

Виртуальная реальность

занятие №2

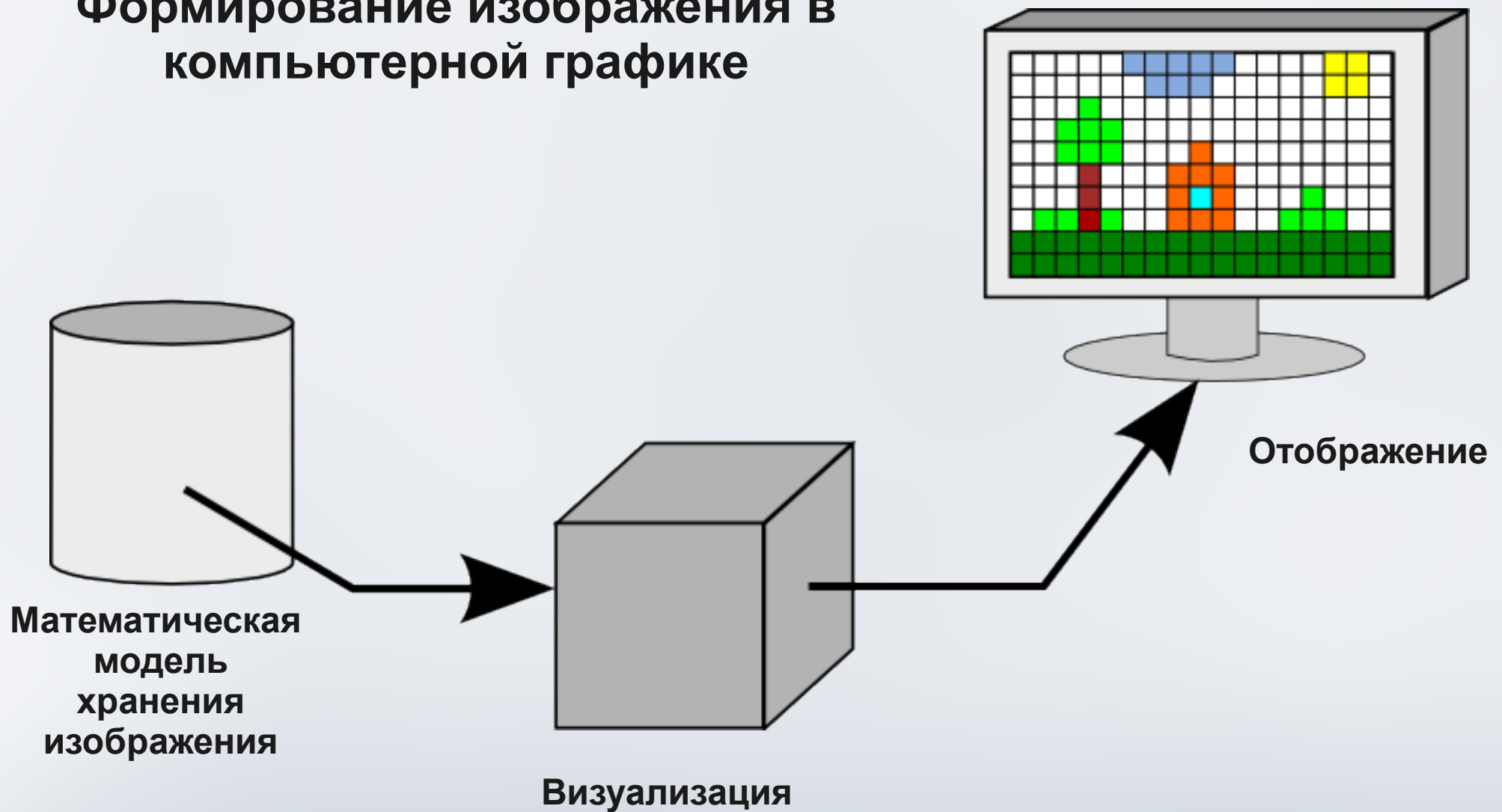
Рябинин Константин Валентинович

e-mail: icosaeder@ya.ru

jabber: icosaeder@jabber.ru

Пермь, 2012

Формирование изображения в компьютерной графике



Классификация графики по способу хранения (от способа хранения зависит визуализация)

	2D	3D
Растровая	Матрица точек	Воксели
Векторная	Описание контуров и заливок	Описание многоугольников

Классификация графики по способу хранения (от способа хранения зависит визуализация)

	2D	3D
Растровая	Матрица точек	Воксели
Векторная	Описание контуров и заливок	Описание многоугольников

Описание многоугольников – описание вершин и связей между ними

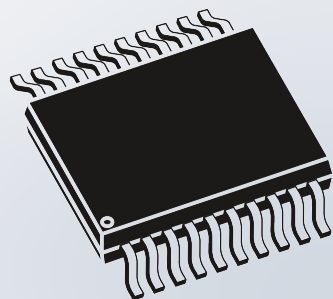
Устройство графического приложения:



Логика



Движок



Графическое
оборудование



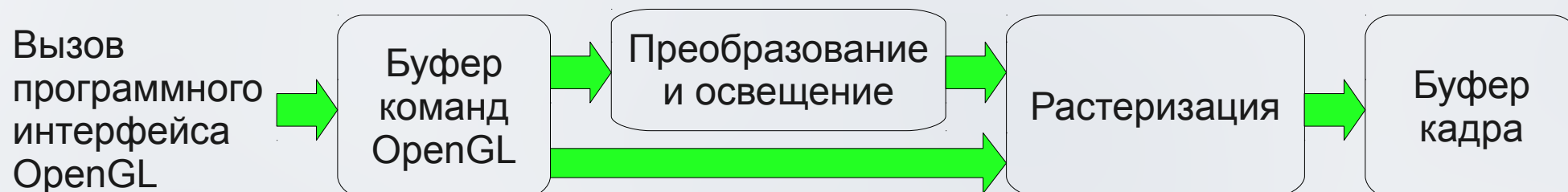
WebGL – API для языка JavaScript, позволяющее строить двумерные и трёхмерные изображения в любом совместимом браузере без использования плагинов и с поддержкой аппаратного ускорения.

- WebGL** – адаптация API OpenGL ES для языка JavaScript
- OpenGL ES** – сокращённая версия стандарта OpenGL, изначально ориентированная на использование во встраиваемых системах (на мобильных устройствах)
- OpenGL** – открытый интерфейс к графическому оборудованию для создания трёхмерных изображений в реальном времени

Основные особенности:

- Полная универсальность
- Низкий уровень функций

Конвейер



Машина состояний

- Хранилище различных настроек, представляющих собой множество пар свойство-значение
- Бинарные атрибуты изменяются функциями
`void glEnable(GLenum cap)`
`void glDisable(GLenum cap)`
- Небинарные атрибуты изменяются специализированными функциями
- Для состояний есть свой стек, управляемый функциями
`void glPushAttrib(GLbitfield mask)`
`void glPopAttrib(void)`

- Объекты в трёхмерной графике представляют собой поверхности, аппроксимированные множеством многоугольников



- Атомарная управляемая единица геометрии – **вершина**
- Вершина имеет набор атрибутов (типа float), интерпретация которых, вообще говоря, лежит на программисте:
 - Координаты в пространстве
 - Координаты нормали
 - Координаты текстуры
 - Цвет
 - ...
- Вершины объединяются в примитивы:
 - Линии
 - Многоугольники (треугольники)

- Для произведения растеризации, системе для каждой вершины необходимо получить **от программиста** как минимум следующую информацию:
 - Тип примитива, в который входит вершина (соответственно, вместе с данной вершиной должны быть указаны все остальные, входящие в примитив)
 - Проекцию вершины на плоскость экрана, выраженную **в однородных координатах**
- Перед тем, как быть спроектированной на плоскость экрана, вершина может претерпевать различные пространственные преобразования. Этими преобразованиями достигается «размещение» объектов на сцене

- Все типовые задачи «размещения» объектов решаются при помощи аффинных преобразований вершин этих объектов:
 - Параллельного переноса
 - Масштабирования
 - Поворота
- Для этого удобно использовать аппарат матриц, так как он предоставляет единый механизм осуществления как аффинных преобразований, так и преобразований проекции
- Если «классический» («олдскульный») OpenGL *заставлял* использовать матрицы (имел функции работы с ними на уровне API), новые версии дают полную свободу выбора механизма (и не имеют никаких вспомогательных функций для этого)

- При использовании матриц все преобразования сводятся к умножению вектора однородных координат вершины на матрицу преобразования, в результате чего получается вектор «новых» координат данной вершины:

$$\begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

ось X новой системы координат

ось Y новой системы координат

ось Z новой системы координат

начало новой системы координат

Матрица переноса

Матрица масштаба

Матрица поворота вокруг оси

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

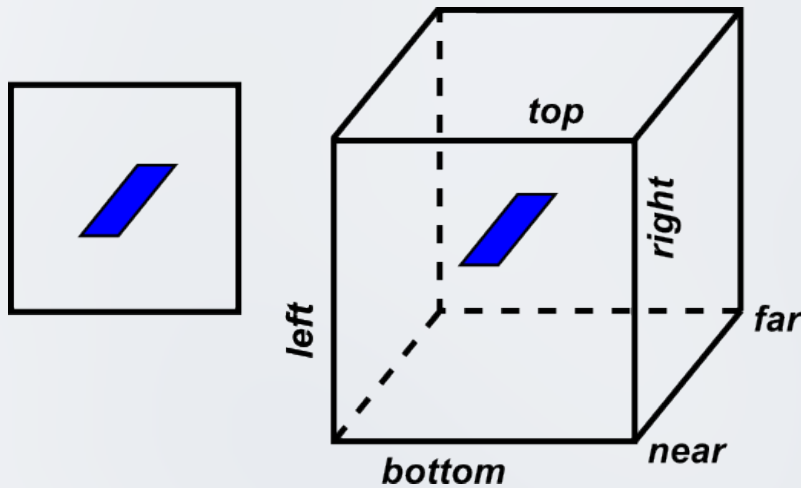
$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ yx(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ xz(1-c)-ys & yz(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

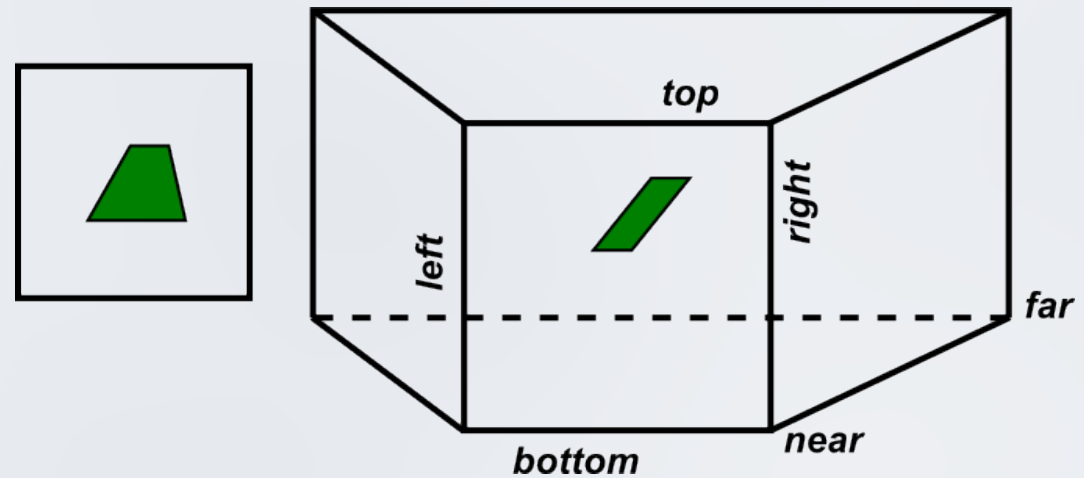
$$c = \cos \theta, \quad s = \sin \theta, \quad |(x, y, z)| = 1$$

→ Заданием параметров проекции определяется видимая область пространства

Параллельная проекция



Перспективная проекция

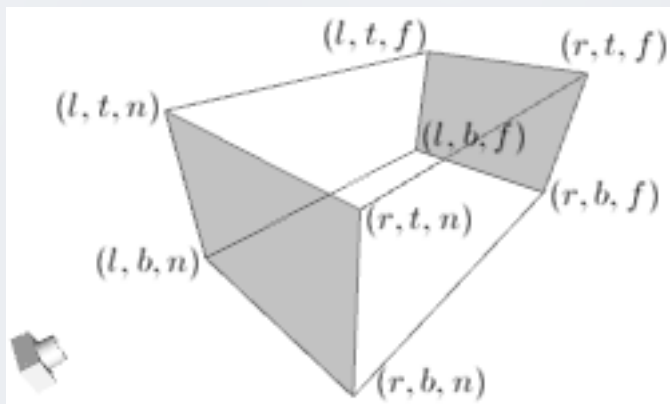


$$\begin{pmatrix} \frac{2}{\text{right} - \text{left}} & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\ 0 & \frac{2}{\text{top} - \text{bottom}} & 0 & -\frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} \\ 0 & 0 & \frac{-2}{\text{far} - \text{near}} & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

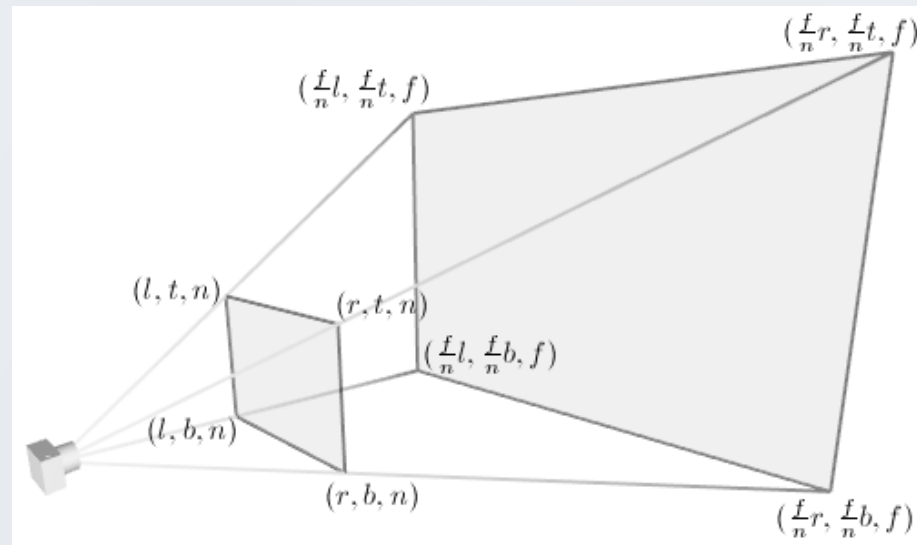
$$\begin{pmatrix} \frac{2 \text{ near}}{\text{right} - \text{left}} & 0 & \frac{\text{right} + \text{left}}{\text{right} - \text{left}} & 0 \\ 0 & \frac{2 \text{ near}}{\text{top} - \text{bottom}} & \frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} & 0 \\ 0 & 0 & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} & -\frac{2 \text{ far near}}{\text{far} - \text{near}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

- Заданием параметров проекции определяется видимая область пространства

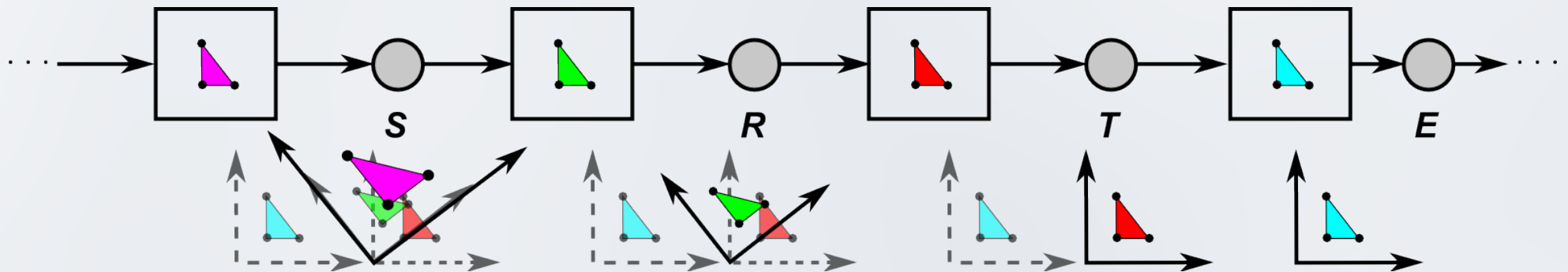
Параллельная проекция



Перспективная проекция



- Важным свойством матричных преобразований является их комбинированность:



- В связи с этим, в компьютерной графике имеется паттерн хранения и применения преобразований **Model-View-Projection**:

$$v' = (P \cdot V \cdot M) \cdot v$$

v' – вектор координат, передаваемый системе для произведения растеризации

v – вектор координат вершины

P – матрица проекции

V – матрица вида (преобразование камеры)

M – матрица модели (преобразование размещения объекта на сцене)

$$M = M_{\text{родителя}} \cdot M_{\text{объекта}}$$

- Камера – это псевдообъект в трёхмерном пространстве, характеризующий положение наблюдателя
- Камера – лишь полезная метафора, на низком уровне она выражена матричным преобразованием, математически ничем не отличающимся от всех остальных
- Часто преобразование камеры является лишь аффинным
- В связи с этим, *иногда* преобразование камеры не хранят отдельно, а «смешивают» его с преобразованием размещения, получая матрицу, которую принято называть ModelView (в «классическом» OpenGL было именно так)

- В итоге, преобразование координат, осуществляемое в графическом приложении, имеет вид:

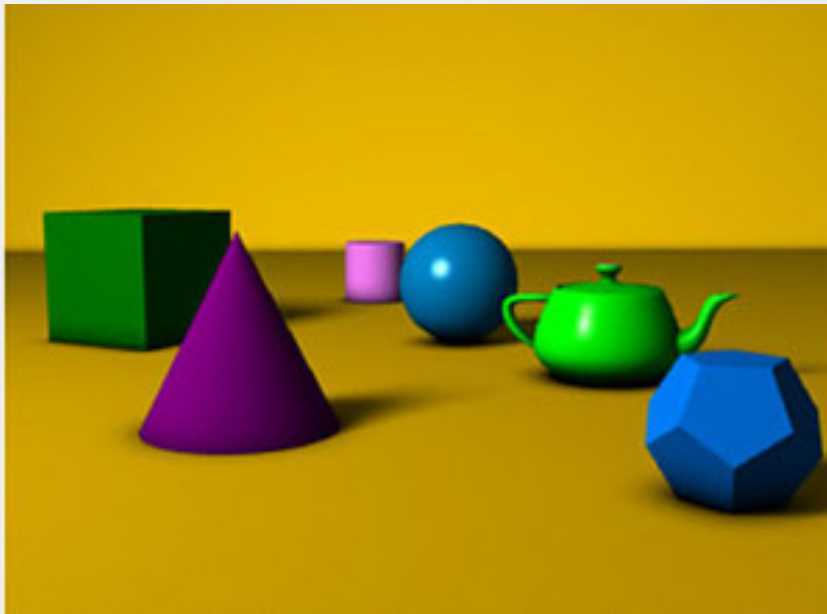


* Преобразования из **первого ряда** должен выполнять **программист**, а преобразования из **второго ряда** **система** выполняет автоматически

Буфер глубины (zBuffer) – это структура данных для сохранения глубины каждой точки изображения

- Чаще всего представлен двумерным массивом
- В современных системах реализуется аппаратно
- zBuffer характеризуется разрядностью своих ячеек
- Каждая новая точка **отрисовывается на экране** и записывается в ячейку буфера только тогда, когда уже записанное значение больше текущего (обратная ситуация носит название wBuffer)
- Если значения оказались равными (с учётом принятой погрешности) – ситуация «борьбы», необходима арбитражная стратегия
- Так как расчёт цвета точки – наиболее трудоёмкий процесс, рекомендуется, чтобы объекты были отсортированы по удалённости
- Сортировка по удалённости необходима, если объекты используют alpha-смешивание

Трёхмерная сцена



Представление в z-буфере



- Дамп z-буфера может быть использован в постобработке изображения – он предоставляет данные о фактической глубине сцены в каждой точке

Буфер цвета – это структура данных для сохранения цвета каждой точки изображения

- Представлен двумерным массивом**
- Фактически представляет собой визуализацию сцены (результат рендеринга)**
- Точка сохраняется в буфере цвета только если она прошла тест видимости и только тогда, когда полностью вычислен её цвет**