УПРОЩЕНИЕ ТРИАНГУЛЯЦИОННЫХ МОДЕЛЕЙ ПОВЕРХНОСТИ

Н.С. МИРЗА, А.В.СКВОРЦОВ, Р.В.ЧАДНОВ

В настоящее время очень много графических программных систем использует цифровые модели поверхности. Для задач ГИС, САПР, систем трёхмерного моделирования и других графических систем очень актуальной является проблема обработки больших (детализированных) поверхностей, так как для работы с поверхностями, которые могут не помещаться в оперативную память компьютера, требуется мощное аппаратное обеспечение. Поэтому разработано большое число алгоритмов генерализации (упрощения) поверхности, которые позволяют значительно уменьшить детализацию поверхности, вызывая существенное увеличение скорости выполнения задач, связанных с обработкой поверхностей.

Основные понятия и постановка задачи

В данной статье под термином «поверхность» будем понимать однозначную функцию высот от планового положения точек. Такие поверхности ещё называют *2,5-мерными,* чтобы отличить это понятие от значения общего понятия «3-мерная поверхность», принятого в математике.

Одним из самых распространённых методов представления поверхности является *нерегулярная сеть треугольников* (TIN), представляющая собой набор треугольников, образующих в проекции на ось *XY триангуляцию* – планарный граф, все конечные грани которого являются треугольниками [1]. Таким образом, TIN – это триангуляция, каждому узлу которой поставлена в соответствие его высота (координата *Z*).

Определение. Пусть имеется TIN T, содержащая N узлов. В задаче построения упрощающей TIN (задаче генерализации) требуется найти такую TIN t, что [1]:

- она содержит заданное количество узлов $n = |t| < N$ и имеет минимальное отклонение d от T: $d(T,t) = \min_{\tau} d(T,\tau), |\tau| = n$ (задача, управляемая геометрией);
- она имеет отклонение d по вертикали от T не более чем на заданную величину $n(t) = \min_{\tau} n(\tau)$, и имеет минимальное количество узлов $n(t) = \min_{\tau} n(\tau),\ d(T,\tau) < \varepsilon$ (задача, управляемая ошибкой).

Задача упрощения TIN в обоих вариантах является NP-сложной, поэтому на практике используются приближённые алгоритмы [1], которые применяют некоторый критерий качества упрощённой модели.

Все методы упрощения триангуляционных поверхностей можно разделить на следующие классы в зависимости от критерия оптимальности [2]:

- методы, основанные на локальном критерии оптимальности, то есть методы, в которых точность вычисления связана с каждым элементом поверхности (таким как узел, ребро или треугольник);
- методы, основанные на глобальном критерии оптимальности, т.е. методы, в которых точность вычисления связана исключительно со всей поверхностью;
- методы, основанные на некотором специальном критерии оптимальности;
- экспертные методы, не использующие критерий оптимальности (экспертные методы), то есть методы, полностью основанные на визуальной оценке пользователя.

Методы упрощения триангуляционных моделей

Все существующие методы упрощения TIN основаны на одной из двух стратегий:

1. Стратегия «сверху вниз» начинает работу с максимально упрощённой TIN. Далее постепенно в TIN добавляются некоторые узлы в порядке согласно некоторому критерию до тех пор, пока не будет достигнуто нужное число узлов или величина ошибки будет удовлетворительной [3].

2. Стратегия «снизу вверх» начинает работу с самой детализированной TIN, к которой постепенно применяются некоторые *локальные модификации* – операции, заменяющие одну маленькую группу смежных треугольников на другую, покрывающую ту же область (рис. 1).

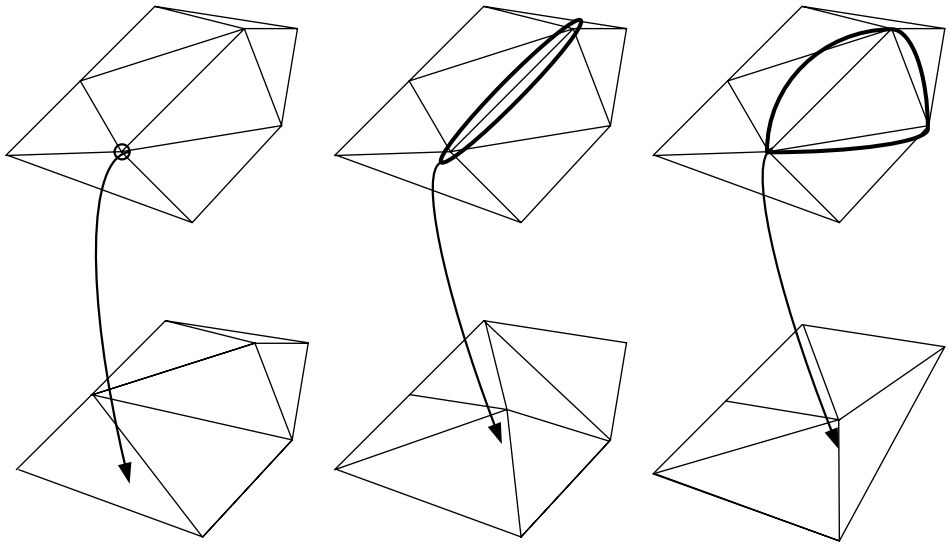

Рис. 1. Локальные модификации: *а* – удаление узла; *б* – коллапс ребра; *в* – коллапс треугольника

Преимуществом стратегии «снизу вверх» является то, что она основана исключительно на операции вставки узла в триангуляцию, которая используется при построении TIN, в отличие от стратегии «сверху вниз», основанной на операции удаления узла, которая достаточно сложна и должна быть реализована в дополнение к обычному алгоритму построения TIN.

Однако стратегия «сверху вниз» позволяет параллельно с упрощением построить так называемую иерархию упрощений, сохраняемую в специальной структуре – *мультитриангуляции* [4], с помощью которой можно получить поверхность переменного разрешения, имеющую важное практическое значение.

Поэтому далее приводится ряд методов, основанных на стратегии «снизу вверх» [8]:

1. Объединение треугольников, лежащих в одной плоскости. Идея метода такова: производится поиск треугольников, лежащих в одной плоскости, после чего найденные треугольники удаляются и происходит заполнение образовавшейся дырки.

а)

2. Управляемое удаление узлов, рёбер и треугольников. Данный метод представляет собой итеративный процесс, на каждом шаге которого применяется некоторая локальная модификация, выбранная согласно критерию минимизации ошибки между исходной и упрощённой поверхностью.

3. Перестроение поверхности. Данный метод основан на понятии «кривизна поверхности» [9]. Сначала случайным образом в исходную поверхность вставляется заданное число узлов, которые затем перемещаются в места максимальной кривизны поверхности. После чего производится итеративный процесс удаления узлов.

4. Оптимизация энергетической функции. Данный метод основан на итеративном процессе применения локальных модификаций, оптимизирующих *энергетическую функцию* [5], которая характеризует качество упрощённой модели. Узлы, дающие наименьшее приращение энергетической функции, итерационно удаляются.

5. Кластеризация. Данный метод основан на построении кластеров из близлежащих узлов (рёбер, треугольников) и последующей замене их на более упрощённые фрагменты.

6. Иерархическое представление. Данный метод основан на построении дерева октантов [6] или элементов объёмного изображения (вокселей) [7].

Заметим, что некоторые из описанных алгоритмов упрощения могут применяться как для 2,5-мерных поверхностей, так и для 3-мерных.

Далее приводится сравнение описанных методов упрощения по ряду характеристик [2].

1. По соотношению исходной и упрощённой модели все методы делятся на следующие классы:

- методы, сохраняющие топологию (управляемое удаление узлов, рёбер и треугольников; оптимизация энергетической функции) либо не сохраняющие топологию (кластеризация, иерархическое представление);

- методы, основанные на выборе некоторого множества узлов (управляемое удаление узлов, рёбер и треугольников; объединение треугольников, лежащих в одной плоскости) либо на перестроении (оптимизация энергетической функции; перестроение; иерархическое представление).

Как правило, методы, не сохраняющие топологию, работают быстрее, но для решения многих задач сохранение топологии является критически важным.

Заметим, что сохранение топологии актуально только для 3-мерных поверхностей, так как при применении любого из описанных алгоритмов упрощения для 2,5-мерных поверхностей топология сохраняется в любом случае.

Методы, основанные на перестроении, работают несколько точнее, но они применимы не для всех задач.

2. По цели упрощения все методы делятся на следующие классы:

- методы, для которых задана некоторая величина ошибки ε и необходимо построить модель с минимальным числом узлов;
- методы, для которых задано минимальное число узлов и необходимо минимизировать разницу между исходной и упрощённой моделями.

3. По методу вычисления ошибки все методы делятся на следующие классы:

- методы, основанные на локальном критерии оптимальности (управляемое удаление узлов, рёбер и треугольников), либо методы, основанные на глобальном критерии (перестроение; оптимизация энергетической функции);
- методы, сохраняющие некоторые атрибуты поверхности, такие как цвет, структурные линии и т.п. (управляемое удаление узлов, рёбер и треугольников) .

Как правило, на практике чаще всего пользуются наиболее простым способом упрощения – итеративным процессом, который можно разбить на следующие этапы:

Этап 1. Классификация узлов.

Этап 2. Вычисление ошибки, вызванной применением локальных модификаций.

Этап 3. Выбор локальной модификации, вызывающей минимальную ошибку.

Этап 4. Применение локальной модификации с соответствующим локальным перестроением TIN.

Для классификации каждый узел относят к одному из следующих классов [10]:

1. *Простой узел* – узел, окружённый полным замкнутым циклом из треугольников (рис. 2, *а*).

2. *Граничный узел* – узел, который лежит на границе поверхности, то есть окружен не полным циклом из треугольников (рис. 2, *б*).

Для определения следующих классов узлов введём термин «*внутренний край»* поверхности.

Определение. Если угол, образованный плоскостями двух смежных треугольников, больше некоторого заданного угла, то считается, что данные треугольники образуют *внутренний край* поверхности.

3. *Узел внутреннего края* – простой узел, используемый в образовании двух внутренних краёв поверхности (рис. 2, *в*).

4. *Угловой узел* – простой узел, используемый в образовании одного, трёх или более внутренних углов поверхности (рис. 2, *г*).

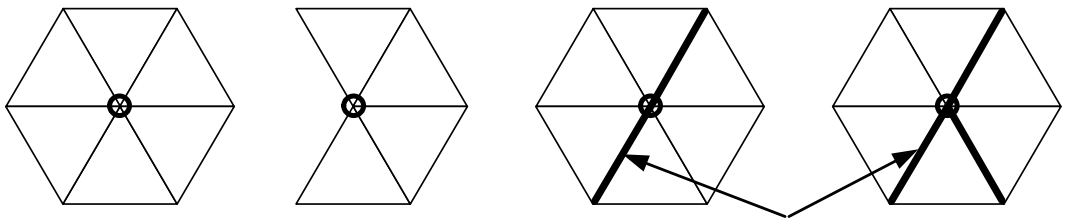

Рис. 2. Классификация узлов TIN: *а* – простой узел; *б* – граничный узел; *в* – узел внутреннего края; *г* – угловой узел

Узлы третьего и четвёртого типа не удаляются, а для удаления узлов второго типа необходимо учитывать неполноту цикла окружающих треугольников.

Для расчёта ошибки, вызванной той или иной локальной модификацией, как правило, используется величина отклонения (в заданной метрике [11, 12]) между исходной и упрощённой поверхностями, которое возникнет, если применить локальное преобразование.

Для реализации любого локального преобразования необходим алгоритм быстрого удаления узла из триангуляции, так как коллапс ребра и треугольника можно свести к операции вставки и удаления узлов из триангуляции.

Алгоритм удаления узла из триангуляции

Далее авторами приводится разработанный алгоритм удаления узла, на котором базируются все локальные модификации триангуляции.

В качестве языка описания алгоритма используется язык программирования Object Pascal.

Алгоритм удаления узла из триангуляции

Входные данные:

1. Триангуляция, построенная на множестве узлов `Nodes`.

2. Удаляемый узел `DeletedNode`.

Выходные данные:

Триангуляция, построенная на множестве узлов `Nodes - DeletedNode`.

Структуры данных:

1. В структуре исходной триангуляции должны содержаться связи каждого треугольника со всеми его соседями.

2. Списки узлов:

```
TNodeList = class
public
    // Добавить элемент в список
    procedure Add(Item: TNode);
    // Удалить последний элемент из списка
    procedure DeleteBack;
    // Элементы списка по индексу
    property Items[i: integer]: TNode read GetTriangle; default;
    // Число элементов в списке
    property Count: integer read GetTriangleCount;
    // Последний элемент в списке
    property Back: TNode read GetBack;
    // Первый элемент в списке
    property Front: TNode read GetFront;
end;
```

Класс `TNode` должен иметь следующую структуру:

```
TNode = class
    X, Y, Z: double;
end;
```

Используемые процедуры, функции и переменные:

1. `Up: TNodeList` – узлы верхней части, заполняемой после удаления `DeletedNode` дырки, `Low: TNodeList` – узлы соответствующей нижней части, `Join: TNodeList` – узлы объединённых `Up` и `Low`, отсортированные по координате `X`.

2. `S: TNodeList` – стек узлов, используемый для выбора узлов новых треугольников.

3. `Current: TNode` – текущий узел из стека `S`, `Front: TNode` – первый узел из стека `S`.

4. Список из трёх узлов `TriangleNodes: TNodeList`, в который сохраняются узлы для образования очередного нового треугольника. Узлы должны быть расположены по часовой стрелке.

5. **function** `JoinVertices(List1, List2: TNodeList): TNodeList;` Объединяет два входных списка в один, упорядоченный по координате `X`.

6. **function** `Adjacent(v1, v2: TNode): boolean;` Определяет смежны ли узлы `v1` и `v2`.

7. **function** `SetRightOrder(v1, v2, v3: TNode): TNodeList;` Возвращает список из узлов `v1`, `v2`, `v3` в порядке, соответствующем ходу построения. Если `Current` принадлежит `Up`, то порядок верен, иначе следует поменять `v2` и `v3` местами.

8. **function** `ClockwiseOrder(NodeList: TNodeList): boolean;` Определяет, по часовой ли стрелке расположены точки в списке.

9. **procedure** `AddTriangle(NodeList: TNodeList);` Добавляет новый треугольник из точек `NodeList` в триангуляцию и устанавливает все ссылки, необходимые для поддержания целостности структуры триангуляции. Производится удаление `DeletedNode` и всех смежных треугольников.

Структура алгоритма:

Шаг 1. Формируется список узлов `Nodes`, смежных с `DeletedNode`. Затем формируется список треугольников, смежных с удаляемыми треугольниками.

Шаг 2. Узлы Nodes делятся на две части: Up и Low такие, что для них справедливо следующее:

1. $\forall p \in$ Up $p.y \geq$DeletedNode.Y

2. $\forall p \in$ Low $p.y <$DeletedNode.Y

Шаг 3. Для узлов Up производится следующее:

```
i:=0;
while (UP.Count>=3) and (i< UP.Count-2) do
begin
     if ClockwiseOrder(UP[i], UP[i+1], UP[i+2]) then
     begin
        AddTriangle(UP[i], UP[i+1], UP[i+2]);
        if i>0 then i:=i+1;
     end
     else i:=i+1;
end;
```

Аналогичный алгоритм выполняется для Low .

Таким образом, будет выполняться следующее:

1. $\forall p1, p2 \in$ Up $p1.x < p2.x$, где $p1$ и $p2$ – смежные узлы при обходе узлов Nodes по часовой стрелке

2. $\forall p1, p2 \in$ Low $p1.x > p2.x$, где $p1$ и $p2$ – смежные узлы при обходе узлов Nodes по часовой стрелке

Это означает, что все узлы в Up и Low будут отсортированы по X.

Шаг 4. Дальнейшее заполнение дырки производится алгоритмом, описанным в [13].

Шаг 4.1. Все узлы Up и Low объединяются таким образом, чтобы общий список узлов Join был отсортирован по X.

Шаг 4.2. Используя стек узлов S, в цикле извлекаются три очередные точки и образуется новый треугольник.

При выполнении данного шага алгоритма возможно возникновение трёх случаев:

Случай 1. Текущий узел смежен с последним узлом в стеке и не смежен с первым узлом в стеке (рис. 3, *а*).

Случай 2. Текущий узел не смежен с последним узлом в стеке (рис. 3, б).

Случай 3. Образование так называемого «веерного полигона» (рис. 3, *в*), когда текущий узел смежен с первым и последним узлами стека.

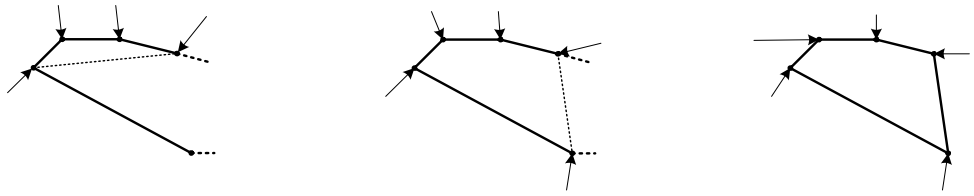

Рис. 3. Различные ситуации в алгоритме удаления узла из триангуляции: *а* – текущий узел алгоритма удаления узла смежен с последним узлом в стеке и не смежен с первым; *б* – текущий узел не смежен с последним узлом стека; *в* – текущий узел смежен как с первым, так и с последним узлом стека.

Далее для удобства описания приводится программный код шага 4.

```
Join:=JoinVertices(Up, Low);
S.Add(Join[0]);
S.Add(Join[1]);
i:=2;
while Join.Count>=3 do
begin
    Current:=Join[i];
    Front:=S.Back;
    // Случай, показанный на рис. 3, а
    if Adjacent(S.Back,Current) and (not Adjacent(S.Front,Current)) then
    begin
       TriangleNodes:=SetRightOrder(S.Back.Previous, Current, S.Back);
       while (S.Count>=2) and ClockwiseOrder(TriangleNodes) do
       begin
          AddTriangle(TriangleNodes);
          S.Pop;
       end;
       S.Add(Current);
    end;
    // Случай, показанный на рис. 3, б
    else if not Adjacent(S.Back, Current)
    begin
       TriangleNodes:=SetRightOrder(S.Back.Previous, Current, S.Back);
```

```
            while (S.Count>=2) and ClockwiseOrder(TriangleNodes) do
            begin
               AddTriangle(TriangleNodes);
               S.Pop;
            end;
            S.Pop;
            S.Add(Front);
            S.Add(Current);
         end;
         // Случай, показанный на рис. 3, в
         else
         begin
            TriangleNodes:=SetRightOrder(S.Back.Previous, Current, S.Back);
            while (S.Count>=2) and ClockwiseOrder(TriangleNodes) do
            begin
               AddTriangle(TriangleNodes);
               S.Pop;
            end;
         end;
      end;
```

Конец алгоритма.

Данный алгоритм имеет трудоёмкость $O(N)$ в худшем, где N – число узлов в триангуляции. Трудоёмкость складывается из следующих составляющих. Трудоёмкость выполнения шага 1 зависит от метода поиска треугольника триангуляции по заданному узлу. В наихудшем случае трудоёмкость поиска может составить $O(N)$, но если воспользоваться алгоритмом с кэшированием поиска треугольников, то трудоёмкость поиска смежных узлов и треугольников будет составлять $O(1)$ в среднем. Трудоёмкость разделения узлов на списки `Up` и `Low` равна $O(k)$, где k – число узлов в `NodesList`, т.е. число узлов в дырке, образовавшейся после удаления `DeletedNode`. Трудоёмкость алгоритма устранения «заломов» шага 3 равна $O(k)$. Алгоритм слияния узлов списков `Up` и `Low` работает с трудоёмкостью $O(k)$. И, наконец, трудоёмкость алгоритма заполнения оставшейся дырки шага 4 тоже равна $O(k)$.

Заключение

Разработанный авторами эффективный алгоритм удаления узла позволяет достаточно быстро строить упрощённые модели поверхностей с помощью локальных модификаций триангуляции. Преимуществом предложенного алгоритма удаления узла из триангуляции перед остальными аналогичными алгоритмами является линейная (относительно числа узлов дырки) трудоёмкость заполнения дырки, образовавшейся в триангуляции после удаления узла.

ЛИТЕРАТУРА

1. Скворцов А.В. Триангуляция Делоне и её применение. – Томск: Изд-во Том. ун-та, 2002. – 128 с.
2. Cignoni P., Montani C., Scopigno R. A Comparison of mesh simplification algorithms // Computer & Graphics. – 1998. – Vol. 22, № 1. – P. 37–54.
3. Heckbert P., Garland M. Survey of polygonal surface simplification // SISGRAPH. – 1997. – Course 25.
4. Puppo E. Variable resolution triangulations // Computational Geometry. – 1998. – Vol. 11. – P. 219–238.
5. Hoppe H., De Rose T., Duchamp T., McDonald J. Stuetzle W. Mesh Optimization // Computer Graphics Proceedings. – 1993. – P. 19–26.
6. Eck M., De Rose T., Duchamp T., Hoppe M., Lounsbery M., Stuetzle W. Multiresolutional analysis of arbitrary meshes // Computer Graphics Proceedings. – 1995. – P. 173–181.
7. Certain A., Popovich J., De Rose T., Duchamp T., Salesin D., Stuetzle W. Interactive multiresolutional surface viewing // Computer Graphics Proceedings.– 1996. –P. 91–98.
8. Puppo E., Scopigno R. Simplification, LOD and multiresolutional principles and applications // EUROGRAPHICS'97. – 1997. –Vol. 16, № 3.
9. Turk G. Re-tiling polygonal surfaces // Computer Graphics Proceedings. – 1992. – P. 55–64.
10. Schroeder W., Zarge J., Lorensen W. Decimation of triangle meshes // Computer Graphics. – 1992. – Vol. 26. – P. 65–70.
11. Garland M., Heckbert P. Surface simplification using quadric error metrics // Computer Graphics Proceedings. – 1997.
12. Klein R., Liebich G., Straber W. Mesh reduction with error control // Visualization Proceedings. – 1996. – P. 311–318.
13. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение / Пер. с англ. – М.: Мир, 1989. – 478 с.