МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра информационно-измерительных систем

Артиков Тимур Неъматжанович

Разработка модуля многопроходных шейдерных эффектов для систем визуализации реального времени

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
по направлению высшего профессионального образования

230100.68 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема диссертации утверждена распоряжением по НГУ № 8 от «11» января 2012 г.
Тема диссертации скорректирована распоряжением по НГУ № 535 от «14» декабря 2012 г.

Руководитель

Долговесов Б. С.

к.т.н., зав. лаб. ИАиЭ СО РАН

Новосибирск, 2013 г.

Оглавление

Введение

Современные системы визуализации для повышения реалистичности получаемого изображения используют различные графические эффекты. Широко применяются эффекты постобработки – преобразования изображения трехмерной сцены, выполняемые после визуализации всех объектов сцены. К таким эффектам относятся цветовая коррекция, имитация расфокусировки камеры, туман, эффект смазывания при движении и другие. Так же используется предварительная обработка входных изображений (например, с видеокамер). Одним из ее примеров является выделение актера на монохромном фоне.

В данной работе представлен модуль многопроходных эффектов обработки изображений. Предложенное решение используется в системе визуализации, разрабатываемой лабораторией ИАиЭ СО РАН. Эффекты модуля реализованы в виде набора фильтров – алгоритмов, принимающих на вход несколько изображений и возвращающих другие в качестве результата. Для обеспечения высокой производительности обработка изображений фильтрами осуществляется на графическом процессоре с использованием шейдеров. Кроме этого, путем объединения фильтров в граф, обеспечивается возможность построения сложных эффектов из нескольких более простых.

Был предложен формат описания графа фильтров эффектов на скриптовом языке. Каждому фильтру можно указать шейдер, входные изображения, свойства выходного изображения – абсолютные или относительные размер, цветовое пространство, формат пикселей и т.д. Модуль эффектов по описанию эффекта строит конвейер обработки изображений с помощью шейдеров.

В ходе работы над проектом были поставлены следующие задачи:

1. Изучение предметной области
2. Формулирование требований к разрабатываемому модулю
3. Разработка способа задания и применения эффектов обработки изображений
4. Разработка формата описания эффектов
5. Разработка модуля и его интеграция с системой визуализации
6. Создание примеров эффектов обработки изображений

Разработанный модуль эффектов, позволил добавить в систему визуализацию ряд новых возможностей. Для создания эффекта разработчику необходимо написать код фильтров изображений на языке шейдеров и описание графа фильтров на скриптовом языке.

Текст работы состоит из трех глав. В первой главе описывается предметная область и требования, предъявляемые к разрабатываемому модулю. Вторая глава посвящена предлагаемым подходам к реализации модуля и формату описания эффектов. В третьей главе рассматривается программная реализация модуля и его практическая апробация при создании эффектов.

Глава 1. Постановка задачи

Эффекты обработки изображений

Визуализация в системах виртуальной реальности – это процесс получения изображения трехмерной сцены на основе ее свойств. К таким свойствам относятся форма и положение полигональных моделей сцены, свойства материалов, текстуры, наличие источников света.

Для улучшения качества полученного изображения сцены к нему могут быть применены некоторые преобразования – эффекты постобработки [3]. Они позволяют имитировать такие визуальные явления как туман, сияние, смазывание при движении и другие. Кроме того на сцене могут присутствовать изображения, которые требуют предварительной обработки.

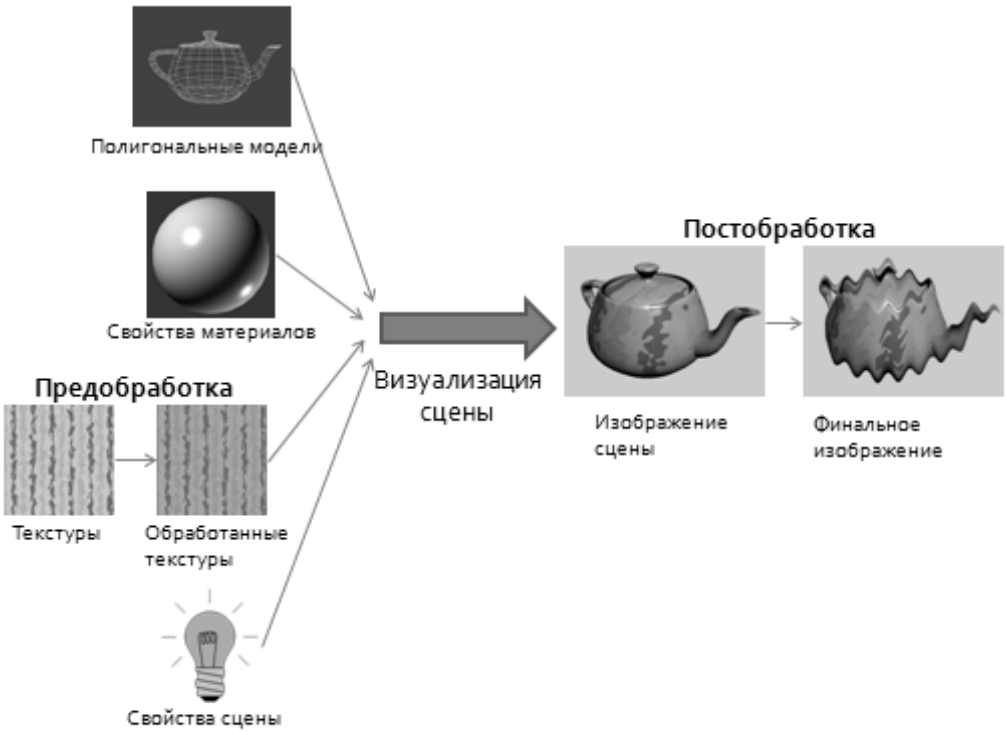

Рисунок 1. Процесс визуализации

Каждый эффект обработки изображений принимает на вход несколько изображений возвращает одно в качестве результата, цвет пикселей которого вычисляется по заданному алгоритму. Как правило, каждый пиксель обрабатывается независимо от других, что позволяет использовать параллельные вычисления, в том числе вычисления на графических процессорах.

Некоторые эффекты обработки изображений могут состоять из нескольких последовательных стадий, каждая из которых, в свою очередь, является другим более простым эффектом.

Для эффектов, назначенных на изображение трехмерной сцены, важной является возможность доступа к информации о форме объектов сцены. Например, при реализации эффекта тумана, сила тумана рассчитывается на основании расстояния от точки сцены до камеры, эту информацию можно получить из буфера глубины сцены.

Эффекты обработки динамических изображений (видео), могут обрабатывать несколько последовательных кадров изображения, подобные преобразования относятся к классу временной фильтрации.

Требования к разрабатываемому модулю

Целью данной работы было создание модуля эффектов обработки изображений для системы визуализации, разрабатываемой в ИАиЭ. Данная система обладает рядом особенностей. Основное ее применение – системы виртуальной реальности, позволяющие совмещать трехмерные виртуальные сцены с изображениями реальных актеров. Система может применяться в таки приложениях как обучающие демонстрации, лекции, тренажеры, видео конференции [2].

Рисунок 2. Пример применения системы визуализации

Система предполагает интерактивное взаимодействие с пользователем, поэтому визуализация не может быть произведена заранее, а рассчитывается в режиме реального времени. Значительная часть вычислений при этом осуществляется с использованием современных графических процессоров. Так же важной особенностью системы является возможность воспроизведения и отображения большого количества потоков видео.

Разрабатываемый модуль эффектов должен добавить в систему возможность назначить любому входному изображению некоторое преобразование – эффект. Так же эффекты можно назначать на порты вывода трехмерных сцен. Требуется предусмотреть возможность объединения нескольких эффектов в цепочку для их последовательного применения. Необходима поддержка временной фильтрации динамических изображений, в частности для эффекта устранения чересстрочности видео (деинтерлейсинг).

В силу того что система работает в режиме реального времени, и частота обновления кадров составляет 60 раз в секунду, важным требованием является требование к производительности. Модуль должен позволять задействовать ресурсы графического процессора. Базовые эффекты, поставляемые вместе с модулем, должны быть реализованы максимально эффективно, а накладные расходы работы модуля пренебрежимо малы по сравнению с непосредственно обработкой изображений.

Так же одним из требований было совместимость формата описания эффектов с форматом базы данных, в которой хранятся все данные в системе, необходимые для визуализации.

Набор эффектов, которые будут назначаться на изображения, сильно зависит от конкретной трехмерной сцены, требований к качеству и производительности, поэтому необходимо предусмотреть возможность удобного создания новых эффектов, а так же их гибкой конфигурации и настройки.

Существующие решения

Были проанализированы возможности модулей, отвечающих за эффекты, в наиболее распространенных системах визуализации реального времени. В качестве таких систем рассмотрены системы различных «весовых» категорий: OGRE 3D [12] – бесплатная библиотека трехмерной графики с открытым исходным кодом; Unity [13] – среда разработки интерактивных приложений, преимущественно компьютерных игр; Unreal Development Kit (UDK) [14] – профессиональная система разработки компьютерных игр, предназначенная для крупных проектов.

Все вышеперечисленные системы обладают широкими возможностями по обработке изображений и включают в свою поставку большое количество готовых эффектов.

В Unity эффекты описываются на императивном языке общего назначения C# в виде последовательности вызовов функций, таких как создание или удаление изображения, копирование, обработка пикселей изображения. Недостатком этого подхода

является необходимость вручную управлять изображениями, которые выделяются под промежуточные результаты – программист может забыть освободить временные изображения, или использовать их неоптимальным образом. В то же время использование императивного языка позволяет задать некоторую сложную логику обработки в зависимости от входных параметров.

В OGRE 3D для задания эффектов используется специально разработанный для этого скриптовый язык. Он позволяет задать свойства изображений, выделяемых под промежуточные результаты, и последовательность операций над ними.

В отличие от двух предыдущих решений, в которых используется модель вычислений, основанная на потоке команд (control flow) в UDK используется модель вычислений, основанная на потоке данных (data flow) [6]. Эффект задается в виде направленного графа, узлы которого – фильтры, обрабатывающие изображения, а дуги передача изображений между фильтрами. Последовательность запуска фильтров, создания и удаления временных изображений определяется системой автоматически на основе конфигурации связей в графе.

В UDK граф фильтров задается в редакторе с графическим пользовательским интерфейсом. Альтернативный подход – задание графа в текстовом виде. Каждый из этих способов имеет свои преимущества, у графического представления – это наглядность и удобство редактирования. У текстового – наличие развитых средств редактирования, таких как поиск, замена, копирование фрагментов текста; при этом может быть использован любой из существующих текстовых редакторов.

Ни в одной из перечисленных систем визуализации не была обнаружена возможность временной фильтрации. Вероятно, это связано с тем, что рассмотренные системы не ориентированы на работу с видео, при которой возникает необходимость в этой функции.

По мнению автора работы, наилучший способ задания эффектов используется в UDK – представление в виде графа фильтров, благодаря преимуществам модели вычислений, основанной на потоке данных. Создание специального редактора для построения и модификации графа – является спорным решением в силу того, что разработкой эффектов, как правило, занимаются программисты, для которых текстовое представление является достаточно удобным.

Глава 2. Предлагаемое решение

Обработка изображений с помощью шейдеров

Для обеспечения необходимой производительности обработка изображений в модуле осуществляется на графическом процессоре. Существует несколько технологий вычислений общего назначения на видеокарте, однако, некоторые из них поддерживаются лишь на видеокартах одного производителя (NVIDIA CUDA, AMD FireStream), а другие лишь на самых современных видеокартах (вычислительные шейдеры). Поэтому для обработки изображений были выбраны пиксельные шейдеры, которые обладают рядом ограничений и не предназначены для произвольных вычислений, но в то же время поддерживаются широким классом видеокарт.

Визуализация с использованием видеокарты осуществляется в несколько последовательных стадий, которые образуют графический конвейер. Некоторые стадии графического конвейера являются программируемыми. Программы для них называются шейдерами [9]. Вершинный шейдер отвечает за преобразования вершин полигональных моделей, пиксельный шейдер – за вычисление цвета пикселей. При визуализации трехмерных сцен в вершинном шейдере, как правило, происходит проецирование вершин в пространство экрана, а в пиксельном – текстурирование и расчет освещения.

Рисунок 3. Графический конвейер

Рассмотрим, как шейдеры позволяют осуществить обработку изображений. Укажем в качестве выхода визуализации не экран, как это происходит по умолчанию, а изображение, в котором будет получен результат обработки. В качестве геометрии, подаваемой на вход графического конвейера, используем квадрат единично размера. Вершинный шейдер для любых преобразований – одинаковый, он размещает вершины квадрата в углы выходного изображения так, чтобы он покрыл все изображение. В результате растеризации к каждому пикселю выходного изображения будет применен пиксельный шейдер. В пиксельном шейдере осуществляется необходимый расчет цвета пикселя на основе его координат и параметров, общих для всего пиксельного шейдера.

Ниже приведен пример простейшего пиксельного шейдера изменения яркости изображения:

```
sampler2D inputImage;
float k;
float4 main(float2 coord)
{
	float4 color = tex2D(inputImage, coord);
	return float4(color.rgb * k, color.a);
}
```

В качестве параметров шейдера подается входное изображение и коэффициент изменения яркости, на который умножается результирующий цвет. Данное преобразование является поточечным, то есть выборка цвета из входного изображения происходит в координатах, совпадающих с координатами обрабатываемого пикселя. Однако, ничто не мешает изменять эти координаты, производить несколько выборок, в том числе из нескольких входных изображений.

Обработка изображений с помощью шейдеров имеет ряд ограничений. Эти ограничения различаются на различных видеокартах и включают в себя ограничения на количество шейдерных инструкций, на количество параметров шейдеров, отсутствие поддержки побитовых операций, вещественных чисел двойной точности, возможности обмена данными между потоками, обрабатывающими различные пиксели. Несмотря на эти ограничения, пиксельные шейдеры позволяют решать широкий класс задач обработки изображений и были успешно использованы в разработанном модуле.

Представление эффекта в виде графа фильтров

Кроме однократной обработки изображений шейдерным фильтром модуль эффектов позволяет создавать сложные, многопроходные эффекты из нескольких фильтров. Данная возможность необходима по нескольким причинам. Во-первых, это повышает удобство использования, можно создать несколько фильтров и комбинировать их в различных сочетаниях. Во-вторых, шейдер фильтра, производящий слишком сложные вычисления, может не скомпилироваться в силу ограничений на число шейдерных инструкций. Кроме того некоторые алгоритмы обработки могут быть реализованы более эффективно в несколько проходов, чем в один.

В работе предлагается задания эффекта в виде графа фильтров. Вершины графа – фильтры, т.е. алгоритмы, обрабатывающие изображения с помощью шейдеров. Каждый фильтр принимает на вход несколько изображений и возвращает одно выходное изображение. Дуги в графе показывают передачу изображений между фильтрами, они соединяют выходы одних фильтров с входами других. В каждый вход может входить лишь одна дуга, количество дуг выходящих из выхода – неограниченно. Граф при этом не должен иметь циклов. В графе присутствует выделенная вершина без входов, она обозначает входное изображение эффекта. Один из фильтров помечен как выходной фильтр.

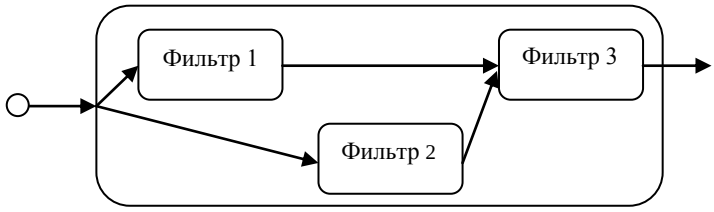

Рисунок 4. Задание эффекта в виде графа фильтров

Процесс обработки изображений с помощью графа фильтров заключается в последовательном запуске фильтров в некотором порядке, с последующей передачей результата соединенным с ним фильтрам. Порядок запуска фильтров задается топологией графа – фильтр может быть запущен при условии, что все входные изображения уже посчитаны. В силу ацикличности графа такой порядок всегда существует. Результатом работы эффекта является результат работы выходного фильтра.

Рассмотрим пример графа фильтров для эффекта «сияние». Данных эффект добавляет светящиеся ореолы вокруг ярких объектов.

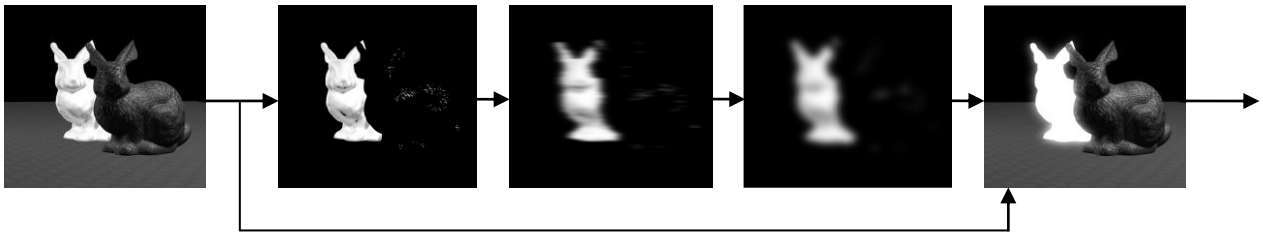

Рисунок 5. Граф фильтров эффекта «сияние»

На первом шаге из изображения выделяются участки, яркость которых превышает заданный порог. Затем к этим участкам применяется несколько шагов размытия. Полученный результат складывается с исходным изображением.

Реализация размытия в эффекте сияние является примером случая, когда обработка в несколько проходов может быть более эффективной, чем в один. Фильтр размытия Гаусса с ядром NxN требует N^2 операций на обработку каждого пикселя. Если же выполнять его в два последовательных прохода – горизонтальное размытие с ядром $1xN$, и вертикальное с ядром $Nx1$, то потребуется всего $2N$ операций.

Представление эффектов в виде графа фильтров является частным случаем модели вычислений, основанных на потоке данных. Альтернативный способ – это модель вычислений, основанных на потоке команд, в которой алгоритм обработки записывается в виде последовательности операций над данными. Пример эффекта «сияние» в данной модели мог бы быть записан следующим образом:

```
void bloomEffect(Image dst, Image src)
{
      Image img1 = createImage(src.width, src.height);
      applyShader(img1, src, "brightPass");
      Image img2 = createImage(src.width, src.height);
      applyShader(img2, img1, "blurX");
      applyShader(img1, img2, "blurY");
      destroyImage(img1);
      applyShader(dst, src, img2, "add");
      destroyImage(img2);
}
```

Видно, что в данном случае происходит явное создание и удаление изображений под промежуточные результаты. При этом применена неочевидная оптимизация – одно и то же изображение переиспользуется для хранения результатов различных фильтров, за счет чего достигается экономия памяти. В предлагаемом решении программист не должен

об этом заботиться, система сама осуществляет управление временными изображениями и делает это оптимальным способом.

Дополнительные изображения сцены

В процессе визуализации из трехмерной сцены получается двухмерное изображение. В этом изображении хранится информация о цвете поверхности объектов с учетом текстурирования, расчетом освещения и т. д. Однако часть информации о сцене теряется. По изображению сцены нельзя определить расстояние от камеры до какой-либо заданной точки сцены, нормаль к поверхности в этой точке, цвет поверхности без учета освещения и т. д. Возможность получения этой дополнительной информации о сцене позволила бы реализовать некоторые эффекты обработки изображений сцен. Такая возможность была реализована в модуле эффектов.

Добавлен специальный тип фильтров, который не имеет ни одного входа, а на выходе возвращает изображение, в пикселях которого содержится дополнительная информация о сцене. Такие изображения будем называть дополнительными изображениями сцены.

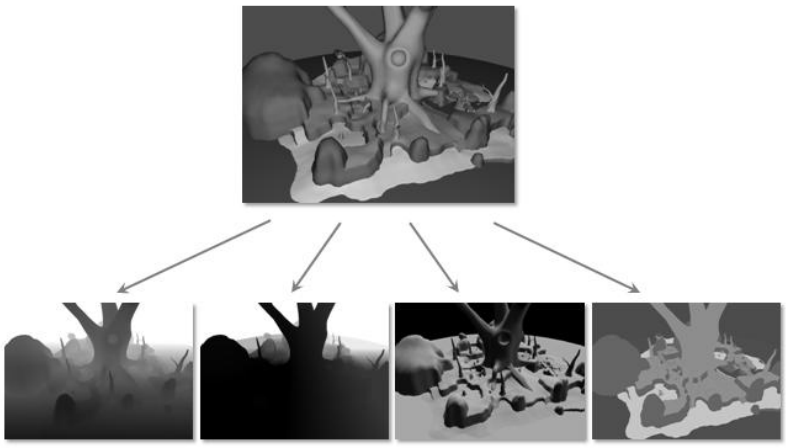

Рисунок 6. Дополнительные изображения сцены

Для получения дополнительного изображения фильтр производит повторную визуализацию сцены, при этом шейдер материала заменяется специальным шейдером, который вместо цвета возвращает необходимую информацию о сцене. Так же используется возможность современных видеокарт создавать изображения с вещественным 32-х битным форматом пикселей (а не целочисленным 8-ми битным, который используется для хранения цвета), что позволяет хранить информацию, принимающую широкий диапазон значений.

В качестве примера эффекта, использующего дополнительные изображения сцены, можно привести эффект расфокусировки камеры – размытие изображения сцены с переменным радиусом, который зависит от расстояния от сцены до камеры.

Временная фильтрация

Одной из особенностей системы визуализации, для которой разрабатывался модуль эффектов, является работа с видео. Эффект, представленный в виде графов фильтров, может быть применен к видео, в результате каждый кадр видео будет обработан эффектом. Однако в ряде случаев этой возможности недостаточно, и требуется обработка нескольких последовательных кадров одновременно, т.е. временная фильтрация.

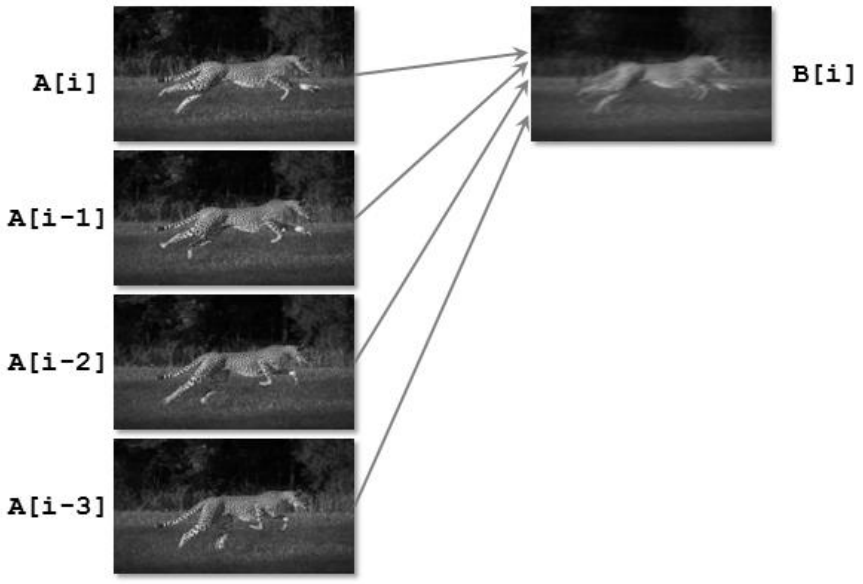

Рисунок 7. Временная фильтрация

Для поддержки временной фильтрации у фильтров было введено понятие «памяти». Память – это очередь фиксированного размера, в которой накапливаются изображения – результаты работы фильтра за несколько кадров. При заполнении очереди наиболее старые изображения удаляются из очереди, освобождая место новым. В результате фильтры, на вход которым подается выход фильтра с памятью, могут использовать несколько изображений с предыдущих кадров. Так же памятью обладает эффект в целом, в ней накапливаются кадры входного изображения эффекта.

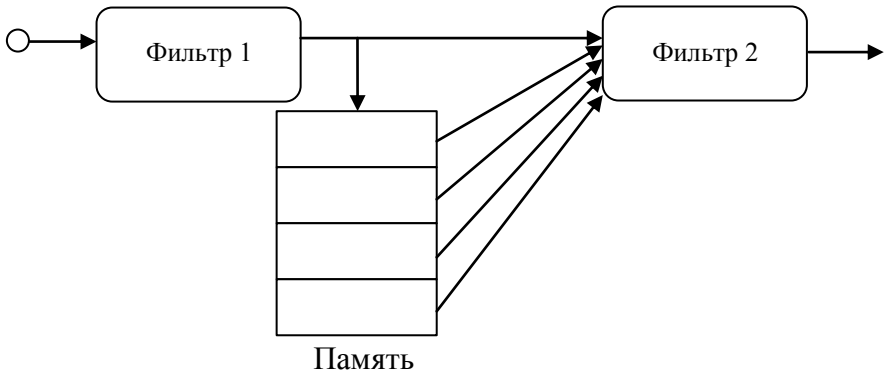

Рисунок 8. Память фильтров

Примером временной фильтрации является размытие при движении – усреднение цвета пикселей изображений за несколько кадров.

Формат описания эффектов на языке XQL

Данные, необходимые для визуализации в системе, хранятся в единой объектно-ориентированной базе данных [1]. База данных имеет древовидную структуру. Узлы дерева обладают именем и хранят данные одного из нескольких типов: целое число, вещественное число, строка, массив или пустое значение. Адресом узла называется последовательность имен узлов в пути от корня дерева до этого узла, разделенных точкой. Каждый узел обладает уникальным адресом. Предполагается, что некоторые узлы в базе соответствуют объектам, а их дочерние узлы – свойства объектов.

Для работы с базой данных существует программный интерфейс для языков программирования, а так же специальный скриптовый язык XQL (eXtended Query Language). XQL обладает простым синтаксисом, код на нем состоит из последовательности присваиваний узлам базы некоторых значений:

```
MATERIAL.wood.DIFFUSE = (255, 100, 100)
MATERIAL.wood.MAPNAME = "wood.jpg"
```

Для присваивания значений узлам с общим началом адреса существует сокращенная запись с использованием фигурных скобок. Вышеописанный пример эквивалентен следующему коду:

```
MATERIAL.wood = {
      DIFFUSE = (255, 100, 100)
      MAPNAME = "wood.jpg"
}
```

Язык XQL удобен для описания любых иерархических данных. С его помощью в системе задаются трехмерные сцены, графический пользовательский интерфейс, настройки визуализации. Поэтому логичным решением является использовать его в том числе и для описания эффектов. В базу данных была добавлена ветка `EFFECT`, хранящая описания эффектов. Эффект содержит несколько фильтров (pass), один из которых отмечен как выходной фильтр:

```
EFFECT.e1 = {
      PASS = {
            P1 = <pass description>
            P2 = <pass description>
            …
            Pn = <pass description>
      }
      OUT = Pi
}
```

Каждому фильтру можно указать:

- Имя шейдера, с помощью которого фильтр будет обрабатывать изображения – `EFFECT.*.PASS.* = <shaderName>` (где "*" означает произвольное имя)
- Входные изображения, которые могут быть входом всего эффекта или выходом другого фильтра – `EFFECT.*.PASS.*.<imgName> = IN | PASS.<passName>`
- Свойства выходного изображения, такие как размер, масштаб, формат пикселей –

```
EFFECT.*.PASS.*.OUT = {
      SIZE = (<width>, <height>)
      SCALE = (<scaleX>, <scaleY>)
      FORMAT = RGB | RGBA | YUV | …
}
```

Чтобы фильтр не производил обработку изображения, а визуализировал сцену специальным шейдером для получения дополнительного изображения сцены, необходимо указать свойство фильтра `EFFECT.*.PASS.*.DRAW = SCENE`.

Размер памяти фильтров для временной фильтрации указывается в узле `EFFECT.*.PASS.*.OUT.MEMORY` и аналогично для входа эффекта – `EFFECT.*.IN.MEMORY`.

Рассмотрим действия, которые необходимо совершить для создания эффекта с помощью модуля на примере эффекта «сияние».

Необходимо написать описания эффекта на языке XQL:

```
// bloom.xql
EFFECT.bloom = {
    PASS = {
        brightPass = SHADER.brightPass {
            img = IN
        }
        blurX = SHADER.blurX {
            img = PASS.brightPass
            OUT.SCALE = (0.5, 0.5)
        }
        blurY = SHADER.blurY {
            img = PASS.blurX
            OUT.SCALE = (0.5, 0.5)
        }
        final = SHADER.add {
            img1 = IN
            img2 = PASS.blurY
        }
    }
    OUT = final
}
```

Затем для каждого фильтра необходимо написать шейдеры, на языке шейдеров. Ниже приведен шейдер для фильтра brightPass:

```
// brightPass.cgfx
float threshold = 0.9;
sampler2D img;
float4 main(float2 tc : TEXCOORD0) : COLOR
{
     float4 color = tex2D(img, tc);
     float brightness = dot(float3(0.3, 0.59, 0.11), color.rgb);
     return brightness < threshold ? float4(0, 0, 0, 1) : color;
}
```

После этого эффект можно использовать, то есть назначить его на изображение или на порт вывода сцены и задать ему параметры шейдеров. Одновременно может быть назначено несколько различных эффектов, которые будут применяться друг за другом.

```
// main.xql
IMAGE.img1 = "1.jpg" {
    EFFECT.01 = bloom {
        blurSize = 15
        threshold = 0.7
    }
}
MODEL.m1 = "bunnies.xql"
PORT.p1 = {
    MODEL = m1
    EFFECT.01 = bloom
    EFFECT.02 = grayscale
}
```

Глава 3. Программная реализация

Модуль эффектов

Важным свойством базы данных в системе визуализации является то, что она служит не только для хранения данных, но и для межмодульного взаимодействия. Одни модули записывают в базу данные, а другие модули считывают их, тем самым осуществляется взаимодействие. Ключевую роль в системе играет графическое ядро (Renderer), этот модуль производит расчет изображений трехмерных сцен. Так же существуют модули для загрузки изображений и полигональных моделей, расчета анимации, проигрывания видео и другие.

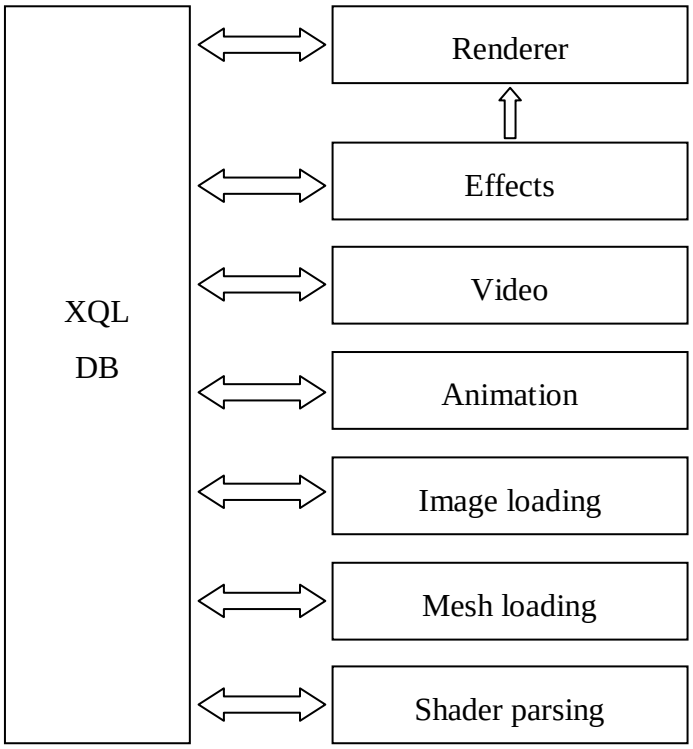

Рисунок 9. Структура системы визуализации

Модуль эффектов взаимодействует с базой данных для считывания описания эффектов. С графическим ядром модуль взаимодействует напрямую, путем вызова функций. Это решение было выбрано для обеспечения приемлемой производительности, так как в противном случае происходит слишком большое количество обращений к базе данных. Модуль может вызывать следующие функции графического ядра:

- Компиляция шейдера с заданным исходным кодом
- Установка параметров шейдера
- Создание изображения заданного размера и формата для промежуточных результатов обработки

- Установка изображения в качестве выхода для визуализации
- Визуализация квадрата с использованием шейдера для осуществления обработки изображения фильтром
- Визуализация объектов сцены с использованием шейдера для получения дополнительного изображения сцены
- Копирование изображений

Каждому узлу базы вида `EFFECT.*` соответствует класс описания эффекта (`EffectDesc`), содержащий такую информацию как размер памяти входа, имя выходного фильтра, а так же список описаний фильтров (`EffectPassDesc`). В описании фильтра (`EffectPassDesc`) указываются шейдер, входные изображения, тип (обрабатывает изображения или рисует сцену) и свойства выходного изображения – размер, формат пикселей, размер памяти.

Для изображений и портов вывода сцены, которым назначены эффекты, создается цепочка эффектов (`EffectChain`). Цепочка эффектов содержит последовательность эффектов (`Effect`), созданных на основе соответствующих описаний. Каждый эффект содержит список фильтров (`EffectPass`) и параметры шейдеров фильтров. Порядок фильтров в эффекте определяется на основе описания графа фильтров с помощью топологической сортировки фильтров [5]. Некоторые фильтры могут иметь память (`EffectImageMemory`).

Процесс обработки изображений с помощью эффектов запускается при изменении списка эффектов в цепочке, при изменении параметров шейдеров или при изменении данных в изображении. Обработка изображения с помощью цепочки эффектов заключается в последовательном применении эффектов в этой цепочке. Обработка изображения эффектом – это, в свою очередь, последовательное применения фильтров эффекта. Для обработки изображения фильтром шейдеру фильтра устанавливаются параметры эффекта и входные изображения; в качестве выхода визуализации устанавливается временное изображение, с размером и форматом соответствующими описанию фильтра; выполняется визуализация прямоугольника или сцены в зависимости от типа фильтра. Если фильтр обладает памятью, то результирующее изображения копируется в память.

Для промежуточных результатов обработки модуль эффектов запрашивает у ядра создание временных изображений. Создавать изображения при каждой обработке – неэффективно, поэтому в модуле имеется пул изображений, из которого при необходимости берутся изображения нужного размера и формата.

Примеры использования модуля

Было реализовано несколько эффектов, демонстрирующих работоспособность модуля.

Эффект сияния

Данный эффект моделирует явление возникновения светящихся ореолов вокруг ярких объектов. Стандартным способом его реализации является размытие ярких участков изображения с помощью фильтра Гаусса (рисунок 5). При больших размерах ядра фильтра производительность обработки становится недопустимо низкой для расчета в реальном времени, поэтому данный метод не позволяет реализовать сияние с достаточно большим размером ореолов.

В работе предлагается альтернативный способ реализации эффекта сияния большого радиуса. Размытие происходит в несколько шагов с небольшим размером ядра фильтра Гаусса (3x3 пикселя), при этом на каждом шаге размер изображений уменьшается в два раза. Уменьшение изображений не только увеличивает производительность, но и позволяет достичь более сильного размытия.

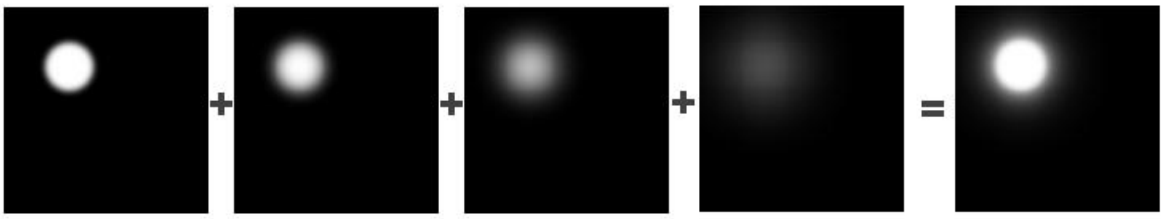

Рисунок 10. Размытие в несколько шагов

Полученные на каждом шаге изображения увеличиваются до размеров первоначального изображения с помощью бикубической интерполяции, смешиваются между собой с некоторыми коэффициентами и прибавляются к исходному изображению.

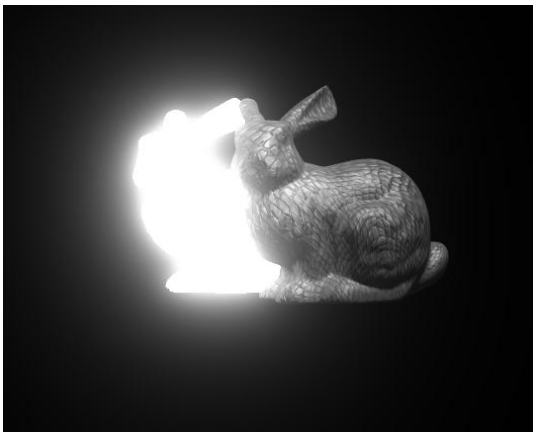

Рисунок 11. Эффект сияние

Цветовая коррекция

Цветовая коррекция – это класс преобразований изображений, в которых цвет пикселей результирующего изображения вычисляется лишь на основе цвета соответствующих пикселей исходного изображения (а не их координат или цвета соседних пикселей).

Было реализовано два варианта эффекта цветовой коррекции. В первом варианте к изображению применяется последовательность наиболее часто используемых фильтров цветовой коррекции: изменение яркости, контраста, насыщенности, тона, гамма-коррекция. Пользователь может изменять силу каждого из фильтров.

Во втором варианте эффекта цветовая коррекция задается с помощью трехмерной таблицы преобразований. Значения функции преобразования цвета задаются в ячейках таблицы, а промежуточные результаты вычисляются с помощью линейной интерполяции, что позволяет использовать произвольную функцию преобразования (точнее ее кусочно-линейную аппроксимацию).

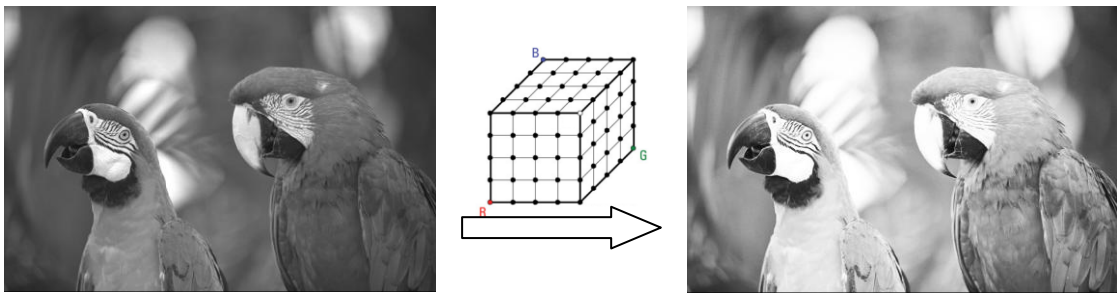

Рисунок 12. Цветовая коррекция с использованием таблицы преобразований

Имитация расфокусировки камеры

При съемке на реальную камеру объекты, находящиеся вне фокуса, становятся размытыми. Для моделирования данного явления при визуализации виртуальных сцен используются эффекты постобработки [8].

Был реализован метод, основанный на размытии изображения сцены с переменным радиусом, зависящим от расстояния от сцены до камеры. С помощью возможности модуля создания дополнительных изображений сцены получается изображение расстояний от точек поверхности сцены до камеры. На основе этого изображения и параметров камеры рассчитывается изображение степени размытия – чем больше расстояние от точки сцены до фокуса камеры, тем больше степень размытия. Исходное изображение обрабатывается фильтрами размытия различного размера. Результат эффекта

получается путем интерполяции между размытыми изображениями на основе значения из изображения степени размытия.

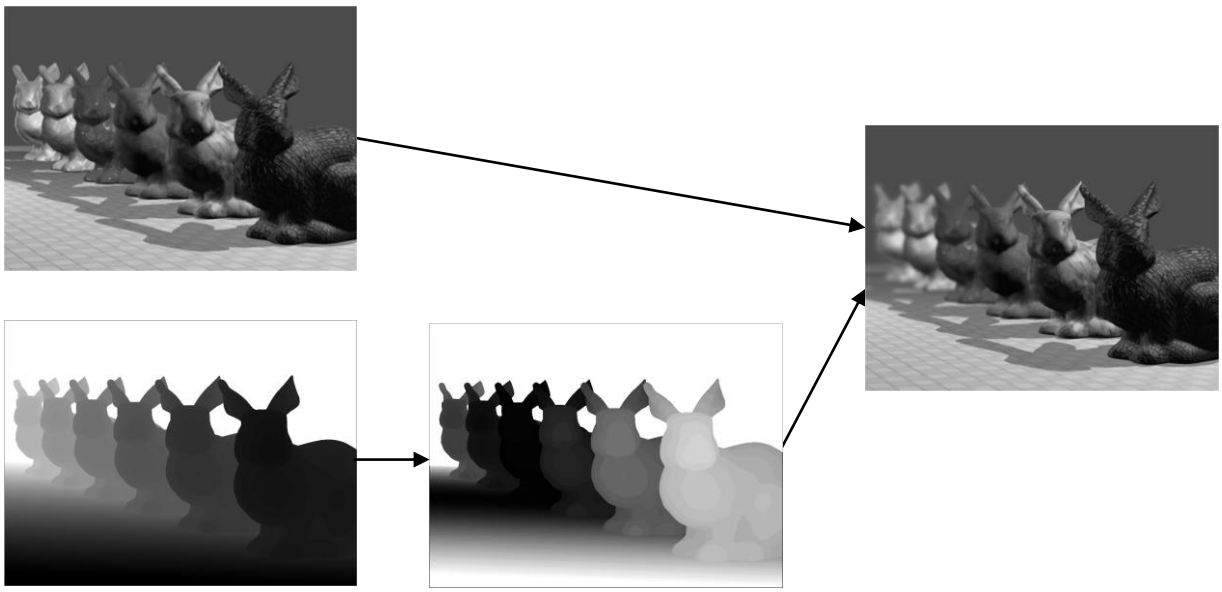

Рисунок 13. Эффект расфокусировки камеры

Эффекты обработки видео

Модуль эффектов используется не только моделирования каких-либо визуальных явлений, а так же для некоторых вспомогательных функций, в частности, для обработки видео. Проигрывание видео осуществляется соответствующим модулем, который преобразует видеоданные в последовательность изображений. Последний этап декодирования видео заключается в переводе изображения из формата сжатия YUV2 [15] в формат RGB. Данное преобразование может быть осуществлено на видеокарте с использованием шейдеров, поэтому для этих целей используется модуль эффектов.

Кроме того, некоторые видео могут иметь формат с чересстрочной разверткой. При выводе их на устройства отображения с прогрессивной развёрткой, такие как компьютерные мониторы, возникает эффект «гребенки». Методы для избавления от этого эффекта называются алгоритмами деинтерлейсинга.

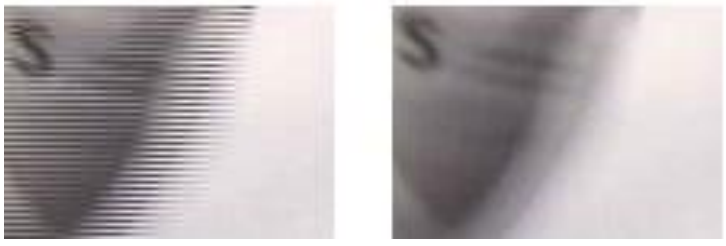

Рисунок 14. Деинтерлейсинг

Был реализован алгоритм деинтерлейсинга YADIF (однопоточная реализация для CPU была переписана на язык шейдеров). Данный алгоритм требует обработки трех последовательных кадров видео, что обеспечивается возможностью временной фильтрации в модуле эффектов.

Эффект выделения актеров на монохромном фоне (chroma key) позволяет совмещать видео актеров с виртуальными сценами [4]. Для каждого пикселя входного изображения определяется прозрачность на основании расстояния между цветом пикселя и заданным цветом фона в цветовом пространстве YUV. Полученная маска прозрачности обрабатывается фильтром устранения шума, а к изображению применяется цветовая коррекция для устранения освещения актера светом, отраженным от экрана, на фоне которого происходила съемка.

Рисунок 15. Выделение актера на монохромном фоне

Глобальное освещение

Алгоритмы глобального освещения – это алгоритмы расчета освещения, в которых учитывается не только влияние лучей света, падающих на поверхность непосредственно из источника света (прямое освещение), но и лучей, отраженных от других поверхностей (отраженное освещение). Стандартные методы, основанные на трассировке лучей, на данный момент не позволяют достичь производительности необходимой для расчета в реальном времени.

Был реализован метод приближенного расчета глобального освещения, представляющий собой эффект постобработки изображения сцены [10]. На вход ему подается изображение сцены, в котором рассчитано лишь прямое освещение, а так же дополнительные изображения, содержащие для каждого пикселя изображения сцены позицию в пространстве, нормаль к поверхности и отражательные свойства материала.

Для пикселя изображения, в котором осуществляется расчет освещения, берется соответствующая ему точка в пространстве (точка P на рисунке 16) и множество

случайных точек поверхности сцены в ее окрестности. Рассчитывается, какое количество света отражается в этих точках в направлении точки P, это отраженное освещение прибавляется к исходному прямому освещению.

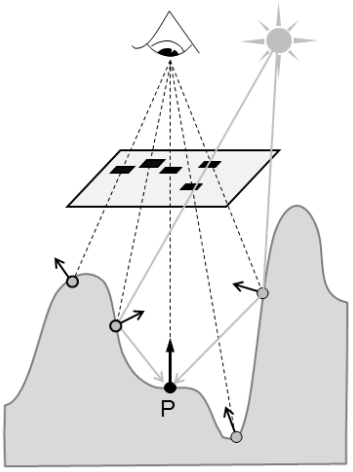

Рисунок 16. Расчет отраженного освещения

Следует отметить, что метод является приближенным, отраженное освещение учитывается лишь от видимых объектов, рассчитывается лишь однократное отражение лучей света без учета пересечений с объектами сцены. Из-за небольшого количества точек, от которых рассчитывается отраженное освещение, изображение получается зашумленным, поэтому дополнительно применяется фильтр устранения шума.

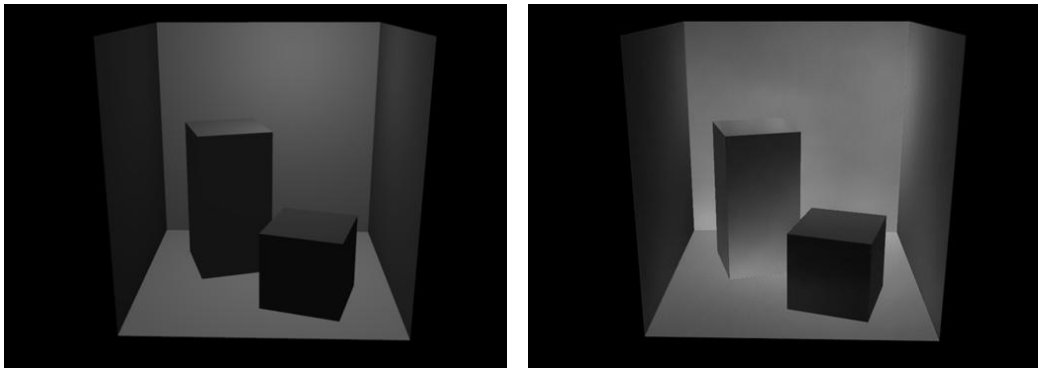

Рисунок 17. Глобальное освещение

Важным достоинством метода является независимость времени расчета от сложности геометрии сцены (количества полигонов), так как обрабатываются изображения. Метод не требует предварительного расчета и подходит для полностью динамических сцен. На современных видеокартах при размере изображения 1680x1050 расчет занимает меньше 20 миллисекунд, что удовлетворяет требованиям работы в режиме реального времени.

Заключение

Были получены следующие результаты:

1. На основе изучения существующих подходов к организации эффектов обработки изображений был предложен способ задания эффектов в виде графов фильтров. Фильтры могут быть двух типов – обрабатывающие изображения с помощью шейдеров или визуализирующие сцену для получения дополнительных изображений сцены. Некоторые фильтры могут иметь «память», что позволяет реализовать временную фильтрацию.
2. Был предложен формат описания эффектов на скриптовом языке XQL, позволяющий задавать свойства фильтров и соединять их в граф.
3. Был разработан модуль эффектов для системы визуализации Лаборатории синтезирующих систем визуализации ИАиЭ СО РАН, который по описанию эффектов осуществляет обработку изображений.
4. С помощью модуля реализованы эффекты обработки видео: преобразование форматов, выделение актера на монохромном фоне, деинтерлейсинг.
5. Реализованы эффекты постобработки трехмерных сцен: сияние, цветовая коррекция, имитация расфокусировки камеры, глобальное освещение.

Возможности модуля не ограничены реализованными эффектами, и в дальнейшем планируется реализовать другие эффекты, необходимость в которых возникнет при использовании системы. Также могут быть расширены возможности самого модуля: добавлены фильтры с несколькими выходами, возможность динамического изменения графа фильтров на основе параметров эффекта.

Литература

1. Долговесов Б. С. Объектно-ориентированная база данных в интерактивных системах 3d визуализации. // Вестник Новосибирского Государственного Университета. 2011.
2. Долговесов Б. С. 3D графика реального времени: от тренажеров до виртуальных студий // Graphicon 2005.
3. Дьячков В. Реализация процессора эффектов постобработки. // www.uraldev.ru. 2008.
4. Ковальков М.А. Разработка и реализация алгоритма рирпроекции на базе современного графического акселератора. // Graphicon 2006.
5. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Глава 22.4. Топологическая сортировка // Алгоритмы: построение и анализ . 2005.
6. Цилькер Б.Я. Орлов С. А. Глава 15. Вычислительные системы с управлением вычислениями от потока данных. // Организация ЭВМ и систем. 2004.
7. Akenine-Moller T. Real-time rendering. 2008.
8. Hammon E. Practical Post-Process Depth of Field. // GPU Gems 3 Chapter 28. 2008.
9. Kessenich J. The OpenGL® Shading Language. 2010.
10. Ritschel T. Approximating Dynamic Global Illumination in Image Space. 2009.
11. Selan J. Chapter 24. Using Lookup Tables to Accelerate Color Transformations. // GPU Gems 2. 2004.
12. OGRE 3D compositor scripts documentation http://www.ogre3d.org/docs/manual/manual_29.html
13. Unity image effects documentation http://docs.unity3d.com/Documentation/Components/comp-ImageEffects.html
14. UDK Post Process Effects documentation http://udn.epicgames.com/Three/PostProcessEffectsHome.html
15. YUV pixel formats // http://www.fourcc.org/yuv.php