TU BRAUNSCHWEIG
Dr.-Ing. Martin Eisemann
Institut für Computergraphik
Lorenz Rogge (rogge@cg.tu-bs.de)

July 2, 2012

Echtzeit Computergraphik SS 2012
Assignment 11

Present your solution to this exercise on Monday, July 9th, 2012.

The exercises are going to take place in the CIP pool, room G40 in Mühlenpfordstraße 23. Please make sure your solutions compile and run on the CIP pool computers. Note that you need a y-number, which can be obtained at the Gauß-IT-Zentrum, to use the computers. If for some reason you are not able to attend the exercise, you may send your solution to rogge@cg.tu-bs.de instead.

Splines are a very comfortable way to describe and interpolate positions on a specific curve in space. In this exercise you will implement Catmull-Rom splines to compute positions on a curve used for animating objects. Given a time value, you are able to interpolate a position corresponding to this time value on a defined curve.

11.1 Interpolation with Catmull-Rom Splines (40 Points)

Splines allow to smoothly interpolate positions in space that lie on a curved path. This path can be described by only few control points, which define, dependent on the type of spline used, how the curve lies in space.
In this task complete the missing parts of the `Path` class used in this framework. Implement the technique of Catmull-Rom Splines that allows to interpolate a curve through a set of predefined control points. A set of points creating a circular path is already given in `Ex11.cpp`. To properly interpolate a position for a given interpolation step - `Path` uses time for interpolation - complete the empty method `Path::getPositionForTime`. Using the matrix notation presented in the lecture, you should be able to compute the correct position for a given time t. Do as follows:

- Select the correct path segment for the given time t. Note that the very first and the last control point do not belong to actual path segments but help to define the tangential directions of the segments adjacent to them. Every path segment consists of *four* control points $P_i, P_{i+1}, P_{i+2}, P_{i+3}$, while the actual interpolated path connects only the control points P_{i+1} and P_{i+2}.
- Compute your interpolated point for a given segment using the matrix multiplication presented in the lecture:

$$P(t) = \frac{1}{2} \cdot \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} \cdot \begin{pmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} P_i \\ P_{i+1} \\ P_{i+2} \\ P_{i+3} \end{pmatrix} \tag{1}$$

The already given framework uses `ControlPoint` to represent a control point with a specific position in space `pos` and a time value `time` describing the position of this control point along the path. Since the first and last control points are not relevant for the actual interpolated path, a time value of -1 might help to identify them. The path using N control points $(P_0 \dots P_{N-1})$ exists and allows interpolation from time $t_0 = P_1.time$ till $t_{N-3} = P_{N-2}.time$. A member of `Path` then allows to loop the path, i.e. time values exeeding the maximum interpolation time t_{N-3} should result in a interpolated position at beginning of the path. Vice versa when using time values less than t_0.

The implemented method finally should return a `ControlPoint` which interpolated position can be used in the animation of objects.
The framework provides a class `Timer`, that allows to read continuous time values, when executing the rendering loop. When initializing the render loop, simply call `Timer::start` and while rendering the current time may be acquired using `Timer::getTime`. This time value may be used to get an interpolated position of a spline path.
Having these tools, you are now able to render an animated solar system. The framework already defines three `MeshObj` objects:

- The sun using a emissive texture only.
- The planet mars using a diffuse texture and a normal map.
- A moon object again using a diffuse texture and a normal map.

Create a scene, where the mars rotates around the sun placed at $(0, 0, 0)$ at a certain speed and let the moon rotate around the mars at *another* speed. Using the timer `mTimer` you can pick adequate time values for the current rendering time step.

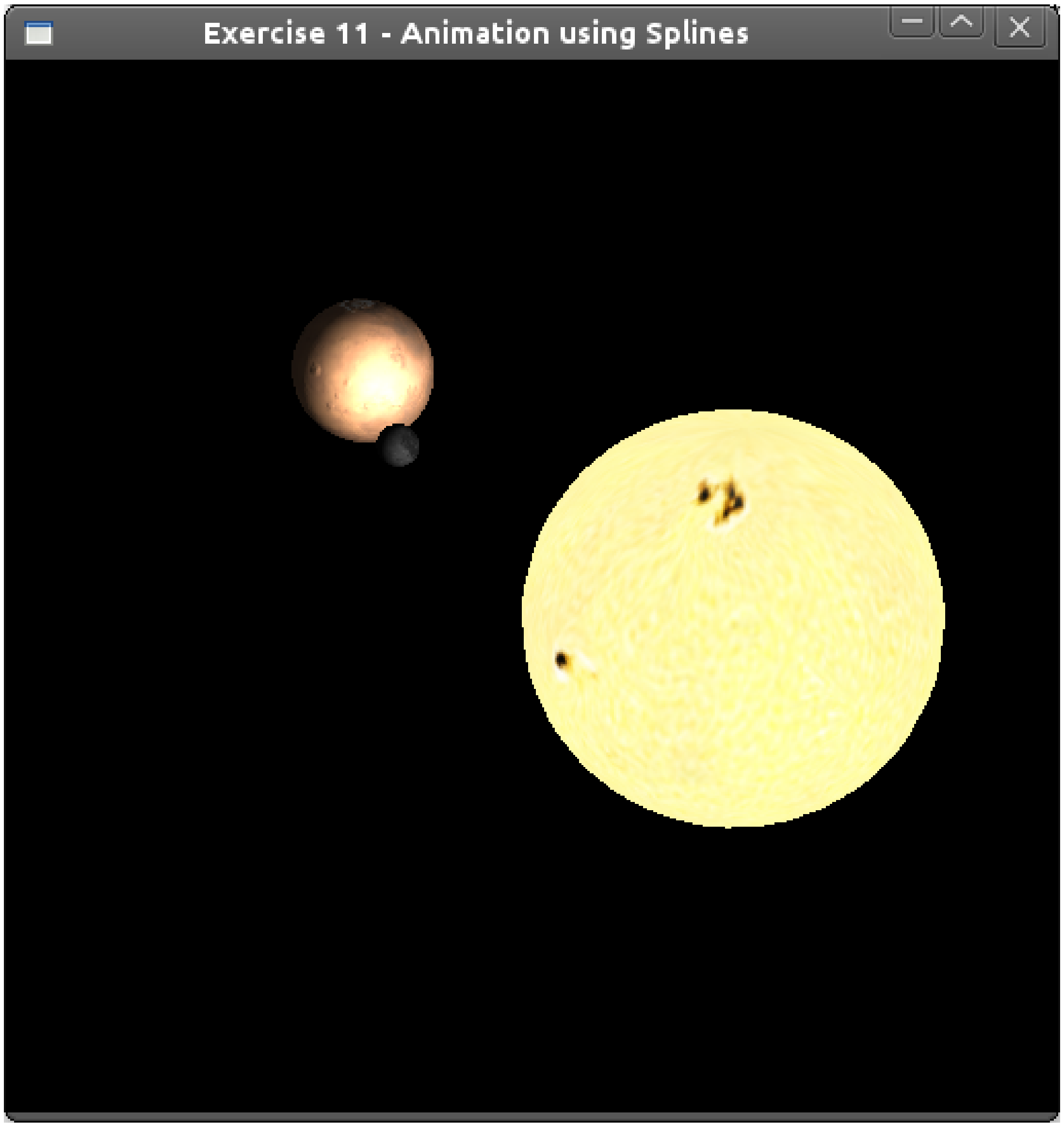

Figure 1: Animated, rotating planets using spline interpolation