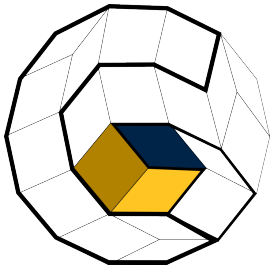
TU BRAUNSCHWEIG
Dipl. Inform. Martin Eisemann
Institut für Computergraphik
Lorenz Rogge (rogge@cg.tu-bs.de)



July 5, 2011

Echtzeit Computergraphik SS 2011
Assignment 10

In this exercise we use Splines to interpolate positions in space used for continuous animation of objects. In this case, we want to simulate planets rotating around a sun. In addition to the last exercise, a special shader has to be implemented as well, allowing to create a blooming effect of the sun and around objects in front of it. See figure for an example 1. Please present your solution to this exercise on Monday, July 11th, 2011.

10.1 Interpolation with Catmull-Rom Splines (20 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 `Ex10.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 looo, 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 (just as in the last exercises) and an ambient color term.
- A moon object again using a diffuse texture and a normal map and an ambient color term.

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. Note: It is not neccessary to create more than the already given path `mPath`. Using the timer `mTimer` you can pick adequate time values for the current rendering time step.

10.2 Bloom rendering of glowing objects (20 Points)

Similar to the last exercise we want to make use of FrameBufferObjects to create a blurring effect. This time, the framework already defines the objects to render an animate, which are the sun, the planet mars and a moon, that should rotate around it.
Since the sun is a completely emissive object which is natually very bright, the objects directly in front and especially their boundaries might be superimposed by the brightness of the sun.
So, in this task, implement a technique creating this *blooming* effect. This can be done with the following steps:

- Render the solar system into a FBO, while rendering only the depth values of the sun, but everything else. This can be done by using `glColorMask` when rendering the sun. Render this data into a texture attached to your FBO.
- Keep the current depth buffer and render the sun again, but this time fully colored to another texture. As you will see, all parts the are occluded by other object will not be rendered. Since you rendered the sun a second time while keeping the old depth buffer. It might not be rendered, since the depth values say your second rendering is fully *occluded*. To prevent this, you can change the depth-test behaviour using `glDepthFunc`.
- Now that we have at texture of our planets and one of our sun, we can start applying our blooming effect. Implement a shader that works like a blur filter (similar to the last exercise) and apply it only to the sun's texture.
- As a final step, combine the original rendering of the planets with the blurred sun rendering by using a simple shader adding given input textures to the final output.

Note: You don't need an explicit texture of the depth values in this exercise. Thus you may simple use a `Renderbuffer` instead of a texture for your depthbuffer.
The final outcome should look like figure 1.

Figure 1: Object boundaries in front of the glowing sun are blurred by the blooming effect. Note that the planets themselves are not blurred in ANY way. All part not overlapping the sun appear perfectly sharp.