

Détections de collisions avec des SDFs

Paul Luneau, CNAM ENJMIN

2025

Master jeux et médias interactifs numériques, parcours programmation
Cohabitation par le Conservatoire National des Arts et des Métiers et
l'Université de Poitiers

Sommaire

1	Introduction	2
2	Définitions et concepts	2
2.1	SDF	2
2.2	Primitives et opérateurs	2
2.3	Gradient	3
2.4	Descente de gradient	5
3	Collisions entre les SDF et différents objets	6
3.1	Collisions SDF-Cercle	6
3.2	Collisions SDF-Triangle	6
3.3	Collisions SDF-SDF	8
4	Conclusion	9
5	Implémentation	10
	Bibliographie	10



le **cnam**
enjmin



1 Introduction

Les Signed Distance Fields (SDFs) sont une manière alternative aux meshes de représenter des objets. Ils permettent de facilement combiner, soustraire, déformer des objets ce qui les a rendu très populaires pour rendre des scènes.

Les SDFs étant aussi très efficaces pour connaître la distance entre un point et un objet et pour déterminer si ce point est à l'intérieur ou à l'extérieur de celui ci, ils peuvent être des manières efficaces de représenter des objets dans les simulations physiques et d'obtenir facilement des mesures comme la pénétration d'un objet dans un autre.

Cependant, leur nature continue et le peu d'autres informations que les SDFs fournissent à priori semblent compliquer le calcul de collisions. Se pose alors la question “Comment parvenir à détecter les collisions entre des objets décrits par des SDFs et d'autres objets ?”. Pour répondre à cette question, nous commencerons par définir les concepts que nous utiliserons puis nous tenterons de répondre à cette question en présentant des méthodes pour différents types d'objets de complexité croissante : des cercles, des triangles et enfin d'autres objets décrits par des SDFs.

2 Définitions et concepts

2.1 SDF

Un SDF ou Signed Distance Field est une fonction $f(P) : \mathbb{R}^2 \rightarrow \mathbb{R}$ qui associe à chaque point P d'un espace la distance entre P la surface d'un objet. La distance est dite “signée” car $f(P) < 0$ pour les points à l'intérieur de l'objet. Par simplicité, nous nous limiterons ici à des SDF de $\mathbb{R}^2 \rightarrow \mathbb{R}$ mais il est tout à fait possible d'utiliser des SDF de $\mathbb{R}^3 \rightarrow \mathbb{R}$

Exemple: le SDF d'un cercle de rayon r et de centre $C(C_X; C_Y)$ (voir Fig.2a)

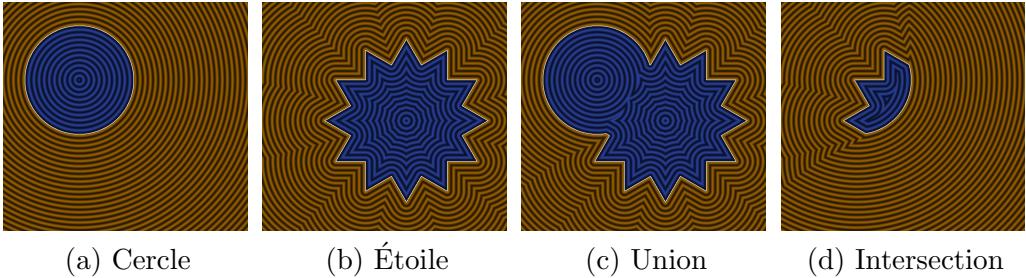
$$f(P) = \sqrt{(P_X - C_X)^2 + (P_Y - C_Y)^2}$$

2.2 Primitives et opérateurs

Une manière d'utiliser les SDFs pour décrire des objets est de partir de formes géométriques simples, des primitives. Ces primitives peuvent par exemple être un cercle, un rectangle, une étoile à n branches, un polygone quelconque... (pour une liste plus complète [1]).

À partir de ces primitives, il est possible d'appliquer des opérations pour obtenir d'autres SDFs, représentant l'union, la soustraction, l'intersection entre plusieurs SDFs plus simples.

Figure 2: Différents SDFs, combinaisons et opérations



2.3 Gradient

Le gradient $\Delta f(P) : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ d'un SDF $f(P)$ est une fonction telle que $\|\Delta f(P)\| = 1$ et qui donne la direction dans laquelle $f(P)$ augmente le plus vite.

Pour obtenir ce gradient, j'ai d'abord essayé de remplacer complètement les SDFs par d'autres fonctions $g(P) : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ telles que $g(P)$ est le vecteur entre P et le point le plus proche à la surface de l'objet. Ainsi, il suffirait de trouver une telle fonction pour en déduire le SDF et son gradient. En effet $f(P) = \|g(P)\|$ et $\Delta f(P) = \frac{-g(P)}{\|g(P)\|}$. Je n'ai malheureusement pas obtenu un résultat satisfaisant avec cette méthode car à part pour quelques primitives simples comme un cercle ou un rectangle, je n'ai pas réussi à trouver de telles fonctions. Je n'ai pas non plus su comment bien gérer les opérations de combinaison entre ces fonctions.

J'ai ensuite essayé de revenir aux SDFs et d'évaluer f à 4 points autour du point P pour lequel on cherche $\Delta(P)$ puis de calculer la différence le long les axe X et Y ainsi :

$$\begin{aligned}
\vec{u} &= \begin{pmatrix} o & 0 \end{pmatrix} \\
\vec{v} &= \begin{pmatrix} 0 & o \end{pmatrix} \\
\vec{d}_0 &= \begin{pmatrix} f(P+u) & f(P+v) \end{pmatrix} \\
\vec{d}_1 &= \begin{pmatrix} f(P-u) & f(P-v) \end{pmatrix} \\
\Delta f(P) &= \frac{\vec{d}_0 - \vec{d}_1}{\|\vec{d}_0 - \vec{d}_1\|}
\end{aligned}$$

Cette méthode a l'avantage d'être assez rapide à calculer mais le résultat présente beaucoup d'artefacts (*voir Fig. 3*).

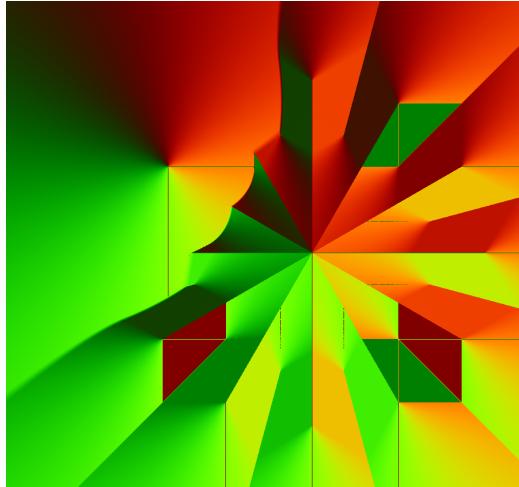


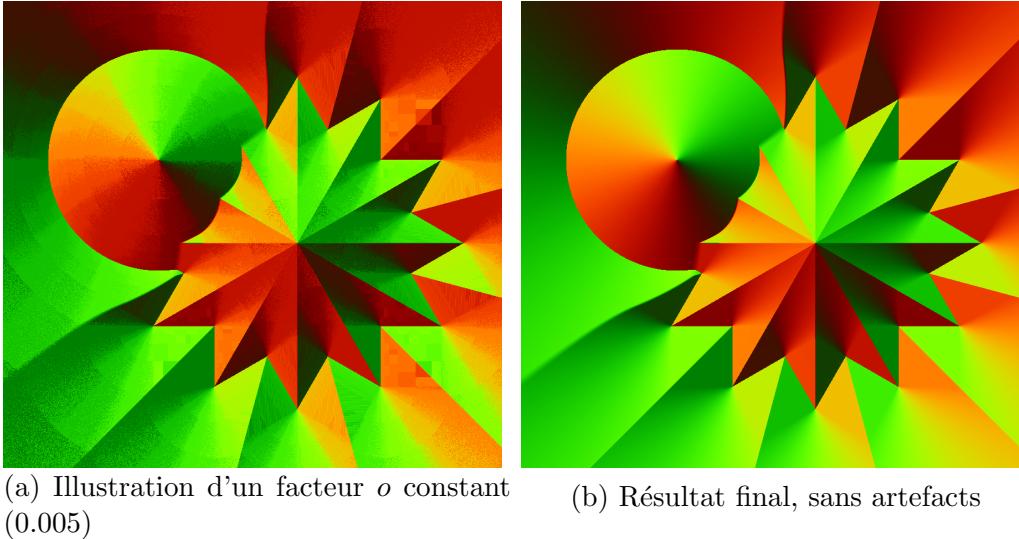
Figure 3: Première méthode de calcul du gradient, présentant des artefacts

Finalement, la méthode qui a fini par fonctionner consiste à évaluer le SDF à 3 points autour de P et par dichotomie, converger vers l'angle α vers lequel $f(P')$ est le plus grand, avec $P' = (Px + \cos(\alpha) \times o ; Py + \sin(\alpha) \times o)$. Le facteur o est la distance par rapport à P à laquelle on évalue $f(P')$. Le choix de cette valeur change beaucoup la qualité du résultat (*voir Fig. 4a*) et d'après les tests que j'ai pu faire, plus la valeur de $f(P)$ est importante, plus le facteur o doit l'être aussi pour un obtenir bon résultat. Dans l'implémentation finale o est défini ainsi :

$$o = \max \left(0.005, \frac{f(P) \times 0.005}{20} \right)$$

Cette formule n'a pas été beaucoup testé et demanderait sûrement de l'affinage.

Figure 4: Résultat



2.4 Descente de gradient

La descente de gradient est un algorithme servant à trouver un minimum d'une fonction f dont on connaît le gradient Δf . Elle consiste à partir d'un premier point P_0 et d'itérer dessus de la manière suivante :

$$P_{n+1} = P_n - \alpha \Delta f(P_n)$$

Le facteur d'apprentissage α est une valeur qui détermine à quelle vitesse on descend le gradient. Si sa valeur trop faible, la descente est très lente, demandant beaucoup d'itérations mais si elle est trop haute, on risque de dépasser un minimum et d'osciller autour sans se stabiliser.

Généralement, on arrête d'itérer si $\Delta f(P_n) < \varepsilon$, avec ε une petite valeur, signifiant qu'on a atteint un minimum de la fonction.

3 Collisions entre les SDF et différents objets

3.1 Collisions SDF-Cercle

Les collisions entre un SDF f et un cercle de rayon r et de centre C sont les plus simples à détecter. Du fait que tous les points du cercle sont à la même distance du centre, on peut ignorer la direction dans laquelle se trouve le point le plus proche sur le SDF. Il suffit alors d'imaginer que le cercle est un point et que l'intérieur du SDF est l'ensemble des points P pour lesquels $f(P) < r$. Ou autrement dit, si $f(C) < r$, c'est que le cercle et le SDF se chevauchent.

3.2 Collisions SDF-Triangle

Pour un triangle, il ne suffit pas d'évaluer le SDF à chacun de ses sommet, en effet, il existe des configurations pour lesquelles le SDF et le triangle se chevauchent mais où aucun des sommet n'est dans le SDF.

On peut alors chercher le point P du triangle pour lequel $f(P)$ est le plus petit, soit le point le plus proche ou profondément enfoncé dans le SDF et vérifier si $f(P) < 0$. L'approche naïve serait de partir du centre du triangle (la moyenne de ses sommets) et par descente de gradient, trouver des points P_n pour lessquels $f(P_n)$ est de plus en plus petit, tout en contraignant P_n au triangle.

Cette méthode est cependant loin d'être parfaite :

- La descente de gradient a tendance à trouver des minimum locaux et pas le minimum global ce qui fait que le résultat n'est pas forcément fiable et dépend du point de départ
- Pour que la descente soit rapide quand le point est loin du SDF, il est préférable de multiplier le facteur d'apprentissage par $f(P)$. Cependant, en se rapprochant de la surface de l'objet, $f(P)$ va se rapprocher de 0 et donc ralentir la descente alors que l'objectif est de rentrer dans l'objet. Ainsi, s'il n'y a pas assez d'itérations, il est commun que le point trouvé soit à la surface de l'objet. On peut essayer de palier à cela en multipliant a par $\max(f(P), \mu)$ au lieu de $f(P)$ avec μ une valeur minimum de descente mais fixer cette valeur est assez arbitraire et dépendant de la situation.
- La méthode est très coûteuse en terme de temps de calcul. Dans mes tests, le minimum pour avoir un résultat correct était autour de 64 itérations pour la descendre de gradient et pour chaque itération, on

calcule le gradient du SDF avec la méthode proposée précédemment, nécessitant d'évaluer le SDF une dizaine de fois, soit un total de plus de 640 évaluations du SDF.

Macklin et al. [2] proposent une version améliorée de l'algorithme :

Pour un triangle ayant pour sommets $a, b, c \in \mathbb{R}^2$ commencer par exprimer P en coordonnées barycentriques u, v, w telles que :

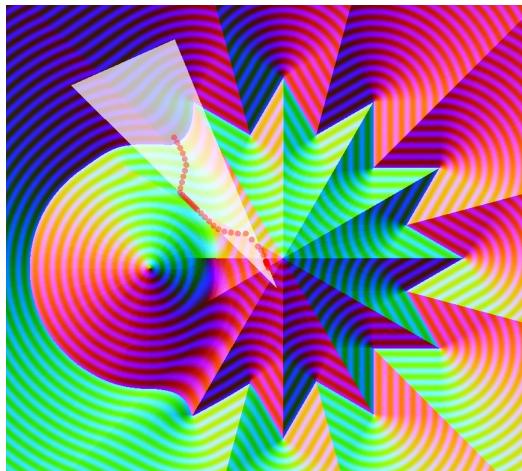
$$P = ua + vb + wr \quad (1)$$

$$\text{avec } u, v, w \geq 0 \quad (2)$$

$$\text{et } u + v + w = 1 \quad (3)$$

Puis chercher à minimiser $f(ua + vb + wr)$ grâce à la descente de gradient projetée, une variation de la descente de gradient où à chaque itération, le nouveau point est “projété” sur l’ensemble des valeurs possibles, défini par les contraintes que l’on a posé précédemment (2-3). Dans ce contexte, ça revient à contenir le point dans le triangle.

Figure 5: Descente projetée dans un triangle le long de Δf en partant du centre du triangle



Cependant, ces méthodes ne permettent de trouver qu'un seul point de contact par triangle alors qu'il est tout à fait possible qu'un triangle soit en contact avec un SDF en plusieurs points. Il est donc nécessaire d'avoir une densité de triangle importante pour obtenir des résultats fiables. J. Pelletier-Guénette, A. Mercier-Aubin et S. Andrews proposent dans un article de 2025 une méthode de subdivision automatique des triangles pour palier à cela [3].

Pour optimiser la détection des collisions, il est possible de commencer par vérifier si le cercle circonscrit au triangle chevauche le SDF et ainsi d'éliminer beaucoup de vérifications et d'évaluations du SDF.

3.3 Collisions SDF-SDF

Déterminer les intersections entre deux SDFs est un sujet plus complexe, notamment pour des questions d'optimisations et pour s'assurer que toutes les intersections sont bien trouvées. Nous nous limiterons ici à voir comment trouver des points faisant partir des deux SDFs, s'il en existe. Un article de Pengfei Liu, Yuqing Zhang et al. [4], d'où l'algorithme suivante provient, propose un aperçu plus complet des problématiques que cette question pose.

Ainsi, trouver un point d'intersection entre deux SDFs f_a et f_b revient à chercher un point P tel que $f_a(P) \leq 0$ et $f_b(P) \leq 0$.

On peut alors prendre un point de départ P_0 , le projeter sur la surface de f_a si $f_a(P_0) > 0$ pour s'assurer que le point est dans f_a et, par descente de gradient projetée le long de Δf_b ainsi :

$$P_{n+1} = P_n - \alpha \Delta f_b(P_n), \text{ avec } \alpha \text{ le facteur d'apprentissage}$$

Et en projetant P_{n+1} sur f_a de sorte que $f_a(P_{n+1}) \leq 0$

Cette projection sur f_a peut être faite grâce à l'application suivante :

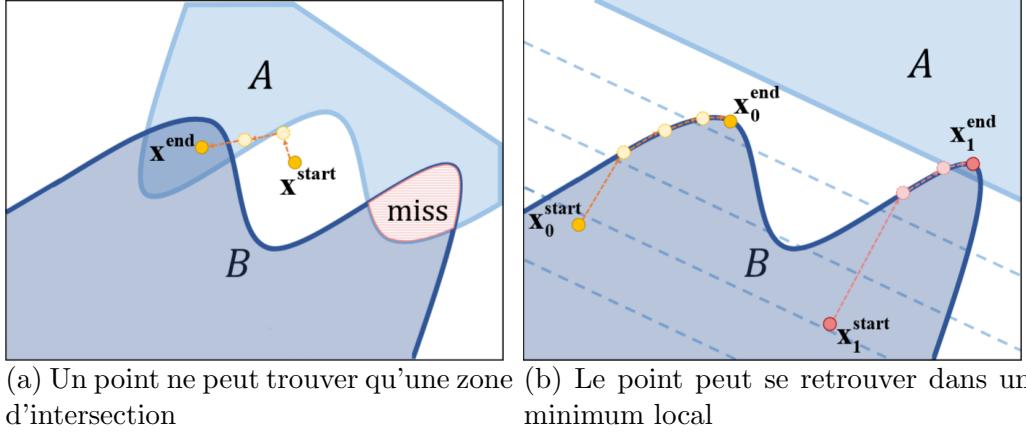
$$V(P) = \begin{cases} P, & \text{si } f_a(P) \leq 0 \\ P - \Delta f_a(P)f_a(P), & \text{sinon} \end{cases}$$

La condition pour arrêter d'itérer est ici soit :

- $\|P_{n+1} - P_n\| < \varepsilon$, ce qui signifie que l'on ne peut plus vraiment descendre, soit parce que l'on est dans un minimum de f_b , soit que continuer à descendre le long de Δf_b nous fait sortir de f_a
- $f_a(P_n) < 0$ et $f_b(P_n) < 0$, ce qui signifie que P_n est un des points à l'intersection des deux SDFs

S'il y a plusieurs intersections entre les deux SDFs, elles seront ignorées si un seul point de départ est utilisé. Également, utiliser plusieurs points de départ peut permettre de ne pas se retrouver bloqué dans un minimum local, ignorant les intersections potentielles *voir Fig.(6)*

Figure 6: Limitations (*Illustration de Pengfei Liu, Yuqing Zhang et al. [4]*)



4 Conclusion

Ainsi, bien que de manière non-exhaustive, nous avons pu présenter plusieurs méthodes de détections des collisions entre des objets et des SDFs.

En regardant les dates de parution des articles sources (2020, 2024, 2025), on voit que c'est un sujet encore actif dans la recherche et on peut espérer voir des avancées et l'amélioration des algorithmes de détection des collisions dans les prochaines années.

De mon côté, j'ai beaucoup appris avec ce projet et j'aimerais continuer à travailler dessus afin d'améliorer et optimiser le code déjà présent ainsi que d'aller plus loin en implémentant un petit moteur physique 2D au lieu de simplement détecter les collisions.

5 Implémentation

Dépot Git de l'implémentation du projet, d'où proviennent la plupart des illustrations :

<https://github.com/paulluneaug/SDFPhysics>

Bibliographie

- [1] Inigo Quilez. *2D distance functions*. URL: <https://iquilezles.org/articles/distfunctions2d/>.
- [2] Macklin et al. “Local Optimization for Robust Signed Distance Field Collision”. In: (2020). URL: <https://mmacklin.com/sdfcontact.pdf>.
- [3] A. Mercier-Aubin et S. Andrews J. Pelletier-Guénette. “Real-Time Triangle-SDF Continuous Collision Detection”. In: (2025). URL: https://profs.etsmtl.ca/sandrews/pdf/SCA25_TriSDF_ccd.pdf.
- [4] Yuqing Zhang et al. Pengfei Liu. “Real-time Collision Detection between General SDFs”. In: (2024). URL: http://www.cad.zju.edu.cn/home/jin/papers/Real_Time_CD_between_SDFs.pdf.