JavaWildfire 5.5

Complex Numbers Operations
for c_var & cut_c variations
by
Jesús Sosa

c_var variation

This variation owes his name to the expression “complex variation”, this is a powerful variation which can be customized with your own code to produce a completely new variation. The customized code follows some rules from the java programing language, but you can code your own variation with basic knowledge of complex numbers and math expressions.

cut_c variation

This variation owes his name to the expression “cutting complex”, this is a powerful variation which can be customized with your own code to produce a completely new variation. The customized code follows some rules from the java programing language, but you can code your own variation with basic knowledge of complex numbers and math expressions.

Complex Numbers Primer

Every Complex number is made of two parts, one real and other imaginary both parts are represented as a sum of two parts but the imaginary it is qualified with a letter i

Example of a Complex number: $6 + 3i$

Complex numbers can be saved in variables, lets be **x** a complex variable
So we can save a complex number in a complex variable.

Example **x**= $3 + 2i$

In computers we need a way to represent complex numbers and complex variables

vec2 it is a object type for a pair of numbers, this pair of numbers represent the real and the imaginary part of a complex number.

To construct a complex number you have to use the new operator over the vec2 object and pass two arguments, first argument it is the real part and second argument the imaginary part.

Example:

vec2 z= new vec2(3.0,6.0);

in the previous example We have created a complex number in variable z with a value z=3+6i, the real and imaginary part of the complex number are saved in properties x and y of the z variable, in the example above z.x=3.0 , and z.y=6.0

Complex Numbers Basic Operations

Complex numbers like real numbers have rules to perform operations with them, by example we can add two complex numbers a and b adding the real parts of both numbers and the imaginary parts of both numbers to get a new complex number which is the sum of two complex.

Lets be:

a= 3+2i, b= 5+3i then a+b is:

a+b = (3+5) + (2+3)i , a+b=8+5i

To perform sum of complex numbers we can use a function c_add

vec2 c_add(vec2 a,vec2 b)
This function receive two arguments, first it is the first complex number to add and second argument it is the second complex number to add, but this function have to return a new complex number which is the result of adding the two numbers in the arguments

Example: a=3+4i, b=1+2i, c=a+b=(3+1) +(4+2)i, c=4+6i

```
vec2 a =new vec2(3,4);
vec2 b =new vec2(1,2);
vec2 c= c_add(a,b);
```

To perform subtraction of complex numbers we can use a function c_sub

vec2 c_sub(vec2 a,vec2 b)
This function receive two arguments, first it is the complex number from which we are going to subtract and the second complex the number to subtract, this function have to return a new complex number which is the result of subtract the two numbers in the arguments

Example: a=3+4i, b=1+2i, c=a+b=(3-1) +(4-2)i, c=2+2i

```
vec2 a =new vec2(3,4);
vec2 b =new vec2(1,2);
vec2 c= c_sub(a,b);
```

To perform multiplication of complex numbers we can use a function c_mul

vec2 c_mul(vec2 a,vec2 b)
This function receive two arguments, the first complex number and the second complex the number we want to multiply, also this function returns a new complex number which is the result of multiplication the two numbers in the arguments

Example: a=3+4i, b=1+2i, c=a*b=(3*1)+(4*2)i, c=3+8i

```
vec2 a =new vec2(3,4);
vec2 b =new vec2(1,2);
vec2 c= c_mul(a,b);
```

To perform division of complex numbers we can use a function c_div

vec2 c_div(vec2 a,vec2 b)
This function is a little more complicated but also receive two arguments,the first it is the numerator and the second argument it is the divisor

Example: a=3+4i, b=1+2i, c=a/b, c=((3*1)+(4*2))/(4*4 + 2*2) + ((4*1)-(3*2))/(4*4 + 2*2)i

```
vec2 a =new vec2(3,4);
vec2 b =new vec2(1,2);
vec2 c= c_div(a,b);
```

Besides of the operations of add, subtract, multiplication and division of complex, numbers, the complex numbers also can evaluate other operations using different functions, lets see some other functions we can use to make operations with complex numbers.

vec2 c_one()
This function construct a complex number 1 + 0i

Example:

```
   vec2 z=c_one();
```

vec2 c_i()
This function construct a complex number 0 + 1i

Example to save in z =0 + 1i

```
   vec2 z=c_i();
```

vec2 c_ni()
This function construct a complex number 0 - 1i

Example to save in z = 0 -1i

```
   vec2 z=c_ni();
```

vec2 c_conj(vec2 c)
This function construct a conjugated complex of the argument
Formula: c.x,-c.y

Example to save in z = 3+2i;

```
   vec2 a= new vec2(3,-2);
   vec2 z= c_conj(a);
```

vec2 c_from_polar(double r, double theta)
This function construct a complex number from polar coordinates
Formula: a.x=r,a.y=theta

vec2 c_to_polar(vec2 c)

This function construct a complex number from Cartesian Coordinates
Formula: a.x=c.x,a.y=c.y

vec2 c_exp(vec2 c)
This function Computes `e^(c)`, where `e` is the base of the natural logarithm.
Formula: e^c

vec2 c_exp(double base, vec2 c)
Raises a doubleing point number to the complex power `c`.
Formula: base^c

vec2 c_ln(vec2 c)
Computes the principal value of natural logarithm of `c`.
Formula: ln(c)

vec2 c_log(vec2 c, double base)
Returns the logarithm of `c` with respect to an arbitrary base.

vec2 c_sqrt(vec2 c)
Compute the square root of the complex number in the argument

vec2 c_pow(vec2 c, double e)
Raises c to the power of the real number in the second argument

vec2 c_pow(vec2 c, vec2 e)
Raises `c` to a complex power `e`.

vec2 c_sin(vec2 c)
Compute the sin of the complex argument

vec2 c_cos(vec2 c)
Compute the cos of the complex argument
formula: cos(a + bi) = cos(a)cosh(b) - i*sin(a)sinh(b)

vec2 c_tan(vec2 c)
Compute the tan of the complex argument

boolean c_eq(vec2 a, vec2 b)
Test if two complex are equals

vec2 c_atan(vec2 c)
Compute the arctangent of the complex argument
formula: arctan(z) = (ln(1+iz) - ln(1-iz))/(2i)

vec2 c_asin(vec2 c)
Compute the arcsin of the complex argument
formula: arcsin(z) = -i ln(sqrt(1-z^2) + iz)

vec2 c_acos(vec2 c)
Compute the arccos of the complex argument
formula: arccos(z) = -i ln(i sqrt(1-z^2) + z)

vec2 c_sinh(vec2 c)

Compute the sin hyperbolic of the complex argument

vec2 c_cosh(vec2 c)
Compute the cos hyperbolic of the complex argument

vec2 c_tanh(vec2 c)
Compute the tan Hyperbolic of the complex argument

vec2 c_asinh(vec2 c)
Compute the arc sin hyperbolic of the complex argument
formula: arcsinh(z) = ln(z + sqrt(1+z^2))

vec2 c_acosh(vec2 c)
Compute the arc cos hyperbolic of the complex argument
formula: arccosh(z) = 2 ln(sqrt((z+1)/2) + sqrt((z-1)/2))

vec2 c_atanh(vec2 c)
Compute the arc tan hyperbolic of the complex argument
formula: arctanh(z) = (ln(1+z) - ln(1-z))/2

vec2 c_rem(vec2 c, vec2 modulus)
Attempts to identify the gaussian integer whose product with `modulus`
is closest to `c`

vec2 c_inv(vec2 c)
Compute the inverse of the complex argument
Formula: 1/c

vec2 iabs(vec2 c)
Compute the absolute value of the imaginary part
Formula: c.x,abs(c.y)

vec2 c_iabs(vec2 c)
Compute the absolute value of the imaginary part
Formula: c.x,abs(c.y)

vec2 c_abs(vec2 c)
Compute the absolute value of the real part
Formula: abs(c.x),c.y

Build your own custom complex formula for c_var & cut_c variations

c_var and cut_c variations are compatible variations, this means you can use the same code between the two variations.

If you open the code Parameter in this variations you will see one import line. This import line defines the object type vec2, the object to work with complex numbers, this line is necessary to work with vec2 objects, also in this parameter it is defined a function f with an complex argument and return a complex value, inside this function you can perform complex number operations and at the end you need to return a complex value, the instruction return perform this last task.

You can play different custom complex functions if you comment code and uncomment only one of the returns. Two followed backslashes means comment in java.

The following example comments default last two lines
// vec2 a=c_add(c_inv(z),c_exp(c_inv(z)));
// return c_add(a,c_exp(new vec2(0.0, 0.0)));
and uncomment the
return c_sin(z);

```
import js.glsl.vec2;
public vec2 f(vec2 z)
{
  //vec2 a= c_add(z,new vec2(0.0,0.0));
  //vec2 b= c_sub(z,new vec2(0.0,0.0));
  //  return c_mul(a,b);

  // return c_exp(z);

  // return c_log(z,10.0);

  // return c_sqrt(z);

  // return c_pow(z,-2.0);

   return c_sin(z);

  // return sinh(z);

  // return c_cos(z);

  // return c_sinh(z);

  // return c_cosh(z);

  // return cosh(z);

  // return c_acos(z);

  // return c_asin(z);

  // return c_atan(z);

  // return tanh(z);

  // vec2 pow=new vec2(-2.0,-.60);
  // return c_pow(z,pow);

  // return c_conj(z);

  // return c_inv(z);

  // vec2 a=c_add(c_inv(z),c_exp(c_inv(z)));
  // return c_add(a,c_exp(new vec2(0.0, 0.0)));
}
```

You may try uncomment different return values that appear in the default code to see how this variation change.

To create your own custom code, you really can use any of the operations functions for complex numbers used in expressions as you wish, the following example define a function f(z) = e^z, where the function evaluate the basis of natural logarithm "e" raised to the power of a complex number z, in this case we are using the c_exp function.

```
import js.glsl.vec2;
public vec2 f(vec2 z)
{
  return  c_exp(z);

}
```

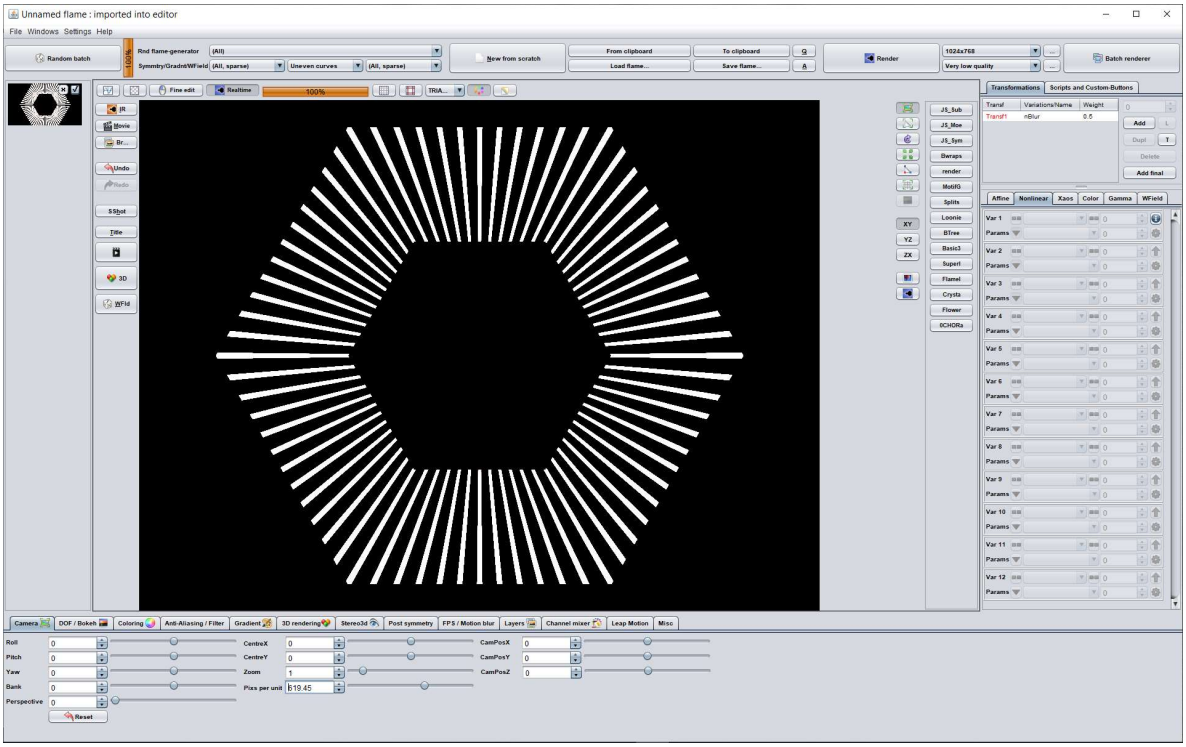
But you really can build a very complex function "public vec2 f(vec2 z)"

Lets by example if we want to code the function f(z)= (2z+1)/(z(z^2+1)
You need to code like this:

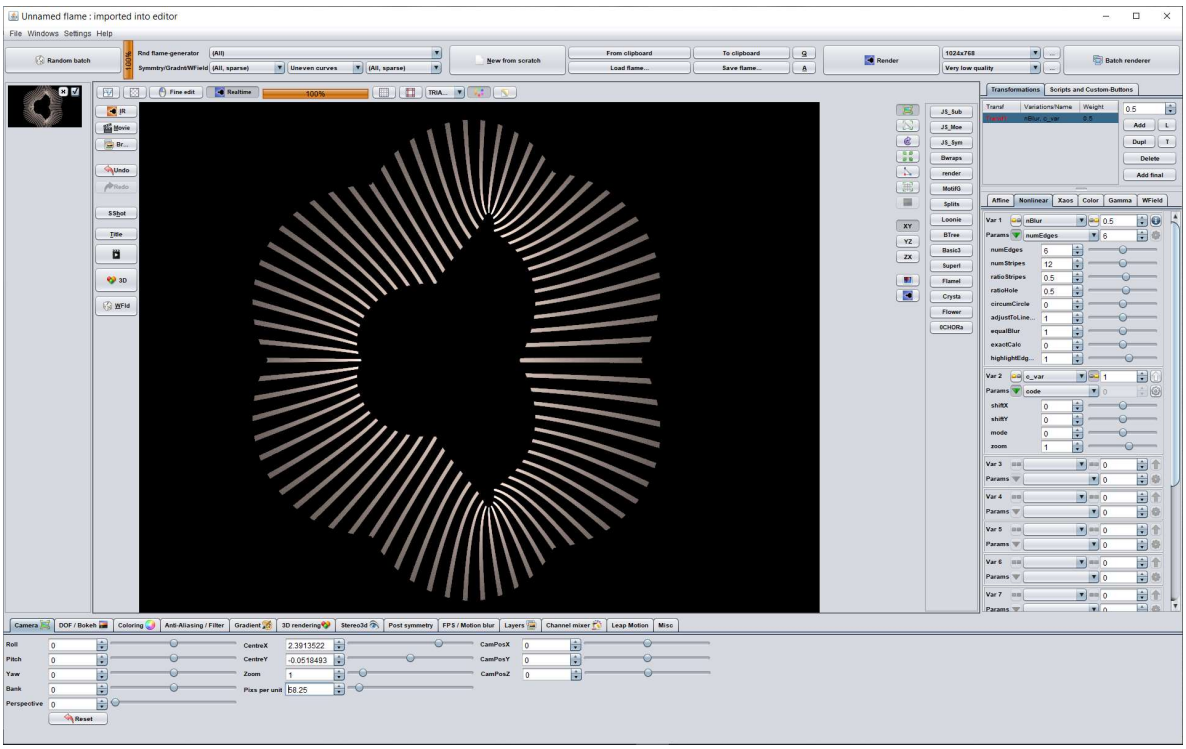
```
import js.glsl.vec2;
public vec2 f(vec2 z)
{
    // f(z)= (2z+1)/(z(z^2+1)
  vec2 a= new vec2(2,0);
  vec2 n1= c_add(c_mul(a,z),c_one());
  vec2 n2= c_mul(z,c_add(c_pow(z,2) ,c_one()));
  return  c_div(n1,n2);

}
```

Now you can use your custom complex function as a "post c_var" variation
First create a flame with some content, lets start with a nblur

Original nblur variation



Now add a “post c_var” variation and substitute the code parameter with the custom complex function code for f(z)= (2z+1)/(z(z^2+1)



You also can use the same function in the cut_c variation.
Start a new flame from scratch and add a “cut_c” variation, edit the code parameter and add the same code for the function f(z)= (2z+1)/(z(z^2+1), you will get an grid of dots as seen after this transformation.