Визуализация полупрозрачных объектов трехмерных сцен реального времени

Коростелев Е. И., Долговесов Б. С., Мазурок Б. С.
Институт Автоматики и Электрометрии СО РАН
Новосибирск, Россия
kore3d@gmail.com

Аннотация

В статье предлагается два метода эффективной визуализации полупрозрачных объектов, использующих один проход растеризации трехмерных объектов и один либо два дополнительных прохода постобработки. Применяемые концепции разработаны с учетом специфики графических процессоров последнего и предыдущих поколений, что позволяет использовать их в системах визуализации разного уровня.

Один из методов основан на трафаретных масках и реализуется для Direct3D9-совместимых графических процессоров, а другой требует аппаратной поддержки последних технологий Direct3D11, используемых для построения динамических списков слоев прозрачности. Кроме этого, на графическом процессоре выполняется сортировка слоев прозрачности. Ее детали рассматриваются в соответствующем разделе статьи.

Ключевые слова: *порядко-независимая прозрачность, визуализация реального времени, A-буфер, сортировка на графическом процессоре.*

1. ВВЕДЕНИЕ

Одной из известных проблем компьютерной графики реального времени является визуализация полупрозрачных объектов. Полупрозрачность используется при визуализации стеклянных поверхностей, природных явлений (применяются системы частиц), а также в системах автоматизированного проектирования (САПР). В САПР полупрозрачность позволяет наглядно представить внутреннюю структуру проектируемого изделия.

Стандартным методом визуализации полупрозрачных объектов с использованием различных аппаратных решений является применение формулы альфа-смешивания пикселей растеризуемых полигонов (alpha blending):

$$dest\_rgb = dest\_rgb \cdot (1 - src\_alpha) + src\_rgb \cdot src\_alpha,$$

где dest – текущее значение в буфере цвета, а src – значение цвета смешиваемого с ним пикселя.

Результат применения представленной формулы зависит от порядка смешивания, и его нарушение приводит к появлению визуальных дефектов. Так, например, в левой части рисунка 1 нарушен порядок растеризации объектов от дальнего к ближнему, и поэтому альфа-смешивание дает неправильный результат – дальние объекты перекрывают ближние и визуально имеют неверное расположение. В итоге, на рисунке дракон визуально находится перед остальными объектами сцены, что не соответствует реальному расположению. Чтобы этого избежать, на сегодняшний день, применяют сортировку объектов в пространстве наблюдателя. Но она не эффективна, поскольку не решает проблему целиком и ресурсоемка при большом количестве сортируемых объектов. Кроме этого, она требует дополнительного разбиения пересекающихся примитивов.

Рисунок 1: Пример результата стандартного альфа-смешивания и правильного альфа-смешивания

В системах визуализации сортировка элементов сцены, как правило, производится на центральном процессоре и требует постоянного обновления упорядоченных данных в памяти видеокарты. Устранение передачи этих данных по шине возможно путем переноса вычислений на графический процессор (GPU). В этом случае, освобождаемые ресурсы центрального процессора могут быть использованы для решения более важных задач – обработка взаимодействия с пользователем, задачи искусственного интеллекта, подкачка данных и другие. Но в настоящее время сортировка на GPU объектов трехмерной сцены практически не применяется. Причиной является отсутствие эффективного решения для случая большого числа объектов.

Существуют другие подходы, позволяющие гарантировать правильный порядок смешивания. Их основой является сортировка по z-координате пикселей, участвующих в альфа-смешивании. Благодаря этому, результат визуализации не зависит от порядка растеризации полигонов и, соответственно, сортировка полупрозрачных объектов не требуется. Такие методы принято называть методами порядко-независимой прозрачности (order-independent transparency). Одним из их достоинств является корректная обработка случая пересекающихся примитивов.

Одним из методов порядко-независимой прозрачности является метод расслоения по глубине (depth peeling) [1]. Он

обладает низкой производительностью, поскольку для разделения на слои (рисунок 2) применяется многопроходная визуализация. Худшая его производительность достигается на трехмерных сценах из множества анимированных полупрозрачных объектов с различными настройками материалов. Так, использование множества параметров приводит к многократному переключению состояний графического конвейера, что существенно снижает производительность. Уменьшение ресурсоемкости в этом случае достигается путем сокращения числа используемых проходов и объема памяти, резервируемого под слои прозрачности. Оптимизации такого рода являются основой различных модификаций этого метода: dual depth peeling [2], reverse depth peeling [3] и другие [4]. Но, они, как и оригинальный метод, широко не применяются, поскольку их производительности не достаточно для современных систем визуализации.

Рисунок 2: Концепция расслоения по глубине.

В ходе развития возможностей аппаратных решений предлагались и другие методы порядко-независимой прозрачности. Часть из них использует чересстрочную растеризацию [5] для записи значения слоя прозрачности поверх изображения непрозрачных, другие – реализуют концепцию K-буфера [6,7] или A-буфера [8,9,10]. Под K-буфером подразумевается абстракция структуры памяти для хранения ровно K слоев прозрачности, а A-буфер, в свою очередь, подразумевает хранение произвольного числа слоев прозрачности. Эти методы имеют ряд серьезных ограничений: требуют современного аппаратного обеспечения и обладают низкой производительностью, обрабатывая при этом лишь небольшое число слоев.

В различных публикациях также рассматриваются методы, реализация которых в настоящий момент невозможна [11,12,13]. Это ограничение связано с отсутствием у современных графических процессоров функции атомарного исполнения произвольных блоков инструкций.

Далее, в статье предлагаются два новых метода, которые обладают неплохой производительностью и имеют меньше ограничений по сравнению с существующими. Кроме этого, они позволяют эффективно визуализировать полупрозрачные объекты, как на новом аппаратном обеспечении (на основе архитектуры AMD Terascale2 и NVIDIA Fermi [14]), так и на аппаратном обеспечении предыдущих поколений (Direct3D9-совместимые видеокарты).

2. ПОРЯДКО-НЕЗАВИСИМАЯ ПРОЗРАЧНОСТЬ

В данном разделе статьи описываются два новых метода порядко-независимой прозрачности, которые обеспечивают правильную визуализацию полупрозрачных объектов. При этом непрозрачные объекты визуализируются отдельно, и значения цвета и глубины после этого применяются для отсечения невидимых полупрозрачных пикселей и в формуле альфа-смешивания. Для этого используется чтение буфера глубины, либо отдельной текстуры, заполненной соответствующими значениями. Важной деталью обоих методов является сортировка слоев прозрачности на графическом процессоре. Ее особенности представлены в следующем разделе статьи.

2.1 Метод трафаретных масок

Основная идея метода заключается в использовании маскирования пикселей для разделения их на слои прозрачности. При этом применяется тест и буфер трафарета. В буфер предварительно записываются номера выделяемых слоев прозрачности, а на этапе визуализации выполняется проверка текущих значений на равенство нулю. Если тест пройден, то значение перезаписывается на 255, если нет – уменьшается на единицу. Таким образом, в буфер цвета попадают только пиксели, соответствующие маскам слоев прозрачности. Пиксели различных слоев при этом записываются по соседству и образуют группы. В итоге, разрешение получаемого изображения получается меньше. На рисунке 3 приводится пример его заполнения, где хорошо наблюдается наличие групп размером 4 на 4.

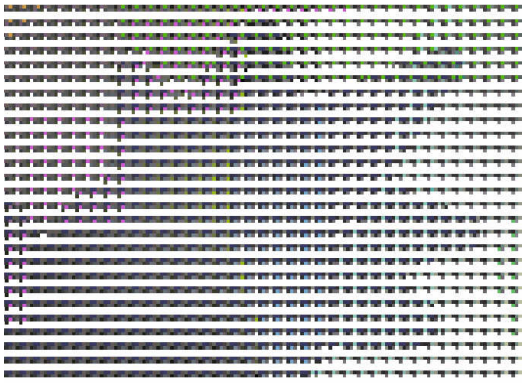

Рисунок 3: Пример заполнения буфера цвета (цвета инвертированы)

Дополнительно используется еще один буфер. В него производится запись значения z-координаты (в пространстве наблюдателя) соответствующего пикселя. Кроме этого, в реализации метода буферы представляются текстурами формата R8G8B8A8 и R16F. То есть 32 бита для 4 канального целочисленного значения цвета и 16 бит для 1 вещественного значения глубины, соответственно.

Сохранение разрешения финального изображения обеспечивается за счет использования буферов большего размера, чем размеры формируемого кадра (применяется supersampling). В этом случае, N-кратное увеличение буфера результата растеризации позволяет выделять до N слоев прозрачности без потери разрешения. Далее, отдельным проходом постобработки, выполняется его чтение и сортировка. Полученные слои прозрачности упорядочиваются по z-координате, и значения их цветов смешиваются по формуле альфа-смешивания.

Расчет слоев прозрачности по группе пикселей в общем случае приводит к появлению визуальных дефектов. Они возникают на границах полигонов, поскольку в этой области растеризатор формирует не все пиксели группы. Таким образом, происходит модификация не всех масок и выделяются не те слои прозрачности. В итоге, слои могут быть пропущены либо продублированы. Суть этой проблемы поясняется рисунком 4. Пусть левый треугольник полностью покрывает группу A и частично B, D, E, а правый – полностью E, F и частично B. В этом случае в группах B, D, E при растеризации будут модифицированы не все маски, что приведет к неверному альфа-смешиванию и визуальному дефекту.

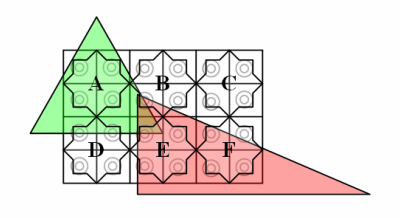

Рисунок 4: Пример растеризации треугольников

Устранение описанных дефектов производится путем фильтрации областей финального изображения, где потенциально могло произойти неправильное альфа-смешивание. Нахождение таких мест производится проверкой трех условий: разница суммы глубины по всем слоям прозрачности меньше некоторого константного значения, совпадение четности числа слоев прозрачности, расстояние в кубе RGB цветов меньше некоторой константы. При невыполнении любого из них соответствующий пиксель помечается некоторым значением ошибки (например, 0.25, 0.75 или 0.5, соответственно) Пример выделения таких областей приводится на рисунке 5 (линии на границах).

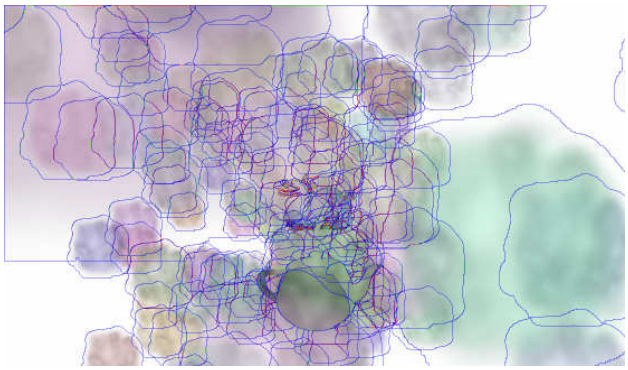

Рисунок 5: Пример маркирования потенциально дефектных пикселей (используется альфа-тест)

Используемый алгоритм фильтрации заключается в анализе изображения и обработке помеченных текселей модифицированным фильтром Гаусса. Применяемая модификация заключается в выборе для усреднения из 12 ближайших соседей только текселей, имеющих минимальное значение ошибки. Не соответствующие этому условию соседи могут использоваться с меньшими коэффициентами.

2.2 Метод динамических списков слоев прозрачности

Рассматриваемый далее метод является более универсальным по сравнению с методом трафаретных масок. Он не требует дополнительной фильтрации и использует меньше памяти под хранение слоев прозрачности. Экономия памяти достигается благодаря построению динамических списков поверх линейной памяти. Ее заполнение выполняется на этапе визуализации полупрозрачных объектов и реализует концепцию A-буфера [8]. Далее, отдельным проходом постобработки, элементы каждого списка сортируются и используются для расчета правильного альфа-смешивания значений цветов.

Общая схема работы алгоритма построения динамических списков описывается следующими шагами:

- Атомарно увеличить счетчик занятой памяти.
- Атомарно обновить текущую ячейку буфера индексов начал списков предыдущим значением счетчика.
- Записать в память цвет, глубину и адрес следующего элемента, который соответствует предыдущему значению в текущей ячейке буфера индексов начал списков.

Выполнение этих шагов гарантирует запись данных текущего слоя в незанятую ранее область памяти и приводит к построению связных списков. В результате, начало формируемых списков всегда лежит по адресу, указанному в буфере индексов начал списков, а последний элемент списка хранит значение, которым был инициализирован буфер.

В качестве оптимизаций, используется один дополнительный буфер для хранения длины списка и память под требуемые буферы выделяется на видеокарте в виде объектов 2D-текстур. Для доступа на запись используется технология UAV (unordered access view), входящая в состав DirectX версии 11. Кроме этого, для синхронизации операций между потоками, применяются атомарные операции.

Использование метода на современном аппаратном обеспечении позволяет эффективно обрабатывать до 64 слоев прозрачности. В большинстве случаев этого более чем достаточно.

3. СОРТИРОВКА НА GPU

В этом разделе рассматриваются методы сортировки за один проход на графическом процессоре большого количества коротких последовательностей данных. Обрабатываемые последовательности имеют длину не более 64 и хранятся в памяти видеокарты. Другие длины не рассматриваются, поскольку для задачи сортировки слоев прозрачности этого достаточно и при большем N более предпочтительна сортировка в несколько проходов постобработки [15].

Решение задач на графическом процессоре имеет свою специфику. Так, например, потоки выполняются на симметричных процессорах группами, и потоки в группе выполняют одну инструкцию [14]. В связи с этим, операция ветвления является одной из дорогих. Поэтому предпочтительно использование слабоветвящихся алгоритмов. Кроме этого, особенности архитектуры не позволяют реализовывать рекурсивные вызовы. Все эти факторы влияют на выбор эффективного алгоритма.

Проводилось тестирование производительности различных алгоритмов сортировки: вставками (insertion sort), Шелла с $h = 2^p3^q$ (Shell sort), битоническая (bitonic sort), нечетно-четная слиянием Бэтчера (odd-even merge sort) и сортирующая сеть [16,17]. Сортирующие сети, путем взаимодействия компараторов (блоков сравнения-обмена), позволяют сортировать последовательности некоторой ограниченной длины. В качестве такой сети в тестировании используется цепочка компараторов, получаемая путем развертки алгоритма нечетно-четной сортировки слиянием Бэтчера. Максимальная длина сортируемой последовательности при этом фиксируется некоторым N. Пример такой сети представлен на рисунке 6.

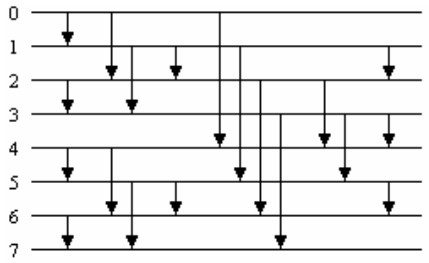

Рисунок 6: Сортирующая сеть для 8 значений

Как известно, алгоритмы Шелла с $h = 2^p3^q$ и битонической сортировки дают эффективные сортирующие сети. Но они, по сравнению с используемой, имеют большее число компараторов. Так, например, битоническая имеет $S(2^k) = 2^{k-2}\cdot k\cdot(k+1)$ компараторов, что больше нечетно-четной сети Бэтчера – $S(2^k) = 2^{k-2}\cdot k\cdot(k-1)+2^k-1$.

В таблице 1 приводятся результаты измерения времени работы алгоритма на видеокарте NVIDIA GeForce 8600 MGT. Компиляция программ для видеокарты выполнялась в рамках шейдерной модели третьей версии. В качестве данных для сортировки использовались последовательности длины от 0 до N, заданные стандартным генератором псевдослучайных чисел. Общее число последовательностей при этом составляло 65536. По полученным результатам видно, что упорядочение сортирующей сетью наиболее эффективно.

Сортировка	N = 4	N = 8	N = 16	N = 32
Вставками	3 мс	13 мс	58 мс	296 мс
Шелла ($h = 2^p3^q$)	6 мс	61 мс	480 мс	3887 мс
Битоническая	4 мс	34 мс	229 мс	1392 мс
Нечетно-четная слиянием	4 мс	30 мс	198 мс	1214 мс
Сортирующая сеть	1 мс	2 мс	4 мс	17 мс

Таблица 1: Производительность методов сортировки 65536 N-элементных последовательностей на GeForce 8600 MGT

В таблице 2 приводятся данные о производительности рассматриваемой сортирующей сети. Производительность указана в элементах в секунду и верна для последовательностей длины N. Данная таблица позволяет приближенно оценить вклад сортировки в общее время работы предлагаемых методов визуализации полупрозрачности.

Сортирующая сеть компараторов с N входами	Производительность на GeForce 8600 MGT
N = 4	1786 Мэлемент/сек
N = 8	608 Мэлемент/сек
N = 16	337 Млемент/сек
N = 32	133 Мэлемент/сек

Таблица 2: Производительность сортировки сетью компараторов N-элементных последовательностей

Для окончательной оценки производительности сортировок и выбора оптимальной для GPU приводится дополнительный график (рисунок 7). Отраженные на графике результаты получены на той же видеокарте и показывают зависимость времени (в мс) сортировки от числа сортируемых элементов в последовательности (определяется значением N). Последовательность при этом ограничена 32 элементами и число сортируемых последовательностей равно 65536. Верхнее ограничение на длину последовательности задается для фиксирования числа используемых при расчетах регистров вычислительных блоков видеокарты.

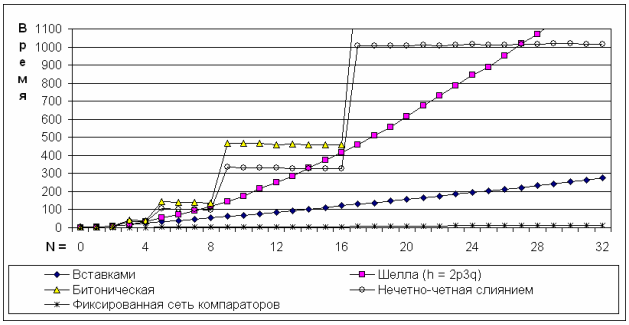

Рисунок 7: Сравнение методов сортировки N элементов из 32-элементных последовательностей

По результатам тестирования предпочтение отдается методу упорядочения сортирующей сетью, построенной алгоритмом нечетно-четной сортировки слиянием Бэтчера. Далее этот метод используется перед этапом альфа-смешивания, чтобы упорядочить слои прозрачности.

4. РЕЗУЛЬТАТЫ

Тестирование методов проводилось на сценах, состоящих из множества полупрозрачных объектов и имеющих большое число слоев прозрачности. В качестве тестовых аппаратных платформ использовались: компьютер с видеокартой ATI Radeon HD5670 и лэптоп с видеокартой NVIDIA GeForce 8600 MGT. Демонстрационные приложения запускались в оконном режиме и разрешении 1680x1050, а их производительность измерялась под операционной системой Microsoft Windows 7 Professional x64. Пример результата работы приложения приводится на рисунке 8.

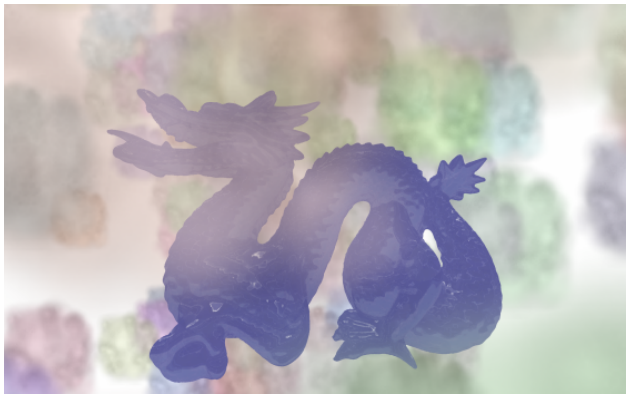

Рисунок 8: Визуализация полупрозрачных объектов с использованием трафаретных масок

Для сравнения на рисунке 9 приводится результат визуализации полупрозрачных объектов путем применения стандартного альфа-смешивания. При этом он является некорректным – объекты визуально располагаются неверно.

Рисунок 9: Результат стандартного альфа-смешивания

4.1 Производительность метода трафаретных масок

Тестирование метода проводилось на сцене из 384 текстурированных треугольников с моделями прозрачного дракона из 871318 треугольников и непрозрачного чайника из 2256 треугольников. Число слоев прозрачности – от 0 до 15.

В таблице 3 приводится статистика производительности при разном числе обрабатываемых слоев прозрачности. Результат соответствует описанной сцене. Кроме этого, значение Мпикс/сек характеризует количество полупрозрачных пикселей, обрабатываемых за одну секунду. Графа таблицы с прочерком – метод стандартного альфа-смешивания.

Число слоев	Кадров в секунду	GeForce 8600 MGT
-	40	155 Мпикс/сек
8	11	41 Мпикс/сек
16	5	19 Мпикс/сек

Таблица 3: Производительность метода трафаретных масок

4.2 Производительность метода динамических списков слоев прозрачности

Тестирование метода динамических списков слоев прозрачности проводилось на сцене из 2048 треугольников с RGBA-текстурами с моделью прозрачного робота из 261431 треугольников и непрозрачной моделью чайника из 2256 треугольников. Число слоев прозрачности на пиксель при построении изображения находилось в диапазоне от 0 до 57. Статистика производительности метода приводится в нижеследующей таблице. При этом графа с прочерком соответствует методу стандартного альфа-смешивания.

Число слоев	Кадров в секунду	Radeon HD 5670
-	210	1661 Мпикс/сек
8	25	165 Мпикс/сек
16	21	161 Мпикс/сек
32	13	102 Мпикс/сек
64	7	55 Мпикс/сек

Таблица 4: Производительность метода динамических списков слоев прозрачности

5. ЗАКЛЮЧЕНИЕ

В статье рассмотрена проблема визуализации полупрозрачных объектов трехмерных сцен реального времени, используемых в современных системах виртуальной реальности и системах автоматизированного проектирования. Предложено два новых метода визуализации, которые могут быть легко встроены в существующий цикл визуализации и могут применяться на видеокартах разных поколений.

6. СПИСОК ЛИТЕРАТУРЫ

[1] C. Everitt, “Interactive order-independent transparency”, Technical report, NVIDIA Corp., 2001.

[2] L. Bavoil, K. Myers, “Order independent transparency with dual depth peeling”. Technical report, NVIDIA Corp., 2008.

[3] N. Thibieroz, “Robust Order-Independent Transparency via Reverse Depth Peeling”, ShaderX6, pp. 211–226.

[4] Meng-Cheng Huang Fang Liu et al., “Efficient depth peeling via bucket sort”, Proceedings of the Conference on High Performance Graphics 2009, pp.51-57.

[5] D. Pangerl, “Deferred Rendering Transparency”, Article 2.7, ShaderX7.

[6] K. Myers, L. “Bavoil, Deferred Rendering using a Stencil Routed K-Buffer”, ShaderX6, 2008, pp. 189–198.

[7] Meng-Cheng Huang Fang Liu et al., “Multi-Fragment Effects on the GPU using Bucket Sort”, Article 8.1, GPU PRO, AK Peters, 2010.

[8] L. Carpenter, “The A-buffer, an antialiased hidden surface method”, Proceeding of the 11th annual conf. On Computer graphics and interactive techniques, 1984, pp.103-108.

[9] K. Myers, L. Bavoil, “Stencil routed A-Buffer”, ACM SIGGRAPH Technical Sketch, 2007.

[10] C. Pepper, “Prefix sum pass to linearize A-buffer storage”, Patent, Microsoft Corp., 2006.

[11] B.-Q. Liu, L.-Y. Wei, Y.-Q Xu, “Multi-Layer Depth Peeling via Fragment Sort”, Tech report, Microsoft Research Asia, 2006.

[12] L. Bavoil et al., “Multi-Fragment Effects on the GPU using the k-Buffer”, Proceedings of the 2007 symposium on Interactive 3D graphics and games, 2007, pp.97-104.

[13] D. Pangerl, “ZT-Buffer Algorithm”, Article 2.8, ShaderX5: Advanced Rendering Techniques, Wolfgang Engel, Ed., Charles River Media, 2007, p.151–157.

[14] NVIDIA, “NVIDIA’s Next Generation CUDA Compute Architecture: Fermi”, 2010.

[15] Kipfer, R. Westermann, “Improved GPU Sorting”, Chapter 46, GPU Gems II.

[16] D. Knuth, Volume 3: Sorting and Searching, The Art of Computer Programming, Addison-Wesley, 1997, pp.219–247.

[17] R. Sedgewick, “Sorting Algorithms”, Algorithms in C++ (2nd edition), Addison-Wesley, 1992, pp. 93–192

Real-time rendering of semi-transparent objects

Abstract

The article proposes two techniques for efficient rendering of semi-transparent objects by sinlge-pass rasterization of 3d objects and one or two additional stages of postprocessing. Applied concepts were developed with a glance to features of the latest and previous generations of GPU. This fact makes it possible to use these concepts in different virtual reality systems.

One of these methods is based on masking by stencil test and can be applied with Direct3D9-compatible videocards, while the other one requires a hardware support of the latest technologies used to create dynamic lists of transparent pixels. In addition, methods use the gpu for sorting multiple layers of semi-transparent pixels. This aspect is described in the relevant section of the article.

Keywords: *order-independent transparency, real-time rendering, A-buffer, gpu sorting.*

About the authors

Evgeny I. Korostelev is a Ph.D. student at Institute of Automation and Electrometry SB RAS. His contact email is kore3d@gmail.com

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

Boris S. Mazurok is a scientific researcher of Synthesizing Visualization Systems Laboratory at Institute of Automation and Electrometry SB RAS. His contact email is boris@albatros.iae.nsk.su