Final Project: Animation-Off

CMSC 23700: University of Chicago
Final Presentations: March 3rd & 5th, 5:00 PM-6:20 PM
Pre-Submission: Feb 22nd 11:59PM

Introduction

Congratulations for making it this far in the course! Your final project will be much more open-ended than your previous assignments.

In short, you will be turning in a 10 second animation (a movie clip) that you produce based on your own creations and code, including an implementation of one or more graphics algorithms of your choice. You may include audio to go along with your animation (in fact, this can make the animation look more compelling).

We will convene during the last week of class (**March 3rd & 5th**), and watch everyone's animations. You will select your preferred presentation date by filling out a Google form (first come, first served) which you will receive in class. We will then assign you to one of the two days; roughly half the class will present their final animations on each day. **Your deadline for uploading all your materials is your assigned presentation date.** Crowd favorite animations will also get their video clip featured on the course website. See the writeup from 2022.

Final Submission Format (Due March 3rd or 5th)

1. An `animation.mp4` file, which is a 10s video animation
2. **All** files used to generate the `animation.mp4` file
   (a) This will likely include `blender_sample.py`, `interpolation.py` and `save_video.py`.
   (b) If you created any other files to implement your other algorithms, or made any changes to `blank.blend`, **you must include those as well**
   (c) If you decide to use a renderer other than Blender, your files will look different, just make sure to include everything needed to generate your animation
3. a `writeup.pdf` file, where you:
   (a) describe what you did in producing your animation, the graphics algorithm(s) you implemented, any challenges encountered and steps you took.
   (b) include a diagram that illustrates or has aided your implementation of the graphics algorithm(s) you chose to implement.

Pre-Submission Instructions (Due Feb 22nd)

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. Your pre-submission will contain:

1. A 1-paragraph description in a file **`plan.md`** describing what you plan to do for your final project. This will describe a rough outline of your "animation storyline" plus what algorithms 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. Your B-spline implementation in `interpolation.py` (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 B-spline 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, just make sure to include all the files necessary to generate an output `animation.mp4`. You can work on the assignment up until your presentation date. **The files must be submitted before class begins on your assigned presentation date.**

Every submission should implement the B-spline portion mentioned below (which you should submit as part of the pre-submission). In addition to the B-spline 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.

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 and submit it as **`writeup.pdf`**. As in your previous assignments, in your writeup you must include a diagram that illustrates or has aided your thought process and implementation of the graphics algorithm(s) you chose to implement.

Required packages

- python $\geq$ 3.8

- numpy $\geq$ 1.21
- pillow $\geq$ 10
- Blender $\geq$ 4.2
- imageio (`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. If you have issues running the script, this documentation may be helpful.

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 may be 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 affect your grade.

Starter code

Example starter code for rendering is provided for you in the `blender_sample.py`. This code includes instructions for the command you should run to render using Blender, how to load 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.

The starter code also comes with a Blender file, `blank.blend`. If you use Blender for this project, you will be using `blender_sample.py` to manipulate the scene defined in `blank.blend`. This file defines completely empty Blender scene, but we've included some basic code in `blender_sample.py` that defines a light, plane, and camera (along with the script for importing an obj), as well as some examples of how to define materials and animations. Feel free to add to, edit or delete this code as you see fit to make your final video as expressive as possible. However, if you are already comfortable with Blender, or are interested in learning it, feel free to make changes to `blank.blend` using Blender itself. Note that **you may not use Blender to implement your B-splines or final algorithm(s), these should be implemented in your Python code**.

B-splines

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.

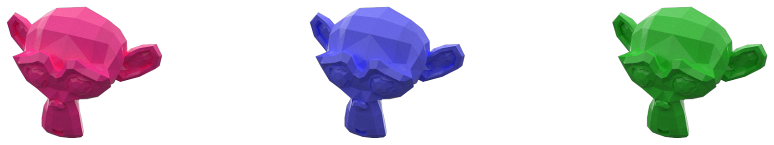

Figure 1: An example of three control points. RGB color values are (left to right): [1, 0.2, 0.6], [0.4, 0.4, 1], [0.2, 0.8, 0.2].

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 B-splines.

```python
# 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
```

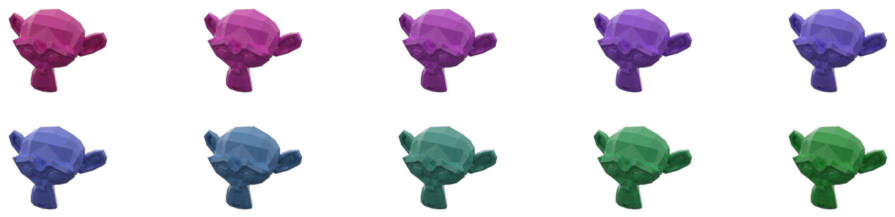

Figure 2: Color approximation results using B-splines. Notice how the input colors are *approximated* rather than interpolated. We do not recover the same exact color inputs!

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 in addition to B-splines. Items with more points are more difficult than items with fewer 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 your writeup.

Important: You must write your own implementation of the technique(s) of choice. Some of these may be implemented already in Blender as modifiers/nodes/functions you can call, but **using Blender's (or other existing software package's) implementation of a technique does not count as implementing it yourself.**

- Loop subdivision: source [1 pt]
- Displacement mapping: displace vertices along normal, optionally after subdivision source 1 source 2 [1 pt]
- Mesh parameterization (i.e., LSCM from Richard's Lecture) [2 pt]
- Edge collapse with calculation of quadrics to select edges: paper [1 pt]
- Edge flips: source [0.5 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]
- An alternative spline method (see Lecture 11 – Introduction to Animation); note that you can only implement one additional spline [0.5 pt]

Additional useful resources on mesh editing and the halfedge data structure.

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. The winner gets a special feature on the course website, and bragging rights.

Ideas for showcasing a graphics algorithm in your animation. You may be wondering how you would ever turn mesh processing operations (e.g. edge flips, vertex splits, subdivision) into a compelling visual feature of your animation. A good approach we've suggested and seen in the past is rendering your mesh **with the wireframe visible**, showing the progress of the algorithm on the mesh as the camera pans around the mesh or while the mesh moves around.

(To be clear, you do not have to make your algorithm the main point of your animation, you just have to have used it in the production of your animation.)

Don't lose points on your animation!

Please read these rules carefully. Below you will see ways that you can lose points on your submission, and the point amount.

1. The animation (including the audio) may not contain any derogatory/explicit language; the animation itself must not be explicit, suggestive, violent, or anything that seems classroom-inappropriate. **Penalty: zero on the final project.** If you have any questions about what might be deemed appropriate please ask us directly!

2. Missing the final presentation. **Penalty: 30% off the final project.** Please note that this excludes extenuating circumstances, which should be coordinated directly with the college advisor.

3. Late submission. **Penalty: 30% off the final project.** Please note that this excludes extenuating circumstances, which should be coordinated directly with the college advisor.

4. The submission in Gradescope should pass all the Gradescope tests – which are simply an existence check for each of the necessary files **Penalty: 5% off for each missing file in Gradescope.** For example, if you are missing an `animation.mp4` and a `writeup.pdf`, you will get 10% off your grade.

5. You must include your animation.mp4 in *both* the Google Drive folder and the Gradescope submission. **Penalty: 5% off of the final project.**

6. You must describe in your documentation.pdf where EACH of your decided implementations are. For example, please point us to the file and the function names. We should be able to easily navigate into the file and view your implementations on gradescope. **Penalty: 10% off of the final project.**

7. Do not tar.gz any of your python/code files that are needed to review the implementation component of the project. (If you need to compress other data files, that's fine. Just describe in your writeup how they are organized and used in producing your animation.) **Penalty: 10% off of the final project.**

8. You must implement the underlying logic in your chosen graphics algorithms (from the above list). *You cannot simply call the Blender API (or other existing package/library) to run the algorithm for you!* **Penalty for calling Blender etc. API for the graphics algorithm implementation part: 30% off of the final project.**

   Note: you may use the Blender API to apply *additional* graphics techniques in your animation, but they won't count for the implementation component of this project.