RenderWare Graphics

Руководство пользователя

Том 3

Связаться с нами

Критерион Софтвер Лтд.

Для получения общей информации о RenderWare Graphics отправьте электронное письмоinfo@csl.com .

Отношения с разработчиками

Для получения информации о поддержке отправьте электронное письмоdevrels@csl.com .

Продажи

Для получения информации о продажах обращайтесь:rw-sales@csl.com

Благодарности

Участники	Команды разработки графики и документации RenderWare

Оглавление

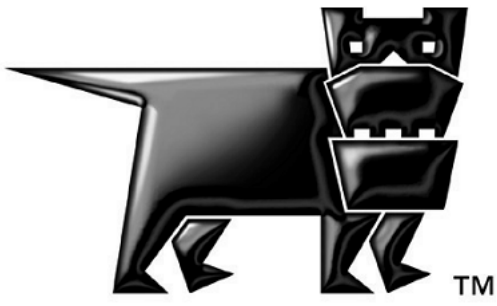
Часть F

Библиотеки утилит

Глава 27

2D-графика

Наборы инструментов



27.1 Введение

В этой главе рассматриваются два набора инструментов для работы с 2D-графикой:**Рт2д**и**RtCharset**.

27.1.1 Набор инструментов 2D

Рт2д,*2D-инструментарий*, предоставляет богатый API 2D-графики, который в полной мере использует ускорение, предоставляемое современным 3D-графическим оборудованием. Это обеспечивает поддержку функций, в том числе:

- смешивание
- сглаживание
- прозрачность
- быстрое вращение
- поддержка растровых и контурных шрифтов

27.1.2 2D-объекты

2D-объекты основаны на разделе 2D Toolkit, объясняющем, как сохранять объекты, содержащие кисти, пути и шрифты, чтобы вы могли повторно использовать их и манипулировать ими. 2D-объекты, которые можно создать, следующие:

- формы
- сцены
- выбрать регионы
- строки объекта

27.1.3 Набор инструментов для работы с символами

RtCharset,*Набор инструментов для набора символов*, обеспечивает поддержку отображения текста с использованием базового, быстрого растрового шрифта. Его скорость делает его полезным для отображения отладочной и метрической информации.

Набор инструментов Character Set Toolkit не содержит излишеств, что делает его относительно негибким и непригодным для использования в большинстве выпущенных продуктов.

27.2 Использование 2D-инструментария

The**Рт2д**API позволяет разработчику создавать 2D-графические изображения, используя всю мощь графического движка RenderWare. 2D Toolkit использует традиционные 2D-графические примитивы, такие как*кисти*и*пути*, чтобы облегчить работу с инструментом.

27.2.1 Инициализация

The**Rt2dOpen()**Функция инициализирует 2D Toolkit. Однако 2D Toolkit необходимо указать, какую камеру использовать, прежде чем можно будет выполнить рендеринг.**Rt2dOpen()** Функция поэтому принимает**RwCamera**указатель в качестве единственного параметра.

Эту функцию необходимо вызвать до того, как можно будет использовать любые 2D-функции в наборе инструментов.

После**Rt2dOpen()**был вызван, можно заменить текущую камеру на другую, используя**Rt2dDeviceSetCamera()**функция.

Весь рендеринг выполняется на выбранной камере. Рендеринг на камеру, к которой прикреплен растр для последующего использования в качестве текстуры (**rwТИП РАСТРАКАМЕРАТЕКСТУРА**) также поддерживается.

Камеры

Объект виртуальной камеры RenderWare Graphics подробно рассматривается в главе под названием:*Камера*.

Закрытие 2D-инструментария

Это необходимо сделать до завершения работы RenderWare Graphics, и это достигается с помощью**Rt2dClose()**.

27.2.2 Абстракция устройства

Теперь, когда камера настроена как цель для рендеринга, следующим шагом будет опросить ее и определить ее размеры и другие полезные свойства. Для этой цели API 2D Toolkit включает набор**Rt2dDevice...()** функции.

Координатное картографирование

Камера использует**RwRaster**для визуализированной графики. Поскольку этот растровый объект может быть произвольного размера,**Rt2dDeviceGetMetric()**Функцию можно использовать для получения текущего начала координат и сопоставления ширины/высоты:

```
успех = Rt2dDeviceGetMetric ( &x, &y, &width, &height );
```

Переменные**x**,**y**,**ширина**и**высота**являются**RwReal**значения, которые получат источник выходного устройства (**x**и**y**) и его ширина и высота.

Также возможно использовать**Rt2dDeviceSetMetric()**функция для изменения этих значений, чтобы приложение могло работать с более низким или более высоким внутренним разрешением.

2D Toolkit может визуализировать 2D-графику в любом масштабе. Он также может визуализировать графику в плоскости, установленной на произвольном расстоянии от камеры (см. *Наслаивание*, ниже), поэтому другая функция устройства,**Rt2dDeviceGetStep()**, предназначен для определения размера пикселя при текущей глубине и масштабе.

The**Rt2dDeviceGetStep()**Функция заполняет два 2D-вектора, представляющих шаг в один пиксель в**x**и**y**осей и заполняет третий вектор смещением относительно начала координат.

Наслаивание

2D Toolkit визуализирует в плоскости, параллельной плоскости обзора камеры. Глубина плоскости — ее расстояние от камеры — может быть изменена с помощью **Rt2dDeviceSetLayerDepth()**.

По умолчанию плоскость находится на глубине 1,0 единицы. Любое новое значение должно быть больше нуля.

Флаги трубопровода

Rt2dSetPipelineFlags()принимает комбинацию флагов, определяющих, как будет выполняться рендеринг. Флаги определяются **RwIm3DTransformFlags** перечисления, хотя только**rwIM3D_NOCLIP** и**rwIM3D_ALLOPAQUE**флаги дают полезные результаты.

Ценности логически**ИЛИ**с текущими настройками флага немедленного режима.

27.2.3 Текущая матрица трансформации

2D Toolkit опирается на базовый 3D графический движок. Это означает, что 2D координаты должны быть преобразованы в 3D пространство, прежде чем можно будет выполнить какой-либо рендеринг.

The*Текущая матрица трансформации*(CTM) используется для преобразования 2D-данных в 3D-пространство. Поддержка нескольких CTM осуществляется с использованием стекового механизма.

- **Rt2dCTMPush()**помещает новый CTM в стек, делая его активным CTM.

- **Rt2dCTMPop()**удаляет верхний CTM из стека и уничтожает его, делая активным следующий CTM в стеке.

CTM можно вращать, масштабировать и переводить. Эти преобразования соответствующим образом влияют на визуализированные изображения, позволяя накладывать слои, вращать и прокручивать 2D-графику.

27.2.4 Пути

Ан**Rt2dPath**объект определяет путь в 2D пространстве. Путь может содержать одну или несколько линий или кривых и может быть как открытым, так и закрытым.

Рендеринг пути включает указание*щетка*(покрытый в*27.2.5 Кисти*), который используется для рисования пути.

Кисть определяет цвет и текстуру, используемые при рисовании (известную как *поглаживание*) или заполнение пути.

Создание путей

The**Rt2dPathCreate()**функция возвращает указатель на пустой объект пути (**Rt2dPath**). Когда путь создан, он*заперт*.

После создания информация о пути добавляется к объекту с использованием одной или нескольких функций, перечисленных в следующей таблице.

ИСПОЛЬЗОВАТЬ	К
Rt2dPathMoveto()	Перейти к определенной координате
Rt2dPathLineto()	Нарисуйте линию от текущего местоположения до другого
Rt2dPathRLineto()	Нарисуйте линию от текущего местоположения до другого (используются относительные координаты)
Rt2dPathCurveto()	Нарисуйте кривую от текущего местоположения до другого
Rt2dPathRCurveto()	Нарисуйте кривую от текущего местоположения до другого (контрольные точки указаны в относительных координатах)
Rt2dPathRect()	Добавьте прямоугольник к пути
Rt2dPathRoundRect()	Добавьте к контуру скругленный прямоугольник.
Rt2dPathOval()	Добавьте овал к траектории

Блокировка и разблокировка путей

Пути можно заблокировать с помощью**Rt2dPathLock()**и разблокирован с помощью **Rt2dPathUnlock()**.

Открытые и закрытые пути

Ан*открыть*путь имеет два отдельных конца в разных местах.

А*закрыто*Путь описывает замкнутый многоугольник и начинается и заканчивается в одном и том же месте.

Открытый путь можно закрыть, позвонив**Rt2dPathClose()**. Эта функция добавит дополнительную линию, соединяющую два конца пути.

Удаление путей

Когда ваше приложение закончило работу с объектом path, его необходимо уничтожить. Это выполняется вызовом**Rt2dPathDestroy()**.

Пути рендеринга

Путь может быть отображен либо*поглаживание*это, или по*заполнение*это.

Обводка пути — когда путь рисуется как линия — включает указание кисти, которая определяет цвет и/или текстуру, которые будут использоваться во время рендеринга. Текстура будет выложена плиткой или растянута вдоль пути в соответствии с UV-отображением, установленным с помощью**Rt2dBrushSetUV()**. Ширина штриха задается вызовом**Rt2dBrushSetWidth()**.

Чтобы нарисовать путь, позвоните**Rt2dPathStroke()**, который принимает указатель как на контур, так и на кисть.

Заполненные пути

Заполнение рассматривает путь как окно, через которое будет видна кисть. Кисть визуализируется как текстура того же размера, что и ограничивающий прямоугольник пути. Сам путь используется как маска, так что части текстуры, находящиеся за пределами пути, не визуализируются.

Rt2dPathFill()используется для заполнения указанного пути с использованием цветов и координат текстуры заданной кисти. Путь должен быть замкнутым, чтобы эта функция работала правильно. Цвет заливки для каждой точки внутри пути определяется билинейной интерполяцией цветов кисти, предполагая, что они представляют цвета четырех углов ограничивающего прямоугольника пути. Следовательно, цвет заливки зависит от относительного расстояния каждой внутренней точки от угловых точек ограничивающего прямоугольника пути. Если кисть также указывает координаты текстуры и изображение текстуры, путь заполняется изображением, предполагая, что углы ограничивающего прямоугольника имеют координаты текстуры кисти.

Из-за используемого алгоритма,**Rt2dPathFill()**приведет к неопределенным результатам, если указанный путь вогнут или пересекает сам себя.

Значение вставки

2D Toolkit поддерживает*вставка*значение для каждого пути. Это значение указывает смещение от центра штриха пути. Это смещение используется при рендеринге, так что путь может быть повторно отрисован вставкой или выступом от предыдущего рендеринга того же пути.

Например, путь, описывающий простой прямоугольник, может быть отображен один раз, а затем повторно отображен с измененным значением вставки таким образом, чтобы внутри исходного поля появился второй прямоугольник.

Сглаживание кривых

Необходимо заменить кривые короткими отрезками линий — процесс, известный как *сглаживание*—так что кривая может быть визуализирована. В таких случаях **Rt2dPathFlatten()**Функция может быть использована для преобразования кривых в пути в сегменты прямых линий. Значение допуска, которое управляет количеством полученных прямых линий, может быть изменено с помощью **Rt2dPathSetFlat()**. По умолчанию это значение равно 0,5.

В идеале выравнивание должно выполняться только один раз для каждой кривой, так как повторные вызовы**Rt2dPathFlatten()**может существенно повлиять на производительность.

Ограничительные рамки

У путей есть ограничивающий прямоугольник, который представляет собой 2D-ячейку с границами, полностью содержащую путь. Этот прямоугольник важен при визуализации формы, описываемой путем, поскольку кисть, используемая для заполнения формы, будет масштабироваться в соответствии с размерами ограничивающего прямоугольника пути.

The**Rt2dPathGetBBox()**Функция используется для получения ограничивающего прямоугольника для указанного пути. Этот ограничивающий прямоугольник использует**Rt2dBBox**структура, которая определяет двумерный ящик.

```
{
  RwReal    x;    /* x-координата нижнего левого угла */ /* y-
  RwReal    y;    координата нижнего левого угла */ /*
  RwReal    ж;        ширина  * /
  RwReal    час; /*  высота  * /
}
```

Обтравочные контуры

RenderWare Graphics будет избегать выполнения операций обрезки, если базовая платформа позволяет это делать. (Это касается PlayStation®2 платформы, например.) Однако вашему приложению может потребоваться выполнить обрезку для своих собственных целей. По этой причине,*контур обрезки* представляющая область, в которой происходит весь рендеринг, может быть получена с помощью **Rt2dDeviceGetClippath()**. Рендеринг 2D-графики с помощью 2D Toolkit не будет виден за пределами этого пути, поэтому его можно использовать для расчетов отсечения.

Обтравочный контур — обычный**Rt2dPath**объект и обычно представляет собой прямоугольник.

Пути опорожнения и копирования

Существующий путь можно очистить с помощью вызова**Rt2dPathEmpty()**. Эта функция удалит все существующие данные пути из структуры.

Копирование путей можно выполнить вызовом**Rt2dPathCopy()**. Эта функция вызовет**Rt2dPathEmpty()**на пути назначения; конкатенация не выполняется. Путь назначения должен быть создан с использованием **Rt2dPathCreate()**.

27.2.5 Кисти

Объект кисти (**Rt2dBrush**) представляет информацию о цвете и текстуре, используемую при обводке или заполнении пути. Кроме того, кисти можно присвоить ширину, которая определяет ширину обводимого пути.

Создание кистей

Кисти созданы с использованием**Rt2dBrushCreate()**, который возвращает указатель на допустимый объект кисти. Затем этот объект необходимо инициализировать полезными данными.

Характеристики

Объект кисти содержит ряд свойств, которые может задать приложение. Свойства вместе с функциями, используемыми для их задания, перечислены в следующей таблице. Эти свойства должны быть заданы приложением до использования кисти для рендеринга.

СВОЙСТВО	API-ФУНКЦИИ
РГБА. Используются четыре цветовых вектора. Они определяют цвет в каждом из четырех углов кисти. Цвета билинейно интерполируются при рендеринге.	**Rt2dBrushSetRGBA()**
Текстура. это определяет битовую карту, которая будет использоваться при рендеринге кисти. Текстура будет растянута и/или выложена плиткой в соответствии со значениями UV кисти (см. следующую запись).	**Rt2dBrushSetTexture()**
Текстурные координаты. Четыре пары UV хранятся в кисти. Для рисования они определяют координаты текстуры в углах результирующего примитива. При заполнении пути они определяют координаты текстуры в углах ограничивающего прямоугольника пути. В обоих случаях координаты текстуры углов упорядочиваются против часовой стрелки, а внутренние координаты определяются билинейной интерполяцией.	**Rt2dBrushSetUV()**
Ширина. Это определяет ширину обводимого контура.	**Rt2dBrushSetWidth()** **Rt2dBrushGetWidth()**

Рендеринг

Важно отметить, что кисти можно визуализировать только с помощью контуров. Например, рекламный щит (также известный как*спрайт*(некоторые программисты 2D-игр) будут созданы следующим образом:

1. Определите кисть с текстурой

2. Установите значения УФ-излучения в соответствии с требованиями.

3. Определите путь в форме простого прямоугольника.

4. Нарисуйте путь с помощью кисти и**Rt2dPathFill()**

Рендеринг выполняется с использованием 3D-движка RenderWare Graphics, поэтому данные текстуры могут содержать альфа-информацию, что позволяет реализовать эффекты полупрозрачности и прозрачности.

27.2.6 Текущая матрица трансформации

Матрица преобразования используется для преобразования вершин из одного пространства — например, локального или объектного пространства — в другое — обычно пространство устройства. 2D Toolkit использует матрицы преобразования во время рендеринга, пропуская все вершины, которые он обрабатывает, через матрицу преобразования при рендеринге.

2D Toolkit поддерживает стек матриц преобразования, и верхняя матрица в стеке — это*матрица преобразования тока*(CTM). CTM — это матрица, используемая во время рендеринга, но другие матрицы могут быть добавлены или удалены из стека произвольно.

Например, спидометр в гоночном автомобиле можно было бы визуализировать непосредственно на 3D-рендере приборной панели под правильным углом, масштабом и расстоянием от камеры, используя только 2D Toolkit. Этого можно было бы достичь, настроив необходимое преобразование в CTM.

Процесс рендеринга

Рендеринг с 2D Toolkit в основном касается путей, которые обводятся или заполняются кистями. При рендеринге пути 2D Toolkit преобразует путь в полоски треугольников, которые затем могут быть отрендерены.

Объектные пространства

Пути определяются в локальном объектном пространстве. Это означает, что первая вершина в пути всегда является началом координат для этого пути; последующие вершины в пределах того же пути определяются относительно этого локального начала координат.

Для того, чтобы отрисовать путь, его вершины должны быть преобразованы в пространство устройства. Это включает обработку пути через CTM. Обработанные вершины затем преобразуются в полосы треугольников 3D Immediate Mode и визуализируются в соответствии с настройками кисти.

Таким образом, CTM определяет преобразование, необходимое для перемещения вершин пространства объектов траектории в пространство экрана.

Стек CTM

Когда**Rt2dOpen()**функция называется,*Стек CTM*инициализируется, содержащим один CTM.

Инициализация CTM

Матрица преобразования по умолчанию — это единичная матрица. Вы можете сбросить текущую матрицу преобразования.

Поскольку все пути определены в объектном пространстве, вполне вероятно, что вашему приложению понадобится несколько матриц преобразования — часто по одной для каждого пути. По этой причине 2D Toolkit предоставляет API стека CTM для управления матрицами преобразования.

Матрицы преобразования хранятся как полные**RwMatrix**объекты. Таким образом, CTM можно скопировать в**RwMatrix**объект с призывом **Rt2dCTMRead()**.

Использование**RwMatrix**объекты означает, что преобразования могут использовать все три оси:**x**,**y**и**z**. Преобразование и рендеринг 2D-графики с использованием 2D Toolkit выполняются аппаратно везде, где это возможно, поэтому программисту 2D-графики становится доступен ряд методов, которые ранее требовали интенсивной обработки процессором, например, вращение и наложение слоев.

Использование стека CTM

Ранее мы видели, что CTM является самой верхней матрицей в стеке. Когда нужна новая матрица преобразования, текущая матрица преобразования может быть перемещена вниз по стеку, а новая CTM помещена наверх стека. Это достигается вызовом**Rt2dCTMPush()**. Новая CTM представляет собой копию проталкиваемой матрицы преобразования.

Чтобы вернуться к предыдущей матрице преобразования, текущую матрицу можно вытащить из стека, удалив ее в процессе, так что следующая матрица в стеке станет текущей матрицей преобразования. Это выполняется вызовом **Rt2dCTMPop()**.

2D-инструментарий*всегда*выполняет рендеринг с использованием текущей матрицы преобразования, т.е. самой верхней матрицы преобразования в стеке CTM.

Настройка преобразований

Когда допустимый CTM находится в стеке, 2D Toolkit может использоваться для установки преобразований, требуемых приложением. В большинстве случаев приложению потребуется только преобразовать вершины в 2D-плоскости. Поэтому предоставляются специальные функции 2D-преобразования, как показано в следующей таблице:

К	ИСПОЛЬЗОВАТЬ
Применить трансляцию к текущей матрице преобразования (CTM), используя указанные x- и y-компоненты	**Rt2dCTMTranslate()**
Применить масштабное преобразование к текущей матрице преобразования, используя указанные коэффициенты масштабирования по осям x и y.	**Rt2dCTMScale()**
Применить поворот к текущей матрице преобразования, используя указанный угол поворота.	**Rt2dCTMRotate()**

Все преобразования предварительно объединены с CTM.

Рендеринг и камеры

Рт2дфункции рендеринга должны вызываться внутри **RwCameraBeginUpdate()**и**RwCameraEndUpdate()**используя последнюю камеру, установленную для операций Rt2d через**Rt2dOpen()**или **Rt2dDeviceSetCamera()**.

27.2.7 Шрифты

2D Toolkit поддерживает три конкретных типа шрифтов:

1. *Метрики 1*–Формат растрового шрифта, требующий растрового изображения (также может быть указана необязательная маска). Связанный**.встретил**Файл — текстовый файл — используется для определения позиций каждого символа в растровом изображении.

2. *Метрики 2*–Вариант*Метрики 1*. Этот формат использует маркерные пиксели в растровом изображении для определения местоположения каждого символа, устраняя необходимость указывать их явно в**.встретил**текстовый файл.

   Главным преимуществом этого формата является поддержка нескольких файлов растровых изображений и большего количества символов. Это делает его более подходящим для приложений, где необходимо использовать нелатинские символы (например, китайские, греческие, японские или корейские наборы символов).

3. *Метрики 3*–Контурный шрифт. Этот формат в некоторой степени основан на Adobe® Формат шрифта Type 1, в котором каждый символ явно определяется как путь с использованием текстовых инструкций внутри**.встретил**файл.

Форматы файлов полностью описаны в разделе*27.5 Форматы файлов шрифтов*, в конце этой главы.

Рендеринг контурных шрифтов

Важно отметить, что контурные шрифты*всегда*Обведено. Невозможно отобразить заполненные символы.

Использование шрифтов

Прежде чем шрифт можно будет использовать, его сначала нужно прочитать. Это выполняется вызовом**Rt2dFontRead()**. Эта функция берет имя**.встретил**file. Путь поиска для файла метрик шрифта должен быть установлен до этого вызова с помощью **Rt2dFontSetPath()**. Результатом является указатель на**Rt2dFont**объект.

В качестве альтернативы двоичный шрифт можно загрузить из**RwStream**с использованием **Rt2dFontStreamRead()**. Это загружает только данные шрифта и возвращает **Rt2dFont**объект. Как контурные, так и растровые шрифты загружаются одним и тем же методом. Если шрифт является растровым, все связанные текстуры будут загружены в словарь текстур, если они еще не присутствуют. Фрагмент данных шрифта идентифицируется заголовком фрагмента**rwID_2DFONT** .

А**Rt2dFont**объект записывается в**RwStream**с использованием **Rt2dFontStreamWrite()**.**Rt2dFontStreamGetSize()**может использоваться для запроса размера в байтах**Rt2dFont**фрагмент данных в**RwStream**.

При работе с контурными шрифтами параметр высоты в большинстве случаев**Rt2dFont** функции определяют коэффициент масштабирования, с которым должен отображаться шрифт, а также верхнюю границу его ограничивающего прямоугольника. Для растровых шрифтов высота определяет только верхнюю границу ограничивающего прямоугольника. Масштабирование растровых шрифтов должно выполняться путем установки масштабного преобразования в CTM.

Для визуализации текста с использованием выбранного шрифта предусмотрены две функции:

1. **Rt2dFontShow()**визуализирует строку, отображаемую с использованием указанного шрифта и визуализируемую с использованием указанной кисти. Также необходимо указать 2D-вектор, задающий нижнюю левую координату ограничивающего прямоугольника текста, и, следовательно, его положение на экране, а также высоту визуализированного текста. 2D-вектор обновляется, чтобы указывать на конец позиции строки на экране. Это делается для того, чтобы все последующие строки визуализировались в правильной позиции после текущей строки.

   Отображается только одна строка, обрезается и преобразуется с использованием текущей матрицы преобразования.

2. **Rt2dFontFlow()**похож на**Rt2dFontShow()**, за исключением того, что он отображает строку в виде прямоугольника (**Rt2dBBox**), размещая текст в соответствии с указанным выравниванием. 2D Toolkit будет размещать текст в поле до тех пор, пока не закончится текст или не закончится место. В последнем случае текст будет обрезан.

Поддерживаемые флаги выравнивания для**Rt2dFontFlow()**перечислены в следующей таблице:

ФЛАГ	РЕЗУЛЬТАТ
rt2dJUSTIFYLEFT	Линии выровнены по левому краю ограничивающей рамки.
rt2dJUSTIFYCENTER	Линии центрируются внутри ограничивающей рамки.
rt2dJUSTIFYRIGHT	Линии выровнены по правому краю ограничивающей рамки.

Уничтожение шрифта

Когда ваше приложение закончит использовать определенный шрифт,**Rt2dFont** объект должен быть уничтожен с помощью вызова**Rt2dFontDestroy()**.

Словарь текстур шрифтов

Все текстуры для растровых шрифтов хранятся в словаре текстур, **RwTextDictionary** . Этот словарь может быть основным словарем или локальным словарь только для текстур шрифтов.

Rt2dFontTextDictionarySet()используется для установки активного словаря для хранения текстур шрифтов.**Rt2dFontTextDictionaryGet()**используется для запроса текущего активного словаря.

Шрифт Юникод

Unicode позволяет представить 1 миллион уникальных символов и поддерживает**Rt2dFont** объект.**Rt2dFont**поддерживает только первые 64 000 символов, которые могут быть закодированы с использованием двух байтов на символ. Этого все еще достаточно для кодирования большинства символов основных языков.

Явных функций для включения поддержки Unicode нет.**Rt2dFont**автоматически классифицируется как шрифт Unicode при чтении файла метрик шрифта. Если файл метрик содержит символы, которые не входят в набор символов ASCII, шрифт будет классифицирован как шрифт Unicode, в противном случае он классифицируется как шрифт ASCII.

Классификация шрифта важна, поскольку она влияет на обработку строки. Строки, использующие шрифт Unicode, должны быть в двухбайтовом формате, чтобы символы Unicode могли быть закодированы. Это также включает символы ASCII в строке. Строки, использующие шрифт ASCII, считаются однобайтовыми.

Рендеринг строки Unicode выполняется с использованием стандартных функций рендеринга строк,**Rt2dFontFlow()**и**Rt2dFontShow()**.

Вспомогательные функции

API шрифтов 2D Toolkit включает в себя некоторые дополнительные функции, помогающие определить, где и как отображать строку:

- **Rt2dFontGetHeight()**вернет реальную высоту растрового шрифта, как он будет выглядеть при рендеринге, принимая во внимание CTM шрифта и настройки представления. Использование высоты растрового шрифта при рендеринге текста гарантирует, что есть однозначное сопоставление с дисплеем; следовательно, рендеринговый размер текста остается независимым от текущих преобразований. (Для контурных шрифтов**Rt2dFontGetHeight()**функция будет*всегда*возвращает значение**1.0**.)

- **Rt2dFontGetStringWidth()**— это служебная функция, которая возвращает ширину заданной строки, если она была отображена с указанным шрифтом и высотой.

- **Rt2dFontSetIntergapSpacing()**устанавливает интервал между отдельными символами при отображении шрифта. Это позволяет размещать символы дальше или ближе друг к другу, чем обычно.

- **Rt2dFontIsUnicode()**— это служебная функция для запроса, классифицируется ли шрифт как шрифт Unicode, содержащий символы Unicode. Строки, использующие шрифт Unicode, должны быть в двухбайтовом формате. Строки, использующие чистый шрифт ASCII, должны быть в однобайтовом формате.

27.3 2D-объекты

27.3.1 Введение

В предыдущем разделе этой главы рассматривалось создание и использование кистей, путей и шрифтов. В этом разделе показано, как можно сохранять объекты, содержащие кисти, пути и шрифты, чтобы можно было повторно использовать и манипулировать ими. Группы объектов можно добавлять в сцены, где ими можно манипулировать вместе.

Есть четыре объекта, которые можно использовать для хранения данных кисти, пути и текста. Этими объектами можно манипулировать и визуализировать.

Четыре объекта:

1. **Формы**: фигуры представляют собой набор кистей и путей, которые объединяются с помощью узлов. Фигуры можно сохранять и добавлять в сцены. В этой главе уже рассматривалось, как создавать пути и кисти.

2. **Строки объектов**: строка объекта — это объект, содержащий текст. В этой главе уже рассматривалось, как создавать кисти и использовать шрифты.

3. **Выберите регионы**: область выбора — это область на экране. Они невидимы и могут использоваться, например, для кнопок и кликабельных областей. Области выбора не визуализируются и требуют другого объекта, например, фигуры, для представления их в сцене.

4. **Сцены**: сцена — это контейнер **Рт2д** объекты, которыми можно манипулировать и которые можно визуализировать. Сцена также является **Рт2д** объект.

27.3.2 Создание объектов

Ниже описано, как создать каждый тип объекта.

Создание формы

Следующие шаги описывают процедуру, необходимую для создания фигуры.

1. **Rt2dShapeCreate()** создает форму.

2. **Rt2dBrushCreate()** создает кисть, используя функции, описанные ранее в этой главе. Кисти должны быть созданы для заливки и обводки.

3. **Rt2dPathCreate()** создает путь.

   а. **Rt2dPathLock()** блокирует путь. Когда путь создан, он заблокирован.

   б) Постройте путь необходимой формы.

   в. **Rt2dPathUnlock()** открывает путь.

4.**Rt2dShapeAddNode()**складывает вместе форму, контур, заливку кистью и мазки кистью.**Rt2dShapeGetNodeCount()**можно использовать для определения количества узлов.

Пример кода

```
/*
  * Прямоугольник
  * /
{
        Rt2dObject      * форма;
        Rt2dPath     * путь;
        Rt2dBrush     * мазок кистью;

        Rt2dCTMPush();
        Rt2dCTMTranslate(WinBBox.w * 0.2f, WinBBox.h * 0.7f);

        форма   = Rt2dShapeCreate();
        путь   = Rt2dPathCreate();

        Rt2dPathLock(путь);
        Rt2dPathRect(путь, -0.1f, -0.1f, 0.2f, 0.2f);
        Rt2dPathUnlock(путь);

        strokeBrush = Rt2dBrushCreate(); Rt2dBrushSetRGBA(strokeBrush, &col[6],
        &col[6}, &col[2], &col[2]);

        Rt2dBrushSetWidth(strokeBrush, 0.03f);

        Rt2dShapeAddNode(форма, путь, NULL, strokeBrush);

        Rt2dObjectApplyCTM(форма);
        Rt2dObjectSetVisible(форма, ИСТИНА);

        Rt2dCTMPop();
}
```

Создание строки

Следующие шаги описывают процедуру, необходимую для создания строки объекта.

1.**Rt2dObjectStringCreate()**создает строку объекта.

2.**Rt2dObjectStringGetBrush()**получает кисть, используемую для рендеринга строки. Кисть влияет на цвет/заливку текста и ширину рисунка.

Пример кода

```
{
```

```
Rt2dObject      * нить;
Rt2dBrush     * мазок кистью;

Rt2dCTMPush();
Rt2dCTMTranslate(WinBBox.w * 0.2f, WinBBox.h * 0.2f);

/* установить шрифт */

строка = Rt2dObjectStringCreate("Привет, мир",
RWSTRING("helv"));

strokeBrush = Rt2dObjectStringGetBrush(string);
Rt2dBrushSetRGBA(strokeBrush, &col[6], &col[6], &col[2], &col[2]);

Rt2dBrushSetWidth(strokeBrush, 0.01f);

Rt2dObjectStringSetHeight(строка, 0.2f);

Rt2dObjectApplyCTM(строка);

Rt2dObjectSetVisible(строка, ИСТИНА);
Rt2dCTMPop();
}
```

Создание области выбора

Следующие шаги описывают процедуру, необходимую для создания области выбора.

1.**Rt2dPickRegionCreate()**создает область выбора.

2.**Rt2dPickRegionGetPath()**возвращает путь.

а.**Rt2dPathLock()**блокирует путь. Когда путь создан, он блокируется.

б) Постройте путь для требуемой области выбора.

в.**Rt2dPathUnlock()**открывает путь.

Пример кода

```
{
    Rt2dObject *pickregion;
    Rt2dPath *path;

    Rt2dCTMPush();
    Rt2dCTMTranslate(WinBBox.w * 0.45f, WinBBox.h * 0.45f);

    pickregion = Rt2dPickRegionCreate();

    path = Rt2dPickRegionGetPath(pickregion);
```

```
        Rt2dPathLock(путь);
        Rt2dPathRect(путь, 0.4f,0.4f,0.4f,                0,4ф);
        Rt2dPathUnlock(путь);

        Rt2dObjectApplyCTM(pickregion);

        Rt2dCTMPop();
}
```

Создание сцены

Rt2dSceneCreate()создает сцену. По умолчанию сцена заблокирована.

```
MainScene = Rt2dSceneCreate();
```

Работа со сценами более подробно описана в следующем разделе.

27.3.3 Добавление объектов в сцену

Объекты можно добавлять на сцену одним из двух способов:

Метод 1

1.**Rt2dSceneLock()**блокирует сцену. Сразу после создания сцены она по умолчанию блокируется.

2. Создание объектов с использованием**Rt2dShapeCreate()**, **Rt2dPickRegionCreate()**,**Rt2dObjectStringCreate()**или **Rt2dSceneCreate()**.

3.**Rt2dSceneAddChild()**добавляет**Rt2dObject**s в сцену. После того, как дочерний объект был добавлен в сцену, сцена становится владельцем объекта.

4.**Rt2dSceneGetChildCount()**используется для получения количества детей.

5. Звонок**Rt2dSceneUnlock()**чтобы разблокировать сцену.

Пример кода

```
        Rt2dSceneLock(Основная сцена);

        Rt2dObjectAddChild(Основная сцена, форма);

        Rt2dSceneUnlock(Основная сцена);


        /* форма могла сместиться во время разблокировки */
```

```
форма = Rt2dSceneGetChildByIndex(MainScene, 0);

Rt2dObjectMTMSetIdentity(Основная сцена);
```

Метод 2

1.**Rt2dSceneLock()**блокирует сцену. Сразу после создания сцены она блокируется.

2. Создание объектов с использованием**Rt2dSceneGetNewChildShape()**,
Rt2dSceneGetNewChildPickRegion(),
Rt2dSceneGetNewChildObjectString()или
Rt2dSceneGetNewChildScene().

3.**Rt2dSceneUnlock()**чтобы разблокировать сцену.

Пример кода

```
{
    Rt2dObject      * зигзаг;
    Rt2dPath     * путь;
    Rt2dBrush     * мазок кистью;

    Rt2dCTMPush();
    Rt2dCTMTranslate(WinBBox.w * 0.3f, WinBBox.h * 0.3f);

    Rt2dSceneLock(Основная сцена);

    зигзаг = Rt2dSceneGetNewChildShape(MainScene);

    /* установить путь */
    путь = Rt2dPathCreate();

    /* установить кисть */
    strokeBrush = Rt2dBrushCreate();

    Rt2dShapeAddNode(зигзаг, путь, NULL, strokeBrush);
    Rt2dObjectApplyCTM(зигзаг);
    Rt2dObjectSetVisible(зигзаг, ИСТИНА);

    Rt2dSceneUnlock(Основная сцена);

    /* форма могла сместиться во время разблокировки */ zigzag
    = Rt2dSceneGetChildByIndex(MainScene, 1);

    Rt2dCTMPop();
}
```

27.3.4 Сериализация объектов

Все объекты могут быть переданы в потоковом режиме. См. Явные функции потоковой передачи в *Сериализация*глава. Все объекты должны быть разблокированы перед трансляцией.

Для форм используйте:

- **Rt2dShapeStreamRead()**
- **Rt2dShapeStreamWrite()**
- **Rt2dShapeStreamGetSize()**

Для строк объектов используйте:

- **Rt2dObjectStringStreamRead()**
- **Rt2dObjectStringStreamWrite()**
- **Rt2dObjectStringStreamGetSize()**

Для выбора регионов используйте:

- **Rt2dPickRegionStreamRead()**
- **Rt2dPickRegionStreamWrite()**
- **Rt2dPickRegionStreamGetSize()**

Для сцен используйте:

- **Rt2dSceneStreamRead()**
- **Rt2dSceneStreamWrite()**
- **Rt2dSceneStreamGetSize()**

27.3.5 Манипуляции с объектами

Манипулирование объектом в сцене

Для манипулирования объектом в пределах сцены необходимо выполнить следующие шаги:

1.**Rt2dSceneUnlock()**разблокирует сцену. Это не обязательно, но рекомендуется.

2.**Rt2dSceneGetChildByIndex()**получает указатель на определенный объект или **Rt2dSceneForAllChildren()**получает указатели на все объекты.

3. Манипулируйте объектом, используя следующие функции: **Rt2dObjectMTMRotate()**, **Rt2dObjectMTMScale()**, **Rt2dObjectMTMTranslate()**, **Rt2dObjectSetColorMultiplier()**, **Rt2dObjectSetColorOffset()**, **Rt2dObjectSetDepth()**,**Rt2dObjectSetVisible()**.

   Строками объектов можно манипулировать с помощью **Rt2dObjectStringSetBrush()**,**Rt2dObjectStringSetFont()**, **Rt2dObjectStringSetHeight()**и**Rt2dObjectStringSetText()**.

4.**Rt2dObjectApplyCTM()**копирует текущую матрицу преобразования (CTM) в матрицу преобразования моделирования объекта (MTM). Это необходимо для применения изменений камеры, т.е. изменения точки обзора.

5.**Rt2dSceneUpdateLTM()**обновляет LTM, поскольку сцена MTM изменилась и может потребовать пересчета для рендеринга. Если LTM не требует обновления, например, для обнаружения столкновений, вы можете подождать до окончания рендеринга, поскольку функции рендеринга обновляют LTM.

6.**Rt2dSceneSetDepthDirty()**сообщает сцене, что при следующем рендеринге глубина объекта может измениться. Эта функция требуется, если **Rt2dObjectSetDepth()**использовался для манипулирования объектом.

Пример кода

```
Rt2dSceneUnlock(Основная сцена);

Rt2dSceneGetChildByIndex(Основная сцена, 2);

Rt2dObjectMTMScale(зигзаг, 0.6f, 0.6f);

цвет.красный =0.5f;
цвет.зеленый   = 1,0ф;
цвет.синий=     0,5ф;
цвет.альфа      = 1,0ф;

Rt2dObjectSetColorMultiplier(зигзаг, &цвет);
```

Манипулирование объектами вне сцены

Объектами, которые не находятся на сцене, можно управлять, как описано выше, и используя **Rt2dObjectCopy()**.

27.3.6 Рендеринг объектов

Все объекты, которые имеют**Rt2dObjectSetVisible(ИСТИНА)**можно визуализировать по отдельности или в составе сцены.

Функции рендеринга следующие:

- **Rt2dShapeRender()**
- **Rt2dObjectStringRender()**
- **Rt2dSceneRender()**

27.3.7 Уничтожение объекта

Объекты можно уничтожить одним из двух способов в зависимости от того, являются ли они частью сцены или нет.

Если объект не является частью сцены, используйте следующие функции для уничтожения объекта:

- **Rt2dObjectStringDestroy()**
- **Rt2dPickRegionDestroy()**
- **Rt2dShapeDestroy()**

Если объект был добавлен в сцену, то владельцем объекта является сцена. В этом случае используйте**Rt2dSceneDestroy()**уничтожить всю сцену и все дочерние объекты.

27.3.8 Объекты

Тип объекта

Для определения типа объекта можно использовать следующие функции:

- **Rt2dObjectGetObjectType()**
- **Rt2dObjectIsObjectString()**
- **Rt2dObjectIsPickRegion()**
- **Rt2dObjectIsScene()**
- **Rt2dObjectIsShape()**

Матричные функции

Матрицами можно манипулировать с помощью следующих функций:

- **Rt2dObjectGetLTM()**–вернуть мировую матрицу
- **Rt2dObjectGetMTM()**–возвратить объектную матрицу
- **Rt2dObjectMTMChanged()**–обновляет объект при изменении MTM

- **Rt2dObjectSetMTM()**–устанавливает MTM из**RwMatrix**

Использование областей выбора

RtPickRegionIsPointIn()используется для проверки 2d точки относительно выбранной области. Возвращает**истинный**если точка находится внутри области выбора или**ЛОЖЬ**если нет. 2d точка должна быть передана с использованием нормализованных экранных координат.

Например:

2D координата точки	Координаты экрана
0 1	0 640
1	480

27.4 Набор инструментов для работы с символами

Набор инструментов для набора символов (**RtCharset**) содержит упрощенный API вывода текста. Шрифт представляет собой моноширинный дизайн, который встроен в файл библиотеки инструментария и не может быть изменен без доступа к исходному коду. Основное назначение набора символов — отображение сообщений отладки, тестирования и диагностики во время выполнения.

Инструментарий поддерживает исключительно текстовые строки ASCII. Unicode и другие многобайтовые форматы не могут быть использованы.

Инициализация

Перед рендерингом любых текстовых строк необходимо инициализировать набор инструментов для работы с символами. Это выполняется с помощью вызова**RtCharsetCreate()**, который возвращает указатель на**RtCharset**объект.

Параметры для**RtCharsetCreate()**разрешить приложению выбирать цвета переднего плана и фона для шрифта, который является одноцветным растровым шрифтом. Чтобы переопределить эти цвета позже в вашем приложении, используйте **RtCharsetSetColors()**.

Дескриптор шрифта

Набор инструментов для работы с символами может быть перестроен с использованием других шрифтов, поэтому лицензиаты исходного кода RenderWare Graphics не должны предполагать, что один и тот же шрифт всегда будет встроен в двоичные файлы набора инструментов.

API предоставляет метод, который можно использовать для определения свойств встроенного шрифта:**RtCharsetGetDesc()**. Эта функция возвращает указатель на**RtCharsetDesc**объект, который содержит следующие элементы, описание встроенного шрифта:

- **считать**–количество символов в наборе
- **высота**–высота каждого символа в пикселях
- **высота плитки**–высота растра в символах
- **ширина плитки**–ширина растра в символах
- **ширина**–ширина каждого символа в пикселях

Все они относятся к типу**RwInt32**.

Поскольку шрифт всегда моноширинный, эту информацию можно использовать для определения области экрана, необходимой для отображаемой строки.

Рендеринг

Предусмотрены две функции рендеринга строк:**RtCharsetPrint()**и **RtCharsetPrintBuffered()**.

Обе функции принимают одни и те же параметры: указатель на допустимый **RtCharset** объект; указатель на саму текстовую строку и**x**и**y**координаты верхнего левого угла строки, которая будет отображена на экране.

Функция буферизованной печати,**RtCharsetPrintBuffered()**, не отобразит строку немедленно. Вместо этого вывод буферизируется. Этот буфер должен быть очищен вызовом**RtCharsetBufferFlush()**после завершения печати.

Поскольку процесс рендеринга шрифта использует треугольники непосредственного режима, **RtCharset** функции рендеринга должны быть размещены между вызовами **RwCameraBeginUpdate()**и**RwCameraEndUpdate()**.

Уничтожение шрифта

Когда шрифт, содержащийся в**RtCharset**объект, больше не требуется, его необходимо уничтожить с помощью вызова**RtCharsetDestroy()**.

27.5 Форматы файлов шрифтов

В этом разделе описываются три формата шрифтов, поддерживаемых 2D Toolkit.
. встретил("метрики") файлы.

Файлы метрик должны быть в формате UTF-8. Символы Unicode кодируются с использованием формата UTF-8 в разделе кода символа.

«Метрика 1» (растровое изображение)

Шрифт «Metrics 1» — это растровый шрифт, для которого требуется растровое изображение. После файла изображения можно указать необязательную маску. Имена файлов изображения и маски не должны содержать пробелов.**.встретил**Файл используется для определения доступных символов и их размеров. Значения позиции — это пиксельные координаты на изображении.

Формат файла «Метрика 1» следующий:

МЕТРИКИ1
<битовая карта шрифта> [<битовая карта маски шрифта>]
<базовая линия>
<код символа> <слева> <сверху> <справа> <снизу> <код символа> <слева> <сверху> <справа> <снизу> . . .

Ниже показан фрагмент файла Metrics 1. Этот фрагмент взят из "**cn12.мет**" файл, который определяет шрифт в стиле "Courier New" размером 12 пунктов с растровым изображением шрифта ("**cn12.bmp**") и маска ("**mcn12.bmp**").

```
МЕТРИКИ1
cn12.bmp   mcn12.bmp
5
  32     0     0    10    18 # ' '
  33    11     0    21    18  #  '!'
  34    22     0    32    18  #  '"'
  35    33     0    43    18  #  '#'
...
```

«Метрика 2» (растровое изображение)

"Metrics 2" также является форматом растрового шрифта, требующим растрового изображения. После файла изображения можно указать необязательную маску. Имена файлов изображения и маски не должны содержать пробелов.

«Метрики 2»**.встретил**файл содержит только список символов, доступных в битовой карте. Размеры каждого символа закодированы в шрифте изображения.

Каждый символ на изображении окружен границей. Это отмечает размер битовой карты символа. Начало битовой карты символа обозначается пикселем маркера в верхнем левом углу каждой границы. Поэтому важно, чтобы значения цвета пикселя маркера и границы не использовались где-либо еще, в противном случае символ будет использовать неверную область битовой карты для символа.

Тот же маркерный пиксель должен также присутствовать в левом нижнем углу для битовой карты первого символа. Это используется для определения высоты набора шрифтов. В противном случае шрифт не будет загружен правильно.

Область, используемая для битовой карты персонажа, отступает на 2 пикселя от четырех границ. Это сделано для того, чтобы предотвратить появление граничных пикселей при отображении персонажа.

«Metrics 2» также поддерживает несколько растровых изображений для шрифта, поэтому шрифт может быть распределен по нескольким растровым изображениям. Это может быть использовано для разбиения большого изображения на более мелкие части или для поддержки шрифтов с большим количеством символов, таких как греческие, китайские или японские кандзи.

Можно указать до четырех растровых изображений.

Формат файла «Метрика 2» следующий:

МЕТРИКИ2
<битовая карта шрифта> [<битовая карта маски шрифта>]
<базовая линия>
<персонажи>
[<битовая карта шрифта>] [<битовая карта маски шрифта>]
[<базовая линия>]
[<символы>]

Ниже показан фрагмент файла "Metrics 2". Этот фрагмент взят из "**иллюмин.мет**" файл, который определяет шрифт, использующий два битмапа/маски пары.

```
МЕТРИКИ2
illum0.bmp illum0m.bmp
10
ABCDEFGHIJKLMNOPQRSTUVWXYZ
золото1.bmp золото1м.bmp
10
abcdefghijklmnopqrstuvwxy;z,.!:?- 0123456789
```

«Метрики 3» (Контур)

«Metrics 3» — это формат контурного шрифта. Каждый символ использует ряд 2D-векторных команд для описания геометрической формы символа.

Каждый символ шрифта начинается со строки символов. Геометрическое описание начинается с**начинать**ключевое слово и заканчивается на**конец**Ключевое слово. Нет ограничений на количество 2D-команд для шрифта. Окончательный**переместиться** Команда используется для установки ширины символа — полученное местоположение будет позицией, в которой должен начинаться следующий символ.

Формат файла «Метрика 3»:

```
METRICS3
<font name>
'<character>'
начинать
переместиться   <x>   <y>
линияto   <x>   <y>
кривая к <x0> <y0> <x1> <y1> <x2> <y2> закрыть
путь
перейти к <x> <y>
конец
```

Ниже представлен фрагмент одного из файлов "Metrics 3". Этот фрагмент взят из "**гл.мет**" файл, который определяет контурный шрифт.

Символы, которые являются частью стандарта ASCII, но которые не определены в самом шрифте, должны быть определены с помощью одного**переместиться**инструкция (как для символа «!» в показанном примере), а не опущена полностью.

```
METRICS3
ч
' '
начинать
переместиться 0,999    0.0
конец
'!'
начинать
переместиться 0,28    0.0
конец
'"'
начинать
переместиться 0,09    0,215
линияto 0,115    0,175
линияto 0,14    0,125
линияto 0,16    0,075
линияto 0,185    0,015
линияto 0,205    - 0,035
линияto 0,24    - 0,085
линияto 0,275    - 0,09
линияto 0,305    - 0,075
линияto 0,32    - 0,025
линияto 0,33    0,025
линияto 0,325    0,08
линияto 0,285    0,125
линияto 0,24    0,155
```

линиятo 0,185 0,185
линиятo 0,12 0,22
близкая тропа
перейти на 0,998 0.0
конец

27.6 Резюме

В этой главе были рассмотрены как 2D Toolkit, так и Character Set Toolkit.

27.6.1 Набор инструментов 2D

Использует**Рт2д**объект.

Ключевые моменты

- Аппаратно ускоренная 2D-графика на поддерживаемых платформах
- *Кисти*определить текстуры и цвета
- *Пути*являются примитивами, которые могут быть любой комбинацией линий и кривых Безье
- Примитивы всегда визуализируются на камеру
- Для отображения путей в пространстве устройства требуется матрица преобразования.
- Растровые и контурные шрифты поддерживаются как для ASCII, так и для многобайтовых наборов символов (включая Unicode).

Пути и кисти

- Кривые в кистях должны быть выровнены перед рендерингом.
- Для прорисовки контура требуется кисть.
- Пути могут быть*погладил*или*заполненный*
- При обводке контура используется кисть, которая рисует вдоль контура, в результате чего получается линия.
- Заполнение контура создает окно в форме контура, через которое видна кисть.
- Кисть может содержать цвета и/или текстуру.
- Рендеринг кисти, определенной с использованием нескольких цветов, приводит к эффекту градиентной заливки.

Камера

Набор инструментов для 2D-графики работает путем рендеринга на камеру (**RwCamera**) объект. Функции рендеринга должны быть размещены между вызовами **RwCameraBeginUpdate()**и**RwCameraEndUpdate()**используя последнюю камеру, установленную для**Рт2д**операции через**Rt2dOpen()**или**Rt2dDeviceSetCamera().**

Текущая матрица трансформации

- 2D Toolkit использует стек матриц преобразования для преобразования своих 2D-вершин из локального (объектного) пространства в пространство устройства.
- Самая верхняя матрица в этом стеке называется матрицей текущего преобразования (CTM).
- CTM используется для*все*2D-рендеринг с помощью инструментария

Шрифты

- Поддерживаются два формата растровых шрифтов: «Метрика 1» и «Метрика 2».
- Поддерживается один формат контурного шрифта: «Метрика 3».
- Текстовые строки отображаются внутри ограничивающих рамок.
- Текст может быть выровнен по левому краю, по правому краю или по центру.

27.6.2 Rt2dObjects

В этом разделе объясняется, что такое объекты и как их можно создавать и использовать.

Всего имеется четыре объекта:

1. Формы

2. Строки объектов

3. Выберите регионы

4. Сцены

Этими объектами можно манипулировать, визуализировать и уничтожать по отдельности или как частью сцены.

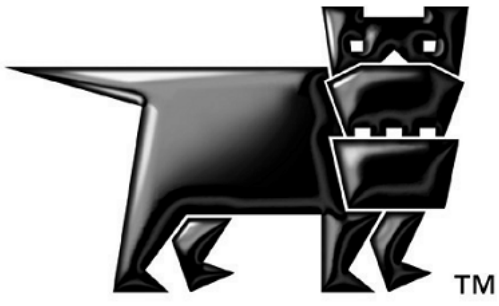
27.6.3 Набор инструментов для работы с символами

Ключевые моменты

- Набор инструментов для набора символов (**RtCharset**) в первую очередь предназначен для отладки и диагностики.
- Он быстрый, но негибкий и поддерживает только текст ASCII.
- Шрифт, используемый**RtCharset**встроен в библиотеку и не может быть изменен.

Глава 28

Маэстро



28.1 Введение

28.1.1 Обзор Маэстро

Maestro — это набор компонентов, которые можно использовать для проектирования и воспроизведения 2D-интерфейсов пользователя в играх. Пользовательский интерфейс обычно разрабатывается с использованием Macromedia Flash. Опубликовано**swf**файлы могут быть преобразованы в форму, пригодную для воспроизведения во время выполнения в RenderWare Graphics.

Примерами использования Maestro являются системы меню и экраны для навигации, настройки системы и выбора персонажей.

Маэстро не Флэш и не**swf**Player. Flash используется как путь авторинга для 2D пользовательских интерфейсов. Это похоже на экспортеры 3ds max и Maya, которые не поддерживают все функции 3ds max и Maya.

К компонентам, связанным с Maestro, относятся следующие, которые схематически показаны на следующей странице.

- **2dconvrt**

  инструмент командной строки, который используется для преобразования**swf**анимации в формат, который может быть прочитан RenderWare Graphics

- **Рт2д**

  набор инструментов, реализующий низкоуровневые 2D-функции

- **Rt2dAnim**

  инструментарий, который расширяет функции**Рт2д**набор инструментов для включения анимаций. Он содержит объект, называемый**Rt2dМаэстро**Этот объект координирует воспроизведение 2D-анимаций и может считаться нервным центром компонентов Maestro.

- **2dviewer**

  просмотрщик, для воспроизведения анимаций Maestro. Может использоваться для тестирования экспортированных **. а.н.м.**Файлы. Код предоставляется и демонстрирует, как воспроизводить**.а.н.м.** файлы.

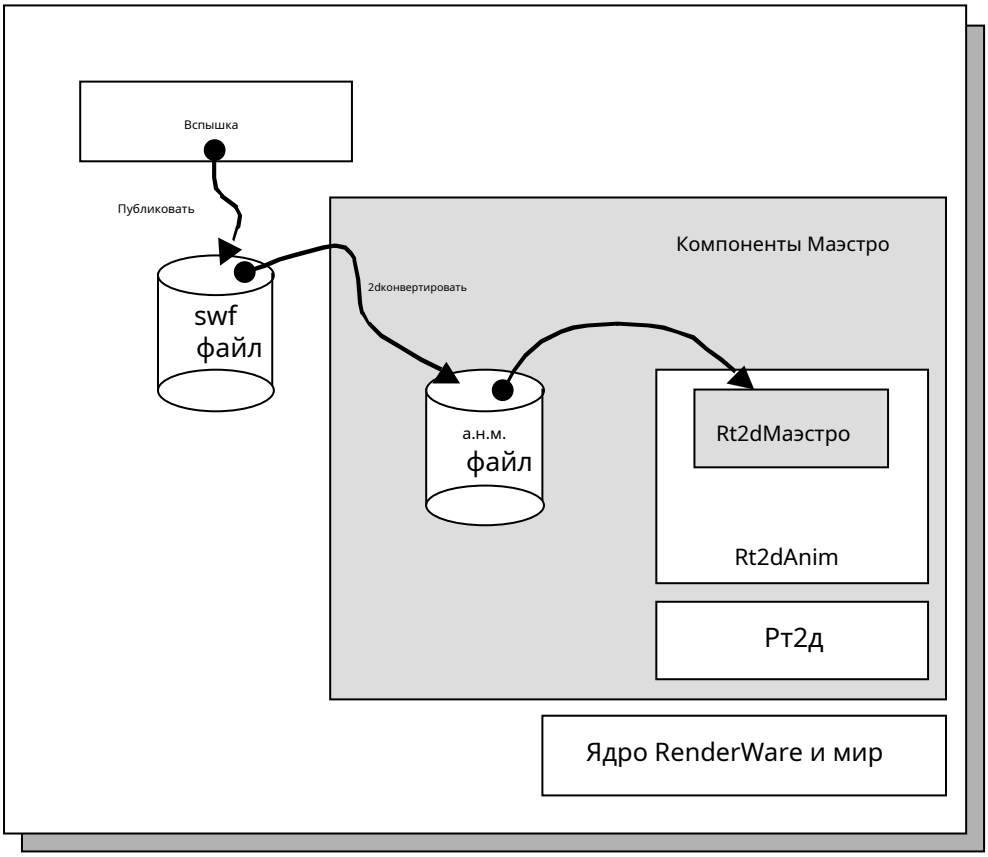
  По соглашению, активы, используемые игроком Maestro, имеют**.а.н.м.**расширение файла

- **маэстро1**

  пример, демонстрирующий использование Maestro для воспроизведения пользовательского интерфейса, созданного во Flash.

Функции Flash, поддерживаемые Maestro, реализованы в**Рт2д**и**Rt2dAnim**наборы инструментов. Инструмент 2dconvrt удаляет все функции**swf**файл, который не может поддерживаться во время выполнения.

В качестве простого примера, 2d toolkits не поддерживают звук. Любые звуковые эффекты, которые хранятся в**swf**файл не воспроизводится средой выполнения Компоненты Maestro. Звуковые эффекты по-прежнему могут быть вызваны; они просто не импортируются с использованием формата файла Maestro.



Когда анимация создается и сохраняется во Flash,**фла**файл создается. Чтобы иметь возможность использовать**фла**Для создания файла в RenderWare Graphics необходимо выполнить следующие шаги:

1. Опубликовать**фла**Файл. Это создает**swf**файл.

2. Преобразовать**swf**файл в**а.н.м.**файл.

3. Прочитайте и воспроизведите**а.н.м.**файл в RenderWare Graphics.

Что касается разработки на нескольких платформах, в идеале это должна быть одна и та же платформа..**фла**и.**swf**следует использовать файлы .

Последовательность пользовательских интерфейсов для консольных игр гораздо более ограничена, чем для ПК из-за отсутствия мыши. Рекомендуется создавать Flash-анимацию так, как если бы она предназначалась для консоли. Это определяет поток управления и компоновку анимации.

После завершения создания интерфейса, совместимого с консолью, для каждого экрана можно создать наложение для ПК.

Flash-контент на базе ПК в целом не будет похож на обычную продукцию Flash на ПК. Это связано с необходимостью оставаться независимым от платформы на уровне секвенирования.

28.1.2 Этот документ

Настоящий документ состоит из шести разделов.

Он предназначен как для художников, так и для программистов. Художникам следует прочитать разделы 28.1, 28.2 и 28.3. Программистам следует прочитать весь документ.

Первый раздел — это введение.

Во втором разделе перечислены те функции Flash, которые поддерживаются в RenderWare Graphics, и те, которые не поддерживаются.

Третий описывает процесс генерации и публикации контента. Подробно описывается дизайн пользовательского интерфейса для Maestro.

В четвертом показано, как импортировать контент и просматривать его с помощью RenderWare Graphics.

Пятая пятерка описывает механизм воспроизведения, в частности, какие хуки существуют для разработчика. Темы включают в себя, как получать информацию от и к проигрывателю во время воспроизведения.

В последнем разделе обобщены основные темы.

28.1.3 Другие ресурсы

Ссылка на API

- **Рт2д**набор инструментов
- **Rt2dAnim**набор инструментов

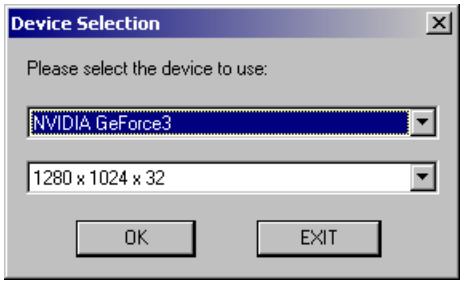
- Глава «2D Graphics Toolkit» в руководстве пользователя
- **2dconvrt**Документация по инструменту находится в docs\tools\2dconverter.pdf.
- **2dviewer**зритель
- **маэстро1**пример, краткое описание которого находится в разделе

28.1.4 Использованиемаэстро1Пример

Пример maestro1 доступен из меню «Пуск» Windows, в разделе *Программы RenderWare Graphics <платформа> Примеры SDK.*

Дважды щелкните по**маэстро1**папку, затем запустите либо **maestro1_d3d8.exe**, **maestro1_d3d9.exe**или**maestro1_opengl.exe** исполняемый файл.

Следующий диалог должен показать:



Нажмите «OK», и пример начнется со следующего:

На этом этапе пример будет реагировать на нажатия клавиш, таких как ENTER, BACKSPACE и клавиши со стрелками. ENTER соответствует SELECT, а backspace — CANCEL на консоли.

Нажатие клавиши ENTER на этом экране перенесет вас в основную часть примера.

28.2 Графика Flash и RenderWare

Конвертер Flash-to-.ANM был разработан для поддержки Flash 3. Это ограничивает действия, которые можно выполнять из более поздней версии Flash.

Мы выбрали версию Flash 3, поскольку она максимально соответствует функциям, которые поддерживает Maestro. Если вы используете более позднюю версию приложения Flash-авторизации, то вам необходимо экспортировать**swf**с использованием формата файла Flash 3. Диапазон поддерживаемых и неподдерживаемых функций в первую очередь обусловлен набором функций**Рт2д**и**Rt2dAnim**Наборы инструментов. Процесс преобразования из **swf**в формат RenderWare Graphics игнорирует те эффекты, которые наборы инструментов не могут поддерживать.

28.2.1 Поддерживаемые функции

Поддерживаемые функции импортера Flash:

- Статический 2D векторный контент
  - Стили линий
  - Стили сплошной заливки
  - Изогнутые пути
  - Альфа-прозрачность
  - Базовые текстовые строки
- Растровые заливки**.png**формат.

  - Для заливки битовых карт следует использовать битовые карты высоты и ширины, равные степени 2, например, 256*256 или 128*512. Большинство современных аппаратных средств требуют этого. В Maestro битовые карты, размер которых не равен степени 2, подвергаются повторной выборке. Художнику лучше самому изменять размер своих текстур, чем позволять этому происходить автоматически.
- Сопоставление шрифтов RenderWare Graphics со шрифтами Flash
- Анимации статического контента
- Использование 3D-оборудования для ускорения рендеринга
- Z-упорядочение/наслоение
- Интерактивность пользователя, включая кнопки
- Подмножество действий Flash 3, включая**ДействиеПерейти кМетки**, **ДействиеУстановитьЦель**(неродственный),**ДействиеПерейти к кадру**,**ДействиеGetURL**, **ДействиеСледующийКадр**,**ДействиеПредыдущийКадр**,**ДействиеИграть**,**ДействиеСтоп**.

  - Внутри Maestro есть соответствующие сообщения, которые могут быть перехвачены кодом разработчика. См. раздел 28.5.4.
- Действия могут быть вызваны как действия кадра или при переходах кнопок.

- Используется экспортная связь кнопок. Это позволяет использовать кнопку контроллера консоли нажимает для запуска действий, связанных с кнопкой Flash. Подробности см. в разделе 28.5.6.

- Комплексный инструмент импорта, позволяющий извлекать данные из различных источников простые статические объекты до сложного интерактивного контента

28.2.2 Неподдерживаемые функции

Ниже перечислены неподдерживаемые функции импортера Flash. Где это возможно, подробно описаны обходные пути.

- Специфические возможности Flash 4.0 и Flash 5.0.

- ActionScript появился во Flash 4.0. Замените его кодом, написанным на C. Это может быть вызвано Flash-запуском "**ПолучитьURL**"спусковой механизм. Смотрите раздел 28.5.4.

- Градиентная и радиальная заливка. Растровые заливки можно использовать для аппроксимации Градиент. Растровые изображения могут быть растянуты, поэтому подойдут и небольшие растровые изображения.

- Обрезка слоев.

- Фильмы в кнопках. Отдельный клип может быть запущен кнопкой прозрачная (но активная) кнопка. Это относится к состоянию MouseOver, поэтому фильмы не могут воспроизводиться в кнопке, когда мышь наведена на кнопку.

- Относительные цели в кнопках. Единственная поддерживаемая относительная цель — это прямой родитель спрайта.

- Растровое изображение заполняет .**jpg**или .**jpeg**Формат. Сохраняйте в формате без потерь один раз в Флэш. Для этого:

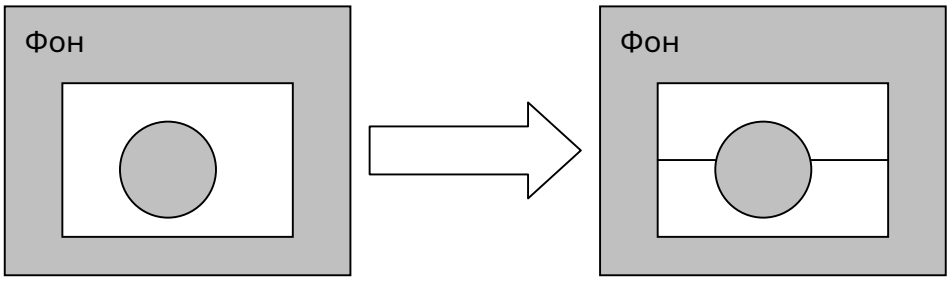
  - В строке меню выберите*Окно Библиотеки*.
  - Щелкните правой кнопкой мыши по имени изображения, чтобы открыть контекстное меню.
  - Выберите свойства из меню
  - Для «Сжатия» выберите «Без потерь».

- Звучит.**ПолучитьURL**«Триггеры могут использоваться для синхронизации звуковых событий.

- Шрифты Flash и TrueType. Используйте**шрифтalias.txt**файл для настройки Псевдонимы шрифтов RenderWare Graphics.

- Формы морфинга. Разбейте морфинг на отдельные позиции ключевые кадры. Рассмотрите возможность использования**Rt2dAnim**функция интерполяции (см.**Rt2dAnim**API ссылка) для обеспечения исключительно плавного воспроизведения анимации.

- Объекты с отверстиями, не касающимися края. Распространенная практика во Flash — создайте "рамку" с вырезом в центре. Поскольку 2D-фигуры вычерчиваются как большие непрерывные области в RenderWare Graphics, эти области непрозрачны. Разбейте объект на два отдельных объекта в разных слоях, и он будет вычерчен правильно.



- Сложные вогнутые криволинейные области. Обычно они будут отображаться правильно, но в некоторых случаях будут небольшие ошибки переполнения. Разделение кривых, которые формируют края вогнутых областей, должно помочь облегчить это.

28.3 Создание 2D-контента для использования в RenderWare Graphics

В этом разделе описывается:

- публикация Flash**фла**файл на Flash**swf**файл
- элементы пользовательского интерфейса
- создание и использование виртуального контроллера для тестирования пользовательского интерфейса, который будет управляться на консоли
- использование соглашений об именовании для упрощения разработки

Кроме того,**маэстро1**пример демонстрирует пользовательский интерфейс в действии. Приложение I показывает часть планирования последовательности, которая была выполнена для**маэстро1**.

28.3.1 Публикация SWF

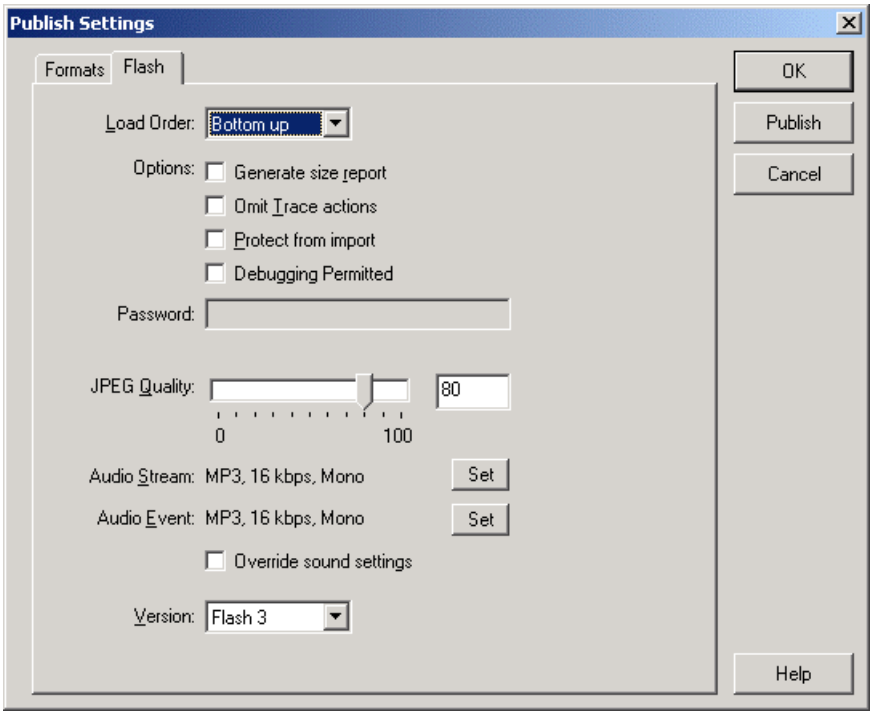
Вспышка**фла**файлы должны быть опубликованы как**swf**s для импорта в RenderWare Graphics. Чтобы преобразовать**фла**к**swf**подать**фла**Файл необходимо опубликовать в формате Flash 3.

RenderWare Graphics поддерживает подмножество Flash 3. В Flash 5 параметры публикации можно изменить и установить на Flash 3, а все параметры, которые не поддерживаются во Flash 3, будут выделены.

Чтобы гарантировать, что при создании чего-либо во Flash вы используете только функциональность Flash 3, в настройках Flash выполните следующие действия:

Файл *Настройки публикации* *Вкладка Flash*

Убедитесь, что установлена версия Flash 3.



При каждом запуске фильма будет появляться только окно объекта, сообщающее, были ли использованы какие-либо действия, не относящиеся к Flash 3.

Издательский

The**swf**может быть опубликовано двумя способами:

1. *Файл* *Настройки публикации* *Вкладка Flash*нажмите на*Публиковать*
2. *Файл* *Публиковать*

28.3.2 Элементы пользовательского интерфейса

Следующие элементы Flash могут быть полезны при создании пользовательского интерфейса:

- символы
- кнопки
- клипы из фильмов
- маркировка
- действия
- графика
- текст
- соглашения об именовании

Символы

Удобно настраивать большинство элементов в Flash Movies как символы. Это упрощает редактирование, особенно когда символ используется повторно.

Если идентичный фрагмент текста используется в нескольких сценах, можно определить символ, содержащий этот текст. Обновление этого символа приведет к обновлению текста во всех местах, где этот символ используется.

- Имена символов не сохраняются в **.swf** файлы и, следовательно, недоступны из кода.

Кнопки

ВАЖНО: Большинство консолей не имеют мыши. Дизайн большинства пользовательских интерфейсов на базе Maestro должен это учитывать. Они не могут быть созданы точно так же, как веб-интерфейсы, поскольку не существует концепции положения мыши и активной кнопки.

Кнопки используются для создания «виртуального контроллера», который может использоваться для имитации контроллера консоли. Кнопки предназначены для использования в качестве заполнителей для действий Flash, которые будут назначены им во время создания контента. Это описано в разделе 28.3.3.

В Maestro принято создавать кнопки только с состоянием нажатия, чтобы они не отображались. Изображения различных состояний кнопок должны зависеть от текущего кадра фильма, а не от состояния кнопки, зависящего от мыши.

Если вы не разрабатываете приложение для платформы с мышью, единственные события кнопок, которым полезно назначать действия, — это «нажатие» и «отпускание».

Созданные кнопки также должны иметь установленные свойства связи. Свойства связи используются, когда кнопки экспортируются из Flash и затем импортируются в RenderWare Graphics.

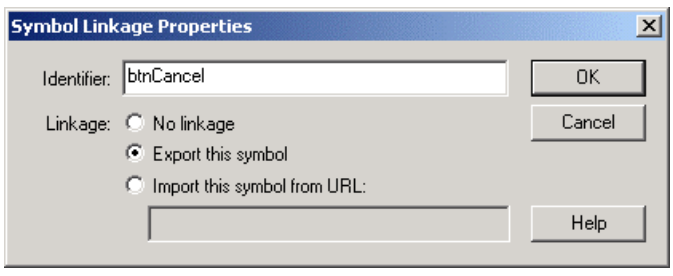
- Программы, использующие анимацию Maestro, должны ссылаться на кнопки по именам. Смотрите раздел 28.5.6, чтобы узнать, почему. Свойства связей можно использовать для экспорта имен кнопок.

Чтобы назначить строку идентификатора кнопке:

Выберите символ кнопки в библиотеке.

Щелкните правой кнопкой мыши по имени символа и выберите «Связь».

Выбирать *Экспортировать этот символ* и введите строку идентификатора.



Видеоклипы

Видеоклипы — это анимации, которые воспроизводятся внутри других анимаций. Они могут воспроизводиться внутри основной анимации или других видеоклипов. Они могут содержать любые элементы Flash, описанные в этом разделе.

Видеоклипы можно использовать для создания простых анимационных последовательностей. Затем их можно размещать как отдельные объекты в более крупных анимациях. Это упрощает дизайн более крупной анимации, разбивая ее на части.

Хотя видеоклипы обычно анимированы, их можно «остановить», чтобы они оставались на определенном кадре.

Это делает их полезными для реализации аспектов пользовательского интерфейса, таких как положения ползунков или отображаемые состояния элементов управления.

Для программиста видеоклипы примерно соответствуют**Rt2dAnim**с дополнительными функциями, такими как действия, метки кадров и кнопки.

Имена экземпляров клипов фильмов экспортируются, что означает, что конкретные фильмы можно искать в коде. Информацию о том, как это сделать, см. в разделе о строковых метках (28.5.3).

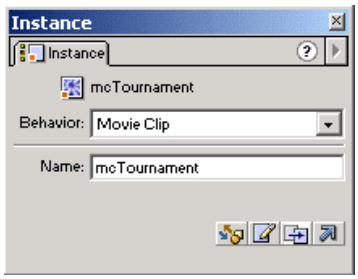
Поскольку названия фрагментов фильмов экспортируются, а названия символов — нет, фрагменты фильмов фактически являются единственным способом именования графики, чтобы их можно было найти в коде.

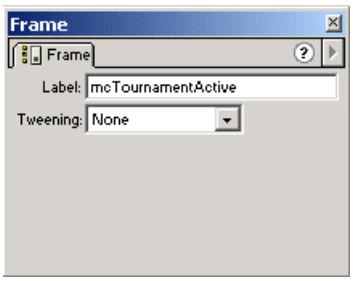
Возможные варианты использования фрагментов фильмов:

- поиск текущего кадра фильма. Вы можете использовать этот метод, чтобы получить позицию ползунка.**маэстро1**пример использует этот метод на экране редактирования имени игрока.
- определение местоположения объекта таким образом, чтобы можно было изменить его отображаемую позицию в коде
- нахождение объекта, чтобы можно было посмотреть и изменить текстуру
- нахождение текстовой строки, чтобы вы могли изменить ее в коде. Вы также можете сделать это, пройдя по основному дереву сцены, как это сделано в**маэстро1**пример поиска некоторого текста маркера.

Маркировка

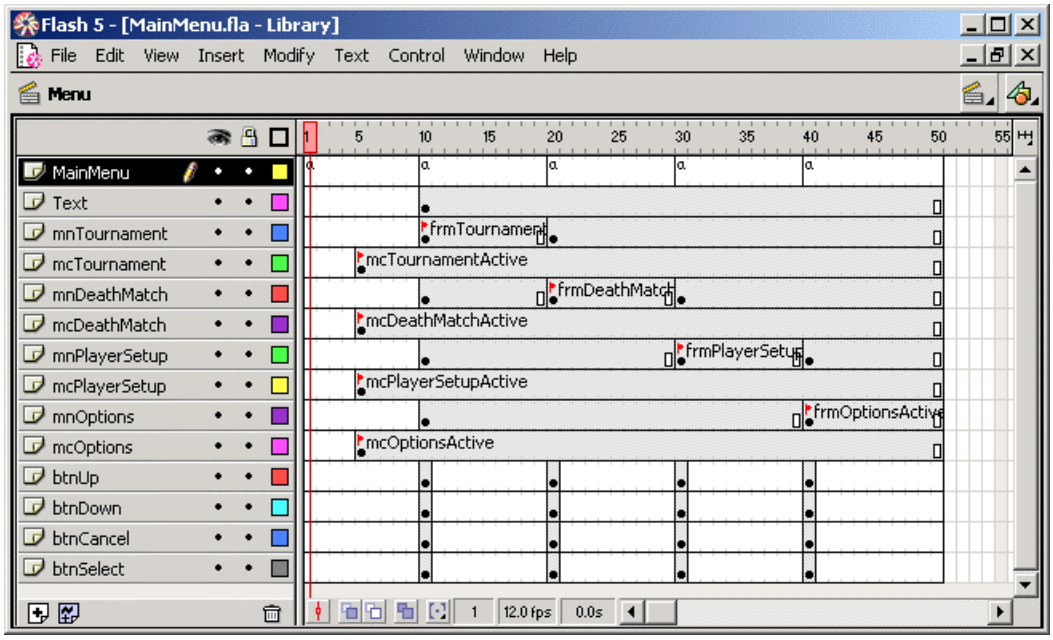
Клипы фильмов должны быть тщательно помечены. Метки экземпляров используются при создании действий для кнопок выбора для запуска клипов фильмов. Маркировка клипов фильмов позволяет управлять ими с помощью действий Flash из других клипов фильмов.





Метки доступны из RenderWare Graphics. См. раздел 28.5.3.

Отдельные кадры могут быть помечены в клипах. Действия могут использоваться для перехода воспроизведения к новому кадру.



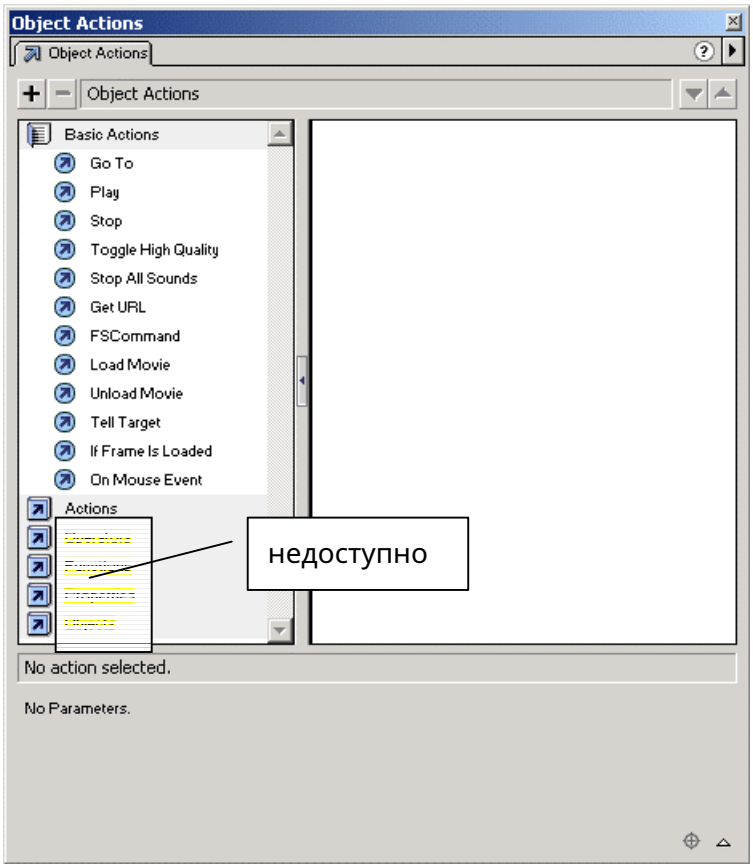
Действия

Если настройки публикации установлены на Flash 3 (см. Публикация SWF), Flash отображает недопустимые действия желтым цветом. Доступные действия остаются незатененными. Доступны все «Основные действия» и ограниченное количество «Действий».

Действия могут быть вызваны определенным кадром воспроизводимой анимации или переходом кнопки, например «вкл (нажатие)» или «вкл (отпускание)».

Полезные действия: «Перейти», «Воспроизвести», «Остановить», «Получить URL», «Сообщить цель», «По событию мыши» и «Получить URL». Другие действия не экспортируются и поэтому игнорируются.

GetURL особенно важен, так как он обеспечивает способ запуска действий в вызывающей программе. См.**маэстро1**пример и раздел 28.5.4.



Графика

Растровые заливки очень эффективны с точки зрения пространства и скорости.

Если**.swf**будет использоваться на нескольких платформах, может не получиться получить соотношение один к одному между битовой картой и экраном. Использование более высокого разрешения, чем строго требуется, может помочь.

Если объекты будут изменяться в размерах посредством анимации, векторные иллюстрации будут лучше. Иллюстрации в этой форме не зависят от размера.

Недостатком векторной графики является то, что она может стать неэффективной, если не будет отображаться в таком виде во Flash.

Распространенной практикой во Flash является преобразование растрового изображения в векторное. Этого определенно следует избегать в Maestro, поскольку такой объект будет гораздо менее эффективен для хранения и рендеринга.

Текст

Текст можно добавить с помощью текстового инструмента Flash. Необходимо соблюдать осторожность при выборе шрифтов, поскольку для корректного отображения их придется импортировать в RenderWare Graphics.

Для программистов,**2dconvrt**Утилита имеет возможность сопоставлять шрифты Flash со шрифтами RenderWare Graphics. Обратитесь к**2dconvrt**документацию для получения более подробной информации.

Соглашения об именовании

Для удобства и ясности полезно принять стандарт именования объектов Flash. Из названия может быть неясно, о каком типе объекта идет речь.

Добавление к имени указания типа объекта упрощает работу с Flash-файлами на протяжении всего процесса создания и импорта контента.

Приложение II содержит предлагаемые соглашения об именовании.

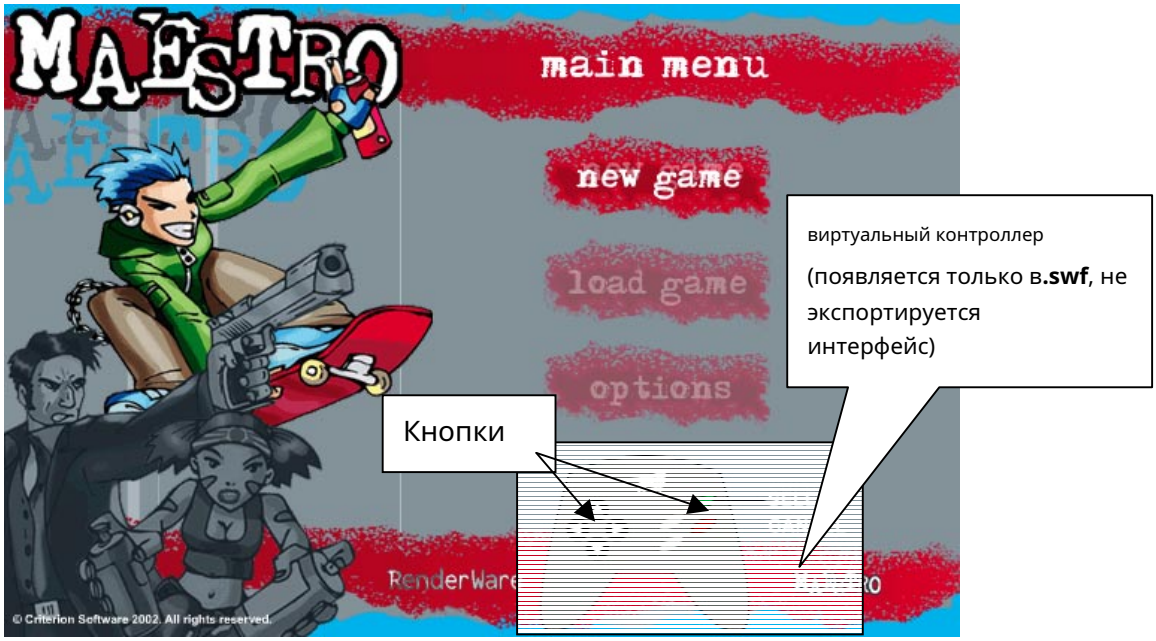
Программистам предлагается рассматривать это как аналог расширений файлов, а не венгерской нотации.

28.3.3 Виртуальные контроллеры и консольные арты

ВАЖНО: Большинство консолей не имеют мыши. Дизайн большинства пользовательских интерфейсов на базе Maestro должен это учитывать.

Из-за отсутствия указателя мыши кнопки, реагирующие на щелчки мыши, не могут использоваться на консолях. Изображения или анимации, отображающие картинки кнопок, могут использоваться, но обычно это не будут кнопки Flash.

Чтобы предоставить место для ввода данных пользователем, *виртуальный контроллер* необходимо использовать во время создания контента (см. рисунок ниже). Это *виртуальный контроллер* служит в качестве макета панели управления консоли. Имеет дублирующие версии кнопок направления контроллера (вверх, вниз, влево-вправо), а также кнопки выбора и отмены.



Имея *виртуальный контроллер* присутствует во время разработки позволяет *действия* для назначения нажатию кнопок. Например, нажатие «выбрать» на Frame1 с выделенными «опциями» может иметь действия, которые перенесут игрока на Frame10 с отображением «Меню опций».

В контенте, созданном для Maestro, «элементы управления» будут отображаться как «активные» на отдельных кадрах, а не элементы управления, реагирующие на наведение мыши и нажатие кнопок мыши.

The*виртуальный контроллер*Графика не будет экспортирована для производственной версии вашего произведения, но ее действия кнопок будут. Действия кнопок могут быть напрямую принудительно вызваны из кода, как описано в разделе 1.4.

Другие изображения кнопок, представленные на этом рисунке («новая игра», «загрузить игру», «опции»), не являются кнопками Flash; это обычная графика и анимация.

28.4 Импорт Flash-файлов в RenderWare Graphics

После публикации Flash-файла на**.swf**файл, это**.swf**необходимо преобразовать в форму, готовую для использования внутри RenderWare Graphics.

Преобразование выполняется с помощью инструмента командной строки,**2dconvrt**. Результатом преобразования является RenderWare Graphics**.a.н.м.**файл, который можно воспроизвести внутри программы или с помощью**2dviewer**программа.

Этот раздел предназначен для программистов и описывает

- конвертация Flash**swf**файл в RenderWare Graphics**a.н.м.**файл
- просмотр графики RenderWare**a.н.м.**файл с**2dviewer**программа

28.4.1 Импорт SWF в RenderWare Graphics

Вспышка**swf**s преобразуются в**a.н.м.**с использованием**2dconvrt**инструмент.

The**2dconvrt**инструмент подробно описан в**2dconvrt**документация по инструменту.

Использование инструмента 2dconvrt

The**2dconvrt**инструмент преобразует**swf**с к**a.н.м.**с.**a.н.м.**s можно воспроизводить и обрабатывать в RenderWare Graphics.

The**2dconvrt**tool — это инструмент командной строки; он не имеет графического пользовательского интерфейса.

По умолчанию,**2dconvrt**экспортирует растровые изображения и**a.н.м.**файл, содержащий все сцены, анимацию и информацию о взаимодействии с пользователем, необходимые для воспроизведения Flash-анимации.

Конвертация SWF

Используя команды, описанные в**2dconvrt**документация,**.swf**s можно преобразовать следующим образом в командной строке:

- Тип**2dconvrt *<пример>*.swf**

Это создает**a.н.м.**файл,***<пример>*.anm**

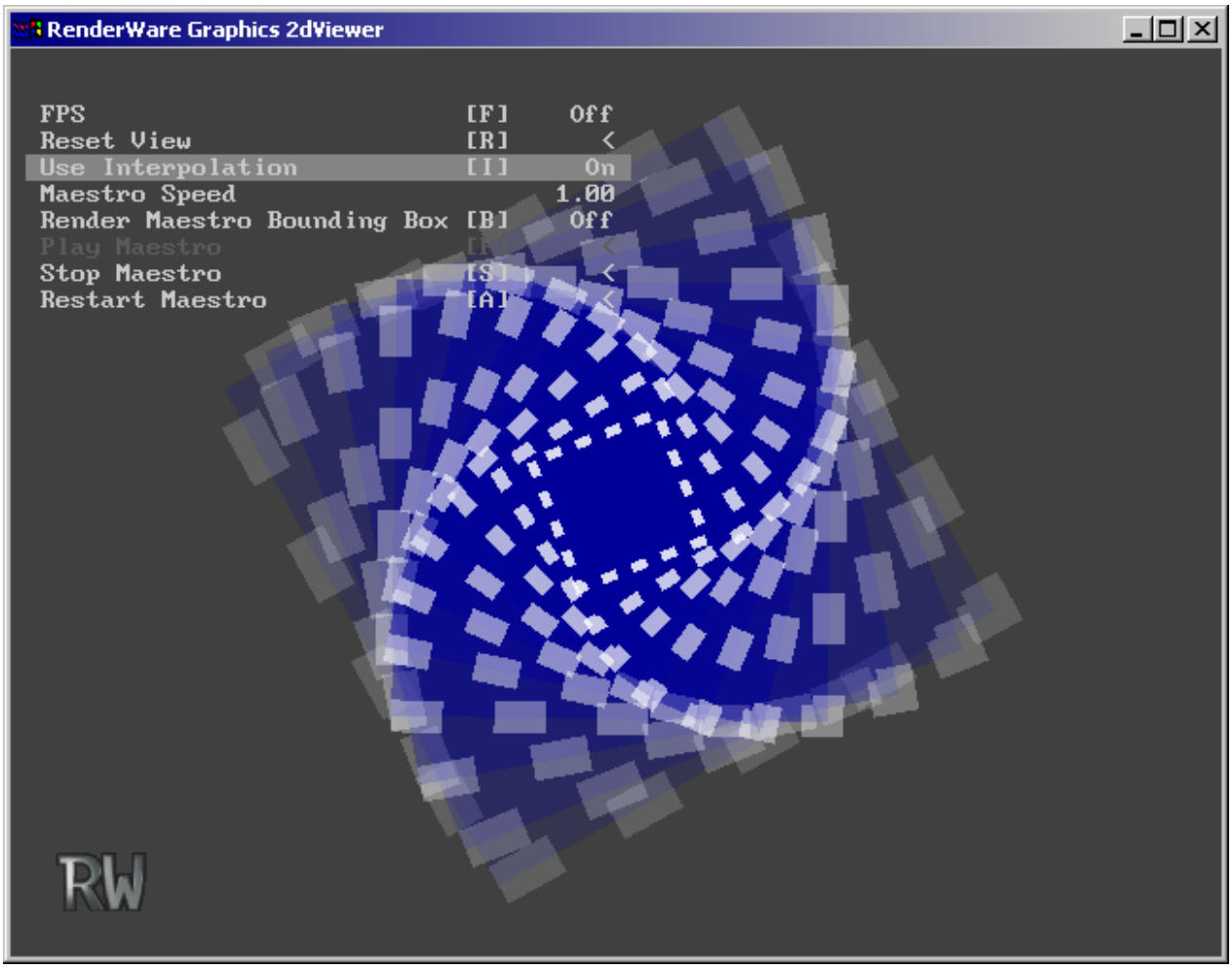
28.4.2 2D-просмотрщик

Как и в случае с 3D-частями RenderWare Graphics, мы предоставляем простой просмотрщик, позволяющий просматривать преобразованные**а.н.м.**файл, который можно легко просмотреть. Как специфичный для платформы, так и
Версия Win32 просмотрщика предоставляется в SDK. Исходный код этого приложения также включен.

Для отображения**а.н.м.**файл в RenderWare Graphics:

1. Запустите**2dviewer**инструмент.

2. Щелкните и перетащите**а.н.м.**файл на**2dviewer**.



2dviewer с анимацией по умолчанию,swirly.anm

28.5 Разработка с Maestro

В этом разделе подробно рассказывается, как разрабатывать с помощью Maestro.

Описана потоковая передача и воспроизведение анимации Maestro.

Также описаны методы получения и вывода информации из Maestro во время воспроизведения анимации. Это может быть удобно для связывания контроллеров или реагирования на события в анимации.

Maestro предоставляет эффективные и удобные средства получения дескрипторов внутренних данных по имени.

Наконец, описывается использование мышиных вводов. Это полезно в случае, если вы создаете интерфейсы для ПК.

28.5.1 Введение

API Maestro включает функции потоковой передачи, обновления времени и рендеринга.

Также представлен механизм передачи сообщений на основе обратного вызова, с помощью которого внутренние события Flash-анимации могут быть переданы обратно вызывающему коду. Пользовательский обработчик сообщений может быть включен в цепочку перед обработчиком сообщений по умолчанию. Это позволяет уведомлять пользовательский код о внутренних событиях анимации.

Пользователь также может публиковать внешние события в анимации через интерфейс передачи сообщений.**Rt2dMessage**структуры могут быть переданы Маэстро для заставить Maestro выполнять действия, не указанные во Flash-файле. **Rt2dMessage**s также можно перехватить, подключив пользовательский обработчик сообщений. Таким образом, сообщения можно проверять по мере их прохождения через обработчик обработки сообщений.

После создания и загрузки Flash-контента для воспроизведения в Maestro возникает проблема, как связать код с элементами, созданными в среде Flash. Например, если в файле Flash есть анимация с именем "/imcSubAnim1/Slider1/", как можно определить текущий кадр этой анимации?

Чтобы решить эту и другие проблемы, можно получить доступ ко многим элементам Flash по имени в экспортированном файле**..а.н.м.**файл. Это реализовано с помощью таблицы строковых меток, которая позволяет искать имена, экспортированные из Flash. Различные**Rt2dStringLabel**функции позволяют получить доступ к этим данным.

В этом разделе описывается, как использовать эти части**Rt2dAnim**библиотека, которая относится к**Rt2dМаэстро**.

Все API 2D-анимации основаны на функциональности иерархической сцены, предоставляемой в **Рт2д**Набор инструментов. См. главу руководства пользователя 2D Graphics Toolkit и**Рт2д**и**Rt2dAnim** разделы в справочнике API для получения дополнительной информации информация.

Поскольку у Maestro нет памяти, виртуальные клавиатуры, переключатели и ползунки требуют некоторой поддержки кода. В большинстве случаев это делается для того, чтобы получить информацию из анимации, например, факт перемещения ползунка. В других случаях может быть желательно внешне задать положение ползунка при входе на экран, отображающий этот ползунок.

Как правило, проще создать навигацию изнутри Flash, чем пытаться написать код C/C++, имитирующий навигацию системы меню.

28.5.2 Воспроизведение файла ANM в RenderWare Graphics

The**Rt2dAnim**набор инструментов содержит**Rt2dМаэстро**функции для воспроизведения**а.н.м.** файл в RenderWare Graphics.

Прежде чем что-либо**Rt2dМаэстро**функции могут быть вызваны,**Rt2dAnim**набор инструментов должен быть открыт с помощью**Rt2dAnimOpen**функция. При выключении, в конце концов **Маэстро**объекты были уничтожены,**Rt2dAnimЗакрыть**надо вызывать.

Rt2dМаэстроуправляет последовательностью 2D-анимации с помощью взаимодействия с пользователем.**Rt2dМаэстро**содержит сцену. Сцена содержит 2D-объекты, которыми можно манипулировать.

Следующее**Rt2dМаэстро**обсуждается функциональность::

1. Сериализация маэстро и маэстро сцены

2. Размещение сцены маэстро на дисплее

3. Применение обновлений времени

4. Обработка сообщений

5. Рендеринг

6. Уничтожение маэстро

Сериализация

Предполагается, что анимация Maestro будет предоставлена из внешнего источника, например, инструмента 2D-конвертации.**2dconvrt**. Как только анимация Maestro станет доступна в**.а.н.м.**файл, его можно транслировать в соответствии со стандартной практикой RenderWare Graphics.

Перед потоковой передачей следует убедиться, что пути к шрифтам и текстурам установлены правильно.**Rt2dМаэстро**. Обратите внимание, что конкретно Для анимации могут потребоваться шрифты или текстуры, но они могут находиться не в том же каталоге, что и сама анимация.

Следующий код воспроизводится в анимации, предполагая, что пути к шрифтам и текстурам были заданы заранее:

```
RwStream *поток = NULL;
Rt2dMaestro *маэстро = NULL;

поток = RwStreamOpen(rwSTREAMFILENAME, rwSTREAMREAD,
                                        <имя_потока>);

если( !поток      )
{
      возвращаться   (Rt2dMaestro *)NULL;
}

если (!RwStreamFindChunk(поток, rwID_2DMAESTRO,
                                    (RwUInt32 *)НУЛЬ, (RwUInt32 *)НУЛЬ)
{
      вернуть (Rt2dMaestro *)NULL;
}

маэстро    = Rt2dMaestroStreamRead(NULL,поток);

возвращаться   маэстро;
```

Размещение рендеринга Maestro на дисплее

Сцену маэстро необходимо разместить на дисплее. В примере ниже масштаб и переводы, необходимые для размещения Маэстро, были выбраны заранее. Вам нужно будет определить значения, которые будут работать с вашим пользовательским интерфейсом.

```
Rt2dObject *MaestroScene = Rt2dMaestroGetScene(Maestro);
Rt2dObjectMTMScale(MaestroScene, 0.002f, 0.002f);
Rt2dObjectMTMTranslate(MaestroScene, 100.0f, 100.0f);
```

- The**маэстро1**пример демонстрирует другой способ, которым это можно сделать.

Маэстро строит поверх**Рт2д**библиотека. Стандарт **Rt2dCTM*<операция>***Библиотечные функции также можно использовать для позиционирования отображаемого вывода Maestro.

Применение обновлений времени

Во время воспроизведения необходимо информировать Maestro о том, что время идет. Быстрое листание анимаций может осуществляться без необходимости обновления сцены.

Следующий шаг — дать команду Maestro обновить сцену, которая будет визуализироваться.

Если состояние сцены необходимо знать перед рендерингом (например, для обнаружения столкновений), его можно проверить после этого шага обновления сцены.

```
/* Сообщаем Маэстро, сколько времени прошло */
Rt2dMaestroAddDeltaTime(<Маэстро>,<deltaTime>);

/* Заставить Маэстро применить любые обновления к сцене, которую он
 * элементы управления. Это не обновляет LTM сцены
 * управляется Маэстро. Если обнаружение столкновений было
 * будет выполнено до рендеринга, Rt2dSceneUpdateLTM будет
 * должны быть вызваны первыми на место происшествия, полученные из
 * Rt2dMaestroGetScene
 */
Rt2dMaestroОбновитьАнимации(<Маэстро>);
```

Обработка сообщений

Способ связи Maestro с вызывающим кодом — через интерфейс обмена сообщениями. Сообщения могут передаваться в Maestro и из него.

Maestro может генерировать сообщения во время фазы обновления времени в ходе выполнения**Rt2dMaestroДобавитьДельтаВремя**.

Maestro может быть оснащен индивидуальным обработчиком сообщений. **Rt2dMaestroSetMessageHandler**для перехвата этих сообщений.

Сообщение также можно отправить Maestro с помощью**Rt2dMaestroPostMessage** API. Они не будут обработаны, пока**Rt2dMaestroProcessMessage**называется.

Раздел 28.5.4 подробно описывает этот процесс, как и**Rt2dAnim**Справочник API инструментария.

Рендеринг

После обновления позиций отображаемых объектов в сцене Maestro можно приступать к рендерингу этой сцены.

```
/* Если изменения точки обзора производятся извне, то базовая
 * уровень сцены должен быть обновлен. Это обычное дело в
 * большинство примеров, которые можно вращать, увеличивать и т. д.
 */
если( <ПросмотрИзменено>)
{
```

```
        Rt2dObject *MaestroScene =
Rt2dMaestroGetScene(<Маэстро>);
        Rt2dObjectMTMChanged(MaestroScene);
        <ПросмотрИзменено>=ЛОЖЬ;
}

/* Рисуем сцену, контролируемую маэстро */
Rt2dMaestroRender(<Маэстро>);
```

Разрушение

Маэстро уничтожен:

Rt2dMaestroУничтожить(Маэстро);

Сцена маэстро также разрушена, поскольку маэстро владеет сценой.

28.5.3 Метки строк

Rt2dStringLabelпредставляет собой структуру ссылки на строку, которая используется **Rt2dMaэстро**чтобы обеспечить возможность связывания внутренних и внешних данных по имени без снижения производительности.

Rt2dMaэстрохранит таблицу строковых меток. Когда**Rt2dMaэстро**создается или передается, заполняется таблица строковых меток. Вызывающая функция затем может искать интересующие ее имена в таблице. Индекс, указывающий, где было найдено имя, может использоваться в качестве дескриптора для идентификации этого имени.

-

Строки появляются в**.a.n.m.**файл. Проверить, что имена экспортированы, можно с помощью любого шестнадцатеричного редактора или VisualStudio.

Кроме того, в таблице хранится идентификатор, чтобы отметить, на какой тип данных ссылается имя. Пользователь может хранить дополнительные данные в таблице для каждого имени. Это обеспечивает удобное место для размещения обратных вызовов или расположения флагов. Затем это будет использоваться пользовательским обработчиком сообщений, подключенным к **Rt2dMaэстро**.

Типы сущностей Rt2dStringLabel

Строковая метка может обозначать один из нескольких различных типов сущностей в пределах **Rt2dMaэстро**. При поиске определенной строковой метки тип сущности может быть предоставлен программистом. Допустимые типы сущностей:

rt2dANIMLABELTYPEANIM	Метка анимации
rt2dANIMLABELTYPEFRAME	Этикетка рамки
rt2dANIMLABELTYPEBUTTON	Метка кнопки
rt2dANIMLABELTYPEURL	URL-метка; используется для расширений

rt2dANIMLABELTYPEURLэто тип, используемый для**Rt2dStringLabel**экспортируется для действия "GetURL" в сгенерированном Flash-контенте. Удобно для представления пользовательских именованных триггеров.

Обычно текст, содержащийся в метке строки, совпадает с текстом, указанным в редакторе Flash.

Имена экземпляров анимации, обозначаемые как**rt2dANIMLABELTYPEANIM**, являются особым случаем. Клипы Flash-роликов могут содержаться внутри других клипов, что приводит к появлению «дерева» именованных анимаций.

Имена экземпляров анимации могут быть установлены через панель «Экземпляр» во Flash. Они аналогичны по принципу действия именам каталогов.

Имена экспортируются в их полной квалифицированной форме. Символ "/" используется в качестве разделителя. Начальные и конечные разделители добавляются автоматически. Примеры показаны в таблице ниже.

АНИМАЦИЯ ЭТИКЕТКА	ОПИСАНИЕ
/	основная анимация
/imcSubMovie1/	поданимация основной анимации
/imcSubMovie1/imcSlider1/	анимация внутри первой поданимации
/imcSubMovie1/imcOnOff1/	анимация внутри первой поданимации
/imcSubMovie2/	поданимация основной анимации
/imcSubMovie2/imcSlider1/	анимация внутри второй поданимации

Использование функций доступа Rt2dStringLabel

The**Rt2dMaestroFindStringLabel**Функция может использоваться для поиска метки строки, хранящейся в таблице меток строк внутри**Rt2dМаэстро**.

```
Rt2dStringLabel     * этикетка;
RwInt32    индекс;
метка =    Rt2dMaestroFindStringLabel(
        <маэстро>,    rt2dANIMLABELTYPEURL,    "startGameTrigger",
        &индекс");
```

Возвращаемый индекс — это индекс строковой метки во внутренней таблице строковых меток Maestro. Этот индекс хранится как один из целочисленных параметров для нескольких сообщений Maestro (см.**Rt2dAnim**Подробную информацию см. в справочнике API).

Пользовательский обработчик сообщений, который отслеживает сообщения, проходящие через систему, может просматривать параметры этих сообщений. Некоторые сообщения используют индекс метки строки в качестве параметра, например**rt2dТИПСООБЩЕНИЯВОСПРОИЗВЕДЕНИЕ**

После того, как строковая метка найдена, ее свойства можно изменять непосредственно через возвращаемый указатель,

```
/* Сохранение некоторых пользовательских данных */
Rt2dStringLabelSetUserData(<метка>, и<startGameEvent>) ;
```

В этом случае,*<начатьGameEvent>*произвольные данные, указанные программистом. Например, там могут храниться идентификаторы или обратные вызовы. Эти сохраненные пользовательские данные затем могут использоваться в пользовательском дескрипторе сообщения.

Индекс внутри таблицы меток строк может быть сохранен и впоследствии использован для восстановления указателя.

```
/* Извлечение данных пользователя
*/ RwBool   *flag;
Rt2dStringLabel         * этикетка;
этикетка    = Rt2dMaestroGetStringLabelByIndex(
                                                    <Маэстро>,<индекс>);
флаг   = Rt2dStringLabelGetUserData(метка);
```

Сам указатель не следует хранить в течение длительного времени, так как **Rt2dMaэстро**может перемещать таблицу строк в памяти.

28.5.4 Сообщения

Rt2dMessageэто структура сообщения, которая используется**Rt2dMaэстро**для координации анимационных последовательностей.

Внешний код также может быть использован**Rt2dMessage**уведомлять Маэстро о внешних событиях.

Это осуществляется с помощью использования**Rt2dMaestroPostСообщения**и **Rt2dMaestroProcessMessages**функции. После публикации в **Rt2dMaэстро**, **Rt2dMessage**s удерживаются в очереди. Вызов **Rt2dMaestroProcessMessages** заставляет сообщения в очереди обрабатываться до тех пор, пока очередь не опустеет.

Обработка каждого сообщения**Rt2dMaэстро**осуществляется посредством внутреннего цикла сообщений.

Можно подключить пользовательский обработчик сообщений к**Rt2dMaэстро**. **Rt2dMessage**Структуры передаются этому обработчику. Их можно проверить, чтобы определить, когда произошли определенные события анимации, перед передачей обработчику сообщений по умолчанию.

Rt2dMessage

Rt2dMessageсодержит несколько фрагментов информации.

```
структура Rt2dMessage
{
    Rt2dMessageType     messageType; /* идентификатор сообщения */
    RwInt32             index;     /* индекс строкового имени
                                      * анимации сообщения
                                      * относится к
                                      * /
    RwInt32         intParam1; /* первый параметр (зависит от сообщения) */
    RwInt32         intParam2; /* второй параметр (зависит от сообщения)*/
```

};

The**Тип сообщения**определяет, как**Rt2dMaэстро**интерпретирует сообщение.

Ниже приведены наиболее важные сообщения:

rt2dТИПСООБЩЕНИЯGETURL

используется для запуска внешних событий из Flash-анимации

rt2dТИПСООБЩЕНИЯКНОПКАПОМЕТКЕ

используется для запуска действий, связанных с кнопкой внутри Flash-анимации.

Другие сообщения описаны в справочнике API.

Этот тип сообщения определяет, как следует интерпретировать другие параметры.

индексобычно используется для определения того, какая анимация внутри **Rt2dMaэстро** к которому относится сообщение. Это не сам номер анимации, а индекс имени метки строки для этой анимации.

Сообщения, отправленные извне, могут быть отправлены определенным анимациям.

Типы сообщений

Ниже приводится описание всех параметров сообщения и их основное использование.

Ко всем объектам всегда обращаются по их индексам, если не указано иное.

rt2dТИПСООБЩЕНИЯGETURL:

индекс	Индекс анимации
IntParam1	Индекс неиспользуемой StringLabel
IntParam2	GetURL

Это сообщение всегда отправляется из Maestro с намерением, что внешний код отреагирует на него.

Это сообщение следует использовать как механизм расширения. Во время генерации контента, "**ПолучитьURL**" действие может быть указано строкой, например "**GetURL("НачатьИгру")**".

Подключив пользовательский обработчик сообщений, "**ПолучитьURL**" сообщение может быть перехвачено и использовано для запуска внутриигровых событий. Обработчик по умолчанию ничего не делает с этим сообщением. Подробнее см. в разделе 28.5.3 **Rt2dStringLabel**c.

rt2dТИПСООБЩЕНИЯКНОПКАПОМЕТКЕ:

индекс	Анимация, должна быть –1
IntParam1	Индекс перехода StringLabel
IntParam2	кнопки

Это сообщение всегда отправляется внутрь Maestro из вызывающего кода через **Rt2dMaestroPostMessage**функция.

Это сообщение запускает действия, связанные с переходом кнопки на кнопке, идентифицированной строковой меткой. Его можно использовать для передачи внешних нажатий кнопок на определенные кнопки, идентифицированные именем, зарегистрированным в строковой метке. Подробнее о StringLabels см. в разделе 1.4.4.

Это сообщение должно быть передано всем видимым анимациям с помощью **Rt2dMaestroForAllVisibleAnimations**функция с соответствующим перезвонить.

Два перехода, которые представляют интерес, это **rt2dANIMBUTTONSTATEIDLETOOVERDOWN**и **rt2dANIMBUTTONSTATEOVERDOWNTOIDLE**. Они соответствуют «кнопке «кнопка нажата» и «кнопка отпущена».

Более подробную информацию см. в разделе 28.5.6.
rt2dТИПСООБЩЕНИЯКНОПКАПОМЕТКЕ.

Другие параметры сообщения описаны в справочнике API.

28.5.5 Подключение пользовательского обработчика сообщений

Ниже приведен пример пользовательского обработчика сообщений:

```
статическое Rt2dMessage *
ViewerMessageHandler(
        Rt2dMaestro *маэстро, Rt2dMessage *сообщение)
{
    переключатель(сообщение->типсообщения)
    {
    случай  rt2dТИПСООБЩЕНИЯSTOP:
        <МаэстроБег>=ЛОЖЬ; перерыв;

    случай  rt2dТИПСООБЩЕНИЯВОСПРОИЗВЕДЕНИЕ:
        <МаэстроБег>=ИСТИНА;
        перерыв;
    по умолчанию:
        перерыв;
    }

    вернуть Rt2dMessageHandlerDefaultCallBack(
                маэстро,сообщение);
}
```

Он будет предоставлен Maestro во время инициализации:

```
Rt2dMaestroSetMessageHandler(<Маэстро>,
                             ViewerMessageHandler);
```

В этом случае обработчик сообщений по умолчанию вызывается непосредственно в конце пользовательского обработчика.

В качестве альтернативы исходный обработчик сообщений можно было бы получить с помощью **Rt2dMaestroGetMessageHandler**. Возвращенный указатель мог быть сохраняется и позже вызывается в пользовательском обработчике сообщений через сохраненный указатель.

Преимущество этого подхода заключается в возможности объединения в цепочку нескольких пользовательских обработчиков сообщений.

28.5.6 Запуск переходов кнопок по имени

Иногда интерфейс «укажи и щелкни» с использованием мыши не подходит для использования на определенных платформах, в частности на консолях.

В этих случаях более целесообразно напрямую запускать события нажатия кнопки в интерактивной анимации. Например, если была нажата кнопка на контроллере консоли, было бы удобно запускать действия на определенной кнопке в анимации.

Flash можно заставить экспортировать имя кнопки для внешней связи. Если это сделано, кнопка может быть вызвана именем с помощью **rt2dТИПСООБЩЕНИЯКНОПКАПОМЕТКЕ**сообщение.

После загрузки Maestro, но желательно до воспроизведения, получается индекс кнопки с именем «btnDown».

```
RwInt32         искать;

Rt2dMaestroFindStringLabel(
    Maestro, rt2dANIMLABELTYPEBUTTON,
    "btnDown", &lookup
);
```

Позже во время воспроизведения может быть установлено сообщение, указывающее на нажатие кнопки. Это сообщение должно быть отправлено всем видимым анимациям, и для этой цели**Rt2dMaestroForAllVisibleAnimations**Можно использовать функцию API.

```
/* Определить структуру для использования внутри
  * Rt2dMaestroForAllVisibleAnimations
  * перезвонить */
typedef struct ButtonByLabelPacket ButtonByLabelPacket; struct
ButtonByLabelPacket
{
    RwInt32    кнопкаID;
    RwUInt32    animButtonState;
};

/* Обратный вызов для отправки сообщения в определенную анимацию */
Rt2dMaestro* BtnCallBack (Rt2dMaestro *maestro, Rt2dAnim *anim,
                          Rt2dAnimProps *props, void *pData)
{
    Сообщение Rt2dMessage;
```

```
        сообщение.ТипСообщения    =  rt2dMESSAGETYPEBUTTONBYLABEL; /* Это будет
        сообщение.индекс = -1;       заменено на */ /* «текущий» номер анимации
                                     при */ /* использовании вместе с */

                                     /* Rt2dAnimForAllVisibleAnimations */

        сообщение.intParam1
            = ((ButtonByLabelPacket *)pData)->buttonID;
                                     /* индекс метки кнопки */
        сообщение.intParam2
            = ((ButtonByLabelPacket *)pData)->animButtonState;

        /* Отправить сообщение */
        Rt2dMaestroPostMessages(maestro, &message, 1);

        вернуться маэстро;
    }

    . . .

    /* и код, который публикует сообщение для всех анимаций */
        . . .
        Пакет ButtonByLabelPacket;

        package.buttonID          =  искать;
        package.animButtonState   =  animButtonState;
        Rt2dMaestroForAllVisibleAnimations(
                                  <Маэстро>, BtnCallBack,(void*)&packet);

        . . .
         /* Заставить Maestro отреагировать на сообщение */
        Rt2dMaestroProcessMessages(<Маэстро>);
```

28.5.7 Взаимодействие с мышью на ПК

Пользовательские интерфейсы на основе Maestro изначально должны быть разработаны так, чтобы их можно было использовать на консолях, где мышь и указатель недоступны. Консоли страдают от ограничений, которых нет у ПК, поэтому проще портировать интерфейс с консоли на ПК, чем с ПК на консоль.

Для интерфейсов на ПК желательна интерактивность мыши. Возникает необходимость информировать Maestro об изменении положения мыши и статуса кнопок.

Maestro предоставляет сообщения с целью доставки обновлений положения мыши и состояния кнопок. Эти сообщения **rt2dТИП СООБЩЕНИЯКНОПКИ МЫШИСОСТОЯНИЕ**и**rt2dТИПСООБЩЕНИЯMOUSEMOVE**.

Хотя можно использовать**Rt2dМаэстро**Являясь всего лишь механизмом воспроизведения анимации, он предназначен для поддержки полной интерактивности пользователя в виде событий мыши.

Маэстро может быть проинформирован о нажатии кнопки мыши:

```
если (<leftButtonPushed>) {

        Rt2dMessage             сообщение;
        сообщение.ТипСообщения     = rt2dТИПСООБЩЕНИЯСОСТОЯНИЕКНОПКИ МЫШИ;
        сообщение.индекс = -1;
        message.intParam1 = (RwInt32)TRUE; /* Кнопка нажата */
        Rt2dMaestroPostMessages(<Маэстро>, &сообщение, 1);
}
```

или что мышь была перемещена:

```
message.messageType = rt2dMESSAGETYPEMOUSEMOVETO;
message.index = 0;

сообщение.intParam1    =  (RwInt32)MouseStatus->pos.x;
сообщение.intParam2    =  (RwInt32)MouseStatus->pos.y;

Rt2dMaestroPostMessages(Maestro, &message, 1);
```

После публикации сообщений необходимо уведомить Маэстро по телефону **Rt2dMaestroProcessMessages**функция.

```
Rt2dMaestroProcessMessages(Maestro);
```

Обратите внимание, что за один цикл обновления времени / обновления анимаций / рендеринга может быть отправлена только одна пара сообщений перемещения мыши и нажатия кнопки. Это связано с тем, что действия кнопок могут изменять набор видимых кнопок, которые должны быть проверены во время операции обработки сообщений.

28.6 Резюме

Maestro — это не Flash-плеер. Это цепочка инструментов импорта для 2D-пользовательских интерфейсов. Maestro поддерживает подмножество функций Flash 3. Для большинства неподдерживаемых функций есть обходные пути с некоторым описанием.

Были описаны шаги импорта пользовательского интерфейса, включая публикацию Flash-файла, преобразование его в форму, читаемую RenderWare Graphics, а также метод просмотра преобразованного файла.

Обсуждались элементы пользовательских интерфейсов, созданных с помощью Flash и Maestro. К ним относятся символы, слои, кнопки, статическая графика и видеоклипы, действия и текст. Также было подробно рассмотрено использование RenderWare Graphics API для воспроизведения 2D-анимаций и пользовательских интерфейсов.

Пользовательские интерфейсы на базе Maestro изначально должны быть ориентированы на консоли, поскольку их проще портировать с консоли на ПК, чем наоборот.

'Виртуальный контроллер' может использоваться как заглушка для действий, которые будут связаны с контроллером реальной консоли. Этот контроллер полезен во время тестирования.

Rt2dMaэстронакладывается на более простую систему анимации в**Rt2dAnim**, которая в свою очередь накладывается на иерархическую систему управления 2D-сценой в **Рт2д**набор инструментов.

Rt2dMaэстроОбъекты могут передаваться внутри стандартных потоков RenderWare Graphics.

Сообщения используются внутри**Rt2dMaэстро**для координации анимационных последовательностей.**Rt2dMaэстро**Цикл обработчика сообщений по умолчанию можно объединить с пользовательским обработчиком сообщений.

Активная часть цикла анимации состоит из трех этапов – информирование **Rt2dMaэстро**время прошло, обновление позиций отображаемых элементов и затем рендеринг этих элементов.

Rt2dStringLabelможет использоваться для регистрации и поиска строк, экспортированных из Flash.

Действия, связанные с нажатием кнопок, могут быть вызваны с помощью **rt2dТИПСООБЩЕНИЯКНОПКАПОМЕТКЕ**сообщение. Это позволяет связать кнопки консоли с именованными кнопками в анимации.

События мыши могут быть переданы**Rt2dMaэстро**через сообщения.

28.7 Приложение I – Планирование системы меню

В этом приложении подробно описывается

- планирование и создание системы меню во Flash

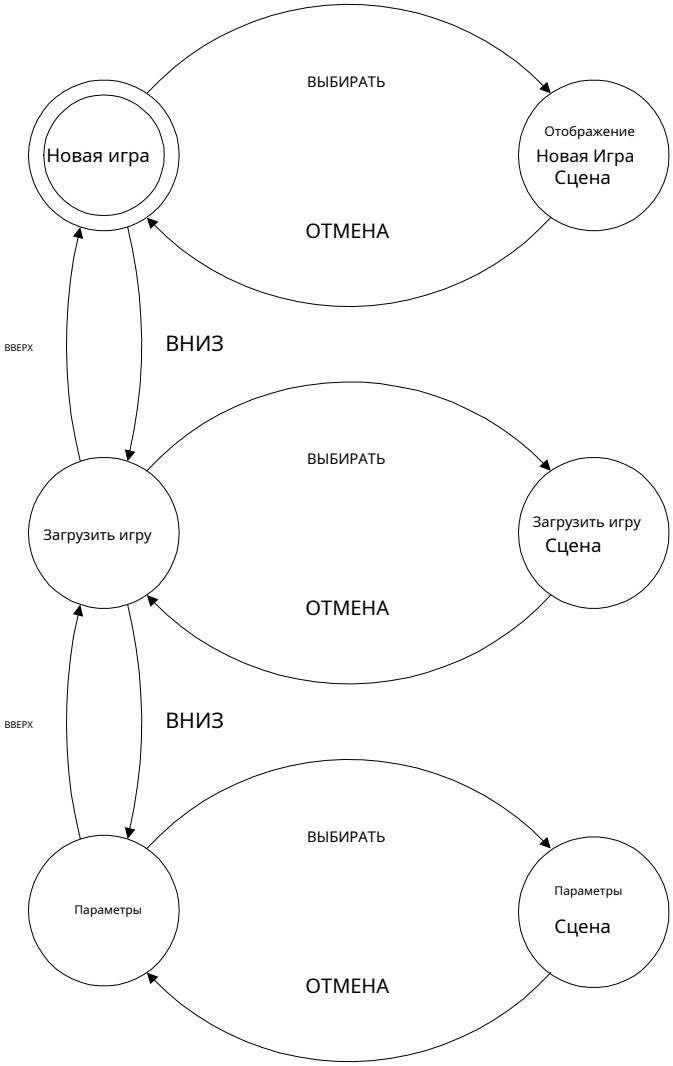
Планирование меню

При настройке меню хорошей идеей будет точно спланировать, что вы хотите сделать. В этом разделе Flash-файлы из**маэстро1**Примеры были использованы:**комбинация.fla**и**комбинация.swf**.

Взгляните на**комбинация.swf**чтобы увидеть, как организован фильм. Кнопки виртуального контроллера можно использовать для навигации по системе меню. При экспорте графика контроллера удаляется, но кнопки остаются, поэтому их можно вызывать из кода.

Диаграммы состояний могут быть полезным инструментом для использования на этапах планирования меню. Диаграмма состояний, основанная на**маэстро1**пример создан; см. далее страница.

Рамки главного меню



28.8 Приложение II – Соглашения об именовании

В этом приложении описывается предлагаемое соглашение об именовании объектов во Flash.

Для удобства и ясности полезно принять стандарт именования объектов Flash. Из названия может быть неясно, о каком типе объекта идет речь.

Добавление к имени указания типа объекта упрощает работу с Flash-файлами на протяжении всего процесса создания и импорта контента.

ПРЕФИКС	ТИП ДАННЫХ/СИМВОЛА
кнопка	кнопка
фрм	рамка
гр	графический
мс	отрывок из фильма
имк	именованный экземпляр фрагмента фильма
mn	меню
смн	подменю
текст	текст

Рекомендуется, чтобы имена:

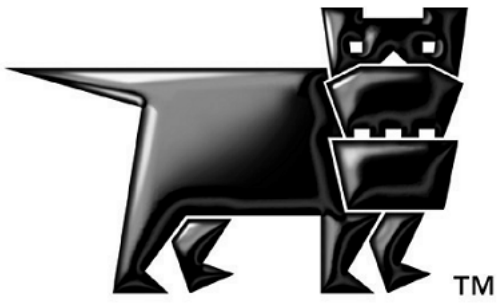
- избегайте пробелов и специальных символов
- начните имя переменной или объекта с буквы
- используйте уникальные имена
- использовать систему для определения типа и области применения

См.www.macromedia.com для получения более подробной информации.

Глава 29

Данные пользователя

Плагин



29.1 Введение

Плагин User Data позволяет расширять определенные объекты RenderWare Graphics с помощью определяемых пользователем структур данных. Расширяемые объекты:

- **RpWorld**
- **RwFrame**
- **RpGeometry**

RenderWare Graphics поддерживает экспорт пользовательских данных с помощью инструментов экспортера пакета моделирования.

Типичные области применения включают:

- Определение физических характеристик геометрии модели, таких как физические свойства полигонов
- Обозначение жесткости суставов в скелете модели со скином
- Настройка атрибутов объектов RenderWare Graphics, специфичных для приложения, таких как количество сущностей, разрешенных в секторе мира, или необходимость применения специального эффекта к определенной модели

29.2 Возможности плагина

Плагин пользовательских данных представлен**RpUserData**и должен быть подключен, как и любой другой плагин RenderWare Graphics, перед использованием.

29.2.1 Пользовательские массивы данных

Плагин User Data предоставляет API, который позволяет приложению определять структуры данных в терминах одного или нескольких из трех примитивных типов:*интс*, *реальи струны*. Эти типы хранятся в*массивы*.

Массив является основным типом данных в**RpUserData**плагин, так как нет явного **RpUserData**объект. Все пользовательские данные хранятся в массивах, которые прикрепляются к нужному объекту.

Каждый массив может содержать только один тип примитива, поэтому, если вашему приложению необходимо хранить более одного типа данных, ему потребуется определить эквивалентное количество массивов каждого типа.

Массив содержится в структуре, содержащей следующие элементы:

- А*имя*
  – доступ получен**RpUserDataArrayGetName()**
- А*формат данных*
  – доступ получен**RpUserDataArrayGetFormat()**
- Ан*количество элементов*
  – доступ получен**RpUserDataArrayGetNumElements()**
- Один или несколько*записи массива*
  – доступ осуществляется с помощью одной из функций доступа, перечисленных в разделе 29.4.2

Следует отметить, что**RpUserData**не прилагает никаких усилий для связывания записей массива с определенными вершинами или другими сущностями. Плагин просто хранит массивы данных; приложение должно сохранять любую связь.

Имена массивов

Каждый массив поддерживает*имя*элемент. Имя может быть любой строкой символов ASCII, завершающейся нулем.

Поскольку плагин User Data не хранит никаких дополнительных данных о том, с чем связаны данные (например, вершины или полигоны), имя часто используется для хранения этой информации. При работе с экспортерами моделей RenderWare Graphics поле имени обычно заполняется именем свойства.

Формат данных

Этот элемент определяет формат массива данных — будь то массив целых чисел, действительных значений или строк.

Формат данных задается с помощью одной из трех констант:

- **rpINTUSERDATA**–для 32-битных целочисленных данных
- **rpREALUSERDATA**–для 32-битных реальных данных (с плавающей точкой)
- **rpSTRINGUSERDATA**– (**беззнаковый символ ***), используется для строк

Количество записей

Это определяет длину массива.

Массив может содержать одно или несколько значений одного типа.

Записи массива

Записи массива представляют собой пользовательские данные.

Это непрозрачный тип данных, поэтому записи должны добавляться и обрабатываться исключительно через API плагина User Data. При добавлении записей массива строк копируется строка пользователя, а плагин обрабатывает выделение памяти.

29.3 Хранение данных пользователя

Пользовательские данные обычно создаются художниками в рамках их пакета моделирования. Инструменты экспортера RenderWare Graphics поддерживают экспорт таких пользовательских данных из 3ds max и Maya.

Кроме того, разработчики могут использовать специальные инструменты для добавления специальных данных как в автономном режиме, так и в качестве постобработки или во время выполнения.

29.3.1 Экспортеры

Экспортеры моделей, поставляемые с RenderWare Graphics, предоставляют средства вставки пользовательских данных, введенных в моделере, в экспортируемый файл модели. Однако разные экспортеры поддерживают это по-разному.

Поддержка пользовательских данных в двух основных пакетах моделирования описана ниже. Полные сведения можно найти в руководстве художника для соответствующего пакета моделирования.

3ds макс

Пакет моделирования 3ds max поддерживает экспорт пользовательских данных только на **RwFrame** объекты. Пользовательские данные можно экспортировать с помощью одного из двух методы:

1. *Свойства пользователя*. Это создает массив**rpSTRINGUSERDATA**записи.

2. *Пользовательские атрибуты*. Это наиболее гибкий вариант. Имена массивов будут получены из меток атрибутов, а тип атрибута будет преобразован в один из трех типов массивов пользовательских данных. Однако настройка и использование могут занять больше времени.

Диалоговое окно экспортера позволяет художнику выбрать, каким из двух методов будут созданы пользовательские данные. При необходимости можно использовать оба метода.

Майя

Экспортер графики RenderWare для Maya поддерживает включение пользовательских данных в**RwFrame**,**RpGeometry**и**RpWorldSector**объекты.

Пользовательские данные вводятся художником с помощью*Слепые данные*Механизм в Maya. Экспортер RenderWare Graphics для Maya преобразует типы в эквивалентные типы пользовательских данных во время фазы экспорта.

Поскольку Maya использует такие методы, как сварка вершин и интерполяция в процессе экспорта, прямое однозначное соответствие между данными модели и пользовательскими данными не гарантируется.

29.3.2 Процедурная генерация

Пользовательские данные могут быть созданы процедурно с использованием**RpUserData**API. Процесс включает в себя следующие этапы:

1. выделение места для данных в целевом объекте

2. получение указателя на массив

3. заполнение массива данными

В этом разделе описывается процесс на примере добавления массива пользовательских данных в**RpGeometry**объект.

В примере кода предполагается:

- **RpUserData**и**RpWorld**плагины были прикреплены;
- the **моягеометрия**объект уже создан и инициализирован.

Выделение массива

Пространство для массива пользовательских данных должно быть выделено на объекте до того, как он может быть заполнен. Это достигается с помощью одной из трех функций:

ДЛЯ:	ИСПОЛЬЗОВАТЬ:
Геометрические объекты	**RpGeometryAddUserDataArray()**
Мировые Сектора	**RpWorldSectorAddUserDataArray()**
Рамки	**RwFrameAddUserDataArray()**

В нашем примере используется**RpGeometry**объект, поэтому используется первая функция. Поскольку нам понадобится индекс, возвращаемый функцией**RpGeometryAddUserDataArray()** функция позже, мы сохраняем это в**arrayIndex**—ан**RwInt32**переменная.

Сначала необходимо инициализировать некоторые константы и переменные:

```
# определить NUMARRAYELEMENTS 10

/* Данные, которые мы хотим сохранить в массиве: */ RwInt32
myData[]={ 1, 3, 5, 7, 9, 5, 2, 3, 1, 19, 21 };

char * arrayName = "Пример массива";
RpUserDataArray      * мойМассив;
RwInt32 я,       arrayIndex;
RpGeometry    * моягеометрия;

. . .
```

Далее,**RpGeometry**объект необходимо инициализировать с помощью вызова плагина World**RpGeometryCreate()**Функция. (См.*Динамические модели*(Подробнее об этой функции см. в разделе).

Приложение теперь может создавать пространство для массива пользовательских данных на **RpGeometry**объект:

```
arrayIndex =RpGeometryAddUserDataArray( myGeometry, arrayName,
rpINTUSERDATA, NUMARRAYELEMENTS );
```

Заполнение массива

На данном этапе выделено место под массив.

Флаг, переданный в третьем параметре вызова **RpGeometryAddUserDataArray()** сообщает функции определить пространство для массива целых чисел, но массив пока не содержит никаких значений. Чтобы заполнить этот массив, нам нужно получить указатель на новый массив.

К объекту можно добавить несколько массивов пользовательских данных, поэтому **RpUserData**API предоставляет**...GetUserDataArray()**функции для доступа к ним по индексу:

```
мойМассив =RpGeometryGetUserDataArray(myGeometry, arrayIndex);
```

Предполагая,**моймассив**не содержит нулевого значения, что указывает на ошибку, приложение теперь может заполнить массив.

В этом примере массив заполняется путем копирования данных из**RwInt32**множество, **моиДанные[]**. Каждый из трех типов пользовательских данных — целое число, вещественное число и строка — имеет свои выделенные функции доступа. В этом случае нам нужно использовать **RpUserDataArraySetInt()**:

```
для (i = 0; i< ЧИСЛОВЫЕЭЛЕМЕНТЫ; i++) {

    RpUserDataArraySetInt(мойМассив, я,     моиДанные[i]);
}
```

29.3.3 Доступ к данным пользователя

Извлечение данных из массива, прикрепленного к произвольному объекту, обычно выполняется во время выполнения. Например, массив пользовательских данных, представляющий полигоны в секторе мира, может быть опрошен во время обнаружения столкновений.

В следующем примере массив пользовательских данных, содержащий **RpWorldSector**объект находится и к нему осуществляется доступ.

Номер индекса массива заранее неизвестен, поэтому пример найдет нужный массив, проверив его имя.

- В этом примере инициализация переменных опущена для ясности.

Нахождение массива

Предполагая,**мирСектор**содержит указатель на допустимый**RpWorldSector**объект, содержащий наш массив пользовательских данных, мы должны сначала определить, сколько массивов содержится в нем. Это достигается вызовом **RpWorldSectorGetUserDataArrayCount()**:

```
numUserDataArrays =
        RpWorldSectorGetUserDataArrayCount(мировой сектор);
```

Приложение теперь может проходить по массивам в секторе мира и проверять имя каждого из них. Вызов функции, необходимый для этого, — **RpWorldSectorGetUserDataArray()**:

```
для ( i=0; i<numUserDataArrays; i++ ) {

  userDataArray=RpWorldSectorGetUserDataArray(мировой сектор,        я);
```

Проверка имени массива

В этом примере нас интересует только массив с именем «Slipperiness», поэтому стандартная функция C используется для сравнения со строкой, возвращаемой функцией **RpUserDataArrayGetName()**:

```
если (strcmp(RpUserDataArrayGetName(пользовательскиеДанныеArray),
        "Скользкость")==0)
{
```

Проверка формата массива

Чтобы определить, представляет ли массив интересующие его данные, программа выполняет следующие тесты:

- Является ли формат массива**rpINTUSERDATA**тип?
- Соответствует ли количество элементов в массиве количеству полигонов в объекте мирового сектора?

```
/* Массив найден. Проверка корректности данных: */ if (
        RpUserDataArrayGetFormat(userDataArray)==
        rpINTUSERDATA &&
        RpUserDataArrayGetNumElements(userDataArray)==
        worldSector->numPolygons)
```

Если массив проходит эти тесты, остается только извлечь данные.

Извлечение данных

Каждый поддерживаемый пользовательский тип данных — целое число, вещественное число и строка — поддерживается выделенной функцией доступа. В этом примере мы получаем доступ к целым числам, поэтому мы используем**RpUserDataArrayGetInt()**функция:

```
        {
                скользкость     =  RpUserDataArrayGetInt(userDataArray,
                                   полиИндекс);
        }
  }
}
```

Функции доступа для различных типов данных перечислены в таблице ниже.

ТИП МАССИВА	ФУНКЦИИ ДОСТУПА
rpINTUSERDATA	**RpUserDataArrayGetInt()** **RpUserDataArraySetInt()**
rpREALUSERDATA	**RpUserDataArrayGetReal()** **RpUserDataArraySetReal()**
rpSTRINGUSERDATA	**RpUserDataArrayGetString()** **RpUserDataArraySetString()**

29.3.4 Удаление данных пользователя

Ваше приложение может захотеть удалить пользовательские данные, которые были добавлены в объект RenderWare Graphics. Например, вы можете хранить данные, которые преобразуются во внутренний формат при запуске приложения, и вам нужно будет удалить память сохранения пользовательских данных позже.

Удаление массива

ДЛЯ:	ИСПОЛЬЗОВАТЬ:
Геометрические объекты	**RpGeometryRemoveUserDataArray()**
Мировые Сектора	**RpWorldSectorRemoveUserDataArray()**
Рамки	**RwFrameRemoveUserDataArray()**

Помимо указателя на объект, содержащий массив, эти функции принимают индексный номер для удаляемого массива пользовательских данных. Этот индексный номер должен быть получен либо путем сохранения индекса, возвращаемого функциями add, либо путем поиска массивов по заданному имени, как подробно описано в разделе Доступ к пользовательским данным.

Функции удаления пользовательских данных возвращают указатель на объект, из которого был удален массив, в случае успешного выполнения и NULL в случае возникновения ошибки.

Удаление массива пользовательских данных не делает недействительными индексы массива, которые все еще используются. Индекс удаленного массива может быть возвращен последующим вызовом одной из функций add.

29.4 Резюме

29.4.1 Основные свойства

Плагин User Data используется для присоединения пользовательских данных к одному из трех объектов RenderWare Graphics:

- **RpGeometry**
- **RpWorldSector**
- **RwFrame**

Структура массива пользовательских данных

Пользовательские данные хранятся в**RpUserDataArray**структура, включающая в себя следующие элементы:

- А*имя*
  – доступ получен**RpUserDataArrayGetName()**
- А*формат данных*
  – доступ получен**RpUserDataArrayGetFormat()**
- Ан*количество элементов*
  – доступ получен**RpUserDataArrayGetNumElements()**
- Один или несколько*записи массива*
  – доступ осуществляется с помощью одной из функций доступа, перечисленных в разделе 1.4.2

Типы данных

Массив*формат данных*может быть одного из следующих трех типов:

- Целочисленные значения (тип**rpINTUSERDATA**)
- Реальные значения (тип**rpREALUSERDATA**)
- Струны (тип**rpSTRINGUSERDATA**)

29.4.2 Функции доступа

Массив может быть определен для хранения трех типов данных: целые числа, значения с плавающей точкой (действительные числа) или строки. Функции доступа предоставляются для каждого типа, как показано в следующей таблице:

ТИП МАССИВА	ФУНКЦИИ ДОСТУПА
rpINTUSERDATA	**RpUserDataArrayGetInt()** **RpUserDataArraySetInt()**
rpREALUSERDATA	**RpUserDataArrayGetReal()** **RpUserDataArraySetReal()**
rpSTRINGUSERDATA	**RpUserDataArrayGetString()** **RpUserDataArraySetString()**

29.4.3 Создание

Использование экспортеров

Два пакета моделирования, 3ds max и Maya, поддерживают экспорт массивов пользовательских данных. Методы достижения этого различаются в двух пакетах; поэтому см. Руководство художника для каждого пакета моделирования для получения подробной информации, специфичной для каждого моделлера.

Процедурное создание

Создание массива требует трех шагов:

1. Создайте пространство для одного или нескольких массивов внутри объекта, используя одну из следующих функций:

    - **RpGeometryAddUserDataArray()**
    - **RpWorldSectorAddUserDataArray()**
    - **RwFrameAddUserDataArray()**

2. Присвойте имена массивам и задайте их форматы данных.

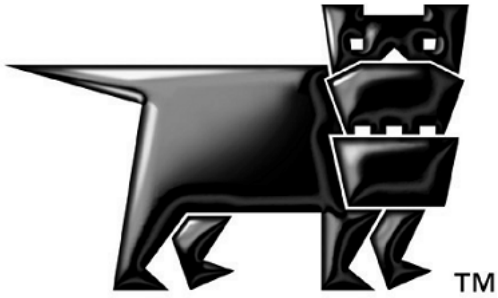
3. Заполните массивы данными, используя функции доступа, перечисленные в 1.4.2.

Часть G

PowerPipe

Глава 30

PowerPipe
Обзор



30.1 Введение

30.1.1 Что такое PowerPipe?

PowerPipe — это архитектура для обработки данных. Это очень общая архитектура, способная обрабатывать практически любые формы данных (например, HTML, сетевые пакеты или данные с джойстика с обратной связью по усилию — все это может быть обработано PowerPipe), но в контексте RenderWare Graphics она используется для обработки трехмерных геометрических данных, т. е. для их визуализации.

PowerPipe позволяет специфицировать «конвейеры», которые сконструированы для обработки определенного типа входных данных и создания определенного эффекта рендеринга выходных данных. Конвейеры инкапсулируют эту функциональность рендеринга удобным образом, так что ее можно обрабатывать аналогично текстурам или материалам для целей разработки контента приложения.

30.1.2 Трубопроводы и узлы

Конвейеры PowerPipe состоят из серии «узлов», которые обрабатывают данные в пакетах. Пакеты берутся из входных данных и передаются от узла к узлу в конвейере. Каждый узел содержит методы для обработки подмножества данных в пакете перед передачей их по конвейеру. Конвейер может разветвляться и рекомбинироваться таким образом, что может активироваться различное поведение в зависимости от деталей входных данных.

Узлы инкапсулируют небольшие компоненты функциональности рендеринга. Например, один узел может преобразовывать 3D-точки в мировом пространстве в пространство камеры, а другой может обрезать треугольники до пирамиды видимости камеры. Такая инкапсуляция функциональности рендеринга в узлах внутри конвейеров имеет много существенных преимуществ:

- обеспечивает удобное и простое создание пользовательских эффектов рендеринга с использованием готовых узлов компонентов;
- позволяет эффективно повторно использовать код рендеринга;
- облегчает взаимодействие между кодом (узлами), написанным разными авторами для разных целей;
- базирование процессов рендеринга на конвейерах, узлах и пакетах позволяет им хорошо масштабироваться до высокопараллельных многопроцессорных систем;
- позволяет создавать разнообразные и сложные эффекты рендеринга с минимальными затратами на разработку;
- Независимые от платформы конвейеры рендеринга обеспечивают мгновенную переносимость.

30.1.3 Использование PowerPipe в реальном мире

PowerPipe позволяет быстро разрабатывать и экспериментировать с пользовательской функциональностью рендеринга. Однако в реальной разработке (и особенно в разработке игр) производительность рендеринга имеет большое значение.

Поскольку PowerPipe является такой общей архитектурой, узлы конвейера могут инкапсулировать функциональность рендеринга настолько мелкозернистым или крупнозернистым способом, насколько это необходимо. Чтобы быстро создать прототип эффекта рендеринга, разработчик может объединить несколько существующих узлов, опционально добавив новый собственный узел. Пользовательские конвейеры также являются удобным способом выполнения «визуальной отладки» во время разработки (например, пользовательский «отладочный конвейер» может отображать нормали вершин в модели, выделяя ошибочные нормали, изменяя их цвет с течением времени). Однако, когда окончательный набор эффектов рендеринга для приложения выбран, протестирован и настроен, разработчик может затем захотеть оптимизировать критические для производительности конвейеры, объединив все узлы в каждом конвейере в один узел.

В случае платформ, которые в настоящее время поддерживаются RenderWare Graphics, большая часть обработки, связанной с рендерингом, теперь выполняется аппаратной графической подсистемой (например, векторным процессором VU1 в консоли PS2), часто называемой «аппаратным преобразованием и освещением» (или «HW T&L»). Это означает, что в конечном высокопроизводительном коде конвейеры PowerPipe выполняют небольшую обработку (обычно не более чем настройку состояния рендеринга) перед передачей геометрических данных в эту подсистему рендеринга. В этом документе описания использования PowerPipe будут в основном относиться к разработке платформенно-независимых конвейеров, чтобы охватить больше доступных функциональных возможностей PowerPipe. Общие (платформенно-независимые) конвейеры и узлы, предоставляемые RenderWare Graphics, сродни эталонному растеризатору Direct3D, обеспечивая базовую поддержку для всего оборудования. В последующих главах будет рассмотрено создание и использование оптимизированных платформенно-специфичных конвейеров рендеринга, которые могут достигать гораздо более высокой производительности на своей целевой платформе.

30.1.4 Другие документы

Вот еще несколько документов, имеющих отношение к PowerPipe, к которым вы, возможно, захотите обратиться:

- Следующая глава в этом руководстве пользователя, озаглавленная *Узлы трубопровода*, является продолжением этой главы и содержит сведения об узлах PowerPipe.
- Справочник по API PowerPipe и разделы, посвященные конкретным платформам.
- В этом руководстве пользователя есть глава, посвященная PowerPipe для PS2, под названием *PS2AllОбзор*.

30.2 Трубопроводы

В этом разделе будут рассмотрены следующие темы, связанные с трубопроводами PowerPipe:

- использование трубопроводов;
- возможности построения трубопроводной структуры;
- поток данных в конвейерах;
- строительство трубопроводов.

Сейчас мы их рассмотрим...

30.2.1 Использование трубопровода

Исполнение трубопровода

Конвейер PowerPipe может быть выполнен с помощью функции **RxPipelineExecute()**, где данные для обработки (обычно это объект RenderWare Graphics, такой как**RpАтомный**) передается как один из параметров. Однако,**RxPipelineExecute()**обычно вызывается из другой функции API, например**RpAtomicRender()**.
Соглашение в RenderWare Graphics заключается в том, что конвейеры PowerPipe прикрепляются к объектам, которые они могут визуализировать. Такие хуки для конвейеров предусмотрены для**RwIm3D**,**RpАтомный**с,**RpWorldSector**си**RpМатериал**с (а также для различных других объектов в плагинах). Стандартные конвейеры, предоставляемые в каждом из этих случаев, описаны в разделе*Общие трубопроводы* ниже.

Материальные трубопроводы

Характер трубопроводов, присоединенных к**RpМатериал**s нуждается в дальнейшем объяснении. Как вы знаете, оба**RpАтомный**си**RpWorldSector**сможет иметь много **RpМатериал**с привязаны к их геометрии (**RpМатериал**ассоциируется с каждым треугольником в объекте при его построении, во время выполнения или в пакете моделирования). Поскольку группировка треугольников по материалу имеет важное значение для получения приемлемой производительности рендеринга, геометрия подразделяется на**RpMesh**с,по одному на каждого **RpМатериал**который используется. Каждый**RpМатериал**имеет связанный конвейер PowerPipe, поэтому он определяет, как**RpMesh**сиспользуя это **RpМатериал**должны быть обработаны. Этот трубопровод называется просто «материальным трубопроводом».

Объектные трубопроводы

RpАтомныйси**RpWorldSector**стакже имеют связанные конвейеры PowerPipe, которые называются «конвейерами объектов». Цель состоит в том, чтобы конвейеры объектов занимались всей обработкой на уровне объектов (например, настройкой матрицы преобразования объекта, определением того, какие источники света на него влияют или извлечением соответствующих данных плагина для каждого объекта), а затем передавали один пакет геометрических данных на **RpMesh**к соответствующему материальному трубопроводу.

Конвейеры против обратных вызовов рендеринга

RpАтомныйси**RpWorldSector**стакже содержат обратные вызовы рендеринга, которые являются функциями, вызываемыми всякий раз, когда объект должен быть отрендерен. Эта функция в свою очередь (в случае обратных вызовов по умолчанию, в любом случае) выполняет конвейер объекта. В зависимости от потребностей разработчика может быть удобнее выполнять некоторые задачи, связанные с рендерингом объектов, перегружая этот обратный вызов, а не создавая пользовательский конвейер PowerPipe.

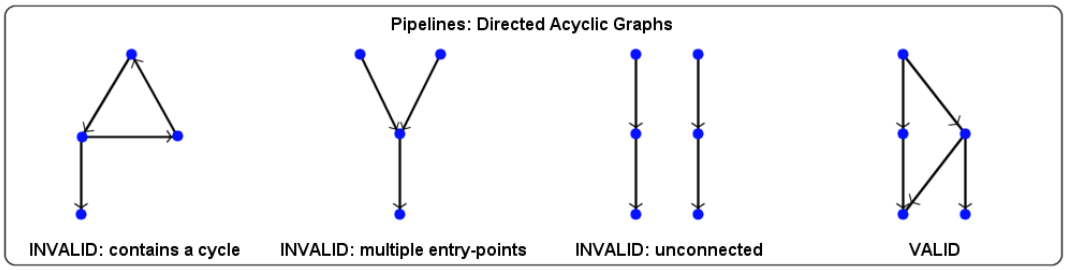
Далее следует список функций API, используемых для извлечения и указания конвейеров и рендеринга обратных вызовов для**RpАтомный**с**, RpWorldSector**си**RpМатериал**с.
Более подробную информацию см. в справочнике API:

- **RpAtomicGetPipeline()**
- **RpAtomicSetPipeline()**
- **RpAtomicGetRenderCallBack()**
- **RpAtomicSetRenderCallBack()**
- **RpWorldSectorGetPipeline()**
- **RpWorldSectorSetPipeline()**
- **RpWorldGetSectorRenderCallBack()**
- **RpWorldSetSectorRenderCallBack()**

30.2.2 Структура трубопровода

Трубопроводы PowerPipe описываются**RxPipeline**структура. Разработчику никогда не нужно получать доступ к содержимому этой структуры напрямую, поскольку все элементы настраиваются функциями API, используемыми при построении конвейеров.

Структура узлов в конвейере ограничена "направленным ациклическим графом". Это означает, что связи между узлами имеют четкое направление (узел A передает пакеты узлу B, но не наоборот) и что при следовании по этим связям не может быть образовано никаких петель. Кроме того, конвейеры должны иметь один узел точки входа (это место, где начинается выполнение или куда передаются пакеты, если они поступают из другого конвейера).



Учитывая эти ограничения, могут быть сформированы сложные структуры конвейеров (это полезно, например, когда необходимая функциональность рендеринга является контекстно-специфичной для конкретного типа объекта и эффекта рендеринга), содержащие ветви, которые могут либо заканчиваться в «тупике», либо сходиться с другими ветвями. Для облегчения этого каждый узел имеет один «вход» и один или несколько «выходов», через которые могут проходить пакеты.

После обработки пакета узел конвейера может отправить его вниз по любой из ветвей, выходящих из узла, каждая такая ветвь соответствует «выходу» узла. Большинство выходов ведут к другим узлам в конвейере, но также может быть определен выход, который передает пакеты в другой конвейер. Это особенно полезно в случае объектного конвейера, который должен передавать пакеты в соответствующий материальный конвейер для каждого**RpMesh**в объекте. Трубопровод, прикрепленный к**RpМатериал**может изменяться во время выполнения, поэтому связи между объектными и материальными конвейерами не могут быть созданы во время построения конвейера. В принципе, это осуществимо, но может потребовать создания очень большого количества конвейеров (произведение количества объектных конвейеров и количества материальных конвейеров), каждый со множеством ответвлений. Хотя эти конвейеры не были бы неэффективными, это вряд ли удобно, поэтому в большинстве случаев объектные и материальные конвейеры остаются отдельными.

Однако один из недостатков передачи пакетов между конвейерами заключается в том, что она может быть довольно медленной (в зависимости от текущей платформы), поскольку предварительная обработка потока данных*в пределах*конвейер (как описано в следующем разделе), который выполняется при построении конвейера, не был выполнен для передачи данных*между*трубопроводы.

30.2.3 Поток данных в конвейерах

Каждый узел в конвейере PowerPipe ожидает найти определенные типы данных в пакетах, которые он обрабатывает. Например, узел, который преобразует вершины из пространства объектов в пространство камеры, будет ожидать найти вершины пространства объектов в получаемых им пакетах. Таким образом, данные в пакетах разбиваются на «кластеры» — кластер является просто массивом определенного типа данных. В этом случае пакет будет включать кластер, содержащий список вершин пространства объектов.

-

Данные, содержащиеся в пакетах и кластерах, распределяются с использованием*PowerPipe*куча – это описано в следующей главе,*Узлы трубопровода*, в разделе*Куча трубопровода*.

TheRxClusterDefinitionструктура

The**RxClusterDefinition**структура определяет кластер. Эта структура показана здесь:

```
структура RxClusterDefinition {

  RwChar                  * имя;
  RwUInt32                  defaultStride;
  RwUInt32                  Атрибуты по умолчанию;
  константный символ      * Набор атрибутов;
};
```

Чтобы создать кластер внутри пакетов, проходящих по конвейеру, необходимо создать**RxClusterDefinition**структура и ссылаться на эту структуру в определении одного или нескольких узлов в конвейере. Определение узлов подробно описано в главе*Узлы трубопровода*. **имя**кластера действует просто как идентификационная метка.**defaultStride** элемент — это шаг связанного с ним типа данных (больший шаг может использоваться для обеспечения выравнивания элементов данных, а меньший шаг может использоваться для усечения ненужных конечных членов элементов данных). **defaultAttributes** являются флагами, определяющими платформенно-зависимые свойства кластера. **атрибутSet**member — это строка, определяющая набор атрибутов, к которым принадлежат атрибуты кластера (например, для кластеров, специфичных для PS2, набор атрибутов — «PS2»), чтобы их можно было правильно интерпретировать.

Зависимости кластера

В пакете может быть много кластеров, каждый из которых ссылается на **RxClusterDefinition**. Каждый узел конвейера будет иметь доступ только к тем кластерам, типы данных которых он знает, как интерпретировать. В конвейере один узел может инициализировать данные в кластере (из данных, содержащихся в исходном объекте), а другой может уничтожить кластер. Между тем другие узлы могут изменять данные в кластере или изменять длину массива кластера.

Когда узел определен (опять же, это будет подробно рассмотрено в главе*Узлы трубопровода*), кластеры, к которым ему необходимо получить доступ, указаны. Узел может указать одну или несколько из следующих вещей о своем доступе к данному кластеру:

- Узел требует, чтобы данные кластера были инициализированы до достижения этого узла;
- Узел хочет сам инициализировать данные кластера;
- Узел хочет завершить работу кластера;
- Узел будет использовать кластер, если он присутствует, но будет продолжать корректно функционировать и при его отсутствии.

Эта информация используется во время построения конвейера для оптимизации потока данных во время выполнения в конвейере и для проверки того, что все требования узлов в конвейере могут быть удовлетворены. Этот процесс известен как "преследование зависимости" и происходит во время функции **RxPipelineUnlock()**. Например, если узлу требуются нормали объектного пространства, то отслеживание зависимостей проверяет для всех пакетов, входящих в узел, что кластер нормалей объектного пространства был инициализирован предыдущим узлом в конвейере. В**RWDEBUG**Будут выдаваться сообщения о сборке, ошибках и предупреждениях, чтобы помочь отладить проблемы строительства трубопровода на этом этапе. Строительство трубопровода описано в следующем разделе.

Модель конвейерного исполнения

Следует отметить, что для повышения эффективности выполнения конвейера была принята следующая модель отправки пакетов: когда узел отправляет пакет следующему узлу в текущем конвейере (или головному узлу другого конвейера), выполнение программы фактически переходит к методу body этого узла. Это означает, что выполнение тела узла*вложенный*, что подразумевает, что в любой момент времени фактически существует только один пакет.

Чтобы пояснить это последствие, после того как пакет будет полностью обработан (его треугольники и вершины будут отправлены в API растеризации), текущее тело узла выйдет, как и тело узла, которое его вызвало (передало ему пакет) и т. д., вплоть до узла, который создал пакет в первую очередь (обычно ImmInstance.csl, AtomicInstance.csl или WorldSectorInstance.csl — эти узлы будут представлены позже, в разделе *Общие трубопроводы*). В этот момент может быть создан другой пакет и отправлен по конвейеру, хотя в каждый момент времени во время выполнения конвейера будет существовать только один пакет. Именно по этой причине был создан узел Clone.csl. Его цель — клонировать входящие пакеты и отправлять клоны на несколько выходов (этот узел более подробно описан в следующей главе,*Узлы трубопровода*).

Учитывая эту модель выполнения, авторам узлов необходимо тщательно продумать состояние. Изменения в состоянии (например, состояние рендеринга), вызванные последующими узлами в конвейере, вступят в силу после отправки пакета, и пакет, скорее всего, больше не будет содержать действительных данных.

30.2.4 Строительство трубопровода

Строительство трубопровода PowerPipe состоит из следующих этапов:

1. Создать конвейер;

2. Заблокировать для редактирования;

3. Укажите узлы и топологию трубопровода, добавив фрагменты и соединительные пути между ними;

4. Выполнить предварительную настройку узлов через их API;

5. Разблокировать конвейер (выполняется отслеживание зависимостей);

6. Выполните настройку узлов после разблокировки через их API.

RxPipelineCreate()

Трубопроводы создаются с помощью функции**RxPipelineCreate()**. После создания конвейеры пусты (они не содержат узлов) и находятся в «разблокированном» виде (в том же смысле, что и разблокированные**RpGeometry**с).

RxPipelineLock()

Для редактирования трубопровод должен быть заблокирован с помощью функции **RxPipelineLock()**. Редактирование используется для указания узлов и топологии конвейера, а также для инициализации каждого узла с использованием любых функций API, которые он может иметь.

RxLockedPipeAddFragment()иRxLockedPipeAddPath()

Ключевые функции, используемые при добавлении узлов в конвейер и создании желаемой структуры:**RxLockedPipeAddFragment()**и **RxLockedPipeAddPath()**. **RxLockedPipeAddFragment()**используется для добавления линейной цепочки узлов, называемой «фрагментом», к конвейеру. В этой цепочке каждый узел соединен со следующим своим*первый*выход. Этот фрагмент изначально не связан ни с какими другими фрагментами, добавленными в конвейер.

Максимальное количество узлов, которое может содержать конвейер, по умолчанию задается значением**RXPIPELINEDEFAULTMAXNODES**. Это значение можно переопределить, изменив значение_**rxPipelineMaxNodes***до*Инициализируется RenderWare Graphics.

RxLockedPipeAddPath()используется для присоединения выхода одного узла к входу другого узла, таким образом потенциально связывая отдельные фрагменты и создавая или рекомбинируя ветви.

Для подготовки параметров для этих функций используйте следующие вспомогательные функции:**RxPipelineFindNodeByName()**, **RxPipelineNodeFindInput()**и**RxPipelineNodeFindOutputByName()**. Они определяют местоположение узлов, входов и выходов узлов в конвейере до или после разблокировки конвейера (эти объекты перемещаются, когда конвейер разблокирован).

RxPipelineUnlock()

После завершения редактирования конвейера его следует разблокировать с помощью функции**RxPipelineUnlock()**. Эта функция выполняет «погоня за зависимостью» представлена в предыдущем разделе*Поток данных в трубопроводах*. **RxPipelineUnlock()** автоматически определяет точку входа в конвейер (если это невозможно, то структура конвейера недействительна), но это можно переопределить, вызвав**RxLockedPipeSetEntryPoint()**до **RxPipelineUnlock()**.

После разблокировки конвейера можно выполнить дополнительную настройку отдельных узлов внутри конвейера, используя любые функции API, которые могут иметь эти узлы.

Эта предварительная и последующая настройка узлов объясняется в главе *Узлы трубопровода*(Короче говоря, цель состоит в том, чтобы настроить частные данные, которые принадлежат узлам и используются во время выполнения узла).

Пример кода

Вот пример функции создания конвейера, которая связывает фиктивные узлы в конвейер с двумя ветвями, которые расходятся от первого узла, а затем воссоединяются в конечном узле:

```
RxPipeline*
CreateMyPipeline(void)
{
    RxPipeline *newPipe;

    /* Создать пустой конвейер */
    newPipe = RxPipelineCreate(); if (NULL !
    = newPipe)
    {
        RxLockedPipe *lockedPipe;

        /* Блокируем новый конвейер для
        редактирования */ lockedPipe =
        RxPipelineLock(newPipe); if (NULL != lockedPipe)
        {
            RxNodeDefinition    * inspectNode, *shineNode;
            RxNodeDefinition    * glowNode, *completeNode;
            RxNodeInput вход;
```

```
RxNodeOutput вывод;
RxPipeline *результат;

/* Извлечь указатели на определения
 * узлов, которые вы хотите использовать */
 inspectNode    =  RxNodeDefinitionGetInspect();
утверждать(NULL  !=  inspectNode);
shineNode =     RxNodeDefinitionGetShine(); !=
утверждать(NULL  shineNode);
glowNode =    RxNodeDefinitionGetGlow();
утверждать(NULL  !=  glowNode);
полныйУзел      =  RxNodeDefinitionGetComplete();
утверждать(NULL  !=  completeNode);

/* Добавляем линейную цепочку из трех узлов в конвейер */
lockedPipe = RxLockedPipeAddFragment(lockedPipe,
                                     НУЛЕВОЙ,
                                     inspectNode,
                                     shineNode,
                                     полныйУзел,
                                     НУЛЕВОЙ);
assert(NULL != lockedPipe);

/* Добавляем еще один узел в конвейер */ lockedPipe =
RxLockedPipeAddFragment(lockedPipe,
                                     НУЛЕВОЙ,
                                     glowNode);
assert(NULL != lockedPipe);

/* Связать отдельный узел с исходным фрагментом */ plNode
= RxPipelineFindNodeByName(
                        lockedPipe, "Inspect.csl", NULL, NULL);
утверждать(NULL != plNode);
plNode2 = RxPipelineFindNodeByName(
                        lockedPipe, "Glow.csl", NULL, NULL); != plNode2);
утверждать(NULL
выход      =  RxPipelineNodeFindOutputByName(
                        plNode, "СветящийсяВыход");
вход       =  RxPipelineNodeFindInput(plNode2);
результат  =    RxLockedPipeAddPath(
                    заблокированный канал, выход, вход);
утвердить(NULL != результат);

plNode     =  plNode2;
plNode2    =  RxPipelineFindNodeByName(
                        lockedPipe, "Complete.csl", NULL, NULL);
утверждать(NULL != plNode2);
вывод = RxPipelineNodeFindOutputByName(
                        plNode, "StandardOut");
```

```
                    вход        = RxPipelineNodeFindInput(plNode2); =
                    результат   RxLockedPipeAddPath(
                                        заблокированный канал, выход, вход);
                    утвердить(NULL != результат);

                    /* Здесь выполняется предварительная разблокировка узла конвейера */

                    результат = RxLockedPipeUnlock(lockedPipe); если
                    (NULL != результат)
                    {
                        /* Здесь выполняется настройка узла конвейера после разблокировки */

                        возврат(результат);
                    }
                }

                RxPipelineDestroy(newPipe);
            }

            возврат (NULL);
}
```

Ниже приведен список функций API, используемых при строительстве трубопроводов, многие из которых упомянуты в главе.*Узлы трубопровода*. Более подробную информацию см. в документации по API:

Функции для управления трубопроводами:

- **RxPipelineCreate()**
- **RxPipelineDestroy()**
- **RxPipelineClone()**
- **RxPipelineLock()**

Функции для управления заблокированными трубопроводами:

- **RxLockedPipeUnlock()**
- **RxLockedPipeAddFragment()**
- **RxLockedPipeAddPath()**
- **RxLockedPipeDeletePath()**
- **RxLockedPipeDeleteNode()**
- **RxLockedPipeReplaceNode()**
- **RxLockedPipeGetEntryPoint()**

- **RxLockedPipeSetEntryPoint()**

Функции для управления узлами конвейера:

- **RxPipelineFindNodeByName()**
- **RxPipelineFindNodeByIndex()**
- **RxPipelineNodeFindInput()**
- **RxPipelineNodeFindOutputByName()**
- **RxPipelineNodeFindOutputByIndex()**
- **RxPipelineNodeCloneNodeDefinition()**
- **RxPipelineNodeRequestCluster()**
- **RxPipelineNodeReplaceCluster()**
- **RxPipelineNodeGetInitData()**
- **RxPipelineNodeCreateInitData()**

30.3 Общие конвейеры

Как описано выше в разделе выше*Использование трубопровода*, RenderWare Graphics связывает конвейеры с**RwIm3D**,**RpАтомный**с,**RpWorldSector**песок**RpМатериал**s. Для каждого из них есть предоставленный "общий" конвейер - то есть тот, который визуализирует эти вещи "стандартным" образом и который будет работать на всех платформах (хотя, возможно, не оптимально ни на одной из них). Эти конвейеры предоставляются в**RtGenCPipe**набор инструментов.

Каждый из этих конвейеров описан в текущем разделе вместе со списком функций API, используемых для извлечения этих конвейеров и установки других значений по умолчанию вместо них. Конвейеры, специфичные для платформы, рассматриваются в следующем разделе. Это будут фактические значения по умолчанию на любой данной платформе, хотя общие конвейеры всегда будут доступны.

Также следует отметить, что хотя разделение объектных и материальных конвейеров является подходом, принятым по умолчанию, можно создавать объектные конвейеры, которые являются "все в одном", т.е. которые выполняют всю работу по рендерингу объекта, полностью игнорируя материальные конвейеры. Это часто более эффективно, хотя, конечно, менее гибко.

30.3.1 РвИм3Д

В настоящее время используются два типа трубопроводов:**RwIm3D**рендеринг:

1. **RwIm3DTransform()**использует конвейер для преобразования вершин из мирового пространства (или пространства объектов, теперь, когда есть необязательный**RwMatrix** параметр этой функции) в экранное пространство;
2. **RwIm3DRenderPrimitive()**и**RwIm3DRenderIndexedPrimitive()** оба отправляют треугольники, созданные из преобразованных вершин, в API растеризации.

В действительности, в современных системах, которые имеют возможности HW T&L, **RwIm3DTransform()**просто кэширует указатель на исходные вершины, которые затем доступен для функций рендеринга, которые выполняют сами преобразования вершин. Конвейер внутри**RwIm3DTransform()**обычно состоит из одного узла и выполняет очень мало обработки. Однако общие конвейеры работают "старомодным" способом (т. е. все на главном ЦП), и это то, что будет описано в этом разделе.

Поведение**RwIm3D**конвейеры преобразования и рендеринга в отношении состояния рендеринга просто позволяют всем состояниям рендеринга сохраняться. Поэтому перед вызовом **RwIm3DRenderPrimitive()**или**RwIm3DRenderIndexedPrimitive()**(или выполнение конвейера рендеринга напрямую), вам следует настроить все необходимые вам состояния рендеринга, такие как растр текстуры и режимы альфа-смешивания. Конвейер не будет изменять это состояние рендеринга во время его выполнения.

Универсальный конвейер преобразования RwIm3D

Вот структура дженерика**RwIm3D**конвейер преобразований:

ImmInstance.csl
↓
Трансформировать.csl
↓
ImmStash.csl

ImmInstance.csl

Целью узла ImmInstance.csl является инициализация пакета, содержащего несколько стандартных кластеров (они описаны в главе *Узлы трубопровода*), из данных, переданных из**RwIm3DTransform()**.

Трансформировать.csl

Узел Transform.csl преобразует вершины пространства объекта в пространство камеры и генерирует вершины как пространства камеры, так и пространства экрана, а также выполняет тесты усеченной пирамиды для каждой вершины, которые впоследствии будут использоваться при отсечении треугольников.

ImmStash.csl

Узел ImmStash.csl «сохраняет» содержимое входящих пакетов в глобальной структуре для использования в последующих выполняемых операциях.**RwIm3D**конвейеры рендеринга.

Расширение «.csl"в строках имен узлов используется для идентификации узлов как принадлежащих Criterion Software Ltd.

Универсальный конвейер рендеринга треугольников RwIm3D

Вот структура дженерика**RwIm3D**Конвейер рендеринга для примитивов на основе треугольников:

ImmRenderSetup.csl
↓
ImmMangleTriangleIndices.csl
↓
CullTriangle.csl
↓
ClipTriangle.csl
↓
ОтправитьTriangle.csl

ImmRenderSetup.csl

Целью узла ImmRenderSetup.csl является инициализация пакета из данных, «сохраненных» предыдущим**RwIm3D**конвейер преобразования и добавления индексов, переданных из вызывающего**RwIm3D**функция рендеринга.

ImmMangleTriangleIndices.csl

ImmMangleTriangleIndices.csl преобразует индексы из примитивов tri-strip и tri-fan в индексы для примитива tri-list. Это связано с тем, что большинство узлов-обработчиков треугольников могут обрабатывать только tri-list.

CullTriangle.csl

CullTriangle.csl удаляет невидимые треугольники из пакета, т.е. те, которые отсечены с обратной стороны или полностью находятся за пределами пирамиды обзора текущей камеры.

ClipTriangle.csl

ClipTriangle.csl обрезает треугольники по пирамиде видимости.

ОтправитьTriangle.csl

SubmitTriangle.csl настраивает состояние рендеринга и отправляет 2D-треугольники в API растеризации.

Универсальный конвейер рендеринга линий RwIm3D

Вот структура дженерика**RwIm3D**Конвейер рендеринга для линейных примитивов:

ImmRenderSetup.csl
↓
ImmMangleLineIndices.csl
↓
ClipLine.csl
↓
SubmitLine.csl

ImmMangleLineIndices.csl

ImmMangleLineIndices.csl по своей функции аналогичен ImmMangleTriangleIndices.csl, преобразуя индексы из примитивов полилиний в индексы для примитива списка линий.

ClipLine.csl

ClipLine.csl обрезает линии по пирамиде видимости.

SubmitLine.csl

SubmitLine.csl настраивает состояние рендеринга и отправляет линии в API растеризации.

Более подробную информацию о каждом из узлов, упомянутых в этом разделе, можно найти в справочнике по API для следующих функций:

- **RxNodeDefinitionGetImmInstance()**
- **RxNodeDefinitionGetTransform()**
- **RxNodeDefinitionGetImmStash()**
- **RxNodeDefinitionGetImmRenderSetup()**
- **RnodeDefinitionGetImmMangleTriangleIndices()**
- **RxNodeDefinitionGetImmMangleLineIndices()**
- **RxNodeDefinitionGetCullTriangle()**
- **RxNodeDefinitionGetClipTriangle()**
- **RxNodeDefinitionGetClipLine()**
- **RxNodeDefinitionGetSubmitTriangle()**
- **RxNodeDefinitionGetSubmitLine()**

Вот список функций API, используемых для извлечения общих конвейеров, используемых в**RwIm3D**a также извлечение и указание трубопроводов, которые в настоящее время находятся в

использовать:

- **RwIm3DGetGenericTransformPipeline()**
- **RwIm3DGetGenericRenderPipeline()**
- **RwIm3DGetTransformPipeline()**
- **RwIm3DGetRenderPipeline()**
- **RwIm3DSetTransformPipeline()**
- **RwIm3DSetRenderPipeline()**

Обратите внимание, что при извлечении или указании**RwIm3D**конвейер рендеринга, функции API требуют параметр, указывающий**RwPrimitiveType**к которому должен применяться конвейер. Нет конвейера по умолчанию для работы с **rwPRIMTYPEPOINTLIST**примитивный тип, поскольку не существует "стандартного" способа, которым такие примитивы должны визуализироваться. Мы предоставляем поддержку этого примитивного типа, поскольку это удобная структура для построения пользовательских конвейеров вокруг (особенно для таких объектов, как системы частиц).

30.3.2 RpАтомный

Общий**RpАтомный**и**RpWorldSector**Объектные конвейеры используют тот же общий материальный конвейер, поэтому он будет описан в следующем разделе.

Универсальный конвейер объектов RpAtomic

Вот структура дженерика**RpАтомный**Конвейер объектов:

AtomicInstance.csl
↓
AtomicEnumerateLights.csl
↓
МатериалScatter.csl

AtomicInstance.csl

Целью узла AtomicInstance.csl является создание экземпляра данных вершин и треугольников в**RwResEntry**и создать один пакет на**RpMesh**в объект, кластеры которого ссылаются на этот экземпляр данных. Если текущий **RpАтомный**является ли морф-анимированным, интерполяция ключевых кадров будет происходить во время инстанцирования (следовательно, переинстансирование должно происходить каждый раз, когда состояние анимации**RpАтомный**изменения). Треугольные индексы создаются как трилисты, даже если топология источника**RpАтомный**указан с tri-strips. Это связано с тем, что большинство узлов-обработчиков треугольников могут обрабатывать только tri-lists.

AtomicEnumerateLights.csl

AtomicEnumerateLights.csl создает кластер источников света, содержащий указатели на **RpLight**s для всех глобальных источников света и локальных источников света, области влияния которых перекрываются**RpWorldSector**s, который текущий**RpАтомный**пересекается.

МатериалScatter.csl

Узел MaterialScatter.csl отправляет текущий пакет в материальный конвейер, указанный в**RpМатериал**текущего**RpMesh**.

Более подробную информацию о каждом из узлов, упомянутых в этом разделе, можно найти в справочнике по API для следующих функций:

- **RxNodeDefinitionGetAtomicInstance()**
- **RxNodeDefinitionGetAtomicEnumerateLights()**
- **RxNodeDefinitionGetMaterialScatter()**

Вот список функций API, используемых для получения общего**RpАтомный** конвейер объектов, а также извлечение и указание конвейеров по умолчанию:

- **RpAtomicGetGenericPipeline()**
- **RpAtomicGetDefaultPipeline()**
- **RpAtomicSetDefaultPipeline()**

30.3.3 RpWorldSector

Как уже упоминалось выше, общий**RpWorldSector**и**RpАтомный**Объектные конвейеры используют тот же общий материальный конвейер, поэтому он будет описан в следующем разделе.

Универсальный конвейер объектов RpWorldSector

Вот структура дженерика**RpWorldSector**Конвейер объектов:

```
WorldSectorInstance.csl
  ↓
WorldSectorEnumerateLights.csl
  ↓
МатериалScatter.csl
```

WorldSectorInstance.csl

Целью узла WorldSectorInstance.csl является создание экземпляра данных вершин и треугольников в**RwResEntry**и создать один пакет на**RpMesh**в объект, кластеры которого ссылаются на этот экземпляр данных. Треугольные индексы создаются как трилисты, даже если топология источника **RpWorldSector** указан с tri-strips. Это связано с тем, что большинство узлов-обработчиков треугольников могут обрабатывать только tri-lists.

WorldSectorEnumerateLights.csl

WorldSectorEnumerateLights.csl создает кластер источников света, содержащий указатели на**RpLight**s для всех глобальных источников света и локальных источников света, области влияния которых перекрывают текущий**RpWorldSector**.

Более подробную информацию о каждом из узлов, упомянутых в этом разделе, можно найти в справочнике по API для следующих функций:

- **RxNodeDefinitionGetWorldSectorInstance()**
- **RxNodeDefinitionGetWorldSectorEnumerateLights()**
- **RxNodeDefinitionGetMaterialScatter()**

Вот список функций API, используемых для получения общего **RpWorldSector** конвейер объектов, а также извлечение и указание конвейера по умолчанию и конвейера по умолчанию для конкретного**RpWorld**:

- **RpWorldGetGenericSectorPipeline()**
- **RpWorldGetDefaultSectorPipeline()**
- **RpWorldSetDefaultSectorPipeline()**
- **RpWorldGetSectorPipeline()**
- **RpWorldSetSectorPipeline()**

30.3.4 RpМатериал

Как упоминалось выше, общий трубопровод материалов используется как **RpWorldSector**и**RpАтомный**общие объектные конвейеры.

Трубопровод общего назначения

Вот структура конвейера общих материалов:

Трансформировать.csl
↓
CullTriangle.csl
↓
Свет.csl
↓
PostLight.csl
↓
ClipTriangle.csl
↓
ОтправитьTriangle.csl

Свет.csl

Узел Light.csl добавляет световые вклады от каждого**RpLight**в кластере огней (созданном AtomicEnumerateLights.csl или WorldSectorEnumerateLights.csl (как описано выше) к цвету вершины каждой вершины в текущем пакете. Если в исходном объекте присутствуют цвета предварительной подсветки, цвета вершин будут инициализированы с учетом этого узлом преобразования. Если в исходном объекте отсутствуют цвета предварительной подсветки, то узел экземпляра инициализирует цвета вершин непрозрачным черным).

PostLight.csl

Узел PostLight.csl фиксирует значения цвета вершин в диапазоне [0-255] и преобразует**RwRGBAReal**значения, накопленные узлом Light в **RwRGBA** значения, которые будут использоваться в вершинах, представленных в API растеризации. Узел ClipTriangle.csl должен располагаться после узлов освещения, поскольку он должен интерполировать (для обрезанных треугольников)*финал*цвета вершин.

Более подробную информацию о каждом из узлов, упомянутых в этом разделе, можно найти в справочнике по API для следующих функций:

- **RxNodeDefinitionGetTransform()**
- **RxNodeDefinitionGetCullTriangle()**
- **RxNodeDefinitionGetPreLight()**
- **RxNodeDefinitionGetLight()**
- **RxNodeDefinitionGetPostLight()**
- **RxNodeDefinitionGetClipTriangle()**
- **RxNodeDefinitionGetSubmitTriangle()**

Ниже приведен список функций API, используемых для извлечения общих материальных конвейеров, а также для извлечения и указания конвейера по умолчанию:

- **RpMaterialGetGenericPipeline()**
- **RpMaterialGetDefaultPipeline()**
- **RpMaterialSetDefaultPipeline()**

30.4 Конкретные конвейеры платформы

Конвейеры и узлы конвейеров, специфичные для платформы, необходимы по двум основным причинам. Во-первых, разные платформы используют разное оборудование и аппаратную архитектуру. При создании общего кроссплатформенного API для получения одинаковых результатов требуется использование разного кода на разных платформах. Часто для максимально эффективного использования каждой платформы требуются совершенно разные подходы. Во-вторых, конвейеры и узлы конвейеров, специфичные для платформы, могут использоваться для раскрытия любых уникальных возможностей данной платформы.

Примером платформенно-зависимого конвейерного кода является архитектура PS2All, используемая для построения конвейеров для PS2. В случае PS2 большая часть обработки рендеринга выполняется на векторном процессоре VU1. Это означает, что необходимо создать данные экземпляра в соответствующей форме для передачи на этот процессор механизмом DMA и для управления микрокодом, выполняемым на VU1. Поскольку затраты на выполнение кода CPU могут быть очень высокими на PS2 (из-за его небольших кэшей CPU), PS2All был разработан для выполнения как можно меньше работы на основном CPU при настройке VU1 и данных, которые он будет обрабатывать. PS2All также был разработан с учетом высокой настраиваемости, чтобы разработчики могли еще больше снизить нагрузку на CPU при рендеринге, используя преимущества знаний, специфичных для приложения.

В этом руководстве пользователя есть глава под названием *PS2All Обзор*, охватывающий платформенно-специфичные конвейеры, используемые на PS2. Главы, описывающие детали конвейеров, созданных для использования на других платформах, будут добавлены в свое время. На данный момент справочная документация API содержит много платформенно-специфичной документации. Некоторые отправные точки:

- Модули→PowerPipe→Расширения мира→PS2 Все
- Модули→Платформо-специфический

30.5 Распространенные ловушки и ошибки

Распространенные проблемы, возникающие при строительстве трубопроводов PowerPipe, будут рассмотрены в следующей главе.*Узлы трубопровода*.

При создании пользовательских конвейеров настоятельно рекомендуется прочитать справочную документацию API, касающуюся соответствующих функций API и узлов конвейера. Хотя эта глава дает полезный обзор PowerPipe сверху вниз, она является скорее дополнением, чем заменой справочной документации API, которая является подробной и всеобъемлющей.

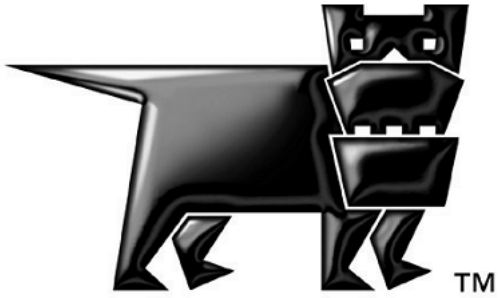
30.6 Резюме

В этой главе представлен обзор архитектуры PowerPipe. В ней представлены цели и преимущества PowerPipe, а также основные концепции конвейеров, узлов и пакетов. В ней рассмотрено использование конвейеров в RenderWare Graphics. В ней рассмотрены возможные структуры конвейеров, элементарные аспекты потока данных в конвейерах и рассмотрено построение конвейеров. В ней описаны общие конвейеры, поставляемые с RenderWare Graphics, и рассмотрены конвейеры, специфичные для платформы.

В следующей главе под названием*Узлы трубопровода*, будут рассмотрены детали создания пользовательских узлов конвейера. Более подробно будут рассмотрены общие узлы, поставляемые с RenderWare Graphics (включая некоторые, не упомянутые в этой главе).

Глава 31

Узлы трубопровода



31.1 Введение

Прежде чем читать эту главу, вам следует сначала прочитать главу *Обзор PowerPipe*, поскольку эта глава относится к представленным в ней концепциям.

В этой главе будут рассмотрены шаги, необходимые для создания узла конвейера PowerPipe: создание "определения узла" и методов узла. Также будут представлены общие (независимые от платформы) узлы, поставляемые с RenderWare Graphics, и кластеры, которые они используют.

31.1.1 Определение узла

Определение узла определяет, как узел связывается с другими узлами в конвейере. Когда узел добавляется в конвейер, это делается через ссылку на его определение узла, которое имеет тип **RxNodeDefinition**«Экземпляр» узла в пределах конкретного конвейера называется узлом конвейера и имеет тип **RxPipelineNode**.

Как упоминалось в предыдущей главе, *Обзор PowerPipe*, данные, обрабатываемые определенным узлом (т. е. "кластеры", к которым он обращается), будут связаны с другими узлами в конвейере, когда конвейер "разблокируется" во время его построения. Процесс "преследования зависимостей", который это влечет за собой, будет более подробно рассмотрен в этой главе.

31.1.2 Методы узлов

Методы узла обеспечивают функциональность во время построения и выполнения конвейера. Все, кроме одного, методы узла являются методами инициализации, которые используются во время построения конвейера для настройки области личных данных узла. Эта область имеет произвольный размер и используется для хранения данных, которые будут использоваться во время выполнения метода тела узла. Это общий способ параметризации **RxPipelineNode**, чтобы сделать его поведение специфичным для конкретного конвейера, в котором он находится. Узел может иметь дополнительный набор функций API, которые могут использоваться во время построения конвейера (или после этого в некоторых случаях) для изменения личных данных узла и, таким образом, изменения его поведения во время выполнения.

Последний метод узла — это его метод «тела» (типа **RxNodeBodyFn**), и представляет собой метод, который определяет поведение узла во время выполнения (т. е. то, как он обрабатывает данные и как он влияет на поток данных по конвейеру).

Следует отметить, что, как и в предыдущей главе, *Обзор PowerPipe*, описания использования PowerPipe в этой главе будут в основном относиться к разработке конвейера, независимого от платформы, чтобы охватить больше доступных функций PowerPipe. В последующих главах будет рассмотрено создание и использование оптимизированных узлов, специфичных для платформы, и конвейеров рендеринга, которые могут достичь гораздо более высокой производительности на целевой платформе.

31.1.3 Другие документы

Вот еще несколько документов, имеющих отношение к PowerPipe, к которым вы, возможно, захотите обратиться:

- Предыдущая глава в этом руководстве пользователя, озаглавленная *Обзор PowerPipe*, которую следует прочитать перед этой главой, в ней подробно описываются трубопроводы PowerPipe и их конструкция.
- Справочник по API PowerPipe и разделы, посвященные конкретным платформам.
- В этом руководстве пользователя есть глава, посвященная PowerPipe для PS2, под названием *PS2All Обзор*.

31.2 Определение узла

В этом разделе содержатся следующие пункты:

- Пример кода, демонстрирующий**RxNodeDefinition**создание, для справки через следующие два подраздела;
- Структуры типов, используемых в примере кода;
- Описание спецификации входных требований и выходов узла;
- Описание методов, связанных с узлом.

Сейчас мы их рассмотрим...

31.2.1 Пример кода

Далее следует пример кода, демонстрирующий создание определения узла, к которому можно обратиться, читая следующие два подраздела. В качестве примера используется определение узла для узла UVInterp.csl:

```
RxNodeDefinition *
RxNodeDefinitionGetUVInterp(void) {

    статический RxClusterRef clOfInterest[] =
      { {&RxClScrSpace2DVertices, rxCLALLOWABSENT, rxCLRESERVED},
        {&RxClRenderState,        rxCLALLOWABSENT,  rxCLRESERVED},
        {&RxClИнтерполянты,       rxCLALLOWABSENT,  rxCLRESERVED},
        {&RxClUVs,                rxCLALLOWABSENT,  rxCLRESERVED}   };

# определять NUMCLUSTERSOFINTEREST     \
        (sizeof(clOfInterest)        /  размер(clOfInterest[0]))

    статический RxClusterValidityReq inputReqs[NUMCLUSTERSOFINTEREST] =
      {  rxCLREQ_REQUIRED,
        rxCLREQ_REQUIRED,
        rxCLREQ_REQUIRED,
        rxCLREQ_OPTIONAL };

    статический RxClusterValid вывод1Clusters[NUMCLUSTERSOFINTEREST] =
      {  rxCLVALID_VALID,
        rxCLVALID_VALID,
        rxCLVALID_VALID,
        rxCLVALID_VALID };

    статический RxClusterValid вывод2Clusters[NUMCLUSTERSOFINTEREST] =
      {  rxCLVALID_VALID,
        rxCLVALID_VALID,
```

```
        rxCLVALID_NOCHANGE,
        rxCLVALID_NOCHANGE };

    статический RwChar UVsOut[]          = RWSTRING("UVsOut");
    static RwChar PassThrough[] = RWSTRING("PassThrough");

    статические выходы RxOutputSpec[] =
      {  {UVsOut,          output1Кластеры, rxCLVALID_NOCHANGE},
        {PassThrough, output2Clusters, rxCLVALID_NOCHANGE} };

# определять  NUMOUTPUTS          \
        (размер(выходы)    /  размер(выходы))

    статический RwChar UVInterp_csl[] = "UVInterp.csl";

    статический RxNodeDefinition nodeUVInterpCSL =
      {  RwChar                        * UVInterp_csl,
        {  (RxNodeBodyFn)              UVInterpNode,
          (RxNodeInitFn)               НУЛЕВОЙ,
          (RxNodeTermFn)               НУЛЕВОЙ,
          (RxPipelineNodeInitFn)       _UVInterpNodePipelineNodeInitFn, NULL,
          (RxPipelineNodeTermFn)
          (RxPipelineNodeConfigFn)NULL,
          (RxConfigMsgHandlerFn)       НУЛЕВОЙ },
        {  (RwUInt32)                  NUMCLUSTERSOFINTEREST,
          (RxClusterRef)               clOfInterest,
          (RxClusterValidityReq)       входные требования,
          (RwUInt32)                   КОЛИЧЕСТВО ВЫХОДОВ,
          (RxOutputSpec)               выходы},
        (RwUInt32)                     размер(RxNodeUVInterpSettings), ЛОЖЬ,
        (RxNodeDefEditable)
        (RwInt32)                      0 };

    RxNodeDefinition *результат = &nodeUVInterpCSL;

    RWAPIFUNCTION(RWSTRING("RxNodeDefinitionGetUVInterp"));

    RWRETURN(результат);
};
```

Обратите внимание, что использование**#определять**Продемонстрированный в этом примере код является обычной практикой при создании определений узлов для узлов, предоставляемых RenderWare Graphics. Цель состоит лишь в том, чтобы уменьшить вероятность внесения ошибок в определения узлов при их редактировании.

31.2.2 Структуры

RxNodeDefinition

Вот структура**RxNodeDefinition**тип, как указано в приведенном выше примере кода:

```
структура RxNodeDefinition {

    RwChar                  * имя;
    Методы RxNode             nodeMethods;
    RxIoSpec                  ио;
    RwUInt32                  pipelineNodePrivateDataSize;
    RxNodeDefEditable         редактируемый;
    RwInt32                   ВходныеПайпсыCnt;
};
```

Методы RxNode

Вот структура**Методы RxNode**, подтип**RxNodeDefinition**:

```
структура RxNodeMethods
{
    RxNodeBodyFn              узелBody;
    RxNodeInitFn              nodeInit;
    RxNodeTermFn              nodeTerm;
    RxPipelineNodeInitFn      pipelineNodeInit;
    RxPipelineNodeTermFn      pipelineNodeTerm;
    RxPipelineNodeConfigFn    pipelineNodeConfig;
    RxConfigMsgHandlerFn      configMsgHandler;
};
```

RxIoSpec

Вот структура**RxIoSpec**, подтип**RxNodeDefinition**:

```
структура RxIoSpec
{
    RwUInt32                  числоКластеровИнтереса;
    RxClusterRef            * кластерыИнтереса;
    RxClusterValidityReq    * входные требования;
    RwUInt32                  numOutputs;
    RxOutputSpec            * выходы;
};
```

RxClusterRef

Вот структура**RxClusterRef**, подтип**RxIoSpec**:

```
структура RxClusterRef
{
    RxClusterDefinition          * clusterDef;
    RxClusterForceПрисутствует   силаПрисутствует;
    RwUInt32                     сдержанный;
};
```

RxOutputSpec

Вот структура**RxOutputSpec**, подтип**RxIoSpec**:

```
структура RxOutputSpec
{
    RwChar                  * имя;
    RxClusterValid          * выходные кластеры;
    RxClusterValid            всеДругиеКластеры;
};
```

Представители всех этих типов будут описаны в следующих двух подразделах.

31.2.3 Требования к входным данным и выходным данным

Для того, чтобы его можно было правильно вставить в поток данных в конвейере, узел должен точно указать свои входные требования и выходы. Входные требования узла — это требования, которые узел предъявляет к данным в любых пакетах, которые входят в узел. Каждый из множества выходов, которые может иметь узел, может быть указан в терминах состояния данных в пакетах, которые покидают узел через этот выход.

Кластеры интересов

Чтобы указать входные требования и выходы узла, вы должны сначала указать, какие кластеры интересуют узел. Это кластеры, которые узел (в любом из своих состояний поведения) может выбрать для создания, записи, чтения или уничтожения. Эти интересующие кластеры указаны в массиве **кластерыИнтерес**(тип**RxClusterRef**) в**ио**член (тип **RxIoSpec**) принадлежащий **RxNodeDefinition**. Количество интересующих кластеров указано в **числоКластеровИнтереста**член**ио**. Максимально допустимое количество кластеров интересов определяется **RXNODEMAXCLUSTERSOFINTEREST**.

Для каждого кластера интересов должен быть один**RxClusterRef**запись в этом массиве. Каждая запись содержит три вещи:

1. Указатель на определение интересующего кластера (тип **RxClusterDefinition**) идентифицировать его;

2. Перечисленное значение**силаНастоящая**(тип**RxClusterForceПрисутствует**), который в большинстве случаев может быть установлен на**rxCLALLOWABSENT**. Это будет объяснено подробнее позже;

3.**RwUInt32**ценить**сдержанный**, который, что неудивительно, зарезервирован для внутреннего использования и который следует просто установить на**rxCLЗАРЕЗЕРВИРОВАНО**.

-

Максимальное количество интересующих кластеров, которое**RxNodeDefinition**может указать, задается значением**RXNODEMAXCLUSTERSOFINTEREST**.

В*Пример кода*выше,**clOfInterest**массив кластеров интереса. Он выражает интерес к четырем кластерам:

- 2D вершины экранного пространства;
- Состояние рендеринга;
- Интерполянты (создаются путем отсечения для ускорения многопроходного рендеринга – см. справочную документацию API) **RxNodeDefinitionGetClipTriangle()**для получения более подробной информации);
- Второй набор текстурных UV-координат (первый набор находится в вершинах экранного пространства).

Ни один из этих кластеров не указан как обязательно присутствующий.

Спецификация входных требований

Входные требования узла указаны в массиве**входные требования** (перечислимый тип)**RxClusterValidityReq**) в**ио**член **RxNodeDefinition**. Этот массив должен содержать одну запись для каждого интересующего кластера, значения которой могут быть одними из следующих:

- **rxCLREQ_DONTWANT -**узел должен использовать это значение, если он намерен использовать рассматриваемый кластер, но не будет использовать какие-либо данные, которые уже могут присутствовать в кластере. Обычно это означает, что узел перезапишет или повторно инициализирует данные кластера.
- **rxCLREQ_REQUIRED**-Узел должен использовать это значение, если ему требуется, чтобы кластер содержал действительные данные при входе в узел.
- **rxCLREQ_OPTIONAL**-узел должен использовать это значение, если оно ***способный***использовать любые данные, которые уже могут присутствовать в кластере (например, если данные могут быть использованы для оптимизации работы узла), но не***нуждаться***сделать это, чтобы функционировать.

В вышеизложенном*Пример кода*,**inputReqs**массив входных требований. Он указывает, что все интересующие кластеры должны войти в узел с допустимыми данными, за исключением дополнительного кластера UVs, который может отсутствовать.

Спецификация выходов

Каждый из выходов узла должен быть указан в записи **выходы**массив (тип **RxOutputSpec**) принадлежащий**ио**член **RxNodeDefinition**. Количество выходов должно быть указано в **numOutputs**член**ио**. Максимально допустимое количество выходов указано**RXNODEMAXВЫХОДЫ**.

The**RxOutputSpec**состоит из трех членов:

1.**имя**содержит строку, используемую для идентификации вывода. Это используется во время построения конвейера, при редактировании топологии конвейера (см. пример кода в предыдущей главе,*Обзор PowerPipe*);

2. Массив**выходКластеры**(типа**RxClusterValid**);

3.**всеДругиеКластеры**(также типа**RxClusterValid**).

The**выходКластеры**Массив должен содержать одну запись для каждого интересующего кластера, значения которой могут быть одними из следующих:

- **rxCLVALID_NOCHANGE**–это значение указывает, что кластер будет находиться в том же состоянии (т.е. содержать допустимые данные или нет) при выходе из узла через текущий выход, в котором он был при входе в узел.
- **rxCLVALID_VALID**–это значение указывает, что кластер будет содержать действительные данные при выходе из узла через текущий выход.
- **rxCLVALID_INVALID**–это значение указывает, что кластер не будет содержать действительных данных при выходе из узла через текущий выход.

Любые кластеры в текущем пакете, которые не являются одним из интересующих узел кластеров, будут обработаны, как указано**всеДругиеКластеры**. Это дает узлу возможность уничтожить все остальные кластеры в пакете, установив значение **rxCLVALID_INVALID**, хотя в большинстве случаев он установлен на **rxCLVALID_NOCHANGE**.

-

Максимальное количество выходов, которое**RxNodeDefinition**может указать, задается значением **RXNODEMAXВЫХОДЫ**.

В вышеизложенном*Пример кода*, определены два выхода, в**выходы**Массив, названный "UVsOut" и "PassThrough".**вывод1Кластеры**Массив определяет состояние кластеров, проходящих через «UVsOut» – все кластеры будут содержать действительные данные.**выход2Кластеры**определяет состояние кластеров, проходящих через «PassThrough» — первые два кластера будут содержать действительные данные, а последние два кластера сохранят свое состояние с момента входа в узел. **выходы**массив ссылается на эти выходные спецификации.

Погоня за зависимостью

Входные и выходные спецификации узла используются в процессе отслеживания зависимостей (представленном в разделе построения конвейера предыдущей главы, *Обзор PowerPipe*), который возникает, когда трубопровод находится в конце строительства трубопровода, внутри функции**RxPipelineUnlock**.

Целью отслеживания зависимостей является анализ того, где каждый кластер используется в конвейере, и при этом оптимизация потока данных в конвейере во время выполнения. Кроме того, он проверяет, что все требования узлов в конвейере могут быть удовлетворены (например, если узел требует нормалей объектного пространства, то для всех пакетов, входящих в узел, кластер нормалей объектного пространства должен быть инициализирован предыдущим узлом в конвейере).

Преследование зависимости состоит из следующих этапов:

1. Структура трубопровода «разворачивается» в линейный массив узлов. Этот процесс известен как топологическая сортировка;

2. Топологическая сортировка всегда должна давать уникальный результат, поскольку конвейеры PowerPipe ограничены наличием подключенного,*ациклический*графовая структура и иметь только одну точку входа. Следовательно, если топологическая сортировка не удалась, то конвейер не является связным ациклическим графом с одной точкой входа и поэтому недействителен;

3. Время жизни каждого кластера отслеживается через возможные пути выполнения в конвейере. Узел будет определен для создания кластера для конкретного пути выполнения, если это первый узел в пути, для которого кластер выходит через выход, который помечает кластер как **rxCLVALID_VALID**. Узел будет определен для уничтожения кластера, если он является последним узлом в пути, входная спецификация которого перечисляет кластер как **rxCLREQ_REQUIRED**или **rxCLREQ_OPTIONAL**или если выход узла помечает кластер как**rxCLVALID_INVALID** ;

4. Обнаружены невыполненные зависимости узлов. Если узел запрашивает кластер как**rxCLVALID_VALID**в точке пути, где находится кластер определяется как «мертвый», то конвейер недействителен. Если два или более путей выполнения сходятся в узле, который запрашивает кластер как **rxCLVALID_VALID**и кластер недействителен для*каждый*входящий путь, то конвейер недействителен. Это потому, что конвейеры, в которых требования к узлу*мощь*не быть выполненными во время выполнения, не допускаются.

5. После определения валидности конвейера время жизни кластеров на разных путях будет объединено, и для каждого узла в конвейере будет составлен минимальный список активных в данный момент кластеров, который будет сохранен в**RxPipelineNode**структура.

Создание минимальных списков активных кластеров имеет два преимущества. Во-первых, и это тривиально, меньше памяти используется для хранения этих списков, и это может обеспечить небольшое повышение скорости узлов, поскольку они обращаются к спискам во время выполнения. Во-вторых, память, используемая мертвыми кластерами, может быть освобождена как можно скорее в конвейере.

31.2.4 Методы узлов

Метод тела узла (тип**RxNodeBodyFn**) будет рассмотрен в следующем разделе (в конце концов, именно там происходит вся обработка данных во время выполнения). В этом разделе будут рассмотрены оставшиеся методы узла, функция которых, в общем и целом, обычно заключается только в инициализации области закрытых данных узла конвейера.

RxNodeInitFn

The**RxNodeInitFn**данного узла (т.е.**RxNodeDefinition**), вызывается во время **RxPipelineUnlock()**первый раз, когда конвейер, содержащий этот узел (т.е. ссылающийся на него)**RxNodeDefinition**) разблокирован. Ожидаемое использование этой функции — настроить некоторую глобальную память рабочего пространства или, возможно, таблицу поиска. Затем это может использоваться функцией тела узла во время выполнения конвейера для*все*трубопроводы, в которых он появляется.

Ниже представлен прототип**RxNodeInitFn**:

```
typedef RwBool   (*RxNodeInitFn)
    ( RxNodeDefinition * self );
```

Единственный параметр — указатель на определение узла-владельца. Возвращаемое значение**ЛОЖЬ**будет означать ошибку и вызывать**RxPipelineUnlock()** потерпеть неудачу.

RxNodeTermFn

The**RxNodeTermFn**является дополнением к**RxNodeInitFn**. Для данного узла он вызывается в последний раз, когда конвейер, содержащий этот узел, заблокирован или уничтожен. Ожидается, что он будет использоваться просто для освобождения памяти, выделенной в дополнительном**RxNodeInitFn**.

Ниже представлен прототип**RxNodeTermFn**:

```
typedef void (*RxNodeTermFn)
    ( RxNodeDefinition * self );
```

Единственным параметром является указатель на определение узла-владельца.

RxPipelineNodeInitFn

Для данного узла его**RxPipelineNodeInitFn**вызывается во время **RxPipelineUnlock()**(после того как**RxNodeInitFn**, если присутствует) для любого конвейера, содержащего узел. Ожидаемое использование этой функции — настройка области частных данных узла, которая выделяется**RxPipelineUnlock()**.

Ниже представлен прототип**RxPipelineNodeInitFn**:

```
typedef RwBool   (*RxPipelineNodeInitFn)
      ( RxPipelineNode * self );
```

Единственный параметр — указатель на узел-владелец конвейера. Возвращаемое значение **ЛОЖЬ**будет означать ошибку и вызывать**RxPipelineUnlock()**потерпеть неудачу.

The**RxPipelineNodeInitFn**используется в*Пример кода*выше было **_UVInterpNodePipelineNodeInitFn**.

Приватная область данных узла будет выделена во время**RxPipelineUnlock()**, до его**RxPipelineNodeInitFn**называется. Размер этой области памяти должен быть указан в**pipelineNodePrivateDataSize**член **RxNodeDefinition**.

Функции**RxPipelineNodeCreateInitData()**и **RxPipelineNodeGetInitData()** может использоваться для выделения и извлечения «данных инициализации» для узла конвейера перед вызовом **RxPipelineUnlock()**для содержащего конвейера. Это может быть использовано для настройки информации, которая будет параметризовать настройку частных данных, выполняемую узлом**RxPipelineNodeInitFn**. Очевидно, что это не обязательно (так как настройка личных данных может быть легко выполнена после вызова **RxPipelineUnlock()**) но может быть удобно в некоторых случаях. См. справочную документацию API для**RxPipelineNodeCreateInitData()**для получения более подробной информации.

Первоначальной целью схемы инициализационных данных была поддержка функции **RxPipelineClone()**. Основная идея заключается в том, что конвейер, созданный во внешнем коде, будет настроен неизвестными вызовами API. Данные инициализации могут эффективно «запоминать» эффект этих вызовов и, таким образом, облегчать автоматическую инициализацию узла конвейера в клонированном конвейере. Использование **RxPipelineClone()**больше не рекомендуется и может быть удален из API в будущих версиях.

RxPipelineNodeTermFn

The**RxPipelineNodeTermFn**является дополнением к **RxPipelineNodeInitFn**. Для данного узла он называется (перед **RxNodeTermFn**, если присутствует), когда конвейер, содержащий этот узел, заблокирован или уничтожен. Ожидаемое использование этой функции — освобождение выделений, на которые ссылаются в области личных данных узла.

Ниже представлен прототип**RxPipelineNodeTermFn**:

```
typedef void (*RxPipelineNodeTermFn)
    ( RxPipelineNode * self );
```

Единственным параметром является указатель на узел-владелец конвейера.

Приватная область данных узла будет освобождена во время**RxPipelineLock()** или **RxPipelineDestroy()**, после его**RxPipelineNodeTermFn**называется.

Справочная документация API для**RxPipelineNodeConfigFn**и **RxConfigMsgHandlerFn** типы и функции **RxPipelineNodeSendConfigMsg()**дайте подробности об их использовании. В настоящее время они не используются ни одним из узлов конвейера, поставляемых с RenderWare Graphics, и маловероятно, что разработчикам они понадобятся. Они могут быть удалены из API в более поздних версиях.

31.3 Метод тела узла

The**RxNodeBodyFn**содержит функциональность обработки данных, которая определяет поведение узла во время выполнения. Функция тела узла может делать много вещей:

- Узел может создавать пакеты;
- Он может уничтожать или изменять входящие в него пакеты;
- Это может привести к прекращению выполнения конвейера;
- Он может передавать входящие пакеты на свои выходы;
- Он может создавать, изменять или уничтожать кластерные данные, содержащиеся в пакетах;
- Он может получать доступ к данным в пределах частной области узла или полностью за пределами области действия конвейера;
- Он может получать доступ к функциям RenderWare Graphics API или функциям, специфичным для платформы.

Существует очень мало ограничений на действия, которые могут быть выполнены внутри**RxNodeBodyFn**, хотя есть несколько ограничений:

- **RxPipelineExecute()**не следует называть;
- Содержащий его конвейер не должен редактироваться или уничтожаться (это довольно очевидно);
- В любой момент времени может существовать только один пакет.

Этот последний пункт упоминался в предыдущей главе.*Обзор PowerPipe*, в разделе *Поток данных в трубопроводах*. Рекомендуется кратко ознакомиться с этим, прежде чем продолжить.

Ниже представлен прототип**RxNodeBodyFn**:

```
typedef RwBool   (*RxPipelineNodeBodyFn)
        ( RxPipelineNode * сам,
          const RxPipelineNodeParam *params );
```

The**себя**Параметр указывает на текущий узел конвейера.**параметры** параметр указывает на структуру типа**RxPipelineNodeParams**, как показано здесь:

```
структура RxPipelineNodeParam {

    пустота      * dataParam;
    RxHeap   * куча;
};
```

Целью инкапсуляции параметров узлов в этой структуре является просто обеспечение возможности прозрачного расширения списка параметров узлов в будущих версиях RenderWare Graphics (и без увеличения стоимости вызовов **RxNodeBodyFn**Функции). Макросы API **RxPipelineNodeParamGetData()**и **RxPipelineNodeParamGetHeap()** следует использовать для извлечения двух членов этой структуры. **dataParam**член содержит**данные**указатель передан **RxPipelineExecute()**. Элемент кучи содержит указатель на кучу (пользовательский распределитель памяти), используемую для текущего выполнения конвейера — это будет объяснено далее в*Куча трубопровода*ниже.

Если**RxNodeBodyFn**возвращается**ЛОЖЬ**то это означает ошибку. Это приведет к тому, что конвейер завершит работу как можно скорее (никакие дальнейшие функции тела узла не будут выполнены) и**RxPipelineExecute()**вернется**НУЛЕВОЙ**.

-

Обратите внимание, что возвращение**истинный**преждевременно (т. е. до отправки пакета, входящего в текущий узел) не всегда может привести к выходу конвейера без выполнения каких-либо дополнительных тел узлов. Например, узел, отправивший пакет текущему узлу, может создать новый пакет и отправить его другому узлу.

The**RxNodeBodyFn**используемый*Пример кода*выше есть**UVInterpNode**.

В следующих двух подразделах будет рассмотрен API, используемый в **RxNodeBodyFn**для обработки пакетов и кластеров. После этого будет представлен пример кода, содержащий полный**RxNodeBodyFn,**который даст пример того, как API может использоваться на практике.

31.3.1 Манипуляция пакетами

RxPacketFetch()

Если узел ожидает получить пакет от предыдущего узла в конвейере, он может извлечь этот пакет с помощью функции**RxPacketFetch()**. Узел следует проверить (по крайней мере в отладочной сборке) наличие этого пакета перед его использованием. Предыдущие узлы в конвейере могут вести себя не так, как ожидалось.

RxPacketCreate()

Если узел не получает или не ожидает получения пакета, то он может вместо этого создать его, вызвав**RxPacketCreate()**. Обратите внимание, что из-за Модель выполнения тела вложенного узла (описанная в*Поток данных в трубопроводах* раздел главы*Обзор PowerPipe*) только один пакет может существовать одновременно. Следовательно, вы должны уничтожить существующий пакет перед созданием нового.

RxPacketDispatch() и RxPacketDispatchToPipeline()

Отправка пакета через один из выходов текущего узла может быть осуществлена с помощью функции**RxPacketDispatch()**. Выход, на который должен быть отправлен пакет, указывается просто как нулевой индекс в**выходы**массив, содержащийся в**ио**член узла **RxNodeDefinition**(Чтобы избежать использования неправильного индекса, может быть полезно использовать явно именованный**#определять**как индекс и использовать это**#определять**в функции, которая инициализирует узел**RxNodeDefinition**). Отправка пакета на другой конвейер может быть достигнута с помощью функции **RxPacketDispatchToPipeline()**.

Оба**RxPacketDispatch()**и**RxPacketDispatchToPipeline()**может быть передан указатель пакета NULL. Это позволяет узлу передавать выполнение новому узлу или конвейеру без необходимости создания или извлечения пакета. Хорошим примером узла, который не создает и не извлекает пакеты, является узел PVSWorldSector.csl (представленный в главе*Потенциально видимые наборы*), единственной целью которого является преждевременное завершение конвейера на основе информации о видимости в данных плагина визуализируемого объекта (т. е. если объект перекрыт, узел PVS гарантирует, что он не будет визуализирован).

Обратите внимание, что из-за модели выполнения вложенного узла отправка пакета вызывает *разрушение*этого пакета. Это связано с тем, что выполнение будет продолжаться в рамках отправки до конца конвейера, после чего пакеты автоматически уничтожаются. Следовательно, альтернативой уничтожению одного пакета перед созданием нового является отправка первого пакета.

31.3.2 Манипуляция кластером

В пределах**RxNodeBodyFn**, кластер инкапсулирован внутри**RxCluster** структура, как показано здесь:

```
структура RxCluster
{
    RwUInt16            флаги;
    RwUInt16            шаг;
    пустота             * данные;
    пустота             * текущие данные;
    RwUInt32            numAlloced;
    RwUInt32            numИспользовано;
    RxPipelineCluster   * clusterRef;
    RwUInt32            атрибуты;
    RwUInt32            площадка[1];
}
```

The**прокладка**член просто дополняет структуру до ровных 32 байтов и **clusterRef** member предназначен для внутреннего использования. Остальные члены будут описаны как часть следующих описаний API манипуляции кластером.

Инициализация данных кластера

Кластер (как описано в*Поток данных в трубопроводах*раздел предыдущей главы*Обзор PowerPipe*) просто содержит массив элементов данных.**шаг**из этих элементов данных хранится в члене шага **RxCluster**. Это значение устанавливается при инициализации данных кластера, что может быть выполнено одним из двух способов.

Во-первых,**RxClusterInitializeData()**может быть использован для инициализации массива данных кластера с новым распределением, заданной длины и заданного шага. Если кластер уже содержал данные, они будут сначала освобождены.**данные**член будет указывать на новый массив и**numAlloced**член запишет свою длину. См. API справочник для **RxClusterInitializeData()**для получения более подробной информации.

Во-вторых,**RxCluster**можно сделать так, чтобы он указывал на существующий массив данных, используя либо**RxClusterSetData()**или**RxClusterSetExternalData()**. Как подробно объясняется в справочной документации API для этих двух функций, данные кластера могут быть «внутренними» или «внешними». По умолчанию кластер будет иметь «внутренние» данные, что означает, что он выделяется (обычно **RxClusterInitializeData()**) в куче, которая используется для текущего выполнения конвейера (куча будет объяснена вскоре). Данные кластера являются «внешними», если они выделены вне текущей кучи. PowerPipe может выделять, перераспределять и освобождать внутренние данные кучи, но не внешние данные, поэтому внешние данные защищены от модификации.

Размер массива данных кластера можно изменить с помощью функции **RxClusterResizeData()**. Это сохраняет шаг и данные существующего массива (это может включать в себя копирование, согласно**RwRealloc()**).

Флаги состояния кластера

Статус данных кластера как «внутренний» или «внешний» хранится в**флаги** член**RxCluster**структура. Флаги кластера имеют тип**RwUInt16**, используемый как битовое поле, которое может содержать следующие флаги:

- **rxCLFLAGS_CLUSTERVALID**–если этот флаг установлен, это в основном означает, что массив данных этого кластера был инициализирован и еще не уничтожен. Если этот флаг не установлен, все остальные члены**RxCluster**структуру следует считать недействительной.

- **rxCLFLAGS_EXTERNAL**–этот флаг означает, что данные кластера являются «внешними», т.е. не выделены из текущей кучи.

- **rxCLFLAGS_EXTERNALMODIFIABLE**–этот флаг (который на самом деле является новым флагом, объединенным с помощью ИЛИ**rxCLFLAGS_EXTERNAL**флаг) означает, что, хотя данные кластера являются внешними по отношению к текущей куче и, следовательно, не могут быть освобождены или изменены в размере, их можно редактировать на месте.

- **rxCLFLAGS_MODIFIED**–этот флаг устанавливается каждый раз, когда **RxClusterLockWrite()**(видеть*Блокировка кластера*ниже) называется. По сути, его можно использовать для мониторинга того, были ли изменены данные кластера.

Данные кластера могут быть уничтожены с помощью функции **RxClusterDestroyData()** . Эта функция освобождает данные кластера (если они внутренний) и затем отмечает кластер как недействительный, очищая его **rxCLFLAGS_EXTERNAL**флаг.

Блокировка кластера

Для того, чтобы получить**RxCluster**указатель из пакета, для использования и/или изменения его данных вы можете использовать функции **RxClusterLockRead()**и**RxClusterLockWrite()**. **RxClusterLockWrite()**необходимо использовать, если данные кластера должны быть изменены каким-либо образом (включая изменение размера или уничтожение массива данных кластера). Обратите внимание, что **RxClusterUnlock()**существует, но пока он делает это просто для обеспечения симметрии путем зеркального отображения**RxClusterLockRead()**и **RxClusterLockWrite()**.

Когда кластер с внешними данными заблокирован для записи, его данные копируются в текущую кучу, чтобы кластер можно было сделать внутренним. Изменение размера данных кластера также сделает кластер внутренним. Уничтожение внешних данных не освободит данные, а просто пометит кластер как недействительный.

Куча трубопровода

"Текущая куча", упомянутая несколько раз выше, представляет собой область памяти и пользовательский распределитель, разработанный специально для использования с PowerPipe. Он был оптимизирован для обеспечения очень быстрого распределения на основе трех условий, которые обычно встречаются при выполнении конвейеров PowerPipe:

1. Когда выполнение конвейера завершается, все выделенные во время его выполнения памяти, которые все еще присутствуют в куче, могут быть безопасно отброшены;

2. Любое освобождение блоков памяти будет происходить примерно в обратном порядке по отношению к тому, в котором эти блоки памяти были первоначально выделены;

3. Будет сделано лишь небольшое количество распределений.

Куча, используемая во время выполнения конвейера, может быть извлечена, как упоминалось выше, путем передачи**RxPipelineNodeParam**параметр **RxNodeBodyFn**к **RxPipelineNodeParamGetHeap()**. При желании вы можете использовать эту кучу для быстрого выделения временного рабочего пространства.

PowerPipe в настоящее время использует одну глобальную кучу для всех конвейерных исполнений, которая будет автоматически увеличиваться при необходимости. Необходима только одна куча, поскольку RenderWare Graphics в настоящее время не является многопоточной, поэтому в определенный момент времени может выполняться только один конвейер. Вы можете получить глобальную кучу с помощью функции**RxHeapGetGlobalHeap()**. Если использование кучи будет полезным в других областях вашего приложения, вы можете создать ее с помощью функции **RxHeapCreate()**. Более подробную информацию можно найти в справочной документации API для этой и других функций API кучи.

Каждый раз, когда**RxPipelineExecute()**вызывается, он очистит глобальную кучу перед началом выполнения конвейера. Чтобы разрешить сохранение данных кучи от одного выполнения конвейера к другому, логический параметр **RxPipelineExecute(), heapReset,** может быть установлено значение**ЛОЖЬ**. Эта возможность используется в**RwIm3D**(на некоторых платформах), где данные, сгенерированные конвейером, выполняются в **RwIm3DTransform()**должен быть в состоянии выдержать несколько вызовов **RwIm3DRenderPrimitive()**или **RwIm3DRenderIndexedPrimitive()**, каждый из которых выполняет конвейер, использующий кэшированные данные.

Доступ к кластерным данным

Для облегчения доступа к элементам данных в кластере (для чтения или записи) в нем есть дополнительный элемент указателя.**RxCluster** структура, **текущие данные**. Это действует как «курсор» для текущей точки доступа к данным кластера. Этот курсор можно сбросить, чтобы он указывал на первую запись в массиве данных кластера, используя**RxClusterResetCursor()**. Следующие функции вызывают сброс курсора кластера: **RxClusterLockRead()**, **RxClusterLockWrite()**, **RxClusterInitializeData()**,**RxClusterResizeData()**, **RxClusterSetData()**и**RxClusterSetExternalData()**.

RxClusterGetCursorData()может использоваться для доступа к данным (соответствующего типа) в курсоре кластера.**RxClusterGetIndexedData()**может использоваться для прямого доступа к определенному элементу массива данных кластера.

Чтобы увеличить позицию курсора на один элемент в массиве данных кластера, используйте**RxClusterIncCursor()**. Чтобы уменьшить курсор на ту же величину, используйте**RxClusterDecCursor()**. Обратите внимание, что проверка границ не выполняется даже в отладочной сборке.

Использование массива данных кластера

The**numИспользовано**член**RxCluster**используется для отслеживания количества элементов в массиве данных кластера, которые были использованы (тело узла не обязательно будет знать заранее, сколько элементов кластера ему нужно будет использовать, поэтому оно может выделить консервативно большой массив данных). Обратите внимание, что это предполагает, что массив данных кластера заполняется непрерывно, от начала к концу. Макрос**RxClusterGetFreeIndex()**вернет текущее значение**numИспользовано**а затем увеличиваем его значение, намереваясь указать на первый свободный элемент массива данных кластера, предполагая, что он скоро будет использован.

RxClusterSetData()и**RxClusterSetExternalData()**оба установлены **numИспользовано** быть равным**numAlloced**.**RxClusterInitializeData()**наборы **numИспользовано**к нулю и **RxClusterResizeData()**уменьшает**numИспользовано**при необходимости (никогда не должно быть больше, чем**numAlloced**!).

Ответственность за обеспечение того, чтобы**numИспользовано**участник держится в курсе событий – это*важный*, так как невыполнение этого требования может привести к повреждению данных на последующих этапах передачи.

31.3.3 Пример кода

Далее следует пример кода, демонстрирующий типичные задачи, выполняемые в рамках**RxNodeBodyFn**:

```
RwBool
MyNodeBody(УзелПриемаКонвейера        * себя,
           RxPipelineNodeParam        * параметры)
{
    RxPacket      * пакет;
    RxCluster     * clNorms, *clCols;
    МоиЛичныеДанные * данные;
    RwReal          шкала;

    данные = (MyPvtData *)self->privateData;
    assert(NULL != data);
    масштаб = данные->масштаб;

    пакет = RxPacketFetch(self); если
    (NULL != пакет)
    {
        clNorms = RxClusterLockRead(пакет, CLNORMALSINDEX);
        assert(NULL != clNorms);
        clCols = RxClusterLockWrite(пакет, CLCOLORSINDEX, self); assert(NULL !=
        clCols);

        assert(clNorms->numUsed == clCols->numUsed);

        если ((NULL != RxClusterGetCursorData(clNorms, RwV3d)) &&
            (NULL != RxClusterGetCursorData(clCols, RwRGBA)))
        {
            RwUInt32    clРазмер, i;
            RwRGBA      цвет = {0, 0, 0, 255}; rTemp;
            RwReal

            clSize = clNorms->numUsed; для (i
            = 0;i < clSize;i++) {

                rTemp = RxClusterGetCursorData(clNorms, RwV3d)->x;
                assert((rTemp <= 1.0f) && (rTemp >= -1.0f));

                rTemp = 0,5f*(1,0f + rTemp)*масштаб;
                цвет.красный = (RwUInt8)(255,0f*rTemp);
               * RxClusterGetCursorData(clCols, RwRGBA) = цвет;

                RxClusterIncCursor(clNorms);
                RxClusterIncCursor(clCols);
            }
```

```
            RxPacketDispatch(пакет, RENDEROUTPUT, self);

            возврат (ИСТИНА);
        }
    }
    еще
    {
        RpAtomic *объект;

        объект = RxPipelineNodeParamGetData(params);
        утверждать(NULL != объект);

        пакет = RxPacketCreate(self);
        assert(NULL != пакет);

        если ((NULL != пакет) && (NULL != объект)) {

            clNorms = RxClusterLockWrite(
                            пакет, CLNORMALSINDEX,      себя);
            assert(NULL != clNorms); clCols =
            RxClusterLockWrite(
                            пакет, CLCOLORSINDEX,       себя);
            assert(NULL != clCols);

            clНормы  =  RxClusterInitializeData(
                            clNorms, 1, sizeof(RwV3d));
            clCols      =       RxClusterInitializeData(
                            clCols, 1, sizeof(RwRGBA));

            если ((NULL != RxClusterGetCursorData(clNorms, RwV3d)) &&
                (NULL != RxClusterGetCursorData(clCols, RwRGBA)) )
            {
                RwV3d   по умолчаниюОбычный;
                RwRGBA  Цвет по умолчанию;

                по умолчаниюОбычный  = * RpAtomicMyPluginGetNormal(объект);
                defaultColor         =  RpAtomicMyPluginGetColor(объект);

               * RxClusterGetCursorData(clNorms,          РвВ3д)   =
               defaultNormal;
                clNorms->numUsed =     1;
               * RxClusterGetCursorData(clCols,           РвРГБА)   =
               defaultColor;
                clCols->numUsed =      1;

                RxPacketDispatch(пакет, GEOMGENOUTPUT, self);

                возврат (ИСТИНА);
            }
```

```
            }
        }

        возврат (ЛОЖЬ);
}
```

Тело узла, показанное выше, имеет два интересующих кластера: кластер "нормалей" и кластер "цветов". Целью узла является настройка кластера цветов на основе x-координаты векторов, хранящихся в кластере нормалей. Это происходит следующим образом:

1. Первым действием узла является доступ к его частной области данных и получение коэффициента масштабирования, используемого в его основных вычислениях.

2. Далее узел пытается получить текущий пакет. Если такого пакета не существует, узел переходит к шагу 4. Предполагая, что пакет существует, узел заблокирует кластер нормалей для чтения и кластер цветов для записи. Он проверяет, что два кластера содержат одинаковое количество используемых элементов, поскольку его обработка требует взаимно-однозначного сопоставления между этими элементами.

3. Узел проверяет, существуют ли данные для двух кластеров, а затем переходит к пошаговому просмотру элементов данных каждого кластера, устанавливая цветовые элементы на значения, определяемые x-координатами нормальных векторных элементов кластера и масштабным коэффициентом из области личных данных узла. После выполнения этой обработки пакет отправляется на первый выход узла. Предполагается, что это приведет к узлам, которые будут использовать цветовой кластер пакета при рендеринге геометрических данных, содержащихся в других кластерах пакета.

4. Если пакет не найден при входе в этот узел, узел создаст новый пакет. Он инициализирует кластеры нормалей и цветов, чтобы они содержали по одному элементу, значения которых определяются данными плагина в объекте (**RpAtomic**) в настоящее время визуализируется. Затем он отправит новый пакет на второй выход узла. Предполагается, что это приведет к узлам, которые сгенерируют некоторую геометрию, используя заданные здесь значения нормали и цвета.

Далее следует список функций для использования в**RxNodeBodyFn**. Более подробную информацию можно найти в справочной документации API для этих функций:

Функции манипулирования RxPipelineNodeParam

- **RxPipelineNodeParamGetData()**
- **RxPipelineNodeParamGetHeap()**

Функции манипулирования RxPacket

- **RxPacketCreate()**

- **RxPacketDestroy()**
- **RxPacketFetch()**
- **RxPacketDispatch()**
- **RxPacketDispatchToPipeline()**

Функции манипуляции RxCluster

- **RxClusterLockRead()**
- **RxClusterLockWrite()**
- **RxClusterUnlock()**
- **RxClusterInitializeData()**
- **RxClusterResizeData()**
- **RxClusterDestroyData()**
- **RxClusterSetData()**
- **RxClusterSetExternalData()**
- **RxClusterSetStride()**
- **RxClusterGetCursorData()**
- **RxClusterGetIndexedData()**
- **RxClusterGetFreeIndex()**
- **RxClusterResetCursor()**
- **RxClusterIncCursor()**
- **RxClusterDecCursor()**

31.4 Предоставленные узлы

В этом разделе будут предоставлены краткие описания общих (то есть платформонезависимых) узлов, поставляемых с RenderWare Graphics в**RtGenCPipe** набор инструментов. Однако прежде чем это сделать, он представит стандартные кластеры, которые используются этими узлами. Платформозависимые кластеры и узлы будут представлены в последующих главах.

31.4.1 Стандартные кластеры

В этом разделе содержится введение в каждый из кластеров, используемых универсальными узлами:

- **RxClMeshState**
- **RxClRenderState**
- **RxClObjSpace3DVertices**
- **RxClCamSpace3DVertices**
- **RxClScrSpace2DVertices**
- **RxClИндексы**
- **RxClUV**
- **RxClRGBAs**
- **RxClCamNorms**
- **RxClИнтерполянты**
- **RxClVSteps**
- **RxClLights**
- **RxClРассеивание**

RxClMeshState

The**RxClMeshState**кластер содержит данные типа**RxMeshStateVector**, как показано здесь:

```
структура RxMeshStateVector {

    RwInt32          Флаги;
    пустота          * ИсходныйОбъект;
    RwMatrix         Obj2World;
    RwMatrix         Obj2Cam;
    RwТекстура       * Текстура;
    RwRGBA           МатКол;
    RxPipeline       * Трубопровод;
    RwPrimitiveType  ПримТип;
    RwUInt32         КоличествоЭлементов;
    RwUInt32         ЧислоВершин;
    RwInt32          ClipFlagsOr;
    RwInt32          ClipFlagsAnd;
    пустота          * Исходная сетка;
    пустота          * Объект данных;
};
```

Цель кластера состояния сетки — отслеживать некоторые общие черты геометрии, содержащейся в текущем пакете. Обычно массив данных кластера состояния сетки будет содержать только один элемент. Члены **RxMeshStateVector**Теперь будет описана структура:

- The**Флаги**член держит флаги типа**RxGeometryFlag**, который очень похож на**RpGeometryFlag**и**RpWorldFlag**. Эти флаги будут установлены узлом экземпляра (универсальные узлы экземпляра — это ImmInstance.csl, AtomicInstance.csl и WorldSectorInstance.csl).

- **SourceObject**настраивается ImmInstance.csl для указания на внутреннюю структуру (типа**rwIm3DPool**, используемый**RwIm3DTransform()**). Он настраивается AtomicInstance.csl и WorldSectorInstance.csl для указания на **RpМатериал**связанный с**RpMesh**из которого был создан текущий пакет. AtomicInstance.csl и WorldSectorInstance.csl настраивают **ИсточникMesh** указать на источник**RpMesh**и**ОбъектДанные**для отражения указателя void, переданного**RxPipelineExecute()**(который также доступен через **RxPipelineNodeParamGetData()**), тогда как ImmInstance.csl оставляет эти значения неинициализированными.

- **Obj2World**и**Obj2Cam**являются матрицами, содержащими, соответственно, преобразование объектного пространства в мировое пространство и преобразование объектного пространства в пространство камеры. Эти матрицы будут установлены узлом экземпляра.

- **Текстура**,**МатКол**и**Трубопровод**это текстура, цвет материала и конвейер материала. AtomicInstance.csl и WorldSectorInstance.csl настроят их из источника**RpMesh**текущего пакета. ImmInstance.csl инициализирует эти значения как NULL, непрозрачный белый и NULL соответственно.

- **PrimType**примитивный тип геометрии текущего пакета. **КоличествоЭлементов**— это количество элементов этого примитивного типа, содержащихся в индексах пакета (например, 17 для примитива на основе треугольника означает 17 треугольников, что преобразуется в 51 индекс для трилиста и 19 индексов для триполосы или тривеера). **ЧислоВершин**— это количество вершин, содержащихся в кластере(ах) вершин пакета. Обязательно обновите эти значения, где это уместно, при изменении размера кластеров и изменении количества используемых записей в массиве данных каждого кластера. Эти значения будут инициализированы узлом экземпляра.

- **ClipFlagsOr**и**ClipFlagsAnd**очищаются до нуля узлами экземпляра (ImmInstance.csl, AtomicInstance.csl и т.д.) и затем устанавливаются Transform.csl. Они являются, соответственно, побитовым ИЛИ и побитовым И **clipФлаги**всех вершин в текущем пакете.**ClipFlagsOr** может быть использовано для определения того,*любой*вершин пакета лежит за пределами одной из плоскостей отсечения текущей камеры и **ClipFlagsAnd** может быть использовано для определения того,*все*вершин пакета лежат вне некоторой плоскости. Тип**clipФлаги**(которые хранятся в вершинах пространства камеры – см. описание **RxClCamSpace3DVertices**кластер ниже) используется**RwClipFlag**. Эти флаги имеют смысл только после того, как было выполнено преобразование относительно пирамиды видимости камеры.

RxClRenderState

The**RxClRenderState**кластер содержит данные типа **RxRenderStateVector**, как показано здесь:

```
структура RxRenderStateVector {

    RwUInt32                Флаги;
    RwShadeMode             Режим тени;
    RwBlendFunction         SrcBlend;
    RwBlendFunction         DestBlend;
    RwRaster              * ТекстураРастр;
    RwTextureAddressMode    АдресРежимU;
    RwTextureAddressMode    AddressModeV;
    RwTextureFilterMode     Режим фильтра;
    RwRGBA                  Цвет границы;
    RwFogType               Тип тумана;
    RwRGBA                  Цвет тумана;
    RwUInt8                 * Таблица тумана;
};
```

Цель кластера состояния рендеринга — прикрепить различные настройки состояния рендеринга к геометрии, содержащейся в текущем пакете. Обычно массив данных кластера состояния рендеринга будет содержать только один элемент.

Использование этого кластера со временем изменилось. Первоначально он был создан, потому что исходная модель для отправки пакетов отличалась от текущей системы (с ее вложенным выполнением тела узла) и требовала, чтобы настройка состояния рендеринга была отложена до тех пор, пока узел терминала не «отправит» в конвейере. Теперь это не так, и RenderWare Graphics теперь принимает постоянную модель состояния рендеринга, где любой заданный фрагмент кода отвечает за настройку необходимого ему состояния рендеринга, но не за восстановление предыдущего состояния рендеринга. Следовательно, большинство узлов теперь устанавливают состояние рендеринга по мере выполнения, а не помещают состояние рендеринга в кластер состояния рендеринга для отложенной настройки.

The**ТекстураРастр**,**АдресModeU**,**AddressModeV**и**ФильтрРежим** члены кластера состояния рендеринга по-прежнему учитываются узлами отправки, как и **rxRENDERSTATEFLAG_VERTEXALPHAENABLE**флаг в**Флаги**член. Это просто сохраняет поведение состояния рендеринга конвейеров рендеринга до PowerPipe.

Члены**RxRenderStateVector**Теперь будет описана структура.

The**Флаги**элемент содержит флаги типа**RxRenderStateFlag**, которые объединяют множество булевых состояний рендеринга для повышения эффективности.

Режим тенисодержит желаемый режим затенения треугольника, типа **RwShadeMode**.

SrcBlendи**DestBlend**содержат желаемые режимы смешивания исходного и конечного фрагментов, типа**RwBlendFunction**.

АдресModeUи**AddressModeV**являются режимами адресации текстур U и V, типа**RwTextureAddressMode**.**ФильтрРежим**это режим фильтрации текстур, типа**RwTextureFilterMode**и**Цвет границы**желаемый цвет границы текстуры.

Тип туманаэто описанный тип тумана, типа**RwFogType**.**FogColor**желаемый цвет для тумана и**FogTable**это 256-запись**RwUInt8**таблица тумана.

Эти функции образуют API для**RxRenderStateVector**тип:

- **RxRenderStateVectorCreate()**
- **RxRenderStateVectorDestroy()**
- **RxRenderStateVectorGetDefaultRenderStateVector()**
- **RxRenderStateVectorSetDefaultRenderStateVector()**
- **RxRenderStateVectorLoadDriverState()**

RxClObjSpace3DVertices

The**RxClObjSpace3DVertices**кластер использует**RxObjSpace3DVertex** тип, который определяется по-разному для каждой платформы, но доступен через общий набор функций API (каждая из которых начинается с "**RxObjSpace3DVertex**"). Этот тип вершин описывает 3D-вершины в пространстве объектов, включая положение, цвет, нормаль и координаты текстуры. Данные вершин, полученные при создании экземпляров на большинстве платформ, относятся к этому типу.

RxClCamSpace3DVertices

The**RxClCamSpace3DVertices**кластер использует**RxCamSpace3DVertex** типа, как показано здесь:

```
структура RxCamSpace3DVertex {

    RwV3d       камераВертекс;
    RwUInt8     клипФлаги;
    RwUInt8     площадка[3];
    RwRGBAReal  кол;
    RwReal      у;
    RwReal      в;
};
```

Этот тип вершины можно изменить с помощью набора функций API (каждая из которых начинается с «**RxCamSpace3DVertex**"). Он описывает 3D вершины в пространстве камеры, и описания его членов теперь будут следующими:

CameraVertex— это трехмерная координата вершины в пространстве камеры.

Кол— это цвет с плавающей точкой, который используется во время освещения для накопления светового вклада от всех источников света, влияющих на текущую вершину.

Уи**В**— текстурные координаты для текущей вершины.

ClipFlagsимеет тип**RwClipFlag**. Это кодирует связь между положением текущей вершины и плоскостями усеченной пирамиды видимости.

RxClScrSpace2DVertices

The**RxClScrSpace2DVertices**кластер использует**RxScrSpace2DVertex** тип, который определяется по-разному для каждой платформы, но доступен через общий набор функций API (каждая из которых начинается с "**RxScrSpace2DVertex**"). Этот тип вершины описывает 2D вершины в экранном пространстве, включая цвет, положение и координаты текстуры. Положение будет содержать значение Z в экранном пространстве (оно будет сопоставлено с ZBuffer) и на некоторых платформах может содержать 3D-положение в пространстве камеры и обратное положение Z. Координаты текстуры на некоторых платформах могут быть предварительно умножены на обратное положение камеры по оси Z.**RxScrSpace2DVertex** тип — это тот, который передается в API 2D-растеризации (это то же самое, что **RwIm2DVertex**).

RxClИндексы

The**RxClИндексы**кластер использует**RxVertexIndex**тип. Этот кластер содержит индексы вершин, которые определяют топологию геометрии текущего пакета, которая должна интерпретироваться как примитивный тип, указанный в кластере состояния сетки **primType**член. Индексы индексируют массивы данных **RxClObjSpace3DVertices**, **RxClCamSpace3DVertices**и **RxClScrSpace2DVertices**кластеры. Обратите внимание, что (как упоминалось в*Общие трубопроводы*раздел предыдущей главы,*Обзор PowerPipe*) большинство общих узлов PowerPipe могут работать только с индексами трисписка (узлы, специфичные для строк, в основном могут работать только с индексами строчного списка).

RxClUV

The**RxClUV**кластер использует**РхУФ**типа, как показано здесь:

```
структура RxUV
{
    RwReal   у;
    RwReal   в;
};
```

Этот кластер используется для включения дополнительного набора координат текстуры вершин в конвейер (его использование будет описано далее в описании узла UVInterp.csl ниже). Его массив данных должен быть параллелен массиву данных **RxClObjSpace3DVertices**,**RxClCamSpace3DVertices**и **RxClScrSpace2DVertices**кластеры.

RxClRGBAs

The**RxClRGBAs**кластер использует**RwRGBAReal**тип. Этот кластер используется для включения дополнительного набора цветов вершин в конвейер (его использование будет описано далее в описании узла RGBAInterp.csl ниже). Его массив данных должен быть параллелен массиву данных**RxClObjSpace3DVertices**, **RxClCamSpace3DVertices**и**RxClScrSpace2DVertices**кластеры.

RxClCamNorms

The**RxClCamNorms**кластер использует**RxCamNorm**тип, который такой же, как **RwV3D**тип. Его цель — хранить нормали пространства камеры (полезные для рендеринга эффектов, таких как отображение окружения), которые не включены в**RxClCamSpace3DVertices**кластер.

RxClИнтерполянты

TheRxClИнтерполянтыкластер используетRxInterpтипа, как показано здесь:

```
структура RxInterp
{
    RxVertexIndex   оригинальныйВерт;
    RxVertexIndex   родительскийVert1;
    RxVertexIndex   родительскийVert2;
    RwReal          интерп;
};
```

Этот кластер содержит информацию, которая может использоваться для ускорения обрезки треугольников во время многопроходного рендеринга. Он (опционально) создается узлом ClipTriangle.csl и используется узлами UVInterp.csl и RGBAInterp.csl. Его использование будет описано более подробно в разделах, посвященных этим узлам ниже.

RxClVSteps

TheRxClVStepsкластер используетRxVStepтипа, как показано здесь:

```
структура RxVStep
{
    Шаг RwUInt8;
};
```

Этот кластер может быть использован для пропуска обработки неиспользуемых вершин в пакете, тем самым ускоряя работу узлов, которые в противном случае обрабатывали бы *все*вершин пакета. Например, PreLight.csl, Light.csl и PostLight.csl все запрашивают этот кластер какrxCLREQ_OPTIONAL. Если он присутствует, то они пропустят обработку "неиспользуемых" вершин. Такими вершинами могут быть, например, те, которые принадлежат только треугольникам с обратной гранью (т.е. невидимым и, следовательно, неотрисованным).

Каждый элемент вRxClVStepsМассив кластера содержитшагзначение, которое равно: количеству вершин, после ранее обработанной вершины, которые могут быть пропущены. Чтобы использовать эти значения, начните с начала RxVStepи массивы вершин и действуем следующим образом:

1. Обработать одну вершину;

2. Пропустить "**шаг**" вершины;

3. Увеличьте курсорRxVStepмассив по одному элементу.

Повторяйте этот процесс до тех пор, пока весь массив вершин не будет обработан. Если RxVStepмассив содержит допустимые данные, вам не нужно проверять границы его курсор.

RxClLights

The**RxClLights**кластер использует**RxLight**тип, который является просто указателем на**RpLight**.

RxClРассеивание

The**RxClРассеивание**кластер использует**RxScatter**типа, как показано здесь:

```
структура RxScatter
{
    RxPipeline      * трубопровод;
    RxPipelineNode  * узел;
};
```

Этот кластер используется для направления пакета по определенному пути в конвейере. Он используется узлом Scatter.csl, который описан ниже.

31.4.2 Общие узлы

В этом разделе содержится введение в каждый из общих узлов, включенных в RenderWare Graphics в**RtGenCPipe**Набор инструментов. Они перечислены здесь примерно в том порядке, в котором они встречаются в общих конвейерах (как описано в главе*Обзор PowerPipe*):

- **ImmInstance.csl**
- **Трансформировать.csl**
- **ImmStash.csl**
- **ImmRenderSetup.csl**
- **ImmMangleLineIndices.csl**
- **ImmMangleTriangleIndices.csl**
- **CullTriangle.csl**
- **ClipTriangle.csl**
- **ClipLine.csl**
- **ОтправитьTriangle.csl**
- **SubmitLine.csl**
- **AtomicInstance.csl**

- **AtomicEnumerateLights.csl**
- **WorldSectorInstance.csl**
- **WorldSectorEnumerateLights.csl**
- **МатериалScatter.csl**
- **Scatter.csl**
- **PreLight.csl**
- **Свет.csl**
- **PostLight.csl**
- **FastPathSplitter.csl**
- **RGBAInterp.csl**
- **UVInterp.csl**
- **Клон.csl**

Также есть введение в*Отладочные узлы*.

Расширение "**.csl**" в строках имени узла используется для идентификации узла как принадлежащего Criterion Software Ltd.

ImmInstance.csl

Цель ImmInstance.csl — создать и инициализировать пакет. Он создает экземпляры данных, переданных**RwIm3DTransform()**в **RxClObjSpace3DVertices**кластер и инициализирует**RxClMeshState**и **RxClRenderState**кластеры соответственно. Узел имеет один выход, через которые проходят инициализированные пакеты. Входные требования этого узла:

RxClObjSpace3DVertices	-	**rxCLREQ_DONTWANT**
RxClMeshState	-	**rxCLREQ_DONTWANT**
RxClRenderState	-	**rxCLREQ_DONTWANT**

Характеристики первого выхода этого узла:

RxClObjSpace3DVertices	-	**rxCLVALID_VALID**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_VALID**

См. справочную документацию API для **RxNodeDefinitionGetImmInstance()**для получения более подробной информации.

Трансформировать.csl

Цель Transform.csl — преобразовать вершины объектного пространства в пространство камеры и, где это возможно, в пространство экрана. Он инициализирует **RxClCamSpace3DVertices**и**RxClScrSpace2DVertices**кластеры из **RxClObjSpace3DVertices**кластер; все они в конечном итоге будут иметь одинаковое количество элементов в своих массивах данных (таким образом, что индексы вершин в **RxClИндексы**кластер будет применяться одинаково ко всем трем). Флаги отсечения генерируются для каждой вершины и сохраняются в**RxClCamSpace3DVertices** Кластер.**ClipFlagsOr**и**ClipFlagsAnd**члены **RxClMeshState**кластер (см. раздел об этом выше) настраивается путем объединения флагов всех вершин в пакете. Этот узел также выполняет ту же настройку освещения, что и PreLight.csl (см. раздел об этом ниже для получения более подробной информации).

Узел имеет два выхода. Входные требования этого узла:

RxClObjSpace3DVertices	-	**rxCLREQ_REQUIRED**
RxClCamSpace3DVertices	-	**rxCLREQ_DONTWANT**
RxClScrSpace2DVertices	-	**rxCLREQ_DONTWANT**
RxClMeshState	-	**rxCLREQ_REQUIRED**

Характеристики первого выхода этого узла:

RxClObjSpace3DVertices	-	**rxCLVALID_NOCHANGE**
RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClMeshState	-	**rxCLVALID_VALID**

Характеристики второго выхода этого узла:

RxClObjSpace3DVertices	-	**rxCLVALID_NOCHANGE**
RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClMeshState	-	**rxCLVALID_VALID**

См. справочную документацию API для **RxNodeDefinitionGetTranform()** для получения более подробной информации.

ImmStash.csl

Целью ImmStash.csl является «сохранение» содержимого входящего пакета (всех перечисленных ниже кластеров) в глобальной структуре состояния, так чтобы пакет можно было восстановить в последующем.**RwIm3D**конвейеры рендеринга с помощью узла ImmRenderSetup.csl (см. ниже).

Этот узел не имеет выходов. Входящие пакеты уничтожаются после того, как их содержимое было спрятано. Входные требования этого узла:

RxClObjSpace3DVertices	-	**rxCLREQ_OPTIONAL**
RxClCamSpace3DVertices	-	**rxCLREQ_OPTIONAL**
RxClScrSpace2DVertices	-	**rxCLREQ_OPTIONAL**
RxClMeshState	-	**rxCLREQ_OPTIONAL**
RxClRenderState	-	**rxCLREQ_OPTIONAL**

См. справочную документацию API для **RxNodeDefinitionGetImmStash()** для получения более подробной информации.

ImmRenderSetup.csl

ImmRenderSetup.csl создает пакет и инициализирует его из глобальных данных «хранилища», созданных в предыдущем**RwIm3D**конвейер преобразования с помощью узла ImmStash.csl.

Этот узел имеет два выхода. Пакеты с индексами проходят через первый выход, а пакеты без индексов проходят через второй выход. Входные требования этого узла:

RxClObjSpace3DVertices	-	**rxCLREQ_DONTWANT**
RxClCamSpace3DVertices	-	**rxCLREQ_DONTWANT**
RxClScrSpace2DVertices	-	**rxCLREQ_DONTWANT**
RxClMeshState	-	**rxCLREQ_DONTWANT**
RxClRenderState	-	**rxCLREQ_DONTWANT**
RxClИндексы	-	**rxCLREQ_DONTWANT**

Характеристики первого выхода этого узла:

RxClObjSpace3DVertices	-	**rxCLVALID_VALID**
RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_VALID**

Характеристики второго выхода этого узла:

RxClObjSpace3DVertices	-	**rxCLVALID_VALID**
RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_INVALID**

См. справочную документацию API для **RxNodeDefinitionGetImmRenderSetup()**для получения более подробной информации.

ImmMangleTriangleIndices.csl

Целью ImmMangleTriangleIndices.csl является преобразование индексов в форме tristrip или tri-fan в форму tri-list или генерация индексов tri-list, если индексы в данный момент отсутствуют. Это необходимо, поскольку большинство узлов-обработчиков треугольников обрабатывают только примитив tri-list и не могут обрабатывать неиндексированные tri-lists. Если это изменится в будущем, этот узел может быть удален.

Этот узел имеет один выход. Входные требования этого узла:

RxClMeshState	-	**rxCLREQ_REQUIRED**
RxClИндексы	-	**rxCLREQ_OPTIONAL**

Характеристики первого выхода этого узла:

RxClMeshState	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_VALID**

См. справочную документацию API для **RxNodeDefinitionGetImmMangleTriangleIndices()**для получения более подробной информации.

ImmMangleLineIndices.csl

Целью ImmMangleLineIndices.csl является преобразование индексов в форме полилинии в форму списка линий или генерация индексов списка линий, если в данный момент индексы отсутствуют. Это необходимо, поскольку большинство универсальных узлов обработки линий обрабатывают только примитив списка линий и не могут обрабатывать неиндексированные списки линий. Если это изменится в будущем, этот узел может быть удален.

Этот узел имеет один выход. Входные требования этого узла:

RxClMeshState	-	**rxCLREQ_REQUIRED**
RxClИндексы	-	**rxCLREQ_OPTIONAL**

Характеристики первого выхода этого узла:

RxClMeshState	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_VALID**

См. справочную документацию API для **RxNodeDefinitionGetImmMangleLineIndices()** для получения более подробной информации.

CullTriangle.csl

Этот узел удаляет треугольники из кластера индексов, если они обращены назад по отношению к текущей камере. Треугольники, полностью находящиеся за пределами экрана (вне пирамиды видимости), также удаляются.

Узел имеет два выхода. Пакеты, в которых*все*Треугольники отбраковываются и отправляются на второй выход. Входные требования этого узла:

RxClCamSpace3DVertices	-	**rxCLREQ_REQUIRED**
RxClScrSpace2DVertices	-	**rxCLREQ_REQUIRED**
RxClИндексы	-	**rxCLREQ_REQUIRED**
RxClMeshState	-	**rxCLREQ_REQUIRED**

Характеристики первого из выходов этого узла:

RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_VALID**
RxClMeshState	-	**rxCLVALID_VALID**

Характеристики второго выхода этого узла:

RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_INVALID**
RxClMeshState	-	**rxCLVALID_VALID**

См. справочную документацию API для **RxNodeDefinitionGetCullTriangle()** для получения более подробной информации.

ClipTriangle.csl

ClipTriangle.csl обрезает треугольники до усеченной пирамиды текущей камеры. Любые новые вершины, сгенерированные во время обрезки, проецируются (так что и положения в пространстве камеры, и положения в пространстве экрана, и координаты текстуры являются правильными) и добавляются к концам **RxClCamSpace3DVertices**и **RxClScrSpace2DVertices**массивы вершин кластера и **RxClMeshState** кластеры'**ЧислоВершин**Член обновлен. Новые треугольники добавлены в **RxClИндексы**кластер и**RxClMeshState**кластера**КоличествоЭлементов** Участник обновлен.**RxClИнтерполянты**Кластер, который выводится, используется для ускорения многопроходного рендеринга (см. разделы по UVInterp.csl и RGBAInterp.csl ниже).

Узел имеет два выхода. Пакеты, в которых все треугольники обрезаны, отправляются на второй выход. Входные требования этого узла:

RxClCamSpace3DVertices	-	**rxCLREQ_REQUIRED**
RxClScrSpace2DVertices	-	**rxCLREQ_REQUIRED**
RxClИндексы	-	**rxCLREQ_REQUIRED**
RxClMeshState	-	**rxCLREQ_REQUIRED**
RxClRenderState	-	**rxCLREQ_OPTIONAL**
RxClИнтерполянты	-	**rxCLREQ_DONTWANT**

Характеристики первого из выходов этого узла:

RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_VALID**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_NOCHANGE**
RxClИнтерполянты	-	**rxCLVALID_VALID**

Характеристики второго выхода этого узла:

RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_INVALID**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_NOCHANGE**
RxClИнтерполянты	-	**rxCLVALID_INVALID**

См. справочную документацию API для **RxNodeDefinitionGetClipTriangle()** для получения более подробной информации.

ClipLine.csl

ClipLine.csl обрезает линии до усеченной пирамиды текущей камеры. Любые новые вершины, сгенерированные во время обрезки, проецируются (так что и положения в пространстве камеры, и положения в пространстве экрана, и координаты текстуры являются правильными) и добавляются к концам**RxClCamSpace3DVertices**и **RxClScrSpace2DVertices**массивы вершин кластеров и**RxClMeshState** кластера **ЧислоВершин**Член обновлен. Новые строки добавлены в **RxClИндексы**кластер и **RxClMeshState**кластера**КоличествоЭлементов** участник обновлен.

Узел имеет два выхода. Пакеты, в которых все линии обрезаны, отправляются на второй выход. Требования к входу этого узла:

RxClCamSpace3DVertices	-	**rxCLREQ_REQUIRED**
RxClScrSpace2DVertices	-	**rxCLREQ_REQUIRED**
RxClИндексы	-	**rxCLREQ_REQUIRED**
RxClMeshState	-	**rxCLREQ_REQUIRED**
RxClRenderState	-	**rxCLREQ_OPTIONAL**

Характеристики первого из выходов этого узла:

RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_VALID**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_NOCHANGE**

Характеристики второго выхода этого узла:

RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_INVALID**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_NOCHANGE**

См. справочную документацию API для **RxNodeDefinitionGetClipLine()**для получения более подробной информации.

ОтправитьTriangle.csl

SubmitTriangle.csl отправляет 2D-треугольники в API растеризации. Узел имеет один выход, и пакеты проходят через него без изменений. Цель этого — разрешить изменение пакетов и повторную отправку позже в конвейере для выполнения многопроходного рендеринга.

Поведение SubmitTriangle.csl относительно состояния рендеринга следующее: он устанавливает растр текстуры, режим фильтра текстуры, режимы адресации текстуры и флаг альфа вершины, все из входящего**RxClRenderStateVector**кластер. Все остальные состояния рендеринга сохраняются как есть. Эти состояния настроены на сохранение поведения состояния рендеринга таким же, каким оно было в конвейерах до PowerPipe.

Обратите внимание, что в то время как общие узлы отправки отправляют 2D-примитивы в**РвИм2Д** API растеризации, платформенно-зависимые узлы могут использовать преимущества HW T&L и отправлять 3D-примитивы непосредственно на оборудование. В этом случае отбраковка, трансформация, обрезка и освещение

будут выполняться аппаратным обеспечением и будут исключены из конвейера.

Узел имеет один выход, через который пакеты проходят без изменений. Входные требования этого узла:

RxClScrSpace2DVertices	-	**rxCLREQ_REQUIRED**
RxClИндексы	-	**rxCLREQ_OPTIONAL**
RxClMeshState	-	**rxCLREQ_REQUIRED**
RxClRenderState	-	**rxCLREQ_OPTIONAL**

Характеристики первого выхода этого узла:

RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_NOCHANGE**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_NOCHANGE**

См. справочную документацию API для **RxNodeDefinitionGetSubmitTriangle()** для получения более подробной информации.

SubmitLine.csl

SubmitLine.csl отправляет 2D-линии в API растеризации. Узел имеет один выход, и пакеты проходят через него без изменений. Цель этого — разрешить изменение пакетов и повторную отправку позже в конвейере для выполнения многопроходного рендеринга.

Поведение SubmitLine.csl относительно состояния рендеринга следующее: он устанавливает растр текстуры, режим фильтра текстуры, режимы адресации текстуры и флаг альфа вершины, все из входящего**RxClRenderStateVector**кластер. Все остальные состояния рендеринга сохраняются как есть. Эти состояния настроены на сохранение поведения состояния рендеринга таким же, каким оно было в конвейерах до PowerPipe.

Обратите внимание, что в то время как общие узлы отправки отправляют 2D-примитивы в**РвИм2Д** API растеризации, платформенно-зависимые узлы могут использовать преимущества HW T&L и отправлять 3D-примитивы непосредственно на оборудование. В этом случае отбраковка, трансформация, обрезка и освещение будут выполняться оборудованием и будут исключены из конвейера.

Узел имеет один выход, через который пакеты проходят без изменений. Входные требования этого узла:

RxClScrSpace2DVertices	-	**rxCLREQ_REQUIRED**
RxClИндексы	-	**rxCLREQ_OPTIONAL**
RxClMeshState	-	**rxCLREQ_REQUIRED**
RxClRenderState	-	**rxCLREQ_OPTIONAL**

Характеристики первого выхода этого узла:

RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_NOCHANGE**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_NOCHANGE**

См. справочную документацию API для **RxNodeDefinitionGetSubmitLine()**для получения более подробной информации.

AtomicInstance.csl

AtomicInstance.csl создает один пакет на**RpMesh**в источнике **RpGeometry**. Он помещает геометрические данные в**RxClObjSpace3DVertices** и**RxClИндексы**кластеризует и инициализирует**RxClMeshState**и **RxClRenderState**кластеры с соответствующими значениями. Обратите внимание, что созданные индексы всегда будут такими же, как для примитива tri-list. Преобразование будет выполнено, если источник**RpGeometry** использует другой**RwPrimitiveType**.

Узел имеет один выход, через который проходит инстанцируемая геометрия. Входные требования этого узла:

RxClObjSpace3DVertices	-	**rxCLREQ_DONTWANT**
RxClИндексы	-	**rxCLREQ_DONTWANT**
RxClMeshState	-	**rxCLREQ_DONTWANT**
RxClRenderState	-	**rxCLREQ_DONTWANT**

Характеристики первого выхода этого узла:

RxClObjSpace3DVertices	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_VALID**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_VALID**

См. справочную документацию API для **RxNodeDefinitionGetAtomicInstance()** для получения более подробной информации.

WorldSectorInstance.csl

WorldSectorInstance.csl создает один пакет на**RpMesh**в источнике **RpWorldSector**. Он помещает геометрические данные в **RxClObjSpace3DVertices**и**RxClИндексы**кластеризует и инициализирует **RxClMeshState**и**RxClRenderState**кластеры с соответствующими значениями. Обратите внимание, что созданные индексы всегда будут такими же, как для примитива tri-list. Преобразование будет выполнено, если источник**RpWorldSector** использует другой **RwPrimitiveType**.

Узел имеет один выход, через который проходит инстанцируемая геометрия. Входные требования этого узла:

RxClObjSpace3DVertices	-	**rxCLREQ_DONTWANT**
RxClИндексы	-	**rxCLREQ_DONTWANT**
RxClMeshState	-	**rxCLREQ_DONTWANT**
RxClRenderState	-	**rxCLREQ_DONTWANT**

Характеристики первого выхода этого узла:

RxClObjSpace3DVertices	-	**rxCLVALID_VALID**
RxClИндексы	-	**rxCLVALID_VALID**
RxClMeshState	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_VALID**

См. справочную документацию API для **RxNodeDefinitionGetWorldSectorInstance()**для получения более подробной информации.

AtomicEnumerateLights.csl

Целью AtomicEnumerateLights.csl является определение того, какие источники света в мире освещают текущий**RpАтомный**и разместить указатели на каждый из этих огней в**RxLight**кластер (**RxLight**структура - это просто указатель на **RpLight**).

Узел имеет один выход, через который пакеты проходят со своим новым **RxLight**кластер. Входные требования этого узла:

RxClLights - **rxCLREQ_REQUIRED**

Характеристики первого выхода этого узла:

RxClLights - **rxCLVALID_VALID**

См. справочную документацию API для **RxNodeDefinitionGetAtomicEnumerateLights()**для получения более подробной информации.

WorldSectorEnumerateLights.csl

Цель WorldSectorEnumerateLights.csl — определить, какие источники света в мире освещают текущий**RpWorldSector**и разместить указатели на каждый из этих огней в**RxLight**кластер (**RxLight**структура - это просто указатель на **RpLight**).

Узел имеет один выход, через который пакеты проходят со своим новым **RxLight**кластер. Входные требования этого узла:

RxClLights - **rxCLREQ_REQUIRED**

Характеристики первого выхода этого узла:

RxClLights - **rxCLVALID_VALID**

См. справочную документацию API для **RxNodeDefinitionGetWorldSectorEnumerateLights()**для дальнейшего подробности.

МатериалScatter.csl

Целью узла MaterialScatter.csl является распределение пакетов по материальным конвейерам на основе указателя конвейера в их **RxClMeshState**кластер. Этот узел требует как**rxCLREQ_OPTIONAL**множество стандартных кластеров, так что если они присутствуют в конвейере, они будут распространяться от этого узла к конечному материальному конвейеру (в отличие от завершения до этого узла путем отслеживания зависимостей – см. выше) *Погоня за зависимостью*раздел). Для любых других кластеров, которые вы хотите распространить до конца текущего конвейера, а затем до материальных конвейеров, используйте**RxPipelineNodeRequestCluster()**во время строительства трубопровода для изменения требований узла MaterialScatter.csl.

Узел не имеет выходов; все пакеты переходят из него в другие конвейеры. Входные требования этого узла:

RxClMeshState	-	**rxCLREQ_REQUIRED**
RxClObjSpace3DVertices	-	**rxCLREQ_OPTIONAL**
RxClИндексы	-	**rxCLREQ_OPTIONAL**
RxClRenderState	-	**rxCLREQ_OPTIONAL**
RxClLights	-	**rxCLREQ_OPTIONAL**

См. справочную документацию API для **RxNodeDefinitionGetMaterialScatter()** для получения более подробной информации.

Scatter.csl

Узел Scatter.csl отправляет пакеты по определенным ветвям конвейера в зависимости от данных в каждом пакете (необязательно)**RxScatter** кластера или на личных данных узла (сам по себе**RxScatter**структура).

Узел имеет 32 выхода (максимально допустимое количество) для облегчения экстремального разветвления конвейера. Ни один из них не должен быть фактически подключен. Требования к входу этого узла:

RxClРассеивание	-	**rxCLREQ_OPTIONAL**

Характеристики всех выходов этого узла:

RxClРассеивание	-	**rxCLVALID_NOCHANGE**

См. справочную документацию API для**RxNodeDefinitionGetScatter()** для получения более подробной информации.

PreLight.csl

Prelight.csl инициализирует значения цвета в**RxClCamSpace3DVertices** кластер перед зажиганием. Он может использовать дополнительный**RxClVSteps**кластер (создан, например, узлом отбраковки задней грани) для ускорения этого процесса путем пропуска неиспользуемых вершин.

Узел имеет один выход, через который проходят предварительно освещенные вершины. Входные требования этого узла:

RxClMeshState	-	**rxCLREQ_REQUIRED**
RxClObjSpace3DVertices	-	**rxCLREQ_REQUIRED**
RxClCamSpace3DVertices	-	**rxCLREQ_REQUIRED**
RxClVSteps	-	**rxCLREQ_OPTIONAL**

Характеристики первого выхода этого узла:

RxClMeshState	-	**rxCLVALID_VALID**
RxClObjSpace3DVertices	-	**rxCLVALID_VALID**
RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClVSteps	-	**rxCLVALID_NOCHANGE**

См. справочную документацию API для **RxNodeDefinitionGetPreLight()**для получения более подробной информации.

Свет.csl

Для каждого света в**RxClLights**кластер, узел Light.csl накапливает свет (используя соответствующую функцию освещения - окружающий, точечный и т.д.) в цветах вершин**RxClCamSpace3DVertices**кластер. Он может использовать необязательный **RxClVшаги**кластер (создан, например, узлом отбраковки задней грани) для ускорения этого процесса путем пропуска неиспользуемых вершин.

Узел имеет один выход, через который проходят освещенные вершины. Входные требования этого узла:

RxClMeshState	-	**rxCLREQ_REQUIRED**
RxClObjSpace3DVertices	-	**rxCLREQ_REQUIRED**
RxClCamSpace3DVertices	-	**rxCLREQ_REQUIRED**
RxClLights	-	**rxCLREQ_OPTIONAL**
RxClVSteps	-	**rxCLREQ_OPTIONAL**

Характеристики первого выхода этого узла:

RxClMeshState	-	**rxCLVALID_NOCHANGE**
RxClObjSpace3DVertices	-	**rxCLVALID_NOCHANGE**
RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClLights	-	**rxCLVALID_NOCHANGE**
RxClVSteps	-	**rxCLVALID_NOCHANGE**

См. справочную документацию API для**RxNodeDefinitionGetLight()**для получения более подробной информации.

PostLight.csl

Узел PostLight.csl фиксирует значения цвета в **RxClCamSpace3DVertices** кластеризуем в диапазон [0,255] и затем копируем эти значения в **RxClScrSpace2DVertices**кластер. Если материал
цвет геометрии не {255, 255, 255, 255}, то значение освещения для каждой вершины умножается на цвет материала (нормализованный на 1/255) перед выполнением зажима и копирования. Узел может использовать необязательный **RxClVшаги**кластер (создан, например, узлом отбраковки задней грани) для ускорения этого процесса путем пропуска неиспользуемых вершин.

Узел имеет один выход, через который проходят вершины post-lite.
Входные требования этого узла:

RxClMeshState	-	**rxCLREQ_REQUIRED**
RxClCamSpace3DVertices	-	**rxCLREQ_REQUIRED**
RxClScrSpace2DVertices	-	**rxCLREQ_REQUIRED**
RxClVSteps	-	**rxCLREQ_OPTIONAL**

Характеристики первого выхода этого узла:

RxClMeshState	-	**rxCLVALID_NOCHANGE**
RxClCamSpace3DVertices	-	**rxCLVALID_VALID**
RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClVSteps	-	**rxCLVALID_NOCHANGE**

См. справочную документацию API для **RxNodeDefinitionGetPostLight()**для получения более подробной информации.

FastPathSplitter.csl

Узел FastPathSplitter.csl предназначен для использования с**RpWorldSector**s. Он определяет, находится ли ограничивающий прямоугольник сектора, из которого был создан текущий пакет, полностью в пределах пирамиды видимости текущей камеры. Если это так, он отправляет пакет на второй выход, который должен пропустить узел отсечения, используемый в текущем конвейере.

Узел имеет два выхода. Пакеты отправляются через второй выход, если все вершины лежат в пределах усеченной пирамиды видимости и, следовательно, этап отсечения конвейера можно пропустить. Требования к входным данным этого узла:

RxClMeshState - **rxCLREQ_REQUIRED**

Характеристики первого выхода этого узла:

RxClMeshState - **rxCLVALID_NOCHANGE**

Характеристики второго выхода этого узла:

RxClMeshState - **rxCLVALID_NOCHANGE**

См. справочную документацию API для **RxNodeDefinitionGetFastPathSplitter()**для получения более подробной информации.

UVInterp.csl

UVInterp.csl обновляет**RxClScrSpace2DVertices**кластер с новым набором правильно обрезанных текстурных координат. Он использует необязательный **RxClИнтерполянты**кластер (сгенерированный ClipTriangle.csl) для интерполяции координат текстуры для обрезанных треугольников. Частные данные этого узла могут использоваться для его включения и выключения во время выполнения и для изменения состояния рендеринга.

Узел имеет два выхода. Пакеты отправляются без изменений на второй выход, если булево значение**uvInterpOn**в личных данных узла установлено значение**ЛОЖЬ**.

Входные требования этого узла:

RxClScrSpace2DVertices - **rxCLREQ_REQUIRED**
RxClRenderState - **rxCLREQ_REQUIRED**
RxClИнтерполянты - **rxCLREQ_OPTIONAL**
RxClUV - **rxCLREQ_REQUIRED**

Характеристики первого выхода этого узла:

RxClScrSpace2DVertices - **rxCLVALID_VALID**
RxClRenderState - **rxCLVALID_VALID**
RxClИнтерполянты - **rxCLVALID_NOCHANGE**
RxClUV - **rxCLVALID_VALID**

Характеристики второго выхода этого узла:

RxClScrSpace2DVertices	-	**rxCLVALID_NOCHANGE**
RxClRenderState	-	**rxCLVALID_NOCHANGE**
RxClИнтерполянты	-	**rxCLVALID_NOCHANGE**
RxClUV	-	**rxCLVALID_NOCHANGE**

См. справочную документацию API для **RxNodeDefinitionGetUVInterp()** для получения более подробной информации.

RGBAInterp.csl

RGBAInterp.csl обновляет**RxClScrSpace2DVertices**кластер с новым набором правильно обрезанных цветов. Он использует необязательный**RxClИнтерполянты** кластер (сгенерированный ClipTriangle.csl) для интерполяции координат текстуры для обрезанных треугольников. Частные данные этого узла могут использоваться для его включения и выключения во время выполнения и для изменения состояния рендеринга.

Узел имеет два выхода. Второй выход будет использоваться, если булево значение **rgbaИнтерпон**в личных данных узла установлено значение**ЛОЖЬ**, или если **RwRGBAReal**Кластер отсутствует или пуст. Входные требования этого узел:

RxClScrSpace2DVertices	-	**rxCLREQ_REQUIRED**
RxClRenderState	-	**rxCLREQ_DONTWANT**
RxClИнтерполянты	-	**rxCLREQ_OPTIONAL**
RxClRGBAs	-	**rxCLREQ_OPTIONAL**

Характеристики первого выхода этого узла:

RxClScrSpace2DVertices	-	**rxCLVALID_VALID**
RxClRenderState	-	**rxCLVALID_VALID**
RxClИнтерполянты	-	**rxCLVALID_NOCHANGE**
RxClRGBAs	-	**rxCLVALID_VALID**

Характеристики второго выхода этого узла:

RxClScrSpace2DVertices	-	**rxCLVALID_NOCHANGE**
RxClRenderState	-	**rxCLVALID_NOCHANGE**
RxClИнтерполянты	-	**rxCLVALID_NOCHANGE**
RxClRGBAs	-	**rxCLVALID_NOCHANGE**

См. справочную документацию API для **RxNodeDefinitionGetRGBAInterp()**для получения более подробной информации.

Клон.csl

Благодаря механизму выполнения вложенного конвейера (см.*Поток данных в трубопроводах*раздел предыдущей главы,*Обзор PowerPipe*), в данный момент времени может существовать только один пакет. Цель Clone.csl — создавать клоны каждого пакета, который в него входит, и отправлять их на различные выходы узла. Clone.csl — необычный узел, поскольку у него нет фиксированного **RxNodeDefinition**. Вместо этого пользователь должен создать**RxNodeDefinition** через функцию**RxNodeDefinitionCloneCreate()**, указывающий количество выходов и те, на которые должны быть отправлены клоны пакетов в одном или нескольких режимах работы (которые могут переключаться между выполнениями конвейера).

Clone.csl может быть полезен, когда конвейеру необходимо изменить исходные данные двумя или более различными способами, чтобы достичь желаемого эффекта рендеринга. Вместо того, чтобы генерировать исходные данные дважды, Clone.csl может генерировать несколько клонов из каждого пакета и отправлять каждый из них по другой ветке конвейера.

Обратите внимание, что если узел клонирования используется только для того, чтобы данные кластера можно было изменить, но восстановить в исходное состояние далее по конвейеру, то есть альтернатива, которая в большинстве случаев будет проще и эффективнее. Она заключается в создании вспомогательного кластера для хранения ссылки на данные исходного кластера и пометки исходного кластера как «внешнего». Это гарантирует, что любое изменение его данных вызовет копирование, так что исходные, неизмененные данные по-прежнему будут доступны через вспомогательный кластер. Копирование может быть более затратным, чем узел клонирования, хотя узел клонирования фактически не предотвратит копирование и не всегда особенно дешев в выполнении. Clone.csl попытается оптимизировать флаги кластеров в клонах пакетов, так что там, где кластеры не*нуждаться*помечены как «внешние» (что приводит к копированию данных кластера, если узел, расположенный ниже по конвейеру, изменяет их), они таковыми не являются.

Более подробную информацию см. в справочной документации API по следующим функциям:

- **RxNodeDefinitionCloneCreate()**
- **RxNodeDefinitionCloneDestroy()**
- **RxPipelineNodeCloneDefineModes()**
- **RxPipelineNodeCloneOptimize()**
- **RxPacketCacheCreate()**
- **RxPacketCacheClone()**
- **RxPacketCacheDestroy()**

Отладочные узлы

Для отладки выполнения конвейера часто бывает полезно вставить в конвейер узел «отладки», который отслеживает содержимое проходящих через него пакетов.**RxPipelineInsertDebugNode()**позволяет вам вставить любой узел по вашему выбору в существующий конвейер. В точке, где должен быть вставлен узел,**RxPipelineInsertDebugNode()**определяет, какие кластеры активны и содержат допустимые данные в исходном конвейере. Затем он изменяет определение узла таким образом, чтобы запрашивать только эти кластеры. Это гарантирует, что новый узел не вызовет никаких изменений в зависимостях кластера в конвейере.

См. справочную документацию API для**RxPipelineInsertDebugNode()** для получения более подробной информации.

31.5 Распространенные ловушки и подводные камни

31.5.1 Проблемы строительства трубопровода

Когда пользователи начинают создавать конвейеры и пользовательские узлы, они часто обнаруживают, что**RxPipelineUnlock()**не удается, что означает, что конвейер в некотором роде недействителен. Вот несколько вещей, которые нужно проверить, когда это происходит:

Прежде всего убедитесь, что вы используете отладочную версию библиотек RenderWare Graphics, и проверьте поток отладки.**RxPipelineUnlock()**выведет сообщения об ошибках, которые должны помочь вам определить, где в вашем конвейере находится проблема. Возможно, он обнаружил тривиальную ошибку в конвейере (например, он не содержит узлов!) или в узле конвейера (например, он не содержит метода body, слишком много интересующих кластеров или слишком много выходов — обратите внимание, что узел может иметь действительно нулевые выходы).

Стоит проверить, что длина массива в вашем**RxNodeDefinition** имеют смысл. Частой ошибкой является добавление дополнительного интересующего кластера к узлу без расширения входных и выходных спецификаций массивов.

Ошибки могли возникнуть во время поиска зависимостей из-за ошибки в топологии конвейера – он имеет более одной точки входа или содержит циклы или несвязанные подграфы. Если это происходит, проверьте использование функций, таких как**RxPipelineAddFragment()**и**RxLockedPipeAddPath()** при строительстве трубопровода.

Наиболее распространенной проблемой при построении конвейера является сбой отслеживания зависимости из-за невыполненных входных требований узла. В этом случае узел и кластер, о которых идет речь, должны быть упомянуты в потоке отладки. На основе этой информации вам нужно будет выяснить, почему узел не получает того, что ожидает от кластера, о котором идет речь.

Одна из возможностей заключается в том, что узел требует, чтобы кластер содержал действительные данные при входе в узел, но при этом не было другого предшествующего узла (на*любой*путь выполнения, помните) инициализировал кластер. Возможно, предыдущий узел уничтожил кластер (отметив его как**rxCLVALID_INVALID**для одного выхода).

Небольшая возможность заключается в том, что только один узел в конвейере использует данный кластер. В этом случае кластер не будет создан для использования в конвейере, *пока не*the **силаНастоящая**член**кластерыИнтерес**Массив спецификации входных данных узла установлен в значение**rxCLFORCEPRESENT**(в отличие от обычного**rxCLALLOWABSENT**) для этого кластера.

Необычно, когда узел требует присутствия кластера, если только данные, созданные для этого кластера, не должны передаваться последующему узлу в конвейере, хотя это может быть допустимым запросом. Если узел хочет выделить некоторую память в качестве временного рабочего пространства для использования во время обработки данных, то он может сделать это, не помещая данные в кластер. Однако, если узел хочет разрешить последующим узлам использовать созданные им данные, если они могут их использовать, то он*воля*поместить данные в кластер. В этом случае (учитывая, что нельзя обоснованно ожидать, что он будет заглядывать вперед в конвейер во время выполнения, чтобы определить, могут ли какие-либо последующие узлы использовать рассматриваемый кластер), узел потребует присутствия кластера независимо от того, используются ли его данные последующими узлами или нет.

Связанный случай — это конечные узлы конвейера (узлы в конце ветви конвейера), которые отправляют пакеты в другие конвейеры. Примером такого узла является узел MaterialScatter.csl. Этот узел не выполняет никакой обработки данных; он просто отправляет пакеты в соответствующий материальный конвейер для текущего**RpMesh**. Однако для того, чтобы гарантировать, что данные кластера, для кластеров, требуемых принимающим конвейером, сохраняются достаточно долго, чтобы достичь этого узла, он должен запросить эти кластеры, чтобы они присутствовали в его входных требованиях. Все они запрашиваются как**rxCLVALID_OPTIONAL** на случай, если они действительно отсутствуют по какой-то уважительной причине.

В случае, если созданный пользователем кластер (для аргумента значений «температуры» по вершинам) необходимо распределить по пользовательским материальным конвейерам с помощью MaterialScatter.csl, кластер можно добавить к входным требованиям узла с помощью функции**RxPipelineNodeRequestCluster()**. Это необходимо сделать во время строительства трубопровода, до**RxPipelineUnlock()** называется. Смотрите справочную документацию API для **RxPipelineNodeRequestCluster()**для получения более подробной информации.

Со временем этот раздел может быть дополнен, если будут выявлены дополнительные типичные проблемы при строительстве трубопроводов.

31.5.2 Производительность конвейера

При попытке улучшить производительность конвейера следует помнить две вещи:

1. Когда данные кластера помечены как «внешние», любые изменения данных приведут к их полному копированию. Это, скорее всего, будет затратным.

2. Произвольный доступ к данным кластера (с использованием **RxClusterGetIndexedData()**) будет медленнее, чем последовательный доступ (с использованием**RxClusterGetCursorData()**и**RxClusterIncCursor()**).

31.5.3 RxCluster->numUsed

Ответственность за обеспечение того, чтобы**numИспользовано**член **RxCluster**поддерживается в актуальном состоянии – это*важный*, так как невыполнение этого требования может привести к повреждению данных на последующих этапах передачи.

Обратите внимание, что если**numИспользовано**сводится к нулю для кластера, узел, который в данный момент его обрабатывает, имеет два варианта:

1. Уничтожить данные кластера с помощью**RxClusterDestroyData()**, чтобы обозначить, что кластер не содержит действительных данных и поэтому является недействительным/мертвым,

2. Оставьте кластер как есть, чтобы обозначить, что кластер по-прежнему действителен, даже если он не содержит используемых элементов.

Разницу между этими двумя состояниями кластера можно легко проверить в последующих узлах, чтобы определить, является ли кластер все еще допустимым (но не содержит используемых элементов) или недопустимым (не имеет выделенного массива данных). Однако обычно будет проще и удобнее, если предикация состояния кластера может быть выполнена неявно ветвью конвейера. Например, узел, который может очистить массив кластера, может быть указан с двумя выходами, для одного из которых кластер выходит в недопустимом состоянии, а для другого — в допустимом состоянии.

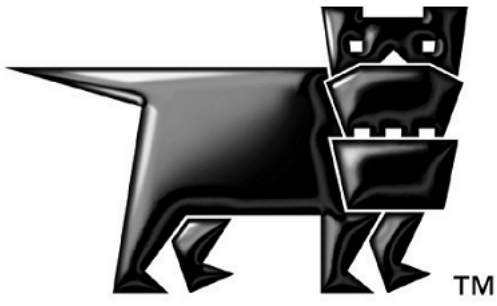
31.6 Резюме

В этой главе представлен обзор построения пользовательских узлов PowerPipe. В ней рассмотрены элементы, задействованные в создании определения узла: спецификация входов и выходов узла, методы узла и другие значения. В ней подробно описаны требования к кластеру узлов и процесс поиска зависимостей. В ней описано использование различных методов узлов. Для метода тела узла в ней представлен пример кода и обсужден порядок обработки пакетов и выполнения конвейера, а также подробно рассмотрены создание, доступ и изменение кластера. Наконец, в ней представлены кластеры, используемые универсальными узлами, поставляемыми с RenderWare Graphics, а затем сами узлы.

Как уже упоминалось ранее, создание и использование оптимизированных, платформенно-зависимых узлов и конвейеров рендеринга будут рассмотрены в последующих главах (*PS2All Обзор*, например, уже доступен).

Приложение

Рекомендовано Чтение



Введение

В этом приложении содержится список книг, журналов и интернет-ресурсов, рекомендованных командой разработчиков RenderWare Graphics.

Традиционные обзоры этих книг не были предприняты, поскольку их можно найти в Интернете – особенно в онлайн-магазинах. Вместо этого был дан приблизительный «уровень читателя», который делится следующим образом:

- **Начинающий**
  Не предполагает никаких предварительных знаний в области программирования 3D-графики;
- **Средний**
  Предполагается наличие некоторого опыта программирования 3D-графики;
- **Передовой**
  Только для гуру!

Некоторые книги предназначены специально для студентов университетов, и для их изучения требуется свободное владение математикой.

Важно отметить, что *все* перечисленные книги требуют некоторого понимания компьютерного программирования. Также можно с уверенностью сказать, что если вы никогда не программировали приложение для 2D-графики, то 3D-графика, скорее всего, покажется вам нелегкой задачей.

Если вы никогда не работали с рендерингом 2D-графики, вам доступен ряд онлайн- и печатных ресурсов. Это область компьютерной графики, которая очень хорошо обеспечена литературой.

Другая документация

Дополнительную информацию о RenderWare Graphics можно найти по адресу:

- Справочник по графическому API RenderWare для вашей целевой платформы
- Графика RenderWare**PDF**c
- Ваша учетная запись клиента в полностью управляемой системе поддержки RenderWare Graphics https://support.renderware.com и его поисковая база знаний

-

Для тех, кто совсем недавно начал заниматься компьютерной графикой, а также для менеджеров и продюсеров, которые просто хотят узнать, что это такое, "**Как работает компьютерная графика**", Олина Латропа (Wiley Computer Publishing. ISBN: 0471130400) рекомендуется в качестве хорошего учебника. Эта книга не предполагает понимания компьютерного программирования или математики.

Книги

Учебники

«Компьютерная графика: принципы и практика» (2йИздание, на языке C)

Авторы: Фоли, Ван Дам и др.

Издатель: Addison Wesley Longman Publishing Co.

ISBN: 0201848406

Уровень читателя: Новичок.

Примечания:*Это считается стандартным учебником по предмету. Предназначен для студентов университетов.*

«Продвинутые методы анимации и рендеринга: теория и практика»

Авторы: Алан Уотт, Марк Уотт

Издатель: Addison Wesley Longman Publishing Co.

ISBN: 0201544121

Уровень читателя: средний.

Примечания:*"«Watt & Watt» охватывает множество алгоритмов и методов, используемых в этой области, и считается обязательным справочником для большинства программистов 3D-графики. Рекомендуемое фоновое чтение по сплайнам и патчам.*

«Рендеринг в реальном времени» (2йВерсия)

Авторы: Томас Мёллер, Эрик Хейнс.

Издатель: AK Peters Ltd.

ISBN: 1568811829

Уровень читателя: средний.

Примечания:*Рассматривает множество алгоритмов, используемых в этой области, и приводит плюсы и минусы большинства из них.*

«3D-игры: Том 1: Рендеринг в реальном времени и программные технологии»

Авторы: Алан Уотт, Фабио Поликарпо

Издатель: Addison-Wesley Pub Co.

ISBN: 0201619210

Уровень читателя: средний/продвинутый.

Справочники

«Проектирование 3D-игрового движка»

Автор: Дэйв Эберли

Издатель: Морган Кауфманн

ISBN: 1558605932

Уровень читателя: средний/продвинутый.

Примечания:*Высоко оцененная книга по теме. Тяжелая теория – особенно математика. Нейтральная к API/платформе. Вероятно, одна из самых полных и хорошо написанных по этой теме.*

«Жемчужины игрового программирования»

Редактор: Марк ДеЛура

Издатель: Charles River Media

ISBN: 1584500492

Уровень читателя: средний/продвинутый.

Примечания:*Похожая по концепции на основополагающие книги Graphics Gems (см. ниже), эта книга содержит широкий спектр алгоритмов, советов, приемов и методов, охватывающих большинство аспектов программирования, дизайна и разработки компьютерных игр.*

«Жемчужины игрового программирования 2»

Редактор: Марк ДеЛура

Издатель: Charles River Media

ISBN: 1584500549

Уровень читателя: средний/продвинутый.

Примечания:*Второй том, с более чем 70 совершенно новыми статьями, написанными более чем 40 экспертами по программированию. Он имеет шесть всеобъемлющих разделов, включая новый раздел звукового программирования*.

«Жемчужины игрового программирования 3»

Редактор: Данте Трелья

Издатель: Charles River Media

ISBN: 1584500549

Уровень читателя: средний/продвинутый.

«Графические жемчужины» (тома I - V)

Авторы: (Разные)

Издатель: Academic Press

ISBN: 0122861663

Уровень читателя: средний/продвинутый.

Примечания:*Это очень популярная серия книг, каждая из которых содержит множество алгоритмов, профессиональных приемов и других крупиц полезной информации.*

«Компьютерная графика и виртуальные среды: от реализма к реальному времени»

Авторы: Мэл Солтер, Энтони Стид, Йоргос Хрисанту

Издатель: Addison-Wesley Pub Co.

ISBN: 0201624206

Уровень читателя: средний/продвинутый.

Книги (API/зависит от платформы)

OpenGL

«Руководство по программированию OpenGL»

Авторы: Мейсон Ву, Джеки Нейдер, Том Дэвис, Open Architecture Review Board

Издатель: Addison Wesley Longman Publishing Co.

ISBN: 0201604582

Уровень читателя: начальный/средний.

Примечания:*Эту книгу также называют «Красной книгой». Она считается исчерпывающим руководством по программированию OpenGL и трехмерной графики.*

«Справочное руководство OpenGL» (третье издание)

Автор: OpenGL Architecture Review Board (редактор: Дэйв Шрайнер).

Издатель: Addison Wesley Longman Publishing Co.

ISBN: 0201657651

Уровень читателя: начальный/средний.

Примечания:*Официальное справочное руководство по OpenGL.*

Microsoft DirectX/Direct3D

«Продвинутое программирование 3D-игр с DirectX 8.0»

Автор: Питер Уолш, Адриан Перес

Издатель: Wordware Publishing

ISBN: 155622513X

Уровень читателя: средний.

Примечания:*Предполагает некоторый опыт программирования игр (и знание C++), но имеет хорошее покрытие Direct3D. Полный обзор можно найти на GameDev.Net.*Видетьwww.gamedev.net

«Внутри DirectX»

Автор: Брэдли Барген, Теренс Питер Доннелли

Издатель: Microsoft Press

ISBN: 1572316969

Уровень читателя: Новичок.

Примечания:*Охватывает основы программирования DirectX. Эта книга не**нет** охватывает Direct3D и 3D-графику, но рекомендуется разработчикам, не имеющим опыта работы с DirectX.*

«Приемы и методы рендеринга в реальном времени в DirectX»

Автор: Демпски

Издатель: Премьер Пресс

ISBN: 1931841276

Уровень читателя: средний.

Примечания:*Содержит объяснения того, как реализовать часто запрашиваемые функции с использованием API DirectX 8, этот текст должен быть интересен как графическим дизайнерам, так и программистам игр. Отличная книга для тех, кто ищет справочное руководство по вершинным и пиксельным шейдерам.*

«Программирование игр со спецэффектами с помощью DirectX 8.0»

Автор: Маккаски

Издатель: Премьер Пресс

ISBN: 1931841063

Уровень читателя: средний.

Примечания:*"Эта книга научит читателей всему, что им нужно знать о семнадцати потрясающих эффектах для программирования игр; включая динамически генерируемые ландшафты, туман, размытость движения и отображение окружения. Подробные объяснения каждого трюка, а также легко разбираемые примеры кода позволяют читателям превратить свои игры из повседневного упадка в передовую усладу для глаз"*

«Программируемый графический конвейер Microsoft DirectX 9»

Автор: Корпорация Microsoft

Издатель: Microsoft Press

ISBN: 0735616531

Уровень читателя: средний/продвинутый.

Журналы

«Журнал графических инструментов» (AK Peters, Ltd.)

Ежеквартальный журнал, порожденный серией книг "Graphics Gems", указанной выше. С их веб-сайта:

"The*журнал графических инструментов*ежеквартальный журнал, основной задачей которого является предоставление сообществу исследователей, разработчиков и производителей компьютерной графики практических идей и методов, которые решают реальные проблемы».

Их веб-сайт находится по адресу:www.acm.org/jgt/

«Журнал разработчиков игр» (CMP Game Media Group)

Охватывает все аспекты разработки компьютерных игр. Для получения дополнительной информации см.www.gdmag.com .

Кроме того, Game Developer Magazine Article Companion представляет собой сборник интересных статей, опубликованных в электронном виде из журнала. Смотреть www.darwin3d.com/gamedev.htm

«Журнал доктора Добба» (Miller Freeman, Inc.)

Dr. Dobb's — один из самых старых журналов по общему программированию. Охватывает все аспекты ИТ, а не только компьютерную графику. Онлайн на www.ddj.com

Веб-сайты

«MSDN Онлайн»–Сайт Microsoft Developer Network.

msdn.microsoft.com

Сайт "OpenGL.Org"–Официальный сайт OpenGL.

www.opengl.org

"Гамасутра"–Онлайн-альтер-эго Game Developer Magazine. Вводный материал по патчам хорош, помимо прочего.

www.gamasutra.com

www.gamasutra.com/features/20000530/sharp_pfv.htm .

Флипкод, еще один сайт по разработке компьютерных игр с множеством ресурсов и статей:

www.flipcode.com

"GameDev.Сеть", веб-сайт по разработке компьютерных игр, содержащий множество ссылок и статей, а также размещающий список рассылки по 3D-алгоритмам:

www.gamedev.net

"**Двоичное разбиение пространства для ускоренного удаления скрытых поверхностей и рендеринга статических сред**", докторская диссертация, охватывает деревья BSP, уровень детализации, конвейер рендеринга, потенциально видимые наборы, порталы и другие темы.

www.acm.org/tog/editors/erich/bsp/aj.pdf

Японские сайты

Если вы интересуетесь общими компьютерными технологиями, это отличный сайт, где можно узнать общую информацию и методы, связанные с ИТ.

www.atmarkit.co.jp

Самая ценная информация — это сам игровой продукт и новости индустрии. Например,

www.zdnet.co.jp (ZD net Япония) иwww.famitsu.com

Эти сайты могут быть полезны всем разработчикам, читающим по-японски.

Документацию RenderWare Graphics на японском языке можно загрузить здесь:

www.criterion.co.jp - Японский сайт Criterion Software (на японском языке)

Группы новостей USENET

Иерархия comp.* содержит широкий спектр других новостных групп, связанных с такими областями, как ИИ, физическое моделирование и т. д. Это лишь небольшая выборка...

Группы общей компьютерной графики

Эти группы новостей предназначены для обсуждения различных аспектов программирования 3D-графики:

комп.графика.алгоритмы

комп.графика.анимация

комп.графика.разное

комп.графика.рендеринг.разное

Группы по разработке компьютерных игр

Эти группы предназначены для обсуждения дизайна и разработки компьютерных игр, а также самой индустрии:

comp.games.development.искусство

comp.games.development.audio

разработка.игр.комп.дизайна

индустрия.разработки.комп.игр

разработка.комп.игр.программирование

комп.игры.разработка.программирование.разное

Индекс

Индекс

Номера страниц, выделенные жирным шрифтом, указывают на наиболее важную ссылку на предмет, где существует несколько ссылок. Номера страниц, показанные ниже, относятся к Тому III Руководства пользователя.

Д

Э

Ф

Г

Я

Л

М

Н

О

П

Р

Т

У

В

Вт