Subject: Re: Raster Control
Posted by mdelfede on Tue, 01 Apr 2008 20:14:47 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Tue, 01 April 2008 21:57Not only it is divided into C/ASM, but both code paths are different.

The ASM version works purely with integers, while the C version with (double) will lead to decimal division at least, if not multiply (I'm not that big C guru to fully understand how that casting to double will propagate to the rest of calculations).

To get maybe identical version in C you may try:

return ((x * y) / z);

I'm not sure it will compile absolutely identically, but I think the chances are very good... feel free to compile into ASM firstly and check the result of GCC ASM (-S command line switch?).

Well, maybe will be the same, maybe not.
The problem solved by Iscale is rounding working with integers.
To have no precision loss, you should have a temporary integer with twice the integer width, whichever it is... OR use some MULDIV advanced algorithm.
Because of that some assembly language has a built-in muldiv instruction.
The big problem is that in C you've no guarantee of integer sizes, you just know that :
char <= short <= int <= longint <= longlongint
so, you can't know what size to use in intermediate calculation.... because of that in Iscale they use double.
In GCC32 you should have int(32), long(32) and long long(64), so using longlong would be ok.... but I guess that in GCC64 the int is 64 byte wide (maybe), so it won't work either.
BTW, nor the asm code will work without some cheks.

IMHO, the right approach would be to typedef some integer sizes in Core.h :

typedef char int8
typedef short int16
typedef int int32
typedef longlong int64

depending on machine, then use

int32 Iscale(int32 a, int32 b, int32 c)

so it's compiler sake to give needed warnings.

Max

typedef