CS 354 Project 2: Ray-Parametric Surface Intersection

Assignment

In this project, we are going to implement an algorithm for computing the intersection between a ray and a cubic Bezier patch. The ray is given by

$$L(t) = \boldsymbol{p}_0 + t\boldsymbol{d}, \quad t \geq 0 \tag{1}$$

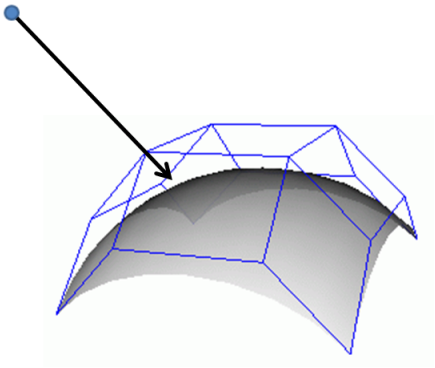
where $\boldsymbol{p}_0$ is the source location, and $\boldsymbol{d}$ is the direction of the ray. The cubic Bezier path is given by

$$S(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} \boldsymbol{c}_{ij} B_i(u) B_j(v), \tag{2}$$

where $\boldsymbol{c}_{ij}, 0 \leq i \leq 3, 0 \leq j \leq 3$ are the control points; $B_i(u), 0 \leq i \leq 3$ are basis functions given by

$$B_0(u) = (1-u)^3, \qquad B_1(u) = 3(1-u)^2 u, \qquad B_2(u) = 3(1-u)u^2, \qquad B_3(u) = u^3. \tag{3}$$

You are also required report the normal direction at the intersection.
Extra credit. Implement a phase shading model for the Bezier patch. You can use the programming framework of project 1 for visualization.



Submission

You need to submit an executable that preloads a Bezier patch, takes a ray (a source and a direction) as input, and outputs the intersection and the associated normal direction. The input and the output should be done in an iterative manner. If there is no intersection, the program shall terminate early.

You can use the following configuration for debugging, although it is recommended that you try as many configurations as you can:

$$\boldsymbol{p}_0 = \begin{pmatrix} 0 \\ 4 \\ 0 \end{pmatrix}, \qquad \boldsymbol{d} = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} \tag{4}$$

$$\boldsymbol{c}_{00} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{c}_{01} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \quad \boldsymbol{c}_{02} = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}, \quad \boldsymbol{c}_{03} = \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix},$$
$$\boldsymbol{c}_{10} = \begin{pmatrix} 1 \\ 0.5 \\ 0 \end{pmatrix}, \quad \boldsymbol{c}_{11} = \begin{pmatrix} 1 \\ 0.25 \\ 1 \end{pmatrix}, \quad \boldsymbol{c}_{12} = \begin{pmatrix} 1 \\ 0.5 \\ 2 \end{pmatrix}, \quad \boldsymbol{c}_{13} = \begin{pmatrix} 1 \\ 0 \\ 3 \end{pmatrix},$$
$$\boldsymbol{c}_{20} = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{c}_{21} = \begin{pmatrix} 2 \\ 0.5 \\ 1 \end{pmatrix}, \quad \boldsymbol{c}_{22} = \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix}, \quad \boldsymbol{c}_{23} = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix},$$
$$\boldsymbol{c}_{30} = \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{c}_{31} = \begin{pmatrix} 3 \\ 0.5 \\ 1 \end{pmatrix}, \quad \boldsymbol{c}_{32} = \begin{pmatrix} 3 \\ 0.5 \\ 2 \end{pmatrix}, \quad \boldsymbol{c}_{33} = \begin{pmatrix} 3 \\ 0.5 \\ 3 \end{pmatrix},$$

Grading

- Outputs at least one intersection (50 pts);
- Outputs all intersections (including no intersection) (30 pts);
- Normal computation (20 pts);

Tips

- We do not provide starter code. We recommend you to use the programming framework of Project I.
- It is recommend that you write your own C++ class for a Cubic Bezier patch. The class should have similar functionality as the triangular mesh class.
- To check the correctness of the output, you can convert the cubic Bezier patch into a triangular mesh, and apply ray-mesh intersection to obtain an approximate intersection. As the resolution of the mesh increases, the resulting intersection becomes more accurate.
- For extra credit, you can also use the triangular mesh to check the rendering effects.
- It is important for you to carefully think about how to implement your algorithm before doing the implementation. You should follow the general procedure of obtaining an initial guess and then refining it using root-finding strategies. However, each component can be implemented in many different ways...