УДК 519.67

МЕТОД КРИВОЛИНЕЙНЫХ КООРДИНАТ В КОМПЬЮТЕРНОЙ ГЕОМЕТРИИ

М.Е. Степанов

В статье, посвящённой компьютерной геометрии, излагается метод, с помощью которого можно эффективно строить компьютерные изображения объектов технического, природного и художественного характера. Он основан на использовании кинематической интерпретации построения кривых в различных системах криволинейных координат. Приведены примеры построения сложных изображений.

The article concerned with computational geometry discusses the method which allows effective computer imaging of technical, natural and artistic objects. This method is based on kinematic interpretation of plotting in various systems of curvilinear coordinates. The examples of complicated imaging are given.

---

1. ВВЕДЕНИЕ

Многие математики и философы науки уверены, что математические объекты в той или иной форме существует реально. Это мнение очень ярко было выражено Шарлем Эрмитом: «Я верю, что числа и функции анализа не являются произвольным созданием нашего разума; я думаю, что они существуют вне нас в силу той же необходимости, как и объекты реального мира, и мы их ворочаем или открываем и изучаем точно так же, как это делают физики, химики или зоологи» [1].

Для математика, работающего в области прикладной математики, такой взгляд на математические объекты представляется в высшей степени естественным. Для него на первый план выходят не доказательства (хотя и они отчасти сохраняют своё значение), а деятельность иного рода. Поскольку важнейшим структурным элементом любой деятельности является целеполагание, разделение математики на чистую и прикладную может быть проведено именно по этому признаку. Математика может рассматриваться либо как цель научного исследования (чистая математика), либо как средство для достижения внематематических целей (прикладная математика) [2].

Поскольку число областей, к которым прилагается математика, чрезвычайно велико, столь же многочисленны по форме и методы математиков-прикладников. Из всего этого многообразия выделим только приёмы работы, сходные по характеру с конструированием. Часто именно так происходит получение новых результатов в сфере компьютерной геометрии. В тех случаях, когда оно направленно на разработку эффективных методов построения компьютерных изображений, функции используются, как конструктивные элементы этих изображений. В частности иногда их графики объединяются, как своеобразные твёрдые фигурные стержни, для получения гладкой кривой, которая обладает теми или иными свойствами. Подобный способ интеллектуальной работы можно было бы назвать предметно-образным манипулировани-

ем математическими объектами. Компьютерный геометр, используя образы линий, «ворочает» кривые подобно предметам и составляет из них изображения.

Один из широко известных методов гладкого сопряжения кривых основан на использовании сплайнов. Гладкая кривая, проходящая через заданные точки плоскости и в каждой из них имеющая заданное направление, конструируется как график кусочно-многочленной функции, состоящей из графиков многочленов третьей степени. Этот подход имеет значительные преимущества перед классическим аппаратом построения интерполяционных многочленов [3]. Рассматриваемый в данной статье метод также предполагает составление гладких кривых из отдельных кусков, гладко сопрягаемых между собой.

Ещё одним активно используемым способом построения гладких кривых является метод Пьера Безье. Изначально его идея была основана на геометрических и кинематических соображениях и лишь в дальнейшем получила алгебраическое истолкование, связанное с многочленами Бернштейна [4]. Предлагаемый нами метод также основывается на идеях геометрического и кинематического характера, но может быть истолкован и аналитически.

Метод криволинейных координат, к описанию которого мы приступаем, основан на соображениях, изложенных автором в статье [5]. Вкратце их суть такова.

– Многие геометрические объекты естественным образом истолковываются как траектории движущихся точек или линий. Это связано с тем, что кинематика «является геометрией движущихся материальных объектов» [6]. Именно по этой причине использование механического движения есть важнейший путь к пониманию внутренних закономерностей формы объекта.

– Для анализа формы геометрического объекта может быть использован метод сложных движений, приводящий к разработке алгоритма построения изображения. Этот метод позволяет строить цепочку математических вычислений, отталкиваясь от последовательности порождающих объект простых движений.

– Для разработки алгоритма первоначально проводится анализ формы объекта, направленный на поиск сложного движения, приводящего к построению объекта как некоторой траектории. Кроме того, соответствующее сложное движение разлагается в совокупность простых движений.

– Каждое простое движение является движением пространственно-подобного элемента (плоскости, прямой, точки) в пространстве большей размерности.

– Движение каждого такого элемента описывается аффинным или евклидовым преобразованием.

– Алгоритм возникает как цепочка последовательных вычислений координат движущихся точек. Соответствующие координаты изменяются в результате простых движений, а в совокупности эти простые движения создают целостное сложное движение.

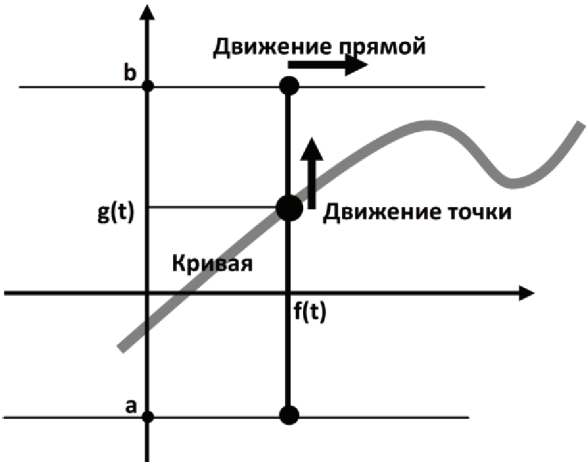
– Перейдём к описанию метода криволинейных координат.

2. КИНЕМАТИЧЕСКАЯ ИНТЕРПРЕТАЦИЯ ПОСТРОЕНИЯ КРИВЫХ В ДЕКАРТОВЫХ КООРДИНАТАХ

Пусть на плоскости задана декартова система координат. Рассмотрим плоскую кривую, которая описывается с помощью двух параметрических уравнений $x = f(t)$ и $y = g(t)$. Будем считать параметр t временем. Тогда построение кривой можно провести, соединив в одно сложное движение следующие простые движения.

1. Перемещение по плоскости прямой, всегда остающейся параллельной оси ординат, так, что абсциссы всех точек, лежащих на ней, в момент времени t равны $f(t)$.
2. Перемещение по упомянутой прямой точки так, что её ордината в момент времени t равна $g(t)$.

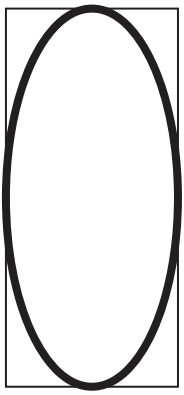
Очевидно, что в итоге точка вычертит соответствующую кривую. Естественно, что при практических вычислениях достаточно позаботиться о перемещении не всей прямой, а только отрезка достаточной длины, лежащего на ней. Для описания движения отрезка достаточно умения вычислять координаты его концов. Положение же движущейся по прямой точки, иногда удобно задавать не относительно системы координат, а относительно концов этого отрезка.



Продемонстрируем, как выглядит соответствующий алгоритм с помощью программы, написанной на языке Q-бейсик. (Этот язык минимально отвлекает тонкостями программирования от структуры алгоритмов, и, кроме того, близок к математической символике).

Пример 1. Построить эллипс, вписанный в прямоугольник, задаваемый неравенствами $a \le y \le b$ и $c \le x \le d$.

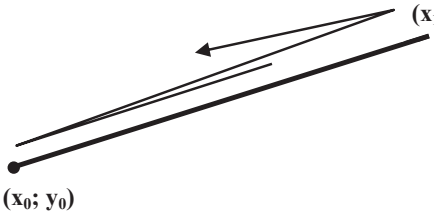
Используем тот факт, что эллипс задаётся параметрическими уравнениями, в которых одна из функций является косинусом, а вторая – синусом.



```
pi = 3.14159                    Задание числа пи
a = 100                         Параметры,
b = 400                             задающие
c = 200                                 положение
d = 300                                     прямоугольника
line (c, a) – (d, b), , b       Прорисовка прямоугольника
x0 = (c + d) / 2                Абсцисса центра прямоугольника
r = (d – c ) / 2                Амплитуда колебания по горизонтали
for t = 0 to 1 step .001        Цикл, изменяющий время
u = 2 * pi * t                  Изменение угла от нуля до двух пи
x = x0 + r * cos(u)             Колебание вертикального отрезка
y = a + (1 + sin(u)) * (b – a) / 2   Колебание точки по отрезку
pset (x, y)                     Установка точки на экране
next t                          Конец цикла
```

Конечно, можно предположить, что переход отязыка координат к языку движений в той форме, в которой он сделан к данному моменту, ничего нового не дал. На самом деле это не так, поскольку язык движений позволяет перенести метод на новые геометрические конфигурации.

Пример 2. На плоскости заданы координаты вершин произвольного четырёх угольника. Вписать в него овал, касающийся всех сторон четырёхугольника.



Для выполнения поставленной задачи необходимо связать с четырёхугольником движение отрезка, а с отрезком – движение точки по нему. Сделать это достаточно просто. Основой решения является то, что гармонические колебания точки по отрезку определяются про-

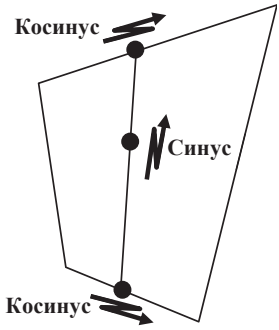
стой формулой, аналогичной той, которая использована в предыдущей программе (третья строка от конца).

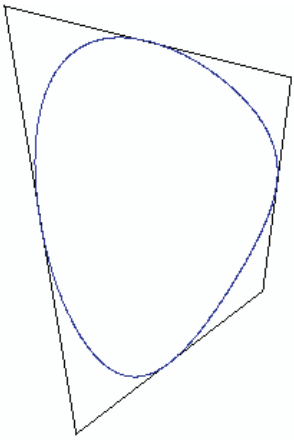
При колебании точки по наклонному отрезку по соответствующей формуле меняются и абсцисса, и ордината:

$x = x_0 + f(t)\cdot(x_1 - x_0)$,

$y = y_0 + f(t)\cdot(y_1 - y_0)$, где f(t) – либо синус, либо косинус.

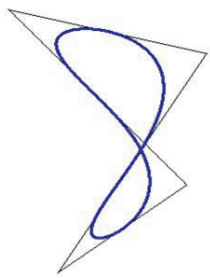
Выберем две противоположных стороны четырёхугольника и заставим две точки синхронно колебаться по ним. Эти точки являются концами движущегося отрезка, по которому со сдвигом по фазе колеблется точка, заметающая овал.





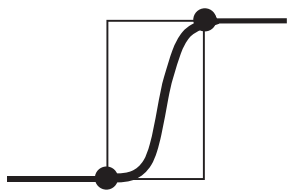
```
screen 12
pi = 3.14159
x0 = 100: y0 = 100                    Координаты
x1 = 300: y1 = 150                        вершин
x2 = 280: y2 = 200                            четырёхугольника
x3 = 150: y3 = 300
line (x0, y0) – (x1, y1)              Прорисовка
line (x1, y1) – (x2, y2)                  четырёхугольника
line (x2, y2) – (x3, y3)
line (x3, y3) – (x0, y0)
for t = 0 to 1 step .001              Цикл, изменяющий время
u = 2 * pi * t
xa = x0 + (1+ cos(u))*(x1 – x0)/2     Колебание одного
ya = y0 + (1 + cos(u)) * (y1 – y0) / 2    конца отрезка
xb = x3 + (1+ cos(u))*(x2 – x3)/2     Колебание второго
yb = y3 + (1 + cos(u)) * (y2 – y3) / 2    конца отрезка
x = xa + (1+ sin(u))*(xb – xa)/2      Колебание точки
y = ya + (1 + sin(u)) * (yb – ya) / 2     по отрезку
pset (x, y)                           Установка точки на экране
next t                                Конец цикла
```

Отметим следующее обстоятельство. Координаты вершин четырёхугольника были выбраны так, чтобы его стороны не имели самопересечений. Однако алгоритм не перестаёт работать и при произвольном выборе координат. Например, если поменять координаты третьей и четвёртой вершин, две стороны четырёхугольника пересекутся, а овал свернётся в восьмёрку.



Прежде чем рассматривать новые способы использования нашей кинематической модели, вернёмся к исходному её варианту, связанному с декартовой системой координат. Очевидно, что общая скорость точки, вычерчивающей кривую, имеет две составляющие – горизонтальную, равную производной по времени функции $x = f(t)$, и вертикальную, равную производной по времени функции $y = g(t)$.

Если $g'(t_0) = 0$, то скорость точки в момент времени t_0 направлена горизонтально, а, следовательно, касательная к кривой в точке $x_0 = f(t_0)$ имеет горизонтальное направление. Это обстоятельство позволяет выполнить гладкое сопряжение двух прямых, параллельных оси абсцисс, с помощью функции $y = g(t)$, для которой $g'(0) = 0$ и $g'(1) = 0$.



Пример 3. Получить гладкое сопряжение двух прямых, параллельных оси абсцисс.

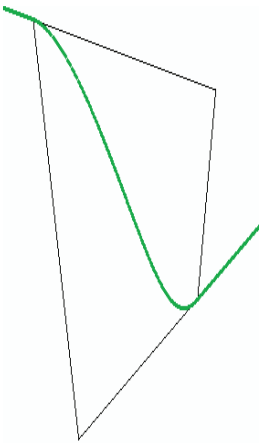
Уточним постановку задачи. Мы будем сопрягать лучи прямых, на которых лежат горизонтальные стороны прямоугольника из примера 1 ($a \le y \le b$ и $c \le x \le d$). В этих обозначе-

ниях лучи могут быть описаны следующим образом. Первый луч: $y = a, x \le c$, второй луч: $y = b, x \ge d$. Для сопряжения лучей естественно использовать синусоиду.

Пример 4. Получить гладкое сопряжение двух прямых, имеющих общее положение. Подобно тому, как кинематическая схема из примера 1 в примере 2 была перенесена на четырёхугольник общего вида, и сейчас результаты примера 3 можно также перенести на четырёхугольник.

```
screen 12
pi = 3.14159
a = 400
b = 100
c = 200
d = 300
line (c, a) – (d, b), 3, bf
line (c, a) – (0, a)                    Первый луч
line (d, b) – (640, b)                  Второй луч
for t = 0 to 1 step .001
u = – pi / 2 + pi * t
x = c + t * (d – c)                     Построение
y = a + (1 + sin(u)) * (b – a ) / 2          синусоиды,
circle (x, y), 1                                сопрягающей лучи
```

Практически в примере 4 решена задача, аналогичная задаче построения сплайна: провести гладкую кривую, проходящую через две заданные точки плоскости и в каждой из них имеющую заданное направление. Однако в отличии от сплайна, связанного с системой координат, наша гладкая кривая ориентирована на соответствующие направления.



```
screen 12: pi = 3.14159
x0 = 200: y0 = 100: x1 = 400: y1 = 150
x2 = 380: y2 = 300: x3 = 250: y3 = 400
line (x0, y0) – (x1, y1): line (x1, y1) – (x2, y2)
line (x2, y2) – (x3, y3): line (x3, y3) – (x0, y0)
for t = – 1 to 0 step .01                    Первый луч
x = x0 + t * (x1 – x0): y = y0 + t * (y1 – y0)
circle (x, y), 1, 3: next t
for t = – 1 to 0 step .01                    Второй луч
x = x2 + t * (x3 – x2): y = y2 + t * (y3 – y2)
circle (x, y), 1, 3: next t
for t = 0 to 1 step .001: u = – pi / 2 + pi * t    Сопрягающая
xa = x0 + t * (x1 – x0): ya = y0 + t * (y1 – y0)          кривая
xb = x3 + t * (x2 – x3): yb = y3 + t * (y2 – y3)
x = xa + (1 + sin(u)) * (xb – xa) / 2
y = ya + (1 + sin(u)) * (yb – ya) / 2: circle (x, y), 1, 3: next t
```

Эти направления в базовых точках задаются векторами. Выбрав из каких-либо соображений длины двух отрезков, исходящих из двух точек в направлении векторов, мы придём к четырёхугольнику, как в предыдущем примере.

Пример 5. Провести через две точки гладкую кривую, имеющую в них заданные направления.

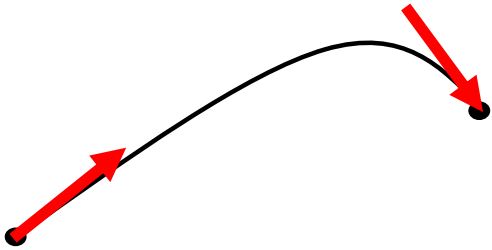
Как мы установили, можно просто повторить программу из предыдущего примера. Однако мы дадим решение, отличающееся от предыдущего рядом деталей. Прежде всего, вместо синусоиды в качестве функции g(x) используем сплайн.

Итак, $g(x) = ax^3 + bx^2 + cx + d$, а её производная равна $3ax^2 + 2bx + c$. Наложим на g(x) следующие условия: $g(0) = 0, g(1) = 1, g'(0) = g'(1) = 0$.

Их можно задать с помощью следующей линейной системы уравнений

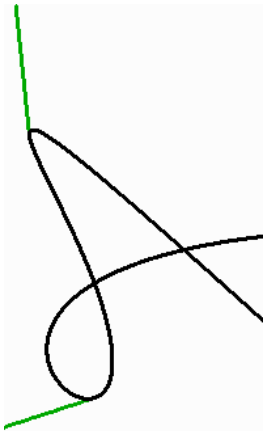
$$\begin{cases} d = 0 \\ a + b + c + d = 1 \\ c = 0 \\ 3a + 2b + c = 0. \end{cases}$$

Поскольку $a = -2,\ b = 3,\ c = d = 0,\ g(x) = -2x^3 + 3x^2$.



Пусть базовые точки имеют координаты $(x_0; y_0)$ и $(x_1; y_1)$. Для указания направлений зададим два угла u_0 и u_1, которые в свою очередь определят два единичных вектора $(\cos u_0; \sin u_0)$ и $(\cos u_1; \sin u_1)$. В качестве длин соответствующих отрезков примем некоторое число r.

Чтобы лучше представить общую картину сопряжения, отобразим в программе не только сопрягающий кусок кривой, но и её общее поведение в области сопряжения.



```
screen 12: pi = 3.14159
x0 = 300: y0 = 200: x1 = 350: y1 = 400          Концы отрезков
r = 100: u0 = 7 * pi / 15: u1 = .9 * pi         Углы наклона отрезков
for t = –1 to 0 step .01
x = x0 + t * r * cos(u0): y = y0 + t * r * sin(u0)   Первый отрезок
circle (x, y), 1, 3: next t
for t = 0 to 1 step .01
x = x1 + t * r * cos(u1): y = y1 + t * r * sin(u1)   Второй отрезок
circle (x, y), 1, 3: next t
for t = –1 to 3 step .001
xa = x0 + t * r * cos(u0)                       Сопрягающая
ya = y0 + t * r * sin(u0)                            кривая
xb = x1 + (t – 1) * r * cos(u1)                        на основе
yb = y1 + (t – 1) * r * sin(u1)                          сплайна
x = xa + (–2 * t ^ 3 + 3 * t ^ 2) * (xb – xa)
y = ya + (–2 * t ^ 3 + 3 * t ^ 2) * (yb – ya)
circle (x, y), 1, 2
next t
```

Практически в примере 5 показан приём проведения гладкой кривой через заданные точки плоскости в заданных направлениях. Сформулируем общее правило построения таких кривых.

1. Кривая в целом состоит из кусков, соединяющих соседние точки (точно также как при построении сплайнов).
2. Если это не указано заранее, перед соединением двух соседних точек A и B, нужно выбрать начальную точку (пусть это будет точка A) и конечную (B).
3. Пусть вектора **a** и **b** указывают направление кривой в точках A и B соответственно. Построим два отрезка произвольной длины AX и YB такие, что вектор AP = $\lambda_1 \cdot \mathbf{a}$ (отрезок исходит из точки A) и вектор QB = $\lambda_2 \cdot b$ (отрезок входит в точку B), где $\lambda_1 > 0$ и $\lambda_2 > 0$.
4. Заставим синхронно двигаться по отрезкам две точки M и N. Движение задаётся формулами

$$\begin{cases} x_M = x_A + t \cdot (x_P - x_A), \\ y_M = y_A + t \cdot (y_P - y_A), \end{cases}$$

$$\begin{cases} x_N = x_Q + t \cdot (x_B - x_Q), \\ y_N = y_Q + t \cdot (y_B - y_Q), \end{cases}$$

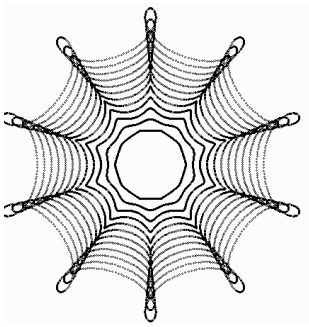
где время t меняется от 0 до 1.

5. По отрезку MN заставим двигаться точку, вычерчивающую кривую. Её координаты вычисляются по формулам

$$\begin{cases} x = x_M + g(t)\cdot(x_N - x_M), \\ y = y_M + g(t)\cdot(y_N - y_M). \end{cases}$$

На функцию $g(t)$ наложены условия: $g(0) = 0, g(1) = 1, g'(0) = g'(1) = 0$.

Пример 6. Используя соответствующий алгоритм, построить семейство замкнутых кривых, проходящих через вершины правильных n-угольников и касающихся описанных окружностей.

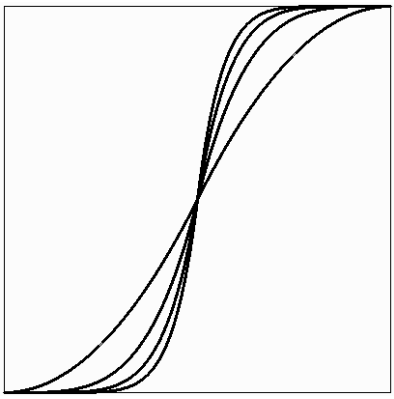


```
screen 12: pi = 3.14159
n = 10: du = 2 * pi/n                    Число вершин n-угольника
xc = 320: yc = 240                       Координаты его центра
r = 5                                    Длина базовых отрезков
for rc = 50 to 160 step 10               Радиус описанной окружности
for i = 0 to n
x0 = xc + rc * cos(i * du)               Вычисление
y0 = yc + rc * sin(i * du)                   координат
x1 = xc + rc * cos(i * du + du)              двух вершин
y1 = yc + rc * sin(i * du + du)                  n-угольника
u0 = i * du + pi/2: u1 = u0 + du         Направление касательных
for t = 0 to 1 step .01
xa = x0 + t * r * cos(u0)                Построение
ya = y0 + t * r * sin(u0)                    сопрягающей
xb = x1 + (t – 1) * r * cos(u1)                  кривой
yb = y1 + (t – 1) * r * sin(u1)
x = xa + (-2 * t^3 + 3 * t^2) * (xb - xa)
y = ya + (-2 * t^3 + 3 * t^2) * (yb - ya)  Увеличение отношения r к rc
circle (x, y), 1, 2                          приводит к изменению
next t, i: r = r + 40: next rc                   формы кривой
```

В комментариях к программе из предыдущего примера, указано, что изменение длины сторон четырёхугольника, определяющего характер сопряжения, меняется форма сопрягающей кривой. Ещё одним, и весьма естественным, способом изменения формы кривой является выбор функции g(t). Отметим, что при эмпирическом подборе формы конструируемой гладкой кривой, каждая из таких функций играет роль своеобразного лекала. В следующем примере мы рассмотрим возможности использования лекал, составленных из кусков многочленов произвольных степеней, что позволяет изменять кривизну сопрягающей кривой.

Пример 7. Построить на отрезке [0; 1] гладкую функцию $g(t)$, на которую наложены условия: $g(0) = 0, g(1) = 1, g'(0) = g'(1) = 0$, и составленную из кусков двух многочленов n-й степени.

Искомая кривая может быть составлена из графика исходной функции y = ctn и графика функции y = 1 – c(1 – t)n, которая получена центрально симметричным отражением исходного графика относительно точки (0,5; 0,5). Коэффициент c следует выбрать, исходя из условия g(0,5) = 0,5. Очевидно, что он равен 2n–1. Гладкость функции обеспечивается тем обстоятельством, что центральная симметрия сохраняет параллельность прямых.



```
screen 12                                  
x0 = 120                                   Местоположение начала
y0 = 440                                         координат на экране
a = 400                                    Масштаб
line (x0, y0)-(x0 + a, y0 – a), , b        Единичный квадрат
for n = 2 to 8 step 2                      Изменение степени, n = 2, 4, 6, 8
for t = 0 to 1 step .0001                  Изменение аргумента
if t < .5 then g = t^n * 2^(n - 1)         Вычисление значений
else g = 1 – (1 – t) ^ n * 2^(n – 1)             кусочной функции
x = x0 + t * a                             Переход к экранным
y = y0 – g * a                                   координатам
circle (x, y), 1                           Установка точки на графике
next t, n
```

Пример 8. Построить на отрезке [0; 1] гладкую функцию *g(t),* на которую наложены условия: $g(0) = 0, g(1) = 1, g'(0) = g'(1) = 0$, и которая составлена из кусков (полусплайнов) двух многочленов разных степеней (n и m), что позволит получить асимметричную кривую. Кроме того, сопряжение графиков должно происходить в точке t_0, лежащей на отрезке [0; 1].

Искомая кривая может быть составлена из графика функции $y = c_1 t^n$ и графика функции $y = 1 - c_2(1 - t)^m$. Условия гладкого сопряжения задаётся двумя уравнениями, выражающими равенство функций и их производных в точке t_0:

$$c_1 \cdot t_0^{\,n} = 1 - c_2 \cdot (1 - t_0)^m$$
$$n \cdot c_1 \cdot t_0^{\,n-1} = m \cdot c_2 \cdot (1 - t)^{m-1}$$

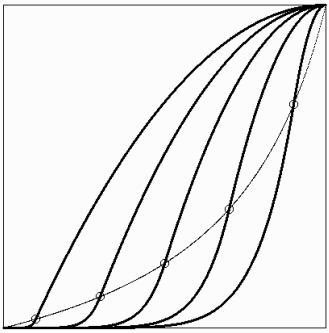
Решение этой системы линейных уравнений даёт следующие выражения для коэффициентов c_1 и c_2:

$$c_1 = \frac{m}{(n - n \cdot t_0 + m \cdot t_0) \cdot t_0^{n-1}} \text{ и } c_2 = \frac{1 - c_1 \cdot t_0^n}{(1 - t_0)^m}$$

Используя полученное выражение для c_1, можно вычислить ординату точки сопряжения: $g(t_0) = \frac{m \cdot t_0^n}{(n - n \cdot t_0 + m \cdot t_0) \cdot t_0^{n-1}} = \frac{m \cdot t_0}{(n - n \cdot t_0 + m \cdot t_0)}$. Покажем, что функция $v(t) = \frac{m \cdot t}{(n - n \cdot t + m \cdot t)}$ непрерывна на отрезке [0; 1].

Если $n = m$, то $v(t_0) = t_0$. Если же n и m не равны между собой, то мы имеем дело с гиперболой. Кроме того, знаменатель соответствующего выражения обращается в ноль при $t_0 = \frac{n}{n - m}$. Если при этом n меньше m, то t_0 отрицательно. Если же n больше m, то t_0 больше единицы. Из этого следует, что единственная точка разрыва функции $v(t)$ лежит вне отрезка [0; 1].

Кроме непрерывности функции v(t) на отрезке [0; 1], можно сделать вывод и о её монотонности на нём. Но $v(0) = 0$ и $v(1) = 1$, значит $v(t)$ возрастает, а точка сопряжения всегда попадает в единичный квадрат. Следовательно, для любых n и m гладкое сопряжение возможно при любом значении t_0.



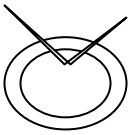
```
screen 12
x0 = 120: y0 = 440: a = 400
n = 8: m = 2                                        Степени многочленов
line (x0, y0) – (x0 + a, y0 – a), 0, b
for t0 = 0 to 1 step .001                           Построение
g0 = m * t0 / (n – n * t0 + m * t0)                   геометрического
x = x0 + t0 * a: y = y0 – g0 * a: pset (x, y)           места точек
next t0                                                   сопряжения
for t0 = .1 to .91 step .2                          Перебор точек сопряж.
c1 = m / ((n – n * t0 + m * t0) * t0^(n – 1))       Вычисление
c2 = (1 – c1 * t0 ^ n) / (1 – t0) ^ m                 коэффициентов
g0 = c1 * t0 ^ n: x = x0 + t0 * a: y = y0 – g0 * a  Выделение точки
circle (x, y), 5                                      сопряжения
for t = 0 to 1 step .0001                           Построение
if t < t0 then g = c1*t^n else g = 1 – c2*(1 – t)^m   гладкой
x = x0 + t * a: y = y0 - g * a: circle (x, y), 1        кривой, состоящей
next t, t0                                                из двух кусков
```

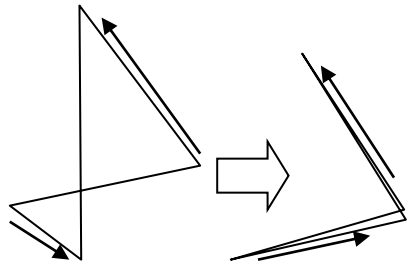
3. ПРИМЕНЕНИЕ КИНЕМАТИЧЕСКОГО МЕТОДА ДЛЯ ПОСТРОЕНИЯ ИЗОБРАЖЕНИЙ

Описанный выше алгоритм, а также соответствующие сопрягающие функции, можно использовать в работе дизайнера при создании простых, но выразительных изображений.

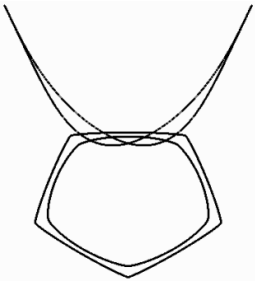
Пример 9. Разработать стилизованное изображение зодиакального знака Тельца.

За основу нами взята стилизация символа из [7] – пятиугольник, изображающий голову, и дуга, изображающая рога. Базовым изображением для нас послужат две концентрических окружности (вернее эллипса), в которые будут вписаны два криволинейных пятиугольника тем же способом, использованным в примере 6, а также вырожденный четырёхугольник с самопересечениями, у которого две вершины сливаются в одну.





В этот четырёхугольник вписываются два сплайна из примера 8, для одного из которых *n = 3, m = 5*, а для другого наоборот *n = 5, m = 3*.



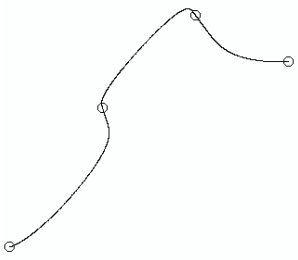
```
screen 12: pi = 3.14159
nstor = 5: du = 2 * pi / nstor                      Число сторон
xc = 320: yc = 300: k = .8: r = 20: rc = 140        Параметры эллипсов
for j = 1 to 2: for i = 0 to nstor                  Построение
x0 = xc + rc * cos(i * du + pi / 2)                       пятиугольников
y0 = yc + k * rc * sin(i * du + pi / 2)
x1 = xc + rc * cos(i * du + du + pi / 2)
y1 = yc + k * rc * sin(i * du + du + pi / 2)
u0 = i * du + pi / 2: u1 = u0 + du
for t = 0 to 1 step .01
xa = x0 + t * k * r * cos(u0 + pi / 2)
ya = y0 + t * k * r * sin(u0 + pi / 2)
xb = x1 + (t – 1) * k * r * cos(u1 + pi / 2)
```

```
yb = y1 + (t – 1) * k * r * sin(u1 + pi / 2)
x = xa + (–2 * t ^ 3 + 3 * t ^ 2) * (xb – xa)
y = ya + (–2 * t ^ 3 + 3 * t ^ 2) * (yb – ya)
circle (x, y), 1, 2: next t, i
r = 50: rc = 120: next j                          Переход ко 2-му эллипсу
n = 3: m = 5: t0 = 1 / 3                          Параметры сплайна
c1 = m / ((n – n * t0 + m * t0) * t0 ^ (n – 1))
c2 = (1 – c1 * t0 ^ n) / (1 – t0) ^ m
v1 = xc + 250 * cos(3 * pi / 4)                   Вершины
w1 = yc - 250 * sin(3 * pi / 4) – 100                   четырёхугольника
v2 = xc + 250 * cos(pi / 4)
w2 = yc – 250 * sin(pi / 4) – 100
for j = 1 to 2: for t = 0 to 1 step .001          Построение
xa = v1 + t * (x0 – v1): ya = w1 + t * (y0 – w1)        Сплайна
xb = x0 + t * (v2 – x0): yb = y0 + t * (w2 – y0)
if t < t0 then g = c1 * t ^ n else g = 1 – c2 * (1 – t) ^ m
x = xa + g * (xb – xa): y = ya + g * (yb – ya)
circle (x, y), 1, 2: next t
p = v1: q = w1: v1 = v2: w1 = w2: v2 = p: w2 = q: next
```

Пример 10. Разработать программу для построения одним росчерком линий, определяемых упорядоченным набором точек (узлов). Вне узлов линия должна быть гладкой, а в узлах либо гладкой, либо имеющей излом под заданным углом.

В данном примере предполагается обобщение метода из примера 6. Кривая определяется упорядоченным набором узлов, лежащих на плоскости. В каждом узле заданы углы наклона входящей и исходящей касательных, определяющих либо направление, либо излом кривой. Кроме того, в каждом узле заданы длины входящей и исходящей стороны четырёх-угольника, а также параметры сплайна сопрягающего данный узел со следующим.

В предлагаемой программе по четырём узлам строится кривая, форма которой выбрана произвольным образом.



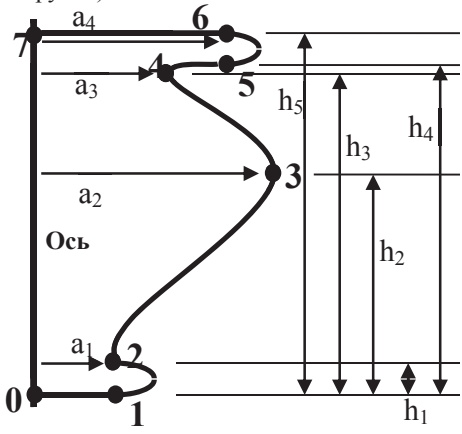
```
screen 12: pi = 4 * atn(1): k = 3                 Число сплайнов
dim x(k), y(k), uin(k), uout(k), uiz(k)           Массивы
dim rin(k), rout(k), n(k), m(k), t0(k)                  параметров
data 100, 200, 300, 400: rem абсциссы узлов
data 300, 150, 50, 100: rem ординаты узлов
data 0, .3, .85, 0: rem углы входящих касательных
data 0, 0, 0, 0: rem углы излома
data 50, 60, 40, 50: rem длины входящих сторон
data 70, 30, 80, 40: rem длины исходящих сторон
data 2, 4, 6, 2: rem степени первого полусплайна
data 7, 5, 3, 2: rem степени второго полусплайна
data .3, .5, .7, .2: rem точка сопряжения полусплайнов
for i = 0 to k: read x(i): next i                 Считывание и
for i = 0 to k: read y(i): circle (x(i), y(i)), 5: next i
for i = 0 to k: read uin(i): uin(i) = 2 * pi * uin(i): next i
for i = 0 to k: read uiz(i): uiz(i) = 2 * pi * uiz(i)   дополнительная
uout(i) = uin(i) + uiz(i): next i
for i = 0 to k: read rin(i): next i
for i = 0 to k: read rout(i): next i               коррекция
for i = 0 to k: read n(i): next i
for i = 0 to k: read m(i): next i
for i = 0 to k: read t0(i): next i                 параметров
```

```
for i = 0 to k – 1                                                    Построение
c1 = m(i) / ((n(i) – n(i) * t0(i) + m(i) * t0(i)) * t0(i)^(n(i) – 1))   сплайнов,
c2 = (1 – c1 * t0(i) ^ n(i)) / ((1 – t0(i)) ^ m(i))                   составляющих
v1 = x(i) + rout(i) * cos(uout(i))                                    единую
w1 = y(i) – rout(i) * sin(uout(i))                                    кривую
v2 = x(i + 1) – rin(i + 1) * cos(uin(i + 1))
w2 = y(i + 1) + rin(i + 1) * sin(uin(i + 1))
for t = 0 to 1 step .001
xa = x(i) + t * (v1 – x(i)): ya = y(i) + t * (w1 – y(i))
xb = v2 + t * (x(i + 1) – v2): yb = w2 + t * (y(i + 1) – w2)
if t < t0(i) then g = c1 * t ^ n(i) else g = 1 – c2 * (1 – t) ^ m(i)
x = xa + g * (xb – xa): y = ya + g * (yb – ya): pset (x, y): next t,
```

Пример 11. С помощью разработанного алгоритма построить эстетически значимое изображение.


стамнос

В качестве образца изображения выберем один из видов античных ваз – стамнос [8]. Будем ориентироваться на его очертания и пропорции, но без буквального воспроизведения (в частности у нашего изображения отсутствуют ручки).



Естественным приёмом при построении соответствующего изображения является использование зеркальной симметрии. По этой причине нас будет интересовать только правая половина контура, очерчивающего изображение. При определении его формы на роль узлов кроме концевых точек (0, 7) будем брать точки, в которых достигаются локальные экстремумы абсциссы (3, 4), точки перегиба (5), излома (2) и точки заведомой смены характера кривой (1, 6).

Что касается программы, то в целом она совпадает с программой из предыдущего примера. Отличия связаны с набором параметров и процедурой формирования координат узлов. Именно эту часть программы мы и приводим. Кроме того, помимо команды установки точки ***pset (x, y),*** вводится и команда установки симметричной точки ***pset (2 * x(0) – x, y)***.

```
k = 7                                                  Число сплайнов
a1 = 90: a2 = 205: a3 = 105: a4 = 135                  Параметры
h1 = 10: h2 = 210: h3 = 300: h4 = 320: h5 = 330          вазы
dim x(k), y(k): rem координаты узлов                   Формируются с помощью
                                                       параметров a1 – a4 и h1 – h5
                                                       Считываются из хранилищ
dim uin(k), uout(k), uiz(k), rin(k), rout(k), n(k), m(k), t0(k)   типа data
                                                       Формирование
x(0) = 320: x(1) = x(0) + a1: x(2) = x(1): x(3) = x(0) + a2          координат
x(4) = x(0) + a3: x(5) = x(0) + a4: x(6) = x(5): x(7) = x(0)           узлов
y(0) = 410: y(1) = y(0): y(2) = y(0) – h1: y(3) = y(0) – h2
y(4) = y(0) – h3: y(5) = y(0) – h4: y(6) = y(0) – h5: y(7) = y(6)   Углы входящих касатель-
data 0, 0, .5, .25, .25, 0, .5, .5                                    ных, а далее все парамет-
```

Отметим, что меняя параметры $a_1 – a_4$ и $h_1 – h_5$, можно варьировать форму вазы весьма значительно. Это обстоятельство приводит нас к важному вопросу, который требует специального рассмотрения. Здесь же мы только затронем его.

Речь идёт о методах математического описания формы. «Есть ярко выраженная потребность в строго разработанной науке морфологии, которая бы занималась принципами, лежащими в основе описания форм. Математика описания форм, как возможная область исследования, до настоящего времени строго не развивалась, или даже не была признана в достаточной степени» [9]. Несомненно, для компьютерной геометрии этот вопрос является центральным.

В дальнейшем мы ограничимся тем, что на основе приёма, использованного в предыдущем примере, сформулируем эмпирическое правило описания формы плоских кривых. Однако сначала нами будут изложены некоторые соображения общего характера, позволяющие оценить значение затронутой проблемы. При этом мы обратимся к некоторым понятиям и идеям искусствоведения и биологии.

Во многих культурах при создании изображений художественного и религиозного характера используется канон. Канон – «совокупность твёрдо установленных правил, определяющих в художественном произведении нормы композиции и колорита, систему пропорций, либо иконографию данного типа изображения» [10]. Для ряда стран и эпох, характеризующихся высоким уровнем развития изобразительного искусства, типично стремление осмыслить сущность канона и обосновать его с помощью тех или иных математических принципов. «Для античности и Возрождения характерны попытки рационалистическим путём найти идеальную закономерность в пропорциях человеческого тела и вывести неизменные, математически обоснованные правила построения человеческой фигуры» [10].

Несомненно, что истоком этой тенденции стало древнеегипетское искусство. «Подобно тому, как сущность философской концепции определяется тем, что принимается ею за первооснову, так же и канон как специфическая форма художественного мышления должен был иметь свою первооснову, которая бы организовывала и уравновешивала все составляющие его компоненты. И здесь в качестве формообразующей структуры канона выступает система пропорциональных соотношений» [11].

Кроме понятия канона мы затронем ещё и одно понятие – архетип (от греческих arche – начало и typos – образ) [12]. Термин «архетип» может быть успешно заменён русским словом «прообраз». В биологии архетип – это «представление о первичном типе, прототипе, строения скелета всех позвоночных животных, выдвинутое Р. Оуэном (1847). Теория архетипов основывается на сопоставлении общих признаков, свойственных скелету различных позвоночных, и создаёт отвлечённый образ, идеальный тип скелета, не реализованный полностью ни в одном из вымерших или ныне живущих животных… Дарвин переосмыслил понятие архетипа, представлял его не как абстрактный прототип, а как реально существовавшую некогда прародительскую форму» [10].

При описании морфологии плоских кривых мы будем отталкиваться от понятий архетипа, канона и пропорции, толкуя их весьма свободно. Мы решаем задачу построения кривой, обладающей определёнными свойствами. (Более правильно было бы говорить о построении целого класса кривых, одновременно имеющих и узнаваемую форму, и обладающих индивидуальными чертами).

Роль архетипа кривой будут играть либо набор узлов, заданный непосредственно, либо некоторая конкретная кривая, с выделенными на ней узлами. Обычно мы будем задавать кривую-архетип параметрически.

Переход от архетипа к канону, то есть к подвижной кривой, обретающей нужную форму и, тем самым, наполняющейся смыслом, осуществляется с помощью векторов смещающих узлы в новое положение. Наконец, роль пропорций играют параметры кривой одного росчерка (углы наклона входящей и исходящей касательных, длины входящей и исходящей стороны четырёхугольника, а также параметры сплайна сопрягающего данный узел со следующим).

Если обратиться к построению вазы, роль архетипа играет вертикальный отрезок, относительно которого указаны смещения узлов канона. Канон определяет очертания вазы. Но конкретную форму ваза приобретает только после выбора соответствующих пропорций.

Отметим, что предлагаемый подход имеет связь с ещё одной биологической теорией. «В последней главе своего фундаментального труда «О росте и форме» д'Арси Томпсон выдвинул гипотезу, которую он рассматривает как идею общего аналитического подхода к изучению формы у близкородственных видов. Эта гипотеза формулируется весьма просто и в основном сводится к следующему: если принять форму какого-то данного организма за эталон и отнести эту форму к некоторой декартовой прямоугольной системе координат, то форму всякого, организма, достаточно близкого к начальному, можно рассматривать как результат непрерывной деформации исходной координатной системы» [13].

Предлагаемый нами подход как раз и направлен не на построение одной, пусть и обладающей исключительными свойствами, кривой. Речь идёт о создании некого «морфологического пространства», определяемого набором многочисленных, но геометрически обусловленных параметров. Их изменение обеспечивает возможность целенаправленной непрерывной деформации исходного изображения.

Именно с этим обстоятельством связано введение варианта, при котором архетип задаётся в виде кривой. Ведь непосредственное задание узлов, казалось бы, даёт тот же результат без лишних вычислений. Однако кривая может послужить более удобным средством построения и использования деформаций для получения новых форм. К этому вопросу мы обратимся позднее.

Окончательно сформулируем правила построения плоских кривых, обладающих определёнными морфологическими свойствами.

1. Выбирается архетип, представляющий собой либо набор узлов, указанных непосредственно, либо кривую, заданную параметрически. В последнем случае на кривой-архетипе фиксируются базовые точки. Отметим, что универсальным средством параметрического задания кривых являются ряды Фурье.

2. С каждой базовой точкой связывают вектор, задающий её перемещение в узел, который будет лежать на кривой, которую мы строим.

3. Для каждого узла задаются углы наклона входящей и исходящей касательных. Обычно эти углы одинаковы при поиске кривых, имеющих сходную морфологию и наиболее приемлемых с точки зрения поставленной задачи.

4. Для окончательного построения конкретной кривой задаются остальные параметры, то есть длины входящей и исходящей стороны четырёхугольника и параметры сплайнов, сопрягающих соседние узлы. Именно их подбор приводит к построению кривой, принимающей окончательный индивидуальный облик.

Пример 12. Пусть на плоскости задано упорядоченное множество точек. Построить замкнутую кривую, проходящую через эти точки.

Тем самым, мы демонстрируем метод построения кривой-архетипа. Построение кривой проводится с помощью конечных рядов Фурье [14].

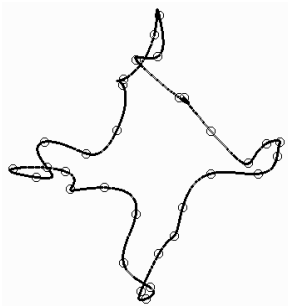
Пусть задано дискретное множество точек 0, $\frac{\pi}{n}, \frac{2\pi}{n}, \ldots \frac{(2n-1)\pi}{n}$. Функция f(i), где i – номер точки (нумерация с нуля), может быть задана в виде

$$f(i) = \frac{a_0}{2} + \sum_{k=1}^{n-1} \left(a_k \cos\frac{\pi}{n} ki + b_k \sin\frac{\pi}{n} ki\right) + \frac{a_n}{2}\cos \pi i ,$$

где $a_k = \frac{1}{n}\sum_{i=0}^{2n-1} f(i)\cos\frac{\pi}{n} ki$ (k = 0, 1, … n) и $b_k = \frac{1}{n}\sum_{i=0}^{2n-1} f(i)\sin\frac{\pi}{n} ki$ (k = 1, 2, … , n–1).

Используя этот факт, мы строим кривую следующим образом. Базовые точки заданы парами координат (xi, yi). Тем самым, мы имеем на дискретном множестве из 2n базовых

точек две функции $x = x(i)$ и $y = y(i)$. Разложив каждую из них в конечный ряд Фурье, мы можем построить кривую-архетип. Подчеркнём, что это не осмысленное изображение, а носитель базовых точек.



В программе точки, расположенные на астроиде ($x = cos^3 u$, $y = sin3 u$) [15], подвергнуты дополнительным случайным сдвигам.

```
screen 12: pi = 4 * atn(1): n = 16
dim x(2 * n), y(2 * n), a(2 * n), b(2 * n), a1(2 * n), b1(2 * n)
x0 = 320: y0 = 240: a = 200
for i = 0 to 2 * n step 1: u = i * pi / n: x(i) = cos(u) ^ 3 + rnd(1) * .3: next i
for j = 0 to n: a(j) = 0: b(j) = 0
for i = 0 to 2 * n – 1: a(j) = a(j) + x(i) * cos(j * i * pi / n) / n
b(j) = b(j) + x(i) * sin(j * i * pi / n) / n: next i, j
for i = 0 to 2 * n step 1: u = i * pi / n: y(i) = sin(u) ^ 3 + rnd(1) * .3: next i
for j = 0 to n: a1(j) = 0: b1(j) = 0
for i = 0 to 2 * n – 1: a1(j) = a1(j) + y(i) * cos(j * i * pi / n) / n
b1(j) = b1(j) + y(i) * sin(j * i * pi / n) / n: next i, j
for i = 0 to 2 * n – 1: xe = x0 + a * x(i): ye = y0 – a * y(i)
circle (xe, ye), 3: next i
for u = 0 to 2 * pi step pi / (8 * n): xx = a(0) / 2 + a(n) * cos(u * n) / 2
yy = a1(0) / 2 + a1(n) * cos(u * n) / 2: for i = 1 to n – 1
xx = xx + (a(i) * cos(i * u) + b(i) * sin(i * u))
yy = yy + (a1(i) * cos(i * u) + b1(i) * sin(i * u)): next i
xe = x0 + a * xx: ye = y0 – a * yy: circle (xe, ye), 1: next u
```

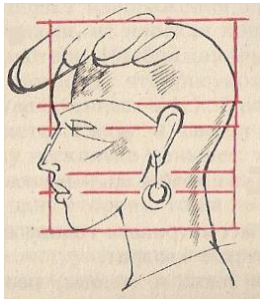
Формирование абсцисс
Разложение в ряд Фурье

Формирование ординат
Разложение в ряд Фурье

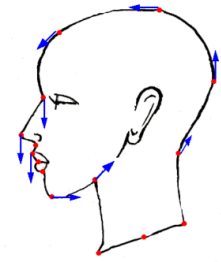
Построение архетипа

Пример 13. Разработать метод построения профиля человеческого лица.

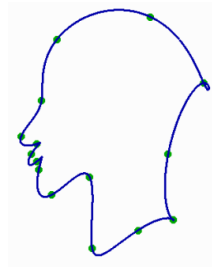
В качестве образца мы взяли рисунок из [16]. Он был подвергнут компьютерной обработке, а затем на нём было выделено чётное число (2n) базовых точек. Построенная по их координатам параметризованная кривая (архетип) сама по себе плохо передаёт особенности исходного изображения. По этой причине необходимо использовать сплайны.



Исходное изображение



Изображение с базовыми точками



Фурье-архетип

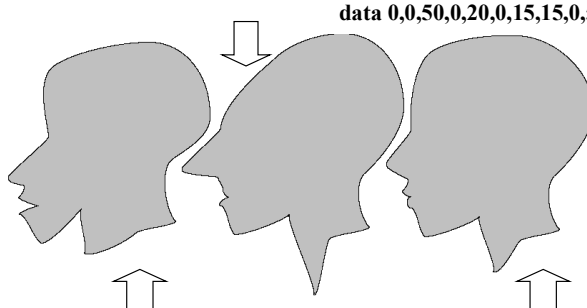
В качестве архетипа используем набор базовых точек (без параметризованной кривой).

```
screen 12: pi = 4 * atn(1): n = 8: k = 2 * n – 1
rem Координаты точек архетипа
dim x(k), y(k)
data 400,180,295,50,110,95,80,215,37,285,75,300,65,320,75,335,80,350
data 100,400,175,365,180,505,270,470,340,450,330,320,400,180
```

```
for i = 0 to k step 1: read x(i), y(i): x(i) = x(i) * .8: y(i) = y(i) * .8: next i
rem Смещения (углы и расстояния)
dim ugsd(k), rsd(k)
data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,.25,0
data 0,0,10,0,8,0,5,15,0,-5,20,50,50,50,20,0
for i = 0 to k step 1: read ugsd(i): ugsd(i) = ugsd(i) * 2 * pi: next i
for i = 0 to k step 1: read rsd(i): x(i) = x(i) + rsd(i) * cos(ugsd(i)):
y(i) = y(i) - rsd(i) * sin(ugsd(i)): next i
rem Параметры сплайнов
dim uin(k), uout(k), uiz(k), rin(k), rout(k), n(k), m(k), t0(k)
data .25, .5, .625, .75, .75, .75, .625, 0, 0, 0, .125, .625, .125, .125, .125, .25
data 0, 0, 0, 0, 0, 0, 0, .5, 0, 0, .5, .25, 0, .25, 0, 0
data 30, 80, 50, 50, 10, 10, 10, 5, 10, 20, 20, 20, 20, 20, 30, 50
data 80, 50, 50, 10, 10, 10, 5, 10, 20, 20, 20, 20, 20, 30, 50, 30
data 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
data 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
data .5, .5, .5, .5, .95, .95, .5, .5, .5, .5, .5, .5, .5, .5, .5
for i = 0 to k: read uin(i): uin(i) = 2 * pi * uin(i): next i:
for i = 0 to k: read uiz(i): uiz(i) = 2 * pi * uiz(i)
uout(i) = uin(i) + uiz(i): next i: for i = 0 to k: read rin(i): next i
for i = 0 to k: read rout(i): next i: for i = 0 to k: read n(i): next i
for i = 0 to k: read m(i): next i: for i = 0 to k: read t0(i): next i
for i = 0 to k – 1: c1 = m(i) / ((n(i) - n(i) * t0(i) + m(i) * t0(i)) * t0(i) ^ (n(i) – 1))
c2 = (1 – c1 * t0(i) ^ n(i)) / ((1 – t0(i)) ^ m(i))
v1 = x(i) + rout(i) * cos(uout(i)): w1 = y(i) – rout(i) * sin(uout(i))
v2 = x(i + 1) – rin(i + 1) * cos(uin(i + 1)): w2 = y(i + 1) + rin(i + 1) * sin(uin(i + 1))
for t = 0 to 1 step .001: xa = x(i) + t * (v1 – x(i)): ya = y(i) + t * (w1 - y(i))
xb = v2 + t * (x(i + 1) – v2): yb = w2 + t * (y(i + 1) – w2)
if t < t0(i) then g = c1 * t ^ n(i) else g = 1 – c2 * (1 – t) ^ m(i)
x = xa + g * (xb - xa): y = ya + g * (yb - ya): pset (x, y): next t, i
```

Роль смещений базовых точек архетипа давно осознана художниками, что подтверждается словами Альбрехта Дюрера: «Я передвину все отвесные и поперечные линии в квадрате профильного изображения совершенно произвольно и помещу их наискось, как придётся, а затем снова нарисую по ним линии лица» [17]. Продемонстрируем лёгкость изменения профиля с помощью одних смещений базовых точек.

```
dim ugsd(k), rsd(k)
data 0,0,0,0,.5,0,0,0,0,0,0,0,.25,.25,.25,0
data 0,0,50,0,20,0,15,15,0,5,20,50,50,50,20,0
```



```
dim ugsd(k), rsd(k)
data 0,0,0,0,0,0,0,0,0,.5,.5,.25,.25,.25,.25,.25,0
data 0,0,10,30,8,0,0,20,10,–5,20,70,70,70,50,0
```

```
dim ugsd(k), rsd(k)
data 0,.5,.25,0,.25,0,0,.125,0,0,0,.125,.25,.25,.25,0
data 0,25,10,0,10,0,0,5,0,5,20,50,50,50,20,0
```

Конечно, попытка прорисовки человеческого профиля является достаточно рискованным предприятием. Дело в том, что здесь затрагиваются тонкие эстетические ощущения. На самом деле и в технике дело обстоит аналогичным образом. «От формы изделия зависит его эстетическое восприятие» [18]. Поэтому при современном состоянии возможностей компьютерной графики необходимо рисковать.

К тому же совершенно ясно, что подбор параметров, осуществляемый вручную, весьма неудобен. Но на основе описанного алгоритма можно написать программу, позволяющую осуществлять выбор архетипа, смещений и параметров сплайнов в интерактивном режиме. Подобная программа может использоваться как удобный инструмент художника или дизайнера. Однако этот круг вопросов мы в данной статье не рассматриваем.

4. ПОСТРОЕНИЕ ЛИНИЙ В КРИВОЛИНЕЙНЫХ КООРДИНАТАХ

Перейдём к аналитической интерпретации предложенного выше кинематического метода. Для этого ещё раз объясним, как точка с декартовыми координатами (t1; t2) переносится на плоскость, связанную с четырёхугольником (сохраняются обозначения примера 2).

Основой отображения является соответствие вершин единичного квадрата (0; 0), (1; 0), (1; 1), (0; 1) вершинам четырёхугольника (x0; y0), (x1; y1), (x2; y2), (x3; y3) соответственно. Координаты двух точек A(t1; 0) и B(t1; 1), определяющих в декартовой системе координат вертикальную прямую, переходят в точки P и Q:

$$x_P = x_0 + t^1 \cdot (x_1 - x_0),$$
$$y_P = y_0 + t_1 \cdot (y_1 - y_0),$$

$$x_Q = x_3 + t_1 \cdot (x_2 - x_3),$$
$$y_Q = y_3 + t_1 \cdot (y_2 - y_3).$$

Теперь можно вычислить координаты точки, в которую переходит точка (t1; t2):

$$x = x_P + t2 \cdot (x_Q - x_P),$$
$$y = y_P + t2 \cdot (y_Q - y_P).$$

Исключив переменные xP, yP, xQ, yQ, получим окончательные выражения, задающие отображение:

$$x = x_0 + t_1 \cdot (x_1 - x_0) + t_2 \cdot (x_3 - x_0) + t_1 t_2 \cdot (x_2 - x_3 - x_1 + x_0),$$
$$y = y_0 + t_1 \cdot (y_1 - y_0) + t_2 \cdot (y_3 - y_0) + t_1 t2 \cdot (y_2 - y_3 - y_1 + y_0).$$

Поскольку переменные x и y выражаются через переменные t_1 и t_2 нелинейно, мы имеем дело с криволинейной системой координат.

Вообще говоря, криволинейная система координат представляет собой отображение прямоугольной системы координат $(t_1; t_2)$ в систему координат $(x; y)$ с помощью аналитических функций $x = x(t_1, t_2)$ и $y = y(t_1, t_2)$. Изменяя $t1$ при постоянных значениях t_2 (и наоборот, изменяя t2 при постоянных значениях t1) можно вычертить координатные линии криволинейной системы координат [19].

Поскольку отображение осуществляется с помощью гладких функций, отображение переводит кривые, которые касались друг друга в прямоугольной системе координат, в соприкасающиеся кривые в криволинейной системе координат. Именно это обстоятельство и является важнейшим для предлагаемого нами метода.

Следует особо отметить, что важное с теоретической точки зрение требование взаимной однозначности отображения при практическом применении метода криволинейных координат иногда может быть опущено (примеры 2 и 6).

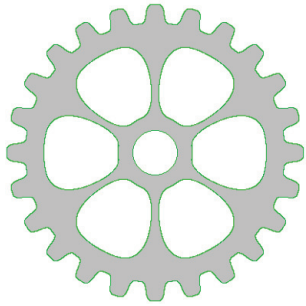
Пока мы использовали только весьма частный вид криволинейных систем координат. Однако при перенесении метода на системы координат общего вида никаких затруднений возникнуть не может. Тем не менее, прежде чем приводить примеры, дадим словесное описание алгоритма.

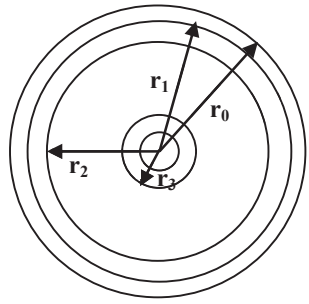
1. Пусть задана криволинейная система координат, заданная с помощью функций $x = x(t1, t2)$ и $y = y(t_1, t_2)$. Ставится задача гладкого сопряжения двух кривых $x = x(t_1, 0)$ и $y = y(t_1, 1)$, являющихся координатными линиями данной криволинейной системы координат.
2. Для более точной постановки задачи на соответствующих кривых необходимо указать точки, через которые проходит сопрягающая кривая, например, точки, определяемые параметрами $t_1 = 0$, $t_2 = 0$ и $t_1 = 1$, $t_2 = 1$.
3. Сопряжение осуществляется графиком функции $t_2 = g(t_1)$, на которую наложен условия: $g(0) = 0, g(1) = 1, g'(0) = g'(1) = 0$. График строится в криволинейны координатах на отрезке [0; 1].
4. Переход к сопряжению произвольных координатных линий в произвольной точк может быть произведено с помощью введения новой параметризации. Например, если точки сопряжения (а, значит, и координатные линии) определяются параметрами $t_1 = a$, $t_2 = c$ и $t_1 = b$, $t_2 = d$, можно ввести новые параметры p_1 и p_2, помощью формул $t_1 = a + p_1 \cdot (b - a)$ и $t_2 = c + p_2 \cdot (d - c)$, где p_1 и p_2 меняются

Пример 14. Построить изображение зубчатого колеса.

Так же как и в случае построения профиля человеческого лица мы ограничимся созданием, хотя и узнаваемого, но абстрактного изображения. Известно, профиль зубьев колёс, как правило, имеет эвольвентную боковую форму, предложенную Эйлером. Однако это далеко не единственный вариант выбора формы зубчатых колёс. «Задаваясь произвольно профилем зуба одного колеса, можно построить сопряжённый профиль зуба другого колеса» [20]. Мы же, демонстрируя сопряжение координатных линий в полярной системе координат, всего лишь построим узнаваемые очертания шестерни.

В программе многократно строятся гладкие сопряжения двух параллелей в полярной системе координат. Кроме зубьев построим спицы, число которых в четыре раза меньше числа зубьев (m). Сопряжения, очерчивающие как зубья, так и спицы, осуществляются с помощью сплайнов степени n из примера 7. При построении зубцов параллели сопрягаются, как описано в алгоритме. При построении спиц сопрягаемые параллели параметризуются в противоположных направлениях.





```
screen 12: pi = 4 * atn(1)                                          Параметры
x0 = 300: y0 = 240: r0 = 200: r1 = 175: r2 = 150: r3 = 50: r4 = 30: n = 5: m = 24   шестерни
for i = 0 to m: ug = 2 * pi * i / m: for t = 0 to 1 step .001: u = ug + pi * t / m   Построение зубцов
if t < .5 then k = t ^ n * 2 ^ (n – 1) else k = 1 – (1 – t) ^ n * 2 ^ (n - 1)
v0 = x0 + r0*cos(u): w0 = y0 – r0*sin(u): v1 = x0 + r1*cos(u): w1 = y0 – r1*sin(u)
x = v0 + k * (v1 – v0): y = w0 + k * (w1 – w0): pset (x, y), 2: next t
ug = 2 * pi * i / m + pi / m: for t = 0 to 1 step .001: u = ug + pi * t / m
if t < .5 then k = t^n * 2 ^ (n – 1) else k = 1 – (1 – t) ^ n * 2 ^ (n - 1)
v0 = x0 + r1*cos(u): w0 = y0 – r 1*sin(u): v1 = x0 + r0*cos(u): w1 = y0 – r0*sin(u)
x = v0 + k * (v1 – v0): y = w0 + k * (w1 – w0): pset (x, y), 2: next t: next i
for i = 0 to m / 4: ug = 8 * pi * i / m: for t = 0 to 1 step .001: du = 7 * pi * t / m   Построение спиц
u1 = ug + du: u2 = ug + 7 * pi / m – du
if t < .5 then k = t ^ n * 2 ^ (n – 1) else k = 1 – (1 – t) ^ n * 2 ^ (n – 1)
v0 = x0 + r2 * cos(u1): w0 = y0 – r2 * sin(u1)
v1 = x1 + r3 * cos(u2): w1 = y1 – r3 * sin(u2)
x = v0 + k * (v1 – v0): y = w0 + k * (w1 – w0): pset (x, y), 2: next t: next i
for i = 0 to m / 4: ug = 8 * pi * i / m: for t = 0 to 1 step .001: du = 7 * pi * t / m
u1 = ug – du: u2 = ug – 7 * pi / m + du
if t < .5 then k = t ^ n * 2 ^ (n – 1) else k = 1 – (1 – t) ^ n * 2 ^ (n – 1)
v0 = x0 + r2 * cos(u1): w0 = y0 – r2 * sin(u1)
v1 = x1 + r3 * cos(u2): w1 = y1 – r3 * sin(u2)
x = v0 + k * (v1 – v0): y = w0 + k * (w1 – w0)
pset (x, y), 2: next t: next i: circle (x0, y0), r4, 2
```

Легко понять, что факт успешного использования полярной системы координат, связан с формой изображаемого объекта. Таким образом, в тех случаях, когда мы хотим воспользоваться «готовой» системой криволинейных координат, мы в значительной степени ограничиваем себя в выборе формы изображения.

Одним из неисчерпаемых источников, дающих доступ к новым системам криволинейных координат, являются конформные отображения. Взяв произвольную аналитическую функцию $w = f(z)$, где $z = x + yi$ и $w = u + vi$, и выделив её действительную и мнимую части: $u = u(x, y)$ и $v = v(x, y)$, мы получим криволинейную систему координат [21].

В качестве примера рассмотрим частный случай дробно-линейных преобразований, которые являются движениями плоскости Лобачевского в модели Пуанкаре [22]. Каждое из этих отображений w = w(z) зависит от трёх параметров r_0, u_0, φ и имеет вид $w = e^{i\varphi}\dfrac{z - z_0}{1 - z\bar{z}_0}$, где $z_0 = r_0 e^{iu_0} = a_0 + b_0 i$.

Пусть $z = r \cdot e^{u} = a + b_i$, тогда $a = r \cos u$, $b = r \sin u$. Вводя новые обозначения, получим цепочку вычислений для получения действительной и мнимой части w. Эти вычисления составят основу алгоритма для построения координатных линий соответствующей криволинейной системы координат.

Пусть $z - z_0 = p_1 + q_1 i$, тогда $p_1 = a - a_0$ и $q_1 = b - b_0$.

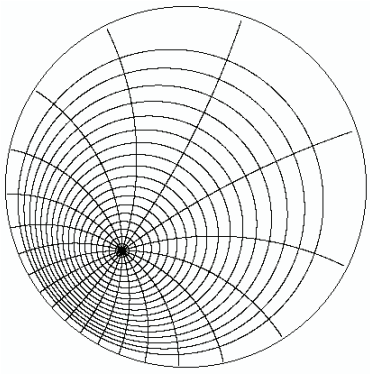
Далее положим, что $1 - z\bar{z}_0 = p_2 + q_2 i$. Тогда $p_2 = 1 - aa_0 - bb_0$ и $q_2 = ab_0 - ba_0$.

Получаем, что $p_3 + q_3 i = \dfrac{z - z_0}{1 - z\bar{z}_0} = \dfrac{p_1 + q_1 i}{p_2 + q_2 i} = \dfrac{p_1 p_2 + q_1 q_2}{p_2^2 + q_2^2} + \dfrac{p_2 q_1 - p_1 q_2}{p_2^2 + q_2^2} i$.

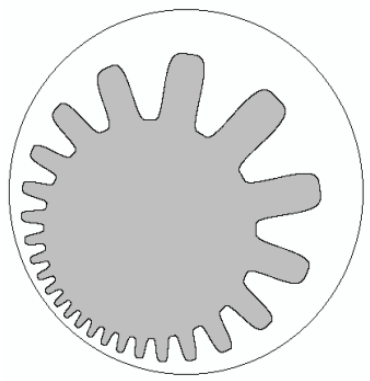
Следовательно, $p_3 = \dfrac{p_1 p_2 + q_1 q_2}{p_2^2 + q_2^2}$ и $q_3 = \dfrac{p_2 q_1 - p_1 q_2}{p_2^2 + q_2^2}$. Наконец, $w = (\cos\varphi + i \sin\varphi)(p_3 + q_3 i)$.

Окончательно получаем $Re\ w = p_3 \cos\varphi - q_3 \sin\varphi$ и $Im\ w = p_3 \sin\varphi + q_3 \cos\varphi$.

Пример 15. Построить координатные линии криволинейной системы координат, соответствующие конкретным значениям трёх параметров r_0, u_0, и φ.



```
screen 12: pi = 4 * atn(1): xx = 320: yy = 240:
                                    rr = 200: circle (xx, yy), rr
r0 = 1 / 2: u0 = pi / 4: ug = 0: rem ug соответствует φ
a0 = r0 * cos(u0): b0 = r0 * sin(u0)
for r = 0 to .95 step .05: for u = 0 to 2 * pi step pi / 200
a = r * cos(u): b = r * sin(u): p1 = a – a0: q1 = b - b0
p2 = 1 – a * a0 – b * b0: q2 = b0 * a – a0 * b
p3 = (p1 * p2 + q1 * q2) / (p2 ^ 2 + q2 ^ 2)
q3 = (p2 * q1 – p1 * q2) / (p2 ^ 2 + q2 ^ 2)
re = p3 * cos(ug) – q3 * sin(ug): im =
                                    p3 * sin(ug) + q3 * cos(ug)
x = xx + rr * re: y = yy – rr * im: pset (x, y): next u, r
for r = 0 to .99 step .01: for u = 0 to 2 * pi step pi / 10
a = r * cos(u): b = r * sin(u): p1 = a – a0: q1 = b – b0
p2 = 1 – a * a0 - b * b0: q2 = b0 * a – a0 * b
p3 = (p1 * p2 + q1 * q2) / (p2 ^ 2 + q2 ^ 2)
q3 = (p2 * q1 – p1 * q2) / (p2 ^ 2 + q2 ^ 2)
re = p3 * cos(ug) – q3 * sin(ug):
                                    im = p3 * sin(ug) + q3 * co
x = xx + rr * re: y = yy – rr * im: pset (x, y): next u, r
```



С точки зрения геометрии окружностей на евклидовой плоскости мы построили два ортогональных пучка окружностей [23]. С точки же зрения геометрии Лобачевского построены пучок концентрических окружностей и пучок прямых, проходящих через общий центр окружностей. Таким образом, на плоскости Лобачевского получен аналог полярной системы координат.

Пример 16. Построить изображение зубчатого колеса на плоскости Лобачевского.

В приводимой программе ограничимся построением только внешних очертаний шестерни без спиц и втулки. Окружность на рисунке представляет собой абсолют плоскости Лобачевского.

```
screen 12: pi = 4 * atn(1): dim r(1), re(1), im(1), a(1), b(1): r(0) = .9: r(1) = .7
xx = 320: yy = 240: rr = 200: circle (xx, yy), rr: r0 = 1 / 2: u0 = pi / 4: ugol = 0
a0 = r0 * cos(u0): b0 = r0 * sin(u0): n = 5: m = 24
for i = 0 to m: ug = 2 * pi * i / m: for t = 0 to 1 step .01: u = ug + pi * t / m
for j = 0 to 1: a(j) = r(j) * cos(u): b(j) = r(j) * sin(u)
p1 = a(j) – a0: q1 = b(j) – b0: p2 = 1 - a(j) * a0 – b(j) * b0: q2 = b0 * a(j) – a0 * b(j)
p3 = (p1 * p2 + q1 * q2) / (p2 ^ 2 + q2 ^ 2): q3 = (p2 * q1 – p1 * q2) / (p2 ^ 2 + q2 ^ 2)
re(j) = p3 * cos(ugol) – q3 * sin(ugol): im(j) = p3 * sin(ugol) + q3 * cos(ugol): next j
if t < .5 then k = t ^ n * 2 ^ (n – 1) else k = 1 – (1 – t) ^ n * 2 ^ (n – 1)
v = re(0) + k * (re(1) – re(0)): w = im(0) + k * (im(1) – im(0))
x = xx + rr * v: y = yy – rr * w: pset (x, y): next t: rbuf = r(0): r(0) = r(1): r(1) = rbuf
ug = 2 * pi * i / m + pi / m: for t = 0 to 1 step .01: u = ug + pi * t / m: for j = 0 to 1
a(j) = r(j) * cos(u): b(j) = r(j) * sin(u): p1 = a(j) – a0: q1 = b(j) – b0
p2 = 1 – a(j) * a0 – b(j) * b0: q2 = b0 * a(j) – a0 * b(j)
p3 = (p1 * p2 + q1 * q2) / (p2 ^ 2 + q2 ^ 2): q3 = (p2 * q1 – p1 * q2) / (p2 ^ 2 + q2 ^ 2)
re(j) = p3 * cos(ugol) - q3 * sin(ugol): im(j) = p3 * sin(ugol) + q3 * cos(ugol): next j
if t < .5 then k = t ^ n * 2 ^ (n – 1) else k = 1 – (1 – t) ^ n * 2 ^ (n – 1)
v = re(0) + k * (re(1) – re(0)): w = im(0) + k * (im(1) – im(0)): x = xx + rr * v
 y = yy – rr * w: pset (x, y): next t: rbuf = r(0): r(0) = r(1): r(1) = rbuf: next i
```

Построенное нами изображение можно рассматривать как некий курьёз, однако, это не так. Кратко аргументируем это утверждение. Прежде всего, отметим, что с момента возникновения геометрии Лобачевского – первой неевклидовой геометрии, возникла и идея неевклидовой механики. Уже в 1829 Лобачевский писал: «Оставалось бы исследовать, какого рода перемена произойдёт от введения воображаемой Геометрии в Механику и не встретится ли здесь принятых уже и несомнительных понятий о природе вещей…» [24]. С тех пор развитие механики в частности и физики в целом шло в направлении геометризации этих наук [25]. Общеизвестно, что с точки зрения общей теории относительности пространство и время образуют риманово пространство.

В статистическую физику вошла идея многомерного фазового пространства [26]. В значительной степени именно по этой причине получила развитие теория динамических систем. При этом понятие динамической системы эквивалентно понятию автономной системы дифференциальных уравнений [27]. При этом оказывается, что в общем случае фазовое пространство является не евклидовым пространством, а некоторым дифференцируемым многообразием [28].

Подобное развитие науки привело к тому, что механика в настоящее время часто трактуется как геометрия неевклидовых пространств и дифференцируемых многообразий [29]. Конечно, прежде всего, речь идёт о механике материальной точки, но логика науки, несомненно, потребует обобщения всех разделов механики на случай неевклидовых пространств. Должна получить развитие неевклидова динамика твёрдого тела, а в своё время и теория неевклидовых машин и механизмов.

В настоящее время появился даже более мощный стимул разработки подобных идей. Речь идёт о создании виртуальных миров, хотя бы в чисто изобразительном плане. Подобную задачу издавна решает искусство. Достаточно вспомнить творчество М. К. Эшера и его картины «Рука с отражающим шаром», «Три шара», «Капля росы» и «Глаз» [30]. Искривлённые изображения дают прекрасное представление о неевклидовой геометрии.

Искривлять изображения могут не только линзы и кривые зеркала, но и интуиция художника. Анри Бергсон, обосновывая право художника по-своему изображать мир, сказал: «Какое расположение известных нам кривых сможет когда-нибудь со штрихом карандаша великого художника?» [31]. Идея искажения мира получила своё развитие в сюрреализме. «В изобразительном искусстве основные тенденции сюрреализма были представлены имитацией художественных приёмов первобытного искусства, творчества детей и душевнобольных, вычленением конкретных объектов из естественной для них среды, их «эстетизацией» путём отстранения от реальной функции или парадоксального сочетания с иными объектами» [10]. На связь сюрреализма с неевклидовыми геометриями наводят «мягкие часы» с картины Сальвадора Дали «Постоянство памяти». Таким образом, неевклидовы геометрии связаны с поиском новых эстетических идей и воплощением новых миров.

Наконец необходимость построения изображений в неевклидовых пространствах возникает в связи с некоторыми идеями естествознания. В. И. Вернадский стремился построить единую картину мироздания, используя новейшие открытия математики, физики, химии, биологии. Он писал: «Может быть в ней (Вселенной) найдутся места, где нельзя будет с точностью современной научной работы руководствоваться этой (евклидовой) геометрией» [32]. Вскоре учёный нашёл такие места очень близко: «Состояние пространства, отвечающего телу живого организма, как бы оно мало или велико не было, диссимметрично. Это проявляется в правизне и левизне – в неравенстве явлений посолонь и противосолонь». И далее: «П. Кюри совершенно правильно учёл возможность разных форм диссимметрии и выразил геометрическую структуру, связь при этом выявленную в положении, что дисимметрическое явление вызывается такою же дисимметрической причиной. Исходя из этого принципа – можно назвать его принципом Кюри – следует, что особое состояние пространства жизни обладает особой геометрией, которая не является обычной геометрией Евклида» [33]. Это предположение Вернадского получило более конкретное развитие. С направлениями конкретного развития этих идей можно познакомиться, например, в работах [34, 35].

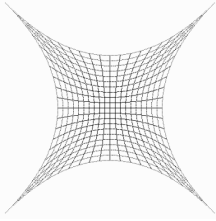
Ограничившись перечисленными соображениями, отметим, что криволинейные координаты позволяют строить изображения, связанные с неевклидовыми геометриями. Важность этих изображений связана с разработкой некоторых видов виртуальных миров и исследованиями ряда пространственных структур реального мира. Сюда можно отнести

1. Создание иллюзии движения тел в неевклидовых пространствах, в том числе для изучения неевклидовых механизмов.
2. Создание эстетических эффектов, прежде всего, дизайнерского характера.
3. Исследование морфологии реальных объектов, построенных по принципам неевклидовых геометрий.

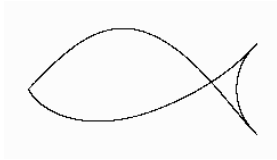
Рассмотрим пример преобразования форм с помощью криволинейных координат в духе д'Арси Томпсона. Отметим, что при этом криволинейные координаты играют роль своеобразной искривляющей линзы, через которую мы как бы рассматриваем изображение, построенное в декартовых координатах.

Пример 17. Построить простейшее зооморфное изображение и подвергнуть его преобразованиям в криволинейной системе координат.

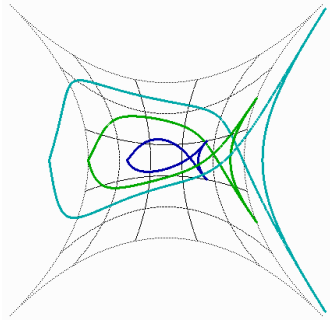
Примитивное изображение рыбы строится из трёх сплайнов по методике, рассмотренной в примере 10. Криволинейную систему координат зададим с помощью формул $v = y \cdot (x_2 + 1)$ и $u = x \cdot (y_2 + 1)$. Если исходное изображение подвергнуть на декартовой плоскости гомотетии, то в криволинейных координатах будет меняться форма.



Координатные линии системы координат



Исходное изображение рыбы в декартовых координатах



Преобразование формы при гомотетии

Построение координатных линий

```
screen 12: for x = –1 to 1 step .01: for y = –1 to 1.01 step .4: u = y * (x ^ 2 + 1)
v = x * (y ^ 2 + 1): xe = 320 + 100 * u: ye = 240 – 100 * v: pset (xe, ye): next y, x
for x = –1 to 1.01 step .4: for y = –1 to 1 step .01: u = y*(x^2 + 1): v = x (y^2 + 1)
xe = 320 + 100*u: ye = 240 – 100*v: pset (xe, ye): next y, x: pi = 4*atn(1): k = 3
```

Параметры сплайнов

```
dim x(k), y(k), uin(k), uout(k), uiz(k), rin(k), rout(k), n(k), m(k), t0(k)
data 100, 240, 300, 280, 300, 200, 100, 240
    data .125, –.125, .125, .325
        data 0, .5, .5, .5
            data 200, 200, 50, 150
                data 200, 50, 25, 100
                    data 2, 2, 2, 2
                        data 2, 2, 2, 2
                            data .5, .5, .5, .5
for i = 0 to k: read x(i), y(i): x(i) = (x(i) – 200) / 200: y(i) = (y(i) – 240) / 200: next i
for i = 0 to k: read uin(i): uin(i) = 2 * pi * uin(i): next i
for i = 0 to k: read uiz(i): uiz(i) = 2 * pi * uiz(i): uout(i) = uin(i) + uiz(i): next i
for i = 0 to k: read rin(i): rin(i) = rin(i) / 200: next i
for i = 0 to k: read rout(i): rout(i) = rout(i) / 200: next i
for i = 0 to k: read n(i): next i: for i = 0 to k: read m(i): next i
```

```
for i = 0 to k: read t0(i): next i
for i = 0 to k – 1: c1 = m(i) / ((n(i) – n(i) * t0(i) + m(i) * t0(i)) * t0(i) ^ (n(i) – 1))      Построение изображения
c2 = (1 – c1 * t0(i) ^ n(i)) / ((1 – t0(i)) ^ m(i)): v1 = x(i) + rout(i) * cos(uout(i))
w1 = y(i) – rout(i) * sin(uout(i)): v2 = x(i + 1) – rin(i + 1) * cos(uin(i + 1))
w2 = y(i + 1) + rin(i + 1) * sin(uin(i + 1)): for t = 0 to 1 step .001
xa = x(i) + t * (v1 – x(i)): ya = y(i) + t * (w1 – y(i))
xb = v2 + t * (x(i + 1) – v2): yb = w2 + t * (y(i + 1) – w2)
if t < t0(i) then g = c1 * t ^ n(i) else g = 1 – c2 * (1 – t) ^ m(i)      Гомотетия
x = xa + g * (xb – xa): y = ya + g * (yb – ya): for kf = 1 to 3      с коэффициентом kf
v = (kf * y) * ((kf * x) ^ 2 + 1): u = (kf * x) * ((kf * y) ^ 2 + 1)
xe = 320 + 100 * u: ye = 240 + 100 * v: circle (xe, ye), 1, kf: next kf: next t, i
```

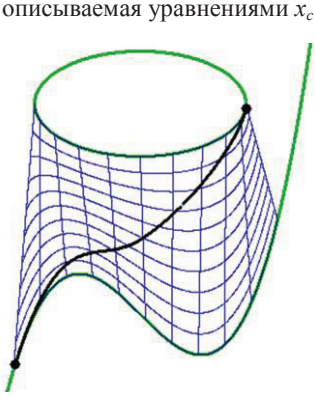
5. СОЗДАНИЕ КРИВОЛИНЕЙНЫХ СИСТЕМ КООРДИНАТ С ЗАДАННЫМИ СВОЙСТВАМИ

Поскольку, как мы уже отмечали, характер криволинейной системы координат до известной степени определяет вид изображений, которые удобно строить с её помощью, желательно рассмотреть методы создания криволинейных систем координат с заданными свойствами. Начнём с решения следующей задачи.

Пример 18. На плоскости находятся две кривые, заданные параметрически. Осуществить их гладкое сопряжение.

В известном смысле сопряжение двух кривых осуществляется так же, как и сопряжение прямых в примере 4, но вместо традиционного четырёхугольника используется четырёхугольник криволинейный. При этом в примере 4 мы фактически строили криволинейную систему координат. То же самое происходит и здесь. Опишем процесс построения криволинейной системы координат по двум заданным кривым.

Пусть заданы две кривых, описываемые двумя парами параметрических уравнений от одного параметра: $x_a = f_1(t_1)$, $y_a = g_1(t_1)$ и $x_b = f_2(t_1)$, $y_b = g_2(t_1)$. При изменении параметра t_1 по первой кривой движется точка A $(x_a; y_a)$, а по второй – точка B $(x_b; y_b)$. Задав значение второго параметра t_2, на отрезке AB можно построить точку C с координатами $x_c = x_a + t_2 \cdot (x_b - x_a)$ и $y_c = y_a + t_2 \cdot (y_b - y_a)$. В результате построена криволинейная система координат, описываемая уравнениями $x_c = f_1(t_1) + t_2 \cdot (g_1(t_1) - f_1(t_1))$ и $y_c = f_2(t_1) + t_2 \cdot (g_2(t_1) - f_2(t_1))$.



Две исходные кривые являются координатными линиями при значениях параметра $t_2 = 0$ и $t_2 = 1$. Теперь сопряжение этих кривых можно осуществить с помощью уже описанного нами алгоритма.

Напомним, что сопряжение осуществляется графиком функции $t^2 = g(t_1)$, на которую наложеныусловия:
$g(0) = 0, g(1) = 1, g'(0) = g'(1) = 0$ Приведём конкретный пример. При этом построим не только сопрягающую линию, но и координатные линии криволинейной системы координат. Мы сопряжём график функции $y = x^3 - x - 1$ и дугу эллипса: $x = \cos u$, $y = 1 + 0{,}5 \cdot \sin u$.

```
screen 12: pi = 4*atn(1): x0 = 320: y0 = 240: a = 100
for v1 = –2 to 2 step .001: w1 = v1 ^ 3 – v1 – 1: x = x0 + v1 * a      Построение
y = y0 – w1*a: circle (x, y), 1, 2: next v1                              графика
for u = 0 to 2*pi step pi/500: v2 = cos(u): w2 = 1 + .5*sin(u)      Построение эллипса
x = x0 + v2*a: y = y0 – w2*a: circle (x, y), 1, 2: next u
for t1 = 0 to 1.01 step .1: for t2 = 0 to 1 step .01: v1pi = pi2*t1t1*2.5: u =      Построение
```

```
w1 = v1^3 – v1 – 1: v2 = cos(u): w2 = 1 + .5*sin(u): p = v1 + t2*(v2 – v1):                координатных
q = w1 + t2*(w2 – w1): x = x0 + p*a: y = y0 – q * a: pset (x, y), 1: next t2, t1            линий
for t1 = 0 to 1 step .01: for t2 = 0 to 1.01 step .1: v1 = –1.2 + t1*2.5: u = pi + pi*t1
w1 = v1 ^ 3 – v1 – 1: v2 = cos(u): w2 = 1 + .5 * sin(u): p = v1 + t2 * (v2 – v1)
q = w1 + t2*(w2 – w1): x = x0 + p*a: y = y0 – q*a: pset (x, y), 1: next t2, t1
for t1 = 0 to 1.01 step .001: if t1 < .5 then t2 = 2*t1^2 else t2 = 1 – 2 * (1 – t1)^2
v1 = –1.2 + t1*2.5: u = pi + pi*t1: w1 = v1^3 – v1 – 1: v2 = cos(u): w2 = 1 + .5*sin(u)   Построение
p = v1 + t2 * (v2 – v1): q = w1 + t2 * (w2 – w1): x = x0 + p * a: y = y0 – q * a           сопрягающей
circle (x, y), 1, 3:next t1                                                               кривой
```

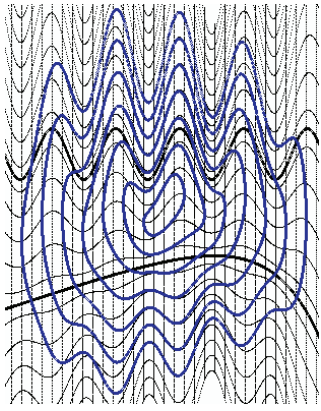
В очередной раз отметим, что при построении криволинейной системы координат мы не следим за взаимной однозначностью соответствующих отображений, но для наших целей это и не важно.

Рассмотрим важный частный случай построения криволинейной системы координат. Его важность определяется тем, что функции с графиками нужной формы легко сконструировать с помощью традиционных математических методов, например, интерполяции.

Пример 19. Заданы две функции *f(x)* и *g(x)*. Построить криволинейную систему координат, в которой координатными линиями являются графики этих функций и вертикальные прямые x = const. Начертить в этой системе координат несколько эллипсов с общим центром и эксцентриситетом.

Поскольку мы имеем дело с частным случаем построения, проведённого в предыдущем примере, используем уже полученный результат. В обозначениях примера 18 имеем: $t_1 = x$, $t_2 = t$, $xa = t_1$, $y_a = f(t_1)$ и $x_b = t_1$, $y_b = f(t_1)$. В результате координатные линии описываются уравнениями $y = f(x) + t\cdot(g(x) - f(x))$ с параметрами *x* и *t*. Каждая координатная линия создаваемой системы координат не только является линейной комбинацией двух исходных функций, но, можно сказать, что мы имеем дело с барицентрической системой координат на множестве всех координатных линий [36].

Для построения конкретной системы координат выбраны функции $f(x) = -e^{x-1} +$ и $g(x) = \frac{\sin 5x}{2} + 2$, что позволяет создать достаточно причудливую картину искривления плоскости. Эта деформация естественным образом отражается на форме эллипсов, входящ в семейство. При этом хорошо видно, как форма исходных функций налагается на фор эллипсов. Особенно это относится к синусоиде.



```
screen 12: x0 = 320: y0 = 300: a = 50
for x = –3.2 to 3.2 step .01: f = (–exp(x – 1) + x)/3: g = sin(5*x)/2 + 2
 xx = x0 + a * x: yf = y0 – a * f: yg = y0 – a * g
circle (xx, yf), 1: circle (xx, yg), 1: next x
for t = –1.5 to 3.001 step .2: for x = –3.2 to 3.2 step .01
f = (–exp(x – 1) + x) / 3: g = sin(5 * x) / 2 + 2: fg = f + t * (g – f)
xx = x0 + a * x: yfg = y0 – a * fg: pset (xx, yfg): next x, t
for t = –1.5 to 3 step .01: for x = –3.2 to 3.2 step .2
f = (–exp(x – 1) + x) / 3: g = sin(5 * x) / 2 + 2: fg = f + t * (g – f)
xx = x0 + a * x: yfg = y0 – a * fg: pset (xx, yfg): next x, t
pi = 4*atn(1): for r = .2 to 1.6 step .2: for u = 0 to 2*pi step pi/1000
x = 2 * r * cos(u): t = r * sin(u) + 1 / 2: f = (–exp(x – 1) + x) / 3
g = sin(5 * x) / 2 + 2: fg = f + t * (g – f)
xx = x0 + a * x: yfg = y0 – a * fg: circle (xx, yfg), 1, 1: next u, r
```

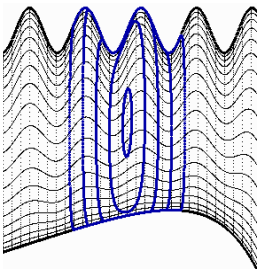
Зависимость формы эллипсов от исходных функций наводит на мысль о том, что можно достаточно просто создавать кривые определённых очертаний хотя бы в первом при-

ближении. Их в частности можно рассматривать как архетипы, которые в дальнейшем будут использованы для детального приближения к нужной форме.

Однако при этом возникают вопросы технического характера, которые следует предварительно рассмотреть. Поскольку построение системы координат связано с барицентрической системой координат, при увеличении абсолютных значений параметра t растут абсолютные значения масс. В результате неограниченно нарастают колебания координатных линий.

Обсудим способы решения этой проблемы. Начнём с варианта, использующего замену параметра t на своеобразную характеристическую функцию $\chi(t)$, обладающую следующими свойствами: $\chi(t) = 0$ при $t \le 0$; $\chi(t) = t$ при $0 \le s(t) \le 1$; $\chi(t) = 1$ при $t \ge 1$. Функция система координат не будет распространена на всю плоскость, а сосредоточится в полосе между функциями f(x) и g(x).

Пример 20. Построить криволинейную систему координат, сосредоточенную в полосе между функциями f(x) и g(x). Начертить в этой системе координат несколько концентрических окружностей.

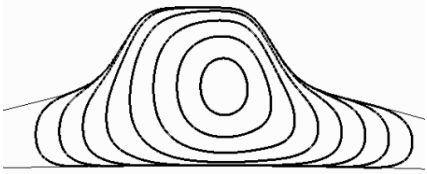


```
screen 12: x0 = 320: y0 = 300: a = 50: for x = –3.2 to 3.2 step .01
f = (–exp(x – 1) + x) / 3 – 1: g = sin(5 * x) / 2 + 3: xx = x0 + a * x
yf = y0 – a * f: yg = y0 – a * g: circle (xx, yf), 1, 0: circle (xx, yg), 1, 0: next x
for t = –.5 to 1.501 step .05: for x = –3.2 to 3.2 step .01: f = (–exp(x-1) + x)/3 – 1
g = sin(5 * x) / 2 + 3: if t < 0 then s = 0
if t > 0 and t < .5 then s = 2 * t ^ 2
if t > .5 and t < 1 then s = 1 – 2 * (1 – t) ^ 2
if t > 1 then s = 1
fg = f + s * (g – f): xx = x0 + a * x: yfg = y0 – a * fg: pset (xx, yfg), 0: next x, t
for t = –.5 to 1.5 step .01: for x = –3.2 to 3.2 step .2: f = (–exp(x-1) + x) / 3 – 1
g = sin(5 * x) / 2 + 3: if t < 0 then s = 0
if t > 0 and t < .5 then s = 2 * t ^ 2
if t > .5 and t < 1 then s = 1 – 2 * (1 – t) ^ 2
if t > 1 then s = 1
fg = f + s * (g – f): xx = x0 + a * x: yfg = y0 – a * fg: pset (xx, yfg), 0: next x, t
pi = 4 * atn(1): for r = .1 to 1.301 step .3: for u = 0 to 2 * pi step pi / 1000
x = r*cos(u): t = r*sin(u) + .5: f = (-exp(x – 1) + x)/3 – 1: g = sin(5*x)/2 + 3
if t < 0 then s = 0
if t > 0 and t < .5 then s = 2 * t ^ 2
if t > .5 and t < 1 then s = 1 – 2 * (1 – t) ^ 2
if t > 1 then s = 1
fg = f + s*(g – f): xx = x0 + a*x: yfg = y0 – a*fg: circle (xx, yfg), 1, 1: next u, r
```

Отметим, что с ростом радиуса окружностей они впечатываются в графики функций f(x) и g(x). Тем самым у нас возникает ещё один способ сопряжения координатных линий с помощью впечатывания в полосу окружности, эллипса или иной кривой.

Пример 21. Построить линию, которую можно рассматривать как архетип кузова легкового автомобиля.

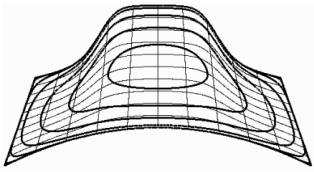
Задача решается с помощью построения семейства концентрических эллипсов в криволинейной системе координат и выбора одного из них. Форма кузова определяется выбором функций, ограничивающих полосу. В нашем конкретном случае использованы функции $f(x) = -1 - \frac{x^2}{180}$ и $g(x) = 1 - \frac{x^6}{1+x^6} - \frac{x^2}{20}$. Не обсуждая достоинства и недостатки полученных форм, отметим, каким простым алгоритмом они были получены. Кроме того, результат получен отнюдь не случайным образом, а с помощью целенаправленных действий.



```
screen 12: x0 = 320: y0 = 240: a = 100: pi = 4 * atn(1)
for x = -3.2 to 3.2 step .01: f = –1 – x ^ 2 / 180
g = 1 – x ^ 6 / (x ^ 6 + 1) – x ^ 2 / 20
xx = x0 + a * x: yf = y0 – a * f: yg = y0 – a * g: pset (xx, yf)
pset (xx, yg): next x
for r = .2 to 1.8 step .2: for u = 0 to 2 * pi step pi / 500
x = r * 1.5 * cos(u) + 1 / 3: t = r * sin(u) / 2 + 1 / 2
f = –1 – x ^ 2 / 180: g = 1 – x ^ 6 / (x ^ 6 + 1) – x ^ 2 / 20
if t < 0 then s = 0
if t > 0 and t < .5 then s = 2 * t ^ 2
if t > .5 and t < 1 then s = 1 - 2 * (1 – t) ^ 2
if t > 1 then s = 1
fg = f + s * (g – f): xx = x0 + a * x: yfg = y0 – a * fg
circle (xx, yfg), 1: next u, r
```

Пример 22. Построить криволинейную систему координат, сосредоточенную в конечной области, представляющей собой часть полосы между функциями *f(x)* и *g(x)* и ограниченную слева и справа двумя отрезками. Начертить в этой системе координат несколько концентрических окружностей.

В данном случае мы используем две характеристических функции s1(t1) и s2(t2) с соответствующими свойствами.



```
screen 12: x0 = 320: y0 = 240: a = 100: pi = 4 * atn(1)
for t1 = 0 to 1 step .001: for t2 = 0 to 1.01 step .1
x1 = –2.5 + t1 * 5: x2 = –2 + t1 * 4: f = –1 – x1 ^ 2 / 10
g = 1 – x2 ^ 6 / (x2 ^ 6 + 1) – x2 ^ 2 / 20: x = x1 + t2 * (x2 – x1)
y = f + t2*(g – f): xx = x0 + a*x: yf = y0 – a*y: pset (xx, yf): next t2, t1
for t1 = 0 to 1.01 step .1: for t2 = 0 to 1 step .001
x1 = –2.5 + t1 * 5: x2 = -2 + t1 * 4: f = –1 – x1 ^ 2 / 10
g = 1 – x2 ^ 6 / (x2 ^ 6 + 1) – x2 ^ 2 / 20: x = x1 + t2 * (x2 – x1)
y = f + t2*(g – f): xx = x0 + a*x: yf = y0 – a*y: pset (xx, yf): next t2, t1
for r = .1 to .71 step .1: for u = 0 to 2 * pi step pi / 1000
t2 = r * cos(u) + 1 / 2: t1 = r * sin(u) + 1 / 2
if t1 < 0 then s1 = 0
if t1 > 0 and t1 < .5 then s1 = 2 * t1 ^ 2
if t1 > .5 and t1 < 1 then s1 = 1 – 2 * (1 – t1) ^ 2
if t1 > 1 then s1 = 1
if t2 < 0 then s2 = 0
if t2 > 0 and t2 < .5 then s2 = 2 * t2 ^ 2
if t2 > .5 and t2 < 1 then s2 = 1 – 2 * (1 – t2) ^ 2
if t2 > 1 then s2 = 1
x1 = –2.5 + s1 * 5: x2 = –2 + s1 * 4: f = –1 – x1 ^ 2 / 10
g = 1 – x2 ^ 6 / (x2 ^ 6 + 1) – x2 ^ 2 / 20: x = x1 + s2*(x2 – x1)
y = f + s2*(g-f): xx = x0 + a*x: yy = y0- a*y: circle (xx, yy), 1: next u, r
```

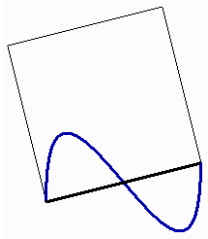
В рассмотренной системе координат две граничных линии представляют собой отрезки. Покажем, что можно использовать любые криволинейные границы нашей области.

Пример 23. На плоскости заданы две точки $(x_0; y_0)$ и $(x_1; y^1)$, задающие отрезок. Построить на нём как на части оси абсцисс график некоторой функции f(x), причём функцию нужно выбрать так, чтобы её график проходил через заданные точки.

При построении наклонного графика используются формулы аффинного отображения [5]. Распишем вычислительный алгоритм, который будет положен в основу построения криволинейной системы координат. Уточним, что мы переносим на отрезок ту часть графика, которая соответствует изменению аргумента в пределах от a до b.

Прежде всего, вычислим координаты точки $x_2 = x_1 + (y_1 - y_0)$: $y_2 = y_1 - (x_1 - x_0)$, которая определяет направление оси ординат. Теперь три точки $(x_0; y_0)$, $(x_1; y_1)$ и $(x_2; y_2)$ определяют аффинное отображение: $x = x0 + (x_1 - x_0)\cdot t_1 + (x_2 - x_0)\cdot t_2$ и $y = y_0 + (y_1 - y_0)\cdot t_1 + (y_2 - y_0)\cdot t_2$. Параметр t_1 меняется в пределах от 0 до 1 и задаёт соответствие между точками отрезка с концами $(x_0; y_0)$ и $(x_1; y_1)$ и отрезка [a; b]. Абсцисса и значения функции вычисляются по формулам $t = a + t_1(b - a)$ и $t_2 = f(t)$.

Теперь мы можем сформулировать важное для последующих построений положение. Пусть на плоскости заданы две точки $(x_0; y_0)$ и $(x_1; y_1)$. Кроме того, задана функция $f(x)$ на отрезке [a; b]. Тогда функция $t_2 = g(t_1) = f(a + t_1(b - a))$ позволяет с помощью аффинного отображения построить на соответствующем отрезке график, функции f(x). При изменении t_1 от 0 до 1 кривая строится от точки $(x_0; y_0)$ к точке $(x_1; y_1)$. Для построения кривой от точки $(x_1; y_1)$ к точке $(x_0; y_0)$ следует использовать функцию $t_2 = g(1 - t_1)$.

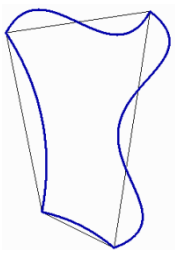


```
screen 12: x0 = 320: y0 = 240: x1 = 480: y1 = 200
x2 = x1 + (y1 – y0): y2 = y1 – (x1 – x0)
x3 = x0 + (y1 – y0): y3 = y0 – (x1 – x0)
line (x0, y0) – (x1, y1): line (x1, y1) – (x2, y2)
line (x2, y2) – (x3, y3): line (x3, y3) – (x0, y0)
for t1 = 0 to 1 step .0001: t = –1+2*t1: t2 = (t- 1)*t*(t+ 1)
x = x0 + t1 * (x1 – x0) + t2 * (x3 – x0)
y = y0 + t1 * (y1 – y0) + t2 * (y3 – y0)
circle (x, y), 1, 14: next t1
```

Координаты вспомогательных точек, дополняющих отрезок до квадрата

Пример 24. На плоскости заданы четыре точки $A_0\ (x_0; y_0)$, $A_1\ (x_1; y_1)$, $A_2\ (x_2; y_2)$, $A_3\ (x_3; y_3)$. Построить криволинейную область, ограниченную графиками функций fi(x), каждый из которых построен на отрезке $A_iA_{i+1\ mod4}$, где i = 0, 1, 2, 3.

Построение основано на алгоритме, приведённом в предшествующем примере.

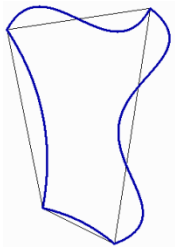


```
screen 12: dim x(4), y(4): data 100,100,300,70,250,400,150,350,100,100
for i = 0 to 4: read x(i), y(i): next i: for i = 0 to 3: line (x(i), y(i))-(x(i+1), y(i+1)): next i
for i = 0 to 3: x0 = x(i): y0 = y(i): x1 = x(i + 1): y1 = y(i + 1)
x2 = x1 + (y1 – y0): y2 = y1 – (x1 – x0): x3 = x0 + (y1 – y0): y3 = y0 – (x1 – x0)
for t1 = 0 to 1 step .0001: if i = 0 then t = –1 + 2 * t1: t2 = (t – 1) * t * (t + 1) / 3
if i = 1 then t = –1 + 3 * t1: t2 = – (t + 1) * t * (t – 1) * (t – 2) / 10
if i = 2 then t = t1: t2 = t * (t – 1) / 4
if i = 3 then t = –1 + 2 * t1: t2 = (t ^ 2 – 1) / 10
x = x0 + t1*(x1 – x0) + t2*(x3 – x0): y = y0 + t1*(y1 – y0) + t2*(y3 – y0)
circle (x, y), 1, 14: next t1, i
```

Пример 25. Построить криволинейную систему координат, сосредоточенную в конечной области, построенной в предыдущем примере.

Дадим словесное описание алгоритма построения криволинейной системы координат, используя обозначения двух предыдущих примеров.

1. По функциям $f_0(x)$ и $f_2(x)$ построим функции $g_0(t_1)$ и $g_2(1 - t_1)$.
2. При каждом значении параметра t_1 с помощью аффинных отображений и этих функций построим точки $(u_0; v_0)$ и $(u_2; v2)$.
3. По функциям $f_1(x)$ и $f_3(x)$ построим функции $g_1(t_2)$ и $g_3(1 - t^2)$.
4. При каждом значении параметра t_1 рассматриваем взвешенную функцию $g(t_2) = g_3(1 - t_2) + t_1\cdot(g_1(t_2) - g_3(1 - t_2))$.
5. Привязываем эту функцию к отрезку с концами $(u_0; v_0)$ и $(u_2; v_2)$.
6. Строим на плоскости точку (u; v), соответствующую фиксированным параметрам t_1 и t_2. Тем самым, построение криволинейной системы координат завершено.



```
screen 12: dim x(4), y(4): data 100,100,300,70,250,400,150,350,100,100
for i = 0 to 4: read x(i), y(i): next i: for i = 0 to 3: line (x(i), y(i))-(x(i+1), y(i+1)): next i
for i = 0 to 3: x0 = x(i): y0 = y(i): x1 = x(i + 1): y1 = y(i + 1)
x2 = x1 + (y1 – y0): y2 = y1 – (x1 – x0): x3 = x0 + (y1 – y0): y3 = y0 – (x1 – x0)
for t1 = 0 to 1 step .0001: if i = 0 then t = –1 + 2 * t1: t2 = (t – 1) * t * (t + 1) / 3
if i = 1 then t = –1 + 3 * t1: t2 = – (t + 1) * t * (t – 1) * (t – 2) / 10
if i = 2 then t = t1: t2 = t * (t – 1) / 4
if i = 3 then t = –1 + 2 * t1: t2 = (t ^ 2 – 1) / 10
x = x0 + t1*(x1 – x0) + t2*(x3 – x0): y = y0 + t1*(y1 – y0) + t2*(y3 – y0)
circle (x, y), 1, 14: next t1, i
```

Поскольку в качестве границ области выбираются графики произвольных функций, нами получен способ построения криволинейных координат, обладающих любыми требуемыми морфологическими свойствами. Ниже мы лишь кратко коснёмся ещё некоторых вопросов, связанных с построением криволинейных координат, хотя на самом деле они требуют очень подробного изучения, поскольку здесь возникает целый ряд математических проблем.

6. НЕКОТОРЫЕ ВОПРОСЫ МАТЕМАТИЧЕСКОГО ХАРАКТЕРА, СВЯЗАННЫЕ С СОЗДАНИЕМ КРИВОЛИНЕЙНЫХ СИСТЕМ КООРДИНАТ

Указанный в примере 25 способ параметризации криволинейного четырёхугольника имеет ту особенность, что при использовании значений параметров t^1 и t_2 за пределами отрезка [0; 1] мы выходим за пределы этого четырёхугольника. Система криволинейных координат получает продолжение либо на всю плоскость, либо на область, имеющую достаточно сложную форму.

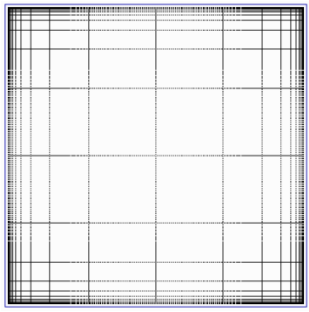
Можно сразу указать ещё два способа параметризации криволинейного четырёхугольника, при которых мы не выходим за его пределы. Прежде всего, это замена параметров t_1 и t_2 на функции $s_1(t_1)$ и $s_2(t_2)$ из примера 22. При этом способе точки, характеризующиеся большими абсолютными значениями t_1 и t_2, впечатываются в границу.

Следующий способ параметризации криволинейного четырёхугольника связан с отображением в него бесконечной декартовой плоскости.

Пример 26. Построить отображение декартовой плоскости на внутренность квадрата: $-1 < x < 1, -1 < y < 1$.

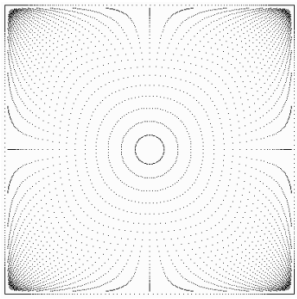
Используем отображение, задаваемое с помощью формул $x = \dfrac{p}{\sqrt{p^2+1}}$ и $y = \dfrac{q}{\sqrt{q^2+1}}$.

Можно сказать, что получена модель евклидовой плоскости внутри квадрата. Прямые и окружности при этом изображаются более сложными кривыми, но зато плоскость изображена во всей ее бесконечности. Координатные линии становятся прямолинейными отрезками, но уплотняются к границам квадрата.



```
screen 12: x0 = 320: y0 = 240: a = 200
line (x0 – a, y0 – a)-(x0 + a, y0 + a), 14, b
for p = –5 to 5 step .01: for q = –5 to 5 step .5
x = p / sqr(p ^ 2 + 1): y = q / sqr(q ^ 2 + 1)
xe = x0 + a * x: ye = y0 – a * y
pset (xe, ye): next q, p
for p = –5 to 5 step .5: for q = –5 to 5 step .01
x = p / sqr(p ^ 2 + 1): y = q / sqr(q ^ 2 + 1)
xe = x0 + a * x: ye = y0 – a * y
pset (xe, ye): next q, p
```

Пример 27. Представим себе, что на евклидовой плоскости расположена растущая окружность, которую мы отображаем в квадрат. Достигнув достаточно больших размеров, она сколь угодно тесно «прижмется» к квадрату. Таким образом, можно параметризовать кривую, являющуюся «почти квадратом».



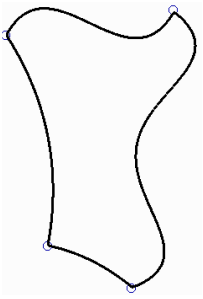
```
screen 12: x0 = 320: y0 = 240: a = 200: pi = 4 * atn(1)
for r = .1 to 5 step .1: for u = 0 to 2 * pi step pi / 50
p = r*cos(u): q = r*sin(u): x = p/sqr(p^2+1): y = q / sqr(q^2 +1)
xe = x0 + a * x: ye = y0 – a * y: pset (xe, ye): next u, r
for u = 0 to 2 * pi step pi / 50000
p = 100 * cos(u): q = 100 * sin(u)
x = p / sqr(p ^ 2 + 1): y = q / sqr(q ^ 2 + 1)
xe = x0 + a * x: ye = y0 – a * y
pset (xe, ye): next u
```

В программе построена система концентрических окружностей. Очертание квадрата построены с помощью окружности большого радиуса. Поскольку, как легко видеть, радиусы окружностей, кроме четырёх, устремлены в вершины квадрата, при построении «почти квадрата» основное время тратится на четыре угловых точки. По этой причине эти точки лучше миновать. Для этого последний цикл желательно заменить следующим вложенным циклом.

```
for i = 0 to 3: u0 = -pi / 4 + i * pi / 2: for u = u0 + .35 to u0 + pi / 2 – .7 step pi / 10000
p = 100 * cos(u): q = 100 * sin(u): x = p / sqr(p ^ 2 + 1): y = q / sqr(q ^ 2 + 1)
xe = x0 + a * x: ye = y0 – a * y: pset (xe, ye): next u, i
```

Вообще использование семейства концентрических окружностей приводит к тому, что в области строится система координат, сходная с полярными координатами.

Пример 28. Построить отображение декартовой плоскости в криволинейный четырёхугольник и выполнить параметризацию его границы с помощью окружности, прижатой к ней.



```
screen 12: pi = 4 * atn(1): dim x(4), y(4): data 100,100,300,70,250,400,150,350,100,100
for i = 0 to 4: read x(i), y(i): circle (x(i), y(i)), 5, 14: next i
rad = 10: for ug = 0 to 2 * pi step pi / 10000: p1 = rad * cos(ug): p2 = rad * sin(ug)
t1 = .5 * p1 / sqr(p1 ^ 2 + 1) + .5: t2 = .5 * p2 / sqr(p2 ^ 2 + 1) + .5
x3 = x(0) + (y(1) – y(0)): y3 = y(0) – (x(1) – x(0))
t = –1 + 2 * t1: g = (t – 1) * t * (t + 1) / 3: u0 = x(0) + t1 * (x(1) – x(0)) + g * (x3 – x(0))
v0 = y(0) + t1*(y(1)–y(0))+g*(y3–y(0)): x3=x(2)+(y(3) – y(2)): y3 = y(2) – (x(3) – x(2))
t = 1 – t1: g = t * (t – 1) / 4: u2 = x(2) + (1 – t1) * (x(3) – x(2)) + g * (x3 – x(2))
v2 = y(2) + (1 – t1) * (y(3) – y(2)) + g * (y3 – y(2)): u = u0 + (v2 – v0): v = v0 – (u2 – u0)
t = –1+2*(1–t2): g1=–(t^2 – 1) / 10: t = –1 + 3 * t2: g3 = – (t + 1) * t * (t – 1) * (t – 2) / 10
g = g1 + t1 * (g3 – g1): x = u0 + t2 * (u2 – u0) + g * (u – u0)
y = v0 + t2 * (v2 – v0) + g * (v – v0): pset (x, y): next ug
```

Этот метод параметризации можно использовать для построения архетипов сложных изображений, в том числе связанных с гладкими замкнутыми кривыми.

Пример 29. Построить криволинейную систему координат, сосредоточенную в конечной области, ограниченной гладкой кривой.

Решение задачи требует сочетания методов из примеров 10, 25 и 28. Сначала в вершинах четырёхугольника задаются направляющие векторы, по которым строятся сплайны, сопрягающиеся в гладкую кривую. Затем строится система координат по четырём функциям-сплайнам. И, наконец, вводится аналог полярной системы координат.

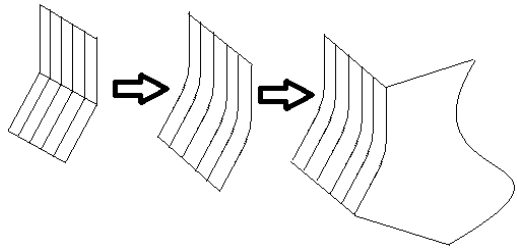
Возможна и следующая постановка задачи. Пусть задана гладкая замкнутая кривая. Необходимо построить криволинейную систему координат в области, ограниченной этой кривой. Для достаточно «хороших» кривых возможен следующий подход. На кривой выбираются четыре точки – вершины четырёхугольника. Для четырёх образовавшихся дуг строятся приближения (например, интерполяционные или сплайновые). После чего, как и в предыдущем примере, строится система координат.

Здесь прослеживается некоторое сходство с важной задачей теории функций комплексной переменной. Речь идёт о построении конформного отображения внутренности единичного круга на область, ограниченную некоторой кривой. Теорема Римана доказывает существование таких отображений [37]. Однако построение соответствующего отображения на практике весьма затруднительно. В частности используется приближённый эмпирический метод П. В. Мелентьева [38].

Предлагаемый нами метод построения криволинейных координат не приводит к построению конформных отображений. Тем не менее, отметим следующее обстоятельство. Конформное отображение внутренности единичного круга на область, ограниченную некоторой кривой, переносит в эту область модель Пуанкаре геометрии Лобачевского [39]. Отображение же построенное нами переносит в ту же область геометрию Евклида. Здесь возникают определённые ассоциации с четвёртой проблемой Гильберта [40], обсуждать которые, мы не имеем возможности. Укажем лишь, что после того, как евклидова плоскость отображена в некоторую область, там возможно использование группы автоморфизмов, являющихся образами группы преобразований евклидовой плоскости. Создаётся своеобразная локальная геометрия в ограниченной области. Это открывает определённые перспективы для развития идей д'Арси Томпсона.

Кроме самостоятельной формообразующей функции криволинейные координаты в криволинейном четырёхугольнике позволяют складывать обширные координатные системы в виде своеобразной мозаики. Если на плоскости расположены два обычных четырёхугольника, имеющих общую сторону, то, искривив общую сторону по некоторой функции, остальные стороны искривляем произвольным образом и строим в каждом криволинейную систему координат. Если же плоскость или её часть покрыта мозаикой четырёхугольников, все входящие четырёхугольники можно искривить. Вычислительная сложность работы с мозаичной системой координат, конечно, возрастает. Но если необходимо создать сложную форму, работа со сложными алгоритмами становится оправданной.

Для мозаичной системы координат легко прослеживаются аналогии с гладкими многообразиями и полиэдрами. Криволинейные четырёхугольники подобны картам, но стыкуются по границам. При этом стыковка координатных линий происходит с изломом. Но возможно и построение криволинейной системы координат с гладким сопряжением координатных линий. Общая схема построения такова.



Берётся исходный прямолинейный четырёх угольник и подвергается искривлению. К нему пристраивается второй прямолинейный четырёхугольник и также подвергается искривлению. Координатные линии, переходящие друг в друга, гладко сопрягаются сплайнами на участках небольшой длины, включая и границы четырёхугольников, переходящие друг в друга. Сопряжённые координатные линии заметают новую область. К этой области пристраивается новый четырёхугольник и операция повторяется.

Подобным способом можно построить области различной формы, в которых введена криволинейная система координат. В том числе могут быть созданы и неодносвязные области. В «дыры» этой области могут быть вложены системы координат построенные по принципу локальных геометрий.

Выбирая в каждой области только одно семейство координатных линий, мы практически строим в этой области векторное поле. В областях с локальной геометрией таким приёмом можно получить структуры, напоминающие предельные циклы.

Векторные поля на плоскости возникают в частности при рассмотрении соответствующих систем дифференциальных уравнений. Изучение этих полей несёт существенную информацию о решениях системы, даже если они не могут быть получены в явном виде [41]. У нас векторные поля могут быть построены без дифференциальных уравнений в их классическом понимании. При этом снова намечается интересная связь с теорией динамических систем. В частности на основе понятия топологической эквивалентности векторных полей можно устанавливать связь между эмпирически построенными векторными полями и дифференциальными уравнениями [42].

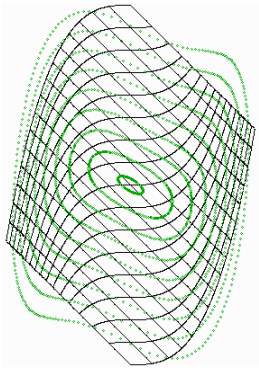
Наконец, рассмотрим ещё один полезный способ построения криволинейной системы координат – по заранее заданным координатным линиям.

Начнём со следующей задачи. Пусть заданы два семейства функций, зависящих от параметров t_1 и t_2: $y = F(x, t_1)$ и $x = G(y, t_2)$. При этом через каждую точку плоскости проходит график ровно одной функции из каждого семейства. И, кроме того, любой график из первого семейства с любым графиком из второго семейства пересекается в единственной точке. (Речь идёт о сети на плоскости [43]). Тогда две неявные функции $y = F(G(y, t_2), t_1)$ и $x = G(F(x, t_1), t_2)$, зависящие от параметров t_1 и t_2 задают криволинейную систему координат. Однако для работы с ней необходимо искать корни соответствующих уравнений, что может потребовать довольно сложных вычислений.

Пример 30. Рассмотрим две непрерывных, строго возрастающих функции f_1 и f_2, заданных на всей действительной оси и принимающих все действительные значения от $-\infty$ до $+\infty$. Кроме того, пусть $f_1(0) = f^2(0) = 0$. Это свойство в сочетании с остальными позволяет утверждать, что рассматриваемые функции отрицательны при отрицательных значениях аргумента и положительны при положительных. Введём два семейства функций $y = f_1(x) + t_1$ и $y = -f_2(x) - t_2$. В обозначениях предыдущего абзаца $F(x, t_1) = f_1(x) + t_1$ и $G(y, t_2) = f_2^{-1}(-y - t_2)$. Уравнение для вычисления x по заданным значениям параметров t_1 и t_2 принимает вид: $f_1(x) + f_2(x) + t_1 + t_2$.

Обозначим сумму $t_1 + t^2$ через c и рассмотрим функцию $u(x) = f_1(x) + f_2(x) + c$. Очевидно, что эта функция непрерывна. Она строго возрастает и принимает все действительные значения от $-\infty$ до $+\infty$. Следовательно, уравнение $u(x) = 0$ имеет ровно один действительный корень. Чтобы определить интервал, на котором расположим корень используем то обстоятельство, что $u(0) = c$. Положим $x_0 = f_2^{-1}(-c)$. По свойству функции f_2 величины c и x_0 имеют разные знаки. Далее $u(x_0) = f_1(x0) + f_2(x0) + c = f_1(x0) + f_2(f_2^{-1}(-c)) + c = f_1(x_0)$. Знаки величин x_0 и $u(x_0)$ совпадают по соответствующему свойству функции f_1.

Итак, единственный корень уравнения u(x) = 0 находится на отрезке, имеющем концы $x_0 = f_2^{-1}(-c)$ и $x_1 = 0$. Установив этот факт, мы можем прибегнуть к одному из стандартных методов приближённого решения уравнений, например, к методу деления отрезка пополам.



Для конкретного примера выбраны функции $f_1(x) = x3$ и $f_2(x) = x$. Построены координатная сетка и семейство концентрических окружностей.

```
screen 12: xe0 = 320: ye0 = 240: ae0 = 100
for t1 = –2 to 2.01 step .25: for t2 = –2 to 2 step .01
c = t1 + t2: x0 = –c: x1 = 0
1    u0 = x0 ^ 3 + x0 + c: u1 = x1 ^ 3 + x1 + c: x = (x0 + x1) / 2: u = x ^ 3 + x + c
if u * u0 < 0 then x1 = x else x0 = x
if abs(x1 – x0) <= .001 then 2 else 1
2 u1 = x ^ 3 + t1: xe = xe0 + x * ae0: ye = ye0 – u1 * ae0: pset (xe, ye): next t2, t1
for t1 = –2 to 2 step .01: for t2 = –2 to 2.01 step .25: c = t1 + t2: x0 = –c: x1 = 0
```

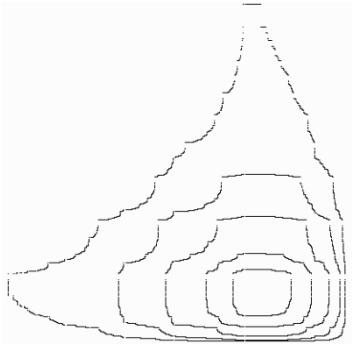
```
3    u0 = x0 ^ 3 + x0 + c: u1 = x1 ^ 3 + x1 + c: x = (x0 + x1) / 2: u = x ^ 3 + x + c
if u * u0 < 0 then x1 = x else x0 = x
if abs(x1 - x0) <= .001 then 4 else 3
4 u1 = x ^ 3 + t1: xe = xe0 + x * ae0: ye = ye0 – u1 * ae0: pset (xe, ye): next t2, t1
pi = 4 * atn(1): for ug = 0 to 2 * pi step pi / 100: for r = .1 to 2.51 step .3
t1 = r * cos(ug): t2 = r * sin(ug): c = t1 + t2: x0 = –c: x1 = 0
5    u0 = x0 ^ 3 + x0 + c: u1 = x1 ^ 3 + x1 + c: x = (x0 + x1) / 2: u = x ^ 3 + x + c
if u * u0 < 0 then x1 = x else x0 = x
if abs(x1 – x0) <= .001 then 6 else 5
6 u1 = x^3 + t1: xe = xe0 + x*ae0: ye = ye0 – u1*ae0: circle (xe, ye), 1: next r, ug
```

Не обсуждая общий случай, когда каждая координатная линия имеет индивидуальную форму, отметим, что даже в самых простых ситуациях могут возникнуть интересные варианты криволинейных систем координат.

Пример 31. Пусть координатными линиями являются два семейства прямых: пучок вертикальных прямых и ортогональный им пучок горизонтальных прямых. В отличие от предыдущего примера прямые каждого пучка не параметризованы. Задача как раз и состоит в выборе параметризации.

Если положить $x = t_1$ и $y = t_2$, мы получим обычную декартову систему координат. Если же задать параметризацию формулами $x = f_1(t1)$ и $y = f_2(t_2)$, где функции f_1 и f_2 – две непрерывных, строго возрастающих функции, заданных на всей действительной оси и принимающих все действительные значения от $-\infty$ до $+\infty$, возникнет криволинейная система координат.

Она может быть довольно причудливой. Используем в качестве функций f_1 и f_2 λ-функции Орема [44]. Поскольку эти функции нигде не дифференцируемы, образы гладких кривых становятся в новой системе координат фрактально изломанными.



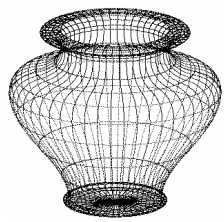
```
screen 12: m = 12: n = 2 ^ m: lm1 = .3: lm2 = .2: dim y1(n), y2(n)
y1(0) = 1: y1(n) = 2: y2(0) = 1: y2(n) = 2
for i = 0 to m – 1: s = n / (2 ^ i): for j = 0 to n – s step s
y1(j + s / 2) = lm1 * y1(j) + (1 – lm1) * y1(j + s)
y2(j + s / 2) = lm2 * y2(j) + (1 – lm2) * y2(j + s): next j, i
pi = 4 * atn(1): for r = .1 to .51 step .1
for u = 0 to 2 * pi step pi / 10000
t1 = r * cos(u) + .5: t2 = r * sin(u) + .5
i = int(t1 * n): j = int(t2 * n)
xe = 120 + (y1(i) – 1) * 400
ye = 40 + (y2(j) – 1) * 400
pset (xe, ye): next u, r
```

7. ИСПОЛЬЗОВАНИЕ МЕТОДА КРИВОЛИНЕЙНЫХ КООРДИНАТ В ТРЁХМЕРНОМ ПРОСТРАНСТВЕ

В компьютерной геометрии базовыми задачами, связанными с построением трёхмерных изображений, являются задача построения пространственных кривых и задача построения поверхностей. Методы решения этих задач чрезвычайно многообразны. Мы же рассмотрим только те из них, которые либо непосредственно используют выше изложенные алгоритмы, либо являются их обобщением.

Пример 32. Построить объёмное изображение стамноса (пример 11) с помощью вращения плоской кривой вокруг оси.

Описание алгоритма построения тел вращения дано в [5].

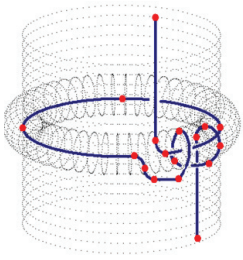


```
screen 12: pi = 4 * atn(1): cx0 = 320: cy0 = 400: cr0 = 150: ck0 = .3: cu0 = pi/6
cx2 = cx0: cy2 = cy0 - cr0: k = 7: a1 = 90: a2 = 205: a3 = 105: a4 = 135: h1 = 10
h2 = 210: h3 = 300: h4 = 320: h5 = 330
dim x(k), y(k), uin(k), uout(k), uiz(k), rin(k), rout(k), n(k), m(k), t0(k)
x(0) = 0: x(1) = x(0) + a1: x(2) = x(1): x(3) = x(0) + a2: x(4) = x(0) + a3
x(5) = x(0) + a4: x(6) = x(5): x(7) = x(0) + a3: y(0) = 410
y(1) = y(0): y(2) = y(0) – h1: y(3) = y(0) – h2: y(4) = y(0) – h3: y(5) = y(0) – h4
y(6) = y(0) – h5: y(7) = y(6): data 0, 0, .5, .25, .25, 0, .5, .5
data 0, 0, –.25, 0, 0, 0, 0, 0
data 20, 20, 20, 150, 150, 20, 20, 20
data 20, 20, 150, 150, 20, 20, 20, 20
data 2, 4, 2, 2, 2, 2, 2, 2
data 2, 4, 2, 2, 2, 2, 2, 2
data .5, .5, .5, .5, .5, .5, .5, .5
for i = 0 to k: read uin(i): uin(i) = 2 * pi * uin(i): next i
for i = 0 to k: read uiz(i): uiz(i) = 2 * pi * uiz(i): uout(i) = uin(i) + uiz(i): next i
for i = 0 to k: read rin(i): next i: for i = 0 to k: read rout(i): next i
for i = 0 to k: read n(i): next i: for i = 0 to k: read m(i): next i
for i = 0 to k: read t0(i): next i
for cug = 0 to 2 * pi step pi / 25: cx1 = cx0 + cr0 * cos(cu0 + cug)
cy1 = cy0 – cr0 * ck0 * sin(cu0 + cug): for i = 0 to k – 1
c1 = m(i) / ((n(i) – n(i) * t0(i) + m(i) * t0(i)) * t0(i) ^ (n(i) – 1))
c2 = (1 – c1 * t0(i) ^ n(i)) / ((1 – t0(i)) ^ m(i))
v1 = x(i) + rout(i) * cos(uout(i)): w1 = y(i) – rout(i) * sin(uout(i))
v2 = x(i + 1) – rin(i + 1) * cos(uin(i+1)): w2 = y(i + 1) + rin(i+1) * sin(uin(i+1))
for t = 0 to 1 step .01: xa = x(i) + t * (v1 – x(i)): ya = y(i) + t * (w1 – y(i))
xb = v2 + t * (x(i + 1) – v2): yb = w2 + t * (y(i + 1) – w2)
if t < t0(i) then g = c1 * t ^ n(i) else g = 1 – c2 * (1 - t) ^ m(i)
x = xa + g * (xb – xa): y = ya + g * (yb – ya): t1 = x / 200: t2 = 2 – y / 200
p = cx0+t1*(cx1–cx0) + t2*(cx2 – cx0): q = cy0 + t1*(cy1 – cy0) + t2 * (cy2- cy0)
pset (p, q): next t, i, cug
```

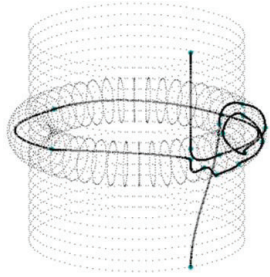
Пример 33. Обобщить алгоритм из примера 10, чтобы приспособить его для построения пространственных кривых.

Прежде всего, необходимо выбрать способ для указания направления векторов в трёхмерном пространстве. Используем для этой цели сферические координаты. Направление вектора при этом задаётся двумя углами – долготой (u_1) и широтой (u_2). При этом координаты соответствующего единичного вектора равны: $x = cos\ u_1\ cos\ u_2$, $y = = sin\ u_1\ cos\ u_2$, $z = sin\ u_2$. Сопряжение же векторов в пространстве с помощью сплайнов проводится, практически, так же как и на плоскости с помощью пространственного четырёхугольника.

В связи с этим в программе возникают два входящих угла, два исходящих угла и два угла излома для них. В примере будет построен узел в трёхмерном пространстве, поскольку на плоскости любой узел является тривиальным [45], а, значит, заузленность кривой существенным образом связана с её трёхмерностью. Естественно, работа по созданию пространственной кривой намного более сложна, чем с плоской, и здесь особенно уместным было бы использование интерактивной программы на основе соответствующего алгоритма.



Эскиз узла



Компьютерный рисунок

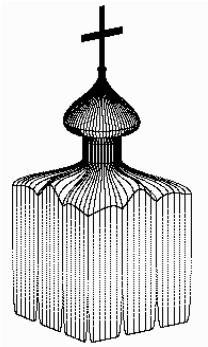
Для создания достаточно выразительного узла распределим точки гладкого сопряжения на двух поверхностях: цилиндре и торе. Направление векторов будет также ориентировано на соответствующие поверхности – чаще всего вектора направлены по касательным к круговым сечениям цилиндра и тора.

```
screen 12: pi = 4*atn(1): k=17:dim x(k),y(k),z(k): x(0)=cos(0):y(0)= sin(0): z(0)=1:x(1)=cos(0): y(1)=sin(0): z(1)=.1
for i=2 to 5: x(i)=cos(pi / 40 + (i – 2) * pi / 2): y(i) = sin(pi / 10 + (i – 2) * pi / 2): z(i) = 0: next i
x(6) = cos(0): y(6) = sin(0): z(6) = 0: x(7) = cos(pi / 20): y(7) = sin(pi / 20): z(7) = –.1
x(8)=cos(pi/10)*(1+.2*cos(–pi/2)):y(8)=sin(pi/10)*(1+.2*cos(–pi/2)):z(8) =.2*sin(–pi/2)
for i = 9 to 16: u1 = pi / 5 + (i – 8) * pi / 35: u2 = –pi / 2 + (i – 9) * pi / 2
x(i) = cos(u1) * (1 + .2 * cos(u2)): y(i) = sin(u1) * (1 + .2 * cos(u2))
z(i) = .2 * sin(u2): next i: x(17) = cos(0): y(17) = sin(0): z(17) = –1
cx0 = 320: cy0 = 240: cr0 = 150: ck0 = .3: cu0 = –pi / 3: cx3 = cx0: cy3 = cy0 – cr0
cx1 = cx0 + cr0 * cos(cu0): cy1 = cy0 - cr0 * sin(cu0) * ck0
cx2 = cx0 + cr0 * cos(cu0 + pi / 2): cy2 = cy0 – cr0 * sin(cu0 + pi / 2) * ck0
for i = 0 to k: xe = cx0 + x(i) * (cx1 - cx0) + y(i) * (cx2 - cx0) + z(i) * (cx3 – cx0)
ye = cy0 + x(i) * (cy1 – cy0) + y(i) * (cy2 – cy0) + z(i) * (cy3 – cy0): circle (xe, ye), 3, 3: paint (xe, ye), 3, 3: next i
dim uin1(k),uin2(k),uout1(k),uout2(k),uiz1(k),uiz2(k),rin(k),rout(k), n(k), m(k), t0(k)
data 0, 0, .3, .55, .8, 1.05, 0, .3, .35, .4, 0, .45, 0, .5, 0, .55, 0, 0
data –.25, –.25, 0, 0, 0, 0, –.25, 0, 0, 0, .25, 0, –.25, 0, .25, 0, –.25, –.25
data 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
data 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
data .25, .25, .25, 1, 1, 1, .1, .25, .25, .25, .25, .25, .25, .25, .25, .25, .25, 1
data .25, .25, 1, 1, 1, .1, .25, .25, .25, .25, .25, .25, .25, .25, .25, .25, 1, .25
data 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
data 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
data .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5
for i=0 to k:read uin1(i): uin1(i)=2*pi*uin1(i): next i: for i = 0 to k: read uin2(i): uin2(i) = 2 * pi * uin2(i): next i
for i = 0 to k: read uiz1(i): uiz1(i) = 2 * pi * uiz1(i): uout1(i) = uin1(i) + uiz1(i): next i
for i = 0 to k: read uiz2(i): uiz2(i) = 2 * pi * uiz2(i): uout2(i) = uin2(i) + uiz2(i): next i
for i = 0 to k: read rin(i): next i: for i = 0 to k: read rout(i): next i: for i = 0 to k: read n(i): next i
for i = 0 to k: read m(i): next i: for i = 0 to k: read t0(i) next i
for i = 0 to k – 1: c1 = m(i) / ((n(i) – n(i) * t0(i) + m(i) * t0(i)) * t0(i) ^ (n(i) – 1))
c2 = (1–c1*t0(i)^n(i))/((1–t0(i))^m(i)): v1= x(i) + rout(i) * cos(uout1(i)) * cos(uout2(i))
w1 = y(i) + rout(i) * sin(uout1(i)) * cos(uout2(i)): s1 = z(i) + rout(i) * sin(uout2(i))
v2 = x(i + 1) – rin(i + 1) * cos(uin1(i + 1)) * cos(uin2(i + 1)): w2=y(i+1)–rin(i+1)*sin(uin1(i+1))*cos(uin2(i+1))
s2= z(i+1)–rin(i+1)*sin(uin2(i + 1)): for t = 0 to 1 step .01: xa = x(i) + t * (v1 – x(i)): ya = y(i) + t * (w1 – y(i))
za = z(i) + t * (s1 – z(i)): xb = v2 + t * (x(i + 1) – v2): yb = w2 + t * (y(i + 1) – w2)
zb = s2+t*(z(i+1) – s2): if t < t0(i) then g = c1 * t ^ n(i) else g = 1 – c2 * (1 – t) ^ m(i)
x = xa + g * (xb – xa)* y = ya + g * (yb – ya): z = za + g * (zb – za)
xe = cx0 + x * (cx1 – cx0) + y * (cx2 – cx0) + z * (cx3 – cx0)
ye = cy0 + x * (cy1 – cy0) + y * (cy2 – cy0) + z * (cy3 – cy0): circle (xe, ye), 1: next t, i
```

Ещё раз обратимся к вопросу о построении поверхностей. Обычным способом построения поверхностей является их заметание движущейся кривой. Алгоритм из предыдущего примера позволяет строить поверхности любой степени сложности. Пусть в пространстве построено несколько занумерованных кривых, которые параметризованы одним параметром. Если на каждой кривой задано векторное поле, то для каждого значения параметра мы имеем возможность с помощью нашего алгоритма построить кривую. Совокупность этих кривых и заметает поверхность.

На практике эту схему можно упростить следующим образом. Пусть все занумерованные кривые составлены из одинакового числа сплайнов. Это допущение сразу же позволяет считать кривые одинаково параметризованными. Далее, пусть векторные поля заданы не по всей кривой, а только в узлах сопряжения сплайнов. Тогда мы получаем возможность вычертить на поверхности сеть, хорошо отражающую её форму.

Не вдаваясь в дальнейшие детали, покажем возможности алгоритма на простом примере.



Пример 34. С помощью описанного выше метода построить эстетически значимое изображение.

Мы построим объёмное изображение храма, хотя и схематическое, но отражающее некоторые стилевые особенности архитектуры Владимиро-Суздальского княжества XII – XV веков [46]. Узлы, составленной из сплайнов линии, скользят по пяти окружностям и графикам двух функций. Для простоты все направляющие вектора сплайнов на всех кривых направлены вертикально вниз. По этой причине направления векторов могут быть заданы непосредственно координатами, без использования сферической системы координат.

```
screen 12:pi = 4*atn(1): cx0 = 320: cy0 = 400: cr0 = 60: ck0 = .3: cu0 = pi/5: cx3 = cx0
cy3 = cy0 – cr0: cx1 = cx0 + cr0 * cos(cu0): cy1 = cy0 – cr0 * sin(cu0) * ck0
cx2 = cx0 + cr0 * cos(cu0 + pi / 2): cy2 = cy0 - cr0 * sin(cu0 + pi / 2) * ck0: k = 6
dim rin(k), rout(k), n(k), m(k), t0(k): for i = 0 to k: rin(i) = .5: rout(i) =.5: next i
for i = 0 to k: n(i) = 2: m(i) =2: t0(i) =.5: next i: dim r(4), h(4), x(6), y(6), z(6)
r(0) = .05: h(0) = 3.9: r(1) = .05: h(1) = 3.8: r(2) = .6: h(2) = 3.1: r(3) = .3: h(3) = 2.8
r(4) = .3: h(4) = 2.5: xe = cx0 + h(0) * (cx3 – cx0): ye = cy0 + h(0) * (cy3 – cy0)
circle (xe, ye), 4,,,,.5: paint (xe, ye)
for j = 1 to 2: for t1 = –1 to 1 step .001
if j = 1 then ugol = pi * t1 / 4 – pi / 2 else ugol = –pi * t1 / 4 – pi
for i = 0 to 4: x(i) = r(i) * cos(ugol): y(i) = r(i) * sin(ugol): z(i) = h(i): next i
if j = 1 then y(5) = –1: x(5) = 1 * t1 else x(5) = –1: y(5) = 1 * t1
argum = 3 * pi * t1 / 2: z(5) = abs(cos(argum)) / (4 + argum ^ 2) + 2
if j = 1 then y(6) = –1: x(6) = 1 * t1 else x(6) = –1: y(6) = 1 * t1
z(6) = (1 – abs(cos(argum))) ^ 10: for i = 0 to 6xe = cx0+x(i)
                        *(cx1–cx0)+y(i)*(cx2–cx0)+z(i)*(cx3–cx0):ye = cy0+x(i)
                                                *(cy1–cy0)+y(i)*(cy2–cy0)+z(i)*(cy3–cy0)
pset (xe, ye): next i: next t1: for t1=–1 to 1.01 step 1/9: if j = 1 then ugol =
                                                    pi*t1/4– pi/ 2 else ugol = –pi * t1 / 4 – pi
for i = 0 to 4: x(i) = r(i) * cos(ugol): y(i) = r(i) * sin(ugol): z(i) = h(i): next i
if j = 1 then y(5) = –1: x(5) = 1 * t1 else x(5) = –1: y(5) = 1 * t1
argum = 3 * pi * t1 / 2: z(5) = abs(cos(argum)) / (4 + argum ^ 2) + 2
if j = 1 then y(6) = –1: x(6) = 1 * t1 else x(6) = –1: y(6) = 1 * t1: z(6) =
                                                (1 – abs(cos(argum))) ^ 10: for i = 0 to k – 1
c1 = m(i) / ((n(i) – n(i) * t0(i) + m(i) * t0(i)) * t0(i) ^ (n(i) – 1)):c2 =
                                                    (1 – c1 * t0(i) ^ n(i)) / ((1 – t0(i)) ^ m(i))
v1 = x(i): w1 = y(i): s1 = z(i) – rout(i): v2 = x(i + 1): w2 = y(i + 1): s2 =
                                                    z(i + 1) + rin(i + 1): for t = 0 to 1 step .01
xa = x(i) + t * (v1 – x(i)): ya = y(i) + t * (w1 – y(i)): za = z(i) + t * (s1 – z(i)): xb =
                                                        v2 + t * (x(i + 1) – v2)
yb = w2+t*(y(i + 1) – w2): zb = s2 + t * (z(i + 1) – s2): if t < t0(i) then g = c1 * t ^ n(i)
else g = 1 – c2 * (1 – t) ^ m(i)
x = xa + g * (xb – xa): y = ya + g * (yb – ya): z = za + g * (zb – za)
xe = cx0+x*(cx1–cx0)+y*(cx2 – cx0) + z * (cx3 – cx0): ye = cy0 + x * (cy1 – cy0) +
                                                    y * (cy2 – cy0) + z * (cy3 – cy0)
pset (xe, ye), 0: next t, i: next t1: next j: for x = –r(0) to r(0)+.01 step r(0)/3
for z = h(0) – .1 to h(0) + 1 step .01: y = 0: xe = cx0+x*(cx1–cx0)+y*(cx2–cx0)+z*(cx3–cx0)
ye=cy0+x*(cy1–cy0)+y*(cy2–cy0)+z*(cy3–cy0): pset (xe, ye): next z, x: for x = –.4 to .4 step .01
for z = .5 + h(0) to .61 + h(0) step .01: y = 0: xe = cx0+x*(cx1–cx0)+y*(cx2–cx0)+z*(cx3–cx0)
ye=cy0+x*(cy1–cy0)+y*(cy2–cy0)+z*(cy3–cy0): pset (xe, ye): next z, x
```

8. ВЫВОДЫ

1. Представлены алгоритмы построения плоских и пространственных кривых, обладающих любой степенью морфологической сложности.
2. На основе алгоритма построения пространственных кривых предложен алгоритм построения поверхностей.
3. Все перечисленные алгоритмы могут быть использованы для создания графических редакторов, предоставляющих большие возможности конструирования кривых и поверхностей в интерактивном режиме.
4. Рассмотрены способы построения кривых в различных системах криволинейных координат.
5. Описаны разнообразные методы построения систем криволинейных координат. Они основываются как на эмпирическом подходе, так и на идеях теории функций комплексного переменного, неевклидовых геометрий и теории динамических систем.
6. Указаны перспективы применения соответствующих идей в естествознании, технике и дизайне. Даны примеры построения изображений. При этом особое внимание следует обратить на простоту используемых алгоритмов.
7. Использование различных систем криволинейных координат на плоскости открывает определённые возможности для построения прикладной морфологии плоских кривых.

ЛИТЕРАТУРА

1. Клайн М. Математика. Утрата определённости. Мир, М. 1984.
2. Блехман И. И., Мышкис А. Д., Пановко Я. Г. Механика и прикладная математика: Логи-ка и особенности приложения математики. Наука, М., 1983.
3. Завьялов Ю. С., Квасов Б. И., Мирошниченко В. Л. Методы сплайн-функций. Наука, М., 1980.
4. Роджерс Д., Адамс Дж. Математические основы машинной графики. Мир. М. 2001.
5. Степанов М. Е. Метод сложных движений в компьютерной геометрии // Моделирование и анализ данных. – 2011. – № 1.
6. Берёзкин Е. Н. Курс теоретической механики. Изд. МГУ. М. 1974.
7. Уокер Б. Женская энциклопедия. Символы, сакралии, таинства. Астрель. М. 2005.
8. Словарь античности. Прогресс. М. 1989.
9. Лорд И. А., Уилсон С. Б. Введение в дифференциальную геометрию и топологию. Математическое описание вида и формы. Институт компьютерных исследований. М., Ижевск. 2003.
10. Большая Советская Энциклопедия.
11. Померанцева Н. А. Эстетические основы искусства Древнего Египта. Искусство. М. 1985.
12. Большой Энциклопедический Словарь.
13. Розен Р. Принцип оптимальности в биологии. Мир. М. 1969.
14. Хемминг Р. В. Численные методы для научных работников и инженеров. Наука. М. 1972.
15. Савёлов А. А. Плоские кривые: Систематика, свойства, применения. Либроком. М. 2009.
16. Василевская Л. А. Специальное рисование. Высшая школа. М. 1989.
17. Дюрер А. Трактаты. Дневники. Письма. Азбука. СПб. 2000.
18. Завьялов Ю. С., Леус В. А., Скороспелов В. А. Сплайны в инженерной геометрии. Машиностроение. М. 1985.
19. Математический энциклопедический словарь. Советская энциклопедия, М., 1988.
20. Матвеев Ю. А., Матвеева Л. В. Теория механизмов и машин. Альфа-М. М., 2009.

21. Маркушевич А. И. Краткий курс теории аналитических функций. Гос. изд. физ.-мат. лит. М., 1961.
22. Фукс Б. А. Неевклидова геометрия в теории конформных и псевдоконформных отображений. Гос. изд. тех.-теор. лит. М.-Л., 1951.
23. Энциклопедия элементарной математики. Книга четвёртая. Гос. изд. физ.-мат. лит. М., 1963.
24. Григорьян А. Т. Механика в России. Наука. М., 1978.
25. Уилер Дж. А. Предвидение Эйнштейна. Мир. М., 1970.
26. Лошак Ж. Геометризация физики. Регулярная и хаотическая динамика. М. – Ижевск, 2006.
27. Дубровин Б. А., Новиков С. П., Фоменко А. Т. Современная геометрия. Наука. М., 1979.
28. Математическая энциклопедия. Том 3. Советская энциклопедия. М., 1979.
29. Розенталь И. Л. Механика как геометрия. Наука. М., 1990.
30. Эшер М. К. Графика. АРТ-Родник, М., 2008.
31. Куликова И. С. Сюрреализм в искусстве. Наука. М., 1970.
32. Вернадский В. И. Живое вещество. Наука. М., 1978.
33. Вернадский В. И. Научная мысль как планетарное явление. Наука. М., 1991.
34. Петухов С. В. Биомеханика, бионика и симметрия. Наука. М., 1981.
35. Петухов С. В. Геометрии живой природы и алгоритмы самоорганизации. // Математика-Кибернетика – 1988. – № 6. Знание. М.
36. Балк М. Б., Болтянский В. Г. Геометрия масс. Наука. М., 1987.
37. Лаврентьев М. А., Шабат Б. В. Методы теории функций комплексного переменного. Наука. М., 1965.
38. Мелентьев П. В. Приближённое конформное преобразование. // Труды НИИММ, т. 2.
39. Прасолов В. В. Геометрия Лобачевского. МЦНМО. М., 2004.
40. Погорелов А. В. Четвёртая проблема Гильберта. Наука. М., 1974.
41. Красносельский М. А., Перов А. И., Поволоцкий А. И., Забрейко П. П. Векторные поля на плоскости. Гос. изд. физ.-мат. лит. М., 1963.
42. Палис Ж., Ди Мелу В. Геометрическая теория динамических систем. Мир. М., 1986.
43. Математическая энциклопедия. Том 4. Советская энциклопедия. М., 1980.
44. Степанов М. Е. Об одном классе непрерывных функций // Моделирование и анализ данных. Труды факультета информационных технологий МГППУ. – Вып. 4, 2009.
45. Кроуэлл Р., Фокс Р. Введение в теорию узлов. Мир. М., 1967.
46. Пилявский В. И., Тиц А. А., Ушаков Ю. С. История русской архитектуры. Стройиздат. Л., 1984.

Работа поступила 26.01.2012