Computer Arithmetic Final Assignment

Professor:Lambert Spangberg

Peyman Pouyan

P.N: 8312306296

sx07pp6@student.lth.se

Assignment1:

A function for inverting from to float to fix:

If the integer part is zero I will truncate the floating point (Assignment 3).But when the integer part is not zero I do like this:

Imagine the number is: 1110011.110101

I first shift the number to left as positions as the value of fb is, for example if fb=2 the shift would be 2 times to left and the number would be: 111001111.0101 then I truncate this number with function FIX and Fix(111001111.0101)_= 111001111.0000 then I shift back the numbers: 1110011.1100 .

I do the same for Integer part and I shift ib positions.

```
function newfix=converter(data, ib, fb)

newfix=data;

if (ib==0)

   newfix =truncate(data,fb);

else
```

```
    %Fraction part Truncation
    newfix=newfix.*(2^fb);
    newfix=fix(newfix);
    newfix=newfix./2^fb;
    %Integer part truncation
    newfix= newfix./(2^ib);
    newfix = newfix-fix(newfix);
    newfix = newfix.*2^ib;
end;
```

Assignment 2:

Testing the converter with different ib and fb:

20.6=10100.1001100110011001

Test : converter(20.6,10,10)= 20.5996

Test: converter(20.6,5,10)= 20.5996

Test: converter(20.6,4,10)= 4.5996 ,so with ib<5 I cannot have the number.

Test: converter(20.6,5,5)= 20.5938

Test: converter(20.6,5,3)= 20.5000

Assignment 3:

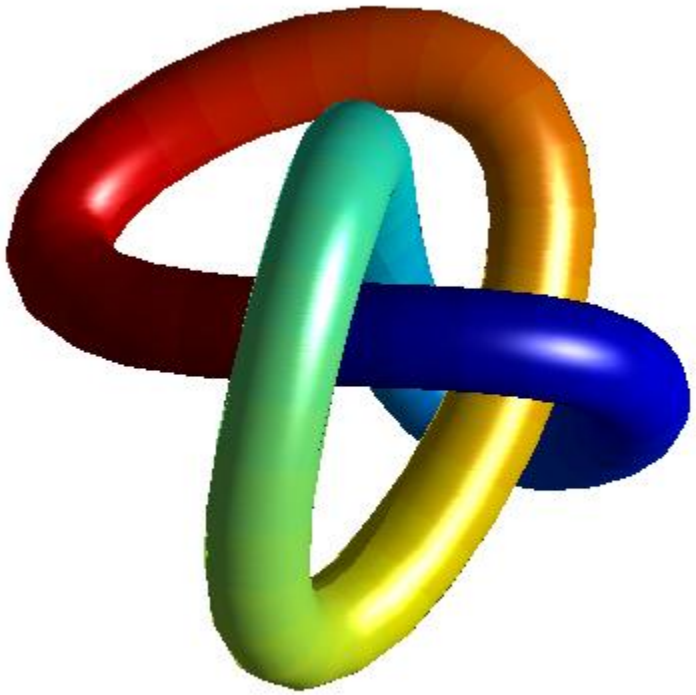
Function which truncates the floating point:

This function looks for all numbers in matrix if the number is zero sets the exponent to zero and if it's not zero finds the exponent ,In the next process shifts the floating point 2^(exp-num)positions ,which exp is exponent and num is the number of fb,then it Truncates the number with Fix function then shifts back the bits.

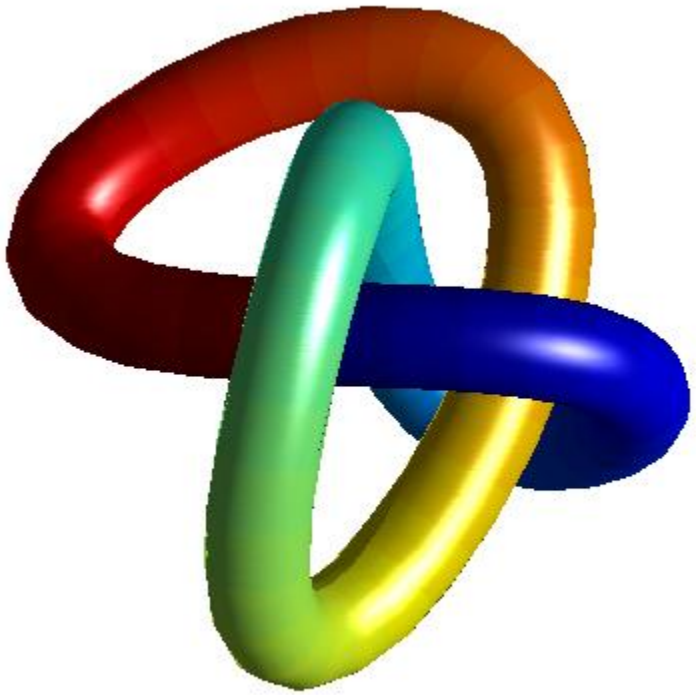
```
function newfloat=truncate(data,num)
x=0;
for i=data
   x=x+1;
   y=0;
   for j=i'
      y=y+1;
      if j==0
         exponent=0;
      else
exponent(y,x)=floor(log(j)/log(2));
      end
   end
end
newfloat = data./(2.^(exponent-num));
newfloat=fix(newfloat);
newfloat = newfloat.*(2.^(exponent-num));
```

Figures:

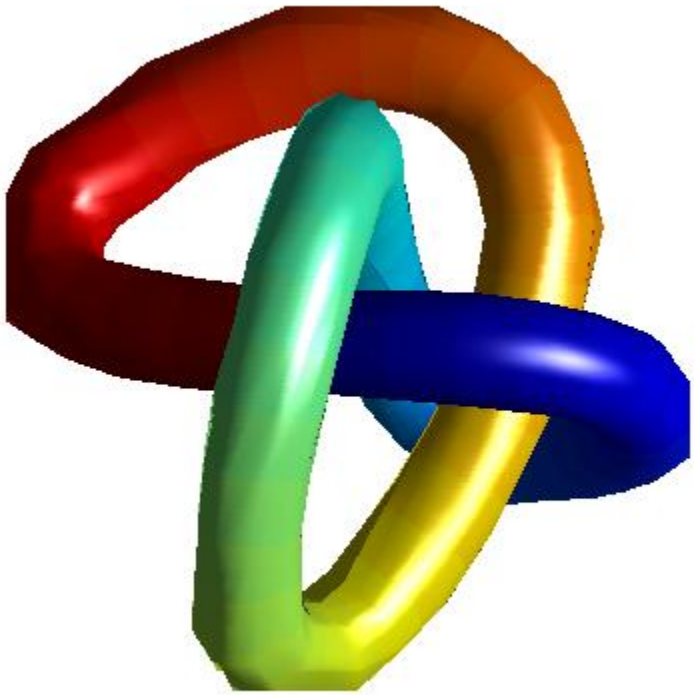
Ib=0 and fb=53



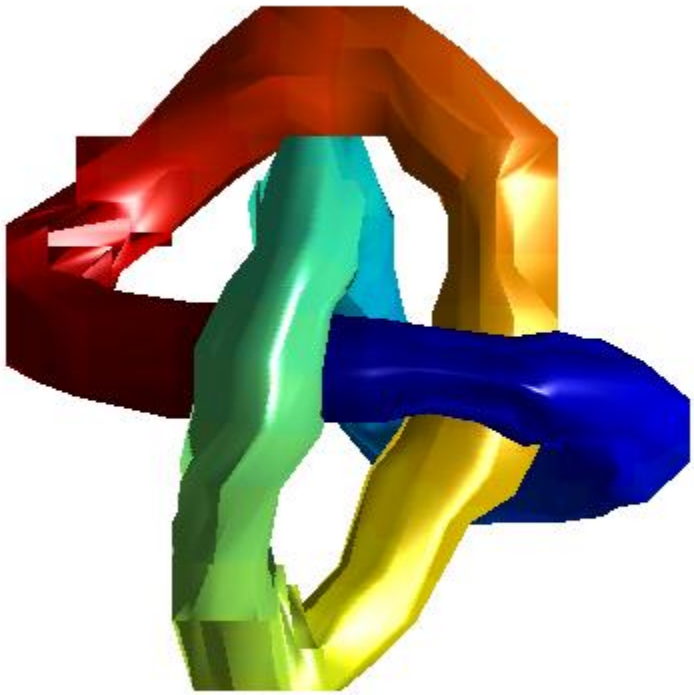
Ib=0 and fb=27



Ib=0 and fb=4,so it loses the smooth with fb=4



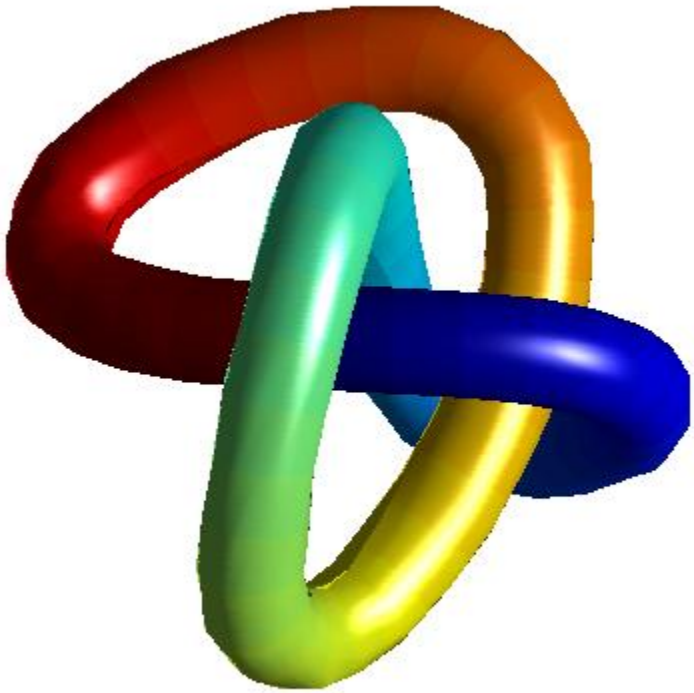
Ib=0 and fb=2



Assignment 4:

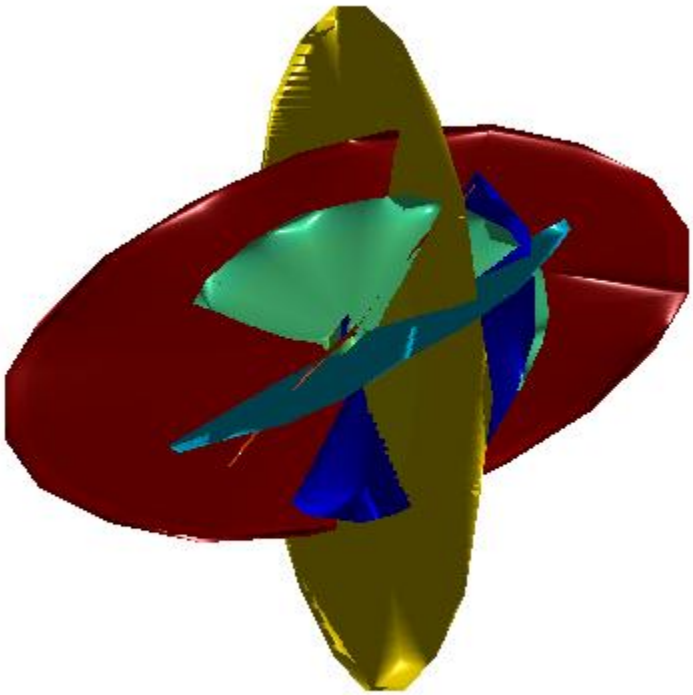
It seems that with 6 bits of mantissa we can have something smooth that looks like the main one.

With ib=0 and fb=6

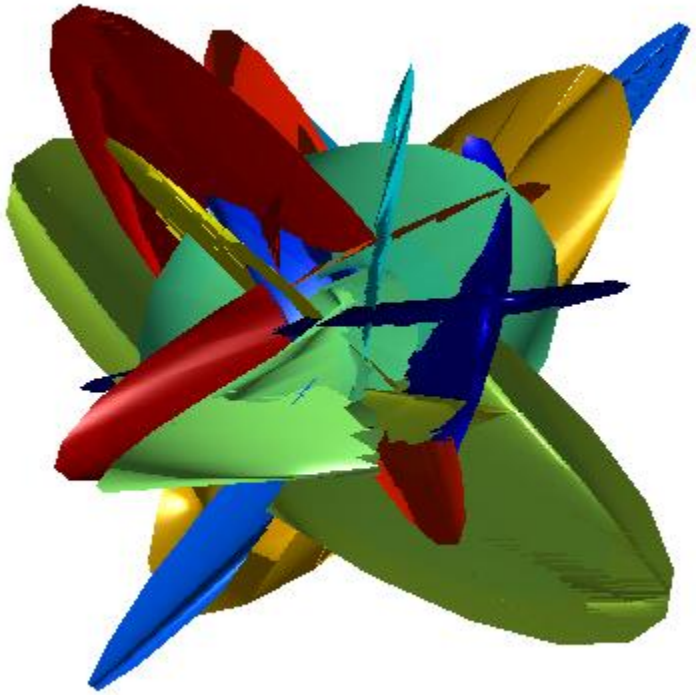


Assignment 5:

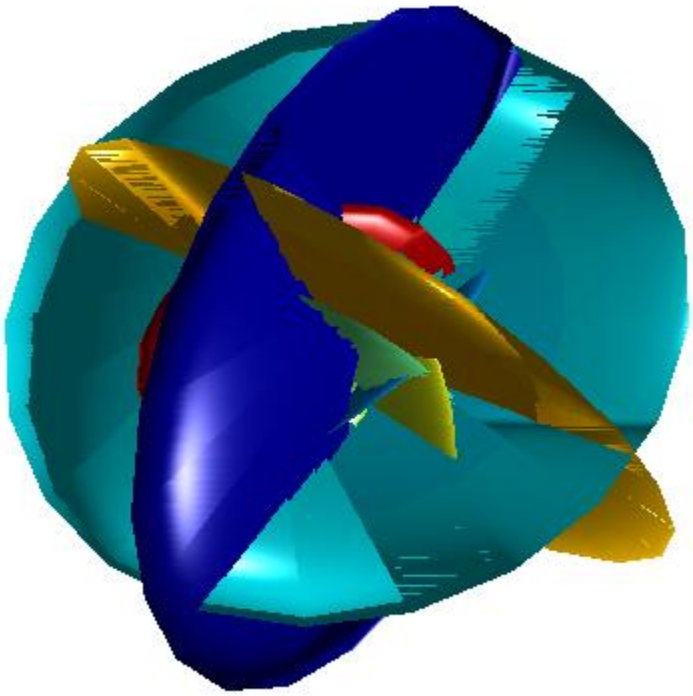
Ib=10 and fb=0



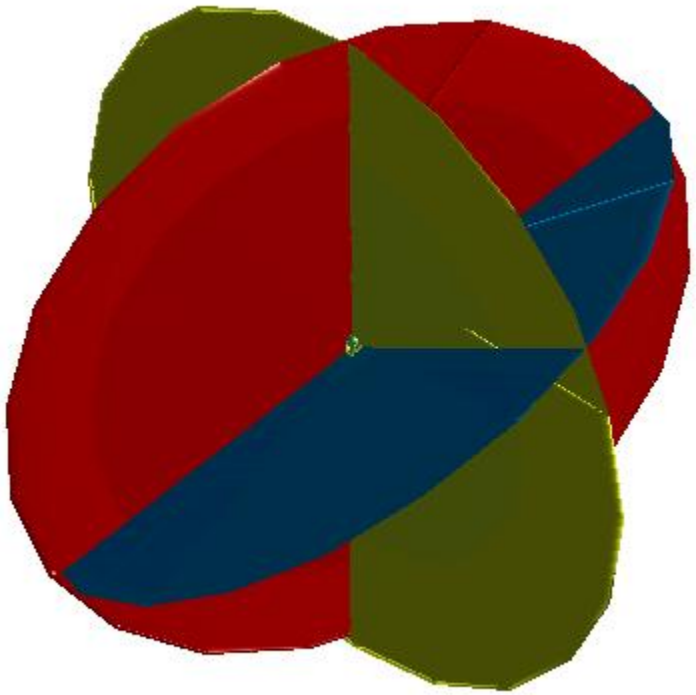
Ib=5 and fb=5



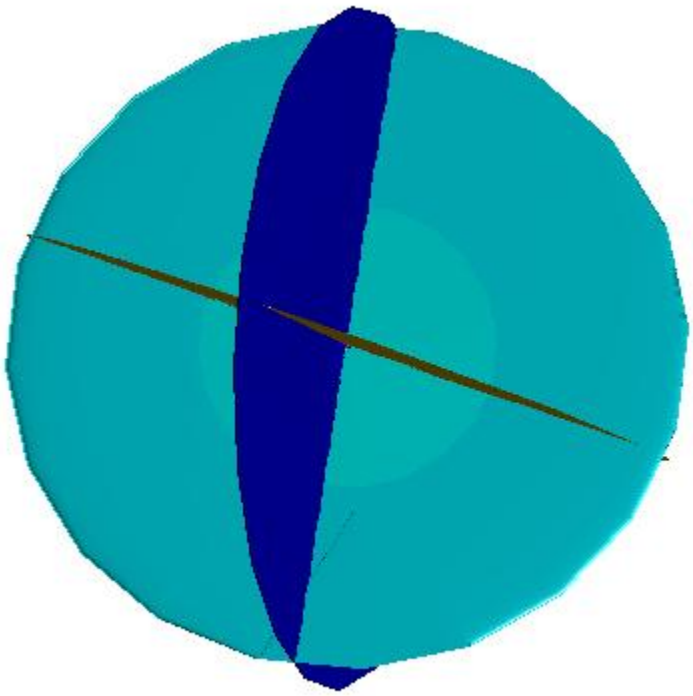
Ib=10 and fb=5



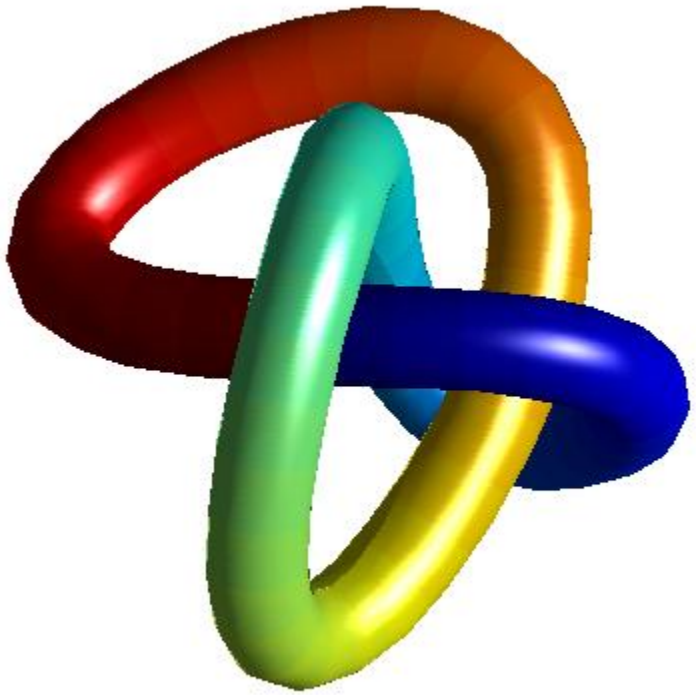
Ib=10 and fb=6



Ib=21 and fb=6



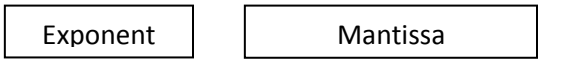
Ib=22 and fb=6



We can see with 22 bits of integer part we can have the same figure as the original one.

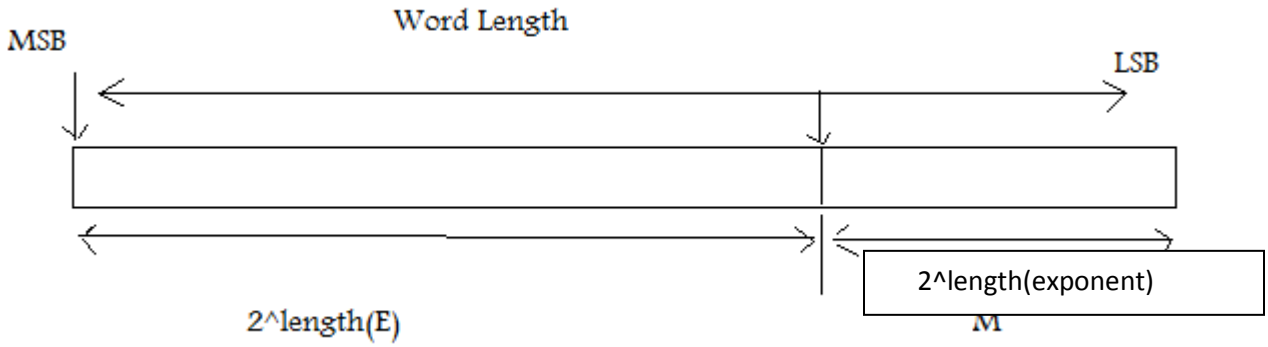
Assignment 6:

Floating Representation



Scale Precision bits

Fixed point Representation:



We can see that floating point is just like a window which moves along the fixed point representation and sets this movement by changing the exponent. What is seen is that fixed point needs a lot of bits to implement in comparison with floating point while floating point is more complex

Scale(length of E) =log2(word length-M)

Assignment 7:

Yes, there are some numbers that need to be in their representation to have better results, numbers which have big mantissa, I mean big numbers .For example for an addition of a big number (big word length)with a small number(small word length) with fixed point representation there is a need for a big adder, which for floating point is not the case.

Computer Arithmetic Final Assignment

Professor:Lambert Spangberg

Peyman Pouyan

P.N: 8312306296

sx07pp6@student.lth.se

Assignment1:

A function for inverting from to float to fix:

If the integer part is zero I will truncate the floating point (Assignment 3).But when the integer part is not zero I do like this:

Imagine the number is: 1110011.110101

I first shift the number to left as positions as the value of fb is, for example if fb=2 the shift would be 2 times to left and the number would be: 111001111.0101 then I truncate this number with function FIX and Fix(111001111.0101)_= 111001111.0000 then I shift back the numbers: 1110011.1100 .

I do the same for Integer part and I shift ib positions.

```
function newfix=converter(data, ib, fb)

newfix=data;

if (ib==0)

   newfix =truncate(data,fb);

else
```

```
    %Fraction part Truncation
    newfix=newfix.*(2^fb);
    newfix=fix(newfix);
    newfix=newfix./2^fb;
    %Integer part truncation
    newfix= newfix./(2^ib);
    newfix = newfix-fix(newfix);
    newfix = newfix.*2^ib;
end;
```

Assignment 2:

Testing the converter with different ib and fb:

20.6=10100.1001100110011001

Test : converter(20.6,10,10)= 20.5996

Test: converter(20.6,5,10)= 20.5996

Test: converter(20.6,4,10)= 4.5996 ,so with ib<5 I cannot have the number.

Test: converter(20.6,5,5)= 20.5938

Test: converter(20.6,5,3)= 20.5000

Assignment 3:

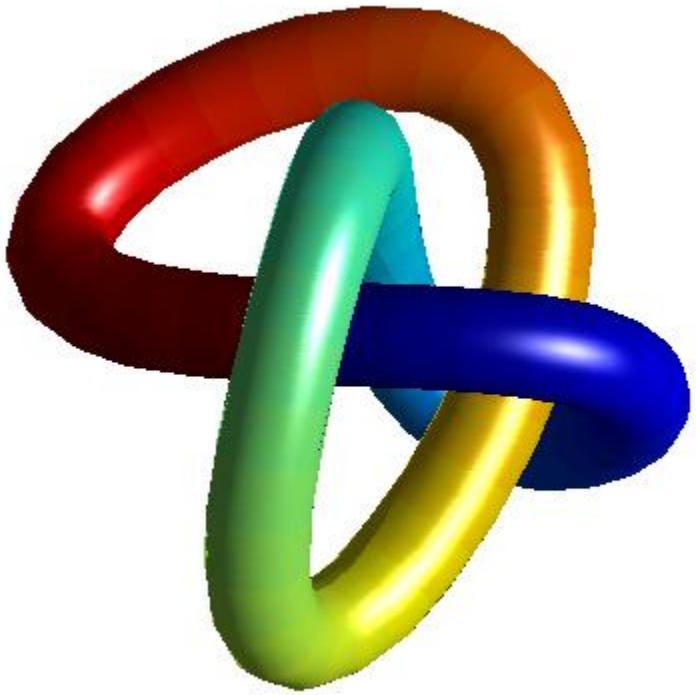
Function which truncates the floating point:

This function looks for all numbers in matrix if the number is zero sets the exponent to zero and if it's not zero finds the exponent ,In the next process shifts the floating point 2^(exp-num)positions ,which exp is exponent and num is the number of fb,then it Truncates the number with Fix function then shifts back the bits.

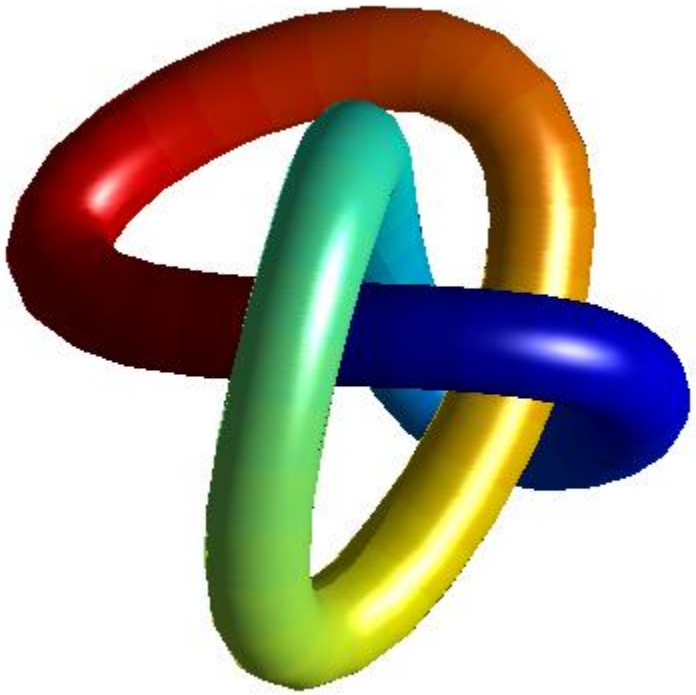
```
function newfloat=truncate(data,num)
x=0;
for i=data
   x=x+1;
   y=0;
   for j=i'
      y=y+1;
      if j==0
         exponent=0;
      else
exponent(y,x)=floor(log(j)/log(2));
      end
   end
end
newfloat = data./(2.^(exponent-num));
newfloat=fix(newfloat);
newfloat = newfloat.*(2.^(exponent-num));
```

Figures:

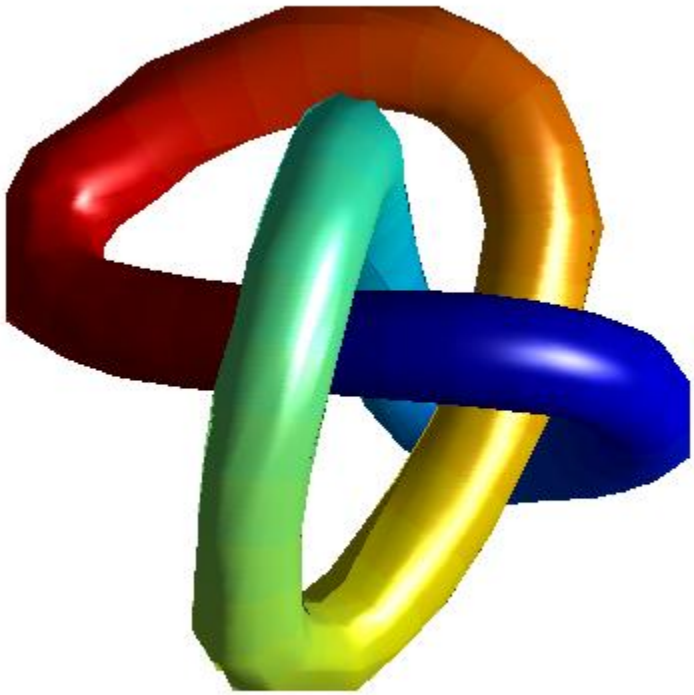
Ib=0 and fb=53



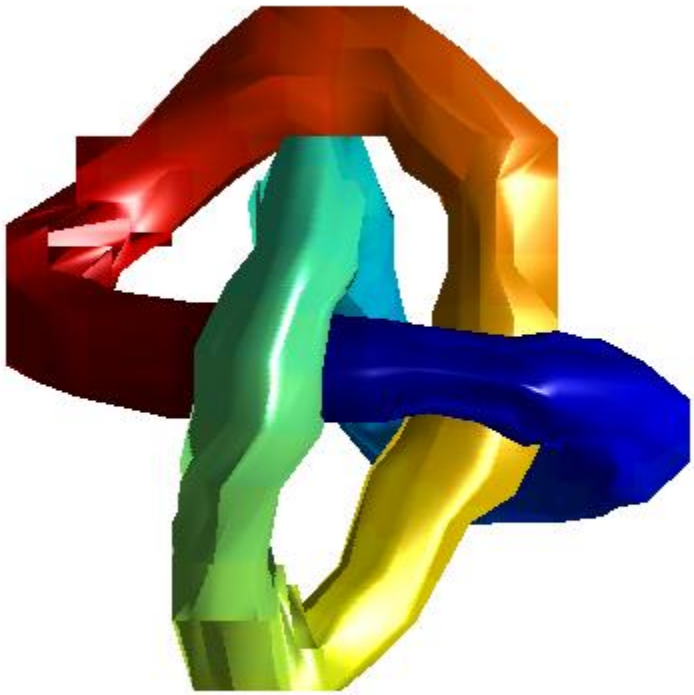
Ib=0 and fb=27



Ib=0 and fb=4,so it loses the smooth with fb=4



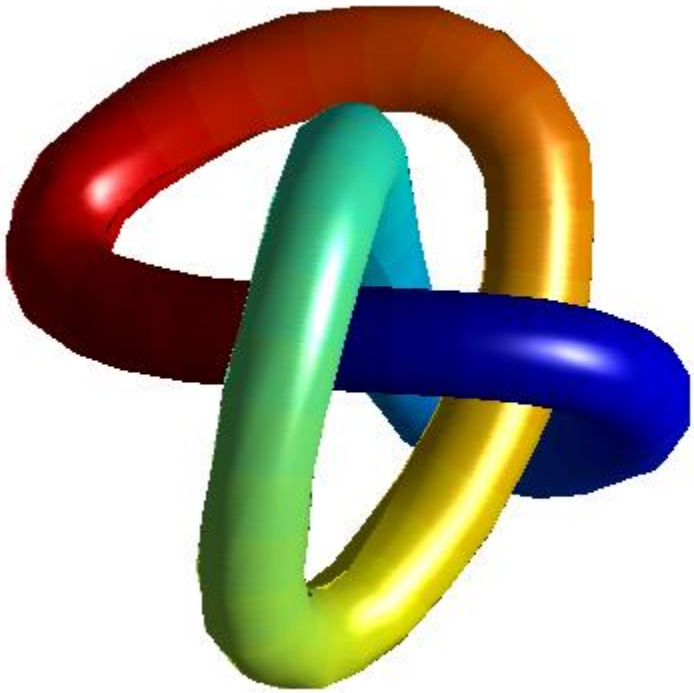
Ib=0 and fb=2



Assignment 4:

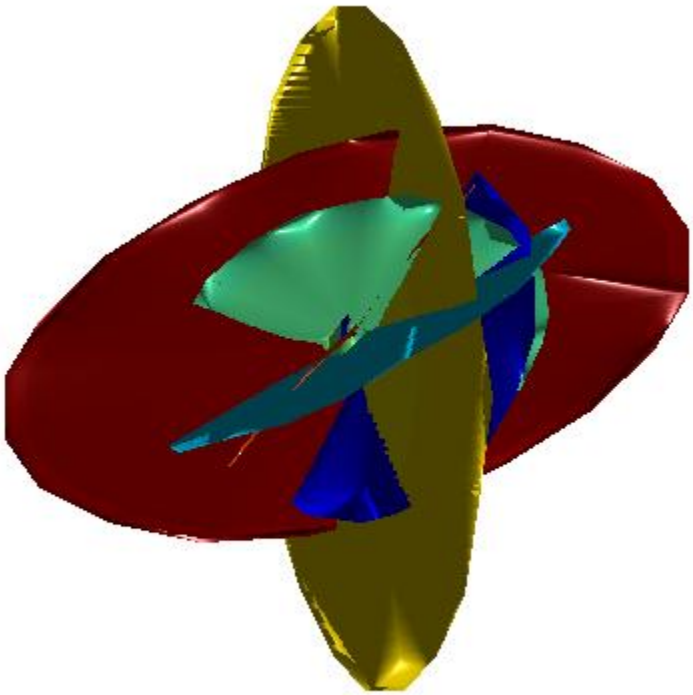
It seems that with 6 bits of mantissa we can have something smooth that looks like the main one.

With ib=0 and fb=6

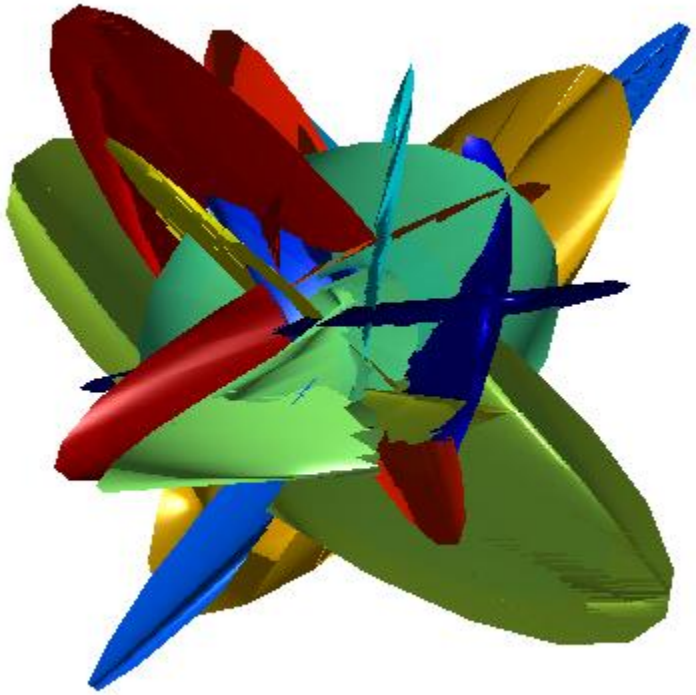


Assignment 5:

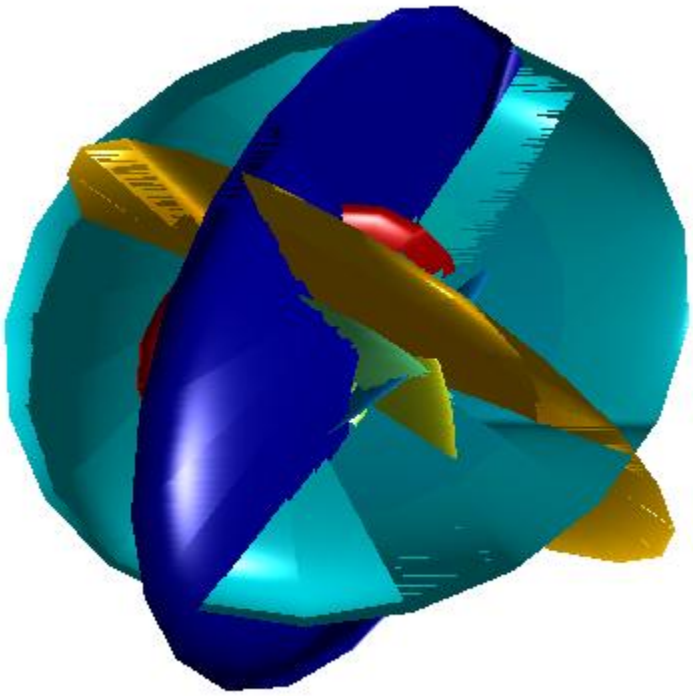
Ib=10 and fb=0



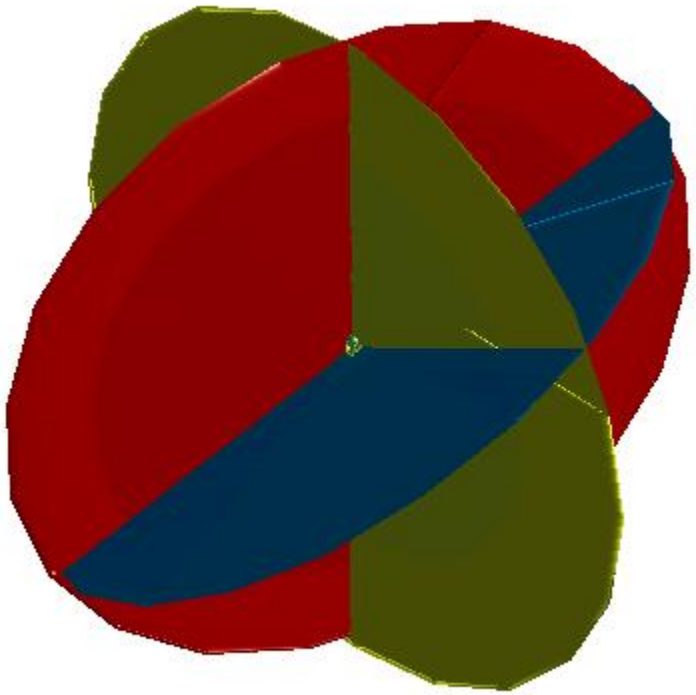
Ib=5 and fb=5



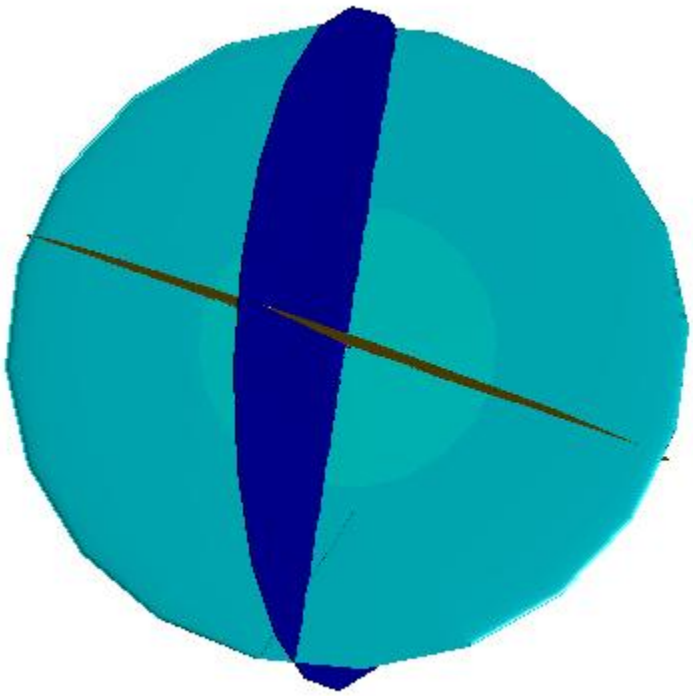
Ib=10 and fb=5



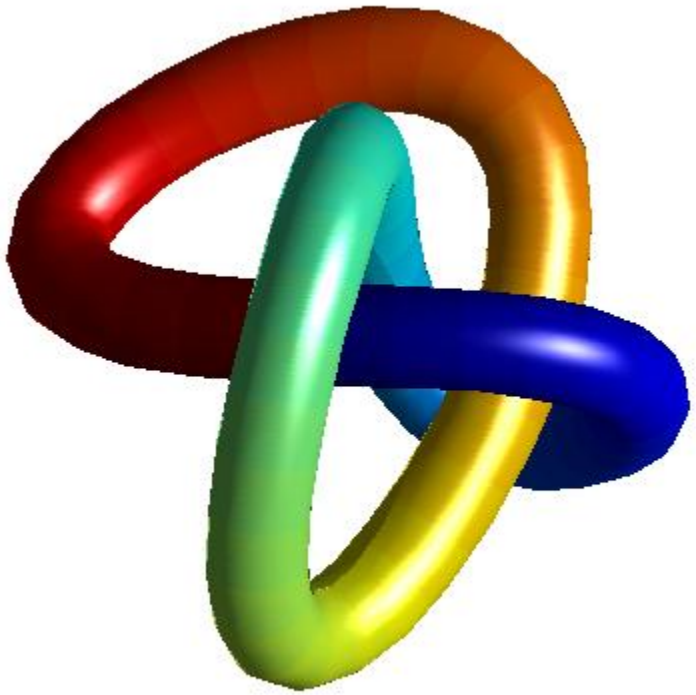
Ib=10 and fb=6



Ib=21 and fb=6



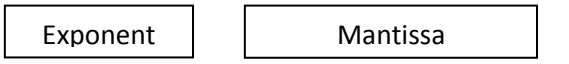
Ib=22 and fb=6



We can see with 22 bits of integer part we can have the same figure as the original one.

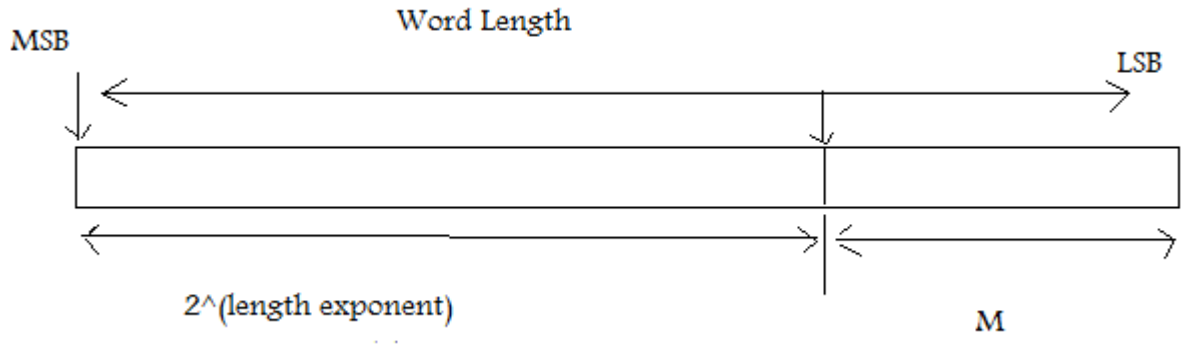
Assignment 6:

Floating Representation



Scale Precision bits

Fixed point Representation:



We can see that floating point is just like a window which moves along the fixed point representation and sets this movement by changing the exponent. What is seen is that fixed point needs a lot of bits to implement in comparison with floating point while floating point is more complex

Scale(length of E) =log2(word length-M)

Assignment 7:

Yes, there are some numbers that need to be in their representation to have better results, numbers which have big mantissa, I mean big numbers .For example for an addition of a big number (big word length)with a small number(small word length) with fixed point representation there is a need for a big adder, which for floating point is not the case.

Computer Arithmetic Final Assignment

Professor:Lambert Spangberg

Peyman Pouyan

P.N: 8312306296

sx07pp6@student.lth.se

Assignment1:

A function for inverting from to float to fix:

If the integer part is zero I will truncate the floating point (Assignment 3).But when the integer part is not zero I do like this:

Imagine the number is: 1110011.110101

I first shift the number to left as positions as the value of fb is, for example if fb=2 the shift would be 2 times to left and the number would be: 111001111.0101 then I truncate this number with function FIX and Fix(111001111.0101)_= 111001111.0000 then I shift back the numbers: 1110011.1100 .

I do the same for Integer part and I shift ib positions.

```
function newfix=converter(data, ib, fb)

newfix=data;

if (ib==0)

   newfix =truncate(data,fb);

else
```

```
    %Fraction part Truncation
    newfix=newfix.*(2^fb);
    newfix=fix(newfix);
    newfix=newfix./2^fb;
    %Integer part truncation
    newfix= newfix./(2^ib);
    newfix = newfix-fix(newfix);
    newfix = newfix.*2^ib;
end;
```

Assignment 2:

Testing the converter with different ib and fb:

20.6=10100.1001100110011001

Test : converter(20.6,10,10)= 20.5996

Test: converter(20.6,5,10)= 20.5996

Test: converter(20.6,4,10)= 4.5996 ,so with ib<5 I cannot have the number.

Test: converter(20.6,5,5)= 20.5938

Test: converter(20.6,5,3)= 20.5000

Assignment 3:

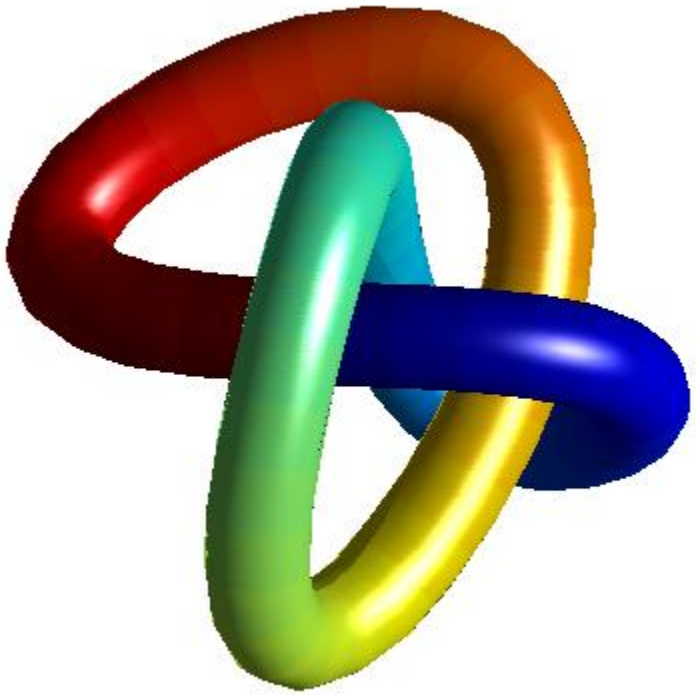
Function which truncates the floating point:

This function looks for all numbers in matrix if the number is zero sets the exponent to zero and if it's not zero finds the exponent ,In the next process shifts the floating point 2^(exp-num)positions ,which exp is exponent and num is the number of fb,then it Truncates the number with Fix function then shifts back the bits.

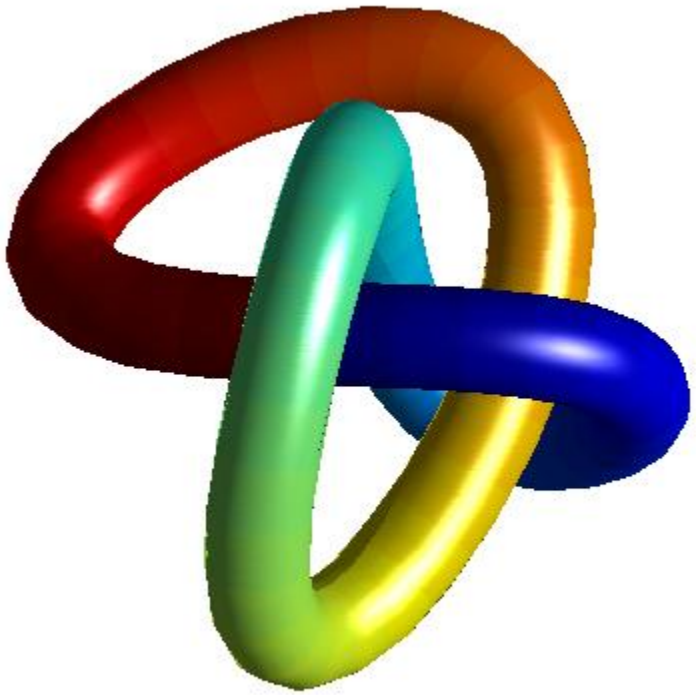
```
function newfloat=truncate(data,num)
x=0;
for i=data
   x=x+1;
   y=0;
   for j=i'
      y=y+1;
      if j==0
         exponent=0;
      else
exponent(y,x)=floor(log(j)/log(2));
      end
   end
end
newfloat = data./(2.^(exponent-num));
newfloat=fix(newfloat);
newfloat = newfloat.*(2.^(exponent-num));
```

Figures:

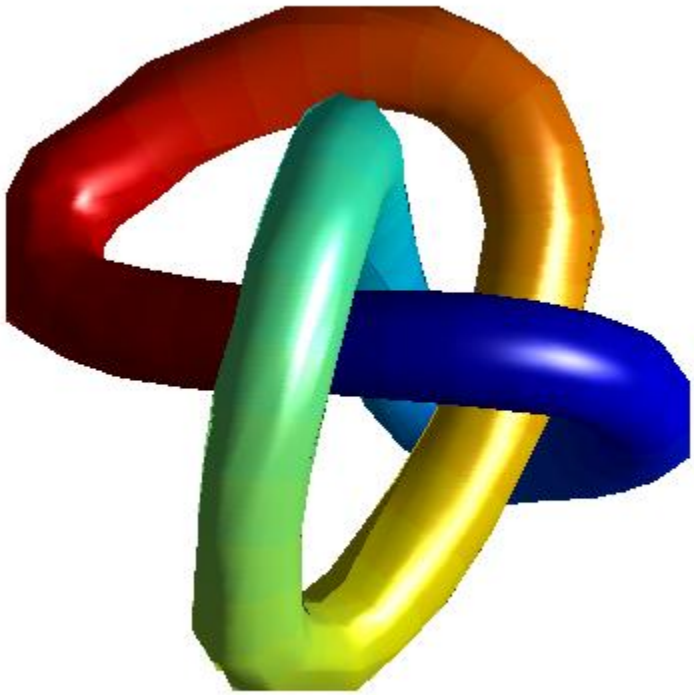
Ib=0 and fb=53



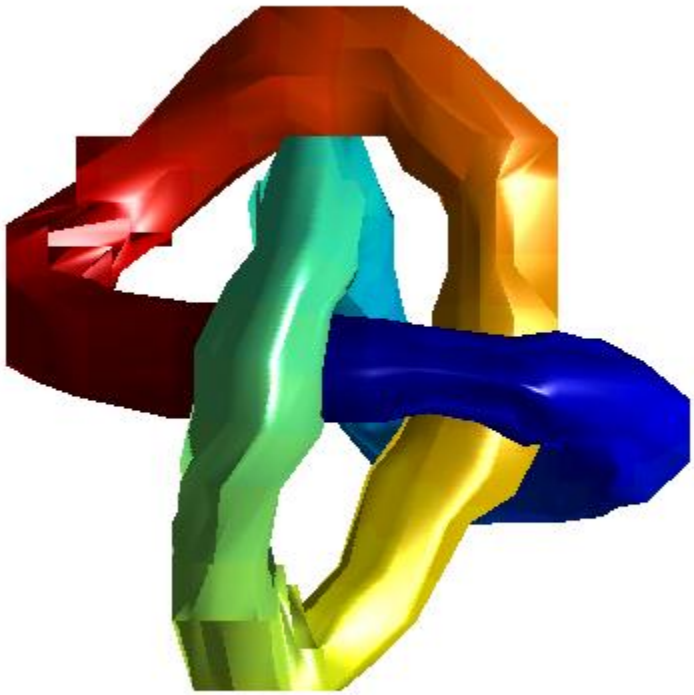
Ib=0 and fb=27



Ib=0 and fb=4,so it loses the smooth with fb=4



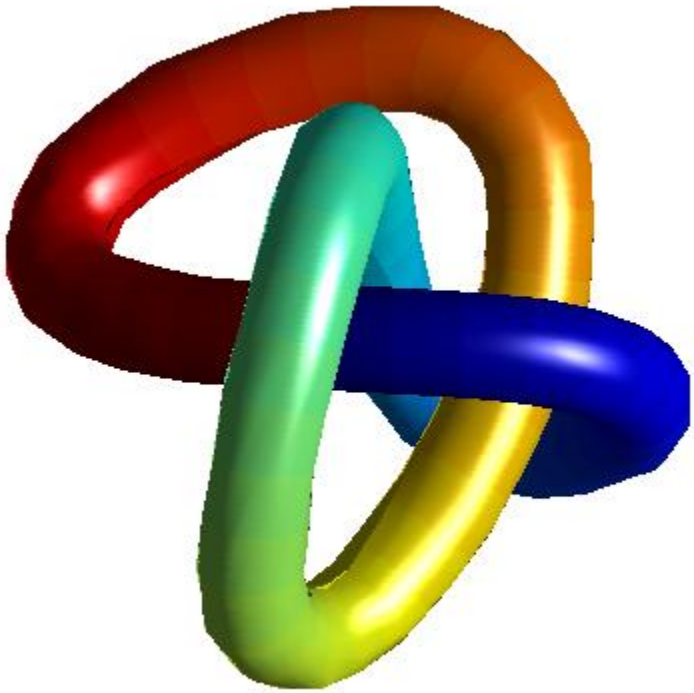
Ib=0 and fb=2



Assignment 4:

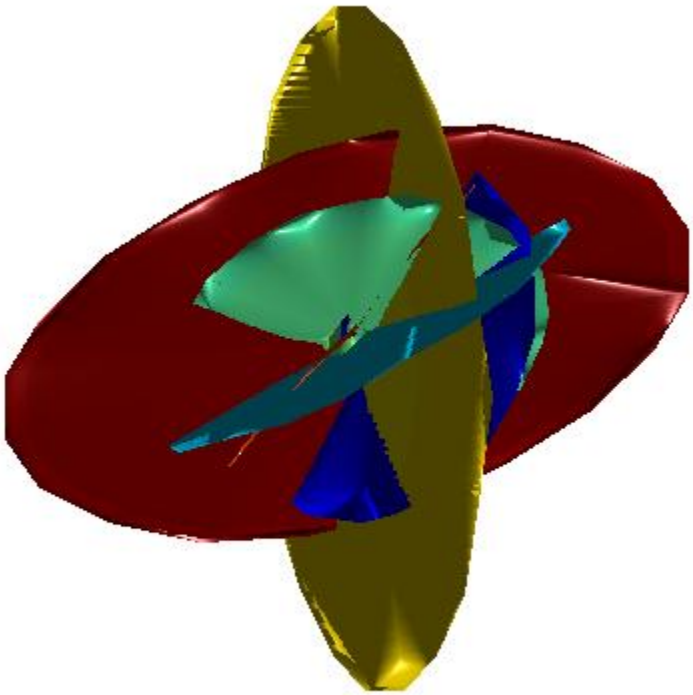
It seems that with 6 bits of mantissa we can have something smooth that looks like the main one.

With ib=0 and fb=6

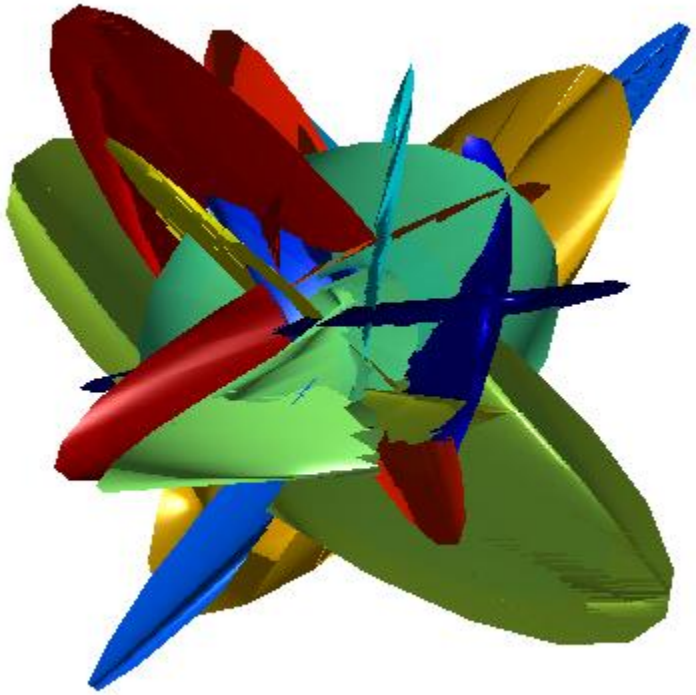


Assignment 5:

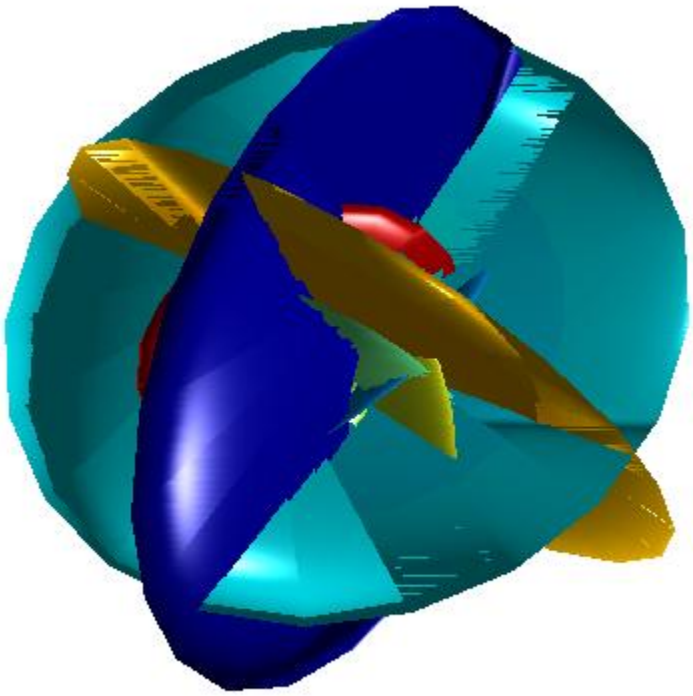
Ib=10 and fb=0



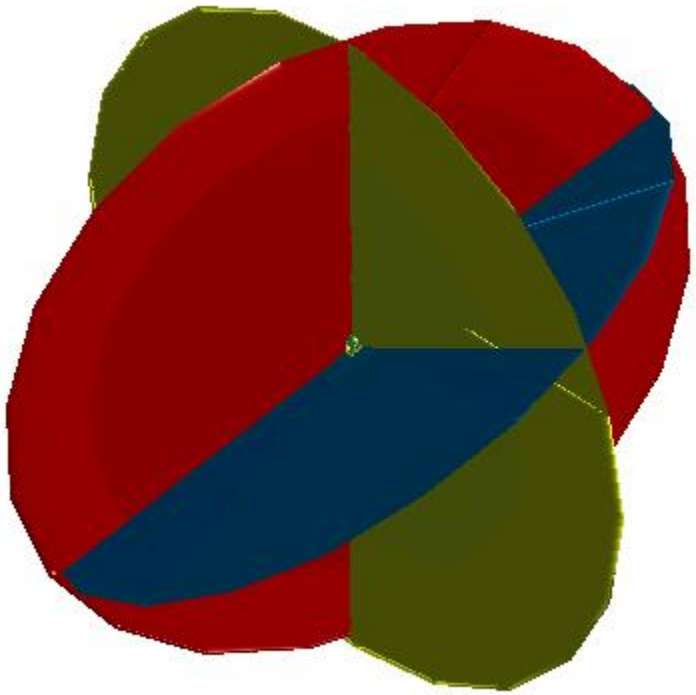
Ib=5 and fb=5



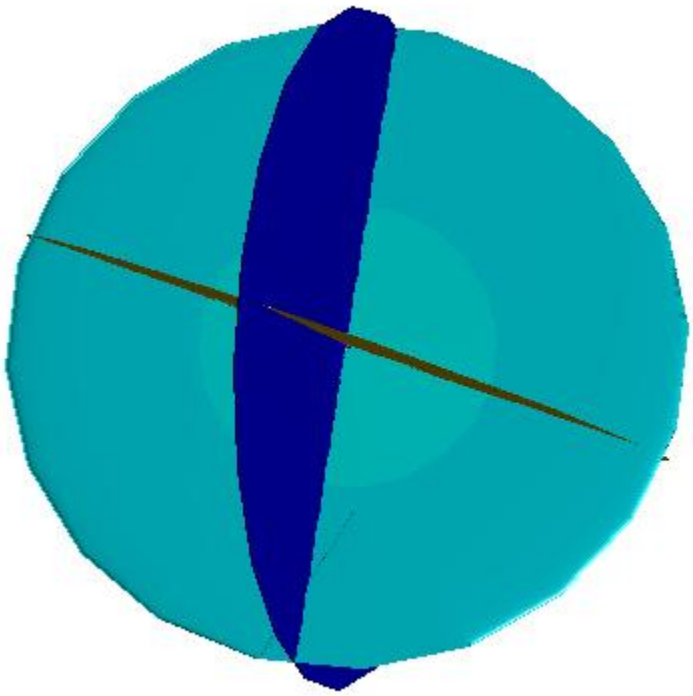
Ib=10 and fb=5



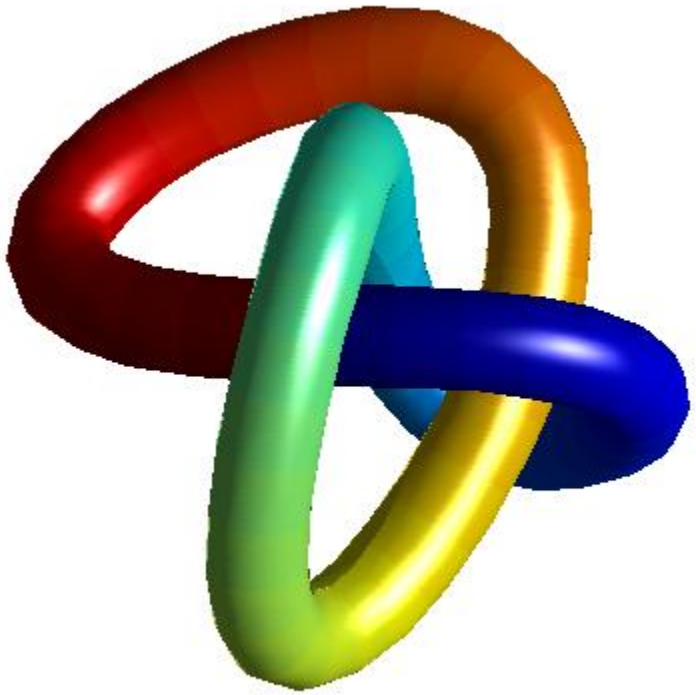
Ib=10 and fb=6



Ib=21 and fb=6



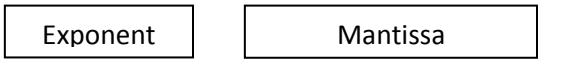
Ib=22 and fb=6



We can see with 22 bits of integer part we can have the same figure as the original one.

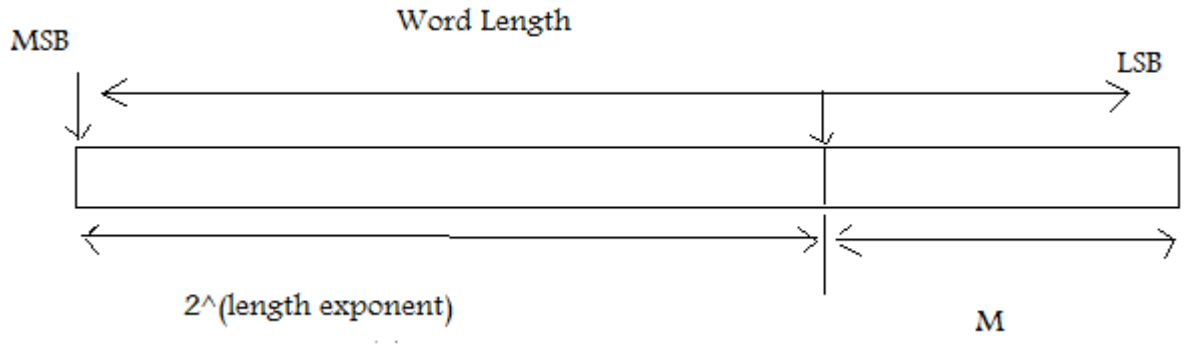
Assignment 6:

Floating Representation



Scale Precision bits

Fixed point Representation:



We can see that floating point is just like a window which moves along the fixed point representation and sets this movement by changing the exponent. What is seen is that fixed point needs a lot of bits to implement in comparison with floating point while floating point is more complex. So for floating point we need less hardware but it is harder to implement when for the fixed point bigger hardware is needed but it is more simple to implement.

Scale(length of E) =log2(word length-M)

Assignment 7:

Yes, there are some numbers that need to be in their representation to have better results, numbers which have big mantissa, I mean big numbers .For example for an addition of a big number (big word length)with a small number(small word length) with fixed point representation there is a need for a big adder, which for floating point is not the case.