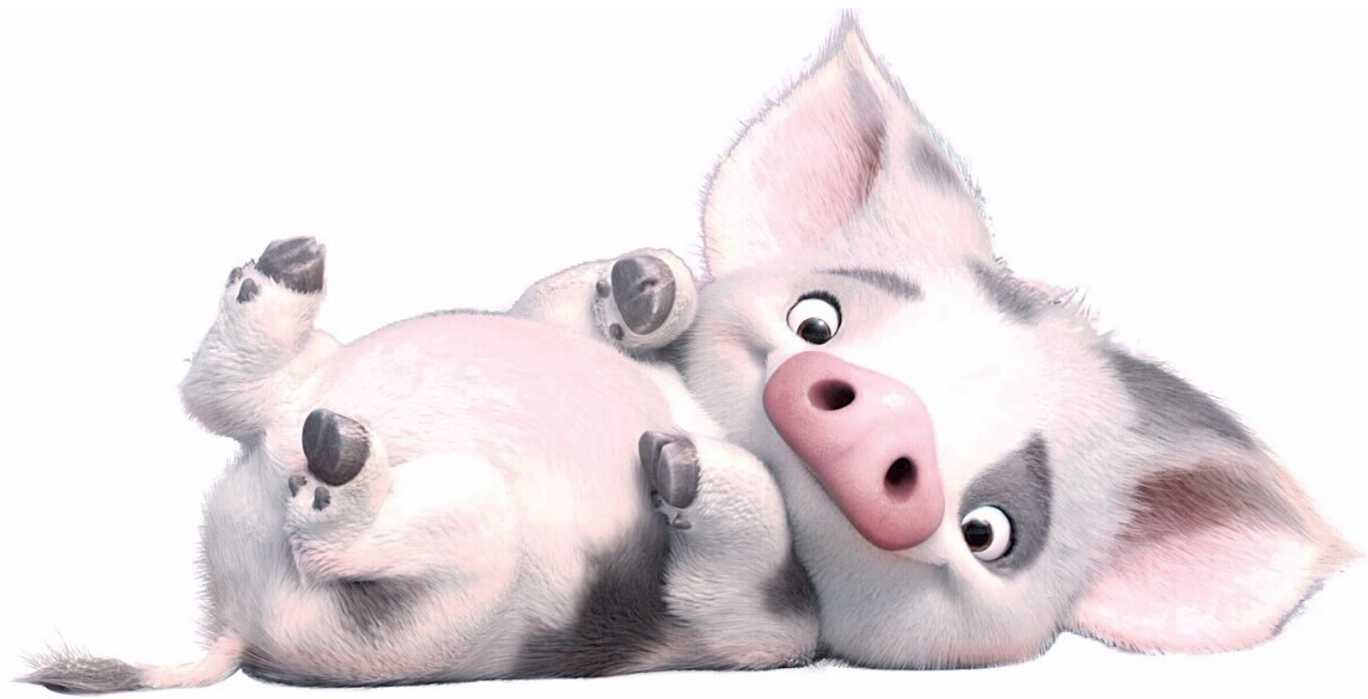
HELP SESSION

ANIMATOR

OUTLINE

- Application interface
- Project requirements
  - Curves: Bezier, B-splines, Catmull-roms
  - Particle system (w/ forces + collisions)
- Artifact tips!

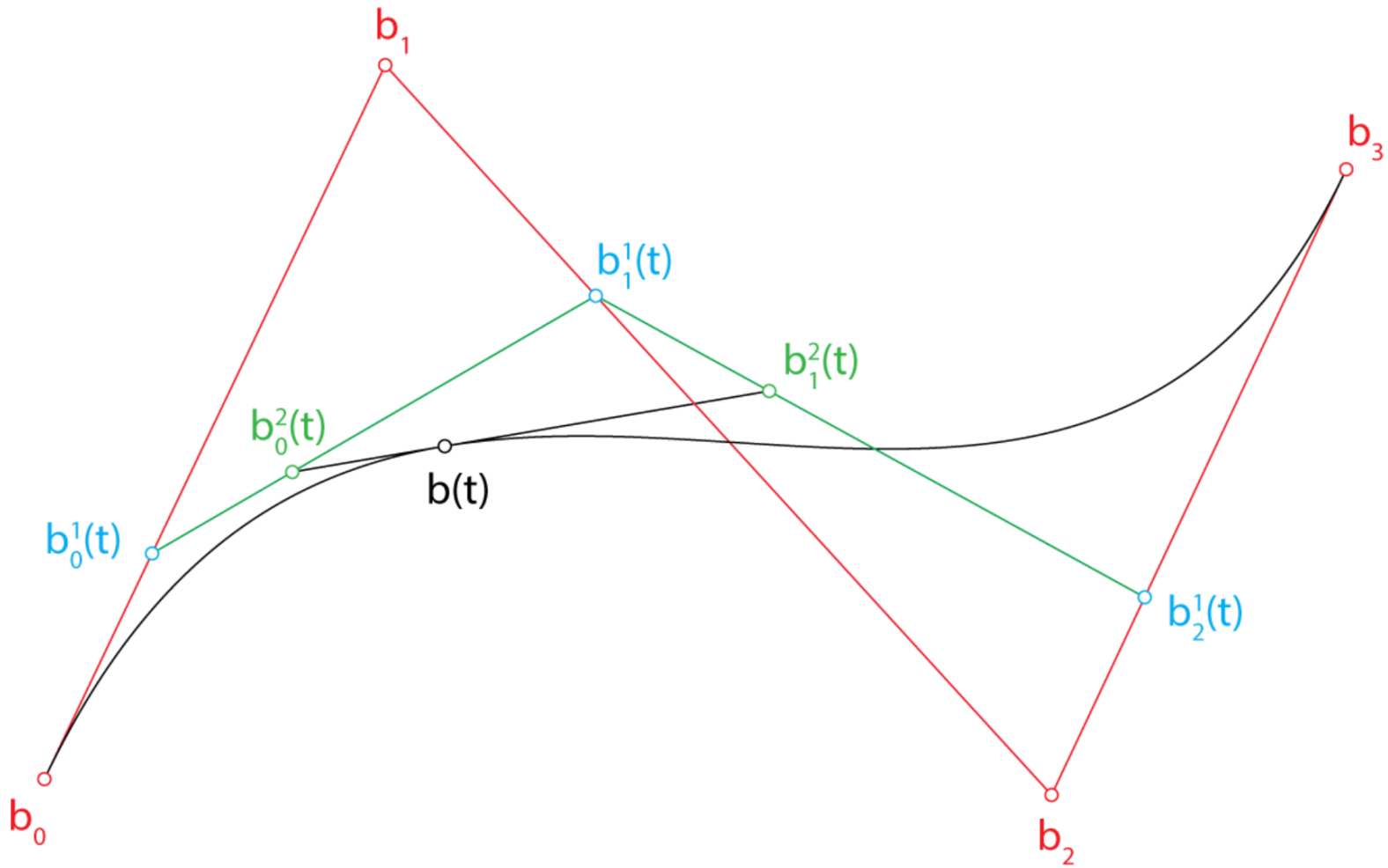


GETTING STARTED

- Clone the Animator skeleton code
  - `git clone git@gitlab.cs.washington.edu:cse457-20sp/animator/YOUR_REPO.git animator`
  - Note: if you want to include any extra credit from Modeler, you'll have to copy or merge that code over
- Animation tab in the bottom window
  - Left: Keyable properties for the selected object
  - Right: Graph window
  - Bottom: Time slider
- Interface represented by **AnimationWidget** - add extra UI here

DEMO

CURVES



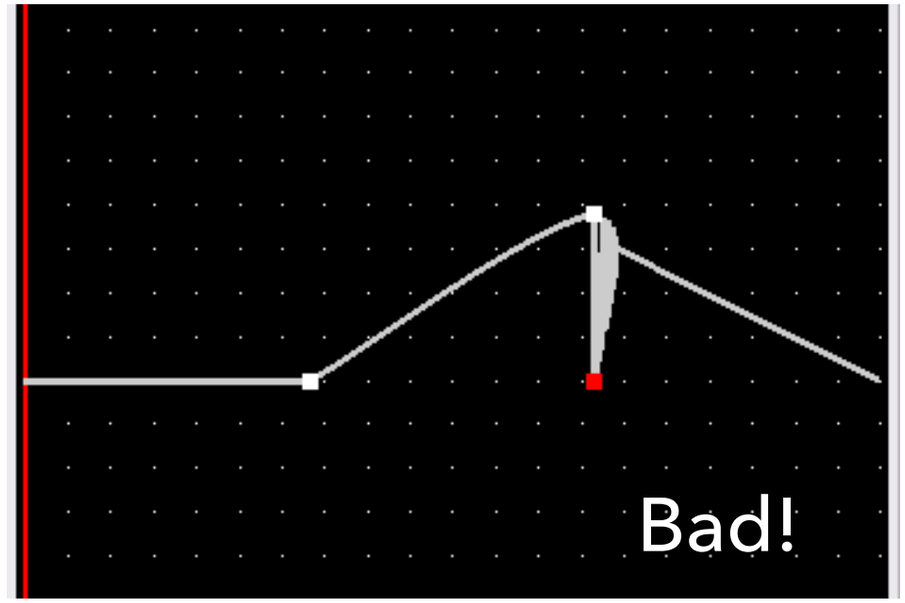
CURVE EVALUATOR

- Implement the evaluateCurve function for each curve
  - `ctrl_pts` - a sorted collection of control points that the user specifies in the graph editor
  - `density` - how many times to sample between control points
- Note the CurveEvaluator is constructed with:
  - `max_x` - animation length in seconds
  - `wrap_y` - a flag for whether to wrap end to beginning (EC)
- Use the LinearCurveEvaluator code as an example

REQUIRED CURVES

- Bezier
  - Adjacent Bezier curves shares endpoints
- Catmull-Rom
  - Interpolate endpoints (double them)
  - Make sure your curve is a function!!
- B-Spline
  - Interpolate endpoints (triple them)

HOW IT WORKS



- Control points are sorted for you
- Your evaluated control points will also be sorted, so…
  - They must be a function! x should not decrease.
- Evaluation function draws line segments between each of your evaluated points to create a smooth curve
  - Use control points to calculate your evaluated points which draw your curve - should always extend from time 0 to animation_length
  - How might you calculate evaluated points so your curve wraps?

BEIZER CURVES

$$b_0^3(u) = (1-u)^3$$
$$b_1^3(u) = 3u(1-u)^2$$
$$b_2^3(u) = 3u^2(1-u)$$
$$b_3^3(u) = u^3$$

- Use the Bernstein polynomials from lecture
- Use linear interpolation when there are not enough control points (< 4 for a set)
- Base requirement: sample u at regular intervals for 0 <= u <= 1 (use the density parameter)
  - EC: Adaptive subdivision with de Casteljau's algorithm (see website)

CATMULL-ROM CURVES

- C^1 continuity
- Similar to Bezier, but now you evaluate a transformed set of points
- Use linear interpolation when there are not enough control points (< 3 for a set)
- Double your endpoints to interpolate!

B-SPLINE CURVES

- C^2 continuity
- Another transformation on your set of control points (called de Boor points)
- Use linear interpolation when there are not enough control points (< 3 for a set)
- Triple your endpoints to interpolate!

PARTICLE SYSTEMS

REQUIREMENTS

- Use Euler's method to update position/velocity (see lecture notes)
- 2 distinct forces
  - Calculate using different equations (ex. gravity and drag are distinct because gravity eq is of form f=ma, whereas drag is defined in terms of a drag coefficient and velocity)
- Collision detection with sphere and plane
  - Use the restitution constant given by UI slider
- Should behave properly when parented within your hierarchy

PARTICLESYSTEM CLASS

- Skeleton provides rough outline - fill in the REQUIREMENT sections to properly run and update the simulation
- Should have pointers to all particles and a marching variable (**time_to_emit_**)
- Suggestion:
  - Particle class - use inheritance if you plan on making multiple types of simulations
  - Force class - perhaps a generic Force class and a variety of distinct forces that inherit from it
    - It’s also possible to model collisions as forces

MAKE CALCULATIONS IN WORLD SPACE!

- If you spawn your particles from a node in your hierarchy that isn't the root, it should still behave correctly
- Find the world coordinates for your particles - not local
  - Why? Ex. If we apply gravity in the local coordinates of your particle system, then the force in the -y direction is dependent on the orientation of that node, not the -y of the world
  - Apply the model view matrix (i.e. **model_matrix_**) to your position, velocity, etc. vectors
- Do the same with your collision forces

NOW MAKE IT EVEN COOLER

- Curves
  - Tension control for Catmull Rom
  - Allow control points to have (or not have) C0, C1, C2 continuity
  - Curve wrapping (UI provided already)
- Particles
  - Cloth simulation
  - Flocking
  - Billboarding
    - And transparent textures -> Fire, snow, leaves
  - Baking
    - Improves performance for complicated simulations with many particles

TIPS FOR GOOD ARTIFACTS

LIGHTS CAMERA ACTION!

HAVE A PLAN

- This artifact takes more time than the others - we give you a week
- Keep it simple, have realistic goals. If you finish early, go back and enhance
- Sketch out storyboards and key poses/frames before implementing
  - Much easier to iterate on paper than in the animator program
- Complicated != better. Well animated simple models are more entertaining than poorly animated complicated models
- Read John Lasseter's article on animation principles!! https://courses.cs.washington.edu/courses/cse457/15sp/projects/animator/linkedItems/lasseter.pdf

TIPS FOR YOUR MODELS

- You may update or add more models as you like
- Many modeler artifacts were not properly “rigged”
  - Fix this now or else you won’t be able to animate
  - Ex. body parts have joints. If it bends, use either a sphere node or an empty node.
    - Translate the child to where you’d like it. Now when you rotate the parent (joint), your child node pivots correctly
- A blinn-phong shader with texture mapping can add a lot, and is fairly easy to implement
  - Look at the provided texture.frag and texture.vert as reference
  - Find or make your own textures by using checkers.png as a reference for how the texture is mapped on your 3D objects (and then use Paint, GIMP, Photoshop, etc.)

CHOICE OF CURVES

- Catmull-Rom is usually the preferred curve choice
  - But unless your project supports the option to add C1 discontinuity at will, you might find yourself fighting the Catmull-Rom to create pauses and control the timing
- Bezier spline works well for things like animating a bouncing ball

IMPORTANT COMPOSITIONAL COMPONENTS

- Timing
  - Consider timing and shot planning before getting specific about joint rotations or positions
  - Total length **MUST** be < 60sec. We recommend 24 or 30 fps.
- SFX + Music
  - Greatly enhances cohesion of your artifact
  - If your idea includes a theme or stylization, very effective to time your animation with events in the theme music
- Lighting
  - Like sound, super important compositionally - can signal story and mood
- Camera Angle
  - Changing perspective between two shots or panning/zooming camera can add depth
  - Do not go overboard! And remember the *180 degree rul*e.

PUTTING IT TOGETHER

- Make sure you keep your original model .yaml file separate
- We recommend breaking up your intended artifact into shorter clips or “shots” and combining them in the end
  - Easier to split up work
  - Can incrementally complete your artifact
  - Save a new .yaml file for each shot, and build off the base of your original model (or from your last shot)
- **SaveAs often** - there are no undos
- Blender is installed on the labs and we provide a tutorial
  - Adobe After Effects and Premiere can also composite your frames into a movie - and much more easily too
  - < 60s, and must be H.264 mp4 format

THE END

GOOD LUCK