Peak your Frequency: Advanced Search of 3D CAD Files in the Fourier Domain*

Technical Report

Dimitris Mouris† Charles Gouert†

Nektarios Georgios Tsoutsos‡

University of Delaware

Abstract

As an ever-increasing number of industries adopt additive manufacturing (AM), also known as *3D printing*, to their production lifecycles, the need for efficient retrieval and reuse of the existing 3D CAD models in order to facilitate new designs becomes more and more essential. However, traditional search techniques perform poorly in the context of searching CAD designs. In this paper, we propose Fourier Fingerprint Search (FFS), a retrieval framework for 3D models that deduces and leverages critical shape characteristics for search. FFS introduces a novel search methodology that incorporates these characteristics and uses two advanced matching techniques that operate at different granularities and take into account unique patterns associated with each design. In addition, FFS supports both exact and partial matching in order to provide helpful and robust search results for any scenario. We investigate a diverse set of features and enhancements for search that allows for high adaptability in all situations, such as dividing shapes into smaller parts, surface interpolation, and two different types of rotation. We evaluate Fourier Fingerprint Search using the FabWave CAD dataset with approximately 3000 manufacturing models with different configurations. Our experimental results demonstrate the efficiency and high accuracy of our approach for both exact and partial matching, rendering FFS a powerful framework for CAD model search.

*This report is based upon work supported by the National Science Foundation under Grant No. 1931916. Any opinions, findings, and conclusions or recommendations expressed in this report are those of the author(s) and do not necessarily reflect the views of the NSF.

†The first two authors have an equal contribution.

‡Corresponding author email: tsoutsos@udel.edu

1 Introduction

Additive manufacturing (AM), or 3D printing, has been adopted by a variety of industries including engineering, manufacturing, automotive, aerospace, and medical [1–3]. First conceptualized in 1984, stereolithography was the earliest 3D printing technique, utilizing a laser to solidify light-activated resin layer by layer until the 3D object is fully constructed [4]. Seven years later, the world's first stereolithographic fabrication machine was created [5]. However, it wasn't until the mid 2000s that AM technologies started to see widespread adoption with the reduced costs of fabrication machines and the subsequent plethora of open source 3D printing resources, such as development kits [6]. Indeed, in contemporary society, AM is considered the next industrial revolution [7] and the number of three-dimensional computer-aided design (CAD) models available online continues to grow exponentially.

As a result, people are routinely exposed to 3D printed materials in all facets of everyday life from medicine to automobiles. Modern medicine has seen widespread adoption of additive manufacturing products such as prosthetics, implants, and surgical tools. In addition, the automotive industry commonly employs AM techniques to create fixtures and molds. It is clear that 3D printing technologies will continue to play a large role in technological development and society as a whole in the near future. Evidently, it is imperative for these industries to have the ability to effectively and efficiently retrieve and reuse the existing CAD models in order to facilitate new designs [8].

Unfortunately, existing search techniques are ill-suited when applied to 3D CAD models. The traditional text-based information retrieval that search engines use has very limited functionality with CAD models as this restricts the search for keywords in filenames, captions, or context. This type of search only considers file metadata and not the contents of the files themselves, in this case the physical characteristics of the 3D models. For this reason, many works have explored more sophisticated techniques to search for 3D parts. Researchers have incorporated techniques that derive unique information from 3D objects such as spherical harmonics [9], or a combination of Fourier descriptors and Zernike moments that are derived from transforms over disks and spheres to represent properties of 3D objects [10]. Other works have proposed using clustering techniques to group similar shapes together [11] as well as geometric approximation techniques, such as representing a model as a set of cubes [12]. In an interesting direction that is orthogonal to search, other efforts related to AM have focused on protecting 3D design intellectual property against counterfeiting [13–16].

To enable design reuse, partial retrieval is essential [17]. Partial retrieval returns matches that are similar in some way to the queried model, such as incorporation of similar or identical components. As an illustration, consider an automotive corporation that has a database of 3D models of car rims and the corporation wants to find out which existing rims look similar to a new rim design. Exact matching will not return any results, as the new rim design is very different

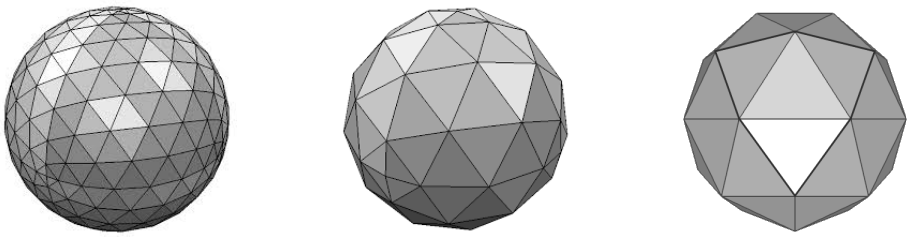
from all existing ones. With partial matching, by querying the database with a 3D model of the targeted component, the database should return a list of rims that share similarities with the new rim design. Previous works related to AM model search are approaching this problem with varying degrees of success that depends on different factors such as choosing certain parameters, and pruning the database of 3D models.

One of the major challenges with searching over the 3D models themselves, as opposed to solely the metadata, is how to derive important characteristics from the models to allow for a robust search mechanism that encompasses both exact and partial matching. To solve this problem, we propose using the Fast Fourier Transform (FFT) [18] to convert models to the frequency domain and deduce peak frequencies. We can use this peak information, as well as distances between peaks, to create a unique identifier, or "fingerprint", for each object. The use of fingerprints also saves a great deal of space in database storage as the only information about a design that needs to be saved for search capabilities is its unique fingerprint.

To address this challenge, in this work we propose a novel shape-based retrieval framework for 3D models called Fourier Fingerprint Search (FFS) and our contributions are summarized as follows:

- **Querying:** An enhanced search methodology that incorporates important design features and uses advanced matching techniques that take into account unique patterns associated with each object. Notably, FFS supports both exact and partial matching, presenting results with identical as well as similar shape characteristics derived from the Fourier domain.
- **Personalized search:** We investigate a variety of different modes and enhancements for search, such as shape slicing and interpolation, which offers high adaptability to real-life scenarios.
- **Advanced features:** We analyze two different forms of rotation and transformation to improve matching of 3D CAD models at different angles and positions. Likewise, we evaluate an approach that utilizes 3D FFT to enhance the accuracy of search.

Roadmap: In Section 2 we discuss preliminaries and present an overview of the 3D file formats that our framework interfaces with, as well as efficient audio search techniques that inspired this work. Section 3 outlines the fundamental concepts used in the proposed framework, while Section 4 elaborates on implementation details and various enhancements. In Section 5 we present our experimental results and analysis, while Section 6 discusses prior research efforts. Lastly, Section 7 discusses our concluding remarks.



(a) Fine resolution **(b)** Semi-fine resolution **(c)** Coarse resolution

Figure 1: An STL approximation of a sphere composed of triangular planes with different resolution settings.

2 Preliminaries

2.1 3D CAD file formats

STereoLithography, Standard Triangle Language, or Standard Tessellation Language (STL) is one of the most common file format in additive manufacturing and CAD systems. This file format describes the raw surface of a model broken down logically into a series of small triangles (facets). Although it is not possible to use facets to perfectly represent curved surfaces, increasing the number of triangles renders the inaccuracy negligible. Thus, the same model can be represented with different resolution settings that can affect the shape of the converted object. For instance, in Fig. 1 three spheres of different resolutions are depicted: Fig. 1 (a) has higher resolution and thus the STL file comprises more triangles and larger files compared to Fig. 1 (b) and (c). In addition, STL files come in two flavors: binary and ASCII. Because binary STL files are more compact, they are more widely used than their ASCII counterparts. Lastly, the STL format is open source and well documented.

Alternative CAD file formats include STEP and Initial Graphics Exchange Specification (IGES), which are both widely used for digital exchange of information among 3D CAD systems. Both STEP and IGES are ASCII structured, rendering them easy to read but not optimal in terms of storage requirements. At the same time, there exist various other formats that are used by CAD software, yet most of them use a proprietary structure. Thus, for this work, we focus on the STL file format considering that it is universal and all 3D printers and slicers can read it. We remark that conversion from STEP and other file formats to STL is straightforward using automated tools.

2.2 Fast Fourier Transform

The fast Fourier transform (FFT) [18] is an algorithm that computes the discrete Fourier transform (DFT) of a signal and converts it from the time or space domain to a representation in the frequency domain. FFTs are widely used in various fields and for different purposes, such as digital signal processing, data compression, polynomial multiplication, or even multiplication of large integers

[19, 20].

Given a list of complex numbers $x_0, \ldots, x_{N-1}$, the DFT is defined by the following formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi}{N} kn} \qquad k = 0, 1, \ldots, N-1 \tag{1}$$

where $e^{\frac{i2\pi}{N}}$ is a primitive Nth root of unity. Evaluating the definition of Eq. 1 directly requires $O(N^2)$ operations since there are N outputs X_k, and each output requires a sum of N terms. The fast Fourier transform offers an $O(NlogN)$ implementation of the DFT computation, resulting in a significant runtime improvement.

The Fourier transform, and thus FFT, can be generalized to higher dimensions. Many discrete signals are functions in the 2D space and we can compute their corresponding frequency content using a 2D discrete Fourier transform. Conceptually, the 2D transform is a sequence of 1D transforms with respect to the X and Y axes, resulting in a nested summation as shown in Eq. 2. It is possible to further extend the transform to higher dimensions by introducing an additional nested sum for each dimension.

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \left(e^{\frac{-i2\pi}{N_1} k_1 n_1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} e^{\frac{-i2\pi}{N_2} k_2 n_2} \right) \qquad k_\ell = 0, 1, \ldots, N_\ell - 1, \quad \ell = 1, 2 \tag{2}$$

2.3 Audio Recognition Using Spectrograms

Recognition and search in audio samples has been made possible by using a spectrogram of the sample [21]. A spectrogram is a visual way of representing the signal strength, or the spectrum of frequencies of a signal as it varies with time. Essentially, a spectrogram of an audio signal provides a way to *see what you hear*. Of course, in order to get the frequencies from an audio sample and generate its spectrogram, a Fourier transformation is necessary.

Given a spectrogram of the sample, audio recognition algorithms [22] find time-frequency points that have high energy content (*peaks*), and filter out the remaining points. The list of peaks is also referred to as a "constellation map", since the points often resemble a star field. Different audio recognition algorithms treat the constellation map differently, however, the main theme is to generate an identifier of the audio sample using a combination of the frequency peaks. One possible approach to create an identifier of the sample is to hash pairs of peaks along with their timing offsets on the spectrogram, as we illustrate in Fig. 2. The set of the generated hashes can represent the unique identifier of the frequency-domain sample.

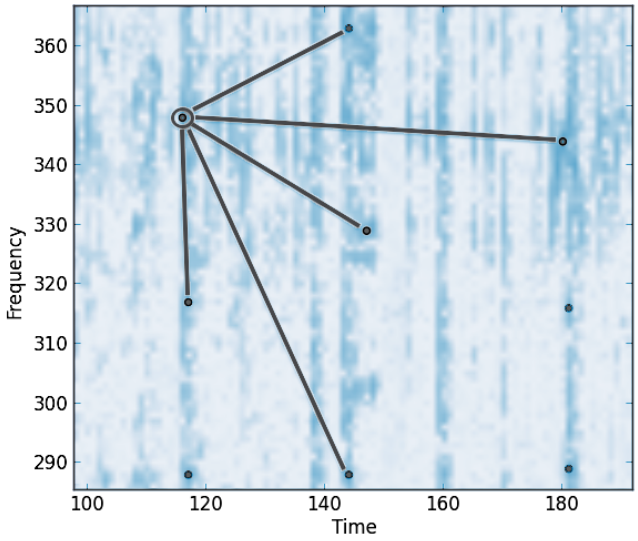


Figure 2: Time-frequency spectrogram: darker color refers to higher magnitudes. Audio recognition is possible by finding peak frequencies with the highest magnitudes along with their corresponding timing offsets, and generating hash digests for different peak pairs.

3 Fourier Fingerprint Search (FFS)

Motivated by the idea of searching in the sound domain, we propose a novel framework for 3D CAD model search, dubbed Fourier Fingerprint Search (FFS), that enables retrieval based on both exact and partial shape matching. FFS comprises two main phases, the offline database population and the online search/query. Both phases follow a similar process in order to generate the identifier, referred to as *fingerprint*, of a CAD file. A fingerprint consists of a list of hash digests, which in this work we call *signatures*, that uniquely define a 3D model. Moreover, it is essential for FFS that the addition of thousands of CAD files in the database should not decrease the probability of finding a correct match. Thus, the fingerprint should be robust and uniquely represent the 3D models. The organization of our framework is depicted in Fig. 3 and described in more detail by the following steps.

3.1 Scale, Slice, and Project CAD Model

The first step in our approach entails mapping the 3D points from a given CAD model into a 3D grid representation within FFS, reducing the 3D grid into a sequence of projections on 2D grids and then computing a number of hash digests that act as unique signatures for the initial 3D object. Towards that end, we start by scaling the absolute dimensions of all 3D points in the our CAD model to match the (user-defined) dimensions of our 3D grid. This ensures that the coordinates of the 3D points are neither too small nor too big

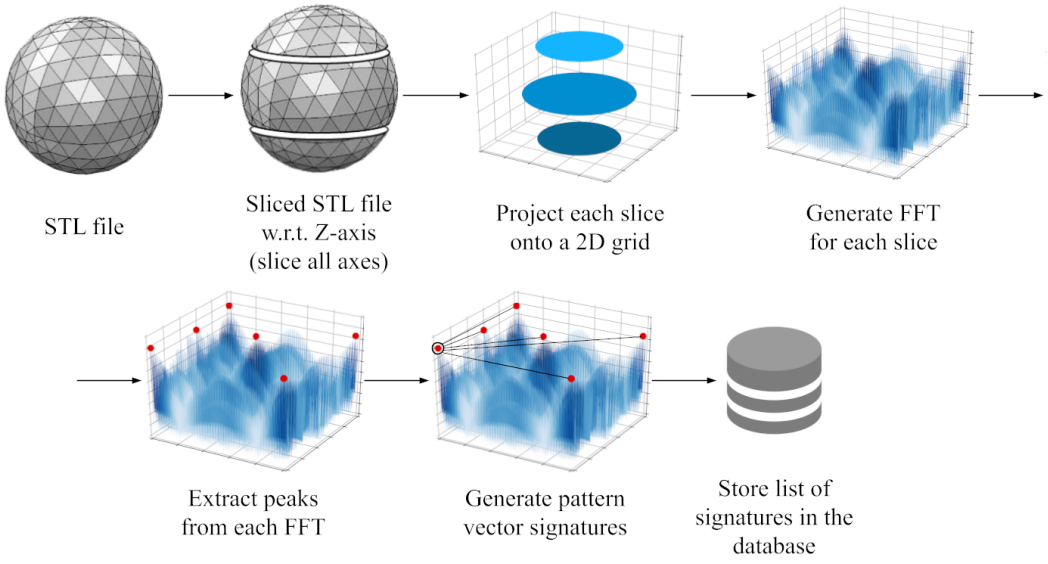


Figure 3: Overview of our approach: (a) The first step is to slice the STL file in a configurable number of slices for all X, Y, and Z axes. (b) Then project each 3D slice onto a two-dimensional binary grid. The total number of grids for each axis is equal to the predetermined number of slices. (c) The next step is to compute the 2D FFT for each grid and extract the peaks. (d) For each peak and its neighbors we compute a number of hash digests that form the fingerprint of the STL file. (e) Finally, the sets of all hashes for all the axes are stored in the database.

for the selected grid size.

After scaling, FFS logically divides the populated 3D grid into N three-dimensional *slices* across a selected axis of orientation. Each slice corresponds to a part of the CAD model, so the union of all N slices in the correct order represents the original design. The intuition for defining this *sequence of 3D slices* arises from the need to introduce the notion of continuity and sequence in a 3D model, similar to what time represents for an audio sample.

Next, FFS projects the three-dimensional points of each 3D slice onto N different 2D grids, so that the original 3D object is ultimately transformed into a sequence of N logically consecutive and parallel 2D projections across a given axis of orientation. Both the slicing and the projection operations are repeated for all axes of the 3D model (i.e., X, Y, and Z) so that the generated 2D grids across different orientations provide a more comprehensive description of the given model. The first arrow of Fig. 3 shows an example of slicing a CAD model into three 3D grids along the vertical axis, while the second arrow represents the projections onto three 2D grids.

3.2 Detect Peaks

One crucial step of the FFS framework is the peak detection operation, which affects all of the succeeding steps. We apply the 2D FFT algorithm in each 2D grid generated by the 'scale, slice and project' step discussed in the previous section and we get the Fourier-domain representation of each 2D grid. Notably, each such frequency representation is a 2D array of Fourier-domain magnitude values that is indexed by a pair of frequencies; to remove possible ambiguity, we will be referring to these 2D arrays as *2D Fourier Domain Arrays (2D-FDAs)*. Moreover, since we previously generated N slices for each of the three axes of the 3D model, after applying the FFT on the 2D grids, we end up with $3 \times N$ 2D-FDAs; these 2D-FDAs define the Fourier representation of the original CAD model across three different orientations.

Consecutively, for each generated 2D-FDA, FFS detects the frequency-domain coordinates (i.e., 2D-FDA indices) of the *magnitude peaks* using a local maximum filter [23]. The filtering result is a Boolean mask of the peaks: *True* is assigned to a magnitude value that is the neighborhood maximum, and *False* is assigned otherwise. Our framework applies the Boolean mask to each 2D-FDA and obtains a set with the magnitudes of the peaks along with their coordinates in the frequency domain (i.e., a peak is defined as a 3-tuple comprising a pair of frequencies and the corresponding magnitude). Given the set of peaks for each 2D-FDA, it is possible to filter them further with respect to their absolute magnitudes and only keep the P most significant (i.e., the P with the highest magnitude). The value of P is configurable in FFS and introduces a trade-off between computation time, storage size, and accuracy as it affects the number of generated signatures.

3.3 Generate Signatures

In order to uniquely represent a 3D CAD file, we introduced the notion of a fingerprint, which comprises a set of intricately chosen hashes that act as signatures of the corresponding 3D object. In FFS, each such signature is defined as *a hash digest over a pair of magnitude peaks drawn from either the same or adjacent 2D-FDAs*. In each such pair, the first peak acts as an "anchor" and the second peak acts as a "target". This is another essential step of FFS, since this collection of signatures should be robust and unique for each different CAD design in order to avoid collisions and false positives. Likewise, we also require support for partial matching using the same fingerprint, which entails that the signature generation algorithm should allow for small shifting of the peak points.

One straightforward approach would be to take the Cartesian product of all of the peaks from all 2D-FDAs and generate one hash digest for each possible pair. This approach will result in a very large number of signatures, yet it may increase accuracy during exact matching. Nevertheless, this will significantly restrict partial matching: for example, a signature of a pair of peaks where the first peak is drawn from the first 2D-FDA and the second peak is drawn from

the last 2D-FDA will be too specific and cause overfitting. In effect, although our goal is to maintain the association between consecutive 3D slices in the Fourier domain, we should avoid associating all possible peaks at the same time (i.e., avoid computing a signature with two peaks from non-adjacent 3D slices).

To avoid the pitfalls of the straightforward approach above, instead FFS performs the following: (a) iterates over the set of peaks detected during the peak detection step (Section 3.2), (b) selects "anchor" peaks from the set, and (c) generates a defined number of signatures for each such anchor using a set of "target" peaks from adjacent downstream 2D-FDAs. The number of signatures corresponding to each anchor peak is referred to as the *fan-out* of that peak, and FFS allows configuring this parameter. An illustration of this approach is shown in the second to last step of Fig. 3: for the selected anchor peak, FFS selects 5 target peaks from adjacent 2D-FDAs (i.e., fan-out of the anchor) and generates one signature for each *(anchor, target)* combination. The larger the fan-out parameter, the smaller the flexibility for partial matching, and the more space requirements to store the corresponding fingerprint in the database. Conversely, if the number of generated signatures for each anchor peak (i.e., the fan-out) is too small, the CAD model fingerprint may not be sufficiently unique, since not having enough hashes increases the probability of collisions among different fingerprints.[1] Thus, it is important to properly tune the fan-out parameter to maximize accuracy.

In more detail, each FFS signature is a cryptographic hash digest over the two frequency-domain coordinates of the anchor peak and the two frequency-domain coordinates of the corresponding target (i.e., a hash of the array indices corresponding to each peak within its 2D-FDA). To introduce the notion of sequence and further reduce the probability of signature collisions, the input of each aforementioned hash digest further incorporates a "distance measure" between (a) the 2D-FDA of the anchor peak and (b) the downstream 2D-FDA of each target peak. In this case, for a given CAD model that has been sliced into a sequence of N slices that are subsequently projected onto 2D grids, we define the distance between two 2D-FDAs as the *offset difference* between the corresponding 2D grids with respect to the slicing axis (X, Y or Z). For example, when FFS is slicing a 3D object along the Z axis to generate a sequence of N 2D grid projections perpendicular to Z, the offset difference between the i-th 2D grid and the j-th 2D grid is $d = |j - i|$, and this is also the "distance measure" between the two corresponding 2D-FDAs for that axis. Moreover, since the potential symmetry in a given CAD model may cause collisions between the signatures generated for different axes, FFS also incorporates an axis identifier as an input to each hash digest, which further minimizes the probability of such collisions.

At the end of this step, FFS generates a single fingerprint for the target 3D model, which is organized as *a graph over all signatures from all anchors.*

[1] Note, here fingerprint collisions refer to having two 3D objects identified with the same set of hash digests, and is different from potential collisions in the hash function itself. In FFS each hash digest is computed using the SHA-1 hash function.

Then, all hashes that comprise this fingerprint are either stored in a database, when FFS is in its training phase, or are queried against the already-populated database to find potential matches during the search phase.

3.4 Fingerprint Search Techniques

In this section we propose two different techniques that employ the fingerprint of a CAD file to identify the top K most similar matches. To perform a query against an already populated database, we start with a candidate CAD file that is the query input. First, FFS performs on-the-fly the fingerprinting step (discussed in subsection 3.3) using the input query, and generates all corresponding signature hashes across the three axes of the 3D model. Then, FFS uses the computed fingerprint of the input file to search in the database for all similar fingerprints. Since a signature in the database may have been generated by more than one CAD file during the learning phase, for each signature of the query file we actively track every possible matching file in the database that has the same signature in its fingerprint. As soon as the database is searched using all possible signatures of the query file, we filter the results and return the top K most similar files to the input query. FFS ranks each result using its *Fingerprint Efficiency Index (FEI)* score, which is our proposed similarity confidence metric computed based on the percentage of matched hashes; FEI is further discussed in Section 5.3 along with our experimental results.

3.4.1 Matching Neighborhoods

Our first methodology is motivated by pattern matching techniques and is based on subgraphs of all signatures associated with the same anchor. Towards that end, we propose the notion of *matching neighborhoods*, where our search takes into account *patterns of signatures* that encode snippets of frequency domain information of each 3D model. In this case, we define a "neighborhood" as the collection of signatures that arise from the same anchor point, so that a match between neighborhoods would be equivalent to a pattern match in the Fourier domain representation of 3D models. We remark that the size of each neighborhood is constant and equal to the fan-out parameter introduced in Section 3.3.

To implement the proposed neighborhoods matching technique, FFS generates each signature while also keeping track of the anchor point associated with it. During the learning phase for each CAD model, our database is populated with tuples that encode the hash of each anchor point's coordinates, along with the list of signatures corresponding to the neighborhood defined by that anchor point. Then, during the search phase, FFS queries its database and computes a sorted list of CAD files whose neighborhoods are the closest match to those of the input query. Specifically, we consider that two neighborhoods are matching when they both contain at least min_{sig} identical signatures, where min_{sig} is a user-configurable threshold parameter. The latter allows tuning the degree of similarity between two neighborhoods, which affects the flexibility for partial

matching: For instance, if min_{sig} is maximum (i.e., equals the neighborhood size), FFS is tuned for limited flexibility towards partial matching as it will attempt to match all the signatures in each neighborhood. Conversely, having a smaller min_{sig} value (e.g., 2) would result in less strict matching, which allows retrieval of similar-looking 3D shapes (i.e., partial matches).

3.4.2 Fine-grained Matching

Our second methodology implements a more fine-grained approach by matching the signatures from the input query with the existing signatures from different CAD models in the database. We refer to this technique as *fine-grained matching* since its goal is to prioritize individual shape characteristics over bigger – *yet harder to match* – patterns. When FFS uses the fine-grained approach, it computes the total number of signatures matched against each existing CAD model in the database that has at least one match. These results are then sorted in descending order so that CAD files with the highest percentage of signature matches against the input query are moved to the top of the list. Finally, FFS returns the top K CAD files most similar to the query and also reports the similarity confidence (FEI index) as a fraction of the signatures matched divided by the total number of query signatures.

4 Advanced Matching Techniques

In this section, we propose a variety of different techniques that enhance the accuracy of our 3D shape recognition framework. First, we outline two methods of rotation in order to consider objects from a variety of angles to get a more complete and unique internal representation of 3D models. Second, we introduce a technique to increase the search accuracy of both the neighborhoods and fine-grained techniques by upscaling the resolution of model representations without affecting the CAD files themselves. Third, we implement our search algorithms using high-order Fourier domain representations for our shapes. The aforementioned techniques can be combined together or work as stand-alone features to improve the accuracy of FFS; the following sections provide a more elaborate description of our techniques.

4.1 Rotation of Slices

The FFS framework populates its database offline with numerous fingerprints, so that during the online search phase it can quickly match the fingerprint of the query object against those stored in the database. In this optimization, our goal is to increase the probability of matching a CAD model when it has been *rotated or transformed*. In particular, during enrollment of new CAD models in the FFS database, we apply our fingerprint generation algorithm to various rotations of the 3D model along all three axes. This is possible by using the following steps: we first slice the object, project it on 2D grids, rotate the grids,

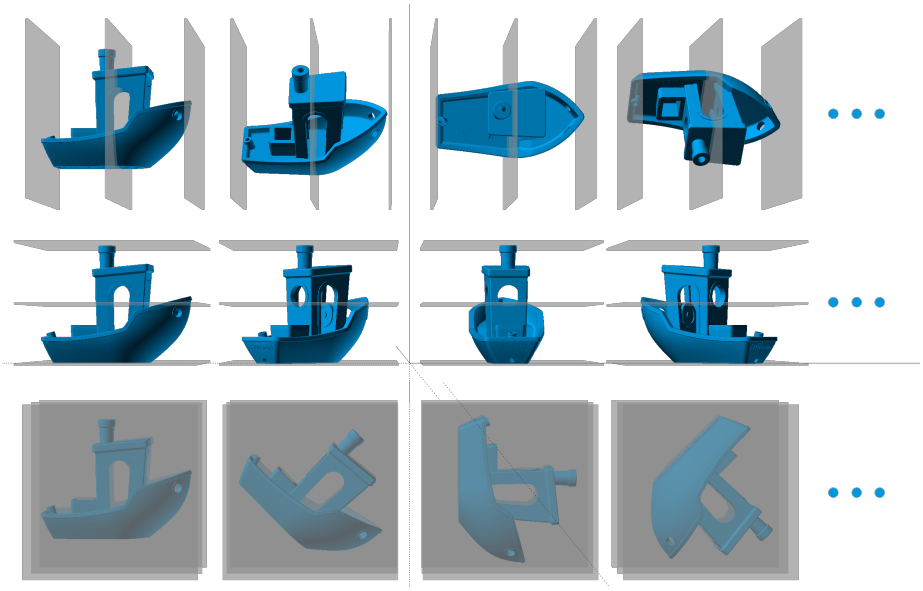


Figure 4: Rotation of Slices: FFS populates its database with different rotations of each slice to increase the accuracy of search.

compute FFTs, generate the signatures, and finally store all of them in the database.

For clarity, we refer to this technique as "rotation of slices", since one can interpret it as having the 3D model steady, rotate the slices and then apply the rest of the steps. Fig. 4 illustrates our rotation of slices approach, where a collection of signatures is generated and stored in the database for each different rotation using 45 degree increments in this example. Notably, using the 45 degree increments depicted in Fig. 4, we would end up with 8× more signatures (i.e., $360/45 = 8$), all describing the same model from different angles. This technique offers a trade-off between offline enrollment time, database storage cost and the probability of finding matches during the online search phase.

With respect to the number of rotations per axis (e.g., 8 in Fig. 4), our evaluation shows that even with a small number of rotations, the search accuracy of FFS is further improved. At the same time, our framework can minimize the storage requirements of the rotation of slices technique by identifying duplicate (redundant) signatures. For example, rotations of a symmetric shape (such as a sphere) do not provide additional information about the 3D object and the generated signatures would be the same for each rotation. In this case, FFS stores a single copy of the fingerprint in the database, optimizing space requirements.

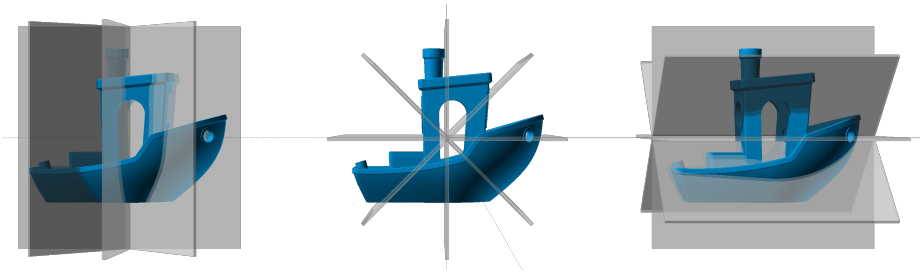


Figure 5: Star Rotations: FFS rotates the object with respect to X, Y, and Z axes and generates signatures for each rotation.

4.2 Star Rotation

Similar to how 3D object scanners work (e.g., [24]), FFS offers the ability to rotate a 3D model on an axis parallel to the projection plane. The intuition is that analyzing a 3D model from different angles offers a more comprehensive understanding of its geometry and features. For instance, a tetragonal pyramid observed directly from below may appear indistinguishable from a cube; however, rotating the pyramid on an axis perpendicular to the observation angle would immediately reveal additional features (i.e., its triangular faces), which allows for correct deduction of the object's true nature.

To enable this capability in our FFS framework, we first define an axis of rotation and a virtual circle on a plane perpendicular to the selected axis, so that its center is located on the axis of rotation. Next, we compute evenly spaced points on our virtual circle that will define rotations by an incremental angle. For example, if the incremental angle is chosen to be ten degrees, this choice would result in 36 evenly spaced points along the circle's circumference. Then, for each selected point on the virtual circle, FFS computes the parametric equation of a line given two points: the center of the circle and a given point on the circumference; for each computed line, we can further define a 3D plane that extends parallel to our selected axis of rotation. In Figure 5, we illustrate an example of different 3D planes that slice a target model across three rotational axes. Since the 3D planes that intersect on each rotational axis form a star shape (when looking at them head-on), we refer to this type of rotation as "star rotation" (cf. "rotation of slices" in the previous section).

Generation of Rotational Slices: While slicing a given CAD model into N 3D slices with respect to the X, Y, and Z axes (as presented in Section 3.1), FFS also computes the *set intersection* between the points on each aforementioned 3D plane and the points within each 3D slice. The resulting set intersections correspond to the rotational slices of the 3D object across each axis of rotation. Each rotational slice is then processed by FFS for peak detection, signature generation and storage in the database.

The star rotation technique further increases the accuracy of both exact and partial matching, while presenting a tradeoff between the incremental angle of

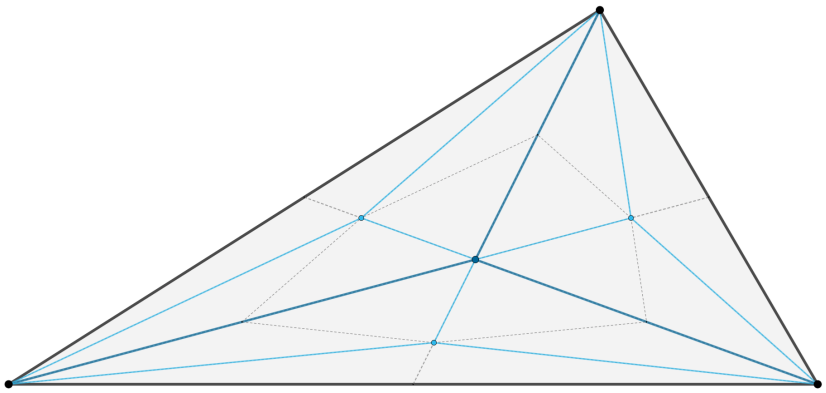


Figure 6: FFS interpolates the STL triangles to upscale the resolution and enhance search by recursively creating new triangles from the centroids of the existing ones.

rotation and the computation/memory overhead associated with the increased number of rotational slices. In FFS, this incremental angle, and hence the number of rotational slices, is configurable and users can balance accuracy with storage requirements for their particular application. In Section 5, we further discuss our experimental evaluation using different choices for incremental angles.

4.3 Interpolation

One of the drawbacks of using the STL file format is that some CAD files are created at low resolutions and hence are composed of relatively large triangles (facets), as shown in Fig. 1(c). This gives a somewhat incomplete picture of the shape as the points defined in the STL file are spaced far apart, leaving large gaps between them. Moreover, since FFS analyzes 3D objects based on the vertices of each facet, having a low number of facets would create a sparse 3D grid during the slicing step discussed in Section 3.1. To address this concern, FFS can upscale the resolution of the 3D object using interpolation over its internal representation of the 3D model.

As illustrated in Figure 6, for each triangle (facet) composing the given 3D object, FFS finds the centroid of the triangle and adds the newly computed point to the quantized 3D grid of the object. Next, our framework defines three new triangles using the centroid of the original triangle and the three original vertices; this process is repeated recursively until all new centroids already have a mapping in the quantized 3D grid of the object. In effect, recursion stops when the entire space inside of the original triangle is fully mapped to points of the 3D grid. Essentially, interpolation enables FFS to convert the *outline* of a triangle into a *solid* triangle, establishing a more accurate internal representation of the original object. We remark that this technique can be used in tandem with our other techniques to improve the accuracy and quality of FFS matches.

Table 1: Configurable thresholds of FFS.

Parameter	Description
grid-size	The 2D (or 3D) slice grid dimensions.
N	Number of slices to divide 3D model.
P	Number of peaks to keep after filtering.
fan-out	Number of signatures for each peak.
K	Max number of search results.
min_{sig}	Min hashes to match in a neighborhood.
slices-rotation	Enable rotation of slices.
star-rotation	Enable star rotation.
interpolation	Enable interpolation.
3D-FFT	Use 3D FFT.

4.4 3D Fourier Domain Representation

As discussed in Section 3.2, FFS employs the 2D FFT to generate 2D-FDAs and detect peaks. Nevertheless, instead of projecting 3D slices onto 2D grid projections, it is possible to transform the 3D slices directly using the *three-dimensional FFT*. In this case, each 3D slice that is being processed by the 3D FFT is a (relatively thin) rectangular prism. Specifically, the thickness of the 3D slice impacts the total number of 3D points processed together by the 3D FFT: decreasing the slice thickness decreases the required memory overhead for each FFT invocation, while it increases the total number of 3D slices to be processed. Thus, in this memory/time tradeoff, having thin slices ensures that only a relatively small part of the 3D object needs to be kept in memory at any time, and each 3D slice can be discarded after FFT processing and signature generation.

Using a 3D FFT eliminates any loss of information caused by projecting 3D points on 2D grids, so it offers a more complete representation of the 3D model in the Fourier domain. At the same time, each three-dimensional transformation incurs increased overheads with respect to computation time, memory overhead, as well as time to populate and query the database compared to the 2D FFT approach. In our experiments (Section 5) we provide a more elaborate comparison of the overheads of the 3D FFT transformations.

5 Evaluation of our Framework

The goal of our experimental evaluation is to assess the accuracy of FFS and how our proposed advanced matching techniques improve the CAD model retrieval results. FFS contains various thresholds and configurable parameters (shown in Table 1) that affect the timing and storage overheads, as well as the accuracy of our framework. Table 2 summarizes performance and top-3 accuracy for our

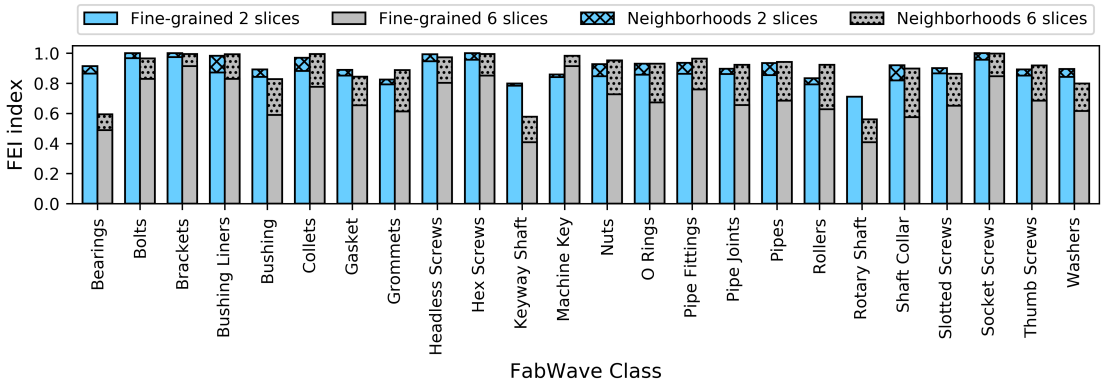


Figure 7: Fingerprint Efficiency Index (FEI): FEI scores for all FabWave classes using 2 and 6 slices for both the fine-grained and neighborhood-based matching ($min_{sig} = 2$). The patterned bars stacked on top of the solid-color ones indicate the FEI improvement attributed to the use of neighborhoods over the fine-grained approach (higher is better).

exact match experiments using the FabWave dataset [25]. In this case, top-3 accuracy measures if one of the first three retrieved answers correctly identifies the input query model; in general, top-N accuracy measures if the expected answer is among the first N retrieved results.

5.1 Experimental Setup

For evaluating FFS we employ the FabWave CAD dataset that contains approximately 3000 unique manufacturing models. This dataset consists of 24 classes of STL files corresponding to various models of components such as screws, bolts, and washers and was designed to support manufacturing research. Our experimental results are obtained on a t3.2xlarge AWS EC2 instance running with eight virtual processors up to 2.5 GHz and 32 GB RAM. We implemented FFS using Python 3.6.8 and LevelDB 1.22, an open-source key-value storage library provided by Google.[2] Finally, the host system is running Ubuntu 18.04 with the 4.15.0 Linux kernel and OpenSCAD 2019.05, which is used to transform 3D models in our experiments.

5.2 Database Learning

Enrolling in our database all the FabWave design files takes approximately 20.8 minutes and 55.3 minutes for $N = 2$ and $N = 6$ slices per axis respectively. We remark that this is a one-time cost, as once the files are loaded they will remain in the database until removed. Files can be added to the database in approximately 0.4 second per file (using $N = 2$ slices) and 1.08 seconds per file (using $N = 6$ slices). For the FabWave design files, which consist mostly of

[2]FFS framework will be released as an open-source repository on GitHub.com.

small components such as screws and bolts, it is ideal to choose a small number of slices as the N parameter. If N is too large, the number of peaks per slice will be minimal and therefore, each generated fingerprint encodes less unique information about the corresponding model. At the same time, as reported in Table 2, increasing the number of slices offers a trade-off with performance: having more slices would increase the number of projections on 2D grids that need to be transformed to the Fourier domain using 2D FFT.

In the FabWave dataset, we observed that there are many identical files with different filenames in some classes. This was discovered through hashing the actual contents of all STL files (i.e., excluding the first and the last line of the STL which encodes only file name information) with SHA-1 and creating a key-value dictionary with a filename as the key and a corresponding hash as the value. After sorting this dictionary by hash and found many identical hashes which indicate that multiple STL files in FabWave have identical contents. To prevent this from adversely affecting our search results, we removed all redundant files, of which there were exactly 618. The resulting dataset used for our experiments consists of the 3079 unique files.

5.3 FEI Index

In addition to top-N accuracy of our framework, we further define a custom metric called "fingerprint efficiency index" or *FEI*, which corresponds to the number of signatures (or neighborhoods) that are matched divided by the total number of signatures/neighborhoods in the query file. For each query, this metric is computed for every file in the database that has at least one signature or neighborhood match with the input. Essentially, this metric is a performance indicator of how uniquely a given object is represented by its fingerprint. The range of FEI is 0.0 to 1.0, where a score of 0.0 indicates that no signatures/neighborhoods in the query matched with a particular file in the database and a FEI score of 1.0 indicates that all signatures/neighborhoods in the query matched. From a user's perspective, files with a FEI score close to 0.0 have little in common with the query object while results closer to 1.0 indicate an exact match. FFS will return the top K files in the database with the highest FEI score for a given query.

In Fig. 7 we report the FEI scores for all the classes in the FabWave dataset using both our search techniques with $N = 2$ and $N = 6$ slices. For neighborhoods, we set the min_{sig} parameter to 2, leaving more room for partial matches. As expected, the neighborhoods technique achieves higher FEI averages for most of the classes compared to the fine-grained method. This observation is attributed to the following: First, the total number of signatures that our fine-grained technique uses for matching is significantly higher than the total number of neighborhoods. Moreover, the fine-grained technique operates on individual signatures and identifies a match when a database file has the same signature in its fingerprint as the input query, whereas the neighborhood approach identifies a match when at least min_{sig} signatures are matched within the same group. As

Table 2: Fingerprint generation cost and top-3 accuracy experiments for exact matching using the FabWave dataset with *fan-out*= 10, $P = 10$ peaks, and $min_{sig} = 10$.

FabWave Class	Number of Files	Timing (sec)		Exact Matching Accuracy			
		Fingerprint generation		Fine-grained		Neighborhoods	
		Number of Slices		Number of Slices		Number of Slices	
		2	**6**	**2**	**6**	**2**	**6**
Bearings	58	22.2	62.0	1.0	1.0	1.0	1.0
Bolts	11	6.4	11.9	1.0	1.0	1.0	1.0
Brackets	52	17.3	48.1	1.0	1.0	1.0	1.0
Bushing Damping Liners	13	5.9	14.3	1.0	1.0	1.0	1.0
Bushing	310	117.5	331.5	1.0	1.0	1.0	1.0
Collets	70	24.8	67.4	1.0	1.0	1.0	1.0
Gasket	73	28.7	77.6	1.0	1.0	1.0	1.0
Grommets	202	86.8	224.2	1.0	1.0	1.0	1.0
Headless Screws	154	106.5	211.4	1.0	1.0	1.0	1.0
Hex Head Screws	84	45.7	95.0	1.0	1.0	1.0	1.0
Keyway Shaft	250	68.9	195.3	0.97	1.0	0.97	0.94
Machine Key	76	18.8	53.1	1.0	1.0	1.0	1.0
Nuts	47	18.9	52.9	1.0	1.0	1.0	1.0
O Rings	376	164.3	445.7	1.0	1.0	1.0	1.0
Pipe Fittings	28	14.6	35.3	1.0	1.0	1.0	1.0
Pipe Joints	10	5.6	13.0	1.0	1.0	1.0	1.0
Pipes	9	5.8	12.8	1.0	1.0	1.0	1.0
Rollers	14	5.9	16.0	1.0	1.0	1.0	1.0
Rotary Shaft	250	70.2	217.2	0.73	0.98	0.72	0.85
Shaft Collar	57	23.9	66.8	1.0	1.0	1.0	1.0
Slotted Flat Head Screws	112	39.3	112.3	1.0	1.0	1.0	1.0
Socket Head Screws	154	75.6	179.7	1.0	1.0	1.0	1.0
Thumb Screws	23	8.6	23.1	1.0	1.0	1.0	1.0
Washers	646	264.5	748.8	1.0	1.0	1.0	1.0
Total	**3079**	**1246.9**	**3315.4**	-	-	-	-
Average	-	**0.40**	**1.08**	**0.975**	**0.999**	**0.974**	**0.983**

a result, our neighborhood-based matching operates at a more coarse granularity compared to fine-grained matching.

5.4 Exact Matching

One important test of any search framework is its ability to successfully query an object identical to an existing database entry and retrieve the expected output. Table 2 presents the top-3 accuracy for *exact matching* across all FabWave classes, as well as the fingerprint generation cost while populating the FFS database with 3D models of each class. In particular, since both the enrollment

and search phases compute the same set of signatures for each file in the dataset, the fingerprint generation cost of a single file is approximately the same in both cases. In fact, the only difference between these two phases is that enrollment generates a fingerprint to add it in the database while searching computes a fingerprint from the input file to query the database. The average time to generate one fingerprint and either store it in the database or retrieve it is 0.4 seconds and 1.08 seconds with $N = 2$ and $N = 6$ slices respectively. Notably, the overall top-3 accuracy over 3079 CAD models in the dataset approaches 99.9% using $N = 6$ slices for our fine-grained technique.

Our analysis reveals that the CAD files within the same class have very similar shape characteristics: in many cases, the only difference between files is a small change in length or volume. For example, as we observe in Table 2, the "Rotary Shaft" class has relatively lower accuracy for $N = 2$ slices with either of our search techniques. This observation is attributed to the large number of files in this class (namely 250), as well as the high similarity among these models. Indeed, the "Rotary Shaft" class is composed of mostly oblong objects (i.e., objects that are long and at the same time very thin). As a result, using only $N = 2$ slices does not sufficiently break down this peculiarity, which causes degraded FEI score and accuracy results. Conversely, using more slices increases the accuracy to 98% for our fine-grained technique.

5.5 Classification

An essential indicator of the viability of a search framework is its ability to return values in the same category, or class, as the input query, especially when the query does not exist in the database. As already discussed, FFS returns the top K similar 3D models that match the query file; in case of the FabWave dataset, which is organized in 24 classes, the classification operation identifies files from the same class as the input query within the top K results of FFS, given that at least K files exist in that FabWave class. Thus, when a user queries the database using an object from a specific class (e.g., a "hex head screw"), the top K results should include objects from the same class (i.e., hex head screws).

To evaluate our framework's classification accuracy, we perform 10-fold cross-validation experiments by splitting each class of CAD files in the dataset into 10 subsets of similar size. For our cross-validation process, we select 9 subsets from each class and train the FFS database with the union of all these subsets (i.e., 216 subsets across 24 classes, corresponding to 90% of our dataset); each CAD file from the remaining 10% of the dataset is used as a test set to query the database. This selection process is repeated a total of 10 times so that each iteration uses a different test set for queries, and all CAD files in the FabWave dataset are used as an input query exactly once. Our classification results are summarized in Fig. 8 for different *fan-out* parameters.

Fan-out experiments: The exact matching results presented in Table 2 correspond to a *fan-out* value of 10, which was experimentally selected to optimize

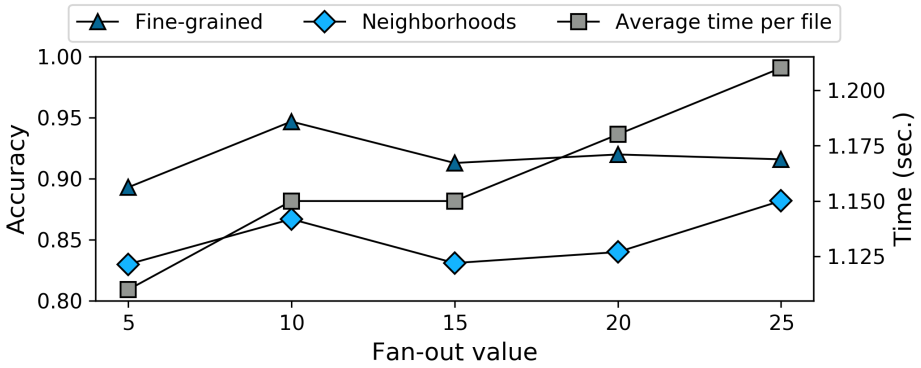


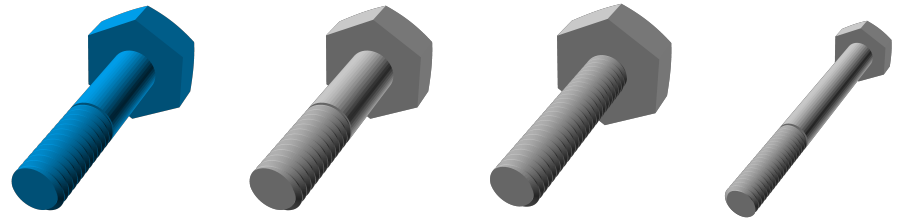
Figure 8: Classification using 10-fold cross-validation: Experimental results for average top-3 classification accuracy and average fingerprint generation time with different *fan-out* values using 10-fold cross-validation. Analysis based on both the fine-grained and the neighborhoods matching technique for all 24 FabWave dataset classes using $N = 4$ slices and $min_{sig} = 5$.

accuracy. To further illustrate the effect of this parameter on classification, in our 10-fold cross validation experiments we vary the *fan-out* value from 5 to 25 (Fig. 8). As with exact matching, we observe that FFS achieves the highest classification accuracy with a relatively small fan-out value of 10. This is attributed to the fact that a larger fan-out value would generate more signatures per CAD model and thus incur a higher collision probability between them. Conversely, having a smaller fan-out value (e.g., 5) does not yield enough signatures to accurately describe the CAD model, which results in decreased matching performance. Thus, for a given dataset, the fan-out parameter needs judicious tuning to optimize accuracy. Using a *fan-out*= 10, our classification experiments show that FFS can achieve 95% top-3 accuracy on average using fine-grained matching.

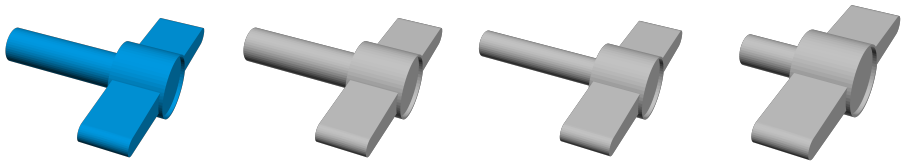
5.6 Partial Matches

Like classification, our experiments further verify the ability of FFS to retrieve models similar to an input query (i.e., partial matching). Fig. 9 illustrates this capability using example designs from the "Hex Head Screws", "Thumb Screws", and "Gasket" class of the dataset: The first column shows the input query models, while the next three columns show the top-3 results for each query. The three examples in the figure illustrate that FFS returns designs from the same class as the query (e.g., for Fig. 9a, all top-3 results are "hex screws" instead of a different class, such as "headless screws" that have a similar shape). Moreover, we observe that the returned results look very similar to the query model and, in some cases, the differences are minute and not obvious at first glance:

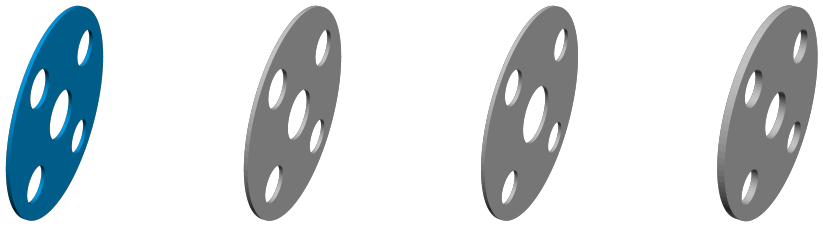
1. The query in Fig. 9a returns an exact match first and two additional "Hex Head Screws", one with more threads and one with the same number of



(a) Hex Head Screw Query **(b)** Match #1 **(c)** Match #2 **(d)** Match #3



(e) Thumb Screw Query **(f)** Match #1 **(g)** Match #2 **(h)** Match #3



(i) Gasket Query **(j)** Match #1 **(k)** Match #2 **(l)** Match #3

Figure 9: Search with partial matching: Fine-grained FFS search for partial matching using "Hex Head Screw", "Thumb Screw", and "Gasket" queries. The first column of each row (blue highlight) is the input query model followed by the top 3 retrieved matches.

threads but is skinnier and longer.

2. Using the query of Fig. 9e retrieves an exact match first as well as two similar "Thumb Screws", one with a skinnier shaft and one with a shorter shaft.

3. The search for a "Gasket" (Fig. 9i) first returns an exact match along with two additional matches, one with with different hole diameters and one slightly thicker.

Queries using altered files: We extend our analysis of partial matching by modifying the CAD files of the FabWave dataset in a variety of ways and then use the altered files as queries to a database already populated with the original

dataset. Our modifications include (a) perturbing a CAD file, (b) degrading its resolution (i.e., reducing number of triangles as illustrated in Fig. 1) and (c) rotating the 3D object by varying degrees across all three axes. The next paragraphs elaborate on each modification approach and discuss the performance of FFS for partial matching using altered files.

5.6.1 Random Perturbations

For all classes of the dataset, we introduce perturbations to the STL files by randomly permuting the order of the STL triangles to generate different files with respect to binary contents, yet encoding exactly the same 3D share; this transformation is applied directly to each CAD file using Python scripts. In our results, we observe no changes in both accuracy and FEI of each CAD file in each class, since the relative order in which the triangles appear in a file does not affect the 3D model (i.e., the same set of triangles encodes the same information irrespective of the order). Indeed, FFS is internally sorting the STL vertices before processing them, so its internal representation of the 3D model does not depend on the order of triangles. This experiment illustrates the robustness of our approach when the contents of STL files are perturbed (e.g., when the files are opened and then saved by CAD programs that alter the order of triangles representing the design).

5.6.2 Degrading the Model Resolution

A more aggressive approach to evaluate the limits of our framework's partial matching capabilities is by altering the resolution of the 3D object. In this case, it is possible to degrade the resolution of an CAD model by randomly removing some triangles from the STL file. To evaluate this method, we created Python scripts that tamper with STL files and remove 5% random facets, which sometimes creates open and discontinuous surfaces. In case, FFS was still able to successfully match the degraded STL models with the original (untampered) design in the database with a negligible accuracy difference.

5.6.3 Random Rotations

In addition to perturbations, we also evaluate the robustness of the partial matching using random rotations in the design. Specifically, we employ the OpenSCAD software to generate variants of the original 3D models in the FabWave dataset so that each variant is rotated by 1-5 degrees in the X, Y and Z axis. In our experiments, we use the rotated variants as input queries queries against a database that is populated with the original files, and analyze the returned results. For each rotated variant, partial matching is expected to retrieve the original (non-rotated) CAD file from the database, along with similar files from the same class as the input. At the same time, querying with random rotations is expected to decrease the FEI score since the rotated files are not enrolled in the database.

Table 3: Partial matching using random rotations: Average top-5 search accuracy for partial matching when rotating the query axes using OpenSCAD. Our analysis uses $N = 4$ slices, *fan-out* $= 10$, $P = 10$, and $min_{sig} = 5$. FG and NB correspond to our fine-grained and neighborhoods techniques respectively.

FabWave Class	**Partial Matching Accuracy**					
	***x* rotation**		***x, y* rotation**		***x, y, z* rotation**	
	FG	**NB**	**FG**	**NB**	**FG**	**NB**
Bearings	0.84	0.76	0.57	0.5	0.47	0.4
Bolts	0.55	0.09	0.27	0.27	0.36	0.36
Brackets	0.94	0.79	0.92	0.63	0.85	0.65
Bushing Damping Liners	0.92	0.92	0.92	0.92	0.92	0.92
Bushing	0.98	0.92	0.76	0.7	0.68	0.5
Collets	1.0	1.0	1.0	1.0	1.0	1.0
Gasket	0.95	0.93	0.4	0.44	0.45	0.52
Grommets	0.89	0.75	0.83	0.76	0.85	0.73
Headless Screws	0.96	0.95	0.92	0.93	0.93	0.94
Hex Head Screws	0.94	0.93	0.95	0.96	0.95	0.95
Keyway Shaft	0.98	0.86	0.98	0.94	0.95	0.98
Machine Key	0.78	0.61	0.76	0.39	0.72	0.38
Nuts	1.0	0.96	0.96	0.94	0.96	0.91
O Rings	0.99	0.98	0.98	0.97	0.98	0.96
Pipe Fittings	1.0	1.0	1.0	1.0	1.0	0.93
Pipe Joints	0.6	0.4	0.2	0.2	0.3	0.1
Pipes	0.78	0.78	0.78	0.56	0.89	0.89
Rollers	0.43	0.36	0.64	0.36	0.57	0.29
Rotary Shaft	0.87	0.8	0.85	0.84	0.88	0.92
Shaft Collar	1.0	1.0	0.98	0.98	0.96	0.96
Slotted Flat Head Screws	0.87	0.67	0.8	0.69	0.54	0.43
Socket Head Screws	0.98	0.93	0.98	0.94	0.98	0.94
Thumb Screws	0.96	0.96	0.96	0.91	0.91	0.91
Washers	0.83	0.95	0.83	0.94	0.74	0.85
Average	**0.915**	**0.886**	**0.86**	**0.845**	**0.823**	**0.804**

In Table 3 we analyse the impact of random rotations to the top-5 accuracy of search when rotating on the X axis, both X and Y or all three axes respectively. As we observe in our results, the impact of random rotations across all three axes degrades accuracy by about 15-20%. Indeed, FFS can retrieve the correct model as a top-5 result with 80-83% probability using both of our matching techniques, *without enabling any of the advanced features* proposed in Section 4. Notably, random rotations have a bigger impact on some classes (such as "Bearings" and "Gasket") compared to others (e.g., "Collets" and "Pipe Fittings"). This is attributed to shape characteristics of each class and how rotations across all axes affects the generated signatures. For instance, rotating a Gasket (Figs. 9i)

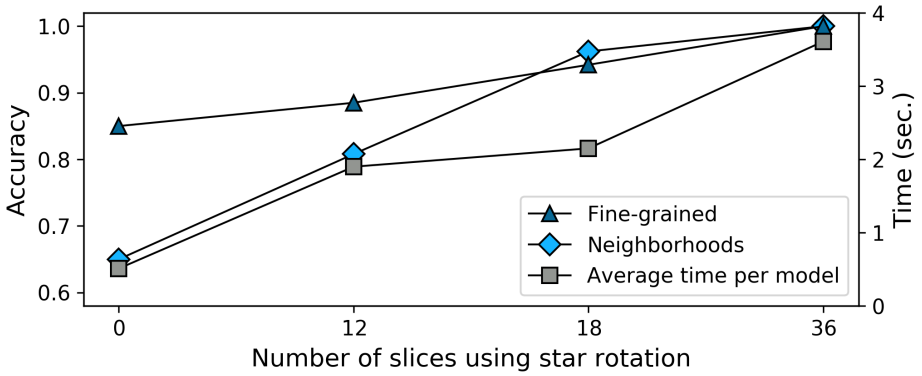


Figure 10: Enhanced search with star-rotations: Average fingerprint generation time per CAD model and top-5 accuracy for partial matching employing the star-rotations optimization for the "Brackets" class with $N = 2$ slices, *fan-out* $= 10$, $P = 10$ peaks and $min_{sig} = 5$. Increasing the number of rotations from zero (i.e., no rotation) to 36 improves the average accuracy by 53% for neighborhoods and 18% for fine-grained.

across its long side (i.e., using an axis across its diameter) can impact the detected peaks in the Fourier domain and increase the fingerprint differences with the original design. Thus, to increase the matching accuracy in these cases, we can further employ our advanced feature techniques, as discussed next.

5.6.4 Rotation of Slices and Star Rotations

Employing our two rotation techniques can significantly improve the accuracy of FFS during partial matching. Our evaluation focuses on the "Brackets" class of FabWave, which yields non-optimal partial matching accuracy when the input query is randomly rotated across all three axes (Table 3).

Rotation of Slices: In this experiment, we divide each 3D model in $N = 2$ slices and populate the FFS database with fingerprints of each file as well as the fingerprints generated using rotational slices at 45 degree increments (Section 4.1). Notably, generation of rotational fingerprints is a one-time (offline) cost only applicable to enrollment, and does not re-occur during search; to match an input query, FFS *does not need to compute the rotational slices* of the input file as part of the query fingerprint. This is possible since the rotations are already stored in the database, so the query fingerprint will properly match the closest rotational fingerprint in the database. Nevertheless, our analysis shows that the FabWave dataset does not benefit from rotation of slices, since most components already have high rotational symmetry.

Star Rotations: This optimization offers accuracy improvements as it allows FFS to encode additional information about each enrolled 3D model from multiple angles, as discussed in Section 4.2. Our experimental results for the "Brackets" class are presented in Fig. 10 for a varying number of rotations, starting

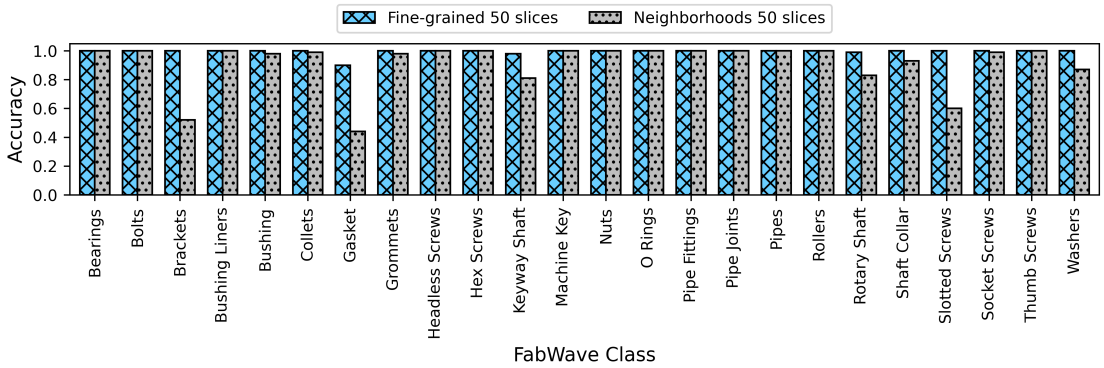


Figure 11: 3D FFT Accuracy: Top-3 accuracy scores for all FabWave classes using 50 slices for both the fine-grained and neighborhoods techniques (with $min_{sig} = 2$).

from zero (i..e, no rotations) to 36. Compared to the rotation of slices discussed in the previous paragraph, using 36 star rotations can substantially increase the average top-5 accuracy by 18% for fine-grained search as well as by 53% for neighborhoods, resulting in 100% average accuracy. In this case, maximizing the accuracy is traded for increased computational overheads, so the average fingerprint generation time increases from about 0.5 seconds to about 3.7 seconds. Likewise, using 18 star rotations offers 10% and 47% accuracy improvement for the fine-grained and neighborhoods-based search respectively, with 2.2 seconds average fingerprint generation time.

5.7 Exact Matching with 3D Fourier Transforms

As discussed in Section 4.4, FFS can use a three-dimensional Fourier transform to avoid any information loss from projecting 3D slices onto 2D grids. In this Section, we explore the benefits of using the 3D FFT with exact matching experiments across all 24 FabWave classes. An important observation is that increasing the number of 3D slices helps manage the 3D FFT memory overhead, as the latter depends on the number of points in each 3D grid; thus, for our 3D FFT evaluation we tune FFS to use 50 3D slices. As illustrated in our top-3 accuracy experiments (Fig. 11), the 3D-FFT performs equally well as the original 2D-FFT approach for our fine-grained technique, but neighborhoods accuracy is degraded for certain FabWave classes. This is due to the expanded size of the neighborhoods, since the increased number of slices yields a significantly larger number of signatures compared to the 2D-FFT approach (which requires fewer slices such as $N = 2$ or 6). Moreover, when the min_{sig} parameter is kept low to increase partial matching capabilities, fewer identical features in the design are required to return a match. As a consequence, the neighborhoods approach occasionally results in partially matched files receiving a higher or equal FEI score to the expected exact match.

We remark that the 3D-FFT approach incurs an increased computation overhead for fingerprint generation (approximately 14 seconds per file), which impacts the time required for database enrollment as well as for executing a query. Likewise, a consequence of using more 3D slices is the increased number of generated signatures, which in turn increases the storage requirements for the FFS database. At the same time, while the use of higher-dimension Fourier transforms offers a more complete internal representation of the 3D models at an increased computational cost, our experiments show equivalent accuracy benefits for the FabWave dataset compared to the 2D-FFT default configuration.

5.8 Storage Requirements

The storage requirements of FFS are highly dependent on the parameters used to generate the fingerprint of each CAD file. The required size of the signatures of a file is expressed by the following formula:

$$sigs_{size} = 3 \times N \times (P - fanout) \times fanout \times hash_{size}, \tag{3}$$

where N is the number of slices and P is the number of peaks. Essentially, for each one of the 3 axes and for each slice, FFS calculates $(P - fanout) \times fanout$ signatures, where each signature is $hash_{size}$ bytes. In our implementation we used the SHA-1 hash algorithm that computes 160-bit (i.e., 20-byte) digests. The FFS database stores these 20-byte hashes, along with the corresponding CAD file names, which can be of arbitrary length. Moreover, to enable our neighborhoods technique, FFS needs to further track anchor and signature pairs in the database. Thus, to enroll all 3079 FabWave dataset files and support our two search techniques with *fan-out* = 10, $P = 15$ peaks and $N = 2$, 4, or 6 slices, FFS requires 90, 230 and 370 MB of database storage respectively.

6 Related Work

Audio recognition and search impose a challenging problem. Prior to Shazam [22], a popular music discovery algorithm, there was no efficient way to process and search for an audio sample in a big database. Shazam employs the notion of a fingerprint for a song, which can uniquely identify that audio sample. A fingerprint consists of a number of hash IDs that combine key frequencies with their timing offsets on the spectrogram of the song. A database is populated with fingerprints of a large song collection offline, and during search, the fingerprint of the query audio sample is generated and matched against the large set of fingerprints in the database. Many sound recognition services [26], such as SoundHound, Apple's Siri, and Google's Assistant, follow a paradigm similar to Shazam.

Like audio search, shape recognition and retrieval is essential for the Additive Manufacturing industry in order to accelerate development of new designs. Nevertheless, the increased dimensionality imposed by searching 3D objects, as

opposed to one dimensional audio data, results in a far more complex problem. Many works focus on two-dimensional data (e.g., [27]), while recently search in the three dimensional space has become a more active area of research [28]. Iyer *et al.* [29] categorize various shape search methods in manufacturing feature recognition [30], local feature descriptors [31], and global feature descriptors such as Zernike moments [10] and spherical harmonics [9, 32] of different parts of a given 3D object, to calculate similar shapes. The categorization continues with graph and tree based techniques [12, 33, 34], which generate an interpretation of a CAD model that incorporates all possible association sequences (i.e., matchings) between model surfaces and scene surfaces, as well as histogram-based methods (e.g., [35, 36]). Likewise, the authors of [11] propose a method for searching 3D models using two-dimensional views, called adaptive views clustering. Finally, [37] presents a technique that utilizes convolutional neural networks to enable 3D object recognition and [38] proposes three partial retrieval modes including normal retrieval, exact retrieval and relaxed retrieval targeting the early design stage.

7 Concluding Remarks

In this paper, we present an efficient and highly configurable framework for searching 3D CAD models. Our methodology is based on the generation of unique and efficient signatures that encode important characteristics of these models in the frequency domain. To deduce these characteristics, we leverage multi-dimensional FFTs and find the magnitude peaks in the frequency domain. Based on the generated signatures, we develop two alternative search techniques: the first focuses on the fine-grained characteristics of the CAD models, while the second introduces the notion of neighborhood-based search that offers more coarse-grained matching. Furthermore, we optimize the accuracy of these techniques by introducing two forms of rotational signatures that encore information about 3D designs across different angles.

We extensively evaluate our search framework using the FabWave CAD dataset, which consists of over 3000 realistic 3D CAD designs. In our experiments we analyze the exact matching performance of our techniques and report up to 99.9% average top-3 accuracy for fine-grained matching. Likewise, without rotational signatures, our partial matching experiments show an average top-5 accuracy of 80-83%, which can be further increased up to 100% using our star-rotation optimization technique. Our experiments further analyze the impact of changing various configurable parameters, such as the number of rotational slices; these parameters can be fine-tuned to optimize performance of our search framework for different target datasets.

References

[1] M. Savastano, C. Amendola, D. Fabrizio, E. Massaroni *et al.*, "3D printing in the spare parts supply chain: an explorative study in the automotive industry," in *Digitally supported innovation*. Springer, 2016, pp. 153–170.

[2] S. C. Joshi and A. A. Sheikh, "3D printing in aerospace and its long-term sustainability," *Virtual and Physical Prototyping*, vol. 10, no. 4, pp. 175–185, 2015.

[3] R. Bracci, E. Maccaroni, and S. Cascinu, "Bioresorbable airway splint created with a three-dimensional printer," *New England Journal of Medicine*, vol. 368, pp. 2043–2045, 2013.

[4] C. W. Hull, "Apparatus for production of three-dimensional objects by stereolithography," *United States Patent 575,330*, 1984.

[5] C. W. Hull, S. T. Spence, D. J. Albert, D. R. Smalley, R. A. Harlow, P. Steinbaugh, H. L. Tarnoff, H. D. Nguyen, C. W. Lewis, T. J. Vorgitch *et al.*, "Methods and apparatus for production of three-dimensional objects by stereolithography," 1991.

[6] R. Arnott, "The RepRap project—open source meets 3D printing," in *Computer and Information Science Seminar Series*, 2008.

[7] W. Gao, Y. Zhang, D. Ramanujan, K. Ramani, Y. Chen, C. B. Williams, C. C. Wang, Y. C. Shin, S. Zhang, and P. D. Zavattieri, "The status, challenges, and future of additive manufacturing in engineering," *Computer-Aided Design*, vol. 69, pp. 65–89, 2015.

[8] T. G. Gunn, "The mechanization of design and manufacturing," *Scientific American*, vol. 247, no. 3, pp. 114–131, 1982.

[9] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs, "A search engine for 3D models," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 1, pp. 83–105, 2003.

[10] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung, "On visual similarity based 3D model retrieval," in *Computer graphics forum*, vol. 22, no. 3. Wiley Online Library, 2003, pp. 223–232.

[11] T. F. Ansary, M. Daoudi, and J.-P. Vandeborre, "A Bayesian 3-D search engine using adaptive views clustering," *IEEE Transactions on Multimedia*, vol. 9, no. 1, pp. 78–88, 2006.

[12] D. A. Keim, "Efficient Geometry-Based Similarity Search of 3D Spatial Databases," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '99. Association for Computing Machinery, 1999, p. 419–430.

[13] N. Gupta, F. Chen, N. G. Tsoutsos, and M. Maniatakos, "ObfusCADe: Obfuscating additive manufacturing cad models against counterfeiting," in

Proceedings of the 54th Annual Design Automation Conference, 2017, pp. 1–6.

[14] S. E. Zeltmann, N. Gupta, N. G. Tsoutsos, M. Maniatakos, J. Rajendran, and R. Karri, "Manufacturing and security challenges in 3D printing," *Jom*, vol. 68, no. 7, pp. 1872–1881, 2016.

[15] N. G. Tsoutsos, H. Gamil, and M. Maniatakos, "Secure 3D printing: Reconstructing and validating solid geometries using toolpath reverse engineering," in *Proceedings of the 3rd ACM Workshop on cyber-physical system security*, 2017, pp. 15–20.

[16] F. Chen *et al.*, "Embedding tracking codes in additive manufactured parts for product authentication," *Advanced Engineering Materials*, vol. 21, no. 4, p. 1800495, 2019.

[17] W. C. Regli and M. Spagnuolo, "Introduction to shape similarity detection and search for CAD/CAE applications," *Computer-Aided Design*, vol. 9, no. 38, pp. 937–938, 2006.

[18] H. J. Nussbaumer, "The Fast Fourier Transform," in *Fast Fourier Transform and Convolution Algorithms*. Springer, 1981, pp. 80–111.

[19] L. Marple, "Computing the discrete-time 'analytic' signal via FFT," *IEEE Transactions on signal processing*, vol. 47, no. 9, pp. 2600–2603, 1999.

[20] M. Fürer, "Faster integer multiplication," *SIAM Journal on Computing*, vol. 39, no. 3, pp. 979–1005, 2009.

[21] W. Koenig, H. Dunn, and L. Lacy, "The sound spectrograph," *The Journal of the Acoustical Society of America*, vol. 18, no. 1, pp. 19–49, 1946.

[22] A. Wang *et al.*, "An industrial strength audio search algorithm." in *Ismir*, vol. 2003. Washington, DC, 2003, pp. 7–13.

[23] P. Du, W. A. Kibbe, and S. M. Lin, "Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching," *Bioinformatics*, vol. 22, no. 17, pp. 2059–2065, 2006.

[24] C. Rocchini *et al.*, "A low cost 3D scanner based on structured light," in *Computer Graphics Forum*, vol. 20, no. 3. Wiley Online Library, 2001, pp. 299–308.

[25] B. Starly, A. Bharadwaj, and A. Angrish, "FabWave CAD Repository Categorized Part Classes," 2019. [Online]. Available: http://rgdoi.net/10.13140/RG.2.2.31167.87201

[26] A. S. Master *et al.*, "Systems and methods for sound recognition," Mar. 8 2016, uS Patent 9,280,598.

[27] M. S. Lew, *Principles of visual information retrieval*. Springer Science & Business Media, 2013.

[28] J. W. Tangelder and R. C. Veltkamp, "A survey of content based 3D shape retrieval methods," in *Proceedings Shape Modeling Applications, 2004.* IEEE, 2004, pp. 145–156.

[29] N. Iyer *et al.*, "Three-dimensional shape searching: state-of-the-art review and future trends," *Computer-Aided Design*, vol. 37, no. 5, pp. 509–530, 2005.

[30] M. Ramesh, D. Yip-Hoi, and D. Dutta, "Feature based shape similarity measurement for retrieval of mechanical parts," *J. Comput. Inf. Sci. Eng.*, vol. 1, no. 3, pp. 245–256, 2001.

[31] A. Cardone, S. K. Gupta, A. Deshmukh, and M. Karnik, "Machining feature-based similarity assessment algorithms for prismatic machined parts," *Computer-Aided Design*, vol. 38, no. 9, pp. 954–972, 2006.

[32] A. Angrish, B. Craver, and B. Starly, "FabSearch: A 3D CAD Model-Based Search Engine for Sourcing Manufacturing Services," *Journal of Computing and Information Science in Engineering*, vol. 19, no. 4, 2019.

[33] P. J. Flynn and A. K. Jain, "BONSAI: 3D object recognition using constrained search," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 10, pp. 1066–1075, 1991.

[34] A. S. Deshmukh, A. G. Banerjee, S. K. Gupta, and R. D. Sriram, "Content-based assembly search: A step towards assembly reuse," *Computer-aided design*, vol. 40, no. 2, pp. 244–261, 2008.

[35] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin, "Shape distributions," *ACM Transactions on Graphics (TOG)*, vol. 21, no. 4, pp. 807–832, 2002.

[36] C. Y. Ip, D. Lapadat, L. Sieger, and W. C. Regli, "Using shape distributions to compare solid models," in *Proceedings of the seventh ACM symposium on Solid modeling and applications*, 2002, pp. 273–280.

[37] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[38] J. Bai, S. Gao, W. Tang, Y. Liu, and S. Guo, "Design reuse oriented partial retrieval of CAD models," *Computer-Aided Design*, vol. 42, no. 12, pp. 1069–1084, 2010.