УДК 004.925

Р.В. Мальчева, канд. техн. наук, доц.,
М. Юнис, аспирант
ГВУЗ «Донецкий национальный технический университет», Украина
raisa@cs.dgtu.donetsk.ua

Повышение производительности FPGA-базированной системы визуализации в реальном времени

В данной работе выполнен анализ архитектурных решений, направленных на повышение быстродействия стадии поиска пересечения луча с объектом в системах синтеза изображений методом трассировки лучей. Обоснована целесообразность применения FPGA технологии для построения высокопроизводительных многопроцессорных графических систем. Описана FPGA-базированная система реального времени, позволяющая генерировать изображения методом трассировки лучей. Предложена модификация системы с целью повышения ее производительности.

Ключевые слова: трассировка лучей, FPGA, реальное время, интерполяция.

Введение

Алгоритм трассировки лучей – один из методов геометрической оптики, позволяющий проводить исследования оптических систем путем отслеживания взаимодействия отдельных лучей с поверхностями. В более узком значении, это технология построения изображений трехмерных моделей в компьютерных программах, использующая отслеживание обратной траектории распространения луча (от экрана до источника) [1].

Основная идея метода трассировки лучей достаточно проста: для каждого фрагмента окна, соответствующего пикселю экрана, движок рендеринга проводит прямой луч от центра системы наблюдателя (камеры) и анализирует его пересечения с элементами сцены. При наличии таковых, параметры ближайшего пересечения используются для определения цветовых компонент пикселя. Однако, только поиска пересечения недостаточно для создания реалистичного образа сцены. Необходимо определить освещение и затенение пикселя, что требует проведения вторичных лучей (в отличие от первичных лучей, определяющих видимость объектов сцены). Кроме того следует учитывать характеристики отражения и преломления материала. Другими словами, необходимо знать, какое количество света отражается в точке пересечения первичного луча, а также количество света, проходящее через материал в этой точке. В результате получается несколько типов лучей. Первичные лучи применяются для определения видимости объекта. А вторичные лучи подразделяются на лучи тени / освещения, лучи отражения и лучи преломления.

Метод трассировки лучей позволяет использовать задание сцены поверхностями в математической форме, в том числе и криволинейными. Таким образом, задание построения изображения любой поверхности решается точно, а не приближенно, как это происходит при аппроксимации поверхности, например, набором треугольников. Безусловным преимуществом метода также является качественная обработка эффектов прозрачности. Кроме того, можно получать реалистичные мягкие тени, а также поддерживать все методы закрашивания Фонга [1].

Постановка проблемы исследований

Метод трассировки лучей имеет и недостатки. Формируемые поверхности не имеют текстуры (шероховатостей) подобно реальным объектам, они гладкие. Для того, чтобы это компенсировать, дополнительно используются текстуры или bump-mapping, который обрабатывает поверхность объекта по заданному образцу, чтобы гладкая поверхность выглядела шероховатою и неоднородною.

Но главным недостатком метода трассировки лучей является его производительность. В то время, как метод растрирования или метод сканирующей строки использует когерентность данных для распределения вычислений, метод трассировки лучей каждый раз начинает процесс определения цвета пикселя с начала, рассматривая каждый луч отдельно. При этом для отображения сцены длительностью 1 кадр и размерностью N на N пикселей требуется выполнить $N \times N$ однотипных операций рекурсивного расчета цвета пикселя, что, в идеале, требует применения $N \times N$ графических процессоров для рендеринга сцены в

реальном времени [1]. Поэтому существуют различные алгоритмы и методы ускорения трассировки лучей.

Задачей данных исследований является выбор аппаратурной платформы и модернизации системы визуализации с целью повышения ее производительности.

Анализ существующих решений

Реализация трассировки лучей на SIMD

Single Instruction Multi Data (SIMD) архитектура используется для реализации метода трассировки лучей при помощи алгоритмов и команд пакетной обработки данных, т.е. лучи обрабатываются группами, а не последовательно (отдельно), специальными пакетными командами [2]. К сожалению, такой подход является эффективным только для сцен средней сложности и высоко когерентных лучей. Пакеты некогерентных лучей штрафуются, потому что лучи могут быть неактивными во время обработки. В этом случае все Ns операций выполняются с каждой командой (где Ns – ширина SIMD), и только небольшая часть этих операций (n < Ns) совершает «полезную» работу. Остальные слоты SIMD инструкций (Ns - n) выполняют «лишние» операции, которые б не выполнялись в системе, обрабатывающей лучи последовательно. В зависимости от свойств пакета лучей, SIMD обработка может привести к низкой загрузке системы. В наихудшем случае, если только один луч является «активным», эффективность системы равна 1/ Ns.

Для повышения загрузки системы в Университете шт. Юта разработан алгоритм, который использует неявное упорядочивание потоков лучей при помощи континуального уплотнения лучей в подпотоки во время операций, таких как пересечение и затенение [3].

Реализация трассировки лучей на MIMD

Multiple Instruction stream, Multiple Data stream (MIMD) архитектура включает несколько процессоров, которые функционируют асинхронно и независимо, выполняя разные команды над различными частями данных. Обработка разделена на несколько потоков, каждый с собственным аппаратным процессором, в рамках единого программного обеспечения процесса или в рамках множественных процессов. Поскольку система имеет несколько потоков, которые ожидают выполнения (системные или предназначенные для пользователя потоки), эта архитектура эффективно использует аппаратные ресурсы. Существует две основные категории параллельной реализации алгоритмов трассировки лучей, основанных на MIMD архитектуре: использование алгоритмов разделения изображения и алгоритмов объектно – пространственного деления [4].

В алгоритмах разделения изображения пиксели результирующего изображения распределяются между процессорами таким образом, чтобы каждый процессор отвечал только за трассировку «своих» пикселей. При этом база данных сцены должна быть доступна для всех процессоров, т.е. необходима общая память. Следует заметить, что несколько процессоров могут реализовывать такую схему достаточно хорошо без применения протоколов доступа к общей памяти в случае, если компоненты сцены будут направлены на все узлы при инициализации. Главным недостатком такого подхода являются большие аппаратные затраты. Однако, это решение может быть реализовано на аппаратуре общего назначения, что является достоинством в коммерческой среде.

Объектно – пространственное разделение является развитием предыдущего подхода. Каждый процессор отвечает за все трассирования лучей в рамках пространственной ячейки (кластера), а объекты базы данных распределяются между процессорами соответственно пространственному разделению [4]. Все лучи, попадающие в кластер, проверяются на пересечение с объектами, находящимися в этом кластере. Лучи, покидающие кластер, попадают на обработку к другому процессору, который отвечает за соответствующий кластер.

Не смотря на то, что разделение и дублирование памяти не являются проблемой, несбалансированные нагрузки, увеличение количества лучей и фрагментация объектов создают проблемы для этого подхода.

Реализация трассировки лучей на графических процессорах

Вычисления на GPU развивались и развиваются стремительно. Компанией NVIDIA, являющейся одним из основных производителей видеочипов, разработана платформа под названием CUDA (Compute Unified Device Architecture). Платформа CUDA – C-подобный язык программирования со своим компилятором и библиотеками для организации вычислений на GPU с учетом прямого доступа к аппаратным возможностям GPU [5].

Высокая производительность GPU объясняется особенностями его архитектуры (до 512 вычислительных ядер, например, «Fermi») и использованием массивно параллельных вычислений. В скорости доступа к видеопамяти GPU также имеют значительное превосходство.

Аналогичные результату в разработке и реализации алгоритма трассировки лучей показывает корпорация AMD ATI. Они создали AMD Cinema 2.0 (на основе движка OTOY), который реализует трассировку лучей в реальном времени на графических процессорах Radeon и центральных процессорах AMD Phenom [6].

Реализация трассировки лучей на FPGA

В Саарландском университете (Германия) создана система SaarCOR на базе FPGA [7]. Пример ее структуры для генерации сфер приведен на рис.1 [8]. Трассировщик лучей (Ray Tracer) представляет собой микропрограммный автомат.

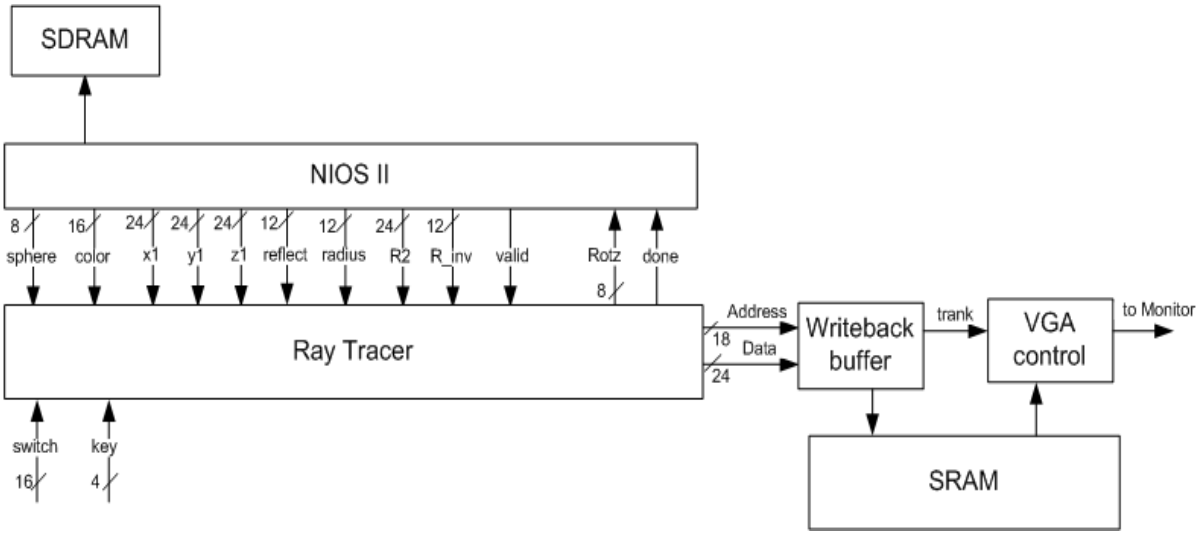

Рисунок 1 – Структура системы SaarCOR

Он отвечает за расчет цвета всех пикселей и записывает результаты в буфер записи (Writeback buffer), который согласован с VGA контроллером и имеет доступ к памяти SRAM.

Трассировки лучей аппаратно сопряжен с центральным процессорным элементом NiosII через таблицу сфер. Он использует одно состояние в микропрограмме управления, во время которого происходит считывание данных из центрального процессора.

Для взаимодействия между программой (написана на С), работающей на процессоре NIOS II, и оборудованием FPGA используются следующие выходы:

- *sphere* [7:0] используется для контроля общего количества сфер и сферы, которая в настоящее время обрабатывается; значение старших 4-х бит (устанавливается при инициализации программного обеспечения) задает число сфер плюс один источник света; младшие 4 бита устанавливаются каждый раз, когда новая сфера передается в Трассировщик;

- *color* [15:00] указывает цвет сферы - по 5 бит для каждой RGB компоненты, бит 16 устанавливается, если сфера является источником света; такая дает хороший баланс между диапазоном цветов и объемом памяти, необходимой для их хранения;

- *x1* [23:00], *y1* [23:00], *z1* [23:00] – трехмерная координата центра текущей сферы; представлены в формате с фиксированной запятой (12 бит для целой и 12 бит для дробной части); при передаче из программного обеспечения для согласования с форматом, используемым в Ray Tracer, координаты умножаются 4096 (2^{12});

- *reflect* [11:00] - отражательная способность сферы (12-битное целое);

- *radius* [11:00] - радиус сферы (12-битное целое); для облегчения взаимодействия между NIOS II и FPGA только целая часть радиуса сохраняется в памяти;

- *R2* [23:00] - квадрат радиуса сферы (24-битное целое); его вычисление в программной части экономит аппаратные средства; если радиус остается постоянным, то значение необходимо рассчитывается только один раз при инициализации блока;

- *R_inv* [11:00] обратный радиус сферы (12-битное целое после умножения на 4096);

- *valid* - бит достоверности, остается в состоянии 1 до тех пор, пока следующая сфера в списке не будет готова к отправке; после этого программа сбрасывает бит достоверности и устанавливает данные.

Входы:

- *Rotz* [7:0] - устанавливает различные режимы для вращения всех сфер вокруг осей x, y, z, что позволяет аппаратно контролировать вращение вокруг начала координат, чтобы показать различные сцены;

- *done* – бит завершения обработки: находится в состоянии 1 после команды «сброс» или в конце каждого кадра; только при наличии высокого уровня на этом входе программное

обеспечение последовательно посылает новую информацию для следующего кадра.

VGA контроллер отвечает за выборку пикселей из SRAM, синхронизацию и создание передней и задней видеостраниц, необходимых для VGA дисплеев. Система реализована для разрешений 320 × 240 и 512 × 480.

Также имеются входные переключатели и кнопки, которые используются для настройки оборудования и перемещения положений наблюдения, источника света и положения экрана.

Работая на частоте 90 МГц, эта аппаратная реализация алгоритма трассировки лучей достигает в реальном времени частоту от 20 до 60 кадров в секунду при широком диапазоне 3D сцены с поддержкой текстурирования, несколько источников света, и несколько уровней отражения или прозрачности. Особенностью разработки является повторное использование блока трансформации для задач, которые включают в себя эффективный поиск пересечения лучей с треугольниками. Несмотря на дополнительную поддержку динамических сцен, такой подход реализации снижает общую стоимость аппаратной части на 68% [7].

Предлагаемая модификация системы

Основная идея модификации системы состоит в добавлении блока межпиксельной интерполяции (Interpixel interpolation). Структура система приведена на рис.2. При этом Ray Tracer будет трассировать меньшее количество пикселей, чем разрешение окончательного изображения, с некоторым шагом. Цветовые значения непрослеженных пикселей будет формировать блок Interpixel interpolation, применяя линейную [9] или блочную интерполяцию. Шаг трассировки может варьироваться в зависимости от значения коэффициента максимального различия в цвете трассируемых пикселей [10].

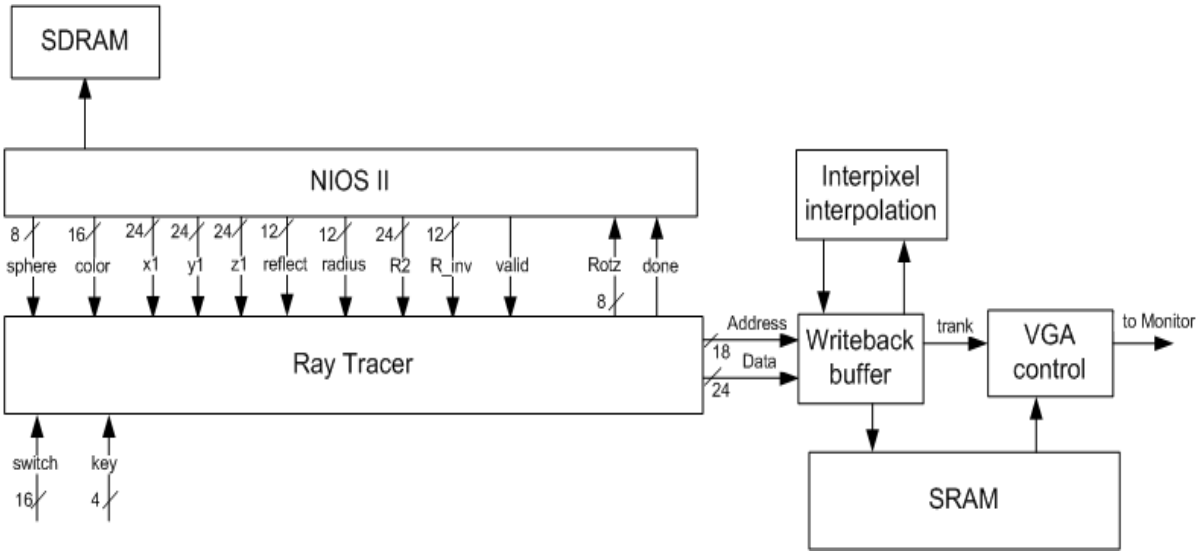

Рисунок 2 – Предлагаемая модификация системы

Моделирование модифицированного алгоритма показало уменьшение требуемого количества параллельно работающих трассировщиков на 12-15% при линейной и до 20% при применении блочной интерполяции.

Аппаратурная реализация алгоритма строчной межпиксельной интарполяции рассмотрена в [10].

Функциональная схема устройства для реализации алгоритма блочной межпиксельной интерполяции приведена на рис.3. Размер блока - 16 пикселей (4 × 4). Слева показано, что угловые пиксели блока трассируются в процессорных элементах (PE) Трассировщика. Назначение основных регистров:

- R00-R11 - 24-битные регистры; содержат значения пикселей, трассируемых в текущий момент; загрузка регистров производиться по сигналу RegLD при установленном сигнале BusRecvCtrl;

- Ra0-Ra7 - 24-разрядные регистры; хранят значения внутриблочных пикселей, на случай неудачной интерполяции;

- R0001-R1100 - 24-разрядные регистры; содержат результат вычисления интерполированного значения;

- px0-px15 - 24-битные регистры; содержат информацию о текущем состоянии блока;

- InterpolOK – флаг, единичное значение которого означает, что интерполированные пиксели имею цветовые различия в пределах заданного коэффициента расхождения в цвете; устанавливается на основе значений inOK0-inOK5.

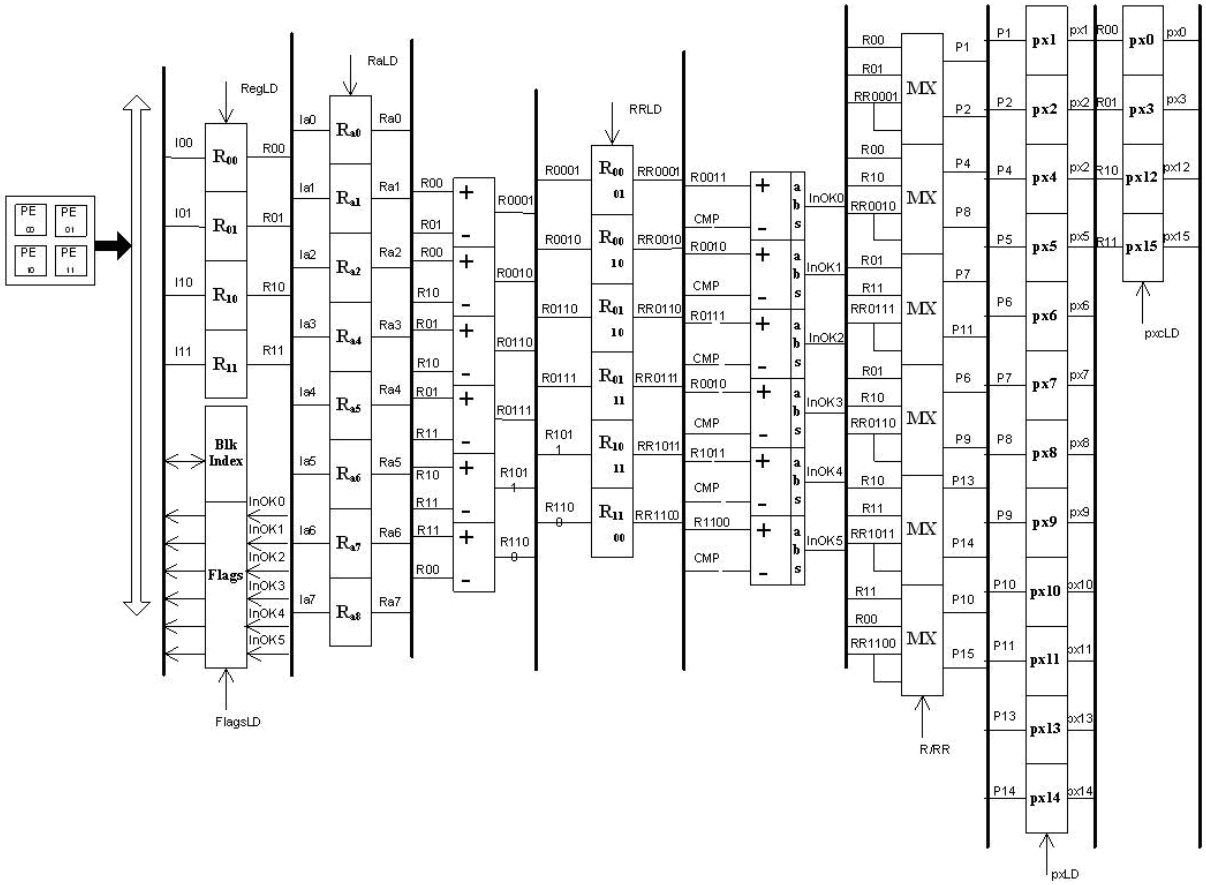

Рисунок 3 – Функциональная схема устройства блочной межпиксельной интерполяции

Заключение

В данной работе выполнен анализ архитектурных решений при построении графических систем на базе метода трассировки лучей. Выполнен поиск прототипа системы, реализованной на FPGA, и дано описание системы SaarCor.

Предложена модификация системы путем введения шага трассирования лучей на основе коэффициента различия в цвете и выполнения строчной или блочной межпиксельной интерполяции. Разработана

Направлением дальнейших исследований является анализ ресурсов FPGA, необходимых для реализации блока интерполяции.

Список литературы

1. Foley J. Computer Graphics. Principles and practice / J. Foley, A. Dam. - [2nd ed.]. - In C. - AWPC, 1997. – 1175 pp.
2. SIMD Ray Stream Tracing / [Ingo Wald, Christiaan P. Gribble, Solomon Boulos, Andrew Kensler]; University of Utah, 2007. – P. 1.
3. Интерактивная трассировка лучей с использованием SIMD инструкций INTEL [Електроний ресурс] / Intel. – 2009. – Режим доступа: http://software.intel.com/ru-ru/articles/ interactive-ray-tracing/.
4. Humphreys Greg. TigerSHARK: A Hardware Accelerated Ray-tracing Engine / Greg Humphreys, C. Scott Ananian. - Princeton University, 2005. – P. 3.
5. Программный ускоряющий движок NVIDIA OptiX [Електронний ресурс]. – Режим доступа: http://www.nvidia.ru/object/optix_ru.html.
6. AMD Cinema 2.0: новое слово визуализации [Електронний ресурс]. – Режим доступа: http://www.thg.ru/technews/20080619_110000.html.
7. Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip / [Jörg Schmittler, Sven Woop, Daniel Wagner, Wolfgang J. Paul, and Philipp Slusallek] // Computer Science; Saarland University. - Germany, 2004. – P. 8.

8. Yunfan Zhang. FPGA Ray Tracer [Електронний ресурс] / Yunfan Zhang. – Режим доступа: http://www.eeweb. com/project/yunfan_zhang/fpga-ray-tracer.
9. Мальчева Р.В. Аппаратурная реализация алгоритма строчной межпиксельной интерполяции / Р.В. Мальчева, А.А. Баркалов, М. Юнис // Радіоелектронні і комп'ютерні системи. – Харків, «ХАІ», 2012. - № 6 (58). - С. 70-74.
10. Мальчева Р.В. Исследование влияния шага трассирования лучей и коэффициента различия в цвете на время выполнения формирования изображения / Р.В. Мальчева, М. Юнис, А. Джамиль // Наукові праці ДонНТУ. Серія «Інформатика, кібернетика та обчислювальна техніка». – 2011. – Вип. 14(188). – С. 195-201.

Надійшла до редакції 10.04.2013

Р.В. МАЛЬЧЕВА, М. ЮНІС
ДВНЗ «Донецький національний технічний університет»

ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ FPGA-БАЗОВАНОЇ СИСТЕМИ ВІЗУАЛІЗАЦІЇ У РЕАЛЬНОМУ ЧАСІ

Виконаний аналіз архітектурних рішень, які спрямовані на прискорення стадії пошуку перетину променя з об'єктом у системах синтезу зображень методом трасування променів. Обгрунтована доцільність застосування FPGA технології для побудови багатопроцесорних графічних систем. Описана FPGA-базована система реального часу, яка дозволяє створювати зображення методом трасування променів. Запропонована модифікація системи з метою підвищення її продуктивності.

Ключові слова: трасування променів, FPGA, реальний час, інтерполяція.

R.V. MALCHEVA, M. YUNIS
Donetsk National Technical University

INCREASE IN PRODUCTIVITY OF A FPGA-BASED REAL-TIME VISUALIZATION SYSTEM

Performance of existing software implementations of Ray-tracing algorithm is still strictly limited by modern processors that require many processors to achieve real-time performance. Therefore, a structure of ray-tracing hardware implementation on FPGA (SaarCOR) is analyzed. Operating at the frequency of 90 MHz, the hardware implementation of ray tracing algorithm achieves real-time rate of 20 to 60 frames per second in a wide range of 3D scenes to support texturing, multiple light sources and multiple levels of reflection or transparency. The feature of this system is reusing the transformation unit for tasks decision including the effective search of the intersection of rays with triangles. Despite the additional support for dynamic scenes this approach reduces the overall cost of the hardware part by 68%. Also to accelerate ray tracing algorithm the mathematical core on FPGA can be used. This kernel supports multiplication, addition, subtraction, division, square root, comparison operations. Due to the use of such funds the classical ray tracing algorithm is transformed into a hardware-software tool that sends data to the extended hardware, which is based on FPGA, through one of the interfaces, and then receives the result calculation on the same interface. To modify this system we propose a method for interpixel interpolation based on the assumption that adjacent pixels of traced images have roughly the same colour options. The main idea is: to trace the pixels with some step, depending on the quality needs; to obtain the values of untraced colour components by the use of block interpolation; to check the result, if the coefficient of colour differences is more then tolerable, to perform adjustment of an interpolation step. The algorithms of line and block interpixel interpolation to accelerate ray tracing are developed and simulated using Message Passing Interface for .NET. technology (cluster DonNTU). Analysis of test results shows that the interpixel interpolation algorithm reduces the synthesis of the scenes from 12,5% to 15%, depending on the selected interpolation step and coefficient of differences in colour. A modification of SaarCor system is proposed. To go to the FPGA implementation of the interpixel interpolation algorithm a functional organization of the device is developed.

Keywords: ray-tracing, FPGA, real time, interpolation.