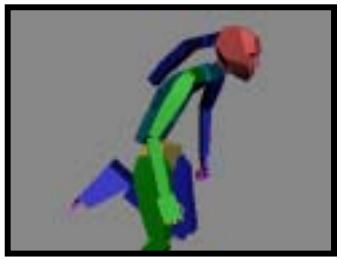
게임에서의 실시간 애니메이션 이해와 응용

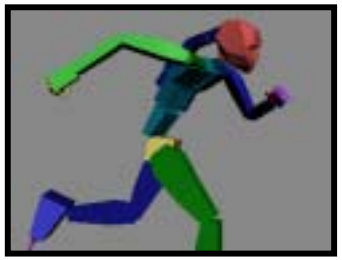
Super Mario Sunshine - Nintendo

KGC2004 프로그래밍 세션

발표자 : 김성익(noerror@hitel.net)
2004.10.16

I. INTRODUCTION



애니메이션 ?

일반적인 메시 모델에서의 애니메이션 표현

+ 모델 (메시, 텍스처) 교체

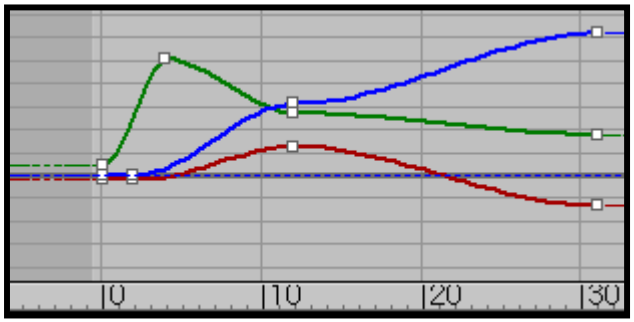
자원의 제약으로 제한적으로 사용된다. (자유도가 높지만 자원 활용도가 떨어지고, 개발 비용도 많이 든다.)

+ 메시의 버텍스의 움직임

버텍스 모핑 (자원 활용도 떨어짐, 퍼포먼스 불이익이 생길 수 있음. 제한적으로 사용됨.)
키프레임 애니메이션 (!!)

키프레임 애니메이션 ?

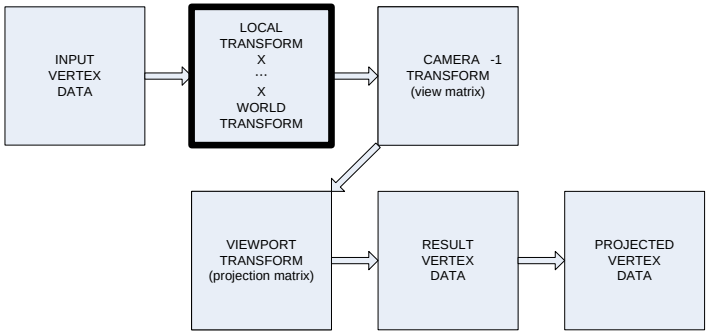
메시의 버텍스와 페이스 정보는 유지한 채로 회전, 이동의 TM 매트릭스를 이용해서 메시의 버텍스를 움직인다.
특정 프레임(혹은 시간)의 회전 값과 , 위치 값의 키 세트를 가진다.



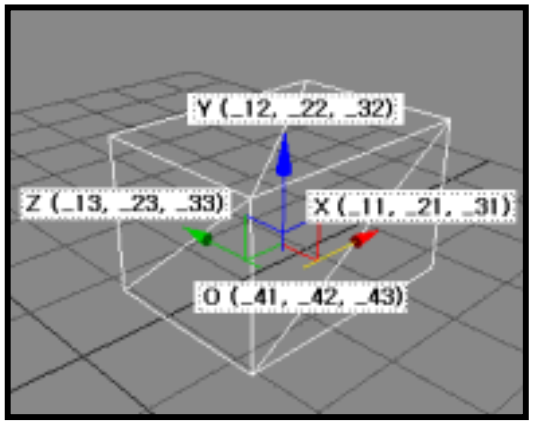
II. BASIC TECHNIQUE
A. 키 프레임 애니메이션 구성

TM 애니메이션

입력 메시데이타를 변형할 필요없다. (하드웨어 가속의 장점을 활용할 수 있다.)



일반적으로 결과는 행렬(matrix)의 형태가 된다.



TM 애니메이션의 구성

+ 시간 (time, frame)

어느 시간의 키(key) 정보인가.

+ 회전 (rotation)

3x3 행렬, 사원수(quaternion)의 형태, 혹은 오일러 각 (eular angle)

$$\begin{vmatrix} \_11 & \_12 & \_13 & 0 \\ \_21 & \_22 & \_23 & 0 \\ \_31 & \_32 & \_33 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

+ 이동 (translation)

벡터 (vector) 의 형태. 위치 값

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Xt & Yt & Zt & 1 \end{vmatrix}$$

+ 스케일 (scaling)

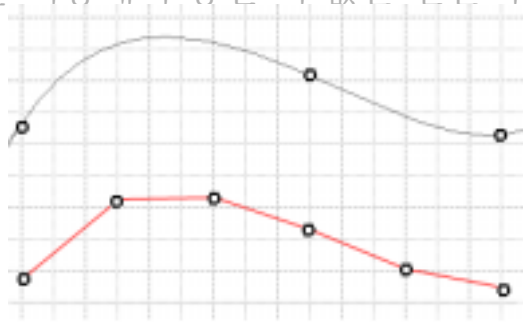
$$\begin{vmatrix} Xs & 0 & 0 & 0 \\ 0 & Ys & 0 & 0 \\ 0 & 0 & Zs & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

B. 보간 (1)

제한된 애니메이션 키 정보

보간(Interpolation)이란

두 지점의 중간 값을 얻는 것으로 애니메인션에서 보간을 이용해서 중간 키 값을 얻을 수 있다.



+ 스플라인보간 (Spline Interpolation)

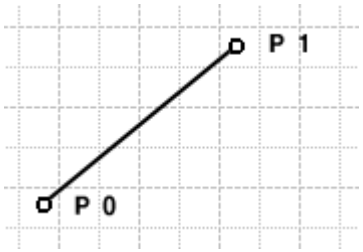
제어 점으로 곡선 모양의 중간 값을 얻는 방식

+ 선형보간 (Linear Interpolation)

두 값 사이를 직선으로 얻는 방식으로, 계산이 간단하여 실시간에서 사용한다.

선형보간 (Linear Interpolation)

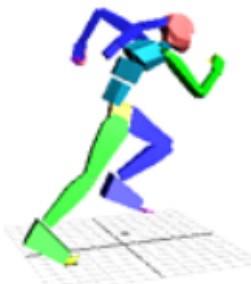
기울기가 일정한 사이 값을 구함



$$P(t) = P_0 * (1-t) + P_1 * t = P_0 + (P_1 - P_0) * t$$

단, $0 <= t <= 1$

```
D3DXVECTOR3 Lerp(const D3DXVECTOR3 * v1,
                 const D3DXVECTOR3 * v2, float t)
{
    D3DXVECTOR3 v;
    v.x = v1->x + (v2->x - v1->x) * t;
    v.y = v1->y + (v2->y - v1->y) * t;
    v.z = v1->z + (v2->z - v1->z) * t;
    return v;
}
```



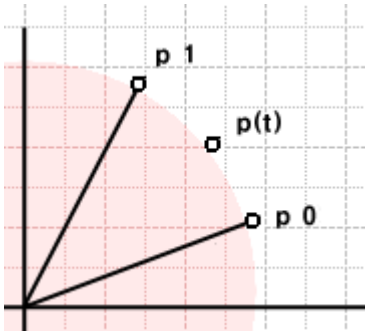
보간 (2)

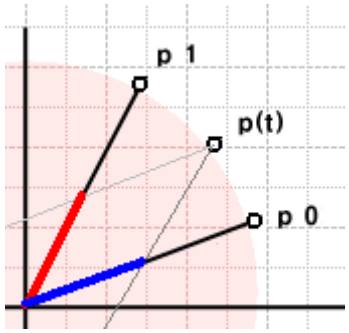
구면 선형보간 (Spherical Linear Interpolation)

매트릭스나, 쿼터니언 같은 회전 성분의 경우 단순히 값을 보간해서 중간 회전값을 얻을 수 없다.

매트릭스는 회전 보간이 힘들고, 오일러각은 짐벌락(Gimbal Lock)이 발생하므로, 쿼터니언 형태로 보간 한 후, 매트릭스로 변환한다

구면 보간은 벡터의 길이가 일정하도록 회전 량을 보간한다.





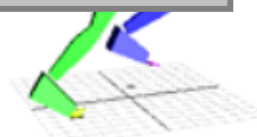
P(t) = A(t) P0 + B(t) P1

A(t) = sin(rad * (1-t)) / sin(rad)

B(t) = sin(rad * t) / sin(rad)

단, |P0| = |p1|, 0 <= t <= 1, rad는 p0, p1 사이각

```
D3DXQUATERNION Slerp(const D3DXQUATERNION * qa,
                     const D3DXQUATERNION * qb, float t)
{
    D3DXQUATERNION qt;
    float cr, sr, rad, t1, t2;
    cr = qa->x * qb->x + qa->y * qb->y +
         qa->z * qb->z + qa->w * qb->w;

    if (cr < 0.0f) {
        qt.x = -qa->x;
        qt.y = -qa->y;
        qt.z = -qa->z;
        qt.w = -qa->w;
        cr = -cr;
    }   else {
        qt.x = qa->x;
        qt.y = qa->y;
        qt.z = qa->z;
        qt.w = qa->w;
    }

    rad  = (float) acos(cr);
    sr   = (float) sin(rad);

    t1   = (float) sin(rad * (1.0f - t)) / sr;
    t2   = (float) sin(rad * t) / sr;

    qt.x = qt.x * t1 + qb->x * t2;
    qt.y = qt.y * t1 + qb->y * t2;
    qt.z = qt.z * t1 + qb->z * t2;
    qt.w = qt.w * t1 + qb->w * t2;
    return qt;
}
```



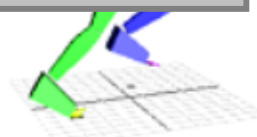
보간 (3) - 키 보간 예제

적용 예제 (1)

+ 3개의 회전, 이동의 키를 설정한 후에 중간 타이밍 값을 얻어서 보간한다.

+ 예제 다운로드 (소스 및 실행파일)
http://digibath.com/repository/kgc04_anim.zip

```
void Render(float time)
{
    D3DXQUATERNION rot;
    D3DXVECTOR3 pos;
    D3DXMATRIX mat;
    int i;

    //   Find Key
    //
    for(i=0; i<SAMPLEKEYN-1; i++)
        if (_samplekey[i+1].t >= time)
            break;

    //   Calculate Weight
    //
    t = (time - _samplekey[i].t) /
        (_samplekey[i+1].t -_samplekey[i].t);

    //   Calculate Interpolated Rotation
    //
    rot = Slerp((D3DXQUATERNION*)&_samplekey[i].rot,
                (D3DXQUATERNION*)&_samplekey[i+1].rot, t);

    //   Calculate Interpolated Position
    //
    pos = Lerp((D3DXVECTOR3*)&_samplekey[i].pos,
               (D3DXVECTOR3*)&_samplekey[i+1].pos, t);

    //   Calculate TM
    //
    D3DXMatrixAffineTransformation(&mat, 1.0f, NULL,
                                   &rot, &pos);

    //   Render
    //
    g_pD3DDevice->SetTransform(D3DTS_WORLDMATRIX(0), &mat);
    g_pD3DDevice->SetVertexShader(g_pTeaPot->GetFVF());

    g_pTeaPot->DrawSubset(0);
}
```

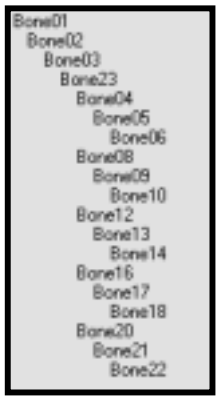


C. 계층적 구조

계층적 구조 (hierarchical structure)

전통적으로 애니메이션 모델에서 각 오브젝트는, 상위 오브젝트에 구속되어 애니메이션 되는 계층적인 형태를 가진다.

오브젝트는 부모(parent) – 자식(child)의 관계로 연결된다.



렌더링 수도 코드 (pseudo code)

```
렌더링(오브젝트, 부모월드매트릭스)
{
    월드매트릭스 = 오브젝트.로컬매트릭스 * 부모월드매트릭스
    오브젝트.렌더링(월드매트릭스)
    for(i=0; i<오브젝트.자식수; i++)
        렌더링(오브젝트.자식[i], 월드매트릭스)
}
```

각 오브젝트의 애니메이션 정보는 로컬 매트릭스에 적용된다.

적용 예제 (2)

+ bip 달리기 동작을 읽어와서 애니메이션을 적용시킨다.

버텍스 좌표가 로컬 좌표가 아니므로 렌더링시 로컬 좌표로 변환하는 invmat를 원드 매트릭스 앞에 곱한다

오브젝트 연결은 자식 리스트를 가지고 있지 않고 자식(child)을 연결(next) 시키는 구조 (결과 동일)

III. ADVANCED TECHNIQUE
A. 모션 블랜딩 (Motion Blending)

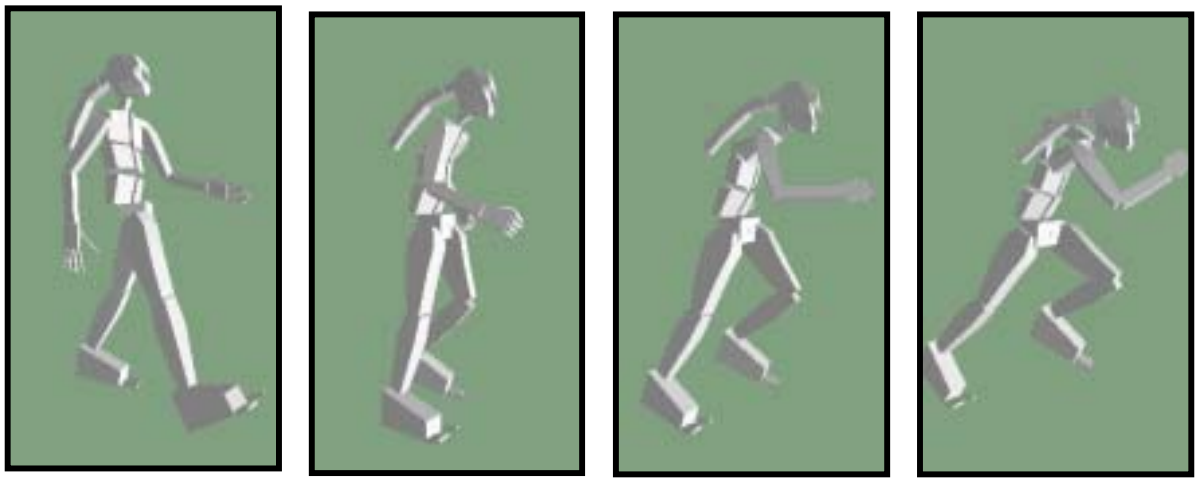
모션 연결

걷기 동작에서 달리기 동작으로 모션을 바꿀 경우 갑자기 달리기 동작이 나올 경우 애니메이션이 자연스럽지 못하다.

모션 간의 키를 보간하면 중간 동작을 만들 수 있어서 자연스럽게 동작을 연결할 수 있다.

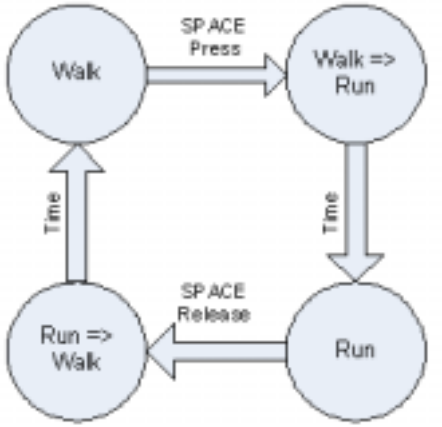
적용 예제 (3)

+ 스페이스바를 누르면 걷기와 달리기 동작으로 전환한다.



첫번째 그림과 마지막 이미지는 각 걷기와 달리기 모션의 이미지이고, 중간에 두 장의 이미지는 각 25%, 50% 동작이 섞인 동작의 이미지다.

반응에 대한 처리는 간단하게 하기 위해 상태 머신으로 구성했다.



B. 역기구학 (1)

Forward Kinematics

일반적인 애니메이션의 방식으로, 오브젝트에 회전, 이동의 키가 있는 형태다.
상위 오브젝트는 하위 오브젝트의 영향을 받지 않는다.

역기구학(Inverse Kinematics)

애니메이션 정보는 특정 오브젝트 (End Effect)의 목표점으로 상위 오브젝트는 하위 오브젝트의 영향을 받는다. (영향을 받는 최상위 오브젝트를 Chain Root 라고 한다.)

게임에서 IK를 이용하면 좀 더 풍부하고 동적인 애니메이션이 가능해진다.

ICO – Sony

IK의 특성

+ 유일한 해가 존재하지 않는다.

결과물의 목표는 단순히 EndEffect가 목표점에 도달하는 것이며, 다양한 해가 존재한다.

+ 해가 존재하지 않을 수도 있다.

목표가 도달 불가능한 경우에는 해가 없다.

게임에서의 IK 해 (IK Solver)

기계학(mechatronis)분야에 비해 게임의 IK solver는 수치적인 정확도보다는 적은 계산, 처리의 단순함에 더 비중을 둘 수 있다.

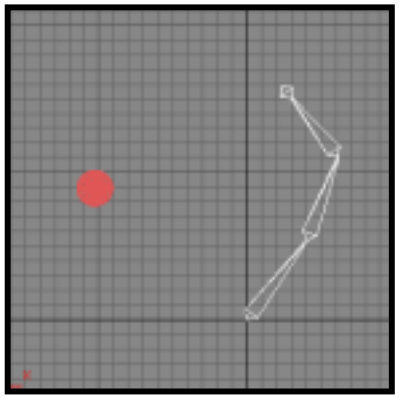
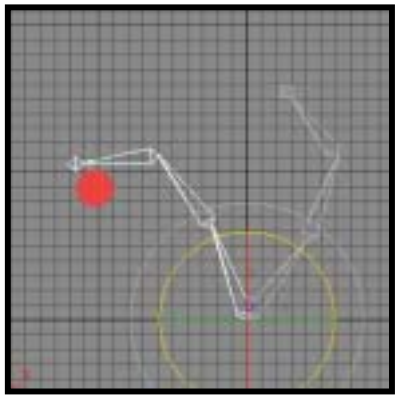
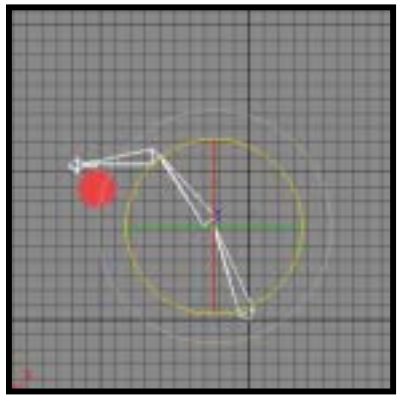
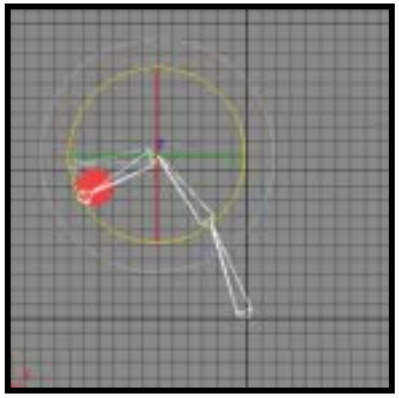
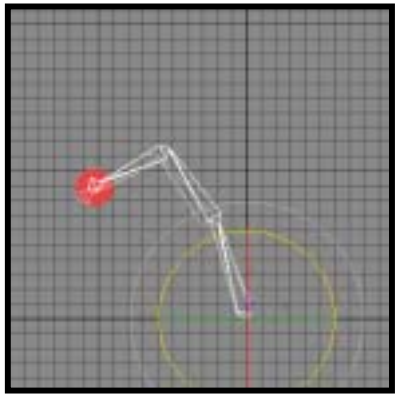
수치적인 해결보다는 휴리스틱(heuristic)한 방법으로 해를 근사 하는 방법이 더 적합하다.

역기구학 (2)

IK Sover

+ 재귀적 접근

Chain Root의 오브젝트부터 시작해서 상위 오브젝트로 이동하면서 End Effector와 타겟점이 일직선 상에 있도록 오브젝트를 회전한다. (같은 함수를 반복적으로 계층적 구조로 적용한다.)

+ 점진적 접근

한 사이클만에 해를 구할 수 없지만 반복할 수록 End Effector는 Target 지점에 가까워진다.

End Effector와 목표점의 거리가 0에 가깝거나 더 이상 가까워지지 않는 상태가 해가 된다.

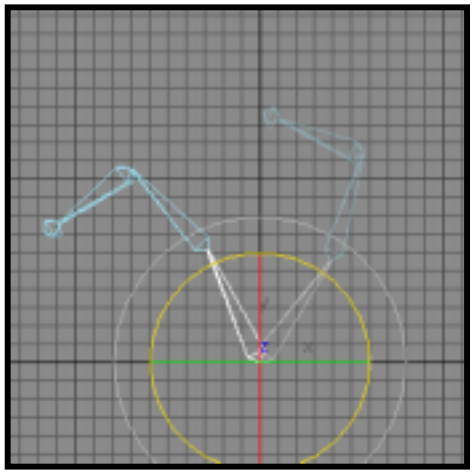
+ 결과 형태

최종 결과는 End Effector가 목표점에 도달하는 모양의 개별 오브젝트들의 회전 값이다.

역기구학 (3)

오브젝트 회전

2d 이거나 회전 축이 X, Y, Z축의 하나라면 회전 트랜스 폼 행렬을 구하기가 쉽지만 일반적인 상황에서는 단순하게 구할 수 없다.



위의 그림처럼 회전한다고 하면 화면에서 나오는 방향으로 약 40도 정도 회전하면 된다.

즉, 회전축과 회전 방향에 대한 정보가 필요하다.

회전 축은 외적의 특성을, 회전 각도는 내적의 특성을 이용한다.

편의상 V_{OLD}, V_{NEW}는 단위 벡터로 가정하면

회전축방향 $V_{CP} = (V_{OLD} \times V_{NEW})$

회전축 Axis = $V_{CP} / |V_{CP}|$

회전각도 Rad = $\cos^{-1}(V_{OLD} \cdot V_{NEW}) * 2$

회전 행렬 구하기

"회전축, 각도 => 사원수 => 행렬"의 과정으로 구한다.

회전축, 각도 => 사원수의 공식

$Q.xyz = Axis / \sin(rad * 0.5)$
$Q.w = \cos(rad * 0.5)$

외적의 크기가 sin(회전각도)인 점을 이용해서 회전 각도가 절반인 V_{MID} 를 구하면

$V_{MID} = Normalize(V_{OLD} + V_{NEW})$

$Q.xyz = CrossProduct(V_{MID}, V_{NEW})$
$Q.w = DotProduct(V_{MID}, V_{NEW})$

로 직접 사원수를 구할 수도 있다.

사원수를 이용하여 행렬로 변환하면 되며, 만약 회전 값을 사원수로 저장하고 있다면 비용이 적은 사원수 곱으로 회전 값을 누적 시킨다.

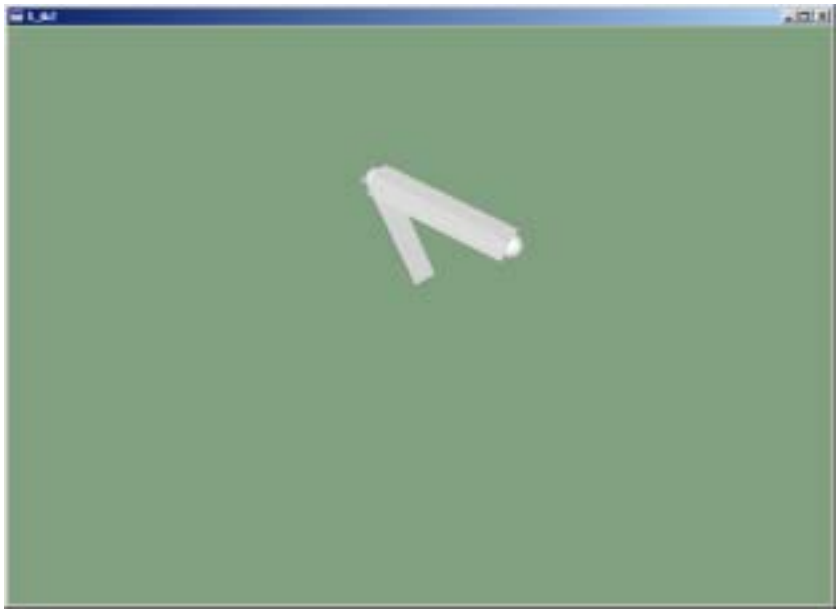
역기구학 (4)

적용 예제 (4)

임의의 회전 값과 위치 값을 가지는 오브젝트가 target을 향하도록 회전한다.

적용예제 (5)

계층적인 구조에서의 역기구학을 구현한다.



구속 (contrain)

자유 회전이 가능한 오브젝트가 아니라면 축이나 각도에 구속을 받게 된다.

이 때는 회전 축의 값과 각도 값을 수정해서 구속시킬 수 있다.

+ 회전축 제한

회전 축을 고정하기 위해서 V_{OLD}, V_{NEW}에 회전축 성분을 제거한다.

$$V_{OLD}' = V_{OLD} - (V_{OLD} \cdot V_{FIXED}) * V_{FIXED}$$
$$V_{NEW}' = V_{NEW} - (V_{NEW} \cdot V_{FIXED}) * V_{FIXED}$$

+ 회전량 제한

급격한 회전각을 제한하기 위해선 Q.rad 가 $Cos(RAD_{MAX} / 2)$ 보다 작은 경우에

$$rad\_2 = acos(Q.rad)$$
$$Q.xyz' = Q.xyz * Sin(rad\_2) / Sin(RAD_{MAX} / 2)$$
$$Q.w' = Cos(RAD_{MAX} / 2)$$

C. 본을 이용한 강체 물리학

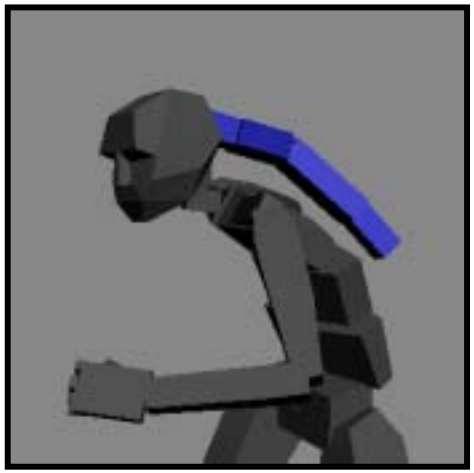
게임 내의 물리학

하드웨어 가속을 고려할 경우 셰이더를 이용하지 않는 버텍스 레벨의 시뮬레이션은 최적이 아니다.

하드웨어 가속의 고려할 경우 TM 레벨에서의 오브젝트의 물리학 적용이 유리하다

적용

오브젝트의 연결부를 조인트(joint)로 처리하여 각 조인트를 파티클로 가정하여 모델링한다.



파티클 간의 거리가 일정하도록 구속하는 조건으로 자유롭게 물리를 적용한다.

물리가 적용된 후의 파티클의 구성을 토대로 본의 TM 매트릭스를 설정한다. (역기구학의 회전 행렬 구하는 방식을 활용한다.)

적용 분야

쇠사슬, 포니 테일(ponytail), 래그돌(ragdoll) 등의 조인트가 있는 강체 물리에 적당하다.

Hitman – IO Interactive

IV. TECHNICAL ISSUE (1)

쳐다보기 (lookat)

게임에서 쳐다보기 처리는 캐릭터에 생명력을 주는 가장 쉽고 효과적인 방법이다.

The Legends of Zelda : wind waker - Nintendo

아주 제한적이고 예측 가능하기 때문에 머리 오브젝트에 대해서 오일러 회전각을 이용하여 행렬을 구하는 것도 무리가 없다.

적당히 보간을 이용해서 자연스럽게 쳐다보는 행동을 ON/OFF 하면 좋다.

애니메이션 공유와 활용

보통 캐릭터의 경우 오브젝트의 계층 구조가 동일할 경우 다른 모델의 애니메이션을 공유할 수 있다.

캐릭터 애니메이션의 경우 대부분 회전 애니메이션이고, 이것을 공유하는 것이기 때문에 본의 길이가 다르다면 (팔이 길다던가 다리가 길다던가) 완전히 동일한 모션을 취한다고 볼 수는 없다.

애니메이션 내부 계층을 그룹화해서 부분별(상체, 하체, ...)로 별도의 애니메이션을 적용할 수 있다.

미끄러짐 없는 이동

애니메이션의 이동량과 상관없이 게임내에서 일정량 좌표를 갱신할 경우 미끄러지는 듯한 현상이 발생한다.

제작하는 애니메이션 자체가 제자리 걸음이 아니라 미끄러짐 없이 이동하도록 개발하며, 최상위 본의 이동치 만큼 좌표를 갱신하고,
갱신한 축의 좌표를 0 로 만들어준다.

모션 블랜딩 중에는 이동치도
가중치에 따라 처리한다.

TECHNICAL ISSUE (2)

사원수 압축

애니메이션이 증가할 경우 메모리의 제약이 생기며, 메모리를 많이 차지할 수록 속도에 영향을 준다.

쿼터니언의 특성을 이용하여 고정 소수점 (Fixed Point) 수로 관리한다. 양자화 (Quantization)

+ 쿼터니언 x, y, z, w는 –1 에서 1 사이 값을 가진다.

값의 범위가 정해져 있다. 즉, 2바이트로 변환한다고 하면 -32768 에서 32767 사이의 범위로 매핑 가능하다.

+ 쿼터니언 (x, y, z, w) 와 (–x, -y, -z, -w)는 같다.

w를 0에서 1사이로 만들 수 있으므로 w 값의 해상도을 더 높일 수도 있다.

$x^2+y^2+z^2+w^2=1$ 의 조건도 활용하면 데이터를 더 줄일 수도 있으나, 실시간으로 사용한다면 간단한 게 좋다.

사원수 압축은 속도 향상의 개념도 있지만, 만약 16바이트의 회전 키를 8바이트로 줄이는 경우, 같은 용량으로 애니메이션 샘플링 주기를 두 배 이상 높일 수 있으므로 훨씬 자연스러운 모션을 얻을 수 있다.

최적화와 엔진 구성

메모리 최적화된 데이터 구조

애니메이션 인터페이스

메모리에 따른 키 최적화

V. 질문 / 답