OpenGL. Графический конвейер. Геометрические преобразования

* Введение в OpenGL
* Что такое OpenGL
* Архитектура OpenGL
* Графический конвейер
* Геометрические преобразования
* OpenGL и геометрические преобразования

Терминология

* **Пиксель**- это наименьшая точка, которую можно различить на экране. Он является единицей двухмерного изображения.
* **Буфер**- это область для временного хранения данных.
* **Двойная буферизация**- это один из способов рендеринга, при котором существует два буфера. Содержимое первого (переднего) буфера (**front buffer**) показываеться на экран и вы видите какое то изображение. В это время на заднем буфере (**back buffer**) подготавливается следующий кадр (рисуется). Когда кадр на заднем буфере готов, тогда передний и задний буфер меняются местами. И затем все это повторяется. С помощью такой технологии избегается мерцание экрана.
* **Рендеринг**- это процесс подготовки, выдачи (прорисовки) изображения на экран. В общем, это все действия, которые связаны с выдачей картинки на экран.
* **Камера**- Камера- это место, из которого вы смотрите на экран.
* **Трансформация**- нахождение координат точки в заданной системе координат, используя координаты точки в другой системе (это переносы, или вращения). При трансформации положение точки не меняется.
* **Мировая система координат**- это система координат, которая считается неподвижной. Обычно относительно этой системы координат задаются положение камеры, и объектов.
* **Примитив**- это основная единица, из которой строятся сложные объекты. Примитивом может быть: точка, линия, треугольник, или что ни будь подобное…
* **Z-buffer**- также часто вместо этого слова употребляется слово Буфер Глубины (Depth Buffer). Это буфер величиной с экран, в котором хранятся Z координаты пикселей. Z-ось добавляет третью величину- глубину.
* **Матрица**- в компьютерной графике, это массив чисел (обычно 4х4), который содержит значения векторов систем координат.
* **Проекция**- это перевод координат из пространства (камеры) на экран, или с 3х мерных координат в 2х мерные.

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*
* *Цель создания библиотеки* - обеспечение низкоуровневого доступа к возможностям графического аппаратного обеспечения, не теряя платформенной независимости

Почему OpenGL?

* Аналогичные библиотеки: DirectX (Direct3D), Java 3D
* OpenGL
  + **Стабильность (с 1992 г.)**
  + **Переносимость**
    - Независимость от оконной и операционной системы
  + **Легкость применения**
    - Простой интерфейс, реализации для различных ЯП
    - Низкие затраты на обучение
      * *Подходит для обучения студентов!*

Характерными особенностями OpenGL, которые обеспечили распространение и развитие этого графического стандарта, являются

* Стабильность

Дополнения и изменения в стандарте реализуются таким образом, чтобы сохранить совместимость с разработанным ранее программным обеспечением.

* Надежность и переносимость

Приложения, использующие OpenGL, гарантируют одинаковый визуальный результат вне зависимости от типа используемой операционной системы и организации отображения информации. Кроме того, эти приложения могут выполняться как на персональных компьютерах, так и на рабочих станциях и суперкомпьютерах.

* Легкость применения

Стандарт OpenGL имеет продуманную структуру и интуитивно понятный интерфейс, что позволяет с меньшими затратами создавать эффективные приложения, содержащие меньше строк кода, чем с использованием других графических библиотек.

Хронология событий

* 1992г. - Open GL 1.0
* 1997г. – Open GL 1.1
  + Появились текстурные объекты и массивы вершин
* 1998г. – Open GL 1.2
  + 3D-текстуры и функции обработки изображений
* 2001г. – OpenGL 1.3
  + Кубические текстуры, сжатые текстуры, мультитекстурирование
* 2002г. – OpenGL 1.4
  + Автоматическое генерирование mip-уровней текстур, доп. функции смешивания, depth-текстуры, рисование множественных массивов вершин одной командой
* 2003г. – OpenGL 1.5
  + Vertex Buffer Objects, Shadow mapping comparison functions, Occlusion queries, Non-Power-of-Two textures

Open GL 2.0

* Говорили, что, по сравнению **с** [**DirectX**](https://ru.wikipedia.org/wiki/DirectX), главной проблемой OpenGL является консорциум (который и должен заниматься развитием OpenGL), в который входит большое количество компаний с различными интересами, что приводит к длительному периоду принятия новой версии спецификации. OpenGL версии 2.0 была представлена 3Dlabs в ответ на беспокойство относительно медленного и нечёткого направления развития OpenGL.
* 3Dlabs предложила ряд существенных дополнений к стандарту, наиболее значимым из которого было добавление к ядру OpenGL языка шейдеров GLSL (OpenGL Shading Language).
* Это позволяет программисту заменить фиксированный конвейер OpenGL небольшими программами на специальном языке для создания различных эффектов, таких, как [bump mapping](https://ru.wikipedia.org/wiki/Bump_mapping), [normal mapping](https://ru.wikipedia.org/wiki/Normal_mapping), [parallax mapping](https://ru.wikipedia.org/wiki/Parallax_mapping), [HDR](https://ru.wikipedia.org/wiki/High_Dynamic_Range) и т. Д.
* Опубликован  в 2004 году
* Основное нововведение – высокоуровневой язык шейдеров **GLSL** (OpenGL Shading Language), предоставляющих приложениям возможность реализации собственных механизмов рендеринга при помощи замены стандартных обработчиков вершин и фрагментов

Основные возможности OpenGL

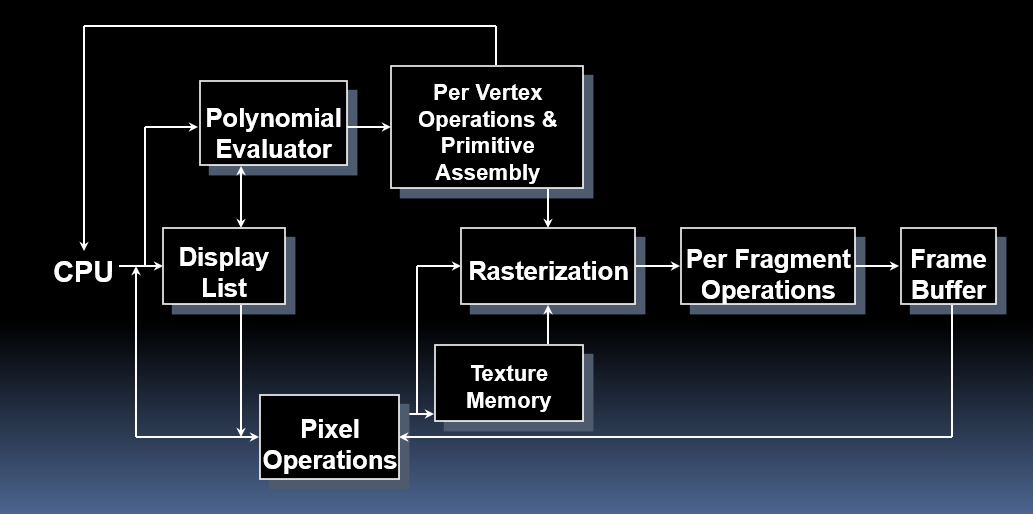
* + Набор базовых примитивов: точки, линии, многоугольники и т.п.
  + Видовые и координатные преобразования
  + Удаление невидимых линий и поверхностей (z-буфер)
  + Использование сплайнов для построения линий и поверхностей
  + Наложение текстуры и применение освещения
  + Добавление специальных эффектов: тумана, изменение прозрачности,сопряжение цветов (blending), устранение ступенчатости (anti-aliasing).

OpenGL. Основы

* + Визуализация/Rendering
    - Обычно выполнение команд OpenGL
    - Преобразование геометрических / математических описаний объектов в значения буфера кадра
  + OpenGL может исполнить/render:
    - Геометрические примитивы
      * Линии, точки, многоугольники, и т.д ...
    - Bitmaps и изображения
      * Изображения и геометрия связаны между собой   
         через текстурирование

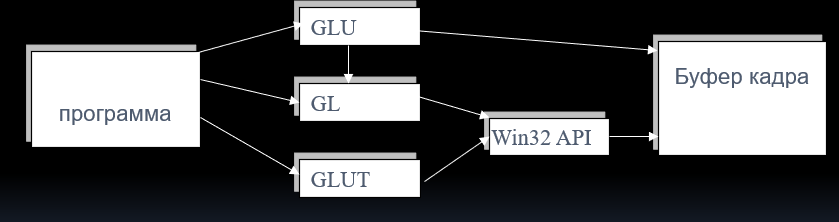
Основные возможности OpenGL

Это самая важная диаграмма, представляет поток графической информации, так как она обрабатывается от CPU в буфер кадра.  
Существуют два потока данных. Верхний для геометрических, вершинных на основе примитивов. Нижний для пикселя на основе, графических примитивов. Текстурирование сочетает в себе два типа примитивов вместе.



Организация OpenGL

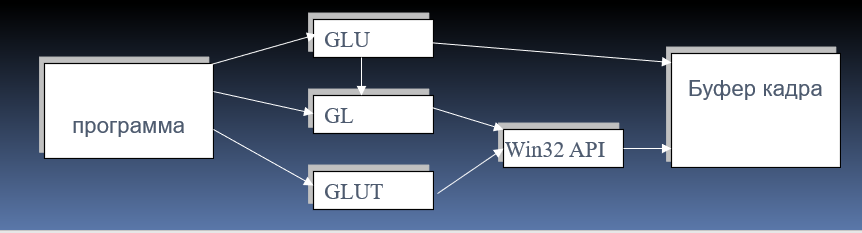
* Состоит из набора библиотек
  + пример для Win32



**Были созданы библиотеки для отображения информации с помощью оконной подсистемы для операционных систем Windows и Unix (WGL и GLX соответственно), а также библиотеки GLAUX и GLUT, которые используются для создания так называемых консольных приложений.**

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**



* Библиотека GLAUX уступает по популярности написанной несколько позже библиотеке GLUT, хотя они предоставляют примерно одинаковые возможности. В дальнейшем в данном пособии в качестве основной будет рассматриваться библиотека GLUT, предоставляющая широкий набор средств взаимодействия с пользователем.
* В состав библиотеки GLU вошла реализация более сложных функций, таких как
  + - набор популярных геометрических примитивов (куб, шар, цилиндр, диск),
    - функции построения сплайнов,
    - реализация дополнительных операций над матрицами и т.п.
* Все они реализованы через базовые функции OpenGL.

С какими геометрическими моделями работает OpenGL?

* OpenGL работает с моделями, заданными в граничном полигональном представлении

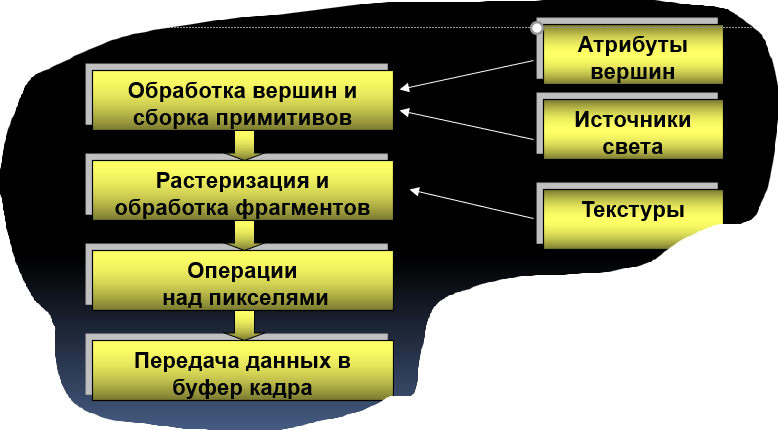
Поверхность приближается набором полигональных граней (face, polygon)

Границы граней описываются ребрами (egde)

Часть отрезка, формирующего ребро, заканчивается вершинами (vertex)

Архитектура

С точки зрения архитектуры, графическая система OpenGL является конвейером, состоящим из нескольких этапов обработки данных:



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

Синтаксис команд OpenGL

* **GLint** - int
* **GLuint** - unsigned int
* **GLfloat** - float

Команды они имеют следующий вид:

**glCommand\_name[n][type][v]**   
 Таким образом имя любой команды состоит из нескольких частей:

* + **gl** Это имя библиотеки, в которой находиться данная функция. Мы можем использовать и другие библиотеки, а именно: glu- библиотека GLU, glut- функции из библиотеки GLUT, glaux- библиотека GLAUX.
  + **Comand\_name** Это имя команды, например InitWindowSize (к стати, эта команда нам должна быть уже известна).
  + **[n]** Число аргументов.
    - **[type]** Тип аргумента. Возможные типы

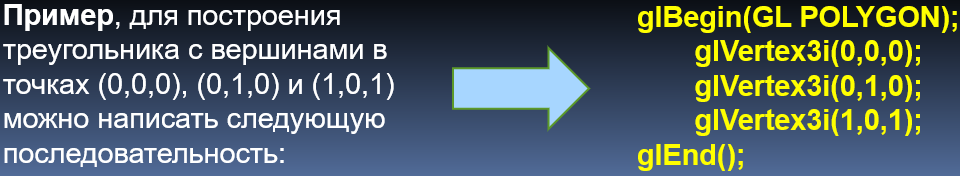
**f** GLfloat   
**d** GLdouble   
**s** GLshort

**[v]** Если указан этот параметр, то это означает, что в качестве параметров функции используется указатель на массив.

Типы примитивов OpenGL

Предусмотрено десять различных геометрических объектов:

* - **point** - последовательность точек;
* - **line strip** - ломаная линия;
* - **line loop** - замкнутая ломаная линия;
* - **separate line** - отрезок линии;
* - **polygon** - line loop, задающая границы выпуклого многоугольника;
* - **triangle strip** - множество связных треугольников;
* - **triangle fan** - множество треугольников, связанных наподобие веера;
* - **separate triangle** - отдельный треугольник;
* - **quadriteral strip** - связные четырехугольники;
* - **independent quard** - отдельный четырехугольник.



Атрибуты вершин

* Каждая вершина кроме *положения в пространстве* может иметь несколько других атрибутов
  + Материал
  + Цвет
  + Нормаль
  + Текстурные координаты

***Внимание****: всегда используется ТЕКУЩИЙ набор атрибутов*

* *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)*
* 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,
* 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)

Полигоны в OpenGL

Причина, по которой OpenGL накладывает подобные ограничения, заключается в том, что в аппаратуру намного проще заложить функции визуализации некоторого определенного класса полигонов.

Простые полигоны визуализировать легко.

Трудные случаи тяжело обнаружить быстро.

Для увеличения быстродействия OpenGL «скрещивает пальцы» и предполагает, что все полигоны простые.

* В общем случае полигоны могут быть достаточно сложны, поэтому OpenGL накладывает очень серьезные ограничения в отношении того, что считается полигональным примитивом.
  + Во-первых, ребра полигонов в OpenGL не могут пересекаться (в математике полигон, удовлетворяющий этому условию, называется *простым*).
  + Во-вторых, полигоны OpenGL должны быть выпуклыми. Полигон является выпуклым, если отрезок, соединяющий две точки полигона (точки могут быть и внутренними, и граничными) целиком лежит внутри полигона (то есть все его точки также принадлежат полигону).

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

Закраска полигона

glBegin (GL\_TRIANGLES);

glColor3f (1.0, 0.0, 0.0); /\* red \*/

glVertex2f (5.0, 5.0);

glColor3f (0.0, 1.0, 0.0); /\* green \*/

glVertex2f (25.0, 5.0);

glColor3f (0.0, 0.0, 1.0); /\* blue \*/

glVertex2f (5.0, 25.0);

glEnd();



* Кроме задания самих примитивов можно определить метод их отображения на экране, где под примитивами в данном случае понимаются многоугольники.

Сначала надо определить понятие лицевых и обратных граней.

* Под гранью понимается одна из сторон многоугольника, и по умолчанию лицевой считается та сторона, вершины которой обходятся против часовой стрелки. Направление обхода вершин лицевых сторон можно изменить вызовом команды **void glFrontFace(GLenum mode)** со значением параметра **mode** равным **GL\_CW**, а отменить- с **GL\_CCW**.
* Чтобы изменить метод отображения многоугольника используется команда **void glPolygonMode(GLenum face, Glenum mode)**
* Параметр mode определяет, как будут отображаться многоугольники, а параметр face устанавливает тип многоугольников, к которым будет применяться эта команда и может принимать следующие значения:

**GL\_FRONT для лицевых граней**

**GL\_BACK для обратных граней**

**GL\_FRONT\_AND\_BACK для всех граней**

Координатная система

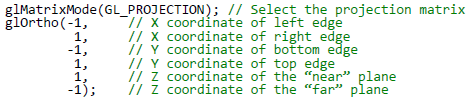
* Декартовы координаты в математике, инженерии
  + обычно моделируются с плавающей точкой
  + как правило, X направлено вправо, Y растет вверх
  + Дисплей (физические) координаты
  + только целое число
  + как правило, X направлено вправо, Y увеличивается вниз
  + 1 единица = 1 пиксель
* Но мы хотим быть изолированными от физических координат дисплея

OpenGL является лишь посредником

Координатная система в OpenGL

* Координаты OpenGL
  + Выберите соглашение
    - Для нас: X увеличивает право, Y увеличивается вверх
  + Единицы основаны на размере окна или экрана
    - Видимая область протягивается, чтобы заполнить окно
    - Единицы в процентах от размера окна, не соответствуют физическим единицам или пикселям

Определяем систему координат с помощью матрицы проецирования

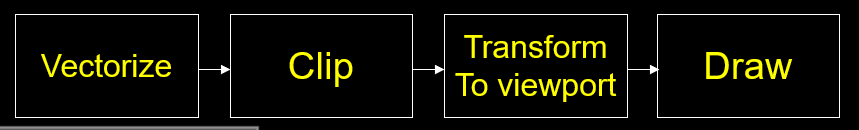


* Два варианта о том, как можно думать
  + Нарисуйте все в системе координат OpenGL
  + Это неудобно: вместо того, чтобы выбрать свои собственные абстрактные системы координат в соответствии с вашими потребностями для каждого объекта, а затем указать все примитивы к OpenGL с использованием этих координат. Укажите **преобразование** для отображения объекта в координаты OpenGL.

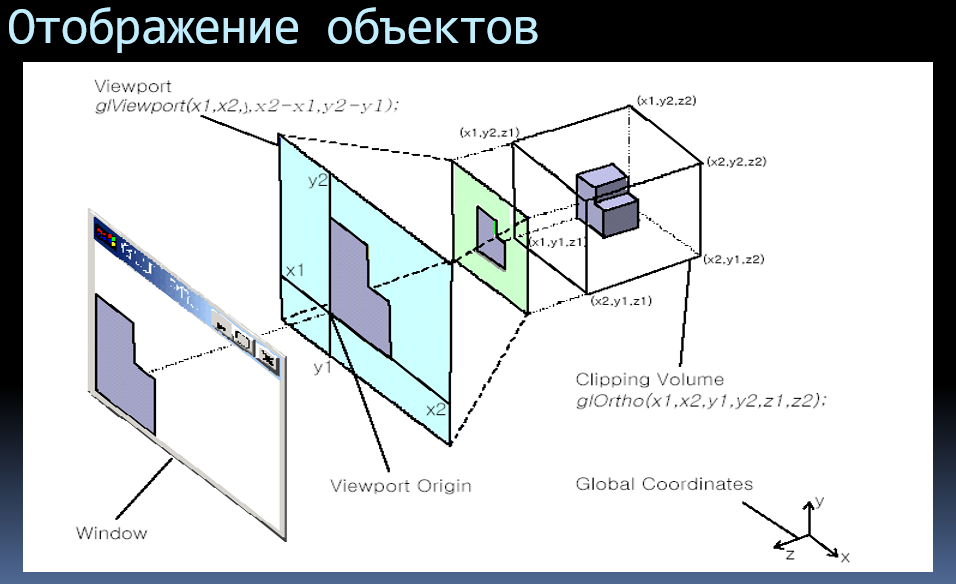
Когда мы говорим «трансформация», мы обычно подразумеваем композицию преобразований масштаба, поворот и переноса

Поле вывода и поле вывода отсечения в OpenGL

* Проецирование 3D объектов на 2D экран (процесс конвейера CG)



* Область просмотра
  + Двумерный (2D) прямоугольник, в который 3D сцена проецируется  
    векторизация
* Векторизация
  + Представление 3D-объектов в виде линейных или плоских частей объектов или элементов
* Отсечение (клип)
  + Удалите не заинтересованные областей из vecterized элементов



Создадим треугольник/triangle

void DrawTriangle()

{

glColor3f(1.0f,0.0f,0.0f);

glBegin(GL\_TRIANGLES);

glVertex3f(0.0f,0.0f,0.0f);

glVertex3f(1.0f,0.0f,0.0f);

glVertex3f(0.0f,1.0f,0.0f);

glEnd();

}

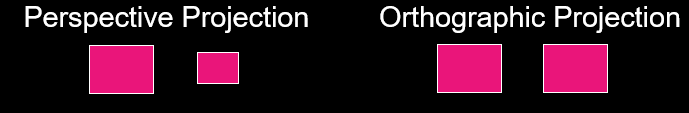
Позиционирование

* Определить местоположение и направление треугольника
  + glTranslatef(1.0f,1.0f,0.0f);
  + glRotatef(45.0f,0.0f,1.0f,0.0f);
  + DrawTriangle();

**Угол обзора**

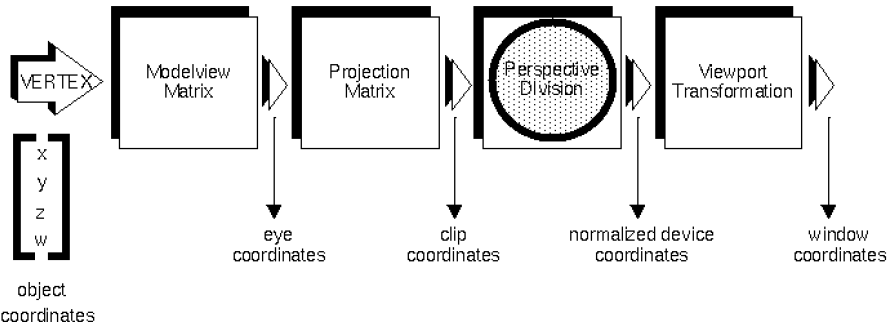
* Изменение угла обзора
  + glMatrixMode(GL\_PROJECTION);
  + glLoadIdentity();
  + glOrtho(-1.0f,1.0f,-1.0f,1.0f,-1.0f,10.0f);
  + glMatrixMode(GL\_MODELVIEW);
  + gluLookAt(1.5f,1.5f,1.0f,1.5f,1.5f,-0.5f, 0.0f,1.0f,0.0f);

Проецирование



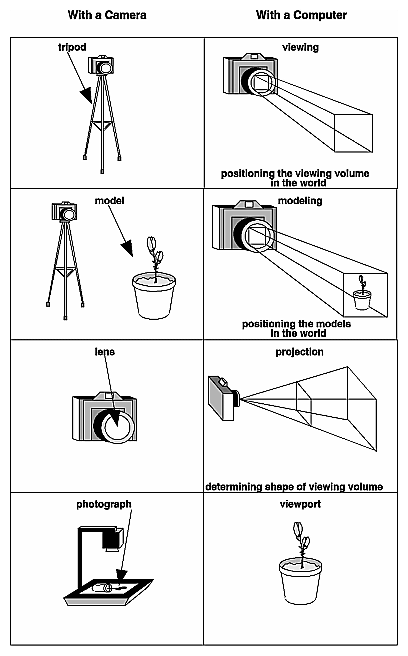
* Выбор проекции
  + Perspective Projection
    - glFrustum(-1.0f,1.0f,-1.0f,1.0f,1.0f,100.0f);
  + Orthographic Projection
    - glOrtho(-1.0f,1.0f,-1.0f,1.0f,0.0f,100.0f);

Преобразование вершин



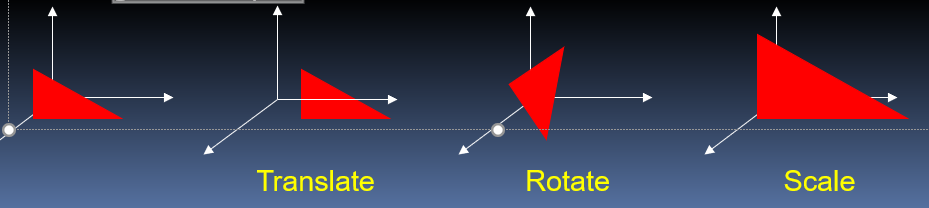
Просмотр/Viewing

* Позиционирование объекта
* Изменение угла обзора
* Проекционные и трансформационные преобразования
* Область просмотра и определения координат



Модельные преобразования

* Поместить объект и изменить угол обзора
* glMatrixMode(GL\_MODELVIEW)
* glLoadIdentity() – Матрица идентичности
  + glTranslatef(x,y,z)
  + glRotatef(degree,x,y,z)
  + glScalef(x,y,z)



Преобразования просмотра

* Changes the position and orientation of the viewpoint
* Generally it consists of translations and rotations.
  + gluLookAt()

**Проекционные преобразования**

* Определение объема просмотра
  + определяет, как объект проецируется на экран
  + glMatrixMode(GL\_PROJECTION);
  + glLoadIdentity();
  + glFrustum(left,right, bottom,top, near, far);
  + glOrtho(left,right, bottom,top, near, far);

**Перспективная проекция I**

* glFrustum(left, right, bottom, top, near, far);
* (left, right, bottom, top) - near clipping plane
* near и far задают расстояния от точки обзора до ближней и дальней плоскостей
  + отсечения.Простой для понимания, но не интуитивно понятный в использовании

**Перспективная проекция I**I

* gluPerspective(fovy, aspect, near, far);
* Создает матрицу для симметричной усеченной пирамиды в перспективе.

**Орфографическая проекция**

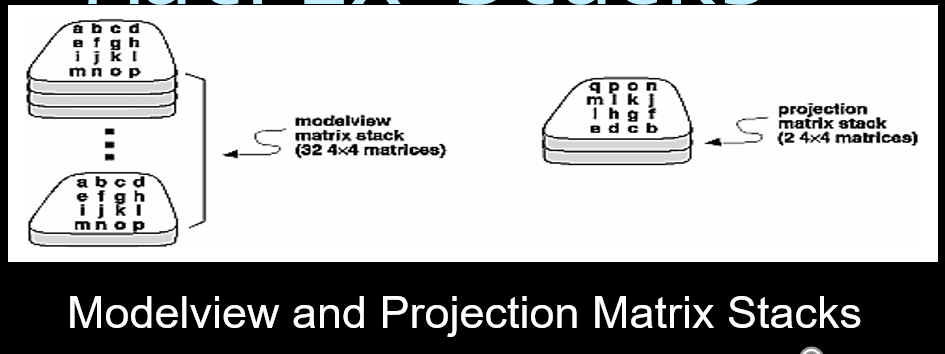
* объем просмотра представляет собой прямоугольный параллелепипед
* glOrtho(left, right, bottom, top, near, far);

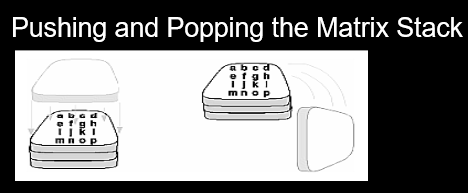
**Преобразование видового экрана**

* Окно просмотра — это прямоугольная область окна, в которой рисуется изображение.Измеряется в координатах окна, которые отражают положение пикселей на экране относительно левого нижнего угла окна..
  + glViewport(x, y, width, height);

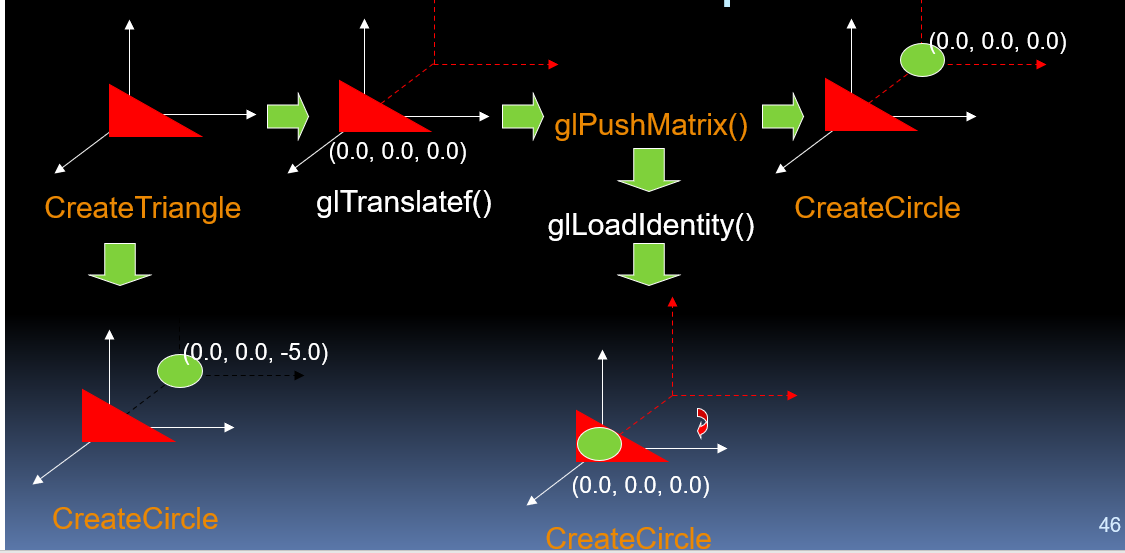
**Соотношение сторон (Aspect Ratio)**

* Сопоставление объема просмотра с окном просмотра
* Отображаемое изображение может быть искажено в зависимости от соотношения (w/h)
* Matrix StacksA stack of matrices is useful for constructing hierarchical models, in which complicated objects are constructed from simpler ones.
  + glMatrixMode()
  + glLoadMatrix(), glMultMatrix()
  + glPushMatrix(), glPopMatrix()
  + glLoadIdentity()





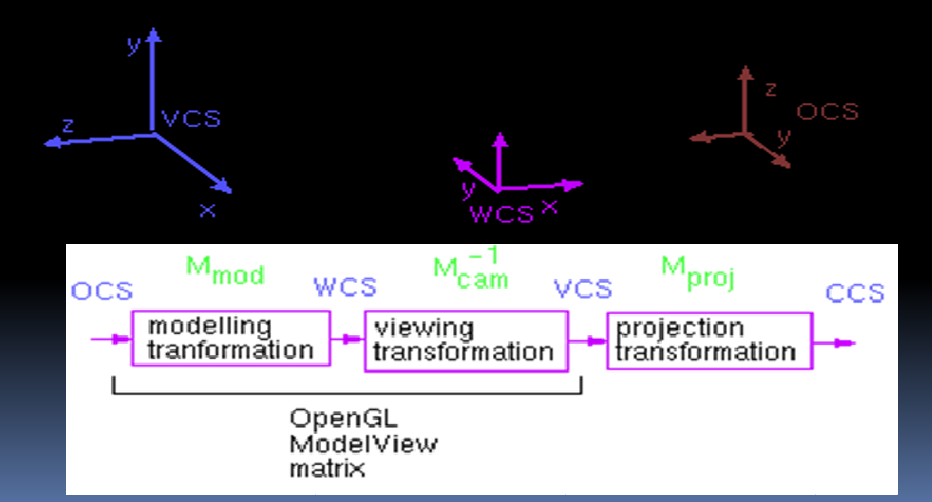
Matrix Stacks Example



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



vocs| модельные преобразования (в мировой координатной системе) Modeling Transform (into World Coordinate System)

vwcs| видовые преобразования (в видовой системе координат)

Viewing Transform (into Viewing Coordinate System)

vvcs| преобразования проецирования ( усеченные КС)

Projection Transform (into Clipping Coordinate System)

vccs|

Perspective Division (into Normalized Device Coordinate System)

vndcs|

Viewport Transform (into Device Coordinate System)vdcs

**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