B-Spline Approximations – Optimization Details

CS779 Final Project by Kaleb Alway

1 Overall Pipeline

My B-Spline approximation program takes the following steps to compute an approximation. Below, n refers to the degree and p refers to the number of control points.

1.1 Sampling

The program samples the input curve. It uniformly selects N sample points $S_1, \cdots, S_N$ from the input data, where N is a compile-time constant. This was chosen to be a constant so that long curves do not result in very large optimization problems which can take a long time to solve.

1.2 Degree and Control Points

The program aims to minimize the following function:

$$f(n, p) = (c \cdot n + d \cdot p + 1) \cdot g(n, p)$$

where c is a user-controlled parameter adjusted by the "Degree Param" slider, d is a user-controlled parameter adjusted by the "Points Param" slider, and $g(n, p) = \min_{P_1, \cdots, P_p} \text{dist}_n(P_1, \cdots, P_p)$ is the minimum distance (according to the currently selected distance function, discussed below) between the B-spline with degree n and p control points and the sampled points.

A gradient descent method is used to approximate this minimization problem. The gradient is approximated by solving the minimization problem $\min \text{dist}_n(P_1, \cdots, P_p)$ for the surrounding values of p and n.

1.3 Parameterization

The program selects N parameters $u_1, \cdots, u_N$, each corresponding to one sampled point, according to the currently selected parameterization method (discussed below).

1.4 Knot Selection

The program uses the parameters to compute the knot vector. Since we want to interpolate the endpoints, the first n and last n knots are fixed to 0 and 1 respectively. The rest of the knots are computed by the *averaging method* as follows:

$$t_j = \frac{1}{n} \sum_{i=j-n+1}^{j} u_i$$

1.5 Optimization

With the parameterization and knot vector set, the program minimizes (or approximately minimizes) the distance function to obtain the optimal B-spline control points.

2 Parameterization Methods

2.1 Uniform Parameterization

Uniform parameterization is quick and simple. The parameters are set as

$$u_i = \frac{i - 1}{N - 1}$$

for $i \in \{1, \cdots, N\}$.

2.2 Chord Length Parameterization

Chord length parameterization makes the distance between knots proportional to the distance between sampled points. The parameters are set as follows:

$$u_i = \frac{\sum_{j=1}^{i-1} ||S_j - S_{j+1}||}{\sum_{j=1}^{N-1} ||S_j - S_{j+1}||}$$

2.3 Centripetal Parameterization

Centripetal parameterization is a generalization of chord length parameterization which introduces another level of user customization via the variable α. The parameters are set as follows:

$$u_i = \frac{\sum_{j=1}^{i-1} ||S_j - S_{j+1}||^\alpha}{\sum_{j=1}^{N-1} ||S_j - S_{j+1}||^\alpha}$$

The α variable can be controlled in my program by adjusting the "Centripetal" counter.

3 Distance Functions

In the below, $C_{n,P}(u)$ is the point obtained by evaluating the B-spline with degree n and control points P_i at u.

3.1 Least Squares

The Least Squares distance uses the sum of the squared Euclidean distance (ie. the 2-norm) between the sampled points and their corresponding parameterized point on the curve, ie:

$$\text{dist}_n(P_1, \cdots, P_p) = \sum_{i=1}^{N} ||S_i - C_{n,P}(u_i)||_2^2$$

The optimizer for this distance function solves the problem exactly by solving a system of linear equations.

3.2 Manhattan

The Manhattan option instead uses the Manhattan distance, also known as the 1-norm.

$$\text{dist}_n(P_1, \cdots, P_p) = \sum_{i=1}^{N} ||S_i - C_{n,P}(u_i)||_1$$

The optimizer for this option uses gradient descent, and it tends to converge very quickly at the level of precision which is used (regular C++ `doubles`). However, it is still only an approximation; there is a compile-time constant which sets a max number of gradient descent iterations.

3.3 LogSumExp

The LogSumExp option uses a common convex approximation of the max function known as LogSumExp.

$$\text{dist}_n(P_1, \cdots, P_p) = \log\left(\sum_{i=1}^{N} \exp(||S_i - C_{n,P}(u_i)||^2)\right)$$

The optimizer for this option uses gradient descent as well, however it is much slower than the Manhattan optimizer. This is because it uses the MPFR library to work with high-precision floating point values with very high exponents, which is a lot less efficient than native `doubles`. This distance function also usually takes a lot longer to converge with gradient descent. Again, there is a cutoff on the number of iterations performed, so this is only an approximation.