*К.т.н., доцент **Дмитриев С.Ф.**, тел. (3852) 24-76-18, e-mail:* dmitrlm@mail.ru*, Алтайский государственный университет; ст. преп., аспирант **Лященко Д.Н.**, тел. (3852) 62-83-80; e-mail:* LDN-ray@yandex.ru*, Алтайский государственный аграрный университет; инженер, аспирант **Новоженов А.В.**, тел. (3852) 48-57-14, Алтайский госуниверситет; .к.х.н., д.т.н., профессор **Ишков А.В.**, e-mail:* olg168@rambler.ru*, Алтайский государственный аграрный университет (г. Барнаул)*

УДК 004.92

ДИНАМИЧЕСКОЕ РЕГУЛИРОВАНИЕ УРОВНЯ ДЕТАЛИЗАЦИИ ТРЕХМЕРНЫХ МОДЕЛЕЙ СЛОЖНЫХ ИЗМЕРИТЕЛЬНЫХ ПРИБОРОВ

В.М. Калачев, Н.Н. Решетникова

Предложен алгоритм, который позволяет динамически регулировать уровень детализации полигональных моделей сложных 3D-объектов в интерактивных сценах. Визуальные артефакты при упрощении полигональных сеток не возникают.

Ключевые слова: 3D-объект, полигональная модель, уровень динамической детализации, интерактивность

Современные мощности персональных компьютеров позволяют решать задачи компьютерной графики на качественно новом уровне. Становится возможным моделирование и отображение на экране компьютера пространства, окружающего жизнь человека, и организация взаимодействия с виртуальным пространством. Несмотря на рост вычислительных мощностей, персональные компьютеры и даже специализированные графические станции не в состоянии отображать полноценные трехмерные изображения аналогичные тем, которые “формирует” человеческий глаз, поддерживая при этом интерактивность происходящего.

Поэтому одной из основных проблем визуализации интерактивных трехмерных сцен является производительность конкретной системы отображения виртуального пространства, так как вычисления, которые предваряют визуализацию трехмерного образа, достаточно тяжеловесны. Реализованные в полигональной технологии ограничения, в соответствии с которыми любой трехмерный объект создается из набора полигонов (треугольников, однозначно определяющих плоскость в пространстве) и наложенных на эти треугольники текстурных изображений, лишь частично решают проблему.

Постановка задачи

Одним из методов, позволяющих ускорить процесс визуализации трехмерных объектов, является представление этих объектов с различной степенью визуального разрешения или различными уровнями детализации (Level Of Detail, **LOD**) [1]. Уровень детализации есть некоторая аппроксимация начальной модели объекта с использованием меньшего количества геометрических деталей - полигонов.

Как известно качество изображения, воспринимаемое глазом, имеет свойство ухудшаться с увеличением дистанции до объекта. Именно на этом свойстве основана большая часть современных LOD алгоритмов. Исходя из текущего расположения наблюдателя, алгоритм регулирует уровень детализации поверхности модели объекта.

При реализации видо - независимого LOD алгоритма, создается специальная структура, из которой в процессе визуализации в реальном времени извлекается нужный объект с определенным количеством полигонов. Преимуществом здесь является плавность изменения детализации, а также возможность поэтапного ее увеличения. При этом уровень детализации для каждого объекта определяется точно, а не выбирается из нескольких заранее созданных вариантов, тогда и полигонов используется ровно столько, сколько требуется. Использование такого метода приводит к экономии вычислительных ресурсов.

Алгоритмическая реализация

Рассмотрим в обобщенном виде работу видо-независимого алгоритма упрощения полигональных сеток по шагам.

Шаг 1. После запуска алгоритма, выполняется проверка полигональной сетки целевого объекта на присутствие в ней полигонов, которые имеют четыре и более вершин. Если такие полигоны имеются, то алгоритм произ-

водит их триангуляцию. Для этого требуется определить, какие именно многоугольники будут подвергаться триангуляции: выпуклые или невыпуклые. Дадим определение выпуклому многоугольнику.

Пусть дан многоугольник F. Тогда следующие утверждения эквивалентны:

- все углы F меньше 180°;
- F лежит по одну сторону от любой прямой, содержащей его сторону;
- F целиком содержит любой отрезок, соединяющий две принадлежащие ему точки;

При выполнении любого из этих условий многоугольник F называется выпуклым. Ниже на рисунке 1 представлены невыпуклый (слева) и выпуклый (справа) многоугольники.

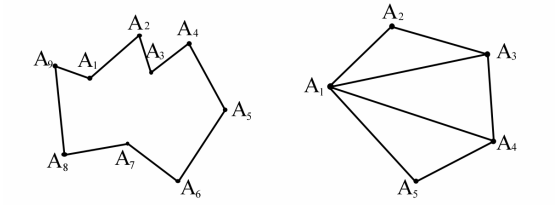

Рисунок 1 – Невыпуклый и выпуклый многоугольники

Если треугольник невыпуклый, необходимо преобразовать его в выпуклый и продолжить его триангуляцию, воспользовавшись теоремой о триангуляции полигона [2]:

Теорема: n-угольник может быть разбит на n - 2 треугольника проведением n - 3 хорд.

Если вся сетка состоит только из треугольников, этот шаг пропускается.

Шаг 2. Построение иерархического дерева вершин, которое представляет собой трехмерный полигональный объект. Дерево вершин строится заранее, на подготовительном этапе. Каждой вершине модели назначается соответствующий узел дерева вершин.

Дерево вершин строится от высоко детализированной сетки M к низко детализированной сетке M^0, записывая связи потомок - родитель в иерархическую структуру по всей поверхности объекта (рисунок 2).

Каждый узел дерева вершин хранит информацию о вершине и указатели на:

- свои два потомка;
- на родителя;
- на два списка смежных треугольников: *постоянный* и *текущий*;

Видо-независимый алгоритм упрощения полигональных сеток для изменения детализации объекта, использует локальные операции *свертывания ребра* и *разделения вершины*, представленных на рисунке 3. Операция свертывания ребра сворачивает ребро (V_b V_a) в одиночную вершину V_p, что приводит к удалению этого ребра, так же как и удалению смежных с ребром треугольников. Операция, обратная свертыванию ребра, называется разделение вершины. Каждая такая операция заменяет вершину двумя новыми вершинами, соединенными ребром, создавая при этом одну дополнительную вершину и два дополнительных треугольника.

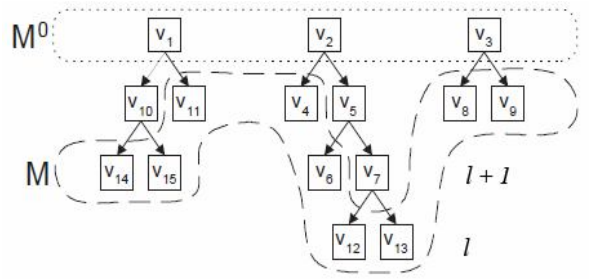

Рисунок 2 - Бинарное дерево вершин. M0 – упрощенная сетка, M – оригинальная

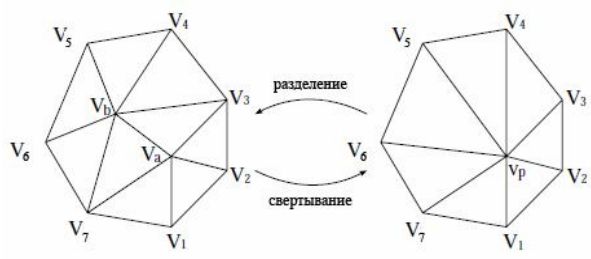

Рисунок 3 – Пример свертывания ребра / разделения вершины

Для хранения указателей на треугольники, которые удаляются после операции свертывания, используется список постоянных смежных треугольников (**PAT** - Permanent Adjacent Triangle). Для хранения указателей на текущие смежные треугольники, используется список текущих смежных треугольников (**CAT** - Current Adjacent Triangle). (рисунок 4)

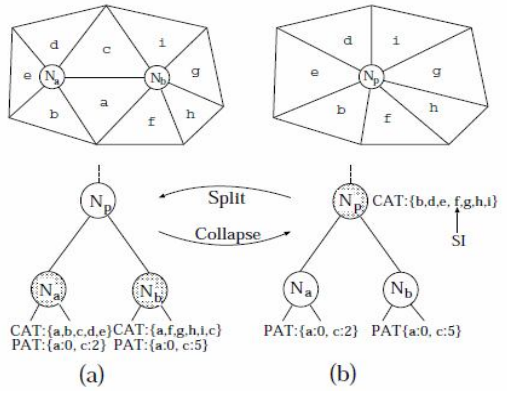

Рисунок 4 – Списки смежных треугольников после операций разделения или свертывания

Шаг 3. В результате построения ассоциативного дерева, формируется приоритетный ряд, содержащий в себе все возможные ребра, отсортированные по длине, которые могут быть безопасно свернуты при операции упрощения.

Длина ребра вычисляется с помощью определенной метрики расстояния. Для определения, какое именно ребро будет свертываться в данный момент времени, будем использовать сортировку ребер по их длине, и свертывать в первую очередь самые короткие. Свертывание ребер приведет к тому, что у соседних ребер будет изменяться длина. Поэтому, необходимо делать обновление данных и сортировку ребер после каждой операции свертывания.

Расстояние между двумя вершинами, которые предстоит объединить, можно вычислить с помощью любого приемлемого измерения расстояния между двумя точками. В нашем случае используем Евклидову геометрию. Для двух точек a = (x_a, y_a, z_a) и b = (x_b, y_b, z_b), расстояние d_{ab} выражается в следующем виде:

$$d_{ab} = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2} \qquad (1)$$

После формирования приоритетного ряда, выполняем следующее:

проверяем, может ли выбранное ребро безопасно быть свернутым;

если оно может быть безопасно свернуто, то выполняем операцию свертывания;

повторяем этот процесс до момента достижения порогового значения минимального числа треугольников, заданного пользователем;

Шаг 4. При выполнении упрощения полигональной сетки трехмерных моделей при помощи LOD алгоритмов, могут возникать визуальные артефакты, такие как свечение и разрыв в текстуре или геометрические ошибки, такие как щели в полигонах.

Появление таких артефактов связано с заворачиванием сетки. Заворачивание сетки происходит, когда смещение местоположения вершины заставляет смежный треугольник менять свою ориентацию. Эти артефакты свойственны любой схеме, в которой выполняются такие операции, как свертывание ребра. В результате свертывания ребра, треугольник может «завернуться» сам на себя или изменить положение нормали. Свертывание ребра можно считать безопасным, если это не приводит к любым заворачиваниям или длинным узким треугольникам.

Для определения безопасности свертывания ребра, будем использовать два эвристических правила:

- для любого треугольника, смежного с любой из двух сворачиваемых вершин, различие между углами нормалей этих треугольников до и после сворачивания ограничено некоторым порогом, заданным пользователем;
- для любого треугольника, смежного с любой из двух сворачиваемых вершин, качество этого треугольника не должно выпадать из заданного пользователем порога;

Определим качество треугольника с площадью a и длинами трех сторон l_0, l_1 и l_2 по формуле, предложенной Gueziec [3]:

$$c = \frac{4\sqrt{3}a}{l_0^2 + l_1^2 + l_2^2} \qquad (2)$$

Используя выражение (2), можно сделать вывод, что качество вырожденного треугольника близко к нулю, а равностороннего треугольника стремится к единице. Исключим из этих проверок треугольники, которые становятся вырожденными после свертывания ребра (смежные с этим ребром треугольники).

Безопасность свертывания или разделения может быть потеряна во время прохода по дереву иерархии вершин во время выполнения алгоритма.

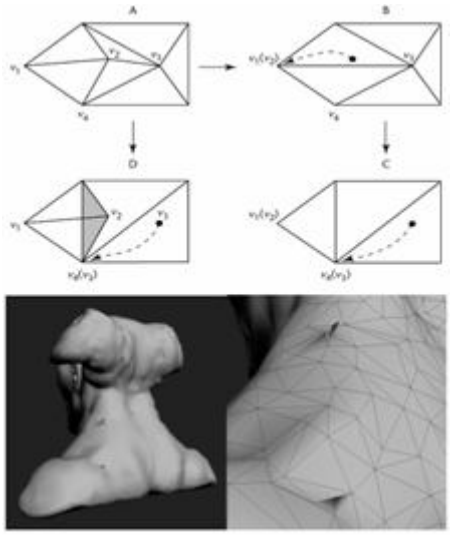

Рисунок 5 – Пример нежелательного возникновения заворачивания сетки

Рисунок 5 показывает пример, как может возникать нежелательное заворачивание в

адаптивной сетке, хотя все подходящие пары вершин, которые были определены в статическом состоянии, были правильные.На рисунке 5, пример «A» показывает начальное состояние сетки. Строя дерево вершин, в первую очередь объединяем вершину v_2 в v_1 и получаем сетку «B», а затем объединяем вершину v_3 в v_4 для получения сетки «C». Теперь предположим, что во время выполнения требуется отобразить вершины v_1, v_2 и v_4, с возможностью объединить вершину v_3 в v_4. Однако, если объединить вершину v_3 в v_4 напрямую, двигаясь из сетки «A» в сетку «D», получим заворачивание сетки в месте, где его не должно быть.

Для предотвращения таких заворачиваний во время выполнения, другие более ранние алгоритмы используют метод определенных зависимостей среди узлов иерархического дерева вершин. Определим границу свертывания, чтобы установить какие вершины формируют границу области влияния.

Определенные зависимости разрешают свертывать ребро (v_b v_a) только тогда, когда существуют все определенные вершины границы области влияния этой операции свертывания, и эти вершины являются смежными с ребром (v_b v_a). В качестве примера рассмотрим рисунок 6.

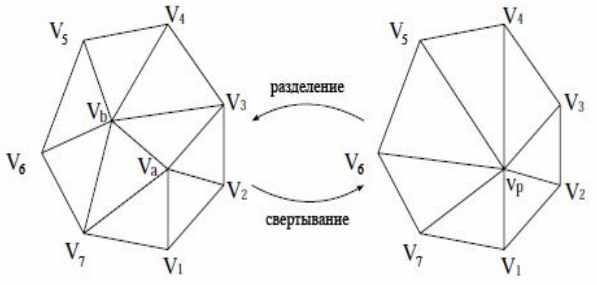

Рисунок 6 – Пример свертывания ребра / разделения вершины

Определенные зависимости могут быть выражены в определении 1:

Определение 1. Определенные зависимости для объединения двух вершин v_a и v_b : - свертывание - вершина v_a может объединиться с вершиной v_b только тогда, когда все вершины v_1, v_2, …, v_7 представлены как соседи вершин v_a и v_b ;- разделение - вершина v_p может быть безопасно разделена в вершину va и vb только если вершины v1, v_2, …, v_7 существуют и смежны с v_p. ;

Предлагается ввести новый способ определения порядка свертывания ребер и разделения вершин для предотвращения заворачивания во время выполнения упрощения. Назовем этот способ *идентификацией вершин*. Основная идея этого метода – назначение идентификаторов всем вершинам полигональной сетки, ребра которых должны быть свернуты.

Если оригинальная сетка модели объекта имеет n вершин, то этим вершинам назначаются идентификаторы (**ID**) 0, 1, …., n - 1.

Далее, каждый раз, когда в результате свертывания ребра создается новая вершина, значение ID для этой новой вершины будет назначаться на один больше, чем предыдущее самое большое значение ID существующей вершины. Этот процесс продолжается до тех пор, пока целиком не будет построено дерево вершин.

Прежде, чем операция разделения или операция свертывания выполнится, проводим две простые проверки, основанные на сравнении ID вершин. Это необходимо для обеспечения правильной безопасной последовательности произведенных операций, чтобы избежать появления заворачивания сетки.

Определение 2. Сравнение идентификаторов вершин: - свертывание ребра: ребро (a, b) может быть безопасно свернуто, если ID вершины их родителя меньше, чем ID вершины родителя сворачиваемых смежных вершин; - разделение вершины: Вершина p может быть безопасно разделена во время выполнения, если ID ее вершины больше, чем ID вершин ее соседей.

Рассмотрим пример свертывания на рисунке 7.

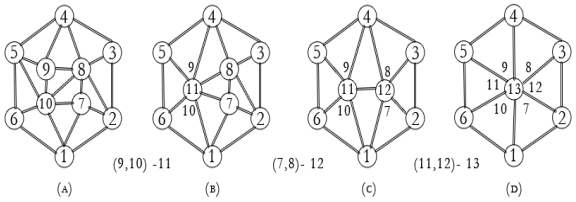

Рисунок 7 – Последовательность свертывания

Рисунок 7 (a) отображает оригинальную сетку, рисунки 7 (b), (c) и (d) показывают последовательность свертываний (v_9, v_{10}), (v_7, v_8) и (v_{11}, v_{12}) соответственно. Список определенных зависимостей для свертывания (v_7, v_8) – это E (v_7, v_8) = {v_1, v_2, v_3, v_4, v_{11}}. Это означает, что узлы смежные с v_7 и v_8 должны быть как раз элементами списка E (v_7, v_8) перед свертыванием (v_7, v_8).

Если для определения порядка свертывания применить предложенный метод идентификации вершин, то выглядеть это будет следующим образом: свертывание (v_7, v_8) мо-

жет произойти только после свертывания (v_9, v_{10}) в v_{11} и перед свертыванием любой вершины {v_1, v_2, v_3, v_4, v_{11}}.

Аналогичным образом, разделение узла v_{12} должно происходить только перед разделением узла v_{11}. При такой реализации метода идентификации вершин, будем хранить два целых числа для каждого узла, которые представляют собой максимальный идентификатор (M) смежных вершин и минимальный идентификатор (m) родителей сворачиваемых смежных (граничных) вершин.

Эти два целых числа обновляются каждый раз при изменении ребра(границы) в результате операций разделения или свертывания. В качестве примера, на рисунке 7, целая пара чисел (M ; m) узлов v_7 и v_8 для начального состояния сетки, показанного на рисунке 7 (a) имеет значение (10 ; 11), тогда как эта же пара на рисунке (b) имеет значение (11 ; 13).

Когда узел n не может быть разделен из-за того, что не удовлетворяет условиям безопасности разделения, то необходимо оставить этот узел n на время, пока узлы, которые препятствуют его разделению, не разделятся.

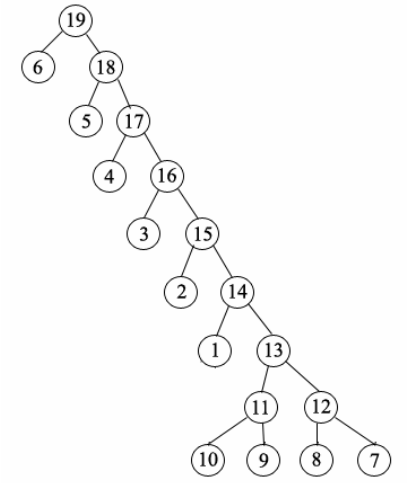

Рисунок 8 – Пример бинарного дерева вершин для полигональной сетки, представленной на рисунке 7

На рисунке 8 представлено описанное выше ассоциативное дерево вершин для примера полигональной сетки на рисунке 7.

Шаг 5. После определения всех подходящих ребер, которые можно свернуть в процессе выполнения операции свертывания, необходимо задать порядок, в котором эти ребра будут свертываться.

Вершинам n оригинальной полигональной сетки трехмерного объекта назначаются идентификаторы следующего вида : 0, 1, …., n - 1. Каждый раз, когда в результате свертывания ребра создается новая вершина, новой вершине назначается идентификатор, имеющий значение на один больше, чем значение идентификатора предыдущей вершины. Этот процесс продолжается до тех пор, пока целиком не будет построено дерево вершин. Прежде, чем выполнится операция разделения или операция свертывания, делаем несколько простых проверок, сравнивая идентификаторы вершин, что гарантирует правильную последовательность произведенных операций, которые помогут избежать заворачивания сетки.

Шаг 6. Необходимо определить некое пороговое значение, при достижении которого, будут выполняться операции свертывания ребер и разделения вершин. В качестве такого критерия, нами используется screen space error (размер трехмерного объекта на экране) [4]. Эта величина измеряется в пикселях – рисунке 9.

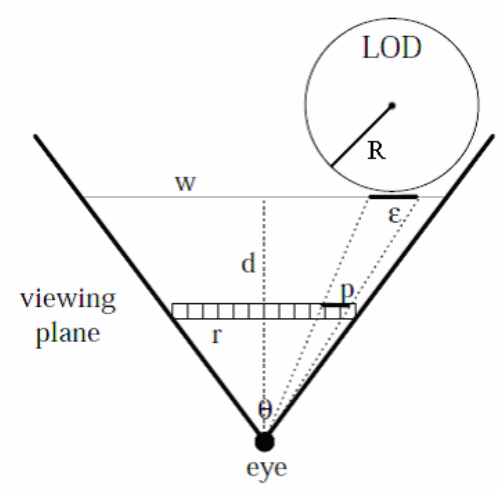

Рисунок 9 – Преобразование геометрической ошибки объекта e в экранную ошибку p

Приведем небольшой пример. Установим пороговое значение экранного размера объекта в 300 пикселей. Когда размер объекта на экране будет занимать более 300 пикселей, упрощение трехмерного объекта не выполняется, модель представлена с оригинальной сеткой. Если же, камера удалилась от объекта на такое расстояние, что размер объекта на экране стал менее 300 пикселей и лежит в диапазоне:

- От 250 до 300 пикселей, то выполнять упрощение / детализацию сетки до степени упрощения в 80% от оригинальной сетки;
- От 200 до 250 пикселей, то выполнять упрощение / детализацию сетки до сте-

пени упрощения в 60% от оригинальной сетки;

- От 150 до 200 пикселей, то выполнять упрощение / детализацию сетки до степени упрощения в 40% от оригинальной сетки;
- От 100 до 150 пикселей, то выполнять упрощение / детализацию сетки до степени упрощения в 20% от оригинальной сетки;
- Менее 100 пикселей, то выполнять упрощение / детализацию сетки до степени упрощения 10% от оригинальной сетки;

Шаг 7. Определим узлы, которые лежат на контуре полигонального объекта. Треугольники этих узлов формируют силуэт объекта, но поскольку необходимо сохранить форму упрощаемой модели, то ребра этих треугольников исключаем из кандидатов на упрощение [5].

Будем проверять каждый сворачиваемый узел дерева вершин, и если этот узел лежит на контуре упрощаемой модели, то вершины свертывать нельзя.

Если любой вектор в конусе нормалей является ортогональным к любому вектору в конусе видимости, то узел потенциально лежит на контуре объекта.

Шаг 8. Если видимый на экране полигональный объект удовлетворяет критерию экранного пространства, то выполняется операция свертывания ребра (упрощение), либо операция разделения вершины (детализация), в зависимости от положения камеры.

Особенности реализации и тестирования алгоритма

Видо-независимый алгоритм выполняется до тех пор, пока целевой объект находится в области видимости камеры. Как только объект становится невидимым, алгоритм прекращает свою работу. При повторном попадании объекта в область видимости камеры, работа алгоритма начинается с шага 5.

На рисунке 10 приведен пример полигональной модели электроизмерительного прибора, полученной в результате работы рассмотренного выше алгоритма.

Предложенный алгоритм был успешно применен при разработке моделей многочисленных приборов, интерьеров, а также при моделировании ритуальных построек и знаковых скульптур в проекте «Виртуальная реконструкция святилища Аполлона в Дельфах».

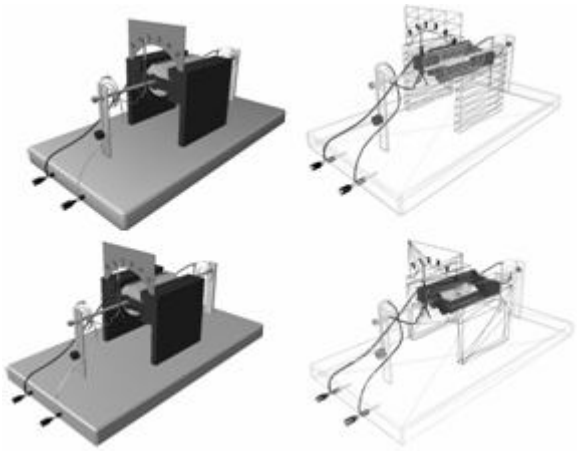

Рисунок 10 –Исходная модель (вверху) – 86 802 треугольника и упрощенная модель (внизу), состоящая из 5381 треугольника

Заключение

Следует отметить, что разработанный алгоритм, позволяет динамически регулировать уровень детализации трехмерных объектов в интерактивных сценах, тем самым, ускоряя процесс визуализации кадров, а также позволяет избегать визуальных артефактов при упрощении полигональных сеток трехмерных моделей сложных геометрических объектов.

СПИСОК ЛИТЕРАТУРЫ

1. Luebke, D. Level of Detail for 3D Graphics./ D. Luebke et al. -Morgan Kaufmann Publishers, 2003.
2. Ласло, М. Вычислительная геометрия и компьютерная графика на С++. / М. Ласло. -М.: Бином, 1997.
3. Surface simplification with variable tolerance / Guéziec. // Proc. of the Sec. Int. Symp. on Medical Robotics and Computer Assisted Surgery - MRCAS'95, 1995.
4. Шикин, А.В. Компьютерная графика. Полигональные модели./ А.В. Шикин, А.В. Боресков/. -М.: Диалог-МИФИ, 2001.
5. El-Sana, J. Generalized View-Dependent Simplification. // J. El-Sana, , A. Varshney . - Computer Graphics Forum. 1999, pp. 83-94.

*Аспирант **Калачев В.М.**, тел. +7 (905) 226-96-96, e-mail: vimika@mail.ru; к.т.н., доцент **Решетникова Н.Н.**, тел. +7 (911) 983-00-07, e-mail: reni_07@list.ru, Государственный университет аэрокосмического приборостроения (г. Санкт-Петербург).*