CSc 165
Computer Game Architecture

12 - Animation

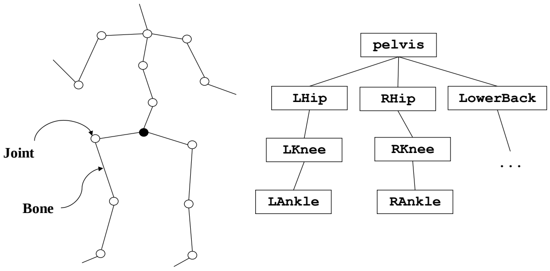
Overview

- Approaches to Animation
- Skeletal Animation
- Animation Transformations
- Keyframe Interpolation
- Keyframe Sequences

Approaches to Animation

- Traditional or "Cel" animation
  - Developed (and still used) for cartooning
- Rigid Hierarchy
- Per-Vertex
  - Morph Targets
- Skeletal

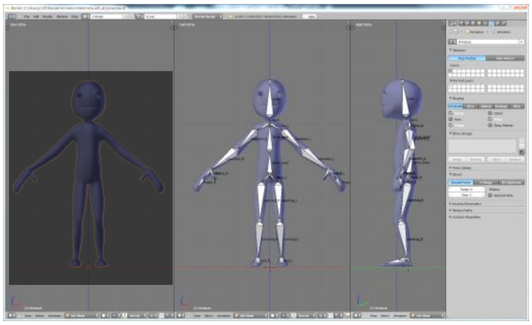
Animation "Skeleton"



Skeletal Animation Process

- Define ***model***
  - vertices/faces/groups
- Define ***skeleton***
  - Bones and Joints
- Associate model vertices with joints
  - "*Rigging*" the model
- Create movement *poses* (***keyframes)***
  - Move joints (vertices follow)
  - Save ***skeleton*** *position/orientation* data as keyframes

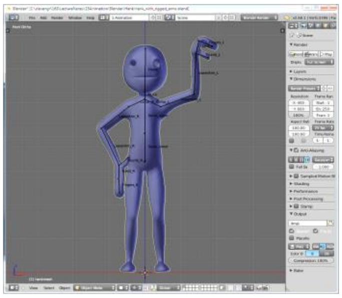
Defining a Skeleton



Model credit: *Essential Blender*, Roland Hess, www.blender.org

Rigging the Mesh

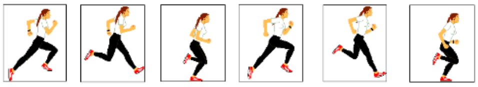
Pose mode



KeyFraming

- Multiple model orientations (views, poses)
  - A single view is called a “frame”
- Each pose represents a “key” view
- Display (render) key views in sequence
  - Or, *interpolate* between keyframes

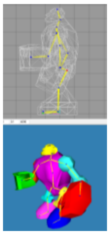
Keyframe Drawing



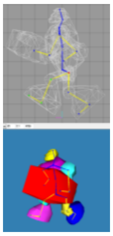
Keyframe: 0 1 2 3 4 5

- **Animated model file stores each keyframe**
- **Application code repeatedly:**
  - **Sets (specifies) “current frame”**
  - **Invokes `model.draw()`**
- **`draw()` renders triangles <u>using current frame</u>**

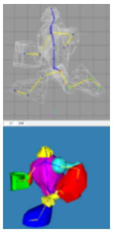
KeyFrames



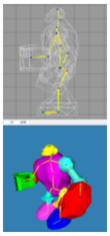
KeyFrame 1 (time = 0)



KeyFrame 2 (time = 15)

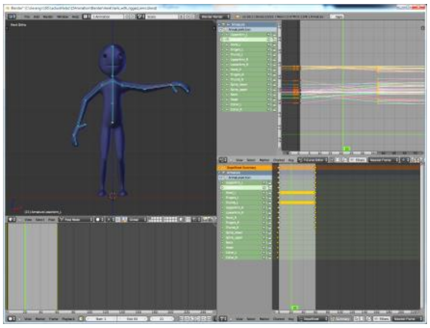


KeyFrame 3 (time = 30)



KeyFrame 4 (time = 45)

Blender Keyframing



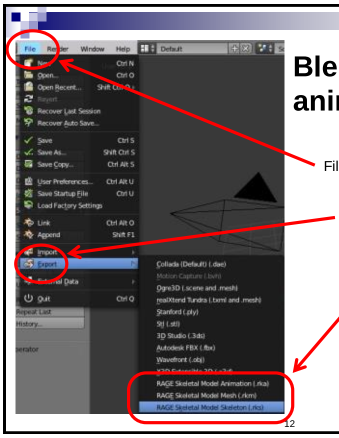
3D View

Graph Editor

Dope Sheet

Timeline

Blender – export animation for RAGE



File

Export

RAGE addons

If a model has multiple animations, each animation is exported separately

Models usually have multiple animations

examples: WoW Models

- *Idle*
- *Walk*
- *Run*
- *Attack*
- *Laugh*
- *Beg*
- *Die*
- …

Crocodile

Chimera

Female Goblin

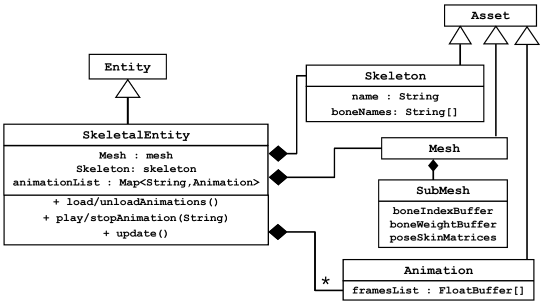
Loading Animated Models in RAGE

```
protected void setupScene(Engine eng, SceneManager sm) throws IOException
{
   // load skeletal entity - in this case it is an avatar
   // parameters are:  entity name, mesh file, skeleton file
   SkeletalEntity manSE =
          sm.createSkeletalEntity("manAv", "man.rkm", "man.rks");

   // loading its texture in the standard way
   Texture tex = sm.getTextureManager().getAssetByPath("man.jpg");
   TextureState tstate = (TextureState) sm.getRenderSystem()
                        .createRenderState(RenderState.Type.TEXTURE);
   tstate.setTexture(tex);
   manSE.setRenderState(tstate);

   // attach the skeletal entity to a scene node
   SceneNode manN = sm.getRootSceneNode().createChildSceneNode("manNode");
   manN.attachObject(manSE);

   // load the model's animations
   manSE.loadAnimation("walkAnimation", "walk.rka");
   manSE.loadAnimation("waveAnimation", "wave.rka");
   . . .
}
```

Playing and Updating Animations in RAGE

```
private void doTheWave()
{   SkeletalEntity manSE =
       (SkeletalEntity) engine.getSceneManager().getEntity("manAv");
    manSE.stopAnimation();
    manSE.playAnimation("waveAnimation", 0.5f, LOOP, 0);
}


protected void update(Engine engine)
{   . . .
    // update the animation
    SkeletalEntity manSE =
       (SkeletalEntity) engine.getSceneManager().getEntity("manAv");
    manSE.update();
}
```

SkeletalEntity Class



Asset

Entity

Skeleton
name : String
boneNames: String[]

SkeletalEntity
Mesh : mesh
Skeleton: skeleton
animationList : Map<String,Animation>
+ load/unloadAnimations()
+ play/stopAnimation(String)
+ update()

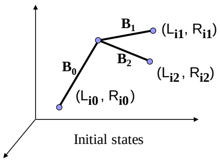
Mesh

SubMesh
boneIndexBuffer
boneWeightBuffer
poseSkinMatrices

*

Animation
framesList : FloatBuffer[]

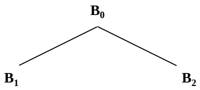
Bone Hierarchy



Initial states

BoneName
Parent Bone
Initial rotation
Rest location

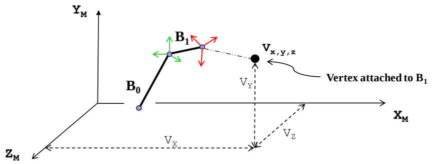
relative to parent joint; i.e., initial transform from parent space to this joint's space



Vertex Transformations

Vertices must be "relocated" (transformed according to their attached bones' transforms) before being drawn

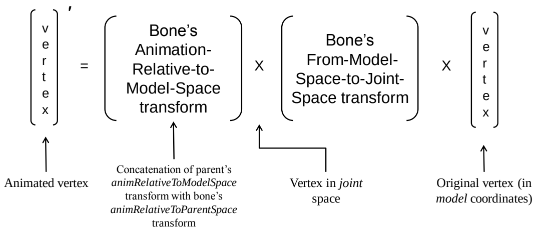
- e.g., multiply each vertex by its joint's "*initial transformations"*
- Vertices are in "global" (model) coordinates
- Bone transforms are in "local" (bone) space, and relative to parent bone



Bone (Model) Animation

- For each bone:
  - Select nearest keyframe based on current time
  - Compute the bone's animation transform from the associated bone's keyframe transform values, applying parents' transforms recursively up to the root bone
- Gather the transformed bones to send to vertex shader
- In the vertex shader – for each vertex:
  - Apply the assigned bone's "animation transform" to vertex
  - If vertex is attached to more than one bone, use a weighted sum (assuming weight-painting was used)
  - Output transformed vertex.

Animating a Vertex


vertex '
=
Bone's Animation-Relative-to-Model-Space transform
X
Bone's From-Model-Space-to-Joint-Space transform
X
vertex
Animated vertex
Concatenation of parent's *animRelativeToModelSpace* transform with bone's *animRelativeToParentSpace* transform
Vertex in *joint* space
Original vertex (in *model* coordinates)

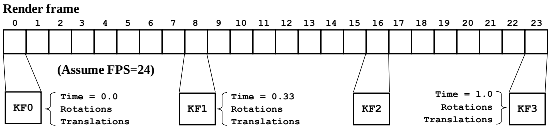
optional -- Keyframe Interpolation

Need *many* keyframes to insure smooth animation

- possible overhead issues

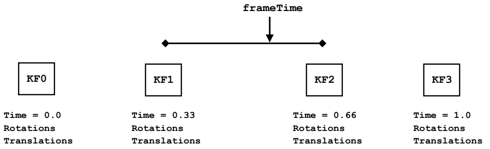
Solution:

- reduce number of keyframes
- *interpolation* for intermediate frames



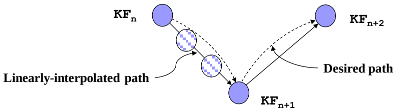
Keyframe Interpolation (cont.)

- Find the "missing keyframe" time
  `frameTime = frameNumber / FPS`
- Select nearest keyframes
- *Interpolate* position and rotation

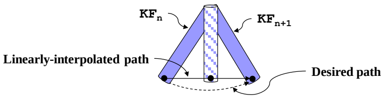


Linear Interpolation Problems

Bouncing ball doesn't "look right":

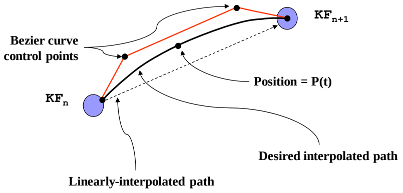


Rotating character arm shortens:



Non-Linear Translation

- Define desired path with (e.g.) cubic curve
- Interpolate position by evaluating curve at **`time=t`**


Bezier curve control points
KF_{n+1}
Position = P(t)
KF_n
Desired interpolated path
Linearly-interpolated path

Keyframe Interpolation vs. Lots of Keyframes

- Keyframe interpolation allows for a smaller model file
- Having the DCC export more keyframes allows the animation to capture advanced DCC animation capabilities
- RAGE export (.rks) files export a keyframe for each frame, to allow taking full advantage of Blender’s animation tools.