МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Рубцовский индустриальный институт
(филиал) федерального государственного бюджетного
образовательного учреждения высшего образования
«Алтайский государственный технический университет им. И.И. Ползунова»
(РИИ АлтГТУ)

Кафедра «Прикладная математика»

Е.А. ДУДНИК

ГЕОМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ

Лабораторный практикум

Учебно-методическое пособие для студентов направления
«Информатика и вычислительная техника»

Рубцовск 2020

Дудник Е.А. Геометрическое моделирование: методическое пособие для студентов обучающихся по направлению подготовки «Информатика и вычислительная техника» /Рубцовский индустриальный институт. – Рубцовск, ИТО, 2020, – 68 с.

В данном пособии представлен лабораторный практикум по дисциплине «Геометрическое моделирование» рассчитано на обучающихся по направлению подготовки «Информатика и вычислительная техника», знакомых с основами программирования, базовыми геометрическими алгоритмами, линейной алгебры и аналитической геометрии.

Рассмотрены и одобрены
на заседании НМС РИИ.
Протокол № 1 от 27.02.2020.

Рецензент: к.т.н., доцент Э.С. Маршалов

Содержание

Лабораторная работа №1

ПРОСТЕЙШИЕ ТРЁХМЕРНЫЕ ПРИМИТИВЫ

1. Целью работы является самостоятельный расчет координат точек для построения простейших трёхмерных примитивов средствами OpenJSCAD.

2. Приобретаемые знания и навыки

- моделирование трёхмерных объектов в среде OpenJSCAD;
- применение к объектам операций конструктивной сплошной геометрии (CSG).

3. Теоретическая часть

Конструктивная сплошная геометрия (CSG, Constructive Solid Geometry)

Конструктивная сплошная геометрия – это технология, позволяющая моделировать сложные трёхмерные объекты путём комбинации более простых объектов и применения к ним двоичных операций на множествах.

OpenJSCAD использует библиотеку csg.js. Основные трёхмерные примитивы – куб (**CSG.cube**), сфера (***CSG.sphere***), цилиндр (***CSG.cylinder***) и многогранник (***CSG.polyhedron***) – будут рассмотрены в этой работе.

Для построения примитивов потребуется рассчитывать координаты ключевых точек, например, центра объекта (***center***), и указывать их в коде в квадратных скобках. Рекомендуется располагать объекты в I октанте.

Конструктивная сплошная геометрия позволяет применять к объектам операции объединения (***union***), пересечения (***intersect***) и разности (***subtract***), что позволяет создать визуально более сложные объекты.

3.1 Пример применения примитива cube

Синтаксис создания простейшего куба с параметрами по умолчанию:

```
CSG.cube()      // координаты центра [0,0,0],
                // радиус(расстояние от центра до граней) – 1
```

Примитив CSG.cube позволяет создать не только собственно кубы, но также и прямоугольные параллелепипеды разных размеров. Для этого существуют два способа. Первый способ предполагает указание значений параметров ***center*** и ***radius***.

```
CSG.cube({
        center: [0, 0, 0], //координаты центра объекта
        radius: [2, 3, 4]  //радиус
})
```

Во втором способе нужно указать координаты двух противоположных углов куба ***corner1*** и ***corner2***.

```
CSG.cube({
                corner1: [-2, -3, -4],      //куб, заданный координатами
                corner2: [2, 3, 4]          //двух противоположных вершин
})
```

В результате применения обоих способов были созданы одинаковые прямоугольные параллелепипеды с ребрами 4, 6 и 8 и центром в точке *O*.

Обратите внимание: грани куба всегда параллельны координатным плоскостям. Можно изменить это с помощью аффинных преобразований.

ЗАДАНИЕ

Расположите на сцене три параллелепипеда, так, как показано на рисунке. Точка *A* является центром первого параллелепипеда, точка *B* – вершиной третьего. Первый и второй параллелепипеды имеют одну общую вершину, лежащую в плоскости *xOy*. Значения *l1*, *w1*, *l2*, *w2*, *h2*, *l3*, *w3*, а также координаты точек *A* и *B* приведены в таблице. Высоту первого и третьего параллелепипедов предлагается рассчитать самостоятельно.

Пропорции параллелепипедов меняются в зависимости от варианта задания, но расположение их друг относительно друга остаётся неизменным.

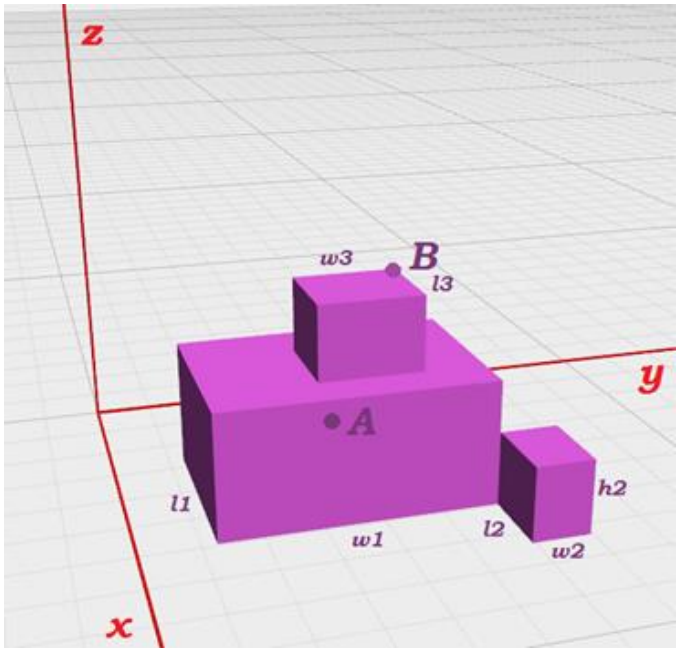


Рис. 1. Три примитива CSG.cube

Варианты

1	xA = 4.8, yA = 4.8, zA = 1.2, xB = 3.7, yB = 5.6, zB = 6.0, l1 = 3.4, w1 = 3.4, l2 = 1.3, w2 = 1.6, h2 = 1.9, l3 = 1.1, w3 = 1.7
2	xA = 7.6, yA = 5.7, zA = 2.6, xB = 7.4, yB = 7.7, zB = 7.3, l1 = 5.5, w1 = 5.8, l2 = 4.1, w2 = 2.6, h2 = 3.6, l3 = 2.3, w3 = 3.3
3	xA = 3.8, yA = 2.8, zA = 1.4, xB = 3.2, yB = 4.6, zB = 4.5, l1 = 3.3, w1 = 4.4, l2 = 1.5, w2 = 1.1, h2 = 2.1, l3 = 1.6, w3 = 1.2
4	xA = 4.0, yA = 3.6, zA = 1.1, xB = 2.8, yB = 3.6, zB = 4.1, l1 = 3.5, w1 = 4.8, l2 = 1.2, w2 = 1.5, h2 = 1.3, l3 = 1.4, w3 = 1.1
5	xA = 5.1, yA = 2.7, zA = 1.6, xB = 3.2, yB = 4.6, zB = 4.5, l1 = 5.3, w1 = 5.2, l2 = 2.3, w2 = 2.6, h2 = 1.7, l3 = 2.1, w3 = 1.7

6	xA = 4.3, yA = 3.6, zA = 1.1, xB = 3.6, yB = 4.9, zB = 3.5, l1 = 3.3, w1 = 4.7, l2 = 1.4, w2 = 1.0, h2 = 1.3, l3 = 1.6, w3 = 1.8
7	xA = 4.5, yA = 3.9, zA = 1.1, xB = 3.9, yB = 4.7, zB = 3.1, l1 = 3.4, w1 = 3.8, l2 = 1.8, w2 = 1.4, h2 = 1.9, l3 = 1.3, w3 = 1.9
8	xA = 4.5, yA = 3.9, zA = 1.1, xB = 2.7, yB = 4.0, zB = 4.1, l1 = 5.4, w1 = 4.3, l2 = 1.2, w2 = 1.4, h2 = 1.7, l3 = 1.8, w3 = 1.5
9	xA = 5.1, yA = 2.7, zA = 1.6, xB = 4.4, yB = 2.7, zB = 4.5, l1 = 3.4, w1 = 3.4, l2 = 1.3, w2 = 1.6, h2 = 1.9, l3 = 1.1, w3 = 0.7
10	xA = 3.8, yA = 2.8, zA = 1.4, xB = 3.2, yB = 3.6, zB = 4.5, l1 = 5.8, w1 = 3.3, l2 = 1.2, w2 = 1.4, h2 = 1.7, l3 = 1.5, w3 = 1.5
11	xA = 5.6, yA = 2.7, zA = 1.6, xB = 5.4, yB = 3.7, zB = 4.3, l1 = 4.5, w1 = 4.8, l2 = 2.3, w2 = 1.9, h2 = 2.9, l3 = 2.0, w3 = 1.4
12	xA = 5.1, yA = 2.7, zA = 1.6, xB = 3.2, yB = 4.6, zB = 4.5, l1 = 5.3, w1 = 6.2, l2 = 1.5, w2 = 1.1, h2 = 1.9, l3 = 2.1, w3 = 3.6
13	xA = 3.1, yA = 3.8, zA = 1.7, xB = 2.8, yB = 3.6, zB = 4.1, l1 = 3.7, w1 = 3.4, l2 = 1.4, w2 = 1.2, h2 = 2.0, l3 = 1.6, w3 = 1.1
14	xA = 4.8, yA = 4.8, zA = 1.2, xB = 3.2, yB = 4.6, zB = 4.5, l1 = 5.3, w1 = 5.2, l2 = 1.5, w2 = 1.1, h2 = 1.9, l3 = 2.1, w3 = 1.7
15	xA = 4.3, yA = 3.6, zA = 1.1, xB = 3.4, yB = 5.7, zB = 3.3, l1 = 4.5, w1 = 4.8, l2 = 1.2, w2 = 1.5, h2 = 1.6, l3 = 1.4, w3 = 2.1
16	xA = 4.5, yA = 3.9, zA = 1.1, xB = 4.8, yB = 5.2, zB = 3.5, l1 = 3.5, w1 = 3.8, l2 = 2.2, w2 = 1.5, h2 = 1.3, l3 = 1.1, w3 = 2.5
17	xA = 4.8, yA = 4.8, zA = 1.2, xB = 5.4, yB = 6.7, zB = 4.3, l1 = 4.5, w1 = 4.8, l2 = 1.3, w2 = 1.6, h2 = 1.9, l3 = 1.1, w3 = 2.7
18	xA = 4.3, yA = 3.6, zA = 1.1, xB = 2.7, yB = 4.0, zB = 3.1, l1 = 5.8, w1 = 3.3, l2 = 1.2, w2 = 1.4, h2 = 1.7, l3 = 1.5, w3 = 1.5
19	xA = 7.6, yA = 5.7, zA = 2.6, xB = 7.4, yB = 7.7, zB = 7.3, l1 = 5.4, w1 = 5.4, l2 = 1.3, w2 = 1.6, h2 = 1.9, l3 = 1.1, w3 = 1.7
20	xA = 3.1, yA = 3.8, zA = 1.7, xB = 2.8, yB = 3.6, zB = 4.1, l1 = 4.5, w1 = 4.8, l2 = 1.3, w2 = 1.6, h2 = 1.9, l3 = 1.6, w3 = 1.1

3.2. Пример применения примитива sphere

Синтаксис создания сферы с параметрами по умолчанию:

```
CSG.sphere() //радиус 1 и центр в точке O
```

Примитив CSG.sphere так же, как и CSG.cube, определяется значениями параметров ***center*** и ***radius***.

```
CSG.sphere({
        center: [0, 0, 2],
        radius: 3,
        resolution: 128
})
```

Отличительной особенностью этого примитива является параметр ***resolution***, или коэффициент тесселяции. От его значения зависит, на какое количество полигонов будет разбита поверхность сферы. Значение коэффициента по умолчанию равно 12, число полигонов при этом 72. Для коэффициента тесселяции сферы, равного 2^n, число полигонов будет равно 2^{2n-1}.

Рис. 2. Зависимость числа полигонов от коэффициента тесселяции

Таким образом, при увеличении коэффициента время рендеринга модели будет быстро возрастать.

3.3. Пример применения примитива cylinder

Синтаксис создания цилиндра с параметрами по умолчанию:

```
CSG.cylinder()  //радиус 1, координаты центров
                                    //оснований [0,-1,0] и [0,1,0]
```

Чтобы задать координаты центров оснований цилиндра вручную, обращайтесь к параметрам start и end. Число полигонов, на которые разделена боковая поверхность цилиндра, равно значению коэффициента тесселяции resolution.

```
CSG.cylinder({
        start: [0, -2, 0],
        end: [0, 2, 0],
```

```
        radius: 1,
        resolution: 16
    })
```

Радиусы оснований могут различаться, за них отвечают параметры radiusStart и radiusEnd. Если один из них равен 0, то объект становится конусом.

```
    CSG.cylinder({
        start: [0, -1, 0],
        end: [0, 1, 0],
        radiusStart: 1,
        radiusEnd: 2,
        resolution: 16
    })
```

3.4. Операции конструктивной сплошной геометрии

Технология CSG позволяет применять к трехмерным объектам бинарные операции над множествами:

- $A \cup B = \{x: x \in A \vee x \in B\}$ – *объединение*
- $A \setminus B = \{x: x \in A \wedge x \notin B\}$ – *разность*
- $A \cap B = \{x: x \in A \wedge x \in B\}$ – *пересечение*

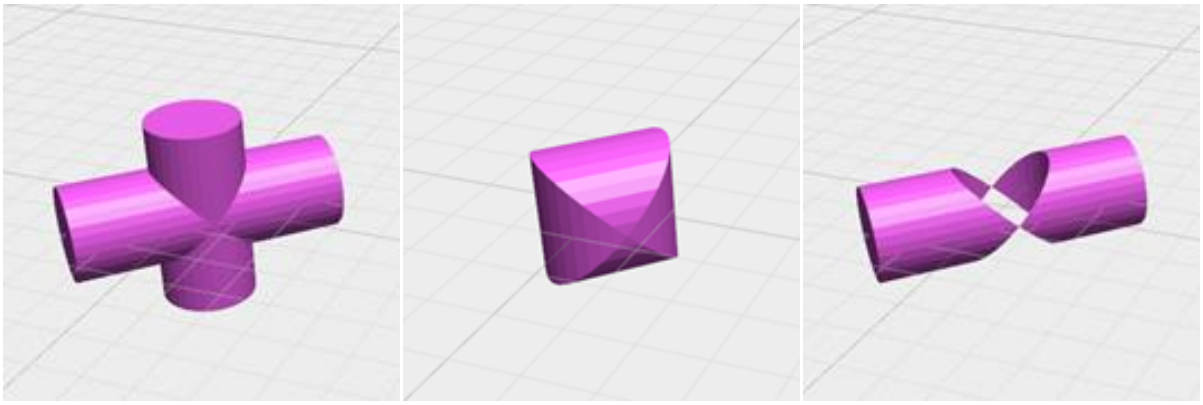


Рис. 3. Объединение, пересечение и разность двух цилиндров

```
CSG.cylinder({
        start:[-2,-2,0],
        end:[2,2,0]
}).union(
        CSG.cylinder({
        start:[0,0,-2],
        end:[0,0,2]
        })
)
```

```
CSG.cylinder({
        start:[-2,-2,0],
        end:[2,2,0]
}).subtract(
        CSG.cylinder({
        start:[0,0,-2],
        end:[0,0,2]
        })
)
```

```
CSG.cylinder({
        start:[-2,-2,0],
        end:[2,2,0]
}).intersect(
        CSG.cylinder({
        start:[0,0,-2],
        end:[0,0,2]
        })
)
```

Рассмотренные операции можно также применять к нескольким объектам одновременно. Для этого объекты, как элементы массива, должны быть перечислены в квадратных скобках через запятую.

```
CSG.cylinder({
        start:[0,0,-2],
        end:[0,0,2]
}).intersect([
        CSG.cylinder({
        start:[-2,-2,0],
        end:[2,2,0]
        }),
        CSG.cylinder({

        start:[0,0,4],
        end:[0,0,-1],

        radiusStart:2,

        radiusEnd:0
        }),
        CSG.cylinder({
        start:[0,0,-4],

        end:[0,0,1],

        radiusStart:2,

        radiusEnd:0
        })
])
```

```
CSG.cylinder({
        start:[0,0,-2],
        end:[0,0,2]
}).subtract([
        CSG.cylinder({
        start:[-2,-2,0],
        end:[2,2,0]
        }),
        CSG.cylinder({

        start:[0,0,4],
        end:[0,0,-1],

        radiusStart:2,

        radiusEnd:0
        }),
        CSG.cylinder({
        start:[0,0,-4],

        end:[0,0,1],

        radiusStart:2,

        radiusEnd:0
        })
])
```

```
CSG.cylinder({
        start:[0,0,-2],
        end:[0,0,2]
}).intersect([
        CSG.cylinder({
        start:[-2,-2,0],
        end:[2,2,0]
        }),
        CSG.cylinder({

        start:[0,0,4],
        end:[0,0,-1],

        radiusStart:2,

        radiusEnd:0
        }),
        CSG.cylinder({
        start:[0,0,-4],

        end:[0,0,1],

        radiusStart:2,

        radiusEnd:0
        })
])
```

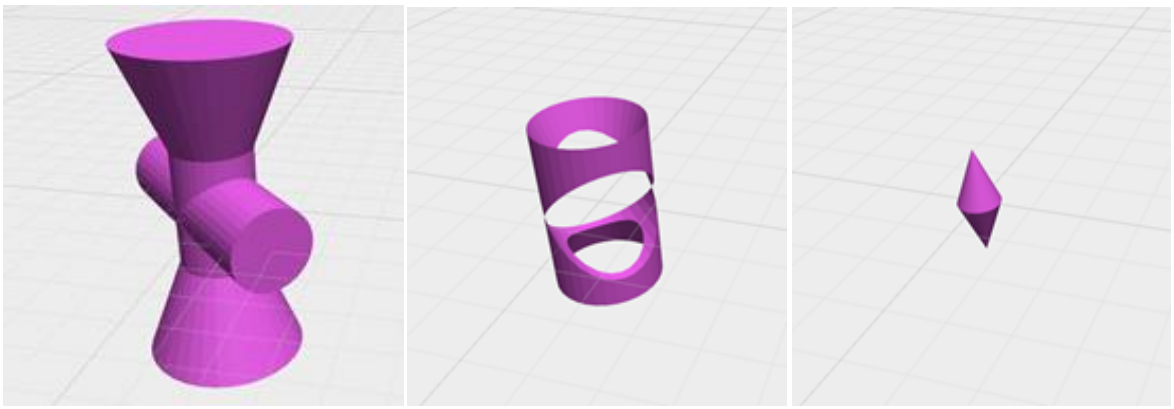


Рис. 4. Объединение, пересечение и разность двух цилиндров и двух конусов

Лабораторная работа №2

ПОСТРОЕНИЕ ПРОСТРАНСТВЕННОЙ МОДЕЛИ

1. Цель

Изучение основ построения чертежа по 3D-технологии по заданным видам модели в среде OpenJSCAD.

2. Приобретаемые знания и навыки

- моделирование трехмерных объектов в среде OpenJSCAD на основе примитивов;
- построение пространственной модели по заданным видам модели.

3. Теоретическая часть

По представленным видам сначала строится аксонометрическое изображение предмета. По нему в последствие строится 3D модель.

Для построения аксонометрической проекции сначала установим положение осей. Оси фронтальной диметрической проекции располагают, как показано на рис. 1, а: ось x – горизонтально, ось z – вертикально, ось y – под углом 45° к горизонтальной линии.

Положение осей изометрической проекции показано на рис. 1, б. Оси x и y располагают под углом 30° к горизонтальной линии (угол 120° между осями).

При построении фронтальной диметрической проекции по осям x и z (и параллельно им) откладывают действительные размеры; по оси y (и параллельно ей) размеры сокращают в 2 раза, отсюда и название «диметрия», что по-гречески означает «двойное измерение».

При построении изометрической проекции по осям x, y, z и параллельно им откладывают действительные размеры предмета, отсюда и название «изометрия», что по-гречески означает «равные измерения».

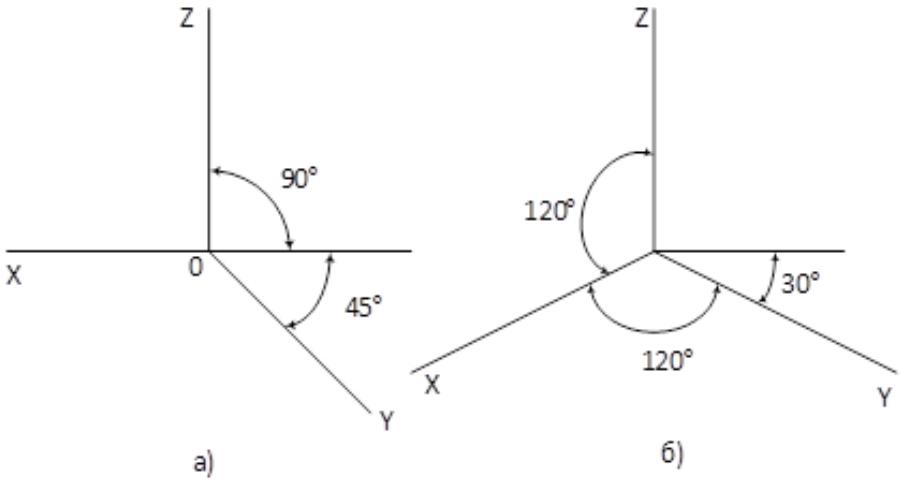


Рис. 1. Оси для построения проекций

4. Задание

По представленным видам необходимо построить аксонометрическую проекцию и визуализировать объект средствами OpenJSCAD.

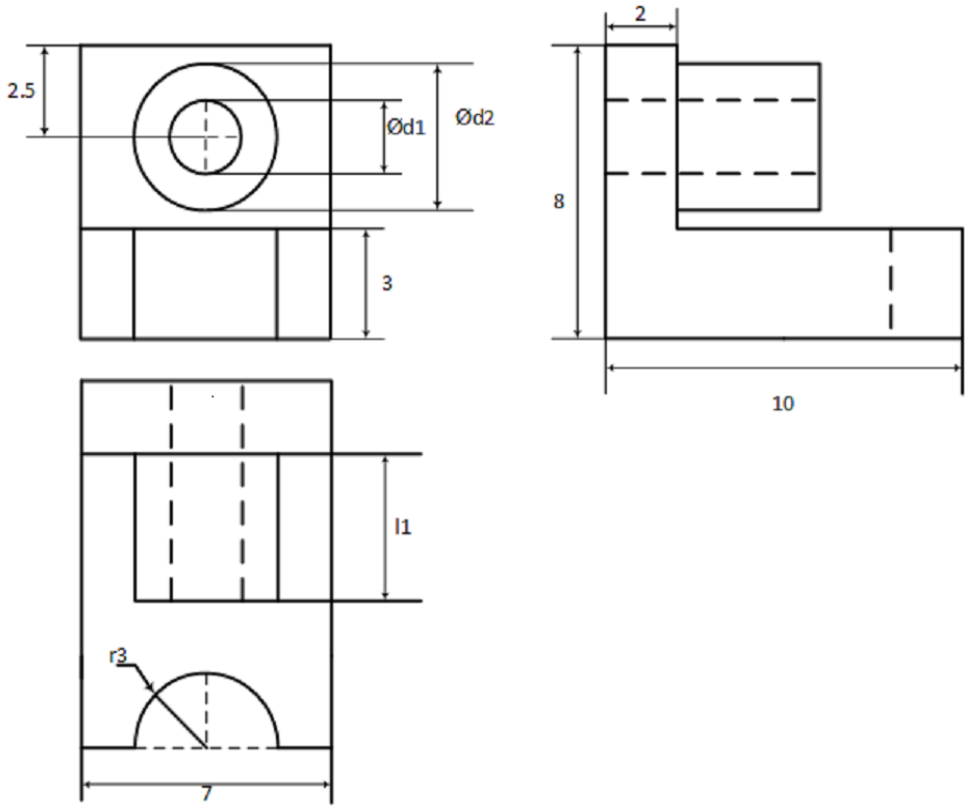


Рис. 2. Горизонтальная, фронтальная и профильная проекции объекта

Варианты заданий:

№ варианта	r1	r2	r3	l1
1	1.5	2	1.5	4.5
2	1.5	2	3	6
3	0.7	1.5	1.5	3
4	2	2.5	2.5	5.5
5	1	2.5	1	4
6	0.5	1.5	2	3.5
7	0.5	1	2	5
8	1	2	2.5	5.5
9	1	2	2	2.5
10	1	2.5	2	4
11	1.5	2.5	2.2	4.5
12	1.7	2.5	2.2	5
13	1.6	2.4	2.1	3.7
14	1.9	2	1.6	4.7
15	2	2.2	2.1	3.6
16	1.8	2.5	1.8	2
17	1.6	1.9	1.9	2.9
18	1.5	2	2.6	3.9
19	1.3	1.9	2.7	4.8
20	1.4	1.9	2.4	4.3

Разрешение всех примитивов при моделировании установить равным 32.

Объект может располагать в любом месте пространства моделирования, но должен быть сориентирован относительно осей координат следующим образом:

- фасад детали расположен параллельно оси Ox;
- вид слева соответственно параллелен оси Oy.

Таким образом, аксонометрическая проекция может быть представлена следующим образом (рис. 3).

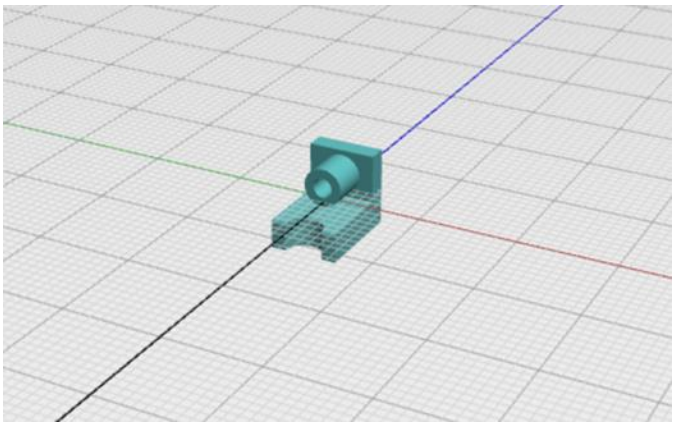


Рис. 3. Ориентация модели в пространстве:

- красная ось – положительная полуось Ox;
- зеленая ось – отрицательная полуось Ox;
- синяя ось – положительная полуось Oy;
- черная ось – отрицательная полуось Oy.

5. Методические рекомендации

Сущность 3D-технологии заключается в создании пространственной виртуальной модели детали. Для построения трехмерной модели детали необходимо представить ее форму как совокупность простых геометрических элементов. Ими могут быть такие предусмотренные в OpenJSCAD примитивы, как: призма, цилиндр, сфера, конус, тор и т.д. Более сложные участки можно представить как тела вращения или выдавливания. Деталь формируется путем объединения, вычитания или пересечения созданных элементов.

Пример построения чертежа:

Создадим новый объект в среде OpenJSCAD. С помощью простых графических примитивов и операций над ними построим пространственную модель представленной на рис. 4 детали.

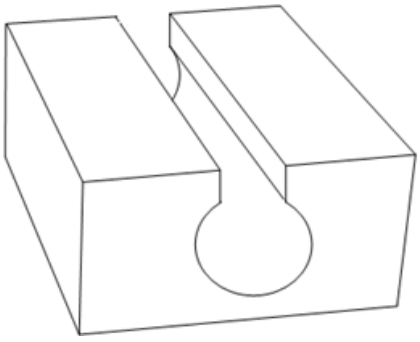


Рис. 4. Аксонометрическая проекция детали

Для этого построим два параллелограмма и цилиндр. Далее вычтем из большего параллелограмма меньший и получившийся цилиндр. Для вычитания одного объекта из другого необходимо выбрать инструмент «Difference».

Этапы выполнения преобразований над объектами представлены на рис. 5.

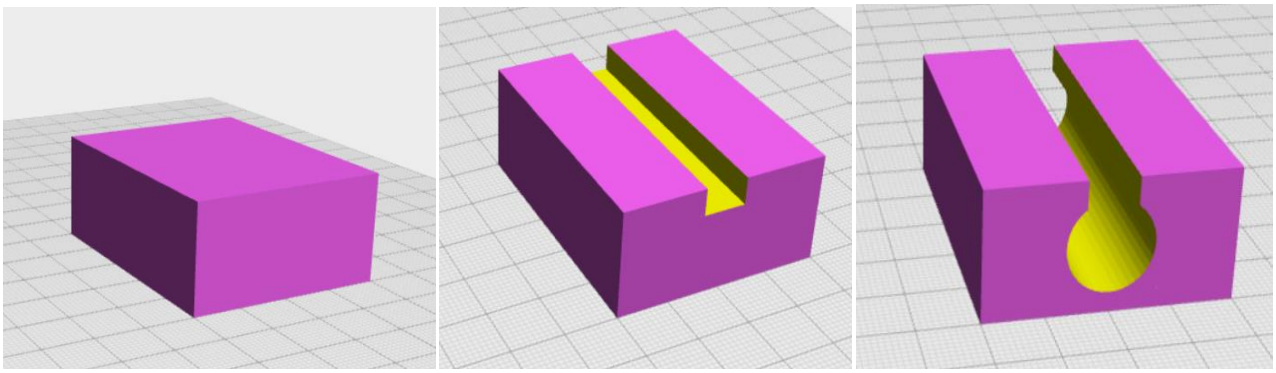


Рис. 5. Этапы построения трехмерной модели детали

Для построения рекомендуется использовать следующие инструменты OpenJSCAD:

- CSG.cylinder({ start: […], end: […], radiusStart, radiusEnd. resolution: 32});
- CSG.cube({ corner1: […], corner2: […]});
- obj.rotateX();
- obj.rotateY();
- obj.rotatez();
- difference((),()).

Лабораторная работа № 3

АФФИННЫЕ ПРЕОБРАЗОВАНИЯ

1. Цель

Изучение аффинных преобразований в трёхмерном пространстве.

2. Приобретаемые знания и навыки

- расчёт координат матрицы преобразования;
- моделирование трехмерных объектов в среде OpenJSCAD на основе рассчитанных координат.

3. Теоретическая часть

3.1. Пространственные преобразования, общий вид

Пространственное преобразование является однозначным, так как каждой точке A в n-мерной системе координат ставится в соответствие некая единственная точка B в N-мерной системе. В этом случае точка B называется ***образом***, а точка A – ***прообразом***. Координаты точки B рассчитываются по формуле:

$$\begin{bmatrix} b_1 = f_1(a_1, a_2, \dots, a_n) \\ b_2 = f_2(a_1, a_2, \dots, a_n) \\ \dots \\ b_N = f_N(a_1, a_2, \dots, a_n) \end{bmatrix},$$

где f_i – ***функция преобразования***, аргументами которой являются координаты точки *A*.

Обратное преобразование выглядит как $a_i = F_i(b_1, b_2 \dots, b_n)$, где F_i – функция обратного преобразования, аргументы – координаты точки *B*.

В случае, когда размерности начальной и конечной систем координат не совпадают, то есть, $n \neq N$, осуществить однозначное преобразование зачастую не удаётся. Например, по двухмерным координатам нельзя без дополнительных условий определить трёхмерные координаты объектов.

По виду функций преобразования различают линейные и нелинейные преобразования. Линейным называется преобразование, чья функция преобразования f_i для всех $i = 1, 2, \dots, N$ является линейной относительно $(a_1, a_2, \dots, a_n)$, то есть $f_i = k_{i1}a_1 + k_{i2}a_2 + \dots + k_{i,n}a_n + k_{i,n+1}$, где k_{ij} – константы.

При $n = N$ (если преобразования координат происходят в системах одной размерности) линейные преобразования называются ***аффинными***.

Аффинное преобразование имеет следующие свойства:

- отображает n-мерный объект в n-мерный – точку в точку, линию в линию, поверхность в поверхность;
- сохраняет параллельность линий и плоскостей;
- сохраняет пропорции параллельных объектов – длин отрезков на параллельных прямых и площадей на параллельных плоскостях.

3.2. Аффинные преобразования в трёхмерном пространстве

Аффинные преобразования можно описать в виде формулы и в матричном виде:

Формула	*Матрица*
$\begin{cases} x' = Ax + By + Cz + D \\ y' = Ex + Fy + Gz + H \\ z' = Kx + Ly + Mz + N \end{cases}$	$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = [T] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$

где *x, y, z* – координаты точки-прообраза, *x', y', z'* – координаты точки-образа после преобразования. В матричном случае координаты точек являются однородными и записываются в виде четвёрки чисел *x, y, z, 1*. Матрица преобразования *T* имеет размер 4x4, элементы матрицы *A, B, C, ..., P* – константы.

$$T = \begin{bmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ M & N & O & P \end{bmatrix}$$

Каждый элемент матрицы отвечает за определённый вид преобразования:

- перенос (*D, H, L*);
- сдвиг (*B, C, E, G, I, K*);
- масштабирование (*A, F, K, P*);
- зеркальное отображение относительно координатных осей или начала координат (*A, F, K, P*);
- вращение вокруг осей координат (*A, B, C, E, F, G, I, J, K*);
- проецирование (*M, N, O*) (в этой работе не рассматривается).

Любое аффинное преобразование можно представить в виде суперпозиции этих преобразований. Математически всё сводится к последовательному перемножению матриц преобразования четвёртого порядка. К примеру, если сначала выполняется преобразование:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} & A & \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

а затем

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix} = \begin{bmatrix} & B & \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix},$$

то правомерна и такая запись:

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix} = \begin{bmatrix} & B & \end{bmatrix} \begin{bmatrix} & A & \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Однако, вместо двух преобразований можно выполнить только одно:

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix} = \begin{bmatrix} & C & \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

где матрица *C* равна произведению *B* x *A*. Способ умножения матриц рассмотрен в методических указаниях к данной работе.

Рассмотрим более подробно виды аффинных преобразований.

а) Параллельный перенос (трансляция)

$$\begin{bmatrix} 1 & 0 & 0 & D \\ 0 & 1 & 0 & H \\ 0 & 0 & 1 & L \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + D \\ y + H \\ z + L \\ 1 \end{bmatrix}$$

Элементы главной диагонали матрицы переноса приравниваются к единице, остальные элементы первых трёх столбцов приравниваются к нулю. Параметры *D, H, L* – параметры переноса – отвечают за перенос трёхмерных объектов вдоль осей координат. Новые координаты получаются из старых путём суммирования их с параметрами переноса.

Пример переноса параллелепипеда, заданного восемью точками:

$$\begin{bmatrix}1&0&0&3\\0&1&0&4\\0&0&1&5\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&0&4&4&0&0&4&4\\2&4&4&2&2&4&4&2\\0&0&0&0&2&2&2&2\\1&1&1&1&1&1&1&1\end{bmatrix}$$

$$=\begin{bmatrix}3&3&7&7&3&3&7&7\\6&8&8&6&6&8&8&6\\5&5&5&5&7&7&7&7\\1&1&1&1&1&1&1&1\end{bmatrix}$$

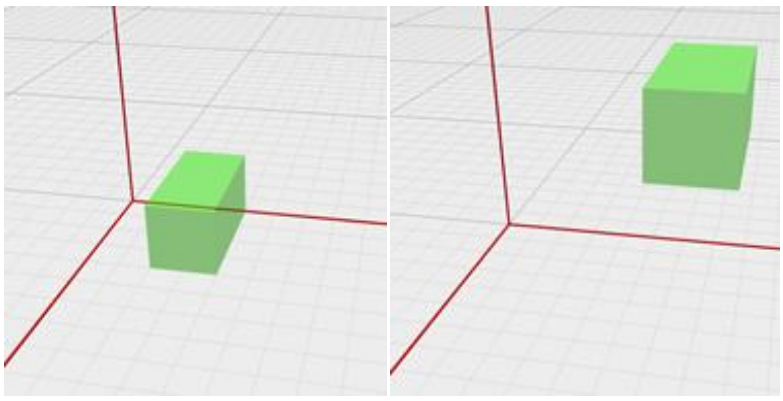


Рис. 1. Перенос

б) Сдвиг

$$\begin{bmatrix}1&B&C&0\\E&1&G&0\\I&J&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}x+Ey+Iz\\Bx+y+Jz\\Cx+Gy+z\\1\end{bmatrix}$$

Элементы главной диагонали матрицы трёхмерного сдвига приравниваются к единице, *D, H, L, M, N, O*приравниваются к нулю. Элементы *B, C, E, G, I, J* – параметры сдвига. Параметры *E, I* сдвигают объект вдоль оси *Ox* пропорционально координатам *y, z*. Аналогично происходит сдвиг вдоль осей *Oy* и *Oz*.

Пример сдвига параллелепипеда, заданного восемью точками:

$$\begin{bmatrix}1&0.2&-0.3&0\\0.5&1&0.6&0\\-0.4&0.4&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}3&3&5&5&3&3&5&5\\2&5&5&2&2&5&5&2\\0&0&0&0&3&3&3&3\\1&1&1&1&1&1&1&1\end{bmatrix}$$

$$=\begin{bmatrix}4&5.5&7.5&6&2.8&4.3&6.3&4.8\\2.6&5.6&6&3&3.8&6.8&7.2&4.2\\0.3&2.1&1.5&-0.3&3.3&5.1&4.5&2.7\\1&1&1&1&1&1&1&1\end{bmatrix}$$

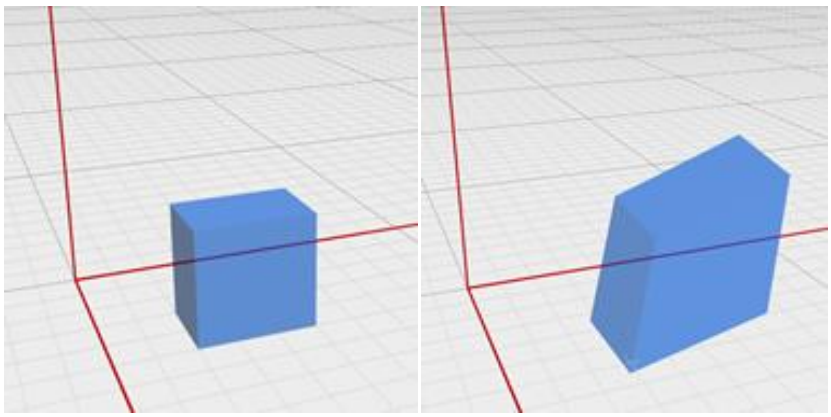


Рис. 2. Сдвиг

в) Масштабирование (сжатие и растяжение)

Диагональные элементы матрицы масштабирования задают локальное и общее масштабирование.

$$\begin{bmatrix} A & 0 & 0 & 0 \\ 0 & F & 0 & 0 \\ 0 & 0 & K & 0 \\ 0 & 0 & 0 & P \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} Ax/P \\ Fy/P \\ Kz/P \\ P/P \end{bmatrix}$$

Здесь A, F, K – коэффициенты масштабирования вдоль осей Ox, Oy, Oz соответственно.

Пример локального масштабирования параллелепипеда, заданного восемью точками:

$$\begin{bmatrix} 0.7 & 0 & 0 & 0 \\ 0 & 0.8 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 3 & 3 & 0 & 0 & 3 & 3 \\ 1 & 3 & 3 & 1 & 1 & 3 & 3 & 1 \\ 0 & 0 & 0 & 0 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 2.1 & 2.1 & 0 & 0 & 2.1 & 2.1 \\ 0.8 & 2.4 & 2.4 & 0.8 & 0.8 & 2.4 & 2.4 & 0.8 \\ 0 & 0 & 0 & 0 & 7.5 & 7.5 & 7.5 & 7.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

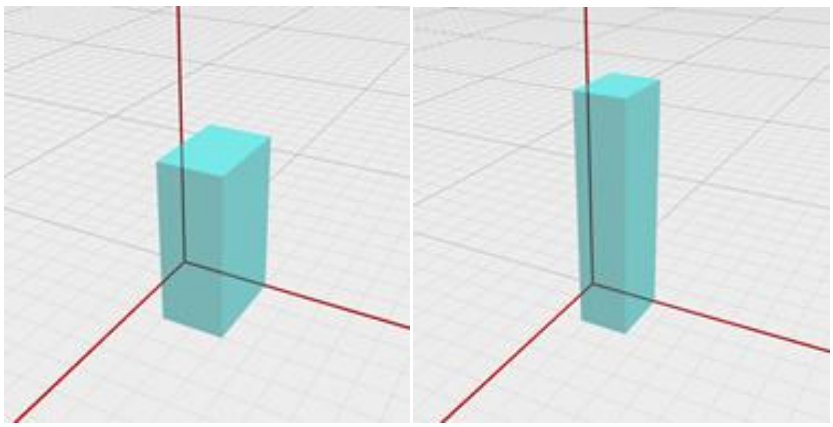


Рис. 3. Локальное масштабирование

Общее масштабирование можно осуществить, изменив четвёртый элемент на диагонали *P*, обычно равный единице. Вместе с ним изменится однородный координатный множитель (четвёртая из однородных координат) каждой из преобразуемых точек. Чтобы получить обычные или физические координаты, каждый координатный вектор требуется умножить на *1/P*.

Пример общего масштабирования:

$$\begin{bmatrix} 0.7 & 0 & 0 & 0 \\ 0 & 0.8 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 0 & 0 & 3 & 3 & 0 & 0 & 3 & 3 \\ 1 & 3 & 3 & 1 & 1 & 3 & 3 & 1 \\ 0 & 0 & 0 & 0 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 4.2 & 4.2 & 0 & 0 & 4.2 & 4.2 \\ 1.6 & 4.8 & 4.8 & 1.6 & 1.6 & 4.8 & 4.8 & 1.6 \\ 0 & 0 & 0 & 0 & 15 & 15 & 15 & 15 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

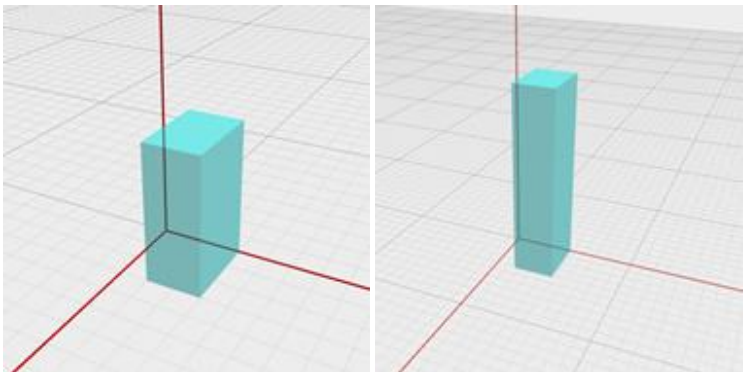


Рис. 4. Общее масштабирование

г) Зеркальное отображение

$$\begin{bmatrix} A & 0 & 0 & 0 \\ 0 & F & 0 & 0 \\ 0 & 0 & K & 0 \\ 0 & 0 & 0 & P \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} Ax/P \\ Fy/P \\ Kz/P \\ P/P \end{bmatrix}$$

Матрица отображения образуется приравниванием всех элементов, кроме элементов главной диагонали, к нулю. Элементы *A, F, K* равны единице, если отображение относительно соответствующей оси не требуется, и минус единице – если требуется. К примеру, если параметр *A* = -1, то координата *x* всех точек меняет свой знак на противоположный. Если параметр *P* равен -1, то все три координаты меняют знак, то есть отображение идёт относительно начала координат, при этом изменять *A, F, K* не требуется.

Если значения параметров по модулю отличны от единицы, то это считается комбинацией масштабирования и отражения.

Пример локального отражения:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \\ 5 \\ 1 \end{bmatrix}$$

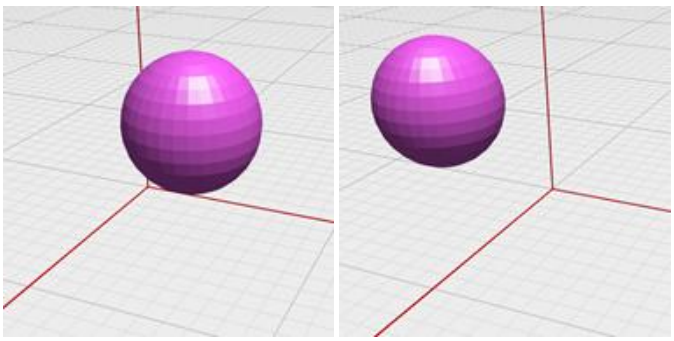


Рис. 5. Локальное отражение

Пример глобального отражения:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ -4 \\ -5 \\ 1 \end{bmatrix}$$

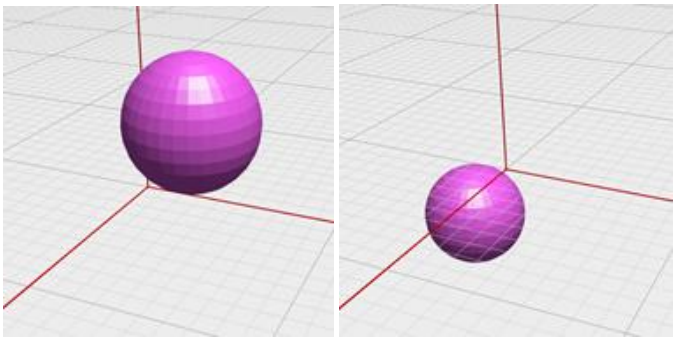


Рис. 6. Общее отражение

д) Вращение

Матрицы вращения вокруг осей *Ox*, *Oy* и *Oz* имеют вид:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} \cos\varphi & 0 & \sin\varphi\,0 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

В матрицах вращения сочетаются масштабирование и сдвиг. От знаков множителей sin φ зависит направление поворота против или по часовой стрелке.

Таблица значений тригонометрических функций sin и cos для наиболее популярных углов приведена в методических указаниях к данной работе.

4. Задание

Задан объект *P* с размерами *Pa, Pb, Pc, Ph*. Составить матрицы преобразований и вершин, перемножить и получить координаты преобразованного объекта.

Порядок преобразований:

1. перенести на D вдоль оси Ox, на H вдоль оси Oy и на L вдоль Oz;
2. повернуть на φ градусов вокруг какой-либо координатной оси;

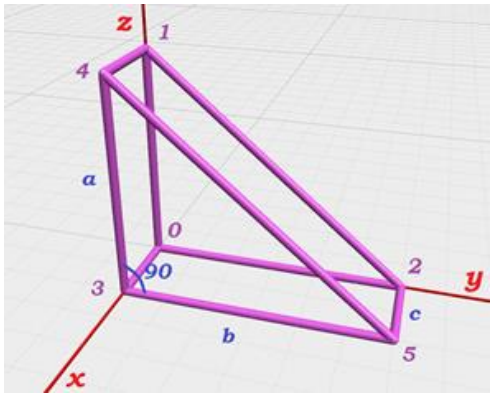
3. отразить относительно начала координат (P = -1) либо относительно одной из осей (A = -1, F = -1 или K = -1 соответственно);

4. сдвинуть (параметры сдвига: B, C, E, G, I, J);

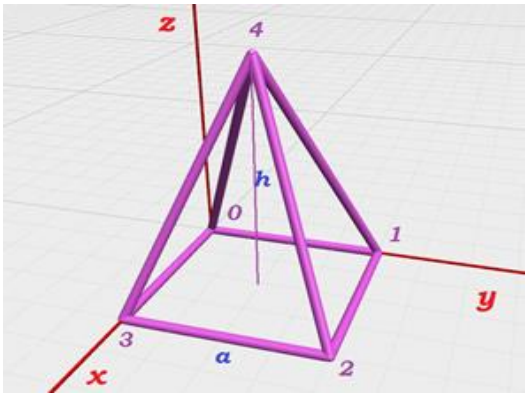
5. растянуть/сжать (параметры масштабирования: A, F, K, P.

Исходные объекты *P* четырёх видов представлены на рис. 7. Точка *0* каждого объекта находится в начале координат.

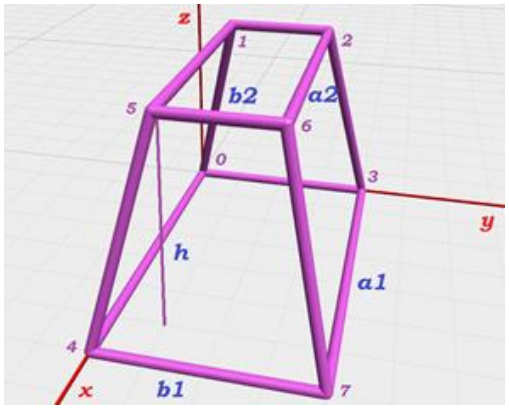
Пример решения рассмотрен в методических указаниях к работе.



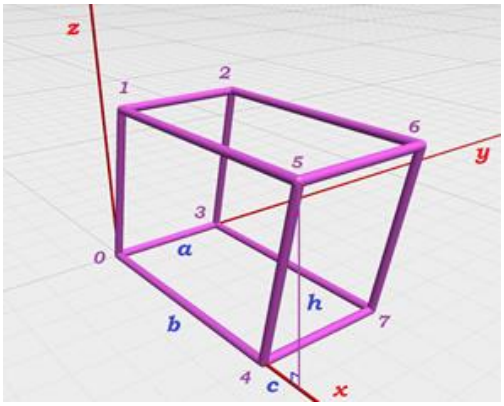
а) треугольная призма



б) квадратная пирамида



в) усеченная пирамида



г) наклонный параллелепипед

Рис. 7. Многогранники

№	Объект, рис. №	Размеры	Преобразования
1	7а	Pa = 4 Pb = 5 Pc = 2	1) D = -2, H = 3, L = 6 2) φ = 45, ось Oz 3) Отражение относительно начала координат 4) B = 0.3, C = -0.5, E = 0.8, G = 1.3, I = 1, J = 0.6 5) A = 0.9, F = 0.5, K = 1.1, P = 1
2	7б	Pa = 4 Ph = 5	1) D = -5, H = 1, L = -2 2) φ = 120, ось Oy 3) Отражение относительно оси Ox 4) B = -0.3, C = 1, E = 0.8, G = -0.7, I = 0.8, J = 0.4 5) A = 0.5, F = 0.8, K = 0.8, P = 1

3	7в	Pa1 = 7 Pb1 = 4 Pa2 = 5 Pb2 = 2 Ph = 4	1) D = -1, H = 4, L = 2 2) φ = 75, ось Oy 3) Отражение относительно оси Oy 4) B = 1, C = 0.3, E = 0.5, G =-0.4, I = -0.5, J = 0.5 5) A = 0.7, F = 0.5, K = 1.2, P = 1
4	7г	Pa = 3 Pb = 6 Pc = 1 Ph = 4	1) D = 3, H = 4, L = 1 2) φ = 105, ось Ox 3) Отражение относительно начала координат 4) B = 1.1, C = 0.2, E = -0.4, G = -0.9, I = 0.6, J = 1 5) A = 1, F = 0.5, K = 0.5, P = 1
5	7а	Pa = 6 Pb = 7 Pc = 4	1) D = 4, H = -2, L = 5 2) φ = 60, ось Oz 3) Отражение относительно оси Oz 4) B = 0.7, C = 1.2, E = -0.5, G = 0.7, I = 0.4, J = 1 5) A = 0.8, F = 0.5, K = 0.7, P = 1
6	7б	Pa = 5 Ph = 6	1) D = 4, H = 3, L = -1 2) φ = 60, ось Ox 3) Отражение относительно начала координат 4) B = 0.9, C = 0.5, E = 1, G = -0.3, I = -0.7, J = 0.8 5) A = 0.4, F = 0.4, K = 0.3, P = 1
7	7в	Pa1 = 8 Pb1 = 6 Pa2 = 4 Pb2 = 4 Ph = 5	1) D = -2, H = 3, L = 6 2) φ = 45, ось Oz 3) Отражение относительно начала координат 4) B = 0.3, C = -0.5, E = 0.8, G = 1.3, I = 1, J = 0.6 5) A = 0.9, F = 0.5, K = 1.1, P = 1
8	7г	Pa = 2 Pb = 5 Pc = 2 Ph = 5	1) D = -2, H = -4, L = 5 2) φ = 135, ось Oy 3) Отражение относительно начала координат 4) B = 0.9, C = -0.5, E = 1, G = 0.9, I = 0.3, J = 0.7 5) A = 0.4, F = 0.5, K = 0.5, P = 1
9	7а	Pa = 6 Pb = 8 Pc = 2	1) D = -1, H = 4, L = 2 2) φ = 75, ось Oy 3) Отражение относительно оси Oy 4) B = 1, C = 0.3, E = 0.5, G =-0.4, I = -0.5, J = 0.5 5) A = 0.7, F = 0.5, K = 1.2, P = 1
10	7б	Pa = 7 Ph = 7	1) D = 4, H = 3, L = -1 2) φ = 60, ось Ox 3) Отражение относительно начала координат 4) B = 0.9, C = 0.5, E = 1, G = -0.3, I = -0.7, J = 0.8 5) A = 0.4, F = 0.4, K = 0.3, P = 1

11	7в	Pa1 = 6 Pb1 = 7 Pa2 = 2 Pb2 = 3 Ph = 3	1) D = 3, H = 4, L = 1 2) φ = 105, ось Ox 3) Отражение относительно начала координат 4) B = 1.1, C = 0.2, E = -0.4, G = -0.9, I = 0.6, J = 1 5) A = 1, F = 0.5, K = 0.5, P = 1
12	7г	Pa = 6 Pb = 7 Pc = 1 Ph = 6	1) D = -5, H = 1, L = -2 2) φ = 120, ось Oy 3) Отражение относительно оси Ox 4) B = -0.3, C = 1, E = 0.8, G = -0.7, I = 0.8, J = 0.4 5) A = 0.5, F = 0.8, K = 0.8, P = 1
13	7а	Pa = 6 Pb = 7 Pc = 3	1) D = -2, H = -4, L = 5 2) φ = 135, ось Oy 3) Отражение относительно начала координат 4) B = 0.9, C = -0.5, E = 1, G = 0.9, I = 0.3, J = 0.7 5) A = 0.4, F = 0.5, K = 0.5, P = 1
14	7б	Pa = 6 Ph = 8	1) D = -2, H = 3, L = 6 2) φ = 45, ось Oz 3) Отражение относительно начала координат 4) B = 0.3, C = -0.5, E = 0.8, G = 1.3, I = 1, J = 0.6 5) A = 0.9, F = 0.5, K = 1.1, P = 1
15	7в	Pa1 = 5 Pb1 = 7 Pa2 = 3 Pb2 = 5 Ph = 4	1) D = -2, H = -4, L = 5 2) φ = 135, ось Oy 3) Отражение относительно начала координат 4) B = 0.9, C = -0.5, E = 1, G = 0.9, I = 0.3, J = 0.7 5) A = 0.4, F = 0.5, K = 0.5, P = 1
16	7г	Pa = 5 Pb = 5 Pc = 3 Ph = 5	1) D = 4, H = -2, L = 5 2) φ = 60, ось Oz 3) Отражение относительно оси Oz 4) B = 0.7, C = 1.2, E = -0.5, G = 0.7, I = 0.4, J = 1 5) A = 0.8, F = 0.5, K = 0.7, P = 1
17	7а	Pa = 8 Pb = 6 Pc = 3	1) D = 3, H = 4, L = 1 2) φ = 105, ось Ox 3) Отражение относительно начала координат 4) B = 1.1, C = 0.2, E = -0.4, G = -0.9, I = 0.6, J = 1 5) A = 1, F = 0.5, K = 0.5, P = 1
18	7б	Pa = 5 Ph = 7	1) D = -1, H = 4, L = 2 2) φ = 75, ось Oy 3) Отражение относительно оси Oy 4) B = 1, C = 0.3, E = 0.5, G =-0.4, I = -0.5, J = 0.5 5) A = 0.7, F = 0.5, K = 1.2, P = 1

19	7в	Pa1 =6 Pb1 = 8 Pa2 = 4 Pb2 = 6, Ph = 3	1) D = 4, H = 3, L = -1 2) φ = 60, ось Ox 3) Отражение относительно начала координат 4) B = 0.9, C = 0.5, E = 1, G = -0.3, I = -0.7, J = 0.8 5) A = 0.4, F = 0.4, K = 0.3, P = 1
20	7г	Pa = 4 Pb = 7 Pc = 2 Ph = 4	1) D = -2, H = 3, L = 6 2) φ = 45, ось Oz 3) Отражение относительно начала координат 4) B = 0.3, C = -0.5, E = 0.8, G = 1.3, I = 1, J = 0.6 5) A = 0.9, F = 0.5, K = 1.1, P = 1

Двумерные массивы *faces* для каждого из многогранников. *R* – расчётная матрица преобразованных координат.

```
//ПРИЗМА
obj.push(CSG.polyhedron({
  points: R
  faces: [
    [0,1,2],[3,5,4],[3,4,1],[3,1,0],
    [3,0,2],[3,2,5],[5,1,4],[5,2,1]
  ]
}))
```

```
//КВАДРАТНАЯ ПИРАМИДА
obj.push(CSG.polyhedron({
  points: R,
  faces: [
    [0,4,1],[2,1,4],[3,2,4],
     [3,4,0],[2,3,0],[2,0,1]
  ]
}))
```

```
//УСЕЧЕННАЯ ПИРАМИДА
obj.push(CSG.polyhedron({
  points: R,
  faces: [
    [0,1,2],[0,2,3],[0,5,1],[0,4,5],
    [3,2,6],[3,6,7],[4,6,5],[4,7,6],
    [5,2,1],[5,6,2],[7,4,0],[7,0,3]
  ]
}))
```

```
//НАКЛОННЫЙ ПАРАЛЛЕЛЕПИПЕД
obj.push(CSG.polyhedron({
  points: R,
  faces: [
    [0,1,2],[0,2,3],[0,5,1],[0,4,5],
    [3,2,6],[3,6,7],[4,6,5],[4,7,6],
    [5,2,1],[5,6,2],[7,4,0],[7,0,3]
  ]
}));
```

5. Методические рекомендации

5.1. Умножение матриц

Даны матрицы *A* и *B*. Матрица *A* содержит *k* строк и *n* столбцов, матрица *B* – *m* строк и *k* столбцов. Произведением матриц *B x A* является матрица *C* с *n* строк и *m* столбцов.

b_{11}	…	b_{1k}
b_{21}	…	b_{2k}
…	…	…
b_{m1}	…	b_{mk}

a_{11}	…	a_{1n}
a_{21}	…	a_{2n}
…	…	…
a_{k1}	…	a_{kn}

c_{11}	…	c_{1n}
c_{21}	…	c_{2n}
…	…	…
c_{m1}	…	c_{mn}

Каждый элемент c_{ij} матрицы C равен сумме произведений элементов i-ой строки матрицы B на соответствующие элементы j-го столбца матрицы A, то есть $\sum^{k}_{\lambda=1} = b_{i\lambda}a_{\lambda j}$.

5.2. Таблица значений cos и sin для наиболее популярных углов поворота

Значение угла α в градусах	Значение угла α в радианах	sin α	cos α
0	0	0	1
15	π/12	(√3 - 1)/2√2	(√3 + 1)/2√2
30	π/6	1/2	√3/2
45	π/4	√2/2	√2/2
60	π/3	√3/2	1/2
75	5π/12	(√3 + 1)/2√2	(√3 - 1)/2√2
90	π/2	1	0
105	7π/12	(√3 + 1)/2√2	-(√3 - 1)/2√2
120	2π/3	√3/2	-1/2
135	3π/4	√2/2	-√2/2
150	5π/6	1/2	-√3/2
180	π	0	-1
210	7π/6	-1/2	-√3/2
240	4π/3	-√3/2	-1/2
270	3π/2	-1	0
360	2π	0	1

При составлении матриц поворота не требуется считать численные значения тригонометрических функций. Записать элемент матрицы можно двумя способами: Math.sin(Math.PI/**3)**, либо аналогичным ему Math.sqrt(**3**)/**2.**

Обратите внимание, что угол должен быть задан не в градусах, а в радианах, с использованием константы Math.PI.

Если угол положителен, то поворот осуществляется против часовой стрелки в правосторонней системе координат.

5.3. Пример решения задачи.

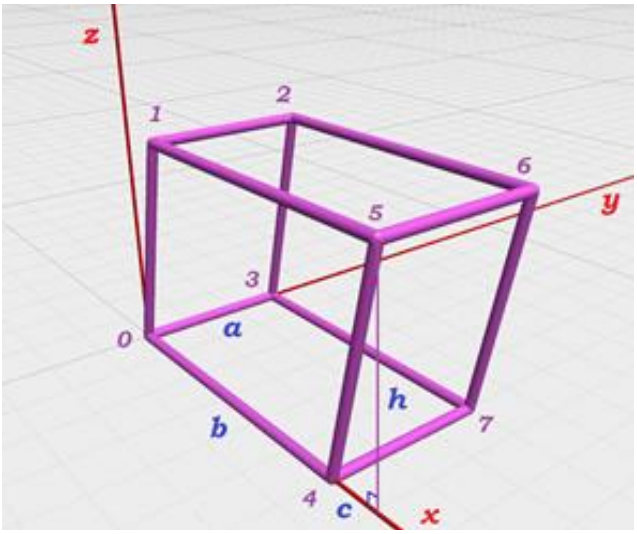


Рис.8. Исходный объект для задания

Объект P: наклонный параллелепипед, Pa = 3, Pb = 6, Pc = 1, Ph = 4.Порядок преобразований: 1) перенести на 3 вдоль оси *Ox*, на 4 вдоль оси *Oy* и на 5 вдоль *Oz*;2) повернуть на 30 градусов вокруг *Oy*; 3) отразить относительно начала координат: *P = 1.*; 4) сдвинуть, параметры сдвига:*B = 0.2, C = 0.3, E =0.1, G = 0.5, I = 0.5, J = 0.3*;5) растянуть/сжать, параметры масштабирования: *A = 0.7, F = 0.8, K = 0.5.*

Решение

Составим матрицу координат вершин *V* размера 8x3. Вершины расположены в порядке нумерации (Рис. 8). В таком виде матрица вершин используется при объявлении параметра CSG.polyhedron. Переведём координаты вершин в однородные, добавив столбец, где все элементы равны 1. После этого транспонируем матрицу. Размер новой матрицы 4x8.

$$V = \begin{bmatrix} x & y & z \\ 0 & 0 & 0 \\ 1 & 0 & 4 \\ 1 & 3 & 4 \\ 0 & 3 & 0 \\ 6 & 0 & 0 \\ 7 & 0 & 4 \\ 7 & 3 & 4 \\ 6 & 3 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} x & y & z & w \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 4 & 1 \\ 1 & 3 & 4 & 1 \\ 0 & 3 & 0 & 1 \\ 6 & 0 & 0 & 1 \\ 7 & 0 & 4 & 1 \\ 7 & 3 & 4 & 1 \\ 6 & 3 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 1 & 0 & 6 & 7 & 7 & 6 \\ 0 & 0 & 3 & 3 & 0 & 0 & 3 & 3 \\ 0 & 4 & 4 & 0 & 0 & 4 & 4 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Матрица общего преобразования *T* размера 4x4 равна произведению матриц всех отдельных преобразований. *T = T5 x T4 x T3 x T2 x T1.*

$$T1 = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T2 = \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \quad T4 = \begin{bmatrix} 1 & 0.2 & 0.3 & 0 \\ 0.1 & 1 & 0.5 & 0 \\ 0.5 & 0.3 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T5 = \begin{bmatrix} 0.7 & 0 & 0 & 0 \\ 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Последовательно умножим матрицы:

$$T5xT4) \begin{bmatrix} 0.7 & 0 & 0 & 0 \\ 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} 1 & 0.2 & 0.3 & 0 \\ 0.1 & 1 & 0.5 & 0 \\ 0.5 & 0.3 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.7 & 0.14 & 0.21 & 0 \\ 0.08 & 0.8 & 0.4 & 0 \\ 0.25 & 0.15 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T5xT4xT3) $\begin{bmatrix} 0.7 & 0.14 & 0.21 & 0 \\ 0.08 & 0.8 & 0.4 & 0 \\ 0.25 & 0.15 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0.7 & 0.14 & 0.21 & 0 \\ 0.08 & 0.8 & 0.4 & 0 \\ 0.25 & 0.15 & 0.5 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$

T5xT4xT3xT2)

$$\begin{bmatrix} 0.7 & 0.14 & 0.21 & 0 \\ 0.08 & 0.8 & 0.4 & 0 \\ 0.25 & 0.15 & 0.5 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} x \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 0.35\sqrt{3} - 0.105 & 0.14 & 0.35 + 0.105\sqrt{3} & 0 \\ 0.04\sqrt{3} - 0.2 & 0.8 & 0.04 + 0.2\sqrt{3} & 0 \\ 0.125\sqrt{3} - 0.25 & 0.15 & 0.125 + 0.25\sqrt{3} & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

T) $\begin{bmatrix} 0.35\sqrt{3} - 0.105 & 0.14 & 0.35 + 0.105\sqrt{3} & 0 \\ 0.04\sqrt{3} - 0.2 & 0.8 & 0.04 + 0.2\sqrt{3} & 0 \\ 0.125\sqrt{3} - 0.25 & 0.15 & 0.125 + 0.25\sqrt{3} & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} x \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$

$$\begin{bmatrix} 0.35\sqrt{3} - 0.105 & 0.14 & 0.35 + 0.105\sqrt{3} & 1.995 + 1.575\sqrt{3} \\ 0.04\sqrt{3} - 0.2 & 0.8 & 0.04 + 0.2\sqrt{3} & 2.8 + 1.12\sqrt{3} \\ 0.125\sqrt{3} - 0.25 & 0.15 & 0.125 + 0.25\sqrt{3} & 0.475 + 1.625\sqrt{3} \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Запись такой матрицы для задания примитива в OpenJSCAD может быть выполнена следующим образом:

[**0.35***Math.sqrt(**3**)-**0.105**,**0.14**,**0.35**+**0.105***Math.sqrt(**3**),**1.995**+**1.575***Math.sqrt(**3**)],[**0.04***Math.sqrt(**3**)-**0.2**,**0.8**,**0.04**+**0.2***Math.sqrt(**3**),**2.8**+**1.12***Math.sqrt(**3**)],[**0.125***Math.sqrt(**3**)-**0.25**,**0.15**,**0.125**+**0.25***Math.sqrt(**3**),**0.475**+**1.625***Math.sqrt(**3**)],[**0**,**0**,**0**,**-1**]

Умножим матрицу преобразований на матрицу координат:

$$TxV = \begin{bmatrix} 0.35\sqrt{3} - 0.105 & 0.14 & 0.35 + 0.105\sqrt{3} & 1.995 + 1.575\sqrt{3} \\ 0.04\sqrt{3} - 0.2 & 0.8 & 0.04 + 0.2\sqrt{3} & 2.8 + 1.12\sqrt{3} \\ 0.125\sqrt{3} - 0.25 & 0.15 & 0.125 + 0.25\sqrt{3} & 0.475 + 1.625\sqrt{3} \\ 0 & 0 & 0 & -1 \end{bmatrix} x \begin{bmatrix} 0 & 1 & 1 & 0 & 6 & 7 & 7 & 6 \\ 0 & 0 & 3 & 3 & 0 & 0 & 3 & 3 \\ 0 & 4 & 4 & 0 & 0 & 4 & 4 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Транспонированная матрица вершин в однородных координатах:

x	y	z	w
1.995+1.575√3	2.8+1.12√3	0.475+1.625√3	-1
3.29 + 2.345√3	2.76 + 1.96√3	0.725 + 2.75√3	-1
3.71 + 2.485√3	5.16 + 1.96√3	1.175 + 2.75√3	-1

2.415 + 1.575√3	5.2 + 1.12√3	0.925 + 1.625√3	-1
1.365 + 3.675√3	1.6 + 1.36√3	-1.025 + 2.375√3	-1
2.66 + 4.445√3	1.56 + 2.2√3	-0.775 + 3.5√3	-1
3.08 + 4.445√3	3.96 + 2.2√3	-0.325 + 3.5√3	-1
1.785 + 3.675√3	4 + 1.36√3	-0.575 + 2.375√3	-1

Разделим все координаты на однородный координатный множитель и удалим последний столбец, для перевода в обычные координаты. Таким образом, итоговая матрица координат выглядит следующим образом:

$$\begin{bmatrix} x & y & z \\ -1.995-1.575\sqrt{3} & -2.8-1.12\sqrt{3} & -0.475-1.625\sqrt{3} \\ -3.29-2.345\sqrt{3} & -2.76-1.96\sqrt{3} & -0.725-2.75\sqrt{3} \\ -3.71-2.345\sqrt{3} & -5.16-1.96\sqrt{3} & -1.325-2.75\sqrt{3} \\ -2.415-1.575\sqrt{3} & -5.2-1.12\sqrt{3} & -0.925-1.625\sqrt{3} \\ -1.365-3.675\sqrt{3} & -1.6-1.36\sqrt{3} & 1.025-2.375\sqrt{3} \\ -2.66-4.445\sqrt{3} & -1.56-2.2\sqrt{3} & 0.775-3.5\sqrt{3} \\ -3.08-4.445\sqrt{3} & -3.96-2.2\sqrt{3} & 0.325-3.5\sqrt{3} \\ -1.785-3.675\sqrt{3} & -4-1.36\sqrt{3} & 0.575-2.375\sqrt{3} \end{bmatrix}$$

Остаётся построить многогранник, где матрица вершин – это матрица координат:

[[-**1.995-1.575***Math.sqrt(**3**),-**2.8-1.12***Math.sqrt(**3**),-**0.475-1.625***Math.sqrt(**3**)],
[-**3.29-2.345***Math.sqrt(**3**),-**2.76-1.96***Math.sqrt(**3**),-**0.725-2.75***Math.sqrt(**3**)],
[-**3.71- 2.345***Math.sqrt(**3**),-**5.16-1.96***Math.sqrt(**3**),-**1.175-2.75***Math.sqrt(**3**)],
[-**2.415-1.575***Math.sqrt(**3**),-**5.2-1.12***Math.sqrt(**3**),-**0.925-1.625***Math.sqrt(**3**)],
[-**1.365-3.675***Math.sqrt(**3**),-**1.6-1.36***Math.sqrt(**3**),**1.025-2.375***Math.sqrt(**3**)],
[-**2.66-4.445***Math.sqrt(**3**),-**1.56-2.2***Math.sqrt(**3**),**0.775-3.5***Math.sqrt(**3**)],
[-**3.08-4.445***Math.sqrt(**3**),-**3.96-2.2***Math.sqrt(**3**),**0.325-3.5***Math.sqrt(**3**)],
[-**1.785-3.675***Math.sqrt(**3**),-**4-1.36***Math.sqrt(**3**),**0.575-2.375***Math.sqrt(**3**)]],

5.4. Пример применения примитива polyhedon

Объект произвольной формы можно создать с помощью CSG.polyhedon. Синтаксис описания этого объекта сложнее, чем у прочих примитивов. Для его создания требуется определить массив координат вершин и массив точек треугольных полигонов, в совокупности составляющих поверхности граней.

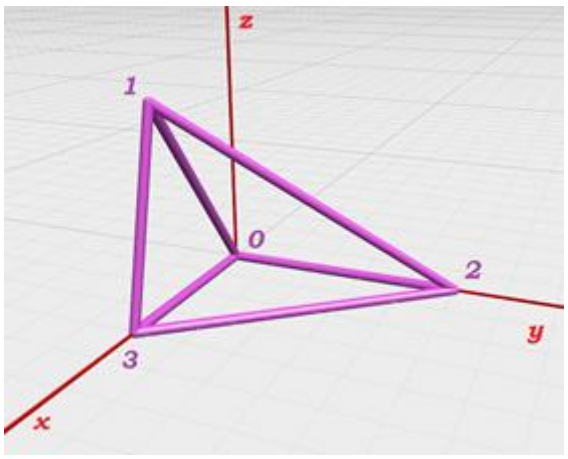


Рис. 9. Прямоугольная пирамида

В качестве примера рассмотрим прямоугольную пирамиду и способ её описания с помощью CSG.polyhedron.

```
CSG.polyhedron({
        points: [[0,0,0],[3,0,4],[0,5,0],[4,0,0]],
        faces: [[0,2,1],[0,3,2],[3,0,1],[1,2,3]]
})
```

Массив ***points*** содержит координаты точек в определенном порядке. Каждой точке неявно ставится в соответствие порядковый номер, начиная с 0. Эти порядковые номера используются в массиве ***faces***, где каждый элемент является, в свою очередь, массивом из трех точек – вершин полигона.

Неважно, с какой точки начинать перечисление вершин каждого полигона. Однако, порядок перечисления точек важен, от него зависит, где будет внешняя сторона грани, а где внутренняя. Основной принцип: если грань повернута к нам внутренней стороной, то точки нужно перечислять против часовой стрелки, а если внешней – то по часовой стрелке.

В рассматриваемом примере только одна грань повернута к нам внешней стороной. Это грань с вершинами в точках 1, 2, 3. Её вершины мы перечисляем по часовой стрелки – [1,2,3]. Вершины всех остальных граней перечисляем против часовой стрелки.

Лабораторная работа № 4

АЛГОРИТМ БРЕЗЕНХЕМА ДЛЯ РИСОВАНИЯ ПРЯМЫХ ЛИНИЙ

1. Цель

Целью данной работы является самостоятельное построение прямой по алгоритму Брезенхема.

2. Приобретаемые знания и навыки

- моделирование двухмерных объектов в среде OpenJSCAD;
- построение прямой по алгоритму Брезенхема.

3. Теоретическая часть

Преимуществом алгоритма Брезенхема является отсутствие арифметики с плавающей точкой и округления. Идеи, использующиеся в этом методе важны еще и потому, что они фигурируют также и в других типах утилит, таких как алгоритмы рисования окружности или эллипса. Алгоритм Брезенхема является классическим примером инкрементного алгоритма, в котором вычисляется положение каждого пикселя вдоль прямой на основе информации о предыдущем пикселе (этот алгоритм иногда называют цифровым дифференциальным анализатором – DDA по названию механического устройства, предназначенного для решения дифференциальных уравнений в приращениях). В нем используются только целые величины, отсутствует умножение, а также содержится компактный и эффективный внутренний цикл, генерирующий искомые пиксели.

Существуют различные вариации алгоритма Брезенхема. Мы будем рассматривать вариант, известный под названием алгоритм средней точки. Этот алгоритм генерирует для прямых линий в точности такие же пиксели, что и алгоритм Брезенхема, и такой подход можно расширить для рисования более сложных форм, например, окружностей и эллипсов. Предположим, что нам даны целочисленные концевые точки прямой (a_x, a_y) и (b_x, b_y). Требуется определить наилучшую последовательность промежуточных пикселов для рисования прямой.

Для упрощения изложения метода рассмотрим частный случай, когда $a_x < b_x$ (то есть точка b лежит справа от точки a) и наклон прямой находится между 0 и 1. Для удобства определим границы отрезка прямой по x и y:

Ширина: $W = b_x - a_x$ (1)

Высота: $H = b_y - a_y$ (2)

В силу установленных нами ограничений H и W являются положительными, и $H < W$. Тогда по мере возрастания x от a_x до b_x соответствующее значение y возрастет от a_y до b_y. Однако y растет медленнее, чем x. При возрастании x на единицу наилучшее целое значение y будет иногда оставаться неизменным, а иногда увеличиваться на единицу. Алгоритм средней точки быстро определяет, которая из этих двух ситуаций имеет место. Еще раз отметим, что y в данном случае никогда не уменьшается и не возрастает более чем на 1.

Уравнение прямой, проходящей через точки (a_x, a_y) и (b_x, b_y), имеет вид:

$$-W(y - a_y) + H(x - a_x) = 0 \quad (3)$$

Левая часть этого уравнения равна 0 для всех точек (x, y), лежащих на этой прямой. Для удобства дальнейшего использования дадим имя этому выражению. Однако перед присвоением имени выражение необходимо удвоить, поскольку это избавит нас от неудобного коэффициента 1/2 в основных формулах. Таким образом, мы определяем следующую функцию:

$$F(x,y) = -2W(y - a_y) + 2H(x - a_x) \quad (4)$$

Функция F(x,y) обладает следующим важным свойством: ее знак определяет, находится ли пара значений (x, y) выше или ниже прямой:

- если $F(x, y) < 0$, то (x, y) лежит выше прямой;
- если $F(x, y) > 0$, то (x, y) лежит ниже прямой.

Пример:

Уравнение отрезка прямой между точками (3, 7) и (9, 11) имеет вид: F(x,y) = –12(y – 7) + 8(x – 3), причем точки на прямой, такие как (7, 29/3), удовлетворяют уравнению F(x, y) = 0. Какой знак имеет функция для точек A = (4, 4) и B = (5, 9) и где по отношению к отрезку прямой лежат эти точки?

Ответ:

Точка A лежит ниже прямой и для нее функция F равна 44. Точка B лежит выше прямой и для нее функция F равна – 8.

Как же мы решим, какие пиксели включить?

На рис.1 изображены некоторые пиксели вблизи прямой. Предположим, что мы откуда-либо знаем, что для p_x наилучшее значение y есть и p_y, и хотим определить наилучшее значение y для следующего значения x, равного $p_x + 1$. Мы хотим знать, находится ли прямая в точке $p_x + 1$ ближе к точке $L = (p_x + 1, p_y)$ или к точке $U = (p_x + 1, p_y + 1)$.

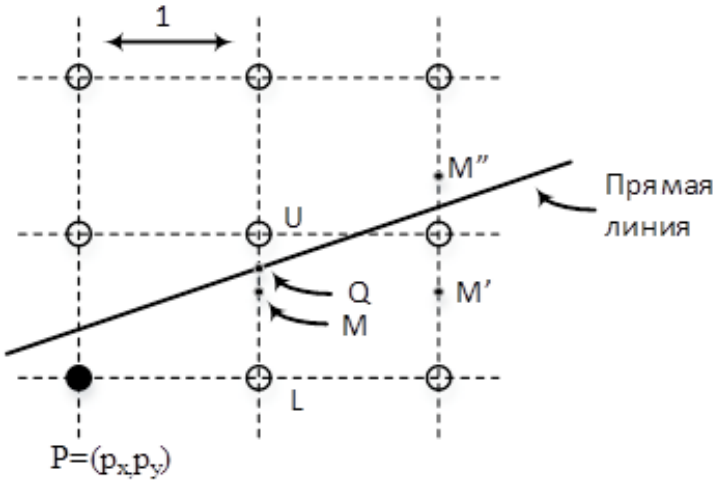


Рис.1. Схема для вывода алгоритма средней точки

Мы принимаем решение, включить пиксель U или L, в зависимости от того, лежит идеальная прямая выше или ниже средней точки $M = (p_x + 1, p_y + 1/2)$, расположенной посередине между точками U и L. Если мы вычислим функцию F в точке M, то ее знак укажет нам, проходит ли идеальная прямая выше или ниже точки M:

- если $F(M) < 0$, то точка M лежит выше прямой, и мы выбираем точку L;
- если $F(M) > 0$, то точка M лежит ниже прямой, и мы выбираем точку U.

Таким образом, если $F(M) > 0$, то включаемый пиксель находится выше, чем предыдущий, поэтому мы увеличиваем y на 1. В противном случае y остается неизменным. Остальное уже дело алгебраической техники: найти способ быстрого вычисления F(x, y). Суть метода заключается в том, чтобы вычислить ее инкрементно, основываясь на том, насколько ее значение должно измениться от одного шага к следующему.

Для точки M:

$F(M) = –2W(p_y + 1/2 – a_y) + 2H(p_x + 1 – a_x)$.

Рассмотрим изменение функции F при переходе от $x = p_x + 1$ к точке $x = p_x + 2$. Как показано на рис.1, следующей средней точкой будет либо M' либо M". Если мы не прибавили к y 1 на предыдущем шаге, то точкой M станет $M' = (p_x + 2, p_y + 1/2)$, а если прибавили, то точкой M станет точка $M'' = (p_x + 2, p_y + 3/2)$.

Случай 1. Если функция F была отрицательна на предыдущем шаге (т.е. значение y не изменялось), то

$F(p_x + 2, p_y + 1/2) = –2W(p_y + 1/2 – a_y) + 2H(p_x + 2 – a_x)$.

Вычтем из данного равенства равенство для F(M), чтобы определить насколько последнее равенство больше, чем F(M): в результате получится F(M) + 2H.

Случай 2. Если функция F была положительна на предыдущем шаге (т.е. значение y увеличилось на 1), то

$F(p_x + 2, p_y + 3/2) = –2W(p_y + 3/2– a_y) + 2H(p_x + 2 – a_x)$.

Вычтем из данного равенства равенство для F(M), чтобы определить насколько последнее равенство больше, чем F(M): в результате получится F(M) – 2(W – H).

В каждом из случаев к «контрольной величине», т.е. к исходному значению функции, добавляется некоторая константа: 2H или –2(W – H).

Осталось определиться, с чего начинать этот процесс. Мы знаем, что при $x=a_x$ $y=a_y$. Поэтому первый экземпляр средней точки $M = (a_x + 1, a_y + 1/2)$ и тогда

$F(x ,y) = –2W(a_y + 1/2 – a_y) + 2H(a_x + 2 – a_x) = 2H – W$.

Если бы мы предварительно не умножили функцию на 2, то результат был бы H – 0,5W, что сделало бы невозможным использование целых значений всех участвующих величин!

Таким образом, перед началом процесса мы полагаем $F = 2H – W$, $x = a_x$, $y = a_y$. Затем на каждом шаге делаем следующее:

- присваиваем пикселю (x, y) нужное значение цвета;
- увеличиваем x на 1;
- если $F < 0$, то добавляем к F величину 2H; в противном случае увеличиваем y на 1 и добавляем к F величину -2(W - H) = 2(H - W).

Все вышеизложенное и определяет алгоритм Брезенхема для данного частного случая. Поскольку в этом алгоритме предусмотрена повторная инициализация для каждой новой прямой, то он является повторяемым: повторное рисование какой-либо прямой другим цветом полностью замещает первую прямую: например, рисование цветом фона целиком удаляет ранее нарисованную прямую.

Алгоритм Брезенхема предельно прост, в его внутреннем цикле содержатся только несколько операций сравнения и сложения. Для достижения максимальной скорости его легко реализовать на языке ассемблера. Этот алгоритм часто используется аппаратно в графических устройствах специального назначения, где обеспечивается еще большая его скорость.

4. Задание

Необходимо самостоятельно согласно алгоритму Брезенхема рассчитать координаты пикселей, закрашивая которые, можно построить прямую через заданные точки. Визуализацию попиксельной закраски следует выполнять с использованием инструментов OpenJSCAD. В качестве исходной информации задается две точки: [p1x, p1y], [p2x, p2y].

Необходимо рассчитать:

- координаты для построения прямой по алгоритму Брезенхема.

Необходимо построить:

- для каждой расчетной точки построить квадрат со стороной 0.6 (центр каждого квадрата сместить по обеим осям на +0.5).

Варианты заданий:

№ варианта	Координаты точек
1	[3, 5] [15, 9]
2	[-3, -2] [9, 3]
3	[-3, -3] [10, 1]
4	[10, 0] [24, 5]
5	[-10, -6] [1, 0]
6	[15, -1] [27, 4]
7	[0, 10] [14, 14]
8	[23, 0] [35, 5]
9	[-10, 15] [6, 20]
10	[3, 5] [16, 10]
11	[30, 0] [43, -4]
12	[-10, 4] [3, 9]
13	[-16, 14] [-3, 18]
14	[20, 20] [33, 25]
15	[-2, -2] [12, 3]
16	[30, 10] [42, 14]
17	[14, -10] [28, -5]
18	[1, 0] [14, 4]
19	[0, 0] [13, 4]
20	[5, -5] [18, 0]

5. Методические рекомендации

Рассмотрим на примере расчет функции F и ее влияние на значения расчетных координат.

Пусть: a = (4, 1), b = (16, 4).

Тогда: W = 12, H = 3.

Поскольку наклон данной прямой линии равен 0.25, мы ожидаем, что значение y будет увеличиваться только примерно на каждом 4-ом шаге по x. Начальное значение функции F отрицательно F = 2H – W = 6 – 12 = (–6).

При каждом увеличении x мы делаем одно из двух: если значение функции F отрицательно, то добавляем 2H = 6; в противном случае вычитаем из нее 18 и увеличиваем y.

В результате получаем следующую последовательность значений:

X:	4	5	6	7	8	9	10	11	12	13	14	15	16
Y:	1	1	2	2	2	2	3	3	3	3	4	4	4
F:	-6	0	-18	-12	-6	0	-18	-12	-6	0	-18	-12	-6

Построение квадратов позволит визуализировать процесс заливки пикселей. Для построения квадратов рекомендуется использовать следующие инструменты OpenJSCAD:

- CAG.rectangle({ center: [...], radius: 0.6})

Функция rectangle позволяет построить квадрат с заданным центром и длиной стороны. Таким образом, можно увидеть, как выглядит прямая, которую мы будем строить «попиксельно».

Лабораторная работа № 5

ПОСТРОЕНИЕ КРИВОЙ БЕЗЬЕ

1. Цель

Целью данной работы является наглядная демонстрация процесса аппроксимации при построении кривых Безье при разном количестве точек и одинаковых точках начала и конца.

2. Приобретаемые знания и навыки

- моделирование трехмерных объектов в среде OpenJSCAD на основе ломаных и кривых;
- построение кривых Безье при различном количестве исходных точек.

3. Теоретическая часть

Кривая Безье строится по принципу аппроксимации. Начнем с простого и элегантного алгоритма де Кастельо, с помощью которого создаются кривые Безье. В этом алгоритме для вычисления вполне определенного значения точки P(t) для каждого значения t от 0 до 1 используется последовательность точек P_0, P_1…. Таким образом, это дает возможность сгенерировать кривую по совокупности точек. Изменение этих точек влечет за собой и изменение кривой. Такое конструирование основывается на шагах «твининга», реализация которых достаточна проста.

Аффинная комбинация точек, выраженная уравнением:

$P = A(1-t) + Bt$, (1)

выполняет линейную интерполяцию между точками A и B.

Точка P(t), которая является t-ой частью на пути по прямой линии от A до B в момент t. Каждый компонент результирующей точки формируется как линейная интерполяция соответствующих компонентов точек A и B.

$P(t) = a + (b - a)*t$, (2)

Возможно создание интересной анимации, которая показывает, как одна фигура совершает «твин-преобразование» в другую. Начнем с трех точек: P_0, P_1, P_2, как показано на рис. 1. Выберем некоторое значение параметра t в интервале от 0 до 1, например, t = 0,3, после чего определим местонахождение точки A, находящейся на t-ой части пути вдоль прямой, соединяющей точки P_0 и P_1. Подобным образом определим точку B на t-ой части пути между конечными точками P_1, P_2 (при том же самом t). Мы знаем, что эти новые точки определяются выражениями:

$A(t) = (1 - t)\ P_0 + tP_1$

$B(t) = (1 - t)\ P_1 + tP_2$, (3)

Повторим для вновь полученных точек процесс линейной интерполяции (вновь при том же значении t): найдем точку P(t), лежащую на t-ой части между точками A и B.

$P(t) = (1 - t)A + tB$, (4)

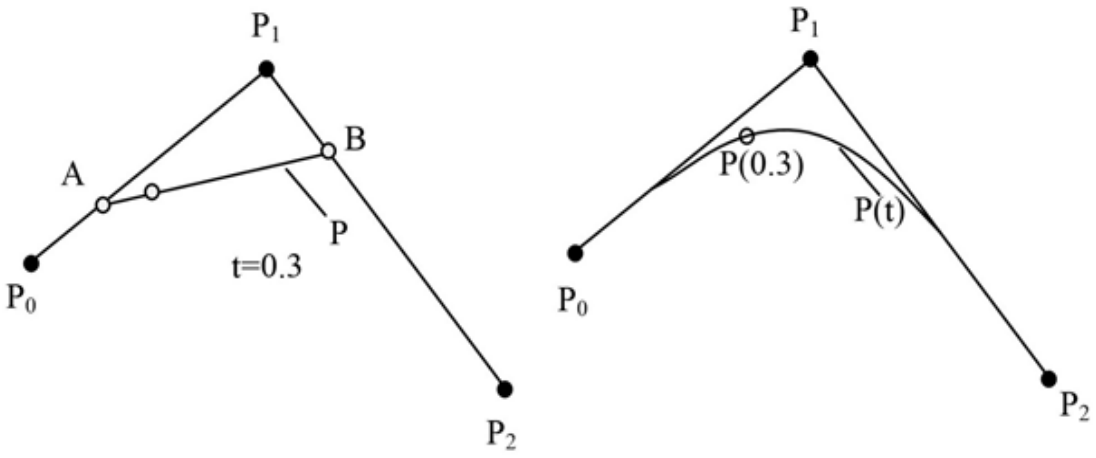


Рис. 1. Алгоритм де Кастельо для трех точек

Если выполнить этот процесс для каждого t в диапазоне от 0 до 1, то будет сформирована кривая P(t), приведенная на рис. 1(б). Какова параметрическая форма этой кривой? После непосредственной подстановки уравнения (1) в уравнение (2) получим:

$P(t) = (1 - t)^2P_0 + 2t(1 - t)P_1 + t^2P_2$, (5)

Параметрическая форма кривой P(t) является квадратичной относительно t, поэтому данная кривая является параболой. Она останется параболой, даже если t будет изменяться от минус бесконечности до плюс бесконечности. Очевидно, что эта кривая проходит через точку P_0 при t = 0 и через P_2 при t = 1. Таким образом, мы имеем корректно определенный процесс для генерирования плавной параболической кривой на базе трех заданных точек.

Чаще всего применяют семейство кривых Безье на базе четырех контрольных точек. Четырехточечная кривая Безье имеет следующую параметрическую форму:

$P(t) = P_0 (1 - t)^3 + P_13(1 - t)^2t + P_23(1 - t)t^2+P_3t^3$, (6)

Что является кубическим полиномом относительно t. Каждой контрольной точке Pi в этом кубическом полиноме придается определенный вес, после чего эти взвешенные точки складываются.

4. Задание

Визуализация построения кривых Безье по точкам осуществляется с использованием инструментов OpenJSCAD. В качестве исходной информации задается пять точек: [p1x, p1y], [p2x, p2y], [p3x, p3y], [p4x, p4y], [p5x, p5y]. Необходимо построить:

- 5 сфер с радиусом 3, разрешением 32 (цвет – по умолчанию) с центрами в указанных точках;
- ломаную линию по пяти указанным точкам (визуализировать инструментом rectangularExtrude с параметрами: ширина = 2, высота = 2, разрешение = 32);
- кривую Безье синего цвета, построенную по двум точкам: [p1x, p1y], [p5x, p5y] (визуализировать инструментом rectangularExtrude с параметрами: ширина = 2, высота = 5, разрешение = 32);

- кривую Безье зеленого цвета, построенную по трем точкам: [p1x, p1y], [p3x, p3y], [p5x, p5y] (визуализировать инструментом rectangularExtrude с параметрами: ширина = 2, высота = 5, разрешение = 32);
- кривую Безье оранжевого цвета, построенную по четырем точкам: [p1x, p1y], [p2x, p2y], [p4x, p4y], [p5x, p5y] (визуализировать инструментом rectangularExtrude с параметрами: ширина = 2, высота = 5, разрешение = 32);
- кривую Безье черного цвета, построенную по пяти точкам: [p1x, p1y], [p2x, p2y], [p3x, p3y], [p4x, p4y], [p5x, p5y] (визуализировать инструментом rectangularExtrude с параметрами: ширина = 2, высота = 5, разрешение = 32).

Варианты заданий:

№ варианта	Координаты точек
1	[100, 100] [-75, 25] [-75, 50] [0, 100] [-100, 100]
2	[100, 100] [-25, 25] [-45, 50] [0, 150] [-100, 100]
3	[75, 75] [25, 25] [0, 125] [-50, 25] [-75, 75]
4	[80, 80] [-60, 25] [-20, 100] [-50, 25] [-95, 55]
5	[-80, -80] [50, 25] [0, 80] [-50, 25] [-95, 55]
6	[80, 80] [60, 44] [0, 80] [-50, 0] [78, 50]
7	[50, 50] [88, 98] [0, 80] [-50, 0] [78, -50]
8	[40, 40] [88, 98] [0, 80] [-90, 0] [-100, 100]
9	[60, 60] [30, 98] [35, 0] [-90, 0] [78, -50]
10	[60, 60] [-40, 95] [35, 50] [-80, 0] [-8, 0]
11	[80, 80] [20, 80] [68, 22] [-32, 30] [-100, 100]
12	[-100, -100] [20, -80] [65, 18] [-33, 35] [-100, 100]
13	[-100, -100] [100, -100] [100, 100] [-100, 100] [0, 0]
14	[-77, -77] [-77, 77] [77, 77] [77, -77] [0, 0]
15	[-90, -90] [-77, 77] [77, 77] [77, -77] [-90, -90]
16	[-90, -90] [40, 100] [-50, -50] [100, 40] [-90, -90]
17	[-90, -90] [30, 90] [0, 0] [90, 30] [-90, -90]
18	[100, 100] [0, 70] [0, -50] [90, 30] [-80, 0]
19	[100, 100] [0, 70] [80, -10] [90, 30] [-80, 0]
20	[100, 100] [-100, 100] [-100, 50] [0, 50] [0, -50]

5. Методические рекомендации

Построение сфер позволит визуализировать исходные точки. Чертеж будет нагляднее. Для построения сфер рекомендуется использовать следующие инструменты OpenJSCAD:

- CSG.sphere({center: [...], radius: 2, resolution: 32});

Для построения ломаных и кривых линий рекомендуется использовать следующие инструменты OpenJSCAD:

- appendPoints([[x,y],...]);
- appendBezier([[x,y],...], options).

Функция appendPoints позволяет построить ломаную по точкам. А функция rectangularExtrude – визуализировать ее. Таким образом, можно увидеть, как выглядит ломаная, которую мы будем стремиться аппроксимировать по алгоритму Безье.

Функция appendBezier по заданным точкам строит кривую согласно алгоритму Безье. В ходе выполнения работы можно наглядно увидеть, как меняется форма кривой в зависимости от количества и состава точек.

Лабораторная работа № 6

ПОСТРОЕНИЕ B-СПЛАВНОВОЙ КРИВОЙ

1. Цель

Целью данной работы является самостоятельный расчет координат точек для построения B-сплайна и визуализация результата средствами OpenJSCAD.

2. Приобретаемые знания и навыки

- моделирование трехмерных объектов в среде OpenJSCAD на основе ломаных и кривых;
- расчет координат и построение кривых с помощью B-сплайнов.

3. Теоретическая часть

У кривых Безье каждая точка Pi оказывает влияние на кривую для всех значений t от 0 до 1. Желателен же локальный контроль. Решение – множество стыковочных функций, имеющих поддержку только на части интервала [0,1].

Данные функции должны:

- быть простыми в вычислении;
- Sf(t) = 1 для любого t из [a,b];
- иметь поддержку лишь на небольшом участке интервала [a,b] для обеспечения локального контроля;
- интерполировать определенные контрольные точки;
- быть достаточно гладкими (f(t) – гладкие внутри интервала, начинаются и заканчиваются с нулевым значением производной).

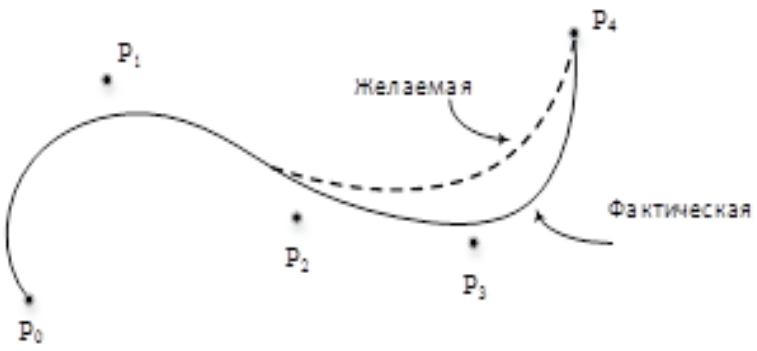


Рис.1. Локальный контроль

Кусочно-полиномиальные кривые и сплайны

Поиск кандидатов на роль стыковочных функций.

У кубической формы (вида $R(t) = at^3 + bt^2 + ct + d$) недостаточно гибкости, чтобы «изогнуться» так, как этого требуется. Выход – кусочные полиномы (несколько полиномов низкого порядка, различные на различных интервалах t).

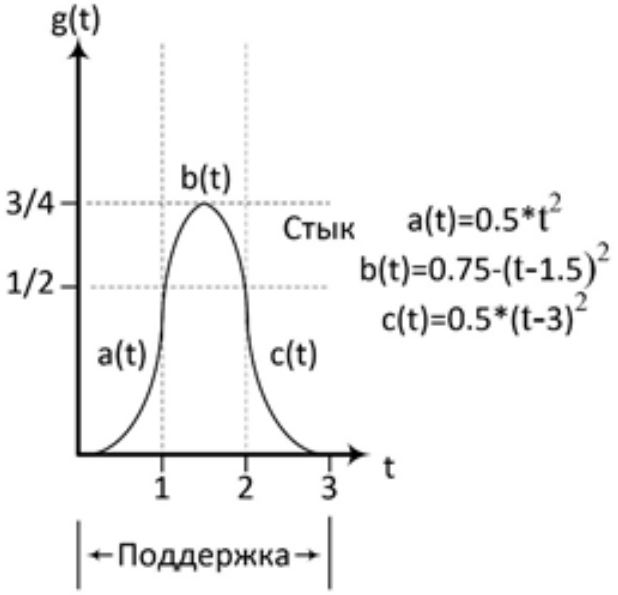


Рис. 2. Примерная форма кусочных полиномов g(t)

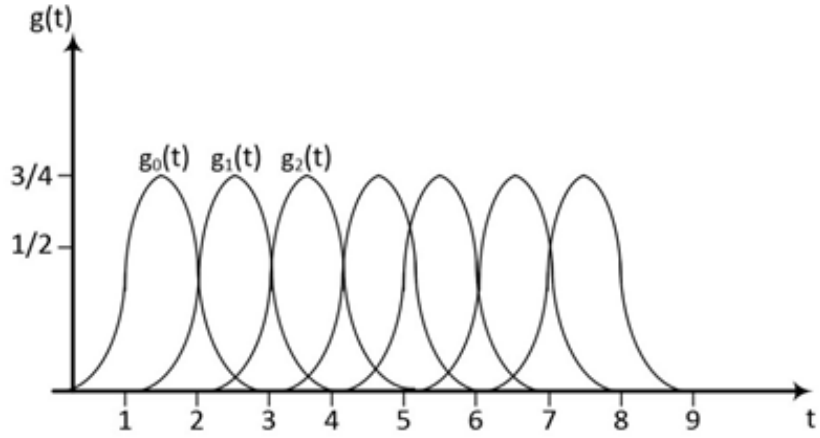
- Поддержка кривой g(t): [0,3], функции определены в диапазонах;
- Точки, в которых встречается пара отдельных сегментов – стыки;
- Значения t в стыках – узлы

Сплайн-функция M-ой степени – это кусочно-полиномиальная функция степени M, обладающая в каждом узле (M-1) гладкостью. g(t) – пример сплайн-функции – кусочно-полиномиальной функции, обладающей достаточной гладкостью. g(t) – квадратичный сплайн (непрерывная всюду первая производная).

Построение из g(t) множества стыковочных функций

$g_k(t) = g(t-k)$, где $k = 0, 1, \ldots$

Сумма gk(t) равна единице (тут выполняется для t в пределах от 2 до 7).



Задание

ЭТОТ ЭЛЕМЕНТ КУРСА ОЦЕНИВАЕТСЯ КАК LAB ВЕС: 1.0

Добавить в закладки

4. Задание

Целью данной работы является самостоятельный расчет координат точек для построения B-сплайна. Визуализация построения осуществляется с использованием инструментов OpenJSCAD. В качестве исходной информации задается трехточечный полигон: $P(t) = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}$,

где t=[1, 5] и значения t – интервал, на котором необходимо построить B-сплайн: t = [t_0, …, t_k] с фиксированным шагом Dt.

Необходимо рассчитать построить B-сплайн g(t) кривой 2-ого порядка по пяти точкам в соответствии с заданными значениями t и визуализировать его с помощью инструментов OpenJSCAD. А именно нужно:

- самостоятельно рассчитать координаты пяти точек для построения B-сплайна;
- построить ломаную линию по трем исходным точкам (визуализировать инструментом rectangularExtrude с параметрами: ширина = 0.05, высота = 0.05, разрешение = 32);
- построить ломаную линию по пяти расчетным точкам (визуализировать инструментом rectangularExtrude с параметрами: ширина = 0.05, высота = 0.05, разрешение = 32).

Варианты заданий:

5. Методические рекомендации и пример расчета

Для расчета координат точек определим исходные значения:

Заданный полигон: $P(t) = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}$;

Промежуток t: t = [t_0, …, t_k];

Порядок кривой: m = 2;

L = число точек в полигоне -1 = 2;

Число стыковочных функций k = 0, 1…L, k = 0, 1, 2.

B-сплайн строится как результат взвешенного суммирования стыковочных функций $g_k(t)$, где $g_k(t)=g(t-k)$.

$P_x(t) = \sum^{L}_{i=0} P_{xi} * g_i(t)$

$P_y(t) = \sum^{L}_{i=0} P_{yi} * g_i(t)$

Сама стыковочная функция поддерживается на промежутке t=[0,3], состоит из трех полиномиальных сегментов. Значение ее в каждый момент времени t определяется следующим образом:

Полиномиальные сегменты – кривые второго порядка, задаются уравнениями:

$$g_k(t) = \begin{cases} a(t-k), \text{если } 0 \le t-k < 1, \\ b(t-k), \text{если } 1 \le t-k < 2, \\ c(t-k), \text{если } 2 \le t-k < 3, \\ 0, \quad \text{в противном случае} \end{cases}$$

Расчет координат точек для заданного промежутка t проводится следующим образом:

$$a(t) = \frac{1}{2}t^2 \;\; b(t) = \frac{3}{4} - (t-\frac{3}{2})^2 \; c(t) = \frac{1}{2}(t-3)^2$$

- получить выражения для расчета стыковочных функций $g_0(t)$, $g_1(t)$, $g_2(t)$;
- на промежутке t = [t0, …, tk] с шагом Dt вычислить значение $g_0(t)$, $g_1(t)$, $g_2(t)$;
- на промежутке t = [t0, …, tk] с шагом Dt взвешенным суммированием получить значение P_x и P_y;
- визуализировать полученную кривую средствами OpenJSCAD.

Например:

Заданный полигон: $P(t) = \begin{bmatrix} 2 & 0 & -4 \\ 0 & -2 & 0 \end{bmatrix}$

Промежуток t: t = [2, 2.25, 2.5, 2.75, 3];

Порядок кривой: m = 2;

L = 2;

Число стыковочных функций k = 0, 1, 2.

Стыковочные функции имеют вид:

$$g_0(t) = \begin{cases} a(t), \text{если } 0 \le t < 1, \\ b(t), \text{если } 1 \le t < 2, \\ c(t), \text{если } 2 \le t < 3, \\ 0, \text{в противном случае} \end{cases}$$

$$g_1(t) = \begin{cases} a(t-1), \text{если } 1 \le t < 2, \\ b(t-1), \text{если } 2 \le t < 3, \\ c(t-1), \text{если } 3 \le t < 4, \\ 0, \text{в противном случае} \end{cases}$$

$$g_2(t) = \begin{cases} a(t-2), \text{если } 2 \le t-k < 3, \\ b(t-2), \text{если } 3 \le t-k < 4, \\ c(t-2), \text{если } 4 \le t-k < 5, \\ 0, \text{в противном случае} \end{cases}$$

Дальнейшие расчеты представлены в таблице 1.

Таблица 1.

t	$g_0(t)$	$g_1(t)$	$g_2(t)$	$g_0(t)$	$g_1(t)$	$g_2(t)$	Px(t)	$P_y(t)$
2	c(t)	b(t-1)	a(t-2)	0,5	0,5	0	1	-1
2.25	c(t)	b(t-1)	a(t-2)	0,281	0,699	0,031	0,4375	-1,375
2.5	c(t)	b(t-1)	a(t-2)	0,125	0,75	0,125	-0,25	-1,5
2.75	c(t)	b(t-1)	a(t-2)	0,031	0,69	0,281	-1,06	-1,375
3	c(t)	b(t-1)	a(t-2)	0	0,5	0,5	-2	-1

После получения координат переходим к работе в OpenJSCAD. Для построения ломаных линий рекомендуется использовать следующие инструменты OpenJSCAD:

- appendPoints([[x,y], ...]);

Функция appendPoints позволяет построить ломаную по точкам. А функция rectangularExtrude – визуализировать ее. Таким образом, можно увидеть, как выглядит ломаная, которую мы будем стремиться аппроксимировать и сам расчетный B-сплайн.

Лабораторная работа № 7

ПОСТРОЕНИЕ NURBS КРИВОЙ

1. Цель

Целью данной работы является расчет и построение NURBS-кривых, изучение влияния веса точки на вид кривой.

2. Приобретаемые знания и навыки

- моделирование трехмерных объектов в среде OpenJSCAD на основе ломаных и кривых;
- построение NURBS-кривых на основе базисных функций третьего порядка при неравномерном узловом векторе;
- рассмотрение влияния веса точки на вид результирующей кривой.

3. Теоретическая часть

Термин «NURBS» является аббревиатурой и расшифровывается как Non-Uniform Rational B-spline, где:

- «Non-Uniform» (неоднородный) означает, что область влияния контрольной точки на форму кривой может быть различной;

• «Rational» (рациональный) означает, что математическое выражение, описывающее форму моделируемой кривой, есть отношение двух полиномов. Эта особенность позволяет точнее моделировать различные кривые, например, конические сечения;

• «B-spline» (basis spline, базовый сплайн) – способ математического описания кривой интерполяцией между тремя и более контрольными точками;

NURBS кривая выражается следующим параметрическим уравнением:

где P_i – управляющая точка, w_i – ассоциированный с ней вес и $N_{i,\ m}$ – базовая функция порядка m (m – порядок (order), p – степень (degree), m = p + 1), определенная рекурсивно следующим образом ($k = i, ..., m$):

$$N_{i,1}(t)=\begin{cases}1, t_i \le t < t_{i+1}\\ 0, иначе\end{cases}$$

$$\forall k>1, N_{i,k}(t)=\frac{t-t_i}{t_{i+k}-t_i}N_{i,k-1}(t)+\frac{t_{i+k+1}-t}{t_{i+k+1}-t_{i+1}}N_{i+1,k-1}(t)$$

Из формул видно, что точка кривой является средневзвешенной управляющей точкой, причем удельный вес каждой точки зависит от одного параметра.

Сплайны Безье – это NURBS, у которого веса всех управляющих точек равны 1 и который состоит из 1-го сплайнового сегмента. Таким образом, NURBS имеет все преимущества Безье-сплайнов, а также следующие:

• возможность локального управления кривизной сплайна;

• наличие весов для управляющих точек, делающих сплайны еще более гибкими.

Единственный недостаток – это несколько более сложное математическое описание NURBS по сравнению с Безье сплайнами, однако порядок этого усложнения не высок.

Обратимся к ключевой букве в названии NURBS — «R», что означает «rational» (рациональный). Рациональные кривые, в сравнении с обычными (нерациональными – «non-rational») B-сплайнами, обладают двумя дополнительными и очень важными свойствами:

• они обеспечивают корректный результат при проекционных трансформациях (например, масштабировании), а нерациональные B-сплайны – только при аффинных трансформациях (например, перемещениях);

• их можно использовать для моделирования кривых любого вида, включая конические сечения (окружности, эллипсы, параболы и гиперболы).

Эти свойства достигаются за счет четырехмерного представления обычной трехмерной контрольной точки {x, y, z}. Это значит, что каждая контрольная точка представляется четырьмя координатами {x, y, z, w}. Последняя координата w означает вес (weight) контрольной точки.

Изначально координата w равняется единице, но при увеличении этого значения для контрольной точки увеличивается степень ее воздействия на форму кривой и последняя сильнее выгибается в сторону контрольной точки (рис. 1).

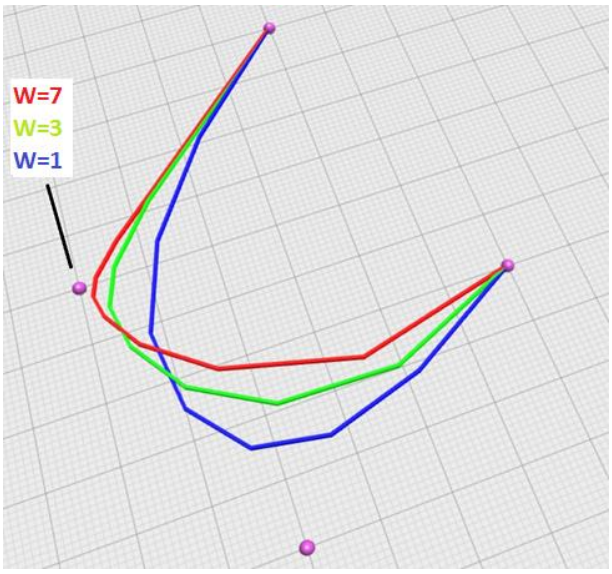


Рис. 1. Изменение формы кривой при изменении веса контрольной точки

Необходимо отметить, что существенным является только относительное изменение весов контрольных точек. Если вдвое увеличить веса всех контрольных точек, то форма кривой не изменится.

4. Задание

Необходимо рассчитать и построить NURBS-кривые в соответствии с заданными точками, узловым вектором и весами. Для расчета точек NURBS-кривых использовать таблицы значений базовых функций третьего порядка.

Визуализация построения NURBS-кривых точкам осуществляется с использованием инструментов OpenJSCAD. В качестве исходной информации задается четыре точки: [p1x, p1y], [p2x, p2y], [p3x, p3y], [p4x, p4y] и два варианта веса для каждой из них [$w_1$1, $w_1$2, $w_1$3, $w_1$4] и [$w_2$1, $w_2$2, $w_2$3, $w_2$4]. Для каждого варианта остаются неизменны значения. Необходимо рассчитать:

- координаты NURBS-кривых (9 точек) для каждого вектора весов с использованием таблицы значений базисных функций третьего порядка;
- округлить значения для построения по следующему принципу: округляем до НАИМЕНЬШЕГО числа с тремя знаками после запятой, БОЛЬШЕГО или РАВНОГО аргументу. Так дробь 2.3703649952833614 после округления будет равна не 2.37, а 2.371.

И построить:

- 4 сферы с радиусом 1, разрешением 32 (цвет – по умолчанию) с центрами в указанных точках;
- ломаную линию по четырем указанным точкам (визуализировать инструментом rectangularExtrude с параметрами: ширина = 1, высота = 1, разрешение = 32);

• ломаную линию синего цвета, построенную по расчетным точкам для веса w1 (визуализировать инструментом rectangularExtrude с параметрами: ширина = 1, высота = 1, разрешение = 32);

• ломаную линию красного цвета, построенную по расчетным точкам для веса w2 (визуализировать инструментом rectangularExtrude с параметрами: ширина = 1, высота = 1, разрешение = 32).

Варианты заданий:

№ варианта	Координаты точек	Векторы веса
1	$P=\begin{bmatrix}10 & 30 & 60 & 20\\ -20 & 30 & -20 & -60\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 0.5 2 1]
2	$P=\begin{bmatrix}-70 & 30 & 40 & -20\\ 0 & 10 & 50 & 80\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 2 0.2 1]
3	$P=\begin{bmatrix}40 & 0 & -10 & -50\\ 0 & 10 & 50 & 0\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 2 0.5 1]
4	$P=\begin{bmatrix}40 & 60 & 0 & 50\\ 10 & 0 & -20 & 30\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 0.5 2 1]
5	$P=\begin{bmatrix}140 & 60 & 50 & -20\\ -20 & 20 & -20 & 20\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 1.5 0.3 1]
6	$P=\begin{bmatrix}100 & -60 & -50 & 20\\ 30 & -30 & 30 & 10\end{bmatrix}$	w1 = [2 2 2 2] w2 = [1 1.5 0.8 1]
7	$P=\begin{bmatrix}40 & -60 & 40 & -40\\ -50 & -30 & 50 & 10\end{bmatrix}$	w1 = [2 2 2 2] w2 = [2 1 4 2]
8	$P=\begin{bmatrix}0 & -30 & -20 & 70\\ 20 & 0 & -50 & -60\end{bmatrix}$	w1 = [2 2 2 2] w2 = [2 0.5 3 2]
9	$P=\begin{bmatrix}80 & -30 & -20 & 50\\ -20 & 50 & 0 & -80\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 0.5 3 1]
10	$P=\begin{bmatrix}-50 & 80 & 10 & -70\\ 100 & -20 & 20 & 40\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 0.5 5 1]
11	$P=\begin{bmatrix}-50 & -80 & 10 & 0\\ 30 & -20 & 0 & 40\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 5 0.2 1]
12	$P=\begin{bmatrix}50 & 80 & 0 & 30\\ -20 & -80 & -50 & 20\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 4 0.3 1]
13	$P=\begin{bmatrix}40 & 70 & -10 & 0\\ -20 & 0 & 80 & 10\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 3 0.7 1]
14	$P=\begin{bmatrix}-40 & 0 & 10 & 10\\ 10 & 0 & 10 & -10\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 0.2 1.8 1]
15	$P=\begin{bmatrix}-40 & 0 & 10 & 10\\ 10 & 0 & 10 & -10\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 0.3 1.7 1]
16	$P=\begin{bmatrix}-40 & 0 & 10 & 10\\ 10 & 0 & 10 & -10\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 2 0.5 1]
17	$P=\begin{bmatrix}-40 & 40 & 60 & -10\\ 60 & 10 & -30 & 10\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 0.1 0.1 1]

18	$P=\begin{bmatrix}10 & 30 & 50 & -20\\ 40 & 50 & -10 & -40\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 2 0.1 1]
19	$P=\begin{bmatrix}10 & -20 & 10 & 0\\ 0 & 50 & 50 & 10\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 5 5 1]
20	$P=\begin{bmatrix}-10 & 0 & -50 & -20\\ -40 & 50 & 30 & -40\end{bmatrix}$	w1 = [1 1 1 1] w2 = [1 0.5 2 1]

5. Методические рекомендации

Для расчета координат точек определим исходные значения:

Заданный полигон: $P(t) = \begin{bmatrix}x_1 & x_2 & x_3 & x_4\\ y_1 & y_2 & y_3 & y_4\end{bmatrix}$

Узловой вектор неравномерный: t = [0, 0, 0, 1, 2, 2, 2];

Порядок базисных функций: m = 3;

L = число точек в полигоне -1 = 3;

Длина требуемого узлового вектора: L+m+1 = 7

Число стыковочных функций k = 0, 1…L, k = 0,1,2,3;

Промежуток t = [0..2], Δt = 0.25.

NURBS-кривая строится как результат взвешенного суммирования стыковочных функций Nk(t):

$$P(t)=\frac{\sum_{i=0}^{L} N_{i,m}(t)P_i w_i}{\sum_{i=0}^{L} N_{i,m}(t)w_i}$$

Сама стыковочная функция поддерживается на промежутке t = [0,2], значение ее в каждый момент времени t определяется следующим образом: $N_{i,m}$ – базовая функция порядка m (m – порядок (order), p – степень (degree), m = p + 1), определенная рекурсивно следующим образом (k = 1, ..., m):

$$N_{i,1}(t)=\begin{cases}1, t_i \le t < t_{i+1}\\ 0, иначе\end{cases}$$

$$\forall k>1, N_{i,k}(t)=\frac{t-t_i}{t_{i+k}-t_i}N_{i,k-1}(t)+\frac{t_{i+k+1}-t}{t_{i+k+1}-t_{i+1}}N_{i+1,k-1}(t)$$

Для выполнения лабораторной работы значения базисных функций представлены в таблице.

t	N0(t)	N1(t)	N2(t)	N3(t)	$\sum N_{i,m}$
0	1	0	0	0	1
0.25	0.563	0.406	0.031	0	1
0.5	0.25	0.625	0.125	0	1
0.75	0.063	0.656	0.281	0	1
1	0	0.5	0.5	0	1

1.25	0	0.281	0.656	0.063	1
1.5	0	0.125	0.625	0.25	1
1.75	0	0.031	0.406	0.563	1
2	0	0	0	1	1

Расчет координат точек для заданного промежутка t проводится следующим образом:

- на промежутке t = [t0, …, tk] с шагом t взвешенным суммированием для веса w1 получить значение Px и Py;
- на промежутке t = [t0, …, tk] с шагом t взвешенным суммированием для веса w2 получить значение Px и Py;
- округлить значения для построения по следующему принципу: округляем до НАИМЕНЬШЕГО числа с тремя знаками после запятой, БОЛЬШЕГО или РАВНОГО аргументу. Так дробь 2.3703649952833614 после округления будет равна не 2.37, а 2.371.
- визуализировать исходные данные и полученные ломаные линии средствами OpenJSCAD.

Построение сфер позволит показать исходные точки. Чертеж будет нагляднее. Для построения сфер рекомендуется использовать следующие инструменты OpenJSCAD:

- CSG.sphere({center: [...], radius: 1, resolution: 32}).

Для построения ломаных линий рекомендуется использовать следующие инструменты OpenJSCAD:

- appendPoints([[x,y],...]).

Функция appendPoints позволяет построить ломаную по точкам. А функция rectangularExtrude – показать ее. Таким образом, можно увидеть, как выглядит ломаная, которую мы будем стремиться аппроксимировать по с помощью NURBS-кривых, и визуализировать результат расчетов.

Лабораторная работа № 8

ТЕЛА ВРАЩЕНИЯ

1. Цель

Изучение инструментов среды OpenJSCAD, позволяющих моделировать тела вращения.

2. Приобретаемые знания и навыки

- моделирование тел вращения в среде OpenJSCAD на основе значений параметров.

3. Теоретическая часть

Тела вращения возникают при вращении 2D-объекта вокруг некоторой оси.

Прямой круговой цилиндр образуется вращением прямоугольника вокруг одной из его сторон.

Прямой круговой конус образуется вращением прямоугольного треугольника вокруг одного из его катетов. Другой катет при этом описывает основание конуса, а гипотенуза – боковую поверхность.

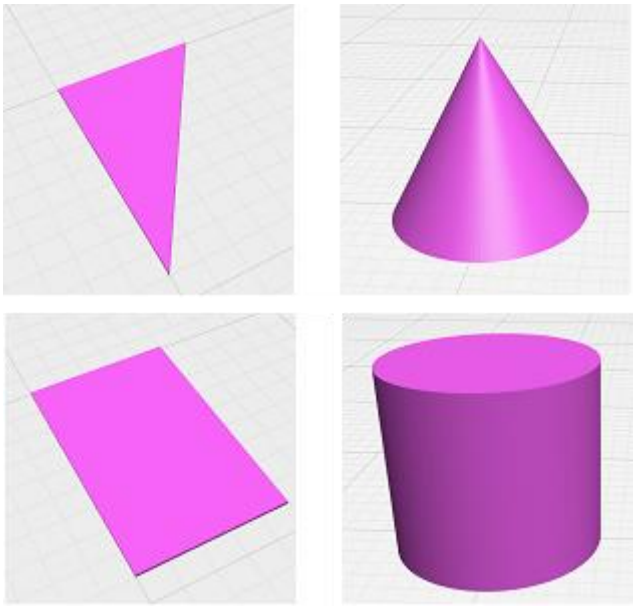


Рис. 1. Фигуры вращения – прямой круговой цилиндр, прямой круговой конус

Шар образуется вращением полукруга вокруг его диаметра.

Шаровой сектор образуется вращением кругового сегмента вокруг одного из его радиусов.

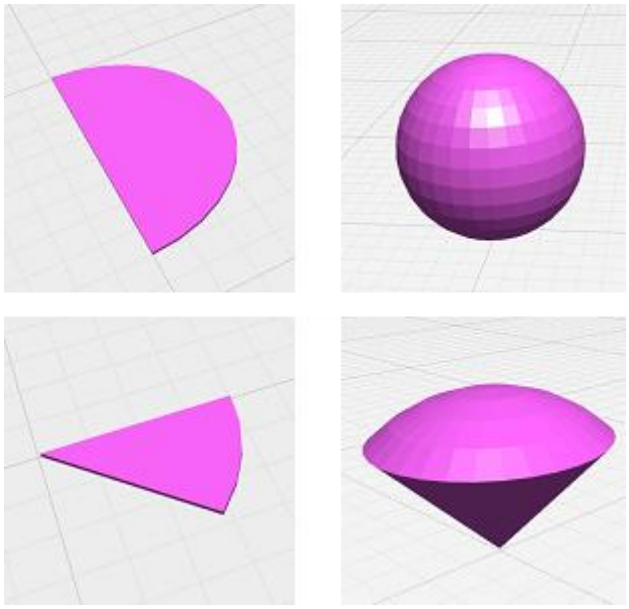


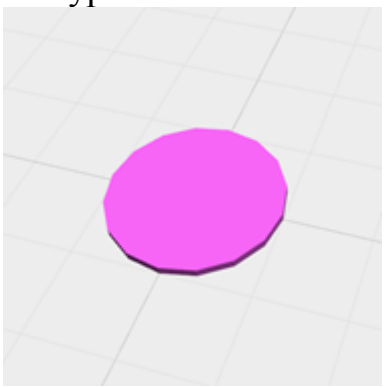
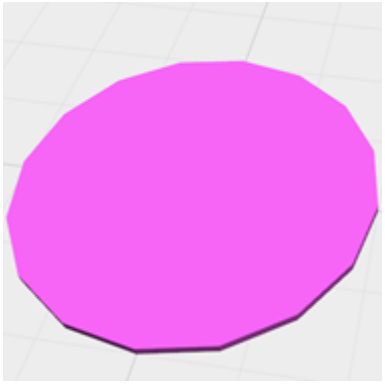
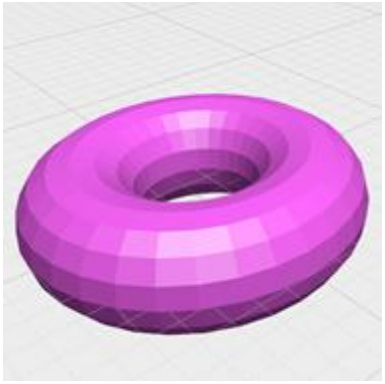
Рис. 2. Фигуры вращения – сфера, шаровой сектор

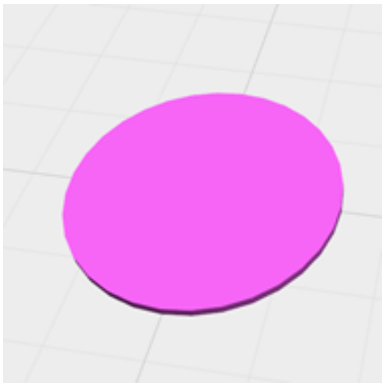
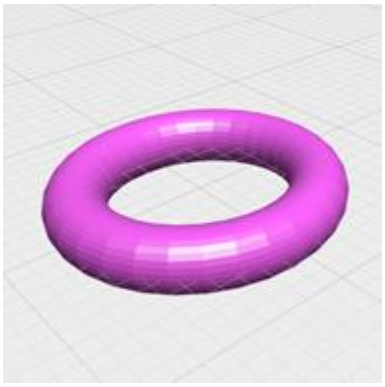
3.1. Примитив *torus*

Top образуется вращением окружности вокруг прямой, не пересекающей окружность и лежащей с ней в одной плоскости (рис. 3).

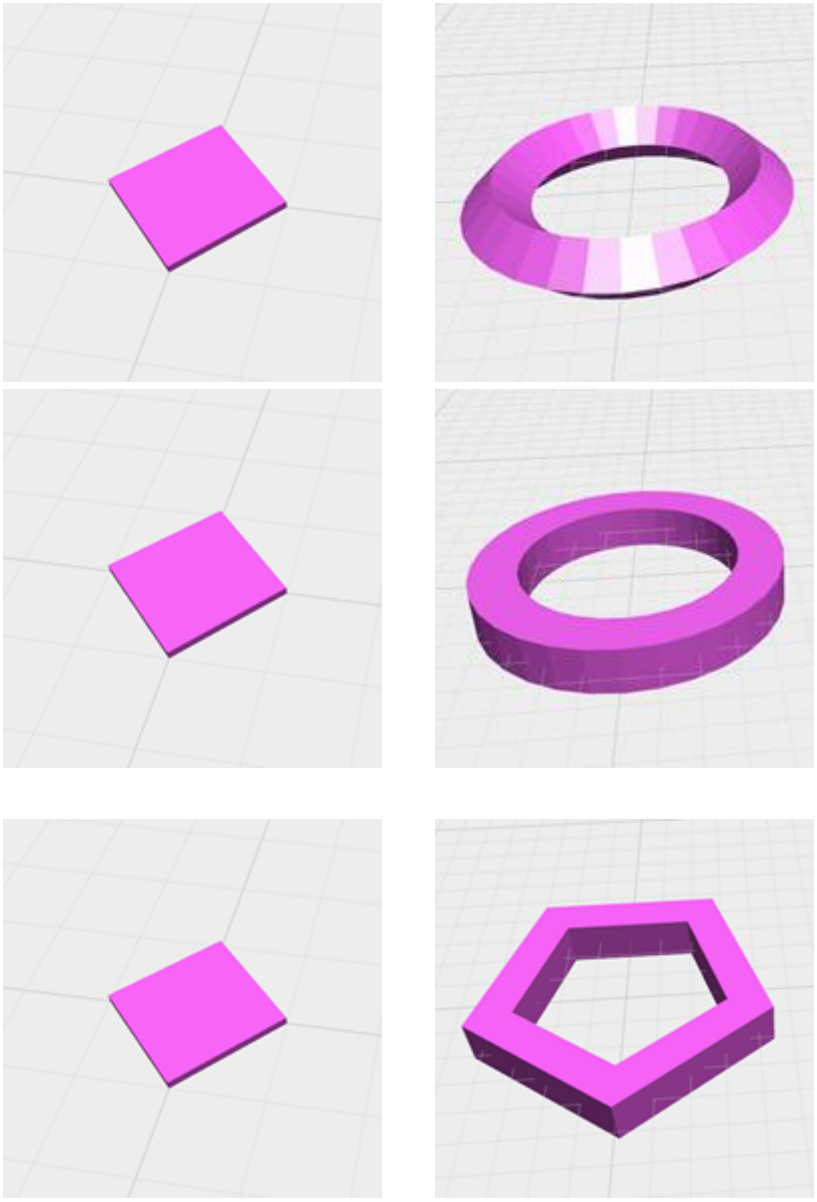
Основные параметры примитива ***torus***:

- *ri* = радиус плоской фигуры *A*, образующей тор (по умолчанию: 1),
- *ro* = расстояние от оси до центра фигуры *A* (по умолчанию: 4),
- *fni* = коэффициент сглаживания фигуры *A* (по умолчанию: 16),
- *fno* = коэффициент сглаживания тора (по умолчанию: 32),
- *roti* = угол поворота фигуры *A* (по умолчанию: 0)

Код	Фигура *A*	Тело вращения
/* ri = 1 ro = 4 fni = 16 fno = 32 roti = 0 */ torus()		
/* ri = 2 ro = 4 fni = 16 fno = 32 roti = 0 */ torus({ ri: 2 })		

/*
ri = 1.5
ro = 6
fni = 32
fno = 32
roti = 0 */
torus({
  ri: 1.5,
  ro: 6,
  fni: 32
})
		

```
/*
ri = 1
ro = 4
fni = 4
fno = 32
roti = 0 */
torus({
  fni: 4
})
/*
ri = 1
ro = 4
fni = 4
fno = 32
roti = 45 */
torus({
  fni: 4,
  roti: 45
})
/*
ri = 1
ro = 4
fni = 4
fno = 5
roti = 45 */
torus({
  fni:4,
  fno:5,
  roti: 45
})
```



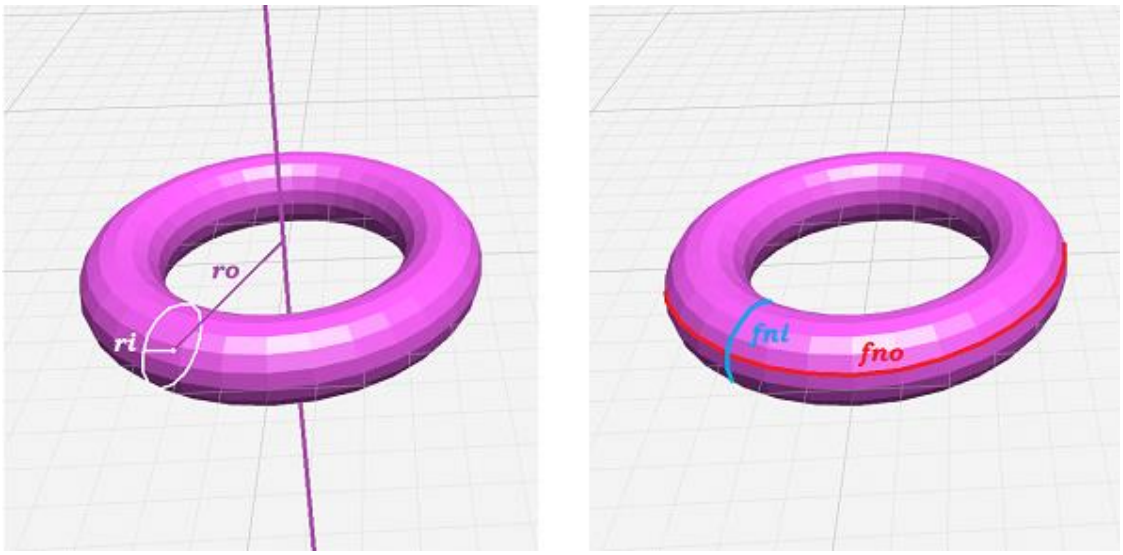


Рис. 3. Фигуры вращения – тор

Фигура *A* всегда является правильным выпуклым многоугольником, если точнее – примитивом *CAG.circle()*, чей коэффициент сглаживания *resolution* отвечает за количество углов (преломлений боковой поверхности примитива). Посмотрим, как изменится тело вращения при изменении параметров фигуры *A*.

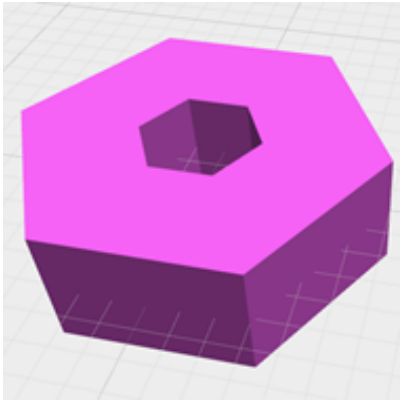
Видно, что траектория обращения двумерной фигуры *A* вокруг оси в трёхмерном пространстве также является правильным многоугольником *B*, число углов которого зависит от параметра *fno*.

3.2. Инструмент *rotate_extrude*

Данный инструмент позволяет вращать произвольный 2D-объект вокруг оси *Oz*, образуя таким образом трёхмерное тело вращения. Несмотря на то, что 2D-объект находится в плоскости *xOy*, в процессе преобразования он разворачивается так, чтобы ось вращения совпадала с *Oz*.

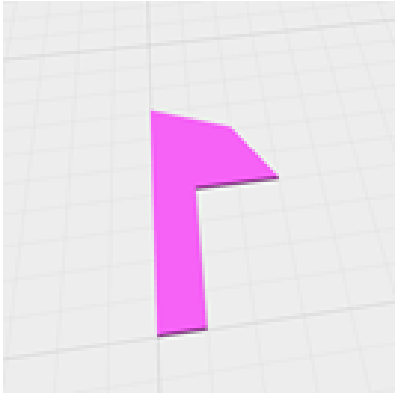
Рассмотрим пример. 2D-объект является примитивом *CAG.rectangle* размером 2x2. Траекторией вращения является правильный шестигранник (fn = 6), радиус вращения равен 4.

```
rotate_extrude(
    {fn:6},
    CAG.rectangle({
        radius: [2,2]
    }).translate([4,0,0])
);
```



Теперь зададим произвольный 2D-объект с помощью примитива *CAG.fromPoints* и набора точек.

```
CAG.fromPoints([
   [-7,0], [0,0], [0,1],
   [-4,1], [-4,3], [-6,2]
]);
```



```
rotate_extrude(
  CAG.fromPoints([
    [-7,4], [0,4], [0,5],
    [-4,5], [-4,7], [-6,6]
  ]).translate([0,-4,0]).rotateZ(-90)
);
```

4. Задание

Пусть правильный многоугольник *A* обращается вокруг начала координат в плоскости *xOy* по траектории, заданной многоугольником *B*. Внутренний угол многоугольника (выраженный в градусах) *A* равен a_1, периметр – p_1. Внутренний угол (выраженный в градусах) и периметр многоугольника *B* равны, соответственно, a_2 и p_2. Смоделируйте примитив *torus* на основе полученных данных.

Для задания численных значений параметров, выражаемых через тригонометрические функции, следует использовать переменную *Math.PI* и функцию *Math.sin*. В противном случае, высока вероятность ошибки округления. Параметр *roti* задавать не требуется, следует использовать значение по умолчанию. Необходимые для расчётов формулы находятся в методических указаниях к работе.

№	Задание
1	$a_1 = 144°$, $a_2 = 108°$, $p_1 = 5$, $p_2 = 40$;
2	$a_1 = 170°$, $a_2 = 90°$, $p_1 = 4.5$, $p_2 = 32$;
3	$a_1 = 60$, $a_2 = 150°$, $p_1 = 4.5$, $p_2 = 24$;
4	$a_1 = 140°$, $a_2 = 144°$ $p_1 = 4.5$, $p_2 = 20$;
5	$a_1 = 135°$, $a_2 = 160°$, $p_1 = 2.4$, $p_2 = 9$;
6	$a_1 = 120°$, $a_2 = 135°$, $p_1 = 3$, $p_2 = 16$;
7	$a_1 = 90°$, $a_2 = 144°$, $p_1 = 2$, $p_2 = 20$;
8	$a_1 = 150°$, $a_2 = 160°$, $p_1 = 7.5$, $p_2 = 13.5$;
9	$a_1 = 120°$, $a_2 = 170°$, $p_1 = 9$, $p_2 = 18$;
10	$a_1 = 150°$, $a_2 = 90°$, $p_1 = 3.6$, $p_2 = 12$;
11	$a_1 = 135°$, $a_2 = 90°$, $p_1 = 2.4$, $p_2 = 20$;
12	$a_1 = 108°$, $a_2 = 160°$, $p_1 = 2$, $p_2 = 18$;

13	$a_1 = 140°$, $a_2 = 120°$, $p_1 = 9$, $p_2 = 36$;
14	$a_1 = 140°$, $a_2 = 150°$, $p_1 = 3.6$, $p_2 = 8$;
15	$a_1 = 160°$, $a_2 = 135°$, $p_1 = 9$, $p_2 = 16$;
16	$a_1 = 144°$, $a_2 = 108°$, $p_1 = 10$, $p_2 = 20$;
17	$a_1 = 170°$, $a_2 = 90°$, $p_1 = 9$, $p_2 = 16$;
18	$a_1 = 60°$, $a_2 = 150°$, $p_1 = 9$, $p_2 = 12$;
19	$a_1 = 144°$, $a_2 = 60°$, $p_1 = 5$, $p_2 = 24$;
20	$a_1 = 120°$, $a_2 = 150°$, $p_1 = 3$, $p_2 = 16$.

5. Методические рекомендации

Правильный многоугольник – выпуклый многоугольник, у которого равны все стороны и все углы.

Внутренние углы в правильном многоугольнике равны между собой и определяются по формуле:

$a = 180° * (n - 2) / n$, где n – число углов (сторон) многоугольника.

Радиус описанной вокруг правильного многоугольника окружности (расстояние от центра до вершины):

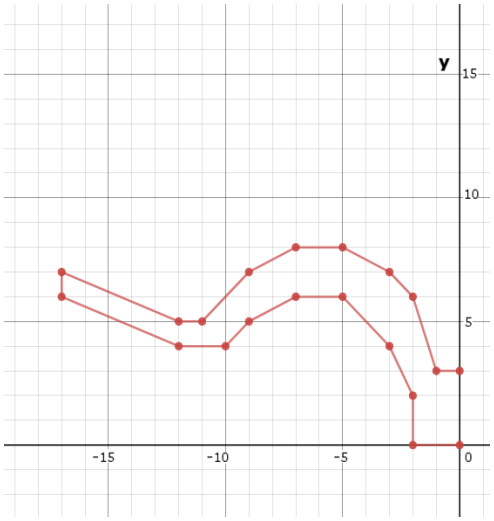
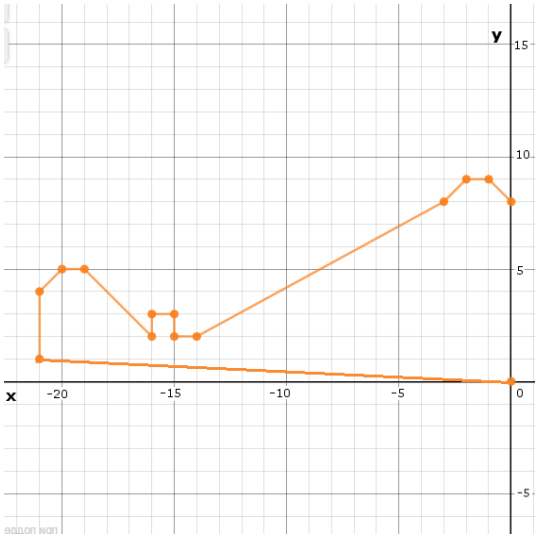
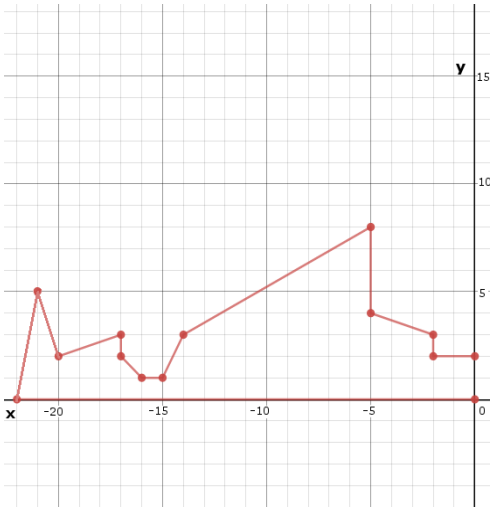
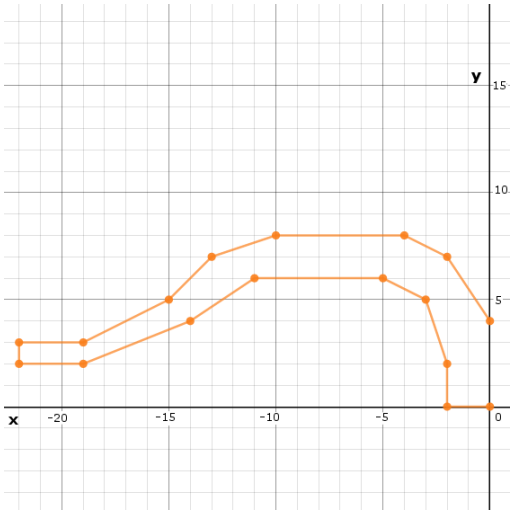
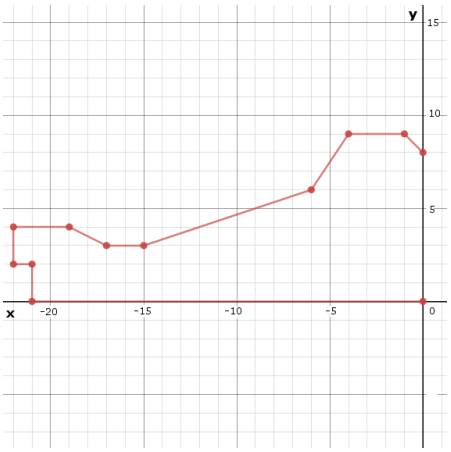
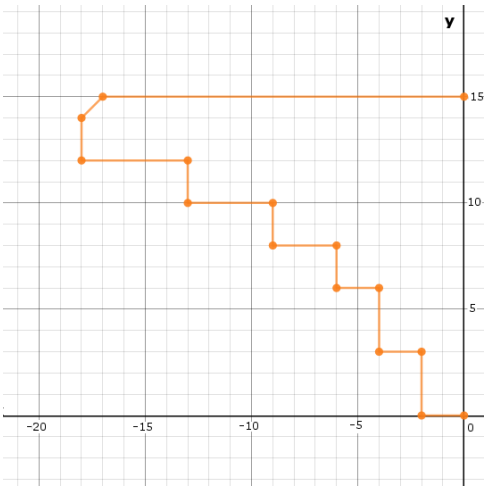
$R = a / (2*\sin(\pi/n))$, где a – длина стороны многоугольника, n – число углов (сторон) многоугольника.

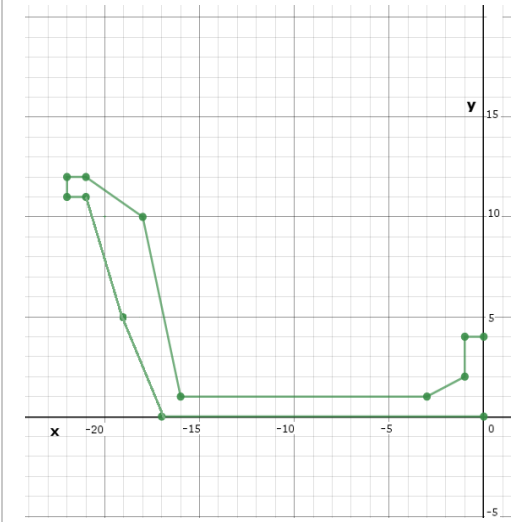
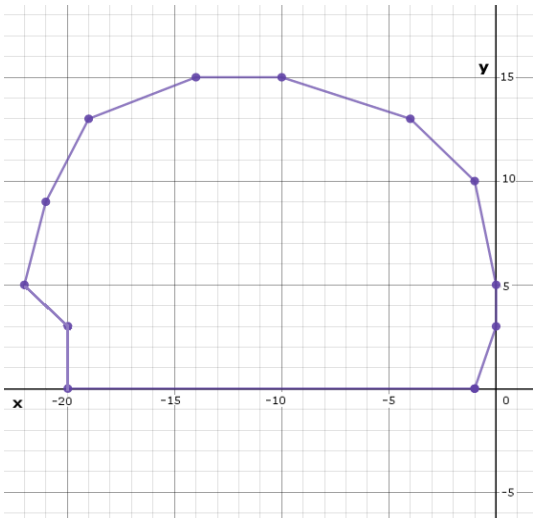
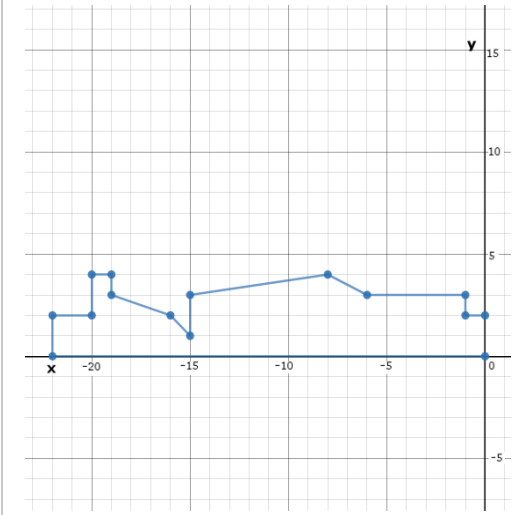
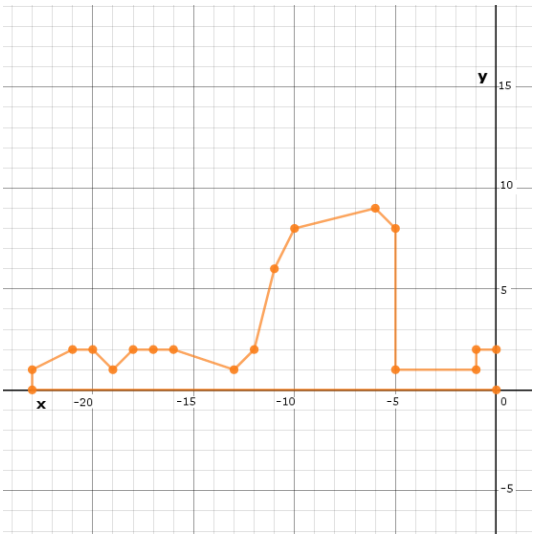
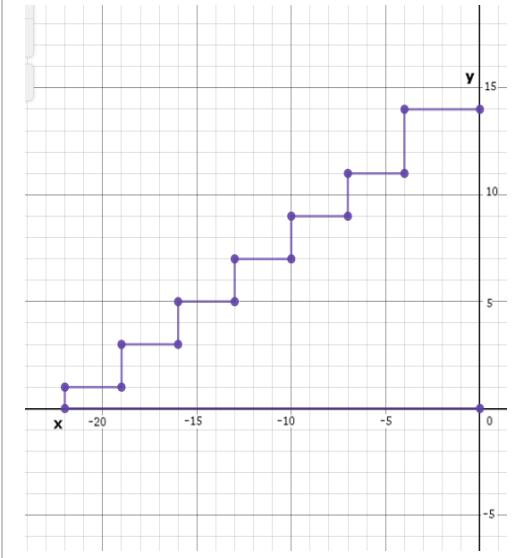
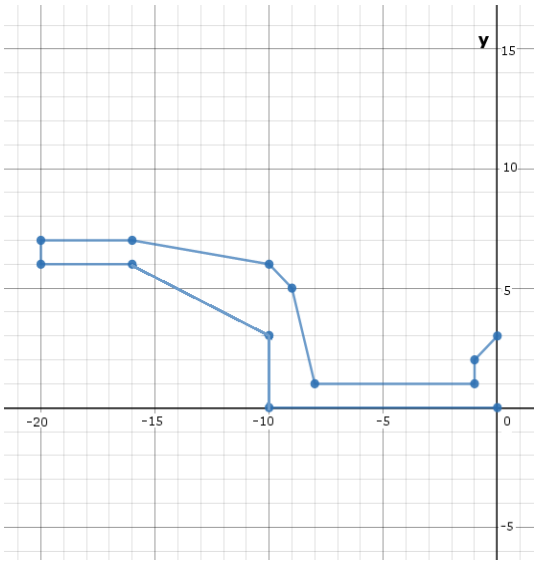
Периметр правильного многоугольника: $P = na$.

4. Задание

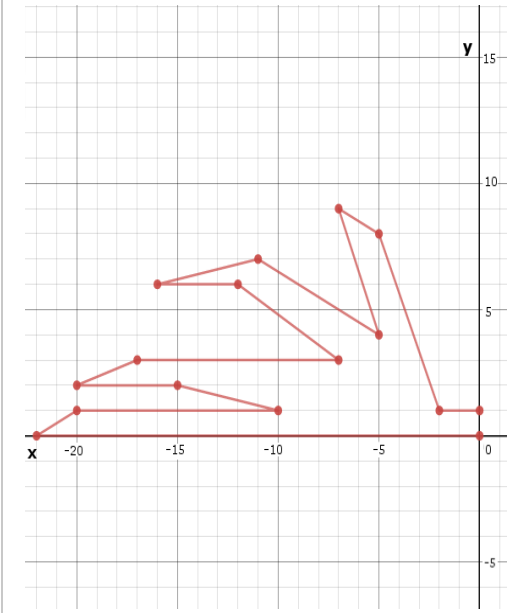
Составьте массив точек по рисунку и смоделируйте тело вращения, образованное поворотом получившегося 2D-объекта вокруг оси *Oz*. Точки образуют замкнутый контур, порядок обхода точек – по часовой стрелке, все координаты – целые числа.

Для моделирования тела вращения используйте функции CAG.fromPoints([…]) и rotateExtrude, параметр resolution последней установить равным 16. Вызов функций описан в методических рекомендациях в п.5.2. Обратите внимание, что использование функции rotateExtrude и ее синтаксис отличается от представленной ранее rotate_extrude, хотя суть операций идентична.

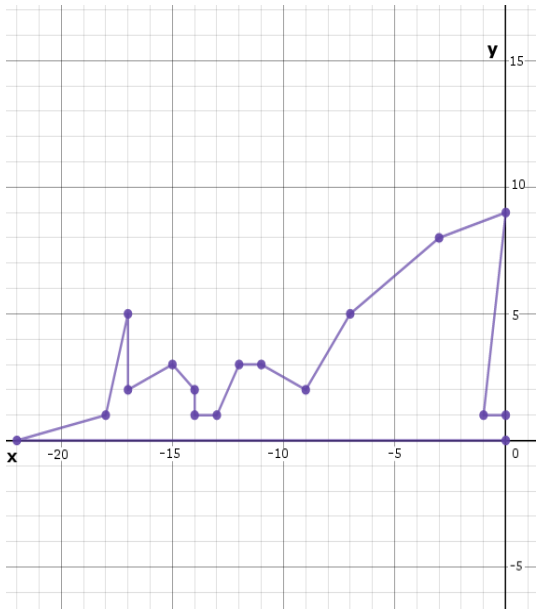
№	Задание	№	Задание
1		2	
3		4	
5		6	

7	 y 15 10 5 x -20 -15 -10 -5 0 -5	8	 y 15 10 5 x -20 -15 -10 -5 0 -5
9	 y 15 10 5 x -20 -15 -10 -5 0 -5	10	 y 15 10 5 x -20 -15 -10 -5 0 -5
11	 y 15 10 5 x -20 -15 -10 -5 0 -5	12	 y 15 10 5 -20 -15 -10 -5 0 -5

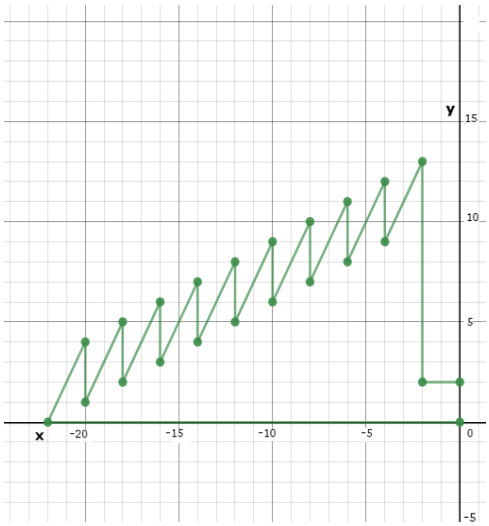
13



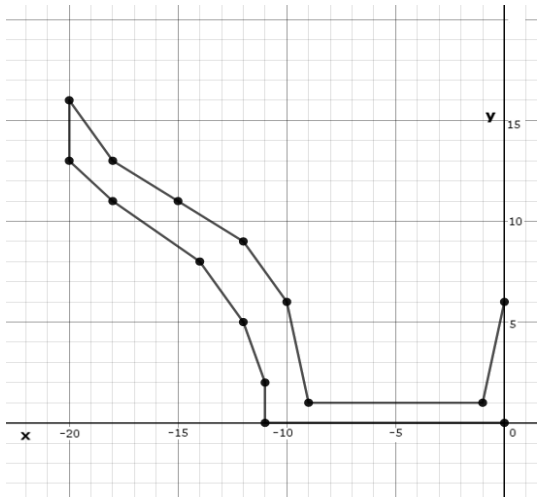
14



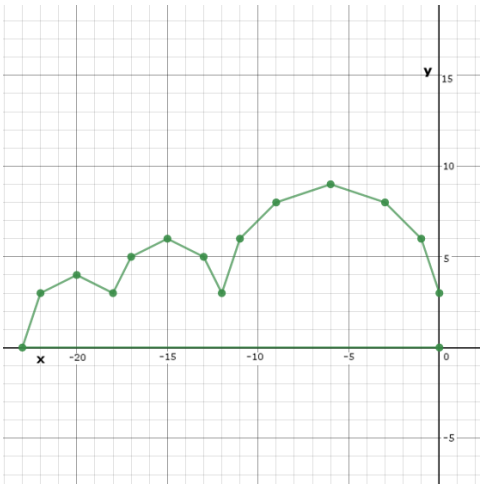
15



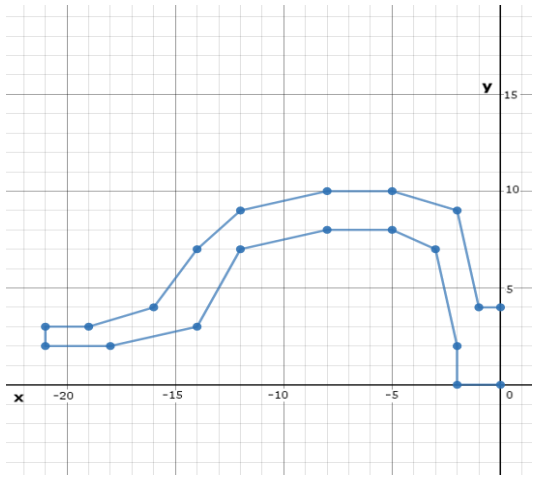
16

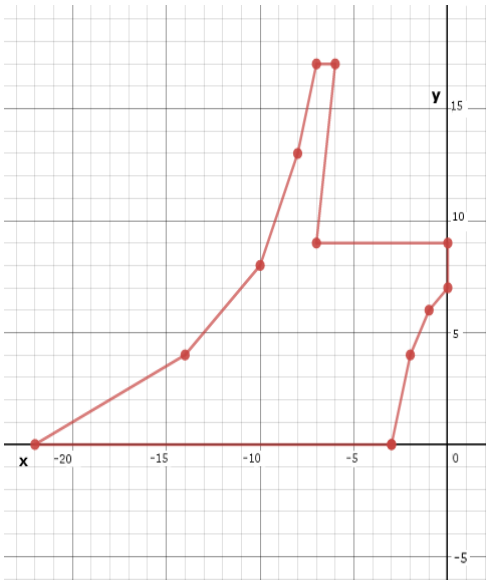
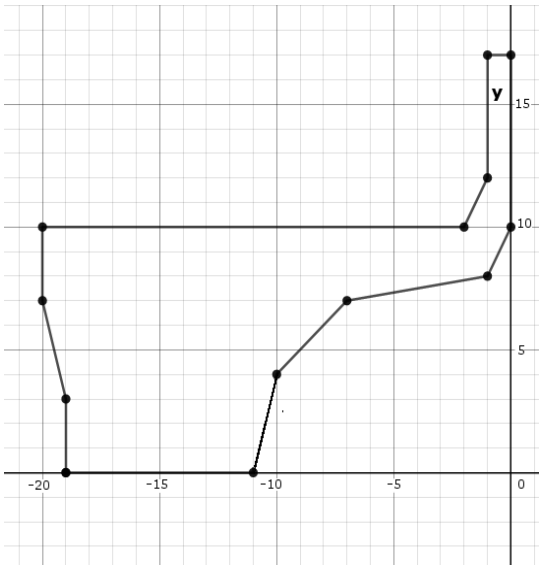


17



18



19		20	

5. Методические рекомендации

Для построения фигуры вращения рекомендуется использовать следующие инструменты OpenJSCAD:

```
CAG.fromPoints(
  [массив точек]
).rotateZ(-90).rotateExtrude({
  resolution: 16
})
```

Лабораторная работа № 9

ДИЗЕРИНГ

1. Цель

Изучение дизеринга на примере пары соседних пикселей, построение чертежа с заданием цвета по 3D-технологии в среде OpenJSCAD.

2. Приобретаемые знания и навыки

- моделирование трехмерных объектов в среде OpenJSCAD на основе примитивов с заданием цвета;
- расчет оттенка на основе метода дизеринга.

3. Теоретическая часть

Цвет	**R**	**G**	**B**
Белый	255	255	255
Голубой	0	191	255

Желтый	255	255	0
Зеленый	0	255	0
Красный	255	0	0
Пурпурный	255	0	255
Синий	0	0	255
Черный	0	0	0

Аддитивная цветовая модель RGB используется для описания цветов, которые получаются с помощью устройств, основанных на принципе излучения. В качестве основных цветов выбран красный (Red), зеленый (Green), синий (Blue). Иные цвета и оттенки получается смешиванием определенного количества указанных основных цветов: Ц = r * R + g * G + b * B.

Соотношение коэффициентов r, g, b определяется с помощью треугольника Максвелла. Из заданной точки проводятся линии, перпендикулярные сторонам треугольника. Длина каждой линии и показывает соответствующую величину коэффициента r, g или b. Одинаковые значения r = g = b имеют место в центре треугольника и соответствуют белому цвету. Сумма коэффициентов равна высоте треугольника, а при высоте, равной единице: r + g + b = 1.

Цвет, создаваемый смешиванием трех основных компонент, можно представить вектором в трехмерной системе координат R, G и B. Черному цвету соответствует центр координат – точка (0, 0, 0). Белый цвет – это вектор (1, 1, 1).

Таблица цветов

Дизеринг – это обмен разрешающей способности на количество цветов (dithering – дрожание, разрежение). Простейшими вариантами дизеринга можно считать создание оттенка цвета парами соседних пикселей. Ячейка номер 1 дает оттенок цвета $C = (C_1 + C_2) /2$, где C_1 и C_2 - цвета, которые графическое устройство способно непосредственно воспроизвести для каждого пикселя.

4. Задание

Согласно представленному заданию рассчитать новый оттенок цвета по методу дизеринга, получить его разложение по составляющим (R G B). Визуализацию результата расчета провести следующим образом: построить цилиндр с параметрами:

- h = R – высота цилиндра (численно равна расчетному значению красной составляющей);

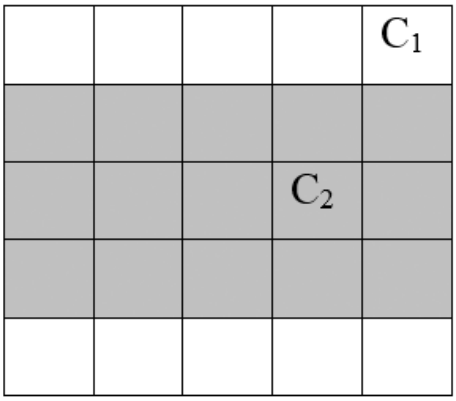
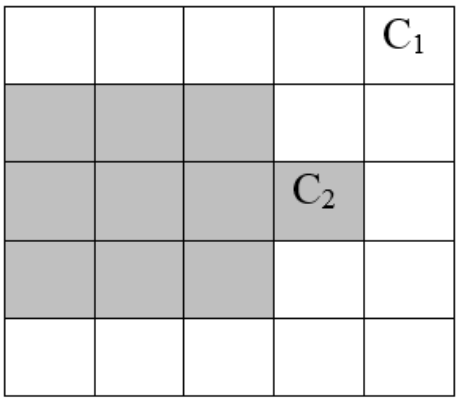
• r_1 = G – начальный радиус (численно равен расчетному значению зеленой составляющей) – нижний радиус;

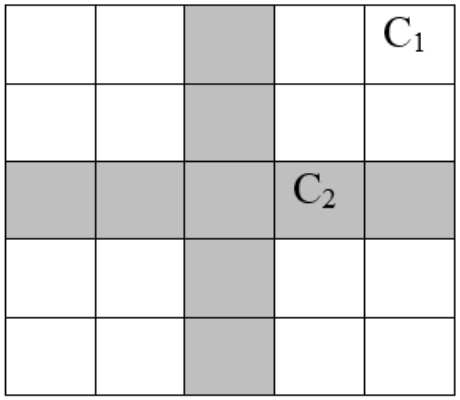
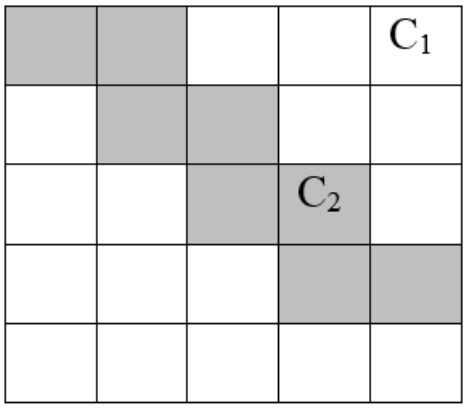
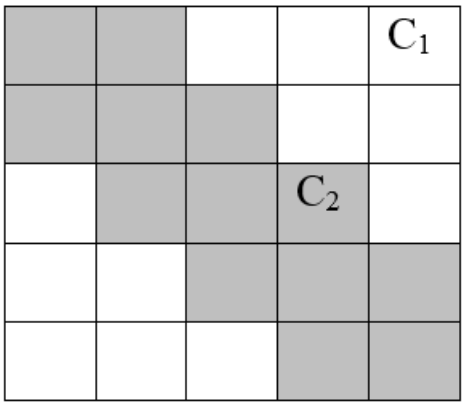
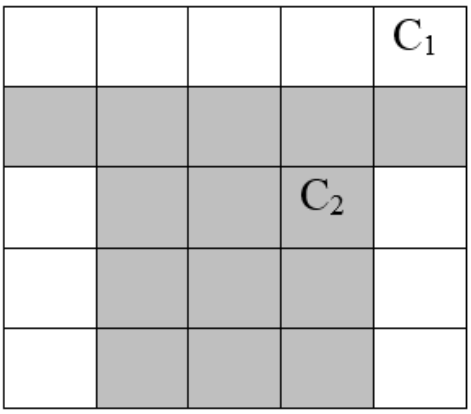
• r_2 = B – конечный радиус (численно равен расчетному значению синей составляющей) – верхний радиус;

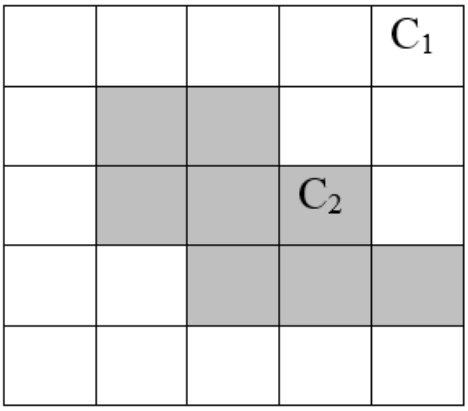
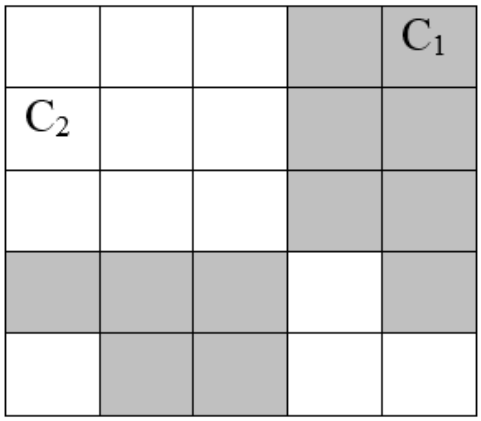
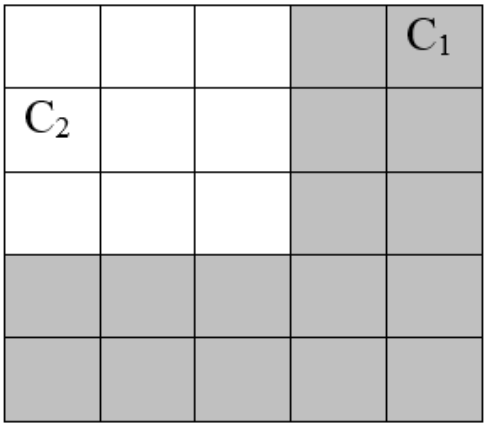
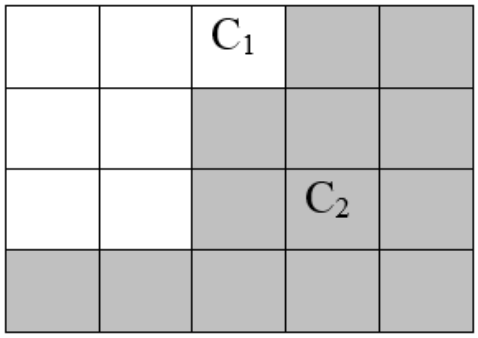
• цвет цилиндра задать согласно полученному решению (для избежания ошибок округления параметры функции setColor указывать в виде дробей R/255, G/255 и B/255).

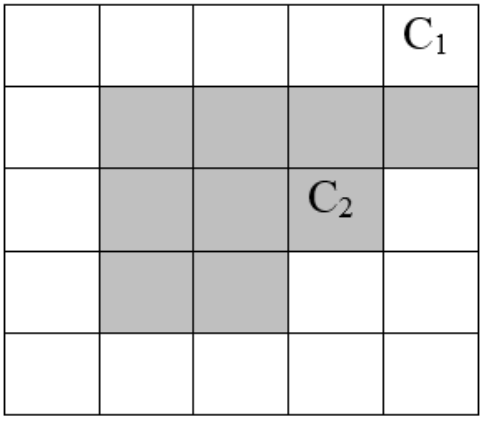
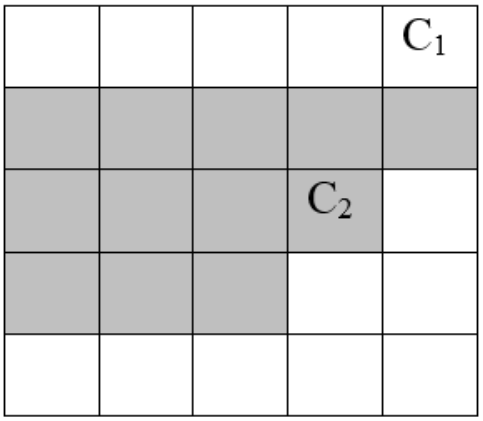
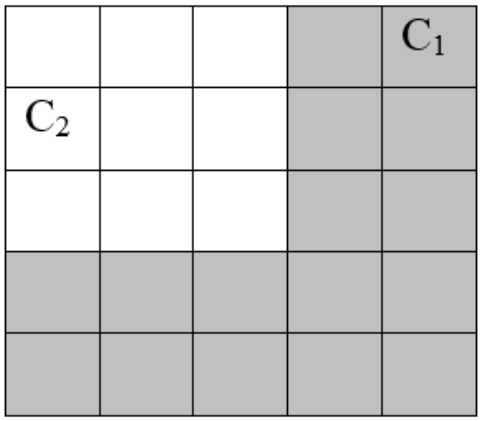
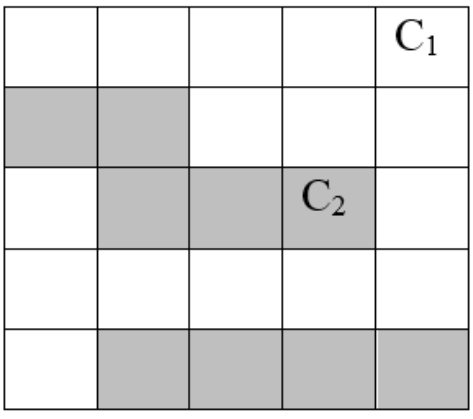
Построение осуществляется с использованием инструментов OpenJSCAD. Параметры по умолчанию: разрешение = 32. Готовому объекту задать цвет согласно полученным расчетам. Объект ориентировать относительно осей координат следующим образом: начальный радиус расположен ВНИЗУ, конечный радиус расположен ВВЕРХУ. Координаты [x, y, z] центра цилиндра принять равными [0, 0, 0].

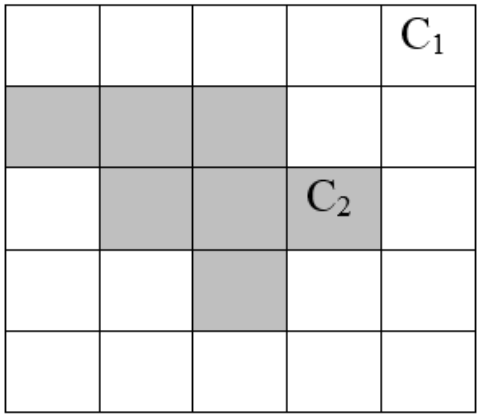
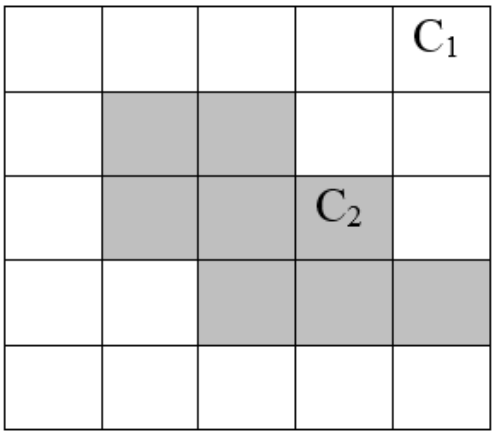
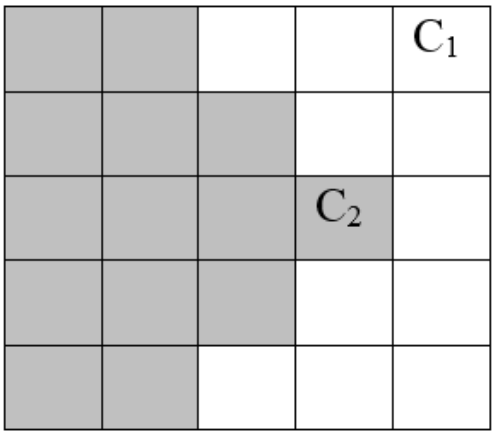
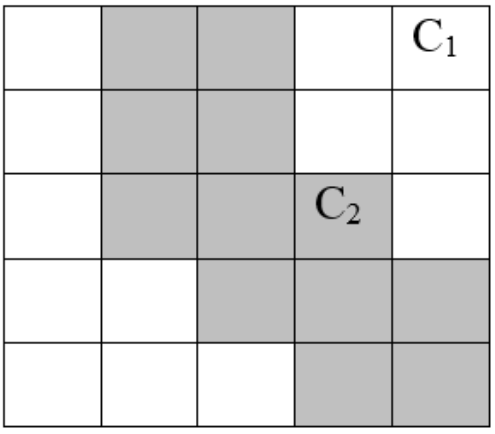
Варианты заданий:

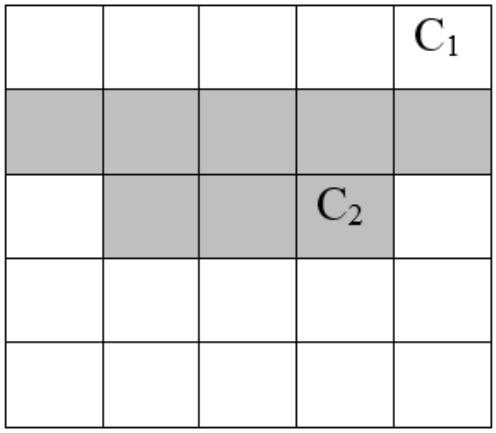
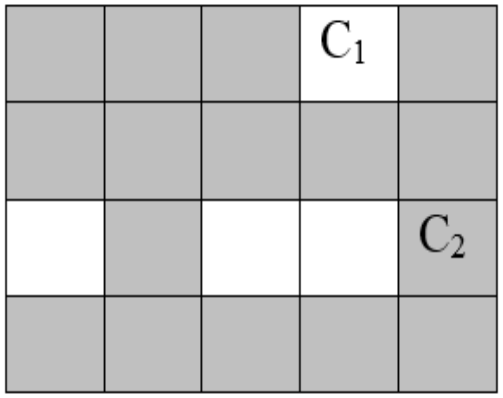
№ варианта	Рисунок	Цвета
1		C_1 – желтый C_2 – красный
2		C_1 – синий C_2 – красный

3		C_1 – желтый C_2 – красный
4		C_1 – белый C_2 – черный
5		C_1 – красный C_2 – белый
6		C_1 – белый C_2 – красный

7		C_1 – голубой C_2 – синий
8		C_1 – белый C_2 – синий
9		C_1 – красный C_2 – белый
10		C_1 – белый C_2 – пурпурный

11		C_1 – черный C_2 – белый
12		C_1 – синий C_2 – белый
13		C_1 – красный C_2 – белый
14		C_1 – белый C_2 – синий

15		C_1 – красный C_2 – синий
16		C_1 – голубой C_2 – синий
17		C_1 – желтый C_2 – красный
18		C_1 – желтый C_2 – синий

19		C_1 – зеленый C_2 – белый
20		C_1 – зеленый C_2 – черный

Расчет нового оттенка осуществляется следующим образом:

$$C = \frac{C_1 S_1 + C_2 S_2}{S}$$, где:

S – общая площадь ячейки;

S_1 – часть площади, занимаемая цветом C1;

S_2 – часть площади, занимаемая цветом C2;

C_1 – интенсивность каждой из R-G-B составляющих для цвета C1;

C_2 – интенсивность каждой из R-G-B составляющих для цвета C2.

Площадь одного пикселя принимаем за единицу. Тогда площадь, занимаемая пикселями, равна их количеству и $S = S_1 + S_2$. Так для ячейки размером 2X2, где половина пикселей белая, а половина черная получим оттенок:

$$C_R = \frac{C_{1R} S_1 + C_{2R} S_2}{S} = \frac{255 * 2 + 0 * 2}{4} = 127.5$$

$$C_G = \frac{C_{1G} S_1 + C_{2G} S_2}{S} = \frac{255 * 2 + 0 * 2}{4} = 127.5$$

$$C_B = \frac{C_{1B} S_1 + C_{2B} S_2}{S} = \frac{255 * 2 + 0 * 2}{4} = 127.5$$

C (127.5, 127.5, 127.5) – светло-серый.

Для построения цилиндра рекомендуется использовать следующие инструменты OpenJSCAD:

```
CSG.cylinder({
  start: [0, -1, 0],    //начальная точка построения (центр дна)
  end: [0, 1, 0],       //конечная точка построения (центр вершины)
  radiusStart: 1,       //начальный радиус
  radiusEnd: 2,         //конечный радиус
  resolution: 32        //разрешение
})
```

Для задания цвета рекомендуется использовать следующие инструменты OpenJSCAD:

```
.setColor([
 1,                     //красная составляющая
 0.5,                   //зеленая составляющая
 0.3                    //синяя составляющая
])
```

Параметры (R,G,B,) функции setColor задаются как пропорция 1:1:1. Для перехода расчетные значения необходимо разделить на 255.

Лабораторная работа № 10

ПОСТФИЛЬТРАЦИЯ

1. Цель

Изучение методов растровой обработки изображения – методов постфильтрации.

2. Приобретаемые знания и навыки

- расчет значений интенсивности пикселя при применении различных масок постфильтрации;
- моделирование трехмерных объектов в среде OpenJSCAD на основе примитивов;

3. Теоретическая часть

Под фильтрацией изображений понимают операцию, имеющую своим результатом изображение того же размера, полученное из исходного по некоторым правилам. Обычно интенсивность (цвет) каждого пикселя результирующего изображения обусловлена интенсивностями (цветами) пикселей, расположенных в некоторой его окрестности в исходном изображении.

Правила, задающие фильтрацию (их называют фильтрами), могут быть самыми разнообразными. Линейные фильтры представляют собой семейство фильтров, имеющих очень простое математическое описание. Вместе с тем они позволяют добиться самых разнообразных эффектов. Будем считать, что задано исходное полутоновое изображение A, и обозначим интенсивности его

пикселей A(x,y). Линейный фильтр определяется вещественнозначной функцией F, заданной на растре. Данная функция называется ядром фильтра, а сама фильтрация производится при помощи операции дискретной свертки (взвешенного суммирования).

При использовании метода двойного опроса при вычислении интенсивности каждого пикселя дисплея производится усреднение девяти соседних замеров, причем каждый из них одинаково важен. Такая форма фильтрации или размытия границ (bluring) может быть улучшена, если придать центральному замеру больший вес, а восьми его соседям – меньший. Маска 1/16 (рис. 1) называется «окно Барлетта» 3x3.

1/16	1/16	1/16
1/16	1/8	1/16
1/16	1/16	1/16

Рис. 1. Окно Барлетта

4. Задание

Результаты опроса сцены представлены в виде таблицы 1. Нумерация столбцов и строк начинается нуля (номера приведены в таблице). Необходимо для пикселя с заданным номером определить результат применения:

- маски 1/16 («окно Барлетта» 3x3) – f2;
- маски 1/8 (3x3) – f1.

Необходимо построить:

- трехмерный объект на основе окружности радиусом f2 (округлив полученное расчетное значение до целого) с помощью функции extrude на высоту f1 (округлив полученное расчетное значение до целого) с разрешением 100.

Правила округления значений: округляем до НАИМЕНЬШЕГО числа с тремя знаками после запятой, БОЛЬШЕГО или РАВНОГО аргументу. Так дробь 2.3703649952833614 после округления будет равна не 2.37, а 2.371. А дробь 41.9375 будет равна 41.938.

Таблица 1

	0	*1*	*2*	*3*	*4*	*5*	*6*
0	17	21	31	36	13	35	21
1	28	43	28	16	4	18	23

2	31	53	42	30	17	34	42
3	48	52	53	60	62	18	21
4	41	54	33	15	60	24	24
5	21	35	36	13	30	19	27

Варианты заданий

№ варианта	Координаты пикселя
1	[1, 1]
2	[1, 2]
3	[1, 3]
4	[1, 4]
5	[1, 5]
6	[2, 1]
7	[2, 2]
8	[2, 3]
9	[2, 4]
10	[2, 5]
11	[3, 1]
12	[3, 2]
13	[3, 3]
14	[3, 4]
15	[3, 5]
16	[4, 1]

17	[4, 2]
18	[4, 3]
19	[4, 4]
20	[4, 5]

5. Методические рекомендации

Рассмотрим применение окна Барлетта на примере.

Таблица 2

17	**21**	31	**36**	9
28	43	28	16	4
31	**53**	42	**30**	17
50	52	53	60	62

Каждое числовое значение таблицы 2 описывает интенсивность опроса сцены. Жирным выделены числа, соответствующие центрам различных пикселов дисплея. На каждый такой квадрат поочередно накладывается квадратная маска – окно Барлетта (рис. 1). Рассмотри значение 30. Его средневзвешенное равно:

S = 30/2 + (28 + 16 + 4 + 42 + 17 + 53 + 60 + 62) / 16 = 32.625 ≈ 33.

При применении маски 1/8 (3x3) центральное значение игнорируется. И интенсивность пикселя определяется как среднее значение интенсивности восьми соседних пикселей.

Визуализация результатов расчетов проводится в среде OpenJSCAD. Для построения трехмерной модели рекомендуется использовать следующие инструменты OpenJSCAD:

- - CAG.circle({center: [x, y], radius: r, resolution: });
- - extrude({offset: [x, y, z], twistangle: , twiststeps: });

Функция circle возвращает круг с центром в точке, заданной параметром center; радиусом, равными r. Параметр resolution задает разрешение, с которым строится круг.

Функция extrude({offset: [x, y, z], twistangle: , twiststeps: }) – функция, позволяющая выдавливанием формировать трехмерный объект на основе плоской фигуры. Параметр offset: [x, y, z] задает конечную точку траектории, twistangle – угол, на который повернется основа по сравнению с начальным положением за время движения по траектории. Параметр twiststeps задает число поворотов, за которое достигается заданный угол поворота.

Дудник Евгения Александровна

ГЕОМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ

Лабораторный практикум

Учебно-методическое пособие для студентов направления «Информатика и вычислительная техника»

Подписано к печати 10.03.20. Формат 60х84 1/16.
Усл. печ. л. 4,25. Тираж 30 экз. Заказ № 201753. Рег. № 34.

Отпечатано в ИТО Рубцовского индустриального института
658207, г. Рубцовск, ул. Тракторная, 2/6.