Project 1 for CS3451 2018

Jarek Rossignac

1 CS3451—Fall 2018—Project 1

1.1 Three phases (one week each)

You will have 1 week per phase. But please try to do phase 1 and 2 right away, so you have time to do and document phase 3. You can collaborate, but **each student submits his/her code, video, write-up.**
All submissions are **due on due day BEFORE class starts** .

1.1.1 Phase 1, due August 30: Cubic Neville curve and animation (code+video)

In your version of the code, change **names** and **photos** to be **yours**
Make sure that the name of your **team partner** shows on the help screen
Implement cubic Neville (a few lines of code, should be trivial, but test it!)
Capture short **video** (individual) while you edit the control points and animate the circle
Submit **zip** of YOUR INDIVIDUAL code and YOUR video
    You can collaborate and help each other, but must EACH **submit private code and video**
You will receive up to **30 points** for this if your code is sufficiently simple/elegant and works correctly and if you have submitted everything on time. These are easy to get points, so make sure that you get them.
If you fail, but submit by the following week, you will get at most 20 points for this.

1.1.2 Phase 2, due Sept 6: Uniform, Chordal, Centripetal knot computation (code+video+report)

In your version of the code, implement the **three knot computation methods**
You may collaborate, but EACH student **submits his/her own code, video, write-up**
    Compare results to make sure that your implementation is correct
    Discuss with partner what you see as the pros/cons of each method
BUT: produce **your own report** with your figures, discussing +/- of each knot computation method
Create your own video showing the most important advantages of each method
You will receive up to **30 points** for this, if you submit on time, if your code is correct, and if you discussed and showed (in the images and/or video) most of the advantages/drawbacks of each method.
If you fail, but submit by the following week, you will get at most 20 points for this phase.

1.1.3 Phase 3, due Sept 18: Your extensions (40 points max)

The two **members of a team cannot** submit **similar extensions** (doing so will split the received points)
Collaborate: ***give each other feedback on extensions, on video and on write-up***.
Each student submits a zip of his/her own code that shows his/her own extensions.
Make sure that the help screen **explains how to use it** and that it loads the file for a **nice example**.
Submit a **report**: that explains what your extension does, how you implemented it. Include pictures.
Submit a short video with a title screen that has your name and the list of your extensions.
You will receive a total of up to **40 points** for this effort if you submit your code, video, and write-up on time.
No submissions for project 1 will be accepted after that final deadline. So, to be sure, start working on phase 3 early, **back up your work often**, and **submit a preliminary version 2 days ahead of schedule**.

1.2 Suggestions for extensions (ask about them in class)

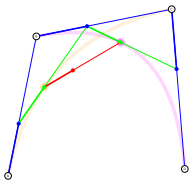
1.2.1 Examples of easy extensions (10 extra points each)

Support editing of two cubic curves and **animate** the **edge** joining a moving point on each
Add **time-warp function** for slow start and finish (use cosine or cubic time warp)
Change the time manager code in main() to have the disk go **back and forth** during animation

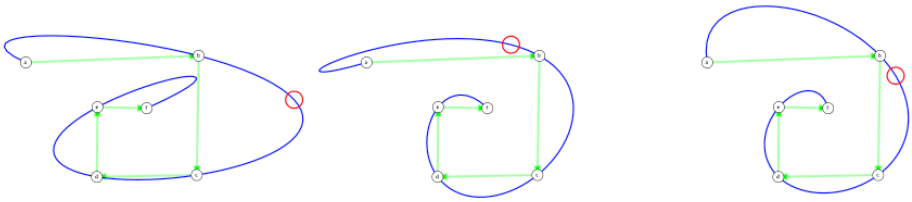
1.2.2 Medium extensions (between 15 and 25 points each)

Implement an **animation** that **explains** how and why the cubic **Neville algorithm works**

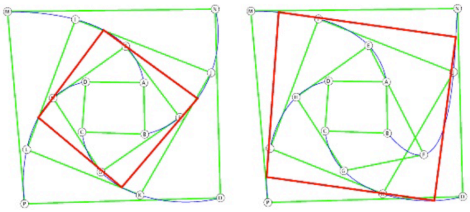
You may use, as guideline, the animation provided in the basecode for Bezier cubics



Extend the scheme from 4 to **6 control points** (see Quintic Neville below) and show (video) what is good and bad about it.



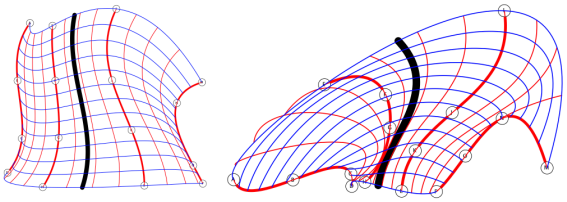
Support four keyframe control **quads** and animate a **morph** (red quad) through them



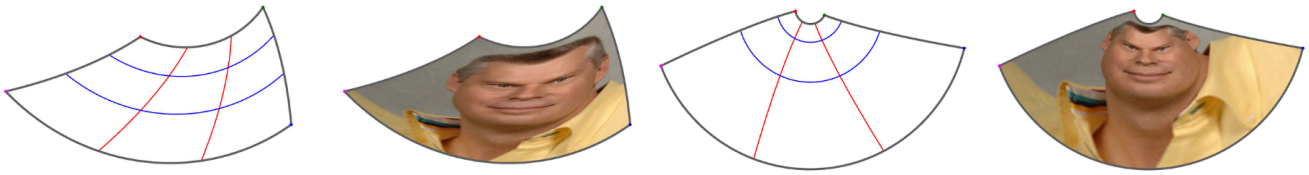
Use second cubic (separate mode) to let the artist control the time-warp via an **animation curve**

1.2.3 Hard extensions (40 extra points if done right and documented adequately)

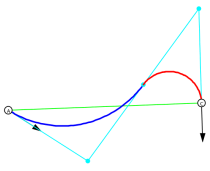
Support 4 keyframe Neville curves and **animate curve** interpolating them. Explain which knot paradigm you use in each direction and why!



Support 4x4 control grid of control points and **a tensor product bi-cubic surface** or a Coon's patch with cubic Neville border edges, and use it to warp a texture.



Propose another interpolating curve, make it work, and demonstrate its advantages. For example, make a smooth curve that is made of two or three smoothly joint **circular arcs** interpolates the four control points. Explain what your construction optimizes and how.



2 Basecode (to be reviewed in class)

Download and unzip "Basecode Points Vectors Textures 2018 August 1.zip"
Start processing and open it.

2.1 How to use it

Run |>

Help

screen with pix of student and names

```
CLASS: CS3451 Fall 2018

PROJECT 1: Neville animation

STUDENT: Jack Ross

PARTNER: James Bond

MENU <SPACE>:hide/show

POINTS click&drag:closest, x:moveAll, z:zoomAll,

POINTS ]:square, /:align r:read, w:write

DISPLAY f:fill, #:Point IDs, v:vectors

ANIMATION a:on/off, ,/.:speedControl

Bezier b:curve, B:constructon

Neville n:curve, N:constructon

KNOTS 0:uniform, 1:chordal 2:centripetal

FILENAME FN C:set to content of clipboard

CAPTURE CANVAS to FN ~:pdf, !:jpg, @:tif, ':filming restart/stop

4 QUADS: 4:squares, R:read(FN), W:write(FN), f:fill, t:texture
```

SPACE to hide/see help screen

Edit & save control points

Click&drag to move closest control points
x to move all, z to zoom
mouse wheel
w to save (write) config of control points to file
r to read it back

0, 1, 2 to switch method for computing knots b and c (nothing happens: not implemented)

Sketch>showSketchFolder.. parent.. IMAGES…

Animation

a to start/stop
, or . to adjust speed

Pictures and videos

~ , !, @ to make pictures in different format
` to start/stop filming (film only a few seconds)
Saves frames in /IMAGES/MOVIE_FRAMES_TIF_DELETE_AFTERWARDS
Tool > Moviemaker to make a movie

Delete the frame images (Do not submit them!!!)

n Hide/show Neville
v hide/show vectors between control points
b Hide/show Bezier
B Hide/show Bezier construction
Advanced
16 control points
    4 to make nice squares (that you can edit)
    R to load saved points from file
    C to set filename for saving them from clipboard
    W to save them into that file
    R to load from that file
f fill quad
t Texture map

2.2 What is in the code and where

2.2.1 Do not complain or criticize

You do not have to use this code and are free to write your own, provided that it is clear enough for the TAs to understand and that you are not using code developed by others for the key parts (Neville interpolation, knot methods, and your various extensions).

If you decide to work with the basecode provided, feel free to:

- reformat it as you like.
- restructure it and add comments.
- improve and add functionality.
- add classes (but I suggest that you think about your extension before you design these)

Please do NOT use the PVector library nor any library or code that computes or displays interpolating curves!

The code uses files in the **data** folder and has several **tabs** (each in a different .pde file)

2.2.2 Data folder

Must be in folder "main" and **must contain**:

- you picture in studentFace.jpg (small resolution) (and partner's)
- points.pts , which contains the number and coordinates of the control points
- SixteenPoints.pts, if you want to use 16 points (read it by pressing R)
- AdobeFanHeitiStd-Bold-32.vlw which contains the font I use

2.2.3 Main tab

The name of that file must be same as folder name! I call it "main" and put it in a parent folder for which I change the name to keep track of the different versions (base code, my solution, demo of this or that…).

Libraries

Global variables:
    Colors
    Files

Image for title/help page and for texture in animation
Method used: 0, 1, 2 for the knot values
Boolean toggles for behavior control, animation duration and current time, control points,
Points: A, B, C, D, and P are just **pointers** to points, not assigned yet

2.2.3.1 setup

setup() executed once to create Point objects, read coordinates, read images…
FrameRate = 30 fps
Read control point locations from file into PNT array Point[], declared line 45 in tab points
Makes A, B, .. point to entries in Point[] : for convenience and future extensions to more points

2.2.3.2 draw

draw() executed 30 times per second (if fast enough)
start PDF recording (all commands) if user want a picture of the canvas
If showInstructions, we do what is in tab "help" (background, photo, title, **name**, **partner's name**…)
Else… displays frame (for the current time of the animation)
Erase screen by painting the background
If animating, update current **time** for animation
If we have 4 control points (not 16, which may be used for extensions). Renter that mode by pressing 'r'
- compute knots according to chosen method
- This is where you add your code (for UG)
- Normalize knots to [0,1], provided, so you don't forget
- Draw green vectors AB at A, BC at B…. using a loop on Point[i]
- Draw Neville cubic curve in blue using polyline with 90 vertices
- Compute and show current position of point Q on curve for current time as a red circle
- Draws 9 intermediate dots along edge [A,D] as small dots (DISCUSS "float")
  - Use this as an example, but draw them on the Neville curve instead
- Show the 4 control points
  - Using letter names or
  - Using small dots
- Bezier (provided for demo and example)
  - Curve
  - Construction

If we have 16 control points
- This is for grad students and for extensions, enter that mode by pressing 'R' or '4'
- But if you press 'w' in that mode, then you will overwrite the file with 4 points, so be careful, save a copy!
- Draw 4 quads
  - Use a color ramp to distinguish them
- Draw correspondence lines between their vertices
  - using semitransparent grey lines (to see what is underneath)
- Show control points
  - Using letter names or
  - Using small dots
- Compute corners of the animated morphing quad
- Draw quad using texture my face as map
  - or as semitransparent outline

Finish capturing the PDF picture
Snap pictures of this frame if user requested them

2.2.4 GUI

Keys performing actions or toggling states
Most keys arranged by rows (in keyboard order)
Picture capture of canvas
Show/hide control-point labels
Video capture of canvas
Change method for knots
‘4’ makes 4 squares that you can edit
‘r’ to load back the 4 point config
Animation (show for 4 and for 16 using ‘4’ and ‘r’)
Fill yellow (when 16)
Texturing
‘w’ to write and replace the saved points
‘C’ using “FourS” in the clipboard, then ‘R’
    Show textured animation
‘,’ and ‘.’ to adjust animation speed
SPACE show instructions (title screen)
Println() and key and keycode

When mousePressed, assign P to the closest control point
    Loop over control points
    Update selection by comparing distance

When mouseDragged,
    If no key pressed: teleport P (the control point that it is assigned to)
    But if x or z is pressed, edit all the control points by the same transform (translate, zoom)

MouseWheel can be used to zoom of all points (using LERP from center)

2.2.5 Help

Background
Picture(s)
Line counter
Text

2.2.6 Points

Functions that create points
LERP (useful for Neville)
Distance
Drawing geometric primitives using beginShape, vert(), endShape
Writing label
Texture mapping

Point Class and methods
Point Array processing
    Point[]
    counter
        Points must be declared (create the object for each point) before you use them.
        Reading them from file does not create them, just replaces their coordinates.
        ReadPoints prints message on bottom pane, useful for debugging sometimes
    MakeSquares
    Write
    Read

2.2.7 Curves

Neville functions: for you to edit
Bezier (for demo)
    DrawCurve
    ShowConstruction ** This stops mouse pick&drag for some reason (still in vestigating)

2.2.8 Quads

LERP between 1st and last quad: To demo animation for extensions (which should interpolats all 4 quads, not just the first and last)

2.2.9 Tools

Text on screen
Pictures and directions for making movies
Clipboard (used to change filename for 'R' and 'W' and saving pictures)

2.2.10 Vectors

Create from components or from other vectors or from pairs of points
Add (scaled) vector to point
Measures: norm, angle , dot, det
LERP vs LPM (log-spiral interpolation between vectors: for grad students)
Display vectors as lines or arrows
Class

2.3 Using the base code for the project

Download Processing: https://processing.org/download/
Download and unzip "Basecode Points Vectors Textures 2018 August 1.zip"
Unzip and run
Understand how to use it (I'll demo it and explain some of it in class)
Understand its structure
For Phase 1 and 2:
Replace a few lines where it says: //**UG** ADD YOUR CODE HERE
    In tab "curves": line 5, 11, 17 (Neville)
    In tab "main": lines 113 and 118 (knots chordal and centripetal)
    In tab "help": line 13 and 14 (names)
    Make sure that you replace my photo with yours in the file /data/studentFace.jpg

and add partner's photo
For Phase 3, make a separate version.
You may load 16 control points (press '4' or 'R').

2.4 Programming with Points

Create: PNT A = P(100,200); // Canvas coordinates = (right,down)
Change coordinates: A.setTo(150,180);
Drag with mouse: A.teleport();
Access coordinates: P.x, P.y
Display small green disk: fill(green); noStroke(); drawCircle(P,6);
Display red circle: noFill(); stroke(red); strokeWeight(2); drawCircle(P,r);
Show as label in circle: s howLabelInCircle(A,"a");
Measure distance |AB|: float d = dist(A,B);
Create point on line(A,B) at parameter t from A with unit=|AB|: PNT P = LERP(A,t,B);
Draw curve C, assuming that C(t) returns a point on C at parameter t
beginShape(); for(float u=0; u<=1.+du/2; u+=du) vert(C(t)); endShape();
Example: How to draw 8 small dots uniformly spaced between A and B?

2.5 Programming with VECTORS

CREATE
VCT W = V(100,200); // from coordinates
VCT W = V(A,B); // between two points
VCT W = V(U); // copy

DISPLAY VECTORS
arrow(P,Q);
arrow(P,V);
drawVectorLineFrom(P,V);

CREATE BY TRANSFROMING ANOTHER VECTOR
VCT U = Rotated(V,PI/6);
VCT U = Scaled(1./100,V);
VCT U = Divided(V,100);
VCT U = Normalized(V);

MEASURS FROM VECTORS
float d = normOf(W); // magnitude
float a = angle(V,W); // cw
float c = dot(V,W); // |V| |W| cos(V^W)
float s = det(V,W); // |V| |W| sin(V^W)

WEIGHTED SUM
W = Sum(U,V); // U+V
W = Sum(U,v,V); // U+vV
W = Sum(u,U,v,V); // uU+vV

INTERPOLATION
W = LERP(U,t,V); // Linear
W = LPM(U,t,V); // log-polar morph (steady)

CREATE POINT AS POINT+VECTOR
PNT B = P(A,V); // A+V
PNT B = P(A,v,V); // A+vV