IPA

2012 年度 未踏 IT 人材発掘・育成事業 採択案件評価書

1. 担当PM

プロジェクトマネージャー： 原田 康徳 PM
（日本電信電話株式会社 NTT コミュニケーション科学基礎研究所 主任研究員）

2. 採択者氏名

チーフクリエータ：小山 裕己
（東京大学大学院 情報理工学系研究科コンピュータ科学専攻（修士課程）（五十嵐研究室））

3. 委託金支払額

1,792,000 円

4. テーマ名

「こだわり」を簡単に実現できるアニメーション作成システム

5. 関連Webサイト

なし

6. テーマ概要

魅力的な 3D キャラクタアニメーション（以下、アニメーション）を作るためには、作者が細かい部分まで「こだわり」を発揮することが重要である。そこで本提案では、簡単に「こだわり」を実現できるアニメーション作成システムを開発する。アニメーションの作成にはスケルタルアニメーションと呼ばれる手法が一般的に用いられている。この手法は、ユーザがキャラクタのスケルトン（骨格）の動きを指定すると、それに付随してキャラクタの表面メッシュ（皮膚など）が動き、さらに髪の毛などの動きが物理シミュ

レーションによって生成されるというものである。

本提案ではスケルタルアニメーションの手法を用いてアニメーションを作成するが、特に髪の毛などの物理シミュレーションを用いて動きを生成する部分に対して、キャラクタの個性を表現する、つまり「こだわる」ことができる点が最大の特徴となる。

「こだわり」の実現方法としては、例えば

（1）キャラクタが激しく動いたりしたとき

（2）キャラクタがある特定のポーズをとったとき

などに対して、それぞれキャラクタの個性を表す髪の毛などの形状を指定することによって、キャラクタの個性がよく表れたアニメーションを生成できるようにする。

本システムはフリーソフトウェアとしての公開を目指し、より多くのユーザを獲得できるようにする。本システムを用いて多くのユーザがそれぞれの「こだわり」を実現したアニメーションを作成し、YouTube やニコニコ動画といった動画共有サイトへの作品の公開が発展していくことが期待される。

7. 採択理由

物理エンジンの普及により簡単にリアルなアニメーションが作れるようになったことで、逆に似たようなアニメーションが多くなってしまった。すべての計算をゆだねて簡単にリアルさが得られるがゆえに、ちょっとした修正でも不自然さが出てしまう。小山君の作るシステムは、計算方法だけでなく、こだわりを自然に指示することができるインタフェースの設計もまさに未踏の領域である。出来上がるシステムも楽しみであるが、それが普及したときの個性豊かなデジタルコンテンツで満ち溢れた世の中も楽しみである。

8. 開発目標

3DCG技術とアニメーション制作

従来、アニメーション制作には大別して2つの方法しか無かった。それは、ドラえもんやドラゴンボールなどに代表されるセルアニメと、ビデオカメラ等で現実世界を撮影する実写アニメである（図1）。セルアニメはパラパラ漫画のように沢山の絵を描いて切り換えることでアニメーションさせる方法で、特に日本で独自のデフォルメ手法が発達している。

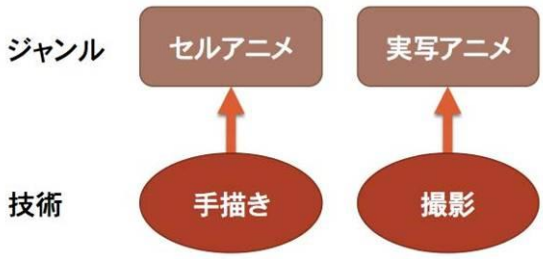

図1：3DCGの登場以前のアニメーション制作の状況

ところが、3DCG 技術の発達により、アニメーション制作を取り巻く状況は大きく変化してきた（図 2）。

まず、1995 年に公開された映画『トイ・ストーリー』（原題：Toy Story）を筆頭に、3DCG 独特の人形のようなキャラクタによるカートゥーンアニメーションというジャンルが登場した。現在に至るまでにこの手のアニメーション映画は数多く制作され、我々はごく自然にそれらを鑑賞するようになった。テレビコマーシャル等でもこの手の表現は日常茶飯事となった。

また、フォトリアリスティックな 3DCG 表現による実写さながらのアニメーションも多く作られるようになり、一つのジャンルを形成している。例えば株式会社 SQUARE ENIX による『Agni's Philosophy』は 3DCG でありながら実写に見紛うハイクオリティなアニメーションとして有名である。

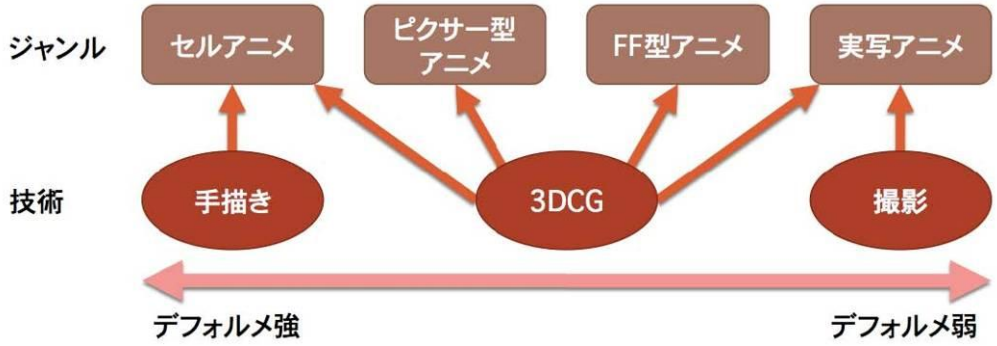

図2：3DCGの登場以降のアニメーション制作の状況

さて、ここで注目しておきたいのは、3DCG 技術がアニメーション表現として新たなジャンルを生み出した点だけではなく、既存のセルアニメや実写アニメの制作にも大きな影響を与えているという点である。

セルアニメの制作方法は、現在過渡期を迎えていると言っても過言ではないかもしれない。2012 年に制作された多くのセルアニメは、その制作工程の一部または全てにおいて 3DCG 技術が採用されている。特に映画『009 RE:CYBORG』は全編が 3DCG

技術によって制作されていることで話題を呼んだ作品である。また、従来の方法で制作されるセルアニメにおいても、将来的にゲーム化や映画化される際に 3DCG によって再構築されるケースが増えたため、そのキャラクタデザインが 3DCG を意識したものになっていることが多いという話もある。

実写アニメに対する 3DCG 技術の影響も、もちろん大きい。映画『アバター』(原題: Avatar)では実写でありながらも架空の生物を 3DCG 技術で描画することで、話題を呼んだ。

こうした 3DCG 技術の発達によるアニメーション制作を取り巻く状況の変化として無視できないのが、初音ミクの存在である。初音ミクは、初音ミクをモチーフとしたコンテンツを限られた範囲内で自由に公開することが可能なキャラクタとして有名であり、有志らによる初音ミクの三次元キャラクタモデルの無料公開、及び樋口優氏によるアニメーション作成ソフトウェアである MikuMikuDance の無料公開によって、たちまちエンドユーザによるアニメーション制作ブームが巻き起こった。ここで生まれるアニメーションはセルアニメやピクサー型アニメに属するものである。

以上のように、3DCG 技術は、もはやアニメーション制作において必要不可欠な技術である。3DCG 技術はアニメーション制作の状況を大きく変えてきたし、これからもその重要性は高まっていくと考えられる。

ここで、アニメーション制作のうち重要となるのはキャラクタのアニメーション、すなわちキャラクタアニメーションである。ここまでは単にアニメーションという言葉を用いてきたが、キャラクタアニメーションと置き換えても以上の議論は成立する。以下では単にアニメーションといった場合はキャラクタアニメーションを指す場合がある。

「こだわり」の重要性

さて、ここから先は特にデフォルメの強いジャンルのアニメーション(主にセルアニメ及びピクサー型アニメ)について考えていく。

こうしたアニメーションの多くは、個性的で魅力的であることを目標に制作されている。そこで私は、個性的な魅力的なアニメーションの条件の一つとして、アーティストの「こだわり」が実現されていることが重要だと考える。アーティストが小さな「こだわり」を重ねていくことで初めてアニメーションに個性と魅力が生まれるのである。

「こだわり」という言葉自体は大変広い意味の言葉だが、ここではアーティストの「ここはこういう表現にしたい」という情熱や欲求のことを「こだわり」と呼ぶこととする。

以下に「こだわり」の一例を示す(図3)。まず、アーティストが自身で制作したアニメーションを見たときに、前髪の一部によって目が隠れてしまっていたこと(図28、左)に気付いたとする。このとき、「この前髪は目を隠さないように避けて欲しい」と考えたとする。そこでアーティストが手を加えて、前髪が目にかからないように調節した(図3、右)ならば、これが「こだわり」の実現である。このように、一つ一つの「こだわり」は小

さいかもしれないが、こういった小さな「こだわり」を重ねていくことが重要であると考える。

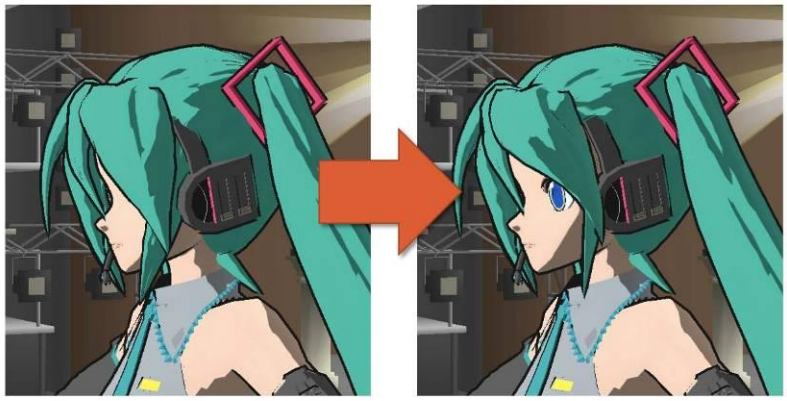

図3：「こだわり」の一例

こうした「こだわり」の実現に必要なものが何かというと、それはアーティストの表現が自由であることであると考える。ここでの表現の自由とは法律などの社会的制約を受けないということではなく、画面の都合だとか、画材の都合だとか、そういった技術的な自由のことである。

例えば、実写アニメでは一度撮影して得られたアニメーションは、特殊な技術を用いない限りは、役者の位置を変えたり、役者のポーズを変えたりといった操作は困難である。こういった観点ではアーティストは不自由であり、「こだわり」の実現が難しいといえる。

2Dキャラクタアニメーションにおける「こだわり」

では2Dキャラクタアニメーション、すなわち従来のセルアニメではどうだったかというと、アーティストは非常に自由であったといえる。

例えば、キャラクタの頭部よりも巨大な眼球を有するキャラクタを描いたり（図4、左）、キャラクタを見る方向に応じて顔の輪郭を大きく変えたり（図4、中）、三次元的な構造を考えることが難しいような髪型を用いたり（図4、右）することが容易であり、更に、これらの極端なデフォルメを維持したままアニメーションさせることもできる。

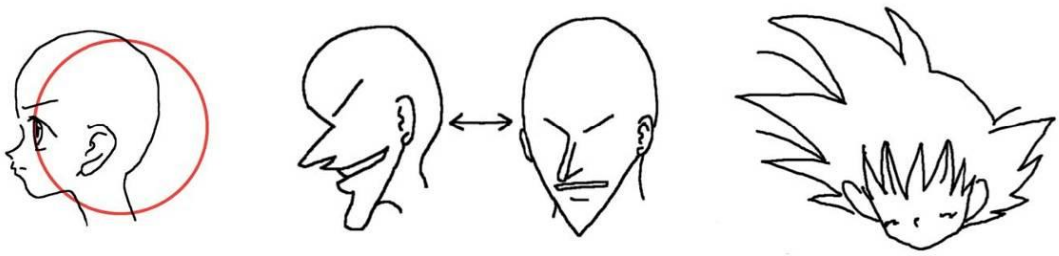

図4：2Dキャラクタアニメーションにおけるアーティストの自由度の高さを示した例

このように、2D キャラクタアニメーションでは技術的な制約が少なく、アーティストは非常に自由であった。したがって、アーティストの大抵の「こだわり」は実現可能であったといえる。

3Dキャラクタアニメーションにおける「こだわり」

しかしながら、3DCGによるキャラクタアニメーションでは、2Dキャラクタアニメーションに比べてアーティストの自由度が低いといえる。

例えば、2Dキャラクタアニメーションで問題なく表現できた内容を、そのまま3DCGで表現しようとしても、上手くいかないことが多い。このことを示す具体例としては、2Dキャラクタアニメーションと同様の顔の輪郭のデフォルメを3DCGで表現しようとしても、見る方向を変えるとたちまち表現が崩壊してしまう（図5）ことなどが挙げられる。

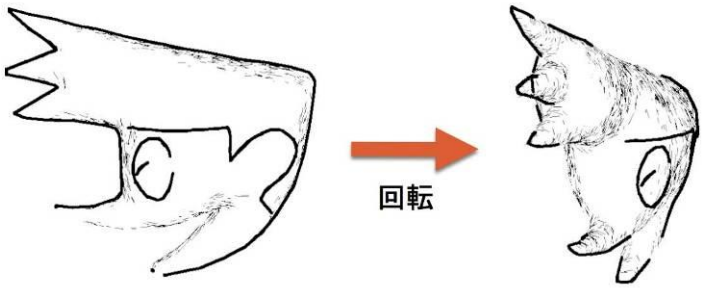

図5：3Dキャラクタアニメーションにおいて自由な表現を試みたが失敗した例

このように、3Dキャラクタアニメーションにおけるアーティストの表現が自由でない分、「こだわり」の実現を諦めるしかない部分が発生していると考えられる。このような不自由さの原因として、

1.どの方向からもキャラクタを見ることができてしまうこと

2.物理演算によって生成された動きをアーティストは調節できないこと

の二つの問題点が考えられる。

1.については、どの方向からでもキャラクタを見ることができるという3DCGの特徴でもあり特長でもある性質が、逆にどの方向から見ても三次元的な整合性のとれた形状のデフォルメしか許さないという制約を課してしまっているということである。

2.についても同じで、物理演算によってリアリスティックな動きを自動で生成することができるという3DCGの特徴でもあり特長でもある性質が、逆に無個性な動きしか表現できなくさせているということである。また、アーティストが物理演算結果をコントロールすることは現在のシステムでは大変難しい。これは、揺れもの、特に髪の毛や衣服の動きの表現において問題となる。

こういった問題点を解決しない限りは、3Dキャラクタアニメーションは無個性で魅力

のない状態、つまりアーティストの「こだわり」が実現されていない状態に陥りやすくなってしまう。アニメーション制作における3DCGの役割の大きさについては上述した通りだが、3DCGの重要性が今後ますます高まっていくのと同時に、これらの問題も大きくなっていくと考えられる。

本プロジェクトの目的

以上のような背景及び問題点を踏まえ、本プロジェクトの目的を以下のように設定した。

まず最終的な目的は、3DCGによって制作されるアニメーションを、より個性的に、そしてより魅力的にすることである。そのための手段として、3Dキャラクタアニメーションにおけるアーティストの表現の自由度を向上させるようなソフトウェアを開発し、アーティストがより多くの「こだわり」を実現できるようにする。

特に、主にデフォルメが重要となるジャンルにおいてアーティストの表現の制約となっている上述の二つの問題点を緩和ないし解決することを具体的な目的とする。

9. 進捗概要

進捗概要

目的を達成するため、アーティストが「こだわり」を実現するための機構を持つ物理エンジンである「こだわり物理エンジン」の開発を行った。

本物理エンジンは、一般的な3DCGソフトウェアに搭載されている物理エンジンと同様、揺れものの動きを自動生成するための物理演算を行うものである。揺れものとは、主に髪の毛や衣服など、キャラクタの動きや外力などに応じて受動的な物体として物理的に自然な挙動をするべき部分（図6）を指す言葉である。

図6：三次元キャラクタモデルにおける揺れものの例

本物理エンジンの特徴は、見る方向に応じて揺れものの形状を変化させることが可能な点である。例えば、正面から見た場合の髪型、左側から見た場合の髪型、右側から見た場合の髪型等をそれぞれ別々の髪型として表現することができ、実行時にはその時点でのキャラクタを見る方向に応じて髪の毛の物理演算の内容を適応させることができる。

この機能は、上述の従来の 3D キャラクタアニメーションにおける表現上の制約を二つ同時に緩和ないし解決するものである。まず、三次元的な表現でありながら、最終的な出力画面が二次元的であることを利用して、三次元的な整合性のとれた形状のデフォルメだけでなく、見る方向に応じた形状のデフォルメを許すというのが一点目である。次に二点目は、従来は全てをシステムによる自動計算に任せるしかなかった物理演算に対して、見る方向に応じて形状を指定するというアプローチによって、アーティストがコントロールを獲得できるという点である。

アーティストは本物理エンジンを用いることによって従来の 3D キャラクタアニメーションの制作に比べ自由に表現することができるようになるため、より多くの「こだわり」の実現が可能となる。

本物理エンジンを用いて実現できる「こだわり」の一例として、キャラクタの髪の毛の、特にアホ毛と呼ばれる部分の向きに関する適用例を紹介する（図 7）。アホ毛は特に日本のセルアニメでよく用いられる表現であり、特徴として、どの方向から見ても常に横向きに見えるという性質がある。従来の物理エンジンではこのような特徴を表現するという「こだわり」を実現するのが困難だったのに対し、本物理エンジンでは簡単に実現可能である。

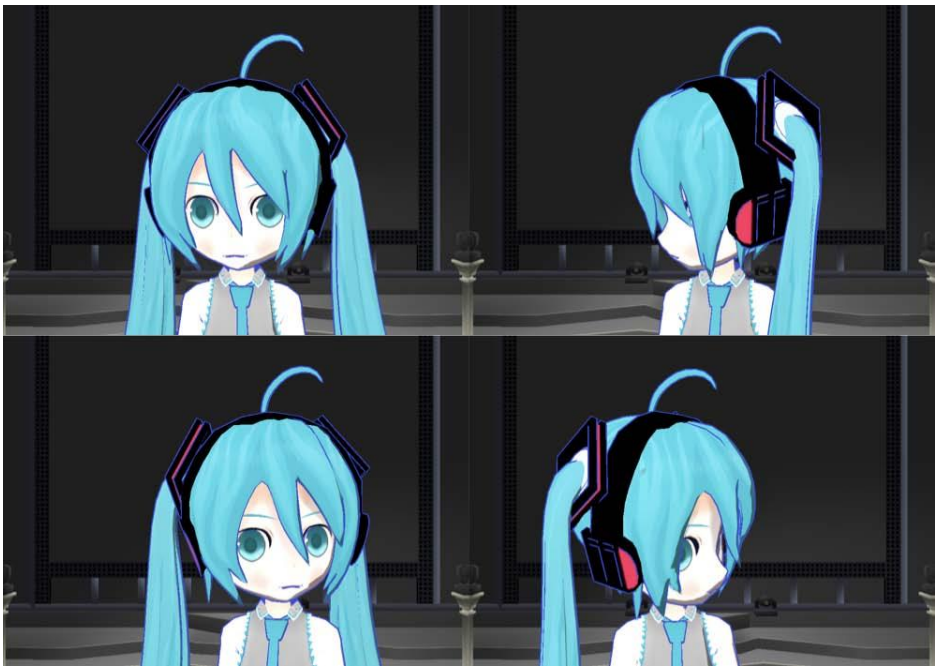

図7：どの方向から見てもアホ毛が横向きに見える「こだわり」の例

本プロジェクトの成果物は、上述のような物理エンジンを実現するためのアルゴリズムと、それを実装したライブラリ、そして Unity プラグインである。アルゴリズムについては学術論文としてまとめ、発表を行う予定である。ライブラリと Unity プラグインについてはウェブページにてソースコードの無料配布を行っている。このように技術をオープンな形で提供することにより、学術的にも産業的にも広く貢献できると考える。

更に、本物理エンジンを用いて効率的にアニメーション制作を行うためのユーザインタフェースとして、スケッチによって簡単に素早く揺れものの形状を編集できる機能を開発した。この機能は Unity プラグインの一部として実装されている。

開発内容

(1)アルゴリズム

(a)アルゴリズムの概観

本プロジェクトで開発した物理演算アルゴリズムは、アーティストからの入力として、通常の物理演算と同様に揺れものの基本形状を受け取るのに加え、キャラクタを見る方向とその方向から見たときの揺れものの形状のペアを 0個以上受け取る。実行時には、その時点でキャラクタを見ている方向を考慮して物理演算を行うことによって、見る方向に応じて揺れものの形状を変化させるという「こだわり」を実現する。なお、実行時には入力として受け取らなかった方向からキャラクタを見ることももちろん許され、どの方向から見ても自然に見えるように物理エンジンが適切に処理する。

図8はこれらを模式的に示したものであり、ここではウサギのキャラクタの耳を揺れものとして、見る方向と形状のペアを 3個入力した場合を示している。

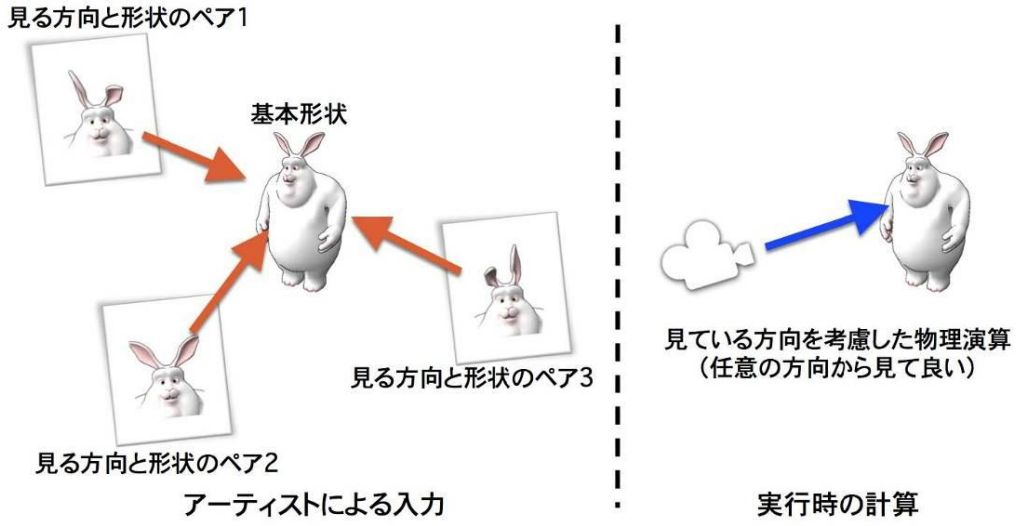

図8：開発アルゴリズムにおけるアーティストによる入力と実行時の計算の模式図

アルゴリズムの内容としては、まず事前計算として入力形状及び基本形状の解析

を行い、形状同士の補間が可能な形式に変換する。実行時には、キャラクタを見ている方向を元に形状同士の補間のための適切な重みを計算し、実際に形状同士を補間することによって新たな形状を計算し、これを物理演算に反映させるという流れになっている。このアルゴリズムを疑似コードで表現すると、次のようになる:

入力形状及び基本形状を解析する;
Loop{
見ている方向を元に重みを計算する;
得られた重みを用いて形状を補間する;
弾性体としての基本形状を補間によって得られた形状に更新する;
通常の物理演算を実行する;
}

続いて、図9に本プロジェクトで開発した物理演算アルゴリズムと、個々の手法の関係を示す。開発した物理演算アルゴリズムは、3DCGにおける最先端の学術研究の物理演算アルゴリズムを独自に拡張し、「こだわり」を実現するための、見る方向に応じて揺れものの形状を変化させる機能を付与したものである。この機能を実現するために、大きく分けて三つの手法を新たに提案している。

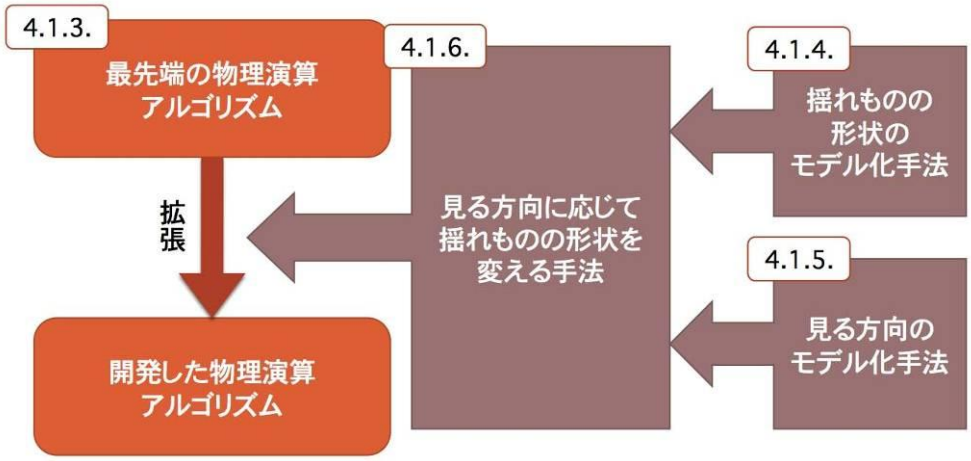

図9：開発した物理演算アルゴリズムと各手法の関係

以降では、まず3Dキャラクタアニメーションに用いられるキャラクタモデルのデータ構造と仕組みについて説明し(b)、続いてベースとなる物理演算アルゴリズムの紹介をする(c)。次に揺れものの形状のモデル化に関する手法(d)とキャラクタを見る方向のモデル化に関する手法(e)について説明し、これらを用いて見る方向に応じて揺れものの形状を変化させる手法(f)を解説する。

(b)キャラクタモデルのデータ構造と仕組み

ここではまず今回使用したキャラクタモデルのデータ構造とその仕組みについて簡単に説明する。なお、ここで説明するキャラクタモデルの方式は、一般的な3Dキャラク

タアニメーション制作でも用いられるものである。

キャラクタモデルは大きく分けてskeletonとskinの二つのデータ構造からなる（図10）。Skeletonはキャラクタの姿勢や動きを定義するためのもので、その名の通り骨格のような役割を持つ。また、これは実際に画面に描画されることはない。Skinはキャラクタの見た目を定義するためのもので、その名の通り皮膚のような役割を持つ。また、これはskeletonの状態に応じて適宜変形されて画面に描画される。つまり、キャラクタを動かすにはskeletonを動かせばよく、あとはシステムが自動的にskinを変形させて描画してくれる。

図10：Skeletonとskinの例

Skeleton は joint と呼ばれる構成要素による木構造を有している。Joint は関節のような役割を持ち、データ構造としては三次元座標上の位置と方向を持つ構造体として表現することができる。また、skeleton は通常骨盤に対応する joint を根とする木構造とすることが多い。ここで注意が必要なのは、髪の毛や衣類等の揺れものに対しても skeleton、特に枝分かれのない木構造の skeleton を入れておく必要があるということである。ここでは揺れものに入っている枝分かれのない木構造の skeleton を、joint が一列に並んでいることから、joint chain と呼ぶことにする（図 11）。

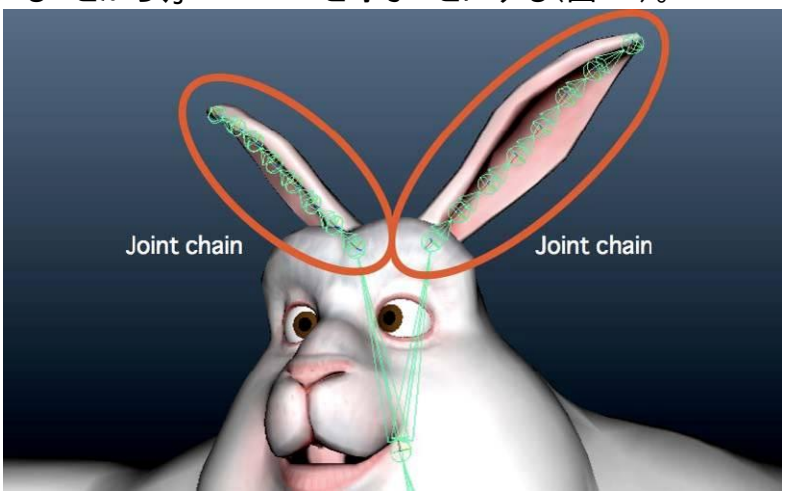

図11：揺れものに設定されているjoint chainの例

揺れものに対する物理演算を行うとは、joint chain の動きを計算することに他ならない。すなわち、joint chain に含まれている各 joint の位置を、物理法則に基づいて、物理エンジンが次々と計算していくということである。

(c)Joint Chain の物理演算手法

続いて、開発した物理演算アルゴリズムのベースとなる物理演算アルゴリズムについて簡単に述べる。

Joint chain の動きを計算するベースとなる手法として、2011 年に Müller らによって提案された oriented particles と呼ばれる手法を採用した。Oriented particles は shape matching と position based dynamics を元にした手法で、三次元的な構造の solid、二次元的な構造の shell、一次元的な構造の rod に関して弾性体物理演算を行うことができる手法である。弾性体とは外力等に応じて変形可能な物体で、外力等を除去すると内部的な力によってその物体の基本形状に戻る性質のある物体のことである(図12)。Joint chain の動きの計算には、一次元的な構造の rod の弾性体物理演算が活用できる。

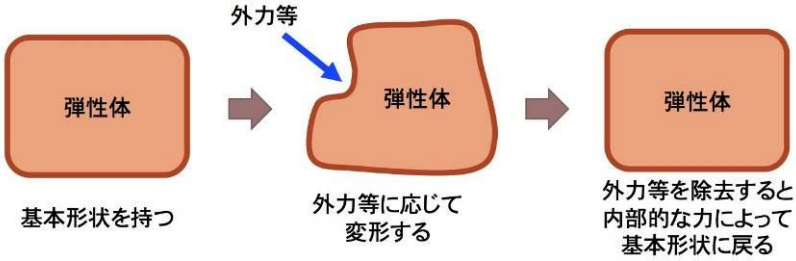

図12:弾性体の性質を簡単に説明した模式図

Oriented particlesの長所としては、リアルタイムで計算が可能であり、数値計算が無条件で安定であるなどの点が挙げられる。本プロジェクトで開発したアルゴリズムはこの手法をベースにしているため、これらの長所は踏襲されている。

また、3Dキャラクタアニメーションにおける揺れものは基本的にはinextensibility、すなわち長さが変わらないという性質を持っているべきだと考えられる。しかしながら、Oriented particlesは単純な弾性体物理演算でしかないため、inextensibilityを保障できない。そこで、position based dynamicsで提案されている距離制約を用いてinextensibilityを実現した。

以上の方法により、3Dキャラクタアニメーションにおける自然なjoint chainの動きが表現できるようになった。以降ではこの方法を拡張して「こだわり」を実現していくための手法について具体的に説明していく。

(d)Joint Chainの形状のモデル化手法

上述の疑似コードで示した通り、本アルゴリズムではjoint chain同士の形状の補間を行う。三次元的な形状を補間することは学術研究としても未だに難しい問題であるが、本アルゴリズムで扱うのはjoint chainという特殊なデータ構造を持つ形状に対する補間であるため、ここではjoint chainに特化した形状補間方法を提案する。

Joint chainにおける形状とは、全てのjointの位置と方向の状態のことを指す。つまり、n個のjointを持つjoint chainの形状 Cについて、 C =(X,O)と表すことができる。ただし、X =(x1,x2,･,xn)は全てのjointの位置の組を表しており、$\mathbf{x}_i \in \mathbb{R}^3$ は各jointの位置の座標である。また O =(o1,o2,･,on)は全てのjointの方向の組を表しており、$\mathbf{o}_i \in \mathbb{R}^4$ は各jointの方向を四元数で表したものである。Joint chain同士の形状の補間をするとは、 m個の形状 C1,C2,･,Cmに対して、重みw、すなわちm個の実数

$w_1, w_2, \cdots, w_m \left(\sum_{i=1}^{m} w_i = 1 \right)$ を用いて

$$C_{\mathbf{w}} = \sum_{i=1}^{m} w_i C_i = \left(\sum_{i=1}^{m} w_i X_i, \sum_{i=1}^{m} w_i O_i \right)$$

を計算することである。したがって形状補間に必要なのは、jointの位置の組X同士の補間の定義と、jointの方向の組 O同士の補間の定義である。これらが定義できるように、位置の情報と方向の情報をモデル化する必要がある。以下では位置の情報 Xに関するモデル化と、方向の情報 Oに関するモデル化を分けて考える。

まず、位置の情報 *X*に関するモデル化を考える。Joint chainの性質をまとめると、

- 形状が変わっても隣接するjoint間の距離は変化しない
- 隣接するjoint同士には親子関係があり、親側の末端と子側の末端を定義できる
- 親側の末端のjointはキャラクタのkinematicなjointに繋がっている

などが挙げられる。位置の情報を補間するのに単純な位置座標の補間を行ってしまうと、隣接するjoint間の距離が変化してしまうので不適である。したがって、位置座標以外の表現で位置の情報 *X*を表現してから補間を行い、そこからまた位置座標に戻すという過程を踏む必要がある。

上述のjoint chainの性質を考慮すると、joint chainの位置の状態を記述するには、子側の末端のjoint以外の全てのjointについて、そのjointの親の位置からそのjointの位置への方向ベクトルを、そのjointの位置からそのjointの子の位置への方向ベクトルに回転させるような回転変換を用いる方法が考えられる。ここでの回転変換とは三次元空間中の任意の回転を表すことができる必要があり、且つ後の操作で補間をする必要があるため、回転変換を表す媒体として四元数を用いることとする。この方法はつまり、親側の末端のjointから数えて*i*番目のjointに対する回転変換を表す四元数を $\mathbf{r}_i \in \mathbb{R}^4$ として、四元数の組 $R = (\mathbf{r}_1, \mathbf{r}_2, \cdots, \mathbf{r}_{n-1})$ を用いて位置の情報 *X* を記

述し直すという方法である。図13にこの方法を模式的に示す。

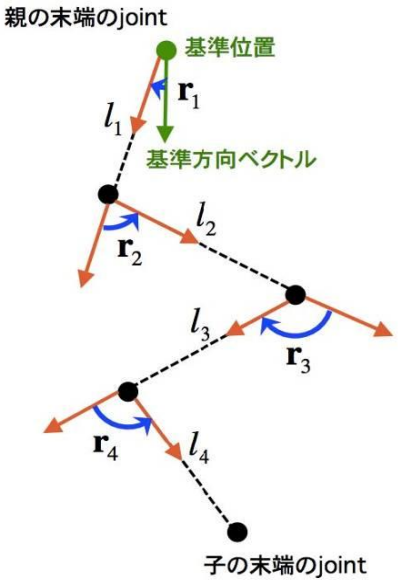

図13：Joint chainの位置情報のモデル化の模式図

ただし、親側の末端のjointは常に基準位置、すなわち原点にあるとし、親側の末端のjointにおける回転は基準方向ベクトルからの回転を計算するものとする。また、各joint間の距離は $l_1, l_2, \cdots, l_{n-1} \in \mathbb{R}$ と表すこととし、これらは形状が変わっても値が変わることがないものである。

Jointの位置座標の組Xから回転を用いて位置の情報を表現し直したR を計算するには、まず各joint間の距離 $l_1, l_2, \cdots, l_{n-1}$ を計算し、続いて親側の末端のjointから順に回転変換を計算していくだけで良い。逆にR から位置座標の組X を計算するには、まず親側の末端のjointの位置座標を基準位置として、基準方向ベクトルを回転させて次のjointの位置への方向ベクトルを得る。続いて事前に計算しておいた距離だけその方向ベクトルに沿って移動した点を次のjointの位置座標とする。この操作を子側の末端のjointの位置が決まるまで繰り返すだけで良い。この位置座標の計算方法はforward kinematicsと呼ばれる手法に良く似た操作である。

以上の方法を用いることで、座標として表されたjoint chainの形状の位置成分を、四元数による回転の表現によって記述し直すことができた。あとは、座標の表現での補間を行う代わりに、回転の表現での補間を行えば良い。

続いて、方向の情報Oに関するモデル化を考える。こちらは元々四元数による方向の表現が為されており、且つ上述のような位置の情報のモデル化を行っているため、単純に個々の方向を補間するだけで良い。

以上のようなjoint chainの形状のモデル化の方法によって、揺れものに関して、アーティストによって入力された形状及びキャラクタモデルの元々の形状同士を自由な

重みで補間することができるようになった。

(e)見る方向のモデル化手法

続いて、アーティストによって入力された見る方向と、実行時に実際に見ている方向から、その時点における補間の重みを計算する手法について提案する。

まず、ここではキャラクタモデルの kinematic な joint を一つ選び、その joint に対する相対的な視点への方向ベクトルのことを、キャラクタモデルを見る方向と定義することにする。ここで、joint は位置の他に方向の情報も持つので、joint の方向も考慮する必要があることに注意する。

視点の位置を $\mathbf{x}_{\text{camera}} \in \mathbb{R}^3$、選んだkinematicなjointの位置を $\mathbf{x}_{\text{kinematic}} \in \mathbb{R}^3$、選んだkinematicなjointの方向を四元数による表現で $\mathbf{o}_{\text{kinematic}} \in \mathbb{R}^4$ と表すことにする。このとき、選んだjointから視点への方向ベクトルは

$$\mathbf{d} = \frac{\mathbf{x}_{\text{camera}} - \mathbf{x}_{\text{kinematic}}}{\left\| \mathbf{x}_{\text{camera}} - \mathbf{x}_{\text{kinematic}} \right\|} \in \mathbb{R}^3$$

と表される。更に、jointの方向による影響を打ち消すために、方向ベクトル $\mathbf{d}$ を $\mathbf{o}_{\text{kinematic}}^{-1}$ だけ回転させた方向ベクトル $\mathbf{d}'$、すなわち

$$\left(0; d_x', d_y', d_z'\right) = \mathbf{o}_{\text{kinematic}}^{-1} \cdot \left(0; d_x, d_y, d_z\right) \cdot \overline{\mathbf{o}_{\text{kinematic}}^{-1}}$$

によって定義される方向ベクトル $\mathbf{d}' = \left(d_x', d_y', d_z'\right) \in \mathbb{R}^3$ を、このアルゴリズムにおける見る方向として用いる。

の形状の補間に用いる重み wを計算すれば良い。

上述のように定義した見る方向を用いて、アーティストの入力である m個の見る方向を $\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_m$ と表し、現在見ている方向を dと表すことにする。このとき、以下のアルゴリズムによって重み $\mathbf{w} = \left(w_0, w_1, w_2, \cdots, w_m\right) \in \mathbb{R}^{m+1}$ を計算する。

```
for (i = 1 to m) {
        コサイン距離 =内積 (d[i], d);
        正規化された距離 = (コサイン距離 + 1.0) * 0.5;
        w[i] =累乗 (正規化された距離,影響力[i]);
}
合計値 = w[1] + w[2] + … + w[m];
if (合計値 >= 1.0) {
        for (i = 1 to m) {
                w[i] = w[i] /合計値;
        }
        合計値 = 1.0;
}
```

w[0] = 1.0 −合計値;

ここで影響力とは 0より大きい実数値で、アーティストは入力毎に影響力の値を調節することによって、入力の見る方向の影響力を調節することができる。例えば、*i*番目の入力に対して影響力の値を非常に小さく設定することによって、 **d**がある程度**di**から離れてもwiの値を大きくすることができる。このようにして得られた

wは $\sum_{i=0}^{m} w_i = 1$ という制約を充たしているので、そのまま形状の補間のための重みとして用いることができる。すなわち、 C_0をjoint chainの基本形状、 C_1, C_2, C_mをアーティストによるjoint chainの入力形状として、

$$C\mathbf{w} = \sum_{i=0}^{m} wiCi$$

が補間結果の形状である。

上述のような重みと補間の計算方法の特徴としては、まずアーティストが見る方向を全く入力しなかった場合には、常に基本形状を補間結果として現れることになる。また、見る方向の入力が少なくても、入力が疎らな方向に対しては基本形状が強く補間結果に影響を与えるだけであるので、問題はない。逆にアーティストが見る方向を非常に沢山入力したりしても、それらは滑らかに補間されるので問題はない。

なお、他にも類似の目的で見る方向から重みを計算している事例が見受けられるが、アーティストによる入力の数に大きく左右されやすいこと、基本形状の扱いが特殊になってしまうこと、見る方向の制御が非直感的であるなどの点において、提案した手法の方がより実践的な手法だと考える。

以上の方法により、見る方向に応じて揺れものの形状を変化させるアルゴリズムが実現できた。

⑵ライブラリ

本プロジェクトでは見る方向に応じて揺れものの形状を変化させることができる新しいアルゴリズムを開発したが、技術的には3DCG分野の最先端のアルゴリズムを多く利用しており、また多少の数学的な知識がないと難解な部分もあるため、この分野を専門としているプログラマ以外では実装が難しいと思われる部分も多い。

そこで、この分野の専門でないプログラマ、具体的にはゲームプログラマやグラフィクスソフトウェア支援ツール開発者などでも本プロジェクトで開発したアルゴリズムを使えるように、C++で実装されたライブラリの開発を行い、ソースコードを無償で配布することにした。

なお、C++はゲームの開発やグラフィクスソフトウェア支援ツールの開発では最も使用されている言語の一つであるため、本ライブラリもC++を用いて実装を行った。

このライブラリによって、誰でもすぐに自分のシステムに組み込んでアルゴリズムを試すことができるだけでなく、アルゴリズムの解説を元に自身で実装したい場合にも、実装の参考として使うことができる。

(3)Unityプラグイン

(a)Unityとは

上述のように、アルゴリズムと、それを実装したライブラリの開発を行ったが、これだけではまだ実際にそのまま使ってもらうことができるシステムができたわけではない。そこで、実際に使えるシステムを作り、またアルゴリズムの実用性や有効性を説得力のある形で提示するために、Unityのプラグインという形でアルゴリズムを実装した。

そもそもUnityとは、現在最も注目されているゲームエンジンの一つである。特に3DCGを用いたゲームを作るためのシステムであり、作成したゲームはクロスプラットフォームで動作し、ウェブブラウザや、iOSやAndroidといった携帯端末向けのゲームも作成することができる。また、Unityでは専用の統合開発環境(図14)が提供されており、ゲームプログラマはこの開発環境上で3DCGのシーンの編集などを行う。

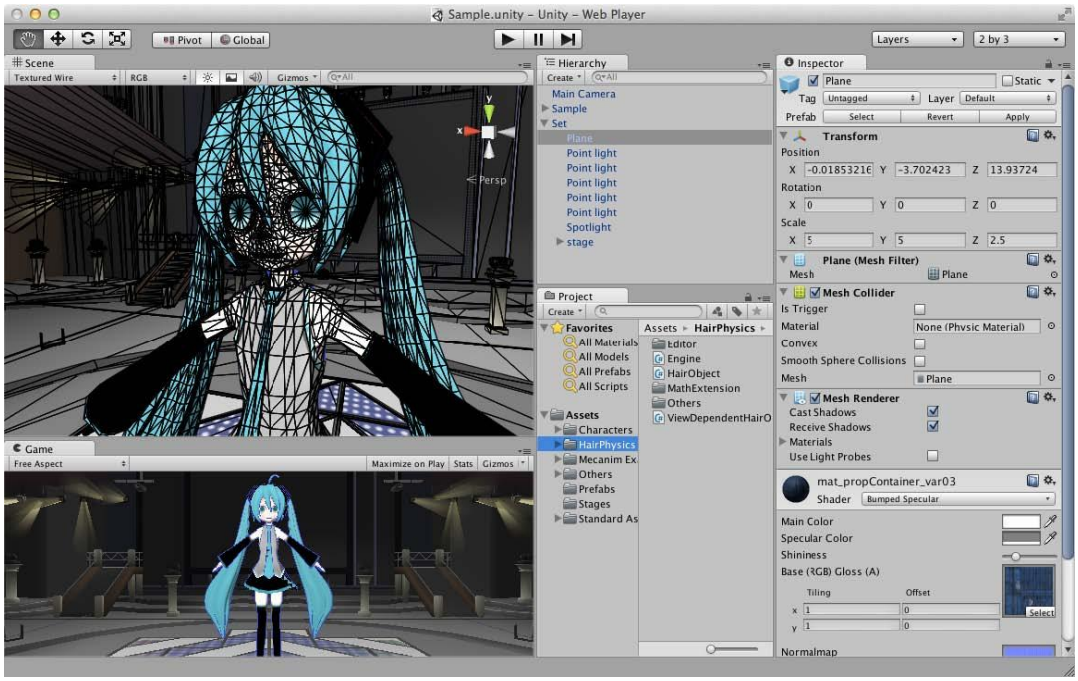

図14：Unityの統合開発環境の画面の例

また、近年Unityが注目されるようになってきた原因の一つとして、ゲーム以外のア

プリケーション開発にもUnityが有効であるという点が挙げられる。Unityで作られるゲームは、本質的にはインタラクティブな3DCGアプリケーションであるので、この範疇に含まれるようなアプリケーション、例えばKinectアプリやARアプリなども作成することができる。また、プロジェクションマッピングなどを用いたメディアアートにもUnityが使われている事例も報告されている。

Unityの特徴として、開発環境自体がプログラマブルだという点が挙げられる。例えば、画面上に新たなウインドウを表示したり、インスペクタに新たな要素を表示したり、ボタンを設置してそこからスクリプトを実行したりするといった拡張が可能である。

UnityにおけるプログラミングはC#またはJavaScriptによって行う。シーン中のオブジェクトは全てGameObjectと呼ばれるクラスのオブジェクトであり、これらのオブジェクトにソースコードをコンポーネントとして付与していくことでゲーム開発を行う。このような考え方はオブジェクト指向ではなくコンポーネント指向と呼ばれる。また、GameObject同士は木構造による関係性を持つ。

(b)Unity上で物理演算をするためのデータ構造の設定

Unityでは様々なキャラクタモデルのファイルフォーマットに対応しているが、FBXと呼ばれるファイルフォーマットでキャラクタモデルを作成し、Unityにインポートして使うのが一般的である。

本プラグインを使用するには、まずFBXファイルをインポートして得られるFBX Objectを用いて図15のような木構造になるようにGameObjectを配置する。ただし、Character Object及びPhysical Objectには空のGameObjectを作成して配置しておく。

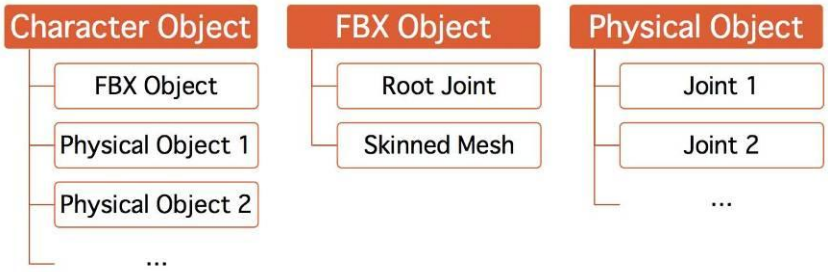

図 15:本プラグインを使用するのにための GameObject の木構造

Character Objectに対しては、Engineというスクリプトコンポーネントを付与する。このスクリプトは物理エンジンの本体であり、他で提案されているダイナミクスが実装されている。

続いてPhysical Objectに対しては、それぞれViewDependentPhysicalObjectというスクリプトコンポーネントを付与する。このスクリプトには本プロジェクトで開発したアルゴリズムなどが実装されている。

最後に、Physical Objectの子にあたるJointに対してはAnimatedParticleというスクリプトコンポーネントと、OrientedParticleというスクリプトコンポーネントを付与する。前者は物理演算結果に対してkinematicな効果を与えるために必要である。後者は他での実装の一部である。更に、衝突を扱う必要がある場合にはUnityによって用意されているRigidbodyコンポーネントとSphere Colliderコンポーネントを付与する必要がある。

(c)データ構造の設定を自動化するスクリプト

上述のようなデータ構造をユーザが手動で設定するのは大変であり、またプラグインの学習コストも高くなってしまうため、このような設定を自動化するようなスクリプトを実装した。このスクリプトは上部のメニューバーから呼び出すことができる(図16)。

図16：上部メニューバーからスクリプトを呼び出す様子

スクリプトを呼び出すと、FBX Objectや揺れものに対応するjoint chainなどを指定する画面が現れ、それらを指定すると、自動でシーンに必要なデータ構造が生成される。

(4)形状編集のためのスケッチインタフェース

本プロジェクトで開発した物理エンジンでは、アーティストの入力として複数のjoint chainの形状を受け取る。そのためには、アーティストはjoint chainの形状を編集する作業が必要になる。

このような形状編集機能は一般的な3DCGオーサリングソフトウェアやUnityにも搭載されてはいるものの、多くはjointを一つ一つ選択し、位置や方向を微調整していく必要がある。このような方法ではjointの個数に比例して作業量が増えるし、何より直感的とは言えない。特に、本プロジェクトのように見る方向は固定した上でその方向から見た形状を編集したい場合にも、そのような方法では頻繁に視点を変えながら位置や方向の微調整をしていく必要があり、アーティストにとって易しいとは言い難い。

そこで、見る方向に応じて揺れものの形状を編集するためのユーザインタフェースとして、スケッチするだけで簡単に揺れものの形状を操作できる機能を開発し、Unityプラグイン上に実装した(図17)。スケッチによる形状編集機能は、見る方向を変える

ことなく一回のストロークで形状を編集できるため、本プロジェクトの目的との相性が良い。

図17：スケッチインタフェースを用いて髪型を編集する例

元々は独自にアルゴリズムを考案して実装したものではあるが、内部的には他で提案されているアルゴリズムに近いものとなっている。そこでは植物に特化した3Dモデリングのためのものとして提案しているが、本プロジェクトではこれを揺れものの joint chainの形状を編集する目的に応用し、また見る方向を変えることなく編集可能であるという利点を利用している。

10. プロジェクト評価

開発期間の 8 割は、実験ばかりやっていたようである。新しいアルゴリズムを試したり、まったく違う応用を試したり。今までのものをすべて止めて、別の応用をしてみたいと言い出したこともあった。

その間、常に応用と具体的なユーザ層を見据えて進めていた。すべては、「こだわり」を効果的に出せるところはどこか、である。終了 2 か月前で、ユーザに提供したい技術と応用が固まった。しかし、それまで想定していた提供の仕方を、まったく変えた。それまでは、アニメーション生成用ソフトへの拡張として考えていたが、プロもアマチュアもそのソフトを使って、その技術の恩恵を受けるようなシーンが想像できなかった。そこで、アニメーション用ではなくて、3D ゲームプラットフォームへの拡張に切り替えた。それまで、そのツールを使用した経験はなかった。そこから 1 月ほどで、細部まで熟知し、うまい抜け道を見つけて、拡張を施し、実用的な速度で動作する実装をした。この能力の高さは特筆すべきである。

開発結果は、非常にわかりやすく説得力のあるシステムが作れたと思う。これを利用したソフトウェアを作るであろうユーザも想像できる。

11. 今後の課題

今回の開発成果が世の中に使いやすいようなパッケージングをして公開することは、すぐにでもやって欲しい。もしかしたら、その技術を効果的に使用した、本格的なアプリまでも、小山君自身が作って見せないと伝わらないかもしれないが、彼ならそれもやれてしまうだろう。

さらにその先のことで、本人は研究者の道に進みたいようであるが、論文や、そのアルゴリズムを紹介するだけの小さなプログラムの公開だけではなく、今回のように、きちっとユーザがその新しい技術を使えるようなレベルにまで落としこんで、実際にユーザが使うシーンが想定できるようなものを提供する、ということを今後も続けていって欲しい。