Программная реализация размытия по Гауссу

Контрольная работа по курсу «Инженерная и компьютерная графика»

Выполнил: ст. гр. 4032 Турбин А.
Проверил: доц. Митрошин А. А.

РГРТУ, 2007

Введение

Размытие изображений играет большую роль в современных областях компьютерной графики. Размытие изображений часто бывает направлено на имитацию близорукости (в тех случаях, когда близорукость становится желательной или даже необходимой). Так, размытие отдельных частей изображения часто используют из соображений цензуры. В ряде случаев размытие является неотъемлемой частью различных техник коррекции изображения, направленных на устранение специфических дефектов (излишняя детализация, дефекты сканирования, царапины, пыль). Известно, что фотомодели и их фотографы используют специальные процедуры размытия фотографических изображений для достижения эффекта «устранения морщин». Размытые изображения также лучше поддаются сжатию (так, при сохранении в формате JPEG графический файл имеет меньший размер, а также менее выраженные артефакты компрессии). Различные техники размытия изображения доступны во всех современных графических редакторах (таких как Adobe Photoshop, The GIMP).

Одним из наиболее важных алгоритмов размытия изображений является т. н. *размытие по Гауссу*.

1. Математический аппарат

Размытие по Гауссу — это характерный фильтр размытия изображения, который использует нормальное распределение (также называемое Гауссовым распределением, отсюда название) для вычисления преобразования, применяемого к каждому пикселю изображения. Уравнение распределения Гаусса в N измерениях имеет вид:

$$G(r) = \frac{1}{(2\pi\sigma^2)^{N/2}} e^{-r^2/(2\sigma^2)},$$

или, в частном случае, для двух измерений:

$$G(u, v) = \frac{1}{2\pi\sigma^2} e^{-(u^2+v^2)/(2\sigma^2)},$$

где r — это радиус размытия, $r^2 = u^2 + v^2$, σ — стандартное отклонение распределения Гаусса. В случае двух измерений эта формула задает поверхность, имеющей вид концентрических окружностей с распределением Гаусса от центральной точки. Пиксели, где распределение отлично от нуля используются для построения матрицы свертки, которая применяется к исходному изображению. Значение каждого пикселя становится средне взвешенным для окрестности. Исходное значение пикселя принимает наибольший вес (имеет наивысшее Гауссово значение), и соседние пиксели принимают меньшие веса, в зависимости от расстояния до них.

В теории, распределение в каждой точке изображения будет ненулевым, что потребовало бы вычисление весовых коэффициентов для каждого пикселя изображения. Но, на практике, когда рассчитывается дискретное приближение функции Гаусса, не учитывают пиксели на расстоянии свыше 3σ, т.к. они достаточно малы. Таким образом, программе, фильтрующей изображение, достаточно рассчитать матрицу $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$, чтобы гарантировать достаточную точность приближения распределения Гаусса.

К тому же имеется круговая симметрия, и размытие Гаусса может быть применено к двухмерному изображению как два независимых одномерных преобразования —

это свойство называется линейной сепарабельностью (linearly separable). То есть эффект от применения двухмерной матрицы аналогичен применению двух одномерных матриц: сначала одну по всем горизонтальным направлениям, а потом другую — по вертикальным. Это довольно полезное свойство, так как благодаря ему вычисление может быть произведено всего за $O(n \times M \times N) + O(m \times M \times N)$, а не за $O(m \times n \times M \times N)$ в случае несепарабельной матрицы свертки (ядра), где M, N — это размеры фильтруемого изображения и m, n — размер ядра фильтра.

2. Программная реализация

Программная реализация разработана в среде Free Pascal.

2.1. Интерфейс программы

Программа `glub.exe` работает из командной строки и использует механизм передачи аргументов командной строки для ввода и вывода информации. А именно, программа использует первые три позиционные параметра командной строки:

1. Параметр σ, который соответствует «радиусу» размытия.
2. Имя входного файла (путь к файлу), к которому будет применён алгоритм размытия по Гауссу.
3. Имя выходного файла (путь к файлу), в котором будет сохранено размытое изображение.

Таким образом, программа не имеет графического интерфейса, а её поведение полностью определяется позиционными параметрами командной строки. Запуск программы может выглядеть, например, так:

```
OS> glubr 10 vika.jpg vika2.jpg
```

В этом примере происходит размытие изображения `vika.jpg` с параметром $\sigma = 10$. Размытое изображение сохраняется в файле `vika2.jpg`.

Для чтения и записи изображений используются стандартные библиотеки среды Free Pascal. При открытии файла происходит диспетчеризация по расширению файла (так, в последнем примере у изображения `vika.jpg` расрширение `jpg`, поэтому будет задействован декодер JPEG-файлов). Программа поддерживает форматы JPEG, BMP и PNG, как наиболее распространённые. После чтения входного файла в программе доступна матрица $M \times N$ RGB-пикселей. Для того, чтобы улучшить переносимость кода в другие Pascal-системы, добавлены вспомогательные процедуры чтения и записи пикселей матрицы (`GetRGB` и `SetRGB`).

При записи выходного файла также происходит выбор формата на основе расширения (в примере выходной файл `vika2.jpg` будет записан в формает JPEG). Также при записи выходного изображения устанавливаются некоторые параметры, специфичные для выходного формата. Поскольку размытие создает большое количество полутонов, при сохранении создаются полноцветные изображения (32-битная представление цвета в BMP, отключение индексной палитры в PNG). Для JPEG файлов устанавливается качество сжатия 90 (высокое качество).

2.2. Реализация алгоритма

В реализации используется свойство линейной сепарабельности, которое описано в предыдущем разделе. Вместо двумерной матрицы преобразования используется симметричный «массив» преобразования (который мы далее называем «окном»), в котором центральный элемент имеет наибольшее значение. Массив последовательно используется сначала при «горизонтальном», а потом при «вертикальном» проходах размытия (при горизонтальном проходе массив используется как «строка», а при вертикальном — как «столбец»). Для каждого пикселя изображения, соответствующего «центральному» элементу окна, новое значение пикселя (точнее, его RGB-компонентов) вычисляется как линейная комбинация самого пикселя и его ближайших «соседей», в соответствии с весовыми коэффициентами окна (данная техника известна под названием «плавающее окно», *sliding window*).

Таким образом, с точки зрения реализации, размытие по Гауссу состоит в том, что значение каждого пикселя изображения «усредняется» с соседними пикселями в соответствии с весовыми коэффициентами «плавающего окна», при этом на первом проходе усреднение происходит по «горизонтальным» соседям, а на втором — по «вертикальным».

«Односторонний» размер окна вычисляется по «правилу трёх сигм»:

```
N:=ceil(3*sigma);
```

(здесь `sigma` — это параметр σ, который передаётся в программу через первый позиционный аргумент командной строки) и далее создается «массив окна»

```
window: array[-N..N] of real;
```

В данной реализации массив `window` инициализируется «ненормализованными» значениями, то есть по формуле

$$\texttt{window}(i) = \begin{cases} e^0 = 1, & \text{если } i = 0; \\ e^{-i^2/(2\sigma^2)}, & \text{если } i \in 1..N; \\ \texttt{window}(-i), & \text{для отрицательных индексов.} \end{cases}$$

Перенормировка происходит всякий раз после *фактического* использования весов. Это необходимо для корректного размытия «крайних» пикселей. При обработке каждого пикселя сумма всех использованных весов должна быть равна единице, что обеспечивает сохранение (в среднем) яркости изображения при усреднении пикселей. Но при обработке «крайних» пикселей используется только часть окна (один из «хвостов» распределения «свисает» за пределами изображения). Без дополнительной перенормировки крайние пиксели после размытия оказались бы более «тусклыми».

В связи с реализацией алгоритма следует сделать ещё одно важное замечание: «усреднение» каждого пикселя должно осуществляться так, как если бы его соседи ещё не были усреднены (то есть линейная комбинация должна вычисляться для исходных пикселей строки при горизонтальном проходе и для исходных пикселей столбца при горизонтальном проходе). Однако свойство линейной сепарабельности гарантирует независимость проходов. Это означает, что при обработке каждого столбца или строки нужно сохранять «вновь размытые» пиксели во временном массиве, а потом уже «подменять» эти пиксели в изображении целиком (то есть всю строку или весь столбец за раз).

3. Текст программы

```
1  program gblur;
2  uses
3      math,
4      FPReadPNG, FPWritePNG,
5      FPReadBMP, FPWriteBMP,
6      FPReadJPEG, FPWriteJPEG,
7      FPImage,      { TFPMemoryImage }
8      SysUtils;     { ExtractFileExt, LowerCase }
9  type
10     { пиксель rgb; используется тип real
11       для упрощения промежуточных вычислений }
12     TRGB = record r,g,b: real; end;
13 var
14     image: TFPMemoryImage;
15     reader: TFPCustomImageReader;
16     writer: TFPCustomimageWriter;
17     inputFile, outputFile: string;
18     fileExt: string;
19     sigma: real;
20
21 { вспомогательные процедуры для чтения и записи пискелей }
22 procedure GetRGB(x,y: integer; var rgb: TRGB);
23 var
24     c: TFPColor;
25 begin
26     c:=image.colors[x,y];
27     rgb.r:=c.red;
28     rgb.g:=c.green;
29     rgb.b:=c.blue;
30 end;
31 procedure SetRGB(x,y: integer; rgb: TRGB);
32 var
33     c: TFPColor;
34 begin
35     c.red:=round(rgb.r);
36     c.green:=round(rgb.g);
37     c.blue:=round(rgb.b);
38     { непрзрачный альфа-канал }
39     c.alpha:=alphaOpaque;
40     image.colors[x,y]:=c;
41 end;
42
43 { размытие по Гауссу на матрице image }
44 procedure do_gblur;
45 const
46     { максимальный размер окна }
47     maxWin = 100;
48     { максимальный линейный размер рисунка }
49     maxDim = 4096;
50 var
```

```
51      { окно }
52      window: array[-maxWin..maxWin] of real;
53      N: 1..maxWin;
54      { переменные циклов и вспомогательные переменные }
55      i, j, k, l: integer;
56      { сумма элементов окна, для нормализации }
57      sum: real;
58      { 2*sigma*sigma }
59      s2: real;
60      { временный массив для формирования строки/столбца }
61      tmp: array[0..maxDim] of TRGB;
62      { текущий пиксель }
63      pix: TRGB;
64      { ещё один один пиксель для итерации }
65      p: TRGB;
66  begin
67      s2:=2*sigma*sigma;
68      { размер окна по "правилу трёх сигм" }
69      N:=ceil(3*sigma);
70      { центральный элемент окна пока будет exp(0) = 1 }
71      window[0]:=1;
72      { инициализация окна }
73      for i:=1 to N do begin
74          window[i]:=exp(-i*i/s2);
75          window[-i]:=window[i];
76      end;
77      { отладочная печать окна }
78      for i:=-N to N do
79          write(window[i]:9:6);
80      writeln;
81      { первый проход - горизонтальное размытие }
82      for j:=0 to image.height-1 do begin
83          for i:=0 to image.width-1 do begin
84              { будем вычислять сумму использованных коэффициентов
85                для нормализации; это нужно делать каждый раз
86                потому что для размытия крайних пикселей
87                используется только часть окна }
88              sum:=0;
89              { это [i,j] пиксель который будем формировать }
90              pix.r:=0; pix.g:=0; pix.b:=0;
91              { проходимся окном вокруг этого пикселя }
92              for k:=-N to N do begin
93                  { l - индекс другого пикселя,
94                    который вносит вклад }
95                  l:=i+k;
96                  if (l >= 0) and (l < image.width) then begin
97                      GetRGB(l, j, p);
98                      pix.r:=pix.r+p.r*window[k];
99                      pix.g:=pix.g+p.g*window[k];
100                     pix.b:=pix.b+p.b*window[k];
101                     sum:=sum+window[k];
102                 end;
```

```
103                 end;
104                 { нормализация - сумма использованных коэффициентов
105                   окна должна быть равна 1 }
106                 pix.r:=pix.r/sum;
107                 pix.g:=pix.g/sum;
108                 pix.b:=pix.b/sum;
109                 { пиксель готов - сохраняем во временный массив }
110                 tmp[i]:=pix;
111             end;
112             { строка готова - сохраняем временный массив
113               в само изображение }
114             for i:=0 to image.width-1 do
115                 SetRGB(i, j, tmp[i]);
116         end;
117         { второй проход - вертикальное размытие }
118         for i:=0 to image.width-1 do begin
119             for j:=0 to image.height-1 do begin
120                 sum:=0;
121                 pix.r:=0; pix.g:=0; pix.b:=0;
122                 for k:=-N to N do begin
123                     { отличие в том, что итерация теперь будет
124                       не по строке, а по столбцу }
125                     l:=j+k;
126                     if (l >= 0) and (l < image.height) then begin
127                         GetRGB(i, l, p);
128                         pix.r:=pix.r+p.r*window[k];
129                         pix.g:=pix.g+p.g*window[k];
130                         pix.b:=pix.b+p.b*window[k];
131                         sum:=sum+window[k];
132                     end;
133                 end;
134                 pix.r:=pix.r/sum;
135                 pix.g:=pix.g/sum;
136                 pix.b:=pix.b/sum;
137                 tmp[j]:=pix;
138             end;
139             for j:=0 to image.height-1 do
140                 SetRGB(i, j, tmp[j]);
141         end;
142     end;
143
144     begin { основная программа }
145         { проверка числа аргументов командной строки }
146         if paramcount <> 3 then begin
147             writeln('Usage: gblur sigma inputFile outputFile');
148             Halt(1);
149         end;
150         { чтение параметра sigma }
151         sigma:=0;
152         Val(paramstr(1), sigma);
153         if sigma <= 0 then begin
154             writeln('Bad sigma value ', paramstr(1));
```

```
155             Halt(1);
156         end;
157         { загрузка изображения }
158         inputFile:=paramstr(2);
159         fileExt:=LowerCase(ExtractFileExt(inputFile));
160         delete(fileExt,1,1);
161         if (fileExt = 'jpg') or (fileExt = 'jpeg') then
162             reader:=TFPReaderJPEG.Create
163         else if fileExt = 'bmp' then
164             reader:=TFPReaderBMP.Create
165         else if fileExt = 'png' then
166             reader:=TFPReaderPNG.Create
167         else begin
168             writeln('Unknown input file format: ', inputFile);
169             Halt(1);
170         end;
171         image:=TFPMemoryImage.Create(0,0);
172         image.UsePalette:=false;
173         image.LoadFromFile(inputFile, reader);
174         { применение размытия по Гауссу }
175         do_gblur;
176         { сохранение изображения }
177         outputFile:=paramstr(3);
178         fileExt:=LowerCase(ExtractFileExt(outputFile));
179         delete(fileExt,1,1);
180         if (fileExt = 'jpg') or (fileExt = 'jpeg') then
181             begin
182                 writer:=TFPWriterJPEG.Create;
183                 TFPWriterJPEG(writer).CompressionQuality:=90;
184             end
185         else if fileExt = 'bmp' then
186             begin
187                 writer:=TFPWriterBMP.Create;
188                 TFPWriterBMP(writer).BitsPerPixel:=32;
189             end
190         else if fileExt = 'png' then
191             begin
192                 writer:=TFPWriterPNG.Create;
193                 TFPWriterPNG(writer).Indexed:=false;
194             end
195         else
196             begin
197                 writeln('Unknown output file format: ', outputFile);
198                 Halt(1);
199             end;
200         image.SaveToFile(outputFile, writer);
201         reader.Free;
202         writer.Free;
203         image.Free;
204 end.
205 { ex:set ts=8 sts=4 sw=4 et: }
```