Benjamin Eskildsen
2/11/2016
Computer Science Senior Project Proposal
Advisor: Dr. Holly Rushmeier

In supervised machine learning, the computer can approximate a desired solution by either comparing the output it generates to training examples or by evaluating the output with some objective fitness function, but often times in computer graphics, if any examples exist at all then the problem has already been solved and an objective fitness function is impossible because the graphics are, by nature, subjective. Currently, if someone is trying to make a detailed and procedurally-modelled computer graphics artifact, then they have undertake the time-consuming process of manually tuning individual parameters of their rendering program to see if those changes bring the output closer to what is desired -- essentially just guess-and-check. Stanford Phd student David Ritchie's work on procedural modelling uses constraints, sequential monte-carlo methods, and other optimizations to speed up this creation process, but artistic aesthetic is eschewed in favor of speedy and automatic generation. The goal of my project is to create a webgl-based computer graphics environment that keeps the artist in the loop so that the final product can leverage computer-aided design strategies that speed up creation while maintaining artistic quality.

If an artist is trying to render a very realistic-looking computer model of quartz crystal, there are many parameters that would need to be tuned: color, different types of reflectance, transparency/translucency, rate of generation and size of crystal columns, even the cloudy impurities within. To do this manually would require either highly detailed knowledge about

quartz crystal, or extreme patience (and likely both). My artist-in-the-loop graphics environment will prioritize the tuning of parameters to allow the artist to easily explore the space of possible models. While writing the webgl code, the artist will specify which program variables should be tunable rather than arbitrary constants. When run, the model will be shown along with a sidebar containing sliders for all of the tunable parameters so that the artist can immediately see the impact of changing constants (eg. see the effect of changing transparency). In order to compare multiple possible renderings, the artist could select a parameter and have the program display a range of possible models sampled from the given domain, all side-by-side. If the artist selected a second parameter, then a 2-dimensional matrix would appear showing samples that vary those parameter values across the two axes (eg. to see how color and translucency interact). I also would like to use active learning algorithms so that the computer could suggest eg. a series of paired samples and have the artist select which is preferable, more efficiently narrowing in on the subjectively best model.

One potential challenge for the project is that oftentimes ray tracing is the best way to achieve hyper-realistic rendering, which can take several seconds to render even relatively simple examples. This could potentially make the environment aggravating to navigate especially when many samples must be rendered side-by-side. To get around this issue I can see three options: leverage the fact that repeated renderings will usually be very similar and only differ in one variable at a time to reduce redundant computations (Disney (among others) uses a method called path-tracing that takes a similar approach); stick to rasterization rather

than ray-tracing and optimize for that technique; or tolerate long-ish rendering times. I will try my best to implement the first option, but a combination of 1 and 3 is not unlikely.

I plan to implement the project with rapid prototyping so that I can figure out which features should be fixed and which features are missing entirely. I want a version of the most basic slider-based parameter tuner ready in one or two weeks, in another two or three weeks I could have a prototype active learning algorithm running. I can spend the remaining month of the semester refining, debugging, or adding additional features. Somewhere in there I will take a look at how to optimize the ray tracing process, but given its uncertainty, there's not a good way to put a deadline on it. I'll also have a rendering project or two going on in parallel to test the usefulness of features and prove the project's effectiveness in the end. The deliverables will then include a graphics rendering made with the environment along with the graphics environment itself.