CSc 165
Computer Game Architecture

12 – Animated Models

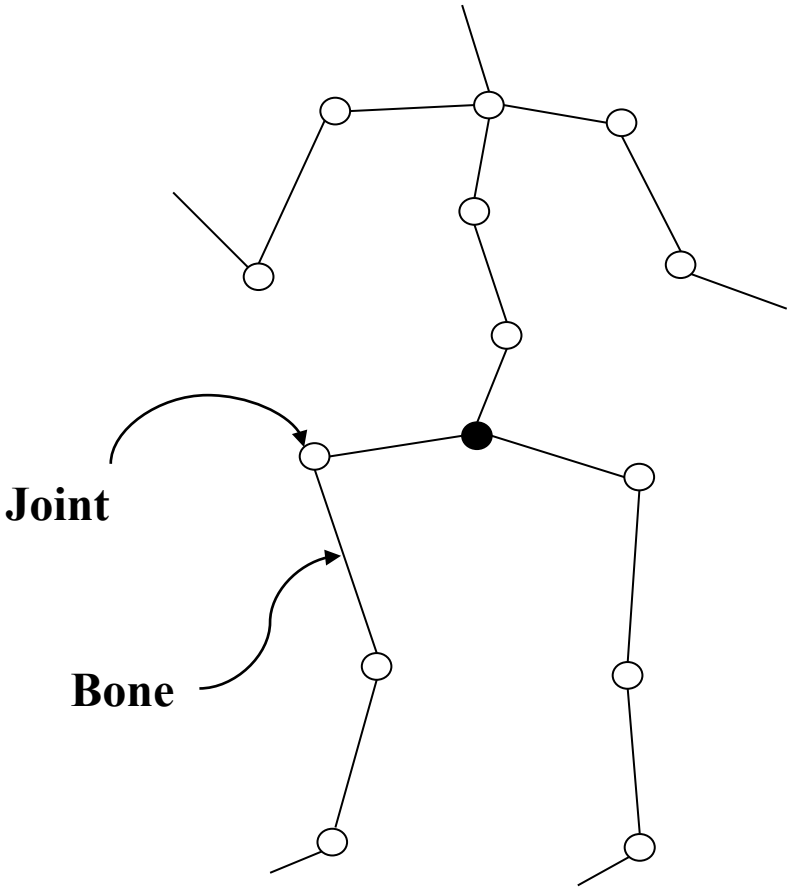
Overview

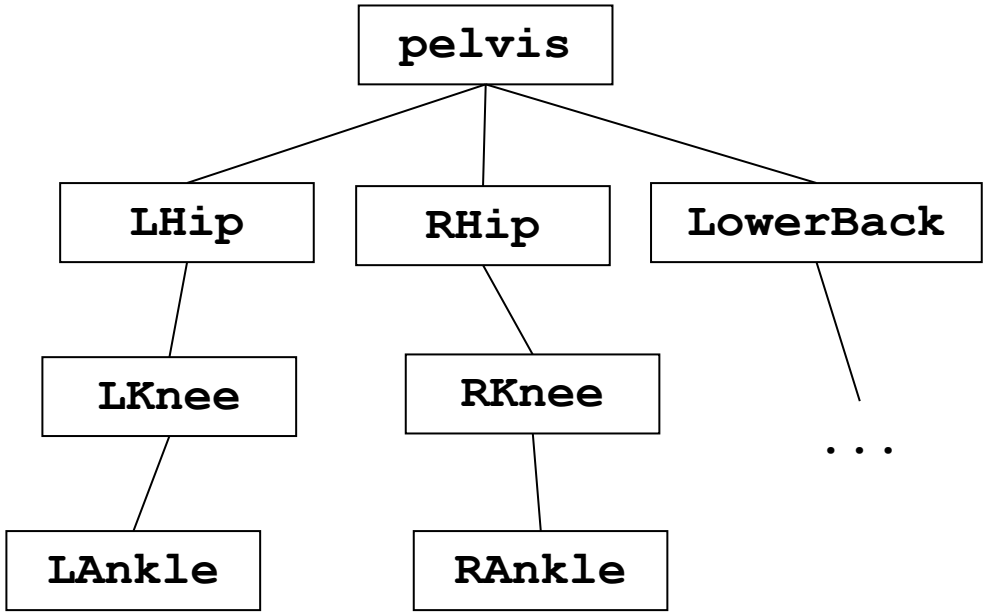
- Approaches to Animation
- Skeletal Animation
- Animation Transformations
- Keyframe Interpolation
- Keyframe Sequences

Approaches to Animation

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

Animation "Skeleton"

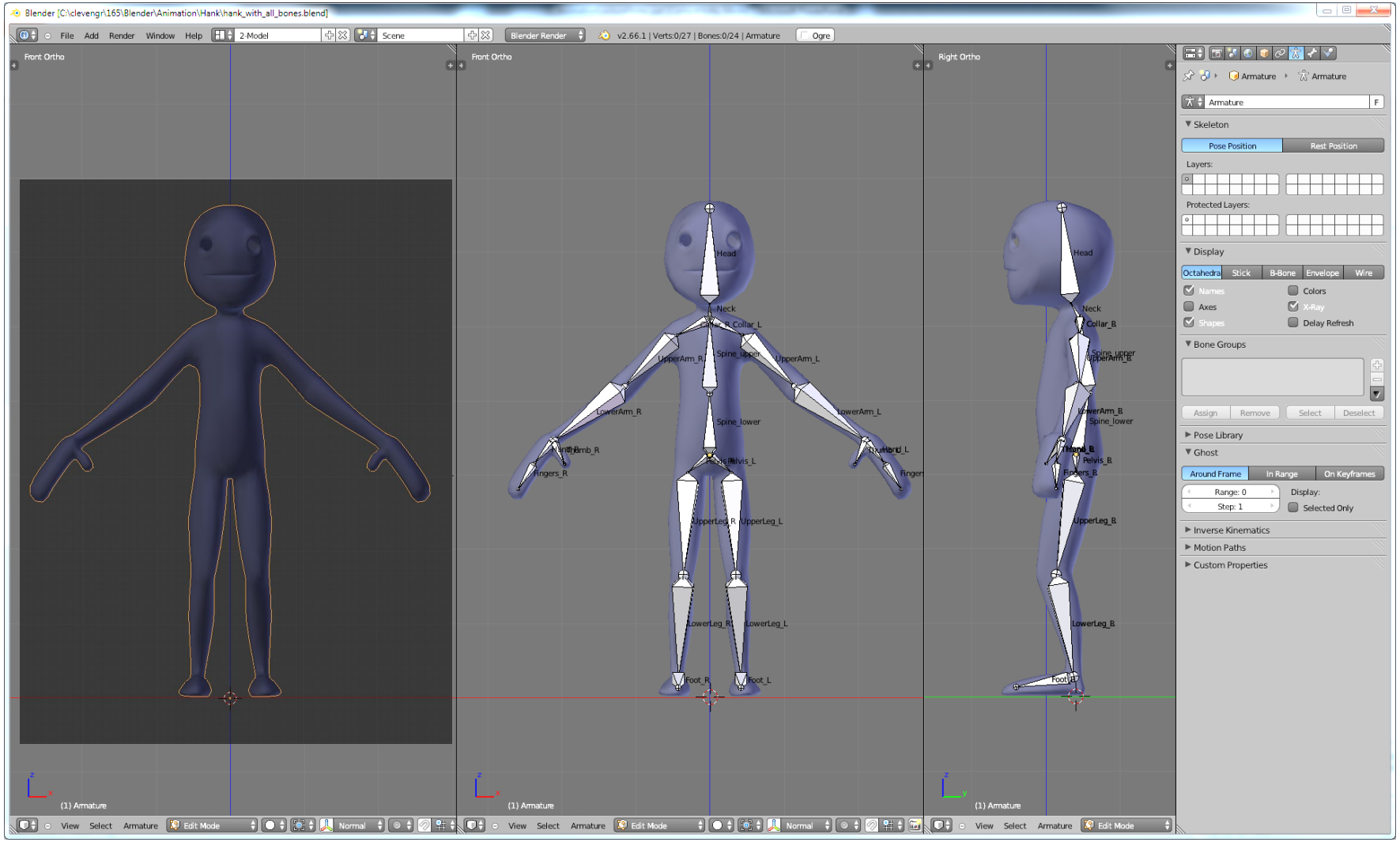




Skeletal Animation Process

- **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**

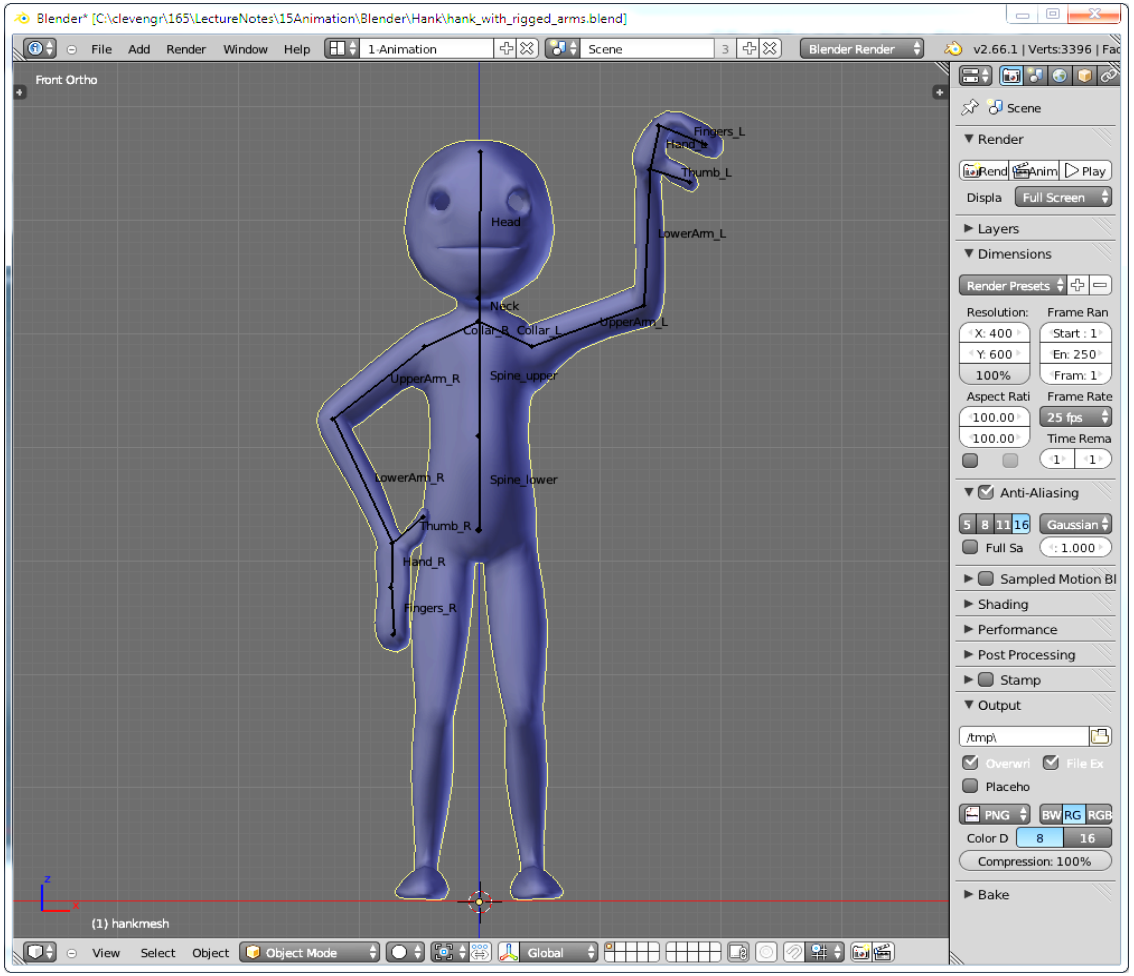
Defining a Skeleton



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

Rigging the Mesh

Pose mode



KeyFraming

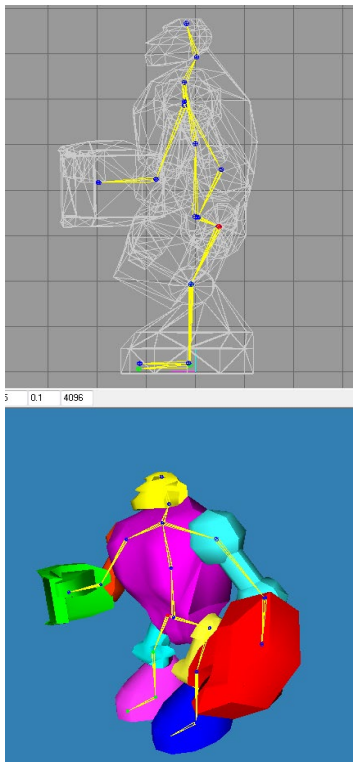
- Multiple model orientations (views, poses)
  - A single view is called a “frame”
- Each pose represents a “key” view
- Display (render) key views in sequence
  - Or, *interpolate* between keyframes

Keyframe Drawing

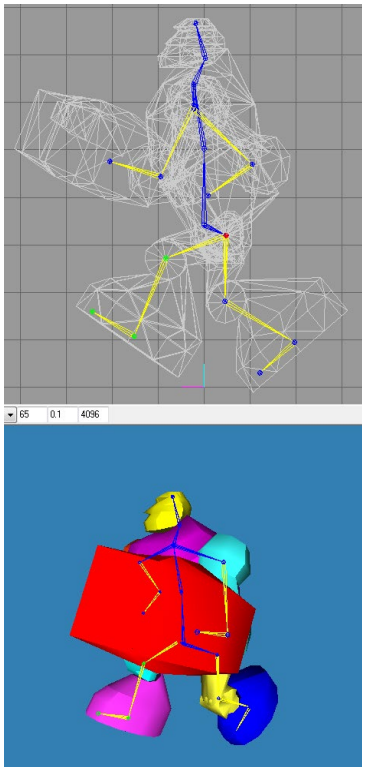
Keyframe: 0 1 2 3 4 5

- **Animated model file stores each keyframe**
- **Application code repeatedly:**
  - **Sets (specifies) "current frame"**
  - **Invokes `model.updateAnimation()`**
- **`updateAnimation()` moves the vertices of the model according to the animation, prior to rendering.**

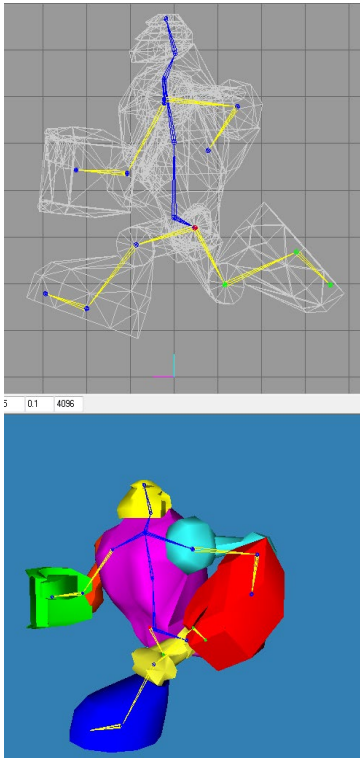
KeyFrames



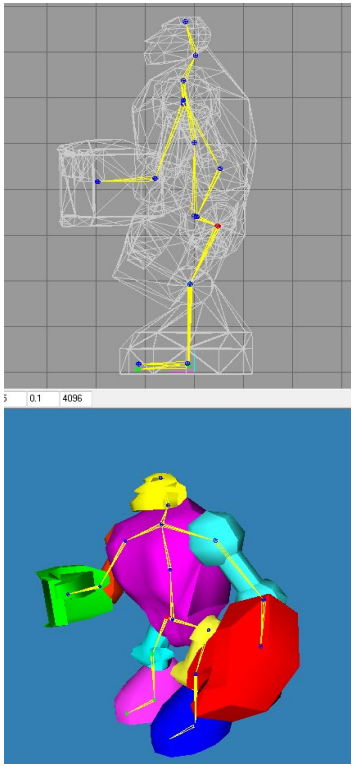
KeyFrame 1
(time = 0)



KeyFrame 2
(time = 15)



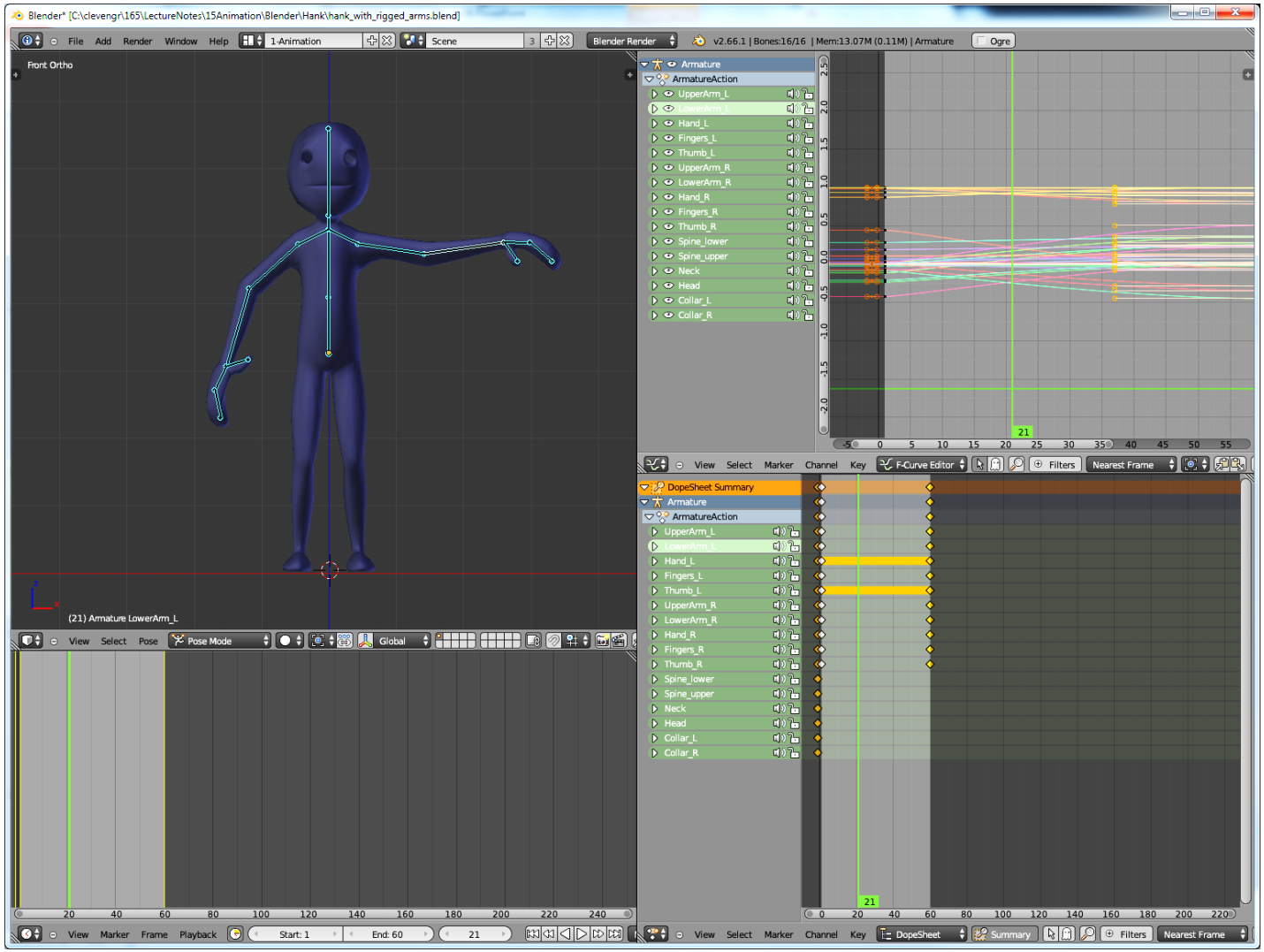
KeyFrame 3
(time = 30)



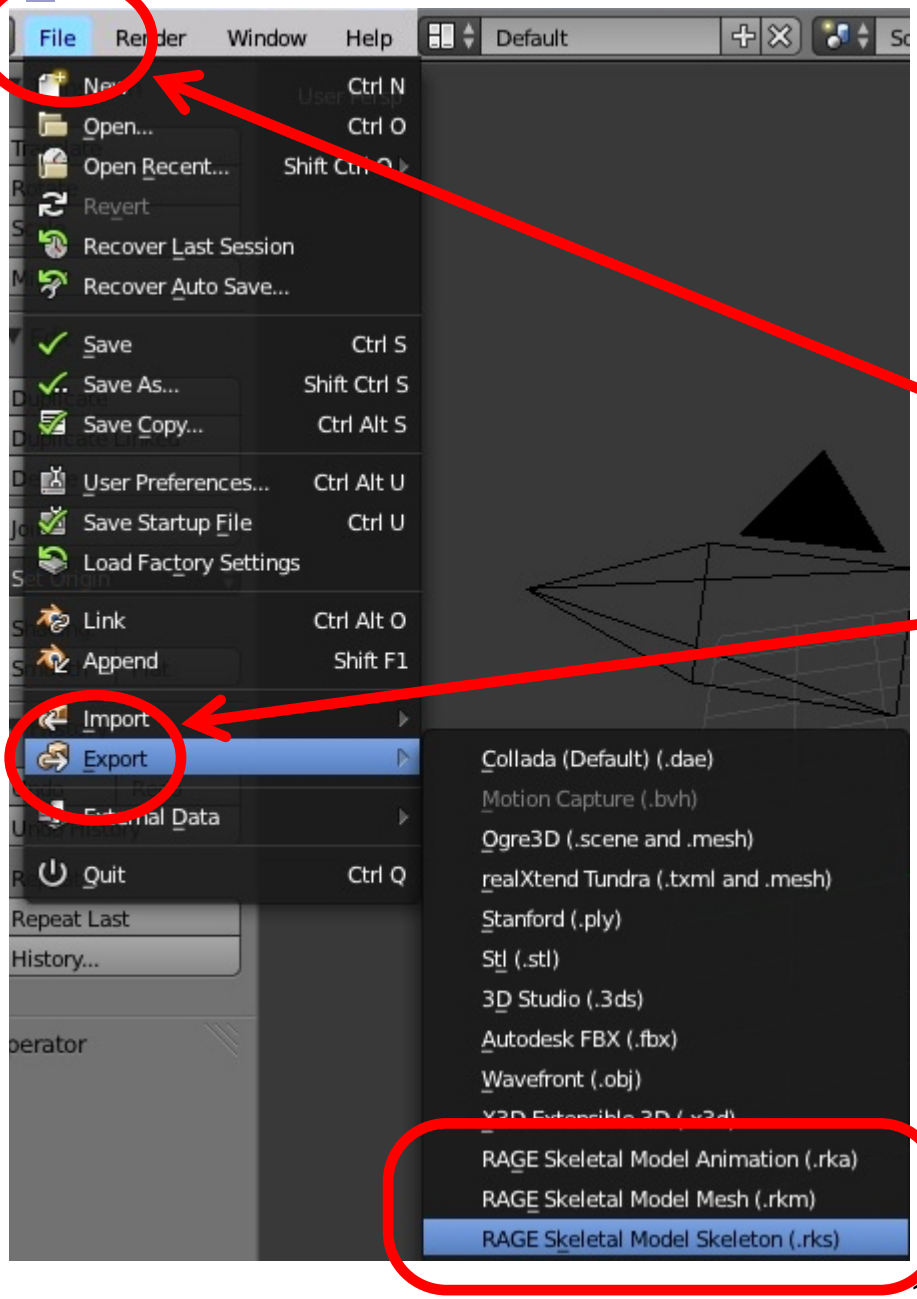
KeyFrame 4
(time = 45)

Blender Keyframing

3D View



Blender – export animation for TAGE



File

Export

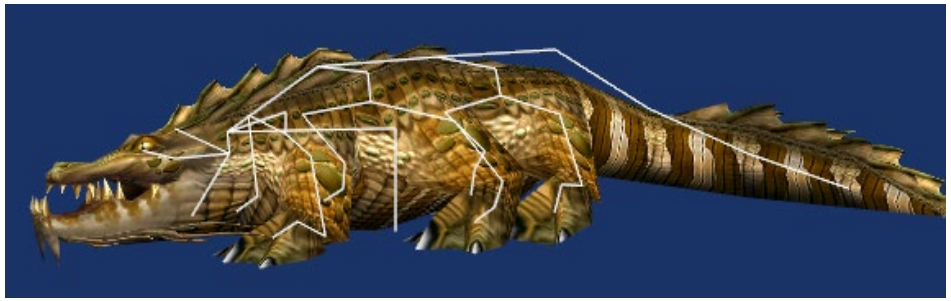
RAGE addons

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

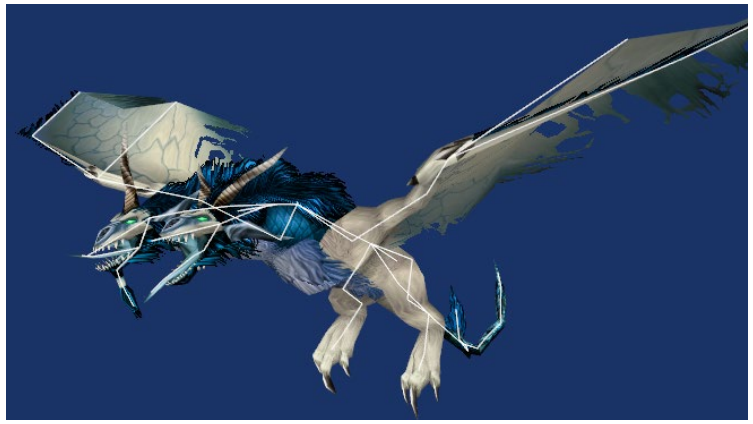
Models usually have multiple animations

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

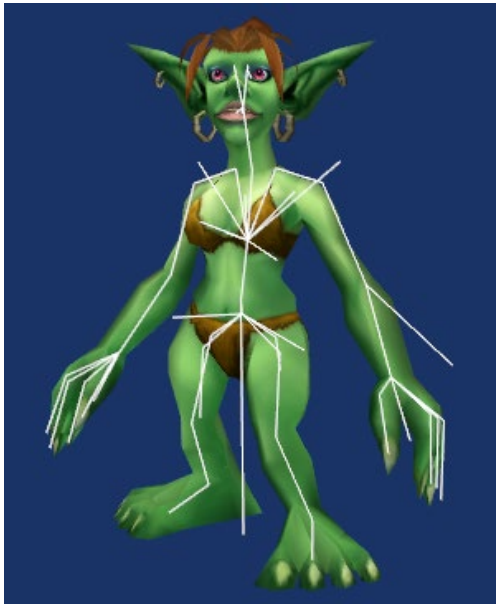
examples: WoW Models



Crocodile



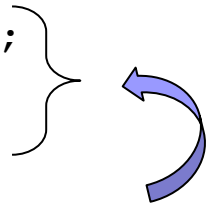
Chimera



Female Goblin

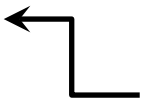
Loading Animated Models in TAGE

```
@Override
public void loadShapes()
{  robotS = new AnimatedShape("robot.rkm", "robot.rks");
   robotS.loadAnimation("WAVE", "robotWave.rka");
   robotS.loadAnimation("WALK", "robotWalk.rka");
   ...
}
```



Assumes these files are all in the assets/animations folder

```
@Override
public void loadTextures()
{  robotT = new TextureImage("robot.jpg");
   ...
}
```



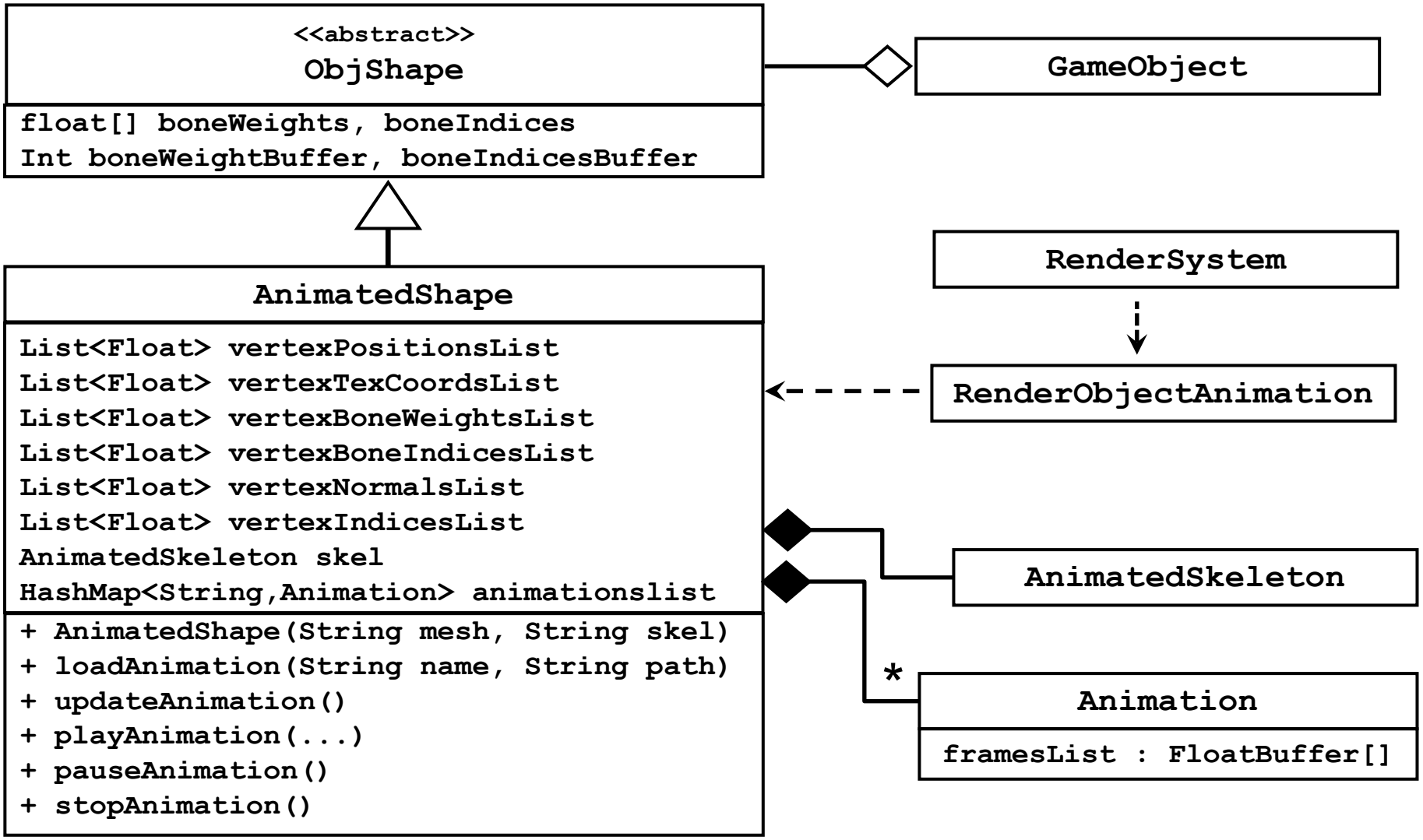
This file is in the assets/textures folder

```
@Override
public void buildObjects()
{  Matrix4f initialTranslation, initialRotation, initialScale;
   robot = new GameObject(GameObject.root(), robotS, robotT);
   ...
}
```

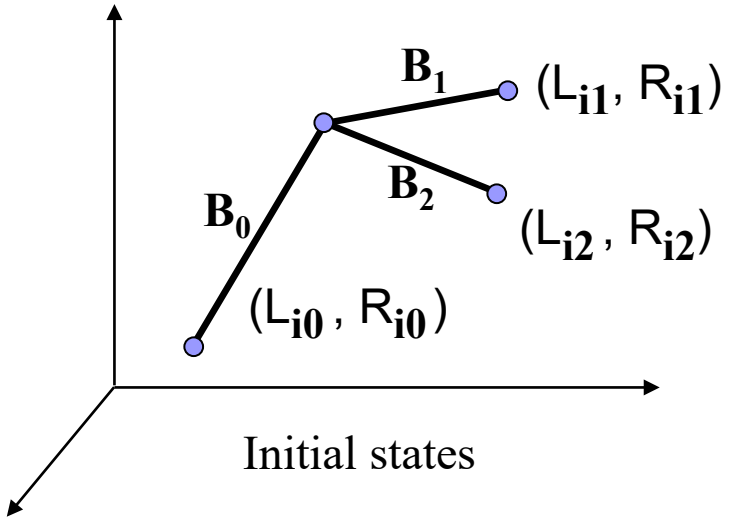
Playing and Updating Animations in TAGE

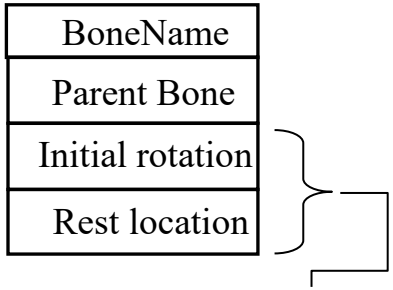
```
public void update()
{  robotS.updateAnimation();
}

public void keyPressed(KeyEvent e)
{  switch (e.getKeyCode())
   {  case KeyEvent.VK_W:
      {  robotS.stopAnimation();
         robotS.playAnimation("WALK", 0.5f, AnimatedShape.EndType.LOOP, 0);
         break;
      }
      case KeyEvent.VK_V:
      {  robotS.stopAnimation();
         robotS.playAnimation("WAVE", 0.5f, AnimatedShape.EndType.LOOP, 0);
         break;
      }
      case KeyEvent.VK_S:
      {  robotS.stopAnimation();
         break;
      }
   }
   super.keyPressed(e);
}
```

AnimatedShape Class (TAGE)

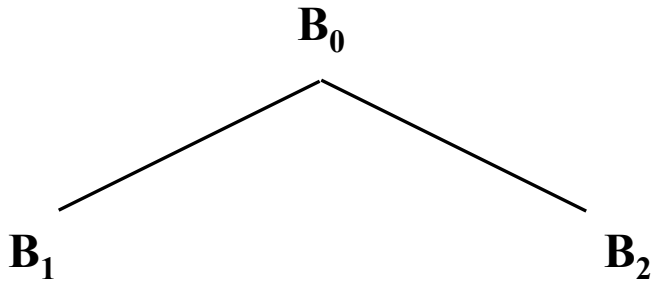


Bone Hierarchy





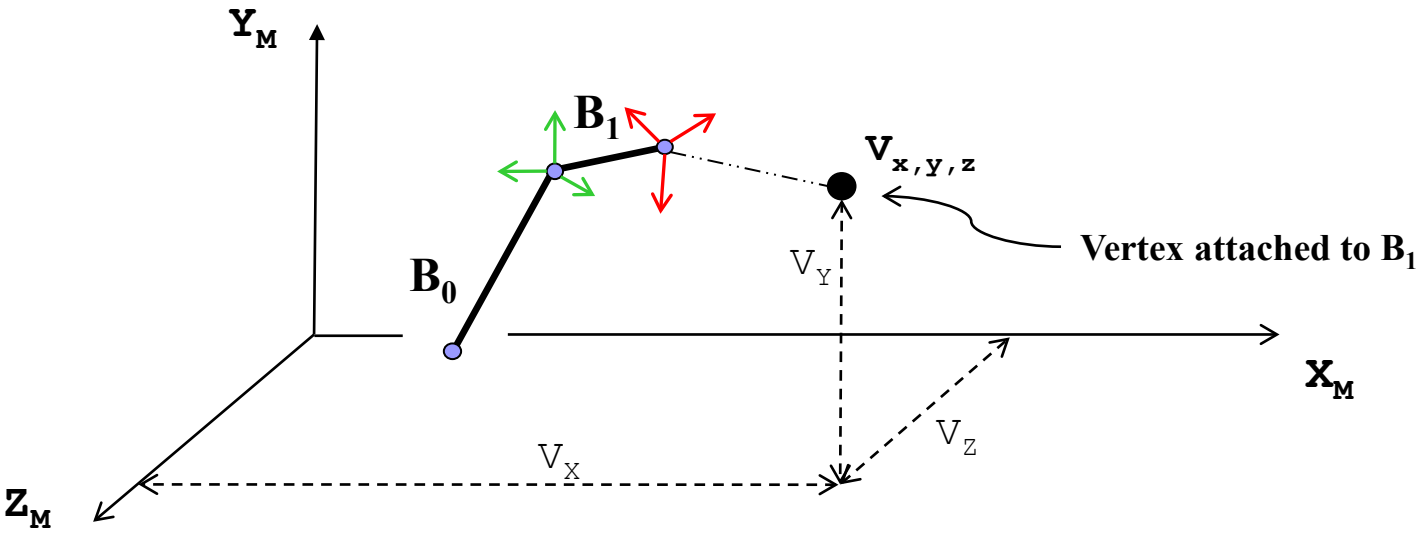
relative to parent joint; i.e., initial transform from parent space to this joint's space



Vertex Transformations

Vertices must be “relocated” (transformed according to their attached bones’ transforms) before being drawn

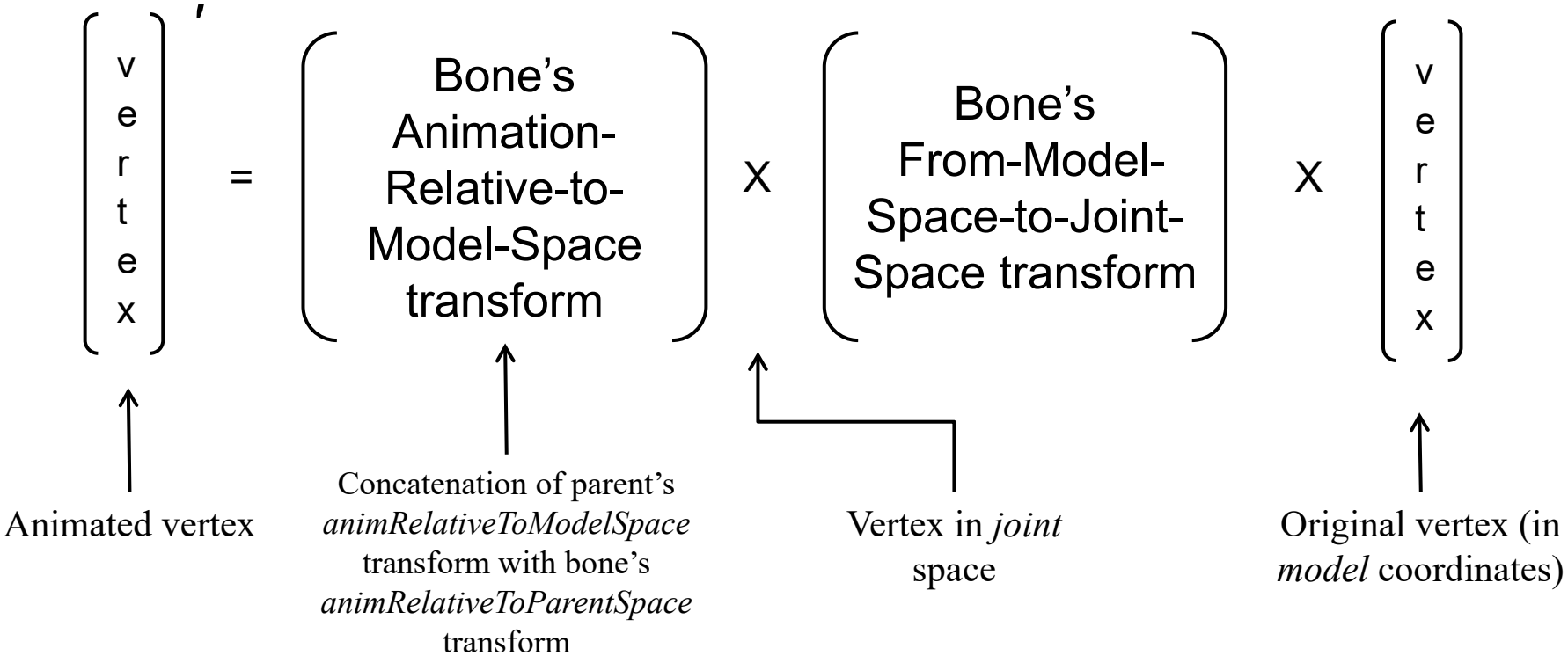
- e.g., multiply each vertex by its joint’s “*initial transformations*”
- Vertices are in “global” (model) coordinates
- Bone transforms are in “local” (bone) space, and relative to parent bone



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



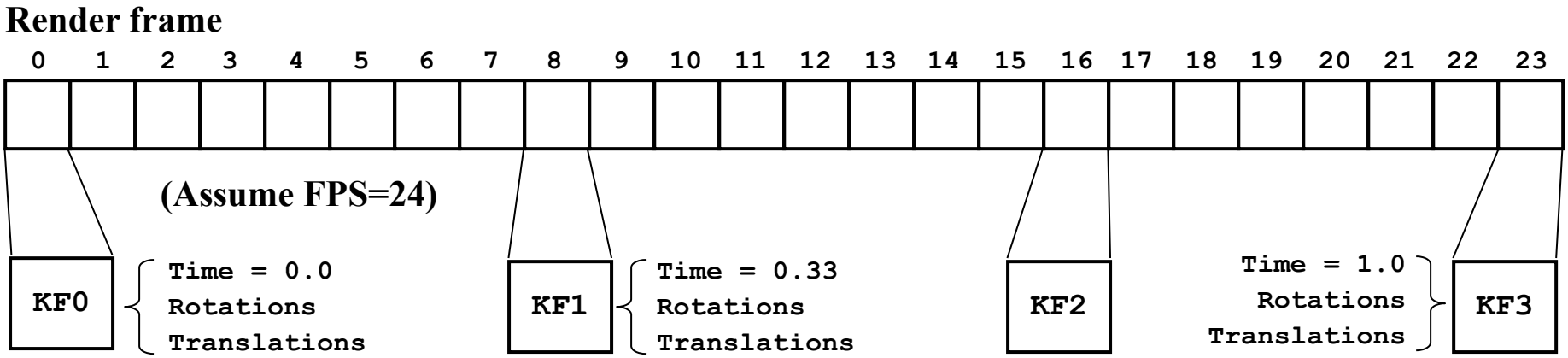
optional -- **Keyframe Interpolation**

Need *many* keyframes to insure smooth animation

- possible overhead issues

Solution:

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

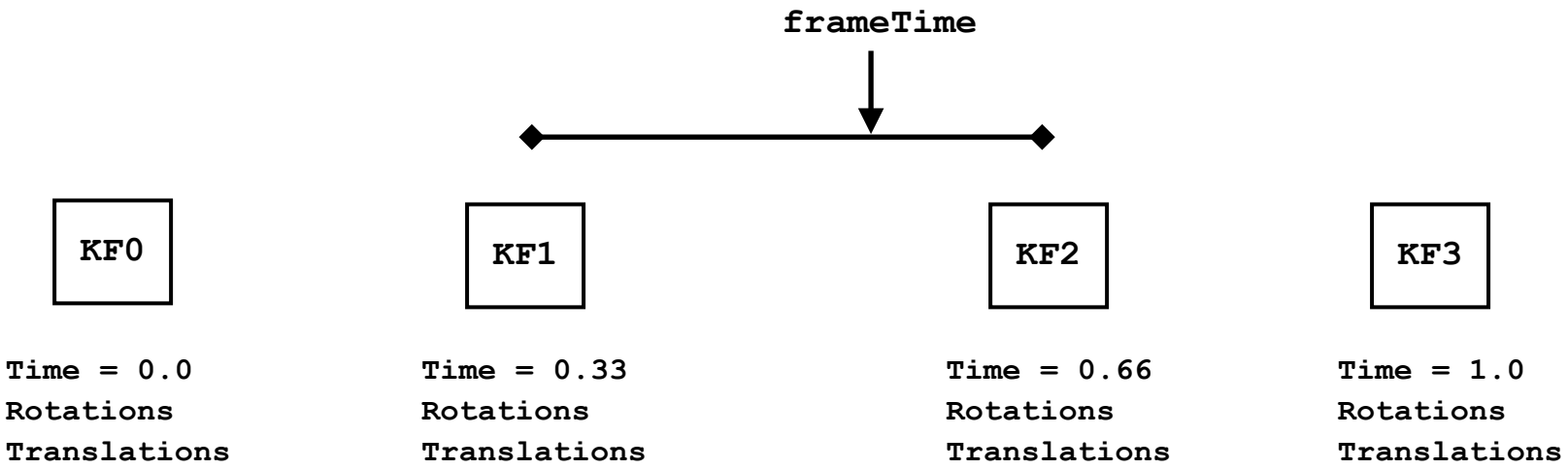


Keyframe Interpolation (cont.)

- Find the “missing keyframe” time

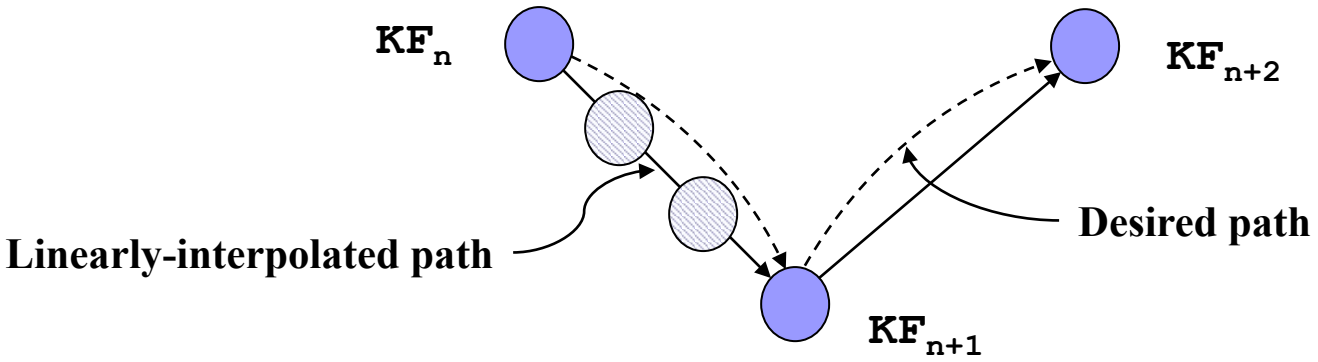
```
frameTime = frameNumber / FPS
```

- Select nearest keyframes
- *Interpolate* position and rotation

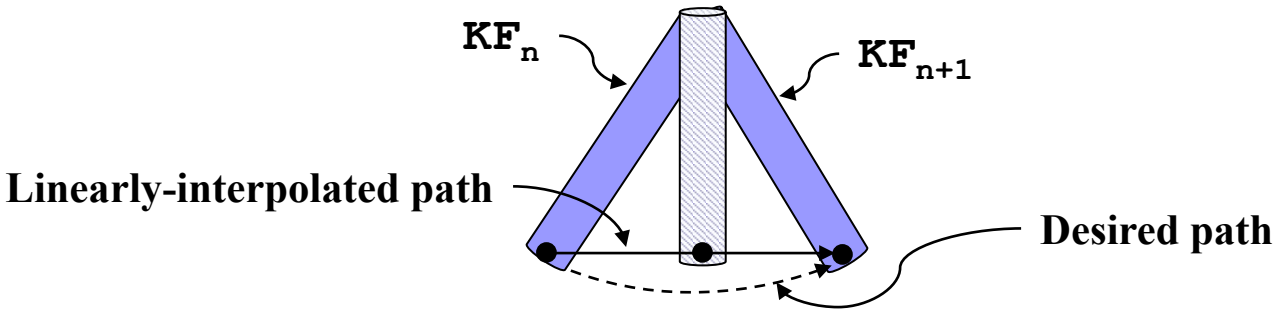


Linear Interpolation Problems

Bouncing ball doesn't "look right":

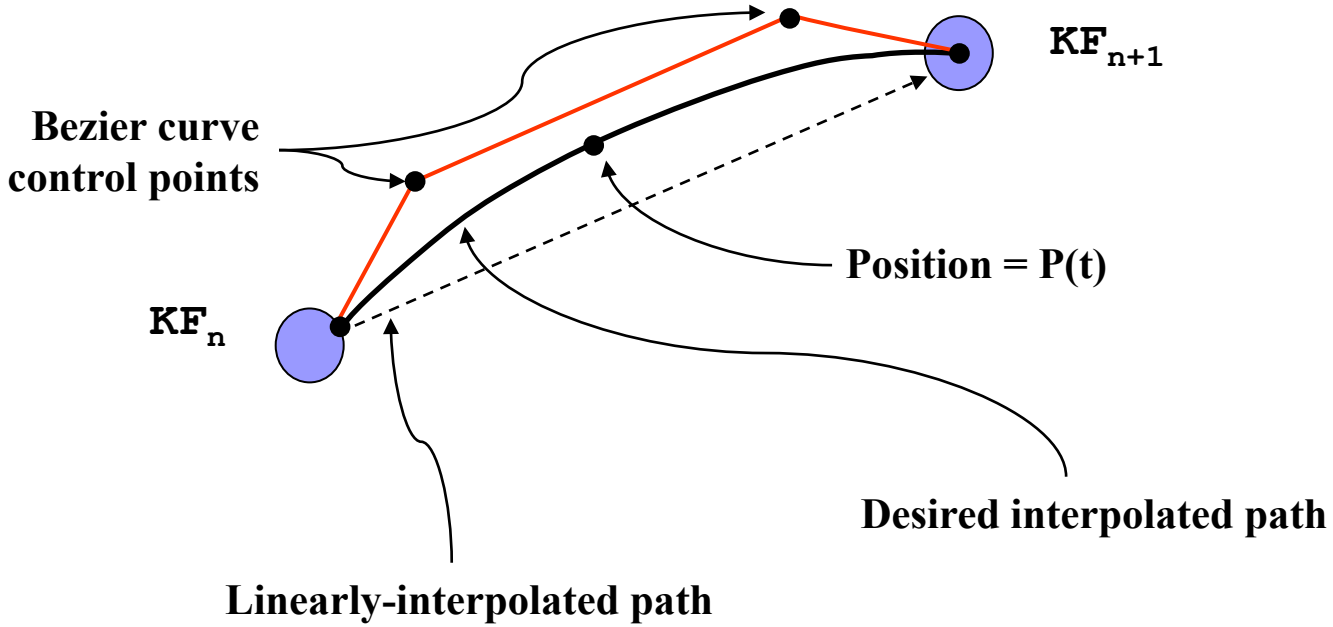


Rotating character arm shortens:



Non-Linear Translation

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



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
- TAGE export (.rks) files export a keyframe for each frame, to allow taking full advantage of Blender's animation tools.