УДК 004.04
DOI https://doi.org/10.32838/2663-5941/2021.5/19

Несук О.О.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Потапова К.Р.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Тарасенко-Клятченко О.В.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

ПРО ОСОБЛИВОСТІ ЗОБРАЖЕННЯ ГРАФІЧНИХ ОБ'ЄКТІВ З УРАХУВАННЯМ АПАРАТНО ПРИСКОРЕНОГО ПІДХОДУ

У статті наведено результати досліджень щодо вивчення процесу рендерингу зображень, а саме швидкості їх відтворення. Нині комп'ютерна графіка є основною технологією цифрової фотографії, кіно, відеоігор, додатків для мобільних телефонів і багатьох спеціалізованих програм. Розроблено велику кількість спеціалізованого обладнання та програмного забезпечення, причому дисплеї більшості пристроїв керуються апаратним забезпеченням комп'ютерної графіки. Це величезна область інформатики.

Натепер існує стандартизований алгоритм рендерингу, за яким на зображенні представляються об'єкти в просторі. Створене зображення виводиться на екран пристрою, візуалізуючи графічну інформацію. Об'єкти вимальовуються на графічному представленні в порядку, що залежить від відстані об'єкта до спостерігача, починаючи з найдальшого. Кожен наступний об'єкт накладається залежно від своєї позиції в просторі. Ці об'єкти перекривають собою інші або взагалі можуть бути поза полем зору, витрачаючи графічний ресурс перезаписом бітової інформації пікселів у стеку. Є різні методи, які дозволяють уникнути частини операцій графічного процесора, що виконуються «впусту». Такими методами є, наприклад, алгоритми frustum culling та occlusion. Однак навіть за використання таких підходів ми все одно втрачаємо частку графічного ресурсу (операцій графічного процесора).

Для забезпечення високої продуктивності й розв'язання проблеми втрати графічного ресурсу був спроєктований зворотний алгоритм представлення зображень. Відповідно до такого підходу об'єкти в комп'ютерному просторі відтворюються в порядку, починаючи з найближчого об'єкта й закінчуючи найдальшим. Водночас кожен наступний об'єкт відрізається за контуром уже представленої частини повного зображення. Таким чином мінімізується втрата графічного ресурсу шляхом перезапису пікселів у стеку. Так, кожен піксель зображення записується лише один раз, економлячи графічний ресурс.

Ключові слова: *растрова графіка, рендеринг, frustum culling, occlusion, зворотний алгоритм, overdraw.*

Постановка задачі. Під час створення зображення виникає проблема ефективного використання графічного ресурсу, тобто перезапису частини бітових даних пікселів у стеку, що є марним використанням графічних потужностей. Для вирішення цієї задачі було розроблено зворотний алгоритм виведення зображення. Таким чином, стандартні алгоритми рендерингу вже не у повній мірі відповідають сучасним вимогам з огляду на велику кількість і складність операцій із графічними об'єктами.

Є проблема марної витрати графічного ресурсу внаслідок появи множини пікселів, які вповільнюють частоту зміни кадрів. У цій роботі розглядається зворотний алгоритм рендерингу, завдяки якому мінімізується кількість операцій із графічними об'єктами без втрати візуальної складової «картинки».

Аналіз останніх досліджень і публікацій. У сучасних системах найпоширенішим механізмом відтворення зображення є *draw call*. Це програмний виклик, який містить усю інформацію про візуалізацію об'єкта, яку потрібно відтворити, та передає її до графічного процесора.

Для повної візуалізації кожного об'єкта виконується декілька викликів, що може призвести до проблеми *overdraw* – надлишкового нашаровування зображень одне на одне. Цей надлишок

з'являється після відтворення одного об'єкта, який частково чи повністю перекривається іншим, що показує втрату графічного ресурсу.

Є декілька методів для підвищення ефективності виводу зображень, наприклад, *frustum culling* та *occlusion culling*, які відсікають об'єкти, що знаходяться поза зором, але ці алгоритми не вирішують повністю проблему *overdraw*.

Постановка завдання. З огляду на результати вивчення літературних джерел за проблемою, цілями статті є аналіз стандартного алгоритму рендерингу зображень та відомих методів підвищення його ефективності, розробка зворотного алгоритму візуалізації об'єктів.

Виклад основного матеріалу дослідження. Для візуалізації зображення використовується віртуальна камера. Налаштування віртуальної камери, яке зазвичай виконується за допомогою спеціальних функцій, визначає поле, об'єкти всередині якого будуть представлятися на екрані.

Поле зору *frustum* – це простір, який містить усе, що потенційно видно на екрані. Цей простір визначається відповідно до налаштувань віртуальної камери, і при використанні перспективної проекції приймає форму усіченої піраміди.

Вершина цієї піраміди - це положення камери, а основа піраміди - віддалена площина. Піраміда усічена на ближній до спостерігача площині.

Всі об'єкти, які потенційно можна буде побачити в результаті рендерингу, знаходяться всередині, принаймні частково, усіченої піраміди, тому немає необхідності відтворювати те, що знаходиться поза нею, оскільки це не буде видно (рис. 1).

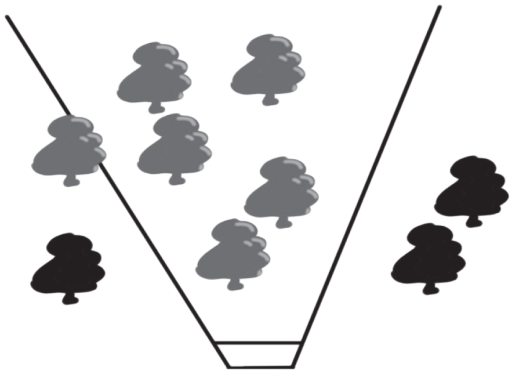

Рис. 1. Модель відтворення об'єктів, що знаходяться в полі зору віртуальної камери

На рис. 1 показано поле зору віртуальної камери, на екрані буде відтворено усі світлі дерева, у той час, як всі темні дерева не буде видно за результатом рендерингу.

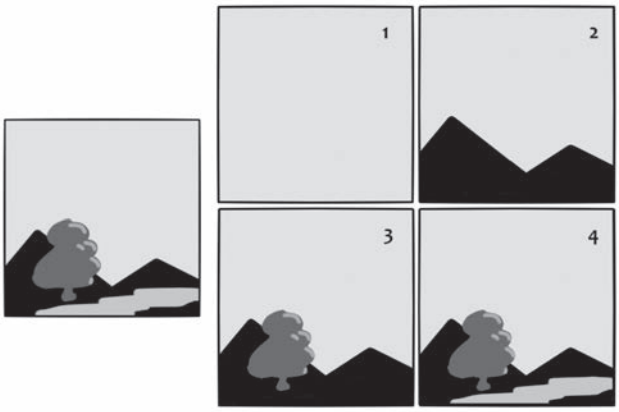

Рис. 2. Стандартний алгоритм рендерингу об'єктів

За стандартного алгоритму (рис. 2), об'єкти представляються на екрані у порядку від найдальшого до найближчого. Це призводить до перекривання деяких частин об'єктів один одним, що уповільнює процес рендерингу. Деякі алгоритми, наприклад, frustum culling та occlusion, підвищують швидкість рендерингу кадру, ігноруючи об'єкти, які не буде видно на зображенні.

Метою методу покращення ефективності рендерингу *frustum culling* є ідентифікування об'єктів, що знаходяться всередині поля зору (повністю або частково), та вилучення тих, що не знаходяться всередині. До графічного процесора надсилається візуальна інформація лише тих об'єктів, що розташовані у полі зору, навіть якщо спостерігач бачить їх лише частково. Врешті-решт, все, що вимагається від графічного процесора, - це представити на екрані всі об'єкти, які потенційно може бачити спостерігач, заощаджуючи на обробці всіх тих об'єктів, які є невидимими. Крім того, цей метод покращує швидкість виведення нових кадрів та саму продуктивність програми, оскільки лише ті об'єкти, які є частиною видимого 3D-простору, зберігаються у графічній пам'яті, і це легше за представлення усього 3D-простору.

Цей метод має сенс, якщо частина об'єктів тривимірного світу, яка знаходиться у полі зору, значно менша, ніж сам 3D-простір. Тобто, коли увесь простір знаходиться у полі зору, особливого сенсу використовувати метод *frustum culling* немає, оскільки жоден об'єкт не можна ігнорувати.

Також цей метод представляє всі об'єкти, навіть ті, які видно лише частково, повністю витрачаючи на це графічний ресурс.

Також існує метод *occlusion culling*, ефективніший за *frustum culling*. *Occlusion culling* створює ієрархію об'єктів, які потенційно буде видно на картинці. Таким чином, у програмному полі

будуть показані лише ті об'єкти, які точно видно на екрані. Це зменшує кількість викликів *drawcall* та підвищує швидкодію роботи програми.

У цей час *зворотний алгоритм* передбачає обрізання частини об'єктів, які перекриваються вже виведеним зображенням (рис. 3). Об'єкти виводяться у порядку, вказаному на рис. 3.

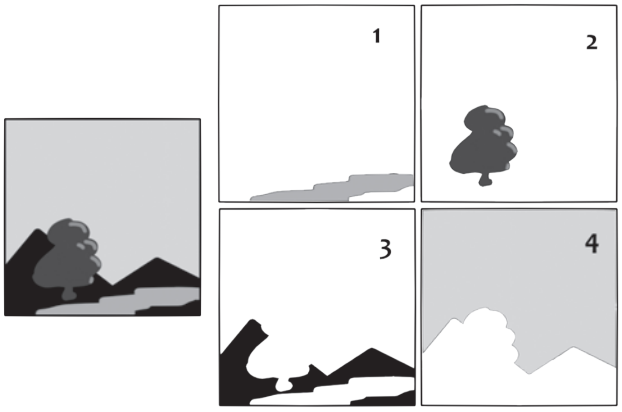

Рис. 3. Зворотній алгоритм рендерингу об'єктів

Під час нашаровування цих зображень можна отримати повне зображення, яке не відрізняється від такого, що було створене за стандартним алгоритмом. Використовуючи зворотний алгоритм, можна уникнути перезапису окремих частин зображення.

Частина пікселів X_{i+1}, які виводяться для кожного наступного об'єкта, дорівнює деякій частині пікселів Y_{i+1}, яка знаходиться поза множиною вже виведених попередніх об'єктів $\bigcup_{0}^{i} X$. Тобто за формулою:

$$X_{i+1} = Y_{i+1} \oplus \left(Y_{i+1} \cap \sum_{0}^{i} X \right), \qquad (1)$$

перший об'єкт буде відтворюватися повністю. За відсутності використання альфа каналу, кількість виведених пікселів буде дорівнювати полю програми.

Порівнявши цей алгоритм із стандартним, ми бачимо, що у стандартному алгоритмі сума всіх представлених пікселів дорівнює $\bigcup_{n} Y_i$, тоді як у зворотному ця сума становить $\bigcup_{0} X_i$, де:

$$X_i \text{ " } Y_i . \qquad (2)$$

Необхідно зауважити, що під час роботи зворотний алгоритм витрачає більше операцій на запис пікселя у стек через перевірку того, чи не є місце поточного пікселя вже записаним.

Для здійснення моделювання використано мову C++ та бібліотеку *freeglut*. За алгоритмом, представленим на рис.4, виконується виведення зображень на екран у зворотному порядку.

Для ефективної роботи алгоритму усі об'єкти заносяться в буфер у порядку близькості кожного з них до камери. Після цього виводиться перший об'єкт. Одразу за ним представляється наступний об'єкт, кожен піксель якого перевіряється для недопущення перезапису клітинки пам'яті у стек повного зображення. Алгоритм обходить таким чином кожен піксель зображення об'єкта та записує у стек тільки ті елементи «картинки», які не будуть перезаписувати інформацію у стек.

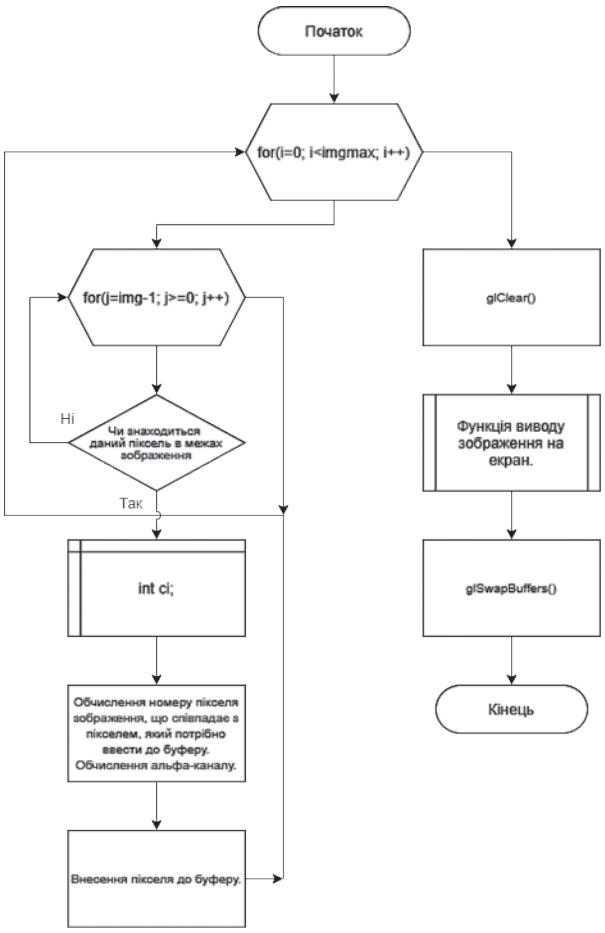

Рис. 4. Алгоритм зворотного рендерингу

Ця функція виконує зворотний алгоритм представлення зображення на екрані. Вона працює із зображеннями у зворотному напрямку, від найближчого до найдальшого та виводить повну картину на екран. Береться найближче зображення, алгоритм проходить по кожному пікселю та виводить його на екран, заносячи в масив значення його альфа-каналу. Якщо це значення не дорівнює нулю, буде обчислено це значення для наступного пікселя у тій самій точці. Після завершення проходу, переходимо до наступного зображення на черзі та повторюємо процес, при цьому оминаючи вже представлені частини зображення. Таким чином, відбувається економія часу на операції потрібні OpenGL на виведення зображення на екран.

```
void display_inverted(){
    unsigned short shift[scrwidth][scrheight]={0};
    int xsh=0;
    int ysh=0;
    glBegin(GL_POINTS);
    for(int i=img; i>=0; i--){
        for(int y=0; y<pic[i].h; y++){
            bool flag=false;
            for(int x=0; x<pic[i].w; x++){
                xsh=pic[i].x+x;
                ysh=pic[i].y+y;
                while(shift[xsh][ysh]>0){
                    x+=shift[xsh][ysh];
                    xsh=pic[i].x+x;
                    if(x>=pic[i].w){
                        flag=true;
                        break;
                    }
                }
                if(flag){
                    break;
                }
                    draw_alpha(xsh, ysh, i);
                shift[xsh][ysh]=pic[i].w-x;
            }
        }
    }
    glEnd();
}
```

Рис. 5. Фрагмент коду програми

Після закінчення проходу по всьому буферу, повний кадр виводиться на екран. Для підвищення ефективності перевірки, у матрицю цілих ненульових чисел, сторони якої відповідають розмірам програмного простору, заносимо значення зміщення, що дозволить перемістити лічильник поточного пікселя об'єкта на вільне місце у стеку зображення. Це дає змогу пропускати вже оброблені області стеку, які не потребують змін, та пришвидшить візуалізацію кадрів.

Таке перенесення лічильника надає змогу збільшити ефективність роботи алгоритму. У залежності від розмірів та області, яку займає поточний об'єкт в програмному полі, можливо побачити пришвидшення обробки наступних.

При виконанні аналізу роботи алгоритмів, бралася до уваги можливість створення одного й того ж самого зображення із використанням стандартного та зворотного алгоритмів, для створення ідентичного результуючого зображення.

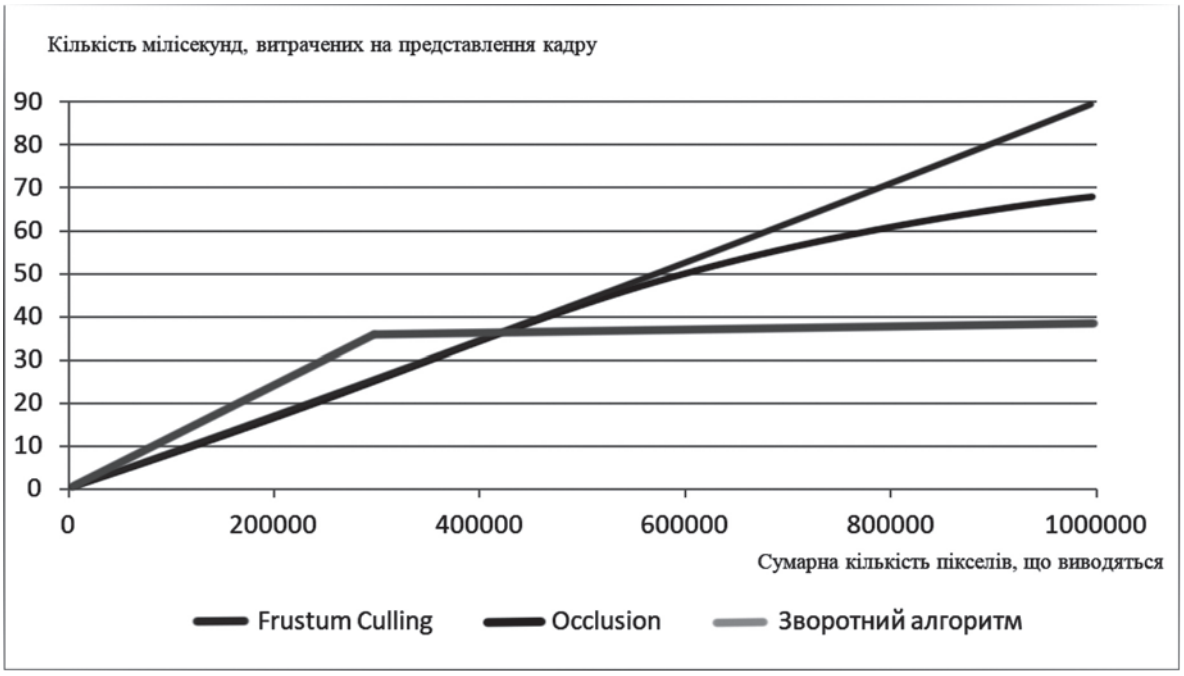

Рис. 6. Порівняльний графік зворотного алгоритму та стандартним алгоритмом з точки зору використання ресурсів за різної кількості виведених пікселів

Суммарна кількість пікселів об'єктів, що виводяться на поле 640 на 480 пікселів	Час виводу одного кадру стандартним алгоритмом, мс	Час виводу одного кадру зворотним алгоритмом, мс
302000	26,667	36,350
604000	52,950	36,517
906000	76,667	36,983

Рис. 7. Порівняльна таблиця ефективності стандартного та зворотного алгоритмів

Отримані результати роботи зворотного алгоритму показали ефективність даного підходу як при малій кількості об'єктів у програмному полі, так і при великій їх кількості. При цьому, зворотний алгоритм має значну перевагу над стандартним з точки зору витраченого часу на проведення операції зміни кадру при умові, що кількість перезаписаних пікселів стандартним алгоритмом була більша за ~30% від їх сумарної кількості на площині дисплея.

Як видно із таблиці, кількість часу використаного на візуалізацію програмного поля зворотним алгоритмом майже стала із підвищенням кількості об'єктів та їх складності. Додатковий час відводиться на перевірку полів та перенесення лічильника, що займає набагато менше часу за перезапис візуальної інформації у стеку.

Висновки. Підсумовуючи вищенаведене, зворотний алгоритм є значно більш ефективним і швидким за стандартні алгоритми. Завдяки його використанню може підвищуватись швидкодія і відбуватись раціональне використання графічного ресурсу системи. З огляду на це, зворотний алгоритм є доцільним для використання щодо програм, які виводять велику кількість об'єктів, що перекривають частини інших.

Список літератури:

1. Foley J.D. Computer graphics: Principles and practice. URL: https://www.amazon.com/Computer-Graphics-Principles-Practice-3rd/dp/0321399528/ref=dp_ob_title_bk.

2. Mahrsncer S. Fundamentals of Computer Graphics. URL: https://www.amazon.com/Fundamentals-Computer-Graphics-Steve-Marschner/dp/1482229390/.

3. Несук О.О., Потапова К.Р. Зворотній алгоритм виведення зображення *Priority directions of science and technology development* : Abstracts of the 7th International scientific and practical conference. SPC "Sci-conf.com.ua". Kyiv, Ukraine, 2021. P. 360–365. URL: https://sci-conf.com.ua/vii-mezhdunarodnaya-nauchno-prakticheskaya-konferentsiya-priority-directions-of-science-and-technology-development-21-23-marta-2021-goda-kiev-ukraina-arhiv/.

4. Mathematics for Computer Graphics (Undergraduate Topics in Computer Science). URL: https://www.amazon.com/Mathematics-Computer-Graphics- Undergraduate-Science/dp/1447173341/.

5. Lnegyel E. Mathematics for 3D Game Programming and Computer Graphics. Third Edition. URL: https://www.amazon.com/Mathematics-Programming- Computer-Graphics-Third/dp/1435458869/.

Nesuk O.O., Potapova K.R., Tarasenko-Klyatchenko O.V. ON THE FEATURES OF DISPLAYING GRAPHIC OBJECTS TAKING INTO ACCOUNT A HARDWARE ACCELERATED APPROACH

This article presents the results of research on the process of rendering images, namely the speed of their reproduction. Today, computer graphics are the main technology in digital photography, film, video games, mobile applications and many specialized programs. A large number of special hardware and software have been developed, and the displays of most devices are controlled by computer graphics hardware. This is a huge area of computer science.

There is currently a standardized rendering algorithm that represents objects in space in an image. The created image is displayed on the screen of the device, visualizing graphic information. Objects are drawn on a graphical representation in an order that depends on the distance of the object to the observer, starting from the farthest. Each subsequent object is superimposed depending on its position in space. These objects overlap others, or may be out of sight, wasting the graphics resource by overwriting the bit information of the pixels in the stack. There are various methods that allow you to avoid some of the GPU operations performed "in vain". Such methods are, for example, the algorithms frustum culling and occlusion. However, even with these approaches, we still lose a share of the graphics resource (GPU operations).

To ensure high performance and solve the problem of loss of graphics resources, an algorithm for reverse image presentation was designed. With to this approach, objects in computer space are rendered in order, starting from the nearest object and ending with the farthest. In this case, each subsequent object is cut along the contour of the already presented part of the full image. This minimizes graphics resource loss by overwriting the pixels in the stack. Thus, each pixel of the image is recorded only once, saving on the number of performed operations.

Key words: *raster graphics, rendering, frustum culling, occlusion, inverse algorithm, overdraw.*