Использование объёмного метода для восстановления 3D поверхностей

А.Г. Позин
Государственный Научно-Исследовательский Институт Авиационных Систем,
Москва, Россия
pozinan@mail.ru

Аннотация

Статья описывает процесс создания единой 3D модели из набора выровненных фрагментов. Использование объёмного алгоритма делает эту задачу интуитивно понятной и простой в реализации. Также описывается структура данных уменьшающая требования к памяти необходимой при построении больших моделей.

Ключевые слова: *3D сканирование, восстановление поверхностей.*

1. ВВЕДЕНИЕ

Часто под решением задачи 3D сканирования объекта понимается получение единой односвязной модели состоящей из треугольников, возможно с текстурой. Методы получения отдельных фрагментов 3D объекта (отдельных сканов), а также приведение их в одну систему координат хорошо проработаны и описаны [1,2,3]. Однако получение единой законченной модели из набора фрагментов до сих пор не является тривиальной. Для решения этой проблемы были разработаны различные алгоритмы, которые можно разбить на несколько характерных групп:

- Алгоритмы, работающие с беспорядочным набором 3D точек. [4,5,6]. На вход подаются только трёхмерные координаты точек в пространстве. Это делает алгоритмы этого типа практически универсальными, а иногда и единственно применимыми. Однако отказ от использования дополнительной информации о точках может стоить качества восстанавливаемой модели.
- Алгоритмы, выбирающие локально лучший фрагмент на каждом участке формируемой 3D модели. [7] Эти алгоритмы хорошо ведут себя на “хороших” данных, когда ошибки сшивки фрагментов практически отсутствуют.
- Алгоритмы, дискретно представляющие пространство и решающие задачу на полученной сетке. [8] Исходные данные преобразуются в дискретный вид, происходит их обработка и затем результат преобразуется назад из дискретного вида в исходный формат.

2. АЛГОРИТМ

В данной статье описывается алгоритм, реализующий дискретный подход.

Алгоритм использует неявную скалярную функцию D(x) заданную на регулярной 3D решётке. После основного шага алгоритма, из решётки, методом марширующих кубиков, строится полигональное представление изоповерхности D(x)=0, которая и является результатом работы алгоритма.

Функция D(x) представляет собой средневзвешенное функций $d_i(x)$ соответствующих отдельным фрагментам с соответствующими им весовыми функциями $w_i(x)$.

Каждое значение функции $d_i(x)$ в точке x это расстояние от x до фрагмента i вдоль соответствующего направления сканирования. Расстояние считается со знаком, то есть если с одной стороны фрагмента расстояния положительные, то с другой они отрицательные. На рисунке 3 расстояния разных знаков показаны разными цветами. Усреднение происходит по обычной формуле:

$$D(x) = \frac{\sum w_i(x) \bullet d_i(x)}{\sum w_i(x)}$$

На рисунке 1 показано примерное поведение результирующей функции полученной из двух после усреднения.

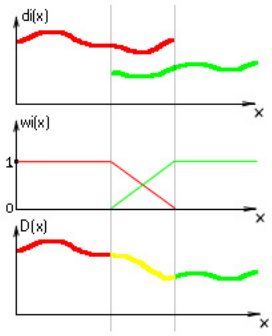

Рисунок 1. Сверху – исходные фрагменты. В центре – их весовые функции. Внизу – результат сложения с весами.

Простой алгоритм формирования $d_i(x)$ может выглядеть так:

- Для каждого элемента 3D решётки (вокселя) x:
- Спроецировать центр x на фрагмент вдоль направления сканирования. Определить треугольник пересечения ABC и вычислить точку P принадлежащую ему – проекцию x на фрагмент.
- Вычислить расстояние от P до x, принять его за значение $d_i(x)$ для вокселя x.
- Зная веса w_A, w_B, w_C точек A,B,C линейной интерполяцией найти вес точки P – w_P. Принять w_P за $w_i(x)$ для вокселя x.

На практике нет смысла перебирать все вокселы решётки, достаточно ограничиться только лежащими на небольшом расстоянии **r** от фрагмента.

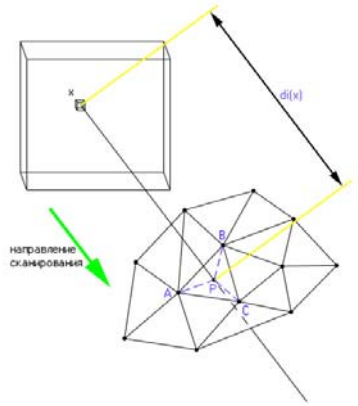

Рисунок 2. Вычисление $d_i(x)$

Задача вычисления весов в каждой вершине исходной модели может решаться по-разному. На практике в качестве веса был принят параметр удаления точки от границы фрагмента приведённый к интервалу [0, 1]. Даже при использовании такого простого подхода результаты получаются очень хорошими.

Вычисление D(x) происходит по указанной выше формуле, причём для её построение не требуется одновременного присутствия всех $d_i(x)$ и $w_i(x)$ в памяти, процесс построения D(x) может быть инкрементальным. Достаточно вместо окончательных значений хранить для каждого вокселя значения a(x) и b(x):

$$a(x) = \sum_i w_i(x) \bullet d_i(x) \quad \text{и} \quad b(x) = \sum_i w_i(x),$$

тогда D(x) может быть вычислена как

$$D(x) = a(x) / b(x).$$

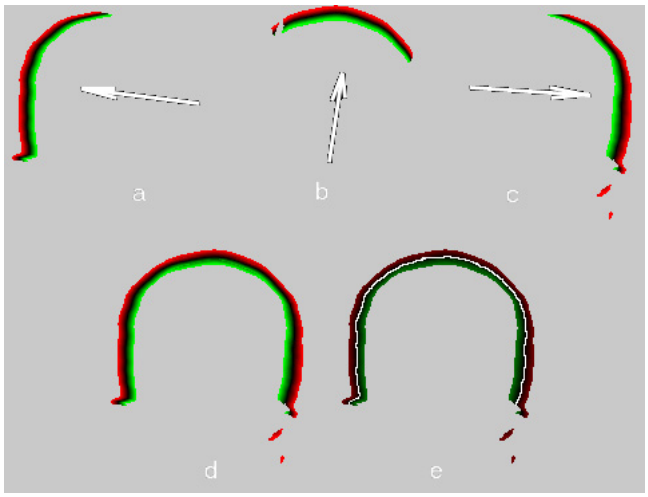

Рисунок 3. (a,b,c) -три объединяемых фрагмента и направления их сканирования. (d) – D(x) - неявная функция объединения трёх фрагментов и, (e) - результат выделения изоповерхности.

На рисунке 3 показан процесс формирования D(x) для трёх фрагментов. Интенсивностью показано удаление от фрагмента, разными цветами показаны противоположные стороны поверхности

Одним из параметров построения $d_i(x)$ является “толщина слоя” – параметр, задающий, на каком расстоянии от поверхности еще стоит формировать $d_i(x)$. При выборе толщины слоя приходится принимать компромиссное решение. С одной стороны, его не стоит делать слишком маленьким, так как объединяемые фрагменты могут расходиться достаточно сильно, и в этом случае вместо одной единой модели мы получим модель, состоящую из двух отдельных фрагментов. С другой стороны, если взять толщину слоя слишком большой, то распространяющаяся $d_i(x)$ может дойти до фрагмента, к которому она не имеет никакого отношения и испортить результат (рисунок 4.)

В качестве опорного значения при выборе толщины слоя можно взять величину расхождения отдельных фрагментов друг от друга, или величину среднеквадратической ошибки возникающей при сшивке.

Рисунок 4. Катастрофический результат при выборе чрезмерно большой толщины слоя.

3. АЛГОРИТМ МАРШИРУЮЩИХ КУБИКОВ

Этот алгоритм предназначен для генерации полигонального приближения изоповерхности некоторой скалярной функции F(x) заданной на равномерной 3D сетке значений [9]. Получаемая поверхность является аппроксимацией изоповерхности некоторого уровня a, то есть геометрического места точек удовлетворяющих условию F(x) = a. Для простоты изложения будем считать, что ищется изоповерхность уровня 0. Для приведения к этому случаю общего, достаточно из всех значений в сетке вычесть требуемый уровень a.

Алгоритм работает следующим образом:

- Для каждой ячейки сетки выбираются 8 скалярных значений лежащих в её вершинах. Таким образом, образуется куб со значениями в вершинах.
- Если все эти значения положительны или все отрицательны, значит, искомая поверхность не проходит внутри этого куба и он пропускается.
- Если же часть значений положительны, а часть отрицательны, значит, поверхность пересекает некоторые рёбра куба. На рёбрах, опирающихся на вершины со значениями противоположных знаков, намечаются точки пересечения, координаты которых затем вычисляются линейной интерполяцией в соответствии со значениями в вершинах образующих ребро.
- На образующихся точках пересечения, для каждого кубика, строится некоторый набор треугольников задаваемый таблицей. Для каждой вершины

возможны два состояния – больше 0 и меньше нуля, сам 0, не нарушая логики, можно считать, например, положительным. Так как всего вершин в кубе 8, получается таблица размером $2^8 = 256$, каждый элемент которой содержит список треугольников для данного случая.

Алгоритм прост в реализации, но, несмотря на это, даёт хорошие результаты. В сложных случаях возможно построение поверхности имеющей разрывы, для устранения этого недостатка разработан специальный вариант алгоритма учитывающий все случаи и строящий доказано непрерывную поверхность [10], однако в случае работы с достаточно гладкими поверхностями, при не очень больших размерах кубиков эти проблемы не проявляются.

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Для эффективного применения алгоритма марширующих кубиков к задаче построения сложной 3D модели требуется создание специального программного обеспечения позволяющего работать с большими объёмами данных. Пусть, например, есть поверхность шаровидной формы с размерами 400x400x400 мм, требуется построить её полную 3D модель с разрешением не ниже 0.5 мм. Соответственно, для применения описанного алгоритма необходимо создать массивы a(x), b(x) с размером 800x800x800 каждый и динамически их обновлять. Если хранить числа в формате float (4 байта), то для хранения этих массивов потребуется 2*800*800*800*4 = 3.8 ГБ. Работать с такими объёмами данных в операционной системе с ядром Win32 напрямую просто невозможно из-за ограничения адресного пространства под программу пользователя в 2ГБ. Таким образом, возникает жизненная необходимость сократить размер требуемой памяти хотя бы до размера 1-1.5ГБ. Справиться с этой задачей позволяют специальные структуры данных. В работе [8] было предложено хранить данные в памяти в упакованном формате RLE (run length encoding), что, в среднем, позволяло сэкономить размер используемой памяти примерно на порядок. Однако при использовании формата RLE возникает дополнительное требование к алгоритму – он должен работать с данными последовательно, строка за строкой. Это вызвано тем, что для работы с кусочком данных следует сначала его раскодировать, а сразу после обработки закодировать. Процесс же кодирования/декодирования необходимо вести последовательно, от самого начала данных. Это ограничение делает такой подход неудобным в использовании и не позволяет реализовывать наиболее эффективные алгоритмы, использующие произвольный доступ к данным.

В качестве альтернативы было принято решение для эффективной структуры хранения 3D данных использовать октадерево (octree). Общий принцип построения дерева заключается в рекурсивном делении пространства на 8 частей одинакового размера и в дальнейшем хранение информации только о непустых образованных частях. Процесс деления завершается по достижении какого-то минимального размера образованной части (листа дерева). Вид дерева с размером листа в 4 элемента в 2D случае (quadtree) показан на рисунке 5. Использование дерева позволило значительно уменьшить требования к памяти, в среднем на порядок, при возможности произвольного обращения к данным

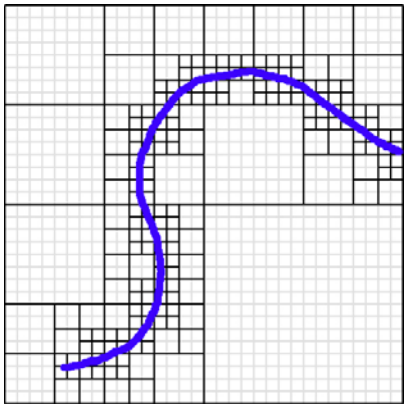

Рисунок 5. Представление пространства вокселей в виде 2D дерева с размером листа 2x2 вокселя.

Другой проблемой требующей разрешения является быстрое определение точки пересечения луча исходящего из центра вокселя с фрагментом. Промежуточная задача – поиск треугольника пересекающегося с этим лучом. На практике была выбрана такая последовательность действий для каждого фрагмента:

- Спроецировать все треугольники фрагмента на плоскость перпендикулярную направлению сканирования. При этом любой луч параллельный направлению сканирования при проецировании на эту плоскость превращается в точку.
- Для поиска треугольника на плоскости, содержащего точку, использовать любой быстрый алгоритм решающий эту задачу. Например, поиск с использованием дерева, последовательный поиск вдоль прямой на плоскости или использование кэширующего массива.

В качестве простого, но эффективного решения задачи поиска треугольника содержащего точку был выбран кэширующий массив. При этом вначале быстро определяется элемент массива, в который попала точка и затем эта точка проверяется на пересечение с каждым треугольником, ассоциированным с этим элементом массива.

5. РЕЗУЛЬТАТЫ

На рисунках 6,7 приведены результаты работы алгоритма. Все эксперименты проводились на машине P4-1700, 512МБ.

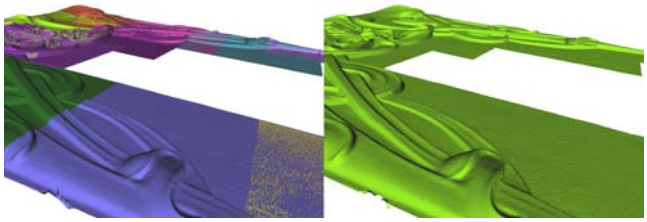

Рисунок 6.

Рисунок 6 - “Доска” – часть рельефной гипсовой плиты размером 900x700 мм, 12 фрагментов, всего 2 млн. треугольников.

Рисунок 7 – “Манекен” – резиновая модель головы, размер 177x296x230 мм , 32 фрагмента, всего 1 млн. треугольников.

Модель	Шаг построения	Толщина слоя	Время построения	Конечное кол-во треугольников
Доска	0.5 мм	3 мм	457 сек	5 419 473
Доска	1 мм	5 мм	84 сек	1 334 256
Доска	2 мм	5 мм	19 сек	328 510
Манекен	0.5 мм	3 мм	190 сек	1 642 637
Манекен	1 мм	5 мм	43 сек	408 122
Манекен	2 мм	5 мм	11.5 сек	100 776

Количественные результаты работы алгоритма.

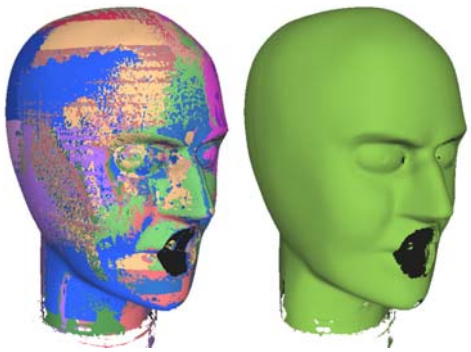

Рисунок 7.

5.1 Дальнейшее использование результатов

Полученные модели можно подвергать дополнительным преобразованиям, в зависимости от назначения, например, упрощать. В Microsoft DirectX SDK 9.0 входят функции по упрощению моделей методом Progressive Mesh [11]. При этом количество треугольников или точек в упрощённой модели можно задавать заранее.

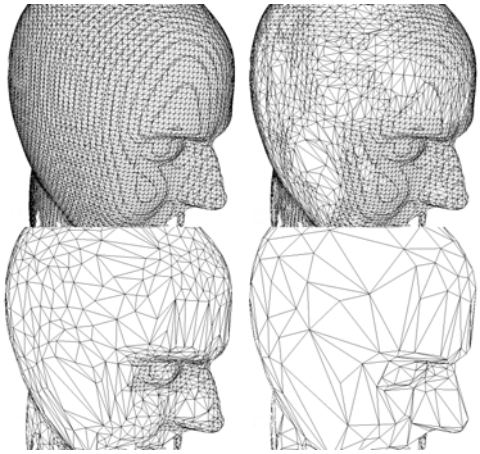

Рисунок 8. Исходная модель (44639 треугольника) и упрощённые – 25000, 5000 и 1000 треугольников.

5.2 Дальнейшее развитие технологии

Полученный алгоритм можно дорабатывать в качественном и количественном направлении. Например, его можно модернизировать для работы с текстурированными фрагментами и получения конечной модели визуально лучше соответствующей сканируемому объекту. Для повышения скорости и уменьшения требований к памяти можно разбивать исходные данные на части, решать задачу локально и затем объединять результаты в конечную модель. Такой подход мог бы практически устранить проблему чрезмерного использования памяти. Также интересной является идея построения более плотной модели на основе только что созданной более грубой, последовательно доводя качество модели до необходимого уровня.

Отдельной проблемой является задача восстановления модели. Например, устранение дыр во фрагментах вызванных недостаточной областью видимости или технологическими проблемами (такими как блики, прозрачные участки и т.д.). Описанный алгоритм позволяет решать и эту проблему, однако требует значительной модификации.[12]

6. БИБЛИОГРАФИЯ

[1]Paul J. Besl and Neil D. McKay “A Method for Registration of 3-D Shapes.”

[2] Natasha Gelfand, Leslie Ikemoto, Szymon Rusinkiewicz, Marc Levoy “Geometrically Stable Sampling for the ICP Algorithm.”

[3] Kari Pulli “Multiview Registration for Large Data Sets.”

[4] N.Amenta, M.Bern., and M.Kamvysselis. A new Voronoi-based surface reconstruction algorithm.

[5] N.Amenta, S.Choi and R.Kolluri. The power crust, unions of balls and the medial axis transform.

[6] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Computer Graphics (*SIGGRAPH *’92 Proceedings)*, volume 26, pages 71–78, July 1992.

[7] G. Turk and M. Levoy. Zippered polygon meshes from range images. In Proceedings *of SIGGRAPH ’94 (Orlando, FL, July 24-29, 1994)*, pages 311–318. ACM Press, July 1994.

[8] Brian Curless and Marc Levoy, A Volumetric Method for Building Complex Models from Range Images, Stanford University

[9] W.E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction lgorithm. In *Computer Graphics (SIGGRAPH ’87 Proceedings)*, volume 21, pages 163–169, July 1987.

[10] C. Montani, R. Scateni, and R. Scopigno. A modified look-up table for implicit disambiguation of marching cubes. *Visual Computer*, 10(6): 353–355, 1994.

[11] Hugues Hoppe, Microsoft Research, Progressive Meshes

[12] James Davis Stephen R. Marschner Matt Garr Marc Levoy, Filling Holes in Complex Surfaces using Volumetric Diffusion. Appears in Proc. First International Symposium on 3D Data Processing, Visualization, Transmission

Об авторе.

Позин Андрей Григорьевич, закончил Московский Физико-Технический Институт. Аспирант ГосНИИАС, инженер.
Почтовый адрес pozinan@mail.ru

Using volumetric method for 3D surface reconstruction

Abstract

This paper describes process of creating single union 3D model from the number of previously aligned scans. Using the volumetric algorithm make this task intuitive clear and easy for implementation. In addition, it describes the data structure reducing memory requirement needed for processing large models.

Keywords: 3D scanning, surface reconstructing.

About the author

Pozin Andrey Grigorievich is an engineer of GosNIIAS, Ph.D. student. His contact email is pozinan@mail.ru