*Studia i Materiały Informatyki Stosowanej, Tom 4, Nr 7, 2012*
*str. 26-32*

# PROGRAMOWANIE FLAME FRAKTALI ORAZ FLAME ANIMACJI W SYSTEMACH APOPHYSIS ORAZ FLAM3

**Iwona Filipowicz**

*Uniwersytet Kazimierza Wielkiego*
*Instytut Techniki*
*ul. Chodkiewicza 30, 85-064 Bydgoszcz*
*e-mail: iwofil@ukw.edu.pl*

**Streszczenie**: *W pracy przedstawimy metody tworzenia fraktali typu flame w programie Apophysis oraz programowania fraktalnej animacji, bazującej na technice keyframingu, w systemie Flam3. Flame fraktal jest atraktorem iterowanego układu funkcji i powstaje w wyniku realizacji probabilistycznego algorytmu IFS. W pracy prezentujemy oryginalne przykłady flame-fraktalnych obiektów, które z jednej strony obrazują złożoność matematycznych struktur, a z drugiej strony charakteryzują się wysokim samopodobieństwem oraz symetrią i mogą stanowić niepowtarzalny motyw na tkaninach, tworzywach i innych materiałach.*

**Słowa kluczowe:** *Flame fraktal, iterowany układ funkcji, grafika i animacja fraktalna*

## Programming flame fractal and flame animation in systems Apophysis and Flam3

**Abstract:** *Abstarct: In this paper we present methods for creating a flame fractal and a fractal animation, based on a keyframing, in Apophysis and Flam3. Flame fractal is an attractor of the iterated function system and arises as a result of a probabilistic algorithm of IFS. In this paper we present original examples of flame-fractal objects, which on the one side, illustrate the complexity of the mathematical structures, on the other side, have a high self-similarity and symmetry and can provide a unique motive on fabrics, plastics and other materials.*

**Keywords:** *Keywords: Flame fractal, iterated function system, fractal art*

### 1. WSTĘP

Fraktale wywodzą się z matematyki eksperymentalnej ([2], [3]). Zaskakują nas bogactwem form, intrygujących struktur i charakterystycznych stylów. Od ponad 20 lat przyciągają uwagę naukowców, artystów i architektów. Wiele obiektów rzeczywistych i zjawisk naturalnych jest modelowanych za pomocą geometrii fraktali. Należą do nich powierzchnie planet, systemy komórkowe, struktury polimerów, obłoki gazu międzygwiezdnego, gromady galaktyk, powierzchnie białek, linie brzegowe, zbocza górskie, itd.
Fraktale i analiza fraktalna wykorzystywane są do opisu ekologicznego krajobrazu, projektowania budynków, przetwarzania obrazów wideo, lepszego poznania danych finansowych (fraktale finansowe), interpretacji obrazów medycznych. Coraz częściej w technice, biologii i fizyce stosowany jest opis fraktalny, który umożliwia symulacje i modelowanie obserwowanych zjawisk. W produkcjach kinematograficznych za pomocą fraktali generuje się wirtualne światy do złudzenia przypominające rzeczywistość, tworzy efekty specjalne i triki filmowe. Fraktale są nierozerwalnie związane z rozwojem komputerów, bez których nie byłoby możliwe ani generowanie fraktalnych struktur ani wyznaczanie fraktalnych charakterystyk.
W ostatnich latach szczególne zainteresowanie artystów i architektów wzbudziły fraktale typu flame, tj. płomienne fraktale. Najczęściej mają one postać kłębiastych, kolorowych zwojów, płomieni, rulonów, szpulek, ślimaków, spiral i innych zdumiewających kształtów. Mogą zatem stanowić niepowtarzalne nadruki, wzory, desenie i motywy na tkaninach, tworzywach, surowcach i różnych materiałach. Struktury fraktalne wykorzystuje się również do generowania dwuwymiarowych i trójwymiarowych tekstur, dzięki którym przedmioty zyskują unikatowe kolory i faktury.

Animacje tworzone w oparciu o płomienne fraktale są szczególnie efektowne i finezyjne. Często prezentują obracające się elastyczne kształty, abstrakcyjne zwoje i pętle świecących się włókien. Animacja fraktalna jest narzędziem, które uruchamia komunikację z drugim człowiekiem na zupełnie innym, nieziemskim poziomie, np. flame-animacja Glenna Marshalla (zwycięzca Prix Ars Electronica) przenosi widza do fantastycznego świata, w którym organizmy podobne roślinom kiełkują, wzrastają, rozwijają się, kwitną, jałowieją i obumierają. Płomienie fraktale są używane przez artystów i designerów w obrazach, filmach, książkach, reklamach oraz różnorodnych sztukach multimedialnych.

W tej pracy przedstawimy metody tworzenia fraktali typu flame w programie Apophysis. Następnie zaprezentujemy proces projektowania animacji fraktalnej w systemie Flam3. W trakcie renderowania animacji będziemy wykorzystywać technikę keyframingu, bazującą na klatkach (ramkach) kluczowych i pośrednich.

W ostatniej części pracy przedstawimy oryginalny przykład animowanego flame fraktala, który z jednej strony obrazuje złożoność matematycznych struktur, a z drugiej strony charakteryzuje się finezyjnym pięknem przypadkowej kompozycji. Jednakże w tej fraktalnej strukturze odnajdujemy wysokie samopodobieństwo oraz symetrię. Fraktalne zwoje i płomienie nie są zanadto monotonne i regularne, ale też nie dręczą całkowitym bezładem i mogą stanowić ciekawy nadruk na jedwabnej tkaninie i papeterii.

## 2. MATEMATYCZNE I ALGORYTMICZNE PODSTAWY GENEROWANIA FRAKTALI TYPU FLAME

Podstawy tworzenia płomiennych fraktali przedstawili Scott Draves oraz Erik Reckase w pracy [1]. W 1992 roku S. Draves zamieścił w internecie pierwszą wersją algorytmu generowania flame fraktali (tzw. flame algorytm), który był stosowany w systemach GIMP, Photoshop oraz Ultrafractal. Bazą projektowania płomiennych fraktali są odwzorowania zwężające. Odwzorowanie F jest zwężające w przestrzeni $R^2$ , jeśli $F(R^2) \subseteq R^2$ oraz jeżeli dla wszystkich $x_1$, $x_2 \in R^2$ spełniony jest warunek Lipschitza $\rho(F(x_1), F(x_2)) \leq \lambda\rho(x_1, x_2)$, gdzie $\rho$ jest odległością w $R^2$, a $\lambda \in (0,1)$ stałą Lipschitza. Zbiór k różnych odwzorowań zwężających $F_0$, $F_1$,…, $F_{k-1}$ na przestrzeni $R^2$ nazywamy iterowanym układem funkcji (w skrócie IFS) i oznaczamy przez $\{R^2, F_0, \ldots, F_{k-1}\}$. Niech $\Psi(R^2, h)$ będzie przestrzenią domkniętych i ograniczonych podzbiorów płaszczyzny z odległością Hausdorffa. IFS generuje odwzorowanie *W* przestrzeni $\Psi(R^2, h)$ dane wzorem

$$W(A) = F_o(A) \cup F_1(A) \cup \ldots \cup F_{k-1}(A) \qquad A \in \Psi(R^2, h)$$

Odwzorowanie *W* jest zwężające w przestrzeni metrycznej zupełnej i na podstawie twierdzenia Banacha posiada punkt stały, który zgodnie z terminologią wprowadzoną przez M. Barnsleya nazywamy atraktorem układu IFS. Flame fraktal jest właśnie takim atraktorem.

Każdemu z odwzorowań $F_j$ przyporządkowujemy prawdopodobieństwo $p_j$. Nowy układ oznaczamy przez $\{R^2, F_0, F_1, \ldots, F_k ; p_0 , p_1 , \ldots , p_{k-1}\}$ i nazywamy układem IFS z prawdopodobieństwami, przy czym zakładamy, że $p_0 + p_1 + \ldots + p_{k-1} = 1$.

Rozpatrujemy następującą „grą w chaos". Wybieramy dowolny punkt $x_0 \in R^2$ i losujemy jeden z numerów 0, 1, …, k-1 tak, aby prawdopodobieństwo wylosowania numeru j wynosiło $p_j$. Przekształcamy punkt $x_0$ odwzorowaniem o wybranym numerze j oraz rysujemy na ekranie komputera punkt $x_1 = F_j(x_0)$. W taki sam sposób tworzymy następne punkty iteracyjne $x_1$ , $x_2$ , …, $x_n$ , …tj. punkt $x_{n+1}$ jest obrazem punktu $x_n$ przekształconym przez odwzorowanie $F_j$ o numerze j wylosowanym z prawdopodobieństwem $p_j$ . Jeżeli punkt $x_0$ należał do atraktora układu, to wszystkie punkty $x_1$ , $x_2$ , …, $x_n$ , … również należą do tego atraktora. W przeciwnym wypadku punkty te zbliżają się do atraktora z prędkością ciągu geometrycznego. Wystarczy, więc pominąć kilkadziesiąt początkowych wyrazów, aby z dokładnością określoną przez rozdzielczość ekranu, następne punkty znalazły się już w obrazie atraktora. Punkty $x_1$ , $x_2$ , …, $x_n$ , … zazwyczaj „chaotycznie błądzą po atraktorze". Dobrze jest wybierać punkt początkowy, który należy do atraktora, na przykład punkt stały dowolnego z odwzorowań $F_j$.

Probabilistyczny algorytm układu IFS, zwany „grą w chaos", jest następujący ([1], str. 3):

*(x, y) = losowo wybrany punkt z kwadratu [-1,1] × [-1, 1]*
*powtarzaj {*
*j losowy wybór numeru spośród 0, 1, …, k-1*
*(x, y) = $F_j$(x, y)*
*rysuj (x, y) } // oprócz pierwszych 20 iteracji*
*}*

Udowodniono, że prawie każda realizacja powyższego algorytmu tworzy ciąg punktów wypełniający atraktor, bowiem zależy on tylko od odwzorowań $F_0$, $F_1$, …, $F_{k-1}$ i nie zależy od prawdopodobieństw $p_0$ , $p_1$ , … , $p_{k-1}$. Prawdopodobieństwa te decydują jednak o gęstości rozkładu punktów na atraktorze ([2], str. 25-43).

W flame algorytmie odwzorowania $F_j$, j=0,1,…,k-1 zdefiniowano następującym wzorem:

$$(1) \qquad F_j = \sum_i v_{ij} V_i \left(a_j x + b_j y + e_j , c_j x + d_j y + f_j\right)$$

gdzie $V_i$: $R^2 \rightarrow R^2$ są różnorodnymi odmianami (funkcjami), $v_{ij}$ to współczynniki, natomiast $a_j$, $b_j$, $c_j$, $d_j$, $e_j$, $f_j$ są parametrami afinicznymi, tzn. $a_j$, $b_j$, $c_j$, $d_j$ - współczynniki przekształcenia linowego, $e_j$, $f_j$ - parametry translacji. Obszerna lista odmian została zamieszczona w pracy ([1], załącznik – katalog odmian), aczkolwiek nieustannie powstają nowe propozycje funkcji.

W 2004 roku M. Townsend opublikował specjalne oprogramowanie *Apophysis* (na licencji GPL) stanowiące implementację flame algorytmu. Ten generator ciągle jest rozbudowywany i wciąż powstają jego nowe wersje (*Aphophysis* dla Windows, *Oxidizer* dla Macintosh, *Qosmic* dla Linux, *Apple 3D* i *FrOs*t dla wszystkich platform).

W systemie *Apophysis* rozbudowano algorytm „gry w chaos" o możliwość dołączania przekształcenia $F_{final}$ finalnego (końcowego) i kolorów. Przekształcenie finalne nie podlega procedurze losowania i zawsze zostaje wykonane, przy czym każdemu fraktalowi można przyporządkować tylko jedno $F_{final}$. Oznaczmy przez $c_j$, $c_{final}$ początkowe wartości koloru odwzorowań $F_j$ oraz $F_{final}$ odpowiednio.

Rozszerzona procedura „gry w chaos", zwana flame algorytmem, przedstawia się następująco ([1], str. 9):

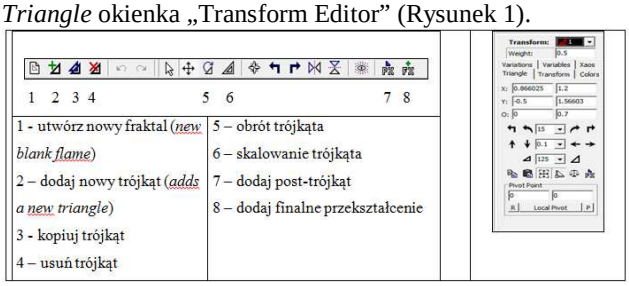
*(x, y) - losowo wybrany punkt z kwadratu [-1,1] × [-1, 1]*
*c - losowo wybrana wartość koloru*
*powtarzaj {*
*j - losowy wybór numeru spośród 0, 1, …, k-1*
$(x, y) = F_j(x, y)$
$c = (c+c_j)/2$

$(x_f, y_f) = F_{final}(x, y)$
$c_f = (c+c_{final})/2$
*rysuj* $(x_f, y_f, c_f)$ *} // oprócz pierwszych 20 iteracji*
*}*

Efektem działania tego algorytmu jest flame fraktal.

## 3. PROGRAMOWANIE FRAKTALI TYPU FLAME W SYSTEMIE *APOPHYSIS*

Programowanie płomiennego fraktala rozpoczyna się od zdefiniowania naszego własnego układu IFS. Zgodnie ze wzorem (1) wybieramy odmiany $V_i$, współczynniki $v_{ij}$, parametry afiniczne $a_j$, $b_j$, $c_j$, $d_j$, $e_j$, $f_j$, prawdopodobieństwa (wagi) $p_j$ oraz wartości kolorów $c_j$. W celu wprowadzenia powyższych danych tworzymy skrypt w okienku „Script Editor" systemu *Apophysis*. Ta metoda wymaga chociażby pobieżnej znajomości języka skryptowego. Jest on podobny do innych języków programowania. Drugi sposób definiowania naszego układu IFS polega na wprowadzeniu danych do odpowiednich formularzy, które znajdują się w zakładkach: *Variation*, *Variables*, *Colors*, *Transform*, *Triangle* okienka „Transform Editor" (Rysunek 1).




Rysunek 1. Pasek narzędziowy i zakładki w okienku „Transform Editor"

Definiowanie układu IFS rozpoczynamy od wyboru ikonki 1 (*new blank flame*) z paska narzędziowego. Dla każdego odwzorowania $F_j$ w zakładce *Variations* wybieramy odmiany $V_i$ oraz przypisujemy wartości współczynników $v_{ij}$. Zakładka *Variables* służy do określenia parametrów odmian $V_i$. W zakładce *Colors* dokonujemy wyboru wartości koloru oraz szybkości przenikania kolorów. Parametry a-f przekształcenia afinicznego $(x, y) \rightarrow (ax+by+e, cx+dy+f)$ wpisujemy do tabelki znajdującej się w zakładce *Transform.*

| Artykuł I. X | a | c |
|---|---|---|
| Y | b | d |
| O | e | f |

Wielkości a-f możemy również określić w zakładce *Triangle.* Otóż, z każdym odwzorowaniem $F_j$ stowarzyszony jest trójkąt o wierzchołkach X, Y, O widoczny w okienku „Transform Editor". Dowolna zmiana lokalizacji punktów X, Y, O definiuje obrót, translację, skalowanie w zakładce *Triangle.*

Cały proces tworzenia fraktala obserwujemy w okienku głównym programu *Apophysis*. Wygenerowany obiekt możemy poddać modyfikacji w okienku „Mutation". Kolorystykę i lokalizację fraktala ustalmy w zakładce „Gradient" (ponad 700 modeli), a wartości jaskrawości, nasycenia, kontrastu barw precyzujemy w zakładce „Rendering" okienka „Adjust". Możemy także zmienić tło (czarne-domyślne).

## 4. PROGRAMOWANIE ANIMACJI FRAKTALNEJ W SYSTEMIE FLAM3 Z WYKORZYSTANIEM SKRYPTU „TW_MORPHPRVREN"

System *Flam3* oraz skrypt „tw_morphPrvRen.asc" programują animację fraktalną techniką keyframingu, bazującą na klatkach (ramkach) kluczowych i pośrednich. Tą metodą „dynamizuje się" między innymi takie przekształcenia, jak przesunięcie, obracanie, skalowanie i zgniatanie. Skrypt generuje ramki pośrednie na podstawie informacji o animacji zawartych w kolejnych ramkach kluczowych. Ramki pośrednie są tworzone metodą interpolacji liniowej (ustawienie domyślne) lub gładkiej (co najmniej 4 ramki kluczowe). Dla każdej ramki pośredniej program przekształca dwa obrazy tak, by pasowały do siebie równocześnie zlewając je ze sobą. Przekształcanie interpolacyjne jest kombinacją deformowania, wstawiania i zlewania. W kolejnych krokach obraz jest coraz mniej podobny do oryginału, coraz bardziej za to upodobniając się do obrazu końcowego.

W okienku „Preview" systemu *Apophysis* możemy oglądać zaprogramowaną animację. *Flam3* pozwala wykonać jej renderowanie. Wszystkie klatki są zapisane jako pliki graficzne, standardowo z rozszerzeniem „png". Teraz wystarczy uruchomić aplikację do tworzenia animacji, wczytać w żądanej kolejności pliki graficzne, ustawić czas wyświetlania, dodać wizualne efekty oraz podkład muzyczny lub dźwiękowy i już można obserwować ewoluujące ożywione struktury fraktalne. Do wykreowania animacji fraktalnej często wykorzystuje się programy takie, jak Sony Vegas 7, Blender, AniGIF oraz Adobe After Effects, który wytycza nowe standardy w dziedzinie produkcji oraz tworzenia ruchomych grafik z efektami specjalnymi.

Programowanie animacji rozpoczynamy od utworzenia w systemie *Apophysis* kilku flame fraktali, które będą stanowiły klatki kluczowe. Zalecane jest wykreowanie nie więcej niż 6 płomiennych fraktali. W następnym kroku uruchamiamy skrypt „tw_morphPrvRen.asc" i wprowadzamy następujące dane dotyczące renderowania: *Flame Name, Output path: (domyślna: c:\flam3\anims\), Render Width, Render Height, Render Quality, Filter Radius, Oversample, Filetype (domyślny format: png) oraz „Number of frames per Animation stage"* (tj. liczba klatek pośrednich, domyślna wartość= 100). Rozmiary fraktalnego obrazu ustalamy w okienku parametrów*: Render Width* oraz *Render Height* (domyślna wartość = 512). Trzy parametry *Render Quality, Oversample* oraz *Filter Radius* determinują fizyczną jakość renderowania. Działają podobnie jak antialising, który zmniejsza błędy zniekształceniowe obrazu. Parametr *Render Quality* określa gęstość wyświetlanych punktów, która jest proporcjonalna do liczby iteracji flame algorytmu (domyślna wartość = 20). *Filter Radius* jest miarą natężenie filtru rozmycia obrazu podczas etapu końcowego renderowania. W wielu przypadkach dobrze sprawdza się wartość domyślna z 0.4 ale można ją redukować, aby uzyskać ostrzejszy rysunek. *Oversampling* jest linearnym mnożnikiem dla wymiarów renderowanego fraktala. Następnie obraz jest skalowany do wymiarów ustalonych przez parametry *Render Width* oraz *Render Height*. Zapewnia to lepszą jakość fraktala, aczkolwiek zwiększa rozmiar pliku graficznego.

**Przykład: animacja fraktalna (3 klatki kluczowe, 150 klatek pośrednich)**

Najpierw utworzymy 3 klatki kluczowe. Pierwszą klatką kluczową może być dowolny flame fraktal wykreowany w okienku „Transform Editor" lub zaprogramowany za pomocą skryptu. W tym przykładzie wykorzystamy plik *apo.flame*, który tworzy płomienny fraktal wygenerowany przez odwzorowania: *linear*[1], *sinusoidal*[2], *heart*[3], *spherical*[4]. Plik *apo.flame* zamieszczamy w załączniku.

Po uruchomieniu programu *Apophysis-2.09.exe* należy otworzyć plik *apo.flame* i zapisać go jako pierwszą klatkę kluczową za pomocą poleceń: *File, Save Parameters*. Po wykonaniu tych czynności pierwsza klatka kluczowa (*apo_01*) ukazuje się w lewej części okna (patrz Rysunek 2).

Drugą klatkę kluczową (*apo_02*) stanowi grafika fraktalna wykreowana przez trzy odwzorowania $F_0$, $F_1$, $F_2$. Odwzorowania należy zdefiniować w okienku „Transform Editor" (po wyborze ikonki „New blank flame" (Rysunek 1) dla $F_0$ oraz ikonki „Adds a new triangle" dla $F_1$, $F_2$).

Odwzorowanie $F_0$ ma następujące wartości: *linear = 0.95, spherical = 0.05, Weight = 0.3* oraz parametry $a_0 - f_0$ przekształcenia afinicznego ustalone następująco:

| | | |
|---|---|---|
| X | 0.132515 | -0.494554 |
| Y | 0.494554 | 0.132515 |
| O | -0.2 | -0.3 |

W zakładce „Colors" podajemy liczby 1.0 oraz 0.2 odpowiednio dla parametrów *Transform color* oraz *Color speed*. Odwzorowanie $F_1$ ma wartości: *linear = 0.95, spherical = 0.05, Weight = 0.9* oraz parametry $a_1 - f_1$ przekształcenia afinicznego ustalone następująco:

[1] V (x, y) = (ax+by, cx+dy), a, b, c, d parametry liniowe

[2] V(x, y) = (sinx, siny)

[3] V(x, y) = r ·(sin(Θr), -cos(Θr)), $r = \sqrt{x^2 + y^2}$, Θ = arctan(x/y)

[4] V(x,y) = (1/2)·(x, y)

| | | |
|---|---|---|
| X | 1.20741 | 0.323524 |
| Y | -0.323524 | 1.20741 |
| O | -0.2 | 0.2 |

W zakładce "Colors" ustalamy wartości koloru i szybkość przenikania, tzn. *Transform color = 1.0 Color speed = 0.5*
Odwzorowanie $F_2$ ma wartości: *linear = 0.95, spherical = 0.05, Weight = 0.3* oraz parametry $a_2 - f_2$ przekształcenia afinicznego określone następująco:

| | | |
|---|---|---|
| X | 0 | -0.32768 |
| Y | 0.32768 | 0 |
| O | -0.3 | -0.7 |

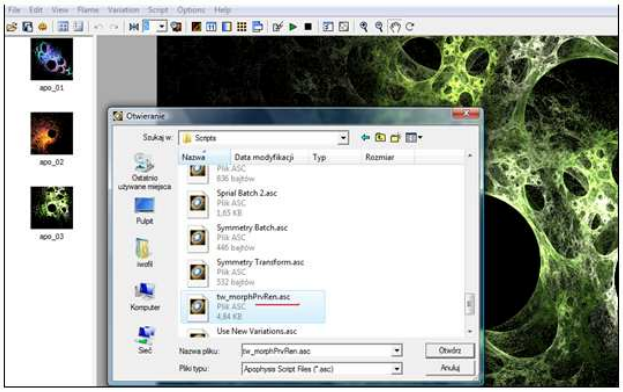
W zakładce "Colors" wprowadzamy: *Transform color = 0.79* oraz *Color speed = 0.*
Można jeszcze dokonać zmian kolorystycznych lub wariacyjnych poprzez opcje *View – Gradient* lub *View - Mutation*. Teraz zarejestrujemy nasz fraktal jako drugą klatkę kluczową animacji (*apo_02* , Rysunek 2). Teraz przystępujemy do programowania trzeciej klatki kluczowej (*apo_03*), która zostanie wykreowana przez dwa odwzorowania $F_0, F_1$.
Odwzorowanie $F_0$ ma wartości: *linear = 0.94, spherical = 0.06, Weight = 0.5* oraz parametry $a_0 - f_0$ przekształcenia afinicznego ustalone następująco:

| | | |
|---|---|---|
| X | -0.987056 | -0.160383 |
| Y | -0.160384 | 0.987054 |
| O | 0.76612 | 0.89481 |

Odwzorowanie $F_1$ ma wartości: *linear = 0.95, spherical = 0.05, Weight = 0.9* oraz parametry $a_1 - f_1$ przekształcenia afinicznego ustalone następująco:

| | | |
|---|---|---|
| X | -0.16938 | 0.662123 |
| Y | -0.562743 | -0.044169 |
| O | 1.14694 | -0.160617 |

W zakładce "Colors" wprowadzamy *Transform color = 1.0* Teraz rejestrujemy nasz fraktal jako trzecią klatkę kluczową animacji (*apo_03*, Rysunek 2).
Aby obserwować zaprogramowaną animację wystarczy uruchomić skrypt „tw_morphPrvRen". Wybierzmy opcje: *Script – Open* i otwórzmy plik „tw_morphPrvRen.asc" (Rysunek 2), a następnie *Script – Run „tw_morphPrvRen.asc"*. Automatycznie pojawi się okienko "Parameter Input" do którego należy wpisać „1" i nacisnąć „OK". Wówczas zostanie uruchomiony „mały" podgląd animacji fraktalnej.



Rysunek 2. Lista klatek kluczowych (*apo_01*, *apo_02*, *apo_03*)

W systemie *Flam3* wszystkie klatki kluczowe oraz pośrednie animacji są zapisywane w postaci plików graficznych, domyślnie z rozszerzeniem *png*. Ten proces rozpoczynamy od ponownego uruchomienia skryptu "tw_morphPrvRen", a do okienka „Parameter Input" wpisujemy cyfrę 2. Dalej, ustalamy następujące wartości parametrów renderowania animacji: *Render Width = 500, Render Height = 500, Render Quality = 20, Filter Radius = 0.4, Oversample = 1, Filetype = png, Number of frames per Animation stage = 50, Flam3 software path = c:\flam3*. Po zakończeniu działania skryptu w folderze *anims (C:\flam3\anims)* pojawią się dwa pliki *apo.flame* oraz *apo-animate.bat*. Aby rozpocząć renderowanie dwukrotnie klikamy plik *apo-animate.bat.* Jeśli wybraliśmy 3 klatki kluczowe i 50 klatek pośrednich między dwiema klatkami kluczowymi, to uzyskamy 150 wszystkich klatek animacji. Czas ich renderowania to ok. 3 godz. Po tym okresie komplet klatek zostaje zapisany w folderze *anims* (*C:\flam3\anims*), jako *apo_0000.png, apo_0001.png, ....., apo_0149.png* Teraz wystarczy uruchomić program do tworzenia animacji, wczytać pozyskane pliki graficzne, określić czas wyświetlania, dodać efekty specjalne, ustalić parametry zapisu i już można obserwować pięknie ewoluujące ożywione struktury fraktalne.

## 5. PODSUMOWANIE

Algorytm generowania flame fraktali należy do projektu przetwarzania rozproszonego Electric Sheep, którego celem jest wypracowanie uniwersalnego wygaszacza ekranu. Projekt stanowi abstrakcyjne dzieło sztuki zapoczątkowane przez Scotta Dravesa. Ten wygaszacz współtworzą tysiące internautów z całego świata. Ich komputery łączą się by razem tworzyć ewoluujące abstrakcyjne animacje zwane

„owcami". Każdy widz obserwujący obrazy na ekranie może głosować na ulubione animacje. Im bardziej popularna „owca", tym dłużej może żyć, rozmnażać się i przekazywać swoje geny. W ten sposób projekt Elelectric Sheep ewoluuje by zachwycać publiczność. Można także zaprojektować własną fraktalną animację i umieścić ją w projekcie puli gentypów. Programowanie i animowanie fraktali nie tylko obrazuje złożoność matematycznych struktur, ale także przyczynia się do odkrywania unikatowych projektów designerskich, szczególnie w dziedzinach artystycznych, wzornictwie przemysłowym i sztuce użytkowej.

**Literatura**

[1] Draves S.: Reckase E.: *The Fractal Flame Algorithm*, 2001, 2008, *http://en.wikipedia.org/wiki/Fractal_flame*
[2] Kudrewicz J.: *Fraktale i chaos*, Wydawnictwa Naukowo-Techniczne, Warszawa 2007.
[3] Peitgen H.O., Jürgens H., Saupe D.*: Granice Chaosu. Fraktale*, Wydawnictwo Naukowe PWN, Warszawa 2002.

Załącznik

Skrypt *apo.flame* programujący pierwszą klatkę kluczową wygenerowaną przez $F_0$, $F_1$, $F_2$ oraz parametry renderowania.

1) programowanie $F_0$

```
Clear;
AddTransform
With Transform do
Begin
linear:= 1.0;
sinusoidal:= 0.05;
spherical:= 0.05;
Transform.a:=0.308448;
Transform.c:=-0.731461;
Transform.b:=-0.820557;
Transform.d:=-0.181328;
Transform.e:=0.672477;
Transform.f:=0.291451;
weight := 0.448416;
color:=0;
 End;
```

2) programowanie $F_1$

```
AddTransform
With Transform do
Begin
linear:= 1.0;
heart:=0.130544;
spherical:= 0.05;
Transform.a:= 0.343699;
Transform.c:=0;
Transform.b:= 0.028065;
Transform.d:=-0.527663;
Transform.e:= 0.333806;
Transform.f:=0.91862;
 weight := 0.171914;
color:=0.5;
 End;
```

3) programowanie $F_2$

```
AddTransform
With Transform do
Begin
linear:= 1.0;
spherical:= 0.05;
Transform.a:= -0.412516;
Transform.c:= -0.538064;
Transform.b:= 0.350117;
Transform.d:= -0.881458;
Transform.e:= -0.632035;
Transform.f:= 1.470529;
weight := 0.37967;
color:=1;
 End;
```

4) parametry renderowania

```
RotateFlame(122.33);
Flame.gamma:= 2.5;
Flame.brightness:= 3;
Flame.SampleDensity := 1;
Flame.Oversample := 1;
Flame.FilterRadius:= 0.4;
Flame.Scale := 17.01;
Flame.Zoom := 0.816;
Flame.x := 0.121;
Flame.y := 0.371;
```

*Studia i Materiały Informatyki Stosowanej, Tom 4, Nr 7, 2012*
*str. 26-32*