# CS171 Final Project: Ray Tracing NURBS surfaces

GROUP NUMBER: 10
MEMBER 1: XIYUE PENG
MEMBER 2: LUOJIA HU

## 1 INTRODUCTION

In this project, we implemented an algorithm for directly ray tracing NURBS surfaces without generating mesh first and ray tracing the mesh.

## 2 IMPLEMENTATION DETAILS

In this section, we will discuss the NURBS surface representation and the 'direct' ray tracing NURBS algorithm we implemented.

### 2.1 NURBS Surface Representation

A **Non-Uniform Rational B-spline Surface**, or **NURBS**, is a generalization of B-spline surfaces. It is defined by a set of control points, a set of weights, and a set of knot vectors. The control points are the vertices of the surface, and the weights are the weights of the vertices. The knot vectors are the knot vectors of the surface. The surface is defined by the following equation:

$$\mathbf{S}^w(u,v) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} \mathbf{P}_{i,j}^w w_{i,j} B_{i,p}(u) B_{j,q}(v) P_{i,j} \tag{1}$$

where $B_{i,p}(u)$ is the $i$th B-spline basis function of degree $p$ at $u$, $P_{i,j}$ is the $i$th control point of the $j$th row, and $w_{i,j}$ is the weight of the $i$th control point of the $j$th row. The degree of the surface is $p$ in the $u$ direction and $q$ in the $v$ direction. The surface is defined on the domain $[u_0, u_{n+p+1}] \times [v_0, v_{m+q+1}]$, where $u_0 = v_0 = 0$ and $u_{n+p+1} = v_{m+q+1} = 1$. The knot vectors are defined as follows:

$$u_i = \begin{cases} 0 & \text{if } i \le p \\ \frac{i-p}{n+1} & \text{if } p < i \le n \\ 1 & \text{if } i > n \end{cases} \tag{2}$$

$$v_j = \begin{cases} 0 & \text{if } j \le q \\ \frac{j-q}{m+1} & \text{if } q < j \le m \\ 1 & \text{if } j > m \end{cases} \tag{3}$$

where $n$ and $m$ are the number of control points in the $u$ and $v$ directions, respectively. The knot vectors are non-decreasing, and the knot vectors are clamped at the ends. The knot vectors are also called **clamped knot vectors**.

### 2.2 Creating BVH for NURBS

Similar to what we did in ray tracing a Triangle Mesh, we need to create a BVH for the NURBS surface. The difference is that we need to subdivide the original NURBS surface's control points into a **refined control mesh**, instead of calculating triangle mesh coordinates directly.

Flattening TODO:

BVH TODO:

In order for an accurate initial guess during intersection test, we need to refine, or subdivide the control mesh into a finer one. We use a curvature-based refinement of knot vectors. The number of knots going to be added is based on the curvature of the surface (?).

$$n_1 = C_1 \cdot \max_{t_i, t_{i+1}} \{curvatre\}$$

### 2.3 Ray Tracing NURBS Directly

*2.3.1 Transform ray equation.*

Recall that a ray can be represented by $\mathbf{r} = \mathbf{o} + t\mathbf{d}$, where $\mathbf{o}$ is the ray's origin and $\mathbf{d}$ is the normalized direction. However in order for ray intersection test for NURBS, we will rewrite the ray $\mathbf{r}$ as the intersection of two planes, namely $\{\mathbf{p} | \mathbf{P_1} \cdot (\mathbf{p}, 1) = 0\}$ and $\{\mathbf{p} | \mathbf{P_2} \cdot (\mathbf{p}, 1) = 0\}$ where

$$\mathbf{P}_1 = (\mathbf{N}_1, d_1), \mathbf{P}_2 = (\mathbf{N}_2, d_2)$$

The parameters, namely $\mathbf{N}_1$, $\mathbf{N}_2$, $d_1$, and $d_2$, are calculated by the original ray parameter $\mathbf{o}$ and $\mathbf{d}$, determined by the following equations:

The normal vector of the first plane is always perpendicular to the ray direction $\mathbf{d}$.

$$\mathbf{N}_1 = \text{normalize}\left(\begin{cases} (\mathbf{d}_y, -\mathbf{d}_x, 0) & \text{if } |\mathbf{d}_x| > |\mathbf{d}_y| \text{ and } |\mathbf{d}_x| > |\mathbf{d}_z| \\ (0, \mathbf{d}_x, -\mathbf{d}_y) & \text{otherwise} \end{cases}\right)$$

The normal vector of the second plane is

$$\mathbf{N}_2 = \text{normalize}\left(\mathbf{N}_1 \times \mathbf{d}\right)$$

The distance of the two planes from the origin is

$$d_1 = -\mathbf{N}_1 \cdot \mathbf{o}$$

$$d_2 = -\mathbf{N}_2 \cdot \mathbf{o}$$

As shown in **??**, the first plane is perpendicular to the ray direction $\mathbf{d}$, with a normal lying in $xy$-plane or $xz$-plane, depending on the largest magnitude of the components of the direction. The second plane is perpendicular to both the ray direction $\mathbf{d}$ and the first plane's normal vector $\mathbf{N}_1$.

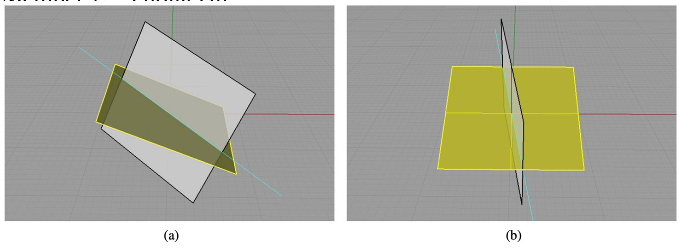

Fig. 1. The intersection-of-plane representation of a ray. If $|\mathbf{d}_x| > |\mathbf{d}_y|$ and $|\mathbf{d}_x| > |\mathbf{d}_z|$ holds, the first plane is rotated about the $z$-axis and the other plane is perpendicular to it (a). If it does not holds, the first plane is instead rotated about the $x$-axis(b)

.

In this way, an intersection point on the surface **S** can be found by solving the following system of equations:

$$\begin{cases} \mathbf{P}_1 \cdot (\mathbf{S}(u,v), 1) = 0 \\ \mathbf{P}_2 \cdot (\mathbf{S}(u,v), 1) = 0 \end{cases}$$

Note that we are solving for $u$ and $v$ values during intersection, instead of direcly getting the intersection point information. That's why we are ray tracing NURBS directly.

*2.3.2 Ray-Bezier-Patch Intersection Test.*
One of the most important part in ray tracing NURBS is to calculate intersection between a ray and a surface patch. We are going to solve this problem by *Newton-Rhapson method*.

We start from an initial guess of $u$ and $v$, which is the mid point of the valid range. We iterate the following until convergence:

$$\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} - \begin{pmatrix} \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} \\ \frac{\partial g}{\partial u} & \frac{\partial g}{\partial v} \end{pmatrix}^{-1} \begin{pmatrix} f(u_n, v_n) \\ g(u_n, v_n) \end{pmatrix}$$

The Jacobian matrix is defined as

$$\mathbf{J} = (\mathbf{F}_u, \mathbf{F}_v)$$

where

$$\mathbf{F}_u = \begin{pmatrix} \mathbf{N}_1 \cdot \mathbf{S}_u(u,v) \\ \mathbf{N}_2 \cdot \mathbf{S}_u(u,v) \end{pmatrix}$$

$$\mathbf{F}_v = \begin{pmatrix} \mathbf{N}_1 \cdot \mathbf{S}_v(u,v) \\ \mathbf{N}_2 \cdot \mathbf{S}_v(u,v) \end{pmatrix}$$

We use the following conditions to determine whether we should stop the Newton iteration.

- We limit the number of iterations to 7. If no solution can be found after the max number of iteractions, we stop and return no intersection. According to [], the average number of iterations needed to produce convergence is 2 or 3 in practice, so 7 is enough.
- If the distance to the root is nearly zero (less than a predetermined $\epsilon$ in practice), we conclude that an intersection happens. We then evaluate the intersection information (including position, normal) by the value of $u$ and $v$.

  $$\|\mathbf{F}(u_n, v_n)\| < \epsilon$$

  Since the value of $u, v$ we get is approximate, it may not necessarily lie along the ray. Therefore we perform a projection along the ray direction to get the intersection point.

  $$t = (\mathbf{P} - \mathbf{o}) \cdot \mathbf{d}$$

- We don't allow the new estimate further from the root compared with the last estimate. That is

  $$\|\mathbf{F}(u_{n+1}, v_{n+1})\| > \|\mathbf{F}(u_n, v_n)\|$$

  If this happens, we assume divergence, so we stop and return no intersection.
- We don't allow $u$ and $v$ to exceed the parametric domain of the surface.

  $$u \notin [u_{min}, u_{max}] \text{ or } v \notin [v_{min}, v_{max}]$$

  If after some iterations, the new estimate is out of the domain, we stop and return no intersection.

After each iteration, we will also assure that the Jacobian matrix **J** is not singular before updating the new estimate. To determine its singularity, we test if

$$\det(\mathbf{J}) = 0$$

If the Jacobian is singular, we iterate by the following equation

$$\begin{pmatrix} u_{k+1} \\ v_{k+1} \end{pmatrix} = \begin{pmatrix} u_k \\ v_k \end{pmatrix} + 0.1 \cdot \begin{pmatrix} \text{drand48}() \cdot (u_0 - u_k) \\ \text{drand48}() \cdot (v_0 - v_k) \end{pmatrix}$$

where drand48() returns a random floating number between 0 and 1.

## 2.4 Evaluation

TODO:

## 3 RESULTS