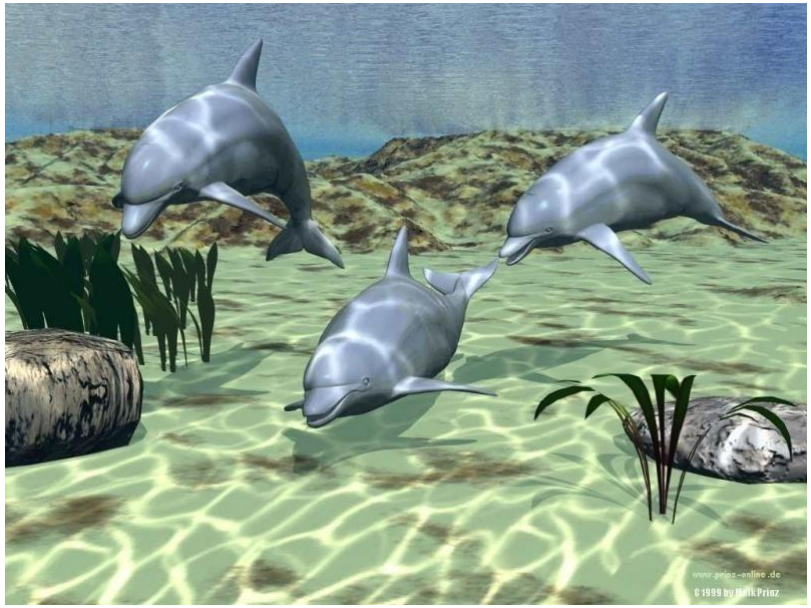
Трехмерная графика, 3D графика

Трехмерная графика - компьютерная графика для отображения изображений, имеющих длину, ширину и глубину.

Трехмерная графика в отличие от двухмерной дает более реалистичное представление образов

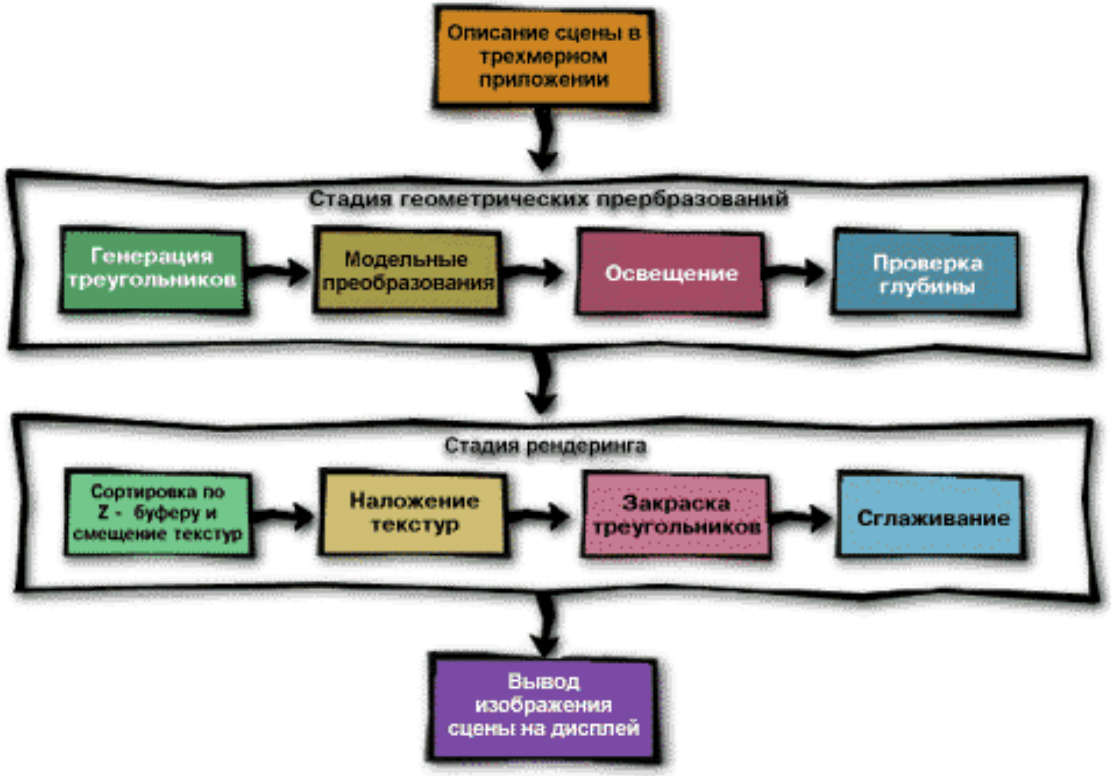
Главной целью системы трехмерного графического синтеза является создание высококачественных фотореалистичных изображений со скоростью видео.



Графический конвейер

В большинстве подсистем трехмерной графики применяется графический конвейер.

Конвейер - это логическая группа вычислений, выполняемых последовательно, которые дают на выходе синтезируемую сцену



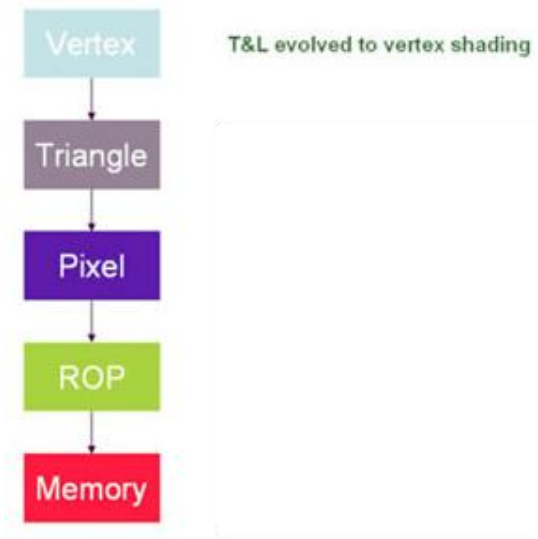
Конвейер разделен на множество этапов, на каждом из которых ***аппаратно*** или ***программно*** выполняется некоторая функция. Наличием переходов между этапами конвейера обеспечивается возможность выбора между программной и аппаратной реализацией очередного этапа.

Такой подход к настройке конвейера позволяет приложениям трехмерной графики получать преимущества аппаратной реализации

Реализация конвейера может чисто ***программной***, полностью ***аппаратной*** или ***смешанной*** (программно-аппаратной)

Классический конвейер

Конвейер в видеопроцессоре, является аппаратным обеспечением для реализации обработки графики. Просчёт графики ведётся следующим способом:



Видеопроцессор получает от хоста (центрального процессора) информацию об объекте, который необходимо обработать. В дело вступает вершинный процессор ядра.

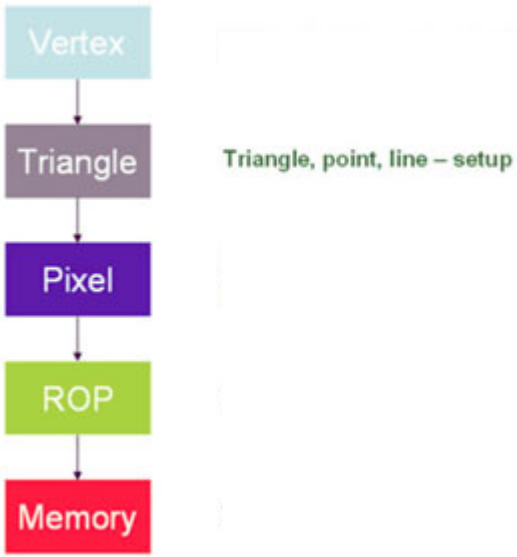
Он на основании полученных данных строит конкретный объект с фиксированными координатами, называемый вершиной (**vertice**).

На данном этапе проводятся отдельные дополнительные операции над этими вершинами – например, преобразование и освещение, изменение объекта шейдерами и т.п.

Аппаратный Transform & Lighting (T&L) впервые был внедрён NVIDIA в ядро GeForce 256 в 1999 году, затем дополнительные возможности обработки вершин добавлялись с поддержкой новых версий DirectX.

Эта технология (T&L) заключается в преобразовании координат вершин в плоские координаты, отображаемые на мониторе, и вычислении их освещенности. Это весьма ресурсоемкие и сложные вычисления, особенно при большом количестве вершин. Ранее они выполнялись на центральном процессоре, что отнимало значительную часть процессорного времени, либо на отдельных процессорах освещения и трансформации.

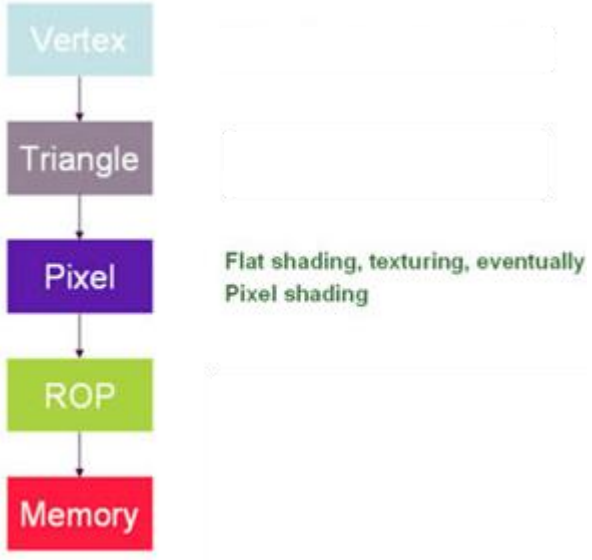
Просчёт графики ведётся следующим способом



Следующая ступень конвейера – сборка (setup). На этом этапе вершины собираются в примитивы – треугольники (полигоны), линии или точки

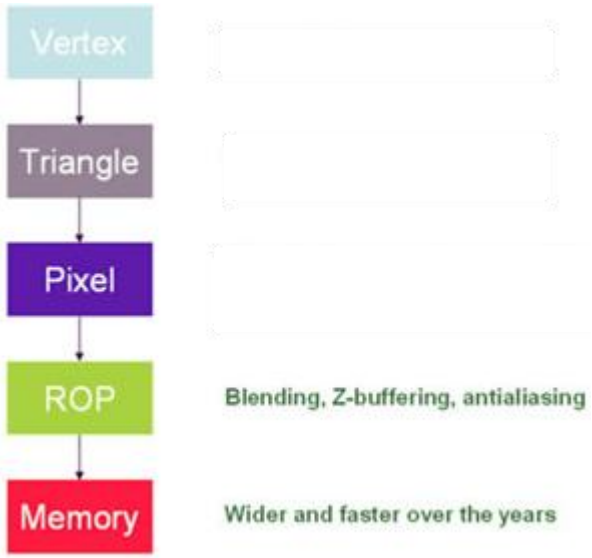
О видимом объекте пока речь не идёт, это абстрактная информация о том, что вершины объединены в какой-то геометрический объект

Эта информация переходит дальше по конвейеру – в пиксельный процессор



Пиксельный процессор, определяет конечные пиксели, которые будут выведены в кадровый буфер, и проводит над ними различные операции: затенение или освещение, текстурирование, присвоение цвета, данных о прозрачности и т.п. В целом эту операцию называют растеризацией, потому что объект разбивается на отдельные фрагменты – пиксели. Понятие «растровый» означает «состоящий из точек».

Также в современных графических процессорах на данной стадии на сцену выходят ***пиксельные шейдеры***, способные отобразить специфическое изображение (шейдеры освещения, затенения, блеска и т.п.) Над пикселями проводится Z-тестирование (выясняется глубина каждой точки, так как мы говорим о трёхмерном изображении), и линии сглаживаются (antialiasing). Вся эта информация передаётся на следующую стадию (Z-данные – в Z-буфер).



Затем фрагменты с присвоенными им координатами цветности (RGBA, где A – alpha, обозначающая дополнительную характеристику пикселя (прозрачность) обрабатываются блоком **ROP** (Raster Operations).

Из Z-буфера вычитываются данные о расположении конкретных пикселей, чтобы отбросить те, которые будут скрыты другими объектами и не видны пользователю.

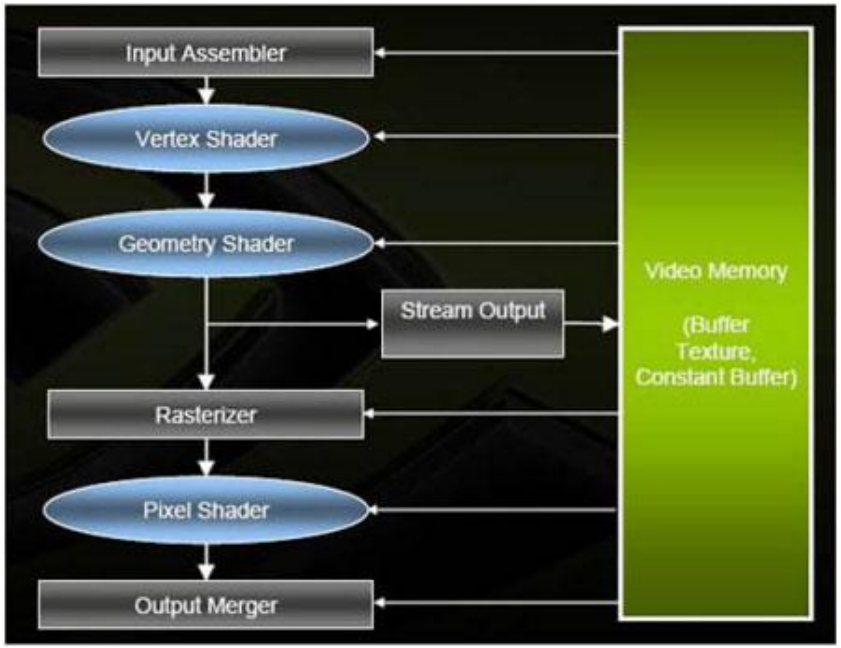
Фрагменты снова собираются в полигоны, состоящие из отдельных пикселей, и весь массив уже обработанной картинки передаётся в кадровый буфер для последующей выборки и вывода на экран.

Проблема подобной организации обработки графики состоит в том, что если уже обработанные данные нужно снова запросить и изменить, приходится дождаться завершения всего конвейера и заново вычитать их из кадрового буфера – или вообще снова получить от хоста.

Если речь идёт об изменении на поздней стадии (например, работа пиксельного шейдера, скажем, освещение), то все предыдущие стадии конвейера заново обрабатываются почем зря, тратя на это вычислительную мощность других блоков.

Также разделение функциональных блоков ядра на отдельные группы (***пиксельный, вершинный процессоры***) очень сильно ограничивает разработчиков графических приложений.

Революционным шагом в порядке обработки данных, которые в дальнейшем станут трёхмерным изображением, является концепция ***потоковой обработки***.



Потоковый вывод данных даёт возможность намного быстрее отослать на повторную обработку данные, уже прошедшие через вершинный или геометрический шейдер, с помощью специального потокового буфера (stream buffer). Благодаря этому не тратится время на ожидание завершения работы пиксельного шейдера и растеризации.

В DirectX 10: введён новый вид шейдеров – ***geometry shaders***. Они позволяют работать с геометрией не на уровне отдельной вершины, как в случае с вершинными шейдерами, а на уровне примитивов. Это означает, что не нужно больше менять каждую вершину, собираемую затем в линию, можно геометрическим шейдером изменить всю линию. Это серьёзно разгружает центральный процессор. Работа с геометрическим шейдером позволит затрачивать намного меньшее количество тактов графического процессора на просчёт сложных геометрических преобразований, в частности, всеми любимых реалистичных волос.

Архитектура на основе унифицированных (универсальных) шейдерных конвейеров способствует увеличению реальной производительности графического процессора.

При использовании специализированных конвейеров, то есть когда одни предназначены исключительно для обработки вершинных (геометрических) шейдеров, а другие — пиксельных, часто возникают ситуации, когда при расчёте сцены основная нагрузка ложится на пиксельные конвейеры, а вершинные частично простаивают. Или наоборот.

Унифицированная архитектура избавлена от этих недостатков, так как каждый шейдерный конвейер может обрабатывать как вершинные, так и пиксельные команды.

Описание сцены

До начала работы геометрических преобразований необходимо описать трехмерную сцену, изображение которой необходимо синтезировать.

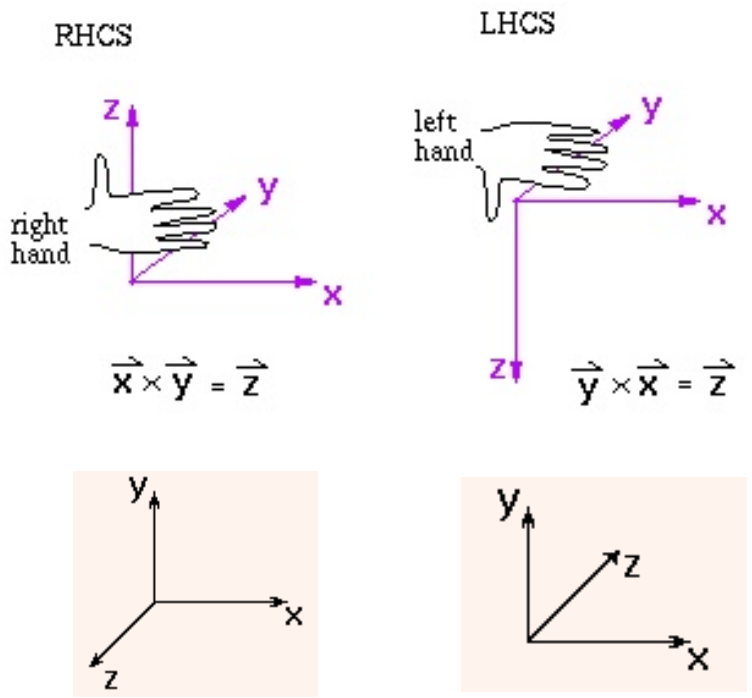
Трехмерное приложение оперирует объектами, описанными в некоторой глобальной системе координат.

Чаще всего здесь используется ортогональная (декартова) система координат, в которой положение каждой точки задается ее расстоянием от начала координат по трем взаимно перпендикулярным осям X, Y и Z.

В некоторых случаях используется и сферическая система координат, в которой положение точки задается удалением от центра и двумя углами направления.

В глобальных координатах приложение создает объекты. В этом же пространстве располагаются источники освещения, а также определяется точка зрения и направление взгляда наблюдателя.

Правосторонняя ***Левосторонняя***



Описание объекта

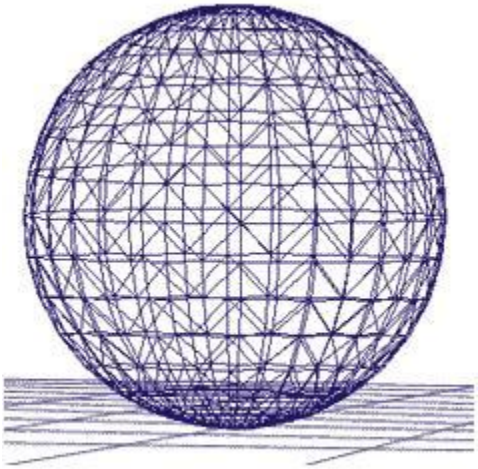
Объекты могут иметь разнообразную форму, описанную каким-либо математическим способом. Проще всего иметь дело с многогранниками, у которых каждая грань представляет собой часть плоскости, ограниченной полигоном. Описание такого тела относительно несложно - оно состоит из упорядоченного списка вершин.

Современные методы компьютерной графики основаны на представлении объекта в виде набора плоских многоугольников (Практически всегда эти многоугольники разбиваются на простейшие **треугольники**. На то есть много причин — удобство работы, ограниченные возможности оборудования, но главная — большинству алгоритмов закраски изображения нужно, чтобы полигоны были плоскими, то есть чтобы все их вершины лежали в одной плоскости. А для треугольников это требование выполняется автоматически).

Генерация треугольников

Объект задается вершинами, определяющими ключевые точки, и полигонами, которые образованы линиями, соединяющими вершины. Такая модель объекта называется ***проволочной***.

```
*MESH_VERTEX 577 21.3728 46.4102 -4.6324
*MESH_VERTEX 578 20.5746 46.4102 -7.0891
*MESH_VERTEX 579 19.4747 46.4102 -9.4265
*MESH_VERTEX 580 18.0906 46.4102 -11.6075
*MESH_VERTEX 581 16.4440 46.4102 -13.5979
*MESH_VERTEX 582 14.5609 46.4102 -15.3662
*MESH_VERTEX 583 12.4711 46.4102 -16.8845
*MESH_VERTEX 584 10.2075 46.4102 -18.1290
*MESH_VERTEX 585 7.8057 46.4102 -19.0799
*MESH_VERTEX 586 5.3037 46.4102 -19.7223
*MESH_VERTEX 587 2.7409 46.4102 -20.0461
*MESH_VERTEX 588 0.1577 46.4102 -20.0461
*MESH_VERTEX 589 -2.4051 46.4102 -19.7223
```

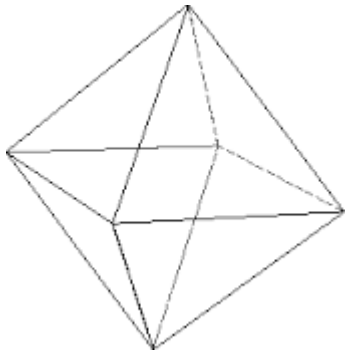


Для написания программы необходимо хранить информации об объектах в следующем виде:

- Координаты вершин хранятся в одномерных массивах X[ ], Y[ ], Z[ ].
- Информация о плоскостях - массив, в котором хранятся номера образующих его вершин.
- Конечный элемент - со значением не попадающим в область допустимых значений

Например в VRML октаэдр можно задать как совокупность точек:

```
Coordinate3 {
    point [
            0          0          1.41421,
          1.41421      0          0,
            0        1.41421      0,
        -1.41421       0          0,
            0        -1.41421     0,
            0          0        -1.41421,
        ]
          }
```



Затем надо указать какие точки составляют плоскость:

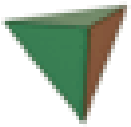
```
IndexedFaceSet {
    coordIndex [
            0,1,2,-1,
            0,2,3,-1,
            0,3,4,-1,
            0,4,1,-1,
            1,4,5,-1,
            1,5,2,-1,
            2,5,3,-1,
            3,5,4,-1,
        ]
    }
```

И если надо какие точки составляют ребра:

```
IndexedLineSet {
    coordIndex [
            0,1,-1,
            0,2,-1,
            0,3,-1,
            0,4,-1,
            1,2,-1,
            1,4,-1,
            1,5,-1,
            2,3,-1,
            2,5,-1,
            3,4,-1,
            3,5,-1,
            4,5,-1, ]
    }
```

В трёхмерном евклидовом пространстве существует всего пять правильных многогранников. Многогранник называется правильным, если:

- он выпуклый;
- все его грани являются равными правильными многоугольниками;
- в каждой его вершине сходится одинаковое число рёбер.



Тетраэдр
Число вершин – 4
Число ребер – 6
Число граней 4

Октаэдр
Число вершин – 6
Число ребер – 12
Число граней 8

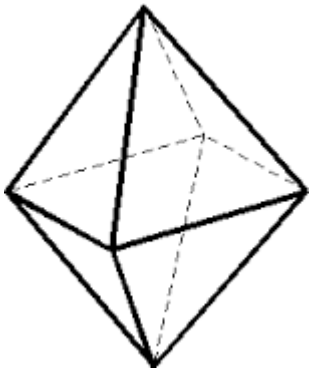
Икосаэдр
Число вершин – 12
Число ребер – 30
Число граней 20

Гексаэдр или куб
Число вершин – 8
Число ребер – 12
Число граней 6

Додекаэдр
Число вершин – 20
Число ребер – 30
Число граней 12

Октаэдр

Число вершин – 6 Число ребер – 12 Число граней 8



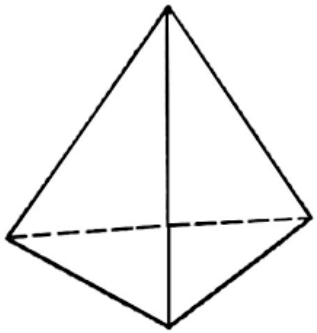
```
Coordinate3 {
point [
0          0          1.41421,
1.41421    0          0,
0          1.41421    0,
-1.41421   0          0,
0          -1.41421   0,
0          0          -1.41421,
    ]
  }
```

```
IndexedFaceSet {
coordIndex [
        0,1,2,-1,
        0,2,3,-1,
        0,3,4,-1,
        0,4,1,-1,
        1,4,5,-1,
        1,5,2,-1,
        2,5,3,-1,
        3,5,4,-1,
        ]
  }
```

```
IndexedLineSet {
    coordIndex [
0,1,-1,
0,2,-1,
0,3,-1,
0,4,-1,
1,2,-1,
1,4,-1,
1,5,-1,
2,3,-1,
2,5,-1,
3,4,-1,
3,5,-1,
4,5,-1,
]
    }
```

Тетраэдр

Число вершин – 4 Число ребер – 6 Число граней 4



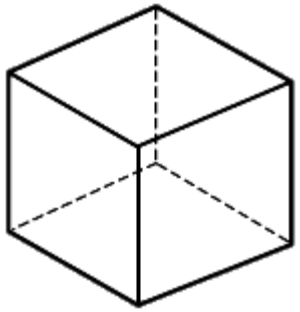
```
Coordinate3 {
point [
1 1 1,
1 -1 -1,
-1 1 -1,
-1 -1 1,
]
    }
```

```
IndexedFaceSet {
    coordIndex [
0,1,2,-1,
0,2,3,-1,
0,3,1,-1,
1,3,2,-1,
]
    }
```

```
IndexedLineSet {
    coordIndex [
0,1,-1,
0,2,-1,
0,3,-1,
1,2,-1,
1,3,-1,
2,3,-1,
]
    }
```

Гексаэдр или куб

Число вершин – 8 Число ребер – 12 Число граней 6



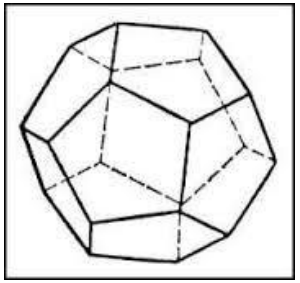
```
Coordinate3 {
point [
0 0 1.22474,
1.1547 0 0.408248,
-0.57735 1. 0.408248,
-0.57735 -1. 0.408248,
0.57735 1. -0.408248,
0.57735 -1. -0.408248,
-1.1547 0 -0.408248,
0 0 -1.22474,
]
    }
```

```
IndexedFaceSet {
    coordIndex [
0,1,4,2,-1,
0,2,6,3,-1,
0,3,5,1,-1,
1,5,7,4,-1,
2,4,7,6,-1,
3,6,7,5,-1,
]
    }
```

```
IndexedLineSet {
    coordIndex [
0,1,-1,
0,2,-1,
0,3,-1,
1,4,-1,
1,5,-1,
2,4,-1,
2,6,-1,
3,5,-1,
3,6,-1,
4,7,-1,
5,7,-1,
6,7,-1,
]
    }
```

Додекаэдр

Число вершин – 20 Число ребер – 30 Число граней 12



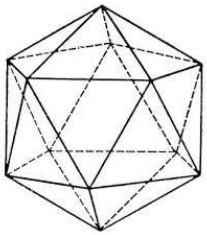
```
Coordinate3 {
point [
0 0 1.07047,
0.713644 0 0.797878,
-0.356822 0.618034 0.797878,
-0.356822 -0.618034 0.797878,
0.797878 0.618034 0.356822,
0.797878 -0.618034 0.356822,
-0.934172 0.381966 0.356822,
0.136294 1. 0.356822,
0.136294 -1. 0.356822,
-0.934172 -0.381966 0.356822,
0.934172 0.381966 -0.356822,
0.934172 -0.381966 -0.356822,
-0.797878 0.618034 -0.356822,
-0.136294 1. -0.356822,
-0.136294 -1. -0.356822,
-0.797878 -0.618034 -0.356822,
0.356822 0.618034 -0.797878,
0.356822 -0.618034 -0.797878,
-0.713644 0 -0.797878,
0 0 -1.07047,
]
}
```

```
IndexedFaceSet {
coordIndex [
0,1,4,7,2,-1,
0,2,6,9,3,-1,
0,3,8,5,1,-1,
1,5,11,10,4,-1,
2,7,13,12,6,-1,
3,9,15,14,8,-1,
4,10,16,13,7,-1,
5,8,14,17,11,-1,
6,12,18,15,9,-1,
10,11,17,19,16,-1,
12,13,16,19,18,-1,
14,15,18,19,17,-1,
]
}
```

```
IndexedLineSet {
coordIndex [
0,1,-1,
0,2,-1,
0,3,-1,
1,4,-1,
1,5,-1,
2,6,-1,
2,7,-1,
3,8,-1,
3,9,-1,
4,7,-1,
4,10,-1,
5,8,-1,
5,11,-1,
6,9,-1,
6,12,-1,
7,13,-1,
8,14,-1,
9,15,-1,
10,11,-1,
10,16,-1,
11,17,-1,
12,13,-1,
12,18,-1,
13,16,-1,
14,15,-1,
14,17,-1,
15,18,-1,
16,19,-1,
17,19,-1,
18,19,-1,
]
}
```

Икосаэдр

Число вершин – 12 Число ребер – 30 Число граней 20



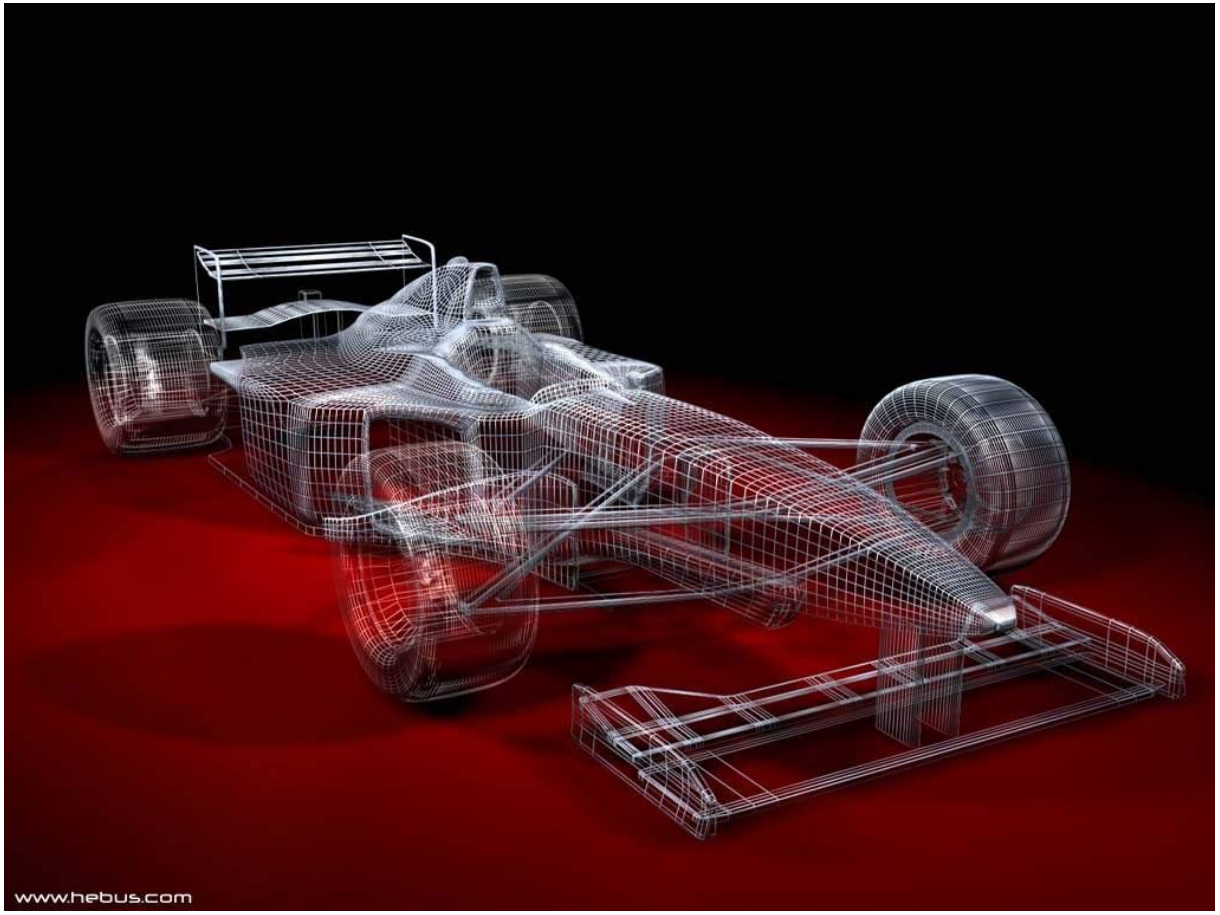
```
Coordinate3 {
point [
0 0 1.17557,
1.05146 0 0.525731,
0.32492 1. 0.525731,
-0.850651 0.618034 0.525731,
-0.850651 -0.618034 0.525731,
0.32492 -1. 0.525731,
0.850651 0.618034 -0.525731,
0.850651 -0.618034 -0.525731,
-0.32492 1. -0.525731,
-1.05146 0 -0.525731,
-0.32492 -1. -0.525731,
0 0 -1.17557,
]
    }
```

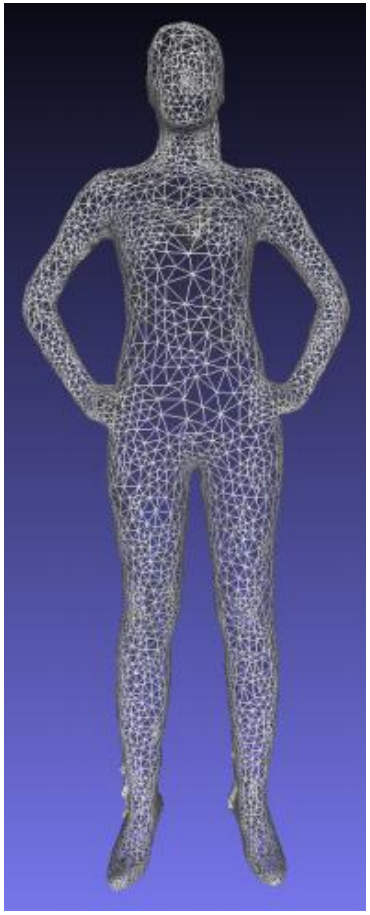
```
IndexedFaceSet {
    coordIndex [
0,1,2,-1,
0,2,3,-1,
0,3,4,-1,
0,4,5,-1,
0,5,1,-1,
1,5,7,-1,
1,7,6,-1,
1,6,2,-1,
2,6,8,-1,
2,8,3,-1,
3,8,9,-1,
3,9,4,-1,
4,9,10,-1,
4,10,5,-1,
5,10,7,-1,
6,7,11,-1,
6,11,8,-1,
7,10,11,-1,
8,11,9,-1,
9,11,10,-1,
]
    }
```

```
IndexedLineSet {
    coordIndex [
0,1,-1,
0,2,-1,
0,3,-1,
0,4,-1,
0,5,-1,
1,2,-1,
1,5,-1,
1,6,-1,
1,7,-1,
2,3,-1,
2,6,-1,
2,8,-1,
3,4,-1,
3,8,-1,
3,9,-1,
4,5,-1,
4,9,-1,
4,10,-1,
5,7,-1,
5,10,-1,
6,7,-1,
6,8,-1,
6,11,-1,
7,10,-1,
7,11,-1,
8,9,-1,
8,11,-1,
9,10,-1,
9,11,-1,
10,11,-1,
]
    }
```

Таким же образом можно задавать и более сложные объекты.

ris//vessel.txt





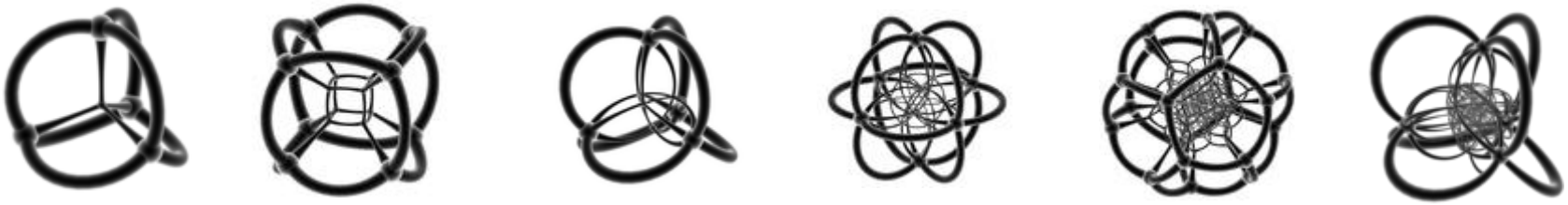
Правильные многогранники характерны для философии Платона, в честь которого и получили название «***платоновы тела***»

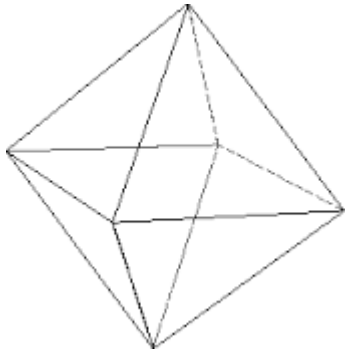
Евклид дал полное математическое описание правильных многогранников в последней, XIII книге Начал.

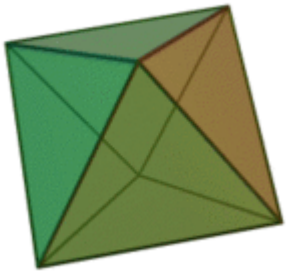
В размерности n = 4

Существует 6 правильных четырёхмерных многогранников:

- 4-мерный симплекс (грань — тетраэдр, символ Шлефли {3, 3, 3}).
- Тессеракт или 4-мерный куб (грань — куб, {4, 3, 3}).
- 16-гранник (грань — тетраэдр, {3, 3, 4}).
- 24-гранник (грань — октаэдр, {3, 4, 3}).
- 120-гранник (грань — додекаэдр, {5, 3, 3}).
- 600-гранник (грань — тетраэдр, {3, 3, 5}).







К сожалению, картинка из одних лишь вершин (и даже вершин, соединенных ребрами) удовлетворит далеко не всех пользователей. Поэтому на стадии рендеринга производится удаление невидимых линий.

OBJ

OBJ — это формат файлов описания геометрии, разработанный в Wavefront Technologies для их анимационного пакета Advanced Visualizer.

Формат файлов OBJ — это простой формат данных, который содержит только 3D геометрию:

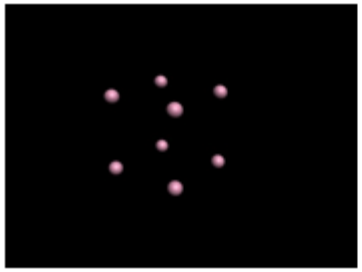
- позицию каждой вершины,
- связь координат текстуры с вершиной,
- нормаль для каждой вершины,
- а также параметры, которые создают полигоны.

Это комментарий

Вершины

Строка начинающаяся с **v** представляет собой описание вершины.

```
#    Список вершин, с координатами (x,y,z[,w]),
#    w является не обязательным и по умолчанию 1.0.

v 0.123 0.234 0.345 1.0
v ...
```



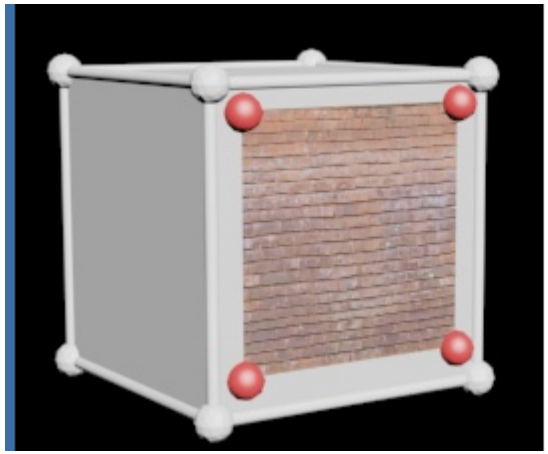
Координаты x y z
Каждой точки

```
# 3ds Max Wavefront OBJ Exporter v0.97b - (c)2007 guruware
# File Created: 02.06.2012 13:38:18

#
# object Box01
#

v  -0.0050 0.0000 0.0050
v  -0.0050 0.0000 -0.0050
v  0.0050 0.0000 -0.0050
v  0.0050 0.0000 0.0050
v  -0.0050 0.0100 0.0050
v  0.0050 0.0100 0.0050
v  0.0050 0.0100 -0.0050
v  -0.0050 0.0100 -0.0050
# 8 vertices
```

Текстуры

Строка начинается с символа vt

На каждую плоскость нашего 3D объекта мы можем натянуть текстуру просто указав координаты картинки как она будет располагаться на плоскости

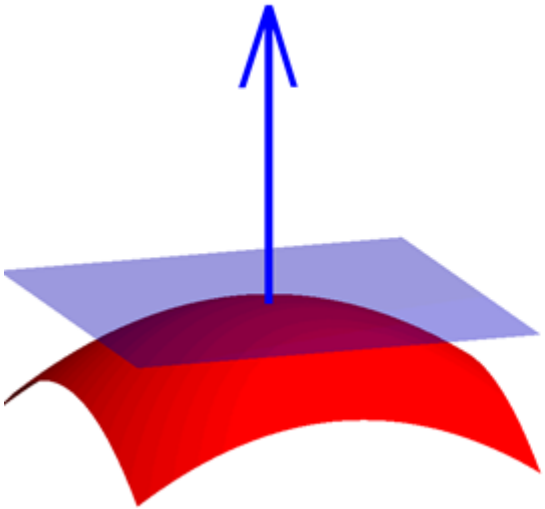


Координаты x y z
Для наложения
текстуры на грань
объекта

```
# 6 vertex normals

vt 1.0000 0.0000 0.0000
vt 1.0000 1.0000 0.0000
vt 0.0000 1.0000 0.0000
vt 0.0000 0.0000 0.0000
# 4 texture coords
```

Нормали

Вектор нормали к поверхности в точке совпадает с нормалью к касательной плоскости в этой точке. По этому параметру вычисляется как будет освещаться конкретная точка объекта.



Координаты x y z каждой нормали Для просчета освещения

```
# 8 vertices

vn 0.0000 -1.0000 -0.0000
vn 0.0000 1.0000 -0.0000
vn 0.0000 0.0000 1.0000
vn 1.0000 0.0000 -0.0000
vn 0.0000 0.0000 -1.0000
vn -1.0000 0.0000 -0.0000
# 6 vertex normals
```

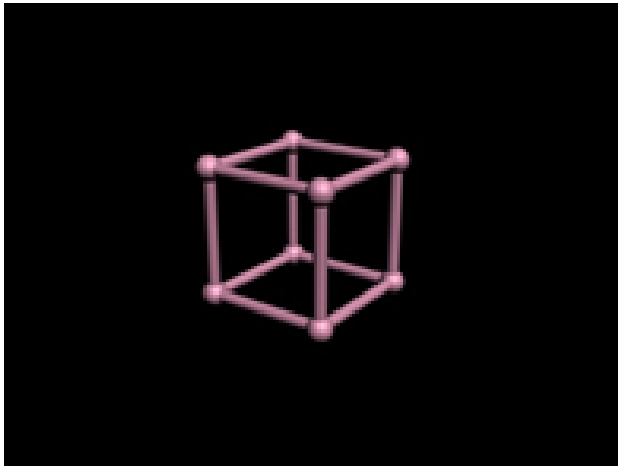
Поверхности

По сути 3D объект это просто набор точек соединённый линиями в правильном порядке.

Строка начинающаяся с **f** представляет собой индекс поверхности.

Каждая поверхность (полигон) может состоять из трех или более вершин.

Индексация начинается с первого элемента, а не с нулевого как принято в некоторых языках программирования, так же индексация может быть отрицательной.



```
g Box01
s 2
f 1/1/1 2/2/1 3/3/1 4/4/1
s 4
f 5/4/2 6/1/2 7/2/2 8/3/2
s 8
f 1/4/3 4/1/3 6/2/3 5/3/3
s 16
f 4/4/4 3/1/4 7/2/4 6/3/4
s 32
f 3/4/5 2/1/5 8/2/5 7/3/5
s 64
f 2/4/6 1/1/6 5/2/6 8/3/6
# 6 polygons
```

Наряду с вершинами могут сохраняться соответствующие индексы текстурных координат.

f v1/vt1 v2/vt2 v3/vt3 v4/vt4 ...

f 3/1 4/2 5/3

Также допустимо сохранение соответствующие индексы нормалей.

f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 v4/vt4/vn4 ...

f 6/4/1 3/5/3 7/6/5

При отсутствии данных о текстурных координатах допустима запись с пропуском индексов текстур.

f 6//1 3//3 7//5

Библиотека материалов

Информация о внешнем виде объектов(материалы) передается в файлах-спутниках в формате MTL (Material Library). OBJ при необходимости ссылается на такой файл с помощью директивы:

mtllib [имя внешнего MTL файла]

MTL является стандартом, установленным компанией Wavefront Technologies. Вся информация представлена в ASCII виде и абсолютно читабельна для человека.

Информация о простых материалах в файле выглядит следующим образом:

```
newmtl название_материала1 # Объявление очередного материала
# Цвета

Ka 1,000 1,000 0,000      # Цвет окружающего освещения (желтый)
Kd 1,000 1,000 1,000      # Диффузный цвет (белый)

# Параметры отражения

Ks 0,000 0,000 0,000      # Цвет зеркального отражения (0;0;0 - выключен)
Ns 10,000                 # Коэффициент зеркального отражения (от 0 до 1000)

# Параметры прозрачности

 d 0,9                    # Прозрачность указывается с помощью директивы d
 Tr 0,9                   # или в других реализациях формата с помощью Tr

#Следующий материал newmtl название_материала2
 ...
```

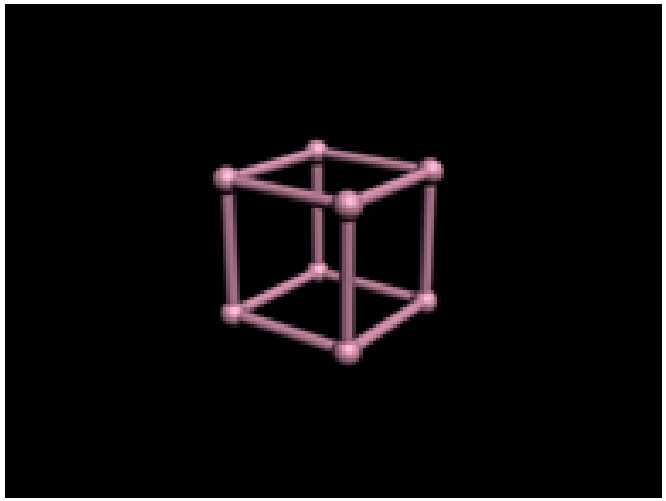
Существует несколько наборов параметров описания поверхности.

- Файл хранит только координаты точек и как они связанны между собой.
- Файл хранит координаты точек и координаты нормалей для поверхности.
- Файл хранит координаты точек и координаты текстур.
- Файл хранит полный список параметров

1. Файл хранит только координаты точек и как они связанны между собой

cube_polygon.obj

```
# 3ds Max Wavefront OBJ Export
# File Created: 07.12.2013 14:

#
# object Box001
#

v  -0.0005 0.0000 0.0005
v  -0.0005 0.0000 -0.0005
v  0.0005 0.0000 -0.0005
v  0.0005 0.0000 0.0005
v  -0.0005 0.0010 0.0005
v  0.0005 0.0010 0.0005
v  0.0005 0.0010 -0.0005
v  -0.0005 0.0010 -0.0005
# 8 vertices

g Box001
f 1 2 3 4
f 5 6 7 8
f 1 4 6 5
f 4 3 7 6
f 3 2 8 7
f 2 1 5 8
# 6 polygons
```

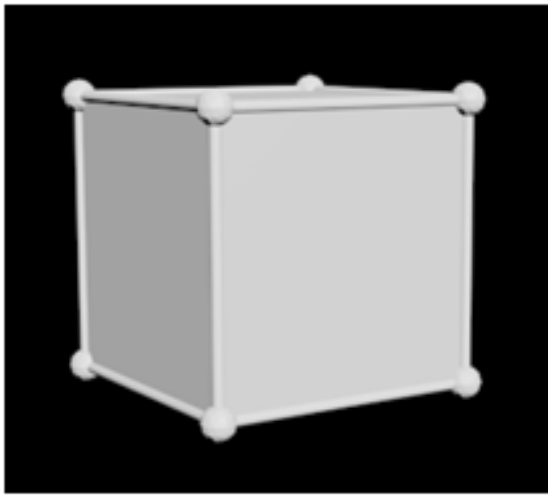


2. Файл хранит координаты точек и координаты нормалей для поверхности

cube_polygon_normal.obj

```
# 3ds Max Wavefront OBJ Expor
# File Created: 07.12.2013 14

#
# object Box001
#

v  -0.0005 0.0000 0.0005
v  -0.0005 0.0000 -0.0005
v  0.0005 0.0000 -0.0005
v  0.0005 0.0000 0.0005
v  -0.0005 0.0010 0.0005
v  0.0005 0.0010 0.0005
v  0.0005 0.0010 -0.0005
v  -0.0005 0.0010 -0.0005
# 8 vertices

vn 0.0000 -1.0000 -0.0000
vn 0.0000 1.0000 -0.0000
vn 0.0000 0.0000 1.0000
vn 1.0000 0.0000 -0.0000
vn 0.0000 0.0000 -1.0000
vn -1.0000 0.0000 -0.0000
# 6 vertex normals

g Box001
f 1//1 2//1 3//1 4//1
f 5//2 6//2 7//2 8//2
f 1//3 4//3 6//3 5//3
f 4//4 3//4 7//4 6//4
f 3//5 2//5 8//5 7//5
f 2//6 1//6 5//6 8//6
# 6 polygons
```

ПОЛИГОНЫ



3. Файл хранит координаты точек и координаты текстур

cube_polygon_texture.obj

```
# 3ds Max Wavefront OBJ Exporter v0.97
# File Created: 07.12.2013 14:17:03

#
# object Box001
#

v  -0.0005 0.0000 0.0005
v  -0.0005 0.0000 -0.0005
v  0.0005 0.0000 -0.0005
v  0.0005 0.0000 0.0005
v  -0.0005 0.0010 0.0005
v  0.0005 0.0010 0.0005
v  0.0005 0.0010 -0.0005
v  -0.0005 0.0010 -0.0005
# 8 vertices

vt 1.0000 0.0000 0.0000
vt 1.0000 1.0000 0.0000
vt 0.0000 1.0000 0.0000
vt 0.0000 0.0000 0.0000
# 4 texture coords

g Box001
f 1/1 2/2 3/3 4/4
f 5/4 6/1 7/2 8/3
f 1/4 4/1 6/2 5/3
f 4/4 3/1 7/2 6/3
f 3/4 2/1 8/2 7/3
f 2/4 1/1 5/2 8/3
# 6 polygons
```

3. Файл хранит полный список параметров

cube_polygon_texture_normal.obj

```
# 3ds Max Wavefront OBJ Exporter v0.97b -
# File Created: 07.12.2013 14:17:57

#
# object Box001
#

v  -0.0005 0.0000 0.0005
v  -0.0005 0.0000 -0.0005
v  0.0005 0.0000 -0.0005
v  0.0005 0.0000 0.0005
v  -0.0005 0.0010 0.0005
v  0.0005 0.0010 0.0005
v  0.0005 0.0010 -0.0005
v  -0.0005 0.0010 -0.0005
# 8 vertices

vn 0.0000 -1.0000 -0.0000
vn 0.0000 1.0000 -0.0000
vn 0.0000 0.0000 1.0000
vn 1.0000 0.0000 -0.0000
vn 0.0000 0.0000 -1.0000
vn -1.0000 0.0000 -0.0000
# 6 vertex normals

vt 1.0000 0.0000 0.0000
vt 1.0000 1.0000 0.0000
vt 0.0000 1.0000 0.0000
vt 0.0000 0.0000 0.0000
# 4 texture coords

g Box001
f 1/1/1 2/2/1 3/3/1 4/4/1
f 5/4/2 6/1/2 7/2/2 8/3/2
f 1/4/3 4/1/3 6/2/3 5/3/3
f 4/4/4 3/1/4 7/2/4 6/3/4
f 3/4/5 2/1/5 8/2/5 7/3/5
f 2/4/6 1/1/6 5/2/6 8/3/6
# 6 polygons
```