ЭФФЕКТИВНАЯ ОРГАНИЗАЦИЯ ПРОЦЕССА РЕНДЕРИНГА НА ГРАФИЧЕСКОМ КОНВЕЙЕРЕ

Д. И. Мазовка, В. В. Краснопрошин

Белорусский государственный университет
Минск, Беларусь
e-mail: mazovka@bk.ru, krasnoproshin@bsu.by

Описана методология построения процессов визуализации с использованием технологии графического конвейера. Предложенный подход фокусируется на проблеме разработки программного обеспечения в 3D-графике и ставит целью оптимизацию соответствующих затрат.

Ключевые слова: графический конвейер; система визуализации.

EFFICIENT ORGANIZATION OF RENDERING PROCESS ON GRAPHICS PIPELINE

D. I. Mazouka, V. V. Krasnoproshin

Belarusian State University
Minsk, Belarus

This paper describes a methodology for rendering processes construction using graphics pipeline technology. The proposed technique is focused on the problem of program development in 3D graphics and aims to optimize related expenses.

Keywords: graphics pipeline; visualization system.

ВВЕДЕНИЕ

Компьютерная визуализация (англ. rendering) относится к компьютерной графике и является одним из интенсивно развивающихся направлений информатики.

Одной из основных проблем компьютерной графики является сложность, возникающая при разработке графических программных продуктов. Компании-разработчики тратят значительные объемы ресурсов, чтобы их продукты выглядели как можно лучше. Так, в области высокобюджетных игр затраты на их разработку достигают сотни миллионов долларов [1]. Хотя большая часть средств не относится к технологической стороне проектов, тем не менее затраты на разработку графической составляющей продукта порой достигают 10 % общего бюджета.

Для получения конкурентного преимущества и снижения стоимости разработки графических компонентов отдельные компании создают собственные специализированные библиотеки или целые системы визуализации, оптимизированные под конкретные проекты с заранее определенными ограничениями. Последние называются графическими движками. Такие движки призваны облегчить процесс написания про-

грамм, интенсивно использующих компьютерную графику. Однако и в этом случае приходится тратить большие средства на их постоянную поддержку и развитие.

Примерами движков являются программные библиотеки: Ogre, VTK, Unreal Engine, Unity, Frostbite, idTech и др. Список доступных движков достаточно большой [2] и постоянно расширяется, что свидетельствует о том, что проблема визуализации является актуальной и не имеющей на данный момент единого решения.

В данной работе задача визуализации рассматривается на самом базовом уровне и делается попытка выявить проблемы, которые усложняют решение. Это позволит в дальнейшем разработать метод эффективной организации процесса визуализации на графическом конвейере.

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Для начала определим понятие визуализации как некоторого процесса, переводящего некоторый набор данных (называемый сценой) в изображение:

$$Process(Scene) \rightarrow Image. \tag{1}$$

Сцена *Scene* обычно представляется в виде неупорядоченного множества объектов { $Object_1$, $Object_2$, ..., $Object_N$ }. Каждый из этих объектов в свою очередь включает в себя данные релевантные процессу визуализации: $Object_i$ = { $Data_{i1}$, $Data_{i2}$, ..., $Data_{iM(i)}$ }.

Задачей визуализации назовем проблему отрисовки какого-либо набора данных на компьютере. Формально определим ее следующим образом: пусть имеется некоторая сцена и требования к результирующему изображению, требуется организовать (построить) процесс визуализации, который переводит исходную сцену в изображение, удовлетворяющее поставленным условиям.

Рассмотрим, как задача визуализации решается на стандартном графическом конвейере.

Процесс визуализации на конвейере строится с помощью последовательности инструкций, которую можно обобщить в следующем виде:

$$\begin{aligned} &(Set_{11},...,Set_{1M(1)},Draw)_1,\\ &(Set_{21},...,Set_{2M(2)},Draw)_2,\\ &...\\ &(Set_{N1},...,Set_{NM(N)},Draw)_N, \end{aligned} \tag{2}$$

где Set_{ij} – инструкции, которые меняют определенные состояния конвейера, например, источники данных, размер и топологию геометрии. Количество вызовов *Set* может варьироваться в каждой группе. Инструкции *Draw* являются командами конвейера для выполнения процедуры рендеринга с использованием текущих параметров. Обычно одна группа инструкций соответствует отрисовке одного объекта.

Весь набор данных задачи визуализации может быть представлен в виде матрицы размера $N \times M+1$. Строки этой матрицы соответствуют группам инструкций отрисовки, а столбцы – параметрам рендеринга. К примеру, один столбец может быть отведен для текстур, а другой – для геометрии. Клетки матрицы не обязаны всегда содержать какое-либо значение, если какая-то группа отрисовки не использует определенный тип данных, то соответствующая клетка в матрице будет пустая:

$$Data = \begin{pmatrix} Data_{11} & Data_{12} & ... & Data_{1M} & 1 \\ Data_{21} & Data_{22} & ... & Data_{2M} & 1 \\ ... & ... & ... & ... & 1 \\ Data_{N1} & Data_{N2} & ... & Data_{NM} & 1 \end{pmatrix}. \quad (3)$$

Вектор инструкций *Render* является обобщенной группой инструкций, необходимой для отрисовки строки матрицы *Data*:

$$Render = \begin{pmatrix} Set_1 \\ ... \\ Set_2 \\ Draw \end{pmatrix}. \quad (4)$$

Матрица *Data* и вектор *Render* вместе задают полный процесс визуализации:

$$Result = Data \times Render = \begin{pmatrix} Set_1(Data_{11}) & ... & Set_M(Data_{1M}) & Draw \\ Set_1(Data_{21}) & ... & Set_M(Data_{2M}) & Draw \\ ... & ... & ... & Draw \\ Set_1(Data_{N1}) & ... & Set_M(Data_{NM}) & Draw \end{pmatrix}. \quad (5)$$

Матрица *Result* может быть выполнена на графическом конвейере путем обхода инструкций по строкам слева-направо и сверху-вниз.

Далее будем использовать матрицу *Data* для оценки сложности соответствующего процесса визуализации. Простейший способ оценки сложности состоит в определении размера матрицы. Чем большей была исходная сцена, тем больше в ней объектов и тем больше строк в матрице *Data*. Чем сложней алгоритм рендеринга сцены, тем больше столбцов.

Количество строк в *Data* (в практических задачах) измеряется в тысячах, и это число будет продолжать расти с развитием технологий [3]. Количество столбцов в то же время может быть десятками или сотнями, это число можно оценить, например, по количеству существующих состояний конвейера [4].

Теперь вернемся к самому процессу визуализации. Используя графический конвейер, формула визуализации может быть расширена следующим образом:

$$Scene \xrightarrow{Process} Data \times Render = Result \xrightarrow{Pipeline} Image. \quad (6)$$

Таким образом, для того чтобы решить задачу визуализации, нужно построить алгоритм (*Process*), который конструирует матрицу *Data*, и вектор *Render* для исходной сцены *Scene*. Общая схема такого алгоритма содержит следующие шаги:

1. Определить все возможные типы данных объектов сцены *Scene* (соответствуют столбцам матрицы *Data*).
2. Построить вектор *Render* для этих типов (*M* инструкций).
3. Для каждого объекта сцены (*N*):
   1) определить позицию (строку) объекта в матрице *Data* исходя из требований к результирующему изображению;
   2) скопировать данные объекта в матрицу.

В итоге сложность построения алгоритма может быть оценена как *O(N) + O(M)*.

Очевидно, что для сложных задач визуализации построение процесса с помощью представленной схемы выходит за рамки практической осуществимости. Более того, построение самой сцены является итерационным процессом, который требует постоянных коррекций исходного набора данных. Это соответственно приводит к множеству изменений в процессе визуализации и большим трудозатратам.

Далее в статье приводится метод, позволяющий снизить затраты на разработку процесса визуализации, который также является достаточно гибким, чтобы позволить изменения в исходных данных без значительного негативного эффекта.

АНАЛИЗ

Для того чтобы понизить сложность алгоритма, попробуем реструктуризировать матрицу *Data* с помощью отдельных подпроцессов рендеринга, которые будут применяться к отдельным подмножествам данных. Другими словами, будем строить процесс визуализации в следующем виде:

$$Result = Data \times Render = \begin{pmatrix} Data_1 \times Render_1 \\ Data_2 \times Render_2 \\ \dots \\ Data_s \times Render_s \end{pmatrix}, \quad (7)$$

где $Data_i$ – подмножества *Data*, а $Render_i$ – соответствующие подпроцессы. Подмножество данных матрицы *Data* – это матрица, состоящая из строк *Data*, выбранных с помощью некоторого алгоритма. Подпроцесс $Render_i$ – это вектор инструкций, состоящий из некоторых или всех инструкций основного вектора *Render*. Количество подмножеств *s* может быть любым числом в промежутке от 1 до *N*.

Предположим, матрица *Data* содержит частично упорядоченные кластеры объектов, включая неупорядоченные кластеры (объекты могут иметь один и тот же порядок в простейшем случае). К примеру, неупорядоченный кластер может содержать непрозрачные объекты, а частично упорядоченный кластер – прозрачные. Порядок объектов в кластере определяет последовательность, в которой эти объекты должны быть отрисованы. Если порядок допускает множество различных последовательностей, то любая из них может быть использована для визуализации кластера.

Для каждого кластера $Data_i$ определим функцию, которая на входе принимает произвольное множество объектов, а на выходе выдает упорядоченное подмножество объектов для соответствующего кластера:

$$Sample_i(Objects) = \langle Object, Object \in Objects \wedge Object \in Data_i \rangle \cdot \quad (8)$$

Функция *Sample* задается с помощью предиката, который определяет, принадлежит ли данный объект кластеру $Data_i$, и дополнительно с помощью отношения частичного порядка.

Имея в наличии функции *Sample,* определенные для всех кластеров в матрице *Data*, задача построения подмножеств по исходной сцене становится тривиальной:

$$Sample_i(Scene) = Data_i \quad (9)$$

для *i* от 1 до *s*.

В результате формула процесса визуализации может быть расширена:

$$
\begin{aligned}
&Scene \overset{Process}{\rightarrow} Sample_1(Scene) \times Render_1 + \ldots \\
&+ Sample_s(Scene) \times Render_s = \\
&Data_1 \times Render_1 + \ldots + Data_s \times Render_s = \\
&Data \times Render = Result \overset{Pipeline}{\rightarrow} Image \quad .
\end{aligned}
\tag{10}
$$

В этом представлении схема построения алгоритма *Process* следующая:

1. Определить все возможные кластеры, с помощью которых можно разбить сцену *Scene*.
2. Для каждого из этих кластеров (*s*):
   1) определить функцию *Sample*;
   2) построить вектор *Render*.

В этом случае задача визуализации сводится к определению кластеров, количество которых может быть значительно меньшим по сравнению с количеством объектов во всей сцене. В дополнение к этому, если результирующая сцена претерпевает изменения, которые не требуют задания новых кластеров, то процесс не нуждается ни в каких изменениях. Таким образом, предложенный подход является более гибким по сравнению с тривиальным.

ВЫВОДЫ

В работе рассмотрена проблема эффективной организации процесса визуализации с использованием стандартного конвейера. Предложен подход к оценке сложности процессов и сформулирован базовый метод их построения. Работа служит основой для дальнейшей формализации подхода, предложенного в предыдущих работах [5], [6]. В дальнейшем планируется рассмотреть взаимозависимость между кластерами и способы манипулирования промежуточными результатами рендеринга.

БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ

1. The Economist. Why video games are so expensive to develop, 24 September 2014.
2. Graphics engines database [Electronic resource]. URL: http://devmaster.net/devdb/engines
3. Forbes, DirectX 12 Delivers: AMD, Nvidia, And Intel Hardware Tested With Awesome Improvements, 26 March 2015.
4. Direct3D 11 Reference, Core Reference, Core Interfaces, ID3D11DeviceContext interface, Microsoft, MSDN.
5. Мазовка Д. И. Формальный подход к решению задачи визуализации // Международный конгресс по информатике: информационные системы и технологии (CSIST-2011) : материалы конф. Минск, 2011.
6. Krasnoproshin V., Mazouka D. Novel Approach to Dynamic Models Visualization // J. of Computational Optimization in Economics and Finance. 2013. Vol. 4, iss 2–3. P. 113–124.