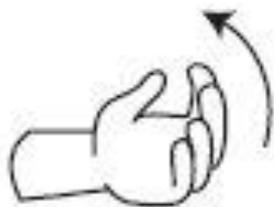
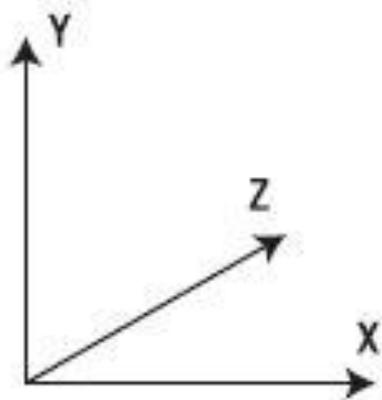


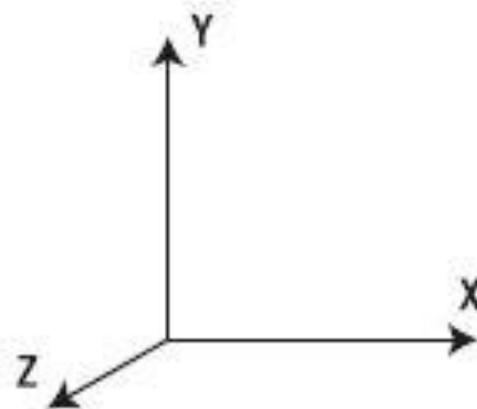


# **3D АФФИННЫЕ ПРЕОБРАЗОВАНИЯ**

# Аффинные преобразования в пространстве



**Левосторонняя  
трехмерная  
система координат**



**Правосторонняя  
трехмерная  
система координат**

# Аффинные преобразования в пространстве

## I. Вращение в пространстве

Матрица вращения вокруг оси абсцисс на угол  $\varphi$  :

$$[R_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi & \sin \varphi & 0 \\ 0 & -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Аффинные преобразования в пространстве

**Матрица вращения вокруг оси ординат на угол  $\psi$ :**

$$[R_y] = \begin{bmatrix} \cos \psi & 0 & -\sin \psi & 0 \\ 0 & 1 & 0 & 0 \\ \sin \psi & 0 & \cos \psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Аффинные преобразования в пространстве

**Матрица вращения вокруг оси аппликата на угол  $\theta$ :**

$$[R_z] = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Аффинные преобразования в пространстве

## II. Масштабирование:

$$[D] = \begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Коэффициенты:

- $\alpha > 0$  – вдоль оси абсцисс,
- $\beta > 0$  – вдоль оси ординат,
- $\gamma > 0$  – вдоль оси аппликат.

# Аффинные преобразования в пространстве

## III. Отражение

Матрица отражения относительно плоскости  $XOY$ :

$$[M_z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Аффинные преобразования в пространстве

**Матрица отражения относительно плоскости YOZ:**

$$[M_X] = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Аффинные преобразования в пространстве

**Матрица отражения относительно плоскости XOZ:**

$$[M_Y] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

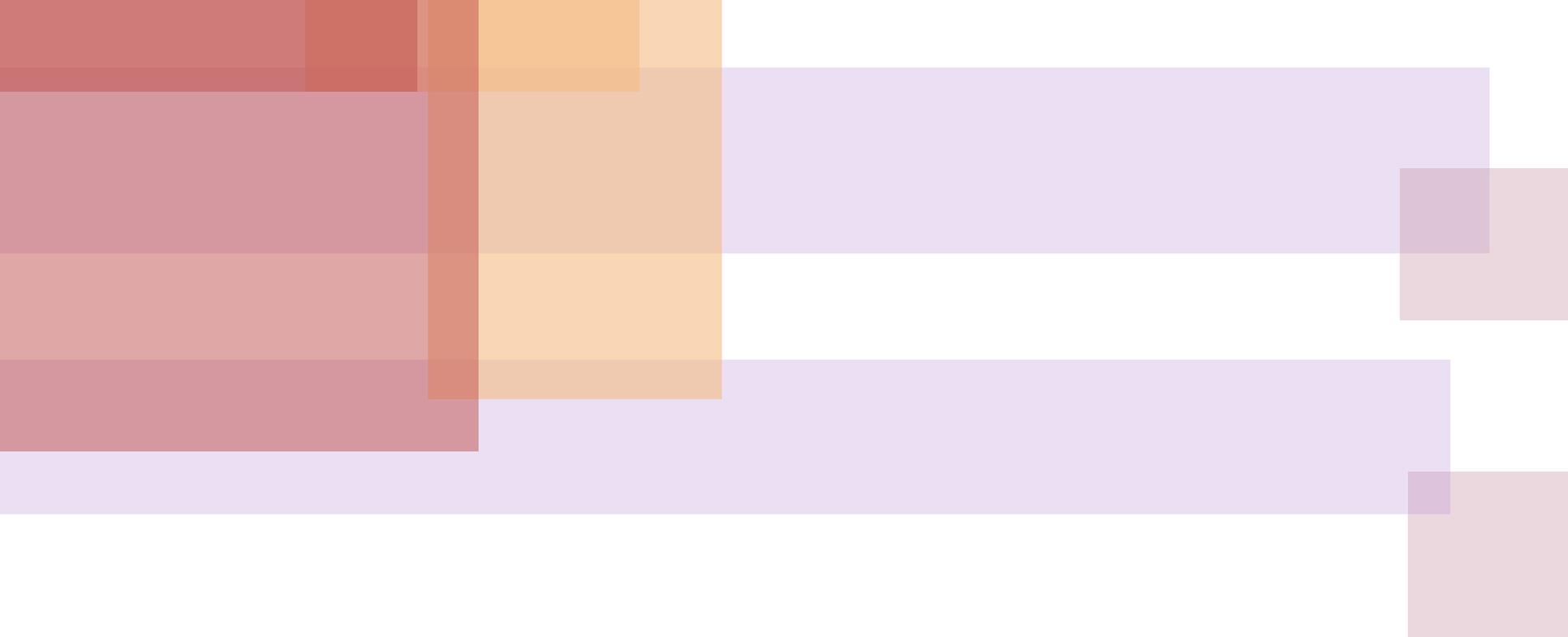
# Аффинные преобразования в пространстве

## IV. Перенос

Матрица переноса:

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \lambda & \mu & \nu & 1 \end{bmatrix}$$

где  $(\lambda, \mu, \nu)$  – вектор переноса.



# **ПРОЕКТИВНЫЕ ПРЕОБРАЗОВАНИЯ**

# МЕТОДЫ И АЛГОРИТМЫ ТРЕХМЕРНОЙ ГРАФИКИ

Примитивы вывода в мировых координатах

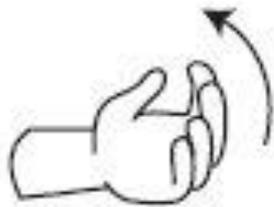
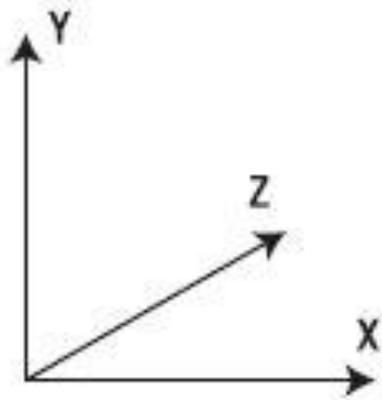
Отсечение по объему видимости

Проецирование на картинную плоскость

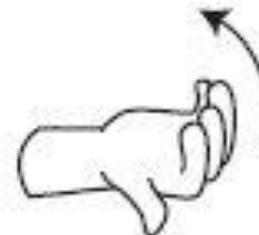
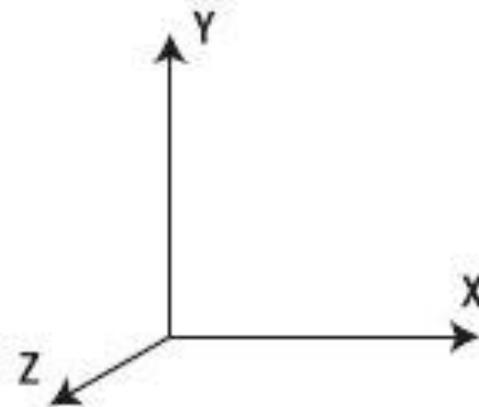
Преобразование в координаты устройства

# Аффинные преобразования в пространстве

## Системы координат

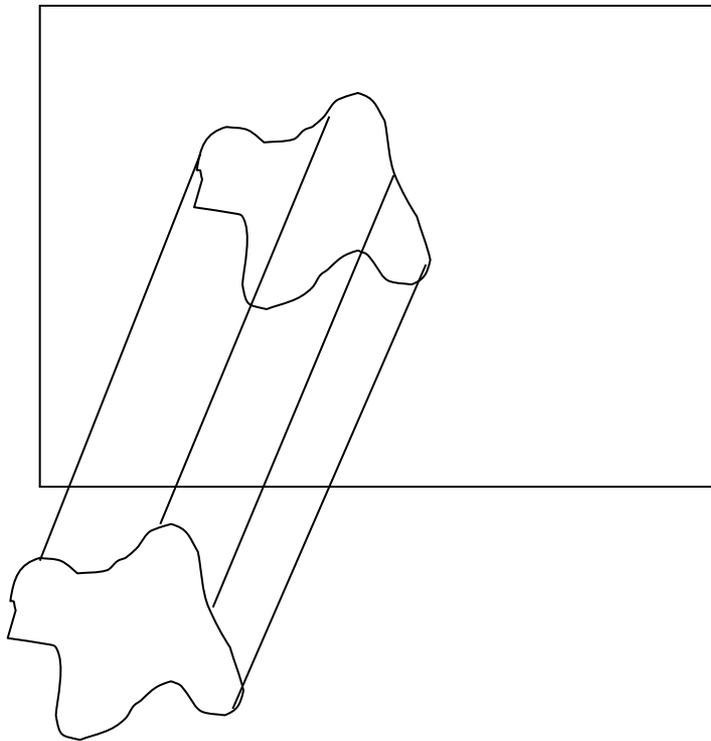


**Левосторонняя  
трехмерная  
система координат**

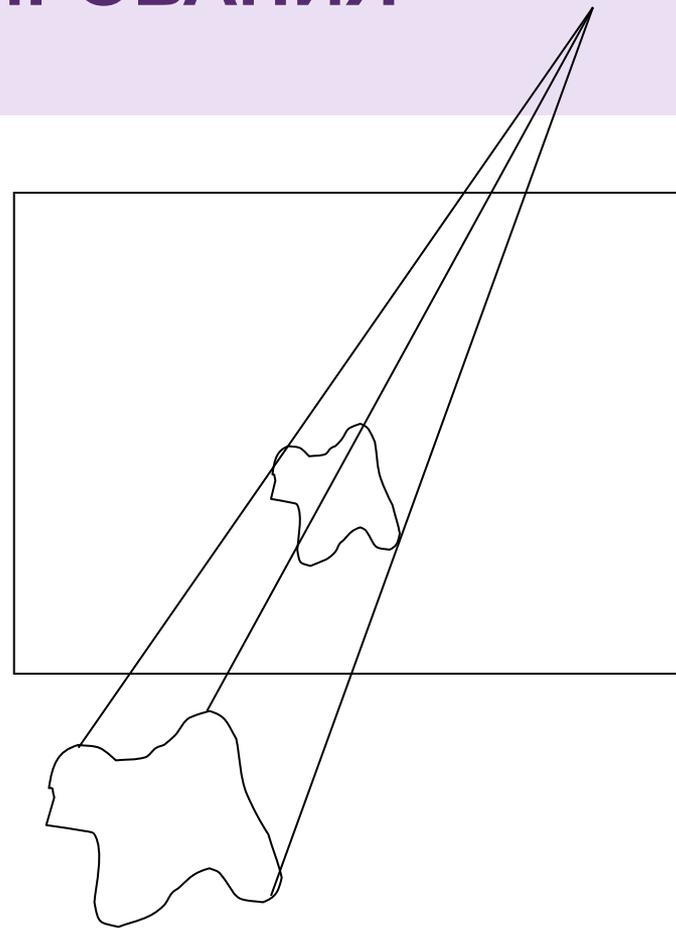


**Правосторонняя  
трехмерная  
система координат**

# ВИДЫ ПРОЕКТИРОВАНИЯ



**Параллельная  
проекция**



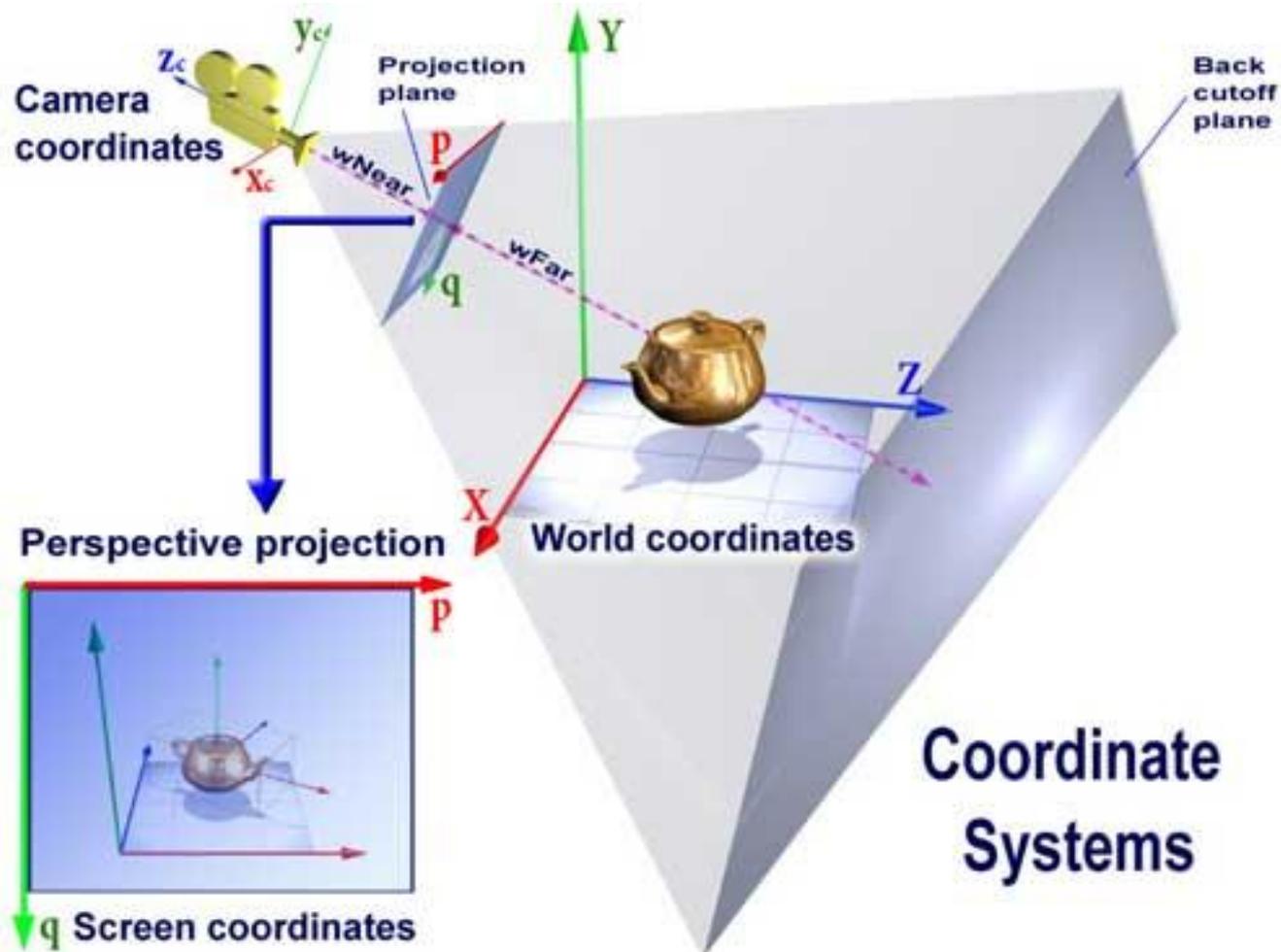
**Перспективная  
проекция**

# Преобразования координат

Представим цепочку преобразований координат от мировых к экранным следующим образом



# Системы координат

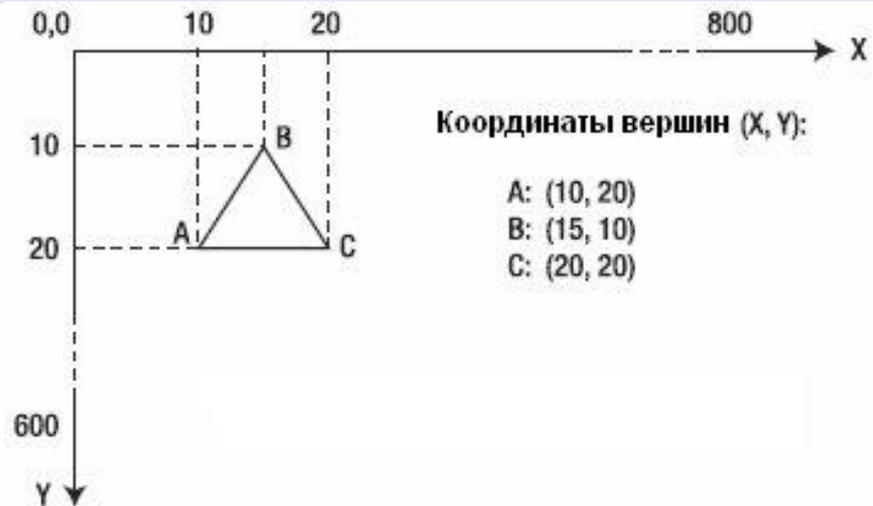
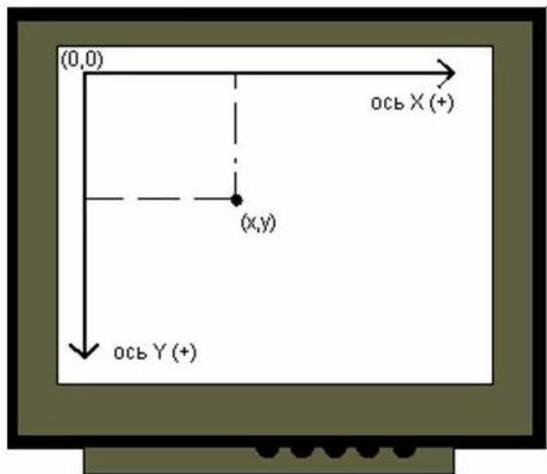


# Мировая (глобальная) система координат

*Мировая система координат (МСК) — xyz — содержит точку отсчета (начало координат) и линейно независимый базис, благодаря которым становится возможным цифровое описание геометрических свойств любого графического объекта в абсолютных единицах.*



# Экранная система координат

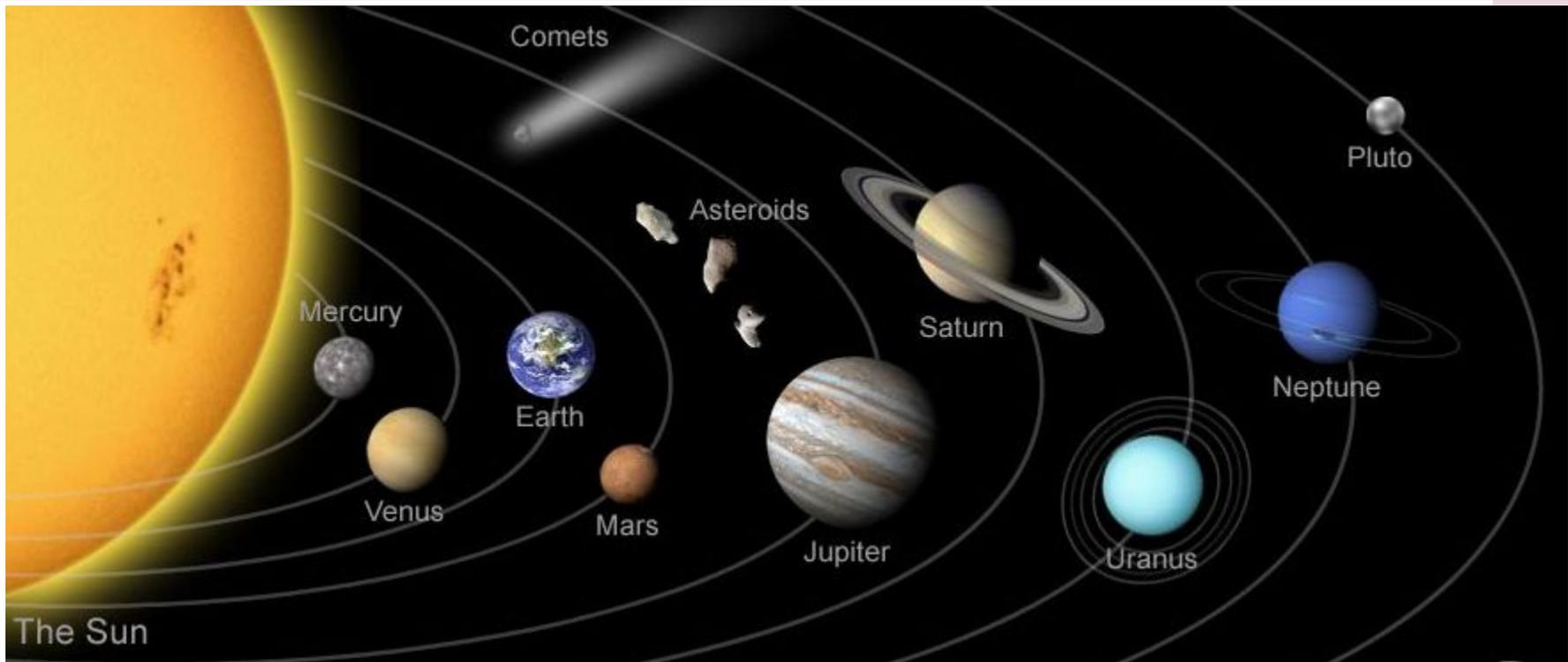


.....

*Экранная система координат (ЭСК) —  $x_3y_3z_3$  — система координат, в которой задается положение проекций геометрических объектов на экране дисплея.*

.....

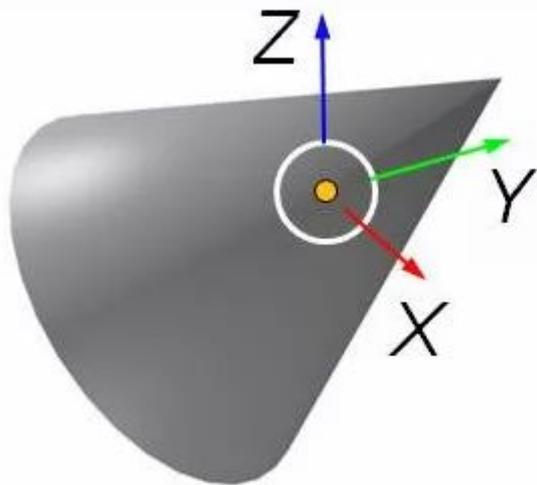
# Система координат сцены



*Система координат сцены (СКС) —  $x_c y_c z_c$  — описывает положение всех объектов сцены — некоторой части мирового пространства с собственным началом отсчета и базисом, которые используются для описания положения объектов независимо от МСК.*

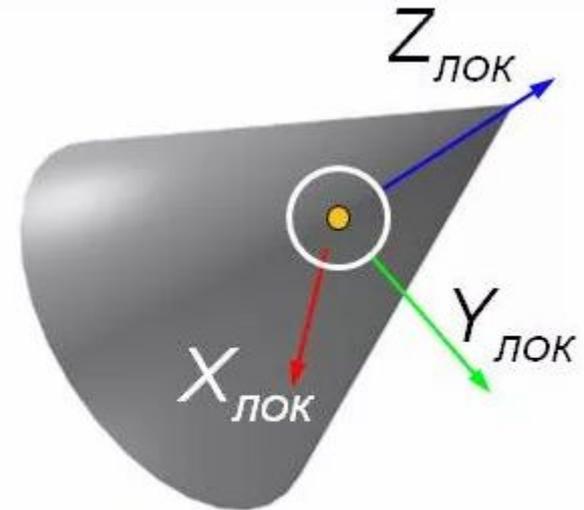
.....

# Объектная система координат



глобальная («мировая»)

не зависит от положения объекта



локальная

связана с объектом

.....  
*Объектная система координат (ОСК) —  $x_0y_0z_0$  — связана с конкретным объектом и совершает с ним все движения в СКС или МСК.*  
.....

tandalone <DX11>

ools Window Help



Scene

Textured

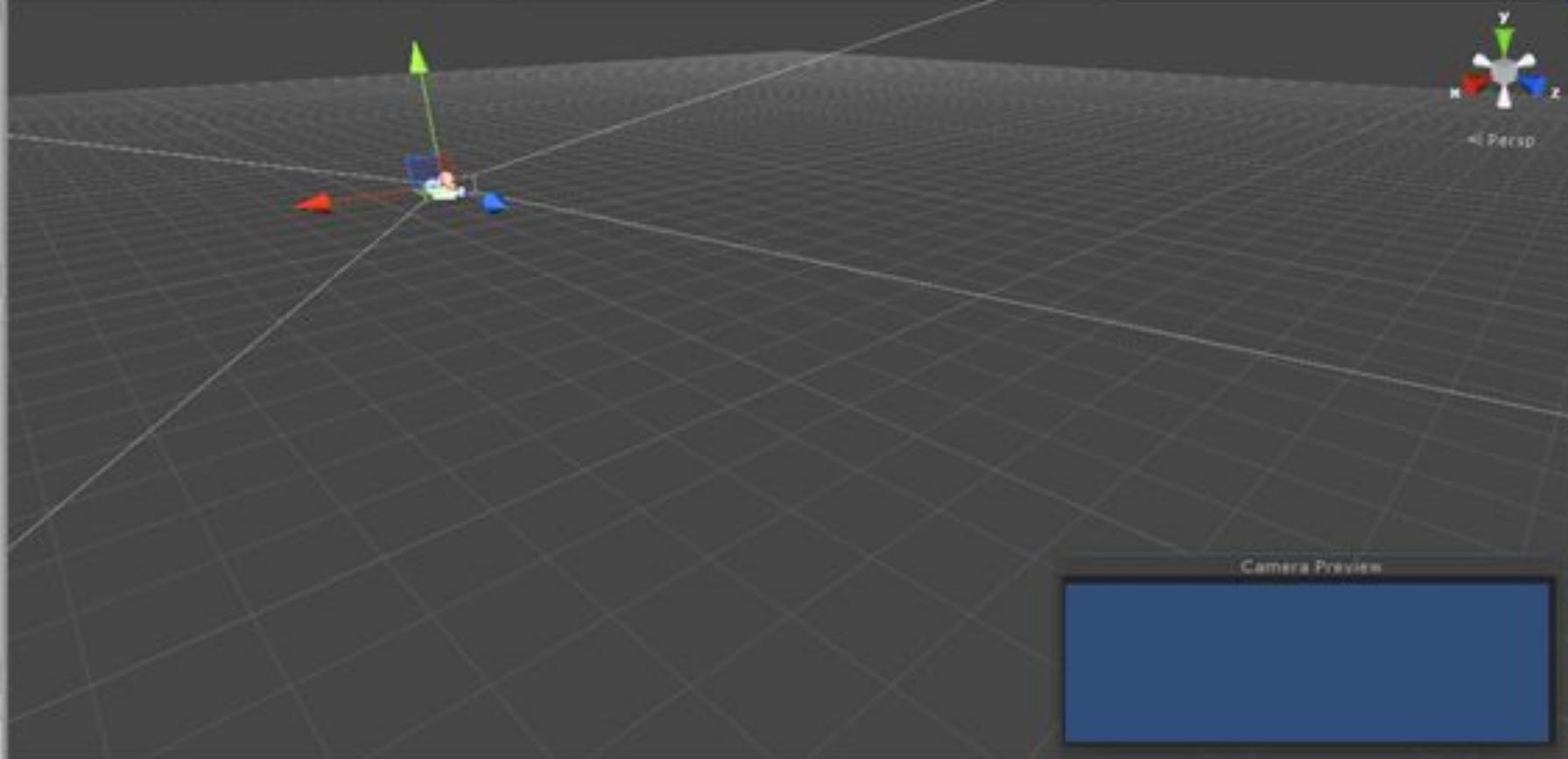
RGB

2D

Effects

Gizmos

All



Camera Preview



Game

Free Aspect

Maximize on Play

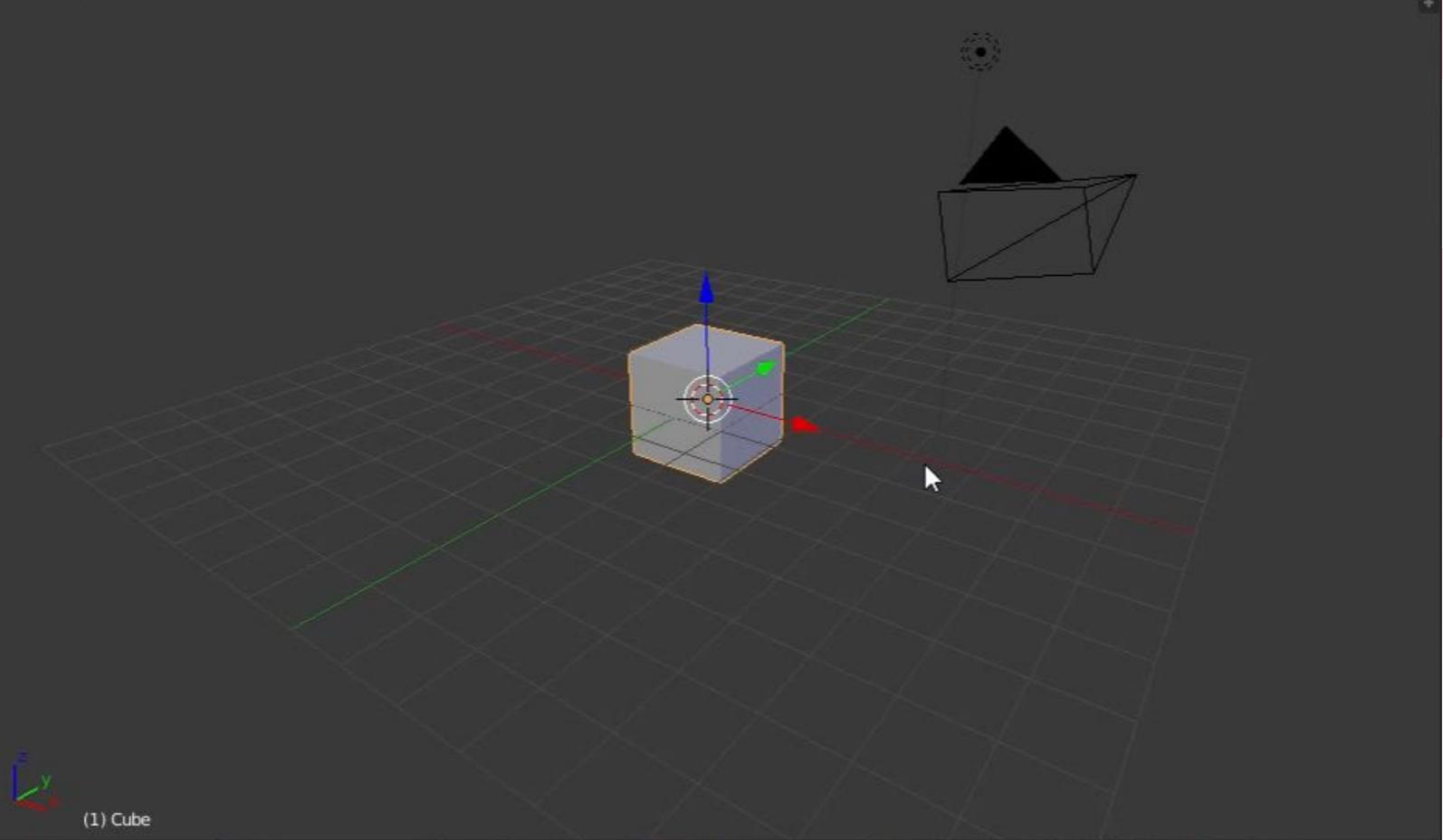
Stats

Gizmos

- ▼ Преобразования
  - Перемещение
  - Вращение
  - Масштаб
  - Отразить
- ▼ Правка
  - Дублировать
  - Дубли...язями
  - Удалить
  - Объединить
  - Задат... точку
  - Затенение:
    - Гладко
    - Плоско
  - Перенос данн...
  - Данны
  - Разме

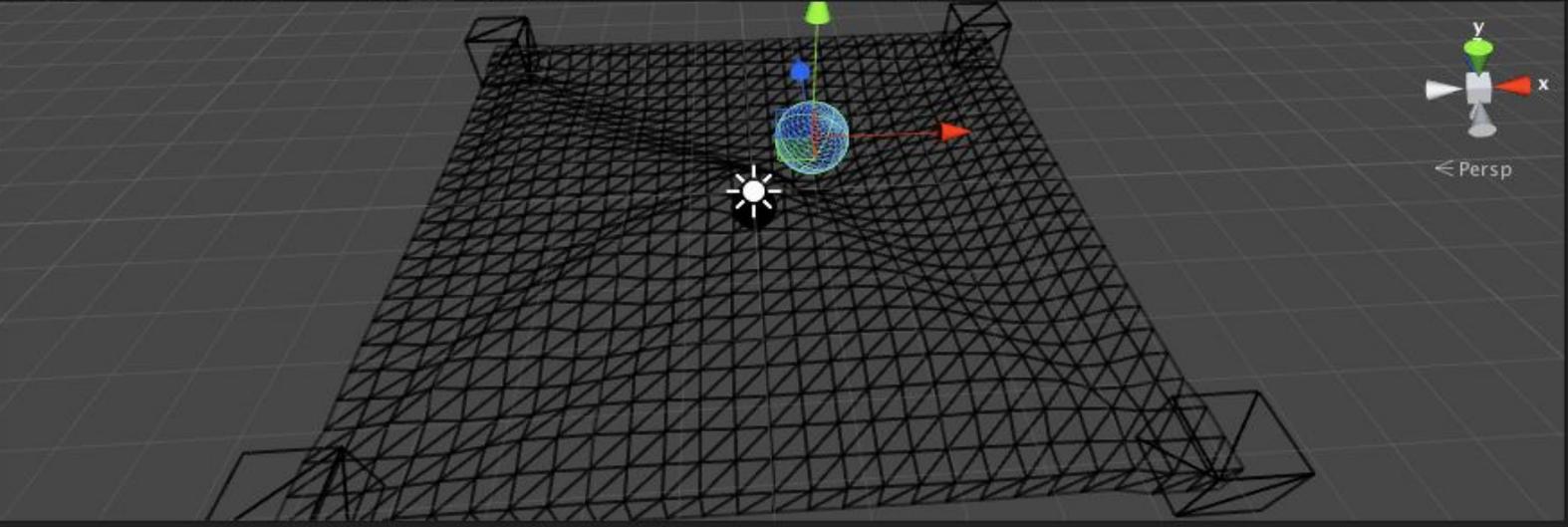
▼ Оператор

Польз. персп.



Hand icon, Plus icon, Refresh icon, Rotate icon, Lock icon, Pivot, Global, Play, Pause, Stop

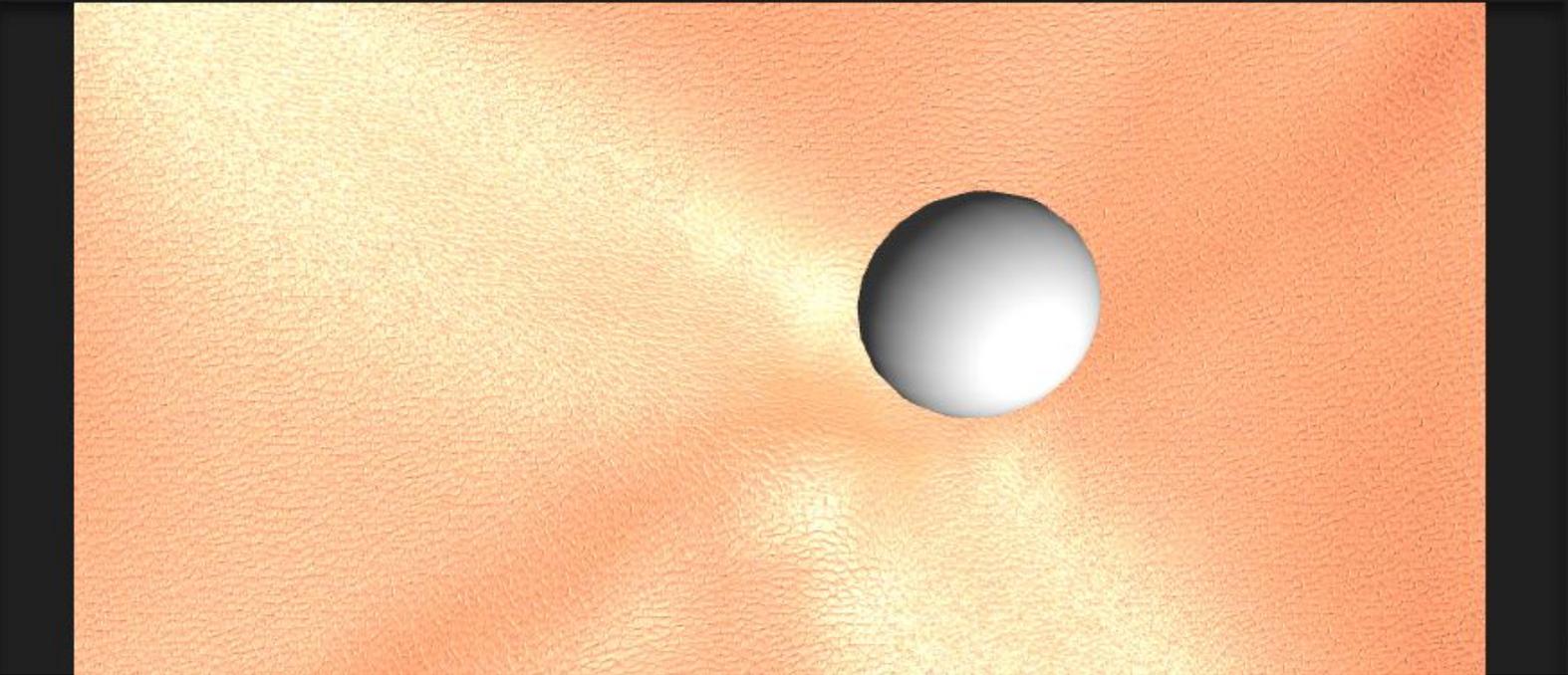
# Scene, Animator, Wireframe, RGB, 2D, Effects, Gizmos, All



Hierarchy

- Create + All
- Main Camera
- Skinv2
- Directional light
- Colliders
- Point light
- Sphere
- Sphere
- Walls
- Rigidbody dragger

Game, Animation, WXGA Landscape (1280x800), Maximize on Play, Stats, Gizmos

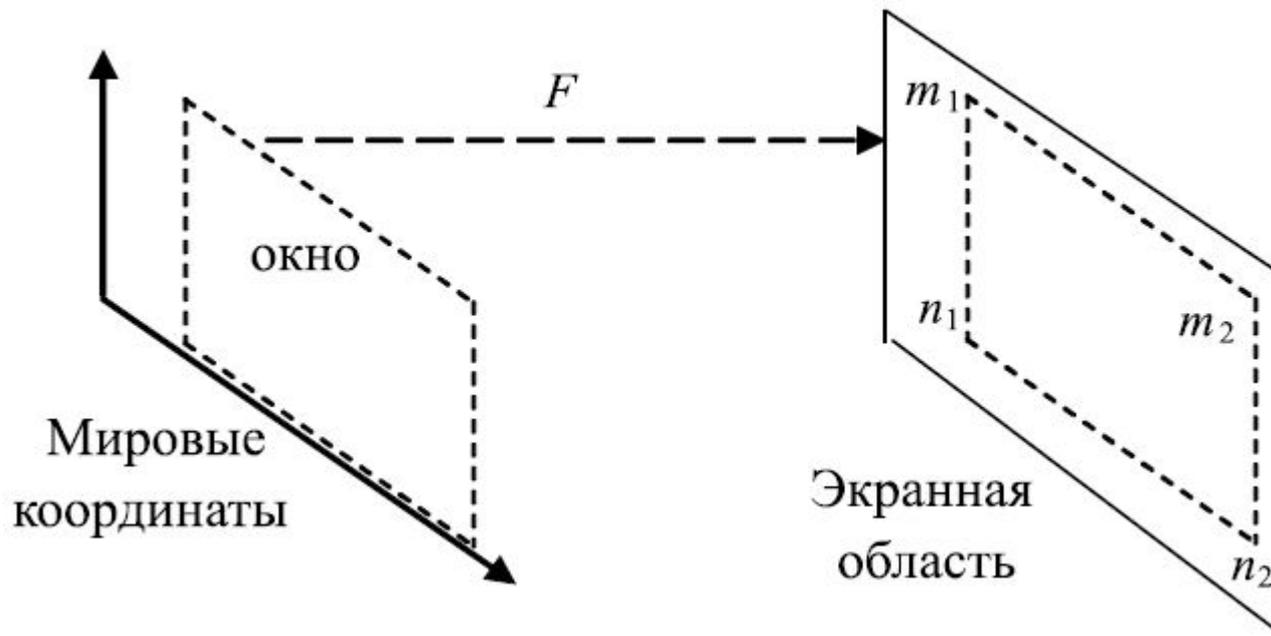


Console

Clear, Collapse, Clear on Play, Er

# Область визуализации и функция кадрирования

Пусть рабочая область визуализации ограничена прямыми  $X = n_1$ ,  $Y = m_1$ ,  $X = n_2$ ,  $Y = m_2$ .  $X$ ,  $Y$  — координаты на экране. Где  $m_1$ ,  $m_2$ ,  $n_1$ ,  $n_2$ , — координаты растра, и пусть  $W = R \times R$  — произвольное прямоугольное окно в мировой системе координат. Тогда функция  $F: W \rightarrow [m_1 : m_2] \times [n_1 : n_2]$  будет называться функцией кадрирования.



# Функция кадрирования

Вначале вычисляются коэффициенты масштабирования по осям

$$f_x = \frac{X_{\max} - X_{\min}}{x_{\max} - x_{\min}}, \quad f_y = \frac{Y_{\max} - Y_{\min}}{y_{\max} - y_{\min}},$$

Затем вычисляем расстояние  $X - X_{\min}$  точки изображения от левого края экранной области:

$$\begin{cases} X = X_{\min} + f_x(x - x_{\min}) \\ Y = Y_{\min} + f_y(y - y_{\min}) \end{cases}$$

# ПРОЕКЦИИ

## ПАРАЛЛЕЛЬНЫЕ

ОРТОГРАФИЧЕСКАЯ

АКСОНОМЕТРИЧЕСКАЯ

КОСОУГОЛЬНАЯ

ТРИМЕТРИЧЕСКАЯ

ДИМЕТРИЧЕСКАЯ

ИЗОМЕТРИЧЕСКАЯ

СВОБОДНАЯ

КАБИНЕТНАЯ

## ПЕРСПЕКТИВНЫЕ

ОДНОТОЧЕЧНАЯ

ДВУХТОЧЕЧНАЯ

ТРЕХТОЧЕЧНАЯ

# Параллельное проектирование

## *Ортографическая проекция*

$$[P_x] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица проектирования вдоль оси X на плоскость YOZ

Если плоскость проектирования параллельна координатной плоскости, необходимо умножить матрицу на матрицу переноса:

$$[P_x^\lambda] = [P_x] \cdot [T_x] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \lambda & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \lambda & 0 & 0 & 1 \end{bmatrix}.$$

$$[P_y^\mu] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \mu & 0 & 1 \end{bmatrix}, \quad [P_z^\nu] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \nu & 1 \end{bmatrix}.$$

## *АксонOMETрическая проекция*

**Триметрия** – нормаль к картинной плоскости образует с оортами координатных осей попарно различные углы.

$$[T] = \begin{bmatrix} 0.866 & 0.354 & 0 & 0 \\ 0 & 0.707 & 0 & 0 \\ 0.5 & -0.612 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

## *АксонOMETрическая проекция*

**Диметрия** – два угла между нормалью к картинной плоскости и координатными осями равны.

$$[D] = \begin{bmatrix} 0.926 & 0.134 & 0 & 0 \\ 0 & 0.935 & 0 & 0 \\ 0.378 & -0.327 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

# АксонOMETрическая проекция

**Изометрия** – все три угла между нормалью к картинной плоскости и координатными осями равны.

$$[I] = \begin{bmatrix} 0.707 & -0.408 & 0 & 0 \\ 0 & 0.816 & 0 & 0 \\ -0.707 & -0.408 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

# Параллельное проектирование

## *Косоугольная проекция*

При косоугольном проектировании на плоскость  $XOY$  имеем:

$$(0 \ 0 \ 1 \ 1) \boxtimes (\alpha \ \beta \ 0 \ 1).$$

Матрица преобразования имеет вид:

$$[K_z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \alpha & \beta & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

# Косоугольная проекция

1. Свободная проекция:

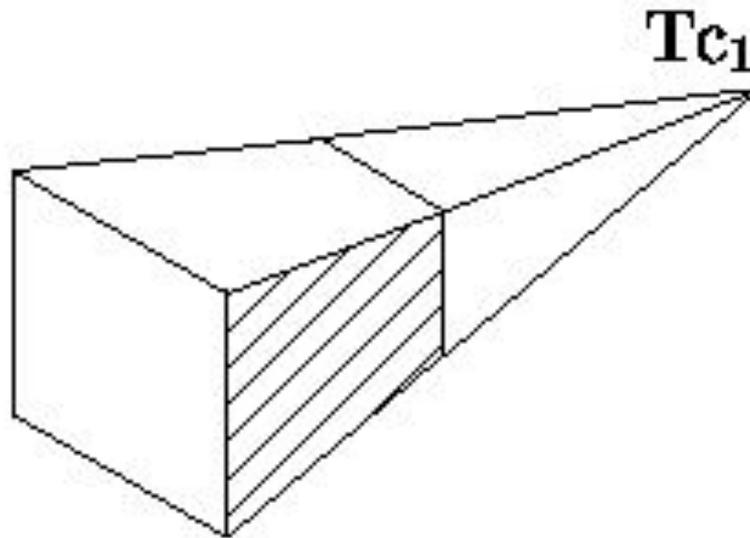
$$\alpha = \beta = \cos \frac{\pi}{4}.$$

2. Кабинетная проекция:

$$\alpha = \beta = \frac{1}{2} \cos \frac{\pi}{4}.$$

# Перспективное проектирование

*Одноточечное перспективное преобразование*



$$[E_z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

## Одноточечное перспективное преобразование

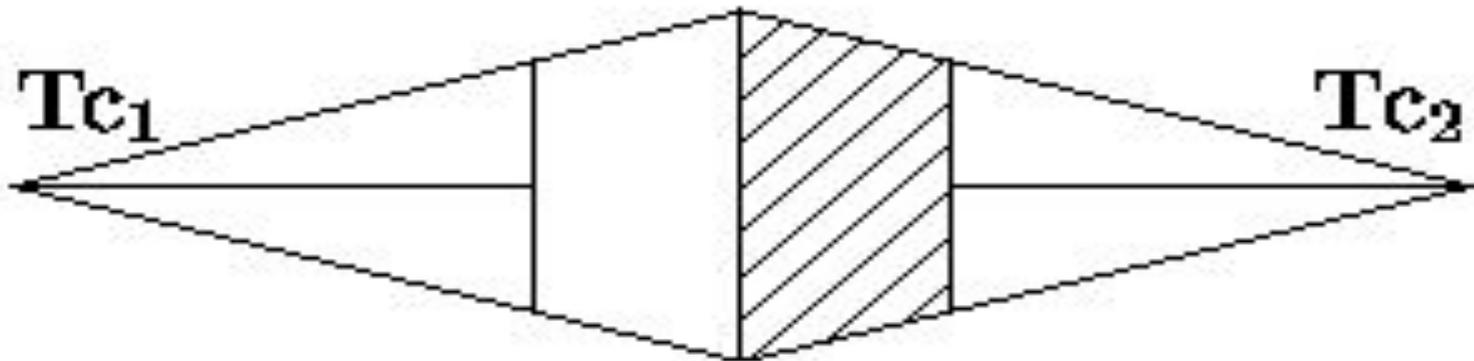
$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot [E_z] = \begin{bmatrix} \frac{x}{r \cdot z + 1} & \frac{y}{r \cdot z + 1} & 0 & 1 \end{bmatrix}.$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \cdot [E_z] = \begin{bmatrix} 0 & 0 & 1 & r \end{bmatrix}.$$

$$\begin{bmatrix} 0 & 0 & \frac{1}{r} & 1 \end{bmatrix}$$

# Перспективное проектирование

*Двухточечное перспективное преобразование*

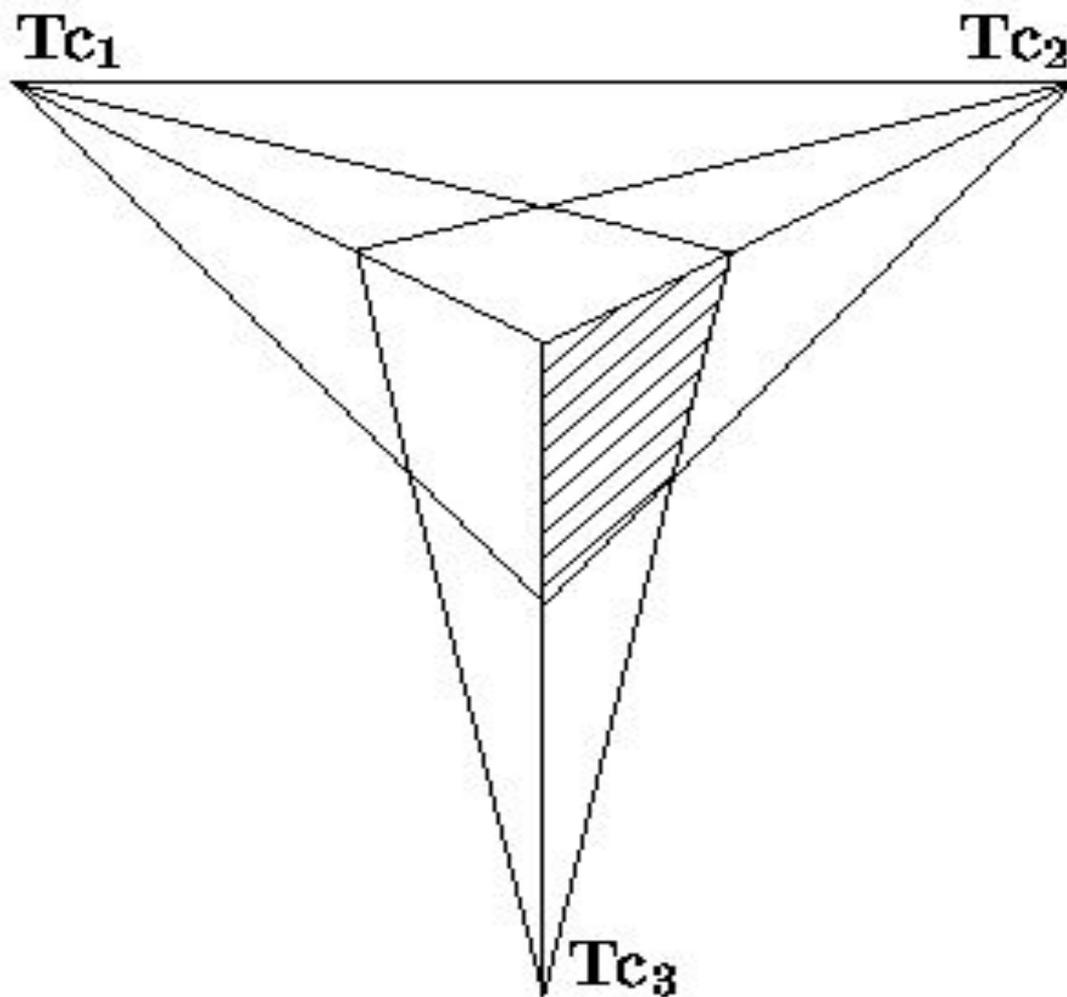


## Двухточечное перспективное преобразование

$$\begin{aligned} & [x \quad y \quad z \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \\ & = \begin{bmatrix} \frac{x}{p \cdot x + q \cdot y + 1} & \frac{y}{p \cdot x + q \cdot y + 1} & \frac{z}{p \cdot x + q \cdot y + 1} & 1 \end{bmatrix}. \\ & \begin{bmatrix} 1/p & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1/q & 0 & 1 \end{bmatrix} \end{aligned}$$

# Перспективное проектирование

*Трехточечное перспективное преобразование*



$$[x \quad y \quad z \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{x}{p \cdot x + q \cdot y + r \cdot z + 1} & \frac{y}{p \cdot x + q \cdot y + r \cdot z + 1} & \frac{z}{p \cdot x + q \cdot y + r \cdot z + 1} & 1 \end{bmatrix}.$$

▪ на оси x:  $\begin{bmatrix} 1/p & 0 & 0 & 1 \end{bmatrix}$

▪ на оси y:  $\begin{bmatrix} 0 & 1/q & 0 & 1 \end{bmatrix}$

▪ на оси z:  $\begin{bmatrix} 0 & 0 & 1/r & 1 \end{bmatrix}$



**OpenGL (Open Graphic Library) – библиотека графических функций, интерфейс для графических прикладных программ.**

# Преимущества OpenGL

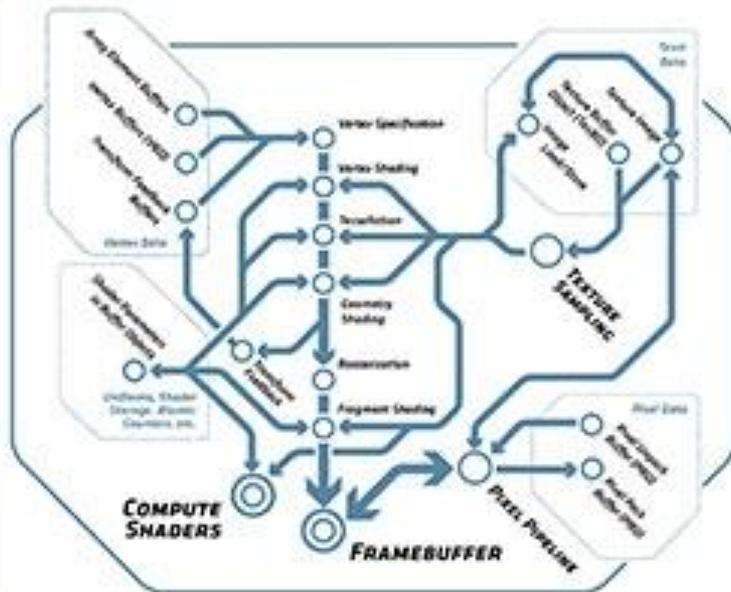
**Open Graphics Library — открытая графическая библиотека, графическое API — спецификация, определяющая независимый от языка программирования кроссплатформенный программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.**

# OpenGL

- Включает более 250 функций для рисования сложных трёхмерных сцен из простых (2D) примитивов.
- Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях.

# История развития OpenGL

## Continuing OpenGL Innovation



Bringing state-of-the-art functionality to cross-platform graphics

OpenGL 2.0

OpenGL 2.1

OpenGL 3.0

OpenGL 3.1  
OpenGL 3.2  
OpenGL 3.3/4.0  
OpenGL 4.1  
OpenGL 4.2  
OpenGL 4.3  
OpenGL 4.4  
OpenGL 4.5

2004

2005

2006

2007

2008

2009

2010

2011

2012

2013

2014

DirectX 9.0c

DirectX 10.0

DirectX 10.1

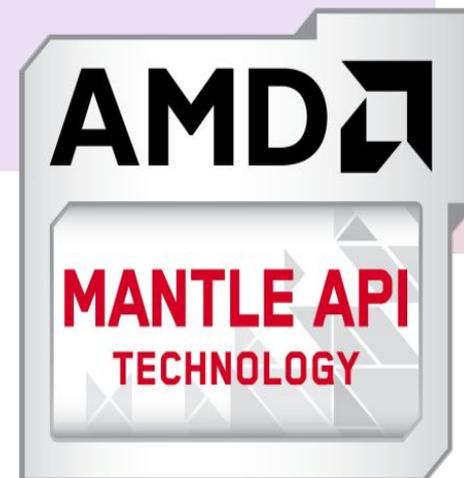
DirectX 11

DirectX 11.1

DirectX 11.2

# История развития OpenGL

- ▶ **2015 Mantle** - спецификация низкоуровневого API, разработанная компанией AMD в качестве альтернативы DirectX и OpenGL.
- ▶ **Февраль 2016** - Первый публичный релиз Vulkan.
- ▶ **Март 2018** – Vulkan 1.1, открытый графический API доступен на Windows, Linux, Android, Nintendo Switch и различных облачных системах, но его не поддерживает ни одна платформа Apple.

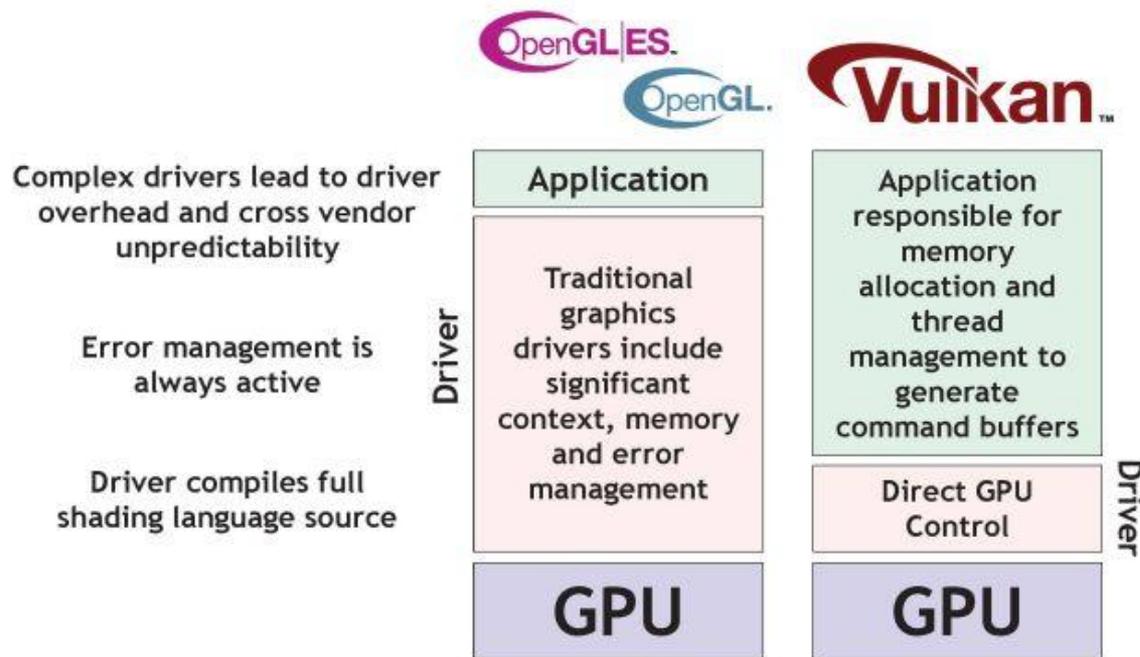


# MoltenVK

- ❑ **Единственная возможность  
работать с Vulkan на MacOS**
- ❑ **Стоимость  $\approx 250\$$**
- ❑ **Работает с потерей  
производительности**



# Vulkan Explicit GPU Control



- Ручное управление памятью
- Отсутствие проверок в драйвере
- Спроектирован для многопоточности

<b>Сравнение</b>	<b>OpenGL 2.x</b>	<b>OpenGL 4.x</b>	<b>DirectX 12</b>	<b>Vulkan</b>
Платформы	<b>Win/Unix/Mac</b>	<b>Win/Unix/Mac</b>	<b>Win</b>	*
Временная эффективность	Минимальная	Средняя	Максималь-ная	Максималь-ная

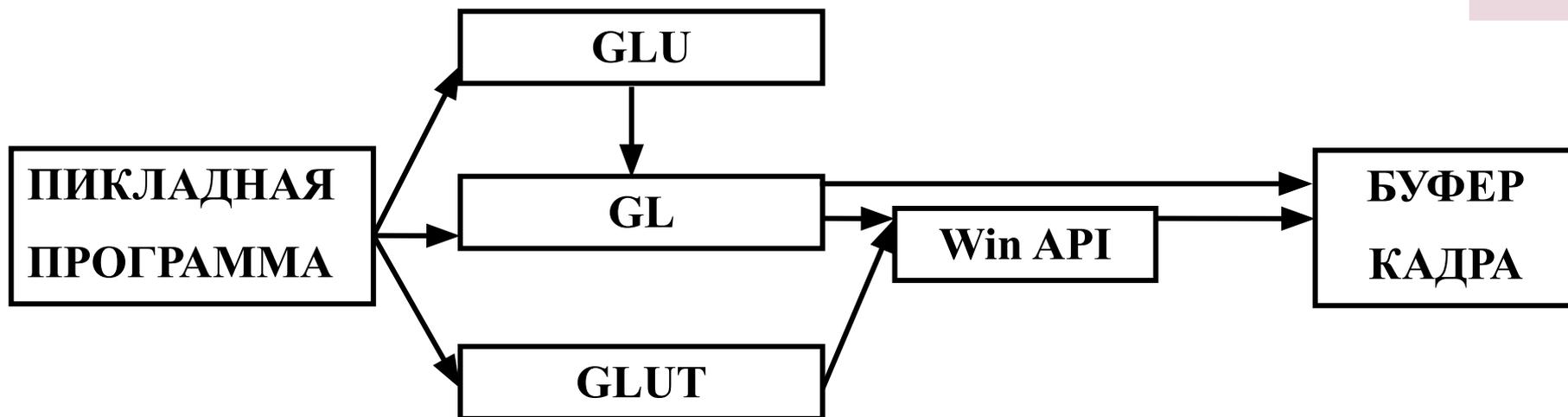
•- **Vulkan это:**

• а) Лишь стандарт, а значит реализация лежит на разработчиках драйверов.

• б) Очень молодой стандарт (~2 года)

► На данный момент нет поддержки для MacOS.

# Интерфейс OpenGL



- Graphics Library (GL)
- Graphics Library Utility (GLU)
- Graphics Library Utility Toolkit (GLUT)

# Соглашение о наименовании функций

```
glBegin(<тип>);    // указываем тип примитива  
    glVertex[2 3][ i f v](...); // первая вершина  
    .....// остальные вершины  
    glVertex[2 3][ i f v](...); // последняя вершина  
glEnd;
```

*В OpenGL левый нижний угол области вывода имеет координаты [-1; -1], правый верхний [1; 1].*

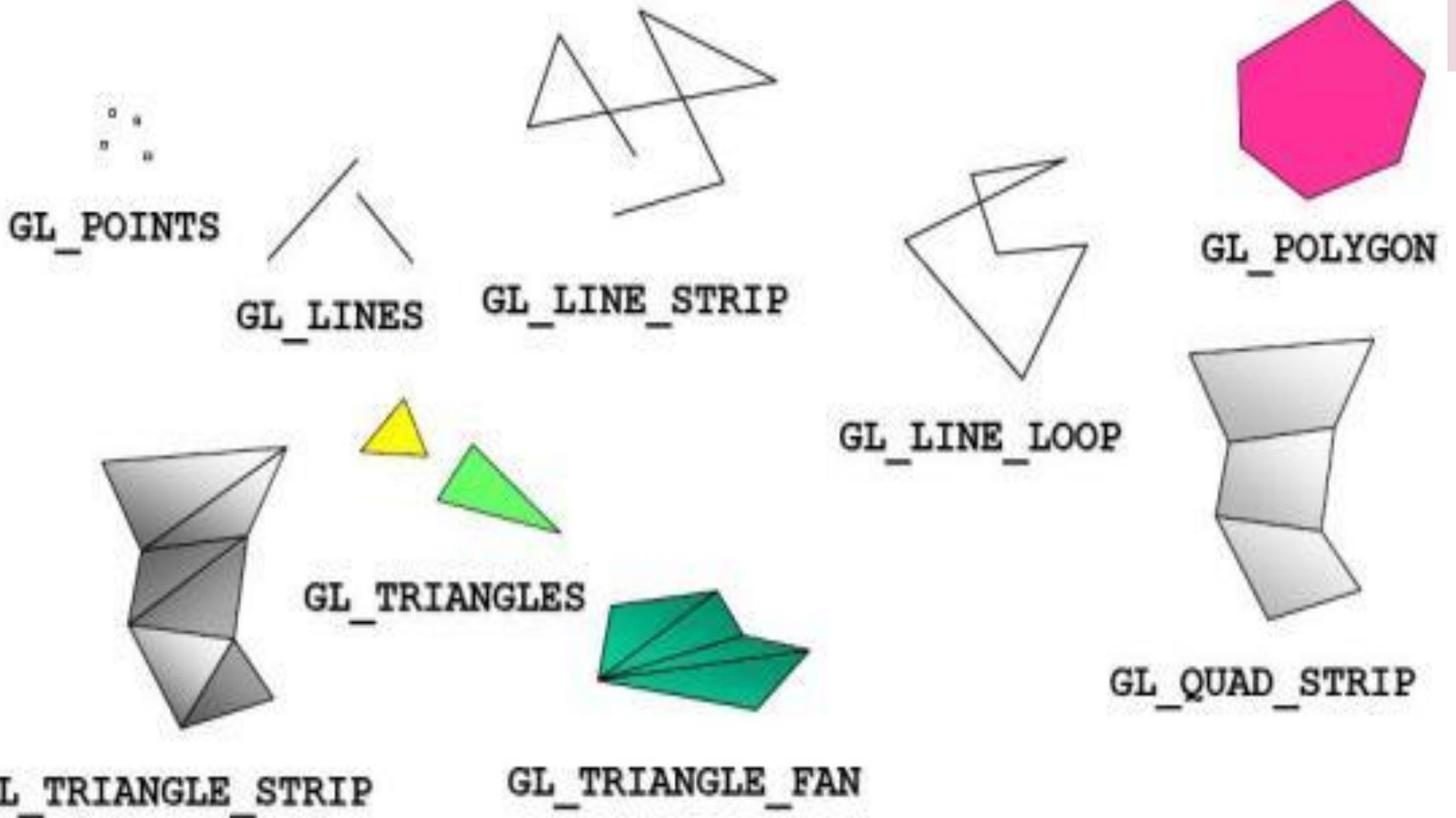
# Визуализация двумерных объектов

<b>Значение &lt;тип&gt;</b>	<b>Описание</b>
<b>GL_POINTS</b>	<b>Каждый вызов glVertex задает отдельную точку</b>
<b>GL_LINES</b>	<b>Каждая пара вершин задает отрезок</b>
<b>GL_LINE_STRIP</b>	<b>Рисуется ломанная</b>
<b>GL_LINE_LOOP</b>	<b>Рисуется ломанная, причем ее последняя точка соединяется с первой</b>

# Визуализация двумерных объектов

<b>GL_TRIANGLES</b>	<b>Тройки вершин образуют треугольник</b>
<b>GL_TRIANGLE_STRIP</b>	<b>Связанные треугольники</b>
<b>GL_TRIANGLE_FAN</b>	<b>Связанные треугольники с общей первой вершиной</b>
<b>GL_QUADS</b>	<b>Каждые четыре вершины образуют четырехугольники</b>
<b>GL_QUAD_STRIP</b>	<b>Связанные четырехугольники</b>
<b>GL_POLYGON</b>	<b>Один выпуклый многоугольник</b>

# Визуализация двумерных объектов



## Область вывода

После применения матрицы проекций на вход следующего преобразования подаются усеченные координаты, для которых значения всех компонент  $(x_c, y_c, z_c, w_c)T$  находятся в отрезке  $[-1,1]$ .

После этого находятся нормализованные координаты вершин по формуле:

$$(x_n, y_n, z_n)T = (x_c/w_c, y_c/w_c, z_c/w_c)T.$$

Область вывода:

`glViewport(x, y, width, height: GLint).`

# МАТРИЦЫ OPENGL

**glMatrixMode(mode);**

Mode:

- ▶ Видовая: ***GL\_MODELVIEW***,
- ▶ Проекций: ***GL\_PROJECTION***,
- ▶ Текстуры: ***GL\_TEXTURE***

***glLoadIdentity = glPushMatrix  
glPopMatrix***

# ПРЕОБРАЗОВАНИЯ OPENGL

- ▶ Масштабирование:

`glScale[2 3] [I f d] ( $\alpha$ ,  $\beta$ ,  $\gamma$ );`

$\alpha$  – относительно оси абсцисс,

$\beta$  – ординат;  $\gamma$  – аппликат.

- ▶ Поворот:

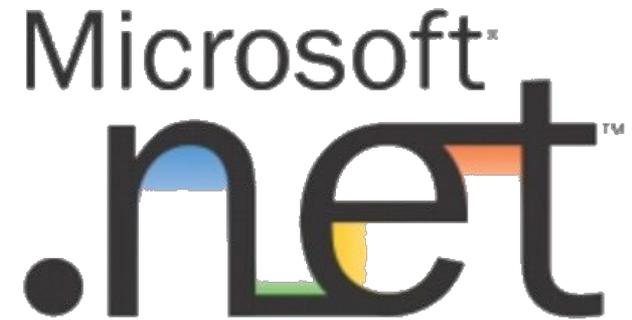
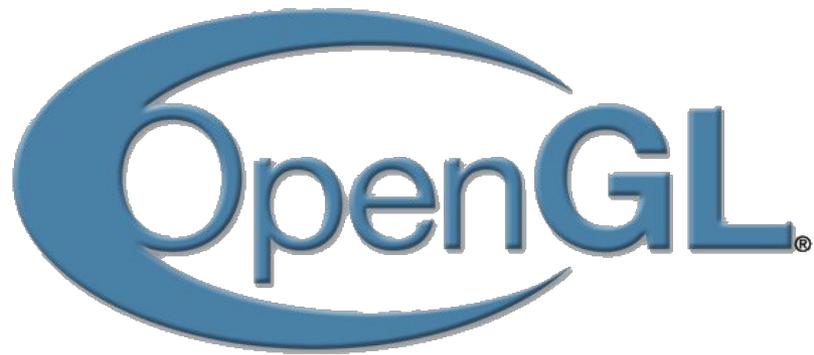
`glRotate[2 3] [I f d] ( $\alpha$ ,  $x$ ,  $y$ ,  $z$ );`

Значения  $x$ ,  $y$ ,  $z \in \{0, 1\}$

- ▶ Сдвиг:

`glTranslate[2 3] [I f d] ( $dx$ ,  $dy$ ,  $dz$ );`

# Tao Framework: OpenGL + C#



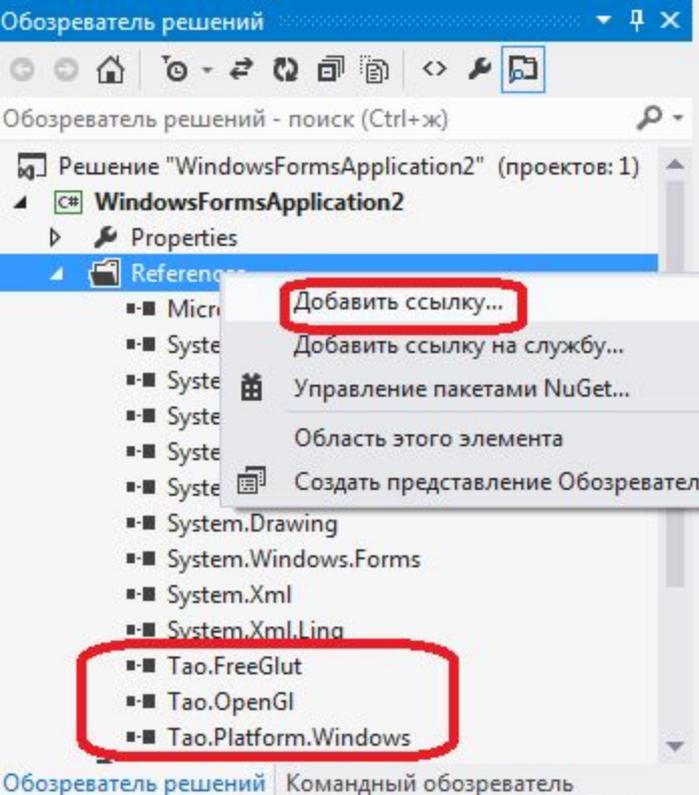
TAO

# Установка

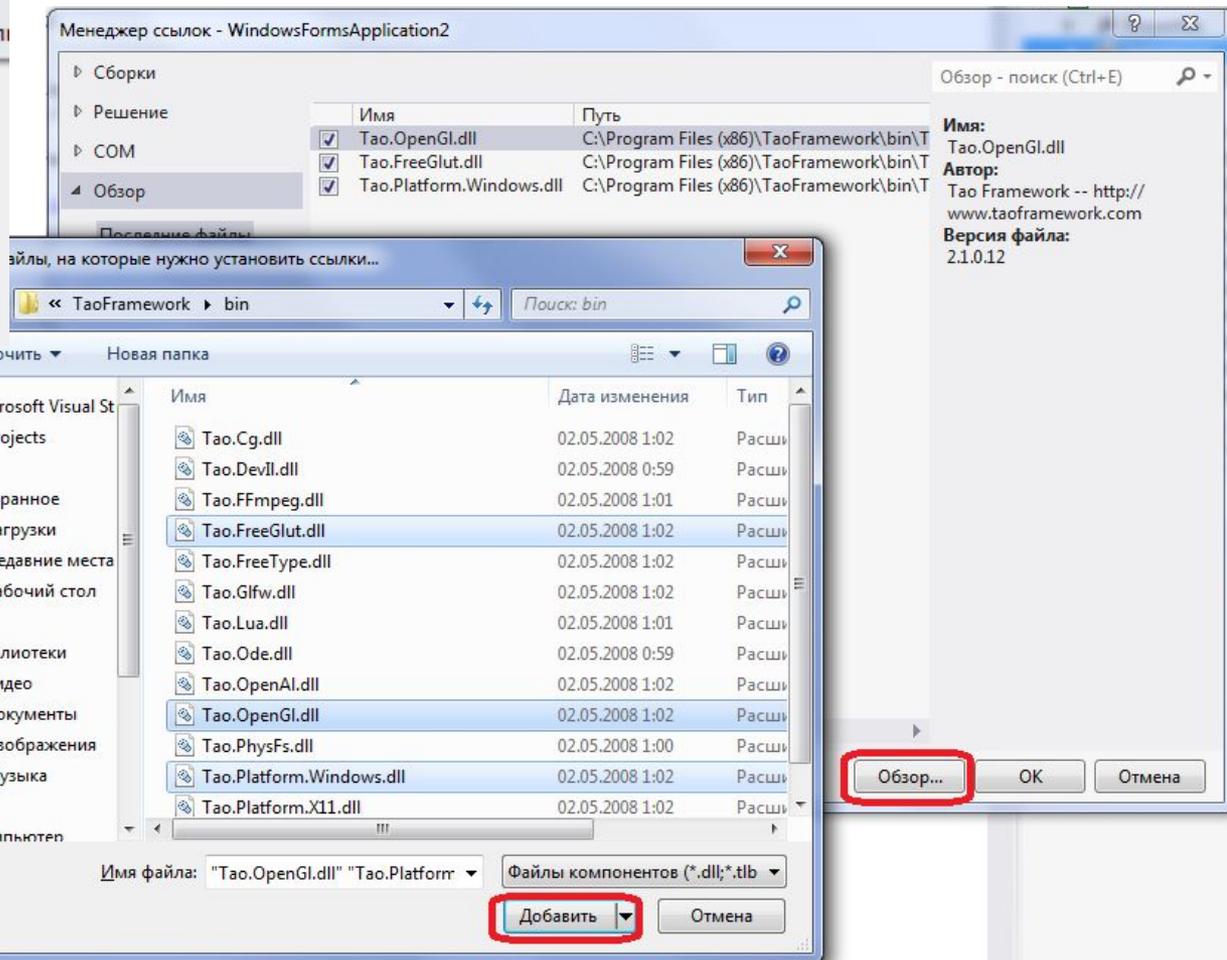
Скачать Tao Framework можно с официальной страницы фреймворка на sourceforge.net:

[www.sourceforge.net/projects/taoframework](http://www.sourceforge.net/projects/taoframework)

Установка крайне проста, состоит из нажатий кнопки "далее", ошибки могут возникнуть в случае, если у вас нет .NET 2.0 или выше, но, по умолчанию, он устанавливается вместе с Visual Studio.



В открывшемся окне «Добавить ссылку» перейдите к закладке «Обзор». После этого перейдите к директории, в которую была установлена библиотека **Tao Framework**. (По умолчанию – "C:\Program Files\Tao Framework").



- Панель элементов
- ▷ Все формы Windows Forms
  - ▷ Стандартные элементы упра...
  - ▷ Контейнеры
  - ▷ Меню и панели инструментов
  - ▷ Данные
  - ▷ Компоненты
  - ▷ Печать
  - ▷ Диалоговые окна
  - ▷ Взаимодействие WPF
  - ▷ Visual Basic PowerPacks
  - ▷ Общие

Выбор элементов панели элементов

Компоненты System.Activities		Компоненты Silverlight		Компоненты WPF	
Компоненты .NET Framework				COM-компоненты	
Имя	Пространство имен	Имя сборки			
<input checked="" type="checkbox"/> SerialPort	System.IO.Ports	System			
<input type="checkbox"/> ServiceBase	System.ServiceProcess	System.ServiceProcess			
<input checked="" type="checkbox"/> ServiceController	System.ServiceProcess	System.ServiceProcess			
<input type="checkbox"/> ServiceInstaller	System.ServiceProcess	System.ServiceProcess			
<input type="checkbox"/> ServiceProcessInstalle	System.ServiceProcess	System.ServiceProcess			
<input checked="" type="checkbox"/> SetState	System.Workflow.Activities	System.Workflow.Activities			
<input checked="" type="checkbox"/> SimpleOpenGIControl	Tao.Platform.Windows	Tao.Platform.Windows			
<input checked="" type="checkbox"/> SiteMapDataSource	System.Web.UI.WebControls	System.Web			
<input checked="" type="checkbox"/> SiteMapPath	System.Web.UI.WebControls	System.Web			

Фильтр:

SimpleOpenGIControl

Язык: Инвариантный язык (Инвариантная страна)  
Версия: 1.0.0.5

Очистить Обзор...

OK Отмена Сброс

- Панель элементов
- Вставить
  - Список
  - Показать все
  - Выбрать элементы...
  - Сортировать элементы по алф
  - Сброс панели элементов
  - Добавить вкладку
  - Удалить вкладку
  - Переименовать вкладку
  - Вверх
  - Вниз

- Панель элементов
- Источники данных
- ▷ Все формы Windows Forms
  - ▷ Стандартные элементы управления
  - ▷ Контейнеры
  - ▷ Меню и панели инструментов
  - ▷ Данные
  - ▷ Компоненты
  - ▷ Печать
  - ▷ Диалоговые окна
  - ▷ Отчеты
  - ▷ Взаимодействие WPF
  - ▷ Visual Basic PowerPacks
  - ▲ Общие
    - Указатель
    - SimpleOpenGIControl

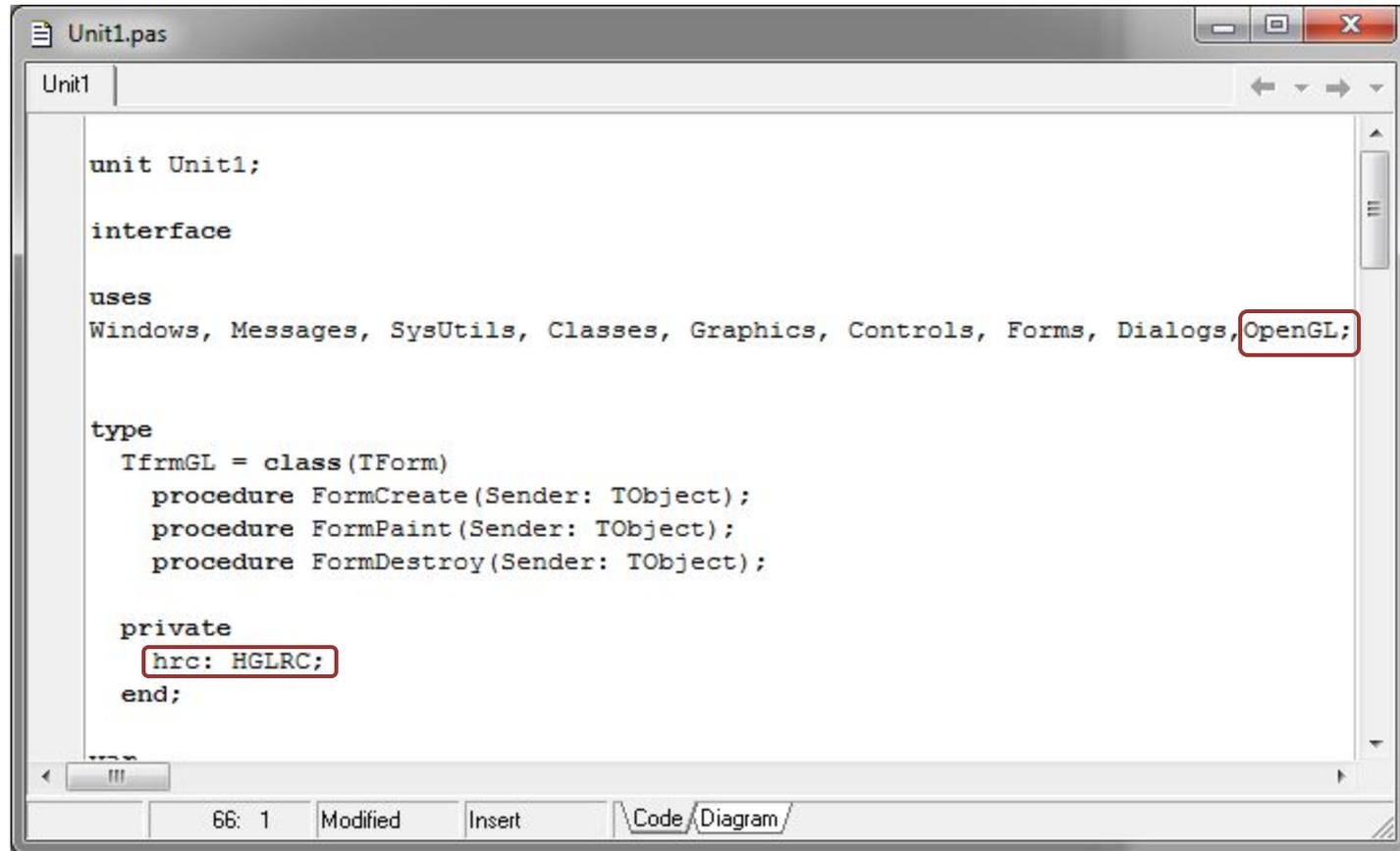
Панель элементов

Поиск по панели элементов

руктор] X

[- [X

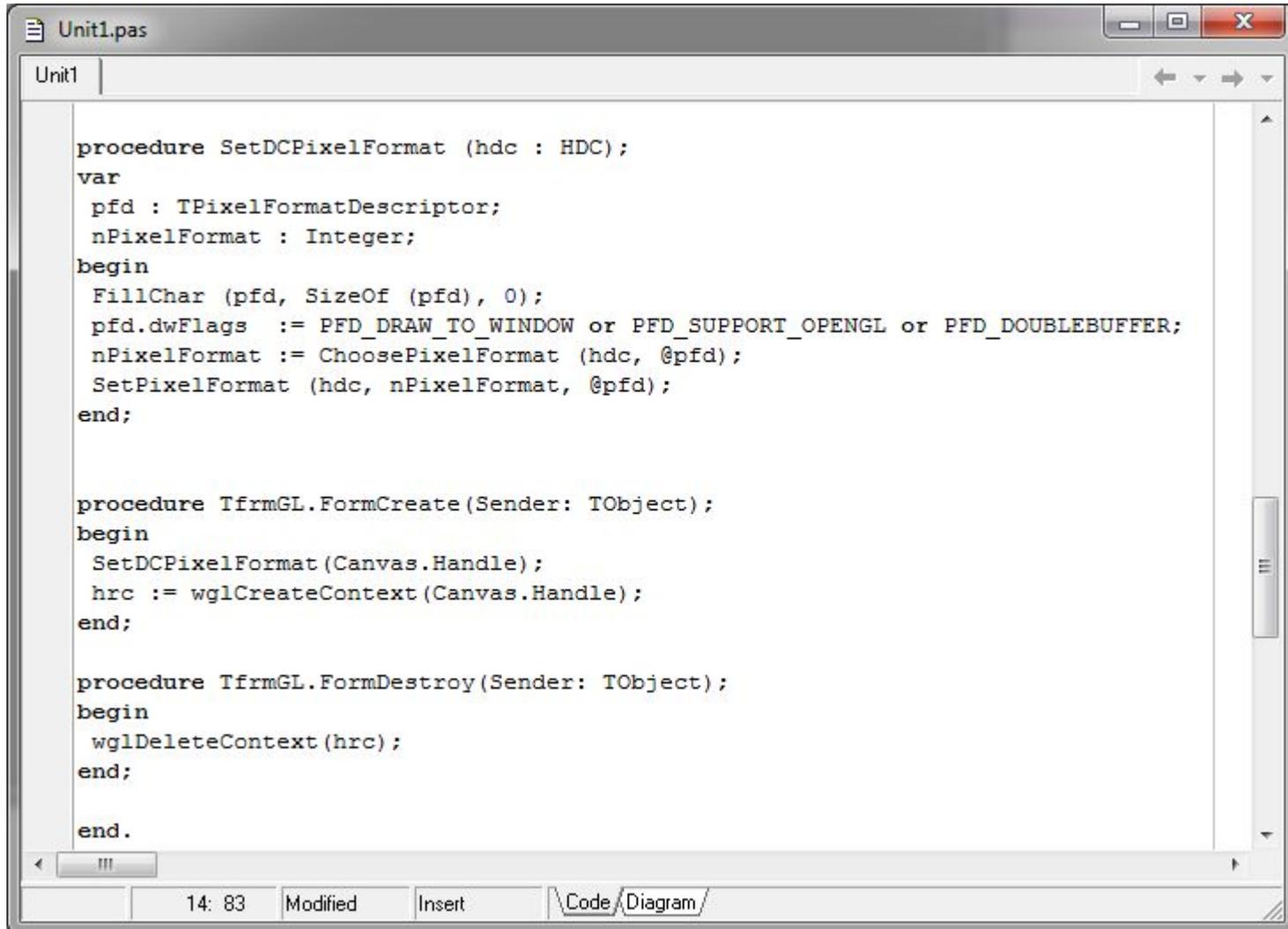
# OpenGL + Delphi (Object Pascal)



```
Unit1.pas  
Unit1  
  
unit Unit1;  
  
interface  
  
uses  
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, OpenGL;  
  
type  
  TfrmGL = class(TForm)  
    procedure FormCreate(Sender: TObject);  
    procedure FormPaint(Sender: TObject);  
    procedure FormDestroy(Sender: TObject);  
  
  private  
    hrc: HGLRC;  
  end;  
  
end;
```

66: 1 Modified Insert Code Diagram

# Визуализация двумерных объектов



```
Unit1.pas
Unit1

procedure SetDCPixelFormat (hdc : HDC);
var
  pfd : TPixelFormatDescriptor;
  nPixelFormat : Integer;
begin
  FillChar (pfd, SizeOf (pfd), 0);
  pfd.dwFlags := PFD_DRAW_TO_WINDOW or PFD_SUPPORT_OPENGL or PFD_DOUBLEBUFFER;
  nPixelFormat := ChoosePixelFormat (hdc, @pfd);
  SetPixelFormat (hdc, nPixelFormat, @pfd);
end;

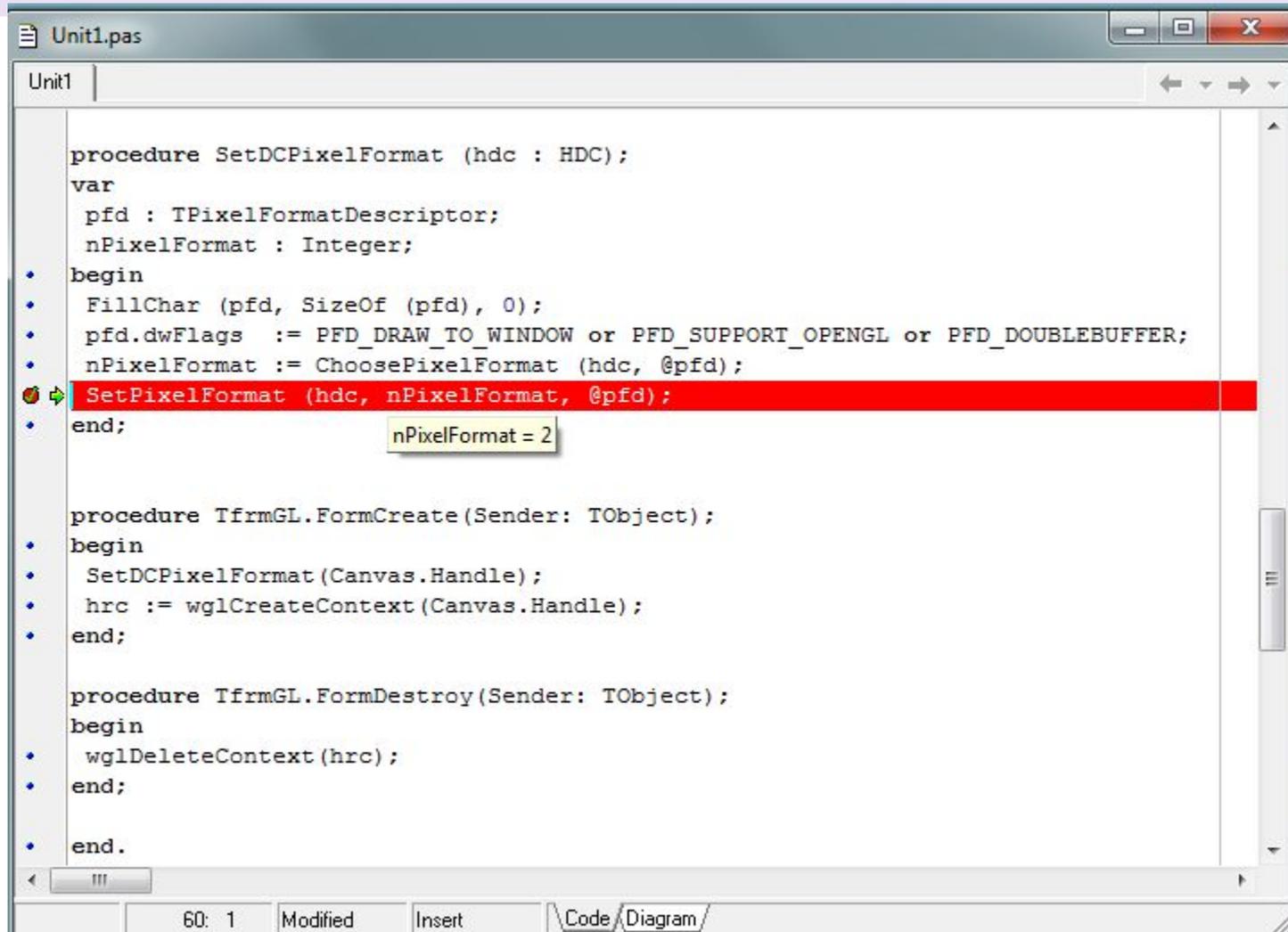
procedure TfrmGL.FormCreate(Sender: TObject);
begin
  SetDCPixelFormat(Canvas.Handle);
  hrc := wglCreateContext(Canvas.Handle);
end;

procedure TfrmGL.FormDestroy(Sender: TObject);
begin
  wglDeleteContext(hrc);
end;

end.
```

14: 83 Modified Insert Code/Diagram

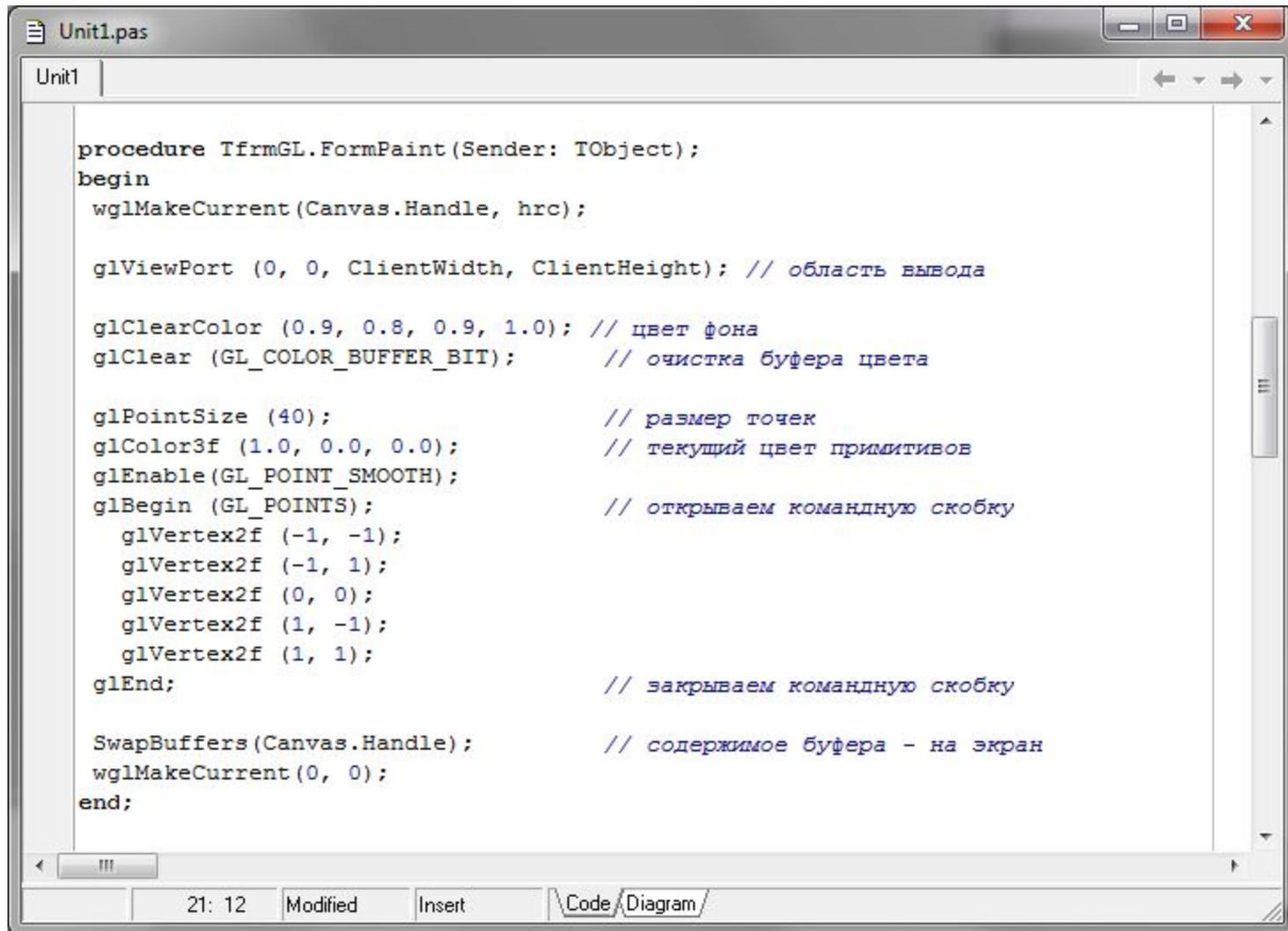
# Визуализация двумерных объектов



```
Unit1.pas  
Unit1  
procedure SetDCPixelFormat (hdc : HDC);  
var  
  pfd : TPixelFormatDescriptor;  
  nPixelFormat : Integer;  
begin  
  FillChar (pfd, SizeOf (pfd), 0);  
  pfd.dwFlags := PFD_DRAW_TO_WINDOW or PFD_SUPPORT_OPENGL or PFD_DOUBLEBUFFER;  
  nPixelFormat := ChoosePixelFormat (hdc, @pfd);  
  SetPixelFormat (hdc, nPixelFormat, @pfd);  
end;  
nPixelFormat = 2  
procedure TfrmGL.FormCreate (Sender: TObject);  
begin  
  SetDCPixelFormat (Canvas.Handle);  
  hrc := wglCreateContext (Canvas.Handle);  
end;  
procedure TfrmGL.FormDestroy (Sender: TObject);  
begin  
  wglDeleteContext (hrc);  
end;  
end.
```

60: 1 Modified Insert Code/Diagram

# Визуализация двумерных объектов



```
Unit1.pas
Unit1

procedure TfrmGL.FormPaint(Sender: TObject);
begin
  wglMakeCurrent(Canvas.Handle, hrc);

  glViewport (0, 0, ClientWidth, ClientHeight); // область вывода

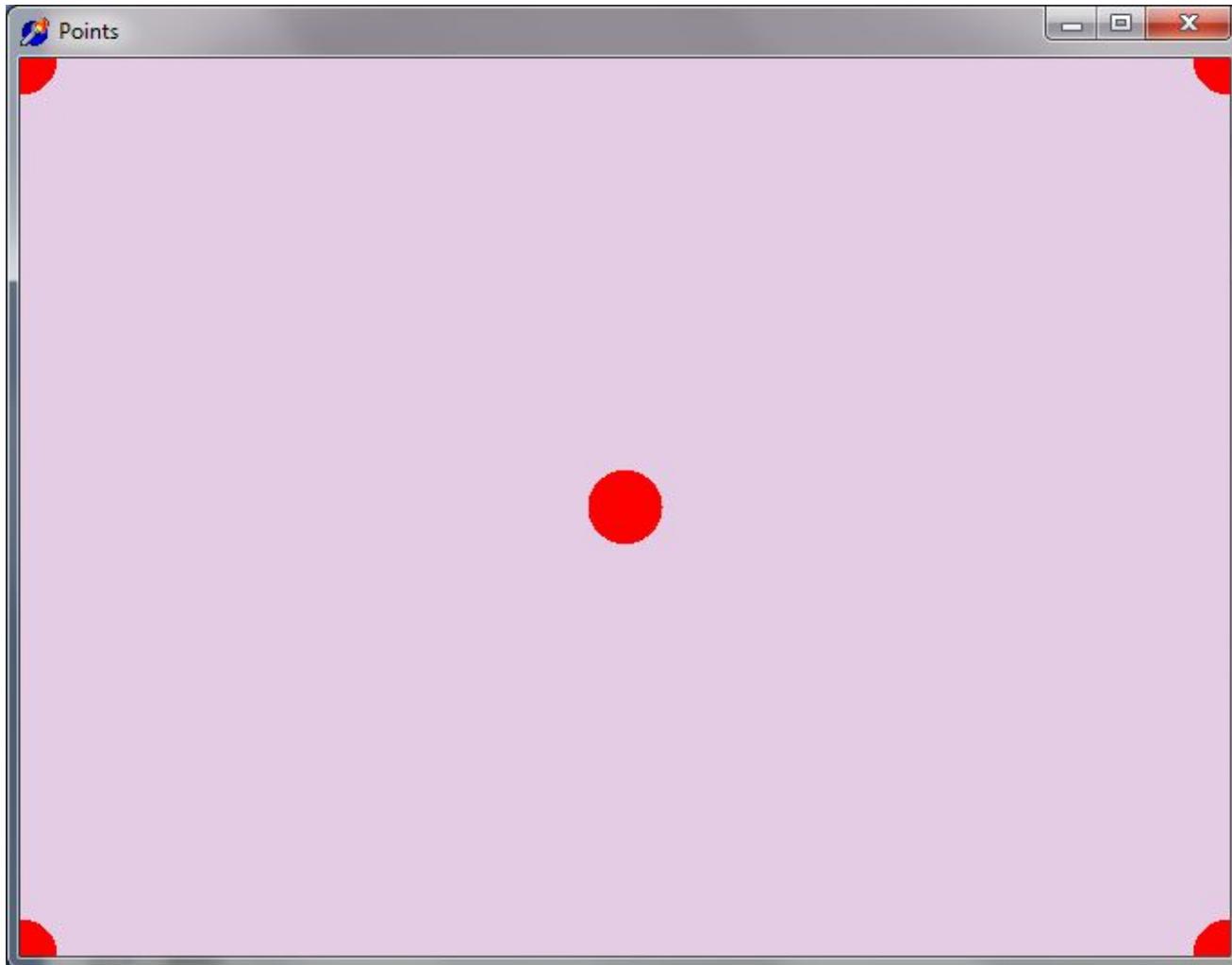
  glClearColor (0.9, 0.8, 0.9, 1.0); // цвет фона
  glClear (GL_COLOR_BUFFER_BIT);      // очистка буфера цвета

  glPointSize (40);                   // размер точек
  glColor3f (1.0, 0.0, 0.0);          // текущий цвет примитивов
  glEnable (GL_POINT_SMOOTH);
  glBegin (GL_POINTS);                // открываем командную скобку
  glVertex2f (-1, -1);
  glVertex2f (-1, 1);
  glVertex2f (0, 0);
  glVertex2f (1, -1);
  glVertex2f (1, 1);
  glEnd;                               // закрываем командную скобку

  SwapBuffers(Canvas.Handle);          // содержимое буфера - на экран
  wglMakeCurrent(0, 0);
end;
```

21: 12 Modified Insert Code/Diagram/

# Визуализация двумерных объектов



Form1

Тип

- Points
- Lines
- LINE\_STRIP
- LINE\_LOOP

Список координат

```
-0.74 -0.68
-0.15 0.77
0.59 -0.63
-0.91 0.21
0.54 0.26
```

Clear

Form1

Тип

- Points
- Lines
- LINE\_STRIP
- LINE\_LOOP

Список координат

```
-0.74 -0.68
-0.15 0.77
0.59 -0.63
-0.91 0.21
0.54 0.26
```

Clear

Form1

Тип

- Points
- Lines
- LINE\_STRIP
- LINE\_LOOP

Список координат

```
-0.74 -0.68
-0.15 0.77
0.59 -0.63
-0.91 0.21
0.54 0.26
```

Clear

Form1

Тип

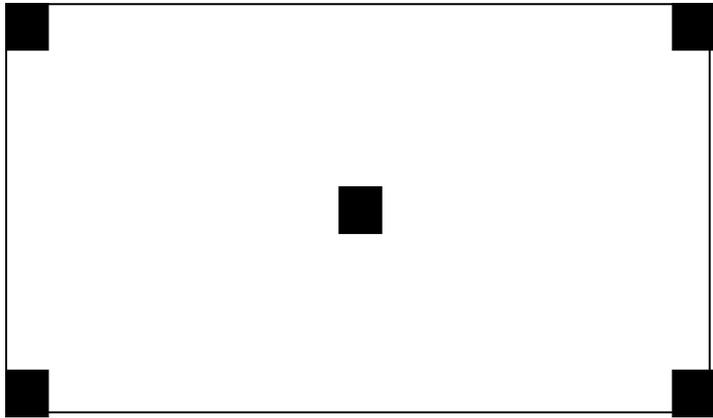
- Points
- Lines
- LINE\_STRIP
- LINE\_LOOP

Список координат

```
-0.74 -0.68
-0.15 0.77
0.59 -0.63
-0.91 0.21
0.54 0.26
```

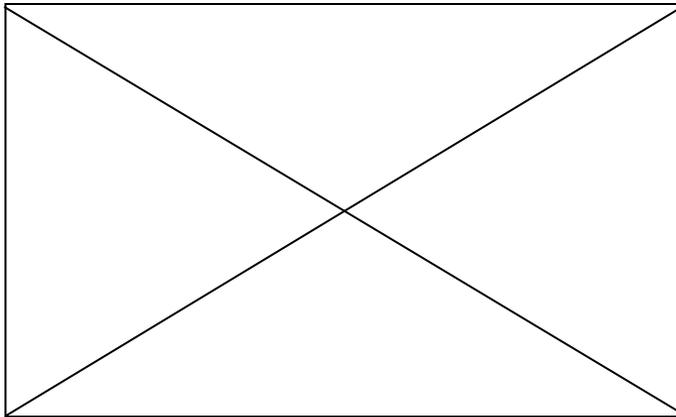
Clear

# GL\_POINTS



```
glPointSize (20);  
glColor3f  (1.0, 1.0,  
1.0);  
glBegin (GL_POINTS);  
    glVertex2f (-1, -1);  
glVertex2f (-1, 1);  
glVertex2f (0, 0);  
glVertex2f (1, -1);  
glVertex2f (1, 1);  
glEnd;
```

# GL\_LINES



```
glLineWidth (2.5);  
glBegin (GL_LINES);  
    glVertex2f (-1, -1);  
    glVertex2f (1, 1);  
    glVertex2f (-1, 1);  
    glVertex2f (1, -1);  
glEnd;
```

# СОБЫТИЕ, СООБЩЕНИЕ, КОНТЕКСТ

```
procedure TfmTest.Button1Click(Sender: TObject);  
var  H : HWND;  
begin  
  H := FindWindow('TForm1', 'Form1');  
  If H <> 0 then ShowMessage('Есть Form1!')  
    else ShowMessage('Нет Form1!')  
end;
```

**Все окна при своем создании регистрируются в операционной системе и получают уникальный идентификатор, называемый "ссылка на окно".  
Тип этой величины в Delphi – HWND**

**(HWND - WiNDow Handle - ссылка на окно)**

# ССЫЛКИ НА КОНТЕКСТ УСТРОЙСТВА И ВОСПРОИЗВЕДЕНИЯ

**var**

**dc : HDC; { ссылка на контекст устройства }**

**hrc: HGLRC { ссылка на контекст воспроизведения }**

**Контекст устройства** - величина типа HDC (функция GetDC).  
Ссылке на контекст устройства в Delphi соответствует свойство Canvas.Handle формы, принтера и некоторых компонентов (вместо DC можно использовать Canvas.Handle) .

**Графическая система OpenGL**, как и любое другое приложение Windows, также нуждается в ссылке на окно, на котором будет осуществляться воспроизведение – специальной *ссылке на контекст воспроизведения* – величина типа HGLRC (Handle openGL Rendering Context, ссылка на контекст воспроизведения OpenGL).

# КОНТЕКСТ

Приложению для того, чтобы воспользоваться функциями воспроизведения Windows, необходимо только указать ссылку на контекст устройства, содержащую средства и характеристики устройства вывода.

Win32 Programmer's Reference фирмы MicroSoft о контексте устройства сообщает следующее: *"Контекст устройства является структурой, которая определяет комплект графических объектов и связанных с ними атрибутов, и графические режимы, влияющие на вывод. Графический объект включает карандаш для изображения линии, кисть для краски и заполнения, растр для копирования или прокрутки частей экрана, палитру для определения комплекта доступных цветов, области для отсечения и других операций, и маршрута для операций рисования"*.

# ФОРМАТ ПИКСЕЛЯ

Прежде чем получить контекст воспроизведения, сервер OpenGL должен получить детальные характеристики используемого оборудования. Эти характеристики хранятся в специальной структуре, тип которой - **TPixelFormatDescriptor** (описание формата пикселя). Формат пикселя определяет конфигурацию буфера цвета и вспомогательных буферов.

**PFD\_DRAW\_TO\_WINDOW**

**PFD\_SUPPORT\_OPENGL**

**PFD\_DOUBLEBUFFER**

# МИНИМАЛЬНАЯ ПРОГРАММА

Uses Windows, Messages, SysUtils, Classes,  
Graphics, Controls, Forms, Dialogs, OpenGL;

.....

Private hrc: HGLRC;

# ФОРМАТ ПИКСЕЛЯ

```
procedure SetDCPixelFormat (hdc : HDC);
var
  pfd : TPixelFormatDescriptor;
  nPixelFormat : Integer;
begin
  FillChar (pfd, SizeOf (pfd), 0);
  pfd.dwFlags := PFD_DRAW_TO_WINDOW or
PFD_SUPPORT_OPENGL or PFD_DOUBLEBUFFER;
  nPixelFormat := ChoosePixelFormat (hdc, @pfd);
  SetPixelFormat (hdc, nPixelFormat, @pfd);
end;
```

# ПРОРИСОВКА ОКНА

```
procedure TForm1.FormPaint(Sender: TObject);
begin
    wglMakeCurrent(Canvas.Handle, hrc); // устанавливает текущий
        контекст воспроизведения
    glViewport (0, 0, ClientWidth, ClientHeight); // область вывода
    glClearColor (0.5, 0.5, 0.75, 1.0); // цвет фона
    glClear (GL_COLOR_BUFFER_BIT); // очистка буфера цвета
    glPointSize (20); // размер точек
    glColor3f (1.0, 0.0, 0.5); // текущий цвет примитивов
    glBegin (GL_POINTS); // открываем командную скобку
        glVertex2f (-1, -1);
        .....glEnd; // закрываем командную скобку
    SwapBuffers(Canvas.Handle); // содержимое буфера - на экран
    wglMakeCurrent(0, 0);
end;
```

# СОЗДАНИЕ ФОРМЫ

```
procedure TfrmGL.FormCreate(Sender: TObject);  
begin  
  SetDCPixelFormat(Canvas.Handle);  
  hrc := wglCreateContext(Canvas.Handle);  
end;
```

# КОНЕЦ РАБОТЫ ПРИЛОЖЕНИЯ

```
procedure TfrmGL.FormDestroy(Sender: TObject);  
begin  
    wglDeleteContext(hrc);  
end;
```