Field Definition: Closed Form expression

Users can define closed form expressions. The expression can include tensor operators and variables. Differentiation is applied by differentiating in respect to some variable(s).

In Action

It is natural to define a function with an expression: F(x) = x^2

In the surface language we added function cfexp() where the first argument exp is an expression and the second x is a variable.

```
tensor [] exp = x*x;
ofield#2(2)[] F = cfexp(exp,x);//define F with variable x
tensor[2] v = [3,7];
tensor[] outF = inst(F,v);//evaluate F with argument v
```

We commonly refer to the right-hand-side to variable F as a cfexp (closed-form expression). The cfe is created with variable x, but is actually evaluated with v.

$$outF = F(v) = v_0^2 + v_1^2$$

The cfexp is a Diderot "ofield" type, but is treated the same as a Diderot "field type". The user can apply other tensor and field operators on the cfexp including differentiation.

```
ofield #1(2)[2] GF = ∇F;
tensor[] outGF = inst(GF,v);
```

The differentiation of the cfexp is computed in respect to the variable v. We illustrate the expected structure below:

$$outGF = \nabla F(v) = \begin{bmatrix} 2 * v_0 \\ 2 * v_1 \end{bmatrix}$$

A function can be defined with multiple variables.

$$F(a,b) = a + b$$

and similarly a cfexp can be defined with multiple variables

```
real[] a = 1; real b = 7;
tensor [] exp = a+b;
ofield#k(d)[] G = cfexp(exp,a);
ofield#k(d)[] H = cfexp(exp,a,b);
ofield#k(d)[] I = cfexp(exp,b);
```

The distinction between G, H, and I is that differentiation is applied in respect to either one or two variables.

$$\nabla G_a = \nabla a + b$$

$$\nabla H_{ab} = \nabla a + \nabla b$$

$$\nabla I_b = a + \nabla b$$

Details

- Branch: Diderot-Dev
- Syntax: “cfexp()"
    - Use an ofield type ofield#k(d)[β], which is the same as a Diderot field but has an “o” in front of it.
    - Declare a closed form expression with cfexp(exp,v0) and evaluate that oField with “inst()”
    - “exp” is the core computation that includes operators on and between variables
    - “v0” is the variable we differentiate in respect to. We accept 1-3 “v0” terms
    - cfexp(): tensor[α] × tensor[β] . . . → ofield#k(d)[α]
    - inst(): ofield#k(d)[α] × tensor[β] · · · → tensor[α]
- Text: see [Doc]
- Issues/Future Work:
    - Need to define/initiate all variables before cfexp() is called.
    - OField type doesn’t describe types for multiple inputs, need to change typechecker
    - Remove k-continuity
    - Needs more extensive testing

Directory Organization

- Base Case Examples
    - [f$_{v}$ = v] : X1/v.diderot,
    - [f$_{v}$ = v • v] : X1/vv.diderot,
    - [f$_{v}$ = (v • v) $*$ v] : X1/vvv.diderot, and
    - [f$_{s}$ = s^3] : X2/sss.diderot
- Multiple variables in core computation and differentiate in respect to one variable
    - [f$_{x}$ =(1−|x|)^4]Sphere: X3/sphere.diderot,
    - [f$_{x}$ =(x−cutPos)•curNorm] clip: X3/clip.diderot,
    - [f$_{x}$= (1 $*$ (1 − |x|))^3] Circle : X4/circle.diderot, and
    - [f$_{x}$ = (1 − |x|/y)^4] Enr : X4/enr.diderot
- Multiple variables are in the core computation and we differentiate in respect to multiple variables
    - [f_{sv} = s $*$ v] : X5/m1.diderot,
    - [f_{svx} = s $*$ v + x] : X5/m2.diderot, and
    - [f_{abc} = a^3bc^2] : X5/m3.diderot