修士論文　　2023 年度

リアルタイムグラフィックスにおける
回転剛体の連続的衝突判定精度向上に関する研究

東京工科大学大学院

バイオ・情報メディア研究科

メディアサイエンス専攻

山本　輝

修士論文　　2023 年度

リアルタイムグラフィックスにおける
回転剛体の連続的衝突判定精度向上に関する研究

指導教員　　渡辺 大地 教授

東 京 工 科 大 学 大 学 院

バ イ オ ・ 情 報 メ デ ィ ア 研 究 科

メ デ ィ ア サ イ エ ン ス 専 攻

山本　輝

論 文 の 要 旨

論文題目	リアルタイムグラフィックスにおける 回転剛体の連続的衝突判定精度向上に関する研究
執筆者氏名	山本　輝
指導教員	渡辺 大地 教授
キーワード	リアルタイムグラフィックス、コンピュータゲーム、回転運動、 衝突判定、干渉判定

[**要旨**]

今日、コンピュータゲームにおける剛体同士の衝突判定は、高速に判定を行える境界ボリュームが使用されることが多い。しかし、コンピュータ内での剛体の移動が離散的であるため、剛体の運動速度が非常に大きい場合は、剛体同士が衝突せずにすり抜けてしまう場合がある。この問題に対処するために様々な CCD(Continuous Collision Detection、連続的衝突判定) 手法が提案されており、銃弾などの運動速度の大きい剛体を扱う FPS(First Person Shooting) ゲームなどで利用されてきた。一例として Unity で採用されている Sweep-based CCD 手法と、Speculative CCD(以降、S-CCD) 手法があるが、Sweep-based CCD 手法は角運動を無視しているため回転運動に対応しておらず、S-CCD 手法は回転運動の場合に衝突判定精度が低いという問題がある。本研究ではこの問題を解決するために、直方体の回転運動軌跡の近似形状である扇形によって運動軌跡を補間し、回転剛体の衝突判定精度の向上を目的とした。ただし、扇形は非凸形状になる場合があり GJK アルゴリズムなどの凸形状に対する干渉判定手法を利用できない。そこで、本稿では扇形とコンピュータゲーム上で頻繁に登場する幾つかのプリミティブ形状との干渉判定を提案する。

提案手法と S-CCD 手法の衝突判定精度を検証によって比較すると、提案手法のほうが回転運動の回転角度や、回転運動を行う剛体である直方体の形状に関わらず高精度に衝突判定を行えていることが判明した。また、提案手法がリアルタイムグラフィックス上で使用可能であるかどうかを確認するため、提案手法の実行速度をコンピュータゲーム上で用いられる DCD(Discrete Collision Detection) 手法の一例である境界ボリューム手法と CCD 手法の一例である S-CCD 手法と比較したところ、一部の形状との干渉判定では遅い実行速度を示したものの、概ねリアルタイムグラフィックス上には問題ないことが判明した。

A b s t r a c t

Title	A Research on Improvement of Accuracy of Collision Detection for Rotaion Rigid Bodies In Real-Time Graphics
Author	Hikaru Yamamoto
Advisor	Taichi Watanabe
Key Words	Real-Time Graphics, Computer Game, Rotational Movement, Continuous Collision Detection, Discrete Collision Detection

[**summary**]

Today, collision detection between rigid bodies in computer games often uses Bounding Volumes that allow for fast decision making. However, because rigid bodies move discretely in the computer, they are tunneling each other without colliding if their motion velocity is very large. Various CCD (Continuous Collision Detection) methods have been proposed to address this problem and have been used in FPS (First Person Shooting) games that handle rigid bodies with large velocities such as bullets. One example is the Sweep-based CCD method and the Speculative CCD (hereinafter referred to as S-CCD) method used in Unity. However, the Sweep-based CCD method ignores angular motion and does not support rotational motion, while the S-CCD method has low collision detection accuracy for rotational motion. To solve this problem, this study interpolates the motion trajectory by using a fan shape, which is an approximate shape of the rotational motion trajectory of a rectangular body, to improve the accuracy of collision detection for rotating rigid bodies. However, fan shapes are sometimes non-convex and cannot use interference determination methods for convex shapes such as the GJK algorithm. Therefore, this paper proposes a collision detection method between a fan shape and some primitive shapes that frequently appear in computer games.

Comparison of the accuracy of collision detection between Our method and the S-CCD method through verification revealed that Our method is more accurate in detecting collisions regardless of the rotation angle of the rotational motion and the shape of the rectangular body that is the rigid body performing the rotational motion. In order to confirm whether this method can be used for real-time graphics, we compared the execution speed of this method with the boundary volume method, which is an example of the DCD (Discrete Collision Detection) method used in computer games, and the S-CCD method, which is an example of the CCD method. The results showed that although the method showed slower execution speed when determining collision with some shapes, there were generally no problems in real-time graphics.

目　次

図 目 次

第 1 章

はじめに

1.1 背景と目的

今日のコンピュータゲームで実装されている剛体同士の衝突は、様々な手法によって判定を行っている。その一つに、境界ボリュームを用いた手法 [1] がある。境界ボリュームは、AABB(Axis-Aligned Bounding Box、軸並行境界ボックス)[2] や OBB(Oriented Bounding Box、有向境界ボックス)[3]、球、カプセル形状 [4]、k-DOPs[5] などがあり、その特徴は、同一形状同士の干渉判定が容易であり、かつ計算が高速なことである。そのため、Unity などのゲームエンジンでも採用されている手法である。

また、Gilbert らにより提案された GJK アルゴリズム [6][7] は、境界ボリュームよりも剛体の本来の干渉判定範囲に近い凸形状同士の干渉判定を高速に行えるため、コンピュータゲームや Bullet[8] 等の物理エンジンライブラリなどに利用されてきた。

GJK アルゴリズム同様に、凸形状同士の干渉判定を行う手法として SAT(Separating Axis Theorem、分離軸定理) 手法 [9] があり、凸形状同士の干渉判定には、どちらかが使用されている場合が多い。

境界ボリューム手法はその高速性故に様々な手法の高速化に用いられてきた。コンピュータゲーム中には壁や床、プレイヤーキャラクターなどの様々な剛体が多数存在する。これらの剛体すべてに対して GJK アルゴリズムや SAT 手法などの詳細な干渉判定を行ってしまうと、膨大な計算量となってしまう。そこで、境界ボリューム手法で干渉していると判定した剛体同士のみに詳細な干渉判定を行うようにすることで、全体の計算量を削減できる。このような高速化の手法は BVH(Bounding Volume Hierarchy)[10][7] と呼ばれ、AABB を用いた AABBtree[11] や、OBB を用いた OBBtree[12] などの様々な手法がある。BVH はレンダリングにおけるレイトレーシングの高速化 [13] 等にも用いられている。

境界ボリューム手法や GJK アルゴリズムのように、ある時間における剛体の位置をもとに干渉判定を行う手法を DCD(Discrete Collision Detection、離散的衝突判定) と呼ぶが、これらの手法では剛体同士がすり抜けてしまう場合がある。それは、運動を行う剛体の運動速度が非常に大きい場合である。コンピュータゲーム内での剛体の運動は、1 フレームごとに処理を行うため、剛体の運動速度が大きい場合には本来衝突する予定の座標を通り過ぎてしまい、DCD 手法では剛体同士の衝突を検知できない場合がある。

剛体の運動速度が大きい場合には、剛体の移動軌跡などから他剛体との衝突判定を行う手法があり、CCD(Continuous Collision Detection、連続的衝突判定)[14] と呼ぶ。コンピュータグラフィックスの分野の中でも特に医療シミュレーションやロボットシミュレーションなどの高精度なシミュレーションを必要とする分野で研究が行われてきた [15][16][17]。

しかし、CCD 手法は DCD 手法よりも計算コストが高く、コンピュータゲームにおいては、計算速度を優先するために衝突判定精度は低くしている場合が多い。コンピュータゲームにおける CCD 手法の一例として、NVIDIA 社の PhysX エンジンを使用する Unity では、Sweep-based CCD と Speculative CCD(以降、S-CCD) という手法 [18][19] がとられている。

Sweep-based CCD 手法は剛体の平行移動を想定した手法であり、剛体の運動方向に沿って剛体の形状をスイープさせることによって、連続的な衝突判定を可能にしている。しかし、この手法は剛体の角運動を無視するため、剛体が回転運動を行う場合に対処できない。

Sweep-based CCD 手法の類似手法に、Swept Sphere Volume(以降、SSV) 手法 [20] がある。運動前の剛体を包み込む球状境界ボリュームを運動後の位置までスイープすることによってカプセル形状を構成し、衝突判定を行う手法である。そのため、FPS(First Person Shooting) ゲームの銃弾のように、高速で動く剛体を扱うコンピュータゲームで使用する手法である [21]。SSV 手法も、剛体の角運動は取り扱わない。

S-CCD 手法はこれら 2 つの手法とは異なり、並進運動のみではなく角運動にも対処した手法である。しかし、S-CCD は運動前の剛体と運動後の剛体を AABB で囲むことによって衝突判定を行うため、本来の運動軌跡よりも広範囲に境界ボリュームを拡張してしまう。その為、本来の運動軌跡では衝突するはずのない剛体に対して、誤って衝突したと判定してしまう可能性がある。並進運動の場合には Sweep-based CCD 手法や SSV 手法を使用することで衝突判定精度を高精度に維持しながら衝突判定を行えるが、回転運動の場合には S-CCD 手法を使用するため、衝突判定精度を維持しにくい。

本研究ではこの問題に対処するため、回転運動軌跡の近似形状である扇形によって運動軌跡の補間を行い、回転運動を行う剛体の衝突判定精度が向上することを目的とした。

扇形は非凸の形状になる場合があるため、凸形状であれば干渉しているかどうかを判定できる GJK アルゴリズムや SAT 手法などを使用できない。そこで、本稿では扇形とコンピュータゲー

ムに頻出する球やカプセルといったプリミティブ形状との干渉判定を提案する。

そして、本研究では回転運動を考慮した CCD 手法である S-CCD 手法と、扇形による運動軌跡の補間の衝突判定精度を比較し、S-CCD 手法よりも衝突判定精度が向上したかを確認した。その結果、提案手法はどのような回転角度、直方体の形状においても S-CCD 手法よりも高精度に衝突判定を行えたことが判明した。また、提案手法はリアルタイムグラフィックス上での使用を想定しているため、実行速度を調査し、コンピュータゲーム上で使用される DCD 手法の一例である境界ボリューム手法、S-CCD 手法と実行速度を比較し、リアルタイムでの使用に問題がないかどうかを検証した。その結果、点、円、球との干渉判定は非常に高速であるが、長方形、カプセルとの干渉判定は他手法と比較しても遅い実行速度であることが判明した。S-CCD 手法との実行速度の比較では、実際に衝突したかどうかを判定する干渉判定の実行速度では提案手法のほうが劣るものの、運動軌跡の補間形状計算の実行速度では提案手法のほうが S-CCD 手法よりも優れており、2 つの実行速度を合計した場合、提案手法のほうが S-CCD 手法よりも高速に衝突判定を行えていることが判明した。

1.2 論文構成

本稿は全 5 章で構成する。第 2 章では、提案手法について述べ、第 3 章では提案手法について行った検証について述べる。第 4 章では考察を行い、第 5 章では研究のまとめを述べる。

第 2 章

提案手法

2.1　本研究における扇形の定義

本節では、回転運動軌跡の近似形状である扇形の本研究における定義と、扇形とプリミティブ形状の干渉判定について述べる。

始めに、回転運動により生成される扇形の定義について述べる。細長い剛体であればあるほど回転時に衝突を無視する可能性も高くなるため、回転を行う剛体は二次元平面においては長方形とし、三次元空間では直方体とする。本研究で想定する回転運動は、回転を行う剛体の中心点と回転中心点を通る直線と回転軸が回転を行う剛体のローカル座標軸と常に軸並行であるとし、回転中心点は回転を行う剛体の外部、または面上にあるとする。また、回転運動前後で回転中心点から回転を行う剛体の中心点までの距離は変わらないとする。図 2.1 に、本研究で想定する回転運動の一例を示す。

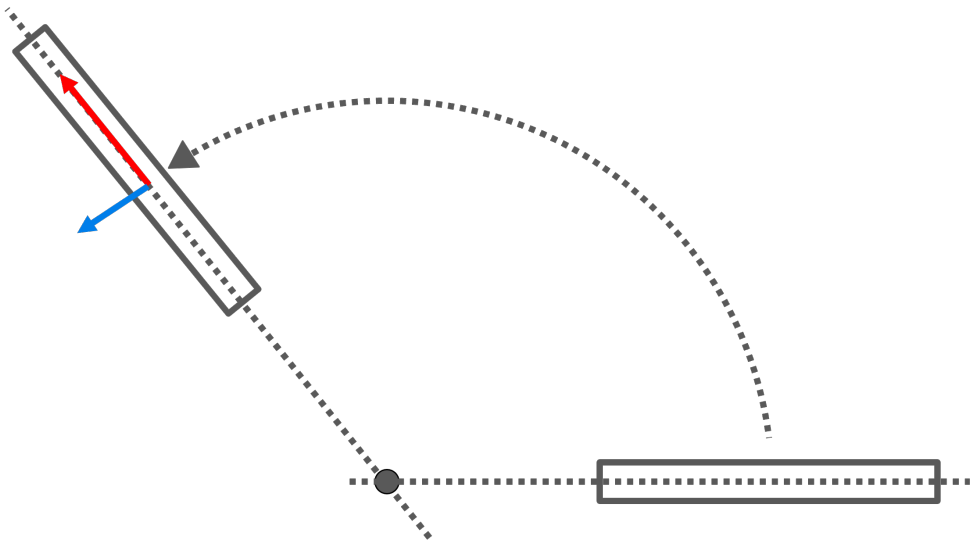

図 2.1　想定する回転運動の一例

また、これにより生成される扇形はピースケーキ型と、バウムクーヘン型を想定する。図 2.2 と図 2.3 にピースケーキ型とバウムクーヘン型の一例を示す。

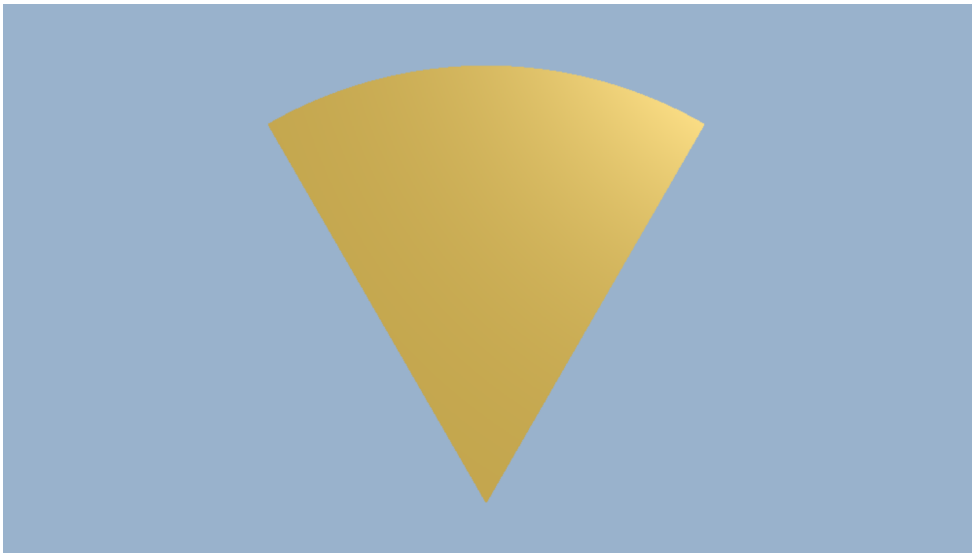

図 2.2　ピースケーキ型

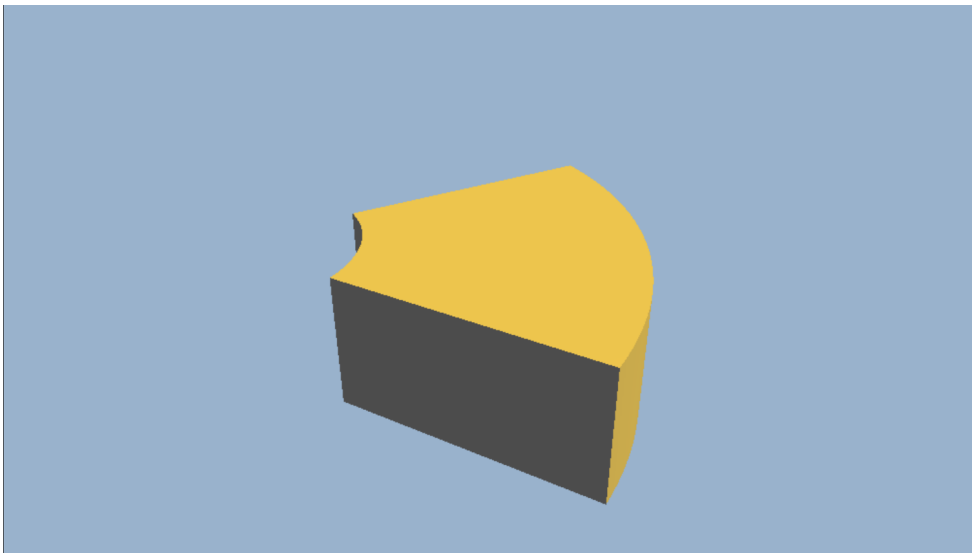

図 2.3　バウムクーヘン型

続いて、扇形を構成する要素について考える。二次元平面における扇形は Krishnan らが提案した Spherical Shell[22] と同一の形状となる。しかし、三次元空間における扇形は異なる形状であるため、本研究では Spherical Shell を参考に、扇形の構成要素を以下のように定義した。

- 回転中心点は点 O とする
- 回転軸を表す単位ベクトルを **U** とする

- 中心軸を表す単位ベクトルを $\mathbf{A}$ とする
- 範囲角度の半角を θ とする
- 扇形を構成する 2 つの円のうち小さい円 (以降、内円) の半径を r、大きい円 (以降、外円) の半径を R とする
- 回転軸方向の厚みを $-h$ から h の $2h$ とする

上述した定義の模式図を図 2.4 に示す。

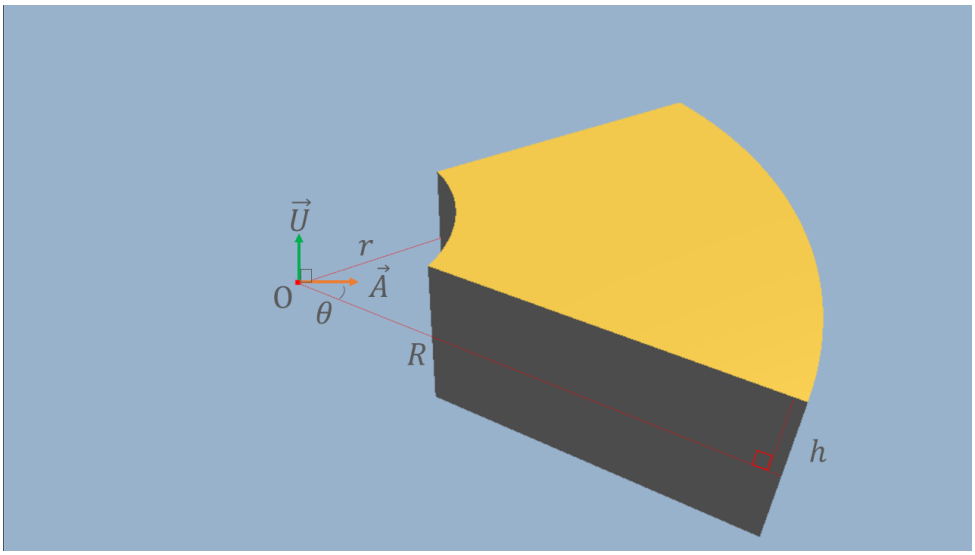

図 2.4　扇形の各点の定義図

続いて、直方体の回転運動から扇形を構成する方法を述べる。ここで、回転運動を行う前の直方体の中心点の位置ベクトルを $\mathbf{P}$、ローカル座標軸をそれぞれ $\mathbf{X}'$、$\mathbf{Y}'$、$\mathbf{Z}'$ とし、ローカル座標軸方向の直方体の厚みをそれぞれ W_x、W_y、W_z とする。ただし、前述した回転運動の制約から、回転軸 $\mathbf{U}$ は $\mathbf{Z}'$ と同一であり、回転中心点の位置ベクトルを $\mathbf{O}$ としたとき、$(\mathbf{P}-\mathbf{O})$ と $\mathbf{Y}'$ は同じ向きである。また、回転角度を α としたとき、扇形の各構成要素の値は式 (2.1) のようになり、中心軸ベクトル $\mathbf{A}$ はロドリゲスの回転公式 [23] を用いて、式 (2.2) から求められる。ただし、$\times$ は外積記号を表す。

$$
\begin{aligned}
\theta &= \frac{\alpha}{2} \ , \\
r &= |\mathbf{P} - \mathbf{O}| - \frac{W_y}{2} \ , \\
R &= |\mathbf{P} - \mathbf{O}| + \frac{W_y}{2} \ , \\
h &= \frac{W_z}{2} \ .
\end{aligned}
\tag{2.1}
$$

$$
\mathbf{A} = (\cos\theta)\mathbf{Y}' + (1 - \cos\theta)(\mathbf{Y}' \cdot \mathbf{U})\mathbf{U} + (\mathbf{U} \times \mathbf{Y}')\sin\theta \tag{2.2}
$$

次に、コンピュータゲームに頻出する幾つかのプリミティブ形状と扇形の干渉判定を定義する。判定数式を簡易化するため、本章では扇形を構成する要素の一部を次のように設定する。

- 回転中心点 O を原点とする
- 回転軸ベクトルを $\mathbf{U} = (0, 0, 1)$ とする
- 中心軸ベクトルを $\mathbf{A} = (1, 0, 0)$ とする

もし、扇形の回転中心点、回転軸ベクトル、中心軸ベクトルが上述の値ではない場合、座標変換を施すことで、本稿で提案する干渉判定手法を利用することが出来る。ここで、座標変換を施す点の位置ベクトルを Q とし、$\mathbf{U}$ と $\mathbf{A}$ に直行なベクトル $\mathbf{K}$ を $\mathbf{K} = \mathbf{U} \times \mathbf{A}$ としたとき、座標変換後の点の位置ベクトル $\mathbf{Q}'$ の各成分は式 (2.3) から算出できる。

$$
\begin{pmatrix} Q'_x \\ Q'_y \\ Q'_z \end{pmatrix} = \begin{pmatrix} A_x & A_y & A_z \\ K_x & K_y & K_z \\ U_x & U_y & U_z \end{pmatrix} \begin{pmatrix} Q_x - O_x \\ Q_y - O_y \\ Q_z - O_z \end{pmatrix} \tag{2.3}
$$

2.2 点との内外判定

本節では、任意の点 P が扇形の内部に存在するかどうか判定する方法について述べる。

点 P が三次元空間における扇形の厚み空間内部に存在する場合、始めに式 (2.4) を満たす。

$$
|P_z| \leq h \tag{2.4}
$$

続いて、扇形の回転軸から点 P までの距離が扇形の内円半径以上、外円半径以下であるかを判定する。

$$r^2 \leq {P_x}^2 + {P_y}^2 \leq R^2 \tag{2.5}$$

最後に、xy 平面において点 P が扇形範囲角度内に存在するかを判定する。まず、点 P を xy 平面に投影した点 P' の位置ベクトルと扇形の中心軸ベクトル $\mathbf{A}$ のなす角の余弦値 P_c を計算する。式 (2.6) から計算した P_c と扇形範囲角度の半角 θ の余弦値を比較し、点 P' が扇形範囲角度内に存在するかを判定する。

$$P_c = \frac{\mathbf{A} \cdot \mathbf{P'}}{|\mathbf{P'}|} = \frac{P_x}{\sqrt{{P_x}^2 + {P_y}^2}} \, , \tag{2.6}$$

$$P_c \geq \cos\theta \, . \tag{2.7}$$

式 (2.4)、式 (2.5)、式 (2.7) の条件式を満たすとき、点 P は扇形内部に存在する。

2.3 二次元平面における長方形との干渉判定

本節では、扇形と長方形の干渉判定手法について述べる。

長方形と扇形の干渉判定は、まず扇形を構成する線分、円と長方形を構成する線分の交差点を算出する。その後、第 2.2 項で述べる扇形と点の内外判定に交差点を用いることによって干渉判定を行う。交差点の算出には第 2.9.1 項で述べる線分同士の交差点算出 [24] と、第 2.9.2 項で述べる線分と円の交差点算出 [25] を利用する。

続いて、扇形が長方形に完全に内包されている場合や、長方形が扇形に完全に内包されている場合のように交差点が存在しない場合について述べる。これらの場合には、扇形内の任意の 1 点が長方形内部に存在するかどうかを第 2.2 節で述べた扇形と点の内外判定を用いて判定するか、または、長方形内の任意の 1 点が扇形内部に存在するかどうかを第 2.9.3 項で述べる長方形と点の内外判定を用いて判定することにより、扇形と長方形が干渉しているかどうかを判定する。

2.4　二次元平面における円との干渉判定

本節では、中心点が点 P、半径が r_c の円と扇形の干渉判定について述べる。

まず、点 P が扇形の範囲角度内に存在するかどうかを判定する。これは、式 (2.7) を利用する。式 (2.7) の結果が真であるなら、式 (2.5) を円の半径 r_c 分拡張させた式 (2.8) を満たすかどうか判定することで、扇形と円が干渉しているかどうかがわかる。

$$(r - r_c)^2 \leq {P_x}^2 + {P_y}^2 \leq (R + r_c)^2 \tag{2.8}$$

式 (2.5) の結果が偽であるなら、扇形を構成する 2 つの線分と円の中心点 P の最小距離を計算し、扇形と円が干渉しているかどうかを判定する。ここで、扇形の 2 つの線分をそれぞれ線分 L、M としたとき、線分 L の方向ベクトル $\mathbf{G}$ と、線分 M の方向ベクトル $\mathbf{H}$ は式 (2.9) のようになる。

$$\begin{cases} \mathbf{G} &= (\cos\theta, \sin\theta) \\ \mathbf{H} &= (\cos\theta, -\sin\theta) \end{cases} \tag{2.9}$$

線分 L の端点を点 G'、G" とし、線分 M の端点を点 H'、H" とすると、各端点の位置ベクトルは式 (2.10) のようになる。

$$\begin{cases} \mathbf{G}' &= r\mathbf{G} \\ \mathbf{G}'' &= R\mathbf{G} \\ \mathbf{H}' &= r\mathbf{H} \\ \mathbf{H}'' &= R\mathbf{H} \end{cases} \tag{2.10}$$

最後に、線分 L、M と円の中心点 P との最短距離を第 2.9.4 項で述べる点と線分の最短距離計算 [26][7] によって算出する。算出した最短距離の値が r_c 以下であるとき、扇形と円は干渉している。

2.5　三次元空間における球との干渉判定

本節では、中心点が点 P、半径が r_s の球と扇形の干渉判定について述べる。

始めに、球と扇形の厚み空間が干渉しているかを判定する。これは、式 (2.4) を球の半径分拡張させた式 (2.11) で判定を行う。

$$|P_z| \leq h + r_s \tag{2.11}$$

式 (2.11) を満たすとき、次の処理を行う。球は半径の異なる円が層状に重なった形状であると考えられることから、扇形の厚み空間内における円の最大半径 R_m を計算する。

球の中心点 P を式 (2.4) に代入した結果が真であるとき、R_m は球の半径 r_s と等しい。式 (2.4) の結果が偽であるとき、式 (2.12) により、R_m を求める。また、R_m についての模式図を図 2.5 と図 2.6 に表す。

$$R_m = \sqrt{{r_s}^2 - (|P_z| - h)^2} \tag{2.12}$$

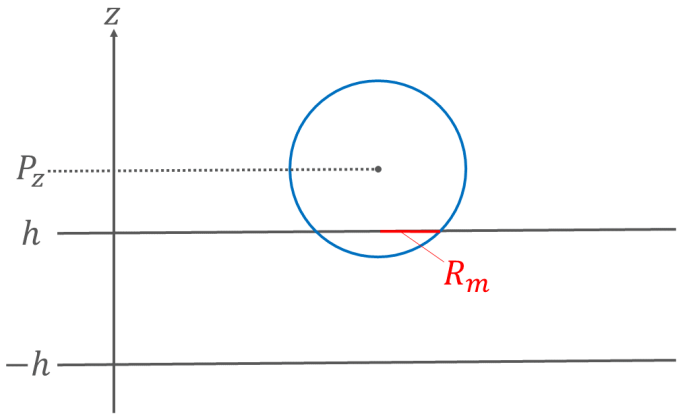

図 2.5　P_z が厚み空間外にある時

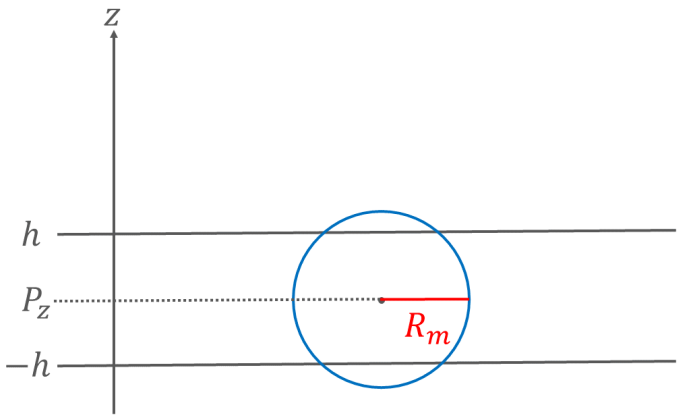

図 2.6　P_z が厚み空間内にあるとき

続いて、球の中心点 P を xy 平面に投影し点 P' を得る。これにより、中心点が点 P'、半径が R_m の円を第 2.4 節の干渉判定に代入することが出来る。代入した計算結果が球と扇形の干渉判定の結果となる。

2.6　二次元平面におけるカプセルとの干渉判定

本節では、始点 S と終点 E、半径 r_c からなるカプセルと扇形の干渉判定について述べる。

始めに、始点 S と終点 E を端点とする線分 (以降、カプセル線分) の方向ベクトル $\mathbf{V}$ を $\mathbf{V} = \mathbf{E} - \mathbf{S}$ とすると、線分上の点 P は媒介変数 t を用いて、式 (2.13) のように表せる。ここで t は式 (2.14) という値域を持つ。

$$\mathbf{P} = \mathbf{S} + t\mathbf{V} \ , \tag{2.13}$$

$$0 \leq t \leq 1 \ . \tag{2.14}$$

カプセルは、カプセル線分に沿って半径 r_c の円がスイープしてできる形状である [20]。このことから、中心点を P とし、半径が r_c の円 C が扇形と干渉する条件を考える。

まず、円 C が扇形を構成する内円、外円の少なくとも片方と干渉する条件は、式 (2.15) のようになる。

$$(r - r_c)^2 \leq {P_x}^2 + {P_y}^2 \leq (R + r_c)^2 \tag{2.15}$$

式 (2.15) を展開してえられる媒介変数 t の値域は、次の 2 つの二次方程式の解から導き出される。

$$P_x{}^2 + P_y{}^2 = (R + r_c)^2 \ , \tag{2.16}$$
$$P_x{}^2 + P_y{}^2 = (r - r_c)^2 \ . \tag{2.17}$$

ここで、$P_x = S_x + tV_x$、$P_y = S_y + tV_y$ より、式 (2.16)、式 (2.17) は t の二次式となる。また、式 (2.16) の解を a_R、$b_R(a_R \leq b_R)$ とし、式 (2.17) の解を a_r、$b_r(a_r \leq b_r)$ とすると、式 (2.15) から得られる t の値域は、式 (2.16) の判別式 D_R と式 (2.17) の判別式 D_r を用いて、次の 3 つの場合に分けられる。また、判別式 D_R と D_r から成る 3 つの場合分けは、図 2.7 に示す状況を表している。

- $D_R < 0$ のとき：値域なし
- $D_R \geq 0$ かつ $D_r \geq 0$ のとき：$a_R \leq t \leq a_r$、$b_r \leq t \leq b_R$
- $D_R \geq 0$ かつ $D_r < 0$ のとき：$a_R \leq t \leq b_R$

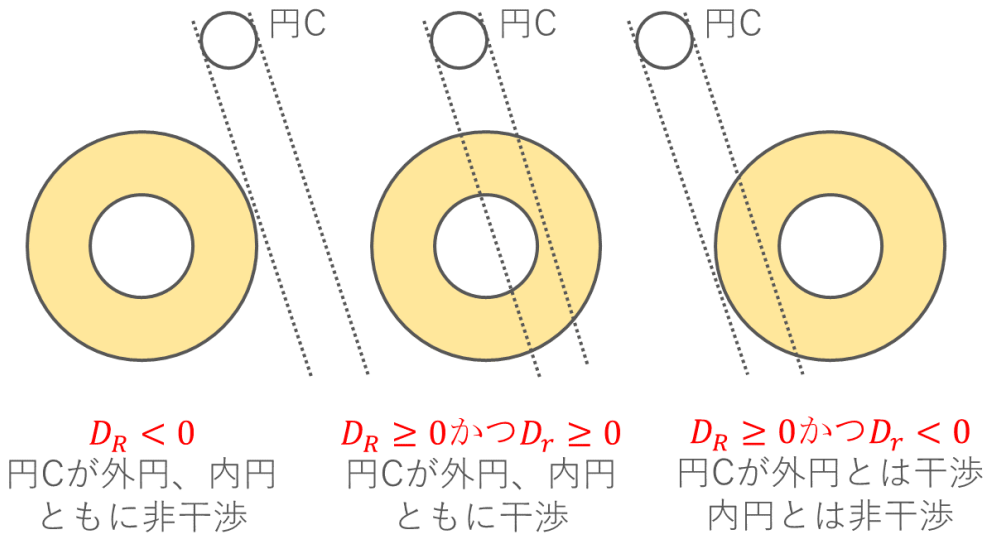

図 2.7　判別式からわかる 3 つの場合

続いて、点 P が扇形の範囲角度内に存在する t の範囲を求める。ここで、扇形を構成する 2 つ

の線分の方向ベクトル **G**、**H** は次のようになる。

- $\mathbf{G} = (\cos\theta, \sin\theta)$
- $\mathbf{H} = (\cos\theta, -\sin\theta)$

点 P の位置ベクトルと線分の方向ベクトル **G**、**H** の外積演算により算出されるベクトルをそれぞれ、$\mathbf{G}'$ と $\mathbf{H}'$ とすると、$\mathbf{G}'$ と $\mathbf{H}'$ の z 成分から、点 P が扇形の範囲角度内に存在するかどうかを判定できる。ここで、G'_z は式 (2.18)、H'_z は式 (2.19) のようになる。

$$\begin{aligned} G'_z &= P_x G_y - P_y G_x \\ &= S_x G_y - S_y G_x + (V_x G_y - V_y G_x)t \end{aligned} \tag{2.18}$$

$$\begin{aligned} H'_z &= P_x H_y - P_y H_x \\ &= S_x H_y - S_y H_x + (V_x H_y - V_y H_x)t \end{aligned} \tag{2.19}$$

点 P が扇形の範囲角度内に存在するかどうかの条件式は、扇形範囲角度の半角 θ の値から、式 (2.20) のように場合分けを行う。

$$\begin{cases} G'_z \geq 0 \text{ and } H'_z \leq 0 & \text{if } 0 \leq \theta < \frac{\pi}{2} \\ G'_z \geq 0 \text{ or } H'_z \leq 0 & \text{if } \frac{\pi}{2} \leq \theta \leq \pi \end{cases} \tag{2.20}$$

ただし、式 (2.20) において G'_z、H'_z から t の値域を求めるとき、式 (2.18) の $V_x G_y - V_y G_x$ と式 (2.19) の $V_x H_y - V_y H_x$ の正負によって不等式の符号を考慮する必要がある。

式 (2.14) と、式 (2.15)、式 (2.20) のすべてを満たす t の値域が存在するとき、扇形とカプセルは干渉している。

しかし、式 (2.14) と、式 (2.15)、式 (2.20) のすべてを満たす t の値域が存在しないとしても、扇形とカプセルが干渉している場合がある。図 2.8 にその一例を示す。そのため、式 (2.14) と、式 (2.15)、式 (2.20) のすべてを満たす t の値域が存在しないならば、第 2.9.5 項で述べる線分と線分の最短距離計算 [26][7] を用いて、扇形を構成する 2 つの線分と、カプセル線分の最小距離をを用いて計算する。こうして計算できた最小距離 d_G と d_H のどちらか、または両方が r_c 以下の時、扇形とカプセルは干渉している。

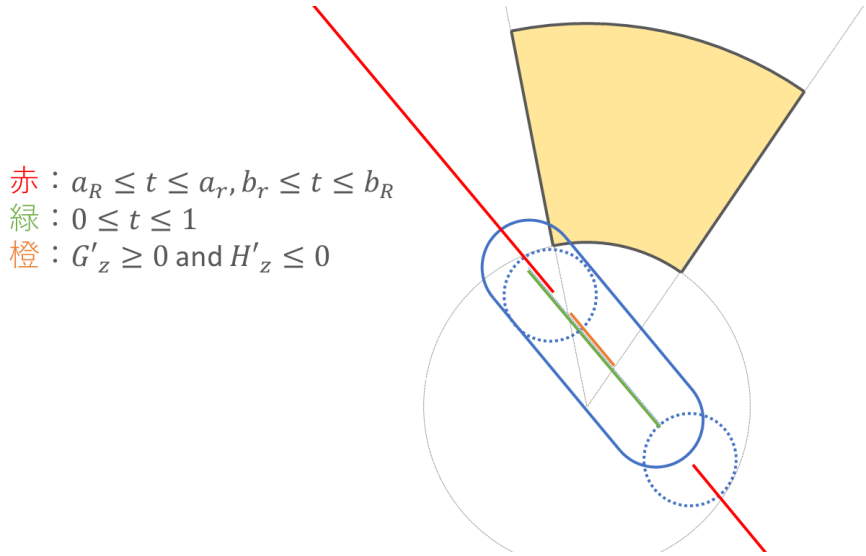

図 2.8　式 (2.14) と、式 (2.15)、式 (2.20) のすべてを満たす t の値域が存在しないが、扇形とカプセルが干渉している一例

2.7　三次元空間におけるカプセルとの干渉判定

本節では、始点 S と終点 E、半径 r_c からなるカプセル C と扇形の干渉判定について述べる。本節の干渉判定は近似解であるため、本来は干渉していないが干渉していると判定する様な誤った判定結果になる場合があるが、それについては後述する。

始めに、カプセル C が扇形の厚み空間と干渉しているかどうかを確認する。ここで $S_z > E_z$ であるとすると、式 (2.21) を満たすときカプセル C と厚み空間は干渉していないことがわかる。カプセル C と扇形の厚み空間が干渉していないなら、カプセル C と扇形が干渉している可能性はないため、以降の干渉判定処理を行う必要は無い。

$$S_z + r_c < -h \text{ or } E_z - r_c > h \tag{2.21}$$

三次元空間におけるカプセルと扇形の干渉判定は、第 2.5 節で述べた扇形と球の干渉判定同様、二次元平面の干渉判定に置き換えることを目指す。まず、カプセルと扇形の厚み空間からなる二次元形状（以降、疑似カプセル）について考える。カプセルは球が平行移動したときの移動軌跡であるため、第 2.5 節で述べた厚み空間内の最大半径の円から疑似カプセルの形がわかる。図 2.9 に疑似カプセルの模式図を示す。カプセルと扇形の厚み空間の位置関係が図 2.9 の左側のように

なっているとき、疑似カプセルの形状は図右側の黒色の形状になる。ただし、カプセルの始点と終点がどちらも厚み空間内に存在するとき、疑似カプセルの形は二次元平面でのカプセル形状と完全に同一となる。また、カプセルの始点と終点を端点とする線分が z 軸と平行であるとき、疑似カプセルの形状は円と完全に同一となる。

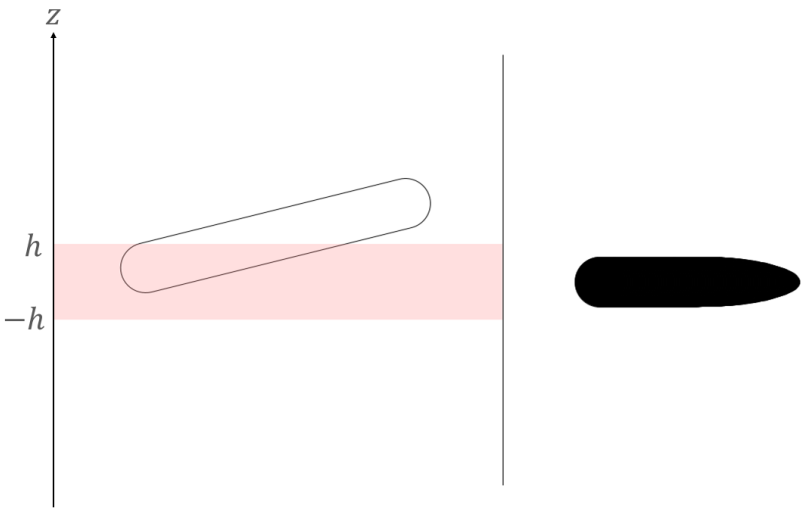

図 2.9　扇形の厚み空間とカプセルの位置関係からわかる疑似カプセルの形状

扇形の厚み空間とカプセル C からなる疑似カプセル α を二次元形状のカプセル C' で囲うことで、三次元空間におけるカプセル C との干渉判定結果を近似する。そのため、カプセル C' に囲われた空間の中で、疑似カプセル α が存在していない箇所と扇形が干渉している場合、本来は扇形とカプセル C は干渉していないという判定結果になるはずが、干渉しているという誤った判定結果になってしまう。図 2.10 に疑似カプセルを二次元平面におけるカプセルで囲んだ様子を示す。

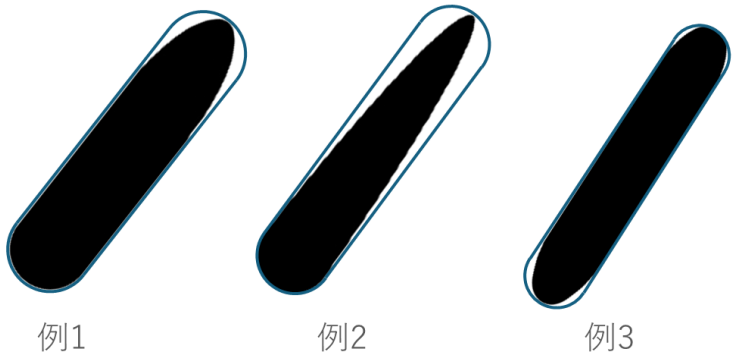

図 2.10　疑似カプセルを二次元平面におけるカプセルで囲んだ様子

ここからは、扇形の厚み空間とカプセル C からなる疑似カプセル α をカプセル C' で囲う方法を述べる。カプセル C' が始点 S' と終点 E'、半径 $r_c{}'$ からなるとしたとき、カプセル C' が疑似カプセル α に外接するように点 S'、E' の位置、半径 $r_c{}'$ の大きさを計算する。

始めに、半径 $r_c{}'$ の大きさを計算する。まず、点 S を中心点とする球 a と点 E を中心点とする球 b の厚み空間内での最大半径 r_s と r_e を第 2.5 節で述べた式 (2.12) を用いて計算する。ただし、球 a と球 b の半径はどちらも r_c である。このとき、$r_c{}'$ は式 (2.22) から求められる。

$$r_c{}' = \begin{cases} r_c & \text{if } S_z \geq -h \text{ and } E_z \leq h \\ \max(r_s, r_e) & \text{if } S_z < -h \text{ or } E_z > h \end{cases} \tag{2.22}$$

次に、点 S'、点 E' の位置を計算する。始めに、点 S と点 E を端点とする線分 m の方向ベクトル $\mathbf{V}$ を $\mathbf{V} = \mathbf{E} - \mathbf{S}$ とおく。また、$\mathbf{V}$ を xy 平面に射影したベクトルを $\mathbf{V}'$ とおく。ここで、線分 m の属する直線 M 上の点を中心点とし、半径が r_c である球群からなる疑似カプセル β について考える。まず、直線 M 上の点で z 成分が h である点 Q を式 (2.23) から計算する。

$$\mathbf{Q} = \mathbf{S} + \frac{h - S_z}{V_z}\mathbf{V} \tag{2.23}$$

続いて、点 Q を xy 平面に投影した点 Q' から、疑似カプセル β 上の点で $-\mathbf{V}'$ 方向に最も遠い点までの距離 d を計算する。点 Q' から疑似カプセル β 上の点までの $-\mathbf{V}'$ 方向の距離は関数 $L(s)$

から求められる。式 (2.24) に関数 $L(s)$ を示す。ただし、変数 s は媒介変数であり、式 (2.25) を満たす。また、変数 w はベクトル $\mathbf{V}$ の z 成分と半径 r_c の比から式 (2.26) より求められる。図 2.11 に変数 w の模式図を示す。

$$\begin{aligned} L(s) &= sw + \sqrt{{r_c}^2 - (sr_c)^2} \\ &= sw + r_c\sqrt{1 - s^2} \end{aligned} \tag{2.24}$$

$$0 \leq s \leq 1 \, , \tag{2.25}$$

$$w = \frac{r_c}{-V_z}\sqrt{{V_x}^2 + {V_y}^2} \, . \tag{2.26}$$

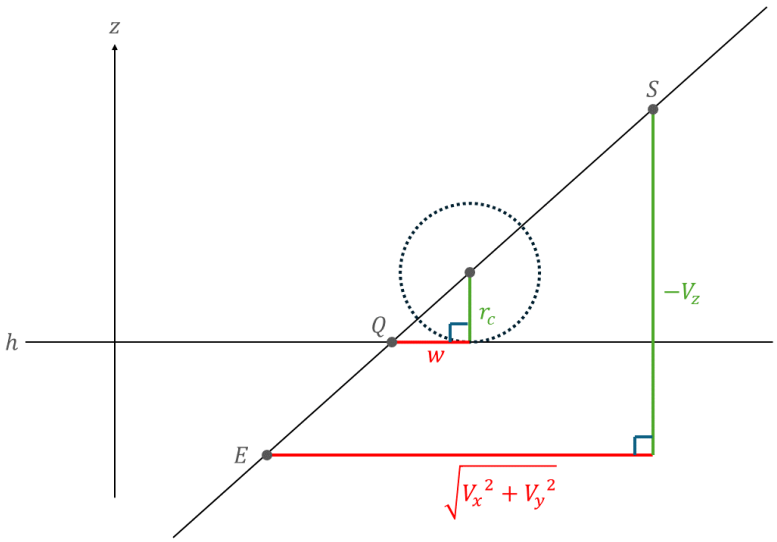

図 2.11　変数 w の模式図

ここで、$L(s) = d$ となるのは、関数 $L(s)$ の導関数 $L(s)'$ の値が 0 となるときである。式 (2.27) に導関数 $L(s)'$ を示す。$L(s)' = 0$ のとき、式 (2.27) は二次方程式となるが、媒介変数 s は式 (2.25) を満たすため、s の値は式 (2.28) により求められる。

$$L(s)' = w - \frac{rs}{\sqrt{1 - s^2}} \tag{2.27}$$

$$s = \frac{w}{\sqrt{w^2 + {r_c}^2}} \tag{2.28}$$

求めた媒介変数 s から、直線 M 上の点 G、H の z 座標を式 (2.29) より算出する。ここで、点 G、H を中心点とする球 g、h は疑似カプセル β を構成する球群の中で、それぞれ $-\mathbf{V}'$ 方向、$\mathbf{V}$ 方向に最も遠い点を持つ球である。

$$\begin{aligned} G_z &= Q_z + sr_c \ , \\ H_z &= -G_z \ . \end{aligned} \tag{2.29}$$

求めた G_z、H_z を用いて、xy 平面に投影した際に点 S'、E' になる可能性がある点 S"、E" を式 (2.30) より計算する。

$$\begin{aligned} \mathbf{S}'' &= \mathbf{S} + \frac{\min(S_z, G_z) - S_z}{V_z}\mathbf{V} \ , \\ \mathbf{E}'' &= \mathbf{S} + \frac{\max(E_z, H_z) - S_z}{V_z}\mathbf{V} \ . \end{aligned} \tag{2.30}$$

この時、点 S"、E" を xy 平面に投影した点を点 S'、E' と設定すると、カプセル C' が疑似カプセル α に外接していない場合が存在する。それは、点 S"、E" を中心点とし、半径 r_c の球 a' と球 b' の扇形の厚み空間内での最大半径 ${r_a}'$、${r_b}'$ がカプセル C' の半径 ${r_c}'$ と等しくない時である。そこで、カプセル C' が疑似カプセル α に外接するように、先ほど計算した点 S"、E" を $\mathbf{V}'$ 方向、$-\mathbf{V}'$ 方向に ${r_a}'$、${r_b}'$ と ${r_c}'$ の差分平行移動させてから、xy 平面に投影する。ここで、点 S"、E" を平行移動させた点をそれぞれ点 J、K とすると、点 J、K の位置ベクトルは式 (2.31) により求められる。

$$\begin{aligned} \mathbf{J} &= \mathbf{S}'' + \frac{{r_c}' - {r_a}'}{\sqrt{{V_x}^2 + {V_y}^2}}\mathbf{V} \\ \mathbf{K} &= \mathbf{E}'' - \frac{{r_c}' - {r_b}'}{\sqrt{{V_x}^2 + {V_y}^2}}\mathbf{V} \end{aligned} \tag{2.31}$$

最後に、点 J、K をそれぞれ xy 平面に投影し、点 S'、E' を得ることで、三次元空間におけるカプセルと扇形の干渉判定を二次元平面の問題に置き換え、干渉判定結果を近似することが出来る。

2.8　干渉判定の様子

本節では、第 2.2 節から第 2.6 節までに述べた扇形と各プリミティブ形状との干渉判定をプログラム上で実装した様子を図で示す。扇形と対象のプリミティブ形状が干渉している場合には、対象のプリミティブ形状の色が赤色になり、非干渉の場合には青色になるようにしている。プログラムの実装には Fine Kernel Tool Kit System[27] を用いて作成した。バージョンは FK Core22 4.2.11.0 である。なお、各図における色のついた線分は次のような意味を持つ。

- 黒線の格子:xy 平面を表す
- 赤線:x 軸 (1, 0, 0) を表す
- 緑線:y 軸 (0, 1, 0) を表す
- 青線:z 軸 (0, 0, 1) を表す

図 2.12 と図 2.13 に第 2.2 節で述べた扇形と点の内外判定の様子を示す。

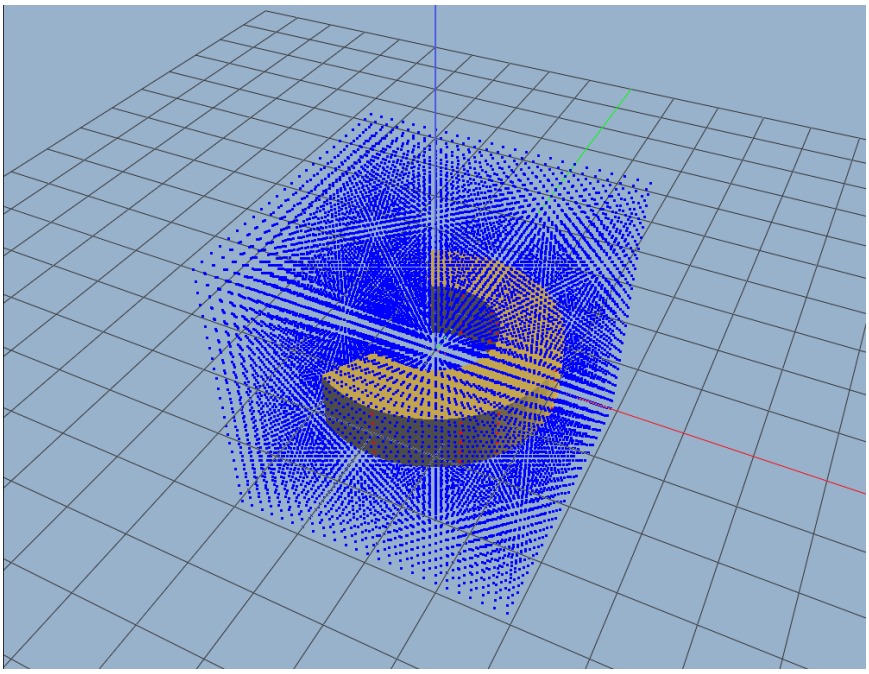

図 2.12　点群内に扇形が存在する様子

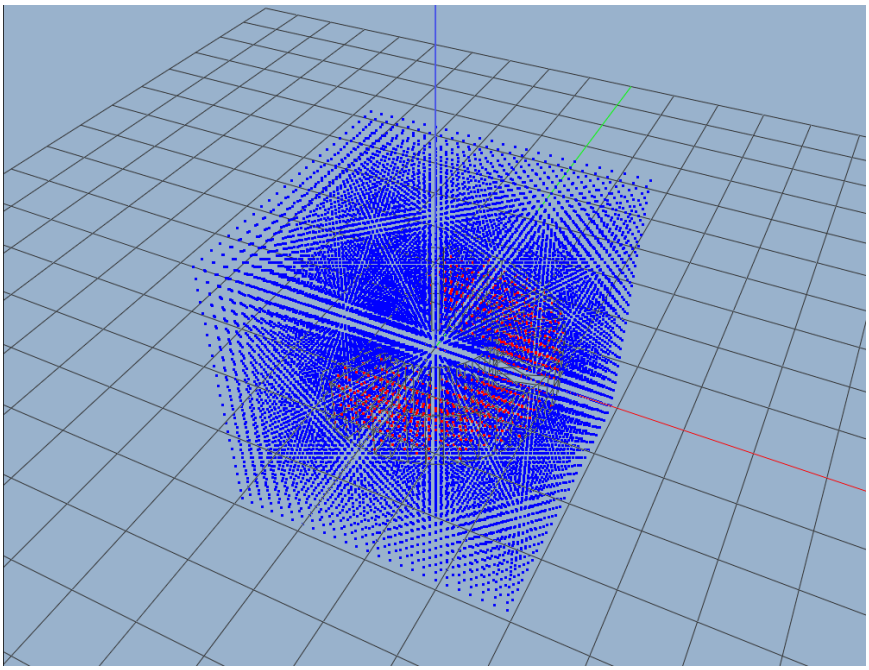

図 2.13　扇形内の点群の色が変化している様子

図 2.14 と図 2.15 には第 2.3 節で述べた、扇形と二次元平面における長方形との干渉判定の様子を示す。

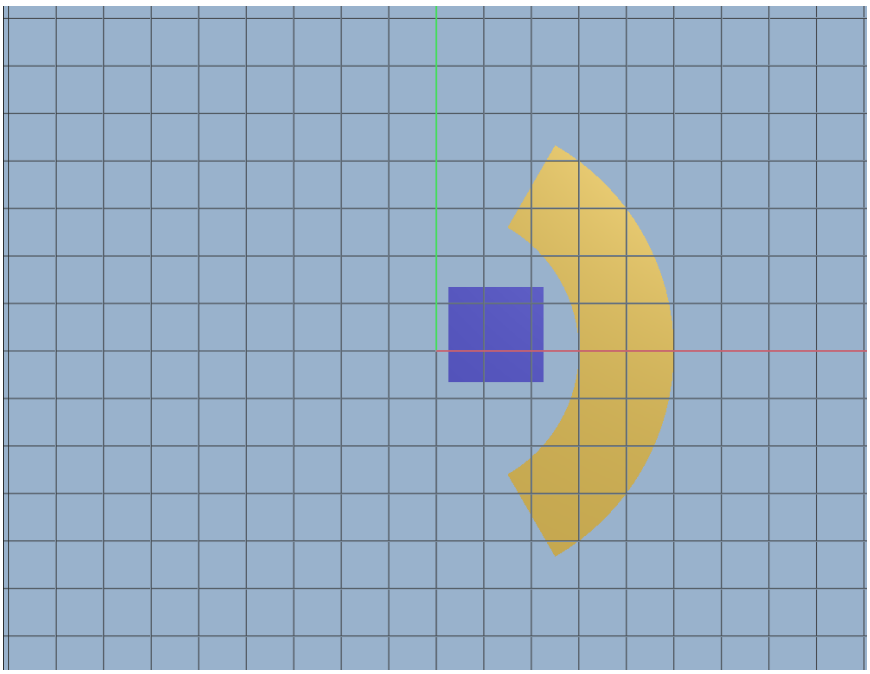

図 2.14　扇形と長方形が非干渉の様子

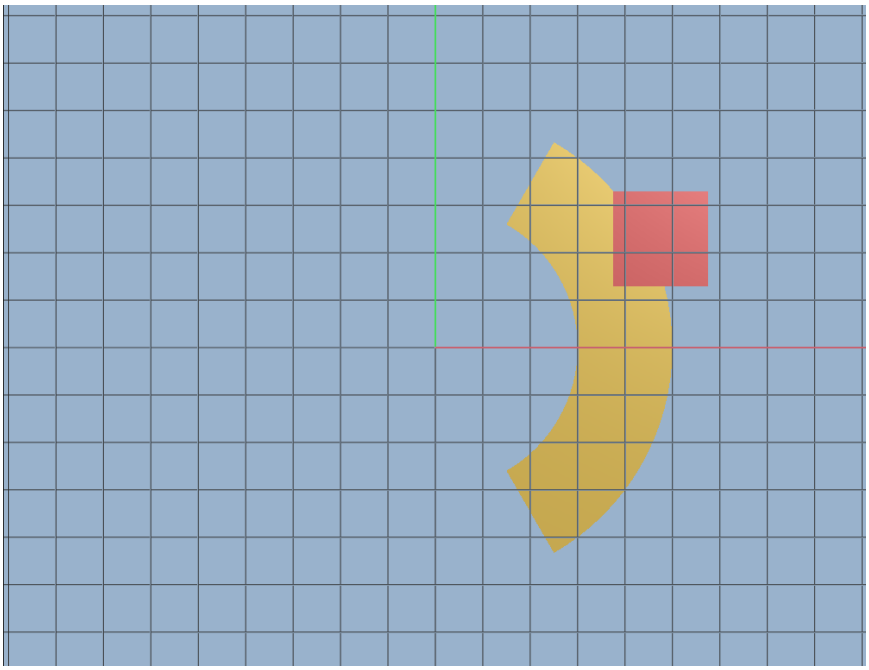

図 2.15　扇形と長方形が干渉している様子

図 2.16 と図 2.17 には、第 2.5 節で述べた、扇形と三次元空間における球との干渉判定の様子を示す。

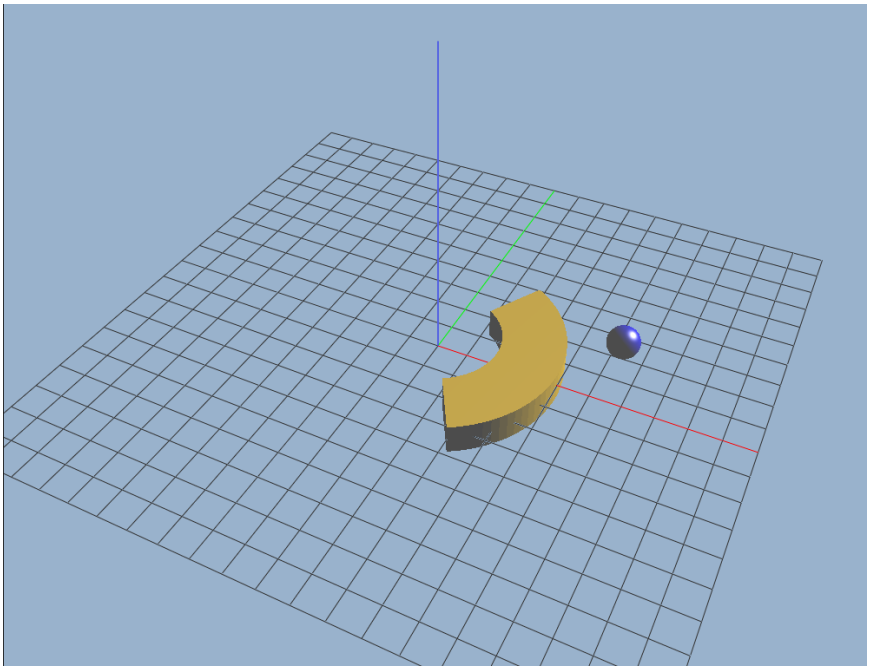

図 2.16　扇形と球が非干渉の様子

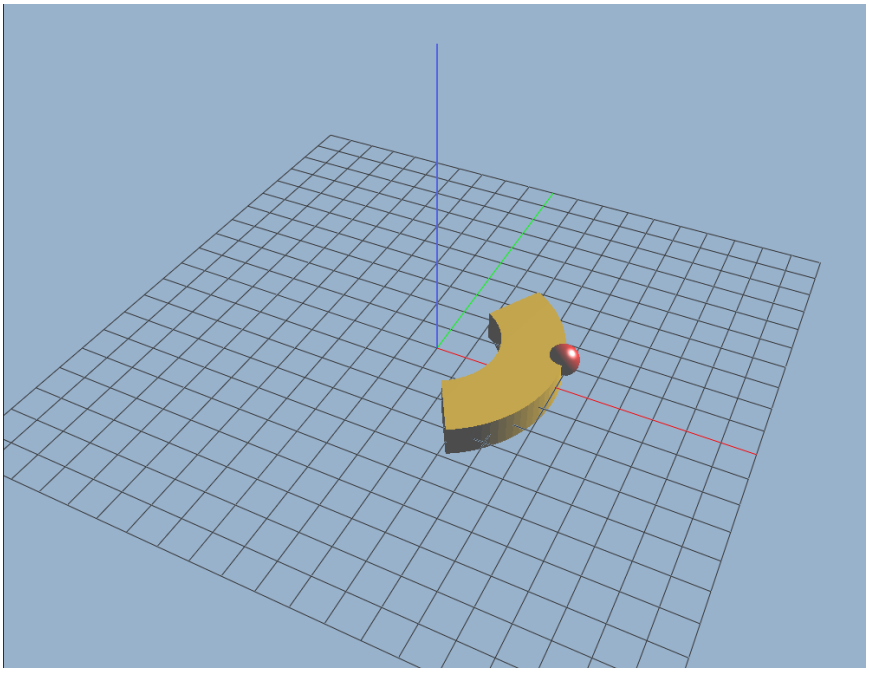

図 2.17　扇形と球が干渉している様子

図 2.18 と図 2.19 に、第 2.6 節で述べた、扇形と二次元平面におけるカプセルとの干渉判定の様子を示す。

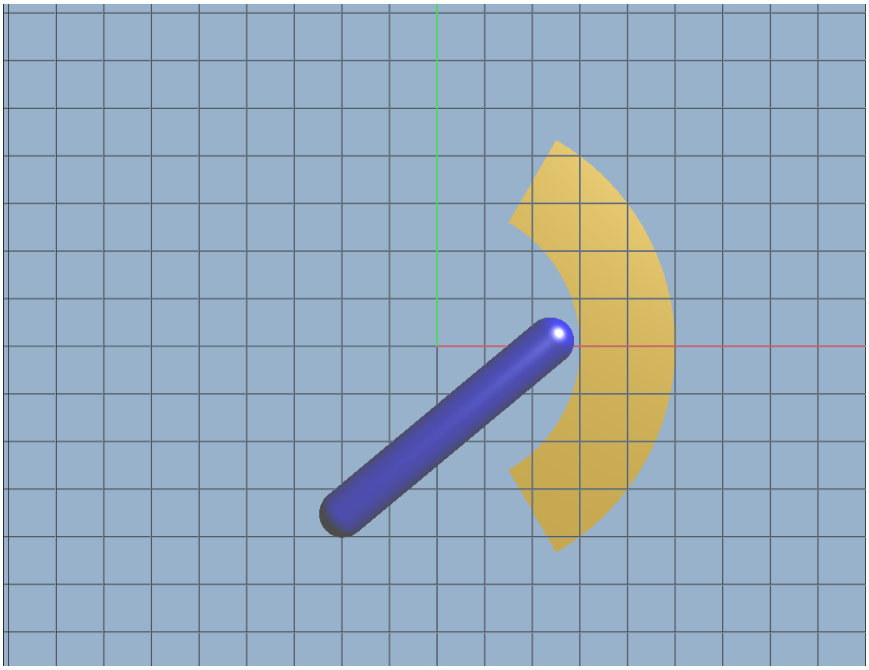

図 2.18　扇形と二次元平面におけるカプセルが非干渉の様子

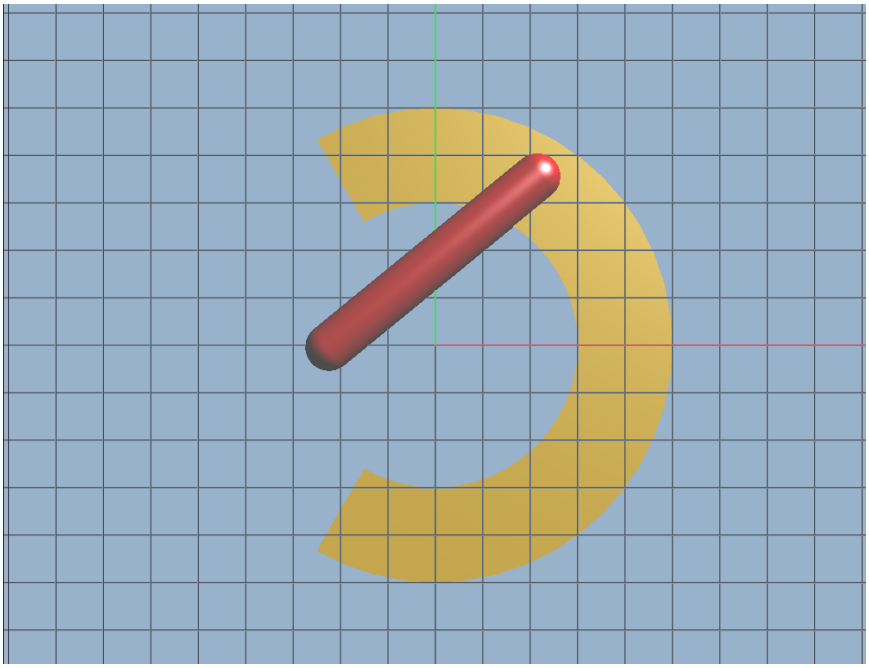

図 2.19　扇形と二次元平面におけるカプセルが干渉している様子

最後に、図 2.20 と図 2.21 に、第 2.7 節で述べた、扇形と三次元空間におけるカプセルとの干渉判定の様子を示す。また、図 2.22 に扇形と三次元空間におけるカプセルとの干渉判定結果が誤った結果になる場合の例を示す。

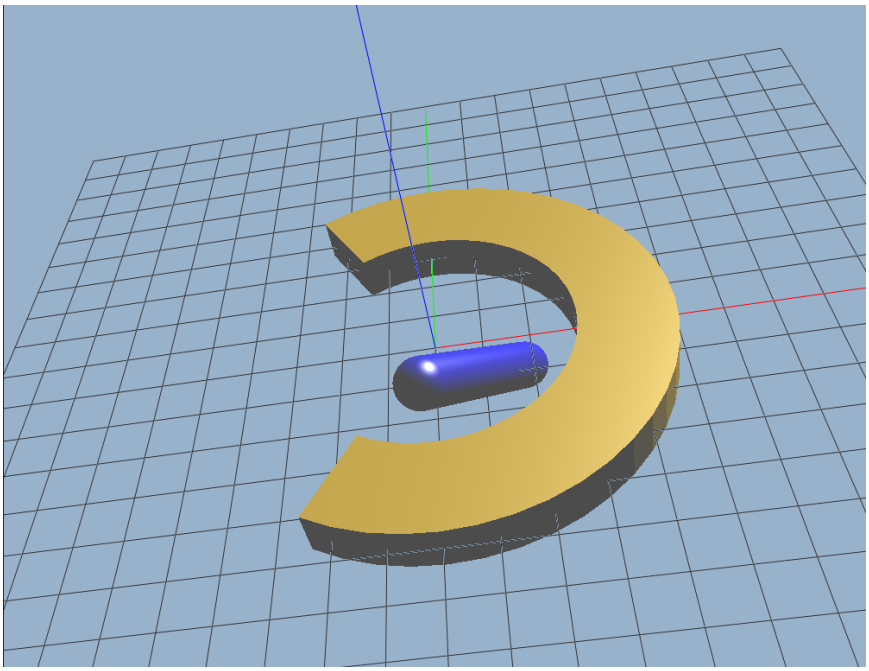

図 2.20　扇形と三次元空間におけるカプセルが非干渉の様子

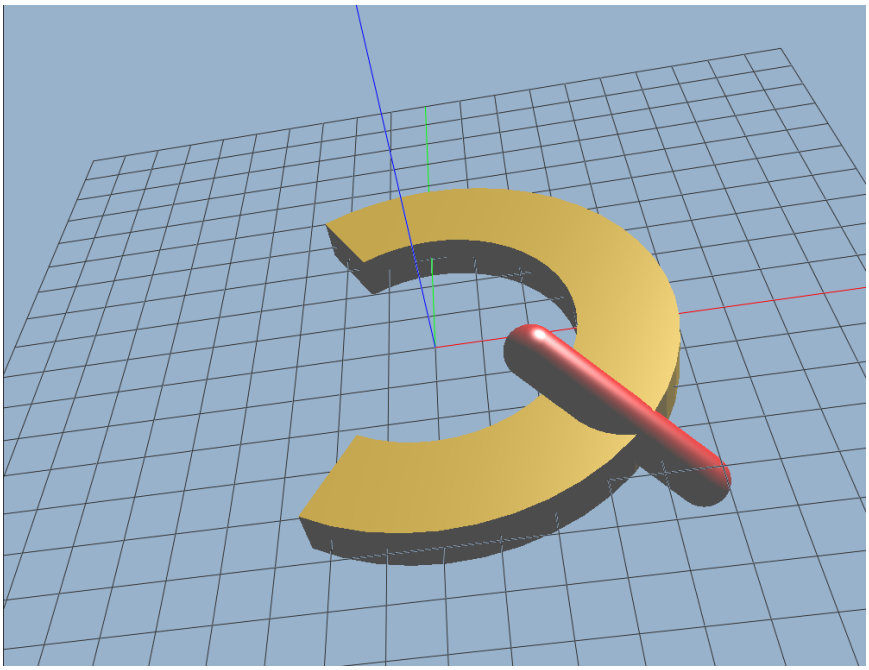

図 2.21　扇形と三次元空間におけるカプセルが干渉している様子

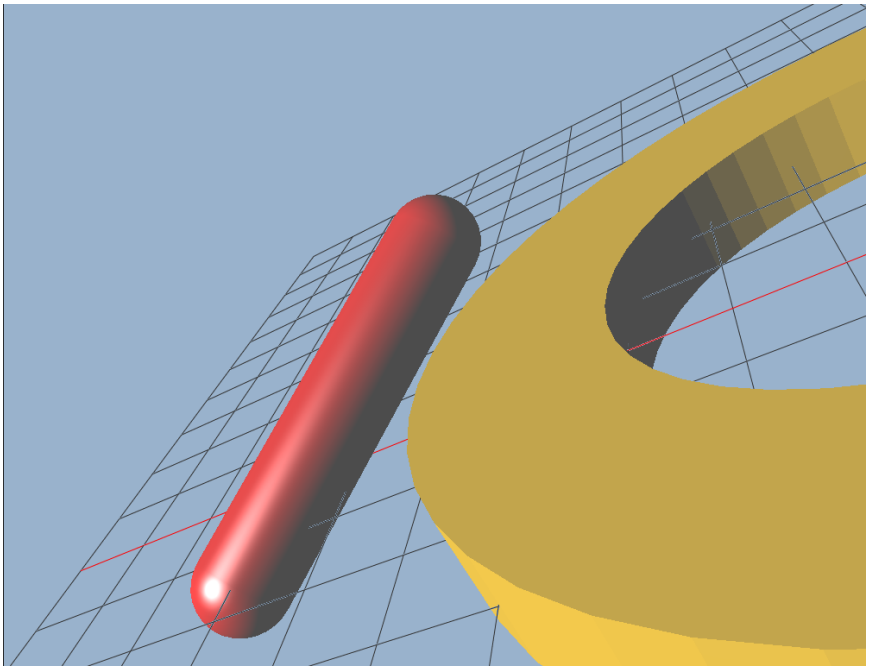

図 2.22　扇形と三次元空間におけるカプセルの干渉判定結果が誤った判定になる場合

2.9 提案手法で使用した基本処理

本節では、本稿で提案する干渉判定手法の中で使用した基本幾何計算についてまとめる。

2.9.1 線分同士の交差点算出

本項では、線分同士の交差点算出について述べる。

点 E、F を端点とする線分を線分 L とし、同様に点 G、H を端点とする線分を線分 M とする。線分 L と線分 M が交差するとき、交差点を点 P とすると、媒介変数 s、t を用いて点 P の位置ベクトルは式 (2.32) のように表せる。

$$\mathbf{P} = \mathbf{E} + s(\mathbf{F} - \mathbf{E}) = \mathbf{G} + t(\mathbf{H} - \mathbf{G}) \tag{2.32}$$

式 (2.32) を x, y 成分についてそれぞれ整理すると、式 (2.33) のようになる。

$$\begin{cases} E_x + s(F_x - E_x) = G_x + t(H_x - G_x) \\ E_y + s(F_y - E_y) = G_y + t(H_y - G_y) \end{cases} \tag{2.33}$$

式 (2.33) を s, t について解くと、式 (2.34) のようになる。

$$\begin{aligned} s &= \frac{(G_x - E_x)(H_y - G_y) - (G_y - E_y)(H_x - G_x)}{(F_x - E_x)(H_y - G_y) - (G_y - F_y)(H_x - G_x)} \\ t &= \frac{(E_x - G_x)(F_y - E_y) - (E_y - G_y)(F_x - E_x)}{(H_x - G_x)(F_y - E_y) - (H_y - G_y)(F_x - E_x)} \end{aligned} \tag{2.34}$$

線分 L、M が平行でないとき、式 (2.34) から s, t の値が決まるため、s, t いずれかの値を式 (2.33) に代入することで、点 P の座標を求めることができる。

また、s, t が式 (2.35) の条件を満たさないとき、線分 L と線分 M は交差していない。

$$0 \leq s \leq 1, 0 \leq t \leq 1 \tag{2.35}$$

2.9.2 線分と円の交差点算出

本項では、円と線分の交差点算出について述べる。

点 E、F を端点とする線分を線分 L とし、点 C を中心とする半径 r の円を円 C とする。線分 L と円 C が交差するとき、その交点を点 P、Q とする。

媒介変数 s を使うと、点 E、F を通る直線上の点 P の位置ベクトルは式 (2.36) のようにあらわせる。

$$\mathbf{P} = \mathbf{E} + s(\mathbf{F} - \mathbf{E}) \tag{2.36}$$

線分 L と円 C が交差しているとき、円 C の方程式に点 P の x, y 成分を代入することで s の値が決定し、交点を求めることができる。点 P の x, y 成分を円 C の方程式に代入すると式 (2.37) のようになる。

$$(P_x - C_x)^2 + (P_y - C_y)^2 = r^2 \tag{2.37}$$

式の簡略化のため、ベクトル $\mathbf{V}$、$\mathbf{W}$ を式 (2.38) のように定義する。

$$\begin{cases} \mathbf{V} &= \mathbf{F} - \mathbf{E} \\ \mathbf{W} &= \mathbf{E} - \mathbf{C} \end{cases} \tag{2.38}$$

式 (2.37) を s についてまとめると、式 (2.39) のようになる。

$$(V_x^2 + V_y^2)s^2 + 2(V_x W_x + V_y W_y)s + (W_x^2 + W_y^2 - r^2) = 0 \tag{2.39}$$

二次方程式の解の公式より、式 (2.39) から s の値を求めることができる。求めた s の値を $0 \leq s \leq 1$ に制限し、式 (2.36) に代入することにより、点 PQ の値を求めることができる。また、二次方程式の解の公式の判別式から交差点の数を判別することができる。判別式を D とすると、交差点の数と点 P、Q の関係は次のようになる。

- $D > 0$ の時、交差点の数は 2 つで、$\mathbf{P} \neq \mathbf{Q}$
- $D = 0$ の時、交差点の数は 1 つで、$\mathbf{P} = \mathbf{Q}$
- $D < 0$ の時、交差点は存在しない

2.9.3 長方形と点の内外判定

本項では、ある点が長方形の内部に存在するかどうかを判定する方法について述べる。

ここで、ある点を P、長方形の各頂点を点 E、F、G、H とする。始めに、長方形の各線分方向のベクトルと、各線分の端点から点 P へのベクトルの外積演算を行う。外積演算を行った結果のベクトルをそれぞれ $\mathbf{C}_E$、$\mathbf{C}_F$、$\mathbf{C}_G$、$\mathbf{C}_H$ とすると、それぞれのベクトルは式 (2.40) により算出される。ただし、$\times$ は外積演算記号を表す。

$$\begin{cases} \mathbf{C}_E &= (\mathbf{F}-\mathbf{E}) \times (\mathbf{P}-\mathbf{E}) \\ \mathbf{C}_F &= (\mathbf{G}-\mathbf{F}) \times (\mathbf{P}-\mathbf{F}) \\ \mathbf{C}_G &= (\mathbf{H}-\mathbf{G}) \times (\mathbf{P}-\mathbf{G}) \\ \mathbf{C}_H &= (\mathbf{E}-\mathbf{H}) \times (\mathbf{P}-\mathbf{H}) \end{cases} \tag{2.40}$$

ベクトル $\mathbf{C}_E$、$\mathbf{C}_F$、$\mathbf{C}_G$、$\mathbf{C}_H$ 成分すべてが 0 以上であるとき、または、すべてが 0 以下であるとき、ある点 P は長方形の内部に存在する。

2.9.4 点と線分の最短距離計算

本項では、ある点とある線分の最短距離計算について述べる。

ここで、ある点を点 Q、ある線分を始点 S と終点 E からなる線分 N' とする。始めに、点 Q と線分 N' が属する直線 N との最短距離を計算する。線分 N の方向ベクトルを $\mathbf{V} = \mathbf{E} - \mathbf{S}$ とすると、直線 N 上の点 N の位置ベクトルは媒介変数 t を用いて、式 (2.41) のように計算される。

$$\mathbf{N} = \mathbf{S} + t\mathbf{V} \tag{2.41}$$

点 Q から直線に最も近い点は、点 Q から直線に向けて伸ばした垂線との交点となる。そのため、点 Q と直線の最短距離を表すベクトルと直線の方向ベクトルの内積は必ず 0 となり、式 (2.42) が成り立つ。

$$(\mathbf{N}-\mathbf{Q}) \cdot \mathbf{V} = 0 \tag{2.42}$$

式 (2.42) を t について整理すると、式 (2.43) が成り立つ。

$$t = \frac{(\mathbf{Q} - \mathbf{S}) \cdot \mathbf{V}}{|\mathbf{V}|^2} \tag{2.43}$$

続いて、点 N が線分 N' 上の点になるようにするため、式 (2.43) により計算した媒介変数 t の値を 0 から 1 に制限する。t の値を 0 から 1 に制限した後の値を t' とすると、t' は式 (2.44) で求められる。

$$t' = \max(0, \min(t, 1)) \tag{2.44}$$

よって、線分 N' 上の点 N' は式 (2.45) により求められるため、点 Q と線分 N' の最短距離 d は式 (2.46) のようになる。

$$\mathbf{N}' = \mathbf{S} + t'\mathbf{V} \ , \tag{2.45}$$

$$d = |\mathbf{Q} - \mathbf{N}'| \ . \tag{2.46}$$

2.9.5 線分と線分の最短距離計算

本項では、線分と線分の最短距離計算について述べる。

まず、2 つの線分をそれぞれ、線分 L'、M' とする。次に、線分 L' の端点を点 C、D とし線分 M' の端点を点 E、F とする。また、それぞれの方向ベクトル $\mathbf{V}_L$、$\mathbf{V}_M$ は、(2.47) のようになる。

$$\begin{cases} \mathbf{V}_L &= \mathbf{D} - \mathbf{C} \\ \mathbf{V}_M &= \mathbf{F} - \mathbf{E} \end{cases} \tag{2.47}$$

始めに、それぞれの線分が属する直線 L と直線 M の最短距離を計算する。媒介変数 t_L と t_M を用いて、それぞれの直線上の点 L と点 M は式 (2.48) のようになる。

$$\begin{cases} \mathbf{L} &= \mathbf{C} + t_L\mathbf{V}_L \\ \mathbf{M} &= \mathbf{E} + t_M\mathbf{V}_M \end{cases} \tag{2.48}$$

直線 L と直線 M の最短距離は、点 L と点 M を通る直線が、どちらの直線とも垂直に交わっている場合の、$|\mathbf{M} - \mathbf{L}|$ の値となる。よって、式 (2.49) と (2.50) が成り立つ。

$$\mathbf{V}_L \cdot (\mathbf{M} - \mathbf{L}) = 0 \,, \tag{2.49}$$
$$\mathbf{V}_M \cdot (\mathbf{M} - \mathbf{L}) = 0 \,. \tag{2.50}$$

式 (2.49) から式 (2.51) が得られるので、式 (2.50) を同様に t_M について解いた式 (2.52) に式 (2.51) を代入して得らる式 (2.53) から、t_M を算出する。また、こうして算出した t_M を式 (2.51) に代入することで t_L を算出できるため、直線 L と直線 M の最短距離を求めることが出来る。

$$t_L = \frac{\mathbf{V}_L \cdot (\mathbf{E} - \mathbf{C} + t_M \mathbf{V}_M)}{|\mathbf{V}_L|^2} \,, \tag{2.51}$$
$$\mathbf{V}_M \cdot (\mathbf{E} - \mathbf{C}) + t_M |\mathbf{V}_M|^2 - t_L \mathbf{V}_L \cdot \mathbf{V}_M = 0 \,, \tag{2.52}$$
$$t_M = \frac{\mathbf{V}_M \cdot (\mathbf{E} - \mathbf{C})|\mathbf{V}_L|^2 - \mathbf{V}_L \cdot (\mathbf{E} - \mathbf{C})(\mathbf{V}_L \cdot \mathbf{V}_M)}{(\mathbf{V}_L \cdot \mathbf{V}_M)^2 - |\mathbf{V}_L|^2 |\mathbf{V}_M|^2} \,. \tag{2.53}$$

続いて、線分 L' と線分 M' の最短距離計算を行う。先ほど算出した t_L と t_M の片方でも 0 から 1 の間の値でない場合、点 L、M の片方、もしくはどちらも線分上に存在しない点であることがわかる。そこで、t_L と t_M のどちらの値も 0 から 1 に収まるように、処理を施していく。

まず、t_L を 0 から 1 の値になるように、補正をかける。この補正には、第 2.9.4 項で述べる式 (2.44) を利用する。t_L 補正後の値から線分 L' 上の点 L' を算出し、点 L' と線分 M を第 2.9.4 項の処理に代入し、この時の媒介変数の値 $t_M{}'$ を算出する。もし、$t_M{}'$ の値が 0 から 1 の間となれば、t_L と $t_M{}'$ から算出される点 L' と点 M' はどちらも線分上の点であることがわかるため、この時点で最短距離の算出が可能となる。

もし、$t_M{}'$ の値が 0 から 1 で無いならば、先ほど t_L に施した処理同様、$t_M{}'$ の値を 0 から 1 に補正し、補正後の値から算出される線分 M' 上の点 M' と線分 L' を第 2.9.4 項の処理に代入し、媒介変数の値 $t_L{}'$ を算出する。$t_L{}'$ の値が 0 から 1 の間であるならば、この時点で最短距離の算出が可能となる。

もし、$t_L{}'$ の値が 0 から 1 の間出ないならば、線分 L と線分 M の最短距離は、互いの端点同士の距離であることが判明する。そこで、$t_{L'}$ に対しても、値が 0 から 1 の間となるように補正をかけ、$t_L{}'$ と $t_M{}'$ から算出できる線分 L' と線分 M' の端点同士の距離を最短距離として算出する。

第 3 章

検証

本章では、既存手法である S-CCD 手法と、扇形を用いた運動軌跡の補間の 2 つの衝突判定について衝突判定精度の比較を行う。また、本稿で提案する扇形とプリミティブ形状の干渉判定の実行速度を調査し、境界ボリューム同士の干渉判定手法、S-CCD 手法と実行速度を比較する。

検証プログラムは、検証プログラムは第 2.8 節と同様の環境で作成した。

3.1 衝突判定精度の検証

本研究では、直方体が 2.1 節で述べた回転運動を行ったとき、直方体の回転運動軌跡と S-CCD 手法を適応した場合の干渉判定範囲、扇形による運動軌跡の補間を適応した場合の干渉判定範囲がどれだけ一致しているかから衝突判定精度を算出する。本検証を行うにあたり、直方体の回転運動の諸条件は次のように設定した。

- 以下で使用する疑似乱数は、C#言語の機能を用いた。
- 回転運動は原点を回転中心点とし、疑似乱数により決定される回転軸に沿って回転を行う。この時、疑似乱数により回転角度は 0 以上 2π 以下の値をとる。
- 直方体の中心点と回転中心点の距離は、疑似乱数により 12.5 以上 25 以下の値をとる。
- 直方体の横幅、高さ、奥行きは、疑似乱数により 0 以上 25 以下の値をとる。

続いて、衝突判定精度の算出方法について述べる。始めに、原点を中心に格子状に 100 万個の点を配置する。ここで、隣り合った点同士の距離が 1 になるように配置し、各点の座標の x、y、z 成分は-49.5 から 49.5 の値をとる。続いて、上述した回転運動から直方体の回転運動軌跡、S-CCD による干渉判定範囲、扇形の干渉判定範囲を計算する。直方体の回転運動軌跡は、回転運動を 100 回に分割して行うことで近似解を算出する。以降、直方体の回転運動軌跡の近似解を詳細 DCCD(Discrete CCD) 範囲、S-CCD による干渉判定範囲を S-CCD 範囲、扇形の干渉判定範囲を扇形範囲と呼称する。ここで、格子状に配置した各点を次の A 群から G 群の 7 つに分類し、各分類の個数を集計した。

A ：詳細 DCCD 範囲のみが内包している点

B ：S-CCD 範囲のみが内包している点

C ：扇形範囲のみが内包している点

D ：詳細 DCCD 範囲と S-CCD 範囲の両方が内包している点

E ：詳細 DCCD 範囲と扇形範囲の両方が内包している点

F ：S-CCD 範囲と扇形範囲の両方が内包している点

G ：詳細 DCCD 範囲、S-CCD 範囲、扇形範囲のすべてが内包している点

格子状の点を A 群から G 群のいずれかに分類し可視化した画像を図 3.1 から図 3.7 に示す。

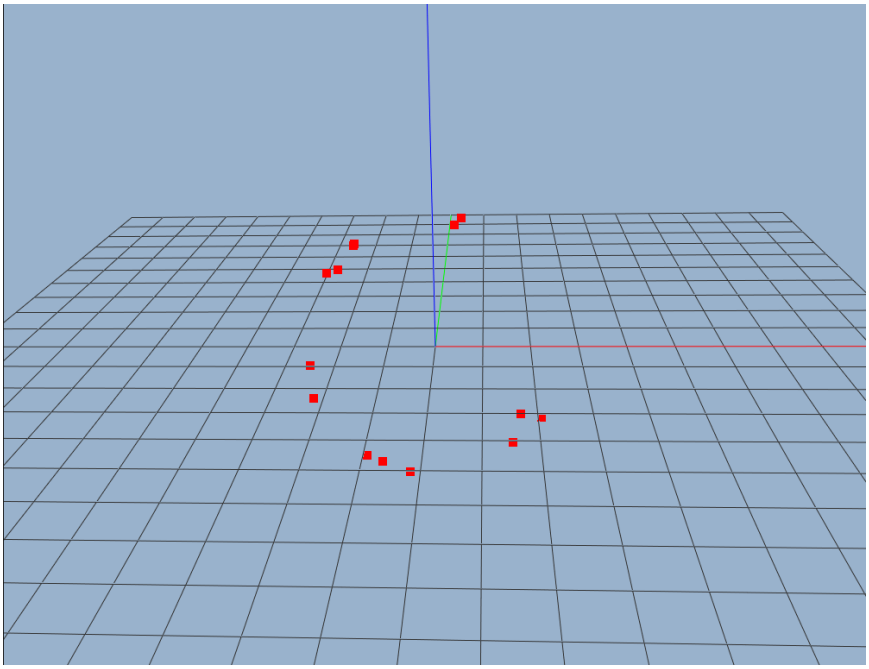

図 3.1　A 群に分類された点群

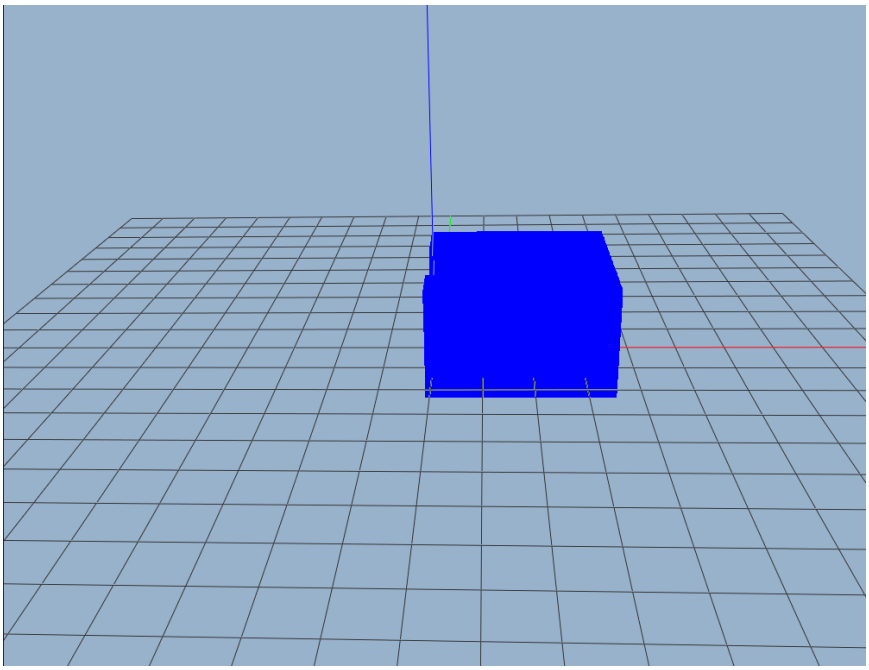

図 3.2　B 群に分類された点群

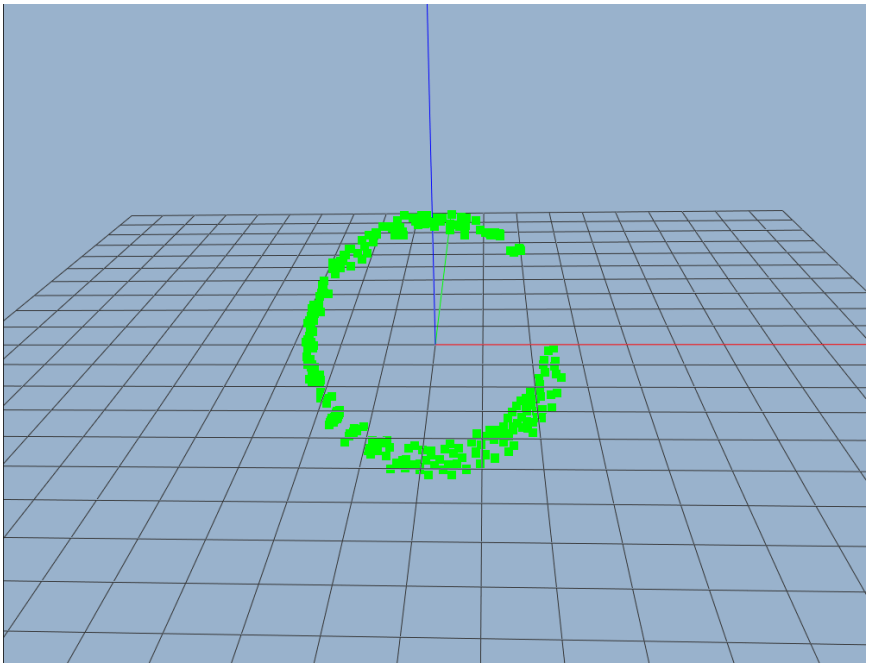

図 3.3　C 群に分類された点群

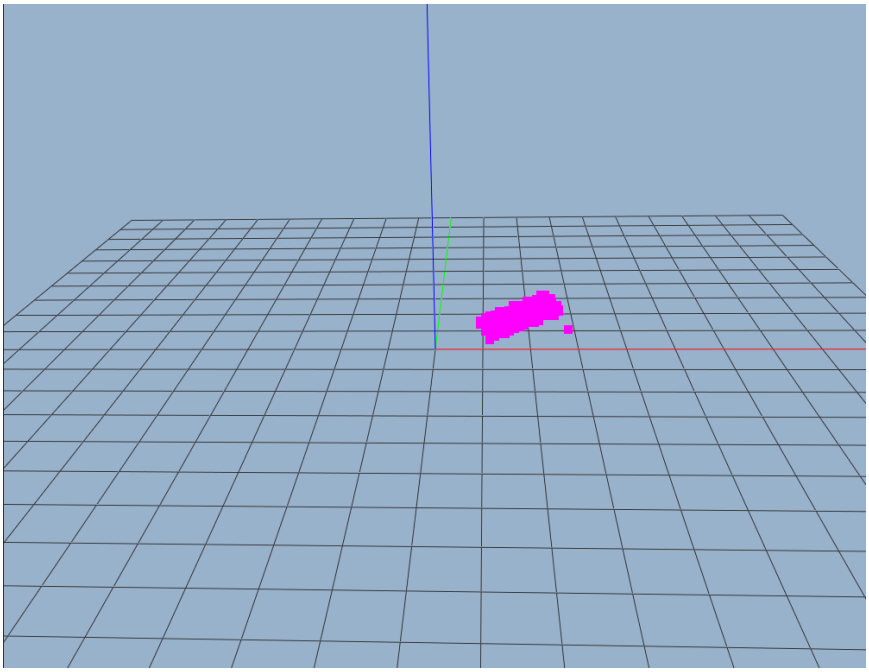

図 3.4　D 群に分類された点群

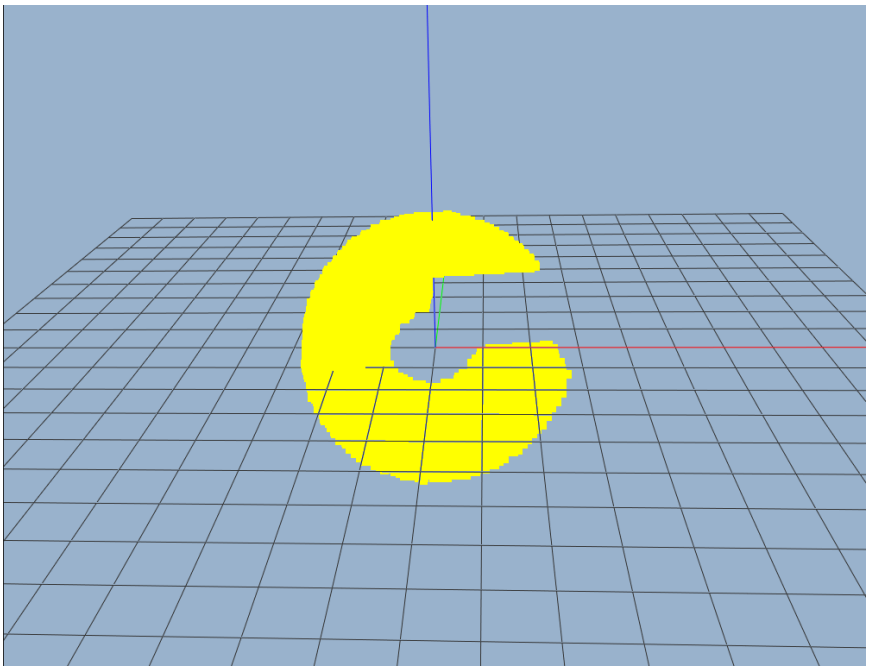

図 3.5　E 群に分類された点群

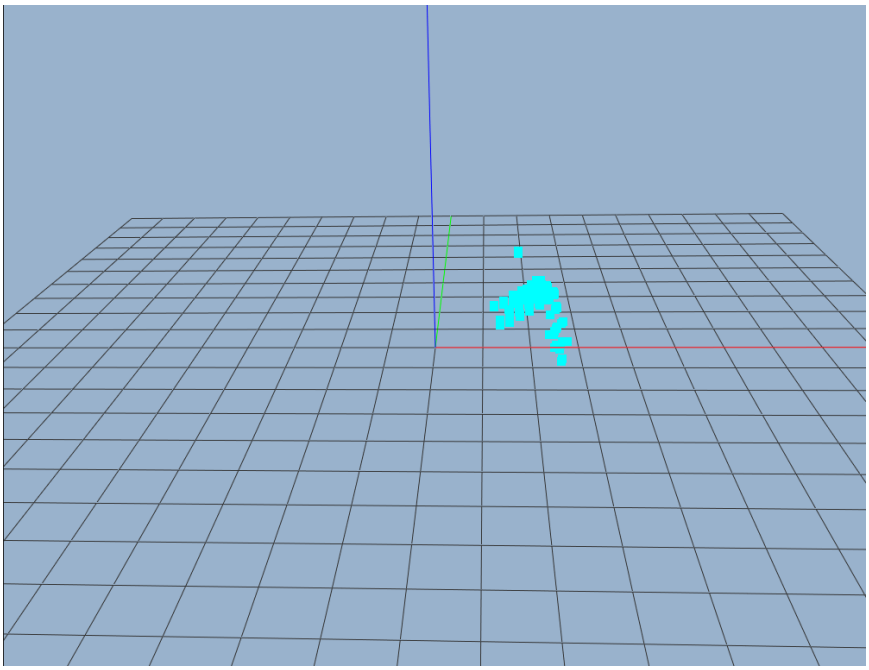

図 3.6　F 群に分類された点群

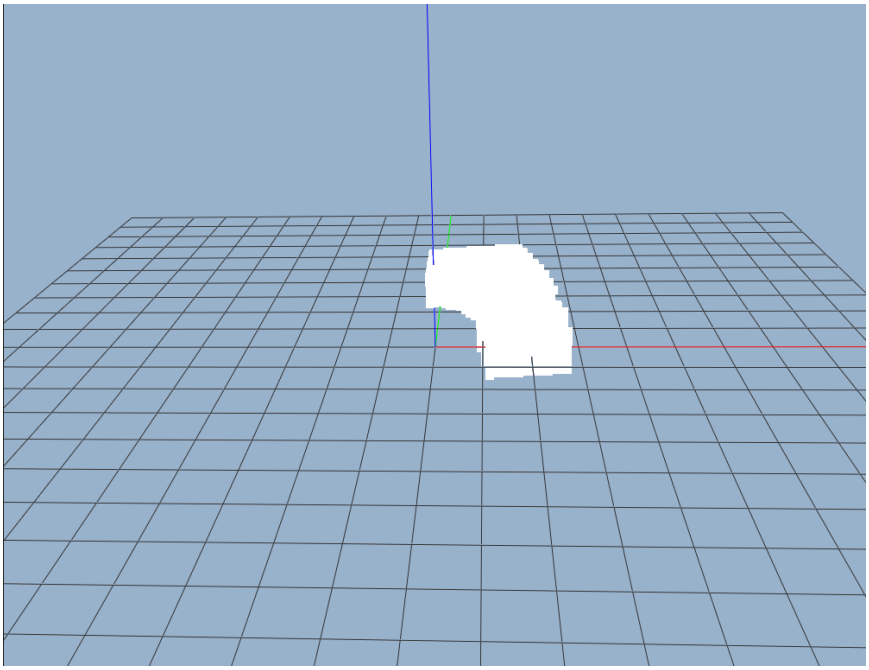

図 3.7　G 群に分類された点群

各分類の個数を n_1 から n_7 で表したとき、S-CCD 手法の衝突判定精度は式 (3.1) から、扇形による補間の衝突判定精度は式 (3.2) から求められる。

$$\frac{n_4 + n_7}{(\sum_{i=1}^{7} n_i) - n_3} \tag{3.1}$$

$$\frac{n_5 + n_7}{(\sum_{i=1}^{7} n_i) - n_2} \tag{3.2}$$

また、それぞれの手法が誤判定を行う場合の違いについて分析を行う。誤判定とは、ある剛体 α と回転運動を行う剛体 β が存在するとき、剛体 α と剛体 β の回転運動軌跡の衝突判定結果と、剛体 β の回転運動に対して CCD 手法を適用した際の衝突判定結果が異なることである。誤判定には過剰判定と不足判定の 2 つの種類が存在する。以下に、2 つの誤判定の違いを示す。

- 過剰判定：剛体 α と剛体 β の回転運動軌跡は衝突していないが、CCD 手法を適用した場合に衝突していると判定してしまう場合
- 不足判定：剛体 α と剛体 β の回転運動軌跡は衝突しているが、CCD 手法を適用した場合に衝突していないと判定してしまう場合

衝突判定精度同様に、S-CCD 手法と提案手法の過剰判定率、不足判定率を算出する。S-CCD 手法の過剰判定率は式 (3.3) から、不足判定率は (3.4) から算出でき、提案手法の過剰判定率は式 (3.3) から、不足判定率は (3.4) から算出できる。

$$\frac{n_2 + n_6}{(\sum_{i=1}^{7} n_i) - n_3} \tag{3.3}$$

$$\frac{n_1 + n_5}{(\sum_{i=1}^{7} n_i) - n_3} \tag{3.4}$$

$$\frac{n_3 + n_6}{(\sum_{i=1}^{7} n_i) - n_2} \tag{3.5}$$

$$\frac{n_1 + n_4}{(\sum_{i=1}^{7} n_i) - n_2} \tag{3.6}$$

10 万件の直方体の回転運動データから、S-CCD 手法と提案手法の平均衝突判定精度、平均過剰判定率、平均不足判定率を算出する。ただし、式 (3.2) か式 (3.1) のどちらか一方でも分母の値が 0 になるようなデータは含まれない。

次に、回転角度による衝突判定精度の違いと、直方体の形状による衝突判定精度の違いについて調べる。回転角度による衝突判定精度の違いは、検証プログラムより得られた 10 万件のデータを回転角度を 10 度ずつ区切った 36 個の回転角度範囲に分類し、それぞれの回転角度範囲ごとに平均衝突判定精度を算出した。ただし、角度範囲ごとのデータ数はいずれも 2600 件から 2900 件の間であった。直方体の形状による衝突判定精度の違いは、まず、検証プログラムより得られた 10 万件のデータそれぞれに対し、直方体の縦幅から横幅を引いた縦横差を算出する。ここで直方体の縦幅とは、第 2.1 節で述べた直方体の厚みのうち $\mathbf{Y}'$ 方向の厚みである W_y のことを指し、横幅とは $\mathbf{X}'$ 方向の厚みである W_x のことを指す。算出した縦横差から 10 万件のデータを昇順に並べ、5000 件のデータごとに平均衝突判定精度を算出した。

3.2 衝突判定精度の比較

表 3.1 に各手法の衝突判定精度の検証結果を示す。

表 3.1 各手法の衝突判定精度と誤判定率

	S-CCD	提案手法
平均衝突判定精度	4.27%	67.68%
平均過剰判定率	93.90%	1.25%
平均不足判定率	1.83%	31.07%

表 3.1 から、提案手法のほうが S-CCD 手法よりも 63.41% 高精度で衝突判定を行えていることが判明した。また、それぞれの手法が誤判定を行った際、S-CCD 手法は過剰判定を行う確率が高く、提案手法は不足判定を行う確率が高いことが判明した。

次に、図 3.8 に S-CCD 手法、提案手法の回転角度範囲ごとの衝突判定精度を折れ線グラフにまとめた図を示す。

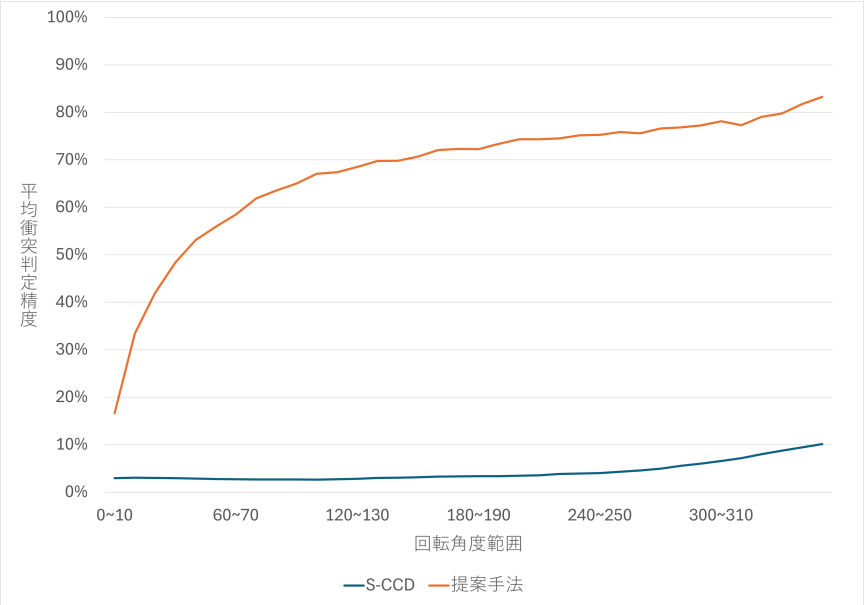

図 3.8　回転角度範囲ごとの衝突判定精度

図 3.8 から、どの回転角度範囲においても提案手法は S-CCD 手法よりも平均衝突判定精度が高いことが判明した。提案手法の衝突判定精度のみに注目すると、回転角度が大きくなるほど衝突判定精度も高くなる傾向にあることが判明した。

最後に、図 3.9 に S-CCD 手法、提案手法の直方体の形状の違いによる衝突判定精度を折れ線グラフにまとめた図を示す。

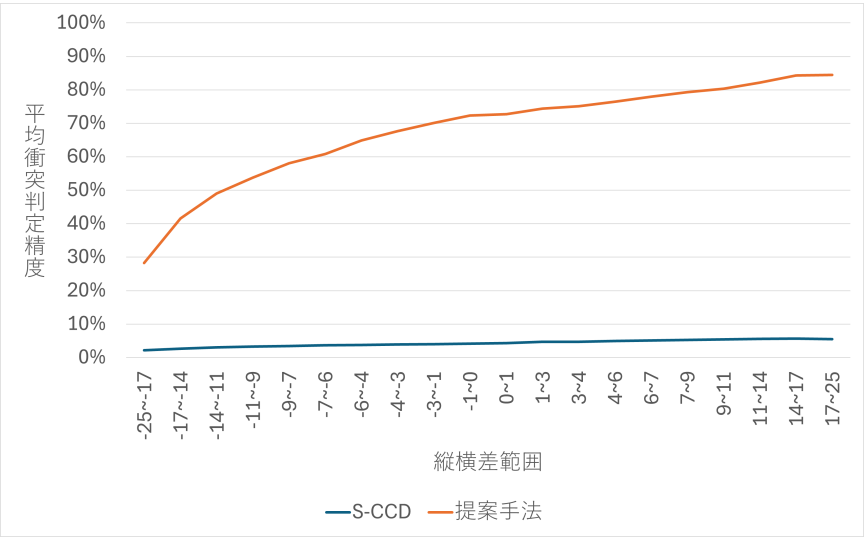

図 3.9　直方体の縦横差ごとの衝突判定精度

図 3.9 から、直方体の形状に関わらず提案手法は S-CCD 手法よりも平均衝突判定精度が高いことが判明した。提案手法の衝突判定精度のみに注目すると、縦幅のほうが横幅よりも大きくなるにつれ平均衝突判定精度も高くなっていることが判明した。

3.3 実行速度の検証

本研究では、第 2 章で提案した扇形とプリミティブ形状との干渉判定の実行速度を調査し、DCD 手法の一例である境界ボリューム手法と、CCD 手法の一例である S-CCD 手法との比較を行った。

始めに、DCD 手法の一例である境界ボリューム手法との実行速度の比較方法を述べる。境界ボリュームは次の 4 つの形状の干渉判定手法と提案手法の実行速度の比較を行った。

- 球
- カプセル
- AABB
- OBB

実行速度の計測では、各手法を 10 万回連続で実行したときの実行速度を計測した。各形状の形や大きさ、向きは C#言語の疑似乱数機能を用いて決定した。ただし、扇形は第 2.1 節で述べた、判定数式を簡易化するための条件を必ず満たすように扇形の回転中心点 O、回転軸 **U**、中心軸 **A** の値は設定した。そのため、座標変換は行っていない。各手法の実行速度データを 1 万件ずつ集計し、平均値、最低値、最高値を算出し比較を行った。

次に、CCD 手法の一例である S-CCD 手法との実行速度の比較方法を述べる。CCD 手法の実行速度の計測では、第 3.1 節で述べた方法により決定される直方体の回転運動に対して CCD 手法を適応したときに、C#言語の疑似乱数機能によって決定された中心点の位置、半径を持つ 10 万個の球と衝突判定を行ったときの実行速度を計測する。ただし、DCD 手法の実行速度の計測とは違い、直方体の回転運動から各手法によって運動軌跡の補間範囲を算出する時間も含み、第 3.1 節で述べた直方体の回転運動では回転軸は疑似乱数機能により決定されるため、提案手法の実行速度の計測では球の中心点に対する座標変換にかかる時間も含む。そこで、実際に衝突したかど

うかを計算する座標変換と干渉判定のみを行ったときの実行速度と、運動軌跡の補間形状の計算のみを行ったときの実行速度、どちらも同時に行ったときの実行速度の 3 つの実行速度を 1 万件ずつ集計し、平均値、最低値、最高値を算出し比較を行った。

実行速度の検証プログラムは、以下の環境で実行した。

- OS : Windows 10
- CPU : Intel(R) Core(TM) i9-10900K CPU @ 3.70 GHz
- メモリ : 32GB

3.4 実行速度の比較

本節に記載されている実行速度はすべて、単位は ms(ミリセカンド) である。

まず、各 DCD 手法の実行速度の平均速度、最高速度、最低速度を表 3.2 と表 3.3 に示す。また、図 3.10 と図 3.11 に各 DCD 手法の実行速度の箱ひげ図を示す。

表 3.2　二次元平面における DCD 手法と提案手法の実行速度 (ms)

	球	カプセル	AABB	OBB	扇形と円	扇形と長方形	扇形とカプセル
平均速度	9.26	55.24	5.00	36.42	41.03	294.10	50.54
最高速度	3.21	26.13	2.71	11.44	4.70	200.97	15.02
最低速度	64.96	238.98	18.00	149.74	92.60	538.34	109.07

表 3.3　三次元空間における DCD 手法と提案手法の実行速度 (ms)

	球	カプセル	AABB	OBB	扇形と点	扇形と球	扇形とカプセル
平均速度	10.43	48.80	5.91	35.81	1.71	24.69	90.97
最高速度	2.88	24.03	2.72	16.00	1.01	3.64	49.77
最低速度	66.47	133.68	24.75	104.63	3.70	81.99	135.87

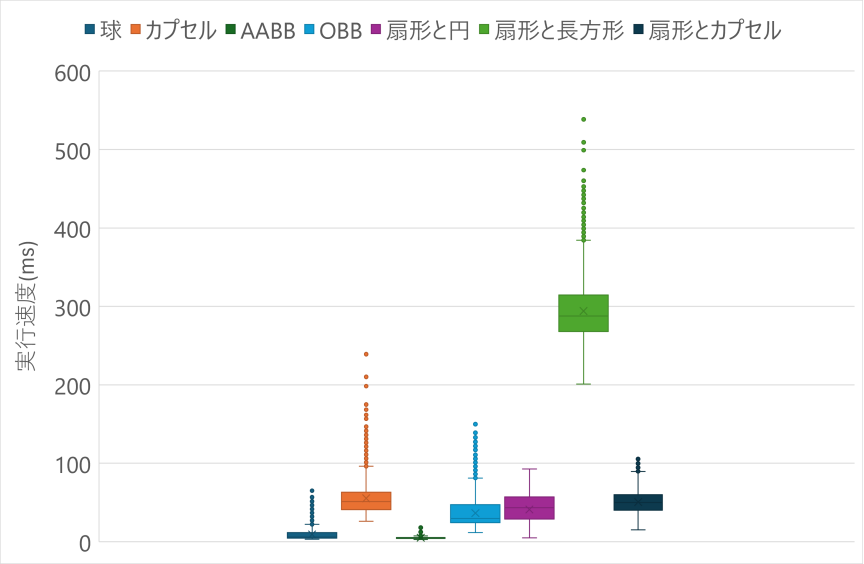

図 3.10　二次元平面における DCD 手法の実行速度 (ms) の箱ひげ図

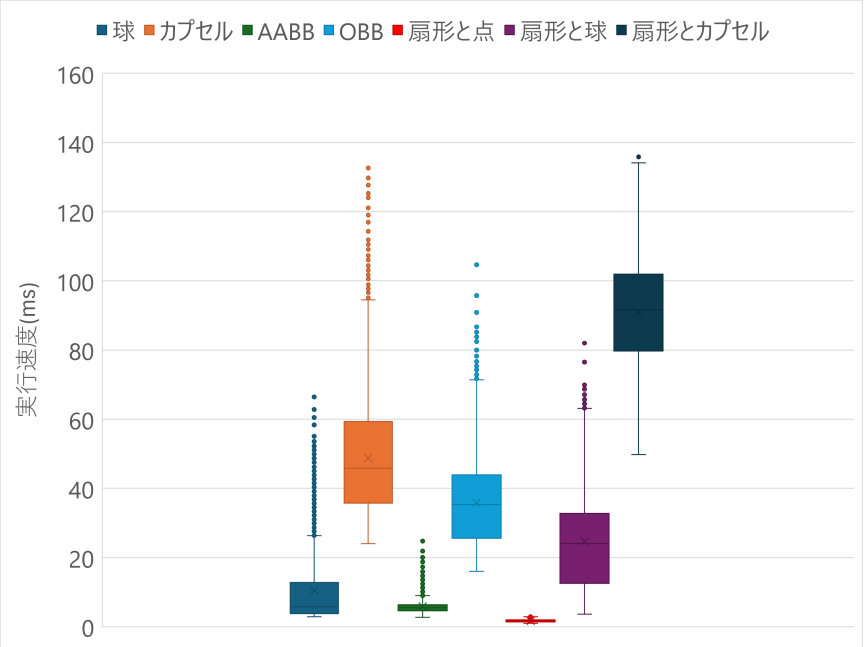

図 3.11　三次元空間における DCD 手法の実行速度 (ms) の箱ひげ図

まず、二次元の手法に注目する。扇形と円は平均速度ではカプセル同士の干渉判定手法よりは高速であることが判明した。ただし、最高速度、最低速度を見ると、カプセルのほかに OBB 同士の干渉判定よりも実行速度は優れていることが判明し、リアルタイムでの実行に十分に利用できることが判明した。続いて、扇形と長方形に注目すると、どの値においても他の手法よりも非常

に実行速度が遅いことが判明した。少ない数の剛体同士の判定であれば、リアルタイムでの実行は問題ないが、非常に多くの剛体同士の干渉判定として利用する事は難しい可能性があることが判明した。最後に、扇形とカプセルに注目すると、概ね扇形と円と同じ評価を下すことが出来ることが判明した。ただし、扇形と円よりはどの値においても遅い実行速度であることが判明した。

次に、三次元の手法に注目する。まず扇形と点に注目すると、どの値においても非常に高速であり、球同士の干渉判定や AABB 同士の干渉判定よりも高速に内外判定を行えていることが判明した。次に扇形と球を見ると、扇形と円の時とは違いカプセル同士の干渉判定手法のほかに、OBB 同士の干渉判定手法よりも全ての値において高速であるということが判明した。最後に扇形とカプセルに注目すると、どの手法よりも実行速度が遅いことが判明した。ただし、二次元の手法である扇形と長方形ほど遅いわけではなく、最低速度はカプセル同士の手法とほぼ同速度であるなど、リアルタイムでの活用には問題ないことが判明した。

続いて、S-CCD 手法と提案手法の座標変換と干渉判定のみを行ったときの実行速度を表 3.4 と箱ひげ図 3.12 に、運動軌跡の補間形状の計算のみを行ったときの実行速度を表 3.5 と箱ひげ図 3.13 に、どちらも同時に行ったときの実行速度を表 3.6 と箱ひげ図 3.14 に示す。

表 3.4　座標変換と干渉判定のみを行ったときの CCD 手法の実行速度 (ms)

	S-CCD	提案手法
平均速度	13.25	112.02
最高速度	3.77	69.23
最低速度	48.12	193.11

表 3.5　運動軌跡の補間形状の計算のみを行ったときの CCD 手法の実行速度 (ms)

	S-CCD	提案手法
平均速度	183.81	22.02
最高速度	163.19	18.80
最低速度	286.69	41.47

表 3.6　CCD 手法の実行速度 (ms)

	S-CCD	提案手法
平均速度	199.95	119.45
最高速度	164.40	95.87
最低速度	276.81	239.62

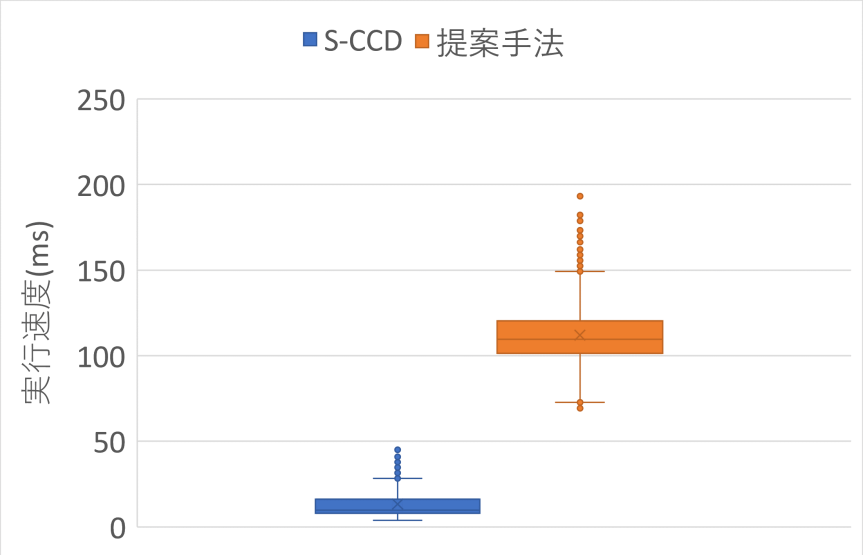

図 3.12　座標変換と干渉判定のみを行ったときの CCD 手法の実行速度 (ms) の箱ひげ図

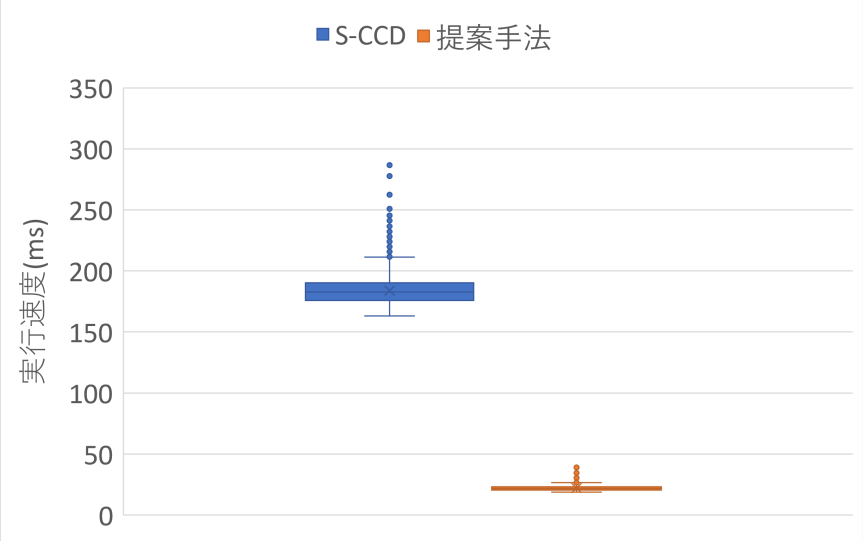

図 3.13　運動軌跡の補間形状の計算のみを行ったときの CCD 手法の実行速度 (ms) の箱ひげ図

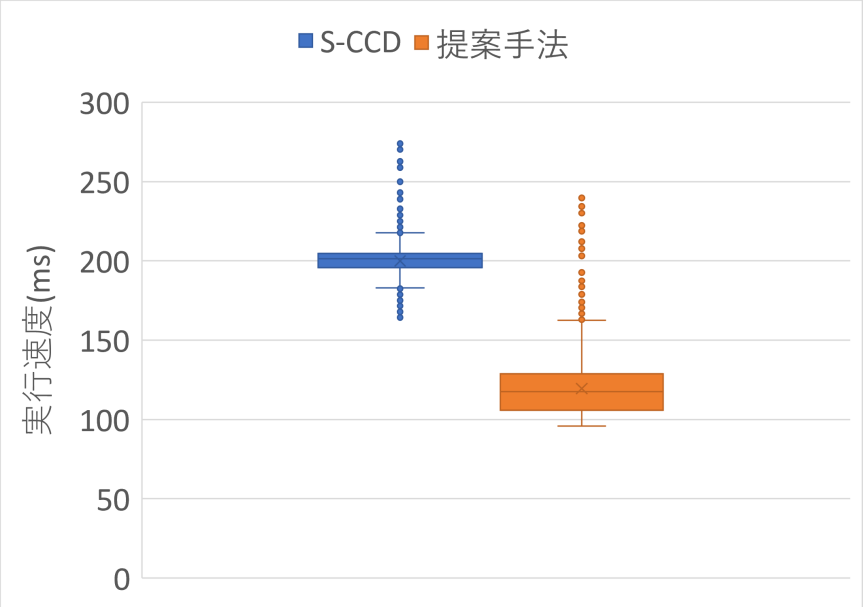

図 3.14　CCD 手法の実行速度 (ms) の箱ひげ図

始めに、座標変換と干渉判定のみを行ったときの実行速度に注目すると、S-CCD 手法に比べ提案手法の平均速度は約 100ms ほど遅い実行速度であることが判明した。また、最高速度、最低速度を見ても S-CCD 手法と提案手法の実行時間の差は大きいということが判明した。次に、運動軌跡の補間形状の計算のみを行ったときの実行速度に注目すると、座標変換と干渉判定のみを行ったときの実行速度とは逆に、提案手法のほうが S-CCD 手法よりも非常に高速であることが判明した。最後に、どちらも同時に行ったときの実行速度に注目すると、S-CCD 手法よりも提案手法のほうが高速であることが判明した。このことから、S-CCD 手法の実行速度の大部分が運動軌跡の補間形状であるのに対し、提案手法の実行速度の大部分は座標変換と干渉判定であることが判明した。

第 4 章

考察

本章では第 2.7 節で述べた三次元空間におけるカプセル形状との干渉判定にて発生する誤判定の解決方法についてと、第 3 章の検証結果、提案手法の応用性、発展性について考察を述べる。

4.1 三次元空間におけるカプセル形状との干渉判定にて発生する誤判定の解決方法について

第 2.7 節では、三次元空間におけるカプセルと扇形の干渉判定問題を、扇形の厚み空間からなるカプセルの二次元形状（疑似カプセル）を考え、疑似カプセルを二次元平面におけるカプセル形状で囲むことで、二次元のカプセルとの干渉判定問題に置き換えていた。疑似カプセルは完全なカプセル形状や円形状になる場合があり、この場合には誤判定なく干渉判定を行える。しかし、完全なカプセル形状や円形状以外になる場合、疑似カプセルと疑似カプセルを囲むカプセルの間にできる隙間により、誤判定が生じる。疑似カプセルの形状は、三次元空間におけるカプセルを構成する始点と終点を結ぶ線分上の点を xy 平面に投影した点を中心点とし、半径を第 2.5 節で述べた R_m とする円群 C によりわかる。そのため、円群 C のうち一つでも扇形と干渉している円を見つけることが出来れば、三次元空間におけるカプセルと扇形の干渉判定をどんな場合でも誤判定なく行うことが出来ると考える。本稿で提案した扇形と円の干渉判定は、距離の判定、角度の判定、扇形を構成する線分と円の干渉判定という 3 つの処理を行っている。これら 3 つの処理を円群 C に対応した処理に拡張・修正することが出来れば、この誤判定問題を解決できる可能性がある。

4.2 衝突判定精度について

本稿で提案する扇形を用いた回転運動の補間は、第 3.2 節で述べた通り、S-CCD 手法よりも高い平均衝突判定精度である。また、回転角度が大きくなるにつれ、平均衝突判定精度も上昇する手法である。ただし、前述の結果は言い換えると回転角度が小さい時は平均衝突判定精度が低いということでもあり、本節ではその原因について考える。

考えられる原因として、直方体の回転運動から扇形を構築する際に、扇形の大きさが不適切であることが挙げられる。提案手法では、回転前の直方体の中心から回転後の直方体の中心を補間するように扇形を構築する。そのため、回転角度が小さい時には構築される扇形も細長いものと

なり、扇形によって補間される回転運動軌跡も少ないものとなってしまう。図 4.1 に直方体の回転角度が小さい時、構築される扇形の様子を示す。

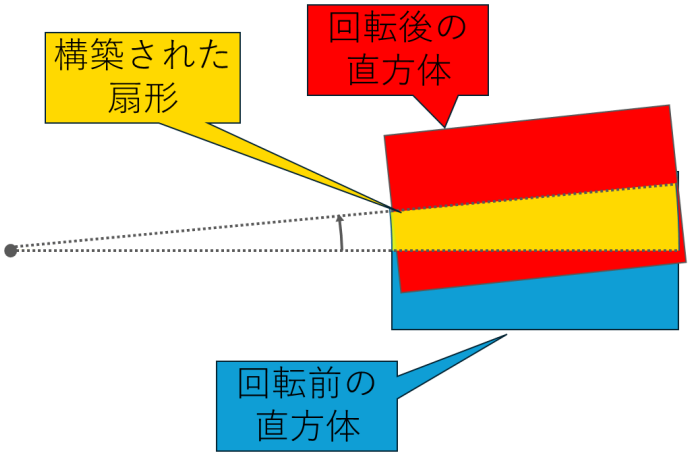

図 4.1　回転角度が小さい時の扇形

この問題を解決するためには、直方体の回転運動から扇形を構築する方法を変える必要がある。提案手法では、扇形を構築する際に直方体の横幅を考慮していない。そこで、扇形を構築する際に直方体の横幅も考慮して扇形の形状を決めることが出来れば、回転角度が小さい場合でも平均衝突判定精度を高くできる可能性があると考える。

続いて、直方体の形状の違いによる平均衝突判定精度の違いについて考える。第 3.2 節の検証結果より、直方体の横幅が縦幅よりも大きい時に平均衝突判定精度が低下することが判明した。これは、前述した通り扇形を構築する際に直方体の横幅を考慮していないことが原因であると考える。ただし、横幅が縦幅よりも大きい直方体が回転運動を行う際、回転運動軌跡は扇形に近しい形状にはなりにくい。また、横幅が縦幅よりも大きい直方体が高速に回転運動を行った場合、本研究で想定している縦幅が横幅よりも大きい直方体が高速に回転運動を行った場合に比べて剛体同士が衝突を無視してすり抜ける確率は低いと考える。そのため、本手法は第 2.1 節で述べた想定通り、縦幅が横幅よりも大きい直方体の回転運動に対して使用するのが望ましいと考えられる。縦幅が横幅が縦幅よりも大きい直方体が高速に回転運動をするコンピュータゲームの例としてはバットや剣などの細長い剛体が登場するものが挙げられ、野球ゲームや、アクションゲーム、VR(Virtual Reality) 剣戟ゲームなどが該当すると考える。

4.3　実行速度について

第 3.4 節の検証結果より、扇形と点の内外判定、扇形と円、扇形と球の干渉判定は比較的高速な実行速度であることが判明した。対して、扇形と長方形、扇形とカプセルの干渉判定は遅い実行速度であることが判明した。長方形との干渉判定が非常に遅い実行速度となった原因としては、基本幾何計算を多用していることであると推測する。扇形と点の内外判定、扇形と点の内外判定、扇形と円、扇形と球の干渉判定ではプリミティブ形状特有の数学的性質を利用している部分が多いのに対し、長方形との干渉判定では線分という幾何学的要素に対する処理を元に干渉判定を行っている。幾何学的要素に対する処理は簡易な処理ではあるものの、剛体同士の干渉判定などに利用すると何度も処理を行わなければいけなくなり、結果として実行速度が低下しやすい。扇形と長方形の干渉判定ほどではないものの、扇形とカプセルの干渉判定の実行速度が遅い理由も同様の点にあると考える。そのため、長方形やカプセルの数学的性質をうまく活用できれば、より高速に干渉判定を行える可能性があると考える。

続いて、S-CCD 手法との実行速度の比較について考察を行う。S-CCD 手法との実行速度の比較結果から、提案手法のほうが高速に衝突判定を行えていることが判明した。提案手法のほうが実行速度が高速だったのは第 3.4 節の検証結果から、運動軌跡の補間形状である扇形を構築するのが高速であったためだとわかる。ただし、第 2.1 節で述べた通り、直方体の回転運動には制約を課している。また、第 4.2 節で述べた通り、現在直方体の回転運動から扇形を構築する際には直方体の横幅は考慮していない。そのため、第 2.1 節で述べた制約を満たさない場合でも扇形が回転運動を補間できるような扇形の構築方法や、回転運動を行う形状を完全に内包するように扇形を構築する方法の場合には、剛体の回転運動から扇形を構築する際により時間がかかると推測する。

4.4　応用性

提案手法には 2 つの応用性があると考える。

一つは、境界ボリュームとしての使用である。二次元形状の扇形は、コンピュータゲームにおいてキャラクターの視覚範囲や攻撃範囲を表すために使用されることがある。そのため、本研究

で定義した三次元形状の扇形にも同様の応用方法があると考える。また、境界ボリュームとして使用することで、衝突判定システム中のブロードフェイズに使用できる可能性がある。ブロードフェイズとは、境界ボリュームなどの高速な干渉判定により、GJK アルゴリズムなどの厳密な干渉判定や衝突判定を行う剛体同士のペアを削減する仕組みであり、本研究で提案手法の比較対象とした S-CCD 手法は主にブロードフェイズで使用される手法である。第 4.3 節で述べたように、直方体以外の様々な形状の回転運動から扇形を構築できるようにする場合、第 3.4 節で述べた結果とは異なり、S-CCD 手法よりも提案手法のほうが実行速度は遅くなる可能性があるが、第 3.2 節で述べた通り、S-CCD 手法よりも提案手法のほうが衝突判定精度は優れているため、厳密な干渉判定、衝突判定を行う剛体同士のペアを S-CCD 手法よりも減らすことができ、結果として多数の剛体同士の干渉判定、衝突判定を高速化できる可能性がある。提案手法をブロードフェイズに用いる方法は、人体とボールの接触など詳細な衝突判定を必要としやすいスポーツゲームなどで利用できる可能性がある。

もう一つの応用性として、衝突予想がある。本稿で定義した扇形は回転運動から構築されるが、回転運動時の回転軸から、数フレーム連続で回転運動を行う剛体が将来的に他の剛体と衝突するかどうかを予想できる。これは VR ゲームなどのセンシングデバイスを用いたコンピュータゲームで、プレイヤーが自身の動きの結果を予想する一助になるのではないかと考える。

4.5 発展性

提案手法には三つの発展性があると考える。

一つ目は、より多くの形状と扇形との干渉判定を定義することである。本研究では、コンピュータゲームに頻出する幾つかのプリミティブ形状との干渉判定を定義した。しかし、近年では、GJK アルゴリズムなどの凸形状同士の干渉判定を利用することにより、より複雑な形状同士の干渉判定を行えている。そこで、扇形との干渉判定をより多くの形状に対して定義できれば活躍の場面が増えるのではないかと考える。また、扇形同士の干渉判定が定義できれば、回転運動を行う剛体同士の干渉判定結果を容易に近似できるため、こちらも提案手法の活躍の場面を増やす一助になるのではないかと考える。

二つ目は、第 4.2 節で述べた回転運動から扇形を構築する方法を変えることである。本研究で

は回転運動を行うのは直方体と限定しているが、他の形状の回転運動に対しても扇形を構築できるようになれば、より利用しやすい手法になると考えられる。また、本研究では、回転運動時の直方体の姿勢に制約を課したが、任意の姿勢に対しても扇形を構築できるようになれば、回転運動の補間手法としてより活躍の場面を増やせるのではないかと考える。

三つ目は、並列化を考慮した最適化である。近年では干渉判定や衝突判定を GPU を用いて高速化する研究 [28][29][30] もおこなわれており、本稿で提案した手法も並列化を考慮した最適化を行えれば、よりリアルタイム環境での利用がしやすい手法になると考えられる。

第 5 章

まとめ

本稿では、直方体の回転運動軌跡の近似形状である扇形を定義し、直方体の回転運動を扇形によって補間し衝突判定を行う CCD 手法を提案した。

扇形は非凸の形状になる場合があり、GJK アルゴリズムなどの凸形状に対する干渉判定手法は使えないため、本稿ではコンピュータゲームに頻出する幾つかのプリミティブ形状と扇形との干渉判定を提案した。扇形と干渉判定を行うことのできるプリミティブ形状として二次元形状である円、長方形、カプセルと、三次元形状である点、球、カプセルがある。ただし、三次元形状であるカプセルと扇形の干渉判定は近似解の算出となっている。

提案手法が回転運動に対応した CCD 手法の一例である S-CCD 手法よりも高い衝突判定精度であるかを確認するため、提案手法と S-CCD 手法の平均衝突判定精度を算出し比較した。その結果、提案手法は S-CCD 手法よりも平均衝突判定精度が高く、回転角度ごとに平均衝突判定精度を算出すると、提案手法は回転角度が上昇するにつれて平均衝突判定精度も上昇することが判明した。

また、提案手法の実行速度を算出し、DCD 手法の一例である境界ボリューム手法と、CCD 手法の一例である S-CCD 手法と比較することで、リアルタイムでの実行に問題はないか確認した。境界ボリューム手法と比較を行った結果、扇形と点、扇形と円、扇形と球の干渉判定は十分に高速な実行速度であるが、扇形と長方形、扇形とカプセルの干渉判定は OBB 同士の干渉判定などと比べても遅い実行速度であることが判明した。S-CCD 手法との実行速度の比較では、実際に衝突したかどうかを判定する干渉判定の実行速度では提案手法のほうが劣るものの、運動軌跡の補間形状計算の実行速度では提案手法のほうが S-CCD 手法よりも優れており、2 つの実行速度を合計した場合、提案手法のほうが S-CCD 手法よりも高速に衝突判定を行えていることが判明した。

提案手法は縦幅が横幅よりも大きい直方体のような細長い剛体が回転運動をする際に衝突判定精度が非常に高精度になるため、野球ゲームや剣などが登場するアクションゲーム、VR 剣戟ゲームにおいて有効に活用できると考えられる。ただし、本研究では回転運動に対して制約を課しているため、より多くの場面で提案手法を活用できるようにするためには、剛体の回転運動から扇形を構築する方法を改善する必要がある。

謝辞

本研究を進めるにあたり多くのご指導やアドバイスをして頂いた渡辺先生と阿部先生に心より感謝いたします。

渡辺先生には学部 1 年次から修士 2 年までの 5 年間、授業や演習、先端メディア、研究活動など多くの場面で大変お世話になりました。大学入学当初には漠然としていたゲーム制作に関わりたいという心がゲームプログラミングへの関心に変化したのは、渡辺先生の授業やお話しの影響が大きかったです。研究テーマを決める際にも衝突判定というテーマを選択したのは、先端メディアなどを通してゲームプログラミングの奥深さを知れたからです。研究活動においても、私一人では行き詰ってしまった際に新しいアイディアのヒントとなるようなアドバイスを頂けたり、本当に感謝しています。

阿部先生には学部 3 年次から研究活動においてサポートいただき、大変お世話になりました。毎週の研究室ミーティングで私が発表を行った際に阿部先生から頂いたアドバイスで、自身の発表の改善点に気づくことが出来たり、研究を進めるための方針がずれていないか確認することが出来ました。本当に感謝しています。

また、卒業した先輩方を含めた同じ研究室の仲間たちと、学部からの付き合いである友人たちに心より感謝します。

先輩方には研究以外にも就職活動や大学院での授業などのアドバイスをいただき、大変お世話になりました。TA の時間などに色々なお話をしていただき、授業や就職活動など大学院生活の中での行動指針を定めることが出来ました。また、研究でも議論を通して様々なアドバイスをいただき、研究活動を頑張るモチベーションを高めることが出来ました。本当に感謝しています。

友人たちにはと過ごした時間は、大学、大学院生活を通して心の支えになりました。自分の好きなものや研究について語り合ったり、ともに遊んだりした時間はこの 5 年間の生活を非常に有意義なものにしてくれました。嫌なことがあった時に相談することで心が沈みすぎないようにしてくれたり、嬉しいことがあった時には共有することでこちらもよい時間を過ごすことが出来ました。本当にありがとう。

最後に、常に影から支えてくれていた母に心から感謝します。小さいころからずっとゲームが好きだった私のために、ゲーム制作を学ぶことが出来るこの大学を見つけてくれて嬉しかったです。私はこの大学に入ることが出来てよかったです。ずっと迷惑をかけ続けてしまっていますが、

あなたのおかげで私はここまで成長することが出来ました。本当に感謝しています。

再度になりますが、大学生活、大学院生活を通してお世話になった皆様に、心より感謝の意を述べます。本当にありがとうございました。

参考文献

[1] Lazaros Lazaridis, Maria Papatsimouli, Konstantinos-Filippos Kollias, Panagiotis Sarigiannidis, and George F.Fragulis. Hitboxes: A Survey About Collision Detection in Video Games. In *HCI in Games: Experience Design and Game Mechanics*, pp. 314–326. Springer, 2021.

[2] Chaoqiang Tu and Lizhen Yu. Research on Collision Detection Algorithm Based on AABB-OBB Bounding Volume. In *2009 First International Workshop on Education Technology and Computer Science*, Vol. 1, pp. 331–333, 2009.

[3] Zhou xiangning. Research of collision detection based on obb in skinned mesh. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, Vol. 6, pp. V6–643–V6–645, 2010.

[4] Dehan Kong, Yongshan Liu, and Na Cui. Collosion Detection Research Based on Capsule Bounding Volume *. *Journal of Computational Information Systems*, Vol. 10(7), p. 2743 – 2750, 2014.

[5] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, No. 1, pp. 21–36, 1998.

[6] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, Vol. 4, No. 2, pp. 193–203, 1988.

[7] ChristerEricson 著, 中村達也訳. ゲームプログラミングのためのリアルタイム衝突判定. 株式会社ボーンデジタル, 2005.

[8] bulletphysics.org. Bullet real-time physics simulation. `https://pybullet.org/wordpress/`. 参照: 2023.8.29.

[9] Cheng Liang and Xiaojian Liu. The research of collision detection algorithm based on separating axis theorem. *Int. J. Sci*, Vol. 2, No. 10, pp. 110–114, 2015.

[10] Simena Dinas. *Bounding Volume Hierarchies for Rigid Bodies*, pp. 1–5. 01 2019.

[11] Gino van den Bergen. Efficient Collision Detection of Complex Deformable Models Using

AABB Trees. *J. Graph. Tools*, Vol. 2, No. 4, p. 1 – 13, 1998.

[12] S.Gottschalk, M.C.Lin, and D.Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. *Computer Graphics*, Vol. 30, p. 171 – 180, 1997.

[13] Daniel Meister, Shinji Ogaki, Carsten Benthin, Michael J.Doyle, Michael Guthe, and Jiri Bittner. A Survey on Bounding Volume Hierarchies for Ray Tracing. *Computer Graphics Forum*, Vol. 40, pp. 683–712, 2021.

[14] Quan Nie, Yingfeng Zhao, Li Xu, and Bin Li. A survey of continuous collision detection. In *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*, pp. 252–257, 2020.

[15] Hamzah Asyrani Sulaiman, Mohd Azlishah Othman, Mohd Muzafar Ismail, Maizatul Alice Meor Said, Azri Ramlee, Mohamad Harris Misran, Abdullah Bade, and Mohd Harun Abdullah. Distance computation using axis aligned bounding box (AABB) parallel distribution of dynamic origin point. In *2013 Annual International Conference on Emerging Research Areas and 2013 International Conference on Microelectronics, Communications and Renewable Energy*, pp. 1–6, 2013.

[16] Stephane Redon, Abderrahmane Kheddar, and Sabine Coquillart. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, Vol. 21, , 05 2002.

[17] Edvin Åblad, Domenico Spensieri, Robert Bohlin, and Ann-Brith Strömberg. Continuous Collision Detection of Pairs of Robot Motions Under Velocity Uncertainty. *IEEE Transactions on Robotics*, Vol. 37, No. 5, pp. 1780–1791, 2021.

[18] Unity Technologies. CCD (連続的衝突判定). `https://docs.unity3d.com/ja/2019.4/Manual/ContinuousCollisionDetection.html`. 参照: 2022.6.9.

[19] NVIDIA Corporation. Advanced Collision Detection. `https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/AdvancedCollisionDetection.html`. 参照: 2022.7.12.

[20] Eric Larsen, Stefan Gottschalk, Ming Lin, and Dinesh Manocha. Fast Proximity Queries with Swept Sphere Volumes. *Technical Report TR99-018, Department of Computer*

Science, Universityl of North Carolina, Chapel Hill, 1999.

[21] Respawn Entertainment. Extreme SIMD: Optimized Collision Detection in 'Titanfall'. https://www.gdcvault.com/play/1025126/Extreme-SIMD-Optimized-Collision-Detection. 参照: 2022.6.27.

[22] Shankar Krishnan, Amol Pattekar, Ming C. Lin, and Dinesh Manocha. Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries. In *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics : The Algorithmic Perspective: The Algorithmic Perspective*, WAFR '98, pp. 177–190, USA, 1998. A. K. Peters, Ltd.

[23] KIT 物理ナビゲーション. ロドリゲスの回転公式 (Rodrigues' rotation formula). https://w3e.kanazawa-it.ac.jp/math/physics/category/physical_math/linear_algebra/henkan-tex.cgi?target=/math/physics/category/physical_math/linear_algebra/rodrigues_rotation_formula.html. 参照: 2023.2.6.

[24] zu_rin. 2 線分の交点座標 (2 次元). https://qiita.com/zu_rin/items/09876d2c7ec12974bc0f. 参照: 2022.6.17.

[25] yaketake08. 直線 (線分) と円の交点. https://tjkendev.github.io/procon-library/python/geometry/circle_line_cross_point.html. 参照: 2022.6.17.

[26] IKD. ○×つくろーどっとコム ゲームつくろー! 衝突判定編. http://marupeke296.com/COL_main.html. 参照: 2022.7.1.

[27] Fine Kernel Project. Fine kernel toolkit system. https://gamescience.jp/FK/. 参照: 2023.8.20.

[28] Peng Du, Elvis S. Liu, and Toyotaro Suzumura. Parallel continuous collision detection for high-performance gpu cluster. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '17, New York, NY, USA, 2017. Association for Computing Machinery.

[29] Tianyu Wang, Jiong Chen, Dongping Li, Xiaowei Liu, Huamin Wang, and Kun Zhou. Fast gpu-based two-way continuous collision handling. *ACM Trans. Graph.*, Vol. 42,

No. 5, jul 2023.

[30] Floyd M. Chitalu, Christophe Dubach, and Taku Komura. Bulk-synchronous parallel simultaneous bvh traversal for collision detection on gpus. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '18, New York, NY, USA, 2018. Association for Computing Machinery.

発表実績

ポスター発表

1. 山本輝，阿部雅樹，渡辺大地，回転剛体の衝突判定精度向上に関する研究，NICOGRAPH2022，2022.

口頭発表

1. 山本輝，阿部雅樹，渡辺大地，リアルタイムグラフィックスにおける回転剛体の連続的衝突判定精度向上に関する研究，映像表現・芸術科学フォーラム 2023，2023.

口頭発表 (査読付き)

1. 山本輝，阿部雅樹，渡辺大地，リアルタイムグラフィックスにおける回転剛体の衝突判定精度向上に関する研究，NICOGRAPH2023，2023.