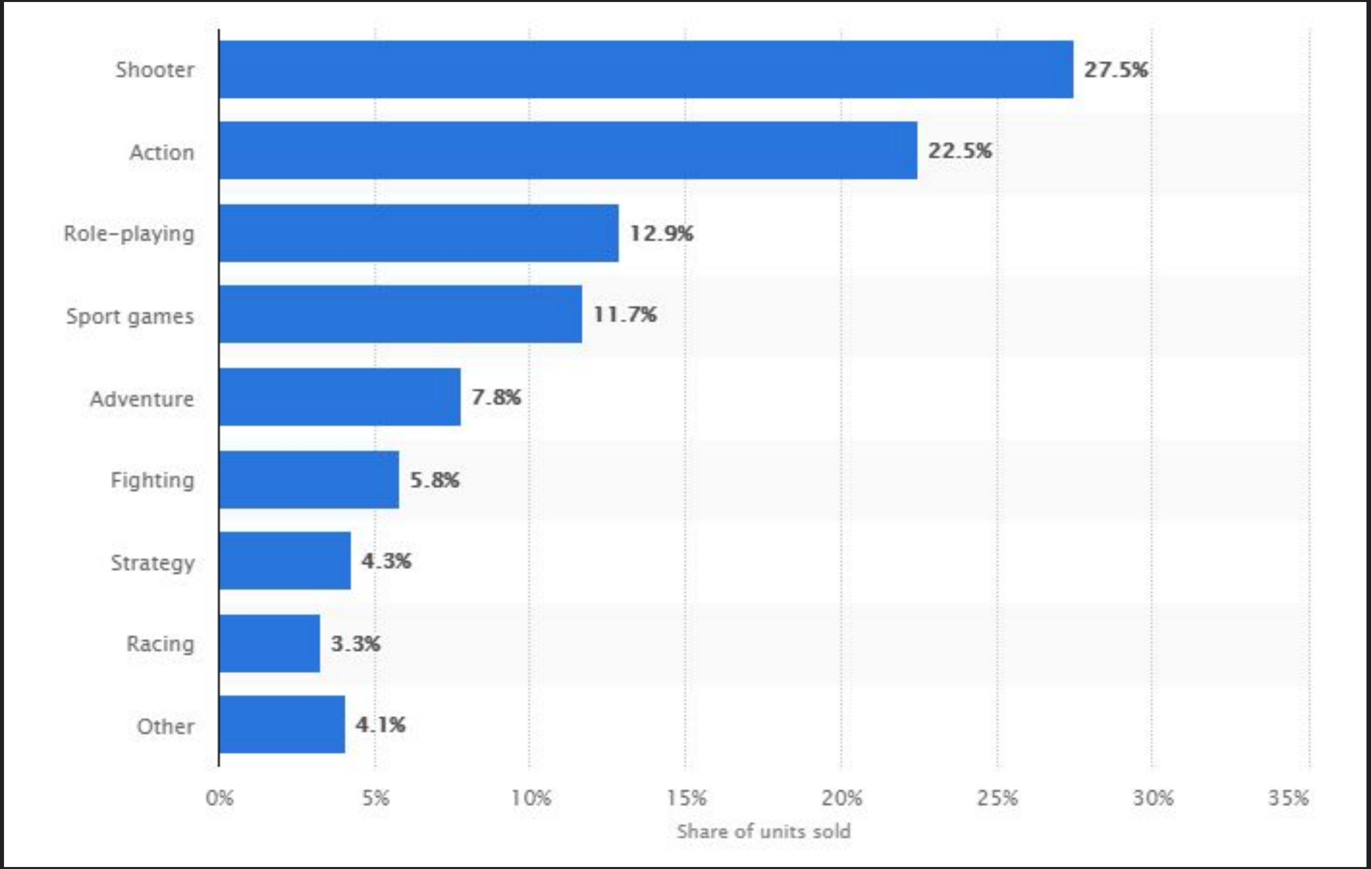
First-person 3d shooters

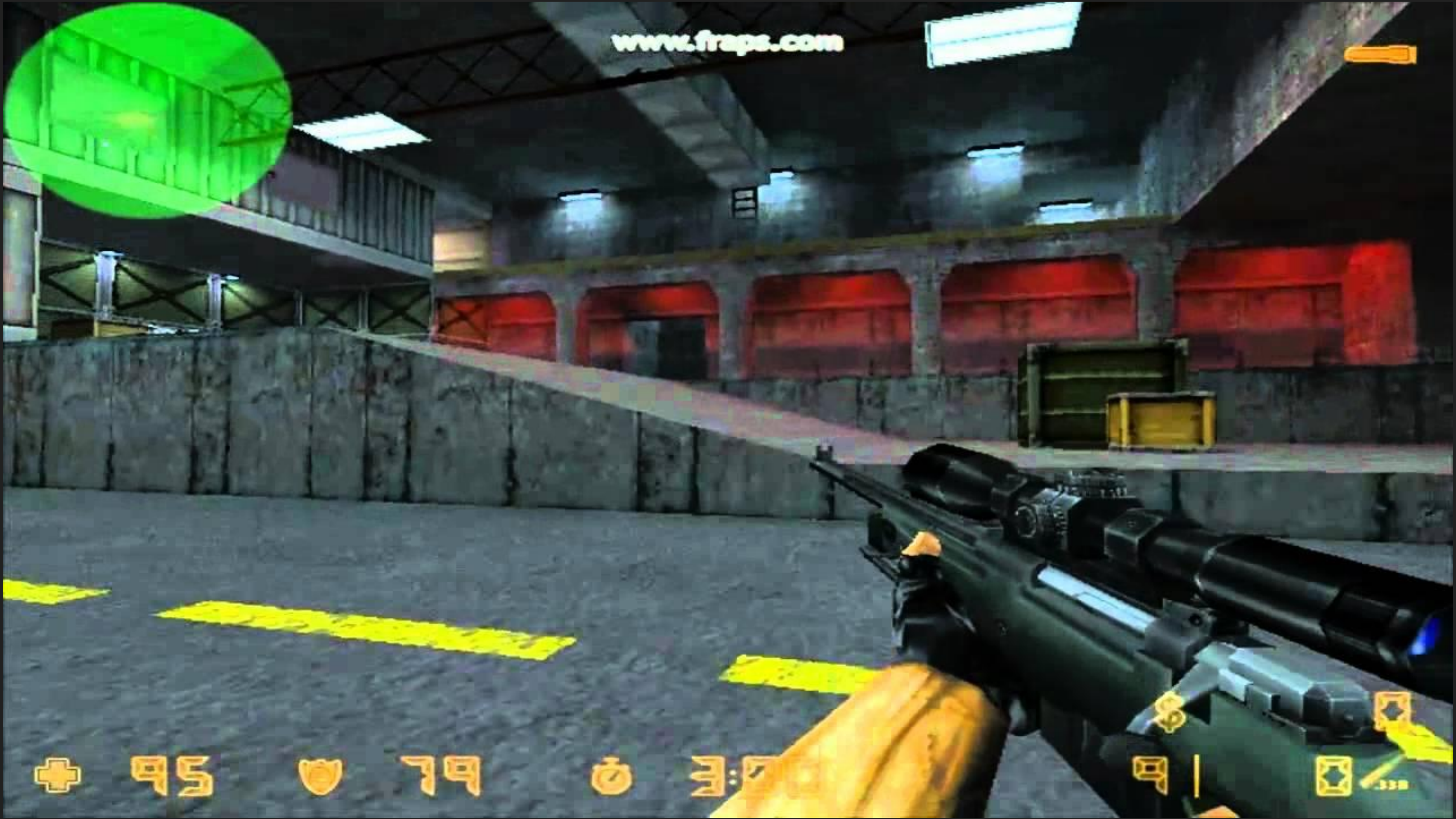
history

By Ershov Gleb

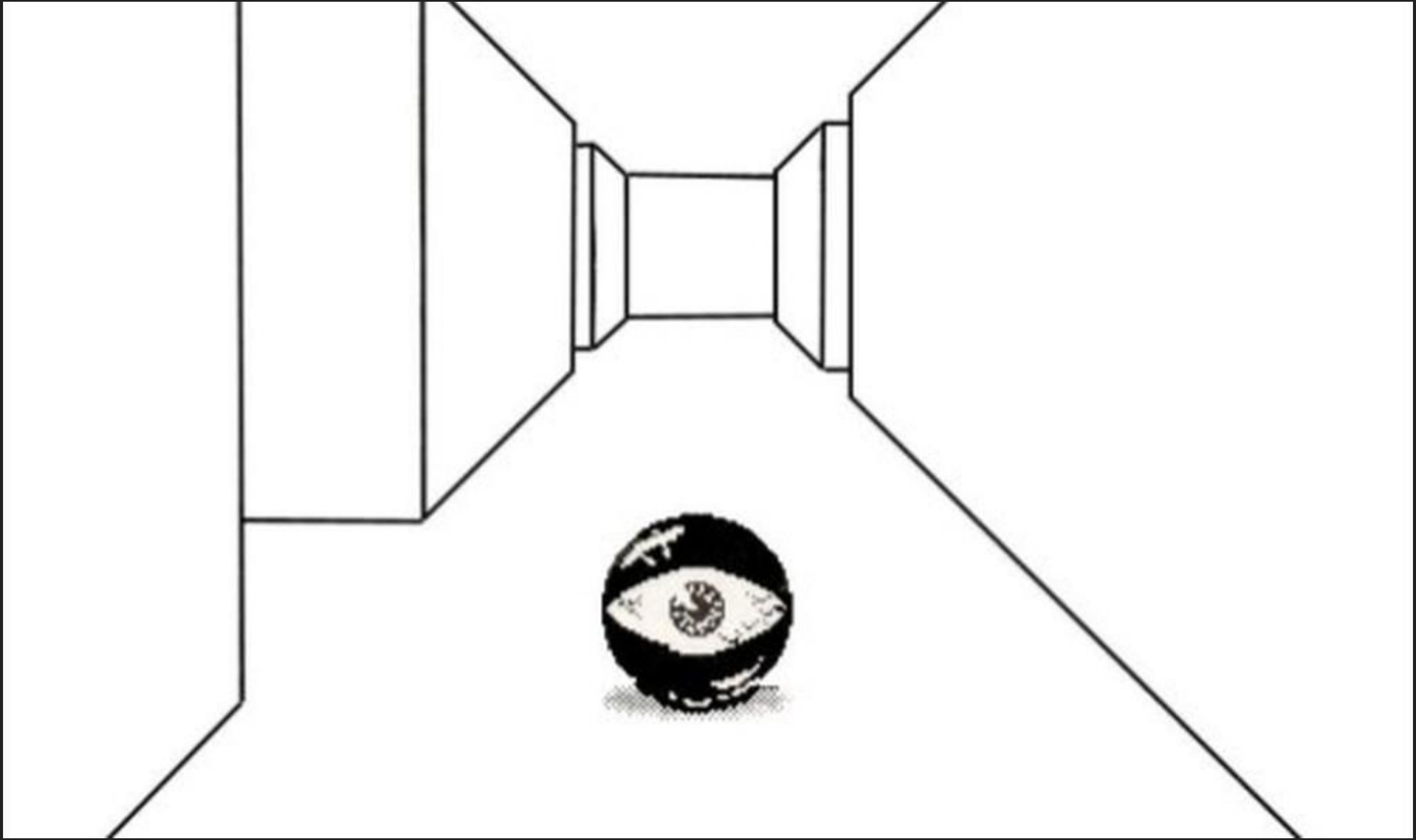


2016 US. market The graph shows the distribution of U.S. video game sales by genre.

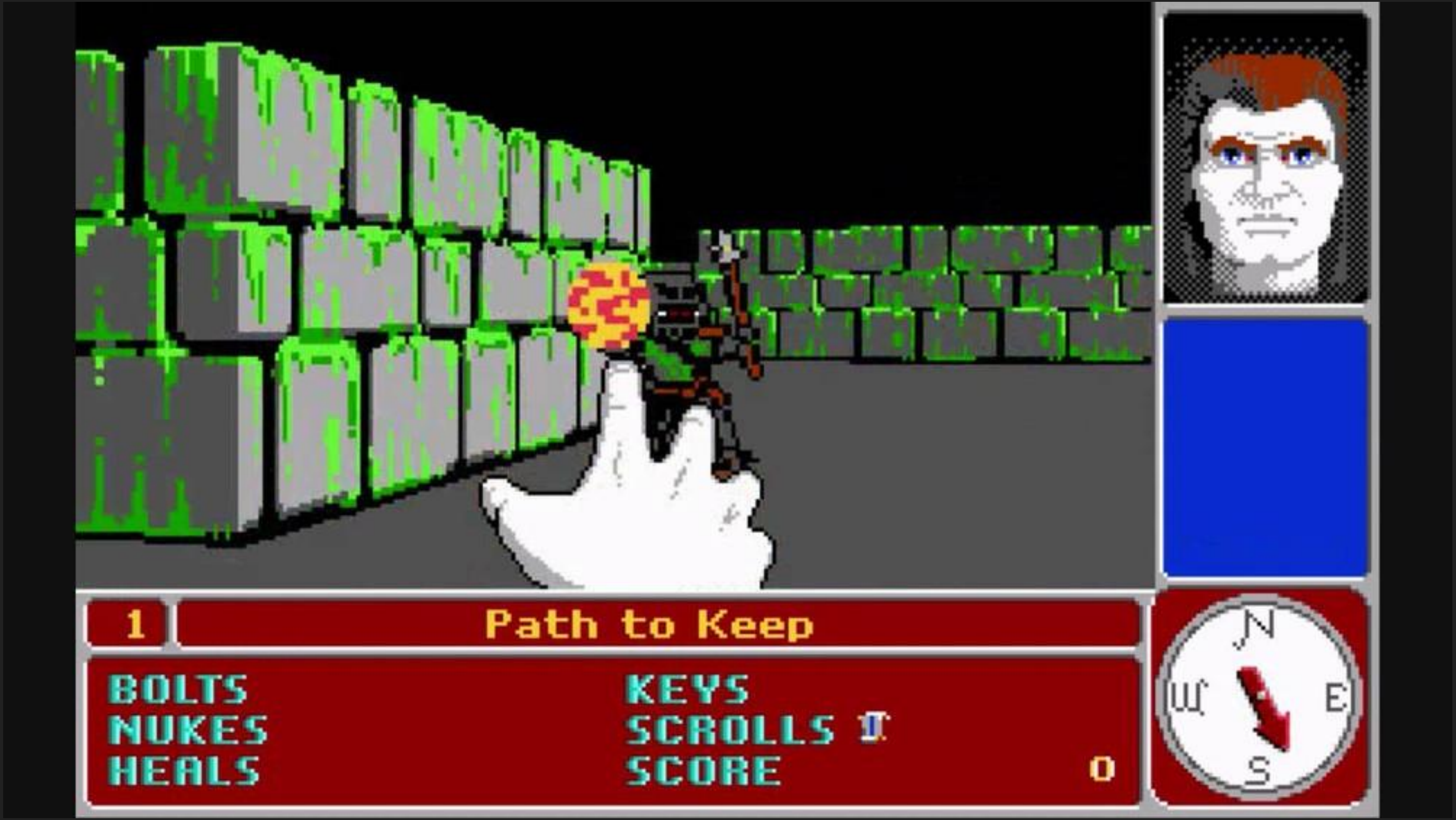
Enemy Front (2014)



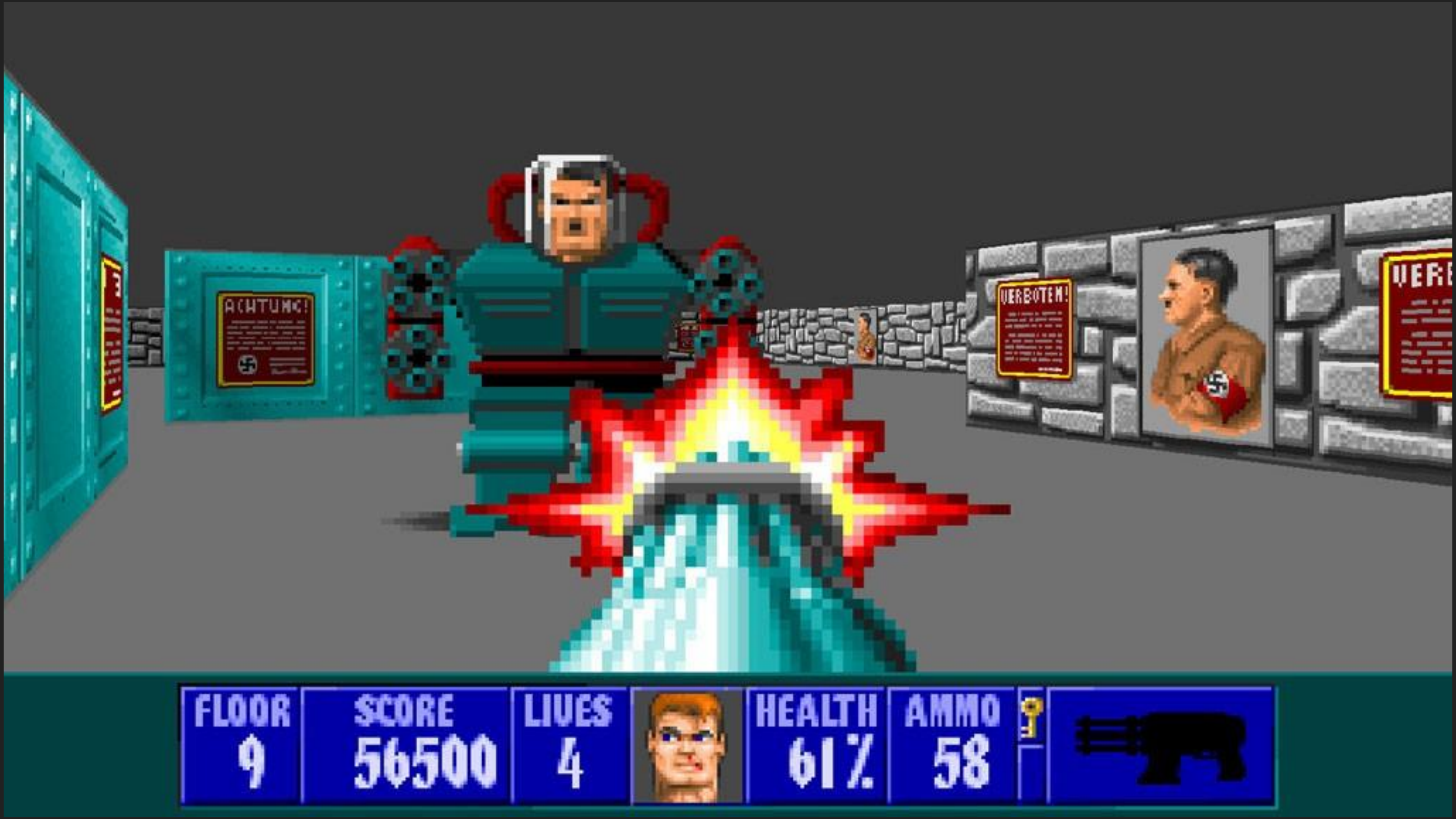
Counter Strike 1.5 (2000)


Maze War (1974)

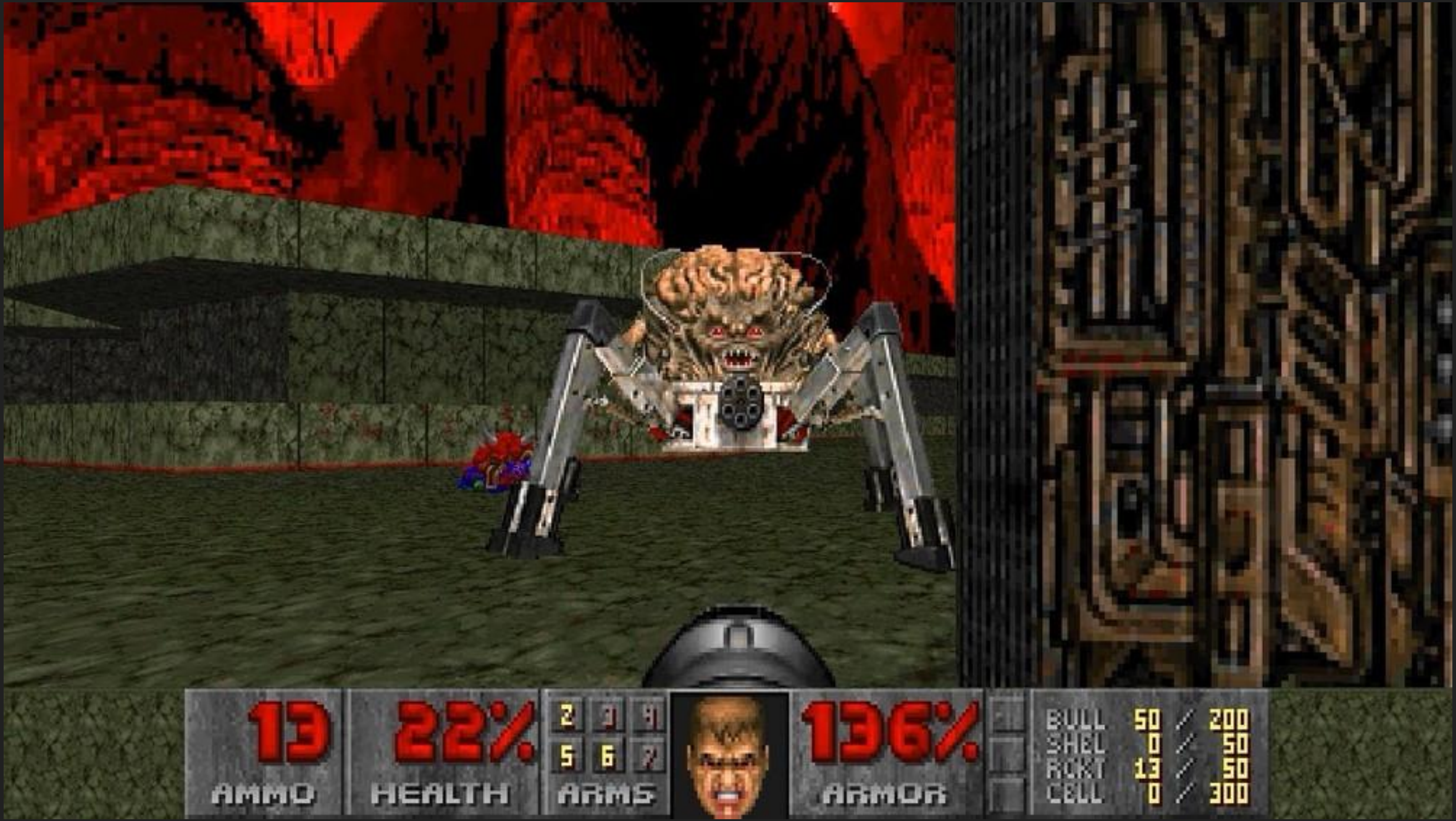
id Software team (1992)



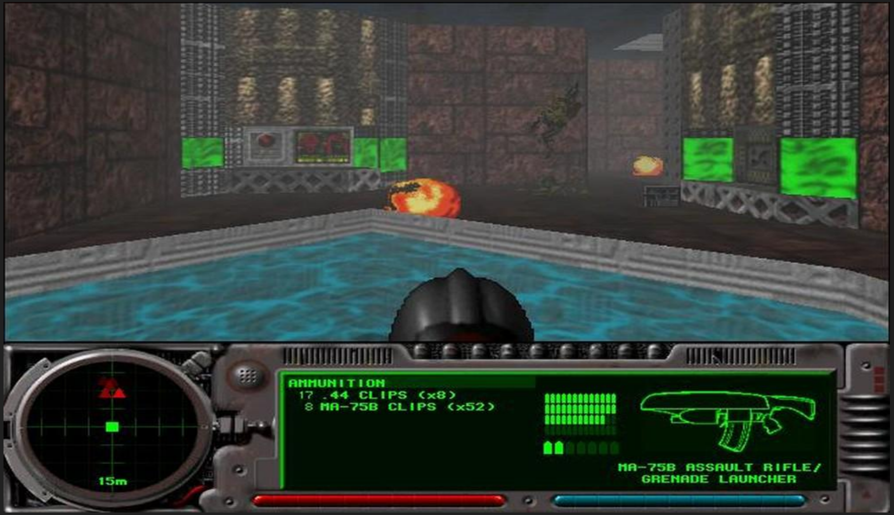
Catacomb 3D (1991)



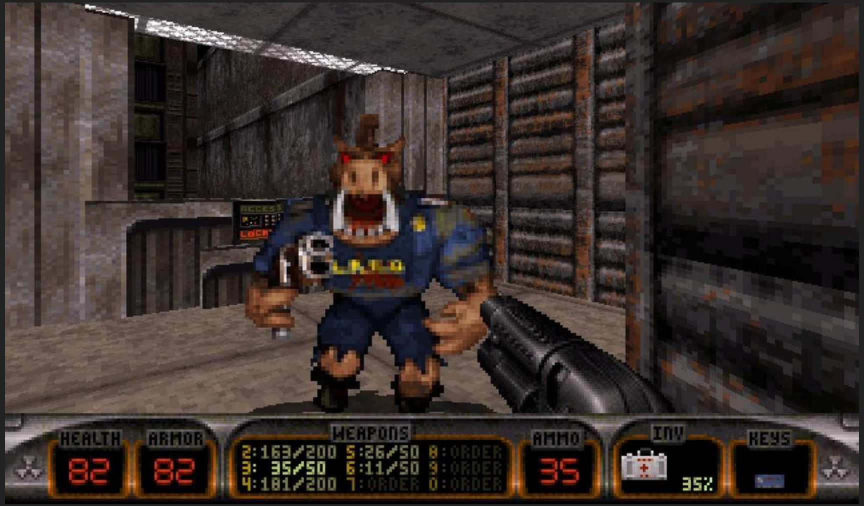
Wolfenstein 3D (1992)


13
AMMO
22%
HEALTH
2 3 4
5 6 7
ARMS
136%
ARMOR
BULL 50 / 200
SHEL 0 / 50
RCKT 13 / 50
CELL 0 / 300

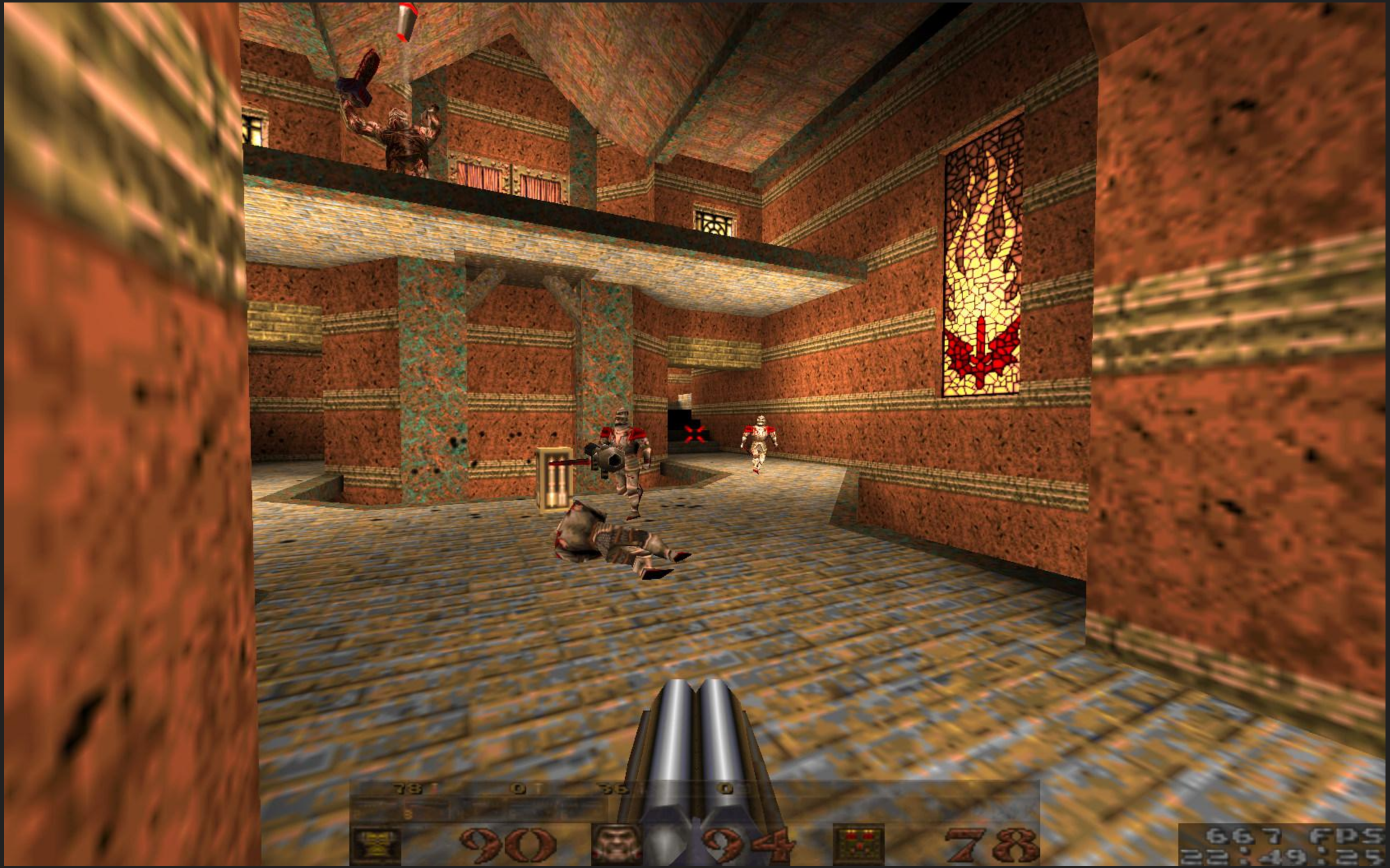
Doom (1993)



Marathon (1994)



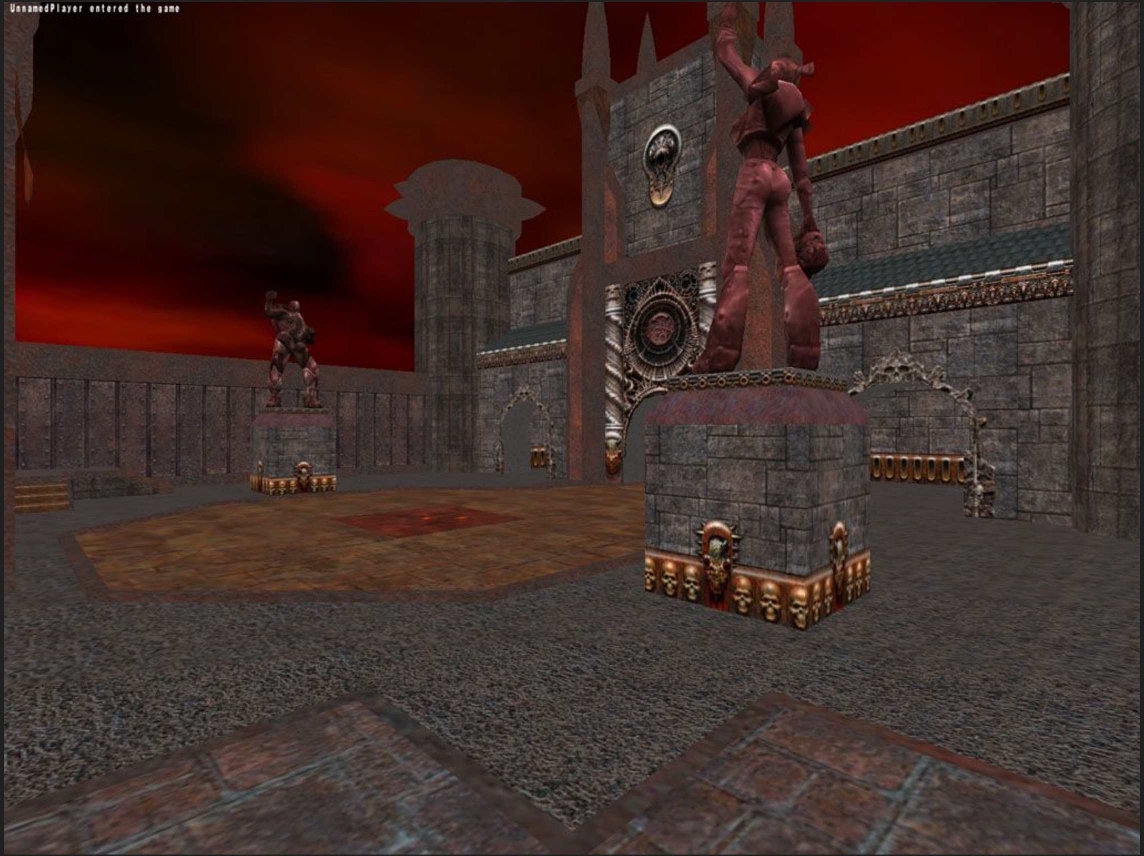
Duke Nukem 3D (1996)



Quake (1996)

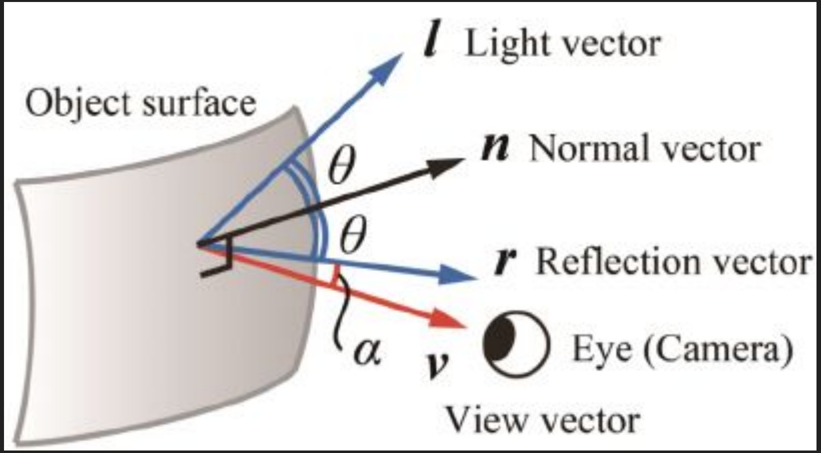
Quake (1996)

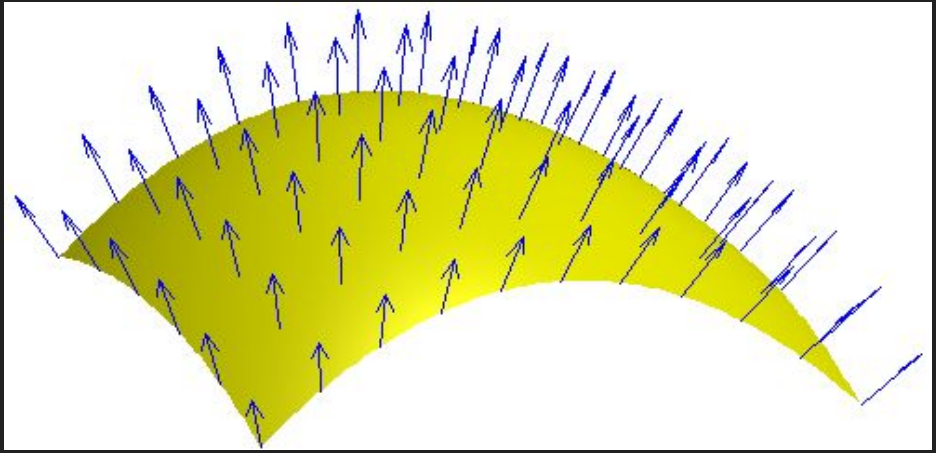
Without shaders


UnnamedPlayer entered the game

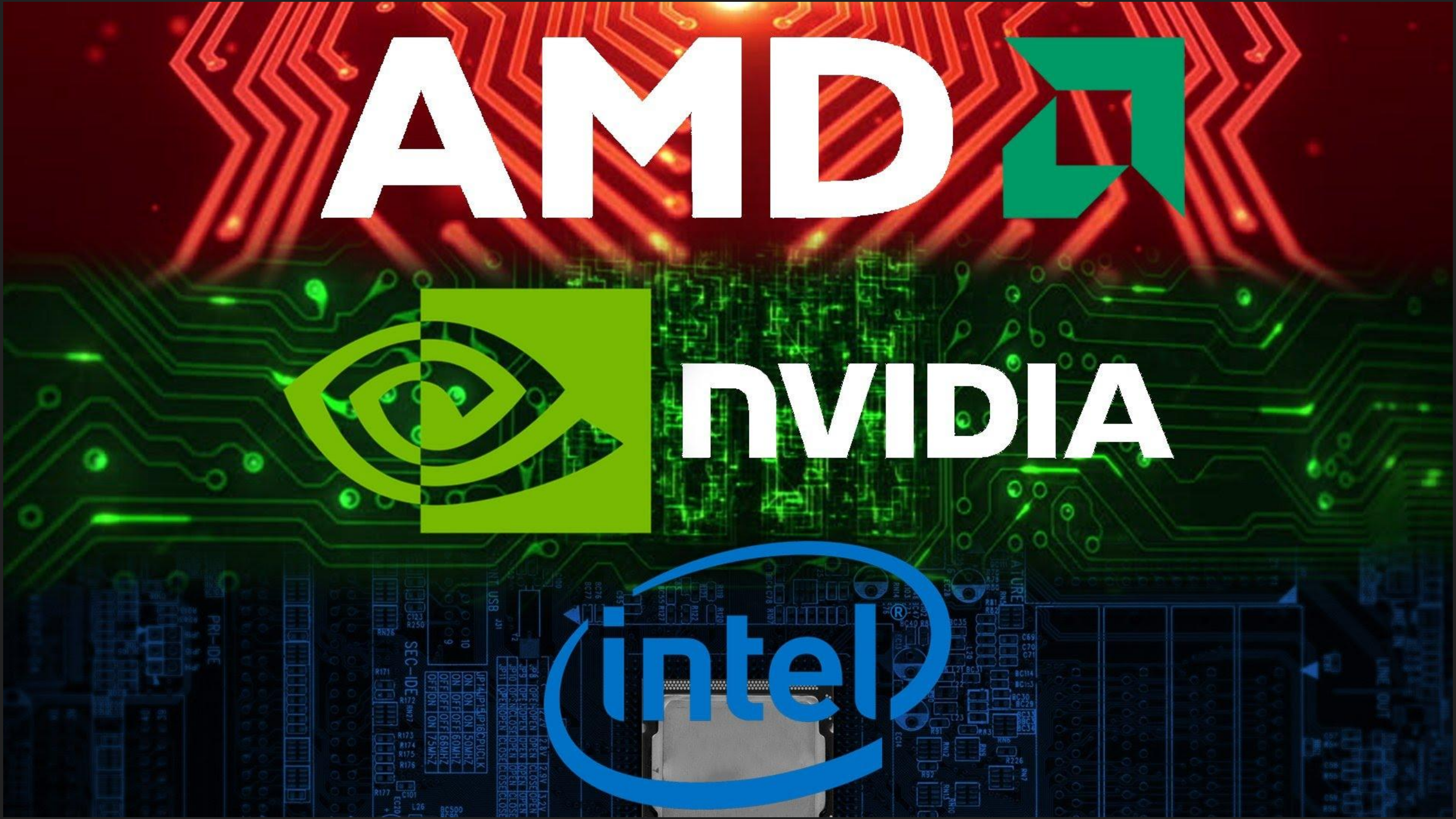
Shaders

Together





Vector normalization problem

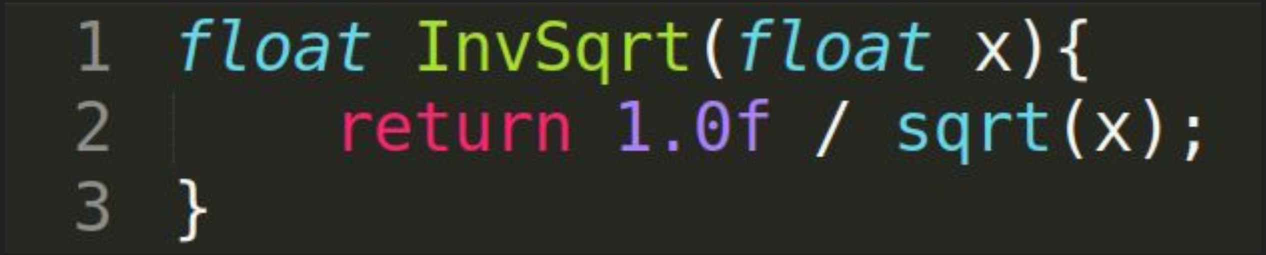


GPU vendors

$$\frac{1}{\sqrt{x^2 + y^2 + z^2}}$$

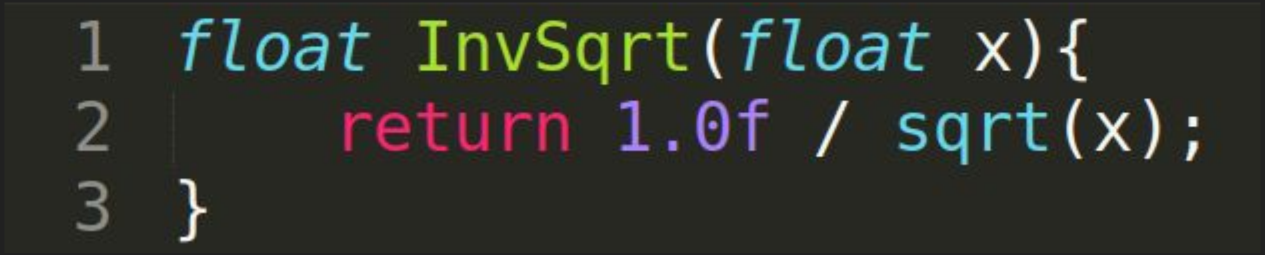
Vector normalization formula

Too simple, isn’t it?



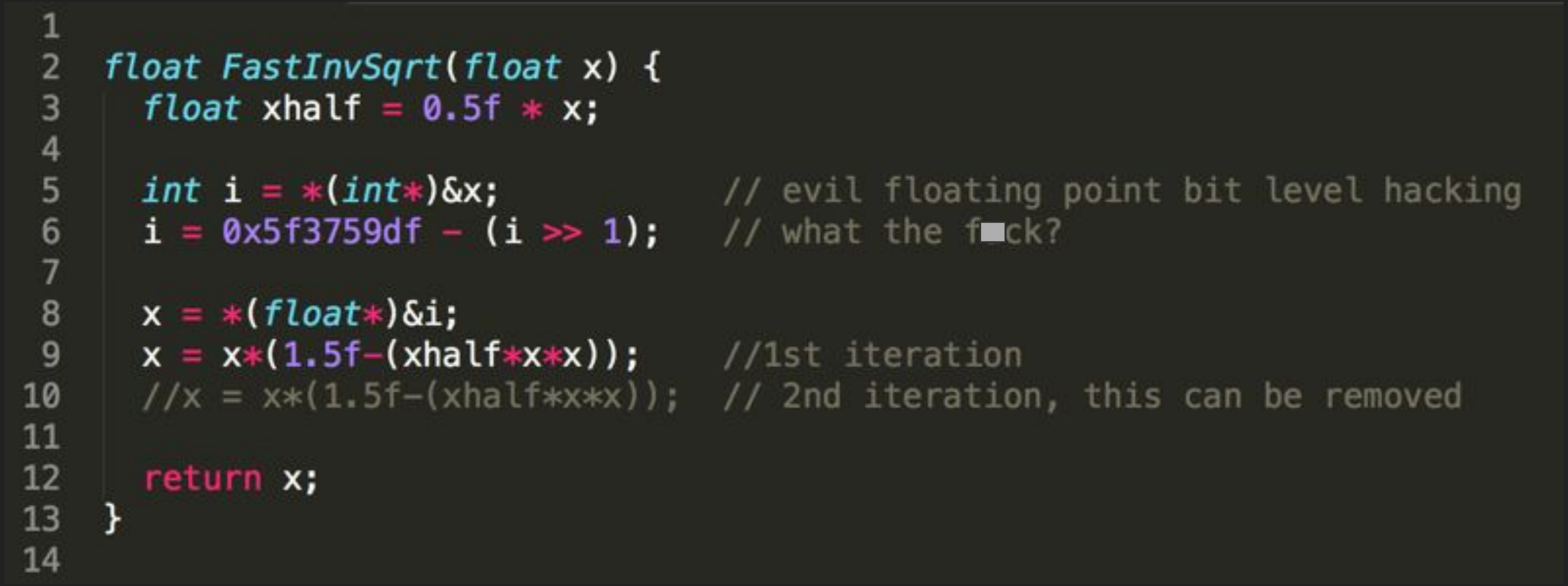
```
float InvSqrt(float x){
    return 1.0f / sqrt(x);
}
```

Too simple, isn’t it?



```c
float InvSqrt(float x){
    return 1.0f / sqrt(x);
}
```

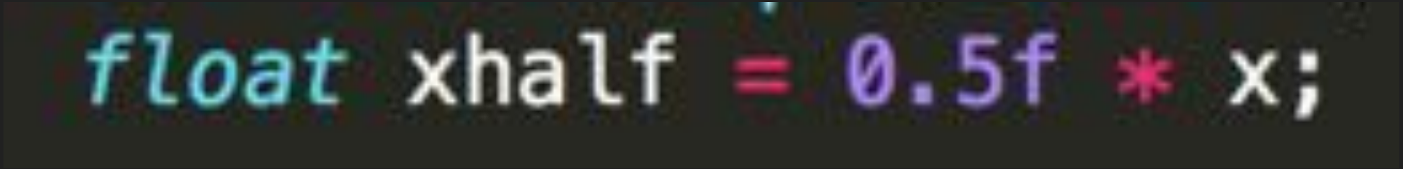
And also too slow



```c
float FastInvSqrt(float x) {
  float xhalf = 0.5f * x;

  int i = *(int*)&x;           // evil floating point bit level hacking
  i = 0x5f3759df - (i >> 1);   // what the f[redacted]ck?

  x = *(float*)&i;
  x = x*(1.5f-(xhalf*x*x));    //1st iteration
  //x = x*(1.5f-(xhalf*x*x));  // 2nd iteration, this can be removed

  return x;
}
```

Code to compute the inverse square root from Quake engine

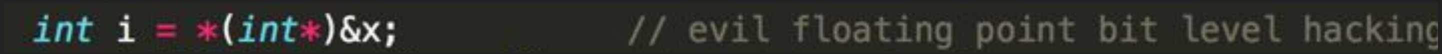
Algorithm



```
float xhalf = 0.5f * x;
```

1. Compute half of input integer

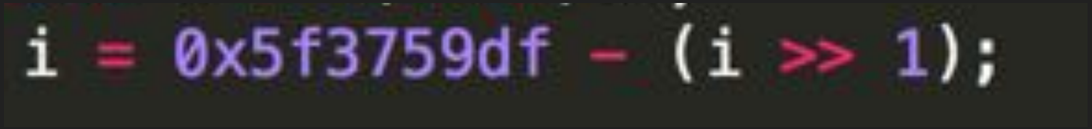
Algorithm



```
int i = *(int*)&x;          // evil floating point bit level hacking
```

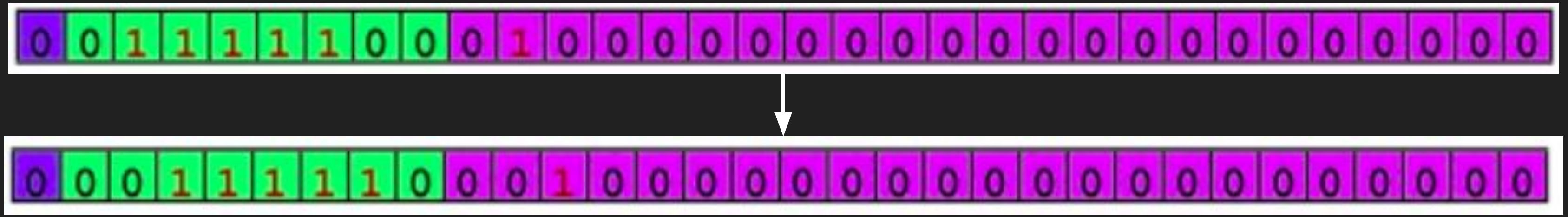
1. Compute half of input integer
2. Cast float to int

Algorithm

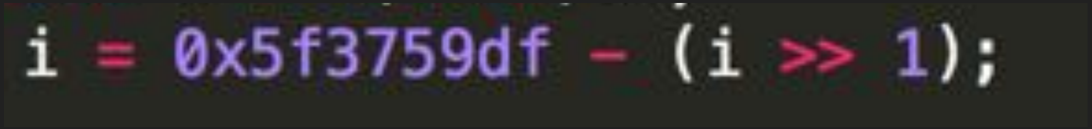


```
i = 0x5f3759df - (i >> 1);
```

1. Compute half of input integer
2. Cast float to int
3. Bitwise shift to the right



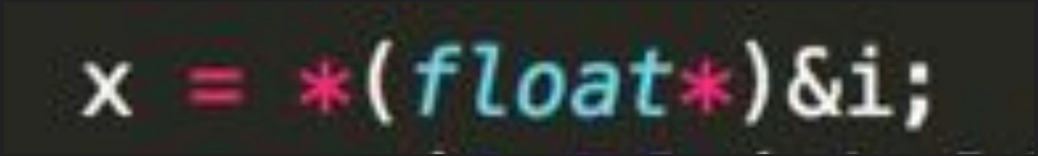
Algorithm



```
i = 0x5f3759df - (i >> 1);
```

1. Compute half of input integer
2. Cast float to int
3. Bitwise shift to the right
4. Subtract result from magic constant 5F3759DF16

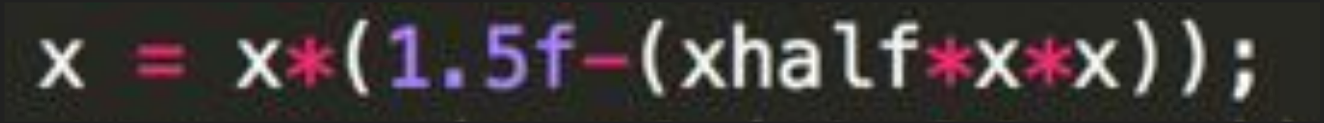
Algorithm



```
x = *(float*)&i;
```

1. Compute half of input integer
2. Cast float to int
3. Bitwise shift to the right
4. Subtract result from magic constant 5F3759DF16
5. Cast result back to float

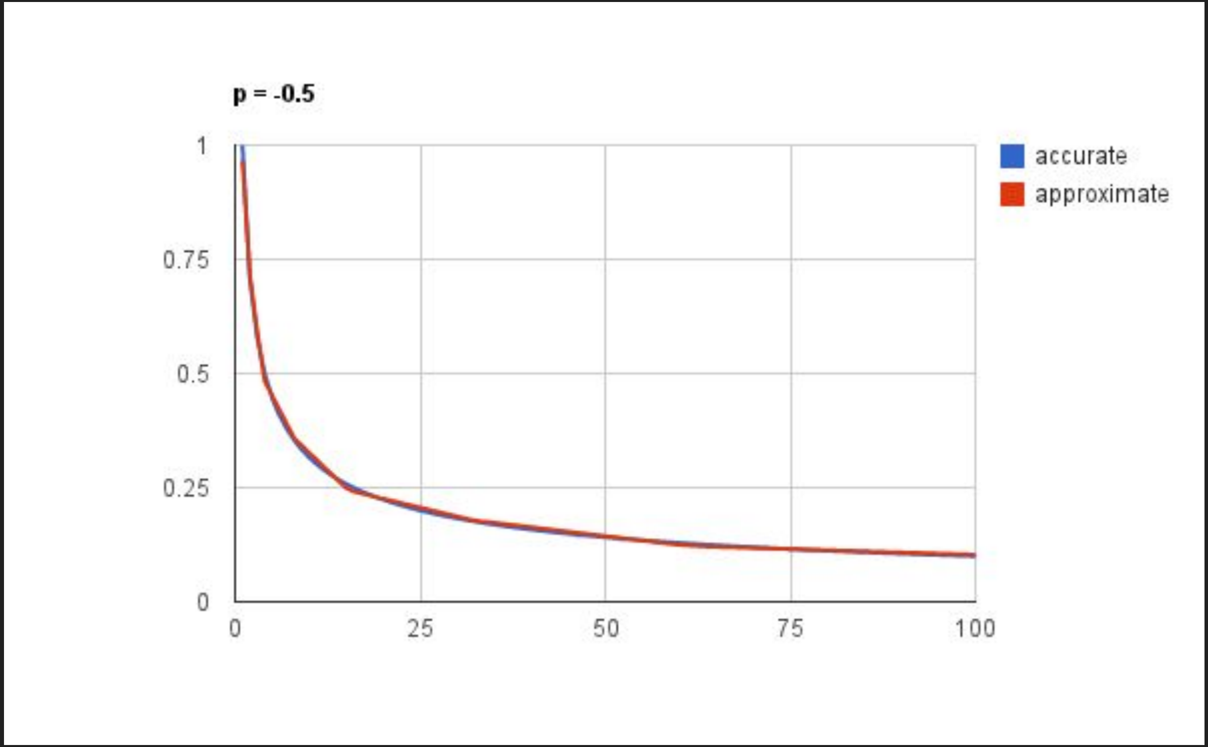
Algorithm



```
x = x*(1.5f-(xhalf*x*x));
```

1. Compute half of input integer
2. Cast float to int
3. Bitwise shift to the right
4. Subtract result from magic constant 5F3759DF16
5. Cast result back to float
6. Make one iteration of Newton method to get more accurace

Accuracy



Speed

3-4 times faster

References

- https://en.wikipedia.org/wiki/Newton’s_method
- http://h14s.p5r.org/2012/09/0x5f3759df.html
- https://www.slideshare.net/maksym_zavershynskyi/fast-inverse-square-root