1. **Цветовосприятие. Цвет и свет. Устройство человеческого глаза.**

Цвет существует, только если представлены три его компонента: зритель, предмет и освещение. Несмотря на то, что чисто белый свет воспринимается как бесцветный, в действительности он содержит все цвета видимого спектра. Когда белый свет достигает объекта, поверхность избирательно поглощает одни цвета и отражает другие; только отражённые цвета создают у зрителя восприятие цвета.

Человеческий глаз воспринимает этот спектр, используя для зрения комбинацию из клеток-палочек и клеток-колбочек. Палочки имеют более высокую светочувствительность, но различают только интенсивность света, тогда как колбочки могут также различать цвета, но лучше всего функционируют при ярком свете. В каждом глазе есть три типа колбочек, каждый из которых более чувствителен к коротким (К), средним (С) или длинным (Д) световым волнам. Комбинация сигналов, возможных во всех трёх колбочках, описывает диапазон цвета, который мы можем видеть.

Человеческое восприятие цвета максимально чувствительно к свету в жёлто-зелёном диапазоне спектра.

1. **Цветовые модели. Модель RGB.**

Для описания цветовых моделей существуют 2 системы цветности:

Аддитивная: аддитивный синтез цвета предполагает получение цвета смешением излучений. В аддитивном синтезе под белым цветом мы понимаем смешение основных излучений в максимальном количестве, а чёрный цвет - полное отсутствие излучений.

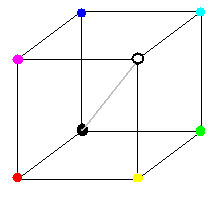
Субтрактивная: при субтрактивном синтезе компоненты излучения попадают в глаз не напрямую, а преобразуясь оптической средой - окрашенной поверхностью. Ее окраска выполняет функцию, преобразователя энергии излучения источника света. Отражаясь от нее или проходя насквозь, одни лучи ослабляются сильнее, другие слабее.

В основе одной из наиболее распространенных цветовых моделей, называемой RGB моделью, лежит воспроизведение любого цвета путем сложения трех основных цветов: красного (Red), зеленого (Green) и синего (Blue). Каждый канал - R, G или B имеется свой отдельный параметр, указывающий на количество соответствующей компоненты в конечном цвете.

Основные цвета разбиваются на оттенки по яркости (от темного к светлому), и каждой градации яркости присваивается цифровое значение (например, самой темной – 0, самой светлой – 255).

В модели RGB цвет можно представить в виде вектора в трехмерной системе координат с началом отсчета в точке (0,0,0). Максимальное значение каждой из компонент вектора примем за 1. Тогда вектор (1,1,1) соответствует белому цвету. Внутри полученного куба и «находятся» все цвета, образуя цветовое пространство.  
Важно отметить особенные точки и линии этой модели:

1. Начало координат: в этой точке все составляющие равны нулю, излучение отсутствует (черный цвет)
2. Точка, ближайшая к зрителю: в этой точке все составляющие имеют максимальное значение (белый цвет)
3. На линии, соединяющей предыдущие две точки (по диагонали), располагаются серые оттенки: от черного до белого (серая шкала, обычно — 256 градаций). Это происходит потому, что все три составляющих одинаковы и располагаются в диапазоне от нуля до максимального значения
4. Три вершины куба дают чистые исходные цвета, остальные три отражают двойные смешения исходных цветов.



1. **Цветовые модели. Модель CMY.**

Для описания цветовых моделей существуют 2 системы цветности:

Аддитивная: аддитивный синтез цвета предполагает получение цвета смешением излучений. В аддитивном синтезе под белым цветом мы понимаем смешение основных излучений в максимальном количестве, а чёрный цвет - полное отсутствие излучений.

Субтрактивная: при субтрактивном синтезе компоненты излучения попадают в глаз не напрямую, а преобразуясь оптической средой - окрашенной поверхностью. Ее окраска выполняет функцию, преобразователя энергии излучения источника света. Отражаясь от нее или проходя насквозь, одни лучи ослабляются сильнее, другие слабее.

Модель CMY использует также три основных цвета: Cyan (голубой), Magenta (пурпурный, или малиновый) и Yellow (желтый). Эти цвета описывают отраженный от белой бумаги свет трех основных цветов RGB модели. Формирование цвета происходит на белом фоне.

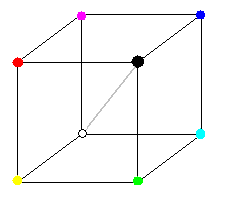
Например, соединение в равных долях всех трех красок CMY в одной точке приведет к тому, что весь белый свет не будет отражен, а, следовательно, цвет окажется черным. А вот одновременно и в равной пропорции, нанесенные всевозможные пары из тройки CMY дадут нам основные цвета RGB.

Цвета модели CMY являются дополнительными к цветам RGB. Дополнительный цвет - цвет, дополняющий данный до белого.

Так, например, дополнительный для красного цвета – голубой, для зеленого – пурпурный, для синего - желтый.

Особенные точки и линии модели:

1. Начало координат: при полном отсутствии краски (нулевые значения составляющих) получится белый цвет
2. Точка, ближайшая к зрителю: при смешении максимальных значений всех трех компонентов должен получиться черный цвет.
3. Линия, соединяющая предыдущие две точки (по диагонали). Смешение равных значений трех компонентов даст оттенки серого.
4. Три вершины куба дают чистые исходные цвета, остальные три отражают двойные смешения исходных цветов.



Цветовая модель CMY является основной в полиграфии. В цветных принтерах также применяется данная модель. Но для, что для того, чтобы распечатать чёрный цвет, необходимо большое количество краски и кроме того смешение всех цветов модели CMY на самом деле даст не чёрный, а грязно-коричневый цвет. Поэтому, для усовершенствования модели CMY, в неё был введён    дополнительный цвет - чёрный. Он является ключевым цветом при печати, поэтому последняя буква в названии модели - K (Key), а не B. Таким образом, модель CMYK является четырёхканальной. В этом заключается ещё одно отличие её от RGB.

1. **Цветовые модели. Модель LUV.**

Один из существенных минусов цветового пространства XYZ - это то, что оно не является визуально равномерным и не может использоваться для вычисления цветовых расстояний. Поэтому CIE продолжила разработку визуально равномерного пространства.

В результате было создано цветовое пространство CIE Luv, позволяющее определить различение цветов для человека с "усредненным" зрением, (т.е. различные люди неодинаково воспринимают разницу между цветами). Свое название пространство получило благодаря его компонентам L, u и v. Параметр L соответствует яркости цвета, u отвечает за переход от зеленого к красному (при увеличении), а при увеличении параметра v происходит переход от синего к фиолетовому. Если u и v равны 0, то, меняя L, получаем цвета, являющиеся градациями серого. Это цветовое пространство было разработано для количественного измерения различия двух цветов.

Немного о свойствах величин L, u, v:

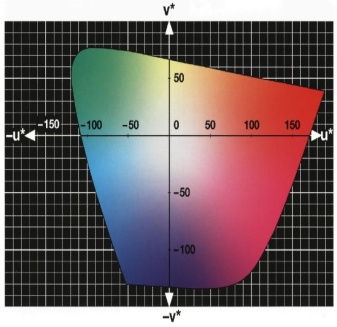
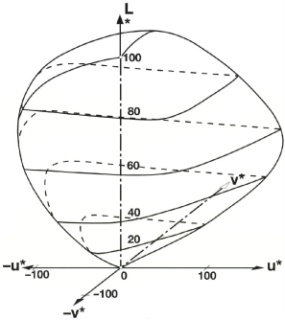
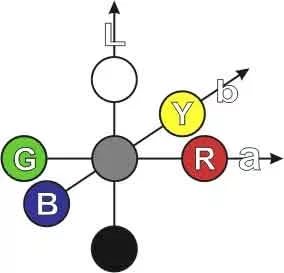
·L меняется от 0 до 100;

·u, v лежат в пределах -200, 200;

·u отвечает за переход от зеленого к красному (при увеличении u);

·v отвечает за переход от синего к фиолетовому (при увеличении v);

·если u и v равны 0, меняя L, получаем изображение, содержащее градации серого (grayscale).

1. **Цветовые модели. Модель HSV.**

HSV ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Hue, Saturation, Value — тон, насыщенность, значение) или HSB ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Hue, Saturation, Brightness — тон, насыщенность, яркость) — [цветовая модель](https://ru.wikipedia.org/wiki/%D0%A6%D0%B2%D0%B5%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C), в которой [координатами](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82%D0%B0) [цвета](https://ru.wikipedia.org/wiki/%D0%A6%D0%B2%D0%B5%D1%82) являются:

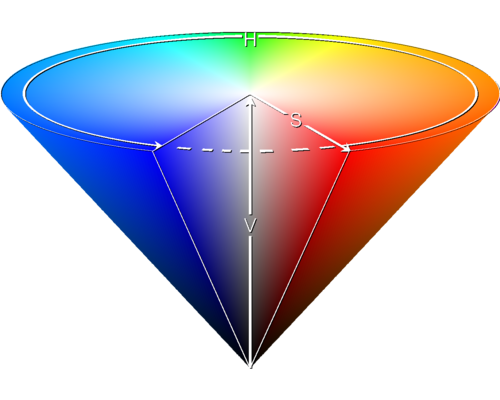
Шкала оттенков — Hue

Hue — цветовой [тон](https://ru.wikipedia.org/wiki/%D0%A2%D0%BE%D0%BD_(%D1%86%D0%B2%D0%B5%D1%82))(например, [красный](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B0%D1%81%D0%BD%D1%8B%D0%B9_%D1%86%D0%B2%D0%B5%D1%82), [зелёный](https://ru.wikipedia.org/wiki/%D0%97%D0%B5%D0%BB%D1%91%D0%BD%D1%8B%D0%B9_%D1%86%D0%B2%D0%B5%D1%82) или сине-голубой). Варьируется в пределах 0—360°, однако иногда приводится к диапазону 0—100 или 0—1.

Saturation — [насыщенность](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D1%8B%D1%89%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C_(%D1%86%D0%B2%D0%B5%D1%82)). Варьируется в пределах 0—100 или 0—1. Чем больше этот параметр, тем «чище» цвет, поэтому этот параметр иногда называют [чистотой цвета](https://ru.wikipedia.org/w/index.php?title=%D0%A7%D0%B8%D1%81%D1%82%D0%BE%D1%82%D0%B0_%D1%86%D0%B2%D0%B5%D1%82%D0%B0&action=edit&redlink=1). А чем ближе этот параметр к [нулю](https://ru.wikipedia.org/wiki/0_(%D1%87%D0%B8%D1%81%D0%BB%D0%BE)), тем ближе цвет к нейтральному [серому](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D1%8B%D0%B9_%D1%86%D0%B2%D0%B5%D1%82).

Value (значение цвета) или Brightness — яркость. Также задаётся в пределах 0—100 или 0—1.

Способ визуализации цветового пространства — [конус](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%83%D1%81). Как и в цилиндре, оттенок изменяется по окружности конуса. Насыщенность цвета возрастает с отдалением от [оси](https://ru.wikipedia.org/wiki/%D0%9E%D1%81%D1%8C) конуса, а яркость — с приближением к его основанию. Иногда вместо конуса используется шестиугольная правильная [пирамида](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%80%D0%B0%D0%BC%D0%B8%D0%B4%D0%B0_(%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F)).



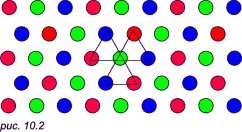
1. **Векторные и растровые дисплеи. Растровая развертка.**

Векторные дисплеи. Они состоят из дисплейного процессора, дисплейной буферной памяти и ЭЛТ (электронно-лучевая трубка). Буфер служит для запоминания подготовленного на ЭВМ дисплейного списка (или дисплейной программы); последняя включает команды вывода точек и отрезков (по координатам конечных точек), а также команды вывода литер. Команды рисования точек, отрезков и литер интерпретируются дисплейным процессором, который преобразует цифровые значения в аналоговые напряжения, управляющие электронным лучом, последний вычерчивает линии на люминофором покрытии ЭЛТ.

Растровые дисплеи. В растровых дисплеях дисплейные примитивы (отрезки, литеры и закрашенные участки — обычно многоугольники) хранятся в памяти для регенерации в виде совокупности образующих их точек, называемых пикселями. Пиксель – наименьший элемент изображения, который может изменить яркость или цвет. Изображение формируется на растре, представляющем собой совокупность горизонтальных растровых строк, каждая из которых состоит из отдельных пикселов; таким образом, растр — это матрица из пикселов, покрывающая всю площадь экрана. Все изображение последовательно сканируется по отдельным строкам растра в направлении сверху вниз, при этом изменяется лишь интенсивность электронного луча для каждого пикселя в строке. Этот процесс сканирования называется **растровой разверткой.**

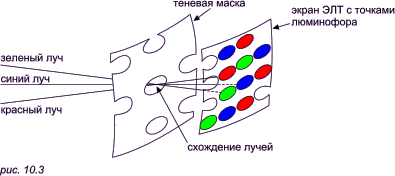
1. **Растровые дисплеи на электронно-лучевой трубке.**

Устройство цветной растровой ЭЛТ - в ней находятся три электронные пушки, по одной на каждый основной цвет: красный, зеленый и синий. Электронные пушки часто объединены в треугольный блок, соответствующий подобному треугольному блоку точек красного (R — red), зеленого (G — green) и синего (B — blue), люминофоров на экране ЭЛТ (рис. 10.2).



Для того, чтобы электронные пушки возбуждали только соответствующие им точки люминофора (например, красная пушка возбуждала только точку красного люминофора), между электронными пушками и поверхностью экрана помещена металлическая решетка.

Это так называемая теневая маска стандартной цветной ЭЛТ с теневой маской. Отверстия в ней образуют такие же треугольные блоки, как и точки люминофора. Цветовые пушки расположены таким образом, что их лучи сходятся и пересекаются в плоскости теневой маски (рис. 10.3). После прохождения через отверстие красный луч, например, защищен или маскирован от пересечения с зеленой или синей точкой люминофора. Он может пересечь лишь красную точку. Изменяя интенсивность электронного луча для каждого основного цвета, можно получить различные оттенки. Комбинация этих оттенков дает большое количество цветов для каждого пикселя. Обычно в дисплее с высоким разрешением на каждый пиксель приходится от двух до трех цветовых триад.



1. **Растровые дисплеи на газоразрядной трубке.**

Устройство и принцип работы газоразрядных дисплеев

Плазменная панель представляет собой матрицу газонаполненных ячеек, заключенных между двумя параллельными стеклянными пластинами, внутри которых расположены прозрачные электроды, образующие соответственно шины сканирования, подсветки и адресации. Разряд в газе протекает между разрядными электродами на лицевой стороне экрана и электродом адресации на задней стороне.

Принцип действия монитора основан на плазменной технологии: используется эффект свечения инертного газа под воздействием электричества

Работа плазменной панели состоит из трех этапов:

1. инициализация, в ходе которой происходит упорядочивание положения зарядов среды и её подготовка к следующему этапу (адресации

2. адресация, в ходе которой происходит подготовка пикселя к подсвечиванию. На шину адресации подается положительный импульс, а на шину сканирования отрицательный. На шине подсветки напряжение устанавливается равным +150 В.

3. подсветка, в ходе которой на шину сканирования подается положительный, а на шину подсветки отрицательный импульс. Это приводит к разряду в газовой среде. После разряда происходит повторное распределение ионов у шин сканирования и подсветки. Смена полярности импульсов приводит к повторному разряду в плазме. Таким образом, меняя полярность импульсов обеспечивается многократный разряд ячейки.

Один цикл «инициализация -- адресация -- подсветка» образует формирование одного подполя изображения. Складывая несколько подполей можно обеспечивать изображение заданной яркости и контраста. В стандартном исполнении каждый кадр плазменной панели формируется сложением восьми подполей.

Главными недостатками такого типа мониторов является довольно высокая потребляемая мощность, возрастающая при увеличении диагонали монитора и низкая разрешающая способность, обусловленная большим размером элемента изображения.

1. **Растровые дисплеи на жидких кристаллах.**

Экран жидкокристаллического дисплея (ЖКД) состоит из двух стеклянных пластин, между которыми находится масса, содержащая жидкие кристаллы, которые изменяют свои оптические свойства в зависимости от прилагаемого электрического заряда. Жидкие кристаллы сами не светятся, поэтому ЖКД нуждаются в подсветке или во внешнем освещении. Основным достоинством ЖКД являются их габариты (экран плоский). К недостаткам можно отнести недостаточное быстродействие при изменении изображения на экране, что особенно заметно при перемещении курсора мыши, а также зависимость резкости и яркости изображения от угла зрения.

Конструктивно дисплей состоит из следующих элементов:

1. ЖК-матрицы (первоначально — плоский пакет стеклянных пластин, между слоями которого и располагаются жидкие кристаллы
2. источников света для [подсветки](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D1%81%D0%B2%D0%B5%D1%82%D0%BA%D0%B0_%D0%96%D0%9A-%D0%B4%D0%B8%D1%81%D0%BF%D0%BB%D0%B5%D0%B5%D0%B2);
3. контактного [жгута](https://ru.wikipedia.org/wiki/%D0%96%D0%B3%D1%83%D1%82) (проводов);
4. корпуса, чаще [пластикового](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D1%81%D1%82%D0%B8%D0%BA), с металлической рамкой для придания жёсткости.

Проходящий через ячейки свет может быть естественным  (в ЖК-дисплеях без подсветки). Но чаще применяют [искусственный источник света](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B8%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D1%81%D0%B2%D0%B5%D1%82%D0%B0), кроме независимости от внешнего освещения, это также стабилизирует свойства полученного изображения.

1. **Трассировка и растеризация.**

**Трассировка** — преобразование изображения из [растрового](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) представления в [векторное](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0). Процесс, обратный [растеризации](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F).

Программа трассировки анализирует цвета пикселов исходного изображения, затем распределяет пиксели между заданным количеством групп с тем, чтобы цвета в каждой из них были близки друг к другу. Для каждой из группы определяется наиболее характерный цвет. После этого определяются границы области (или областей), составленных из пикселов каждой группы. Наконец, рисуются многоугольники с найденными границами, целиком закрашенные соответствующим характерным цветом.

**Растеризация** - отображения некоторого графического объекта на экране растрового монитора будет сводиться к задаче поиска ограниченного набора растровых точек, наилучшим образом аппроксимирующих (самые приближенные) изображение (преобразование изображения из [векторного](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) представления в [растровое)](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0).

Изображение как объект данных принято представлять с помощью одной из двух моделей: растровой или векторной:

**Растровая модель** - рассматривает изображение как прямоугольный массив пикселов. Таким образом, растровое изображение должно содержать информацию о размерах картинки и перечисление цветовых значений всех составляющих её пикселов. В качестве цветового значения часто применяют тройку чисел R, G, B, заключённых в диапазоне от 0 до 255. Если же количество различных цветов, встречающихся у пикселов изображения, невелико, можно включить в модель изображения палитру — таблицу всех встречающихся цветов, а в качестве цветового значения пиксела взять индекс его цвета в таблице.

Растровые изображения плохо переносят даже простые геометрические преобразования, такие как сдвиги, масштабирование и поворот.

**Векторная модель** - изображение считается сформированным из графических примитивов — линий и закрашенных простых фигур. Каждая фигура моделируется как набор свойств, которые отвечают за её форму, положение, цвет и другие вещи. В многоугольнике, к примеру, за форму и положение на картинке отвечает список координатных пар его вершин.

Векторная графика, в отличие от растровой, отлично выдерживает геометрические преобразования — достаточно подвергнуть преобразованиям координаты характерных точек у каждого графического примитива.

**11. Растеризация отрезков. Алгоритм ЦДА.**

Отрезок является простейшим геометрическим примитивом (после точки), задача отображения которого на экране является задачей растеризации (подбор точек, наиболее точно аппроксимирующих заданный отрезок). В дальнейшем под отрезком будет понимать ограниченный сегмент прямой линии, задаваемый координатами своих концов.

Пусть отрезок задан вещественными координатами концов ( x 1 , y 1 ) {\displaystyle (x\_{1},y\_{1})} (x1,y1;( x 2 , y 2 ) {\displaystyle (x\_{2},y\_{2})} (x2,y2). Растровыми (целочисленными) координатами концевых точек становятся округлённые значения исходных координат: *x s t a r t = round ⁡ ( x 1 ) {\displaystyle x\_{\mathrm {start} }=\operatorname {round} (x\_{1})} xstart = round(x1)*, *xend = round(x2)y s t a r t = round ⁡ ( y 1 ) {\displaystyle y\_{\mathrm {start} }=\operatorname {round} (y\_{1})} ; yx s t a r t = round ⁡ ( x 1 ) {\displaystyle x\_{\mathrm {start} }=\operatorname {round} (x\_{1})} yyyystart = round(y1), yend = round(y2)y s t a r t = round ⁡ ( y 1 ) {\displaystyle y\_{\mathrm {start} }=\operatorname {round} (y\_{1})} x e n d = round ⁡ ( x 2 ) {\displaystyle x\_{\mathrm {end} }=\operatorname {round} (x\_{2})} y e n d = round ⁡ ( y 2 ) {\displaystyle y\_{\mathrm {end} }=\operatorname {round} (y\_{2})}.*

Большее по абсолютной величине число, *(xend  - xstart)* ( x e n d − x s t a r t ) {\displaystyle (x\_{\mathrm {end} }-x\_{\mathrm {start} })} или *(yend – ystart)( y e n d − y s t a r t ) {\displaystyle (y\_{\mathrm {end} }-y\_{\mathrm {start} })} ,* увеличенное на 1 принимается за количество шагов L {\displaystyle L} L цикла растеризации.

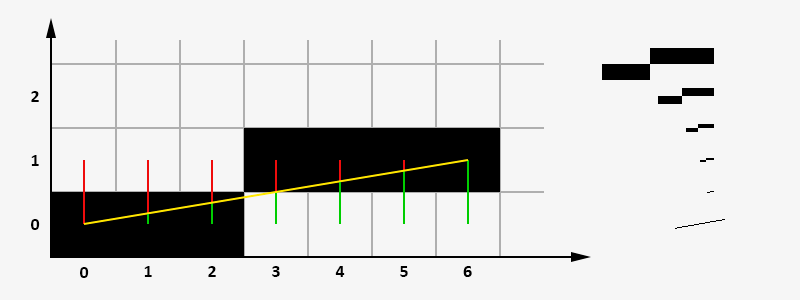
В начале цикла вспомогательным вещественным переменным *x {\displaystyle x} x* и *y {\displaystyle y} y* присваиваются исходные координаты начала отрезка*: x = x 1 {\displaystyle x=x\_{1}} x = x1 ; y = y 1 {\displaystyle y=y\_{1}} y = y1*. На каждом шаге цикла эти *вещественные* переменные получают приращения *(xend  - xstart)/L ; ( x e n d − x s t a r t ) {\displaystyle (x\_{\mathrm {end} }-x\_{\mathrm {start} })} $(yend – ystart)/L* ( y e n d − y s t a r t ) {\displaystyle (y\_{\mathrm {end} }-y\_{\mathrm {start} })} ( x e n d − x s t a r t ) / L {\displaystyle (x\_{\mathrm {end} }-x\_{\mathrm {start} })/L} ( y e n d − y s t a r t ) / L {\displaystyle (y\_{\mathrm {end} }-y\_{\mathrm {start} })/L} . *Растровые* же координаты, продуцируемые на каждом шаге, являются результатом округления соответствующих вещественных значений *x {\displaystyle x} x* и *y {\displaystyle y} y*.

**12. Растеризация отрезков. Алгоритм Брейзенхема.**

Отрезок является простейшим геометрическим примитивом (после точки), задача отображения которого на экране является задачей растеризации (подбор точек, наиболее точно аппроксимирующих заданный отрезок). В дальнейшем под отрезком будет понимать ограниченный сегмент прямой линии, задаваемый координатами своих концов.

**Принцип работы алгоритма**. Берётся отрезок и его начальная координата *x*. К x в цикле прибавляем по 1 в сторону конца отрезка. На каждом шаге вычисляется ошибка — расстояние между реальной координатой *y* в этом месте и ближайшей ячейкой сетки. Если ошибка не превышает половину высоты ячейки, то она заполняется.

Сначала вычисляется угловой коэффициент *(y1 — у0)/(x1 — x0)*. Значение ошибки в начальной точке отрезка *(0,0)* принимается равным нулю, и первая ячейка заполняется. На следующем шаге к ошибке прибавляется угловой коэффициент и анализируется её значение, если ошибка меньше *0.5*, то заполняется ячейка *(x0+1, у0)*, если больше, то заполняется ячейка *(x0+1, у0+1)* и из значения ошибки вычитается 1.



**13. Растеризация площадных примитивов. Алгоритмы с затравкой.**

В связи с понятием растеризации площадного примитива обычно рассматривают два случая:

- растеризация многоугольника, задаваемого своими вершинами.

- заполнение некоторой замкнутой области на растре.

Методы затравочного заполнения исходят из того, что задана некоторая точка (затравка) внутри контура и задан критерий принадлежности точки границе области (например, задан цвет границы). В алгоритмах ищут точки, соседние с затравочной и расположенные внутри контура. Если обнаружена соседняя точка, принадлежащая внутренней области контура, то она становится затравочной и поиск продолжается рекурсивно.

Поместить затравочный пиксель в стек

Пока стек не пуст:

Извлечь пиксель из стека

Инициализировать пиксель

Для каждого из четырех соседних пикселей:

Проверить, является ли он граничным и был ли он инициализирован

Если нет, то поместить пиксель в стек

**14. Растеризация площадных примитивов. Алгоритмы со списком ребер.**

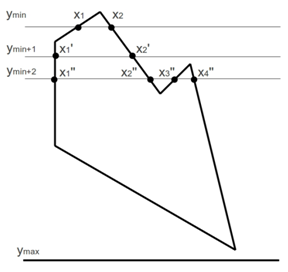
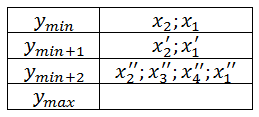
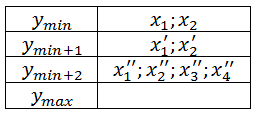
В связи с понятием растеризации площадного примитива обычно рассматривают два случая:

- растеризация многоугольника, задаваемого своими вершинами.

- заполнение некоторой замкнутой области на растре.

***Алгоритмы со списком рёберных точек***  
Этот алгоритм подходит для тех случаев, когда закрашиваемая область может быть задана в виде многоугольника. Каждое ребро многоугольника задаётся координатами его концов *x1,y1* и *x2,y2* (при этом *y2≥y1*). Условимся, что *x* возрастает слева направо, а *y* сверху вниз.  
Подавляющее большинство алгоритмов растеризации многоугольников основаны на предположении, что любая секущая пересекает границу многоугольника чётное число раз. Это утверждение неверно только в двух случаях:

* Когда секущая прямая содержит ребро;
* Когда секущая прямая содержит вершину, а смежные рёбра лежат по одну сторону от секущей прямой.

Эти два случая довольно легко обнаружить, поэтому при рассмотрении алгоритмов растеризации многоугольников будем считать, что приведённое выше предположение всегда верно.  
[](https://habrastorage.org/storage/habraeffect/a6/f0/a6f073d6507a85b800ab5cadf6043660.png)  
Алгоритм, основанный на работе со списком рёберных точек, состоит из трёх основных этапов:  
На первом этапе растеризуются все негоризонтальные рёбра многоугольника. Для каждого значения y составляется список *x*-координат, закрашенных при растеризации. для рассматриваемого многоугольника получим (при движении по часовой стрелке):  
[](https://habrastorage.org/storage/habraeffect/c5/3f/c53f083abc78dd596f6c38e75005da1d.PNG)  
На втором этапе для каждого значения *y* списки упорядочиваются по возрастанию. списки будут выглядеть следующим образом:  
[](https://habrastorage.org/storage/habraeffect/90/61/9061a8e2d72890b6f609b2f2576fec29.PNG)  
На третьем этапе в каждой строке заполняются отрезки вида [x2i-1, x2i].

Преимущество этого алгоритма в том, что каждый пиксель обрабатывается строго один раз.

Существует модификация данного алгоритма, оптимизированная по расходу памяти. В ней на каждом шаге обрабатываются только те рёбра, которые пересекаются с текущей линией развёртки.

**Алгоритм со списком активных ребер (видоизменение предыдущего)**

Вместо того чтобы хранить в памяти точки пересечения контура с каждой строкой растра, ограничимся лишь текущей строкой. организуем список "активных" ребер (САР), в котором будем хранить информацию о ребрах, пересекаемых текущей строкой. Удобство в том, что при переходе к новой строке достаточно лишь удалить "закончившиеся" ребра и добавить вновь появившиеся.

**15. Методы антиалайсинга.**

*Сглаживание (англ. anti-aliasing) — технология, используемая для устранения эффекта «зубчатости», возникающего на краях одновременно выводимого на экран множества отдельных друг от друга плоских или объёмных изображений.*  
Основной принцип сглаживания — использование возможностей устройства вывода для показа оттенков цвета, которым нарисована кривая. В этом случае пиксели, соседние с граничным пикселем изображения, принимают промежуточное значение между цветом изображения и цветом фона, создавая градиент и размывая границу.

**Типы сглаживания**  
*SSAA (Supersample anti-aliasing,* избыточная выборка сглаживания) — самое тяжелое сглаживание. При четырехкратном (4х) сглаживании видеокарта готовит картинку в разрешении вчетверо выше, чем выводит на экран, потом происходит усреднение цвета соседних пикселей и вывод на экран в исходном разрешении. Получается, что виртуальная плотность пикселей вдвое выше, чем у экрана, и лесенки практически перестают быть заметными. Очень сильно сказывается на производительности. Поэтому начиная с DirectX 8 было заменено всеми производителями графических процессоров на MSAA.

*MSAA (Multisample anti-aliasing)* — улучшенная версия SSAA, которая потребляет гораздо меньше ресурсов. К примеру — зачем сглаживать то, что находится внутри текстуры, если лесенки есть только на краях? Если текстура представляет собой прямую линию под углом к игроку, то можно сгладить лишь один участок и продолжить эффект на весь край текстуры. В результате нагрузка на видеокарту становится ощутимо меньше, и по тяжести даже 8х MSAA оказывается ощутимо легче 4х SSAA при сравнимом качестве картинки.

*CSAA и CFAA (Coverage Sampling anti-aliasing и Custom-filter anti-aliasing)* —  по сути несколько улучшенный MSAA (позволяют выбирать дополнительные отсчёты «перекрытия» пикселя, по которым можно уточнять итоговое значение цвета попадающего на край треугольника экранного пикселя). 8x CSAA/CFAA дает сравнимое с 8x MSAA качество картинки, однако потребляет примерно столько же ресурсов, столько и 4х MSAA. На сегодняшний момент оба сглаживания не используются — разработчики игр решили использовать унифицированные для всех видеокарт сглаживания.   
  
*FXAA (Fast approXimate anti-aliasing)* — нетребовательное быстрое сглаживание. Алгоритм прост — совершается один проход по всем пикселям изображения и усредняются цвета соседних пикселей. Это слабо нагружает видеокарту, однако сильно мылит картинку, делая далекие объекты вообще неузнаваемыми

*MLAA (MorphoLogical anti-aliasing)* — аналог FXAA от Intel. Работает схожим образом, однако алгоритм сложнее — все изображение разбивается на Z, L и U -образные части, и сглаживание происходит смешением цветов пикселей, входящих в каждую такую часть.

Это единственное сглаживание, работающее полностью на процессоре, поэтому практически не влияет на fps в играх при мощном процессоре. Из-за более сложного алгоритма изображение получается более качественным, чем с FXAA, однако до 2x MSAA все еще далеко.

*SMAA (Subpixel Morphological anti-aliasing)* — смесь FXAA и MLAA. По сути несколько улучшенный MLAA, но работающий на видеокарте (так как процессор для сглаживания подходит гораздо хуже). Дает картинку, сравнимую с MLAA, лучше, чем FXAA, однако потребляет больше ресурсов

**16. Проецирование.**

Проецирование– процесс получения изображения предмета на какой-либо поверхности

Проецирование некоторой точки осуществляется при помощи *плоскости проекции*, на которой определена система координат, в которой окажется спроецированная точка, а так же *проектора* – луч, испущенный из проецируемой точки, который при пересечении с плоскостью проекции (или *плоскостью проецирования*) даст конечный результат.

Известны два метода проецирования: **центральное и** **параллельное**.

В центральных проекциях все проекторы сходятся в единой точке, называемой *точкой схода*. Разумеется, что угол, под которым они пересекают плоскость проекции различен.

В параллельных проекциях все проекты направлены параллельно. Они входят в плоскость проекции под одним и тем же углом, а точка схода в таком случае может рассматриваться как максимально удаленная.

В случае параллельной проекции, когда угол между проекторами и плоскостью проекции составляет 90 градусов, такой вид проекции называется *прямоугольной проекцией*. Иначе – *косоугольной*.  
Пытаясь растеризовать некоторый трехмерный объект на плоском экране можно пойти двумя путями:

1. Разработать алгоритмы растеризации объекта в трехмерном пространстве, затем для каждой полученной трехмерной точки произвести проецирование. Данный способ нетривиален и разрешение задачи в таком случае становится крайне неочевидно

2. Описать объект посредством некоторых опорных вершин, затем, применяя *только* линейные преобразования спроецировать эти вершины на плоскость с последующей растеризацией объекта, построенного на базе этих вершин.

**Точка** является простейшим геометрическим примитивом

**Отрезок** Для проецирования отрезка необходимо найти проекции его концов, которые будут использоваться при его растеризации.

Список отрезков (lint list) и ломаная (line strip).

**Многоугольник** Для проецирования многоугольника на экран достаточно спроецировать каждую из его вершин

**17. Математические преобразования. Матрицы преобразований.**

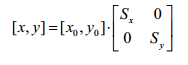
Так как каждый объект, двумерный, либо трехмерный можно описать как совокупность опорных вершин, то само преобразование будет сведено к преобразованиям над этими вершинами. Преобразования, производимые над объектом, будем считать *линейными*, либо *аффинными*. Для этих преобразований действительны свойства:  
- прямые остаются прямыми;  
- плоскости остаются плоскостями;  
- параллельные прямые и параллельные плоскости остаются параллельными;  
- отношения отрезков и площадей сохраняются;

**Сдвиг**Сдвиг (перенос) представляет собой приращение координат на некоторую величину

Математически его можно представить в виде:

*x = x0 +T y = y0 +T z = z0 +T*или в векторной форме:  
*P = P0 +T* , где *T* - вектор переноса

**Масштаб**Масштаб есть домножение соответствующей координаты на некоторое число. Масштаб  
*всегда* производится относительно начала координат.  
Математически операция масштаба эквивалентна:

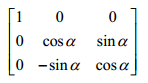
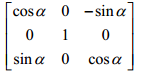
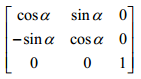
*x = x0 \*S y = y0 \*S z = z0 \*S*или в векторной форме:  
*P* = *P*0 \*S , где S- коэффициент масштабирования  
формуле выше речь идет о покомпонентном умножении двух векторов Так же масштаб может быть записан в матричной форме: 

**Поворот в плоскости**

осуществляется относительно начала координат  
Математически он может быть представлен:  
В скалярной форме:   
*x = x0cosa-y0sina  
y = x0sina-y0cosa*  
или в матричной:



где α - угол поворота, отсчитываемый против часовой стрелки.

**Поворот в пространстве**В отличие от поворота в плоскости, поворот в пространстве представляет собой уже не движение по дуге, а движение по сфере. В этом случае сам поворот становится не очевиден. Для разрешения неопределенности принято производить поворот в пространстве относительно некоторой оси, в качестве которой может быть выбрана *OX*, *OY* или *OZ*.  
Преобразования вращения вокруг осей могут быть выведены из случая для плоскости с фиксацией одной из осей. Далее перечислены матрицы для вращения:  
*OX*: *OY*:*OZ:*  
  
Для совершения поворота относительно произвольно направленной оси необходимо сначала произвести совмещение оси с одной из осей координат, затем повернуть на необходимый угол, после чего произвести преобразования обратные смещению.

**18. Композиция преобразований. Конвейер преобразований.**

Результатом каждого произведенного преобразования является произведение вектора и матрицы, то есть так же вектор. Данный вектор может быть снова использован для проведения преобразований, и так далее. Таким образом, будет справедливо следующее:  
*P1 =P0 M1 P2 =P1 M2 P3 =P2 M3*  
В то же время в силу дистрибутивности операции умножения матриц имеем:  
*P3* = ((*P0* ⋅ *M*1) ⋅ *M*2) ⋅ *M*3 = *P*0 ⋅ (*M*1 ⋅ *M*2 ⋅ *M3* )  
Данное выражение говорит о том, что все преобразования, которые мы собираемся произвести над объектом, могут быть сведены к умножению всего одного вектора на одну матрицу, если предварительно все матрицы преобразования были последовательно перемножены.

композиция преобразований позволяет не сохранять все произведенные над объектом преобразования, а накапливать их в одной единственной матрице.

**Конвейер преобразований**Для современных приложений выработался единый подход в описании систем координат и процессов перехода между ними. Так на сегодняшний день практически все средства визуализации рассматривают пространство и производимые преобразования над визуализируемыми объектами как последовательность, состоящую из трех (как вариант двух – с некоторым объединением) преобразований.  
**Модельная (мировая) матрица**Основная задача модельной матрицы - сэкономить память для описания одинаковых объектов, присутствующих в различных частях сцены. Действительно, многие приложения реального времени, такие, как компьютерные игры, зачастую вынуждены использовать множество одинаковых по своей структуре полигональных объектов. При этом, с одной стороны, дублирование информации могло бы оказаться весьма  
избыточным. С другой – каждый объект характеризуется уникальной позицией и ориентацией в пространстве. Для разрешения именно этой ситуации вводится модельная матрица (модельное (мировое) преобразование). Изначально объект описывается в некой исходной позе – в центре своей собственной системы координат. Модельная матрица же представляет собой композицию преобразований, необходимых для того, чтобы позиционировать клонированный объект в конечную точку с заданной ориентацией и масштабом. Таким образом, конечное положение i-ой вершины может быть вычислено как произведение исходной i-ой вершины и модельной матрицы. Необходимости хранить все преобразованные вершины нет. Они могут вычисляться непосредственно перед визуализацией. Существует так же и иной подход к модельной матрице. Фактически, модельная матрица описывает не только композицию преобразований, но и положение, а также ориентацию, системы координат исходного объекта для данного клона. Она используется для того, чтобы вынести координаты конкретной вершины из исходного состояния – локальной системы координат в глобальную.  
**Видовая матрица**Матрица вида на логическом уровне ассоциируется с местоположением в пространстве и ориентацией некоторого *наблюдателя*. Как известно, наблюдатель может перемещаться по сцене абсолютно независимо от перемещения располагающихся на сцене объектов. Если преобразования, связанные с определением ориентации каждого объекта относительно наблюдателя сохранять в модельные матрицы каждого объекта, то при перемещении самого наблюдателя матрицы пришлось бы пересчитывать, даже не смотря на тот факт, что сами объекты не претерпели никаких изменений. Для обеспечения независимых изменений в различные системы координат вводится  
вторая матрица – видовая. Для получения координаты точки на экране относительно наблюдателя необходимо внести ее координаты в локальную систему его координат. Поэтому, хотя матрица вида и  
описывает положение и ориентацию наблюдателя, на практике используется обратная ей матрица – осуществляющая преобразование внесения точки в систему координат.  
**Матрица проекции**Процесс проецирования так же можно описать при помощи матриц преобразования. Причем матрицы достаточно гибко позволяют настраивать сам процесс, предлагая реализации как для прямоугольных, так и для перспективных проекций. Матрица проецирования является последней, замыкающей конвейер преобразований. Фактически она описывает само преобразование проекции, т.е. получения координат  
пересечения проектора точки и плоскости проецирования. Данное преобразование получить не сложно, так как нам уже известны координаты точки в локальной системе наблюдателя, а также расстояние до плоскости проецирования, равно как и ряд других параметров.

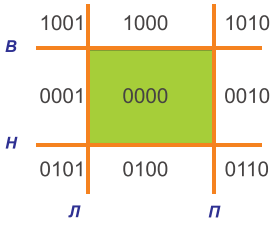
**19. Отсечения. Метод и алгоритм Коэна и Сазерленда.**

**Алгоритм Коэна — Сазерленда** — алгоритм отсечения отрезков, то есть алгоритм, позволяющий определить часть отрезка, которая пересекает прямоугольник.

• Вводится четырехразрядный код.  
• Крайний правый бит кода считается первым.  
• В соответствующий бит заносится 1

– для 1 бита - если точка левее окна;  
– для 2 бита - если точка правее окна;  
– для 3 бита - если точка ниже окна;  
– для 4 бита - если точка выше окна;

• В противном случае в бит заносится 0.

****

3 случая:

1. Если сумма кодов = 0, то обе точки лежат внутри окна, и отрезок видимый.
2. Если побитовое произведение кодов не равно нулю, то отрезок полностью невидим.
3. Если побитовое произведение = 0 и сумма ≠ 0, то отрезок не является тривиально невидимым.

Алгоритм определяет код концов отрезка. Если оба кода равны нулю, то отрезок полностью находится в прямоугольнике. Если *битовое И* кодов не равно нулю, то отрезок не пересекает прямоугольник (так как это значит, что оба конца отрезка находятся с одной стороны прямоугольника). В прочих случаях, алгоритм выбирает конец отрезка (или один из концов), имеющий ненулевой код (то есть находящийся вне прямоугольника), находит ближайшую к нему точку пересечения отрезка с одной из прямых, образующих стороны прямоугольника, и использует эту точку пересечения как новый конец отрезка. Укороченный отрезок снова пропускается через алгоритм.

• Уравнение бесконечной прямой, проходящей через точки:  
*P1(x1,y1)* и *P2(x2,y2)*, имеет вид  
*y = m( x - x1 ) + y1* или  
*y = m( x - x2 ) + y2*, где  
*m = (y2- y1) / (x2 - x1) -* наклон данной прямой.  
• Точки пересечения этой прямой со сторонами окна имеют координаты:

|  |  |  |
| --- | --- | --- |
| с левой: | *xл , y = m(xл - x1) + y1* | *m*≠∞ |
| с прав.: | *xп , y = m(xп - x1) + y1* | *m*≠∞ |
| с верхн.: | *yв , x = x1+(1/m)(yв - y1 )* | *m*≠0 |
| с нижн.: | *yн , x = x1+(1/m)(yн - y1)* | *m*≠0 |

**20. Отсечения. Аглоритм Сазерленда-Ходжмана.**

Данный алгоритм предназначается для отсечения произвольного многоугольника произвольным ***выпуклым*** окном.

Основан на идее постепенного усечения фигуры сторонами окна.

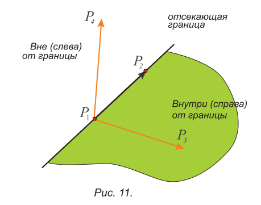
Отсекаемый многоугольник представляется списком вершин.  
Каждая пара вершин (в том числе последняя и первая) задают отрезок.

В процессе отсечения каждый из отрезков анализируется независимо.

Для начала анализируется положение первой вершины. Если она внутри, то происходит добавление ее в конечный список вершин. Ели она снаружи – добавления не происходит. И в том, и в другом случае первая вершина становится текущей. Далее алгоритм работает по шагам для каждой грани, образованной текущей вершиной и следующей. При этом совершаются следующие действия:

* Если грань находится внутри области, то следующая точка добавляется в список.
* Если грань выходит из области, то точка пересечения добавляется в список, а следующая вершина – нет.
* Если грань находится вне области, то ни одна вершина не добавляется.
* Если грань входит в область, то добавляются в конечный список как точка пересечения, так и следующая вершина.

*Видимость вершины можно определять разными способами:*• как в АКБ с помощью скалярного произведения вектора, нормали на вектор, начинающийся в точке на границе и заканчивающийся в пробной точке;  
• при подстановке координат пробной точки в уравнение ориентированной прямой или плоскости.  
• Третий способ сводится к проверке знака координаты *z* у векторного произведения ребра и отсекающей границы

• ***P3* - находится внутри при этом** *P*1*P*2⊗ *P*1*P*3 <0 • ***P4* - находится вне при этом** *P*1*P*2⊗ *P*1*P*4 >0

**21. Отсечения. Алгоритм Вейлера-Азертона.**

Назначение алгоритма – отсечение произвольного многоугольника произвольным окном.  
Предполагается, что каждый из многоугольников задан списком  
вершин, в порядке обхода по часовой стрелке  
В случае пересечения границ отсекаемого многоугольника и окна возникают точки двух типов:  
– входные точки, когда ориентированное ребро отсекаемого многоугольника входит в окно,  
– выходные точки, когда ребро отсекаемого многоугольника идет с внутренней на внешнюю стороны окна.

**Общая схема действия**  
1. Строятся списки вершин отсекаемого многоугольника и окна.  
2. Отыскиваются все точки пересечения. При этом не

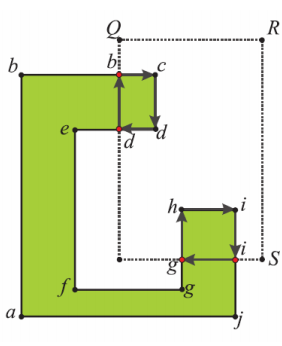
учитываются точки касания и случаи, при которых ребро

инцидентно стороне окна либо совпадает со стороной окна.

3. Списки координат вершин отсекаемого многоугольника и окна дополняются новыми вершинами - координатами точек пересечения. Причем если точка пересечения **K** находится на ребре **A**, **B**, то **A**, **B** превращается в **A**, **K**, **B**. При этом устанавливаются двухсторонние связи между одноименными точками пересечения в списках вершин отсекаемого многоугольника и окна.

Определение части обрабатываемого многоугольника, попавшей в окно  
выполняется следующим образом:  
– Если не исчерпан список входных точек пересечения, то выбираем очередную входную точку.  
– Двигаемся по вершинам отсекаемого многоугольника пока не обнаружится следующая точка пересечения; все пройденные точки, не включая прервавшую просмотр, заносим в результат; переключаемся на просмотр списка вершин окна.  
– Двигаемся по вершинам окна до обнаружения следующей точки пересечения; все пройденные точки, не включая последнюю, прервавшую просмотр, заносим в результат; переключаемся на список вершин обрабатываемого многоугольника.

***Эти действия повторяем пока не будет достигнута исходная вершина — очередная часть отсекаемого многоугольника, попавшая в окно, замкнулась. Переходим на выбор следующей входной точки в списке отсекаемого многоугольника.***

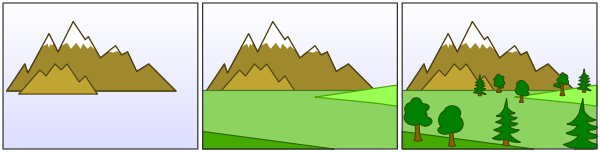


**22. Удаление невидимых поверхностей. Алгоритм художника.**

Алгоритмы удаления заключаются в определении линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

Алгоритм художника — простейший программный вариант решения «проблемы видимости» в трехмерной компьютерной графике.

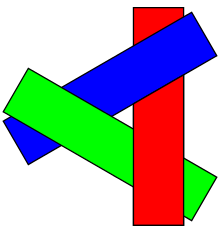
Сначала рисуются наиболее удалённые части сцены, потом части, которые ближе. Постепенно ближние части начинают перекрывать отдаленные части более удалённых объектов. Задача программиста при реализации алгоритма художника — отсортировать все полигоны по удалённости от наблюдателя и начать выводить, начиная с более дальних.

[](https://commons.wikimedia.org/wiki/File:Painter's_algorithm.svg?uselang=ru)

Проблемы алгоритма:

1. Взаимно перекрывающиеся полигоны при любой сортировке будут выведены неверно
2. Второй распространённой проблемой является то, что система прорисовывает также области, которые впоследствии будут перекрыты, на что тратится лишнее процессорное время.

Эти недостатки привели к разработке метода [Z-буфера](https://ru.wikipedia.org/wiki/Z-%D0%B1%D1%83%D1%84%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), который можно рассматривать как развитие алгоритма художника.

[](https://commons.wikimedia.org/wiki/File:Painters_problem.svg?uselang=ru)

**23. Удаление невидимых поверхностей. Буфер глубины.**

Метод Z-буфера – это один из простейших алгоритмов удаления невидимых поверхностей. Работает этот алгоритм в пространстве изображения. Идея Z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов каждого пикселя в пространстве изображения, а Z-буфер предназначен для запоминания глубины (расстояния от картинной плоскости) каждого видимого пикселя в пространстве изображения. Поскольку достаточно распространенным является использование координатной плоскости XYZ в качестве картинной плоскости, то глубина равна координате Z точки, отсюда и название буфера.

В процессе работы значение глубины каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в Z-буфер. Если это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель заносится в этот буфер и, кроме того, производится корректировка Z-буфера новым значением глубины. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по X и Y наибольшего значения функции Z(X,Y).

Главное преимущество алгоритма - его простота. Основной недостаток алгоритма - большой объем требуемой памяти.

**24. Удаление невидимых поверхностей. Алгоритм Робертса.**

В алгоритме Робертса требуется, чтобы все изображаемые тела или объекты были выпуклыми и замкнутыми. Невыпуклые тела должны быть разбиты на выпуклые части.

Определить не лицевые плоскости:

1. Вычислить скалярное произведение вектора R (нормаль), на грань из матрицы объекта.
2. Если скалярное произведение больше нуля – грань невидима (нормаль не смотрит на нас).
3. Удалить весь описывающий многоугольник, лежащий в данной плоскости - это избавляет от необходимости отдельно рассматривать невидимые линии, образуемые пересечением пар невидимых плоскостей.

Для ортогональной проекции - скалярное произведение отрицательно, тогда это лицевая сторона, нет, то не лицевая

**25. Удаление невидимых объектов. Frustum.**

Существует множество алгоритмов отсечения(Culling) невидимой геометрии, однако, самый простой и эффективный алгоритм отсечения - View Frustum Culling. Его необходимо применять как можно раньше, т.к. при незначительных затратах процессорного времени может удастся отбросить значительную часть невидимой геометрии и не выполнять ее дальнейшую обработку.

Для корректного проектирования полигонов на экран ко всем вершинам применяется проекционная матрица и выполняется перспективное деление, в результате чего координаты вершин видимой части сцены лежат внутри единичного куба. Если после умножения на матрицу вершина оказалась вне экрана (т.е. вне единичного куба), ее можно не рисовать. Таким образом, для всех вершин в пространстве сцены ограничивающим объемом являлась усеченная пирамида видимости Frustum, а в пространстве камеры Frustum преобразуется в единичный куб.

Это можно объяснить немного проще. Нет необходимости рисовать объекты, находящиеся сзади или сбоку от камеры и не попадающие в поле зрения. Отрисовываются только те объекты, которые находятся в поле видимости, или в объеме видимости. Этот объем и является усеченной пирамидой,  все что находится вне пирамиды находится и вне экрана.

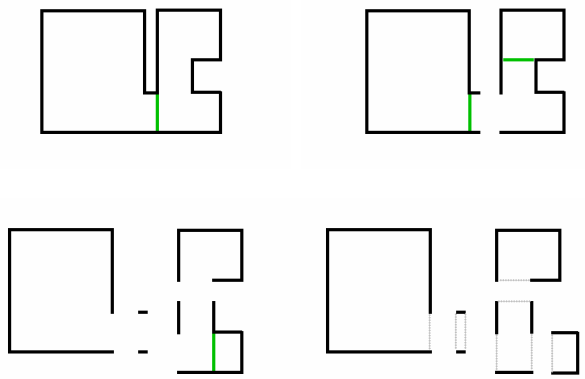
Суть алгоритма в следующем. Исходя из сказанного выше, для определения видимости вершины необходимо проверить, находится ли обрабатываемая вершина внутри усеченной пирамиды, либо домножить вершины на матрицу проецирования и проверить, находится ли вершина внутри единичного куба. На практике удобно использовать первый вариант: найти уравнения всех плоскостей пирамиды и проверить, находится ли точка перед этими плоскостями.

**26. Удаление невидимых объектов. BSP.**

BSP (Binary Space Partition)-деревья существуют в пространстве, состоящем из полигонов и разбитого некой плоскостью на два полупространства. Каждое из них так же разбивается пополам некой плоскостью до тех пор, пока не будет найдено пространство полигонов, образующих некую выпуклую оболочку. В этом случае дальнейшее разбиение прекращается.

Такое разбиение изначального пространства в конечном итоге описывается бинарным деревом. Оно и получило название BSP.

При создании BSP-дерева очень важно, чтобы оно было сбалансированным, т.е. каждая плоскость разбивала бы множество полигонов на два равных по количеству подмножества.



При отрисовке сцены необходимо спуститься с корня BSP дерева к листьям. Для каждого узла поочередно отобразить оба поддерева, причем поддерево, в котором находится наблюдатель отобразить вторым.  
Определить, в каком поддереве находится наблюдатель можно через положение точки относительно плоскости.  
Особое свойство выпуклых оболочек, которыми фактически и являются листья дерева, заключается в том, что полигоны, оболочку образующие, при отрисовке никогда не экранируют друг друга – значит, могут быть отрисованы в любом порядке.

Недостатки:  
BSP-дерево является статичным объектом. Если хотя бы один из полигонов изменит свое положение в пространстве, то это потребует полного пересчета структуры, что приведет к немалым затратам.  
Не смотря на то, что BSP-деревья решают задачу корректного экранирования объектов, они, как и алгоритм художника не сокращают множество отрисовываемых объектов до необходимого.

**27. Удаление невидимых объектов. Octree.**

Octree - восьмеричное дерево.

Данный алгоритм производит разделение объектного пространства на восемь подпространств. Общую схему работы можно представить следующими шагами:

1. Помещаем всю сцену в куб. Этот куб описывает все элементы сцены и является корневым (root) узлом дерева.
2. Проверяем количество примитивов в узле, и если полученное значение меньше определённого порогового, то производим связывание (ассоциацию) данных примитивов с узлом. Узел, с которым ассоциированы примитивы, является листом (leaf).
3. Если же количество примитивов, попадающих в узел, больше порогового значения, производим разбиение данного узла на восемь подузлов (подпространств) путём деления куба тремя плоскостями. Мы распределяем примитивы, входящие в родительский узел, по дочерним узлам. Далее процесс идёт рекурсивно, т. е. для всех дочерних узлов, содержащих примитивы, выполняем пункт 2.

Упрощение алгоритма: теперь, в процессе получения изображения мы рекурсивно выполняем следующую процедуру: начиная с базового (root) куба, мы проверяем, попадает ли данный куб в поле зрения (viewing frustum). Если НЕТ - на этом всё и заканчивается, если же ДА - перемещаемся вглубь рекурсии на один шаг, т. е. поочерёдно проверяем видимость каждого из восьми подузлов корневого узла и т.д.. Преимущество заключается в том, что если определено, что данный узел не виден, то можно смело не выводить и всю геометрию этого узла - она тоже будет не видна. Таким образом, ценой всего лишь нескольких проверок, мы отбросим значительную часть сцены. А в случае, если не виден корневой узел, на экран не будет выведено ничего.

Недостатки:

1. Проблемы разбиения на части сложных изображений
2. Вывод всех объектов, находящихся в поле viewing frustum, но на самом деле, в конечном счёте невидимых.

**28. Удаление невидимых объектов. CPG.**

В основе алгоритма CPG лежит идея, порожденная алгоритмами Робертса и BSP, заключающаяся в том, что сколь угодно сложную геометрию, в общем случае, можно представить как совокупность множества выпуклых оболочек.

В CPG существуют три базовых понятия:

1. наблюдатель (viewer) – некая математическая точка пространства, описывающая местоположение камеры.
2. ячейка (cell), представляющая собой выпуклую оболочку. Ячейка является минимальной единицей пространства.
3. портал (portal) – плоская фигура (обычно четырехугольник), находящаяся на стыке двух ячеек и связывающая их. Из одной ячейки в другую можно попасть только через портал. Основная задача порталов – усечение конуса видимости.

Итак, на данном этапе, мы имеем следующую картину:

1. Изначально определяется конус видимости для наблюдателя. Этот конус видимости является текущим. Через него мы будем видеть нашу сцену. Все, что не попадает в конус видимости, может считаться невидимым.
2. Определяется ячейка, содержащая наблюдателя. Ячейка, содержащая наблюдателя, считается видимой, а все ее объекты (попадающие в текущий конус видимости) добавляются в список отрисовки.
3. Далее рекурсивно для текущей ячейки проверяем видимость порталов. Если конус видимости видит портал (полностью либо частично), соединяющий текущую ячейку А с ячейкой В, то ячейка В так же считается видимой через конус видимости,  
   полученный усечением текущего конуса видимости данным порталом. И так далее.

Заключение :

Использование CPG в приложениях графики реального времени на сегодняшний день способно отбросить более 90% потенциально видимых полигональных объектов. В условиях растущей детализации подобных объектов, а так же усложнения алгоритмов их отрисовки, CPG зарекомендовал себя как стандарт «де-факто» в компьютерных играх с большими виртуальными мирам.  
Однако, являясь с одной стороны, панацеей для сцен, описывающихся множеством выпуклых ячеек (внутренними помещениями), CPG мало применим для сцен открытых пространств, которые трудно разбить на подобные ячейки.

**29. Удаление невидимых объектов. Occlusion.**

Occlusion Culling является развитием алгоритма z-buffer. В последнее время данный алгоритм получил широкое распространение в связи с возможностью его аппаратной поддержки.

Идея алгоритма заключается в том, что объект повышенной сложности (либо несколько подобных объектов) заключается в более простую фигуру (чаще всего - куб).

Затем этот куб пытаются растеризовать с использованием z-буфера, не занося при этом никакой информации ни в буфер глубины, ни в буфер кадра, а лишь считая количество пикселей, проходящих тест глубины.

Если количество пикселей достаточно мало, чтобы считать фигуру невидимой, то ни один другой объект, в нее заключенный, так же не будет виден.

Со стороны может показаться, что в данном случае мы получаем достаточно сомнительный прирост в производительности, заменяя объекты повышенной сложности на другую фигуру, да еще и больших размеров (чем больше пикселей выводится на экран, тем больше затрат). Однако, данная технология оправдывает себя. Тесты, связанные с occlusion culling оптимизируются современными GPU(графический процессор), увеличивая скорость обработки подобных фигур в десятки раз.

**30. Интерполяция. Линейная интерполяция.**

Интерполяция — это способ вычислить промежуточное значение функции по нескольким уже известным ее значениям.

Линейная интерполяция предполагает вычисление промежуточного значения функции по двум точкам (условно проведя прямую между ними). Например, если известны значения функции в двух точках f(x1) и f(x2), то разумно предположить что значение в третьей точке, находящейся между первой и второй, можно найти графически, она лежит на отрезке, соединяющем x1 и x2.

Отсюда:

, где

Фактически, данная формула будет работать и для x, выходящих за рамки интервала [x0, x1].В этом случае, способ нахождения величины y будут называть экстраполяцией.

Главным недостатком такого способа интерполяции будет являться отсутствие возможности построения гладких функций для набора узлов длиннее, чем в два элемента. В таком случае график кривой будет принимать вид ломаной:  
Так же, общий вид функции для интерполяции более чем для двух узлов трудно реализуем математически.

**31. Интерполяция. Полиномиальные интерполяции.**

Если в качестве интерполяционной функции строится алгебраический многочлен – полином, то говорят о полиномиальной интерполяции. Этот полином может иметь различную форму записи. Различают записи с помощью формул Ньютона и Лагранжа.

Интерполяционный многочлен Лагранжа:

Интерполяционные многочлены Ньютона:

*Первая интерполяционная формула Ньютона для таблично заданной функции*

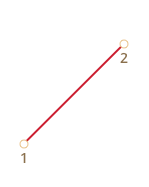
*Вторая интерполяционная формула Ньютона для таблично заданной функции*

**32. Интерполяция. Кривые Безье.**

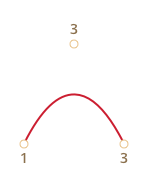
Кривые Безье используются в компьютерной графике для рисования плавных изгибов, в CSS-анимации.

[Кривая Безье](http://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%B2%D0%B0%D1%8F_%D0%91%D0%B5%D0%B7%D1%8C%D0%B5) задаётся опорными точками.Их может быть две, три, четыре или больше:

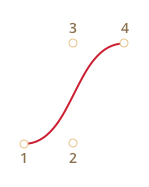
По двум точкам



По трём точкам: *P*



По четырём точкам:



1. **Точки не всегда на кривой.**
2. **Степень кривой равна числу точек минус один**. Для двух точек – это линейная кривая (т.е. прямая), для трёх точек – квадратическая кривая (парабола), для четырёх – кубическая.
3. **Кривая всегда находится внутри** **выпуклой оболочки, образованной опорными точками:**

**33. Типы источников света.**

Источники направленного света (directional light sources). Источник направленного света находится в бесконечно-удаленной точке. В этом случае допустимо считать, что все лучи света от него распространяются параллельно и для всех точек можно использовать один и тот же вектор направления освещения. Хорошим примером такого источника света является Солнце.

Точечные источники света (point light sources). В отличие от источников направленного света, точечные источники находятся в определенной точке пространства с конечными координатами, и свет от них распространяется равномерно по всем направлениям.

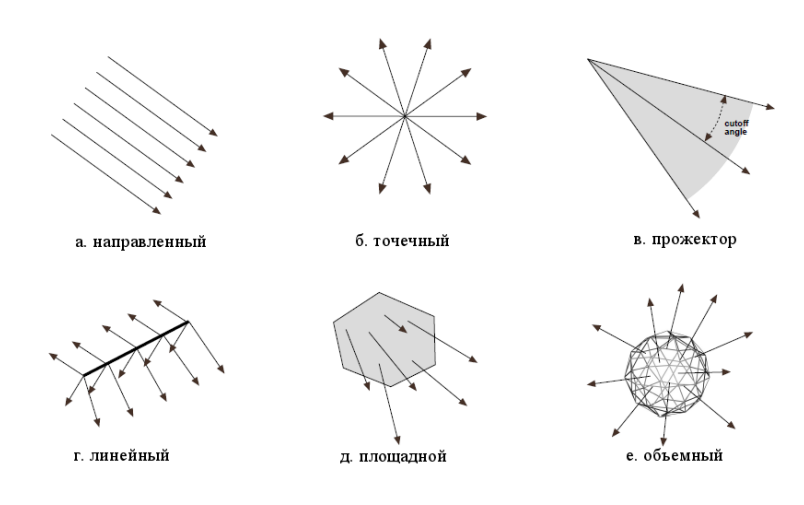
Также для точечных источников учитывается эффект поглощения света, когда интенсивность излучения уменьшается с расстоянием (distance attenuation). Убывание интенсивности излучения с расстоянием задается следующей формулой:

f_{att}(d)=\frac{1}{k_{const} + k_{linear}d + k_{quadratic}d^2}

Прожекторы (spot light sources). Свет от такого источника распространяется лишь в определенном направлении и получается конус света. Обеспечивается это с помощью наложения ограничений на вектор направления \vec{L} от освещаемой точки до источника света.

Рассеянный свет – свет проникающий куда угодно (ambient и diffuse)

Так же существуют линейные, площадные и объемные источники света.



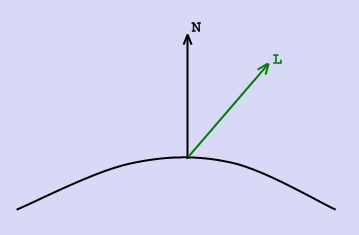
**34. Эмпирические модели освещения.**

Как правило, данные модели подразумевают некий набор параметров, не имеющих физической интерпретации, но позволяющих с помощью подгона получить нужный вид конечной модели. Они обычно эффективны в плане быстродействия и иногда дают более качественный результат за счет большего контроля за выразительностью, чем за точностью.

Примеры эмпирических моделей: **модель Ламберта**, **модель Фонга**.

**Модель Ламберта**

Модель Ламберта моделирует идеальное диффузное освещение. Считается, что свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль N) и направление на источник света (вектор L). Рассеянная составляющая рассчитывается по закону косинусов (закон Ламберта):



Для удобства все векторы, описанные ниже, берутся единичными. В этом случае косинус угла между ними совпадает со скалярным произведением.

http://grafika.me/files/les_screens/light_lambert.jpg

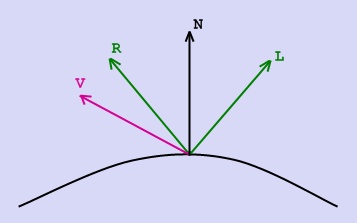
Id– рассеянная составляющая освещенности в точке,  
kd– свойство материала воспринимать рассеянное освещение,  
id– мощность рассеянного освещения,  
L – направление из точки на источник,  
N - вектор нормали в точке.

Модель Ламберта является одной из самых простых моделей освещения. Данная модель очень часто используется в комбинации других моделей, практически в любой другой модели освещения можно выделить диффузную составляющую. Данная модель может быть очень удобна для анализа свойств других моделей (за счет того, что ее легко выделить из любой модели и анализировать оставшиеся составляющие).

**Модель Фонга**

Модель Фонга – классическая модель освещения. Модель представляет собой комбинацию диффузной составляющей (модели Ламберта) и зеркальной составляющей и работает таким образом, что кроме равномерного освещения на материале может еще появляться блик.

.



Прим. В общем случае вектора V, L и N не лежат в одной плоскости.

http://grafika.me/files/les_screens/light_fong.jpg

где  
Is – зеркальная составляющая освещенности в точке,  
ks – коэффициент зеркального отражения,  
is – мощность зеркального освещения,  
R – направление отраженного луча,  
V - направление на наблюдателя,  
α - коэффициент блеска, свойство материала.

**35. Физические модели освещения.**

Физически обоснованные модели стараются аппроксимировать свойства некоторого реального материала. Такие модели учитывают особенности поверхности материала, например слои материала (моделирование кожи или тонких пленок) или же поведение частиц материала (моделирование снега, песка, различных жидкостей).

С развитием технологий и увеличением вычислительно мощности видеокарт все легче становиться использовать в трехмерной графике сложные вычислительные алгоритмы. Иногда, конечно, приходится немного пожертвовать производительностью ради красивого вида.

**Модель освещения Кука-Торранса**

Одной из наиболее продвинутых и согласованных с физикой является модель освещения Кука-Торранса. Она также основана на модели поверхности состоящей из микрограней, каждая из которых является идеальным зеркалом. Модель учитывает коэффициент Френеля и взаимозатенение микрограней.

Так как эта модель используется для расчета отраженного света, то рассеянный свет мы будем вычислять по классической формуле Ламберта.

Теперь рассмотрим модель Кука-Торренса. Количество отраженного света зависит от трех факторов:   
1. Коэффициент Френеля (F)   
2. Геометрическая составляющая, учитывающая самозатенение (G)   
3. Компонент, учитывающий шероховатость поверхности (D)

Общая формула для вычисления отраженного света такова:

Формула вычисления отраженного света. | Быстрая реализация модели освещения Кука-Торренса с использованием GLSL

1) Вычисление геометрической составляющей: image005.png | Быстрая реализация модели освещения Кука-Торренса с использованием GLSL

где N – нормаль в точке, V – вектор взгляда, L – положение источника света,

H – нормализованная сумма векторов L и V. Все векторы должны быть нормированы.

2) Компонент, учитывающий шероховатость поверхности – это распределение микрограней поверхности, для более точного учета отраженного от них света.

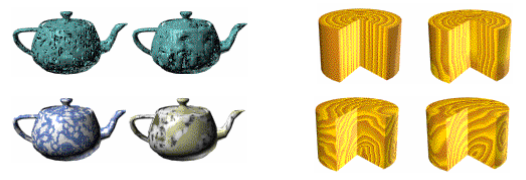
3) Коэффициент Френеля. image011.png | Быстрая реализация модели освещения Кука-Торренса с использованием GLSL

**36. Текстуры. Процедурное текстурирование.**

Текстура как объект характеризуется не только самой матрицей изображения, но также и набором параметров. Такими параметрами могут быть как размерность самой текстуры, так и режим повторений, настройки фильтрации, и другие. Все эти настройки в купе с матрицей хранятся в «одном месте», которое называется «текстурным объектом».

Текстура не обязательно должна являться некоторым изображением, которое будет накладываться на объект. И хотя такой подход является частным случаем применения текстуры, изначально текстуры созданы именно для вариации каких-либо свойств, а таких свойств может быть достаточно много.

Процедурные текстуры - это текстуры, описываемые математическими формулами.  
Наиболее часто встречающиеся процедурные текстуры: разные виды шума, дерево, вода, лава, дым, мрамор, огонь и т.п., то есть те, которые сравнительно просто можно описать математически. Небольшая модификация формулы позволяет в рамках одного типа материала (к примеру, дерева) получает бесконечное количество вариаций.



Недостатки:  
 Однако применение процедурных текстур ограничено случаями недеформируемых поверхностей. Действительно, если некоторая полигональная поверхность будет деформироваться, то соответствующим образом должна деформироваться и соответствующая математическая формула. Реализация же подобного явления с точки зрения математики может быть весьма неочевидна.  
 Вторым недостатком процедурных текстур является их крайняя ограниченность. Дело в том, что математическим формулам поддаются только простые материалы, обладающие свойством регулярности. Подобная ограниченность требует применения иных более сложных методов задания вариации свойств материалов.

**37. Текстуры. Плоское текстурирование**

Текстура как объект характеризуется не только самой матрицей изображения, но также и набором параметров. Такими параметрами могут быть как размерность самой текстуры, так и режим повторений, настройки фильтрации, и другие. Все эти настройки в купе с матрицей хранятся в «одном месте», которое называется «текстурным объектом».

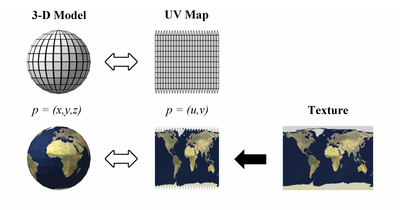
Текстура не обязательно должна являться некоторым изображением, которое будет накладываться на объект. И хотя такой подход является частным случаем применения текстуры, изначально текстуры созданы именно для вариации каких-либо свойств, а таких свойств может быть достаточно много.

Объемное и процедурное текстурирование в пространстве обладают весьма неприятным недостатком, происходящим из попытки описать свойства материала в объеме. Ведь конечная полигональная модель не имеет объема. Вдобавок, нас интересуют свойства  
материала лишь на плоской поверхности полигонов. Имитация материала внутри модели, равно как и за ее пределами является задачей избыточной. Вдобавок, пространственная функция не может учитывать деформации объекта.

Плоское или двумерное текстурирование решает задачу вариации свойств материала только на поверхности объекта. Однако, полигоны трехмерны. В прочих методах текстурирования свойства конкретного фрагмента были функцией трехмерной системы  
координат. В нашем же случае, свойства фрагмента так же должны быть функцией некоторой системы координат.

Барицентрические координаты?  
  
Текстурные координаты?

UV-преобразование или развёртка в трёхмерной графике (англ. UV map) — соответствие между координатами на поверхности трёхмерного объекта (X, Y, Z) и координатами на текстуре (U, V). Значения U и V обычно изменяются от 0 до 1[1]. Развёртка может строиться как вручную, так и автоматически.



**38. Текстуры. Фильтрация.**

Текстура как объект характеризуется не только самой матрицей изображения, но также и набором параметров. Такими параметрами могут быть как размерность самой текстуры, так и режим повторений, настройки фильтрации, и другие. Все эти настройки в купе с матрицей хранятся в «одном месте», которое называется «текстурным объектом».

Изображение на экране состоит из множества пикселей, количество которых определяется разрешением. Для вывода цветного изображения необходимо определять цвет каждого пикселя. Определяется его цвет посредством наложения текстурных изображений на полигоны, которые расположены в трёхмерном пространстве. Текстурные изображения состоят из пикселей, вернее, текселей, то есть тексель - это пиксель двухмерного изображения, наложенного на ЭБ-поверхность. Главная проблема заключается в следующем: какой тексель или тексели определяют цвет пикселя на экране.

**Point Sampling** (поточечная выборка) - самый простой способ определения цвета пикселя. Этот алгоритм основан на текстурном изображении: выбирается всего один тексель, который ближе всех расположен к центру светового пятна, и по нему происходит определение цвета пикселя. Нетрудно догадаться, что это совершенно не верно. Во- первых, цвет пикселя определяется несколькими текселями, а мы выбрали только один. Во-вторых, форма светового пятна может измениться, а алгоритм не принимает это во внимание. Главным недостатком поточной выборки является тот факт, что когда полигон расположен близко к экрану, количество пикселей будет значительно выше, чем текселей, из-за чего качество изображения очень сильно пострадает. У Point Sampling есть преимущество. Из-за того, что определение цвета пикселя осуществляется всего по одному текселю, данный метод не критичен к пропускной способности памяти, а это автоматически даёт данному способу фильтрации колоссальные преимущества в том плане, что на фильтрацию по данной схеме затрачивается очень мало ресурсов.

**Bi-Linear Filtering** - билинейная фильтрация, основанная на методе использования интерполяционной техники. Для определения нужных текселей используется основная форма светового пятна, то есть круг. Изображение получается более качественным, блочность отсутствует, однако близкие к экрану полигоны выглядят расплывчато. Расплывчатость - отнюдь не самая главная проблема билинейной фильтрации. Дело в том, что аппроксимация выполняется корректно лишь для объектов, расположенных параллельно экрану или точке наблюдения, в то время как 99% объектов в любой компьютерной игре расположены непараллельно к точке наблюдения. Отсюда можно сделать вывод, что 99% объектов будут аппроксимироваться неправильно. Ко всему прочему билинейная фильтрация значительно требовательней к пропускной полосе данных памяти.

**Tri-Linear filtering** - трилинейная фильтрация, представляет собой симбиоз билинейной фильтрации и mip-текстурирования. Для того чтобы правильно вывести пиксель на экран, необходимо скомбинировать значения всех текселей, которые и будут накладываться на этот пиксель. Но пропускная способность памяти не бесконечна, а это огромнейший обьём работы, который неблагоприятно отразится на быстродействии видеокарты. Метод MIP-текстурирования основан на генерации и хранении множества версий исходного текстурного изображения. Эти версии имеют большое количество разрешений, каждое из которых всё меньше и меньше исходного. Во время текстурирования пикселя вам достаточно выбрать версию текстурного изображения. По сути, mip-уровни представляют собой заранее рассчитанные, более маленькие версии исходной текстуры, из-за чего обеспечивается лучшая аппроксимация. В итоге имеем несколько лучшее качество фильтрации, нежели у билинейной. Стоит отметить, что качество улучшается незначительно, а требования к ширине полосы пропускания памяти в сравнении с той же Bi-Linear Filtering удваиваются.

**Anisotropic filtering.** Для получения хороших результатов фильтрации недостаточно определять цвет по нескольким пикселям. Вполне очевидно - для того чтобы добиться лучшего качества, необходимо использовать все пиксели светового пятна с усреднённым значением. Мы также должны учитывать тот факт, что форма светового пятна изменяется вместе с изменением положения полигона относительно точки наблюдения. Собственно говоря, этим и предлагает пользоваться анизотропная фильтрация.

Использование такого количества пикселей требует огромной пропускной способности памяти, поэтому анизотропная фильтрация более- менее сносно может функционировать лишь на мощных современных видеокартах. Помимо всего этого, Anisotropic filtering предлагает задействовать разнообразные фильтры для аппроксимации формы светового пятна. Эти фильтры имеют форму эллипса для нескольких возможных углов положения полигона. Уровень анизотропной фильтрации определяется числом текселей, которые обрабатываются при вычислении конечного пикселя. Самыми распространёнными уровнями фильтрации являются 2х (16 текселей), 4х (32 текселя), 8х (64 текселя) и 16x (128 текселей). Очевидно, что при повышении уровня анизотропной фильтрации нагрузка на полосу пропускания памяти также увеличивается, а это неминуемо сказывается на производительности.

**39. Текстуры. Мультитекстурирование.**

Текстура как объект характеризуется не только самой матрицей изображения, но также и набором параметров. Такими параметрами могут быть как размерность самой текстуры, так и режим повторений, настройки фильтрации, и другие. Все эти настройки в купе с матрицей хранятся в «одном месте», которое называется «текстурным объектом».

Текстура не обязательно должна являться некоторым изображением, которое будет накладываться на объект. И хотя такой подход является частным случаем применения текстуры, изначально текстуры созданы именно для вариации каких-либо свойств, а таких свойств может быть достаточно много.

Ярким примером имитации множества независимых свойств на поверхности материала являются эффекты мультитекстурирования. Суть всех эффектов сводится к тому, что для некоторой поверхности наличествует определенное количество свойств, вариацию которых нельзя описать единой текстурой. Примером такой ситуации могут являться случаи, когда для двух различных свойств поверхности необходимо использовать различные системы координат.

**Декали (Decals)**

Данный эффект используется для нанесения на поверхность некоторого объекта дополнительного рисунка. В видеоиграх это могут быть всевозможные напыления, граффити, следы крови или дыры от выстрелов.

Суть заключается в том, что для полигонального объекта одновременно используются две текстуры с различными системами координат - текстура самой поверхности и текстура рисунка.

Текстура рисунка (декали) должна представлять из себя растровое изображение с прозрачными областями. В тех местах, где тексель (texel - texture element) декали будет прозрачным, мы будем считать конечным свойством свойство основной текстуры. В случае непрозрачного текселя декали мы будем брать за основное свойство - свойство декали.

Таким образом, для вычисления свойства поверхности в некоторой точке достаточно применить следующую формулу:

Result = Decal \* alpha + Base \* ( 1 - alpha ),

Где Decal - это свойство декали в данном фрагмента, Base - свойство основного материала стены, а alpha - прозрачность материала декали.

**Детали (Detail)**

Часто возникают ситуации, при которых одна текстура используется для покрытия большой площади полигональной модели. В случае приближения наблюдателя к такой поверхности, из-за низкой детализации текстуры поверхность начинает выглядеть размытой. Однако существует сравнительно простой способ искусственного повышения детализации за счет использования карты деталей. Такая карта является текстурой, в которую записаны отклонения в большую или меньшую сторону от результата, полученного при семплирования свойств основного материала.

Result = Base + Detail

**40. Генерация тени. Теневые карты.**

Все предметы в окружающем мире при освещении отбрасывают тень. Тень выражается зрительно уловимым силуэтом, возникающим на произвольной поверхности благодаря присутствию объекта (тела или вещества; например, в газообразном или жидком состоянии) между ней и источником света. Именно тень в сочетании со светом позволяет человеку видеть форму предметов. Поэтому для того чтобы изображение, выводимое на экран, выглядело более привычным и больше радовало глаз, необходимо правильно сочетать в них свет и тень.

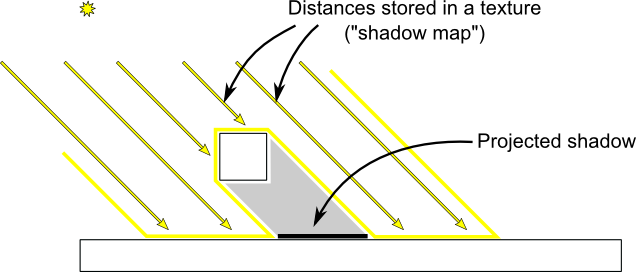
**Теневые карты.**

В основе этого метода лежит крайне простая идея, что освещенные фрагменты - это те фрагменты, которые видны из положения источника света.

Основной алгоритм получения теневой карты состоит из двух проходов.

 На первом шаге в положении источника света размещается виртуальная камера и производится рендеринг всей сцены при помощи этой камеры. Таким образом, мы получим все освещенные фрагменты. Далее извлекаются значения из буфера глубины и копируются в специальную текстуру (теневую карту - shadow map).

На втором этапе проводится обычный рендеринг сцены из положения наблюдателя. Этот процесс состоит из трех основных компонентов: первый - найти координаты объекта, относительно того как его видно из точки освещения, второй - это тест, который сравнивает эту координату с значением из теневой карты. Если первая величина меньше или равна второй, то данный фрагмент проходит тест глубины для источника света и, следовательно, освещен.



Одним из полезных свойств построения теневой карты является то, что ее построение не зависит от сложности геометрии на сцене, необходимо лишь получить буфер глубины для каждого из обсчитываемых источников освещения.

Но при этом для каждого источника света придется строить свою теневую карту и вычислять соответствующие значения буфера глубины. Также данный метод является относительно неточным и поэтому возможны различные артефакты, например, ступенчатость.

**41. Генерация тени. Radiosity.**

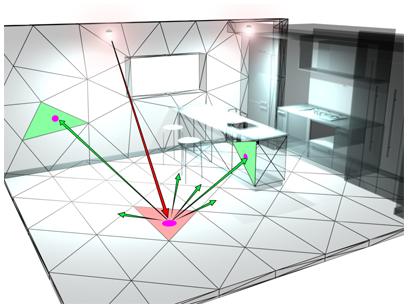
Все предметы в окружающем мире при освещении отбрасывают тень. Тень выражается зрительно уловимым силуэтом, возникающим на произвольной поверхности благодаря присутствию объекта (тела или вещества; например, в газообразном или жидком состоянии) между ней и источником света. Именно тень в сочетании со светом позволяет человеку видеть форму предметов. Поэтому для того чтобы изображение, выводимое на экран, выглядело более привычным и больше радовало глаз, необходимо правильно сочетать в них свет и тень.

Radiosity (метод излучательности) — один из методов глобального освещения (global illumination, GI). Суть Radiosity состоит в том, что все поверхности сцены разбиваются на небольшие фрагменты — патчи (patches), каждый из которых наделён свойствами излучать, поглощать и отражать свет.

Процесс вычисления освещения по алгоритму radiosity состоит из набора итераций, каждая из которых уточняет результат расчёта (radiosity solution). Для каждого патча на сцене подсчитывается полученная им от других патчей энергия, а также доля этой энергии, которая будет излучена патчем на следующей итерации.

В результате алгоритм radiosity позволяет получать реалистичные эффекты вторичных отражений, неточечных источников света, мягких теней и т.д.

Изначально radiosity является очень ресурсоёмким, пригодным только для предрассчитанного освещения (precalculated lightning). Однако существует ряд оптимизаций, позволяющих значительно снизить количество вычислений, и ускорить их (в частности, использование GPU для расчёта).



**42. Спецэффекты. Спрайты.**

Спецэффе́кт, специа́льный эффе́кт, комбинированная съёмка (англ. special effect, сокр. SPFX, SFX или FX) — совокупность технологий, в результате которых на экране получается изображение, не существовавшее в действительности, но для зрителей не отличающееся от реального. Спецэффекты также часто применяются, когда естественная съёмка сцены слишком затратна по сравнению со спецэффектом (например, съёмка масштабного взрыва). Спецэффекты применяются и для улучшения или модификации уже предварительно отснятого. Спецэффекты условно разделяют на две группы — визуальные и механические эффекты. К визуальным относятся оптические эффекты (комбинированные съёмки), а также компьютерная графика. Механические (физические) спецэффекты — это обработка материалов перед съемкой.

Спрайт - частично прозрачное двухмерное изображение, которое отображается на специальной плоскости в трехмерной сцене. Плоскость спрайта всегда перпендикулярна оси, исходящей от камеры. Изображение можно масштабировать, чтобы имитировать перспективу, его можно поворачивать, он может перекрывать другие объекты и закрываться, но его можно рассматривать только под одним углом. Этот метод рендеринга также называется рекламным щитом. Наблюдение спрайта под несоответствующим углом приводит к разрушению иллюзии.

Если один спрайт представляет из себя неподвижную картинку, то быстро сменяющая друг друга серия этих картинок составляет анимацию, называемую спрайтовой. Такая серия картинок называется атласом спрайтов. Данный вид анимации отличается тем, что на экране сменяется не кадр (фрейм) целиком, а только его маленький кусочек, где появляются спрайты. Также спрайтовую анимацию иногда называют программной.

**Разрушение иллюзии**

Наблюдение спрайта под несоответствующим углом приводит к разрушению иллюзии. В зависимости от математического аппарата рендеринга («движка»), разрушение может происходить двумя способами.

1. «Эффект портрета». Спрайтовый объект постоянно держится к наблюдателю нормалью к поверхности, что приводит к тому, что он визуально «поворачивается» все время к нему «лицом». В небольших пределах это незаметно, но при взгляде, скажем, на монстра сверху он может оказаться «лежащим на спине» на горизонтальном полу и при этом перебирающим ногами, да еще и бегущим в нужном направлении, не меняя более чем странной позы. Для борьбы с этим эффектом применяются наборы из большого числа спрайтов, снятых с разных направлений, которые подменяют друг друга в зависимости от угла наблюдения. 2. «Эффект Ельцина. Возникает, если спрайт не поворачивается нормалью к поверхности к наблюдателю, выдавая «плоскостность» объекта даже при небольших изменениях угла наблюдения.

Различают следующие виды спрайтов: Billboard — спрайт, постоянно повёрнутый лицом к камере (по аналогии с рекламными щитами на автодорогах, которые повёрнуты под наиболее выгодным углом).

Impostor — спрайт, который заменяет трёхмерную модель на большом удалении.

**43. Спецэффекты. Системы частиц.**

Спецэффе́кт, специа́льный эффе́кт, комбинированная съёмка (англ. special effect, сокр. SPFX, SFX или FX) — совокупность технологий, в результате которых на экране получается изображение, не существовавшее в действительности, но для зрителей не отличающееся от реального. Спецэффекты также часто применяются, когда естественная съёмка сцены слишком затратна по сравнению со спецэффектом (например, съёмка масштабного взрыва). Спецэффекты применяются и для улучшения или модификации уже предварительно отснятого. Спецэффекты условно разделяют на две группы — визуальные и механические эффекты. К визуальным относятся оптические эффекты (комбинированные съёмки), а также компьютерная графика. Механические (физические) спецэффекты — это обработка материалов перед съемкой.

Систе́ма части́ц — используемый в компьютерной графике способ представления объектов, не имеющих чётких геометрических границ (различные облака, туманности, взрывы, струи пара, шлейфы от ракет, дым, снег, дождь и т. п.).

Система частиц состоит из определённого (фиксированного или произвольного) количества частиц. Математически каждая частица представляется как материальная точка с дополнительными атрибутами, такими как внешний вид, скорость, ориентация в пространстве, угловая скорость, и т. п. В ходе работы программы каждая частица изменяет своё состояние по определённому, общему для всех частиц системы, закону. Например, частица может подвергаться воздействию гравитации, менять размер, цвет, скорость и так далее, и, после проведения всех расчётов, частица визуализируется. Частица может быть визуализирована точкой, треугольником, спрайтом, или даже полноценной трёхмерной моделью.

В большинстве реализаций, новые частицы испускаются так называемым «эмиттером». Эмиттером может быть точка, тогда новые частицы будут возникать в одном месте. Так можно смоделировать, например, взрыв: эмиттером будет его центр. Эмиттером может быть отрезок прямой или плоскость: например, частицы дождя или снега должны возникать на высоко расположенной горизонтальной плоскости. Эмиттером может быть и произвольный геометрический объект: в этом случае новые частицы будут возникать на всей его поверхности. При этом обычно появление каждой новой чатситцы должно происходить с удалением какой-либо старой.

На протяжении жизни частица редко остаётся в покое. Частицы могут двигаться, вращаться, менять свой цвет и/или прозрачность, и сталкиваться с трёхмерными объектами. Часто у частиц задана максимальная продолжительность жизни, по истечении которого частица исчезает.

В трёхмерных приложениях реального времени (например в компьютерных играх) обычно считается, что частицы не отбрасывают тени друг на друга, а также на окружающую геометрию, и что они не поглощают, а излучают свет. Без этих упрощений обсчёт системы частиц будет требовать больше ресурсов.

**44. Спецэффекты. Постпроцессинг.**

Спецэффе́кт, специа́льный эффе́кт, комбинированная съёмка (англ. special effect, сокр. SPFX, SFX или FX) — совокупность технологий, в результате которых на экране получается изображение, не существовавшее в действительности, но для зрителей не отличающееся от реального. Спецэффекты также часто применяются, когда естественная съёмка сцены слишком затратна по сравнению со спецэффектом (например, съёмка масштабного взрыва). Спецэффекты применяются и для улучшения или модификации уже предварительно отснятого. Спецэффекты условно разделяют на две группы — визуальные и механические эффекты. К визуальным относятся оптические эффекты (комбинированные съёмки), а также компьютерная графика. Механические (физические) спецэффекты — это обработка материалов перед съемкой.

Постобработка (post processing) - это все действия по изменению изображения трехмерной сцены после этапа её визуализации. То есть те действия, которые производятся уже над готовым двумерным изображением. Эффекты могут быть разнообразными: от изменения яркости и контрастности, до эффектов искажения и размытия. Наиболее распространенные эффекты постобработки: Bloom, Motion Blur, Depth Of Field.

Bloom — это один из кинематографических эффектов постобработки, при помощи которого наиболее яркие участки изображения делаются еще более яркими. Это эффект очень яркого света, проявляющийся в виде свечения вокруг ярких поверхностей, после применения bloom фильтра такие поверхности не просто получают дополнительную яркость, свет от них (ореол) частично воздействует и на более темные области, соседствующие с яркими поверхностями в кадре.

Смазывание в движении (motion blur) происходит при из-за движения объектов в кадре. Так происходит, если объект перемещается относительно камеры или камера относительно объекта, и величина смазывания дает нам представление о величине скорости движения объекта. Возможные применения эффекта motion blur в играх: все гоночные игры (для создания эффекта очень высокой скорости движения).

Depth of field (глубина резкости) - это размывание объектов в зависимости от их положения относительно фокуса камеры. Объекты, расположенные на определенном расстоянии от камеры находятся в фокусе и выглядят на картинке резкими, а более удаленные от камеры или приближенные к ней объекты выглядят, наоборот, размытыми, резкость снижается постепенно при увеличении или снижении расстояния.

В компьютерной графике каждый объект отрендеренного изображения идеально четкий, так как линзы и оптика не имитируется при расчетах. Поэтому, для достижения фото- и кинореалистичности приходится применять специальные алгоритмы, чтобы сделать для компьютерной графики нечто похожее. Эти техники симулируют эффект разного фокуса для объектов, находящихся на разном расстоянии.

**45. Анимирование. Вершинная анимация. Морфинг.**

Анимирование- это создание иллюзии движения объектов на экране монитора. На экране создаются рисунки начального и конечного положения движущихся объектов, а все промежуточные состояния рассчитывает и изображает компьютер, выполняя расчеты, опирающиеся на математическое описание данного вида движения. Полученные рисунки, выводимые последовательно на экран с определенной частотой, создают иллюзию движения.

Один из методов более совершенной анимации — метод вершинной (вертексной) анимации объектов — связан с представлением объекта как цельной полигональной модели (еще говорят, что он должен представлять собой одну сетку).

Например, при применении этого метода к человеческому лицу, оно сначала моделируется с нейтральным выражением, а затем создается «morph target» для каждого другого выражения. The "morph target" это деформированная версия фигуры. Когда лицо анимируется, аниматор может затем плавно изменять (или «смешивать») между базовой фигурой и одной или несколькими «morph target».

При использовании для анимации лица эти «morph target» часто называются «ключевыми позами». Интерполяции между ключевыми позициями при визуализации анимации обычно представляют собой небольшие и простые преобразования движения, вращения и масштаба, выполняемые программным обеспечением 3D. Данный процесс называется морфинг.

Морфинг (англ. morphing, трансформация) — технология в компьютерной анимации, визуальный эффект, создающий впечатление плавной трансформации одного объекта в другой. Морфинг также часто используется для создания анимации, когда не стоит задача добиться эффекта превращения одного объекта в другой, а требуется лишь выстроить промежуточные состояния между двумя (и более) ключевыми положениями анимируемого объекта.

При методе вершинной анимации приходится хранить положение каждой вершины поли-гональной модели в каждом кадре анимации. То, что анимации моделей хранятся как бы по трехмерным кадрам, приводит к росту размера файлов пропорционально количеству кадров.

**46. Анимирование. Скиннинг.**

Анимирование- это создание иллюзии движения объектов на экране монитора. На экране создаются рисунки начального и конечного положения движущихся объектов, а все промежуточные состояния рассчитывает и изображает компьютер, выполняя расчеты, опирающиеся на математическое описание данного вида движения. Полученные рисунки, выводимые последовательно на экран с определенной частотой, создают иллюзию движения.

Скиннинг-Операция обворачивания скелета 3D-модели полигональной сеткой

Всё вместе, кости и дополнительные элементы, называется риг, а процесс построения скелета и управляющих элементов – риггинг. После того как скелет и управляющие элементы готовы их нужно связать с моделью, чтобы при анимации она двигалась за костями скелета, этот процесс называется скининг. Скининг (от английского слова skin - кожа, skinning - процесс создания кожи, встречается также написание скиннинг) - это один из этапов сетапа 3d-персонажа, когда модель персонажа привязывается (скинится) к скелету. Делается это для того, чтобы при движении скелета двигалась и сама модель персонажа. Это означает, что созданный скелет будет влиять на вершины модели и соответственно перемещать их. Участки нашей модели крепятся к соответствующим участкам скелета (шея к шейным костям, фаланги пальцев к соответствующим костям и т.д.). Делая скининг, аниматор обозначает на поверхности модели места влияния костей на определенные участки, наибольшее внимание уделяется местам сгибов, при этом для более точных и плавных эффектов сгибания необходимо связывать каждую вершину не с одной, а с несколькими суставами, чтобы каждый из оказывал свое влияние на вершину полигональной сетки ведь нужно настроить все так чтоб при анимации за костями двигалась только необходимая часть модели. Каждому суставу присваивается определенный вес. Он определяет степень влияния сустава на вершину. Сумма всех весов для каждой вершины равна 1. Однако даже при распределении весов могут возникнуть артефакты. Например, может получиться слишком гладкий сгиб локтя. В зависимости от сложности модели (наличие складок, морщин) работа над скиннингом может занять продолжительное временя.

**47. Анимирование. Скелетная анимация.**

Анимирование- это создание иллюзии движения объектов на экране монитора. На экране создаются рисунки начального и конечного положения движущихся объектов, а все промежуточные состояния рассчитывает и изображает компьютер, выполняя расчеты, опирающиеся на математическое описание данного вида движения. Полученные рисунки, выводимые последовательно на экран с определенной частотой, создают иллюзию движения.

Скелетная анимация заключается в том, что создаётся скелет, представляющий собой как правило древообразную структуру костей, в которой каждая последующая кость «привязана» к предыдущей, то есть повторяет за ней движения и повороты с учётом иерархии в скелете. Далее каждая вершина модели «привязывается» к какой-либо кости скелета. Таким образом, при движении отдельной кости двигаются и все вершины, привязанные к ней. Благодаря этому задача анимации сильно упрощается, потому что отпадает необходимость анимировать отдельно каждую вершину модели, а достаточно лишь задавать положение и поворот костей скелета.

Скелетная анимация с развесовками представляет собой более продвинутый вариант скелетной анимации, в ней каждая вершина модели может быть связана не с одной, а с несколькими костями. При этом для каждой кости определяется свой вес, то есть величина влияния этой кости на перемещение вершины. Чем больше вес какой-то кости, тем сильнее вершина смещается под её влиянием.

Благодаря развесовкам можно анимировать плавные изгибы поверхностей и т. п.

**48. Анимирование. Motion Capture.**

Анимирование- это создание иллюзии движения объектов на экране монитора. На экране создаются рисунки начального и конечного положения движущихся объектов, а все промежуточные состояния рассчитывает и изображает компьютер, выполняя расчеты, опирающиеся на математическое описание данного вида движения. Полученные рисунки, выводимые последовательно на экран с определенной частотой, создают иллюзию движения.

Motion capture — метод анимации персонажей и объектов. Применяется для создания спецэффектов в фильмах. Широко используется в игровой индустрии.

Существуют два основных вида систем motion capture:

1. Маркерная система motion capture, где используется специальное оборудование. На человека надевается костюм с датчиками, он производит движения, требуемые по сценарию, встаёт в условленные позы, имитирует действия; данные с датчиков фиксируются камерами и поступают в компьютер, где сводятся в единую трёхмерную модель, точно воспроизводящую движения актёра, на основе которой позже (или в режиме реального времени) создаётся анимация персонажа. Также этим методом воспроизводится мимика актёра (в этом случае на его лице располагаются маркеры, позволяющие фиксировать основные мимические движения).

2. Безмаркерная технология, не требующая специальных датчиков или специального костюма. Безмаркерная технология основана на технологиях компьютерного зрения и распознавания образов. Актер может сниматься в обычной одежде, что сильно ускоряет подготовку к съемкам и позволяет снимать сложные движения (борьба, падения, прыжки, и т. п.) без риска повреждения датчиков или маркеров. На сегодняшний день существует программное обеспечение «настольного» класса для безмаркерного захвата движений. В данном случае не требуется специального оборудования, специального освещения и пространства. Съёмка производится с помощью обычной камеры (или веб-камеры) и персонального компьютера.

На сегодняшний день существуют большое количество маркерных систем захвата движений. Различие между ними заключается в принципе передачи движений]:

1. Оптические пассивные. На костюме, входящем в комплект такой системы, прикреплены датчики-маркеры, которые названы пассивными, потому что отражают только посланный на них свет, но сами не светятся. В таких системах свет (инфракрасный) на маркеры посылается с установленных на камерах высокочастотных стробоскопов и, отразившись от маркеров, попадает обратно в объектив камеры, сообщая тем самым позицию маркера.

Минус оптических пассивных систем:

· Длительность размещения маркеров на актёре

· При быстром движении или близком расположении маркеров друг к другу система может их путать (технология не предусматривает идентификацию каждого маркера)

2. Оптические активные названы так потому, что вместо светоотражающих маркеров, которые крепятся к костюму актёра, в них используются светодиоды с интегрированными процессорами и радио-синхронизацией. Каждому светодиоду назначается идентификатор, что позволяет системе не путать маркеры друг с другом, а также узнавать их, после того как они были перекрыты и снова появились в поле зрения камер. Во всём остальном принцип работы таких систем схож с пассивными системами.

Минусы активных систем:

· Отсутствие возможности захвата движений и мимики лица

· Дополнительный контроллер, крепящийся к актёру и подключенный к маркерам-светодиодам, сковывает его движения

· Хрупкость и относительно высокая стоимость маркеров-светодиодов

3. Магнитные системы, в которых маркерами являются магниты, а камерами — ресиверы, система высчитывает их позиции по искажениям магнитного потока.

Минусы магнитных систем:

· Магнитные системы подвержены магнитным и электрическим помехам от металлических предметов и окружения (электропроводки помещения, оргтехники, арматуры в плитах строения)

· Переменчивая чувствительность сенсоров в зависимости от их положения в рабочей зоне

· Меньшая по сравнению с оптическими системами рабочая зона

· Отсутствие возможности захвата движений и мимики лица

· Дополнительный контроллер, прикреплённый к актёру и подключенный к магнитным маркерам, или даже связка проводов, тянущаяся от актёра к компьютеру.

· Высокая стоимость магнитных маркеров

4. Механические системы напрямую следят за сгибами суставов, для этого на актёра надевается специальный механический mocap-скелет, который повторяет следом за ним все движения. В компьютер при этом передаются данные об углах сгибов всех суставов.

Минусы механических систем:

· Mocap-скелет, с дополнительным контроллером, прикреплённым к актёру и подключенным к сенсорам сгибов, а в некоторых случаях и провода, тянущиеся от скелета, сильно ограничивают движения актёра.

· Отсутствие возможности захвата:

1. Движений и мимики лица

2. Движений тесного взаимодействия двух и более актёров (борьба, танцы с поддержками и т. д.)

3. Движений на полу — кувырки, падения и т. д.

· Риск поломки механики при неосторожном использовании.

5. Гироскопические / инерциальные системы для сбора информации о движении используют миниатюрные гироскопы и инерциальные сенсоры, расположенные на теле актёра — также как и маркеры или магниты в других mocap-системах. Данные с гироскопов и сенсоров передаются в компьютер, где происходит их обработка и запись. Система определяет не только положение сенсора, но также угол его наклона.

Минусы гироскопических / инерциальных систем:

· Отсутствие возможности захвата движений и мимики лица

· Дополнительный контроллер, прикреплённый к актёру и подключенный к магнитным маркерам, или даже связка проводов, тянущаяся от актёра к компьютеру.

· Высокая стоимость гироскопов и инерциальных сенсоров

· Для определения положения актёра в пространстве нужна дополнительная мини-система (оптическая или магнитная)