How to render 256x SSAA anti-aliasing picture quality game very quickly in GPU

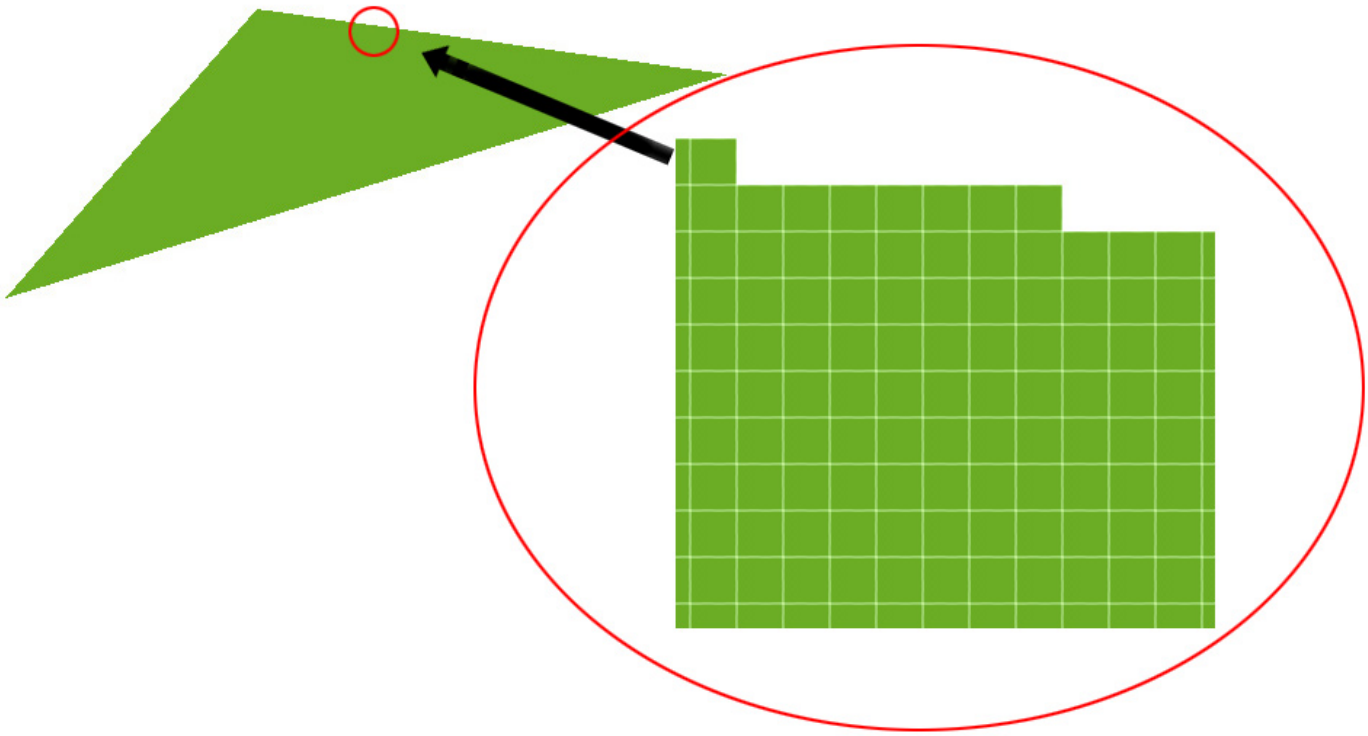
- sulin huang.

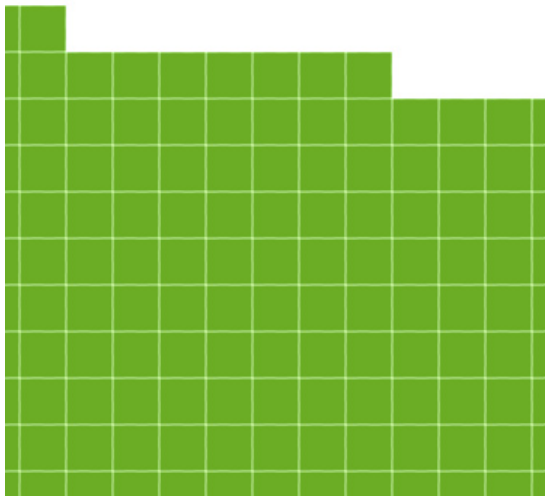
We all know that the game picture is composed of many triangles. Now let's go deep into the bottom triangle rendering and quickly improve the anti-aliasing rendering speed to make it reach or exceed the quality of 256x SSAA (if the color depth of the display supports)

In the traditional rendering method, if you use 256x SSAA images, you need to sample each point of the screen 256 times and average them to get the final pixel color, which is extremely expensive to use the graphics card resources.

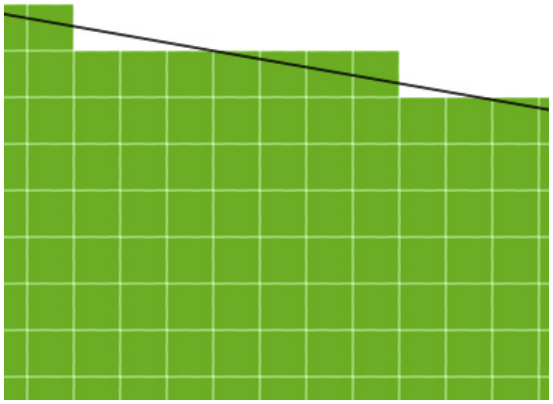
Let's greatly improve it.

We enlarge the image to one pixel at a time.

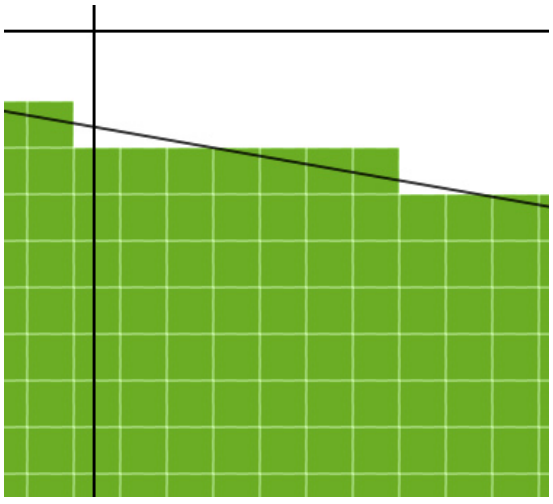




The black line in the figure is the pixel position (floating point, non-integer) that the triangle edge passes through mathematically

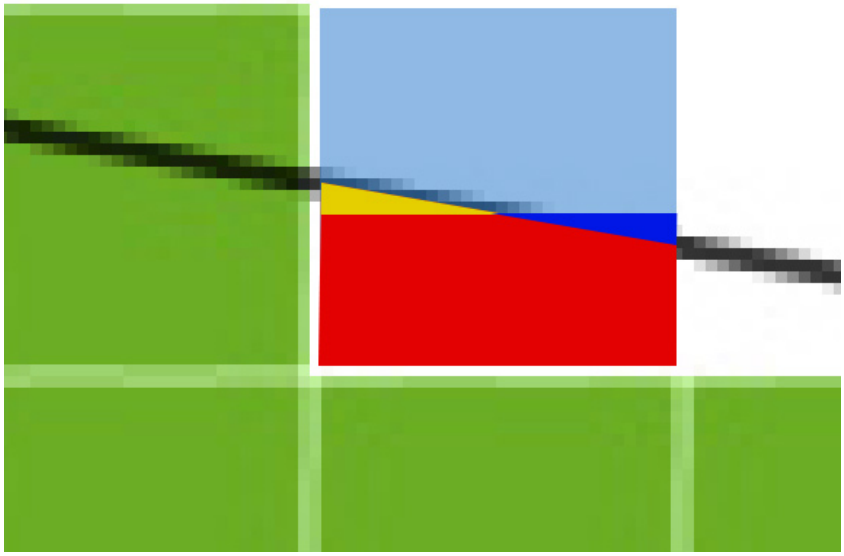


Let's draw a coordinate system for it.



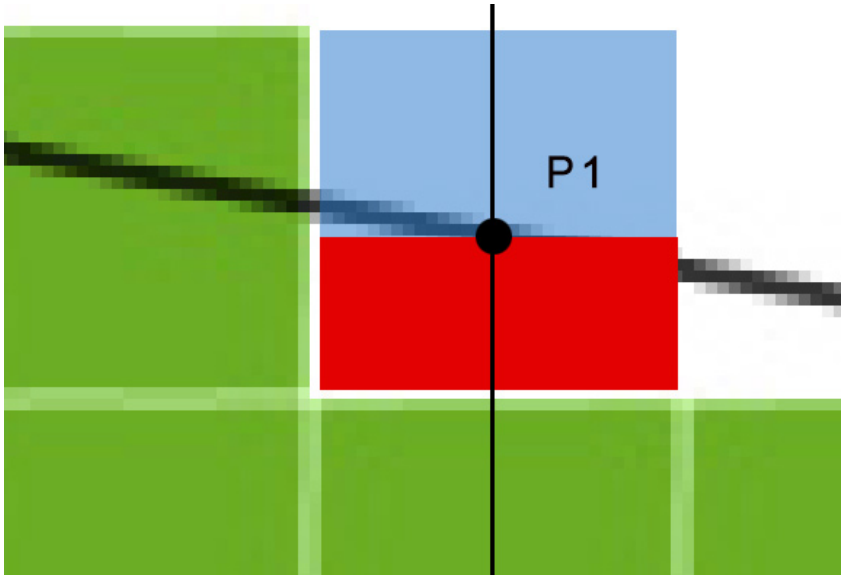
In fact, we should calculate the area of the red area. We can calculate the proportion of the red area to the whole pixel square.

Mathematically, it is easy to strictly prove that the yellow triangle area and the blue triangle area are equal.



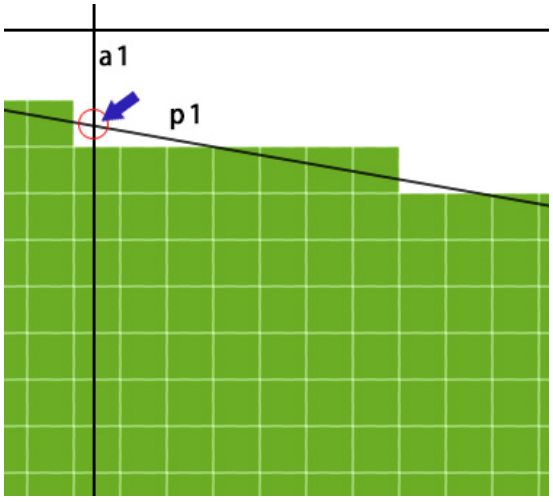
So, in fact, we can calculate such areas.

Because one side of the red rectangle and the light blue square pixel block is equal in length, the ratio is actually the ratio of the rectangle height to 1px, that is, p1 point.



The three vertices of each triangle are known, which means that each side of the triangle is known to us.

Intersection calculation of too simple line segments. We can easily calculate the specific value p1 of y of this diagonal line at this x position. Floating-point must be calculated.

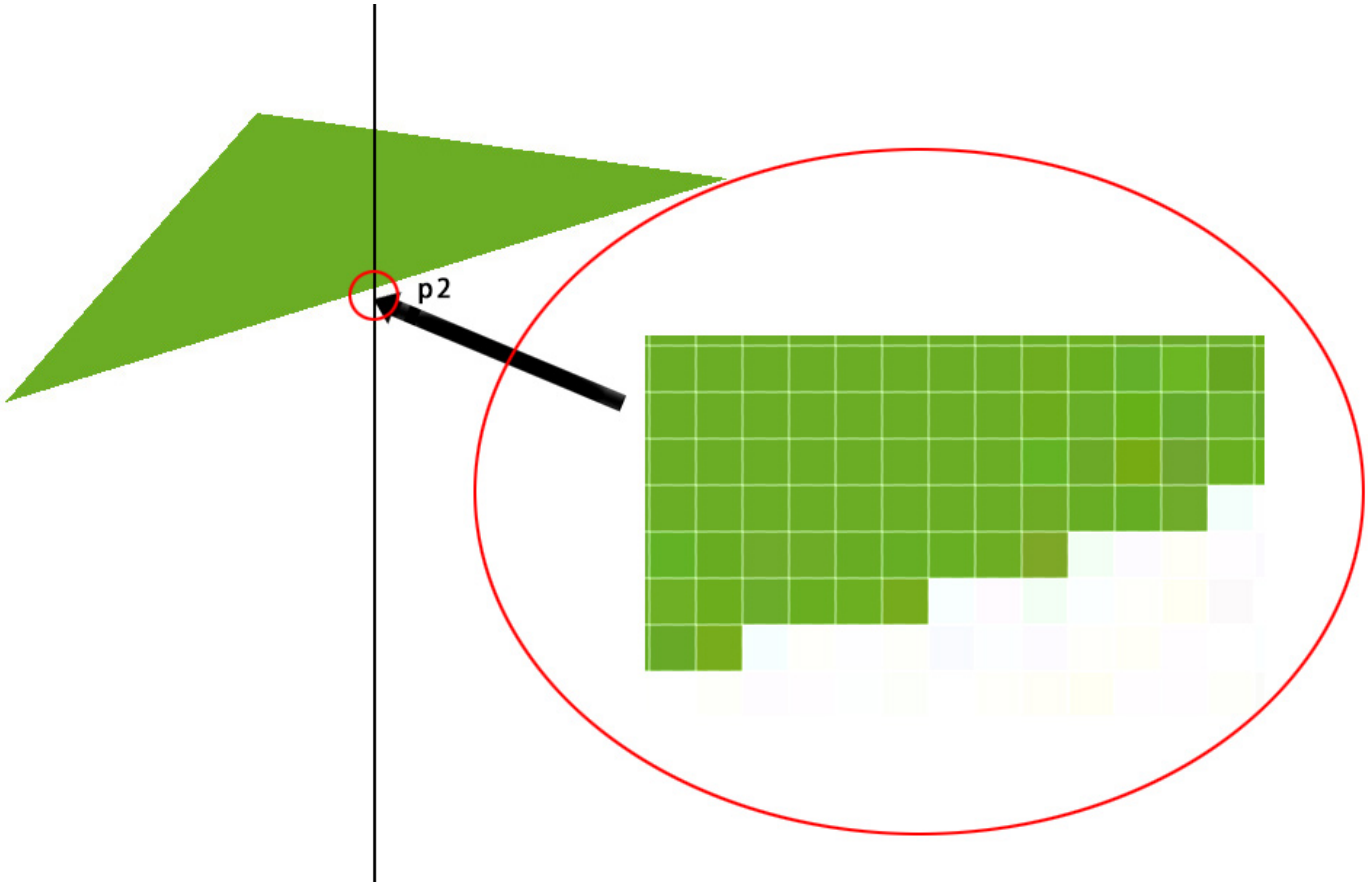


At this point, we get a mathematical floating-point value, assuming 5.643. Pay attention to the decimal part. This is the data we need.In fact (1-0.643)=0.357 is the proportion value we want.

Now let's quantify it according to the level of 0-255 (relative to the color depth of the current display)

At this point, we set up an alpha channel to store it as the two-dimensional data corresponding to the image (we need this data to calculate later in the real rendering

Similarly, there is also a triangle at the bottom of this vertical line. We can calculate the floating point value of its intersection in the same way.

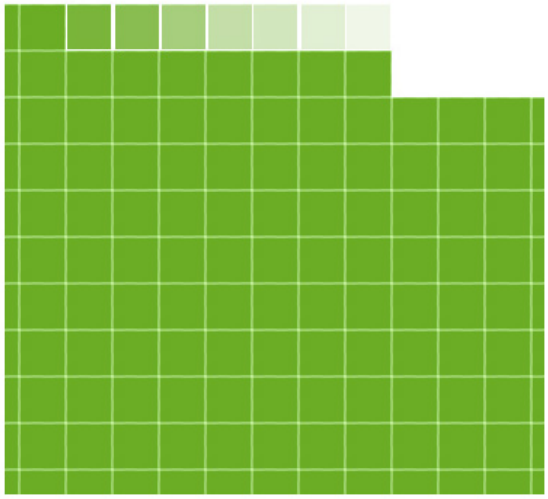


At this point, we get the alpha channel values of all the vertical pixels.

Above the p1 point, the alpha value of all pixel points is 0, and no rendering is required. At point p1 and point p2, all the pixel points directly have an apha value of 255, which requires rendering. Then we get the pixel alpha values of the two edges. (assuming 0.357 and 0.85), map them to the bit storage supported by the current display, assuming 0-255

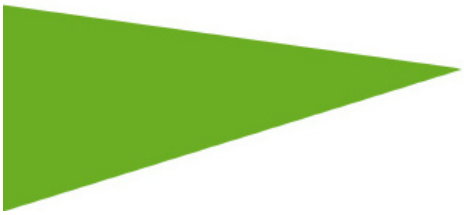
Now for an edge, our calculated value is assumed to be 0.357 0.215. 0143 0.05 0.951 0.751 0.65 0.54 0.45

We can get a stepped alpha value.。

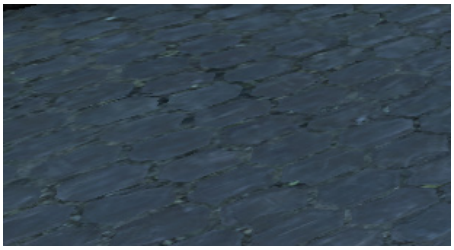


For the vertex pixels of a triangle, we can use the traditional method of sampling multiple points to calculate its transparency (because the number of pixels is very small, this calculation is very low)

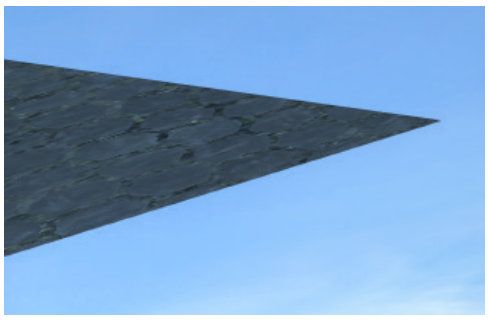
Let's zoom in.



Now, we have an alpha value for each pixel. Using this Apha value, it is easy to calculate the proportion of the final color of the current triangle and the color behind it at the edge of each pixel.



Mix them, and finally:



It is the same for all triangles:

So far, we are finished.

Thank you for reading.