1. Аппаратно-программные компоненты графической системы и структуры их взаимодействия. Конвейерная архитектура графических систем. Организация библиотеки OpenGL.

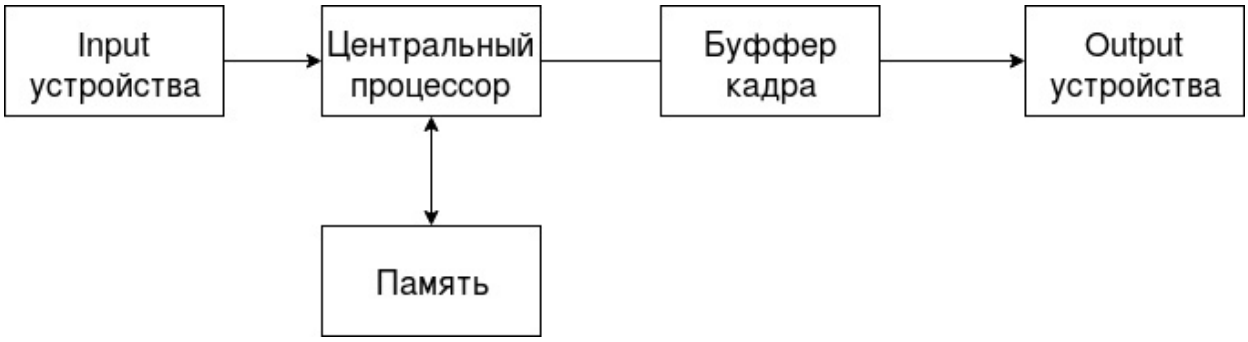
КГ - область информатики, в сферу интересов которой входят все аспекты формирования изображения.

Области применения:

1. моделирование
2. проектирование
3. GUI
4. представление

Аппаратные средства графической системы имеет следующую структуру:

Система КГ - вычислительная система общего назначения (с расширенной функциональностью отдельных компонент).



Буфер кадра и пиксели:
Все современные графические системы использую растровый способ создания изображения:

- изображение - массив точек (изображение дискретно);
  точки называются пикселями (px);
  растр - это массив точек;
- каждый пиксель имеет четко определенное положение на экране (пиксели локализованы)
- массив кодов зацветки пикселей хранится в буфере кадров (frame buffer)
- под код буфера кажда может выделяться отдельная память (VRAM || DRAM)

Характеристики буфера кадра:

1. глубина (depth) - количество бит, выделенных отдельному коду зацветки пикселя (как много информации о цвете пиксель может хранить)

1 бит	**2 цвета**
8 бит	2^8 цветов
16 бит	2^16 цветов
> 24 бит	true-color

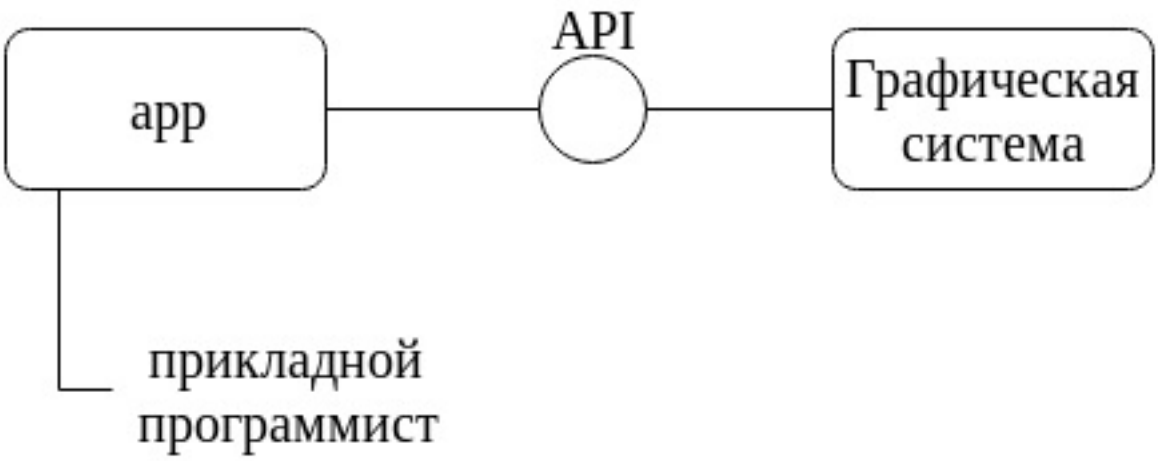
2. разрешающая способность (разрешение)

В простых графических системах используется один процессор, на который возлагаются все **задачи КГ:**

1. формирование примитивов (точки, линии, треугольники, ...)
2. проецирование (изначально x,y,z -> проецируется на плоский экран x,y)
3. растровое преобразование (перевод в коде зацветки пикселя)

Для ускорения процесса подзадачи 1-3 могут выносится в отдельный процессор.

Интерфейс между прикладной программой и графической системой — это множество функций, которые в совокупности образуют графическую библиотеку. Спецификация этих функций называется API – интерфейс прикладного программирования (application programming interface) – OpenGL, Direct3D.
Модель системы прикладного программирования показана схематически на рисунке:



API должен обеспечивать работу с:

- объектами
- наблюдателями (виртуальная камера)
- источники света
- материал объекта

Характеристики графического API:

1. Объекты описываются массивами вершин. Вершина - абстрактное вспомогальное понятие. Существует набор примитивов:
   - точки
   - отрезки
   - треугольники
   - многоугольники
   - сферы
   - и т.д.
2. Повороты, перемещение, масштабирование.
3. Источники света. Доступны разные типы - точечный, прожектор и фоновый. Существует возможность регулирования оптических свойств, яркости, интенсивности.
4. Существует возможность задания глянцевой или матовой поверхности, цвета и степени неровности.

Конвейерная архитектура (СБПС - плата). В данной архитектуре формирование изображения разбивается на четыре этапа:

1. геометрические преобразования - формирование *представления объектов* сцены в разных системах координа; используется аппарат линейной алгебры (матричной математики)

2. отсечение (clipping) - процесс происходит с использованием отсекающей прямоугольной рамки на плоскости проекции;

    - объекты, проекции которых попадают во внутреннюю часть отсекающей рамки - учавствуют в формировании изображения;
    - объекты, которые пересекают отсекающую рамку - учавствуют частично;
    - остальные отсекаются

3. проективные преобразования - переход с объемной сцены в плоскую, виды:

    - перспективная;
    - ортогональная;
    - косоугольная

4. растровые преобразования - преобразования описания двумерных объектов в коды зацветки пикселей **(px)** в буфере кадра.

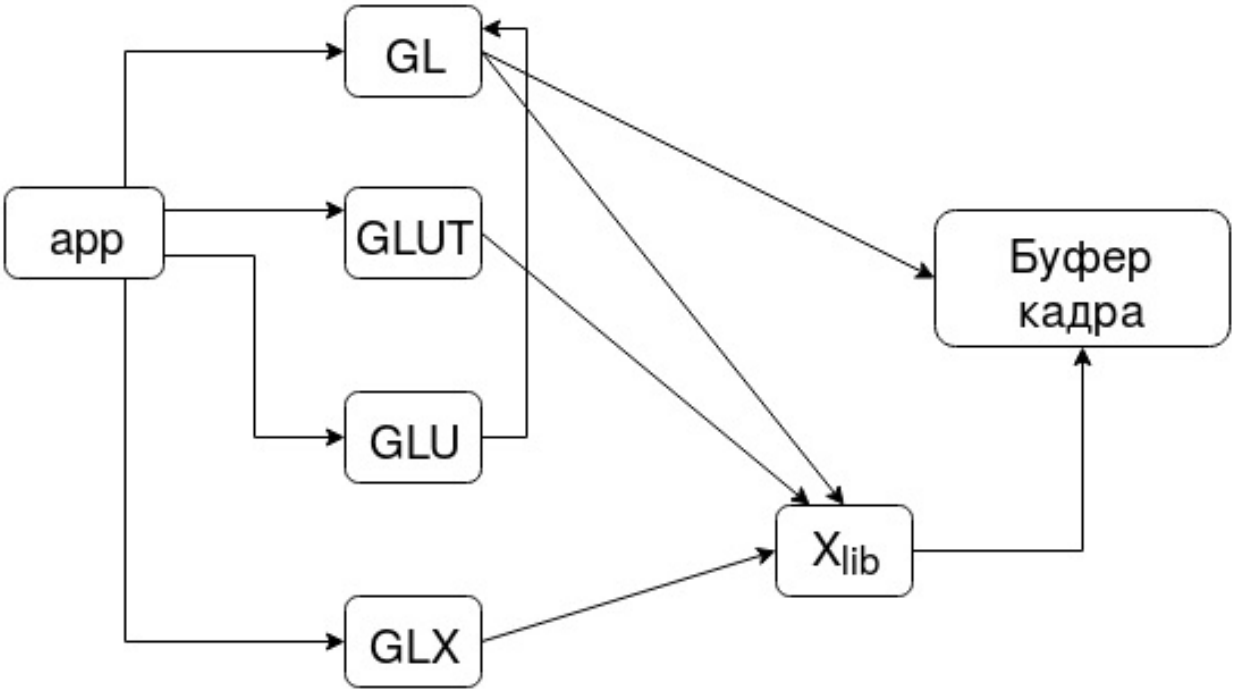
Производительность графического конвейера:

В рассматриваемой структуре обработки графической информации исользуются операции двух типов. На начальной стадии - операции с координатами вершин - числа с плавающей точкой. Все операции после растрового преобразования требуют выполнения побитовых операций на уровне содержимого буфера кадра.

Произведение графического конвейера = производительность при работе с плавающей точкой + производительность при побитоых операциях

Для оптимизации эти два процесса разносятся по разным платам.

Организация библиотеки OpenGL:



OpenGL Utility Library (GLU) — Библиотека графических утилит, надстройка над OpenGL, использующая её функции для рисования более сложных объектов. Состоит из большого количества функций, использующих библиотеку OpenGL для предоставления пользователю более простого и мощного интерфейса трёхмерной графики, основанного на более примитивном, предоставляемом базовыми функциями OpenGL. Обычно потсаляется вместе с библиотекой OGL.

- OpenGL Utility Toolkit (GLUT) - библиотека утилит для приложений под OpenGL, которая в основном отвечает за системный уровень операций ввода-вывода, при работе с операционной системой. Из функций можно привести следующие: создание окна, управление окном, мониторинг за вводом с клавиатуры и событий мыши. Она также включает функции для рисования ряда геометрических примитивов: куб, сфера, чайник. GLUT даже включает возможность создания несложных всплывающих меню.
- GLX – библиотека-расширение X Window System для создания интерфейсов.
- Xlib (X library, русск. библиотека «икс») — библиотека функций клиента системы X Window, написанная на языке Си.

2. OpenGL: примитивы, массивы вершин.

Под вершиной понимается точка в трехмерном пространстве, координаты которой можно задавать следующим образом:

```
void glVertex[2 3 4][s i f d](type coords)
void glVertex[2 3 4][s i f d]v(type *coords)
```

Координаты точки задаются максимум четырьмя значениями: x, y, z, w, при этом можно указывать два (x,y) или три (x,y,z) значения, а для остальных переменных в этих случаях используются значения по умолчанию: z=0, w=1. Как уже было сказано выше, число в названии команды соответствует числу явно задаваемых значений, а последующий символ – их типу.

Координатные оси расположены так, что точка (0,0) находится в левом нижнем углу экрана, ось x направлена влево, ось y- вверх, а ось z- из экрана. Это расположение осей мировой системы координат, в которой задаются координаты вершин объекта, другие системы координат будут рассмотрены ниже.

Однако чтобы задать какую-нибудь фигуру, одних координат вершин недостаточно, и эти вершины надо объединить в одно целое, определив необходимые свойства. Для этого в OpenGL используется понятие примитивов, к которым относятся точки, линии, связанные или замкнутые линии, треугольники и так далее. Задание примитива происходит внутри командных скобок:

```
void glBegin (GLenum mode)
void glEnd (void)
```

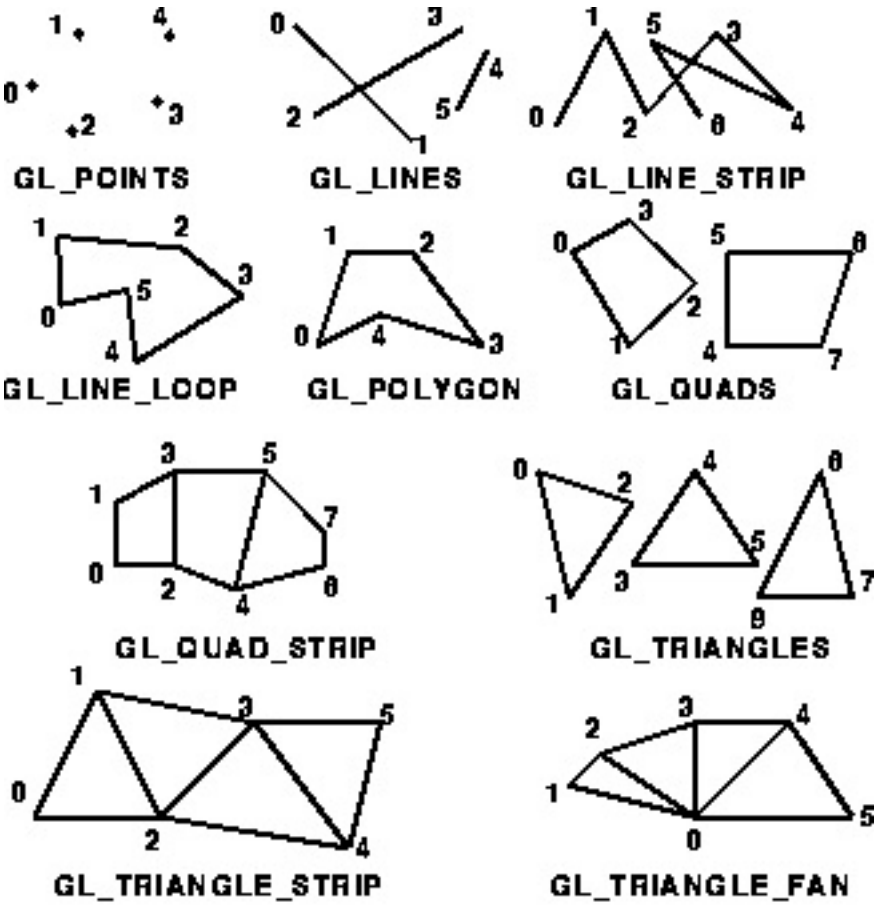
Параметр mode определяет тип примитива, который задается внутри и может принимать следующие значения:

- GL_POINTS – каждая вершина задает координаты некоторой точки.
- GL_LINES – каждая отдельная пара вершин определяет отрезок; если задано нечетное число вершин, то последняя вершина игнорируется.
- GL_LINE_STRIP – каждая следующая вершина задает отрезок вместе с предыдущей.
- GL_LINE_LOOP – отличие от предыдущего примитива только в том, что последний отрезок определяется последней и первой вершиной, образуя замкнутую ломаную.
- GL_TRIANGLES – каждая отдельная тройка вершин определяет треугольник; если задано не кратное трем число вершин, то последние вершины игнорируются.
- GL_TRIANGLE_STRIP – каждая следующая вершина задает треугольник

вместе с двумя предыдущими.

- GL_TRIANGLE_FAN – треугольники задаются первой и каждой следующей парой вершин (пары не пересекаются).
- GL_QUADS – каждая отдельная четверка вершин определяет четырехугольник; если задано не кратное четырем число вершин, то последние вершины игнорируются.
- GL_QUAD_STRIP – четырехугольник с номером n определяется вершинами с номерами 2n-1, 2n, 2n+2, 2n+1.
- GL_POLYGON – последовательно задаются вершины выпуклого многоугольника.

Примитивы:



Для задания текущего цвета вершины используются команды
void glColor[3 4][b s i f](GLtype components)
void glColor[3 4][b s i f]v(GLtype components)

Первые три параметра задают R, G, B компоненты цвета, а последний параметр определяет alpha-компоненту, которая задает уровень прозрачности объекта.

Массивы вершин

Если вершин много, то чтобы не вызывать для каждой команду glVertex..(), удобно объединять вершины в массивы.

1. Предопределены вершины в контексте OpenGl.
2. Можно хранить 8 типов данных - координаты, цвета, координаты текстуры
3. Упрощение работы с большим количеством вершин

Алгоритм использования:

1. Активация требуемого массива
2. Заполнение данными
3. Рисование на основе данных:
    - индивидуальный доступ к элементам
    - создание списков отдельных переменных
    - последовательная обработка элементов массива

3. OpenGL: алгоритм добавления света в сцену.

Свет позволяет сделать объеное изображение.

Типы света

1. Фоновый (ambient) - свет, поступающий с неопределенного направления и, отражаясь, выходит в неопределенном направлении.
2. Рассеянный (diffuse) - свет, поступающий из одного направления и отражается в разных направлениях. Чем больше угол падения, тем на поверхности он ярче, чем меньше, тем свет заметен меньше.
3. Отраженный (specular) или зеркальный - свет, поступающий и отраженный в одном направлении.
4. Излучаемый (emmisive) - свет, который излучает сама поверхность, при этом поверхность называется самоизлучающей.

Любой свет содержит три компоненты:

- фоновую
- рассеянную (диффузную)
- отраженную

Цвет материала

Согласно модели OGL цвет материала задается в зависимости от того, какой процент приходящего света поверхность отражает.
Цвет материала как и свет имеет три компоненты (3 цвета):

- фоновую
- рассеянную (диффузную)
- отраженную

Значения RGB для источников света и материала

Для источника света значение RGB зависит от процентной доли полной интенсивности каждого цвета.
Для материалов значения RGB берется из отражающей способности каждого цвета. Если мы ставим:
r = 1.0 то будет полностью отражаться красный цвет
g = 0.5 то зеленый будет отражатся 50%

Чтобы добавить на освещение, требуется выполнить несколько шагов:

1. Определить вектор нормали для каждой вершины каждого объекта. Эти нормали задают ориентацию объекта по отношению к источникам света.

2. Создать, выбрать и позиционировать один или более источников света.

3. Создать и выбрать модель освещения, которая определяет уровень глобального фонового света и эффективное положение точки наблюдения (для вычислений, связанных с освещением).

4. Задать свойства материала для объектов сцены.

5. **Определение нормалей для каждой вершины каждого объекта**

```
glBegin(GL_POLYGON);
    glNormal3fv(h0);
    glNormal3fv(v0);
    glNormal3fv(h1);
    glNormal3fv(v0);
    ...
glEnd();
```

Перед вычисление освещенности все вектора автоматически нормализуются (т.е. приводятся к единичному вектору).

Если используется масштабирование, то нормализацию следует проводить вручную.
Как преобразовать вручную:

```
glEnable(GL_NORMALIZE);

// менее затратная нормализация
glEnable(GL_RESCALENORMAL); //однородное масштабирование
```

2. **Создание источника света**

Максимальное количество источников света - 8.
Для управления свойствами источника света используются команды glLight{i f}v:

```
glLightf(GLenum light, GLenum pname, GLfloat param);
glLightfv(GLenum light, GLenum pname, const GLfloat *param);
```

Параметр light указывает OpenGL для какого источника света задаются параметры. Команда glLightf используется для задания скалярных параметров, а glLightfv используется для задания векторных характеристик источников света.

Имя параметра	Значение по умолчанию	Краткий комментарий
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	цвет фонового излучения источника света
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0) или (0.0, 0.0, 0.0, 1.0)	цвет рассеянного излучения источника света (значение по умолчанию для GL_LIGHT0 - белый, для остальных - черный)
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0) или (0.0, 0.0, 0.0, 1.0)	цвет зеркального излучения источника света (значение по умолчанию для GL_LIGHT0 - белый, для остальных - черный)

Пример

```
GLfloat light_ambient[] = {0.0, 0.0, 0.0, 0.1};
GLfloat light_dif[] = {1., 1., 1., 1.};

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_dif);
```

Типы источников света:

- точечные (позиционные, направленные (бесконечно удаленные))
- прожектор (поток света ограничен конусом)
- фоновый

```
glLight3fv(GL_LIGHT0, GL_SPOT_CUTOFF, alpha);
glLight3fv(GL_LIGHT0, GL_SPOT_DIRECTION, ...);
glLight3fv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, ...);
```

3. **Задание модели освещения**

Аспекты:

1. Интенсивность общего фонового освещения
2. Пололжение точки обзора
3. Расчет освещенности для лицевых и обратных граней
4. Флаг, определяющий необходимость отделения отраженного цвета от других состовляющих (важно при текстурировании)

```
glLightModel{if}[v](GLenum pname, param);
// param - перечисленные аспекты
```

4. **Определение материалов объектов**

```
glMaterial{if}[v](GLenum face, GLenum pname, param);
```

- *face* - может принимать три значения:
  - GL_FRONT
  - GL_BACK

    - GL_FRONT_AND_BACK
- *pname*
    - цвет компонента
    - GL_SHNINESS
    - GL_EMISSION
    - ...
- *param*

4. OpenGL: модельные и видовые преобразования, преобразования проецирования, преобразования окна просмотра.

Аналогия с фотографией.

Фото	Виртуальная сцена
Настрока сцены модели	Модельные преобразования
Фиксация фотоаппарата	Видовые преобразования
Настройка фотоаппарата (объектива)	Преобразование проецирования (выбор и настройка проекции)
Определение размера итогового фотоснимка	Настрока (задание) окна просмотра (viewport)

Стадии преобразования вершин:

Объектные координаты (x, y, z, w) -> Модельно-видовая матрица -> Система координат наблюдателя -> Матрица проецирования -> Отсечение кординат -> Перспективное деление -> Нормализация координат -> Преобразование обзорной точки -> Оконные координаты

К модельно-видовым преобразованиям будем относить перенос, поворот и изменение масштаба вдоль координатных осей. Для проведения этих операций достаточно умножить на соответствующую матрицу каждую вершину объекта и получить измененные координаты этой вершины:

(x’, y’, z’, 1)T = M * (x, y, z, 1)T

где M – матрица модельно-видового преобразования.
Включают в себя:

1. Масштабирование **glScale**
2. Поворот **glRotate**
3. Параллельный перенос **glTranslate**

Они ортогональны и нельзя с помощью одного сделать другое.

glTranslate

- умножает текущую матрицу на матрицу, которая перемещает объект с помощью значений x, y, z.

```
glTranslate{fd}(x, y, z);
```

glRotate

- умножает текущую матрицу на матрицу, осуществляющую поворот *против часовой стрелки* на угол **angle** вокруг луча из начала координат в точку (x, y, z). Объект, располагающийся дальше от оси вращения,

будет повернут сильнее (у него больше орбита вращения), чем объект, который располагается ближе к оси вращения.

```
glRotate{fd}(angle, x, y, z);
```

glScale

- умножает текущую матрицу на матрицу, осуществляющую растяжение, сжатие и зеркальное отображение относительно осей.

```
glScale{fd}(x, y, z);
```

Области определения аргументов *x, y, z*:

Значение	Событие
> 1	Растяжение
== 1	Ничего не происходит
< 1, > 0	Сжатие

Все эти преобразования будут применяться к примитивам, описания которых будут находиться ниже в программе. В случае если надо, например, повернуть один объект сцены, а другой оставить неподвижным, удобно сначала сохранить текущую видовую матрицу в стеке командой glPushMatrix(), затем вызвать glRotate..() с нужными параметрами, описать примитивы, из которых состоит этот объект, а затем восстановить текущую матрицу командой glPopMatrix().

Кроме изменения положения самого объекта иногда бывает нужно изменить положение точки наблюдения, что однако также приводит к изменению видовой матрицы. Это можно сделать с помощью команды

void gluLookAt (GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz)

где точка (eyex,eyey,eyez) определяет точку наблюдения, (centerx, centery, centerz)задает центр сцены, который будет проектироваться в центр области вывода, а вектор (upx,upy,upz) задает положительное

направление оси y, определяя поворот камеры. Если, например, камеру не надо поворачивать, то задается значение (0,1,0), а со значением (0,-1,0) сцена будет перевернута.

Фактически, эта команда совершает перенос и поворот объектов сцены, но в таком виде задавать параметры бывает удобнее.

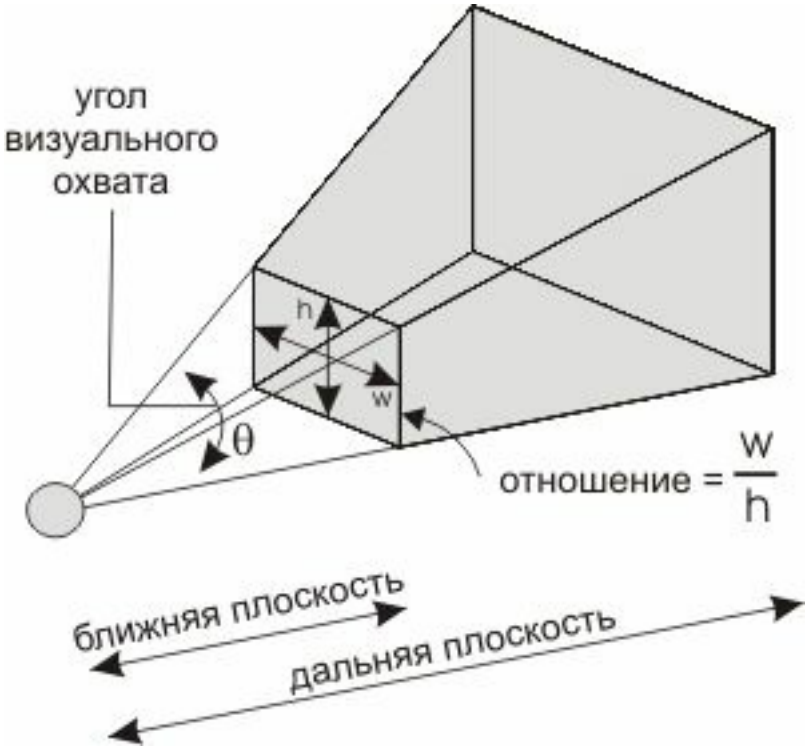
Преобразования проецирования

Перспективная проекция - объекты, находящиеся дальше, становятся меньше.
Наиболее узнаваемой характеристикой перспективной проекции является уменьшение на расстоянии: чем дальше объект находится от камеры (точки наблюдения), тем меньше он будет в финальном изображении. Это происходит потому, что объем видимости перспективной проекции имеет форму усеченной пирамиды (пирамиды, верхушка которой отрезана плоскостью, параллельной ее основанию). Объекты, попадающие в объем видимости проецируются из вершины пирамиды, где находится точка наблюдения. Более близкие к точке наблюдения объекты получаются крупнее, поскольку они занимают пропорционально большее пространство объема видимости. Более далекие объекты оказываются меньше, поскольку они находятся в более широкой части усеченной пирамиды объема видимости. Данный метод проецирования используется для анимации, визуальной симуляции и в любых других приложениях, претендующих на некоторую долю реализма, так как перспективное проектирование похоже на то, как видит человеческий глаз (или камера).

```
glFrustum(GLdouble left,
          GLdouble right,
          GLdouble bottom,
          GLdouble top,
          GLdouble nearVal,
          GLdouble farVal);
```

Создает матрицу перспективного проецирования и умножает на нее текущую матрицу. Объем
видимости задается параметрами (left, bottom,-near) и (right, top,-near) определяющими координаты (x, y, z) левого нижнего и правого верхнего углов ближней отсекающей плоскости; near и far задают дистанцию от точки наблюдения до ближней и дальней отсекающих плоскостей (они всегда должны быть положительными).
Пирамида имеет ориентацию в пространстве по умолчанию. Вы можете

производить повороты
или переносы для управления ее положением, но это весьма сложный процесс, которого почти
всегда можно избежать.




```
gluPerspective(GLdouble fovy,
               GLdouble aspect,
               GLdouble zNear,
               GLdouble zFar);
```

Создает матрицу для пирамиды симметричного перспективного вида и умножает на нее текущую матрицу. Параметр fovy задает угол визуального охвата в плоскости yz, его значение должно лежать в диапазоне [0.0, 180.0]. Параметр aspect – это отношение ширины пирамиды к ее высоте. Параметры near и far представляют дистанции от точки наблюдения до ближней и дальней плоскостей отсечения вдоль отрицательного направления оси z.

Параметры	**Значение**
fovy	Угол зрения (0, 180).
aspect	Отношение сторон усеченной пирамиды видимости, а имеено ширины к высоте.

zNear	Расстояние до близжайшей плоскости.
zFar	Расстояние до дальней плоскости.

Ортографичекая проекция:

Матрица ортографической проекции задает усечённую пирамиду в виде параллелограмма, который является
пространством отсечения, где все вершины, находящиеся вне его объема отсекаются.

```
glOrtho(GLdouble left,
        GLdouble right,
        GLdouble bottom,
        GLdouble top,
        GLdouble nearVal,
        GLdouble farVal);
```

Усеченная пирамида определяет область видимых координат и задается шириной, высотой, ближней и дальней плоскостями. Любая координата, расположенная перед ближней плоскостью, отсекается, точно также поступают и с координатами, находящимися за задней плоскостью.

Преобразования окна просмотра

Окно просмотра представляет собой прямоугольную область окна, где рисуется изображение. Окно просмотра измеряется в оконных координатах, которые отражают позиции пикселей на экране относительно нижнего левого угла окна.

```
glViewport(GLint x,
           GLint y,
           GLsizei width,
           GLsizei height);
```

Парамерт	**Значение**
x, y	Нижний левый угол прямоугольника видового экрана в пикселях. Начальное значение (0,0).
width,	

height	Ширина и высота окна просмотра.

Задает прямоугольник пикселей в окне, в который будет перенесено финальное изображение.
Параметры (x,y) задают нижний левый угол порта просмотра, а параметры width и height– размер прямоугольника порта просмотра. По умолчанию левый нижний угол порта просмотра находится в левом нижнем углу окна, а его размер совпадает с размерами окна.
Отношение ширины порта просмотра к его высоте обычно должно быть таким же, как и соответствующее отношение объема видимости используемой проекции. Если эти два отношения не совпадают, спроецированное изображение при отображении в порте просмотра будет искажено. Изменение размеров окна не влияет на порт просмотра.

```
glViewport(0, 0, winWidth, winHeight);
```

5. OpenGL: понятие текстуры; действия при наложении текстур. Списки отображения.

Наложение текстуры на поверхность объектов сцены повышает ее реалистичность, однако при этом надо учитывать, что этот процесс требует значительных вычислительных затрат. Под текстурой будем понимать некоторое изображение, которое надо определенным образом нанести на объект. Для этого следует выполнить следующие этапы:
выбрать изображение и преобразовать его к нужному формату
загрузить изображение в память
определить, как текстура будет наноситься на объект и как она будет с ним взаимодействовать.

Текстура - прямоугольный массив данных. Данные:

- цвета
- яркость
- альфа-каналы

Единица текстуры - **тексель**.
Текстуры усложняются и возникают проблемы с приминением, когда поверхности не являются стандартными.
В зависимости от поверхности, тектуры могут различаться и методы использования могут меняться.

Подготовка текстуры

При создании образа текстуры в памяти следует учитывать следующие требования.

Во-первых, размеры текстуры как по горизонтали, так и по вертикали должны представлять собой степени двойки. Это требование накладывается для компактного размещения текстуры в памяти и способствует ее эффективному использованию. Использовать только текстуры с такими размерами конечно неудобно, поэтому перед загрузкой их надо преобразовать.

Во-вторых, надо предусмотреть случай, когда объект по размерам значительно меньше наносимой на него текстуры. Чем меньше объект, тем меньше должна быть наносимая на него текстура и поэтому вводится понятие уровней детализации текстуры. Каждый уровень детализации задает некоторое изображение, которое является как правило уменьшенной в два раза копией оригинала. Такой подход позволяет улучшить качество нанесения текстуры на объект.

Методы наложения текстуры

При наложении текстуры, как уже упоминалось, надо учитывать случай, когда размеры текстуры отличаются от размеров объекта, на который она накладывается. При этом возможно как растяжение, так и сжатие изображения, и то, как будут проводиться эти преобразования может серьезно повлиять на качество построенного изображения. Для определения положения точки на текстуре используется параметрическая система координат (s,t), причем значения s и t находятся в отрезке [0,1].

Координаты текстуры

Перед нанесением текстуры на объект осталось установить соответствие между точками на поверхности объекта и на самой текстуре. Задавать это соответствие можно двумя методами: отдельно для каждой вершины или сразу для всех вершин, задав параметры специальной функции отображения.

Первый метод реализуется с помощью команд

```
void glTexCoord[1 2 3 4][s i f d](type coord)
void glTexCoord[1 2 3 4][s i f d]v(type *coord)
```

Чаще всего используется команды вида glTexCoord2..(type s, type t), задающие текущие координаты текстуры. Вообще, понятие текущих координат текстуры аналогично понятиям текущего цвета и текущей нормали, и является атрибутом вершины.

Второй метод реализуется с помощью команд

```
void glTexGen[i f d](GLenum coord, GLenum pname, GLtype param)
void glTexGen[i f d]v(GLenum coord, GLenum pname, const GLtype *params)
```

Параметр coord определяет для какой координаты задается формула и может принимать значение GL_S, GL_T; pname определяет тип формулы и может быть равен GL_TEXTURE_GEN_MODE, GL_OBJECT_PLANE, GL_EYE_PLANE. С помощью params задаются необходимые параметры, а param может быть равен GL_OBJECT_LINEAR, GL_EYE_LINEAR, GL_SPHERE_MAP. Рассмотрение всех возможных комбинаций значений аргументов этой команды заняло бы слишком много места, поэтому в качестве примера рассмотрим, как можно задать зеркальную текстуру. При таком наложении текстуры изображение будет как бы отражаться от поверхности объекта, вызывая интересный оптический эффект. Для этого сначала надо создать два целочисленных массива коэффициентов s_coeffs и t_coeffs со значениями (1,0,0,1) и (0,1,0,1) соответственно, а затем вызвать команды: glEnable(GL_TEXTURE_GEN_S); glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR); glTexGendv(GL_S, GL_EYE_PLANE, s_coeffs); и такие же команды для координаты t с соответствующими изменениями.

Списки отображения

Список отображения (дисплейный список):

- два варианта хранения данных - список отображения и непосредственный режим (данные как есть);
- информация обрабатывается точно так же, как в непосредственном режиме;
- актуальны при использовании больших наборов данных.

Список отображения (display list) - удобный, эффективный способ именования и организации набора команд OGL.

Пояснения:

1. Созданиие и выполнение списка

```
glNewList();
...
glEndList();
```

Обе функции работают в сзязке. Если задать только конец, то компилятор выдаст ошибку.

Одновременно может быполняться только одина список отображения. Нельзя вложить один список в другой.

2. Присвоение имени списку и его создание

Функция позволяет генерировать уникальные неиспользуемые индексы:

```
GLuint glGenLists(GLsizei range);
// range - диапазон
// возвращает целое число - начало блока индексов,
// если возвращает 0 - то выделить не может
```

Функция создания или замены списка отображения:

```
void glNewList(GLuint list, GLenum mode);
```

list - идентификатор

mode:

- *GL_COMPILE* - используется, когда необходимо, чтобы команды OGL обрабатывались во время их размещения в списке
- *GL_COMPILEAND_EXECUTE* - используется, когда необходимо, чтобы команды, до помещения их в список, выполнялись в непосредственном режиме

1. Вызов списка

Функция вызова списка:

```
void glCallList(GLuint list);
```

Когда разрушается контекст программы список тоже разрушается.
Список нельзя никуда сохранить, он существует только в контексте программы.

4. Удаление списка

Функция удаляет набор списков с последовательными индексами. Начиная с *list* в количестве *range*.
При попытке удаления несуществующего списка ничего не происходит.
После удаления, удаленный индекс становится доступен.

```
void glDeleteLists(GLuint list, GLsizei range);
```