UM EECS 487
Lab 8: Splines

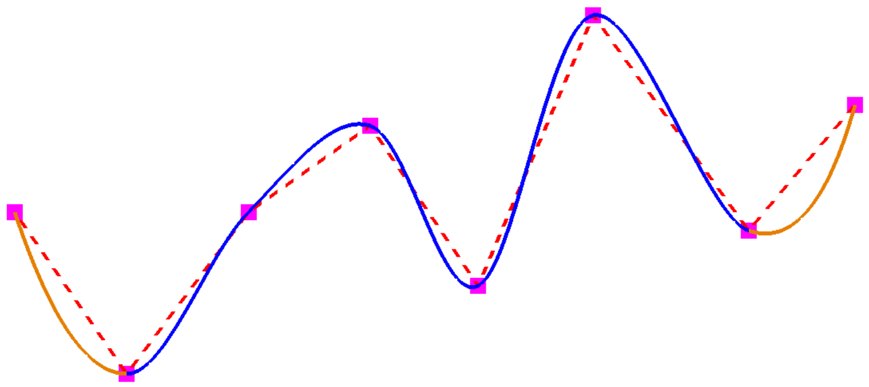

FIGURE 1. Screenshot showing one particular arrangment of points and [red] linear interpolation, [blue] Catmull-Rom spline interpolation, and [orange] quadratic interpolation.

In this lab we experiment with interpolating curves through a given set of points. The provided program, `splines` displays a set of 8 points in a row which you can move around by dragging them with the mouse. Each pair of points is displayed with a linearly interpolated dashed line connecting them. Pressing '`1`' toggles the display of this line on and off. Pressing '`2`' toggles the display of a curve connecting the points interpolated using the Catmull-Rom spline. Pressing '`3`' "extends" the Catmull-Rom spline to also interpolates the two endpoints.

1. CATMULL-ROM SPLINE

- Ignore compiler warnings about some unused variables. You will use them.
- All the points are stored in the global array `XVec2f points[NUM_POINTS]`.
- Inspect `draw_linear()` in `draw_splines.cpp`. We simplify a bit here. Instead of linearly interpolating the line between two control points manually, we call OpenGL and let the graphics engine's rasterizer do it for us.
- **Task 1**: Implement `draw_catmull_rom()` in `draw_splines.cpp`. This should interpolate all points except the first and last. For each pair of successive points, calculate the coefficients of the spline by calling `cspline_coefs()`, which you're asked to implement, and then approximate the section between them with a number of short line-segments, with the help of `cspline_eval()`, which you're also to implement. You are free to choose the number of samples for the parameter `u` between the points to create these line segments.
- **Task 2:** By definition, the Catmull-Rom spline is a cubic spline and it does not interpolate the two end points. Implement `draw_quadratic_endpieces()` in `draw_splines.cpp` to draw a quadratic spline that interpolates the first two endpoints, and then draw another quadratic spline that interpolates the last two endpoints. The two quadratic splines you draw must have the same tangent as the Catmull-Rom spline where they meet it, i.e., at the second control point and at the penultimate control point. You need to first define the quadratic spline that satisfies this requirement, one for each endpiece. Next construct the

constraint matrices and compute the basis matrices. Finally, you can draw the two end-pieces. Once you've constructed the constraint matrices, the rest of this task is a simple adaptation of the previous task.

2. Bézier Curves

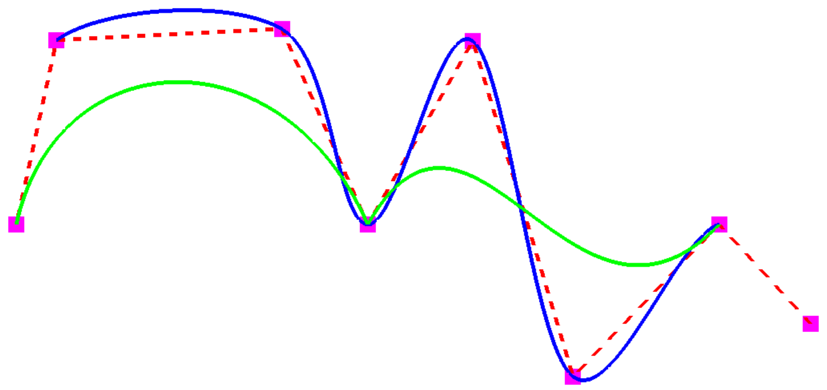

Figure 2. Screenshot showing one particular arrangment of points and [red] linear interpolation, [blue] Catmull-Rom spline interpolation, and [green] Bézier interpolation.

- **Task 3:** Implement `draw_cubic_bezier()` in `drawsplines.cpp` to approximate the curve between four successive points. Calculate the coefficients of the spline for point-sets $\{0, 1, 2, 3\}$ and $\{3, 4, 5, 6\}$. Choose the number of samples for the curve parameter u, and approximate the section with a number of short line-segments. In the provided program, pressing '`4`' toggles the display of a curve connecting the points interpolated using the Bézier spline.
- Move the points around and examine the advantages and limitations of each spline with respect to continuity (C^0 continuity), smoothness (C^1 continuity or differentiability), interpolating vs. approximating behavior, and computational effort.