Adaptive Supersampling Using Machine Learning Techniques

Kevin Winner
winnerk1@umbc.edu

Abstract

Previous work in adaptive supersampling methods have utilized algorithmic approaches to analyze properties of the object space or image space which might benefit from increased supersampling. These techniques generally increase the computational complexity of simple supersampling, reducing the potential performance gain from employing adaptive supersampling. In this paper, we describe an experimental technique which creates a learned model of the adaptive supersampling problem in a given domain using techniques from the field of machine learning. This model can then be used to quickly and efficiently compute a distribution of samples in the adaptive supersampling problem.

1 Introduction

Supersampling techniques are simple yet powerful tools for performing antialiasing in ray tracing by increasing the sampling resolution of the image. Each pixel of the image is divided into many subpixels, which are each sampled independently. These samples are then combined to produce a single color which is used for the final pixel value rendered.

Unfortunately, supersampling techniques require significant additional computation for every pixel in the image. In order to sample at a resolution fine enough to properly antialias the complex regions of the image, it is also necessary to sample the simplest regions equally. Since additional samples are not necessary in these regions, a great deal of computation time is wasted sending rays through these extraneous samples.

This leaves us the problem of deciding which samples are necessary and which can be skipped. Solutions to this problem are known as adaptive supersampling techniques. This has been a productive area of research, the majority of which fall into one of two categories: image space strategies and object space strategies. Image space strategies use information about pixel colors and image information to select samples, while object space strategies analyze the scene and possible near-misses of objects by rays.

Most image space techniques suffer from one of two problems. They can require a user to tweak a threshold or similar variable to control the sensitivity of the algorithm. This threshold is often difficult to determine efficiently and can vary from one scene to another. Additionally, the ideal threshold might vary across the image, which is difficult to control. Image space techniques can also differ significantly from the ”true” image which would be produced by standard supersampling at the same depth as the adaptive method.

In this paper, we present an experimental image space technique which selects samples based on an observed history of which types of samples actually yielded changes to the final pixel value. We represent this history using a classifier constructed using machine learning techniques. The classifier is constructed using a large collection of samples from the scene as training data. Once constructed, the classifier represents an approximate codification of the knowledge gleaned by performing the complete high-depth supersampling of the image.

With a learned classifier, we are able to query the classifier correctly about the expected impact of a potential sample. This query does not require any additional data collection and is exceptionally quick when using the methods we describe here. The classifiers we construct are able to accurately predict as many as 99.9% of the samples in our test scenes, in a fraction of the time taken by the more thorough supersampling techniques.

Section 2 describes relevant related work in more detail. Section 3 presents the details of our adaptive supersampling method. Sections 4 and 5 include and discuss the results of our experiments with this technique. Section 6 shows the next steps for research with this technique.

2 Related Work

Adaptive supersampling in image space has been a popular technique for easily and quickly performing adaptive supersampling. The image space technique was introduced by Turner Whitted, who used what is now referred to as an image space approach to adaptive supersampling to reduce the cost of supersampling. [Whitted 1980] They cast four rays into the corners of each pixel. If the colors returned from each of these rays does not differ by much, then no more sampling is performed. Otherwise, the pixel is subdivided and more rays are sent out. This technique typically had many problems with missing small objects or shadows which did not cover at least one corner of a pixel.

Recently, there have been a number of papers on improving the image space approach to adaptive supersampling. The primary research focus has been on intelligent strategies for deciding whether to subdivide a pixel. In 2002, Jaume Rigau et al. developed a technique which characterized the contrast of a pixel with its neighbors in order to determine important boundaries. [Rigau et al. 2002] In 2003, they refined this idea to represent the entropy of a subpixel and the information gain of a subdivision. [Rigau et al. 2003]

3 Implementation

Our system presents a two-pass implementation of a machine learning adaptive supersampling solution. The first pass of the system generates the data necessary to build a machine learning model using a standard supersampling implementation. Once the data has been collected, we construct a predictive model (or classifier) using machine learning techniques. This model takes as input for an instance the vector of colors of each of the sampled subpixels and predicts the probability that the final color of that subpixel would change by recurring again. With this model constructed, we run a second pass of the renderer which uses an adaptive supersampling strategy defined by the predictions of this model. In the following

subsection we go into further detail about each of the steps of the implementation.

3.1 Collecting image space data

Learning a classifier for machine learning requires a large collection of labeled training data. This means producing data which has all the attributes we expect to have available when running the system online as well as correctly labeled classes which represent the answer to the question we are asking the model in the online case. Since we are trying to approximate a supersampling solution with very high depth, we use a pass of this high depth solution to gather our data.

In our case, we used an open source ray tracing implementation which included supersampling options called Sunflow which was written by Christopher Kulla and is available under the MIT license. [Kulla] Sunflow implements a recursive solution to supersampling wherein each pixel is divided into 4 subpixels. Each of these subpixels is then sampled at their centers and recursively subdivided until we reach the target depth, at which point the samples are then aggregated into a single color and passed back up the recursion tree.

At each pixel (or subpixel) then we construct an instance of the training data based on the following data we have available before we divide and recur on the subpixels: the current recursion depth and the colors of each of the 4 initial samples of the subpixels. We also compute the color which the pixel would have been if we had not subdivided it and the color which resulted from that subdivision. This color difference is attached to each instance and used as the class label for that instance.

3.2 Creating a classifier

Once the training data has been collected, we can use it to build and train a machine learning classifier. Our implementations of the various machine learning algorithms described here were borrowed from Weka, an industry standard machine learning toolkit with many contributors. [Mark Hall 2009] There are many classifier types available, but we have selected two commonly used algorithms: naive Bayes and C4.5 (a decision tree algorithm).

Naive Bayes is a probabilistic model which learns probabilities for the possible class labels based on a derivation from Bayes theorem. The implementation of naive Bayes in Weka builds a model based on the maximum-likelihood process where the model is refined until it makes correct predictions on the training data as often as possible. The drawback to naive Bayes is that it assumes that all the variables in the data are independent of each other, which is clearly not true in our case. However, research has shown that in spite of the simplifications naive Bayes makes, it very often performs surprisingly well [Zhang 2004], which is why we chose it as one of our algorithms.

The other algorithm we use is J48, which is an implementation of the C4.5 decision tree algorithm included in the Weka toolkit. Decision trees are binary trees where each node in the tree filters each instances into one of two subtrees. When the instances reach a leaf node, they are assigned a class. Each non-leaf node selects instances based on a single variable, based on the principle of information gain for that variable.

Our application uses mostly default options for the Weka implementations of these algorithms, as the defaults are generally sufficient for our purposes. When test data was needed (in some of our experiments), we used a hold out set from the training data as the training data was a complete sample and additional unlabeled data was not available.

3.3 Supersampling with a classifier

In our recursive subdivision implementation of supersampling, the switch to adaptive supersampling comes from sometimes choosing not to subdivide a pixel which is unlikely to produce a noticeable effect on the final image. The machine learning classifiers we construct are built to predict which pixels will need to be subdivided.

Each top-level subpixel will always be sampled at least 4 times. The original 4 samples will be used as input to the machine learning classifier, which predicts a probability that the resulting change in color of the pixel will be above a certain threshold. In our case, the threshold is user configurable, but we found that low values between 0.01 and 0.1 typically produced better models. In this case, the threshold is the total absolute difference between the RGB components of the colors, represented as floating point values from 0 to 1.

4 Results

We evaluate the results of running our system using both naive Bayes and decision trees on supersampling depths of 1 and 2. Due to memory limitations with our testing equipment, we were unfortunately unable to construct training datasets when the supersampling depth was any larger than 2. We show the accuracy of both these methods using cross validation of the classifiers on the training data. We also provide a sample image produced using each of these methods.

For each combination of machine learning algorithm and training dataset, we also have both a high accuracy and low accuracy model. The low accuracy model predicts whether instances will have a total change over 0.1 as a result of subdivision and the high accuracy model predicts whether those instances will change more than 0.02, a harder problem.

For reference, all of our experiments were performed on the scene gumbo_and_teapot which is available from the Sunflow project as part of their example package. It consists of many overlapping objects and shadows on a flat background. In our sample images, we have cropped the image to a single teapot from the center of the scene to highlight the effects of supersampling.

In the figures that follow, NB stands for naive Bayes, DT means decision tree, and NB+/DT+ are the results from reducing the threshold to 0.02 from 0.1.

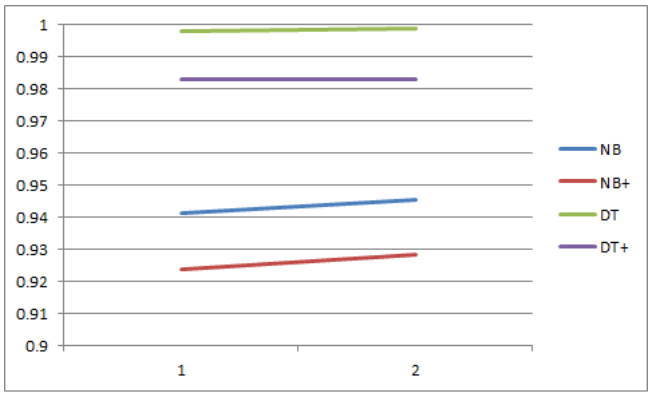

Figure 1: *The results of the cross-validation test*

Figure 2: *No supersampling*

Figure 3: *Supersampling depth 1*

Figure 4: *Supersampling depth 2*

Figure 5: *NB Supersampling depth 1*

Figure 6: *NB+ Supersampling depth 1*

Figure 7: *NB Supersampling depth 2*

Figure 8: *NB+ Supersampling depth 2*

Figure 9: *DT Supersampling depth 1*

Figure 10: *DT+ Supersampling depth 1*

Figure 11: *DT Supersampling depth 2*

5 Analysis of Results

Our cross validation test showed several interesting results. One of the most striking results is that the decision tree methods had excellent results in every case. This means that adaptive supersampling with this method should produce nearly identical pictures to the equivalent supersampling technique, while requiring many fewer samples during runtime.

Another notable result from the cross validation test is that naive Bayes performed significantly worse than the decision tree methods. This is most likely a result of the assumptions made by the naive Bayes learning algorithm that all the variables are independent and distributed according to a gaussian distribution.

6 Future Work

The major limitation of this architecture is the prohibitive cost of collecting the requisite training data. Collecting all the training data requires a process which takes longer than the render itself. The next step for this research is certainly to address this issue, and there are several potential avenues for doing so.

The first approach would be to improve the process of data collection by reducing the number of samples needed to build an effective classifier. This can be done by reducing the depth of the sampling process used to collect the data, but this presumably has an effect on the accuracy of the resultant classifier. Other methods might include an intelligent sampling solution or an aggregation strategy, but many of these problems are analogous to the original adaptive supersampling problem.

Another approach would be to improve the portability of the classifier. The collection of data is an expensive step, but only needs to be done once for each model. Experiments with cross-training classifiers on similar scenes might show that a single model can be used for many views of the same scene.

Incorporating object-space knowledge into the classifier would also potentially improve portability. This could also make the model applicable to real-time systems, but would require a representation of the relevant object-space knowledge which could be used as input for a classifier.

A final approach would be to perform online learning of the model. In this situation, the classifier would be constructed and adjusted in real time as the render was completed. Unfortunately due to the nature of this approach, aliasing artifacts would appear at the borders of regions which the classifier had not previously sampled. These regions would likely be either oversampled or undersampled

until the model had been updated to handle them correctly, possibly leading to visible artifacts.

In the future, we also hope to generate more experimental results which examine the impact of cross training on multiple related or unrelated scenes. We would also like to conduct similar experiments to the ones included in this paper, but using much higher supersampling depths than we were able to produce with our current equipment.

7 Conclusion

We have presented a new technique for performing adaptive supersampling which has the potential to be both cost-effective and accurate. While the model is expensive to construct, queries are quick and informative, creating a very powerful tool. Additionally, once constructed, the classifier can be reused many times, distributing the cost of building the model.

We consider this paper to be an important building block for other techniques which further extend the concept of combining machine learning algorithms and supersampling strategies. Our results demonstrate that this technique has significant potential, but there are still open problems in the way of its adoption, which we outlined in the previous section.

References

KULLA, C. Sunflow.

MARK HALL, EIBE FRANK, G. H. B. P. P. R. I. H. W. 2009. The weka data mining software: An update. *SIGKDD Explorations 11*.

RIGAU, J., FEIXAS, M., AND SBERT, M. 2002. New contrast measures for pixel supersampling. In *PROCEEDINGS OF CGI'02*, Springer-Verlag London Limited, 439–451.

RIGAU, J., FEIXAS, M., AND SBERT, M., 2003. Entropy-based adaptive sampling.

WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM*, 343–349.

ZHANG, H. 2004. *The Optimality of Naive Bayes*.