Адаптивный метод рендеринга динамических трехмерных сцен

В.И. Гонахчян[1]
pusheax@ispras.ru
[1]Институт системного программирования им. В.П. Иванникова РАН, Москва, Россия

Рендеринг динамических трехмерных сцен представляет сложность из-за невозможности выполнения предобработки для объединения и упрощения полигональных моделей, сохранения информации о видимости. При рендеринге сцен с большим количеством подвижных объектов производительность зачастую падает из-за перезаписи командных буферов, выполнения лишних проверок видимости. Предлагается модель производительности, согласно которой вычисляется требуемый объем памяти и выполняется оценка времени выполнения основных этапов прямого рендеринга. Предлагается адаптивный метод рендеринга динамических сцен, который выбирает наиболее эффективный метод использования командных буферов и количество аппаратных проверок видимости в зависимости от состояния сцены. В отличие от существующих методов рендеринга динамических сцен, предложенный метод учитывает затраты на составление и посылку командных буферов, вычисляет оптимальное количество проверок видимости, используя модель производительности рендеринга. Результаты тестирования предложенного адаптивного метода показали его эффективность при рендеринге динамических сцен с большим количеством объектов.

Ключевые слова: *рендеринг, составление командных буферов, удаление невидимых поверхностей, модель производительности рендеринга.*

Adaptive Rendering of Dynamic 3D Scenes

V.I. Gonakhchyan[1]
pusheax@ispras.ru
[1]Ivannikov Institute for System Programming of the RAS, Moscow, Russia

Rendering of dynamic 3d scenes is challenging because it is impossible to perform preprocessing to merge and simplify polygonal models, to precalculate visibility information. The dynamic behavior of objects (visibility change, movement) is causing command buffers rebuilding and rejecting of invisible objects often does not result in performance gains. We propose an adaptive method for visualizing dynamic scenes, which selects the most efficient method for recording and using command buffers and the number of hardware occlusion queries. Proposed adaptive method is based on the performance model, which performs an estimation of the execution time of the main stages of forward rendering. Testing results of the proposed method showed its effectiveness when rendering large dynamic scenes.

Keywords: *rendering, command buffer recording, occlusion culling, performance model of rendering.*

1. Введение

Динамические сцены содержат большое количество объектов с меняющимися характеристиками: видимость, положение, материал. В некоторых приложения, таких как САПР, необходимо иметь возможность выделять и передвигать любые объекты в сцене. Это вызывает затраты, связанные с перестроением иерархии, перезаписыванием командных буферов. Для решения этой проблемы предлагается метод составления иерархии с пространственными индексами, которая также используется для сокращения затрат на буферизацию команд рендеринга. Для определения затрат на буферизацию и времени рендеринга предлагается модель производительности рендеринга.

Большие трехмерные сцены содержат невидимые объекты, которые, тем не менее, отправляются на графический процессор для рендеринга. Методы удаления невидимых поверхностей (occlusion culling) используются для отбраковки невидимых объектов. При рендеринге динамических сцен нельзя выполнить предобработку для определения видимости объектов и зачастую используются аппаратные проверки видимости (hardware occlusion queries). Однако в некоторых случаях выполнение аппаратных проверок видимости может понизить производительность рендеринга. Существующие методы, такие как иерархические проверки видимости, отправка одного запроса на группу объектов, плохо справляются с масштабными сценами. В данной работе, на основе предложенной модели производительности вычисляется оптимальное количество проверок видимости.

Во время рендеринга динамические сцены сильно изменяются. Обычно новая сцена содержит маленькое количество объектов, затем количество объектов постепенно увеличивается, назначаются новые положения, материалы, видимость. Для рендеринга различных состояний сцены требуются различные способы, учитывающие характеристики сцены в данный момент времени. В данной статье, предлагается адаптивный метод рендеринга, который подбирает наиболее эффективный метод буферизации объектов и количество запросов видимости для текущего состояния сцены.

Рассмотрим работы по оценке времени рендеринга. В работе [6] предложен адаптивный метод рендеринга с заданной частотой кадров. Для каждого кадра выбирается подходящий уровень детализации объектов для поддержки целевой частоты кадров. В [12] учитывается попадание вершин в кэш память после трансформации, предлагается суммировать времена работы различных этапов конвейера для получения консервативной оценки времени рендеринга. Существующие методы оценки времени рендеринга не учитывают работу по составлению и отправке буферов команд, которая может отнимать значительное время при большом количестве объектов. В данной работе предлагается использовать новые возможности современного графического интерфейса Vulkan для измерения времени рендеринга.

Существует множество методов удаления невидимых поверхностей [5]. В данной работе рассматривается рендеринг динамических сцен, поэтому наибольший интерес представляют методы удаления невидимых поверхностей без предобработки сцен [2,8]. В этих работах исследованы проверки видимости (occlusion query). Для

сокращения времени ожидания результатов запросов видимости предложили выполнять отправку запросов видимости для листьев иерархии, проверять результаты видимости в следующем кадре. Иерархические аппаратные проверки видимости подходят для рендеринга динамических сцен и используются в данной работе.

Некоторые методы выполняют генерацию буфера команд на графическом процессоре (indirect rendering) [4]. При этом не выполняется дорогостоящая передача команд на графический процессор. Однако метод рассчитан на пообъектное отсечение, выполнение иерархических проверок видимости представляется затруднительным.

Рассмотрим развитие программных интерфейсов для рендеринга трехмерных сцен. В OpenGL до версии 1.4 команды рендеринга посылались на графический процессор в режиме “immediate mode” (команды и вершины отправлялись на графический процессор каждый кадр). В OpenGL 1.5 были добавлены объекты “VBO”, которые позволили хранить вершины в памяти графического процессора [9]. Это помогло ускорить взаимодействие центрального и графического процессоров, но проблемы составления командных буферов и валидации состояния остались. Расширение OpenGL “NV_command_list” помогло ускорить составление и отправку буферов команд. Это повысило эффективность рендеринга сцен САПР с большим количеством объектов [10]. Недавно разработали новый графический интерфейс Vulkan, который позволил более эффективно работать с командными буферами и при необходимости отключать валидацию состояния. В программной реализации методов данной работы используется Vulkan.

Исследование производительности составления командных буферов в Vulkan было проделано в работах [3,11]. В отличие от существующих работ, в данной статье предлагается использовать вспомогательные буфера (secondary command buffer) для хранения команд рендеринга в узлах иерархии. Производительность составления и отправки командных буферов в Vulkan была исследована в работе [7].

2. Модель производительности

В этой главе предлагается модель производительности, позволяющая рассчитать время выполнения прямого рендеринга динамической сцены за один проход на центральном и графическом процессорах.

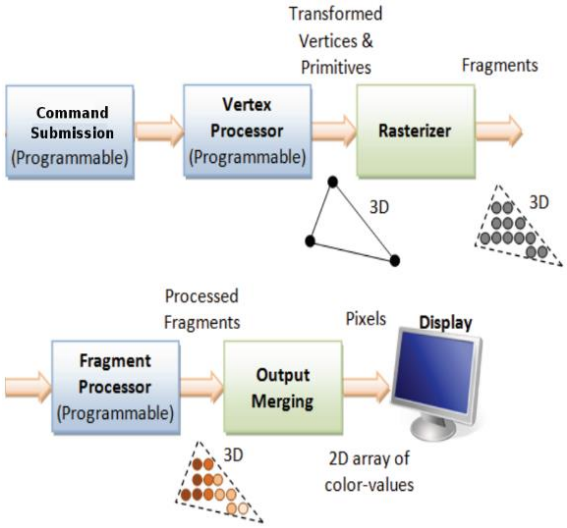

Рис. 1. Основные этапы рендеринга объектов сцены.

На рис. 1 изображены основные этапы рендеринга, которые включают обработку объекта на центральном процессоре и графическом процессоре. Любой из этих этапов может быть узким местом при выполнении рендеринга. Первый этап (Command Submission) происходит на центральном процессоре. Команда рендеринга содержит информацию о состоянии конвейера графического процессора и смещение в памяти, по которому хранятся вершины (индексы вершин) объекта. Команды посылаются по шине на графический процессор. Далее устанавливается новое состояние конвейера или продолжается использование старого состояния. При обработке вершин (Vertex Processor) выполняется трансформация вершин для перевода вершин в плоскость изображения. Далее происходит растеризация для определения пикселей, которые входят в треугольник. Во время следующего этапа (Fragment Processor) происходит вычисление цвета каждого пикселя треугольника с учетом источников освещения и текстур. Далее происходит объединение результатов и вывод на экран.

Зачастую на практике производительность падает именно на программируемых этапах рендеринга. Первый этап может быть лимитирующим при рендеринге сцен с большим количеством объектов. Второй этап — когда сцена содержит большое количество вершин. Четвертый этап — когда выводится большое количество пикселей на экран, либо вычисления в фрагментном шейдере достаточно дорогие.

Рендеринг происходит наиболее быстро, когда вершины и командные буфера хранятся в памяти графического процессора. Необходимо определить заранее, что все объекты сцены и ресурсы поместятся в память. Формула для оценки объема потребляемой памяти:

$$M_{scene} = N_{vert} * M_{vert} + N_{index} * M_{index} + N_{tc} * M_{tc} + N_{norm} * M_{norm} + \sum_i M_i^{tex} + \sum_i M_i^{buf} + \sum_i M_i^{desc}, \quad (1)$$

где M_{vert} — объем памяти, который занимает одна вершина,

M_{index} — объем памяти, который занимает один индекс,

M_{tc} — объем памяти, который занимают текстурные координаты одной вершины,

M_{norm} — объем памяти, который занимает одна нормаль,

M_i^{tex}— объем памяти, который занимает текстура,

M_i^{buf} — объем памяти, который занимает командный буфер или буфер с матрицей трансформации и цветом,

M_i^{desc} — объем памяти, который занимает описание ресурса графического интерфейса,

N_{vert}, N_{index}, N_{tc}, N_{norm} — количество вершин, индексов вершин, текстурных координат, нормалей.

Предположим, что геометрия и текстуры сцены S загружены в видеопамять, для ускорения рендеринга используется иерархия H. Во время рендеринга происходит составление и отправка команд рендеринга на графический процессор и выполнение рендеринга на графическом процессоре. Выведем формулу для оценки времени выполнения прямого рендеринга (составление буферов команд происходит на центральном процессоре, рендеринг выполняется за один проход) полигональной сцены S с учетом буферизации команд и проверок видимости:

$$T(S,H) = T_{buf}(S) + \max\left(T_{state}(S), T_{vert}(S), T_{frag}(S)\right) + T_{fc}(H) + T_{buf}(H) + \max\left(T_{state}(H), T_{vert}(H), T_{frag}(H), T_{z-buffer}(H)\right), \quad (2)$$

где S — множество объектов сцены,

H — множество узлов иерархии,

$T_{buf}(S)$ — время составления и отправки буферов команд,

$T_{state}(S)$ — время установки состояния конвейера,

$T_{vert}(S)$ — время выполнения преобразования вершин,

$T_{frag}(S)$ — время вычисления цвета пикселей,

$T_{fc}(H)$ — время выполнения отсечения узлов методом “Frustum Culling”,

$T_{z-buffer}(H)$ — время проверок видимости треугольников узлов иерархии H.

В формуле (2) выполняется суммирование, потому что команды составляются на центральном процессоре до посылки, а для точного выполнения проверок видимости требуется сначала выполнить рендеринг всех объектов сцены.

Для вычисления членов T_{vert}, T_{frag} требуются количество треугольников объекта $N_{\Delta}(O_i)$ и количество фрагментов (пикселей) $f(O_i)$:

$$T_{vert}(O_i) = N_{\Delta}(O_i)T_{\Delta}(m_k), \quad (3)$$

$$T_{frag}(O_i) = f(O_i)T_{frag}(m_k), \quad (4)$$

где $T_{\Delta}(m_k)$ — время рендеринга одного треугольника с материалом m_k,

$T_{frag}(m_k)$ — время рендеринга одного фрагмента (пикселя) с материалом m_k.

Для вычисления $T_{frag}(m_k)$ производится вывод треугольника на весь экран, потому что в этом случае преобладающее количество вычислений будет происходить во фрагментном шейдере. Для вычисления $T_{\Delta}(m_k)$ производится рендеринг большого количества маленьких треугольников, потому что в этом случае самой долгой стадией конвейера будут загрузка геометрии из памяти графического процессора и выполнение трансформации вершин. Для вычисления $T_{state}(O_i)$ происходит рендеринг одного маленького треугольника, что вызывает задержку, связанную со сменой состояния конвейера. Для измерения $T_{vis}(n_j)$ происходит рендеринг сцены с большим количеством треугольников и выполнение проверок видимости узлов иерархии.

В таблице 1 приведены результаты тестирования предложенной модели производительности. Выполняется генерация сцены, состоящей из случайно расположенных параллелепипедов, измеряется точное время рендеринга и вычисляется время по формуле (2), погрешность оценки. Показано, что модель производительности работает достаточно точно на различных генерируемых данных.

Табл. 1. Тестирование предложенной модели производительности при увеличении количества объектов

Кол-во объектов	Точное время выполнения рендеринга (миллисек.)	Оценка на основе предложенной модели (миллисек.)	Погреш-ть оценки (%)
10	0.00657	0.00611	7.05
20	0.00700	0.00668	4.64
30	0.00743	0.00751	1.14
100	0.01452	0.01338	7.88
1000	0.091	0.089	2.38
5000	0.442	0.424	4.19
50000	4.07	4.19	3.46
100000	7.71	8.38	8.77
300000	22.95	25.14	9.54

3. Оценка количества невидимых объектов

В этой главе предлагается метод оценки количества невидимых объектов в трехмерных сценах. Требования, предъявляемые к методу: консервативность, предсказуемое время выполнения.

Узлы иерархии с количеством объектов, значительно превышающим среднее количество объектов в одном узле, обозначаются, как затратные, этот статус распространяется снизу-вверх по иерархии. При этом родительский узел является затратным, если хотя бы один дочерний узел затратный. Для оценки количества невидимых объектов достаточно проверить видимость затратных узлов. Для повышения эффективности проверки видимости выполняются при обходе иерархии сверху вниз. Таким образом, у метода есть два параметра: минимальное количество объектов в затратных узлах, количество проверяемых узлов.

В таблице 2 приведено время выполнения метода (время обхода иерархии и выполнения проверок видимости) на генерированной сцене, содержащей 70000 объектов, при увеличении количества проверяемых узлов. Таким образом, предложенный метод позволяет за ограниченное время получить консервативную оценку количества невидимых объектов.

Табл. 2. Эффективность предложенного метода при увеличении количества проверяемых узлов

Количество проверяемых узлов	Оценка количества невидимых объектов предложенным методом	Время выполнения предложенного метода (миллисек.)
10	0	0.031
20	10513	0.059
30	20858	0.084
100	27523	0.261
300	39617	0.753
1000	42554	2.75
2000	45839	4.9
10000	48782	23.42

4. Адаптивный метод рендеринга

Предлагается адаптивный метод рендеринга больших динамических сцен, который использует предложенные выше метод оценки количества невидимых объектов и модель производительности графического процессора для выбора наиболее эффективного способа рендеринга. Кратко опишем основные способы рендеринга типовых состояний сцены:

1. Если в сцене мало объектов, предлагается выполнять перезапись буфера команд каждый кадр, отсечение объектов, не попадающих в область видимости камеры.
2. В сцену добавили некоторое количество объектов, при этом составление командного буфера стало отнимать существенное время. В этом случае нужно использовать иерархию для разбиения пространства, подготовить командные буфера для каждого узла иерархии, выполнять отсечение узлов, не попадающих в область видимости камеры.
3. После добавления очередного объекта количество невидимых объектов превысило пороговое значение. Предлагается выполнять аппаратные проверки видимости.

Рассмотрим более подробно каждый способ и условия переходов между ними. В первом способе каждый кадр измеряется время записи команд рендеринга. Адаптивный метод использует первый способ, пока выполняется условие:

$$T_{buf}(S) + T_{fc}(S) < T_{buf}(H) + T_{fc}(H), \quad (5)$$

где $T_{buf}(H)$ — время на составление и загрузку буферов команд, соответствующих узлам дерева, T_{fc} — время на отсечение объектов, не попадающих в область видимости камеры.

Первый способ рендеринга используется для динамичных сцен, в которых большинство объектов меняет свое положение, а также в случае, когда отсечение делать выгоднее на уровне объектов, чем на уровне узлов иерархии. Для сцен с малым количеством подвижных объектов и существенным временем записи ($T_{buf}(S) \gg T_{buf}(H)$) условие (5) перестает выполняться и происходит переход ко второму способу рендеринга. Для определения требований по памяти используется формула (1).

Для сокращения затрат на построение иерархии при рендеринге динамических сцен для пространственного индексирования объектов используется окто-дерево. Также каждому узлу окто-дерева соответствует буфер команд (secondary command buffer). Каждый объект попадает только в один буфер команд (single reference octree). Каждый кадр происходит оценка количества невидимых объектов с помощью метода, предложенного в главе 3. Адаптивный метод использует второй способ рендеринга до тех пор, пока выполняется условие:

$$T(S) < T(S_{vis}) + T_{vis}(H), \quad (6)$$

где $T(S_{vis})$ — время рендеринга видимых объектов,

$T_{vis}(H)$ — время выполнения проверок видимости.

Условие (6) выполняется, когда все объекты видимые или время выполнения проверок видимости неоправданно высокое. Переход к третьему способу рендеринга происходит, когда количество невидимых объектов достаточно высокое и, выполняя проверки видимости, можно сэкономить много времени.

Определим оптимальное количество проверок видимости. Время рендеринга сцены определяется по формуле:

$$T(q) = c(q)t_c + s(q)t_s + e(q)t_e, \quad (7)$$

где q — количество проверок видимости узлов иерархии,

$c(q)$ — количество записываемых команд рендеринга,

$s(q)$ — количество отправляемых командных буферов,

$e(q)$ — количество примитивов на лимитирующем этапе конвейера (вершин или пикселей),

t_c — среднее время записи одной команды рендеринга,

t_s — среднее время отправки командного буфера,

t_e — среднее время выполнения лимитирующего этапа конвейера (трансформации вершины, расчета цвета фрагмента).

Во время рендеринга накапливается статистика, которая позволяет определить функции $c(q)$, $s(q)$, $e(q)$. Количество невидимых вершин (пикселей) монотонно растет с увеличением количества проверок видимости. Зачастую рост происходит быстрее при малых q, а затем останавливается при больших q. Анализ показал, что логарифмическая зависимость является наилучшим приближением количества невидимых вершин с ростом количества проверок видимости. Накопив достаточное количество точек, можно выполнить регрессию для определения функций:

$$c(q) = \alpha_n c_n(n_0 - s_1 \ln(1+q)) + q, \quad (8)$$

$$s(q) = (n_0 + 1 - s_1 \ln(1+q)), \quad (9)$$

$$e(q) = (e_0 + pq - e_1 \ln(1+q)), \quad (10)$$

где c_n — среднее количество объектов в узле,

α_n — средняя доля перезаписываемых командных буферов, которая определяется по динамике объектов сцены до текущего момента времени,

n_0 — количество непустых узлов в иерархии,

e_0 — количество вершин (пикселей) при рендеринге сцены,

p — количество вершин (пикселей), приходящихся на один узел иерархии,

s_1 — коэффициент пропорциональности в зависимости количества невидимых узлов от количества проверок видимости,

e_1 — коэффициент пропорциональности в зависимости количества невидимых вершин (пикселей) от количества проверок видимости.

Определение минимума (7) вместе с ограничениями на количество проверок видимости $0 \le q \le n_0$ является задачей нелинейного программирования. Функция $T(q)$ является непрерывной и дифференцируемой на множестве ограничений, поэтому минимум является либо стационарной точкой, либо лежит на границе множества ограничений (теорема 14.1 [1]). Для нахождения стационарной точки решим уравнение $T'(q) = 0$ и получим:

$$q^* = \frac{\alpha_n c_n s_1 t_c + s_1 t_s + e_1 t_e}{p t_e + t_c} - 1. \quad (11)$$

Сравнив $T(0)$, $T(n_0)$, $T(q^*)$, можно найти минимальное время выполнения рендеринга сцены и соответствующее значение q.

5. Тестирование

Тестирование проводится на трех динамических сценах (см. рис. 2), характеристики которых приведены в таблице 3. В процессе анимации сцен происходит добавление новых объектов, удаление временных объектов, изменение цвета объектов. Количество объектов в тестовых сценах растет со временем.

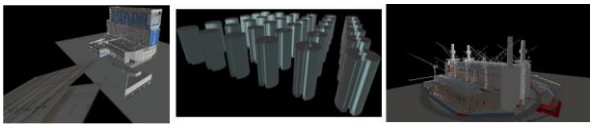

Рис. 2. Тестовые сцены 1, 2, 3.

Табл. 3. Характеристики тестовых сцен

Сцена	Количество вершин	Количество треугольников	Количество объектов
1	32483139	10827713	71961
2	30462912	10154304	221796
3	94388454	31462818	270431

Тесты проводятся на компьютере с конфигурацией: Intel Core i7-7700 3.6GHz, 16GB RAM, NVIDIA Geforce GTX 1070. Тест заключается в перемещении камеры по сцене, проигрывании анимации и измерении времени рендеринга кадра с применением одного из трех методов ускорения рендеринга:

1. Отсечение узлов дерева, не попадающих в камеру (frustum culling). Объекты видимых узлов дерева добавляются в командный буфер при иерархическом обходе дерева сверху вниз.
2. Проверка видимости всех листовых узлов дерева на графическом процессоре (hardware occlusion query). Проверки осуществляются относительно буфера глубины графического процессора с помощью команд API (vkCmdBeginQuery, vkGetQueryPoolResults). Получение результатов запросов видимости происходит с задержкой в несколько кадров для избежания простоя центрального процессора. Также используется техника составления вспомогательных командных буферов для узлов дерева и отсечение узлов дерева, не попадающих в камеру.
3. Предложенный адаптивный метод, который выбирает наиболее эффективный способ рендеринга в данный момент времени.

При рендеринге использовались три кадровых буфера (triple buffering), выполнялся вывод на экран последнего записанного кадрового буфера (mailbox), не применялось сглаживание, запись командных буферов осуществлялась в однопоточном режиме.

Сравнение предложенного адаптивного метода с методами “Frustum culling”, “Occlusion queries” представляет интерес, потому что данные методы

используются во многих программах трехмерного рендеринга.

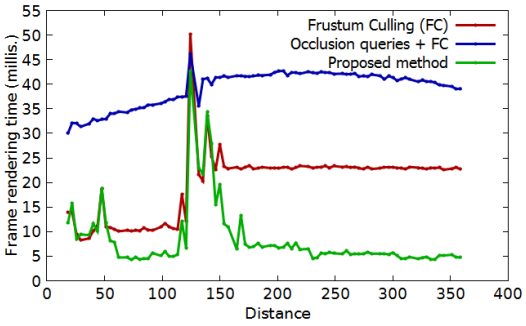

Рис. 3. Сравнение времени рендеринга кадра трех методов рендеринга при проходе по сцене 1 и одновременном проигрывании анимации.

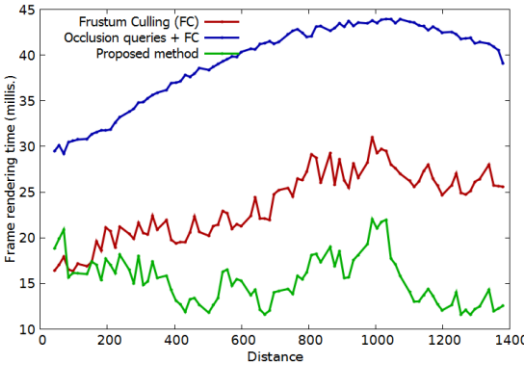

Рис. 4. Сравнение времени рендеринга кадра трех методов рендеринга при проходе по сцене 2 и одновременном проигрывании анимации.

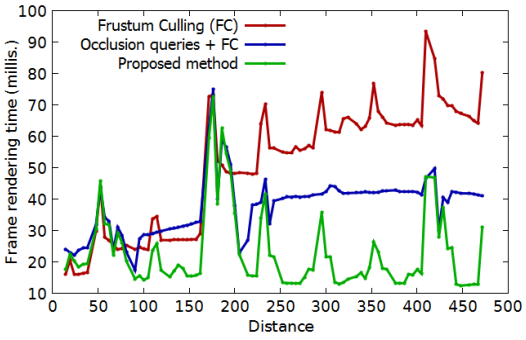

Рис. 5. Сравнение времени рендеринга кадра трех методов рендеринга при проходе по сцене 3 и одновременном проигрывании анимации.

Тестирование показывает, что метод выполнения аппаратных проверок видимости всех листовых узлов отнимает значительное время и метод “Frustum culling” работает эффективнее, чем “Occlusion queries” (см. рис. 3, 4). В сценах с относительно большим количеством полигонов проверки видимости все-таки дают прирост производительности (см. рис. 5). Адаптивный метод последовательно применяет три описанных способа рендеринга по мере добавления объектов в сцену во время тестирования. В начале анимации сцен используется простой способ записи командных буферов и отсечение объектов. Когда запись команд рендеринга объектов отнимает значительное время, происходит переход ко второму способу рендеринга. Затем на основе накопленной статистики вычисляется количество проверок видимости для эффективного рендеринга. Ухудшение производительности адаптивного метода (скачки на графике) в основном связано с перезаписыванием командных буферов узлов иерархии при появлении новых объектов. В среднем адаптивный метод дает наилучший результат на рассмотренных тестовых сценах.

6. Заключение

Была предложена модель производительности прямого рендеринга для оценки объема потребляемой памяти и времени выполнения рендеринга с учетом буферизации команд и проверок видимости. Был разработан и реализован адаптивный метод управления командными буферами и проверками видимости для эффективного рендеринга динамических трехмерных сцен. Адаптивный метод использует предложенную модель производительности рендеринга и метод оценки количества невидимых объектов для выбора наиболее эффективного способа рендеринга данного состояния динамической сцены. Тестирование показало, что предложенный адаптивный метод эффективно справляется с рендерингом динамических сцен.

7. Литература

[1]. Черняк, А. А., Черняк, Ж. А., Метельский, Ю. М., Богданович, С. А. Методы оптимизации: теория и алгоритмы. 2-е-изд. М.: Юрайт, 2017. 357 с.

[2]. Bittner J., Wimmer M., Piringer H., Purgathofer W. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful // Computer Graphics Forum. 2004. Vol. 23, No 3. pp. 615–624.

[3]. Blackert A. Evaluation of Multi-Threading in Vulkan. Linköping University, 2016.

[4]. Bucci J. and Doghramachi H. Deferred+: next-gen culling and rendering for dawn engine. URL: https://eidosmontreal.com/en/news/deferred-next-gen-culling-and-rendering-for-dawn-engine.

[5]. Cohen-Or D., Chrysanthou Y.L., Silva C.T., Durand F. A survey of visibility for walkthrough applications // IEEE Trans. Visual. Comput. Graphics. 2003. Vol. 9, No 3. pp. 412–431.

[6]. Funkhouser T.A., Séquin C.H. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments // Proceedings of the 20th annual conference on Computer graphics and interactive techniques. ACM Press, 1993. pp. 247–254.

[7]. Gonakhchyan V. Efficient command buffer recording for accelerated rendering of large 3d scenes // Proceedings of the 12th International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing. 2018. pp. 397–402.

[8]. Guthe M., Balázs Á., Klein R. Near Optimal Hierarchical Culling: Performance Driven Use of Hardware Occlusion Queries // Eurographics Symposium on Rendering. 2006. pp. 207–214.

[9]. History of OpenGL. URL: https://www.khronos.org/opengl/wiki/History_of_OpenGL#OpenGL_1.5_.282003.29.

[10]. Lorach T. Approaching Zero Driver Overhead // SIGGRAPH 2014.

[11]. Shiraef J. An exploratory study of high performance graphics application programming interfaces. University of Tennessee at Chattanooga, 2016.

[12]. Wimmer M., Wonka P. Rendering time estimation for real-time rendering // Proceedings of the 14th Eurographics workshop on Rendering. 2003. pp. 118–129.

Об авторах

Гонахчян Вячеслав Игоревич, младший научный сотрудник ИСП им. В.П. Иванникова РАН. E-mail: pusheax@ispras.ru.