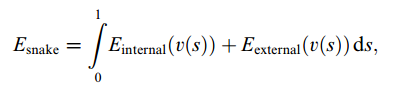
## Snake能量说明：

能量公式：



第一项是图像的内部能量，其产生源于曲线的弯曲和形变，内部能量可以写成下式：

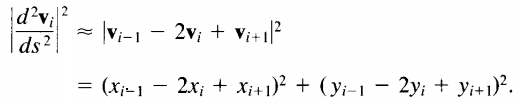


第一项为连续性能量（有的书称平滑能量，弹性能量），如果曲线有一个gap，该值会很大，第二项为弯曲能量，曲线曲率越大，该值会越大。

此两项的近似计算方法如下：



可见，要想该能量值小，相邻两点的距离必须靠近，因此，它被称为弹性力。



对每一个初始点，检查其领域，在程序中是5×5的区域，共25个候选点，对每一个假造点分别计算其所具有的能量，能量由三部分组成：连续性能量（Continuity term），弯曲能量（Curvature term），图像能量（Image energy term）。

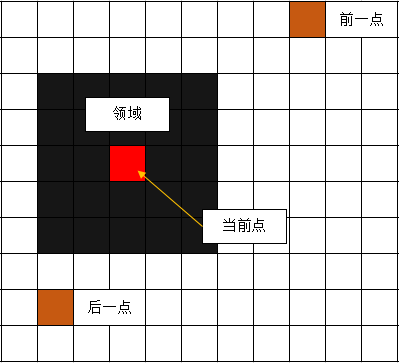
连续性能量的表达为：abs(avgDist - norm(curPos - points(:, iMinusOne)))，avgDist为链上点的平均距离，curPos为当前候选点，points(:, iMinusOne)为前一点，norm函数是求向量的模。表达试的意义为当前候选点与前一点距离，与平均距离做差，求绝对值。

弯曲能量表达式为：norm(points(:,iMinusOne)-2\*curPos+points(:,iPlusOne))^2，即前一点与后一点相加再减去两个当前点。

图像能量为当前候选点对应的归一化梯度值。

总能量为：Es = alpha(realI) .\* eCont(:) + beta(realI) .\* eCurv(:) + gamma(realI) .\* eImage(:);

每一个候选点对应一个ES，选择最小ES的候选点作为当前点要移动到的点。



## Matlab代码：

neighSize = 0;

for j = 1:5,

for k = 1:5,

neighSize = neighSize + 1;

neighbors ((j-1)\*5 + k, :) = [ j-1-5+3 k-1-5+3 ];

end

end

% Set up the path

path('C:\MATLAB6p5p1\work', path);

% Create the main figure

mainFig = figure('Name', 'Snakes',...

'NumberTitle', 'off');

% Prompt the user for the file to input

fname = 0;

while fname == 0,

[ fname, pathname ] = uigetfile('\*.bmp', 'Select the image...');

if fname == 0,

msgbox('Cannot open file.', 'Error', 'error');

end

end

% Read in the image file

im = imread([ pathname fname ], 'bmp');

% Convert to grayscale if necessary

if isrgb(im)

im = rgb2gray(im);

end

im = double(im);

origImage = im;

% Show the image

imshow(im, []);

set(mainFig, 'Resize', 'off');

% Put the image through Gaussian filter to smooth out

% the grooves in the image derivative

%gausImage = upConv(im, [ 1 6 15 20 15 6 1 ] /64);

% Get two separate derivatives

[ gradImage1, gradImage2 ] = gradient(im);

% Combine the two derivatives

gradImage = -abs(gradImage1) - abs(gradImage2);

gradImage = gradImage - min(gradImage(:));

% Median filter the gradient image to get rid of noise

gradImage = medfilt2(gradImage, [ 4 4 ]);

[ imRows imCols ] = size(gradImage);

curvThreshold = 0.3;

edgeThreshold = (max(gradImage(:)) - min(gradImage(:)))\*0.7 + min(gradImage(:));

% Set up the input variables

numPoints = 0;

inputLoop = 1;

uicontrol('String','Finish', 'Callback', '');

% Input loop - lets user enter points

while (inputLoop)

% Wait for mouse input

waitforbuttonpress;

% Get the current point

currPt=get(gca,'CurrentPoint');

currPt=currPt(1,1:2);

% if the point is the "finish" button,

% then break out of the loop

if currPt(2)>size(im, 1)

break;

end

numPoints = numPoints + 1;

points(1, numPoints)=round(currPt(1)); %Get on screen coordinates

points(2, numPoints)=round(currPt(2));

% Put marker on screen

line(points(1, numPoints),points(2, numPoints), ...

'Marker','s', ...

'Color','g', ...

'MarkerSize',7, ...

'EraseMode','none');

line(points(1, numPoints),points(2, numPoints), ...

'Marker','d', ...

'Color','k', ...

'MarkerSize',6, ...

'EraseMode','none');

end

% Initialize alphas, betas and gammas

alpha(1:numPoints) = 1.0;

beta(1:numPoints) = 1.0;

gamma(1:numPoints) = 1.2;

% Calculate the average distance

avgDist = 0;

for i = 1:numPoints,

if i == numPoints,

iPlusOne = 1;

else

iPlusOne = i + 1;

end;

avgDist = avgDist + norm(points(:, i) - points(:, iPlusOne));

end

avgDist = avgDist / (numPoints);

% whileFlag keep the main loop going

whileFlag = 1;

while whileFlag,

% Counts the number of points which have been moved

ptsMoved = 0;

% Loop to move points to new locations

for i = 0:numPoints,

% Mod arithmetic

realI = mod(i, numPoints) + 1;

if realI == 1,

iMinusOne = numPoints;

else

iMinusOne = realI - 1;

end

if realI == numPoints,

iPlusOne = 1;

else

iPlusOne = realI + 1;

end

% Set eMin to infinity

eMin = Inf;

% Get the neighborhood of pixels in the gradient image

daHood = gradImage((points(2, realI)-2):(points(2, realI)+2), ...

(points(1, realI)-2):(points(1, realI)+2));

% Normalize the gradient neighborhood

daMin = min(daHood(:));

daMax = max(daHood(:));

daRange = daMax - daMin;

if daRange < 5,

daRange = 5;

daMin = daMax - 5;

end

daHood = (daMin - daHood)/(daRange);

% Go through the neighborhood to determine next point

for j = 1:neighSize,

% Location being considered

curPos = points(:,realI) + neighbors(j,1:2)';

% If by the edges of the screen,

% then we are in trouble

if (curPos(1)<2 | curPos(2)<2 | ...

(curPos(1)>(imCols-2)) | ...

(curPos(2)>(imRows-2)))

disp('Point close to edge.');

eCont(j) = NaN;

eCurv(j) = NaN;

eImage(j) = NaN;

else

% Calculate Continuity term

eCont(j) =

abs(avgDist - norm(curPos - points(:, iMinusOne)));

% Calculate curvature term

eCurv(j) = norm(points(:, iMinusOne) ...

- 2\*curPos + points(:, iPlusOne))^2;

% Calculate image energy term

eImage(j) = -(daHood(neighbors(j,2)+3, ...

neighbors(j,1)+3));

end

end

% Normalize continuity and energy terms

% in the current neighborhood

if (max(eCont) - min(eCont)) == 0,

eContRange = 1;

else

eContRange = max(eCont) - min(eCont);

end;

if (max(eCurv) - min(eCurv)) == 0,

eCurvRange = 1;

else

eCurvRange = max(eCurv) - min(eCurv);

end;

eCont = (eCont - min(eCont))/eContRange;

eCurv = (eCurv - min(eCurv))/eCurvRange;

% Add up energies in vector form

Es = alpha(realI) .\* eCont(:) + beta(realI) .\* eCurv(:) +...

gamma(realI) .\* eImage(:);

% Determine the smallest energy in the neighborhood

% save the position and edge magnitude

for j = 1:neighSize,

curPos = points(:,realI) + neighbors(j, 1:2)';

if Es(j) < eMin

eMin = Es(j);

minPos = curPos;

edgeMag = gradImage(curPos(2), curPos(1));

end

end

% If a point is moved to a new location,

% increment the counter of moved points

if (minPos(2) ~= points(2, realI) & ...

minPos(1) ~= points(1, realI))

ptsMoved = ptsMoved + 1;

end

% Move the point to the new location

points(:, realI) = minPos;

edgeStrength(realI) = edgeMag;

end

% Figure out which corners to allow

% Find curvatures

for i = 1:numPoints,

if i == numPoints,

iPlusOne = 1;

else

iPlusOne = i+1;

end

if i== 1,

iMinusOne = numPoints;

else

iMinusOne = i - 1;

end

ui = points(:, i) - points(:, iMinusOne);

if ui == [ 0; 0 ], ui = [ 1 ;1 ] ; end;

uiPlusOne = points(:, iPlusOne) - points(:, i);

if uiPlusOne == [ 0; 0 ], uiPlusOne = [ 1 ;1 ]; end;

curv(i) = norm(ui/norm(ui) - uiPlusOne/norm(uiPlusOne))^2;

end

% Loosen Beta(i)

for i = 1:numPoints,

if i == numPoints,

iPlusOne = 1;

else

iPlusOne = i+1;

end

if i== 1,

iMinusOne = numPoints;

else

iMinusOne = i - 1;

end

if (curv(i) > curv(iMinusOne) & curv(i) > curv(iPlusOne) &...

curv(i) > curvThreshold &...

edgeStrength(i) < edgeThreshold)

disp('Beta loosened.');

beta(i) = 0;

else

beta(i) = 1;

end

end

% Show the image and all of the snake points

imshow(origImage, []);

for i = 1:numPoints,

if i == numPoints,

iPlusOne = 1;

else

iPlusOne = i + 1;

end;

line(points(1, i),points(2, i), ...

'Marker','s', ...

'Color','g', ...

'MarkerSize',7, ...

'EraseMode','none');

line(points(1, i),points(2, i), ...

'Marker','d', ...

'Color','k', ...

'MarkerSize',6, ...

'EraseMode','none');

if i == numPoints,

line(points(1, [ i 1 ]), points(2, [ i 1 ]), ...

'Marker', 'none', ...

'Color', 'r', ...

'EraseMode', 'none');

else

line(points(1, i:(i+1)), points(2, i:(i+1)), ...

'Marker', 'none', ...

'Color', 'r', ...

'EraseMode', 'none');

end;

end

% Flush graphics to the screen

drawnow;

avgDist = 0;

for i = 1:numPoints,

if i == numPoints,

iPlusOne = 1;

else

iPlusOne = i + 1;

end;

avgDist = avgDist + norm(points(:, iPlusOne) - points(:,i));

end

avgDist = avgDist / (numPoints);

% If no points are moving, break out

if ptsMoved == 0,

whileFlag = 0

end

end