если заменить нельзя, то алгоритм вызывает себя для каждой из вновь созданных кривых. Чтобы принять решение о замене кривой отрезком достаточно сравнить расстояние от контрольных точек кривой до отрезка. Если обе контрольные точки отстоят от отрезка не более величины ошибки (обычно, половины пикселя достаточно), то кривую можно заменить отрезком без видимых искажений.

Растрирование контуров (текста)

Символы текста описываются отрезками и кубическими кривыми Безье. Каждый символ состоит из одного или нескольких замкнутых контуров. Для растрирования текста достаточно векторизовать каждую из кривых Безье (например, вышеописанным

рекурсивным алгоритмом) и разделить контура на монотонные по вертикали фрагменты. Далее фрагменты нужно упорядочить по вертикали, инвертировав часть фрагментов, идущих снизу вверх. Теперь можно растрировать текст, начиная с самых верхних строк. На очередной строке необходимо перебрать все фрагменты и с помощью алгоритма Брезенхема получить очередную координату пересечения этого фрагмента с текущей строкой. Полученные координаты необходимо упорядочить по возрастанию и разбить на пары. В каждой паре от первой координаты до второй включительно выполняется заливка цветом символа.

Исключением является растрирование строго горизонтальных отрезков, которые проще растрировать отдельно.

1.1 Микширование в прямом и обратном порядке.

Традиционное микширование (смешивание) цветов: $C_{res} = C \bullet A + C_{back} \bullet (1 - A)$, где C_{back} - цвет фона, C - накладываемый сверху цвет, а A - его непрозрачность (вес). В этой формуле предполагается, что фон полностью непрозрачный. Аналогично можно обработать произвольное число слоев (граней), накладывая их один на другой, каждый со своей прозрачностью. Обратите внимание, что при смене порядка микширования слоев результат может измениться.

А как быть, если у нас нет нижнего непрозрачного слоя?

Рассмотрим стопку из n слайдов, каждый из которых имеет цвет C_n и непрозрачность A_n. С учетом только верхнего слайда результирующий цвет стопки равен $C_{res} = C_1 \bullet A_1 + C_{back} \bullet (1 - A_1)$, где C_{back} - цвет фона. С учетом второго слайда $C_{res} = C_1 \bullet A_1 + C_2 \bullet A_2 \bullet (1 - A_1) + C_{back} \bullet (1 - A_2) \bullet (1 - A_1)$.

Аналогично с учетом всех n слайдов результирующий цвет будет вычисляться по следующей формуле:

$$C_{res} = C_1 \bullet A_1 + C_2 \bullet A_2 \bullet (1 - A_1) + \ldots + C_n \bullet A_n \bullet \prod_{k=1}^{n-1} (1 - A_k) + C_{back} \bullet \prod_{k=1}^{n} (1 - A_k) \quad (1)$$

Если заменить стопку слайдов одним цветом C_x с непрозрачностью A_x, то для него можно написать:

$$C_{res} = C_x \bullet A_x + C_{back} \bullet \left(1 - A_x\right) \quad (2)$$

Из сравнения (1) и (2) можно найти непрозрачность и цвет стопки слайдов:

$$A_x = 1 - \prod_{k=1}^{n} \left(1 - A_k\right) \quad (3)$$

$$C_x = \frac{C_1 \bullet A_1 + C_2 \bullet A_2 \bullet \left(1 - A_1\right) + \ldots + C_n \bullet A_n \bullet \prod_{k=1}^{n-1} \left(1 - A_k\right)}{1 - \prod_{k=1}^{n} \left(1 - A_k\right)} \quad (4)$$

Рассмотрим пару величин: цвет, умноженный на непрозрачность (premultiplied color) $P = C \bullet A$ и величину, обратную непрозрачности, т.е. прозрачность $Tr = 1 - A$. Для этой пары величин формулы (3) и (4) приобретут простой и понятный вид:

$$Tr_x = \prod_{k=1}^{n} Tr_k \quad (5)$$

$$P_x = P_1 + P_2 \bullet Tr_1 + \ldots + P_n \bullet \prod_{k=1}^{n-1} Tr_k \quad (6)$$

Видно, что пара формул (5) и (6) может быть легко реализована итеративно при последовательном микшировании произвольного числа слайдов, начиная с самого верхнего.

Обратите внимание, что при этом получается результирующая прозрачность стопки слайдов и ее цвет, умноженный на непрозрачность (premultiplied color). Результирующие значения цвета и непрозрачности (традиционно используемые в компьютерной графике) могут быть легко восстановлены из вычисленных *Tr* и *P*:

$$A = 1 - Tr$$

$$C = P \big/ A$$

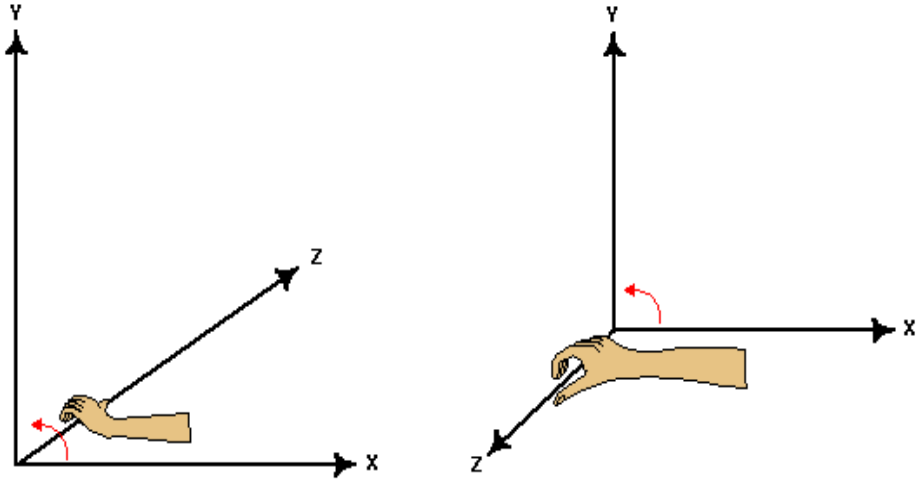
2 Трехмерная графика

2.1 Геометрия

Вектор в пространстве. Системы координат. Проекция. Обобщённые координаты.

Поскольку трехмерная графика имеет дело с изображением трехмерных сцен и объектов, то и при расчетах этих изображений используется соответствующий мат-аппарат. Одним из основных терминов является трёхмерный вектор, представляемый тремя координатами – x, y, z. Говоря о векторе в пространстве, разумеется, необходимо определить и само пространство. Введём понятие системы координат. Система координат определяется одной точкой – началом системы координат и тремя ортогональными нормированными векторами - ортами. Мы будем использовать такие системы координат, в которых при взгляде по орте z – орта x направлена вправо, а орта y вверх. Такие соглашения приняты в пакетах DirectX и OpenGL

Каждая система координат расположена тем или иным образом относительно другой системы координат. Для определённости используют некую «мировую» систему координат,

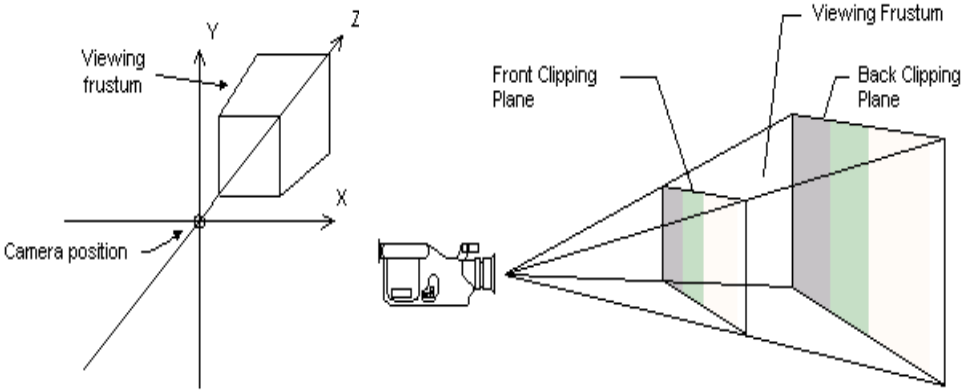


которая является «абсолютной». Все остальные системы координат

определяются относительно неё, либо относительно тех систем координат, которые определяются относительно неё.

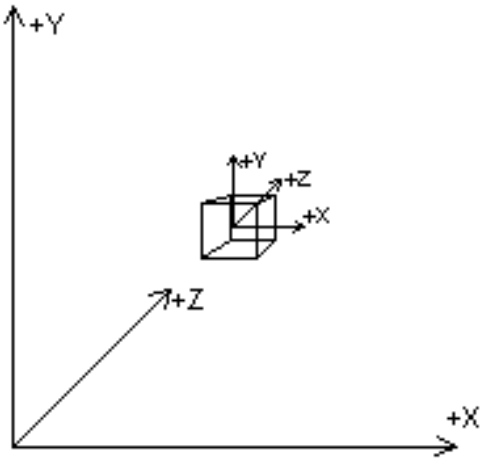
При построении изображений трёхмерного мира очень часто нужно уметь переводить вектора из одной системы координат в другую. Для осуществления этой операции используют векторно-матричный аппарат. Т.е. для каждой системы координат существует такая матрица, которая переводит все вектора в родительскую систему координат (систему координат, в которой определено начало и орты). Соглашение при перемножении вектора на матрицу – строка-вектор, стоящая слева, умножается на столбцы матрицы, стоящей справа.

Изображение трёхмерной сцены, по сути, является двумерным, поскольку представляется на экране монитора. Для того, чтобы понять, где трёхмерная точка будет отображена на двумерной плоскости (плоскости экрана) используется перспективная проекция. Трехмерные пакеты, как правило, позволяют строить и ортографические проекции, но сейчас мы будем обсуждать именно перспективную проекцию.



Вернёмся к вопросу о векторно-матричном аппарате. Если использовать трёхмерные вектора, то матрицы, на которые они умножаются, должны иметь размерность 3x3. При помощи таких матриц можно задавать повороты, масштабирование и прочее, но нельзя задать сдвиг всех векторов на заданный вектор. Для того,

чтобы задать одну систему координат относительно другой, сдвиги нужны обязательно.



Кроме этого, нужно уметь осуществлять перспективное преобразование. Для этого было предложено использовать обобщенные векторно-матричные преобразования. Обобщенный вектор (x, y, z, w) – считается, что вектор всегда можно нормировать так, чтобы w была равна 1.

Матрица в этом случае должна быть четыре на четыре:

$$(x', y', z', w') = (x, y, z, w) \times \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}, \text{ т.е.}$$

$$x' = x \times M_{11} + y \times M_{21} + z \times M_{31} + w \times M_{41}$$
$$y' = x \times M_{12} + y \times M_{22} + z \times M_{32} + w \times M_{42}$$
$$z' = x \times M_{13} + y \times M_{23} + z \times M_{33} + w \times M_{43}$$
$$w' = x \times M_{14} + y \times M_{24} + z \times M_{34} + w \times M_{44}$$

Перенос:

$$(x', y', z', 1) = (x, y, z, 1) \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

Вращение:

вокруг оси X:

$$(x', y', z', 1) = (x, y, z, 1) \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

вокруг оси Y:

$$(x', y', z', 1) = (x, y, z, 1) \times \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

вокруг оси Z:

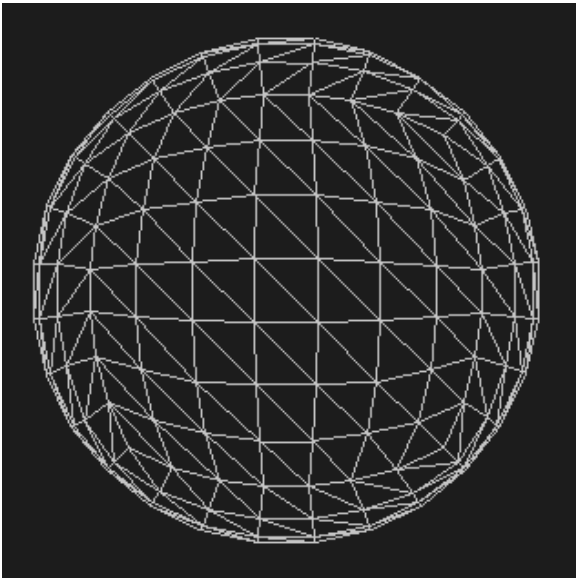
$$(x', y', z', 1) = (x, y, z, 1) \times \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Масштабирование:

$$(x', y', z', 1) = (x, y, z, 1) \times \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Грань. Проекция грани. Клипирование.

Все трехмерные поверхности, отображаемые на экране, имеют описание в виде набора граней. Мы будем обсуждать только треугольные грани, поскольку треугольник очень удобен для растрирования (например, всегда является плоской выпуклой фигурой).



Таким образом, чтобы получить изображение нужно спроецировать каждую грань на проективную плоскость, после чего задача становиться двумерной. Для того, чтобы спроецировать грань, нужно спроецировать ее ребра, а для этого нужно спроецировать ее вершины.

В ряде ситуаций возникают проблемы, например, что делать, если проекцию вершины на плоскость построить нельзя.

Кроме этого, на экране монитора будет отображаться только часть поверхности, ограниченная краями монитора. Соответственно, часть граней не нужно проецировать вовсе, а часть нужно обрезать линиями, соответствующими краям монитора. Часть объектов может находиться далеко, поэтому их тоже не нужно отображать. Для разрешения этих вопросов предложена

операция клипирования, т.е. обрезания не нужных для отображения частей. Обрезание происходит по шести плоскостям – передней (соответствующей плоскости проектирования), задней (также параллельной ортам xy) и четырём плоскостям, проходящим через начало системы координат наблюдателя и через стороны прямоугольника, соответствующего экрану монитора. Операция клипирования предшествует операции проецирования, тем самым решаются выше описанные проблемы.

Односторонние поверхности. Отбраковка. Заливка. Понятие первичного и вторичного буфера

В большинстве случаев поверхность имеет одну сторону и нет необходимости отображать поверхность, если она ориентирована к наблюдателю обратной стороной. Для отбраковки треугольников, повёрнутых к наблюдателю тыльной частью, существует, как правило, настройка механизма отбраковки. Отбраковка осуществляется сообразно тому, в каком порядке происходит обход грани со стороны наблюдателя – по часовой стрелке или против.

После клипирования и проецирования мы получим на экране двумерный многоугольник (дополнительные углы могут образоваться в результате клипирования). Этот многоугольник можно залить цветом, соответствующим данной поверхности. Поступив подобным образом со всеми треугольниками сцены, мы получим изображение сцены на экране.

Построение сцены – процесс не бесконечно быстрый. Поэтому, если грани отображать непосредственно на экран монитора, то мы увидим этот процесс. Для того, чтобы все грани сцены показывались одновременно, используют два буфера с изображением сцены. Первый содержит уже полностью построенное изображение и именно его предъявляют зрителю. А построение следующей сцены осуществляют во второй буфер, в этот момент не видимый. После того, как во втором буфере сцена готова, его меняют с первым (например, во время обратного хода луча кадровой развертки монитора).

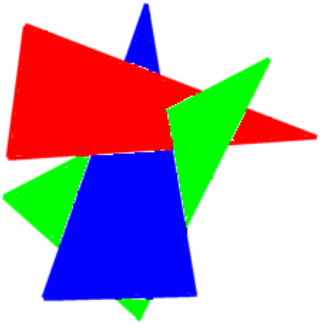
Способы удаления невидимых частей поверхностей («Художник» и «Z»).

В реальной жизни часть объектов закрывает другие и в результате некоторые объекты видны лишь частично. При отображении трехмерной сцены на экран монитора также нужно учесть этот эффект. Алгоритмы, обеспечивающие корректное изображение частично перекрытых объектов, называются алгоритмами удаления невидимых поверхностей. Мы рассмотрим два из них.

Первый называется алгоритмом «художника». Его смысл состоит в том, чтобы рисовать поверхности в порядке удаленности от наблюдателя. Первой рисуется самая удаленная поверхность, а последней – самая близкая. Этот алгоритм имеет несколько недостатков, о которых мы поговорим чуть позже.

Второй алгоритм использует понятие Z буфера. Помимо первичного и вторичного буферов, которые хранят изображение сцены, заводится дополнительный буфер, который хранит для каждого пикселя расстояние до грани. При заливке треугольника для каждого пикселя рассчитывается расстояние до него и сравнивается с тем значением, которое хранится в буфере. Если пиксель ближе, то он копируется в буфер, если дальше, то игнорируется. Недостатком второго алгоритма является большая ресурсоемкость.

Сортировка художника.



Для алгоритма «художника» можно выделить два разных подхода. В первом грани сортируются на каждом кадре. Этот вариант имеет следующие недостатки: сортировка всех граней может занимать довольно много времени. Возможны ситуации, когда отсортировать грани невозможно. Т.е. первая грань закрывает вторую, вторая третью, третья первую. В этом случае необходимо произвести разрез одной из граней.

Во втором случае сортировка производится заранее. Часть граней можно отсортировать статически. Например, есть грани, в полуплоскости нормали которых нет других граней – эти грани не зависимо от положения наблюдателя нужно рисовать последними. Наоборот, могут быть грани, которые находятся в противоположном пространстве остальных граней – их нужно рисовать первыми, так как они не закрывают никого.

Есть и динамические методы сортировки – например метод «разделяющей плоскости». Предположим, что рядом стоят два больших объекта. Предположим, что между ними можно провести плоскость. Тогда если наблюдатель находится по туже сторону от плоскости что и один из объектов, то его нужно рисовать позже, так как другой объект не может его закрывать.

В случае, когда расстояния между несколькими объектами больше их линейных размеров, их можно рисовать сообразно расстоянию до их центров, это тоже даст правильный результат. Этот вариант имеет те недостатки, что опять же есть ситуации, когда нет возможности корректно отсортировать грани и приходится резать какие-либо грани. Кроме того, в случае, когда анимация меняет расположение объектов относительно друг друга, нельзя использовать предварительную сортировку.

2.2 Цвет

Цвет. Нормаль в точке. Источники освещения. Модели освещённости.

- Цвет задаётся тройкой или четвёркой чисел R, G, B, [A].
- Пусть C1 и C2 – два цвета. Определим операции:
- Сложение: C1 + C2 == (r1 + r2, g1 + g2, b1 + b2, a1 + a2).
- Умножение: C1 * C2 == (r1 *r2, g1 * g2, b1 * b2, a1 * a2).

Четвертая компонента цвета (альфа) используется для:

1. Альфа-теста: рисовать пиксели, альфа которых удовлетворяет некоторому условию (см. IDirect3DDevice9::SetRenderState).
2. Альфа-смешивания: цвет пикселя смешивается в некоторой пропорции с цветом, находящемся в буфере кадра (back buffer) (см. IDirect3DDevice9::SetRenderState).

Для лучшего понимания мы рассмотрим несколько упрощенных формул расчета освещенности точки поверхности (N – вектор нормали к поверхности, L – вектор на источник света):

- I = N*L>0 ? N*L : 0 – характерна для космоса, так как отвернутая сторона абсолютно черна
- I = N*L>0 ? N*L + E : E – характерна для атмосферы (есть «фоновая засветка» E), но в этой формуле не учитывается нормировка (N*L + E может быть единицы)

Можно придумывать и другие. Сложность модели зависит от вычислительных возможностей рендерера. В DirectX встроена более сложная формула, но есть возможность разрабатывать свои формулы с учетом тех визуальных характеристик, которые требует конкретное приложение. Обычно учитывают и цвет источника. Например: P = Cm*I*Cl + B, где Cm – цвет материала, Cl – цвет источника света.

Глобальная модель освещения

$$L(x,\vec{w}_0) = L_e(x,\vec{w}_0) + \int_S f_r(x,\vec{w}_i \to \vec{w}_0)L(x',\vec{w}_i)G(x,x')V(x,x')dw_i$$

$L(x,\vec{w}_0)$	Интенсивность света, отраженного в точке x в направлении w_0.
$L_e(x,\vec{w}_0)$	Интенсивность света, излучаемого объектом из точки x.
$L(x',\vec{w}_i)$	Интенсивность света, приходящего от точки x' из направления w_i.
$G(x,x')$	Функция ослабления интенсивности.
$f_r(x,\vec{w}_i \to \vec{w}_0)$	Двулучевая функция отражательной способности.
$V(x,x')$	Функция видимости точки x' из точки x (1 или 0).

Локальная модель освещения

Локальная, поскольку не учитывается расположение других объектов в сцене:

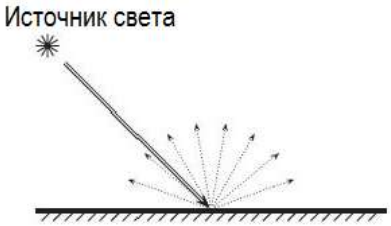
Illumination = Ambient + Diffuse + Specular + Emissive

Ambient – свет, приходящий в точку со всех направлений.

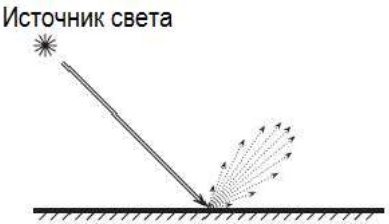
Diffuse – свет, равномерно отражаемый объектом во всех направлениях.

Specula – «зеркально» отражаемый поверхностью свет.

Emissive – свет, излучаемый объектом.



Diffuse



Specular

Ambient = Ia*Pa .

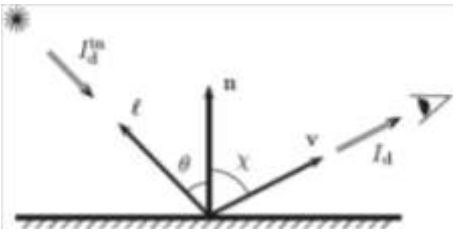
- Ia– интенсивность приходящего рассеяного освещения.
- Pa – коэффициент отражения рассеяного освещения.

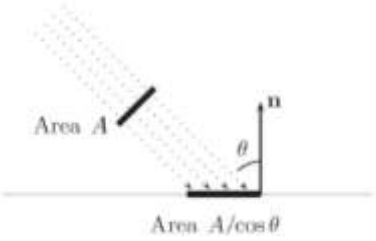
Emissive = I

Diffuse = SIGMA(ILD)

ILD = I_d* P_d * cos(θ) = I_d * P_d * (n, l).

- ||n|| = ||l|| = 1.
- I_d – интенсивность приходящего диффузного света.
- P_d – коэффициент диффузного отражения.





Блик

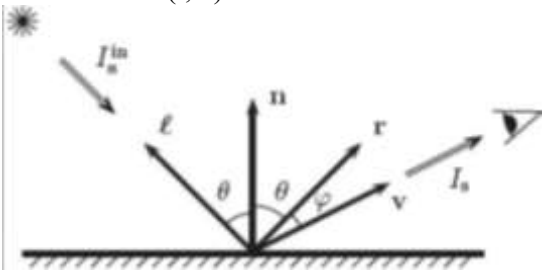
Блик рассчитывается обычно в зависимости от косинуса в степени n между отраженным лучом от источника и направлением на наблюдателя от точки поверхности, для которой ведется расчет. При этом величина n определяет остроту блика.

Specular = SIGMA(ILS)

Specular по Фонгу:

$ILS = I_s * P_s * \cos^f(\theta) = I_s * P_s * (r, v)^f$.

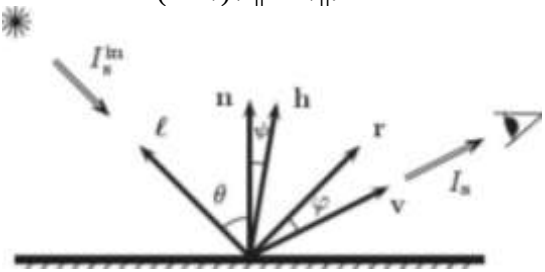
- $||r|| = ||v|| = 1$.
- $r = 2(l, n) * n - l$.



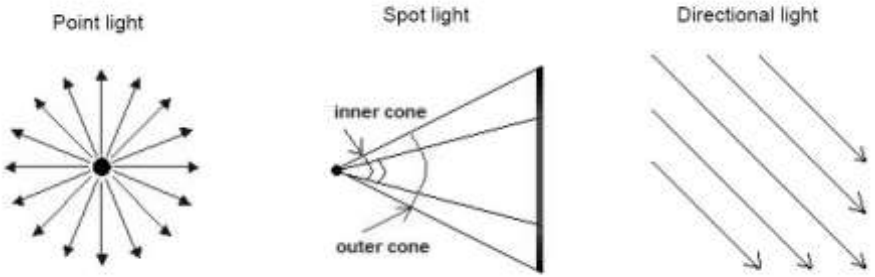
Specular по Блинну:

$ILS = I_s * P_s * \cos^f(\theta) = I_s * P_s * (h, n)^f$.

- $||h|| = ||n|| = 1$.
- $h = (l + v) / ||l + v||$.



Illumination = ambient + diffuse + specular + emissive =
$S(ILD) + S(ILS) + I_a * P_a + I_e =$
$[SIGMA_i(I_{di} * P_d * (n, l_i))] +$
$[SIGMA_i(I_{si} * P_s * (r_i, v)^f)] + I_a * P_a + I_e$



С увеличением расстояния от точки до источника света интенсивность ослабевает.
Attenuation = $1 / (a + b * d + c * d^2)$.
d – расстояние.
a, b, c – одновременно не равные 0 коэффициенты;

Пусть вершина принадлежит n треугольникам, имеющим нормали N_i, где i=[0..n-1].

Гладкие нормали: «наивный» подход

$N = (1/n) * SUMM(N_i)$

Гладкие нормали: «продвинутый» подход

$N = SUMM(d_i * N_i)$, где $1 = SUMM(d_i)$

Например, $d_i = s_i/S$, где S – сумма площадей треугольников, которым принадлежит вершина.

Преобразование нормали из одной систем координат в другую

При переводе нормали из одной системы координат в другую достаточно считать, что четвертая компонента равна 0.

Гуро интерполяция

Это упрощение позволяет рассчитывать освещенность только в вершинах треугольника. При заливке всей площади треугольника используется интерполяция цвета вдоль экрана.

Фонг

В этом методе расчет освещенности (цвета) ведется для каждого пикселя грани.

Туман

Наличие тумана или дымки существенно улучшает реалистичность картинки, а их расчет сравнительно не сложен. Нужно просто взвесить цвет тумана и цвет освещенной поверхности:

$P = (1-f)*C_p + f*C_f$

F – вычисляется из расстояния от наблюдателя до поверхности. Это расстояние и так известно при Z методе удаления невидимых поверхностей.

Полупрозрачность. (Полупрозрачность и Z) Альфа-Блендинг

Еще один эффект очень полезен при построении реалистичного изображения – полупрозрачные поверхности. Для этого при отрисовке полупрозрачной поверхности нужно использовать ранее нарисованный цвет. Это значит, что текущая грань должна рисоваться позже более далеких граней.

$C_{result} = (1-A)*C_{buffer} + A*C_{pixel}$

Для реализации полупрозрачности и других эффектов в системах компьютерной графики обычно используют технологию альфа-блендинга, которая основана на задании функции:

Color = TexelColor * SourceBlend + CurrentPixelColor * DestBlend
где:

Color – результирующее значение

TexelColor – рассчитанное значение

CurrentPixelColor – значение, хранящееся во вторичном буфере
SourceBlend DestBlend – значение, которое может быть:

D3DBLEND_ZERO
Blend factor is (0, 0, 0, 0).

D3DBLEND_ONE
Blend factor is (1, 1, 1, 1).

D3DBLEND_SRCCOLOR
Blend factor is (R_s, G_s, B_s, A_s).

D3DBLEND_INVSRCCOLOR
Blend factor is (1 - R_s, 1 - G_s, 1 - B_s, 1 - A_s).

Так для организации полупрозрачности применяют D3DBLEND_SRCALPHA и D3DBLEND_INVSRCALPHA соответственно, а для отрисовки луны на небе D3DBLEND_ONE, поскольку свет от луны просто добавляется к свету, который идет от неба.

2.3 Текстура

Текстура, текстурные координаты.

Как правило, все поверхности, которые можно моделировать в трехмерной графике, имеют некоторый рисунок. Например, изделие из древесины, поверхность луга, кожа человека.

Текстура представляется, как правило, прямоугольным изображением. Для того, чтобы «намазать» изображение картинки на поверхность желаемым образом, в каждой вершине проставляют текстурные координаты – пара чисел, которые задают точку на изображении, которой эта вершина соответствует.

Фильтрация. Методы расчёта текстурных координат

Поскольку при растеризации может возникнуть ситуация, когда одному элементу изображения текстуры (текселу) соответствует большая область на экране, то чтобы избежать эффекта «мозаики» конкретное значение берут из четырех соседних текселов, взвешивая их сообразно расстоянию. В DirectX для каждого направления в текстуре можно указать, брать ли тексел

просто выборкой ближайшего или нужно его интерполировать тем или иным способом.

Иногда бывают ситуации наоборот, когда большая текстура отображается в небольшое количество пикселей экрана. В этом случае для текстур вычисляют набор текстурных уровней, каждый из которых в два раза меньше по разрешению, и используют тот уровень который адекватен масштабу отображения (мэпирования). Либо используют интерполяцию между соседними уровнями (трилинейная интерполяция).

Мульти – текстурирование, Bump, Тени, Отражения

На моделируемую поверхность можно «натягивать» не одно изображение, а несколько, с применением тех или иных операций над их текселами. Например, на кирпичную стену можно наложить сложное распределение освещенности. Это может дать реалистический эффект и сэкономить текстурную память.

Зачастую поверхности бывают не ровными, причем неровности небольшие и с точки зрения расположения участка поверхности можно считать поверхность плоской, а вот степень освещенности из-за небольших неровностях может существенно отличаться. Для решения таких задач используют текстурирование нормалей, которые используют для освещенности.

В моделях освещенности, которые основаны на заливке спроецированного на экран примитива (в отличие от систем с трассировкой лучей), тени организовать существенно сложнее. Один из методов – построить изображение сцены со стороны источника и использовать z буффер результата как еще одну текстуру на объекты с перемножением текселов. Затененные части будут темными.

Еще один эффект, где важно уметь на одной поверхности совместить несколько текстур, это отражение. Моделей построения отражений несколько, но все они сводятся к тому, что строится еще одно изображение сцены, а выборка происходит сообразно нормали отражающей поверхности.

2.4 Анимация и эффекты

Анимация предметов и камеры

Для того, чтобы от кадра к кадру перемещать предмет, достаточно менять матрицу объекта. Для перемещения камеры – матрицу камеры. Несложно также менять расположение части объекта (вращение башни танка), для этого можно организовать иерархию матриц и в качестве объектной матрицы для башни выставлять перемноженные матрицы танка и башни.

Гораздо более сложной задачей является анимация живых существ.

Веса вершин, Skeleton

Одним из популярных способов анимации живых существ является привязка вершин с теми или иными весовыми коэффициентами к сразу нескольким матрицам, которые в свою очередь могут соответствовать тем или иным костям «скелета» живого существа

Lipsync

Еще одной из характерных задач компьютерной графики является синхронизация движения губ и воспроизводимой речи. Решение делиться на два этапа:

1 Распознавание характерных фонем с привязкой к таймлайну. Это распознавание звука возможно с использованием текста. Так как, например, в играх тексты реплик известны заранее. Эта задача решается заранее, а не в процессе использования приложения. Результатом является анимация матрицы для всех лицевых «костей». Кости в данном случае виртуальные и соответствуют мышцам: левая бровь, правый угол рта и т.д.
2 В процессе выполнения приложения липсинк становиться обычной скелетной анимацией

Эффекты. Системы частиц. Огонь, Дымы. Пары.

При симуляции реального мира много важных явлений не описываются устойчивой поверхностью или даже поверхностью

вообще. Например: огонь, дымы, пары, брызги. Еще на заре компьютерной графики был разработан метод под названием «система частиц» (particle system). Его суть заключается в том, что отображается большое количество мелких объектов с характерным поведением. Т.е много небольших граней двигаются по схожим, но разным траекториям и при этом меняют свой цвет и прозрачность.

С повышением производительности современных видеокарт варианты алгоритма системы частиц могут быть применены и для задач реального времени. Задавая характерное поведение частиц и их свойств можно получать разные эффекты – те же дымы, брызги, пар, снег, дождь.

Трава, деревья.

Растительный мир также важен для построения реалистичного изображения. В настоящее время наибольших успехов в анимации травы удается достичь при использовании текстурных шейдеров – небольших подпрограмм, исполняющихся внутри видеопроцессора.

Создание красивых, сложных деревьев можно эффективно реализовать с помощью рекурсивных алгоритмов, порождающих ветки на концах веток с учетом силы тяжести и с добавлением случайной величины в размер, толщину и цвет порождаемой ветки (или листа). Деревья, видимые лишь с большого расстояния, традиционно формируются из трех-четырех прямоугольных граней с полупрозрачными текстурами.

Ландшафт, Водная поверхность.

Реализация ландшафта имеет свои сложности. Так, если мы стоим на земле, то можем видеть до нескольких километров вдаль или даже дальше, если ландшафт имеет дальние особенности (далекие горы). Детальность ландшафта вокруг нас должна быть довольно высокой. Если весь ландшафт отображать с одинаковой подробностью, то количество примитивов будет огромным даже для современных видеокарт. Существует большое количество алгоритмов, которые позволяют отображать ближние участки ландшафта подробнее дальних. Основные проблемы, связанные с этим, вызваны возможностью наблюдателя перемещаться, тем

самым заставляя менять триангуляцию поверхности «на ходу», не допуская разрывов, скачков и других видимых проблем.

Вода чем-то схожа с ландшафтом. У нее есть то, что делает задачу проще, например, волны сравнительно одной высоты и в дали ее легче сводить к плоскости. Так же есть и очевидная сложность – ее динамика. Реализация воды разбивается, как правило, на два этапа:

1 Расчет карты высот или ее аналогов (например, на основе спектра и обратного преобразования Фурье);
2 Триангуляция. Неплохие результаты дает project grid, но он довольно старый и есть более сложные алгоритмы.

3 Самостоятельный практикум

Для решения предлагаются следующие задачи:

1. Статический уровень с bump/stencil shadows а-ля Doom3. FPS-like камера с collision detection.
2. Патиклы. Статическое окружение, в которое эмитируются патиклы с различными параметрами. Столкновения с геометрией, spawn on collision. Огонь, дым, взрывы. Пост-эффекты.
3. Софтверный рендер. Текстурирование, перспективная коррекция, вертексное освещение, смешивание текстур, альфа-блендинг.
4. Визуализация объемных текстур. Возможность динамики (примеры - game of life, решение разностных диффуров типа Навье-Стокса).
5. Прототип трехмерного интерфейса. Форматы описания интерфейса, события, трехмерные эффекты (возможности – патиклы, текстурирование, тени, пост-эффекты).
6. Игра «Бильярд». От шаров тень, шары имеют гладкий силуэт. Регулируется сила и направление удара. Биллиардный стол находится в некотором environment.
7. Модель солнечной системы. Солнце должно иметь корону. Звезды на «бесконечном» расстоянии. У планет должна быть атмосфера. По космосу летает космический кораблик управляемый игроком. Анимируемый выхлоп корабля.
8. Террэйн. Наблюдатель должен «летать» над террэйном. Террэйн должен быть текстурирован несколькими текстурами. Террэйн должен обладать уровнями детализации. Должно быть небо. На террэйне присутствуют озера, в которых отражается небо.
9. Лабиринт. Генерация «случайных» лабиринтов. Наблюдатель сталкивается со стенами. Пол отражает стены и потолок.
10. Море. Небо. Море анимируется, отражает небо. По морю плавают кораблики. Кораблик должен оставлять пенный след. Кораблики могут обстреливать друг друга из пушек. Взрывы на воде. Взрыв, когда ядро попадает в кораблик. Тонущие кораблики. Одним корабликом управляет игрок, а другим ai.

11. Генератор деревьев. Дерево и листья шевелятся на ветру. Дерево растет на лугу, на котором растет трава, которая тоже анимируется от ветра.
12. Игра «Cannons». На террэйне из п.3 расставляются пушки, которые по очереди обстреливают друг друга. Анимация взрывов. На террэйне остаются следы от взрывов.
13. Игра «догонялки». По лабиринту из п.4 бегают монстры и пытаются догнать игрока, игрок убегает. «2.5» View на лабиринт.
14. Гонки на машинках по террэйну из п.3. Ездят машинки. Побеждает та машинка, которая придет первой из точки 1 в точку 2. На террэйне растут кустики. Машинки оставляет следы на террэйне. Одной из машинок управляет игрок, остальными ai.