Конвертация 3D модели методом оптимизирующего сжатия

С.В. Капустина, М.С. Клюев
Сибирский федеральный университет
e-mail: sv_kapustina@mail.ru

М.С. Клюев

В работе предложен алгоритм сжатия трехмерных графических моделей, который существенно уменьшает количество памяти необходимое для хранения и визуализации. Для реализации описанного алгоритма было создано програмное обеспечение с четырьмя основными уровнями: уровень управления входным файлом, уровень управления выходным файлом, графический уровень и уровень интерфейса. Графический уровень содержит процедуру, реализующую алгоритм конвертирования с оптимизацией.

1. Введение

Большинство существующих приложений используют 3D графику как основное средство визуализации. Современный механизм 3D-рендеринга нуждается в серьезном объеме оперативной памяти и высококачественном информационном оборудовании. Сжатие трехмерных графических моделей существенно уменьшает необходимое для процесса количество памяти.

Основным видом описания и хранения трехмерных моделей на сегодняшний день является полигональная сетка (polygonal mesh), когда поверхность модели состоит из множества граней - плоских n-угольников. Описание каркаса модели при этом состоит из набора проиндексированных вершин и списка граней, которые задаются перечислением индексов входящих в них вершин. Компрессия такой модели может быть разбита на четыре отдельных этапа.

Этап 1. Компрессия связности, то есть информации о входящих в состав модели n-угольниках. Основная цель здесь - максимально сократить число повторений индексов вершин.

Этап 2. Геометрическая компрессия - предсказание и квантизация координат вершин. Для эффективного предсказания необходимо упорядочить вершины по их взаимному расположению.

Этап 3. Компрессия атрибутов поверхности, таких как цвет, отражающие свойства, степень прозрачности и др. Вся эта информация сгруппирована в один или более материалов, и каждой грани модели присваивается свой материал.

Этап 4. Компрессия текстур, которая относится к другой области и в работе не рассматривается.

Данные, полученные на каждом этапе, обычно дополнительно сжимаются каким-либо энтропийным кодеком, чаще всего - арифметическим или Хаффмановским.

2. Алгоритм конвертации

Алгоритм решения задачи оптимизации в общем виде представляется следующим образом. Входной файл в формате Autodesk 3D Studio MAX ASCII обрабатывается с последующим формированием дополнительных данных и структур, проводится оптимизация Triangle Fan, Triangle Strip, анализируются треугольники не прошедшие оптимизацию. Полученный результат сохраняется в выходной файл. Также происходит сохранение отчёта со статистикой оптимизации.

Суть оптимизации Triangle Fan заключается в следующем, система использует вершины $V2$, $V3$ и $V1$ для отображения первого треугольника (рис. 1), $V3$, $V4$ и $V1$ для отображения второго, $V4$, $V5$ и $V1$ для отображения третьего, и т.д. Тем самым каждый последующий треугольник описывается с помощью центральной вершины, одной предыдущей и одной новой. Для успешной реализации всего алгоритма очень большую роль играет чёткая последовательность действий. Так как сегменты вееров треугольников оказывают влияние на корректность нахождения полос треугольников, то алгоритм включает в себя поэтапное исключение из множества не подвергнутых оптимизации (сжатию) треугольников сначала сегментов вееров треугольников, а затем только полос треугольников. Суть оптимизации Triangle Strip заключается в следующем, система

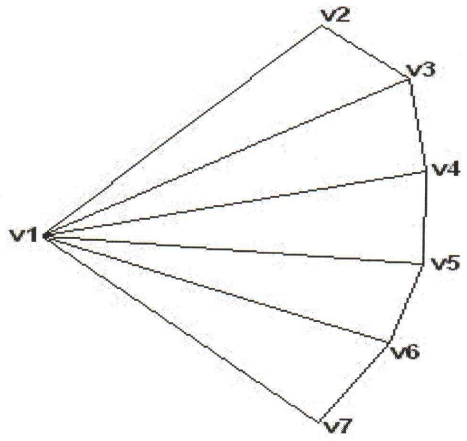

Рис. 1. Пример сегмента веера.

использует вершины $V1$, $V2$, и $V3$ для отображения первого треугольника, $V2$, $V4$, и $V3$ для отображения второго, $V3$, $V4$, и $V5$ для отображения третьего, $V4$, $V6$, и $V5$ для отображения четвёртого и так далее. Заметим, что вершины второго и четвёртого треугольника идут не по порядку; это необходимо, для того чтобы быть уверенным, что все треугольники отображаются по часовой стрелки. Тем самым каждый последующий треугольник описывается с помощью двух предыдущих вершин и одной новой.

Алгоритм нахождения всех сегментов вееров состоит в следующем (рис. 2).

1. Находим все предполагаемые центры сегментов вееров. Центром сегмента веера считается любая вершина принадлежащая более чем шести треугольникам.

2. Сортируем найденные центры по убыванию. Критерием сортировки является количество вхождений вершины в треугольники. Данный шаг необходим для улучшения качества оптимизации т.к. каждый найденный сегмент веера может оказать влияние на оставшиеся.

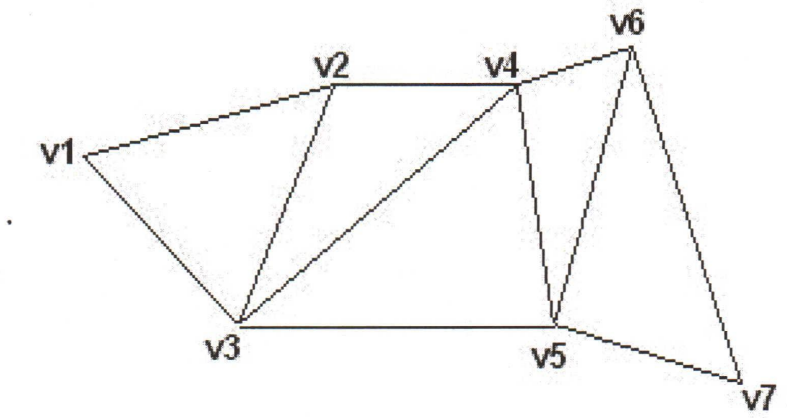

Рис. 2. Пример сегмента полосы.

3. Выбираем центр с наибольшим количеством вхождений вершины в треугольники. Количество вхождений при этом должно быть больше шести.

4. Находим все треугольники, в которые входит найденный центр сегмента веера.

5. Из найденных треугольников на шаге 4 составляется новый сегмент веера.

6. Треугольники найденные на шаге 4 помечаются как пройденные.

7. Если пройдены не все центры сегментов вееров, найденные на шаге 2, то переходим на шаг 3.

На шаге анализа треугольников, не прошедших оптимизацию, происходит декомпозиция тех полос треугольников, которые не удовлетворяют критериям. В данном случае критерием служит минимальное количество треугольников входящих в одну полосу.

Для реализации описанного алгоритма было создано программное обеспечение. Можно выделить четыре основные уровня разработанного ПО:

Уровень управления входным файлом. Данный уровень позволяет производить основные управляющие воздействия над исходными данными. В частности модуль позволяет загружать и сохранять исходные данные. Также он обладает набором функций необходимых для работы алгоритма.

Уровень управления выходным файлом. Данный уровень позволяет производить основные управляющие воздействия над выходными данными. В частности модуль позволяет загружать и сохранять выходные данные. Помимо этого данный модуль содержит процедуру реализующую алгоритм конвертировании с оптимизацией.

Графический уровень. Отвечает за отображения 3D моделей. А также за загрузку и хранение необходимых ресурсов для отображения 3D сцены.

Уровень интерфейса. Данный уровень позволяет пользователю управлять программой, а также получать всю необходимую информацию о ходе работы алгоритма.

Программа логически разбита на 15 модулей (рис. 3). Каждый модуль представляет собой логическую часть общей задачи и содержит все классы, которые необходимы для её решения. Среди всех модулей по значимости стоит выделить uMOTOP, uASE и uKMC, каждый из которых отвечает за основные функциональные возможности программы. Прослеживается чёткая взаимосвязь всех программных модулей с модулем иКОНСТАНТЫ, из названия которого становится понятно, что он содержит общие

базовые классы, а также все константы используемые в программе.

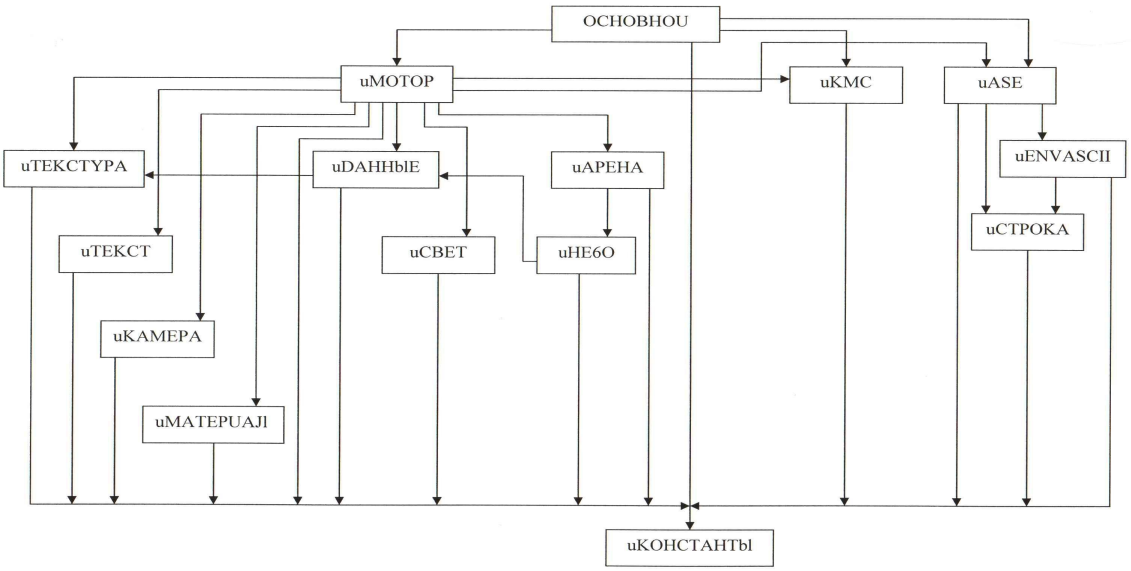

Рис. 3. Схема взаимосвязи программных модулей.

Разработанное ПО позволяет просматривать геометрические модели при их открытии, конвертировать модель, сохранять полученный результат, выводит характеристики геометрических моделей (рис. 4).

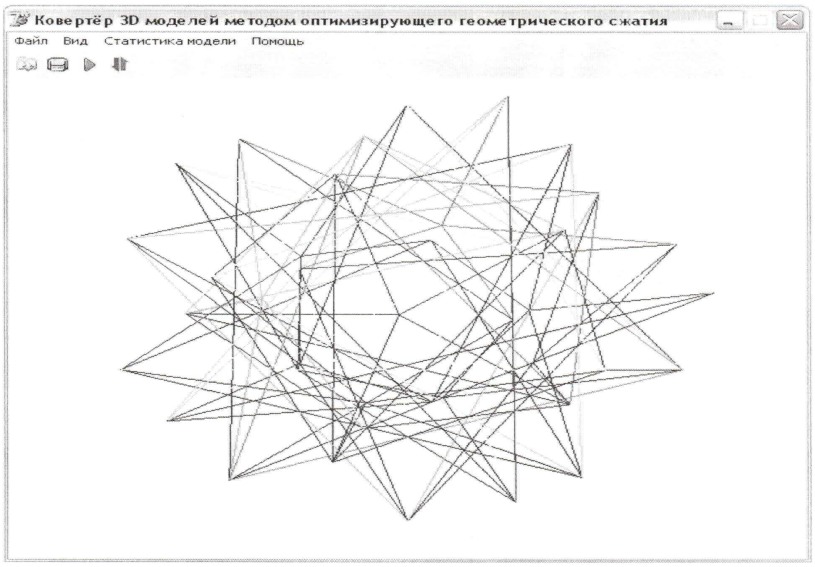

Рис. 4. Главное окно программы

Графический интерфейс ПО позволяет наглядно увидеть исходную модель до и после конвертации.