Introduction to Computer Graphics
Assignment 11 – Animating the Camera Path

Handout date: 17.05.2018

Submission deadline: 25.05.2018, 13:00 h

Late submissions are not accepted

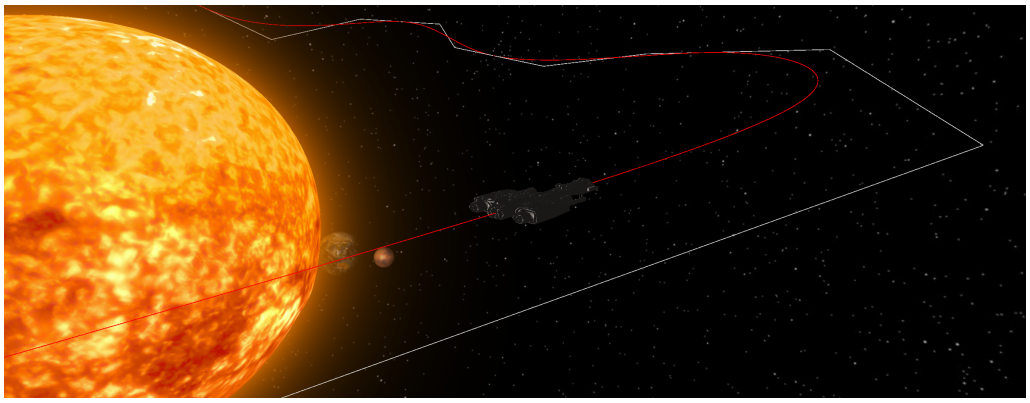

Figure 1: Part of the path given as a piecewise Bézier curve (red), the Bézier control polygons for each curve segment (white), and our spaceship following the path.

In this assignment, you will animate the spaceship so that it continuously flies through the Solar system. To achieve that, we will first prepare a path for the spaceship to follow. In order to make the motion look natural, we will model the path using piecewise Bézier curve with C2 continuity (see Figure 1). In addition to placing the spaceship on the path, you must orient it properly, which involves evaluating the path's tangent at a each time step. Finally, you will "attach" the eye to the spaceship in order to achieve a first-person-shooter-like feel.

Before starting the exercise, please read this assignment's `README.md` file, which explains how to invoke the debugging visualizations. Also, we made several additions to the solar system code base this week, so please work with the new version (copying your solution code from the previous exercises as indicated by the `todo` comments).

Overview

This exercise consists of six subtasks, which we recommend you implement in the following order:

1. Constructing individual Bézier curve control points from a spline control polygon.

2. Evaluating a point on a piecewise Bézier curve.
3. Evaluating the tangent of a piecewise Bézier curve.
4. Placing and orienting the spaceship on the path.
5. Making the spaceship follow the path with constant speed.
6. Attaching the eye to the spaceship.

Extract control points from a control polygon

The path is modeled as a piecewise cubic Bézier curve with C2 continuity (i.e., a cubic spline). We could define the path with a sequence of Bézier control points, four for each cubic segment of the curve. This would, however, be quite inconvenient: when specifying the sequence of control points, we would have to manually ensure control points are appropriately positioned to achieve C2 continuity.

We choose a more convenient approach based on the "A-frame" construction that was presented in class for enforcing C2 continuity of two cubic Bézier curves (please review Lecture 12, slide 20). In this approach, we specify the curve by a control polygon $\mathcal{P}$ that has exactly the right degrees of freedom to define a cubic C2 continuous curve. By deriving our Bézier control points from this control polygon, we are guaranteed C2 continuity by construction.

It turns out that the Bézier control points for the uniform cubic spline defined by $\mathcal{P}$ can be determined with a simple interpolation process. With this process, you will create a Bézier curve segment (by determining its control points) for each edge of $\mathcal{P}$ except the first and last. The second and third control points of the Bézier curve for control polygon edge $e_i \in \mathcal{P}$ lie on e_i, dividing it into thirds. The first control point is then the midpoint of the line connecting the second control point and the previous segment's third control point. Finally, the last control point of a given Bézier curve coincides with the first control point of the next one. See Figure 2 for an illustration. Note that the first and last Bézier segments must be handled specially.

Please fill in the function `PiecewiseBezier::control_polygon_to_bezier_points()` which takes $\mathcal{P}$ (a sequence of 3D points) as input and returns the sequence of control points of the consecutive cubic Bézier curves. After finishing the exercise, you should be able to see the sequence of these control points (visualized as the white polyline) after pressing `C` twice.

Evaluate points on a piecewise Bézier curve

Once you have a sequence of the piecewise Bézier curve control points, your next task is to evaluate a position on the full path given the parameter $t \in [0,1]$. Note, however, that varying t from 0 to 1 traces out the entire path (i.e. across all the cubic Bézier segments). So, to evaluate at a particular t, you must determine which segment contains t and then evaluate that Bézier curve. Because each individual Bézier curve segment is parametrized on $t_s \in [0,1]$, you must also transform t to the segment's corresponding t_s parameter. Please do this in `PiecewiseBezier::eval_piecewise_bezier_curve`.

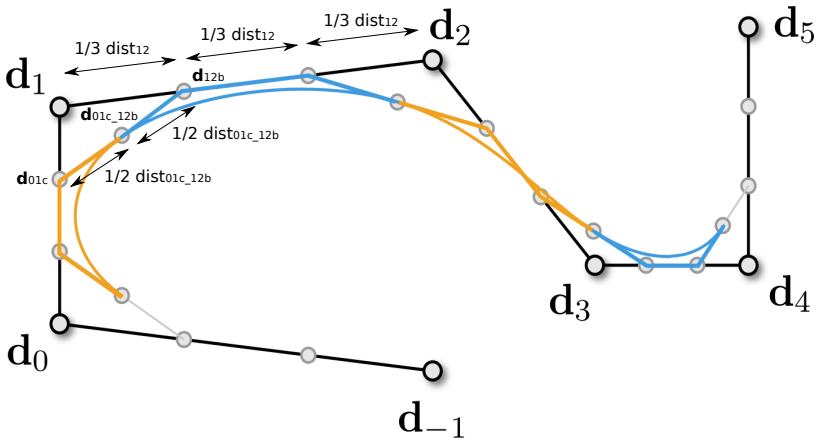

Figure 2: Control polygon of a piecewise cubic Bézier curve.

As a next step, fill in the function `PiecewiseBezier::eval_bezier` which, given an index into the array of consecutive Bézier control points (picking out a particular Bézier segment) and a parameter $t_s \in [0,1]$ evaluates the appropriate point on the Bézier segment. After finishing this task you should be able to visualize the path as a red curve (the curve we provide is a closed loop).

Evaluate the tangent of a piecewise Bézier curve

In order to point the spaceship forward along the path, we need to evaluate the curve's tangent vector (i.e., the instantaneous velocity of a point traveling along the path). There are two possible ways to evaluate the tangent at $\mathbf{c}(t)$, where $\mathbf{c}(t)$ is a cubic Bézier curve parameterized by scalar $t \in [0,1]$.

The first approach involves computing $\frac{d\mathbf{c}(t)}{dt}$, which is given as:

$$\frac{d\mathbf{c}(t)}{dt} = \sum_{i=0}^{n} \mathbf{b}_i \frac{dB_i^n(t)}{dt} = n \sum_{i=0}^{n-1} (\mathbf{b}_{i+1} - \mathbf{b}_i) B_i^{n-1}(t), \tag{1}$$

where $\mathbf{b}_i$ is the ith control point, and B_i^n is the ith Bernstein polynomial of degree n. Please note that $B_i^n = 0$ for $i \notin 0 \ldots n$. This approach amounts to determining a new set of Bézier control points for the quadratic curve $\mathbf{c}'(t)$, and then evaluating this curve.

The second approach takes advantage of the fact that the tangent at t coincides with the line connecting the last two points computed as a byproduct evaluating the curve at t with the de Casteljau algorithm (see Figure 3).

Similarly to the previous task, it is necessary to transform the parameter $t \in [0,1]$, which traces the whole path, to obtain $t_s \in [0,1]$, which traces just the appropriate Bézier segment, s. Note that this means you are evaluating the function $\text{path}(t) = b_s(t_s(t))$, where b_s is the Bezier curve for segment s; you will need to consider this when computing the derivative $\text{path}'(t)$ (hint: chain rule).

Please fill in the function `PiecewiseBezier::eval_bezier_tangent` and implement the computation of the tangent given the parametric distance along the segment t and an

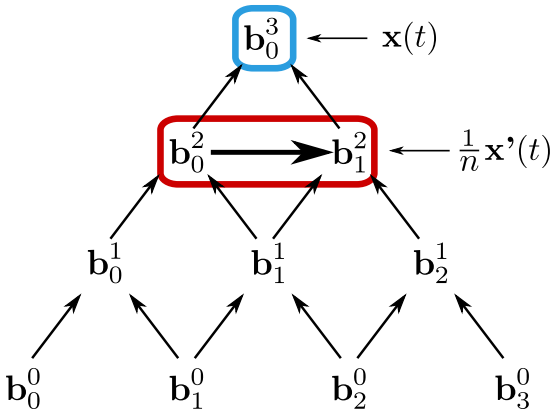

Figure 3: Computing derivative of Bézier curve using de Casteljau algorithm.

offset to the array of control points (analogously to the previous step). For your convenience, we provide you with a visualization of the coordinate frame represented by colored X, Y, Z axis which sits on the path and whose Z-axis coincides with the tangent you compute (press `C` three times).

Place the ship on the right track

Let us finally place and orient the spaceship on the path. To this end, you will have to construct its model matrix by combining the position given as the 3D point on the path (at time t) and the orientation given by the tangent. Of course, we need more than just a tangent vector to define the ship's orientation; we also need to specify the rotation of the ship around the tangent. This part is provided for you by `ship_path_frame_`, which chooses an `up` vector using one of two strategies (chosen by the `T` key). Please complete the appropriate section of `SolarViewer::draw()` (see the `todo` comment); you may find it helpful to use the method `Frame:::xyzToFrame()`.

At this point, your spaceship should sit on the path and should point in the direction of the tangent at that point. Since the ship appears small, please use the debugging visualization of the colored frame, which will help you to spot the ship.

Advance the simulation time with constant speed of the ship

We want the spaceship to follow the path with a constant speed, i.e. given the fixed constant time step dT, we would like the ship to advance by constant Euclidean distance dS. This, however, is generally *not* achieved by incrementing t at a constant rate; unless the control points happen to be spaced perfectly so that our curve has a unit speed parametrization, the ship will accelerate/decelerate as it traverses the curve. Hint: You can read the beginning of the Chapter 14 - Getting 3D normals (A Primer on Bézier curves), which gives the intuition about how to relate the speed to the curve.

Please fill in the `todo` section of the function `Solar_viewer::timer()` and update the parameter t (which corresponds to the variable `ship_path_param_` in the code), such that the velocity of the spaceship is constant. If everything goes well, you should be able to see the spaceship flying along the path around the Sun.

Make the eye follow the ship

The final step is to make the eye follow the spaceship. As in the previous Solar System exercises, we want the eye to overlook the spaceship from slightly above. Furthermore, the eye coordinate system should be defined relative to the spaceship's orientation so that the spaceship appears fixed with respect to the eye (the screen). Figure 4 shows a sample screenshot captured as the spaceship has made it partway around the path. Please note that, as in the previous exercises, you need to press key 7 in order to focus the eye on the spaceship.

Please fill in the appropriate `todo` section in the function `SolarViewer::paint()` and compute the view transformation which makes the eye follow the spaceship, if the boolean flag `in_ship_` is `true`.

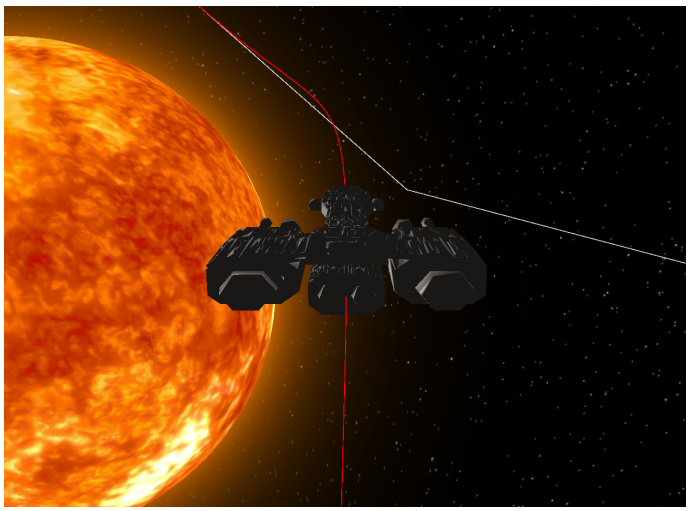

Figure 4: When $y\_angle\_ = 0$, the camera should be positioned directly behind the ship, and the ship should appear stationary.

Grading

The scores for this assignment are broken down as follows:

- 20%: Extracting control points
- 20%: Evaluating points on a piecewise Bézier curve
- 20%: Evaluating a tangent on a piecewise Bézier curve
- 15%: Placing the ship on the path
- 15%: Constant space velocity
- 10%: Making the eye follow the spaceship

What to hand in

A compressed .zip file renamed to `Exercise11-Group`*i*`.zip`, where *i* is your group number. It should contain:

- **Only** the source files you changed (in this case `bezier.cpp` and `solar_viewer.cpp`).
- A couple screenshots showing that you successfully computed the path (please use the path visualization) and oriented the eye to follow the spaceship.
- A `readme.txt` file containing a description of how you solved each part of the exercise (use the same numbers and titles) and whatever problems you encountered. Indicate what fraction of the total workload each project member contributed.