Дискретизация функционально заданных поверхностей

Вяткин С.И.
Институт автоматики и электрометрии, СО РАН, Новосибирск, Россия
sivser@mail.ru

Abstract

***Vyatkin S.I. Functionally-defined surface discretization**. Surface discretization (triangulation) is a well-known topic of computational geometry. A triangulation of a set of points is a way of connecting them to form a simple triangle. The existing algorithms of triangulation differ in terms of their computational complexity, the frequency of the function evaluation (sampling), and the resulting mesh quality. High efficiency of a triangulation algorithm can be achieved by the use of optimization techniques, which can reduce the complexity of the basic algorithm. A new approach for accurate triangulation of functionally defined surfaces using perturbation functions is proposed. For analyzing how close the evolving meshes approaches the functionally defined surface two error metrics and uniformity are used. The metrics measure deviations of the vertices from the functionally defined surface and deviations of mesh normal from the normal of the functionally defined surface.*

Введение

На сегодняшний день разработано много разных функциональных способов задания геометрических 3-х мерных моделей: F-rep [1], неявные (implicit) поверхности (Blobby-модели [2, 3], метасферы [4], поверхности свертки [5-7] и т.д.). Эти способы отличаются друг от друга заданием ядра – базовой функции и операциями, заданными над базовыми функциями. Функционально заданные геометрические объекты с применением функций возмущения изложены в работе [8].

Для разных способов задания функциональных форм используются разные алгоритмы дискретизации их поверхности (полигонизации или триангуляции), которые основаны на возможностях, предоставляемых конкретным способом функционального задания. Эти алгоритмы триангуляции можно разделить на несколько основных типов.

1. Использование частиц для контроля неявных поверхностей. Алгоритм основан на взаимодействии частиц с функциональной поверхностью [9]. Этот алгоритм подходит только для неявных (implicit) поверхностей.
2. Расширяющееся построение на основе ранее построенных граней. Этот алгоритм основан на вычислении положения соседних граней для уже построенных граней [10]. Для этого необходима затравочная грань, вокруг которой строятся остальные грани. Для реализации данного алгоритма необходимо вычислять градиент функции, что в нашем задании делать проблематично.
3. Алгоритмы, разработанные для триангуляции облаков точек, полученных после сканирования поверхности реальных объектов (например, скульптур) [11]. В этих алгоритмах необходимо производить поиск по всему пространству, что сильно расходует память и вычислительные ресурсы. В таких алгоритмах одно из положительных важнейших свойств - условие равномерности сетки, может стать невыполнимым.

Основными требованиями качественной триангуляции в нашем случае являются:

- Разработка алгоритма, использующего преимущества задания функциональных объектов на базе функций возмущения, для нахождения точек поверхности.
- Триангуляция функциональных 3-х мерных объектов с жесткими ребрами.
- Равномерность 3-х мерной сетки в любой подобласти объекта. Треугольная сетка должна иметь треугольники без резких скачков по площади и длинам сторон, то есть быть однородной.
- Анализ степени приближения полученной сетки к функционально заданной поверхности по двум метрикам:
  - отклонение вершин треугольной сетки от истинной поверхности функционального объекта;
  - отклонение нормалей к вершинам треугольной сетки от истинных значений нормалей к вершинам на функциональной поверхности.

Существующие способы [1-7] задания функциональных поверхностей обладают тем

недостатком при триангуляции, что для них нельзя быстро найти точку на поверхности объекта. Для того чтобы найти точку на такой поверхности, необходимо пустить луч, и двигаясь вдоль луча с заданным дискретным шагом, сравнивать знак функции со знаком, полученным на предыдущем шаге. Как только знак меняется, мы находим точку на поверхности с заданной точностью дискретизации луча. Такой способ нахождения точки для всех предыдущих способов функционального задания получил название step-by-step, то есть пошаговое вычисление, которое производит много лишних вычислений и не работает за постоянное время. Таким образом, для функциональных вышеперечисленных поверхностей необходимо сканировать все пространство, то есть проверять каждое подделение дискретизации луча, и точка поверхности не может быть построена за постоянное время.

В способе задания на основе функций возмущения для нахождения точки на поверхности используется алгоритм многоуровневого отслеживания лучей, который основан на отсекании из рассмотрения тех областей пространства, где не находится объект. Алгоритм имеет древовидную структуру поиска, позволяющую быстро находить за постоянное число шагов точку на поверхности [8]. В способе задания объекта на основе функций возмущения пустые области пространства сразу отбрасываются из рассмотрения, благодаря чему мы можем найти точку на поверхности за постоянное число шагов.

В данной работе предложен общий подход к триангуляции функционального объекта, обеспечивающий равномерность сетки объекта в любой ее подобласти. Разработан и реализован алгоритм преобразования трехмерных объектов, заданных функционально на базе функций возмущения, в стандартное представление в виде списков вершин и треугольных граней (сетки) с возможностью триангуляции объектов с жесткими ребрами.

Триангуляция поверхностей, заданных функциями возмущения

Для триангуляции функциональной поверхности используется адаптированный алгоритм многоуровневого отслеживания лучей [8].

Вычисление состоит из двух этапов: деление пространства (рис. 1) по четвертичному дереву (вычисление лучей, пересекающих объект) (рис. 2), затем для вычисленного луча осуществляется бинарный поиск (рис. 3). При делении по четвертичному дереву происходит деление бруска на четыре части и определение расположения бруска относительно объекта. Те бруски, для которых в тесте на пересечение определено, что они находятся заведомо снаружи объекта, - отбрасываются. При достижении максимального уровня деления по четвертичному дереву из каждой ячейки проекции объекта пускается луч (брусок) вдоль оси z. Этот луч делится посередине на два луча, для каждой из половин проводится тест на пересечение. Вначале рассматривается брусок с меньшим значением координаты z. Если он пересекается с объектом или находится внутри объекта, брусок с большим z отбрасывается, так как в этом случае он перекрыт для камеры. Если же брусок с меньшим z заведомо лежит снаружи объекта, то этот брусок отбрасывается и рассматривается брусок с большим z. Затем для оставшегося бруска повторяется деление пополам по координате z. Из всех точек пересечения луча с поверхностью, алгоритм бинарного деления луча выбирает ту, которая является ближайшей к камере, и определяет ее z координату, равную координате бруска на последнем уровне подделения. Остальные точки пересечения луча с поверхностью отбрасываются, так как они перекрыты для камеры ближайшей точкой пересечения луча с поверхностью и не видны (см. рис. 2). Таким образом, алгоритм деления пространства по четвертичному дереву определяет x, y координаты точки поверхности, то есть проекцию на плоскость камеры, а алгоритм бинарного деления луча – z координату.

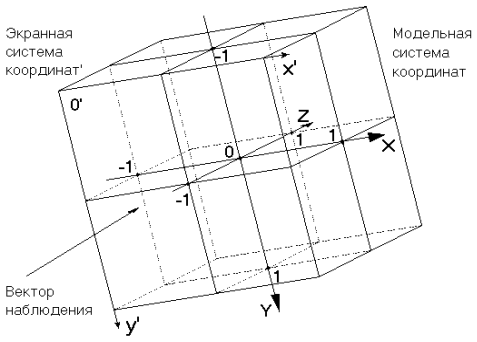

Рисунок 1 – Объектная система координат

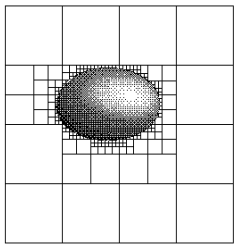

Рисунок 2 – Экранная система координат

Для триангуляции функционального объекта необходимо построить облако точек, которые принадлежат поверхности объекта (рис.

4). Алгоритм нахождения точек поверхности работает в два этапа:

1. построение проекции объекта на плоскость камеры. Проекция строится при помощи обхода пространства по четверичному дереву, также как это реализовано в алгоритме многоуровневого отслеживания лучей. В результате получается набор ячеек – пикселей проекции;

2. трассировка луча через каждую ячейку проекции и нахождение всех его точек пересечения с поверхностью. Через каждую ячейку проекции трассируется в направлении z луч, и бинарным делением пополам находятся все точки пересечения луча с поверхностью.

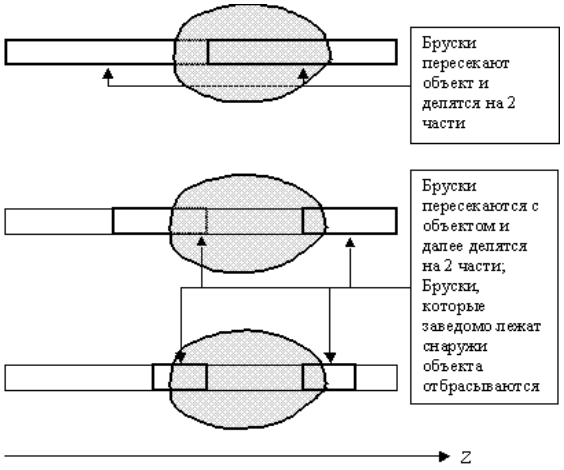

Рисунок 3 – Бинарное деление луча для нахождения точек поверхности

В отличие от алгоритма многоуровневого отслеживания лучей, который используется для отображения поверхности и находит ближайшую точку пересечения луча с поверхностью, для построения точек поверхности необходимо найти все точки пересечения луча с поверхностью (рис. 3). При делении пространства по четверичному и двоичному деревьям образуются ячейки пространства - *вокселы*, размер которых зависит от максимального уровня подделения по четверичному дереву и от максимального уровня подделения по двоичному дереву. Пусть длина пространственного куба равна 1. Тогда если максимальный уровень подделения по четверичному дереву равен QLEV, то размер ячейки-воксела по оси x/y равен $1/2^{QLEV}$. При максимальном уровне подделения луча по бинарному дереву равному BLEV, размер ячейки-воксела по оси z равен $1/2^{BLEV}$.

Вычисленные точки поверхности лежат в ячейках-вокселах, на которые было разбито пространство делением сначала по четверичному дереву, а затем по двоичному. Рассмотрим луч и все вокселы пространства на луче, образованные при бинарном делении при построении точек поверхности. Часть вокселов пустые, которые не принадлежат объекту. Часть точек – вокселы, где находятся точки поверхности. Часть точек – вокселы, которые находятся внутри объекта.

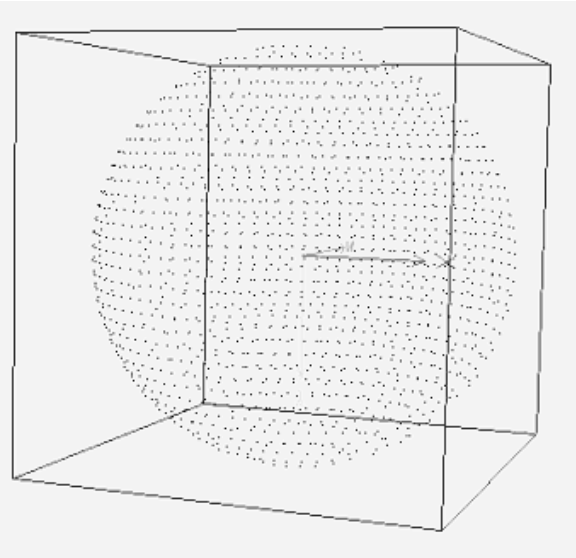

Рисунок 4 – Облако точек

Непустые вокселы, имеющие одну координату x, образуют вертикальное сечение объекта по координате x. Непустые вокселы, имеющие одну координату y, образуют горизонтальное сечение объекта по координате y. То есть каждая точка поверхности лежит в вертикальном сечении объекта и в горизонтальном сечении объекта, образованном из непустых вокселов. Контуры вертикальных и горизонтальных сечений объекта связывают соседние точки поверхности между собой. Для нахождения соседних точек для данной точки по горизонтали/вертикали, нужно:

1. построить горизонтальное/вертикальное сечение, в котором находится заданная точка;

2. построить контур границы сечения.

Двигаясь по контуру сечения нужно выбрать соседние точки для данной точки.

Рисунок 5 – Точка B и ее соседи в вертикальном сечении эллипсоида – точки A и C

На рисунке 5 A, B, C – точки, принадлежащие поверхности функционального объекта. Вокселы, находящиеся внутри объекта, образуют наполнение сечения.

Для нахождения контура сечения используется следующий алгоритм:

- На каждом шаге известен вектор движения от предпоследней точки в сечении до последней (текущей) точки.
- Относительно текущей точки и вектора рассматриваются точки по часовой стрелке: справа от вектора и точки, спереди вектора и точки, слева от вектора и точки, сзади от вектора и точки (рис. 6).
- Первая точка, которая является не пустой, выбирается текущей. Относительно нее и предпоследней точки строится новый вектор.

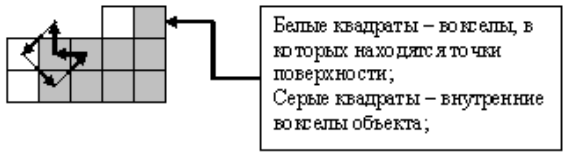

Рисунок 6 – Обход вокселей при поиске следующего направления движения

Движение по контуру напоминает движение колеса по поверхности, которое вращается против часовой стрелки или по часовой, в зависимости от порядка рассмотрения соседей: справа, спереди, слева, сзади или слева, спереди, справа, сзади соответственно. Если текущий воксел является не только не пустым, но в нем находится точка поверхности, то она является ближайшим соседом для предыдущей точки поверхности в контуре при движении по контуру сечения. Для замыкания точек поверхности в четырехугольные полигоны мы от заданной точки поверхности A двигаемся в вертикальном сечении вниз, затем от полученной точки B – в горизонтальном сечении по контуру вправо. Затем от полученной точки C – в вертикальном сечении по контуру вверх, от полученной точки D в горизонтальном сечении по контуру – влево. Если замыкание произошло на начальную точку A, то построение четырехугольного полигона ABCD закончено, состоящего из двух треугольных граней ABC и ACD, аппроксимирующих поверхность функционального объекта (рис. 7). Так как не всегда мы возвращаемся в начальную точку при обходе вниз-вправо-вверх-влево, то в некоторых местах сетки объекта получаются дыры (рис. 8). То есть, сетка, которая получается после связывания точек поверхности в четырехугольные полигоны, имеет дырки. Рассмотрим ребро грани в сетке и все грани, в которые входит это ребро. Если количество связанных с ребром граней равно единице, то это ребро является ребром на границе дырки.

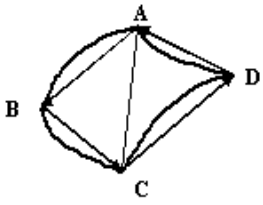

Рисунок 7 – Четырехугольный полигон A->B->C->D, аппроксимирующий поверхность функционального объекта; A, B, C, D – точки поверхности функционально заданного объекта

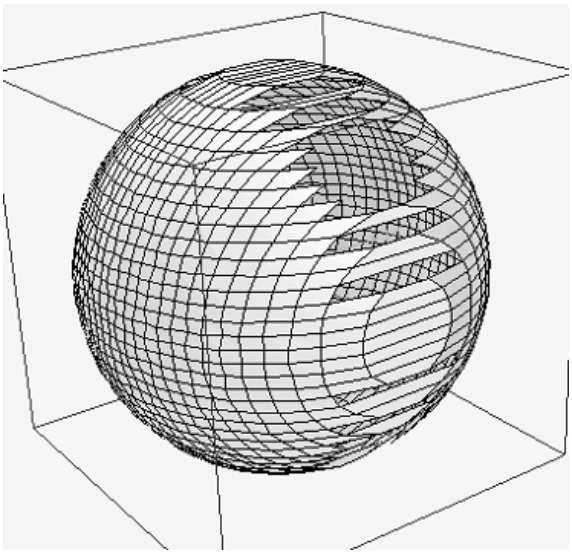

Рисунок 8 – Объединение точек поверхности в четырехугольные полигоны; после объединения видны дырки в сетке

Если количество связанных с ребром граней равно двум, то в этом случае, ребро является той частью треугольной сетки, в которой нет дырок. Поэтому, чтобы найти и запомнить все дырки, мы должны искать ребра, каждое их которых связано только с одной гранью.

Алгоритм поиска дырки работает следующим образом: 1) определить начальное ребро дырки (p_1, p_2); 2) среди всех ребер, которые исходят из p_2, найти то ребро (p_2, p_3), где p_3 не равно p_1, которое принадлежит только одной грани, то есть является ребром на границе дырки; 3) для точки p_n повторить поиск ребра (p_n, p_{n+1}), где p_{n+1} не равно p_{n-1}, которое принадлежит только одной грани, то есть является ребром на границе дырки; 4) поиск дырки закончен, как только мы возвращаемся в начальную точку p_1. Таким образом, находятся все дырки в сетке, полученной после замыкания точек в четырехугольные полигоны.

Для триангуляции дырок используется модифицированный алгоритм Лиепы [12], для построения затяжек с минимальной площадью затяжки или с минимальной суммой ребер

затяжки с временем работы алгоритма $O(n^3)$. Пусть граница дырки задается последовательным набором вершин $V=\{v_0, v_1, \ldots, v_{n-1}\}$, где $P=(v_0, v_1, \ldots, v_{n-1})$ – полигон, с вершинами $v_0, v_1, \ldots, v_{n-1}$, затягивающий дырку. Для каждой треугольной грани зададим весовую функцию W. В качестве W можно выбрать либо значение площади треугольной грани, либо сумму длин ее ребер. Для $0<=i<=j<n$ и некоторой триангуляции подполигона $(v_i,\ldots, v_j)$ определим вес триангуляции как сумму весов всех треугольных граней, образующих подполигон $(v_i,\ldots, v_j)$. Определим $W_{i,j}$ как минимальный вес триангуляции подполигона $(v_i,\ldots, v_j)$. Алгоритм нахождения затяжки с минимальным весом для полигона $P=(v_0, v_1, \ldots, v_n)$ работает следующим образом [12]: 1) для i=0,1,…., n-2 положим функцию веса для двух соседних номеров вершин $W_{i, i+1}=0$; для i=0, 1,…, n-3 положим функцию веса для грани равную площади грани $W_{i, i+2}$=Площадь треугольной грани (i, i+1, i+2), положим j=2; 2) j=j+1. Для i=0,1, …. n-j-1 и k=i+j найдем минимальный вес затяжки полигона $W_{i,k}$=min по $i<m<k$ [$W_{i,m}$ +$W_{m,k}$+Площадь треугольной грани (v_i,v_m,v_k)]. Индекс m, при котором достигается минимум, запоминается как $M_{i,k}$; 3) если $j<n-1$, то возвращаемся на шаг 2, иначе вес минимальной по весу триангуляции равен Wo,n-1; 4) восстановление граней, образующих триангуляцию с минимальным весом, по значениям $M_{i,k}$.

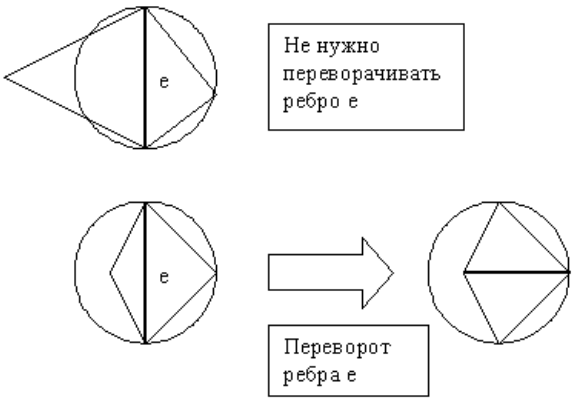

Рисунок 9 – Восстановление ребра

Алгоритм удобно раскладывается в рекурсию, когда на каждом шаге рекурсии для вычисления минимальной триангуляции подполигона $(v_i,\ldots,v_k)$ мы погружаемся вглубь для вычисления промежуточных значений веса и для нахождения минимизирующего индекса m. В качестве веса для треугольной грани можно кроме значения площади грани также взять сумму длин ее ребер. В этом случае триангуляция дырки с таким весом для грани имеет минимальную сумму ребер всех граней, входящих в триангуляцию дырки. Полученные затяжки дырок в сетке по алгоритму, описанному выше, обладают следующим недостатком. Хотя алгоритм обеспечивает минимальность площади затяжки, но треугольные грани, образующие затяжку, не соответствуют по густоте треугольным граням частей сетки без дырок. Для улучшения затяжки мы можем добавить в нее дополнительно точки и перетриангулировать ее, чтобы затяжка была похожа по густоте граней на части сетки без дырок. Основная идея алгоритма добавления вершин состоит в том, чтобы вычислить длину ребер для вершин на границе дырки и затем разбить грани, образующие затяжку, чтобы их длина ребер соответствовала средней длине ребер на границе дырки с последующим *восстановлением* ребер. *Восстановить* ребро грани означает проверить, что для двух соседних граней по этому ребру, их вершины, не принадлежащие ребру, находятся снаружи описанной сферы для противоположной грани. Если это условие не выполняется, ребро переворачивается (рис. 9).

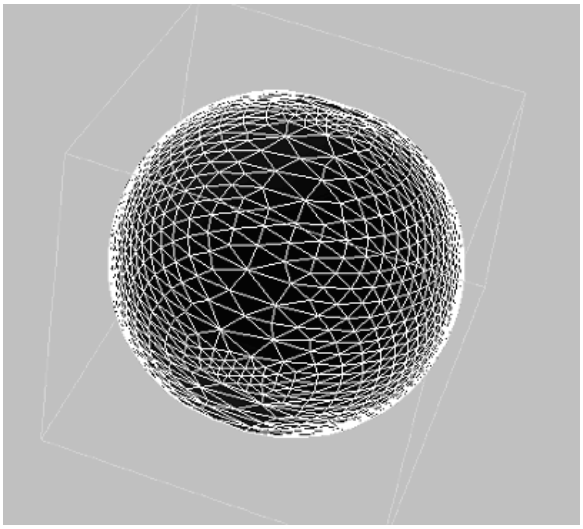

Рисунок 10 – Добавленные вершины и полученные затяжки

Алгоритм добавления вершин в затяжку работает следующим образом: 1)для каждой вершины на границе дырки вычисляется средняя длина ребер, исходящих из нее и не принадлежащих затяжке; 2) для каждой грани (v_i, v_j, v_k) в затяжке вычисляется положение точки-центроида v_c (сумма координат вершин, деленная на три, то есть точка пересечения медиан грани). Центроиду приписывается средняя длина ребер равная $l(v_c)=(l(v_i)+l(v_j)+l(v_k))/3$. Если для m=i,j,k выполняется условие: $\| v_c - v_m \|>l(v_c)$ и $\| v_c - v_m \|>l(v_m)$, то грань разбивается на три меньших грани (v_c, v_i, v_j), (v_c, v_i, v_k), (v_c, v_j, v_k). После этого происходит восстановление ребер (v_i, v_j), (v_i, v_k), (v_j, v_k); 3) если на предыдущем шаге не было разбито на три части ни одной грани, то добавление вершин завершено; 4) восстановление всех внутренних ребер затяжки; 5) если ни одно

ребро не было перевернуто, вернуться к шагу 2. Иначе повторить шаг 4.

В результате работы вышеописанного алгоритма мы имеем сетку треугольников без дырок (рис. 10).

Перед добавлением вершин, затяжки объединяются с соседними к ним гранями сетки, которые проецируются на плоскость камеры с нулевой площадью проекции. Так как эти грани образуют стороны объекта перпендикулярные к плоскости камеры, то такие грани могут быть вытянутыми для объектов кубической формы. Поэтому для таких граней тоже применяется разбиение по алгоритму добавления вершин в затяжку.

Кроме этого, ребро может быть определено термином “жесткое ребро”, когда оно принадлежит двум граням с большим углом между их нормалями (более 45 градусов), восстановление ребра после вставки в грань центроида (центр масс, пересечение медиан) может привести к изменению формы (рис. 11). Поэтому для “жесткого ребра” PQ (рис. 12) вместо восстановления применяется его деление: 1) критерий деления: если s(P) – желаемая средняя длина ребер в точке P, s(Q) – желаемая средняя длина ребер в точке Q. Тогда длины ребер, которые необходимы, вычисляются следующим образом: если sqrt(2.0)*||PQ|| > [s(P)+s(Q)]/2, тогда ребро слишком длинное, и требуется его деление; 2) если ребро должно быть поделено по первому критерию- 1, точка S, после деления ребра имеет координаты S=(s(P)*Q+s(Q)*P) / (s(P)+s(Q)), другими словами можно сказать, что точка S делит ребро по центру только тогда, когда s(P)=s(Q). Если s(P) не равно s(Q), точка S делит ребро PQ пропорционально весам; 3) после деления ребра каждая грань делится на две грани

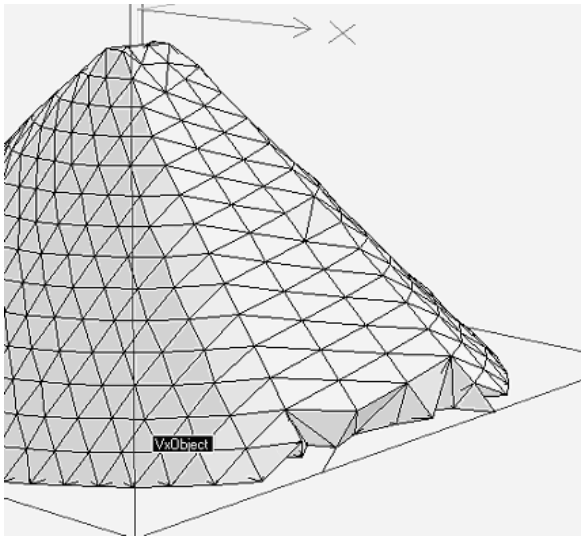

Рисунок 11 – Сетка треугольников с артефактами

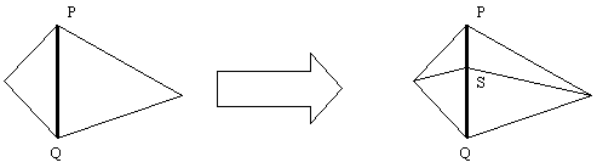

Рисунок 12 – Деление “жесткого ребра”

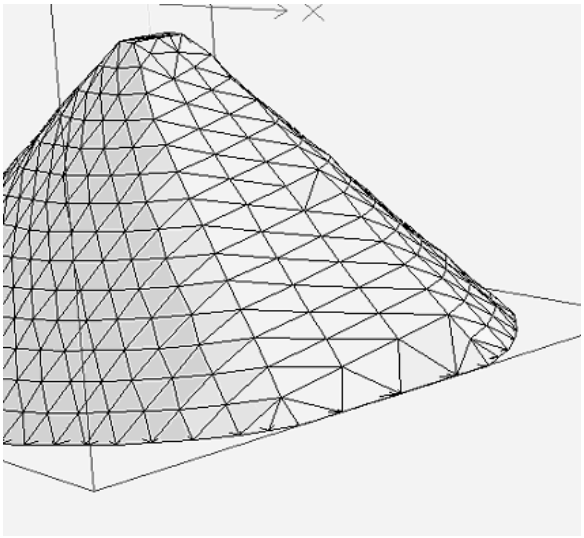

Рисунок 13 – Сетка треугольников без артефактов

Коррекция кривизны поверхности

Для каждой точки p затяжки строится оператор U^1

$$U^1(p) = -p + \frac{1}{w(p)} \sum_i w(p,i)\, p(p,i)$$

$$w(p) = \sum_i w(p,i)$$

где w(p,i) - вес ребра между точкой p и i;
p(p, i) – i-ый сосед точки p. Суммирование идет по всем точкам i, с которыми точка p соединена ребром. Затем на основе U^1 для каждой точки p строится оператор U^2

$$U^2(p) = -U^1(p) + \frac{1}{w(p)} \sum_i w(p,i) U^1(p,i)$$

где $U^1(p,i)$ – оператор U^1 для i-го соседа точки p.
На плоскости U^1 равно 0 для любой точки p. На кривой уже U^1 не равно 0, но $U^2=0$. Так как на кривой поверхности среднее отклонение точки от ее среднего по соседям одинаково как для точки, так и для ее соседей. Поэтому решается уравнение относительно $U^2(p)=0$ – находится новое положение точки p.

Если вес w(p, i)=1 для любого ребра (то есть все ребра имеют примерно одинаковую длину), то

$$U^1(p) = -p + \frac{1}{n(p)} \sum_i p(p,i)$$

где n(p) – количество точек (соседей), с которыми точка p соединена ребром

$$U^2(p) = -U^1(p) + \frac{1}{n(p)} \sum_i U^1(p,i)$$

Тогда уравнение $U^2(p)=0$ линейное относительно p и его можно быстро решить.
Решение при w=1:

$$U^2(p)=0$$

$$-U(p)+\frac{1}{n(p)}\sum_i U(p,i)=0$$

$$p-\frac{1}{n(p)}\sum_i p(p,i)+\frac{1}{n(p)}\sum_i\left[-p(p,i)+\frac{1}{n(p,i)}\sum_j p(i,j)\right]=0$$

$$p-\frac{1}{n(p)}\sum_i p(p,i)+\frac{1}{n(p)}\sum_i\left[-p(p,i)+\frac{1}{n(p,i)}\left[p+\sum_{j,j\neq p} p(i,j)\right]\right]=0$$

$$p-\frac{1}{n(p)}\sum_i p(p,i)+\frac{1}{n(p)}\sum_i\left[-p(p,i)+\frac{p}{n(p,i)}+\frac{1}{n(p,i)}\sum_{j,j\neq p} p(i,j)\right]=0$$

$$p+\frac{p}{n(p)}\sum_i\frac{1}{n(p,i)}-\frac{2}{n(p)}\sum_i p(p,i)+\frac{1}{n(p)}\sum_i\frac{1}{n(p,i)}\sum_{j,j\neq p} p(i,j)=0$$

$$p\left[1+\frac{1}{n(p)}\sum_i\frac{1}{n(p,i)}\right]=\frac{2}{n(p)}\sum_i p(p,i)-\frac{1}{n(p)}\sum_i\frac{1}{n(p,i)}\sum_{j,j\neq p} p(i,j)$$

$$pv=\frac{2}{n(p)}\sum_i p(p,i)-\alpha(p)$$

$$p=\frac{2}{n(p)v}\sum_i p(p,i)-\frac{1}{v}\alpha(p)$$

Если вес w(i, j)=||i-j|, то есть вес равен длине ребра между точками i и j, то уравнение $U^2(p)=0$ уже нелинейно зависит от p, так как при вычислении весов уже нужно брать корень из координат точек. Это уравнение решается методом градиентного спуска. После того как уравнение $U^2(p)=0$ решено для каждой точки p из затяжки, точки затяжки сдвигаются в найденные значения. И снова для каждой точки p из затяжки решается уравнение.

Метод быстрого градиентного спуска

При коррекции кривизны поверхности для каждой точки решается система уравнений

$$U^2(p) = -U^1(p) + \frac{1}{w(p)} \sum_i w(p,i) U^1(p,i) = 0$$

U^2 построен на основе U^1:

$$U^1(p) = -p + \frac{1}{w(p)} \sum_i w(p,i)\, p(p,i)$$

$$w(p) = \sum_i w(p,i)$$

где p(p, i) – сосед точки p и w(p, i) – вес ребра между точками p и i.

При весе ребра w(p, i)=1 система уравнений $U^2(p)=0$ решается точно.
При весе ребра w(p, i)=||p-i|| решить систему уравнений точно невозможно. Система уравнений $U^2(p)=0$ состоит из трех уравнений:

$U^2_1(x,y,z)=0$
$U^2_2(x,y,z)=0$
$U^2_3(x,y,z)=0$

Находим минимум функций трех переменных:

$$f(x,y,z) = \sqrt{(U_1^2(x,y,z))^2 + (U_2^2(x,y,z))^2 + (U_3^2(x,y,z))^2}$$

Для функций многих переменных минимум вычисляется с использованием метода быстрого градиентного спуска. Сначала определяется начальная точка. Градиент строится в этой точке. Градиент к функциям - это вектор, указывающий, в каком направлении необходимо двигаться, чтобы функция росла наиболее быстро. Так, двигаясь вдоль антиградиента, функция будет наиболее быстро уменьшаться. С помощью метода градиентного спуска вычисляется минимум по антиградиенту до уменьшения функции и после, когда она начнет опять увеличиваться, и так строится необходимая точка:

$$x^{(k+1)} = x^{(k)} - \alpha^{(k)} \nabla f(x^{(k)})$$

где $x^{(k)}$ - точка на к-м шаге; $a^{(k)}$ - сдвиг по антиградиенту.
Необходимо найти *a* до уменьшения функции и после, когда она начнет увеличиваться. То есть двигаться вдоль антиградиента до уменьшения функции. При движении вдоль антиградиента, функция многих переменных *f* рассматривается как функция одной переменной, то есть – сдвиг по антиградиенту. Для поиска *a* используется метод вычисления минимума для функций многих переменных - метод интерполяции наименьших квадратов. Этот метод аппроксимирует функцию с помощью полинома второй степени по трем точкам. Затем итерационно строятся следующие три точки, до тех пор, пока не будет найден минимум с необходимой точностью. При быстром градиентном спуске можно решить систему уравнений $U^2(p)=0$ для любых весов.

Заключение

Предложен метод триангуляции функционально заданных объектов, обеспечивающий равномерность построенной сетки объекта. При решении задачи реализованы: 1) алгоритм вычисления точек поверхности функционального объекта; 2) алгоритм связывания точек поверхности в четырехугольные полигоны; 3) алгоритм затяжки дырок в сетке,

полученной после замыкания точек поверхности; 4) алгоритм добавления дополнительных вершин для обеспечения равномерности сетки в любой ее подобласти; 5) алгоритм слияния затяжек с прилегающими гранями; 6) экспорт сетки в формат ASE.

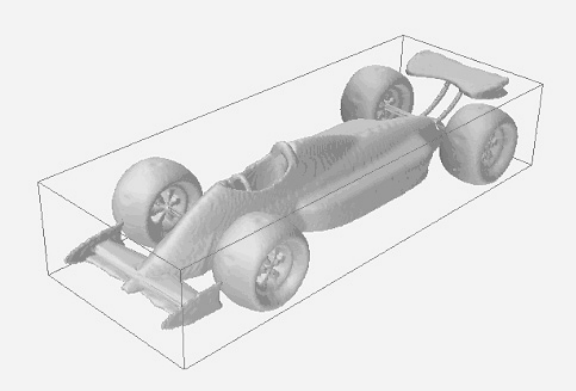

Рисунок 14 – Функциональная модель после триангуляции; Quatro level=9; Binary level =10; Кол-во треугольников=136306; Кол-во вершин=68418; Исходная функциональная модель была задана 83 функциями возмущения

Для триангуляции дырок в сетке и построения на основе триангуляции дырок затяжек был использован и модифицирован алгоритм Лиепы [12].

Реализован метод, обеспечивающий равномерность сетки в любой подобласти объекта, с возможностью точного преобразования объектов с жесткими ребрами.

Реализован экспорт точек поверхности и экспорт сетки треугольников.

Для анализа отклонений (отклонение вершин и нормалей) был разработан анализатор критериев отклонений. Анализ степени приближения полученной сетки к функционально заданной поверхности осуществляется по двум метрикам: отклонение вершин треугольной сетки от поверхности функционального объекта; отклонение нормалей к вершинам треугольной сетки от значений нормалей к вершинам на функциональной поверхности. На рисунке 14 показана полигонизированная функциональная модель.

Литература

1. Pasko A. A., Adzhiev V. D., Sourin, A. I. et al. Function representation in geometric modeling: concepts, implementation and applications // The Visual Computer. 1995. 11(6), P. 429.
2. J. F. Blinn. A generation of algebraic surface drawing. ACM Transactions on Graphics.- July 1982. 1(3), P. 235.
3. S. Muraki. Volumetric shape description of range data using “blobby model”. Computer Graphics.- July 1991. 25(4), P. 227.
4. H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura. Object modelling by distribution function and a method of image generation. The Transactions of the Institute of Electronics and Communication Engineers of Japan. 1985. J68D (4), P. 718.
5. Bloomenthal J., Shoemake K. “Convolution surfaces”, SIGGRAPH’91, Computer Graphics, 1991. Vol.25. No.4. P 251.
6. G. Sealy, G. Wyvill. Smoothing of three dimensional models by convolution. In Computer Graphics International’96. June 1996. P 184.
7. McCormack J., Sherstyuk A. Creating and rendering convolution surfaces. Computing Graphics Forum. 1998. Vol. 17. No.2. P 113.
8. Вяткин С.И., Долговесов Б.С., Есин А.В. и др. Геометрическое моделирование и визуализация функционально-заданных объектов // Автометрия. 1999. N6. С.65.
9. Andrew Witkin, Paul S. Heckbert. Using particles to sample and control implicit surfaces, SIGGRAPH 94. 1994. P.269.
10. S. Akkouche, E. Galin, Adaptive implicit surface polygonization using Marching Triangles, Computer Graphics Forum. 20:2. 2001. P. 67.
11. W.E. Lorensen, H.E. Cline, Marching Cubes: a high resolution 3D surface reconstruction algorithm, SIGGRAPH 87. 1987. P.163.
12. Peter Liepa. “Filling Holes in Meshes”// Proc. Eurographics/ACM SIGGRAPH Symp. Geometry Processing, 2003, P. 200.