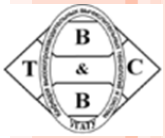
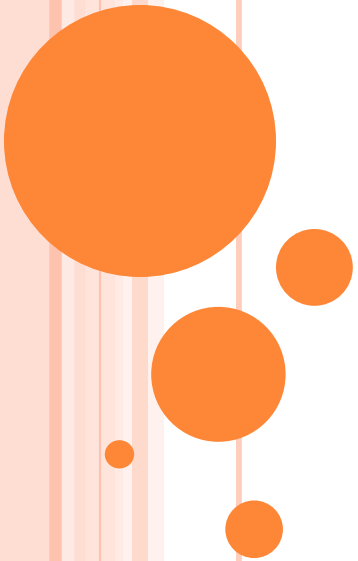


Компьютерная графика. Лекция 6



Основы OpenGL

OpenGL

Open Graphics **L**ibrary — спецификация, определяющая платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

Спецификация OpenGL разрабатывается наблюдательным советом за архитектурой OpenGL (OpenGL Architecture Review Board), в который входят ведущие производители программных и аппаратных средств, в частности, фирмы IBM, Hewlett-Packard, Intel, Apple, Dell, Sun Microsystems.

Спецификация OpenGL

OpenGL — спецификация, то есть документ, описывающий набор функций и их точное поведение.

Производители оборудования на основе этой спецификации создают реализации — библиотеки функций, соответствующих набору функций спецификации.

Если аппаратура не позволяет реализовать какую-либо возможность, она должна быть эмулирована программно.

Развитие OpenGL

- OpenGL – 1992 год
- OpenGL 2.0 - сентябрь 2001 года
- OpenGL 2.1 - июль 2006 года
- OpenGL 3.0 - август 2008 года
- OpenGL 3.1 - март 2009 года
- OpenGL 3.2 - август 2009 года
- OpenGL 3.3, OpenGL 4.0 - март 2010 года
- OpenGL 4.1 - июль 2010 года
- OpenGL 4.3 август 2012 года
- OpenGL 4.4 - июль 2013 года
- OpenGL 4.5 - август 2014 года
- OpenGL 4.6 - июль 2017 года

Использование OpenGL

Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов.

Используется при создании:

- компьютерных игр,
- САПР,
- виртуальной реальности,
- визуализации в научных исследованиях.

На платформе Windows конкурирует с DirectX.

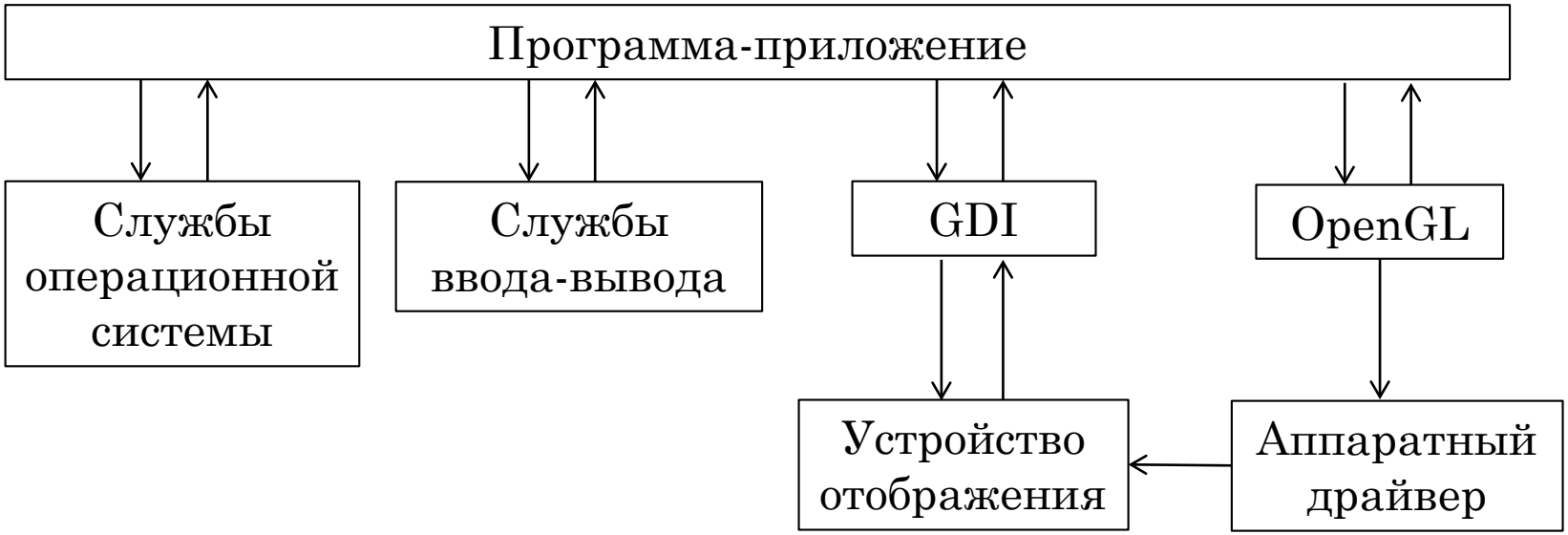
Реализации OpenGL

Эффективные реализации OpenGL существуют для:

- Windows,
- Unix-платформ,
- PlayStation 3
- Mac OS.

Реализации обычно предоставляются изготовителями видеоадаптеров и активно используют возможности последних.

OpenGL с аппаратным ускорением



Основные задачи OpenGL

- Скрыть сложности адаптации различных 3D-ускорителей, предоставляя разработчику единый API.

- Скрыть различия в возможностях аппаратных платформ, требуя реализации недостающей функциональности с помощью программной эмуляции.

Достоинства

- Промышленный стандарт
- Стабильность
- Надежность и переносимость
- Современность
- Масштабируемость
- Легкость в изучении и использовании
- Хорошо документирован
- Независим от языка программирования

Основной принцип работы OpenGL

Получение наборов векторных графических примитивов в виде точек, линий и треугольников с последующей математической обработкой полученных данных и построением растровой картинки на экране и/или в памяти.

Базовые возможности

- Рисование геометрических примитивов (точки, линии, многоугольники)
- Работа с растровыми примитивами
- Цветовые режимы RGBA / Index
- Видовые и модельные преобразования
- Прозрачность
- Удаление невидимых линий и поверхностей
- B-сплайны
- Наложение текстуры
- Интерполяция цветов, Anti-aliasing, туман
- Использование списков изображений (display lists)

Основы

- Ориентированность главным образом на построение изображения в буфере кадра (Frame Buffer) и чтение из него
- Основное назначение OpenGL - интерактивная визуализация трехмерных сцен

Библиотеки OpenGL

- Графическая библиотека OpenGL позволяет рисовать графические примитивы в различных режимах
- Примитивами являются точки, отрезки прямых, многоугольники или растровые прямоугольники
- Переключение режимов рисования, отображение примитивов и другие операции GL описываются при помощи вызовов процедур и функций

Стандартные библиотеки OpenGL

Существует ряд библиотек, созданных поверх или в дополнение к OpenGL

- GLU – стандартное дополнение OpenGL
- GLUT и SDL созданы для реализации возможностей, недоступных в OpenGL:
  - создание интерфейса пользователя (окна, кнопки, меню и др.),
  - настройка контекста рисования (область рисования, использующаяся OpenGL)
  - обработка сообщений от устройств ввода-вывода (клавиатура, мышь и др.)
  - работа с файлами.

Оконные библиотеки OpenGL

Каждый оконный менеджер имеет собственную библиотеку:

- WGL в Windows
- GLX в X Window System.

Библиотеки GLEW (The OpenGL Extension Wrangler Library) и GLEE (The OpenGL Easy Extension library) созданы для облегчения работы с расширениями и различными версиями OpenGL.

Специализированные библиотеки OpenGL

- Open Inventor и VTK позволяют оперировать сложными трёхмерными объектами, что облегчает и ускоряет создание трёхмерной сцены.
- GLM (OpenGL Mathematics) — вспомогательная библиотека, предоставляющая программистам на C++ классы и функции для выполнения математических операций

OpenGL на платформе .NET

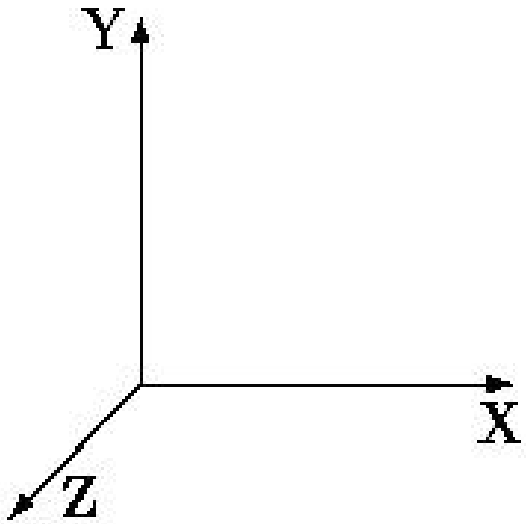
Существует несколько библиотек, позволяющие упростить использование библиотеки OpenGL на платформе Microsoft .NET Framework.
Наиболее широко известны библиотеки Tao Framework и Open Toolkit Library.

Библиотека Open Toolkit Library (OpenTK)

- поддерживает последние версии библиотеки OpenGL и имеет удобный интерфейс вызова функций
- поддерживает не только Microsoft.NET Framework, но и Mono Framework (проект с открытым исходным кодом кросплатформенной реализации Microsoft .NET Framework

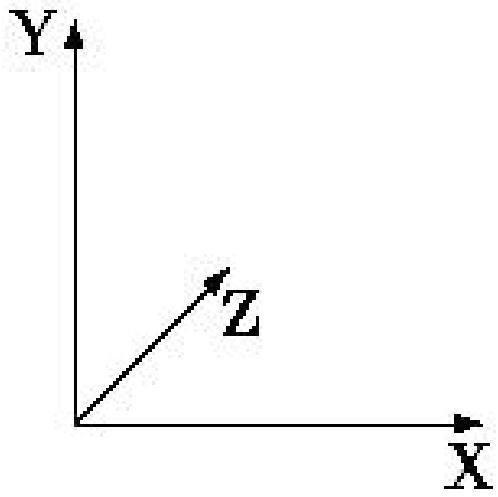
Системы координат OpenGL

Задание координат примитивов OpenGL выполняется в мировой (правосторонней) системе координат, в которой ось OZ направлена на наблюдателя.



Системы координат OpenGL

Для построения проекции используется левосторонняя система координат (видовая), в которой ось OZ направлена от наблюдателя.

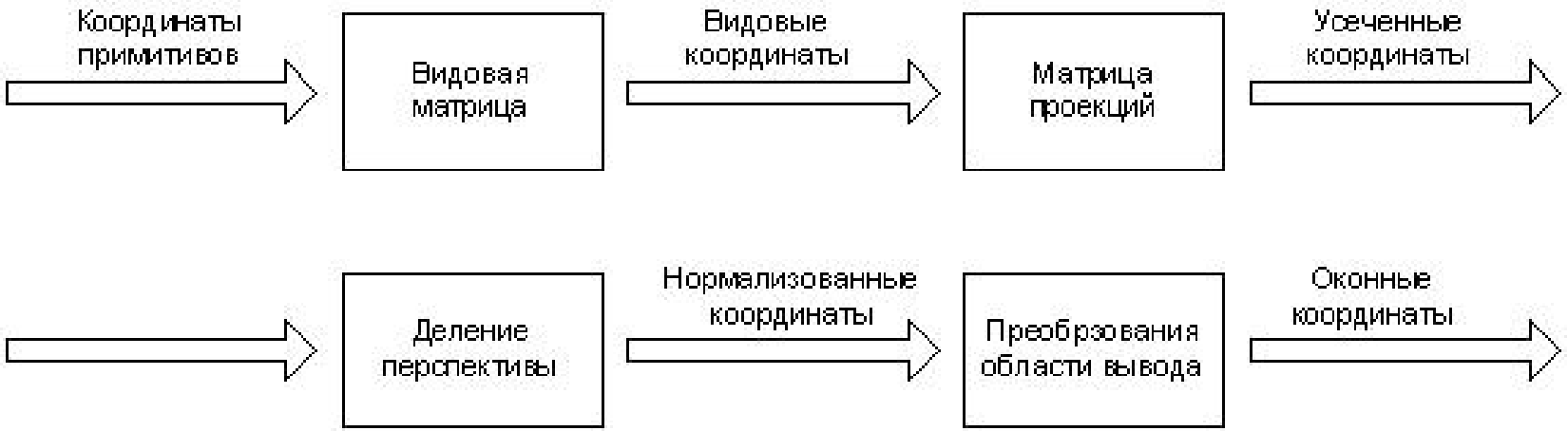


Матрицы преобразований в OpenGL

Преобразование одной системы координат в другую библиотека выполняет с помощью матрицы проекции.

Для задания координат объектов в OpenGL используются однородные координаты. Это позволяет представлять различные преобразования в виде квадратных матриц размером 4*4, а сложные преобразования представлять в виде произведения матриц, реализующих более простые преобразования.

Схема координатных преобразований



Видовые преобразования

Координаты вершин примитивов умножаются на текущую видовую матрицу. Результатом умножения являются видовые координаты вершин:

$$\begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix} = M \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{pmatrix}$$

Обычно видовая матрица используется для таких преобразований как перенос, масштабирование и поворот. Для выполнения этих операций библиотека реализует команды Translate, Scale, Rotate.

Перспективные преобразования

На этом этапе координатных преобразований видовые координаты вершин (полученные на предыдущем этапе вычислений) умножаются на матрицу проекций P.

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = P \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

В библиотеке реализованы две команды для формирования наиболее часто используемых проекций:

- Ortho – формирования матрицы параллельной проекции
- Frustum – получения матрицы центральной перспективной проекции

Деление перспективы

На третьем этапе выполняется преобразование нормализованных координат вершин к декартовым координатам, путем деления каждой из первых трех координат на четвёртую:

$$\begin{pmatrix} x_d \\ y_d \\ z_d \end{pmatrix} = \begin{pmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \end{pmatrix}$$

Преобразования области вывода

На четвертом этапе выполняется преобразование декартовых координат, полученных на предыдущем этапе, к координатам устройства вывода графической информации.

Кроме преобразования масштабирования координат к координатам устройства вывода, на этом этапе обычно выполняется округление полученных координат до целочисленных значений.
В ряде случаев (например, для отображения на мониторе) выполняется изменение направления осей координат.

Этапы использования OpenGL

В состав приложения, формирующего изображение с помощью OpenGL, должно входить несколько программных блоков:

- Инициализация библиотеки OpenGL
- Подготовительные операции для формирования изображения библиотекой OpenGL (при необходимости)
- Определение области вывода изображения
- Основная функция формирования изображения
- Обработка событий мыши и клавиатуры для реализации интерактивности в программе (при необходимости)
- Завершение работы с библиотекой OpenGL

Инициализация OpenGL в библиотеке OpenTK

Инициализацию OpenGL в библиотеке OpenTK выполняет графический компонент GLControl, который размещается на одной из форм приложения.
Вывод изображения, формируемого библиотекой OpenGL, выполняется в области компонента GLControl.

Дополнительные настройки библиотеки OpenGL необходимо выполнять в обработчике события Load формы.

Определение области вывода изображения и матрицы проекции

Область вывода изображения определяется с помощью статического метода класса GL:

```
void Viewport(int x, int y, int width, int height);
```

- x, y – координаты левой верхней вершины прямоугольной области отображения;
- width, height – ширина и высота области вывода.

координаты области вывода задаются относительно компонента GLControl и область вывода ограничивается размерами этого компонента.

Основная функция формирования изображения

Формирование изображения выполняется при каждом обновлении изображения компонента GLControl библиотеки OpenTK.

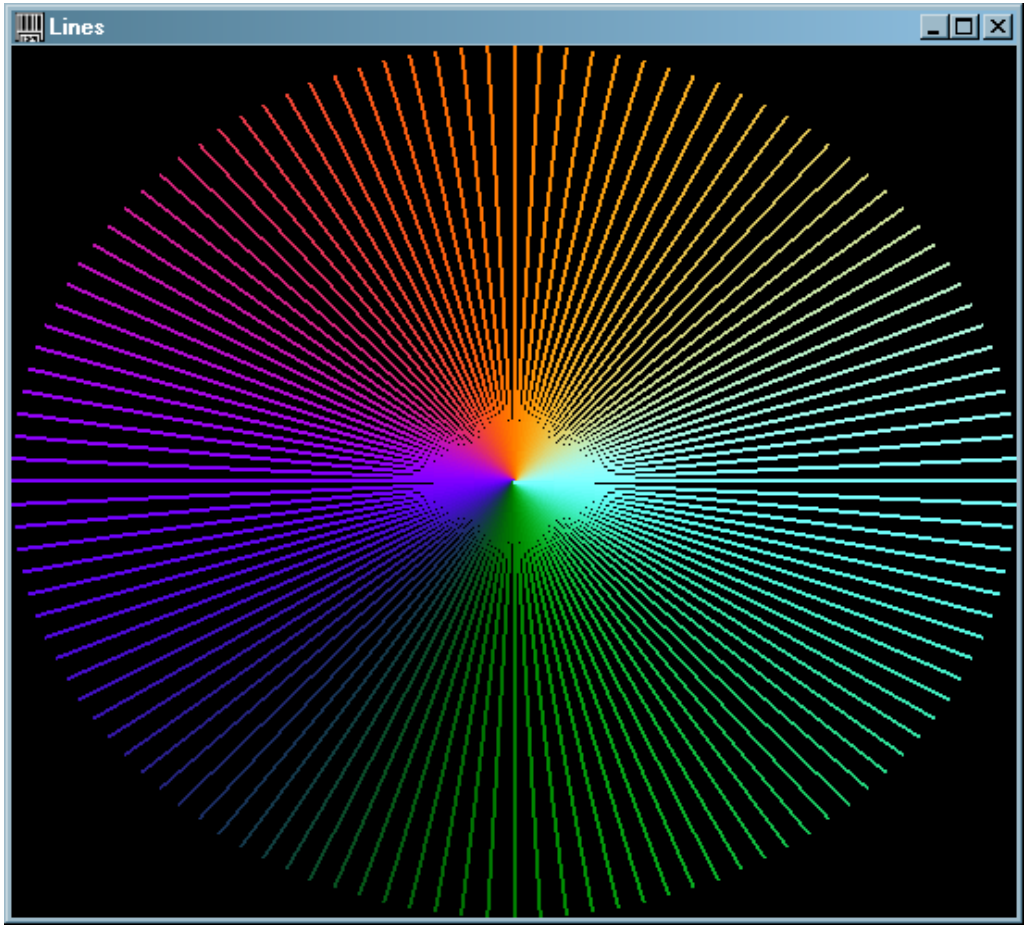
Обычно для этого используется обработчик события Paint компонента GLControl.

Пример функции формирования изображения

```
float x0 = 0;
float y0 = 0;

GL.Begin(BeginMode.Lines);
for (int angle = 0; angle < 360; angle += 3)
{
float angleInRadians = (float)(angle * M_PI / 180.0f);
      float x1 = cosf(angleInRadians);
      float y1 = sinf(angleInRadians);
      GL.Vertex2(x0, y0);
      GL.Vertex2(x1, y1);
}
GL.End();
```

Результат



Примитивы OpenGL

- В OpenGL все изображения формируются из графических примитивов.
- Основные примитивы: точки, отрезки, треугольники, четырехугольники, многоугольники.
- Базовым графическим примитивом для построения сложных объёмных изображений является треугольник.
- Необходимая детализация сложных изображений достигается за счет уменьшения размера примитивов, из которых строится изображение, и увеличения их количества.

Формирование примитивов

Примитив начинается с вызова команды Begin и передачи ей в качестве значения параметра mode целочисленной константы, которая определяет формируемый примитив:

```
void Begin(BeginMode mode);
```

Примитив завершается с помощью команды End:

```
void End();
```

Формирование примитивов (продолжение)

Все примитивы состоят из набора вершин. Для задания расположения вершин в пространстве используются нормализованные координаты.

Координаты каждой вершины определяются с помощью команды OpenGL

```
Vertex[2 3 4][s i f d]

void Vertex2(double x, double y);
void Vertex3(double x, double y, double z);
```

Примитив «точка»

Для формирования точек необходимо указать:

```
GL.Begin(PrimitiveType.Points);
```

Перед формированием точек могут быть указаны их размер и определён режим сглаживания, который будет использоваться при их отображении.
Эти параметры должны быть установлены перед вызовом команды Begin.

```
// задание размера точки
void PointSize(float size);
```

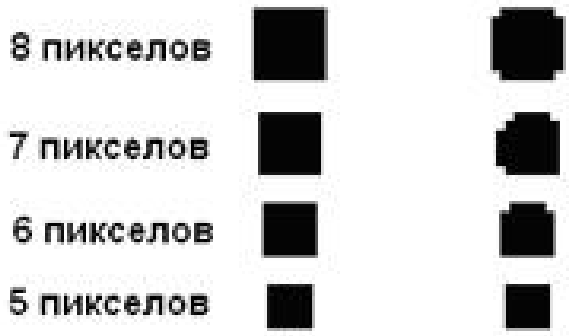
Сглаживание точек

Управление режимом сглаживания точки осуществляется с помощью команд Enable и Disable:

```
void Enable(EnableCap cap);
void Disable(EnableCap cap);
```

Для управления режимом сглаживания необходимо указать значение PointSmooth перечисления EnableCap.
При отключенном режиме сглаживания точка рисуется в виде квадрата.
При включенном режиме сглаживания библиотека пытается изобразить точку в виде круга.

Сглаживание точек



Изображения точки разного размера с использованием режима сглаживания и без него

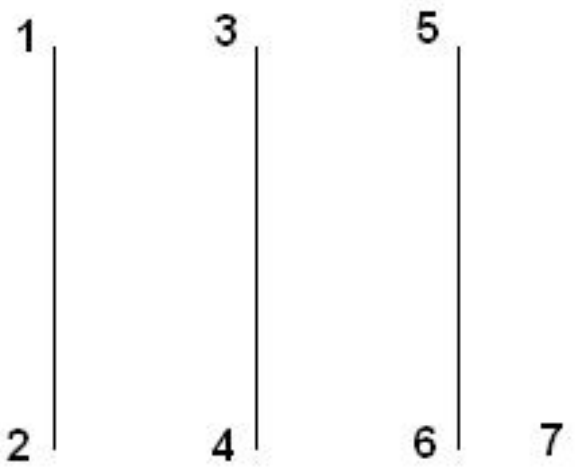
Отрезки

Библиотека реализует три примитива для формирования отрезков:

- несвязанные отрезки
- ломаные линии
- замкнутая ломаная

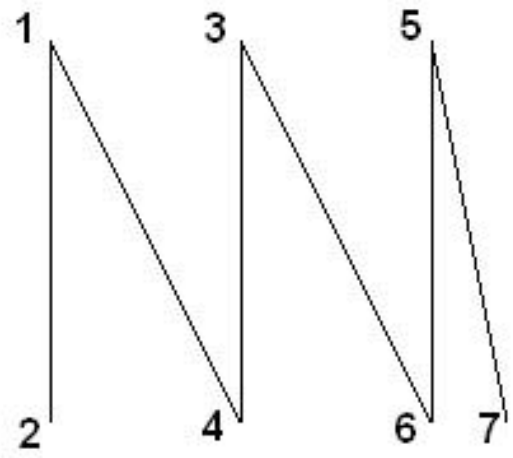
Несвязанные отрезки

```
GL.Begin(BeginMode.Lines);
  GL.Vertex3(-0.5, -0.5, 0);
  GL.Vertex3(-0.5, 0.5, 0);
  GL.Vertex3(0, -0.5, 0);
  GL.Vertex3(0, 0.5, 0);
  GL.Vertex3(0.5, -0.5, 0);
  GL.Vertex3(0.5, 0.5, 0);
  GL.Vertex3(0.7, -0.5, 0);
GL.End()
```



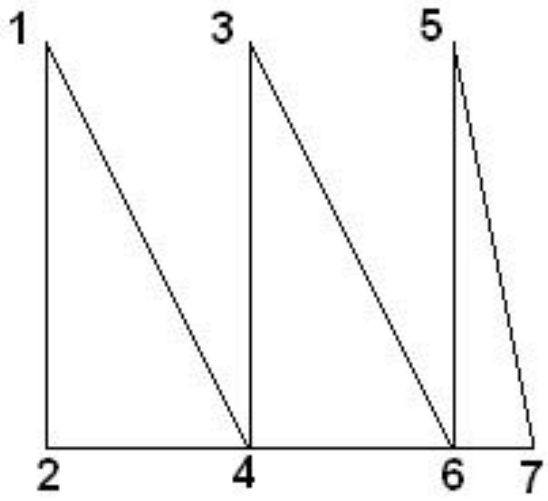
Ломаная линия

```
GL.Begin(BeginMode.LineStrip);
  GL.Vertex3(-0.5, -0.5, 0);
  GL.Vertex3(-0.5, 0.5, 0);
  GL.Vertex3(0, -0.5, 0);
  GL.Vertex3(0, 0.5, 0);
  GL.Vertex3(0.5, -0.5, 0);
  GL.Vertex3(0.5, 0.5, 0);
  GL.Vertex3(0.7, -0.5, 0);
GL.End()
```



Замкнутая ломаная

```
GL.Begin(BeginMode.LineLoop);
  GL.Vertex3(-0.5, -0.5, 0);
  GL.Vertex3(-0.5, 0.5, 0);
  GL.Vertex3(0, -0.5, 0);
  GL.Vertex3(0, 0.5, 0);
  GL.Vertex3(0.5, -0.5, 0);
  GL.Vertex3(0.5, 0.5, 0);
  GL.Vertex3(0.7, -0.5, 0);
GL.End()
```



Многоугольники

Многоугольник представляет собой замкнутую форму, поверхность которой может быть закрашена и границы которой определяются набором вершин, соединённых отрезками.

Многоугольники являются основными компонентами, из которых формируются трехмерные закрашенные объекты.

Библиотека позволяет формировать несколько типов многоугольников: треугольники, четырехугольники, многоугольники.

Свойства многоугольников

У всех многоугольников выделяют лицевую и обратную грани.

Библиотека позволяет для каждой из граней задавать различные свойства, например:

- свойства материала,
- видимость,
- освещённость,
- режим отображения.

По умолчанию лицевой гранью является та, которая получается при обходе вершин против часовой стрелки.

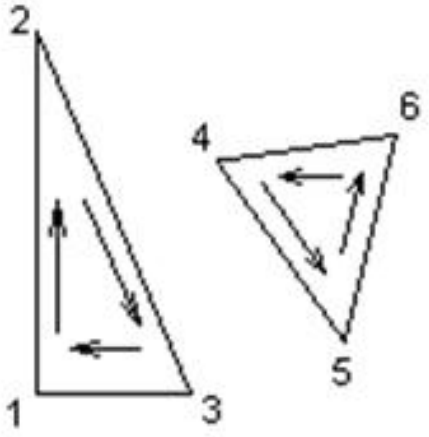
Треугольники

Библиотека реализует три примитива для рисования треугольников:

- несвязанные треугольники,
- треугольники с общей гранью,
- треугольники с общей вершиной.

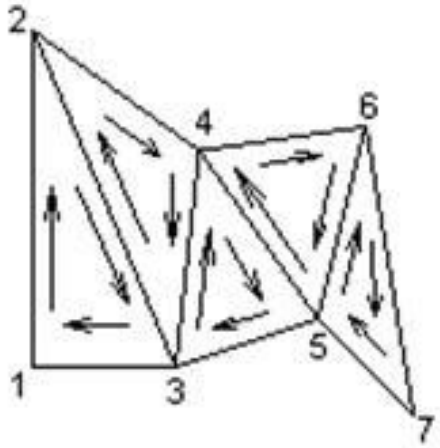
Несвязанные треугольники

```
GL.Begin(BeginMode.Triangles);
// треугольник 1
GL.Vertex3(-0.7, -0.7, 0);
GL.Vertex3(-0.7, 0.7, 0);
GL.Vertex3(-0.1, -0.7, 0);
// треугольник 2
GL.Vertex3(0.0, 0.2, 0);
GL.Vertex3(0.5, -0.5, 0);
GL.Vertex3(0.7, 0.3, 0);
// отдельная вершина игнорируется
GL.Vertex3(0.9, -0.9, 0);
GL.End()
```



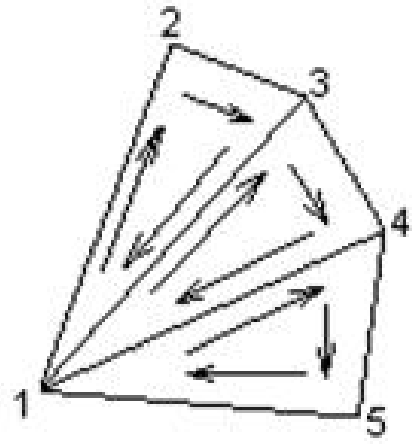
Треугольники с общей гранью

```
GL.Begin(BeginMode.TriangleStrip);
// треугольник 1
GL.Vertex3(-0.7, -0.7, 0);
GL.Vertex3(-0.7, 0.7, 0);
GL.Vertex3(-0.1, -0.7, 0);
// треугольник 2
GL.Vertex3(0.0, 0.2, 0);
GL.Vertex3(0.5, -0.5, 0);
GL.Vertex3(0.7, 0.3, 0);
// отдельная вершина игнорируется
GL.Vertex3(0.9, -0.9, 0);
GL.End()
```



Треугольники с общей вершиной

```
GL.Begin(BeginMode.TriangleFan);
GL.Vertex3(-0.5, -0.5, 0);
GL.Vertex3(0, 0.8, 0);
GL.Vertex3(0.5, 0.6, 0);
GL.Vertex3(0.8, 0.1, 0);
GL.Vertex3(0.7, -0.6, 0);
GL.End();
```



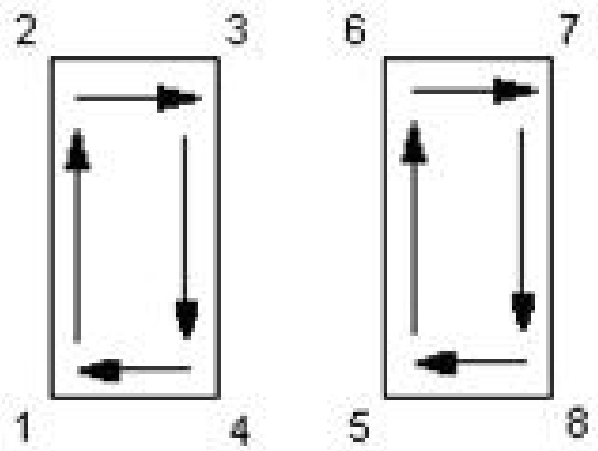
Четырехугольники

Библиотека OpenGL реализуется два примитива для рисования четырехугольников:

- несвязанные четырехугольники
- четырехугольники с общей гранью

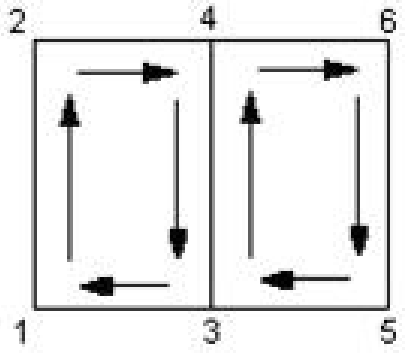
Несвязанные четырехугольники

```
GL.Begin(BeginMode.Quads);
// первый четырехугольник
GL.Vertex3(-0.9, -0.6, 0);
GL.Vertex3(-0.9, 0.6, 0);
GL.Vertex3(-0.3, 0.6, 0);
GL.Vertex3(-0.3, -0.6, 0);
// второй четырехугольник
GL.Vertex3(0.3, -0.6, 0);
GL.Vertex3(0.3, 0.6, 0);
GL.Vertex3(0.9, 0.6, 0);
GL.Vertex3(0.9, -0.6, 0);
GL.End();
```



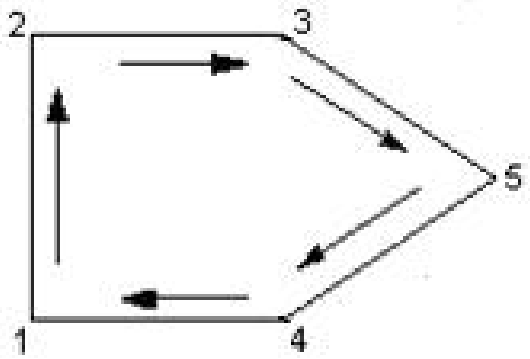
Четырехугольники с общей гранью

```
GL.Begin(BeginMode.QuadStrip);
for (int i = 0; i <= 2; ++i)
{
      GL.Vertex3(-0.9 + 0.8 * i, -0.6, 0);
      GL.Vertex3(-0.9 + 0.8 * i, 0.6, 0);
}
GL.End();
```



Многоугольники

```
GL.Begin(BeginMode.Polygon);
GL.Vertex3(-0.9, -0.6, 0);
GL.Vertex3(-0.9, 0.6, 0);
GL.Vertex3(0, 0.6, 0.3);
GL.Vertex3(0.9, 0, 0);
GL.Vertex3(0, -0.6, -0.3);
GL.End();
```



Режим отображения граней многоугольника

```
void PolygonMode(MaterialFace face, PolygonMode mode);
```

Назначение параметра	Значение параметра face
Лицевые грани	MaterialFace.Front
Обратные грани	MaterialFace.Back
Лицевые и обратные грани	MaterialFace.FrontAndBack

Назначение параметра	Значение параметра mode
грань будет отображаться в виде точек	PolygonMode.Point
грань будет отображаться в виде отрезков, соединяющих вершины	PolygonMode.Line
грань будет закрашиваться	PolygonMode.Fill

Режим отбора граней

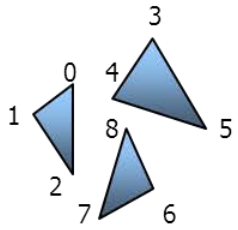
Определяет какие из граней (лицевые, обратные) будут отображаться.

```
void CullFace(CullFaceMode mode);
```

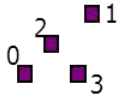
Назначение параметра	Значение параметра
Не будут отображаться только лицевые грани	MaterialFace.Front
Не будут отображаться только обратные грани	MaterialFace.Back
Не будут отображаться и лицевые и обратные грани, т.е. многоугольник отображаться не будет	MaterialFace.FrontAndBack

Режим отбора граней позволяет в ряде случаев уменьшить время формирования изображения за счет отключения формирования граней, которые никогда не видимы.

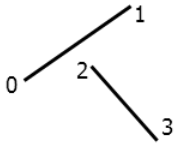
Типы геометрических примитивов OpenGL



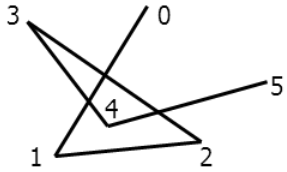
GL_TRIANGLES



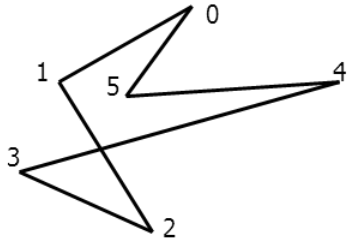
GL_POINTS



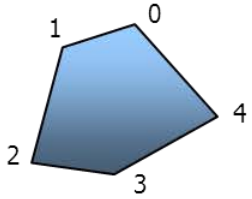
GL_LINES



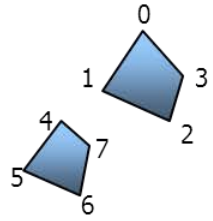
GL_LINE_STRIP



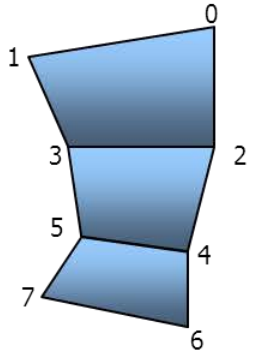
GL_LINE_LOOP



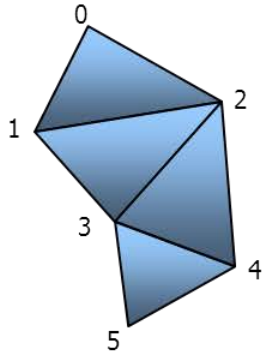
GL_POLYGON



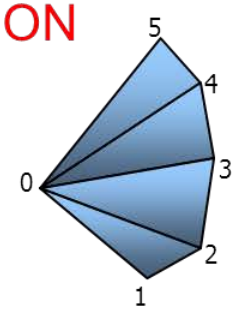
GL_QUADS



GL_QUAD_STRIP



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

*Убран в версии 3.0

Моделирование поверхностей

Результатом решения задачи *моделирования* является множество вершин, однозначно определяющих набор геометрических объектов.

Полигональные сетки (Polygonal meshes)

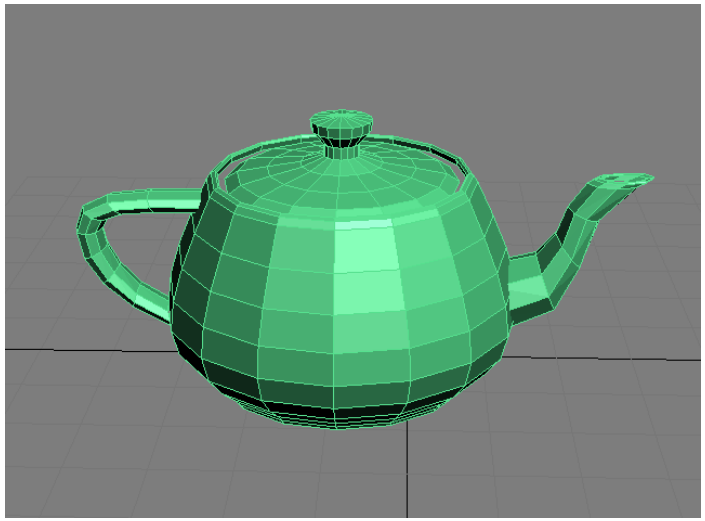
Полигональные сетки – набор полигонов (граней), которые в совокупности формируют оболочку объекта

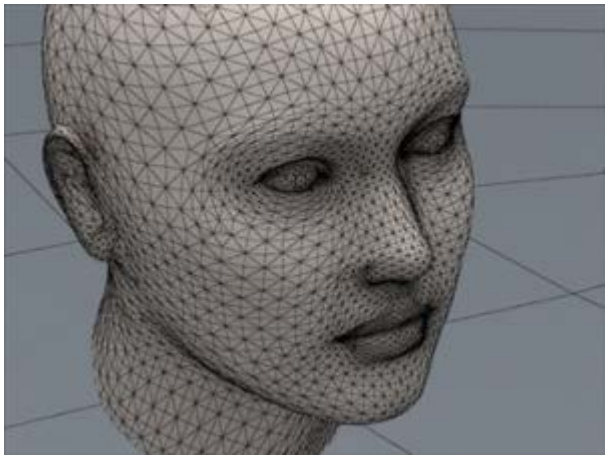
- Это стандартный способ визуального представления широкого класса объемных фигур
- Многие системы визуализации основаны на изображении объектов посредством рисования последовательности полигонов

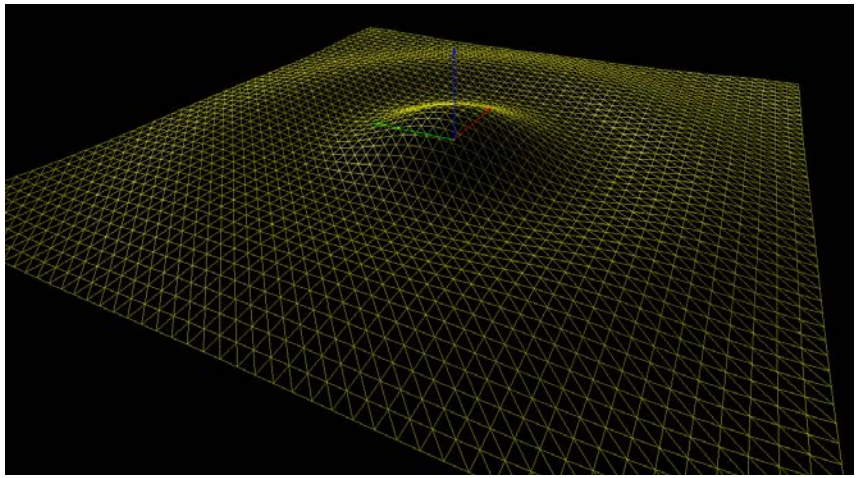
Достоинства полигональных сеток

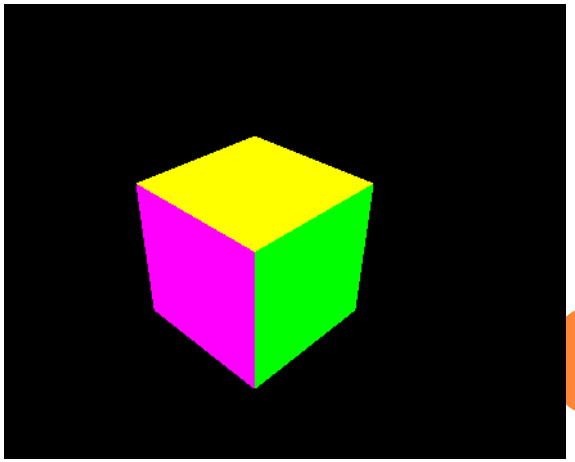
- Основаны на простоте использования полигонов:
  - Легко представлять и преобразовывать
  - Обладают простыми свойствами
    - единственный вектор нормали
    - четко определенные внутренняя и внешняя области
  - Простота рисования
    - подпрограмма закрашивания полигонов или наложения текстуры на плоскую грань
  - Полигональные сетки позволяют представлять трехмерные объекты практически любой степени сложности

Примеры









Монолитные объекты и тонкие оболочки

Полигональные сетки позволяют задавать объекты двух типов:

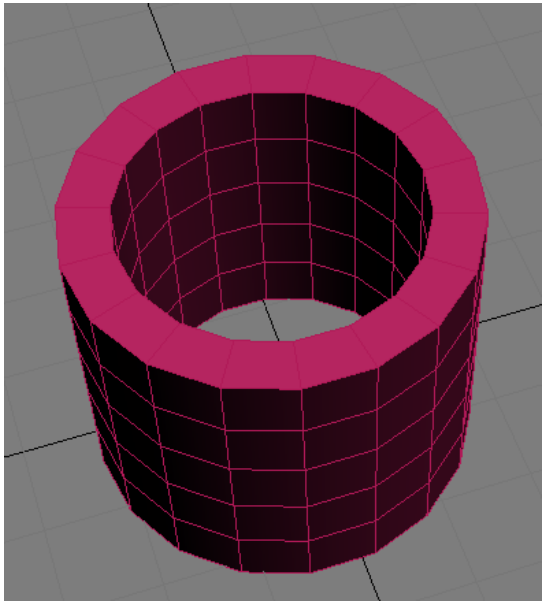
Монолитные (solid) объекты

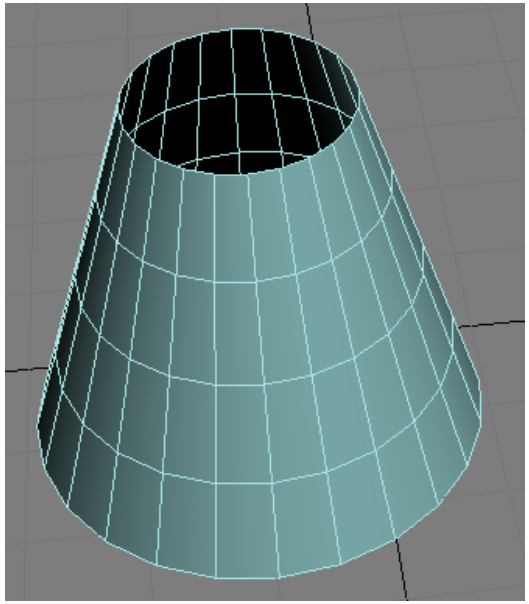
- полигональные грани плотно примыкают друг к другу и ограничивают некоторое пространство
  - Примеры: куб, сфера

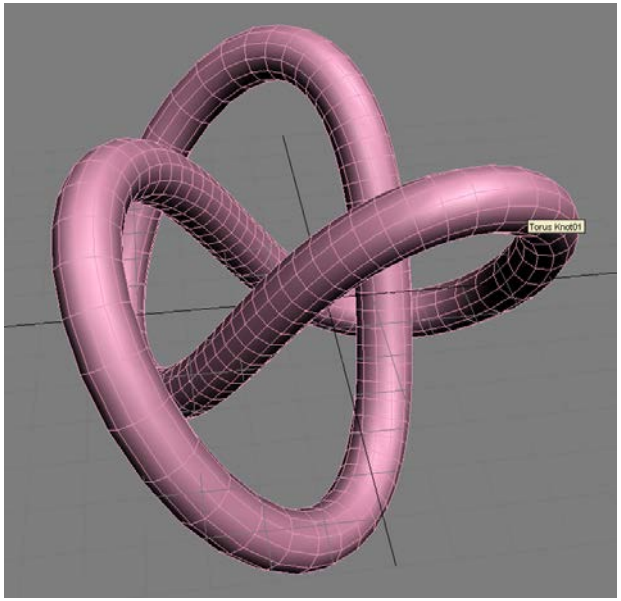
Тонкие оболочки

- Полигональные грани примыкают друг к другу без ограничения пространства, представляя собой поверхность бесконечно малой толщины
  - Пример: график функции $z = f(x,y)$

Примеры







Вершины полигона

Каждый полигон определяется путем перечисления его **вершин**

Вершина задается при помощи перечисления ее **координат** в пространстве

Пример представления вершины полигональной сетки

```
struct Vertex
{
   GLfloat  x;
   GLfloat  y;
   GLfloat  z;
};
```

Нормаль к полигону

Вектор нормали задает направление перпендикуляра грани

При рисовании объекта эта информация используется для определения того, сколько света рассеивается на данной грани

Пример представления нормали полигона

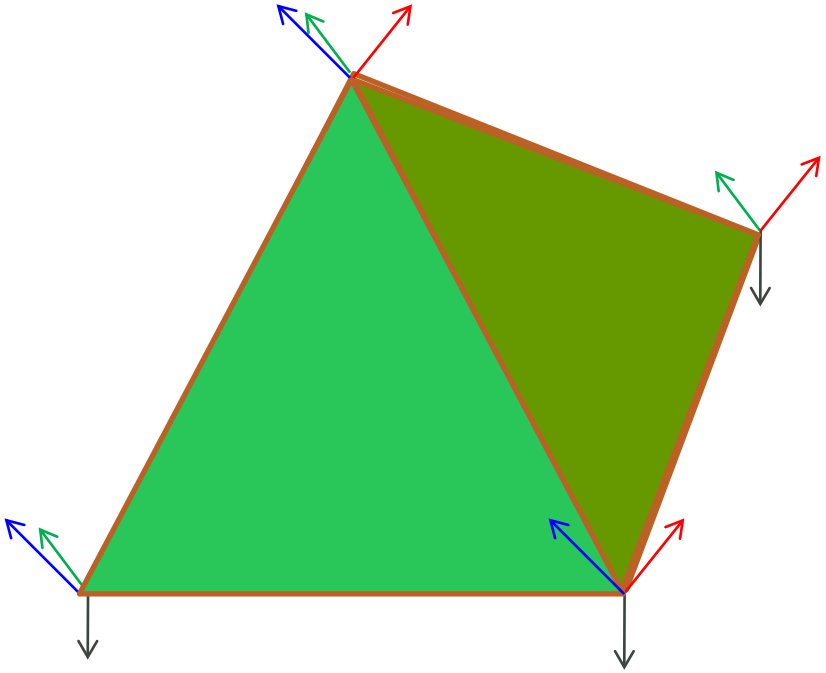
```
struct Normal
{
   GLfloat  x;
   GLfloat  y;
   GLfloat  z;
};
```

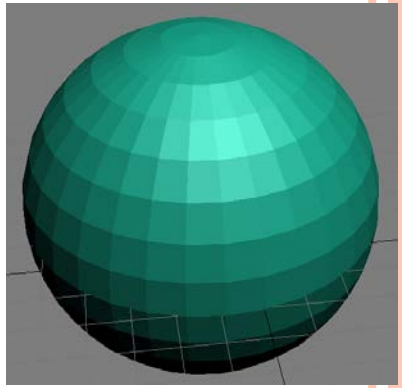
Нормаль

Одним из атрибутов вершины в OpenGL является **вектор нормали** к поверхности в точке вершины

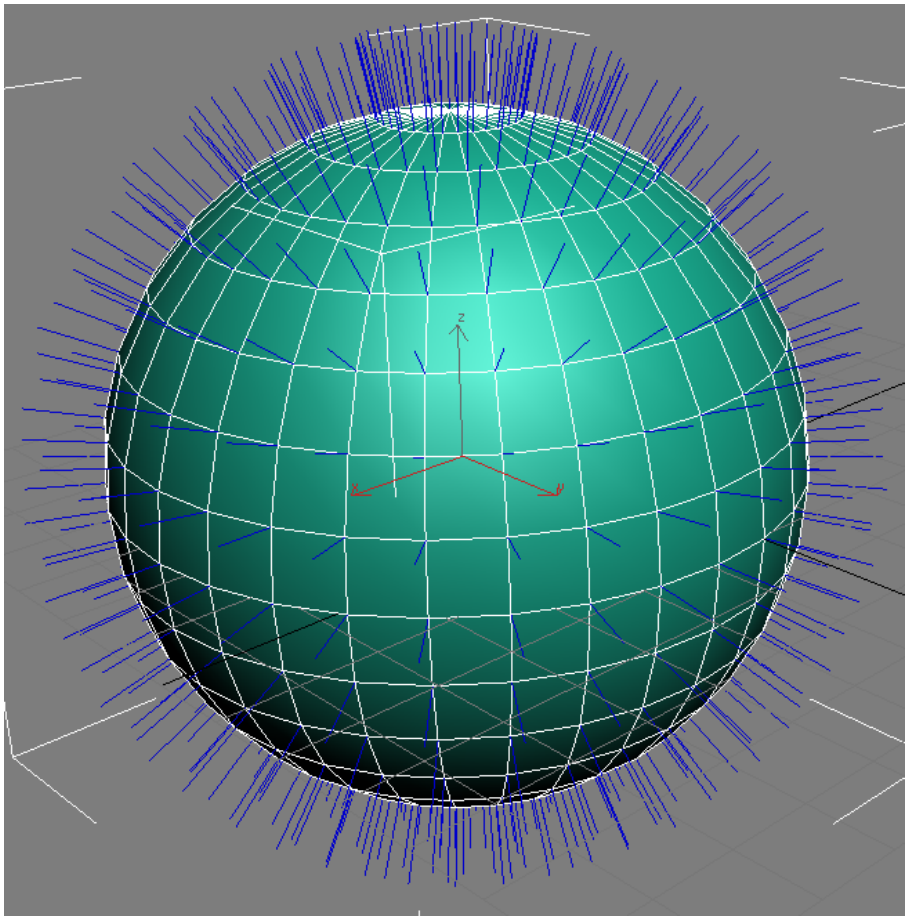
- направлен перпендикулярно поверхности, проходящей через текущую вершину
- OpenGL использует нормали для расчета освещенности граней
- текущий вектор нормали задается при помощи функций **GL.Normal**

Задание нормалей вершин для объекта с плоскими гранями





Задание нормалей вершин примитивов, аппроксимирующих криволинейную поверхность

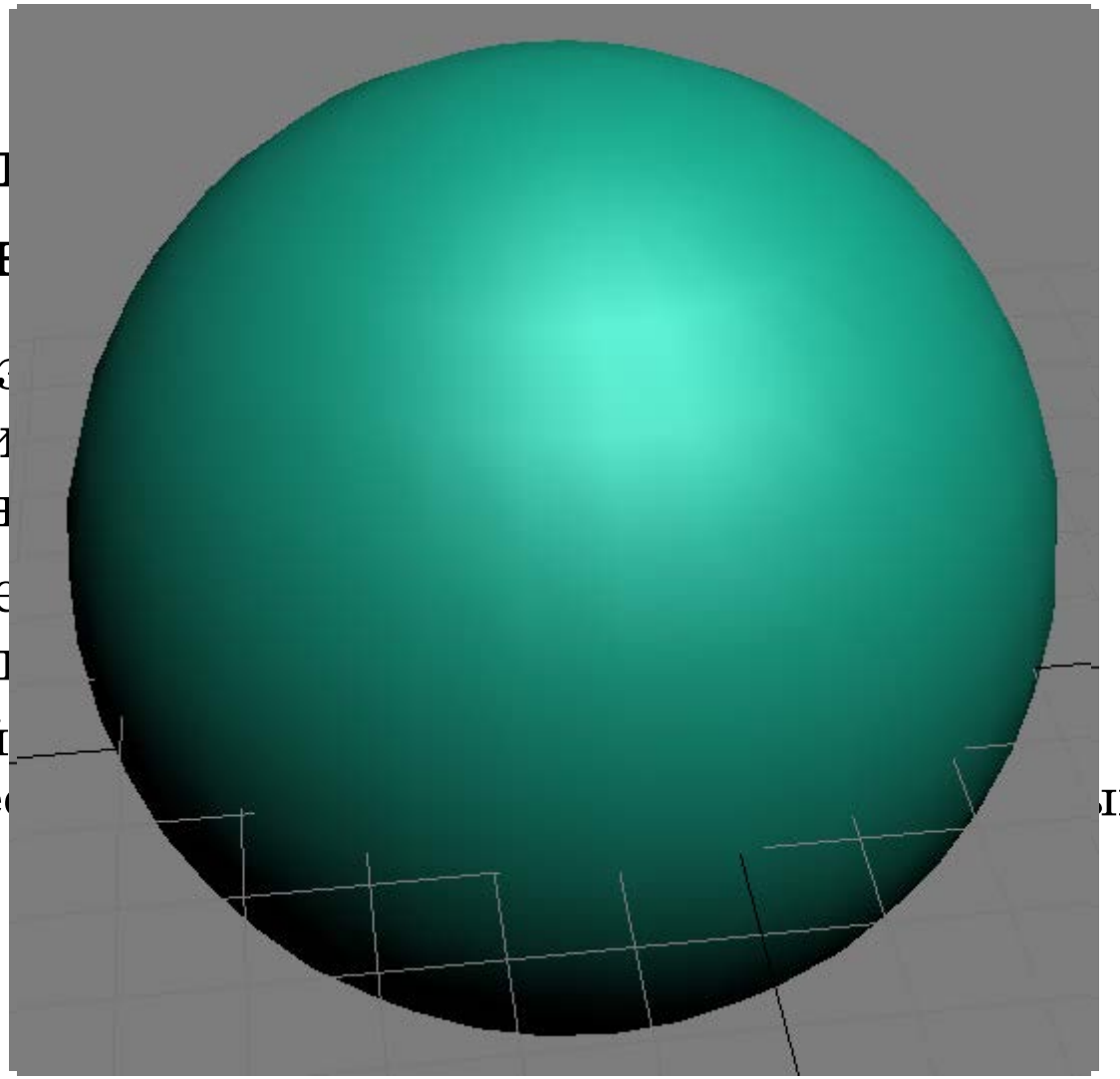


Нормали в вершинах и нормали в поверхностях

- Использование нормалей к грани плохо подходит для визуализации гладких поверхностей, например, сферы
- Удобнее оказывается связывать вектор нормали с каждой вершиной грани
  - Такой способ упрощает процесс отсечения и процесс закрашивания гладких криволинейных форм

Нормал
поверхн

- Использ
подходи
поверхн
- Удобнее
нормал
  - Такой
проце
форм



ых

Нормали в вершинах и нормали в поверхностях

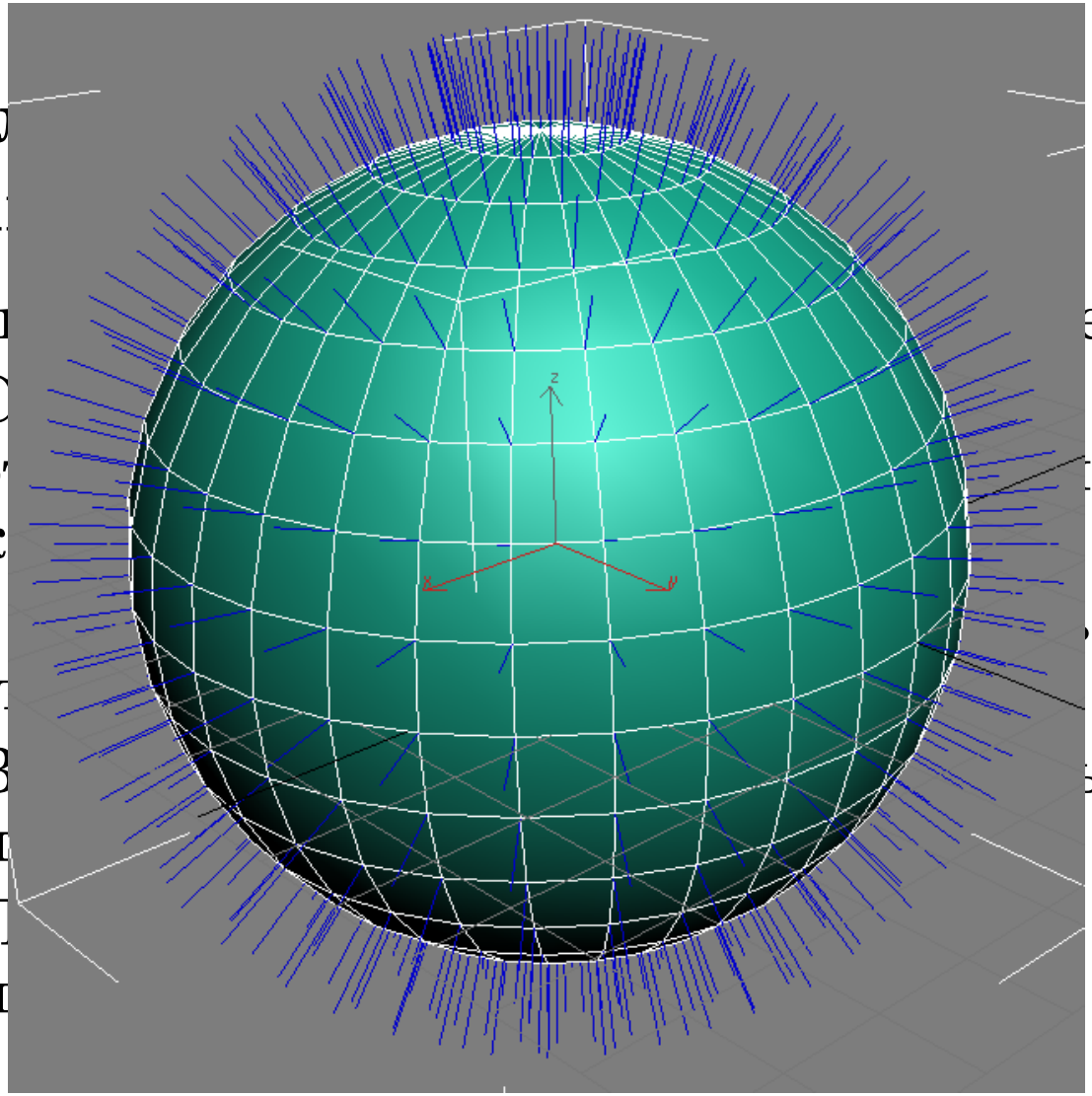
В OpenGL нормаль является атрибутом вершины

- С т.з. быстродействия выгоднее хранить отдельную копию вектора нормали для каждой вершины

Одна и та же вершина может входить в состав нескольких смежных граней

- Вывод: лучше хранить все вершины сетки (с их атрибутами) в отдельном массиве
- При задании граней указывать индексы используемых вершин

Нормал
поверх



В Оре ершины

– С хранить
о али для
к

Одна в состав
нескол

– В ы сетки (с
и

– П индексы
и

Возможные вариации

Если полигональная сетка задается при помощи однотипных примитивов, например, треугольников, то можно представить грани в виде массива индексов вершин.

Необходимо выбирать структуры данных, наиболее подходящих для решения конкретной задачи

Нахождение нормальных векторов (нормалей)

Координаты нормалей для каждой вершины можно задавать:

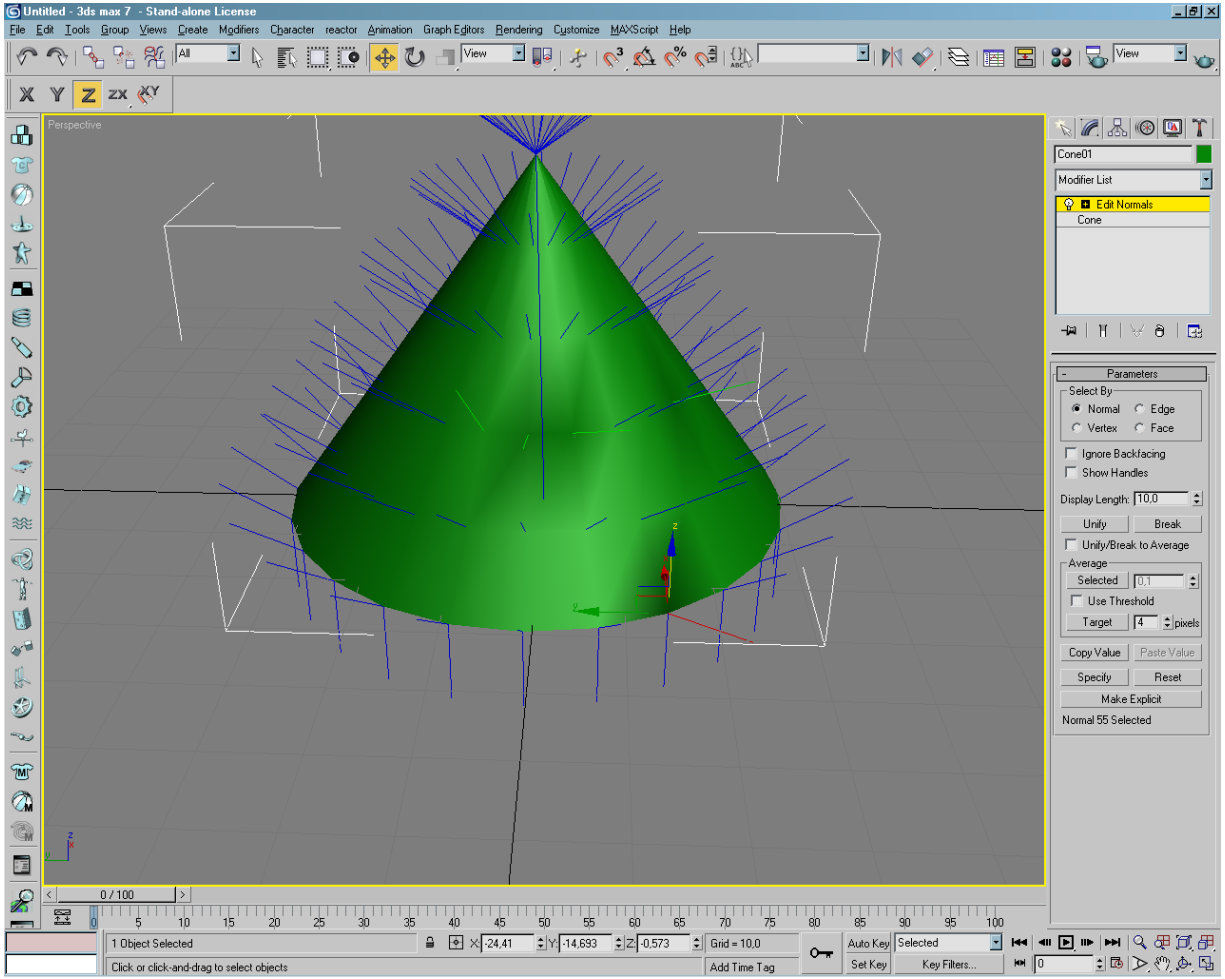
- вручную (в процессе моделирования)
- вычислять аналитически (перпендикуляр к криволинейной поверхности, описываемой функционально)
- вычислять на основе полигональной сетки

Задание нормалей вручную

Позволяет задать нормали к поверхности способом, лучшим с точки зрения дизайнера

Основной недостаток – он очень утомителен и во многих случаях может быть заменен на методы автоматического генерирования нормалей

Редактирование нормалей в программе 3D Studio Max



Аналитический метод нахождения нормалей

Для функционально заданных поверхностей вектор нормали по направлению совпадает с вектором антиградиента в точке поверхности

Нахождение градиента:

- Нахождение вектора частных производных
- Численное дифференцирование

$$F(x, y, z) = x^2 + y^2 - z$$

$$\nabla F(x, y, z) = \left(\frac{dF}{dx} \quad \frac{dF}{dy} \quad \frac{dF}{dz} \right) \approx \left(\frac{\Delta F}{\Delta x} \quad \frac{\Delta F}{\Delta y} \quad \frac{\Delta F}{\Delta z} \right)$$

Пример

Пусть необходимо найти градиент в точке (1,1) к поверхности:

– $Z = x^2 + y^2$

Решение:

$$F(x, y, z) = x^2 + y^2 - z$$

$$\nabla F(x, y, z) = (2x \quad 2y \quad -1)$$

$$\nabla F(1,1) = \nabla F(1,1,1^2 + 1^2) = \nabla F(1,1,2) = (2 \quad 2 \quad -1)$$

Для формирования нормали необходимо нормализовать данный вектор (привести его к единичной длине)

Вычисление нормалей для плоских граней полигональной сетки

Для плоских граней сетки достаточно вычислить перпендикуляр к каждой грани и связать его с каждой из вершин этой грани

- использование векторного произведения векторов, соединяющих соседние вершины граней

Проблемы:

- большие погрешности вычисления в случае выбора почти параллельных векторов
- проблемы с гранями, имеющими больше вершин

Метод Ньюэла для нахождения нормали к плоской грани

Разработан Мартином Ньюэллом, решает указанные проблемы простого способа

$$n_x = \sum_{i=0}^{N-1} (y_i - y_{next(i)})(z_i - z_{next(i)}),$$

$$n_y = \sum_{i=0}^{N-1} (z_i - z_{next(i)})(x_i - x_{next(i)}),$$

$$n_z = \sum_{i=0}^{N-1} (x_i - x_{next(i)})(y_i - y_{next(i)});$$

$$next(j) = (j+1) \bmod N$$

Нахождение нормали к вершинам сетки, описывающим криволинейную поверхность

Грани сетки, описывающей криволинейную поверхность, могут иметь общие вершины

За вектор нормали в таких вершинах можно принять среднее арифметическое нормалей прилегающих граней

Свойства сеток

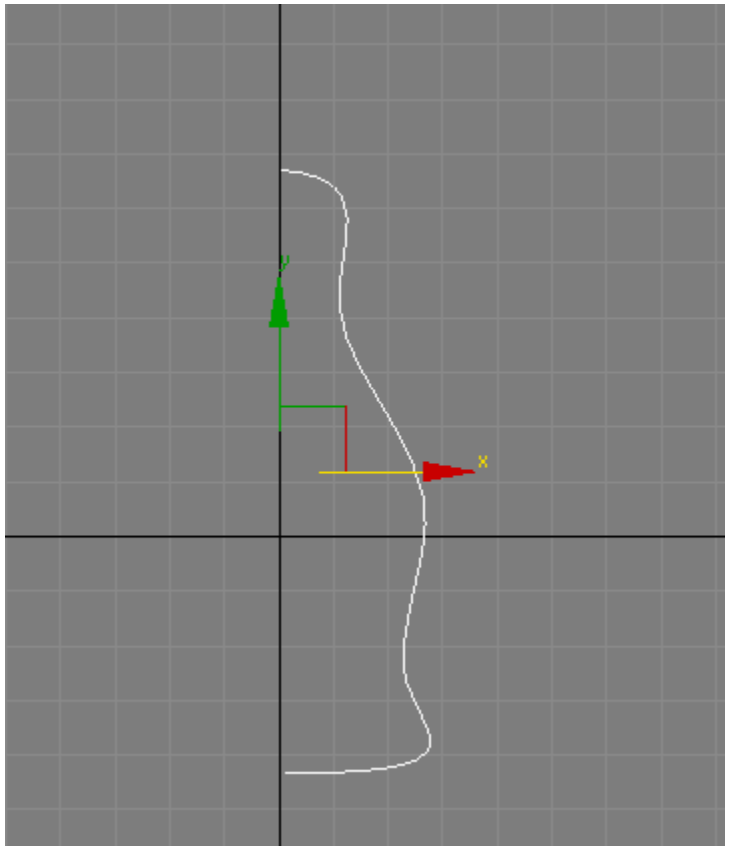
- Монолитность
  - Совокупность грани сетки заключает в себе некоторое пространство
- Связность
    - Между любыми двумя вершинами сетки существует непрерывный путь вдоль ребер полигонов
- Простота
  - Сетка является монолитной и не содержит отверстий
- Плоскостность
  - Каждая грань сетки является **плоским** полигоном
- Выпуклость
  - Отрезок прямой, соединяющий любые две внутренние точки объекта целиком лежит внутри него

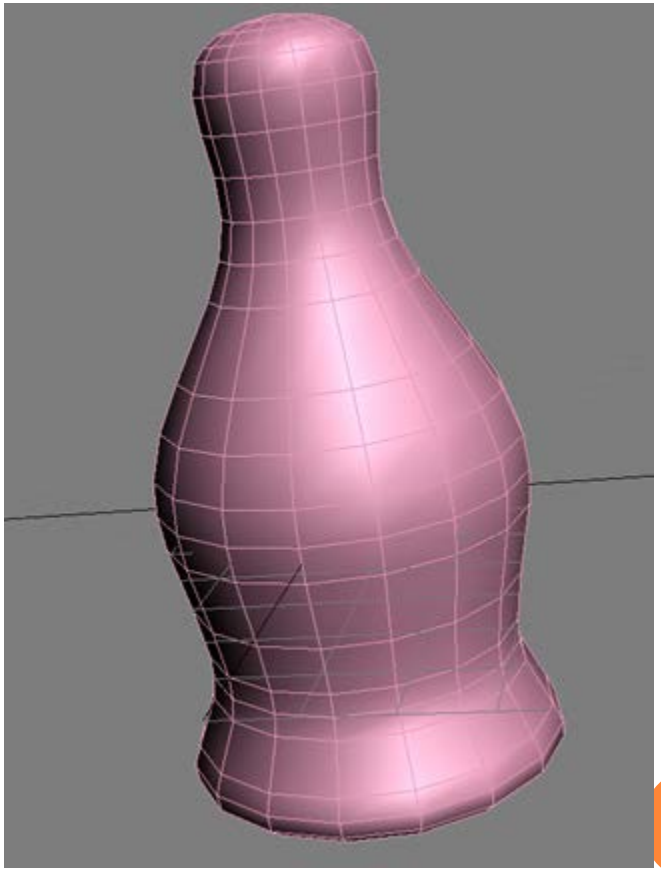
Моделирование поверхностей вращения

Поверхность вращения образуется посредством вращательной развертки с заметанием профильной кривой С вокруг некоторой оси

- Тор
- Пешка
- Сфера
- Купол церкви
- Рюмки, тарелки
- Колба лампы накаливания

Создание поверхности вращения





Поверхности на базе функций двух переменных

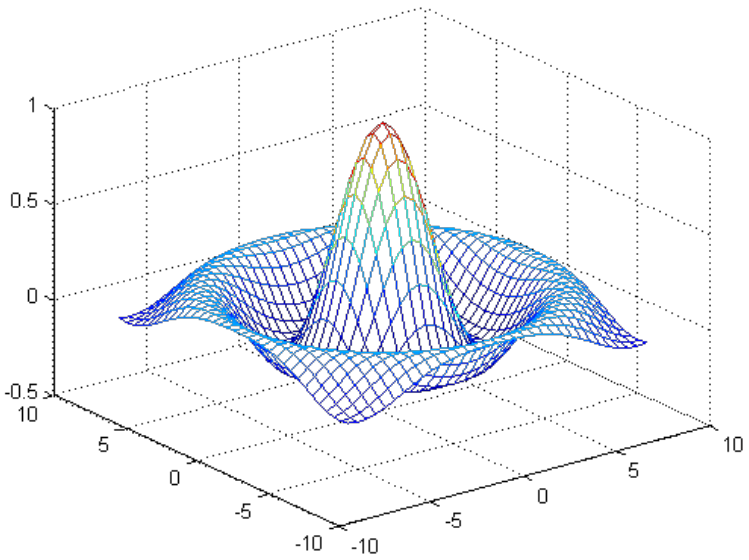
Некоторые поверхности однозначны в одном измерении, поэтому могут быть явно выражены функции двух независимых переменных

Такие функции еще называют полем высот и задают в виде формулы следующего типа:

– $y = f(x, z)$

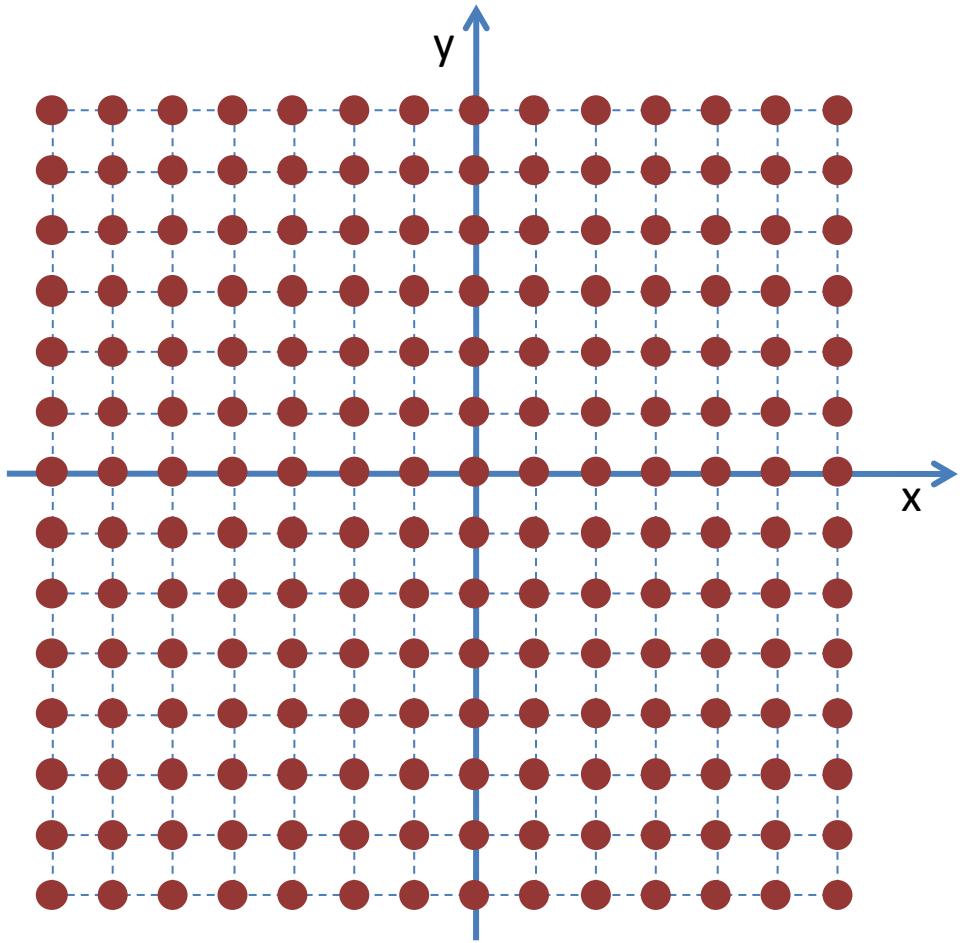
Для визуализации таких поверхностей обычно вычисляют значение y в узлах равномерной сетки вдоль осей x и z, а затем рисуют последовательность ячеек полученной сетки

Пример поверхности заданной, функцией sinc с круговой симметрией

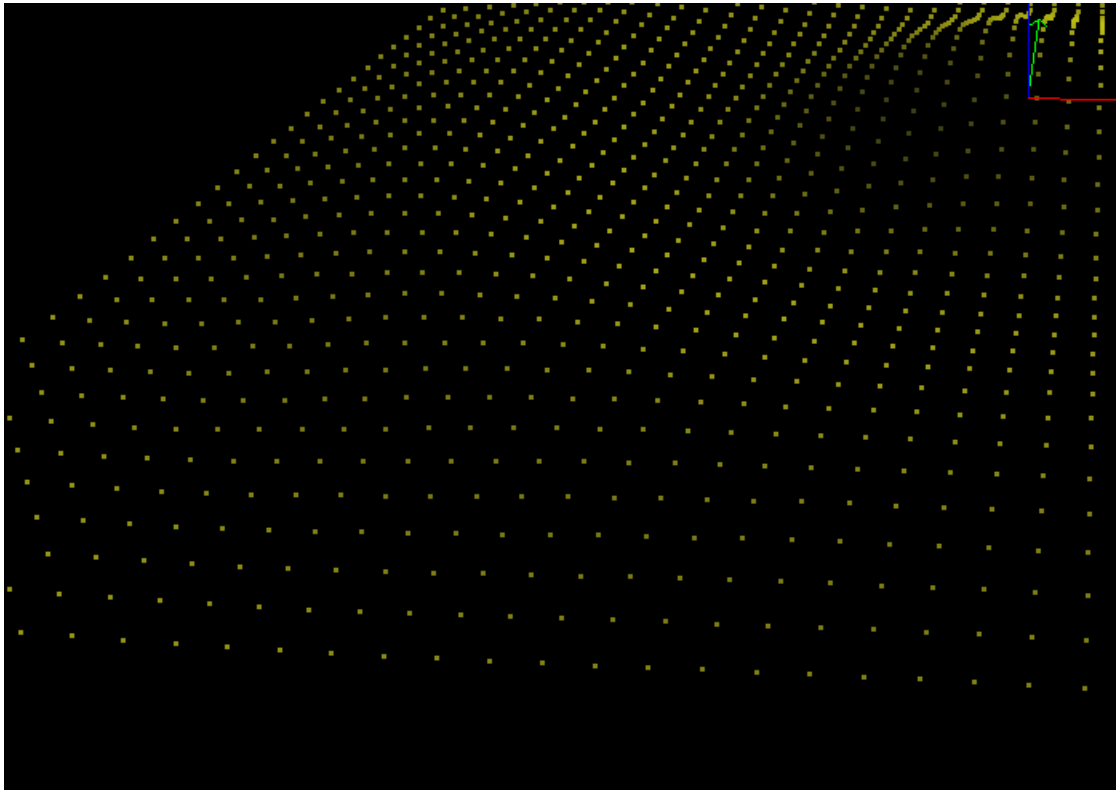


$$y = \frac{\sin(\sqrt{x^2 + z^2})}{\sqrt{x^2 + z^2}}$$

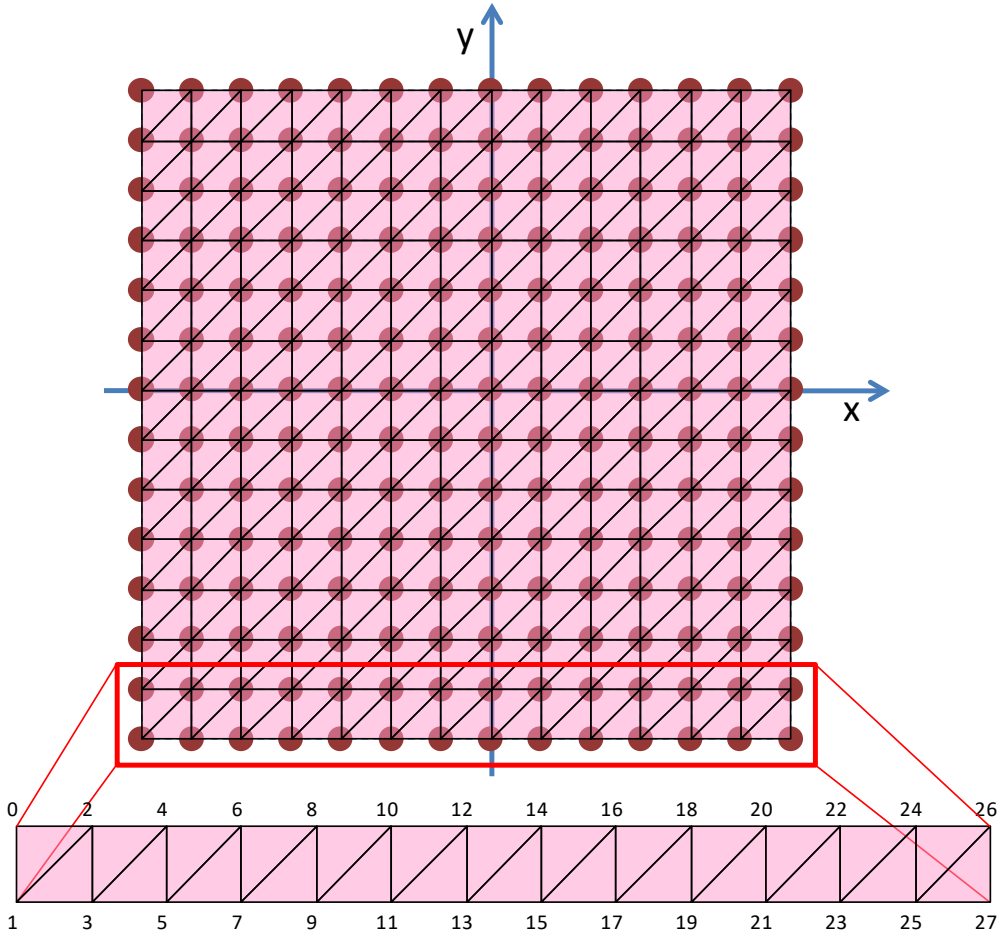
Равномерно разбиваем отображаемую область функции вдоль осей x и y



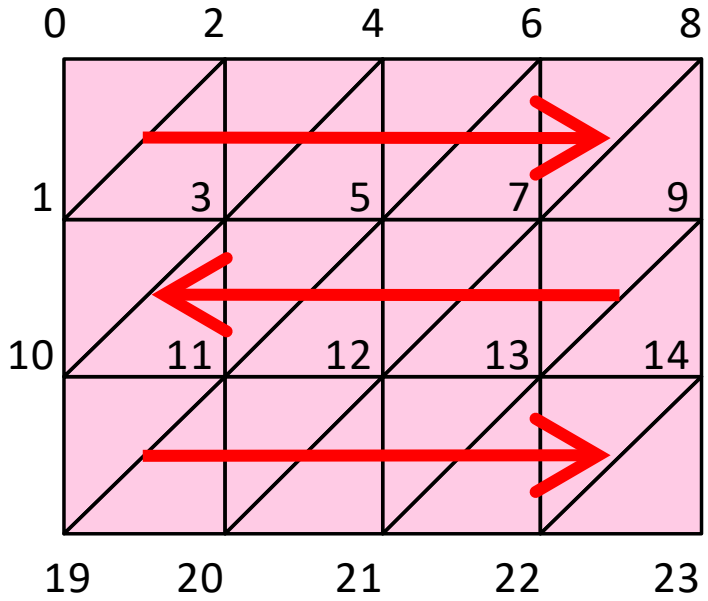
Вычисляем значение координаты z и нормалей в узлах сетки



Рисуем сетку с помощью лент из треугольников



Или даже с помощью одной ленты



0-1-2-3-4-5-6-7-8-9-(9-14)-

14-9-13-7-12-5-11-3-10-1-(10-10)-

10-19-11-20-12-21-13-22-14-23-...

Результат