Использование настраиваемого графического конвейера

М. Самохвалов

В данной статье рассматривается использование принципиально новой архитектуры полностью настраиваемого графического конвейера. Также рассматриваются детали его реализации и возможности по рендерингу нефотореалистичных изображений.

1. Введение в настраиваемый конвейер

Процесс визуализации (рендеринг) состоит из большого числа операций, которые производятся над трехмерными данными, начиная от установки детализации объектов и заканчивая растеризацией треугольников. Все эти операции в сумме составляют графический конвейер (pipeline).

Для простоты принято разделять конвейер на стадии. Примером таких стадий могут служить растеризация примитивов, текстурирование, трансформация и т. д. Эти стадии выполняются в строго определенном порядке, так как являются сильно зависимыми друг от друга.

В данной статье рассказывается о применении нелинейного графического конвейера в виде графа стадий, а также о деталях его реализации в случае нефотореалистичного рендеринга.

2. Настраиваемый конвейер

Описание архитектуры настраиваемого конвейера приведено в [2]. Основными ее положениями можно считать нелинейность выполнения стадий, общую память для стадий и естественный параллелизм конвейера.

2.1. Получение векторного контура

В процессе создания нефотореалистичного рендера возникла основная идея построения контура, которая сформировалась в результате исследований — для нахождения контура в рендере используется аналитическое его описание, а для удаления его невидимых частей используется Z-буфер. При этом мы получаем на выходе алгоритма векторный контур, собранный из видимых частей полного контура. Для реализации этого алгоритма использовалась конфигурация конвейера, показанная на рис. 1.

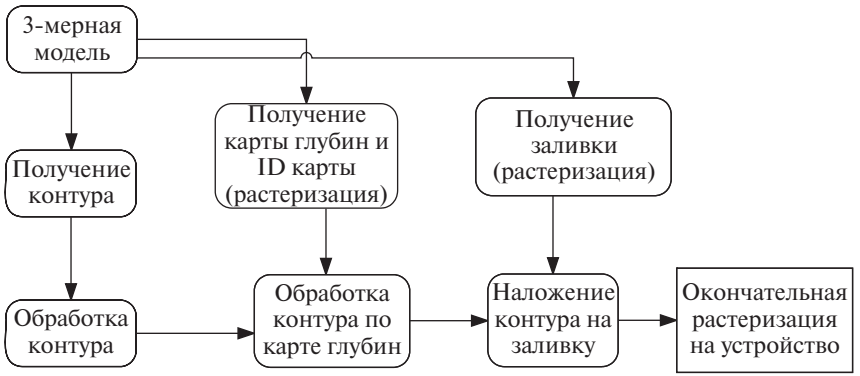

Рис. 1. Пример конвейера для нефотореалистичной визуализации

Этот конвейер отличается от распространенных сейчас наличием стадии, которая обрабатывает контур по данным буфера глубин и ID буферу, а также использует результаты из нескольких стадий.

Используя такой конвейер, мы получаем возможность одновременно использовать внутри конвейера векторные и растровые данные. Контур в данном случае представляет собой набор отрезков (можно также представить его в виде кривой, построенной по ключевым точкам). Далее мы производим растеризацию линий контура и каждую точку проверяем на видимость при помощи алгоритма, сходного с тем, который описан в работе [1], но модифицированным для нашего конвейера.

2.2. Трассировка контура по Z & ID буферам

Модификация алгоритма заключается вот в чем: для различных алгоритмов, применяемых при растеризации и при трассировке контура, может возникнуть «дребезг» пикселей из-за того, что мы переводим векторные данные на дискретное по своей сути изображение. При этом получается, что контур может наезжать на залитые части или наоборот вылезать за их пределы. Разница в [1] была не сильно заметна из-за того, что алгоритмы в этой работе оптимизированы для работы только в паре, а также из-за увеличенной толщины контура (больше 2 пикселей). При этом эти недостатки скрадывались. Разница же в работе алгоритмов растеризации не составляла больше одного пикселя (рис. 2 и рис. 3). В [1] использовалась схема «ближние соседи» для того, чтобы «дребезг» контура меньше влиял на качество растеризации. Таким образом, при определении видимости пикселя брались также и его соседи и, если хотя бы один из соседей имел меньшую величину Z в Z-буфере, то пиксель рисовался.

Недостаток данной схемы в том, что при использовании разных алгоритмов несоответствие линий может доходить до 2 пикселей, поэтому в нашей системе использовалась схема «соседи ближних соседей» (рис. 4).

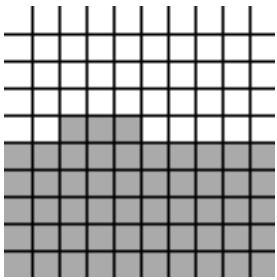

Рис. 2. Z-буфер

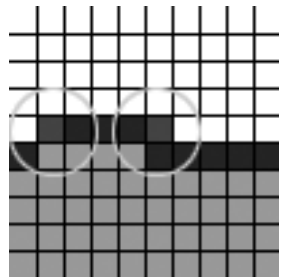

Рис. 3. Контур и проблемные места

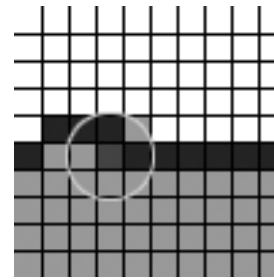

Рис. 4. Артефакт с величиной ошибки в 2 пикселя

При этом возникает еще больше артефактов, которые необходимо убрать из результирующего изображения.

Предложенное развитие алгоритма в этой работе основывается на том факте, что контур, принадлежащий к конкретной грани не должен далеко от нее «уходить».

Для этого был использован так называемый ID-буфер, который делается в процессе растеризации. В этот буфер записываются уникальные идентификаторы треугольника, который растеризуется. При этом при трассировке контура смотрятся значения ID-буфера, которые находятся рядом с проверяемым пикселом и в том случае, когда рядом не оказывается ни одной грани, которой контур принадлежит, то сам контур (точнее его конкретная точка) считается невидимым. Это уменьшает артефакты до минимальной величины. При этом возникают артефакты «нахлеста» контуров. Размер «нахлеста» не превышает одного пикселя. Это можно исправить дополнительным анализом, используя ID самого контура.

Достоинства такого алгоритма:

- Независимость алгоритмов растеризации треугольников от алгоритмов растеризации контура.
- Возможность использовать те же алгоритмы, что и в [1] для устранения артефактов самого контура в его аналитическом представлении.

Недостаток:

- Требуется больше вычислений.

Схема успешно используется в системе нефотореалистичного рендеринга. Также этот пример ясно показывает возможности и мощность алгоритмов, которые пользуются сетевой архитектурой конвейера, так как без использования дополнительных буферов решить задачу убирания артефактов невозможно.

3. Интеграция существующих алгоритмов в конвейер

3.1. Шейдинг

В настоящее время для закраски поверхностей (и не только) применяется операция, называемая шейдингом (shading). Принципы ее работы в архитектурах OpenGL и REYES примерно одинаковы, различается только реализация. Основной принцип заключается в том, что шейдингом занимается микропрограмма, которая работает на одной из стадий конвейера. Эта микропрограмма может заниматься как обработкой векторных данных, так и заливкой при растеризации. В шейдер помимо информации о геометрии и материалах также может поступать информация обо всех источниках освещения в сцене и дополнительные параметры. Такой же метод мы можем применить и в нашей архитектуре.

Каждая единица трехмерной геометрии имеет свой материал. Этот материал связан с шейдером, которым может являться как скриптовый код, так и библиотека, скомпилированная из языков высокого уровня (C++ или другой язык, включая специализированные).

Этот материал поступает на вход алгоритма rasterizer вместе с самой геометрией, информацией об источниках света и т. д. и определяет цвет в точке экранного пространства, который затем будет на результирующей картинке.

В фотореалистичных рендерах, как правило, имеются такие материалы, как Lambert, Phong, Anizotropic, которые отвечают физическим моделям освещения. Эти материалы имеют настройки для определения цвета освещенной поверхности, цвета бликов и т. д.

Как правило, для получения нефотореалистичных эффектов используются специализированные материалы. Для эффекта классического Toon затенения используется материал, который устанавливает 2 цвета. Один из них определяет результирующий цвет для освещенной стороны, другой — для затененной.

Такие специализированные шейдера используются и в нашей системе. Для примера их использования был сделан аналог шейдера Toon. Его реализация, как и в классическом случае, использует 2 цвета и может определять уровень освещенности поверхности по положению источников освещения, которые передаются в шейдер.

В системе используются шейдера, которые могут совмещать как операции над вертексами, так и операции над фрагментами. Все треугольники обрабатываются независимо.

3.2. Фильтры edge detection

В случае, если художник не хочет использовать методы получения векторного контура, он может воспользоваться методом фильтров Edge Detection, который работает с худшим качеством, но опробован временем.

Для этого в системе есть стадии, которые можно подключить к выходу стадии растеризации для получения контуров методом фильтрации. После

чего результаты работы фильтра над Z-буфером и буфером нормалей можно совместить для получения результирующей картинки с контуром.

3.3. Динамический холст

Этот метод используется обычно в последней стадии создания изображений (compose). Один из методов для получения динамического холста можно увидеть в работе [4]. В нашей системе для добавления динамического холста используется отдельная стадия, которая занимается наложением полученной в предыдущих шагах картинки с альфа-каналом на изображение холста.

3.4. Алгоритмы, работающие с данными нескольких стадий

Примером работы таких стадий может служить описанный выше метод для трассировки контура и удаления невидимых его частей.

4. Детали реализации

4.1. Произвольный доступ к данным стадиями

Система нефотореалистичного рендеринга разрабатывалась с самого начала с учетом этой архитектуры, и все условия для среды были выполнены. Для каждой стадии существует ее описание на XML, в котором указаны типа данных, которыми она оперирует. Таким образом, для каждой стадии определяется какие данные необходимо ей передать, эти данные упаковываются и пересылаются стадии. Формат данных может быть как встроенный в рендер (например, изображения), так и внешний. В этом случае всей обработкой занимается стадия, а корректность передаваемых данных проверяется лишь на уровне описания стадии. Все стадии выполняются в одном адресном пространстве, поэтому доступ к данным неограничен.

4.2. Описание стадий и потока выполнения

Для описания стадий был использован XML (см. [5]) как основа для структурного описания.

Вся информация обо всех имеющихся стадиях хранится в локальном репозитарии, который представляет собой XML файл. Каждая стадия имеет свое уникальное имя и уникальный идентификатор. Идентификатор должен быть уникален в сети. Имя стадии предназначено для настройки конвейера и является коротким идентификатором для удобства. Пример описания локальной базы приведен ниже:

```
<?xml version="1.0" encoding="utf-8"?>
<stages>
<stage name="StandartRasterizator"/>
<stageID=''NPRRender.Rasterization.StandartRasterizator''/>
<in name=''geom'' dataTag=
          ''StandartTypes.Geometry.TrisData''>
<out name=''idbuffer'' dataTag=
          ''StandartTypes.GBuffers.IDBufferData''/>
```

```
<out name=''zbuffer'' dataTag=
          ''StandartTypes.GBuffers.ZBufferData''/>
<out name=''nbuffer'' dataTag=
          ''StandartTypes.GBuffers.NBufferData''/>
<out name=''silhbuffer'' dataTag=
          ''StandartTypes.GBuffers.SilhBufferData''/>
<out name=''imagebuffer'' dataTag=
          ''StandartTypes.GBuffers.ImageBufferData''/>
<stage/>
<stage name="NPRRender.Filters.StandartEdgeDetection"/>
...
<stage/>
...
</link>
</stages>
```

Здесь мы видим название стадии, ее тип, входные и выходные параметры. Наименования данных и типы стадий используются по соглашению «имя компании/продукта — общее название — конкретный тип». Для создания собственных алгоритмов есть возможность использовать свои типы данных. Для этого необходимо при соединении стадий в конвейер следить за соединениями. Используя визуальные редакторы можно эту задачу решать автоматически.

При помощи такого репозитария можно конфигурировать компьютеры для выполнения только определенных стадий. Для этого достаточно настроить репозитарий только на выполнения определенных стадий.

Описание конвейера также хранится в XML. Это описание идет вместе со сценой или проектом и предназначено для обработки сетевым рендером.

Типичный пример конвейера:

```
<?xml version="1.0" encoding="utf-8"?>
<pipeline>
<nodes>
<node name=''StandartRasterizator'' alias=''Rasterizator''>
...
</nodes>
<connections>
<connect node_out="contourFinder.out"
node_in="contourTracer.inContour">
<connect node_in="imageOut.image" node_out="Rasterizator.
imagebuffer">
...
</connections>
</pipeline>
</xml>
```

Таким образом, мы связываем стадии в один граф. Стадии, которые на вход принимают только геометрию, по умолчанию являются старто-

выми, т. е. вся обработка начинается с них. Названия стадий должны совпадать с теми, которые указаны в локальном репозитарии.

4.3. Параллельное выполнение стадий

Параллельная работа организована на протоколе TCP-IP. На нем организована система передачи данных и команд на локальные сервисы рабочих станций. При помощи команд можно:

1. Узнать, занята ли рабочая станция рендером в текущее время.
2. Узнать какую сцену, кадр, стадию обрабатывает станция.
3. Выяснить, какие стадии можно выполнять на этой станции.
4. Узнать код и описание последней ошибки, в случае неуспешного выполнения.
5. Передать данные, необходимые для рендера.
6. Начать рендеринг на станции.
7. Отменить рендеринг.

Данные команды представляют собой все необходимое для функционирования системы рендеринга в сети. Реализация сетевой подсистемы выполнено как сервис на локальной машине и является стандартным решением для клиент-серверных приложений.

5. Заключение

Полученные результаты дают право говорить о том, что описанный выше конвейер может успешно использоваться в приложениях как для получения нефотореалистичной графики, так и в других областях, в которых необходимо использовать сложные алгоритмы обработки трехмерных данных. Также этот конвейер способен распараллеливать вычисления в неоднородном сетевом окружении более эффективно, чем линейный. Большим плюсом также является гибкость управления выполнением вычислений и возможности автоматизации работ по созданию графики.

Литература

1. *Isenberg*, *Helper*, *and Strothotte*. Stylizing Silhuettes at Interactive Rates: From Silhuette Edges to Silhuette Strokes. The Eurographics Association and Blackwell Publishers, 2002.
2. *Самохвалов М.* Архитектура настраиваемого конвейера рендеринга с общей памятью (см. в данном сборнике).
3. *Adam Lake*, *Carl Marshall*, *Mark Harris*, *Marc Blackstein*. Stylized Rendering Techniques for Scalable Real-Time 3D Animation. In Proceedings of NPAR 2000, Symposium on Non-Photorealistic Animation and Rendering (Annecy, France. June 2000). P. 13–20. N. Y., 2000. ACM.
4. —itMatthieu Cunzi, Joelle Thollot, Sylvain Paris, Gilles Debunne, Jean-Dominique Gascuel, Fredo Durand. Dynamic Canvas for Non-Photorealistic Walkthroughts, ARTIS/GRAVIR, MIT, REVES/INRIA. Sophia-Antipolis, 2003.
5. Extensible Markup Language (XML) (www.w3.org/XML).