Multicore programming project
Quake III algorithm implementation in CUDA

Miłosz Kulczycki
Przemysław Ryś

Winter semester 2022/2023

1 Introduction

1.1 Motivation

The inverse square root of a floating point number is used in calculating a normalized vector. Programs can use normalized vectors to determine angles of incidence and reflection. 3D graphics programs must perform millions of these calculations every second to simulate lighting. When the code was developed in the early 90', most floating point processing power lagged the speed of integer processing. This was troublesome for 3D graphics programs before the advent of specialized hardware to handle transform and lighting.

The length of the vector is determined by calculating it's Euclidean norm: the square root of the sum of squares of the vector components. When each component of the vector is divided by that length, the new vector will be a unit vector pointing in the same direction. In a 3D graphics program, all vectors are in three-dimensional space.

$$\|\vec{a}\| = \sqrt{v_1^2 + v_2^2 + v_3^2} \tag{1}$$

$$\hat{\boldsymbol{v}} = \frac{\vec{v}}{\|\vec{v}\|} \tag{2}$$

1.2 Source

Quake Engine was designed by ID Software, in 1996. It was precursor of 3D graphic in games, and unlike Wolfenstine 3D which only was an mimic of true 3D, Quake took us to truly 3D game which created path for other first person shooters, but to achieve all of this developers had to deal with a lot of problems. Main problem of developing 3D game in 90‘ was optimisation, because unlike today where we have 3D support in almost every component, developers in those times had to create it all from scratch.

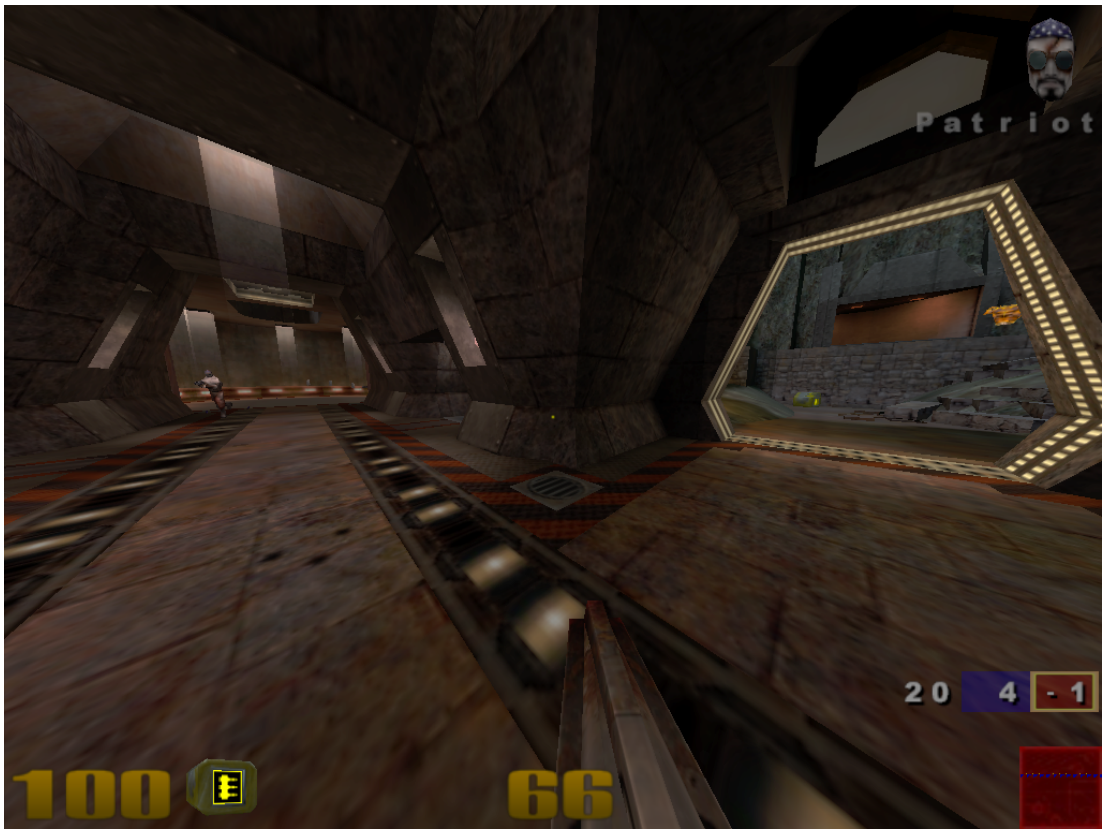

Fig. 1: Screenshot from Quake III Arena, year of publication 1999.

1.3 Algorithm

Fast inverse square root, is an algorithm that estimates $\frac{1}{\sqrt{x}}$ in a short time with very little execution error. This operation is used to normalize a vector, such as scaling it to length 1. This operation is needed by computer graphics programs which use inverse square roots to compute angles of incidence and reflection for lighting and shading. first appeared in the game Quake III Arena (Figure 1). The algorithm only started appearing on public forums in 2002. Computation of square roots

usually depends upon many division operations, which for floating point numbers are computationally expensive. The fast inverse square generates a good approximation with only one division step.

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point
bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the f—k?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this
can be removed

    return y;
}
```

Fig. 2: The code above is a quick implementation of the inverse square root from Quake III Arena, stripped of the C preprocessor directives, but containing the exact original comment text

The key to this algorithm is the magic number 0x5F3759DF, which has been chosen to minimize the approximation error. For single-precision floating point numbers, there is now a full mathematical analysis for determining the magic number.

1.4 Newton Iteration

Starting from $y = \frac{1}{\sqrt{x}}$ we can define a new function

$$f(y) = \frac{1}{y^2} - x \tag{3}$$

The approximation yielded by the earlier steps can be refined by using a root-finding method, a method that finds the zero of a function. The algorithm uses Newton's method: if there is an approximation y_n for y, then a better approximation y_{n+1} can be calculated by taking $y_n - \frac{f(y_n)}{f'(y_n)}$, where $f'(y_n)$ is the derivative of f(y) at y_n and equals:

$$-\frac{2}{y^3} \tag{4}$$

Treating y as a floating-point number, $y = y * (\text{threehalfs} - x/2 * y * y)$; is equivalent to

$$y_{n+1} = y_n \left(\frac{3}{2} - \frac{x}{2} y_n^2 \right) = \frac{1}{2} \cdot y_n \left(3 - x y_n^2 \right) \tag{5}$$

By repeating this step, using the output of the function y_{n+1} as the input of the next iteration, the algorithm causes y to converge to the inverse square root. For the purposes of the Quake III engine, only one iteration was used. A second iteration remained in the code but was commented out.

2 Code Analysis

Our program consisted in comparing the times of performing the inverse root calculation with different methods. We filled three large-sized arrays with numbers representing the squares of coordinates as input, so that we do not have to exponentiate them, which does not affect the overall project, because these are only input data. We used the code from quake unchanged. As mentioned above, it uses both floating point operations and one Newton iteration to get a better approximation. We also used naive functions present in standard C libraries in order to use them as one of the methods and compare the call with the code of the project. In addition, we used the Newton algorithm itself without floating point operations. All methods were performed both on the CPU, where loops were used, traversing all table indexes, and on the GPU, where the loops are used to execute the program by individual threads on the graphics card. For this purpose, we used the syntax of the CUDA language based on C/C++ languages. The following table 3 summarizes the aggregated execution results for a given

approach, i.e. execution time and error, which mainly concerned approximation-based algorithms. Of course, each of the algorithms received arrays of the same dimension. the following call is for a call for N=10000000.

	CPU		GPU	
	Errors	Time	Errors	Time
Naive function	0.000000	46.792	0.000000	22.692
Newton iteration	0.000003	385.099	0.000001	15.380
Quake algorithm	0.000839	150.677	0.000839	6.425

Fig. 3: Program call table.

As we can see in the case of the CPU, the Newton algorithm was characterized by the worst execution time, at the cost of a small error, the execution time was much longer than the naive call. On the other hand, the quake algorithm, which is based on the use of a floating point operation and only one Newtonian iteration, is faster than the Newtonian algorithm itself, but still loses to the naive call. With the GPU, all times were greatly reduced, which we expected when using the multi-threaded method. However, the order of efficiency is reversed in this case, i.e. the quake algorithm turns out to be the most efficient, which is about twice as fast as Newton's iteration alone and about four times faster than the naive call. The error in the calculated value is noticeable only in the fourth decimal place. It should be remembered that in this program we defined the error as the sum of errors in calling after all indexes, which are of the order of thousands. So this is a negligibly small error.

3 Summary

To sum up, the quake algorithm turns out to be really efficient only after using the multi-threaded method, which significantly improves the process of counting inverse roots, and thus, in relation to Quake itself, its optimization. Newton's method as such compared to the naive call is also optimal, but not like the quake algorithm. Taking into account a very small error, it is an algorithm that can be used in a situation where we care about the best optimization.

4 Github

link: `https://github.com/Mishon1235/Quake-III-CUDA`