Practical Animation Techniques

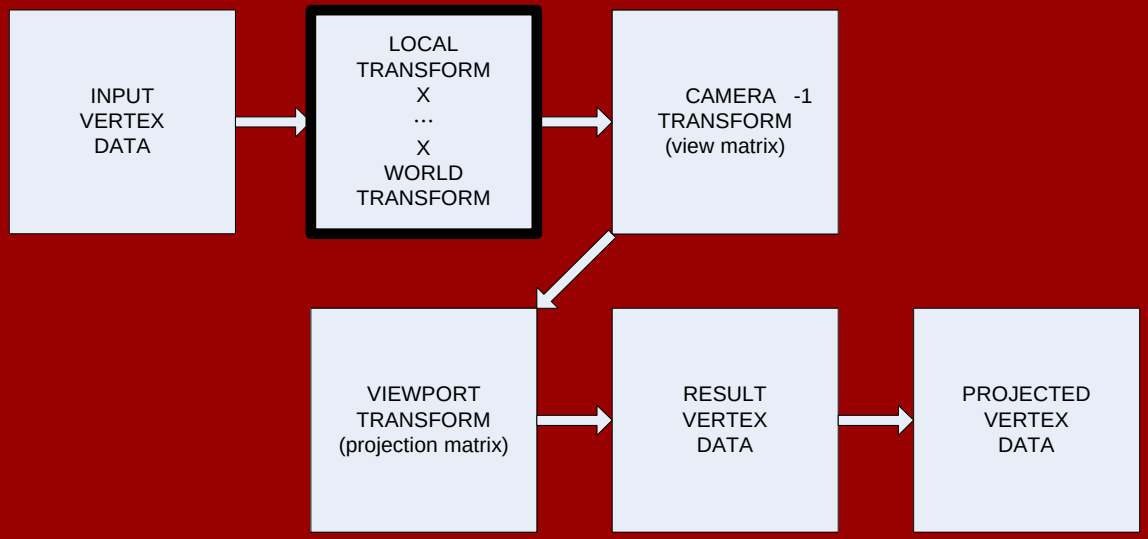
김성익(noerror@hitel.net)
2004.07.04

http://www.digibath.com/noerror

A. Background

basic
기본적인 정점 변환의 단계
변환 행렬의 구성
 확대/축소 변환
 회전 변환
 이동 변환
행렬과 좌표계
계층적 구성

기본적인 정점 변환의 단계
vertex transformations pipeline



- 3차원 폴리곤 그래픽에서 “애니메이션 => 버텍스의 움직임”
- 입력 버텍스 레벨의 애니메이션에는 물리적인 제약이 따름
- 애니메이션을 하기 위해선 트랜스 폼 매트릭스를 활용
- 트랜스 폼 매트릭스를 계산하는 과정은 하드웨어의 가속 대상이 아님

변환 행렬의 구성 1
compounding transformations

* 일반적인 강체변환(Rigidbody Transform)의 구성요소

- 확대/축소(scale)
- 회전 (rotation)
- 이동 (translation)

* 각 요소는 행렬(Matrix)로 표현 가능

* 확대/축소 변환 (scale transformation)

V input = (x, y, z)
V output = V input x M scale
= (x * x', y * y', z * z')

$$M_{scale} = \begin{vmatrix} x' & 0 & 0 \\ 0 & y' & 0 \\ 0 & 0 & z' \end{vmatrix}$$

변환 행렬의 구성 2
compounding transformations

*** 회전 변환** (rotate transformation)

- 회전 행렬은 기본 X축, Y축, Z축이 회전하여 변환된 좌표로 구성
- 행렬을 보고 모양을 읽을 수 있음

$$
\begin{aligned}
V_{input} &= (x, y, z) \\
V_{output} &= V_{input} \times M_{rotation} \\
&= (V_{input} * V_x, V_{input} * V_y, V_{input} * V_z) \\
M_{rotation} &= \begin{vmatrix} V_{x.x} & V_{x.y} & V_{x.z} \\ V_{y.x} & V_{y.y} & V_{y.z} \\ V_{z.x} & V_{z.y} & V_{z.z} \end{vmatrix}
\end{aligned}
$$

*** 이동 변환** (translate transformation)

$$
\begin{aligned}
V_{input} &= (x, y, z) \\
V_{output} &= V_{input} + V_{offset}
\end{aligned}
$$

행렬과 좌표계
local coordinate system

* 확대/축소, 회전과 이동을 동시에 표현하기 위해 Direct3D는 4X4 행렬사용

$$
\begin{aligned}
V_{input} &= (x, y, z, 1) \\
V_{output} &= (x', y', z', 1) \\
M_{rotation+offset} &= \begin{vmatrix} V\,x.x & V\,x.y & V\,x.z & 0 \\ V\,y.x & V\,y.y & V\,y.z & 0 \\ V\,z.x & V\,z.y & V\,z.z & 0 \\ V\,off.x & V\,off.y & V\,off.z & 1 \end{vmatrix}
\end{aligned}
$$

* 벡터 x 변환 행렬 = 변환 행렬로 구성된 좌표계로 변환

* 회전 행령은 사원수(Quaternion)으로 변환가능
* OpenGL의 행렬은 DIrect3D의 전치(Tranpose)한 형태

계층적 오브젝트 구조
hierarchical structures

- 각 오브젝트 변환을 상위 오브젝트에 구속되는 계층적인 형태로 구성
- 오브젝트끼리 부모(parent) - 자식(child)의 링크 구조
- 효율적인 애니메이션이 가능

- 최종 행렬 = 자신의 로컬 행렬 x 부모 오브젝트의 행렬
- 애니메이션 정보 = 오브젝트 로컬 행렬
- 로컬 행렬의 이동성분 = 부모 오브젝트 좌표계에서의 피봇위치

```
void CMeshObject::Render(const Matrix& parentmat)
{
    Matrix mat = m_LocalMat * parentmat;
    RenderMesh(mat);
    for(i=0; i<m_ChildN; i++)
        m_MeshObj[i]->Render(mat);
}
```

```
Bone01
  Bone02
    Bone03
      Bone23
        Bone04
          Bone05
            Bone06
        Bone08
          Bone09
            Bone10
        Bone12
          Bone13
            Bone14
        Bone16
          Bone17
            Bone18
        Bone20
          Bone21
            Bone22
```

B. Animation techniques

Basic
- forward kinematics
- inverse kinematics
- linear interpolation
- spherical linear interpolation
- keyframe animation
- motion transition

Advanced
- inverse kinematics solver
- inverse kinematics – constrains
- quaternion

Game issue
- rigid body physics
- 쳐다보기
- 미끄러짐 없는 이동
- 쿼터니언 압축

기초
basic technique

* Forward Kinematics

- 애니메이션 정보는 각 오브젝트의 최종 변환(회전, 이동, 스케일) 데이타
- 하위 오브젝트(혹은 bone) 은 상위 오브젝트의 영향을 받는다
- 상위 오브젝트는 하위 오브젝트 애니메이션의 영향을 받지 않는다.
- 애니메이션 적용 방향은 상위 오브젝트에서 하위 오브젝트로 전개한다.

* Inverse Kinematics

- 애니메이션 정보는 특정 오브젝트(End Effector)가 도착할 임의의 위치
- End Effector부터 상위 Chain Root 까지의 링크가 영향을 받는다.
- End Effector 상위 오브젝트들은 End Effector의 위치에 영향을 받는다
- End Effector가 임의의 위치가 되는, 각 오브젝트의 변환 해(solve)는 유일하지 않으며 무한히 많을 수 있다.
- 답이 없을 수도 있다

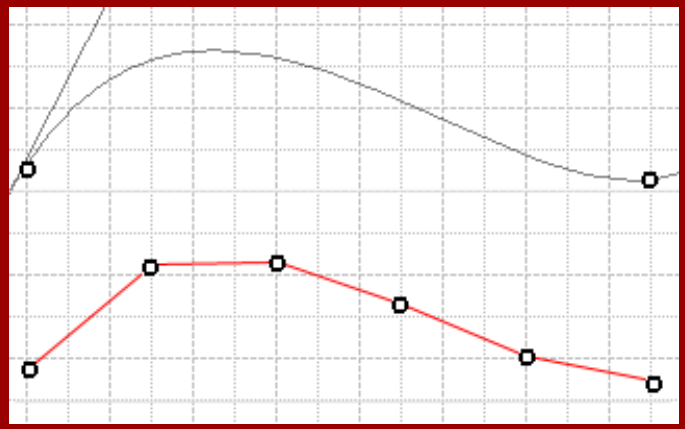
키프레임 애니메이션
keyframe animation

* 애니메이션 정보 = 각 오브젝트의 애니메이션 키(시간, 변환)의 정보

- 시간에 맞는 각 오브젝트의 로컬 변환 값을 세팅

* 키 정보

- 애니메이터가 작업한 키 정보를 익스포트
- spline interpolation으로 애니메이션 키 작업을 한 것을 게임에서 linear interpolation 하면 다른 결과가 나옴
- 적절한 샘플링으로 해결 가능



보간
interpolation

* 자원의 한계로 저장되는 애니메이션 키의 수는 유한

* 임의의 시간에 애니메이션 키가 없는 경우 키 중에 임의의 시간보다 앞의 키 중에서 가장 가까운 키와, 시간보다 뒤의 키 중에 가장 가까운 키를 얻은 후 그 키들의 사이 값으로 애니메이션 키를 구한다.

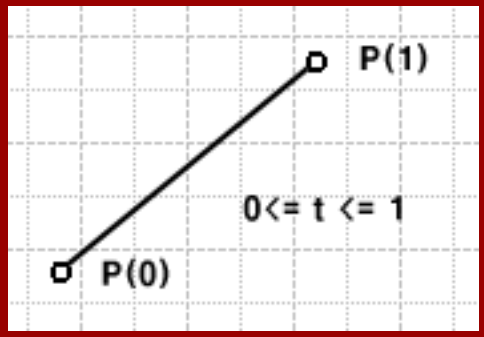
* 두 값의 사이 값을 구하는 것을 보간(interpolation)이라고 한다.

* 애니메이션에서 사용하는 보간은 Spline Interpolation, Linear interpolation 등이 있다.

선형 보간 (LERP)
linear interpolation

* 선형 보간은 변화량이 일정한 보간법이다

- 만약 P(0) = p0, P(1) = p1 이라고 가정
- 0 <= t <= 1
- P(t) = p0 + (p1 - p0) * t



- 벡터는 각 요소 별로 선형 보간 가능

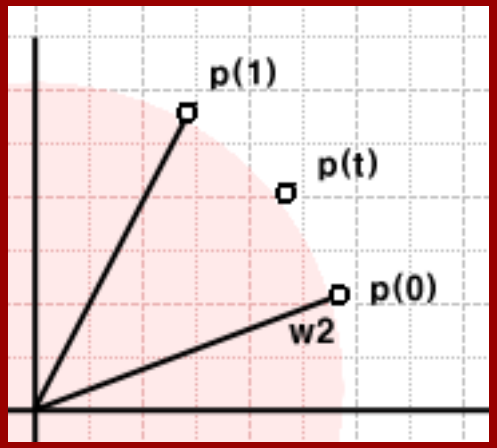
- 행렬의 선형 보간은 회전의 보간과는 다른 의미
- 오일러각으로 사이 값을 구할 경우 각도를 선형보간 가능 하지만 항상 결과가 제대로 나오는 것은 아니므로, 제한적으로 사용된다. (Gimbal Lock 발생)
- 회전의 보간을 위해선 사원수를 사용한다
- 사원수는 spherical linear interpolation을 사용한다

구면 보간 (SLERP)
spherical linear interpolation

* 구면 보간은 벡터 P0 가 길이를 유지한 채로 회전해서 P1가 되었다고 했을 때 회전한 그 사이값을 보간하는 방법

- |P(0)| = |P(1)|
- 0 <= t <= 1
- *rad* = acos(dotproduct(P(0), P(1)))

- P(t) = a(t) P(0) + b(t) P(1)
- a(t) = sin(*rad* * (1-t)) / sin(*rad*)
- b(t) = sin(*rad* * t) / sin (*rad*)



* 회전 성분을 보간하기 위해서 사원수에 대해서 각 성분을 구면 보간한다

자연스러운 모션 전환
motion blending

* 동작 전환시 대응 (동작 A를 취하다 동작 B를 취하는 경우)

- 순간적으로 동작 B로 전이 되는 경우
- 동작 B와 마지막에 취한 동작 A를 전이(transition)하는 경우
- 동작 B와 동작 A의 섞는(blending) 양을 조절하여 전환하는 경우

* 방법적으로는 애니메이션시 키 보간 하는 것과 동일
* 게임 내 자연스러운 동작 전환이 가능함
* 동작에 따라선 전이되는 시간도 조절할 필요 있음

역운동학
inverse kinematics

* IK 사용시 장점

- IK는 동적으로 애니메이션을 생성
- 주변 오브젝트와의 상호작용 가능
- 애니메이션을 풍부하게

* 난점

- 소모하는 CPU 자원에 비해 실제적인 효과가 크게 나지 않을 수 있다
- Constrain등을 섬세하지 하지않으면 이상한 모션이 취해질 수 있다

(시연)

역운동학 – 재귀적 접근

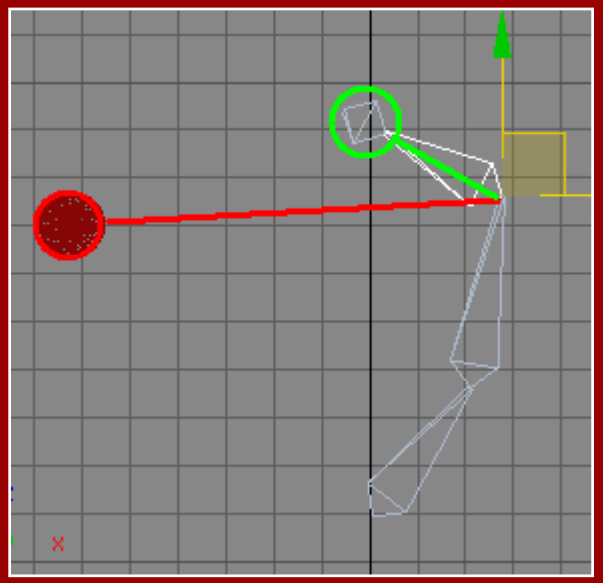
inverse kinematics solver

* 수치적인 해결은 연산량이 많으며, IK 특성상 정확한 해는 존재하지 않는다
* 적은 계산으로 점진적, 재귀적으로 근사적으로 해를 얻는 방법 소개

* 재귀적 루틴

- 하위 오브젝트(혹은 Bone)부터 EndEffect 가 목표점이 나란하게 되도록 회전 (일직선일 때 가장 EndEffect가 목표점이 가장 가까우므로 회전하기 전보다 항상 EndEffect가 목표점과 가까워진다.)
- 다음 오브젝트에 대해서 다시 반복
- ChainRoot 까지 반복

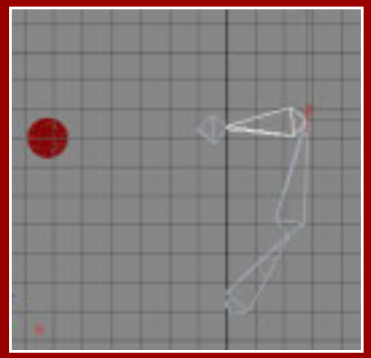
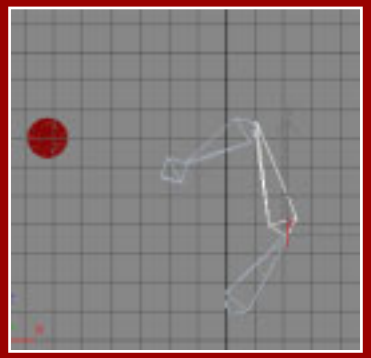
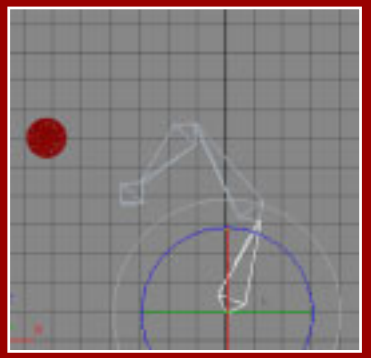
- ChainRoot에서 EndEffect 쪽으로 진행해도 된다. (이 경우가 좀 더 결과가 좋다.)

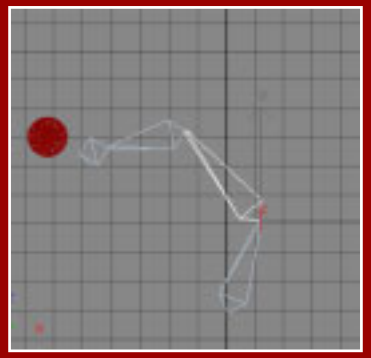
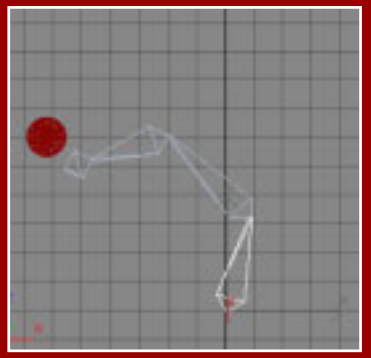


역운동학 – 점진적 해결

inverse kinematics solver

* 연산을 반복할 수록 점차 해답에 가까워진다
* 어느 순간에 해답을 얻을 수 있다

역운동학 – 쿼터니언 적용

inverse kinematics – making quaternion

* 실제적으로 적용하기 위해선 3D상 회전을 적용해야 함
* 쿼터니언의 성질을 이용

- 좌표계 내의 v1이 v2가 되도록 회전할 경우
- x, y, z, w 의 x, y, z 는 v1 x v2 방향
- 주먹으로 표현하면 x, y, z 가 엄지 손가락(회전축), w가 회전량
- v1과 v2가 이루는 각이 rad 이면 x, y, z 의 크기는 sin(rad * 0.5)
- w 는 cos(rad * 0.5)
- 즉, v1과 v2의 반각을 이루는 벡터를 v3 라고 하면 (rad * 0.5 의 특성 이용)
- x, y, z 는 v1 과 v3의 크로스 프로덕트, w 는 도트 프로덕트

- P_1 = Normalize(로컬 좌표계에서의 EndEffector 의 위치)
- P_2 = Normalize(로컬 좌표계에서의 목표점의 위치)
- P_3 = Normalize(p1 + p2)
- x, y, z = CrossProduct(p1, p3), w = DotProduct(p1, p3)

역운동학 - 구속
inverse kinematics - constrains

* 일반적으로 IK는 회전 TM만으로 제한한다
* 회전에도 제약(Constrain)을 줄 수 있다

* **회전각 크기** 제한

- 회전각은 도트 프로덕트로 쉽게 구할 수 있으며, 만약 최대치보다 각도가 큰 경우에는 회전 쿼터니언을 보정한다.
- x, y, z 의 크기는 sin(rad*0.5) 이므로 sin(MAX*0.5) / sin(rad*0.5) 곱해줌
- w 의 크기는 cos(rad*0.5) 이므로 cos(MAX*0.5)로 수정 (부호 주의)

* **회전축** 제한

- 특정 축으로만 회전하도록 구속되어 있다면 로컬 좌표계에서 구한 EndEffector와 Goal 에 구속된 축의 성분을 제거한다

강체 물리학
rigid body physics

* 하드웨어 가속을 고려할 경우 버텍스 시뮬레이션은 최적이 아님
* 렌더링의 가속을 고려하면 TM 레벨에서의 오브젝트의 물리학 적용이 유리

- 기본은 TM 레벨만으로 적용 가능한 물리학에 대해서 시도한다
- 각 조인트를 파티클로 간주하고 물리학 적용하되, 길이는 변하지 않도록 해서 회전 성분만 남긴다
- IK와 마찬가지로 파티클의 이동된 위치를 이용 회전 TM을 구한다

* 쇠사슬, 포니 테일 등의 강체 물리에 적당

* 풍부하고 인터렉티브한 애니메이션이 가능해진다

(시연)

쳐다보기
looking up at face

* 효과
- 애니메이션이 풍부할 수록 기계적이란 느낌을 덜 준다
- 게임에서 쳐다보기 처리는 캐릭터에 생명력을 주는 가장 쉽고 효과적인 방법

* 구현
- 아주 제한적이고 예측가능하기 때문에 머리 오브젝트에 대해서 오일러 회전을 적용하는 것도 무난
- 가중치(Weight)를 준다거나, 스탠다드하게 작성하려면 앞서 다룬 쿼터니언을 이용

미끄러짐 없는 이동
movement based animation

* 애니메이션과 상관없이 일정한 속도로 좌표를 갱신할 경우 미끄러짐 현상 발생

* 이동과 애니메이션 정책

- 제작하는 애니메이션 자체가 제자리 걸음이 아니라 미끄러짐 없이 이동하도록 개발
- 최상위 본의 이동치를 기준으로 처리
- 렌더링 할 때 이동할 축의 값을 강제로 0 으로 세팅 (제자리 걸음하도록)
- 루트본 이동 벡터 구하는 함수가 p 면 p(n) - p(n-1) 의 벡터 중 이동축 성분만 얻어서 이동
- 모션 블랜딩 중에는 이동치도 가중치에 따라 처리

쿼터니언 압축
compressing quaternion

* 애니메이션 데이터가 많아지면 메모리 사용량이 많아
* 메모리 억세스가 많고, 반복적인 부분이 적으며, 데이터량이 많아서 메모리 대역폭의 영향을 받음
* 키 보간을 위해서 대부분 연속된 두개의 데이터를 읽어야 함

* 쿼터니언의 특성을 이용
- x, y, z 는 -1에서 +1 사이값을 가짐
- x, y, z, w 에 모두 음수를 취해도 회전 성분 변화 없음 (w 값의 범위를 0 에서 1사이로 만드는 것도 가능)
- 값의 범위가 제한되므로 충분한 유효 자리수를 확보해서 Fixed Point 로 변환 가능

* x, y, z, w 를 각 2 byte fixed point 로 변환할 경우 메모리를 절반만 차지

질문
question