skinning 23.1.2, but superseded

- in our robot, we will draw each limb as its own cuboid
- in games, we might start with the character as a complicated triangle mesh “skin”
- we want to animate the skin by moving some underlying “bones”
  - maybe do this smoothly at joints
- demo 9/d3d/pallete

rigging

- start with mesh in a natural “rest pose”.
  - we will cover meshes later.
- each each vertex is described using object coordinates, i.e., \( \vec{p} = \vec{o}c \)
- artist designs a geometric skeleton and fits it to the mesh.
- each vertex is associated to one bone by the artist.
- in our robot example, let us add an “r” subscript to mean the initial rest pose matrices.
- define the cumulative matrix for from the object frame to the bone frame, \( N_r := S_r L_r B \)
- this matrix expresses the frame relationship: \( \vec{b}_r = \vec{o}N_r \).
- consider some vertex, with input object-coordinates \( c \), that has been associated with the lower-arm bone.
- We can write this point as \( \vec{p} = \vec{o}c = \vec{b}_r N_r^{-1}c \).
- so \( N_r^{-1}c \) are the bone coordinates of the point

animate

- manipulate the skeleton, by updating some of its matrices to new settings, say \( S_n \), and \( L_n \) where the subscript “n” means “new”.
- define the “new” cumulative matrix for this bone, \( N_n := S_n L_n B \)
  - which expresses the relation: \( \vec{b}_n = \vec{o}N_n \).
- frame has updated as \( \vec{b}_r \rightarrow \vec{b}_n \).
- to move the point \( \vec{p} \) in a rigid fashion along with this frame, then we need to update it using
  \[
  \vec{b}_r N_r^{-1}c \rightarrow \vec{b}_n N_n^{-1}c \\
  = \vec{o}N_n N_r^{-1}c
  \]
- In this case, the eye coordinates of the transformed point are \( E^{-1}O N_n N_r^{-1}c \)
  - giving us our MVM

soft skinning

- allow the animator to associate a vertex to more than one bone. We then apply the above computation to each vertex, for each of its bones, and then blend the results together.
- we allow the animator to set, for each vertex, an array of weights \( w_i \), summing to one,
  - specify how much the motion of each bone should affect this vertex.
- during animation, we compute the eye coordinates for the vertex as
  \[
  \sum_i w_i E^{-1}O(N_n)_i(N_r)_i^{-1}c
  \] (1)
  where the \((N)_i\) are the cumulative matrices for bone \( i \).
- can be implemented in a vertex shader
  - need to pass an array of MVM matrices.