transformations and deformations

basic principle of 3D animation

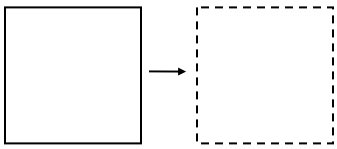
- every object, light, material is defined by a set of parameters
- animation: change parameter values over time
  - animating objects: motion + deformation
  - animating materials: effects
  - animating lights: style + effects
  - animating camera: layout + cuts

setting up objects for animation

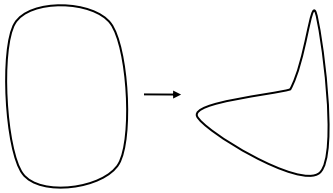
- step 1: specify deformations
  - hierarchies, skinning, blend shapes, etc.
- step 2: provide animation controls
  - pivot points, dummy objects, etc.

deformation types

- object "position" → **rigid body transformation**



- object "shape" → **deformation**

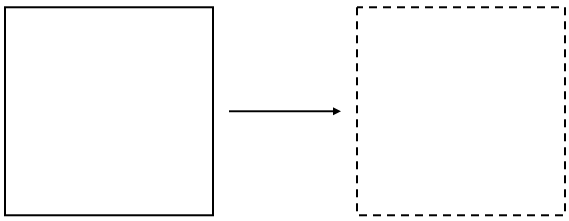


rigid body transformations

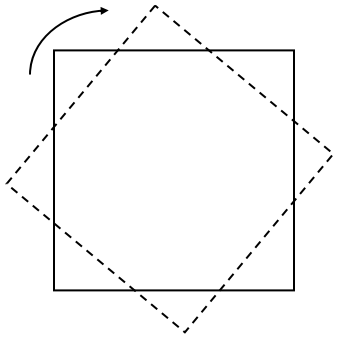
rigid body transformations

- shape of object does not change
  - position of the object does
- formal definition: the relative position of each point of the object does not change
  - i.e. no deformation
- does not provide convincing animation
  - only math abstraction: real objects are non rigid
  - even for rigid bodies, adding deformation helps

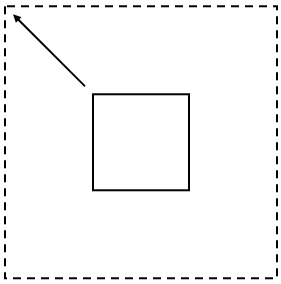
translation



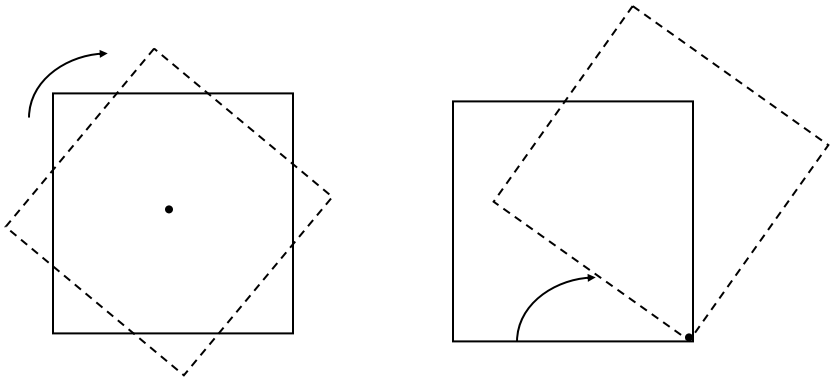
rotation



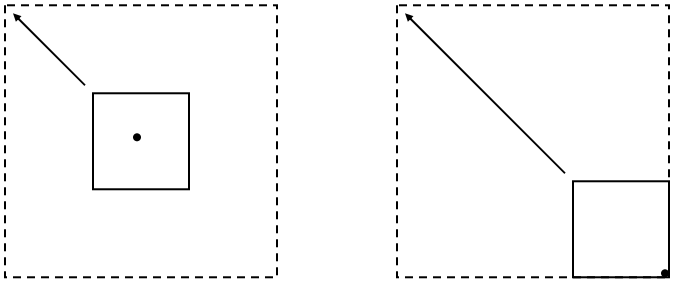
scaling (actually not rigid)



pivot point - for rotation and scaling



pivot point - for rotation and scaling



describing transformations

- functions that transform points to points based on a set of parameters

$$\mathbf{P}' = f(\mathbf{P}, \{\alpha_i\})$$

- all transformations are linear functions
  - can be represented using matrix multiplication
  - unified interface

$$\mathbf{P}' = M \times \mathbf{P}$$

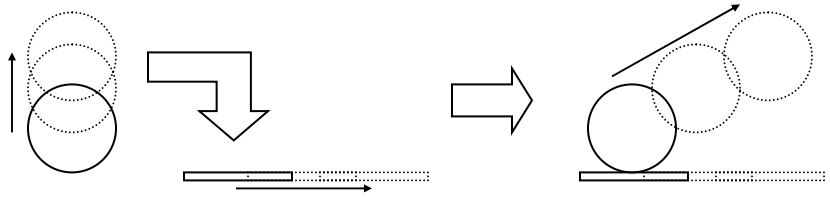
coordinate systems and transformations

- transformations are specified with respect to a coordinate system
  - remember Physics 101?
- important to set up your character properly for animation as well as shading

hierarchical transformations

- specify transformations based on parent objects
  - define hierarchy by repeating it in a chain

$$\mathbf{P}' = f_1(\mathbf{P}) \quad \mathbf{Q}' = f_2(\mathbf{Q}) \longrightarrow \mathbf{P}'' = f_1(f_2(\mathbf{P}))$$



hierarchical transformations

- very useful for animating characters
  - start with shoulder, then arm, then hand, …
- but it is only a helpful system
- often refer to inheriting transformations
  - since you are affected by the parent transform

representing transformations

$$\mathbf{P}' = M \times \mathbf{P}$$

- deformation/animation transformation
  - translation + rotation vector: 6 DOFs
  - not a matrix, since it is hard to control
- controls not same as low level params
  - we need to animate deformations!

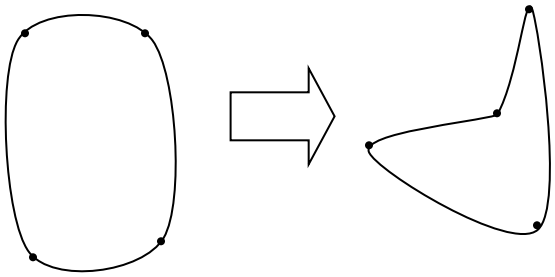
deformations

deformations

- shape of object does change
  - relative position of the object points changes
- provide convincing animation
  - accurate representation of soft objects

most general deformation

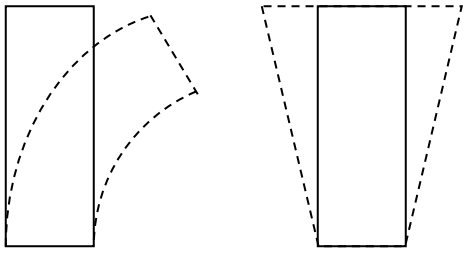
- change independently all control points
  - obtain any possible deformation wanted!
  - *n* x 3 DOFs: basically impossible to control



$$\mathbf{P}' = \mathbf{f}(\mathbf{P})$$

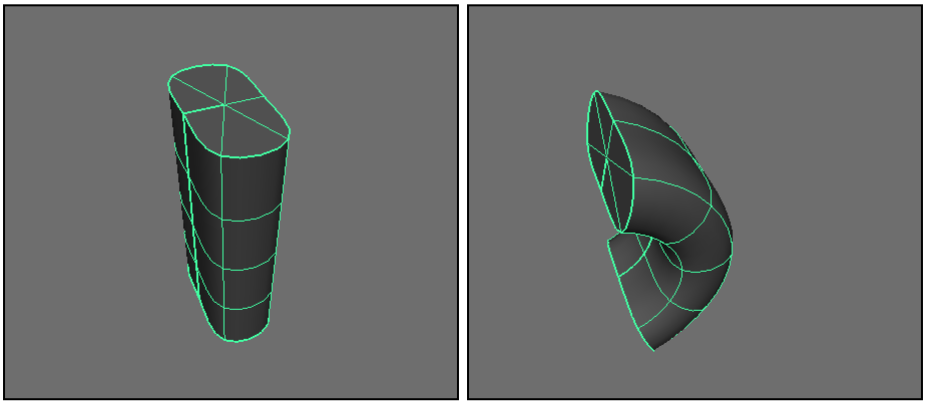
parameterized deformations

- define a deformation function to control points
  - different functions for different deformations
  - *k* params (DOFs): easier to control

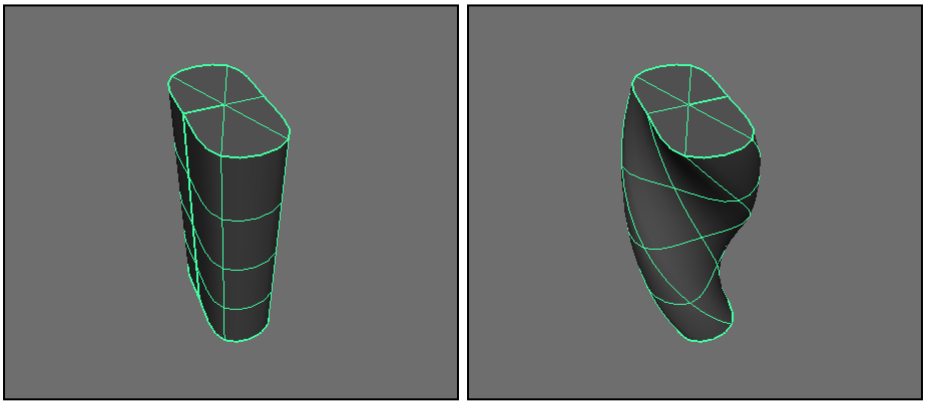


$$\mathbf{P}' = f(\mathbf{P}, \{\alpha_i\})$$

example: blend



example: twist



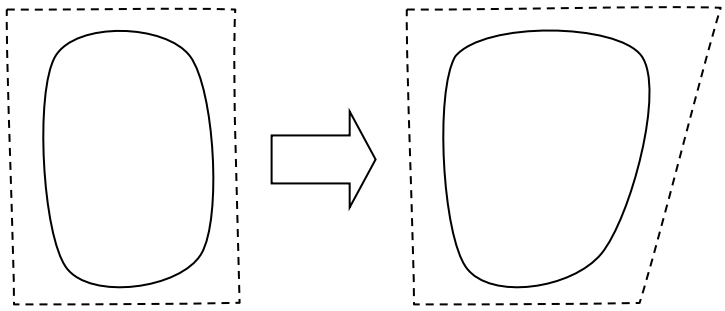
parameterized deformations

- each deformation has its representation
  - no commonality between these representation
  - deformation types are *fundamentally different*
  - software cannot have a unified UI for deformation
    - have to handle "arbitrary" UI
    - often UI fairly poor and ad hoc

lattice deformation

- general smooth deformation
  - size of lattice is smaller than model
  - works on any geometry

$$\mathbf{P}' = f(\mathbf{P}, \{\mathbf{C}_i\})$$

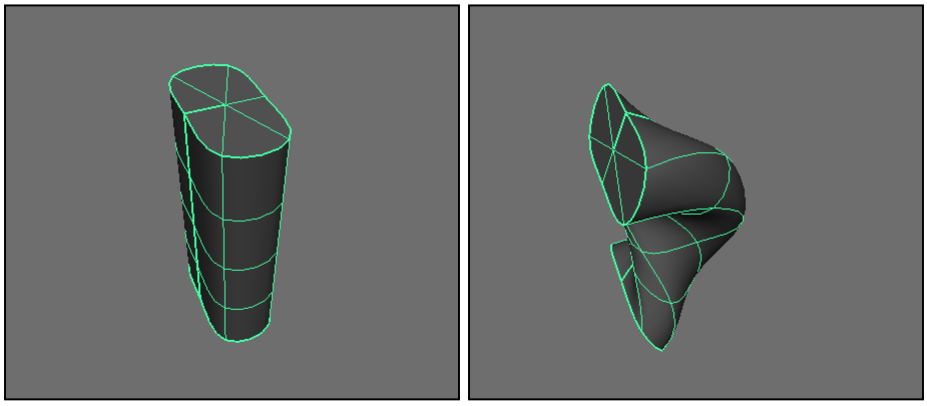


composite deformations

- complex deformations by function composition
  - no unified description, so apply one after the other
  - controlled by union of control params
    - but we do not need to control separately
    - redefine UI: will not cover in class

$$\mathbf{P}' = f_1(f_2(\mathbf{P}))$$

example: bend + twist



deformation and control points

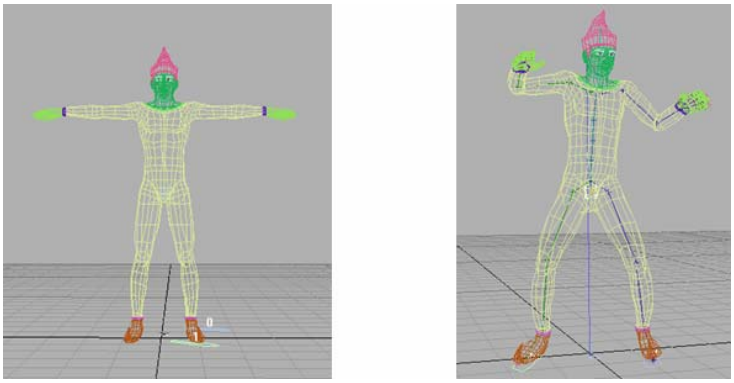
- should we deform control points or the surface?
- in general, deforming control points is wrong
  - cannot prove that the surface is equivalent
- in practice, deforming control points is ok
  - control mesh is tessellated enough
  - many useful transforms are well-behaved

deformations for characters

- combination of lots of simple deformations
  - will need to provide unified controls
- specialized deformations
  - mesh skinning: body deformation
  - blend shapes: face deformation

skinning

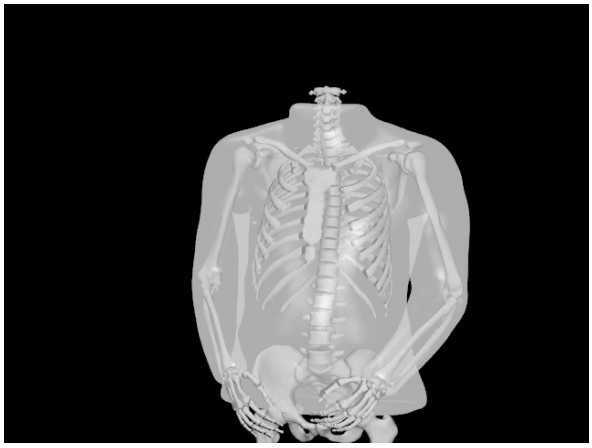
- deform surface around a skeleton



[Domine'/NVIDIA]

skinning

- concepts based on skin/bone interactions



[Fedkiw et al.]

skinning history

- various names: skinning, enveloping, linear blending
- introduced in 3d packages
  - never officially published
  - slightly different in each package
- today often the most used one-stop solution for body deformation

skinning principles

- shape deforms "following" the movement of "bones" close to it

skinning principles

- shape deforms "following" the movement of "bones" close to it
- animation control: set of "bones"
  - coordinate systems with pivot at their based
  - often follow position of real bones in the body
  - inspired by marionettes

skinning principles

- shape deforms "following" the movement of "bones" close to it
- deformation: average of local transformations
  - transform points into bones coordinate system
  - apply bone transformation
  - average the positions relative to each bone
  - transform back to world space

$$\mathbf{P}_i' = \sum_j w_{ij} M_j M_{ref\ j}^{-1} \mathbf{P}_i$$

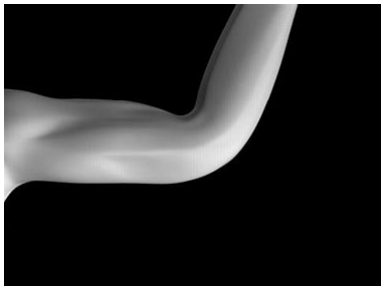
skinning principles

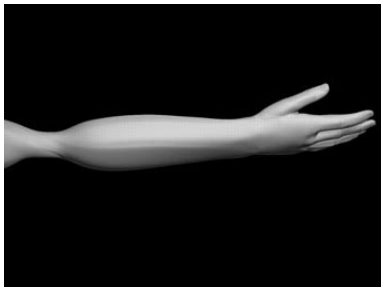
- shape deforms "following" the movement of "bones" close to it
- deformation control: weights/bone positions
  - only close bones should matter
  - for each point define weights for each bone (user)
  - use a weighted average

$$\mathbf{P}_i' = \sum_j w_{ij} M_j M_{ref\ j}^{-1} \mathbf{P}_i$$

skinning issues

- hard to achieve the deformation you want
  - bone positions and weights
- algorithm has fundamental limitations
  - elbow collapse, candy-wrap wrist, no muscle bulging
    - fix by adding other deformers





[Lewis et al., 2000]

skinning subdivs

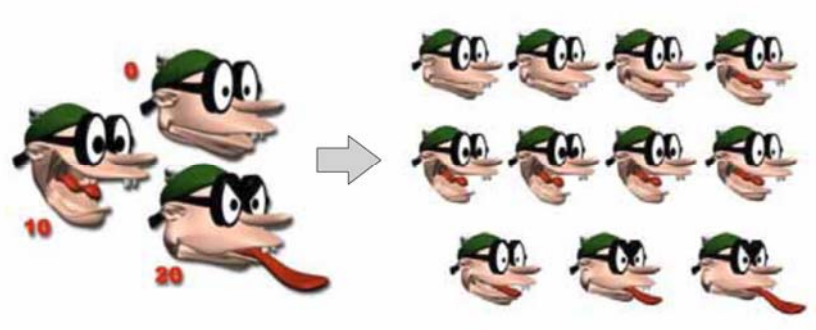
- should only skin control points
  - if not enough points to define weights, subdivide once
  - why? decouple surface quality from animation!

skinning summary

- solution for body deformation
- efficient to compute
  - hardware acceleration available
- good control
  - but hard to set up proper weights
- often used in games as-is
- used in movies as part of more complex setups

blend shapes

- interpolate set of meshes



[3DMax docs/Discreet]

blend shapes principles

- blend between a given set of poses

blend shapes principles

- blend between a given set of poses
- deformation control: sculpted poses
  - weighted average of pose positions
  - same topology for each pose
    - i.e. can only change vertex position

$$\mathbf{P}_i' = \sum_j w_j \mathbf{P}_{ji} \qquad \sum_j w_j = 1$$

blend shapes principles

- blend between a given set of poses
- animation control: weights
  - choose how much each poses matters
  - change weights to obtain different pose

$$\mathbf{P}_i' = \sum_j w_j \mathbf{P}_{ji} \qquad \sum_j w_j = 1$$

blend shapes issues

- cannot create deformations not modeled
  - only smoothly switches between poses
- typically only small interpolation works
  - larger ones deforms weirdly
- for control, might need lots of deformations
  - combine local deformations to make complex ones
    - e.g.: smile + eye expressions

blend shapes summary

- solution for face deformation
- efficient to compute
  - albeit increase memory usage
- great control
  - but only works for small deformation
  - cannot produce “novel” shapes
- often used in games

