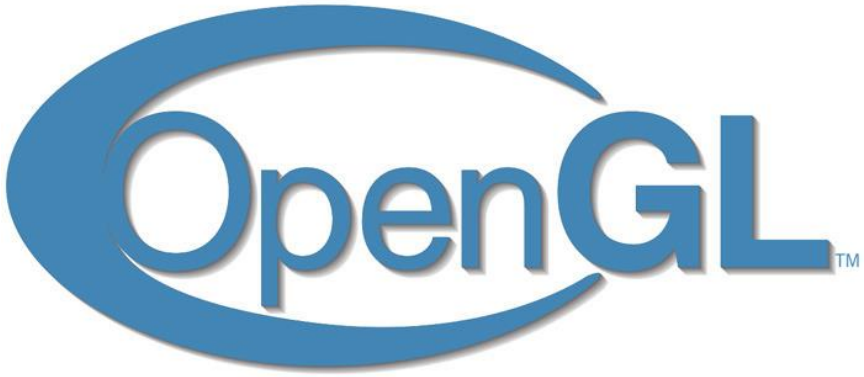

OpenGL
TM

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.
- http://www.opengl.org

OpenGL (Open Graphics Library – открытая графическая библиотека) -спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двухмерную и трёхмерную компьютерную графику.

Преимущества

Аналогичные библиотеки:
DirectX (Direct3D), Vulkan, Java 3D

OpenGL

- Стабильность (с 1992 г.)
- Производители оборудования создают реализации библиотеки согласно этой спецификации (Nvidia, AMD/ATi)
- Переносимость
  - Независимость от оконной и операционной системы
- Легкость применения
  - Простой интерфейс, реализации для различных ЯП
  - Низкие затраты на обучение

Основной особенностью OpenGL можно считать простоту. Ядро OpenGL контролирует процесс обработки примитивов.

Для передачи данных используется процедурная модель.

В каждый момент времени состояние OpenGL определяется через набор переменных, задающих параметры обработки. Каждый новый переданный треугольник проходит обработку в соответствии с текущим состоянием.

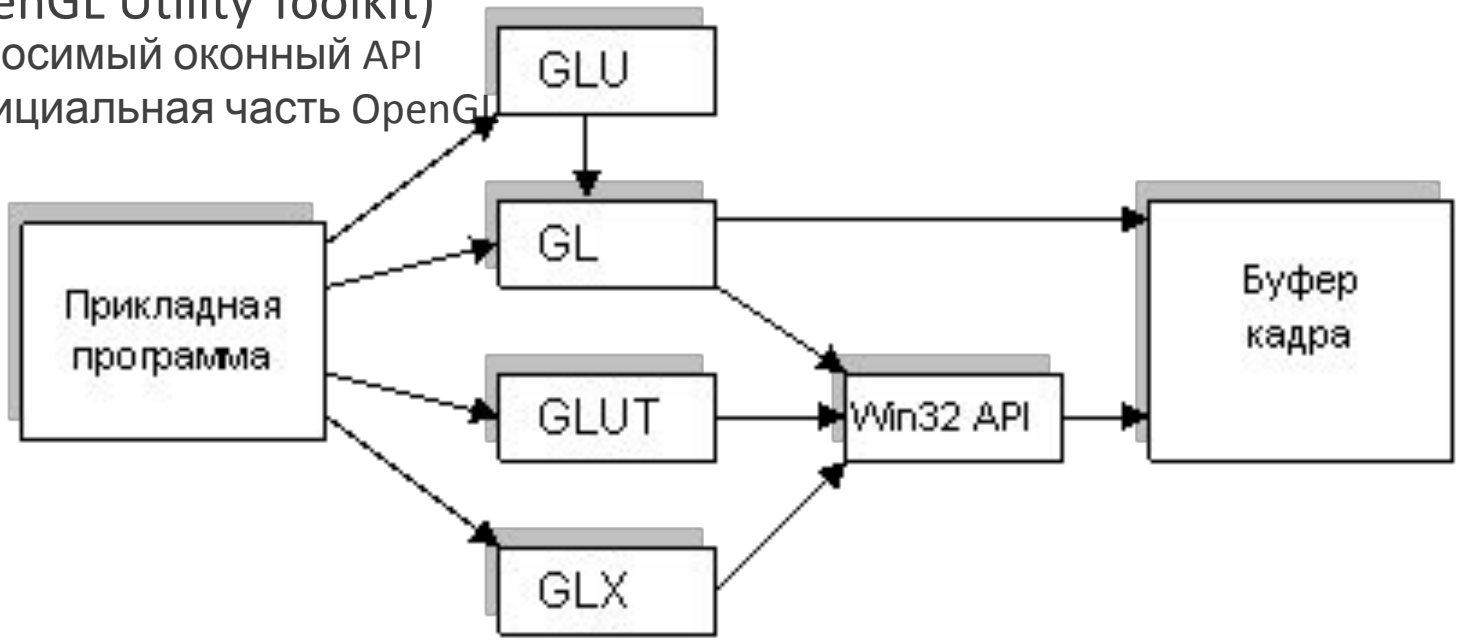
Состоит из набора библиотек

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

```
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h>
```



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

Как рисовать объекты с помощью OpenGL?

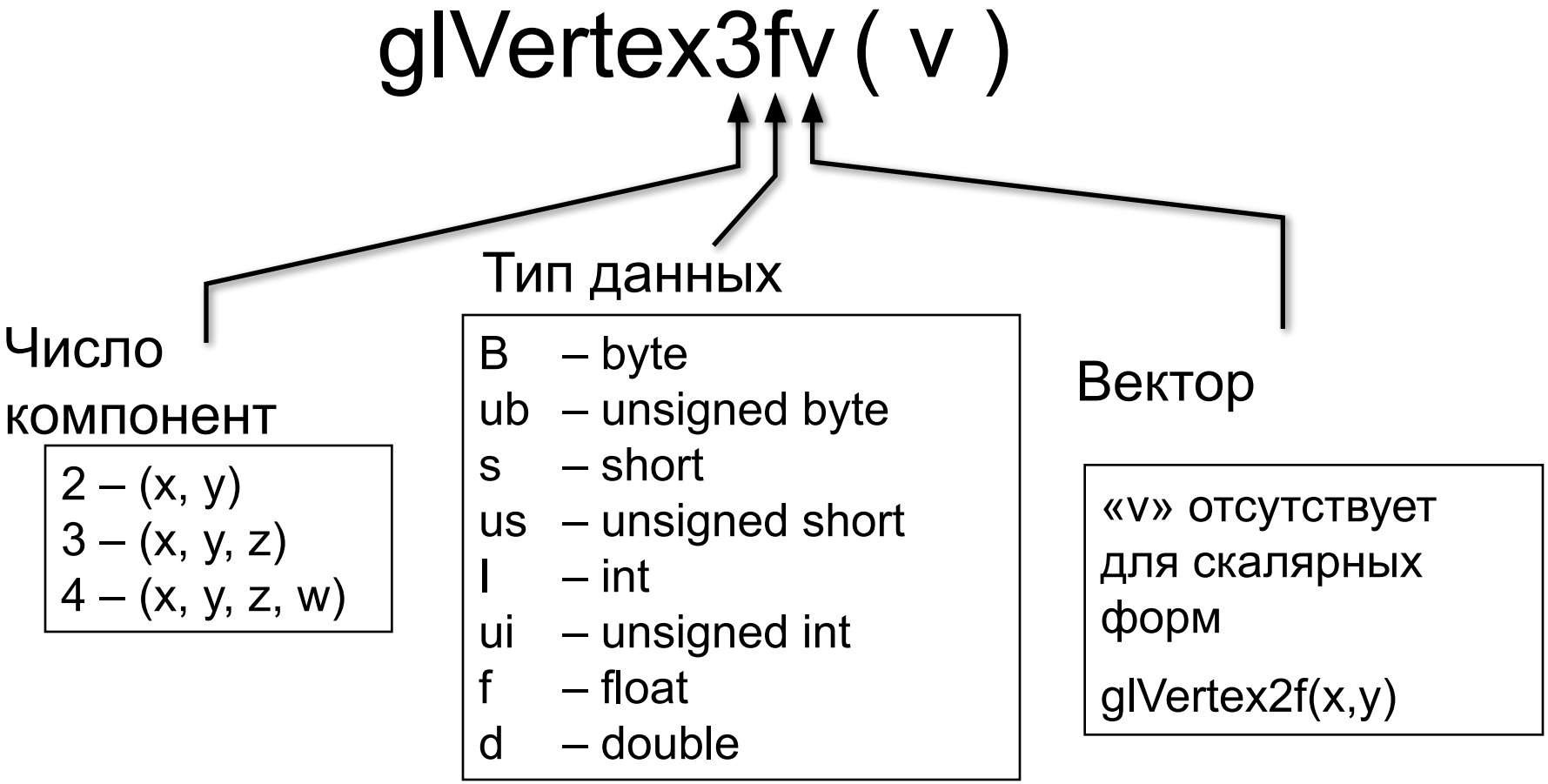
Объекты на экране рисуются путем последовательной передачи в конвейер вершин примитивов, которые составляют объект

- команды передача данных

Обработка данных на каждом этапе конвейера может быть настроена через

- команды изменения состояния

Команды OpenGL



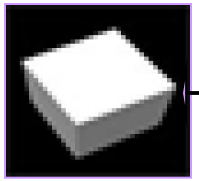
Модель begin/end

void glMatrixMode(…);
void glLoadIdentity();
void glMultMatrixd(…);

void glBegin(GLenum type);
void glVertex(…);
void glNormal(…);
void glColor(…);
void glEnd();

T&L

Rasterization



Pixel Ops

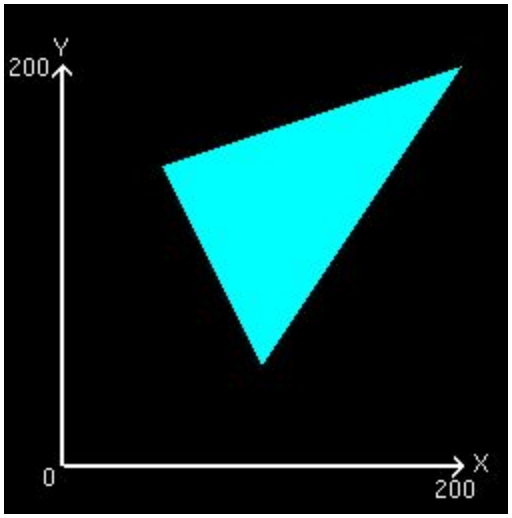
void glTexture2d(…);
void glTexEnv(…);
void glPolygonMode(…);

void glDepthFunc(…);
void glBlendFunc(…);
void glStencilOp(…);

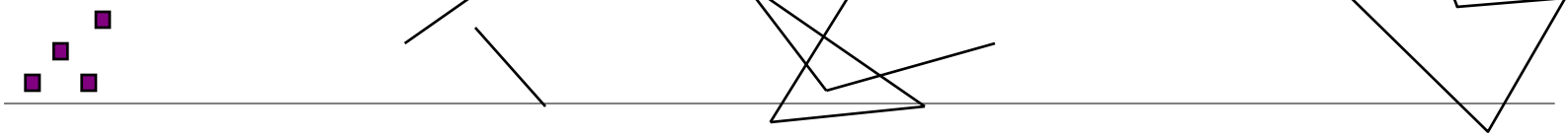
Пример программы

Цветной треугольник

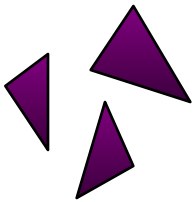
```
glBegin(GL_TRIANGLES);
   glColor2f(0.0f,1.0f);
   glVertex2f(150.0f, 50 .0f);
   glVertex2f(50.0f, 150 .0f);
   glVertex2f(200 .0f, 200 .0f);
glEnd();
```



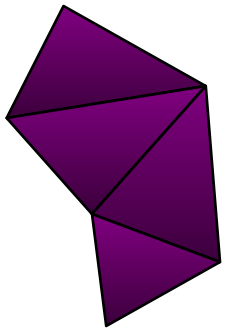
Типы примитивов OpenGL



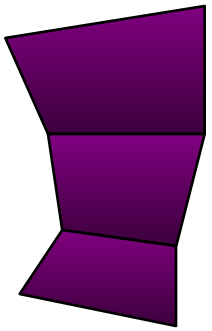
GL_POINTS GL_LINES GL_LINE_STRIP GL_LINE_LOOP



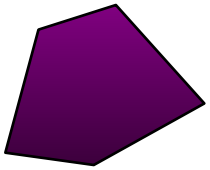
GL_TRIANGLES



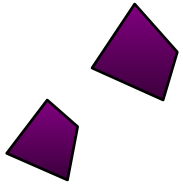
GL_TRIANGLE_STRIP



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*

Сложные фигуры

void **gluSphere** (GLUquadricObj * *qobj*, GLdouble *radius*,

GLint *slices*, GLint *stacks*)

---

void **gluCylinder** (GLUquadricObj * *qobj*,

GLdouble *baseRadius*,

GLdouble *topRadius*,

GLdouble *height*, GLint *slices*,

GLint *stacks*)

void **gluDisk**(GLUquadric* quad,

GLdouble inner,

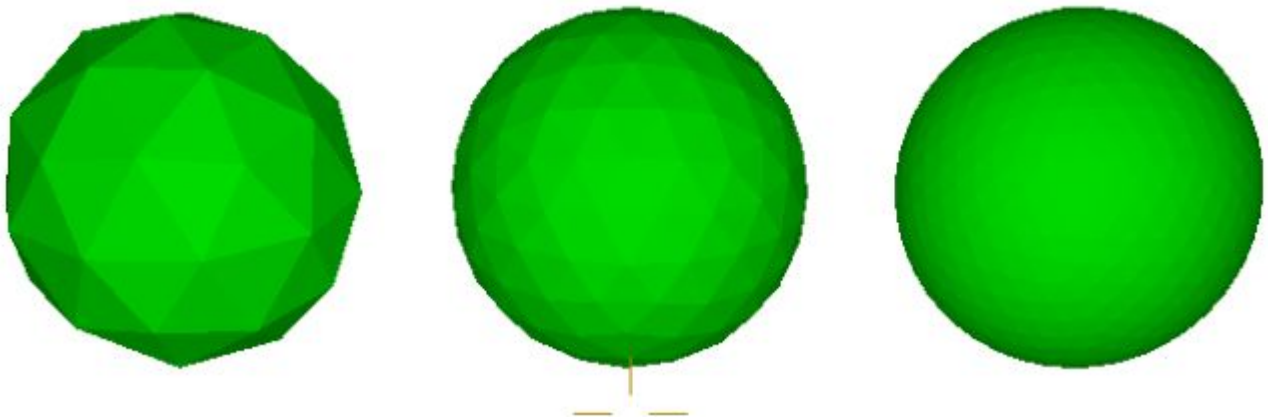
GLdouble outer,

GLint slices,

GLint loops)

параметры ***slices, stacks*** задают число разбиений

gluSphere()



Преобразования координат в OpenGL

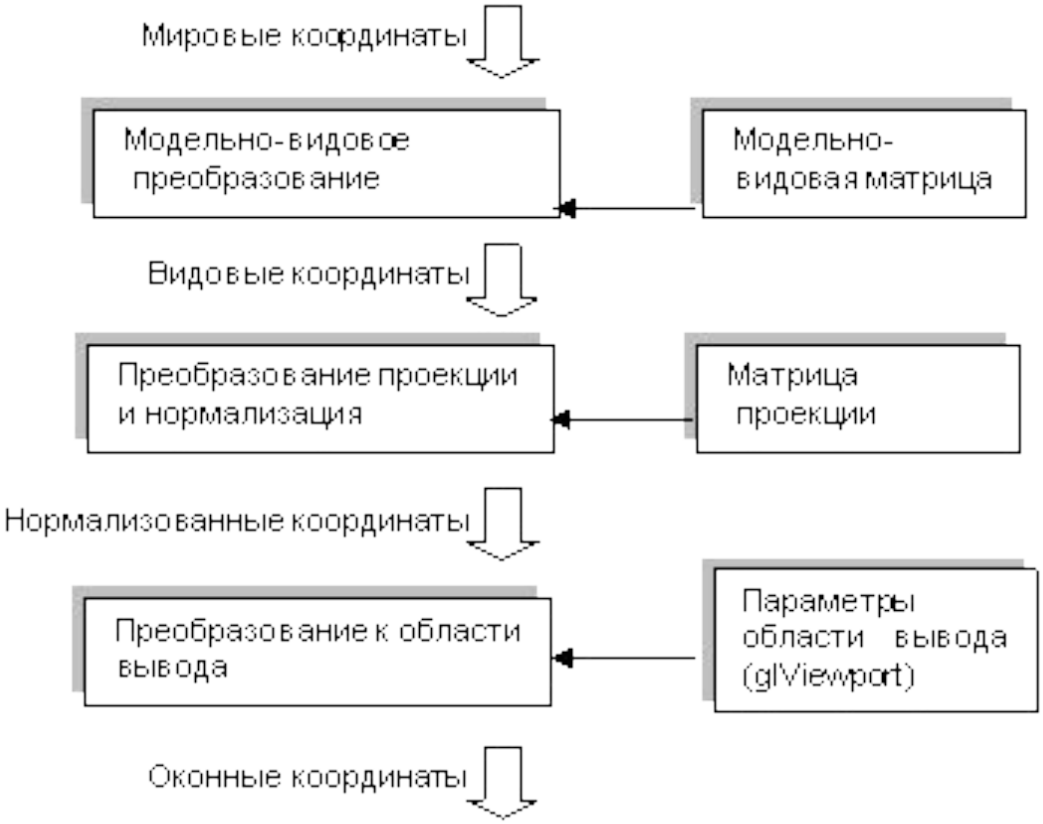
Каждая вершина объекта задается в *локальных координатах модели*

Необходимо определить набор *геометрических преобразований,* таких, что каждая вершина преобразуется в точку на плоскости экрана

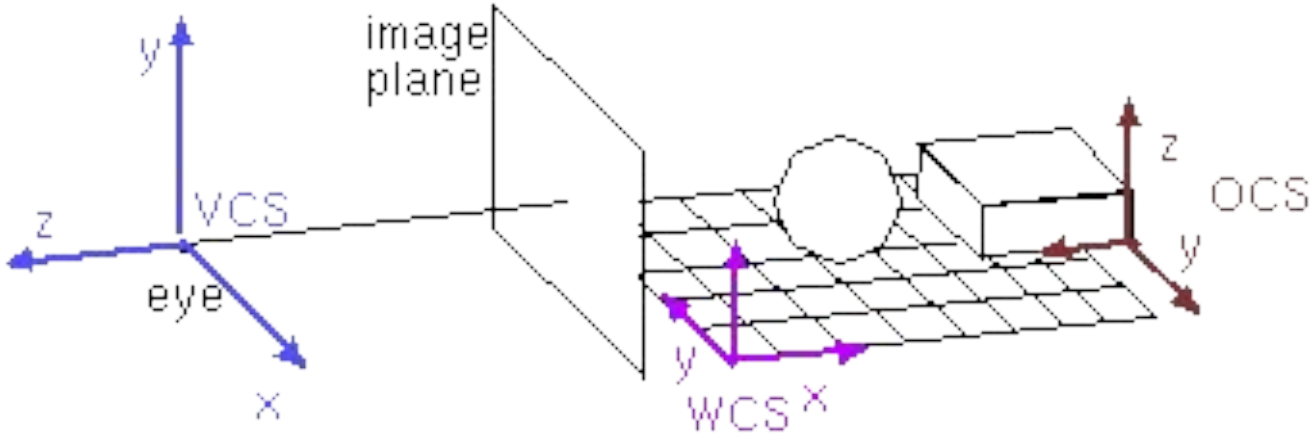
Три последовательных преобразования:

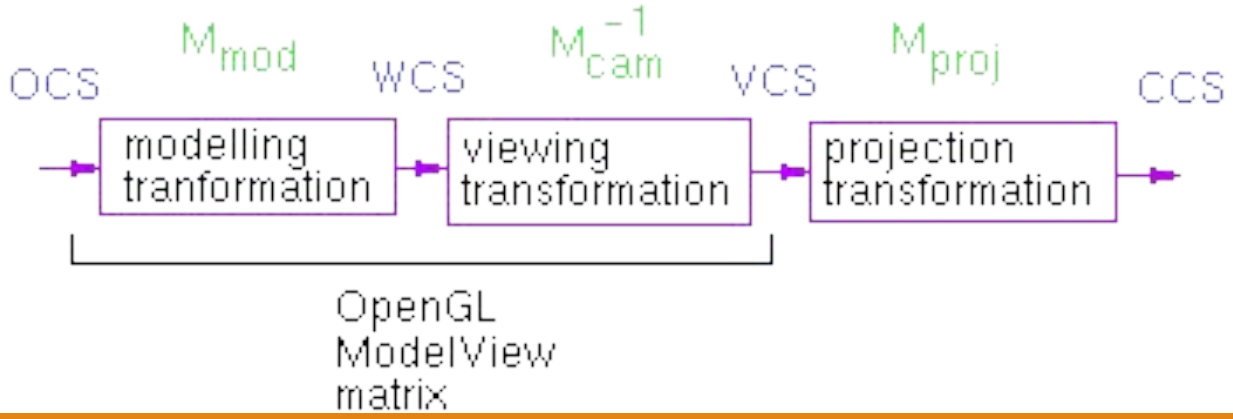
- модельное преобразование
- видовое преобразование
- проективное преобразование

Последовательность преобразований



Графический конвейер





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]);
```

$$M = M \cdot \begin{bmatrix} c[0] & c[4] & c[8] & c[12] \\ c[1] & c[5] & c[9] & c[13] \\ c[2] & c[6] & c[10] & c[14] \\ c[3] & c[7] & c[11] & c[15] \end{bmatrix}$$

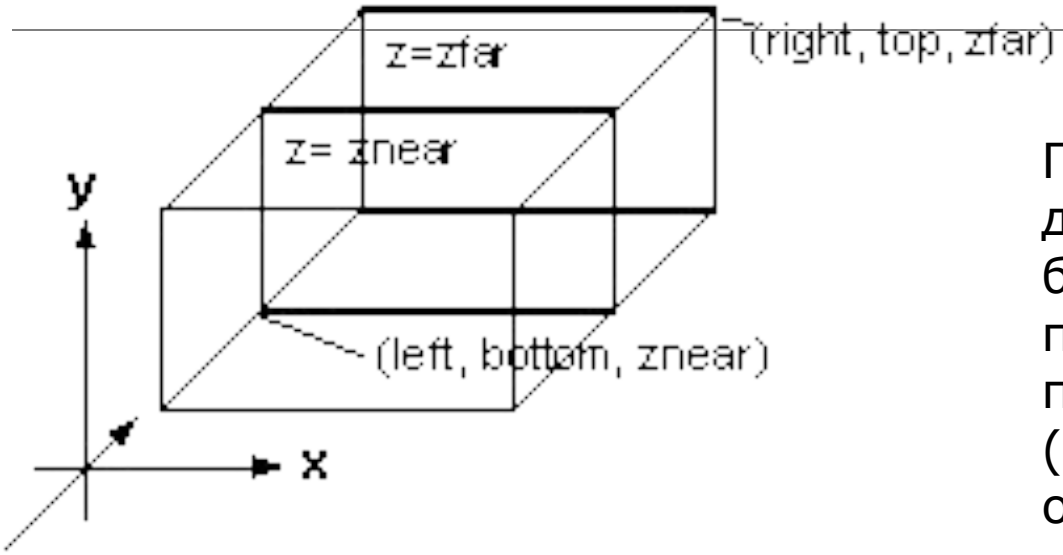
Модельно-Видовые преобразования

```
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);
```

Проекции. Параллельная



Параметры *near* и *far* задают расстояние до ближней и дальней плоскостей отсечения по удалению от точки (0,0,0) и могут быть отрицательными

void **glOrtho** (GLdouble left, GLdouble right,

GLdouble bottom, GLdouble top,

GLdouble near, GLdouble far)

void **gluOrtho2D** (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)

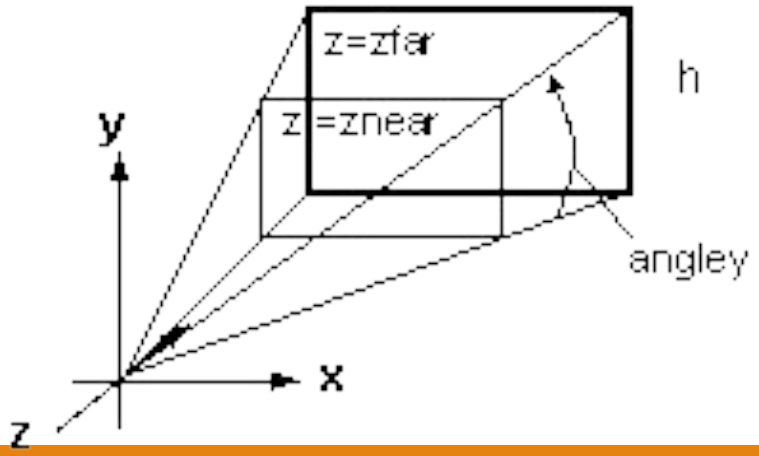
Перспективная проекция

void **gluPerspective** (angley, aspect, znear, zfar)

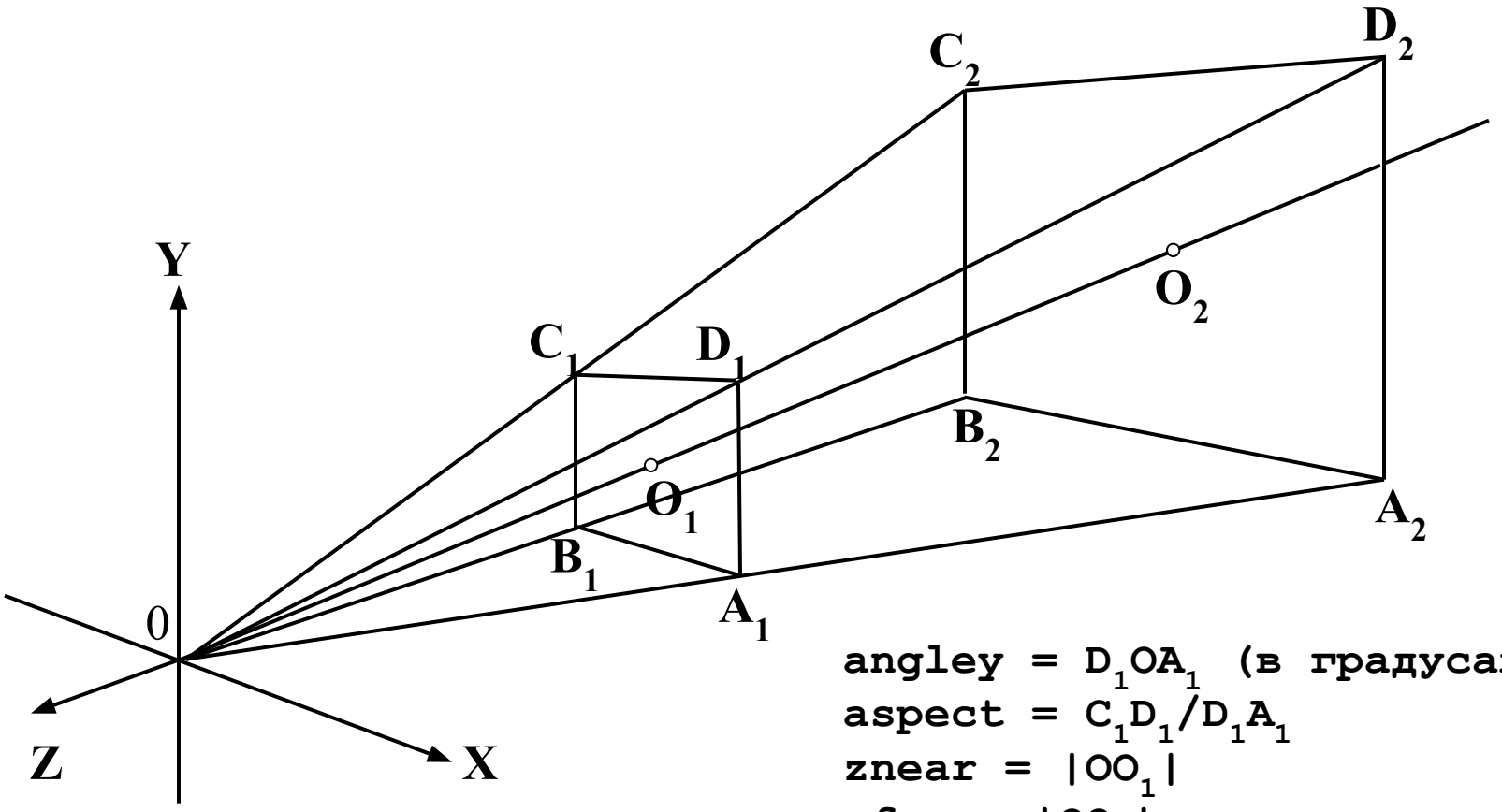
Параметр *angley* определяет угол видимости в градусах по оси y и должен находиться в диапазоне от 0 до 180.

Угол видимости вдоль оси x задается параметром *aspect*, который обычно задается как отношение сторон области вывода

Параметры *zfar* и *znear* задают расстояние от наблюдателя до плоскостей отсечения по глубине и должны быть положительными



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



angley = D_1OA_1 (в градусах)
aspect = C_1D_1/D_1A_1
znear = $|OO_1|$
zfar = $|OO_2|$

Видовое преобразование

- Настройка виртуальной камеры

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

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

Источники света

void **glLight[i f]** (GLenum light, GLenum pname,

GLfloat param)

void **glLight[i f]** (GLenum light, GLenum pname,

GLfloat *params)

light однозначно определяет источник света от 0 до 8

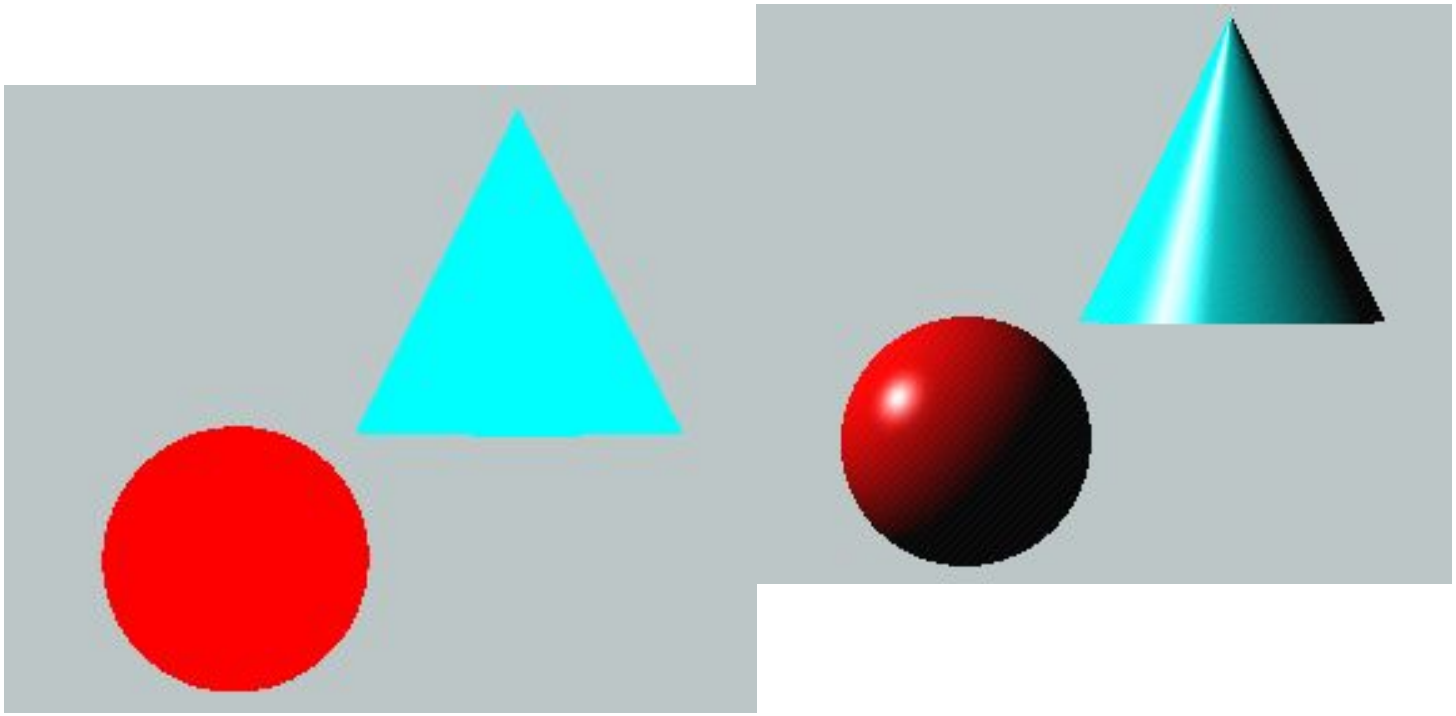
Определение свойств материала объекта имеет смысл, только если в сцене есть источники света. Иначе все объекты будут черными (или, строго говоря, иметь цвет, равный рассеянному цвету материала

Виды материалов

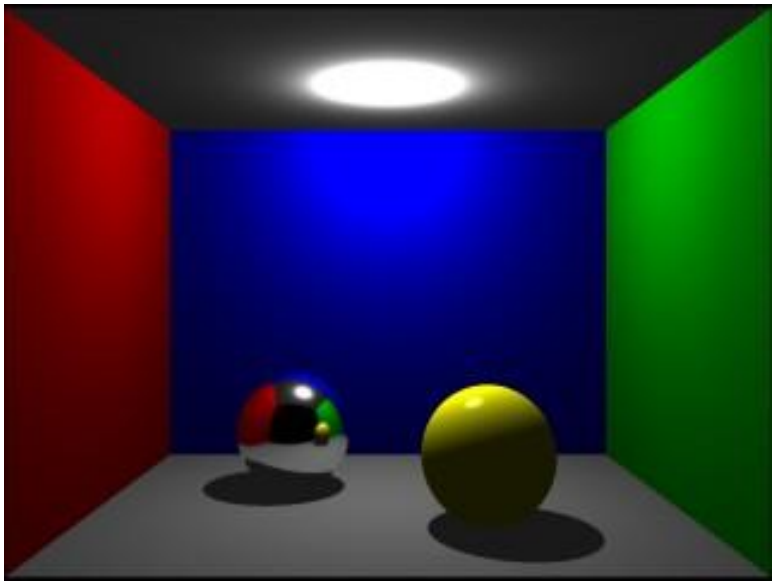
Параметры *pname:*

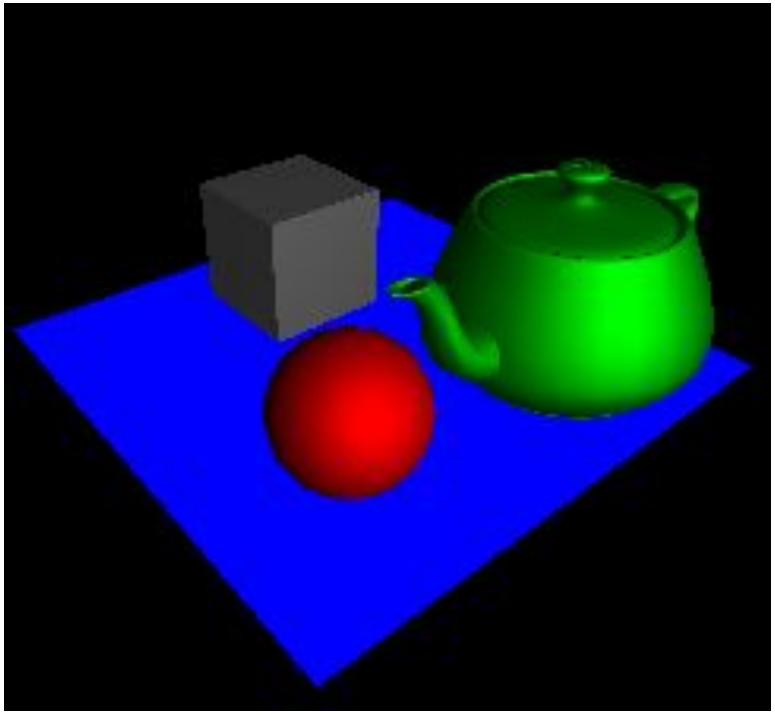
GL_SPOT_EXPONENT	описывает уровень сфокусированный источник света.
GL_SPOT_CUTOFF	конусовидный световой поток создаваемый источником
GL_AMBIENT	фоновое освещение
GL_DIFFUSE	диффузное освещение
GL_SPECULAR	цвет зеркального отражения

Для использования освещения сначала надо установить соответствующий режим вызовом команды glEnable(GL_LIGHTNING), а затем включить нужный источник командой glEnable(GL_LIGHTi).



Фотореализм






NVIDIA.
© NVIDIA CORPORATION 2002


ATi RADEON 9700

DirectX

DirectX - набор API, разработанных для решения задач, связанных с программированием под Microsoft Windows. Наиболее широко используется при написании компьютерных игр.

DirectX состоит из:

- Direct3D (D3D): интерфейс вывода трёхмерных примитивов.
- DirectInput: интерфейс, используемый для обработки данных, поступающих с клавиатуры, мыши, джойстика и пр. игровых контроллеров.
- DirectSound: интерфейс низкоуровневой работы со звуком (формата Wave) Direct2D : интерфейс вывода двухмерной графики

Структура DirectX значительно отличается от OpenGL. DirectX основан на модели COM (Component Object Model). Это означает, что в отличие от простого вызова функций эта модель предполагает выполнение некоторых дополнительных действий, связанных с компонентной архитектурой DirectX.

Vulkan API

Vulkan API — это низкоуровневые кроссплатформенные программные интерфейсы, которые выдают более высокую производительность 3D-графики за счёт снижения издержек по сравнению с другими API типа OpenGL, особенно при наличии специальных функций GPU (API подходит также для рендеринга 2D).

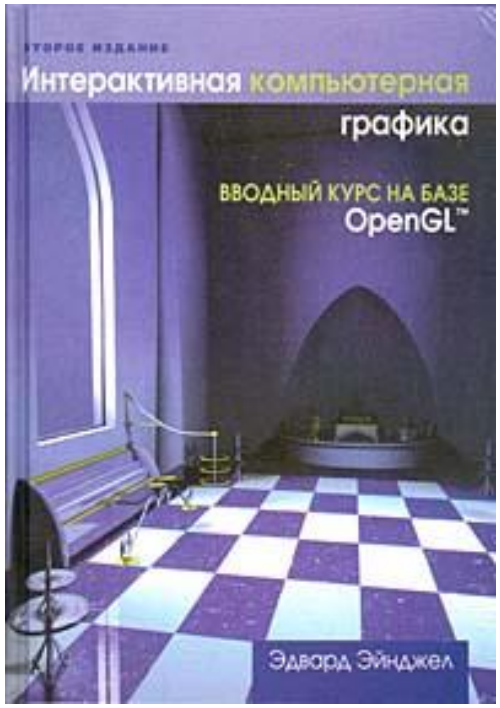
При грамотной реализации Vulkan обеспечивает «от маргинального до полиномиального повышения скорости по сравнению с другими API на том же оборудовании».

Разработчик Khronos Group представил Vulkan API в рамках конференции GDC 2015, а первый релиз состоялся в феврале 2016 года. Первоначально Vulkan API носили название «следующее поколение OpenGL» или просто glNext, но потом название сменили на Vulkan.

Эта технология сильно ускоряет 3D-графику и снижает нагрузку на процессор по примеру Direct3D. Но проблема в том, что Vulkan API требует от разработчика гораздо больше усилий, в то время как API более высокого уровня, такие как OpenGL и DirectX упрощают многие вещи, например, управление памятью.

Литература

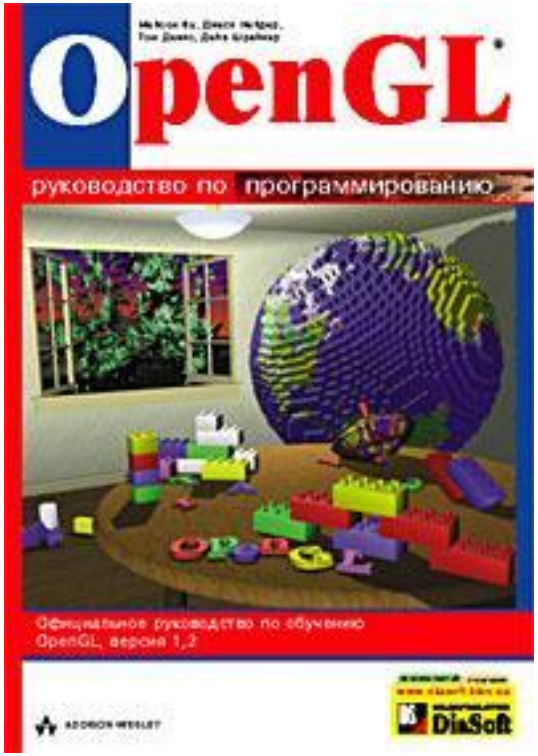
Ю. Тихомиров. OpenGL. Программирование трехмерной графики, БХВ – Петербург, 2002



Эдвард Энджел. Интерактивная компьютерная графика. Вводный курс на базе OpenGL, 2-е изд., Вильямс, 2001

Литература

Ву Мейсон, Нейдер Джеки, Девис Том, Шрайнер Дейв. OpenGL. Руководство по программиста. Диа-Софт, 2002.



Френсис Хилл. OpenGL. Программирование компьютерной графики. Для профессионалов. Питер. 2002