УДК 004.42

Использование размытия по Гауссу в изображениях

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Целью исследования является использование алгоритма размытия по Гауссу. В работе представлено написание программы на языке OpenGL Shading Language, используя алгоритм размытия по Гауссу. В результате программа выводит размытые изображения.
Ключевые слова: OpenGL Shading Language, GLSL, размытие по Гауссу.

Using Gaussian blur in images

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
Student

Abstract
The purpose of the study is to use the Gaussian blur algorithm. The paper presents the writing of a program in the OpenGL Shading Language, using the Gaussian blur algorithm. As a result, the program outputs blurry images.
Keywords: OpenGL Shading Language, GLSL, Gaussian blur.

1 Введение

1.1 Актуальность исследования

Данная статья описывает возможность написание программы на OpenGL Shading Language для размытия изображения.

1.2 Цель исследования

Целью исследования является написание программы по использованию алгоритма размытия по Гауссу.

1.3 Обзор исследований

Ж. Ж. Хванг, К. Х. Рхее рассматривают криминалистического обнаружение Гаусса в измененных изображениях[1]. З. Ду, Х. Ли, Ю. Гуа рассматривают метод восстановление ядра размытия Гаусса для изображения [2]. А. Дас и др. исследуют использование на основе размытия Гауссу как обнаружение следов фотомонтажа и подделки изображения [3]. Р. Гайджар, Т. Завери, А. Шукла показывают классификацию размытия изображения [4]. С. Мучукумар исследует восстановление изображение, которое было размыто и деградировало Гауссовским и импульсивным шумом [5]. В. Г. Хи показывает способ оценки Гауссовского шума равномерным линейным наездом камеры размытого изображения [6].

2. Результаты и обсуждение

2.1 Алгоритм

Размытие по Гауссу - это способ размытия изображения с помощью функции Гаусса [7]. В основу формулы взято нормальное распределение (рис 1)[8].

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$

Рисунок 1. Нормальное распределение или распределение Гаусса, где μ — математическое ожидание (среднее значение), σ — среднеквадратическое отклонение [8]

Однако, при μ = 0 получаем формулу (рис 2).

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Рисунок 2. Функция Гаусса в одном измерениях

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Рисунок 3. Функция Гаусса в двух измерениях, где x, y — координаты точки

Использование размытия по Гауссу к изображению необходимо, чтобы сумма весов функции Гаусса должна быть равно единице, так как требуется избежать изменения яркости изображения, поэтому используется среднее взвешенное (Рис. 4) [9].

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i \cdot x_i}{\sum_{i=1}^{n} w_i}$$

Рисунок 4. Среднее арифметическое взвешенное

Для выполнения размытия по Гауссу для каждого пикселя вычисляется сумма значений всех пикселей в изображении, взвешенную значениями Гауссовой функции для данного пикселя. В результате получится новое значение пикселя [10].

В данной программе используемая функция blur_Gaussian, сначала суммируется функция Гаусса получается переменная k для определения сумма весов. Переменная col (новый пиксель) полученная из функция Гаусса деленный на k и умноженный на исходный пиксель, координаты исходный

пиксель вычисляется (uv * size + vec2(i, i) * dir) / size. uv — координаты экрана обычно значения от 0 до 1 полученный текущий координат пикселя экрана деленный на размер экрана. size — размер изображения. I — текущий инкремент значение от -n до n (Листинг 4).

Листинг 1. Функция Гаусса использовано формула (рис 2, 3).

```
1  float kernel_Gaussian(float s, float x) {
2          return 1.0 / sqrt(2.0 * PI * s * s) * exp(-(x*x)/(2.0*s*s));
3  }
4  float kernel_Gaussian2(float s, vec2 a) {
5          return 1.0 / (2.0 * PI * s * s) * exp(-(a.x*a.x+a.y*a.y)/(2.0*s*s));
6  }
```

Для запуска программы шейдер используется glslViewer [11].

```
glslViewer --fps 24 ./filter_gauisee.frag ./texture/lena.jpeg
```

2.2 Использование размытия по Гауссу

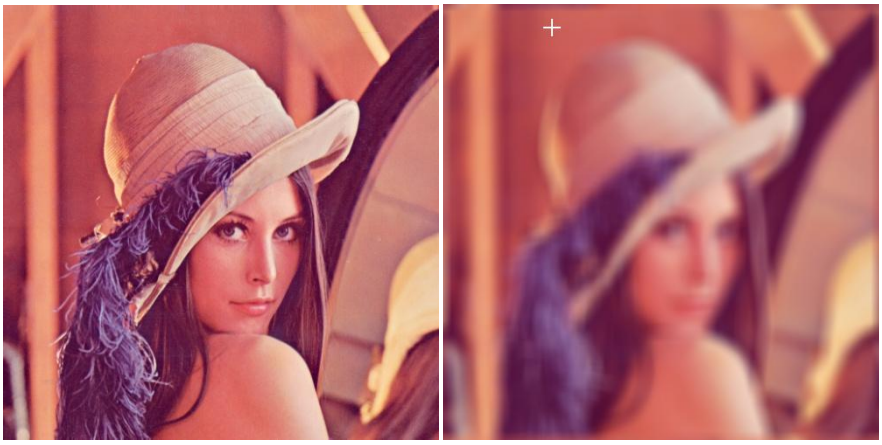

Рисунок 5. Влево исходное изображения, справа размытое изображения, размер фильтра 20

В Рисунок 5 Влево исходное изображения, а справа применено функция (рис 3.) (листинг 2 строка 27) для применение фильтра имеет вычислительную сложность $O(4n^2)$. Для создания функция Гаусса (рис 3) имеет сложность $O(4n^2)$. Для примера, размер фильтра Гаусса 20x20 число обращений к текстуре 400. сумма общее сложности 800.

Листинг 2. Пример использования функция Гаусса.

```
1  vec3 blur_Gaussian2(vec2 uv, sampler2D img, vec2 size, int n) {
2          if (n < 2) {
3                  return texture(img, uv).rgb;
4          }
5          int i, j;
6          float k = 0.0, d = 0.0;
```

```
7           vec3 col;
8           if (size.x <= 0.0 || size.y <= 0.0) {
9                   size = textureSize(img, 0);
10          }
11          for (i = -n; i < n; i++) {
12                  for (j = -n; j < n; j++) {
13                          k += kernel_Gaussian2(4.0*n*n, vec2(i, j));
14                  }
15          }
16          for (i = -n; i < n; i++) {
17                  for (j = -n; j < n; j++) {
18                          d = kernel_Gaussian2(4.0*n*n, vec2(i, j)) / k;
19                          col += d * texture(img, (uv * size + vec2(i, j)) / size).rgb;
20                  }
21          }
22          return col;
23  }
24  void main (void) {
25          vec2 uv = gl_FragCoord.xy/u_resolution.xy * 2.0 - 1.0;
26          vec3 col = vec3(0.0);
27          col = blur_Gaussian2(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), 10);
28          gl_FragColor = vec4(clamp(col, 0.0, 1.0), 1.0);
29  }
```

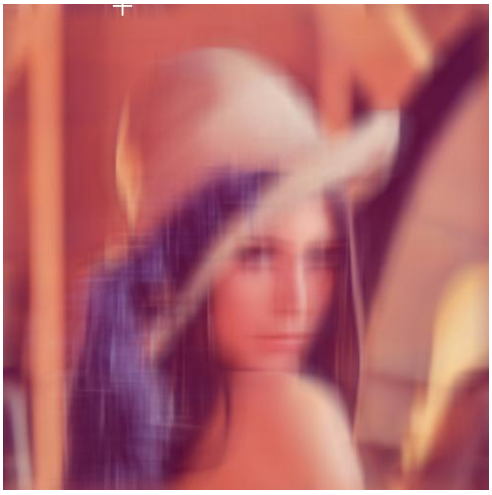

Рисунок 6. Размытое изображение, размер фильтра 20

В рисунке 6 применена функция blur_Gaussian2v2 (листинг 3) для применения фильтра, которая имеет вычислительную сложность O(2(1+2(n-1))). Для создания функция Гаусса (рис 3) имеет сложность O(2(1+2(n-1))) (листинг 4). Для примера, размер фильтра Гаусса 20x20 число обращений к текстуре 78 общее сумма сложности 156.

Листинг 3. Создания функция Гаусса и применение эффект размытия.

```
1   vec3 blur_Gaussian2v2(vec2 uv, sampler2D img, vec2 size, int n, vec4 dir) {
2           if (n < 1) {
3                   return texture(img, uv).rgb;
```

```
	}
	int i, j;
	float k = 0.0, k2 = 0.0, d = 0.0;
	vec3 col;
	vec2 c;
	if (size.x <= 0.0 || size.y <= 0.0) {
		size = textureSize(img, 0);
	}
	k = kernel_Gaussian2(n, vec2(0, 0));
	for (i = 1; i < n; i++) {
		k += kernel_Gaussian2(n, vec2(i, i) * dir.xy);
		k += kernel_Gaussian2(n, -vec2(i, i) * dir.xy);
	}
	k2 = kernel_Gaussian2(n, vec2(0, 0));
	for (j = 1; j < n; j++) {
		k2 += kernel_Gaussian2(n, vec2(j, j) * dir.zw);
		k2 += kernel_Gaussian2(n, -vec2(j, j) * dir.zw);
	}
	d = kernel_Gaussian2(n, vec2(0, 0)) / k;
	col = d * texture(img, (uv * size) / size).rgb;
	for (i = 1; i < n; i++) {
		c = uv * size;
		d = kernel_Gaussian2(n, vec2(i, i) * dir.xy) / k;
		col += d * texture(img, (c + vec2(i, i) * dir.xy) / size).rgb;
		d = kernel_Gaussian2(n, -vec2(i, i) * dir.xy) / k;
		col += d * texture(img, (c - vec2(i, i) * dir.xy) / size).rgb;
	}
	d = kernel_Gaussian2(n, vec2(0, 0)) / k2;
	col += d * texture(img, (uv * size) / size).rgb;
	for (j = 1; j < n; j++) {
	c = uv * size;
		d = kernel_Gaussian2(n, vec2(j, j) * dir.zw) / k2;
		col += d * texture(img, (c + vec2(j, j) * dir.zw) / size).rgb;
		d = kernel_Gaussian2(n, -vec2(j, j) * dir.zw) / k2;
		col += d * texture(img, (c - vec2(j, j) * dir.zw) / size).rgb;
	}
	return col / 2.;
}
```

Листинг 4. Применение размытия Гаусса.

```
col = blur_Gaussian2v2(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), 20);
```

Функция blur_Gaussian2v2 применяет фильтр Гаусса, вертикальное и горизонтальное размытия в заданной направление dir (Листинг 2).

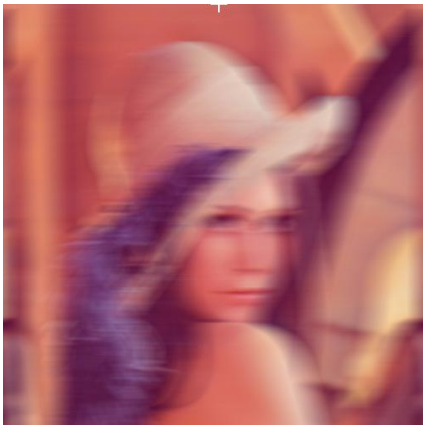

Рисунок 7. Горизонтальное размытие изображения размер фильтра 20

В рисунке 7 применена функция blur_Gaussian (листинг 5) для применения фильтра, которая имеет вычислительную сложность O(2n). Для создания функция Гаусса (рис 3) имеет сложность O(2n) (листинг 6). Для примера, размер фильтра Гаусса 20 число обращений к текстуре 40 общее сумма сложности 80.

Листинг 5.

```
vec3 blur_Gaussian(vec2 uv, sampler2D img, vec2 size, int n, vec2 dir) {
	if (n < 1) {
		return texture(img, uv).rgb;
	}
	int i, j;
	float k = 0.0, d = 0.0;
	vec3 col;
	if (size.x <= 0.0 || size.y <= 0.0) {
		size = textureSize(img, 0);
	}
	for (i = -n; i < n; i++) {
		k += kernel_Gaussian(2.0*n, i);
	}
	for (i = -n; i < n; i++) {
		d = kernel_Gaussian(2.0*n, i) / k;
		col += d * texture(img, (uv * size + vec2(i, i) * dir) / size).rgb;
	}
	return col;
}
```

Листинг 6.

```
col = blur_Gaussian(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), 20, vec2(1.0, 0.0));
```

Функция blur_Gaussian применяет фильтр Гаусса, размытия в заданной направление dir (Листинг 4).

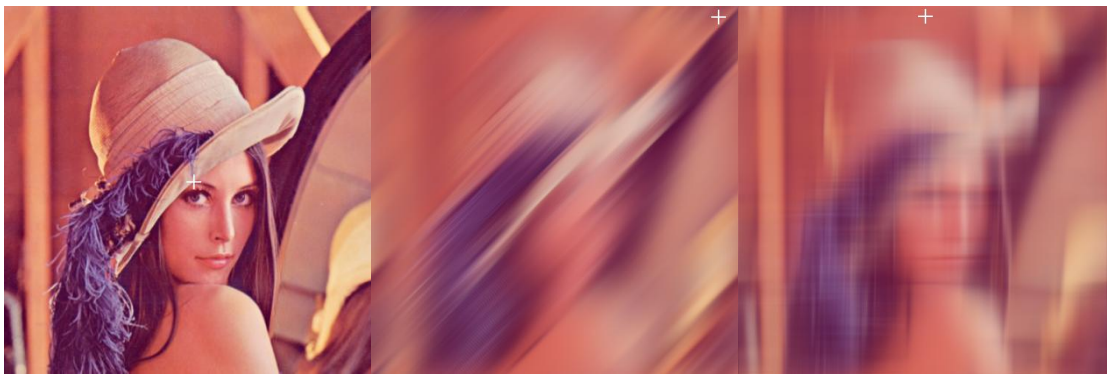

Рисунок 8. Пример использования blur_Gaussian2v2.

Листинг 7. Пример использования эффекта размытия в движения.

```
1  void main (void) {
2          vec2 mv = u_mouse.xy/u_resolution.xy * 2.0 - 1.0;
3          vec2 uv = gl_FragCoord.xy/u_resolution.xy * 2.0 — 1.0;
4          vec3 col = vec3(0.0);
5          col = blur_Gaussian2v2(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), int(length(mv)
   * 30), vec4(mv * 3.0, mv.yx * 3.0));
6          gl_FragColor = vec4(clamp(col, 0.0, 1.0), 1.0);
7  }
```

Рисунок 9. Пример использования blur_Gaussian

Листинг 8. Пример использования эффекта размытия в движения.

```
1  void main (void) {
2          vec2 mv = u_mouse.xy/u_resolution.xy * 2.0 - 1.0;
3          vec2 uv = gl_FragCoord.xy/u_resolution.xy * 2.0 - 1.0;
4          vec3 col = vec3(0.0);
5          col = blur_Gaussian(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), int(length(mv) *
   20), vec2(mv)*3.0);
6          gl_FragColor = vec4(clamp(col, 0.0, 1.0), 1.0);
7  }
```

Листинг 9. Исходный код

```
1  #version 330
2
3  #ifdef GL_ES
4  precision mediump float;
5  #endif
6  #define PI 3.14159265359
7
```

```
uniform vec2 u_mouse;
uniform vec2 u_resolution;
uniform sampler2D u_tex0;

float kernel_Gaussian(float s, float x) {
	return 1.0 / sqrt(2.0 * PI * s * s) * exp(-(x*x)/(2.0*s*s));
}
float kernel_Gaussian2(float s, vec2 a) {
	return 1.0 / (2.0 * PI * s * s) * exp(-(a.x*a.x+a.y*a.y)/(2.0*s*s));
}
vec3 blur_Gaussian(vec2 uv, sampler2D img, vec2 size, int n, vec2 dir) {
	if (n < 1) {
		return texture(img, uv).rgb;
	}
	int i, j;
	float k = 0.0, d = 0.0;
	vec3 col;
	if (size.x <= 0.0 || size.y <= 0.0) {
		size = textureSize(img, 0);
	}
	for (i = -n; i < n; i++) {
		k += kernel_Gaussian(2.0*n, i);
	}
	for (i = -n; i < n; i++) {
		d = kernel_Gaussian(2.0*n, i) / k;
		col += d * texture(img, (uv * size + vec2(i, i) * dir) / size).rgb;
	}
	return col;
}
vec3 blur_Gaussian2v2(vec2 uv, sampler2D img, vec2 size, int n, vec4 dir) {
	if (n < 1) {
		return texture(img, uv).rgb;
	}
	int i, j;
	float k = 0.0, k2 = 0.0, d = 0.0;
	vec3 col;
	vec2 c;
	if (size.x <= 0.0 || size.y <= 0.0) {
		size = textureSize(img, 0);
	}
	k = kernel_Gaussian2(n, vec2(0, 0));
	for (i = 1; i < n; i++) {
		k += kernel_Gaussian2(n, vec2(i, i) * dir.xy);
		k += kernel_Gaussian2(n, -vec2(i, i) * dir.xy);
	}
	k2 = kernel_Gaussian2(n, vec2(0, 0));
	for (j = 1; j < n; j++) {
		k2 += kernel_Gaussian2(n, vec2(j, j) * dir.zw);
		k2 += kernel_Gaussian2(n, -vec2(j, j) * dir.zw);
	}
	d = kernel_Gaussian2(n, vec2(0, 0)) / k;
	col = d * texture(img, (uv * size) / size).rgb;
	for (i = 1; i < n; i++) {
		c = uv * size;
		d = kernel_Gaussian2(n, vec2(i, i) * dir.xy) / k;
		col += d * texture(img, (c + vec2(i, i) * dir.xy) / size).rgb;
		d = kernel_Gaussian2(n, -vec2(i, i) * dir.xy) / k;
		col += d * texture(img, (c - vec2(i, i) * dir.xy) / size).rgb;
	}
	d = kernel_Gaussian2(n, vec2(0, 0)) / k2;
	col += d * texture(img, (uv * size) / size).rgb;
	for (j = 1; j < n; j++) {
		c = uv * size;
		d = kernel_Gaussian2(n, vec2(j, j) * dir.zw) / k2;
		col += d * texture(img, (c + vec2(j, j) * dir.zw) / size).rgb;
		d = kernel_Gaussian2(n, -vec2(j, j) * dir.zw) / k2;
		col += d * texture(img, (c - vec2(j, j) * dir.zw) / size).rgb;
	}
	return col / 2.;
}
vec3 blur_Gaussian2v2(vec2 uv, sampler2D img, int n) {
	return blur_Gaussian2v2(uv, img, vec2(0.0), n, vec4(1.0, 0.0, 0.0, 1.0));
}
vec3 blur_Gaussian2v2(vec2 uv, sampler2D img, vec2 size, int n) {
	return blur_Gaussian2v2(uv, img, size, n, vec4(1.0, 0.0, 0.0, 1.0));
}
vec3 blur_Gaussian2v(vec2 uv, sampler2D img, vec2 size, int n) {
	if (n < 1) {
```

```
86                      return texture(img, uv).rgb;
87              }
88              int i, j;
89              float k = 0.0, d = 0.0;
90              vec3 col;
91              if (size.x <= 0.0 || size.y <= 0.0) {
92                      size = textureSize(img, 0);
93              }
94              k = kernel_Gaussian2(n*n+1.0, vec2(0, 0));
95              for (i = 1; i <= n; i++) {
96                      for (j = 1; j <= n; j++) {
97                              k += kernel_Gaussian2(n*n+1.0, vec2(i, j));
98                              k += kernel_Gaussian2(n*n+1.0, -vec2(i, j));
99                      }
100             }
101             d = kernel_Gaussian2(n*n+1.0, vec2(0, 0)) / k;
102             col = d * texture(img, (uv * size) / size).rgb;
103             for (i = 1; i <= n; i++) {
104                     for (j = 1; j <= n; j++) {
105                             d = kernel_Gaussian2(n*n+1.0, vec2(i, j)) / k;
106                             col += d * texture(img, (uv * size + vec2(i, j)) / size).rgb;
107                             d = kernel_Gaussian2(n*n+1.0, -vec2(i, j)) / k;
108                             col += d * texture(img, (uv * size - vec2(i, j)) / size).rgb;
109                     }
110             }
111             return col;
112     }
113     vec3 blur_Gaussian2v(vec2 uv, sampler2D img, int n) {
114             return blur_Gaussian2v(uv, img, vec2(0.0), n);
115     }
116     vec3 blur_Gaussian2(vec2 uv, sampler2D img, vec2 size, int n) {
117             if (n < 2) {
118                     return texture(img, uv).rgb;
119             }
120             int i, j;
121             float k = 0.0, d = 0.0;
122             vec3 col;
123             if (size.x <= 0.0 || size.y <= 0.0) {
124                     size = textureSize(img, 0);
125             }
126             for (i = -n; i < n; i++) {
127                     for (j = -n; j < n; j++) {
128                             k += kernel_Gaussian2(4.0*n*n, vec2(i, j));
129                     }
130             }
131             for (i = -n; i < n; i++) {
132                     for (j = -n; j < n; j++) {
133                             d = kernel_Gaussian2(4.0*n*n, vec2(i, j)) / k;
134                             col += d * texture(img, (uv * size + vec2(i, j)) / size).rgb;
135                     }
136             }
137             return col;
138     }
139     vec3 blur_Gaussian2(vec2 uv, sampler2D img, int n) {
140             return blur_Gaussian2(uv, img, vec2(0.0), n);
141     }
142     void main (void) {
143             vec2 mv = u_mouse.xy/u_resolution.xy * 2.0 - 1.0;
144             vec2 uv = gl_FragCoord.xy/u_resolution.xy * 2.0 - 1.0;
145             vec3 col, colb = vec3(0.0);
146             //col = blur_Gaussian2v2(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0) * mv[1], int(mv[0] * 100));
147             //col = blur_Gaussian2(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), int(50 * mv[0]));
148             //col = blur_Gaussian2v(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), int(50 * mv[0]));
149             //col = blur_Gaussian2v2(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), int(50 * mv[0]));
150             //col = blur_Gaussian2v2(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), int(length(mv) * 30), vec4(mv * 3.0, mv.yx * 3.0));
151             //col = blur_Gaussian(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), 20, vec2(1.0, 0.0));
152             col = blur_Gaussian(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), int(length(mv) * 20), vec2(mv)*3.0);
153             //colb = blur_Gaussian(uv / 2.0+0.5, u_tex0, textureSize(u_tex0, 0), int(length(mv) * 30), vec2(mv.yx)*5.0);
154             //col = min(col, colb);
155             gl_FragColor = vec4(clamp(col, 0.0, 1.0), 1.0);
156     }
```

3 Выводы

В статье рассмотрена формула размытия Гаусса и использован алгоритм для написания программ на языке GLSL по размытию Гаусса в изображениях.

Библиографический список

1. Hwang J. J., Rhee K. H. Gaussian Forensic Detection using Blur Quantity of Forgery Image //2019 International Conference on Green and Human Information Technology (ICGHIT). IEEE, 2019. С. 86-88.
2. Du Z., Li X., Guo Y. Exposing Blur Kernel from Retouch Image //2013 International Conference on Computer-Aided Design and Computer Graphics. IEEE, 2013. С. 407-408.
3. Das A. et al. Image splicing detection using Gaussian or defocus blur //2016 International Conference on Communication and Signal Processing (ICCSP). IEEE, 2016. С. 1237-1241.
4. Gajjar R., Zaveri T., Shukla A. Invariants based blur classification algorithm //2015 5th Nirma University International Conference on Engineering (NUiCONE). IEEE, 2015. С. 1-5.
5. Muthukumar S. et al. An efficient color image denoising method for Gaussian and impulsive noises with blur removal //2010 IEEE International Conference on Computational Intelligence and Computing Research. IEEE, 2010. С. 1-4.
6. He W. G. Estimation of the noise variance of uniform linear motion blurred images //2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010). IEEE, 2010. Т. 3. С. 64-67.
7. Размытие по Гауссу. // Википедия URL: https://ru.wikipedia.org/wiki/Размытие_по_Гауссу (дата обращения: 2021-08-26).
8. Нормальное распределение. // Википедия URL: https://ru.wikipedia.org/wiki/Нормальное_распределение (дата обращения: 2021-08-26).
9. Среднее арифметическое взвешенное. // Википедия URL: https://ru.wikipedia.org/wiki/Среднее_арифметическое_взвешенное (дата обращения: 2021-08-27).
10. Вольф Д, А. Н. Киселева Open GL 4. Язык шейдеров. Книга рецептов. Москва: ДМК Пресс, 2015. 174 с.
11. patriciogonzalezvivo/glslViewer: Console-based GLSL Sandbox for 2D/3D shaders shaders. // GitHub URL: https://github.com/patriciogonzalezvivo/glslViewer (дата обращения: 2021-08-27)