

特別研究報告書

仮想空間を用いた
群集歩行の衝突シミュレーション

指導教員 石田 亨 教授

京都大学工学部情報学科

大石 隆俊

平成 17 年 2 月 10 日

仮想空間を用いた群集歩行の衝突シミュレーション

大石 隆俊

内容梗概

これまでに、計算機上に構築した仮想空間で大規模な群衆の行動を再現するシミュレーションの研究が数多くなされてきた。仮想空間での群衆行動のシミュレーションには、二次元平面上でのシミュレーションと比較して、実際の人間が操作するアバターを投入して行動を調査できる、群衆行動の直感的な理解がしやすいなどの利点がある。このシミュレーションの主な用途のひとつに「群集災害の再現」があり、狭い空間内で多数の人間が行動している過密状態を再現することによって、群集事故の発生を予測し、事故の防止に役立てようとする試みがなされている。しかし既存の研究では人間を粒子で表現したり、人間の行動ルールを簡単な力学的モデルに置き換えたりするなど人間の行動を大幅に簡略化したモデルが使用されており、群衆内の個々の人間同士の社会的インタラクションに基づく動きが表現されていない。

本研究では人間同士の社会的インタラクションを考慮した群衆行動の再現を行うために既存の歩行者モデルを修正し、その修正を仮想空間環境 FreeWalk/Q に実装してより現実に近い群衆行動の再現を行うことを目的とする。

既存の研究による行動モデルとしては Particle Model や Flocking Model など挙げられるが、これらのモデルには以下のような問題が存在する。

1) 歩行者の位置関係だけを考慮した衝突表現

Particle Model では人間を電荷を持った粒子としてモデル化し、歩行者同士の衝突を粒子同士の斥力による反発によって表現している。Flocking Model でもこれと似た単純なモデルが用いられており、歩行者は他の歩行者に近づくと斥力を受けて押し戻される。これらのモデルでは一定以上近づいた歩行者間には常に斥力が働くことになり、目の前を横切る歩行者に道を譲るといった二人の歩行者が意図的に接近する状況を再現できない。さらに、実際の衝突では側面衝突・追突などの衝突の種類によって歩行者が受ける影響は異なるが、斥力による衝突近似ではこの衝突の種類の違いが考慮されない。

2) 歩行者間の社会的インタラクションの欠如

実際の間人は「他人と衝突しそうになったら立ち止まる」「前を歩いている人を見つけたらついて行く」などといった社会的なインタラクションに基づく行動

決定を行っているが，Particle Model では歩行者間の影響は斥力による反発のみで表される．Flocking Model では歩行者は周囲に他の歩行者がいると接近して群集を作るといった協調行動は行うが，それ以外の人間の社会的なインタラクションに基づく行動は表現できない．

前者の問題を解決するため，本研究では以下の衝突モデルを考案した．

1) 人体形状の長方形近似と進行方向を考慮した衝突

歩行者同士の接触判定を行う際に人体の形状を長方形で近似して判定するモデルを考案し，仮想空間環境 FreeWalk に実装した．このモデルによって，Particle Model や Flocking Model では実際よりも大きく表現されていた人体の前後方向に対する衝突判定が正確に行えるようになり，混雑した空間内の歩行で見られる「目の前を横切る歩行者に道を譲る・前を歩く人に間隔を空けずに付いていく」などの歩行者同士の意図的な接近の表現が可能となった．また，歩行者同士の衝突時に両者が受ける力の方向を，衝突直前の両者の進行方向から計算するモデルを考案し，このモデルに基づく衝突の表現を FreeWalk に実装した．これによって現実に近い衝突の表現が可能となり，衝突によって歩行者が受ける影響をより正しく再現することが可能となった．

以上の衝突モデルを実装した FreeWalk を用いて，実空間での群集歩行の例として社会心理学の教授である釘原教授が 2004 年 9 月に行った群集衝突実験の再現を試みた．この実験の被験者の行動シナリオを作成する際に，後者の問題を解決するため，インタラクション設計言語 Q を用いてシナリオの記述を行った．

2) 社会的インタラクションを考慮した歩行者シナリオの作成

社会的インタラクションを再現するため，実験のビデオ映像から歩行者の社会的インタラクションに基づいていると思われるルールを抽出し，インタラクション設計言語 Q を用いて歩行者の行動シナリオを記述した．Q を用いることにより「他人と衝突しそうになったら立ち止まる」「前を歩いている人を見つけたらついて行く」等の社会的なインタラクションに基づく行動の記述が可能となった．

この群集衝突実験は各 25 人の歩行者から構成される二つの歩行群集の衝突を再現した実験で，両群集の進行方向の違いや群集内の歩行者の行動ルール(他人に衝突しそうな場合に道を譲るか譲らないか)の違いが群集の移動効率にどのように影響するかを調査することを目的としている．実際に FreeWalk/Q による再現を行ったところ，既存のモデルと比較して歩行者の移動状況をより正確に再現できるようになったことが確認できた．

Overcrowding Pedestrians' Simulation in Virtual Space

Takatoshi OISHI

Abstract

So far, many studies about virtual space simulation of crowd walking have been made. Compared with simulations in two-dimensional simulations, there are some advantages in simulations in virtual spaces. For example, you can put avatars in the virtual space and research their movement, see crowd movements as they are, and so on. One of the uses of virtual space simulations is the reproduction of crowd disasters. Many experiments have been done to forecast crowd disasters and to prevent accidents. But in the existing experiments, they use excessively simplified pedestrian models to represent human behaviors and some behaviors based on social interactions are ignored.

In this paper, I aim to complement the problems in existing pedestrian models, and to represent crowd behaviors more realistically by implementing that model into FreeWalk, a software which provides virtual space environment.

As examples of existing pedestrian model, we can list "Particle Model" and "Flocking Model". But for those, there are problems below.

1) Representation of collision which considers only pedestrians' positions

In Particle Model, pedestrians are represented as particles which carry electric charge and collisions among pedestrians are represented as the repulsion among particles. In the model, every two particles which came close always has repulsive force. And it can't represent the situation of two pedestrians coming close intentionally -like "one gives up his way to others who goes across just in front of him". Besides, in the real collisions, the influences which pedestrians took from collisions differ variously by the type of collision like "head-on collision" and "rear-end collision". But the difference is not considered in the collision representation by repulsion model.

2) Lack of social interaction among pedestrians

Real humans take behaviors based on social interactions. For example, we sometimes follow others who walk in front of us. In the Particle Model, existence of social interaction is ignored. In the Flocking Model, pedestrians can only make crowd with others in their neighborhood. But they can't take any behaviors

based on social interaction except that.

To solve the former problem, I conceived new pedestrian model below.

- 1) Rectangle approximation of human body and collision model considering pedestrians' walking direction

I conceived collision detection model using rectangle approximation of human body. With this model, we can represent the shape of human body more realistically. So it became possible to represent intentional approach of pedestrians. Besides, I conceived collision computation considering position and direction of pedestrians. With this computation, we can represent collisions among pedestrians more realistically.

I implemented those functions into FreeWalk and tried to realize the crowd walking experiment conducted by Naoki Kugihara, professor of social psychology. To solve the latter problem, I used "Q", the Interaction Description Language.

- 2) Design of pedestrian scenario considering social interaction

To represent social interactions among pedestrians, I analyzed the movie of the experiment and extracted pedestrians' behavior rules which seem to be based on social interactions. By using Q, it became possible to represent pedestrians' behavior like "following others who walk in front".

In the experiment, two crowds consist of 25-pedestrians-each collide each other. Direction of movement and behavior rule of each pedestrian is changed. And the experiment was aimed to see the relation between efficiency of pedestrians movements and those changes.

After realizing the experiment on FreeWalk/Q, I found new pedestrian model I conceived can represent pedestrian movement more realistically compared with existing pedestrian models.

仮想空間を用いた群集歩行の衝突シミュレーション

目次

第1章	はじめに	1
第2章	群集歩行シミュレーション	3
2.1	群集歩行シミュレーション	3
2.2	既存の歩行者モデル	4
2.2.1	Particle Model による群集シミュレーション	4
2.2.2	Flocking Model による群集シミュレーション	6
第3章	仮想空間における歩行者モデル	8
3.1	歩行者の長方形近似	8
3.2	歩行者間の衝突	10
3.2.1	衝突の方向による分類	10
3.2.2	進行方向を考慮した衝突表現	10
第4章	歩行者モデルの実装	11
4.1	長方形近似モデルの実装	11
4.1.1	進行方向を考慮した衝突の実装	14
第5章	社会的インタラクションを考慮した歩行者シナリオ	17
5.1	群集衝突実験	17
5.2	歩行者シナリオの制約	18
5.3	歩行者のシナリオ記述	19
5.4	衝突実験と仮想空間シミュレーションの比較	22
第6章	おわりに	22
	謝辞	23
	付録	

第1章 はじめに

駅や街中など，人間が集まり群集を形成して行動するような場面では人間同士の衝突や押し合いが起こり，それが原因で群集事故が起こることがある．記憶に新しいところでは2001年7月の明石の花火大会の歩道橋事故があるように，群集事故は一度起きると多数の死傷者を出すことが多く，それゆえに群集の行動については事故の防止を目指して以前から様々な研究がなされている．実際の群集事故現場に似た状況を作り出すために，実空間で数人～数十人の人間を歩かせてその動きを観察する，といった実験的な研究も行われているが，このような研究を行う際には実際に歩行者役となる被験者を集める必要があり，調整をしながら何度も繰り返して実験を行うことは費用や時間の面で難しいという問題点がある．

実空間実験のこのような問題点を解消するものとして，近年では計算機上に構築した仮想空間上で群集の歩行を再現するシミュレーションの研究がなされている．計算機を用いたシミュレーションには仮想空間を用いたものの他にも，二次元平面上を用いたシミュレーションがあるが，仮想空間を用いたシミュレーションは二次元のものと比較して，実際の人間が操作するアバターを投入して行動を調査できる，群衆行動の直感的な理解がしやすいなどの利点がある．仮想空間での群衆行動のシミュレーションには，二次元平面上でのシミュレーションと比較して，実際の人間が操作するアバターを投入して行動を調査できる，群衆行動の直感的な理解がしやすいなどの利点がある．現実空間上の人間の動きを仮想空間上で再現するためには，人間の行動を計算機で処理できる形で表現するための何らかのモデル化が必要となる．仮想空間上における人間の行動はすべてこのモデルにしたがって表現されるため，どのようなモデル化を行うかということはそのまま群衆行動の再現性に結びつく重要な選択となる．代表的な歩行者モデルとしては後述する Particle Model や Flocking Model などがあるが，これら既存の歩行者モデルには以下のような問題が存在する．

1) 二者間の位置関係だけを考慮した衝突表現

密度の高い群集が狭い空間を歩行する際には，群集内の歩行者同士の衝突や押し合いが多発し，それが原因で群集事故が発生する．群集事故の状況を再現することを目的として衝突シミュレーションを行う際には，歩行者同士の衝突をできるだけ現実に近い形で再現することがシミュレーション精度の向上につな

がるものと考えられる。

Particle Model では人間を電荷を持った粒子としてモデル化し、歩行者同士の衝突を粒子同士の斥力による反発によって表現している。歩行者同士には互いの距離の二乗に反比例した斥力が働くため、歩行者は一定の距離以上接近することはできない。Flocking Model でもこれに似たモデルが用いられており、歩行者は他の歩行者に近づくと斥力を受けて押し戻される。これらのモデルでは歩行者は互いに一定の距離以上に近づくことが出来ないが、実際の歩行者同士は意図的に接近することがある。目の前を横切る歩行者に道を譲るといった場合がその例で、実際の群集では自分の進路を横切る他の歩行者と衝突しそうになった時に、その場に立ち止まって人間一人がやっと通れるような狭い隙間をあけ、相手を先に通行させて衝突を回避する場合がある。既存の歩行者モデルではこのような「状況に応じて他人との距離を開けたり詰めたりする」という人間同士の社会的なインタラクションに基づく人間の行動を表現できず、歩行者は常に一定の間隔を開けて歩き続ける。

さらに、実際の衝突では側面衝突・追突などの衝突の種類によって歩行者が受ける影響は異なるが、Particle Model や Flocking Model による衝突近似ではこの衝突の種類の違いが考慮されていない。群集歩行をより現実に近い形で再現するためにはぶつかる方向を考慮した衝突計算を行う必要がある。

2) 歩行者間の社会的インタラクションの欠如

実際の人間は常に周囲の環境を知覚しながら次に取る行動を決定している。周囲に大勢の人間がいる群集歩行状況においては、歩行者は「他人と衝突しそうになったら立ち止まる」「前を歩いている人を見つけたらついて行く」などといった他の歩行者との社会的なインタラクションに基づく行動決定を常に行っている。しかし、既存の歩行者モデルでは人間同士の社会的なインタラクションは考慮されていない。Particle Model では歩行者間の影響は斥力による反発のみで表される。Flocking Model では歩行者は周囲に他の歩行者がいると接近して群集を作るといった協調行動は行うが、それ以外の人間の社会的なインタラクションに基づく行動は表現できない。

既存の歩行者モデルのこのような問題点を解消し、より現実に近い群衆歩行の再現を行うため、本研究では新しい歩行処理のモデルを作成し、そのモデルを仮想空間環境である FreeWalk/Q[1][2] に実装してより現実に近い過密群衆行動の再現を行うことを目的とする。

本稿では、第2章でまず群集歩行シミュレーションの概要について述べ、次に既存の歩行者モデルとして Particle Model と Flocking Model を取り上げ、これらのモデルの詳細と、群集歩行シミュレーションを行う際の問題点について述べる。第3章では、第2章で挙げた既存のモデルの、衝突を再現する上での問題点を解決する新しい歩行者モデルについての説明を行い、このモデルの FreeWalk/Q への実装についての説明を第4章で行う。第5章では実空間の群集歩行の例として2004年9月に大阪大学の社会心理学の教授である釘原教授が行った群集歩行実験を取り上げ、新しい歩行者モデルを用いたこの群集歩行の再現について述べる。最後に第6章で本研究のまとめを行う。

第2章 群集歩行シミュレーション

2.1 群集歩行シミュレーション

今日では仮想空間を用いた群集歩行のシミュレーションが様々な場面で使用されている。身近な例としては映画やテレビゲームなどでよく見られる、街の雑踏や軍隊の行進などのCG映像を作成する際に使われる群衆シミュレーションがあるが、このような用途においては、シミュレーションは実際の人間を使用することなく、コンピュータのみを用いてできるだけ大規模な群集行動の映像を作成するという目的で用いられるため、群集全体としての動きの再現性が重視される。群集内の個々の歩行者の行動は単純なモデルを用いて近似され、歩行者同士の社会的なインタラクションの再現や衝突によって歩行者が受ける影響などはあまり考慮されない。

これとは別の用途として、人間の社会心理学的行動や建築物の避難効率を調査する目的で仮想空間を用いた群集歩行シミュレーションが用いられる場合がある [4]。ビル火災の避難時に狭い通路に殺到する群集や、それぞれの人間が違う目的地を目指しているスクランブル交差点の群集歩行を再現し、歩行空間の形状や群集内の歩行者の行動ルールを少しずつ変更して何度もシミュレーションを繰り返すことによって、設計の安全評価や歩行者の行動ルールと移動効率との関係などが調査される。このような用途においては、シミュレーションを行う目的は実際の歩行者の行動を出来るだけ正確に再現することであるため、映画やテレビゲームの場合と違い、歩行者同士の社会的なインタラクションや、衝突によって歩行者が受ける影響を詳細に扱うことが必要となる。

本研究では後者のシミュレーションをより正確に行うために、既存の歩行者モデルの問題点を解消する新しい歩行者モデルを作成することを目的とする。

2.2 既存の歩行者モデル

本節では仮想空間を用いた群集歩行の再現に用いられている既存の歩行者モデルとして Particle Model と Flocking Model を取り上げ、その衝突計算の特徴と問題点について述べる。

2.2.1 Particle Model による群集シミュレーション

Particle Model は歩行者を静電気や磁気を帯びた粒子として表現し、歩行者間の影響を粒子間に働く斥力と引力によって表現するモデルである [3]。図 2.1 に Particle Model による歩行者の衝突の様子を示す。二人の歩行者 walkerA と walkerB はそれぞれ $moveVecA$ と $moveVecB$ の方向に歩いている。この二人は + の静電気を帯びた粒子として表されている。歩行中の両者が図のように接近すると、両者には互いの位置を結んだ直線の方法に斥力が働き、walkerA は $repulsiveVecA$ の方向に力を受ける。その結果、進行方向は斥力の方向と元の進行方向を合成した方向 $alignVecA$ に修正される。その後両者が離れ、斥力の影響が十分に小さくなると進行方向の補正が終わり、walkerA は再び元の進行方向と同じ方向に進むようになるので、結果として walkerA が通る道筋は図の点線 routeA ようになる。walkerB も同様に斥力による進行方向の補正を受けて $alignVecB$ の方向に進み、walkerA と離れた後で再び元の $moveVecB$ の方向に進むようになる。

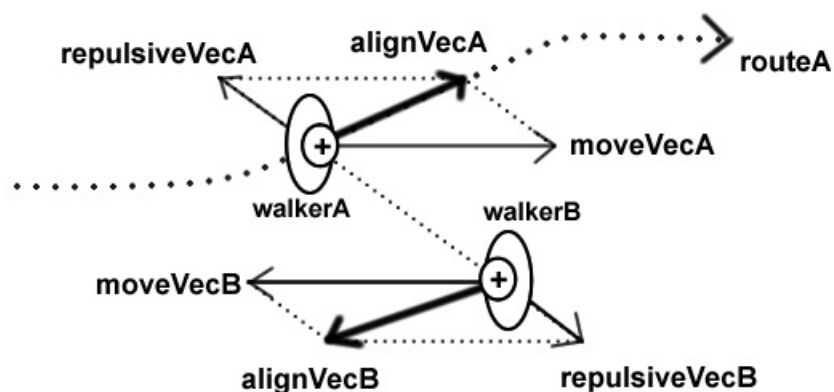


図 2.1: Particle Model による衝突

Particle Model では歩行者の間には常に斥力が働き、歩行者同士は一定の距離以上近づくとすることができない。しかし、実際の人間は歩行の際に他の歩行者と意図的に接近をする場合がある。例として実際の群集の歩行状況で見られる列を形成している歩行者の中を別の歩行者が横切る時に、列の中の一人が道を譲るといった状況を考える(図 2.2)。図の右上方向から左下方向に向かっている 3 人の歩行者集団 A が、図の左上方向から右下方向に向かっている歩行者の列を横切ろうとしている。列の中にある歩行者の一人 B がそのまま列に従って進むと集団 A と衝突することに気付き、道を譲るために立ち止まる。このような状況において、集団 A が実際の歩行者集団であるならば、A は B が立ち止まっているのを見てそのまま直進しても問題がないと判断し、図のように B の目の前を通る進路を取るはずである。しかし、Particle Model によるシミュレーションでは集団 A には歩行者 B との接近による斥力が働いてしまい、結果として A は B が立ち止まっているにもかかわらず、B を大きく迂回する進路を取ることになる。このように、Particle Model では歩行者同士の意図的な接近を表現することができない。上述した歩行者同士の意図的な接近は人間同士の社会的な

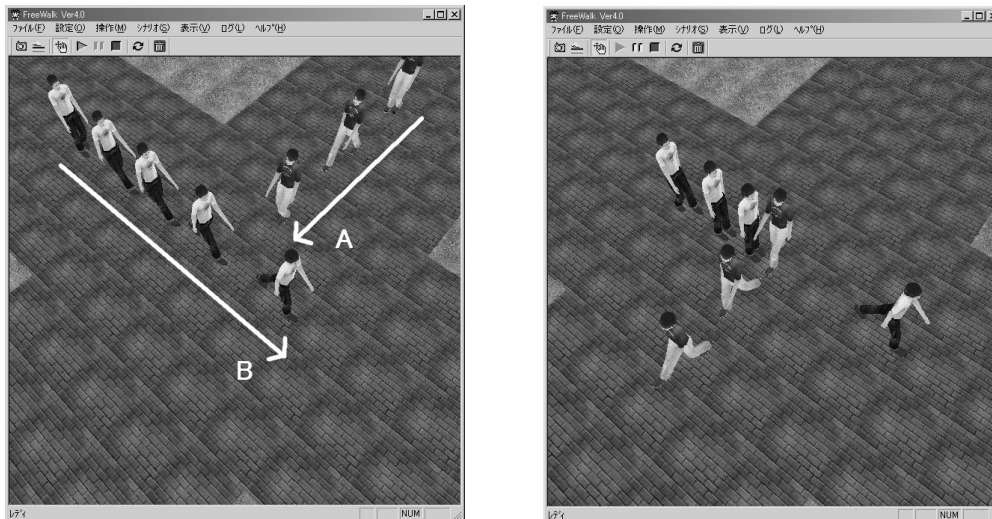


図 2.2: 列の中の一人が道を譲る場合

インタラクションの一種であるが、Particle Model ではこれ以外の社会的なインタラクションも全く考慮されていない。Particle Model では歩行者同士の相

相互作用は粒子の斥力のみに限られており、「同じ方向に向かって歩いている人を前方に見つけたらその人に付いていく」などといった実際の人間の行動ルールを表現することができない。しかし、実際の人間はこのような他人との同士の社会的なインタラクションに基づく行動決定を常に行っている。

2.2.2 Flocking Model による群集シミュレーション

Flocking Model は Craig Reynolds によって考案された、集団の移動行動を再現するためのモデルである [5]。Flocking Model では歩行者の周囲に neighborhood と呼ばれる一定半径の円形領域が設定され、歩行者は次の 3 つのルールに従って neighborhood の中にいる他の歩行者と集団を作り、協調した歩行をする。

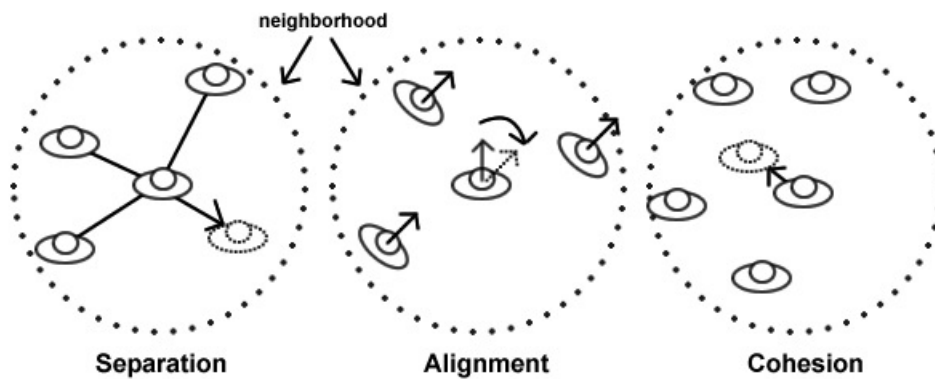


図 2.3: Flocking Model の協調行動

- Separation (図 2.3 左) 歩行者は neighborhood の中にいる他の歩行者から斥力を受ける。
- Alignment (図 2.3 中) 歩行者は neighborhood の中にいる他の歩行者の速度に合わせて自分の進行方向や速度を調節し、集団を維持し続ける。
- Cohesion (図 2.3 右)

歩行者は集団内の他の歩行者と離れすぎないように、neighborhood の中にいる他の歩行者の重心の位置に向かう引力を受ける。

Flocking Model では歩行者は常に周囲の歩行者と歩調をそろえて移動するため、Particle Model では再現できない「同じ方向に向かって歩いている人を前方に見つけたらその人に付いていく」という行動は表現できるが、それ以外の社会的インタラクションに基づく行動は考慮されていない。一例を挙げると、スクランブル交差点のような様々な方向に向かう歩行者が入り交じるような場面

では、歩行者は集団の形成よりも他人との衝突回避を優先し、周囲の状況に応じて遠回りをしたり立ち止まったりするが、Flocking Model では歩行者は常に集団の形成を優先して歩行を続けてしまう。Flocking Model では意図的に集団から離脱する人間の動きは再現できず、一度集団を形成すると歩行者は常に集団に従って歩いてしまうのである。

さらに、Particle Model と Flocking Model に共通した問題点として、斥力による衝突表現では衝突によって歩行者が受ける影響が正しく表現されないことが挙げられる。実際の人間には衝突した際に突き飛ばされやすい方向と突き飛ばされにくい方向がある。一例を挙げると、歩行者同士が正面衝突をする場合、衝突する両者は視界に相手を捕らえて事前に衝突を予測し、衝突に備えることができるため、衝突しても互いに突き飛ばされるということはない。しかし、ある歩行者が他の歩行者に追突される場合、衝突する側の歩行者は正面衝突と同様に衝突を予測できるが、衝突される側の歩行者は突如後ろから押されることになるため、衝突に備えることができず大きく前に突き飛ばされる。既存の斥力による衝突表現では衝突の影響は両者の位置関係だけを元にして決定されるため、このような方向による衝突の影響の違いが正しく表現されない。

以上で述べたように、既存の歩行者モデルには群集歩行の再現に関する問題点が存在する。特に社会心理学的な行動分析や建築物の設計評価のために群集歩行シミュレーションを行う場合、歩行者間の社会的なインタラクションや歩行者同士の衝突は非常に重要な要素であり、これらの要素をより細かく扱うことができる歩行者モデルを考案することは仮想空間シミュレーションの実用性の向上に直接結びつくものと考えられる。

第3章 仮想空間における歩行者モデル

本章では第2章で述べた既存の歩行者モデルの問題点を解消する新しい歩行者モデルを考案する。歩行者モデルの考案にあたって、始めに第2章で挙げた

- 1) 社会的インタラクションに基づく歩行者同士の意図的な接近が表現できない
- 2) 衝突の影響が正しく表現されない

という二つの問題点について、これらを解消する次の二つの解決策を考案し、これを新しい歩行者モデルの中心とした。

1) 歩行者の身体形状の長方形近似

既存のモデルが歩行者同士の意図的な接近を表現できない原因は、歩行者の衝突を斥力によって表現しているため、人体の形状が円形で近似されてしまっていることにあると考えられる。実際の人間の身体は上から見ると前後方向よりも左右方向のほうが長く、円形による近似では体の前後方向に余分な衝突判定が生じてしまう。この結果、「目の前を通ろうとする歩行者に道を譲る」などといった人間同士の意図的な接近状況を表現する際にも余計な斥力が働いてしまい、群集の行動が正しく表現されないことがある。これを解決するために歩行者の身体形状を長方形で近似し、前後方向の余分な衝突判定を解消する歩行者モデルを考案した。

2) 歩行者の進行方向を考慮した衝突計算

実際の衝突では人間同士が正面衝突する場合と追突する場合では衝突時に両者が受ける影響は異なるが、既存のモデルではこのような衝突の方向による影響の違いは考慮されず他人に近づいたら押し返されるという単純なルールを使用しているため、衝突の影響が正しく表現されない。これを解決するために衝突計算の方法を変更し、衝突時の両者の体の向きや位置関係から衝突の種類を判別し、それぞれの種類によって異なった衝突計算を行い、衝突の影響をより細かく扱う衝突モデルを考案した。

以下ではこの二つの解決策の詳細を述べる。

3.1 歩行者の長方形近似

既存のモデルの円形近似による身体形状の表現に代わるものとして、長方形近似による身体形状の表現を考案した。このモデルでは、歩行者の身体の幅と奥行き最大の長さをそれぞれ長辺と短辺とする長方形を歩行者の周りに設定

し、各歩行者の長方形が接触したときに歩行者同士が接触したと判定する。

図 3.1 に Particle Model や Flocking Model で用いられている円形近似と、長方形近似による歩行者の接触判定範囲を示す。点線で表された歩行者に対し、円形近似 (左) と長方形近似 (右) ではそれぞれ実線の範囲に衝突判定が設定されている。円形近似では歩行者の前後に余分な衝突判定が存在し、長方形による近似ではその余分な衝突判定が解消されることがわかる。図 3.2 の左図は、仮想

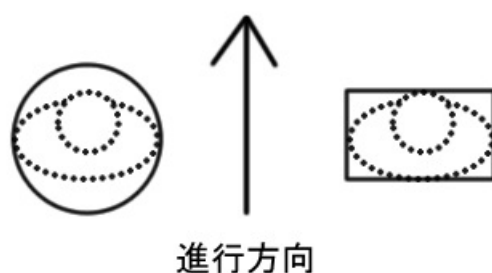


図 3.1: 歩行者の接触判定範囲

空間環境である FreeWalk を用いて円形の接触判定による歩行者の密集状態を表現したものである。また、FreeWalk に本節で述べた長方形近似による接触判定を実装し、同様の密集状態を表現したものが右図である。このような密集状態においては、歩行者同士の間にながら隙間が空いている円形近似に比べて、長方形近似による接触表現では人間同士の密着した状況の自然な表現が可能となっている。

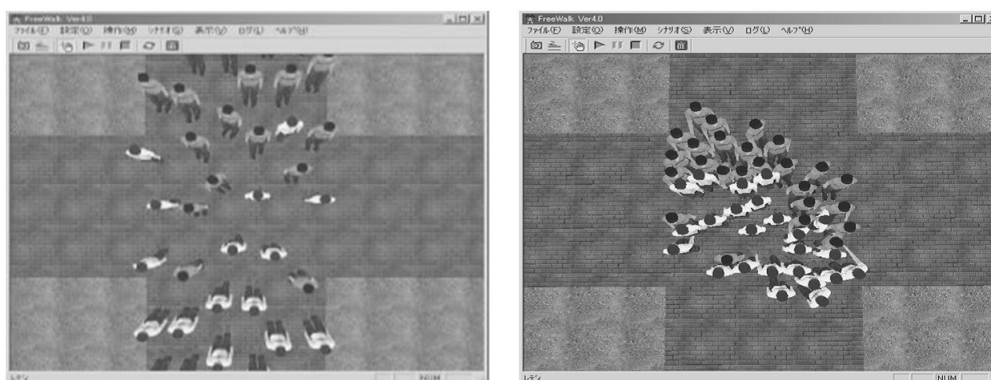


図 3.2: 仮想空間環境 FreeWalk 上での歩行者の密集状態表現

3.2 歩行者間の衝突

3.2.1 衝突の方向による分類

第2章で述べたように，既存のモデルでは衝突によって歩行者が受ける力は衝突した瞬間の二者の位置関係によって決定されるが，実際の衝突では受ける力は二者の衝突時の進行方向によって決まる．また，既存のモデルでは衝突時の両者の体の向きによる衝突の違いが考慮されず，側方からの衝突時と追突時の力の計算に同じ斥力の計算式が利用されている．これらの問題点を解消するため，衝突時の二者の進行方向と体の向きを考慮した衝突計算を考案した．次節ではこの衝突計算の詳細について述べる．

3.2.2 進行方向を考慮した衝突表現

群集歩行内の衝突を再現するために，第5章で後述する群集衝突実験の映像から実際の群集の歩行状況で起こる衝突の種類を調査した．その結果，実際の群集の衝突には大きく分けて次の3種類があることが確認できた（図3.3）．

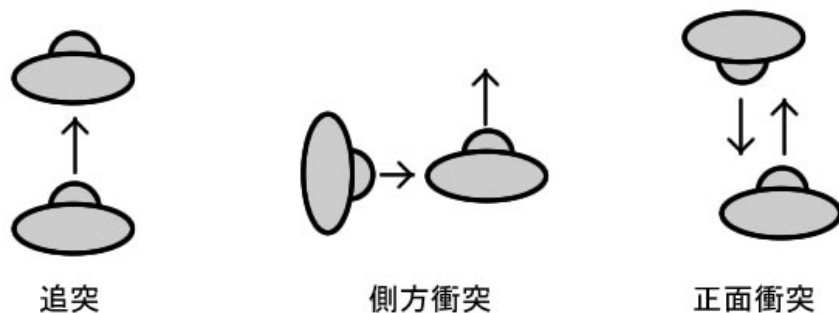


図 3.3: 衝突の種類

1. 追突

ある歩行者に後方から別の歩行者が衝突する．歩行者が列を形成して歩いている時に，前を歩いている歩行者が衝突回避のため立ち止まったり減速したりする場合に起こる．衝突された歩行者は進行方向に押し出され，衝突した歩行者はその場で立ち止まる．

2. 側方衝突

前進中または停止中の歩行者に，側方から別の歩行者が衝突する．列を形成して歩いている歩行者集団の間を異方向からやってきた別の歩行者が強

引に通り返けようとする場合などに起こる。衝突した歩行者は進行方向を変えずに減速しながら歩き続け、衝突された歩行者は衝突した歩行者の進行方向に押されながら歩き続ける。

3. 正面衝突

正反対の方向に歩いている二人の歩行者が正面から衝突する。狭い空間を多数の歩行者が往来している状況で、正対する歩行者同士が相手を回避できなかった場合に起こる。衝突した二人の歩行者はその場で停止した後、平行移動して、相手にぶつからない位置にまで移動した後に再び前進を始める。

以上の3種類の衝突を再現するためには、3.1節で述べた接触判定機能に加えて、衝突の種類を判別する機能と、衝突による歩行者の速度変化(歩く方向と速さの変化)を計算する機能が必要となる。

第4章 歩行者モデルの実装

第3章で述べた歩行者モデルを、仮想空間環境である FreeWalk/Q に実装した。以下ではこの具体的な実装方法について述べる。

4.1 長方形近似モデルの実装

ある一人の歩行者と他の全ての歩行者の接触を長方形近似によって判定する関数 `RectangleCollisionCheck` の擬似コードを図 4.1 に示す。

```
RectangleCollisionCheck(){  GetMyData(); ... 自分の身体サイズや進行方向を取得
  for(自分以外の全ての歩行者について)
    GetWalkersData(); ... 相手の身体サイズや進行方向を取得
    if(WalkersPositionCheck()) ... 相手が衝突が起き得る位置にいるならば
      GetBoudingBox(); ...相手の衝突判定点を計算
      if(CheckCollision()) ...相手の衝突判定点が自分の衝突範囲の中であれば
        return true;
  return false;
}
```

図 4.1: 長方形近似の接触判定関数の擬似コード

標系とは別に，OthersPosition を中心として対象の歩行者の身体の左右方向に X 軸，前後方向に Y 軸を取ったローカル座標系を設定し，ローカル座標系と全体座標系との角度のずれを θ とする．歩行者の左右方向の長さを OthersWidth，前後方向の長さを OthersDepth とすると，各頂点のローカ

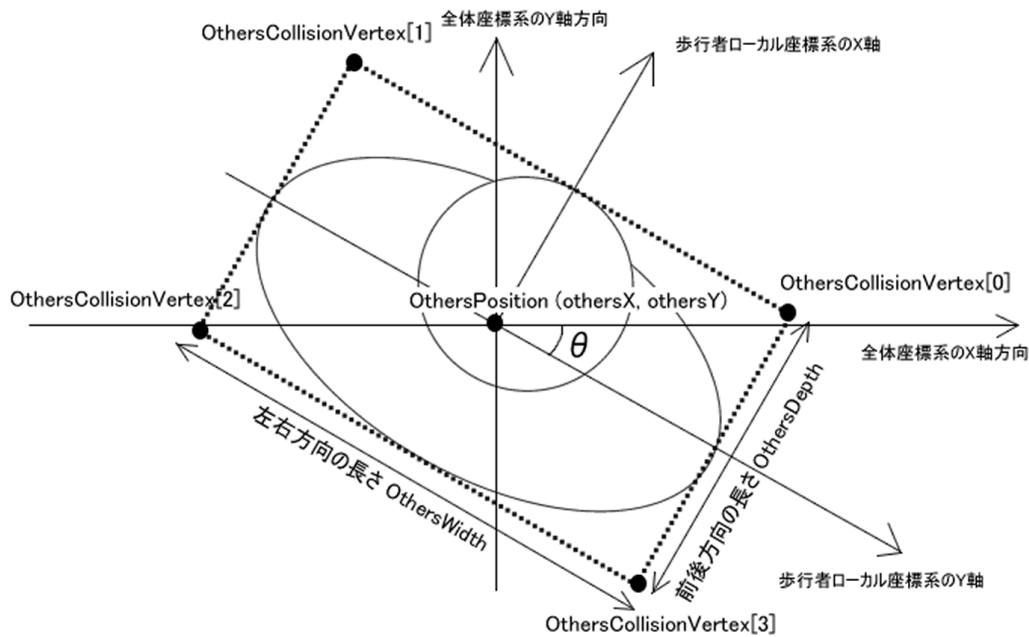


図 4.2: 歩行者の接触範囲の頂点取得計算

ル座標系における座標は $(\pm \text{OthersWidth} \times 1/2, \pm \text{OthersDepth} \times 1/2)$ となる．

この各頂点の座標を全体座標系に変換した座標は， $(\pm \text{OthersWidth} \times 1/2, \pm \text{OthersDepth} \times 1/2)$ を角度 θ だけ回転した座標に，歩行者の全体座標系での位置 OthersPosition を加えたものであるので，以下ようになる．

$$\text{OthersCollisionVertex} = \begin{bmatrix} \text{othersX} \\ \text{othersY} \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \pm \text{OthersWidth} \times \frac{1}{2} \\ \pm \text{OthersDepth} \times \frac{1}{2} \end{bmatrix}$$

- CheckCollision()

この関数では GetBoundingBox() で得た相手歩行者の接触判定の四頂点が，自分の接触判定の長方形に重なっているかどうかを調べる．

図 4.3 に他歩行者との接触判定の詳細を記す．自分のローカル座標系の中心と GetBoundingBox() で得られた相手歩行者の 4 つの衝突判定点の一つをとり，その点を自分のローカル座標系の座標に変換する．変換したローカル座標系の点を (localCollisionVertexX, localCollisionVertexY) とすると，この衝突判定点が自分の接触判定の長方形に重なっている状態は次の式で表される．

$$\left(|localCollisionVertexX| \leq myWidth \right) \\ \&\& \left(|localCollisionVertexY| \leq myDepth \right)$$

相手歩行者の 4 つの衝突判定点全てに対してこの判定を行い，一つでも重なる点が存在すれば自分と相手歩行者は接触することになる．

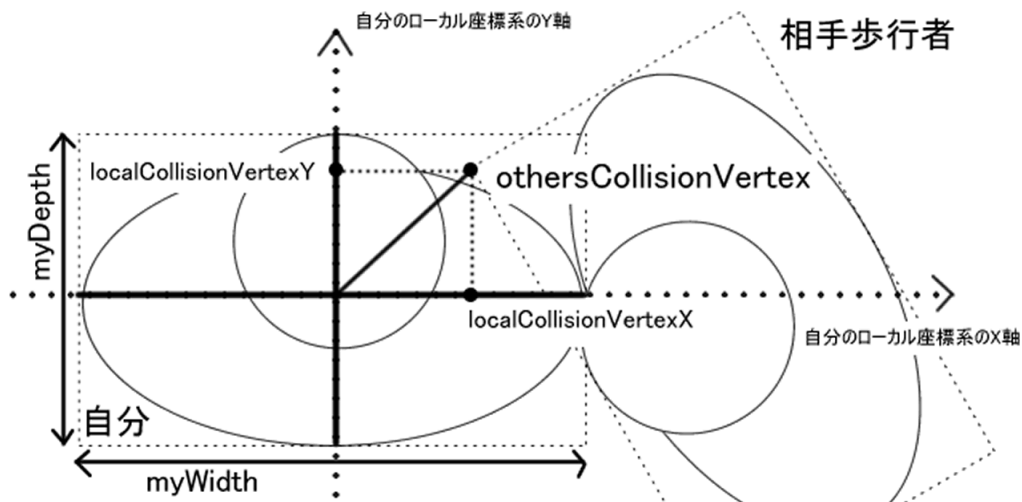


図 4.3: 他歩行者との接触判定

全ての歩行者に対し，以上の getWalkersData() から CheckCollision() までの接触判定の流れを繰り返すことによって，自分に関する全ての接触を検出することができる．

4.1.1 進行方向を考慮した衝突の実装

3.2 節で述べた進行方向を考慮した衝突を実現する実装のコードを図 4.4 に示す，

コード内の各関数ではそれぞれ以下の操作が行われる．

- GetMyVelocity()

```

GetMyVelocity();
for(全ての歩行者について)
    接触を調べる; ...3.1 節の RectangleCollisionCheck() を使用
if(接触するなら)
    GetOthersData(); ...
    GetCollisionType(); ...自分と相手の向きから衝突の種類を判定
ModifyVelocity(); ...衝突により受けた影響を速度に反映
Update(); ...歩行を実行

```

図 4.4: 進行方向を考慮した衝突の擬似コード

この関数では現在位置や目的地の方向から自分の歩く速度 (進行方向と速さ) を決定する。この関数により決定される速度は自分が進もうとしている方向と速さを表すもので、実際の速度は衝突によって変更される場合がある。

- GetOthersData()

接触した相手の進行方向や歩く速さなどのデータを取得する。取得したデータは衝突の種類や衝突する側・される側を決定するのに使われる。

- GetCollisionType()

この関数では次の3つの処理を行う。

1. 衝突の種類を判別
2. 衝突した側・された側の判別
3. 衝突によって受ける影響の決定

以下にそれぞれの処理の詳細を述べる。

1. 衝突の種類を判別

衝突する両者の進行方向から衝突の種類を判別する。図 4.5 のように、衝突した二者の速度ベクトルがなす角度 $VelocityAngle$ をとり、この角度によって衝突を追突・側方衝突・正面衝突のいずれかに分類する。 $VelocityAngle$ が 0 度に近い場合は両者は同じ方向に歩いていると考えられるので追突、180 度に近い場合は両者が正対しているので正面衝突、90 度に近い場合は両者は直角に衝突するので側方衝突であると考えられる。図 4.5 の場合は $VelocityAngle$ は 90 度に近いので両者の衝突は側方衝突であると判別される。

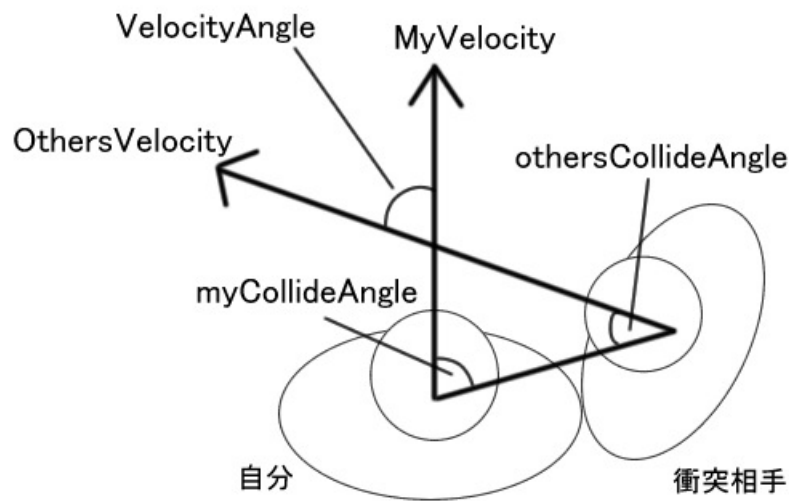


図 4.5: 衝突の種類と衝突した側・された側の判別

2. 衝突した側・された側の判別

正面衝突と側方衝突では，衝突によって自分が受ける影響は，自分が衝突した側であるのか衝突された側であるのかによって大きく異なる．これを判別するために，両者の中心を結んだ直線と，両者の速度ベクトルのなす角度 $myCollideAngle$ と $othersCollideAngle$ をとる (図 4.5)． $myCollideAngle$ は自分の正面方向を 0 度としたときに，衝突相手からの角度に見えるかを表したものであり， $othersCollideAngle$ は逆に相手から自分の見える角度を表すものである．この角度の小さい方が相手をより正面側 (= 進行方向側) に見ていることになるため，この角度の小さい方を衝突した側とする．図 4.5 では $othersCollideAngle < myCollideAngle$ であるので相手が自分に衝突したと判別される．

3. 衝突によって受ける影響の決定

衝突によって歩行者が受ける速度変化を決定する．側方衝突と追突では衝突した側の速度ベクトルの一部が衝突された側に加算され，衝突した側は停止もしくは減速する．正面衝突では両者の速度が 0 となり，両者は相手と衝突しない位置まで平行移動を行う．衝突による速度変化量は次の式で表される．

－ 側方衝突・追突の場合

$$\begin{aligned} \text{(衝突した側の速度変化量)} &= \text{(衝突前の速度ベクトル)} \times \text{(補正係数)} \\ \text{(衝突された側の速度変化量)} &= \text{(衝突した側の速度ベクトル)} \times \text{(補正係数)} \end{aligned}$$

– 正面衝突の場合

$$\text{(衝突した両者の速度変化量)} = \text{(平行移動補正量)}$$

ここで、衝突した側の速度変化量における補正係数とは、衝突後の速度の減少率を表す定数で、-1 から 0 の間の値をとる。衝突された側の速度変化率における補正係数は、衝突する側がされる側に与える力の割合を表す定数で、0 から 1 の間の値をとる。正面衝突の場合の平行移動補正量は正面衝突中の平行移動量を表す値で、これは衝突直前の速度にはほとんど関係がなく、定数で表すことができる。

- ModifyVelocity()/Update() 全ての歩行者に対して衝突判定と速度変化量の計算を行った後に実行される。ModifyVelocity() では GetMyVelocity() によって得られたもともとの速度ベクトルと、衝突による速度の変化量の合計をもとに最終的な速度が決定され、Update() によって歩行が実行される。

第5章 社会的インタラクションを考慮した歩行者シナリオ

第4章で述べた長方形による歩行者近似と方向を考慮した衝突表現機能を含む新しい歩行者モデルを実装した FreeWalk/Q を用いて、実際の群集歩行の再現を行った。以下では再現の手順について述べる。

5.1 群集衝突実験

実際の群集歩行の例として、2004年9月に社会心理学の教授である大阪大学の釘原教授が行った群集衝突実験を取り上げた。この実験は25人の歩行者から構成される二つの歩行群集 A,B が異方向からやってきて幅 5m の道の上で衝突する状況を再現するもので、図 5.1 に示す「交差条件」(左)と「対向条件」(右)の二種類の群集衝突を再現している。また、50人の各被験者には他の人とぶつかりそうになったらできるだけ道を譲ってくださいもしくは他の人とぶつかりそうになってもできるだけ道を譲らないでくださいという二つの教示のどちらかが与えられており、この教示に従って各歩行者は図 5.1 に黒点で示された初期位置からゴールに向かって歩行を行う。さらに各歩行者には実験内容について7つの指示が与えられている。この指示については次節で詳しく述べる。この実験では幅 5m の道という狭い空間上で50人の人間が歩行をするため衝突が起

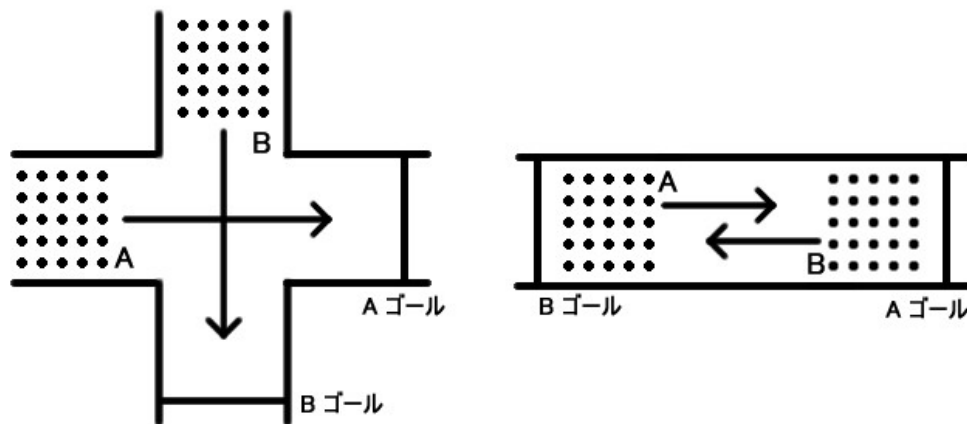


図 5.1: 交差条件 (左) と対向条件 (右)

こりやすく，さらに道を譲る・譲らないという二つの教示を与えることによって，他の歩行者と衝突しそうになったときに立ち止まらずにそのまま進むのか，相手のために道を譲るのかという社会的なインタラクションによる行動パターンの違いが出てくる．既存の歩行者モデルではこのような違いを再現することはできないため，本実験で考案した新しい歩行者モデルを使用することによってこの実験の再現，特に歩行者の社会的インタラクションに基づく行動の再現ができるのであれば，この新しい歩行者モデルは群集歩行の再現に有用であるということができる．

5.2 歩行者シナリオの制約

この群集歩行実験の被験者には表 5.1 に示す 7 つの指示があらかじめ与えられている．

表 5.1 被験者への指示

1. 被験者はスタートの合図で歩き出す。
2. この実験は二つのグループのタイムを競うものであり、全員がより早くゴールしたほうが勝ちである。
3. 決して走ってはならない。
4. 定められた道幅を超えてはならない。
5. 実験中の私語は禁止する。
6. ゴールしてもすぐには立ち止まらず、合図があるまでそのままの勢いで進まなくてはならない。
7. 試行ごとに指定された歩き方 (道を譲る・譲らない) に従って歩くこと。

これらの指示は被験者の歩行の内容に強く影響することが考えられるため、Free-Walk/Q 上でシミュレーションを行う際にもこの存在を考慮する必要がある。これらの被験者制約のうち、被験者のシナリオ作成に反映しにくい4,5,7の3つについてはそれぞれ次の形でシミュレーションに反映させた。

表 5.2 被験者制約の反映

4. 歩行者のシナリオ内で次に踏み出す進路を決める際に、道幅を超える部分には踏み出さないように観測を行う。
5. 各歩行者のシナリオは独立で、他者のシナリオの中身や他者に関する変数を一切参照しないものとする。
7. 各歩行者エージェントには歩き方の指定に対応した異なったシナリオを割り当てる。

5.3 歩行者のシナリオ記述

群集衝突実験の試行を録画した映像について、歩行者の速度・歩行ルート・他人との衝突回避行動などの歩行者の行動を分析し、以下の行動パターンを得た。

- スタートしてから他グループと合流するまでの行動パターン
 - － 先頭の歩行者はまっすぐ前に歩き続ける
 - － 先頭以外の歩行者は前の歩行者を追い抜かず、常に後ろに付いていく。
 - － 先頭以外の歩行者は歩く早さを調整することで前の歩行者との衝突を回避する。
- 他グループと交差している時の行動パターン

- 「道を譲らない」歩行者は前方の状態に関わらず前進を続け、横方向・斜め方向への移動は一切しない。
 - 「道を譲る」歩行者は他の歩行者と接触しそうになった場合には立ち止まる。互いに道を譲り合って1秒ほど膠着状態が続いた場合には歩き出す。
- 他グループと対向している時の行動パターン
 - 「道を譲らない」歩行者は前方の状態に関わらず前進を続け、横方向・斜め方向への移動は一切しない。対向者との衝突が予想される状態でも衝突を避けることはせず、衝突した後に前進に必要な分の最低限の平行移動を行う。
 - 「道を譲る」歩行者は先頭を歩いているときに他の歩行者との衝突が予想される場合、左右のいずれかに一歩進路を変えて衝突を避けようとする。
 - いずれの歩行者も、前方に同じグループの他の歩行者がいる場合にはその歩行者に付いていく。
- 他グループと分離しゴールに向かっていているときの行動パターン
 - 全ての歩行者はゴールに向かってまっすぐ歩き続ける。
 - 前方にいる他者と接触しそうになった場合には速度を落として衝突を回避する。他者を追い抜くことはない。
- 全状態で共通の行動パターン
 - 歩き方の指定に関わらず、全ての歩行者は後ろに下がることはない。前に進めない時には必ず停止する。
 - 前方に他者がいない場合の平均的な歩行速度は「道を譲らない」歩行者 1.85m/s 「道を譲る」歩行者 1.50m/s である。
 - 加速度の影響はほとんど無く、歩行を始めた直後から上記の速度で歩き始める。

分析により得られた以上の行動パターンをもとに歩行者の行動シナリオを作成した。作成したシナリオのうち、道を譲らない歩行者の交差条件・対向条件のシナリオをあらわす状態遷移図をそれぞれ図 5.2，図 5.3 に示す。

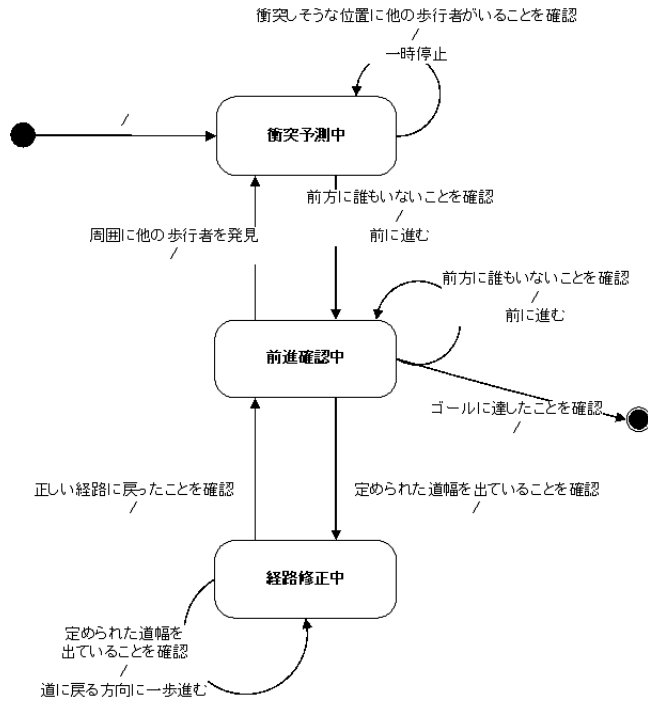


図 5.2: 道を譲る歩行者の交差条件シナリオ

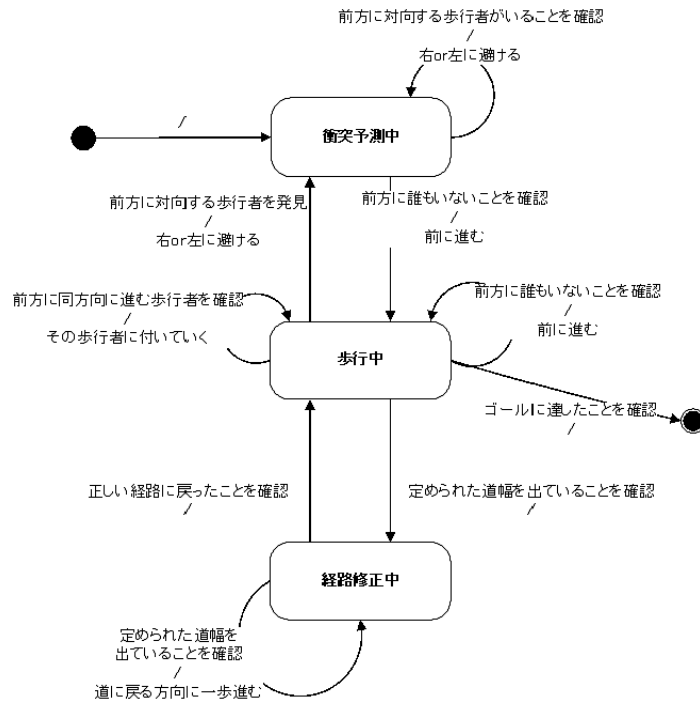


図 5.3: 道を譲る歩行者の対向条件シナリオ

5.4 衝突実験と仮想空間シミュレーションの比較

作成した歩行者のシナリオを使用して群集衝突実験の再現を行った。交差条件・対向条件の各再現状況を図 5.4 に示す。交差条件の再現状況を表す左図からは、図の左側に目の前を通る他歩行者に道を譲っている歩行者がいることがわかる。また、対向条件の再現状況を表す右図からは、右下から左上にかけて列を形成して歩行をしている歩行者集団がいることがわかる。これらの歩行者の行動は自分と衝突しそうな歩行者をみたら立ち止まる、前を歩く歩行者を見たらついていくといった行動ルールによって実現されたものであり、このような歩行者の行動は実際の群集衝突実験の中でも確認されたものである。これは本研究で考案した新しい歩行者モデルによって歩行者同士のインタラクションに基づく行動が再現できている例であり、このことから新しい歩行者モデルは群集内の歩行者の再現に有用であると考えられる。

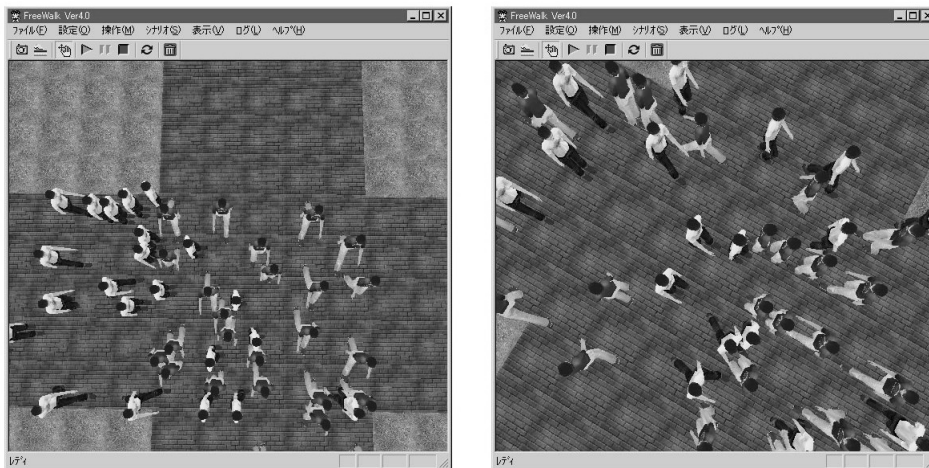


図 5.4: 新しい歩行者モデルによる交差条件 (左) と対向条件 (右) の再現

第6章 おわりに

本研究では、仮想空間における群集歩行シミュレーションについて、既存の歩行者モデルの問題点を解消する新しい歩行者モデルを考案し、このモデルを仮想空間環境 FreeWalk/Q に実装した。歩行者モデルを設計するにあたっては、特に歩行者の社会的インタラクションに基づく行動を表現することを目的とし、次の二点をモデルの中心とした。

1) 長方形近似による歩行者の身体形状の表現

既存の歩行者モデルでは再現できなかった歩行者同士が意図的に接近するという社会的インタラクションに基づく行動を表現するために、人間の身体形状を長方形で近似する接触判定計算を考案した。

2) 接触時の身体の方角を考慮した衝突表現

既存のモデルでは再現されていなかった、正面衝突や追突といった衝突の方角の違いによる衝突の影響の違いを再現する衝突計算を考案した。

この新しい歩行者モデルを実装した FreeWalk/Q を用いて、社会心理学の教授である釘原教授が行った実空間での群集衝突実験の再現を試みた。この実験では狭い空間で大規模な群集が一度に通行するため歩行者同士の衝突が多発する。そのため、この群集の動きを再現するためには衝突を細かく扱うことが必要となる。また、歩行者は衝突を避けるために他者との社会的なインタラクションに基づく行動決定を行っており、これを再現するためには社会的インタラクションの存在を意識した歩行者モデルが必要となる。既存の歩行者モデルではこれらの点が考慮されておらず、そのため、この実験のような群集衝突の状況はうまく再現することができなかった。

群集衝突実験のビデオから歩行者の社会的インタラクションに基づく行動ルールを分析し、それをもとに歩行者の行動シナリオを作成した。このシナリオを用いて群集衝突状況の再現を行ったところ、他人に道を譲る前を歩いている人を見かけたら付いていくといった、既存のモデルでは再現することのできなかった行動を再現することができるようになり、実際の群集歩行状況に近いシミュレーションを行うことができた。この結果から、本研究で考案した歩行者モデルは衝突が多発するような群集の歩行状況の再現に有用であると考えられる。

謝辞

本研究を行うにあたり終始熱心なご指導を賜りました石田亨教授、中西英之助手に深謝いたします。また、実世界での群集衝突実験例を扱うにあたって、大変お世話になりました釘原教授、杉万教授、長弘大樹氏に心より感謝いたします。本研究については石田研究室の先輩方にも様々なご助言をいただきました。この場を借りて御礼申し上げます。

参考文献

- [1] Hideyuki Nakanishi. FreeWalk: A Social Interaction Platform for Group Behavior in a Virtual Space. International Journal of Human Computer Studies, Vol. 60, No. 4, pp. 421-454, 2004.
- [2] Toru Ishida. Q: A Scenario Description Language for Interactive Agents. IEEE Computer, Vol.35, No. 11, pp. 42-47, 2002.
- [3] 岡崎甚幸：建築空間における歩行のためのシミュレーションモデルの研究（その１）磁気モデルの応用による歩行モデル, 日本建築学会論文報告集, 第 283 号, pp.111-119, 1979.
- [4] 岡崎甚幸：建築空間における歩行のためのシミュレーションモデルの研究（その２）混雑した場所での歩行, 日本建築学会論文報告集, 第 284 号, pp.101-109, 1979.
- [5] Craig W. Reynolds : Flocks, Herds, and Schools: A Distributed Behavioral Model, Computer Graphics 21(4) (SIGGRAPH 87 Conference Proceedings) pp. 25-34, 1987.

付録

長方形近似によるフレームごとの衝突判定

```
csBool
fwWalker::collideFrameTime( const fwWalker *myself )
{
    int numWalkers;
    numWalkers = walkerList.getCount();
    if( numWalkers <= 1 ) return FALSE;

    fwWalker *walker, *tempWalker;
    csVec3f myPos, othersPos, v, myDir;
    float r2, len, h, c, myRange, othersRange;
    int i,k;
    csBool hit = FALSE;

#ifdef _FIX_ENABLE_COLLIDE_ //衝突による突き飛ばし判定を使用

    fwHumanWWWalk* ref = (fwHumanWWWalk*)myself ;
    float myWidthRange, myDepthRange, othersWidthRange, othersDepthRange;
    float toWalkerAngle, fromWalkerAngle;
    csVec3f moveDir, pos, tempVec, toWalker, tempMovePos;
    int j;

    pos = myself->getFramePosition();
    moveDir = fwGetDirFromYAngle( CS_DEG2RAD(myself->getFrameAngle()));

    h = myself->getSize()[1];

    if( !myself->getNextFramePosition(myPos) ) return FALSE;

    myWidthRange = myself->getBodyWidth();
    myDepthRange = myself->getBodyDepth();

    myRange = sqrtf((myWidthRange*myWidthRange)+(myDepthRange*myDepthRange)) * 0.5
f;

    if(TRUE){ //常に衝突判定を行う

        for( i=0; i < numWalkers; i++ ) {
            walker = (fwWalker*)walkerList.get(i);
            if( walker == myself || !walker->isVisible() ) continue;

            //othersPos = walker->getStepPosition();
            othersPos = walker->getFramePosition();

            // 人の高さ位置がh高さ以上ならスキップ
            if( fabsf(othersPos[1]-myPos[1]) > h ) continue;

            othersWidthRange = walker->getBodyWidth();
            othersDepthRange = walker->getBodyDepth();
```

```

    othersRange = sqrtf((othersWidthRange*othersWidthRange)+(othersDepthRange*
othersDepthRange)) * 0.5f;

    len = (myRange + othersRange) * FWALKER_COLLIDE_RADIUS_FACTOR;

    v[0] = othersPos[0]-pos[0]; // pos -> othersPos
    v[1] = 0.0f;
    v[2] = othersPos[2]-pos[2];
    c = v.length();

    if( 0.0f <= c && c <= len ) { // 衝突する可能性のない相手は除外する

        csVec3f xp, vert;
        xp.addScaled( pos, c, moveDir );
        vert.sub( othersPos, xp );
        r2 = vert[0]*vert[0]+vert[2]*vert[2];

        csVec3f othersCollideVertex[9];
        csVec3f vertexDir;
        float othersDir, vertexDistance, vertexDegree;

        othersDir = walker->getFrameAngle();

        //人体モデル近似の長方形の4頂点取得
        othersCollideVertex[0].vec[0] = othersPos[0] + cosf(CS_DEG2RAD(othersD
ir)) * othersDepthRange * 0.5f //local 前後方向の補正
            + cosf(CS_DEG2RAD(othersDir + 90.0f)) * others
WidthRange * 0.5f; //local 左右方向の補正
        othersCollideVertex[0].vec[1] = 0.0f;
        othersCollideVertex[0].vec[2] = othersPos[2] + sinf(CS_DEG2RAD(othersD
ir)) * othersDepthRange * 0.5f
            + sinf(CS_DEG2RAD(othersDir + 90.0f)) * others
WidthRange * 0.5f;

        othersCollideVertex[1].vec[0] = othersPos[0] - cosf(CS_DEG2RAD(othersD
ir)) * othersDepthRange * 0.5f
            + cosf(CS_DEG2RAD(othersDir + 90.0f)) * others
WidthRange * 0.5f;
        othersCollideVertex[1].vec[1] = 0.0f;
        othersCollideVertex[1].vec[2] = othersPos[2] - sinf(CS_DEG2RAD(othersD
ir)) * othersDepthRange * 0.5f
            + sinf(CS_DEG2RAD(othersDir + 90.0f)) * others
WidthRange * 0.5f;

        othersCollideVertex[2].vec[0] = othersPos[0] - cosf(CS_DEG2RAD(othersD
ir)) * othersDepthRange * 0.5f
            - cosf(CS_DEG2RAD(othersDir + 90.0f)) * others
WidthRange * 0.5f;
        othersCollideVertex[2].vec[1] = 0.0f;
        othersCollideVertex[2].vec[2] = othersPos[2] - sinf(CS_DEG2RAD(othersD
ir)) * othersDepthRange * 0.5f
            - sinf(CS_DEG2RAD(othersDir + 90.0f)) * others

```

```

WidthRange * 0.5f;

    othersCollideVertex[3].vec[0] = othersPos[0] + cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
                                - cosf(CS_DEG2RAD(othersDir + 90.0f)) * others
WidthRange * 0.5f;
    othersCollideVertex[3].vec[1] = 0.0f;
    othersCollideVertex[3].vec[2] = othersPos[2] + sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
                                - sinf(CS_DEG2RAD(othersDir + 90.0f)) * others
WidthRange * 0.5f;
    othersCollideVertex[4].vec[0] = othersPos[0];
    othersCollideVertex[4].vec[1] = 0.0f;
    othersCollideVertex[4].vec[2] = othersPos[2];

#ifdef _USE_EXACT_COLLISION_CHECK_
    //詳細な衝突判定
    othersCollideVertex[5].vec[0] = othersPos[0] + cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f;
    othersCollideVertex[5].vec[1] = 0.0f;
    othersCollideVertex[5].vec[2] = othersPos[2] + sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f;

    othersCollideVertex[6].vec[0] = othersPos[0] - cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f;
    othersCollideVertex[6].vec[1] = 0.0f;
    othersCollideVertex[6].vec[2] = othersPos[2] - sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f;

    othersCollideVertex[7].vec[0] = othersPos[0] + cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidthRange * 0.5f;
    othersCollideVertex[7].vec[1] = 0.0f;
    othersCollideVertex[7].vec[2] = othersPos[2] + sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidthRange * 0.5f;

    othersCollideVertex[8].vec[0] = othersPos[0] - cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidthRange * 0.5f;
    othersCollideVertex[8].vec[1] = 0.0f;
    othersCollideVertex[8].vec[2] = othersPos[2] - sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidthRange * 0.5f;
#endif

#ifdef _USE_EXACT_COLLISION_CHECK_
    for(j=0; j<9; j++){
#else
    for(j=0; j<5; j++){
#endif
        vertexDir.sub(othersCollideVertex[j], myPos);
        vertexDistance = vertexDir.length();
        vertexDegree = FW_REL_ANGLE(fwGetYAngleFromDir(vertexDir) - fwGetYAngleFromDir(moveDir));

        if(moveDir.dot(vertexDir) <= 0){//後ろにある衝突判定点は衝突計算の必要

```

なし

```
        continue;
    }

    if((fabs(vertexDistance * sinf(CS_DEG2RAD(vertexDegree))) < (myWidthRange * 0.5f))
        && (fabs((vertexDistance * cosf(CS_DEG2RAD(vertexDegree)))) < (myDepthRange * 0.5f))){ //衝突条件

        v.negate(v); // otherPos -> pos
        hit = TRUE;
        break;
    }
}

if(hit){

    this->setCollisionNow(TRUE);
    walker->setCollisionNow(TRUE);

    //衝突の種類と衝突する側・される側の決定
    toWalker.sub(walker->getFramePosition(), this->getFramePosition());
    toWalkerAngle = FW_REL_ANGLE(fwGetYAngleFromDir(toWalker));
    fromWalkerAngle = FW_REL_ANGLE(toWalkerAngle + 180.0f);

    if((fabs(FW_REL_ANGLE(this->getFrameAngle()-othersDir)) >= 20.0f)
        && (fabs(FW_REL_ANGLE(this->getFrameAngle()-othersDir)) <= 160.0f)){ //側方衝突

        this->setBackwardCollide(FALSE);
        this->setForwardCollide(FALSE);
        this->setSideCollide(TRUE);
        walker->setBackwardCollide(FALSE);
        walker->setForwardCollide(FALSE);
        walker->setSideCollide(TRUE);
        if(walker->getCollidedWalkerID() == this->getID()){ //自分は「衝突する側」

            walker->setCollidedNow(TRUE);
            this->setCollideNow(TRUE);

            tempVec = this->getStepMoveDirection();
            tempMovePos.addScaled(walker->getCurFrameMovePosByCollision(), 0.5f, tempVec);
            walker->setCurFrameMovePosByCollision(tempMovePos);
        } else if(this->getCollidedWalkerID() == walker->getID()){ //自分は「衝突される側」

            this->setCollidedNow(TRUE);
            walker->setCollideNow(TRUE);

            tempVec = walker->getStepMoveDirection();
            tempMovePos.addScaled(this->getCurFrameMovePosByCollision(), 0.5f, tempVec);
            this->setCurFrameMovePosByCollision(tempMovePos);
```

```

    } else { //前回とは違う相手と衝突した場合
        if(fabs(FW_REL_ANGLE(toWalkerAngle - this->getFrameAngle())) <=
(fabs(FW_REL_ANGLE(fromWalkerAngle - othersDir)))){ //自分は「衝突する側」
            walker->setCollidedNow(TRUE);
            walker->setCollidedWalkerID(this->getID());
            this->setCollideNow(TRUE);

            tempVec = this->getStepMoveDirection();
            tempMovePos.addScaled(walker->getCurFrameMovePosByCollision(),
0.5f, tempVec);
            walker->setCurFrameMovePosByCollision(tempMovePos);
        } else { //自分は「衝突される側」
            this->setCollidedNow(TRUE);
            this->setCollidedWalkerID(walker->getID());
            walker->setCollideNow(TRUE);

            tempVec = walker->getStepMoveDirection();
            tempMovePos.addScaled(this->getCurFrameMovePosByCollision(), 0.
5f, tempVec);
            this->setCurFrameMovePosByCollision(tempMovePos);
        }
    }
} else if(fabs(FW_REL_ANGLE(this->getFrameAngle()-othersDir)) <= 20.
0f){ //追突
    this->setBackwardCollide(TRUE);
    this->setForwardCollide(FALSE);
    this->setSideCollide(FALSE);
    walker->setBackwardCollide(TRUE);
    walker->setForwardCollide(FALSE);
    walker->setSideCollide(FALSE);
    //この条件下では常に追突する側
    walker->setCollidedNow(TRUE);
    this->setCollideNow(TRUE);
    tempVec = this->getStepMoveDirection();
    tempMovePos.addScaled(walker->getCurFrameMovePosByCollision(), 0.5
f, tempVec);
    walker->setCurFrameMovePosByCollision(tempMovePos);
} else { //正面衝突
    this->setBackwardCollide(FALSE);
    this->setForwardCollide(TRUE);
    this->setSideCollide(FALSE);
    walker->setBackwardCollide(FALSE);
    walker->setForwardCollide(TRUE);
    walker->setSideCollide(FALSE);
    //一時的に衝突する側・される側を設定
    walker->setCollidedNow(TRUE);
    this->setCollideNow(TRUE);
    tempVec = this->getStepMoveDirection();
    tempMovePos.addScaled(walker->getCurFrameMovePosByCollision(), 0.5
f, tempVec);
    walker->setCurFrameMovePosByCollision(tempMovePos);
}
}

```

```

//相手との角度によって衝突する側・される側を決定
toWalker.sub(walker->getFramePosition(), this->getFramePosition());
toWalkerAngle = FW_REL_ANGLE(fwGetYAngleFromDir(toWalker));
fromWalkerAngle = FW_REL_ANGLE(toWalkerAngle + 180.0f);

    if(fabs(FW_REL_ANGLE(toWalkerAngle - this->getFrameAngle())) <= (fab
s(FW_REL_ANGLE(fromWalkerAngle - othersDir)))){ //自分は「衝突する側」
        this->setCollidedNow(FALSE);
        walker->setCollidedNow(TRUE);

        tempVec = this->getStepMoveDirection();

        tempMovePos.addScaled(walker->getCurFrameMovePosByCollision(), 0.5
f, tempVec);

        walker->setCurFrameMovePosByCollision(tempMovePos);
    } else { //自分は「衝突される側」
        this->setCollidedNow(TRUE);
        walker->setCollidedNow(FALSE);

        tempVec = walker->getStepMoveDirection();

        tempMovePos.addScaled(this->getCurFrameMovePosByCollision(), 0.5f,
tempVec);
        this->setCurFrameMovePosByCollision(tempMovePos);

        //全歩行者に対して衝突判定を行う
    }
} //if(hit)
}
}
for( k = 0; k < numWalkers; k++ ) {
    tempWalker = (fwWalker*)walkerList.get(k);
    if(tempWalker->getID() == this->getCollideWalkerID()){
        tempWalker->setCollideWalkerID(-1);
        break;
    }
}
this->setCollideNow(FALSE);
this->setCollideWalkerID(-1);
}
}

return hit;
}
\end{document}

```

他歩行者との衝突予測アルゴリズム

```

csBool
fwWalker::collide( const fwWalker *myself,
    const csVec3f &pos, const csVec3f &moveDir,

```

```

        float stride, float absDegree,
        csVec3f &collideWalkerDir, csVec3f &collideWalkerPoint,
        float &collideWalkerDist )
{

    int numWalkers;
    numWalkers = walkerList.getCount();
    if( numWalkers <= 1 ) return FALSE;

    fwHumanWWWalk* ref = (fwHumanWWWalk*)myself ;

    fwWalker *walker;
    csVec3f othersPos, othersNextPos, v, myPos;
    float r2, myRange, h, othersRange;
    float myWidthRange, myDepthRange, othersWidthRange, othersDepthRange;
    float c, len;
    int i,j;
    csBool hit=FALSE;

    stride *= FWWALKER_COLLIDE_STRIDE_FACTOR ;

    collideWalkerDist = FLT_MAX;
    h = myself->getSize()[1];

    myPos = pos ;

    myWidthRange = myself->getBodyWidth();
    myDepthRange = myself->getBodyDepth();

    myRange = sqrtf((myWidthRange*myWidthRange)+(myDepthRange*myDepthRange)) * 0.5
f;

    for( i=0; i < numWalkers; i++ ) {
        walker = (fwWalker*)walkerList.get(i);
        if( walker == myself || !walker->isVisible() ) continue;

        othersPos = walker->getFramePosition();

        // 人の高さ位置がh高さ以上ならスキップ
        if( fabsf(othersPos[1]-myPos[1]) > h ) continue;

        csVec3f othersTailPos ; //後ろ足の位置
        othersTailPos.addScaled(othersPos, (-1.0f * walker->getStepStride()), walker
->getStepMoveDirection());

        csVec3f myFrontPos ; //1ステップ先の自分の足先

        // 内輪差を考慮
        //
        ref->nextStepPosition(myPos, moveDir, stride, myFrontPos) ;

        othersWidthRange = walker->getBodyWidth();

```

```

othersDepthRange = walker->getBodyDepth();

//othersRange = walker->getRadius() ;
othersRange = sqrtf((othersWidthRange*othersWidthRange)+(othersDepthRange*ot
hersDepthRange)) * 0.5f;

len = (myRange + othersRange) * FWWALKER_COLLIDE_RADIUS_FACTOR;

v[0] = othersPos[0]-pos[0]; // pos -> othersPos
v[1] = 0.0f;
v[2] = othersPos[2]-pos[2];
c = v.length();

// 移動線上に相手がいるかどうか
if( 0.0f <= c && c <= stride+len ) { // 後ろ半分は除外する
    csVec3f xp, vert;
    xp.addScaled( pos, c, moveDir );
    vert.sub( othersPos, xp );
    r2 = vert[0]*vert[0]+vert[2]*vert[2];

#if 0
    //同じ方向をむいている歩行者に対して衝突予測を行わない
    if(((fabs(FW_REL_ANGLE(this->getStepAngle() - walker->getStepAngle())) < 4
5.0f)){
        continue;
    }
#endif

    csVec3f othersCollideVertex[4];
    csVec3f vertexDir;
    float othersDir, vertexDistance, vertexDegree;

    //othersDir = walker->getStepAngle();
    othersDir = fwGetYAngleFromDir(walker->getStepMoveDirection());

    //人体モデル近似の長方形の4頂点取得
    othersCollideVertex[0].vec[0] = othersPos[0] + cosf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
        + cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    othersCollideVertex[0].vec[1] = 0.0f;
    othersCollideVertex[0].vec[2] = othersPos[2] + sinf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
        + sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    othersCollideVertex[1].vec[0] = othersPos[0] - cosf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
        + cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    othersCollideVertex[1].vec[1] = 0.0f;
    othersCollideVertex[1].vec[2] = othersPos[2] - sinf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
        + sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt

```

```

hRange * 0.5f;
    othersCollideVertex[2].vec[0] = othersPos[0] - cosf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
        - cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    othersCollideVertex[2].vec[1] = 0.0f;
    othersCollideVertex[2].vec[2] = othersPos[2] - sinf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
        - sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    othersCollideVertex[3].vec[0] = othersPos[0] + cosf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
        - cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    othersCollideVertex[3].vec[1] = 0.0f;
    othersCollideVertex[3].vec[2] = othersPos[2] + sinf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
        - sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    for(j=0; j<4; j++){
        vertexDir.sub(othersCollideVertex[j], myPos);
        vertexDistance = vertexDir.length();
        vertexDegree = FW_REL_ANGLE(fwGetYAngleFromDir(vertexDir) - fwGetYAngleFromDir(moveDir));

        if(moveDir.dot(vertexDir) <= 0){//後ろにいる歩行者は衝突計算の必要なし
            continue;
        }

        //2004.10.31 大石 体が重なったまま動けなくなる deadlock の解消
        if((fabs((vertexDistance * sinf(CS_DEG2RAD(vertexDegree)))) < (myWidthRange * 0.5f))
            && (fabs((vertexDistance * cosf(CS_DEG2RAD(vertexDegree)))) < (stride * 0.5f))){

            v.negate(v); // otherPos -> pos
            hit = TRUE;

            break;
        }
    } //現在位置による衝突判定終了

    //相手の一歩先に対する衝突判定

#if 0
    //同じ方向をむいている歩行者に対して衝突予測をしない
    if((fabs(FW_REL_ANGLE(this->getStepAngle() - walker->getStepAngle())) < 45.0f){
        continue;
    }
#endif

    othersNextPos.addScaled(walker->getFramePosition(), walker->getStepStride

```

```

(), walker->getStepMoveDirection());
    //othersDir = walker->getStepAngle();
    othersDir = fwGetYAngleFromDir(walker->getStepMoveDirection());

    //人体モデル近似の長方形の4頂点取得
    othersCollideVertex[0].vec[0] = othersNextPos[0] + cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
        + cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
        hRange * 0.5f;
    othersCollideVertex[0].vec[1] = 0.0f;
    othersCollideVertex[0].vec[2] = othersNextPos[2] + sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
        + sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
        hRange * 0.5f;
    othersCollideVertex[1].vec[0] = othersNextPos[0] - cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
        + cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
        hRange * 0.5f;
    othersCollideVertex[1].vec[1] = 0.0f;
    othersCollideVertex[1].vec[2] = othersNextPos[2] - sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
        + sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
        hRange * 0.5f;
    othersCollideVertex[2].vec[0] = othersNextPos[0] - cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
        - cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
        hRange * 0.5f;
    othersCollideVertex[2].vec[1] = 0.0f;
    othersCollideVertex[2].vec[2] = othersNextPos[2] - sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
        - sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
        hRange * 0.5f;
    othersCollideVertex[3].vec[0] = othersNextPos[0] + cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
        - cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
        hRange * 0.5f;
    othersCollideVertex[3].vec[1] = 0.0f;
    othersCollideVertex[3].vec[2] = othersNextPos[2] + sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
        - sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
        hRange * 0.5f;
    for(j=0; j<4; j++){
        vertexDir.sub(othersCollideVertex[j], myPos);
        vertexDistance = vertexDir.length();
        vertexDegree = FW_REL_ANGLE(fwGetYAngleFromDir(vertexDir) - fwGetYAngleFromDir(moveDir));

        if(moveDir.dot(vertexDir) <= 0){//後ろにいる歩行者は衝突計算の必要なし
            continue;
        }

        if((fabs((vertexDistance * sinf(CS_DEG2RAD(vertexDegree)))) < (myWidthRange * 0.5f))

```

```

        && (fabs((vertexDistance * cosf(CS_DEG2RAD(vertexDegree)))) > (myDepth
Range * 0.5f))
        && (fabs((vertexDistance * cosf(CS_DEG2RAD(vertexDegree)))) < (stride
* 0.5f)){
            v.negate(v); // otherPos -> pos
            hit = TRUE;

            break;
        }
    } //相手の一歩先に対する衝突判定 ここまで

}

#if 1 //足の衝突判定
if(!hit) {
    csVec3f othersmoveDir(walker->getStepMoveDirection());

    if( 0.0f <= c && c < stride + len) {
        csVec3f xp, vert;
        xp.addScaled( othersTailPos, c, othersmoveDir );
        vert.sub( myFrontPos, xp );
        r2 = vert[0]*vert[0]+vert[2]*vert[2];

        csVec3f legCollideCheck[4];
        float legCollideCheckDeg[4];

        legCollideCheck[0].vec[0] = othersTailPos[0] - myFrontPos[0];
        legCollideCheck[0].vec[1] = 0;
        legCollideCheck[0].vec[2] = othersTailPos[2] - myFrontPos[2];
        legCollideCheck[1].vec[0] = othersPos[0] - myFrontPos[0];
        legCollideCheck[1].vec[1] = 0;
        legCollideCheck[1].vec[2] = othersPos[2] - myFrontPos[2];
        legCollideCheck[2].vec[0] = myFrontPos[0] - othersPos[0];
        legCollideCheck[2].vec[1] = 0;
        legCollideCheck[2].vec[2] = myFrontPos[2] - othersPos[2];
        legCollideCheck[3].vec[0] = myPos[0] - othersPos[0];
        legCollideCheck[3].vec[1] = 0;
        legCollideCheck[3].vec[2] = myPos[2] - othersPos[2];

        legCollideCheckDeg[0] = fwGetYAngleFromDir(legCollideCheck[0]) - fwGetYAn
ngleFromDir(moveDir);
        legCollideCheckDeg[1] = fwGetYAngleFromDir(legCollideCheck[1]) - fwGetYAn
ngleFromDir(moveDir);
        legCollideCheckDeg[2] = fwGetYAngleFromDir(legCollideCheck[2]) - fwGetYAn
ngleFromDir(othersmoveDir);
        legCollideCheckDeg[3] = fwGetYAngleFromDir(legCollideCheck[3]) - fwGetYAn
ngleFromDir(othersmoveDir);

        if( ((FW_REL_ANGLE(legCollideCheckDeg[0]) * FW_REL_ANGLE(legCollideCheck
Deg[1])) < 0) //足の交差条件
            && ((FW_REL_ANGLE(legCollideCheckDeg[2]) * FW_REL_ANGLE(legCollideChec
kDeg[3])) < 0)){

```

```

v.negate(v); // otherPos -> pos
hit = TRUE;

walker->othersCollideMoveDir = this->getStepMoveDirection();
walker->othersCollideStride = this->getStepStride();

break;

} else {
//others 前足との衝突判定
csVec3f othersFrontPos;
fwHumanWWalk* otherRef = (fwHumanWWalk*)walker ;
ref->nextStepPosition(othersPos, othersmoveDir, walker->getStepStride
(), othersFrontPos);

legCollideCheck[0].vec[0] = othersFrontPos[0] - myPos[0];
legCollideCheck[0].vec[1] = 0;
legCollideCheck[0].vec[2] = othersFrontPos[2] - myPos[2];
legCollideCheck[1].vec[0] = othersPos[0] - myPos[0];
legCollideCheck[1].vec[1] = 0;
legCollideCheck[1].vec[2] = othersPos[2] - myPos[2];
legCollideCheck[2].vec[0] = myFrontPos[0] - othersPos[0];
legCollideCheck[2].vec[1] = 0;
legCollideCheck[2].vec[2] = myFrontPos[2] - othersPos[2];
legCollideCheck[3].vec[0] = myPos[0] - othersPos[0];
legCollideCheck[3].vec[1] = 0;
legCollideCheck[3].vec[2] = myPos[2] - othersPos[2];

legCollideCheckDeg[0] = fwGetYAngleFromDir(legCollideCheck[0]) - fwGet
YAngleFromDir(moveDir);
legCollideCheckDeg[1] = fwGetYAngleFromDir(legCollideCheck[1]) - fwGet
YAngleFromDir(moveDir);
legCollideCheckDeg[2] = fwGetYAngleFromDir(legCollideCheck[2]) - fwGet
YAngleFromDir(othersmoveDir);
legCollideCheckDeg[3] = fwGetYAngleFromDir(legCollideCheck[3]) - fwGet
YAngleFromDir(othersmoveDir);

if( ((FW_REL_ANGLE(legCollideCheckDeg[0]) * FW_REL_ANGLE(legCollideChe
ckDeg[1])) < 0) //足の交差条件
    && ((FW_REL_ANGLE(legCollideCheckDeg[2]) * FW_REL_ANGLE(legCollideCh
eckDeg[3])) < 0)){

v.negate(v); // otherPos -> pos
hit = TRUE;

break;
}
continue ;
}
} else {
continue ;
}
}

```

```

}

if(!hit) {
    csVec3f othersmoveDir(walker->getStepMoveDirection());
    csVec3f vv ;
    //
    // 相手の後ろ足と自分の行き先にもぐりこめるか?
    //
    vv[0] = myFrontPos[0] - othersTailPos[0] ; // others 後ろ足    my 前足(予
定)
    vv[1] = 0.0f;
    vv[2] = myFrontPos[2] - othersTailPos[2] ;
    c = othersmoveDir.dot(vv) ;

    if( 0.0f <= c && c < len) {
        csVec3f xp, vert;
        xp.addScaled( othersTailPos, c, othersmoveDir );
        vert.sub( myFrontPos, xp );
        r2 = vert[0]*vert[0]+vert[2]*vert[2];

        if( r2 < len*len ) {
            v.negate(v); // otherPos -> pos
            hit = TRUE;
        } else {
            continue ;
        }
    } else {
        continue ;
    }
}

if( r2 < collideWalkerDist ) {
    collideWalkerDist = r2;
    collideWalkerPoint = othersPos;
    collideWalkerDir = v;
}
if(hit) { break ; }
}

if( hit ) {
    collideWalkerDist = sqrtf(collideWalkerDist);
    collideWalkerDir.normalize();
}

return hit;
}

```

道を譲らない歩行者の衝突判定計算

```

csBool
fwWalker::strongWalkerCollide( const fwWalker *myself,
    const csVec3f &pos, const csVec3f &moveDir,
    float stride, float absDegree,

```

```

        csVec3f &collideWalkerDir, csVec3f &collideWalkerPoint,
        float &collideWalkerDist )
{

    int numWalkers;
    numWalkers = walkerList.getCount();
    if( numWalkers <= 1 ) return FALSE;

    fwHumanWWWalk* ref = (fwHumanWWWalk*)myself ;

    fwWalker *walker;
    csVec3f othersPos, othersNextPos, v, myPos;
    float r2, myRange, h, othersRange;
    float myWidthRange, myDepthRange, othersWidthRange, othersDepthRange;
    float c, len;
    int i,j;
    csBool hit=FALSE;

    stride *= FWWALKER_COLLIDE_STRIDE_FACTOR ;

    collideWalkerDist = FLT_MAX;
    h = myself->getSize()[1];

    myPos = pos ;

    myWidthRange = myself->getBodyWidth();
    myDepthRange = myself->getBodyDepth();

    myRange = sqrtf((myWidthRange*myWidthRange)+(myDepthRange*myDepthRange)) * 0.5
f;

    for( i=0; i < numWalkers; i++ ) {
        walker = (fwWalker*)walkerList.get(i);
        if( walker == myself || !walker->isVisible() ) continue;

        othersPos = walker->getFramePosition();

        if( fabsf(othersPos[1]-myPos[1]) > h ) continue;

        csVec3f othersTailPos ; //後ろ足の位置
        othersTailPos.addScaled(othersPos, (-1.0f * walker->getStepStride()), walker
->getStepMoveDirection());

        csVec3f myFrontPos ; //1ステップ先の自分の足先

        // 内輪差を考慮
        //
        ref->nextStepPosition(myPos, moveDir, stride, myFrontPos) ;

        othersWidthRange = walker->getBodyWidth();
        othersDepthRange = walker->getBodyDepth();

        //othersRange = walker->getRadius() ;

```

```

    othersRange = sqrtf((othersWidthRange*othersWidthRange)+(othersDepthRange*ot
hersDepthRange)) * 0.5f;

    len = (myRange + othersRange) * FWWALKER_COLLIDE_RADIUS_FACTOR;

    v[0] = othersPos[0]-pos[0]; // pos -> othersPos
    v[1] = 0.0f;
    v[2] = othersPos[2]-pos[2];
    //c = moveDir.dot(v);
    c = v.length();

    // 移動線上に相手がいるかどうか
    if( 0.0f <= c && c <= stride+len ) { // 後ろ半分は除外する
        csVec3f xp, vert;
        xp.addScaled( pos, c, moveDir );
        vert.sub( othersPos, xp );
        r2 = vert[0]*vert[0]+vert[2]*vert[2];

    #if 1
        //同じ方向をむいている歩行者に対しては衝突予測をしない
        if((fabs(FW_REL_ANGLE(this->getStepAngle() - walker->getStepAngle())) < 4
5.0f)){
            continue;
        }
    #endif

        csVec3f othersCollideVertex[4];
        csVec3f vertexDir;
        float othersDir, vertexDistance, vertexDegree;

        //othersDir = walker->getStepAngle();
        othersDir = fwGetYAngleFromDir(walker->getStepMoveDirection());

        //人体モデル近似の長方形の4頂点取得
        othersCollideVertex[0].vec[0] = othersPos[0] + cosf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
            + cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
        othersCollideVertex[0].vec[1] = 0.0f;
        othersCollideVertex[0].vec[2] = othersPos[2] + sinf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
            + sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
        othersCollideVertex[1].vec[0] = othersPos[0] - cosf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
            + cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
        othersCollideVertex[1].vec[1] = 0.0f;
        othersCollideVertex[1].vec[2] = othersPos[2] - sinf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
            + sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
        othersCollideVertex[2].vec[0] = othersPos[0] - cosf(CS_DEG2RAD(othersDir))

```

```

* othersDepthRange * 0.5f
    - cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    othersCollideVertex[2].vec[1] = 0.0f;
    othersCollideVertex[2].vec[2] = othersPos[2] - sinf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
    - sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    othersCollideVertex[3].vec[0] = othersPos[0] + cosf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
    - cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    othersCollideVertex[3].vec[1] = 0.0f;
    othersCollideVertex[3].vec[2] = othersPos[2] + sinf(CS_DEG2RAD(othersDir))
* othersDepthRange * 0.5f
    - sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidt
hRange * 0.5f;
    for(j=0; j<4; j++){
        vertexDir.sub(othersCollideVertex[j], myPos);
        vertexDistance = vertexDir.length();
        vertexDegree = FW_REL_ANGLE(fwGetYAngleFromDir(vertexDir) - fwGetYAngleF
romDir(moveDir));

        if(moveDir.dot(vertexDir) <= 0){//後ろにいる歩行者は衝突計算の必要なし
            continue;
        }

        if((fabs((vertexDistance * sinf(CS_DEG2RAD(vertexDegree)))) < (myWidthRa
nge * 0.5f))
            && (fabs((vertexDistance * cosf(CS_DEG2RAD(vertexDegree)))) < (stride
* 0.5f + 0.3f))){

            v.negate(v); // otherPos -> pos
            hit = TRUE;

            break;
        }
    } //現在位置による衝突判定終了

#if 1
    //同じ方向をむいている歩行者は衝突予測の必要なし
    if((fabs(FW_REL_ANGLE(this->getStepAngle() - walker->getStepAngle())) < 4
5.0f)){
        continue;
    }
#endif

    othersNextPos.addScaled(walker->getFramePosition(), walker->getStepStride
(), walker->getStepMoveDirection());
    //othersDir = walker->getStepAngle();
    othersDir = fwGetYAngleFromDir(walker->getStepMoveDirection());

    //人体モデル近似の長方形の4頂点取得

```

```

        othersCollideVertex[0].vec[0] = othersNextPos[0] + cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
            + cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
hRange * 0.5f;
        othersCollideVertex[0].vec[1] = 0.0f;
        othersCollideVertex[0].vec[2] = othersNextPos[2] + sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
            + sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
hRange * 0.5f;
        othersCollideVertex[1].vec[0] = othersNextPos[0] - cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
            + cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
hRange * 0.5f;
        othersCollideVertex[1].vec[1] = 0.0f;
        othersCollideVertex[1].vec[2] = othersNextPos[2] - sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
            + sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
hRange * 0.5f;
        othersCollideVertex[2].vec[0] = othersNextPos[0] - cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
            - cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
hRange * 0.5f;
        othersCollideVertex[2].vec[1] = 0.0f;
        othersCollideVertex[2].vec[2] = othersNextPos[2] - sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
            - sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
hRange * 0.5f;
        othersCollideVertex[3].vec[0] = othersNextPos[0] + cosf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
            - cosf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
hRange * 0.5f;
        othersCollideVertex[3].vec[1] = 0.0f;
        othersCollideVertex[3].vec[2] = othersNextPos[2] + sinf(CS_DEG2RAD(othersDir)) * othersDepthRange * 0.5f
            - sinf(CS_DEG2RAD(othersDir + 90.0f)) * othersWidth
hRange * 0.5f;
        for(j=0; j<4; j++){
            vertexDir.sub(othersCollideVertex[j], myPos);
            vertexDistance = vertexDir.length();
            vertexDegree = FW_REL_ANGLE(fwGetYAngleFromDir(vertexDir) - fwGetYAngleFromDir(moveDir));

            if(moveDir.dot(vertexDir) <= 0){//後ろにいる歩行者は衝突計算の必要なし
                continue;
            }

            //2004.10.31 大石 体が重なったまま動けなくなる deadlock の解消
            //衝突条件を緩和したので不具合が出る恐れあり
            if((fabs((vertexDistance * sinf(CS_DEG2RAD(vertexDegree)))) < (myWidthRange * 0.5f))
                && (fabs((vertexDistance * cosf(CS_DEG2RAD(vertexDegree)))) < (stride * 0.5f + 0.3f))){

```

```

        v.negate(v); // otherPos -> pos
        hit = TRUE;

        break;
    }
} //相手の一歩先に対する衝突判定 ここまで

}

if(!hit) {
    csVec3f othersmoveDir(walker->getStepMoveDirection());

    if( 0.0f <= c && c < stride + len) {
        csVec3f xp, vert;
        xp.addScaled( othersTailPos, c, othersmoveDir );
        vert.sub( myFrontPos, xp );
        r2 = vert[0]*vert[0]+vert[2]*vert[2];

        csVec3f legCollideCheck[4];
        float legCollideCheckDeg[4];

        legCollideCheck[0].vec[0] = othersTailPos[0] - myFrontPos[0];
        legCollideCheck[0].vec[1] = 0;
        legCollideCheck[0].vec[2] = othersTailPos[2] - myFrontPos[2];
        legCollideCheck[1].vec[0] = othersPos[0] - myFrontPos[0];
        legCollideCheck[1].vec[1] = 0;
        legCollideCheck[1].vec[2] = othersPos[2] - myFrontPos[2];
        legCollideCheck[2].vec[0] = myFrontPos[0] - othersPos[0];
        legCollideCheck[2].vec[1] = 0;
        legCollideCheck[2].vec[2] = myFrontPos[2] - othersPos[2];
        legCollideCheck[3].vec[0] = myPos[0] - othersPos[0];
        legCollideCheck[3].vec[1] = 0;
        legCollideCheck[3].vec[2] = myPos[2] - othersPos[2];

        legCollideCheckDeg[0] = fwGetYAngleFromDir(legCollideCheck[0]) - fwGetYAngleFromDir(moveDir);
        legCollideCheckDeg[1] = fwGetYAngleFromDir(legCollideCheck[1]) - fwGetYAngleFromDir(moveDir);
        legCollideCheckDeg[2] = fwGetYAngleFromDir(legCollideCheck[2]) - fwGetYAngleFromDir(othersmoveDir);
        legCollideCheckDeg[3] = fwGetYAngleFromDir(legCollideCheck[3]) - fwGetYAngleFromDir(othersmoveDir);

        if( ((FW_REL_ANGLE(legCollideCheckDeg[0]) * FW_REL_ANGLE(legCollideCheckDeg[1])) < 0) //足の交差条件
            && ((FW_REL_ANGLE(legCollideCheckDeg[2]) * FW_REL_ANGLE(legCollideCheckDeg[3])) < 0)){

            v.negate(v); // otherPos -> pos
            hit = TRUE;

            break;
        }
    }
}

```

```

} else {
    //others 前足との衝突判定
    csVec3f othersFrontPos;
    fwHumanWWWalk* otherRef = (fwHumanWWWalk*)walker ;
    ref->nextStepPosition(othersPos, othersmoveDir, walker->getStepStride
(), othersFrontPos);

    legCollideCheck[0].vec[0] = othersFrontPos[0] - myPos[0];
    legCollideCheck[0].vec[1] = 0;
    legCollideCheck[0].vec[2] = othersFrontPos[2] - myPos[2];
    legCollideCheck[1].vec[0] = othersPos[0] - myPos[0];
    legCollideCheck[1].vec[1] = 0;
    legCollideCheck[1].vec[2] = othersPos[2] - myPos[2];
    legCollideCheck[2].vec[0] = myFrontPos[0] - othersPos[0];
    legCollideCheck[2].vec[1] = 0;
    legCollideCheck[2].vec[2] = myFrontPos[2] - othersPos[2];
    legCollideCheck[3].vec[0] = myPos[0] - othersPos[0];
    legCollideCheck[3].vec[1] = 0;
    legCollideCheck[3].vec[2] = myPos[2] - othersPos[2];

    legCollideCheckDeg[0] = fwGetYAngleFromDir(legCollideCheck[0]) - fwGet
YAngleFromDir(moveDir);
    legCollideCheckDeg[1] = fwGetYAngleFromDir(legCollideCheck[1]) - fwGet
YAngleFromDir(moveDir);
    legCollideCheckDeg[2] = fwGetYAngleFromDir(legCollideCheck[2]) - fwGet
YAngleFromDir(othersmoveDir);
    legCollideCheckDeg[3] = fwGetYAngleFromDir(legCollideCheck[3]) - fwGet
YAngleFromDir(othersmoveDir);

    if( ((FW_REL_ANGLE(legCollideCheckDeg[0]) * FW_REL_ANGLE(legCollideChe
ckDeg[1])) < 0) //足の交差条件
        && ((FW_REL_ANGLE(legCollideCheckDeg[2]) * FW_REL_ANGLE(legCollideCh
eckDeg[3])) < 0)){

        v.negate(v); // otherPos -> pos
        hit = TRUE;

        break;
    }
    continue ;
}
} else {
    continue ;
}

}

if(!hit) {
    csVec3f othersmoveDir(walker->getStepMoveDirection()) ;
    csVec3f vv ;
    //
    // 相手の後ろ足と自分の行き先にもぐりこめるか?
    //

```

```

    vv[0] = myFrontPos[0] - othersTailPos[0] ; // others 後ろ足    my 前足 ( 予
定 )
    vv[1] = 0.0f;
    vv[2] = myFrontPos[2] - othersTailPos[2] ;
    c = othersmoveDir.dot(vv) ;

    if( 0.0f <= c && c < len ) {
        csVec3f xp, vert;
        xp.addScaled( othersTailPos, c, othersmoveDir );
        vert.sub( myFrontPos, xp );
        r2 = vert[0]*vert[0]+vert[2]*vert[2];

        if( r2 < len*len ) {
            v.negate(v); // otherPos -> pos
            hit = TRUE;
        } else {
            continue ;
        }
    } else {
        continue ;
    }
}

if( r2 < collideWalkerDist ) {
    collideWalkerDist = r2;
    collideWalkerPoint = othersPos;
    collideWalkerDir = v;
}
if(hit) { break ; }
}

if( hit ) {
    collideWalkerDist = sqrtf(collideWalkerDist);
    collideWalkerDir.normalize();
}

return hit;
}

```

道を譲らない歩行者の経路探索

```

csBool
fwWalker::searchSlipThrough( const csVec3f& pos,
    const csVec3f& moveDir,
    const csVec3f& prevDir,
    const float& search_stride,
    const float& relAngle,
    csVec3f& throughDir,
    float& throughStride,
    float& throughAngle)
{
    float stride = search_stride ;
    float dth = ::copysign(FWWALKER_COLLIDE_DIFF_SEARCH_ANGLE, relAngle) ;
}

```

```

dth *= -1.0 ;

csVec3f newDir = moveDir ;

csVec3f collidePos, wallDir, collideDir ;
float collideDist;

float humanMovingAngle = getStepAngle() ; //全体系

csBool isFW, isBK, isRT, isLT ;
isFW = IsForwardControlled();
isBK = IsBackwardControlled();
isLT = IsLeftControlled() ;
isRT = IsRightControlled() ;

if(IsTranslateControlled() && (isLT || isRT) ) {
    humanMovingAngle = fwGetYAngleFromDir(moveDir);
}

// 斥力のあるとき、moveDir = 転向したもの。prevDir=転向前
// 斥力の無いとき、moveDir=targetDir, prevDir=前ステップの向き

float angle = relAngle ;
float difangle = 0.0f ;
int i = 0 ;

//平行移動を一度解除する
this->translateMode = FALSE;
this->forwardWalkMode = FALSE;
this->backwardWalkMode = FALSE;
this->leftWalkMode =FALSE;
this->rightWalkMode =FALSE;

while(fabs(difangle - relAngle) <= FWWALKER_COLLIDE_TURNANGLE_MAX) {

    if(fwStage::collide(getHitPoints(), pos, newDir,
        stride+toeLength,
        humanMovingAngle+angle,
        wallDir, collidePos, collideDist, TRUE)) {

    } else if(strongWalkerCollide(this,
        pos, newDir,
        stride+toeLength*2.0,
        humanMovingAngle+angle,
        collideDir, collidePos, collideDist) ) {

    } else {
        //
        // 抜け道が発見されたので、変数をセットして復帰
        //

```

```

    throughDir = newDir ;
    throughStride = stride ;
    throughAngle = angle ;

    collisionTimeCount = 0;

    return TRUE ;
}

if(fabs(angle) < FW_TOLERANCE) { break ;}

difangle += FWWALKER_COLLIDE_DIFF_SEARCH_ANGLE ;
angle += dth ;

float s = sinf( CS_DEG2RAD(dth) );
float c = cosf( CS_DEG2RAD(dth) );
newDir = fwRotXZ(s, c, newDir);

//回転時の衝突回避
if(collideSearchTurn(this, pos, newDir, humanMovingAngle+angle)){
    break;
}

i++ ;
}

//歩幅を半分にして再度経路探索
stride *= 0.5f;
while(fabs(difangle - relAngle) <= FWWALKER_COLLIDE_TURNANGLE_MAX) {

if(fwStage::collide(getHitPoints(), pos, newDir,
    stride+toeLength,
    humanMovingAngle+angle,
    wallDir, collidePos, collideDist, TRUE)) {

} else if(strongWalkerCollide(this,
    pos, newDir,
    stride+toeLength*2.0,
    humanMovingAngle+angle,
    collideDir, collidePos, collideDist) ) {

} else {
    //
    // 抜け道が発見されたので、変数をセットして復帰
    //
    throughDir = newDir ;
    throughStride = stride ;
    throughAngle = angle ;

    //this->forwardWalkMode = TRUE;

    collisionTimeCount = 0;

```

```

    return TRUE ;
}

if(fabs(angle) < FW_TOLERANCE) { break ;}

difangle += FWWALKER_COLLIDE_DIFF_SEARCH_ANGLE ;
angle += dth ;

float s = sinf( CS_DEG2RAD(dth) );
float c = cosf( CS_DEG2RAD(dth) );
newDir = fwRotXZ(s, c, newDir);

//回転時の衝突回避
if(collideSearchTurn(this, pos, newDir, humanMovingAngle+angle)){
    break;
}

i++;
}

//回転方向を逆に
dth *= -1.0f;

i = 0 ;

while(difangle >= -FWWALKER_COLLIDE_TURNANGLE_MAX) {

    if(fwStage::collide(getHitPoints(), pos, newDir,
        stride+toeLength,
        humanMovingAngle+angle,
        wallDir, collidePos, collideDist, TRUE)) {

    } else if(collide(this,
        pos, newDir,
        stride+toeLength*2.0,
        humanMovingAngle+angle,
        collideDir, collidePos, collideDist) ) {

    } else {
        //
        // 抜け道が発見されたので、変数をセットして復帰
        //
        throughDir = newDir ;
        throughStride = stride ;
        throughAngle = angle ;

        collisionTimeCount = 0;
        return TRUE ;
    }
}

if(fabs(angle) < FW_TOLERANCE) { break ;}

difangle -= FWWALKER_COLLIDE_DIFF_SEARCH_ANGLE ;

```

```

    angle += dth ;

    float s = sinf( CS_DEG2RAD(dth) );
    float c = cosf( CS_DEG2RAD(dth) );
    newDir = fwRotXZ(s, c, newDir);

    //回転時の衝突回避
    if(collideSearchTurn(this, pos, newDir, humanMovingAngle+angle)){
        break;
    }

    i++ ;
}

return FALSE ;
#endif
}

```

道を譲る歩行者の経路探索

```

csBool
fwWalker::searchSlipThrough( const csVec3f& pos,
    const csVec3f& moveDir,
    const csVec3f& prevDir,
    const float& search_stride,
    const float& relAngle,
    csVec3f& throughDir,
    float& throughStride,
    float& throughAngle)
{
    float stride = search_stride ;
    float dth = ::_copysign(FWALKER_COLLIDE_DIFF_SEARCH_ANGLE, relAngle) ;
    dth *= -1.0 ;

    csVec3f newDir = moveDir ;

    csVec3f collidePos, wallDir, collideDir ;
    float collideDist;

    float humanMovingAngle = getStepAngle() ; //全体系

    csBool isFW, isBK, isRT, isLT ;
    isFW = IsForwardControlled();
    isBK = IsBackwardControlled();
    isLT = IsLeftControlled() ;
    isRT = IsRightControlled() ;

    if(IsTranslateControlled() && (isLT || isRT) ) {
        humanMovingAngle = fwGetYAngleFromDir(moveDir);
    }

    // 斥力のあるとき、moveDir = 転向したもの。prevDir=転向前
    // 斥力の無いとき、moveDir=targetDir, prevDir=前ステップの向き

```

```

float angle = relAngle ;
float difangle = 0.0f ;
int i = 0 ;

//平行移動を一度解除する
this->translateMode = FALSE;
this->forwardWalkMode = FALSE;
this->backwardWalkMode = FALSE;
this->leftWalkMode =FALSE;
this->rightWalkMode =FALSE;

//道を譲る歩行者
while(fabs(difangle - relAngle) <= FWALKER_COLLIDE_TURNANGLE_MAX) {

    if-fwStage::collide(getHitPoints(), pos, newDir,
        stride+toeLength,
        humanMovingAngle+angle,
        wallDir, collidePos, collideDist, TRUE)) {

    } else if(collide(this,
        pos, newDir,
        stride+toeLength*2.0,
        humanMovingAngle+angle,
        collideDir, collidePos, collideDist) ) {

    } else {
        //
        // 抜け道が発見されたので、変数をセットして復帰
        //
        throughDir = newDir ;
        throughStride = stride ;
        throughAngle = angle ;

        collisionTimeCount = 0;

        return TRUE ;
    }

    if-fwStage::collide(getHitPoints(), pos, newDir,
        stride*0.5f+toeLength,
        humanMovingAngle+angle,
        wallDir, collidePos, collideDist, TRUE)) {

    } else if(collide(this,
        pos, newDir,
        stride*0.5f+toeLength*2.0,
        humanMovingAngle+angle,
        collideDir, collidePos, collideDist) ) {

    } else {

```

```

//
// 抜け道が発見されたので、変数をセットして復帰
//
throughDir = newDir ;
throughStride = stride *0.5f;
throughAngle = angle ;

collisionTimeCount = 0;

return TRUE ;
}

if(fabs(angle) < FW_TOLERANCE) { break ;}

difangle += FWWALKER_COLLIDE_DIFF_SEARCH_ANGLE ;
angle += dth ;

float s = sinf( CS_DEG2RAD(dth) );
float c = cosf( CS_DEG2RAD(dth) );
newDir = fwRotXZ(s, c, newDir);

//回転時の衝突回避
if(collideSearchTurn(this, pos, newDir, humanMovingAngle+angle)){
    break;
}

i++ ;
}
}
//目的地とは逆回転方向のパスも探索

//回転方向を逆に
dth *= -1.0f;

i = 0 ;

while(difangle >= -FWWALKER_COLLIDE_TURNANGLE_MAX) {

    if(fwStage::collide(getHitPoints(), pos, newDir,
        stride+toeLength,
        humanMovingAngle+angle,
        wallDir, collidePos, collideDist, TRUE)) {

    } else if(collide(this,
        pos, newDir,
        stride+toeLength*2.0,
        humanMovingAngle+angle,
        collideDir, collidePos, collideDist) ) {

    } else {
        //
        // 抜け道が発見されたので、変数をセットして復帰
        //

```

```

    throughDir = newDir ;
    throughStride = stride ;
    throughAngle = angle ;

    collisionTimeCount = 0;

    return TRUE ;
}

if(fabs(angle) < FW_TOLERANCE) { break ;}

difangle -= FWWALKER_COLLIDE_DIFF_SEARCH_ANGLE ;
angle += dth ;

float s = sinf( CS_DEG2RAD(dth) );
float c = cosf( CS_DEG2RAD(dth) );
newDir = fwRotXZ(s, c, newDir);

//回転時の衝突回避
if(collideSearchTurn(this, pos, newDir, humanMovingAngle+angle)){
    break;
}

i++ ;

}
return FALSE ;
#endif
}

歩行者の状態更新関数
void
fwWalker::update()
{
    if( !getBody() || !isVisible() ) return;

    csBool doApplyPose = TRUE ;

    if( isActivated() ) {
        if( isWalkFrameEnd() ) {
            doApplyPose = makeOneStep();
            doApplyPose = TRUE ;
        } else {

            if(collideFrameTime(this)){
                this->setStepStride((this->getStepStride()) * 0.8f);
                nextWalkFrame();
            } else {
                nextWalkFrame();
            }

            if( collideFrameTime(this) && isCollideNow() ) { //フレーム中に他歩行者
                (主にネット経由)との衝突をチェック
            }
        }
    }
}

```

```

        stopCurrentStep();
    } else {
        nextWalkFrame(); // フレーム再生中
    }
}
}

setLastFramePos();

if(doApplyPose)
applyPose(); // 姿勢の更新

csVec3f tempVec;
tempVec.set(this->getCurFrameMovePosByCollision());
tempVec.scale(0.0f, tempVec);
this->setCollidedNow(FALSE);
this->setCollideNow(FALSE);
this->setCollisionNow(FALSE);
this->setCurFrameMovePosByCollision(tempVec);
}

```

衝突による現在座標調整

```

void
fwHumanWWWalk::applyBodyPose()
{
    //
    // 2002.Nov.11
    // 軸足中心に変更
    // 変更が大きいのので、完全に作り直してみました
    //
    // 注意：
    // 0 フレーム目に、今の立ち位置での姿勢が入ります。
    // 最終フレームは、目的の姿勢に 1 フレーム足りません。
    // 影響：
    // 1・turn() 時に、 と切り換えた場合、視点の切り換えが 1 フレ逆戻りしま
    // す。
    // 2・強制停止 (stopCurretStep()) した場合、視点が 1 フレ前後します。
    //
    float normalizedFrame;

    if( frameNo != -1 ) {
        normalizedFrame = float(frameNo)/float(maxFrameCount);

        // 前進歩行時： 軸足を原点に移動
        // 後進歩行時： 前足を軸とするが、無理やり後ろに移動しているため、後ろ足
        // 位置が多少奇妙になる。。
        // 平行移動(左右) 時： ムーンウォークになる。
        //
        Point3f *center = wwwalk->data[frameNo]->center;

        //-----
        // 2002.Nov.12 :
    }
}

```

```

// WalkAnim の絶対座標系軸足指定機能を無視したため、
// M-> degree は信用できません。
//
// local_bodyangle = wwwalk->data[frameNo]->M->degree ;
//
float local_bodyangle = 0.0f ; // WalkAnim 局所座標系での人体正面向き [deg]

float cs, ss ; // 座標変換 sin, cos 値
float lx, ly, lz ; // WalkAnim 局所座標系での腰位置

// 2003.01.27 :
// 足首座標を勘違いしていた
float legw ; // WalkAnim 局所座標系での軸足位置
if( wwwalk->stand() == FREEWWALK_LEFT ) // turn() では変化しない
    legw = footPos_L[0] ; //legw = wwwalk->size[BODY_LEG_L] ;
else
    legw = footPos_R[0] ; //legw = wwwalk->size[BODY_LEG_R] * -1.0 ;

if(standingRotation) {
    //----
    // turn() 時。
    // M->degree は信用しません。
    //local_bodyangle = wwwalk->data[frameNo]->M->degree ;
    //----
#ifdef _FIX_HALF_OF_TURNSTEP_
    // 2003.01.22 : turn() 時の足を交互に
    //
    local_bodyangle = (lastBodyAngle - firstBodyAngle) * float(frameNo)/float
(maxFrameCount / 2.0f );
#else
    local_bodyangle = (lastBodyAngle - firstBodyAngle) * normalizedFrame ;
#endif
}

#ifdef _WALKANIM_04_SMOOTH_TURN_WITH_MOVING
#undef _WALKANIM_04_SMOOTH_TURN_WITH_MOVING
#endif
//-----
// turn() 時、center(x) が微妙に前後移動を含むのを考慮する場合、
// 下記 ifdef を enable にする。
// enable にすると、!!turn :to AGENT で指定された agent が turn 時に
// 微妙に前後移動するため、発行エージェントが足踏み続けます。
//
// 2002.Jan.10 : !!turn :to_s ( エージェント指定 ) での不要な足踏みを避け
るため、undef しました。
//
#ifdef _WALKANIM_04_SMOOTH_TURN_WITH_MOVING
// (利用しません)
// WalkAnim ローカル座標系での center の軌跡を、軸脚中心に変換
// center(y) は、walkAnim の不具合により、常に前のステップを継続するものと
する。
//
ss = sinf( CS_DEG2RAD( local_bodyangle ) ) ;
cs = cosf( CS_DEG2RAD( local_bodyangle ) ) ;

```

```

    lx = center->x *cs + center->z * ss ;
    lz = center->x * (-ss) + center->z * cs ;

    ss = sinf( CS_DEG2RAD(firstBodyAngle) ) ;
    cs = cosf( CS_DEG2RAD(firstBodyAngle) ) ;
    curFramePelvisPos[0] = firstStepPos[0] + lx*ss - lz*cs ;
#ifdef _WALKANIM_TURNDEBUG_TEST
//    curFramePelvisPos[1] = firstStepPos[1] + center->y ;
#else
    curFramePelvisPos[1] = firstStepPos[1] + pelvisHeight ;
#endif
    curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss ;

#else
// turn 時の腰前後移動を行わない場合、こちらを利用します。
//
/* 前フレームと同じなので、計算する必要はありません。
 *
 */
//curFramePelvisPos[0] = firstStepPos[0] ;
//curFramePelvisPos[1] = firstStepPos[1] + pelvisHeight ;
//curFramePelvisPos[2] = firstStepPos[2] ;

#endif

    curFrameAngle = firstBodyAngle+( local_bodyangle ) ;

} else {
//-----
// walk(), run() 時
//
// localdir : キー入力による歩行向き
// stride : 現在のステップの歩幅 (移動量)
//
csVec3f localdir = localMovingDir ;
float stride = curStride ;
if(walkState == WALKANIM_STOP) {
    localdir = localMovingDir_prev ; // STOP 時はキーを離す前の状態を利用
    stride = 0.0 ; // STOP 時の歩幅はゼロ (nextStep() 参
照)
}

if(applyBodyTYPE_TRANSLATE) {
//---
// 平行移動モード
// 高さ以外の重心の動作を無視する。
//
//---
// ----
// ローカル座標系で平行移動。
//
lx = 0.0f ;
//if(localdir[0] != 0.0) { lx = center->x /* * localdir[0] */ ; }
if(localdir[0] < 0.0f ) { //---後ろ歩き

```

```

        lx = center->x * -1.0f ;
    } else if(localdir[0] > 0.0f) {
        lx = center->x ;
    }
}

#ifdef FIX_NOACTION_IN_TRANSLATE_AND_BACKWD
    if(localdir[0] > 0.0f ) { //---前
        ly = center->y ;
    } else {
        ly = pelvisHeight ;
    }
#else
    ly = center->y ;
#endif

lz = stride * normalizedFrame * localdir[2] ;

ss = sinf( CS_DEG2RAD(firstBodyAngle) ) ; //全体座標系へ投射
cs = cosf( CS_DEG2RAD(firstBodyAngle) ) ;

if(this->isCollisionNow()){ //衝突している場合
    if(this->isSideCollideNow()){ //側方衝突
        if(this->isCollidedNow()){ //衝突される側
            curFramePelvisPos[0] = firstStepPos[0] + lx*ss - lz*cs + curFrameMovePosByCollision[0] * COLLIDED_MOVE_FACTOR_X; //衝突する側に押される
            curFramePelvisPos[1] = firstStepPos[1] + ly;
            curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss + curFrameMovePosByCollision[2] * COLLIDED_MOVE_FACTOR_Z;
        } else { //衝突する側

            curFramePelvisPos[0] = firstStepPos[0] + lx*ss - lz*cs + curFrameMovePosByCollision[0] * COLLIDE_MOVE_FACTOR_X; //衝突するので歩く勢いが弱まる
            curFramePelvisPos[1] = firstStepPos[1] + ly;
            curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss + curFrameMovePosByCollision[2] * COLLIDE_MOVE_FACTOR_Z;

        }
    } else if(this->isForwardCollideNow()){ //正面衝突
        if(this->isCollidedNow()){ //衝突される側
            curFramePelvisPos[0] = firstStepPos[0] + FRONT_COLLIDE_MOVE_FACTOR*cs; //衝突する側に押される
            curFramePelvisPos[1] = firstStepPos[1] ;
            curFramePelvisPos[2] = firstStepPos[2] + FRONT_COLLIDE_MOVE_FACTOR*ss;
        } else { //衝突する側
            curFramePelvisPos[0] = firstStepPos[0] + FRONT_COLLIDE_MOVE_FACTOR*cs; //衝突するので歩く勢いが弱まる
            curFramePelvisPos[1] = firstStepPos[1] ;
            curFramePelvisPos[2] = firstStepPos[2] + FRONT_COLLIDE_MOVE_FACTOR*ss;
        }
    } else { //追突
        if(this->isCollidedNow()){ //衝突される側
            curFramePelvisPos[0] = firstStepPos[0] + lx*ss - lz*cs + curFrame

```

```

eMovePosByCollision[0] * COLLIDED_MOVE_FACTOR_X * 1.5f; //衝突する側に押される
    curFramePelvisPos[1] = firstStepPos[1] + ly;
    curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss + curFrameMovePosByCollision[2] * COLLIDED_MOVE_FACTOR_Z * 1.5f;
} else { //衝突する側

    curFramePelvisPos[0] = firstStepPos[0] + lx*ss - lz*cs + curFrameMovePosByCollision[0] * COLLIDE_MOVE_FACTOR_X * 0.7f; //衝突するので歩く勢いが弱まる
    curFramePelvisPos[1] = firstStepPos[1] + ly;
    curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss + curFrameMovePosByCollision[2] * COLLIDE_MOVE_FACTOR_Z * 0.7f;

}
}
} else { //衝突していない場合
    curFramePelvisPos[0] = firstStepPos[0] + lx * ss - lz * cs ;
    curFramePelvisPos[1] = firstStepPos[1] + ly ;
    curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss ;
}

//}

} else {

//-----
// WalkAnim 対処： STOP 時に X 軸方を向いてしまうため、
// STOP 時には bodyangle を考慮しない。
//
// WalkAnim が修正された場合、常に local_bodyangle = M->degree にすること。

// 2002/11/15 : WalkAnim が修正されました。
//
// STOP, START においては、center->M->degree が全く保証されていないようです。
// Walk 時は、normalizedFrame=frame/(NFrames) ; です。
//
//local_bodyangle = wwwalk->data[frameNo]->M->degree ;
//
    local_bodyangle = (lastBodyAngle - firstBodyAngle) * normalizedFrame ;

//TRACE(_T("WALK localangle %g\n"), local_bodyangle) ;

// --- 通常の歩行
// WalkAnim ローカル座標系での center の軌跡を、軸脚中心に変換
//
float center_x = center->x ;
if(localdir[0] < 0.0f ) { //---後ろ歩き
    legw*= -1.0f ;
    center_x *= -1.0f ;
}

ss = sinf( CS_DEG2RAD( local_bodyangle ) ) ; //--- ローカル座標での ce

```

nter 軌跡

```
cs = cosf(CS_DEG2RAD( local_bodyangle ) ) ;
lx = (center_x ) *cs + (center->z +legw) * ss  ;
ly = center->y ;
lz = (center_x ) * (-ss) + (center->z +legw) * cs - legw ;

ss = sinf( CS_DEG2RAD(firstBodyAngle) ) ; //--- 全体座標へ投影
cs = cosf( CS_DEG2RAD(firstBodyAngle) ) ;
curFramePelvisPos[0] = firstStepPos[0] + lx*ss - lz*cs ;
curFramePelvisPos[1] = firstStepPos[1] + ly ;
curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss ;
if(this->isCollisionNow()){ //衝突している場合
    if(this->isSideCollideNow()){ //側方衝突
        if(this->isCollidedNow()){ //衝突される側
            curFramePelvisPos[0] = firstStepPos[0] + lx*ss - lz*cs + curFrameMovePosByCollision[0] * COLLIDED_MOVE_FACTOR_X; //衝突する側に押される
            curFramePelvisPos[1] = firstStepPos[1] + ly;
            curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss + curFrameMovePosByCollision[2] * COLLIDED_MOVE_FACTOR_Z;
        } else { //衝突する側
            curFramePelvisPos[0] = firstStepPos[0] + lx*ss - lz*cs + curFrameMovePosByCollision[0] * COLLIDE_MOVE_FACTOR_X; //衝突するので歩く勢いが弱まる
            curFramePelvisPos[1] = firstStepPos[1] + ly;
            curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss + curFrameMovePosByCollision[2] * COLLIDE_MOVE_FACTOR_Z;
        }
    } else if(this->isForwardCollideNow()){ //正面衝突
        if(this->isCollidedNow()){ //衝突される側
            curFramePelvisPos[0] = firstStepPos[0] + FRONT_COLLIDE_MOVE_FACTOR*cs; //衝突する側に押される
            curFramePelvisPos[1] = firstStepPos[1] ;
            curFramePelvisPos[2] = firstStepPos[2] + FRONT_COLLIDE_MOVE_FACTOR*ss;
        } else { //衝突する側
            curFramePelvisPos[0] = firstStepPos[0] + FRONT_COLLIDE_MOVE_FACTOR*cs; //衝突するので歩く勢いが弱まる
            curFramePelvisPos[1] = firstStepPos[1] ;
            curFramePelvisPos[2] = firstStepPos[2] + FRONT_COLLIDE_MOVE_FACTOR*ss;
        }
    }
} else { //追突
    if(this->isCollidedNow()){ //衝突される側
        curFramePelvisPos[0] = firstStepPos[0] + lx*ss - lz*cs + curFrameMovePosByCollision[0] * COLLIDED_MOVE_FACTOR_X * 1.5f; //衝突する側に押される
        curFramePelvisPos[1] = firstStepPos[1] + ly;
        curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss + curFrameMovePosByCollision[2] * COLLIDED_MOVE_FACTOR_Z * 1.5f;
    } else { //衝突する側
        curFramePelvisPos[0] = firstStepPos[0] + lx*ss - lz*cs + curFrameMovePosByCollision[0] * COLLIDE_MOVE_FACTOR_X * 0.7f; //衝突するので歩く勢いが弱まる
        curFramePelvisPos[1] = firstStepPos[1] + ly;
    }
}
```

```

        curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss + curFrameMovePosByCollision[2] * COLLIDE_MOVE_FACTOR_Z * 0.7f;

    }
}
else { //衝突していない場合
    curFramePelvisPos[0] = firstStepPos[0] + lx *ss - lz * cs ;
    curFramePelvisPos[1] = firstStepPos[1] + ly ;
    curFramePelvisPos[2] = firstStepPos[2] + lx*cs + lz*ss ;
}
curFrameAngle = firstBodyAngle+( local_bodyangle ) ;

}
}

// 高さ方向は、線形補間します。
// curFramePos の [0], [2] は衝突判定に、[1] は stopCurrentStep で利用します。
curFramePos[0] = curFramePelvisPos[0] ; //
curFramePos[1] = firstStepPos[1] + (lastStepPos[1] - firstStepPos[1]) * normalizedFrame ;
curFramePos[2] = curFramePelvisPos[2] ;
}

//--- 2002.Nov.12
// turn() 時、center(x) が微妙に前後移動することを考慮する _WALKANIM_04_SMOOTH_TURN_WITH_MOVING == DEFINED
//
// 考慮しない場合には、Translation は不要
//
// walk()/run() 時は、軸足を中心にして回転 + 移動
//

#ifdef _WALKANIM_04_SMOOTH_TURN_WITH_MOVING
    // 人の位置 (腰)
    getBody()->setTranslation( curFramePelvisPos );
#else
    if(!standingRotation) {
        // 人の位置 (腰)
        getBody()->setTranslation( curFramePelvisPos );
    }
#endif

// 人の向き
csRotation rot;
rot.setAxis( 0.0f, 1.0f, 0.0f );
rot.setAngle( CS_DEG2RAD(curFrameAngle) );
getBody()->setRotation(rot);
}

```