

# Алгоритмические основы мультимедийных технологий

## *Лекция 2*

Рябинин Константин Валентинович

e-mail: [icosaeder@ya.ru](mailto:icosaeder@ya.ru)

jabber: [icosaeder@jabber.ru](mailto:icosaeder@jabber.ru)

Пермь, 2011

## Атрибуты вершин

- Координаты

`void glVertex3f(GLfloat x, GLfloat y, GLfloat z)`

- Координаты нормали

`void glNormal3f(GLfloat nx, GLfloat ny, GLfloat nz)`

- Координаты текстуры

`void glTexCoord2f(GLfloat u, GLfloat v)`

- Цвет

`void glColor3ub(GLubyte r, GLubyte g, GLubyte b)`

- ...

## Вывод множества вершин

```
glBegin(GL_QUADS);  
    glColor3ub(255, 0, 0);  
    glVertex2d(-0.5, -0.5);  
    glColor3ub(0, 255, 0);  
    glVertex2d(0.5, -0.5);  
    glColor3ub(0, 0, 255);  
    glVertex2d(0.5, 0.5);  
    glColor3ub(255, 255, 0);  
    glVertex2d(-0.5, 0.5);  
glEnd();
```

## Типы цепочек вершин

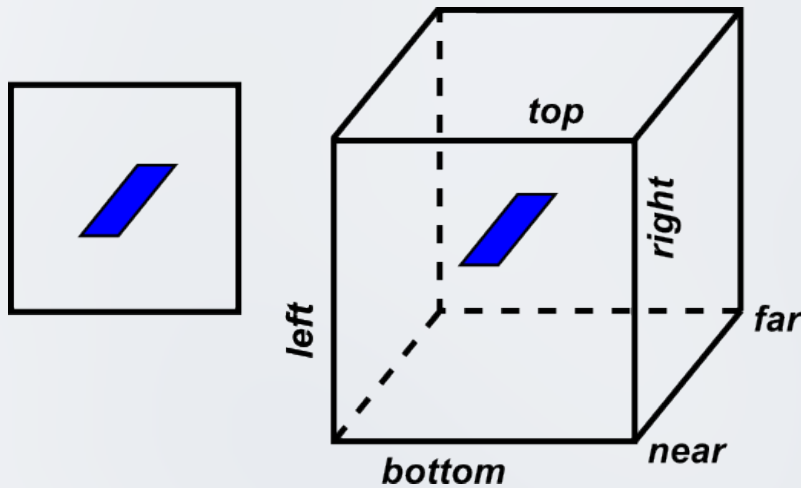
- **GL\_POINTS** – вывод вершин точками
- **GL\_LINES** – соединение каждой следующей пары вершин линией
- **GL\_TRIANGLES** – соединение каждой тройки вершин в треугольник
- **GL\_QUADS** – соединение каждой четвёрки вершин в четырёхугольник
- **GL\_POLYGON** – соединение всей вершин в многоугольник
- ...

## Способы отображения многоугольников

- Закрашенные полигоны  
`glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);`
- Проволочный каркас  
`glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);`
- Точки вершин  
`glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);`

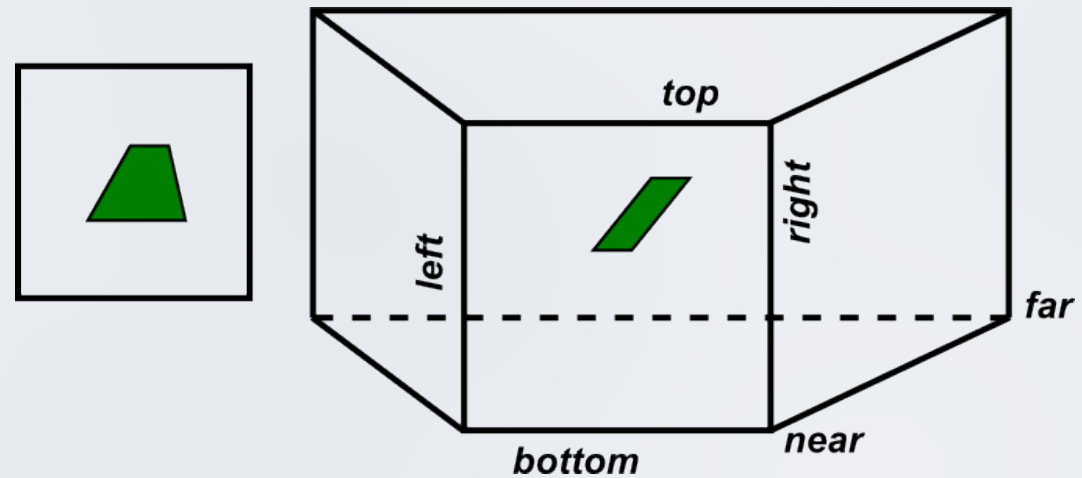
→ Заданием параметров проекции определяется видимая область пространства

Параллельная проекция



`glOrtho`(GLdouble **left**,  
GLdouble **right**,  
GLdouble **bottom**,  
GLdouble **top**,  
GLdouble **near**,  
GLdouble **far**)

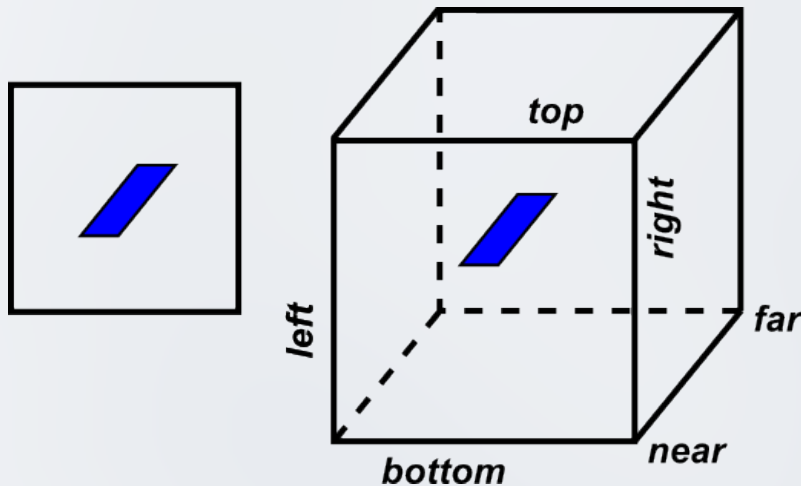
Перспективная проекция



`glFrustum`(GLdouble **left**,  
GLdouble **right**,  
GLdouble **bottom**,  
GLdouble **top**,  
GLdouble **near**,  
GLdouble **far**)

→ Заданием параметров проекции определяется видимая область пространства

Параллельная проекция

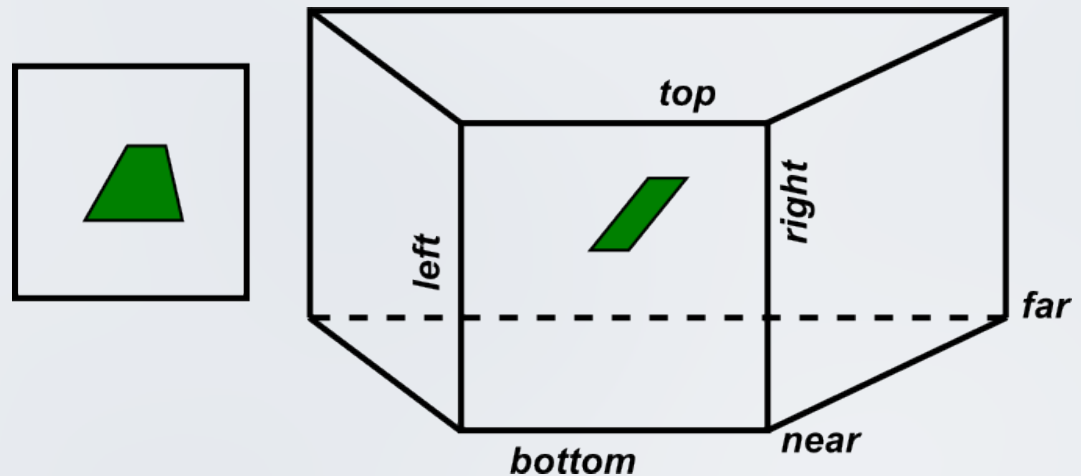


```
gluOrtho2D(GLdouble left,  
            GLdouble right,  
            GLdouble bottom,  
            GLdouble top)
```

~

```
glOrtho(left, right, bottom, top, -1, 1)
```

Перспективная проекция



```
gluPerspective(GLdouble fov,  
               GLdouble aspect,  
               GLdouble near,  
               GLdouble far)
```

~

```
h = tan((fov / 2) / 180 * M_PI) * near  
w = h * aspect  
glFrustum(-w, w, -h, h, near, far)
```

Камера – это псевдообъект в трёхмерном пространстве, характеризующий положение наблюдателя

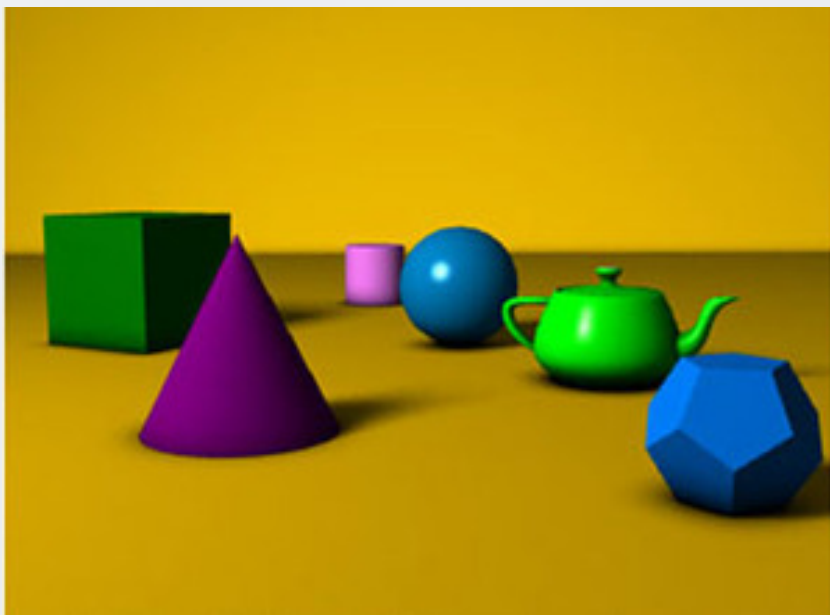
- Далеко не во всех системах камера в явном виде имеет место
- Человеку удобно работать с камерой, поэтому в системах, не предполагающих её наличия, она вводится в качестве метафоры
- В OpenGL камера отсутствует, вместо неё – унифицированный механизм матричных преобразований (MODELVIEW)
- Функция-обёртка, моделирующая камеру:  
`gluLookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ,  
          GLdouble centerX, GLdouble centerY, GLdouble centerZ,  
          GLdouble upX, GLdouble upY, GLdouble upZ)`



**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**

- Наиболее естественный способ – использование событийно-ориентированных систем
- Имитацию событийной ориентированности предоставляет GLUT при помощи регистрации функций обратного вызова на нажатие клавиши (`glutKeyboardFunc`), перемещение мыши (`glutMotionFunc`) и клик мыши (`glutMouseFunc`)
- При отсутствии событийно-ориентированных средств следует осуществлять последовательный опрос устройств ввода

Анимация – **это вдыхание жизни в сцену** :)

Анимация – это последовательное отображение  
(конструктивно) кадров с различным содержанием

Программно-аппаратная поддержка анимации:  
**механизм двойной буферизации**

→ Строго говоря, любое свойство объекта  
может быть анимировано



Важное требование к анимации:  
**сохранение межкадрового соответствия**  
(*frame-to-frame coherence*)

## Анимация по таймеру:

- **Идея: запуск некоторого таймера (должен поддерживаться системой), который в наперёд заданные моменты времени вызывает функцию обновления состояния**
- **Способ имеет право на существование лишь в исключительных случаях**
- **Таймер лучше использовать как триггер анимации, а не как её «драйвер»**

## Непрерывная анимация:

- **Идея: обновление состояния происходит с максимально возможной скоростью, а величина изменения вычисляется на основе желаемой и фактической скорости**

## Алгоритм последовательного опроса устройств, обеспечивающий независимость от производительности

```
const int MAX_SKIP = 1, SKIP_TICKS = 25, TIME_PRECISION_FACTOR = 0;
const float TIME_FACTOR = 1000;
int controlLoops = 0, loops = 0;
int long long nextTick, startTime, endTime;
float delta = 1;
int deltaIsValid = 0;

nextTick = startTime = currentTime();

while (shouldContinue())
{
    controlLoops = 0;
    while (currentTime() > nextTick && controlLoops < MAX_SKIP)
    {
        obtainInformationFromControlDevices();

        nextTick += SKIP_TICKS;
        ++controlLoops;
    }

    changeScene(delta, deltaIsValid);
    renderScene();

    if (loops++ > TIME_PRECISION_FACTOR)
    {
        deltaIsValid = 1;
        endTime = currentTime();
        delta = (endTime - startTime) / (loops * TIME_FACTOR);
        startTime = endTime;
        loops = 0;
    }
}
```