1.Устройство оптической системы человека, свет и цвет, восприятие цвета.

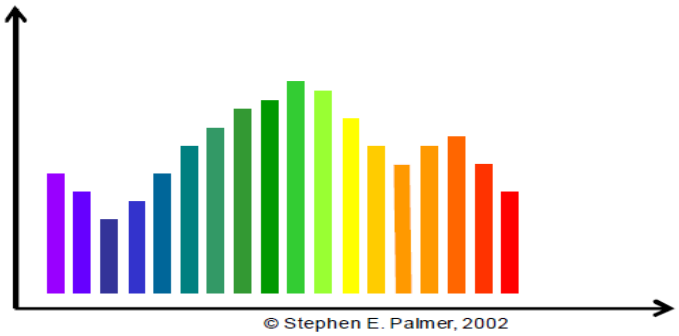
Изображение оптическое – картина, получаемая в результате прохождения через оптическую систему лучей, распространяющихся от объекта, и воспроизводящая его контуры и детали.
Функция интенсивности канала, заданная на 2х мерной сетке (Лекция номер 1, часть 2 слайд 8)

Цвет – это психологическое свойство нашего зрения, возникающее при наблюдении объектов и света, а не физические свойства объектов и света.

Цвет – это результат взаимодействия света, сцены и нашей зрительной системы

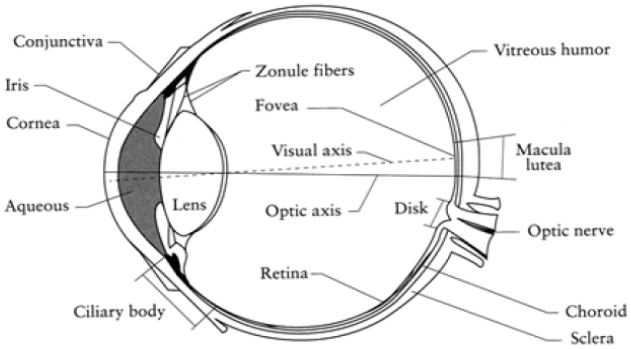
Физика света

Любой источник света можно полностью описать спектром: количество излученной энергии в единицу времени для каждой длины волны в интервале 400 – 700 nm
По y – Энергия (Число фотонов в мс)
По x – Длина Волны.



Видимый цвет - это результат взаимодействия спектра излучаемого света и поверхности.

Сечатка глаза(18 слайд)
<u>Оптическая система человека</u>



Глаз как камера!

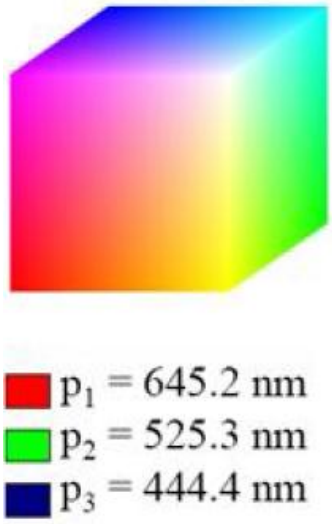
- **Радужка** – цветная пленка с радиальными мышцами
- **Зрачок** - отверстие (апертура), диаметр управляется радужкой
- **Хрусталик** – «линза», меняющая форму под действием мышц
- Где матрица?
  - Клетки-фоторецепторы на сетчатке

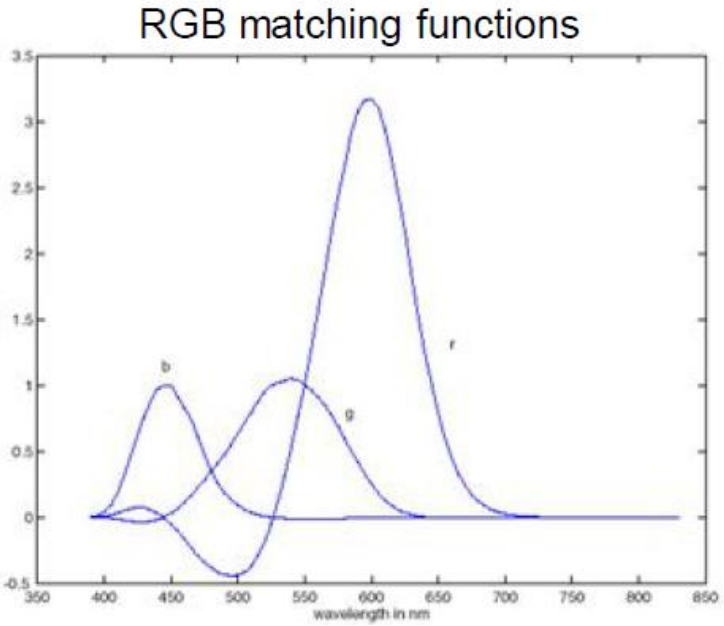
2.Цветовые системы RGB, CMYK, HSV, YIQ, получение цветных изображений.

RGB

Основные цвета – монохроматические (в мониторе им соответствует три вида фосфоров)

Вычитание необходимо для соответствия некоторым длинам волны



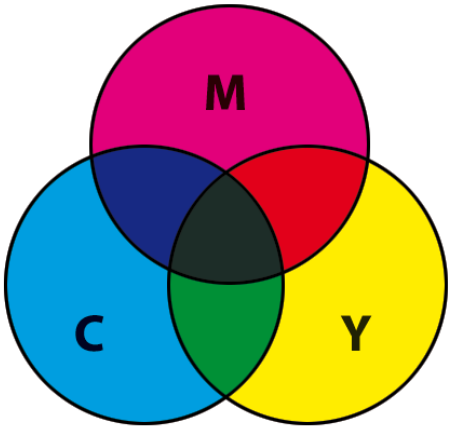


CMYK

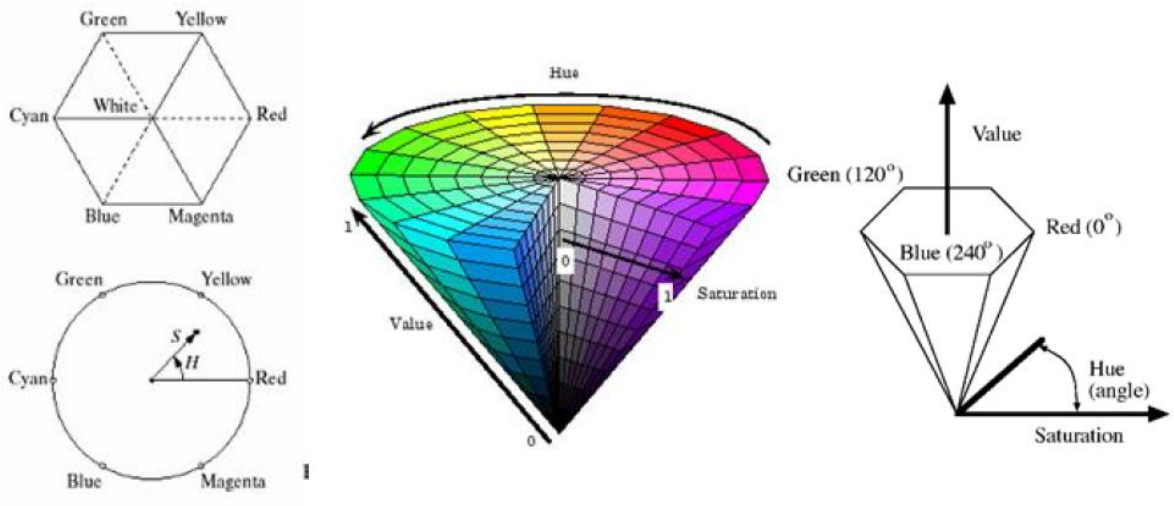
Четырёхцветная автотипия (CMYK) — субтрактивная схема формирования цвета, используемая прежде всего в полиграфии для стандартной триадной печати. Схема CMYK обладает сравнительно с RGBменьшим цветовым охватом.

(голубой, пурпурный, желтый)

Наложение реальных типографских красок CMY



HSV



Координаты выбраны с учетом человеческого восприятия: Hue (Тон), Saturation (Насыщенность), Value (Intensity, Интенсивность)

Слайд 38 (первая лекция, часть 2) Перевод из RGB в HSV.

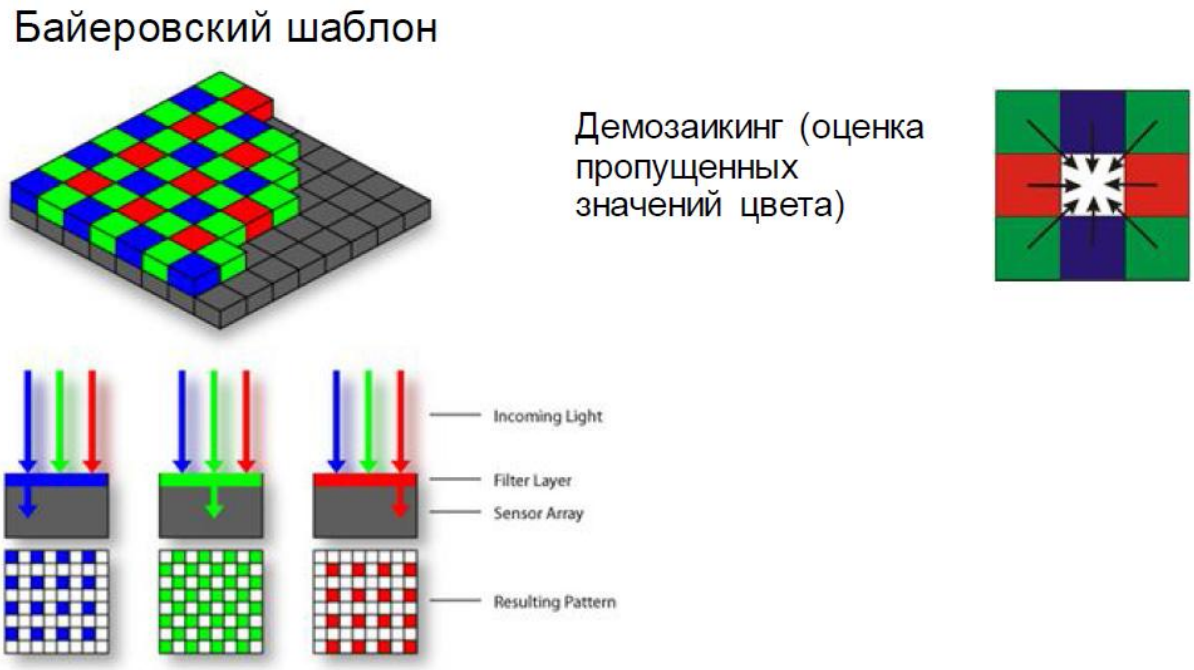
YIQ

YIQ — цветовая модель.

Цвет представляется как 3 компоненты — яркость (Y) и две искусственных цветоразностных (I и Q). Сигнал I называется синфазным, Q - квадратурным. Конверсия в RGBи обратно осуществляется по следующим формулам:

Модель **YIQ** слайд 36(первая лекция, часть 2)

Цветное цифровое изображение



3. Постоянство цвета и освещения. Коррекция контраста и цветности - линейное растяжение, серый мир, метод блика, гамма-коррекция, адаптация «Von Kries», цветовые шаблоны.

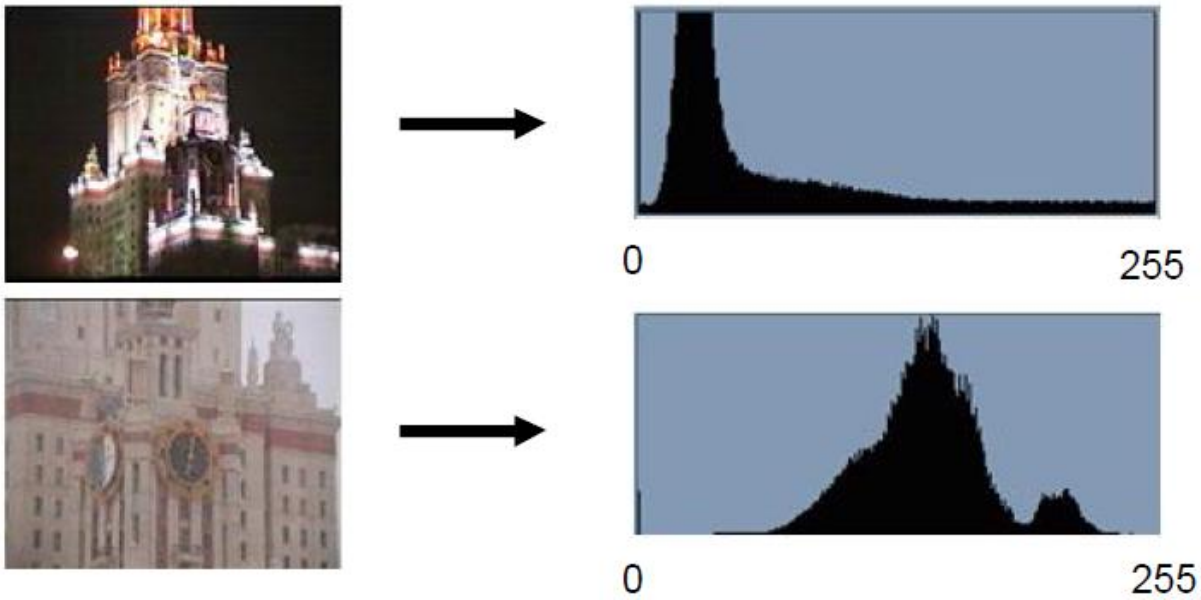
Постоянство цвета и освещения.
Способность зрительской системы человека оценивать собственные отражательные свойства поверхностей в не зависимости от условий освещенности. Пример: белый цвет на свету и в тени.

Коррекция контраста.
Почему изображение может быть слабоконтрастным?

1. Ограниченный диапазон чувствительности датчика.
2. «Плохая» функция передачи датчика.

Способ определения контрастного изображения:

Гистограмма яркости – это график распределения яркостей на изображении. На горизонтальной оси – шкала яркостей тонов от белого до черного, на вертикальной оси – число пикселей заданной яркости.
Пример:



Слайд 10 (Лекция 2) Формула изменения контраста изображения.
Слайд 11(Лекция2) Формула линейной коррекции.
Слайд 27(Лекция 2) Серый мир, следующие слайды – примеры.
Слайд 31(Лекция 2) Модель Блика.

Нелинейная коррекция.
Нелинейная компенсация недостаточной контрастности.

Часто применяемые функции:

- Гамма-коррекция
  - Изначальная цель – коррекция для правильного отображения на мониторе.

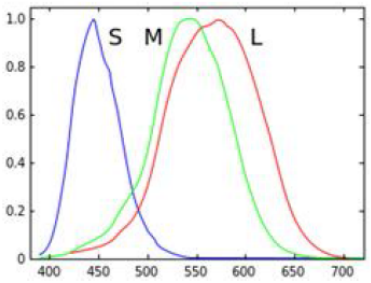
$$y = c \cdot x^{\gamma}$$

- Логарифмическая
  - Цель – сжатие динамического диапазона при визуализации данных

$$y = c \cdot \log(1 + x)$$

Адаптация «Von Kries».

- Модель коррекции, основанная на человеческом восприятии
- У человека может меняться чувствительность каждого типа колбочек
- Схема алгоритма:
  - Перевести из RGB в LMS пространство
  - Вычислить преобразование
  - Перевести обратно



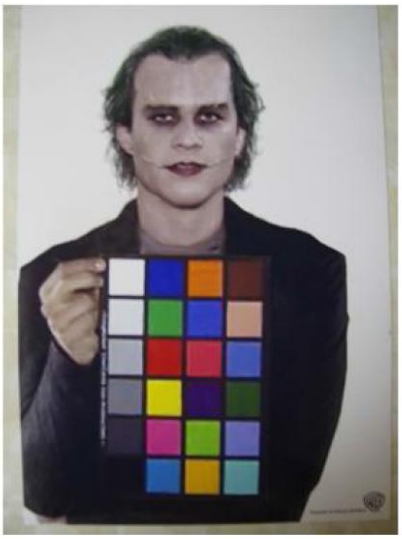
- L(long)
- M(medium)
- S(short)

L = 0.3811*R + 0.5783*G + 0.0402*B
M = 0.1967*R + 0.7244*G + 0.0782*B
S = 0.0241*R + 0.1288*G + 0.8444*B

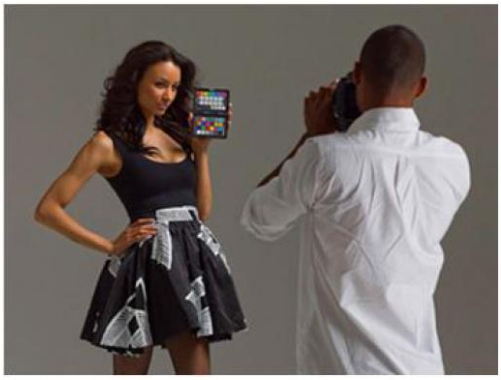
R = 4.4679*L - 3.5873*M + 0.1193*S
G = -1.2186*L + 2.3809*M - 0.1624*S
B = 0.0497*L - 0.2439*M + 1.2045*S

Цветовые шаблоны.

Профессиональная цветокоррекция



Source: The dark knight



Source: http://x-rite.com

Используем цветной шаблон с многими цветами

Какое преобразование в камере?

4.Виды шумов на изображениях и методы их подавления. Линейные фильтры, их свойства и примеры. Медианный фильтр.

Виды шума:

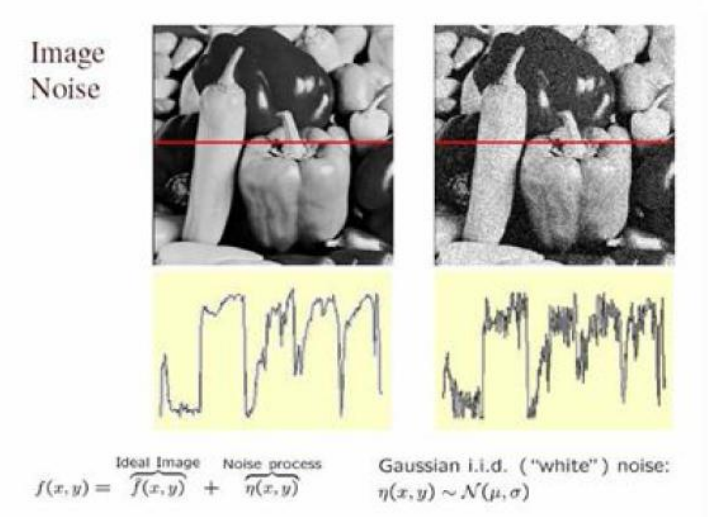
Соль и перец: случайные черные и белые пиксели

• **Импульсный:** случайные белые пиксели

• **Гауссов**: колебания яркости, распределенные по нормальному закону (слайд 27, лекция 2)

Гауссов шум

- Мат.модель: сумма множества независимых факторов
- Подходит при маленьких дисперсиях
- Предположения: независимость, нулевое матожидание



Подавление шума.

Усреднение нескольких кадров слайд 39(Лекция 2)
Подавление гауссова шума слайд 58(лекция 2)
Подавление шума «соль и перец» фильтром Гаусса слайд 62

Свойства фильтра Гаусса

- Свертка с сами собой дает тоже фильтр гаусса
  - Сглаживание несколько раз фильтром с маленьким ядром дает результат, аналогичный свертке с большим ядром
  - Свертка 2 раза с фильтром радиуса σ дает тот же результат, что с фильтром радиуса $\sigma\sqrt{2}$

Основные свойства

- **Линейность:** filter(f_1 + f_2) = filter(f_1) + filter(f_2)
- **Инвариантность к сдвигу:** не зависит от сдвига пиксела: filter(shift(f)) = shift(filter(f))
- Теория: любой линейный оператор, инвариантный к сдвигу, может быть записан в виде свертки

- Чтобы доказать нелинейность фильтра, можно воспользоваться основными свойствами, и показать их не выполнение на примере

Простейшие фильтры:

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Original

0	0	0
0	0	1
0	0	0

Shifted left
By 1 pixel

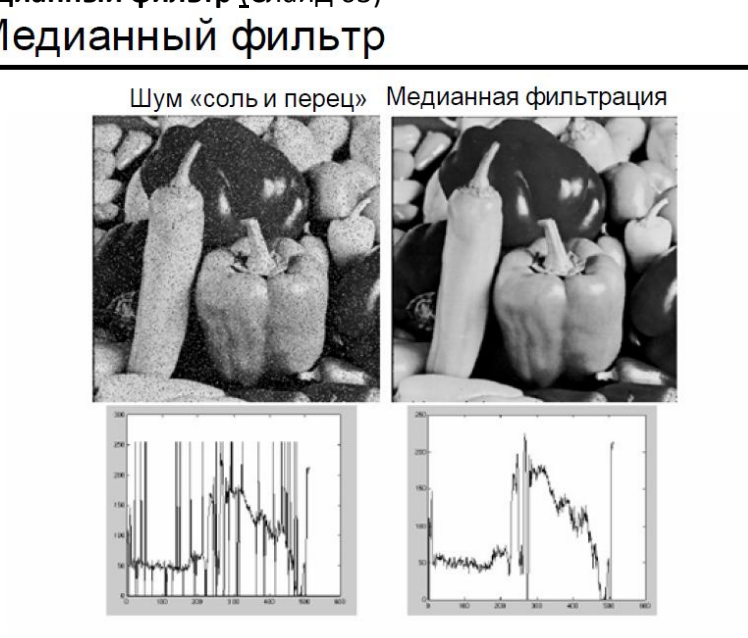
Original

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Blur (with a
box filter)

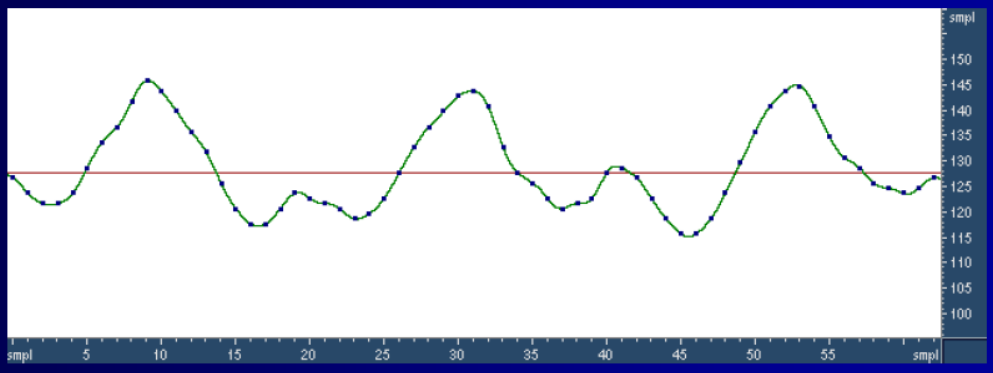
Медианный фильтр (Слайд 63)



5. **Дискретизация сигналов. Теорема Котельникова. Наложение спектров. Восприятие звука**

Оцифровка сигналов

1. Дискретизация по времени (аргумент функции)
2. Квантование по амплитуде (значение функции)



- АЦП (ADC) – аналогово-цифровой преобразователь
  Параметры: частота дискретизации, разрядность квантования (пример: 44.1 кГц, 16 бит – формат Audio CD)

Оцифровка сигналов

MSU GRAPHICS & MEDIA LAB

- При каких условиях по цифровому сигналу можно точно восстановить исходный аналоговый?
- Предположим, что значения амплитуд в цифровом сигнале представлены точно
- Введем понятие спектра аналогового сигнала:

$$X(\nu) = \int_{-\infty}^{+\infty} x(t) \cdot e^{-2\pi i \nu t} dt \qquad x(t) = \int_{-\infty}^{+\infty} X(\nu) \cdot e^{2\pi i \nu t} d\nu$$

(разложение на синусоиды с различными частотами)

$$e^{-pt} = \cos pt - i \cdot \sin pt$$

x(t) – исходный сигнал
X(ν) – спектр, т.е. коэффициенты при гармониках с частотой ν

Теорема Котельникова

MSU GRAPHICS & MEDIA LAB

- Пусть

1. спектр сигнала x(t) не содержит частот выше F, т.е. X(ν)=0 за пределами отрезка [-F, F]
2. дискретизация сигнала x(t) производится с частотой F_s, т.е. в моменты времени nT, здесь $T = F_s^{-1}$
3. $F_s > 2F$

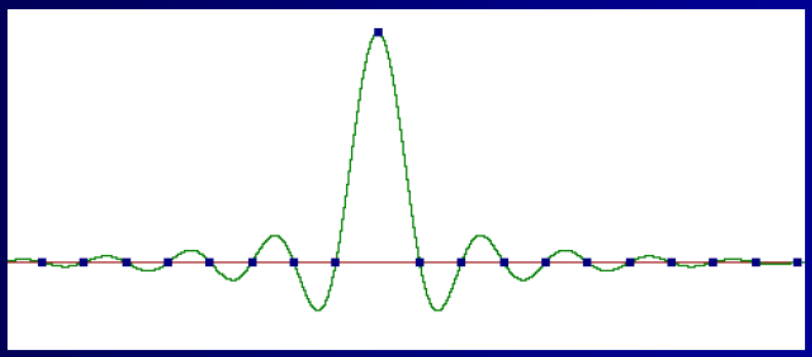
- Тогда исходный аналоговый сигнал x(t) можно точно восстановить из его цифровых отсчетов x(nT), пользуясь интерполяционной формулой

$$x(t) = \sum_{n=-\infty}^{+\infty} x(nT) \cdot Sinc(t - nT) \qquad Sinc(t) = \frac{\sin \pi F_s t}{\pi F_s t}$$

- Как выглядят интерполирующие sinc-функции?

$$x(t) = \sum_{n=-\infty}^{+\infty} x(nT) \cdot Sinc(t - nT)$$

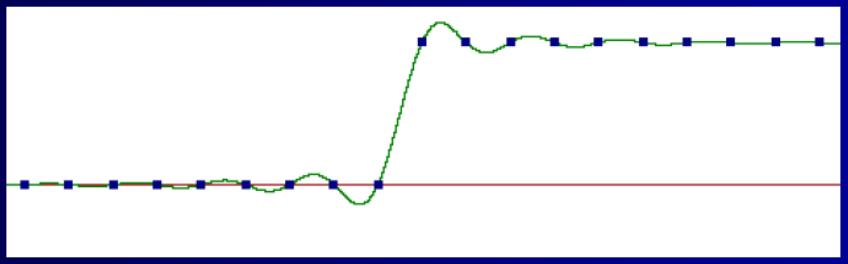
$$Sinc(t) = \frac{\sin \pi F_s t}{\pi F_s t}$$

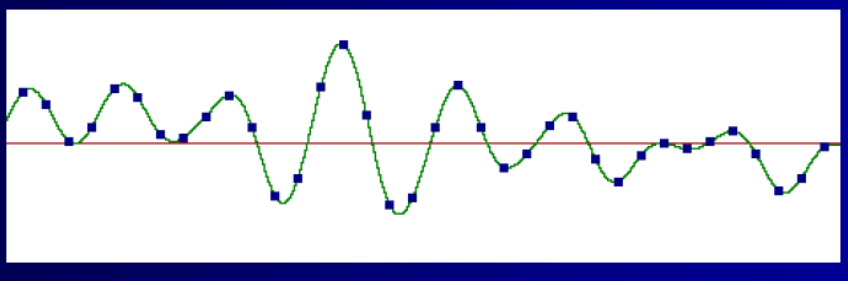


Бесконечно затухающие колебания

- Реконструкция аналоговых сигналов. Sinc-интерполяция.

$$x(t) = \sum_{n=-\infty}^{+\infty} x(nT) \cdot Sinc(t - nT)$$



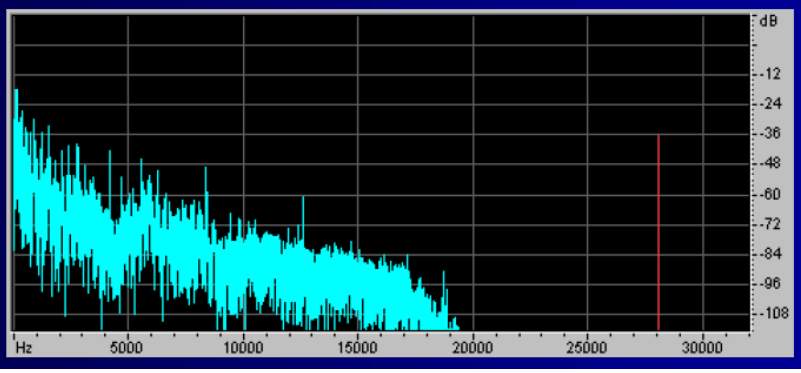


Наложение спектров

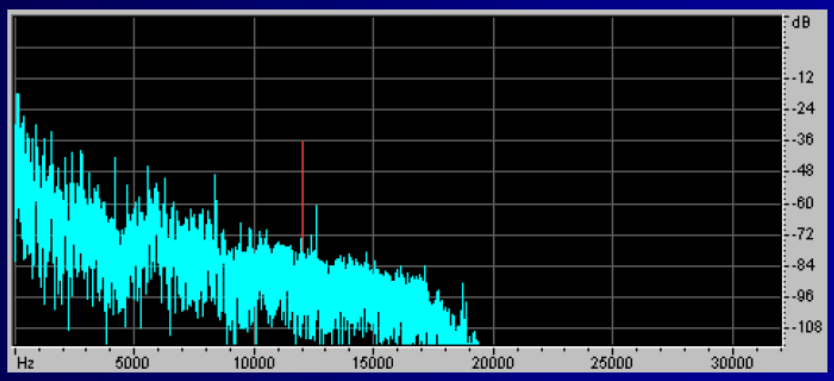
(aliasing)

MSU GRAPHICS & MEDIA LAB

- Что будет, если условия теоремы Котельникова не выполнены?
- Пусть звук не содержит частот выше 20 кГц. Тогда, по теореме Котельникова, можно выбрать частоту дискретизации 40 кГц.
- Пусть в звуке появилась помеха с частотой 28 кГц. Условия теоремы Котельникова перестали выполняться.



- Проведем дискретизацию с частотой 40 кГц, а затем – восстановим аналоговый сигнал sinc-интерполяцией.



- Помеха отразилась от половины частоты дискретизации в нижнюю часть спектра и наложилась на звук. Помеха переместилась в слышимый диапазон. Алиасинг.

- Как избежать наложения спектров?
- Применить перед оцифровкой анти-алиасинговый фильтр
  - Он подавит все помехи выше половины частоты дискретизации (выше 20 кГц) и пропустит весь сигнал ниже 20 кГц.
  - После этого условия теоремы Котельникова будут выполняться и алиасинга не возникнет.
  - Следовательно, по цифровому сигналу можно будет восстановить исходный аналоговый сигнал.

Звук и слух

- Диапазон звуковых сигналов и пороги восприятия

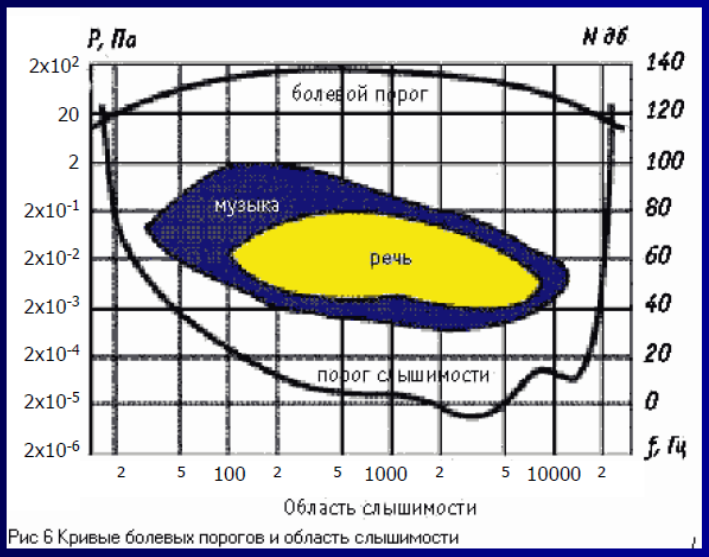


Рис 6 Кривые болевых порогов и область слышимости

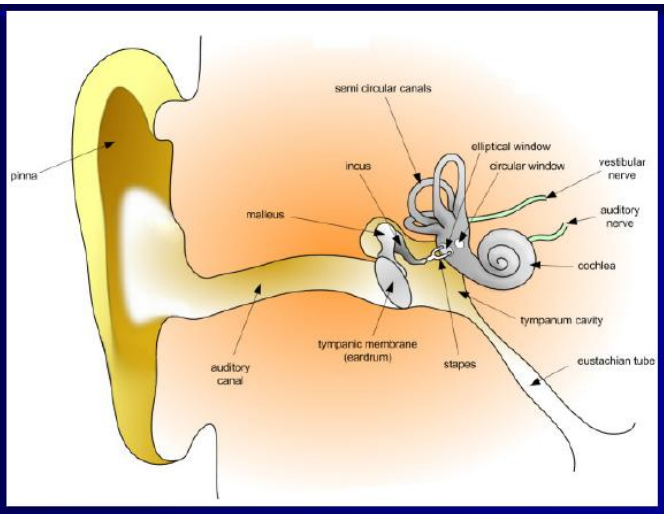


Image from Wikipedia

- Звуковые волны поступают на улитку, возбуждая ее колебания
- Жесткость улитки меняется с расстоянием, поэтому каждая часть резонирует в своем частотном диапазоне

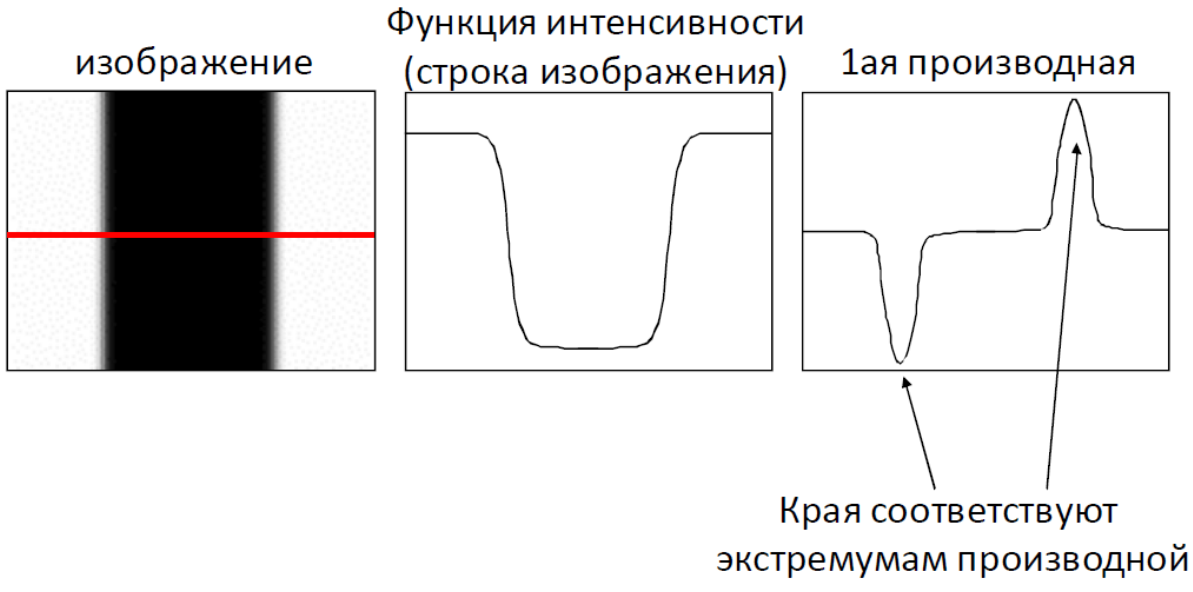
6. Дискретное преобразование Фурье. БПФ. Спектральный анализ, спектрограммы. Быстрая свертка.

Смотри лекция 5.

Дискретное преобразование Фурье. Слайды 25-30,36,37
Спектральный анализ Слайды 31-35,38-44
Быстрая свертка Слайды 45,46

7. Поиск краёв на изображении, алгоритм Canny. Сопоставление шаблонов с использованием краёв, используемые метрики. Дистантное преобразование.

Край – это точка резкого изменения значений функции интенсивности изображения.



Сопоставление шаблонов:

Сопоставление шаблонов

- Фиксируем изображение объекта (шаблон – pattern)
- Будем искать объект в кадре, «прикладывая» шаблон к изображению во всех возможных точках
- Попиксельно будем сравнивать шаблон и фрагмент нового кадра с помощью какой-нибудь метрики
- Например:

$$\frac{1}{n-1}\sum_{x,y}\frac{(f(x,y)-\overline{f})(t(x,y)-\overline{t})}{\sigma_f\sigma_t}$$

(NCC) Normalized cross correlation

Детектор Canny

1. Свертка изображения с ядром – производной от фильтра гаусса
2. Поиск значения и направления градиента
3. Выделение локальных максимумов (Non-maximum suppression)
   - Утоньшение полос в несколько пикселей до одного пикселя
4. Связывание краев и обрезание по порогу (гистерезис)
   - Определяем два порога: нижний и верхний
   - Верхний порог используем для инициализации кривых
   - Нижний порог используем для продолжения кривых

Основные этапы алгоритма

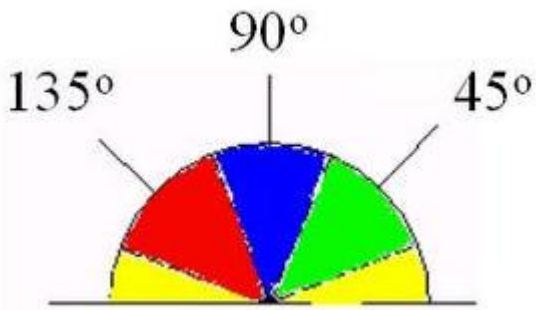
Сглаживание. Размытие изображения для удаления шума. Оператор Кэнни использует фильтр который может быть хорошо приближен к первой производной гауссианы. σ = 1.4:

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}.$$

Поиск градиентов. Границы отмечаются там, где градиент изображения приобретает максимальное значение. Они могут иметь различное направление, поэтому алгоритм Кэнни использует четыре фильтра для обнаружения горизонтальных, вертикальных и диагональных ребер в размытом изображении.

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$
$$\mathbf{\Theta} = \arctan\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right).$$

Угол направления вектора градиента округляется и может принимать такие значения: 0, 45, 90, 135.



Направление градиента

Подавление немаксимумов. Только локальные максимумы отмечаются как границы.

Двойная пороговая фильтрация. Потенциальные границы определяются порогами.

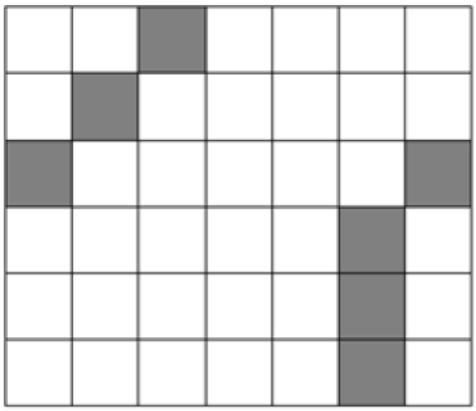
Трассировка области неоднозначности. Итоговые границы определяются путём подавления всех краёв, несвязанных с определенными (сильными) границами.

Перед применением детектора, обычно преобразуют изображение в оттенки серого, чтобы уменьшить вычислительные затраты. Этот этап характерен для многих методов обработки изображений.

Дистантное преобразование.

Distance Transform

«Дистантное преобразование»



2	1	0	1	2	3	2
1	0	1	2	3	2	1
0	1	2	3	2	1	0
1	2	3	2	1	0	1
2	3	3	2	1	0	1
3	4	3	2	1	0	1

Для каждого пикселя вычисляется расстояние до ближайшего пикселя края

- Многопроходный алгоритм (пометить соседей, потом их соседей и т.д.)
- Двухпроходный алгоритм

8. Бинаризация изображений, выделение связанных компонент, математическая морфология.

Пороговая фильтрация (thresholding):Пиксели, которые выше/ниже некоторого порога, заданного «извне», помечаются ,ниже порога помечаются 0.Более интересный способ – определение порога автоматически, по характеристикам изображения,например,анализ гистограммы:

(пример) Анализ симметричного пика гистограммы. Применяется когда фон изображения дает отчетливый и доминирующий пик гистограммы, симметричный относительно своего центра.

Что надо делать:

1. Сгладить гистограмму;

2. Найти ячейку гистограммы hmax с максимальным значением;

3. На стороне гистограммы не относящейся к объекту (на примере – справа от пика фона) найти яркость hp, количество пикселей с яркостью >= hp равняется p% (например 5%) от пикселей яркости которых >= hmax;

4. Пересчитать порог T = hmax- (hp - hmax);

Бинарное изображение – пиксели которого могут принимать только значения 0 и 1.

Бинаризация - построение бинарного изображения по полутоновому / цветному.

Бинаризация изображений, т.е. перевод полноцветного или в градациях серого изображения в монохромное, где присутствуют только два типа пикселей (темные и светлые) имеет большое значение при распознавании образов.Особенно это относится к бинарным объектам, таким, как штриховые коды, текст, чертежи и т.п. Существуют различные подходы к бинаризации, которые условно можно разделить на 2 группы:пороговые,адаптивные.Если говорить кратко, то пороговые методы бинаризации работают со всем изображением,находя какую-то характеристику (порог), позволяющую разделить все изображение на чёрное и белое.Адаптивные методы работают с участками изображений и используются при неоднородном освещении объектов.

Но в бинарных изображениях мы можем встретить неприятность в виде шума,с которой можно повоевать оружием математической морфологии.

Математическая морфология (ММ) — (Морфология от греч. μορφή «форма» и λογία «наука») — теория и техника анализа и обработки геометрических структур, основанная на теории множеств, топологии и случайных функциях. В основном применяется в обработке цифровых изображений, но также может быть применима на графах, полигональной сетке, стереометрии и многих других пространственных структурах.В бинарной морфологии двоичное изображение, представленное в виде упорядоченного набора (упорядоченного множества) черно-белых точек (пикселей), или 0 и 1. Под областью изображения обычно понимается некоторое подмножество точек изображения. Каждая операция двоичной морфологии является некоторым преобразованием этого множества. В качестве исходных данных принимаются двоичное изображение B и некоторый структурный элемент S. Результатом операции также является двоичное изображение.

Для устранения шума нам будут полезны операции :

Множество A обычно является объектом обработки,множество B (называемое структурным элементом) – инструмент обработки.

Сужение (erosion). A (-) B = (AC(+) B)C, где AC– дополнение A. Операция эрозии полезна для удаления малых объектов и различных шумов, но у этой операции есть недостаток – все остающиеся объекты уменьшаются в размере. Этого эффекта можно избежать, если после операции эрозии применить операцию наращивания с тем же структурным элементом.

Расширение (dilation). Операция «расширение» - аналог логического «или.

Закрытие(closing). close(A, B) = (A (+) B) (-) B. Операция замыкания «закрывает» небольшие внутренние «дырки» в изображении, и убирает углубления по краям области. Если к

изображению применить сначала операцию наращивания, то мы сможем избавиться от малых дыр и щелей, но при этом произойдёт увеличение контура объекта. Избежать этого увеличения позволяет операция эрозия, выполненная сразу после наращивания с тем же структурным элементом.

Раскрытие (opening). open(A, B) = (A (-) B) (+) B. Размыкание отсеивает все объекты, меньшие чем структурный элемент, но при этом помогает избежать сильного уменьшения размера объектов. Также размыкание идеально подходит для удаления линий, толщина которых меньше, чем диаметр структурного элемента. Также важно помнить, что после этой операции контуры объектов становятся более гладкими.

Результат морфологических операций во многом определяется применяемым структурным элементом.

Выбирая различный структурный элемент можно решать разные задачи обработки изображений: шумоподавление,выделение границ объекта,выделение скелета объекта,выделение сломанных зубьев на изображении шестерни.

С помощью морфологии можно убрать тонкие линии,точки.Если хотите посмотреть на картинки-то слайды лекции вам в помощь(анализ изображения часть1)

Выделение связных областей.

Связная область-множество пикселей, у каждого пикселя которого есть хотя бы один сосед, принадлежащий данному множеству.

Более симпатичное определение через связанность пикселей:

Два пикселя (B или W) называются связными, если они являются соседями(расстояние между ними равно 1) в выбранной метрике. Определение связности симметрично для черных и белых пикселей. Связная компонента (connected component) изображения (СК) – это связное множество пикселей в соответствии с выбранным типом метрики.

Под маркировкой связных компонент бинарного изображения B будем понимать формирование маркированного изображения LB, в котором каждому пикселю присвоена метка связной компоненты, которой принадлежит данный пиксель.

Метка представляет собой некоторый идентификатор, используемый в качестве уникального имени объекта. Хотя иногда применяются символьные метки, для маркировки связных компонент часто более удобными оказываются метки в виде положительных целых чисел. Под выделением связных компонент понимают присвоение уникальной метки каждому объекту изображения . При последующем анализе данные метки служат в качестве идентификаторов при обращении к объектам. **(в слайдах есть куски кода, осуществляющие маркировку)**

9. Сегментация изображений - последовательное сканирование, k-средних. Понятие текстуры. Признаки областей для распознавания объектов

Сегментация - это способ разделения сцены на «куски», с которыми проще работать.Для записи результата сегментации сделаем карту разметки – изображение, в каждом пикселе которого номер сегмента, которому принадлежит этот пиксель.Визуализировать удобно каждый сегмент своим цветом.

Более «умное» определение, вики нас спасет: в компьютерном зрении, **сегментация** — это процесс разделения цифрового изображения на несколько сегментов (множество пикселей, также называемых суперпикселями). Цель сегментации заключается в упрощении и/или изменении представления изображения, чтобы его было проще и легче анализировать. **Сегментация** изображений обычно используется для того,чтобы выделить объекты и границы (линии, кривые, и т. д.) на изображениях. Более точно,сегментация изображений — это процесс присвоения таких меток каждому пикселю изображения, что пиксели с одинаковыми метками имеют общие визуальные характеристики.Результатом сегментации изображения является множество сегментов, которые вместе покрывают всё изображение, или множество контуров, выделенных из изображения. Все пиксели в сегменте похожи по некоторой характеристике или вычисленному свойству, например по цвету, яркости или текстуре.

Метод К-средних.

Целью задачи кластеризации является разбиение множества объектов на группы (кластеры) на основе некоторой меры сходства объектов(в случае сегментации объекты это пиксели)

Это итеративный метод, который используется, чтобы разделить изображение на К кластеров. Базовый алгоритм приведён ниже:

1. Выбрать К центров кластеров, случайно или на основании некоторой эвристики

2. Поместить каждый пиксель изображения в кластер, центр которого ближе всего к этому

пикселю.

3. Заново вычислить центры кластеров, усредняя все пиксели в кластере

4. Повторять шаги 2 и 3 до сходимости (например, когда пиксели будут оставаться в том же кластере)

Здесь в качестве расстояния обычно берётся сумма квадратов или абсолютных значений разностей между пикселем и центром кластера. Разность обычно основана на цвете, яркости,текстуре и местоположении пикселя, или на взвешенной сумме этих факторов. К может быть выбрано вручную, случайно или эвристически.Этот алгоритм гарантированно

сходится, но он может не привести к оптимальному решению.Качество решения зависит от начального множества кластеров и значения K.

Этот метод:

-Однопараметрический. Требует знания только о количестве кластеров

-Рандомизирован. Зависит от начального приближения

-Не учитывает строения самих кластеров

Последовательное сканирование(для нахождения однородных областей).

Сканируем сверху вниз,слева на право.

1. if I(A) – Iavg(Cl(B)) > δ and I(A) – Iavg(Cl(C)) > δ -

создаем новую область, присоединяем к ней пиксел A

2. if I(A) – Iavg(Cl(B)) < δ xor I(A) – Iavg(Cl(C)) < δ –добавить A к одной из областей

3. if I(A) – Iavg(Cl(B)) < δ and I(A) – Iavg(Cl(C)) < δ :

3.1. Iavg(Cl(B)) - Iavg(Cl(C)) < δ –сливаем области B и C.

3. 2. Iavg(Cl(B)) - Iavg(Cl(C)) > δ–добавляем пиксел A к тому классу, отклонение от которого минимально.

I(A) – яркость пиксела A

Cl(B) – область к которой принадлежит пиксел B

Iavg(Cl(B)) – средняя яркость области к которой принадлежит B

Для анализа требуется вычислить некоторые числовые характеристики (признаки) областей:геометрические признаки, фотометрические признаки.На основе этих характеристик можно классифицировать получаемые области.

Геометрические признаки - некий набор простейших числовых характеристик (площадь,центр масс,периметр(количество пикселей принадлежащих границе области),компактность(отношение квадрата периметра к площади,ориентацию главной оси инерции ,удлиненность (эксцентриситет)).

Фотометрические прзнаки. Для каждой области можно подсчитать некий набор простейших числовых характеристик: средняя яркость, средний цвет (если изображение цветное),гистограмма распределения яркостей (или три гистограммы распределения R, G, B),дисперсию (разброс) яркостей или цвета, текстурные признаки. Разумеется, все это считается по исходному, а не бинарному изображению!

Общее понятие текстуры – это специфика поверхности, особенности ее визуального восприятия и тактильного, в случае, если поверхность можно потрогать. Это понятие

появилось в данной теме потому, что человек для анализа изображения использует не только цвет и яркость пикселя, но и ориентацию краев(градиентов изображения),а так же их распределение. В первичной визуальной коре головного мозга есть клетки, чувствительные к краям определенной ориентации. Для каждой области есть набор таких клеток, чувствительные к краям разной ориентации. Чтобы **анализировать текстуру**, берут фильтры, чувствительные краю определенной ориентации ,сглаживают результат фильтрации ,и в итоге получаются подсвечены области, содержащие текстуру с краями заданной ориентации.

Есть еще такая штука как **банки фильтров**. Берут несколько фильтров разного масштаба и ориентации(это и есть банк фильтров).Каждый пиксель изображения после обработки банком фильтров даёт вектор признаков. Этот вектор признаков эффективно описывает локальную текстуру окрестности пикселя. Это активно используется в сегментации, распознавании изображений и т.д.

10. Задача классификации образов. Общий и эмпирический риск. Метод опорных векторов. Виды ошибок. Оценка качества классификаторов. Удерживание, скользящий контроль. ROC-кривая.

Задача классификации образов.

Классификация основывается на прецедентах**. Прецедент** – это образ, правильная классификация которого известна. **Прецедент** – ранее классифицированный объект, принимаемый как образец при решении задач классификации. Идея принятия решений на основе прецедентности – основополагающая в естественно-научном мировоззрении. Будем считать, что все объекты или явления разбиты на конечное число классов. Для каждого класса известно и изучено конечное число объектов – прецедентов. Задача распознавания образов состоит в том, чтобы отнести новый распознаваемый объект к какому-либо классу.

Измерения, используемые для классификации образов, называются признаками. Признак – это некоторое количественное измерение объекта произвольной природы. Совокупность признаков, относящихся к одному образу, называется вектором признаков. Вектора признаков принимают значения в пространстве признаков. В рамках задачи распознавания считается, что каждому образу ставится в соответствие единственное значение вектора признаков и наоборот: каждому значению вектора признаков соответствует единственный образ. Классификатором или решающим правилом называется правило отнесения образа к одному из классов на основании его вектора признаков.

Задача построить функцию y=f(x), которая для каждого вектора-признаков x даёт ответ y, какому классу принадлежит объект x-это задача построить классификатор(функция f(x)).Любое решающее правило (классификатор) делит пространство признаков на решающие регионы, разделенные решающими границами.

Чтобы классификация была адекватной, нужно подобрать хорошие параметры, для которых потери на новых данных будут минимальны - в этом есть смысл обычения. Насчет функции потерь данных, оценки рисков и прочих прелестей матстата можно(нужно!) глянуть в слайдах анализ изображений часть2)

Метод опорных векторов.

Метод опорных векторов (англ. SVM, support vector machine) — набор схожих алгоритмов вида «обучение с учителем», использующихся для задач классификации и регрессионного анализа. Этот метод принадлежит к семейству линейных классификаторов. Особым свойством метода опорных векторов является непрерывное уменьшение эмпирической ошибки классификации и увеличение зазора. Поэтому этот метод также известен как метод классификатора с максимальным зазором. Основная идея метода опорных векторов — перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей наши классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

Проще говоря, надо найти гиперплоскость, разделяющую «плохие» и «хорошие» примеры, и чтобы отступ между «плохими» и «хорошими» был максимальный.(это касательно линейного классификатора)

Чтобы классификатором хорошо работал на всех данных, надо минимизировать общий риск(слайды в помощь),но считать его напрямую на неограниченном множестве невозможно, поэтому его минимизируют эмпирический риск на ограниченном множестве(на обучающей выборке).

Чтобы оценить предсказательную способность классификатора, используют удерживание или скользящий контроль.

Основная идея в том, чтобы разбить обучающую выборку на 2 части: на одной обучают классификатор(минимизируют эмпирический риск),на другой измеряют ошибки обученного классификатора(т е оценивают предсказательную способность).

Удерживание: есть набор данных с известными ответами, мы это множество разделяем на 2 множества непересекающихся, и одно юзаем для обучения, другую для контроля.

Этот способ быстро и просто рассчитывается, но некоторые сложные прецеденты могут попасть только в одну из выборок и тогда оценка ошибки будет смещенной.

Скользящий контроль: выборку разделяют на d непересекающихся частей и поочередно используют одну из этих частей для контроля,остальные-для тренировки.

Что касается видов ошибок и прочего - это все просто написано в слайдах и можно даже не заморачиваться с поиском сторонней инфы.

11. Выделение объектов на изображении с использованием гистограмм ориентированных градиентов и линейного метода SVM. Скользящее окно. Повышение качества распознавания за счет обработки обучающей выборки.

Бинарная классификация нам не поможет, потому что она может дать ответ на вопрос «есть пешеход или нет?». Нам же нужно еще знать, а где он на изображении затаился. Поэтому воспользуемся такой схемой: берем окно определенного размера, сканируем изображение этим окном, и к каждому окну применяем классификатор»пешеход?».Скользящее окно-форма сегментации изображения. Посчитаем вектор-признаки. Например, ориентацию градиента в каждом пикселе. Но их будет много, поэтому для удобства можно посчитать какие-то статистики (например,гистограммы ориентации градиентов).Соберем обучающую выборку, с изображениями ,где есть объекты, что мы детектим и где нет. Для каждого фрагмента посчитаем вектор признак и сопоставим метку, к какому классу принадлежит(с нужным объектом или нет).На этой выборке обучем классификатор. Каждый опорный вектор-вектор признак одного из примеров. Для улучшения качества классификатора можно поколдовать над обучающей выборкой: бутстраппинг (на первой стадии берём случайные окна для фона. На следующих стадиях выбираем ложные срабатывания детектора как «трудные» примеры) и размножение выборки(шевеление, т.е. перемешиваем, то 1 часть берем, на ней обучаем, на другой проверяем, потом наоборот и тд)

Кто делал задание, все поймет, кто не делал - печалька, слайды в помощь, в самом конце)

12. Растеризация прямых и окружностей. Алгоритм Брезенхема

13. Сплайновые кривые. Кривые Безье. G(0) и G(1)-непрерывность. Поверхности Безье.

Билет 12 и 13-это сплошные картинки, интуитивно понятные. Слайдов выше крыши. Лекции одноименные.

14. Компьютерная графика. Понятие о графическом процессе. Понятие о геометрическом моделировании. Типы моделей, особенности их получения.

Компью́терная гра́фика (также маши́нная графика) — область деятельности, в которой компьютеры используются в качестве инструмента как для синтеза (создания) изображений, так и для обработки визуальной информации, полученной из реального мира. Изучает модели и алгоритмы синтеза.

Графический процесс заключается в следующем: есть модель(геометрическая модель, модель освещения, модель анимации) к ней применяют алгоритм синтеза(получение проекций,заливка)=>изображение(гамма,цветокоррекция,полутонирование).В геометрическое моделирование входят методы получения, представления и обработки моделей. Получить модель можно 3мя основными методами: ручное моделирование(пакеты моделирования Maya, AutoCad и т.д), автоматизированное моделирование(3-д сканирование, реконструкция по фотографии),библиотеки модели(повторное использование созданных моделей).

Модель освещения складывается из свойств материала(параметрические модели,модели отражения\пропускания,ручное моделирование,фотографии\сканы)) и модели освещения(источники света:ручное моделирование или фотографии HDR)

Алгоритмы синтеза изображений решают задачу создания изображения по набору моделей. Результирующее изображение выводится в:

-буфер кадра (монитор):преобразование изображения в излучение, гамма-коррекция, цветокоррекция

– Файл: сжатие изображений или видео

– Принтер: полутонирование и т.п.

Вообще, **модель** – это абстрактное представление сущности реального мира. Данные о физических объектах не могут быть целиком введены в компьютер, поэтому нужно ограничить объем хранимой информации об объекте. Нужно найти вид модели, наилучшим образом отвечающей решаемой задаче.

В комп.графике применяется **геометрическое моделирование**- моделирование объектов различной природы с помощью геометрических типов данных. Принципы выбора модели: соответствие поставленной задаче, максимально использовать возможности графической системы, учесть задачи обработки и редактирования модели.

Виды геометрических моделей:

- Воксельное

-Точечное представление

-Конструктивная геометрия

-Каркасное представление

- Граничное представление первого порядка

- Граничные представление высших порядков

- На основе изображений

- Гибридные модели

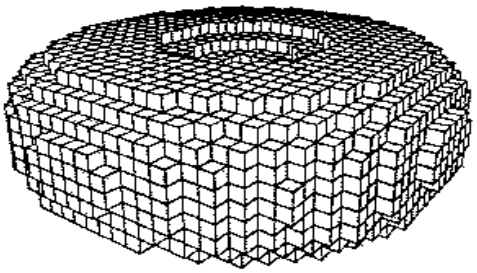
15. Воксельные модели и их свойства. Октарные деревья. Точечные представления и их свойства.

Воксельная модель – дискретизация пространства по равномерной сетке.
Воксель – аналог пикселя, только в 3D.
Структура
–Равномерная сетка, каждый элемент которой показывает, если в нем часть объекта
–Ячейка называется воксель (voxel = volume element)
–Каждый воксель принимает значение 0 или 1
–Может также задавать плотность (0-1)

Способ получения
–Дискретизация трехмерных данных на равномерной сетке

```
bool Voxels[X][Y][Z];
```

```
BYTE Voxels[X][Y][Z];
```



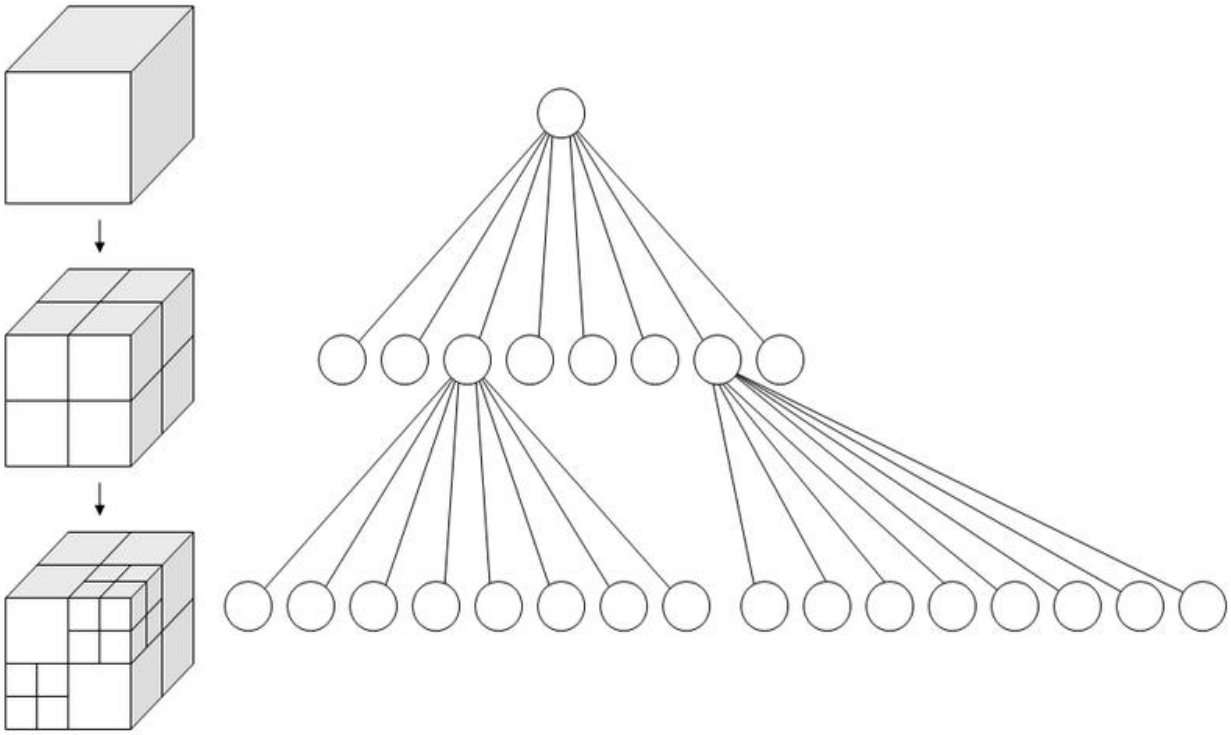
Описывает объем
•Дискретное представление: приближение реального объекта!
•Плохо описываются части объекта, не параллельные сторонам воксельного куба
Явное представление
•Размер данных пропорционален кубу разрешения сетки
1 байт на точку: 2000 x 2000 x 2000 = 7,45 Гб !

Типичные алгоритмы
Пространственные алгоритмы
–Вычисление объема объекта
–Нахождение центра масс
–Булевы операции (пересечение, объединение)

Плохо работают алгоритмы, требующие понятия поверхности!

Октодерево – иерархический вариант воксельной модели.
Октодерево (дерево октантов, англ. octree) — тип древовидной структуры данных, в которой у каждого внутреннего узла есть до восьми «потомков». Деревья октантов чаще всего используются для разделения трёхмерного пространства, рекурсивно разделяя его на восемь октантов.

•Рекурсивное разбиение пространства на восемь октант
•Ветвь дерева:
–Код = B (черный) - заполнено
–Код = W (белый) - пусто
–Код = G (серый) – частично
•Эти подразбиты на 8 потомков

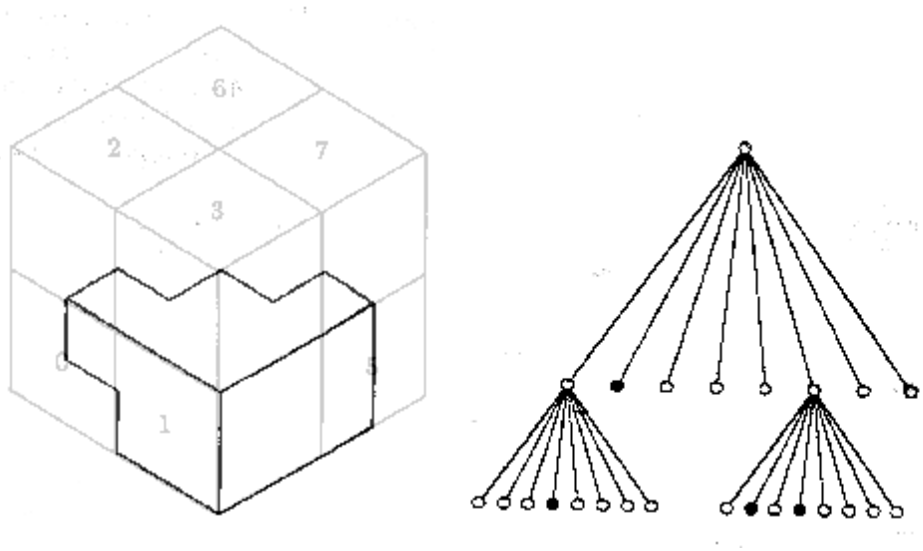


Структура данных октодерева: типичная рекурсивная иерархическая структура struct OctoTreeNode { BYTE Value; OctoTreeNode* Children[8]; };

Октодерево применяется для оптимизации воксельного представления
Свойства:
–Позволяет хранить информацию только о блоках, относящихся к объекту
–Число элементов пропорционально площади поверхности объекта, т.е. квадрату разрешения
–Для разреженных моделей позволяет уменьшить размер в тысячи раз!
Способ получения
–Из воксельного представления или напрямую, через дискретизацию

Вариант линейной записи 1: пути к листовым узлам

•Октанты дерева пронумерованы от 0 до 7
•Конструирование адреса каждой ветви дерева, кроме корня.
•Адрес ветви уровня i – последовательности i чисел от 0 до 7 – путь от корня к этой ветви
•Символ X: если в последовательности чисел меньше, чем максимальное разрешение
•Линейная запись дерева есть просто сортированный массив адресов ветвей с кодом "черный"



Линейная запись 1: {03,1X,51,53}

Вариант линейной записи 2: обход дерева в глубину

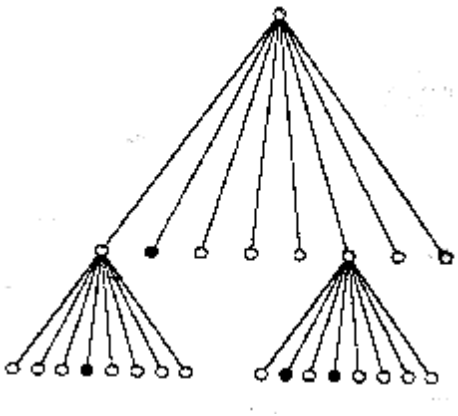
•Обход дерева в фиксированном порядке

•Трехсимвольная запись

–«B»: черная ветвь

–«W»: белая ветвь

–«(» : серая ветвь



Линейная запись 2:

((WWWBWWWWBWWW(WBWBWWWWWW

Октодерево часто применятся для хранения воксельных данных и при визуализации

•Удобно для синтеза:

–Переменный уровень детализации

–Вывод back-to-front

•Усложняются операции, требующие информации о смежных ячейках

•Сложно с анимацией, нужно перестраивать дерево

Точечное представление заменяет регулярную воксельную сетку на нерегулярную.\

•Набор неструктурированных точек

•Требования по памяти пропорциональны количеству точек

•Дискретное, явное представление

Структура точечного представления
•Массив точек с атрибутами
–Атрибуты: положение, цвет, нормаль, размер
•Описывает только принадлежащие объекту части пространства
•Явное хранение координат => возможное увеличение размера
•Нет связанности, инциндентности => для выполнения преобразований обычно строятся дополнительные структуры данных (октодерево)

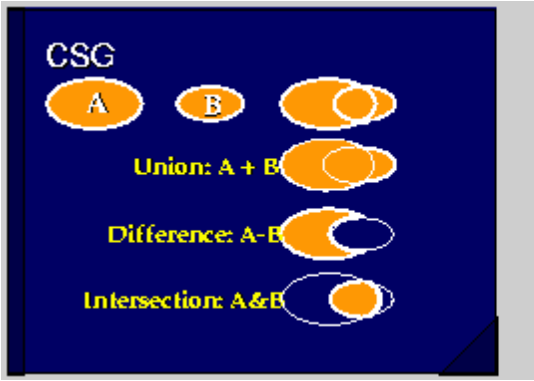
16. Конструктивная геометрия. Свойства CSG-моделей.

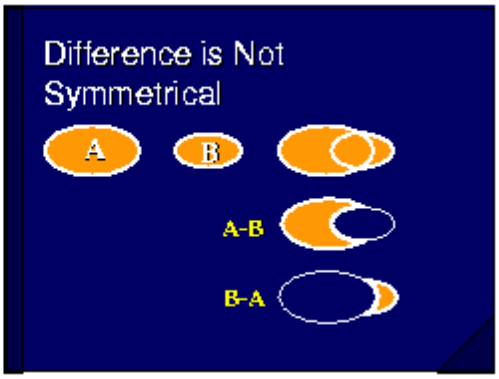
Конструктивная блочная геометрия (Constructive Solid Geometry, CSG) технология, используемая в моделировании твёрдых тел. Конструктивная блочная геометрия зачастую, но не всегда, является способом моделирования в трёхмерной графике и САПР. Она позволяет создать сложную сцену или объект с помощью битовых операций для комбинирования нескольких иных объектов. Это позволяет более просто математически описать сложные объекты, хотя не всегда операции проходят с использованием только простых тел. Так, часто с помощью конструктивной блочной геометрии представляют модели или поверхности, которые выглядят визуально сложными; на самом деле, они являются немногим более чем умно скомбинированные или декомбинированные простые объекты.

Конструктивная геометрия - эффективно описывает и поверхность и объем

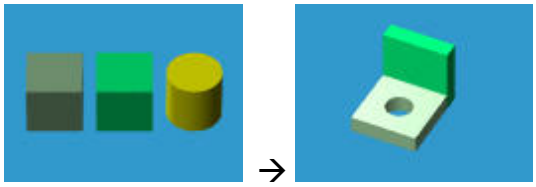
Структура
–Набор базовых примитивов - сфера, куб, цилиндр...
–Операции по их комбинированию
Способ получения
–Ручное моделирование
Свойства
–Описывает объем и поверхность (!)
–Непрерывное
–Явное

Задается набор операций для базовых примитивов:
•Перенос/поворот/масштабирование
•Теоретико-множественные:
•Объединение
•Разность
•Пересечение





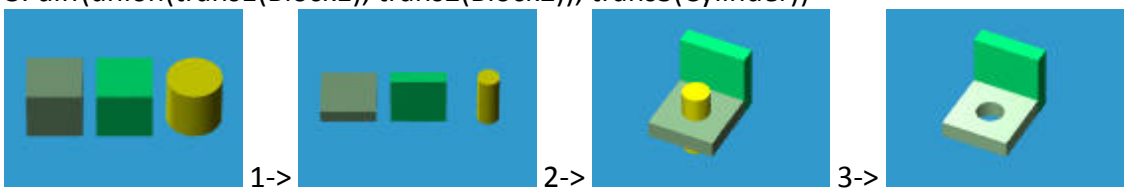
Последовательность операций задает финальный объект
diff(
union(
trans1(Block1),
trans2(Block2)
),
trans3(Cylinder)
)

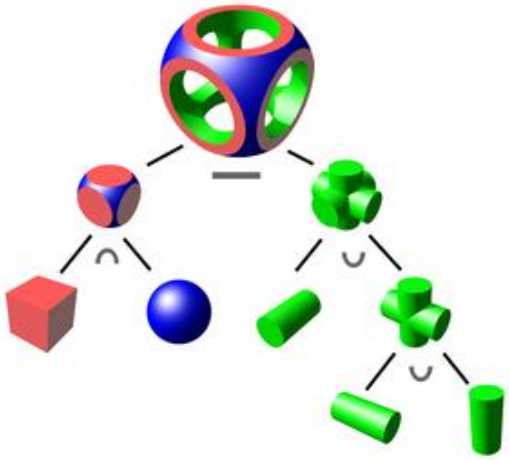


Разбор последовательности выполнения операций над телами
1. diff(union(trans1(Block1), trans2(Block2)), trans3(Cylinder))

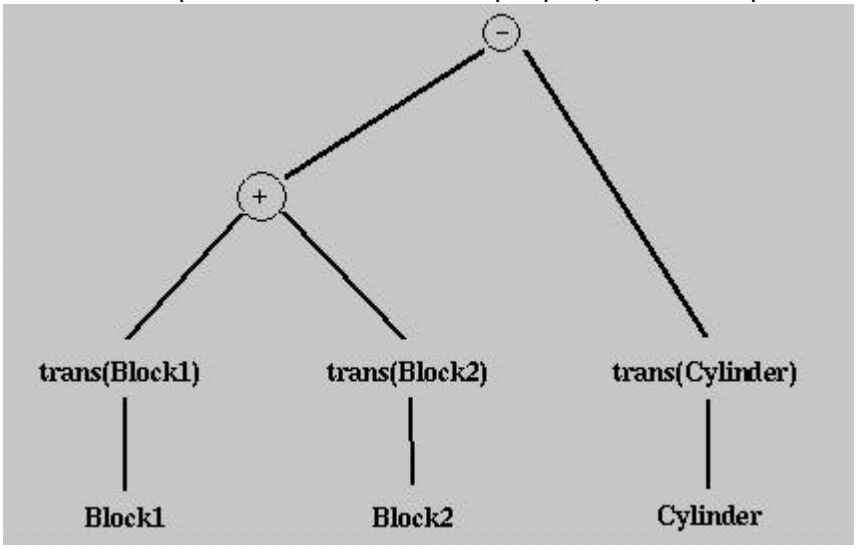
2. diff(union(trans1(Block1), trans2(Block2)), trans3(Cylinder))

3. diff(union(trans1(Block1), trans2(Block2)), trans3(Cylinder))





Структура данных для конструктивной геометрии – направленный ациклический граф
•Дерево из операций и базовых объектов
•Корень – результирующий объект
•Листья – базовые примитивы
•Число потомков равно числу операндов операции
•Из-за повторного использования превращается в направленный ациклический граф.

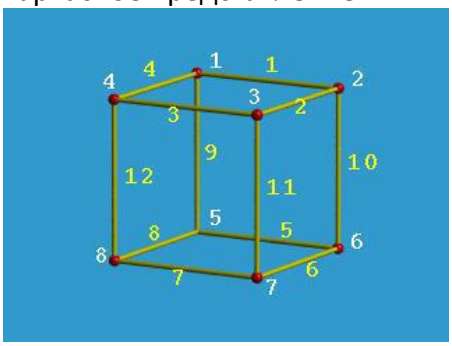


Конструктивная геометрия эффективно работает с «объемными» задачами, но также имеет понятие поверхности
Пространственные алгоритмы
–вычисление объема объекта
–нахождение центра масс
–...

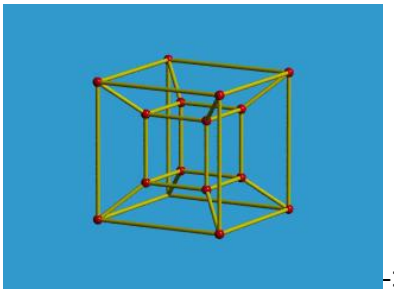
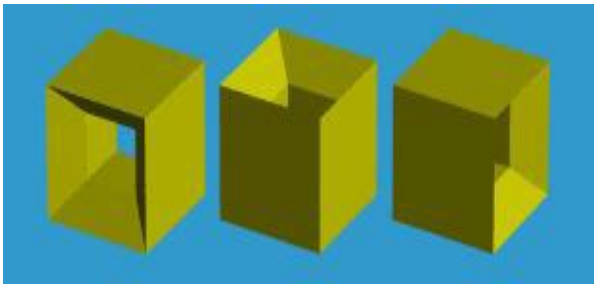
Есть понятие поверхности!
Основной недостаток – на практике можно построить только синтетически
Также сложно визуализировать

17. Геометрическое моделирование. Каркасные модели, полигональные (граничные) модели. Способы задания полигональных моделей. Свойства полигональных моделей. Представления высших порядков.

Объект можно моделировать через выделение характерных точек и ребер
Каркасное представление



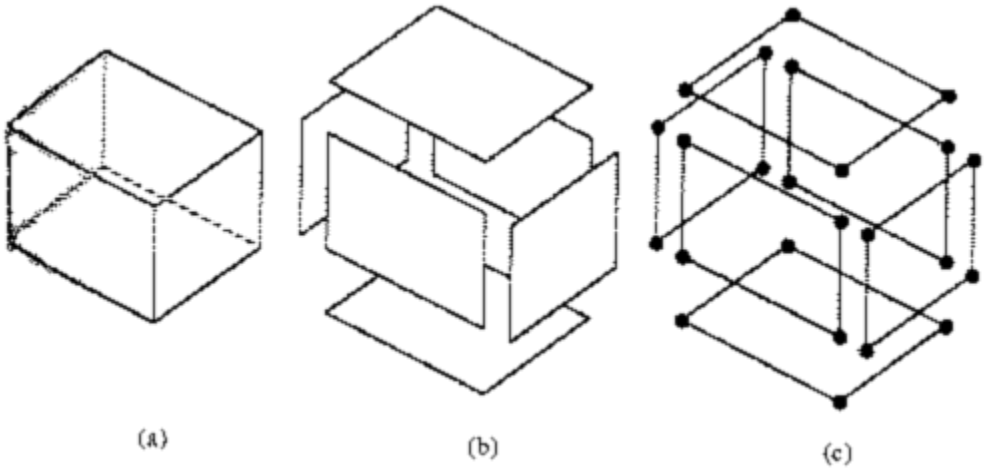
Каркасное представление: неоднозначная интерпретация для случая сплошных тел
Нужна дополнительная
информация! Как верно отобразить?


->?


Граничное представление содержит не только каркас, но грани
(?)Кусочная аппроксимация поверхности объекта
(?)Рассматриваем представления первого порядка: линейная интерполяция
Структура данных: вершины + грани

Граничные представления различаются **по способам хранения** информации о гранях и ребрах
•Явное представление
•Индексированное по вершинам
•Индексированное по ребрам
•«Крылатое» представление

Явное представление хранит список граней
•Каждая грань – полигон, состоящий из последовательности координат вершин, объект состоит из набора граней
•Недостатки
–Взаимоотношения граней заданы неявно
–Координаты вершин дублируются
–Алгоритмы поиска инцидентных ребер требуют полного перебора



Грани	Координаты
f1	x1 y1 z1, x2 y2 z2, x3 y3 z3, x4 y4 z4
f2	x6 y6 z6, x2 y2 z2, x1 y1 z1, x5 y5 z5
f3	x7 y7 z7, x3 y3 z3, x2 y2 z2, x6 y6 z6
f4	x8 y8 z8, x4 y4 z4, x3 y3 z3, x7 y7 z7
f5	x5 y5 z5, x1 y1 z1, x4 y4 z4, x8 y8 z8
f6	x8 y8 z8, x7 y7 z7, x6 y6 z6, x5 y5 z5

Индексированное по вершинам хранит индексы вершин

•Выделение координат вершин в отдельную структуру

•С гранями ассоциируются не координаты вершин, а индексы в массиве координат вершин

•Недостатки

–Аналогично явному представлению

Вершины	Координаты
v1	x1 y1 z1
v2	x2 y2 z2
v3	x3 y3 z3
v4	x4 y4 z4
v5	x5 y5 z5
v6	x6 y6 z6
v7	
v8	

Грани	Вершины
f1	v1 v2 v3 v4
f2	v6 v2 v1 v5
f3	v7 v3 v2 v6
f4	v8 v4 v3 v7
f5	v5 v1 v4 v8
f6	v8 v7 v6 v5
x7 y7 z7	
x8 y8 z8	

Индексированное по ребрам хранит индексы ребер и индексы вершин

•Грани определяются через ребра

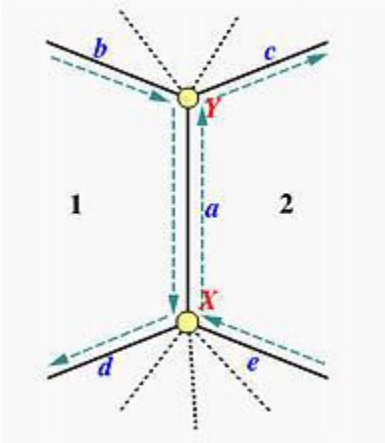
•Ребра задаются вершинами

•Вершины задаются положением в пространстве

Реб	Верш	Верш	Коорд	Грн	Ребра
e1	v1 v2	v1	x1 y1 z1	f1	e1 e2 e3 e4
e2	v2 v3	v2	x2 y2 z2	f2	e9 e6 e1 e5
e3	v3 v4	v3	x3 y3 z3	f3	e10 e7 e2 e6
e4	v4 v1	v4	x4 y4 z4	f4	e11 e8 e7 e3
e5	v1 v5	v5	x5 y5 z5	f5	e12 e5

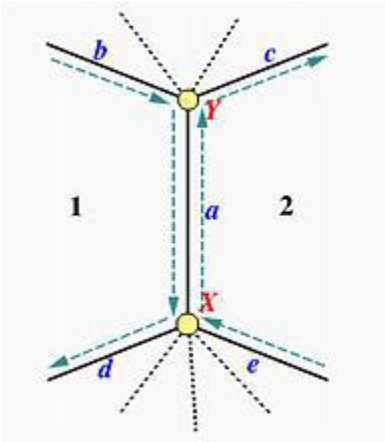
				e4 e8
e6	v2 v6	v6	x6 y6 z6	f6 e12 e11
				e10 e9
e7	v3 v7	v7	x7 y7 z7	
e8	v4 v8	v8	x8 y8 z8	
e9		v5 v6		
e10		v6 v7		
e11		v7 v8		
e12		v8 v5		

«Крылатое» представление связывает грани, ребра, вершины
•“Winded-Edge”
•Добавляется информация о взаимном расположении граней

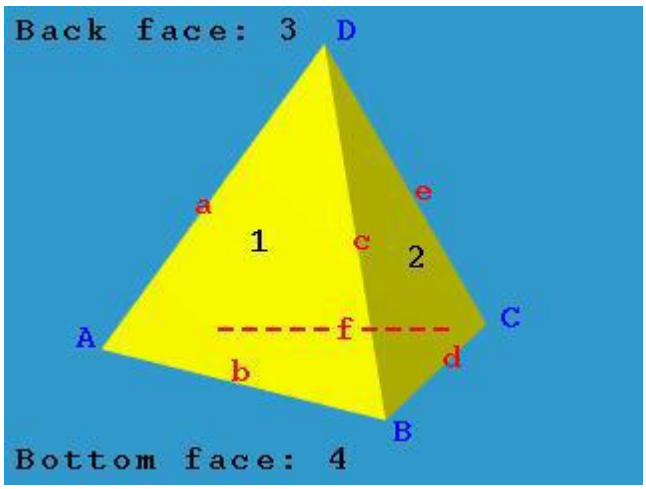


Структура данных построена вокруг ребра
struct Edge { Vertex* X; Vertex* Y; Face* FaceLeft; Face* FaceRight; Edge* LeftPred; Edge* LeftSucc; Edge* RightPred; Edge* RightSucc; };

Ребро	Вершины		Грани		Левый обход		Правый обход	
Name	Begin	End	Left	Right	Pred	Succ	Pred	Succ
a	X	Y	1	2	b	d	e	c

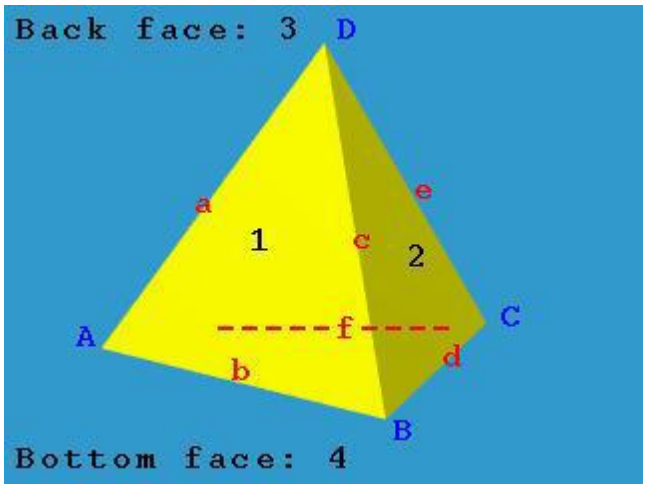


Пример основной таблицы для пирамиды



Edge	Vertices		Faces		Left Traverse		Right Tra
Name	Start	End	Left	Right	Pred	Succ	Pred
a	A	D	3	1	e	f	b
b	A	B	1	4	c	a	f
c	B	D	1	2	a	b	d
d	B	C	2	4	e	c	b
e	C	D	2	3	c	d	f
f	A	C	4	3	d	b	a

Нужны еще две таблицы: для вершин и для ребер



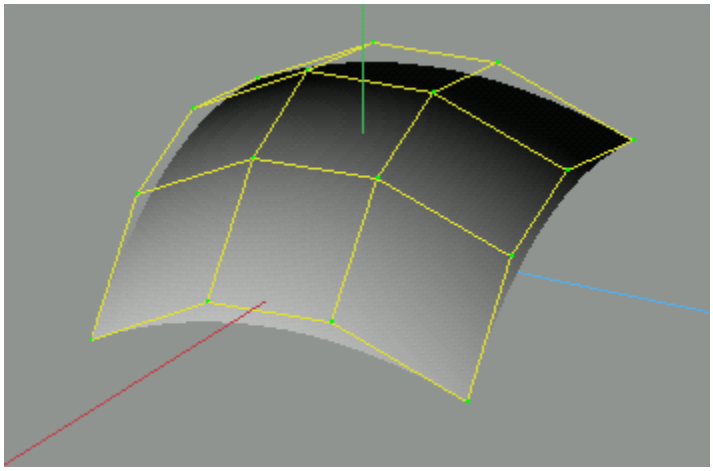
Vertex Name	Incident Edge
A	a
B	b
C	d
D	e

Face Name	Incident Edge
1	a
2	c

3 a
4 b

Граничное представление: **типичные алгоритмы**
•Проверка правильности задания
•Вычисление габаритного объема
•Вычисление нормали в точке
•Вычисление кривизны поверхности
•Нахождение точки пересечения с лучом или кривой
•Определение положения точки относительно поверхности

Граничные представления высших порядков
Контрольные точки + способ интерполяции 2-го порядка и выше (полиномы Берштейна, например)



Итоги
•Геометрическое моделирование
•Модели трехмерных объектов
–Воксельное (+октарное дерево)
–Точечное представление
–Конструктивная геометрия
–Каркасное представление
–Граничное представление (явное, индексированное по вершинам, ребрам, «крылатое»)
–Граничные представление высших порядков

18. Особенности и программная архитектура библиотеки OpenGL. OpenGL 1.x-2.x и OpenGL 3.x-4.x.

OpenGL – это программная библиотека для создания 3D-приложений

OpenGL – кросс-платформенная библиотека функций для создания интерактивных 2D и 3D приложений

Особенности:
–Стабильность (с 1992 г.): Изменения в API вносятся комитетом ARB (Architecture Review Board)
–Переносимость: Независимость от оконной и операционной системы
–Легкость применения: простой интерфейс, низкие затраты на обучение
Аналогичные библиотеки: DirectX

OpenGL – это API и реализация
Стандартизируется прикладной программный интерфейс (API)
•Реализация своя для каждой платформы
•Может существовать несколько реализаций

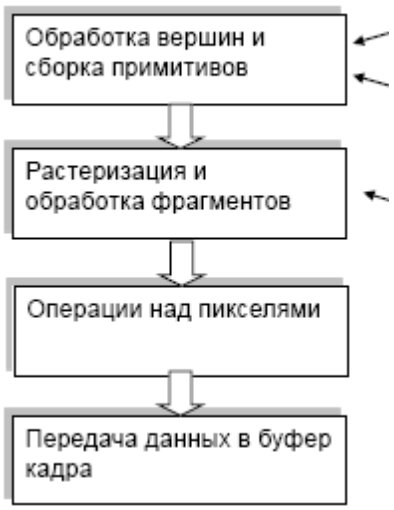
История OpenGL: четыре версии за 18 лет

OpenGL 1.x-2.x VS OpenGL 3.x-4.x
•В OpenGL 3.x+ введен новый «непосредственный» (immediate) API
•API стал более ориентирован на программируемую аппаратуру и крупные приложения
•API версий 1.x и 2.x доступен

OpenGL – прослойка между вашей программой и драйвером видеокарты
Взаимодействует с пользовательскими программами, драйвером видеокарты и с WIN API
Лекция 8, слайд 18 - схема

OpenGL состоит из набора библиотек
•AGL, GLX, WGL
–Связь между OpenGL и оконной системой
•GLU (OpenGL Utility Library)
–Часть OpenGL
–NURBS, tessellators, quadric shapes, etc
•GLUT (OpenGL Utility Toolkit)
–Переносимый оконный API
–Неофициальная часть OpenGL Вместо GLUT можно использовать SDL и другие библиотеки

Программная архитектура OpenGL может быть представлена в виде конвейера



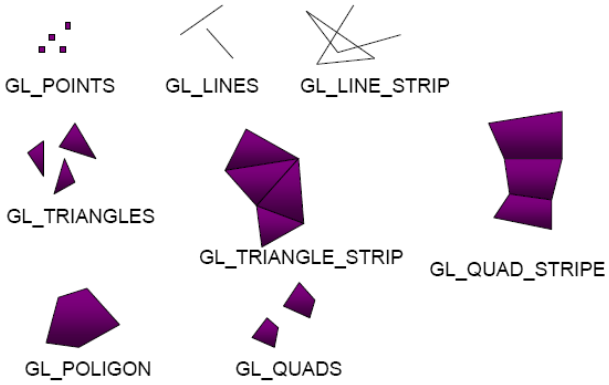
3D координаты -> экранные

Два типа функций (команд): передача данных и изменения состояния
Команды передача данных
–Объекты на экране рисуются путем последовательной передачи в конвейер вершин примитивов, которые составляют объект
Команды изменения состояния
–Настройка обработки данных на каждом этапе конвейера
–OpenGL как конечный автомат

Конвейер OpenGL. Переходим к этапу обработки вершин и сборке примитивов
1.Обработка вершин и сборка примитивов
2.Растеризация и обработка фрагментов
3.Операции над пикселями
4.Передача данных в буфер кадра

OpenGL работает с разными типами моделей, в основном с полигональными
OpenGL работает с моделями, заданными в граничном полигональном представлении
Поверхность приближается набором полигональных граней (face, polygon)
Границы граней описываются ребрами (edge)
Часть отрезка, формирующего ребро, заканчивается вершинами (vertex)

OpenGL может по-разному объединять вершины в полигоны

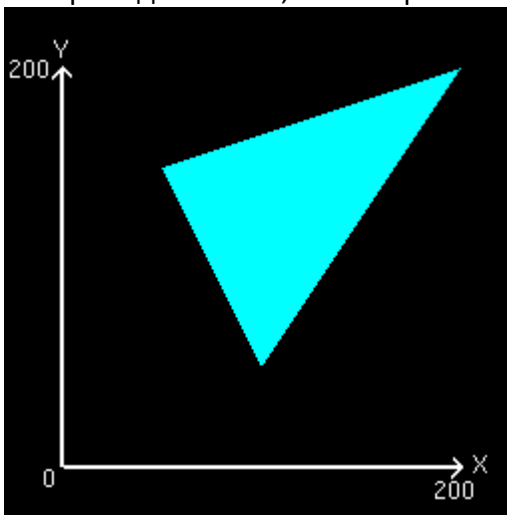


С каждой вершиной ассоциировано несколько атрибутов
Каждая вершина кроме положения в пространстве может иметь несколько других атрибутов
–Материал
–Цвет
–Нормаль
–Текстурные координаты

Внимание: всегда используется ТЕКУЩИЙ набор атрибутов
–**OpenGL как конечный автомат**

Пример кода для вывода треугольника
Цветной треугольник
–glBegin(GL_TRIANGLES);
•glColor3f(0.0f,1.0f,0.0f);
•glVertex2f(150.0f, 50 .0f);
•glVertex2f(50.0f, 150 .0f);
•glVertex2f(200 .0f, 200 .0f);
–glEnd();

Таким образом можно задать любой объект!
Теперь задача в том, чтобы правильно показать этот объект на экране



19. Синтез изображений с помощью растеризации. Свойства алгоритма. Графический конвейер, применение геометрических преобразований. Графический конвейер в OpenGL.

В OpenGL каждая вершина задается в локальной системе координат
•Каждая вершина объекта задается в локальных координатах модели
•Необходимо определить набор геометрических преобразований, таких, что каждая вершина преобразуется в точку на плоскости экрана

Обработка вершин: последовательность преобразований из локальной в экранную систему координат Три последовательных преобразования:
•модельное преобразование (ЛСК -> МСК)
•видовое преобразование (МСК -> ВСК)
•проективное преобразование (ВСК -> ЭСК)

Что такое геометрические преобразования?
Модель
–Например, описание поверхности трехмерного объекта
–Некоторое подмножество точек декартова пространства

Зачем применять преобразования к модели?
–Создание моделей (сцен) из компонент
–Редактирование моделей
–Преобразования в процессе синтеза изображений

Два класса преобразований: линейные и нелинейные
•Линейные преобразования
•Нелинейные преобразования

Нелинейные преобразования – произвольные деформации модели
Произвольное преобразование точек модели
M’=T(M)

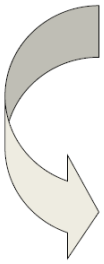
Линейные преобразования –интересующий нас класс преобразований
z’= Ix+ Jy+ Kz+ L
y’= Ex+ Fy+ Gz+ H
x’= Ax+ By+ Cz+ D
Линейное преобразование применяется к каждой точке
модели
Не изменяет топологию!

Для полигональных моделей линейные преобразования достаточно применить к вершинам!
Для полигональных моделей достаточно применить преобразование к вершинам модели!
–Линейная интерполяция

Алгоритмически эффективно, легко векторизуется
Растеризация основана на линейных преобразованиях

Преобразования можно записать в матричной форме

$$x' = Ax + By + Cz + D$$
$$y' = Ex + Fy + Gz + H$$
$$z' = Ix + Jy + Kz + L$$



$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

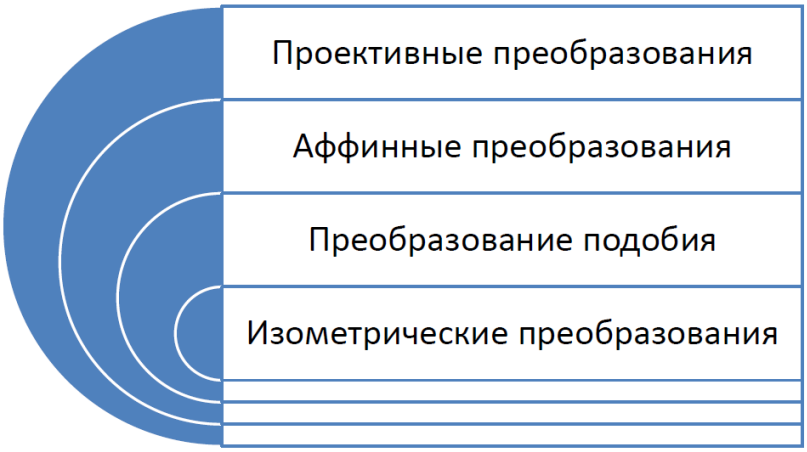
4-я координата W важна!

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

• Позволяет использовать матричную запись для всех линейных преобразований (если использовать матрицы 3x3, невозможно представить перенос)
• Позволяет описать так называемой перспективное деление (нужно для проекции)

Типичные линейные преобразований



Проективные преобразования – прямые в прямые
•w != 1 (после преобразования)
•Прямые переходят в прямые

Аффинные преобразования – сохраняют параллельность
Линейные +
•w = 1
•Сохраняется параллельность линией
•Пример: сдвиг

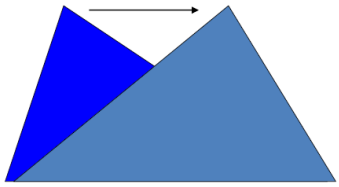
Преобразования подобия – сохраняют углы
Аффинные +
•Сохраняются углы
•Пример: равномерное масштабирование

Изометрические преобразования – сохраняют размеры
Подобия +
•Сохраняются расстояния
•Пример: поворот, перенос

Сдвиг

$$x' = x + ay$$
$$y' = y + bx$$

Аффинное преобразование



$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & a & 0 & 0 \\ b & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Масштабирование

$$x' = ax$$
$$y' = by$$
$$z' = cz$$

Равномерное = преобразование подобия

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Параллельный перенос

$$x' = x + \Delta x$$
$$y' = y + \Delta y$$
$$z' = z + \Delta z$$

Изометрия

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Поворот (2D)

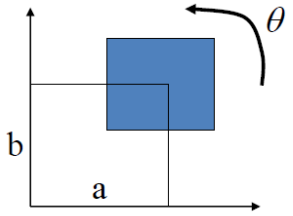
$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

Изометрия

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Введем набор типичных преобразований и их обозначения
- Сдвиг Sh(a,b,c)
- Масштабирование S(a,b,c)
- Перенос T(a,b,c)
- Поворот R(theta) или R(axis, theta)

Суперпозиция преобразований позволяет составлять сложные преобразования из простых

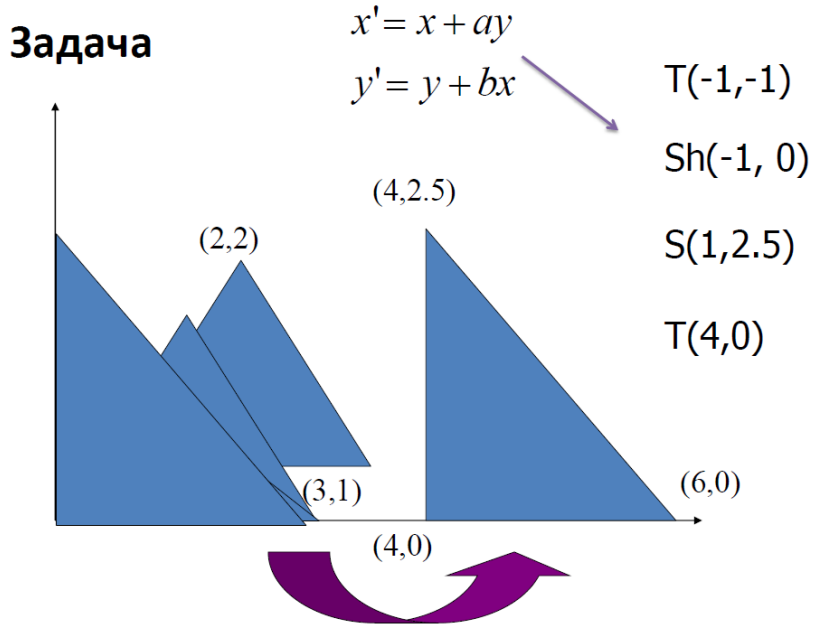


Повернуть вокруг центра

Записывать так

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = (T(a,b) * R(\theta) * T(-a,-b)) \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix})$$

Произносить так



Решение задачи
1.T(-1,-1)
2.Sh(-1, 0)
3.S(1,2.5)
4.T(4,0)

M= T(4,0) * S(1,2.5) * Sh(-1, 0) * T(-1,-1)
P'= M * P

•Относительное преобразование каждого блока зависит только от его геометрии
•Абсолютное преобразование формируется домножением на «родительское преобразование»
•Чтобы двигать три блока вместе, надо менять только параметры первого (родительского) блока
•Чтобы повернуть 2й или 3й блок, нужно менять только параметры локального преобразования

20. Графический конвейер. Иерархия преобразований. Иерархия преобразований в OpenGL.

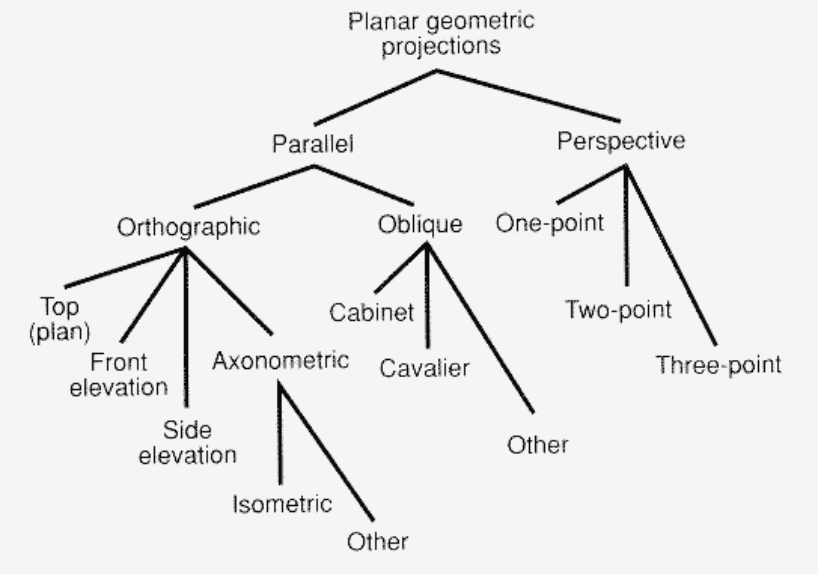
Смотри слайды 51 и далее лекции 8.
Сложные модели создаются с помощью иерархии преобразований
Сложные геометрические сцены создаются путем иерархии моделей со своими преобразованиями

Модель наблюдателя. Проективные преобразования
•Все современные дисплеи визуализируют изображение => необходимо преобразовать 3D данные в 2D !
•Важнейший класс преобразований

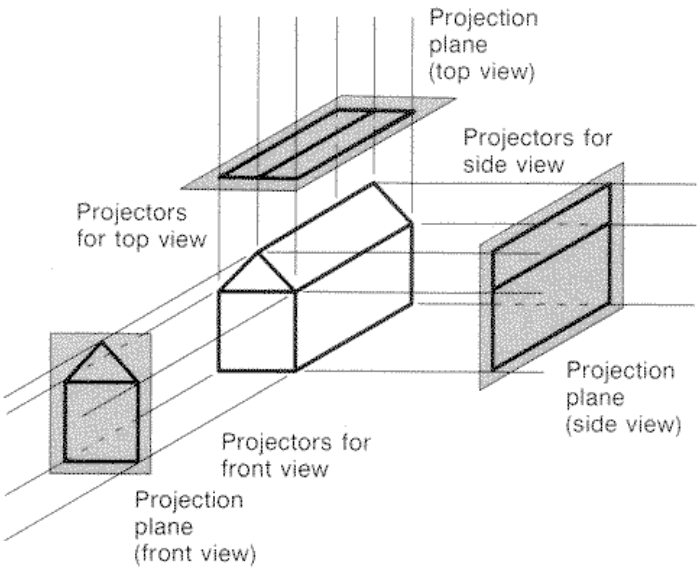
•Для выполнения таких преобразований применяются проективные преобразования
•Описываются матрицей 4x4 (линейным преобразованием)

21. Графический конвейер. Виды проекций, проективные преобразования.

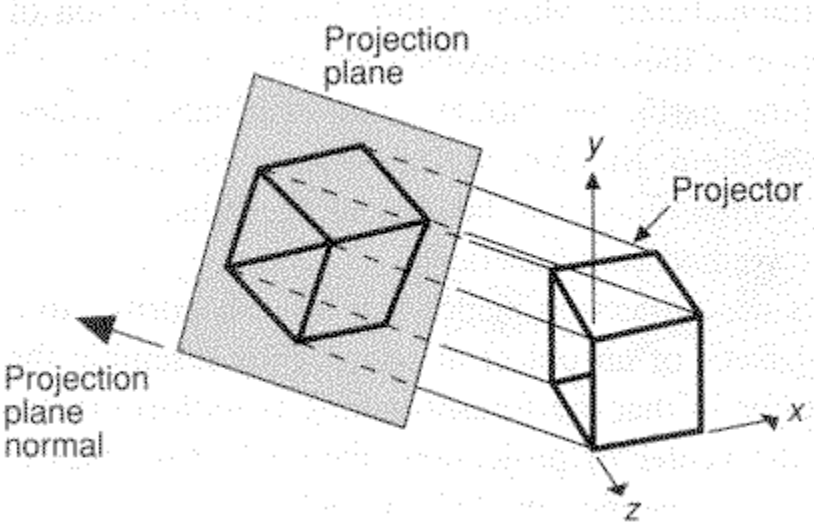
Типы проекций
•Много разновидностей
–Применяются в дизайне и т.п.
•Основные виды
–Параллельные
Ортографические
Косоугольные
–Перспективные
1,2,3-х точечные

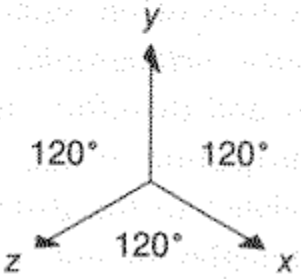


Ортографическая проекция – вдоль осей

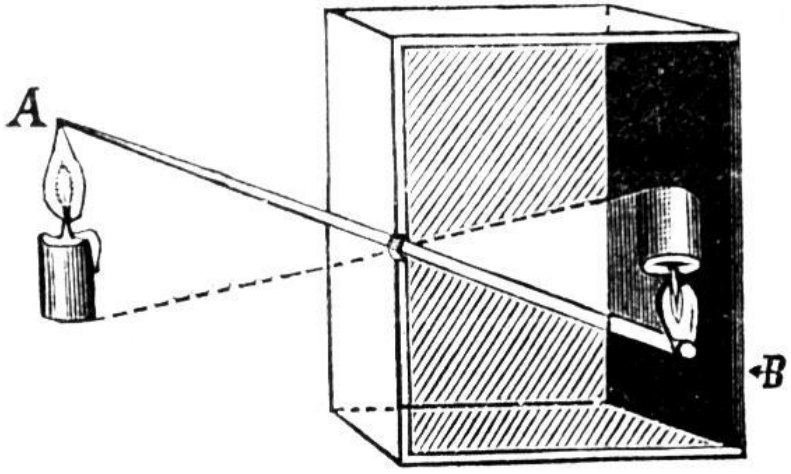


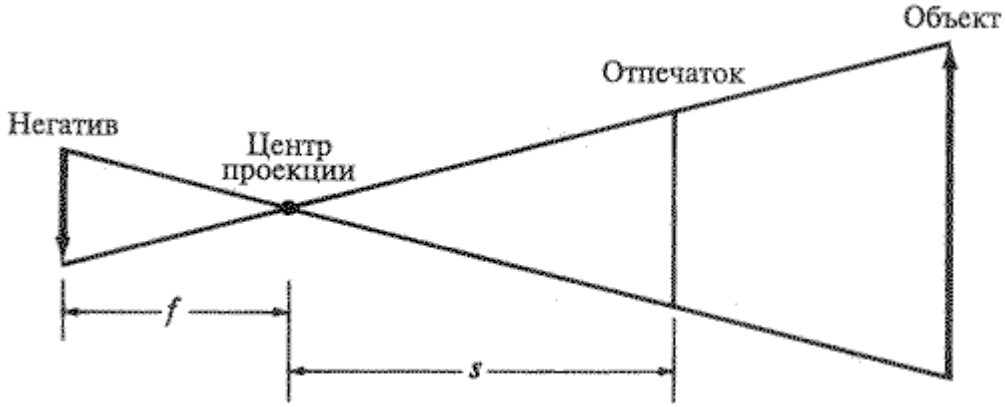
Изометрическая проекция – размеры сохраняются



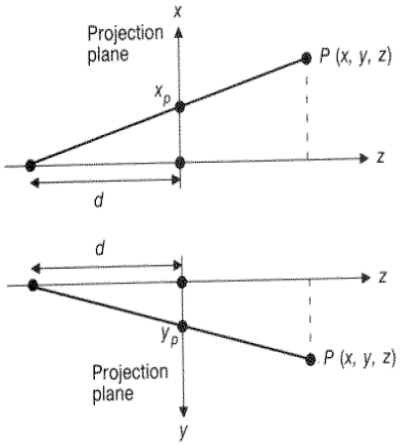


Перспективная проекция пришла из фотографии
Модель объектива (камера-обскура) с бесконечно малым размером диафрагмы





Математическая запись перспективной проекции на плоскость Oxy вдоль оси z



$$\frac{x_p}{d} = \frac{x}{z+d}, \frac{y_p}{d} = \frac{y}{z+d},$$

$$x_p = \frac{d \cdot x}{z+d} = \frac{x}{(z/d)+1}$$

$$y_p = \frac{d \cdot y}{z+d} = \frac{y}{(z/d)+1}$$

Перспективная проекция : возможна запись в матричном виде

$$x_p = \frac{x}{(z/d)+1}$$

$$y_p = \frac{y}{(z/d)+1}$$

$$M_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$$

Запись в матричном виде: Перспективное деление

Применяем матрицу M_{per}

$$M_{per}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ z/d + 1 \end{pmatrix}$$

Необходима нормализация (перспективное деление)

Четвертая компонента не равна 1 !

- Результат уже не в декартовых координатах

Однородные координаты!

$$x = x / w$$
$$y = y / w$$
$$z = z / w$$

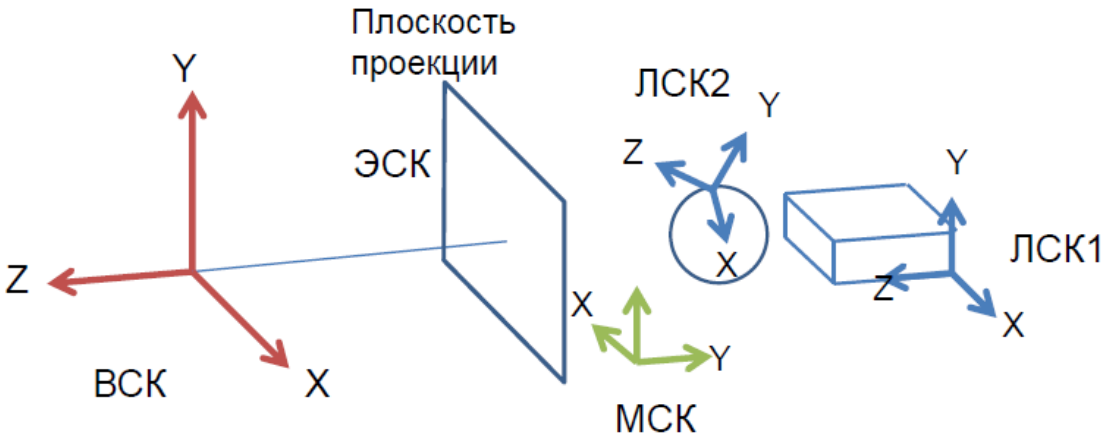
В конвейере OpenGL применяются исключительно линейные и проективные преобразования

•В графическом конвейере OpenGL используются линейные и проективные геометрические преобразования

•Преобразования описываются матрицами **4x4**

•Операции производятся над векторами в однородных координатах

Обработка вершин: три последовательных преобразования (ЛСК->МСК->ВСК->ЭКС)



Модельное преобразование – из локальных в мировые координаты
Переводит модель, заданную в локальных (собственных) координатах, в глобальное (мировое пространство)

Модель «собирается» из частей, с помощью модельных преобразований (обычно композиция переносов, поворотов, масштабирования)
На выходе – модель в единых мировых координатах

Виртуальная камера – нужно задать для получения изображения
•Определяет положение наблюдателя в пространстве
•Параметры
–Положение
–Направление взгляда
–Направление «вверх»
–Параметры проекции
•Положение, направление взгляда и направление «вверх» задаются матрицей видового преобразования

Видовое преобразование – нужно для перемещения мира, который видит камера
•Проективные преобразования описывают «стандартные» проекции, т.е. проецируют фиксированную часть пространства
•Что если мы хотим переместить наблюдателя?

Варианты:
–Изменить матрицу проекции чтобы включить в нее информации о камере
–Применить дополнительное преобразование, «подгоняющее» объекты под стандартную камеру
•Стандартная камера в OpenGL:
–Наблюдатель в (0, 0, 0)
–Смотрит по направлению (0, 0, -1)
–Верх (0, 1, 0)

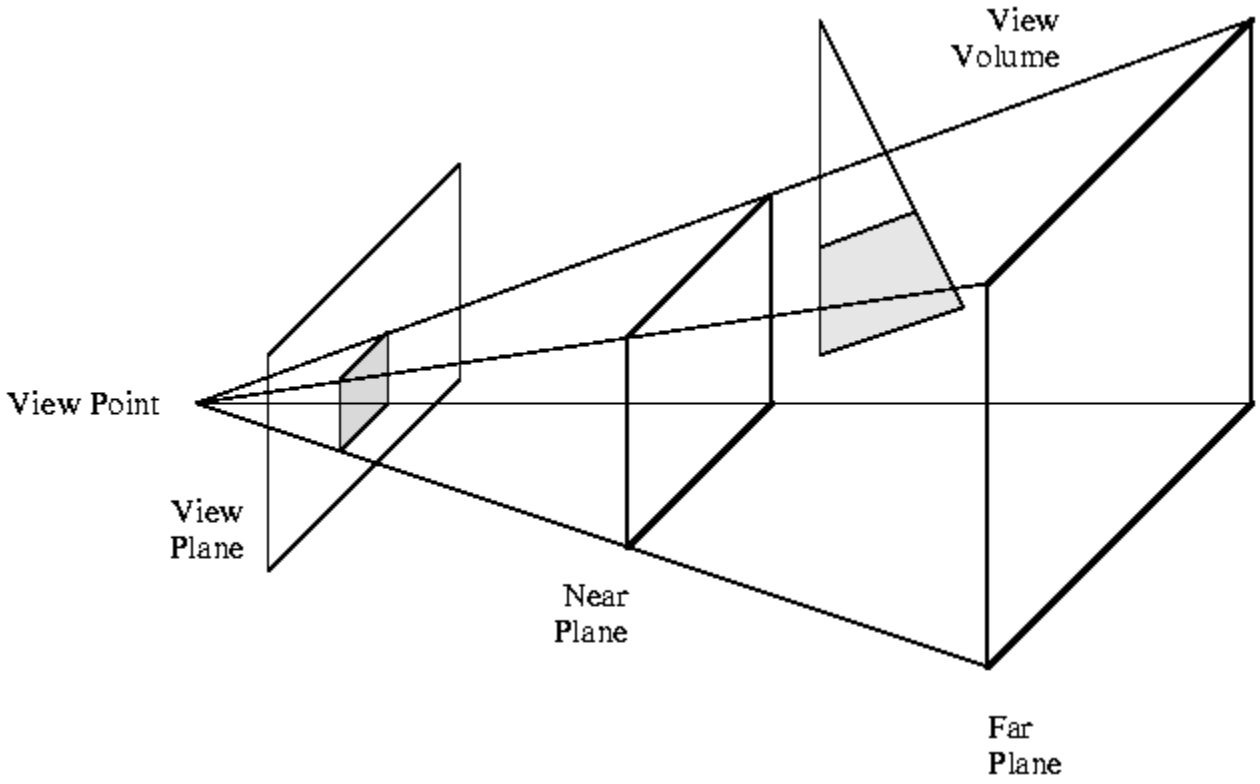
Видовое преобразование – из мировых в видовые координаты
•«Подгоняет» мир под стандартную камеру, преобразует мировую систему координат в видовые координаты (которые подходят для «стандартной» камеры)
•На выходе – модель, готовая к проекции на экран

Проективное преобразование – перспективные и масштабные искажения
•Выполняет 3D преобразование, подготавливая модель к переходу на 2D

Проективное преобразование vs. проекция (1/2)
•Матрица проекции вырожденная
–Фактически, информация от координате z теряется
•Часто необходимо выполнять дополнительные действия уже ПОСЛЕ проецирования
–Например, удаление невидимых линий/поверхностей
•Поэтому часто (e.g. в OpenGL) используется проективное преобразование вместо проекции
–Проективное преобразование невырожденно и позволяет анализировать глубину!

Проективное преобразование vs. проекция (2/2)
•Проективное преобразование переводит модель в еще одну систему координат – пространство отсечения (clip space)
•В пространстве отсечения видимая область превращается в куб (каноническую пирамиду видимости) [-1,-1,-1] – [1,1,1]

При задании проективного преобразования необходимо указать границы отсечения



Преобразование в экранные координаты – простое отбрасывания z-координаты
1.Отбрасываем координату z
2.Умножаем на высоту/ширину окна
–Получаем экранные координаты

Модельно-видовое преобразование = модельное + видовое
•OpenGL не имеет раздельных матриц для видового и модельного преобразования
•Поэтому нужно задавать сразу произведение:
M=Mview*Mmdl

Матрицы преобразований

- Выбираем матрицу преобразований для изменения:

```
void glMatrixMode(Glenum mode);
    mode={GL_MODELVIEW|GL_PROJECTION}
```

- Две основные операции над матрицами

```
void glLoadIdentity();
```

$$M = E$$

```
void glMultMatrixd(GLdouble c[16]);
```

$$M = M \cdot \begin{bmatrix} c[0] & c[4] & c[8] & c[12] \\ c[1] & c[5] & c[9] & c[13] \\ c[2] & c[6] & c[10] & c[14] \\ c[3] & c[7] & c[11] & c[15] \end{bmatrix}$$

Матрицы преобразований (2)
void glTranslated(GLdouble x, GLdouble y, GLdouble z);
void glScaled(GLdouble x,
GLdouble y,
GLdouble z); void glRotated(GLdouble angle, GLdouble ax, GLdouble ay, GLdouble az);
void gluPerspective(GLdouble fov,
GLdouble aspect,
GLdouble znear,
GLdouble zfar);

Видовое преобразование gluLookAt(eyex, eyey, eyez, aimx, aimy, aimz, upx, upy, upz)
Настройка виртуальной камеры
eye – координаты наблюдателя
aim – координаты “цели”
up – направление вверх

Модельно-видовое преобразование (2)

- glMatrixMode(GL_MODELVIEW);
- gluLookAt(..);
- glTranslate(…);
- glRotate(…);
- glTranslate(…);
- glBegin(…);
- …
- glEnd();

Виртуальная камера

Модельное преобразование

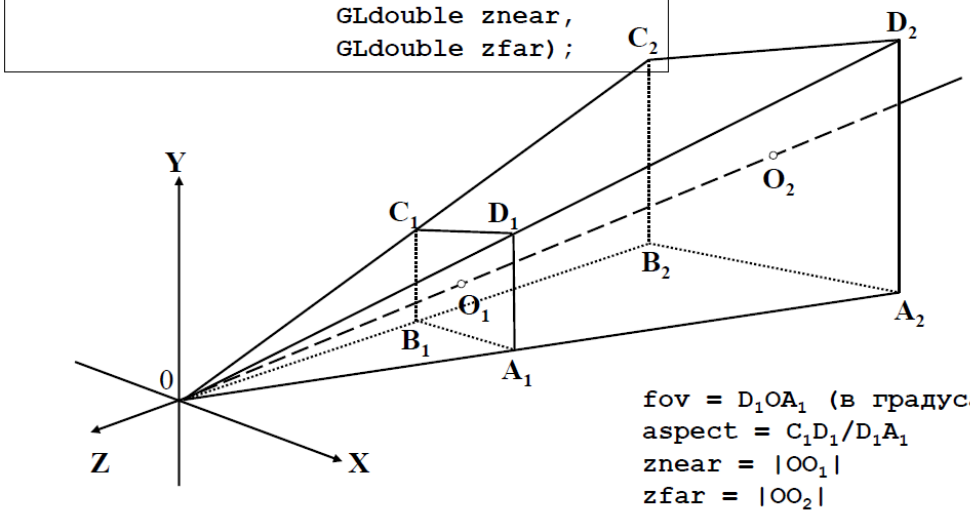
Геометрия

Внимание! При определении геометрии к ней применяется текущий набор матриц преобразования!

Проективное преобразование
•glMatrixModel(GL_PROJECTION);
•gluPerspective(…)

Как работает gluPerspective

```
void gluPerspective(GLdouble fov,
                    GLdouble aspect,
                    GLdouble znear,
                    GLdouble zfar);
```



fov = D_1OA_1 (в градусах)
aspect = C_1D_1/D_1A_1
znear = $|OO_1|$
zfar = $|OO_2|$

Итоги 1/2
•Растеризация – метод синтеза изображений с помощью отображения трехмерной геометрии на экран
•OpenGL
–Кросс-платформенная библиотека функций для создания интерактивных 2D и 3D приложений
–Определение геометрии
•glVertex, glBegin, glEnd

•Геометрические преобразования
–Типы преобразований
•Нелинейные преобразования
•Линейные преобразования (проективные)
•Аффинные преобразования
•Преобразования подобия
•Изометрические преобразования
–Однородные координаты
•Много применений: унификация операций с матрицами, перспективное деление и т.п.
–Комбинация, иерархия преобразований
•Сборка модели из локальных компонент
•Графический конвейер: от локальной модели до точки на экране
–Локальные, мировые, экранные координаты

22. Алгоритм растеризации с помощью строчной развертки. Закраска Гуро и Фонга. Особенности освещения в OpenGL.

Растеризация: процесс вычисления пикселей растра, принадлежащих примитиву
Про алгоритм даже не знаю что писать:(там одни картинки. Лекция 10 третий пункт

Вычисление цвета материала
Варианты:
– Вычисление цвета по модели освещения для каждого
пикселя (per-pixel shading)
– Вычисление на вершинах с последующей интерполяцией
(vertex shading)
– Вычисление цвета для примитива (flat shading)

По-вершинное тонирование требует интерполяционной закраски
По-пиксельное тоже требует интерполяции, если нормали заданы на вершинах

Закраска Фонга (пописельное)
– Расчет нормалей в вершинах
– Линейная интерполяция нормалей по треугольнику
– Расчет цвета по формуле Фонга (или другой)

Закраска Гуро (повершинное)
– Расчет цвета в вершинах
– Линейная интерполяция цвета

Проблемы закраски Гуро – можно пропустить блики
Решение проблемы закраски Гуро: подразбиение полигонов (увеличивает сложность геометрии)

Картинки, примеры и несколько формул можно найти в лекции 10 третий пункт\

Особенности освещения в OpenGL:
Составляющие модели освещения:
– Свойства материала поверхности
    – Определение свойств материала для примитива
        glMaterialfv (face, property, value)
        GL_DIFFUSE
        GL_SPECULAR
        GL_AMBIENT
        GL_EMISSION
        GL_SHININESS
– Свойства источника света
    – цвет источника
    – положение
    – затухание
OpenGL поддерживает два типа источников света
– Локальные (точечные) источники
– Бесконечно удаленные (параллельные) источники
Тип определяется координатой w
– w = 0 параллельный источник
– w != 0 точечный источник (x / w, y / w, z /w)
Точечные источники могут быть
– Всенаправленные (omni)
– Прожекторного типа (spotlight)

– Свойства модели освещения
??? единственное упоминание о чем-то похожем:
Модель Фонга
– Вычисляется на вершинах
– Закраска Гуро или плоская

23. Локальные и глобальные модели освещения. Понятие о ДФО, расчет излучения точки поверхности. Модели освещения Фонга и Ламберта.

Локальные модели учитывает только первичные источники света, **глобальные** – все источники

Двулучевая Функция Отражения (ДФО):
Чему равно излучение поверхности Lo(p, ωo) в направлении ωo при условии излучения по направлению ωi, равной Li(p, ωi) ?

Предполагается, что исходящее излучение зависит только от входящего излучения для данной точки!
В заданном направление излучается энергия, пропорциональная освещенности.
Освещенность пропорциональна площади распределения потока света от источника.

Свойство ДФО:
1) обратимость
2) сохранение энергии

Рассчет излучения можно найти лекции №10 в конце первого пункта.

Модели освещения:
Можно упростить расчет модели, ограничив передаваемые свойства материалов
• Диффузное отражение
• Идеально зеркальное
• Зеркальное отражение

Модель Ламберта
• Ламбертова (идеально диффузная) поверхность выглядит одинаково яркой со всех направлений
• В природе не существует, но есть близкие приближения
• Пример: бумага

Модель Ламберта учитывает только идеальное рассеивание света

Модель Фонга добавляет в модель Ламберта зеркальное отражение

Модель Фонга имеет неприятные особенности, но все равно очень широко применяется
• Не является обратимой
• Не сохраняет энергию

Примеры, картинки и формулы по моделям освещения в лекции №10 во втором пункте
24. Текстуры. Отображение и фильтрация текстур. Текстурирование в OpenGL.
Три основных этапа вычисления цвета:
1) Растеризация: процесс вычисления пикселей растра, принадлежащих примитиву
2) Вычисление цвета пикселя: материал, текстура, фон
3) Вычисление цвета пикселя: материал посчитали, теперь нужен цвет текстуры

Общее отображение текстуры
– Узор, определенный в 2D области, «наклеивается» на объект как кусок обоев и, фактически становится частью объектной базы данных.
– Когда объект перемещается, текстурный узор перемещается вместе с ним

Тут какой-то непонятный кусок из трёх слайдов с картинками цилиндров. Лекция 11

Отображение текстуры для полигональных поверхностей: соответствие на вершинах
Задать соответствие между координатами изображения текстуры и точками объекта
Для полигонального представления используется соответствие на вершинах
– Линейная интерполяция внутри полигона (Гуро)
– Перспективная коррекция

Дальше написана формула, за уточнениями лучше обратиться к лекции 11, но вообще вот она:

Цвет из текстуры может модулировать какой-либо коэффициент модели освещения
$Ir = ka*Ia + Ii (kd*(N.L) + ks*(R.V)^\alpha)$
• отражение от поверхности - surface color texture
• вектор нормали N - bump mapping

• коэффициенты kd, ks, a - specularity mapping
• падающий свет Ii - environment mapping
• геометрия - displacement mapping
• прозрачность - transparency mapping

Фильтрация текстур:

Метод ближайшего соседа: берем ближайший тексель
Выбирается цвет ближайшего соответствующего пикселя текстуры
– Быстрый метод
– Низкое качество

Билинейная фильтрация: усредняем ближайшие четыре пикселя
Четыре пикселя текстуры, ближайшие к текущей точке экрана. Результирующий цвет – смешение цветов этих пикселей
– Быстро, достаточно качественно
– За исключение случаев, когда смотрим на плоскость под углом

Mipmapping (multum in parvo): вычисляем разные размеры фильтра заранее
Выбирается подходящий уровень мипмапа
– чем больше размер "образа" пикселя в текстуре, тем меньший мипмап берется
Далее значения в мипмапемогут усредняться билинейно или методом ближайшего соседа. Дополнительно происходит фильтрация между соседними уровнями мипмапа (трилинейная фильтрация)

Анизотропная фильтрация: вытянутый фильтр
Вместо квадратного фильтра, используется вытянутый.
Позволяет более качественно выбрать нужный цвет для экранного пикселя

По всем фильтрам есть примеры в лекции 11 первый пункт

Текстурирование в OpenGL
– Создаем текстуру - прямоугольный массив с цветами пикселов. Высота и ширина должны быть степенями двойки.
– Получаем номер текстурного объекта

```
GLuint texture;
glGenTextures(1,&texture);
```

– Активизируем текстурный объект

```
glBindTexture(texture);
```

– Загружаем текстуру

```
glPixelStorei(GL_UNPACK_ALIGNMENT,1);
glTexImage2D(GL_TEXTURE_2D,
        0,              // Mip-level
        GL_RGB,         // Формат текстуры
        tex_width,tex_height,
        0,              // Ширина границы
        GL_RGB,         // Формат исходных данных
        GL_UNSIGNED_BYTE,   // Тип данных
        tex_bits);      // Исходные данные
```

– Устанавливаем режимы текстурирования
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```
– Разрешаем текстурирования
```
glEnable(GL_TEXTURE_2D);
```
– Задаем текстурные координаты
```
glTexCoord2d(u,v);
```
– Возвращаем номер текстурного объекта в список свободных
```
glDeleteTextures(1,&texture);
```

25. Методы удаления невидимых поверхностей

При проецировании на экран часто оказывается, что отдельные части объектов (или объекты полностью) перекрываются другими объектами и не видны наблюдателю. Метод растеризации (преобразования координат) сам по себе не позволяет ответить на вопрос, какие объекты видны

Методы удаления невидимых поверхностей
• Удаление нелицевых граней
• Трассировка лучей
• Алгоритм художника
• Буфер глубины

Про лицевые грани в лекции 11 пункт второй. Там картинка с пояснением

Свойства (не-)лицевых граней
I. Для сплошных (solid) тел, ни одна из нелицевыхграней никогда не будет видна даже частично
– При определении видимости нелицевые грани можно всегда отбрасывать
– сокращает число граней для визуализации примерно вдвое
II. Если вся сцена состоит только из одного выпуклого объекта, все лицевые грани и только они будут видны, причем полностью

Удаление нелицевых граней: простой, но только для выпуклых объектов
(+) очень простой, может сократить число граней для растеризации в два раза
(-) работает только для сплошных тел
(-) удаляет все невидимые грани только для сцен с одним выпуклым сплошных объектом, иначе должен использоваться в сочетании с другими методами

Трассировка лучей: неявно решает задачу удаления невидимых граней
При использовании метода трассировки лучей через каждый пиксель картинной плоскости выпускается луч (из положения наблюдателя) в сцену. Далее находится ближайшее пересечение этого луча с объектами сцены. Оно определяет, что именно будет видно в данном пикселе
Трассировка лучей: работает хорошо, но не подходит для растеризации
Метод трассировки лучей
(+) работает для любой геометрии
(-) только для визуализации методом трассировки лучей, т.е. не подходит для растеризации

Алгоритм художника: рисуем грани от дальних к ближним

– Явно сортирует все грани сцены в порядке их приближения к наблюдателю (back-to-front)
– Выводит грани в этом порядке
=> получается корректное изображение

Алгоритм художника по шагам
1. Упорядочить все многоугольники в соответствии с наименьшей (дальней) координатой z
2. Разрешить неоднозначности, которые вызывают наложение z-габаритов
3. Нарисовать многоугольники в порядке возрастания наименьшей координаты z

Алгоритм художника:
(-) линейная сложность сортировки
(-) квадратичная сложность для многоугольников с пересекающимися z-объемами
(-) не может справиться со всеми типами взаимного расположения

Метод двоичного разбиения пространства: та же идея, что в алгоритме художника
Пусть известно, что плоскость пи разбивает все грани (объекты) сцены на два непересекающихся множества в зависимости от того, в каком полупространстве по отношению к данной плоскости они лежат. Тогда ни одна из граней, лежащих в том же полупространстве, что и наблюдатель, не может быть закрыта ни одной из граней из другого полупространства
=> удалось осуществить частичное упорядочение граней исходя из возможности загораживания друг друга.

Метод двоичного разбиения пространства
(+) логарифмическая сложность сортировки объектов по глубине (быстро работает), позволяет быстро выводить полигоны back-to-front
(-) проблемы, аналогичные алгоритму художника
(-) работает для статических сцен, дерево перестраивается долго

Буфер глубины
Каждому пикселю картинной плоскости, кромезначения цвета, хранящемуся в буфере кадра, сопоставляется еще значение глубины
– расстояние вдоль направления проецирования от картиннойплоскости до соответствующей точки пространства

```
foreach(p in pixels)
 if(p.z < zBuffer[p.x, p.y])
 {
   draw( p );
   zBuffer[p.x, p.y] = p.z;
 }
```

Примечание: для работы буфера глубины нужно проективное преобразование
Если бы использовались вырожденные матрицы проекции вместо проективного преобразования, вычислить значение глубины было бы невозможно!

Метод буфера глубины
(+) очень простой, легко реализуется аппаратно

(+) работает для произвольной геометрии
- необязательно с граничным представлением

(-) требует дополнительную память
(-) требует растеризации всех примитивов
(-) не сортирует back-to-front, невозможно реализовать прозрачность

В лекции 11 пункт два так же есть про удаление невидимых поверхностей в OpenGL, но в вопрос это вроде не входит. И как всегда по всем методам в лекции ка

26. Синтез изображений с помощью обратной трассировки лучей. Свойства алгоритма. Способы поиска пересечений.

Растеризация и трассировка лучей – два основных подхода к синтезу изображений, растеризация - прямая проекция геометрии, трассировка - обратная проекция пикселей изображения.

Растеризация - быстро, но каждый треугольник обрабатывается отдельно (Последовательная обработка всех треугольников по одному, большинство реалистичных эффектов требуют доступа ко всей)

Трассировка лучей: моделируем распространение света вдоль лучей

Прямая трассировка: свет идет из источников света
–Плюсы: точная физическая модель

–Минусы: нам нужен только свет, попадающий в камеру => очень неэффективно

Из физики известно, что распространение света обратимо => можно проследить (протрассировать) свет из каждого пикселя.

Обратная трассировка: «свет» идет из каждого пикселя изображения

Трассировка лучей:
1)генерируем луч через каждый пиксель изображения
2)необходимо найти пересечения луча с поверхностью
Пространственные структуры позволяют быстро ограничить множество примитивов(рассматривают трассировку луча по пространственной сетке)
3)Тонирование - получение цвета пикселя
4)Рисуем дополнительные лучи - для источника света и отаражения(выполняем заново шаги 1-2-3)
5) Буфер кадра - заносим результат

Псевдокод (простой трассировки):

```
for each pixel of the screen {
Final color = 0;
Ray = { starting point, direction };
Repeat {
for each object in the scene {
```

```
determine closest ray object/intersection;
}
if intersection exists {
for each light in the scene {
if the light is not in shadow of another object {
add this light contribution to computed color;
}
}
}
Final color = Final color + computed color * previous reflection factor;
reflection factor = reflection factor * surface reflection property;
increment depth;
} until reflection factor is 0 or maximum depth is reached;
}
```

Проблема трассировки лучей – для вычисления глобального освещения нужно трассировать большое количество лучей. Вычислительно сложная задача.

27. Расчет глобального освещения с помощью метода излучательности. Форм-факторы. Свойства алгоритма.

http://www.ray-tracing.ru/articles219.html - со всеми формулами про излучаемость

Основная идея излучательности – сохранение светимости поверхностей по мере продвижения света от источников.

Излучательность = Энергетическая светимость

• Полная энергия, покидающая поверхность единичной площади

• Обозначение: B (=M)

• Единицы измерения: Вт/м2

M= dΦ/ds

Алгоритмы излучательности решают уравнение глобального освещения при наборе ограничивающих предположений:

–Все поверхности ламбертовы (идеально диффузные)

–Поверхности могут быть поделены на участки (патчи) константной излучательности

–Излучательность рассчитывается только на поверхностях

- Требуются дополнительные действия по построению изображения

–Расчет производится для замкнутой системы

- Энергия не пропадает

Эти предположения позволяют сделать уравнение глобального освещения линейным!

Излучательность диффузных поверхностей: простая форма, простая ДФО

Формулы для излучательности диффузных поверхностей и диффузной ДФИ в терминах коэффициента отражения поверхности смотри в 13 лекции, слайд 52.

Упрощаем интеграл освещенности(слайд 53) -> Преобразуем интеграл по телесному углу в интеграл по всем точкам поверхностей сцены (54)-> Заменяем интеграл по точкам геометрии на сумму по поверхностям (56) -> Вводим понятие форм-фактора(57) -> записываем исходное понятие в виде СЛАУ (58)

Алгоритм метода Излучательности:
1)Разбиваем геометрию на площадки
2)Вычисляем форм-факторы
3)Решаем СЛАУ
4)Реконструкция и показ решения

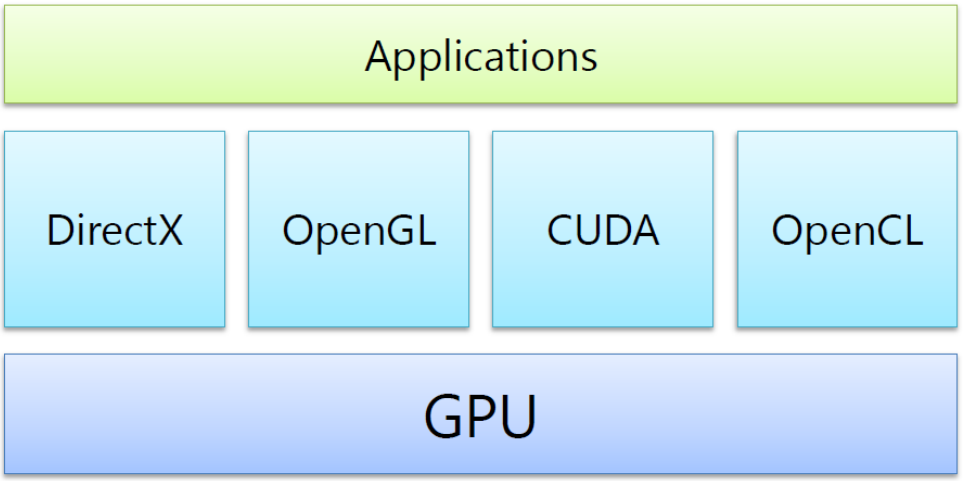
Итог(Основная идея):
Излучательность: рассчитываем перенос энергии между поверхностями сцены, начиная с источников света

("Вики")
Основная трудность в алгоритме – расчет форм-факторов.
Стоит отметить еще один недостаток алгоритма – сложность моделирования мягких теней, т.е. теней от протяженных источников. При моделировании мягких теней возникают т.н. блочные тени. Блочные тени образуются из-за того, что минимальной единицей расчета освещения является патч. Единственный способ побороться с подобными артефактами – произвести подразбиение патчей в области мягкой тени.

28. Понятие о программируемой графической аппаратуре. Устройство современных графических процессоров с точки зрения графических API. Языки программирования GPU, их особенности.

http://ru.wikipedia.org/wiki/OpenGL - о всех версия OpenGL

API программирование GPU



Как выбрать API? - в зависимости от ОС и от железа

OpenGL3
1)Загрузка текста шейдеров из файла

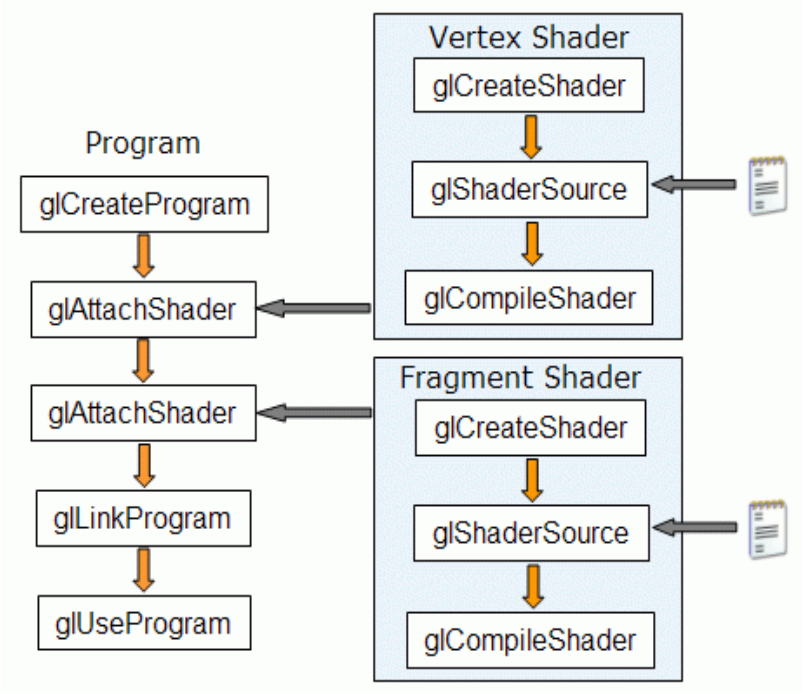
2)Компиляция (vs_text, ps_text) -> (vs, fs)

3)Линковка (vs, fs) => program
•g_prog = ShaderProgram(…);
4)Создание и загрузка данных
•GLuint g_vbo1 = … // загружаем позиции вершин

•GLuint g_vbo2 = … // на GPU

•(g_vbo1, g_vbo2) => g_vao
5)Привязка ресурсов, задание констант
•glUniformMatrix4fv(…);

•bindTexure(…)
6)Указать VS откуда читать данные
•glBindVertexArray(g_vao);
7)Draw call
•glDrawArrays(GL_TRIANGLES, 0, 3); // 3 – кол-во вызовов вершинного шейдера



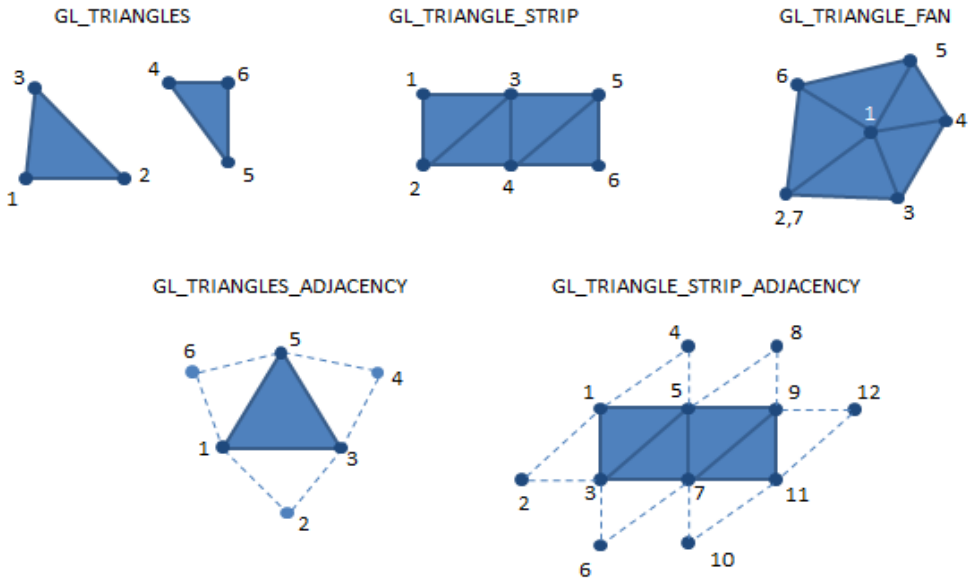
Создание объекта программы

Пример вершинного шейдера(Как переводить позиции вершин в clip space)

```
#version 330
uniform mat4 projectionMatrix;
uniform mat4 modelViewMatrix;
in vec2 vertex;
out vec4 vpos;
void main(void)
{
vec4 viewPos = modelViewMatrix*vec4(vertex,0.0,1.0);
gl_Position = projectionMatrix*viewPos;
vpos = viewPos;
}
```

Графический конвейер

glDrawArrays(<метод>,0,6)



Слайды с различиями между OpenGL 1-2-3-4

OpenGL
•API для доступа к функциональности GPU

•Мультиплатформенный

•Мультиязыковой

•Процедурный

•Динамически расширяемый

•Указатели на функции
    •wglGetProcAdress(…)
•Доступ к объектам осуществляется при помощи целочисленный имен

•Понятие текущего объекта

```
glGenBuffers (1, &m_vbo);
glBindBuffer(GL_ARRAY_BUFFER, m_vbo);
glBufferData(…);
```

Текстурирование в OpenGL
Текстура – хранилище данных, хранилище значений некоторой функции
Выборка из текстуры – процесс получения значения это функции. Включает в себя интерполяцию, фильтрацию, декомпрессия лету (DXT сжатие).

Фильтрация текстур - рассматриваются 4 типа, есть ссылка полезная.

Языки программирования GPU, их особенности.

CUDA (англ. Compute Unified Device Architecture) — программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы NVIDIA. CUDA SDK позволяет программистам реализовывать на специальном упрощённом диалекте языка программирования Си алгоритмы, выполнимые на графических процессорах NVIDIA, и включать специальные функции в текст программы на Си.

Архитектура CUDA даёт разработчику возможность по своему усмотрению организовывать доступ к набору инструкций графического ускорителя и управлять его памятью.
Подробнее - http://ru.wikipedia.org/wiki/CUDA

OpenGL (Open Graphics Library — открытая графическая библиотека, графическое API) — спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двухмерную и трёхмерную компьютерную графику.
Включает более 250 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях. На платформе Windows конкурирует с Direct3D.

Подробнее - http://ru.wikipedia.org/wiki/OpenGL

DirectX (от англ. direct — прямой, непосредственный) — это набор API, разработанных для решения задач, связанных с программированием под Microsoft Windows. Наиболее широко используется при написании компьютерных игр. Пакет средств разработки DirectX под Microsoft Windows бесплатно доступен на сайте Microsoft. Зачастую обновленные версии DirectX поставляются вместе с игровыми приложениями.

Подробнее - http://ru.wikipedia.org/wiki/DirectX

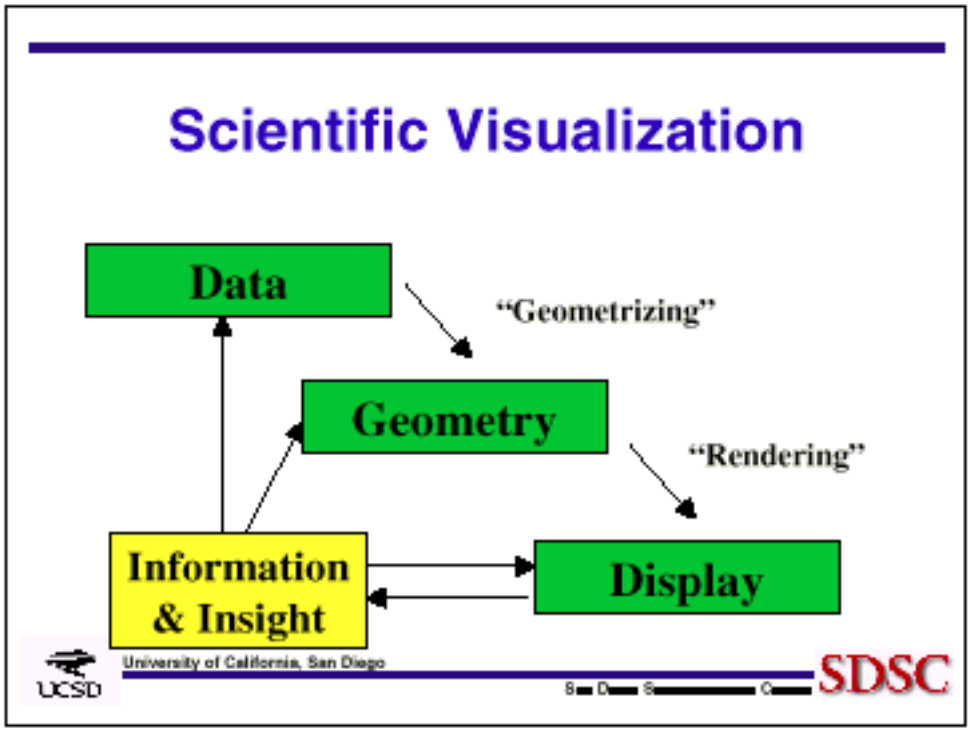
OpenCL (от англ. Open Computing Language — открытый язык вычислений) — фреймворк для написания компьютерных программ, связанных с параллельными вычислениями на различных графических (англ. GPU) и центральных процессорах (англ. CPU). В фреймворк OpenCL входят язык программирования, который базируется на стандарте C99, и интерфейс программирования приложений (англ. API). OpenCL обеспечивает параллелизм на уровне инструкций и на уровне данных и является реализацией техники GPGPU. OpenCL является полностью открытым стандартом, его использование не облагается лицензионными отчислениями.
Цель OpenCL состоит в том, чтобы дополнить OpenGL и OpenAL, которые являются открытыми отраслевыми стандартами для трёхмерной компьютерной графики и звука, пользуясь возможностями GPU. OpenCL разрабатывается и поддерживается некоммерческим консорциумом Khronos Group, в который входят много крупных компаний, включая Apple, AMD, Intel, Nvidia, ARM, Sun Microsystems, Sony Computer Entertainment и другие.

Подробнее - http://ru.wikipedia.org/wiki/OpenCL

29. Задачи визуализации. Понятие о научной визуализации и визуализации информации. Алгоритм «Марширующие кубы»

Задачи:
Научная визуализация
Визуализация информаци
Визуальная аналитика

Научная визуализация – визуализация результатов научных экспериментов или вычислений



Визуализация информации – разработка визуальных метафор для невизуальной информации
Примеры:
–Визуальные представления для текстовой информации

–Визуальных анализ баз данных

Визуальная аналитика
Анализ полученных с помощью моделирования результатов

Изолинии и изоповерхности: определения
•Изолиния: линия, в каждой точке которой измеряемая величина сохраняет одинаковое значение

•Изоповерхность: множество точек пространства, в которых измеряемая величина сохраняет одинаковое значение

•Для скалярной функции F(x,y) изолиния F(x,y) = C

•Для скалярной функции F(x,y,z) изоповерхность F(x,y,z) = C

Алгоритм "Марширующие кубы"

Задача:
–Визуализация изоповерхности

На входе:
–Скалярное трехмерное поле F(x,y,z)

Применени:
Визуализация томографических сканов (XRAY)

Построение изоповерхностей математических функций

Пример использования алгоритма в 2D
Задано двумерное скалярное поле F(x,y) , Задана константа 5 для определения изолиний

1. Дискретизуем функцию на заданной сетке

2. Для каждой вершины находим, выше она (+) или ниже (-) поверхности.

3. Конфигурация +/- для каждой ячейки определяет форму изолинии. Возможно 16 вариантов

4. Для каждой ячейки выбираем один из вариантов.

5. Линейной интерполяцией находим положение вершин

6. Сохраняем полученный сегмент (или сегменты) в итоговую кривую.
В слайдах отличные картинки и все расписано - лекция 14, слайд 19 и далее

Есть алгоритм "Бегущих кубов" - это расширение на трехмерный случай