Моделирование трехмерных поверхностей полигональными сетками

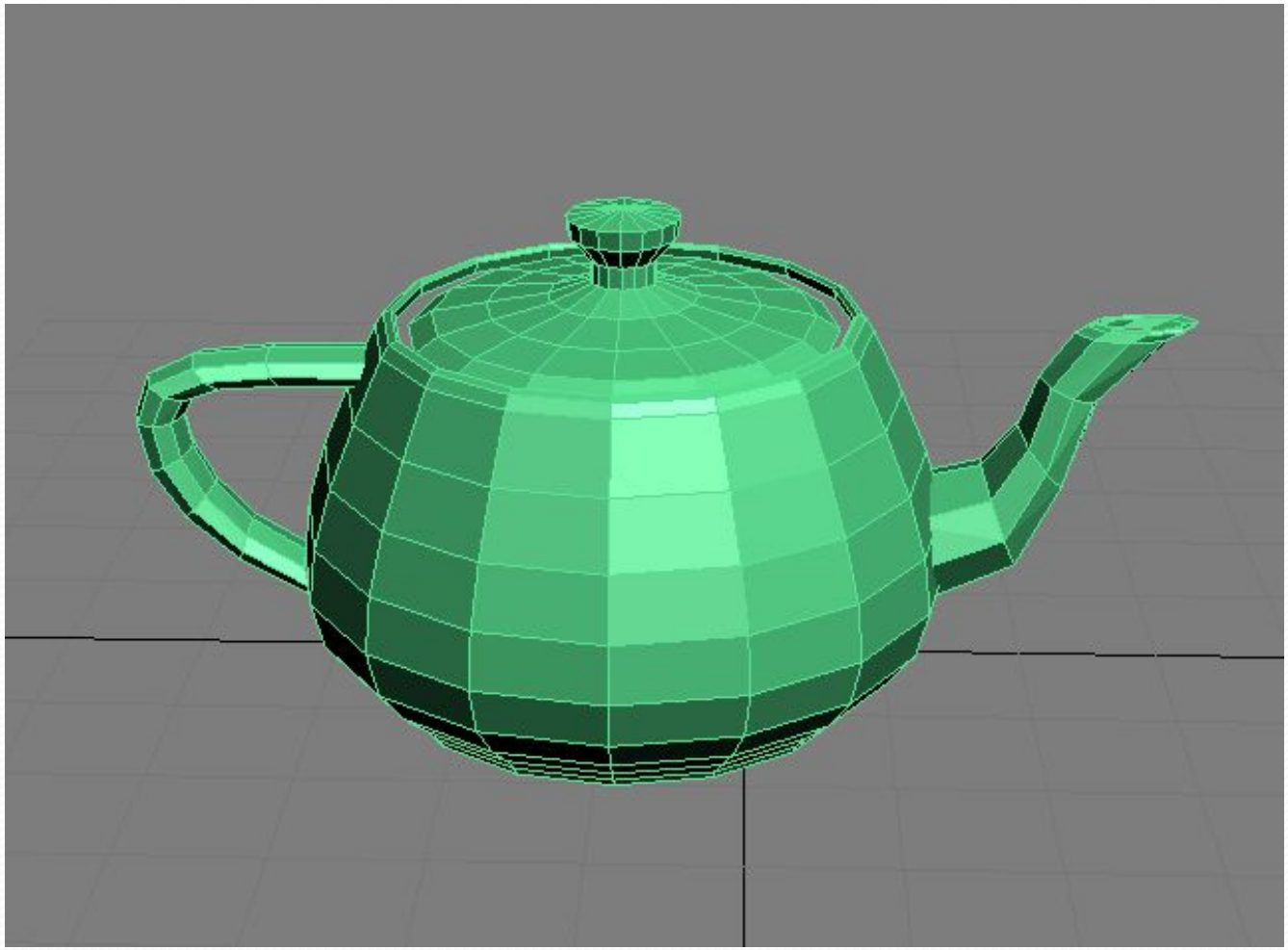
Полигональные сетки (Polygonal meshes)

- **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

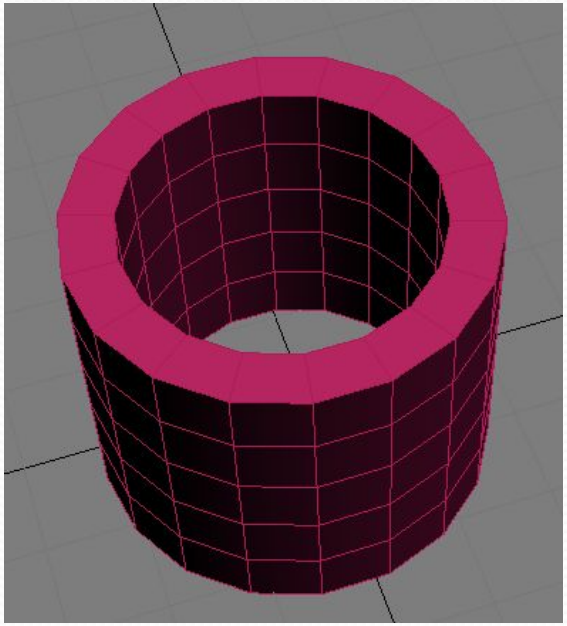
Пример:

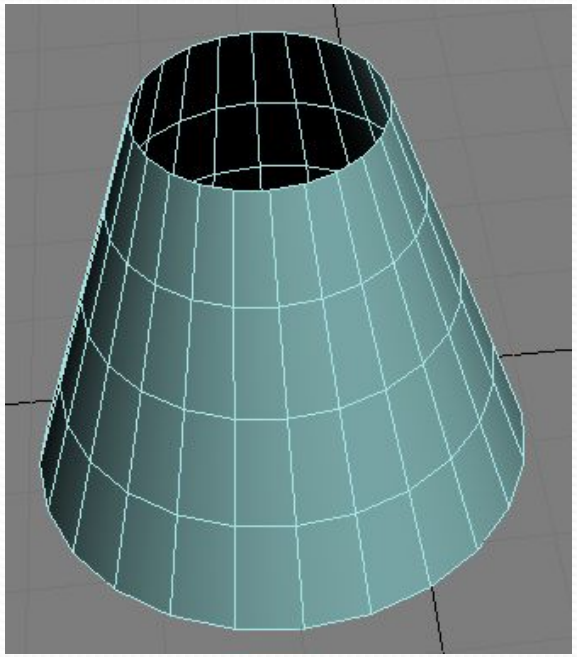


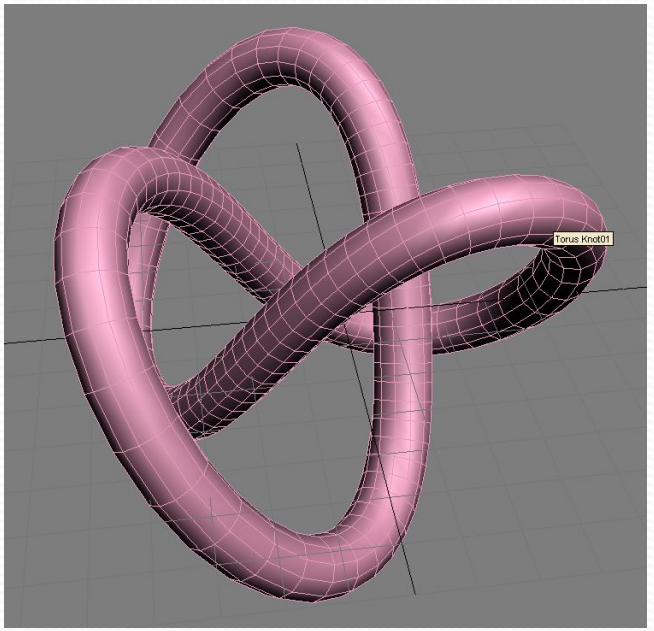
Монолитные объекты и тонкие оболочки

- 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 полигона

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

Нормаль к полигону

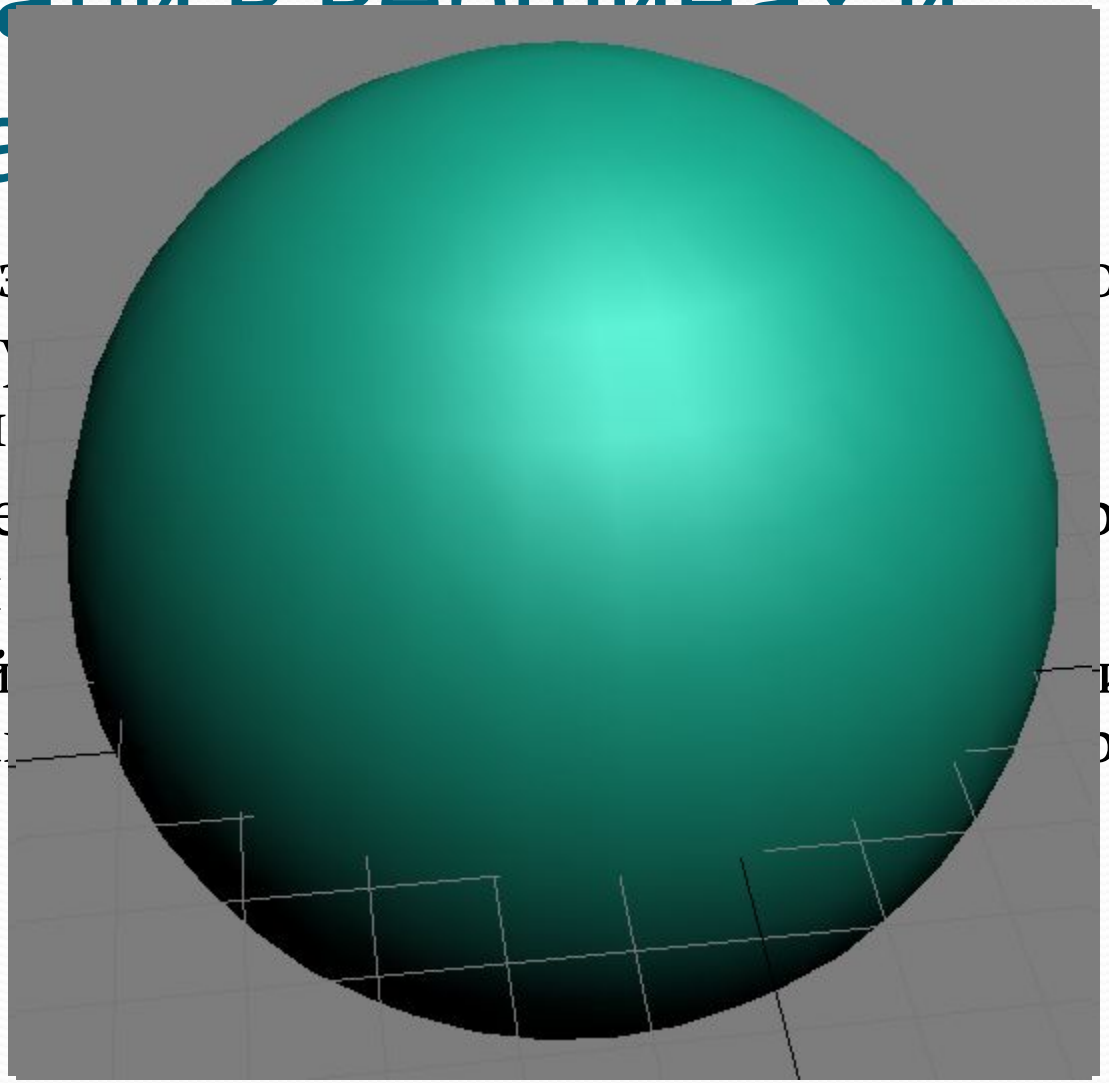
- **Вектор нормали** задает направление перпендикуляра грани
- При рисовании объекта эта информация используется для определения того, сколько света рассеивается на данной грани

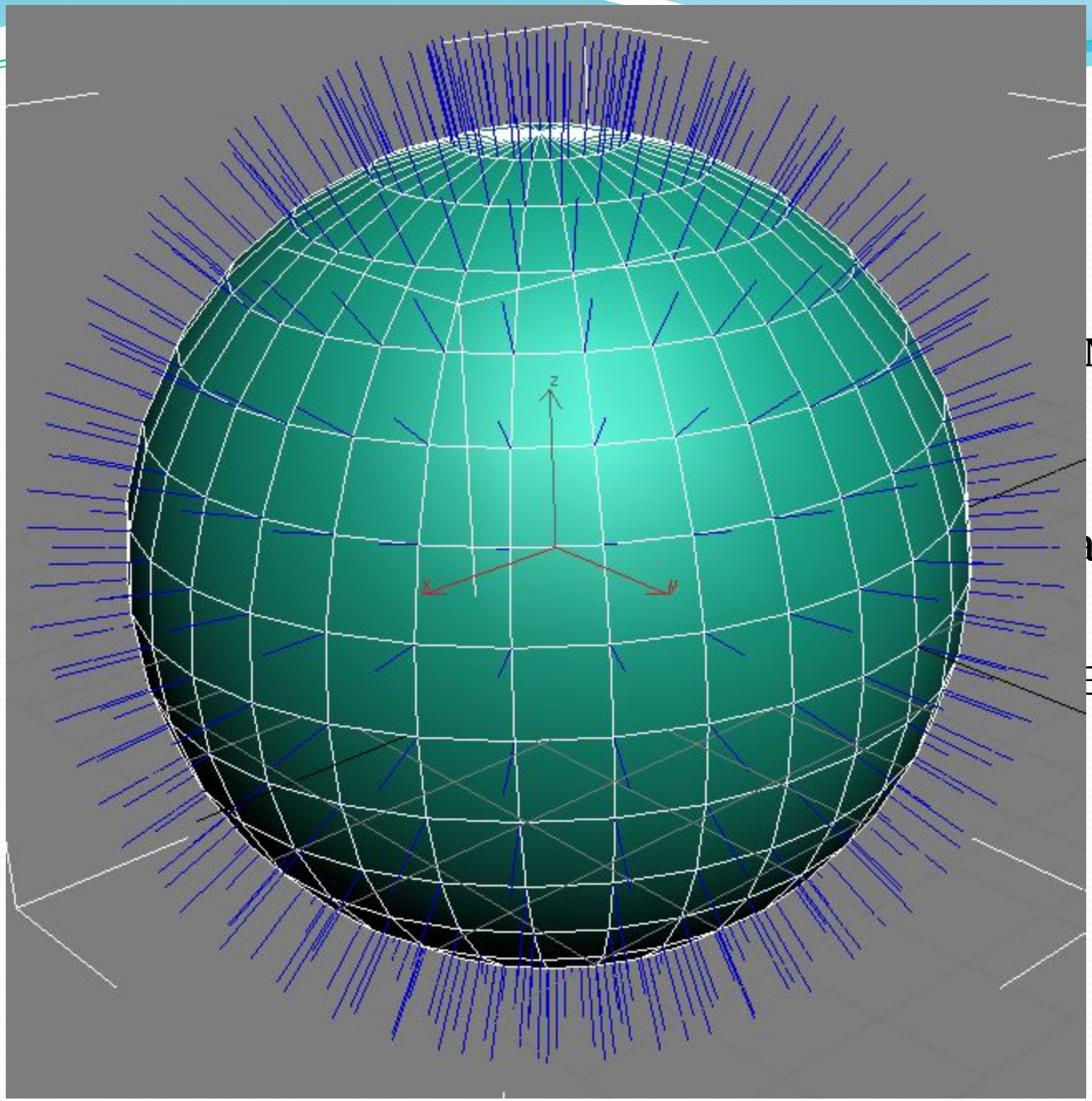
Пример представления нормали полигона

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

Норма… в вершинах и норма…

- Использ… …одходит для визу… наприм…
- Удобнее … …ормали с каждой …
  - Такой … …и процесс закра… …ом





- м
- аждой
- з
- (с их

Пример структур данных для хранения сеток

```
struct Vector3d
{
    GLfloat x, y, z;
};

struct Vertex
{
    Vector3d  position;
    Vector3d  normal;
    // …
};

struct Face
{
    GLushort v0;
    GLushort  v1;
    GLushort  v2;
};
```

```
struct Mesh
{
    GLuint numVertices;
    Vertex *pVertices;

    GLuint numFaces;
    Face   *pFaces;
};
```

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

Пример

```
struct Vector3d
{
    GLfloat x, y, z;
};

struct Vertex
{
    Vector3d position;
    Vector3d normal;
    // …
};
```

```
struct Mesh
{
    GLuint numVertices;
    Vertex *pVertices;

    GLenum primitiveType;
    GLuint numIndices;
    GLushort *pIndices;
};
```

```
void DrawMesh(Mesh *pMesh)
{
    glBegin(pMesh->primitiveType);
    for (GLuint i = 0; i < pMesh.numIndices; ++i)
    {
        GLushort v = pIndices[i];
        glNormalfv(&(pMesh->pVertices[v].normal.x));
        glVertex3fv(&(pMesh->pVertices[v].position.x));
    }
    glEnd();
}
```

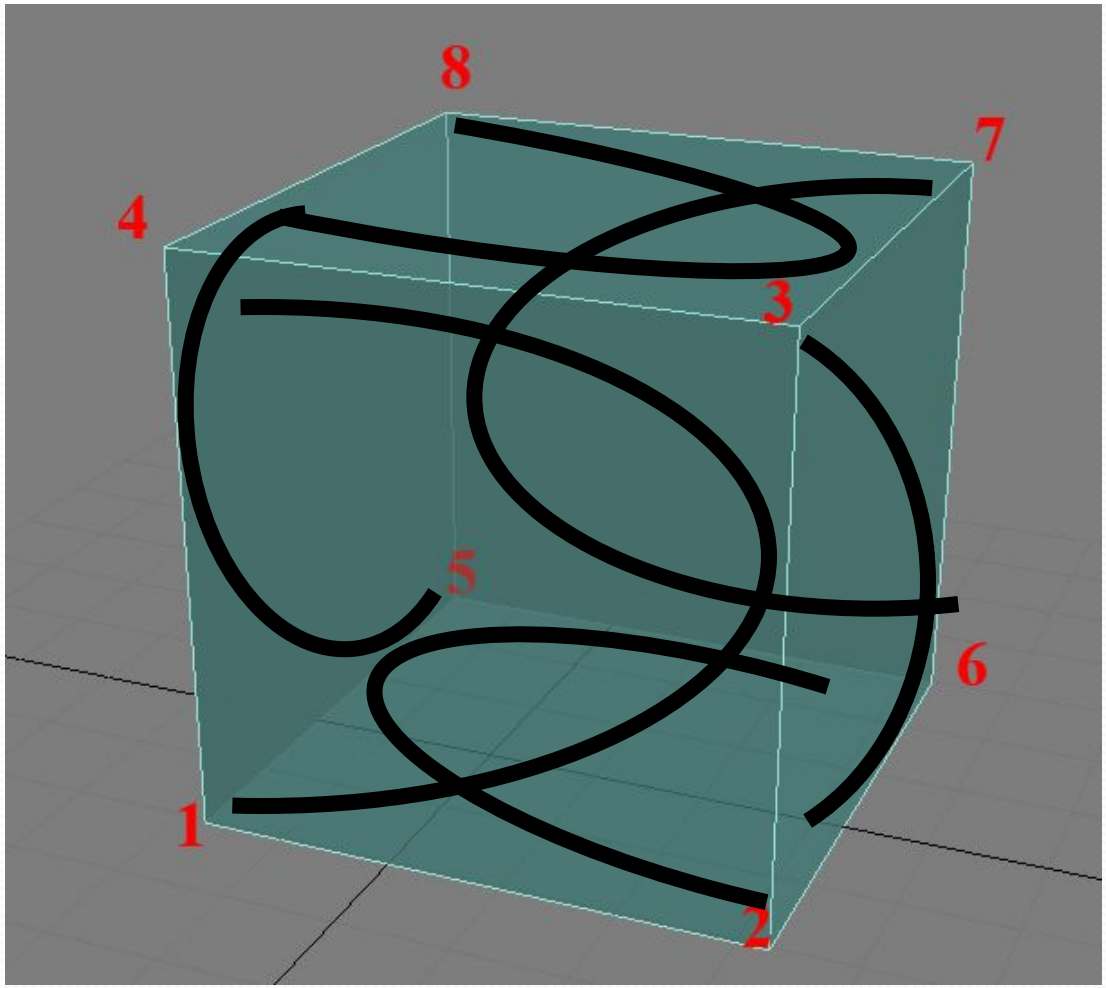
Лицевые и нелицевые стороны граней

- Каждая плоская грань (полигон) имеет две стороны:
  - **лицевую** (видна извне объекта)
  - **нелицевую** (видна изнутри объекта)
  - В один момент времени с заданной точки видна только одна сторона грани
    - Снаружи монолитного объекта видны только лицевые грани
  - OpenGL позволяет эффективно отбрасывать лицевые или нелицевые грани, что ускоряет процесс рисования

Определение видимой стороны грани

- Для определения стороны грани, повернутой к наблюдателю, OpenGL использует направление обхода вершин грани после проецирования
  - OpenGL позволят выбрать направление обхода вершин лицевых граней
    - Направление обхода нелицевых вершин будет противоположным
  - Вершины всех граней сетки необходимо перечислять в одном и том направлении обхода, если смотреть на лицевую сторону граней

Обход сторон куба против часовой стрелки



Команда glFrontFace

- Задает направление обхода вершин грани, соответствующее ее лицевой стороне (Front face):
  - void glFrontFace(GLenum mode)
    где mode:
    - GL_CW – по часовой стрелке (Clockwise)
    - GL_CCW – против часовой стрелки (Counter clockwise), это значение по умолчанию

Режим отбраковки граней (Face culling)

- После того, как направление обхода вершин грани установлено, OpenGL может произвести ее отбраковку
- Для этого необходимо включить режим отбраковки граней и указать какие из граней должны быть отбракованы

Управление режимом отбраковки граней

- glEnable(GL_CULL_FACE)
- glDisable(GL_CULL_FACE)
- void glCullFace(GLenum mode)
  где mode:
  - GL_FRONT
  - GL_BACK
  - GL_FRONT_AND_BACK

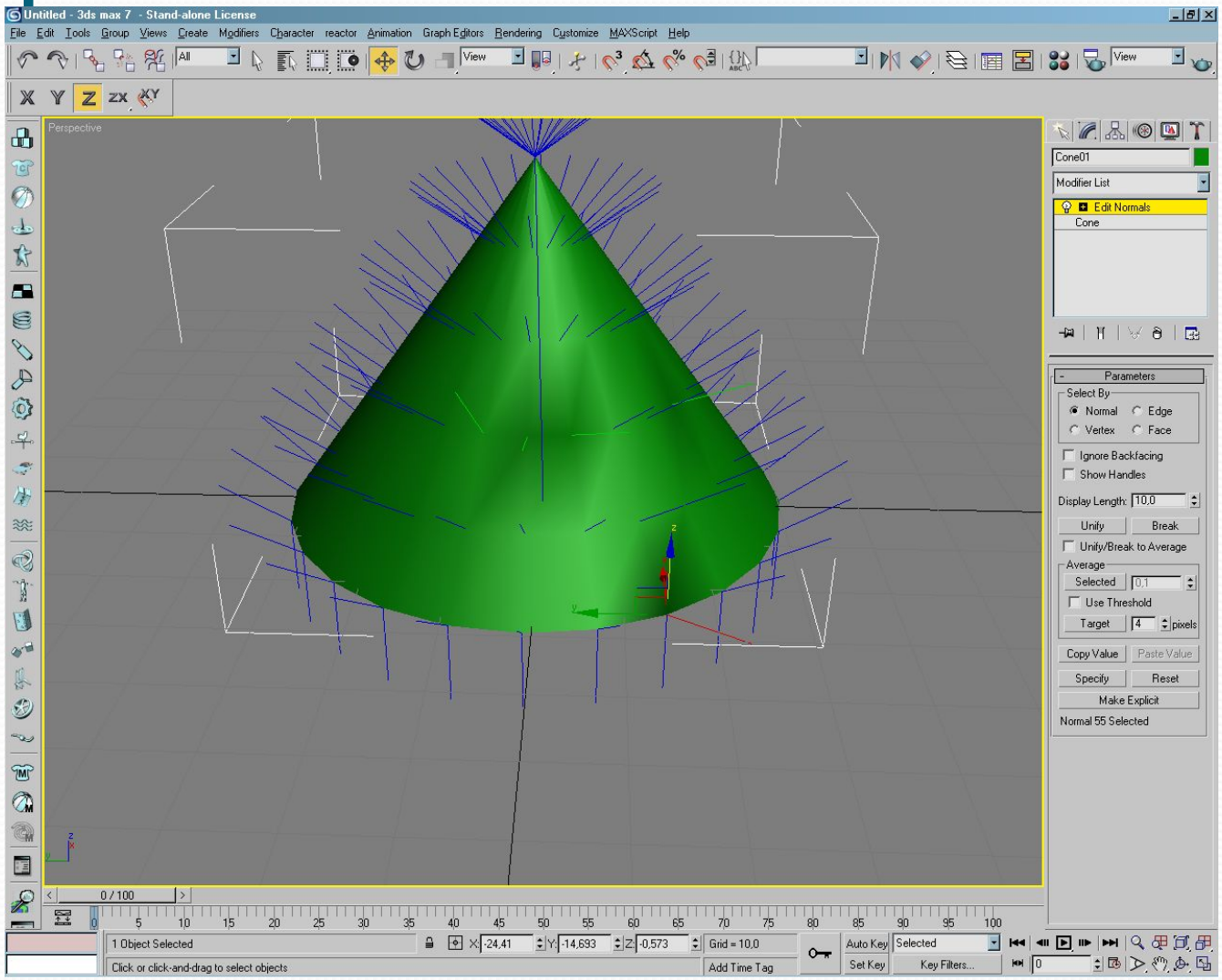
Нахождение нормальных векторов (нормалей)

- 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 градиента в точке поверхности

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 3 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

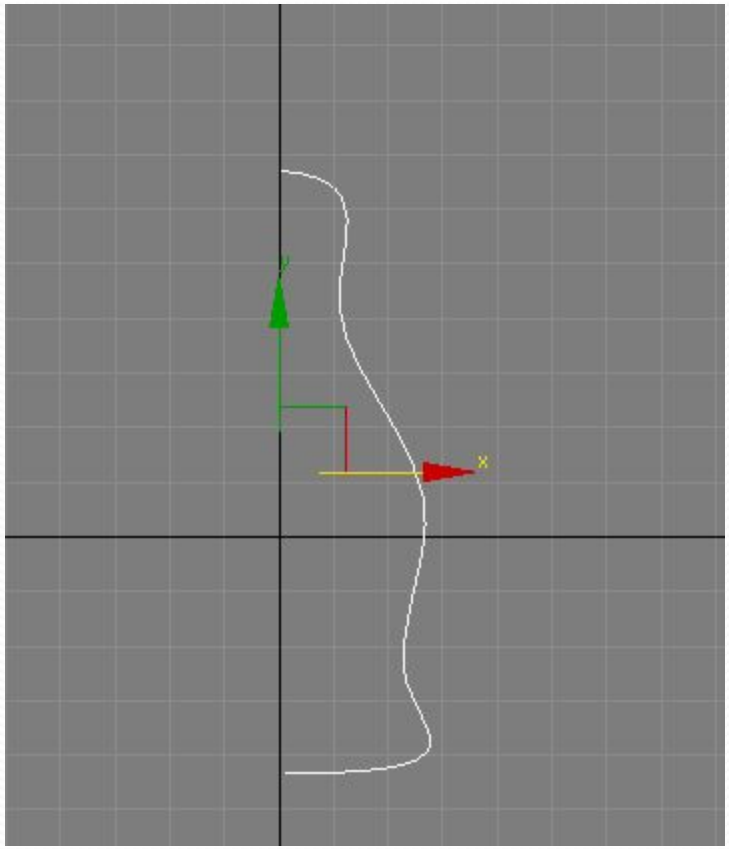
Свойства сеток

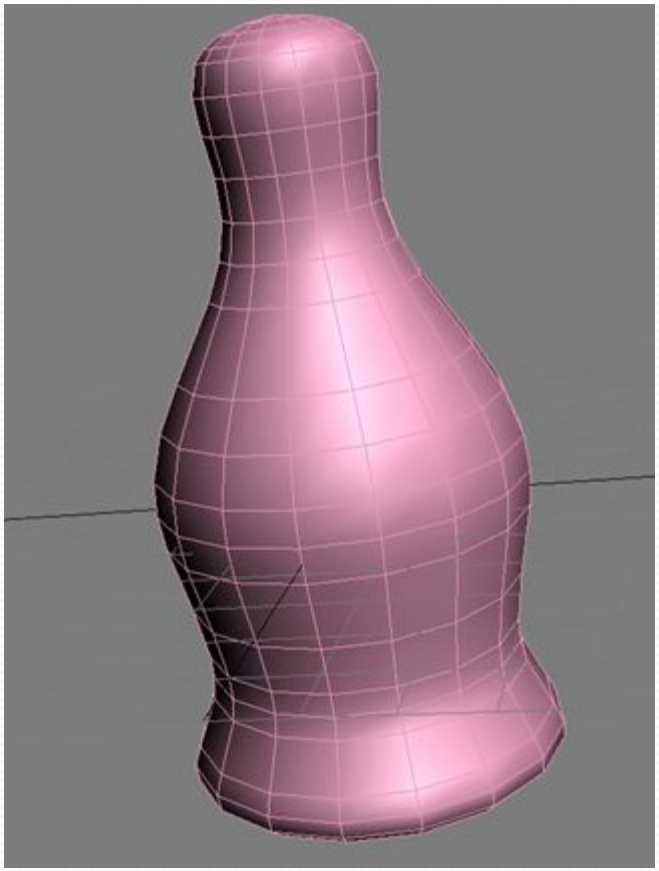
- Монолитность
  - Совокупность грани сетки заключает в себе некоторое пространство
- Связность
  - Между любыми двумя вершинами сетки существует непрерывный путь вдоль ребер полигонов
- Простота
  - Сетка является монолитной и не содержит отверстий
- Плоскостность
  - Каждая грань сетки является **плоским** 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

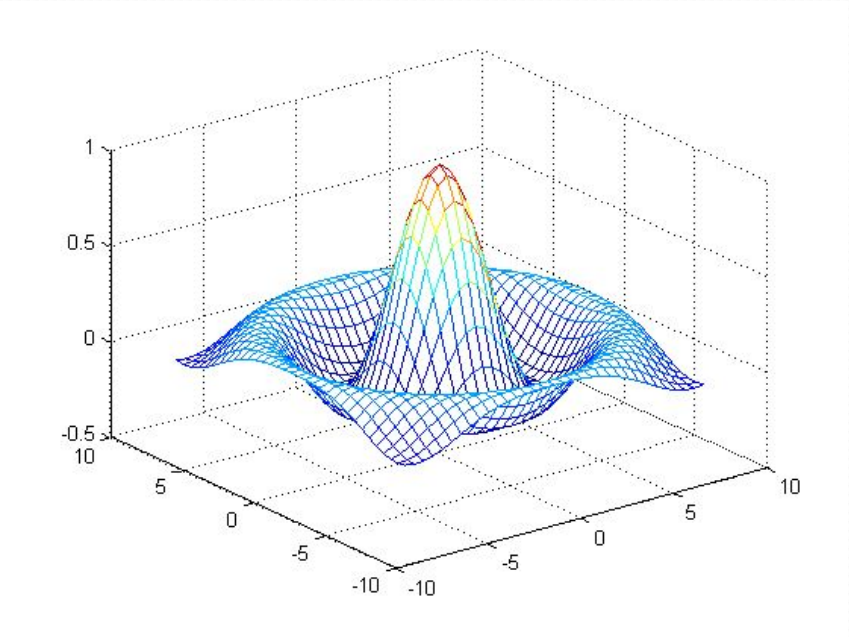




Поверхности на базе функций двух переменных

- 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



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

Визуализация трехмерных сцен

Задачи

- Для визуализации трехмерной сцены при помощи OpenGL необходимо решить ряд задач:
  - Очистка буфера кадра
  - Настройка порта просмотра и матрицы проецирования
  - Установка и ориентирование камеры
  - Размещение объектов на сцене
  - Визуализация объектов
  - Сокрытие невидимых поверхностей
- К счастью OpenGL позволяет эффективно решить все эти задачи

Очистка буфера кадра

- Очистка буфера кадра осуществляет заполнение одного или нескольких буферов, входящих в состав буфера кадра, заданными значениями
  - Буфер цвета (color buffer)
  - Буфер глубины (depth buffer)
  - Буфер трафарета (stencil buffer)
  - Буфер аккумулятора (accumulation buffer)

Команда glClear

- Выполняет очистку одного или нескольких указанных буферов:
  - void glClear(GLbitfield mask)
    где mask – комбинация одного или нескольких значений:
    - GL_COLOR_BUFFER_BIT
    - GL_DEPTH_BUFFER_BIT
    - GL_ACCUM_BUFFER_BIT
    - GL_STENCIL_BUFFER_BIT

Команда glClearColor

- Задает значение цвета, используемого при очистке буфера цветов
  - ```
    void glClearColor(
      GLclampf red,
      GLclampf green,
      GLclampf blue,
      GLclampf alpha)
    ```
- По умолчанию все значения равны о
    ```

Команда glClearDepth

- Задает значение глубины, используемое для очистки буфера глубины
  - void glClearDepth(GLclampd depth)
  - По умолчанию это значение равно 1.0

Команда glClearStencil

- Устанавливает целочисленное значение, используемое для очистки буфера трафарета
  - void glClearStencil(GLint stencil)
    - Допустимые значения – от 0 до 2^m, где m – разрядность буфера трафарета
    - Значение по умолчанию - 0

Установка порта просмотра и матрицы проецирования

- Порт просмотра задает область окна, в которую будет осуществляться вывод примитивов
- Матрица проецирования служит для осуществления перспективного или ортографического преобразования вершин примитивов

Команда glViewPort

- Устанавливает положение и размеры порта просмотра, осуществляя аффинное преобразование вершин из нормализованных координат устройства в оконные координаты
  - void glViewPort(
    GLint x,
    GLint y,
    GLint width,
    GLint height)
    - x,y – координаты левого нижнего угла порта просмотра относительно левого нижнего угла окна (0,0 по умолчанию)
    - width, height – размеры порта просмотра

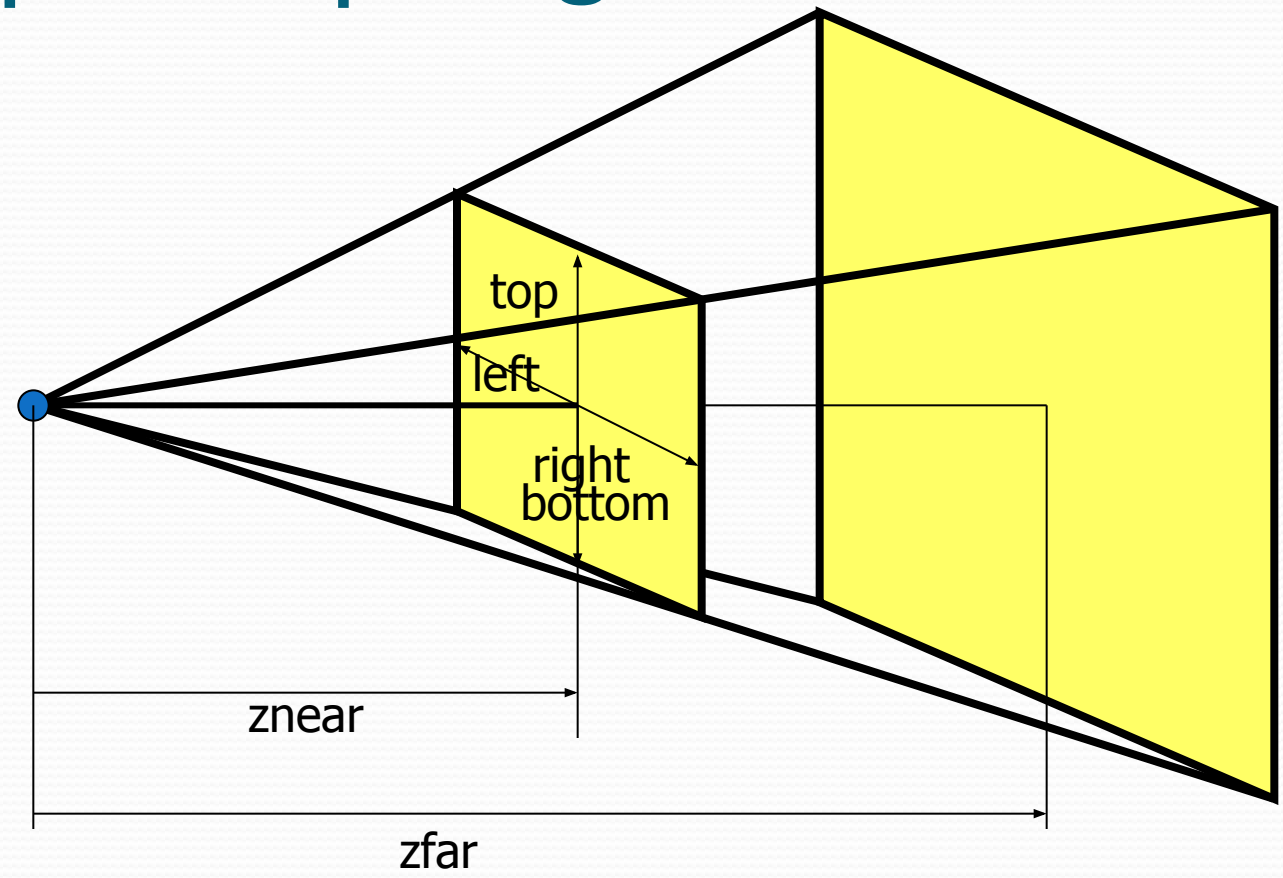
Установка матрицы перспективного преобразования

- OpenGL позволяет построить матрицу перспективного преобразования несколькими способами:
  - По координатам плоскостей, задающих усеченную пирамиду, при помощи функции glFrustum
  - По углу просмотра и пропорциям сторон отображаемого объема при помощи функции gluPerspective

Команда glFrustum

- Задает перспективное преобразование отображаемого объема по заданным координатам ограничивающих этот объем плоскостей
  - void glFrustum(
    GLdouble left,
    GLdouble right,
    GLdouble bottom,
    GLdouble top,
    GLdouble znear,
    GLdouble zfar
)

Геометрический смысл параметров glFrustum



Точность хранения значений в буфере глубины

- Точность хранения значений в буфере глубины определяется не только разрядностью буфера, но и значениями ближней и дальней плоскостей отсечения
- Чем меньше отношение
  r = zfar / znear
  тем выше точность
  - грубо говоря, $\log_2 r$ бит разрядности буфера глубины теряется
  - zfar и znear должны быть положительными
  - zfar > znear

Команда gluPerspective

- Задает матрицу перспективного проецирования по заданному углу обзора вдоль оси Y, соотношению ширины и высоты отображаемого объема и расстояниям до плоскостей отсечения
  - void gluPerspective(
    GLdouble fovy,
    GLdouble aspect,
    GLdouble zNear,
    GLdouble zFar)

Установка и ориентирование камеры

- Библиотека утилит OpenGL (GLU) позволяет задать положение наблюдателя, зная координаты его глаза, точки просмотра и вектора «вверх»
  - ```
    void gluLookAt(
    GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ,
    GLdouble lookX, GLdouble lookY, GLdouble lookZ,
    GLdouble upX, GLdouble upY, GLdouble upZ)
    ```
  - Матрица, задающая положение камеры умножается на текущую матрицу
    ```

Пример установки камеры

```
// текущая матрица – матрица моделирования-вида
glMatrixMode(GL_MODELVEIW);

// сбрасываем ранее заданные преобразования
glLoadIdentity();

// устанавливаем положение и ориентацию камеры
gluLookAt(0,0,0, 1,1,-10, 0,1,0);

// задаем объекты сцены...
```

Размещение объектов на сцене

- Ориентацию и положение объектов на сцене можно задать при помощи аффинных преобразований и функций OpenGL для работы с такими преобразованиями:
  - glTranslate
  - glRotate
  - glScale

Пример

```
// устанавливаем матрицу камеры
// ..

// перенос объекта
glTranslated(3, 3, 2);

// вращение на 30 градусов вокруг оси
x
glRotated(30, 1, 0, 0);

// вращение на 90 градусов вокруг оси
y
glRotated(90, 0, 1, 0);

// Рисование объекта..
```

Обратите внимание на тот факт, что для преобразования объекта команды преобразований применяются в обратном порядке.

Комбинация матричных преобразований

- Каждая вершина примитива умножается на некоторую матрицу T равную:
  $T = P \times V \times M$
  - P – матрица проецирования
    - $P = P_1 [x P_2 [x P_3 ...]]$
  - V x M – матрица моделирования-вида
    - V – матрица камеры
      - $V = V_1 [x V_2 [x V_3 ...]]$
    - M – матрица преобразований объектов
      - $M = M_1 [x M_2 [x M_3 ...]]$

Визуализация объектов

- Визуализация объектов заключатся в рисовании примитивов, составляющих этот объект
  - Выполнение серий командных скобок glBegin()/glEnd()
  - Разработка функций визуализирующих полигональные сетки

Пример:

```
void DrawSomeObject()
{
    glBegin(GL_TRIANGLES);
        glNormald(1, 0, 0);
        glColor3f(0.1f, 1, 1);
        glVertex3f(3, 2, 3);
        //...
    glEnd();
    // ...
}
```

Сокрытие невидимых линий и поверхностей

- Объекты, расположенные ближе к наблюдателю, могут полностью или частично перекрывать объекты, расположенные дальше
- Самый простой способ решения данной задачи – включить тест глубины командой
  - glEnable(GL_DEPTH_TEST)

Вопросы?