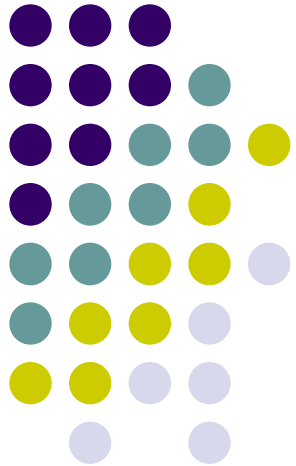
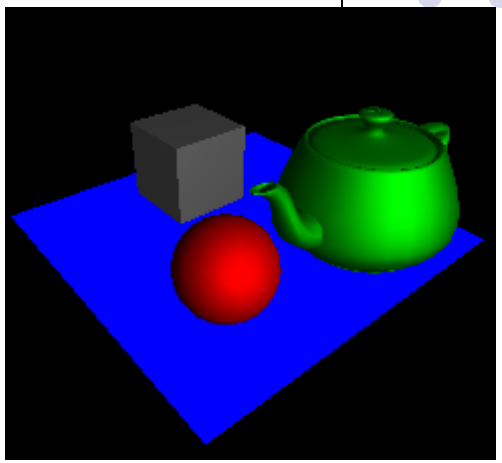
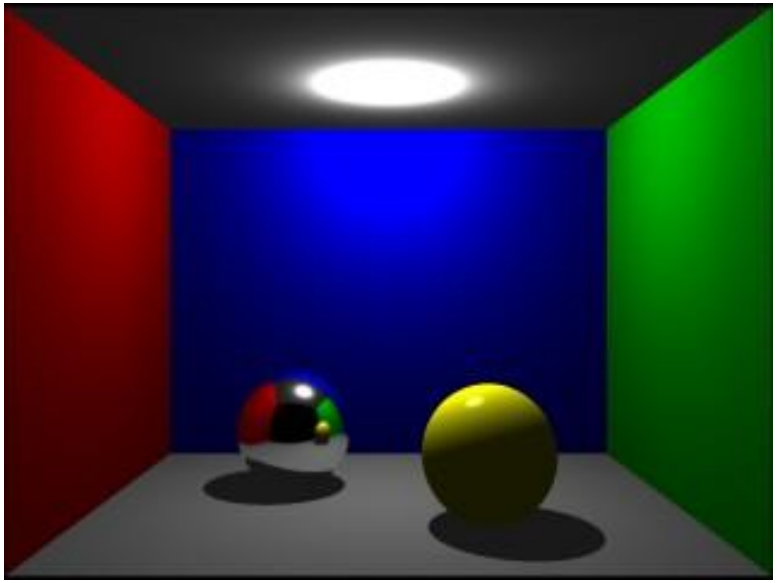
Введение в OpenGL



Фотореализм vs. Скорость

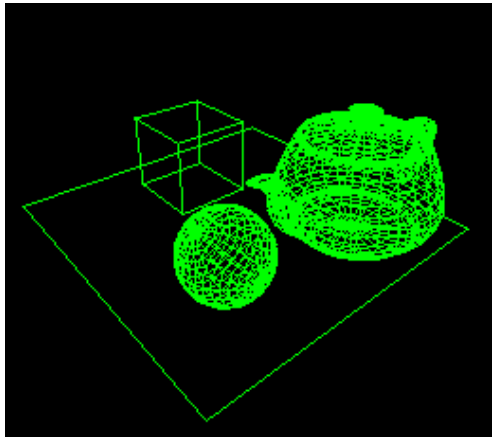
- Фотореализм
  - трассировка лучей
  - излучательность



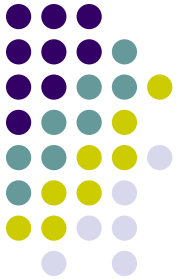


Скорость

- Метод растеризации



Что такое OpenGL?



- 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.

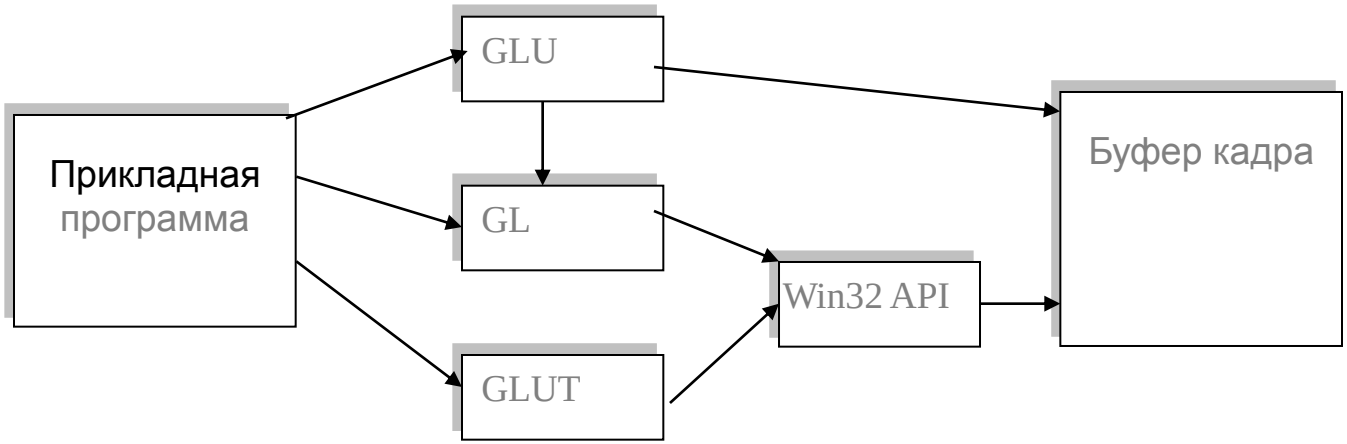
- *Основная функция: интерактивная визуализация трехмерных моделей*


NVIDIA.
© NVIDIA CORPORATION 2002


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

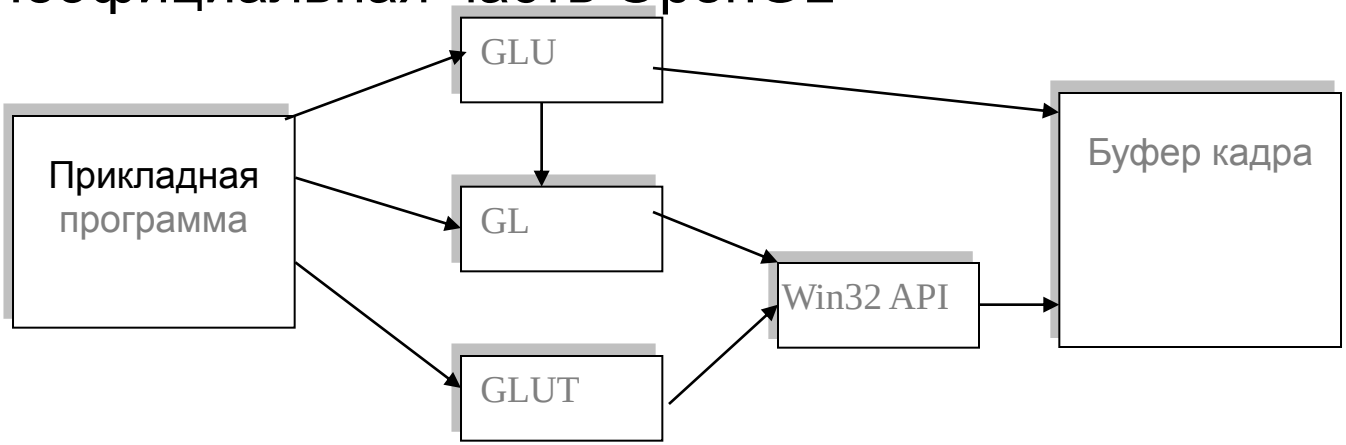
Организация OpenGL

- Состоит из набора библиотек
  - пример для Win32

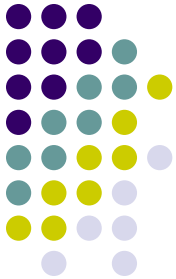


Сопутствующие API

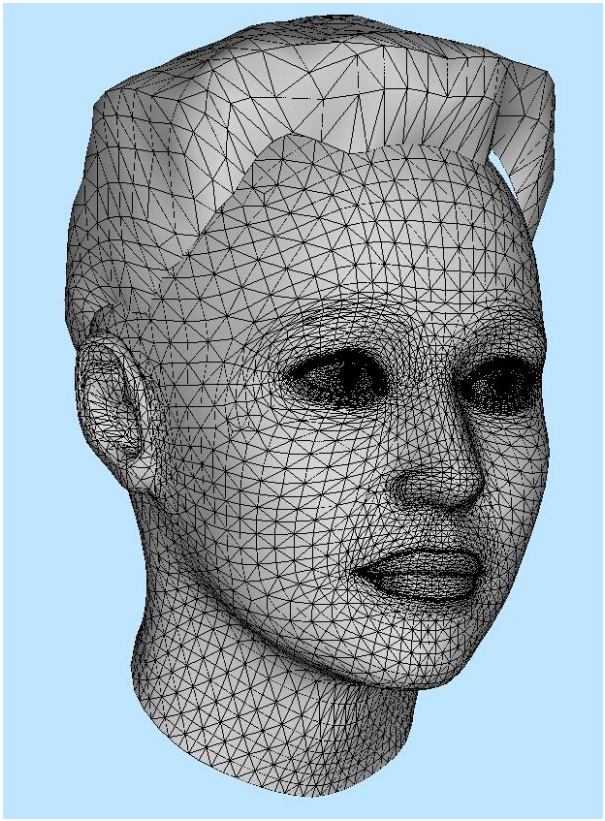
- AGL, GLX, WGL
  - Связь между OpenGL и оконной системой
- GLU (OpenGL Utility Library)
  - Часть OpenGL
  - NURBS, tessellators, quadric shapes, etc
- GLUT (OpenGL Utility Toolkit)
  - Переносимый оконный API
  - Неофициальная часть OpenGL



С какими геометрическими моделями работает OpenGL?



- OpenGL работает с моделями, заданными в граничном полигональном представлении

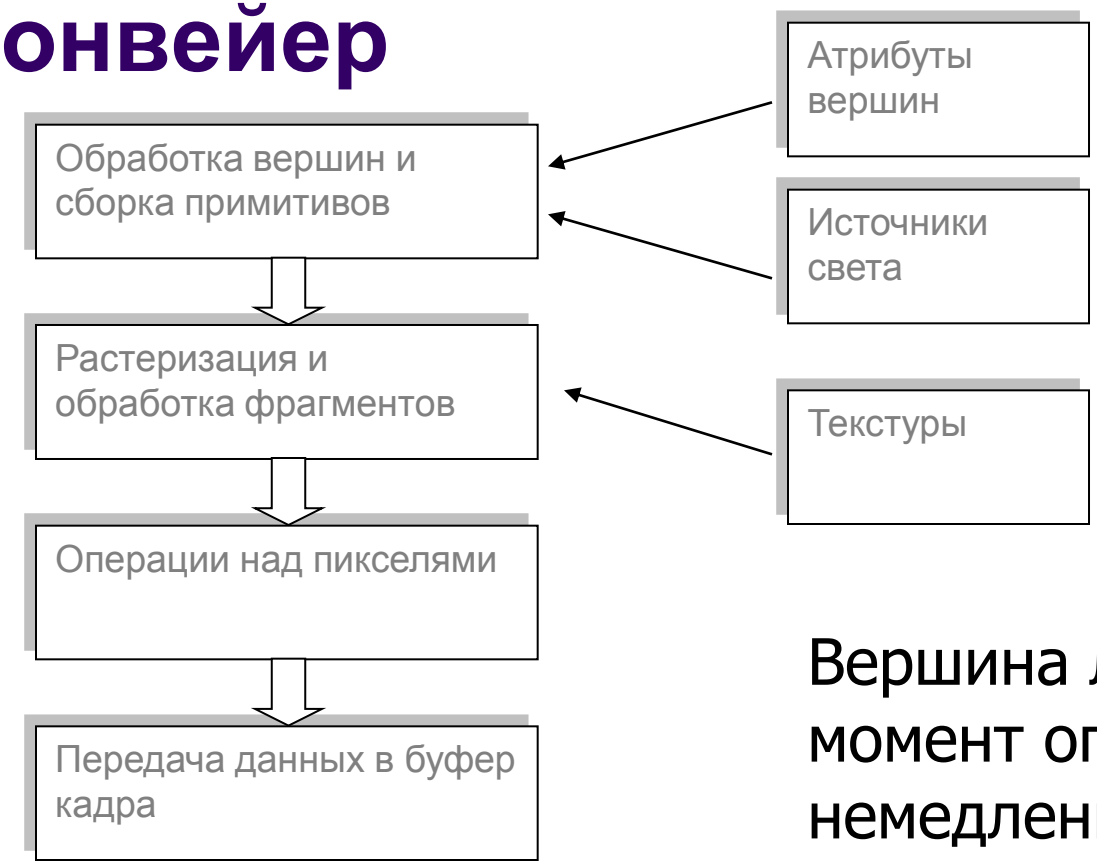


Поверхность приближается набором полигональных граней (face, polygon)

Границы граней описываются ребрами (egde)

Часть отрезка, формирующего ребро, заканчивается вершинами (vertex)

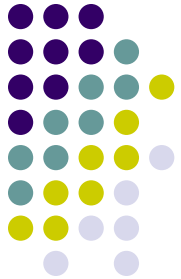
Конвейер



Вершина любого объекта в момент определения немедленно передается в конвейер, и проходит все его ступени

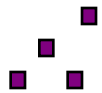
3D координаты -> экранные

Как рисовать объекты с помощью OpenGL?



- Объекты на экране рисуются путем последовательной передачи в конвейер вершин примитивов, которые составляют объект
  - команды передача данных
- Обработка данных на каждом этапе конвейера может быть настроена через
  - команды изменения состояния

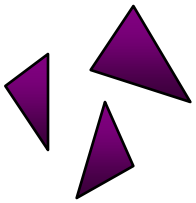
Типы примитивов OpenGL



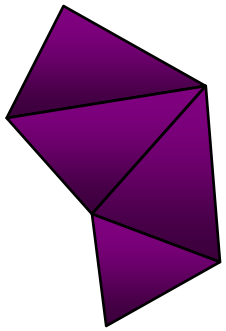
GL_POINTS

GL_LINES

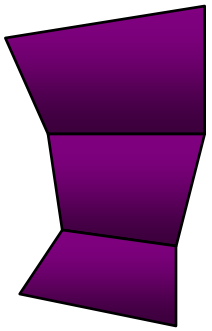
GL_LINE_STRIP



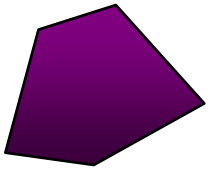
GL_TRIANGLES



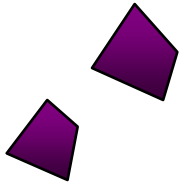
GL_TRIANGLE_STRIP



GL_QUAD_STRIPE

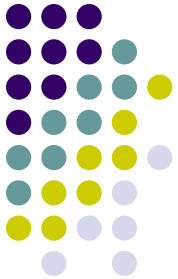


GL_POLIGON



GL_QUADS

Атрибуты вершин

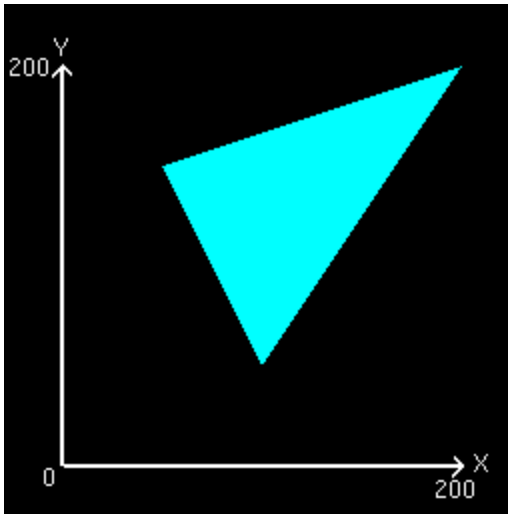


- Каждая вершина кроме *положения в пространстве* может иметь несколько других атрибутов
  - Материал
  - Цвет
  - Нормаль
  - Текстурные координаты
- *Внимание: всегда используется ТЕКУЩИЙ набор атрибутов*
  - *OpenGL – конечный автомат*

Пример кода

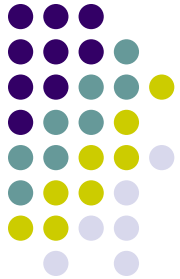
- Цветной треугольник

```
glBegin(GL_TRIANGLES);
    glColor2f(0.0f,1.0f);
    glVertex2f(150.0f, 50 .0f);
    glVertex2f(50.0f, 150 .0f);
    glVertex2f(200 .0f, 200 .0f);
glEnd();
```



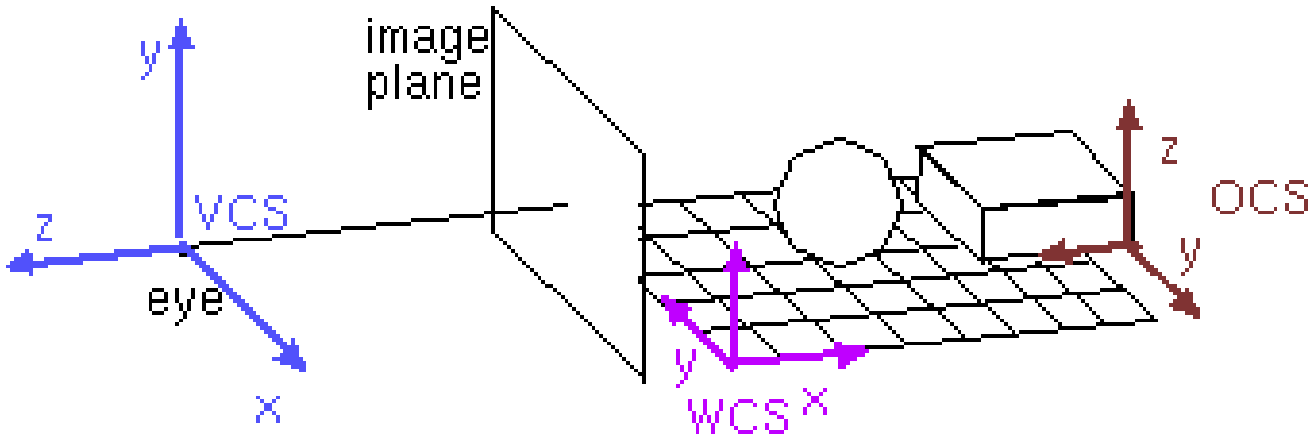
- Таким образом можно задать любой объект! Теперь задача в том, чтобы показать этот объект на экране

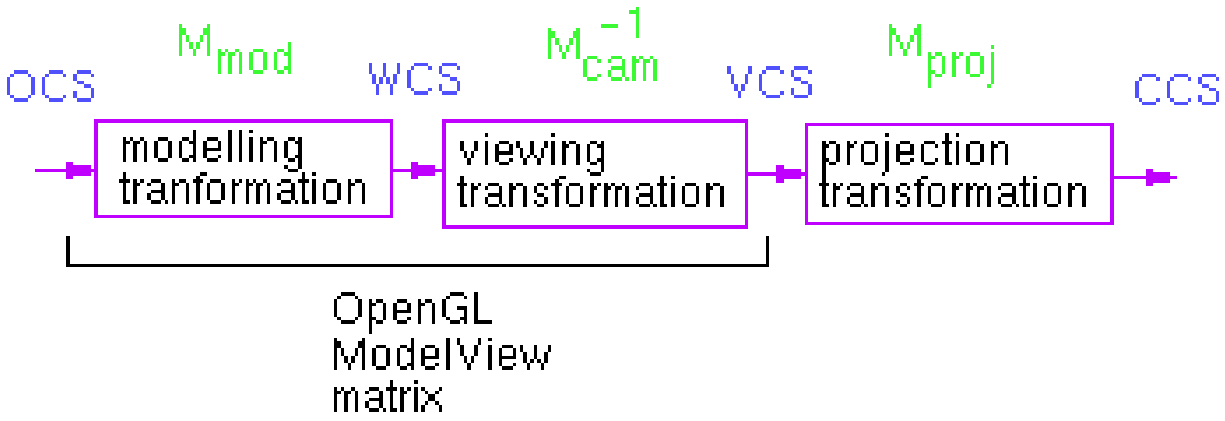
Преобразования координат в OpenGL



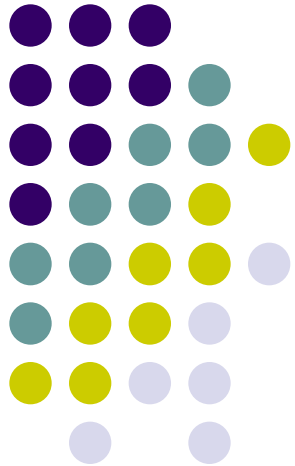
- Каждая вершина объекта задается в *локальных координатах модели*
- Необходимо определить набор *геометрических преобразований,* таких, что каждая вершина преобразуется в точку на плоскости экрана
- Три последовательных преобразования:
  - модельное преобразование
  - видовое преобразование
  - проективное преобразование

Графический конвейер



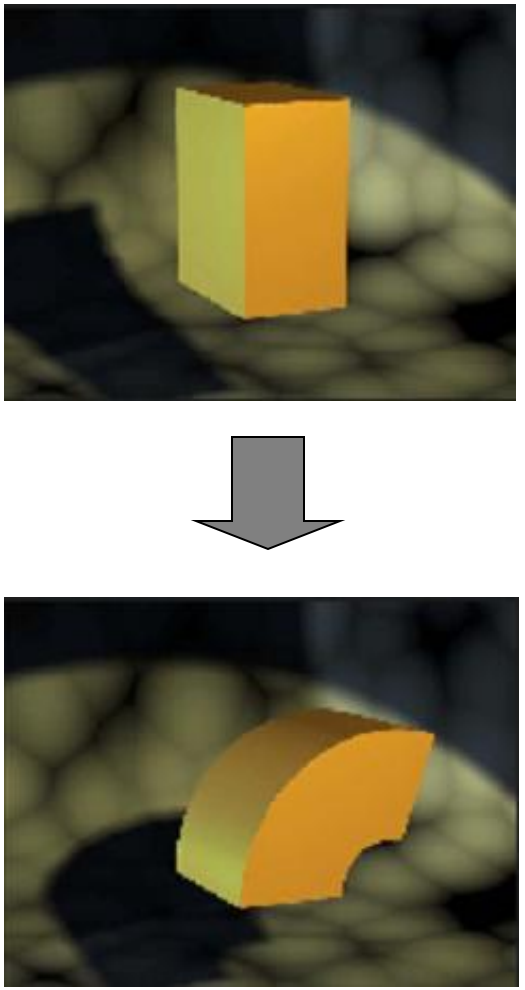


Геометрические преобразования

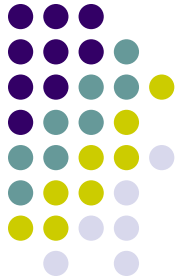


Что такое геометрические преобразования?

- Модель
  - Например, описание поверхности трехмерного объекта
  - Некоторое подмножество точек декартова пространства
- Зачем применять преобразования к модели?
  - Создание моделей (сцен) из компонент
    - Вспомните CSG
  - Редактирование моделей
  - Преобразования в процессе синтеза изображений
    - Получение проекции на 2D экран



Типы преобразований

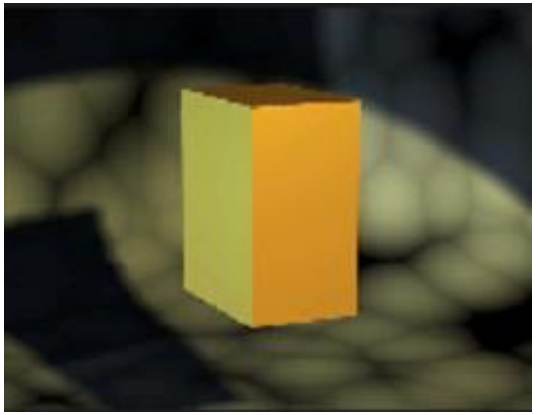


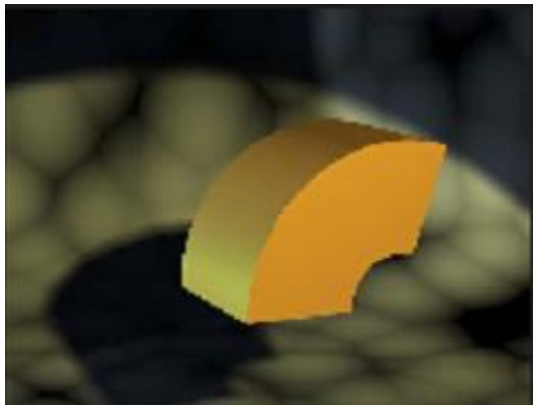
- Будем рассматривать два класса:
  - Нелинейные преобразования
  - Линейные преобразования
    - Важный класс!

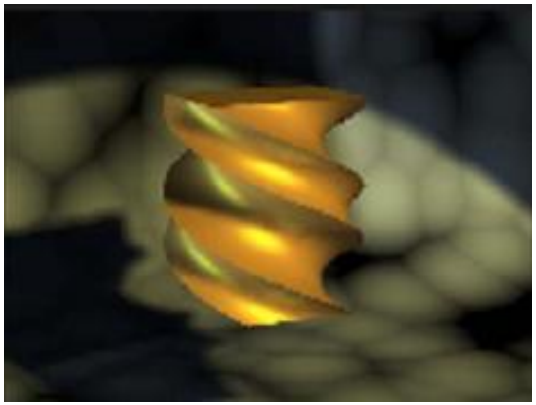
Нелинейные преобразования

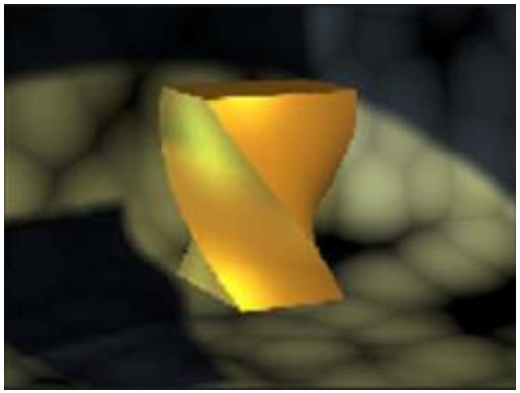
- Произвольное преобразование точек модели

$$M' = T(M)$$









Линейные преобразования

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

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

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

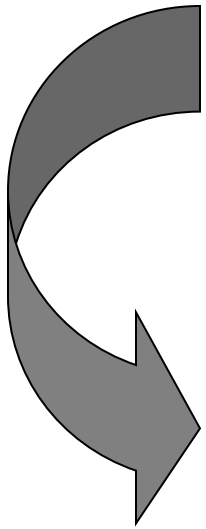
- Линейное преобразование применяется к каждой точке (вершине) модели.
- Не изменяет топологию!

Преобразование в матричной форме

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

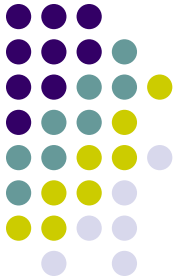
$$y' = Ex + Fy + Gz + H$$

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



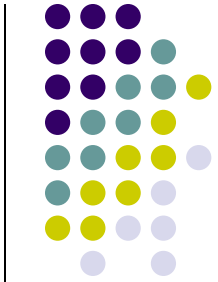
$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Однородные координаты



- Какой смысл имеет использование 4х компонент вектора?
  - Позволяет использовать матричную запись для всех линейных преобразований (если использовать матрицы 3х3, невозможно представить перенос)
  - Позволяет описать так называемой перспективное деление

Типичные линейные преобразований



- Общие линейные преобразования
  - w != 1 (после преобразования)
  - Также называются проективными
  - Прямые переходят в прямые
- Аффинные преобразования
  - w = 1
  - Сохраняется параллельность линией
  - Пример: сдвиг
- Преобразование подобия
  - Сохраняются углы
  - Пример: равномерное масштабирование
- Изометрия
  - Сохраняются расстояния
  - Пример: поворот, перенос

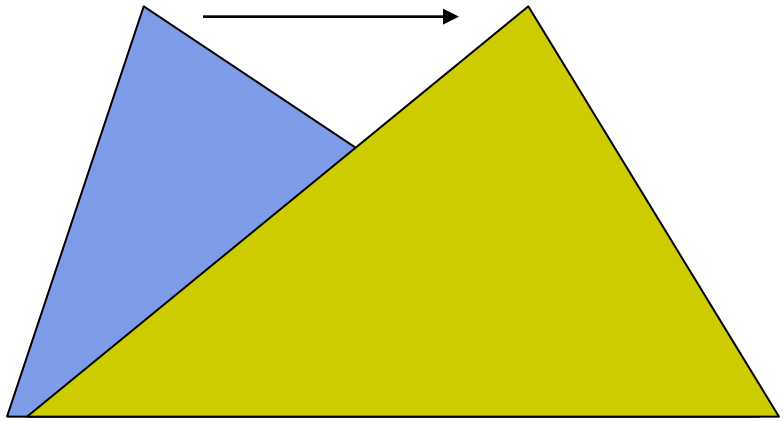
Сдвиг

$$x' = x + ay$$

$$y' = y + bx$$

Аффинное преобразование

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



Масштабирование

$$x' = ax$$

$$y' = by$$

$$z' = cz$$

Преобразование подобия

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

Параллельный перенос

$$x' = x + \Delta x$$

$$y' = y + \Delta y$$

$$z' = z + \Delta z$$

Изометрия

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

Поворот (2D)

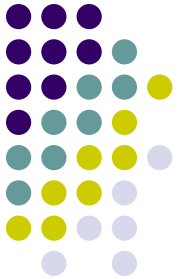
$$x' = x \cos \theta - y \sin \theta$$

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

Изометрия

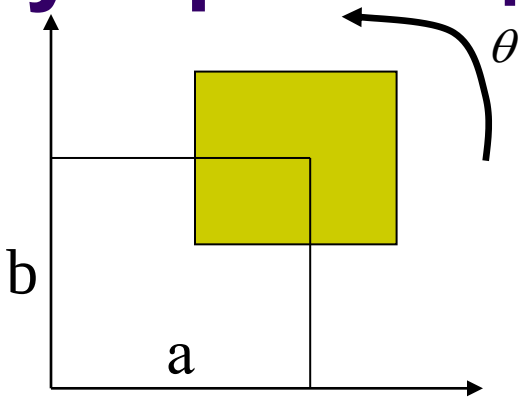
$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Наши обозначения



- Перенос **T(a,b,c)**
- Поворот **R(theta)** или R(axis, theta)
- Масштабирование **S(a,b,c)**
- Сдвиг **Sh(a,b,c)**

Суперпозиция преобразований



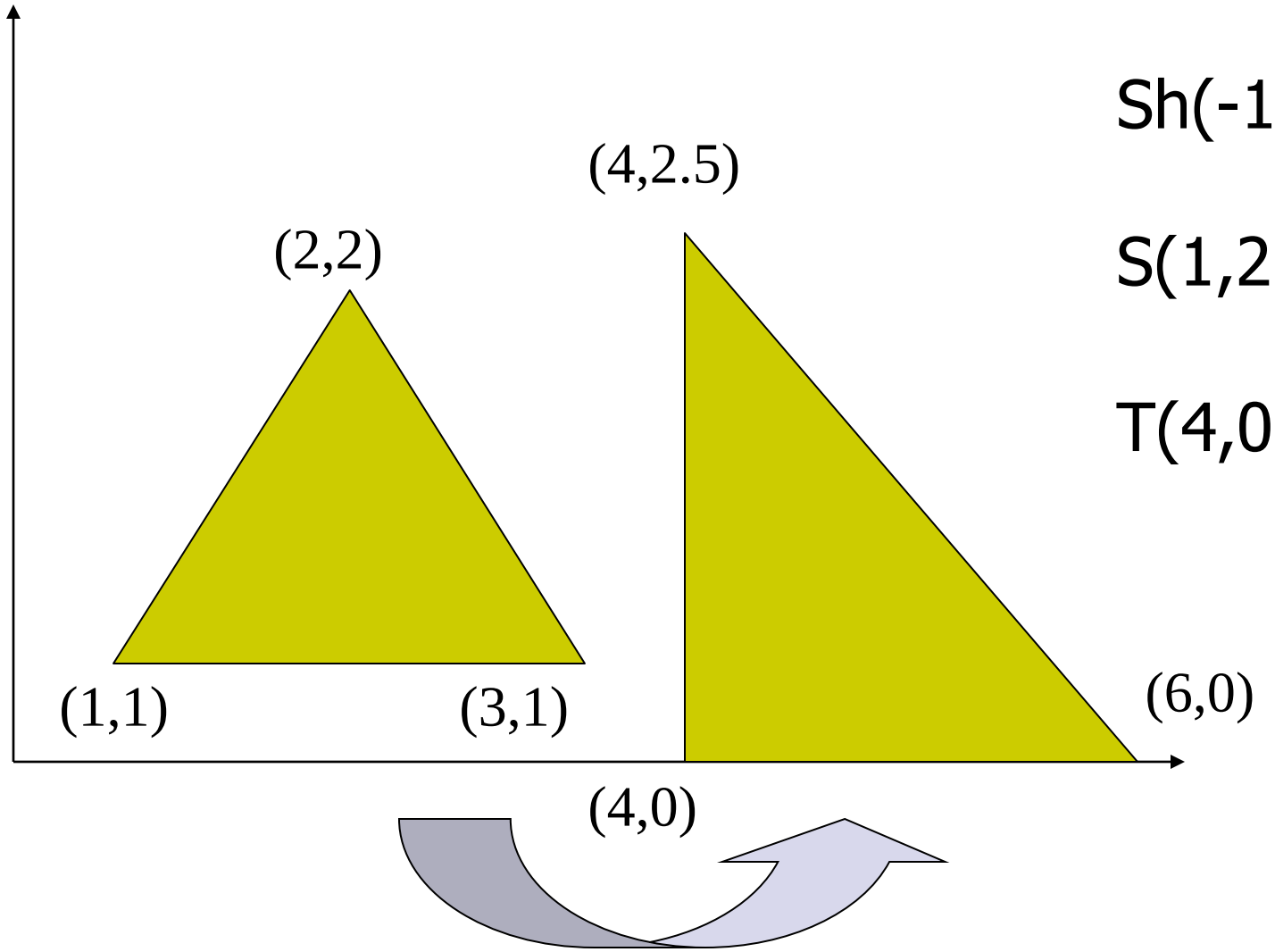
Повернуть вокруг центра

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



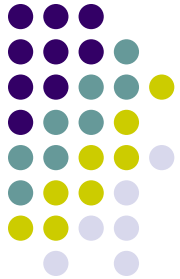
T(-1,-1)

Sh(-1, 0)

S(1,2.5)

T(4,0)

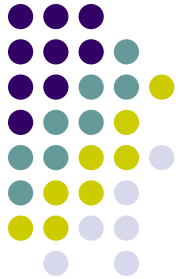
Решение задачи



- T(-1,-1)
- Sh(-1, 0)
- S(1,2.5)
- T(4,0)

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

Проективные преобразования

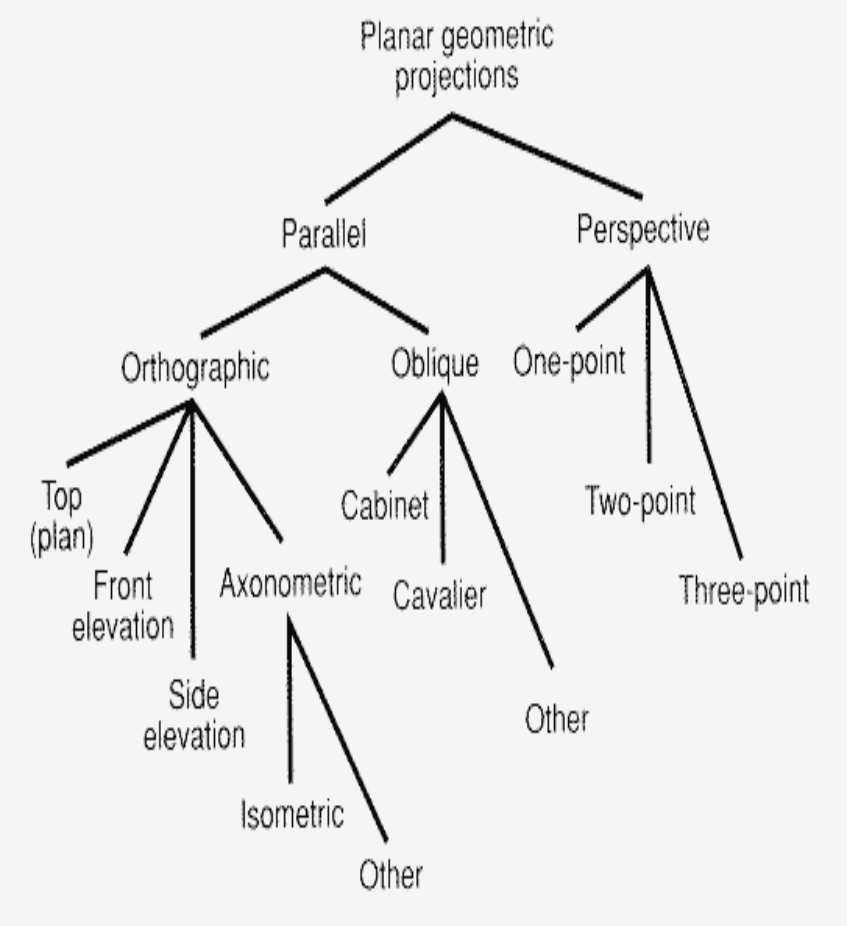


- Важнейший класс преобразований
- Все современные дисплеи визуализируют **изображение** => необходимо преобразовать 3D данные в 2D !

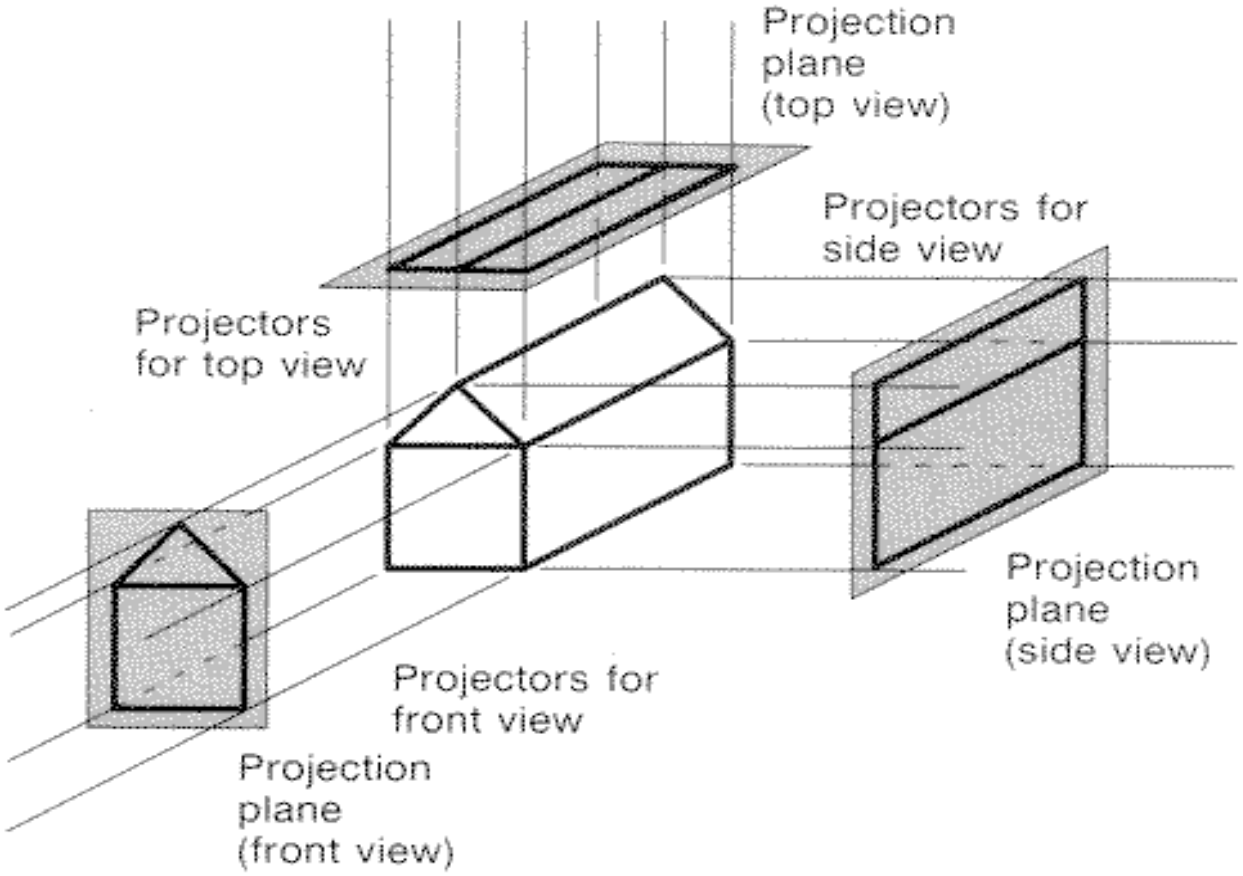
- Для выполнения таких преобразований применяются **проективные преобразования**.
- Описываются матрицей 4x4 (линейным преобразованием)

Типы проекций

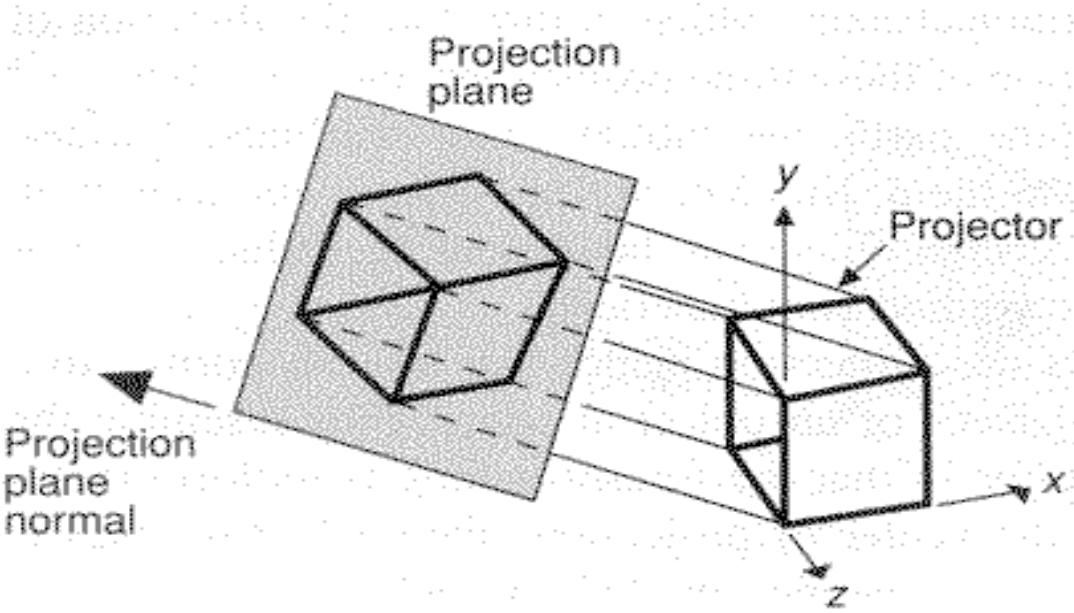
- Много разновидностей
  - Применяются в дизайне и т.п.
- Основные виды
  - Параллельные
    - Ортографические
    - Косоугольные
  - Перспективные
    - 1,2,3-х точечные

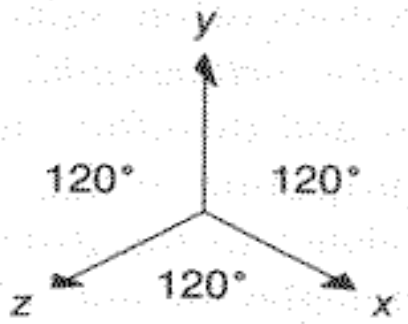


Ортографическая проекция

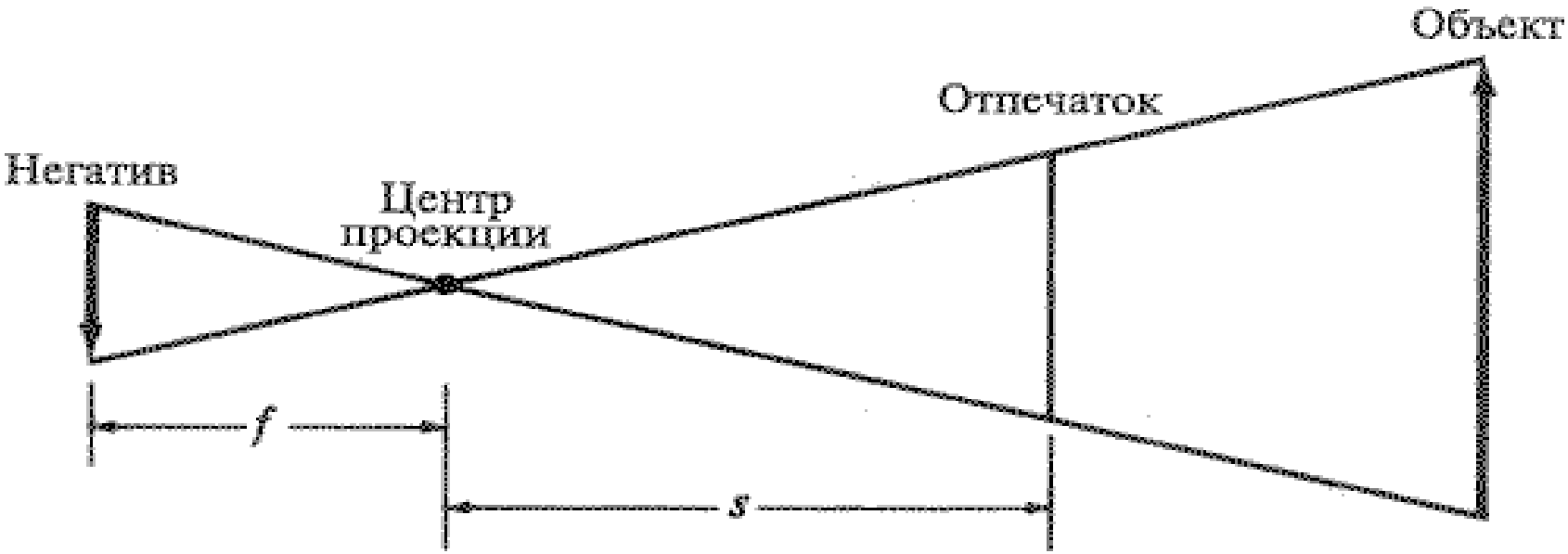


Изометрия

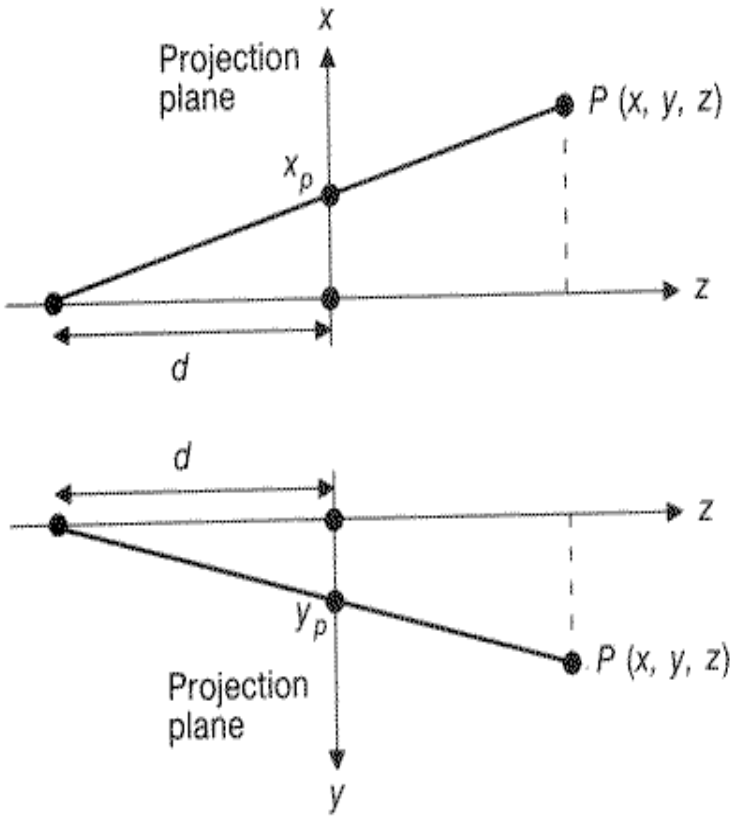

Projection
plane
y
Projector
Projection
plane
normal
x
z


y
120°
120°
120°
z
x

Фотография



Перспективная проекция

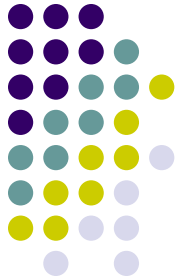


$$\frac{x_p}{d} = \frac{x}{z + d}, \frac{y_p}{d} = \frac{y}{z + d},$$

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

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

Перспективная проекция : запись в матричном виде

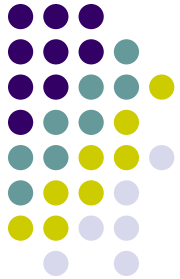


$$x_p = \frac{x}{(z/d) + 1}$$

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

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

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)

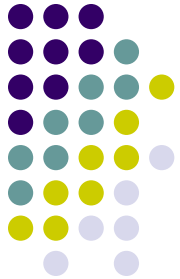
- Четверная компонента не равна 1 !
  - Результат уже не в декартовых координатах.
- Однородные координаты!

$$x = x / w$$

$$y = y / w$$

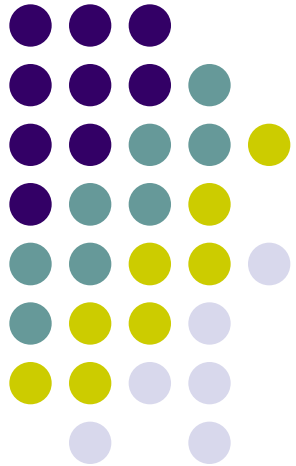
$$z = z / w$$

Подведем итоги

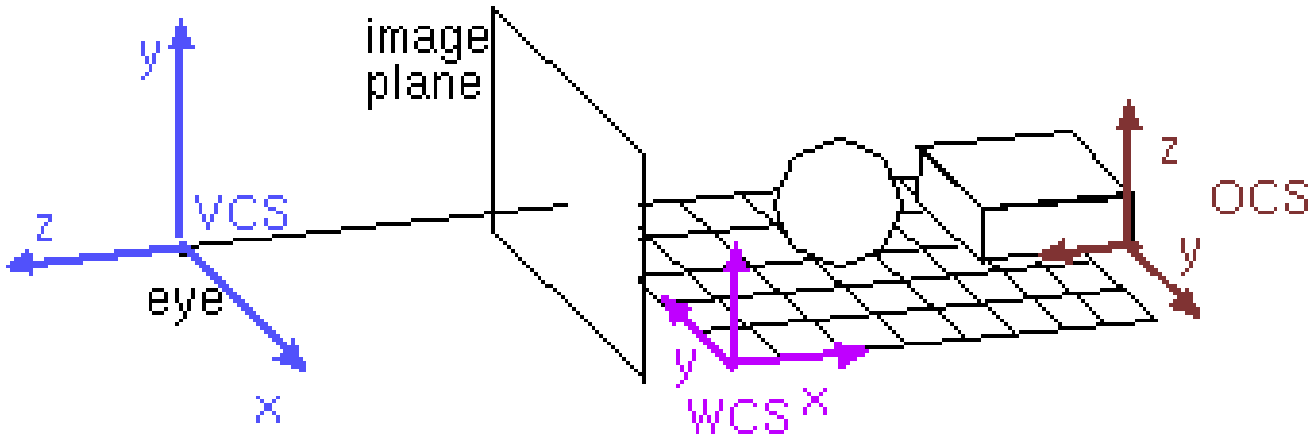


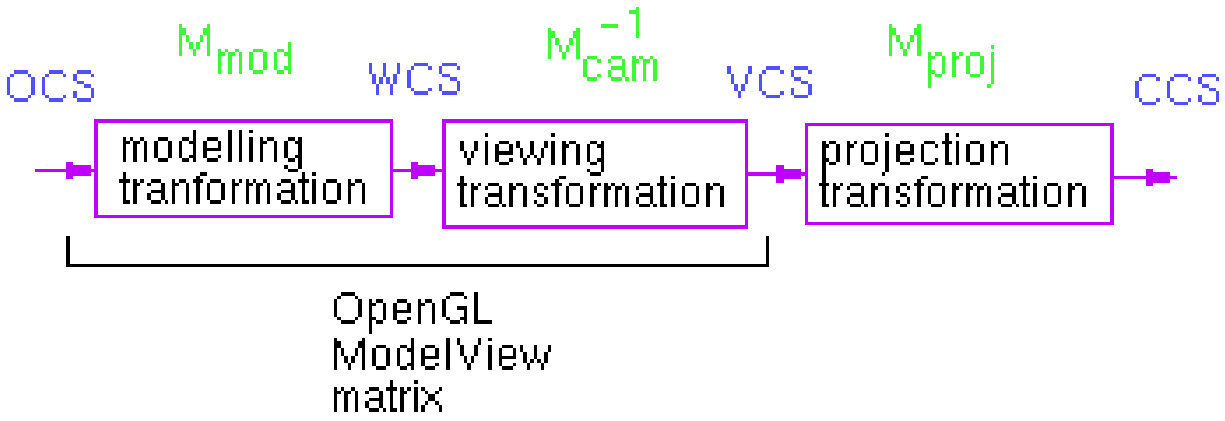
- В графическом конвейере OpenGL используются линейные и проективные геометрические преобразования
- Преобразования описываются матрицами 4x4
- Операции производятся над векторами в однородных координатах

Геометрические преобразования в OpenGL



Графический конвейер

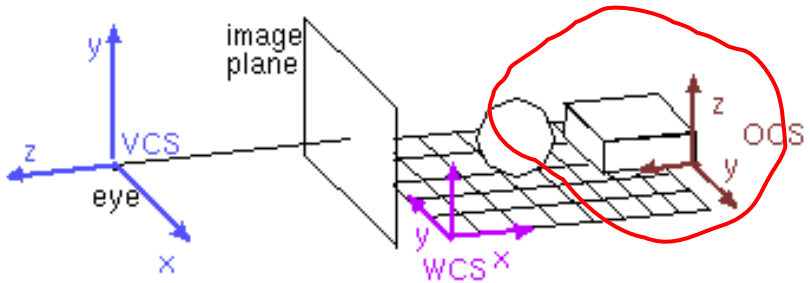


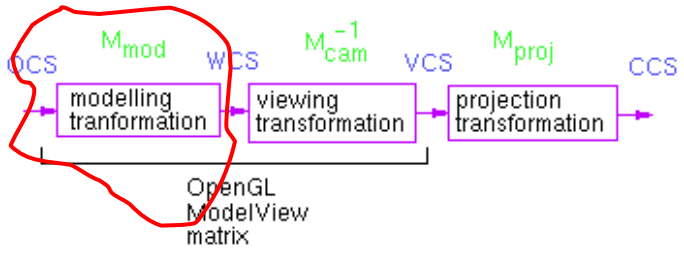


Модельное преобразование

- Переводит модель, заданную в локальных (собственных) координатах, в глобальное (мировое пространство)
- Модель «собирается» из частей, с помощью модельных преобразований (обычно композиция переносов, поворотов, масштабирования)

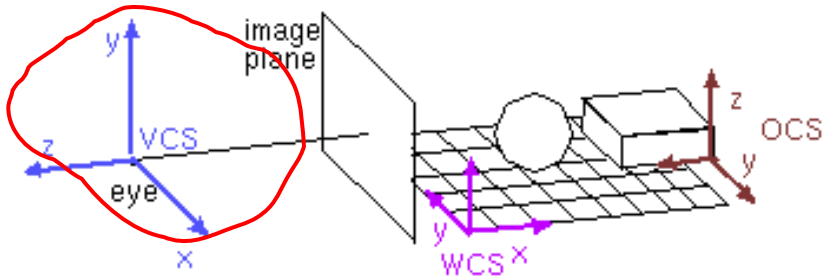
- На выходе – модель в единых мировых координатах

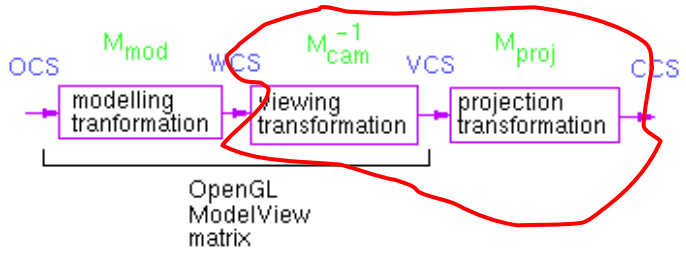




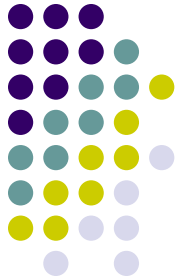
Виртуальная камера

- 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

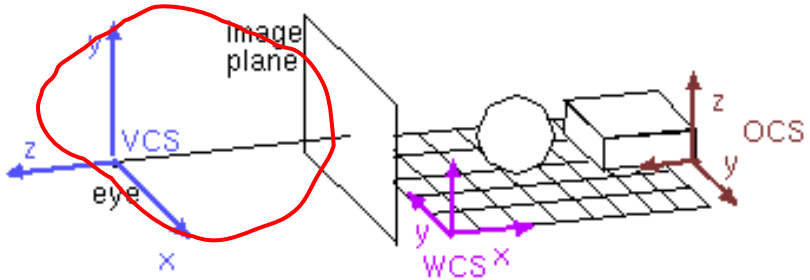


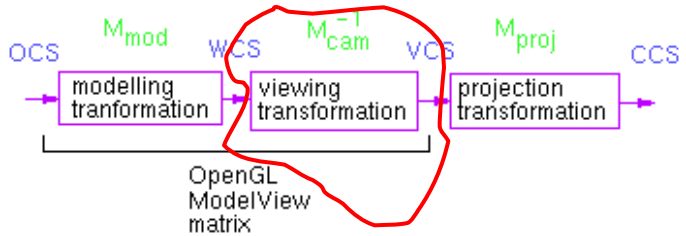
- Для чего нужно еще одно преобразование?

- 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

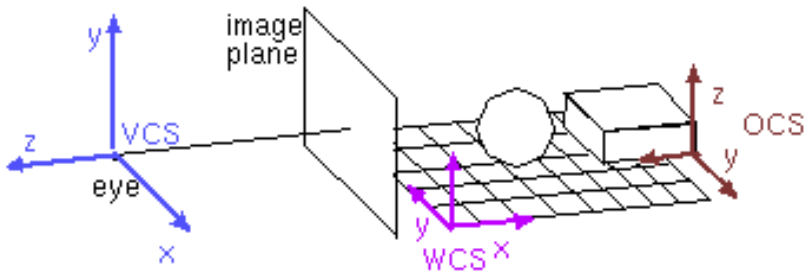
- «Подгоняет» мир под стандартную камеру, преобразует мировую систему координат в видовые координаты (которые подходят для «стандартной» камеры
- На выходе – модель, готовая к проекции на экран

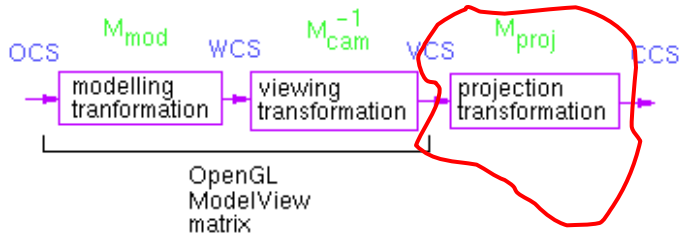




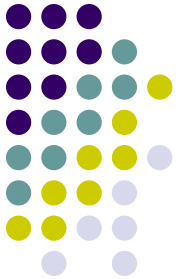
Проективное преобразование

- Выполняет 3D преобразование, подготавливая модель к переходу на 2D
- После перспективного преобразования необходимо отбросить координату z и получить значения в оконных координатах (обычно от –1 до 1)

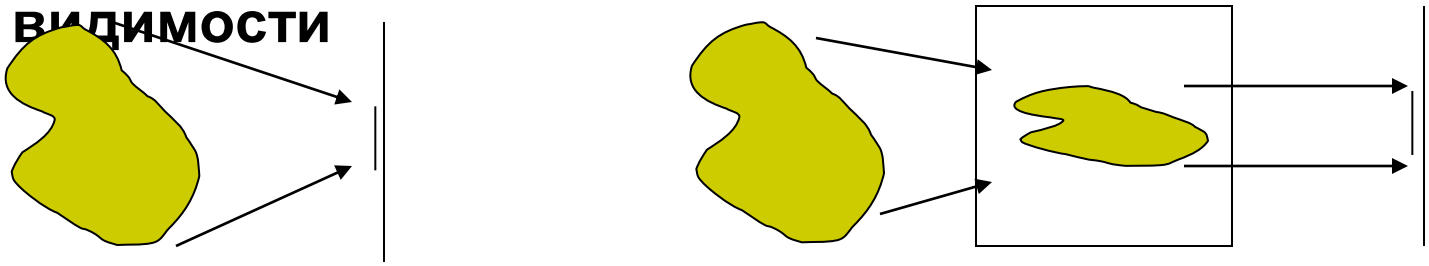




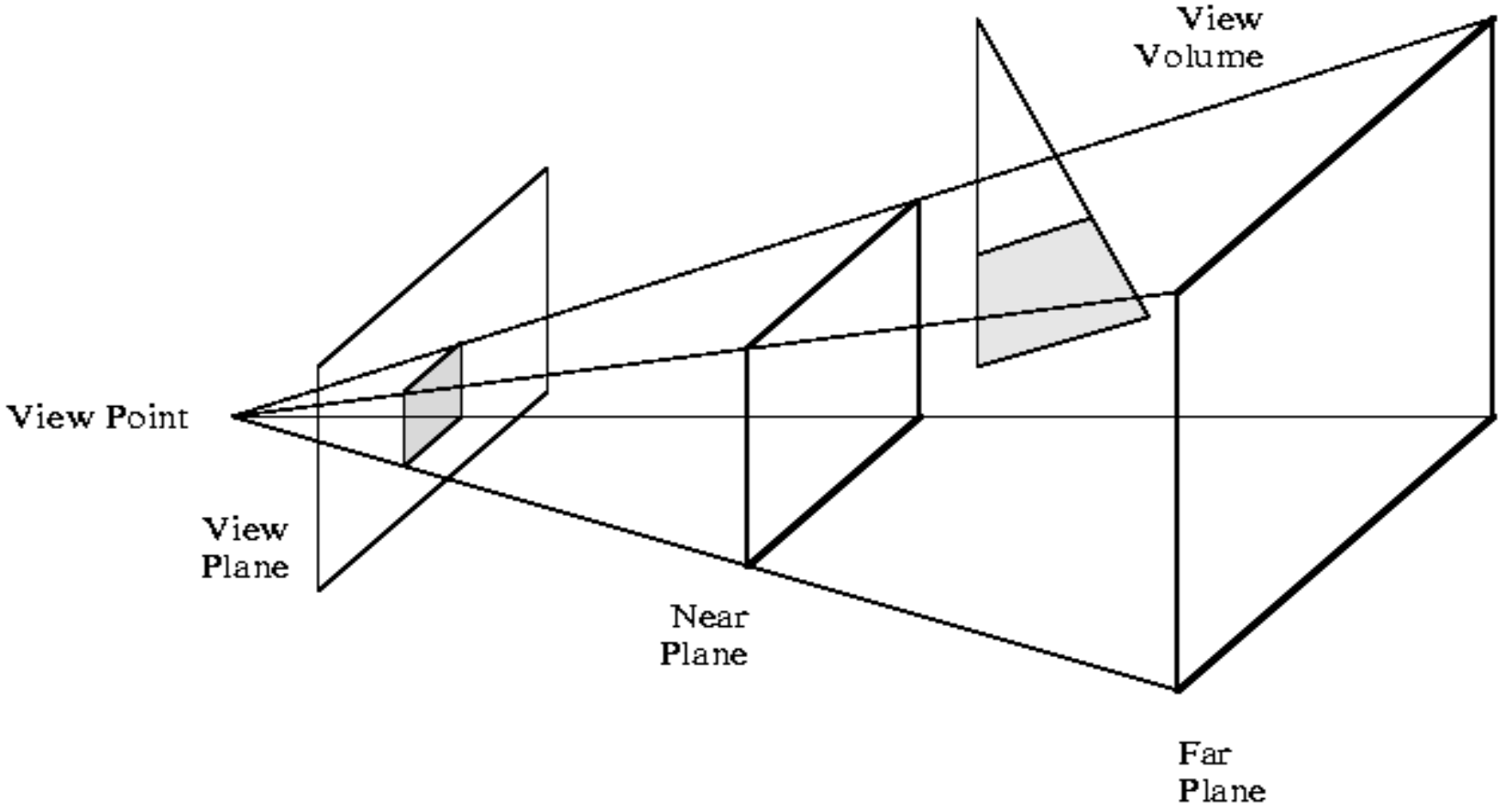
Проективное преобразование vs. проекция



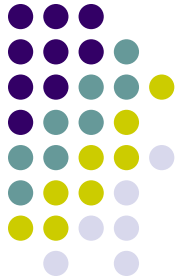
- Матрица проекции вырожденная
  - Фактически, информация от координате z теряется
- Часто необходимо выполнять дополнительные действия уже ПОСЛЕ проецирования
  - Например, удаление невидимых линий/поверхностей
  - Поэтому часто (e.g. в OpenGL) используется перспективное преобразование вместо проекции
    - Перспективное преобразование невырожденно и позволяет анализировать глубину!
- Перспективное преобразование преобразует некоторую область пространства в **каноническую пирамиду видимости**



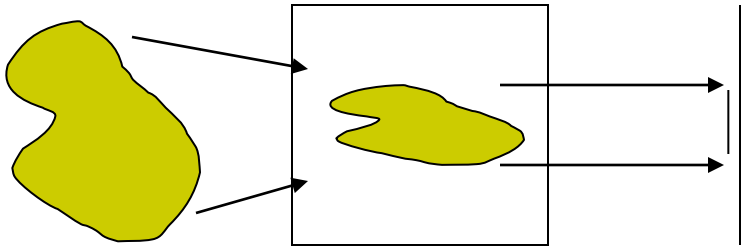
Отсечение



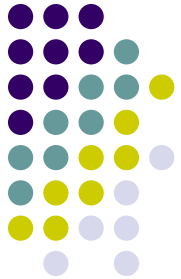
Преобразование в экранные координаты



- Отбрасываем координату z
- Умножаем на высоту/ширину окна
  - Получаем экранные координаты



Модельно-видовое преобразование

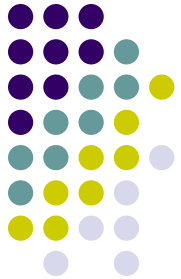


- OpenGL не имеет раздельных матриц для видового и модельного преобразования

- Поэтому нужно задавать сразу произведение:

$$M = M_{view} \cdot M_{mdl}$$

Матрицы преобразований



- Выбираем матрицу преобразований для изменения:

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

- Две основные операции над матрицами:

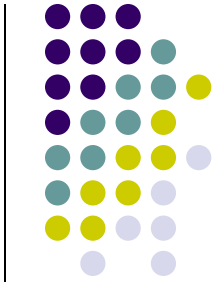
```
void glLoadIdentity();
```

$$M = E$$

```
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)



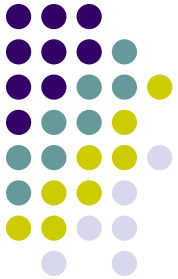
```
void glTranslated(GLdouble x,
                  GLdouble y,
                  GLdouble z);

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

- Настройка виртуальной камеры

$$\text{gluLookAt}(\ eye_x, eye_y, eye_z,\ aim_x, aim_y, aim_z,\ up_x, up_y, up_z)$$

- eye – координаты наблюдателя
- aim – координаты “цели”
- up – направление вверх



Модельно-видовое преобразование (2)

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

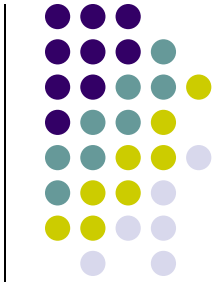
Виртуальная камера

Модельное преобразование

Геометрия

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

Проективное преобразование

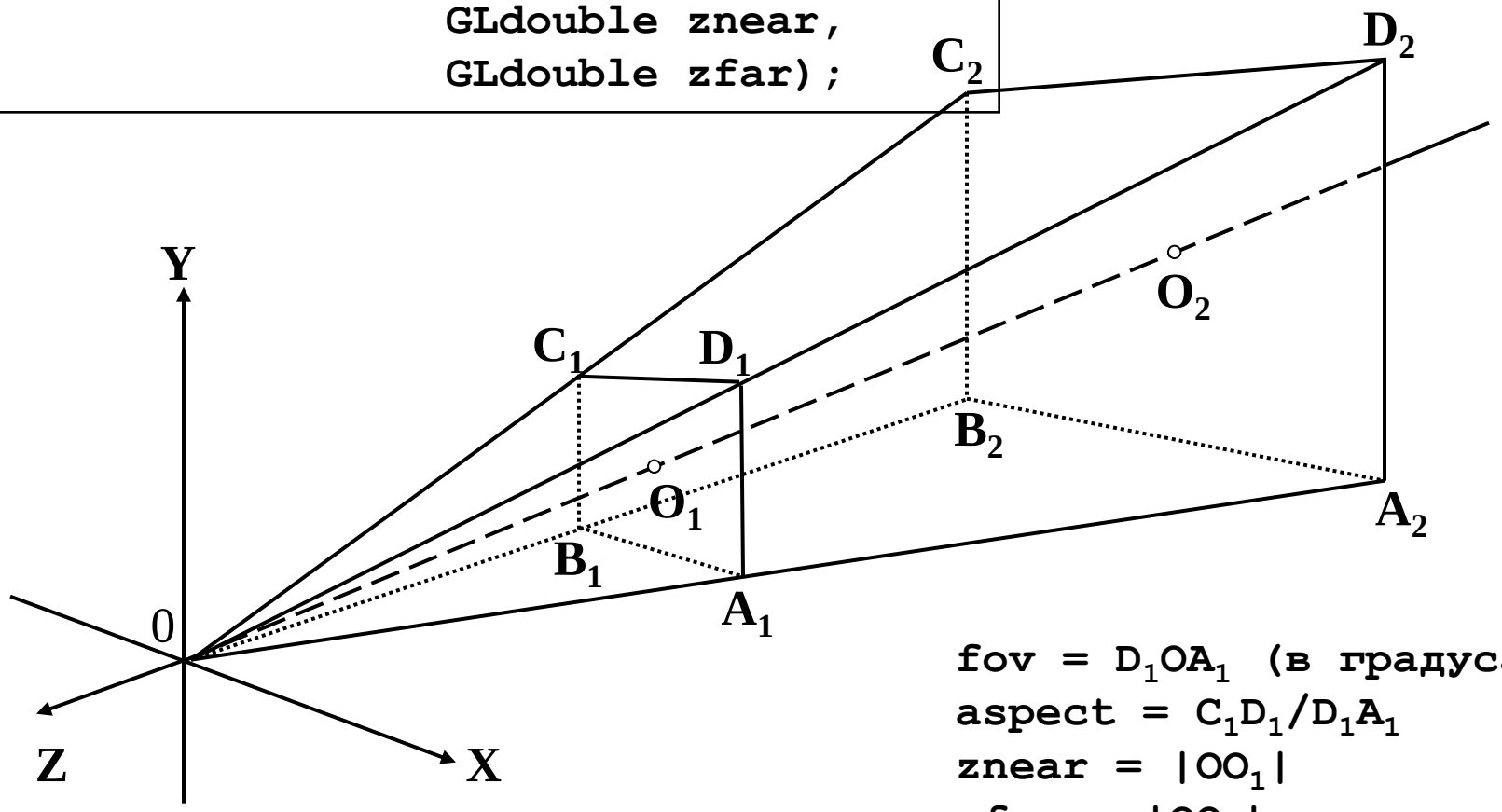


- glMatrixModel(GL_PROJECTION);

- gluPerspective(…)

Как работает gluPerspective

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



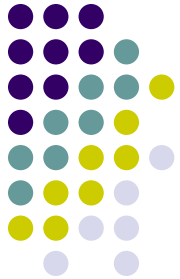
fov = D_1OA_1 (в градусах)

aspect = C_1D_1/D_1A_1

znear = $|OO_1|$

zfar = $|OO_2|$

Итоги



- 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