# Final Project: Animation-Off

CMSC 23700: University of Chicago
Final Exam: May 24th, Wed 10:00 AM-12:00 PM
Pre-Submission: May 7th

## Introduction

Congratulations for making it this far in the course! Your final project will be much more open-ended then your previous assignments. At a high-level, you will be turning in a 10 second animation (movie clip), that you rendered based on your own creations and implementations. We will convene during the final exam period (**May 24th**), and watch everyone's animations. We will vote on the best animation, and the winner will receive a special prize that was sent to me by Pixar. Besides bragging rights, the winner will also get their video clip and a little blurb featured on the course website. See last year's write up.

## Submission Format

1. An animation.mp4 file, which contains a video animation
2. All python files used to generate the animation.mp4 file
   (a) The files must contain a main.py which we can run to generate an output animation.mp4
3. a WRITEUP.md file

## Pre-Submission Instructions

We will first have everyone get up-to-speed on the basics of creating an end-to-end rendering with Blender through a pre-submission. In this part, your pre-submission will contain: 1) a 1-paragraph description describing what you plan to do for your final project. This will describe a rough outline of your "animation story-line" plus what components you plan to implement. 2) A 1 second animation (animation.mp4) of anything of your choosing (and the source files needed to generate it). 3) Implement BSplines (more information below).

Tips: Your final project plan is not a binding contract, rather we want you to start to think about what you might do! We do expect that this may change. The goal of submitting a 1-second .mp4 animation is to ensure that you are able

to generate animations end-to-end using Blender. You are encouraged to use your implemented BSpline to interpolate attributes in your 1-second animation file (e.g., change colors, object locations, etc).

# Final Submission Instructions

You will be submitting a 10-second .mp4 file. You will also submit all the python files required to generate the .mp4 file. The files can take any format you would like, but have a main.py which we can run to generate an output animation.mp4. You can work on the assignment up until the final exam. The files must be submitted before the exam period begins.

In this final project, you may use additional packages. For example, if you would like to use a different renderer, or a different way to save images to a video file. In general, it is OK to use a package **so long as the package does not implement the assignment for you**. You may ask course staff if you are confused about what packages you can use and how. In addition, please prepare a small write-up describing what you did. The write up should be in markdown format, for example, call it WRITEUP.md. If you would like a nice online markdown editor, we suggest hackmd. IDEs like PyCharm and VSCode can also render markdown.

The only strict requirement in this assignment is that you should implement the BSpline portion mentioned below (which you should submit as part of the pre-submission). In addition to the BSpline portion, you will implement other computer graphics algorithms of your choosing. We have provided a list of ideas in this document. However, you are not restricted to this list. If you would like to implement additional algorithms, please discuss your idea with course staff.

**Note, if you are a graduating senior, you may need to turn the final project a bit earlier (more information on the exact date later – please check canvas for an update).**

## Required packages

- python = 3.7
- numpy = 1.24
- pillow = 9.5
- Blender = 3.2.1
- imageio = 2.28.0 (pip install imageio & imageio-ffmpeg)

Note: to install Blender, go to this link and select the version for your operating system and download it. Make sure you know where your blender is downloaded as you will need the path to your blender installation when running blender python.

### Rendering

In this final project we make use of Blender to render our scene. Please refer to the linked documentation for more information about how to control rendering parameters, such as lighting, materials, and other properties.

You are not required to use Blender for this assignment. You may consider other rendering packages as well. However, note that scripting in engines like Unity are **very** challenging, so unless you already have some experience we would not recommend going that route. If you would like to use a different rendering package, please get approval from the course staff in advance. Choice of renderer will not effect your grade.

## Starter code

An example starter code for rendering is provided for you in the blender_sample.py. There are some examples of how to load in an .obj file, set some properties and render an image. There is also an example of how to create a video from a folder of images in the file save_video.py. You can save images to a video in a different way if you'd like, this is only one simple example.

## BSplines

Recall from lecture that we can build a B-spline $\mathbf{f}$ of degree $d$, given $n$ control points $(\mathbf{c}_i)_{i=1}^n$ and $n+d+1$ knots $(t_i)_{i=1}^{n+d+1}$, which we write as:

$$\mathbf{f} = \sum_{i=1}^{n} c_i B_{i,d},$$

where $(B_{i,d})_{i=1}^n$ are the B-spline bases. See also bspline and the lecture notes. We can use B-Splines to interpolate (actually, approximate) control point values. Remember that we can fit a B-Spline to any high-dimensional signal via a 1D B-Spline over each component separately.

You can use B-Splines to approximate between color, camera position, position of objects, lighting parameters, etc. Get creative! Here we walk you through a color approximation example.

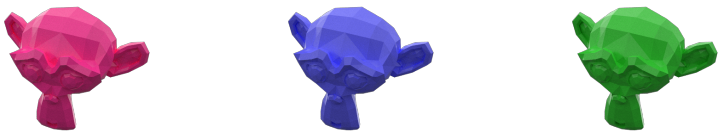

Figure 1: Interpolation (actually, a B-Spline Approximation) between three control points. RGB color values are (left to right): [1, 0.2, 0.6], [0.4, 0.4, 1], [0.2, 0.8, 0.2].

Our spline is for 3-dimensional RGB data, but we fit a spline to each color channel separately. We have three control points: [1, 0.2, 0.6], [0.4, 0.4, 1], [0.2, 0.8, 0.2]. Considering only the red channel, we have [1, 0.4, 0.2]. We also need to define the knots and the degree of the B-Spline. Check the file interpolation.py, which contains the BSpline class. First, you should think about how to implement the is_valid method. Based on what you know from how the equation looks, there are some constraints on the degree, number of knots, and number of control points. Below are a few cases your function should succeed and fail on. Then, implement the bases and interp methods. Bases is recursive, and should be called inside the interp method. You are encouraged to plot and visualize the 1D BSplines. You should also think about how to handle the boundaries (there is no right answer).

```
# this example will not work
d = 2
num_knots = 7
num_control = 3

# this example will work
d = 2
num_knots = 6
num_control = 3
```

Figure 2: Color approximation results using B-splines. Notice how the input colors are *approximated* rather than interpolated. We do not recover the same exact color inputs!

You can reuse this class to interpolate other attributes, like camera position, lighting, object positions, etc. You may also consider implementing other Spline methods that will interpolate instead of approximate the input values.

## List of Implementation Ideas

Here is a list of ideas of things you can implement, and a ranking of their difficulty. Items with more points are more difficult than items with less points. **You should implement 1 pt worth of items from this list**, or get pre-approval for implementing different ideas. If you implement more than 1 pt

worth of items, you will get extra credit! Please talk to the course staff about alternative ideas you would like to implement, and we will tell you if it is a 1 pt or 2 pt implementation.

We also encourage you to implement more things outside of this list! Get creative, and tell us what you find. Make sure to tell us a bit about what you implemented in a write-up.

- Loop Subdivision loop subdivision [1 pt]
- Bump Map (aka a black and white texture image): displace vertices along normal + vertex insertion source 1 source 2 [1.5 pt]
- Mesh Parameterization (i.e., LSCM from Richard's Lecture) [2 pt]
- Edge collapse calculate quadrics paper [1 pt]
- Edge flip source [1 pt]
- Half-edge implementation/edge collapse that handles boundaries source [1 pt]
- Vertex Split (reverse of an edge collapse) source [2 pt]
- 3D trilinear interpolation (can be used to apply Free-Form deformations to deform shapes, see ALIGNet for an explanation) [1 pt]
- Cotangent Laplacian Smoothing slides [0.5 pt]
- Alternative Spline Method (see "lecture 11 – introduction to animation") [0.5 pt]

Additional useful resource: mesh editing and half edge.

## Custom Animation: Using your implemented components

After you've implemented the necessary components, you should think about how to create an interesting 10 second animation. You can build your own meshes, find open source meshes online, add textures, displace vertices, move objects around the scene, move the camera around the scene, change lighting, colors, etc. You are also welcome and encouraged to use any of the code you built for this and previous assignments as well! Create a visually compelling animation and show it off to the class! Top animations will be featured on the course website. Of course, the winner gets a prize, special feature on the course website, and bragging rights.