# Computer Graphics: Character Animation

Dept. of Game Software
Yejin Kim

# Overview

- Computer Animation
- Character Animation
- Tutorials

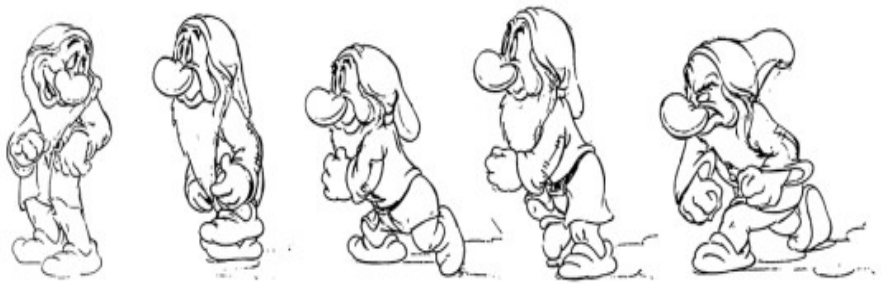
# Computer Animation

- What is Animation?
  - Origin of word: “Animal” or “Anima”에서 유래
  - Animate: “to give life to (생명을 불어넣다)”
  - Time에 따른 object의 space상의 이동
  - Speed: 30 (games) or 24 (films) fps

- Computer animation type
  - Computer-assisted animation
    - 2D & 2.5D
    - Inbetweening (tweening)
    - Inking, virtual camera, managing data, etc.
  - Computer generated animation
    - Key frame animation
    - Physics-based animation

# Computer Animation

- Key frame animation
  - Animation을 구성하는 동작을 구체적으로 명시
  - Key frame: 가장 중요한 장면
    - 숙련된 animator가 제작
    - 주로 시점이 변화거나 중요한 action이 등장하는 scene
  - Inbetweening (tweening) frame: Key frame 사이의 frame
    - Interpolation을 사용하여 computer가 제작
  - 장점
    - Animator가 자유롭게 원하는 scene 생성
  - 단점
    - Animation 품질이 animator의 숙련도에 크게 영향을 받음
    - 실제와 비슷한 동작 생성에 제작 시간이 오래 걸림



# Computer Animation

- Physics-based animation
  - Simulation of physically plausible behaviors at interactive rates
    - e.g. Havok, PhysX, Bullet, etc.
  - 장점
    - Interaction with the environment
    - Less data requirement
  - 단점
    - High calculation
    - Requires complicated rules & constraints
  - Simulation types
    - Rigid/Soft body simulation
    - Fluid simulation
    - Particle systems
    - Flocking
    - Ragdoll

# Character Animation

- Character animation
  - Rigid body animation
  - Articulated body
  - Bone-based(skeletal) animation

# Character Animation

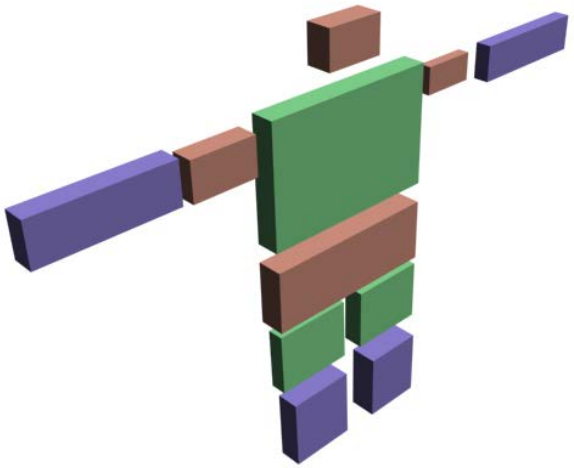
- Rigid body animation
  - Vertex animation (vertex morphing) technique

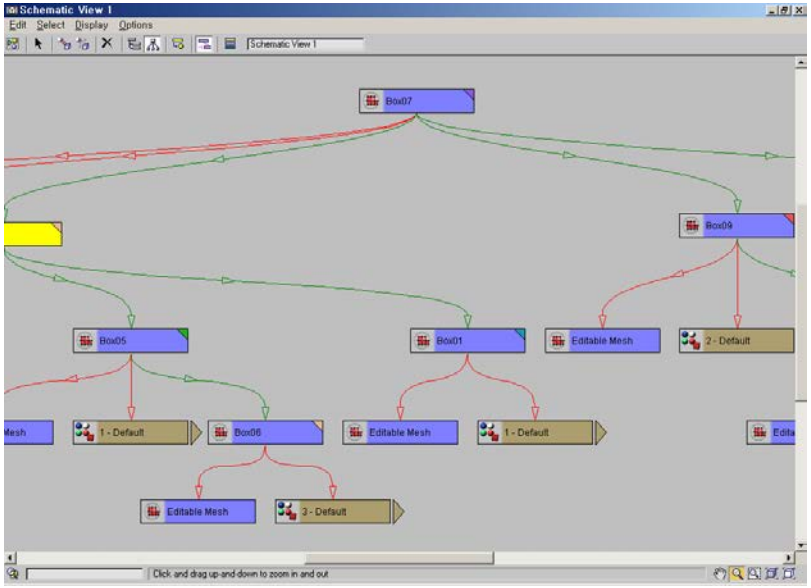
$$V_{world} = (1 - a) \times V_1 + a \times V_2, 0 \leq a \leq 1$$

# Character Animation

- Articulated body: Hierarchical structure
  - Character를 여러 개의 mesh로 나눠서 각 mesh를 부모 자식간의 관계로 상속시키고, mesh마다 이동, 축소, 회전을 포함한 matrix를 매 frame 또는 변화되는 frame마다 저장하여 사용

$$M_{transform} = M_{local} \times M_{animation} \times M_{parent}$$

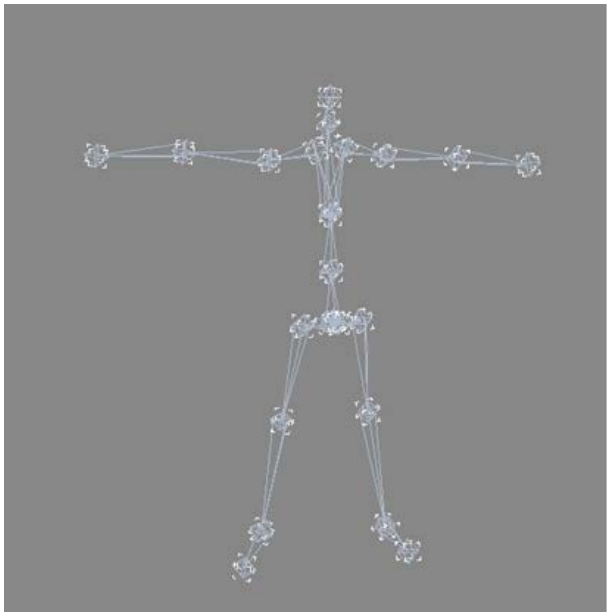


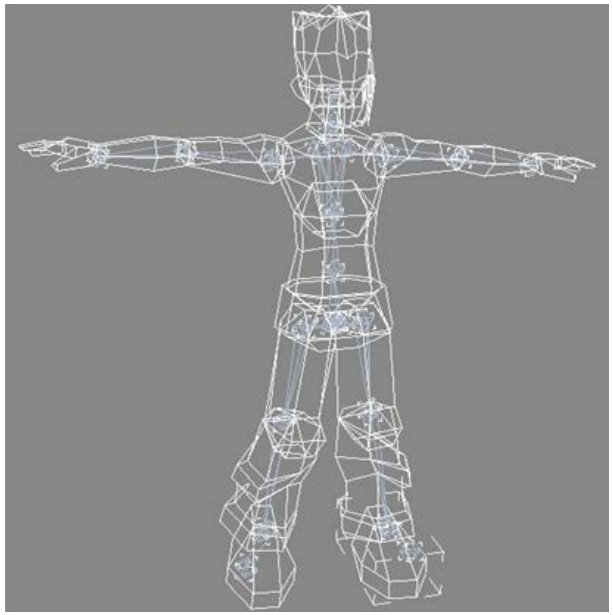


# Character Animation

- Bone-based animation
  - Hierarchical 구조를 가진 mesh model을 효과적으로 제어하기 위해 3D skeleton 생성하고, skeleton에 mesh를 자식으로 붙여 skeleton의 움직임으로 animation을 생성하는 방식

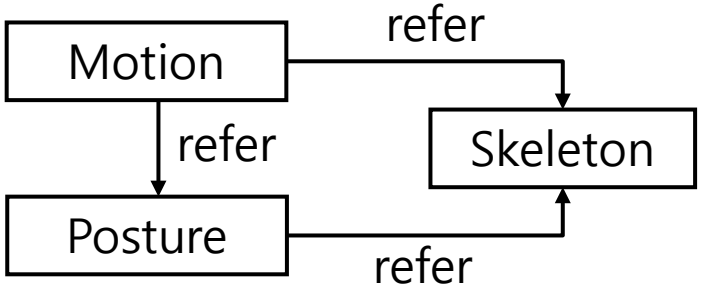
$$M_{bone\_transform} = M_{bone\_local} \times M_{bone\_animation} \times M_{bone\_parent}$$
$$V_{world} = V_{local} \times M_{mesh\_local} \times M_{bone\_transform}$$

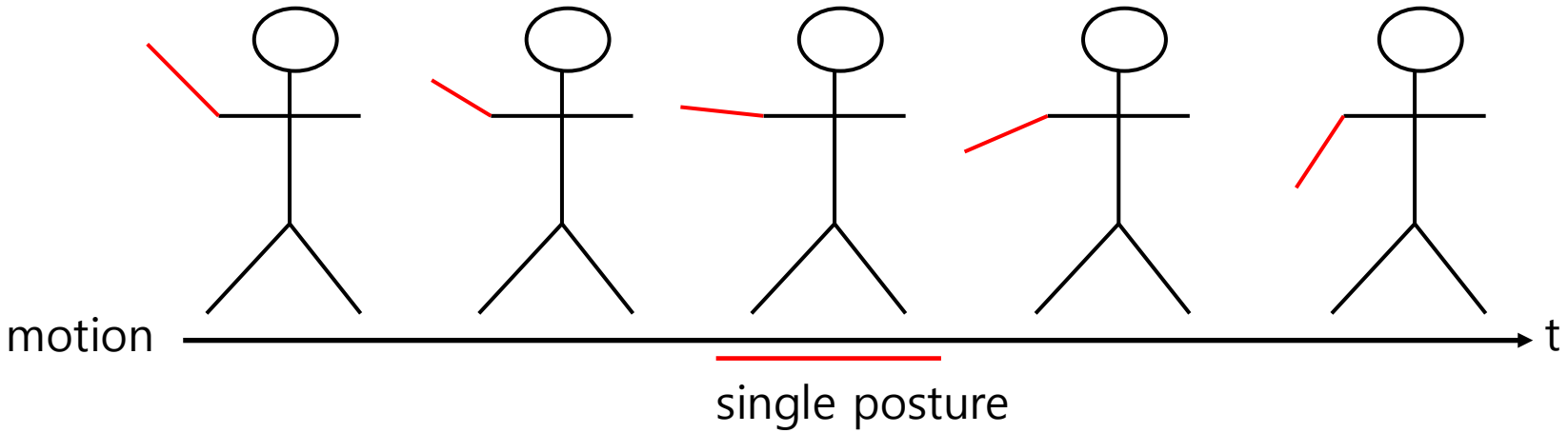




# Character Animation

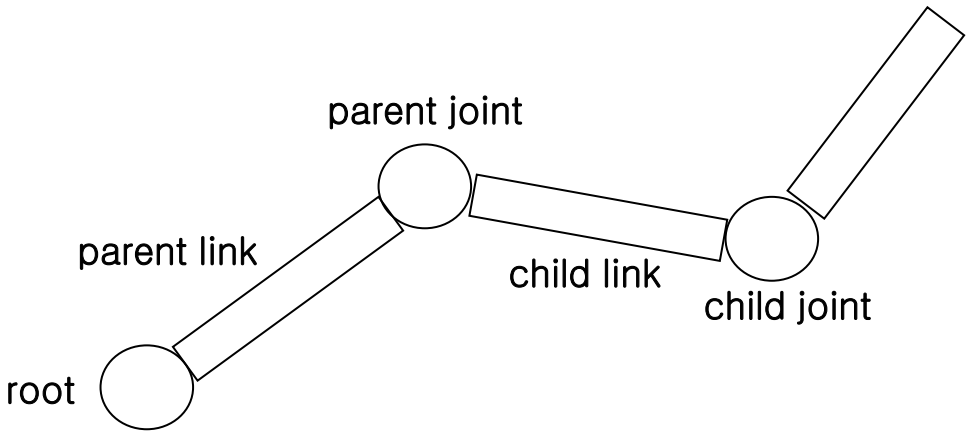
- Data structure for articulated character
  - Skeleton
    - List of joint locations
    - Articulated body model
  - Posture
    - List of joint orientations
    - Forward/Inverse kinematics
  - Motion
    - List of postures
    - Sequence of postures over time

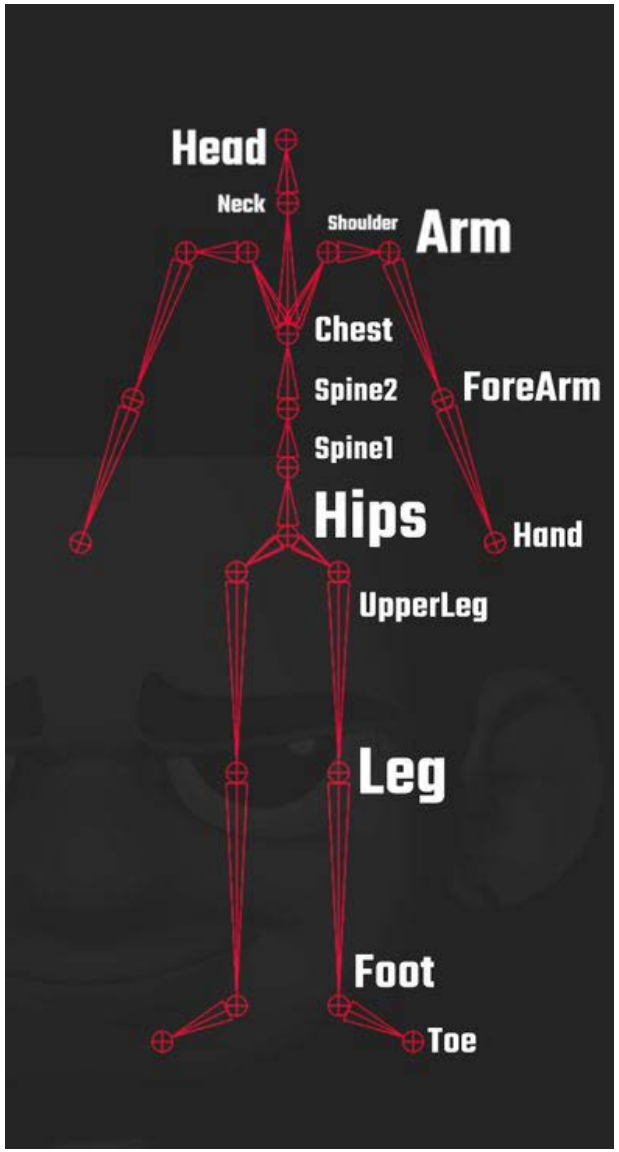


# Character Animation

- Skeleton class
  - Hierarchy of the articulated body
    - Link: a rigid object
    - Joint: connection between two adjacent links
  - e.g. Humanoid skeleton
    - Root: Hips
    - Body parts: Hips → … → Head
      Chest → … → Hand
      Hips → … → Foot

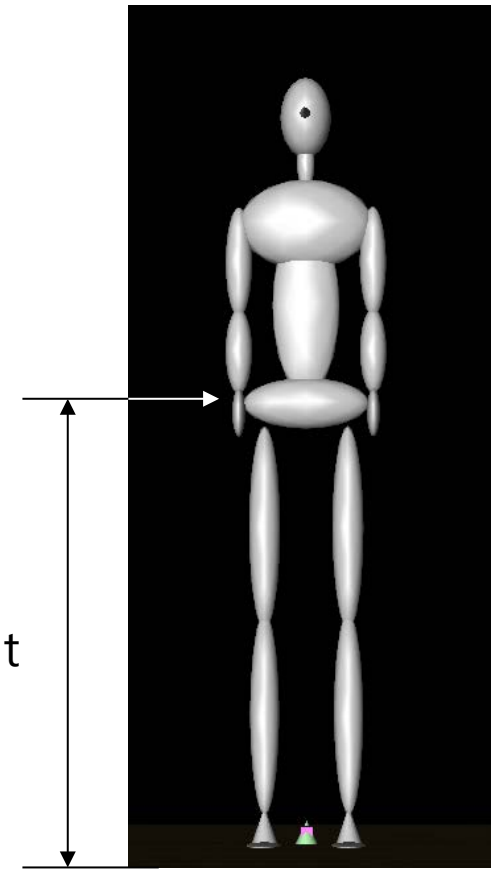


# Character Animation

- Skeleton class
  - e.g. Motion file: ACTOR or ASF
    - Parent coordinate 대비 joints의 위치 이동 값을 포함하고 있음
    - ACTOR file을 loading하여 human object를 초기화
    - e.g. ACTOR data

```
root_height     36.68

pelvis
begin
    displace    ( 0 , 0 , 0 )
end
chest
begin
    parent      pelvis
    displace    ( 0 , 4.38 , 0 )
end
neck
begin
    parent      chest
    displace    ( 0 , 16.35 , 0 )
end
```



# Character Animation

- Skeleton class
  - Describing body configuration
  - The locations of joints : the length of limbs

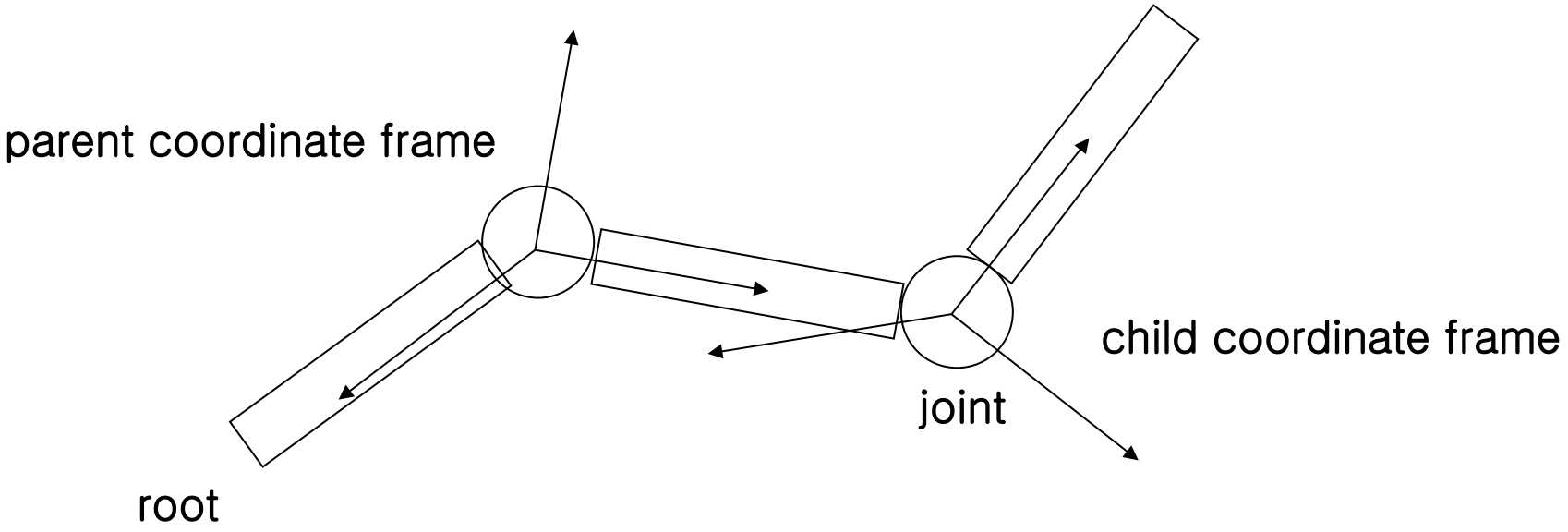
```
class Skeleton {
protected:
          int        num_links;
          int        num_spine_links;
          int        parent_list[SKELETON_NUM_LINKS];
          vector     joint_position[SKELETON_NUM_LINKS];
          m_real     root_height;
}
```

  - Member functions

```
virtual void openSkeleton(char*);
int getNumLinks() const { return num_links; }
static int getLinkNumber(char*);
static char*getLinkName(int);
int getParent(int i) const { return parent_list[i]; }
char isAncestor(int, int);
vector getJointPosition(int i) const { return joint_position[i]; }
matrix4 getJointPositionTransf(int i) const { return translate_transf(joint_position[i]); }
m_real getRootHeight() const { return root_height; }
```

# Character Animation

- Posture
  - 각 link마다 coordinate frame을 지정
    - Joint의 위치는 해당 local coordinate에서 결정
  - 부모 coordinate frame 대비 각 coordinate frame의 변형 정의



# Character Animation

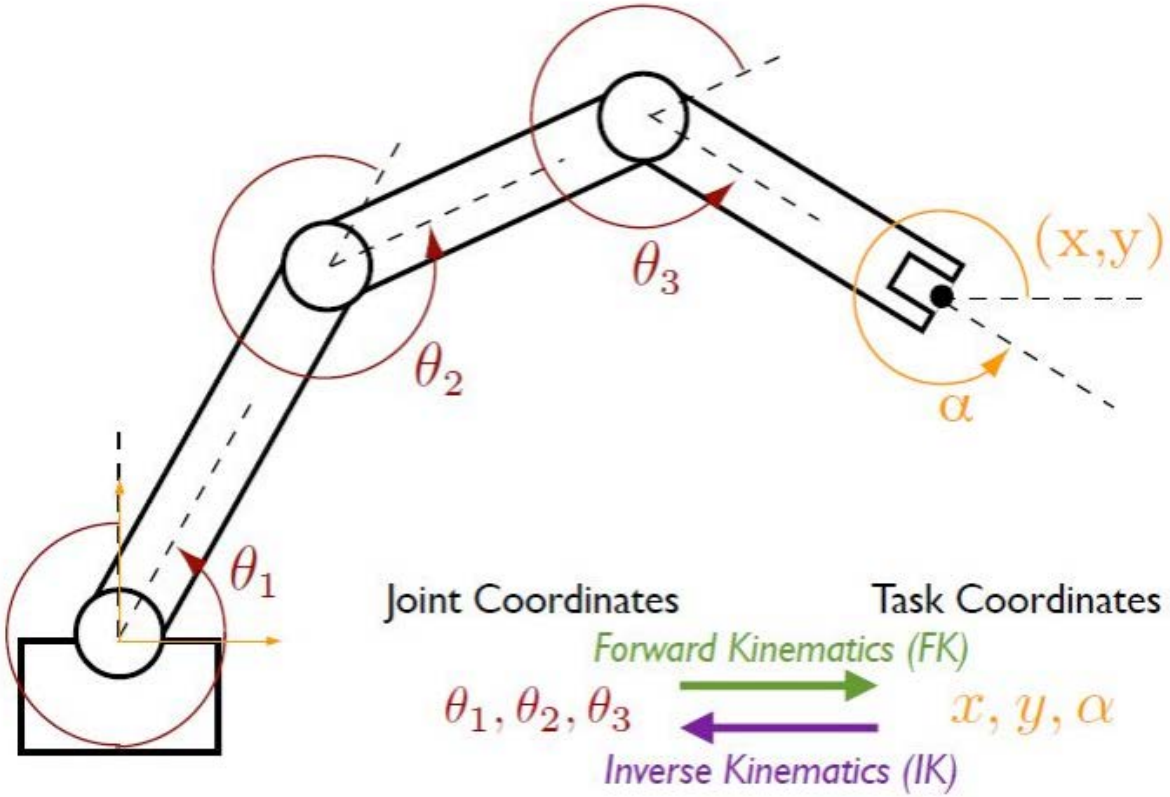
- Kinematics
  - The study of the motion of bodies without reference to mass or external force (a.k.a. geometry of motion)
  - Only use positions, velocities, accelerations of links

- Forward kinematics(FK)
  - Given joint angles (local), calculate (global) position and orientation of the end effector

- Inverse kinematics(IK)
  - Given (global) position and orientation, calculate joint angles (local)

# Character Animation

- FK/IK algorithm



# Character Animation

- Posture class
  - Describing a posture: array of orientations

```
class Posture {
private:
          Skeleton   *pBody;
          m_real     scale;
          vector     trans;
          quarter    rotate[SKELETON_NUM_LINKS];
}
```

  - Member functions

```
Skeleton *setBody(Skeleton *h) { body=h; return h; }
Skeleton *getBody() const { return body; }
vector getTranslation() const;
void setTranslation(vector const&);
void addTranslation(vector const&);
quarter getRotation(int) const;
void setRotation(int, quater const&);
void setRotation(int, vector const&);
void setRotation(int, matrix const&);
void addRotation(int, vector const&);
void setScale(m_real);
Posture& operator=(Posture const&);
matrix4 getBaseTransf(int) const;
matrix4 getGlobalTransf(int) const;
vector getGlobalTranslation(int) const;
quarter getGlobalRotation(int) const;
```

```
matrix4 getGlobalTransf( int i ) const {
     matrix4 t = getTransf(i);

     while( body->getParent(i) != -1 ) {
          t *= translate_transf(getScale()
                    * body->getJointPosition(i));
          i  = body->getParent(i);
          t *= getTransf(i);
     }

     return t;
}
```

# Character Animation

- Skinning technique
  - Mesh를 bone에 붙이는 작업으로 hierarchical 구조의 3D mesh는 joint 간에 끊김 현상이 발생할 수 있는데, 각 bone이 움직일 때 영향을 받는 vertex를 weight 값으로 제어

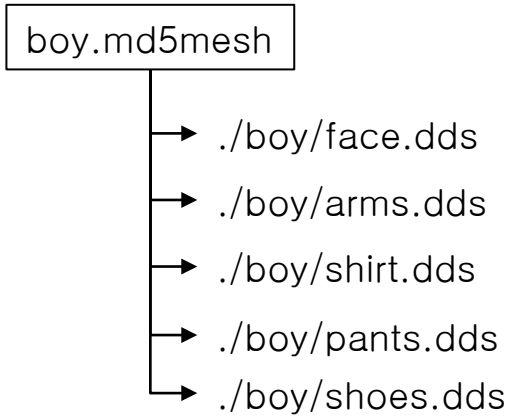
$$V_{world} = V_{local} \times M_1 \times W_1 + V_{local} \times M_2 \times W_2$$

# Tutorials

- Character Model (MD5)
- Character Animation (MD5)
- Character Model (FBX)
- Character Animation (FBX)

# 7-1 Character Model (MD5)

- Loading and displaying MD5 model (*Braynzar Soft)
  - OBJ format does not store animation data
  - md5mesh: containing mesh data (originally developed for Doom 3)
    - Uses a skeletal (joints) structure to define the vertex positions



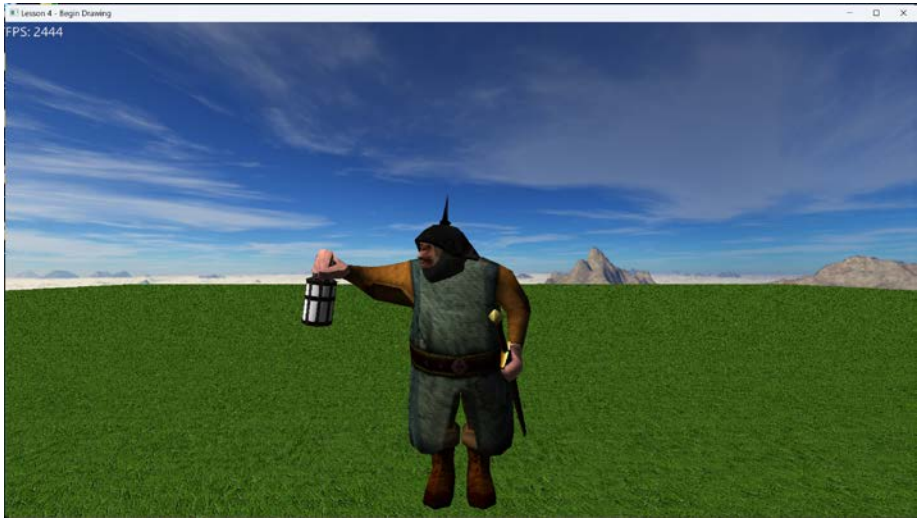

# 7-1 Characgter Model (MD5)

- Defining skeletal structure: md5mesh format
  - Contains a bind pose which is the default position of the model

```
MD5Version 10                          // file version
commandline ""                         // not using

numJoints 27                           // number of joints
numMeshes 5                            // number of subset meshes in this model

joints {                               // joint list
"Bip01"                 -1 ( 0.569962 0.0 -6.39413 ) ( 0.0 0.0 0.707106 ) // joint's "name", ID, (position), (orientation)
"Bip01 Footsteps"       0 ( 0.569962 0.0 -85.008 ) ( 0.0 0.0 0.0 )
...
}

mesh {                                 // start of a subset mesh
// meshes: Head
shader "./boy/face.dds"                // an external material or texture used

numverts 99                            // number of vertices in this subset
vert 0 ( 0.453487 0.77956 ) 0 4        // vertex's index, (uv coordinates), ID of the start weight, number of weights used
...
numtris 139                            // number of triangles in this subset
tri 0 0 2 1                            // ID of the current triangle, IDs of the vertices that make up this triangle
...
numweights 391                         // number of weights in this subset
weight 0 23 0.0681065 ( -61.2806 8.07771 -3.0823 )
// ID of the current weight, ID of the joint that this weight is bound to, weight value, (its position relative to the joints position)
}
```

# 7-2 Character Animation (MD5)

- Loading and displaying MD5 animation
  - md5anim: containing animation data
  - “r” key: shows the next frame

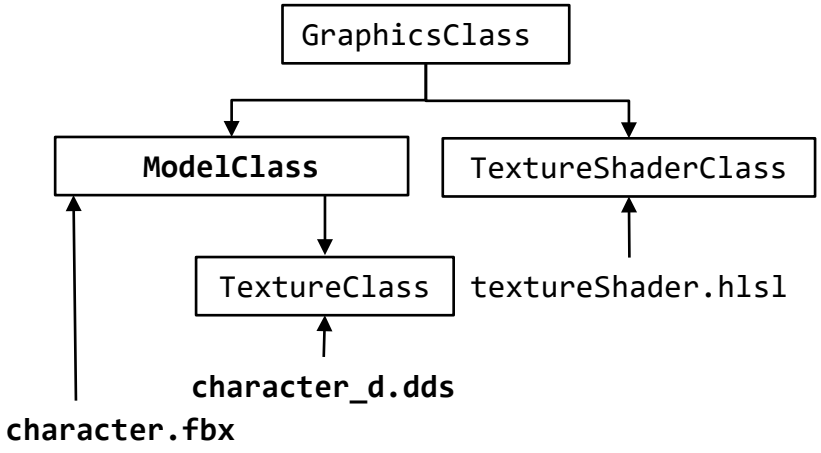
# 7-2 Character Animation (MD5)

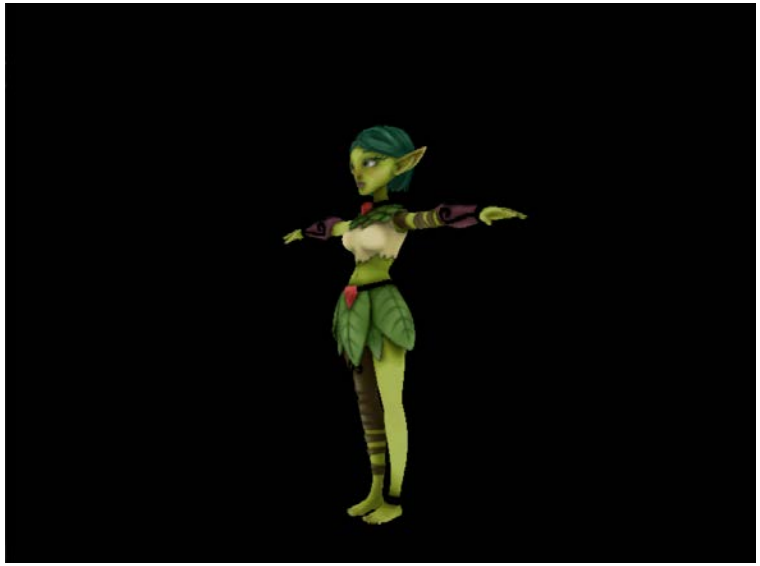
- Defining skeletal structure: md5anim format
  - Contains a bind pose which is the default position of the model

```
MD5Version 10                                   // file version
commandline ""                                  // not using

numFrames 46                                    // number of frames in this animation
numJoints 31                                    // number of joints used in this model
framerate 30                                    // number of frames used for a second
numAnimatedComponents 186                       // number of floating point numbers in each of the frame section

hierarchy {                                     // a hierarchy of skeletal joints which must match with the joint data in .md5mesh
"Bip01"         -1 63 0       // joint's "name", ID of a parent joint (-1: top), parts of joints updating in the frames, start index of updating
"Bip01 Pelvis" 0 63 6
...
}

bounds {                                        // AABB for each of the frames
( -30.4137325286 -23.4689254760 -42.4965248107 ) ( 35.7306137084 6.3094730377 48.7827911376 ) // (min. point), (max point)
...
}

baseframe {                     // default position of each joint in this animation, not necessarily same as the bind pose in "md5mesh"
( 0.5196304321 1.7362534999 4.6482505798 ) ( 0.0000000000 0.0000000000 0.7071063041 ) // (joint's position), (joint's orientation)
...
}

frame 0 {                       // frame data where each float represents one of the positions or orientations in the joints
0.5196304321 1.7362534999 4.6482505798 0.0000000000 0.0000000000 0.7071063041
...}

frame 45 { .... }
```

# 7-3 Character Model (FBX)

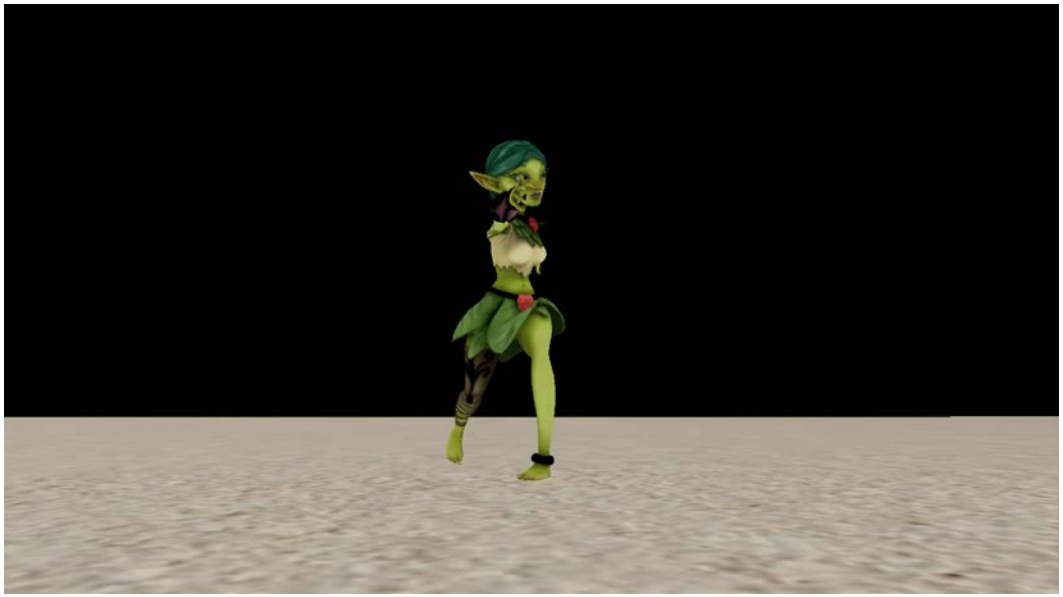
- Loading and displaying FBX model
  - FBX: containing mesh, animation, audio, and image data (originally developed for Motion Builder)
    - Popular support from 3D SWs: Cinema4D, SoftImage, Autodesk, etc.
    - File format: binary or ASCII (FBX SDK)
  - [Project] 속성 → C/C++ → 추가 포함 디렉터리: include
  - .₩lib₩assimp-vc142-mtd.lib: a ASSIMP library file (x86 Debug)
  - .₩include₩assimp: header files for ASSIMP



# 7-4 Character Animation (FBX)

- Loading and controlling FBX animation using key inputs
  - Use ASSIMP lib. to add the animation data
  - WASD: moving the main character with an animation clip
  - Shift: attacking animation



# Q & A