DOI 10.37882/2223–2966.2023.04.33

МОДИФИКАЦИЯ АЛГОРИТМА ОБРАБОТКИ ЭФФЕКТА СВЕЧЕНИЯ ОБЪЕКТОВ НА ИЗОБРАЖЕНИИ

ALGORITHM MODIFICATION OF PROCESSING THE GLOW EFFECT ON THE IMAGE

N. Terentev
R. Kildeev
T. Kolikova
T. Leontieva

Summary. A naive implementation of an image processing algorithm that produces a glowing effect on objects in a scene was presented. Various optimizations for each step of the naive implementation were described and tested. Conclusions were drawn about how these optimizations affect performance and the quality of the resulting image. As a result, a table of results was constructed for each of the presented methods of optimizing the glowing effect image processing algorithm.

Keywords: computer graphics, shaders, optimization methods, bloom.

Терентьев Никита Леонидович
Санкт-Петербургский политехнический университет Петра Великого
terentiev.nl@edu.spbstu.ru
Кильдеев Рустам Ильдарович
Санкт-Петербургский политехнический университет Петра Великого
kildeev.ri@edu.spbstu.ru
Коликова Татьяна Всеволодовна
Старший преподаватель, Санкт-Петербургский политехнический университет Петра Великого
tvk@ics2.ecd.spbstu.ru
Леонтьева Татьяна Владимировна
К.т.н., доцент, Санкт-Петербургский политехнический университет Петра Великого
leontyeva@ics2.ecd.spbstu.ru

Аннотация. Представлена наивная реализация алгоритма обработки изображения, дающего эффект свечения объектов на сцене. Описаны и протестированы различные вариации оптимизаций каждого из шагов наивной реализации алгоритма. Сделаны выводы о том, как данные оптимизации оказывают влияние на производительность, а также на качество получаемого изображения. В результате была построена таблица результатов каждого из представленных вариантов модифицированных способов обработки алгоритма эффекта свечения объектов на сцене.

Ключевые слова: компьютерная графика, шейдеры, методы оптимизации, блум.

Введение

За годы своего существования технологии компьютерной графики значительно изменялись, представляя революционные способы взаимодействия людей с компьютером. Начиная с первых шагов, представляющих собой элементарную 2D-графику, до передовой 3D-отрисовки и виртуальной реальности, компьютерная графика претерпела множество изменений во многих своих отраслях, включая: развлечения, образование, здравоохранение, архитектуру и другие. Последние достижения в технологии компьютерной графики проложили путь к большей реалистичности, предоставляя практически безграничные возможности для творчества и инноваций.

На данный момент технологии компьютерной графики стремительно развиваются, и один из часто используемых, используемых в этой области,— эффект свечения (bloom). Рассматриваемый эффект улучшает визуальное качество изображений, добавляя светящийся ореол вокруг ярких объектов в сцене, что приводит к более реалистичному и привлекательному виду. Однако он требует значительных вычислительных затрат, что затрудняет его применение в приложениях реального времени, осуществляющих воспроизведение высокого количества кадров в секунду. В этой статье мы представляем исследование по различным методам оптимизации алгоритма блума. Будет рассмотрено несколько алгоритмов, проведено сравнение их производительности и качества изображения. Цель

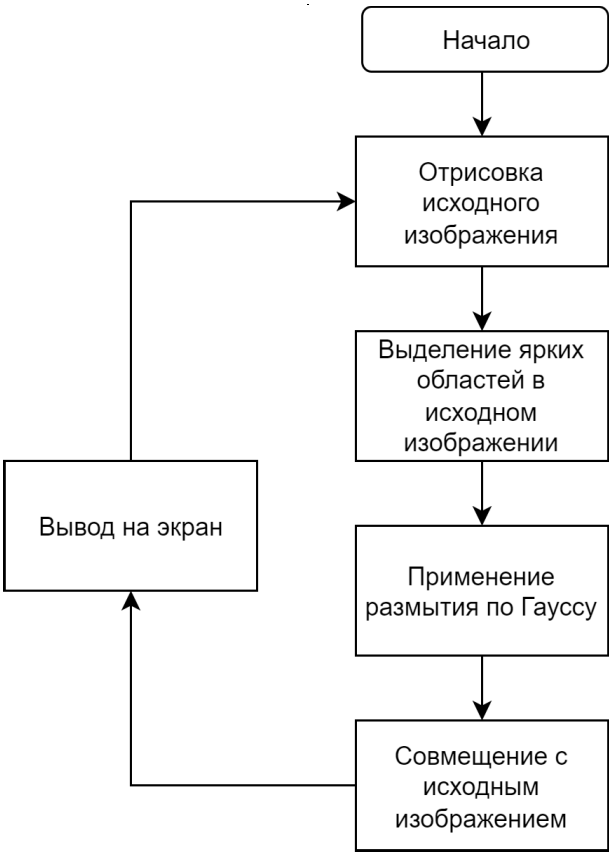

Рис. 1. Блок-схема отрисовки сцены в наивной реализации

этой статьи — внести вклад в развитие реализации эффекта блума в приложениях компьютерной графики путем предоставления основ для создания оптимизированного решения, которое будет обеспечивать баланс между визуальным качеством и вычислительными затратами.

Наивная реализация

В простом представлении, отрисовка сцены, в которой присутствует свечение объектов, можно представить в виде блок схемы:

Алгоритм извлекает из исходного изображения фрагменты, уровень яркости которых превышает определенный порог. Чтобы выделить яркие области изображения, сначала нужно определить, что считается «ярким». Одним из распространенных подходов является использование порогового значения для выявления пикселей, которые ярче определенного уровня. Пороговое значение может быть фиксированным.

Предполагается, что цвета пикселей будут соответствовать исходному изображению, а их яркость будет считаться по следующей формуле:

$$c = \frac{max(0, b - t)}{b}$$

где $b = max(\mathrm{C}_r, C_g, C_b)$ предполагая C_r, C_g, C_b как нормализованные компоненты цвета;

t — пороговое значение яркости

В связи с тем, что t является константой, контрастность изображений снижается. Любой пиксель, превышающий определенный уровень яркости, воспринимается как чисто «белый».

Затем применяется фильтр размытия к текстуре пороговой яркости. Одним из самых простых алгоритмов, который можно применить, является размытие по Гауссу.

Он работает путем свертки изображения с гауссовым ядром, которое представляет собой колоколообразную функцию, присваивающую веса каждому пикселю изображения. Ядро центрируется на текущем обрабатываемом пикселе, а его веса плавно уменьшаются с удалением от центра, что приводит к плавному смешиванию соседних пикселей [1]:

Таблица 1. Матрица весов при $\sigma = 0.7$

0.00000	0.00000	0.00001	0.00003	0.00001	0.00000	0.00000
0.00000	0.00009	0.00198	0.00548	0.00198	0.00009	0.00000
0.00001	0.00198	0.04219	0.11705	0.04219	0.00198	0.00001
0.00003	0.00548	0.11705	0.32472	0.11705	0.00548	0.00003
0.00001	0.00198	0.04219	0.11705	0.04219	0.00198	0.00001
0.00000	0.00009	0.00198	0.00548	0.00198	0.00009	0.00000
0.00000	0.00000	0.00001	0.00003	0.00001	0.00000	0.00000

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

где x и y — координаты пикселя относительно центра ядра;

σ — стандартное отклонение гауссовского распределения;

e — основание натурального логарифма.

Константа нормализации $\frac{1}{2\pi\sigma^2}$

гарантирует, что сумма всех весов ядра равна единице, сохраняя яркость изображения.

Пример матрицы весов ядра радиуса 7 для $\sigma = 0.7$ приведен в таблице 1:

Для применения размытия к изображению производится свертка изображения с гауссовским ядром по следующей формуле:

$$I'(x,y) = \frac{1}{W} * \sum\sum (I(x+i, y+j) * G(i,j))$$

где $I(x, y)$ — значение яркости пикселя в точке (x, y) на входном изображении;

$I'(x, y)$ — соответствующее значение интенсивности на выходном изображении после размытия;

W — сумма всех весов ядра, причем суммирование ведется по всем позициям ядра (i, j) относительно его центра.

Полученное изображение с размытыми яркими участками накладывается на исходное, тем самым получается эффект свечения объектов.

Оптимизация алгоритма определения уровней яркости

Так как в реальных условиях не требуется, чтобы все объекты на сцене имели эффект свечения, то его обработка производится только на некоторых участках изображения, из-за чего возникает проблема с резкими переходами между ними. Если изображение имеет плавные тональные переходы, оно кажется более реалистичным и менее искусственным. Это происходит потому, что мозг человека настроен на плавные, постепенные изменения света и цвета.

Для решения этой проблемы вводится еще одна константа под названием "мягкий порог".

Модифицированная формула для определения яркости изображения будет иметь вид [2]:

$$c = \frac{max(s, b-t)}{b}$$

где $s = \frac{min(max(0,b-t+k),2k)^2}{4k}$;

$k = t * t_s$;

b — максимальная составляющая цвета;

t — значение обычного порога;

t_s — значение мягкого порога

Используя ее, можно смягчить пороговое значение и сделать переход более гладким вместо того, чтобы сразу обрезать значение нулём.

Оптимизация алгоритма размытия изображения

Так как наивная версия алгоритма использует свертку, то сложность алгоритма будет равна $O(n^2)$. Если произвести тестирование на видеочипе GTX 1060 компании Nvidia с ядром размерности 11 при разрешении экрана 1920x1080, то время одного прохода будет приблизительно 1.7 миллисекунд, что не является приемлемым результатом. Более того, для повышения реалистичности эффекта требуется множество таких проходов, что ведет к сильному снижению показателя частоты отрисовки кадров в секунду. Для решения этой проблемы было разработано множество вариантов оптимизаций алгоритма размытия, но в данном случае будут рассмотрены только наиболее выгодные, в первую очередь по производительности.

Одним из основных способов повышения производительности является предвычисление весов матрицы Гаусса [3]. Далее к ним можно обращаться при помощи

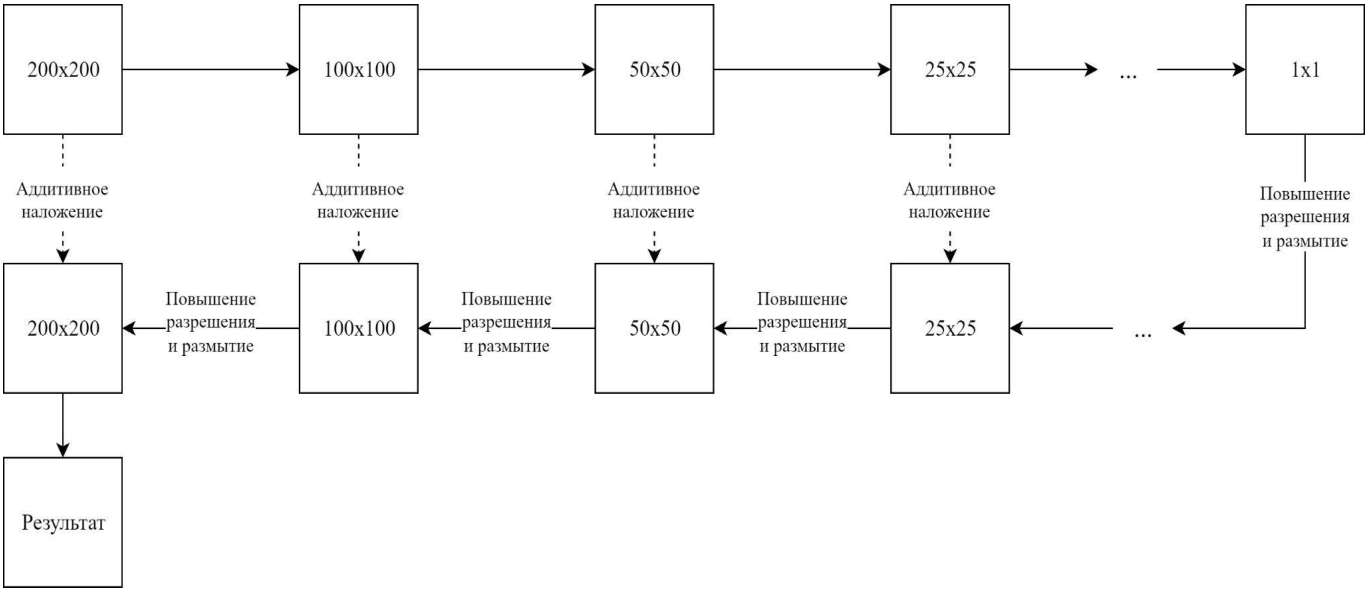

Рис. 2. Алгоритм создания эффекта свечения от «Sledgehammer Games»

сдвигов. Тестирование показало, что время обработки кадра снизилось до 0.37 миллисекунд.

Второй подход учитывает разделяемую свертку функции Гаусса, что приводит к делению алгоритма на 2 прохода: горизонтальный и вертикальный. Таким образом, сложность алгоритма становится $O(n)$. Получаем, что кадр теперь обрабатывается за 0.28 миллисекунд, что уже неплохо, но этого всё еще недостаточно.

Можно произвести сжатие исходного изображения и применить к нему эффект размытия по Гауссу. Это позволит значительно повысить производительность алгоритма, так как в таком случае требуется применять размытие на изображении с меньшим количеством пикселей. Однако вследствие такого подхода картинка может стать «блочной», особенно при использовании небольших ядер. Это может представлять проблему, например, при обработке кадров, в которых происходит какое-либо движение, так как во время него эффект блочности становится более выраженным. Для решения этой проблемы можно применить интерполяцию между размытым изображением с пониженной дискретизацией и конечным изображением при использовании малых значений ядра. Такой подход может помочь сгладить блочность и сделать переход более плавным. Однако этого не всегда достаточно, особенно при создании сложных эффектов. В таком случае использование промежуточных буферов с низким может привести к дальнейшему ухудшению качества изображения. Из-за такого набора проблем и сложностей требуется обратиться к альтернативным способам оптимизации алгоритма размытия изображения.

Комплексный подход к оптимизации алгоритма свечения

Для решения проблемы снижения детализации изображения компания «Sledgehammer Games» рассказала об их алгоритме обработки эффекта свечения [4]. Алгоритм состоит из 4 этапов:

1. Проходы понижения разрешения изображения до размера изображения 1x1 (с сохранением промежуточных результатов) .
2. Проход повышения разрешения изображения и наложение на него алгоритма размытия.
3. Аддитивное наложение размытого изображения с пониженным разрешением такой же размерности.
4. Повторение этапа 2 до тех пор, пока изображение не примет разрешение, равное исходному.

Алгоритм понижения разрешения основывается на ядре, использующем 13 билинейных фильтраций [5] в изображении для дальнейшего смешивания полученных значений с помощью средневзвешенных значений.

Для пояснения алгоритма обозначим координаты текущего пикселя как (u, v), а его размер как.

$$d = \frac{1.0}{\text{исх.разрешение экрана}}$$

Затем, рассчитаем координаты 13 точек выборки следующим образом:

Точка a: (u - 2d, v + 2d)
Точка b: (u, v + 2d)

Таблица 2. Распределение весов точек при обработке алгоритма понижения разрешения изображения

Точки	Вес
e	0.125
a, c, g, i	0.03125
b, d, f, h	0.0625
j, k, l, m	0.125

Таблица 3. Распределение весов точек при обработке алгоритма повышения разрешения изображения

Точки	Вес
e	4
b, d, f, h	2
a, c, g, i	1

Точка c: (u + 2d, v + 2d)
Точка d: (u - 2d, v)
Точка e: (u, v) (текущий пиксель)
Точка f: (u + 2d, v)
Точка g: (u - 2d, v — 2d)
Точка h: (u, v — 2d)
Точка i: (u + 2d, v — 2d)
Точка j: (u - d, v + d)
Точка k: (u + d, v + d)
Точка l: (u - d, v — d)
Точка m: (u + d, v — d)

Далее к каждой точке применяется взвешенное распределение, данные веса представляют собой «вклад» каждой точки выборки в конечный результат. Они были подобраны с условием, что их сумма равняется 1.

После чего общая сумма всех точек записывается на изображение меньшего разрешения.

Алгоритм повторяется до тех пор, пока размер итогового разрешения не станет 1x1.

Алгоритм повышения разрешения использует ядро 3x3 для дискретизации окружающих пикселей.

Рассчитывается 9 точек выборки по аналогии понижению разрешения:

Точка a: (u - qB, v + qB)
Точка b: (u, v + qB)
Точка c: (u + qB, v + qB)
Точка d: (u - qB, v)
Точка e: (u, v) (текущий пиксель)
Точка f: (u + qB, v)
Точка g: (u - qB, v — qB)
Точка h: (u, v — qB)
Точка i: (u + qB, v — qB)

константа qB описывает радиус фильтрации пикселей (достаточно значения 0,5)

Далее к каждой точке применяются веса 3x3, используя шатровый фильтр [6], который вычисляет средневзвешенное значение цветов пикселей в пределах круговой области. Используя веса фильтра, получаем:

После чего общая сумма всех точек делится на 16. В данном случае 16 означает количество пикселей изображения, цвет которых алгоритм использовал при чтении 9 точек с изображения.

При тестировании такого подхода результирующая производительность повысилась до 0.18 миллисекунд.

Алгоритм размытия Kawase

В 2003 году Майк Кавасе [7] предложил метод обработки изображений с помощью фильтра, который предполагает, что каждый проход добавляет небольшое количество размытия, в результате чего на последних итерациях результат становится похожим на алгоритм размытия Гаусса.

Его идея заключается в использовании встроенной аппаратной поддержки билинейной фильтрации на видеокартах. Алгоритм состоит из n последовательных итераций, которые производят вычисление усред-

Таблица 4. Результаты исследования времени обработки одного кадра при различных вариантах модификаций исходных алгоритмов

Алгоритм	Время выполнения (мс)
Наивная реализация алгоритма	1.7
Предвычисление весов Гаусса	0.37
Двойной проход Гаусса	0.28
Алгоритм размытия Kawase 11 итераций	0.125
Алгоритм размытия Kawase 35 итераций	0.27
Комплексный подход	0.18
Вычислительный шейдер	0.07

Исходя из результатов, можно сказать, что комбинация всех вариантов оптимизаций, рассмотренных в статье, дала существенный прирост в скорости обработки одного кадра.

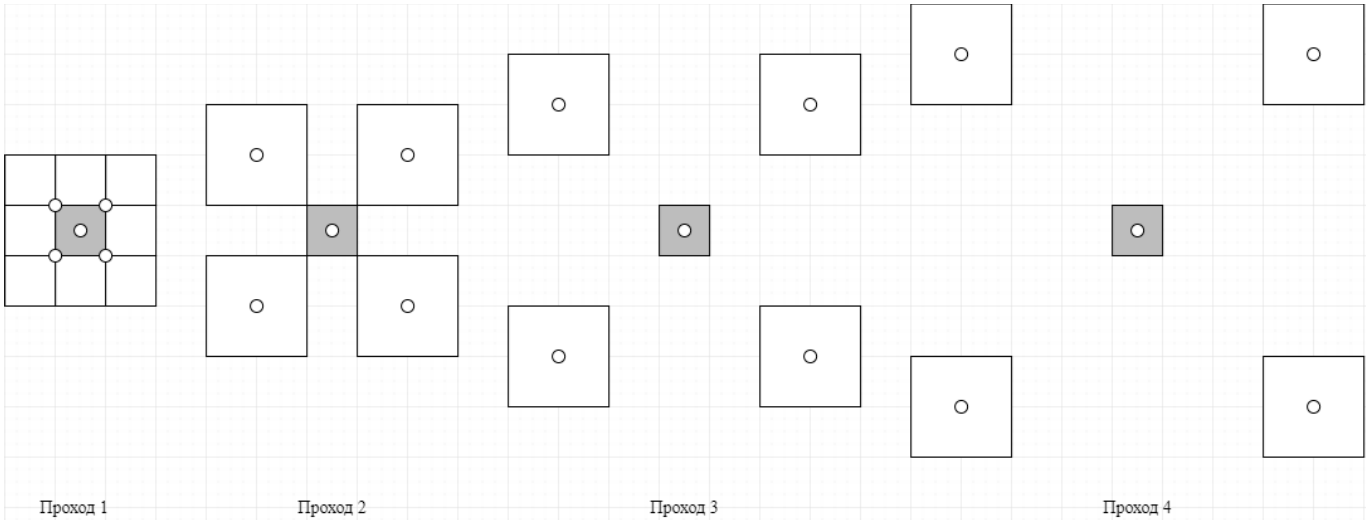

Рис. 3. Обработка одного пикселя на 4-х проходах алгоритма Kawase

ненных значений пикселей (билинейная фильтрация) из соседних областей 2x2, увеличивая при этом радиус их удаления от исходной точки.

Алгоритм демонстрирует высокую производительность в 0.125 мс. Однако качество размытия изображения при 11 итерациях не похоже на размытое с помощью 11 гауссового ядра изображение. Поэтому количество итераций потребовалось увеличить и только при 35 получился результат, который был похож на гауссовское распределение. Хотя производительность в таком случае не была идеальной (0.27 мс), этот показатель можно варьировать для достижения более высокой производительности в ущерб качеству изображения.

Оптимизация метода Гаусса при помощи вычислительного шейдера

Задавшись вопросом еще более углубленной оптимизации предыдущего алгоритма, можно использовать идеи понижения качества изображения и билинейного преобразования при чтении изображения, тем самым получить простую и эффективную реализацию Гаусса, используя специальный вычислительный шейдер на видеокарте. Стоит отметить, что во время работы алгоритма сжатия изображения можно объединить этап определения уровней яркости изображения и первый проход понижения размерности. Можно заметить, что 4–5 проходов для разрешения экрана 1920x1080 достаточно, чтобы сэкономить на вычислениях изображе-

ния с меньшим количеством пикселей без каких-либо потерь в качестве. В отличие от случая использования обычного шейдера, в вычислительном шейдере появляется возможность использования общей памяти графического процессора между разными вычислительными ядрами. Тем самым можно распределить задачу на разные вычислительные ядра, которые будут использовать общий буфер изображения.

Более того, можно использовать линейную интерполяцию между разными уровнями разрешений изображения [8], тем самым "сократив" весь процесс сжатия до 1 шага при помощи взятия среднего значения между 1 и 2, 3 и 4, 4 и 5 проходом алгоритма. Описанный процесс очень удобен, так как позволяет выполнить все необходимые действия, например, на этапе тонального отображения.

Указанных операций достаточно, чтобы время выполнения алгоритма создания эффекта свечения стало равным 0.03 миллисекунд. Обладая текущей производительностью, можно повысить размер ядра Гаусса до 33 и получить итоговую производительность в 0.07 миллисекунд, что будет являться хорошим балансом между качеством изображения и скоростью расчетов.

Заключение

В ходе исследования были изучены различные методы оптимизации наивного алгоритма эффекта свечения объектов в сцене, направленные на улучшение как визуальной составляющей, так и производительности алгоритма. Тестирование было выполнено на видеочипе Nvidia GTX 1060. Результаты исследования приведены в таблице 4.

ЛИТЕРАТУРА

1. Forsyth D. A. Computer Vision: A Modern Approach: 2nd Edition / D. A. Forsyth, J. Ponce. // New Jersey: Prentice Hall, 2011. V. P. 792
2. D.L. Donoho. «Denoising by Soft thresholding» IEEE Trans on Information Theory. 1995, V. 41, N. 3, P. 613–627.
3. Efficient Gaussian blur with linear sampling [Электронный ресурс]. URL: https://www.rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/ (дата обращения: 25.02.2023)
4. Jorge J. NEXT GENERATION POST PROCESSING IN CALL OF DUTY: ADVANCED WARFARE [Электронный ресурс]. URL: http://advances.realtimerendering.com/s2014/index.html (дата обращения: 02.03.2023)
5. Phill D. Bicubic Filtering in Fewer Taps. Shiny Pixels. [Электронный ресурс]. URL: https://vec3.ca/bicubic-filtering-in-fewer-taps/ (дата обращения: 21.02.2023)
6. S.R. Marschner and R. J. Lobb. An evaluation of reconstruction filters for volume rendering. In Proceedings of the conference on Visualization'94, IEEE Computer Society Press, 1994. V. P. 100–107
7. Masaki K. Frame Buffer Post Processing Effects in DOUBLE-STEAL [Электронный ресурс]. URL: https://genderi.org/frame-buffer-postprocessing-effects-in-double-s-t-e-a-l-wreckl.html (дата обращения 24.02.2023)
8. Sungkil Lee, Gerard Jounghyun Kim, Seungmoon Choi. Real-time depth-of-field rendering using anisotropically filtered mipmap interpolation. IEEE Transactions on Visualization and Computer Graphics 15, 3 2009. V. P. 453–464.