Final Project: Ray Tracing Trimmed NURBS

NAME: ZEEN CHI, WENCHAO LI
STUDENT NUMBER: 2020533089
EMAIL: {CHIZE, LIWCH}@SHANGHAITECH.EDU.CN

1   INTRODUCTION

Computer Aided Design (CAD) modeling tools are widely used for the industrial design of models for prototyping and production. The standard surface representation within these systems are trimmed non-uniform rational B-Spline (NURBS) surfaces. Trimmed NURBS surfaces are capable of compactly describing almost any shape and additionally provide local and intuitive control. These patches are commonly visualized by rendering a triangle- based approximation (tessellation) on graphic processing units (GPU), since there is no specialized NURBS rendering hardware readily available. The process of generating a high-quality tessellation is time-consuming, especially for trimmed surfaces requiring fine sampling of the trim boundaries. Therefore, it is complicated to render a NURBS surface with both efficiency and accuracy.

In this final project, we implement a ray tracing system that can render an arbitrary NURBS surface without tessellation. The outline of our approach can be briefly described by Figure 1. We create a set of boxes that bound the underlying surface over a given parametric range. The ray is tested for intersection with these boxes, and for a particular box that is hit, a parametric value within the box is used to initiate root finding. The key issues are determining which boxes to use, how to efficiently manage computing intersections with them, how to do the root finding, and how to efficiently evaluate the geometry for a given parametric value. We use refinement to generate the bounding volume hierarchy, which results in a shallower tree depth than other subdivision-based methods. We also use an efficient refinement-based point evaluation method to speed root-finding. These choices turn out to be both reasonable to implement and efficient.

In section 2 we present the bulk of our method, in particular how to create a hierarchy of bounding boxes and how to perform root finding within a single box to compute an intersection with an untrimmed NURBS surface. In Section 3 we describe how to extend the method to trimmed NURBS surfaces.

As for the result, we render different NURBS surface models with different complexity. We found a duck model that consists of three NURBS patches and rendered it to justify the correctness and efficiency of the untrimmed NURBS rendering scheme. We also found some complex trimmed NURBS models, e.g. Eiffel Tower and a very intricate ring, to show the correctness of the trimmed NURBS rendering scheme.

Note: the formulae derivation described in this report are not derived by us. We list them only to clarify the correctness of the corresponding formulae. However, every part of
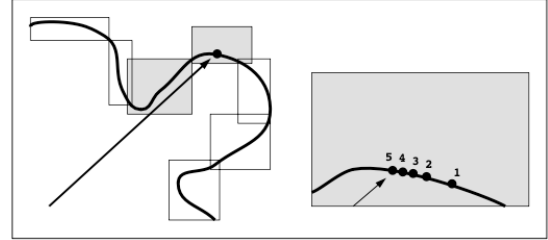
Fig. 1. The basic method we use to find the intersection of a ray and a parametric object shown in a 2D example. Left: The ray is tested against a series of axis-aligned bounding boxes. Right: For each box hit an initial guess is generated in the parametric interval the box bounds. Root finding is then iteratively applied until a convergence or a divergence criterion is met.

the algorithms described below is fully implemented by ourselves.

2   UNTRIMMED NURBS SURFACES RENDERING

2.1   Basic Formulations

A NURBS surface can be formulated by

$$S^w(u,v) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} P_{i,j}^w B_{j,k_u}(u) B_{i,k_v}(v)$$

where the superscript w denotes that our formulation produces a point in rational four space, which must be normalized by the homogeneous coordinate prior to display. The $\{P_{i,j}^w\}_{i=0,j=0}^{M-1,N-1}$ are the control points $(w_{i,j}x_{i,j}, w_{i,j}y_{i,j}, w_{i,j}z_{i,j}, w_{i,j})$ of the $M \times N$ mesh, having basis functions B_{j,k_u} and B_{i,k_v} of orders k_u and k_v respectively defined over knot vectors

$$\tau_u = \{u_j\}_{j=0}^{N-1+k_u}$$

$$\tau_v = \{v_j\}_{j=0}^{M-1+k_v}$$

and the basis functions are formulated by

$$B_{j,k_u}(u) = \begin{cases} 1 & \text{if } k_u = 1 \text{ and } u \in [u_j, u_{j+1}) \\ 0 & \text{if } k_u = 1 \text{ and } u \notin [u_j, u_{j+1}) \\ \frac{u-u_j}{u_{j+k_u-1}-u_j} B_{j,k_u-1}(u) + \\ \frac{u_{j+k_u}-u}{u_{j+k_u}-u_{j+1}} B_{j+1,k_u-1}(u) & \text{otherwise} \end{cases}$$

$$B_{i,k_v}(v) = \begin{cases} 1 & \text{if } k_v = 1 \text{ and } v \in [v_j, v_{j+1}) \\ 0 & \text{if } k_v = 1 \text{ and } v \notin [v_j, v_{j+1}) \\ \frac{v-v_i}{v_{i+k_v-1}-v_i} B_{i,k_v-1}(v) + \\ \frac{v_{i+k_v}-v}{v_{i+k_v}-v_{i+1}} B_{i+1,k_v-1}(u) & \text{otherwise} \end{cases}$$

student number: 2020533089
email: {chize, liwch}@shanghaitech.edu.cn

For a ray tracer, a ray is defined by:

$$\mathbf{r}(t) = \mathbf{o} + \hat{\mathbf{d}} \times t$$

but we need to rewrite it as the intersection of two orthogonal planes $\{\mathbf{p} \mid \mathbf{P}_1 \cdot (\mathbf{p}, 1) = 0\}$ and $\{\mathbf{p} \mid \mathbf{P}_2 \cdot (\mathbf{p}, 1) = 0\}$, where $\mathbf{P}_1 = (\mathbf{N}_1, d_1)$ and $\mathbf{P}_2 = (\mathbf{N}_2, d_2)$, and d_1 and d_2 are the distances from the plane to the world origin, respectively. The normal to the plane is defined as

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

and $\mathbf{N}_2$ is simply

$$\mathbf{N}_2 = \mathbf{N}_1 \times \hat{\mathbf{d}}$$

and d_1 and d_2 can be calculated by

$$d_1 = -\mathbf{N}_1 \cdot \mathbf{o}$$
$$d_2 = -\mathbf{N}_2 \cdot \mathbf{o}$$

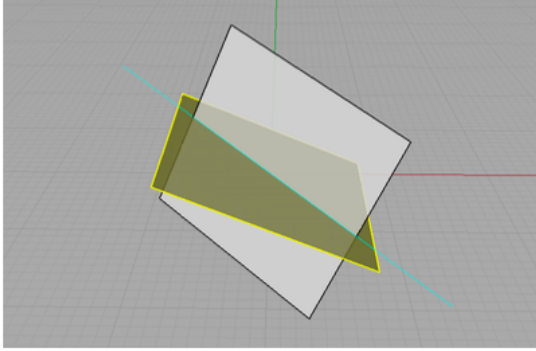The representation of the ray as two orthogonal planes is shown in Figure 2.

Fig. 2. The plane-intersection representation of a ray

2.2 Root Finder

Given a ray as the intersection of planes $\mathbf{P}_1 = (\mathbf{N}_1, d_1)$ and $\mathbf{P}_2 = (\mathbf{N}_2, d_2)$, we need to find the root (u_*, v_*) of

$$\mathbf{F}(u, v) = \begin{pmatrix} \mathbf{N}_1 \cdot \mathbf{S}(u, v) + d_1 \\ \mathbf{N}_2 \cdot \mathbf{S}(u, v) + d_2 \end{pmatrix}$$

We take Newton's method as the root finder for some reasons. First, it converges quadratically if the initial guess is close, which we ensure by constructing a bounding volume hierarchy. Furthermore, the surface derivatives exist and are calculated at cost comparable to that of surface evaluation. This means that there is likely little computational advantage to utilizing approximate derivative methods such as Broyden.

For each iteration of Newton's method, the update rule is

$$\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} - \mathbf{J}^{-1}(u_n, v_n) \cdot \mathbf{F}(u_n, v_n)$$

where $\mathbf{J}$ is the Jacobian matrix of $\mathbf{F}$, which is

$$\mathbf{J} = \begin{pmatrix} \mathbf{N}_1 \cdot \mathbf{S}_\mathbf{u}(u, v) & \mathbf{N}_1 \cdot \mathbf{S}_\mathbf{v}(u, v) \\ \mathbf{N}_2 \cdot \mathbf{S}_\mathbf{u}(u, v) & \mathbf{N}_2 \cdot \mathbf{S}_\mathbf{v}(u, v) \end{pmatrix}$$

There are four terminating criteria. For the success criterion, which is, if we are closer to the root than a threshold ϵ, i.e.,

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

then we report a hit. Otherwise, we should report a miss. We do not allow the new (u_*, v_*) estimate to take us further from the root than the previous one:

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

We also do not allow the iteration to take us outside the parametric domain of the surface:

$$u \notin [u_{k_u-1}, u_N), v \notin [v_{k_v-1}, v_M)$$

And we limit the number of iterations to $N_{\max} = MAXITER$, which is set as 7 by practice.

A final check is made to assure that the Jacobian matrix $\mathbf{J}$ is not singular. While this would seem to be a rare occurrence in theory, we have encountered this problem in practice. In the situation where $\mathbf{J}(u_k, v_k)$ is singular, either the surface is not regular ($\mathbf{S}_\mathbf{u} \times \mathbf{S}_\mathbf{v} = 0$) or the ray is parallel to a silhouette ray at the point $\mathbf{S}(u_k, v_k)$. In either situation, to determine singularity, we test

$$|\det(\mathbf{J})| < \epsilon$$

If the Jacobian matrix is singular, we perform a jittered perturbation of the parametric evaluation point, i.e.,

$$\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} + 0.1 \times \begin{pmatrix} \text{rand}(0, 1) \times (u_0 - u_k) \\ \text{rand}(0, 1) \times (v_0 - v_k) \end{pmatrix}$$

to initiate the next iteration, where (u_0, v_0) is the initial guess of the root.

2.3 NURBS Surface Subdivision

For Newton to converge both swiftly and reliably, the initial guess must be suitably close to the actual root. We employ a flattening procedure, i.e., refining/subdividing the control mesh so that each span meets some flatness criteria both to ensure that the initial guess is a good one, and for the purpose of generating a bounding volume hierarchy for ray culling.

We utilize the curvature-based refinement of the knot vectors, which will be described below.

Suppose we have a B-spline curve $c(t)$. An oracle for determining the extent to which the span $[t_i, t_{i+1}]$ should be refined is given by the product of its maximum curvature and its length over that span. Long curve segments should be divided in order to ensure that the initial guess for the numerical solver is reasonably close to the actual root. High curvature regions should be split to avoid multiple roots. As we are utilizing maximum curvature as a measure of flatness, our heuristic will be overly conservative for curves other than circles. The heuristic value for the number of knots to add to the current span is given by

$$n_1 = C_1 * \max_{[t_i, t_{i+1}]} \{\text{curvature}(\mathbf{c}(t))\} * \text{arclen}(\mathbf{c}(t))_{[t_i, t_{i+1}]}.$$

We also choose to bound the deviation of the curve from its linear approximation. This notion will imbue our heuristic with scale dependence. Thus, for example, large circles will be broken into more pieces than small circles. Erring again on the conservative side, suppose our curve span is a circle with radius r, which we are
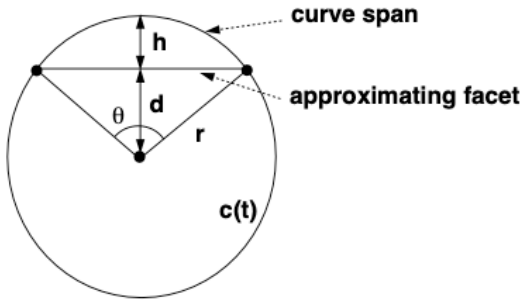
Fig. 3. Illustration of the chord height tolerance heuristic.

approximating with linear segments. A measure of the accuracy of the approximation can be phrased in terms of a chord height h which gives the maximum deviation of the facets from the circle. Observing Figure 3, it can be seen that

$$h = r - d$$

$$= r - r \cos \frac{\theta}{2}$$

$$\approx r \left(1 - \left(1 - \frac{\theta^2}{8}\right)\right)$$

$$= \frac{r\theta^2}{8}.$$

The number of segments n2 required to produce a curve within this tolerance is computed by $2\pi/\theta$. Thus, we have

$$n_2 = \frac{2\pi}{\sqrt{\frac{8h}{r}}}$$

$$= \frac{2\pi\sqrt{r}}{\sqrt{8h}}$$

$$\approx C_2 * \sqrt{\text{arclen}(\mathbf{c}(t))_{[t_i, t_{i+1}]}}$$

Combining the preceding oracles for curve behavior, our heuristic for the number of knots n to add to an interval will be $n_1 * n_2$:

$$n = C * \max_{[t_i, t_{i+1}]} \{\text{curvature}(\mathbf{c}(t))\} * \text{arclen}(\mathbf{c}(t))_{[t_i, t_{i+1}]}^{3/2}$$

where we set $C = 35$ in practice.

After a set of intricate approximations, the formula above can be rewritten as

$$n = C * \frac{\max_{[t_i, t_{i+1}]} \{|\mathbf{c}''(t)|\} * \left[\text{avg}_{[t_i, t_{i+1}]} \{|\mathbf{c}'(t)|\} * (t_{i+1} - t_i)\right]^{3/2}}{\text{avg}_{[t_i, t_{i+1}]} \{|\mathbf{c}'(t)|\}^2}$$

$$= C * \frac{\max_{[t_i, t_{i+1}]} \{|\mathbf{c}''(t)|\} * (t_{i+1} - t_i)^{3/2}}{\text{avg}_{[t_i, t_{i+1}]} \{|\mathbf{c}'(t)|\}^{1/2}}$$

$$= C * \frac{\max_{i-k+3 \le j \le i} \{|\mathbf{A_j}|\} (t_{i+1} - t_i)^{3/2}}{\left(\frac{1}{k-1} \sum_{j=i-k+2}^{i} |\mathbf{V_j}|\right)^{1/2}}.$$

where $\mathbf{V_j} = (k-1)\frac{(\mathbf{P_j} - \mathbf{P_{j-1}})}{t_{j+k-1} - t_j}$ and $\mathbf{A_j} = (k-2)\frac{(\mathbf{V_j} - \mathbf{V_{j-1}})}{t_{j+k-2} - t_j}$.

For each row of the mesh, we apply the above heuristic to calculate how many knots need to be added to each u knot interval, the final number being the maximum across all rows. This process is repeated for each column in order to refine the v knot vector. The inserted knots are spaced uniformly within the existing knot intervals.

As a final step in the flattening routine, we "close" all of the knot intervals in the refined knot vectors $\mathbf{t_u}$ and $\mathbf{t_v}$. By this, we mean that we give multiplicity $k_u - 1$ to each internal knot of $\mathbf{t_u}$ and multiplicity k_u to each end knot. Similarly, we give multiplicities of $k_v - 1$ and kv to the internal and external knots, respectively, of $\mathbf{t_v}$. The result is a Bezier surface patch corresponding to each non-empty interval $[u_i, u_{i+1}) \times [v_j, v_{j+1})$ of $\mathbf{t_u} \times \mathbf{t_v}$, which we can bound using the convex hull of the corresponding refined surface mesh points.

2.4 Bounding Volume Hierarchy (BVH)

We build a bounding volume hierarchy using the points of the refined control mesh we found in the previous section. The root and internal nodes of the tree will contain simple primitives which bound portions of the underlying surface. The leaves of the tree are special objects, which we call interval objects, are used to provide an initial guess (in our case, the midpoint of the bracketing parametric interval) to the Newton iteration. We will now examine the specifics in more detail.

The convex hull property of B-spline surfaces guarantees that the surface is contained in the convex hull of its control mesh. As a result, any convex objects which bound the mesh will bound the underlying surface. We can actually make a stronger claim. Since we closed the knot intervals in the last section [made the multiplicity of the internal knots $k-1$], each non-empty interval $[u_i, u_{i+1}) \times [v_j, v_{j+1})$ corresponds to a surface patch which is completely contained in the convex hull of its corresponding mesh points. Thus, if we produce an axis-aligned bounding box (AABB) for each of these intervals, we will have completely enclosed the surface. This is the basic idea behind the bounding volume hierarchy we use in our implementation.

2.5 KD-Tree

In spite of BVH, we also built a kd-tree as the acceleration structure. The kd-tree is a binary tree, where each node is a hyperplane that splits the space into two subspaces. The kd-tree is constructed by recursively partitioning the space into two subspaces. The splitting hyperplane is chosen following the Surface-Area Heuristics (SAH) function. The kd-tree is used to accelerate the intersection test between the ray and the AABBs. The leaf nodes of kd-tree contains multiple AABBs, which is already elaborated in the above subsection. The intersection test is performed by traversing the kd-tree. At each node, the ray is tested against the splitting hyperplane. If the ray intersects the splitting hyperplane, the ray is tested against the two subspaces recursively. Once the traversal comes to the leaf node, we perform intersection test with the ray and the corresponding AABBs, then return the closest intersection point.

student number: 2020533089
email: {chize, liwch}@shanghaitech.edu.cn

In section 3, we will compare the speed and accuracy of BVH and KD-Tree, to find out which acceleration structure is better for the NURBS surface intersection situation.

2.6 Refined Control Points Evaluation

We need to elaborate how to generate the refined control points D_i^ω. We again formulate our solution in the context of curves, and then generalize the result to surfaces.

Earlier we proposed to evaluate the curve c at t_* by stacking $k - 1$ t_*-valued knots in its knot vector τ to generate the refined knot vector t. The B-Spline basis transformation defined by this refinement yields a matrix A which can be used to calculate the refined control polygon D^ω from the original polygon P^W :

$$D^\omega = AP^W.$$

We are not interested in calculating the full alpha matrix A, but merely rows $\mu - k + 2$ and $\mu - k + 1$, as these are used to generate the points $D_{\mu-k+2}^\omega$ and $D_{\mu-k+1}^\omega$ which are required for point and derivative evaluation. Suppose $t_* \in [\tau_{\mu'}, \tau_{\mu'+1})$. We can generate the refinement for row $\mu + k - 1$ using a triangular scheme

$$
\begin{matrix}
 & & \alpha'_{\mu',0} \\
 & \alpha'_{\mu'-1,1} & \alpha'_{\mu',1} \\
 & \vdots & \vdots \\
\alpha'_{\mu'-\nu,\nu} & \cdots & \alpha'_{\mu',\nu}
\end{matrix}
$$

where ν is the number of knots we are inserting and

$$\alpha'_{j,1} = \delta_{j,\mu'}$$
$$\alpha'_{j,p+1} = \gamma_{j,p}\alpha'_{j,p} + \left(1 - \gamma_{j+1,p}\right)\alpha'_{j+1,p}$$
$$\gamma_{j,p} = \begin{cases} \left(t_* - \tau_{\mu'-p+j-(k-1-\nu)}\right)/d, & \text{if } d = \tau_{\mu'+1+j-} \\ & \quad \tau_{\mu'-p+j-(k-1-\nu)} > 0 \\ \text{arbitrary} & \text{otherwise.} \end{cases}$$

$A_{\mu-k+1,j} = \alpha'_{j,\nu}$ for $j = \mu' - \nu, \cdots, \mu'$ and $A_{i,j} = 0$ otherwise. If n knots exist in the original knot vector τ with value t_*, then $\nu = \max\{k - 1 - n, 1\}$ - that is to say, we always insert at least 1 knot. The quantity ν is used in the triangular scheme above to allow one to skip those basis functions which are trivially 0 or 1 due to repeated knots. As a result of this triangular scheme, we generate basis functions in place and avoid redundant computation of α' values for subsequent levels.

In the refinement scheme we propose, the point on the curve $D_{\mu-k+1}^\omega$ will be a convex blend of the points $D_{\mu-k}^\omega$ and $D_{\mu-k+2}^\omega$. The blend factor will be $\gamma_{\mu',0}$. The factor $\gamma_{\mu',0}$ is introduced at the first level of the recurrence. The leaf terms can be written as

$$\alpha'_{j,\nu} = \left(1 - \gamma_{\mu',0}\right) l_{j,\nu} + \gamma_{\mu',0} r_{j,\nu}$$

with $j = \mu' - \nu, \cdots, \mu'$. $\{l_{j,\nu}\}$ and $\{r_{j,\nu}\}$ are those terms dependent on $\alpha'_{\mu'-1,1}$ and $\alpha'_{\mu',1}$ respectively. They are the elements of the alpha matrix rows $\mu - k$ and $\mu - k + 2$ with $A_{\mu-k,j} = l_{j,\nu}$ and $A_{\mu-k+2,j} = r_{j,\nu}$ for $j = \mu' - \nu, \cdots, \mu'$. We can generate the $\{l_{j,\nu}\}$ by setting $\alpha'_{\mu'-1,1} = 1$ and $\alpha'_{\mu',1} = 0$ and likewise, generate $\{r_{j,\nu}\}$ by setting $\alpha'_{\mu'-1,1} = 0$

and $\alpha'_{\mu',1} = 1$. Thus, $A_{\mu-k,j}$ and $A_{\mu-k+2,j}$ can be generated in the course of generating $A_{\mu-k+1,j}$ at little additional expense.

The procedure above generalizes easily to surfaces, allowing us to generate the desired rows of the refinement matrices A_u and A_v. The refined mesh D^ω is derived from the existing mesh P^W by:

$$D^\omega = A_u P^W A_v^T$$

To produce the desired points we only need to evaluate

$$
\begin{pmatrix}
D_{\mu_u-k_u+1,\mu_v-k_v+1}^\omega & D_{\mu_u-k_u+1,\mu_v-k_v+2}^\omega \\
D_{\mu_u-k_u+2,\mu_v-k_v+1}^\omega & D_{\mu_u-k_u+2,\mu_v-k_v+2}^\omega
\end{pmatrix} =
$$
$$
\begin{pmatrix}
(A_u)_{\mu_u+k_u+1,[\mu'_u-\nu_u...\mu'_u]} \\
(A_u)_{\mu_u+k_u+2,[\mu'_u-\nu_u...\mu'_u]}
\end{pmatrix} P^W_{[\mu'_u-\nu_u...\mu'_u][\mu'_v-\nu_v...\mu'_v]}
$$
$$
\begin{pmatrix}
(A_v)_{\mu_v+k_v+1,[\mu'_v-\nu_v...\mu'_v]} \\
(A_v)_{\mu_v+k_v+2,[\mu'_v-\nu_v...\mu'_v]}
\end{pmatrix}^T
$$

After that, the entrices of matrix

$$
\begin{pmatrix}
D_{\mu_u-k_u+1,\mu_v-k_v+1}^\omega & D_{\mu_u-k_u+1,\mu_v-k_v+2}^\omega \\
D_{\mu_u-k_u+2,\mu_v-k_v+1}^\omega & D_{\mu_u-k_u+2,\mu_v-k_v+2}^\omega
\end{pmatrix}
$$

will be utilized by the NURBS surface derivatives and normal vectors evaluation described in the next section.

2.7 Derivatives and Normal Evaluation

The Newton iteration requires us to perform surface and derivative evaluation at a point (u, v) on the surface. In this section we examine how this can be accomplished efficiently. We begin by examining the problem in the context of B-spline curves. We then generalize the result to surfaces.

We evaluate a curve $c(t)$ by using refinement to stack $k - 1$ knots (where k is the order of the curve) at the desired parameter value t_*. The refined curve is defined over a new knot vector t with basis functions $N_{i,k}(t)$ and new control points $\omega_i D_i$. Recall the recurrence for the B-Spline basis functions:

$$
N_{i,k}(t) = \begin{cases}
1 & \text{if } k = 1 \text{ and } t \in [t_i, t_{i+1}) \\
0 & \text{if } k = 1 \text{ and } t \notin [t_i, t_{i+1}) \\
\frac{t-t_i}{t_{i+k-1}-t_i}N_{i,k-1}(t) + \\
\frac{t_{i+k}-t}{t_{i+k}-t_i}N_{i+1,k-1}(t) & \text{otherwise.}
\end{cases}
$$

Let $t_* \in [t_\mu, t_{\mu+1})$. As a result of refinement, $t_* = t_\mu = \ldots = t_{\mu-k+2}$. According to the definition of the basis functions, $N_{\mu,1}(t_*) = 1$. There are only two potentially non-zero basis functions of order $k = 2$, namely those dependent on $N_{\mu,1} : N_{\mu,2}$ and $N_{\mu-1,2}$. From the recurrence,

$$
N_{\mu,2}(t_*) = \frac{t_* - t_\mu}{t_{\mu+k-1} - t_\mu}N_{\mu,1}(t_*) + \frac{t_{i+k} - t_*}{t_{\mu+k} - t_{\mu+1}}N_{\mu+1,1}(t_*)
$$
$$
= 0 * 1 + \frac{t_{i+k} - t_*}{t_{\mu+k} - t_{\mu+1}} * 0
$$
$$
= 0
$$

and,

$$
N_{\mu-1,2}(t_*) = \frac{t_* - t_{\mu-1}}{t_{\mu+k-2} - t_{\mu-1}}N_{\mu-1,1}(t_*) + \frac{t_{\mu+k-1} - t_*}{t_{\mu+k-1} - t_\mu}N_{\mu,1}(t_*)
$$
$$
= 0 * 0 + 1 * 1
$$
$$
= 1.
$$

Likewise, the only non-zero order $k = 3$ terms will be those dependent on $N_{\mu-1,2} : N_{\mu-1,3}$ and $N_{\mu-2,3}$.

$$N_{\mu-1,3}\left(t_*\right) = \frac{t_* - t_{\mu-1}}{(\cdots)} * 1 + (\cdots) * 0$$
$$= 0$$
$$N_{\mu-2,3}\left(t_*\right) = (\cdots) * 0 + \frac{t_{\mu-2+k} - t_*}{t_{\mu-2+k} - t_{\mu-1}} * 1$$
$$= 1$$

The pattern that emerges is that $N_{\mu-k+1,k}\left(t_*\right) = 1$. A straightforward consequence of this result is

$$\mathbf{c}\left(t_*\right) = \frac{\sum_i N_{i,k}\left(t_*\right) \omega_i \mathbf{D_i}}{\sum_i N_{i,k}\left(t_*\right) \omega_i} = \frac{\omega_{\mu-k+1}\mathbf{D_{\mu-k+1}}}{\omega_{\mu-k+1}} = \mathbf{D_{\mu-k+1}}.$$

The point with index $\mu-k+1$ in the refined control polygon yields the point on the curve. A further analysis can be used to yield the derivative. Given a rational curve

$$\mathbf{c}(t) = \frac{\sum_i N_{i,k}(t)\omega_i \mathbf{D_i}}{\sum_i N_{i,k}(t)\omega_i} = \frac{\sum_i N_{i,k}(t)\mathbf{D_i^\omega}}{\sum_i N_{i,k}(t)\omega_i} = \frac{\mathbf{D}(t)}{\omega(t)},$$

where $\mathbf{D_i^\omega} = \omega_i \mathbf{D_i}$, the derivative is given by the quotient rule

$$\mathbf{c}'(t) = \frac{\omega(t)\left(\mathbf{D}^\omega\right)'(t) - \mathbf{D}^\omega(t)\omega'(t)}{\omega(t)^2}$$

By the preceding analysis $\mathbf{D}^\omega\left(t_*\right) = \mathbf{D}^\omega_{\mu-k+1}$. Likewise, $\omega\left(t_*\right) = \omega_{\mu-k+1}$. The derivative of the B-Spline basis function is given by

$$N'_{i,k}(t) = (k-1)\left[\frac{N_{i,k-1}(t)}{t_{i+k-1} - t_i} - \frac{N_{i+1,k-1}(t)}{t_{i+k} - t_{i+1}}\right].$$

Evaluating the derivative at t_*, we have

$$\left(\mathbf{D}^\omega\right)'\left(t_*\right) = \sum_i N'_{i,k}\left(t_*\right)\mathbf{D}_i^\omega$$
$$= (k-1)\sum_i \mathbf{D}_i^\omega\left(\frac{N_{i,k-1}\left(t_*\right)}{t_{i+k-1} - t_i} - \frac{N_{i+1,k-1}\left(t_*\right)}{t_{i+k} - t_{i+1}}\right)$$
$$= (k-1)\left[\sum_i \mathbf{D}_i^\omega \frac{N_{i,k-1}\left(t_*\right)}{t_{i+k-1} - t_i} - \sum_i \mathbf{D}_i^\omega \frac{N_{i+1,k-1}\left(t_*\right)}{t_{i+k} - t_{i+1}}\right].$$

We know the only non-zero basis function of order $k-1$ is $N_{\mu-k+2,k-1}\left(t_*\right) = 1$. Therefore,

$$\left(\mathbf{D}^\omega\right)'\left(t_*\right) = (k-1)\left[\frac{\mathbf{D}_{\mu-k+2}^\omega}{t_{\mu+1} - t_{\mu-k+2}} - \frac{\mathbf{D}_{\mu-k+1}^\omega}{t_{\mu+1} - t_{\mu-k+2}}\right]$$

Analogously,

$$\omega'\left(t_*\right) = (k-1)\left[\frac{\omega_{\mu-k+2}}{t_{\mu+1} - t_{\mu-k+2}} - \frac{\omega_{\mu-k+1}}{t_{\mu+1} - t_{\mu-k+2}}\right]$$

Plugging in for $\mathbf{c}'(t)$

$$\mathbf{c}'\left(t_*\right) = (k-1)\left[\frac{\frac{\mathbf{D}_{\mu-k+2}^\omega - \mathbf{D}_{\mu-k+1}^\omega}{t_{\mu+1} - t_{\mu-k+2}}\omega_{\mu-k+1} - \mathbf{D}_{\mu-k+1}^\omega \frac{\omega_{\mu-k+2} - \omega_{\mu-k+1}}{t_{\mu+1} - t_{\mu-k+2}}}{\omega_{\mu-k+1}^2}\right]$$
$$= \frac{k-1}{\left(t_{\mu+1} - t_{\mu-k+2}\right)\omega_{\mu-k+1}}\left[\mathbf{D}_{\mu-k+2}^\omega - \mathbf{D}_{\mu-k+1}^\omega \frac{\omega_{\mu-k+2}}{\omega_{\mu-k+1}}\right]$$
$$= \frac{(k-1)\omega_{\mu-k+2}}{\left(t_{\mu+1} - t_*\right)\omega_{\mu-k+1}}\left[\mathbf{D}_{\mu-k+2} - \mathbf{D}_{\mu-k+1}\right].$$

The result for surface evaluation follows directly from the curve derivation, due to the independence of the parameters in the tensor product, so we shall simply state the results:

$$\mathbf{S}\left(u_*, v_*\right) = \mathbf{D}_{\mu_u-k_u+1,\mu_v-k_v+1}$$
$$\mathbf{S_u}\left(u_*, v_*\right) = \frac{(k_u-1)\omega_{\mu_u-k_u+2,\mu_v-k_v+1}}{\left(u_{\mu_u+1} - u_*\right)\omega_{\mu_u-k_u+1,\mu_v-k_v+1}}\times$$
$$\left[\mathbf{D}_{\mu_u-k_u+2,\mu_v-k_v+1} - \mathbf{D}_{\mu_u-k_u+1,\mu_v-k_v+1}\right]$$
$$\mathbf{S_v}\left(u_*, v_*\right) = \frac{(k_v-1)\omega_{\mu_u-k_u+1,\mu_v-k_v+2}}{\left(v_{\mu_v+1} - v_*\right)\omega_{\mu_u-k_u+1,\mu_v-k_v+1}}\times$$
$$\left[\mathbf{D}_{\mu_u-k_u+1,\mu_v-k_v+2} - \mathbf{D}_{\mu_u-k_u+1,\mu_v-k_v+1}\right]$$

The normal $\mathbf{n}(u,v)$ is given by the cross product of the first order partials:

$$\mathbf{n}\left(u_*, v_*\right) = \mathbf{S_u}\left(u_*, v_*\right) \times \mathbf{S_v}\left(u_*, v_*\right)$$

After getting the normal and derivatives, we can easily render the entire NURBS surface under a path tracer together with global illumination.

Until now, we have fully elaborated all the algorithms needed. As long as combining all of them appropriately, we can successfully render the untrimmed NURBS surface using a ray tracer without tessellating the surface.

3 TRIMMED NURBS SURFACE RENDERING

4 RESULTS

For the untrimmed NURBS surface rendering, we successfully render a duck using different algorithms. The duck only consists of three NURBS surface patches. The scene is rendered with resolution 1080×1080 and samples-per-pixel (spp) 1024.

For reference, we firstly render the duck with tessellation. Then, we will show the duck rendered by different methods without tessellation.