**Содержание**

[Товарищи](#h.gsb6wxgo4v0c)

[Общие вопросы компьютерной графики](#h.grot2364hjha)

[1. Понятие компьютерной графики. Связь с другими науками](#h.dmfdv3kke0sg)

[2. Физические аспекты восприятия](#h.xmn32u5olhdg)

[3. Основные цветовые модели](#h.18aftgczotyy)

[4. Векторная графика](#h.b9lk5q85o7f3)

[Хранение и обработка растровых изображений](#h.wifw8gcy8v)

[5. Растровые изображения и их характеристики](#h.8n2o4qmofz7y)

[6. Основные подходы к кодированию цвета и хранению растровых изображений](#h.w9k96grq323)

[7. Сжатие изображений на основе ДКП (алгоритм JPEG)](#h.jfak2aw4aadg)

[8. Фрактальное сжатие изображений. Алгоритм Малла](#h.dz8u3877va0q)

[9. Сжатие изображений без потерь (алгоритм JPEG-LS)](#h.eiezsthu6072)

[10. Формат цифровой фотографии (RAW)](#h.1fzyykaomd4g)

[11. Подавление шумов изображений](#h.l764d1othp4k)

[12. Интерполяция растровых изображений (масштабирование, поворот, изгиб)](#h.bd5fsx6kzoyl)

[13. Бинаризация изображений. Метод Оцу](#h.vf78x97pz53z)

[14. Алгоритм Флойда-Стейнберга](#h.yw7hybgqu9ig)

[15. Метод упорядоченных возбуждений](#h.lvg3k01s7vjj)

[16. Имитация цветов за счёт увеличения разрешения растра (маскирование)](#h.7ofbnya2ofvg)

[17. Равномерное выравнивание гистограммы](#h.c9wl48gcwsej)

[18. Неравномерное выравнивание гистограммы. Тональная и цветовая коррекция. Работа со светами и тенями](#h.fvc87orvwar8)

[19. Работа с резкостью изображений. Метод нерезкой маски](#h.a0ww6qulbyr0)

[20. Выделение границ на основе дифференциальных масок](#h.jdsik6iwexcj)

[21. Детектор краёв Кенни](#h.i1h3tyfo5709)

[22. Векторизация растровых изображений. Метод Хафа](#h.tlfcmd73azmx)

[23. Определитель прямых Барнса](#h.1jmsumxf0x2h)

[24. Методы сегментации изображений. Модели описания сегментов](#h.bnhn6hcjsgya)

[Двумерная растровая графика](#h.d5hly5gtzlq9)

[25. Алгоритм Брезенхема для рисования отрезков](#h.xsorom7cmaec)

[26. Алгоритм Брезенхема для рисования окружностей](#h.xjnkv66qmrnx)

[27. Кривые Безье. Алгоритм де Костельжо](#h.hu41oda8x5ii)

[28. Алгоритмы рисования закрашенного многоугольника](#h.psn6jfb5nq0i)

[29. Алгоритмы заполнения произвольных областей](#h.uu9exh33iddu)

[30. Рисование стилизованных примитивов (толщина и начертание линий, текстуры)](#h.wccv925knld)

[31. Растеризация текста. Шрифты](#h.gds6m8a0b75b)

[32. Важность и постановка задачи двумерного отсечения](#h.5v8jkk8r4w42)

[33. Алгоритм Сазерленда-Коэна](#h.axnrb04sr5bt)

[34. Алгоритм разбиения средней точкой](#h.5hfh4tsdhuer)

[35. FastClip алгоритм (быстрого отсечения)](#h.g564l4hw0rzj)

[36. Алгоритм Лианга-Барски](#h.3l4vufwz9xgb)

[37. Алгоритм Кируса-Бека](#h.oxl0ogtxqov2)

[38. Алгоритм Сазерленда-Ходжмена](#h.qjwaq1hwf75)

[39. Фрактальная графика](#h.7fddifxlbnk5)

[Трёхмерная графика](#h.vvizwijzvcho)

[40. Координатный метод описания объектов. Обобщённые координаты. Матрицы преобразования координат](#h.ju3ji51ir8y2)

[41. Проецирование. Виды координат (модельные, видовые, проекционные, экранные)](#h.fvg98x8b8eh4)

[42. Аналитическая модель описания поверхностей](#h.j53ejm39aab7)

[43. Векторная модель описания поверхностей](#h.hr8q8qegguza)

[44. Воксельная модель описания поверхностей](#h.8srq1oireb5d)

[45. Классификация способов визуализации (рендеринга) трёхмерных сцен](#h.jewtelck2edy)

[46. Алгоритм Робертса для многогранников](#h.uf55ugr5e2u5)

[47. Алгоритм плавающего горизонта для отрисовки поверхностей](#h.dxa1wsir0e0j)

[48. Алгоритм с сортировкой по глубине](#h.2u6t3mlo7gv6)

[49. Алгоритм Z-буфера и построчного сканирования](#h.xrvraws0qe2w)

[50. Алгоритм Варнока](#h.qx3rlhekv04o)

[51. Алгоритм Вейлера-Айзертона](#h.2qunkycmni2n)

[52. Модели отражения света и их использование при закраске](#h.ysb8htjr36d7)

[53. Методы закраски: монотонная закраска, метод Гуро и метод Фонга. Имитация микрорельефа (карты нормалей)](#h.brvmtrlattn2)

[54. Моделирование теней](#h.bpcemouv6et6)

[55. Метод трассировки лучей](#h.xjefc0p4bpcp)

[Трудоёмкая операция нахождения ближайшей грани.](#h.li2nj7lo2rd1)

[Возможная оптимизация метода:](#h.8p15n9ldad4a)

[Современные технологии компьютерной графики](#h.j11t8zyh2q4b)

[56. Основные идеи OpenGL и DirectX. Расширения OpenGL](#h.a5loz9oe6tvq)

[57. Аппаратная архитектура графических ускорителей. Графический конвейер](#h.o14yaegxwinv)

[58. Особенности скоростной обработки данных в графических ускорителях](#h.j136myyss648)

[59. Шейдеры](#h.joprnirl388l)

[Нефотореалистичная компьютерная графика](#h.4e6oyd7kgwo)

[60. Цели и задачи нефотореалистичной компьютерной графики. Артефакты и их классификация](#h.y29a5oic9ogw)

[61. Методы маскирования изображений](#h.7acvddt0zczc)

[62. Маскирование образами (текст, точка, линия)](#h.7z80asbezi68)

[63. Построение мозаик](#h.tyn5ors2q22c)

[64. Построение штриховых иллюстраций и текстур](#h.rjtfhu8usj6c)

[65. Рисование реалистичных отрезков](#h.l22ftfmx6abn)

[[ ХХХ ] 66. Имитация рисования кистью. Клетчатые автоматы](#h.quq6u5c0dv3u)

[[ ХХХ ] 67. Имитация рисования карандашом](#h.n0zydrp12rcc)

[[ ХХХ ] 68. Имитация гравировки](#h.anb4matwf3jm)

[69. Использование G-буферов. Примеры](#h.nkeibsr5cz07)

[[ ХХХ ] 70. Классификация рёбер и модифицированный алгоритм Робертса.](#h.o9v1cwgudm0e)

[[ ХХХ ] 71. Моделирование освещения в НФР.](#h.rwnsomopxk6f)

[[ ХХХ ] 72. Искажение пространства в НФР.](#h.98zw2jcgidgf)

[[ ХХХ ] 73. Искажение размеров объектов.](#h.a9y1vfdwh1i4)

**Легенда:**

* [ ХХХ ] — вопрос не был рассмотрен на лекциях и исключён из списка экзаменационных

# Товарищи

|  |  |
| --- | --- |
| **Товарищ** | **Задача** |
| Байер | фильтрация цветов |
| Барнс | определитель прямых |
| Безье | его кривые |
| Брезенхем | растеризация отрезков и окружностей |
| Варнок | отсечение многоугольников прямоугольным окном |
| Вейлер / Айзертон | отсечение многоугольников прямоугольным окном |
| Ву | растеризация отрезка |
| Гуро | немонотонная закраска |
| Кенни | детектор краёв |
| Кирус-Бек | отсечение отрезка произвольным многоугольником (векторная форма) |
| Костельжо | построение кривой по ломаной |
| Кох | его фрактал |
| Лианг / Барски | отсечение отрезка прямоугольным окном (векторная форма) |
| Малл | Сжатие ВЧ и НЧ |
| Оцу | бинаризация |
| Превитт | дифференциальная маска для вычисления градиента |
| Робертс | отрисовка произвольных многоугольников  дифференциальная маска для вычисления градиента |
| Сазерленд / Коэн | отсечение отрезка прямоугольным окном |
| Сазерленд / Ходжмен | отсечение выпуклым многоугольником произвольного многоугольника |
| Собель | дифференциальная маска для вычисления градиента |
| Флойд / Стейнберг | бинаризация |
| Фонг | немонотонная закраска |
| Хаф | векторизация |

# Общие вопросы компьютерной графики

## 1. Понятие компьютерной графики. Связь с другими науками

**Компьютерная графика** — область программирования, появившаяся благодаря вычислительной технике; инженерная наука (появилась в 50-60-х в США, первая система автоматизированного проектирования LAD (General Motors) предназначалась для чертежей 1963 г.).

Может рассматриваться как:

* совокупность информационно-технических средств
* направление в искусстве

|  |  |
| --- | --- |
| **Преобразование** | **Область** |
| Изображение ⇢ изображение | Обработка изображений |
| Изображение ⇢ данные | Распознавание |
| Данные ⇢ изображение | *Компьютерная графика* |

**Графический формат** — это способ записи графической информации. Графический формат должен:

* содержать всю информацию об изображении;
* описывать широкий класс изображений.

Графическая информация должна иметь средства ввода-вывода.

Типы устройств:

* *растровые* (из точек) (например, мониторы);
* *векторные* (из примитивов).

**Растеризация (отрисовка)** — преобразование векторного изображения в растровое.

**Векторизация** — преобразование растрового изображения в векторное. При этом необходимо находить на изображении графические примитивы и распознавать образы.

При оценке алгоритмов компьютерной графики применяются как стандартные критерии качества:

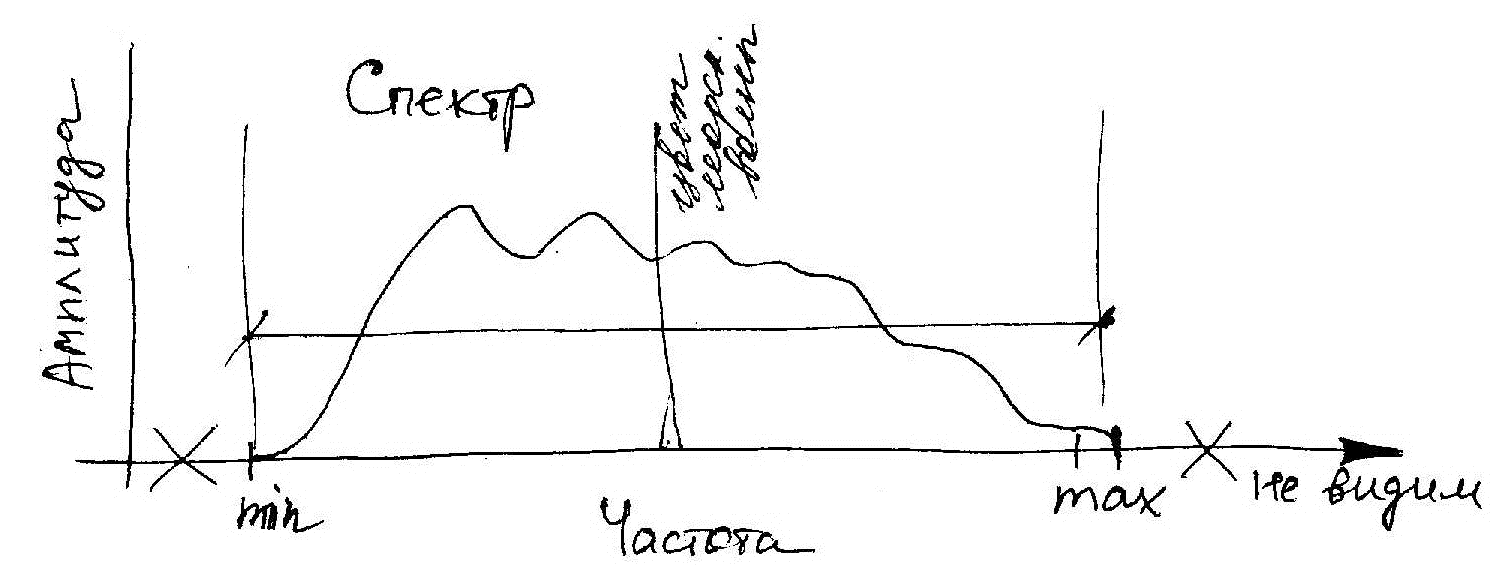
* скорость,
* правильность,
* расход памяти,
* читабельность,

так и специфические:

* реалистичность (сравнение с реальным видео (?)),
* нереалистичность (чертежи, начерченные графопостроителем или человеком).

## 2. Физические аспекты восприятия

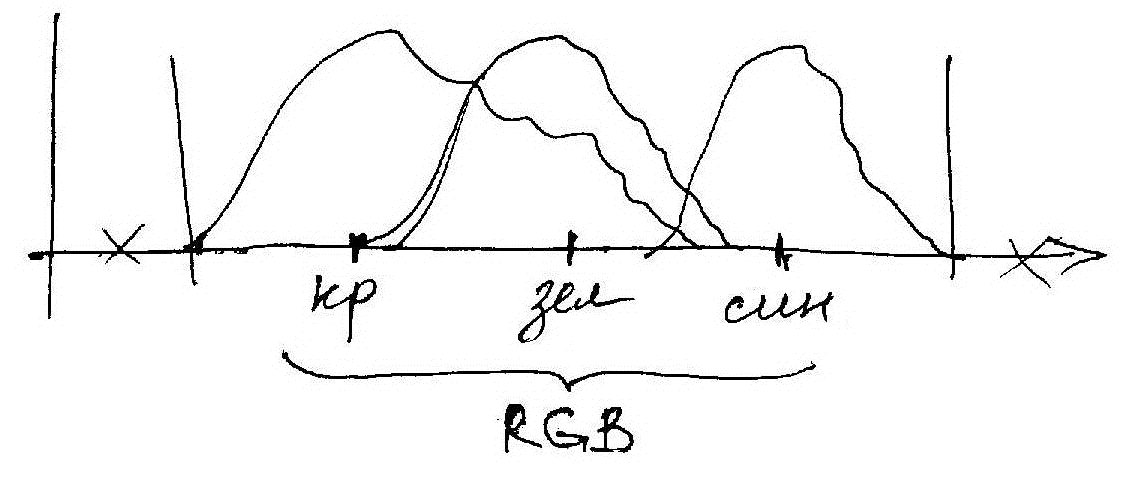
*Амплитудно-частотная характеристика* (АЧХ, спектр) человеческого зрения схематично представлена на графике ниже. Амплитуда волны определяет яркость света.



У человека существует три типа колбочек, реагирующих на различную частоту:

* Колбочки S-типа (*short*) чувствительны в фиолетово-синей части спектра,
* Колбочки M-типа (*medium*) — в зелено-желтой,
* Колбочки L-типа (*long*) — в жёлто-красной.

Наличие этих трех видов колбочек (и палочек, чувствительных в изумрудно-зелёной части спектра) даёт человеку цветное зрение:

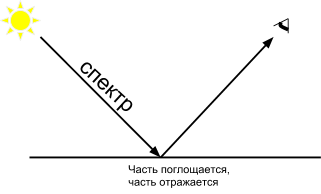


**Колориметрия** — наука о цвете и измерении цвета, возникшая в XIX веке. Позволяет численно выражать цвета с учётом яркости и оттенка.

**Свет** — электромагнитное излучение оптического диапазона.

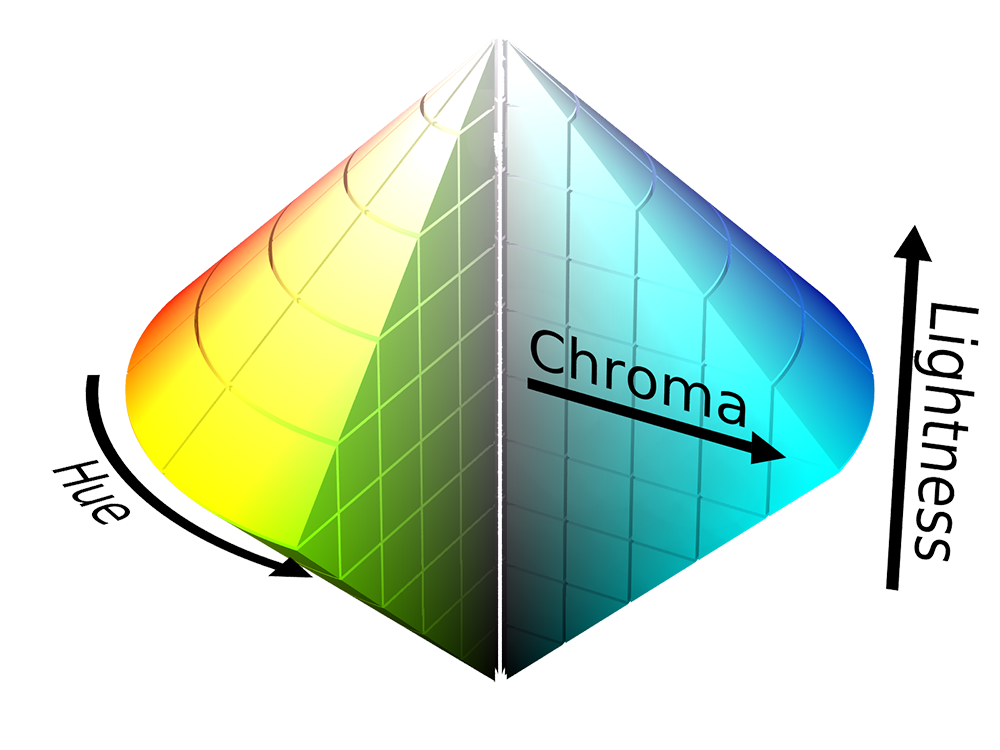
**Цвет** — качественная субъективная характеристика электромагнитного излучения оптического диапазона, определяемая на основании возникающего физиологического зрительного ощущения и зависящая от ряда физических, физиологических и психологических факторов. О как.

**Оттенок** — преобладание определенной комбинации волн.

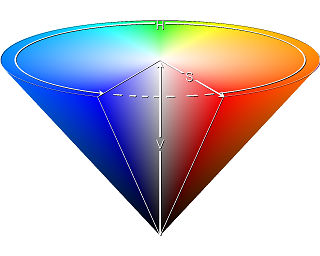


## 3. Основные цветовые модели

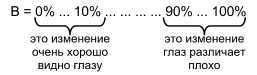
* **RGB** (*red, green, blue*) — стандартная цветовая модель. Покрывает не все цвета, видимые человеком (для этого пришлось бы делать отрицательную компоненту). Используется в устройствах отражающего типа: кинескопах, мониторах, цифровых фотоаппаратах. Является *аддитивной*, поскольку она “подсвечивает” чёрный цвет. Цвета получаются путём добавления к черному. Например, если цвет экрана, освещённого одним прожектором, обозначается в **RGB** как (r1, g1, b1), а цвет того же экрана, освещенного другим прожектором, — (r2, g2, b2), то при освещении двумя прожекторами экран будет иметь цвет (r1+r2, g1+g2, b1+b2).
  + Недостатки:
    - получаемый цвет зависит от устройства (*device dependent*).
  + Примеры:
    - R,G,B = 0 — чёрный
    - R,G,B = max — белый
    - R=G=B — оттенки серого
* **CMY** (*cyan, magenta, yellow — голубой, пурпурный, жёлтый*). Появилась в типографской печати. Используется в устройствах поглощающего типа. Является *субтрактивной*, поскольку она “затемняет” белый цвет. В CMY-модели описываются цвета на белом носителе, т. е. краситель, нанесенный на белую бумагу, вычитает часть спектра из падающего белого света. Например, на поверхность бумаги нанесли голубой (Cyan) краситель. Теперь красный свет, падающий на бумагу, полностью поглощается. Таким образом, голубой носитель вычитает красный свет из падающего белого. Модель показывает, сколько краски необходимо использовать для получения необходимого оттенка;
  + Взаимосвязь с RGB:
    - R = max – C
    - G = max – M
    - B = max – Y
  + Недостатки:
    - получаемый цвет зависит от устройства (*device dependent*).
    - высокие требования к точности печати (все краски должны попасть строго в одну точку)
    - зависимость цвета от температуры, влажности, типа бумаги
    - большой расход краски при печати чёрным цветом (бумага мокнет)
    - часть цветов являются критичными (человек хорошо различает оттенки серого)
  + Примеры:
    - C,M,Y = 0 — белый.
    - C,M,Y = max — чёрный (теоретически)
    - C=M=Y — оттенки серого
* **CMYK** (*cyan, magenta, yellow, key*). Отличается от CMY наличием специального канала, который можно использовать для чёрного цвета. Существуют различные алгоритмы вычисления компоненты K, однако всегда выполняются два условия:
  + K 0
  + K min(C, M, Y)
* **HSB / HSL** (hue, saturation, brightness/lightness — тон, насыщенность, яркость/светлота). Процесс добавления белого цвета к заданному можно представить как уменьшение насыщенности *S*, а процесс добавления чёрного цвета — как уменьшение яркости *V*. Основанию шестигранного конуса соответствует проекция RGB-куба вдоль его главной диагонали.
  + Трёхмерная проекция:
    - HSL:



* + - HSB:



* + Достоинства:
    - независимость от устройства вывода
    - простота определения цвета человеком[источник?]
  + Недостатки:
    - неравномерное распределение шкалы *brightness*:



* + Примеры для HSL:
    - L = 0 — чёрный цвет
    - L = 0.5 — цвет максимальной насыщенности
    - L = 1 — белый цвет
* **Lab** (*lightness, a, b*). Первая компонента определяет яркость, вторая и третья — цвет (от зелёного до пурпурного и от синего до жёлтого соответственно).
  + Достоинства:
    - яркость отделена от цвета
    - линейное изменение параметров с точки зрения человеческого восприятия
    - имеет полный цветовой охват
  + Недостатки
    - сложность вычисления модели

## 4. Векторная графика

Изображение задаётся как набор графических примитивов (точки, линии, отрезки, кривые Безье, текстовые надписи и т.п.). Несколько замкнутых кривых могут образовывать замкнутый контур, который можно залить.

Достоинства:

* Параметры каждого примитива могут изменяться независимо от других
* Очень легко выполнять повороты и масштабирование
* Высокая точность рисования даже при большом увеличении
* Экономичный расход памяти
* Возможность растеризации

Недостатки:

* Все устройства ввода изображений — растровые
* Векторизация изображений — очень сложная задача
* В векторной форме сложно построить реалистичное изображение
* Нет возможности применять эффекты и фильтры

Примеры:

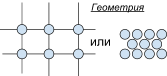
* PS / PDF
* TrueType-шрифты

# Хранение и обработка растровых изображений

## 5. Растровые изображения и их характеристики

Изображение разбивается на множество мелких, неделимых и не пересекающихся друг с другом фрагментов — *пикселов*. При этом каждая точка исходного изображения всегда попадает в некоторый из полученных фрагментов. Цвет каждого пикселя задаётся независимо от других.

Основные параметры растрового изображения:

* *Логический размер* (например, 1000х1000 пикселей)
* *Физический размер* (например, размеры монитора)
* *Разрешение* (плотность) — связывает логический и физический размеры
  + Знаете ли вы?
    - Человеческое зрение различает точки с точностью до одной минуты (1/60 градуса)
    - Для расстояния в 50 см это соответствует разрешению ~~181~~ 175 dpi
    - А для 30 см — ~~192~~ 291 dpi
* *Форма, шаг, размер пикселя*
* *Геометрия растра*
  + Форма пикселя (бывает прямоугольная (НЕ квадратная) и круглая)
  + Взаимное расположение пикселей
* *Глубина цвета* — определяется числом бит, отводящихся для хранения цвета пикселя
* *Наличие α-канала* (прозрачность)

Достоинства растровой графики:

* Универсальность
* Все устройства ввода и многие устройства вывода — растровые
* Скорость обработки изображения не зависит от самого изображения
* Возможность параллельной обработки

Недостатки:

* Большие объёмы памяти и ресурсоёмкость
* Невозможность точного масштабирования и поворота
* Невозможность вывод на плоттер (графопостроитель)

## 6. Основные подходы к кодированию цвета и хранению растровых изображений

Задача хранения растрового изображения сводится к задаче хранения положения пикселя и его цвета.

Способы хранения цвета:

* Непосредственное кодирование (например, RGB24 (24 бит, по байту на цвет), RGBA32)
  + Достоинства:
    - Надёжность
    - Независимость хранения
  + Недостатки:
    - большой расход памяти
* Использование палитры (набор цветов — палитра — хранится отдельно, а в пикселях только ссылки на цвета из палитры). Обычно содержит 2n цветов.
  + Виды палитр
    - аппаратные (неизменяемые, заданные на уровне устройства)
    - программные
  + Достоинства
    - экономичный расход памяти
  + Недостатки
    - ограниченный набор цветов: приемлемо для просмотра/хранения, но неудобно при редактировании. При изменении количества цветов надо перелопатить всю палитру.
  + Примеры:
    - CGA (четыре некоторых цвета)
    - EGA (16 цветов)
    - VGA (256 цветов)
    - SVGA (16 млн.)

Способы хранения изображения:

* **Без сжатия** (в виде двумерного массива пикселей)
  + Достоинства:
    - Простота
  + Недостатки:
    - Большой расход памяти
* **Сжатие без потерь**
  + Виды
    - С использованием специфики изображения
    - Без использования специфики
  + Алгоритмы
    - RLE (*Run-length encoding*) — хранит значение и число повторов

Например, пусть дана строка:

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW

Если мы применим простое кодирование длин серий, то получим следующее:

12W1B12W3B24W1B14W

Однако, в случае, если строка состоит из большого количества неповторяющихся символов, её объем может вырасти. Например, строка

ABCABCABCABCDDEFFFFFFFF

закодируется как

1A1B1C1A1B1C1A1B1C1A1B1C2D1E8F

Проблема решается достаточно просто. Положительные числа используют для записи количества повторяющихся одинаковых символов, а отрицательные — для записи количества неодинаковых:

-12**ABCABCABCABC**2D1E8F

*Если кодировать также повтор групп символов, то можно сжать строку ещё больше:*

*4[ABC]1C2D1E8F*

Ещё одна проблема связана с тем, что численные типы данных на компьютере всегда имеют некоторый предел. Предположим, мы используем signed char для записи длин серий. Тогда мы не можем записать серию длиннее 127 символов одной парой "длина-символ". Если подряд записано 256 символов “A”, их разделяют на минимальное количество групп:

127A127A2A

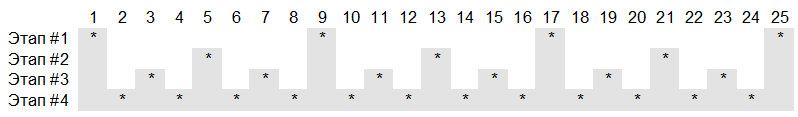
Запись на некотором языке программирования алгоритма RLE с учетом этих ограничений нетривиальна.

* + - *Методы энтропийной компрессии:*
      * LZW — zip-алгоритмы
      * Метод Хаффмана (сначала собирает статистику по изображению)

Каждой последовательности пикселей ставится в соответствие битовая последовательность, длина которой обратно пропорциональна частоте появления цвета/группы пикселей в изображении: чем чаще встречается элемент, тем короче его код, и наоборот. (Мы считываем последовательно символы входного потока и проверяем, есть ли в созданной нами таблице строк такая строка. Если строка есть, то мы считываем следующий символ, а если строки нет, то заносим в поток код для предыдущей найденной строки, заносим строку в таблицу и начинаем поиск снова. LZW реализован в форматах GIF и TIFF. )

* + Достоинства
    - Экономичный расход памяти
    - Сохранение точной информации
  + Недостатки
    - Сложный доступ к данным
    - При изменении изображения может не хватить памяти (?)
    - Невозможность регуляции степени сжатия
* **Сжатие с потерями** — происходят незначительные изменения, незаметные человеку. Все алгоритмы этого класса используют особенности человеческого зрения. Однако существует порог, после которого изображение заметно теряет качество.
  + Достоинства
    - Экономичное расходование памяти
    - Возможность регулировки степени сжатия
  + Недостатки
    - Сложный доступ к данным

Основные форматы

* **bmp**
  + Использует несколько способов хранения цвета
  + Поддерживает α-канал
  + Поддерживает сжатие без потерь
  + Поддерживает палитры
* **tiff**
  + БОльшая глубина цвета
  + Поддерживает несколько цветовых моделей
  + Поддерживает несколько методов цветовой компрессии (изначально формат поддерживал сжатие без потерь, впоследствии формат был дополнен для поддержки сжатия с потерями в формате JPEG)
  + Поддерживает теги
* **gif** (*graphics interchange format*)
  + Сжатие без потерь
  + Однобитный α-канал (без полупрозрачности)
  + Поддерживает только 256-цветную палитру
  + Поддерживает анимацию
  + Поддерживает чересстрочную развёртку — позволяет увидеть картинку до полной загрузки, которая проходит в четыре этапа:
    - 
  + ~~Можно создавать GIF-изображения только с помощью лицензионного ПО~~ Срок действия последнего патента на GIF истёк 11 августа 2006 года (с) ВП
* **png** (*portable network graphics*)
  + Сжатие без потерь
  + Поддерживает двумерную и 4-х-строчную развёртку
  + Поддерживает палитры
  + Поддерживает анимацию
  + Поддерживает теги
* **jpeg**
  + использует для сжатия дискретно-косинусное преобразование (ДКП)
    - JPEG-LS — сжатие без потерь
    - JPEG-2000 — сжатие на основе вейвлетов
    - Сжатие с потерями
  + Хорошо подходит для фото, хуже для рисунков
  + Может хранить служебную информацию (тэги, информацию о фотоаппарате, условиях съёмки и т.п.)

## 7. Сжатие изображений на основе ДКП (алгоритм JPEG)

Человек легче различает яркость, чем цвет. Эту особенность использует цветовая модель YCbCr (*Y — яркость, Cb — хроматический синий, Cr — хроматический красный*): при переводе в неё из RGB информация о яркости сохраняется, а о цвете — частично теряется. YCbCr применяется в телевидении.

**ДКП** — дискретно-косинусное преобразование, используется для сжатия изображений.

Алгоритм:

1. Преобразование RGB → YCbCr: YCbCr = Т \* RGB; Т — матрица коэффициентов 3х3.
2. Изображение разбивается на фрагметы 8х8 px, которые обрабатываются отдельно. Создается матрица DKL=cos(K(L+0.5)π/N), 0≤K,L<N, N=8. Каждый фрагмент раскладывается в линейную комбинацию таких матриц Верхний левый коэффициент — самый большой, потому оказывает наибольшее влияние на результат. AK,L — подбираем.
3. Квантование. Матрица квантования QK,L постоянна и задается на уровне формата. Элементы матрицы коэффициентов А целочисленно делятся на соответствующие значения Q, многие из них обнуляются, отсюда потери. На этом этапе возможна регулировка степени сжатия (например, если в качестве матрицы QK,L взять единичную, потерь не будет, потому что при целочисленном делении любого числа на 1 потери невозможны, в отличие от 9/10).
4. Полученная матрица коэффициентов AK,L (после деления она записана вдоль неглавной диагонали “змейкой”) сжимается методом энтропийной компрессии без потерь.

При декомпрессии действия выполняются в обратном порядке.

## 8. Фрактальное сжатие изображений. Алгоритм Малла

**Алгоритм Малла** — разновидность дискретного вейвлет-преобразования. Применяется для предварительной обработки сигнала с целью уменьшения его информационной избыточности:

1. Входной набор данных разбивается на пары
2. Для каждой пары находится разность и сумма
   * Разность — ВЧ-фильтр (фиксирует мелкие детали)
   * Сумма — НЧ-фильтр (фиксирует крупные изменения)
3. Алгоритм применяется рекурсивно к массиву сумм, а массив разностей записывается в результат.

После выполнения преобразования полученный сигнал хорошо сжимается методом энтропийной компрессии. Преобразование двумерного сигнала (например, изображения) производится аналогично.

**Фрактальное сжатие**. В промышленности не применяется. ~~Всё.~~

В пространстве изображений {x} вводится метрика ρ(x, y) схожести; например, евклидова:

и некоторое сжимающее отображение

По теореме Банаха, отображение, применённое к исходному изображению, даст то же самое изображение:

A(x) = x,

поэтому после многократного применения отображения *A* *любое* изображение “сойдётся” к исходному. В процессе фрактального сжатия для поданного на вход изображения строится сжимающее отображение. Оно кодируется и записывается в файл. При открытии отображение загружается из этого файла и применяется *n* раз к чёрной картинке.

Основная задача — как по множеству X определить отображение A. Этого пока никто не знает.

## 9. Сжатие изображений без потерь (алгоритм JPEG-LS)

Алгоритм пытается предсказать цвет пикселя по цветам его соседей, используя один из четырёх способов:

* C[x,y]=C[x-1,y] // сосед слева
* C[x,y]=C[x,y-1] // сосед сверху
* C[x,y]=C[x-1,y-1] // сосед слева-сверху
* C[x,y]=0.5 \* (C[x-1,y]+C[x,y-1]) // среднее между соседями слева и сверху

Вместо цвета пикселя храним номер метода предсказания (выбираем наиболее точный) и величину ошибки. Если на изображении мало деталей, то ошибки также будут малы. После этого можно:

* округлить погрешности — получим сжатие с потерями
* сжать полученную последовательности *as is* — тогда потерь не будет

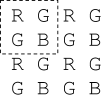
## 10. Формат цифровой фотографии (RAW)

Светочувствительная матрица состоит из датчиков, которые определяют, сколько света попало в точку. Основные термины:

* *диафрагма* — ширина открытия датчика
* *выдержка* — время открытия датчика
* *экспозиционная пара* — совокупность диафрагмы и выдержки
* *недоэкспозиция* — недостаток света
* *переэкспозиция* — избыток света

Датчики как таковые определяют только яркость. Для определения цвета перед ними ставятся фильтры: один пропускает только зелёный свет, другой — красный и т.д.

Расположение датчиков (матрица Байера):

Человек лучше воспринимает зелёный, поэтому таких датчиков больше. Формат RAW хранит показания датчиков.

Цвет белых тел зависит от того, какой источник их освещает. Для правильной передачи цветов необходимо задавать цвет источника освещения, а то будут синие лица. ~~и красные носы~~

*Баланс белого* зависит от источника освещения, задается через *цветовую температуру* — это температура абсолютно чёрного тела, которое излучает такое освещение. Если нагреть абсолютно чёрное тело, то оно будет светиться, а если нет - то поглощать все лучи. От температуры зависит, какого цвета будет излучение. Примечательно, что одним из наиболее близких к абсолютно чёрному телу объектов является Солнце.

Преобразование RAW в растровый формат:

1. *Дебайеризация*. Показания датчиков каждого цвета суммируются Две схемы:
   1. Используются четыре верхних левых датчика, в результате — пиксель между ними.
   2. Используются все 16 датчиков, в результате — девять пикселей, расположенных между ними.

На этом шаге учитывается экспокоррекция и баланс белого.

1. Учёт *логарифмичности ощущения* интенсивности света (~~например, если интенсивность возрастает вчетверо, то человек ощущает это как удвоение~~ — неверно!): различие тёмных тонов ощущается лучше, чем светлых. Т.к. в RGB эта логарифмичность не заложена, 128≈200. 128 — средний цвет на выдачу монитора, 200 — средний цвет при фотографировании
2. Повышение *цветонасыщенности* и резкости (RGB→HSB и S+=10%)
3. Сохранение в \*.jpg

Достоинства RAW:

* Большой цветовой охват и большой динамический диапазон (разница между самой темной и самой светлой точками; в RGB только 256)
* Большая точность передачи цветов
* Возможность корректировки баланса белого
* Высокая резкость изображения
* Возможность обработки в RAW-конвертере
* Возможность многократной обработки

Недостатки RAW:

* Отсутствие единого стандарта (форматы RAW разных производителей несовместимы)
* Не является итоговым изображением, требует переработки
* Требует много памяти

## 11. Подавление шумов изображений

Шумы возникают из-за сбоя датчиков, отклонения цвета пикселя. Фильтрация позволяет подавить шумы, но при этом исчезают и детали.

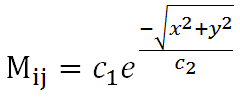
Виды шумов:

* яркостные
* цветовые

Параметры шумов:

* *Амплитуда* — величина отклонения сигнала от верного значения.
* *Частота* — показывает, как часто появляется шум в сигнале.

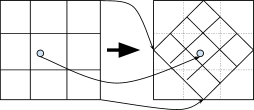
Методы подавления шумов:

* **Сглаживание** (фильтрация по Гауссу):
  +  - матрица фильтрации размера (2n +1)\*(2n +1) показывает вес пикселя рядом с x ,y

g — исходная интенсивность

* **Медианная шумовая фильтрация**. Цвета соседних пикселей помещаются в одномерный массив, сортируются и выбирается среднее (по положению в списке).
  + Достоинство: пиксель принимает интенсивность, которая реально присутствует в его окружении.
  + Недостаток: очень низкая скорость
* **Метод ДКП**. При сжатии вместе с деталями исчезают и шумы.
* **Метод усреднения**. Берём несколько изображений; один и тот же пиксель на всех изображениях будет соответствовать одному и тому же объекту, находим среднее.

## 12. Интерполяция растровых изображений (масштабирование, поворот, изгиб)

Интерполяция растровых изображений требуется при изменении геометрии растра:

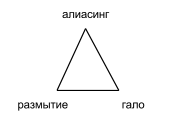
* масштабирование
* поворот, сгиб
* корректировка проблем съёмки (кривой фотик, кривая фотка, кривой горизонт, кривые руки и т.п.)

Интерполяция позволяет определить цвет пикселя по его окрестности.

Алгоритмы интерполяции:

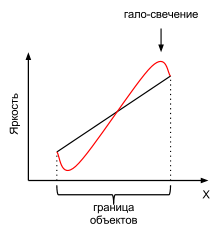
* **Неадаптивные**. Никак не учитывают границы (контуры) в исходном изображении. Способы определения цвета:

1. *Ближайший пиксель* (*nearest-neighbor*)
2. *Билинейный*: вычисляется среднее среди 4-х соседних пикселей (Не совсем так: создаем новое изображение с новым размером, перебираем все его точки. Для каждой точки нового изображения ищем “родительскую”. А от нее берем соседние. Компоненты цвета нового пикселя выбираются как линейная комбинация компонент соседей, умноженных на коэффициенты. Коэффициенты выбираются исходя из величины отклонения реального пикселя от точной величины (из-за того, что арифметика масштабирования не целочисленная))
3. *Бикубический*: берётся 16 соседних пикселей.
4. Использование *сплайнов*. По 16-ти точкам строится двумерный сплайн.

* **Адаптивные**. Алгоритмы рассматривают окрестность каждой точки и пытаются определить, является ли она граничной. В этом случае веса корректируются. Эти алгоритмы не являются универсальными: для каждой интерполяции (поворот, масштаб, изгиб) они специфичны.

Проблемы интерполяции:

1. *Алиасинг* (ступенчатость). Борьба с алиасингом — сглаживание.
2. *Размытие*
3. *Гало-свечение*, возникающее при повышении резкости
4. Каждая интерполяция вносит погрешности, поэтому несколько трансформаций лучше делать сразу.



## 13. Бинаризация изображений. Метод Оцу

**Бинарное изображение**: на каждый пиксель отводится один бит (1=белый, 0=чёрный)

**Чёрно-белое изображение**: на каждый пиксель отводится несколько бит, которые показывают уровень яркости.

Бинарные изображения используются в некоторых устройствах ввода-вывода, чёрно-белые — при задании формы объектов (ч/б прозрачность).

**Задача бинаризации** — перевод чёрно-белого изображения в бинарное.

Методы:

1. *Пороговая бинаризация*

Входное изображение задается интенсивностью **I**(x,y). Выходное изображение определяется как O(x,y) = (**I**(x,y) < A ? 0 : 1). Значение порога А можно менять, чтобы получить приемлемую картинку: при повышении порога лучше просматриваются яркие детали, при понижении — тёмные.

1. *Метод Оцу*.

|  |  |
| --- | --- |
|  | HxW = количество пикселей. Гистограмма яркости показывает, сколько каких пикселов — тёмных и светлых. Значение порога t автоматически выбирается так, чтобы взвешенная сумма внутригрупповых дисперсий была минимальна. |

Более формально:

i=0..max — яркость

c(i) — количество пикселей с яркостью i

— вероятность появления пикселя с яркостью i

— вероятность появления пикселя темнее t

— вероятность появления пикселя светлее t

; — cредняя яркость пикселей в группах.

; — дисперсия (величина разброса среднего значения).

Тогда

t:

1. *Метод имени меня.* Каждый пиксель входного изображения случайным образом заменяется на чёрный или белый, причём вероятность выбора белого цвета зависит от яркости исходного пикселя:



## 14. Алгоритм Флойда-Стейнберга

При использовании метода пороговой бинаризации возможны большие ошибки (отклонения от исходного изображения). Например, область, где яркость всех пикселей чуть ниже порога, станет полностью чёрной. Однако если в такой области чередовать чёрные и белые пиксели, то визуально область будет казаться серой. Меняя соотношение ч/б пикселей, можно манипулировать градациями серого.

При пороговой бинаризации цвет пикселя определяется как

O(x,y) = (**I**(x,y)<A ? 0 : Imax)

При этом возникает ошибка

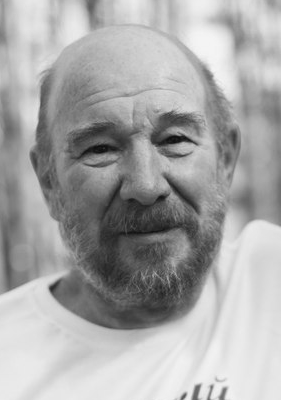
Е(x,y) = I(x,y) – O(x,y)

E(x,y) = I(x,y)<A ? I(x,y) : I(x,y) – Imax

Ошибка Е показывает, насколько цвет стал светлее (E<0) или темнее (E>0), чем в исходном изображении. **Алгоритм Флойда-Стейнберга** учитывает величину ошибки, добавляя её к исходной интенсивности соседних пикселей. Таким образом, ошибка распространяется, и “соседи” пытаются её компенсировать.

|  |  |
| --- | --- |
| for (x = xmin; x < xmax-1; x++)  for (y = ymin; y < ymax-1; y++)  {  if (I[x,y] < A)  { Out[x,y]=0; E=I[x,y]; }  else  { Out[x,y]=1; E=I[x,y]-Imax; }  I[x+1,y]+=E\*3/8;  I[x,y+1]+=E\*3/8;  I[x+1,y+1]+=E/4;  } |  |

Пример работы алгоритма (слева — исходное изображение, справа — после обработки):

## 15. Метод упорядоченных возбуждений

В отличие от пороговой бинаризации, в данном методе порог не фиксирован, а выбирается из матрицы пороговых значений (возбуждений) размером 2Lx2L, где L определяется исходя из размеров изображения. Изображение разбивается на квадраты такого же размера, яркость каждого пикселя каждого квадрата сравнивается с соответствующим ему значением в матрице D. Если она меньше, чем в матрице, то пиксель делается черным, иначе — белым.

Требования к матрице пороговых значений:

* Все элементы различны
* Значения должны быть равномерно распределены — это необходимо для равномерного распределения ч/б пикселей в выходном изображении.
* Размер матрицы D кратен двум.

Матрица строится по формуле:

где M — максимальная яркость. В двумерном случае:



в *n*-мерном:



где UN — матрица размерности N, состоящая из единиц.

## 16. Имитация цветов за счёт увеличения разрешения растра (маскирование)

|  |  |
| --- | --- |
|  | Сочетание ч/б пикселей выбирается исходя из цвета серого пикселя, каждому цвету ставится в соответствие своя комбинация. Существует (n2+1) градаций яркости. Отдельно взятая конфигурация похожа на серый цвет, но при рассмотрении группы может возникнуть *микроузор* — некий фоновый рисунок из-за совмещения узоров, которыми заменяли серые пиксели. Иногда нужно от него избавится (если хотим получить реалистичное изображение), иногда он нужен (см. НФР). |

## 17. Равномерное выравнивание гистограммы

… или “как вытягивают фотографии”. Основные задачи:

* Улучшение восприятия изображения
* Подготовка к выводу на устройство
* Предварительная подготовка изображения перед распознаванием (или просто для просмотра деталей)

При тональной коррекции изменяется яркость, но не цвета. Преобразования проводятся в моделях Lab, YCbCr. Каждый пиксель обрабатывается независимо от других.

|  |  |
| --- | --- |
|  | **Гистограмма** показывает для каждого значения яркости количество пикселей с такой яркостью.  **Тональный диапазон** — диапазон значений интенсивности, которые реально присутствуют в изображении; разница между самой яркой и самой тёмной точкой.  **Яркость изображения** — средняя яркость всех точек. Говорят, что изображение в тёмном (светлом) ключе, когда превалируют тени (света). |

**Контраст** — соотношение между тенями и светами в изображении. Выделяют *локальный* и *глобальный* контраст.

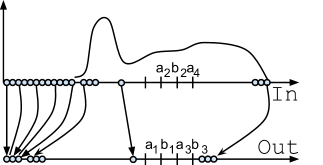
**Локальный контраст** — контраст на участке изображения. Чем больше контраст, тем лучше рассматриваются детали и текстура, однако избыточная контрастность может приводить к нереалистичности.

Задачи тональной коррекции:

1. Должен использоваться весь динамический диапазон (растяжение тонального диапазона).
2. Гистограмма должна быть выровнена.
3. Должна быть сохранена естественность изображения (противоречие с п.1 и 2).

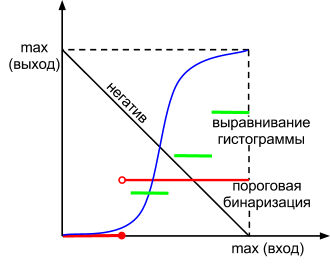
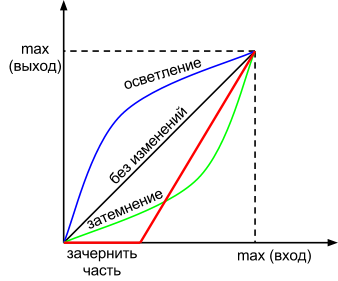
**Равномерное выравнивание гистограммы**

В выходном изображении должны присутствовать все возможные значения интенсивности, и каждое присутствующее значение должно встречаться примерно одинаковое число раз.

Каждому значению интенсивности входного изображения i соответствует количество пикселей Hi с такой интенсивностью. Этому i в выходном изображении будут соответствовать ΣHi пикселей. Неведомая формула: (в коей M, судя по всему, — динамический диапазон). Перебираем a и b, пока сумма не переплюнет значение WH/M, и получим a1 и b1.

**Выравнивание с использованием кривых**

Кривые позволяют перераспределить тональный диапазон, но не позволяют его расширить. Если просто горизонтальная линия, то будет заливка цветом (для ч/б изображений) или равномерное распределение яркости (для цветных). Угол наклона кривой показывает контрастность изображения, а высота кривой — яркость. Разность между max и min значением кривой определяется тональным диапазоном.



На втором рисунке синяя загогулина: убиваем контрастность в светлых и темных областях, но добавляем в средних.

**Постеризация** — отсутствие плавных переходов после обработки.

|  |  |
| --- | --- |
| Некоторые возможности кривых можно задавать тремя параметрами:  А - точка чёрного  В - точка белого  С - точка полутонов | Этот способ задания называется *Levels* (уровни).  Более сложный случай:  А1 — выход точки чёрного  В1 — выход точки белого |
|  |  |

## 18. Неравномерное выравнивание гистограммы. Тональная и цветовая коррекция. Работа со светами и тенями

Кривых теперь недостаточно. Корректируется только часть пикселей (например, хотим осветлить какую-то темную область). Входные параметры:

* *тональный диапазон (ширина)* в % — та часть диапазона, которую мы считаем тенями (светами)
* *радиус* в пикселах
* *степень влияния*: насколько будем осветлять тени / затемнять света.

Пиксель будет затемняться (осветляться) на указанную степень, если выполняются оба условия:

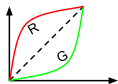
1. Его яркость попадает в заданную тональную ширину
2. В его окрестности заданного радиуса присутствует достаточное количество пикселей, которые тоже попадают в заданную тональную ширину.

**Цветовая коррекция**

C помощью кривых каждый компонент цвета (R,G,B) может обрабатываться независимо. Чтобы сохранить естественность цветов, кривые надо задавать согласованно. Существуют цвета, изменение которых наиболее сильно заметно человеку: цвет кожи и серый цвет. В RAW камера автоматически выполняет цветовую коррекцию.

*Пример*. На рисунке из серого изображения сделали изображение, в котором много красного и мало зеленого:

/\*Вариант: На рисунке представлен фильтр, повышающий уровень красного и понижающий уровень зелёного. \*/

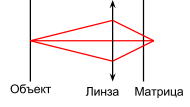
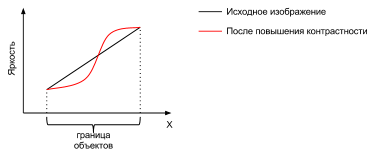


*Пример (Катя)*. Раскидываем изображение на слои (света в отдельный слой, тени в отдельный), производим коррекцию раздельно по слоям.

По-моему, имелось ввиду к текущему изображению добавить слой-маску, для которого провести коррекцию цвета и света, потому что не вижу я инструмента выделения тени и света как слой.

## 19. Работа с резкостью изображений. Метод нерезкой маски

Если лучи, идущие от одной точки объекта, не сходятся в одной точке светочувствительной матрицы камеры, то изображение получается нерезким. После того, как снимок сделан, физически исправить это уже нельзя, однако можно имитировать повышение резкости путём повышения контраста на границах объектов:

Для этого можно использовать **метод нерезкой маски**.

Идея:

* Если при размытии цвет пикселя изменился сильно, то он — граничный.

Параметры:

* *Радиус размытия* — определяет максимальную толщину границ.
* *Степень* повышения локального контраста.
* *Порог* — определяет минимальный перепад интенсивности, необходимый для того, чтобы точка определилась как граничная.

Алгоритм:

1. Создаются вспомогательные изображения:
   * *Размытое* — получается размытием исходного изображения по Гауссу.
   * *Контрастное* — получается повышением глобального контраста исходного изображения.
   * *Маска* — определяется как разность исходного изображения и размытого.
2. Цвет каждого пикселя выходного изображения берётся из:
   * контрастного изображения, если соответствующий пиксель маски больше некоторого порога (т.е. в процессе размытия пиксель изменился сильно)
   * исходного изображения — в противном случае.

**Замечание**. В исходном алгоритме после получения маски она бинаризуется с указанным порогом. Это может пригодиться, если нас интересуют промежуточные результаты работы алгоритма, а не только конечный.

## 20. Выделение границ на основе дифференциальных масок

*Источник:* [*http://habrahabr.ru/post/114452/*](http://habrahabr.ru/post/114452/)

Рассмотрим сначала одномерный случай. Пусть f(x) — дискретно заданная функция, определяющая яркость пикселя с целой координатой X. Определим дискретную производную как

Соответственно, вторая производная — производная первой производной — будет иметь вид:

В общем виде дискретную производную можно представить как сумму

где

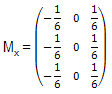
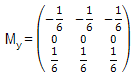
— матрица коэффициентов (весов):

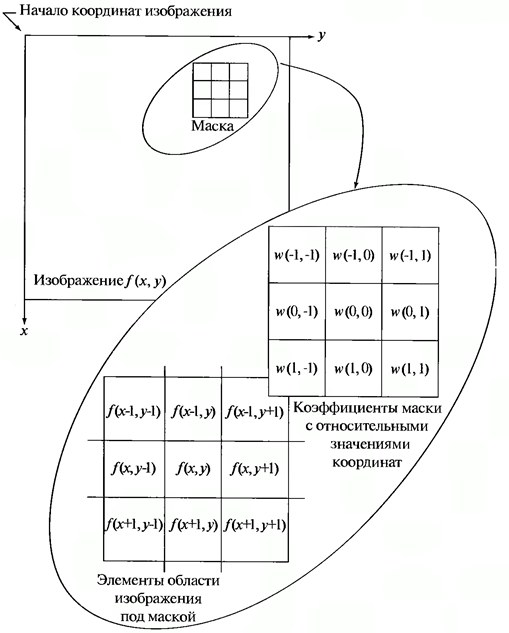
* + для :
  + для :

Для двумерного случая определим дискретную производную как средневзвешенное *одномерных* производных в трёх соседних точках по горизонтали или вертикали:

где

M — матрица весов. При она будет иметь вид:

* + для : 
  + для : 



Для использования целочисленной арифметики и ускорения вычислений указанные матрицы весов умножают на шесть (*маска Превитта*):



Кроме того, можно увеличить вес пикселей той же строки (столбца) для уменьшения эффекта сглаживания (*маска Собеля)*:



А ещё есть *маска Робертса* (соответствует повернутому на 45° градиенту):



~~Чтобы не разрывать читателю шаблон~~, её можно представить в виде матрицы 3х3, в которой первая строка и столбец — нулевые. В качестве центрального элемента можно использовать, например, левый верхний. Благодаря меньшему размеру работает быстрее, чем маски Собеля / Превитта.

Используя одну из этих масок, можно найти частные производные и в каждой точке изображения, а значит и вектор градиента — он показывает, насколько резко изменяется яркость в точке. По его модулю можно определить, является ли пиксель граничным или нет:

* в точках высокого локального контраста:
  + имеет большое значение;
  + меняет знак, если достигает своего экстремума
* в точках низкого локального контраста:
  + первая производная имеет малое значение (либо 0).

**Использование лог-фильтра**. Для каждого пикселя делаем два размытия: маленькое и большое, и берём разницу между этими значениями.

c-радиус сглаживания. Лог-фильтр для одномерного случая:

*При сильном увеличении влияния возможно возникновение эффекта гало.*

**Идея согласованных фильтров**

Маска — это окрестность пикселя, она может быть развернута в вектор чисел. Все возможные значения окрестных пикселов образуют векторное пространство. Его базисом является множество векторов, в каждом из которых только один ненулевой элемент. Аi — базизные векторы, xi - множители. Матрица яркостей вокруг пикселей В этом же линейном пространстве можно задать другой базис, элементы которого будут иметь более сложную структуру, например, наличие вертикальной / горизонтальной границы. Тогда если мы разложим по новому базису окрестность, то получим набор чисел, которые показывают степень похожести окрестности на соответствующий базисный элемент. Анализируя этот набор чисел, можно принимать решение о наличии границы. Элементы, по которым раскладываем — “согласованные фильтры”. Например, если взять маску Превитта Mx, то yi будут показывать оценку вертикальной границы.

## 21. Детектор краёв Кенни

*Источники:*

* [*http://en.wikipedia.org/wiki/Canny\_edge\_detector*](http://en.wikipedia.org/wiki/Canny_edge_detector)
* [*http://habrahabr.ru/post/114589/*](http://habrahabr.ru/post/114589/)

Главное достоинство метода — он позволяет обнаруживать даже широкие границы.

Входные параметры:

* *размах (радиус) сглаживания*
* *верхний порог интенсивности* — определяет минимальное значения градиента для начала детектирования границы
* *нижний порог интенсивности* — определяет минимальное значения градиента для продолжения детектирования границы

Алгоритм:

1. **Сглаживание по Гауссу** с заданным размахом. Границы приобретают гладкость, исключаются разрывы.
2. **Вычисление градиента** в каждой точке с помощью дифференциальных масок. Градиент показывает, в каком направлении яркость возрастает наиболее быстро.
3. **Немаксимальное подавление**. Для каждого пикселя рассматривается пара его соседей: один в направлении градиента, другой в обратном направлении. Если величина градиента среднего пикселя не является максимальной среди этих трёх, то она обнуляется.
4. **Прослеживание контуров**. Поиск начинается с пикселей, у которых значение градиента больше *верхнего* порога интенсивности. Рекурсивно рассматриваются соседние пиксели, у которых значение градиента больше *нижнего* порога интенсивности.

## 22. Векторизация растровых изображений. Метод Хафа

**Задача**: поиск примитивов в растровом изображении.

Все примитивы искомого типа (например, окружности, прямые, треугольники и т.п.) описываются уравнением:

*f(x, y, p) = 0*

где

*x, y* — координаты [граничного] пикселя,

*p* — набор параметров (вектор, обычно вещественный), с изменением которого получаются разные примитивы.

**Алгоритм**:

1. Строится ассоциативный массив А (“аккумулятор”), размерность которого равна количеству элементов в *p*. В начале массив полностью состоит из нулей (или просто пуст).
   * Например, для прямой *y = kx + b* нужно использовать двумерный массив, так как имеются два неизвестных параметра: *p1 = k* и *p2 = b*. Два измерения массива А соответствуют различным значениям параметров *k* и *b*.
2. Для каждой точки <и её соседей> алгоритм определяет, достаточен ли вес границы в этой точке. Если да, то вычисляются параметры всех возможных примитивов, проходящих через эту точку в заданном направлении (используем направление градиента), и увеличиваются соответствующие этим параметрам значения в ячейках аккумулятора.
3. Среди значений массива А находятся те, которые превосходят некоторый порог. Соответствующие им параметры *p* определяют искомые примитивы.

**Замечания** (Босик Босикович)

* С математической точки зрения, множество параметров *p* может быть бесконечным (например, через одну точку можно провести бесконечно много прямых), поэтому в практической реализации потребуется сделать это множество ограниченным и дискретным.
* Возможно, на третьем шаге перед поиском максимумов придётся сначала сгладить (“размыть”) массив *p*.

*Случай отрезков*

|  |  |
| --- | --- |
|  | Ищем отрезки, которые начинаются в (0,0). Для каждого пикселя изображения смотрим, если значение больше некоторого порога, то он потенциально лежит на отрезке, который заканчивается в (a,b). Увеличиваем все элементы массива А, которые соответствуют прямым, через которые может проходить данный пиксель. В результате те значения (a,b), для которых в матрице A находятся значения, большие некоторого порога, будут характеризовать самые вероятные примитивы. |
|  | Какое уравнение лучше использовать для задания прямой?   * y=kx+b — не позволяет описать вертикальную прямую. * ax+by+c=0 — тут три параметра, много. * xcosα + ysinα = d — два параметра, нормально. α-угол, d-расстояние. Зная градиент и α, можем найти d. |

Проблемы:

* Может быть очень большой массив А
* Эффективность зависит от того, каким образом задано уравнение примитива, т.е. сколько параметров в векторе р.
* Для одного (x,y) находим множество (a,b)

|  |  |
| --- | --- |
| *Случай окружности*  Окружность задается параметрами (*a,b,r*): (*a, b*) — центр, *r* — радиус.  Находим градиент в граничной точке. Его направление совпадает с направлением *r*, значит на этой прямой где-то лежит центр окружности. Перебираем все возможные значения радиуса, получаем набор значений (*a,b,r*) и увеличиваем элементы массива А, расположенные по этим координатам. |  |

## 23. Определитель прямых Барнса

Для каждого пикселя изображения находим градиент и в отдельную матрицу заносим значение, соответствующее углу градиента, квантованному на восемь частей: 0 - [0°,44°] ; 1 - [45°,89°] … 7 - [315°, 359°] (для хранения номера сектора достаточно трёх бит). Затем ко всем углам прибавляем 22,5° и в другую такую же матрицу заносим тоже числа от 0 до 7, но используя новые значения углов.

В каждой из двух матриц с помощью волнового алгоритма происходит поиск групп соседних пикселей, имеющих одинаковое значение. Эти пиксели называются связными. Затем соответствующие группы в двух матрицах сравниваются, и в какой группе связных пикселей больше, ту и считаем отрезком. Если один пиксель входит в два отрезка, то считаем, что он принадлежит тому, в котором больше пикселей.

## 24. Методы сегментации изображений. Модели описания сегментов

**Сегментация** — разбиение изображения на монотонные области. Внутри каждого сегмента (до обработки) пиксели имеют примерно одинаковый цвет. На границах сегментов происходит резкое изменение цвета.

**Классы методов сегментации**:

|  |  |
| --- | --- |
| * *Методы, использующие кластеризацию*   + Определяется метрика P(x,y), показывающая схожесть цветов x и y.   + Множество всех цветов, имеющихся на изображении, разбивается на несколько непересекающихся подмножеств — *кластеров* — так, чтобы “расстояние” между любыми двумя элементами одного кластера было *гораздо меньше*, чем расстояние между элементами разных кластеров.   + Геометрически точки одного кластера могут находиться в разных частях изображения * *Методы наращивания областей*   + Создаётся новый кластер, содержащий одну базовую точку   + В кластер добавляются все соседние к базовой точке, схожие с ней по цвету   + Процедура повторяется рекурсивно с добавленными точками   + Модификации:     - Сравнивать цвета с соседними точками     - Сравнивать цвета с базовой точкой     - Сравнивать со средним значением цвета в накопленном кластере (важно, с какого пикселя начинается поиск) | До сегментации    После сегментации |
| * *Метод водораздела*   Градиент яркости/цвета ∇f показывает степень граничности пикселя, ∇∇f — направление, в котором быстрее всего возрастает степень граничности. Берём некоторый пиксель, вычисляем ∇∇f в нём и двигаемся в противоположном направлении до пикселя, в котором будет минимальное значение ∇∇f. Если ∇∇f =0, то (мы достигли локального экстремума градиента) пиксель образует отдельный сегмент, иначе он принадлежит сегменту, расположенному в направлении, обратном ∇∇f . |  |

**Способы хранения сегментов***:*

|  |  |
| --- | --- |
| 1. *Маркированное изображение*. Для каждого пикселя указывается число — номер сегмента, к которому он принадлежит. Используется очень много памяти, зато позволяет определить принадлежность точки сегменту за О(1). 2. *Квадрантные деревья*. Если квадрант полностью входит в некоторый сегмент, то записывается номер этого сегмента, в противном случае квадрант делится на четыре, и процедура рекурсивно повторяется, пока не будет достигнута необходимая точность. Результат хранится в виде дерева. На примере два сегмента 0 и 1. 3. *Кодирование границ*. У сегмента граница замкнутая, ⇒ можно ее обойти по (против) часовой стрелке. Хранятся координаты начальной точки и закодированный путь обхода (←↑→↓). Используется мало памяти, но неудобно работать. |  |

# Двумерная растровая графика

## 25. Алгоритм Брезенхема для рисования отрезков

Основные требования к алгоритму:

1. Высокая скорость работы.
2. Должна использоваться только целочисленная арифметика.
3. Минимальный расход памяти.
4. Желательно, чтобы алгоритм был инкрементным (цикл не должен делать повторные вычисления, а должен корректировать ранее полученные).
5. Примитивы должны быть восьми- или четырёхсвязными (сколько соседних пикселов, с которыми связан текущий).

Методы растеризации отрезка:

1. **DDA** (*Digital Difference Analytics*). Интересен с теоретической точки зрения. Каждая точка отрезка определяется как

x(t) = x0 + Δx \* t

y(t) = y0 + Δy \* t

t ϵ [0,1]

где

Δx = (x1 – x0) / L

Δy = (y1 – y0) / L

L = max( |x1 – x0| , |y1 – y0| )

|  |  |
| --- | --- |
| 1. **Алгоритм Брезенхема**. Рассмотрим случай построения отрезка из точки (0,0) в заданную точку в первом октанте (остальные случаи получаются с использованием поворота, симметрии и т.п.)   Делаем *x* шагов, на каждом шаге перемещаемся либо вправо, либо по диагонали вправо-вверх и считаем ошибку *е* — расстояние между реальной точкой на отрезке и координатами пикселя. |  |

* + Если е < ½ , то идём вправо и увеличиваем ошибку е = е + dy/dx
  + Если e ≥ ½, то идём по диагонали и уменьшаем ошибку: е = е - 1

Недостатки:

* + алиасинг (при большом растре видна ступенчатость)
  + использование вещественной арифметики

Чтобы исправить второй недостаток, можно внести следующие изменения:

* + изначально из *е* вычесть ½
  + сравнивать ошибку не с ½, а с нулём
  + умножить приращения *e* на 2dx:
    - при шаге вправо: е = е + 2dy,
    - при шаге по диагонали: e = e - 2dx.

При отрисовке текущего пикселя необходимо знать ошибку *е* для следующего шага, чтобы принять решение, куда идти.

Сам алгоритм (вариант с целочисленной арифметикой):

x = x1;

y = y1;

Dx = x2-x1;

Dy = y2-y1;

e = 2\*Dy-Dx; // e = -Dx; ?

for (i = 1; i < Dx; i++)

{

Plot(x, y); //отрисовка точки

if (e >= 0)

{

y++;

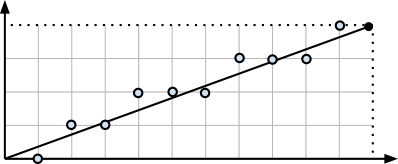
e-=2\*Dx;

}

x++;

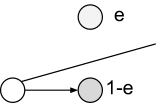
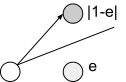
e+=2\*Dy;

}



1. **Алгоритм Ву** (модификация алгоритма Брезенхема)

На каждом шаге рисуем не одну точку, а две, причём яркости распределяем между ними в зависимости от величины ошибки *e*:

а) шаг по горизонтали  б) шаг по диагонали 

## 26. Алгоритм Брезенхема для рисования окружностей

Рассмотрим случай рисования дуги, которая находится в третьем и четвёртом октантах (остальные части окружности рисуются аналогично).

Инициализируем x=-R, y=0. Можем двигаться либо вверх, либо по диагонали вправо-вверх.

Пиксель может находиться либо внутри окружности, либо вне её. Расстояние от пикселя до окружности:

f(x,y) = x2 + y2 - R2

Если f(x,y) > 0, то пиксель находится вне окружности, делаем шаг по диагонали

Иначе — внутри, делаем шаг вверх.

Шаг — это x++ и y++, при этом пересчитываем f(x,y).

При рисовании эллипса необходимо умножать *x* или *y* на некоторый коэффициент. До точки касания эллипса с касательной 45° шаг делается по диагонали, либо по горизонтали, а после — по горизонтали либо по вертикали.

*Отрисовка дуг*

|  |  |
| --- | --- |
|  | Задаются: центр, радиус R, α и β. Для каждого октанта прорисовывается часть дуги, которая попадает в этот октант. Как проверить, попадает ли пиксель в октант? Можно найти точки Z, и если пиксель попадает в промежуток между ними, то он попадает в октант. |

## 27. Кривые Безье. Алгоритм де Костельжо

**Кривые Безье** — это универсальный способ задания произвольных кривых на плоскости или в пространстве. Задаются *многочленом Берштейна*:

где

n — степень кривой

P0 .. Pn — опорные точки, их как минимум две

t ∈ [0,1]

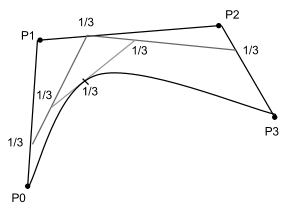
Свойства:

* Гладкость и непрерывность
* Количество изгибов зависит от количества опорных точек
* Степень многочлена Берштейна на один меньше количества опорных точек
* Легко выполнить сложные преобразования (поворот, перенос,..., либо их комбинация)
* Кривая лежит в оболочке опорных точек, которая является многоугольником.
* Кривая всегда проходит через P0 и Pn, и в этих точках касательные к кривой параллельны отрезкам P0P1 и Pn-1Pn.
* Если рассматривать точки в обратном порядке, то форма кривой не изменится.
* Если кривую Безье разбить точкой P(t\*), то мы получим две кривые Безье: t ∈ [0,t\*] и t ∈ [t\*,1]. Таким образом, кривую Безье высшего порядка порядка можно представить в виде набора кривых Безье низшего порядка.

Недостатки:

* Очень много вещественных вычислений
* Трудно подобрать Δt:
  + если взять слишком большим, будут разрывы,
  + если взять слишком маленьким, прорисовка будет повторяться несколько раз.

**Алгоритм де Костельжо**

Задача: построить кривую P(t) по ломаной P0 … Pn.

Сначала берем t\* = ⅓ и делим в таком отношении каждое звено ломаной. Полученные точки соединяем и получаем новую ломаную, степень которой меньше степени исходной ломаной на 1. Повторяем процесс, пока степень не станет равной единице, то есть пока ломаная не выродится в точку. Эта точка будет принадлежать кривой Безье.

Дальше берем t\* = ½ и повторяем алгоритм, и так далее, пока t\* не сольётся с P0 (с заданной точностью)

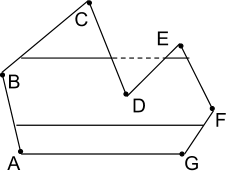
Википедия: полученные в процессе построения промежуточные точки являются опорными точками для двух новых кривых Безье, в точности совпадающих с исходной, и в совокупности дающих исходную кривую Безье.

**NURBS** — рациональные кривые Безье. Для каждой точки вводится ещё и вес (может быть и отрицательным). Он показывает, насколько точка “притягивает” к себе кривую. В этом случае кривая задается: — многочлен Берштейна.

В обычной кривой Безье все w = 1.

## 28. Алгоритмы рисования закрашенного многоугольника

Закрашенная фигура может быть задана границей или растром и точкой, от которой хотим выполнить заливку. Алгоритмы:

1. **Алгоритм с упорядоченным списком рёбер** (для закраски многоугольников)

Горизонтальные прямые, параллельные оси ОХ, пересекают многоугольник в нескольких точках, их число четное.

1. Вычисляются координаты охватывающего прямоугольника: ymin, ymax, xmin, xmax.
2. Находятся координаты всех пикселей, которые принадлежат границам. Не рассматриваются вершины многоугольника и торцевые ребра (принадлежащие охватывающему прямоугольнику).
3. Массив с этими координатами сортируется, причем сначала сравниваются координаты *y*, а потом — *х*.
4. В полученном массиве первая точка соединяется со второй, …, *i*-я c (*i* + 1)-ой. Таким образом получаются горизонтальные отрезки, находящиеся внутри многоугольника, которые называются сканирующими строками.

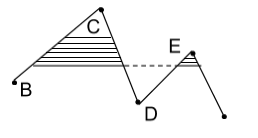
Недостаток:

* + при большом количестве точек на сортировку уйдёт много времени.

Решение недостатка: Разделим массив точек на несколько маленьких. Можно разделить так, чтобы в каждом массиве были точки с одинаковой координатой у. Тогда не надо будет внутри них хранить у - он же у всех одинаковый. Маленькие массивы легко и быстро сортировать, + используется меньше памяти, потому что после того, как нарисованы все прямые с точками из некоторого массива, его можно удалить из памяти и создать новый — это называется “отрисовка по мере необходимости”.

1. **Алгоритм со списком активных рёбер.**

Ребра делятся на активные (растеризованные) и неактивные. Принимаем во внимание ребра по мере прорисовки. То есть, те ребра, которые не участвуют в отрисовке текущей линии, отсутствуют в массиве.



## 29. Алгоритмы заполнения произвольных областей

|  |  |
| --- | --- |
| 1. **Алгоритм заполнения с затравкой** (=заливка)  Вход: граница фигуры, цвета заливки и границы, начальная точка.  Сначала закрашивается начальная точка, а также все те, которые могут быть с ней соединены непрерывным отрезком, не пересекая границу. Потом проверяются все соседние пиксели начальной точки: если их цвет отличается от цвета границы, то делаем рекурсивный вызов.  Недостатки:   * очень много рекурсивных вызовов ⇒ переполнение стека. Для защиты от переполнения можно использовать свой собственный стек, но тогда будет потрачено много памяти. |  |
| 2. **Построчный алгоритм заполнения с затравкой.**  Входные параметры — те же самые.  От начальной точки движемся влево и вправо, пока не достигнем границы. Получаем точки, лежащие на границе, и рисуем отрезок. В цикле рассматриваются все точки над и под этим отрезком, и рекурсивно вызываем процедуру для всех найденных незакрашенных точек. |  |
| **Оптимизация метода**  При проверке строки над/под текущей все её пиксели проверяются на принадлежность границе:   * Если граничных точек нет, то в стек добавляется *одна* (произвольная) точка из проверяемой строки. * Если граничные точки есть, то они разбивают строку на подстроки, от каждой из которых в стек добавляется по одной точке.   В обоих случаях добавляемые точки должны быть незакрашенными. |  |

**Немонотонная закраска**

Цвет каждого пикселя определяется некоторой функцией f(x,y,с), где *х,у* — координаты, *с* — цвет.

* Если f ≡ const, то это *монотонная закраска*.
* Если c ≢ const, то это — *корректировка заливки*, а не заливка.

Например, *эффект градиента* состоит в закраске нескольких точек заданными цветами, а цвета остальных пикселей находятся аппроксимацией этих заданных цветов.

Если *f* задается явно, конкретными значениями, то мы получаем текстурирование. При закраске текстурой для каждой точки изображения (x,y) вычисляется соответствующая точка текстуры (i,j). Наложение текстуры тесно связано с задачей интерполяции, т.к. цвет одного пикселя может определяться исходя из нескольких соседних пикселей.

Если координаты (i,j) линейно зависят от (x,y), то задача вычисления текстурных координат может быть оптимизирована с помощью инкрементной реализации: если точке (x,y) соответствует точка (i,j), то

* точке будет соответствовать
* точке будет соответствовать

где

— частные производные от функций, связывающих (x,y) и (i,j).

## 30. Рисование стилизованных примитивов (толщина и начертание линий, текстуры)

* Для рисования **жирных линий** (прямых, кривых Безье, и т.д.) надо вместо закраски пикселя закрашивать его окрестность некоторым пером, форма которого задаётся. Но чтобы одни и те же пиксели не перерисовывались несколько раз, нужно учитывать угол наклона линии и корректировать форму пера.
* Для рисования **пунктирных линий** отводится массив битов, который задает маску для нужного типа линии (узор): 1 — пиксель надо рисовать, 0 — не надо. При рисовании линии маска последовательно накладывается, и определяется, рисовать пиксель или нет.
* При рисовании **жирной линии пунктиром** за счет толщины пера могут закрашиваться пропуски в пунктире ⇒ необходимо подбирать форму пера исходя из наклона и типа линии.

## 31. Растеризация текста. Шрифты

**Шрифт** — набор изображений букв и знаков, выполненных в едином стиле.

Параметры шрифтов:

* *Начертание* (прямой или курсив)
* *Насыщенность* (отношение толщины букв, штрихов к ширине буквенного просвета. **Bold**)
* *Моноширинность*.
  + Моноширинные шрифты — у всех букв одинаковая ширина. Удобно для использования в псевдографике (рисование таблиц спецсимволами), в ASCII-art и в программировании.
  + Пропорциональные — у всех букв разная ширина.
* *Размер* — измеряется в пунктах (1/72 дюйма). Используется для устройств вывода.
* *Кодировка* — определяет алфавит, используемый для печати.
* *Векторный /* *растровый*
  + *Растровый*: изображения букв хранятся в битовых матрицах. Отрисовка таких шрифтов аналогична отрисовке текстур.
    - Достоинства:
      * Простота и эффективность реализации
      * Требуется мало памяти (изображения отдельных букв имеют слишком малый размер, чтобы эффективно сжиматься при векторизации; кроме того, эти изображения могут быть монохромными).
    - Недостатки:
      * Проблемы интерполяции при трансформации (масштабирование, поворот и др.) Частично решает проблему наличие разных вариантов шрифта для разных размеров.
  + *Векторный*: буквы состоят из глифов. ***Глиф*** — это элемент символа. Глифы бывают простыми (закрашенный примитив, граница которого — кривая Безье) и составными (является комбинацией простых или составных). Векторные шрифты имеют те же преимущества и недостатки, что и векторные изображения: больше изобразительных возможностей, но сложнее отрисовка. Отрисовка сводится к рисованию кривых Безье и закрашиванию примитивов. ***Хинтинги*** — это подпрограммы, которые хранятся вместе со шрифтом и корректируют параметры глифов при значительном масштабировании. Например, при увеличении буквы Ц в 20 раз горизонтальная линия должна расти быстрее, чем вертикальные.

**Задача растеризации**

* Вход: надпись, шрифт, кодировка, координаты
* Выход: растр

## 32. Важность и постановка задачи двумерного отсечения

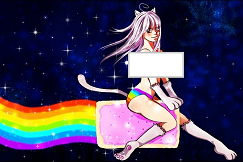
Причины, по которым задача отрисовки важна:

1. **Время**. Если примитив выходит за пределы экрана, то он не будет виден, однако время на его отрисовку всё равно будет потрачено.
2. **Артефакты.** В памяти компьютера изображение хранится в виде последовательности массивов-строк. Если в программе отсутствуют проверки на выход за границы массива, то при попытке доступа к участку изображения за его пределами могут возникнуть графические артефакты. Например, при отрисовке пикселя, находящегося за пределами экрана справа, он будет нарисован слева строкой ниже.

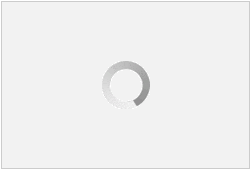
Задача отсечения состоит в том, чтобы создать класс алгоритмов, которые принимают на вход параметры отсекающей фигуры и примитива и определяют, виден ли примитив или нет, а если виден частично — то какая именно часть видна. Простейшее решение — при отрисовке каждого пикселя проверять, выходит ли он за пределы, но это очень медленно.

Параметры отсечения:

* Отсекаемая фигура (примитив, который рисуем)
* Отсекающая фигура (примитив, который задаёт видимую часть; окно видимости)
* Вид отсечения:
  + *Внешнее* — оставляем всё, что *вне* отсекающей фигуры.



* + *Внутреннее* — оставляем всё, что *внутри* отсекающей фигуры.



## 33. Алгоритм Сазерленда-Коэна

|  |  |
| --- | --- |
|  | Это алгоритм внутреннего отсечения отрезка прямоугольным окном.  Продлевая границы окна видимости, получаем девять частей, каждая из которых кодируется четырёхбитным кодом:   * Бит #1 = (область строго левее отсекающего окна) * Бит #2 = (область строго правее отсекающего окна) * Бит #3 = (область строго выше отсекающего окна) * Бит #4 = (область строго ниже отсекающего окна) |

Пусть A и B — коды, соответствующие концам отрезка. Тогда:

|  |  |  |
| --- | --- | --- |
| **Условие** | **Пояснение** | **Вывод** |
| (A **OR** B) = 0 | Коды содержат только нули, то есть оба кода — 0000. | Отрезок полностью виден |
| (A **AND** B) ≠ 0 | Коды содержат единичный бит в одной и той же позиции. | Отрезок полностью не виден |
| (A **AND** B) = 0 | В каждом разряде одного из кодов есть ноль. | Отрезок частично видим. Переносим точку, у которой код ≠ 0, в место пересечения отрезка с одной из границ (например, если включен 1й бит, то с левой, если 2й, то с правой и т.д.) и повторяем алгоритм. |

В результате от отрезка останется только видимая часть.

Недостатки:

* При вычислении точки пересечения отрезка и прямой используется медленная вещественная арифметика
* В некоторых случаях для получения ответа требуется несколько итераций

## 34. Алгоритм разбиения средней точкой

Является модификацией алгоритма Сазерленда-Коэна: точки пересечения находятся не непосредственным вычислением, а методом бинарного поиска. Первый и второй случаи обрабатываются аналогично, а в третьем случае отрезок делится пополам, и каждая из частей обрабатывается отдельно. В конце все видимые части склеиваются в один отрезок, который и будет результатом.

Достоинства:

* Используется только целочисленная арифметика (деление на два реализуется как сдвиг вправо на один бит).

Недостатки:

* Большое количество итераций.

## 35. FastClip алгоритм (быстрого отсечения)

Является оптимизацией алгоритма Сазерленда-Коэна.

Все возможные варианты расположения концов отрезка кодируются восьмибитным кодом (по четыре бита для каждого конца), который интерпретируется одним большим switch с 72 вариантами (всего 92=81, из них девять симметричных). Для каждой ситуации реализуется свой, наиболее оптимальный алгоритм нахождения ответа.

## 36. Алгоритм Лианга-Барски

|  |  |
| --- | --- |
| Отрезок можно представить в векторно-параметрическом виде:  V(t) = v0 + t(v1 – v0), t ∈ [0,1],  или, покоординатно,  x(t) = x0 + t(x1 – x0) = x0 + tΔx  y(t) = y0 + t(y1 – y0) = y0 + tΔy  Потребуем также, чтобы  x0 ≤ x1  y0 ≤ y1 | *~~Рис 1. Резистор~~* |

Тогда задача заключается в нахождении t0 и t1, которые будут соответствовать началу и концу видимой части отрезка. Поставим условие видимости:

xЛ ≤ x(t) ≤ xП

yН ≤ y(t) ≤ yВ

или

xЛ ≤ (x0 + tΔx) ≤ xП

yН ≤ (y0 + tΔy) ≤ yВ

что равносильно системе из четырёх неравенств:

-tΔx ≤ x0 - xЛ

tΔx ≤ xП - x0

-tΔy ≤ y0 - yН

tΔy ≤ yВ - y0

Общий вид этих неравенств:

Pit ≤ Qi  i=1..4

где

P = {-Δx; Δx; -Δy; Δy}

Q = {x0 - xЛ; xП - x0; y0 - yН; yВ - y0}

Каждое из них соответствует одной из сторон отсекающего окна. Инициализируем t0=0 и t1=1 (что означает, что отрезок полностью виден) и последовательно проверяем все условия:

* Если Pi > 0, то t ≤ Qi / Pi ⇒ t1 ≤ Qi / Pi ⇒ t1 := min(t1, Qi/Pi)
* Если Pi < 0, то t ≥ Qi / Pi ⇒ t0 ≥ Qi / Pi ⇒ t0 := max(t0, Qi/Pi)
* Если Pi = 0, то прямая параллельна стороне отсекающего окна. В этом случае:
  + если Qi ≥ 0, то переходим к следующему неравенству,
  + если Qi < 0, то отрезок полностью не виден. break.

## 37. Алгоритм Кируса-Бека

Продолжает идею алгоритма Лианга-Барски для окна видимости в виде выпуклого многоугольника с вершинами F1 … Fn, расположенными по часовой стрелке.

|  |  |
| --- | --- |
|  | Основан на параметрическом представлении отрезка. Строим *n* неравенств Pit≤Qi для *n*-угольника.  Рассмотрим *i*-ю сторону: [Fi,Fi+1].  Находим внутреннюю нормаль Ni.  (Ni,FiFi+1) = 0  nx(xi+1-xi)+ny(yi+1-yi) = 0  Положим nx=1; ny=(xi-xi+1)/(yi+1-yi). (Знаменатель может быть 0, если грань горизонтальная, нормаль вертикальная). |

После нахождения нормали нужно определить, внутренняя она или внешняя.

Если (N, Fi-1Fi) < 0 то нормаль внутренняя. Если >0, умножаем нормаль на -1.

Нашли внутреннюю нормаль.

Pi=(Ni,(V1-V0))

|  |  |
| --- | --- |
|  | Если (Pi > 0) то увеличиваем t0, иначе уменьшаем t1.  Если Pi = 0, то грань коллинеарна с вектором.  Qi(t)=(V(t)-Fi, N)  Qi(t)<0; угол больше 90’, точка снаружи  Qi(t)>0; точка внутри  Qi(t)=0; точка лежит на стороне многоугольника. |

Решая уравнение Qi(t)=0, получим t, обозначающее пересечение отрезка с гранью.

Если Pi>0, то это t — минимальное t0.

Если Pi<0, то это t — максимальное t1.

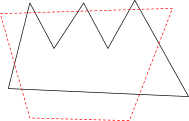
Если Pi=0, то отрезок параллелен.

Qi > 0 — продолжаем, переходим к следующему неравенству.

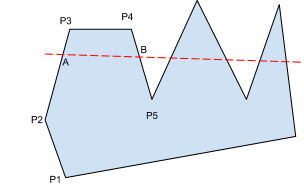
Qi < 0 — отрезок невидим.

Qi=(V0-Fi, Ni)

## 38. Алгоритм Сазерленда-Ходжмена

Отсечение выпуклым многоугольником любого многоугольника.

В цикле рассмотрим стороны отсекающего многоугольника. Каждый раз отрезаем по кусочку прямой. Во внутреннем цикле рассмотрим вершины отсекаемого многоугольника и формируем новый массив вершин.



У каждого ребра может быть четыре состояния:

* Ребро видимо
* Ребро выходит из области видимости
* Ребро невидимо
* Ребро входит в область видимости.

На каждом шаге рассматриваем Pi-1,Pi. S — последняя добавленная в результат вершина. Если [S,Pi] полностью *видимо*, добавляем Pi в результат, S=Pi. (P1P2 - полностью видимо).

Если отрезок *выходит*, точку пересечения (A) добавляем как S. (P2P3).

Если ребро *невидимо*, ничего не добавляем (P3P4).

Если ребро *входит*, добавляем и точку пересечения (B), и конец ребра (P4,P5).

Так идём до конца.

## 39. Фрактальная графика

**Фрактал** — геометрическая фигура, обладающая следующими свойствами:

* Самоподобие при неограниченном увеличении. При любом масштабе получаем картинку, подобную начальной фигуре.
* Эта фигура должна иметь нетривиальную структуру (т.е. не линия и не точка), и при увеличении ни один фрагмент не должен вырождаться в монотонный.

Пример: ветка папоротника.

Способы задания:

1. *Геометрический*. Задается набор правил, которые по данному изображению позволяют построить изображение на следующей итерации. Пример: ломаная Коха. IFS — система интегрирующих функций. Позволяет по изображению А получить его кусочек А1. При фрактальном сжатии по изображению строится IFS.
2. *Алгебраический*. Задается набором рекуррентных формул.
   * Пример: фрактал Мондельброта. Рассмотрим последовательность комплексных чисел {zn}.

z0 = 0

zn+1 = zn2 + c

Для каждого *с* можем построить последовательность {zn}, которая будет начинаться с некоторого N и |zn|<a. N — количество операций для выхода из области.

А теперь — весёлые картинки:

|  |  |
| --- | --- |
| Фрактал Коха (вектор): | Фрактал Коха (растр): |
|  |  |

Фрактал руки (n=3):



# Трёхмерная графика

## 40. Координатный метод описания объектов. Обобщённые координаты. Матрицы преобразования координат

Основные параметры сцены:

* Расположение объектов
* Расположение источников освещения
* Расположение камеры

Простейшие манипуляции:

* Перемещение (x:=x+∆x; y:=y+∆y; z:=z+∆z)
* Поворот (вокруг z: x’ := x cos α + y sin α; y’ := -x sin α + y cos α; z’ := z)
* Масштабирование (x := kxx)
* Отражение по оси: \*(-1)

Некоторые преобразования приходится делать в несколько этапов. Например, увеличение сцены в определённом месте будет выполняться в три шага:

* Смещение начала координат в нужную позицию
* Изменение масштаба
* Возврат начала координат в исходное положение

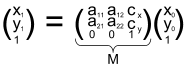
Виды координат:

* декартовы (x, y, z)
* обобщённые (x, y, z, w) — позволяют задать бесконечно удалённые точки (при w=0)

Связь:

* (x, y, z, w)об = (xдек, yдек, zдек, 1)
* (x, y, z)дек = (xоб, yоб, zоб) / w — не всегда возможно

**Аффинное преобразование** — преобразование, которое можно задать в виде:

A2X2 ; или 

Комбинацию аффинных преобразований можно представить как одно аффинное преобразование, матрица которого есть произведение матриц комбинируемых преобразований:

* x1 = M1x0
* x2 = M2x1
* …
* xn = Mnxn-1 = Mn(Mn-1xn-2) = Mn(Mn-1(Mn-2xn-3)) =...= (MnMn-1...M1)x0 = Mx0

Примеры аффинных преобразований координат:

* Мировые → видовые;
* Проекционные → экранные.

## 41. Проецирование. Виды координат (модельные, видовые, проекционные, экранные)

Координаты бывают:

1. *Мировые* (точка в пространстве).
2. *Видовые* (получаются путём передвижения мировых).
3. *Проекционные* (относительно плоскости проецирования, Z — расстояние до объекта).
4. *Экранные* (координаты пикселей).

**Поверхность проецирования** — поверхность, на которой строится изображение (обычно плоскость).

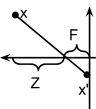
**Проецирование** — процесс (задача) сопоставления каждой точке трёхмерной сцены некоторой точки поверхности проецирования.

**Луч проецирования** — луч, который идет из объекта к поверхности проецирования.

Виды проецирования:

* **Параллельное**: все лучи проецирования параллельны друг другу. Подвиды:
  + **Косоугольное**: лучи не перпендикулярны плоскости проецирования.
  + **Аксонометрическое**: лучи перпендикулярны плоскости проецирования. Если использовать систему координат (x,y,z), в которой z — это расстояние от поверхности проецирования до объекта, то все лучи будут параллельны оси Z (hi, co?). В зависимости от соотношения коэффициентов масштабирования kx, ky, kz (по осям X,Y,Z соответственно) выделяют следующие модификации:
    - **Изометрическое**: kx = ky = kz.
    - **Диметрическое**: kx = ky ≠ kz.
    - **Триметрическое**: kx ≠ ky ≠ kz.
* **Перспективное**: все лучи проецирования проходят через одну точку, они не параллельны. Чем меньше фокусное расстояние, тем меньше размер проекции на плоскости.

С фокусным расстоянием связано увеличение и угол обзора. Чем меньше фокусное расстояние, тем больше угол обзора, и наоборот.

Фокусное расстояние человеческого глаза приблизительно равно 22 мм; каждый глаз по отдельности имеет угол зрения порядка 120-200°, соответственно, зона перекрытия двух глаз составляет порядка 130° — она практически настолько же широка, как у объектива типа "рыбий глаз". Однако по эволюционным причинам наше периферийное зрение пригодно только для обнаружения движения и крупных объектов. Наш центральный угол зрения — порядка 40-60° — максимально влияет на наше восприятие. Субъективно это соотносится с углом, в пределах которого вы сможете вспомнить объекты, не двигая глазами. Это близко к углу зрения портретного объектива с фокусным расстоянием 52 мм.

**Кроп-фактор** — это отношение физического размера пленки к размеру матрицы, бывает от 1 до 2. **Full-frame** — это фотоаппарат, у которого размеры матрицы и пленки совпадают (кроп-фактор = 1).

Если *x* — координата объекта, а *x’* — координата на поверхности проецирования, то . Таким образом, перспективное преобразование можно задать как аффинное (с помощью матрицы 4х4 с ~~блекджеком~~ обобщёнными координатами).

## 42. Аналитическая модель описания поверхностей

Любая поверхность задается с помощью уравнения F(x,y,z) = 0, где x,y,z — мировые координаты.

NURBS: с использованием опорных точек и их весов можем задавать поверхность в пространстве (аналог кривых Безье).

Достоинства:

* Простота
* Малый объём информации
* Универсальность (может быть описано большое множество поверхностей)
* Позволяет легко проверить принадлежность точки объекту

Недостатки:

* Вычислительная сложность (много делений ⇒ сложная арифметика)
* Неудобно работать: чтобы получить координаты *x*, *y*, *z*, надо выражать их из уравнения
* При выполнении аффинных преобразований меняется уравнение
* Отсутствие универсального метода визуализации (для каждого вида поверхности свой)
* Нельзя делать разрез объекта, можно задавать только поверхность

## 43. Векторная модель описания поверхностей

Поверхность объекта представляется в виде набора граней со сторонами (рёбрами). Для каждой грани задаются координаты вершин + дополнительные параметры: цвет, текстура, материал (поведение при освещении), нормаль (какая из сторон лицевая), и т.д.

**Триангуляция** — разбиение произвольной гладкой поверхности на составляющие грани (на практике удобно работать с треугольниками).

Подходы к хранению объектов:

1. В явном виде хранятся **координаты** каждой вершины (x,y,z). Тогда одна точка будет храниться несколько раз ⇒ большой расход памяти, и надо следить, чтобы координаты точек у разных граней совпадали.
   * *Достоинства*: даёт полную свободу редактирования
   * *Недостатки*: требует много памяти
2. Хранятся **рёбра** как пары точек, а для каждой грани (треугольника) хранятся три ссылки на рёбра.
   * *Достоинства*: может быть экономнее, чем первый вариант
   * *Недостатки*: даёт свободу только с точностью до ребра
3. Хранится массив с координатами **вершин**, а для граней хранятся только индексы вершин в этом массиве. В этом случае придется часто лазить в разные ~~нехорошие~~ области памяти.
   * *Достоинства*: максимальная экономия памяти (вершины не повторяются)
   * *Недостатки*: все грани связаны

**Замечание**. Второй метод оказывается эффективным только в том случае, если одно ребро используется во многих гранях, в противном случае расходы памяти могут оказаться даже выше, чем в первом варианте. Сравним этим методы на конкретных примерах:

Пример 1. Рассмотрим кубик.

1. Для каждой грани храним четыре точки, итого получаем 4\*6 = 24 точки.
2. Для каждой грани храним четыре ссылки на рёбра. В списке рёбер все рёбра уникальны. Итого храним 12 рёбер по две точки, 12\*2 = 24 точки.
3. Точки храним в списке уникальных точек, соответственно, их всего 8.

Пример 2. Рассмотрим два треугольника, имеющих одну общую сторону.

1. Две грани по три точки, итого 6 точек.
2. Пять рёбер по две точки, итого **10** точек.
3. Четыре точки.

(— А еще неплохо было бы учесть, что сами объекты ребра и граней занимают память)

(— Занимают, но их в несколько раз меньше, чем вершин, поэтому они, скорее всего, и будут определять “асимптотику” расхода памяти.)

**Детализация** — количество полигонов, описывающих поверхность. Чем выше детализация, тем больше реалистичность и сложность. Виды:

* *Статическая*: для каждого объекта строится одна модель, которая всегда используется при построении изображения.
* *Динамическая*: для объекта строится несколько моделей, с разным уровнем детализации. Если объект далеко, используется модель с низкой детализацией, иначе — с высокой.

*Достоинства векторной модели:*

* Универсальность и простота
* Возможность выполнения аффинных преобразований
* Сравнительно небольшой объем данных
* Наличие универсальных алгоритмов визуализации, а также возможность их аппаратной реализации

*Недостатки векторной модели:*

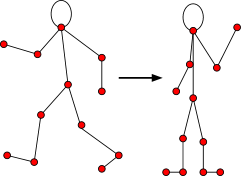
* Сложность выполнения разрезов
* Для хорошего качества изображения необходим высокий уровень детализации.

**Морфинг** — технология, создающая впечатление плавной трансформации одного объекта в другое. Объектами морфинга обычно являются изображения, но могут быть и трёхмерные модели. На конечных изображениях (объектах) обычно задаётся набор соответствующих контрольных точек, после чего строятся промежуточные изображения путём интерполяции положения и цвета пикселей:



~~например когда строится модель лица, есть N выражений лица, и для нужной эмоции берется M<N моделей и вычисляется среднее для каждой точки в зависимости от приоритета~~.

Ещё пример: для каждого ребра скелета строится связанная с ним поверхность. Когда поворачиваются рёбра, поворачивается вся поверхность.



## 44. Воксельная модель описания поверхностей

|  |  |
| --- | --- |
|  | По аналогии с двумерными растровыми изображениями, состоящими из пикселей, можно создавать трёхмерные модели, состоящие из **вокселей** — маленьких кубиков, имеющих определённое положение, цвет и прозрачность. |

Достоинства метода:

* Хорошо подходит для моделирования непрерывных сред
* Хорошо подходит для трассировки лучей
* Позволяет строить сложные модели по данным аналоговых датчиков
* Позволяет строить разрезы (используется в медицине, например, в МРТ)

Недостатки метода:

* Большой расход памяти (N3)
  + ⇒ низкое разрешение
  + ⇒ низкая скорость обработки
* Сложность отрисовки

|  |  |
| --- | --- |
| Для экономии памяти применяют **метод октантного деления** (по сути сжатие с потерями): исходный куб делится на восемь кубов, и анализируется степень монотонности вокселей внутри каждого из них. Если воксели достаточно монотонны, весь куб заменяется одним большим вокселем среднего цвета, если нет — процедура деления рекурсивно повторяется. Полученный результат хранится в виде дерева.  Достоинства:   * Экономия памяти * Динамическое изменение разрешения в разных участках модели   Недостатки:   * Сложность обработки |  |

## 45. Классификация способов визуализации (рендеринга) трёхмерных сцен

**Рендеринг** — это построение двумерного изображения трехмерной сцены согласно заданному положению камеры, освещению, моделей объектов и пр.

*Источники освещения:*

1. *Точечные*: свет выходит из одной точки (характеризуется координатами). Свет падает под разными углами даже в близкие точки объекта.
2. *Бесконечно удаленные источники* (характеризуется направлением). Все лучи параллельны друг другу, угол падения одинаковый у любых точек (например, Солнце).
3. *Фоновое освещение*: не знаем, откуда берутся лучи; их невозможно проследить, в каждой точке луч падает под всеми углами. Результат всех возможных отражений света.
4. *Излучение*: часть поверхности некоторого объекта, которая сама по себе светится, но никакие другие предметы не освещает.

Луч может отражаться, преломляться, поглощаться, а также комбинировать эти действия. Мы видим то, что попало в камеру. *Уравнение рендеринга:* для каждой точки объекта

интенсивность исходящего луча = интенсивность отраженного + преломленного - поглощенного луча = сумма всех входящих лучей – сумма всех поглощенных.

Чем больше препятствий на пути от источника до предмета и от предмета до камеры мы учитываем, тем качественнее изображение. Методы рендеринга различаются тем, что учитывается.

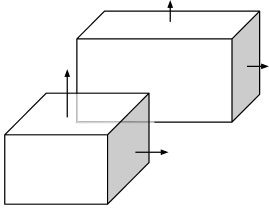
|  |  |
| --- | --- |
| **Метод** | **Что учитываем** |
| 1) Проволочная модель | Модель объекта и положение камеры |
| 2) Удаление невидимых линий (ребер)  а) с закраской  б) без закраски | Учитываем путь луча от объекта до камеры, наличие преград. Если с закраской, то закрашиваем без теней и оттенков. |
| 3) Учет освещения | Все грани рисуем закрашенными, цвет определяем исходя из параметров источника. |
| 4) Учет гладких поверхностей и текстур | Является ли грань частью гладкой поверхности, накладывается ли на эту грань изображение (текстура) |
| 5) Моделирование теней | Учитываются препятствия на пути луча от источника освещения до точек объекта. |
| 6) Учет отражений и преломлений | Ну тут и так всё понятно. |

## 46. Алгоритм Робертса для многогранников

**Вход**: векторная модель объектов и положение камеры.

**Выход**: набор отрезков, которые надо нарисовать - удаляются невидимые грани (в экранных координатах).

Алгоритм:

1. Изначально множество отрезков пустое.
2. Для каждой грани вычисляется нормаль (векторное произведение двух её сторон) и с её помощью определяем ориентацию грани (направлена к нам “лицом” или нет). Обходим точки в направлении по часовой стрелке.
3. Из рассмотрения удаляются грани, нормаль которых “отвернута” от камеры: если скалярное произведение нормали на направление камеры (n, (0,0,1)) > 0, то угол острый, и грань не видна.
4. Все видимые грани сортируются в порядке удаления от камеры.
5. В цикле рассматриваются грани с самой дальней до самой ближней: для каждой грани просматривается множество уже накопленных отрезков (алгоритмом двумерного отсечения):
   1. Если отрезок полностью внутри грани, то он удаляется из результирующего множества
   2. Если отрезок полностью вне грани, то он остается в результате.
   3. Если отрезок пересекается с гранью, то он удаляется из множества, но вместо него добавляется видимая часть

После этого все ребра грани добавляются в множество, и берется следующая грань.

Проблема в двойном цикле: из-за этого низкое быстродействие.

## 47. Алгоритм плавающего горизонта для отрисовки поверхностей

|  |  |
| --- | --- |
|  | Алгоритм применяется для поверхностей, заданных уравнением y = f(x,z). Поверхность можно задать набором функций y(x) = f(x, zi), i=0..n. Создаются два массива: максимальных и минимальных значений *y* для каждого *x*. |

Алгоритм:

1. Инициализируются массивы ymax[x] и ymin[x].
2. Рисуется y = f(x, zn) — это самая ближняя линия.
3. Цикл по *z* в порядке удаления от камеры
   * Цикл по *x:*
     + Вычисляется *y* = *f(x, z)* и проверяется:
       - Если *y* > *y*max[*x*], то изменяем ymax[x] и рисуем точку (*x, y, z)*.
       - Если *y* < *y*min[*x*], то изменяем ymin[x] и рисуем точку (*x, y, z)*.
       - Иначе точка не рисуется

Главный недостаток алгоритма — его узкая специализированность.

## 48. Алгоритм с сортировкой по глубине

**Вход**: векторная модель (параметры сцены, расположение камеры).

**Выход**: растр.

Для каждой грани вычисляется удалённость (Z). Все грани сортируются по убыванию, и начиная с самой дальней, они рисуются с закраской цветом.

Недостатки:

* Низкая скорость, т. к. один и тот же пиксель перерисовывается несколько раз.
* Иногда невозможно определить, какая грань ближе, а какая дальше (например, пересекающиеся грани будут отрисованы неверно).

## 49. Алгоритм Z-буфера и построчного сканирования

|  |  |
| --- | --- |
|  | **Вход**: векторная модель (параметры сцены, расположение камеры)  **Выход**: растр.  Каждая грань рисуется модифицированным алгоритмом закраски многоугольника. Для каждого пикселя растрового изображения хранится ещё и z — расстояние от камеры до пикселя.  Если проецирование ведётся на плоскость XY, а проецируемая грань (треугольник) лежит в плоскости {Ax + By + Cz + D = 0}, то в точку (x,y) экрана будет спроецирована точка (x, y, (– Ax – By – D) / C), причём последняя координата будет обозначать расстояние до камеры. |

Изначально матрица значений Z (Z-буфер) инициализируется некоторым очень большим числом.

* Если Z рисуемого пикселя меньше соответствующего ему элемента в Z-буфере, то пиксель рисуется, и в Z-буфер заносится его значение Z.
* Иначе пиксель пропускается.

Для ускорения вычислений Z можно вычислять его только для одной вершины, а затем добавлять частные производные:

Достоинства:

* Простота реализации.
* Приемлемость для сцен любой сложности.
* Возможность распараллеливания (делим изображение на несколько частей, и каждую обрабатываем в отдельном потоке).

Недостатки:

* Большие объемы памяти
* Z — это вещественное число, вычисление производится с высокой точностью
* При реализации полупрозрачности требуется несколько z: z1, z2... ⇒ еще больше ресурсов. В этом случае можно рисовать сначала все непрозрачные объекты, а потом — все полупрозрачные

**Алгоритм построчного сканирования** (модификация предыдущего алгоритма).

Делаем полигон объектом, который имеет методы:

1. Init
2. GetCurrentY — на какой строке сейчас стоит алгоритм
3. DrawCurrentLine — рисует текущую часть объекта
4. bool next — существует ли следующая строка

В цикле по всем строкам изображения проверяем, пересекает ли текущая строка какой-нибудь полигон, и если да, то рисуется строка полигона (DrawCurrentLine и next) с использованием Z-буфера.

Достоинства:

* храним Z-буфер только для текущей сканируемой строки ⇒ экономия памяти.

## 50. Алгоритм Варнока

**Вход**: векторная модель (параметры сцены, расположение камеры).

**Выход**: растр.

Для каждой грани вычисляются экранные координаты её вершин. Рассматривается множество многоугольников и прямоугольное окно.

|  |  |
| --- | --- |
| Если все многоугольники внешние, то закрашиваем окно цветом фона. |  |
| Если только один многоугольник пересекается с окном, то выполняется его отсечение (окном) и рисуется только видимая часть. |  |
| Если с окном пересекается несколько многоугольников, но при этом самый ближний из них является охватывающим (окно целиком лежит внутри него), то окно закрашивается цветом этого многоугольника. |  |
| Если размер окна = 1 пиксель, то он закрашивается цветом ближайшего многоугольника и на этом работа алгоритма прекращается. | *янезнаюкакэтонарисовать.jpg* |
| Если с окном пересекается несколько многоугольников и ни один из них не является охватывающим, то окно делится на четыре части и для каждой части опять запускается алгоритм с начала. |  |

Замечания

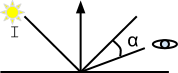
* Глубина рекурсии оказывается небольшой (при размерах 1024х1024 глубина = 10)
* Алгоритм может быть распараллелен.

## 51. Алгоритм Вейлера-Айзертона

На входе и на выходе векторная модель.

1. Все полигоны сортируются по глубине и находится самый близкий к камере многоугольник.
2. Все многоугольники отсекаются этим ближайшим многоугольником и формируются два списка: внутренний (то, что внутри отсекающего многоугольника) и внешний (то, что вне отсекающего многоугольника)
3. Если внутренний список пуст, то отсекающий многоугольник ни с чем не пересекается. Добавляем его в результат, и весь алгоритм выполняется для внешнего списка.
4. Иначе, сравниваются Z отсекающего многоугольника и каждого из внутреннего списка:
   * Если там есть хотя бы один многоугольник, который ближе отсекающего, то к нему рекурсивно применяется весь алгоритм.
   * Иначе отсекающий многоугольник заносится в результат и весь алгоритм выполняется для внешнего списка

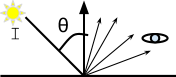
## 52. Модели отражения света и их использование при закраске

**1. Зеркальное отражение света:** Is=Icospα

α — угол между лучом отражения и направлением камеры.

p — степень полировки поверхности.

I — яркость источника

**2. Диффузное отражение** — отражение от матовых поверхностей.

Луч отражается во всех направлениях одинаково, где расположена камера — неважно: Id=Icosθ

θ — угол между нормалью к поверхности и лучом из источника.

**3. Фоновое освещение** (попадает в камеру независимо ни от чего): Ia=const.

**4. Сумма первых двух вариантов:** I = KaIa + KsIs + KdId

Ka,s,d  — коэффициенты, задающие свойства поверхности.

Ia,s,d — яркости разных освещений.

Если источников много, то

I = KaIa+Σ[F(R)(KsIs+KdId)]

где

R — расстояние,

F(R) ϵ [0,1] — функция ослабления.

Например, F(R) = 1 / (K1 + K2R + K3R2)

Если свет не монохромный, то все предыдущие действия производятся отдельно для каждого канала R,G,B. Если вектор С=(R,G,B), то

IC=I\*Cисточника\*Cточки объекта

— например, при освещении синей поверхности красным светом она будет выглядеть чёрной.

## 53. Методы закраски: монотонная закраска, метод Гуро и метод Фонга. Имитация микрорельефа (карты нормалей)

*Методы закраски:*

* **Монотонная закраска**
  + Алгоритм:

1. Вычисляем нормаль к полигону
2. Вычисляем цвет
3. Рисуем закрашенный полигон полученным цветом
   * *Достоинства*:

* Простой, как палка, не требует дополнительных вычислений.
  + *Недостатки* — связаны с вычислением нормали:
* Невозможно моделировать тела вращения.
* Метод себя плохо ведет ~~(~~*~~наказать!~~*), если источник освещения слишком близко расположен к поверхности. Это связано с тем, что угол падения и отражения — разный, а нормали одинаковые.
* **Закраска методом Гуро (немонотонная)**
  + Алгоритм:

1. Вычисляется нормаль к грани
2. Вычисляется нормаль к каждой вершине путем усреднения нормалей для соответствующих граней.
3. Согласно закону освещения, для каждой вершины [треугольника] вычисляется интенсивность (цвет).
4. Используется модифицированный алгоритм рисования закрашенного многоугольника. В каждом пикселе значение цвета определяется как средневзвешенное значений на ребрах, точнее, в местах пересечения рёбер со сканирующей строкой. Интенсивность на ребрах получается путём усреднения интенсивности на вершинах.
   * *Недостатки*:

* Баг метода: подносим источник света к центру прямоугольника, тогда весь прямоугольник будет залит одним цветом (поскольку в углах интенсивность освещения одинаковая), хотя центр должен быть освещён сильнее.
* **Закраска методом Фонга**
  + Алгоритм
* Отличается тем, что путем усреднения вычисляется нормаль к каждому пикселю полигона, а уже потом вычисляется освещение. Теперь все будет честно нарисовано, но дорогой ценой (N=(Nx,Ny,Nz) — вещественный вектор).
  + *Достоинства*:
* Более качественная картинка.
  + *Недостатки*:
* Ресурсоёмкость и понижение скорости
* **Оптимизация метода Фонга**
  + Алгоритм
* У тел вращения угол нормали меняется не сильно. Считаем (α, β) — насколько сильно отклоняется нормаль в точке отклонения от нормали ко всему полигону.
* Создаем “карту” (матрицу), позволяющую из (α, β) получить N

*Имитация микрорельефа:*

* Честное задание шершавой поверхности в виде множества полигонов
  + *Недостаток*: низкая скорость
* **Bump Mapping**. Используется *карта высот* — аналог текстур, но хранящая не цвета, а поправки к координатам поверхностей (в перпендикулярном направлении).
* **Normal Mapping**. Аналогично BM, но хранятся поправки к нормалям.
* **Displacement Mapping**. Аналогично BM, но хранятся смещения в произвольном направлении.

Чем сложнее задаётся карта нормалей, тем тоньше получается рисунок, но и тем больше требуется ресурсов.

## 54. Моделирование теней

Методы:

* **Построение теней на землю.** Перед основной отрисовкой все грани проецируются на землю и рисуются тёмным цветом.
  + *Достоинства*: простота
  + *Недостатки*: тень не падает на предметы и может оказаться “под” ними.
* **Использование расстояний**
  + Размещаем камеру в точке освещения и делаем проекцию сцены. Получаем Z — расстояние от камеры до любой освещённой точки.
  + Помещаем камеру в её нормальное положение и запускаем алгоритм отрисовки. Для каждой точки по проекционной координате вычисляются мировые координаты, потом они переводятся в координаты с точки зрения источника, и проводится анализ освещённости: если Zпроекц. > Zбуфера, то точка находится в тени, иначе — освещена.
* **Метод теневых объёмов**

*Теневой объём* — это фигура, образованная конечным числом усеченных трапеций, которая находится в тени. Всё, что внутри него — в тени. Для каждого пикселя можно посчитать количество объемов, в которые он попадает. Для каждого пикселя сцены можно проанализировать луч, идущий от камеры до объекта. Для него вычисляется, сколько раз он входит в теневой объём и сколько раз он выходит оттуда. Если количество входов равно количеству выходов, то точка освещена, иначе — в тени.

Алгоритм:

* + - 1. Отрисовка всей сцены, как будто она вся в тени. Используется Z-буфер.
      2. Рисуются грани теневого объёма и для каждой из них определяется, она повёрнута к камере или от неё.
      3. Заводится дополнительный буфер, где для каждого пикселя хранится число N теневых объёмов, в которые он попадает. Изначально N=0.
         * Если Z пикселя (из Z-буфера) больше, чем Z грани:

Если грань повёрнута к камере, то N++

Если грань повёрнута от камеры, то N--

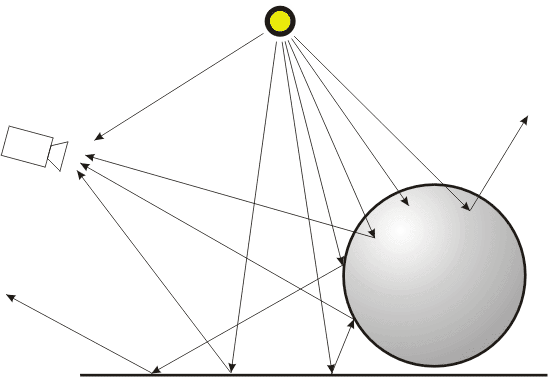
* + - * + Иначе N не изменяется.
      1. Пиксели, у которых N == 0, рисуются без учета теней, поверх изображения

Недостатки:

* Громоздкий метод со всех точек зрения
* Использование информации о глубине (дополнительный буфер, много памяти)

## 55. Метод трассировки лучей

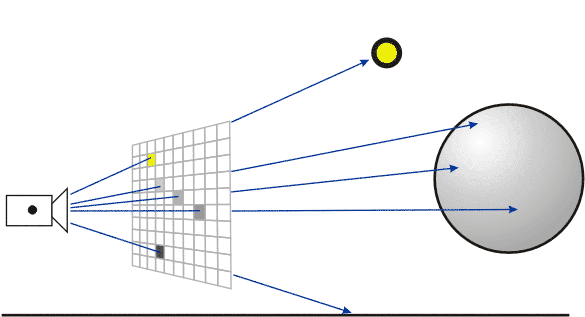
|  |  |
| --- | --- |
|  | На рисунке изображена прямая трассировка луча. Луч от источника преломился, т.к грань не сплошная. |

Прямая трассировка без преломления

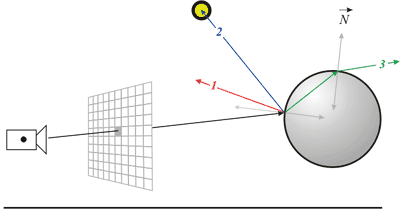
В forward ray tracing траектории лучей строятся от источника ко всем точкам всех объектов сцены (первичные лучи). Затем проверяется ориентация каждой точки относительно источника, и, если она лежит на стороне объекта, обращенной в противоположную от источника сторону, точка из расчетов освещенности исключается. Для всех остальных точек вычисляется освещенность с помощью локальной модели освещения. Если объект не является отражающим или прозрачным, то есть поверхность объекта только диффузно рассеивает свет, траектория луча на этой точке обрывается (заканчивается). Если же поверхность объекта обладает свойством отражения (reflection) и/или преломления (refraction), из точки строятся новые лучи, направления которых совершенно точно определяются законами отражения и преломления.

Для построенных таким образом траекторий новых лучей может быть только три исхода. Луч либо выходит за пределы видимой из камеры области сцены, в этом случае все проделанные для него до этого момента расчеты освещенности отбрасываются, поскольку они не принимают участия в формировании изображения. Или луч попадает в камеру, тогда рассчитанная освещенность формирует цвет соответствующего пиксела изображения. Или луч встречает на своем пути новый объект, тогда для новой точки пересечения повторяется расчет освещенности и построения лучей отражения и преломления в зависимости от свойств поверхности объекта (рекурсия). Построение новых траекторий и расчеты ведутся до тех пор, пока все лучи либо попадут в камеру, либо выйдут за пределы видимой области. Очевидно, что при прямой трассировке лучей мы вынуждены выполнять расчеты для лучей, которые не попадут в камеру, то есть, проделывать бесполезную работу. По некоторым оценочным данным доля таких "слепых" лучей довольно велика.

Второй метод — обратная трассировка лучей, или backward ray tracing. Этот метод расчетов основывается на построении лучей не от источника, от наблюдателя через плоскость экрана вглубь сцены.

Лучи теперь строятся по двум точкам: первая точка, общая для всех лучей - положение камеры (наблюдателя), вторая точка определяется положением пикселя на плоскости видового окна. Таким образом, направление каждого луча строго определено, и количество первичных лучей также известно - это общее количество пикселей видового окна. Например, если видовое окно имеет 800 пикселей по ширине и 600 пикселей по высоте, количество лучей составит 800х600= 480 000.

Каждый луч вдоль заданного направления продляется от наблюдателя вглубь трехмерной сцены, и для каждой траектории выполняется проверка на пересечение со всеми объектами сцены и с отсекающими плоскостями. Если пересечений с объектами нет, а есть пересечение только с плоскостью отсечения, значит луч выходит за пределы видимой части сцены, и соответствующему пикселю видового окна присваивается цвет фона. Если луч пересекается с объектами сцены, то среди всех объектов выбирается тот, который ближе всего к наблюдателю. В точке пересечения с таким объектом строится три новых, так называемых вторичных луча.

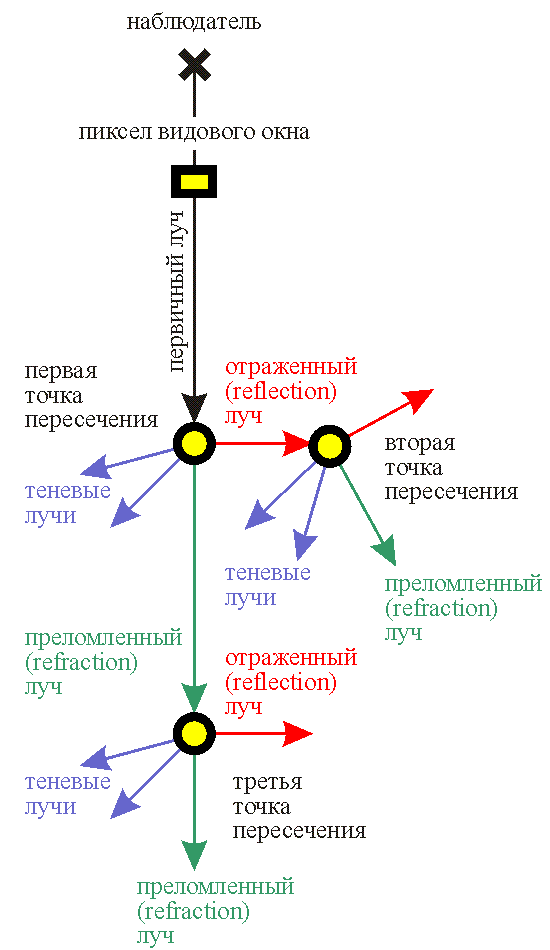


Один луч строится в направлении источника света. Если источников несколько, строится несколько таких лучей, по одному на каждый источник. Основное назначение этого луча - определить ориентацию точки (обращена точка к источнику или от него), наличие объектов, закрывающих точку от источника света, и расстояние до источника света. Если точка обращена в противоположную сторону от источника света или закрыта другим непрозрачным объектом, освещенность от такого источника не рассчитывается, точка находится в тени. В случае затеняющего прозрачного объекта интенсивность освещения уменьшается в соответствии со степенью прозрачности. Если точка закрыта от освещения всеми источниками сцены, ей присваивается фоновый ambient цвет. В противном случае точка освещена, интенсивность и цвет освещения рассчитываются при помощи локальной модели освещенности (diffuse + specular) как сумма освещенностей от всех источников, для которых эта точка не закрыта другими объектами. Этот тип луча получил название shadow ray (иногда его еще называют illumination ray) - теневой луч. Если поверхность объекта не является отражающей и непрозрачна, теневой луч - единственный тип лучей который строится, траектория первичного луча обрывается (заканчивается), и дальнейшие расчеты не выполняются. Рассчитанный цвет (освещенности или тени) присваивается тому пикселю видового окна, через который проходит соответствующий первичный луч.

Второй луч строится, если поверхность объекта обладает отражающими свойствами, и называется reflection ray, или луч отражения. Для отраженного луча проверяется возможность пересечения с другими объектами сцены. Если пересечений нет, то интенсивность и цвет отраженного луча равна интенсивности и цвету фона. Если пересечение есть, то в новой точке снова строится три типа лучей - теневые, отражения и преломления.

Третий луч строится, если поверхность объекта прозрачна, и носит название transparency ray, т.е. луч прозрачности.

Таким образом, для каждого первичного луча можно построить древовидную структуру следующего вида:



Недостатки метода:

# Трудоёмкая операция нахождения ближайшей грани.

* Сложность реализации.
* Большой объём вычислений. ~~Не просто большой, опупенный~~
* Вклад освещенности от каждого нового вторичного луча очень быстро уменьшается, так как коэффициенты свойств отражения и преломления материалов меньше единицы. Например, если коэффициент отражения считать одинаковым для всех поверхностей и равным, допустим 0.3, то вклад четвертого отражения уменьшится в 0.0081 (0.3\*0.3\*0.3\*0.3), то есть - почти в тысячу раз, а общее количество лучей к этому моменту увеличится в 12 раз (для одного источника света и в предположении, что все объекты прозрачны и отражают свет). Поэтому часто трассировку лучей прекращают, когда вклад от следующего узла ветви становится меньше заданной величины.
* Неучет вторичного освещения от диффузно отраженного объектами света
* aliasing - "зазубренность" линий

# Возможная оптимизация метода:

* Источники света только светят, но не отражают и не преломляют (минус красивые лампочки)
* Упрощают свойства отражающих поверхностей (этот пункт для меня загадка, расшифруйте если кто-то знает :) )
* При зеркальном отражении не делаем свечения
* Вводим ограничение на глубину рекурсии
* Вводим ограничение на интенсивность луча. Максимальная интенсивность в камере М -> М1 при первом отражении/преломлении -> …

Достоинства метода:

* Универсальность. Применим для любой сцены.
* Даже при сильных упрощениях даёт реалистичное изображение.
* Легко распараллеливается (каждому потоку своя группа пикселов, для каждого из которых отслеживаем луч)
* Легко устраняется лестничный эффект (для этого достаточно повысить дискретность распределения света)

# Современные технологии компьютерной графики

## 56. Основные идеи OpenGL и DirectX. Расширения OpenGL

**OpenGL** (*Open Graphics Library*) — открытая библиотека и API для работы с двумерной и трёхмерной графикой.

Основные принципы:

* независимость от платформы и языка
* возможность определения всех параметров сцены

Входные данные:

* векторная модель поверхностей
* параметры поверхностей
* параметры расположения камеры и освещения
* дополнительные параметры (например, для задания эффектов сцены)

Выходные данные:

* Растр

Организация:

* набор типов
* набор констант
* набор функций
* набор изменяемых состояний рендеринга
* набор команд

Использование (действия, выделенные жирным, могут повторяться в цикле):

1. Инициализация glInit()
2. **Очистка сцены glClear() (используется принцип z-буфера)**
3. **Определение параметров камеры и освещения**
4. **Задание примитивов**
5. **Генерация сцены:**
   * glFinish() — асинхронно
   * glFlush() — синхронно
6. Завершить работу glDone()

OpenGL реализуется для каждой платформы с помощью расширений:

* **WGL** — расширение OpenGL под Windows, позволяющее работать с HDC (*Handle for Device Context*). Работа с контекстом состоит из следующих этапов:
  + wglCreateContext()
  + wglMakeCurrent()
  + wglDeleteContext()
* **GLUT** — расширение OpenGL, подволяющее выводить изображение целиком в окно.

Команды:

* glClear() — сброс z-буфера / цвета
* glClearColor() — сброс с закраской окна заданным цветом
* glClearIndex() — очистка индексированных цветов, хранящихся в палитре
* glGetColor()
* glSetColor()

Задание примитивов:

glBegin(<команда, определяющая тип примитива>)

<установка параметров (цвет, эффекты)>

...

glVertex(<точка>);

glVertex(<точка>);

glVertex(<точка>);

...

glEnd();

Наборы команд можно сохранять в виде списков (*display lists*) и затем целиком выполнять:

glNewList()

<команды>

glEndList()

...

glCallList()

Работа со стеком атрибутов:

* glPushAttrib() — заталкивает текущие параметры в стек атрибутов
* glPopAttrib() — извлекает параметры из стека и применяет их

Работа с матрицей преобразований:

* glPushMatrix() — заталкивает текущую матрицу в стек
* glPopMatrix() — извлекает матирцу из стека и применяет её

Ещё что-то про

* glMatrixMode()
* glLoadIdentity() — загрузка единичной матрицы
* glLoadMatrix()
* glLoadTransposeMatrix()
* glTranslate() — сдвиг
* glRotate() — поворот
* glScale() — масштабирование

Камера:

* glLookAt()
* glFrustum() — перспективная проекция
* glOrtho() — параллельная проекция
* glViewPort() — определяет, какую часть плоскости проецирования отображать в растр

Освещение. Допускается максимум 8 источников освещения. Включение/выключение освещения производится с помощью процедур glEnable() / glDisable(). Настройка освещения — с помощью glLight() с различными параметрами:

* GL\_POSITION — положения источника
* GL\_SPOT\_DIRECTION — направление освещения
* GL\_CUT\_OFF — угол разброса лучей // meow

Виды освещения:

* GL\_AMBIENT — фоновое
* GL\_DIFFUSE — рассеянное
* GL\_SPECULAR — отражённое

Материалы: используется процедура glMaterial() с параметрами

* GL\_AMBIENT — фоновый цвет материала (?)
* GL\_DIFFUSE — рассеянный
* GL\_SPECULAR — зеркальный

Дополнительные параметры

* GL\_SHINESS — степень отражения
* GL\_EMISSION — настройка излучения

Наложение текстур:

* glTexImage1D() — загрузка одномерной текстуры
* glTexImage2D() — загрузка двумерной текстуры
* glTexCoord() — определяет координату привязки текстуры к объекту

Возможно также смешивание отрисовываемых примитивов с уже имеющимися на сцене. Цвет каждого пикселя определяется как

**R** = **RS** \* SR + **RD** \* DR

**G** = **GS** \* SG + **GD** \* DG

**B** = **BS** \* SB + **BD** \* DB

**A** = **AS** \* SA + **AD** \* DA

где

R,G,B,A — получаемый цвет

RS, GS, BS, AS — цвет точки рисуемого примитива

RD, GD, BD, AD — цвет точки сцены

SR, SG, SB, SA — коэффициенты для примитива

DR, DG, DB, DA — коэффициенты для сцены

Примеры:

* при S\*=1, D\*=0 — рисование поверх
* при S\*=0.5, D\*=0.5 — рисование с 50%-й прозрачностью
* при S\*=0, D\*=1 — никакого рисования

Кроме того, можно задавать функцию смешивания с помощью glBlendFunc().

**DirectX** — набор API от Microsoft, предназначенных для создания мультимедийных приложений:

* Direct3D — трёхмерная графика
* DirectAnimation — ??
* DirectDraw — интерфейс вывода растровой графики (его разработка давно прекращена)
* DirectInput — интерфейс, используемый для обработки данных, поступающих с клавиатуры, мыши, джойстика и пр. игровых контроллеров.
* DirectMusic — интерфейс воспроизведения музыки в форматах Microsoft
* DirectShow — интерфейс, используемый для ввода/вывода аудио и/или видео данных.

Поскольку эти API представлены в виде COM-объектов (в конечном счёте в виде exe- и dll-файлов), то DirectX можно применять при разработке приложений любых языках, поддерживающих COM-технологию.

Объект

* IUnknownInterface — базовый родительский интерфейс
  + AddRef() — увеличивает счётчик ссылок на объект
  + Release() — уменьшается счётчик ссылок и при необходимости уничтожает объект.
  + QueryInterface() — запрос любого другого интерфейса (?)
* IDispatch

Некоторые интерфейсы для работы с 3D-графикой:

* IDirect3D8 — отвечает, в частности, за инициализацию
* IDirect3DDevice8 — интерфейс устройства
* IDirect3DVertexBuffer — работа с буфером вершин
* IDIrect3DTexture8 — работа с текстурами

Работа с DirectX (действия, выделенные жирным, могут повторяться в цикле):

1. Инициализация Direct3DCreate8 (результат: ID3D8)
2. Создание графического устройства CreateDevice (результат: ID3DDevice). Необходимые параметры:
   * код устройства
   * режим графического ускорителя
   * взаимодействие с оконной системой / полный экран
   * количество фоновых буферов
3. **Очистка буферов ID3DDevice::Clear**
4. **Задание сцены**
5. **ID3DDevice8::Present**
6. Освобождение ресурсов Release

Задание сцены:

* IDirect3DVertexBuffer
* IDirect3DDevice::SetViewPort — преобразование мировых координат в экранные
* IDirect3DDevice::SetTransform — преобразование мировых координат в мировые
* IDirect3DDevice::SetLight
* IDirect3DDevice::LightEnable

## 57. Аппаратная архитектура графических ускорителей. Графический конвейер

**Графический ускоритель** — аппаратная реализация алгоритмов.

* Вход: положение объектов, камеры, освещение
* Выход: растр.

Работа графического конвейера:

* выборка, распаковка (координаты, α-канал, индекс-палитры и пр. приводятся к унифицированному формату)
* процессор вершин (преобразование мировых координат в экранные, заливка Гуро/Фонг)
* пост-буфер вершин (вычисление списка уникальных вершин, построение их триад - позволяет избежать повторных вычислений)
* установка треугольников (проверка видимости, отсечение)
* разбиение и установка фрагментов (определение нормали и текстуры для каждого треугольника, эффекты, интерполяция)
* процессор фрагментов (вычисление текстуры и итогового цвета для каждого пикселя)
* смешивание (*blending*)
* запись в кадр (формирование выходного буфера)

## 58. Особенности скоростной обработки данных в графических ускорителях

Особенности обработки:

* Векторные вычисления могут реализоваться аппаратно на уровне процессора и шины (4-х компонентная для графических данных)
* Вычисления независимы друг от друга (SIMD)
* Используется несколько АЛУ => несколько операций выполняются за один такт
* Ориентация на вычисления: так как проверки условий в основном дают одинаковый результат, то на несколько АЛУ выделяется одно управляющее устройство (проверяет условия)
* Потоковый характер вычислений, буферизация: данные идут в одном направлении, разделяются на потоки по типу информации и по направлению.

## 59. Шейдеры

**Шейдеры** — подпрограммы для собственной реализации процессоров вершин и фрагментов.

Графический конвейер не позволяет:

* создавать эффекты
* моделировать мелкие детали (микрорельеф)

Решения:

* Вершинный шейдер (процессор вершин)
  + Вход: вершины и их параметры
  + Выход: вершины в экранных координатах.
* Пиксельный шейдер (процессор фрагментов)
  + Вход: пиксель, его координаты, привязка к текстуре, нормаль и др.
  + Выход: цвет этого пикселя

Языки программирования шейдеров:

* DirectX asm
* GLSL (C-like for OpenGL)
* Cg (*C for Graphics*)
* HLSL (C-like; модифицированный Cg для DirectX)
  + Особенности:
    - Нет рекурсии
    - Нет передачи по ссылке
    - Все функции — [inline](http://en.wikipedia.org/wiki/Inline_function) (вызов функции компилятор заменяет телом функции)
    - Готовый шейдер можно загрузить, скомпилировать и отправить в конвейер
  + Модификаторы:
    - static (~ private)
    - uniform (~public)
    - const (~const)
    - shared (~ public volatile)
* Профессиональные
  + RenderMan
  + Gelato

Хозяйке на заметку:

* **Технология CUDA** позволяет выполнять неграфические вычисления с помощью графического процессора.

# Нефотореалистичная компьютерная графика

## 60. Цели и задачи нефотореалистичной компьютерной графики. Артефакты и их классификация

Причины создания НФР:

* человекорисунки приятнее на вид, чем точные модели
* их акценты человечнее

Цели НФР:

* имитация ИИ (а-ля тест Тьюринга в графике)
* исследование дизайна
* анализ восприятия (т.е. как прикладной инструмент для психологов)

**Артефакты** — отличия реального изображения от компьютерного (например, размытие, эффект Гало, нерезкость). Виды:

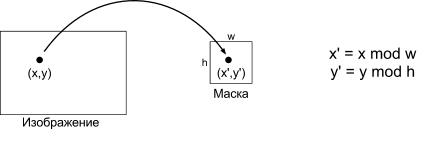
* артефакты модели (входных данных)
* артефакты алгоритма
* артефакты рисования графических примитивов
* артефакты закраски и текстурирования
* артефакты невидимых линий
* артефакты освещения
* артефакты проецирования

Цели внесения артефактов:

* сокрытие второстепенных деталей
* акцентирование важных деталей
* подчёркивание разницы (контрасты)

## 61. Методы маскирования изображений

Маскирование позволяет создавать артефакты на уровне пикселей. Суть маскирования:



Маска позволяет задать микрорисунок. Требования к маске:

* присутствие всех яркостей
* геометрическое распределение

## 62. Маскирование образами (текст, точка, линия)

В качестве маски можно использовать текст, точку, линию. Например рисунок на банкнотах.

## 63. Построение мозаик

Каждый фрагмент мозаики либо монотонен, либо сам является некоторым изображением.

Алгоритм:

1. Выбор примитивов (форма, размер, цвета, текстура; подбираем все элементы, которые могут понадобиться)
2. Выбор схемы размещения
   * Виды:
     + *Нерегулярные* (элементы могут накладываться друг на друга, возможны пустые места)
     + *Регулярные* (элементы не могут накладываться друг на друга)
   * Алгоритм:
     + Если участок изображения монотонен и на нём нет контуров, начинаем подбирать элемент мозаики для него (шаг 3)
     + В противном случае делим изображение на части и рекурсивно повторяем процедуру с каждой
3. Построение схемы размещения — выбираем наиболее подходящий элемент мозаики.
4. Пост-обработка (удаление лишних элементов, цветовая коррекция, размытие и др.)

## 64. Построение штриховых иллюстраций и текстур

Описываемые методы принимают на входе растр и обрабатывают его так, будто оно нарисовано карандашом: штрихи видно не только на микро-, но и на макроуровне.

* Изображение состоит из штрихов (нерегулярных отрезков переменной длины и направлений, но одного цвета и толщины)
* Яркость участков изображений определяется частотой штрихов
* Форма передаётся направлением штрихов
* Изображение текстур тоже передаётся с помощью штрихов
* Больше штрихов

Алгоритм:

1. Создаётся пустое изображение
2. Находится изображение-разность между текущим изображением и целевым (входным)
3. Находится точка максимального отклонения (с учётом размытия штрихов). При этом учитывается:
   * Направление, в котором возрастает разность яркости
   * Границы: если штрих пересекает некоторый объект, то его лучше направить вдоль границы
   * Нормали (в случае 3D): штрих должен лежать в плоскости нарисованной грани
   * ~~Положение звёзд~~ Случайный фактор
4. Определяется длина и направление штриха
5. Штрих рисуется и размывается
6. Переход к п.2, ~~пока не надоест~~. пока не достигнем нужной степени схожести или не исчерпаем время/количество штрихов

**<Нижеследующего, судя по лекциям, у нас не было, но на всякий случай пусть будет>**

Штриховка может использоваться для генерации текстур. При этом текстуру может потребоваться рисовать разной яркостью. При работе со штрихованными текстурами нельзя просто повысить/понизить их яркость.

Штрихованная текстура задаётся в виде упорядоченной последовательности штрихов. Для любого штриха хранится длина, направление и координата.

Сортировка штрихов:

В зависимости от требуемой яркости текстуры рисуются только первые *n* штрихов (чем ярче текстура, тем больше штрихов, и наоборот).

Штрихи делятся на два класса:

* штрихи, задающие форму текстуры;
* штрихи, задающие яркость текстуры.

Требуемая текстура задаётся двумя числами (M, N). Используем, соответственно, два списка (?). В зависимости от требуемой яркости рисуем заданное количество штрихов нужной формы и яркости.

Как и любые растровые изображения, штрихованные текстуры плохо масштабируются (например, если просто растянуть такую текстуру, то все штрихи станут толстыми), поэтому для каждого масштаба нужно честно строить свою текстуру.

**</Вышеизложенного, судя по лекциям, у нас не было, но на всякий случай пусть будет>**

## 65. Рисование реалистичных отрезков

При рисовании линии человеком возникают следующие артефакты:

* Волнистость
* Неточная длина
* Переменная толщина
* Переменная яркость

Алгоритм рисования отрезка использует дополнительные параметры:

* *Траектория линии* — определяется функцией F(t), показывающей отклонение от идеальной линии в перпендикулярном направлении, t ∈ [0,1]
* *Толщина линии* — определяется функцией W(t), t ∈ [0,1]
* *Яркость линии* — определяется функцией L(t), t ∈ [0,1]

Весь отрезок [0,1] разбивается на интервалы с некоторым шагом Δt и для каждого рисуется свой четырёхугольник.

## [ ХХХ ] 66. Имитация рисования кистью. Клетчатые автоматы

## [ ХХХ ] 67. Имитация рисования карандашом

## [ ХХХ ] 68. Имитация гравировки

## 69. Использование G-буферов. Примеры

**G-буфер** — расширенный буфер вывода изображений, в котором для каждого пикселя кроме его цвета может храниться дополнительная информация:

* значение α-канала
* цвет, заданный в другой модели (например, HSV)
* расстояние до камеры
* ID объекта, которому принадлежит точка
* ID грани, которой принадлежит точка
* ID материала
* текстурные координаты
* буфер теней
* вектор освещения
* информация о невидимых линиях
* координаты нормали

Эту информацию можно применять для создания дополнительных эффектов при отрисовке изображения:

* выделение поверхностей и объектов (используем ID объекта / грани)
* эффект тумана (меняем цвет пикселя в зависимости от расстояния до камеры)
* акцентирование границ объектов
* классификация и акцентирование рёбер (чем резче угол, тем больше акцент)
* подготовка данных для штриховки
* размытие тени
  + используется буфер, хранящий информацию о том, какие пиксели являются теневыми, а какие — нет
  + в нём находятся граничные точки
  + в этих точках проводится размытие по Гауссу

## [ ХХХ ] 70. Классификация рёбер и модифицированный алгоритм Робертса.

## [ ХХХ ] 71. Моделирование освещения в НФР.

## [ ХХХ ] 72. Искажение пространства в НФР.

## [ ХХХ ] 73. Искажение размеров объектов.

## 