**Cinema 4D: фракталы и сплайны.**

**Модели вычислений**

Поговорим о моделях вычислений подробнее. Что такое язык программирования, как он представляется многим? Изначально это просто текст, называемый исходным текстом, составлен из фиксированного количества символов, организованных в более сложную структуру – слов. Это не единственный уровень организации, но другие рассматривать не будем. Далее этот текст подвергается обработке компилятором либо интерпретатором, и получаем уже другой текст, состоящий из символов, понятных исполняющему устройству (для процессоров это 0 и 1) , символы алфавита объединены в более сложные структуры, команды (и здесь возможно продолжить лестницу иерархии). Процессор, выполняя эти команды (это и есть понимание процессором полученного текста), проявляет себя в этом внешнем для себя мире какими либо его изменениями (допустим печать на принтере или мониторе). Этот этап можно считать и интерпретацией этого нового текста и реализуется уже на аппаратном уровне. Естественно, возможны варианты относительно организации исходного текста (иерархическая структура, частный случай графа), названо только 2 этажа - символ, слово, но может быть и существенно больше. Организация только в структуру типа дерева не неизбежна, возможен и граф (так построена наша речь, литературные и музыкальные произведения). Текст исполняющего устройства может быть тоже структурирован многими уровнями, допустим при необходимости защиты от взлома (технология llvm). Само исполняющее устройство может быть составным, и части этого устройства не обязательно должны быть однородными (например, совместная работа CPU и GPU в технологии CUDA и опять же llvm). При этом под термином текст можно понимать очень многое – книга, спектакль (включая и балет), кинофильм, музыкальные произведения. Примером может быть, допустим, постановка спектакля по пьесе. Исходный текст, как обычно, текст, но пьесы, компилятором выступает режиссер, интерпретация – конкретный спектакль или фильм. Аналогично и музыкальные произведения. Более экзотический пример – астрология, здесь исходным текстом является положение планет солнечной системы, а результатом компиляции астрологом является уже предсказания астролога.

Займемся, вначале очень простой задачей. Пусть нам задана строка, и мы по этой строке будем строить изображение. То есть наша строка будет аналогом исходного текста программы, результатом действия исполняющего устройства будет некоторое изображение. Более точно:

Постановка задачи. Пусть у нас имеется строка, состоящая только из символов {F, +, -},

а) задан параметр **alpha**, имеющий смысл угла (в радианной мере)

б) в каждый заданный момент положение рисующего устройства задано своими координатами (x, y, z).

в) переход из текущей позиции пера в следующую позицию реализуется рисованием отрезка 1-ой длины по направлению **beta** . Этот шаг выполняется, если при последовательном чтении символов из строки встретили символ F.

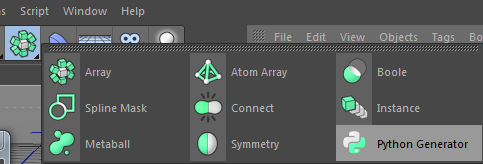
г) если встречается символ "-", то значение угла **beta** уменьшается на **alpha**, для символа "+" угол **beta** увеличивается на значение **alpha**.

д) исходное значение угла **beta** - нулевое ( можно считать и произвольным, если считать рисунки, отличающимися только поворотом, совпадающими)

Этот перечень может быть расширен введением символа **b,** который только передвигает к следующей позиции, но без рисования. Могут быть введены символы возврата, допустим открытая и закрытая скобки (отрисовать содержимое скобок и продолжить, вернувшись к началу скобок, но в другом направлении)

**Применение сплайнов.**

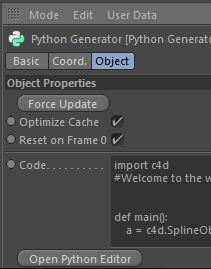
Вначале нам понадобится Python Generator, здесь показано, где его взять.



В итоге в менеджере объектов , будет только один объект:



Остается открыть редактор



И, к примеру, набрать текст:

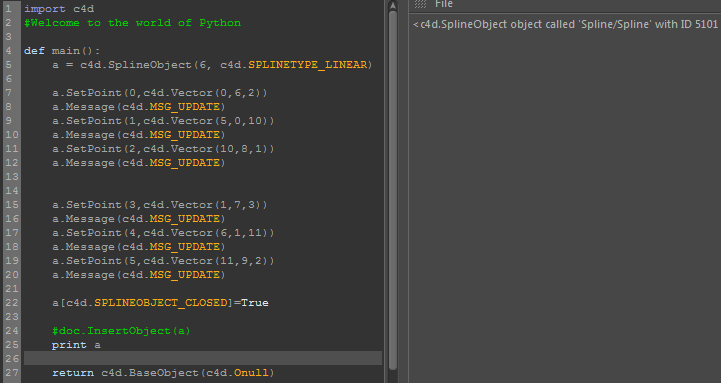
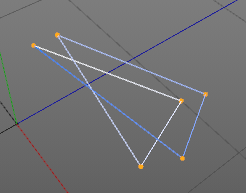


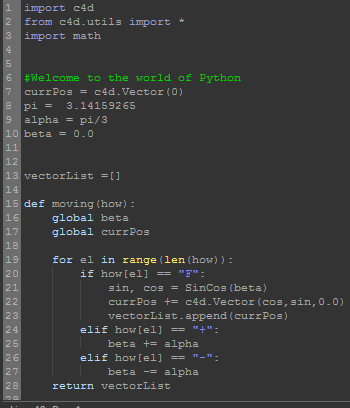
Рис.1

Текст почти самоочевиден (документирует сам себя). Создается объект сплайн с 6 точками. Сплайн линейный, то есть точки соединены прямыми линиями (строка 5). Затем назначаем точки сплайна (строки 7, 9, 11, 15, 17, 19). Между этими строками стоят команды, это требование Cinema 4D. Строка 22 делает сплайн замкнутым. Результат работы становится видимым, если убрать символ комментария # в строке 24 (откомпилировать и выполнить). Справа приведен отладочный вывод консоли Python в Cinema 4D. Картинка на выходе ниже.

.

Отрезок четко белого цвета это начало сплайна, а четко голубым цветом отмечен последний отрезок линейного сплайна.

Но это, конечно же, не решение поставленной задачи. Нам нужно написать функцию, которая по заданной строке нарисует сплайн. Пусть описанием фигуры будет такая строка . Это строка требует, что бы из текущего положения (x,y,z) , с текущим направлением **beta** делаем один шаг единичной длины, затем два поворота по двум символам “++ “ (направо, к примеру) на угол **alpha** и вновь единичный отрезок, опять двойной поворот и вновь единичный отрезок. Если взять **alpha** = π/3, то мы получим равносторонний треугольник.

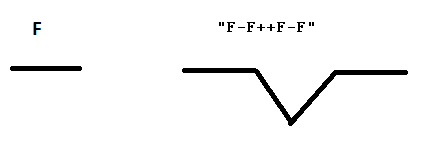
Вначале нам понадобится анализатор строки. Программа читает последовательность символов, последовательно интерпретирует читаемые символы согласно описанным выше правилам. В строках от 1 до 3 мы подгружаем модули из библиотек Python и Cinema 4D. Затем, в строках 7-10 задаем переменные состояния нашей модельной системы. Запись вектора в строке 7 означает, что все 3 компонента (x,y,z) нулевые. Результатом анализа строки будет список вершин сплайна и изначально этот список пустой (строка 13). То есть мы набору символов из алфавита из множества {F, +,-} ставим в соответствие набор векторов из 3-х (а точнее 2-х, поскольку одна из координат 0-вая) мерного пространства. Этот код модель компилятора, о котором говорилось выше. Текст, как набор символов компилируется в набор 3В векторов. Для того, что бы переменные состояния нашей системы были видны внутри функции moving(how), где тип переменной how – строка, мы объявляем эту видимость в строках 16 и 17, объявлением глобальных переменных. Тело функции состоит из элементарных инструкций языка Python и мы оставляем их без пояснений. Пока ясно одно, что получив список векторов, мы уже сможем построить сплайн, на основе примера в начале статьи.

**Трюк с подстановкой**

Если полагать, что мы имеем дело с имитацией языка программирования, то возникает вопрос о списке допустимых слов языка, а значит и о правилах генерации новых. Будем считать, что новые слова образуются на основе уже существующих слов по правилам замены. Пусть оно будет одно:

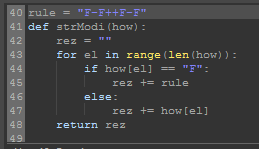
F → . (1)

Поскольку мы знаем смысл сокращения F – это отрезок единичной длины, то сможем придать смысл и этому правилу замены. Вот его графический эквивалент:

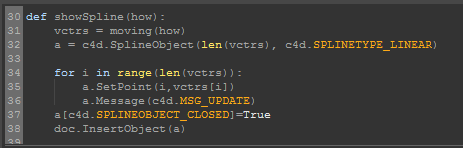


Ко вновь полученному слову языка (напомним, что это модель) можно повторно применить эту же подстановку и получить новое слово, и так без ограничений. В расширенном варианте могут дополнительно выписываться правила преобразования символов пропуска и повтора (которые упоминались выше).

Объекты такого типа называются фракталами, основное свойство – некоторый вариант симметрии, подобия самому себе. Многие языки, русский язык особенно, во многом используют этот вариант порождения новых слов, из уже существующих (спасибо – спаси бог, совесть – со-вести, самурай – сам-у-рай и т.д.). То есть, слова сами поясняют свой смысл (вариант документирования). Но мы попадаем здесь в волшебную область лингвистики, а это уже совсем другая песня. Важно, что неожиданных совпадений подозрительно много, закономерности, скорее всего в единой их глубинной природе. Фрактальную природу имеет курс валют на бирже, форма береговой линии и так далее.

Реализация подстановки кодом не представляет трудностей. 

Работа со строками достаточно проста, к пустой строке, согласно правилам замены, добавляем фрагмент, именуемый переменной rule, как только в исходной строке (формальный аргумент функции strModi – how, который получает фактическое значение при обращении к этой функции из программы). Если мы захотим реализовать повторную замену, то применяем вложенный вызов функции strModi(strModi(….(строка))….).



Все основные элементы этого фрагмента понятны по примеру в начале статьи. Создается список векторов, точек сплайнов, в строке 31, вызовом уже описанной функции moving(). Затем создается базовый объект сплайна (базовый - означает экземпляр класса BaseObject) и каждая точка списка vctrs переносится в сплайн, как в контейнер. Сплайн создается замкнутым (37 строка). Замкнутость нам нужна лишь потому, что мы потом желаем на основе сплайна получать 3D объекты. Полученная конструкция делается видимой в редакторе 3D объекта вставкой в документ (строка 38).

Для экспериментов вы можете, например, попробовать варианты:

Axiom = F+F+F+F

newF = F+F-F-FFF+F+F-F

alpha = π/2

Axiom = F

newF = F-F+F+F+F-F-F-F+F

alpha = π/2

betha = π/4

Если расширить алфавит введением дополнительного символа b, при котором соединения последовательных точек не происходит, то это вызовет изменения в анализаторе строки и структуру списка vectorList[]. Этот список распадется на отдельные подгруппы групп векторов, будет списком списков, и каждая такая подгруппа будет замкнутым сплайном. Функция showSpline() изменится немного – вместо цикла перебора только векторов, будет 2 вложенных цикла сперва по количеству сплайнов, а затем по точкам каждого из них. В терминологии Cinema 4D каждый такой сплайн уже будет называться сегментом

Примеры такого варианта:

Axiom F+F+F+F

Newf = F+b-F-FFF+F+b-F

Newb = bbb

Alpha = π/2

***Но нашей целью является показать принципиальную возможность применения сплайнов в Cinema 4D для создания обширнейшей группы фрактальных 3D объектов***, поэтому, пока, основная часть программы (функция main() ) имеет вид:

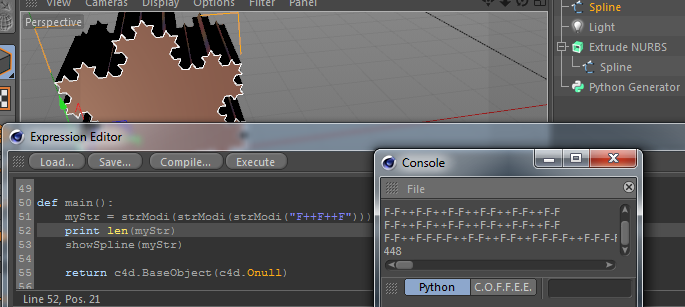


Рис. 2.

Видно, что в результате всего лишь троекратного вложения (строка 51, троекратное вложение функции strModi) получили описание нашего графического объекта строкой в 448 символов и автоматически. На консоли, рис.2, справа, видна только часть этой строки. Наш сплайн, видимый как ярко белый контур, определяется в строке 53.

Если мы пожелаем автоматизировать создание графического объекта, уже из созданного нами сплайна, с помощью какого либо инструмента, то можно воспользоваться примером из предыдущей статьи:

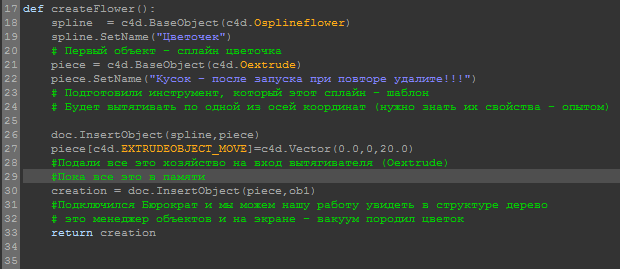


Рис.3.

В этом примере к статье 1 мы взяли уже предопределенный, существующий, сплайн (строка 18), а в нашем случае он вычислен и на рис 2 определен выражением в строке 53. Применяемый инструмент – вытягивание по одной из осей (extrude строка 21 и 27). Инструмент определяется заданием константы (в нашем случае c4d.Oextrude). Изменения вполне очевидны. Меняется только строка 18. Для наглядности мы добавили свет и текстуру к созданному графическому объекту. Текстура стандартная и взята из встроенной в пакет Cinema 4D библиотеки текстур.

В предыдущей статье мы разбирали смысл равенства A=B. Если мы с этих же позиций рассмотрим правило замены (F → , но теперь стрелочку будем считать некоторой новой записью знака равенства (это вариант определения равенства), то все наши очень длинные слова описания графических объектов становятся синонимами. Достаточно только одного – первого, в нашем случае это . Это базовое слово можно назвать и аксиомой, поскольку все остальные получаются выводом. И тогда множество слов в нашем условном мире будет состоять из аксиом (треугольник, квадрат, пятиугольник и т.д) и правила вывода. Увеличение числа допустимых символов алфавита (и интерпретации этих символов исполнительным механизмом) приводит к неограниченному числу вариантов создаваемых 3D объектов. Однако, здесь мы несколько лукавим. Мы вначале предложили схему генерации графических объектов, а затем их просматриваем. На практике дело обстоит иначе, есть объекты и нужно определить способ их генерации. Инструменты Cinema 4D позволяют решить эту задачу библиотекой предустановленных инструментов и представлением объектов в аналитическом (параметрическом) или редактируемом (как набор точек и полигонов) режимах. Но по своему содержанию такой подход полностью совпадает с задачей разработки компиляторов, конкретно, решение проблемы принадлежности слова (сцены в нашем случае) конкретному языку (набору аксиом и правил вывода).

Приведенный код, пока, мало пригоден для использования, так как необходимо знать достаточно много деталей о его работе при использовании (какую переменную и где изменить, каково допустимое значение и т.д.). Мы умышленно выбирали достаточно простые примеры, что бы проиллюстрировать основные идеи, оставляем вам простор для экспериментов. Но в более общем случае работа на процедурном уровне (создание набора функций) не эффективна. Необходима реорганизация кода. В языке Python все является объектом (Лутц, глава VII). Среди этих объектов есть две большие категории тесно связанных по определению, это объекты классов и объекты экземпляров этих классов.

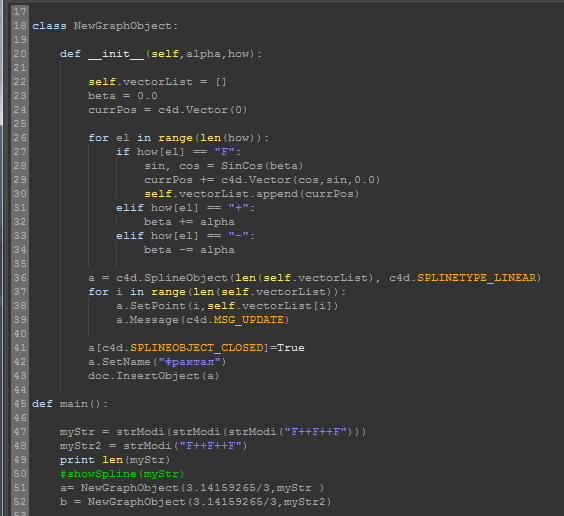
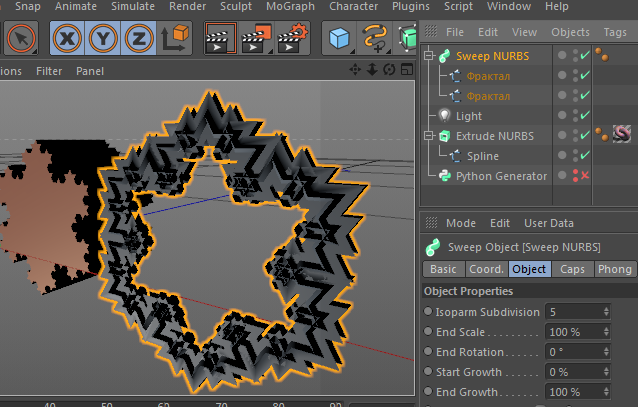


Рис. 4.

Здесь присутствует несколько необычная функция \_\_init\_\_. Эта функция автоматически вызывается, как только я указываю данные для получения экземпляра класса в круглых скобках (это строки 51 и 52 ) и количество параметров должно быть на единицу меньше количества параметров в строке определения 20. Такие функции (автоматически вызываемые) довольно распространены и символы двойного подчеркивания их характерный и обязательный признак. Параметр self выделен цветом, поскольку позволяет в тексте класса ссылаться на тот экземпляр, который этот класс создаст. То есть несет некоторую специфическую для языка Python нагрузку. Из кода ясно, что все экземпляры класса содержат набор векторов, который этот экземпляр порождает. Все остальные строки уже были описаны выше.

И в заключении остается добавить, что весь пакет Cinema 4D представляет собой набор классов и их экземпляров. Так, запись, к примеру, в строке, 42 рис. 4 означает, возьми экземпляр а, у этого экземпляра есть метод SetName, который принимает аргументом строку (“Фрактал”). Если мы посмотрим на итоги работы нашего фрагмента, то они выглядят так:



На переднем плане (желтый контур) представлена фигура полученная произведением 2-х последовательных итераций нашего сплайна. Это уже 3-х мерный фрактал, простейший способ генерации. В менеджере объектов соответствующие объекты развернуты по уровням (Sweep NURBS и Фрактал с 2-мя итерациями скользит по фракталу с 4-мя итерациями). Обратите внимание, что генератор заблокирован (красный крестик), иначе при любом движении мышью будут генерироваться сплайны. Казалось бы ничего страшного, но пакет Cinema 4D не стабилен по утечке памяти. При переполнении очень вероятно уничтожение работы и при случайной генерации столь громоздких сплайнов это событие гарантированно наступит. По крайней мере 4 Гб оперативки это минимум для возможности работать.

Часто приходится слышать от пользователей таких пакетов, что программирование приносит мало проку. Но, надеюсь, вы убедились сами, что создание вручную таких объектов практически исключено.

В следующей статье мы продолжим более глубокое изучение классов, уже самого фреймворка (набора библиотек, включая и библиотеки интерфейса, так этот термин определен у Лутца) Cinema 4D.

Литература:

1. Марк Лутц – Изучаем Python, 4-е издание, Санкт-Петербург – Москва 2011,--1280 стр.
2. Кроновер Р.М. Фракталы и хаос в динамических системах. Основы теории. Москва: Постмаркет, 2000, -- 352 стр.

**Рейдер Эдуард Ефимович, к.ф.-м.н, *freelance. 5 октября 2013 г.***