Graphics P1

150008859

10/03/2019

1 Introduction

For this practical, I implemented the following elements.

1. A tool for visualising Bezier Curves in 2d.
2. A tool for visualising 3d Bezier Curves and Bezier Surfaces.

The tools are written in Java without the use of any additional libraries.

See "**usage.pdf**".

Run the 2d tool or 3d tool by entering the practical directory and running:

```
java -cp bin draw.BezierDraw2d
java -cp bin draw.BezierDraw3d
```

2 Bezier Curves in 2d

2.1 Specifying Control Points

The user specifies the control points by clicking, the curve is generated incrementally. The curve is sampled by varying t, not by arc length.

The granularity of t is fixed to give a balance between the illusion of a continuous line and the performance of the application.

2.2 Sample Points

However, the user can vary the number of sample points during the generation of the tangents and surface normals. These sample points are visualised as circles.

2.3 Tangents and Normals

The first derivative of curve is used to generate tangents.

In 2d space, given a the direction of a tangent (x, y), the two normals are $(y, -x)$ and $(-y, x)$.

However, the practical specification requires curvature normals.

2.4 Curvature Normals

While performing some experimentation with a colleague, we examined the 2nd derivative of the Bezier curve.

As the Bezier curve is not parameterised by arc length, the 2nd derivative does not generate curvature normals.

However, purely by experimentation we noticed that the 2nd derivative appears to point towards the correct side of the curve.

We can use this information to select the correct normal.

By computing the dot product between the normal $(-y, x)$ and the 2nd derivative, the direction of the normal vector is flipped if the dot product is less than 0.

The code for this can be found in “draw.Graphics2dUtil” in the function “drawNormals”.

3 Bezier Curves and Surfaces in 3d

A new tool was developed to visualise objects in 3d. The tool is able to visualise Bezier Curves and Surfaces in 3d.

The ideas here are inspired by OpenGL.

3.1 3d Rendering

In order to render 3d vertices on a 2d screen, 3 matrices are required.

A model matrix M, a view matrix V to model the camera and a perspecive projection matrix P.

A homogeneous vertex p can be transformed to from local space to clip space by performing the multiplication $PVMp$ followed by the perspective divide in which each element of the vertex is divided by the 4th element.

3.2 Generating Curves and Surfaces

By supplying the parameters n for a Bezier Curve or the parameters n and m for a Bezier Surface, the tool generates control points in set pattern.

The curve and the surface is displayed as sample cloud of points.

The Bezier curve is displayed by varying a single parameter t. However, the surface requires sampling two parameters u and v.

3.3 Moving the Camera

The user can explore the visualisation by controlling the Camera. The Camera moves by manipulating the view matrix's position parameter. The Camera can "look around" by using two of Euler's angles Yaw and Pitch. Yaw is rotation around the y axis and pitch is rotation around the x axis.

This approach may suffer from the Gimbal lock.

3.4 Moving Control Points

The user can edit the control points by selecting them. The location of the projected vertex is easily found on the screen. The projected vertex is mapped back to it's original point. The tool displays a set of "mini axis" for that particular point indicating the directions of the x, y and z axis. The user uses the keyboard to move the control point in an intuitive manner.