Wykład wstępny (V) *<< zrealizowany w 2006, przedrostki-nazwy zretuszowane w Acrobat >>*

z Podstaw Przetwarzania Informacji

(środowisko Matlab 6.x 7.x, przykłady wizualizacji)

W ramach serii wstępnych wykładów dotyczących prezentacji możliwości dokonywania przekształceń na danych obrazów 2D w *Image Processing Toolbox*, postanowiono przedstawić pewne wstępne przykłady zastosowań omówionej grupy funkcji, jak również specyficznej składni poleceń w manipulacji na zawartości danych 1D/2D/3D. Na początek, postanowiono przedstawić przykładową postać kodu służącego w generacji fraktalnej obiektu przypominającego kształtem liść paproci (ang. *fern*):

```
%Artur Bernat all rights reserved
%15 April 2006, fern's fractal with stochastic transforms.
%[M,im_final]=fern(szx,szy,magnif,iter,nop);
%szy,szx <=dimensions of 2D map(output size doubled in x)
%magnif  <=magnification coeff. for the figure
%iter    <=number of iteraton,
%nop     <=number of intervals in getting of frames
%[M,img]=fern(150,300,30,95000,30); % default call params.
function [M,im_final]=fern(szx,szy,magnif,iter,nop);
A=[0.0 0.85 0.2 -0.15]; %linear transforms. coeffs.
B=[0.0 0.04 -0.26 0.28];
C=[0.0 -0.04 0.23 0.26];
D=[0.16 0.85 0.22 0.24];
E=[0.0 0.0 0.0 0.0];
F=[0.0 1.6 1.6 0.44];
step=iter/nop;
MAX_TRANSF=4;
mapsA=zeros(szy,szx); mapsB=zeros(szy,szx);
x=0;y=0;
nrp = ceil(MAX_TRANSF.*rand(1,iter));
[min(nrp(:)) max(nrp(:))]
% background;  1-8  ; 9nth control colour
paltsA=[0.1 0.25 0.30; 0 1 0;0.2 1 0; 0.5 1 0; 1 1 0;...
            1 0 0; 1 0 0; 1 0.2 0 ; 1 0.5 0 ; 1 1 0];
M=moviein(nop+2);
axis manual;
j=0;%control counter for catching of frames
for i=1:iter,
    kolor=uint8(nrp(i));
    xl=A(kolor).*x+B(kolor).*y+E(kolor);
    y=C(kolor).*x+D(kolor).*y+F(kolor);
    x=xl;
    Y=1+szy/16+magnif.*y;
    X=1+szx/2+magnif.*x;
    mapsA(uint16(1+Y.*sign(Y)),uint16(1+X.*sign(X)))=uint8(1+nrp(i));
    mapsB(uint16(1+Y.*sign(Y)),uint16(1+X.*sign(X)))=uint8(5+nrp(i));
    if mod(i,step)==0
     imshow([flipud(mapsA(1:szy,1:szx)) mapsB(1:szy,1:szx)],paltsA);
     axis([0 szx+szx 0 szy]); axis manual; j=j+1; [i j]
     M(:,j)=getframe;
    end;
end;
im_final=ind2rgb([flipud(mapsA(1:szy,1:szx)) mapsB(1:szy,1:szx)],paltsA);
movie(M);
movie2avi(M,'fern.avi','compression','Cinepak','FPS',12);
close;
```

Rys1Przykładowy kod `fern.m`, generacji obiektu fraktalnego formie podobnej do liścia paproci

Specyficznym rozwiązaniem powyższej funkcji generacji (rys.1) jest zastosowanie generacji zbioru punktów, przynależnych do tworzonego iteracyjnego obiektu, w oparciu o indeksowaną postać map `mapsA` oraz `mapsB`. Mapy te ulegają następnie złożeniu w użyciem polecenia `[]`, zarówno w czasie tworzenia animacji z procesu tworzenia obiektu fraktalnego, jak i w czasie ostatecznej generacji mapy `im_final`, po za globalną pętlą iteracyjną `for`. Takie rozwiązanie to możliwość bardziej elastycznego wykorzystania, definiowania, jak i

przedefiniowywania w locie palety kolorów. Alternatywne rozwiązanie, jest trudniejsze w implementacji, bo dla przykładu oznacza konieczność obsługi danych w ich reprezentacji RGB lub HSV.

Sam algorytm generacji obiektu fraktalnego, mimo, że nazywany jest algorytmem deterministycznej generacji, to z pewnych względów będzie miał zapewniony losowy zestaw warunków początkowych. Wędrówka punktu po płaszczyźnie XY, jest tutaj opisywana szczególnie prostą liniową kombinacją czynników i wartości bieżących współrzędnych:

$$x_{i+1} = A * x_i + B * y_i + E$$
$$y_{i+1} = C * x_i + D * y_i + F \quad (1)$$

Różnica w wartościach współczynników `A..F`, to inny również powszechnie znany z sylwetki generowanego obiektu, algorytm w następującej implementacji:

```
%Artur Bernat all rights reserved
%15 April 2006, drag's fractal with stochastic transforms.
%[M,im_final]=drag(szx,szy,magnif,iter,nop);
%szy,szx <=dimensions of 2D map(output size doubled in x)
%magnif  <=magnification coeff. for the figure
%iter    <=number of iteraton,
%nop     <=number of intervals in getting of frames
%[M,img]=drag(150,300,25,30000,30); % default call params.
function [M,im_final]=drag(szx,szy,magnif,iter,nop);
A=[ 0.824074  0.088272];%linear transform. coeffs.
B=[ 0.281482  0.520988];
C=[-0.212346 -0.463889];
D=[ 0.864198 -0.377778];
E=[-1.88229   0.78536];
F=[-0.110607  8.095795];
step=iter/nop;
MAX_TRANSF=2;
mapsA=zeros(szy,szx);mapsB=zeros(szy,szx);
x=0;y=0;
nrp = ceil(MAX_TRANSF.*rand(1,iter));
[min(nrp(:)) max(nrp(:))]
%background;  1-8 palette's color  ;
paltsA=[0.15 0.25 0.45; 0 1 0;0.2 1 0; 0.5 1 0; 1 1 0;...
              1 0 0; 1 0 0; 1 0.2 0 ; 1 0.5 0 ; 1 1 1];
M=moviein(nop+2);
axis manual;
j=0;%control counter for catching of frames
for i=1:iter,
    kolor=uint8(nrp(i));
    xl=A(kolor).*x+B(kolor).*y+E(kolor);
    y=C(kolor).*x+D(kolor).*y+F(kolor);
    x=xl;
    Y=(1+szy/16+magnif.*y);
    X=(1+szx/16+magnif.*x);
    mapsA(uint16(1+sign(Y).*Y),uint16(1+sign(X).*X))=uint8(1+nrp(i));
    mapsB(uint16(1+sign(Y).*Y),uint16(1+sign(X).*X))=uint8(5+nrp(i));
    if mod(i,step)==0
     imshow([flipud(mapsA(1:szy,1:szx)) mapsB(1:szy,1:szx)],paltsA);
     axis([0 szx + szx 0 szy]); axis manual; j=j+1; [i j]
     M(:,j)=getframe;
 end;
end;
im_final=ind2rgb([flipud(mapsA(1:szy,1:szx)) mapsB(1:szy,1:szx)],paltsA);
movie(M);
movie2avi(M,'drag.avi','compression','Cinepak','FPS',12);
close;
```

Rys2Przykładowy kod `drag.m`, generacji obiektu sylwetki smoka

Na pierwszy rzut oka, powyższa implementacja (rys.2) posiada dwukrotnie krótsze(!) w liczności swoich pod-elementów, wektory `A...F`. Historia powstania przepisu na konkretne wartości losowo dobieranych współczynników w zawartości wektorów `A..F`, jest raczej nieznana (przynajmniej powszechnie). Ważne jest to, że z reguły, jakiekolwiek losowo, czy

eksperymentalnie wprowadzane zmiany, powodują zupełną deterioracje albo wręcz rozbieżność zupełnie stabilnych algorytmów opartych o zależność (1). W końcu liczność pod-elementów wektorów `A..F` ustawiona na 3, ze specyficzną naprzemienną listą współczynników, to możliwość generacji słynnego trójkąta Sierpińskiego:

```
%Artur Bernat all rights reserved
%15 April 2006, triangle's fractal with stochastic transforms.
%[M,im_final]=triangle(szx,szy,magnif,iter,nop);
%szy,szx <=dimensions of 2D map(output size doubled in x)
%magnif  <=magnification coeff. for the figure
%iter    <=number of iteraton,
%nop     <=number of intervals in getting of frames
%[M,img]=triangle(150,300,130,10000,30); %default call params.
function [M,im_final]=triangle(szx,szy,magnif,iter,nop);
A=[ 0.5 0.5 0.5];%linear transform. coeffs.
B=[ 0   0   0  ];
C=[ 0   0   0  ];
D=[ 0.5 0.5 0.5];
E=[ 0   0   0.5];
F=[ 0   1   0.5];
step=iter/nop;
MAX_TRANSF=3;
mapsA=zeros(szy,szx);mapsB=zeros(szy,szx);
x=0;y=0;
nrp = ceil(MAX_TRANSF.*rand(1,iter));
[min(nrp(:)) max(nrp(:))]
%background;  1-8 palette's color  ;
paltsA=[0.15 0.25 0.45; 0 1 0;0.2 1 0; 0.5 1 0; 1 1 0;...
              1 0 0; 1 0 0; 1 0.2 0 ; 1 0.5 0 ; 1 1 1];
M=moviein(nop+2);
axis manual;
j=0;%control counter for catching of frames
for i=1:iter,
    kolor=uint8(nrp(i));
    xl=A(kolor).*x+B(kolor).*y+E(kolor);
    y=C(kolor).*x+D(kolor).*y+F(kolor);
    x=xl;
    Y=(1+szy/16+magnif.*y);
    X=(1+szx/16+magnif.*x);
    mapsA(uint16(1+sign(Y).*Y),uint16(1+sign(X).*X))=uint8(1+nrp(i));
    mapsB(uint16(1+sign(Y).*Y),uint16(1+sign(X).*X))=uint8(5+nrp(i));
    if mod(i,step)==0
     imshow([fliplr(mapsA(1:szy,1:szx)) mapsB(1:szy,1:szx)],paltsA);
     axis([0 szx + szx 0 szy]); axis manual; j=j+1; [i j]
     M(:,j)=getframe;
 end;
end;
im_final=ind2rgb([fliplr(mapsA(1:szy,1:szx)) mapsB(1:szy,1:szx)],paltsA);
movie(M);
movie2avi(M,'triangle.avi','compression','Cinepak','FPS',12);
close;
```

Rys3Przykładowy kod `triangle.m`, generacji trójkąta Sierpińskiego

We wszystkich powyższych implementacjach, zastosowano *ochronę* przed możliwością wykroczenia konturu generowanego obiektu fraktalnego po za wymiar fizyczny map `mapsA` i `mapsB`. Proste przemnożenie wartości nowo-wyliczanej współrzędnej (ujemnej/dodatniej w znaku) przez wyznaczany znak współrzędnej to możliwość wykreślania w zakresie nieujemnych i pośrednio niezerowych wartości współrzędnych (są to podwyrażenia typu: `1+sign(X).*X` oraz `1+sign(Y).*Y`).

Zasadniczo, zależy nam na czysto losowym lub prawie losowym starcie algorytmu. W tym celu można by na przykład zaimplementować nadrzędną funkcję wywołań pod-sesji kolejnych generacji obiektów fraktalnych, o stosunkowo małej liczności punktów generowanych, lecz również o losowym punkcie startu generowanych współrzędnych `x`,`y`.

W tym celu zaimplementowano skrypt *ferns.m*, który naprzemiennie wywołuje to funkcję w skrypcie *fern_rand.m*, to funkcję w skrypcie *fern_rand_jet.m*:

```
%Artur Bernat, auxilliary script for fern gens.
%function [imout]=ferns(size_x,size_y,magnif,iter,movieN,iterN)
%function [imout]=fernxn_xor(size_x,size_y,magnif,iter,movieN,iterN)
%size_x,size_y   <=input x,y size of bitmap to be generated
%magnif          <=input magnification coeff. for fern leaf
%iter            <=input iters. for each of subcall in fern_rand function
%movieN          <=input movieN frames' number per one avi film
%iterN           <=input global iteration in superior loop of generating
%[img_s_out]=ferns(1600,900,89,157300,3,4);%default call's parameters
%[img_s_out]=ferns(400,225,25,57300,3,4);%2nd set of call's parameters(fast)
function [imout]=ferns(size_x,size_y,magnif,iter,movieN,iterN)
[M,img]=fern_rand(size_x/2,size_y,magnif,iter,movieN,1);
imout=img;
state=0;
for i=1:iterN-1,
    state=bitxor(state,1);
    if state==1
     [M,img]=fern_rand(size_x/2,size_y,magnif,iter,movieN,1+i);
    else
     [M,img]=fern_rand_jet(size_x/2,size_y,magnif,iter,movieN,1+i);
    end;
%    imout=imadd(imout,img);
     for j=1:size_x,
         for k=1:size_y,
          if (img(k,j,1)>imout(k,j,1))
           imout(k,j,1)=img(k,j,1);
          end;
          if (img(k,j,2)>imout(k,j,2))
           imout(k,j,2)=img(k,j,2);
          end;
          if (img(k,j,3)>imout(k,j,3))
           imout(k,j,3)=img(k,j,3);
          end;
         end;
     end;
end;
%----------------------------------------------------------------%
```

Rys4Przykładowy kod nadrzędnych wywołań ferns.m, generacji liścia paproci

Kod implementacji w skrypcie *fern_rand.m* zasadniczo nie różni się od tego, przedstawionego na rys.1. Wartości startowe są dobierane losowo z zakresu [1,17] oraz [1,23] odpowiednio. Ze względu na losowy wybór, w każdej z iteracji pod-elementów z zawartości wektorów *A..F*, możliwe jest w pewnym sensie dopełnienie, tych peryferyjnych elementów liścia paproci, których być może nie udałoby się '*pozyskać*', przy nawet bardzo poszerzonej liczbie iteracji, w jednej globalnej sesji generacji, z użyciem skryptu *fern.m*. Wywołanie funkcji drugiego skryptu *fern_rand_jet.m*, to pewnego rodzaju algorytmiczna *spekulacja/koloryzacja/stylizacja* powierzchni liścia paproci z użyciem predefiniowanej/wbudowanej w środowisko Matlab (v6.5 i v7.1) palety kolorów o nazwie *jet*. Ogólnie, barwy zimne na wszelkiego rodzaju wykresach 3D, wykorzystuje się tutaj w oznaczaniu punktów niżej położonych, a barwy cieplejsze tej palety kolorów, to narzędzie w oznaczaniu punktów wyżej położonych według schematu kolorystycznego palety *JET*.

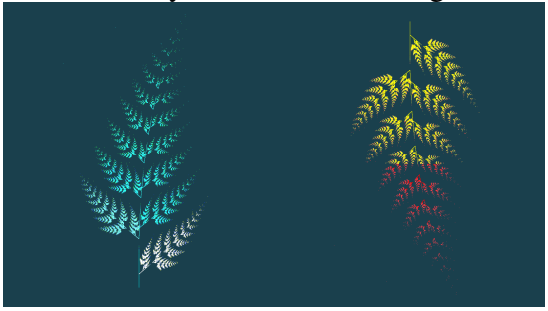
W takim razie (rys.6B) postanowiono 64-elementową paletę *JET*, podzielić na dwoje (31-elementów) i '*obdzielić*' jej zawartością kolorystyczną powierzchnię dwóch liści paproci stosownie do zmiany współrzędnej Y.

W efekcie końcowym w skrypcie nadrzędnym następuje swoistego rodzaju '*domieszkowywanie*' składowych RGB obrazów z wywołań skryptów podrzędnych.

```
%Artur Bernat all rights reserved
%15 April 2006, fern's fractal with stochastic transforms.
%[M,im_final]=fern_rand(szx,szy,magnif,iter,nop);
%szy,szx <=dimensions of 2D map(output size doubled in x)
%magnif  <=magnification coeff. for the figure
%iter    <=number of iteraton,
%nop     <=number of intervals in getting of frames
%[M,img]=fern_rand(150,300,30,95000,30); % default call params.
function [M,im_final]=fern_rand(szx,szy,magnif,iter,nop,Niter);
A=[0.0 0.85 0.2 -0.15]; %linear transforms. coeffs.
B=[0.0 0.04 -0.26 0.28];
C=[0.0 -0.04 0.23 0.26];
D=[0.16 0.85 0.22 0.24];
E=[0.0 0.0 0.0 0.0];
F=[0.0 1.6 1.6 0.44];
step=iter/nop;
MAX_TRANSF=4;
mapsA=zeros(szy,szx); mapsB=zeros(szy,szx);
x=23.*rand(1,1)
y=17.*rand(1,1)
nrp = ceil(MAX_TRANSF.*rand(1,iter));
[min(nrp(:)) max(nrp(:))]
% background;  1-8  ; 9nth control colour
paltsA=[0.1 0.25 0.30; 0 1 0;0.2 1 0; 0.5 1 0; 1 1 0;...
             1 0 0; 1 0 0; 1 0.2 0 ; 1 0.5 0 ; 1 1 0];
M=moviein(nop+2);
axis manual;
j=0;%control counter for catching of frames
for i=1:iter,
    kolor=uint8(nrp(i));
    xl=A(kolor).*x+B(kolor).*y+E(kolor);
    y=C(kolor).*x+D(kolor).*y+F(kolor);
    x=xl;
    Y=1+szy/16+magnif.*y;
    X=1+szx/2+magnif.*x;
    mapsA(uint16(1+Y.*sign(Y)),uint16(1+X.*sign(X)))=uint8(1+nrp(i));
    mapsB(uint16(1+Y.*sign(Y)),uint16(1+X.*sign(X)))=uint8(5+nrp(i));
    if mod(i,step)==0
     imshow([flipud(mapsA(1:szy,1:szx)) mapsB(1:szy,1:szx)],paltsA);hold on;
     title(['Global iter: ' int2str(Niter) ', local inter no. ' int2str(j)]);
     axis([0 szx+szx 0 szy]); axis manual; j=j+1; [i j]
     M(:,j)=getframe;
    end;
end;
im_final=ind2rgb([flipud(mapsA(1:szy,1:szx)) mapsB(1:szy,1:szx)],paltsA);
movie(M);
RandName=100+899*rand(1,1);
movie2avi(M,['fernxn_rand' int2str(RandName) '.avi'],'compression','Cinepak','FPS',3);
disp(['Number of superior calling iteration: ' int2str(Niter)]);
close;
```

Rys5 Przykładowy kod poddrzędnych wywołań `ferns_rand.m`, generacji liścia paproci

Kod przedstawiony na rys.6 jest nieco poszerzony o obliczenia bieżącego koloru palety `JET`, przynależnego danemu przedziałowi wysokości na obrazie generowanym:



Rys6A Paleta `JET` domieszkowana inną okrojoną paletą kolorów skutkuje nieoczekiwanie załamaniem się ciągłości barw po prawej stronie

```
%Artur Bernat all rights reserved
%15 April 2006, fern's fractal with stochastic transforms.
%AND randomized starting x0,y0 point of calculations
%[M,im_final]=fern_rand_jet(szx,szy,magnif,iter,nop,Niter);
%szy,szx <=dimensions of 2D map(output size doubled in x)
%magnif  <=magnification coeff. for the figure
%iter    <=number of iteraton,
%nop     <=number of intervals in getting of frames
%Niter   <=number of superior call iteration(just info)
%[M,img]=fern_rand_jet(150,300,30,95000,30,1); % default call params.
function [M,im_final]=fern_rand_jet(szx,szy,magnif,iter,nop,Niter);
A=[0.0 0.85 0.2 -0.15]; %linear transforms. coeffs.
B=[0.0 0.04 -0.26 0.28];
C=[0.0 -0.04 0.23 0.26];
D=[0.16 0.85 0.22 0.24];
E=[0.0 0.0 0.0 0.0];
F=[0.0 1.6 1.6 0.44];
step=iter/nop;
MAX_TRANSF=4;
mapsA=zeros(szy,szx); mapsB=zeros(szy,szx);
x=23.*rand(1,1);%randomized starting x0
y=17.*rand(1,1);%randomized starting y0
nrp = ceil(MAX_TRANSF.*rand(1,iter));
[min(nrp(:)) max(nrp(:))]
% background;  1-8  ; 9nth control colour
%paltsA=[0.1 0.25 0.30; 0 1 0;0.2 1 0; 0.5 1 0; 1 1 0;...
%             1 0 0; 1 0 0; 1 0.2 0 ; 1 0.5 0 ; 1 1 0];
paltsA=JET;    %64-graded JET predefined palette;
paltsA(1,:)=0; % first from jet palette is equal to zero
palstepY=szy/31;
M=moviein(nop+2);
axis manual;
j=0;%control counter for catching of frames
for i=1:iter,
    kolor=uint8(nrp(i));
    xl=A(kolor).*x+B(kolor).*y+E(kolor);
    y=C(kolor).*x+D(kolor).*y+F(kolor);
    x=xl;
    Y=1+szy/16+magnif.*y;
    X=1+szx/2+magnif.*x;
    mapsA(uint16(1+Y.*sign(Y)),uint16(1+X.*sign(X)))=uint8(1 +mod( ( (1+Y.*sign(Y)) /palstepY ),31));
    mapsB(uint16(1+Y.*sign(Y)),uint16(1+X.*sign(X)))=uint8(32+mod( ( (1+Y.*sign(Y)) /palstepY ),31));
    if mod(i,step)==0
     imshow([flipud(mapsA(1:szy,1:szx)) mapsB(1:szy,1:szx)],paltsA);hold on;
     title(['Global iter: ' int2str(Niter) ', local inter no. ' int2str(j)]);
     axis([0 szx+szx 0 szy]); axis manual; j=j+1; [i j]
     M(:,j)=getframe;
    end;
end;
im_final=uint8(ind2rgb([flipud(mapsA(1:szy,1:szx)) mapsB(1:szy,1:szx)],paltsA));
movie(M);
RandName=100+899*rand(1,1);
movie2avi(M,['fernxn_rand_jet' int2str(RandName) '.avi'],'compression','Cinepak','FPS',3);
disp(['Number of superior calling iteration: ' int2str(Niter)]);
close;
```

Rys6B Przykładowy kod poddrzędnych wywołań `ferns_rand_jet.m`, generacji liścia paproci

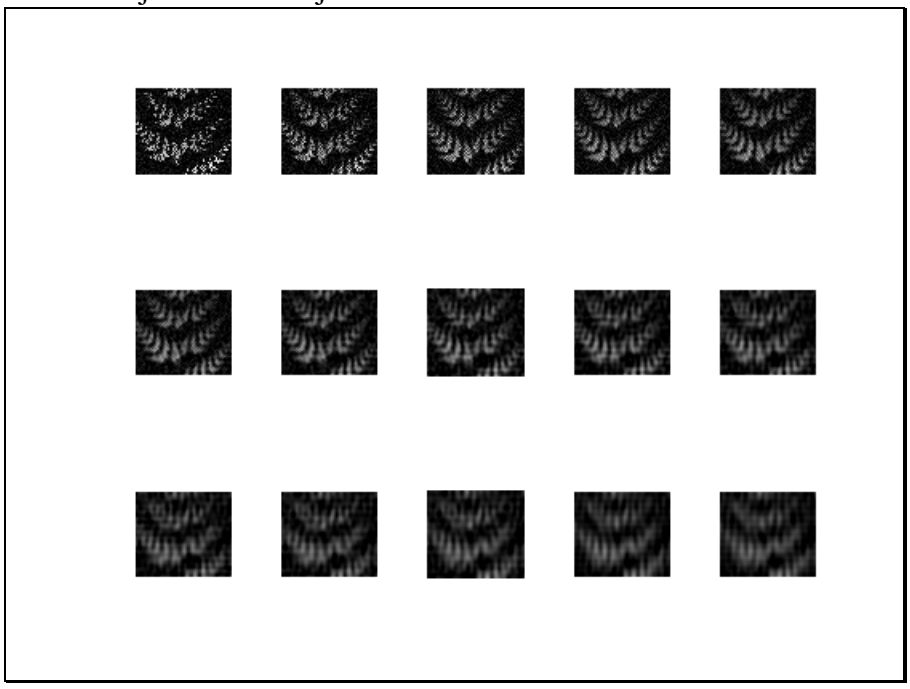
Podsumowując, tą część bieżącego wykładu (wykład V), można powiedzieć, że nie istnieje prosty przepis na przejście od praktyki programistycznej w języku C/C++, na sposób programowania, obsługi zmiennych i całość specyfiki pracy w środowisku Matlab.
Również pierwsze, wstępne przeświadczenie o przydatności tej i czy innej funkcji w wytyczonym zadaniu, z reguły wymaga weryfikacji. Również w tym przypadku, to jest tworzenia specyficznej mieszaniny barw danych uzyskanych z dwóch różnych funkcji generowania tego samego obiektu fraktalnego, lecz z różniącymi się paletami barw, z pozoru wystarczyłoby wywołanie funkcji `imadd`, a treści dwóch obrazów zostałyby zsumowane. Praktyka jednak, czasami każe wkraczać samemu w realizację drobnych zadań pomocniczych względem zadania głównego.
Nawiązując do treści wykładów IVA/B, postanowiono przedstawić poniżej skrypt demonstrujący postępującą degradację zbioru informacji oryginalnego obrazu intensywności luminancji, jako funkcję zawężanego zbioru współczynników prostej transformaty kosinusowej. Zazwyczaj, to czy inne demo w formie interaktywnej może posłużyć w celach dydaktycznych. Tym razem jednak, poniżej przedstawiona treść skryptu `dctmovie`, tworzy dwie animacje zapisywane na dysku twardym. Jedna obrazuje zmienną jakość obrazu po

wykonanym prostym przekształceniu w oparciu o:a) transformatę kosinusową (kompresja stratna), b)redukcję zbioru współczynników, c)transformatę odwrotną kosinusową:

```
%all rights rewerved Artur Bernat
%a script for DCT transf. deterioration of image quality
%function [Min,Mcoeff,img]=dctmovie(im,nsteps,limit);
%im_in=imread('rice.png');  %example of image
% [M1,M2,ims]=dctmovie(im_in,20,300); %default call params
function [Mim,Mcoeff,img]=dctmovie(im,nsteps,limit);
%im=rgb2gray(im);
szx=size(im,2);
szy=size(im,1);
dct_im=dct2(im);
Mim=moviein(nsteps);
Mcoeff=moviein(nsteps);
interv=limit/nsteps;
for i=1:nsteps,
 dct_im(abs(dct_im)<i*interv)=0;
 dct_bw=zeros(szy,szx);
 dct_bw(abs(dct_im)>0)=1;
 idct_im=idct2(dct_im)/255;
 subplot(2,1,1);imshow(idct_im);hold on;
 title(['Frame nr. ' int2str(i) ' ,thresh: ' int2str(i*interv)]);
 Mim(:,i)=getframe;
 subplot(2,1,2);imshow(dct_bw);
 Mcoeff(:,i)=getframe;
end;
img=idct_im;
RandName=100+899*rand(1,1);
movie2avi(Mim,   ['dctmovie' int2str(RandName) '.avi'],'compression','Cinepak','FPS',9);
close;
movie2avi(Mcoeff,['dctcoeffs' int2str(RandName) '.avi'],'compression','Cinepak','FPS',9);
close;
```

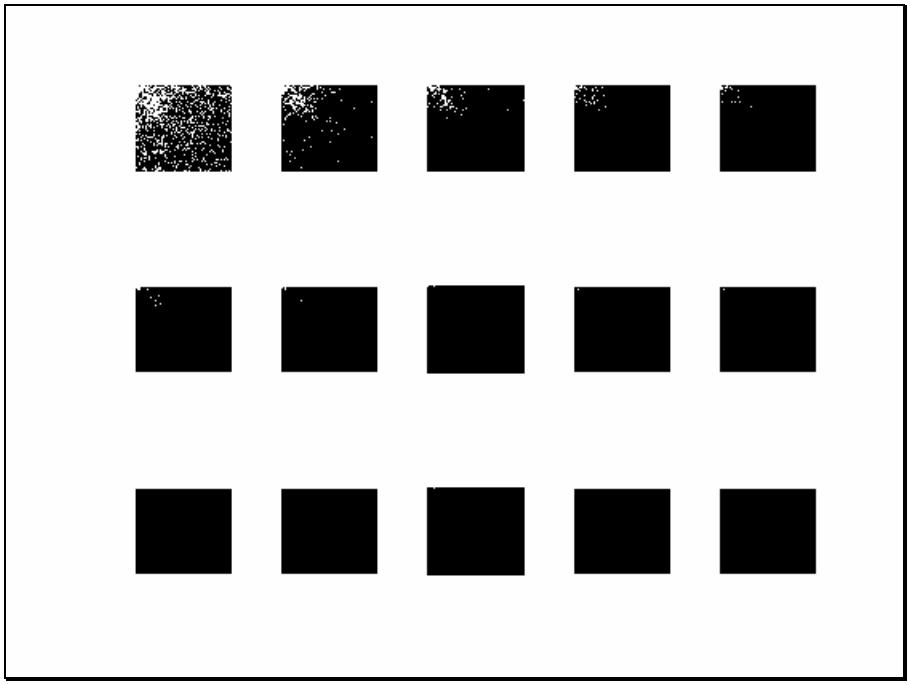
Rys7 Przykładowy kod `dctmovie.m`, składnia filmu obrazującego postępującą degradację treści obrazu

Druga animacja to mapa zbinaryzowanych współczynników transformaty kosinusowej, obrazująca udział wybranych współczynników transformaty w tworzeniu obrazu w realizacji odwrotnej transformacji kosinusowej.



Rys 8 Piętnaście klatek animacji w formie pod-wykresów jako wynik wykonania skryptu o treści z rysunku 7

Na rys.8, powyżej pokazano degradację treści pewnego obrazu intensywności luminacji z górną granicą wartości współczynników transformaty kosinusowej ustawioną na 600, przy rozdzielczości obrazu rzędu 300x330 pikseli.



Rys 9 Piętnaście klatek animacji adekwatnych w swojej treści do danych rekonstruowanego obrazu pierwotnej treśc (na rys.8 powyżej)i, które przedstawiają w formie jasnych pikseli lokalizację współczynników transformaty kosinusowej, wyselekcjonowanych do dalszego przetwarzania.

Jak widać z porównania postaci wyników na rysunkach 8 i 9 treści obrazów rekonstruowanych oraz zakres liczności współczynników wyselekcjonowanych do dalszego przetwarzania, odpowiednio), można stwierdzić potencjalnie wysoką wydajność całego procesu kompresji stratnej, którą można by rozbudować w oparciu o zademonstrowany ciąg wywołań funkcji zaimplementowanych w środowisku Matlab.Z jednej strony, analizowany pierwszy rząd otrzymanych obrazów z rys.8 nie wskazuje na drastyczne obniżenie jakości obrazu oryginalnego. Jednocześnie, z drugiej strony, zakres współczynników prostej transformaty kosinusowej, może być znacznie zawężony w selekcji.Analogicznie, poprzez wizualizację z formie animacji, postanowiono przedstawić pewne potencjalne możliwości wykorzystania w przetwarzaniu treści obrazów 2D prostej i odwrotnej szybkiej transformaty Fouriera:(jakkolwiek bez wydruku wyników końcowych):

```
C:\MATLAB6p1\work\fftmovie.m
26 kwiecień 2006
%rotate white rectangle with FFT transform visualisation
%function [Mfft,Mfttb,ims]=fftmovie(sz)
%[Morg,Mfftt,Mfftb,imss]=fftmovie(100);%default call parameters
function [M,M1,M2,ims]=fftmovie(sz)
M=moviein(4);
M1=moviein(4);
M2=moviein(4);
ims=zeros(sz);
ims((sz/8:7*sz/8),(7*sz/16:9*sz/16))=1;
j=1;
hnd=subplot(1,3,1);
for i=1:4,
    imrot=rot90(ims,i);
 subplot(1,3,1);imshow(imrot);hold on;title('original');hold off;
 F1=fft2(imrot);
 F2=log(abs(F1));
 subplot(1,3,2);imshow(F2);colormap(jet);colorbar;hold on;
 title('FFT image');hold off; M1(:,i)=getframe;
 F3=fftshift(F1);
 subplot(1,3,3);imshow(log(abs(F3)));colormap(jet);colorbar;hold on;
 title('shifted FFT image');hold off;M2(:,i)=getframe;
 M(:,i)=getframe(hnd);
end;
RandName=100+899*rand(1,1);
movie2avi(M ,['FFT2c' int2str(RandName) '.avi'],'compression','Cinepak','FPS',3);
movie2avi(M1,['FFT2a' int2str(RandName) '.avi'],'compression','Cinepak','FPS',3);
movie2avi(M2,['FFT2b' int2str(RandName) '.avi'],'compression','Cinepak','FPS',3);
close;
```

Rys10 Wizualizacja wyników wywołania funkcji szybkiej transformaty Fouriera