Триангуляция Функциональных Форм

Шевелев С.В., Вяткин С.И., Долговесов Б.С.
Институт автоматики и электрометрии, СО РАН, Новосибирск, Россия
serge_shevelev@gorodok.net, sivser@mail.ru, bsd@iae.nsk.su.

Аннотация

В настоящее время для визуализации и моделирования трехмерных объектов широко используются способ, основанный на задании треугольников, аппроксимирующих поверхность объекта. Под этот стандарт сделаны графические карты – акселераторы, поэтому геометрические объекты в этом формате визуализируются в реальном времени (что пока не сделано для функционального формата, что без сомнения будет сделано в будущем, как это было реализовано для воксельного формата: в настоящее время разработан ускоритель VolumePro для воксельного представления). В результате проделанной работы предложен общий подход к триангуляции функционального объекта, обеспечивающий равномерность сетки объекта в любой ее подобласти. Разработан и реализован алгоритм конвертации 3-х мерного объекта, заданного функционально на базе функций возмущения, в стандартное представление в виде списков вершин и треугольных граней (сетки).

***Ключевые слова: Триангуляция**, Рекурсивное Деление Объектного Пространства.*

1. ВВЕДЕНИЕ

Моделирование и визуализация трехмерных объектов являются важнейшими частями компьютерной графики. В настоящее время для визуализации и моделирования трехмерных объектов широко используются способ, основанный на задании треугольников, аппроксимирующих поверхность объекта. В данном способе представления трехмерного объекта для достижения более высокой степени аппроксимации поверхности необходимо увеличивать количество треугольников. Это приводит к расходованию больших вычислительных ресурсов и памяти. Поэтому сегодня широко развиваются новые подходы к моделированию и визуализации трехмерных сцен, ключевым моментом которых является отказ от использования многоугольников в качестве основы для моделирования геометрических форм. В Лаборатории синтезирующих систем визуализации ИАиЭ СО РАН разработан свой способ задания свободных форм на основе 3-х мерных функций второго порядка. Основными характеристиками функционального задания являются:

- Сжатое описание (компрессия) – описание сложного объекта, представленного функционально, на несколько порядков более компактно, чем его аналог в треугольном формате
- Простота геометрических операций (деформирование свободной формы, трехмерный морфинг)
- Унарные операции над функциональными объектами (сжатие, расширение, проекция)
- Бинарные операции над функциональными объектами (теоретико-множественные (булевы) операции, морфинг)
- Операции отношения (быстрое определение столкновений с полной гарантией определения события за постоянное время)

На сегодняшний день наиболее широко используется традиционное стандартное треугольное описание объекта-модели. Таким образом, напрашивается логичный вывод, что необходимо решить задачу конвертации из функционального задания для представления объекта в стандартное широко распространенное треугольное задание. Основными требованиями являются:

- Разработка алгоритма, использующего преимущества задания функциональных объектов на базе функций возмущения, для нахождения точек поверхности.
- Триангуляция функциональных 3-х мерных объектов, что позволит отображать их при помощи существующих графических карт в реальном времени.
- 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 2 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.

2. ИСТОРИЯ ВОПРОСА

На сегодняшний день разработано много разных функциональных способов задания геометрических 3-х мерных моделей: F-rep [1], неявные (implicit) поверхности (Blobby-модели [2], метасферы [3], поверхности свертки [4] и т.д.). Эти способы отличаются друг от друга заданием ядра – базовой функции и операциями, заданными над базовыми функциями. Из двух базовых функций получается результирующая поверхность, как результат обтяжки оболочкой 3-х мерного остова, составленного из выбранных примитивов. Например, в blobby методе из нескольких сфер образуется гантелеобразная капля с перетяжкой. В поверхностях свертки из нескольких скелетонов разных видов (точки, линии, дуги, треугольники, плоскости) образуется натяжка на остов. Для разных типов функциональных форм используются разные алгоритмы триангуляции, которые основаны на возможностях, предоставляемых конкретным типом функционального задания. Эти способы триангуляции можно разделить на несколько основных типов:

- Частицы. Метод основан на взаимодействии частиц с функциональной поверхностью [5]. Этот способ подходит только для неявных (implicit) поверхностей.
- Расширяющееся построение на основе ранее построенных граней. Этот метод основан на вычислении положения соседних граней для уже построенных граней [6]. Этому методу необходима затравочная грань, вокруг которой строятся остальные грани. Для реализации данного метода необходимо вычислять градиент функции, что в нашем задании сделать нельзя.
- Алгоритмы, разработанные для триангуляции облаков точек, полученных после сканирования поверхности реальных объектов (например, скульптур) [7]. В этих алгоритмах необходимо производить поиск по всему пространству, что сильно расходует память и ресурсы. В таких алгоритмах условие равномерности сетки может стать невыполнимым.

В отличие от всех предложенных способов, в рассматриваемом функциональном задании используются функции возмущения относительно базовых примитивов [8-10]. Посредством функций возмущения можно строить сложные поверхности с большим количеством деталей. Существующие способы [1-4] 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 вышеперечисленных функциональных 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.

3. ТРИАНГУЛЯЦИЯ МОДЕЛЕЙ НА БАЗЕ ФУНКЦИЙ ВОЗМУЩЕНИЯ

В данной работе использовался алгоритм многоуровневого отслеживания лучей [8] для визуализации моделей на базе функций возмущения [9], адаптированный для задачи триангуляции. Функционально заданные геометрические объекты с применением функций возмущения достаточно подробно изложены в работах [8-10].

Вычисление состоит из двух этапов: первый этап – деления пространства по четвертичному дереву (вычисление лучей, пересекающих объект), второй этап – для вычисленного луча осуществляется бинарный поиск. При делении по четвертичному дереву происходит деление бруска на 4 части и определение расположения бруска относительно объекта. Те бруски, для которых тест на пересечение определил, 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. 1). 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.

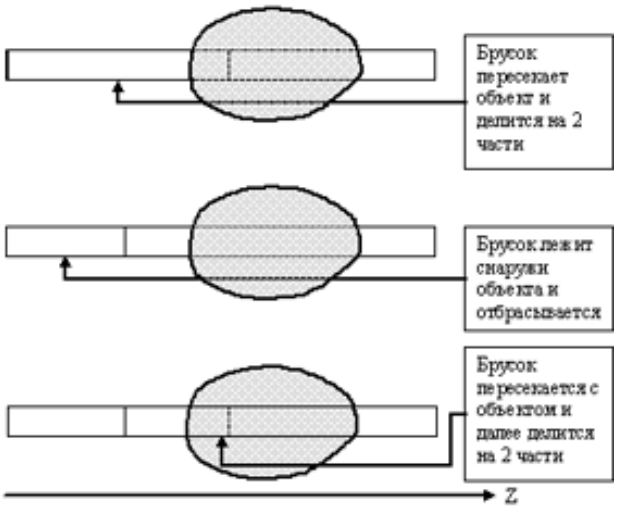

Рис. 1: Бинарное деление луча для нахождения ближайшей точки пересечения луча с поверхностью объекта.

3.1 Построение точек, принадлежащих поверхности функционального объекта

Для триангуляции функционального объекта необходимо построить облако точек, которые принадлежат поверхности объекта. Алгоритм нахождения точек поверхности работает в 2 этапа:

- 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.

- испускание луча через каждую ячейку проекции и нахождение всех его точек пересечения с поверхностью. Через каждую ячейку проекции пускается в направлении z луч и бинарным делением пополам находятся все точки пересечения луча с поверхностью.

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.

3.2 Связывание точек, принадлежащих поверхности функционального объекта, в четырехугольные полигоны

3.2.1 Определение воксела

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:

$$S_{x,y}=1/2^{QLEV}$$

При максимальном уровне подделения луча по бинарному дереву равному BLEV, размер ячейки-воксела по оси z равен:

$$S_z=1/2^{BLEV}$$

3.2.2 Вертикальные и горизонтальные сечения поверхности функционального объекта

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 – в них не находится объект. Часть вокселов – вокселы, где находятся точки поверхности. Часть вокселов – вокселы, которые находятся внутри объекта. 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.

3.2.3 Нахождение соседних точек для точки поверхности

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
3. двигаясь по контуру сечения взять соседние точки для данной точки.

Рис. 2: Точка B и ее соседи в вертикальном сечении эллипсоида – точки A и C.

На Рис. 2 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).
- 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. См. Рис. 3.
- Первая точка, которая является не пустой, выбирается текущей. Относительно нее и предпоследней точки строится новый вектор.

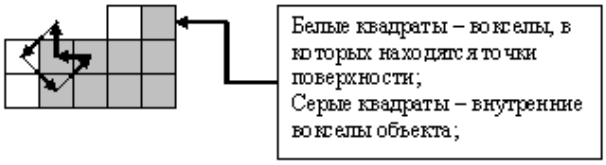

Рис. 3: Обход вокселей при поиске следующего направления движения.

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.

3.2.4 Связывание точек поверхности в четырехугольные полигоны

Для связывания 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 мы замыкаемся* на начальную точку A, то таким образом построен четырехугольный полигон ABCD, состоящий из 2 граней ABC и ACD, аппроксимирующих поверхность функционального объекта (см. Рис. 4).

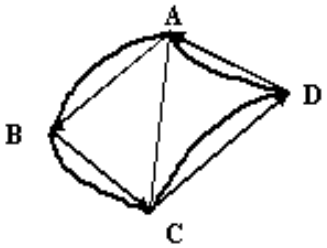

Рис. 4: Четырехугольный полигон A->B->C->D, аппроксимирующий поверхность функционального объекта; A, B, C, D – точки поверхности функционально заданного объекта.

Так как не всегда мы возвращаемся в начальную точку при обходе вниз-вправо-вверх-влево, то в некоторых местах сетки объекта получаются дыры (см. Рис. 5).

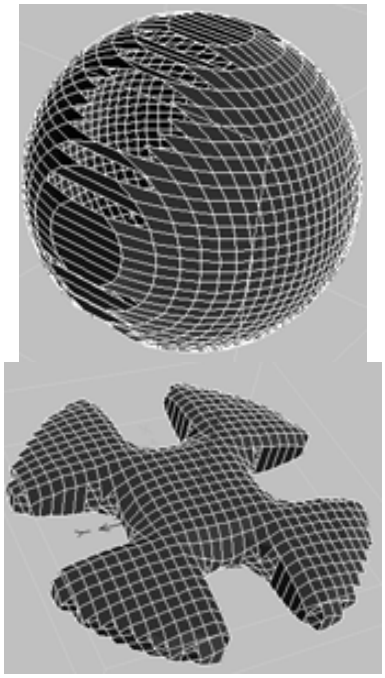

Рис. 5: Объединение точек поверхности в четырехугольные полигоны; после объединения видны дыры в сетке.

3.3 Триангуляция дырок в сетке после связывания точек поверхности в четырехугольные полигоны

3.3.1 Определение и поиск дырок в сетке

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 1, то это ребро является ребром на границе дырки. Если количество связанных с ребром граней равно 2, ребро является той частью треугольной сетки, в которой нет дырок. Отсюда следует что, чтобы найти и запомнить все дырки, мы должны искать ребра, каждое их которых связано только с одной гранью. Алгоритм поиска дырки:

- Определить начальное ребро дырки (p_1, p_2)
- Среди всех ребер, которые исходят из p_2, найти то ребро (p_2, p_3), p_3 не равно p_1, которое принадлежит только одной грани, то есть является ребром на границе дырки
- Для точки p_n повторить поиск ребра (p_n, p_{n+1}), p_{n+1} не равно p_{n-1}, которое принадлежит только одной грани, то есть является ребром на границе дырки
- Поиск дырки закончен, как только мы возвращаемся в начальную точку p_1

Таким образом, находятся все дырки в сетке, полученной после связывания точек в четырехугольные полигоны.

3.3.2 Триангуляция дырок

Для триангуляции дырок используется модифицированный алгоритм Лиепы [11] 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 $V=\{v_0, v_1, \ldots, v_{n-1}\}$, $P=(v_0, v_1, \ldots, v_{n-1})$ – полигон, на вершинах $v_0, v_1, \ldots, v_{n-1}$, 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 [11]:

1. Для i=0,1,…., n-2 положим функцию веса для 2-х 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 возвращаемся на шаг 2. Иначе вес минимальной по весу триангуляции равен Wo,n-1
4. Восстановление граней, образующих триангуляцию с минимальным весом, по значениям $M_{i,k}$

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 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.

3.3.3 Добавление вершин в затяжку дырки

Полученные затяжки дырок в сетке по алгоритму, описанному в предыдущем пункте, 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 (см. Рис. 6).

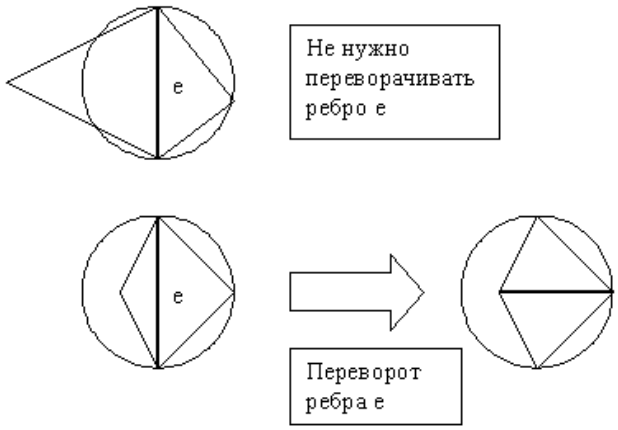

Рис. 6: Переворот ребра.

Алгоритм добавления вершин в затяжку:

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 3, 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 3 подграни (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 3 части ни одной грани, то добавление вершин завершено.
4. Восстановить все внутренние ребра затяжки
5. Если ни одно ребро не было перевернуто, вернуться к шагу 2. Иначе повторить шаг 4.

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 (см. Рис. 7).

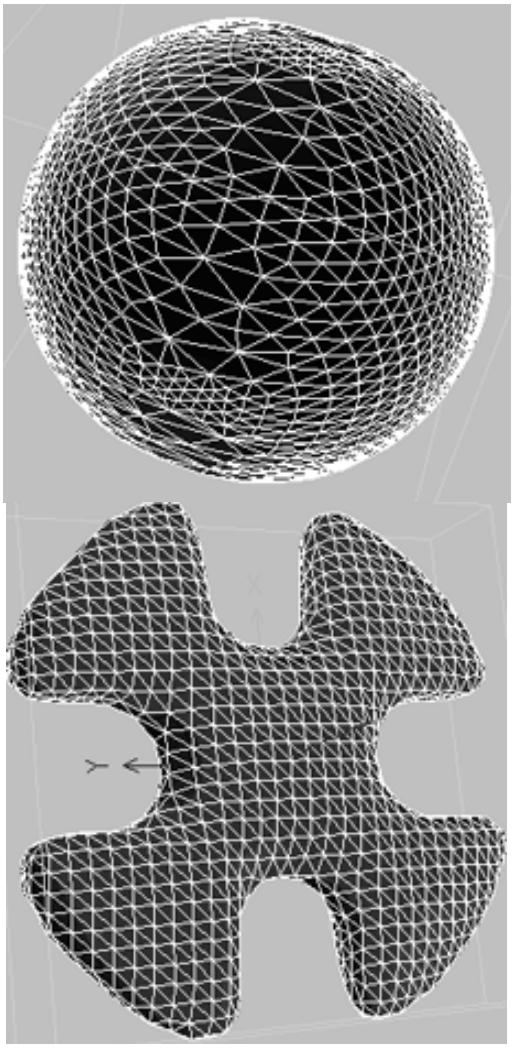

Рис. 9: Добавленные вершины и полученные затяжки.

3.3.4 Соответствие кривизны затяжки кривизне поверхности объекта (Refinement)

Для каждой точки p затяжки строится оператор U^1:

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

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

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:

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

$U^1(p,i)$ – 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

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

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 по 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 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 предлагается попробовать решить методом градиентного спуска.

После того как уранение $U^2(p)=0$ решено для каждой точки p из затяжки, точки затяжки сдвигаются в найденные значения. И снова для каждой точки p из затяжки решается уравнение.

4. ЗАКЛЮЧЕНИЕ

Предложен алгоритм триангуляции функционально заданных объектов, обеспечивающий равномерность построенной сетки объекта. При решении задачи реализовано:

- алгоритм получения точек поверхности функционального объекта
- алгоритм связывания точек поверхности в четырехугольные полигоны
- алгоритм затяжки дырок в сетке, полученной после связывания точки поверхности
- алгоритм добавления дополнительных вершин для обеспечения равномерности сетки в любой ее подобласти
- алгоритм слияния затяжек с прилегающими гранями
- экспорт сетки в формат ASE

Для триангуляции дырок в сетке и построения на основе триангуляции дырок затяжек был использован и модифицирован алгоритм Лиепы [11]. В результате работы написана программа-экспортер – триангулирующая функционально заданный объект и обеспечивающая равномерность построенной сетки объекта, а также экспортирующая сетку в виде треугольных граней в широко распространенный формат ASE. Реализованы способы, обеспечивающие равномерность сетки в любой подобласти объекта. Реализован экспорт точек поверхности и экспорт сетки.

5. REFERENCES

[1] Pasko A.A., Adzhiev V.D., Sourin A.I. et al. Function representation in geometric modeling: concepts, implementation and applications //Visual Computer. 1995. 11, N 6. P. 429.

[2] Muraki S. Volumetric shape description of range data using “blobby model”. //Computer Graphics. 1991. 25, N 4. P. 227.

[3] Nishimura H., Hirai M., Kawai T. et al. Object modeling by distribution function and a method of image generation // The Transactions of the Institute of Electronics and Communication Engineers of Japan. 1985. J68-D(4). P. 718.

[4] Bloomenthal J., Shoemake K. Convolution surfaces // Computer Graphics. 1991. 25, N 4. P. 251.

[5] Andrew Witkin, Paul S. Heckbert, Using particles to sample and control implicit surfaces, SIGGRAPH 94, 1994, pp.269-277.

[6] S. Akkouche, E. Galin, Adaptive implicit surface polygonization using Marching Triangles, Computer Graphics Forum, 20:2, 2001, pp. 67-80.

[7] W.E. Lorensen, H.E. Cline, Marching Cubes: a high resolution 3D surface reconstruction algorithm, SIGGRAPH 87, 1987, pp.163-169.

[8]http://www.cgg.ru/febr/vjat/pivweb.html#a

[9] Sergei I. Vyatkin, Boris S. Dolgovesov, Oleg Y. Guimaoutdinov - "Synthesis of Virtual Environment Using Perturbation Functions", volume III (Emergent Computing and Virtual Engineering), World Multiconference on Systemics,

Cybernetics and Informatics Proceedings, Orlando, FL, USA, July 22-25, 2001, P.350.
[10]http://www.graphicon.ru/2002/pdf/Vyatkin_Dolgovesov_Re.pdf
[11] Peter Liepa “Filling Holes in Meshes”, Eurographics Symposium of Geometry Processing (2003).

Triangulation Of Functional Forms

For rendering function defined shapes based on perturbation functions, either Voxel-Volumes Rendering Algorithm or triangulation followed by fast triangle rendering is used. 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. We propose approach for accurate triangulation of functionally defined objects using perturbation functions. For analyzing how close the evolving mesh approaches the functionally-defined surface we will use two error metrics and uniformity. The metrics measure deviations of the vertices from the functionally-defined surface and deviations of mesh normals from the normal of the functionally-defined surface.

About the authors

Sergei Shevelev is a Ph.D. student at Novosibirsk State University, Department of Physics. His contact email is serge_shevelev@gorodok.net

Sergei I. Vyatkin (Ph.D.) is a scientific researcher of Synthesizing Visualization Systems Laboratory at Institute of Automation and Electrometry, SB RAS.

His contact email is sivser@mail.ru

Boris S. Dolgovesov (Ph.D.) is a head of Synthesizing Visualization Systems Laboratory at Institute of Automation and Electrometry, SB RAS.

His contact email is bsd@iae.nsk.su