Procedural Tree Generation

Compressing 3D tree for faster rendering

Sebastian-Alexandru Manda

Supervisor(s): Prof. Dr. Elmar Eisemann, Dr. Petr Kellnhofer, Lucas Uzolas

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 28, 2024

Name of the student: Sebastian-Alexandru Manda
Final project course: CSE3000 Research Project
Thesis committee: Prof. Dr. Elmar Eisemann, Dr. Petr Kellnhofer, Lucas Uzolas , Dr. Marcel Reinders

Abstract

Trees are essential components of both real and digital environments. Therefore, it is important to have 3D models of trees that are of high quality and computationally efficient. One way to achieve this is by compressing a high-quality model using billboard rendering, which involves partitioning the tree into multiple planes to produce a similar result to the original. Our study explores the compression of 3D models using an optimization loop and adapting billboard rendering techniques. We use computer vision primitives to render basic models, which we then optimize by adjusting the texture to resemble the original tree. The models consist of multiple upright planes that are rotated around the central vertical axis of the original tree. We use different optimization functions, such as L1 and L2 losses, to determine the best approach. We can improve the initial models by bounding the billboards and limiting their heights and widths to that of the trees. Additionally, we can use double-sided textures for the billboards to allow more flexibility for optimizing different species of trees. However, optimizing multiple tree types performs differently for each species, leading to improvements that only benefit certain trees in specific scenarios. Using quantitative metrics, we determined which models perform best and how similar they are to the original after training. We found that our compressed models generally resemble the original while having only a fraction of the original size.

1 Introduction

Rendering tree models quickly has been a topic for discussion since 3D rendering appeared. This is due to the way 3D models are created and rendered traditionally. All parts of the model are represented by polygons, mainly triangles, meaning that large volumes for models can be rendered quicker by lowering the number of triangles that compose the model. However, for trees, rendering the leaves becomes the main issue, as they do not represent a full volume but rather a lot of small triangles for each leaf of the tree. Seeing as trees usually have thousands of leaves in their crown, it becomes a thorough process to calculate and render them when taking into consideration light layering, shading, specular refraction, and many more for each of the leaves.

This is why many methods [1; 2; 3; 4] were devised to try to lessen the process of rendering the trees along with their leaves. The first and most direct approach is to modify the process of rendering these models and try to save computation power in that domain, however, due to the time constraints of this project, this is not a feasible option. The second approach is trying to compact or reduce the initial model into something easier to render while keeping the final result as similar as possible. This leads us to the main question this research is trying to answer:

How to simplify a complex 3D tree into a more compact representation for fast rendering?

To answer this question we have chosen to adapt the methods of slicing and blending trees [1] and billboard clouds [2] into an optimization loop-based algorithm. Billboard clouds partition the model into many planes, also called billboards, at different angles, sizes, and locations to render the final tree as this removes the need to render triangles for each leaf. The slicing and blending approach renders cross-sections of the tree model and uses them for the final model to allow depth and volume to be created through layering without having to use the initial model data of the leaves. Both approaches are, however, an approximation of the original tree as they lack some of its content.

Although the method described so far has already been implemented before, there are few to none that have tried using differentiable rendering [5] for optimizing the billboards that compose the compressed model of the tree. Our approach will use the same concept of billboards, but where the other implementations have applied a rendered image of a slice or from a specific perspective of the tree as the texture of billboards, we will use an optimization loop to create the textures needed by the compressed model. We also intend to find which optimization function is better for this approach, L1 loss or L2 loss. With this in mind, we can split the main question into the following sub-questions:

- How does the number of billboards affect the results?
- How do different species of trees affect the results?
- Is L2 loss better than L1 loss as the optimization function?
- How does having different textures on the sides of the billboards instead of a single texture on both sides affect the results?

To this extent, Sections 2 and 3 will explain some of the related work in this aspect and provide the background necessary for the rest of the paper. Details of our implementation will be found in Section 4. Section 5 will provide a guide for installing and running the experiments while Section 6 will describe the results our experiments have offered. Section 7 discusses further work and general results that answer our sub-questions, after which Section 7 provides details about responsible research for the aforementioned experimental setup. Finally, Section 8 will denote our conclusion for this research.

2 Related Work

Compressing 3D models is universally useful as it allows for faster transfers or processing models at lower qualities if necessary. This is why many methods have been found for this exact purpose.

Standard model compression

Standard compression is the simplest form of reducing both storage and render time for a model. This is done by reducing the number of vertices of a model while keeping its overall shape. Smoother 3D models contain more vertices to

replicate shapes like spheres, but these can usually be reduced to lower polygon structures instead. **Draco** [4] is one of many tool libraries for compressing and decompressing as described. They use a wide variety of tools and methods for compression in C++ and JavaScript. Similarly, **Shape-Adaptive Wavelet Coding of Multi-Height Fields** [6] is a technique used for compressing 3D models or point clouds by partitioning the surface of the model into many patches and then encoding these patches into a compressed data type that can accurately represent the initial surface of the partitioned region of the model.

Model training

Training the model to resemble the original is another approach that could be used as compression. Methods such as **NeRF** (short for Neural Radiance Fields) [7] or **NvDiffRec** [3] follow this approach. **NvDiffRec** is a library that takes a 3D model and attempts to train a new model with the same shape and texture as the original. This is done over many iterations by adjusting the optimized model based on the mean squared error between the original and optimized model. After many of these iterations, a model that resembles the original is formed. Each step in this process can be considered a crude version of the original, being less complex, yet trying to represent the same thing. **NeRF** is a neural network trained to produce a single object or scene. After training, it uses the neural network to produce a rendered image faster than rendering the initial model. Both NVDiffRec and NeRF use differential rendering, being the inspiration for our adaptation of the last type of approach.

The main similarity in all methods or tools mentioned so far is that they are optimized for models with continuous and connected surfaces i.e. not many holes present throughout the model. Trees, however, do not share these attributes. They have many vertices that represent small bodies such as leaves and branches, having a lot of space between them, thus making the methods mentioned above not a very reliable option for our task.

Billboard based rendering

This leads to the final approach, creating a billboard-based model of the tree. **Interactive Vegetation Rendering with Slicing and Blending** [1] is the method of slicing the vertices of the tree into sections, then individually rendering each section and using these images as the model of the tree. Doing this on multiple angles around the vertical axis of the tree gives the final model for this method. **Realistic real-time rendering of landscapes using billboard clouds** [2] tries to optimally partition the initial model into billboards.

Then, a texture with the content of the vertices within the partition is applied to the billboards. Each of them could represent a branch and its leaves or sections of the main crown of the tree and its trunk, together forming the total model of the tree. The methods explained in this paragraph have the best outcomes when it comes to rendering models of trees specifically since they were built with this purpose in mind, unlike the aforementioned ones. However, these methods may be further compressed.

3 Background

This research is heavily focused on adapting the different approaches of billboard rendering and model training. As such, the most relevant terms of computer graphics and computer vision will be detailed in this section.

A 3D model is formed by many points in a 3D coordinate space; these will be referred to as **vertices** from now on. To create a surface, 3 vertices are connected creating a triangle or a **face**. The faces are used to determine the shape of the model. To apply a texture to a model, a **UV map** is necessary as it maps the vertices to a location on a 2D texture. Interpolating this over the many faces of the model places the texture onto a model. All the attributes mentioned so far will also be referenced as the $model$ together.

NvDiffRast is a collection of computer vision primitives that allow the creation of a very complex rendering pipeline through the use of its modular methods. These methods split the task of rendering a 3D object into multiple sub-steps such as rasterization, interpolation, texture sampling, and anti-aliasing.

4 Method

To answer our research question and sub-questions, we will take inspiration from the model optimization approaches [3; 7] and the billboard compression approaches [1; 2]. As the input, we will use an already existing 3D model of the tree. This model will be rendered externally to create a set of images that will be used as our ground truth. Starting from either a preselected model or a default model, we will randomly select angles at which to render our optimized model. Loss is determined between the rendered image and the ground truth for the same angle, which is later used to optimize the texture that wraps the optimized model. Each step of this process is further explained in the following subsections.

4.1 Dataset

As detailed in section 3, NVDiffRast provides primitives for creating our rendering environment. However, this environment does not provide shading and needs to be added manually. To circumvent this, we have decided that instead of rendering the original tree models in the NVDiffRast environment, we will do so externally and then use them as our ground truth.

Blender [8] has been chosen for this purpose. Given a model of a tree, we create a scene around it, placing the camera at an appropriate position, adding ambient light, and rendering an image from that perspective. After the render is complete, we move the camera to the next position around the tree and repeat until the desired number of data points is collected. This process is automated and can return any number of images around the tree at any desired angle interval with any resolution. These images will form the training and testing sets used for the optimization model. Each image is taken at an angle θ from the initial viewing angle 0. This will be useful in forcing the perspective of the ground truth onto the optimized model.

We have chosen 3 tree types from the tree model generator library of Friggog [9]. The species are Quaking Aspen, Willow, and Acer, which can be seen in Figure 1 below. These trees will be used in all the experiments in our evaluation.

Figure 1: Tree models used in experiments, externally rendered in Blender.

4.2 Model and Initialization

As mentioned so far, instead of creating a model resembling the original in shape, we have decided to use billboards as the base for our textures. This means that we focus on optimizing the texture on a starting model by rotating its vertices and then rendering it from that new angle to compare it to the ground truth. This means that we wish to create a simple starting model that can resemble the original model from any angle once a texture has been applied.

One of the simplest 3D shapes is a plane, referred to as a billboard throughout this paper. However, a single billboard can never represent a 3D object when viewing it from different angles. By adding billboards rotated around a vertical axis (y-axis), we can create a model that, once textures are mapped on its surface, may resemble the original. One such model can be seen in Figure 2.

Figure 2: Simplest model: 2 billboards at a 90° angle.

Billboards are 4 vertex planes in a 3D space. The space that is rendered within NVDiffRec is a 2 units length cube with its center and origin at $(0, 0, 0)$, meaning that opposing corners are found at $(-1, -1, -1)$ and $(1, 1, 1)$. For our research, we will be limiting our models to be composed only of vertical planes, i.e. planes that are rotated around the vertical axis (y-axis). This will be the axis of rotation for any future rotation angles.

To create our models, we define a list of billboards, each facing a certain angle. The number of billboards used for the models is arbitrary, starting from the minimum of 2 billboards at a 90° angle (a cross viewed from above) to 3 and 4 billboards uniformly spread apart.

At the beginning of the process, all desired planes are created, rotated at their respective angle, and merged into a whole. A texture for the size of the model is also defined. This texture is optimized to create our final product.

4.3 Rendering Pipeline

Rendering the optimized model is the second most important aspect of our implementation. The entire pipeline can be seen in Figure 3, but we will further explain each step in this subsection. All green steps in the figure represent primitive functions provided by NVDiffRast [5].

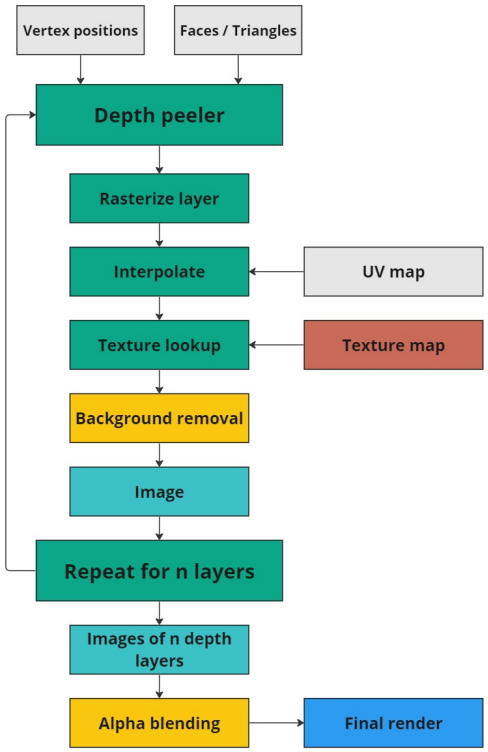

Figure 3: Rendering pipeline. Green steps are primitives provided b NVDiffRast, Gray steps are initial inputs (they do not change throughout the optimization process), Red steps are optimized inputs such as the texture of the model (these inputs change for every step of optimization), Yellow steps are processes implemented we implemented, and Blue steps are results (Light Blue is intermediary and Dark Blue is final).

The **depth peeler** is the first step of rendering. It is a primitive function provided by NVDiffRec that, given the vertices and faces of a model, allows us to rasterize each depth layer of the model. Each depth layer is determined by the number of surfaces on the model that need to be hit for a certain pixel to be shown on the rasterized result. This means that the first layer shows the first surface of the model for each pixel hit. If no surfaces were hit for a pixel, it is empty. This goes on for as many iterations as necessary, to reveal all depth

layers of the model for that angle. The purpose of the depth layer is to create transparency within the rendered image of the model. If the texture applied to the model is transparent, the depth peeler reveals all layers of surfaces that may have been hidden by only rasterizing the first surface.

Once we have the rasterized output for a certain depth layer of the model, we use **Interpolation** along with the UV map of the model to map each visible point on a surface of the model to a coordinate in the texture image.

Using the interpolated image, we can begin with the **texture lookup** step. It applies the color of the pixel on the texture to the pixel in the final image. Because not all pixels in a layer hit a surface, they can be empty. Each empty pixel is given the color of the texture at coordinate $(0, 0, 0)$. Because this can be a random color depending on the texture, we set all pixels in the background, i.e. all pixels that did not hit a surface on the model, to $(0, 0, 0, 0)$, thus **removing the background**.

Once all depth layers have been individually rendered, we use **alpha blending** to combine the pixels from each layer into a single image. Besides the RGB values of the pixels in the texture, they also have an alpha value. The alpha value determines how much of that pixel is visible, allowing the color of the pixel behind it to be seen as well, thus creating transparency. Because the images we output in the data collection step are saved with the PNG extension, we are using straight blending for the alpha value. The resulting image is the final render from the selected angle, which is then compared to the ground truth.

This pipeline can be abstracted to a function f which takes the $model$ and the angle θ for the perspective of the camera around the origin, resulting in the rendered image of the optimized model: $I_{opt} = f(model, \theta)$.

4.4 Optimization Loop

We use Algorithm 1 for our optimization loop. Because we render the ground truth model externally, we will be randomly selecting an angle from the training set to train on. We first use a rotation matrix to rotate the vertices to the angle we are supposed to be viewing from, after which we use our rendering pipeline $f(model, \theta)$ to get the final result. This image is then compared to the ground truth with the same perspective and loss is calculated. For the experiments within this research, we calculate either L1 loss

$$l1_{loss} = |f(model, \theta) - I_{gt}|$$

or L2 loss

$$l2_{loss} = \frac{1}{p} * \sum (f(model, \theta) - I_{gt})^2$$

where I_{gt} is ground truth image and p is the total number of pixels in one of the images. Only one of the loss functions is used in the training of an experiment, never both. When they are both showcased, two separate experiments have taken place. The **Adam** [10] algorithm optimizer is used with an initial learning rate of $lr = 0.01$ that increases during the optimization process. The change in lr is defined as

$$lr_i = 0.1^{\frac{lr_{i-1}}{n}}$$

where n is the total number of iterations and i is the current iteration.

Algorithm 1 Optimization Loop

1: **for** $i = 0, 1, \ldots, n$ **do**
2: Select an angle $\theta = [0, 360)$ from the training set
3: Render optimized model $i\_opt = f(model, \theta)$
4: Fetch ground truth i_gt for the selected θ
5: Calculate either $l1_{loss}$ or $l2_{loss}$ between i_opt and i_gt then propagate backwards
6: **end for**

5 Implementation details

This study used a conda [11] environment as well as CUDA 12.3 [12] to execute all experiments. More information on installation steps can be found in the Readme for the repository of this research [13].

For the 3D models of the trees, we have used the blender plugin Tree-gen [9]. From the different species it provides, we have chosen the Quaking Aspen, the Willow, and the Acer types. The ground truth datasets for each of these models were rendered in Blender [8] at a resolution of 1024×1024. The datasets we created for the Quaking Aspen and Acer have 360 renders around the origin, while the Willow has 180 due to its high vertex model. These images represent an angle θ around the center of the trees that will be used to match the perspective of our optimized model. Moreover, each tree dataset is randomly split into a training set and a test set, where the test set contains $\frac{1}{4}$ of the total samples for that tree. The test set is only used for evaluation, providing the metrics of the experiments.

Our experiments use NVDiffRast [5] primitives to render the optimized model. These images are also rendered at the same resolution of 1024×1024 as the ground truth datasets. Each optimization session for the experiments is run for $n = 10,000$ iterations.

6 Evaluation

In this section we describe the experiments that took place, the results they have provided, and how that created a direction for the optimizations added along the way. After training ends for each of the experiments, we render the angles sampled for the test set of the ground truth. The test set contains $\frac{1}{4}$ of all samples and was not used during the training of the model. All renders are evaluated with 4 metrics, different from the L1 and L2 pixel losses used in the training, and then averaged for the entire test set for the outcomes. These results can be found in Appendix A and are defined as:

- Root Mean Squared Error (RMSE) measures the magnitude of change per pixel between the two images.
- Structural Similarity Index Metric (SSIM) calculates the difference or loss in quality between the two images.
- Peak signal-to-noise ratio (PSNR) also calculates the quality difference between the two pixels, however, this difference is represented in decibels.
- Signal to Reconstruction Error (SRE) mostly measures the difference in brightness or signal power for each pixel of the images. It is also represented in decibels.

6.1 Base models and billboard count

Our first experiment uses the same model as the one described in section 4.2 and in Figure 2. It is the lowest vertex count model that a 3D model can be reduced to. However, in our study, we intended to find out what the impact of more than 2 billboards is on the results. As such, we have also been performing our experiments with models of 3 and 4 evenly spaced billboards, $60°$ and $45°$ respectively. These models can be seen in Figure 4. This experiment was performed using L2 as the loss function.

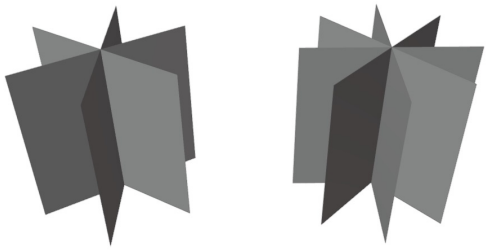

Figure 4: Additional experiment models with 3 billboards and $60°$ between them on the left, and 4 billboards and $45°$ on the right.

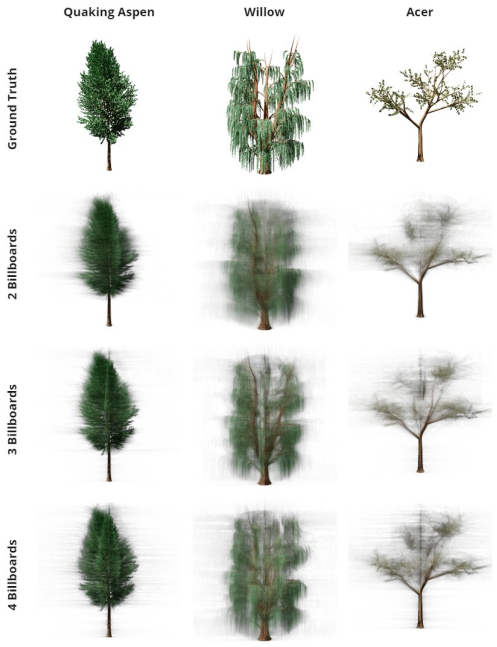

Figure 5: Rendered results with different numbers of billboards for each tree species.

Results

Figure 5 shows the change of the models when adding more billboards. We can observe that the optimized models become clearer the more billboards the models have. Because all billboards have the same texture on both sides, more billboards allow the model to have textures more closely related to the perspective it is viewed from. This provides an advantage over fewer billboards for the model and can be seen in the numerical results as well, where all metrics show better results.

This is not the case, however, for all trees. While the Quaking Aspen's and Willow's results are better with more billboards, the Acer has its best results in the 3 billboard model, signaling that for some types of trees, with less symmetry and sparser leaves and branches, a higher number of billboards is not directly beneficial.

6.2 L1 versus L2

As seen in the results of the previous experiment, optimizing with L2 leads to blurry, out-of-focus images of the tree. While these images do resemble the original, even by looking from afar they can be clearly distinguished from the original. For this purpose, we have also taken into consideration L1 as a possible loss function.

Results

The difference between using L1 and using L2 is simple. As Figure 6 shows, where models optimized using L2 are blurry, the models that use L1 are much clearer, having sharper edges and a clean background. Although it gives great visual improvements, it is not without its drawbacks. Firstly, the model is much darker than the original. Secondly, most of the empty gaps between the leaves and branches of all models are lost. While this happens in L2 models as well, it is not as pronounced. Finally, in contrast to the clearness of the renders from a viewer's perspective, the results show that the L1 models underperform on all metrics for this experiment.

Because results are mixed when it comes to which loss function is better, we have rendered the experiments that follow with an L1 variant as well. This has shown improvements for further experiments.

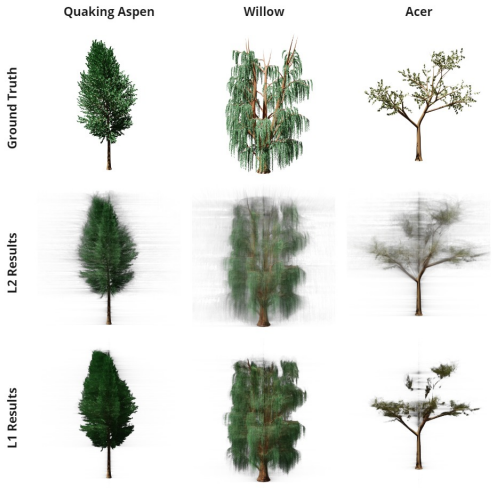

Figure 6: Rendered results of 4 billboard models of all species for L1 and L2 losses.

6.3 Bounded billboards

One disadvantage of L2 is the blurred edge of the model. A solution could be to only make the billboards that make the model as large as necessary for that perspective of the model. When initializing the model, instead of defaulting to cover the entire viewport with each billboard, we can take the ground truth for that angle and find the boundaries of all nontransparent pixels (alpha values of the pixels are above 0). These boundaries can then be translated to 3D coordinates for the boundaries of the billboard and also for the UV map of the available optimized texture, only allocating enough to use on the smaller billboard. This means that a bounded billboard can only be as large as the original model is from the perspective perpendicular to the billboard. It also has the added advantage of reducing the size of the textures for the model, improving processing times.

Results

The main advantages of this method are removing some of the clutter and the blurred edges, and reducing the areas that can be covered by textures, areas that should not contain much color at first thought. However, its results are again mixed. First, it only shows improvements in metrics for L2 models. This is because L1 models naturally do not have the blurred effect of L2. Furthermore, it has advantages only on very symmetrical tree species such as the Quaking Aspen. The Willow and Acer models are highly disadvantaged, due to parts of their models sometimes getting cut off for some perspectives around the model, also reducing the results of metrics for both L1 and L2 models. Figure 7 shows exactly that.

On the other hand, having bounded billboards can turn into an advantage for all models. Optimizing the placement of the billboards before training, to reduce the possibility for parts of the tree to be cut off due to perspective warps, could best utilize tighter billboards.

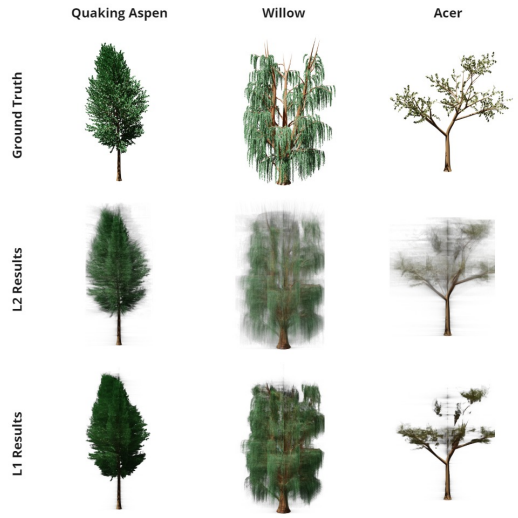

Figure 7: Rendered results of 4 bounded billboard models of all tree species and loss function.

6.4 Two-sided billboards

So far, all billboards were one-sided, meaning that the same texture could be seen on both sides. This can lead to some issues when taking into consideration trees with high-density crowns for which viewing the tree from opposite perspectives may give very different results. Figure 8 shows how even lighting can change between the opposite angles around the tree and how it can affect a model with a single-sided billboard.

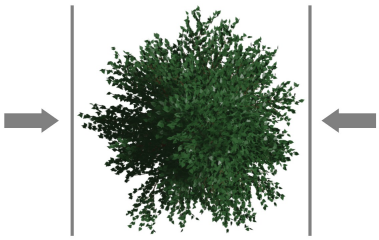

Figure 8: Top view showcasing the different perspectives on the same billboard.

As such, when generating the model, we can simulate two-sided billboards, i.e. the texture on each side is different, by rendering two separate billboards at opposing angles and then indenting them a very small distance in the direction of their normal, thus, when rendering, they will not occupy the same space and create artifacts.

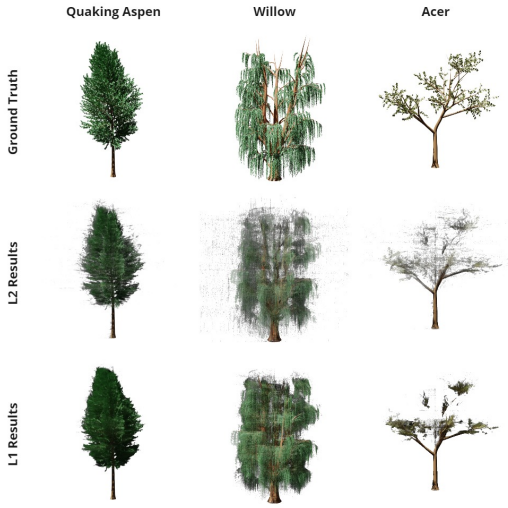

Figure 9: Rendered results of 4 double-sided unbounded billboards for each tree species and loss function.

Results

Figures 9 and 10 showcase the results of these double-sided billboards and bounded double-sided billboards. Metric results for the unbounded double-sided billboards show overall improvements over their single-sided counterparts, however, it greatly depends on the species of the tree and the

number of billboards in the model. For example, most of Quaking Aspen's results show either a lack of improvement or noticeable improvement for both L1 and L2, but the Aspen model has much worse results for the 2 billboards model when optimizing using L1. This shows that different approaches should be taken for each species of tree, as no one method can provide the best results for all.

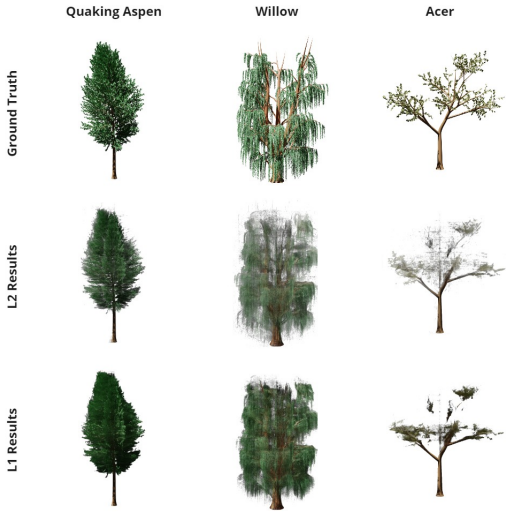

Figure 10: Rendered results of 4 double-sided bounded billboards for each tree species and loss function.

Similar to the results from the previous experiment, bounded billboards offer only slight improvements to symmetric tree species and an overall decrease in the metrics for the other species. This reinforces the idea that separate methods should be devised for each type of tree.

Although having double-sided billboards creates a net improvement in the metrics, disregarding the bias of this method for different tree species, it also comes with some negatives. One such disadvantage is the increased processing time. Because each billboard now has 2 sides, it means that there are double the number of vertices in the model and double the size of the texture to hold it all. Since we are also using a depth peeler to render each optimization step, we now need to render twice as many depth layers as well, as our rasterizer can hit both sides of the billboard before reaching another behind, effectively doubling our render time.

7 Discussion

This section will discuss the results and limitations of our experiments and how to improve them in the future. Moreover, the results subsection will shortly discuss the findings that relate to our proposed sub-questions.

7.1 Limitations and Future work

As mentioned multiple times throughout this paper, there are self-imposed and time limitations for this research. Both types come from the short duration of this research and help limit how much a topic is discussed or explored to still keep the goal of the paper concise.

The first of such limitations is the fact that we only optimize along the y-axis of the model. Implementing, testing, and optimizing for multiple axes would take too long for the span of this project and, taking into consideration that trees are less complex in other dimensions, could provide lower-than-expected improvements.

We have also limited our research to at most 4 billboards per tree. Rendering and optimizing each model takes some time and factoring in double-sided billboards doubles said time. Moreover, 4 billboards leave only 45° between each billboard, enough space for all the billboards to be seen from at least one angle during training.

Another limitation our methods display is the shading of the model. Currently, the models are shaded and rendered externally, thus providing the ground truth images used for optimization. In practice, however, our models would be used in different environments where the shape of our models would not allow for the same shading results. This limits the usability of the models after being compressed as they will not respond to shading perfectly given a different scenario than the one chosen while initially rendering the ground truths. This can be averted by compressing trees with the desired shading from the beginning.

Having a general model for all trees and tree species instead of finding an optimal result before or during the optimization loop is another limitation. This also stems from the time restriction for the research. Focusing on optimizing the texture became the primary goal for the paper and creating more complex models could increase both training time and create issues when it comes to a higher number of billboards, slightly going against the purpose of compression. However, the creation of billboards for any angle with any size has already been implemented for this project. This can be taken further by either finding the best angles to place billboards at before optimization of the model or by allowing the optimizer to decide where to place new billboards throughout the training and starting from a more crude initial model such as a single billboard.

7.2 Results

When it comes to the effects of the number of billboards in a model, starting from the most basic 2-billboard model, we can already see promising results, however, some improvements are to be made. The first aspect we change about this basic model is the number of billboards that make up the model. Increasing their number has a great initial impact on the models, but not for all species of trees. Trees with sparse leaves and branches benefit the least from a higher number of billboards.

Moreover, the different species of trees have a major impact on both the visual results and the quantitative metrics. This can be attributed to the number of dimensions the tree expands in. For example, the Quaking Aspen is mostly symmetrical in shape, thus more billboards allow more of its crown to be visible from more perspectives. On the other hand, for Acer, its low branch count means there are fewer optimal positions a billboard can occupy, thus with our general models, the billboards cover mostly the space between branches instead of aligning with the branches, thus

giving the observed lower metric scores.

Another question we intended to answer is about the best learning function to use. For this, we have experimented with L1 and L2 losses. L2 loss provides better results in terms of metrics but the final models have a blurry appearance. On the other hand, L1 has worse metric scores in comparison and produces darker images but the final models are sharper and clearer. Both functions show advantages and disadvantages for different scenarios and species of tree.

Additionally, the results of single and double-sided billboards were majorly dependent on the type of tree the experiment was performed on. Some types of trees have very thick crowns, similar to the Quaking Aspen. For this type of tree, the best results came from double-sided billboards as the model no longer needs to have the same texture for opposing perspectives, allowing for both better visual results and metric scores, while for trees such as the Acer tree, single-sided billboards tend to give better results.

Finally, the objective of this research is to compress models of trees. To this extent, Table 1 shows the number of vertices our models have compared to the original. Our methods have been able to compress all of the chosen models down to at least 0.028% of the original, allowing for much faster renders with ours than the original. The trade-off, however, is some of the quality of the models, as they can become blurry or darker depending on the used loss function.

	Acer		Willow		Quaking Aspen	
Original	113,155		10,525,153		700,869	
2 Single-sided	8	0.0071%	8	0.000076%	8	0.00114%
3 Single-sided	12	0.0106%	12	0.000114%	12	0.00171%
4 Single-sided	16	0.0141%	16	0.000152%	16	0.00228%
2 Double-sided	16	0.0141%	16	0.000152%	16	0.00228%
3 Double-sided	24	0.0212%	24	0.000228%	24	0.00342%
4 Double-sided	32	0.0283%	32	0.000304%	32	0.00457%

Table 1: Table contains the vertex count for each original tree model. Each subsequent row below the original shows the vertex count of that model that was used to optimize said tree and the ratio between the original and compressed vertex count. These models can be separated by the number and type of billboards that compose them.

8 Responsible Research

The research done in this paper follows responsible research guidelines. The tree models used for creating the datasets originate from the public Blender plugin tree-gen (on GitHub) [9]. These models are free to use for non-commercial purposes as per the GPL-3.0 License the plugin is under. This research also adheres to the NVIDIA license of NVDiffRast [5]. The primitive methods provided by the library are only used for research, thus it is ethical and fully reproducible.

9 Conclusion

Our goal was to find a method for compressing a 3D model of a tree into a more efficient format. We decided to approach this problem by first creating a model composed of billboards. Then we optimize the texture that is applied to it by rendering it and comparing it to the original tree model we are trying to compress.

We found that different species perform best in different settings and, from our experiments, not one type of process could be optimally used for all of them. However, one can adapt our approach depending on what type of tree needs to be compressed. Although our results are not perfect replicas of the original, being somewhat blurry if using L2 loss and losing some of the quality of the original they largely resemble the original. Similarly, L1 loss creates a sharper but darker version of the results of L2 loss.

Moreover, the experiments performed create the fewest vertex models possible, having a fraction in size compared to the original models. Our results have at most 32 vertex models, while the original models have up to $10,525,153$ vertices, reducing the complexity of the model to at least 0.028% of the original.

Finally, this study shows that the methods we have used are possible compression methods and, with improvements, can provide greater results than the models proposed by the billboard rendering and model optimizing approaches that inspired it.

References

[1] A. Jakulin, "Interactive Vegetation Rendering with Slicing and Blending," in *Eurographics 2000 - Short Presentations*, Eurographics Association, 2000.

[2] S. Behrendt, C. Colditz, O. Franzke, J. Kopf, and O. Deussen, "Realistic real-time rendering of landscapes using billboard clouds," *Computer Graphics Forum*, vol. 24, no. 3, p. 507–516, 2005.

[3] J. Munkberg, J. Hasselgren, T. Shen, J. Gao, W. Chen, A. Evans, T. Müller, and S. Fidler, "Extracting Triangular 3D Models, Materials, and Lighting From Images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8280–8290, June 2022.

[4] Google, "Google/draco: Draco is a library for compressing and decompressing 3d geometric meshes and point clouds. it is intended to improve the storage and transmission of 3d graphics.."

[5] S. Laine, J. Hellsten, T. Karras, Y. Seol, J. Lehtinen, and T. Aila, "Modular primitives for high-performance differentiable rendering," *ACM Transactions on Graphics*, vol. 39, no. 6, 2020.

[6] T. Ochotta and D. Saupe, "Compression of point-based 3d models by shape-adaptive wavelet coding of multi-height fields," in *Symposium on Point-Based Graphics 2004* (M. Alexa, ed.), (Aire-la-Ville), pp. 103–112, Eurographics Association, 2004.

[7] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020.

[8] B. Foundation, "Home of the blender project - free and open 3d creation software." Available at https://www.blender.org/.

[9] Friggog, "Friggog/tree-gen: Procedural generation of tree models in blender." Available at https://github.com/friggog/tree-gen.

[10] PyTorch, "Pytorch 2.1 adam optimizer." Available at https://pytorch.org/docs/stable/generated/torch.optim.Adam.html.

[11] "Conda documentation." Available at https://docs.conda.io/en/latest/.

[12] NVIDIA, "Cuda toolkit 12.3." Available at https://developer.nvidia.com/cuda-downloads.

[13] S. Manda, "Sebastianmanda/tree-compression." Available at https://github.com/SebastianManda/tree-compression.

Appendix

			Quaking Aspen			Willow			Acer			Averages
			2 Billboards	3 Billboards	4 Billboards	2 Billboards	3 Billboards	4 Billboards	2 Billboards	3 Billboards	4 Billboards	
Base Models	L2	RMSE	2.24E-05	2.13E-05	2.08E-05	3.72E-05	3.64E-05	3.55E-05	2.95E-05	2.85E-05	2.88E-05	2.89E-05
		SSIM	0.9999966	0.9999971	0.9999974	0.9999894	0.9999903	0.9999909	0.9999932	0.9999939	0.9999937	0.999993633
		PSNR	92.605	93.100	93.340	88.248	88.491	88.698	90.117	90.422	90.373	90.599
		SRE	17.280	17.385	17.637	20.984	20.744	20.902	11.776	11.894	11.820	16.714
	L1	RMSE	2.36E-05	2.29E-05	2.18E-05	4.04E-05	3.74E-05	3.69E-05	3.23E-05	3.18E-05	3.15E-05	3.10E-05
		SSIM	0.9999958	0.9999964	0.9999970	0.9999865	0.9999893	0.9999898	0.9999914	0.9999917	0.9999920	0.999992201
		PSNR	92.049	92.389	92.827	87.448	88.119	88.291	89.279	89.462	89.537	89.933
		SRE	16.401	16.750	17.106	19.511	20.123	20.338	5.230	6.605	7.133	14.355
Bounded Billboards	L2	RMSE	2.20E-05	2.12E-05	2.04E-05	3.84E-05	3.70E-05	3.61E-05	3.22E-05	2.98E-05	2.90E-05	2.96E-05
		SSIM	0.9999968	0.9999973	0.9999976	0.9999882	0.9999898	0.9999909	0.9999908	0.9999930	0.9999936	0.999993107
		PSNR	92.726	93.138	93.496	87.929	88.325	88.585	89.102	89.976	90.276	90.395
		SRE	17.178	17.448	17.654	20.108	20.421	20.827	17.142	14.676	13.016	17.608
	L1	RMSE	2.43E-05	2.27E-05	2.29E-05	4.18E-05	3.91E-05	3.84E-05	3.27E-05	3.22E-05	3.15E-05	3.17E-05
		SSIM	0.9999954	0.9999964	0.9999964	0.9999854	0.9999878	0.9999889	0.9999909	0.9999914	0.9999920	0.999991618
		PSNR	91.801	92.457	92.479	87.231	87.783	88.003	89.165	89.317	89.536	89.753
		SRE	16.046	16.660	16.871	19.249	19.715	20.048	4.773	6.061	6.660	14.009
Double-sided Billboards	L2	RMSE	2.19E-05	2.10E-05	2.04E-05	3.66E-05	3.68E-05	3.65E-05	2.88E-05	2.90E-05	2.88E-05	2.89E-05
		SSIM	0.9999968	0.9999973	0.9999974	0.9999898	0.9999904	0.9999906	0.9999936	0.9999937	0.9999939	0.999993727
		PSNR	92.764	93.201	93.463	88.354	88.440	88.522	90.299	90.262	90.359	90.629
		SRE	17.216	17.592	17.665	20.859	21.033	21.023	11.943	12.093	12.020	16.827
	L1	RMSE	2.44E-05	2.22E-05	2.20E-05	4.07E-05	3.82E-05	4.51E-05	3.27E-05	3.12E-05	3.10E-05	3.19E-05
		SSIM	0.9999952	0.9999967	0.9999970	0.9999863	0.9999892	0.9999874	0.9999908	0.9999921	0.9999924	0.999991893
		PSNR	91.627	92.609	92.731	87.356	87.956	86.994	89.067	89.587	89.632	89.729
		SRE	13.735	16.764	17.325	19.183	20.250	21.570	2.713	7.399	8.047	14.109
Double-sided bounded Billboards	L2	RMSE	2.21E-05	2.09E-05	2.09E-05	3.76E-05	3.60E-05	3.67E-05	3.20E-05	4.18E-05	3.38E-05	3.13E-05
		SSIM	0.9999967	0.9999973	0.9999974	0.9999892	0.9999906	0.9999905	0.9999902	0.9999753	0.9999886	0.999990636
		PSNR	92.674	93.245	93.296	88.150	88.498	88.450	88.928	85.531	88.346	89.680
		SRE	17.134	17.516	17.558	20.447	20.583	20.769	9.995	16.043	11.459	16.834
	L1	RMSE	2.40E-05	2.25E-05	2.24E-05	3.91E-05	3.87E-05	3.97E-05	3.18E-05	3.25E-05	3.19E-05	3.14E-05
		SSIM	0.9999958	0.9999964	0.9999968	0.9999875	0.9999884	0.9999887	0.9999915	0.9999910	0.9999916	0.999991976
		PSNR	91.905	92.521	92.627	87.681	87.826	87.737	89.375	89.156	89.344	89.797
		SRE	16.191	16.702	17.126	19.566	20.022	20.142	3.719	2.052	2.275	13.088
Averages	L2	RMSE	2.21E-05	2.11E-05	2.06E-05	3.74E-05	3.66E-05	3.62E-05	3.07E-05	3.23E-05	3.01E-05	2.97E-05
		SSIM	0.9999967	0.9999972	0.9999975	0.9999892	0.9999903	0.9999907	0.9999920	0.9999890	0.9999924	0.999992776
		PSNR	92.692	93.171	93.399	88.170	88.438	88.564	89.611	89.048	89.839	90.326
		SRE	17.202	17.485	17.629	20.599	20.695	20.880	12.714	13.677	12.079	16.996
	L1	RMSE	2.41E-05	2.26E-05	2.23E-05	4.05E-05	3.83E-05	4.00E-05	3.24E-05	3.19E-05	3.15E-05	3.15E-05
		SSIM	0.9999956	0.9999965	0.9999968	0.9999864	0.9999887	0.9999887	0.9999912	0.9999915	0.9999920	0.999991922
		PSNR	91.846	92.494	92.666	87.429	87.921	87.756	89.221	89.381	89.512	89.803
		SRE	15.593	16.719	17.107	19.377	20.027	20.525	4.109	5.529	6.029	13.891
Total Averages		RMSE	2.31E-05	2.19E-05	2.15E-05	3.90E-05	3.75E-05	3.81E-05	3.15E-05	3.21E-05	3.08E-05	3.06E-05
		SSIM	0.9999961	0.9999968	0.9999971	0.9999878	0.9999895	0.9999897	0.9999916	0.9999903	0.9999922	0.999992349
		PSNR	92.269	92.833	93.032	87.800	88.180	88.160	89.416	89.214	89.676	90.064
		SRE	16.398	17.102	17.368	19.988	20.361	20.702	8.411	9.603	9.054	15.443

Figure 11: Experiment metrics. Cells with orange background denote the highest value between the number of billboards for their metric within a tree type. light blue cells in the Averages column denote the highest average values for their metric.