Компьютерная графика
Лекция 8
31.10/01.11 2013

Синтез изображений с помощью растеризации. OpenGL

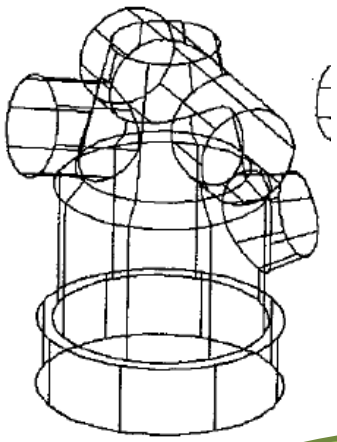
Алексей Викторович Игнатенко
Лаборатория компьютерной графики и мультимедиа
ВМК МГУ

Лекция из трех частей: алгоритм растеризации, OpenGL, геометрические преобразования

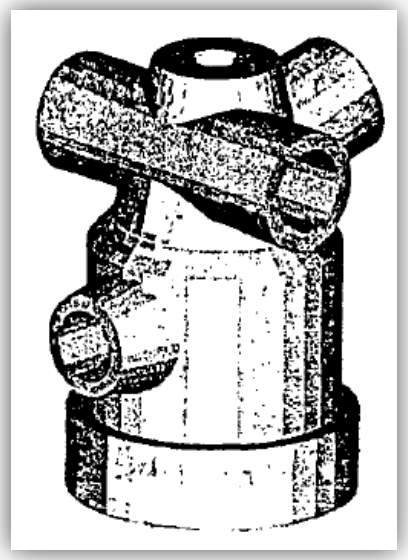
- Алгоритм синтеза изображений с помощью растеризации
- Геометрические преобразования
- OpenGL: Архитектура и основные функции

Графический процесс: типовая последовательность применения алгоритмов

Геометрическая модель
Модель освещения
Модель анимации



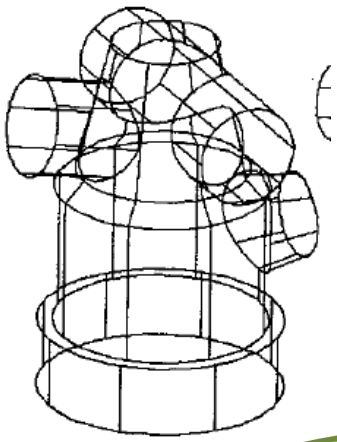
Модель → Алгоритм синтеза → Изображение



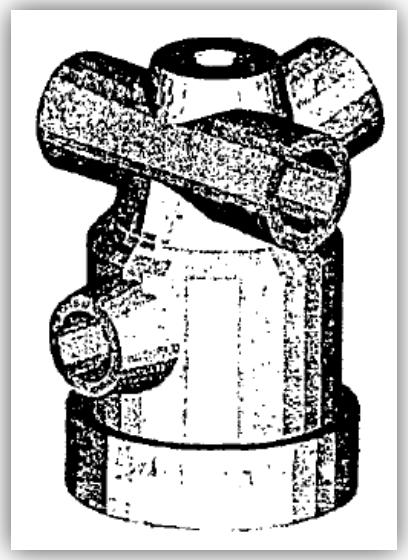
Получение проекции
Заливка

Графический процесс: типовая последовательность применения алгоритмов

Геометрическая модель
Модель освещения
Модель анимации



Модель → Алгоритм синтеза → Изображение



Получение проекции
Заливка

При выборе алгоритма надо выбрать между скоростью и качеством

Скорость

Качество

Выбор условный! Все зависит от особенностей данных и реализации. Появление и развитие GPU сильно изменили баланс

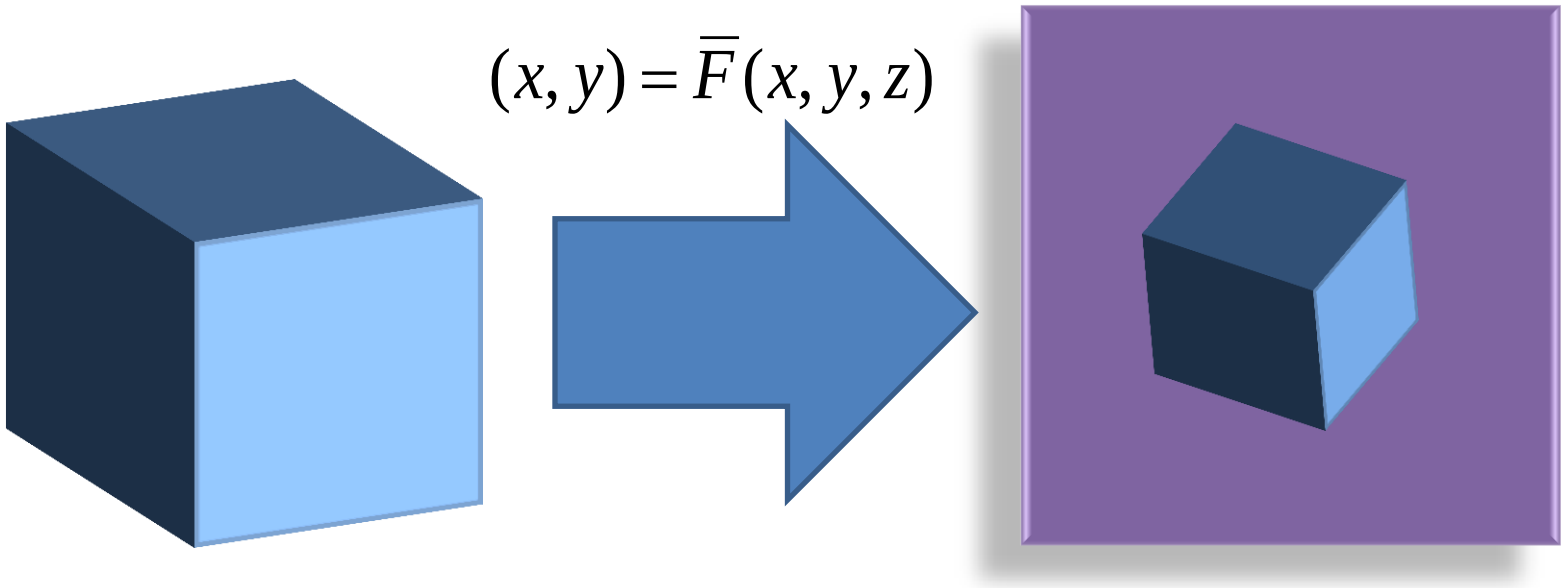
Растеризация

Трассировка лучей/фотонов
Излучательность

Метод растеризации работает с помощью проекции сцены на изображение

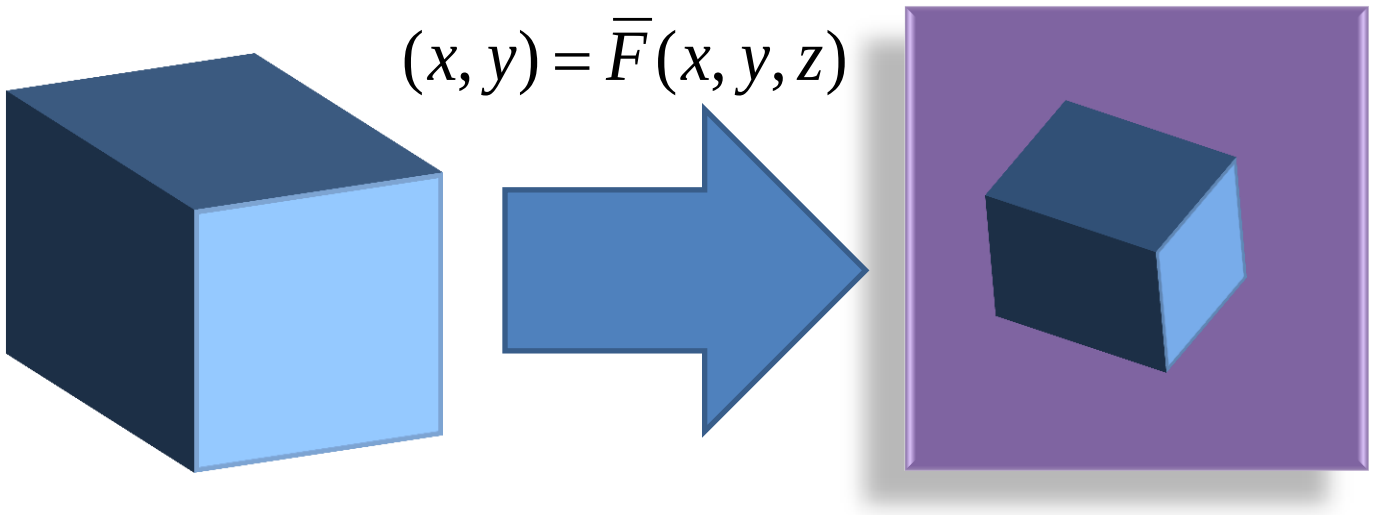
В общем случае растеризация – преобразование векторной информации в растровый формат

В трехмерной компьютерной графике – конкретный алгоритм синтеза изображений



Алгоритм растеризации: функция отображения + расчет цвета пикселей

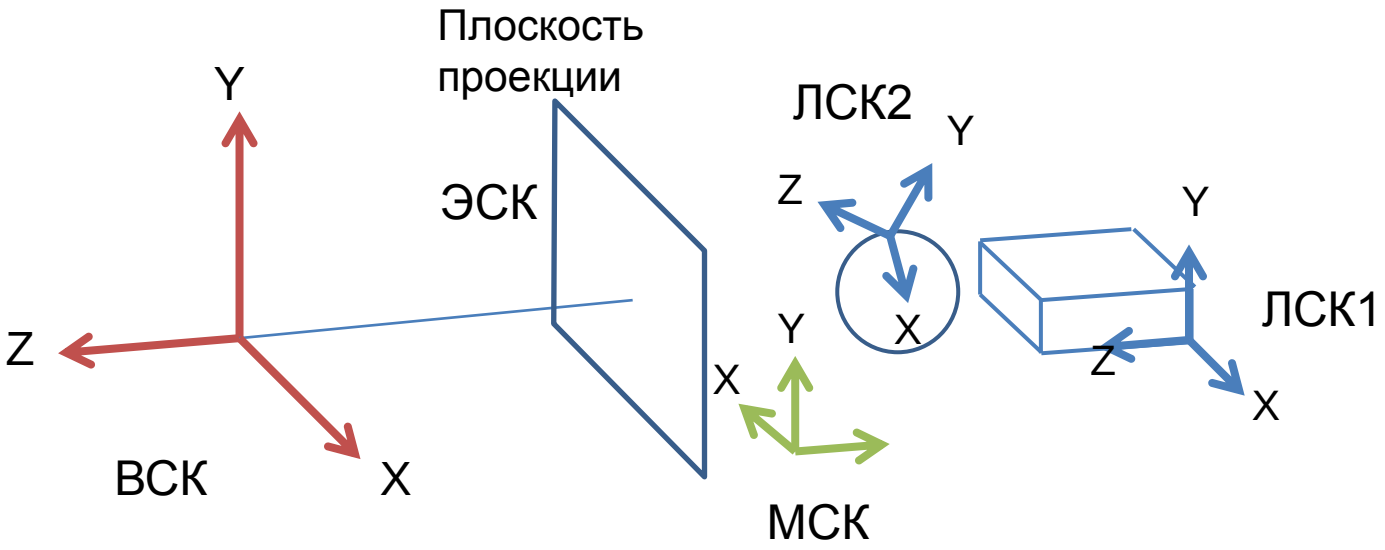
I. Для геометрии модели задается функция отображения в двумерное пространство экрана
II. Функция применяется к модели: определяется множество точек на экране
III. Растеризация – отображение на пиксели
IV. Вычисляется цвет пикселей изображения



Функция проекции на практике задается как комбинация преобразований

Три последовательных преобразования:

- модельное преобразование (ЛСК -> МСК)
- видовое преобразование (МСК -> ВСК)
- проективное преобразование (ВСК -> ЭСК)



Свойства: скорость и поддержка на уровне графических процессоров

- Широко распространен
- Аппаратная поддержка
- OpenGL, DirectX реализуют именно этот подход
- Ориентация на скорость визуализации

- Определяет только проекцию, поэтому может быть совмещен с различными подходами по вычислению цвета

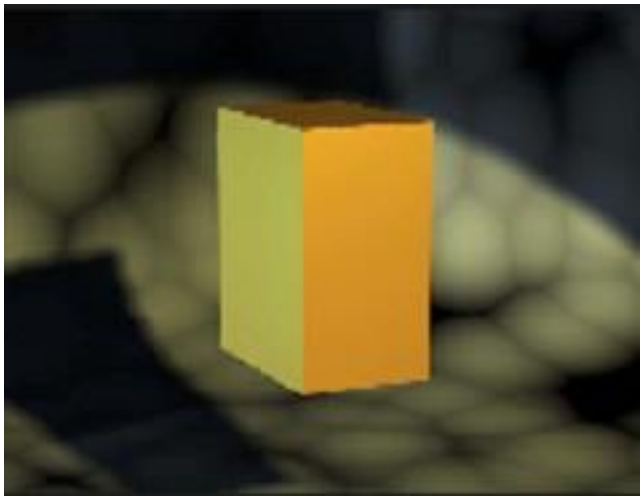
Лекция из трех частей: алгоритм растеризации, OpenGL, геометрические преобразования

- Алгоритм синтеза изображений с помощью растеризации
- Геометрические преобразования
- OpenGL: Архитектура и основные функции

Что такое геометрические преобразования?

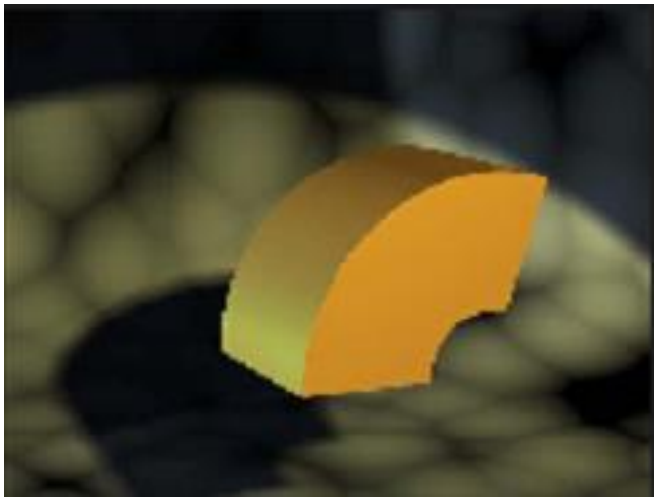
Модель

- 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

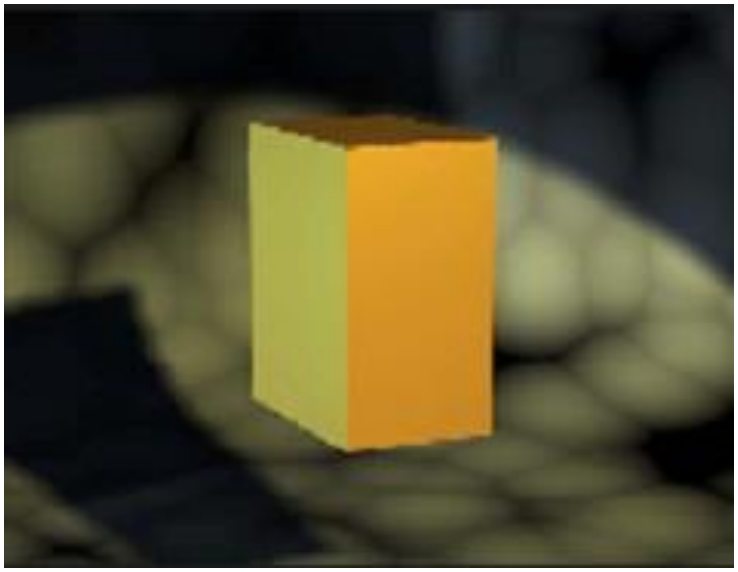


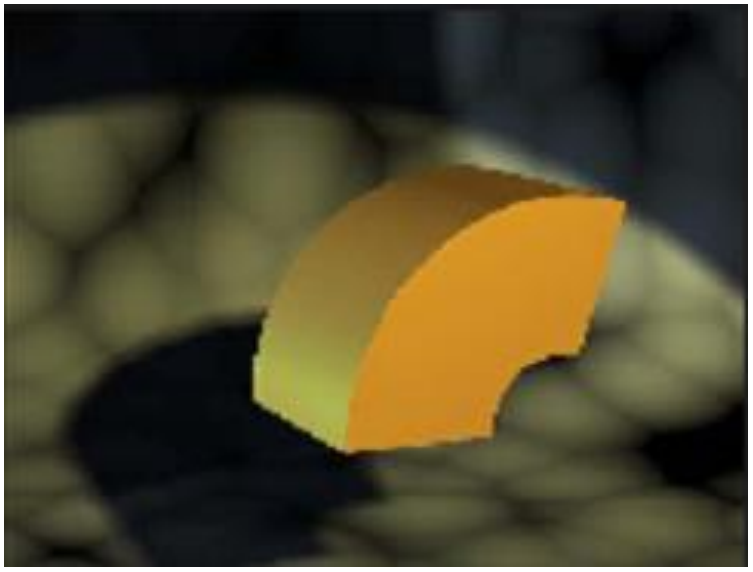
Два класса преобразований: линейные и нелинейные

- Линейные преобразования

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

- Нелинейные преобразования

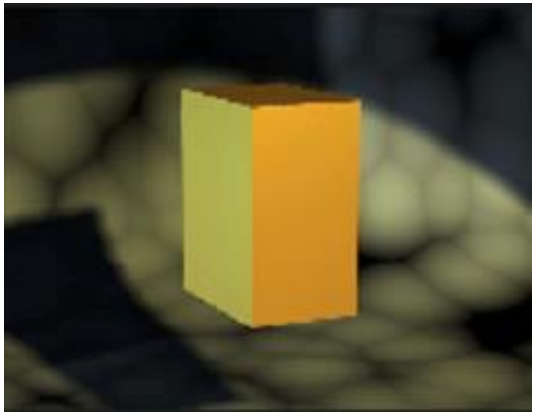


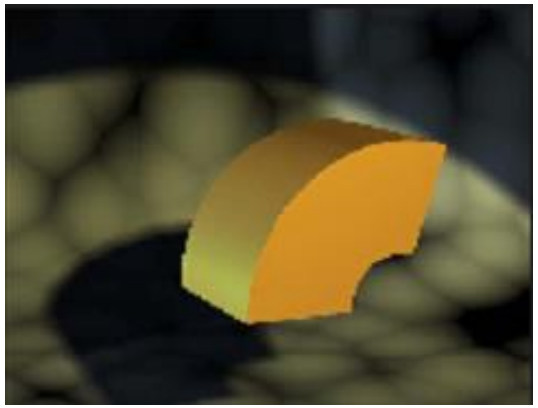


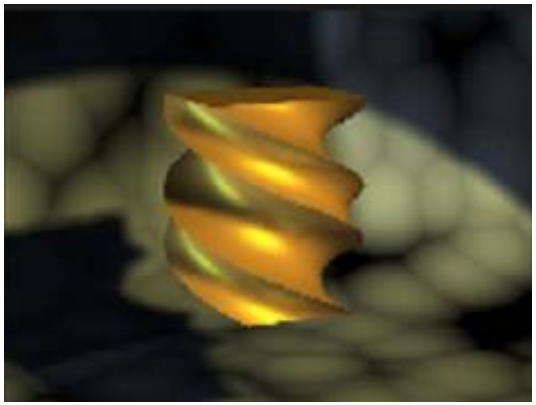
Нелинейные преобразования – произвольные деформации модели

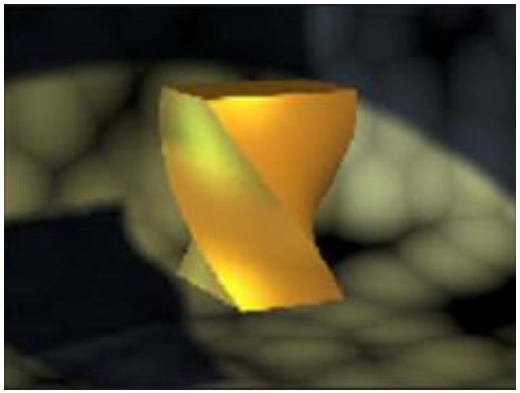
Произвольное преобразование точек модели

$$M' = T(M)$$









Линейные преобразования – интересующий нас класс преобразований

$$x' = Ax + By + Cz + D$$

$$y' = Ex + Fy + Gz + H$$

$$z' = Ix + Jy + Kz + L$$

Линейное преобразование применяется к каждой точке модели

Не изменяет топологию!

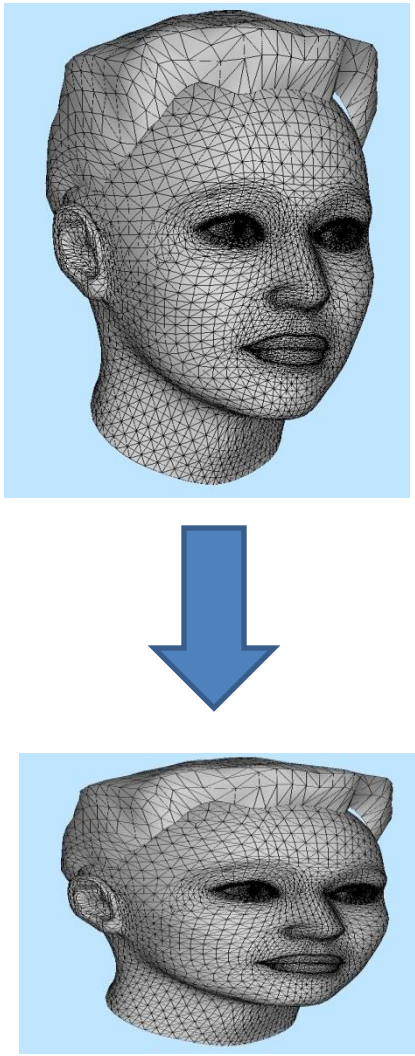
Для полигональных моделей линейные преобразования достаточно применить к вершинам!

Для полигональных моделей достаточно применить преобразование к вершинам модели!

- Линейная интерполяция

Алгоритмически эффективно, легко векторизуется

Растеризация основана на линейных преобразованиях

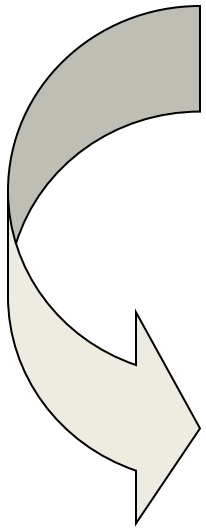


Преобразования можно записать в матричной форме

$$x' = Ax + By + Cz + D$$

$$y' = Ex + Fy + Gz + H$$

$$z' = Ix + Jy + Kz + L$$



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

4-я координата W важна!

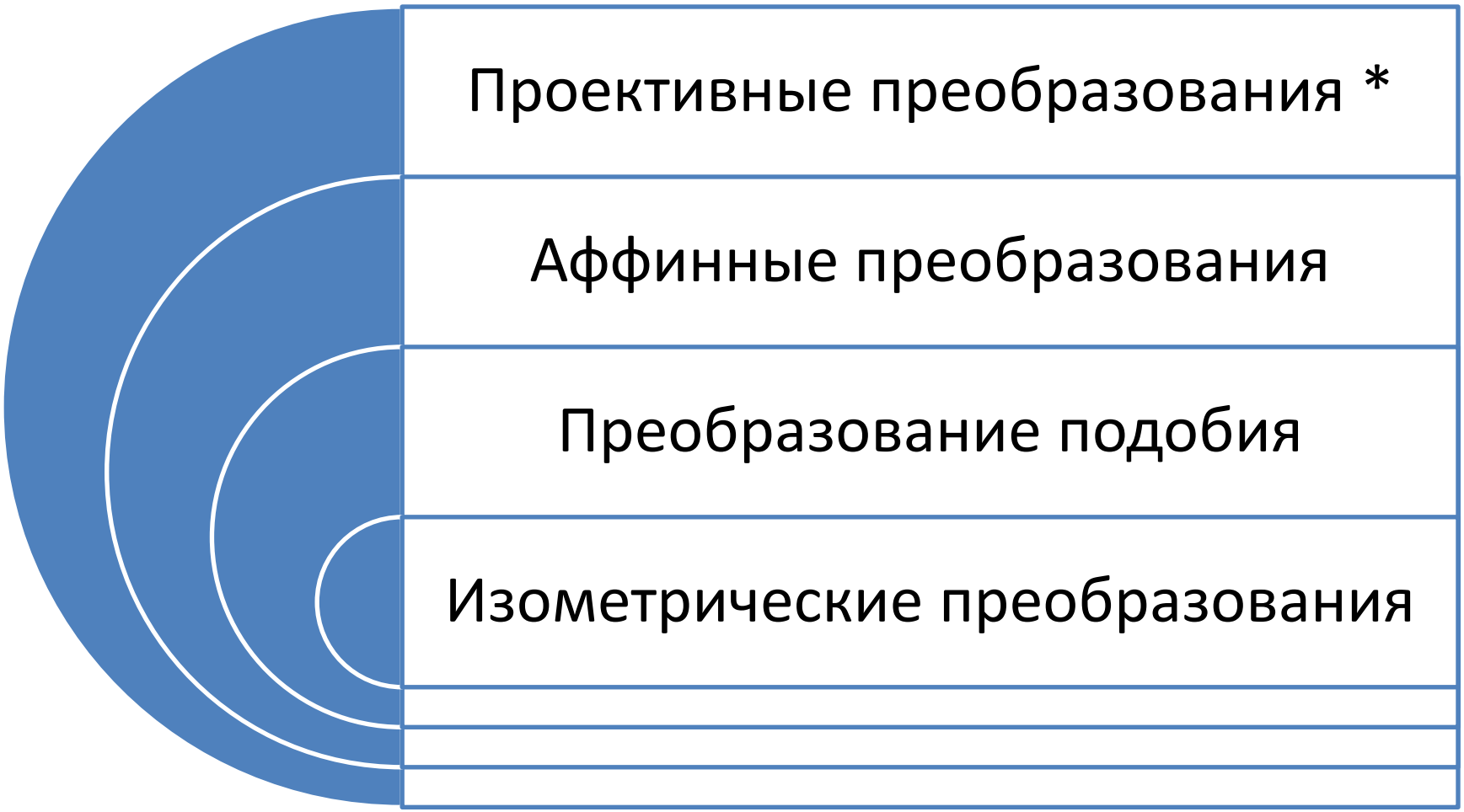
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

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

- Позволяет использовать матричную запись для всех линейных преобразований (если использовать матрицы 3x3, невозможно представить перенос)

- Позволяет описать так называемой перспективное деление (нужно для проекции)

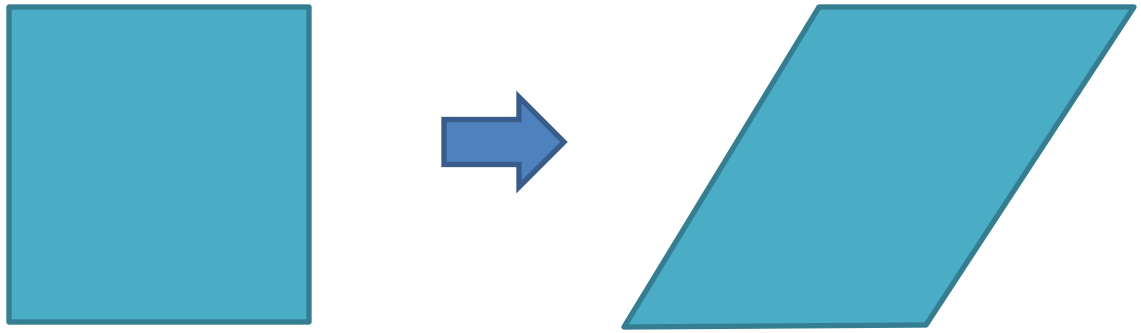
Типичные линейные преобразований



Аффинные преобразования – сохраняют параллельность

Линейные +

- w = 1
- Сохраняется параллельность линией
- Пример: сдвиг



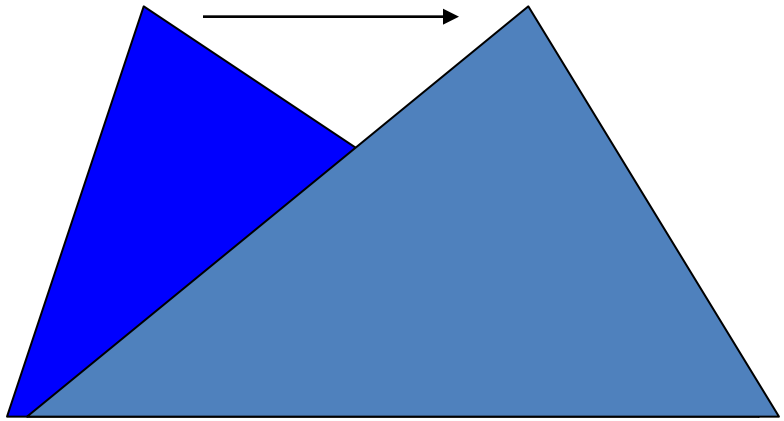
Сдвиг

$$x' = x + ay$$

$$y' = y + bx$$

Аффинное преобразование

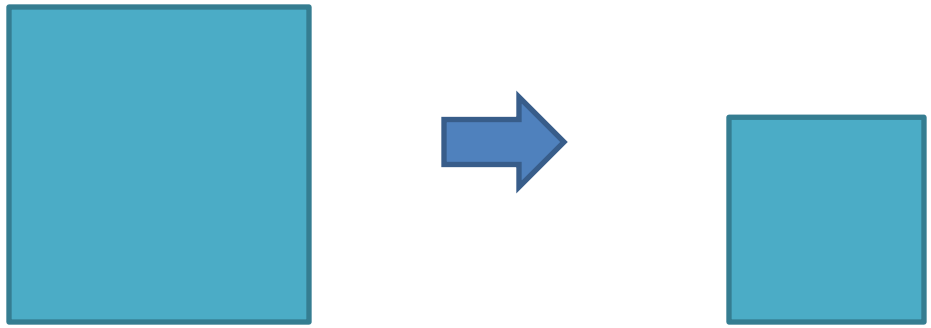
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & a & 0 & 0 \\ b & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



Преобразования подобия – сохраняют углы

Аффинные +

- Сохраняются углы
- Пример: равномерное масштабирование



Масштабирование

$$x' = ax$$

$$y' = by$$

$$z' = cz$$

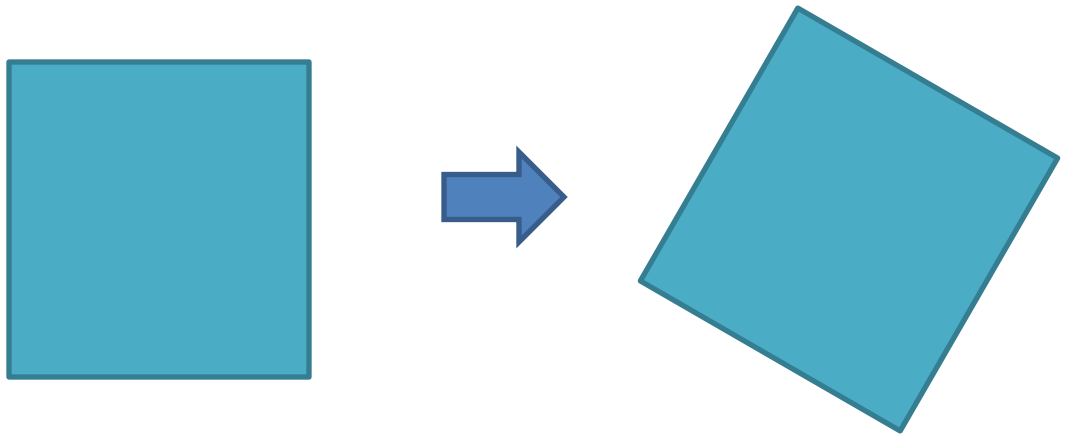
Равномерное = преобразование подобия

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

Изометрические преобразования – сохраняют размеры

Подобия +

- Сохраняются расстояния
- Пример: поворот, перенос



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)

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

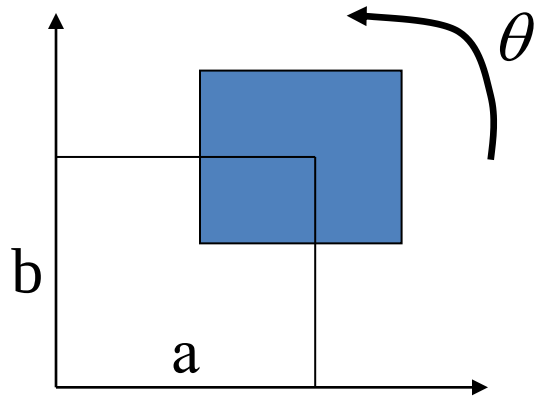
Изометрия

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

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

Задача

$$x' = x + ay$$

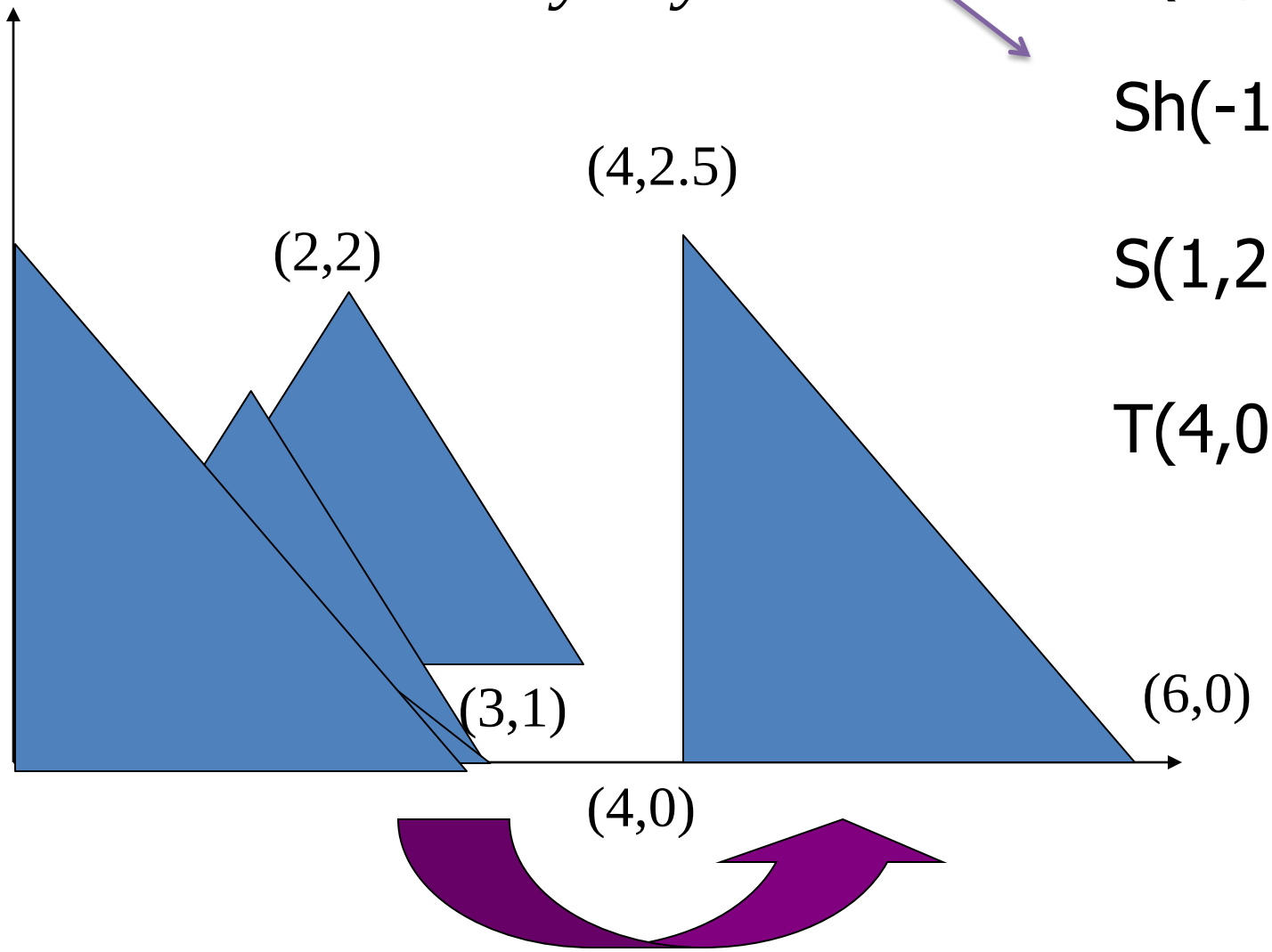
$$y' = y + bx$$

T(-1,-1)

Sh(-1, 0)

S(1,2.5)

T(4,0)



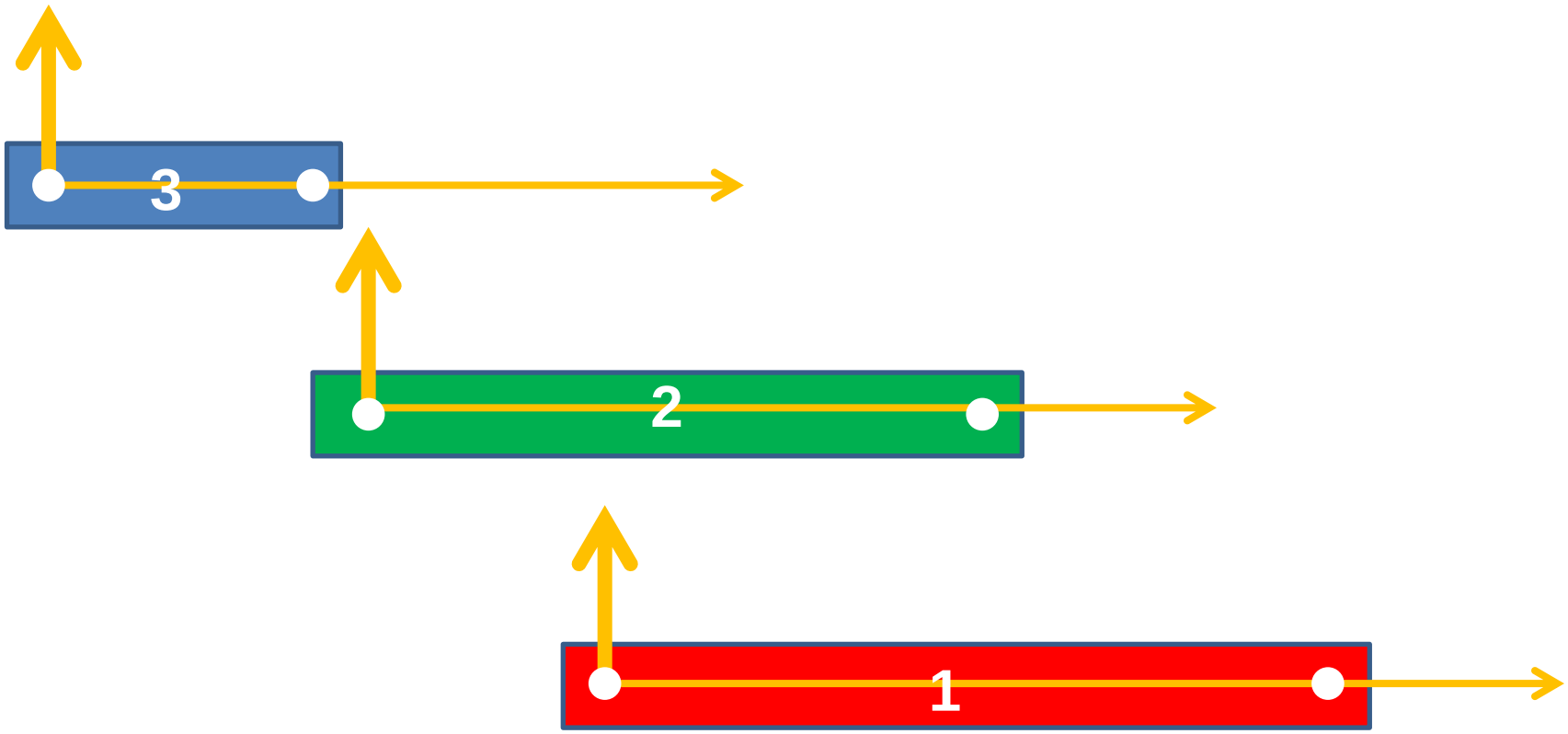
Решение задачи

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)

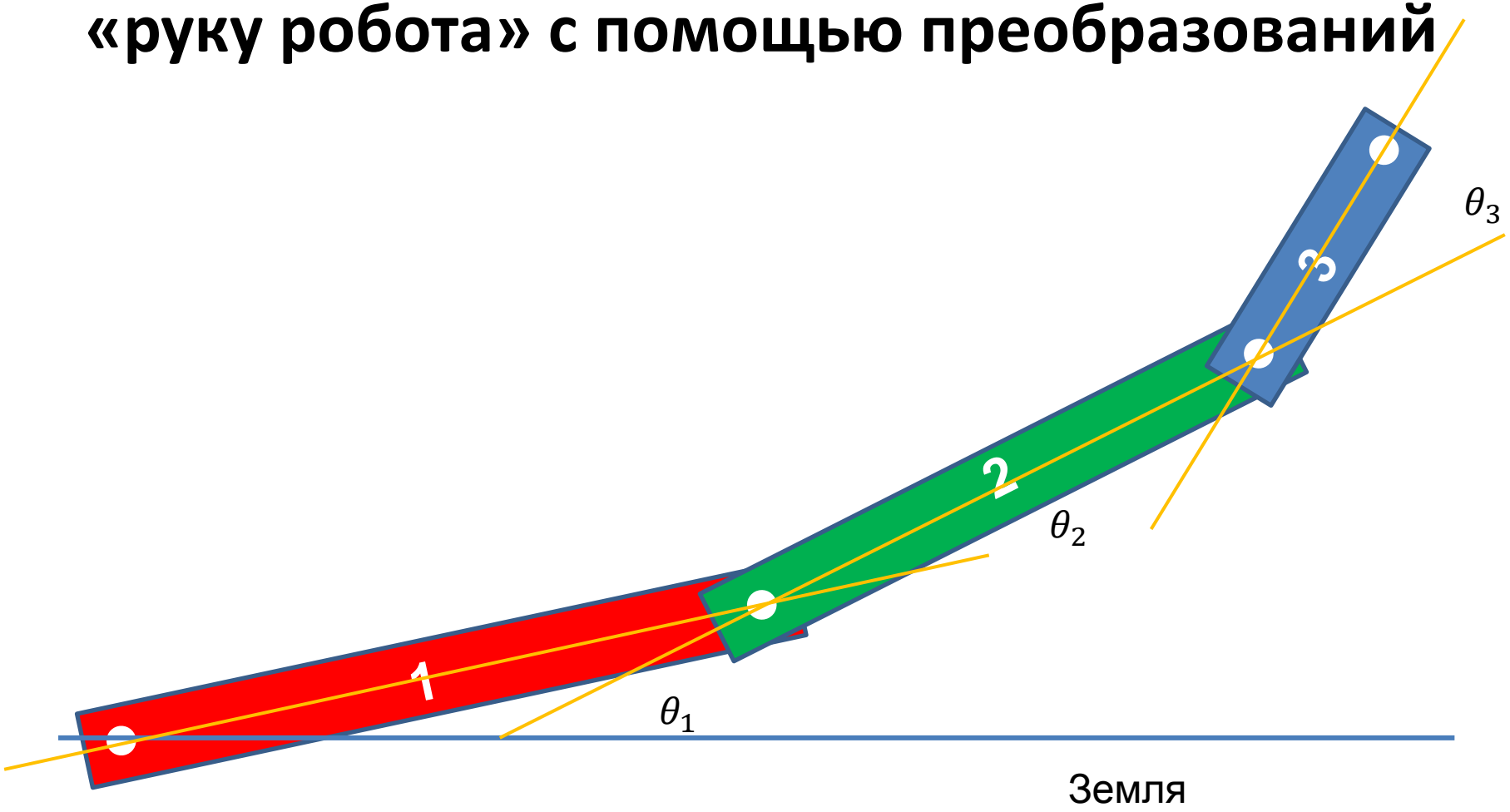
P’= M * P

Иерархия преобразований: даны три блока



Идея слайда © University at California. San Diego Supercomputer Center SDSC

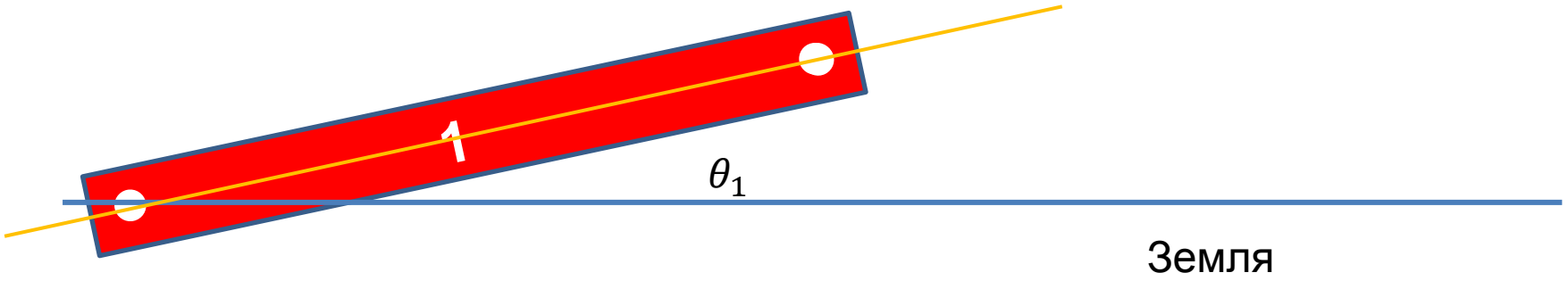
Иерархия преобразований: цель – составить «руку робота» с помощью преобразований



Идея слайда © University at California. San Diego Supercomputer Center SDSC

Шаг 1: двигаем и поворачиваем блок 1

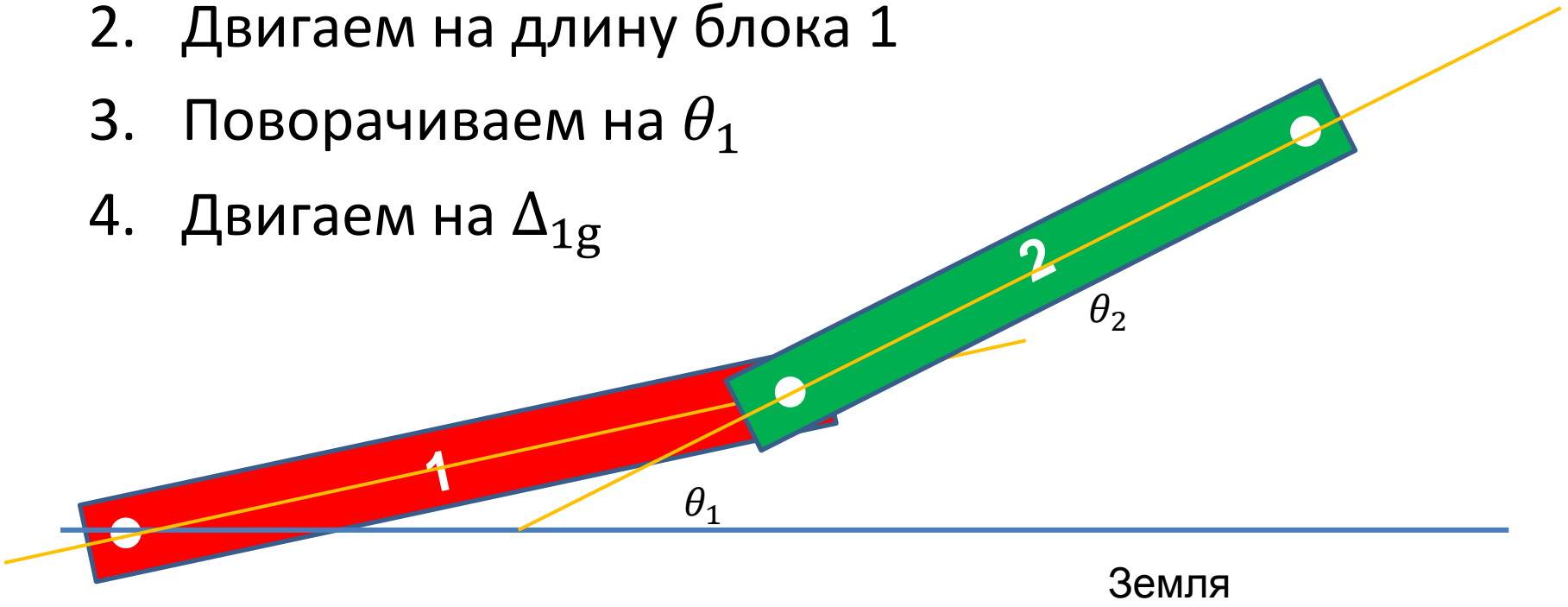
1. Поворачиваем на θ_1
2. Двигаем на Δ_{1g}



Получаем $M_{1g} = T_{1g} * R_{\theta_1}$

Шаг 2: двигаем и поворачиваем блок 2

1. Поворачиваем на θ_2
2. Двигаем на длину блока 1
3. Поворачиваем на θ_1
4. Двигаем на Δ_{1g}

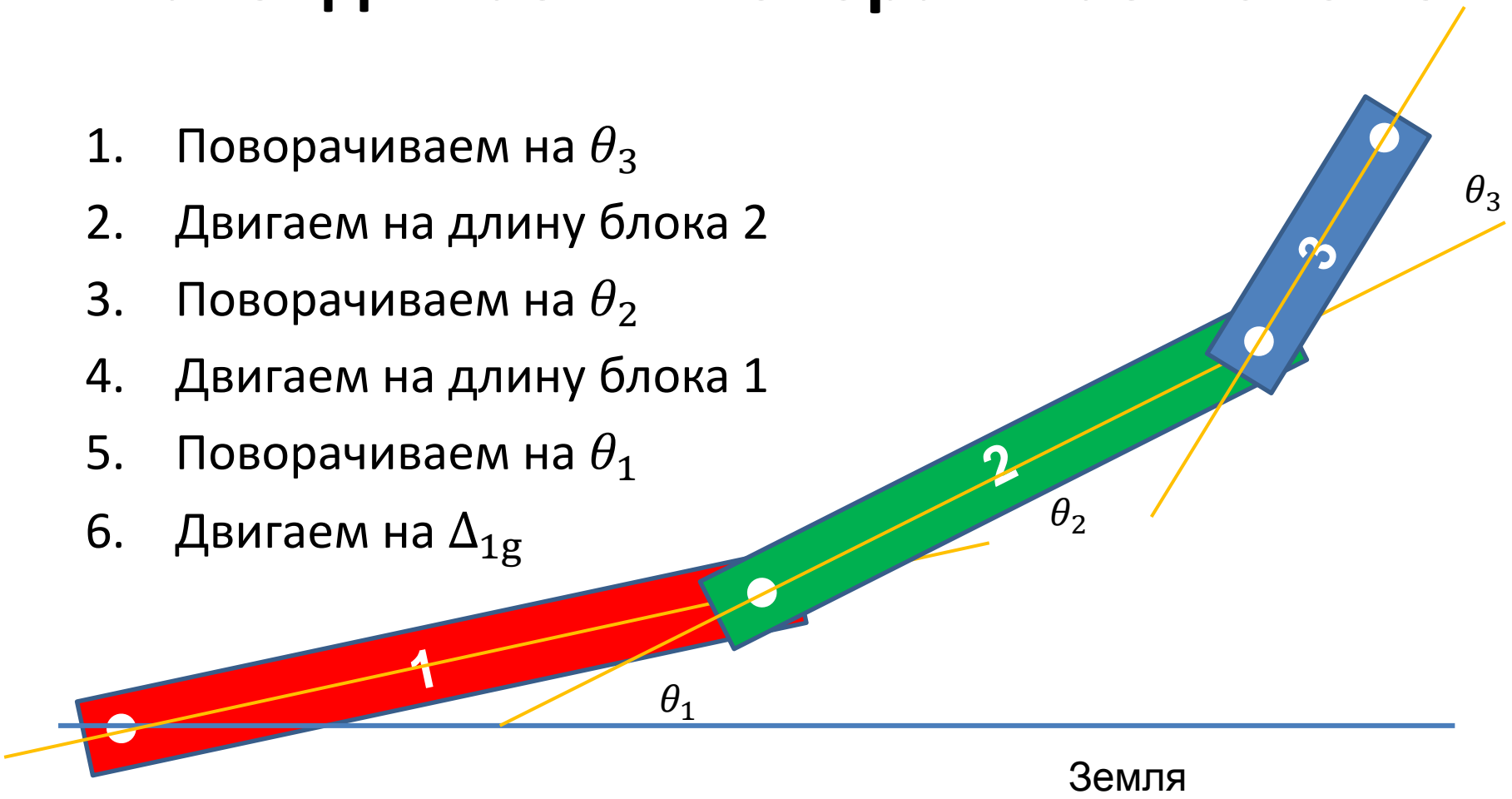


Получаем $M_{2g} = T_{1g} * R_{\theta_1} * T_{21} * R_{\theta_2} =$

$= M_{1g} * T_{21} * R_{\theta_2} = M_{1g} * M_{21}$

Шаг 3: двигаем и поворачиваем блок 3

1. Поворачиваем на θ_3
2. Двигаем на длину блока 2
3. Поворачиваем на θ_2
4. Двигаем на длину блока 1
5. Поворачиваем на θ_1
6. Двигаем на Δ_{1g}



Получаем $M_{3g} = T_{1g} * R_{\theta_1} * T_{21} * R_{\theta_2} * T_{32} * R_{\theta_3} =$

$= M_{1g} * M_{21} * T_{32} * R_{\theta_3} = M_{1g} * M_{21} * M_{32}$

Иерархия преобразований: итог

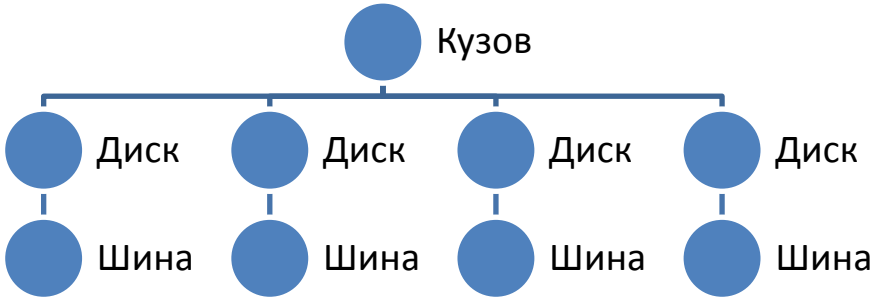
- $M_{1g} = T_{1g} * R_{\theta_1}$
- $M_{2g} = M_{1g} * M_{21}$
- $M_{3g} = M_{2g} * M_{32}$

- 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

Сложные модели создаются с помощью иерархии преобразований

Сложные геометрические сцены создаются путем иерархии моделей со своими преобразованиями

(c) wikimedia

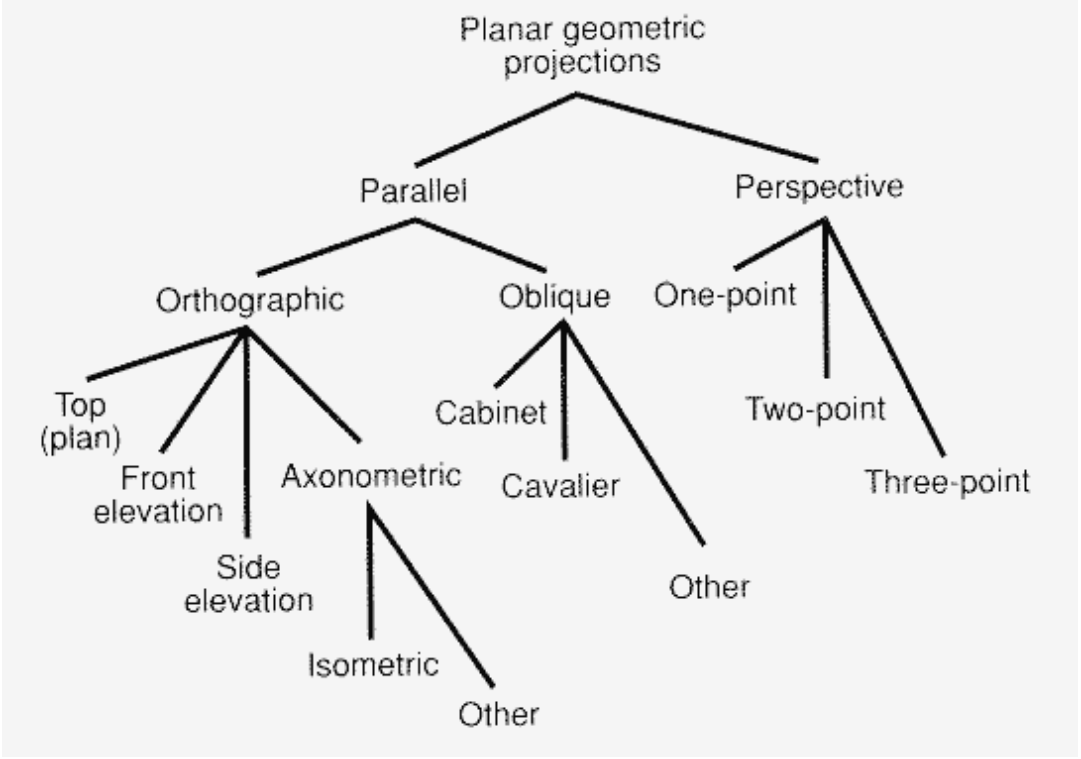


Модель наблюдателя. Проективные преобразования

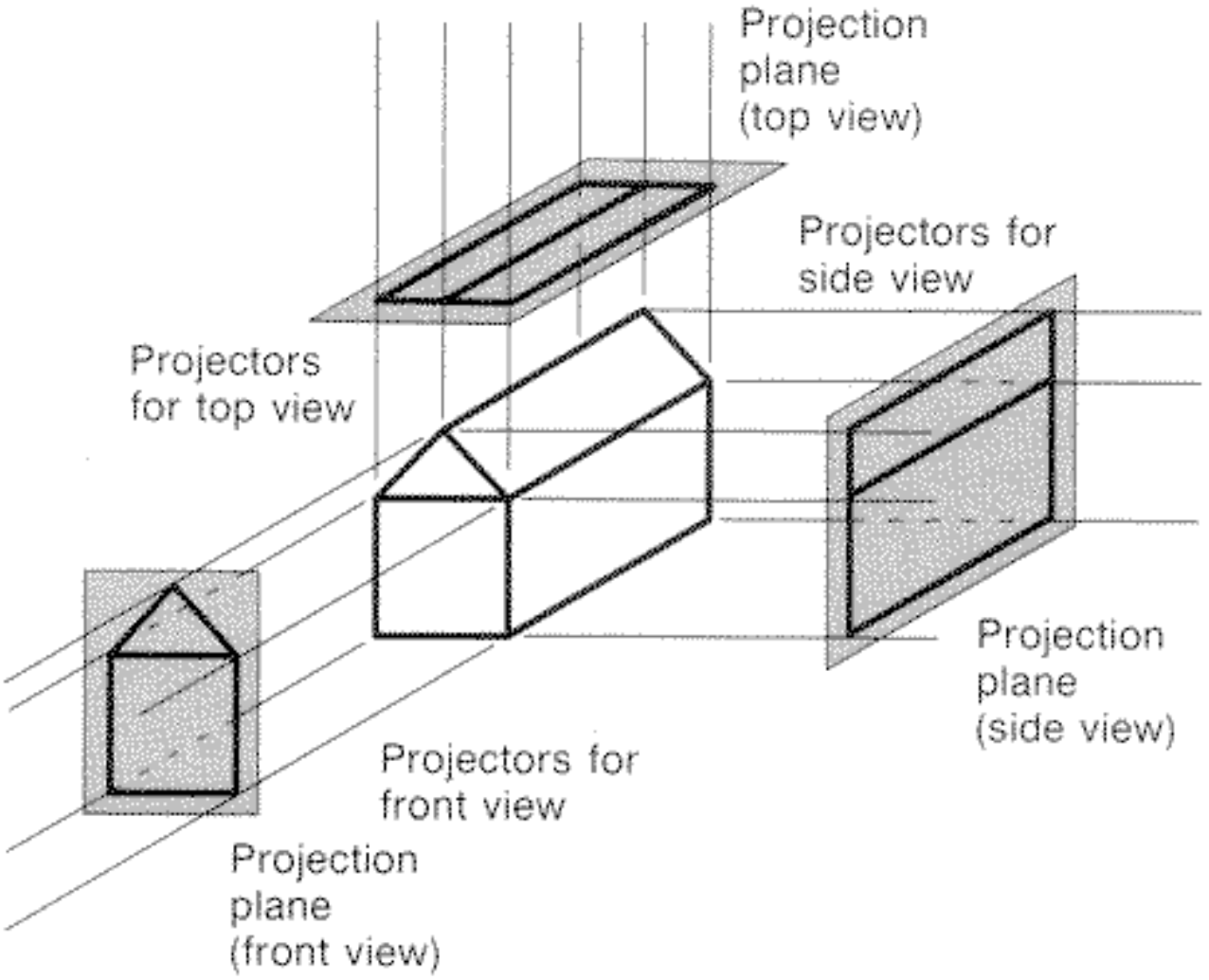
- Все современные дисплеи визуализируют изображение => необходимо преобразовать 3D данные в 2D !
- Важнейший класс преобразований
- Для выполнения таких преобразований применяются проективные преобразования
- Описываются матрицей 4x4 (линейным преобразованием)

Типы проекций

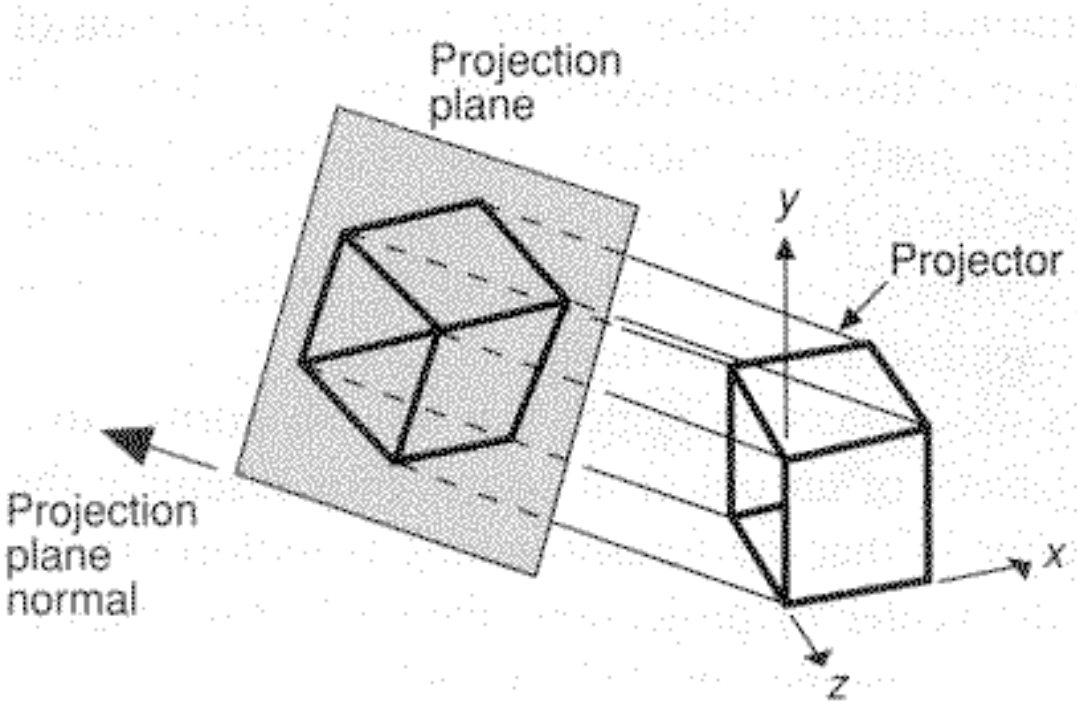
- Много разновидностей
  - Применяются в дизайне и т.п.
- Основные виды
  - Параллельные
    - Ортографические
    - Косоугольные
  - Перспективные
    - 1,2,3-х точечные

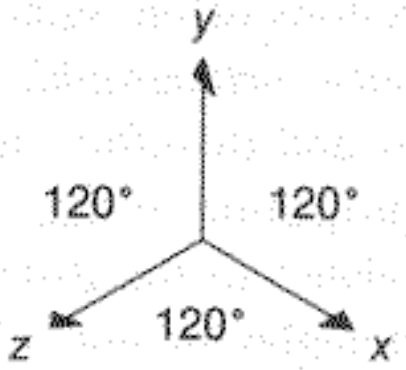


Ортографическая проекция – вдоль осей

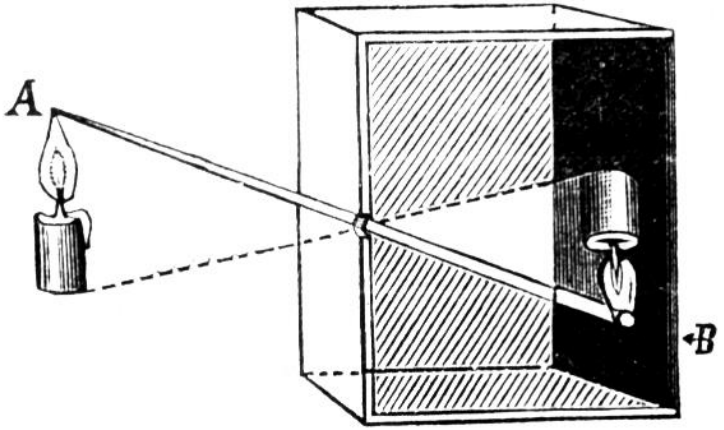


Изометрическая проекция – размеры сохраняются

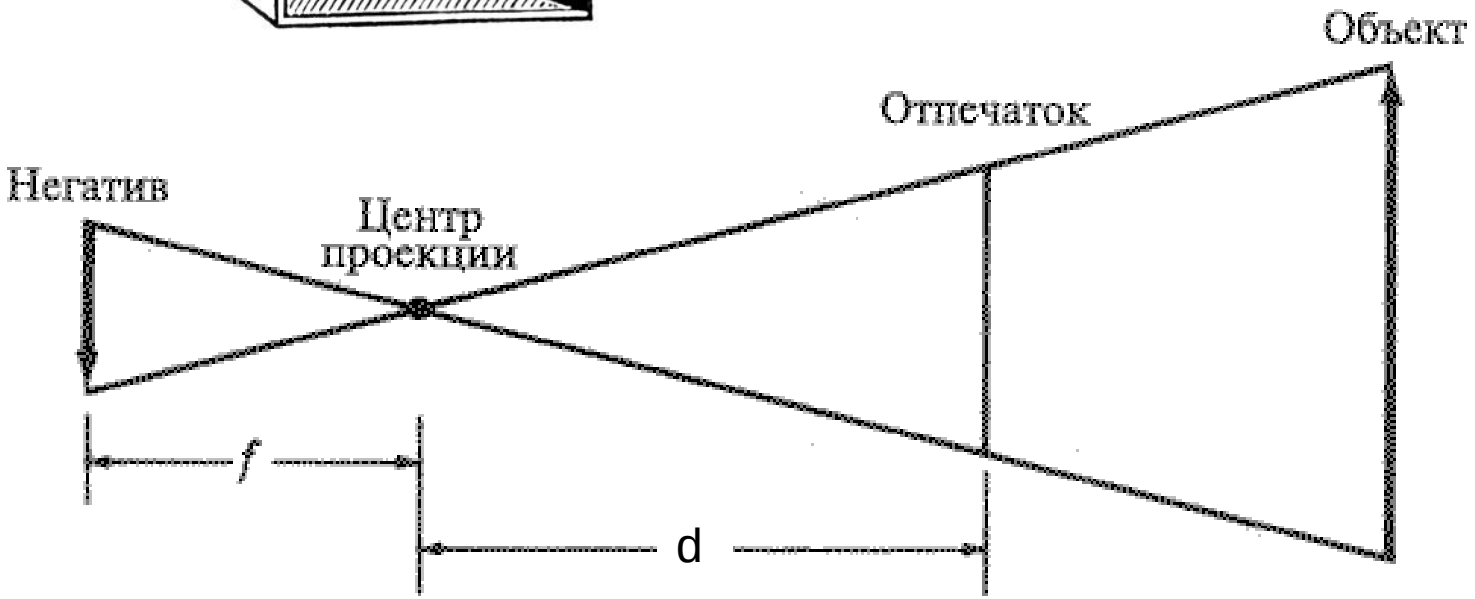




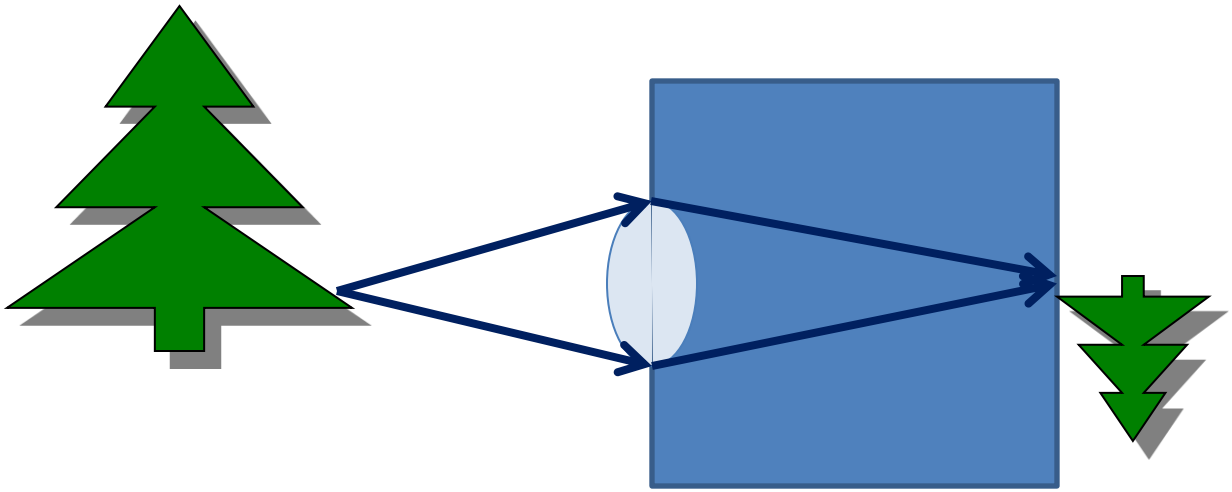
Перспективная проекция пришла из фотографии



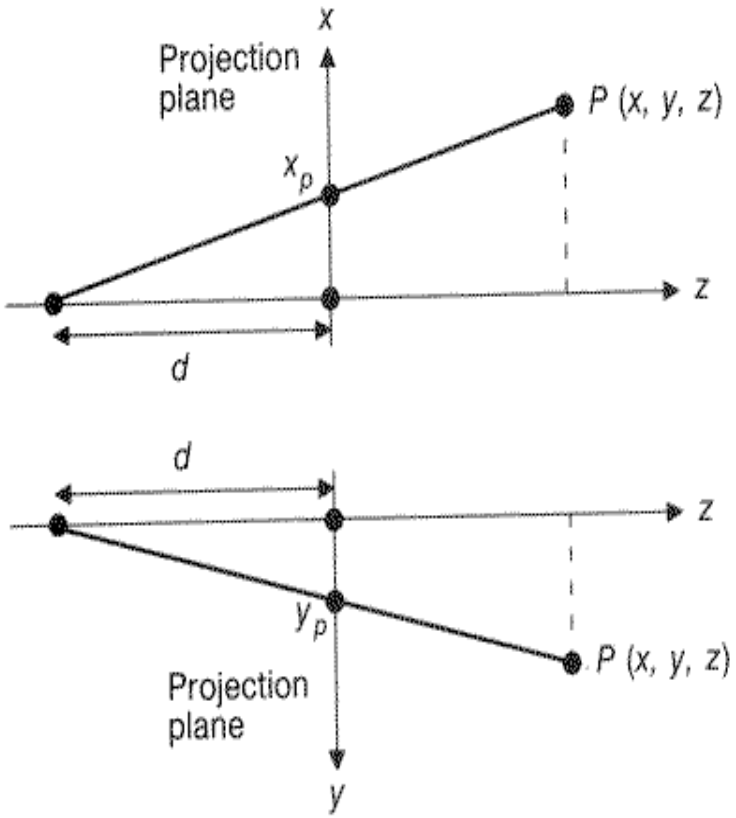
Модель объектива (камера-обскура) с бесконечно малым размером диафрагмы



Оптические системы с линзой



Математическая запись перспективной проекции на плоскость Oxy вдоль оси z



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

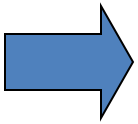
$$x_p = \frac{d \cdot x}{z+d} = \frac{x}{(z/d)+1}$$

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

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!

$$x = x / w$$

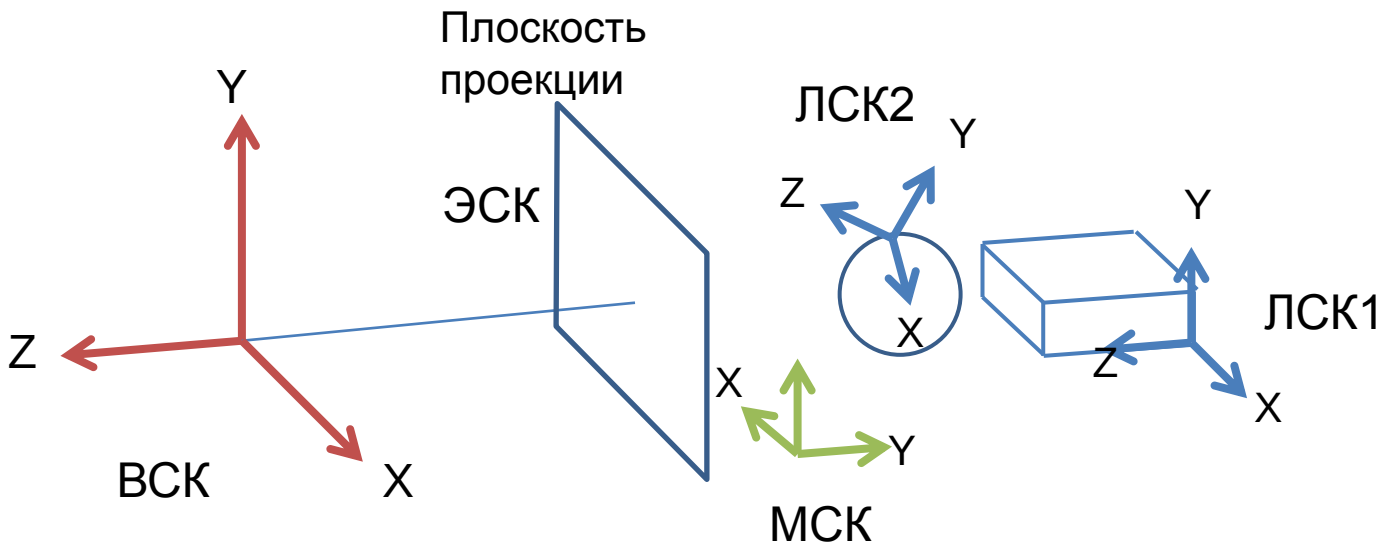
$$y = y / w$$

$$z = z / w$$

В классическом конвейере 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)

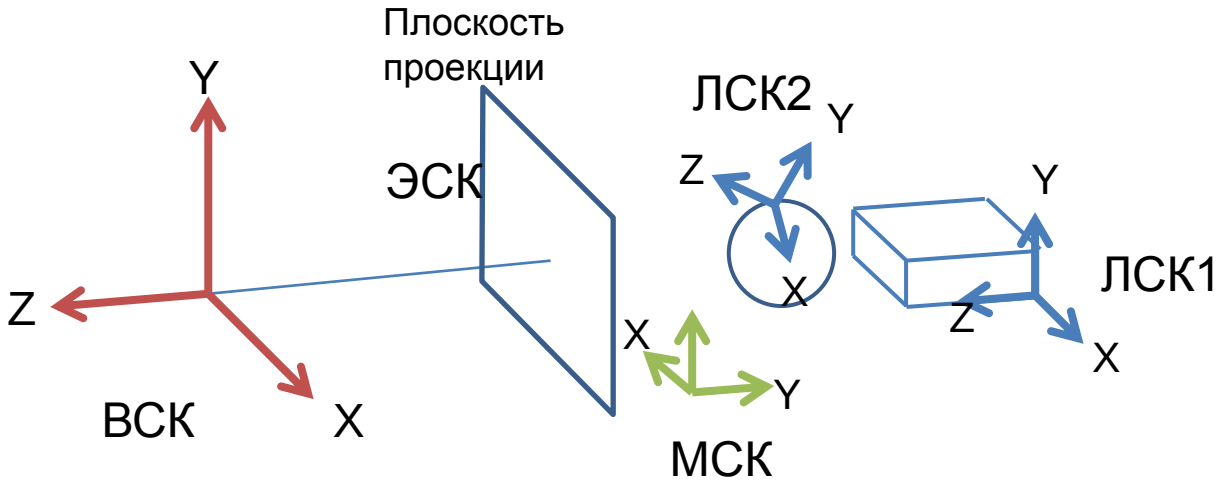


Модельное преобразование – из локальных в мировые координаты

Переводит модель, заданную в локальных (собственных) координатах, в глобальное (мировое пространство)

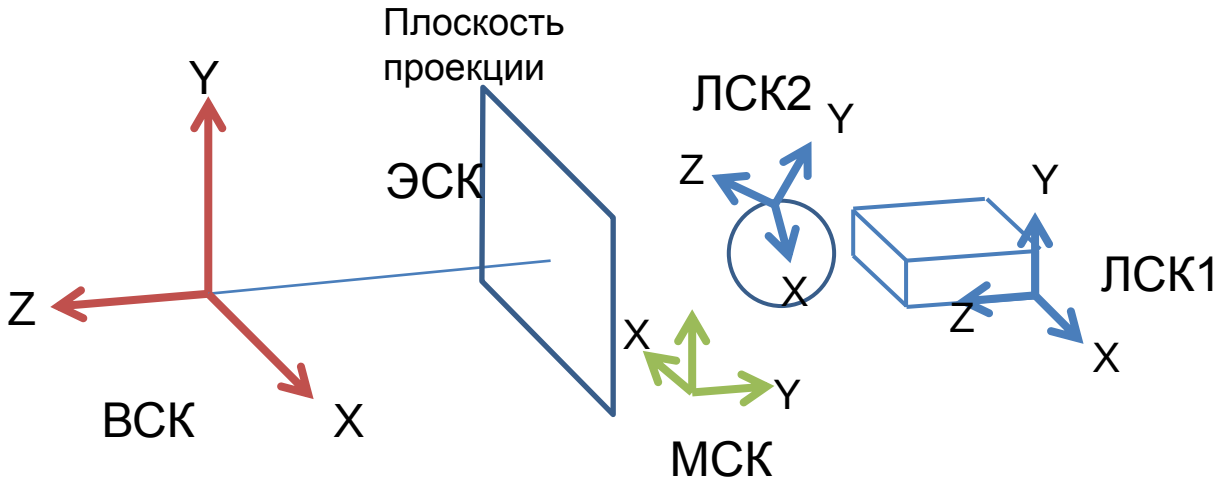
Модель «собирается» из частей, с помощью модельных преобразований (обычно композиция переносов, поворотов, масштабирования)

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

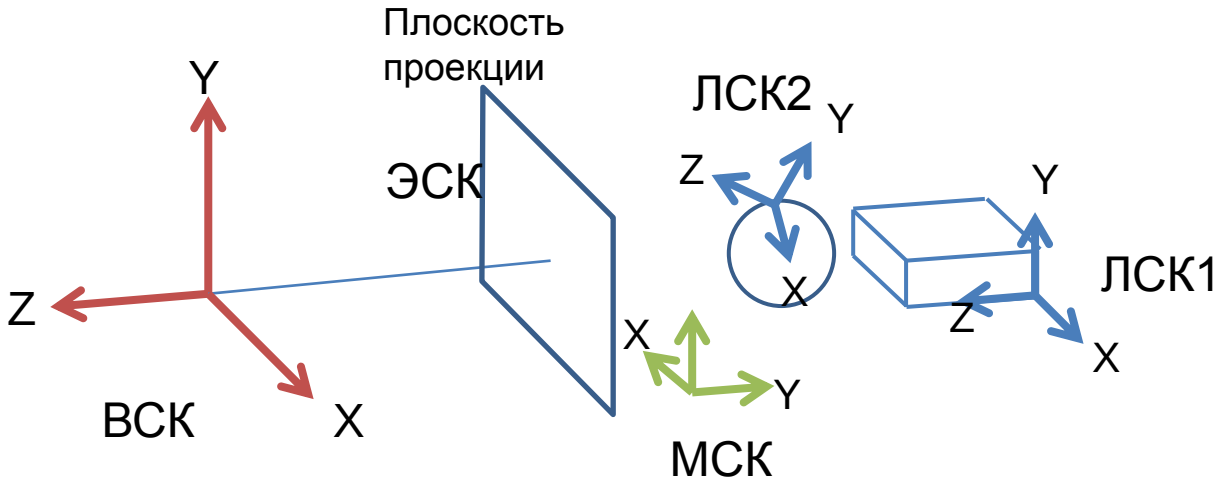
- Что если мы хотим переместить наблюдателя?

  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)

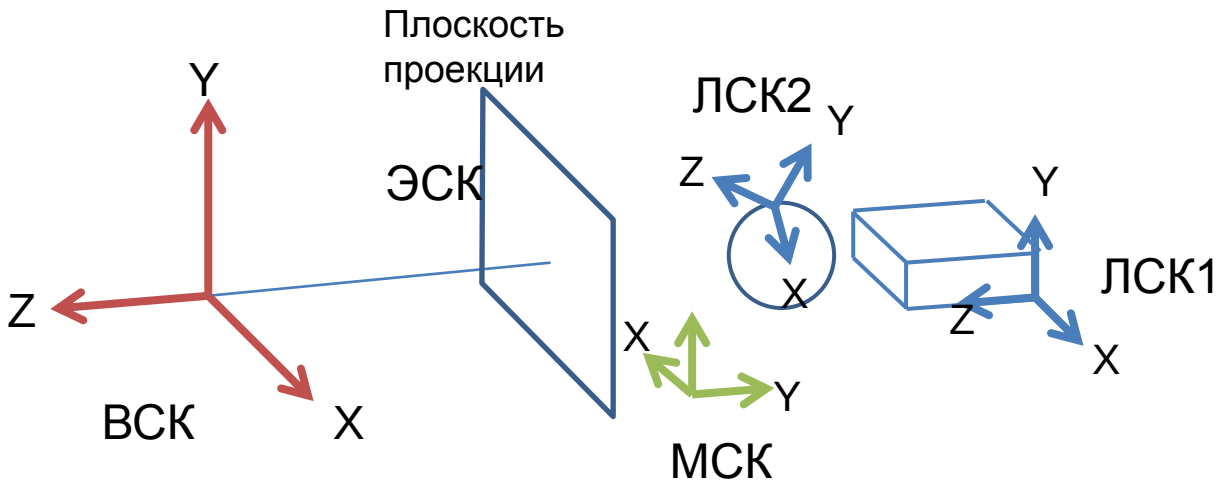
Видовое преобразование – из мировых в видовые координаты

- «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



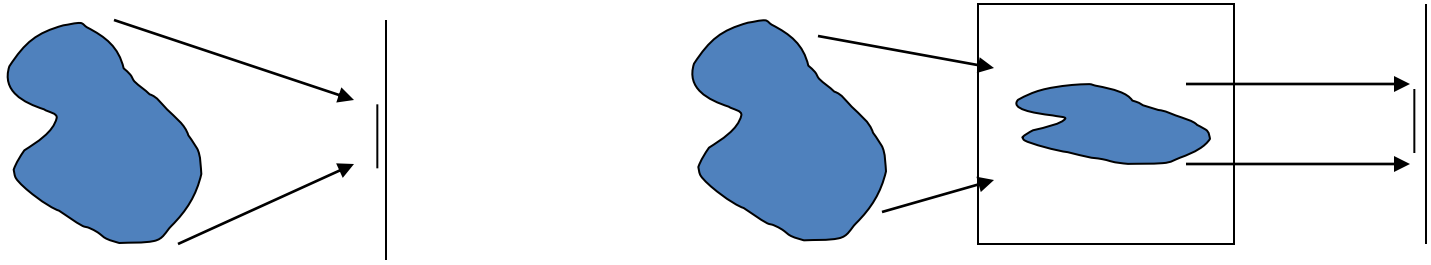
Проективное преобразование – перспективные и масштабные искажения

- Выполняет 3D преобразование, подготавливая модель к переходу на 2D



Проективное преобразование vs. проекция (1/2)

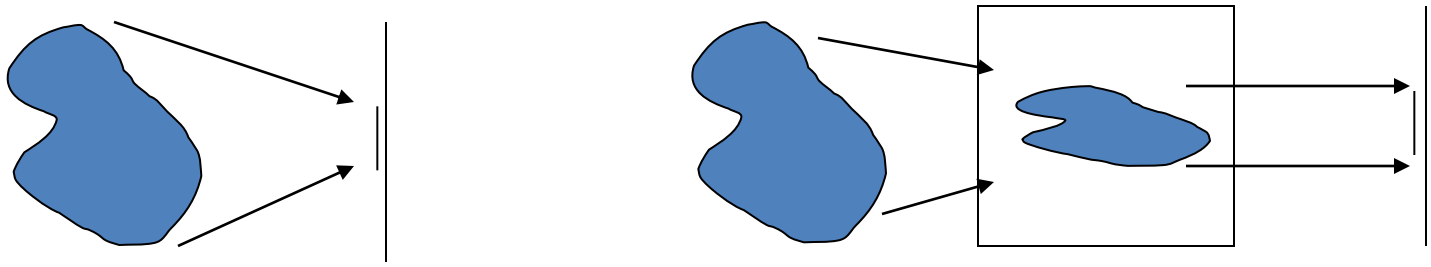
- Матрица проекции вырожденная
  - Фактически, информация от координате z теряется
- Часто необходимо выполнять дополнительные действия уже ПОСЛЕ проецирования
  - Например, удаление невидимых линий/поверхностей
- Поэтому часто (e.g. в OpenGL) используется проективное преобразование вместо проекции
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



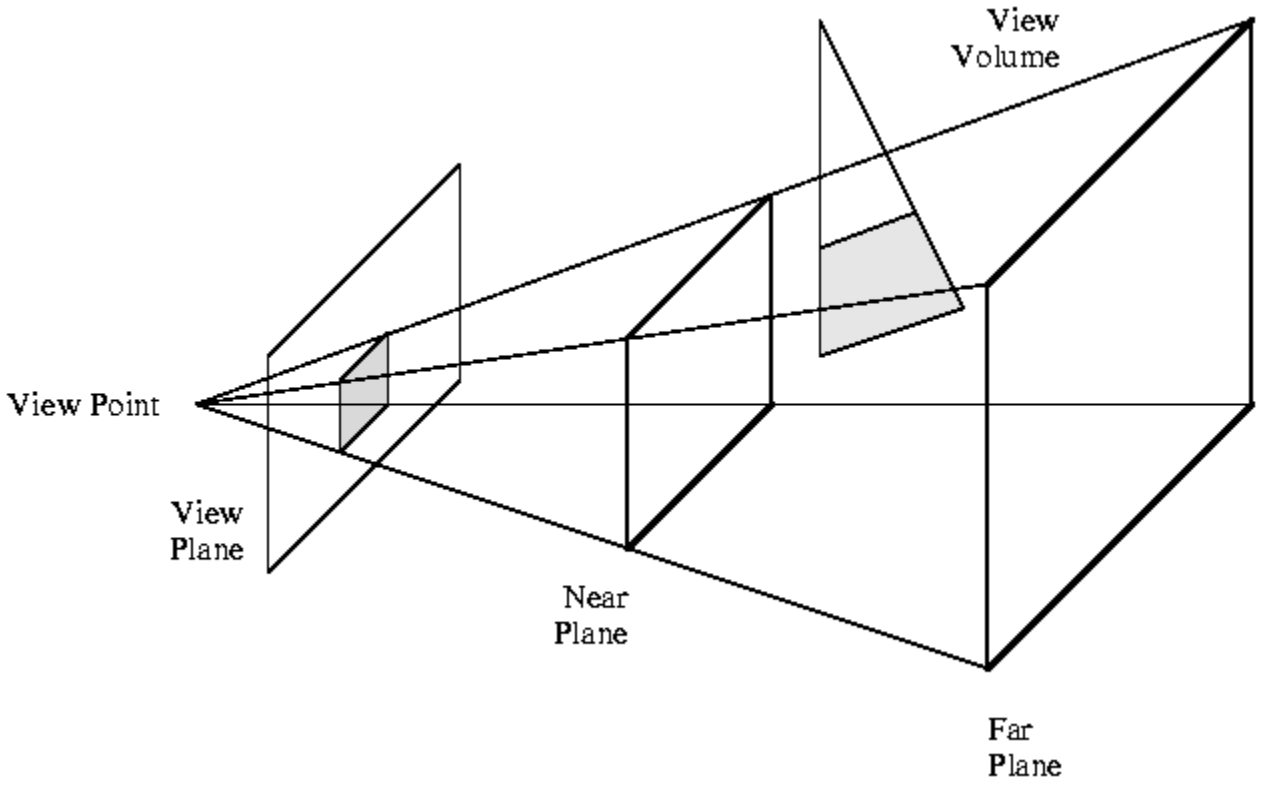
Проективное преобразование vs. проекция (2/2)

- 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]

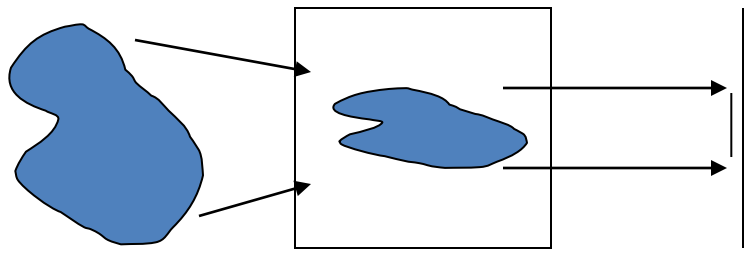


При задании проективного преобразования необходимо указать границы отсечения



Преобразование в экранные координаты – простое отбрасывания z-координаты

1. Отбрасываем координату z
2. Умножаем на высоту/ширину окна
   – Получаем экранные координаты

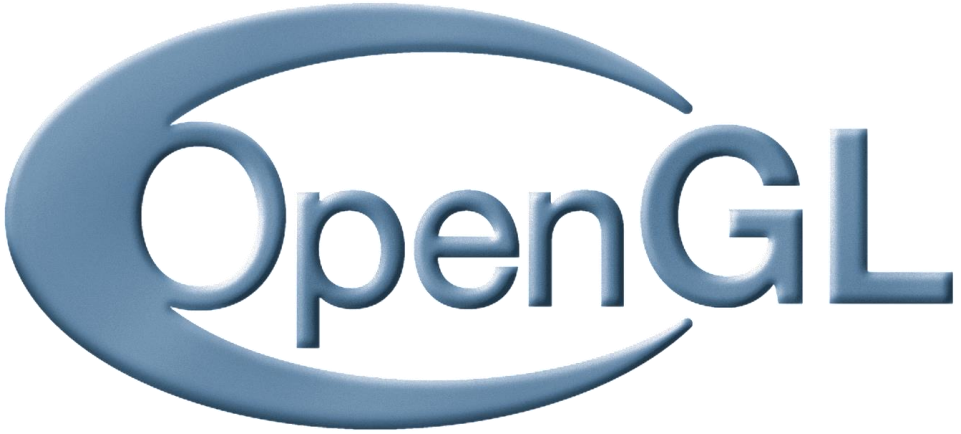


Лекция из трех частей: алгоритм растеризации, OpenGL, геометрические преобразования

- Алгоритм синтеза изображений с помощью растеризации
- Геометрические преобразования
- OpenGL: Архитектура и основные функции

OpenGL – это программная библиотека для создания 3D-приложений

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



Является отраслевым стандартом с 1992 года. Основой стандарта стала библиотека IRIS GL, разработанная фирмой Silicon Graphics Inc.

Основная функция: интерактивная визуализация трехмерных моделей


NVIDIA.
© NVIDIA CORPORATION 2002


www.yeah the movie.de
(c)2002 Spellcraft Studio
nVIDIA.


ATi RADEON 9700

OpenGL проста для изучения и поэтому она выбрана для вводного курса графики

OpenGL

- 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

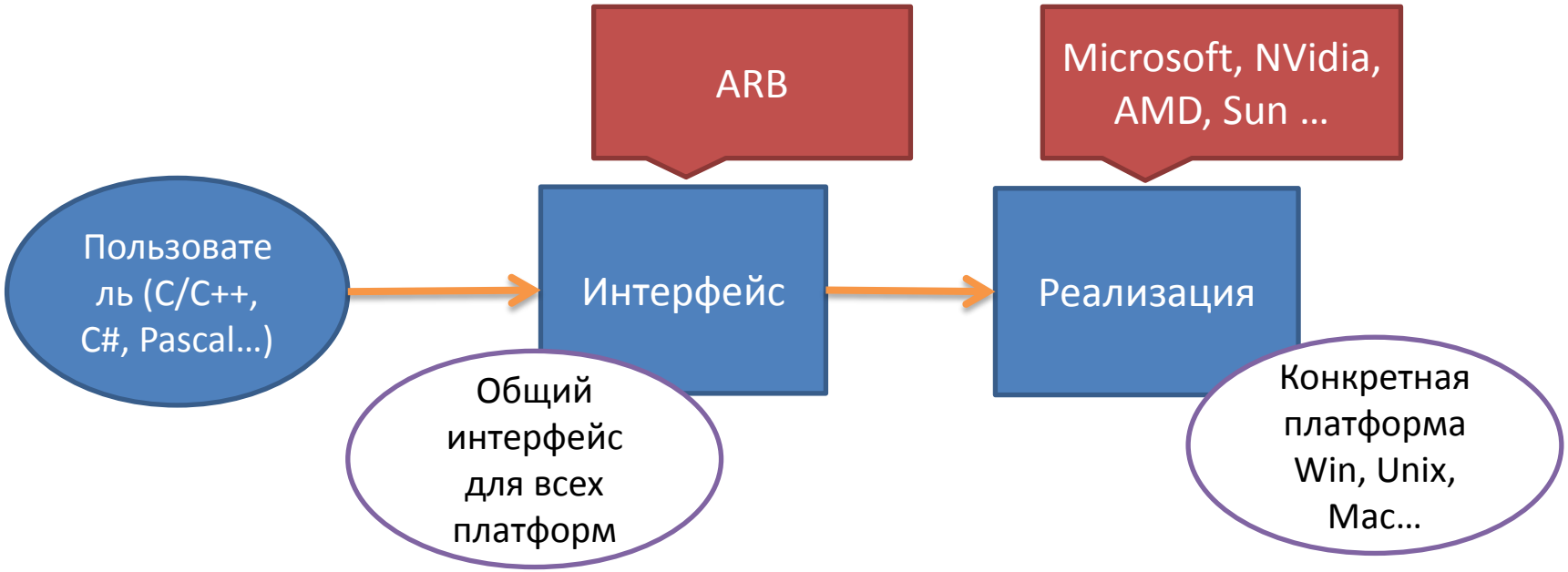
Подходит для обучения студентов!

Аналогичные библиотеки: DirectX

OpenGL – это API и реализация

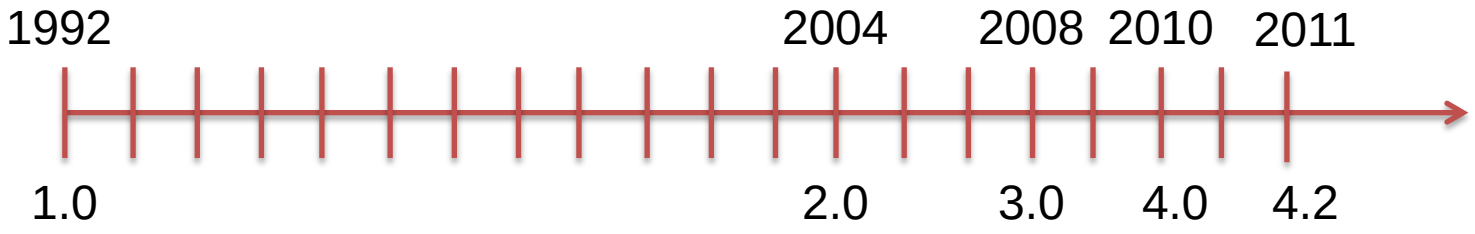
Стандартизируется прикладной программный интерфейс (API)

- Реализация своя для каждой платформы
- Может существовать несколько реализаций



История OpenGL: четыре версии за 18 лет

- 1992 1.0
- 1997 1.1
  - Текстурные объекты
  - Вершинные массивы
- 1998 1.2
  - 3D текстуры
- 2001 1.3
  - Кубические текстуры, мультитекстурование
  - Мультисэмплинг
- 2002 1.4
- 2003 1.5
- 2004 2.0
  - Язык GLSL, поддержка шейдеров
- 2006 2.1
  - Pixel Buffer Objects
- 3.0 2008
  - Frame Buffer Objects
  - Hardware Instancing
  - Vertex Array Object
- 3.1 2009
  - Instancing
- 3.2 2009
  - Geometry Shaders
- 3.3 2010
- 4.0 2010
  - GPU Tesselation
- 4.1 2010
  - OpenGL ES Compatibility
- 4.2 2011
- 4.3 2012



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

Конвейер OpenGL

Итоги 1/2

- Растеризация – метод синтеза изображений с помощью отображения трехмерной геометрии на экран
- 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

Итоги 2/2

- Графический конвейер: от локальной модели до точки на экране
  - Локальные, мировые, экранные координаты
- OpenGL
  - Кросс-платформенная библиотека функций для создания интерактивных 2D и 3D приложений
  - Определение геометрии
    - glVertex, glBegin, glEnd