TU WIEN Informatics

Effizientes Rendern von Wäldern mittels Gruppierter Detailgrade in 3D-Geoinformationssystemen

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Anand Eichner
Matrikelnummer 11808244

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dipl.-Ing. Dr.techn. Daniel Cornel

Wien, 15. Mai 2024

Anand Eichner	Eduard Gröller

TU WIEN Informatics

Efficient Rendering of Forests Using Grouped Levels of Detail in 3D Geo-Information Systems

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Anand Eichner
Registration Number 11808244

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Dipl.-Ing. Dr.techn. Daniel Cornel

Vienna, 15th May, 2024

Anand Eichner	Eduard Gröller

Erklärung zur Verfassung der Arbeit

Anand Eichner

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. Mai 2024

Anand Eichner

Acknowledgements

I want to thank Daniel Cornel, who was responsible for my collaboration with VRVis and helped me get up to speed with their systems. He was also a great help with any questions regarding this paper and the implementation into their existing systems. Further I want to thank my family who have supported me both emotionally as well as financially throughout my studies.

Kurzfassung

Interaktive Visualisierung ist für viele Arbeitsabläufe wichtig. Besonders im Kontext von 3D-Geoinformationssystemen, wo große Datenmengen verarbeitet und für den Nutzer in interaktiver Geschwindigkeit dargestellt werden müssen, um Produktivität und Orientierungssinn zu erhalten. In stark bewaldeten Ländern wie Österreich müssen bei der Visualisierung von Wäldern enorme Mengen an Geometrie dargestellt werden. Naïve Render-Verfahren scheitern selbst bei stark vereinfachter Geometrie. Der Bereich, in dem ein hoher Detailgrad erforderlich ist, ist klein und ändert sich häufig. Der überwiegende Teil der Szene ist weit entfernt und benötigt nur einen geringen Detailgrad. In dieser Arbeit wird versucht, eine Lösung für diese Problemstellung zu finden. Um dies zu bewerkstelligen, wird jeder Baum, der sich nicht in der Nähe der Kamera befindet, durch ein Billboard dargestellt. Um den Rechenaufwand für die Auswahl des geeigneten Detailgrades jedes Baumes zu verringern, werden die Bäume in örtlich begrenzte Gruppen eingeteilt. Die Auswahl der geeigneten Detaillierungsgrade und der Verwurf falls die Bäume sich nicht im Sichtkegel der Kamera befinden, erfolgt dann auf diesen Gruppen. Dieser neue Ansatz wird implementiert, qualitativ evaluiert und mit bestehenden alternativen Ansätzen verglichen. Der Vergleich der Ansätze auf einer Stresstest-Szene zeigt, dass unser neuer Ansatz je nach Szenario zwischen $1,7$ und 6 mal schneller sein kann wie die verglichenen Ansätze, während die visuelle Qualität kaum beeinträchtigt wird.

Abstract

Interactive visualization is important for many workflows. Especially so in the context of 3D geo-informations systems, where large quantities of data have to be processed and presented to the user at interactive speeds for productivity and orientation in the geo-spatial context. In heavily forested countries like Austria enormous amounts of geometry have to be drawn when visualizing forests. Naïve rendering approaches fail, even when using heavily simplified geometry for the individual trees. The region in which details are necessary is small and changes frequently. A major part of the scene is far away and needs little detail. These constraints are what this thesis attempts to find a solution for. Thus each tree is represented by a billboard, if not close to the camera. To decrease the computational complexity of selecting the appropriate level of detail for all trees, they are grouped into batches, for which frustum culling and level of detail selection happens. This new approach is implemented, qualitatively evaluated, and compared with existing alternative approaches. Comparison of the approaches on a stress test scene shows that our new approach can be between 1.7 and 6 times faster than the approaches tested against depending on the scenario, while barely reducing visual quality.

Contents

CHAPTER 1

Introduction

3D visualization of geographic data at interactive speeds is difficult. The huge amount of data required to represent an accurate model of even a small region of the real world is enormous, even when drastic simplifications are made. The visualization of forests is particularly difficult. Trees have a huge amount of geometric and texture detail. Reducing the amount of geometry and texture detail that has to be rendered, is an easy way of increasing performance or ensuring interactivity. Doing so using a naive approach, however, also reduces visual quality, which is not desired.

The trees can either be fully modelled as geometry or approximated using simplified geometry with textures. Simplifying the geometry and relying on textures to fake geometrical detail presents its own problems however. If sparse non-opaque geometry such as leaves and branches is approximated with textures, large sections of the triangle that spans the texture must be transparent. Transparency and alpha mapping is considered a very expensive operation for rendering. If blending is desired then the geometry has to be sorted back to front to achieve correct transparency sorting. There are some approaches for order-independent transparency such as depth peeling, presented by Everitt [Eve01]. These approaches introduce their own costs however, such as a large number of draw passes in the case of depth peeling. Redrawing the entire scene n times for n layers of order-independent transparency is not an option for complex scenes. Even if methods such as alpha-to-coverage are used transparency incurs a cost. Alpha-to-coverage converts the alpha value of the pixel into a coverage mask for the sub-samples of a multi-sample render texture. This still adds an inherent cost to transparency, however, as pixels are on average discarded later as the depth value of the furthest sub-sample is used. Even if alpha clipping i.e. binary transparency is used, which inherently suffers from aliasing artefacts, there is a cost to discarding pixels as opposed to not drawing them in the first place. The rasterizer stage has to compute the interpolant values of the varying shader variables whether the pixel is discarded or not. Thus great care has to be taken to avoid drawing geometry that is not visible or does not improve visual quality. A common

approach to achieve this reduction in drawn geometry, while retaining visual quality, is by defining multiple levels of detail. A level of detail, henceforth referred to as LOD, is a reduced version of the geometry or texture data of an object. This reduced version generally has lower texture resolutions and/or lower triangle count and is thus cheaper to render.

In this thesis we attempt to find a good way to solve the problems discussed above and render large forests in the scope of 3D geo-information systems (GIS) at interactive speeds. Concretely we are trying to formulate an algorithm, with its associated data structures, that minimises the number of draw calls that have to be issued and simultaneously minimises the overhead of determining the correct version of geometry to draw for large numbers of trees. To solve these problems and arrive at the approach presented in this thesis a few key concepts are important and are discussed in the following paragraphs.

Our approach uses an LOD system for the trees. Each tree is either rendered with full detail, or as a view-oriented billboard. Note should be taken that full detail is already extremely simple in our use case. The textures used for the billboards are generated at runtime but in a slow-path, which is only executed if the geometry or material data of the full-detail trees is modified. Due to the potentially large viewing angles imposed by the setting of a 3D GIS, textures for multiple viewing angles are generated. The appropriate texture for each billboard is then selected during rendering in the shader.

The second aspect of our approach is the LOD selection system. To reduce the overhead associated with selecting the LOD for hundreds of thousands, or even millions, of tree objects, we propose grouping regions of a forest into batches. This yields batches of trees for which the same LOD is selected. This reduces the associated overhead while not affecting visual quality. A conservative metric is used for selection. LOD transitions have to be handled correctly by drawing the same batch twice for different LODs while fading between them.

The combination of these two methods forms the foundation of our approach. Specifically we propose a system, which combines runtime generated LOD geometry and textures, with a batching method that groups sections of a forest into batches, for which all LOD selections and view-frustum calculations are made. Nearly all of the per-frame calculations can be easily implemented to run on the GPU, improving data locality. This allows our approach to draw large numbers of trees with very few API calls. Since large sections of a forest are represented by only a small number of batches, the appropriate LOD for huge areas can be computed at a low computational cost. This also allows for the early discarding of large amounts of geometry, further improving performance.

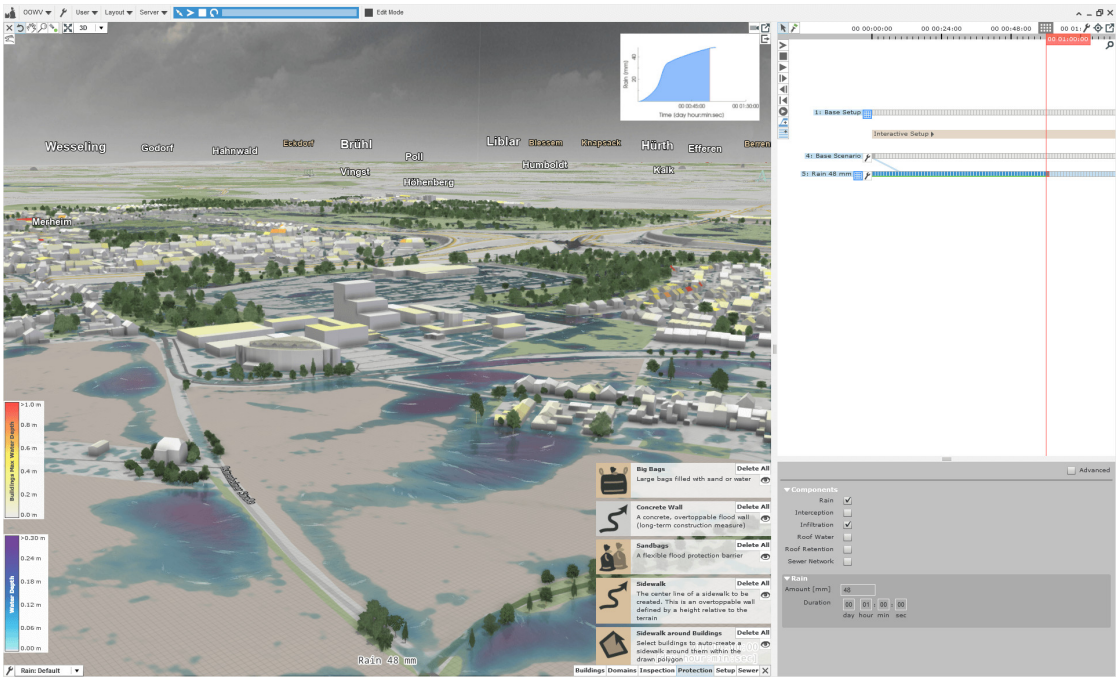


Figure 1.1: Screenshot of the decision support system Visdom by VRVis [VSD]

To evaluate this approach it is implemented in an existing 3D GIS application. The GIS application that is used as a framework for the implementation and evaluation is Visdom [VSD], a decision support system in the context of flood management and public flood risk communication developed by VRVis. A screenshot of the application is shown in Figure 1.1. The area shown in the screenshot only has light tree coverage, however areas with heavier forest coverage can be seen in the background. The trees in those regions are not drawn at all in the screenshot, but could be drawn with the new approach, while retaining the same performance level.

The evaluation of our approach is done by benchmarking sample scenes in the chosen framework and comparing the runtime with two separate render pipelines that are already offered by Visdom. One of these pipelines has equal visual quality, and is the baseline for the visual quality of our approach, while the other one promises better performance at a degraded visual quality level. The target for our approach is to improve on the runtime performance of the first pipeline while not impairing visual quality. The performance of the second pipeline is considered the target for performance, but should not be considered the limit.

CHAPTER 2

Related Work

Figure 2.1: Forests in different games ([BA],[HFW],[ML],[W3])

Rendering Forests is a very common task for many 3D applications. The most notable among them being video games. Forest scenes in various different games can be seen in Figure 2.1 (Top Left: Broken Arrow, Top Right: Horizon: Forbidden West, Bottom Left: Manor Lords, Bottom Right: The Witcher 3). These however generally focus on visual fidelity. Performance is then achieved by careful authoring of the game world by artists manually hiding transitions or manually choosing the level of detail of different regions depending on whether the player can reach these locations or not. Further, even if large game worlds are portrayed, the draw distance of trees is usually kept fairly small

and large open sight lines are usually broken up by artists to hide the limitations of the game engine. While video games are a good case study for different approaches to rendering large forests, video games are an extremely varied medium using many different approaches, most of them not applicable to our application. The visual quality of 3D GIS is generally secondary to usability, workflow, and productivity. This rules out many approaches used by video games as they rely on artist authoring of the landscape to achieve their balance of performance and visual quality. Further even while the portrayed landscapes can often be very large, the total tree count is small compared to real-world datasets. Our approach should work on real-world data as well as semi-procedural data out of the box and not require any human intervention to achieve interactive performance. Games are not the only types of applications that have to render large forests. 3D GISs such as Visdom [VSD] also render large forests in real time and have optimizations for it. These applications favour productivity and interactivity over visual quality. In the case of Visdom for example a more performant approach is offered that renders trees as simplified placeholders. In every case the use of a level of detail system is necessary to render large forests at interactive speeds.

Level of detail systems are a well researched area, with many different approaches all with their own advantages and disadvantages [HD04]. In this thesis a discrete level of detail approach, with some aspects of a view-dependent level of detail, is used. This is similar in character to the approaches presented by Bao et al. [Bao+09] and Zhang et al. [Zha+06]. Our approach reduces geometric complexity yet further, as each individual tree occupies an extremely small section of screen-space, and as such has little individual impact on visual quality. Further, the additional simplification is done as the existing tree meshes in Visdom are already a significantly reduced level of detail as described by Fuhrmann et al. [FUM05] and Bao et al. [Bao+11]. While not directly applicable to our problem, the approaches and solutions presented by Boreal Orienteering [BO], a game about orienteering, can be adapted to our problem as well. Especially their approaches for the lower LODs are a good starting point.

Other modern approaches for high fidelity scene geometry are actively developed, such as Unreal Engine 5's Nanite [BRG21]. Their approach to virtualized geometry, is one such novel level of detail system that, while it does not yet support foliage, is an interesting advancement to look at for further development. Along with Nanite, newer hardware and driver features to support these new rendering pipelines, such as Mesh Shaders, could enable new approaches. The Mesh/Task-Shader pipeline was first introduced by Nvidia, [NV1], and is also available in Vulkan [Khr2]. Such approaches would benefit from more granularity and less overhead for early discards, or even selecting the appropriate LOD during the Task Shader stage for low polygon count LODs. Such approaches would batch small groups of trees into meshlets that can then be discarded at the meshlet level and at a finer grained level in the mesh shader. This would add another level between the batch and individual trees into the hierarchy at which discarding and LOD selection can take place.

CHAPTER 3

Methodology

3.1 Idea

Since most trees are far away given the viewing angles used in a 3D GIS, most trees only cover small areas of the viewport. Therefore it is not cost effective to draw each tree at full detail. Thus a lower LOD should be used for most trees in the scene. Selecting an LOD for large numbers of trees ($> 10k - 100k$ trees) individually carries its own, potentially large, computational cost. Thus we propose a custom approach to LOD selection.

The core idea that leads to the LOD selection scheme presented in this thesis is that an LOD has to be selected for every visible tree in the camera frustum. This limits any algorithm that does selection for every object to a linear runtime complexity. It is desirable to reduce the constant factor of the runtime complexity of any such selection algorithm as much as possible. Since every tree object in our large scale forest is already a very simple object (< 20 triangles), and is usually far away from the camera, we gain very little from such fine-grained selection as switching to a lower LOD earlier is visually nearly indistinguishable unless the user inspects an individual object very closely.

Grouping objects into batches, henceforth referred to as batching, allows some optimizations to be done. Batching allows for the omission of certain per-tree variables as they become invariants for the entire batch. Assuming values such as distance to the camera, containment inside the view frustum, and geometry data as constant within each batch saves computational resources, allowing for the discarding of entire batches of trees at the cost of one computation. The computation of the data needed to make these per-batch calculations is not free. The computation of this data is shifted out of the performance critical section of the program into a preprocessing step that only needs to happen upon changes to the static input data, reducing the cost per frame.

We propose a hierarchical approach to LOD selection. Combining the selection of the LOD over large clusters of trees saves significant runtime for very little penalty in terms of visual quality. Further, we can define two broad cases for a batch of trees. Either the entire batch is contained in a distance interval or it overlaps an LOD transition region, where the transition between two distinct LOD levels happens. In this case the batch needs to be drawn twice with varying alpha values to fade between the two LODs smoothly and avoid pop-in.

The second pillar of our approach for rendering large scale forests consists of the lowest, and in our case second, LOD. Due to the large number of similar objects we decided on the simplest possible geometry for drawing distant trees, i.e. view-oriented billboards. Due to the possibility of large viewing angles, we generate and select from different pre-rendered low resolution images of the trees. When viewing a tree from above it does not make sense to use the same billboard texture as when viewing the same tree from the front, and doing so harms visual quality at large viewing angles. Billboards and texture selection are then generated during the geometry shader stage of the draw call.

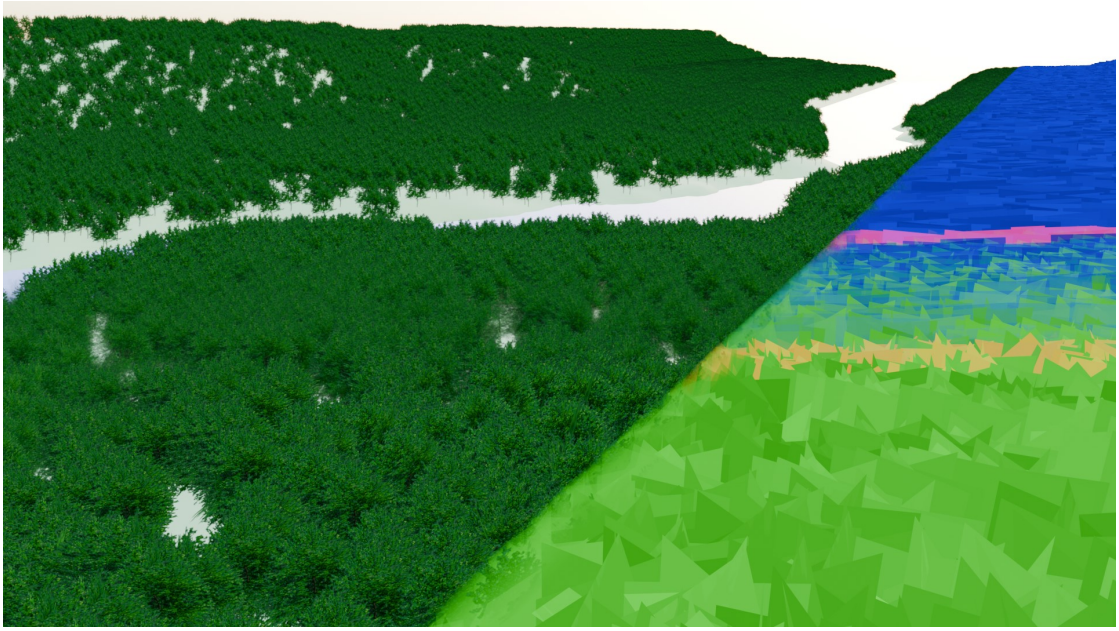


Figure 3.1: Forest with LOD scheme applied for rendering (Green: LOD0, Blue: LOD1, Red: Transition Region Markings

This can also be seen in Figure 3.1. On the right side of Figure 3.1 the LOD of the individual trees can be seen. Green denotes full detail LOD0, whereas blue denotes the billboard level LOD1. The two red lines show the transitional region in which both LODs are drawn, with the further one fading in, while the closer one is fading out. As can be seen in the figure, despite the near field being relatively small, the transition between full detail and billboard is difficult to make out. In the distance the individual trees are

nearly impossible to differentiate from each other, this enables some additional possible optimizations that are discussed in Chapter 7.

3.2 Pipeline structure

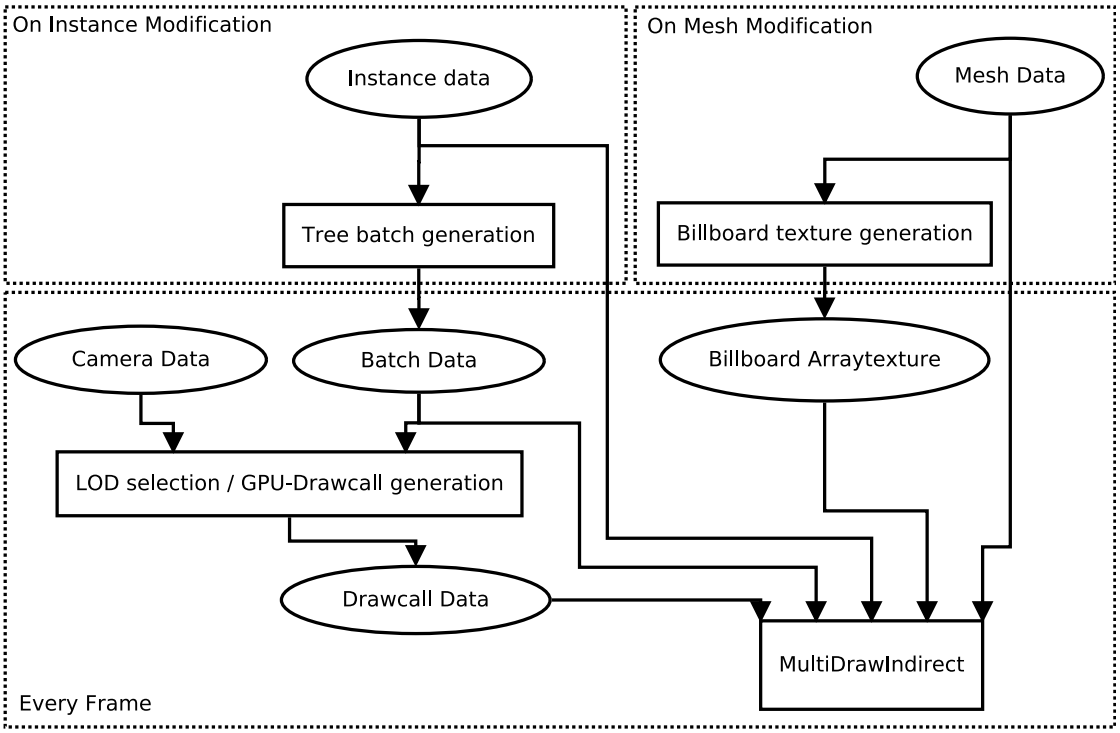


Figure 3.2: Schematic structure of the pipeline

As can be seen in Figure 3.2, the computationally expensive operations, such as batch generation and billboard texture generation, are done only when the scene geometry changes. The data flow between the different operations is indicated using arrows. Data is shown as an ellipse and operations are shown as rectangles. Batch data is only updated if the instance data of the trees, such as the trees transform matrices, changes and the pre-rendered LOD data is only updated if the geometry data or the material data of one of the used trees changes. Only draw command generation happens each frame.

3.3 Billboard textures

3.3.1 Viewing Angles

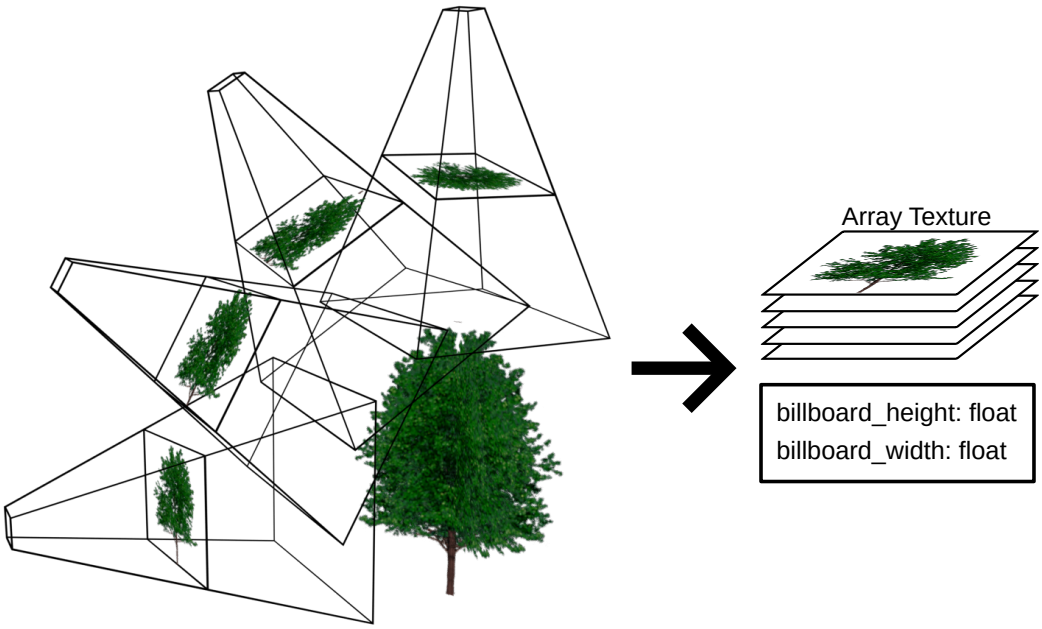


Figure 3.3: Schematic diagram of the LOD billboard textures of a tree

In Figure 3.3 a representation of the LOD data that is generated is shown. In practice orthogonal projection is used to render the bill board textures. The figure shows perspective cones for readability. As mentioned in Section 3.1 it is important that multiple viewing angles of the tree are rendered, since different viewing angles expose different parts of the tree. The number of angles that are rendered is changeable at runtime as the textures are generated from the existing geometry and material data of the trees. The individual textures that are obtained are stored in an array texture. The number of textures presents a trade-off between visual quality and the memory constraints of the application. More angles allow for a smoother transition between horizontal and vertical viewing. Pseudocode for an algorithm implementing the described operation is shown in Algorithm 3.1.

Algorithm 3.1: Render LOD textures

Input: List of mesh data $\mathbf{M_i}$, Number of meshes $\mathbf{N}$, Number of angles $\mathbf{K}$
Output: Array texture $\mathbf{A_j}$, List of LOD metadata $\mathbf{D_i}$

```
Allocate A(N · K)
for i ← 1 to N do
    BB ← Bounding Box of M_i
    Projection ← Calculate Orthogonal Projection From BB
    for k ← 0 to K − 1 do
        View ← Calculate Camera View From Angle((90°/(K−1)) · k)
        Draw Mesh M_i into A_{i·K+k} using Projection, View
    end
    D_i ← Compute LOD metadata (M_i)
end
return
```

The selection of the correct texture is done during the actual rendering in constant time. This is possible because the angles are distributed equally across 90° allowing the computation of the index into the array texture to be done as shown in Equation (3.1).

$$i = a - \left\lfloor \frac{2a}{\pi} \cos^{-1}(\vec{V} \cdot \vec{U}) \right\rfloor \tag{3.1}$$

Where i is the index of the selected image in the array texture, a is the number of angles that have been pre-rendered, $\vec{V}$ is the view direction, and $\vec{U}$ is the up vector of the tree.

The correct texture to use for the billboard during rendering is determined using this index. To avoid obvious jumping upon the switches between these textures, the billboards are not completely view-oriented and instead snap to discrete evenly spaced vertical angles that match the angles of the billboard textures. This reduces the apparent jumping upon the selection of a new angle, as the switch is masked by the perspective distortion of the billboard roughly matching the distortion that occurred when pre-rendering the billboard texture. The perspective distortion thus somewhat smoothly blends between all billboard angles. While this does not perfectly hide the transition it makes it far more subtle compared to keeping the billboard fully view oriented.

3.3.2 Billboard Texture Mipmaps

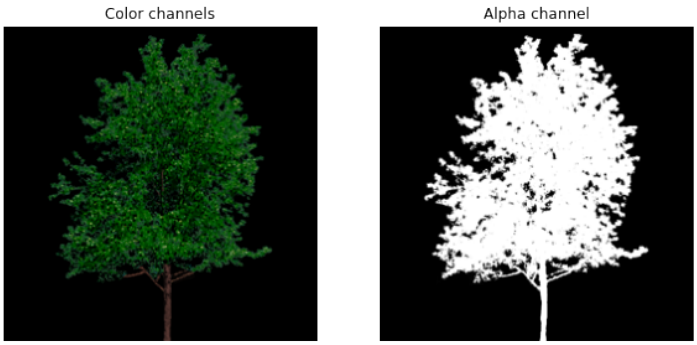


Figure 3.4: Example texture split into color and alpha channels (White: fully opaque; Black: fully transparent)

To avoid aliasing when down-sampling during rendering mipmaps are usually generated for all textures. The basic down-sampling offered by blit operations and further encapsulated in its own function in OpenGL is not sufficient in our case. When down-sampling transparent textures it does not take the alpha values into account when down-sampling the color channels. The reason that this is an issue is apparent in Figure 3.4, where it can be seen that the color channels are zeroed out in the regions of the texture where the alpha values are 0. As there is no conventionally useful information stored in the color channels of fully transparent pixels, since an invisible pixel cannot have any color, they are usually replaced with an arbitrary solid color. This color is constant across the entire image and the replacement is done to improve image compression in common formats (such as PNG *PNG Spec §12.3* [PNG1]).

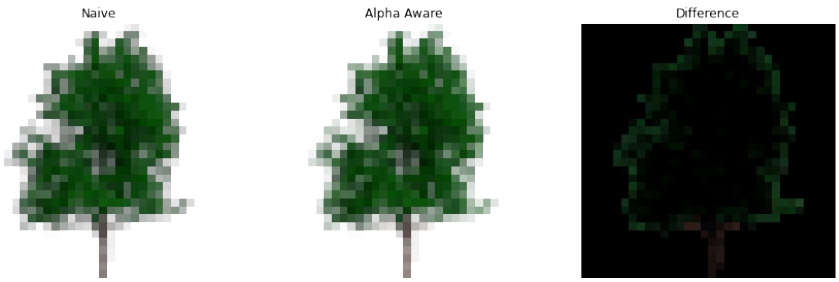


Figure 3.5: Comparison of simple per-channel down-sampling with alpha-aware down-sampling

Since all channels are down-sampled independently of each other using the built in method using blit operations, all pixels that are fully transparent or are adjacent to fully transparent pixels mix with the solid background color that the color channels of those pixels are set to. This leads to dark fringes around the lower resolution versions of the texture when down-sampled this way. Thus a custom algorithm is needed in our case for generating the mipmaps of the tree textures. The implementation of this alpha-aware down-sampling is further described in Section 4.1. The comparison between the built-in method and the alpha-aware method, which is implemented to fix the described issues, is shown in Figure 3.5. This avoids progressive darkening of the textures as they are down-sampled.

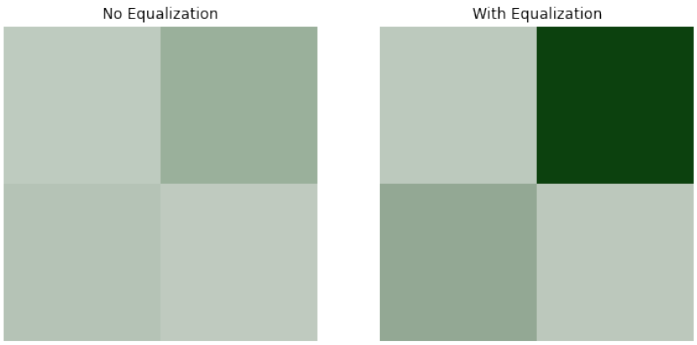


Figure 3.6: Comparison with and without alpha equalization

There is one more problem. As the texture is down-sampled the values approach the mean value of the corresponding channel of the entire texture, which is the desired result in most cases. This leads to high mipmap-levels not having any fully opaque texels left, see Figure 3.6. It is however important that the tree does not become more transparent as higher mipmap-levels are selected. This would lead to the trees changing colour as they blend more with their surroundings at large distances or becoming outright invisible at larger view distances. To correct this an equalization step is done after down-sampling the texture, where the alpha values are mapped back to value range of the original texture linearly. This ensures that even at the highest mipmap-levels there will always be at least one fully opaque texel. Pseudocode for the described alpha-aware method, which avoids the issues that are described, is shown in Algorithm 3.2.

Algorithm 3.2: Alpha-Aware Mipmaps

Input: Square 2D Texture $\mathbf{A}$ with sidelength $2^n, n \in \mathbb{N}$
Output: Mipmaps for $\mathbf{A}$

$(\text{Alpha}_k)_{ij} :=$ alpha values of $(\mathbf{A}_k)$
for $k \leftarrow 1$ **to** *number of Mipmap levels* **do**
  for *each pixel* i, j *in* (A_k) **do**
    // $\tilde{i} \in \{2i, 2i+1\}$ and $\tilde{j} \in \{2j, 2j+1\}$
    $\text{AlphaSum} \leftarrow \left(\sum_{\tilde{i},\tilde{j}} (\text{Alpha}_{k-1})_{\tilde{i}\tilde{j}}\right)$
    $(A_k)_{ij} = \left(\sum_{\tilde{i},\tilde{j}} (A_{k-1})_{\tilde{i}\tilde{j}} \cdot (\text{Alpha}_{k-1})_{\tilde{i}\tilde{j}}\right) / \text{AlphaSum}$
    $(\text{Alpha}_k)_{ij} \leftarrow \text{AlphaSum} \cdot \frac{1}{4}$
  end
  $\text{MinAlpha}_k \leftarrow \min\{(\text{Alpha}_k)_{ij}, \forall i, j\}$
  $\text{MaxAlpha}_k \leftarrow \max\{(\text{Alpha}_k)_{ij}, \forall i, j\}$
end
for $k \leftarrow 1$ **to** *number of Mipmap levels* **do**
  Linearly Remap (Alpha_k) to the value Range of (Alpha_0) using MinAlpha_k and MaxAlpha_k
end
return

3.4 Batch Generation

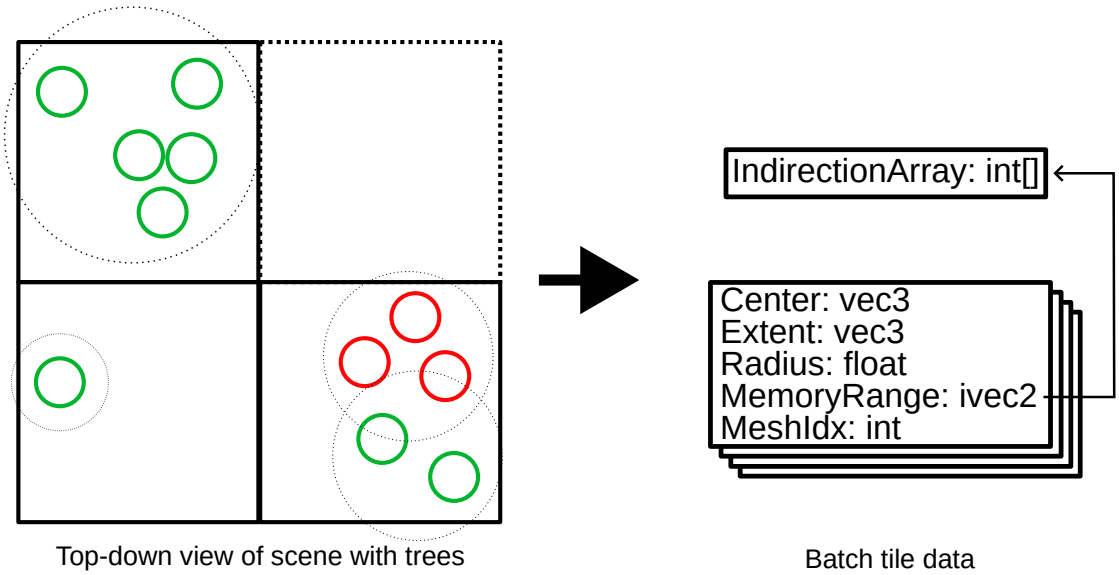


Figure 3.7: Schematic diagram of batches

Trees are grouped into batches based on their horizontal positions on a 2D grid with a given size. Given a forest with a roughly uniform tree density this simple method yields uniform batches, whose size is easily tweaked by changing the size of the grid used to generate them. Each type of tree has a tree mesh that describes its visual appearance, containing the geometry and material data of a tree type. Many trees can share the same tree mesh. As is shown in Figure 3.7, some metadata required for culling and LOD selection is computed for each batch, including the bounding box and the bounding sphere, shown in Figure 3.7 encoded as their Center, Extent, and Radius, as well as an offset and length into the indirection array and the index of the tree mesh used. The different colours shown in Figure 3.7 differentiate between different tree meshes. The dotted circles show the calculated bounding spheres of the batches, note the separation between the tree meshes, shown as circles, and the bounding circles. The gap is due to the perspective chosen, tree meshes have a height that also has to be taken into account when calculating the total bounding sphere of the batch. An additional buffer of indices that provides an extra layer of indirection is used to match batches to the corresponding per-tree data of the trees that are contained in that batch, specifically the transform matrix for each tree. Each batch receives a continuous range of indices in the indirection buffer described above, that lists the trees belonging to this batch. The offset and length of this range is stored for each batch along with the index of the tree mesh that this batch uses. For simplicity during draw command generation, each unique tree mesh has its own batch, even when they occupy the same grid cell.

3.5 Selection & Rendering

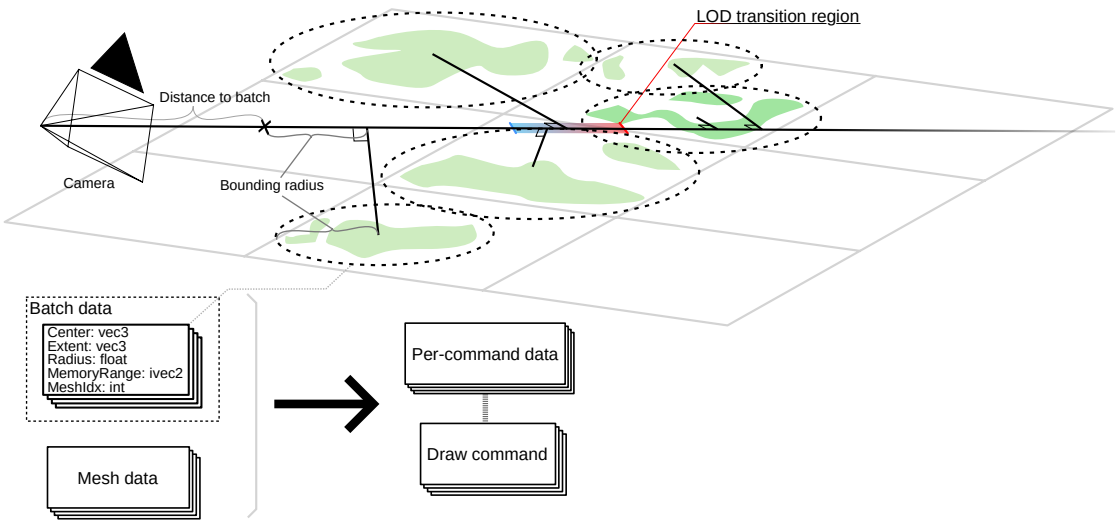


Figure 3.8: Schematic diagram of LOD selection

The batch data and LOD data, which are described in the previous sections, are then used to generate a list of draw commands along with additional per-command data that

is used during rendering. This list of data is then indexed using the index of the draw command during a multi-draw indirect operation that emits all of the draw commands that were generated using a single CPU side API call. To generate the list of draw commands the entire list of batches is iterated. Each batch is projected onto a line given by the view direction to obtain a distance of the batch center to the camera along the view direction. This distance is then used to determine whether the given batch is to be drawn with full detail, using the highest LOD, referred to as LOD0, or as a collection of view oriented billboards, using the lowest LOD, referred to as LOD1. Batches that overlap the LOD transition region must be drawn twice to enable smooth blending of the two LODs. Thus two draw commands are emitted, with differing additional per-command data. One draw command for the full detail LOD0 and a second draw command for LOD1. The fading parameters, used to drive the transition between the two LOD levels, are stored in the additional per-command data along with a flag to denote whether or not the draw command is for regular geometry or for billboards.

The geometric interpretation of the selection process is shown in Figure 3.8. The green regions show tree coverage. The dotted circles are the bounding spheres of the individual batches, projected onto the ground plane. A camera is shown along with the described view ray onto which the center points of the batches are projected. After projection onto the view ray the bounding volumes of the batches become simple one-dimensional intervals where checking overlaps and distances is trivial.

Drawing then happens in a single API call without any intermittent rebinding of resources. The varying behaviour of the shader stages, such as whether the geometry shader stage should generate billboards, or pass the geometry through unchanged, is driven using data from the additional per-command companion data stored for each draw command. The fragment shader stage also behaves differently based on whether the draw command is for billboards or or not, to account for the different texture sources of the billboards.

CHAPTER 4

Implementation

The implementation and evaluation of the approach described in Section 3.1 was done in the scope of a collaboration with VRVis, extending their decision support system Visdom. The implementation of the approach is described in the following sections. First the LOD generation and the batch generation, then the draw command generation and draw command execution is described. The order of the sections matches the order in which the described algorithms are executed during program execution. Lastly the runtime resource requirements are analysed.

4.1 LOD generation

The data used to draw vegetation in Visdom is bundled into mesh sets, each mesh set contains a number of submeshes. The mesh set encapsulates the vertex attribute buffers and shared data of all submeshes. The submeshes then reference ranges of the vertex attribute buffers contained by the mesh set, as well as their own material data. For instanced rendering there is also a list of per-instance transform matrices. All tree meshes are submeshes of the same mesh set containing all the vegetation meshes in Visdom.

When the mesh set, containing the vegetation meshes, is modified, or is first used, the LOD data for every tree mesh is (re)generated. A 2D array texture is allocated with enough layers for every angle and tree mesh. For every tree mesh an orthogonal projection matrix, fitted to the bounding box of the tree mesh, is set up for every evenly spaced viewing angle of the mesh and the mesh is drawn into its respective layer in the array texture without lighting from the respective viewing angle, following Algorithm 3.1 shown in Section 3.3.

After all viewing angles are rendered, mipmaps are generated for every texture layer. As the mipmap generation utilities included in OpenGL have suboptimal handling of transparency for our use case we have implemented our own mipmap generation using

compute shaders as described by Algorithm 3.2 in Section 3.3. This custom mipmap generation is necessary as we need it to preserve the entire range of alpha values for all mipmap levels. The behaviour of the OpenGL function is not desirable in this case, as it would cause the entire tree to become transparent at high mipmap levels. The mipmap generation is implemented in three stages. First the maximum and minimum alpha values for each texture are computed.

In the second stage each texture is down-sampled taking alpha values into account until the highest mipmap level is reached. Finally, in the third stage, the alpha values of the generated mipmap levels from the second stage are corrected using the minima and maxima from the first stage.

Some metadata for billboard and draw command generation also needs to be precomputed during the LOD generation step and uploaded to device memory. For each submesh the offset in the vertex array object and its triangle count need to be stored for draw command generation on the GPU. Further, width and height values for the billboard generation need to be computed based on the bounding box of the submesh.

To avoid later rebinding of meshes during rendering a single dummy triangle, used as an origin and scale reference to generate billboards in the geometry shader stage, is appended to the end of the mesh sets vertex attribute buffers.

4.2 Batch generation

When scene data for the vegetation changes, a set of batches is generated for each tree mesh. To generate the batches, a grid-based approach is used, as described in Section 3.4. The list of all trees is grouped into batches according to a grid of equal size grid cells based on their x and y coordinates. To avoid allocation of a large number of empty grid cells a spatial hashing scheme is used to only allocate grid cells that contain trees. Note, that clustering algorithms like the one presented by Ganganath et al. [GCT14], can be used to generate batches that have a more uniform tree count, leading to more consistent performance for varying forest densities. Implementation of such a clustering method was decided against in this case due to implementation effort. The clustering approach yields improvements when tree coverage is very sparse, where few trees cover many grid cells.

The per-batch tree indices are linearised in a large indirection array used to index into the array of transform matrices of the trees during rendering, to avoid duplication of per-tree data. After all trees are grouped into their respective batches, metadata for each batch is generated for later draw command generation and LOD selection. The collected per-batch metadata contains the geometric center of the batch, the extent of its bounding box, the radius of its bounding sphere, the two-dimensional index of the batch in the grid, the offset and length of the batch data in the indirection array, and the submesh index that this batch belongs to. Both the indirection array and the metadata of all batches are uploaded to device memory for rendering.

4.3 Draw command generation

All LOD selection and draw command generation is done on the GPU to avoid synchronization points with the CPU as much as possible. A command buffer of sufficient size to contain the draw commands for all batches is allocated ahead of time, along with a buffer to contain the additional per-command data for every draw command. An additional buffer to count the number of draw commands per submesh for usage during draw command generation as an atomic counter for indexing is also allocated.

The draw command buffers are cleared between consecutive frames. LOD selection and draw command generation is then done in a single compute shader invocation. For every batch, it is checked whether the batch is beyond the distance at which trees should be drawn, referred to as draw distance, or is entirely outside of the view frustum of the camera, in order to discard the batch early. For every LOD the batch is checked against the transition distance of the LOD, where it replaces the previous LOD, and whether or not it overlaps the transition distance of the next LOD. The draw commands are then generated along with additional per-command data. A pre-computed fade gradient describing the start and end distance between which the opacity should be linearly interpolated to visually fade in, or fade out the drawn objects is stored. Data on whether the draw command should be drawn using billboards, the respective index of the submesh used, and the LOD of the draw command are also stored. If the batch overlaps an LOD transition, then two draw commands are generated for both LODs.

4.4 Draw command execution

Execution of the draw commands is then handled by a single CPU side API call to `glMultiDrawElementsIndirect`. The shaders differ from the regular pipeline of Visdom in some key ways. During the vertex shader stage, the extra indirection from the batch indirection array is used to access the transform matrix of the object.

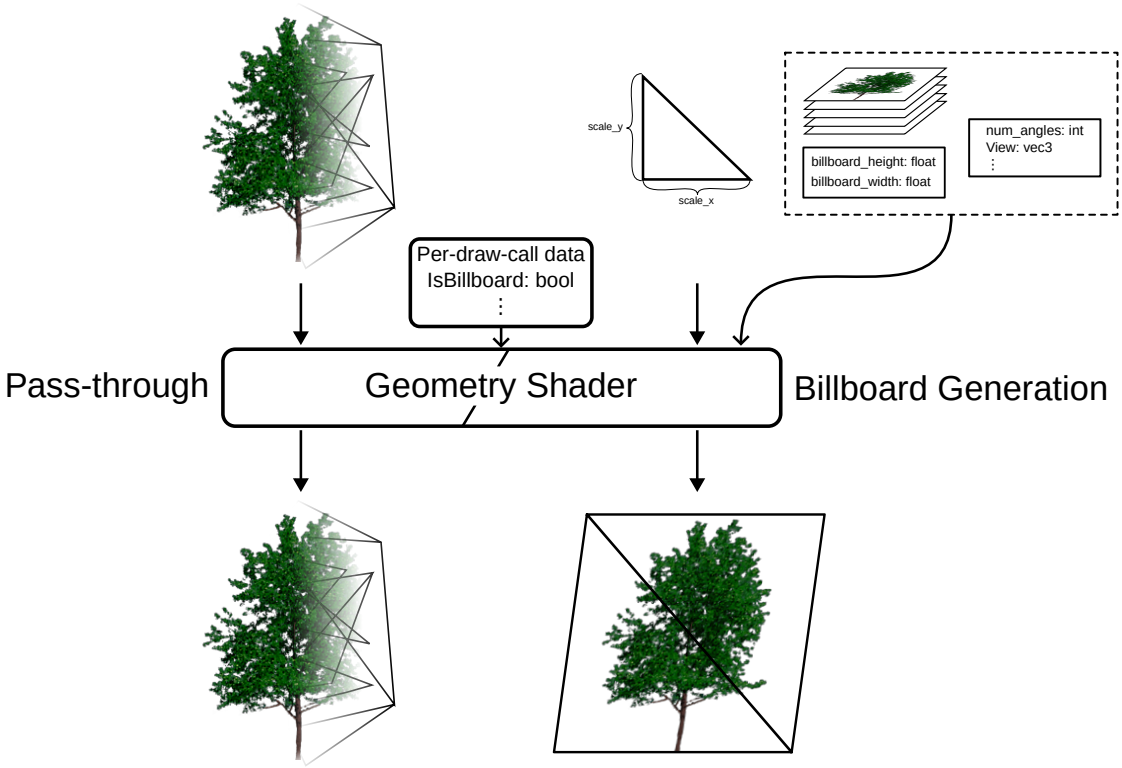


Figure 4.1: Diagram of the two paths in the geometry shader

During the geometry shader stage, a fine-grained discard operation, based on whether the tree is within the draw distance, is performed. Further, if the LOD of the draw command is LOD1, a billboard is generated from the dummy triangle, and the correct texture is selected based on the submesh index of the draw command and the Equation (3.1) described in Section 3.3. If the draw command is a non-billboard LOD, then the triangle primitives are passed through unmodified after the draw distance check. A schematic diagram of the two described paths in the geometry shader can be seen in Figure 4.1.

During the fragment shader stage, the texture load is switched between the texture defined by the tree mesh or a texture from the billboard texture array depending on whether the draw command is for a billboard LOD. Further the alpha value of the fragment is modified based on the fade gradient of the batch, that was computed during draw command generation, and the fragments distance, to achieve a smooth cross-fade between LODs.

4.5 Runtime resource analysis

In this section the additional resource requirements in terms of runtime and space of the approach are analysed. In the following section a few parameters are used to define the resource demands. These parameters are as follows:

- n ... Total number of trees
- t ... Total number of batches
- m ... Total number of tree meshes
- a ... Number of billboard angles
- r ... Resolution of billboard textures
- l ... Number of LOD levels

4.5.1 Space Requirements

The space requirements of different components of our approach are listed below in big-O notation, along with a description of what is stored.

- The indirection array of the batches. One index value is stored for each tree. $\mathcal{O}(n)$
- Additional data for each batch is stored. $\mathcal{O}(t)$
- Additional data for each tree mesh and the offsets of the specific sections in the batch data buffer are stored. $\mathcal{O}(m)$
- One texture is stored for each angle, and tree mesh for the billboard textures. $\mathcal{O}(m \cdot a \cdot r^2)$
- A command buffer with sufficient size l draw commands per batch is pre-allocated. $\mathcal{O}(l \cdot t)$
- During selection an additional buffer with m elements is used. $\mathcal{O}(m \cdot l)$

The total additional memory demand is thus $\mathcal{O}(n + t + m + m \cdot a \cdot r^2 + l \cdot t + m) = \mathcal{O}(n + l \cdot t + m \cdot a \cdot r^2)$ and assuming r, l, a as fixed parameters then the total space complexity is $\mathcal{O}(n + t + m)$. The exact number of additional bytes required is given in the following Equation (4.1).

$$4n + 64t + 24m + 16amr^2 + 52lt + 4lm \tag{4.1}$$

During our testing, batches were generated using a 500 meter grid, and 10 angles with a resolution of 128 by 128 were generated for the billboards. For our application only two LODs were used, LOD0 and LOD1 as described previously. For these parameters a rough estimate can be computed. An exact value can not be determined a priori as the total number of batches depends on the distribution of the individual trees and how they overlap the grid used to generate the batches.

Though an upper bound can be computed. A uniform distribution of trees as a worst case for memory, since it will cover the most grid cells, is assumed. An average $5 \times 5\,km$ region of Austria, with a tree density of $\sim$ 36700 trees/km^2 [Cro+15], is used as an example. Assuming five different tree meshes, the area of 25 km^2 is covered by 500 batches and contains $\sim 9.2 \cdot 10^5$ trees. Following Equation (4.1) yields an estimated maximum additional memory requirement for such a region of $\sim$ 16.9 MB.

4.5.2 Runtime Requirements

The runtime requirements of the approach are analysed. The itemized runtime complexity of the individual algorithms used during precomputation tasks followed by their total runtime complexity is described below. After, the runtime complexity of all computations, that happen every frame, is analysed.

Precomputation

The runtime complexity of each algorithm that is run during precomputation is listed below.

- Each billboard angle for each mesh must be rendered and downsampled. $\mathcal{O}(m \cdot a \cdot \log(r) \cdot r^2)$
- All trees must be grouped into batches $\mathcal{O}(n)$
- Additional metadata for each tree mesh must be extracted. $\mathcal{O}(m)$

This leads to a total time complexity for the precomputations of $\mathcal{O}(m \cdot a \cdot \log(r) + n + m)$, or assuming a, r as fixed parameters $\mathcal{O}(m + n)$.

Per Frame

Each batch needs to be categorized into one or two LODs and their draw commands need to be generated ($\mathcal{O}(l \cdot t)$). An algorithm that runs in $\mathcal{O}(\log l)$ for finding the correct LOD to select could be implemented using interval-trees [AT95] or a binary search. We implemented a naive $\mathcal{O}(l)$ search as the number of LODs is very limited (two). The entire selection and draw command generation is done on the GPU and no data is copied back to the CPU. This avoids extra synchronization points that could stall the rendering pipeline.

4.5.3 Bottlenecks

From the resource analyses done above some insights can be gained. There is an obvious memory sink, that could become a bottleneck, which is apparent from the resource analyses.

The first possible bottleneck is per-batch data. When tree coverage is sparse the number of batches could be large, with each batch containing only a small number of trees. This leads to a large memory and runtime overhead for no gain or even additional costs. Thus the number of batches should be much smaller than the number of trees to reduce both memory consumption as well as runtime of the selection. This can either be achieved by choosing larger grid sizes, or by implementing better batch generation as mentioned in Section 4.1.

The other possible bottleneck is the draw command buffer. The way the draw commands are pre-allocated might be improved with multi-frame pipelining. By reading back the number of draw commands that were generated in the previous frame the buffer can be shrunk or expanded using a heuristic based on the previous frame. Though this creates the problem that there might not be enough space allocated for all the draw commands of the current frame, in which case not all batches that should be drawn can be drawn. While visual fidelity might suffer slightly, when large changes in perspective occur and not all draw commands can be created, memory consumption, and overhead for empty draw commands, could be improved drastically in this way. Queuing another frame immediately if some draw commands could not be issued would alleviate issues with the visuals as soon as possible. Queuing another frame would be necessary in the context of our implementation in Visdom, as frames are generated on demand and not continuously. This would keep interactivity the same while potentially drastically reducing memory demands at the cost of presenting late pop-ins of vegetation on drastic scene changes.

CHAPTER 5

Evaluation

In this chapter we showcase our results and evaluate them quantitatively and qualitatively. We tested our implementation on a modified version of an existing project containing a vast part of the Wachau region of Austria. The project was modified to increase normal tree density by a factor of 10. We ran the same benchmark using our approach as well as the existing vegetation rendering pipeline built into Visdom as well as the primitive impostor tree rendering pipeline also built into Visdom.

The vegetation rendering pipeline, named *Vegetation*, renders the entire geometry for every tree. The geometry drawn is identical to LOD0 used in our new approach and consists of a billboard cloud representation of the tree. The primitive impostor tree rendering pipeline, named *Primitive*, draws billboard quads for each tree, simulating the geometry by doing ray casting on simple geometry in the fragment shader. *Primitive* is offered as an alternative to *Vegetation* with improved performance at the cost of lower visual quality. The geometry, that is simulated for the purpose of these ray cast operations, is a sphere on top of a cylinder. Both of these pipelines do view frustum culling on a per-tree basis.

This stress test shows the scalability of our implementation compared to the two previous implementations inside Visdom that were tested against. Further, a less demanding scene with synthetic data that only contains basic terrain and a normal density of trees, referred to as *Synthetic*, was also tested. There the performance of our approach for different values of the adaptable parameters was compared.

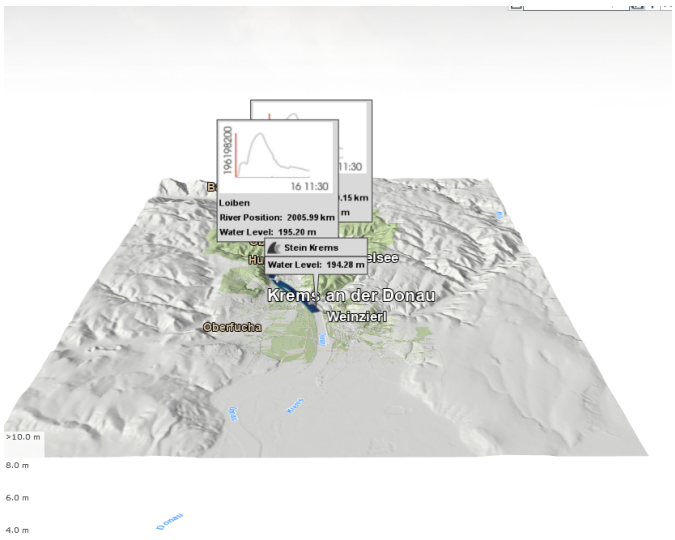
5.0.1 Comparison with previous methods

For the stress test three different benchmarks were run.

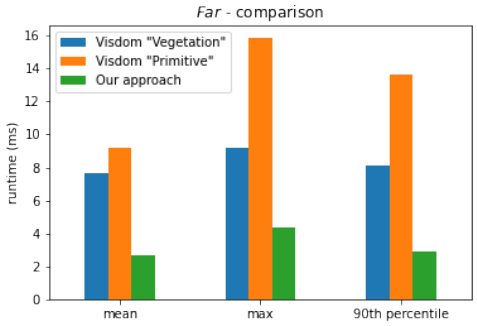
- A *far* benchmark, where the camera is positioned so far away that all trees are beyond the draw distance, and should be culled.

- A *medium* benchmark, where the camera is positioned at a medium distance, this is also the most similar to a real-world scenario.
- A *close* benchmark, where the camera is positioned at a very low altitude above the forest, which is the most computationally expensive of the three benchmarks.

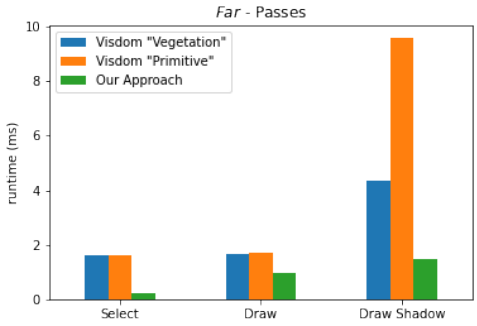
For all three benchmarks the runtime statistics are taken over a 360 degree horizontal camera sweep.



(a) Screenshot from the *Far* benchmark



(b) Runtime in the benchmark

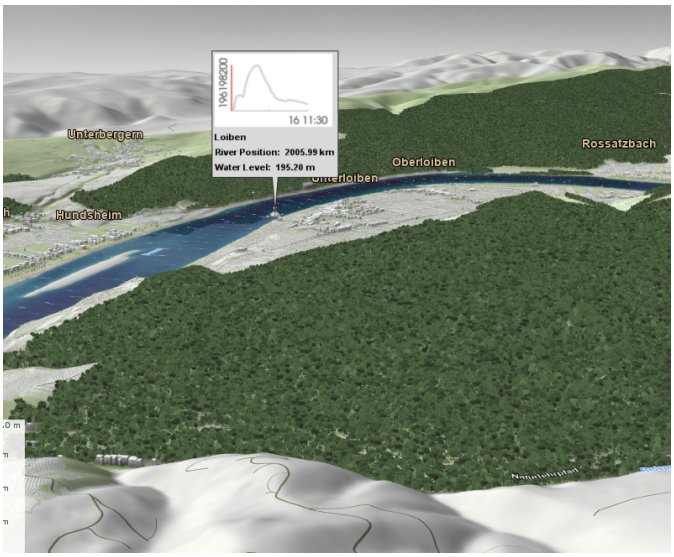


(c) Mean runtime of the different passes

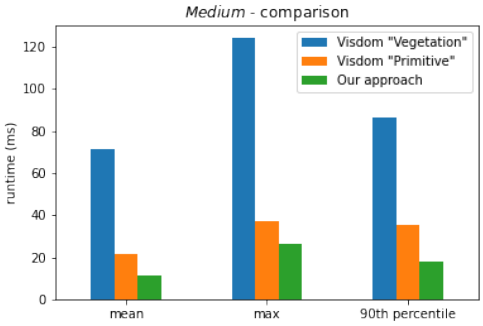
Figure 5.1: *Far* benchmark

In the *far* benchmark (Figure 5.1), a notable improvement over both *Primitive*, which seems to perform uniquely bad in this benchmark, and *Vegetation* can be observed. The hierarchical approach allows us to discard entire regions of forest, and thus drastically reduces overhead in this benchmark. Further, *Primitive* likely suffers in this benchmark due to the usage of `gl_FragDepth`. Due to this the early Z-test, an optimization strategy of the shader pipeline, is disabled [Khr1]. This greatly affects performance

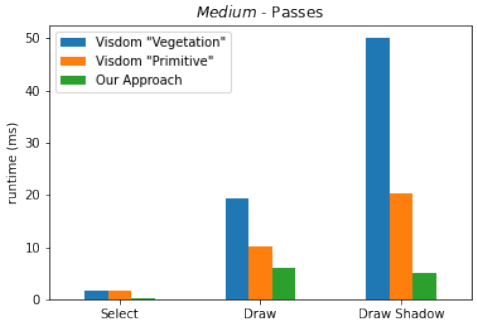
despite nothing being drawn. Further, the reduced computations from only doing view frustum culling on a per-batch basis, instead of a per-tree basis has a significant effect on runtime in this benchmark.



(a) Screenshot from the *Medium* benchmark



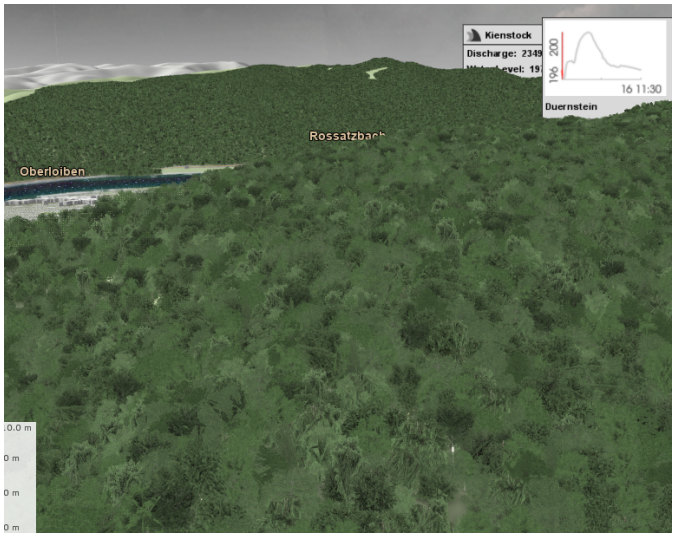
(b) Runtime in the benchmark



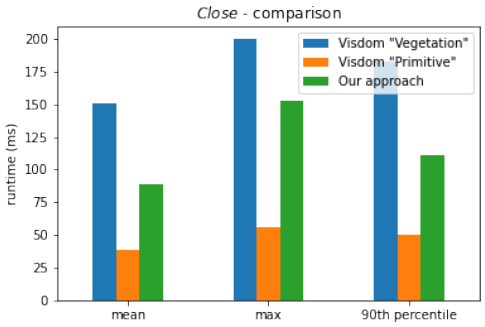
(c) Mean runtime of the different passes

Figure 5.2: *Medium* benchmark

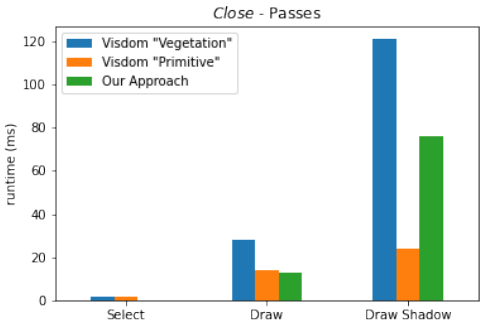
In the *medium* benchmark (Figure 5.2), the largest improvement of the three benchmarks can be observed, as our approach outperforms *Vegetation* by a significant margin, and even outperforms *Primitive*. The *medium* benchmark containing the most realistic camera position, shows the strong performance of our approach in the application scenario for which it was designed. The GPU usage is significantly reduced by only drawing billboards instead of billboard clouds for each tree, without negatively impacting visual quality.



(a) Screenshot from the *Close* benchmark



(b) Runtime in the benchmark



(c) Mean runtime of the different passes

Figure 5.3: *Close* benchmark

The least improvement can be observed in the *close* benchmark (Figure 5.3), where our approach is outperformed by *Primitive*. While our approach is outperformed by *Primitive* for close ranges, it still outperforms *Vegetation* by a significant margin at those same distances. The result further makes sense as this is the benchmark where our approach has the largest overlap with *Vegetation*, as the closest LOD is rendered identically to *Vegetation*.

5.0.2 Comparison of parameter choices

Another test to compare different choices for the adjustable parameters of our method was conducted using a different sample scene with normal tree density, named *Synthetic* as mentioned at the beginning of Chapter 5. The runtime statistics were captured over a distance sweep to a fixed location, moving the camera away until all trees are beyond

the draw distance and then returning the camera to its original position.

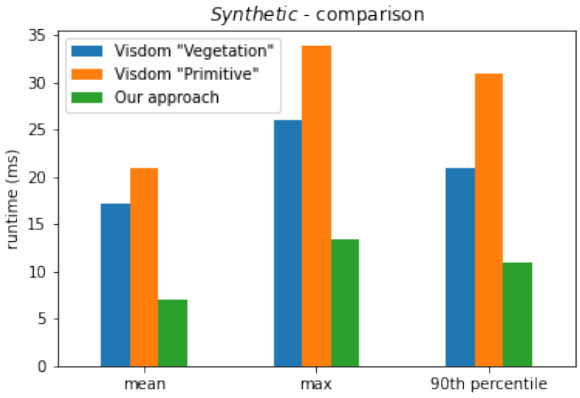
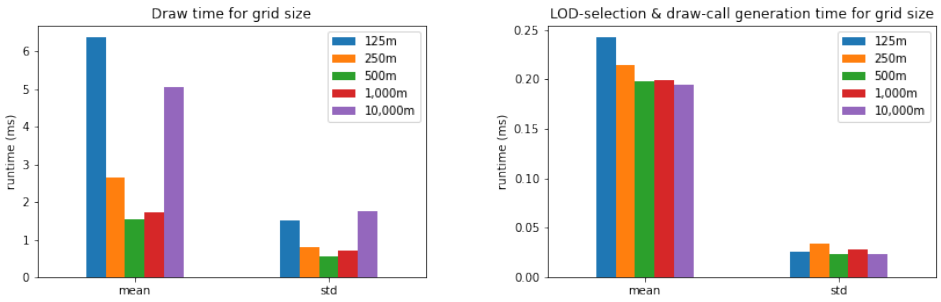


Figure 5.4: Comparison benchmark

In the comparison between our approach and the two previous approaches implemented in Visdom, shown in Figure 5.4, it can be observed that our method outperforms both previous methods. *Primitive* presents worse performance than might be expected, however this can be explained by the same reasoning as in the *Far* benchmark (Figure 5.1).



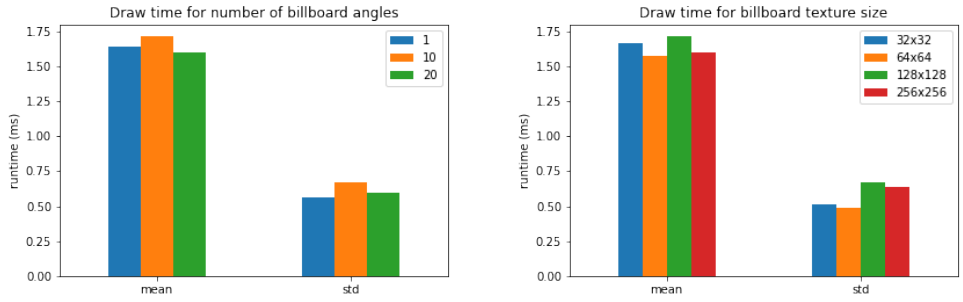
(a) Draw time per grid size

(b) Selection time per grid size

Figure 5.5: Batch grid size benchmark

Varying the size of the grid used to generated the batches, a large performance difference during drawing can be observed for the two extremes (Figure 5.5a). The time taken to select the LOD levels and generate the corresponding draw commands decreases with increasing grid size as expected. The large decrease in performance for very small grid sizes is due to an increasing overhead of dispatching many draw commands, and worse cache locality of the indirection array. The large decrease in performance for very large grid sizes is due to large overdraw, and reduced early discard. For large grid sizes more work has to be done later in the pipeline, as every tree has to be drawn twice for the

LOD transition. From this it can be seen that a grid size of 500 meters is the sweet spot for our application.



(a) Number of billboard angles benchmark (b) Billboard texture size benchmark

Figure 5.6: LOD data benchmarks

As can be seen in Figure 5.6, neither the number of pre-rendered angles, nor the texture size of the billboards seem to have a significant influence on performance. Both increase the memory footprint of the pre-generated LOD data and may affect performance for very large values as memory bandwidth limitations come into effect. Thus they should be chosen as small as possible while retaining the desired visual quality. At a minimum texture size should match the size in pixels that a billboard can occupy on screen, which depends on the transition distance of LOD1 and the size of the trees. During our testing 10 angles with a texture size of $128x128$ were used and gave satisfactory results.

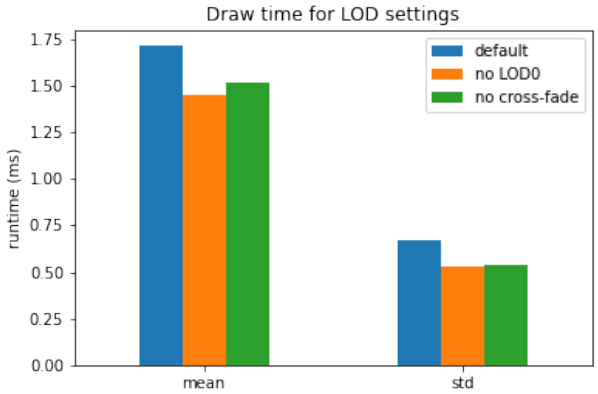


Figure 5.7: LOD settings benchmark

In the LOD settings benchmark (Figure 5.7), it can be observed that reducing the transition range of LOD1 to 0 (listed as "no LOD0") reduces runtime slightly, as all trees are rendered as billboards. Batches that overlap the camera still have a draw command

emitted for them. Further setting the LOD fade distance to 0 (listed as "no cross-fade") improves performance, to a lesser degree than disabling LOD0 however. The cross-fade setting has little performance impact for values smaller than grid size used to generate the batches and should be chosen based on subjective visual quality. The comparison runtime listed as "default" uses an LOD1 transition range of $1000\,m$ and a fade distance of $250\,m$, and matches the values used throughout our testing.

CHAPTER 6

Conclusion

In summary, a robust rendering approach for large forests for use in 3D GIS was presented. The approach was implemented, and evaluated in the scope of the chosen framework and optimal parameter values were determined for the specific use cases of the chosen 3D GIS.

Overall our implementation for rendering large-scale forests provides a significant increase in rendering performance for vegetation-heavy scenes in 3D GIS. In the application that was extended for the purpose of this evaluation it yielded speed-ups ranging from 70% up to 500% over the performance of the two previous approaches. It produces a reliable and extensible foundation for further performance improvements in the rendering pipeline of the chosen framework. There it has fully replaced the previous approaches and is already in production use, enabling higher interactivity for planning tasks without impacting visual quality.

As discussed, there are many alternative approaches to rendering large numbers of trees efficiently, however we believe that our implementation strikes the best balance for our application. Our LOD0 is already considered a lower LOD in the scope of many other approaches. Further ideas for possible improvements over the current implementation are presented in Chapter 7.

CHAPTER 7

Future Work

Future work could improve several aspects of the rendering pipeline in Visdom. The algorithm used to generate the batches could be massively improved. An algorithm that takes tree density into account while creating the batches could further improve performance. This would be accomplished by a reduction of overhead for sparse areas and an increase in resolution for dense areas, which allows for more aggressive culling and LOD selection to further minimise overdraw.

Another increase in performance especially for further away scenery could be accomplished by pre-processing the batch data such that subsets of trees for each batch with predefined densities can be rendered. Due to the shallow viewing angles imposed by large view distances most trees that are drawn overlap each other. Therefore one can reduce the number of trees that are drawn while only affecting the texture and silhouette of the resulting forest slightly. This would allow a reduction in overdraw and potentially an increase in draw distances. The subsets could be represented by storing multiple ranges, and sorting the indices in the batches such that drawing a larger subset evenly increases the density of the drawn trees, without leaving obvious gaps.

Generating tighter bounding spheres for the batches can also boost performance by increasing the number of batches that can be culled. The system could be further extended to render objects other than trees, or other radially symmetric objects, such as buildings. To accomplish this, without losing too much visual fidelity, multiple azimuth angles are generated in addition to the multiple elevation angles already generated. Sorting of the indirect draw commands by distance could further improve performance by reducing overdraw. Introducing another type of LOD that turns an entire batch into a single opaque ultra-low-polygon-count mesh, of similar shape to the convex hull of the batch, could improve visual fidelity by extending the maximum possible draw distance for little extra performance cost.

List of Figures

List of Algorithms

Bibliography

[AT95] Chuan-Heng Ang and Kok-Phuang Tan. "The interval B-tree". In: *Information Processing Letters* 53.2 (1995), pp. 85–89. ISSN: 0020-0190. DOI: 10.1016/0020-0190(94)00176-Y. URL: https://www.sciencedirect.com/science/article/pii/002001909400176Y.

[BA] *Broken Arrow*. Steel Balalaika, Slitherine Games. 2024. URL: https://www.slitherine.com/game/broken-arrow (visited on 05/04/2024).

[Bao+09] Guanbo Bao et al. "Billboards for Tree Simplification and Real-Time Forest Rendering". In: *2009 Third International Symposium on Plant Growth Modeling, Simulation, Visualization and Applications*. 2009, pp. 433–440. DOI: 10.1109/PMA.2009.38.

[Bao+11] Guanbo Bao et al. "Realistic Real-Time Rendering for Large-Scale Forest Scenes". In: *2011 IEEE International Symposium on VR Innovation*. 2011, pp. 217–223. DOI: 10.1109/ISVRI.2011.5759637.

[BO] Jani Honkanen. *Boreal Orienteering: Technology*. URL: http://www.borealorienteering.com/tech (visited on 04/15/2024).

[BRG21] Karis Brian, Stubbe Rune, and Wihlidal Graham. "A Deep Dive into Nanite Virtualized Geometry". In: *ACM Siggraph: Advances in Real-Time Rendering in Games Course* (2021).

[Cro+15] Thomas Ward Crowther et al. *Mapping tree density at scale. Nature. – processed by Our World in Data. "Number of trees per km²" [dataset]*. 2015. URL: https://ourworldindata.org/grapher/number-of-trees-per-km (visited on 05/02/2024).

[Eve01] Cass Everitt. "Interactive Order-Independent Transparency". In: *NVIDIA Corporation* 2 (Oct. 2001).

[FUM05] Anton Fuhrmann, Eike Umlauf, and Stephan Mantler. "Extreme Model Simplification for Forest Rendering." In: *Natural Phenomena*. Jan. 2005, pp. 57–66. DOI: 10.2312/NPH/NPH05/057-066.

[GCT14] Nuwan Ganganath, Chi-Tsun Cheng, and Chi K. Tse. "Data Clustering with Cluster Size Constraints Using a Modified K-Means Algorithm". In: *2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. 2014, pp. 158–161. DOI: 10.1109/CyberC.2014.36.

[HD04] Tan Kim Heok and Daut Daman. “A review on level of detail”. In: *Proceedings. International Conference on Computer Graphics, Imaging and Visualization.* 2004, pp. 70–75. DOI: 10.1109/CGIV.2004.1323963.

[HFW] *Horizon: Forbidden West.* Guerilla Games, Sony. 2022. URL: https://www.playstation.com/de-at/games/horizon-forbidden-west/ (visited on 05/04/2024).

[Khr1] *OpenGL wiki: Early Fragment Test.* Khronos. URL: https://www.khronos.org/opengl/wiki/Early_Fragment_Test#Limitations (visited on 04/15/2024).

[Khr2] Christoph Kubisch. *Mesh Shading for Vulkan.* Khronos. URL: https://www.khronos.org/blog/mesh-shading-for-vulkan (visited on 04/15/2024).

[ML] *Manor Lords.* Slavic Magic, Hooded Horse. 2024. URL: https://manorlords.com/ (visited on 05/04/2024).

[NV1] Christoph Kubisch. *Introduction to Turing Mesh Shaders.* Nvidia. URL: https://developer.nvidia.com/blog/introduction-turing-mesh-shaders/ (visited on 04/15/2024).

[PNG1] *PNG Spec §12.3.* URL: https://www.w3.org/TR/png/#12Encoder-colour-handling (visited on 04/15/2024).

[VSD] *Visdom.* VRVis. URL: https://www.visdom.at (visited on 04/28/2024).

[W3] *The Witcher 3: Wild Hunt.* CDProject: Red. 2015. URL: https://www.thewitcher.com/us/en/witcher3 (visited on 05/04/2024).

[Zha+06] Yi-Kuan Zhang et al. “Image Based Real-Time and Realistic Forest Rendering and Forest Growth Simulation”. In: *2006 Second International Symposium on Plant Growth Modeling and Applications.* 2006, pp. 323–327. DOI: 10.1109/PMA.2006.44.