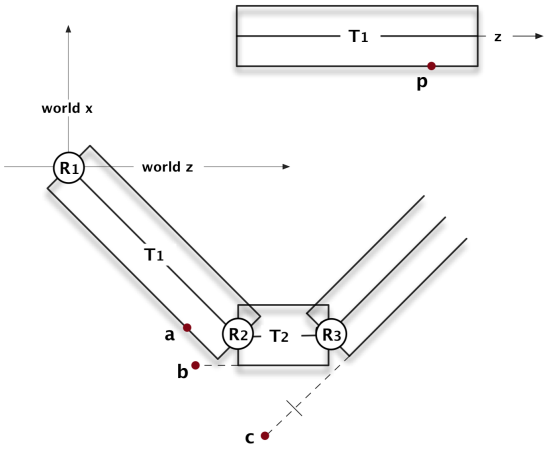
# Skinning math tutorial

j.p.lewis
jun 05




- $W_n^A$ is the animatied transform from the local space of bone $n$ to world space (A is for 'animated'). In the figure,
  - $W_1^A$ is $R_1$, meaning that the world space position of a point $p$ in the local space of bone 1 is $p_w = R_1 p$,
  - $W_2^A$ is $R_1 T_1 R_2$,
  - $W_3^A$ is $R_1 T_1 R_2 T_2 R_3$.
- $W_n^R$ is the corresponding transform from *rest* position of bone $n$ to world space. The rest position typically has most rotations set to zero, so the arms are straight out to the side for example.
  - In the figure, $W_1^R$ is $I$ (identity), $W_2^R$ is $T_1$, and $W_3^R$ is $T_1 T_2$.

The standard skinning algorithm is a weighted blend of a vertex as transformed by various surrounding coordinate systems. In the figure, the vertex $p$ is attached to bone 1, but it is influenced by bones 2,3 also. The world position of this vertex will be

$$p_w = w_1 \times \text{transformed-by(p,bone1)} + w_2 \times \text{transformed-by(p,bone2)} + w_3 \times \text{transformed-by(p,bone3)}$$

with the weights $w_1, w_2, w_3$ chosen by the user (or initialized by the skinning algorithm).

The `transformed-by(p,bone)` takes the point $p$ and moves it to position $b$ for bone2, and position $c$ for bone3. This is done as follows:

- the vertex is first transformed from the local coordinate system of its "parent" bone into the local *rest* coordinate system of the other (transforming) bone. This is done by first taking it into world space, then applying the inverse of the world space rest transform for the other bone – the inverse takes it back into local space. For a point attached to bone $m$, transforming into bone $n$'s local space is

  $$W_n^{R^{-1}} W_m^R$$

  In the figure, the transform for point $p$ (attached to bone1) into bone2's local space is

  $$W_2^{R^{-1}} W_1^R = {T_1}^{-1} I = {T_1}^{-1}$$

and the transform for $p$ into bone3's local space is

$$W_3^{R^{-1}} W_1^R = T_1 {T_2}^{-1} I = {T_2}^{-1} {T_1}^{-1}$$

(remember in general $(AB)^{-1} = B^{-1}A^{-1}$, though in this particular example only transforms are involved, so the order does not matter).

- After the point is found in the local rest space of the other bone, simply transform it into world space using that bone's animated transform. Thus (referring to the figure),

$$\begin{aligned} b &= W_2^A W_2^{R^{-1}} W_1^R p \\ &= (R_1 T_1 R_2)(T_1)^{-1}(I)p \\ &= R_1 T_1 R_2 {T_1}^{-1} p \end{aligned}$$

and

$$\begin{aligned} c &= W_3^A W_3^{R^{-1}} W_1^R p \\ &= (R_1 T_1 R_2 T_2 R_3)(T_1 T_2)^{-1}(I)p \\ &= R_1 T_1 R_2 T_2 R_3 {T_1}^{-1} {T_2}^{-1} p \end{aligned}$$

- The final position of the vertex is then

$$p_w = w_1 p + w_2 b + w_3 c$$

For rendering or other purposes it may be necessary to transform this world space position back into the local coordinate system of the parent bone of the vertex. This is done by multiplying

$$p' = {W_1^A}^{-1}$$

giving $p'$, the "skinned" local position of $p$.

To implement this, functions that compute $W_n^R, W_n^A$ are needed:

- Wrest(bone) returns the 4x4 transform that takes a point from the bone's local space into world space. It computes this by starting at the bone, and walking up its parents to the root of the creature, at each node accumulating the transform by multiplying on the left by the node's local rest position transform. This means that each bone has to know it's rest position as well as it's current animated position.
- Wanim(bone) does the same thing except it accumulates the animated transforms.