О вычислении полей расстояний для многоугольников на плоскости

Павел Воронин, Андрей Адинец
Факультет Вычислительной Математики и Кибернетики
Московский Государственный Университет им. Ломоносова
{pavel.voronin, adinetz}@gmail.com

Аннотация

В статье рассматривается задача вычисления значений поля расстояний для многоугольника в узлах неравномерной сетки на плоскости. Описывается несколько подходов: решение "в лоб", с последующей оптимизацией под векторные и многопоточные вычисления, доступные на большинстве совеременных процессоров; использование иерархических пространственных структур данных, позволяющих оценивать расстояния между группами примитивов; наконец, перенос вычислений на графический процессор (при помощи системы C$). Приводится детальный сравнительный анализ алгоритмов (скорости их работы, ограничений применимости, сильных и слабых сторон). Использование описанных методов демонстрируется на примере задачи построения сетки, приближающей заданную фигуру, и задачи отыскания области, свободной от подвижных нарушителей, при известной траектории движения ищущего.

Keywords: *поля расстояний.*

1. ВВЕДЕНИЕ

Полем расстояний для некоторого объекта называется функция, значение которой в произвольной точке равно расстоянию от этой точки до границы объекта. Дополнительно можно ввести знак поля: точкам внутри объекта соответствуют отрицательные значения, а точкам вне него — положительные.

Поля расстояний получили широкое распространение в самых различных областях машинного зрения, компьютерной графики и математического моделирования. Двумерные дискретные поля расстояний, среди прочего, применяются для сегментации, выделения скелета и сравнения формы фигур [1]. Трёхмерные дискретные поля расстояний используются для обработки и анализа медицинских трёхмерных изображений [2]. Непрерывный вариант трёхмерных полей расстояний применяется в физическом моделировании, определении столкновений, анимации, трассировке лучей и системах автоматизированного проектирования [3]. Непрерывные двумерные поля расстояний нашли своё применение в таких разных областях, как навигация роботов [4], хранение векторных шрифтов [5], построение равномерных сеток с заданной границей [6] и расчёт информационных множеств в задачах динамического поиска [7].

Непрерывные поля расстояний обычно задаются в виде замеров на узлах некоторой сетки, с возможностью интерполяции значения в любой точке. Для вычисления приближённых значений поля разработаны быстрые методы как для равномерных сеток [8, 9], так и для сеток общего вида [10]. С другой стороны, описанные в литературе алгоритмы точного вычисления полей расстояний [11, 12, 13, 14] работают исключительно с равномерными сетками.

Данная статья посвящена вычислению значений поля расстояний в узлах сетки общего вида. В работе отдельно рассматриваются случаи полей расстояний со знаком и без; расчёта поля во всех точках и только в принадлежащих некоторой полосе вокруг границы объекта; разового вычисления и многократного вычисления для одного и того же набора точек, на разных объектов.

Нами было реализовано три подхода: решение "в лоб", полным перебором, но с низкоуровневой оптимизацией кода (SSE, многопоточность); использование иерархических пространственных структур данных; перенос вычислений на графический процессор (при помощи системы C$ [15]). Был проведён подробный сравнительный анализ реализованных алгоритмов для различных сценариев работы. Приводится несколько примеров использования реализованных подходов для решения практических задач.

2. РЕШЕНИЕ ПОЛНЫМ ПЕРЕБОРОМ

Введём обозначения. Пусть даны набор точек $X = \{x_i\}_{i=1}^N$ и многоугольник A со сторонами $\{e_j\}_{j=1}^M$. Требуется найти значение поля расстояний многоугольника DF_A во всех точках из набора X: $DF_A(x_i)$, $1 \le i \le N$. Обозначим расстояние от точки x до отрезка e как $\rho(x, e)$. Тогда, согласно определению, значение поля расстояний DF_Aв точке x равно

$$DF_A(x) = \sigma_A(x) \cdot \rho(x, A), \text{ где}$$

$$\rho(x, A) = \min_{1 \le j \le M} (\rho(x, e_j)),\ \sigma_A(x) = \begin{cases} -1, & x \text{ внутри } A, \\ +1, & \text{иначе.} \end{cases}$$

Рассмотрим отдельно алгоритмы вычисления расстояния $\rho(x, A)$ и знака $\sigma_A(x)$.

2.1 Вычисление расстояний до границы многоугольника

В соответствии с приведённой выше формулой, имеем следующий алгоритм для вычисления расстояний:

Algorithm 1 Расстояния, полный перебор.

```
Для каждого i от 1 до N
    ρ(x_i, A) = +∞
    Для каждого j от 1 до M
        d_ij = ρ(x_i, e_j)
        Если d_ij < ρ(x_i, A), то
            ρ(x_i, A) = d_ij
```

Для повышения скорости работы этого элементарного алгоритма мы использовали следующие приёмы.

Вместо $\rho(x_i, e_j)$ вычисляется его квадрат. Корень берётся один раз, из найденного минимума.

Все необходимые для вычисления $\rho(x_i, e_j)$ данные (координаты точек; координаты концов отрезка, координаты его направляющего вектора, длина) предпросчитываются и хранятся не в виде массива структур, а виде структуры массивов.

Используются параллельные вычисления, доступные на большинстве современных систем. Вершины сетки делятся на столько равных групп, сколько нам доступно потоков — и каждая группа обрабатывается в отдельном потоке (при помощи кросс-платформенной библиотеки Intel TBB[1] [16]).

Используются векторные вычисления, также доступные во всех современных процессорах (например, технология SSE (Streaming SIMD Extensions) на процессорах Intel и AMD).

2.2 Определение принадлежности многоугольнику

Для того, чтобы вычислить значение функции $\sigma_A(x)$, необходимо определить, принадлежит ли точка x многоугольнику A. Как показано в работе [17], в случае многоугольников с большим числом сторон наиболее эффективным является алгоритм, основанный на подсчёте пересечений сторон многоугольника с горизонтальным лучом.

Итак, пусть заданы точка $p = (p_x, p_y)$ и многоугольник A со сторонами $E = e_i{}_{i=1}^{K}$, $e_i = (p^{e_i}, q^{e_i})$. Выпишем алгоритм определения принадлежности точки многоугольнику (в качестве испускаемого луча R из соображений эффективности возьмём луч $y = p_y, x \geq p_x$).

Algorithm 2 Знак, полный перебор.

Для каждого i от 1 до N
    Если ЧислоПересечений(x_i, R, E) — нечётное, то
        $\sigma_A(x) = -1$, иначе
        $\sigma_A(x) = +1$.

Функция ЧислоПересечений(x, R, E)
    $n = 0$.
    Для каждого e из E
        Если ЛучПересекает(x, R, e), то
            $n = n + 1$.
    Верни n.

Функция ЛучПересекает(x, R, E)
    Если $(p_y^e - p_y)(q_y^e - p_y) > 0$, то Верни Нет.
    Если $(p_x^e < p_x)$ и $(q_x^e < p_x)$, то Верни Нет.
    Если R не пересекает e, то Верни Нет.
    Верни Да.

Для некоторых классов многоугольников существуют более простые и эффективные алгоритмы определения принадлежности им заданной точки. В частности, для выпуклых, монотонных и звёздных многоугольников легко строятся алгоритмы, основанные на двоичном поиске [17, 18].

3. ИЕРАРХИЧЕСКИЙ ПОДХОД

При расчётах по алгоритму 1 вычисляются расстояния между от всех точек до всех отрезков. Такой подход не учитывает пространственной когерентности входных данных: расстояния от точки до двух отрезков, находящихся на малом расстоянии друг от друга, отличаются слабо; аналогично, слабо отличаются расстояния до отрезка от двух точек, расположенных близко друг к другу. Поэтому для сокращения объёма вычислений мы применили идеи, использованных в статье [19] для ускорения нахождения кратчайшего расстояния между двумя объектами. Метод основан на использовании иерархий ограничивающих тел для оценки расстояний между группами точек и отрезков с последующим исключением из рассмотрения тех пар групп, которые заведомо не могут быть ближайшими друг к другу. В качестве иерархий ограничивающих тел как для точек, так и отрезков мы использовали kd-деревья [20] (в листьях дерева содержится не менее некоторого фиксированного числа точек $(2^5 - 2^8)$ и по одному отрезку соответственно).

Рассмотрим алгоритм нахождения расстояний $\rho(x_i, E)$ от точек $X = x_i{}_{i=1}^{N}$ до множества отрезков $E = e_j{}_{j=1}^{K}$. Будем обозначать kd-дерево, соответствующее множеству отрезков L как $T(L)$; множество отрезков, соответствующее kd-дереву K — как $S(K)$; поддеревья kd-дерева K — как K_1 и K_2; kd-дерево, соответствующее множеству точек V — как $T(V)$; множество точек, соответствующее kd-дереву K — как $P(K)$.

Algorithm 3 Расстояния от множества точек до множества отрезков, иерархическое отсечение.

$\rho(X, E) = +\infty$
Очередь $Q = \{(T(X), T(E))\}$
Пока Q непуста
    $(t^V, t^S) = Q.pop()$
    Если t^V и t^S листья, то
        Для всех v из $V(t^V)$
            Если $\rho(v, E) < \rho(v, S)$, то $\rho(v, E) = \rho(v, S)$.
        $\rho(V(t^V), E) = \max_{v \in V(t^V)} \rho(v, E)$
    иначе
        $\widetilde{d}_{min} = \widetilde{\rho}_{min}(t^V, t^S)$
        Если $\rho(t^V, E) > \widetilde{d}_{min}$
            $\widetilde{d}_{max} = \widetilde{\rho}_{max}(t^V, t^S)$
            Если $\widetilde{d}_{max} < \rho(t^V, E)$, то $\rho(t^V, E) = \widetilde{d}_{max}$
            Если t^V лист, то
                $Q.push(t^V, t_1^S)$, $Q.push(t^V, t_2^S)$
            иначе
                $\rho(t_1^V, E) = \rho(t_2^V, E) = \rho(t^V, E)$
                Если t_S лист, то
                    $Q.push(t_1^V, t^S)$, $Q.push(t_2^V, t^S)$
                иначе
                    $Q.push((t_1^V, t_1^S) Q.push(t_2^V, t_1^S)$
                    $Q.push(t_1^V, t_2^S)$, $Q.push(t_2^V, t_2^S)$

Алгоритм совершает обход деревьев в ширину и для каждой пары нелистовых поддеревьев вычисляет верхнюю $\widetilde{\rho}_{max}$ и нижнюю $\widetilde{\rho}_{min}$ оценки расстояний между ними. Если текущее значение ρ меньше $\widetilde{\rho}_{min}$, то данные поддеревья не является друг для друга ближайшими, и пара исключается из дальнейшего рассмотрения. В противном случае процедура позднее повторяется для их поддеревьев.

В качестве нижней оценки берётся расстояние между описанными прямоугольниками поддеревьев. В качестве верхней — минимум максимальных расстояний от вершин отрезков, лежащих на границе описанного прямоугольника одного поддерева до описанного прямоуголь-

[1] http://www.threadingbuildingblocks.org/

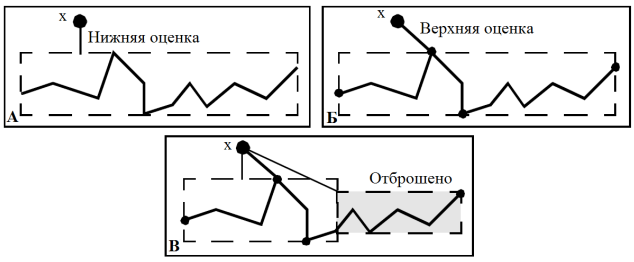

Figure 1: Алгоритм 3: А) Нижняя оценка; Б) Верхняя оценка; В) Отсечение группы отрезков, заведомо не являющихся ближайшими.

ника другого (заметим, что верхняя оценка сложнее нижней, но и вычисляется далеко не всегда).

Работа алгоритма для случая одной точки проиллюстрирована на рис. 1.

Заметим, что если первую строчку алгоритма 3 заменить с $\rho(X,E) = +\infty$ на $\rho(X,E) = d$, то он будет вычислять фукцию $\min(d, \rho(X,E))$.

Тем же образом можно ускорить проверку принадлежности точки многоугольнику (алгоритм 2): опять будем обходить дерево в ширину и будем отбрасывать такие пары поддеревьев, где отрезки лежат левее, выше или ниже точек, проверяя пересечения только с оставшимися после этого отрезками.

4. ВЫЧИСЛЕНИЯ НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ

Как уже отмечалось в п. 2.1, алгоритм расчёта значений поля расстояний на сетке для многоугольника методом полного перебора легко разделить на большое количество параллельных потоков вычислений: расстояние для каждой точки ищется независимо ото всех других. Это свойство делает данный алгоритм идеальным кандидатом для переноса на современные видеокарты. В качестве инструментария для такого переноса мы выбрали язык C$[2].

C$ является высокоуровневым языком программирования, который позволяет писать короткие и переносимые программы, использующие одинаковые исходные коды для различных архитектур графических процессоров (ГПУ). Следовательно, язык C$ не может предоставить программистам доступ к низкоуровневым особенностям конкретных графических платформ. Как следствие, он должен уметь использовать эти особенности автоматически, чтобы позволять достигать уровня производительности, сравнимого с ручной оптимизацией. Это достигается за счет сочетания различных приемов хранения данных и трансляции и оптимизации кода. Здесь приводится краткий обзор используемых приёмов, более подробную информацию можно найти в статье [15].

Данные, используемые в C$-программах на ГПУ, хранятся в массивах C$. В отличие от других языков, C$ не специфицирует ни формат представления данных, ни места, где они хранятся. Массивы C$ могут храниться в памяти хоста или ГПУ, или в обоих местах сразу, и перемещаться между ними прозрачно для программиста. Когда массивы C$ находятся в памяти ГПУ, формат их представления может меняться от одного исполняемого ядра к другому, чтобы наилучшим образом соответствовать решаемой задаче.

C$ представляет код ГПУ как направленный ациклический граф функциональных операций, или ФДАГ (функциональный ДАГ). Его трансляция в код ГПУ состоит из нескольких этапов. На первом этапе граф разбивается на подграфы, каждый из которых примерно соответствует одному вызову ядра на ГПУ. Далее каждый подграф обрабатывается отдельно. Для каждого из них выбирается наилучшее его отображение на ГПУ. Для этого ставится оптимизационная задача минимизации количества вычислений и обменов данными с памятью. Она является задачей геометрической оптимизации [21]. Проблема решается с использованием релаксации и метода барьеров, что позволяет получить оптимальные размеры развёртывания циклов. Они, в свою очередь, передаются на этап генерации кода, на которой также выполняется сокращение общих индексных выражений.

Оптимизации, используемые в системе C$, позволяют использовать быструю память на чипе. Для ГПУ NVidia это разделяемая статическая память, а для ГПУ AMD — большой регистровый банк. Для ГПУ AMD это также позволяет выполнять автоматическую векторизацию. Для вычислительно ёмких задач использование методов системы C$ позволяет достигать ускорения в несколько раз, а иногда и на порядок, по сравнению с тривиальной версией. При этом получаемый код является относительно небольшим и переносимым между различными архитектурами ГПУ.

Для нашей задачи система показала себя очень хорошо: код получился компактным и заработал на видеокартах трёх разных архитектур. При этом автоматическая оптимизация увеличила и без того немаленькую скорость работы алгоритма в 5-9 раз.

5. СРАВНЕНИЕ

Поскольку скорость работы алгоритмов полного перебора (неоптимизированного, оптимизированного и C$-версии) никоим образом не зависит от конкретных данных, с которыми они работают, не имеет никакого значение, на каких данных их тестировать (когда требовалось N точек, мы брали равномерную сетку $\sqrt{N}$x$\sqrt{N}$; в качестве M-угольника мы брали правильный). Однако совсем не так обстоит дело с иерархическим алгоритмом, и чтобы проверить, какой эффект оказывает структура данных на его работу, мы сгенерировали представительные тестовые наборы нескольких классов многоугольников:

- Набор псевдослучайных простых многоугольников, сгенерированных при помощи библиотеки RPG [3], основанной на эвристических алгоритмах, описанных в статье [22].
- Набор псевдослучайных ортогональных многоугольников, сгенерированных по алгоритму, описанному в работе [23]. Алгоритм основан на сложных эвристиках, позволяющих по заданному количеству вершин генерировать широкий класс сеточных многоугольников, с последующей преобразованием в ортогональные многоугольники общего вида.
- Набор случайных звёздных многоугольников, сгенерированных при помощи библиотеки RPG. Вычис-

[2] http://www.codeplex.com/cbucks

[3] http://www.cosy.sbg.ac.at/~held/projects/rpg/rpg.html

ления основаны на точном алгоритме [22], за полиномиальное время находящем все звёздные многоугольники с заданными вершинами.

В таблице 1 приведена зависимость (усреднённого) времени, которое требуется на вычисление поля расстояний для 50-угольников разных типов, от размеров сетки. Видно, что результаты для звёздных многоугольниках и многоугольниках общего вида практически идентичны, в то время как для ортогональных скорость работы выше в 1,3 раза. Скорее всего, это объясняется выбранной пространственной структурой данных: в основе kd-деревьев лежит разделение прямыми, параллельными осям координат, что отлично отражает структуру ортогональных полигонов. В то же время, сравнительна небольшая разница между ними и многоугольниками общего вида говорит о достаточно хорошем качестве приближения.

В таблице 2 приведено среднее время работы описанных алгоритмов для простых 50-угольников и сеток разных размеров. Для больших размеров сеток иерархический метод оказывается в 3 раза быстрее переборного, а ГПУ-метод — почти в 30 раз быстрее иерархического. Для больших многоугольников первый показатель увеличивается, а второй — падает (оба приблизительно до 10).

Заметим, наконец, что из всех приведённых алгоритмов только иерархический позволяет эффективно вычислять расстояние в некоторой полосе вокруг многоугольника, и в задачах, где этого достаточно, может работать быстрее ГПУ-алгоритма (для 1000-угольника, 2^{20} вершин и полосы толщиной 0.01 в квадрате $[0,1] \times [0,1]$ — в 4 раза).

6. ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

В качестве примеров использования рассмотренных выше алгоритмов нам хотелось бы привести две задачи, одинаково активно использующие вычисление полей расстояний, но делающие это принципиально различными способами. В первой задаче сетка фиксирована, а многоугольники всё время разные, причём точный результат требуется лишь в узкой полосе, окружающей границу многоугольника. Во второй же задаче фиксирован многоугольник, а сетки всё время меняются, причём точный результат нужен для всех их узлов.

6.1 Расчёт области, свободной от нарушителей

Пусть на некоторой многоугольной плоской области передвигаются ищущие и активно уклоняющиеся объекты. Скорость первых — V, вторых — $U < V$. Пусть условием обнаружения искомого объекта является попадание в прямую видимость ищущего. Для заданной траектории $P(t), t \in [0,T]$ остаточной областью называется переменное во времени множество $A(t)$ такое, что в любой момент времени t^* при любом поведении искомых, они либо были обнаружены ранее, либо находятся гарантированно вне $A(t^*)$. Рассмотрим задачу нахождения нахождения $A_k = A(t_k)$, $t_k = k\Delta t$, $\Delta t = \frac{T}{N}$, $0 \le k \le N$.

В статье [7] предложен итерационный алгоритм приближённого расчёта остаточных областей, использующий для их представления поля расстояний. Для плоской задачи и условия прямой видимости он записывается следующим образом:

0) $DF_{A_0} = DF_{Vis_0}$

1) $DF_{old} = DF_{A_k} + \Delta U$,

2) $DF_{upd} = DF_{Vis_{k+1}}$,

3) $DF_{new} = \min(DF_{old}, DF_{upd})$,

4) $DF_{A_{k+1}} = DF_{ISO_0 DF_{new}}$.

где $Offset$ — операция сужения\расширения, $ISO_0 DF$ — нулевая линия уровня поля, а Vis_k— многоугольник видимости [24] для ищущего в момент времени t_k.

Как показано в работе [7], шаги 1, 3, 4 выполняются за линейное от числа узлов сетки время. Для вычисления многоугольника видимости требуется $O(M \ln M)$ операций, где M — число вершин границы области поиска [24] (очень быстрые алгоритмы этого типа представлены в библиотеке VisiLibity [25]). Таким образом, единственным потенциально медленным моментом остаётся вычисление поля расстояний многоугольника видимости. Заметим, однако, что достаточно вычислять поле расстояний в точках, удалённых от границы многоугольника видимости не более чем на $\Delta U + e_{max}$, где e_{max} — максимальная длина ребра сетки. Это позволяет воспользоваться наиболее быстрым вариантом иерархического алгоритма.

Неоптимизированный алгоритм полного перебора позволяет работать в интерактивном режиме с сетками размерами не более полутора тысяч вершин. Его оптимизированная версия расширяет это предел приблизительно до пяти тысяч. Оптимизированный иерархический алгоритм и C$-алгоритм позволяют работать с сетками размерами вплоть до ста тысяч вершин, причём на больших сетках иерархический алгоритм показывает несколько бóльшую производительность.

6.2 Генерация равномерных сеток

В работе [6] аппарат полей расстояний был использован для написания очень простого, но качественного генератора сеток (mesh generator). На вход алгоритму подаётся функция, вычисляющая значение поля расстояний для заданного тела в любой точке, некоторый прямоугольник, содержащий это тело, и желаемая средняя длина ребра сетки. Алгоритм начинает со случайной триангуляции прямоугольника, а затем постепенно подгоняет форму сетки к форме тела, прокцируя вершины сетки на границу тела (что легко делать при помощи поля расстояний). На выходе после определённого числа итераций (от десяти для простых тел до нескольких сотен для более сложных) получается достаточно однородная сетка, приближающая заданное полем расстояний тело.

Однако же при росте сложности геометрии исходного тела алгоритм становится малопрактичным: если тело задаётся многоугольником с несколькими тысячами сторон, то даже небольшие сетки могут строиться по несколько минут[4]. Дело в том, что на каждую итерацию алгоритма приходится четыре вычисления поля расстояний для всех узлов сетки. В выложенной автором алгоритма библиотеке поле расстояний для многоугольника рассчитывается неоптимизированным методом полного перебора. Мы заменили его нашими алгоритмами. Поскольку сетки при генерации меняются на каждой итерации, то с иерархическим алгоритмом генератор работает даже медленнее, чем с неоптимизированным полным перебором. Оптимизированный перебор ускоряет работу генератора всего в 2-5 раз. Зато перенос вычислений на ГПУ ускоряет генерацию в 20-30 раз.

[4]Мы пользовались реализацией, выложенной автором метода: http://www-math.mit.edu/~persson/mesh/

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Простые	0.78	0.93	1.25	1.87	2.66	4.36	7.18	13.11	24.18	46.17	94.06
Ортогональные	0.78	0.93	1.25	2.80	2.66	3.74	5.77	10.76	18.41	36.19	72.23
Звёздные	0.93	1.10	1.25	1.87	3.90	4.05	6.71	12.63	22.31	44.93	95.01

Table 1: Время работы иерархического метода для разных размеров сеток (мс).

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Перебор	0.31	0.62	1.40	2.18	4.37	9.05	17.47	36.03	70.98	141.49	287.04
Иерархический	0.78	0.93	1.25	1.87	2.66	4.36	7.18	13.11	24.18	46.17	94.06
ГПУ	0.01	0.01	0.02	0.04	0.07	0.11	0.24	0.47	0.94	1.88	3.75

Table 2: Время работы различных методов (мс).

7. REFERENCES

[1] R. Fabbri, O. M. Bruno, J. C. Torelli, and L. da F. Costa, "2d euclidean distance transform algorithms: A comparative survey," *ACM Computing Surveys*, , no. 40, pp. 2:1–2:44, 2007.

[2] J. Toriwaki and K. Mori, *Digital and Image Geometry: Advanced Lectures*, chapter Distance Transformation and Skeletonization of 3D Pictures and Their Applications to Medical Images, Springer, 2001.

[3] M. W. Jones, J. A. Baerentzen, and M. Sramek, "3d distance fields: A survey of techniques and applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12.

[4] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The field d* algorithm," *Journal of Field Robotics*, vol. 23.

[5] E. J. Jakubiak, R. N. Perry, and S. F. Frisken, "An improved representation for stroke-based fonts," in *Proceedings of ACM SIGGRAPH*, 2006.

[6] P.-O. Persson and G. Strang, "A simple mesh generator in matlab," *SIAM Review*, vol. 46.

[7] П. Воронин, "О новом подходе к расчёту и визуализации информационных множеств в задачах динамического поиска," in *Труды 18-й международной конференции ГрафиКон*, 2008.

[8] M. Akmal Butt and P. Maragos, "Optimum design of chamfer distance transforms," *IEEE Transactions on Image Processing*, vol. 7.

[9] R. Satherley and M. W. Jones, "Vector-city vector distance transform," *Computer Vision and Image Understanding*, vol. 82.

[10] J. A. Sethian, *Level Sets Methods and Fast Marching Methods. 2nd edition*, Cambridge University Press, 1999.

[11] F. Dachille and A. Kaufman, "Incremental triangle voxelization," in *Proceedings of Graphics Interface*, 2000.

[12] S. Mauch, "A fast algorithm for computing the closest point and distance transform," Tech. Rep., Technical Report caltechASCI/2000.077.

[13] C. Sigg, R. Peikert, and M. Gross, "Signed distance transform using graphics hardware," in *Proceedings of IEEE Visualization*, 2003.

[14] A. Sud, M. A. Otaduy, and D. Manocha, "Difi: Fast 3d distance field computation using graphics hardware," *Computer Graphics Forum*, vol. 23.

[15] А. В. Адинец and М. А. Кривов, "Методы оптимизации программ для современных графических процессоров," in *Сборник трудов Всероссийской научной конференции «Научный сервис в сети Интернет-2008: решение больших задач»*, 2008.

[16] J. Reinders, *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*, O'Reilly Media, Sebastopol, CA, 2007.

[17] E. Haines, *Graphics Gems IV*, chapter Point in Polygon Strategies, Academic Press Professional, Cambridge, MA, 1994.

[18] Ф. Препарата and М. Шаймос, *Вычислительная геометрия: введение*, Мир, 1989.

[19] D. Johnson and E. Cohen, "A framework for efficient minimum distance computation," in *Proceedings of IEEE Conf. On Robotics and Animation*, 1998.

[20] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

[21] Ф. П. Васильев, *Методы оптимизации*, Факториал Пресс, 2002.

[22] T. Auer and M. Held, "Heuristics for the generation of random polygons," in *Proc. of 8th Canadian Conf. Computational Geometry*, 1996.

[23] A. P. Tomás and A. L. Bajuelos, "Generating random orthogonal polygons," in *Proc. of CAEPIA-TTIA*, 2003.

[24] T. Asano, S. K. Ghosh, and T. C. Shermer, *Handbook of Computational Geometry*, chapter Visibility in the plane, Elsevier, 2000.

[25] K. J. Obermeyer, "The visilibity library," http://www.VisiLibity.org, 2008.

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

Расширенная версия статьи и реализация описанных алгоритмов доступны для скачивания по адресу http://www.primaler.ru/cg/.