캐릭터 로코모션을 위한 준절차적 애니메이션 자동화

서론

이 논문은 고도로 유연한 캐릭터 이동의 상호작용 합성을 위한 기술의 프레임워크를 제시한다. 이 시스템은 키 프레임 또는 모션 캡쳐된 보행 및 달리기 사이클의 형태로 일습의 예제 모션들을를 사용합니다. 이 시스템은 설계 시간에 각 동작을 자동으로 분석하며 전체 속도뿐만 아니라 각 발의 충격 및 리프트 오프 시간 등의 파라미터를 추출합니다. 런타임에서 시스템은 캐릭터의 현재 속도와 회전 속도에 따라서 동작들을 먼저 혼합한다, 그리고 나서 발이 지면에 올바르게 발을 디딜 수 있도록 역운동학을 이용하여 다리의 뼈의 움직임을 조절한다. 이 시스템은 제공된 예제 움직임이 무슨 스타일이든 인간과 아무 다리의 양을 가진 인간이 아닌 캐릭터 둘다에게 작동합니다. 그것은 임의의 계단 및 경사를 포함하여 모든 표면의 속도, 방향, 곡면성으로 평면에서 특성 속도와 방향에 대해 만들어진 애니메이션을 조정할 수 있다. 이 논문의 주목할 만한 혁신은 지면에 상대적인 발의 정렬에 관한 중요한 정보를 유지할 수 있는 단일 힐과 발가락 구속조건인 풋베이스의 개념의 도입; 좋은 무게감각, 지역적 운동을 생산하는데 사용할 수 있는 지지 지면 높이의 계산이다. 다리 수와 걸음 걸이 스타일에 관계없이, 모든 캐릭터와 걸음걸이 스타일에 효과가 있는 자연적으로 보이는 발 정렬 계산과 다른 속도로 동작을 보간할 때 바람직한 특성을 갖는 산란 데이터 보간 알고리즘이다.

소개

다음은 이 논문 및 해당 분야의 관련 작업을 이해하는 데 핵심적인 선택된 주요 용어와 개념에 대한 개요입니다.

1.3.1 캐릭터 로코모션

로코모션(운동)은 다리가 달린 인물의 보행이나 달리기와 같은 유기체의 자율 운동이다.

이 작품에서 캐릭터는 3D 컴퓨터 게임이나 가상 세계의 맥락에서 다리가 달린 애니메이션 또는 인간이 아닌 3D 캐릭터로 정의된다. 일반적으로 캐릭터는 아바타로서 플레이어에 의해 제어되거나 인공지능(AI)이 있는 NPC(비플레이어 캐릭터)로 컴퓨터에 의해 제어된다. 일부 게임에서는 키보드의 화살표 키를 누르거나 게임 패드, 조이스틱 또는 유사한 컨트롤러를 사용하여 캐릭터를 다양한 방향으로 걷거나 실행하도록 제어할 수 있습니다. 다른 게임에서는 플레이어가 마우스를 땅에 대고 클릭하면 캐릭터가 그 지점을 향해 걸어갈 수 있으며, 때로는 장애물을 자동으로 찾아갈 수도 있습니다. 어느 쪽이든, 어떤 논리는 플레이어 또는 AI 제어 입력을 캐릭터를 움직이는 데 사용되는 단순한 이동 명령으로 변환하는 데 사용된다.

전형적으로, 캐릭터는 세계에서 위치와 방향을 가진 하나의 단단한 몸체로 추상적으로 표현된다. 시각적으로 캐릭터는 어떤 모양이든 가질 수 있고, 이 모양은 꼼꼼하게 생동감을 느낄 수 있지만, 대부분의 게임 논리에 캐릭터는 여전히 하나의 물건일 뿐이다. 앞서 언급한 이동 명령은 일반적으로 특정 방향으로 힘 또는 속도를 가하여 문자 객체의 위치가 연속적인 프레임 위로 이동하게 함으로써 문자 객체를 이동합니다. 단지 게임이나 가상세계가 다리의 움직임을 이끌고 캐릭터를 앞으로 나아가게 하기 위해 다리 근육의 실제 물리적 시뮬레이션에 신경을 쓰는 경우는 드물다.

이 작품에서 캐릭터는 잠재적으로 복잡한 애니메이션 시각적 표현뿐만 아니라 세계에서 위치와 방향을 가진 단순한 객체로 표현된다고 가정한다. 캐릭터 객체를 이동하는 특별한 방법은 가정되지 않고, 캐릭터 객체의 위치와 방향이 연속 프레임 위로 이동하게 될 뿐이다. 이 작품이 관심을 갖는 캐릭터 로코모션은 캐릭터가 걸어다닐 때나 뛰어다닐 때 캐릭터의 시각적 표현 애니메이션을 보기 좋게 만드는 것이다.

1.3.2 골격 애니메이션

골격 애니메이션은 컴퓨터 애니메이션의 기법이며, 특히 인간과 생물(캐릭터)의 애니메이션이다. 골격 애니메이션에서 캐릭터는 두 부분으로 표현된다: 캐릭터(피부)를 그리는 데 사용되는 표면 표현과 애니메이션에만 사용되는 계층적 골격 집합(회전 관절)이다.

루트 본을 제외하고, 골격 계층 구조의 각 뼈에는 상위뼈와 상위뼈의 변환에 상대적인 3차원 변환이 있습니다. 하위 노드의 전체 변환은 상위 노드와 자체 변환의 산물이며, 예를 들어 허벅지 뼈를 움직이면 하퇴부와 발도 움직인다. 루트 본은 글로벌 스페이스에 변화를 가지고 있다.

오늘날 게임 속 캐릭터들은 일반적으로 제한된 시간 동안 저장된 다수의 애니메이션을 가지고 있으며, 종종 학계에서 모션으로 언급된다. 예를 들어, 한 동작은 캐릭터가 정지 상태(유휴 상태)인 반면, 다른 동작은 캐릭터가 보행 중일 수 있습니다. 동작은 애니메이터에 의해 생성되며 모션 캡처를 기반으로 할 수 있습니다. 각 동작은 골격에 있는 각 뼈의 시간 경과에 따른 움직임을 저장합니다.

1.3.3 걷기와 달리기 사이클

일반적으로 걷기와 달리기를 위한 움직임인 일부 동작은 완벽하게 주기적인 방식으로 구성될 수 있다. 사이클 모션에서 모션의 끝에 있는 골격의 구성은 모션이 반복적으로 재생될 때 부드러운 연속 모션이 되도록 시작 부분의 구성과 일치한다. 걷기와 달리기를 위한 주기적 움직임을 각각 걷기 사이클과 달리기 사이클이라고 한다. 이 작업의 목적상 걷기 사이클과 실행 사이클은 구별되지 않습니다. 여기서는 걷기와 달리기의 차이뿐만 아니라 조깅, 슬쩍하는 동작, 그리고 각 발로 한 걸음씩 걷는 캐릭터를 포함하는 다른 주기적인 동작도 순수하게 스타일리쉬한 것으로 간주된다. 걷기 및 달리기 사이클은 단지 몇 개의 짧은 저장 동작에 기초하여 임의의 지속시간의 걷기 또는 달리기를 가능하게 하기 때문에 널리 사용된다.

1.3.4 애니메이션 블렌딩

애니메이션 캐릭터가 있는 오늘날의 게임 및 기타 실시간 응용 프로그램에서는 애니메이션 블렌딩이 여러 용도로 사용되는 경우가 많습니다. 블렌딩을 사용하면 한 캐릭터의 골격을 하나의 저장된 모션으로 이동할 수 있을 뿐만 아니라 여러 모션으로 동시에 이동할 수 있습니다. 주어진 시간에 각 모션에는 가중 평균에 사용되는 가중치가 주어지며, 이 가중치는 결합된 블렌딩 모션의 생성에 사용됩니다. 이러한 가중치는 시간이 지남에 따라 지속적으로 변경될 수 있습니다.

혼합을 사용하면 한 동작에서 다른 동작으로 전환할 때 캐릭터의 이동이 원활하게 이루어질 수 있습니다. 현재 동작의 재생을 갑자기 중단하고 다음 동작을 시작하기보다는 현재 동작의 가중치를 1에서 0으로 점차 줄이는 동시에 다음 동작의 영향력이 0에서 1로 증가할 수 있다. 이 기술을 크로스페이딩이라고 한다.

애니메이션 블렌딩의 또 다른 용도는 저장된 모션과 다른 속성을 가진 모션의 합성이다. 예를 들어, 캐릭터는 앞으로 직진하기 위한 저장된 걷기 사이클과 특정 회전각으로 오른쪽으로 급격히 회전하기 위한 다른 저장 걷기 사이클이 있을 수 있습니다. 이 두 동작을 각각 주어진 무게와 혼합함으로써, 덜 날카로운 회전각을 가질 수 있는 새로운 걷기 사이클이 합성될 수 있다. 학계에서는 함께 혼합된 동작을 예시 동작이라고 부르기도 한다.

1.3.5 키 타임

특정 동작 합성 방법은 제공된 동작의 특정 속성을 알아야 할 수 있다. 키 타임은 모션에서 특정 중요한 이벤트의 시간을 나타냅니다. 예를 들어, 걷기 및 달리기 사이클에는 toe-off와 heel-strike와 같은 주요 시간에 주석을 달아 발이 땅에서 올라가고 지면에 다시 부딪히는 시간을 표시할 수 있다. 모션의 키 타임은 애니메이터가 수동으로 주석을 달 수 있다. 경우에 따라 휴리스틱을 사용하여 자동으로 키 시간을 식별할 수 있습니다.

모션 분석

여기서 고려하는 모션 사이클은 주기적 보행 또는 실행 사이클로, 캐릭터가 일정한 속도로 그 아래 움직일 수 있는 지면과 함께 제자리를 걷는 것이다. 캐릭터는 평평한 수평 지면을 걷는 것으로 가정됨, 모션 사이클에서 캐릭터의 각 다리는 정확히 한 단계를 밟는다. 구현된 시스템에서 수평적 접지를 가정하기 위해 선택되었다. 이러한 가정은 분석을 단순화합니다. 여기에 제시된 것보다 몇 가지 더 정교한 분석 방법으로, 모든 경사면의 표면에서의 움직임을 분석할 수 있었고, 각 움직임의 기울기도 자동으로 탐지될 수 있었다. 그러나 경사면이 다른 표면에 대한 예제 모션을 제공하면 예제 모션의 양이 기하급수적으로 증가할 수 있습니다. 런타임으로 생성된 모션에 대한 입력으로 많은 양의 예제 모션을 사용하는 것이 본 논문에서 취한 접근 방식과 직접적으로 반대되는 것은 아니지만, 그것은 주요 초점은 아니므로 구현의 단순성을 위해 비수평면에서의 모션은 여기서 무시되었다. 캐릭터가 걷다가 뒤돌아서는 회전 동작도 마찬가지다. 이 역시 약간 더 정교하게 분석될 수 있지만, 설명된 방법과 구현된 시스템에서는 다루지 않는다.

1. 모션 주기와 다리 주기들

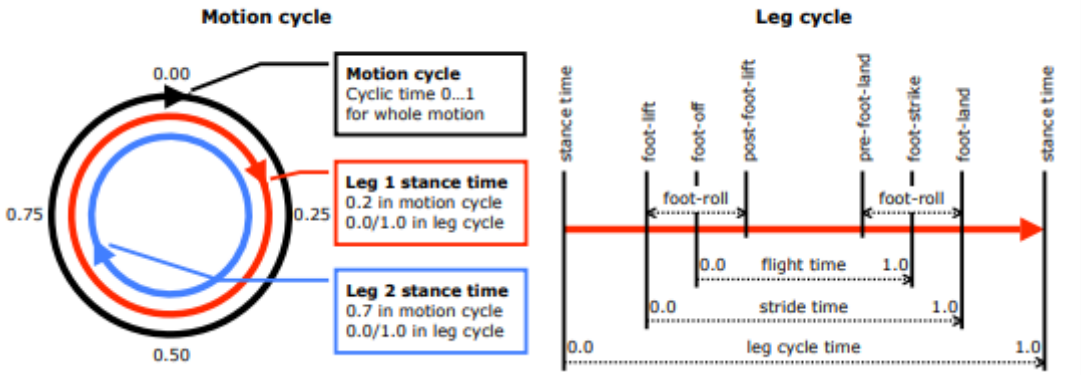
여기서 고려되는 동작 사이클은 주기적인 보행 또는 달리기 사이클[(1)]로, 캐릭터가 일정한 속도로 아래에서 움직일 수 있는 지면과 함께 그 지점을 걷고 있다. 캐릭터는 평평한 수평 지면을 걷는 것으로 가정되며, 모션 사이클에서 캐릭터의 각 다리는 정확히 한 단계를 밟는다. 구현된 시스템에서 수평적 접지를 가정하기 위해 선택되었다. 이러한 가정은 분석을 단순화합니다. 여기에 제시된 것보다 몇 가지 더 정교한 분석 방법으로, 모든 경사면의 표면에서의 움직임을 분석할 수 있었고, 각 움직임의 기울기도 자동으로 탐지될 수 있었다. 그러나 경사면이 다른 표면에 대한 예제 모션을 제공하면 예제 모션의 양이 기하급수적으로 증가할 수 있습니다. 런타임으로 생성된 모션에 대한 입력으로 많은 양의 예제 모션을 사용하는 것이 본 논문에서 취한 접근 방식과 직접적으로 반대되는 것은 아니지만, 그것은 주요 초점은 아니므로 구현의 단순성을 위해 비수평면에서의 모션은 여기서 무시되었다. 캐릭터가 걷다가 뒤돌아서는 회전 동작도 마찬가지다. 이 역시 약간 더 정교하게 분석될 수 있지만, 설명된 방법과 구현된 시스템에서는 다루지 않는다.

시간은 사이클의 시작을 나타내는 0.0과 끝을 나타내는 1.0의 값으로 동작 사이클에 대해 순환적으로 정의되며, 이후 사이클이 다시 시작됩니다. 모션 사이클은 키 프레임 또는 준비된 모션 캡처 애니메이션을 기반으로 합니다. 모션의 일부가 출발점이 되어야 하는 규약이 없기 때문에 이것은 사실상 임의적이다.

---

(1) 걷기 사이클과 달리기 사이클 사이에는 어떤 구별도 없다. 내가 아는 바로는 이 둘을 포괄하는 일반적인 용어는 없다.

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화



(a) 동작 사이클에 상대적인 다리 사이클. 이 예는 모션 사이클에서 서로 반대되는 스탠스 시간을 가진 두 다리를 보여줍니다.

(b) 다리 주기의 주요 시간. 각 레그에는 레그 사이클이 있으며, 레그 사이클 시간은 스탠스 시간에는 0.0에서 스탠스 시간에는 1.0이다.

그림 4.1: 모션 주기와 다리 주기들

휴머노이드 캐릭터에게 이것은 예를 들어 왼발이 아래로 내려간 상태, 오른발, 또는 두 발이 땅에 닿은 상태일 수 있지만 아무것도 가정할 수 없다.

다리의 개수는 얼마든지 있을 수 있고, 다른 다리의 움직임은 서로 완전히 독립적으로 분석됩니다. 각 다리에 대해 휴리스틱을 사용하여 Stance Time을 찾습니다:

Stance Time : 다리가 지면에 가장 단단하고 중립적으로 서 있을 때의 동작 주기에서의 시점.

전체적인 움직임 사이클 외에도, 각 레그에는 자체적인 레그 사이클이 있다. 이러한 다리 사이클의 시작과 끝은 각 다리의 스탠스 시간으로 표시되므로 전체 동작 사이클과 달리 다리 사이클의 시작 시간은 전혀 임의적이지 않다. 다리 사이클과 동작 사이클 사이의 관계는 그림 4.1a에서 확인할 수 있다.

1.1 레그 사이클들의 키 타임

절차적 애니메이션 방법이 제공된 예제 모션을 올바르게 처리하려면 해당 모션의 특정 특성을 알아야 할 수 있습니다. 키 타임은 동작에서 특정 중요한 이벤트의 시간(예: 각 발이 지면에서 들어올리는 시간)을 나타냅니다.

Toe-off와 heel-strike라는 키 타임을 이용해 발이 땅에서 들어올려 다시 땅을 치는 시간을 각각 표기하는 대신, 다르지만 비슷한 키 타임을 사용하는 것이 이번 작업이다. 여기서 설명하는 방법은 휴머노이드 캐릭터 전용이 아니기 때문에, 해당 캐릭터의 해부학과 운동 스타일에 대한 최소한의 가정만 하도록 개념이 일반화되었다. 특히, 여기서 설명하는 시스템은 발 밑의 두 지점을 대략적으로 서로 반대 방향으로 식별할 수 있다고 가정하며, 편의상 다음에서 힐과 발끝 지점으로 언급할 것이다. 그러나 시스템은 이 두 지점 중 어느 지점이 먼저 지면에 닿아 보행 사이클에서 지면을 이탈하는지 가정하지 않습니다. 따라서 toe-off는 foot-off로 일반화되고 heel-strike 시간은 foot-strike로 일반화된다. Foot-off, foot-strike 및 기타 모든 레그 사이클 키 시간은 0에서 1까지의 레그 사이클 시간에 상대적이다(그림 4.1b 참조).

Foot-off : 발이 땅에서 완전히 들어올릴 때의 다리 주기 시간.

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화

Foot-strike : 발이 땅에 처음 부딪힐 때 다리 주기의 시간

두 개의 추가 키 시간을 사용하여 발이 지면에 거의 평평하게 서 있는 시간을 나타냅니다.

Foot-lift : 발이 땅에 더 이상 평평하게 서 있지 않도록 발의 일부분이 들기 시작하는 다리 사이클의 시간.

Foot-land : 발이 다시 땅에 평평하게 서 있을 때 다리 주기의 시간.

Foot-strike와 foot-land 사이의 시간 간격은 지면에 착지할 때 foot-roll의 지속 시간을 결정하는 데 사용된다. 다만 착지 시 뒤꿈치와 발가락이 동시에 땅에 닿으면 foot-strike과 foot-land 시간이 같을 수 있다. 너무 짧은 기간의 foot-roll을 방지하기 위해 pre-foot-land라고 하는 추가 키 시간이 사용됩니다. 일반적으로 pre-foot-land 시간은 foot-strike 시간과 동일하지만, foot-strike와 foot-land 사이의 시간 간격이 주어진 한계치보다 작을 경우, pre-foot-land 시간은 대신 해당 임계값을 뺀 시간이다. 마찬가지로, foot-off 시간보다 크거나 같은 post-foot-lift라고 불리는 키 타임이 사용된다.

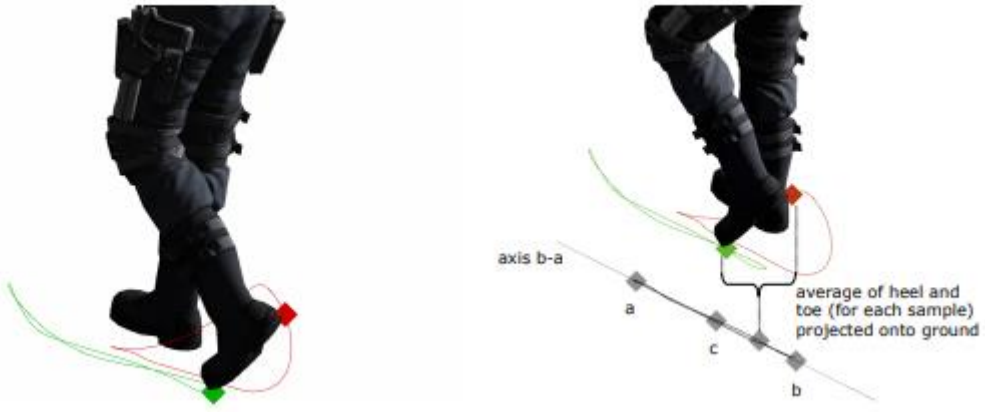
Post-foot-lift : foot-off 또는 foot-lift + 원하는 최소 foot-roll 지속시간 중 더 큰 것이 있다.

Pre-foot-land : foot-strike 또는 foot-land와 동일함 - 원하는 최소 foot-roll 지속 시간(어느 쪽이든 더 작은 길이든).

각 레그에 대해 모션 분석의 일부로 사용자의 개입 없이 스탠스 시간과 모든 레그 사이클 키 시간을 분석한다.

2. 발 움직임 분석

움직임 분석의 첫 번째 단계는 각 다리에 대해 다리 주기의 자세 시간과 키 시간, 발 밑의 궤적 및 발 밑의 방향과 길이를 결정하는 것이다. 먼저 힐과 발가락의 궤적은 모션 사이클 내내 샘플링되며(그림 4.2a 참조) 다른 모든 정보는 여기에서 도출된다.



(왼쪽) 발의 뒤꿈치와 발가락의 궤적은 동작 주기 동안 뒤꿈치와 발가락 위치를 샘플링하여 구한다.

(오른쪽) axis = b - a는 지면에 투영된 힐 및 토우 지점의 평균 궤적을 기준으로 추정됩니다. 평균 점 c가 계산됩니다. a는 c에서 가장 먼 점이고 b는 a에서 가장 먼 점입니다.

그림 4.2: 힐 및 토우 궤적 및 이동 축 추정

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화

2.1 이동 축

힐과 토우 궤적이 주어지면 이동 축의 추정치를 계산할 수 있다. 힐 궤적과 발가락 궤적의 평균은 각 표본의 힐과 발가락 위치의 평균을 구한다.

$$\mathbf{middle}_s = \frac{\mathbf{heel}_s + \mathbf{toe}_s}{2} \qquad (4.1)$$

그런 다음 이 궤적이 지면 위로 투사됩니다. 점 C는 투영된 궤도에 있는 모든 점의 평균입니다. 투영된 궤도에서 점 A는 점 C에서 가장 먼 점이고, 점 B는 점 A에서 가장 먼 점인 투영된 궤도에서 식별된다. 이제 벡터 축 = b - a는 움직임 축의 좋은 추정치입니다. 그림 4.2b를 참조하십시오. 이 시점에서 캐릭터가 어느 방향으로 움직이는지, 또는 어느 속도로 움직이는지는 여전히 알려져 있지 않다.

2.2 스탠스 타임

주어진 움직임의 축을 추정하면, 그 축을 따라 다리의 흔들림을 측정할 수 있으며, 휴리스틱을 사용하여 다리가 지면에 가장 단단하고 중립적으로 서 있을 때의 동작 주기 시간인 스탠스 시간을 찾을 수 있다. 스탠스 시간을 결정하기 위해, 각 표본의 힐과 발가락 높이(그림 4.3a 참조)와 이동 축을 따라 발 중간 위치(방정식 4.1에 정의된)를 기반으로 비용 값이 계산된다(그림 4.3b 참조).

(왼쪽) 뒤꿈치와 발가락 높이가 다리 주기 전체에 걸쳐 표시되는 곡선. 이 위에 균형 곡선이 표시됩니다. 밸런스 값은 특정 시간에 발의 어느 부분이 지면에 더 가까운지를 나타내므로 발을 회전할 때 피벗 포인트 역할을 해야 합니다.

(오른쪽) 추정된 움직임의 축을 따라 뒤꿈치와 발가락의 스윙이며, 그래프에서 수직 검은색 선은 다리 주기의 자세 시간을 나타낸다. 캐릭터는 왼쪽 다리의 스탠스 시간에 굳은 상태로 여기에 묘사됩니다. 이 다리가 지면에 가장 단단하고 중립적으로 서 있는 주기의 지점.

그림 4.3: 캐릭터의 수직축을 따라 그리고 추정된 움직임 축을 따라 힐과 발가락의 움직임.

비용이 가장 낮은 샘플의 시간이 사용됩니다:

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화

$$cost_s = \frac{\max(heelheight_s, toeheight_s)}{\alpha} + \frac{\left|(\mathbf{middle}_s - \mathbf{c}) \cdot \hat{\mathbf{axis}}\right|}{\beta} \qquad (4.2)$$

위치 α와 β의 파라미터는 스탠스 타임의 발이 가능한 한 낮거나 스윙의 중간에 가까운 것이 더 중요한지 여부를 제어하기 위해 조정될 수 있다. 구현된 시스템에서 α는 모든 표본에서 가장 높은 힐 또는 토우 높이로 설정되었고 β는 $\|b-a\|$으로 설정되었다.

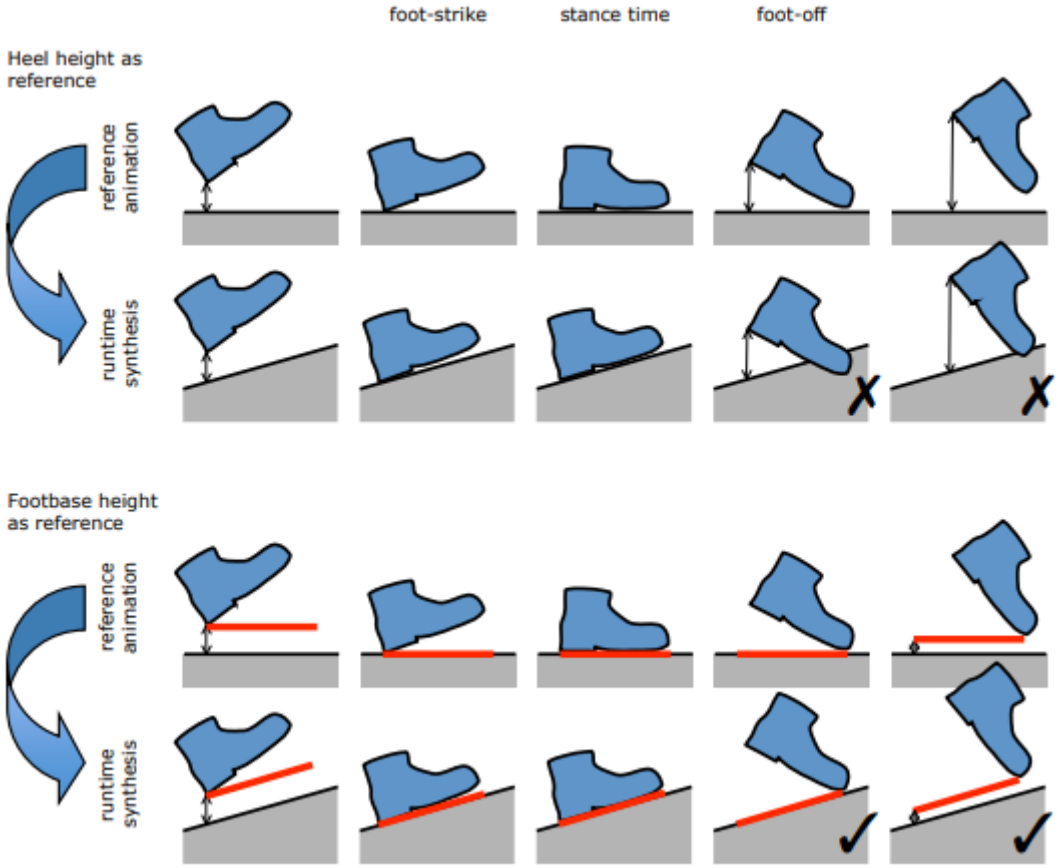
스탠스 시간이 주어지면, 사이클에서 해당 시간에 발 방향이 저장됩니다. 발 방향은 지면 위에 투영된 벡터 toe - heel로, 정상화된 다음 발의 길이에 곱한 것입니다.

$$\mathbf{footdirection} = \hat{\mathbf{d}} \cdot footlength \qquad (4.3)$$

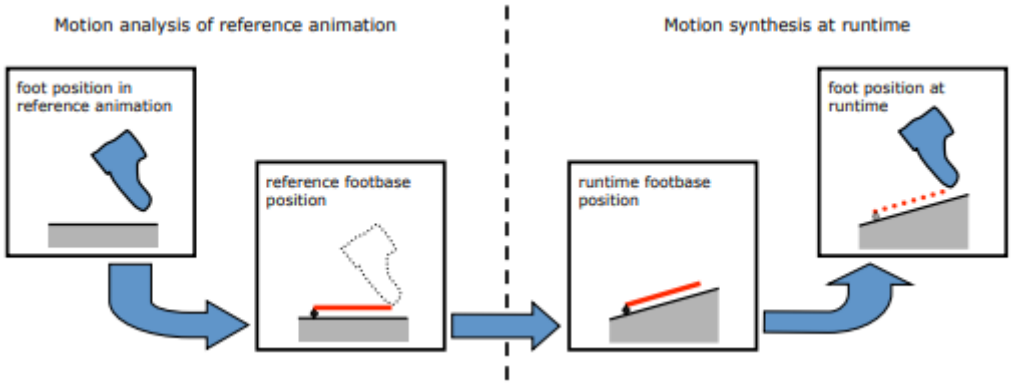
$$\mathbf{d} = (\mathbf{toe} - \mathbf{heel}) \parallel \mathbf{ground} \qquad (4.4)$$

2.3 균형과 발밑

런타임에 발의 움직임을 절차적으로 조정할 때 지면에 상대적인 발의 정렬이 원래 입력 애니메이션의 해당 정렬과 항상 일치하지는 않을 수 있다. 경사진 표면을 걸을 때 또는 한 걸음 위 또는 아래로 걸을 때 적절한 풋 롤을 만들기 위해 발의 정렬을 절차적으로 조정해야 하는 경우가 많습니다. 이러한 경우 발의 어느 부분이 발이 회전하는 피벗 포인트로 사용되는지 고려하는 것이 중요합니다.



(a) 풋 얼라인먼트 비교. 상단에는 힐 높이를 기준으로 사용하는 것이 절차적 런타임 모션 합성에서 어떻게 바람직하지 않은 결과를 가져올 수 있는지 설명되어 있다. 토우 높이를 기준으로 사용하면 유사한 문제가 발생합니다. 이러한 문제를 해결하기 위해 풋베이스가 참조로 사용됩니다.



(b) 기준 풋베이스 위치는 기준 애니메이션에서 발의 위치에서 파생됩니다. 런타임에 발의 위치는 런타임 풋베이스 위치를 기반으로 합니다.

그림 4.4: 풋베이스는 발과 길이가 동일하고 발 아래 지면에 항상 평행한 선 세그먼트입니다. 발뒤꿈치나 발가락이 지면에 더 가까운지에 따라 발바닥에 각각 발꿈치나 발가락으로 닿는다.

항상 정확한 결과를 얻으려면, 회전은 항상 발의 어느 부분이 지면에 가까운지 주위에 있어야 한다. 그림 4.4a를 참조하십시오. 이것을 이루기 위해 저는 발바닥의 개념을 소개합니다. 풋베이스는 발과 길이가 동일하고 발 아래 지면에 항상 평행한 선 세그먼트입니다. 발뒤꿈치나 발가락이 지면에 더 가까운지에 따라 발뒤꿈치나 발가락으로 발바닥에 각각 닿는다.

지정된 시간에 발의 어느 부분이 더 낮은지(따라서 지면에 더 가까운지)를 측정하기 위해 각 표본에 대해 균형 값이 계산됩니다. 밸런스 값이 0에 가까우면 힐이 낮아지고 1에 가까우면 발가락이 낮아지는 것을 나타냅니다(그림 4.3왼쪽 참조). 샘플의 잔액은 다음과 같이 계산됩니다.

$$balance_s = \frac{\arctan\left(\frac{heelheight_s - toeheight_s}{footlength}\alpha\right)}{\pi} + 0.5 \qquad (4.5)$$

α는 민감도 파라미터를 나타내며, $\alpha < 1$은 0.5의 중간값에 대한 편향을, $\alpha > 1$은 0과 1의 극단값으로 치우침을 생성한다. 구현된 시스템에서는 0과 1에 대한 높은 편향이 필요했기 때문에 $\alpha = 20$ 값이 사용되었습니다.

표본의 균형 값이 주어지면 발바닥의 위치를 도출할 수 있다. 발꿈치에 해당하는 발 밑면의 끝은 그 위치로 정의된다. 지면을 따라 발바닥의 방향은 방정식 4.3에 제시된 발 방향에 기초한다.

$$\mathbf{footbase}_s = \mathbf{heelpos}_s \cdot balance_s + (\mathbf{toepos}_s - \mathbf{footdirection}) \cdot (1 - balance_s) \quad (4.6)$$

기준 발밑 위치는 위에서 보는 바와 같이 발의 위치와 정렬에 의해 결정됩니다. 런타임에 발바닥은 발 밑 지면과 평행하도록 연속적으로 조정되며 발의 위치는 발바닥과 발바닥의 위치에 따라 결정된다(그림 4.4b 참조). 이 내용은 7장에서 자세히 다룰 것이다.

각 샘플의 풋베이스 위치가 저장되고 스탠스 시간과 관련된 샘플 위치를 스탠스 위치라고 합니다.

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화

Stance Position: 발이 지면에 가장 단단하고 중립적으로 서 있는 스탠스 시간에 몸을 기준으로 한 발 받침대의 위치.

2.4 키 타임

자세 시간을 알고 발꿈치, 발가락 및 발바닥의 궤적을 사용하여 다리 주기의 키 타임(제4.2.1절 참조)을 찾을 수 있다. 그림 4.5에는 분석의 예가 나와 있지만, 세부 사항은 캐릭터와 보행 또는 주행 스타일에 따라 크게 좌우된다는 점에 유의하십시오. 주요 시간을 찾기 위해 궤적의 몇 가지 특성을 분석한다.

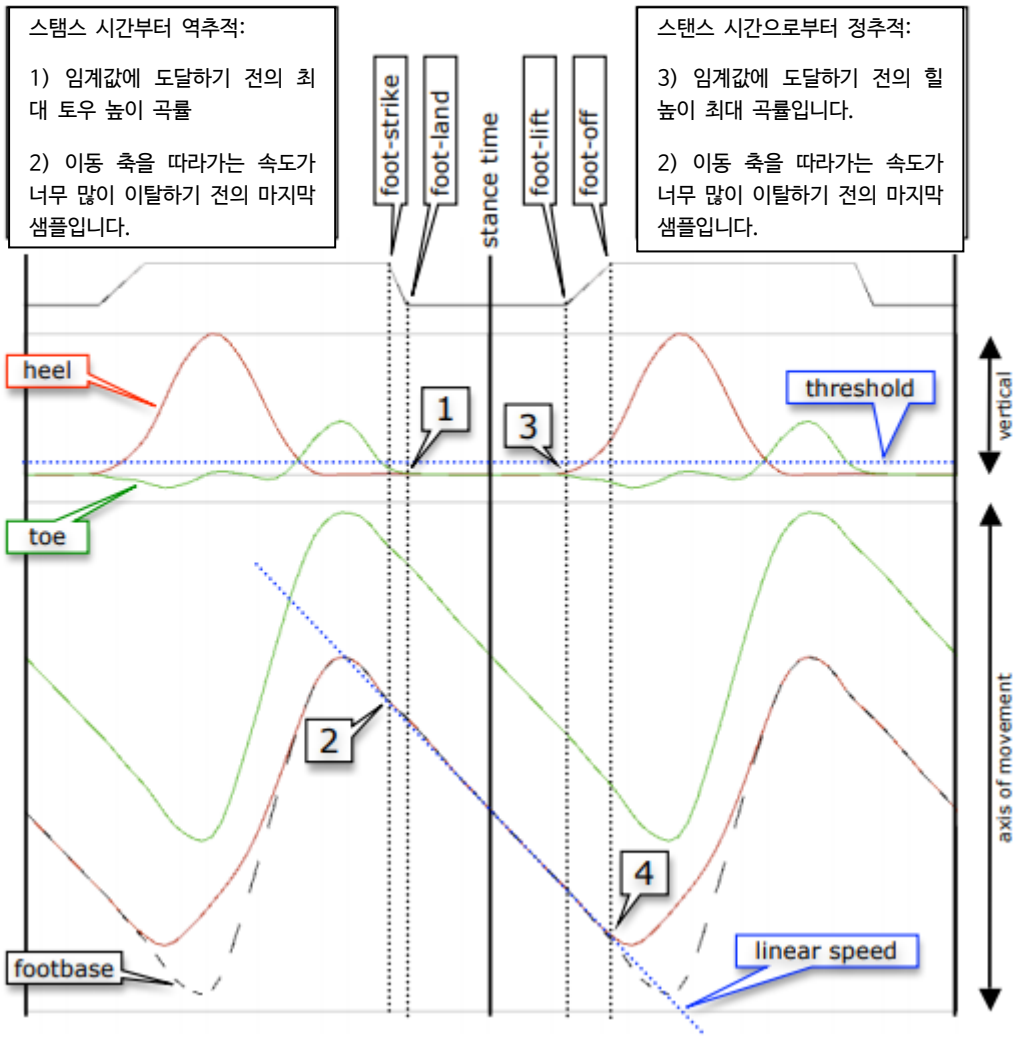


그림 4.5 : 키 타임 계산 예제

- 뒤꿈치 궤도의 수직 구성부품은 정지 시간에서 전방으로 추적된다. 힐이 특정 임계값 고도에 도달하기 전에 힐 궤적이 최대 곡률을 갖는 샘플의 시간은 heel-off 시간으로 일시적으로 저장된다.
- Toe-off 시간을 찾기 위해 토우 궤적의 수직 구성 요소에 대해서도 동일한 추적이 수행되며, 이 시간은 일시적으로 저장됩니다.

- 발 밑면의 수평 구성요소는 스탠스 시간에서 앞으로 추적됩니다. 발이 스탠스 시간에 지면에 놓여 있고, 캐릭터에 비해 지면이 일정한 속도로 움직이기 때문에, 풋베이스는 스탠스 시간 전후 최소 일정 기간 동안 일정한 속도로 움직이고 있다. 이 속도가 계산됩니다. 풋베이스의 수평 이동이 이 일정한 속도에서 너무 많이 벗어나는 시점이 linear-speed-end 시간으로 일시적으로 저장됩니다.

이러한 임시 시간 값이 주어지면 foot-lift, foot-off 및 post-foot-lift의 키 시간이 계산됩니다.

$$\text{foot-lift} = \min(\text{heel-off}, \text{toe-off}, \text{linear-speed-end}) \quad (4.7)$$
$$\text{foot-off} = \max(\text{heel-off}, \text{toe-off}) \quad (4.8)$$
$$\text{post-foot-lift} = \max(\text{foot-lift} + \alpha, \text{foot-off}) \quad (4.9)$$

여기서 α는 한 사이클의 전체 지속시간에 상대적인 비율로 표현되는 풋롤의 최소 지속시간이다. 구현된 시스템에서는 α = 0.2가 사용되었습니다.

나머지 세 개의 키 타임은 완전히 유사한 방식으로 계산되지만, 역방향은 예외입니다. 임시 시간 값 heel-strike, toe-strike 및 linear-speed-begin은 전방이 아닌 정지 시간 및 후방에서 궤적을 추적하여 확인할 수 있다. 이러한 임시 시간 값이 주어지면 foot-land, foot-strike 및 pre-foot-land 키 시간이 계산됩니다.

$$\text{foot-land} = \max(\text{heel-strike}, \text{toe-strike}, \text{linear-speed-begin}) \quad (4.10)$$
$$\text{foot-strike} = \min(\text{heel-strike}, \text{toe-strike}) \quad (4.11)$$
$$\text{pre-foot-land} = \min(\text{foot-land} - \alpha, \text{foot-strike}) \quad (4.12)$$

6개의 키 타임은 영구적으로 저장되며 추가 분석과 런타임에 모두 사용됩니다.

2.5 발걸음 방향과 보폭

다리 주기의 주요 시간을 알 수 있는 경우, 보폭의 방향과 길이는 foot-strike과 foot-off 사이의 발바닥이 이동한 거리(캐릭터 공간)를 측정하여 사소하게 계산할 수 있으며, 벡터 길이를 두 키 시간 사이의 시간 범위로 나눌 수 있다.

$$\mathbf{displacementvector} = \mathbf{footbase}_{footoff} - \mathbf{footbase}_{footstrike} \quad (4.13)$$
$$stridelength = \frac{\|\mathbf{displacementvector}\|}{footoff - footstrike} \quad (4.14)$$
$$\mathbf{stridedirection} = -\hat{\mathbf{displacementvector}} \quad (4.15)$$

보폭 방향은 변위 벡터의 음의 방향이다. 예를 들어, 발이 foot-strike에서 foot-off로 캐릭터 공간에서 뒤로 움직인다면, 이는 캐릭터가 앞으로 걸어가고 있다는 것을 의미한다.

3. 모션 사이클 속성 결정

각 다리에 대해 알려진 보폭 길이와 방향을 사용하여, 움직임 주기의 전체 방향과 거리는 단순한 평균에 의

해 발견되며, 속도는 거리를 움직임 주기의 지속 시간으로 나누어서 발견됩니다:

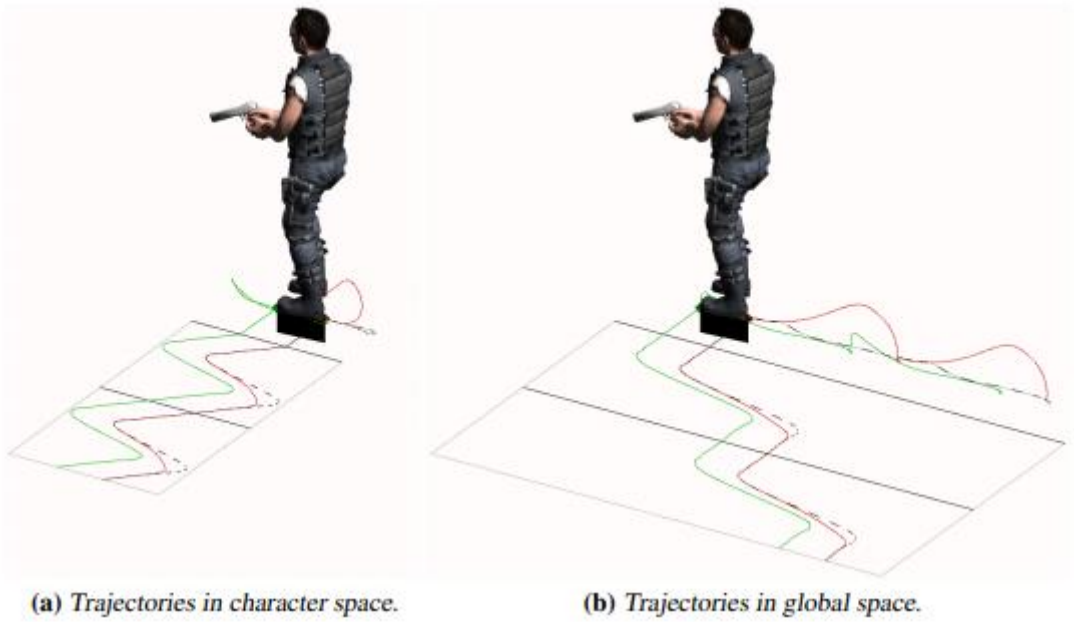
$$cycledistance = \frac{\sum_{l=1}^{legs} stridelength_l}{legs} \quad (4.16)$$

$$\mathbf{cycledirection} = \frac{\sum_{l=1}^{legs} \mathbf{stridedirection}_l}{legs} \quad (4.17)$$

$$cyclespeed = \frac{cycledistance}{cycleduration} \quad (4.18)$$

평균이 그리 정교하지는 않지만, 실제로는 잘 작동한다. 고품질의 움직임의 경우 분석된 길이와 각 다리의 보폭 방향은 어쨌든 동일해야 한다. 이론적으로, 다른 다리의 약진은 캐릭터가 동작 주기 동안 일정한 속도를 유지하지 않는 경우에만 달라질 수 있으며, 이 작업에서는 모든 동작 주기가 일정한 속도를 갖는다고 가정한다. 자동 분석의 부정확성 또는 예시로 인한 움직임의 품질 저하로 인해 다른 보폭 길이 또는 방향이 발생하는 경우, 평균은 오류가 최소로 보이도록 다리 사이에 분산되도록 보장한다. 예를 들어, 제공된 모션 사이클이 한쪽 다리를 다른 쪽 다리보다 더 긴 스텝을 취하는 경우, 런타임에 결과는 한 발이 지면에 있는 동안 뒤로 약간 미끄러지는 반면 다른 발은 약간 앞으로 미끄러지는 것입니다. 이것은 움직임이 있는 그대로 사용된다고 가정하고 발이 땅에 있는 동안 고정 상태를 유지하도록 강제하기 위한 어떠한 조치도 취하지 않는다. 구현된 시스템에서는 이러한 기능을 선택적으로 켤 수 있습니다.

3.1 정규화된 발밑 공간 궤적



(a) *Trajectories in character space.* (b) *Trajectories in global space.*

그림 4.6 : 캐릭터 공간의 힐(빨간색), 토우(녹색) 및 풋베이스(점선 검은색)에 대한 궤적 및 전역 공간.

이 시점에서 각 발의 발바닥 궤적은 캐릭터 공간에서 알려져 있다. 타임에 쉽게 사용하기 위해서는, 풋베이스 궤적이 표준화된 형태로 변환되어야 하며, 여기서 보폭은 항상 위치 (0,0,0)에서 (0,1,0)로 z축을 따라 이동한다. 첫째, 글로벌 공간의 궤적은 먼저 사이클 지속 시간 동안 사이클 거리에 의해 역주기 방향으로 이동하는 기준점에 상대적인 풋베이스(아직도 캐릭터 공간에 있음)를 추적함으로써 발견된다(그림 4.6 참조).

이 궤적은 항상 z축을 따라 있는 회전 Q에 의해 회전된다.

$$cycletime_s = time_s - stancetime_s \quad (4.19)$$

$$\mathbf{reference}_s = \mathbf{stanceposition} - (cycledistance \cdot \mathbf{cycledirection} \cdot cycletime_s) \quad (4.20)$$

$$\mathbf{normalizedfootbase}_s = Q \cdot (\mathbf{footbase}_s - \mathbf{reference}_s) \quad (4.21)$$

여기서 Q는 벡터 사이클 방향을 벡터(0,0,1)로 회전시키는 데 필요한 회전입니다. 이후, 정규화된 보폭의 길이가 항상 1이 되도록 하기 위해 정규화된 보폭의 z 성분을 주기 거리로 나눈다.

❖ 결과

모션 분석을 위해 설명된 방법은 다양한 캐릭터와 모션에 사용되었습니다(자세한 내용은 8장 참조). 분석의 가장 어려운 부분은 정확한 키 타임의 식별입니다. 기술된 휴리스틱은 일반적으로 런타임에 준절차 애니메이션 방법에 의해 사용될 때 좋은 결과를 낼 수 있을 만큼 정확한 키 시간을 찾는 데 유용하지만, 자세히 조사했을 때 확인된 키 타임은 때때로 인간이 적절한 순간으로 선택했을 시간에서 몇 프레임 떨어져 나타난다. 키 타임의 정확한 분석에 영향을 미치는 가장 결정적인 요인은 힐과 토우 궤적 곡선의 곡률을 계산하는 데 사용되는 정확한 방법이다. 이 방법은 힐과 발가락이 지면에서 들어 올려 땅에 착지하는 것으로 식별되는 시점을 결정한다. 노이즈나 부정확성으로 인한 작은 곡률 변화를 배제하기 위해 곡률을 측정하기 전에 어떤 필터링을 적용해야 하는가 하는 문제가 있으며, 굽이나 발가락이 여전히 지면에 가까운 곳에서 곡률 변화를 가장 잘 보장하는 방법에 대한 문제가 있다. 구현된 시스템의 이러한 세부 정보는 현재 시행착오를 기반으로 합니다. 이 문제에 대한 추가 연구는 캐릭터와 애니메이션의 선택이 모두 변수에 대한 다양한 방법과 값으로 테스트되고, 어떤 알고리즘과 변수 값이 키 티임들의 가장 일관된 고품질 식별을 산출하는지 결정하기 위해 구조화된 방식으로 검사 및 등급을 매기는 보다 엄격한 테스트를 수행할 수 있다.

성공적인 자동 동작 분석은 입력으로 사용되는 동작 예제에 대한 특정 가정에 기초한다. 방법에 의해 오류가 거의 없거나 전혀 없이 분석되려면 지정된 모션이 몇 가지 전제조건까지 살아 있어야 합니다:

- 동작은 주기적이어야 하며 각 다리가 정확히 한 단계를 거쳐야 한다.
- 캐릭터의 발은 땅에 있을 때 가장 낮은 위치에 있어야 합니다. 즉, 발이 지면과 교차하지 않는 것이 바람직하다. 이것은 당연하게 여겨질 수 있는 것으로 보일 수 있지만, 이상적인 품질보다 낮은 특정 시험 동작은 동작 주기의 특정 부분 동안 지면과 눈에 띄는 교차점을 보여주었다. 분석 방법은 분석된 모션에서 그러한 결점을 상당 부분 허용할 수 있지만 분석 시 오류 발생 가능성이 증가한다.
- 지면과 접촉할 때 발은 특성에 비례하여 일정한 속도로 움직여야 한다. 게다가, 모든 발은 같은 속도로 움직여야 합니다. 그들은 지면과 접촉에 있는 것을 예를 들어, 캐릭터를 앞으로 일정한 속도로 보행을 위해 캐릭터의 발 모두 거꾸로 그 같은 속도로 이동해야 한다. 이 요구 사항에 부합하지 않는 동작은 원래의 조정되지 않은 상태에서 발 미끄러짐을 나타낼 수 있으며, 자동화된 분석 방법은 예시 애니메이션이 발 미끄러짐을 가지고 있지 않다는 가정에 기초한다. 다시 말해, 이 방법은 이 부분에 대해 일정량의 오류를 허용할 수 있지만 속도가 일관되지 않을수록 분석에서 오류의 위험이 높아집니다.

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화

일반적으로 사전 요구 사항은 정리된 모션 캡쳐로 만들어진 보행 또는 실행 사이클이나 숙련된 애니메이터가 수동으로 수행하는 방식으로 쉽게 충족됩니다. 그러나 처음부터 눈에 보이는 아티팩트로 인해 낮은 품질의 걷기 또는 실행 사이클이 항상 전제 조건을 충족시키지는 못합니다.

❖ 모션 분석 요약

걷기 및 달리기 사이클은 캐릭터 공간에서 캐릭터가 일정한 속도로 그 아래에서 움직일 수 있는 지면과 함께 그 자리를 걷는 주기적인 움직임이다. 캐릭터는 평평한 수평 지면을 걷는 것으로 가정되며, 모션 사이클에서 캐릭터의 각 다리는 정확히 한 단계를 밟는다. 각 레그에 대해 해당 레그가 한 단계 도약하는 방법과 관련하여 여러 가지 속성이 저장됩니다. 분석은 주로 힐과 발가락의 궤적 및 여기서 도출할 수 있는 정보에 기초한다. 다리 주기는 다리에 대해 정의되며, 다리는 지면에 단단하게 서 있고(스탠드 시간), 발을 떼고(foot-lift, foot-off, post-foot-lift) 지면에 다시 착지(pre-foot-land, foot-strike, foot-land)와 같은 특별한 이벤트가 발생할 때 다리 주기의 시간을 여러 키 타임으로 지정한다. 이것들을 발견하면, 움직임의 거리와 속도를 계산할 수 있고, 각 발의 궤적 또한 특수 정규화된 형태로 저장되므로, 런타임에 여러 움직임의 궤적을 함께 혼합하는 것이 간단하다.

준절차적 애니메이션

이전 장에서는 제공된 예제 운동의 특정 특성을 분석하고, 다중 예제 동작을 적절한 동기화 및 적절한 혼합 가중치와 혼합하기 위한 방법이 설명되었다. 이 장에서는 캐릭터가 울퉁불퉁한 지형에서 어떤 방향으로든 걷거나 달릴 수 있도록 혼합 모션을 조정하는 세미 절차 방법을 제안한다. 발이 다음 땅에 착지할 위치를 예측하기 위한 방법, 단계를 밟을 때 발의 궤적과 정렬을 계산하기 위한 방법, 그리고 새로운 발 위치를 수용할 수 있도록 다리와 엉덩이 높이를 조절하기 위한 방법이 설명된다. 모든 방법은 원래 모션에 충실하도록 설계되었으며, 이는 모션이 필요 이상으로 조정되지 않음을 의미한다.

1. 발자국 속성

다리 주기를 다시 한 번 살펴보면, 모션 분석의 일부로 분석된 키 타임은 캐릭터에 대한 준절차적 모션을 계산할 때 런타임에 사용된다. 다리 주기의 주요 시간은 4장에 설명되어 있다. 런타임에, 레그 사이클의 다양한 단계를 통한 진행은 이 시간 값으로 표현된다(그림 7.1b 참조).
Leg Cycle Time : 0에서 다음 스탠스 시간에 1로 선형적으로 증가하는 값입니다..
레그 사이클 시간은 레그 사이클 시간이 0 또는 1일 때 다리가 스탠스 시간에 있도록 모션 사이클 시간을 기준으로 정의된다:

```
leg[l].cycleTime =
   cyclic(motionCycleTime - leg[l].stanceTime);
```

Listing 7.1: *Cycle time of leg* l.

여기서 cyclic(k) = k - floor(k). 다음 값은 다리 사이클의 범위에 대한 부분 집합이며, 이에 비례하여 정의

됩니다.
Stride Time : 0(foot-lift)에서 1(foot-land)로 선형적으로 증가하는 값입니다.
Flight Time : 0(foot-off)에서 1(foot-strike)로 선형적으로 증가하는 값입니다.
또한, 다음과 같은 일반적인 개념은 발과 다리의 준절차적 움직임의 중심이다.
FootPrint : 특정 시간에 발을 디딜 수 있는 땅 위의 장소입니다.
Stride : 한 발자국부터 다음 발자국까지 한 걸음입니다.

2. 지상 발자국 배치

걸을 때, 우리는 우리가 생각하는 것보다 앞에 있는 땅을 더 바라보며, 우리가 다음에 땅을 밟을 대략의 장소를 끊임없이 예측합니다. 오르막길을 걸으면 땅속으로 흔들리지 않기 위해 발을 더 들어 올리고, 단 1인치 높이라도 문틀을 넘어설 때마다 발을 들어 넘어지지 않는다. 아래를 내려다볼 수 없게 하는 큰 상자를 들고 다닐 때를 제외하고는 계단을 내려가는 것이 쉬운데, 이 경우 남은 계단 수를 잘못 계산하면 우리가 생각했던 것보다 한 걸음 더 남은 것으로 판명될 경우 자칫 넘어질 뻔하다. 기대가 관건이다.

이러한 측면에서, 한 걸음 내딛는 것은 발을 지상의 특정 장소(발자국)에서 다른 장소로 들어올리는 문제이다. 이 알고리즘을 사용하면 각 발에는 언제든지 두 개의 발자국이 할당되어 있다고 말할 수 있습니다. 이전 및 다음 설치 공간. 발은 이전 발자국에서 다음 발자국까지 보폭을 만들고, 다리가 다리 주기에서 정지 시간에 도달하면, 다음 발자국은 즉시 이전 발자국으로 바뀌고, 새로운 다음 발자국이 배정된다.

2.1 발자국 예측

본 연구에서, 스탠스 시간은 설치 공간 제약 조건의 위치와 정렬을 기초로 하는 기준 시간으로 선택된다. 스탠스 시간은 스탠스 지속시간의 시작과 종료 사이에 언제든지 발생할 수 있지만, 일반적으로 스탠스 시간 휴리스틱에 의해 결정되는 중간쯤에 있다(31페이지의 섹션 4.3.2 참조). 즉, 캐릭터의 자세가 스탠스 시간에 원래 동작에서 해당 포즈와 가장 가깝게 정렬된다는 의미입니다. 주어진 다리와 관련된 캐릭터의 균형은 다리가 땅에 가장 단단하게 서 있을 때 원래 동작에 충실하도록 유지하는 것이 가장 중요하기 때문에 직관적으로 바람직하다. 그리고 발이 땅에 막 닿았을 때 또는 발을 땅에서 들어올릴 때 덜 중요하기 때문이다.

단계 중에 캐릭터가 갑자기 속도 또는 회전 속도를 변경할 경우, 다음 풋프린트의 예측은 그에 따라 변화해야 한다. 따라서 다음 발걸음의 시간, 위치 및 방향을 지속적으로 예측하고 발이 지면에 닿아 비행 단계가 완료된 시점까지 업데이트한다. 반면에 이전 설치 공간의 위치와 방향은 정적이며 지정 후에는 변경되지 않습니다.

다음 레그 풋프린트 시간의 예측 시간인 nextFootprint Time은 스탠스 시간에 도달할 때까지 남은 레그 사이클 시간을 곱한 다음 현재 시간에 추가함으로써 찾아낸다.

```
leg[l].nextFootprintTime =
   time + cycleDuration * (1 - leg[l].cycleTime);
```

Listing 7.2: *Time of next footprint for leg* l.

다음 풋프린트 시간이 주어지면, 그 당시의 세계 공간에서 캐릭터의 예측된 위치와 방향을 찾을 수 있다.

```
leg[l].nextFootprintCharacterPosition =
   characterPosition
   + (characterVelocity * (leg[l].nextFootprintTime - time));

leg[l].nextFootprintCharacterRotation =
   (characterRotationalVelocity * (leg[l].nextFootprintTime - time))
   * characterRotation;
```

Listing 7.3: *Predicted position and rotation of character at time of next footprint for leg* l.

단순성을 위해 위의 코드 리스트는 회전 속도가 사용되지 않는 동안 속도를 기준으로 캐릭터의 미래 위치를 예측한다. 이것은 많은 경우에 합리적인 결과를 준다. 구현된 시스템의 이후 버전에서는 캐릭터가 따라 움직일 것으로 예측되는 원형 경로의 반지름을 계산하여 예측에서도 회전 속도를 고려한다. 이것의 코드는 독자를 위한 연습으로 남겨져 있다.

캐릭터의 예측된 위치와 방향을 고려할 때, 세계 공간에서의 발자국의 예측된 위치와 방향도 찾을 수 있다.

```
leg[l].nextFootprintPosition =
   leg[l].nextFootprintCharacterPosition
   + leg[l].nextFootprintCharacterRotation * leg[l].stancePosition;

leg[l].nextFootprintRotation =
   leg[l].nextFootprintCharacterRotation * leg[l].stanceRotation;
```

Listing 7.4: *Predicted position and rotation of next footprint for leg* l.

예측된 위치와 방향이 빠르게 변경될 수 있으므로 원활한 연속 변경을 위해 평활화가 필요할 수 있습니다. 예측된 발자국의 영향은 현재 시간이 예측된 발자국 시간에 가까워질수록 발의 실제 위치에 대한 영향을 증가시키므로 구현된 시스템에서는 그에 따라 평활이 증가한다.

A. 발자국이 딛는 지면

위에서 설명한 대로 nextFootprintPosition 및 nextFootprintRotatin은 캐릭터의 현재 위치, 속도, 방향 및 회전 속도에만 기초한다. 지면에 발자국이 위치하도록 하기 위해, 게임 엔진의 충돌 시스템을 사용하여 예측된 위치에서 지면의 높이와 정상 벡터를 찾을 수 있다. 구현된 시스템에서는 이러한 목적으로 Raycast가 사용됩니다. 광선은 예측된 위치 위에서 시작하여 직접 아래쪽을 가리키는 광선을 따라 지면과 교차점을 확인합니다. 그런 다음 지면에 설치되고 지면 표면과 정렬되도록 접지된 예측 발자국 위치와 방향을 계산한다. 다음에서, 접지 후 원래의 비접지 위치와 회전이 더 이상 사용되지 않으므로, nextFootprint Position과 nextFootprint Rotation의 동일한 변수는 접지 위치와 방향을 참조합니다.

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화

효율성과 정확성을 절충하기 위해 구현된 시스템의 각 발에 대해 힐과 토우 위치에 해당하는 두 개의 Raycast가 사용됩니다. 일부 기본 점검은 힐과 토우 모두에 대해 부딪힌 지점이 유효한 경사가 있는지, 그리고 모든 접지가 거기에 부딪혔는지 확인하기 위해 수행됩니다.

3. Footbase 경로

발걸음을 내딛을 때, 이전 발자국에서 다음으로 이어지는 발의 경로는 많은 요소에 의해 좌우된다. 지면에 있는 동안, 발은 현재 발자국 위치와 정렬에 의해 정확히 자리해야 한다. 이건 역운동학을 사용하여 제어할 수 있다. 하지만, 발을 들어올리거나 다음 위치에 착지할 때 모션은 연속적이고 제한된 포즈로 스냅되지 않아야만 하지만 오히려 우아하게 생기를 불어넣어야한다. 다른 도전들은 원래의 키프레임 애니메이션에 충실하면서 전체적인 캐릭터 궤적, 불규칙한 지형, 다른 발의 경로에 따라 발 경로를 자연스럽게 보이게 하는 문제가 포함된다.

A. 개요

다음 예측된 발자국 뿐만 아니라 이전 발자국의 위치, 방향, 및 시점을 고려할 때, 발바닥에서 발바닥까지 비행하는 동안 발바닥은 이전 발자국에서 다음 발자국으로 이동해야 한다. 이것은 혼합된 애니메이션에 따라 완료 되어야만 하며, 또한 월드 좌표계에서 주어진 발자국의 위치와 방향에 따라 수행되어야 합니다. 다음 서브 섹션에서 설명한 대로 분석 단계에서 발견된 표준화된 발밑 궤적이 작동하게 된다. 정규화된 발밑 궤적을 적용하는 동시에 불규칙할 수 있는 지형을 고려하는 것은 다음 서브섹션에서 논의될 여러 하위 문제들로 나뉜 복잡한 문제이다.

1. 원래 비행 과정 적용
   정규화된 발밑 궤적의 진행을 사용하여 발바닥의 가속과 감속을 원래 모션에서 동일하게 유지한다.
2. 궤적 곡선을 따르는 캐릭터
   캐릭터가 걷고 있는 곡선을 따라 오도록 직선 발밑 궤적에 오프셋을 추가한다.
3. 접지 접선 기반 리프팅
   발밑이 불규칙한 지면과 교차하지 않도록 적절한 호에서 들어올린다.
4. 지지 발의 높이로 들어 올립니다.
   지지 다리 또는 다리가 리프팅 발보다 더 높은 높이에 서 있는 경우, 리프팅 발을 같은 높이까지 호로 들어올립니다.
5. 원래 리프팅과 측면이동 적용
   지금까지 계산된 궤적 위에 원래의 모션에 충실하기 위해 정규화된 발밑 궤적에서 측면 및 리프팅 움직임을 추가한다.

B. 기존의 비행 과정 적용

수평 발밑 비행 경로의 가장 단순한 구현부터 시작하여, 다음에서 문제에 대해 다양한 접근들에 대해 토론할 것이다. 가장 단순한 접근은 비행 시간을 보간 변수로 사용한 이전과 다음 발자국 사이에 선형 보간이다.

```
leg[l].footbasePosition =
Lerp (
   leg[l].prevFootprintPosition,
   leg[l].nextFootprintPosition,
   leg[l].flightTime
);
```

Listing 7.5: *Linear interpolation between previous and next footprint based on flight time.*

Lerp 함수가 선형 보간 함수인 경우, 다음과 같이 정의됩니다.

$$\mathrm{Lerp}(a,b,t) = a \cdot (1-t) + b \cdot t \tag{7.1}$$

그러나 이것은 원래 동작에서 발의 움직임 곡선에 충실하지 않기 때문에 비행 중 발의 특정 가속 및 감속과 같은 원래 특성을 보존하지 못한다. 모션 분석 장에서는 각 모션 예제에 대해 각 발에 대해 정규화된 풋베이스 궤적이 저장되는 방법을 설명하였다(37페이지의 섹션 4.4.1 참조). 궤적이 정규화되었기 때문에 의미 있는 평균이 될 수 있으며, 가중 평균은 현재 순간의 예제 모션을 혼합하는 데 사용되는 동일한 가중치와 함께 사용된다. 아티팩트를 방지하기 위해 43페이지의 섹션 5.2에 설명된 대로 다양한 풋베이스 궤적이 시간 뒤틀림을 사용하여 정렬됩니다. 표준화된 발밑 궤적은 (0,0,0)에서 출발하여 (0,0,1)에서 착지한다. 정규화된 경로의 이동이 항상 z축을 따라 있기 때문에, 이동 축을 따라 발의 가속과 감속이 정규화된 풋베이스 궤적에 있는 위치의 z-구성 요소에서 직접 도출될 수 있다. 이를 이전 및 다음 풋프린트 사이의 선형 보간 값에 대한 보간 값으로 사용하면 이미 다음과 같이 현저하게 개선되었습니다.

```
leg[l].footbasePosition =
Lerp (
   leg[l].prevFootprintPosition,
   leg[l].nextFootprintPosition,

   leg[l].normalizedFootbaseTrajectory(leg[l].cycleTime).z
);
```

Listing 7.6: *Linear interpolation between previous and next footprint based on the z component of the normalized footbase trajectory.*

C. 궤적 곡선을 따르는 캐릭터

이 접근 방식의 문제는 속도는 선형적이지 않지만, 이전과 다음 발자국 위치 사이에서 발이 여전히 직선으로 움직인다는 것이다. 한 방향으로 직진할 때는 문제가 없지만, 호를 돌리면 부자연스러운 결과가 나온다. 캐릭터(적어도 사람)가 호를 그리며 걸을 때, 발은 호를 따라가는 경향이 있다. 이것은 단순히 직선으로 움직이면 다른 다리와 교차할 가능성이 높기 때문에 외부 발의 경우 특히 중요하다(그림 7.2a 참조). 발이 적절한 곡선에서 이전 발자국에서 다음 발자국으로 이동하기 위해서는, 즉 자연 비행 중간 위치(그림 7.2b 참조)와 혼합 동작만으로 제어될 경우, 위치에 해당하는 비행 중 중간 지점을 통과하는 것이 바람직하다. 각 다리의 조정은 다른 다리와 관계없이 별도로 처리되기 때문에 다리의 교차를 직접적으로 막을 수는 없지만, 조정에 앞서 혼합된 움직임이 교차하지 않는 한, 조정 후의 움직임 또한 발이 그것의 자연적인 중간 비행을 통과할

때 교차점이 없을 가능성이 매우 높다.

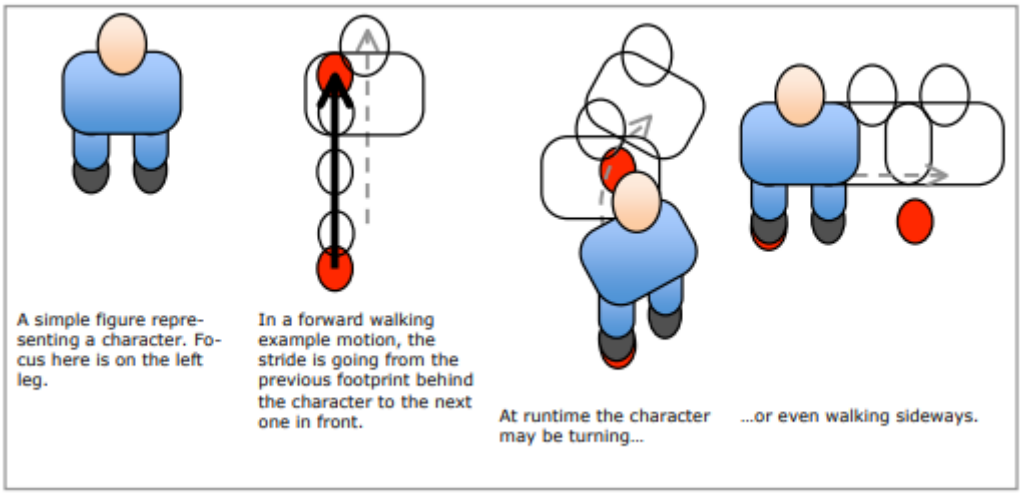
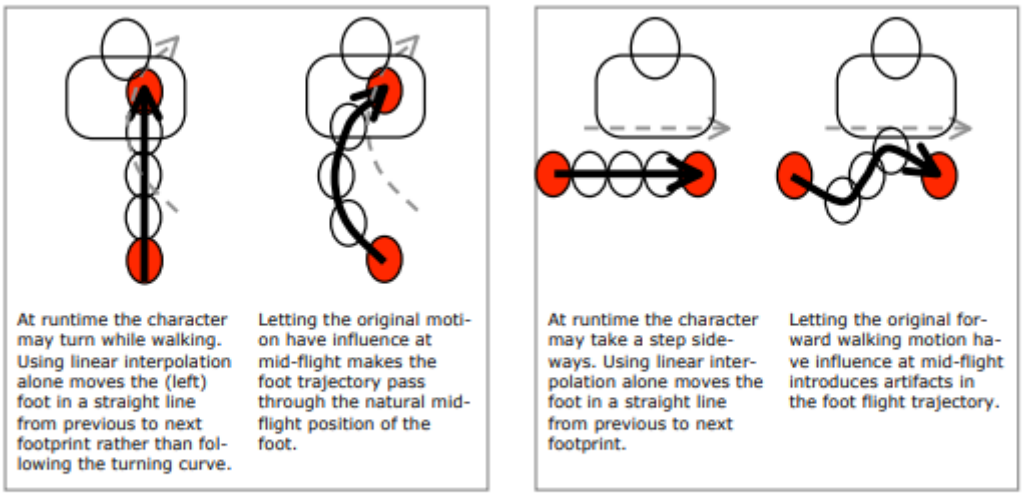


그림 7.3: 풋베이스 비행 궤적을 조정하면 캐릭터가 원래 동작처럼 앞으로 똑바로 걷거나, 회전하거나, 옆으로 걷거나, 다른 방향 또는 회전 각도로 걷거나 상관없이 좋은 결과를 얻을 수 있어야 한다.

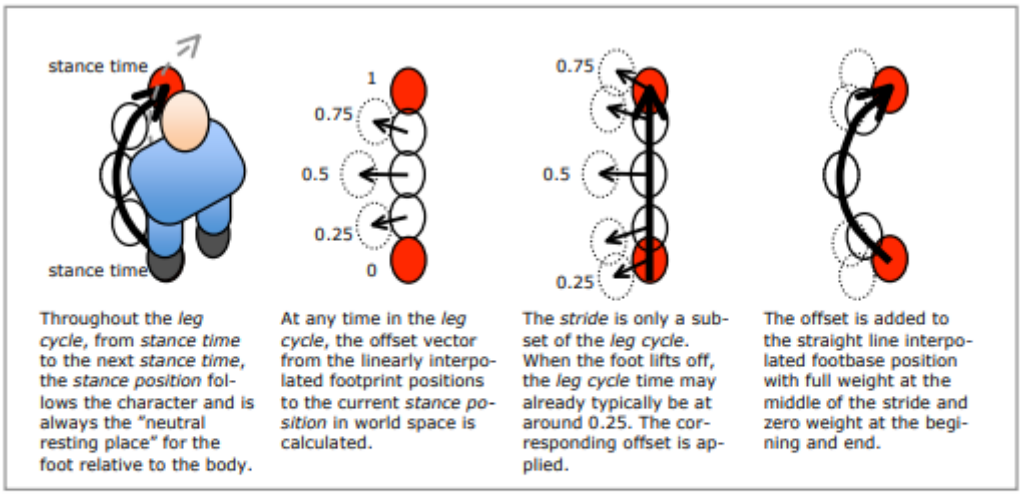
자연 비행 중간 위치를 통과하는 발에 대해 비행 궤적을 구현하는 방법은 여러 가지가 있다. 제가 처음에 실험한 접근법 중 하나는 원래의 혼합된 움직임이 점차적으로 보폭의 중간을 향해 발을 통제하게 하는 것입니다. 그러나 어떤 경우에는 캐릭터가 원래의 움직임과 매우 다른 속도나 방향을 가지고 걷는 경우, 이 솔루션은 눈에 보이는 결함을 가지고 있다. 예를 들어, 캐릭터가 전진 보행 운동을 기반으로 옆으로 걷는 경우, 원래 모션의 전방 헤드 방향은 중간 주변의 궤적에 영향을 줄 것이며(그림 7.4a 참조) 부자연스러운 아티팩트를 생성한다. 나는 결국 위에서 설명한 초기 접근 방식의 결함에 시달리지 않는 다른 접근 방식을 고안했다: 다시 한번 비행 중 주변의 최대 무게로 발 밑 궤적에 특별한 오프셋이 추가되어 발의 자연적인 중간 비행 위치를 통과하게 된다(그림 7.4b 참조). 오프셋 벡터는 다리 주기 시간을 기준으로 이전 및 다음 풋프린트의 선형 보간에 상대적인 세계 공간의 현재 자세 위치이다. 보간 위치는 단지 그것의 보폭 부분만이 아니라 전체 다리 주기가 스탠스 시간에서 다음 스탠스 시간까지 지속되는 동안 이전 위치에서 다음 풋프린트 위치로 선형 속도로 이동한다. 이것은 최소한 일정한 속도로 이동할 때 문자 위치가 이동하는 방식과 유사합니다. 즉, 캐릭터가 일정한 속도로 움직이는 한, 캐릭터의 속도가 원래 움직임의 속도와 동일하든 그렇지 않든 간에 오프셋은 항상 0의 길이이다.

flightProgressionArc = sin(leg[l].normalizedFlightPosition.z · π)

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화



(a) *Foot flight trajectory using influence of the original motion at mid-flight.*



(b) *Foot flight trajectory using offset vectors at mid-flight.*

Figure 7.4: *Adjustment of footbase flight trajectory using (a) influence of original motion at mid-flight, and (b) offset vectors added at mid-flight.*

이 오프셋 가중치 기능을 사용하면 그림 7.2b와 같이 발 궤적이 캐릭터의 전체 경로를 잘 따르는 매우 부드럽고 자연스러워 보이는 결과를 얻을 수 있다. 위의 코드 목록에 오프셋이 계산되어 풋베이스 위치에 추가됩니다.

```
stancePositionInWorld =
   characterPosition + (characterRotation * leg[l].stancePosition);

interpolatedFootprintPosition =
Lerp (
   leg[l].prevFootprintPosition,
   leg[l].nextFootprintPosition,
   leg[l].cycleTime
);

offset = stancePositionInWorld - interpolatedFootprintPosition;

leg[l].footbasePosition += offset * flightProgressionArc;
```

Listing 7.7: *Interpolation between reference footbase in character space and interpolated footbase in global space.*

D. 접지 접선 기반 리프팅

캐릭터가 평평한 수평면을 걷는다고 가정할 수 없기 때문에 발 밑 지형을 고려해야 한다. 카이 정과 한(1999년)이 기술한 방법처럼 여기서 목표는 최소한의 에너지로 지형을 명확하는 발 궤도 곡선을 찾는 것이다. 로컬 지형에 대한 철저한 분석은 런타임에 너무 계산적으로 비용이 많이 드는 것으로 간주되기 때문에, 이러한 위치에서 표면의 정상뿐만 아니라 발자국의 위치만 고려하는 간단한 접근법이 사용된다. 이 정보는 prevFootprintPosition, prevFootprintRotation, nextFootprintPosition 및 nextFootprintRotation 회전에서 사용할 수 있습니다. 이전 발자국과 다음 발자국 사이의 중간점이 발견되며, 두 발자국 각각에 대해 중간 지점에서 수직으로 위로 올라가는 축과 발자국 아래 면의 교차점이 발견된다(그림 7.5a 참조). 이 두 교차점 중 중간점에서 가장 높은 지점까지의 거리는 두 접선 중 높은 곳을 따라 곡선이 따라오도록 발을 위로 들어 올리는 부비동 곡선의 크기를 결정하는 데 사용된다(그림 7.5c 참조). 계산은 예로서 이전 발자국을 사용하여 다음과 같다(그림 7.5a 및 7.5b 참조).

$$\text{mid} = \frac{\text{prev} + \text{next}}{2} \tag{7.3}$$

$$\mathbf{a} = \text{prev} - \text{mid} \tag{7.4}$$

$$b = \mathbf{a} \cdot \mathbf{n}_{prev} \tag{7.5}$$

$$h_{prev} = \frac{b}{\mathbf{n}_{prev} \cdot \text{up}} \tag{7.6}$$

여기서 업은 글로벌 상향 포인팅 유닛 길이 벡터입니다. 교차로 점의 높이를 계산하면 정동 곡선의 크기를 확인할 수 있습니다.

$$h_{max} = \max(h_{prev}, h_{next}) \quad (7.7)$$

$$\texttt{arcMagnitude} = \frac{2 \cdot h_{max}}{\pi} \quad (7.8)$$

위의 코드 목록에 앞서 정의한 발밑 위치의 수평 궤적에 리프팅 호가 추가됩니다.

```
leg[l].footbasePosition += up * arcMagnitude * flightProgressionArc;
```

Listing 7.8: *Lifting arc added to footbase trajectory.*

E. 지지 땅의 높이

원래 동작은 평평한 지면 위에서 특정 거리를 "호버링"하는 캐릭터의 몸통을 가지고 있다. 원래 모션에 충실하려면 이 호버 높이를 런타임에 조정된 모션으로 유지해야 합니다. 그러나 캐릭터가 런타임에 고르지 않은 지형 위를 걸을 때, 다른 발이 다른 높이의 표면을 밟을 수 있으며, 캐릭터 몸통의 높이를 어느 표면과 비교하여 측정해야 하는지를 결정해야 한다.

걷는 동안 어느 시점에서도 캐릭터의 무게는 다른 곳보다 몇 개의 다리에 의해 더 많이 지탱된다. (두 발 보다 다리 하나가 더) 여기서는 모든 캐릭터가 체중을 운반하기 가장 쉬운 다리 구성을 가지고 있다고 가정한다. 예를 들어 인간의 지지 다리는 거의 완전히 뻗은 반면, 더 구부러진 다리로 무게를 운반하는데 더 많은 노력이 필요하다. 현재 캐릭터의 무게를 운반하지 않는 미드 스윙의 다리는 훨씬 적은 노력으로 원래 모션에서 구성보다 더 많이 또는 덜 구부릴 수 있다. 이러한 이유로, 캐릭터의 무게를 지탱하고 있는 다리 또는 다리들의 굽은 정도 또는 늘어난 양을 주로 유지하는 방법이 여기에 제안된다.

캐릭터 몸통의 실제 호버 높이 계산은 84페이지의 섹션 7.4.2에서 설명되며, 현재 캐릭터 무게를 지탱하고 있는 발을 고려하지 않고 발의 위치와 정렬에 직접적으로 의존한다. 대신 이것은 발밑 궤적을 계산할 때 고려된다.

여기서 제안되는 것은 현재 캐릭터의 무게를 지탱하고 있는 표면의 높이에 대한 가중 평균을 나타내는 지지 지면 높이 궤적의 계산이다. 각 발은 몸통 호버 노이 계산에서 발이 캐릭터를 아래로 당기지 않도록 적어도 지지 지면 높이의 높이까지 들어올려져야 한다.

지지 지면 높이 궤적을 찾긱 전에, 각 발에 대해 해당 발의 발자국과 관련하여 발 높이 궤적이 정의된다. 발의 발바닥 높이 궤적은 발 자체가 실제로 움직이는 방법과 직접적인 관련이 없지만 발의 연속적인 발자국을 통과하는 부드러운 곡선으로, 음의 자세 위치로 상쇄된다.

어떤 시점에서든, 발바닥 높이 궤적의 세그먼트는 이전 발자국과 다음 발자국 사이에 정의된다. 접선의 방향은 캐릭터의 속도와 현재 회전에 대한 이전 및 다음 발자국에서 캐릭터 회전에 기초하여 계산된다. 접선 벡터의 길이는 한 단계의 길이와 같은 현재 주기 거리에 의해 결정됩니다.

```
velocityInCharacterSpace =
   inverse(characterRotation) * velocity;
directionInCharacterSpace =
   (projectedOntoGround(velocityInCharacterSpace)).normalized;

prevFootprintTangent =
   leg[l].prevFootprintCharacterRotation
   * directionInCharacterSpace
   * cycleDistance;

nextFootprintTangent =
   leg[l].nextFootprintCharacterRotation
   * directionInCharacterSpace
   * cycleDistance;
```

Listing 7.9: *Tangents along the ground at previous and next footprint for foot* l.

곡선의 평탄도는 S자 곡선을 사용하여 곡선 세그먼트의 길이를 따라 두 접선의 가중치를 제어함으로써 얻어집니다.

```
// S-shaped curve that starts at 0 and ends at 1:
sCurve = -cos(pi * leg[l].strideTime)/2 + 0.5;

leg[l].groundHeightPoint =
Lerp (
   leg[l].prevFootprintPosition
      + prevFootprintTangent * leg[l].cycleTime
   ,
   leg[l].nextFootprintPosition
      + nextFootprintTangent * (leg[l].cycleTime - 1)
   ,
   sCurve
);
```

Listing 7.10: *Calculation of the foot ground height point for foot* l.

발 원점이 아닌 신체 원점에 대한 궤적을 얻기 위해 곡선은 음의 자세 위치로 오프셋됩니다.

```
leg[l].groundHeightPoint +=
   characterRotation * -leg[l].stancePosition;
```

Listing 7.11: *Shifting of the foot ground height trajectory by the negative stance position.*

각 발에 대한 발바닥 높이 궤적이 계산되면, 지지 높이 궤적을 찾을 수 있다. 지지 지면 높이 궤적은 개별 발의 지면 높이 궤적에 대한 가중 평균이다. 현재 접지된 발이나 발에 대한 강력한 지지를 시뮬ㄹ레이션하기 위해, 지반 높이 궤적은 대부분 주어진 시간에 접지된 발이나 발에 읮ㄴ하는 반면, 미드 스윙의 발들이나 발은 대부분 무시된다. 각 발의 영향력은 다리 주기 시간과 관련하여 정의된다.

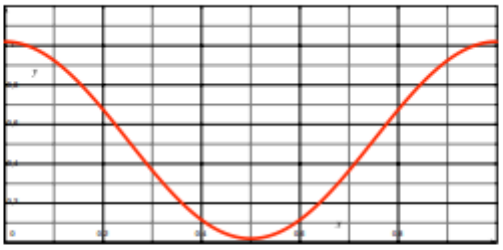


Figure 7.8: *Influence of foot* l *on the supporting ground height trajectory as a function of the leg cycle time.*

여기서 ε은 작은 양의 값입니다. 이 발의 영향은 스탠스 시간에는 1+ ε이고, 미드 스윙에서는 ε로 감소하여, 스탠스 시간에는 1+ ε로 다시 증가한다 (그림 7.8참조)

지지 지면 높이 궤적은 다음과 같이 계산됩니다.

$$\texttt{supportingGroundHeightPoint} = \frac{\sum_{l=1}^{legs} h_l \cdot \texttt{leg[l].groundHeightPoint}}{\sum_{l=1}^{legs} h_l} \tag{7.10}$$

각 발에 대해 리프팅 높이가 계산됩니다. 상승된 높이의 궤적은 비행 시간 기반 호를 기준으로 비행 중 중간 지반 높이 지점의 높이로 보간된다.

$$\texttt{flightTimeArc} = \sin(\texttt{leg[l].flightTime} \cdot \pi) \tag{7.11}$$

지지 지반 높이 지점의 상승 높이는 처음에 들어올린 발 밑 높이보다 낮을 수 있으므로, 상승된 높이가 현재 발 밑 높이보다 높을 때만 상승된 높이까지 상승된다.

```
liftedHeight =
Lerp (
   leg[l].footbasePosition.height,
   supportingGroundHeightPoint.height,
   flightTimeArc
);

if (liftedHeight > leg[l].footbasePosition.height) {
   leg[l].footbasePosition.height = liftedHeight;
}
```

Listing 7.12: *Supporting feet based lifting added to footbase trajectory.*

F. 기존 리프팅과 측면 이동 적용

원래의 움직임으로부터 발밑 궤적에 의해 지시되는 리프팅과 측면 움직임은 다른 요소들을 고려한 후에 마지막에 사소한 것으로 추가된다. 평평한 수평면에서 이동할 때 다른 요인은 모두 오프셋이 0이 됩니다. 리프팅과 측면 이동은 오직 정규화된 발밑 궤적을 기반으로 하기 때문에 원래 움직임의 발밑 궤도와 동일하다.

원래 동작에서 측면 이동과 발 밑면의 리프팅은 모션 분석에서 분석된 정규화된 발밑 궤적의 측면(x) 및 수직(y) 구성 요소를 기반으로 발밑 궤적에 추가됩니다.

```
1 stepDirection = (leg[l].nextFootprintPosition - leg[l].prevFootprintPosition).
      normalized;
2 stepSidewaysDirection = CrossProduct(up, stepDirection);
3
4 leg[l].footbasePosition += stepSidewaysDirection * leg[l].
      normalizedFlightPosition.x;
5 leg[l].footbasePosition += up * leg[l].normalizedFlightPosition.y;
```

Listing 7.13: *Vertical and sideways movement of footbase from original motion added to footbase trajectory.*

4. 다리 조정

각 발에 대한 발 밑 위치와 정렬이 발견되면 다리의 자세를 결정할 수 있습니다. 이것은 최적의 엉덩이 높이, 발 정렬을 찾고, 각 다리의 엉덩이와 발목 사이의 뼈의 정렬을 찾기 위해 역운동학을 사용하는 것을 포함한다.

A. 엉덩이 조정

12페이지의 섹션 1.3.1에서 논했듯이, 오늘날 대부분의 게임들은 여전히 내부적으로 하나의 오브젝트로 캐릭터를 나타낸다. 캐릭터가 얼마나 빨리 걷는지 제어하기 위해, 힘이나 속도가 전체적인 게임 논리에 의해 캐릭터 객체 전체에 할당된다. 일반적으로 물리학을 시뮬레이션하는 게임에서 캐릭터는 벽을 통과하거나 바닥을 통과해서 떨어지는 것을 막는데, 게임 내에서 물리적으로 시뮬레이션된 다른 물체들과 충돌하고 쉴 수 있는 수직 정렬 실린더와 같은 하나의 물리적 물체로서 그를 표현한다.

캐릭터의 다리는 일반적으로 물리적 시뮬레이션에서 완전히 제외되기 때문에, 지면 위의 캐릭터 오브젝트의 고도는 종종 캐릭터가 실제로 땅을 밟는 위치와 직접적인 관계는 없다. 특히 울퉁불퉁한 지형이나 계단에서 이동할 때 캐릭터의 수직 이동은 전통적으로 덜컹거리거나 부자연스럽게 보입니다. 캐릭터가 적절하게 접지된 것처럼 보이게 하기 위해, 일반적으로 루트 본의 고도를 제어하는 캐릭터 오브젝트의 고도에 관계없이, 발의 위치와 정렬에 따라 캐릭터의 루트 본의 고도를 결정하는 방법이 제안된다. 루트 본 위치에 대한 수평적 조정은 이루어지지 않으며, 루트 본의 고도는 원래 동작에서 애니메이션 되는 방법과 완전히 일치하는 방식으로 계산된다.

기존 애니메이션의 어떤 지점에서, 각 다리에 대해 해당 다리의 엉덩이와 발목 사이에 일정한 거리가 있다. 기존 애니메이션에 충실하기 위해, 엉덩이에서 발목까지의 거리는 런타임에서 조정된 애니메이션에서 가능한 한 많이 유지되어야 한다. 그러나 캐릭터가 울퉁불퉁한 지형 위를 런타임에서 걸을 때, 모든 다리에 대해 엉덩이와 발목 사이의 원래 거리를 동시에 유지하는 것은 불가능할 수 있다.

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화

많은 요인들이 발의 조정된 위치에 영향을 미치기 때문에, 그것들은 기존 움직임에서 발의 참조 위치를 기준으로 3차원에서 모두 오프셋할 수 있습니다. 각 다리에 대해 서로 다른 두 개의 잠재적 엉덩이 높이가 계산됩니다. 원하는 엉덩이 높이, 다리가 휘는 정도를 유지할 수 있는 고관절 높이와 최대 고관절 높이를 hdesired, 코바르 (2002)방법으로 다리를 완전히 뻗을 수 있는 높이를 hmax. 이것은 구와 수직선 사이의 교차점을 찾는 것으로 이루어진다.

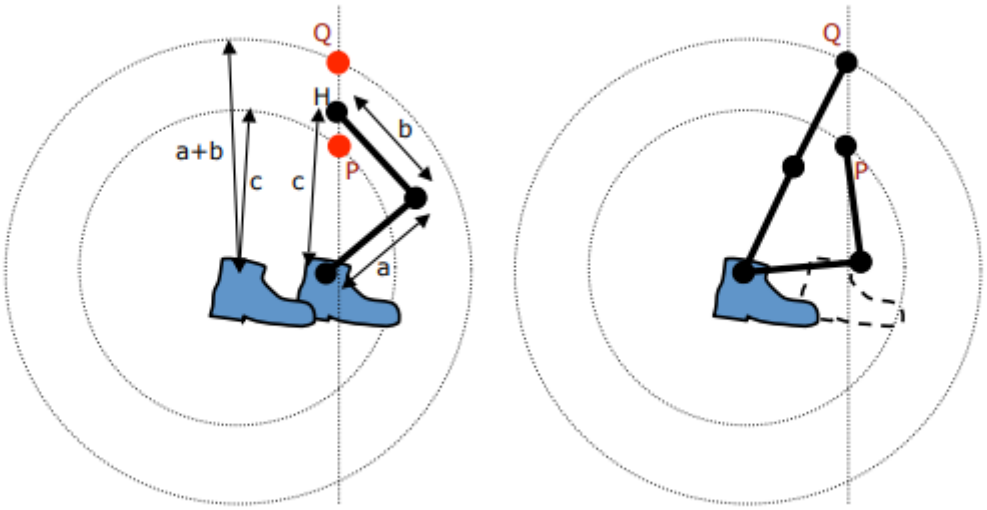


그림 7.9: 엉덩이에 상대적인 발목의 위치는 원래 동작과 런타임에 다른 위치에 있을 수 있으므로 엉덩이의 높이를 조정할 필요가 있을 수 있다. 각 다리에 대해 구체의 교차점은 다리를 완전히 뻗게 하는 최대 엉덩이 높이 Q뿐만 아니라 다리의 원래 굽힘 양을 유지하는 원하는 엉덩이 높이 P를 찾는 데 사용된다. 특히, 교차점은 원래 엉덩이 위치를 통과하는 수직선과 그 중심이 새로운 발목 위치에 있는 두 개의 구에서 발견된다. 한 구는 현재 자세에서 엉덩이와 발목 사이의 원래 거리의 반경을 가지고 있고, 다른 구는 엉덩이에서 발목까지의 다리 길이에 해당하는 반경을 가지고 있다.

특히, 교차점은 원래 엉덩이 위치를 통과하는 수직선과 그 중심이 새로운 발목 위치에 있는 두 개의 구에서 발견된다. 하나의 구는 현재 자세에서 엉덩이와 발목 사이의 원래 거리의 반경을 가지고 hdesired를 찾는데 사용되고, 다른 구는 엉덩이에서 발목까지의 다리 길이에 해당하는 반경을 하지고 있으며 hmax를 찾는데 사용된다(그림 7.9 참조). Hdesired와 hmax의 높이가 기존 모션의 엉덩이 높이에 상대적인 오프셋으로 저장됩니다. 뒷다리가 앞다리보다 길거나 비슷한 동물들처럼 엉덩이가 지면에서 같은 고도를 가지지 않는 캐릭터들의 다른 엉덩이 높이를 비교할 수 있기 때문에 절대 높이보다 오프셋을 저장하는 것이 유리하다.

각 다리에 대해 원하는 최대 고관절 높이를 찾은 후 다음과 같은 세 가지 높이를 확인할 수 있습니다. 최대 고관절 높이 중 가장 낮고, 원하는 고관절 높이 중 가장 낮으며, 평균 고관절 높이입니다.

$$hmax_{min} = \min_{l=1}^{legs} (hmax_l) \quad (7.12)$$

$$hdesired_{min} = \min_{l=1}^{legs} (hdesired_l) \quad (7.13)$$

$$hdesired_{avg} = \frac{\sum_{l=1}^{legs} hdesired_l}{legs} \quad (7.14)$$

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화

엉덩이의 최종 간격 띄우기의 계산은 다음과 같은 여러 가지 고려 사항을 바탕으로 세 가지 높이를 고려합니다.

- 순진한 타협은 모든 다리가 원하는 고관절 높이의 평균인 $hdesired_{avg}$를 선택하는 것이다. 그러나, 이는 일부 다리가 과도하게 늘어나게 할 수 있기 때문에 효과가 없으며, 이는 엉덩이 지점과 발목 지점 사이의 거리가 엉덩이에서 발목까지의 다리 길이보다 더 길다는 것을 의미한다.
- 과도하게 늘어나는 것을 피하기 위해, 엉덩이 높이는 모든 다리의 최대 엉덩이 높이 중 가장 낮은 높이, $hmax_{min}$ 이하이어야 합니다.
- 엉덩이 높이는 $hdesired_{avg}$와 $hmax_{min}$의 최소값으로 선택할 수 있다. 그러나 이것은 $hdesired_{avg}$와 $hmax_{min}$의 값이 서로 교차할 때마다 고관절 궤도에서 2차 불연속성을 발생시키며, 이는 동작이 부드럽지 않음을 의미한다. 게다가, $hmax_{min}$이 고관절 높이로 선택될 때마다, 최소한 한 개의 다리가 완전히 늘어나서 뻣뻣하고 부자연스러운 움직임을 만들어낸다.
- 다리를 원래 동작보다 더 구부러지게 하는 것은 다리가 더 늘어나게 하는 경우보다 바람직하지 않은 인공품들을 만들 가능성이 적다. 따라서 원하는 모든 다리의 엉덩이 높이 중 가장 낮은 부분인 $hdesired_{min}$이 기본 높이로 사용된다. $hdesired_{min}$은 항상 $hdesired_{avg}$와 $hmax_{min}$ 둘 다보다 작거나 같다고 가정할 수 있다.
- 고관절 높이가 $hmax_{min}$에 가까울수록 다리를 너무 뻣뻣하게 만들어 한쪽 다리의 구성 ($hmax_{min}$이 최대 높이인 다리)이 더 많이 손상됩니다. 동시에, 엉덩이 높이가 $hdesired_{avg}$에서 멀어질수록, 현재 조정된 캐릭터 포즈가 원래 동작에서의 현재 캐릭터 포즈랑 비교했을 때 덜 비슷할 것이다.
- $hmax_{min}$이 높을수록, 다리가 너무 뻣뻣해지지 않고 엉덩이 높이가 $hdesired_{avg}$에 근접할 수 있다.

$hdesired_{min}$에서 $hdesired_{avg}$ 및 $hmax_{min}$까지의 높이 차이는 다음과 같다:

$$mintoavg = hdesired_{avg} - hdesired_{min} \quad (7.15)$$

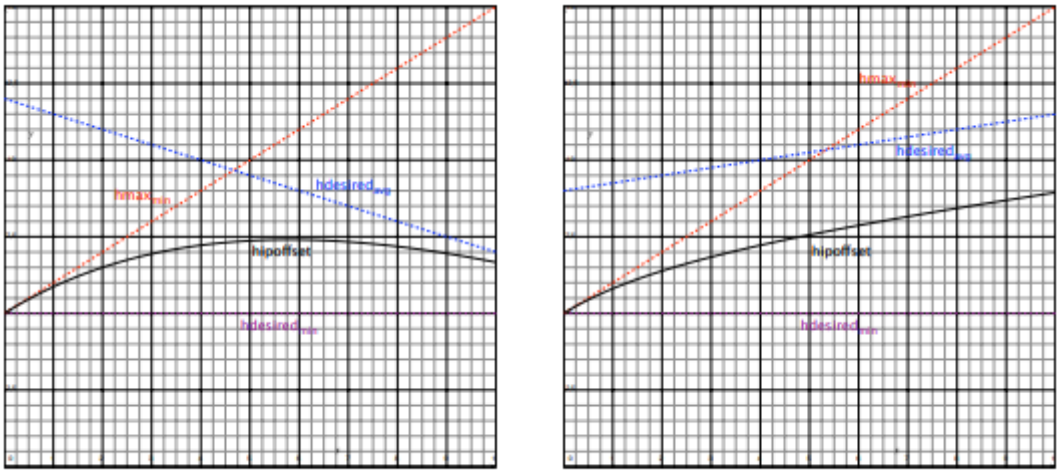
$$mintomax = hmax_{min} - hdesired_{min} \quad (7.16)$$

그런 다음 다음과 같이 엉덩이의 오프셋을 계산합니다:

$$hipoffset = hdesired_{min} + \frac{mintoavg \cdot mintomax}{mintoavg + mintomax} \quad (7.17)$$

그림 7.10은 $hmax_{min}$이 무한대에 접근할 때 $hdesired_{avg}$에 접근하는 hipoffset을 보여주고 있으며 그 반대도 마찬가지이다. 캐릭터의 루트 본은 수직축을 따라 이 hipoffset으로 오프셋되며 각 다리의 엉덩이가 이 오프셋을 상속합니다. 루트 본의 간격 띄우기는 종종 매우 미미하여 특별히 최소화하려고 시도하지 않으며, 특정한 경우에 중요할 수 있다.

캐릭터 로코모션을 위한 준절차적 애니메이션 자동화



(a) *Example 1.* (b) *Example 2.*

그림 7.10: $hdesired_{min}$에서 $hdesired_{avg}$까지의 범위보다 $hdesired_{min}$에서 $hmax_{min}$까지의 범위가 몇 배는 더 클 때, hipoffset은 $hdesired_{avg}$에 접근하며, $hmax_{min}$과 과도하게 늘어난 다리에 도달할 위험이 없을 때, 평균 desired한 엉덩이 높이를 면밀히 추구해야 한다. 반대로, $hdesired_{min}$에서 $hdesired_{avg}$까지의 범위가 $hdesired_{min}$에서 $hmax_{min}$까지의 범위보다 몇 배 더 클 때, hipoffset은 최대치에 근접하는데, 이는 평균 desired한 엉덩이 높이가 더 높을수록 해당 높이에 더 가깝게 다리 스트레칭을 허용해야 한다.

B. 발 정렬

모션 전체에 걸쳐 캐릭터의 발을 정렬하는 것은 전체적인 모션 인식 방법에 매우 중요합니다. 인간은 걸을 때 발을 굴리고, 먼저 발꿈치로 땅을 친 다음, 나머지르 발을 땅에 평평하게 착지하고, 마지막으로 발 뒤꿈치부터 발끝으로 들어올리는 경향이 있다. 하지만 걷기나 달리기의 특정 스타일은 다른 특성을 가진 풋롤을 가질 수 있으며, 인간이 아닌 캐릭터는 이상한 풋롤을 가질 수 있다. 이 작업에 사용된 일반화된 카 타임에 대한 개요는 모션 분석 장의 4.2.1항을 참조하십시오. 또한, 저는 다음 단계에 다리 주기의 다양한 시간 범위를 정의하는 설명을 하였습니다. 풋롤 역학에 대한 논의에서 유용하게 사용됩니다:

Stance Phase : 발이 땅에 평평하게 놓여 있고 캐릭터의 무게를 지탱하는 풋 랜드와 풋 리프트 사이의 단계입니다. 스탠스 시간은 항상 이 단계의 시간 범위 내의 특정 지점에서 발생하며, 종종 이 단계의 중간 부근에서 발생합니다.

Flight Phase : 한 발자국에서 다음 발자국으로의 비행에서 발이 들어올려지는 발을 뗀 이후와 발 떼기 이전 사이의 단계, 그리고 이전 발자국에서 발차기를 완료하고 아직 다음 발자국에서 발차기를 시작하지 않은 단계이다.

Foot-roll : 이론적으로 그 시간은 foot-lift에서 foot-off까지 그리고 foot-strike에서 foot-land까지 이다. 하지만 발 뒤꿈치와 발가락이 동시에 땅에 부딪히면서 (또는 땅에서 들어올릴 때와 비슷) 기존 모션에서 발이 땅에 평평하게 착지할 수 있기 때문에, 0 또는 매우 짧은 시간 동안 풋롤이 발생하며, 이러한 짧은 풋롤 지속 시간이 운동을 수정할 때 좋지 않은 결과를 초래하기 때문에 풋롤은 항상 일정한 최소 길이를 갖도록 강제됩니다. 이 목적을 위해 post-foot-lift 및 pre-foot-land까지 키 시간이 정의된다. 리프팅 풋롤은 foot-lift에서 post-foot-lift까지 그리고 랜딩 풋롤은 pre-foot-land에서 foot-land까지이다.

Stance Phase와 Flight Phase가 전체 다리 주기에 합산되지 않는다는 점에 유의한다. Stance Phase가 종료된 후와 Flight Phase가 시작되기 전에 과도적 풋롤 단계가 있으며, Flight Phase가 종료된 후와 새로운

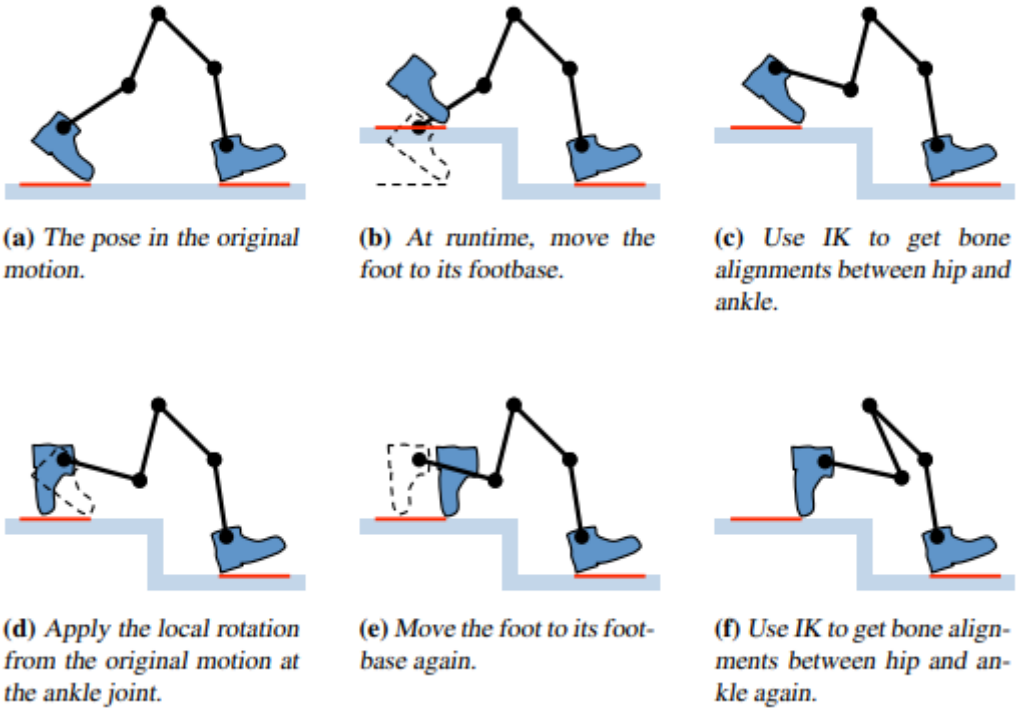
Stance Phase가 시작되기 전의 경우도 마찬가지이다.

인간이 다양한 고르지 않은 표면 위를 걷는 방법을 관찰한 후, 여기서 제안된 방법은 다음과 같은 명제에 기초한다:

Ground surface based alignment : 기존 모션에서 발이 지면에 평평하게 놓여 있는 Stance Phase에서, 발이 조정된 동작에서 놓여 있는 표면의 경사에 관계 없이, 발이 조정된 동작에서도 지면에 평평하게 놓여 있어야 한다. 즉, 지면과 상대적인 발의 정렬 상태를 유지해야 합니다. 이는 캐릭터의 무게를 지탱하고 지면과 최대한 마찰을 일으키는 듯한 인상을 준다.

Ankle joint rotation based alignment : 발이 지면에 전혀 닿지 않는 Flight Phase에서, 조정된 동작은 원래 동작에서 발목 관절의 로컬 회전을 유지하려고 노력해야 한다. 이것은 발을 특정한 각도로 유지하기 위해 어떤 노력도 들이지 않고 보폭을 취하는 인상을 준다 - 발은 단순히 하퇴부에 비해 자연스러운 휴식 정렬에 있다. 이러한 가정은 연구된 인간과 동물의 움직임에도 효과가 있고, 카이 정과 한의 논문과도 일치한다.

Foot-rolls : Ground surface based alignment는 Stance Phase에서 사용해야 한다. Ankle joint rotation based alignment는 Flight Phase에서 사용해야 한다. Stance Phase와 Flight Phase를 연결하는 풋롤 단계에서는 두가지 형태의 정렬 사이에 보간이 사용되어야 한다.



(a) *The pose in the original motion.*

(b) *At runtime, move the foot to its footbase.*

(c) *Use IK to get bone alignments between hip and ankle.*

(d) *Apply the local rotation from the original motion at the ankle joint.*

(e) *Move the foot to its footbase again.*

(f) *Use IK to get bone alignments between hip and ankle again.*

그림 7.11 : 각 프레임에서 런타임에 수행되는 스텝은 원래 모션에 최대한 충실하면서 불규칙한 지형에서 적절한 풋롤을 얻기 위해 수행됩니다. (D)에서의 회전은 발이 원래 동작에서 지면에 평평하게 서 있는 자세 단계에서 0이고 발이 비행 단계에 있을 때 0인 가변 양이 적용된다.

Ankle joint rotation based alignment 방법은 몇 가지 간단한 단계로 세분될 수 있다 (그림 7.11참조). 원래 동작 (A)에서 다리 뼈의 초기 정렬에서 발은 먼저 발바닥 (B)로 이동합니다. 역운동학은 엉덩이와 발목 관절 사이의 뼈의 정렬을 찾는데 사용된다. 원래 동작에서 발목의 로컬 회전이 이제 발목 관절 (D)에 적용된다. 발은 다시 발바닥 (E)으로 이동되고, 역운동학은 다시 엉덩이와 발목 사이의 뼈의 정렬을 찾기 위해 사용된다.

접지면 기반 정렬은 (D)의 회전이 적용되지 않는 경우를 제외하고 동일한 단계를 사용하여 구현할 수 있습니다. 두 가지 방법에 동일한 단계를 사용하면 추가 오버헤드 없이 두 방법 사이를 쉽게 보간할 수 있습니다. 단계(d)의 회전은 단순히 스탠스 단계 중 0, 비행 단계 중 1 및 풋롤 단계 중 선형 보간 값으로 적용된다.

설명한 방법으로 얻은 발 정렬은 런타임에 접지가 기준 지면과 일치하기만 하면 조정된 포즈가 원래 동작과 동일하다는 점에서 원래 동작에 충실하다.

C. 역운동학

위에서는 발과 엉덩이의 위치와 정렬을 찾는 방법이 설명되었다. 엉덩이와 발목 사이의 뼈의 정렬은 역운동학을 사용하여 계산된다. IK 방법의 세부 사항은 본 연구에서 주요 초점이 아니며, IK 시스템은 단순히 관절 각도(포즈)의 원래 구성뿐 아니라 힙과 발목 위치를 취하는 블랙박스처럼 취급되며 출력물이 엉덩이와 발목 제약에 맞는 관절 각도(포즈)의 조정된 구성을 생성한다. 여기에는 다양한 IK 알고리즘을 사용할 수 있으며, 다양한 모양의 결과를 생성할 수 있다. 그러나 본 연구에서 설명한 방법에 대해 전혀 사용할 수 있으려면 IK 알고리즘이 특정 조건을 충족해야 한다.

- 각 프레임에서 IK 조정은 이전 프레임에서 IK 조정된 자세를 무시한 상태에서 처음부터 수행해야 합니다. 이것은 경로 의존성을 피한다. 즉, 주어진 시간 동안 동일한 입력 매개 변수가 다리가 어떻게 포즈에 들어갔는지에 관계없이 항상 동일한 출력 포즈를 취하게 된다.
- IK sovler는 이전 프레임의 IK 조정된 포즈를 입력으로 사용하는 대신 원래 애니메이션에서 샘플링하여 현재 혼합 가중치에 따라 혼합된 현재 프레임에 대해 조정되지 않은 원래 포즈를 사용해야 합니다. IK sovler는 엉덩이와 발목의 제약을 맞추면서 IK 조정된 포즈가 가능한 한 원래의 포즈와 일치하도록 시도해야 한다. 일반적으로, 이것은 조정된 관절 각도가 가능한 한 가깝게 원래 관절 각도와 일치하도록 하는 것을 수반한다. 그것이 정확히 무엇을 의미하는지는 다른 알고리즘으로 다르게 해석될 수 있다.
- 알고리즘이 모든 프레임에서 새로운 조정된 포즈 계산을 시작하더라도 시간이 지남에 따라 떨림 또는 비연속적인 결과를 생성하지 않기 위해, 알고리즘의 요구 사항은 유사한 원래 포즈도 항상 유사한 조정된 포즈로 귀결된다는 것이다.

구현된 시스템은 두 개의 매우 간단한 역운동학 솔루션을 사용한다. 분석된 해결책은 엉덩이와 발목 관절 사이에 두 개의 뼈만 있을 때 (인간은 그렇듯) 사용되는 반면 초기 수치 해결책은 엉덩이와 발목 사이에 두 개 이상의 뼈가 있을 때 사용된다. 그러한 IK 솔루션 중 어떤 것도 회전 각도의 제약도 고려하지 않고, 단지 가능한한 원래 동작의 포즈에 가까운 포즈를 찾으려고 시도한다.

역운동학은 다리 당 프레임 당 두 번 수행된다 (섹션 7.4.3 참조). 이는 분석적인 sovler를 사용할 때 매우 효율적입니다. 두 개 이상의 세그먼트가 있는 다리의 경우, 두 개의 IK 단계를 하나로 결합하기 위해 적절한

제약 조건 처리와 함께 더 나은 수치 sovler를 사용할 수 있다.

5. 걷기 시작 및 중지

이 장에 설명된 방법을 통해 캐릭터의 각 발이 지상의 임의의 위치에서 다른 위치로 한 걸음씩 이동할 수 있습니다. 이 유연성이 확립되면 발이 고르지 않은 지형에서 올바르게 발을 디딜 수 있는 것 이상의 용도로 사용할 수 있습니다. 이 절에서는 서서 걷거나 걷거나 걷기에서 서기로 되돌아오는 절차적 전환을 만드는 간단한 방법을 제안한다.

캐릭터가 가만히 서 있고 캐릭터가 걷거나 달리는 동작에는 내재된 비호환성이 있다. 두 발은 가만히 서 있을 때 항상 땅바닥에 꼿꼿이 서 있고, 걷거나 달리는 동작에서는 모든 발이 동시에 단단히 접지되는 일이 없다. 이 때문에 기존의 혼합과 크로스페이딩 기술만으로는 입석과 보행 사이의 전환이 제대로 이루어질 수 없다. 이 문제를 해결하기 위한 전통적인 접근 방식은 서서 걷는 것과 뒤로 걷는 것 사이의 변화에 대한 예시적인 움직임을 갖는 것이다. 그러나, 그것은 캐릭터가 전환 동작이 시작되는 워크 사이클 동안에만 정지할 수 있다는 드로백을 가지고 있기 때문에 주어진 시간에 정지할 수 없다.

A. 분리된 다리 사이클

여기서 제안된 방법은 전환 예제 모션이 필요하지 않은 자동화된 전환에 기초한다. 대신 캐릭터가 걸음을 멈추고 있을 때 모든 다리가 차례로 정지할 수 있도록 다른 다리의 개별 다리 주기 타이밍이 수정된다. 위에서 설명한 방법에서 다리의 다리 주기는 일반적으로 동기화된 방식으로 회전한다(67페이지의 그림 7.1a 참조). 그러나 다리 주기는 별도로 처리되기 때문에 전체 동작 주기와의 동기화는 원할 때 중단될 수 있다. 캐릭터가 멈춰야 할 때, 각 레그 사이클은 스탠스 시간에 도달하면 단순히 회전을 멈출 수 있고, 반면 다른 레그 사이클은 각각의 스탠스 시간에 도달할 때까지 계속 회전한다.

구현된 시스템에서 시작과 중지 사이의 전환은 몇 가지 간단한 규칙을 따릅니다. 주어진 다리의 다음 단계의 길이가 주어진 한계치보다 낮으면, 다리 주기는 일단 그것의 스탠스 시간에 도달하면 "주차"(정지 상태로 놓임)된다. 다리가 이미 주차되어 있고 다음 단계의 길이가 일정 임계값을 초과하는 경우, "예비" 레그 사이클 값이 스탠스 시간을 지나면 레그 사이클이 "주차 해제"(다시 회전하기 시작)됩니다. "예비" 레그 사이클 값은 주차되지 않았다면 전체 모션 사이클과의 동기화에 따라 가졌을 값이다.

6. 결과

이 장에 설명된 준절차적 애니메이션 방법은 다양한 골격(다양한 인간과 동물 모델), 다른 양의 다리(2 및 4개의 다리), 다양한 양의 예시 동작을 가진 다양한 캐릭터에서 사용되었다(자세한 내용은 8장 참조). 유연한 동작을 생산한다는 명시된 프레임워크 목표가 충족된 것으로 밝혀졌다. 모든 테스트된 문자에 대해 반시술적 방법은 직선, 곡선, 사이드 스텝 및 후진 동작과 그 사이의 모든 것, 임의의 스텝과 경사를 포함한 고르지 못한 지형에 대한 동적 어댑테이션, 그리고 서 있거나 걷거나 달리는 사이에 자연스러운 전환을 생성할 수 있습니다. 게다가, 그 방법들은 어떤 형태의 발 미끄러짐도 막는다.

예제 동작에 대한 조정된 운동의 충실성을 시험하기 위해, 조정된 동작은 예제 동작의 기본 조건과 정확히 일치하는 조건에서 예제 동작과 그래픽으로 비교되었다(8.4장 참조). 수행한 비교에서 측정 가능한 차이는

전혀 발견되지 않았습니다.

7. 준절차적 애니메이션의 요약

캐릭터가 울퉁불퉁한 지형에서 어떤 방향으로든 걷거나 달릴 수 있도록 각 프레임에서 준절차적으로 모션 조정을 수행하여 캐릭터가 임의의 위치에서 또는 임의 위치로 스텝을 밟도록 합니다. 각 다리의 경우, 한 단계 도약해야 하는 지상의 발자국이 계산되며, 이전 발자국은 알려져 있고 다음 발자국은 캐릭터의 현재 속도와 회전 속도에 기초하여 연속적으로 예측된다. 이전부터 다음 발자국까지의 풋베이스 궤적은 발자국의 위치와 정렬, 캐릭터의 움직임, 그리고 예 동작의 표준화된 풋베이스 궤적의 혼합 결과에 기초하여 계산된다. 풋 얼라인먼트 자체는 원래 모션에서 보존할 풋 얼라인먼트의 속성을 결정하는 간단한 모델을 기반으로 계산됩니다. 마지막으로, 캐릭터 엉덩이의 고도는 각 다리의 굽힘 양을 원래 동작의 포즈에 최대한 가깝게 유지하기 위해 조정할 수 있다.

설명된 방법은 다리, 발, 스텝 등의 명확한 개념을 가진 다리 이동 모델을 기반으로 한다는 점에서 절차적이다. 그러나 생성된 모션은 런타임 조건(문자와 지형의 속도)이 원래 모션의 기준 조건과 더 유사할수록 합성 모션이 원래 모션과 더 유사하다는 점에서 완전히 데이터 중심이다. 조건이 정확히 일치하면 합성된 모션이 원래 모션과 정확히 일치합니다.