УДК 004.925.4

О.Н. Романюк[1], д.т.н., проф.,
О.О. Дудник[1], аспірант,
Н.С. Костюкова[2], к.т.н., доц.
[1]Вінницький національний технічний університет
[2] Донецький національний технічний університет
dudnyk@vntu.edu.ua

Реалізація альтернативного конвеєра рендерингу на GPU з використанням обчислювальних шейдерів

Запропоновано метод програмної реалізації альтернативного конвеєра рендерингу на GPU загального призначення шляхом використання обчислювальних шейдерів. Практична реалізація підтвердила працездатність методу на прикладі текстурування з виконанням процедурних операцій в об'єктному просторі.
Ключові слова: конвеєр рендерингу, шейдер, обчислювальний шейдер, текстурування

DOI: 10.31474/1996-1588-2017-2-25-103-108

Вступ

Підвищення реалістичності комп'ютероної графіки досягають за рахунок розробки складніших математичних, моделей, методів та алгоритмів. Впровадження нових методів на базі існуючого апаратного забезпечення, часто, є неможливим. Архітектура сучасних графічних акселераторів орієнтується на загальноприйнятий класичний конвеєр рендерингу [1]. Проте існують методи комп'ютерної графіки, які вимагають специфічного порядку виконання стадій візуалізації [2]. Розробка спеціалізованого апаратного забезпечення для задоволення цих вимог недоцільна, оскільки такі системи не мають масового застосування. Тому актуальним є питання реалізації специфічних конвеєрів рендерингу на GPU загального призначення.

Аналіз літератури та постановка задачі.

В основі архітектури сучасних графічних акселераторів лежить уніфікована шейдерна модель, що передбачає поетапне виконання шейдерних програм у послідовності, що відповідає класичному конвеєру рендерингу, а саме [1]:

1) вершинний шейдер використовується для виконання геометричних перетворень та встановлення атрибутів вершин;
2) геометричний шейдер призначений для обробки заданих примітивів та генерації нових;
3) шейдери теселяції для процедурної генерації геометрії поверхонь довільної форми;
4) фрагментний шейдер використовують для попіксельної обробки фрагментів зображення та визначення кольорів;

Для візуалізації з використанням методів текстурування відповідно до описаної вище моделі, як правило, достатньо вершинного та фрагментного шейдера. У вершинному шейдері встановлюють відповідність координат вершин у просторі та на текстурі, а також виконують MVP-перетворення. Фрагментний шейдер використовують для вибірки кольорів пікселів із текстурної пам'яті та виконання фільтрації текстури за обраним алгоритмом. При цьому координати вершин у просторі та на текстурі в пам'ять графічного акселератора із пам'яті хост-комп'ютера передають шляхом запису в буфер вершин (Vertex Buffer Object, VBO), а дані текстури записують у uniform-слот акселератора (рис. 1) [3].

Описана послідовність операцій не дає можливості практичної реалізації методу текстурування з виконанням процедурних операцій в об'єктному просторі засобами GPU, оскільки метод передбачає перед початком растеризації виконання лише MV-перетворення та повороту полігону, а P-перетворення та зворотній попіксельний поворот виконують після визначення кольорів [2]. Існуючий конвеєр рендерингу не має можливості виконання геометричних перетворень над окремими пікселями та зміни координат пікселів після визначення їх кольорів. Тому для текстурування з виконанням процедурних операцій у об'єктному просторі в реальному часі на GPU можливо лише за умови розробки альтернативного графічного конвеєра та його інтеграції в засоби апаратної візуалізації.

Рисунок 1 – Апаратний рендеринг з використанням текстурування на основі класичного конвеєру

Стандарти OpenGL, починаючи з версії 4.3 2012 року, DirectX - з версії 11 2008 року та Vulkan 1.0 2016 року передбачають можливість використання обчислювальних шейдерів [4, 5, 6]. Обчислювальні шейдери виконуються поза конвеєром рендерингу та використовуються для виконання довільних обчислень засобами GPU. Реалізація альтернативного конвеєра рендерингу можлива шляхом комбінування набору обчислювальних і графічних шейдерів. При цьому для забезпечення обміну даними між хост-комп'ютером і шейдерами різних типів у пам'яті GPU можна використовувати шейдері буфери зберігання (Shader Storage Buffer Object, SSBO), які забезпечують можливість зчитування та запису масивів даних у обчислювальних і фрагментному шейдерах. Об'єми пам'яті, доступні кожному SSBO, виділяються хост-комп'ютером до початку виконання шейдерної програми та не можуть бути змінені в ході її виконання.

Мета роботи – розробка методу застосування альтернативних конвеєрів рендерингу на GPU загального призначення.

Опис методу

Для реалізації текстурування з виконанням процедурних операцій в об'єктному просторі необхідно створити 11 буферів SSBO:

- буфер координат для зберігання координат вершин полігонів;
- буфер текстурних координат для зберігання координат вершин в текстурній площині;
- буфер площ для зберігання площ полігонів;
- буфер поворотів для зберігання матриць зворотного повороту;
- фрагментний буфер для зберігання координат фрагментів в об'єктному просторі;
- буфер кольорів для зберігання кольорів фрагментів;
- індексний буфер для зберігання даних про приналежність кожного фрагмента конкретному полігону;
- буфер зміщень для зберігання зміщень сегментів пам'яті виділених конкретному полігону;
- екранний буфер для зберігання кольорів вихідного зображення;
- буфер глибини для зберігання даних, необхідних для видалення невидимих точок;
- буфер блокування для реалізації мютексів та синхронізації доступу до буферу екрану та глибини.

Загальний конвеєр повинен складатись із 4 обчислювальних шейдерів та 1 фрагментного:

- Шейдер повороту в якості вхідних даних приймає координати вершин полігонів із буфера вершин. Здійснює MV-перетворення і поворот полігона в положення паралельне площині екрану. Потім записує нові координати назад у буфер вершин і визначає матрицю зворотного повороту, яку записує її до буферу поворотів. Після цього визначає площу полігону із врахуванням дискретності простору та записує її у буфер площ. Викликається один раз для кожного полігону (рис. 2).
- Шейдер растеризації в якості вхідних даних приймає координати вершин полігонів із буфера вершин, індекс полігону з індексного буфера та зміщення в пам'яті з буфера зміщень. Здійснює розбиття полігону на фрагменти та записує координати фрагментів до фрагментного буфера відповідно до зміщення. Викликається один раз для кожного полігону (рис. 3).
- Шейдер текстурування в якості вхідних даних приймає координати вершин полігонів із буфера вершин, індекс полігону з індексного буфера, координати пікселя із фрагментного буфера, текстурну карту із текстурного слоту та текстурні координати вершин із буферу текстурних координат. Здійснює визначення текстурних координат пікселя, вибірку кольору із текстури та запис його в буфер кольорів. Викликається один раз на кожен фрагмент (рис. 4).

Рисунок. 2 – Шейдер повороту

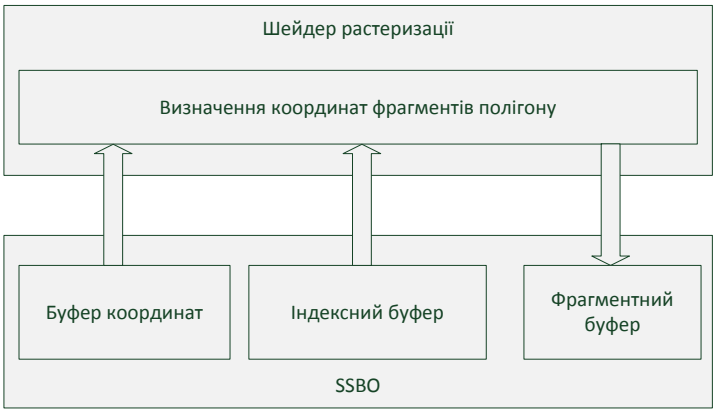

Рисунок 3 – Шейдер растеризації

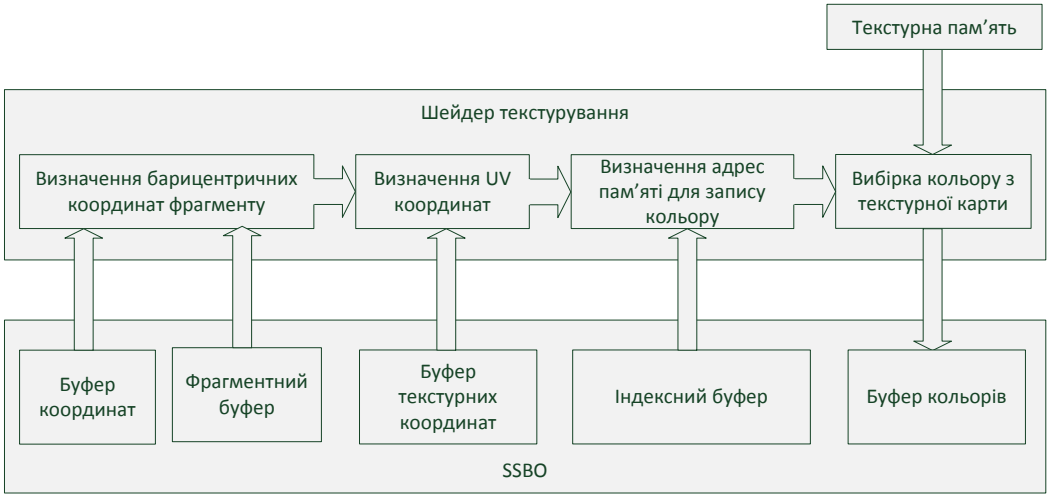

Рисунок 4 – Шейдер текстурування

- Шейдер зворотного повороту у якості вхідних даних приймає координати координати пікселя із фрагментного буфера, колір пікселя із буферу кольорів, індекс полігону із індексного буфера, кольори та вагові коефіцієнти з буферу екрану, z-компоненту найближчого видимого пікселя та індекс полігону, якому він належить із буферу глибини. Здійснює визначення екранних координат пікселя в екранній площині та вагових коефіцієнтів, P-перетворення, запис кольору пікселя та вагового коефіцієнту до буферу екрану, визначення видимості пікселя та запис до буферу глибини. Викликається один раз на кожен фрагмент (рис. 5).

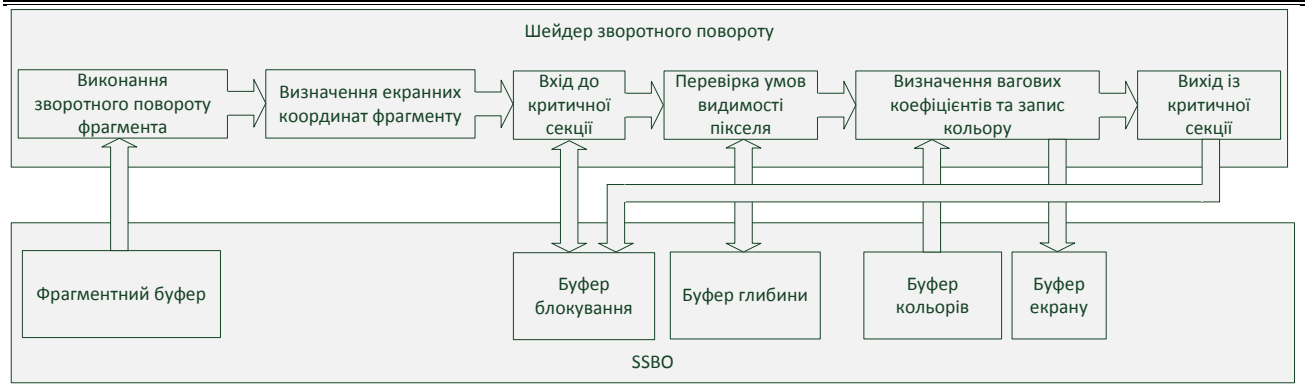

Рисунок 5 – Шейдер зворотного повороту

- Фрагментний шейдер у якості вхідних даних приймає суму інтенсивностей кольорів та суму вагових коефіцієнтів пікселя та визначає колір пікселя на екрані. Викликається один раз на кожен піксель екрану (рис. 6).

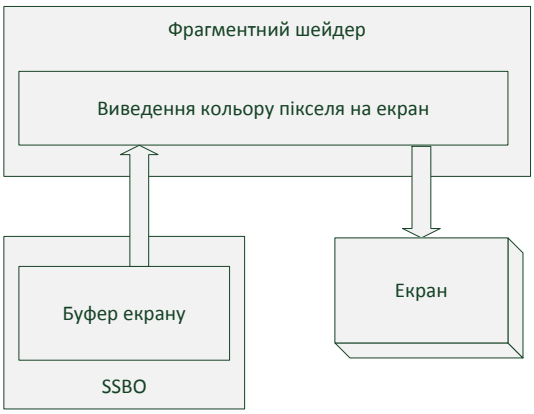

Рисунок 6 – Фрагментний шейдер

Управління усіма етапами процесу візуалізації здійснюється хост-комп'ютером засобами CPU.

До початку процесу рендерингу виділяється пам'ять для буферу координат і текстурних координат пропорційно кількості вершин; буфера площ, зміщень та поворотів пропорційно кількості полігонів; буферів екрану, глибини та блокування пропорційно площі екрану. Після виділення пам'яті до буферів координат завантажуються координати вершин у просторі та в площині текстури. Об'єм пам'яті, необхідний для інших буферів, залежить від кількості фрагментів (площі полігонів із урахуванням дискретності простору), а тому невідомий на початковому етапі рендерингу.

Наступним етапом є завантаження в оперативну пам'ять кольорової текстурної карти та її запис у вільний текстурний слот графічного акселератора у якості іменованого uniform-об'єкта.

Далі управління передається графічному процесору для запуску шейдеру повороту. Після виконання усіх операцій шейдеру повороту з буфера площ в оперативну пам'ять хост-комп'ютера завантажуються площі полігонів. Далі здійснюється виділення пам'яті для фрагментного буфера та буфера кольорів пропорційно сумі площ полігонів.

Оскільки растеризація полігонів на GPU відбувається паралельно, послідовний запис до фрагментного буфера вимагає виконання операцій блокування для синхронізації потоків. Тому доцільніше заздалегідь визначити зміщення в пам'яті для кожного полігону, що виключає одночасний доступ до однієї ділянки пам'яті кількома потоками, та виконувати операції запису в довільну порядку. Після виділення пам'яті для фрагментного буфера та буфера кольору слід визначити зміщення в цих ділянках пам'яті для кожного полігону. Зміщення визначаються ітераційно за формулою:

$$offset_i = offset_{i-1} + S_i,$$

де, $offset_i$ - зміщення полігону в пам'яті буфера; $offset_{i-1}$ - зміщення попереднього полігону в пам'яті буфера; S_i - площа поточного полігону. Для першого полігону зміщення рівне 0.

Наступним кроком є передача управління GPU для виконання програми шейдеру растеризації. Під час растеризації координати фрагментів записуються до фрагментного буферу за адресами, що відповідають зміщенням полігонів. Адреса кожного фрагменту визначається за формулою:

$$address_i = offset_P + i,$$

де, $address_i$ - адреса поточного пікселя в пам'яті фрагментного буфера; $offset_P$ - зміщення поточного полігону в пам'яті фрагментного буферу; i - порядковий номер поточного пікселя в межах полігону.

У подальшому виконується запуск на виконання програми шейдеру текстурування для визначення кольорів фрагментів.

Після визначення кольорів процес підготовки до рендерингу закінчено і виконується

цикл рендерингу кадрів зображення. В основному циклі рендерингу виконуються виконання програми шейдеру зворотного повороту, програми фрагментного шейдеру та обробка сигналів вводу. Вихід із циклу відбувається після надходження відповідного сигналу від користувача. Загальну схему процесу візуалізації запропонованим методом зображено на рисунку 7.

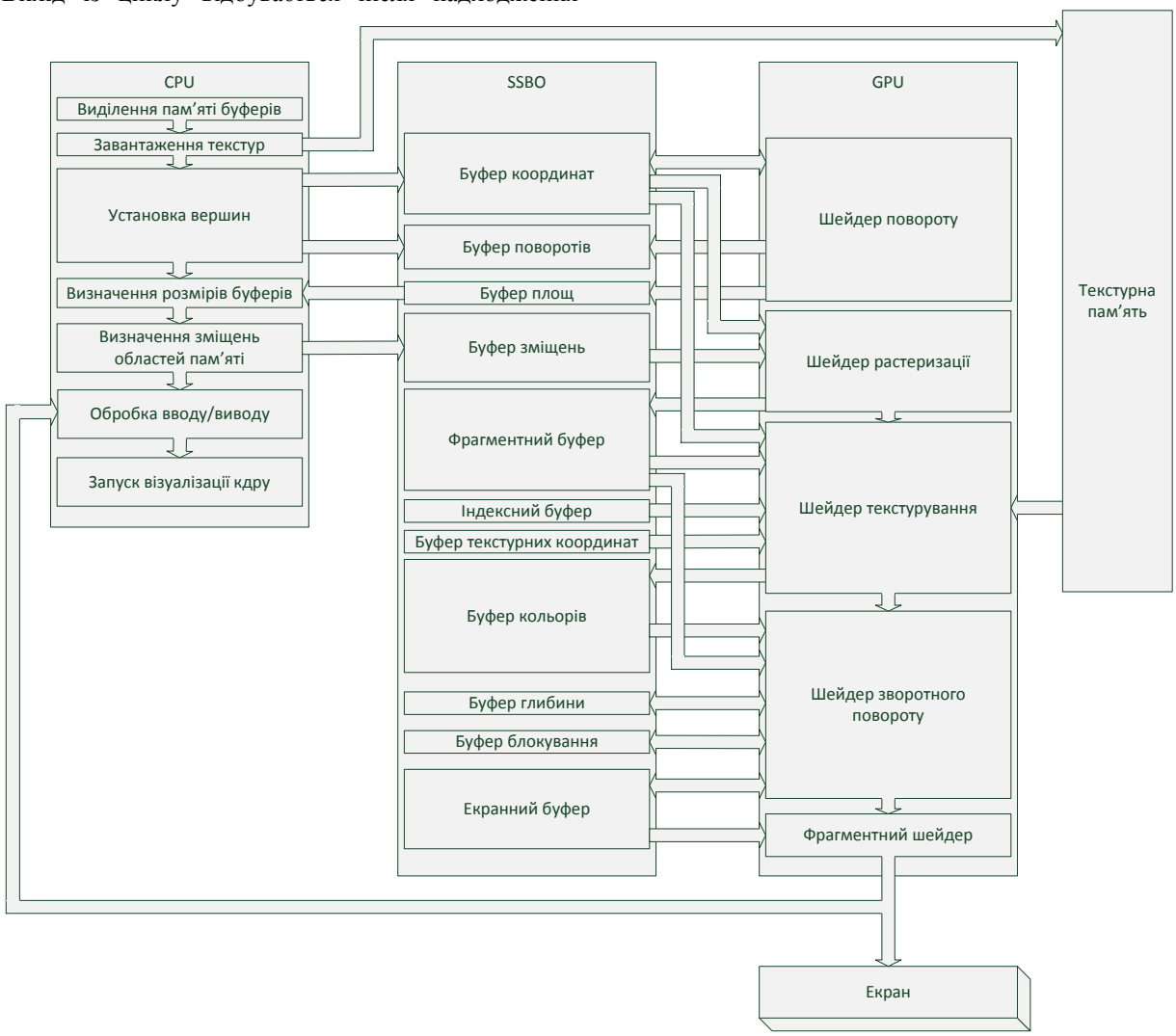

Рисунок 7 – Апаратний рендерингу з використанням методу текстурування з виконанням операцій в об'єктному просторі

Програмне забезпечення для хост-комп'ютера розроблено з використанням мови програмування C++ відповідно до стандарту C++ ISO/IEC C++17 [7]. Взаємодія з відео акселератором відбувається у відповідності до стандарту OpenGL 4.3 [4]. Вихідний код може бути зкомпільованим у якості додатку або бібліотеки для усіх сучасних версій операційних систем Windows, GNU/Linux, MacOS, а також мобільних платформ (за умови наявності апаратної підтримки платформою відповідних версій OpenGL). Перевірено на сумісність із компіляторами GCC, починаючи з версії 7.2.0 та clang, починаючи з версії 4.0.1. Шейдерні програми розроблено мовою програмування OpenGL Shading Language (GLSL) відповідно до стандарту OpenGL 4.3.

В ході тестування виявлено приріст продуктивності відносно рендерингу з використанням текстурування на основі анізотропної фільтрації складає 15%-40%. Продуктивність рендерингу прямо залежить від загальної площі полігонів, тоді як продуктивність рендерингу з використанням анізотропної фільтрації залежить від площі екрану, кількості полігонів та положення полігону відносно спостерігача.

Висновки

Запропоновано метод програмної реалізації альтернативного конвеєра рендерингу на GPU загального призначення шляхом використання обчислювальних шейдерів. Практична реалізація підтвердила працездатність методу на прикладі текстурування з виконанням процедурних операцій в об'єктному просторі.

Список використаної літератури

1. Романюк О. Н. Еволюція конвеєра рендерингу в відео картах / О. Н. Романюк, О. О. Дудник // Збірник матеріалів міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ», 24-25 жовтня 2016 р. – 2016. – С. 440-448.
2. Романюк О. Н. Підвищення продуктивності текстурування з виконанням процедурних операцій в об'єктному просторі / О. Н. Романюк, О. О. Дудник // Наукові праці ДонНТУ. Серія "Інформатика, кібернетика та обчислювальна техніка". - 2016. - № 2 (23). - С. 45-51.
3. Вольф Д. OpenGL 4. Язык шейдеров. Книга рецептов / Дэвид Вольф // ДМК Пресс. – 2015. – 368с.
4. Kessenich J. The OpenGL Shading Language Version: 4.30 Document Revision: 7 / John Kessenich, Dave Baldwin, Randi Rost // The Khronos Group Inc . - Feb-2013
5. Direct3D 11 Graphics // Режим доступу: https://msdn.microsoft.com/en-us/library/windows/desktop/ff476080(v=vs.85).aspx
6. Vulkan 1.0.66 - A Specification / The Khronos Vulkan Working Group // Version 1.0.66, 2017-11-27 09:15:39Z
7. Smith R. Working Draft, Standard for Programming Language C++ N4659 / Richard Smith // 2017-03-21. - Режим доступу: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf

Надійшла до редакції 15.12.2017

References

1. Romanyuk, O. N., Dudnik, O. O. (2016) Evolution of render pipeline in video-cards [Evolyutsiya konveyera renderynhu v video kartakh], Electronic information resources: creation, use of access. Collection of international lectures and practical conferences, Vinnytsya, pp. 440-448.
2. Romanyuk, O. N., Dudnik, O. O. (2016) Increasing the productivity of texturing with performing procedural operations in the object space [Pidvyshchennya produktyvnosti teksturuvannya z vykonannyam protsedurnykh operatsiy v ob″yektnomu prostori], Scientific works of Donetsk National Technical University. Series: Informatics, Cybernetics and Computer Science, No. 2(23), pp. 45-51.
3. Wolf D. (2015). OpenGL 4 Shading Language [OpenGL 4Yazyk sheyderov. Kniga retseptov], Moscow, 368 pp.
4. Kessenich J. The OpenGL Shading Language Version: 4.30 Document Revision: 7 / John Kessenich, Dave Baldwin, Randi Rost // The Khronos Group Inc . - Feb-2013
5. Direct3D 11 Graphics, available at: https://msdn.microsoft.com/en-us/library/windows/desktop/ff476080(v=vs.85).aspx
6. Vulkan 1.0.66 - A Specification/.The Khronos Vulkan Working Group. Version 1.0.66, 2017-11-27 09:15:39Z
7. Smith R. Working Draft, Standard for Programming Language C++ N4659, available at: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf

А.Н. РОМАНЮК[1], А.А. ДУДНИК[1], Н.С. КОСТЮКОВА[2]
[1]Вінницький національний технічний університет
[2] Донецький національний технічний університет

РЕАЛИЗАЦИЯ АЛЬТЕРНАТИВНОГО КОНВЕЙЕРА РЕНДЕРИНГА НА GPU С ИСПОЛЬЗОВАНИЕМ ВЫЧИСЛИТЕЛЬНЫХ ШЕЙДЕРОВ

Обоснована необходимость в использовании нетрадиционных конвейеров рендеринга для различных задач компьютерной графики. Предложен метод программной реализации альтернативного конвейера рендеринга на GPU общего назначения путем использования вычислительных шейдеров. Показано работоспособность метода на примере шейдерной реализации метода текстурирования с выполнением процедурных операций в объектном пространстве.

Ключевые слова: конвейер рендеринга, шейдер, вычислительный шейдер, текстурирование.

A.N. ROMANYUK [1], O.O. DUDNYK[1], N.S. KOSTYUKOVA [2]
[1]Vinnytsia National Technical University
[2] Donetsk National Technical University

IMPLEMENTATION OF THE ALTERNATIVE RENDERING CONVEYOR IN GPU USING COMPUTING SCHEIDERS

In article, the necessity for using non-traditional rendering pipelines for various computer graphics tasks is justified. A method is proposed for software implementation of an alternative rendering pipeline on a general-purpose GPU by using computational shaders. The workability of the method is demonstrated using the example of the shader implementation of the texturing method with the execution of procedural operations in the object space.

***Keywords: rendering pipeline, shader, computating shadesr, texture mapping*.**