Модуль 2 «Математические методы, модели и алгоритмы компьютерной геометрии»
Лекция 10 «Методы изображения поверхностей»

к.ф.-м.н., доц. каф. ФН-11, Захаров Андрей Алексеевич,
ауд.:930а(УЛК)
моб.: 8-910-461-70-04,
email: azaharov@bmstu.ru

МГТУ им. Н.Э. Баумана

7 ноября 2023 г.

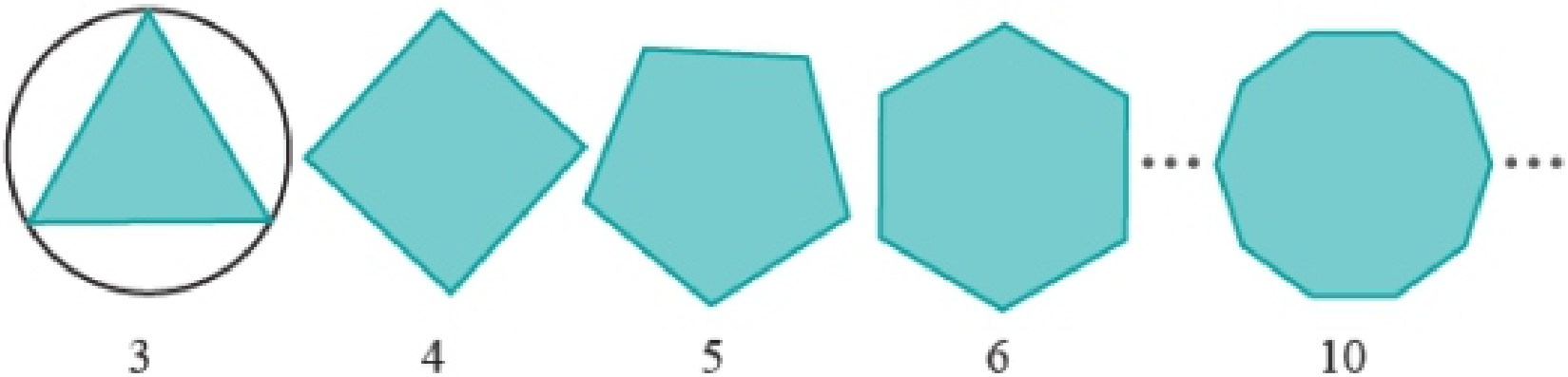
Трёхмерные объекты

Математические описания трёхмерных объектов могут быть достаточно сложными, поэтому для их эффективной визуализации на графическом оборудовании используются следующие приближения:

1. Объекты описываются их поверхностями и могут считаться полыми.
2. Объекты могут быть заданы с помощью набора вершин.
3. Поверхности объектов либо состоят, либо могут быть аппроксимированы плоскими выпуклыми многоугольниками. Это упрощает и ускоряет визуализацию, поскольку все поверхности описываются линейными уравнениями.

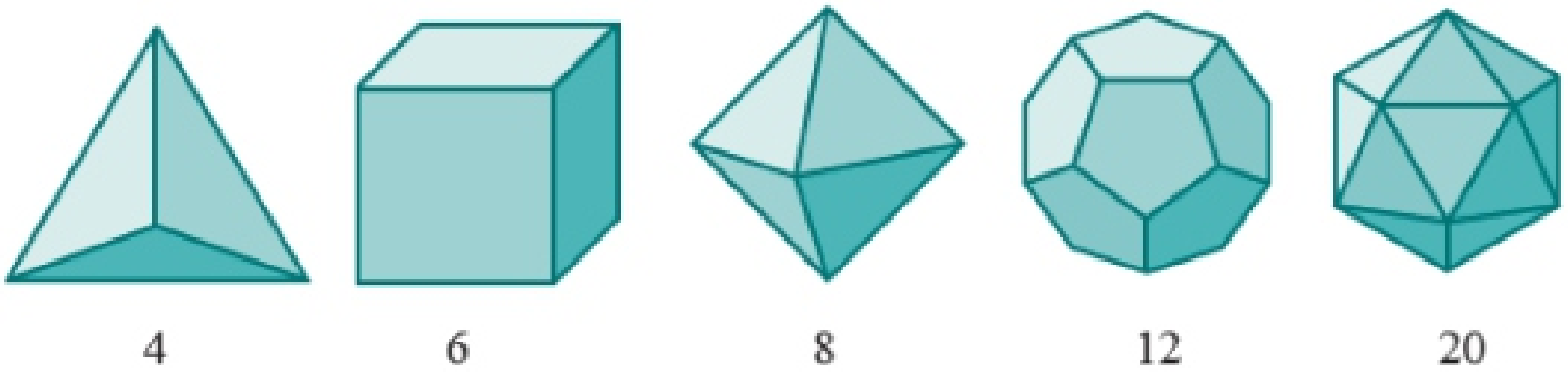
Для каждого многоугольника координаты вершин записываются в таблицы многоугольников вместе с информацией о векторе нормали к поверхности. Графические библиотеки предлагают функции, позволяющие визуализировать сетки из многоугольных поверхностей — наборов треугольников с помощью одной команды. Кроме того, некоторые библиотеки также предлагают функции вывода на экран таких распространённых форм, как куб, сфера или цилиндр, представленных многоугольными поверхностями. В профессиональных графических системах используются быстрые аппаратные реализации схем визуализации с помощью многоугольников, что позволяет отображать 100 миллионов плоских закрашенных многоугольников (обычно треугольников) за секунду, включая наложение текстуры на поверхность и применение специальных эффектов освещения.

Правильные многоугольники



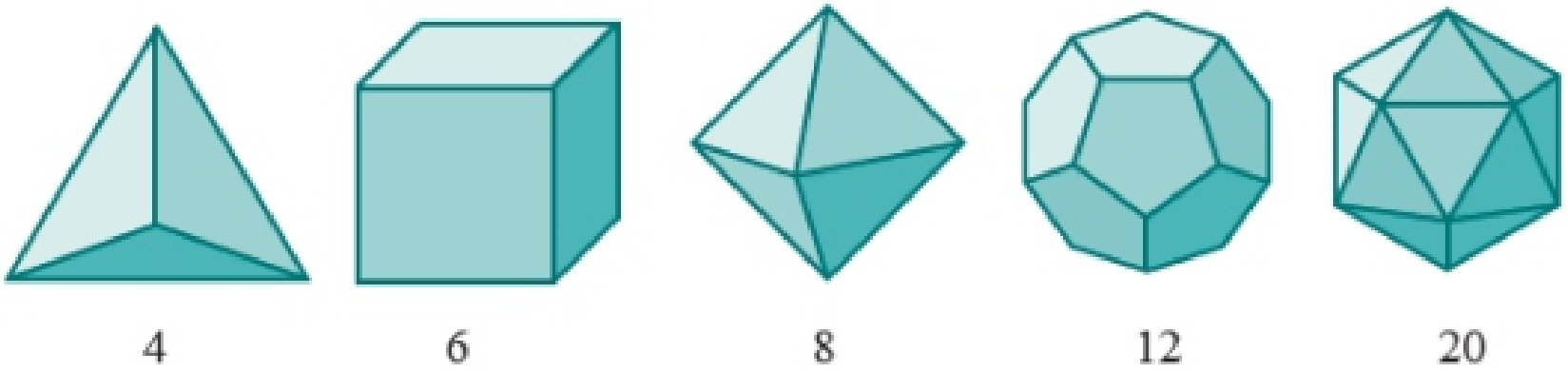
Определение. Правильный многоугольник представляет собой выпуклый многоугольник, стороны которого имеют равную длину и имеют равные внутренние углы в своих вершинах.
Правильный многоугольник с n сторонами выпуклый и его вершины расположены равномерно с угловым шагом $2\pi/n$ вдоль его описанной окружности. Чем больше значение n, тем ближе многоугольник приближает свою описанную окружность.

Правильные многогранники



Правильные многогранники (платоновы тела) являются обобщением правильных многоугольников на трёхмерное пространство.
Определение. Многогранник является твёрдым телом, границей которого является полигональная сетка (т.е. грани являются полигонами).
Определение. Правильный многогранник — многогранник, все грани которого являются однотипными правильными многоугольниками.
Из-за ограничений симметрии граней существует только пять различных типов правильных многогранников. В порядке возрастания числа граней — это тетраэдр, гексаэдр (куб), октаэдр, додекаэдр и икосаэдр.

Правильные многогранники



	Число граней	Число ребер	Число вершин	Число ребер на грань	Число вершин на грань
Тетраэдр	4	6	4	3	3
Гексаэдр (куб)	6	12	8	4	3
Октаэдр	8	12	6	3	4
Додекаэдр	12	30	20	5	3
Икосаэдр	20	30	12	3	5

Гексаэдр ограничен квадратами, додекаэдр — правильными пятиугольниками, а оставшиеся три фигуры — равносторонними треугольниками.

Поверхности

В общем случае уравнение поверхности может задаваться либо в неявной форме:

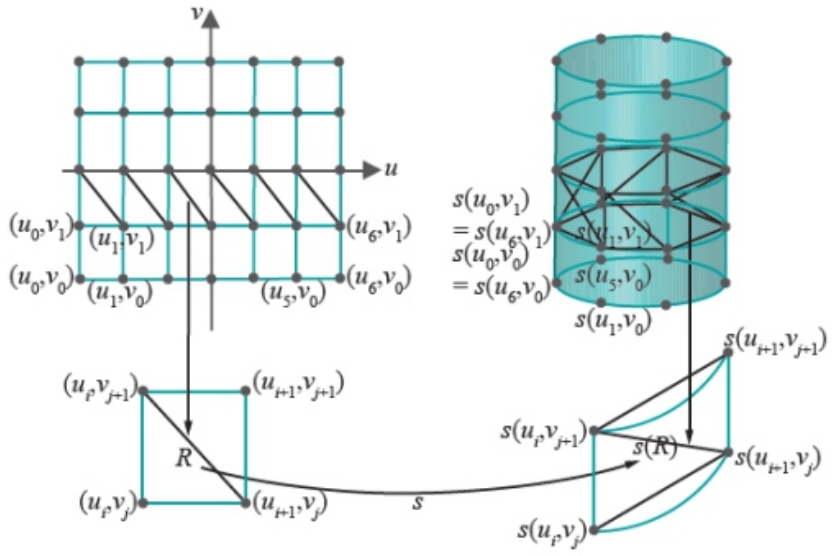
$$F(x, y, z) = 0,$$

либо в параметрической форме, которая состоит из трёх уравнений

$$x = f(u, v); \quad y = g(u, v); \quad z = h(u, v); \qquad (u, v) \in W.$$

Пространство параметров W является подмножеством плоскости $\mathbb{R}^2$.
Будем предполагать, что пространство параметров W является плоским прямоугольником $[a, b] \times [c, d]$.
Точку $(f(u, v), g(u, v), h(u, v))$ на поверхности s обозначают $s(u, v)$.

Визуализация поверхностей



Образуем сетку из $(p+1)(q+1)$ точек в прямоугольной области W:

$$(u_i, v_j), \qquad 0 \leqslant i \leqslant p, \quad 0 \leqslant j \leqslant q,$$

где $a = u_0 < u_1 < \ldots < u_p = b$, $c = v_0 < v_1 < \ldots < v_q = d$.
Найдем образы этих узлов: $s(u_i, v_j), \qquad 0 \leqslant i \leqslant p, \quad 0 \leqslant j \leqslant q$, которые будем использовать в качестве вершин треугольной сетки, аппроксимирующей поверхность s. Эта сетка состоит из следующих $2pq$ треугольных ячеек:

$$\triangle s(u_i, v_j)s(u_{i+1}, v_j)s(u_i, v_{j+1}) \text{ и } \triangle s(u_i, v_{j+1})s(u_{i+1}, v_j)s(u_{i+1}, v_{j+1}),$$

где $0 \leqslant i \leqslant p-1$, $0 \leqslant j \leqslant q-1$.

Визуализация поверхностей

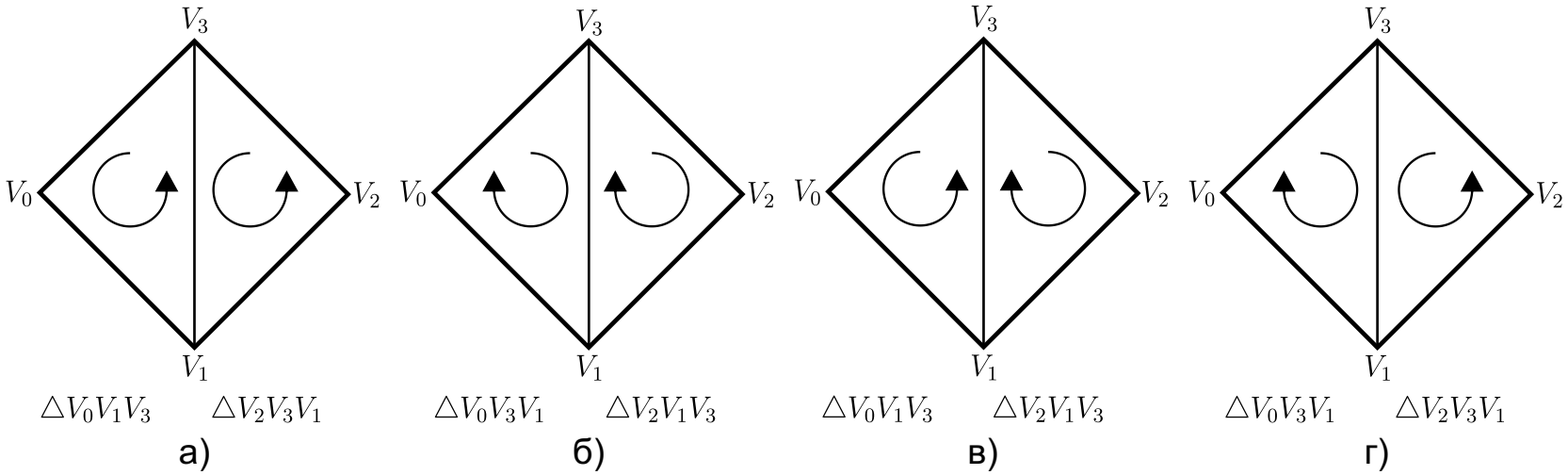


Рис.: Четыре возможных конфигурации задания ориентации для двух смежных треугольников: а) и б) — согласованные, в) и г) — несогласованные

Для ячеек сетки можно определить *ориентацию* путём задания порядка перечисления вершин у граней. Практически во всех графических приложениях ячейки сетки должны иметь согласованные ориентации. Порядок перечисления можно задать двумя способами, и каждое упорядочение определяет свою ориентацию нормалей граничных поверхностей, отличающихся знаком. Поэтому требуется выбрать какая из двух возможных ориентаций будет использоваться в программе.

Визуализация поверхностей

Обычно выбор основывается на видимости ячеек сетки из точки, где расположена камера. Порядок выбирается таким образом, чтобы у видимых ячеек сетки векторы нормали $\mathbf{n}$ были направлены к точке, где находится камера $\mathbf{P}_0$. То есть, если ячейка сетки является плоскостью, то она видима, если $\mathbf{n} \cdot (\mathbf{P}_0 - \mathbf{V}) > 0$, где $\mathbf{V}$ — любая точка, принадлежащая ячейке, например, одна из её вершин. Говорят, что такие ячейки являются передними (*front-facing*). Её вершины, если смотреть с позиции камеры, упорядочены против часовой стрелки. Ячейки, для которых $\mathbf{n} \cdot (\mathbf{P}_0 - \mathbf{V}) \leqslant 0$ называются задними (*back-facing*). В стандартной системе визуализации задние поверхности могут сразу быть отброшены и не участвовать в расчёте геометрических преобразований и освещения, тем самым экономя много времени на рендеринг. Следовательно для сцены с большим количеством замкнутых фигур примерно половина ячеек окажутся задними. Их пропуск приведет к значительному повышению производительности.

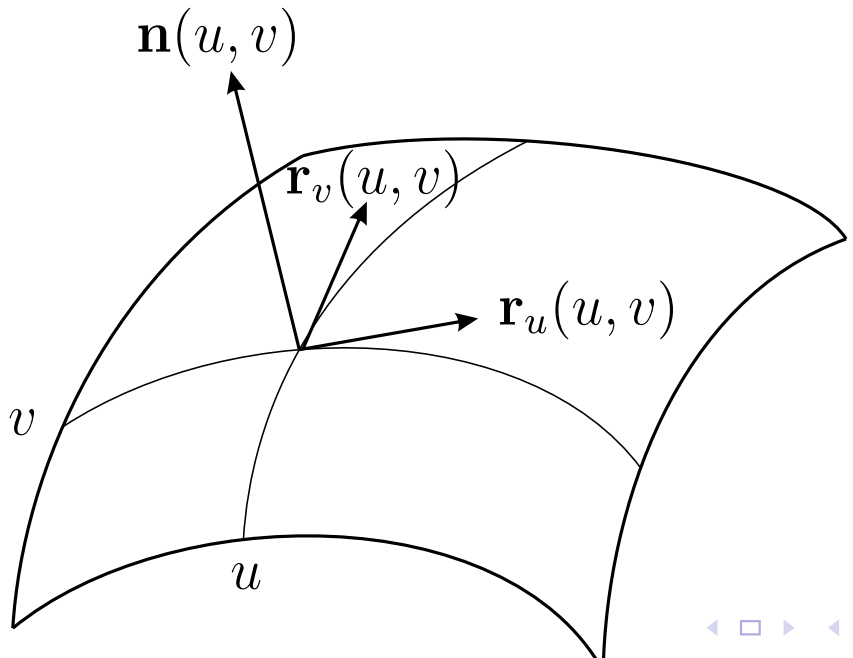
Визуализация поверхностей

Векторное произведение касательных векторов $\mathbf{r}_u(u,v)$ и $\mathbf{r}_v(u,v)$ в любой точке поверхности, заданной в параметрической форме, позволяет определить вектор нормали в этой точке:

$$\mathbf{n}(u,v) = \mathbf{r}_u \times \mathbf{r}_v,$$

где $\mathbf{r}_u = \left[\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u}\right]^T$ и $\mathbf{r}_v = \left[\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v}\right]^T$.

Порядок записи векторов в векторном произведении определяет направление нормали. Когда поверхность определяет границу твёрдого тела важно согласованно записывать вектора в таком порядке, чтобы нормаль всегда была направлена в определённую сторону.



Поверхности второго порядка

В число объектов, наиболее часто используемых в графических приложениях, входят поверхности второго порядка (квадрики), которые описываются уравнениями второго порядка (квадратными) по трем переменным:

$$Ax^2 + By^2 + Cz^2 + Dyz + Ezx + Fxy + Px + Qy + Rz + H = 0.$$

Эти поверхности включают сферы, эллипсоиды, параболоиды и гиперболоиды и в графических пакетах обычно реализуются процедуры генерации этих поверхностей.

Цилиндр

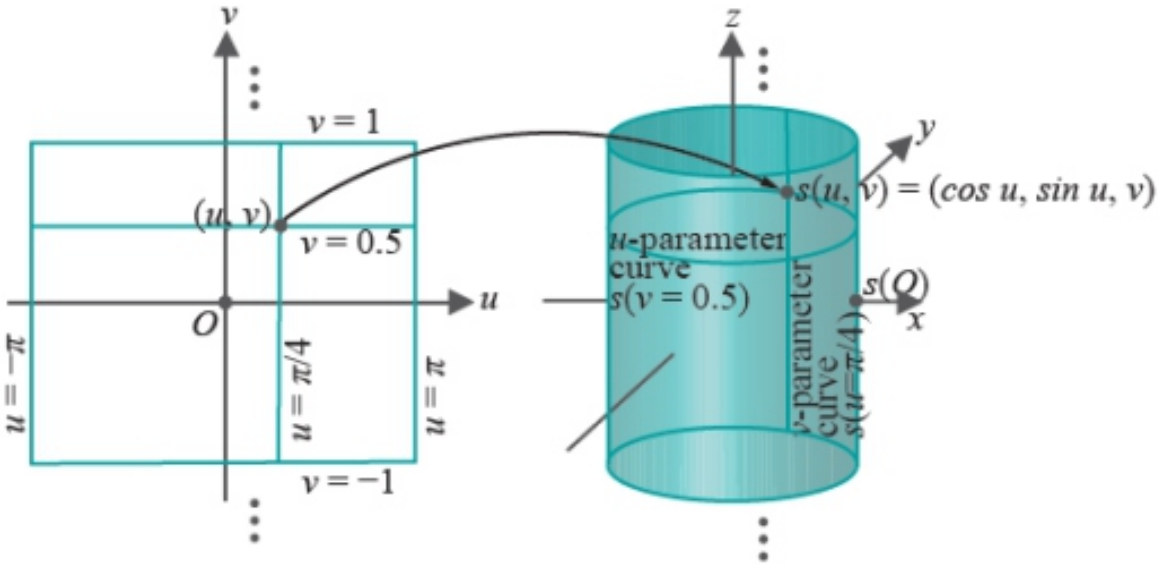


Рис.: Конечная часть пространства параметров, ограниченная линиями $u = \pm\pi$ и $v = \pm 1$, а также соответствующая часть цилиндра

Бесконечно длинный цилиндр с осью вдоль оси z и круговым поперечным сечением радиуса 1 задаётся неявным уравнением

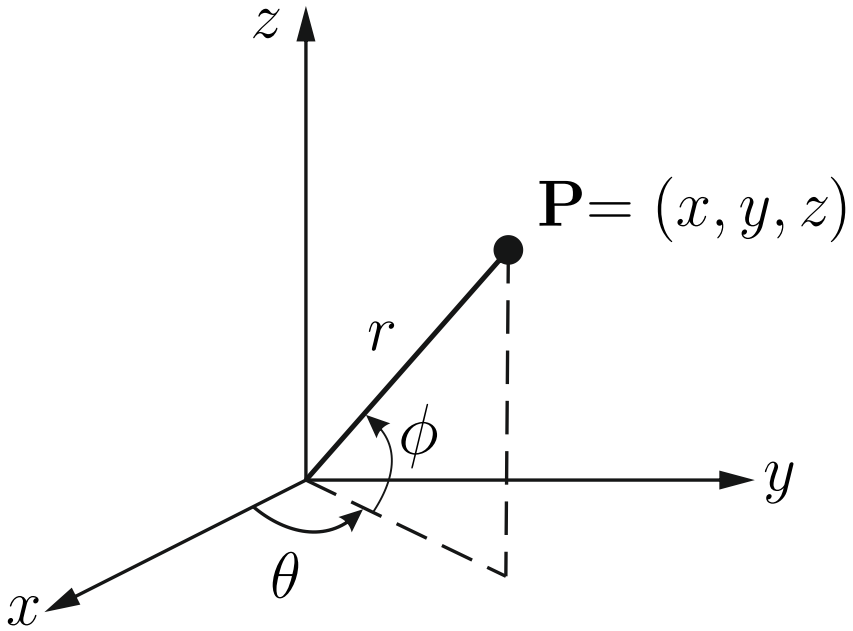
$$x^2 + y^2 = 1$$

Он также задаётся в параметрической форме:

$$x = \cos u, \quad y = \sin u, \quad z = v, \qquad (u, v) \in [-\pi, \pi] \times (-\infty; \infty)$$

где пространство параметров является бесконечно длинным прямоугольным подмножеством плоскости.

Сфера



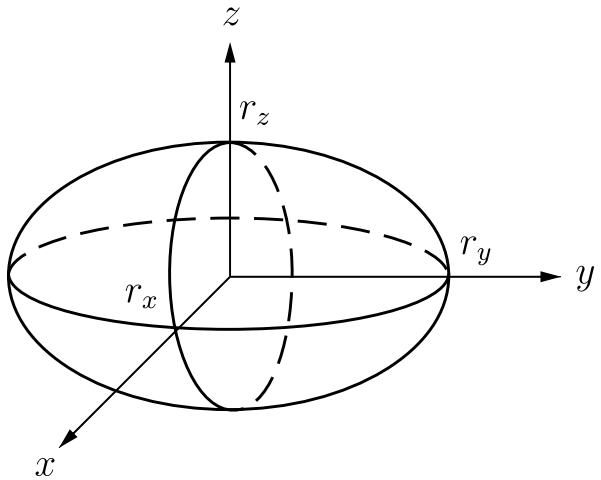
В декартовых координатах сферическая поверхность радиуса r с центром в начале координат определяется как набор точек (x, y, z), удовлетворяющих уравнению:

$$x^2 + y^2 + z^2 = r^2.$$

Сферическую поверхность можно также описать в параметрической форме, используя углы широты и долготы:

$$x = r\cos\phi\cos\theta, \qquad -\pi/2 \leqslant \phi \leqslant \pi/2;$$
$$y = r\cos\phi\sin\theta, \qquad -\pi \leqslant \theta \leqslant \pi;$$
$$z = r\sin\phi.$$

Эллипсоид



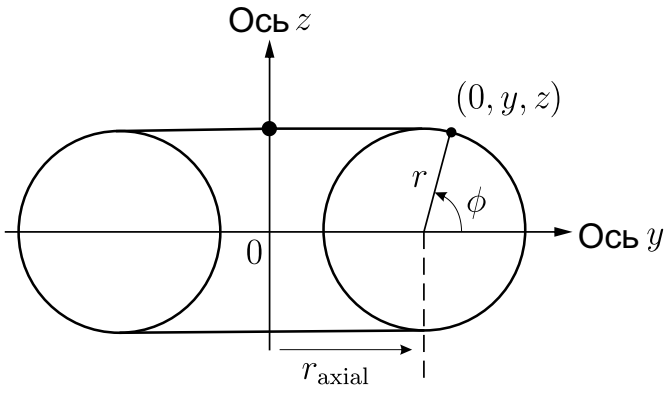
Эллипсоидальную поверхность можно описать как расширение сферической поверхности, где радиусы в трёх взаимно перпендикулярных направлениях могут иметь различные значения. Декартово представление точек на поверхности эллипсоида с центром в начале координат имеет вид:

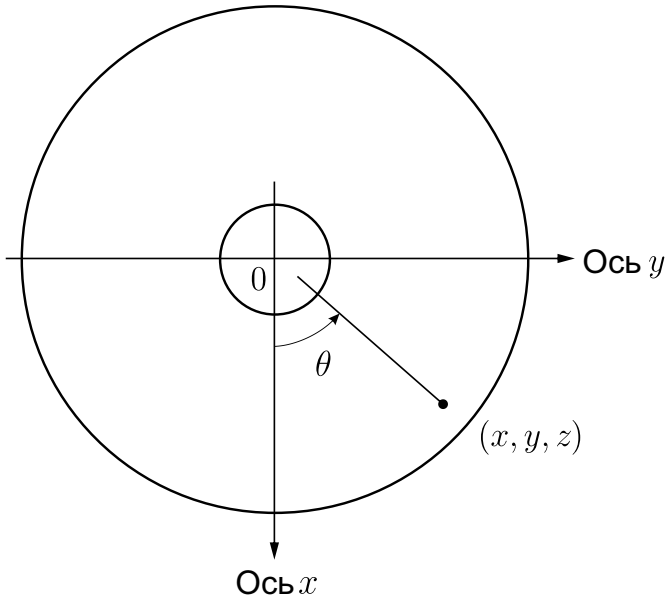
$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1.$$

Параметрическое представление эллипсоида через угол широты ϕ и угол долготы θ выглядит так:

$$\begin{aligned} x &= r_x \cos\phi \cos\theta, \qquad -\pi/2 \leqslant \phi \leqslant \pi/2; \\ y &= r_y \cos\phi \sin\theta, \qquad -\pi \leqslant \theta \leqslant \pi; \\ z &= r_z \sin\phi. \end{aligned}$$

Тор





Наиболее часто тор описывается как поверхность, полученная вращением окружности (или эллипса) вокруг компланарной осевой линии, не проходящей через центр этой окружности. Определяющими параметрами тора являются расстояние от центра окружности до оси вращения r_{axial} и радиус окружности r (предполагается, что $r_{\text{axial}} > r$).
Уравнение поперечного сечения тора (окружности) имеет вид:

$$(y - r_{\text{axial}})^2 + z^2 = r^2.$$

Вращая эту окружность вокруг оси z, получаем тор, точки на поверхности которого в декартовых координатах описываются следующим уравнением:

$$(\sqrt{x^2 + y^2} - r_{\text{axial}})^2 + z^2 = r^2.$$

Тор

Параметрические уравнения тора с круговым сечением имеют вид:

$$x = (r_{\text{axial}} + r\cos\phi)\cos\theta, \qquad -\pi \leqslant \phi \leqslant \pi;$$
$$y = (r_{\text{axial}} + r\cos\phi)\sin\theta, \qquad z = r\sin\phi, \qquad -\pi \leqslant \theta \leqslant \pi.$$

Тор можно также получить вращением не окружности, а эллипса вокруг оси z. Уравнение эллипса в плоскости yz, имеющего большую и малую полуось r_y и r_z, можно записать следующим образом:

$$\left(\frac{y - r_{\text{axial}}}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1,$$

где r_{axial} — расстояние от оси вращения z по оси y до центра эллипса. В результате получается тор, уравнение которого имеет такой вид:

$$\left(\frac{\sqrt{x^2 + y^2} - r_{\text{axial}}}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1.$$

Параметрическое представление тора с эллиптическим сечением:

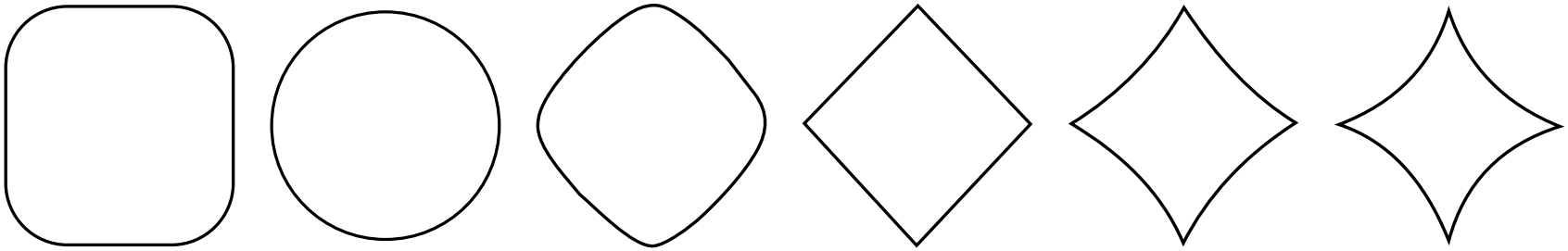
$$x = (r_{\text{axial}} + r_y\cos\phi)\cos\theta, \qquad -\pi \leqslant \phi \leqslant \pi;$$
$$y = (r_{\text{axial}} + r_y\cos\phi)\sin\theta, \qquad z = r_z\sin\phi, \qquad -\pi \leqslant \theta \leqslant \pi.$$

В записанном выше уравнении тора возможны дальнейшие модификации. Например, поверхность тора можно породить вращением окружности или эллипса по эллиптической траектории относительно оси вращения.

Суперквадрики

Данный класс объектов является обобщением поверхностей второго порядка (квадрик). Для получения суперквадрик в уравнение поверхности второго порядка вводятся дополнительные параметры, что даёт большую гибкость в настройке форм объектов. В уравнения кривых вводится один дополнительный параметр, а в уравнениях поверхностей используются два новых параметра.

Суперэллипс



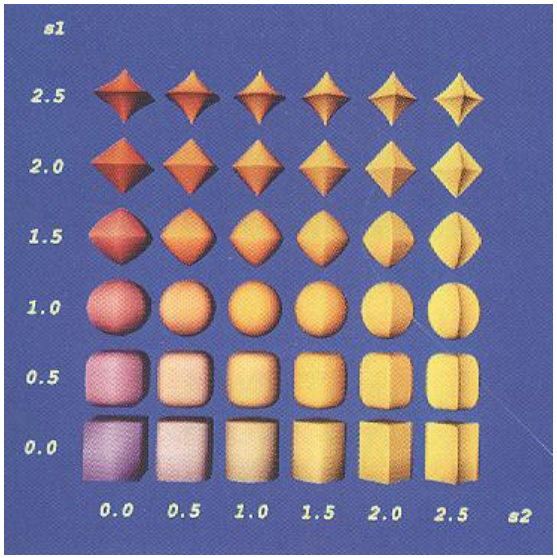
Представление суперэллипса в декартовых координатах получается из соответствующего уравнения эллипса. Одним из возможных вариантов является запись декартова уравнения суперэллипса в такой форме:

$$\left(\frac{x}{r_x}\right)^{2/s} + \left(\frac{y}{r_y}\right)^{2/s} = 1.$$

где параметр s может иметь любое действительное значение. При $s = 1$ получаем обычный эллипс.
Соответствующие параметрические уравнения суперэллипса можно выразить как

$$x = r_x \cdot \mathrm{sign}(\cos\theta) \cdot \cos^s\theta, \qquad -\pi \leqslant \theta \leqslant \pi,$$
$$y = r_y \cdot \mathrm{sign}(\sin\theta) \cdot \sin^s\theta.$$

Суперэллипсоиды



Декартово представление суперэллипсоида получается на основе уравнения эллипсоида введением в него двух степенных параметров:

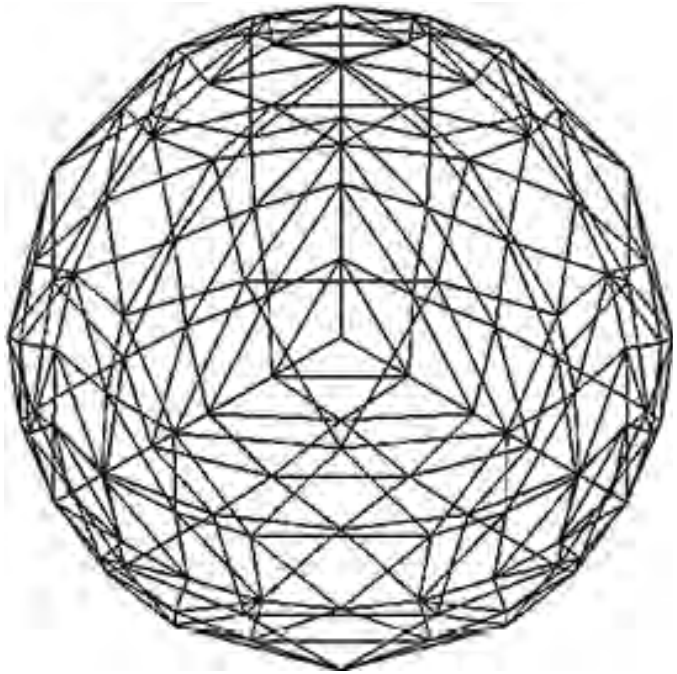
$$\left[\left(\frac{x}{r_x}\right)^{2/s_2} + \left(\frac{y}{r_y}\right)^{2/s_2}\right]^{s_2/s_1} + \left(\frac{z}{r_z}\right)^{2/s_1} = 1.$$

При $s_1 = s_2 = 1$ получается обычный эллипсоид.
Параметрическое представление суперэллипсоида:

$$\begin{aligned} x &= r_x \cdot \mathrm{sign}(\cos\phi) \cdot \mathrm{sign}(\cos\theta) \cdot \cos^{s_1}\phi \cos^{s_2}\theta, & -\pi/2 \leqslant \phi \leqslant \pi/2; \\ y &= r_y \cdot \mathrm{sign}(\cos\phi) \cdot \mathrm{sign}(\sin\theta) \cdot \cos^{s_1}\phi \sin^{s_2}\theta, & -\pi \leqslant \theta \leqslant \pi, \\ z &= r_z \cdot \mathrm{sign}(\sin\theta) \cdot \sin^{s_1}\phi. \end{aligned}$$

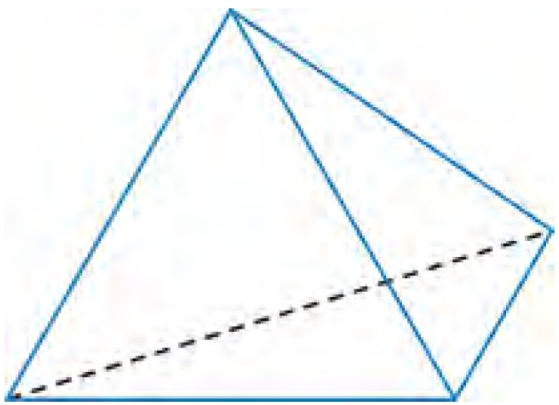
Изосфера

Рассмотрим алгоритм рекурсивного деления для аппроксимаций кривых и поверхностей с любым желаемым уровнем точности на примере аппроксимации поверхности единичной сферы.
Алгоритм построения сферы можно начать с любого правильного многогранника, грани которого можно изначально разделить на треугольники. Например, правильный икосаэдр состоит из 20 равносторонних треугольников и является хорошей отправной точкой для создания сферы. Опишем алгоритм на примере наиболее простой фигуры — тетраэдра.



Изосфера

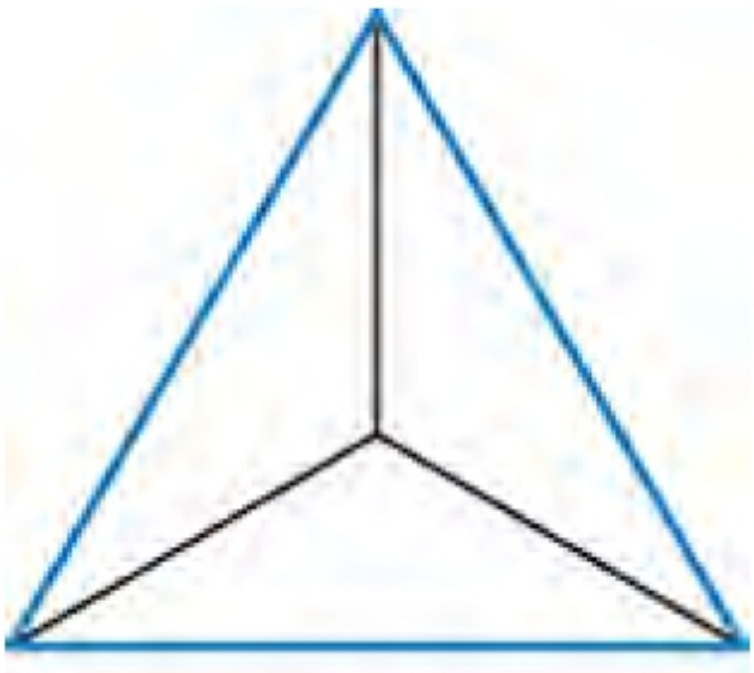
Правильный тетраэдр состоит из четырёх равносторонних треугольников, определяемых четырьмя вершинами: $(0, 0, 1)$, $\left(0, 2\sqrt{2}/3, -1/3\right)$, $\left(-\sqrt{6}/3, -\sqrt{2}/3, -1/3\right)$ $\left(\sqrt{6}/3, -\sqrt{2}/3, -1/3\right)$. Эти точки лежат на единичной сфере с центром в начале координат.

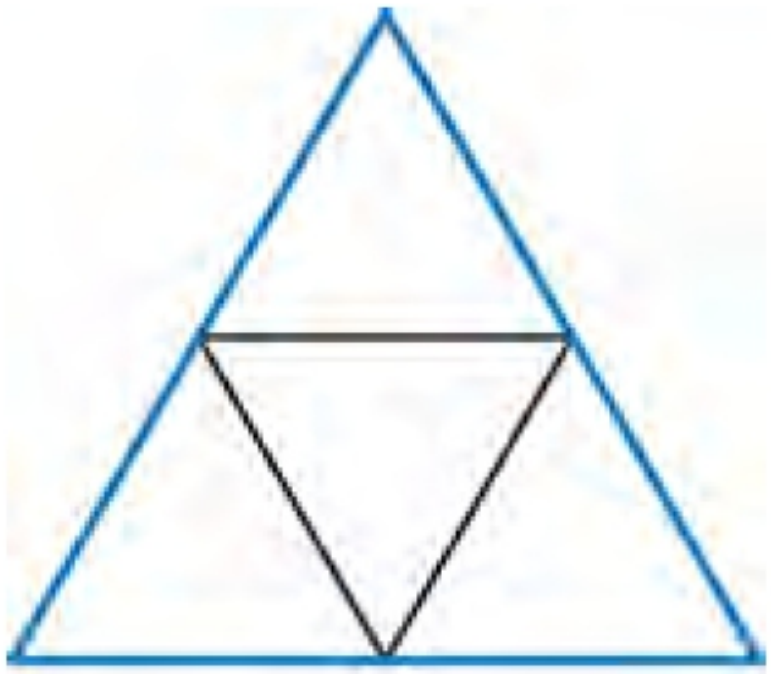


```
va = vec4(0.0, 0.0, -1.0, 1);
vb = vec4(0.0, 0.942809, 0.333333, 1);
vc = vec4(-0.816497, -0.471405, 0.333333, 1);
vd = vec4(0.816497, -0.471405, 0.333333, 1);
```

Изосфера

Получим более близкое приближение к сфере, разделив каждую грань тетраэдра на меньшие треугольники. Например, мы можем вычислить центр масс (центроид) вершин, просто усреднив их, а затем провести линии от этой точки к трем вершинам, создавая три треугольника. Однако этот способ не сохраняет равносторонние треугольники, составляющие правильный тетраэдр. Для сохранения равносторонних треугольников можно воспользоваться алгоритмом, похожим на алгоритм построения треугольника Серпинского. Соединим середины сторон треугольника, образуя четыре равносторонних треугольника.





Изосфера

Далее спроецируем полученные новые вершины на поверхность единичной сферы. Для этого нормализуем их радиус-векторы, чтобы они имели единичную длину.,
Т.о. алгоритм начинается с запуска функции:

```
tetrahedron(va, vb, vc, vd, numTimesToSubdivide);
```

которая делит четыре грани тетраэдра:

```
function tetrahedron(a, b, c, d, n)
{
    divideTriangle(a, b, c, n);
    divideTriangle(d, c, b, n);
    divideTriangle(a, d, b, n);
    divideTriangle(a, c, d, n);
}
```

с помощью алгоритма, похожего на алгоритма построения треугольника Серпинского:

Изосфера

```
function divideTriangle(a, b, c, count)
{
    if (count > 0) {
        ab = normalize(mix(a, b, 0.5));
        ac = normalize(mix(a, c, 0.5));
        bc = normalize(mix(b, c, 0.5));

        divideTriangle( a, ab, ac, count - 1);
        divideTriangle(ab,  b, bc, count - 1);
        divideTriangle(bc,  c, ac, count - 1);
        divideTriangle(ab, bc, ac, count - 1);
    }
    else
        triangle(a, b, c);
}
```

Функция `triangle` добавляет позиции вершин в массив вершин.

Рисование геометрических фигур

WebGL поддерживает следующие графические примитивы: точки, линии, ломаные линии, замкнутые ломаные линии, треугольники, полосы треугольников, вееры треугольников.
Массив данных вершин полностью передаётся в память графического ускорителя посредством специальных буферных объектов. Этот механизм, как правило, обеспечивает самую высокую производительность рендеринга так как данные не передаются через шину ввода/вывода каждый раз, когда происходит рисование.
В вершинном шейдере координаты вершин преобразуются с помощью видовых матриц и матриц проекции, нормали преобразуются обратным транспонированием левой верхней части матрицы модели-вида размером 3×3, текстурные координаты преобразуются с помощью текстурных матриц, накладываются вычисления освещения и свойств материала, вычисляются размеры точек.
После того как определены параметры каждой вершины, начинается этап сборки примитивов. Здесь происходит формирование примитивов: точки из одной вершины, линии из двух вершин и треугольника из трёх вершин.

Команда рисования drawArrays()

При вызове функции gl.drawArrays(), WebGL последовательно считывает данные из загруженного массива вершин и отображает примитивы. Её прототип имеет вид:

```
gl.drawArrays(mode, first, count);
```

Переменная mode определяет, какой тип примитива будет строиться на считываемых вершинах. Допустимые значения: gl.POINTS, gl.LINES, gl.LINE_STRIP, gl.LINE_LOOP, gl.TRIANGLES, gl.TRIANGLE_STRIP, gl.TRIANGLE_FAN. Переменная first определяет номер вершины, с которой должно начаться рисование (целое число), и count определяет количество вершин (целое число), которые нужно нарисовать. Простейший пример вызова этой функции:

```
// Draw one point
gl.drawArrays(gl.POINTS, 0, 1);
```

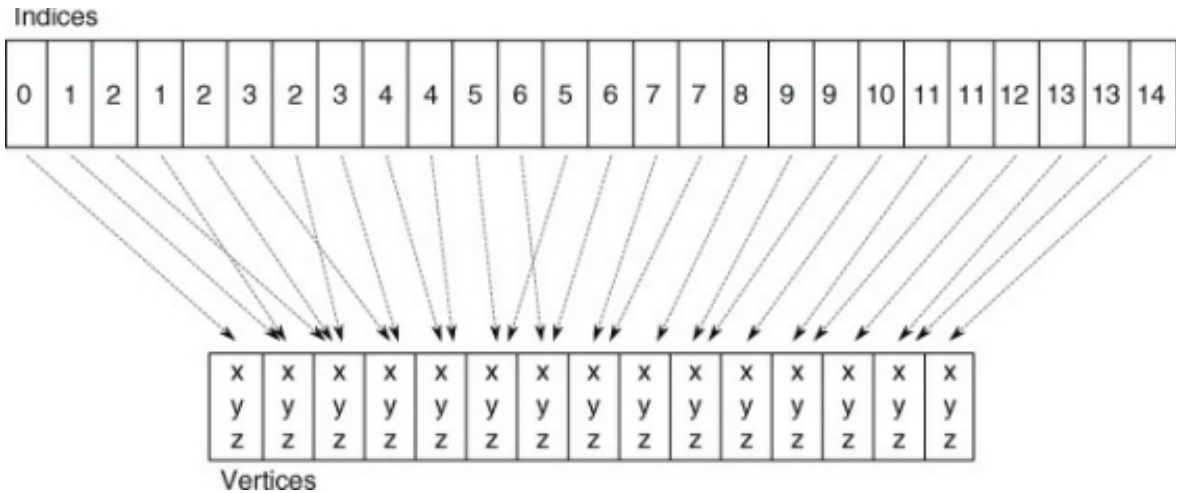
Команда gl.drawArrays() является командой рисования без использования индексов.

Команда рисования drawElements()

Рисование с использованием индексов, включает в себя этап считывания индексов из соответствующего массива для выбора и передачи данных об определяемых этими индексами вершинах.

```
gl.drawElements(mode, count, type, offset);
```

Параметры mode и count имеют то же значение, что и для gl.drawArrays(). Параметр type определяет тип данных, используемый для хранения каждого индекса и может быть gl.UNSIGNED_BYTE (1 байтное число, соответствующее типу Uint8Array) или gl.UNSIGNED_SHORT (2-х байтное число, соответствующее типу Uint16Array). Переменная offset определяет смещение в массиве индексов в байтах, откуда следует начать рисование.



WebGL 2.0

WebGL 2.0 доступна с 27 февраля 2017 года. Она построена на основе OpenGL ES 3.0, для шейдеров поддерживается язык GLSL ES версии 1.00 и 3.00.
WebGL 2.0 несовместима с версией WebGL 1.0 поэтому для подключения её контекста используется отличная от WebGL 1.0 команда:

```
var gl = canvas.getContext('webgl2');
```

Для проверки, работает ли у Вас WebGL 2.0, можно открыть тестовые примеры на сайте:

`http://webglsamples.org/WebGL2Samples/`

В WebGL 2.0 появились новые функции для рисования геометрических фигур, которых не было в WebGL 1.0.

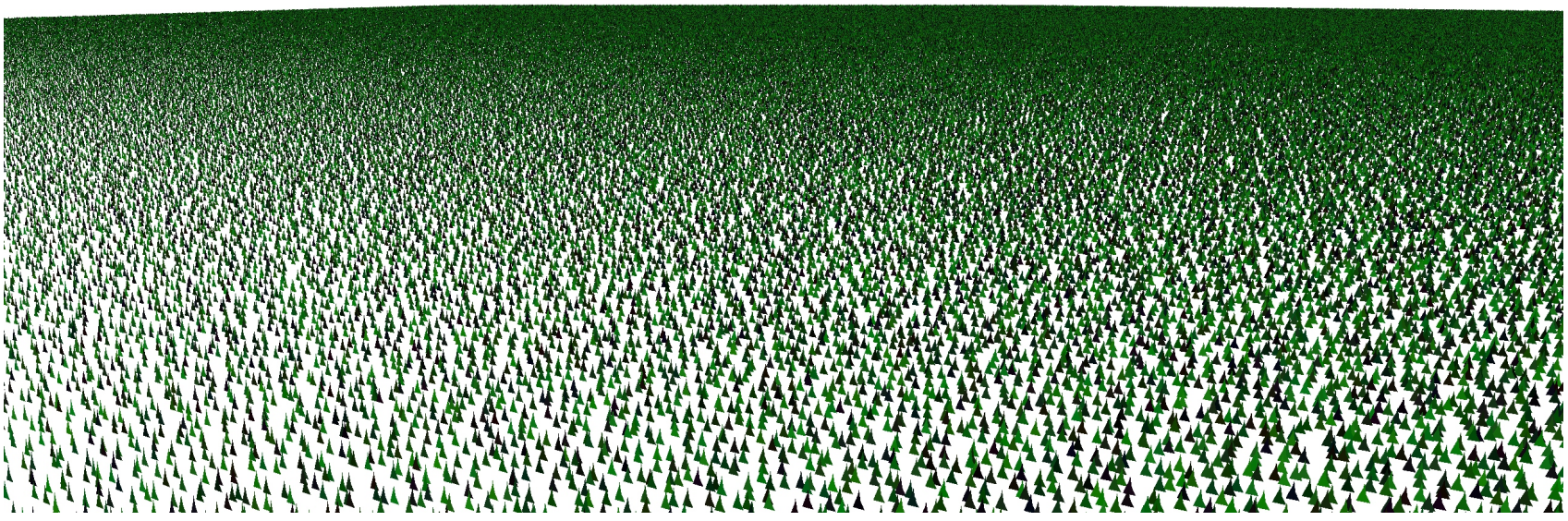
Команда рисования drawRangeElements()

Функция drawRangeElements() выполняет те же действия, что и функция drawElements(). Плюс она позволяет накладывать дополнительные ограничения на диапазон допустимых значений индексов для увеличения быстродействия программы. В этом случае из массива вершин извлекается ограниченное количество данных, которое может полностью поместиться в кэш. Например, если массив вершин содержит 1 000 элементов, а для рисования будут использованы только первые 100, то, в этом случае, можно использовать функцию drawRangeElements() с параметрами start=0 и end=100. Прототип функции имеет следующий вид:

```
gl.drawRangeElements(mode, start, end, count, type, offset);
```

где mode, count, type и offset аналогичны параметрам функции drawElements. Ещё два параметра start и end соответствуют нижней и верхней границе индексов массива вершин. Таким образом, в массиве индексов могут содержаться только значения в диапазоне [start,end]. Указание в массиве индексов значений вне диапазона [start,end] является ошибкой. Однако реализация WebGL не обязательно обнаруживает эту ошибку или рапортует о ней.

Отрисовка множества однотипных объектов



Пусть нужно нарисовать поле на котором растет трава. Оно состоит из тысячи копий по существу идентичных наборов геометрий. Простая реализация устанавливает для каждой травинки соответствующие значения `uniform`-переменных и вызывает функцию отрисовки.
Но травинок могут быть тысячи и миллионы, тогда как количество вершин для представления одной травинки небольшое. Получается, что ресурсы видеокарты будут задействованы не сильно, а большая часть времени будет тратиться на выполнение команд WebGL.

Отрисовка множества однотипных объектов

Для рисования множества копий одной и той же геометрии но с разными параметрами (такими, например, как матрицы трансформации, цвет или размер) существуют специальные разновидности уже знакомых нам функций:

```
gl.drawArraysInstanced(mode, first, count, instanceCount);
gl.drawElementsInstanced(mode, count, type, offset, instanceCount);
```

Эти две функции ведут себя так же, как `gl.drawArrays()` и `gl.drawElements()`, за исключением того, что они сообщают WebGL об отображении множества однотипных объектов. Параметры `mode`, `first` и `count` для функции `gl.drawArraysInstanced()`, и `mode`, `count`, `type` и `offset` для функции `gl.drawElementsInstanced()` имеют те же значения, что и для функций `gl.drawArrays()` и `gl.drawElements()`. Последний аргумент `instanceCount` указывает сколько раз нужно отобразить объект. Если `instanceCount=1`, то `gl.drawArraysInstanced()` и `gl.drawElementsInstanced()` нарисуют объект один раз, что эквивалентно вызову `gl.drawArrays()` или `gl.drawElements()`. Существует два разных способа изменения параметров рисования (например, матриц трансформации, цвета или размера) для однотипных объектов.

Отрисовка множества однотипных объектов

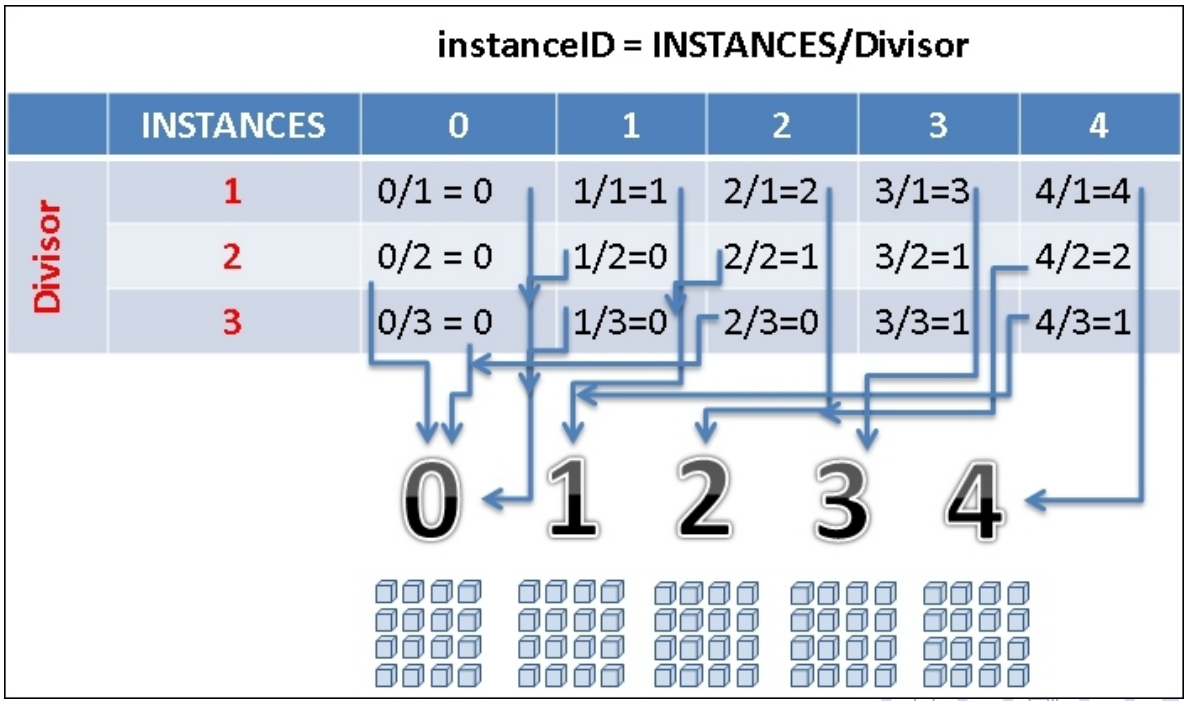
Первый способ основан на преобразовании соответствующих `uniform`-переменных в `attribute`-переменные. Для передачи массива этих данных требуется выделить буферный массив и настроить передачу данных из него в шейдер обычным для `attribute`-переменных образом. Заметим что размерность выделенного массива будет отличаться от размерности массива, который передаёт данные о вершинах. В первом случае она определяется количеством визуализируемых объектов, а во втором количеством вершин объекта.

Отрисовка множества однотипных объектов

Для отделения обычных `attribute`-переменных от `attribute`-переменных, которые передают данные об объекте, используется функция

```
gl.vertexAttribDivisor(location, divisor);
```

Первый параметр этой функции указывает на переменную-атрибут в шейдере, которая будет передавать специфичные данные **для объекта в целом**. Второй параметр `divisor` указывает количество объектов, которые будут иметь указанное значение атрибута. Т.е. не обязательно делать так, чтобы каждый объект имел уникальное значение атрибута, можно нарисовать несколько объектов с одним и тем же значением атрибута.



Отрисовка множества однотипных объектов

Второй способ основан на использовании встроенной в вершинный шейдер переменной `gl_InstanceID`. Она возвращает номер визуализируемого с помощью шейдера объекта (от 0 до `instanceCount-1`) и её можно использовать в качестве индекса в массиве передаваемых в шейдер данных для выбора из него специфичных для объекта значений. В случае, если вызов шейдера осуществляется **не** с помощью функций `gl.drawArraysInstanced()` или `gl.drawElementsInstanced()`, переменная `gl_InstanceID = 0`.

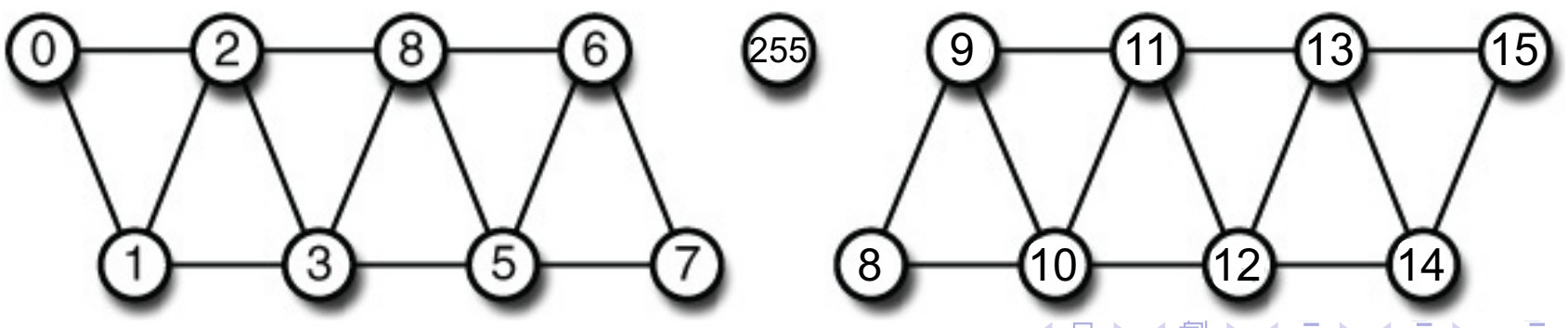
```
attribute vec3 vVertex;
uniform mat4 M[4];
void main()
{
    gl_Position = M[gl_InstanceID] * vec4(vVertex, 1);
}
```

Ускорение отрисовки геометрий

Скорость отрисовки может быть увеличена, если вместо обычных треугольных примитивов использовать их объединения в наборы полос треугольников. Это происходит потому, что каждый отдельный треугольник представлен тремя вершинами, а полоса из треугольников уменьшает это представление до одной вершины на треугольник (не считая первого треугольника в полосе). Если мы перейдем от представления геометрии в виде множества отдельных треугольников к представлению в виде полос, то получим меньше геометрических данных для обработки, и система сможет работать быстрее. Несмотря на это, в реальности визуализация геометрии в виде множества отдельных треугольников может быть выполнена с помощью одного вызова функции `gl.drawArrays()` или `gl.drawElements()`. А если мы визуализируем геометрию в виде набора полос, то для этого может потребоваться больше вызовов этих функций. Если алгоритм генерации полос неоптимальный или если геометрия не поддаётся хорошему представлению в виде полос треугольников, то прироста производительности не будет.

Функции для упрощения представления в виде полос

Функция сброса — ещё один способ уменьшить количество вызовов функций рисования с использованием индексов. Она применяется к примитивам `gl.TRIANGLE_STRIP`, `gl.TRIANGLE_FAN`, `gl.LINE_STRIP` и `gl.LINE_LOOP`. Эта функция сообщает WebGL когда одна полоса (или веер или замкнутая линия) заканчивается и должна начать рисоваться другая. Функция сброса использует специальный маркер, вставленный в массив индексов. Этот маркер является максимальным значением для используемого типа (255 или `0xFF` для индексов типа `UNSIGNED_BYTE`, 65535 или `0xFFFF` для `UNSIGNED_SHORT`, 4294967295 или `0xFFFFFFFF` для `UNSIGNED_INT`). Если это значение встречается в массиве индексов, то происходит завершение команды рисования текущего примитива и тут же запускается следующая команда рисования с теми же параметрами, но со следующего по порядку индекса.
Функция сброса полностью выполняется на стороне GPU и обеспечивает высокую производительность.



Функции для упрощения представления в виде полос

У функции сброса примитива существует менее красивая альтернатива, которая состоит в добавлении индексов, которые приводят к появлению вырожденных треугольников в полосе. Её преимущество состоит в том, что она может использоваться и в WebGL 1.0.
Вырожденный треугольник — это треугольник, в котором две или более вершины совпадают. Вырожденные треугольники имеют нулевую площадь. GPU способен обнаруживать вырожденные треугольники и не проводить их обработку.
Число индексов элементов (или вырожденных треугольников), которые нужно добавить, зависит от типа примитива (полоса или веер). Для полосы также имеет значение количество индексов, определённых в заканчивающейся полосе, так как оно влияет на порядок обхода первого треугольника в новой полосе.

Функции для упрощения представления в виде полос

Когда предыдущая полоса содержит чётное число треугольников, вырождение осуществляется с помощью повторения последнего индекса предыдущей полосы и первого индекса в следующей полосе.

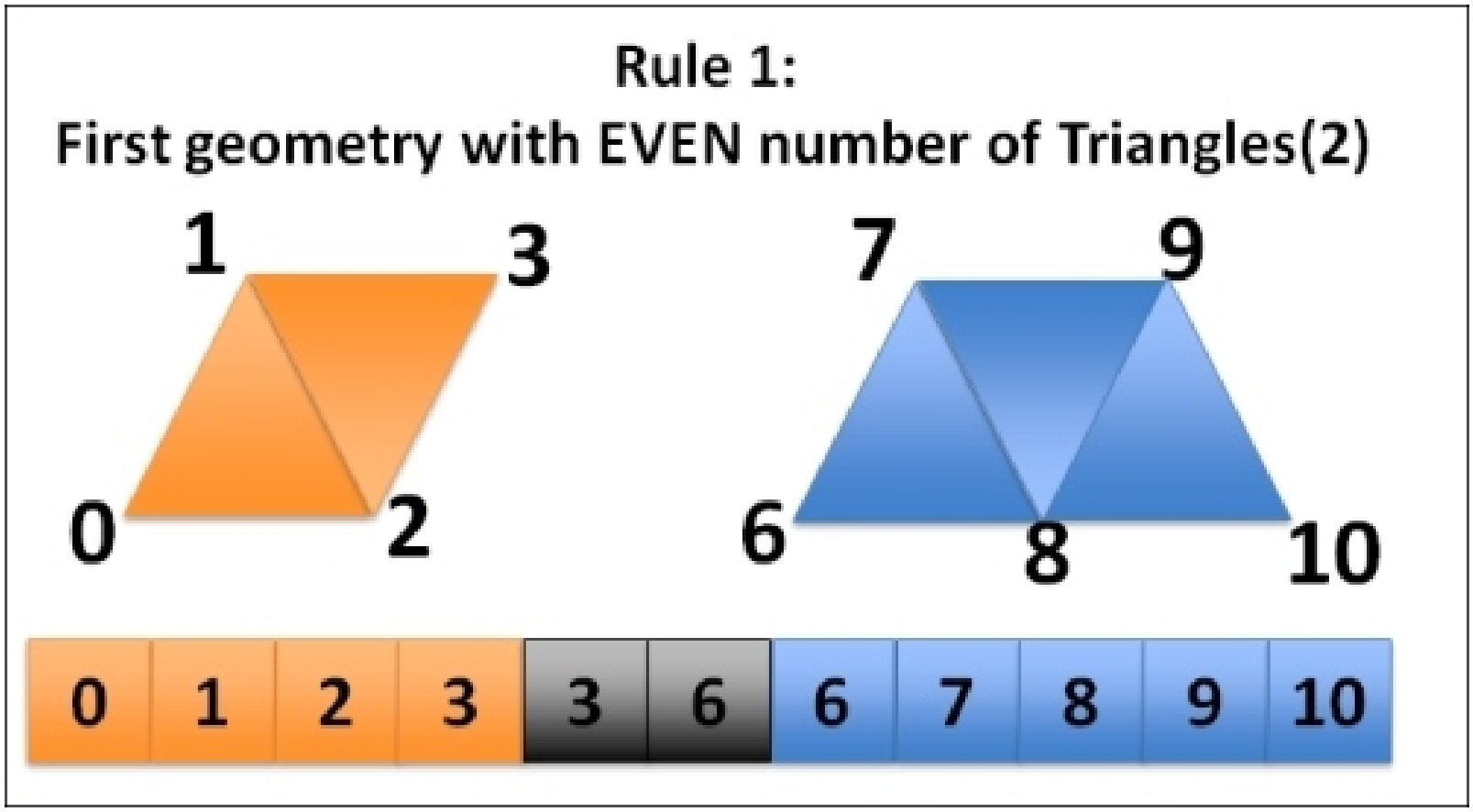


Рис.: Вершины образуют треугольники (0, 1, 2), (2, 1, 3), **(2, 3, 3)**, **(3, 3, 6)**, **(3, 6, 6)**, **(6, 6, 7)**, (6, 7, 8), (8, 7, 9). Жирным выделены вырожденные треугольники

Функции для упрощения представления в виде полос

Если предыдущая полоса содержит нечетное число треугольников, то её последний индекс должен повторяться дважды. Затем, как и в предыдущем случае, следует первый индекс следующей полосы.

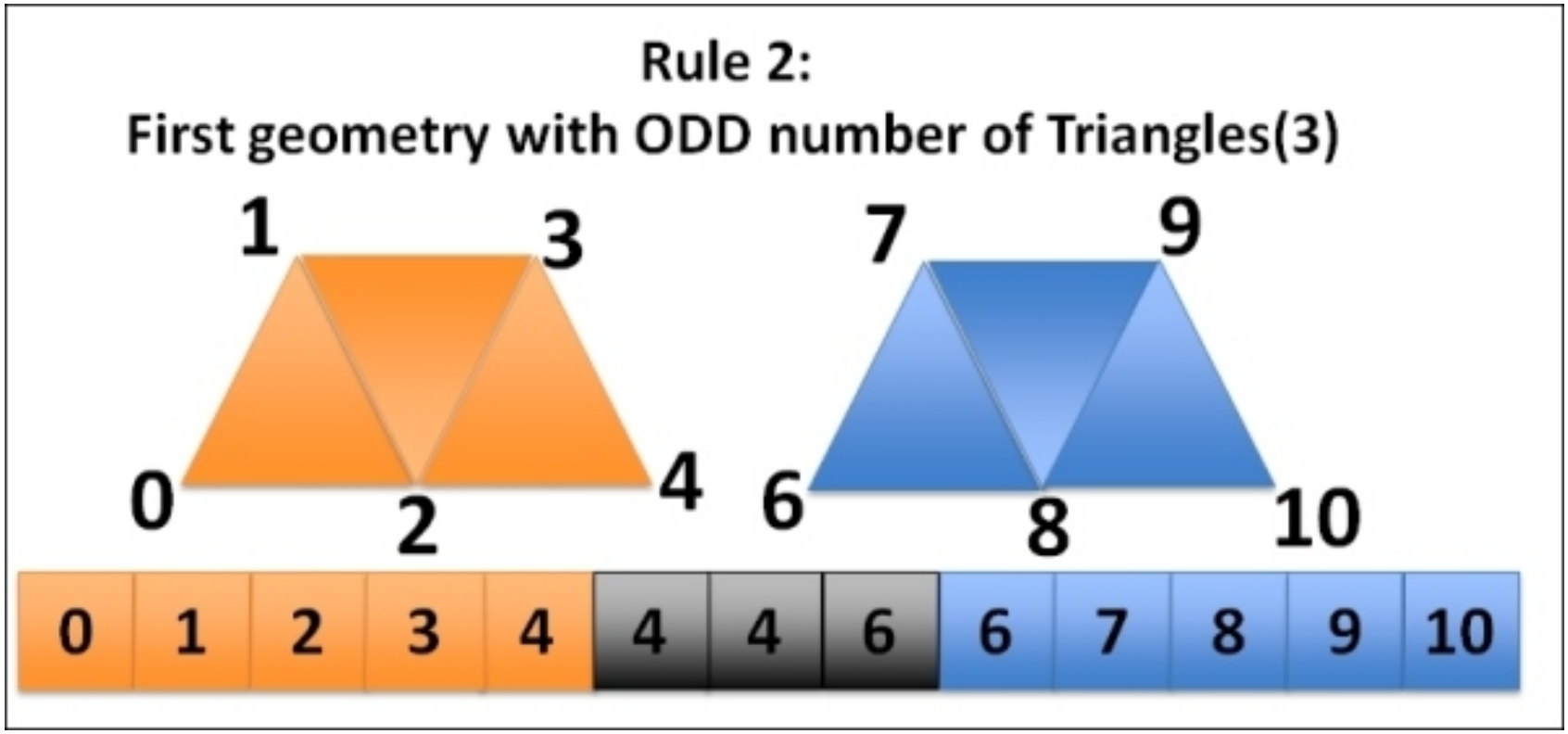


Рис.: Вершины образуют треугольники (0, 1, 2), (2, 1, 3), (2, 3, 4), **(4, 3, 4)**, **(4, 4, 4)**, **(4, 4, 6)**, **(4, 6, 6)**, **(6, 6, 7)**, (6, 7, 8), (8, 7, 9). Жирным выделены вырожденные треугольники