Содержание

Введение

Система IndorCAD содержит встроенный редактор 3D-объектов, работа с которым подробно описана в документации. Однако не зная языка программирования, который описывает трёхмерные объекты в системе, можно пользоваться лишь готовыми моделями: добавлять их в проект и настраивать доступные параметры в инспекторе объектов.

Данное руководство предназначено для того, чтобы помочь пользователям системы IndorCAD при создании собственных 3D-объектов. Изучение формальной грамматики кода позволит любому пользователю системы, ранее не знакомому с языками программирования, редактировать существующие и создавать собственные трёхмерные объекты, которые затем можно использовать при оформлении 3D-вида проекта.

В руководстве даются основные правила создания трёхмерных объектов; на конкретных примерах разбираются особенности кода, его структура, ключевые слова, описания команд.

Общий вид программы

В самом общем виде программа для создания 3D-объектов выглядит следующим образом:

```
Goal → Description Code ‘end’

Description → ‘BeamObject’ 'ObjectName' [Material]
[Distances] [PreviewInfo]‘;’

Code → {ConditionSection | ColorSection | VarSection |
ConstSection | CoordsSection | ShapesSection |
ElementsSection}+
```

Программа состоит из описания, кода и завершается строкой ‘end’.

Первой строкой идёт описание: обязательный **BeamObject**, затем обязательное **Имя** в одинарных кавычках, необязательный **Материал**, используемый по умолчанию, необязательные **Настройки расстояний**, необязательная информация для генерации **Preview**. Заканчивается точкой с запятой.

Пример первой строки:

```
BeamObject 'Светофор горизонтальный' Steel ![200, 0.7,
0.3] {$X=0; Y=-1.5; Z=0};
```

Здесь **BeamObject** — формальный признак того, что это код для разбора.

'Светофор горизонтальный' — имя объекта. Будет отображаться в свойствах выделенного объекта.

Steel — материал. Есть две «встроенные» константы материалов: Steel (=$969696) и Glass (=$FAFAFA). Можно ничего не указывать, а можно указать цвет в формате, как в ColorSection (см. ниже).

![200, 0.7, 0.3] — настройки расстояний: восклицательный знак и три числа в квадратных скобках, разделённые запятыми.

- Первое число (200) — расстояние от камеры, после превышения которого объект вообще не будет отображаться.
- Второе число (0.7) — относительное расстояние (от первого числа),

дальше которого можно пытаться упрощать в 2 раза отрисовку объекта.

- Третье число (0.3) — относительное расстояние (от первого числа), ближе которого нужно пытаться улучшить в 2 раза отрисовку объекта.

**{$X=0;Y=-1.5;Z=0}** — метаинформация для размещения камеры при генерации Preview объекта. Заворачивается в {$ }. Может содержать 6 полей (неуказанные считаются равными нулю), разделённых точкой с запятой:

- X,Y,Z — описывают точку, из которой смотрит камера при генерации Preview;
- dX,dY,dZ — описывают точку, на которую должна смотреть камера при генерации Preview.

После описания объекта начинается основная часть программы: прописываются различные элементы кода, по которым строится 3D-модель. Завершается программа строкой ‘end’.

Описание кода

В этой главе содержится описание таких элементов кода, как скалярное и векторное выражение, набор управляемых условий, набор материалов, переменные, константы, константы для трёхмерных точек, геометрия плоских фигур, а также рассматривается код, формирующий 3D-объект. Все разделы снабжены примерами кода, которые можно использовать при создании и редактировании 3D-объектов.

Скалярное и векторное выражение

По коду часто используются термины Expression (скалярное выражение) и PointExpression (векторное выражение). Немного подробнее о них.

Грамматика:

```
Expression → [{'-'|'+'}] Term [{'-'|'+'} Term]*

Term → Factor [{'*'|'/'} Factor]*

Factor → {<Значение>|Function|Identifier|PointIndentifier.
{X|Y|Z}|'('Expression')'}

Function → {'Sin'|'Cos'|'Tan'|'Abs'|'Round'|'Sqrt'|'Sqr'}
(Factor)

Identifier → {ConstName|VariableName}
```

Выражение — константа, имя переменной или математическое выражение с применением скобок, умножений/делений, сложений/вычитаний, вызовов функций. Тригонометрические функции в качестве аргумента принимают градусы — так удобнее кодировать объекты. При использовании выражений можно обращаться к переменным типа «3D-точка» для получения одной из координат.

Например, Len = Sqrt(Sqr(Pt.X) + Sqr(Pt.Y) + Sqr(Pt.Z))

Грамматика:

```
PointExpression → [{'-'|'+'}] PointTerm [{'-'|'+'}
PointTerm]*

PointTerm → PointFactor [{'*'|'/'} Expression]*

PointFactor → {Point | (PointExpression)}

Point → {Coord3|PointIndentifier}

Coord3 → '['Expression',' Expression [',' Expression]']'
```

Векторное выражение — трёхмерная или двумерная (Z=0) точка или имя точечной константы либо векторное выражение с применением простых операций: сложение/вычитание векторов и умножение/деление вектора на скаляр. Векторная константа (трёхмерная точка) может быть определена

тремя или двумя (Z=0) координатами, разделёнными запятыми, взятыми в квадратные скобки.

Примеры:

```
const
  h=3;
coord
  A = [h,0,0];
  B = [1.2, 0, h];
  C = B*2 + A;
elements
  for i=0 to 10 do
  begin
    C = C + [0, 0, 0.1];
    ...
  end;
```

Набор управляемых условий

```
'condition'

(ConditionName [{$DisplayName}] = Expression [{'test' |
'time' Expression (Expression [, Expression]* )}]
[OrderInfo];)+
```

После ключевого слова **condition** идут описания условных (логических: истина/ложь) переменных. Сначала указывается обязательное имя условия, необязательная метаинформация об отображаемом в настройках имени (используется, если пользователю разрешено изменять значение условия). Затем следует знак равенства и выражение. Условие считается истинным (True), если значение выражения отлично от нуля, иначе — ложным (False). Выражение описывает начальное состояние условия.

После выражения может стоять слово test (без кавычек), что означает изменение значения условия 2 раза в секунду на противоположное — таким образом можно сделать «мигающий» объект. Либо после выражения может стоять слово time (без кавычек), что означает, что далее идёт описание того, как со временем должно меняться значение условия (все значения в секундах).

Первое выражение описывает сдвиг по времени относительно некоторого «нуля» и позволяет как бы сдвинуть значения на константу. Второе выражение описывает общее время цикла, после которого цикл запустится заново. Последующий набор значений, идущий в скобках, указывает время, в течение которого условие сохраняет значение до его смены.

Заканчивается описание условия точкой с запятой, перед которой может быть **OrderInfo** — метаинформация, использующаяся, если есть метаинформация об имени.

Пример секции:

```
condition
  // Простые примеры
  AlwaysTrue = 1; // Условие всегда истинно.
  AlwaysFalse = 0; // Условие всегда ложно.
```

```
Blinking1 = 0 test; // Меняет значение с 0->1->0->1 2 раза в секунду. Идентично = 0 time 0 1 [0.5].

Blinking2 = 1 test; // Мигает в противофазе с Blinking1.

Blinking3 = 0 time 0 2 (0.1, 0.2, 0.1); // Двойной стробоскоп (2 короткие вспышки по 0.1 с и интервалом 0.2 с и периодом 2 с).

// С пользовательским интерфейсом (возможностью изменения условия в ИО)

IsNight {$Фары включены} = 1 {$Order=1}; // Значение по умолчанию = True, в UI свойство сортируется с Order=1.

DrawFixtures {$Отображать арматуру} = 0 {$Order=2}; // Значение по умолчанию = False, в UI свойство сортируется с Order=2.
```

Набор материалов

```
'color'

(MaterialName [{$DisplayName}] = Material ['if'
ConditionName Material];)+

Material → {MaterialName | Color3 [Params3]};

Color3 → '[' Expression ',' Expression ',' Expression']'

Params3 → '[' Expression ',' Expression ',' Expression']'
```

После ключевого слова **color** идут описания используемых материалов: обязательное имя материала, необязательная мета-информация об отображаемом в настройках имени (используется, если пользователю разрешено изменять значение цвета). Затем следует знак равенства и описание материала, после которого может стоять условное применение другого материала. Второй материал будет использоваться, если значение условия, идущего после слова if, истинно.

Заканчивается описание материала точкой с запятой, перед которой может быть **OrderInfo** — метаинформация, использующаяся, если есть мета-информация об имени.

Color3 — R,G,B-компоненты цвета, в квадратных скобках, разделённые запятыми. Допускаются значения от 0 до 255.

Params3 — компоненты материала в квадратных скобках (Diffuse, Ambient, Emissive), разделённые запятыми. Значения от 0.0 до 1.0. Если не указано, то Params3 = [0.4, 0.6, 0.0]

Пример секции:

```
color

  // Простые примеры

  SimpleBlue = [0, 0, 255]; // Синий стандартный.

  SimpleWhite = [255, 255, 255]; // Белый стандартный.

  LightingWhite = [255, 255, 255] [1, 1, 1]; // Белый, све-
  тящийся в темноте.
```

```
BlueOrWhite = SimpleBlue if Blinking1 LightingWhite; // Синий или светящийся белый, если условие Blinking1 = True.

// С пользовательским интерфейсом (возможностью изменения цвета в ИО).

MainColor {$Основной цвет} = SimpleBlue {$Order=1}; // Начальное значение = SimpleBlue, в UI свойство сортируется с Order=1.

SecondaryColor {$Дополнительный цвет} = [255,0,0] [1,0,1] {$Order=2}; // Начальное значение — красный, в UI свойство сортируется с Order=2, светится в темноте.
```

Переменные

```
'var'

(VarName [{$DisplayName}] = Expression [Expression
Expression] [MetaInfo];)+
```

После ключевого слова **var** идут описания переменных, которые могут быть настроены пользователем. Сначала указывается обязательное имя переменной, желательная метаинформация об отображаемом в настройках имени. За значением переменной могут следовать два выражения, означающие минимально и максимально допустимое значение переменной. Это будет использоваться в UI.

Заканчивается описание переменной точкой с запятой, перед которой может быть метаинформация {$ } с дополнительными директивами, позволяющими улучшить UI по настройке объекта. Директивы разделяются точкой с запятой. Возможные директивы:

```
Digits=Value; // Количество знаков после запятой в свой-
стве.

Step=Value; // Шаг изменения значения при вращении колеса
мыши и/или нажатии клавиши вверх/вниз.

Min=Value; // Минимально допустимое значение (перекрывает,
если было указано иное ДО метаинформации).

Max=Value; // Максимально допустимое значение (пере-
крывает, если было указано иное ДО метаинформации).

Order=Value; // Относительный порядок свойства в ИО. Все
свойства сортируются по Order.

ReadOnly; // Признак того, что данное свойство носит спра-
вочный характер (вычисляется) и только отображается поль-
зователю.
```

Пример секции:

```
var

  // Примеры

  Scale {$Масштаб, %} = 100 10 500 {$Digits=0; Step=10};
  // Настраиваемая переменная, которая в UI выглядит как
```

«Масштаб, %», имеющая по умолчанию значение 100 (при минимуме 10 и максимуме 500); при этом количество знаков после запятой = 0, а шаг изменения = 10.

Inc {$Поперечный уклон, $%} = 0 {$Digits=1; Step=1; Min=-60; Max=60; Order=2}; // Настраиваемая переменная «Поперечный уклон, ‰», имеющая по умолчанию значение 0 (при минимуме -60 и максимуме 60), с одним знаком после запятой, шагом 1; при выводе свойств переменная отображается согласно Order=2.

Hmm {$Высота гребня, мм} = 50 50 60 {$Digits=0; Step=1; Order=5}; // Настраиваемая переменная «Высота хорды, мм», имеющая по умолчанию значение 50 (при минимуме 50 и максимуме 60), без знаков после запятой, шагом 1; при выводе свойств переменная отображается согласно Order=5.

K {$Используемый коэффициент}=(A+B+2*C)/2 {$Digits=1; ReadOnly; Order=9}; // Справочное (нередактируемое) значение, отображаемое с одним знаком после запятой, с Order=9; вычисляется по определённым выше переменным и константам как (A+B+2*C)/2.

Константы

‘const’

(ConstName = Expression;)+

После ключевого слова **const** идут описания констант, которые могут быть переопределены в коде ниже в секции **Elements**. Указывается обязательное имя константы, знак равенства и выражение, описывающее начальное значение константы.

Пример секции:

```
const

  // Примеры

  In = Inc/1000; // Уклон, используемый в вычислениях
  (Inc — переменная, определяемая пользователем в секции
  Var).

  H = Hmm/1000; // Приведённое к метрам значение, заданное
  пользователям в миллиметрах. Далее оно может быть исполь-
  зовано в коде для повышения быстродействия и избежания
  копипаста «Hmm/1000».
```

Константы для трёхмерных точек

```
‘coord’

(CoordName = PointExpression;)+
```

После ключевого слова **coord** идут описания координат, которые могут быть переопределены в коде ниже в секции **Elements**. Указывается обязательное имя координаты, знак равенства и выражение, описывающее начальное значение координаты.

Пример секции:

```
coord

  // Примеры

  Pos = [-Length/2, 0, -Length/2]; // Простая трёхмерная
  координата.

  Pos2 = Pos*2; // Удвоенная по каждому измерению коор-
  дината Pos.

  Pos3 = Pos2 - Pos; // Разность между Pos и Pos2 (век-
  тор).

  Start = -Pos + Pos2*2 + (Pos3*R - Pos); // Более сложное
  выражение (такие редко используются).
```

Координаты (объявленные тройки чисел) могут также использоваться в настройках расстояний для объектов или их групп: см. пример ниже в общих правилах описания примитивов.

Описание геометрии плоских фигур

```
‘shape’

(ShapeName = ShapeDescription;)+

ShapeDescription → PartDesription+

PartDesription → Expression (PointExpression [,
PointExpression]* )
```

После ключевого слова **shape** идут описания плоских фигур, использующихся далее в коде для «выдавливания» примитивов. Указывается обязательное имя шейпа, знак равенства и описание фигуры, состоящей из одной части или более. Каждая часть описывается как число точек (Expression); далее следуют сами точки части (PointExpression), разделённые запятой. Если фигура имеет несколько частей (или содержит дырки), то каждая последующая часть описывается сразу после предыдущей в том же формате: число точек [точки].

Пример секции:

```
coord

  // Пример симметричной фигуры из одной части из семи
  точек

  Main = 7

  [-L,0], [-L*3/5,H/3], [-L/5,H*9/10],

  [0,H],

  [L/5,H*9/10],[L*3/5,H/3], [L,0];

  // Пример контурной стрелки светофора (не сплошной, с
  дыркой внутри)

  Arrow2 = 7 [0,0.8], [0.4,0], [0.15,0], [0.15,-0.8],
  [-0.15, -0.8], [-0.15,0], [-0.4,0]

  7 [0,0.6], [0.25,0.1], [0.07,0.1], [0.07,-0.72],
  [-0.07,-0.72], [-0.07,0.1], [-0.25,0.1];
```

Код, формирующий 3D-объект

```
'elements'

Primitives

Primitives → { ConstAssignment | CoordAssignment |
ForStatement | Primitive }+;

ConstAssignment → ConstName '=' Expression';'

CoordAssignment → CoordName '=' PointExpression';'

ForStatement → 'for' ConstName '=' Expression 'to'
Expression ['step' Expression] 'do' Primitive

OnePrimitive → [Condition] [Distances] [Material] {Group |
Sphere | Beam | Cone | Bezier | ZPart | Light | Shape |
Shapes | TimeCounter}

Group → 'begin' Primitives 'end' ';'
```

После ключевого слова **elements** идут описания команд, которые генерируют 3D-объект. В командах могут присутствовать переопределения констант, описанных ранее в секции const, переопределения координат, описанных ранее в секции coord, операторы цикла for и собственно описания примитивов.

При описании цикла необязательно пользоваться константой, описанной ранее в секции const. Если использовать неописанную константу (например, for i=0 to 10), то новая константа будет добавлена в список автоматически. Далее её можно будет использовать в коде.

Примитивы будут рассмотрены ниже, а пока приведём простенький пример секции elements, рисующий 2 спаренных лесенки (побольше и поменьше):

```
BeamObject 'Спаренная лестница';

elements

  // Короткая лесенка

  for i=0 to 4 do

    beam 0.05 [0, 0, 0.1+i*0.15] rel [0,0.5,0];

  // Длинная лесенка
```

```
  for i=0 to 7 do
    beam 0.05 [0,0,0.17+i*0.15] rel [0,-0.5,0];
  // Стойки
  Beam 0.05 [0,0,0] [0,0,1.3];
  Beam 0.05 [0,-0.5,0] rel [0,0,1.3];
  Beam 0.05 [0,0.5,0] rel [0,0,0.8];
end
```

Описание примитивов

В данном разделе содержится информация об общих правилах описания примитивов, а также примеры кода конкретных объектов, таких как сфера, балка, конус, сегмент трёхмерной кривой Безье, металлическая конструкция из уголков, объект свечения, фигура выдавливания, счётчик секунд и группа элементов.

Общие правила описания примитивов

Качество отображения

Если в описании примитива присутствует необязательный параметр [Quality], то это означает, что данный примитив может по-разному отображаться в зависимости от расстояния камеры до него, но только если заданы настройки расстояний отображения (см. ниже). По умолчанию (если не указано) качество считается равным 6, т.е., например, цилиндры будут аппроксимироваться шестигранником и т.д.

Условия отображения примитива

Условия отображения примитива могут зависеть от настраиваемой пользователем опции или переключаться по времени. Описание примитива может начинаться с условия, согласно которому примитив должен или не должен отображаться. Для этого в начало добавляется описание условия:

```
BeamObject 'Условное отображение перекладины';

condition

  A = 1; // Отображать ли перекладину.

elements

  // Безусловные элементы

  for i=-0.5 to 0.5 do

    beam 0.05 [i, 0, 0] rel [0,0,1.5];

  // Условный элемент, зависящий от условия A

  if A Beam 0.02 [-0.5,0,1.4] rel [1,0,0]; // Примитив
  будет отрисован, только если A не равно нулю.

end
```

Настройки расстояний отображения

Настройки расстояний отображения используются для оптимизации времени отрисовки всего объекта. Перед описанием примитива можно добавить настройки расстояний (Distances), которые работают так же, как указано выше при описании всего объекта.

Пример:

```
BeamObject 'Снеговик';
coord
  // Если > 20 метров — объект не рисуем, если > 10 метров
  — упрощаем отрисовку, если < 4 — рисуем более детально.
  Dist = [20, 0.5, 0.2];
const
  h=0;
  R=2;
elements
  // Три шара
  for i=1 to 3 do
  begin
    h=h+r/2;
    sphere [0,0,h] r 12;
    h=h+R/4;
    R=R*0.75;
  end;
  // Нос рисуем только на расстоянии не более 50*R м
  ![50*R, 0.5, 0.2] cone [0, R/2, h-R/4] rel [0, R, 0] R/5
  0 [255,0,0] 6;
  // Отрисовка глаз согласно значениям в Dist
  for i=-1 to 1 step 2 do
    !Dist sphere [-i*r/4,r/2,h] r*0.1 [0,0,0];
end
```

Цвет отображения объекта

Этот способ немного устарел, лучше его не использовать. Описанию примитива может предшествовать название или описание материала:

```
BeamObject 'Три колонны разных цветов';

color

  MyMaterial = [255,0,255];

elements

  Beam 0.05 [0,0,0] rel [0,0,1]; // Элемент будет отрисо-
  ван материалом по умолчанию.

  MyMaterial Beam 0.05 [0,-0.5,0] rel [0,0,1]; // Элемент
  будет отрисован материалом, определённым в MyMaterial.

  [255,255,0] Beam 0.05 [0,0.5,0] rel [0,0,1]; // Элемент
  будет отрисован жёлтым цветом.

end
```

Сфера

Грамматика:

```
‘Sphere’ PointExpression Expression [Material] [Quality]
‘;’
```

Формальное описание:

```
Sphere Position Diameter [Material] [Quality];
```

Описание сферы начинается со слова **sphere**, за которым обязательно следует точка центра сферы, обязательный диаметр, затем необязательный материал и необязательное качество.

Пример из 11 сфер повышающегося качества:

```
BeamObject '11 сфер разного качества';

color

  Yellow = [255,255,0];

elements

  for i=-5 to 5 do

    sphere [-i, 0, 0] 1 Yellow i+8;

end
```

Балка (круглая или прямоугольная)

Грамматика:

```
‘Beam’ [Material] Expression [Expression] PointExpression
[‘rel’] PointExpression [Expression] [‘Q’] [Material]
[Quality]‘;’
```

Формальное описание:

```
Beam [Material] Size1 [Size2] StartPoint [rel] EndPoint
[Rotation] [Q] [Material] [Quality];
```

Описание балки начинается со слова **beam**, за которым может следовать необязательный материал, затем обязательный размер (по одной или двум осям, если не указан второй размер) и необязательный второй размер, который в случае отсутствия считается равным первому. Второй размер игнорируется, если балка круглого сечения (см. ниже).

Затем идут обязательные две точки, указывающие точку начала и конца балки. Если описание второй точки отделено от описания первой ключевым словом **rel**, то координаты второй являются относительными, т.е. отсчитываются от первой как дельта.

После точек может идти необязательный параметр **Rotation** — угол поворота балки относительно оси, соединяющей точки начала и конца; задаётся в градусах. Необязательное слово из одной буквы **Q** означает, что балка прямоугольного сечения. При его отсутствии балка считается круглой. Затем может быть необязательный материал (заменяет, если был описан в начале).

Последний параметр — качество, который имеет смысл только для круглых балок и по умолчанию равен 6.

Пример, рисующий ступеньки в воздухе и две круглые подпорки:

```
BeamObject 'Ступеньки';

const

  W=1;

elements

  // Ступеньки
```

```
  for i=0 to 10 do
    beam 0.1 0.05 [-W/2, i*0.1, 0.05+i*0.1] rel [W,0,0] Q;
  // Подпорки
  beam 0.1 [-0.4, W, 0] rel [0, 0, 1] 12;
  beam 0.1 [ 0.4, W, 0] rel [0, 0, 1] 12;
end
```

Конус или усечённый конус

Грамматика:

```
‘Cone’ PointExpression [‘rel’] PointExpression Expression
[Expression] [Material] [Quality]‘;’
```

Формальное описание:

```
Cone StartPoint [rel] EndPoint StartDiameter [EndDiameter]
[Material] [Quality];
```

Описание конуса начинается со слова **cone**, за которым идут обязательные точки начала (StartPoint) и конца (EndPoint). Если описание EndPoint отделено от описания StartPoint ключевым словом rel, то координаты EndPoint являются относительными, т.е. отсчитываются от StartPoint как дельта. Затем следует обязательный диаметр (на точке начала) и необязательный диаметр на точке конца, который считается равным нулю, если не указан. После этого может быть указан необязательный материал.

Последний параметр — качество, который по умолчанию равен 6.

Пример простой ёлки:

```
BeamObject 'Ёлка';

var

  d {$Диаметр, м} = 1.25 0.1 20 {$Digits=1; Step=0.5};

  h {$Высота, м} = 1.9 0.1 50 {$Digits=1; Step=0.5};

color

  ch {$Основной} = [0,250,140] {$Order=1};

  c {$Ствол} = [250,150,50] {$Order=2};

elements

  for a=0 to 4 do

    cone [0,0,h*(0.2+a*0.1)] [0,0,h*(0.4+a*0.15)] d*(1-
    a*0.1) ch 12;

  // Ствол
```

```
  cone [0,0,0] [0,0,h] d/10 c 12;
end
```

Сегмент трёхмерной кривой Безье

Грамматика:

```
‘Bezier’ PointExpression PointExpression PointExpression
PointExpression Expression [Expression] [Material]
[Quality]‘;’
```

Формальное описание:

```
Bezier StartPoint ControlPoint1 ControlPoint2 EndPoint
Size1 [Size2] [Material] [Quality];
```

Описание кривой Безье начинается со слова **Bezier**, за которым идут четыре обязательные управляющие точки кривой. Относительные координаты НЕ допускаются.

Затем следует обязательный параметр «размер» (Size1) и необязательный второй размер (Size2), который считается равным Size1, если не указан. После этого может быть необязательный материал и необязательное качество (Quality). Параметр «качество» по факту пока не используется (зарезервирован).

По умолчанию форма сечения фрагмента кривой Безье представляет собой крест «+» с указанной шириной и высотой соответствующих «лент». Если один из размеров равен нулю, то получаем плоскую («-» или «|») ленту, извитую по форме кривой. Если используется один размер или равные размеры, то издалека примитив будет выглядеть как изогнутый «провод» или трубка.

Пример использования можно посмотреть на ручке ведра:

```
BeamObject 'Ведро с водой';

var

  d {$Диаметр, м} = 2 0.5 5 {$Digits=1; Step=0.1};

  h {$Высота, м} = 5 0.5 10 {$Digits=1; Step=0.1};

color

  ch = [150,100,0];

  red= [250,120,120];
```

```
  water = [60,100,255];
elements
  // Ведро
  cone [0,0,h*0.35] [0,0,h*0.45] d*0.2 d*0.3 red 12;
  // Вода в ведре
  beam d*0.25 [0,0,h*0.45] [0,0,h*0.451] water 12;
  // Ручка
  bezier [0,-d*0.14,h*0.45] [0,-d*0.1,h*0.55]
  [0,d*0.1,h*0.55] [0,d*0.14,h*0.45] d*0.02 ch;
end
```

Металлическая конструкция из уголков

Грамматика:

```
'ZPart' Expression PointExpression ['rel'] PointExpression
['rel'] PointExpression ['rel'] PointExpression

Expression [Material] ['E'] ['M']';'
```

Формальное описание:

```
ZPart Size Point0 [rel] Point1 [rel] Point2 [rel] Point3
Step [Material] [E] [M];
```

После ключевого слова **ZPart** идёт обязательный параметр **Size**, определяющий ширину уголка конструкции. Затем четыре точки, описывающие габариты конструкции в порядке, указанном на рисунке. Точки могут быть относительными (каждая от предыдущей), если используется ключевое слово **rel** перед описанием точки.

Далее следует обязательный параметр «шаг заполнения» (Step) — расстояние, указанное на рисунке как Step. Шаг автоматически округляется таким образом, чтобы полностью заполнить конструкцию по длине. Затем может быть указан необязательный материал.

В конце могут стоять разделённые пробелами дополнительные модификаторы: латинские буквы E и/или M, дополняющие конструкцию поперечными уголками:

- E — на концах, как на рисунке 2;
- M — после каждого диагонального уголка, как на рисунке 3.

Можно использовать оба модификатора, чтобы получить поперечины и на концах, и после каждой диагонали.

Рисунок 1.

```
Step

<Вставить рисунок>

Рисунок 2. Результат с модификатором E.

<Вставить рисунок>

Рисунок 3. Результат с модификатором M.
```

<Вставить рисунок>

Пример использования Zpart в объекте «Г-образная ферма»:

```
BeamObject 'Г-образная ферма';

var

  SH {$Длина вылета, м} = 2.0 3.0 10.0 {$Digits=1;
  Order=1};

  H {$Высота стойки, м} = 4.5 2.0 10.0 {$Digits=1; Step-
  p=0.1; Order=2};

  D2SM {$Ширина фермы, см} = 20 10 50 {$Digits=0; Step=5;
  Order=3};

const

  D2 = D2SM/100; // Ширина фермы в метрах.

  dd=D2/2; // Половина ширины.

  ss=D2/10;

elements

  {вертикальная}

  zpart ss [-dd,-dd,0] rel [0, D2,0] rel [0,0,H-D2] rel
  [0,-D2,0] dd E;

  zpart ss [ dd,+dd,0] rel [0,-D2,0] rel [0,0,H-D2] rel
  [0, D2,0] dd E;

  zpart ss [-dd,+dd,0] rel [ D2,0,0] rel [0,0,H-D2] rel [-
  D2,0,0] dd E;

  zpart ss [ dd,-dd,0] rel [-D2,0,0] rel [0,0,H-D2] rel [
  D2,0,0] dd E;

  {горизонтальная}

  zpart ss [-dd, SH+dd,H ] [-dd, SH+dd,H-D2] [-dd, dd-
  D2,H-D2] [-dd, dd-D2,H ] dd E;

  zpart ss [ dd, dd-D2,H ] [ dd, dd-D2,H-D2] [ dd,
  SH+dd,H-D2] [ dd, SH+dd,H ] dd E;
```

```
  zpart ss [-dd, dd-D2,H ] [ dd, dd-D2,H ] [ dd, SH+dd, H]
  [-dd, SH+dd,H ] dd E;

  zpart ss [ dd, SH+dd,H-D2] [-dd, SH+dd,H-D2] [-dd, dd-
  D2,H-D2] [ dd, dd-D2,H-D2] dd E;

end
```

Объект свечения

Грамматика:

```
‘Light’ Expression Expression PointExpression
[Material]‘;’
```

Формальное описание:

```
Light Diameter Offset Position [Material]‘;’
```

Описание объекта начинается со слова **light**, за которым следует обязательный диаметр ореола (Diameter).

Затем указывается обязательное расстояние (Offset), на которое будет отодвигаться ореол в направлении камеры, чтобы оказаться ближе к камере и выглядеть более реалистично. После следует обязательный центр источника свечения (Position). От него будет происходить отодвигание ореола.

Завершается описание необязательным материалом (если не указан, считается белым).

Пример из трёх сфер: одна простая, вторая «светится», а третья «мигает» белым:

```
BeamObject 'Демонстрация свечения';

condition

  A = 0 test;

color

  Yellow = [255,255,0];

elements

  sphere [-1, 0, 0] 0.5 Yellow;
  sphere [ 0, 0, 0] 0.5 Yellow;
  light 2 0.26 [0, 0, 0] Yellow;
  sphere [ 1, 0, 0] 0.5;
  if A light 2 0.26 [1, 0, 0];
end
```

Фигура выдавливания

Грамматика:

```
‘shape’ ShapeName Expression [Expression] PointExpression
[‘rel’] PointExpression [Expression] [Material]‘;’
```

Формальное описание:

```
shape ShapeName Size1 [Size2] StartPoint [rel] EndPoint
[Rotation] [Material];
```

После ключевого слова **shape** идёт название шейпа, определённого в секции shape.

Затем следует один обязательный размер Size1 и один необязательный Size2 (если не указан, считается равным Size1). Размеры нужны для масштабирования шейпа, координаты которого могут быть заданы в условных единицах. Разные размеры позволяют растянуть шейп по одной из осей.

После этого указываются две точки (StartPoint и EndPoint), между которыми должен быть «вытянут» шейп. Шейп вытягивается своей «нулевой» координатой вдоль прямой, проходящей через две указанные точки. EndPoint может быть задана относительно StartPoint, если перед ней используется ключевое слово rel.

После точек может быть необязательный параметр Rotation, позволяющий повернуть фигуру вокруг оси, образованной двумя точками. Угол задаётся в градусах.

Также может быть указан материал, которым должен отрисоваться примитив.

Пример:

```
BeamObject 'Блок защитный пластиковый';

var

  L {$Длина, м} = 3.5 0.5 5.0{$Digits=1;Step=0.1;Order=1};

  H {$Высота, м} = 1 0.8 1.5 {$Digits-
  s=2;Step=0.05;Order=2};

shape
```

```
  Main = 11 [0,1], [0.1,0.9], [0.2,0.25], [0.3,0],
  [0.05,0],

  [0,0.05],

  [-0.05,0], [-0.3,0], [-0.2,0.25], [-0.1,0.9], [0,1];

color

  AColor {$Основной} = [255,255,255];

elements

  shape Main h [0,-L/2,0] rel [0,L,0] AColor;

end
```

Счётчик секунд

Грамматика:

```
'TimeCounter' Expression [Expression] PointExpression
['rel'] PointExpression [Expression] [Material]

'(' Expression',' Expression',' Expression','
Expression')'';'
```

Формальное описание:

```
TimeCounter SizeHorz [SizeVert] BackSidePoint [rel]
FrontSidePoint [Rotation] [Material] (TimeShift,
TotalTime, StartTime, StartValue);
```

После ключевого слова **TimeCounter** идут два размера: первый обязательный, второй нет (если не указан, считается равным первому).

Затем указываются две точки, определяющие местоположение и направление счётчика секунд: первая точка — это центр счётчика; вторая точка определяет, в каком направлении будут «выдавлены» цифры. Может быть задана относительно первой при использовании ключевого слова rel.

Далее в скобках должны идти 4 обязательных числа, разделённых запятыми:

1: TimeShift: сдвиг всего правила вперёд в секундах. Нужен для синхронизации нескольких объектов.

2: TotalTime: время полного цикла, после которого счётчик перезапустится.

3: StartTime: время в секундах относительно начала цикла, когда счётчик должен запуститься.

4: StartValue: начальное значение счётчика, с которого начнётся уменьшение.

Группа элементов

Грамматика:

```
‘begin’ Primitives end‘;’
```

Формальное описание:

```
begin {Набор инструкций}* end;
```

Группа примитивов, начинающаяся с ключевого слова **begin** и заканчивающаяся ключевым словом **end**. Используется, например, для группового отсечения отрисовки по условию или в циклах. Между begin и end могут находиться как примитивы, так и операторы модификации констант и координат, а также циклы.

Пример:

```
BeamObject 'Искусственная неровность' {$X=4; Y=6; Z=3.2;
dX=0.8};

var

  Length {$Длина неровности, м} = 7.5 2 50 {$Digits=1;
  Step=0.5; Order=1};

  Inc {$Поперечный уклон, $%} = 0 {$Digits=1; Step=1;
  Min=-60; Max=60; Order=2};

  Wmm {$Длина секции, мм} = 700 300 1000 {$Digits=0; Step-
  p=50; Order=3};

  Lmm {$Длина хорды, мм} = 600 500 700 {$Digits=0; Step-
  p=50; Order=4};

  Hmm {$Высота гребня, мм} = 50 50 60 {$Digits=0; Step=1;
  Order=5};

const

  L = Lmm/1000;

  H = Hmm/1000;

  Wr= Wmm/1000;// Пользовательская ширина одного элемента
  с двумя вставками, м.

  N = Round(Length/Wr); // Количество элементов.
```

```
  W = Length/N; // Реально используемая далее ширина эле-
  мента.

  In = Inc/1000; // Уклон, используемый в вычислениях.

shape

  Main = 7 [-L,0], [-L*3/5,H/3], [-L/5,H*9/10], [0,H],
  [L/5,H*9/10],[L*3/5,H/3], [L,0];

  Yell = 4 [-L*3/5,H/3], [-L*2/5,H*9/10], [L*2/5,H*9/10],
  [L*3/5,H/3];

coord

  Pos = [-Length/2, 0, -Length/2*In];

color

  Black {$Основной} = [80,80,80] [0,1,0];

  Yellow {$Вставки} = [255,255,0] [1,1,0.5];

const // Все три размера в частях секции.

  Y=0.24; // Ширина жёлтой вставки.

  d=0.15; // Расстояние между двумя вставками в одной сек-
  ции.

  d2=(1-2*Y-d)/2; // Расстояние от края секции до первой
  вставки.

elements

  shape Main Pos rel [Length,0,Length*In] Black;

  for i=0 to N-1 do

  begin

    Pos = Pos + [W*d2, 0, W*d2*In];

    shape Yell Pos rel [W*Y,0,W*Y*In] Yellow;

    Pos = Pos + [W*(Y+d),0,W*(Y+d)*In];

    shape Yell Pos rel [W*Y,0,W*Y*In] Yellow;

    Pos = Pos + [W*(Y+d2),0,W*(Y+d2)*In];
```

```
  end;
end
```