Advanced Animation Programming

GPR-450
Daniel S. Buckstein

Animation Blending: Blend Operations
Weeks 8 – 9

License

Animation Blending

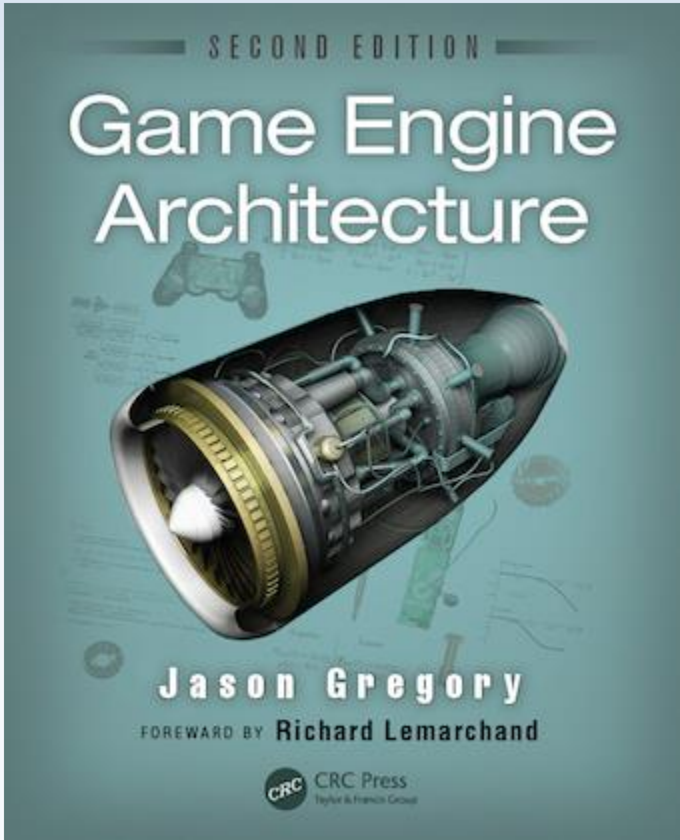
- Recap: poses and pose-to-pose
- Basic pose blending operations
    - How to do skeletal pose blending

Animation Blending

- Some of our systems are derived from here:

- Jason Gregory
- Naughty Dog
- www.gameenginebook.com

- This book is a gold mine.



Recap: Poses & Pose-to-Pose

- ***Joint/node pose data structure***:
- Note: all described ***locally*** for each joint!
- Key Pose:
  – Raw Euler angles (X, Y, Z) or Quaternion
  – Translation
  – Scale
- Can also have *flags* to describe which of these *channels* are active

Recap: Poses & Pose-to-Pose

- Data structures:

```
// ****single pose for a
//  single joint/node
struct HierarchyNodePose {
    vec4 quat_or_euler;
    vec3 translate;
    vec3 scale;
};
```

```
// resource: set of all poses for a character
//  (lists delta poses per-node, per-key and
//  base pose per-node)
struct HierarchyPoseSet {
    const Hierarchy *hierarchy;
    HierarchyNodePose **keyPoseList; // deltas
    HierarchyNodePose *basePose;     // base
    unsigned int keyPoseCount;
}; // omitted: inverse base pose matrices
```

```
// state: current state of all poses per-node
struct HierarchyState {
    const Hierarchy *hierarchy;
    HierarchyNodePose *localPoseList;
    mat4 *localTransformList; // local T
    mat4 *worldTransformList; // global T
}; // omitted: skinning matrices
```

Recap: Poses & Pose-to-Pose

- In addition to *key poses*, skeletons usually have a "*base pose*" or "*initial pose*"
- All pose data can be simplified to represent a *change* from the base pose to current pose:

$$\mathrm{pose}_{n,t} = \mathrm{combine}\left(\mathrm{pose}_{n,\mathrm{base}}\,, \mathrm{poseData}_{n,t}\right)$$

where '*combine*' could be concatenation or simply adding the respective pose components

Recap: Poses & Pose-to-Pose

- You could say that the base pose is a special case, where the pose itself is constant data:

$$\mathrm{pose}_{n,\mathrm{base}} = \mathrm{poseData}_{n,\mathrm{base}}$$

- Fun fact: if the *raw data* doesn't represent a change (i.e. *identity*), you get the base pose:

$$\mathrm{pose}_{n,t} = \mathrm{combine}\left(\mathrm{pose}_{n,\mathrm{base}}\,, \mathrm{identityPose}\right)$$

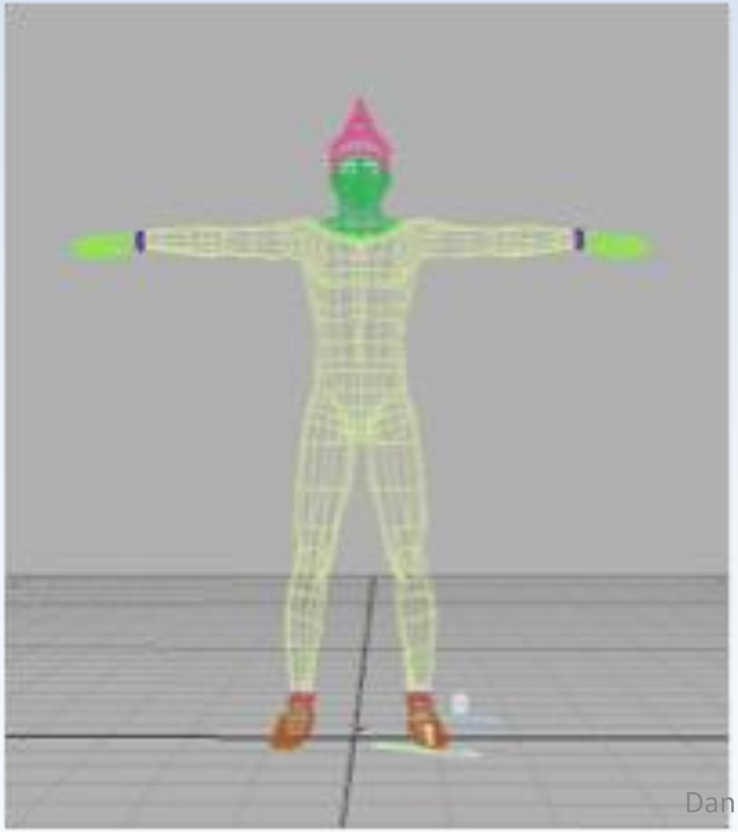
Recap: Poses & Pose-to-Pose

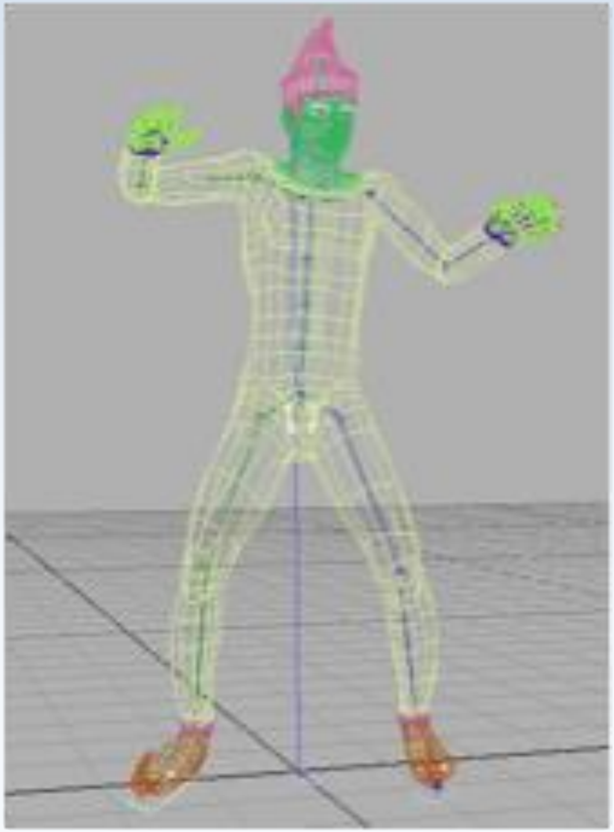
- Base pose to current pose:

Base pose

Apply pose ***change***

Current pose





Recap: Poses & Pose-to-Pose

- To interpolate a rotation + translation combo, the interpolation algorithm must select the appropriate method for each.
- E.g. LERP for position → vector ***LERP***
- E.g. LERP for rotation → quaternion ***SLERP***

Recap: Poses & Pose-to-Pose

- Interpolation and conversion to matrix:

$$T_t = \begin{bmatrix} R_t & \vec{p}_t \\ 0 & 1 \end{bmatrix}$$

$$R_t = \text{convert}\left(\text{slerp}_{\hat{q}_0,\hat{q}_1}(t)\right)$$

$$\vec{p}_t = \text{lerp}_{\vec{p}_0,\vec{p}_1}(t)$$

- More on this method later… ☺

Animation Blending: Operations

- First and foremost, we must define a ***pose***:
- A collection of ***channels***: values that animate over time (i.e. functions of time)
- *Rotation*: 3-channel Euler angles or 4-channel quaternion
- *Scale*: 3-channel axis scalars
    - (all equal if uniform scale)
- *Translation*: 3-channel offset vector

Animation Blending: Operations

- A ***pose*** is mathematically represented as a *set of channels*:

$$P = \{ x_r, y_r, z_r, w_r, x_s, y_s, z_s, x_t, y_t, z_t \}$$

…simplified: $P = \{ \boldsymbol{r}, \boldsymbol{s}, \boldsymbol{t} \}$

where $\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{t}$ are simply storage vectors for multiple channels

Animation Blending: Operations

- ***Pose blending operations***: at the heart of animation blending lies *two operations* or basic algorithms
- ***LERP***: interpolate between poses
- ***ADD***: concatenate poses
- We have already done the first one (see slide 9) but let's break it down using diagrams! ☺

Animation Blending: Operations

- ***Pose LERP***: linear interpolation for poses
- Given input poses P_0 and P_1 and interpolation parameter α, let the result be called P_L:

$$P_0 = \{\, \boldsymbol{r}_0, \boldsymbol{s}_0, \boldsymbol{t}_0 \,\}$$
$$P_1 = \{\, \boldsymbol{r}_1, \boldsymbol{s}_1, \boldsymbol{t}_1 \,\}$$

$$P_\mathrm{L} = \mathrm{LERP}_{P_0,P_1}(\alpha) = \{\, \boldsymbol{r}_\mathrm{L}, \boldsymbol{s}_\mathrm{L}, \boldsymbol{t}_\mathrm{L} \,\}$$

Animation Blending: Operations

- ***Pose LERP***: linear interpolation for poses
- Ultimately a component-wise function:
- For vectors, we should interpolate using
$$v' = \mathrm{lerp}_{v_0,v_1}(t)$$
- For quaternions, we should interpolate using
$$q' = \mathrm{slerp}_{q_0,q_1}(t)$$

  - Outputs are generalized as v' and q'

Animation Blending: Operations

- ***Pose LERP***: linear interpolation for poses
- Ultimately a component-wise function:

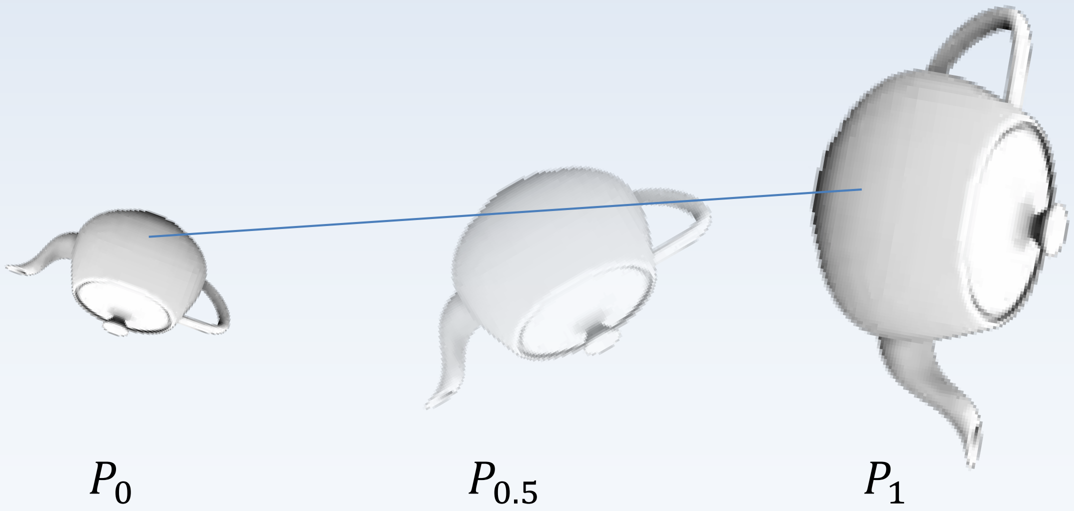
$$\boldsymbol{r}_{\mathrm{L}} = \begin{cases} \mathrm{slerp}_{\boldsymbol{r}_0,\boldsymbol{r}_1}(\alpha) & \text{if using quaternions} \\ \mathrm{lerp}_{\boldsymbol{r}_0,\boldsymbol{r}_1}(\alpha) & \text{if using Euler angles} \end{cases}$$

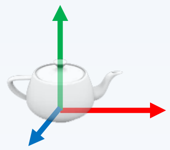
$$\boldsymbol{s}_{\mathrm{L}} = \mathrm{lerp}_{\boldsymbol{s}_0,\boldsymbol{s}_1}(\alpha)$$

$$\boldsymbol{t}_{\mathrm{L}} = \mathrm{lerp}_{\boldsymbol{t}_0,\boldsymbol{t}_1}(\alpha)$$

Animation Blending: Operations

- ***Pose LERP***: linear interpolation for poses
- Ultimately the *'to'* in *pose-to-pose*





Animation Blending: Operations

- ***Pose ADD***: pose concatenation
- Given a *pose* and a *diff pose*, we can combine the two into a new pose
- We have done this before, too: if the base pose is known and we want to find a *key pose*, then the *key pose delta* is our diff
- Again, let's break it down…

Animation Blending: Operations

- ***Pose ADD***: pose concatenation
- Another component-wise function, given input poses P_0 and P_1 we'll call the result P_A

$$P_0 = \{ \boldsymbol{r}_0, \boldsymbol{s}_0, \boldsymbol{t}_0 \}$$
$$P_1 = \{ \boldsymbol{r}_1, \boldsymbol{s}_1, \boldsymbol{t}_1 \}$$

$$P_\mathrm{A} = \mathrm{ADD}_{P_0,P_1}(\quad) = \{ \boldsymbol{r}_\mathrm{A}, \boldsymbol{s}_\mathrm{A}, \boldsymbol{t}_\mathrm{A} \}$$

(no interpolation parameters)

Animation Blending: Operations

- ***Pose ADD***: pose concatenation
- Component-wise function:

$$\boldsymbol{r}_\mathrm{A} = \begin{cases} \boldsymbol{r}_0 \boldsymbol{r}_1 & \text{if using quaternions}^{*} \\ \boldsymbol{r}_0 + \boldsymbol{r}_1 & \text{if using Euler angles} \end{cases}$$

$$\boldsymbol{s}_\mathrm{A} = \mathrm{mulComponents}(\boldsymbol{s}_0, \boldsymbol{s}_1)$$

$$\boldsymbol{t}_\mathrm{A} = \boldsymbol{t}_0 + \boldsymbol{t}_1$$

*Recall that quaternion multiplication is ***non-commutative***, so order matters!

Animation Blending: Operations

- ***Pose ADD***: pose concatenation
- The solution for *scale* is a component-wise function of its own:

$$\mathrm{mulComponents}(\boldsymbol{s}_0, \boldsymbol{s}_1)$$
$$= \{\, x_{\boldsymbol{s}_0} x_{\boldsymbol{s}_1},\; y_{\boldsymbol{s}_0} y_{\boldsymbol{s}_1},\; z_{\boldsymbol{s}_0} z_{\boldsymbol{s}_1} \,\}$$

(just take the product of each matching sub-component… why?)

Animation Blending: Operations

- ***Pose ADD***: pose concatenation
- Usage example: concatenating *base pose* with *key pose delta/diff* to create *key pose*:

$$P_k = \mathrm{ADD}_{P_{\mathrm{base}}, \Delta P_k}(\quad)$$

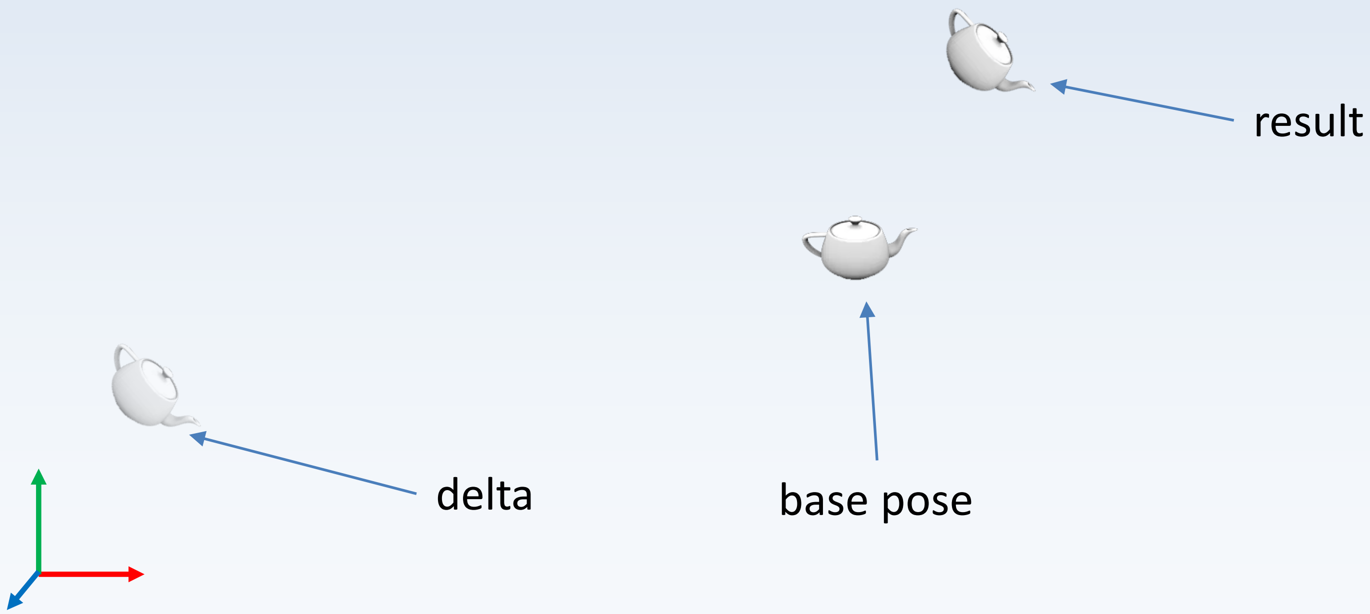
$$\boldsymbol{r}_k = \begin{cases} \boldsymbol{r}_{\mathrm{base}}\,\Delta\boldsymbol{r}_k & \text{if using quaternions} \\ \boldsymbol{r}_{\mathrm{base}} + \Delta\boldsymbol{r}_k & \text{if using Euler angles} \end{cases}$$

$$\boldsymbol{s}_k = \mathrm{mulComponents}(\boldsymbol{s}_{\mathrm{base}}, \Delta\boldsymbol{s}_k)$$

$$\boldsymbol{t}_k = \boldsymbol{t}_{\mathrm{base}} + \Delta\boldsymbol{t}_k$$

Animation Blending: Operations

- ***Pose ADD***: pose concatenation
- Graphical example: base pose is a translated teapot, delta is a rotation and translation…



Animation Blending: Operations

- ***Skeletal pose blending***:
- All of the above applied to a hierarchy (H)

- $H_{\mathrm{L}} = \mathrm{LERP}_{H_0,H_1}(\alpha)$:

For each node *j* in hierarchy *H*,

$$H_{\mathrm{L}\,j} = \mathrm{LERP}_{H_{0\,j},H_{1\,j}}(\alpha)$$

Animation Blending: Operations

- ***Skeletal pose blending***:
- All of the above applied to a hierarchy (H)

- $H_{\mathrm{A}} = \mathrm{ADD}_{H_0,H_1}(\quad)$:

For each pose *j* in hierarchy *H*,

$$H_{\mathrm{A}_j} = \mathrm{ADD}_{H_{0_j},H_{1_j}}(\quad)$$

Animation Blending: Operations

- We can also come up with new operations:

Let $P_I = \{\, \boldsymbol{r}_I, \boldsymbol{s}_I, \boldsymbol{t}_I \}$ be a constant 'identity' pose, where

$\boldsymbol{r}_I = \{\, 0, 0, 0, 1 \,\}$ Satisfies identity quaternion or zero Euler angles

$\boldsymbol{s}_I = \{\, 1, 1, 1 \,\}$ Unit scale

$\boldsymbol{t}_I = \{\, 0, 0, 0 \,\}$ Zero vector

Animation Blending: Operations

- We can also come up with new operations:
- ***Pose scale***: scale from the 'identity' pose

$$\mathrm{SCALE}_P(\alpha) = \mathrm{LERP}_{P_I,P}(\alpha)$$

- We can also use an optimized version of LERP if we know one of the inputs is identity!

Animation Blending: Operations

- We can also come up with new operations:
- ***Pose weighted average***: similar to LERP but with multiple explicit weights:

$$\mathrm{AVG}_{P_0,P_1}(\alpha_0,\alpha_1) = \mathrm{ADD}_{P_0',P_1'}(\quad)$$

$$P_0' = \mathrm{SCALE}_{P_0}(\alpha_0)$$
$$P_1' = \mathrm{SCALE}_{P_1}(\alpha_1)$$

Animation Blending: Operations

- **Summary**:
- Pose blending operation happen on *local* "*delta poses*"
- Delta poses are concatenated/combined (added) with *base pose* to create the *current pose*
- Current pose is converted to transforms, which are finalized in object space using FK

The end.

- Questions? Comments? Concerns?