Научная визуализация, 2024, том 16, номер 1, страницы 1 - 18, DOI: 10.26583/sv.16.1.01

Модификации классических алгоритмов восстановления поверхностей для визуализации функции, заданной на прямоугольной сетке

Н.В. Мунц[1], С.С. Кумков[2]

Институт математики и механики им. Н.Н. Красовского УрО РАН, Екатеринбург, Россия

[1] ORCID: 0000-0003-3234-1267, natalymunts@gmail.com
[2] ORCID: 0000-0002-2690-5380, sskumk@gmail.com

Аннотация

Рассматриваются модификации алгоритмов визуализации вещественнозначных функций двух и трех аргументов, заданных на прямоугольной/параллелепипедальной сетке. В случае двух аргументов график функции является поверхностью, погруженной в трехмерное пространство. Подавляющее большинство систем научной визуализации предлагает процедуры визуализации такой поверхности, однако восстанавливают ее в предположении о непрерывности функции. Предлагается модификация этого алгоритма для случая разрывной функции. Кроме того, алгоритм удаляет «плато», возникающие после срезки функции на некотором уровне (для удаления слишком больших значений).

Визуализация функции трех аргументов подразумевает изображение ее множеств уровня, то есть областей пространства аргументов, где значения функции не превосходят некоторой величины. В случае сеточного задания функции такие множества являются «воксельными», то есть составленными из ячеек сетки. При этом требуется сглаживание поверхности таких множеств, которое осуществляется алгоритмом Marching Cubes и алгоритмами лапласовского семейства. Предлагается модификация алгоритма Marching Cubes для сохранения симметрии поверхности множества относительно координатных плоскостей, осей или некоторой точки, если визуализируемое множество обладает такой симметрией.

Ключевые слова: разрывная функция, функция двух аргументов, множества уровня, функция трех аргументов, сглаживание поверхности, воксельное множество, Marching Cubes, лапласовский алгоритм.

1. Введение

Многие современные вычислительные методы носят сеточный характер. Для численного конструирования той или иной функции на пространстве ее аргументов вводится регулярная или нерегулярная сетка, в узлах которой находятся некоторые значения, приближающие величину искомой функции. При этом возникает проблема представления результатов вычислений для исследования свойств получаемой функции или для отладки вычислительных процедур. Представление, получаемое на выходе вычислительной программы – большие объемы чисел (особенно большие в случае использования мелких сеток, что характерно при получении хорошего приближения искомой функции), плохо подходит для восприятия исследователем. Традиционный путь связан с использованием визуализации, то есть с восстановлением на основе полученного массива чисел тех или иных геометрических объектов, отражающих свойства изу-

чаемой функции, и дальнейшим их показом исследователю. При этом, конечно, для эффективной визуализации доступны только двух- или трехмерные объекты.

Для случая двумерного аргумента естественным представлением функции является ее график. Восстановление графика в таком случае является весьма простой процедурой. Каждая ячейка сетки триангулируется (если ячейки сетки не являются симплексами), и на каждом полученном симплексе производится линейная интерполяция. Так поступает большинство общеупотребительных средств научной визуализации – Matlab, MathCAD, GNUPlot и др. Однако такой подход выдает непрерывный график, что не соответствует природе функции, если она имеет разрывы. Кроме того, часто по смыслу задачи визуализируемая функция может быть *несобственной*, то есть может принимать бесконечные значения в некоторых узлах сетки, что также не учитывается стандартными процедурами. (При численных построениях бесконечные значения заменяются очень большими величинами.) Авторам неизвестны компьютерные библиотеки, содержащие алгоритмы визуализации разрывных графиков несобственных функций. Традиционные рекомендации в таких случаях предлагают вручную выявлять непрерывные ветви функции с конечными значениями и независимо изображать каждую непрерывную ветвь. Однако такие рецепты неудобны, поскольку требуют ручной обработки каждого графика, в том числе, и в ситуации, когда график представляет собой единую поверхность, имеющую разрыв лишь в какой-то своей части.

Если же аргумент функции является трехмерным, то предыдущий подход уже не работает, так как график в этом случае уже является четырехмерным объектом. Четырехмерные объекты эффективно не визуализируются, если только не имеют регулярную структуру (случай шаров, эллипсоидов, правильных многогранников). Численно построенные функции, как правило, не обладают регулярностью и поэтому непосредственно не визуализируются. Традиционным подходом в этом случае является визуализация множеств уровня функции, то есть областей пространства, в которых функция не превосходит некоторой величины. Обладая множествами уровня для различных значений, можно представить устройство функции в целом. Однако при этом так же возникает проблема восстановления поверхности в трехмерном пространстве, как и в случае работы с графиком функции двух переменных. Приятным свойством здесь является то, что поверхность множества уровня непрерывна, а минусом – то, что отсутствует какая-либо структура на заданном множестве точек. Фактически просто задан рой точек, являющийся совокупностью части узлов сетки, на которой производились вычисления. В лучшем случае это будет набор узлов параллелепипедальной сетки, в худшем – нерегулярной.

Показ имеющегося роя как такового достаточно бессмысленен, поскольку при таком подходе никак не выявляется трехмерная структура визуализируемого множества. Другой традиционный подход – в случае параллелепипедальной сетки навесить на каждую точку параллелепипед соответствующей ячейки сетки. Однако если сетка не совсем мелкая, то поверхность множества получается слишком изломанной и требует сглаживания. Классическим алгоритмом сглаживания таких «воксельных» поверхностей является Marching Cubes [1]. Но и он не достигает достаточной визуальной гладкости поверхности, особенно если сетка не слишком мелкая и сглаживаемые параллелепипеды достаточно велики визуально. Другим методом сглаживания является серия «лапласовских» алгоритмов [2,3,4,5]. Суть этих алгоритмов заключается в том, что уменьшается кривизна поверхности: выпуклые части втягиваются, вогнутые – выдавливаются. Эти алгоритмы являются итерационными, и регулировкой числа итераций можно добиваться большей или меньшей степени сглаживания.

Авторы в течение последних лет занимались, в частности, формулировкой и теоретическим обоснованием численного алгоритма построения функции цены дифференциальных игр быстродействия с линией жизни [6,7]. Предложенный численный алгоритм как раз имеет сеточный характер и производит приближенные значения функции цены на параллелепипедальной сетке. Первичное оценивание качества вычисле-

ний при этом было связано именно с визуальным сравнением графиков получаемой функции с примерами, посчитанными другими исследователями другими методами. Такое сравнение как раз требует методов визуализации. Визуализация отягащается именно обстоятельствами, указанными выше: разрывностью и несобственностью получаемой функции и в трёхмерном случае относительно крупными ячейками сетки, что приводит к крупности «кирпичиков» и изломанности поверхности. Вследствие этого было принято решение разработать собственные алгоритмы восстановления трехмерного графика разрывной функции двух переменных и исследовать качество сглаживания получаемых воксельных поверхностей алгоритмом Marching Cubes, лапласовскими алгоритмами и их совместным применением.

В случае визуализации множеств уровня функции трех переменных применялась связка Marching Cubes и лапласовского HC-алгоритма [5]. Первичное сглаживание проводилось с помощью Marching Cubes, более тонкая доработка – с помощью лапласовского алгоритма. Однако при эксплуатации такой процедуры восстановления поверхностей множеств уровня выявились некоторые проблемы: из роя точек, симметричного относительно некоторой из координатных плоскостей, восстанавливалась поверхность, не обладавшая соответствующей симметрией. Основной проблемой здесь было то, что Marching Cubes при обработке некоторых конфигураций точек производит четырехугольные элементы формируемой поверхности, которые триангулируются несимметричным образом. Для преодоления указанной проблемы предложена модификация этого алгоритма, сохраняющая симметричность триангулированной поверхности с точностью до отдельных треугольников.

В данной статье описываются предложенные модификации алгоритмов. Приводятся примеры их работы и сравнение получаемых поверхностей с теми, которые получаются классическими версиями алгоритмов. Статья устроена следующим образом. Во втором разделе статьи рассматривается визуализация трехмерного графика функции двух аргументов, описана процедура визуализации графика, корректно восстанавливающая разрывы функции, приведены различные примеры графиков с разрывными функциями. В третьем разделе рассматривается визуализация трехмерных множеств, предложена модификация алгоритма Marching Cubes, сохраняющая симметрию множества, приведены различные примеры трехмерных множеств. Статья завершается заключением и списком литературы.

2. Визуализация трехмерного графика разрывной функции двух аргументов

2.1. Описание процедуры восстановления поверхности

В популярных системах научной визуализации (например, Matlab и GNUPlot) для восстановления трехмерного графика функции двух аргументов применяется, по сути, одна и та же процедура. Ее входной информацией является набор точек трехмерного пространства, заданный на некоторых узлах прямоугольной сетки. При этом в некоторых системах (например, в Matlab) дополнительно требуется, чтобы набор узлов заполнял прямоугольник:

$$\mathcal{L} = \{x_i = x_0 + i \cdot \Delta,\ y_j = y_0 + j \cdot \Delta,\ z_{i,j}\},\ \ i = 0,\ \ldots,\ N,\ \ j = 0,\ \ldots,\ M.$$

Поверхность графика формируется при помощи следующей процедуры.

Каждая прямоугольная ячейка сетки с вершинами в точках (x_i, y_j), (x_{i+1}, y_j), (x_{i+1}, y_{j+1}), (x_i, y_{j+1}) разделяется диагональю на две треугольных ячейки с вершинами в точках

$$\{(x_i, y_j), (x_{i+1}, y_j), (x_{i+1}, y_{j+1})\} \text{ и } \{(x_i, y_j), (x_{i+1}, y_{j+1}), (x_i, y_{j+1})\}.$$

После этого в трехмерном пространстве строятся треугольники с вершинами в точках, соответствующих вершинам полученных ячеек:

$$\{(x_i, y_j, z_{i,j}), (x_{i+1}, y_j, z_{i+1,j}), (x_{i+1}, y_{j+1}, z_{i+1,j+1})\} \text{ и}$$
$$\{(x_i, y_j, z_{i,j}), (x_{i+1}, y_{j+1}, z_{i+1,j+1}), (x_i, y_{j+1}, z_{i,j+1})\}.$$

Недостатки данного алгоритма очевидны. Во-первых, он подразумевает восстанавливаемый график непрерывным. Если имеются резкие изменения значения функции в соседних узлах сетки, то вместо желаемых разрывов графика в таких местах возникают почти вертикальные «стенки». Во-вторых, в рассматриваемых задачах визуализируемая функция могла быть несобственной, принимая бесконечные значения в некоторых областях, содержащих наборы соседних узлов сетки. При этом бесконечные значения приближались некоторой очень большой величиной. Такая структура функции также давала разрывы по линиям, отделяющим области с конечными и бесконечными значениями функции. Кроме того, из-за больших значений возникал слишком большой диапазон, в котором строилась поверхность графика, при этом самые интересные части графика «задавливались», делались мелкими. В-третьих, имеющиеся процедуры подразумевают, что узлы сетки точек предоставляемого набора точек $\mathcal{L}$ заполняют прямоугольник. Вообще говоря, это может быть не так, например, функция вычисляется на круге, то есть набор узлов сетки заполняет не прямоугольник, а круг.

Указанные недостатки привели к необходимости создания собственного алгоритма восстановления поверхности графика:

1. Производится чтение точек. Во время чтения во внутреннее хранилище процедуры не заносятся точки, в которых функция принимает бесконечные значения. Фильтрация ведется по превышению значением функции в точке некоторого порога v_∞, подбираемого визуально или с использованием знаний о вычислительном алгоритме (каким значением подменялась бесконечность).

2. Перебираются точки $(x_{i,j}, y_{i,j}, z_{i,j})$, оставшиеся после фильтрации. Если для текущей точки все три узла (x_{i+1}, y_j), (x_{i+1}, y_{j+1}), (x_i, y_{j+1}) присутствуют в наборе, то так же, как в классической процедуре, формируются две треугольные ячейки. Если из этих трех узлов в наборе присутствую только два, то формируется одна соответствующая ячейка. Если присутствует только один узел или ни одного, ни одной треугольной ячейки не формируется и текущая точка пропускается.

3. Если при обработке точки сформированы одна или две треугольные ячейки, то эти ячейки обрабатываются. Обработка заключается в проверке отсутствия «разрыва» в этой ячейке, то есть проверяется, что значения функции в вершинах ячейки отличаются попарно не более, чем на заданную величину ε. Если это так, то считается, что разрыва нет, и в формируемую поверхность добавляется треугольник с вершинами в соответствующих точках. Если хотя бы одна пара значений отличается больше, чем на порог ε, то констатируется разрыв и ячейка не дает треугольника в формируемую поверхность.

Формируемая поверхность записывается в выходной файл в формате .obj, после чего просматривается в том или ином 3D-просмотрщике, работающем с этим форматом. Авторы использовали бесплатную систему MeshLab. Параметры v_∞ и ε являются входными данными процедуры восстановления и подбираются индивидуально для каждого визуализируемого графика посредством визуального контроля качества получаемой поверхности.

Отметим, что вследствие фильтрации точек, производимой на шаге 1 приведенного алгоритма, область, в которой строится график, может быть существенно непрямоугольной, неодносвзяной или даже вообще несвязной. Однако этот факт не мешает описанной процедуре восстановить и показать требуемый график на области аргументов любой формы.

2.2 Примеры

Пример 2.1. Управляемая система «машина Дубинса». Говорилось, что авторы исследуют численные процедуры, нацеленные на нахождение функции цены дифферен-

циальных игр быстродействия. Однако эти же процедуры могут быть применены и для построения функции оптимального результата в задачах управления. Для этого в описание системы вводится фиктивное управление второго игрока, не входящее в динамику и стесненное одноточечным множеством (для уменьшения перебора возможных управлений).

Указанным образом численно была исследована классическая управляемая система «машина Дубинса» (см., например, [8,9,10,11]), описывающая простейший вариант движения автомобиля (самолета, судна) в горизонтальной плоскости с постоянной величиной линейной скорости при ограничении снизу на возможный радиус разворота объекта. Исходная система имеет трехмерный фазовый вектор (X, Y, φ), где X, Y — координаты положения объекта на плоскости, φ — текущий курсовой угол движения (отсчитываемый от положительного направления оси X против часовой стрелки). Динамика системы описывается соотношениями

$$\dot{X} = \cos\varphi, \dot{Y} = \sin\varphi, \ \dot{\varphi} = u, |u| \le 1. \quad (1)$$

Величина линейной скорости считается равной 1. Ограничение на управление u соответствует минимальному радиусу разворота (максимально возможной угловой скорости поворота вектора линейной скорости). В начальный момент объект считается расположенным в начале координат, движущимся в положительном направлении оси X: $X(0) = 0$, $Y(0) = 0$, $\varphi(0) = 0$. Целью управления является наискорейшее приведение объекта в заданную точку (X^*, Y^*) с любым направлением скорости в момент приведения.

Трехмерная динамика (1) может быть редуцирована к двумерной посредством перехода к новой подвижной системе координат x, y [15,16]. Начало системы совмещается с текущим положением объекта, ось y направляется по текущему направлению линейной скорости, ось x вводится для получения правой системы координат. Тем самым исключается координата φ курсового угла. Динамика системы в новых координатах имеет вид

$$\dot{x} = -yu, \dot{y} = xu - 1, \ |u| \le 1. \quad (2)$$

Здесь координаты x, y имеют смысл текущего относительного положения цели движения объекта. Эта подвижная точка должна быть приведена наискорейшим образом из начального положения $x(0) = Y^*$, $y(0) = -X^*$ в начало координат, то есть совмещена с текущим положением объекта.

К настоящему времени эта задача достаточно хорошо исследована как аналитическими методами [8,12,13], так и с привлечением численных методов [14]. Традиционно функция оптимального результата в этом случае представляется в виде набора линий уровня для различных значений (см. рис. 1). Неудобством такого метода представления является то, что нельзя рассмотреть разрывы функции, а можно лишь догадаться о них по местам сгущения линий уровня.

На рис. 2 показан график функции оптимального результата задачи «машина Дубинса» для ситуации приведения точки в начало координат (в малую окрестность начала координат). Слева показана поверхность, восстановленная с помощью процедуры, встроенной в систему GNUPlot, справа — с помощью процедуры, предложенной авторами. Как было сказано выше, визуализация поверхности графика в этом случае производилась в системе MeshLab. Цветовая раскраска графика просто подчеркивает разные уровни значения функции от нулевого (пурпурный цвет) до имеющегося максимального (красный цвет). Аналогичная раскраска графиков используется и на других рисунках в данном разделе.

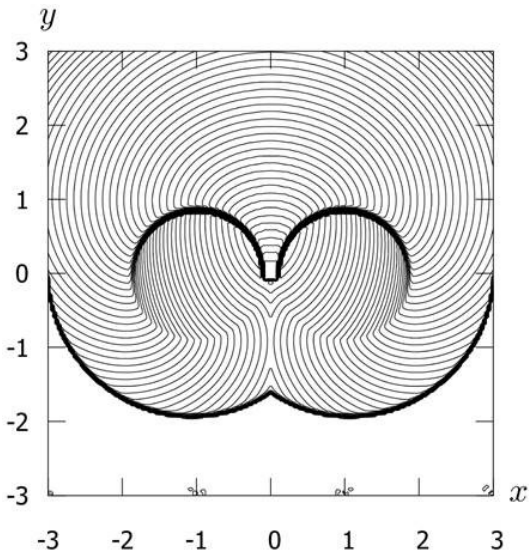

Рис. 1. Одно из традиционных представлений функции оптимального результата в случае двумерного фазового вектора в виде набора линий уровня (для некоторого варианта задачи «машина Дубинса»)

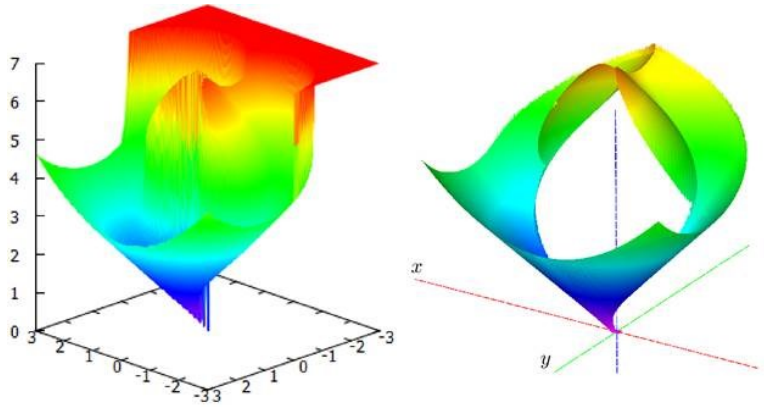

Рис. 2. График функции оптимального результата в задаче «машина Дубинса» с редуцированной динамикой: слева — график функции цены, визуализированный с помощью алгоритмов системы GNUPlot, справа — с помощью системы MeshLab после создания поверхности при помощи авторского алгоритма. Цветовая раскраска графика просто подчеркивает разные уровни значения функции от нулевого (пурпурный цвет) до имеющегося максимального (красный цвет)

Видно отсутствие «плато» в области бесконечных значений функции, которые на левом рисунке были срезаны на уровне 7. Дополнительно не обрабатывался разрыв к бесконечности на зеленой, желтой и оранжевой частях поверхности на правом краю графика, а также разрыв между двумя непрерывными листами поверхности на голубой и зеленой частях поверхности справа в «дырке». Вследствие этого на краях разрыва имеется некоторая «бахрома». Ее причина в том, что линия разрыва достаточно произвольно проходит относительно узлов сетки и значения в «пограничных» узлах соотносятся между собой достаточно произвольно (в смысле, какие из них больше, а какие меньше, и на сколько).

Несмотря на возникающие артефакты, правое изображение более адекватно передает структуру графика, а следовательно, и самой функции, чем левое.

Пример 2.2. Визуализация графика на непрямоугольном множестве. Второй пример также связан с задачей «машина Дубинса» с динамикой (2). Однако теперь график функции оптимального результата вычислялся для узлов сетки, попадающих в круг радиуса 3 с центром в начале координат. График функции оптимального результата показан на рис. 3.

Следует отметить, что GNUPlot справился с задачей визуализации графика на непрямоугольном множестве. Но из-за сложного характера разрыва функции полученное представление совершенно не отражает особенности графика. Поверхность, восстановленная с помощью процедуры, предложенной авторами, значительно лучше отражает характер разрывов функции. На линиях разрыва по-прежнему присутствует небольшая «бахрома».

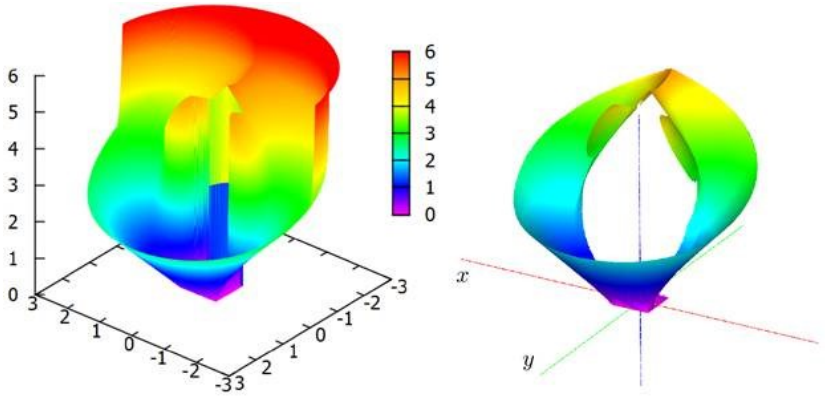

Рис. 3. График функции оптимального результата для некоторого варианта задачи «машина Дубинса», вычисленной на круглом множестве; слева — в системе GNUPlot, справа — в системе MeshLab, поверхность восстановлена авторским алгоритмом. Цветовая раскраска графика просто подчеркивает разные уровни значения функции от нулевого (пурпурный цвет) до имеющегося максимального (красный цвет)

Пример 2.3. Игра «шофер-убийца». Дополнительная обработка графика. На шаге 1 алгоритма, при чтении и фильтрации точек с конечными значениями, или перед ним возможна дополнительная обработка графика: фильтрация по дополнительному критерию, пересчет точек графика (например, масштабирование вдоль оси значений функции) и др. Поскольку файл входных данных представляет собой просто список трехмерных точек (x ,y , значение функции), то фильтрация может производиться (и производилась) исполнением скрипта, который перерабатывает исходный список точек в новый. Средства какой-либо специальной фильтрации (кроме отсечения больших значений) в использованный конвертор не встроены. Дополнительная фильтрация или обработка данных может быть реализована дополнительными скриптами, обрабатывающими файл роя точек до подачи его в процедуру формирования поверхности графика.

Третий пример связан с классической дифференциальной игрой «шофер-убийца», предложенной Р. Айзексом [15,16]. В ней преследователь (автомобиль), имеющий динамику машины Дубинса, преследует убегающего (пешехода), который имеет динамику простых движений, то есть может в каждый момент времени выбрать направление и величину своей скорости:

$$\dot{X_P} = w_P \cos\varphi, \dot{Y_P} = w_P \sin\varphi, \dot{\varphi} = \frac{w_P}{R} u,$$
$$\dot{X_E} = w_E v_X, \dot{Y_E} = w_E v_Y, \qquad (3)$$
$$|u| \le 1, |(v_X, v_Y)| \le 1.$$

Здесь (X_P, Y_P), (X_E, Y_E) — положения преследователя и убегающего на плоскости; φ — курсовой угол, направление линейной скорости преследователя; R — минимальный радиус разворота преследователя; w_P — величина линейной скорости преследователя; w_E — максимальное значение скорости убегающего.

Цель преследователя — наискорейшее приведение убегающего в некоторую заданную свою окрестность, то есть выполнение в как можно ранний момент неравенства $\left|(X_P, Y_P) - (X_{E,} Y_E)\right| \le l$ для заданного l. Убегающий старается избежать поимки или, если это невозможно, отсрочить ее как можно дольше.

Видно, что исходная игра имеет пятимерный фазовый вектор с компонентами X_P, Y_P, φ, X_E, Y_E, однако использованием вышеописанного введения подвижной системы координат может быть приведена к игре с двумерным вектором и динамикой

$$\dot{x} = -\frac{w_P}{R} yu + w_E v_x, |u| \le 1,$$
$$\dot{y} = \frac{w_P}{R} xu - w_P + w_E v_y, \left|(v_x, v_y)\right| \le 1. \quad (4)$$

График функции цены для некоторого варианта этой игры, просчитанный авторами, приведен на рис. 4. На этом рисунке слева приведена визуализация в системе GNUPlot данных, непосредственно полученных из программы. Опять видны стенки на месте разрывов, красное плато в области бесконечных значений. Кроме того, вывод производится в равных масштабах по всем осям, что слишком сплющивает график по оси значений результата. Справа приведена поверхность, восстановленная алгоритмом, предложенным авторами. Кроме того, график дополнительно обрезан по некоторому кругу с центром в начале координат посредством дополнительной фильтрации на первом шаге алгоритма.

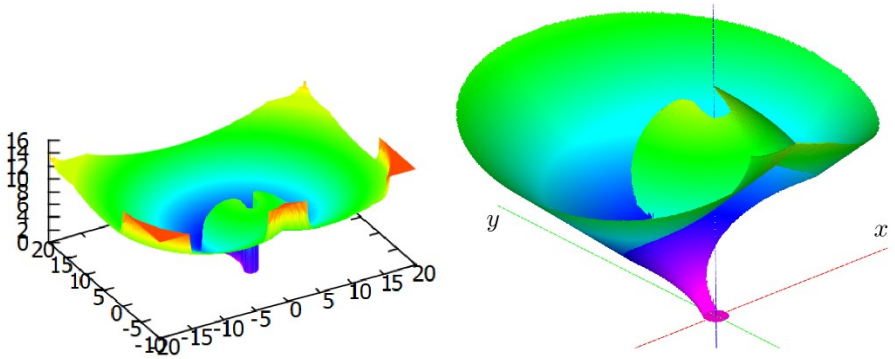

Рис. 4. График функции цены некоторого варианта игры «шофер-убийца»: слева — весь просчитанный график, визуализированный в системе GNUPlot, справа — график, обрезанный по кругу и масштабированный по оси значений функции, восстановленный авторской процедурой и визуализированный в системе MeshLab. Цветовая раскраска графика просто подчеркивает разные уровни значения функции от нулевого (пурпурный цвет) до имеющегося максимального (красный и желто-зеленый цвета на левом и правом подрисунках)

3. Визуализация трехмерных множеств уровня функции трех аргументов

3.1 Описание процедуры восстановления поверхности

Входными данными для данной процедуры является рой точек, полученный фильтрацией всего набора узлов исходной сетки по условию, что значение функции в узле не превосходит заданной величины. Требуется визуализировать множество, «наполняемое» этим роем.

Как говорилось во введении, непосредственный вывод роя не представляет трехмерной структуры множества. Простое навешивание на имеющиеся точки «кирпичиков», равных по размерам ячейке сетки, дает слишком изломанную поверхность.

Традиционным здесь является применение алгоритма Marching Cubes [1], который фактически формирует «кирпичную» поверхность, а потом слегка сглаживает ее, несколько «раздувая» множество. Приятным свойством этого алгоритма является то, что фактически при его работе пропускается этап навешивания «кирпичиков»: он сразу формирует результирующую поверхность в каждой ячейке сетки. Для этого данному алгоритму требуется понимание того, какие вершины очередной ячейки сетки принадлежат визуализируемому множеству, а какие нет. То есть для его работы нужно еще

передать параметры сетки: начальную точку и шаги по осям. Алгоритм организован таким образом, что фрагменты поверхности, построенные в соседних ячейках, автоматически стыкуются друг с другом при любых конфигурациях вершин этих ячеек (см. рис. 5). На рисунке приведены основные конфигурации вершин ячейки. Остальные конфигурации формируются с учетом симметрии. Отметим, что нет конфигураций, описывающих ячейки, у которых пять или более вершин принадлежат визуализируемому рою. В таких случаях ячейка считается «внутренней», слишком окруженной множеством, чтобы в ней появлялись элементы формируемой поверхности.

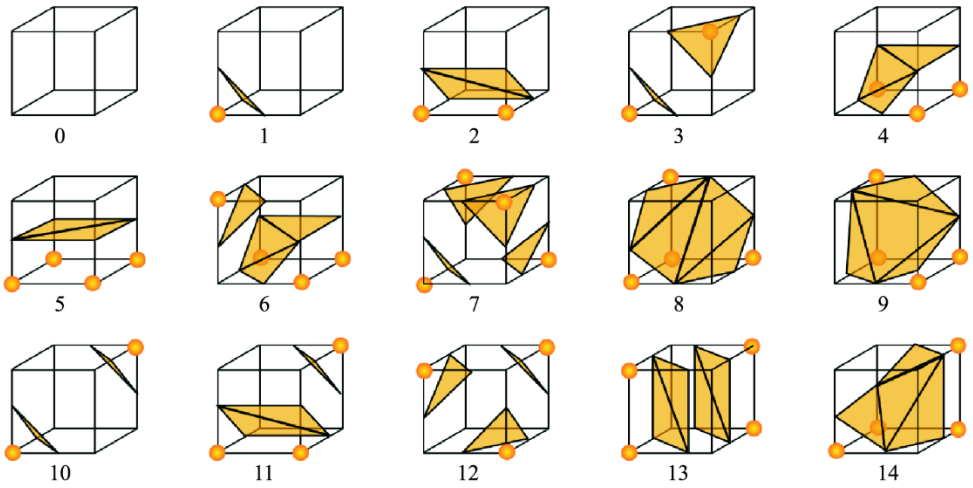

Рис. 5. Основные конфигурации наборов вершин ячейки и элементов достраиваемой поверхности в алгоритме Marching Cubes. Оранжевыми точками отмечены узлы, принадлежащие входному рою (рисунок взят с https://ru.wikipedia.org/wiki/Marching_cubes)

На рис. 6б приведен результат работы алгоритма Marching Cubes при восстановлении поверхности шара, заданного роем точек. Для сравнения на рис. 6а приведено изображение поверхности «воксельного» множества, полученного навешиванием кубика соответствующего размера на каждую точку из заданного роя.

Видно, что из-за малости роя, размер одного кубика относительно велик в сравнении со размером всего множества, поэтому воксельное множество имеет слишком изломанную поверхность. По той же причине результат работы алгоритма Marching Cubes тоже выглядит недостаточно гладким. Заметим, что рассмотрение множеств, задаваемых малым количеством точек тоже важно при исследовании задач управления. Это связано с тем, что эволюция множеств достижимости в моменты времени, близкие к нулю, (эволюция множеств уровня функции цены, соответствующих околонулевым значениям) часто бывает особенно интересна для исследователей.

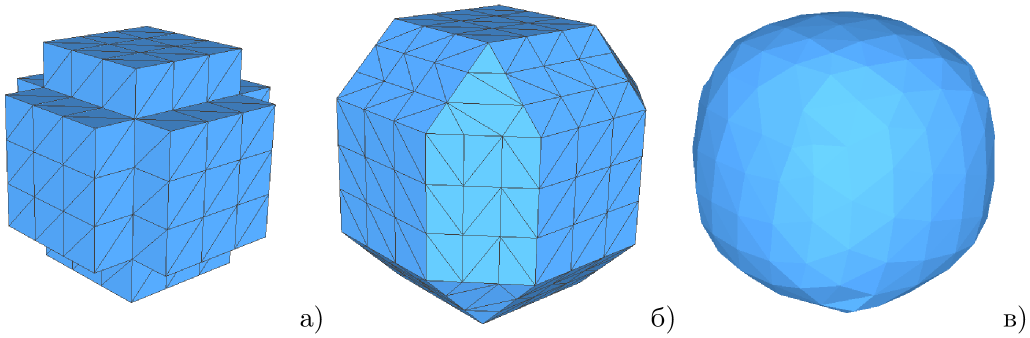

Рис. 6. Визуализация трехмерного шара, заданного роем точек: а) «воксельное» множество; б) результат работы алгоритма Marching Cubes на исходном рое точек; в) результат применения лапласовского алгоритма к поверхности, полученной алгоритмом Marching Cubes

Дальнейшее сглаживание поверхности возможно посредством применения того или иного лапласовского алгоритма [2,3,4,5]. Базовая идея этого семейства алгоритмов заключается в следующем. Для каждой вершины p_i поверхности собирается ее «веер» $\{q_j\}_{j=1}^{N_i}$, то есть совокупность вершин, соседних с p_i в том или ином треугольнике поверхности. После чего вершина p_i подменяется средним арифметическим вершин из веера:

$$p_i' = \frac{1}{N_i}\sum_{j=1}^{N_i} q_j$$

Перемещение всех вершин поверхности производится одновременно. Фактически, при работе этой процедуры поверхность «уплощается»: выпуклые части втягиваются, вогнутые — выдавливаются. Для получения той или иной степени сглаживания алгоритм применяется итеративно несколько раз. Некоторые изменения в вычислительной формуле в тех или иных вариантах связаны с тем, чтобы, в целом, сохранять объем визуализируемого тела. Если такие изменения не вносить, то повторное применение процедуры к выпуклому телу будет его сжимать.

При этом, очевидно, если исходная поверхность сама по себе симметрична относительно какой-либо плоскости, прямой или точки, и при этом так же симметрична ее триангуляция, то результат работы лапласовского алгоритма дает результат, сохраняющий эту симметрию. В противном случае симметрия может нарушаться, что можно наблюдать на рис. 6в. Из-за несимметричности триангуляции поверхности на рис. 6б результат несимметричен. В итоге после применения лапласовского алгоритма результат на рис. 6в ощутимо несимметричен.

В связи с вышеизложенным возникло желание изменить алгоритм Marching Cubes таким образом, чтобы в случае распределения входного роя точек, симметричного относительно тех или иных направлений, связанных с направлениями сетки, триангуляция получаемой поверхности имела бы ту же самую симметрию. Несимметричность триангуляции, производимой Marching Cubes, в первую очередь связана с тем, что в некоторых ситуациях в поверхность реально добавляется элемент, не являющийся треугольником: шаблоны 2, 4, 5, 6, 8, 11, 13 на рис. 5. Однако сам по себе многоугольный элемент не может быть добавлен и должен быть предварительно триангулирован. На рис. 5 видно, что четырехугольники, симметричные в симметричных конфигурациях, разбиваются диагональю на два треугольника, которые уже, вообще говоря, несимметричны. Этот момент и порождает несимметричность триангуляции, которая видна на рис. 6б, где треугольники поверхности выделены тонкими линиями.

Естественным решением было изменить триангуляцию четырехугольных элементов, разбивая каждый их них на четыре треугольника обоими диагоналями (см. рис. 7). При этом симметричные четырехугольники будут триангулированы симметрично.

При этом изменении алгоритма особое внимание следует уделить правильной переработке шаблона 8 и симметричных ему. Остальные отмеченные шаблоны добавляют четырехугольный элемент, который однозначно разбивается на четыре треугольника. Триангуляция шестиугольника, который добавляется в шаблоне 8 и в симметричных ему, уже может быть произведена неоднозначно.

Теперь тот же пример с шаром, что и выше, обрабатывается следующим образом — рис. 8. На рис. 8б видно, что теперь триангуляция поверхности симметрична. Как следствие, результат применения к ней лапласовского алгоритма дает симметричный результат — рис. 8в.

На рис. 9 поверхности с рис. 6в и 8в наложены друг на друга. Камера расположена на одной из координатных осей, и направление зрения параллельно этой оси. Видно, как синяя поверхность, полученная с применением несимметричной версии алгоритма Marching Cubes, выступает за поверхность за поверхность, полученную с применением

симметричной версии. Причем места выступания тоже расположены несимметрично. Симметричность розовой поверхности на этом рисунке видна более ясно, чем на рис. 8.

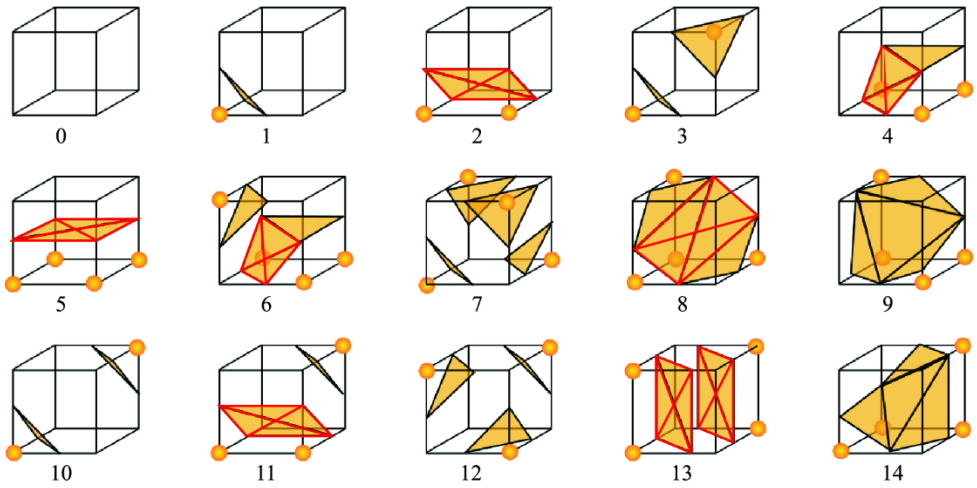

Рис. 7. Исправленные шаблоны алгоритма Marching Cubes

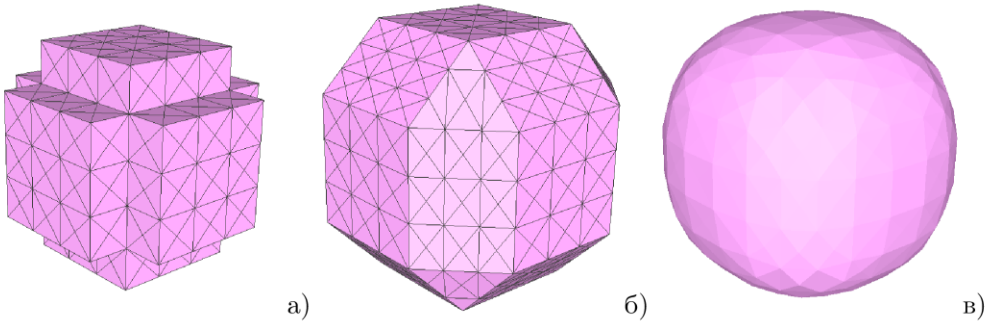

Рис. 8. Визуализация трехмерного шара, заданного роем точек: а) «воксельное» множество; б) результат работы модифицированного алгоритма Marching Cubes на исходном рое точек; в) результат применения лапласовского алгоритма к поверхности, полученной модифицированным алгоритмом Marching Cubes

Рис. 9. Сравнение поверхностей с рис. 6в и 8в

3.2 Примеры

Ниже приведены примеры работы предложенной процедуры восстановления поверхности трехмерных воксельных множеств. Показаны результаты с применением «несимметричного» и «симметричного» вариантов алгоритма Marching Cubes. Так же, как на рис. 6 и 8, результаты работы несимметричной версии алгоритма изображаются голубым цветом, а симметричной — розовым.

Пример 3.1. Машина Дубинса в исходных координатах. В разделе 2 в примерах 2.1 и 2.2 рассматривалась визуализация графика функции оптимального результата для управляемой системы «машина Дубинса» с редуцированной динамикой (2). Однако представляет интерес и изучение задачи в исходном виде с трехмерным фазовым вектором и динамикой (1). Как говорилось, в этом случае судить об устройстве функции оптимального результата можно по набору ее множеств уровня для различных значений. В терминах теории управления такие множества являются множествами достижимости системы к моменту, то есть совокупностью точек фазового пространства, в которых система, стартовав из начального положения или из начального множества, может оказаться не позже заданного момента времени.

Относительно визуализации именно множеств достижимости машины Дубинса встает вопрос относительно угловой координаты φ. По своему смыслу ее значения заключены в диапазоне $[0,2\pi)$ рад (то есть в диапазоне $[0°,360°)$). Однако в этом случае множество имеет весьма странный вид, не очень удобный для анализа. Поэтому считается, что значения координаты φ могут принимать любые значения от $-\infty$ до $+\infty$.

На рис. 10, 11 и 12 приведены множества уровня функции цены для значения 0.2 (множество достижимости из начала координат к моменту 0.2 сек). Множество относительно небольшое, и размер ячеек сетки достаточно велик в сравнении с размерами всего множества, в связи с чем результат применения классического и модифицированного алгоритмов Marching Cubes имеет существенные различия.

На рис. 13 приведены поверхности множества уровня функции цены для большего значения 3π, восстановленные лапласовским сглаживанием результата применения классического и модифицированного вариантов алгоритма Marching Cubes.

Пример 3.2. Игра «изотропные ракеты». В своей книге [15,16] Р. Айзекс рассмотрел игру, сходную с «шофером-убийцей», но в которой преследователь управляет своим ускорением. Убегающий по-прежнему имеет динамику простых движений. Эта игра была названа им «изотропные ракеты». Динамика в исходных координатах описывается соотношениями

$$\begin{gathered} \dot{X_P} = W_X, \dot{W_X} = a\cos\Theta, \dot{X_E} = w_E v_X, a \in [\underline{a}, \overline{a}], \Theta \in [0,2\pi), \\ \dot{Y_P} = W_Y, \dot{W_Y} = a\sin\Theta, \dot{Y_E} = w_E v_Y, |(v_X, v_Y)| \le 1. \end{gathered} \tag{5}$$

Здесь, как и в случае динамики (3), (X_P, Y_P) и (X_E, Y_E) — положения преследователя и убегающего на плоскости. Вектор (W_X, W_Y) — вектор линейной скорости преследователя. Максимальное значение скорости убегающего есть w_E, вектор (v_X, v_Y) — управляющее воздействие убегающего, мгновенное направление его движения. Управление преследователя, ускорение, воздействующее на него, описывается двумя параметрами: a — величина ускорения, принадлежащее заданному интервалу $[\underline{a}, \overline{a}]$; Θ — угол между положительным направлением оси абсцисс и мгновенным направление управляющего ускорения, отсчитываемый против часовой стрелки и выбираемый из диапазона $[0,2\pi)$.

Данная задача имеет фазовый вектор размерности 6. Однако той же заменой координат, которая применялась в примерах в предыдущем разделе (совмещаем начало координат с преследователем, а ось ординат — с направлением мгновенной скорости), можно понизить размерность фазового вектора до 3. Динамика в новых координатах будет иметь следующий вид [15, стр. 244], [16, стр. 302]:

$$\begin{gathered} \dot{x} = -\frac{y}{w_P} \cdot a\sin\theta + w_E v_x, a \in [\underline{a}, \overline{a}], \\ \dot{y} = \frac{x}{w_P} \cdot a\cos\theta + w_E v_y - w_P, \theta \in [0,2\pi), \\ \dot{w_P} = a\cos\theta, |(v_x, v_y)| \le 1. \end{gathered} \tag{6}$$

Здесь (x,y) — разностные координаты преследователь-убегающий, w_P — мгновенное значение скорости преследователя, w_E — максимальное значение скорости убегающего, (v_x, v_y) — вектор мгновенного направления скорости убегающего, a — величина управляющего ускорения преследователя, θ — угол, описывающий направление уско-

рения преследователя. Однако сейчас его удобно отсчитывать от направления скорости преследователя, то есть от оси ординат, и по часовой стрелке, а не против.

В работе [17] рассматривается вариант этой задачи, в которой величина ускорения постоянна $a = \underline{a} = \overline{a} = 1$, а само ускорение может быть направлено только так, чтобы

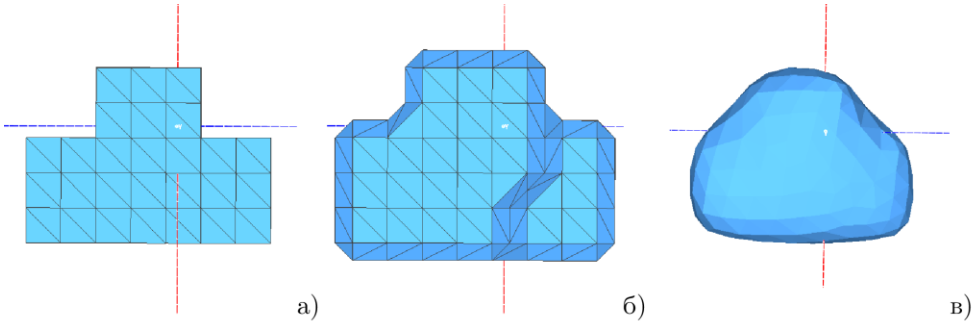

Рис. 10. «Машина Дубинса», множество достижимости к моменту 0.2: а) «воксельное» множество; б) результат работы классического алгоритма Marching Cubes на исходном рое точек; в) результат применения лапласовского алгоритма к поверхности, полученной классическим алгоритмом Marching Cubes

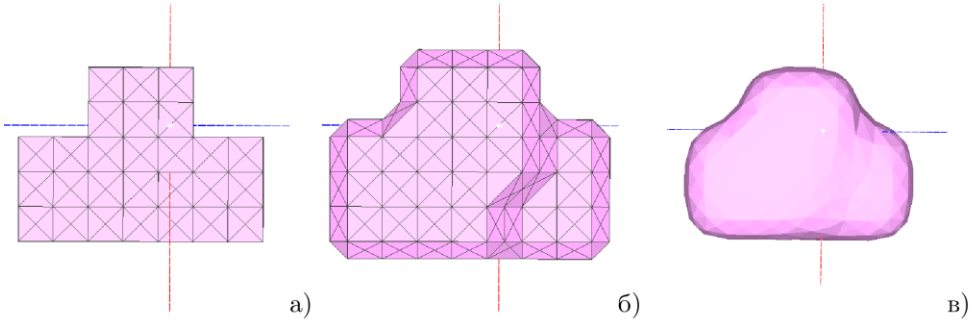

Рис. 11. «Машина Дубинса», множество достижимости к моменту 0.2: а) «воксельное» множество; б) результат работы модифицированного алгоритма Marching Cubes на исходном рое точек; в) результат применения лапласовского алгоритма к поверхности, полученной модифицированным алгоритмом Marching Cubes

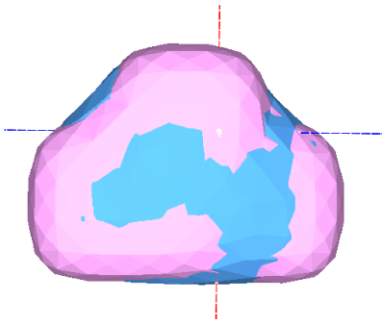

Рис. 12. «Машина Дубинса», сравнение поверхностей с рис. 10в и 11в

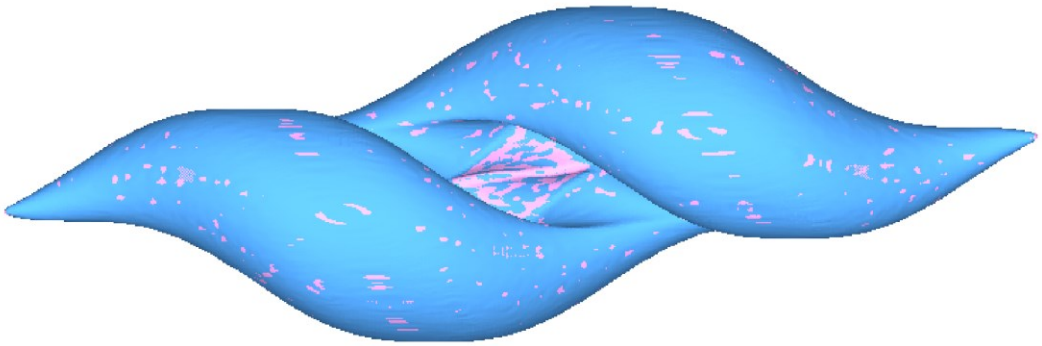

Рис. 13. «Машина Дубинса», множество достижимости к моменту 3π: сравнение поверхностей, полученных с помощью классического алгоритма Marching Cubes (голубая поверхность) и модифицированного (розовая поверхность)

ускорять преследователя, но не замедлять его: $-\pi/2 \le \theta \le \pi/2$. При этом для исключения возникновения слишком большой линейной скорости на ее величину было наложено ограничение $\underline{w} \le w_P \le \overline{w}$. Иными словами, при достижении линейной скоростью ограничения $w = \overline{w}$ допустимыми становятся только управления $\theta = \pm\pi/2$, которые более не влияют на величину линейной скорости, увеличивая ее, а только изменяют направление.

На рис. 14, 15 приведены множества уровня функции цены для значения 5.7. На рис. 14 слева представлена поверхность, полученная с помощью несимметричного алгоритма Marching Cubes, а справа — с помощью модифицированного. Из-за мелкости сетки невооруженным глазом различия не видны. На рис. 15 сравниваются поверхности, полученные с помощью классического (голубые поверхности) и симметричного (розовые поверхности) алгоритмов Marching Cubes.

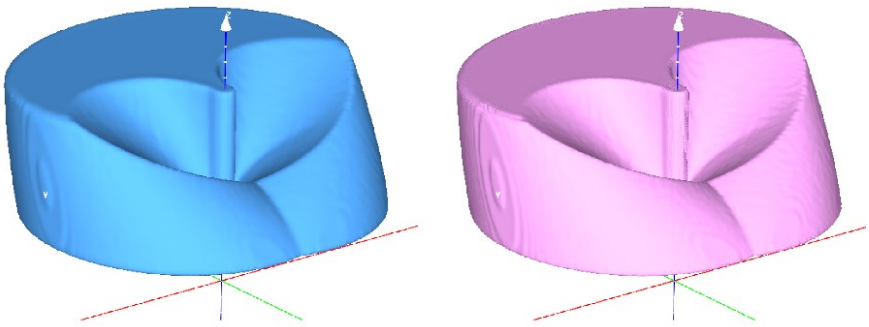

Рис. 14. «Изотропные ракеты», множество достижимости к моменту 5.7: слева — поверхность, полученная при помощи несимметричного алгоритма Marching Cubes, справа применялся симметричный алгоритм

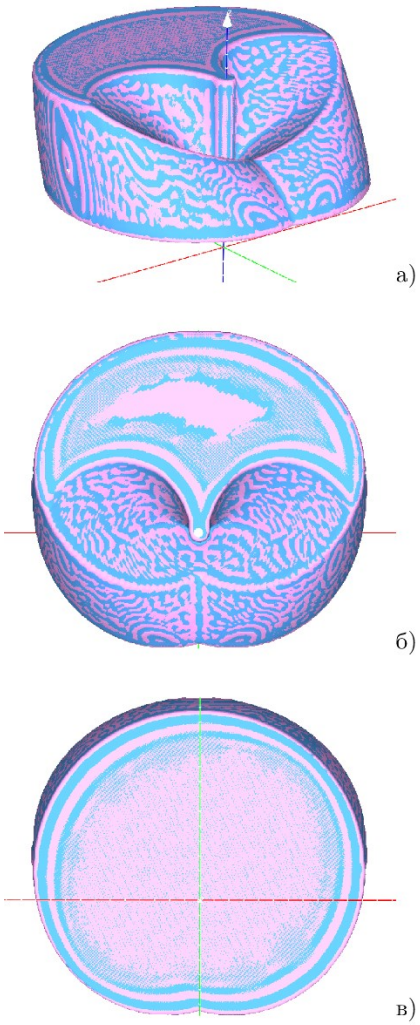

Рис. 15. «Изотропные ракеты», множество достижимости к моменту 5.7: сравнение поверхностей, полученных с помощью классического алгоритма Marching Cubes (голубая поверхность) и модифицированного (розовая поверхность): а) вид сбоку, б) вид сверху (с положительного направления оси Oz), в) вид снизу (с отрицательного направления оси Oz)

4. Программная реализация сглаживания трехмерных множеств

Вычислительная программа была реализована на языке C# для программной платформы .NET Framework 4.8 под операционную систему Miscrosoft Windows. Однако следует отметить, что в течение последних двух лет (после того, как программа была написана) фирма Microsoft выпустила кроссплатформенную версию .NET Core этой среды, а так как программа использует только стандартные возможности языка, то она при необходимости может быть запущена и под другими операционными системами, для которых имеется реализация .NET Core — Linux, MacOS.

Описанные алгоритмы реализованы как набор библиотек и консольных приложений. Консольные приложения принимают на вход файл с данными в формате .txt (набор точек и значений в них), а на выход формируют новый файл в формате .obj (который можно открыть и просмотреть, в частности, с помощью программы MeshLab), содержащий набор точек, треугольников с вершинами в этих точках и векторов нормалей к этим треугольникам.

В целом, алгоритм восстановления поверхности разрывного графика реализован так, как описано в разделе 2.1, реализация НС-алгоритма повторяет описание из ста-

тьи [5]. Интересно отметить некоторые особенности при реализации симметричной версии алгоритма Marching Cubes.

Алгоритм идет по ячейкам сетки и для каждой ячейки просматривает, какие из ее вершин принадлежат визуализируемому рою. Из рис. 5 видно, что вершины добавляемых треугольников могут быть только в следующих точках, координаты которых задаются в локальной системе координат просматриваемой параллелепипедальной ячейки сетки:

```
Point[] vertices = new Point[]
{
  new Point(0.5, 0, 0),
  new Point(1, 0.5, 0),
  new Point(0.5, 1, 0),
  new Point(0, 0.5, 0),
  new Point(0.5, 0, 1),
  new Point(1, 0.5, 1),
  new Point(0.5, 1, 1),
  new Point(0, 0.5, 1),
  new Point(0, 0, 0.5),
  new Point(1, 0, 0.5),
  new Point(1, 1, 0.5),
  new Point(0, 1, 0.5)
};
```

Принадлежность или непринадлежность каждой из восьми вершин визуализируемому рою можно определять, используя байтовую переменную: если вершина принадлежит рою, то в байтовой переменной в соответствующем разряде стоит 1, если не принадлежит, то 0. Таким образом, каждая возможная конфигурация точек роя в вершинах ячейки соответствует какому-то значению от 0 до 255. Это значение используется как индекс в массиве faces, каждый элемент которого также является массивом индексов точек из массива vertices, в которых находятся вершины треугольников, добавляемых при такой конфигурации точек роя. Каждая тройка индексов задает свой треугольник. Размер строки массива обуславливается максимальным количеством треугольников (четыре) в добавляемом элементе поверхности. Последние три элемента в каждой строке вводятся для индикации окончания строки. Индекс -1 означает отсутствие треугольника:

```
int[,] faces = new int[256, 15]
{
  {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
  {0, 8, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
  ..........................
  {9, 5, 7, 9, 7, 2, 9, 2, 0, 2, 7, 11, -1, -1, -1},
  ..........................
  {6, 5, 9, 6, 9, 11, 11, 9, 8, -1, -1, -1, -1, -1, -1},
  ..........................
  {0, 3, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
  {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}
};
```

Последовательность индексов в каждой тройке соответствует обходу соответствующего треугольника против часовой стрелки, если смотреть с конца нормали, внешней к множеству, описываемому заданным роем.

Модифицированная схема (рис. 7) использует аналогичную идею для своей реализации. При этом используется расширенный список возможных вершин добавляемых треугольников:

```
Point[] vertices = new Point[]
{
  new Point(0.5, 0, 0),
  new Point(1, 0.5, 0),
  new Point(0.5, 1, 0),
  new Point(0, 0.5, 0),
  new Point(0.5, 0, 1),
  new Point(1, 0.5, 1),
  new Point(0.5, 1, 1),
  new Point(0, 0.5, 1),
  new Point(0, 0, 0.5),
  new Point(1, 0, 0.5),
  new Point(1, 1, 0.5),
  new Point(0, 1, 0.5),
  new Point(0.5, 0.25, 0.25),
  new Point(0.75, 0.5, 0.25),
  new Point(0.334, 0.667, 0.167),
  new Point(0.5, 0.5, 0.5),
  new Point(0.25, 0.5, 0.25),
  new Point(0.667, 0.667, 0.167),
  new Point(0.5, 0.75, 0.25),
  new Point(0.667, 0.334, 0.167),
  new Point(0.334, 0.334, 0.167),
  new Point(0.25, 0.25, 0.5),
  new Point(0.25, 0.5, 0.5),
  new Point(0.167, 0.667, 0.667),
  new Point(0.334, 0.334, 0.834),
  new Point(0.75, 0.25, 0.5),
  new Point(0.334, 0.167, 0.667),
  new Point(0.834, 0.667, 0.667),
  new Point(0.667, 0.334, 0.834),
  new Point(0.5, 0.25, 0.75),
  new Point(0.667, 0.167, 0.334),
  new Point(0.334, 0.167, 0.334),
  new Point(0.667, 0.834, 0.334),
  new Point(0.334, 0.834, 0.334),
  new Point(0.5, 0.75, 0.75),
  new Point(0.75, 0.75, 0.5),
  new Point(0.834, 0.334, 0.667),
  new Point(0.334, 0.834, 0.667),
  new Point(0.667, 0.667, 0.834),
  new Point(0.667, 0.167, 0.667),
  new Point(0.834, 0.334, 0.667),
  new Point(0.75, 0.5, 0.75),
  new Point(0.834, 0.667, 0.334),
  new Point(0.834, 0.334, 0.334),
  new Point(0.167, 0.334, 0.334),
  new Point(0.167, 0.667, 0.334),
  new Point(0.25, 0.5, 0.75),
  new Point(0.334, 0.667, 0.834),
  new Point(0.667, 0.834, 0.667),
  new Point(0.167, 0.334, 0.667),
  new Point(0.25, 0.75, 0.5)
};
```

Соответствующим образом изменяется массив faces, который теперь имеет размеры 256×24, поскольку максимальное количество треугольников в добавляемом элементе поверхности равно восьми (шаблон 13 на рис. 7).

Заключение

В статье представлены модификации двух классических алгоритмов восстановления поверхностей в трехмерном пространстве. Входными данными для алгоритмов является вещественнозначная функция, посчитанная на прямоугольной/параллелепипедальной сетке, заполняющей, возможно, множество произвольной формы (не обязательно прямоугольник или параллелепипед).

Первый алгоритм связан с восстановлением поверхности графика разрывной функции двух аргументов. Другой особенностью визуализируемой функции является ее несобственность, то есть возможность принимать бесконечные значения. При вычислениях бесконечные значения подменяются какой-то весьма большой величиной. Алгоритмы, представленные в популярных научных системах — MathLab, MathCAD, GNUPlot и др., восстанавливают поверхность графика, просто соединяя треугольниками имеющиеся точки в узлах сетки. Это приводит к возникновению вертикальных «стенок» в местах разрыва функции, а также к появлению «плато» на больших значениях, что деформирует график и делает слаборазличимой ту его часть, именно которая интересна исследователю. Для адекватного отображения разрывов в предлагаемом алгоритме в формируемую поверхность не добавляется треугольный элемент, если разница аппликат хотя бы двух его вершин по модулю больше заданного порога. Для удаления высоких «плато» алгоритм выбрасывает из набора те узлы, в которых значение функции больше заданной величины. В силу такой фильтрации алгоритм приспособлен к ситуации, когда форма множества, на котором производится визуализация графика, может быть существенно непрямоугольной, в том числе, из-за того, что счет изначально велся на непрямоугольном множестве.

Если размеры ячеек сетки не слишком малы, по краям разрывов и переходов к плато может образовываться «бахрома». Линии разрыва проходят относительно узлов сетки достаточно произвольным образом, где-то ближе, где-то дальше, так что значения в узлах могут отличаться друг от друга, а соответствующие точки располагаться на разной высоте. Возможно, требуется дополнительное сглаживание края поверхности, хотя при достаточно мелких шагах сетки эта бахрома не слишком велика.

Второй алгоритм связан с визуализацией множеств уровня функции трех аргументов, то есть областей трехмерного пространства, в которых функция не превосходит заданной величины. Прямолинейный способ визуализации такого рода объектов связан с формированием «воксельных» множеств, когда на каждый имеющийся узел навешивается прямоугольный «кирпичик» соответствующей ячейки сетки. Однако при этом поверхность множества получается чересчур изломанной, особенно если размер ячеек не слишком мал в сравнении с размером самого множества. Традиционно сглаживание такой поверхности производится алгоритмом Marching Cubes, алгоритмами лапласовского семейства или их совместным применением. Однако классический алгоритм Marching Cubes формирует несимметричную триангуляцию поверхности, даже если исходное множество имело симметрию относительно координатных плоскостей, осей или какой-либо точки. Поэтому при дальнейшей обработке такой поверхности несимметричность может возрастать и становиться явной. Предложена модификация алгоритма Marching Cubes, производящая триангуляцию поверхности, сохраняющую указанные симметрии.

Впрочем, возникающие нарушения симметрии имеют размер ячейки сетки и не видны, если размер ячейки сетки мал в сравнении с размерами множества. Предлагаемая модификация является полезной в первую очередь при визуализации небольших множеств.

Список литературы

1. W.E. Lorensen, H.E. Cline, Marching Cubes: A high resolution 3D surface construction algorithm // ACM SIGGRAPH Computer Graphics, Vol. 21, No. 4, 1987, pp. 163–169.

2. S. Canann, M. Stephenson, T. Blacker, Optismoothing: An optimization-driven approach to mesh smoothing // Finite Elements in Analysis and Design, Vol. 13, No. 2–3, 1993, pp. 185–190.

3. L. Freitag, On combining Laplacian and optimization based mesh smoothing techniques // Proceedings of the 1997 Joint Summer Meeting of American Society of Mechanical Engineers (ASME) American Society of Civil Engineers (ASCE) and Society of Engineers Science (SES), 1997, pp. 37–44.

4. N. Amenta, M. Bern, D. Eppstein, Optimal point placement for mesh smoothing // Journal of Algorithms, Vol. 30, No. 2, 1999, pp. 302–322.

5. J. Vollmer, R. Mencl, H. Muller, Improved Laplacian Smoothing of Noisy Surface Meshes // Computer Graphics Forum, Vol. 18, No. 3, 1999, pp. 131–138.

6. Н.В. Мунц, С.С. Кумков, Численный метод решения дифференциальных игр быстродействия с линией жизни // Математическая теория игр и ее приложения, Т. 10, No 3, 2018, С. 48–75.

7. N.V. Munts, S.S. Kumkov, Convergence of Numerical Method for Time-Optimal Differential Games with Lifeline // Annals of the International Society of Dynamic Games, Vol. 17, Games of Conflict, Evolutionary Games, Economic Games, and Games Involving Common Interest. Basel: Birkhäuser, 2020. pp. 103–132.

8. L.E. Dubins, On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents // American Journal of Mathematics, Vol. 79, No. 3, 1957, pp. 497–516.

9. J.-P. Laumond (ed.), Robot Motion Planning and Control, Lecture Notes in Control and Information Sciences, Vol. 229. Springer-Verlag, Berlin Heidelberg, 1998. 354 p.

10. Y. Meyer, T. Shima, P. Isaiah, On Dubins paths to intercept a moving target // Automatica, Vol. 59, 2015, pp. 256–263.

11. E. Bakolas, P. Tsiotras, Optimal synthesis of the asymmetric sinistral/dextral Markov-Dubins problem // Journal of Optimization Theory and Applications, Vol. 150, No. 2, 2011, pp. 233–250.

12. E.J. Cockayne, G.W.C. Hall, Plane Motion of a Particle Subject to Curvature Constraints // SIAM Journal on Control, Vol. 13, 1975, pp. 197–220.

13. Ю.И. Бердышев, Нелинейные задачи последовательного управления и их приложение. Екатеринбург: ИММ УрО РАН, 2015. 193 с.

14. A.A. Fedotov, V.S. Patsko, Investigation of Reachable Set at Instant for the Dubins' Car // Proceedings of the 58th Israel Annual Conference on Aerospace Sciences, Tel-Aviv & Haifa, Israel, March 14–15, 2018, ThL2T1.4, pp. 1655–1669.

15. R. Isaacs, Differential Games. John Wiley and Sons, New York, 1965. 384 p.

16. Р. Айзекс, Дифференциальные игры. М.: Мир, 1967. 479 с.

17. N. Botkin, K.-H. Hoffmann, N. Mayer, V. Turova, Computation of value functions in nonlinear differential games with state constraints // D. Hömberg, F. Tröltzsch (eds.), System Modeling and Optimization. CSMO 2011. IFIP Advances in Information and Communication Technology, Vol. 391, 2013, pp. 235–244.

Scientific Visualization, 2024, volume 16, number 1, pages 1 - 18, DOI: 10.26583/sv.16.1.01

Modifications of Classical Surface Reconstruction Algorithms for Visualization of a Function Defined on a Rectangular Grid

N.V. Munts[1], S.S. Kumkov[2]

N.N. Krasovskii Institute of Mathematics and Mechanics of the Ural Branch of the Russian Academy of Sciences (IMM UB RAS), Yekaterinburg, Russia

[1] ORCID: 0000-0003-3234-1267, natalymunts@gmail.com
[2] ORCID: 0000-0002-2690-5380, sskumk@gmail.com

Abstract

In the paper, modifications of visualization algorithms for real-valued functions of two and three arguments given on a rectangular or parallelepipedal grid are considered. In the case of two arguments, the graph of the function is a surface embedded into the three-dimensional space. The majority of scientific visualization systems offer visualization procedures for such surfaces, but they construct them under the assumption that the functions are continuous. In the paper, for the case of a discontinuous function, a modification of this algorithm is proposed. In addition, the algorithm removes “plateaus” that occur after cutting the function at some level (in order to remove too large values).

Visualization of a function of three arguments implies showing its level sets, that is, regions of the space of arguments where the magnitudes of the function do not exceed a certain value. In the case of a grid function, such sets are “voxel” sets, that is, they are composed of grid cells. With that, some smoothing of the surface of such sets is required, which is carried out by the Marching Cubes algorithm and algorithms of the Laplacian family. A modification of the Marching Cubes algorithm is proposed, which preserves the symmetry of the set surface with respect to the coordinate planes, axes, or some point, if the rendered set has such a symmetry.

Keywords: human state visualization, human factor, pie charts, human resources management.

References

1. W.E. Lorensen, H.E. Cline, Marching Cubes: A high resolution 3D surface construction algorithm // ACM SIGGRAPH Computer Graphics, Vol. 21, No. 4, 1987, pp. 163–169.

2. S. Canann, M. Stephenson, T. Blacker, Optismoothing: An optimization-driven approach to mesh smoothing // Finite Elements in Analysis and Design, Vol. 13, No. 2–3, 1993, pp. 185–190.

3. L. Freitag, On combining Laplacian and optimization based mesh smoothing techniques // Proceedings of the 1997 Joint Summer Meeting of American Society of Mechanical Engineers (ASME) American Society of Civil Engineers (ASCE) and Society of Engineers Science (SES), 1997, pp. 37–44.

4. N. Amenta, M. Bern, D. Eppstein, Optimal point placement for mesh smoothing // Journal of Algorithms, Vol. 30, No. 2, 1999, pp. 302–322.

5. J. Vollmer, R. Mencl, H. Muller, Improved Laplacian Smoothing of Noisy Surface Meshes // Computer Graphics Forum, Vol. 18, No. 3, 1999, pp. 131–138.

6. N.V. Munts, S.S. Kumkov, Chislennyj metod resheniya differencial’nyh igr bystrodejstviya s liniej zhizni // Matematicheskaya teoriya igr i ee prilozheniya, V. 10, № 3, 2018, pp. 48–75. (in Russian)

7. N.V. Munts, S.S. Kumkov, Convergence of Numerical Method for Time-Optimal Differential Games with Lifeline // Annals of the International Society of Dynamic Games, Vol. 17, Games of Conflict, Evolutionary Games, Economic Games, and Games Involving Common Interest. Basel: Birkhäuser, 2020. pp. 103–132.

8. L.E. Dubins, On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents // American Journal of Mathematics, Vol. 79, No. 3, 1957, pp. 497–516.

9. J.-P. Laumond (ed.), Robot Motion Planning and Control, Lecture Notes in Control and Information Sciences, Vol. 229. Springer-Verlag, Berlin Heidelberg, 1998. 354 p.

10. Y. Meyer, T. Shima, P. Isaiah, On Dubins paths to intercept a moving target // Automatica, Vol. 59, 2015, pp. 256–263.

11. E. Bakolas, P. Tsiotras, Optimal synthesis of the asymmetric sinistral/dextral Markov-Dubins problem // Journal of Optimization Theory and Applications, Vol. 150, No. 2, 2011, pp. 233–250.

12. E.J. Cockayne, G.W.C. Hall, Plane Motion of a Particle Subject to Curvature Constraints // SIAM Journal on Control, Vol. 13, 1975, pp. 197–220.

13. Y.I. Berdyshev, Nelinejnye zadachi posledovatel'nogo upravleniya i ih prilozhenie [Nonlinear problems of sequential control and their application]. Ekaterinburg: IMM UrO RAN, 2015. 193 p. (in Russian)

14. A.A. Fedotov, V.S. Patsko, Investigation of Reachable Set at Instant for the Dubins' Car // Proceedings of the 58th Israel Annual Conference on Aerospace Sciences, Tel-Aviv & Haifa, Israel, March 14–15, 2018, ThL2T1.4, pp. 1655–1669.

15. R. Isaacs, Differential Games. John Wiley and Sons, New York, 1965. 384 p.

16. R. Ajzeks, Differencial'nye igry [Differential games]. M.: Mir, 1967. 479 p. (in Russian)

17. N. Botkin, K.-H. Hoffmann, N. Mayer, V. Turova, Computation of value functions in nonlinear differential games with state constraints // D. Hömberg, F. Tröltzsch (eds.), System Modeling and Optimization. CSMO 2011. IFIP Advances in Information and Communication Technology, Vol. 391, 2013, pp. 235–244.