1 Introduction
Textures are essential assets for 3D rendering, but they require a significant amount of time and artistic ability to create. Procedural texture generation has been a mainstay tool often reached for by content creators to mitigate this expensive task. These tools however, are art forms in and of themselves and require a substantial skillset to create. Making these resources available to a wider community requires an efficient way to catalog and search existing and newly created textures. To aid in such an endeavor, the ability to automatically classify and tag textures becomes an especially desirable feature.

Our project investigates automatic image annotation as applied to procedurally generated textures. Unlike other similar image classification projects, ours looks not at the features of the resulting image but instead focuses on the features used to describe its generation. By having the ability to classify and tag texture generation feature sets (called filters from here forward) we attain the ability not only to classify filters extremely quickly but we are able to classify them before they are rendered.

2 Background
Although there are several means of generating textures, one of the more common and most flexible is a node based approach. In this scheme, nodes are chosen to transform, generate, or combine some aspect of the image (color, height, alpha, reflectivity etc) and then chained together into a tree or mesh like structure. Fortunately, we found a popular texture creation tool called FilterForge™ that has open filter format and several thousand community donated filters.

FilterForge™ defines a collection of node types (on the order of fifty consisting of mathematical operations, perlin noise generators, ramps and patterns, and various transforms and blends). These nodes can be connected in a GUI and saved as a filter (a somewhat easy to parse xml file). As an example: to generate a liquid looking texture you may use a perlin noise node to specify the general shape and another its color and combine them into a single image.

An unbounded number of textures, rich in complexity and variety, can be described in such a manner. Although beyond the scope of this project, automatically tagging textures based solely on the nodal attributes in the filters hints at exciting possibilities of employing genetic algorithms to evolve textures in an evolutionary manner without having to render them (which currently is very time consuming).

3 Data Collection
We began by downloading and rendering the entire community library of over 4000 filters available for the FilterForge™
application. We created a website and database to help parse the filter files, randomly generate the nodal parameters, and then invoke the command line renderer. In concert with rendering, we extracted the chosen feature set from the xml file, normalized, and stored in our database.

**Feature Selection** FilterForge™ allows for an unlimited number of nodes each with a wide variety of attributes and user tweakable controls. Choosing (and extracting) features proved to be both the most challenging and the most critical aspect of the project. In the end, we utilized a method similar to that of text classification in that we began with a simple count of each node type in the filter. We then expanded the features to include rendering hints, average color, and some seemingly important features of the more common node types. Our final feature set is as follows:

- the total number of nodes
- the total number of each type of node for types:
  - Color (RGB)/Gradient
  - Perlin Noise
  - Curves (levels, shapes)
- average roughness/perturbation of noise
- average surface color
- lighting
- Reflectivity/metalllic measures

These features were the most consistently varied and valued (not just 0) among the textures. In addition, we felt that they were a good approximation for the level of complexity of each texture and averaged contributing values well.

**Annotations.** Filter classification was done in two manual stages. First, we simply segregated textures into three primary categories: Organic (defined as useful for texturing things which are not man made; stone, earth, grass, dirt, etc), Artificial (things such as cloth, metal, upholstery), and finally Other which were textures that did not appear to be useful for applying to 3D objects.

In all, we ended up with about 1550 organic, 1172 artificial, and 1600 other. Because the next stage was to manually tag each texture with a short description we selected only the Organic textures for consideration. However, we used these overall classifications in our analysis in order to determine a baseline feature-set to use.

We then used our web interface to tag each image. We entered 2-6 word descriptions for the first 1200 organic filters, using adjectives like colors or textures, and types of objects, such as "stone", "wood", or "liquid". These tagged feature sets were used as our primary dataset for training/testing our classifiers.

### 4 Methodology

We approached image annotation as a multi-label classification problem, in which input \( x \in X \), the set of all input texture images, is mapped to \( n \) labels from a set of disjoint labels or tags \( T \). This is an expansion on a multi-class classification, which takes training input marked with one label \( t \in T \) where \( |T| > 2 \), and assigns test input each with a label from \( T \) - the common binary classification model is a specialized case for \( |T| = 2 \) and \( n = 1 \). We thus could divide our problem into smaller multi-class problems, and furthermore into binary classification problems for simple but effective modeling. We used a One-vs-All classification method.

![Figure 2: Organic, Artificial, Other filters (left to right)](image1)

![Figure 3: Examples of tagged filters](image2)
Create \(|T|\) binary classifiers, one for each tag \(t \in T\), and divide the dataset into \(|T|\) label sets of length \(|X|\). For each tag, the set of labels has positive labels for images with that tag, and negative for ‘all’ the other tags. We applied these methods to a simple multiclass problem and then a multi-label problem in succession:

1. Classify an image as ”Organic”, ”Artificial”, or ”Other”
2. Assign a set of tags to each image.

5 One-vs-All

5.1 Multi-class, Single Label

Our first task was to look at the dataset as a whole and our initial classification of Organic vs Artificial. This set was used to determine the feasibility of our feature set and the project as a whole. Utilizing libsvm\[6\], we trained larger and larger sets and averaged training error per label classifier.

![Figure 4: Organic/Artificial/Other - Average Training Error Per Classifier](image)

Our training and test errors appeared to trend towards convergence (slowly) indicating our SVM model was feasible. However, our accuracy was lower than desired as was our learning rate leading us to reevaluate and eventually select a broader feature set. In the end, our feature set provided convergence approaching 75% accuracy which was borderline acceptable. Unfortunately, it also showed that we would have trouble getting accurate predictions on our more specific classifications as the number of textures in each classification was much lower than this general taxonomy.

5.2 One-vs-All - Tagging Images

We then applied our model to the broader set of classifications we created and the same SVM. We first define a common set of metrics\[1\] to describe the accuracy of our multi-label classifier:

- **Precision:** the percent of true positives divided by all positive labels guessed
- **Recall:** the percent of true positives divided by true positives and false negatives
- **Hamming Distance/Error:** The percent of incorrect labels assigned (True/All Assigned) - smaller values are better

Initially, we had very low accuracy - either our classifiers were very inaccurate or had a deceivingly high precision but also high Hamming error, as our classifiers were labeling every image as 0 for the less common tags and assigning most of them to the highest-count tag. To compensate, we weighted each classifier by the percentage of images it labels. This parameter adjusts the cost for mislabeling for a particular tag. We saw an expected drop in accuracy and recall as more labels were assigned, but our average Hamming Distance is at a very low 11.1429%. We used N-Fold cross validation with \(N = 50, 100, \) and \(200\) to examine our model and averaged an average accuracy of 75-80% each time. For precision/recall/Hamming Error we trained on 30% of the class and ran our results on the other 70%. Measurements we took are summarized below.

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>Hamming Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM, unweighted</td>
<td>95.1243</td>
<td>60.326</td>
<td>43.021</td>
</tr>
<tr>
<td>SVM, weighted</td>
<td>82.8683</td>
<td>53.39</td>
<td>11.1429</td>
</tr>
</tbody>
</table>

![Table 1: Comparison for Tag-Classification](image)

We also analyzed each classifier for our most frequent tags to determine how well our model fit data, as these tags would be the most likely to emulate real performance of our model on a larger dataset with more of each type of tag.

![Figure 5: Accuracy on the ten most common tags](image)
6 Auto-Tagging Results

The final output of our program was an auto tagging feature implemented in PHP which annotated filters with tags predicted by the model. Some of the results are displayed above with their generated tags.

As you can see, the tags it applies are mostly relevant. They even appear to be more correct than the accuracy indicates. This is likely due to the subjective nature of tagging and that we generally only notice incorrect tags and not the absence of correct ones.

Tags that performed well were not surprising. Colors, as their features were easy to pin down, as well as shiny and dark for similar reasons. What is promising is that bumpy and rippled appear in germane locations lending weight to the validity of the scheme. It should be noted, that the current tagging scheme has a propensity for stone.

7 Improvements

No attempt was made to distinguish between nodes that controlled surface features such as height and reflectivity and those that controlled color. This distinction had a dramatic effect on the visual aesthetics of the resultant image. We believe including this distinction would add the most value to our model. Parsing complexity hindered our efforts in this regard.

No attempt was made to map out relationships between nodes. Features such as connectivity between nodes, branch depth, and fanout warrant exploration but would require substantially more effort in XML file parsing.

8 Conclusion

We judge our results to be moderately successful. In this case of image classification we are attempting to
judge things that not only have not been rendered but are not images of objects with strictly definable features. Our results demonstrate that convergence to an 80% solution, given enough data and enough control over the feature set, should be achievable. While this is a low number for many applications it is quite adequate for the largely subjective nature of this problem.

References

