Université Joseph Fourier
GRENOBLE
UFR IM²AG

Fonctions implicites

Henry Lefèvre – M2 Mia Parcours GICAO
Aymeric Seguret – M2 Mia Parcours GICAO

Dans le cadre du TP, nous cherchons à représenter une surface vue comme l'isopotentiel d'un champ scalaire de l'espace. C'est-à-dire, qu'une surface est définie par un ensemble de points de l'espace qui vérifient f(x,y,z) = k, avec k l'isopotentiel choisi.

1. Primitives de base

A partir de la base donnée du TP, nous avons créé une classe pour chaque primitive que l'on souhaitait modéliser. Chaque classe hérite de l'objet MyImplicitFunction afin de redéfinir les deux fonctions Eval et EvalDev.

La fonction Eval représente l'évaluation du champ scalaire en un point de l'espace. De même EvalDev représente l'évaluation du gradient.

Principe des différences finies pour le calcul du gradient :

Le calcul revient à approximer le vecteur gradient grâce à l'évaluation locale de la surface. Autrement dit faire une dérivation implicite. Le schéma correspondant pour ce calcul est le suivant :

```
glm::vec3 XXX::EvalDev(glm::vec3 p) const {
   float d = 0.01;
   float dX = Eval(vec3(p.x+d, p.y, p.z)) - Eval(vec3(p.x-d, p.y, p.z));
   float dY = Eval(vec3(p.x, p.y+d, p.z)) - Eval(vec3(p.x, p.y-d, p.z));
   float dZ = Eval(vec3(p.x, p.y, p.z+d)) - Eval(vec3(p.x, p.y, p.z-d));

   return vec3(dX, dY, dZ);
}
```

Les 3 composantes du gradient sont calculées par différences de l'élément suivant par le précédent. Les éléments sont repérés grâce au pas de discrétisation qui est ici égal à 0,01.

Cependant, nous n'utilisons pas le principe des différences finies. A la place nous évaluons les normales en fonction de la surface représentée.

Ainsi, nous avons pu modéliser 3 primitives de bases qui sont : la sphère, le cylindre et le tore

- La sphère

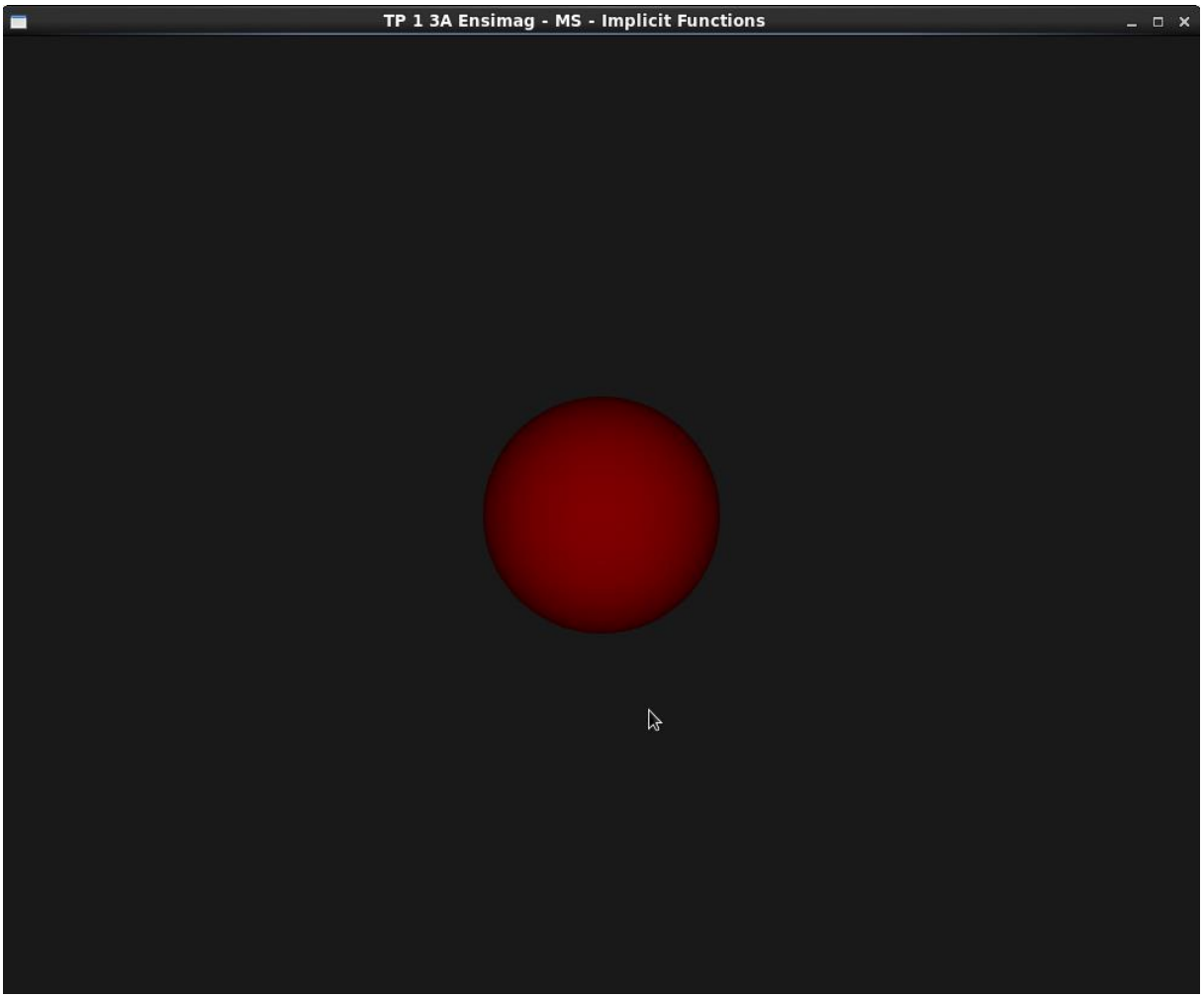
L'équation cartésienne d'une sphère de rayon r et centrée à l'origine est :

$$x^2 + y^2 + z^2 = r^2$$

Dans le cas de la sphère, le champ scalaire correspond à la distance entre le point considéré et le centre de la sphère. On en a déduit la fonction implicite f de l'espace qui correspond à :

$$f(x, y, z) = \sqrt{x^2 + y^2 + z^2}$$

Dans notre cas, nous retournons donc la distance à l'origine. Ici l'isopotentiel représente le rayon de la sphère, nous avons choisi pour cet exemple un isopotentiel k = 0.5



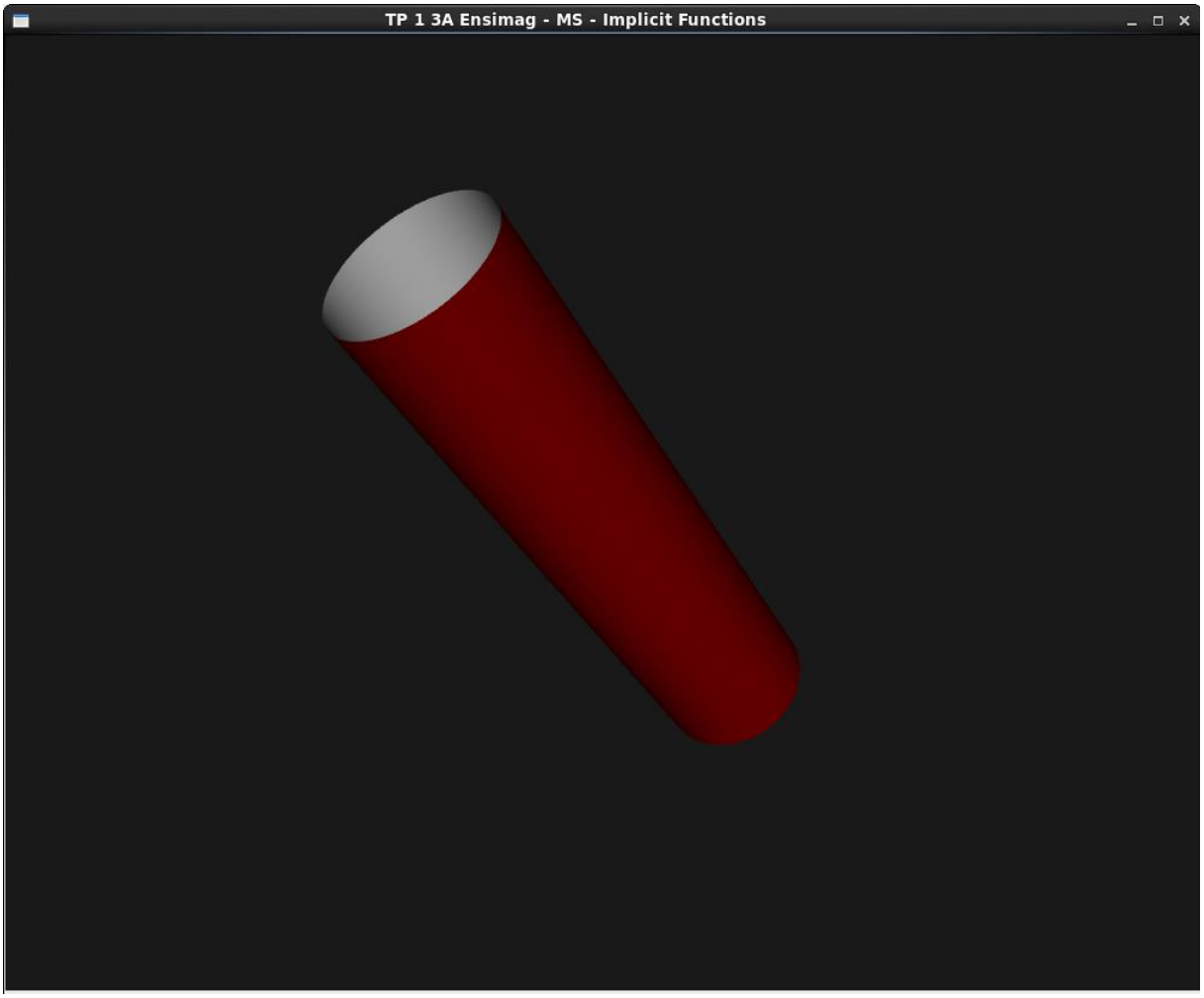
Eval

```
float MyImplicitSphere::Eval(glm::vec3 p) const{
   return glm::length(p);
}
```

EvalDev

```
glm::vec3 MyImplicitSphere::EvalDev(glm::vec3 p) const{
   return p;
}
```

- Le cylindre



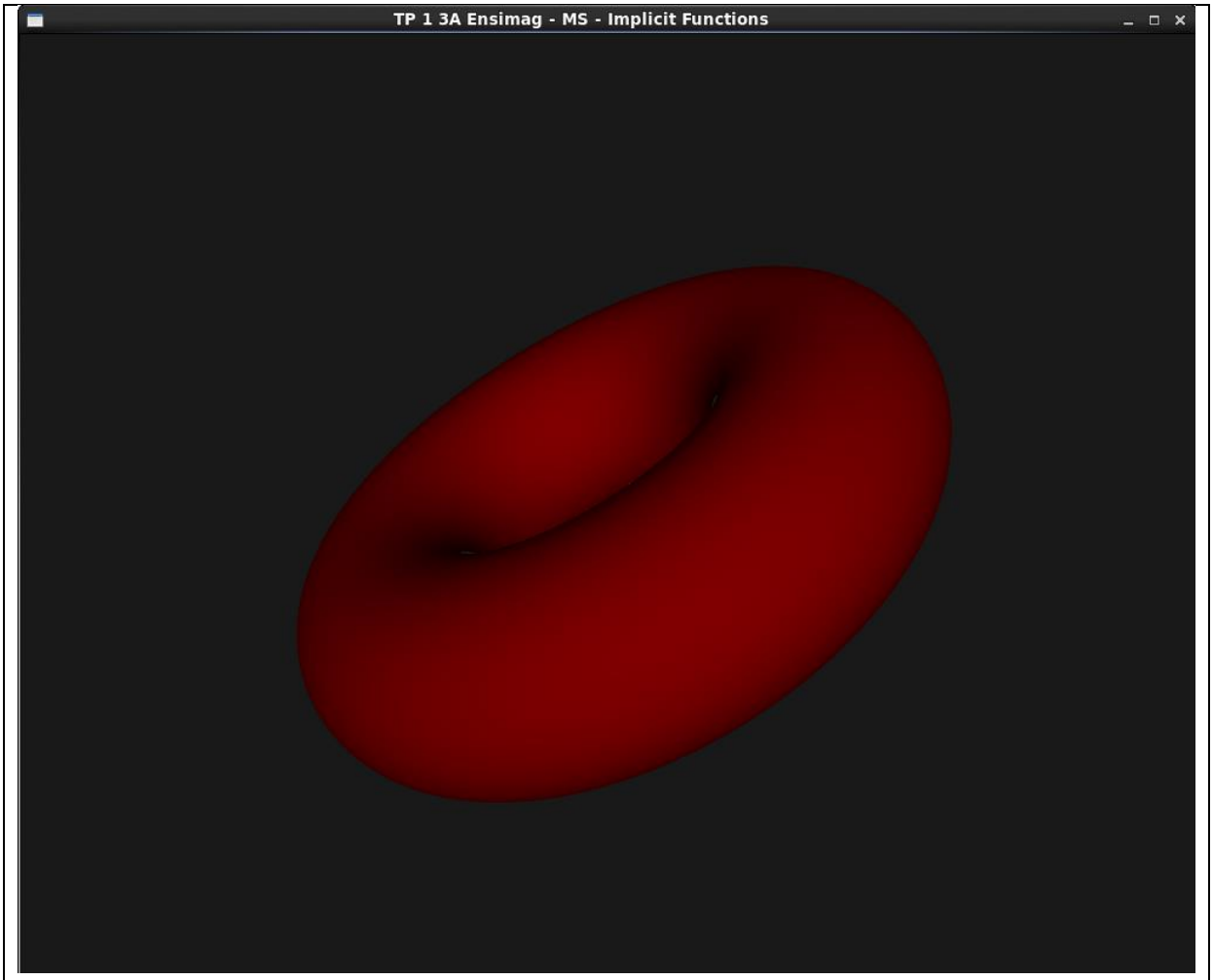
Eval

```
float MyImplicitCylindre::Eval(glm::vec3 p) const{
  float res = 0;
  res = sqrt(p.x * p.x + p.y * p.y);
  return res;
}
```

EvalDev

```
glm::vec3 MyImplicitCylindre::EvalDev(glm::vec3 p) const{
  vec3 res(p.x, p.y, 0);
  return res;
}
```

- Le tore



Eval

```
float MyImplicitTore::Eval(glm::vec3 p) const{
  // f(x,y,z) = (x² + y² + z² + R² - r²)² - 4 R² (x² + z²)
  float k = (p.x * p.x + p.y * p.y + p.z * p.z + R * R - r * r);
  float res = k*k - 4*R*R*(p.x * p.x + p.z * p.z) ;

  return res;
}
```

EvalDev

```
glm::vec3 MyImplicitTore::EvalDev(glm::vec3 p) const{
  glm::vec3 res;
  res.x = 4 * p.x * (p.x * p.x + p.y * p.y + p.z * p.z + R * R - r * r) - 8 * R * R * p.x;
  res.y = 4 * p.y * (p.x * p.x + p.y * p.y + p.z * p.z + R * R - r * r);
  res.z = 4 * p.z * (p.x * p.x + p.y * p.y + p.z * p.z + R * R - r * r) - 8 * R * R * p.z;
  return res;
}
```

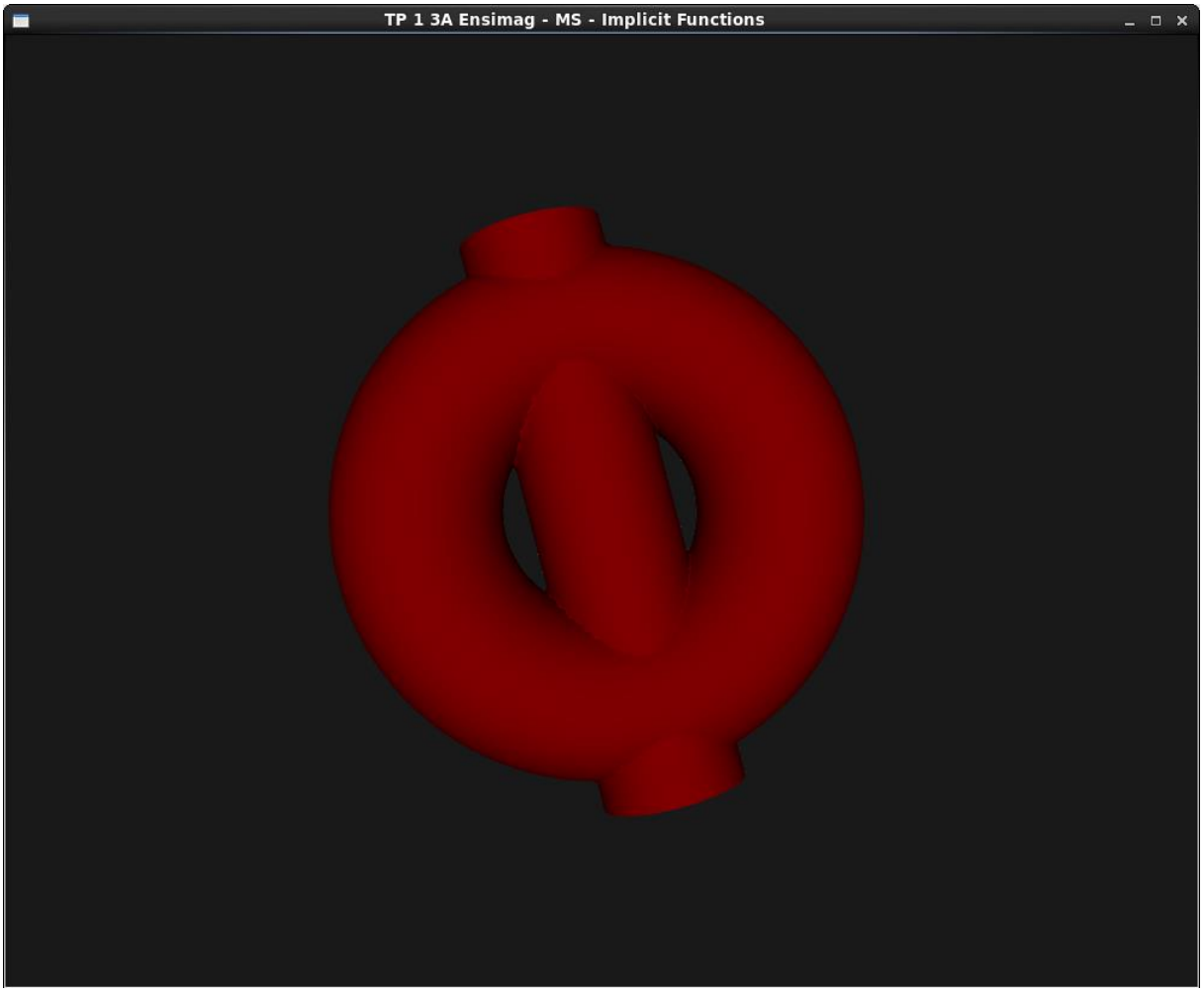
2. Opérateur sur les fonctions implicites

Une fois les primitives créées, nous avons défini des opérateurs pour combiner plusieurs fonctions implicites. Sur le même principe d'héritage que précédemment, chaque classe implémentant un opérateur hérite de la classe MyImplicitFunction. Un opérateur prend en entré 2 fonctions implicites que l'on a appelé f1 et f2. La redéfinition des fonctions Eval et EvalDev va représenter la manière de combiner ces deux fonctions.

On a ainsi écrit deux opérateurs qui sont : l'union et l'intersection de deux fonctions implicites.

L'union retourne le minimum des deux fonctions f1 et f2 pour Eval et EvalDev. Tandis que l'intersection retourne le maximum. Les exemples ci-dessous ont été effectués avec le tore et le cylindre.

- L'union



Eval

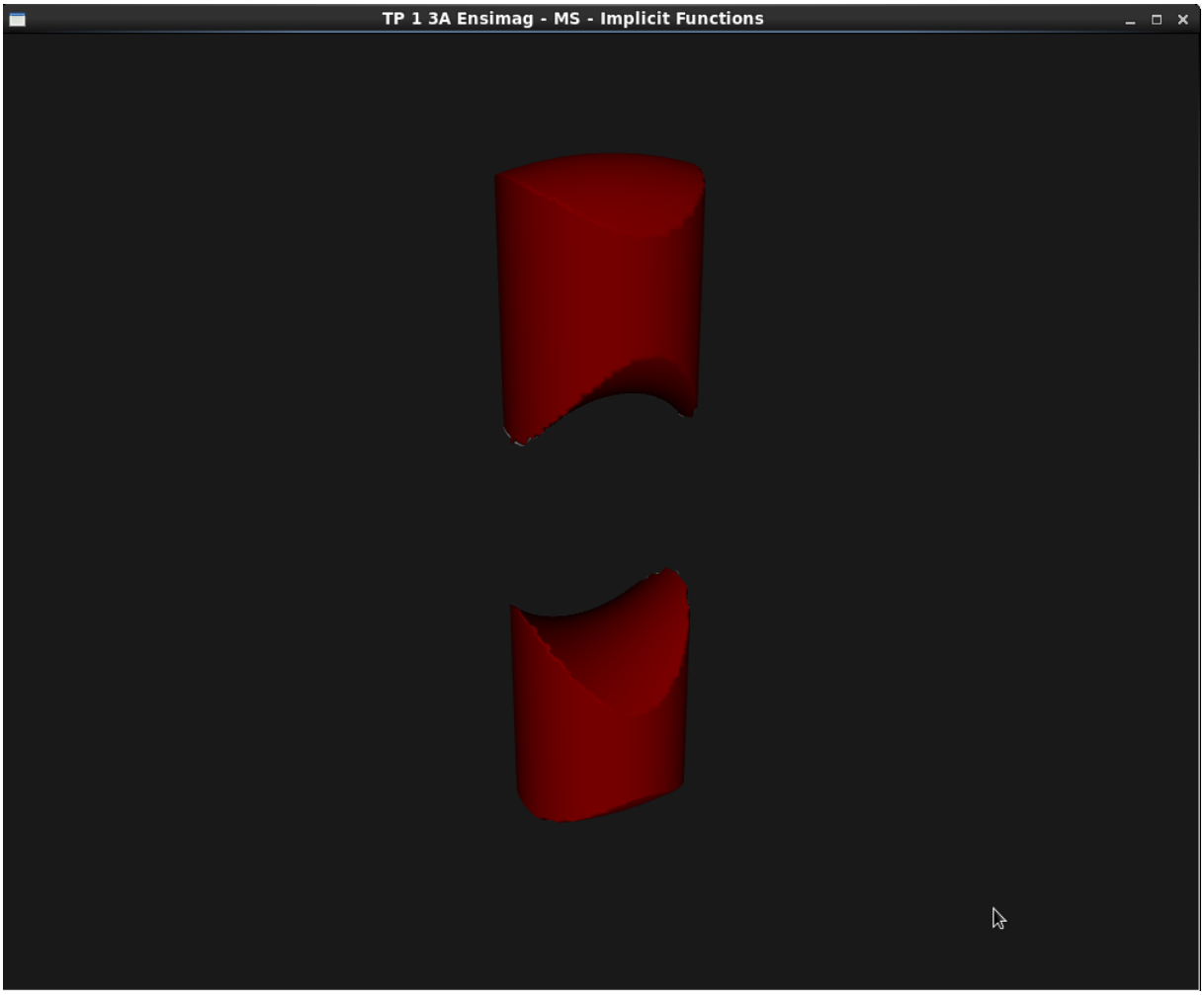
```
float MyImplicitAddition::Eval(glm::vec3 p) const{
  //float res = max( f1->Eval(p), f2->Eval(p) );
```

```
  float res1 = f1->Eval(p);
  float res2 = f2->Eval(p);
  if( res1 <= res2){
    return res1;
  }
  else{
    return res2;
  }
}
```

EvalDev

```
glm::vec3 MyImplicitAddition::EvalDev(glm::vec3 p) const{
  float res1 = f1->Eval(p);
  float res2 = f2->Eval(p);
  if( res1 <= res2){
    return f1->EvalDev(p);
  }
  else{
    return f2->EvalDev(p);
  }
}
```

- L’intersection



Eval

```
float MyImplicitAddition::Eval(glm::vec3 p) const{
  //float res = max( f1->Eval(p), f2->Eval(p) );
  float res1 = f1->Eval(p);
  float res2 = f2->Eval(p);
  if( res1 <= res2){
    return res1;
  }
  else{
    return res2;
  }
}
```

EvalDev

```
glm::vec3 MyImplicitAddition::EvalDev(glm::vec3 p) const{
  float res1 = f1->Eval(p);
  float res2 = f2->Eval(p);
  if( res1 <= res2){
```

```
    return f1->EvalDev(p);
  }
  else{
    return f2->EvalDev(p);
  }
}
```