Cone Normal Stepping

by

Divan Desmond Burger

Submitted in partial fulfillment of the requirements for the degree
Master of Science (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

December 2018

Publication data:

Divan Desmond Burger. Cone Normal Stepping. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, November 2018.

Electronic, hyperlinked versions of this dissertation are available online, as Adobe PDF files, at:

http://cirg.cs.up.ac.za/

http://upetd.up.ac.za/UPeTD.htm

Cone Normal Stepping

by

Divan Desmond Burger
E-mail: divan.burger@gmail.com

Abstract

This dissertation examines several methods of relief mapping, such as parallax mapping and cone step mapping, as well as methods for soft shadowing and ambient occlusion of relief maps. Ambient occlusion is an approximation of global illumination that only takes occlusion into account. New relief mapping methods are introduced to bridge the gap between distance fields and cone maps. The new methods allow calculating approximate distance fields from their cone map approximate ambient occlusion and soft shadows. The new methods are compared with linear, binary, and interval search as well as variants of cone mapping, such as relaxed cone mapping and quad cone mapping. These methods were evaluated with regards to performance and accuracy and were found to be similar in performance and accuracy than the existing methods. The new methods did not outperform existing methods on the tested scenes, but the new methods make use of approximate distance fields and remove the maximum cone angle limitation. It was also shown that in most cases linear search with interval mapping performed the best, given the error metric used.

Keywords: parallax mapping, relief mapping, displacement mapping, heightmap, ray-tracing, cone step mapping.

Supervisor : Dr. M. Helbig
Department : Department of Computer Science
Degree : Philosophiae Doctor of Computer Science

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

Most modern real-time rendering engines use polygons to represent objects in 3D space because of their flexibility and speed at which polygons can be rendered on hardware. The fidelity of an object can be increased by increasing the amount of polygons that are used to represent the object to capture finer and finer details. But more polygons slow down the rendering process and require more data to be stored and processed.

Texture mapping and normal mapping [2] are ways to increase the apparent detail of polygons. These techniques reduce the amount of polygons required to achieve the same amount of perceived detail. Texture mapping simulates color changing across a polygon and normal mapping simulates the surface changing its normal across a polygon. Parallax mapping [14] was introduced to simulate parallax within a polygon. These techniques progressively bring the effect closer to having real geometery. This apparent geometry increases realism without actually adding more real geometry.

Parallax mapping offsets the position in the texture from where color and normal information is fetched from to simulate parallax of the apparent geometry. Enhancements to the technique also simulate effects, such as occlusion and shadowing within a polygon [21] [28].

1.1 Motivation

Three dimensional distance fields provide very good approximations for effects such as ambient occlusion and soft shadows [30], but their requirement of a 3D texture makes them very expensive with regards to storage, memory and memory bandwidth.

Enhanced versions of parallax mapping, such as Steep Parallax Mapping [28], can still be slow, and thus methods of accelerating the effect were introduced, such as cone step mapping (CSM) [8]. CSM uses only a 2D texture to speed up tracing, but lacks some of the nice properties of distance fields which allows them to approximate certain effects, such as ambient occlusion (AO) and soft shadows [37].

This dissertation examines a technique to modify CSM and other algorithms based on it, to allow for the approximation of a distance field from a height map and the acceleration structure used by CSM. This technique allows the use of the approximate distance field to render effects such as AO and soft shadows bridging the gap between CSM and distance fields and without requiring a distance field to be stored.

1.2 Objectives

The goal of the dissertation is to explore the performance of different methods of rendering the parallax, occlusion, shadowing and AO effects, as well as evaluating the new technique to determine if it is viable and what advantages and disadvantages it has.

1.3 Contributions

This dissertation proposes cone normal stepping (CNS) and the variant relaxed cone normal stepping (RCNS), which are modified versions of CSM and relaxed cone stepping (RCS), respectively [8][29]. These techniques improve on their original versions by removing the limit on cone angles, and allowing conservative approximations of the distance field to be used. Removing the limit on cone angles allows the method to potentially skip faster through space.

1.4 Dissertation Outline

The dissertation is divided into the following chapters:

- **Chapter 2** focuses on the existing methods to simulate parallax and other effects. The chapter starts with the history of parallax mapping, the various methods introduced and what the methods contributed to the field. The methods that are used in the rest of the dissertation are then covered in detail.
- **Chapter 3** describes the new methods which are extensions of CSM in detail.
- **Chapter 4** compares the various methods that were covered in detail to determine the performance and visual characteristics of the methods.
- **Chapter 5** provides a summary of the findings with regards to the performance of both the existing and new methods.

The index may be used as a quick guide to find the methods and other techniques discussed, and can be found on page 65. Acronyms used in this dissertation can be found in Appendix A.

Chapter 2

Background

This chapter provides background information that is required for the remainder of the dissertation.

Section 2.1 covers related work with regards to normal mapping. Section 2.2 goes into detail about some of the first methods to try to achieve apparent geometry including how the methods work. Section 2.3 explains how refinement to the intersection point can be done after an intersection is found. Cone Mapping is one of the main methods for accelerating intersection finding and what the new proposed methods are based on. Section 2.4 covers Cone Mapping in detail. Section 2.5 explores Distance fields and some of the reasons they are useful above other acceleration methods.

Shadowing and Ambient Occlusion is explained in Section 2.6. Section 2.7 covers Curved Surfaces and Silhouettes. In Section 2.8 some other acceleration methods that were not previosuly covered are shown.

Section 2.9 shows how similar techniques in screen-space effects could be used. Section 2.10 covers related work comparing some of the methods mentioned in this chapter.

2.1 Normal Mapping

In 1978 James Blinn presented [2] an algorithm to achieve the impression of much more detailed geometry without any increase in the amount of polygons. This algorithm is more commonly called normal mapping but is also known as bump-mapping. Polygons

usually have only one normal vector per vertex that is interpolated across the polygon or a single normal for the whole polygon. Normal mapping allows mapping normals to the surface of the polygon in the same way as colors are mapped onto the surface with texture mapping. The normals can then change direction much more frequently across the surface of the polygon, allowing lighting to be calculated as if the polygon is uneven and not flat.

Normal mapping significantly increases the apparent complexity of the geometry, but lacks certain characteristics, such as self-occlusion, self-shadowing and parallax. When the viewer moves the geometry it does not seem to behave like real geometry, which is affected by depth and the viewer's viewpoint. The apparent geometry within the polygon still seems to behave like a flat polygon even though the lighting behaves as if the geometry was real. Therefore, this technique only works very well when the apparent geometry is very shallow or the viewer is very far away from the object.

It is cheap to use and only requires an extra normal map and the tangent (and optionally the bitangent) to be passed per vertex. Each pixel to be shaded reads from the normal map a normal that is then transformed into the space of the triangle using the given tangent and bitangent. This new normal is then used instead of the normal of the triangle itself for all lighting calculations.

2.2 Parallax and Self-occlusion

Parallax is the effect of geometry moving at different rates at different distances to the viewer when the viewer moves. Parallax is a hint to the human eye about how far away different parts of the geometry are. Self-occlusion includes the effect of geometry hiding or revealing other geometry of the same primitive when the viewer moves. These aspects are by definition dependent on the viewer's relative position to the geometry.

In 2000 Oliveira et al [25] explained how to warp textures to simulate geometric complexity and occlusion. Their method warped the texture before applying normal texture-mapping using 1D warping functions and was implemented on the central processing unit (CPU) and therefore performance was lacking.

Most methods before 2001 relied on processing textures on the CPU, but with the

introduction of pixel shaders in commercial hardware more work could be shifted to the graphics processing unit (GPU) [13][14].

Kaneko et al [14] introduced Parallax Mapping which used a simple texture offset that was calculated completely on the GPU. The method is extremely fast and is very close to pure texture mapping. However, since the calculation is only an approximation there is a lot of distortion in the resultant image if the viewing angle is close to grazing or the heightmap does not have smooth transitions. The viewing angle is grazing if the viewing direction is perpendicular to the normal of the polygon.

In 2002 the Relief Texture Mapping method was partially implemented on the GPU and unlike the Parallax Mapping of Keneko et al [14], the method was relatively accurate. Instead of pre-warping the texture on the CPU, only the offsets were generated on the CPU and the GPU did the warping [13]. One advantage of this approach when compared to the original Relief Texture Mapping method is that only one offset map has to be generated by the CPU and the GPU can use the offset map to transform multiple textures, including a normal map. Combined with normal mapping, this method allows self-occlusion and parallax, as well as relief shading. Reflection mapping of the apparent geometry is also possible by way of sampling an environment cube map. Even though this method is implemented in hardware, it was not yet much faster than the software method executed on the hardware of the time.

Welsh [40] improved on the Parallax Mapping of Keneko et al [14]. The original offset calculation introduced a lot of distortion at grazing angles, because of its depth approximation. Welsh [40] mitigated this by adding a term that flattens the apparent geometry at grazing angles. His approach addressed the distortion at grazing angles at the expense of parallax and also reduced the amount of calculations that were required to calculate the effect in the process.

Policarpo et al [28], McGuire et al [23] and Brawley et al [3] introduced ray-tracing for calculating the intersection point instead of the offset calculation of Welsh [40]. The method by McGuire et al [23] uses only a linear search on a mip-mapped height texture and normal smoothing step to reduce the step artifacts produced by the linear stepping. The method by Policarpo et al [28] introduced the usage of a fixed or dynamic (dependent on hardware) number of linear steps followed by a binary search of the height-map. In

addition, a method of representing full 3D objects using dual-depth relief textures was proposed. This method captures a depth value from the reverse side of the polygon so that the profile of the object can be captured as well.

Methods like 'Relief Texture Mapping' [25] were implemented on the CPU which made the methods unusably slow for real-time use in complex scenes. Methods that are implemented on the CPU are limited by the bandwidth between them and the GPU and CPUs are often busy with other tasks as well. GPUs are also better suited to these types methods. As such we are only considering the GPU-based methods for the rest of this dissertation. It is also assumed that all the primitives are triangles. The rest of the methods can all be represented by some form of ray-casting. These methods were usually implemented in the fragment shader, but some of the original versions could be implemented using the fixed-functionality of the GPUs of the time.

For each fragment of the triangle to be rendered, an eye ray is calculated in the texture space of the triangle. The space is represented by the 2D UV coordinates of the texture, as well as a third coordinate between 0 and 1 representing the depth into the triangle. This will be referred to as the local triangle space. The depth coordinate can assign any value between 0 and 1 as the surface of the triangle itself. The ray is then traced into the primitive to determine where it intersects with the apparent geometry. This intersection is then used to look up the attributes of the material required to shade the fragment, usually including a diffuse and a normal map. The intersection T_n is determined by calculating the t value in the following equation:

$$\mathbf{p}_n = \mathbf{p}_o + t\mathbf{d} \tag{2.1}$$

where $\mathbf{p}_o$ is the point where the eye ray enters the local triangle space, $\mathbf{p}_n$ is the intersection point and $\mathbf{d}$ is the direction of the eye ray. The x and y cordinates of $\mathbf{p}_o$ are the original UV coordinates and are also represented by T_o. The x and y coordinates of $\mathbf{p}_n$ are the final UV coordinates and are represented by $\mathbf{t}_n$.

Nearly all the methods require a height map that represents the height of the apparent geometry in the local triangle space. The x and y coordinates are in the UV space of the triangle. The height value is in the space of the depth coordinate and represents the height of the apparent geometry at that point in the triangle. The value, which is represented by h, is usually transformed so that it is zero at the surface of the triangle

and scaled to change the height of the apparent geometry relative to the normal. This is done using the equation:

$$h_{sb} = sh + b \tag{2.2}$$

where h is the height value from the height map and s and b are the scale and bias respectively. The result h_{sb} is used when intersecting with the ray.

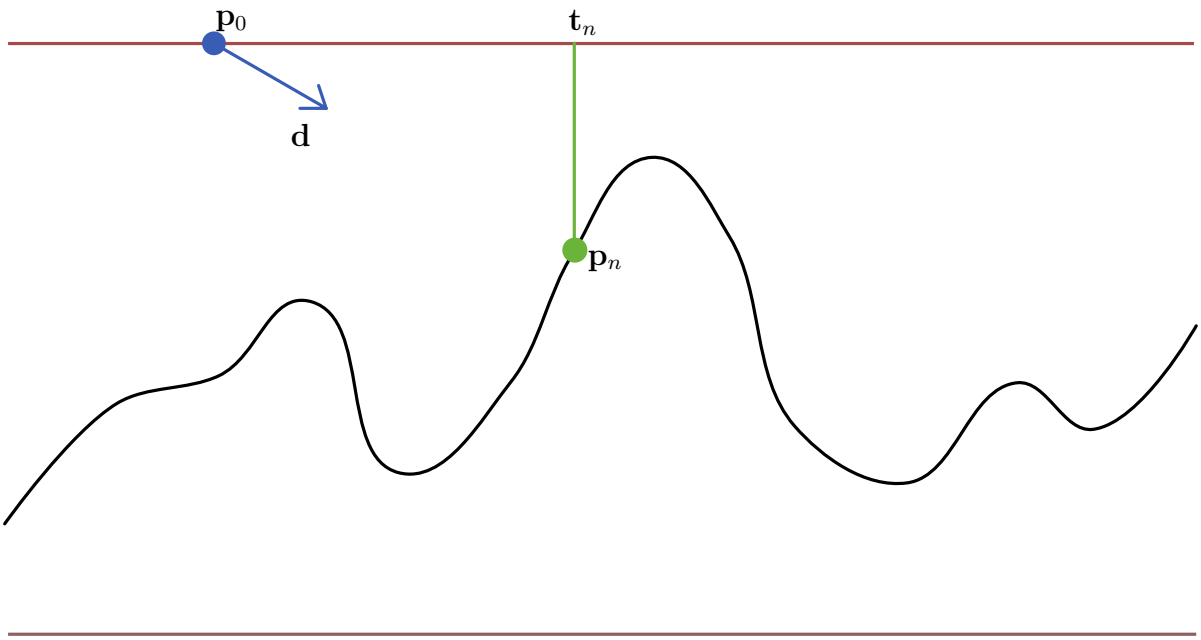


Figure 2.1: The eye ray and the expected intersect position.

Given the ray, defined by $\mathbf{p}_o$ and $\mathbf{d}$, the intersection point on the heightmap must be calculated, as seen in Figure 2.1. The methods to calculate the intersection point can be split into two categories, namely those that only use the height map and those that require generating a separate acceleration structure.

The simplest method is that of parallax mapping [14]. It provides a reasonable approximation of parallax, but can cause much distortion depending on the viewing angle and the characteristics of the height map. The method fetches the height value for the point where the ray enters the triangle and assumes that the apparent geometry is a plane parallel to the triangle at the fetched height. The ray intersects with the plane,

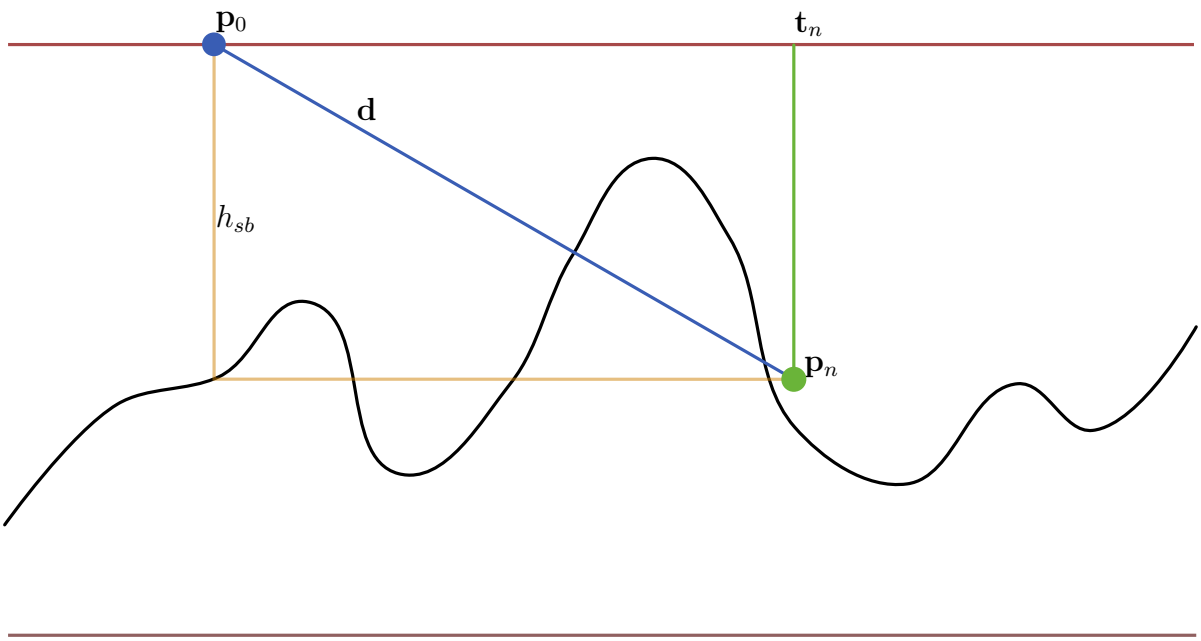


Figure 2.2: Parallax mapping.

returning a new UV coordinate. The value of t is calculated according to the following equation:

$$t = \frac{h_{sb}}{d_z} \tag{2.3}$$

where h_{sb} is defined in Equation 2.2 and $\mathbf{d}$ is the direction of the eye ray and the subscript indicates the component of the vector. For example, d_z indicates a scalar equal to the z component of the vector $\mathbf{d}$.

Equation 2.3 and Equation 2.1 together produce the following equation:

$$\mathbf{p}_n = \mathbf{p}_o + \frac{h_{sb}}{d_z}\mathbf{d} \tag{2.4}$$

Equation 2.4 with the definition of $\mathbf{t}_n$ and $\mathbf{t}_o$, produces:

$$\mathbf{t}_n = \mathbf{t}_o + \frac{h_{sb}}{d_z}\mathbf{d}_{x,y} \tag{2.5}$$

where $\mathbf{d}_{x,y}$ represents a vector of two components consisting of the x and the y components of $\mathbf{d}$.

If the resultant UV coordinate is outside of the triangle's space, the fragment can be discarded if it was not meant to wrap.

Parallax mapping works well enough with smooth heightmaps with no drastic changes in height. If the viewer looks at the triangle at increasingly grazing angles the method causes lots of distortion. This was somewhat fixed by Welsh [40] by applying an offset limit. The method was still somewhat expensive at the time so the limit was implemented by removing the division by d_z in Equation 2.5 so that it becomes:

$$\mathbf{t}_n = \mathbf{t}_o + h_{sb}\mathbf{d}_{x,y} \tag{2.6}$$

This causes the parallax effect to be faded out at increasingly grazing angles, avoiding most of the distortion and reducing the effect.

The linear search, defined in Algorithm 1 and visualized in Figure 2.3, finds the intersection by taking *linearSteps* equally sized steps and stopping when the ray dips below the surface. The step size, *stepSize*, is calculated so that the ray will reach the lowest possible height after all *linearSteps* have been taken.

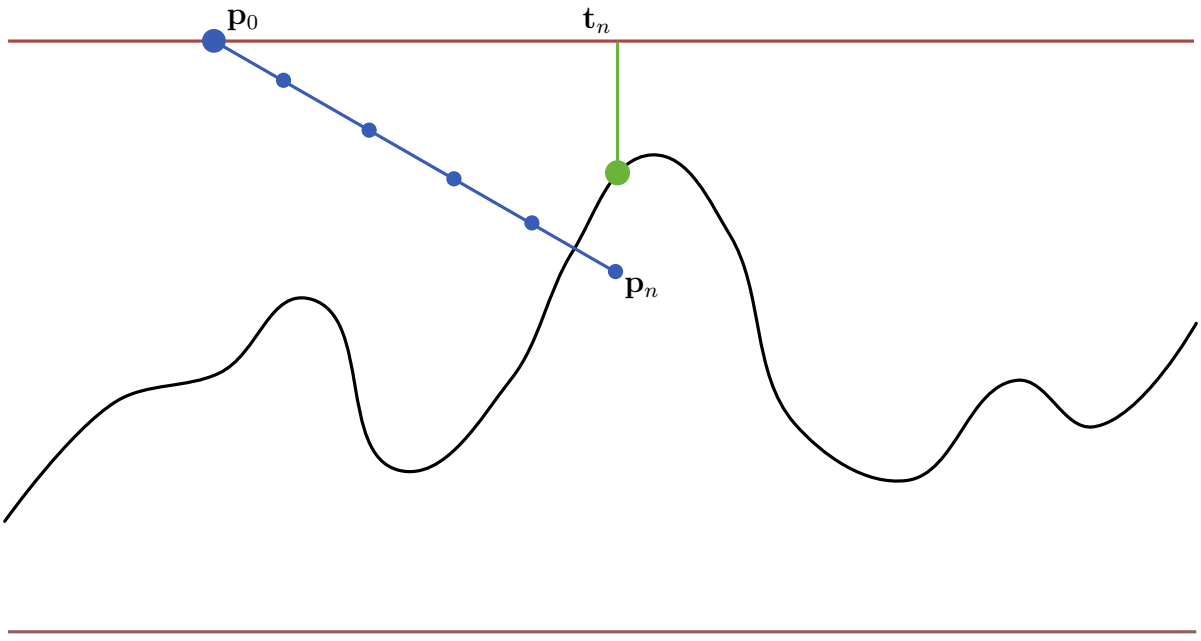


Figure 2.3: Linear search.

Algorithm 1 Linear search.

$\mathbf{p} \leftarrow \mathbf{p}_o$
$t \leftarrow 0$
$stepSize \leftarrow 1.0 \div r_d.z \div linearSteps$
for $i \leftarrow 1$ to $linearSteps$ **do**
  $h \leftarrow$ GETHEIGHT$(\mathbf{p}_{xy})$
  if $h \geq p_z$ **then**
    return p
  else
    $t \leftarrow t + stepSize$
    $\mathbf{p} \leftarrow \mathbf{p}_o + t\mathbf{d}$
  end if
end for

2.3 Intersection Refinement

Most methods use one method for finding a rough intersection point and then use another method for refining the intersection point, called the coarse search and refining search respectively from now on.

The method by Policarpo et al [28] uses full ray-casting to find the intersection point. It starts by using linear search as the coarse search and then binary search for the refining search. The binary search method, defined in Algorithm 2 and visualized in Figure 2.4, refines the intersection point by using the classic binary search algorithm. The algorithm starts by initializing t to the value found by the coarse search (linear search in most cases) and then taking half a step back. The algorithm then either goes forward or backwards a quarter step, depending on whether that point is above or below the surface respectively. This process is repeated, each time taking half the step size of the previous iteration. Each step on average reduces the error by half.

Binary search is a more efficient algorithm for finding the intersection point, but it cannot be used on its own as it can miss features even with an infinite amount of refinement steps. The linear search ensures that features, that are at least as thick as the step size, are not missed.

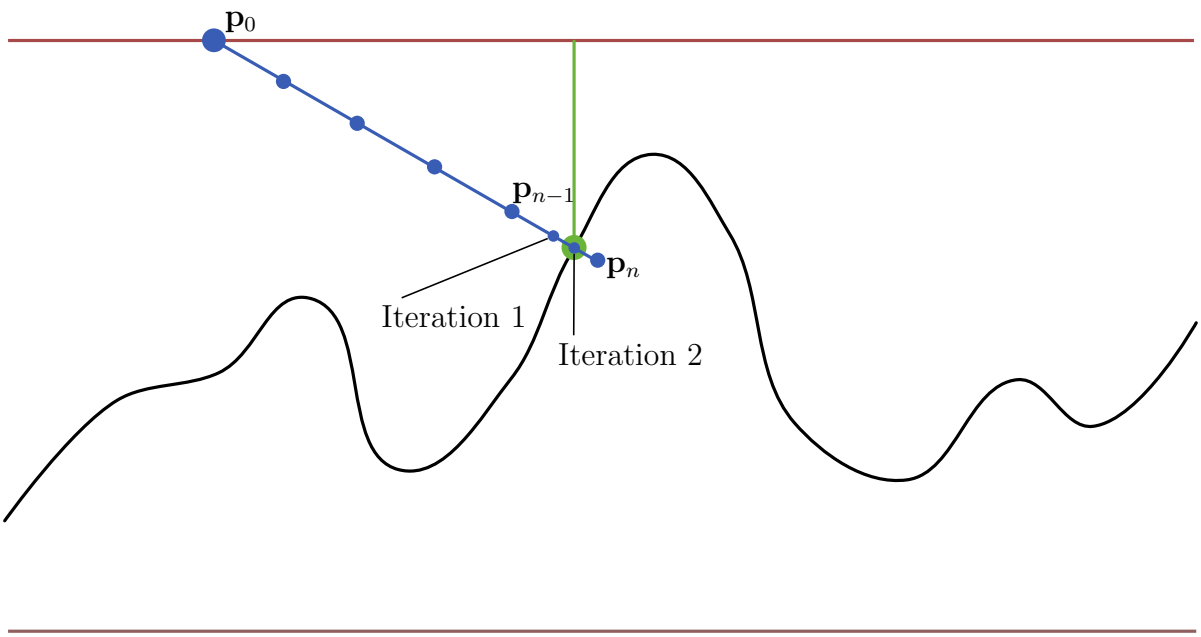


Figure 2.4: Linear search followed by binary search.

Algorithm 2 Binary search.

$s \leftarrow stepSize \div 2$
$t \leftarrow linearT - s$
$\mathbf{p} \leftarrow \mathbf{p}_o + \mathbf{d} \cdot t$
for $i \leftarrow 1$ to $binarySteps$ **do**
    $h \leftarrow$ GETHEIGHT($\mathbf{p}.xy$)
    if $h \geq p.z$ **then**
        $s \leftarrow -s \div 2$
    else
        $s \leftarrow s \div 2$
    end if
    $t \leftarrow t + s$
    $\mathbf{p} \leftarrow \mathbf{p}_o + \mathbf{d} \cdot t$
end for

Risser et al [31] introduced a technique called interval mapping. Normally the intersection to the height field would be found with linear search and then refined using binary search. Interval mapping uses the last two heights sampled from the linear search to do a piece-wise approximation of the height field resulting in faster convergence and less noticable artifacts due to C_0 continuity. A function has C_0 continuity if it is continuous or does not suddenly change in value without going through in between values.

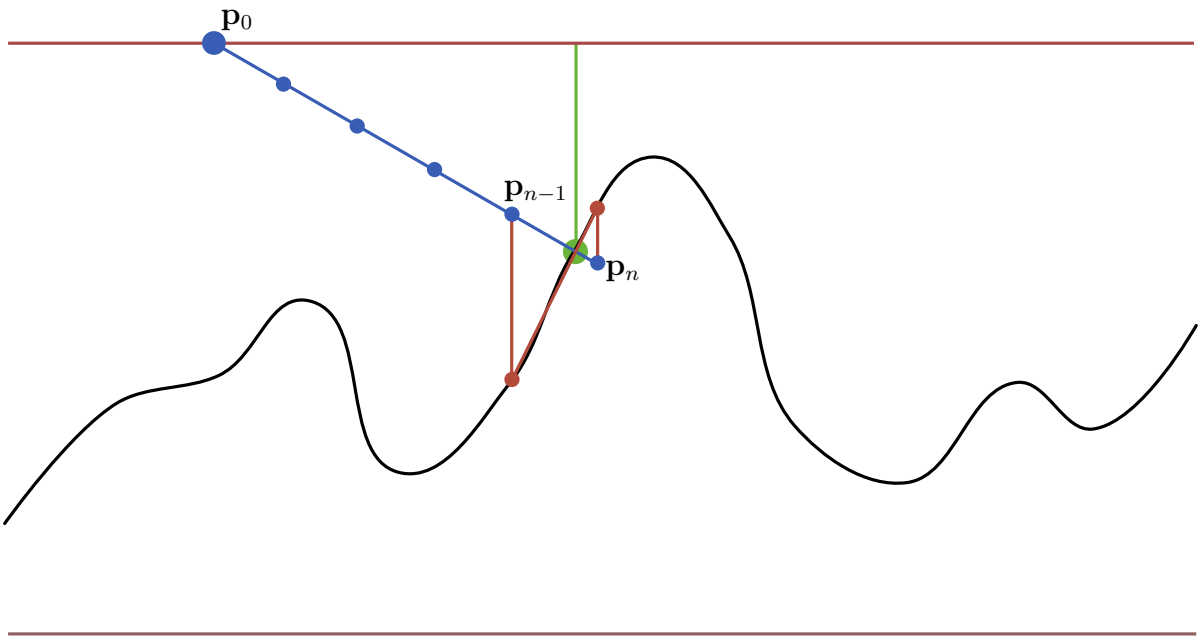


Figure 2.5: Linear search followed by interval search.

Tatarchuk et al [37] proposed an improved height-map intersection approach, level of detail (LOD) techniques, as well as soft-shadowing and a dynamic sampling rate. The proposed intersection method is very similar to interval mapping.

Risser et al [31] introduced a more efficient algorithm for the refining search by approximating the height map as a piece-wise linear function. This approach decreases the amount of iterations required to achieve the same result as binary search and the algorithm mostly achieves C_0 continuity of the calculated intersections which decreases the obviousness of the distortion. This search is referred to as Interval search, defined in Algorithm 3 and visualized in Figure 2.5. This method requires the t value before and

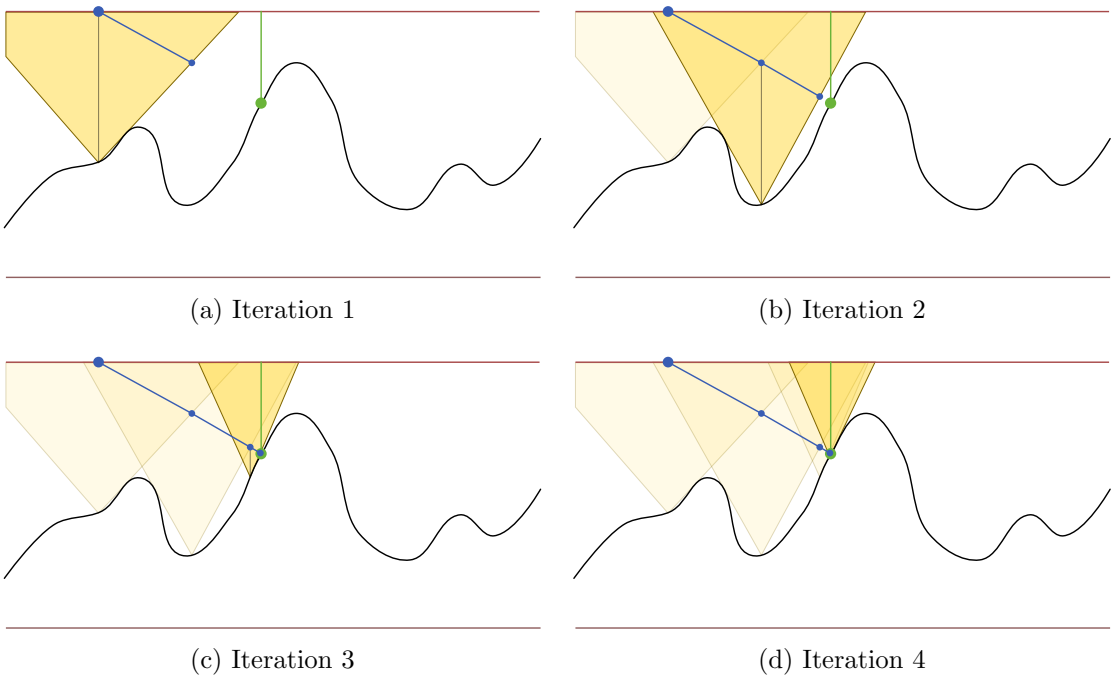
Algorithm 3 Interval search.

$\mathbf{p}_1 \leftarrow \mathbf{p}_o + \mathbf{d} \cdot t_1$
$\mathbf{p}_2 \leftarrow \mathbf{p}_o + \mathbf{d} \cdot t_2$
$h_1 \leftarrow \text{GETHEIGHT}(\mathbf{p}_1.xy)$
$h_2 \leftarrow \text{GETHEIGHT}(\mathbf{p}_2.xy)$
for $i \leftarrow 1$ to $binarySteps$ **do**
  $f \leftarrow \frac{h_1 - p_1.z}{(p_2.z - p_1.z) - (h_2 - h_1)}$
  $t_{int} \leftarrow f(t_2 - t_1) + t_1$
  $\mathbf{p}_{int} \leftarrow \mathbf{p}_o + t_{int}\mathbf{d}$
  $h_{int} \leftarrow \text{GETHEIGHT}(\mathbf{p}_{int}.xy)$
  if $h_{int} \geq p_{int}.z$ **then**
    $\mathbf{p}_1 \leftarrow \mathbf{p}_{int}$
    $t_1 \leftarrow t_{int}$
    $h_1 \leftarrow h_{int}$
  else
    $\mathbf{p}_2 \leftarrow \mathbf{p}_{int}$
    $t_2 \leftarrow t_{int}$
    $h_2 \leftarrow h_{int}$
  end if
end for

after the intersection is found by the coarse search, defined by t_1 and t_2 respectively.

2.4 Cone Mapping

CSM [8] provided a technique that improved on the performance of the ray tracing, by precomputing a texture that can be used to skip through empty space but that would not skip intersections. However the number of steps must be limited, because it may never quite reach the intersection point, which leads to some distortion. CSM also has the problem of requiring the generation of the cone data that cannot be done in real-time and therefore is not suitable for dynamic surfaces such as water. In the game Simcity (2013), CSM was used for all building facades [32].



(a) Iteration 1 (b) Iteration 2

(c) Iteration 3 (d) Iteration 4

Figure 2.6: Cone Step Mapping.

CSM [8] is a coarse search method that requires an acceleration structure. Each height value in the heightmap has a corresponding cone ratio r that together define a cone positioned with its point $\mathbf{p}$ on top of the heightmap. The cone $\mathbf{c}$ is defined as:

$$c_z = \frac{\|\mathbf{c}_{xy} - \mathbf{p}_{xy}\|}{r} + p_z \tag{2.7}$$

The 'cone map' stores $\sqrt{r}$ to increase precision of smaller values as they are usually stored in an 8-bit format.

The search, defined in Algorithm 4 and visualized in Figure 2.6, is performed by intersecting the ray with the cone given at the current position. This process is repeated until either the maximum amount of steps have been taken or the current position is close enough.

RCS [29] improved on CSM by further reducing the average number of iterations required to find an intersection. RCS achieved this by allowing the precomputed cone map to intersect with the height field, i.e. the cones can be much wider for skipping space quickly. Techniques such as binary search can then be applied, as a point below

Algorithm 4 Cone Step Mapping.

$\mathbf{p} \leftarrow \mathbf{p}_o$
$t \leftarrow 0$
for $i \leftarrow 1$ to *coneSteps* **do**
  $h \leftarrow$ GETHEIGHT$(\mathbf{p}_{xy})$
  if $h \geq p_z$ **then**
    return p
  else
    $r \leftarrow$ GETCONERATIO$(\mathbf{p}_{xy})^2$
    $iz \leftarrow \sqrt{1 - r_d.z^2}$
    $s \leftarrow \frac{p.z-h}{\frac{iz}{r}-r_d.z}$
    $t \leftarrow t + s$
    $\mathbf{p} \leftarrow \mathbf{p}_o + \mathbf{d} \cdot t$
  end if
end for

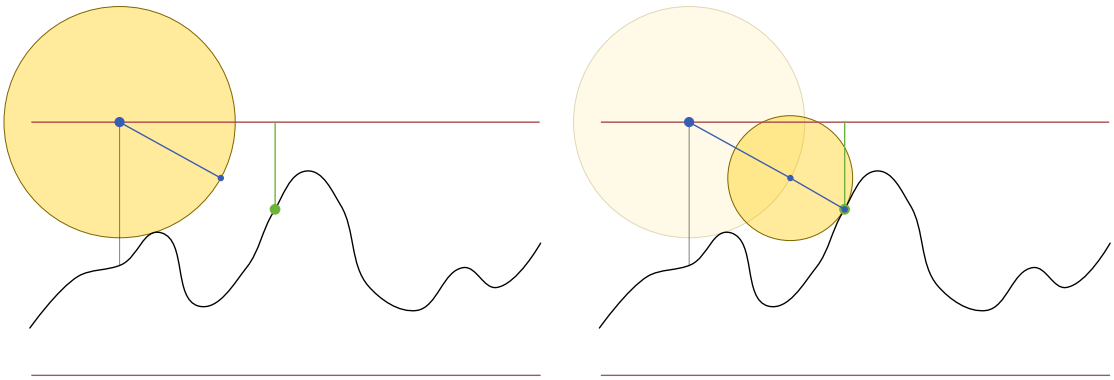
and above the heightmap can be found, which allows for a much more robust way of calculating intersections. Quad cone step mapping (QCSM) [29] was also introduced, which generates a cone map for 4 major directions instead of only the one cone map. This enables faster space skipping, but requires more storage, memory and bandwidth. The results show that the RCS method is superior to CSM.

Anistropic cone mapping (ACM) [5] improves on the CSM method by generating a cone map for more than one direction and is very similar to QCSM. However, ACM does allow the amount of directions to be chosen. Lee. et al [17] proposed a method where for each pixel of the heightmap a cone is projected down and the lowest and highest point within the cone are stored. The data is then used to limit the search to this range when tracing. The algorithm is very dependent on the angle chosen for the cone and artifacts can occur if the traced ray is outside the cone.

In 1996 Paglieroni et al [27] introduced the height distributional distance transform (HDDT). HDDT is a generalization of CSM, where the free-space skipping volumes are only cone-like. A bounding curve is defined and then swept around the cone apex. These shapes are cone-like in that their shape is defined by a circle starting with a radius of

zero around an axis and increasing in size as it is swept along that axis in one direction. The bounding curve defines th e radius of the circle as it is swept along the axis. If the bounding curve is linear and not piece-wise, then it is a cone and thus identical to CSM. HDDT was only, however, implemented on a CPU, where as CSM is a GPU technique.

2.5 Distance Fields



(a) Iteration 1 (b) Iteration 2

Figure 2.7: Ray-marching a distance field.

Donnelly [7] used a 3D volume texture that contains a distance field describing the geometry. The distance field is ray-casted in the pixel shader, and is used to accelerate the ray when it is far away from geometry. It has several advantages over other methods, including having nearly no artifacts and it can represent geometry not supported by height-maps. Unfortunately, the 3D volume texture requires a lot of video memory, limiting the technique to only a few objects at a time.

Figure 2.7 shows how a distance field is ray-marched. Each iteration the current ray position is looked up in the distance field. The resultant value in the distance field indicates the distance to the closest point on the surface, thus the ray can safely move that distance forward in the direction of **d**. The process is then repeated until a certain number of steps or until the returned distance is considered to be close enough.

2.6 Shadowing and Ambient Occlusion

Shadows are a very important part of realistic graphics. Normal mapping as described in Section 2.1 does not take into account any shadowing caused by the apparent geometery of the surface itself.

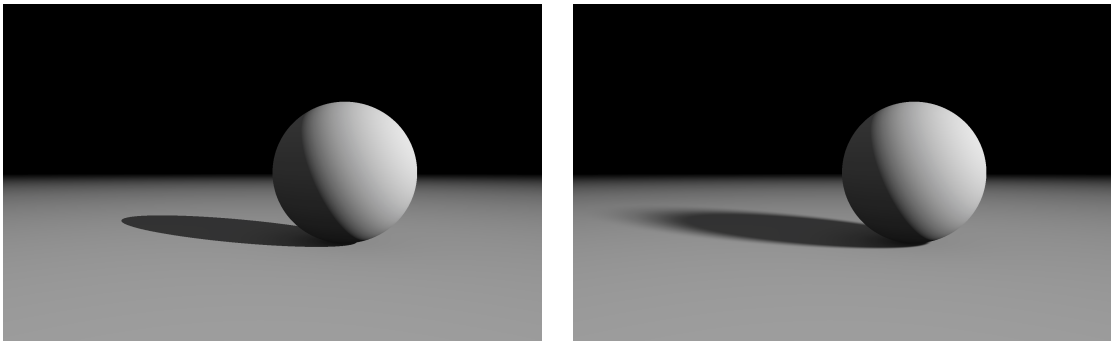
In 1988 a technique called Horizon mapping [21] was introduced, which allowed shadowing of bump-mapped surfaces. The technique is likened to recording at what time each part of the landscape gets shadowed as the sun sets. Unfortunately this requires recording the information for multiple directions as the light can come from any direction. In 2000 [34] this technique was implemented on a GPU to improve speed. The horizon mapping technique was extended to take into account surface curvature when computing shadows using the horizon mapping technique [26].

Kautz et al [15] introduced a new technique for shadowing of bump maps, called an ellipse shadow map. For each pixel of the bump map many ray tests are performed to determine which part of the hemisphere around a pixel has blockers. The resultant data is then fitted with an ellipse stored in a separate map. At run time the light direction is compared with the ellipse to determine if that pixel is shadowed. A soft shadowing method was also introduced. Later the same technique was extended with indirect lighting [10].

Before graphics cards were fast enough to do ray-casting methods such as Horizon Mapping were used [21][34]. However, nowadays ray-casting is the main shadowing method used in parallax maps.

The same methods that are used for the coarse search of occlusion can be used for shadowing. The refinement search is not needed as it is unnecessary to know where the intersection point is, only that there is one.

Soft-shadowing is the term used to refer to the techniques that try to emulate the fact that shadows do not have a hard transition but have a region in between called the penumbra which is where the light is partially occluded. The effect can be seen in Figure 2.8 [11]. Soft-shadowing can be approximated using techniques discussed in [37] for height-maps or in [30] for distance fields. Both approaches are very cheap in terms of computing power as they do not do any extra texture fetches and consist of only a few more instructions per pixel. The soft-shadowing technique of Tatarchuk et al [37] traces



(a) Hard shadow (b) Soft shadow

Figure 2.8: Soft shadowing.

a ray from the intersection point to the light source to find occluders, but also calculates the amount of occlusion by tracking how far the ray travelled into the occluders on the heightmap.

Other effects, such as ambient occlusion, can be calculated for height maps [18], as well as distance fields [9]. Distributed ray-tracing techniques [6] can be used to calculate shadowing and ambient occlusion more accurately, but are a lot more expensive to compute.

2.7 Curved Surfaces and Silhouettes

Jeschke et al [12] proposed several techniques to enable relief mapping on curved surfaces. Shell mapping allows smooth mapping of the apparent detail across a surface, even when the surface is curved. Curved shell mapping further enhances the technique to account for curvature within a polygon. Kharlamov et al [16] introduced a method referred to as silhouette clipping to render the silhouette of objects as an extension to relief mapping. This method is based on the original silhouette clipping method introduced by Sander et al, originally developed to render smooth silhouettes for non-parallax mapped objects [33].

The Crytek 3 game engine used parallax occlusion mapping (POM) for ground details and Silhouette POM for trees [35]. Chen et al [4] introduced a silhouette rendering method that does not require the use of prisms by continuously bending the ray at

each step while finding an intersection. Luo et al [19] introduce dual-space ray casting to allow accurate parallax mapping on curved surfaces and an analytical model for spherical surfaces. An approach that uses multiple layers of relief mapping at different scales, as well as texture-space ambient occlusion, was proposed in [18]. The approach increases the amount of detail that can be represented by the relief maps, without increasing the storage requirements too much.

2.8 Other Acceleration Structures

Tevs et al [38] proposed maximum mipmaps, another approach for accelerating the ray traversal. The maximum mipmaps allow the ray to skip through the space above the heightmap, descending to a more detailed level when necessary. The method also uses a ray for bilinear patch intersection on the lowest level without any further binary/interval search and therefore has negligible artifacts.

Ali et al [1] introduced a few methods specially suited to rendering facades of buildings. The first method uses boxes instead of spheres or cones for the acceleration structure. The second method compresses this new acceleration structure and allows rendering directly from the compressed version.

2.9 Screen-space techniques

Techniques very similar to relief mapping can also be used in screen-space where the depth buffer of the scene is the heightmap. Screen-space techniques are used to simulate arbitrary reflections, shadowing and global illumination effects [20][22][39][41][42]. In 2013 Michal Valient [39] summarized their use of screen-space reflections in their technology demo 'Kill Zone: Shadow Fall'. In 2014 McGuire et al [22] introduced an efficient method of performing screen-space tracing by using the 3D Digital Differential Analyzer (DDA) algorithm instead of linear search, as is typically using by relief mapping.

Wronski [41][42] discussed some of the disadvantages of screen-space reflections. One of the biggest disadvantages is the lack of information, as scenes are not well represented by a single height map. However, the technique can be made to work decently nonethe-

less. Mara et al [20] used depth-peeling techniques to extract more information from the rendered scene to improve the quality of screen-space ray tracing.

2.10 Summary

Comparisons of all of the mentioned methods can be found in other papers as well. Ohrn et al [24] summarized some of the techniques used to add more apparent detail to surfaces, including normal mapping, parallax mapping and displacement mapping. Szirmay-Kalos et al [36] covered a large percentage of the methods mentioned including displacement mapping using vertex shaders.

This chapter explored most of the methods related to parallax mapping, including CSM which the new proposed methods are based on. The new proposed methods, CNS and RCNS, will be explained in the next chapter.

Chapter 3

Cone Normal Stepping

This chapter introduces the new methods and the enhancements to existing methods that approximates a distance field and remove the cone angle limitation from the original CSM.

Section 3.1 introduces a new way of representing the cone map in CSM [8], which allows easy calculation of an approximate distance to nearby geometry as defined in Section 3.2. The new representation does require a new intersection equation, which is presented in Section 3.3.

3.1 Cone Normal

The original CSM [8] and distance fields for relief mapping [7] are actually not that different. Both methods give a conservative distance to step before intersecting the surface, but CSM takes advantage of the fact that the surface is a heightmap. Distance fields represent the closest distance to the surface as a sphere, while a cone step map represents the distance as a cone and only requires a single cone per point on the heightmap. Figure 2.6 shows how a cone map is traversed and Figure 2.7 shows how a distance field is traversed for comparison.

The original CSM method proposed storing a cone ratio that is equivalent to the slope of the side of the cone, similar to the classic formula for a line, $y = mx + c$, except x is replaced by z, y is replaced by the distance along the x and y coordinates and m is

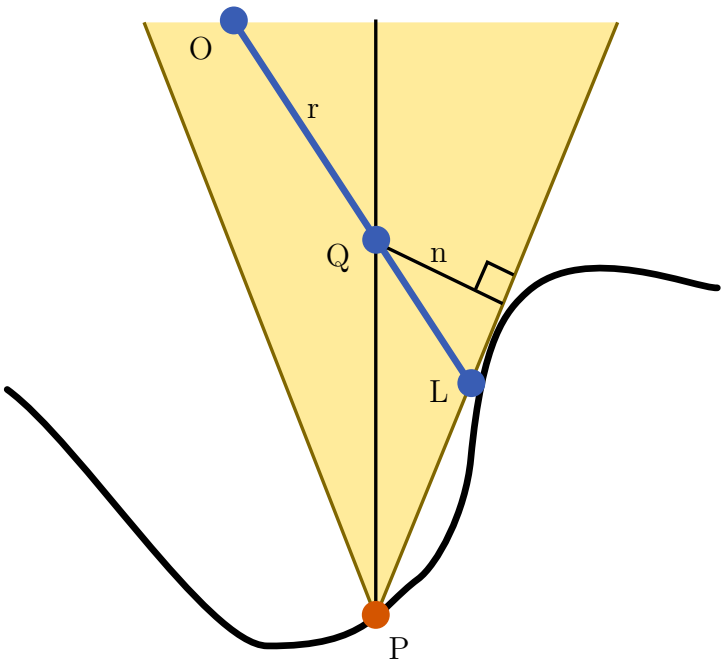


Figure 3.1: Cone defined by a normal.

the cone ratio or r as can be seen in Equation 2.7. One problem with the original formula of a line is that m approaches infinity as the line becomes more vertical or as the cone becomes wider. The original method caps the value of r to 1. The cap can be increased or decreased to trade precision (because the value is quantized) for the maximum angle of the cone that is represented by the cone map.

It is proposed in this dissertation that instead of storing the cone ratio, to instead store the normal of the cone sides. The cones are always pointing upwards in the z direction and thus flatten the problem from a 3D problem with a cone to a 2D problem with two lines by slicing along the plane created by the apex of the cone and the ray.

In Figure 3.1 it can be seen how the cone is defined. P is the apex of the cone that is always at the same height as the heightmap at that point. r is the ray originating from O and passing through Q, which is where the current query position is. L is the point of intersection with the cone from Q in the direction of r and n is the 'normal' of the line. n is normalized and perpendicular to the cone sides. As n is normalized it is only necessary to store one component and regenerate the missing components if needed. n_z was chosen as the component to be stored to simplify calculations.

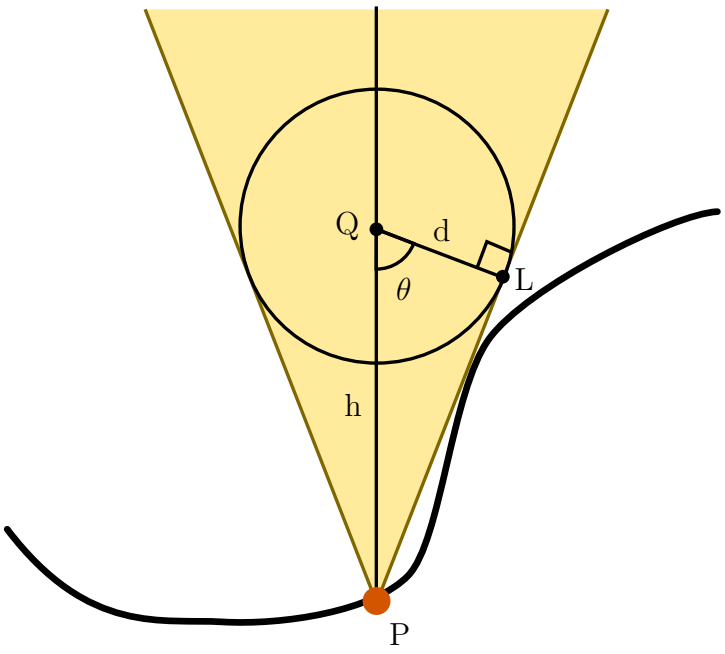


Figure 3.2: Calculating approximate distance from cone.

3.2 Distance field approximation

Distance fields have several advantages in approximating effects like soft-shadowing and ambient occlusion as they provide information about the nearby geometry. Using the modified definition of cone mapping as defined in Section 3.1, a conservative approximation of the distance field can be calculated. The approximation tries to fit the largest sphere centered at the current location in the cone. Equation 3.1 and Figure 3.2 show how this can be calculated. d is the radius of the fitted circle and the approximate distance to the closest geometry.

$$
\begin{aligned}
d &= \|\mathbf{PQ}\| \cos\theta \\
&= \frac{\|\mathbf{PQ}\|\|\mathbf{LQ}\| \cos\theta}{\|\mathbf{LQ}\|} \\
&= \frac{\mathbf{PQ} \cdot \mathbf{LQ}}{\|\mathbf{LQ}\|} \\
&= \mathbf{PQ} \cdot \mathbf{n} \qquad (\mathbf{n} \text{ is } \mathbf{LQ} \text{ normalized}) \\
&= \langle 0, h \rangle \cdot \langle n_{xy}, n_z \rangle \\
&= h n_z
\end{aligned}
\tag{3.1}
$$

The approximate distance can then be used with the distance field methods described in Section 2.6 to approximate soft-shadowing and ambient occlusion effects.

3.3 Intersection

To calculate the intersection with the cone defined using its normal, a ray-plane intersection is calculated as indicated by Equation 3.2. The origin is the cone apex. $\mathbf{p}$ is the current ray position relative to the cone apex, therefore $\mathbf{p}$ is of the form $\langle 0, 0, p_z \rangle$ as the current ray position is directly above the cone apex. A plane is defined by $ax + by + cz + d = 0$, where for the cone $\langle a, b, c \rangle = \mathbf{n}$ and $d = 0$ as the plane passes through the origin. $\mathbf{d}$ is the ray direction and t is the size of the step for this iteration.

$$
\begin{aligned}
t &= \frac{-(\mathbf{p} \cdot \mathbf{n} + d)}{\mathbf{n} \cdot \mathbf{d}} \\
&= \frac{-(\langle 0, 0, p_z \rangle \cdot \langle n_x, n_y, n_z \rangle + 0)}{\mathbf{n} \cdot \mathbf{d}} \\
&= \frac{-p_z n_z}{\mathbf{n} \cdot \mathbf{d}} \\
&= \frac{-h n_z}{\mathbf{n} \cdot \mathbf{d}}
\end{aligned}
\tag{3.2}
$$

The approximate distance, $d = hn_z$, as defined in Equation 3.1, is a byproduct of calculating the intersection.

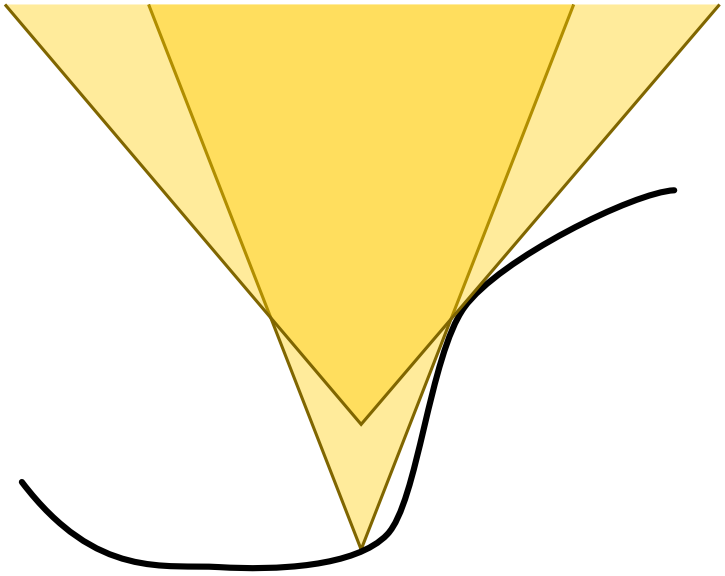


Figure 3.3: Method 1: Multiple cones at different height.

3.4 Multiple cones

Close to the edges of features in the heightmap the cone map produces very narrow cones. Narrower cones lead to smaller steps in ray traversal, which usually result in more iterations being required before the intersection is found. Deriving the distance field from the cone, as discussed in Section 3.2, leads to drastically under-estimated distance values in the upper area of the narrow cones.

The distance field created by a heightmap increases monotonically along the positive z ('up') axis. This is due to the fact that given any other point on the heightmap, if the current point is below the other point then the distance stays the same as the current point moves 'up' until it is at the same height as the other point. Otherwise, when the current point is above the other point, moving it up causes it to move away from the other point, thus increasing distance. As the distance field increases monotonically, cones are a good approximation for the distance field. Using multiple cones allow fitting the field more accurately.

The first method, shown in Figure 3.3, defines multiple cones at different heights above the heightmap. One cone is placed on the heightmap in the same way as in the original CSM method. When calculating the intersection or approximating distance, the

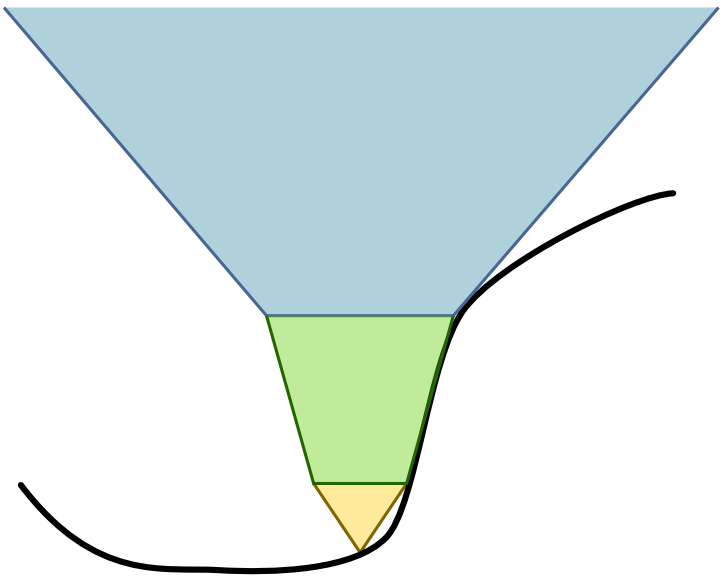


Figure 3.4: Method 2: Multiple piecewise cones.

cone that will lead to the largest step or largest distance respectively is selected. This method is relatively easy to implement, but can only properly approximate distance fields where the derivative of the distance is increasing monotonically. This is due to the fact that extra cones can only widen the area that the cone covers.

A second possible method, shown in Figure 3.4, uses piecewise cones, where only one cone defines for a range of heights and each successive cone continues where the last cone ended, but at a new angle. This method was not evaluated in this dissertation and is a candidate for future work.

In this chapter the new methods were introduced in detail and how the methods can allow for approximating a distance field. The next chapter investigates the performance of the methods introduced in this chapter, as well as the existing methods discussed in Chapter 2.

Chapter 4

Experiments

This chapter examines the performance of existing methods, as well as the methods introduced in the previous chapter. Comparisons are made between the methods to evaluate the differences in performance between these methods.

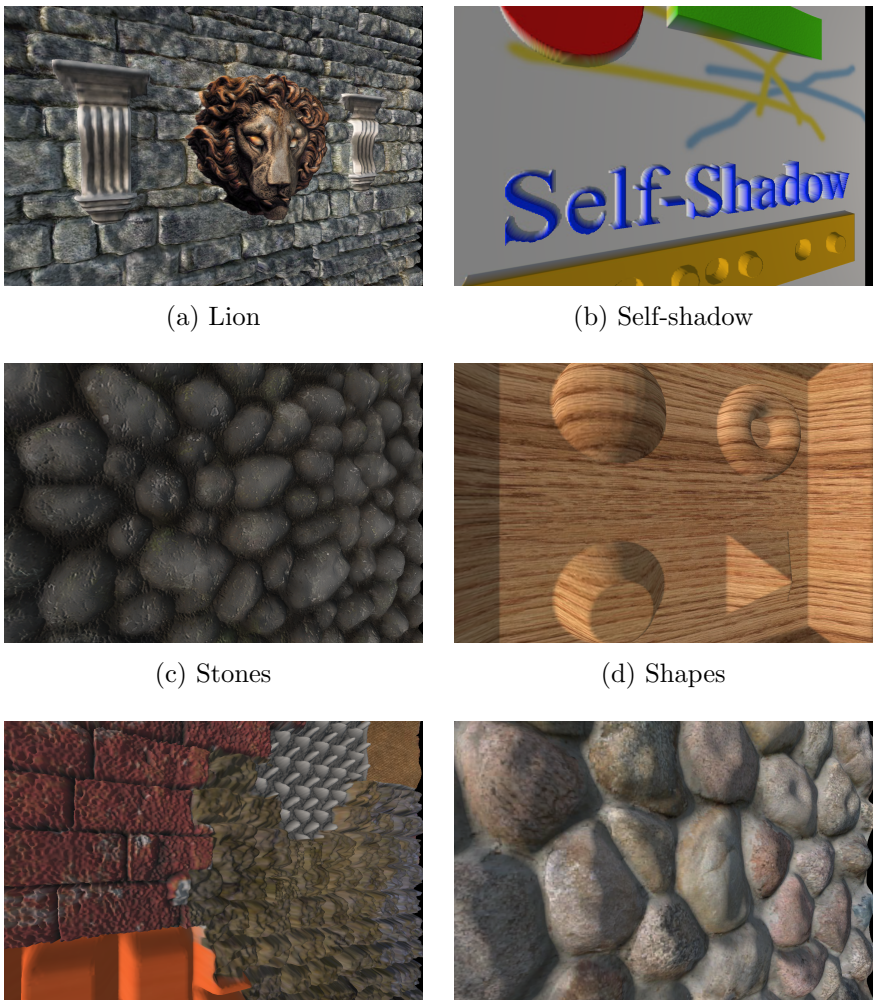
In Section 4.1 the setup used for the experiments are explained, as well as the calculation of the metrics. Section 4.2 explores the results of methods that only use a heightmap to find an intersection. Section 4.3 explores the results of methods that require a separate acceleration structure, including the new (proposed) methods. Section 4.4 and Section 4.5 look at the results from the shadowing and ambient occlusion effects respectively, optionally using the cone map to approximate the effects.

4.1 Experimental Setup

The testing program is written in C++ and uses OpenGL 4.5 for rendering. The parallax mapping methods are implemented as a fragment shader written in GLSL. The program was executed on a Linux machine with a NVIDIA GTX 780 GPU and an Intel i5 3750K CPU. The NVIDIA 384.98 driver was used for the GPU and GNU compiler collection (GCC) 7.2.1 as the C++ compiler. The Linux kernel 4.13.11 was used.

The recorded frame times are averaged over 500 frames. The final mean squared error (MSE) is calculated by capturing the rendered scene, where each pixel contains the error for that pixel. The captured error values are squared and summed to produce

the final metric. The error per pixel is calculated by subtracting the intersection point's height from the height sampled from the height map at that point. The error metric is not perfect as it does not take into account missed geometry but it still works relatively well and it is easy to calculate.



(a) Lion (b) Self-shadow

(c) Stones (d) Shapes

(e) Collage (f) Rock Bump

Figure 4.1: Some of the different scenes that were tested.

Figure 4.1 contains the scenes that were used in the experiments. The scenes 'Lion', 'Self-shadow' and 'Collage' are from 'Steep Parallax Mapping' [23]. The 'Shapes' and 'Rock Bump' scenes are from 'Interval Mapping' [31]. The 'Lion' scene was used extensively for testing as it contains many different types of features, including smoother

stones, as well as sharp edges with large changes in height.

4.2 Heightmap only methods

The methods evaluated in this section only use a heightmap as input to find intersections. These methods do not require generating acceleration structures and therefore are suitable for dynamic heightmaps which change often, sometimes even every frame.

Section 4.2.1 discusses the results of the linear search algorithm, Section 4.2.2 discusses the results of the binary search algorithm, and results of the interval search algorithm are discussed in Section 4.2.3.

4.2.1 Linear search

Table 4.1 presents the mean-squared error and frame time of the linear search with increasingly more steps. The linear search can terminate early, which affects performance, thus the average steps is also shown.

Increasing the amount of linear steps improved the accuracy, but also drastically increased the time taken to render the frame. Linear search has 'step' artifacts where it is possible to see the boundaries between pixels where the step counts differed. The calculated intersections have a C_0 discontinuity between pixels of differing step counts. Linear search can also miss small features causing the silhouette of the features to be incorrect. As the total step count increased, these artifacts vanished, but, as shown in Figure 4.2, even at 50 steps it was still possible to see visual artifacts.

4.2.2 Binary search

Table 4.2 shows that binary search drastically improved on the error metric for each extra step. In Figure 4.3, the results of increasing the amount of steps is shown. Binary search decreased visual artifacts as the MSE indicates, but did not improve the silhouette of features. Binary search still had the same type of artifacts as produced by linear search, but the amount of binary search steps required to minimize the 'step' artifacts were considerably less.

Steps	Average steps per pixel	MSE	Frame time (ms)
5	2.843	0.01956579	0.4313
10	6.099	0.00515373	0.5052
15	9.210	0.00242219	0.5751
20	12.224	0.00136451	0.6414
25	15.260	0.00086101	0.7095
30	18.302	0.00060373	0.7772
40	24.386	0.00033423	0.9068
50	30.359	0.00021447	1.0339
60	36.334	0.00014471	1.1582
75	45.322	0.00008993	1.3447
100	60.290	0.00004856	1.6483
150	90.171	0.00001962	2.2478
200	120.111	0.00000997	2.8662

Table 4.1: Performance of linear search

Table 4.3 compares linear search to linear search with binary search. If two steps of binary search were used after linear search, the amount of linear steps could be lowered to reach the same approximate error. Much less linear steps could be used with only 2 binary steps, therefore improving the frame time.

4.2.3 Interval search

Table 4.4 shows how the error metric improved for each extra step of interval search. The first step of interval search already improved the error by a factor of 115 with 10 linear steps.

Table 4.5 compares binary search and interval search as a refinement search. Interval search initially outperformed binary search, but when more refinement steps were taken the two methods performed equally regarding error. However, interval search was slower.

(a) Linear only, 50 steps (b) Reference

Figure 4.2: Linear search artifacts. Left is linear search with 50 steps and right is the reference (200 linear steps with 8 interval steps).

In Figure 4.4, the comparison between the artifacts of binary search versus interval search can be seen. Binary search had 'step' artifacts that decreased in size with more steps. Interval search had much less artifacts in smooth regions, but had some distortion artifacts near to regions where the curvature of the heightmap was high.

In Graph 4.5, binary search and interval mapping are compared across different scenes. Binary search approximately halved the error for each additional step as expected. Interval mapping reduced the error more than binary search, but the error reduction depended heavily on the scene. Interval mapping did very well on smoother scenes (like 'Rock Bump') or scenes with a lot of flat surfaces (like 'Shadow'), and performed the best for all scenes.

Linear steps	Binary steps	MSE	Frame time (ms)
5	0	0.01956579	0.4271
5	1	0.00141109	0.4731
5	2	0.00038047	0.4946
5	3	0.00009538	0.5177
5	4	0.00002375	0.5369
5	5	0.00000604	0.5606
5	6	0.00000155	0.5760
10	0	0.00515373	0.5219
10	1	0.00037898	0.5489
10	2	0.00009498	0.5645
10	3	0.00002366	0.5890
10	4	0.00000601	0.6075
10	5	0.00000153	0.6312
10	6	0.00000042	0.6471

Table 4.2: Performance of binary search

Linear only			Similar error with 2 binary steps			
Linear steps	MSE	Frame time (ms)	Linear steps	MSE	Frame time (ms)	Frame time vs. linear only (%)
25	**0.00086101**	0.7095	3	0.00099514	**0.4591**	64.7
30	0.00060373	0.7772	4	**0.00059071**	**0.4776**	61.5
40	**0.00033423**	0.9068	5	0.00038047	**0.4910**	54.1
50	0.00021447	1.0339	7	**0.00018856**	**0.5209**	50.4
60	**0.00014471**	1.1582	8	0.00014571	**0.5359**	46.3
75	**0.00008993**	1.3447	10	0.00009498	**0.5629**	41.9
100	0.00004856	1.6483	14	**0.00004802**	**0.6179**	37.5

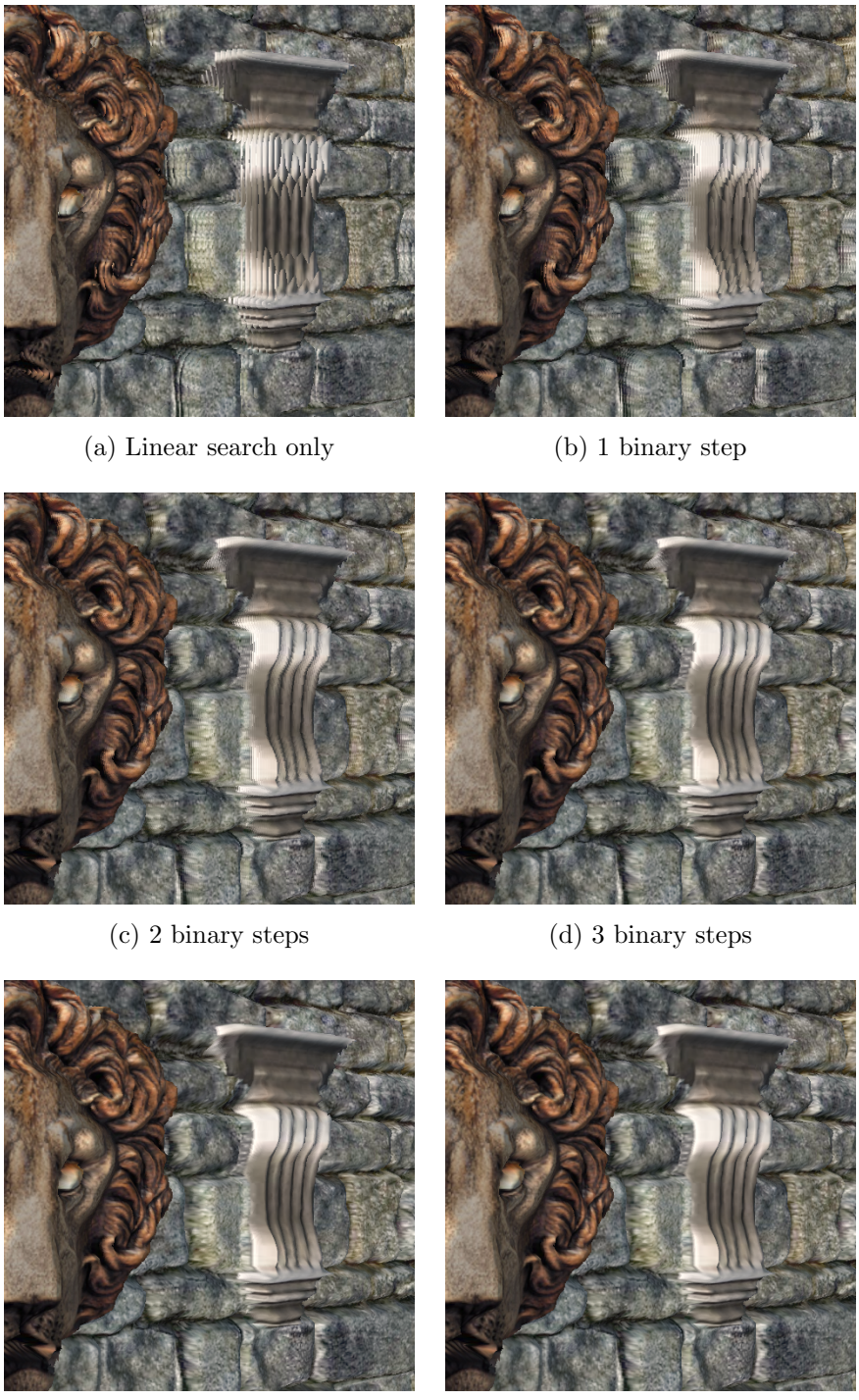
Table 4.3: Linear only vs linear + binary search

Linear steps	Interval steps	MSE	Frame time (ms)
5	0	0.01956579	0.4342
5	1	0.00032148	0.4922
5	2	0.00004102	0.5200
5	3	0.00001035	0.5446
5	4	0.00000384	0.5754
5	5	0.00000174	0.6019
5	6	0.00000096	0.6245
10	0	0.00515373	0.5051
10	1	0.00004616	0.5636
10	2	0.00000407	0.5903
10	3	0.00000090	0.6141
10	4	0.00000033	0.6446
10	5	0.00000015	0.6692
10	6	0.00000008	0.6942

Table 4.4: Performance of interval search

	Binary		Interval		
Refine steps	MSE	Frame time (ms)	MSE	Frame time (ms)	MSE improvement
1	0.00037898	**0.5477**	**0.00004616**	0.5601	8.2x
2	0.00009498	**0.5630**	**0.00000407**	0.5851	23.3x
3	0.00002366	**0.5867**	**0.00000090**	0.6113	26.3x
4	0.00000601	**0.6017**	**0.00000033**	0.6409	18.2x
5	0.00000153	**0.6287**	**0.00000015**	0.6659	10.2x
6	0.00000042	**0.6438**	**0.00000008**	0.6908	5.3x

Table 4.5: Binary vs interval search for refinement search with 10 linear steps



(a) Linear search only

(b) 1 binary step

(c) 2 binary steps

(d) 3 binary steps

(e) 4 binary steps

(f) 5 binary steps

Figure 4.3: Binary search artifacts. Binary search with 10 linear steps.

(a) 2 binary steps

(b) 3 binary steps

(c) 2 interval steps

(d) 3 interval steps

Figure 4.4: Binary vs interval search artifacts. Binary search vs interval search with 10 linear steps.

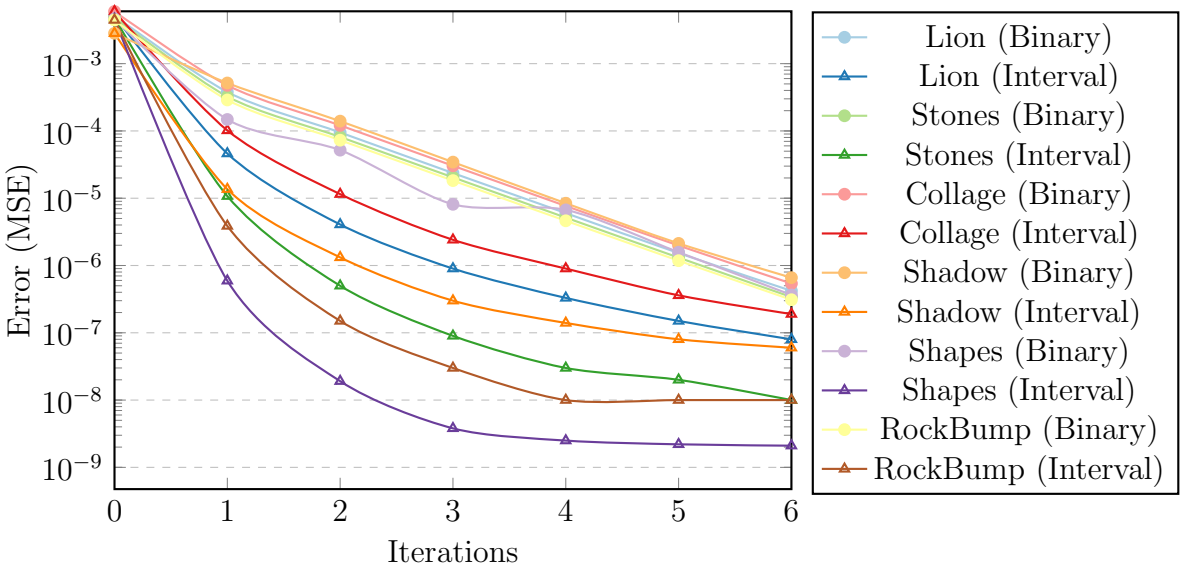


Figure 4.5: Comparison of binary search vs. interval mapping with 10 linear steps.

4.3 Accelerated methods

The methods evaluated in this section have to generate an acceleration structure to speed up the calculation of the intersection. Generating the acceleration structure usually takes very long, and thus these methods are usually not suitable for scenarios where the heightmap changes often.

Section 4.3.1 discusses the performance of the CSM method, Section 4.3.2 discusses QCSM and Section 4.3.3 discusses RCS. The new methods introduced in this dissertation, CNS and RCNS, are discussed in Section 4.3.4. Section 4.3.5 discusses the multiple cone version of the CNS method, referred to as multiple cone normal stepping (MCNS).

4.3.1 Cone Step Mapping

In the original paper on CSM [8], the cone texture (where the cone ratio is stored) is always sampled linearly, which is technically incorrect as the cone ratio is not linear. This incorrect sampling caused artifacts on sharp features, shown in Figure 4.6, which could be fixed by sampling the four texels around the point and taking the minimum cone ratio. Sampling correctly unfortunately deteriorated performance. Adding a refinement search after CSM also fixed the problem with less performance deterioration.

Table 4.6 shows that CSM quickly reached a point where increasing the amount of steps had a minimal impact on the error. The average steps per pixel indicated that most rays terminated very quickly. The addition of a refinement search, such as binary or interval search, helped to reduce the error of the linear blending of the cone map. Interval search did very well error-wise, but was a bit slower.

Graph 4.7 shows that CSM with 4 binary search steps performed very similar across all the scenes, except for the ‘Shapes’ scene where CSM performed really well. The ‘Shapes’ scene was very well suited to CSM due to the large amount of flat surfaces and gentle slopes.

Table 4.7 shows how CSM performed against linear search with regards to error. CSM improved on linear search with very few steps. It was possible to use much less steps with CSM to achieve the same level of distortion as linear search, thus improving on performance.

(a) Linearly sampled (b) Sampling the minimum

Figure 4.6: CSM produces artifacts when sampling the cone map linearly instead of taking the minimum. The red square highlights one of the locations with artifacts.

4.3.2 Quad Cone Step Mapping

Table 4.8 shows that QCSM had a very similar frame time to CSM, but had less error when the amount of steps were low. Even though QCSM had less average steps per pixel, it still performed the same, as the GPU needed to fetch four channel textures. The four channel texture, one for each of the main directions, increased storage by a factor of 4, thus the slight advantages might not be worth the extra storage, memory and bandwidth used.

4.3.3 Relaxed Cone Stepping

RCS [29] requires a refine search, unlike CSM and QCSM. Thus 5 binary search steps were also used with RCS.

Table 4.9 looks at the relative performance between the CSM, QCSM and RCS. RCS

Steps	Avg. steps per pixel	CSM only		2 binary steps	
		MSE	Frame time (ms)	MSE	Frame time (ms)
5	2.02	0.00139619	**0.5239**	**0.00113742**	0.5504
10	4.01	0.00014891	**0.5881**	**0.00010216**	0.6106
15	4.88	0.00005420	**0.6238**	**0.00002371**	0.6596
20	5.28	0.00003606	**0.6722**	**0.00000909**	0.6959
25	5.49	0.00003003	**0.7102**	**0.00000405**	0.7292
30	5.60	0.00002793	**0.7480**	**0.00000254**	0.7683
40	5.71	0.00002668	**0.8269**	**0.00000150**	0.8334
50	5.76	0.00000109	**0.8878**	**0.00000109**	0.9005

Table 4.6: Performance of CSM

Max steps	Linear search MSE	CSM MSE	Relative MSE (%)
5	0.01956579	**0.00129904**	6.63933171
10	0.00515373	**0.00017284**	3.35367006
15	0.00242219	**0.00008325**	3.43677003
20	0.00136451	**0.00006571**	4.81547955
25	0.00086101	**0.00005988**	6.95455337
30	0.00060373	**0.00005781**	9.57522402
40	0.00033423	**0.00005660**	16.93372827
50	0.00021447	**0.00005610**	26.15615238
60	0.00014471	**0.00005593**	38.64743280
75	0.00008993	**0.00005581**	62.06360503

Table 4.7: Error of linear search vs Cone Step Mapping

tended to have the same or better error with a smaller frame time than any of the other two methods when adding a refinement search. RCS performed especially well at lower step counts and did not require a four channel texture like QCSM.

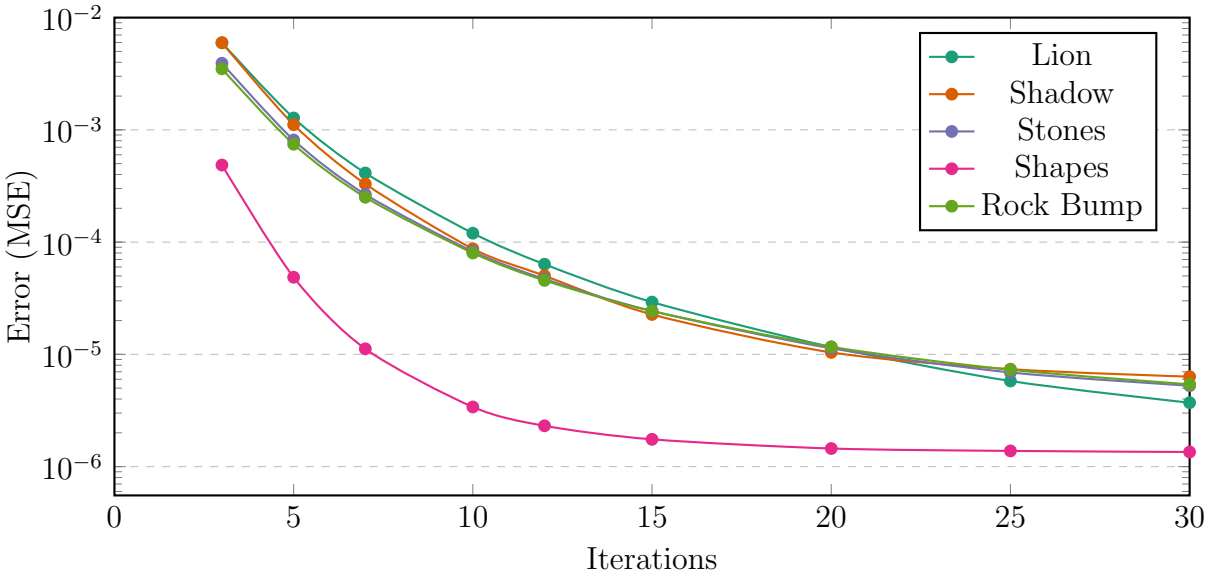


Figure 4.7: CSM with 4 binary search steps across different scenes.

Coarse steps	CSM			QCSM		
	Avg. Steps	MSE	Frame time (ms)	Avg. Steps	MSE	Frame time (ms)
5	2.016	0.00129904	**0.4659**	2.230	**0.00036170**	0.4887
10	4.009	0.00017284	**0.5348**	3.485	**0.00013606**	0.5596
15	4.877	**0.00008325**	**0.5810**	3.905	0.00012041	0.6123
20	5.282	**0.00006571**	**0.6217**	4.062	0.00011698	0.6631
25	5.489	**0.00005988**	**0.6637**	4.131	0.00011598	0.7068

Table 4.8: Cone Step Mapping vs Quad Cone Step Mapping

RCS requires a refinement search as the error was quite high without any refinement search, as can be seen in Table 4.10. Interval search as a refinement search did well when there were only about 2 steps. At higher step counts, binary search at 6 steps performed similar to interval search at 4 steps. Higher step counts of interval search did not improve the error much in this test.

	CSM		QCSM		RCS	
Coarse steps	MSE	Frame time (ms)	MSE	Frame time (ms)	MSE	Frame time (ms)
5	0.00125975	0.6113	0.00025199	0.6190	**0.00016891**	**0.5851**
10	0.00011827	0.6587	**0.00002229**	0.6768	0.00002260	**0.6214**
15	0.00002809	0.6949	**0.00000654**	0.7206	0.00001142	**0.6614**
20	0.00001059	0.7456	**0.00000310**	0.7592	0.00000934	**0.6901**
25	0.00000476	0.7682	**0.00000209**	0.7999	0.00000863	**0.7209**

Table 4.9: Cone Step Mapping vs Quad Cone Step Mapping vs Relaxed Cone Stepping, all with 5 binary search steps

		Binary search		Interval search	
Coarse steps	Refine steps	MSE	Frame time (ms)	MSE	Frame time (ms)
10	0	0.00234359	0.5002		
10	2	0.00053803	**0.5628**	**0.00003801**	0.5860
10	4	0.00004697	**0.6024**	**0.00001146**	0.6414
10	6	0.00001647	**0.6421**	**0.00000974**	0.6906
20	0	0.00233177	0.5809		
20	2	0.00052409	**0.6332**	**0.00002964**	0.6532
20	4	0.00003358	**0.6679**	**0.00000312**	0.7016
20	6	0.00000327	**0.7003**	**0.00000138**	0.7463

Table 4.10: Relaxed Cone Stepping, refined search comparison

4.3.4 Cone Normal Stepping

CNS was defined in Chapter 3, and can also be combined with RCS to produce RCNS.

CNS and CSM are compared in Table 4.11 where CNS produced very similar results to CSM, but CNS was a bit slower. The accuracy was lower for CNS, as it allowed cones up to 180°wide, whereas CSM only allowed for cones up to 90°wide. This negatively

Steps	CSM			CNS		
	Avg. steps per pixel	MSE	Frame time (ms)	Avg. steps per pixel	MSE	Frame time (ms)
5	**2.016**	**0.00125975**	0.5750	**2.016**	0.00126388	0.6023
8	**3.423**	**0.00025695**	0.6592	3.425	0.00025793	0.6535
10	**4.009**	**0.00011827**	0.6699	4.012	0.00011879	0.6783
15	**4.878**	**0.00002809**	0.6895	4.882	0.00002827	0.7219
20	**5.282**	**0.00001059**	0.7195	5.287	0.00001067	0.7618
30	**5.603**	**0.00000268**	0.7940	5.609	0.00000270	0.8416

Table 4.11: Performance of Cone Normal Stepping with 5 binary search steps

affected CNS, as the cones were rarely wider than 90°.

RCNS, the relaxed cone version of CNS, fared much better against RCS, than what CNS fared against CSM. Table 4.12 shows that both RCNS and RCS had similar performance as the extra wide angles that RCNS supports can be utilized, which could be seen in the slightly reduced average step count and lower error. The more complex intersection test caused RCNS to be slightly slower, despite the lower average step count.

Steps	RCS			RCNS		
	Avg. steps per pixel	MSE	Frame time (ms)	Avg. steps per pixel	MSE	Frame time (ms)
5	2.115	**0.00016891**	0.6051	2.115	0.00016902	**0.5862**
8	2.915	0.00004071	0.6378	2.916	**0.00004059**	**0.6206**
10	3.180	0.00002260	**0.6358**	3.181	**0.00002247**	0.6546
15	3.444	**0.00001142**	**0.6512**	3.445	0.00001144	0.6569
20	3.533	**0.00000934**	**0.6735**	3.534	**0.00000934**	0.7034
30	3.595	**0.00000852**	0.7723	3.596	**0.00000852**	**0.7671**

Table 4.12: Performance of Relaxed Cone Normal Stepping with 5 binary search steps

CNS and RCNS did not really have any advantage performance wise over CSM and

RCS respectively, but allowed for a more accurate approximation of the distance field, which can be used for other effects.

4.3.5 Multiple Cone Normal Stepping

	CNS			MCNS		
Steps	Avg. steps per pixel	MSE	Frame time (ms)	Avg. steps per pixel	MSE	Frame time (ms)
5	**2.016**	0.00126388	**0.6088**	2.043	**0.00114881**	0.6696
8	3.425	0.00025793	**0.6599**	**3.395**	**0.00024533**	0.6957
10	4.012	0.00011879	**0.6847**	**3.965**	**0.00011471**	0.7310
15	4.882	0.00002827	**0.7302**	**4.817**	**0.00002777**	0.7855
20	5.287	0.00001067	**0.7691**	**5.214**	**0.00001051**	0.8483
30	5.609	0.00000270	**0.8494**	**5.530**	**0.00000267**	0.9458

Table 4.13: Performance of Multiple Cone Normal Stepping with 5 binary search steps

Using multiple cones with Method 1 described in Section 3.4 is not worthwhile, as shown in Table 4.13. Even though the average step per pixel was slightly lower, as the ray can skip over the top parts of the heightmap faster, most time was spent closer to the heightmap surface causing the gains to be minimal. The extra intersection test(s) required outweighed any benefit from performing less steps. The error started out only slightly better for MCNS, but as the amount of steps increased the difference in error vanished. CNS was faster than MCNS with very similar error.

Graph 4.8 shows how each method performed without any refinement search. A few step counts for each method are plotted such that the x-axis is time and the y-axis, with a log-scale, is MSE. Each method was tested with 5 to 35 steps. A lower MSE for a given frame time is considered better. Both this graph and the following graph used the 'Lion' scene.

CSM and CNS performed really well with CNS being slightly slower for the given amount of error. QCSM performed better for very low step counts, but was quickly

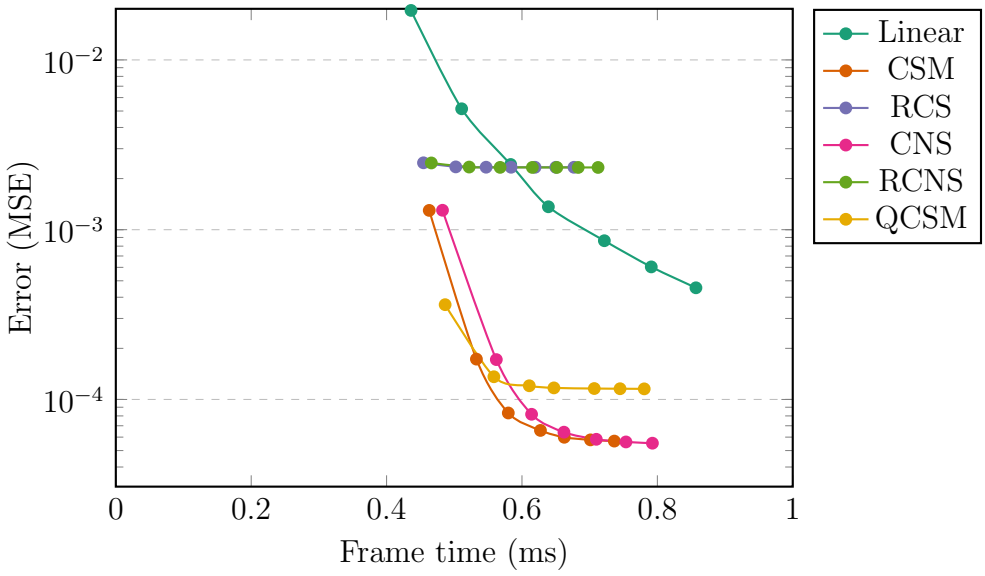


Figure 4.8: Comparison of all coarse search methods without refinement.

overtaken by CSM. RCS and RCNS did not perform very well and did not see any real improvement with more steps due to the fact that they were meant to be used with a refinement search. Linear performed really poor at low step counts, but caught up and surpassed RCS at higher step counts. With no refinement search, CSM was the overall winner at nearly every step count.

Graph 4.9 is the same comparison as Graph 4.8, but adds 3 binary search steps as the refinement search. This graph shows that CSM, CNS and QCSM performed very similar, but with much improved error. RCS and RCNS improved their error by an order of magnitude. RCS and RCNS experienced a small drop in error at 10 steps, but did not improve with more steps. Linear improved drastically with its error dropping by two orders of magnitude. With 3 binary search steps of refinement, Linear search performed the best.

Graph 4.10 shows how each method performed using 3 steps of interval mapping instead. All methods had less error for any given step count. QCSM performed better than both RCS and RCNS, while RCS and RCNS performed better than QCSM at lower step counts. Linear search performed even better compared to all the other methods with interval mapping as the refinement search and was the overall winner.

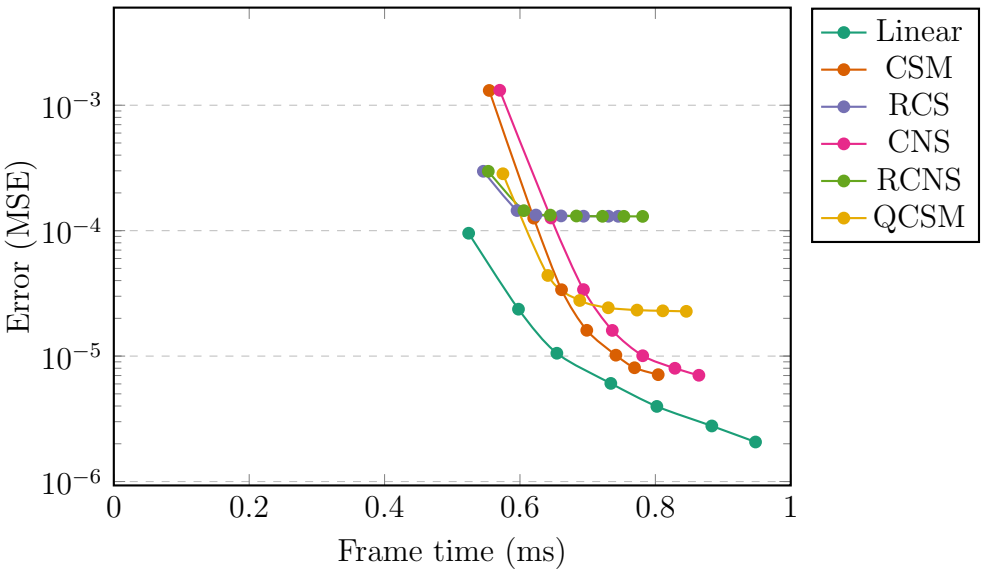


Figure 4.9: Comparison of all coarse search methods with 3 binary search steps.

Graph 4.11 compared CSM with linear search, with either no refinement search, binary search, or interval mapping. CSM with no refinement search performed well for low frame times, but linear search with interval mapping performed the best.

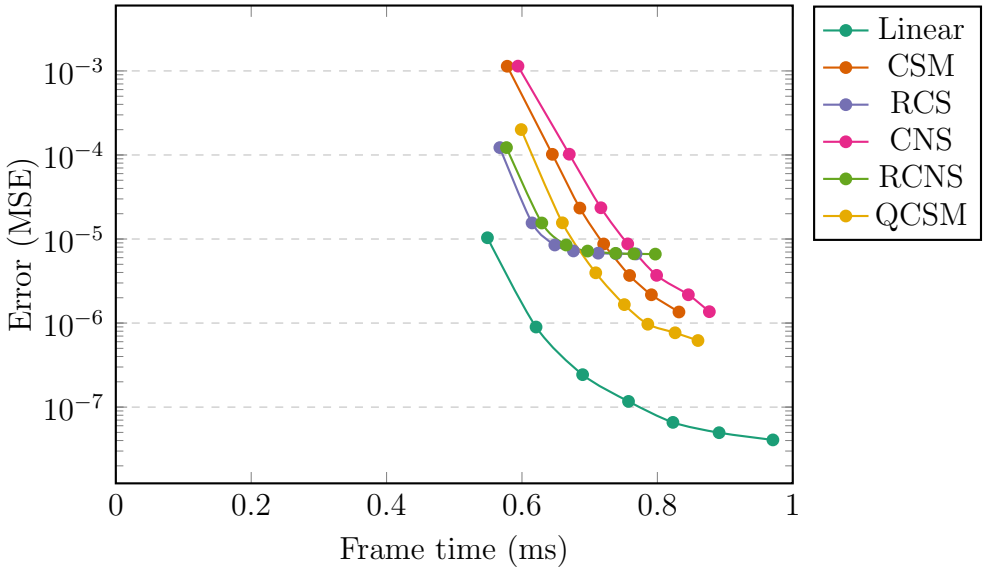


Figure 4.10: Comparison of all coarse search methods with 3 interval search steps.

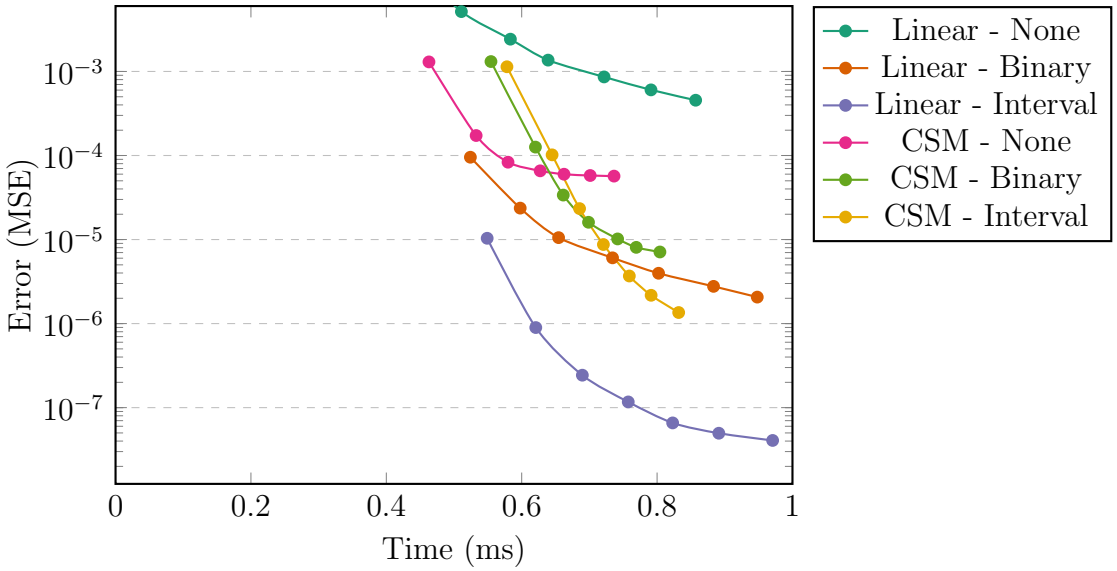


Figure 4.11: Comparison of linear vs. CSM with either no refinement search, binary search or interval search.

4.4 Shadows

Three shadow methods were compared, namely hard shadows, height map soft shadows and cone normal soft shadows. These methods all use linear search as their base. The hard shadow method blocks all light if the ray hits the heightmap. Linear soft shadow attenuates the light relative to how closely the ray travels to the height-map and is described in Section 2.6 [37]. The cone normal soft shadow combines the distance approximation from the normal cone map introduced in Section 3.2 with a distance field soft shadowing method [30].

Steps	Hard shadow frame time (ms)	Heightmap soft shadow Frame time (ms)	Cone normal soft shadow Frame time (ms)
0	0.7281		
10	**0.9710**	0.9736	1.0280
20	**1.1722**	1.1773	1.2876
30	**1.3725**	1.3812	1.5448
40	1.5789	**1.5769**	1.7934
50	**1.7771**	1.7775	2.0428
60	1.9774	**1.9771**	2.2938

Table 4.14: Performance of shadow linear search

Table 4.14 compares the various methods of shadowing using linear search. Shadowing took a significant amount of the frame time as linear search was not very efficient at finding intersections, as shown in Section 4.2.1. The heightmap soft shadow method increased the frame time, and the cone normal soft shadow method increased it even further as it required accessing the cone map texture.

In Figure 4.12 the results of the different methods are presented. The ambient light was turned off to accentuate the shadowing. The soft shadowing methods helped to hide artifacts caused by lower steps and were less harsh. The heightmap soft shadowing only took into account the geometry below the ray, while the cone normal soft shadowing used the cone map to take into account geometry in the area around the ray. This could especially be seen near the top of the shadow cast by the pillars where the cone normal

(a) Hard shadow with 30 steps

(b) Hard shadow with 70 steps

(c) Heightmap soft shadow with 30 steps

(d) Heightmap soft shadow with 70 steps

(e) Cone normal soft shadow with 30 steps

(f) Cone normal soft shadow with 70 steps

Figure 4.12: Soft shadowing methods comparison.

soft shadowing approximated the soft shadow in all directions, while the heightmap soft shadowing only approximated the soft shadow in the direction of the light.

4.5 Ambient occlusion

AO approximates global illumination by attenuating the ambient term relative to the amount of occlusion around the point to be shaded. AO can be approximated using distance fields, as well as using the approximate distance field described in Section 3.2. Calculating the AO from a cone map has several advantages. The AO can be baked into the albedo texture or supplied separately, but the first approach is incorrect for direct lighting, and the second approach requires extra texture memory. Generating the AO term from the cone map allows for a separate AO term without extra textures.

Method	No ambient occlusion frame time (ms)	With ambient occlusion Frame time (ms)
CSM	**0.8592**	0.8604
CNS	**0.8998**	0.9022

Table 4.15: Performance of ambient occlusion approximations

Table 4.15 indicates that there was negligible impact to the frame time when adding the approximate AO term. Unfortunately the approximation could only be used with CSM and CNS, as the cones of RCS and RCNS did not allow generating good approximations to the distance field. The results can be seen in Figure 4.13. Visually, the AO generated by the distance field seemed to do a good job of approximating local AO, darkening corners and crevices, as well as darkening in close proximity of larger features, such as the pillars. The cone ratio from CSM worked very well as an approximate AO term, and looked nearly identical to the cone normal of CNS, except for some extra darkening.

The next chapter provides a summary of the whole dissertation and analyzes the results obtained in this chapter.

(a) No ambient occlusion

(b) Using cone ratio, AO only

(c) Using cone ratio, textured

(d) Using cone normal, AO only

(e) Using cone normal, textured

Figure 4.13: Ambient occlusion approximation comparison.

Chapter 5

Conclusions

This chapter provides a summary, states the conclusions of the experiments and highlights further research that could be done.

Section 5.1 provides a summary of the whole dissertation and provides conclusions based on the results of the previous chapter. Section 5.2 covers work that could still be done in future to further the field.

5.1 Summary of Conclusions

This section summarizes the whole dissertation and is divided into Section 5.1.1 which summarizes the related work, Section 5.1.2 where the newly introduced methods are summarized, and Section 5.1.3 where all the results are combined and conclusions drawn.

5.1.1 Related work

Chapter 1 covered the motivation and objective for this dissertation. There is much room for improvement in the field of parallax and relief mapping and opportunities for adding to the techniques that were mentioned. The objective of this dissertation was to not just compare the existing methods introduced by previous research, but to seek out improvement on the *status quo*. Thus, this dissertation introduced new methods.

Methods introduced in previous related work were covered in Chapter 2, starting with some of the first methods of increasing the apparent complexity of polygons without real

geometry. Texture mapping was the first technique applied to polygons, allowing for the variation of color across the surface of the polygon. This was followed by normal mapping, which allowed for changing the normal across the polygon. Normal mapping allowed more sophisticated lighting giving the impression of apparent geometry where there was none. This technique was, however, only limited to lighting and thus only smaller features could realistically be represented by normal mapping.

Early computers were limited by the amount of real polygon geometry they could render in real-time. The first methods trying to simulate the appearance of more complex geometry were developed. Initial methods warped the texture applied to polygons depending on the location of the viewer to simulated parallax. This was implemented on the central processing unit (CPU), as the graphics processing unit (GPU) hardware of the time only provided fixed functionality, and was very slow.

Later GPU hardware introduced some programmability, allowing for some parts of the warping to be done on the GPU. Instead of warping textures, the texture could rather be sampled in a way that produced the same effect. This was the basis of the Parallax Mapping method. Although this method was inaccurate and produced a lot of distortion in certain cases, it could be completely implemented on the GPU and was very fast to execute, because of its simplicity. Later improvements were made to the method to reduce the distortion and increased the performance, but faded out the effect at grazing angles.

Pixel and vertex shaders introduced by newer GPU hardware allowed for more complex algorithms to be implemented on the GPU. Steep Parallax Mapping used ray casting to determine the intersection point on the heightmap, allowing for a more accurate effect. This method uses an initial linear search to get a rough estimate of the intersection point, followed by a binary search to refine the intersection point. Interval mapping was then proposed as a replacement for binary search.

Cone step mapping (CSM) was proposed which uses a cone map as an acceleration structure for calculating the intersection point. Quad cone step mapping (QCSM) and anistropic cone mapping (ACM) are enhancements to CSM that define a cone per direction to better fit the heightmap. Relaxed cone stepping (RCS) allows the cones defined in CSM to penetrate the heightmap, allowing for faster traversal of the cone

map. Although, this requires a refinement step after the initial traversal.

Shell mapping and Curved shell mapping allow curved relief mapped surfaces without distortion.

Screen space reflection methods are very similar to relief mapping, and similar techniques are used in both methods. One noteworthy new method uses the 3D Digital Differential Analyzer (DDA) algorithm instead of linear search and a refinement search to accurately find the intersection point.

Shadowing can be done for relief mapped surfaces as well. One of the original techniques is horizon mapping, that was later extended to take surface curvature into account. Some simple techniques, such as ellipse shadow map, were introduced for shadowing. This technique was later extended to support soft shadowing and indirect lighting.

When using ray tracing for relief mapping, the same method can be used to calculate shadowing. The shadow rays can also benefit from acceleration structures, but the refinement search is not necessary.

Instead of using a heightmap to represent the apparent geometry of the polygon, distance fields can be used. This has several advantages, including being able to represent more complex geometry and allowing easy approximation of certain effects, such as soft shadows and ambient occlusion.

Improvement to relief mapping techniques have slowed down, but more games and game engines are starting to use these techniques.

5.1.2 Contributions

This dissertation introduced a few new techniques. These techniques bridge the gap between CSM and distance fields. Cone normal stepping (CNS) replaces the representation of the cone maps in CSM from a gradient, to the normal of the cone sides. This removes the limit on the cone angle that is represented by the cone map, and also allows the calculation of a conservative estimate of the distance field. The latter approach allows calculating the same approximations for soft shadows and ambient occlusion, as for distance fields. This change is also applied to RCS to produce relaxed cone normal stepping (RCNS).

Multiple cones are also explored, but unlike QCSM and ACM, the cones are stacked

instead of defined for separate directions in a effort to reduce the amount of steps to reach an intersection. Multiple cones also allow fitting the distance field more accurately, producing a more accurate approximation of the distance field.

5.1.3 Results

Linear search is a very simple method of calculating the intersection point of a ray onto a heightmap. Linear search steps through the heightmap along the ray at regular intervals, checking the heightmap at each step to determine if the ray dipped below the heightmap. Linear search on its own was not very efficient, since even very high step counts of linear search produced clearly visible artifacts, but still required a lot of time to calculate.

Adding a refinement search after linear search was much more efficient. One such refinement method is binary search. This method explored in Section 2.3. Even using 2 steps of binary search drastically reduced the amount of linear steps required to keep the same error by nearly one order of magnitude. This lead to much better performance.

Interval search is another refinement search method explored in Section 2.3. Interval search performed better than binary search with the equivalent amount of steps at lower step counts. Interval search performed well with smooth surfaces, but binary search had less artifacts in areas where the gradient changed quickly. Most textures, however, were mostly smooth and the artifacts caused by interval search was not as bad as those of binary search.

All these methods only require a heightmap to execute and thus work well on dynamic heightmaps or heightmaps that change often. It is always recommended that linear search be used in combination with a refinement search. Interval search worked very well with smooth surfaces, but it is recommend to choose between binary search and interval search depending on the heightmap properties.

Accelerated methods, such as CSM, require the generation of a separate acceleration structure other than the heightmap. This means they are not well suited to dynamic heightmaps but allow for much faster intersection calculations.

CSM required much less steps than linear search to achieve the same level of error, and thus, was much faster than linear search. QCSM and ACM required less steps than CSM to reach the same error level, but performed similarly and used more storage than

CSM.

Cone maps are non-linear and must actually be sampled conservatively by taking the minimum texel instead of interpolating the cone map texture. If this is not done then sharp features will have artifacts. Sampling conservatively was slow and thus using a refinement search step had better performance.

RCS was faster than both CSM and QCSM, but did not have the extra storage requirements of QCSM. RCS requires a refinement search, but due to the non-linearity problem of CSM it is recommended for CSM as well.

The new methods introduced in this dissertation are based on CSM. CNS is a modified version of CSM. CNS performed slightly slower than CSM as the intersection test was slightly more expensive, and accuracy of the cone angle was decreased. CNS, however, allows the calculation of a conservative approximation of the distance field.

RCNS is a modification of the RCS method. RCNS and RCS performed nearly identically. RCNS performed especially well at lower step counts where it had a lower error than RCS. RCS and RCNS, unfortunately, could not be used to approximate the distance field, as the cones did not represent the local geometry in a way that was necessary for the calculation.

The best cone mapping method was either RCS or RCNS, the methods introduced in this dissertation. Both outperformed the other cone mapping methods and RCNS had a slight advantage at lower step counts.

Multiple cone normal stepping (MCNS) extends CNS so that there are multiple cones per texel. Unlike QCSM and ACM, cones are stacked instead of defined for different directions. Stacking the cones allows for larger steps to be taken, and allows for a better approximation of the distance field. Unfortunately, MCNS was slower than using a single cone due the larger steps not making up for the extra cone intersection test.

When comparing all the methods to each other, using a linear search with interval mapping gave the best error for a given frame time, but as noted in Section 4.1, the error metric is not perfect as it may not take into account missed geometry.

Adding shadows was slower than without them. Soft shadows only have a small performance cost over and above hard shadows. CNS allowed for more accurate calculation of soft shadows as it took into account the local geometry and not only the geometry

under the shadow ray. The performance impact of using CNS for soft shadows was significant, but produced nicer soft shadows.

Approximate soft shadows can easily be added to CSM with minimal performance impact. Using the soft shadowing of CNS produced more accurate soft shadows, but had a larger performance impact.

Ambient occlusion (AO) can be approximated using CNS and has a minimal impact on performance. Even though the cone ratio of CSM cannot be used to derive a distance field for calculation of AO, the similarity to the cone normal means that it performs similar.

5.2 Future Work

Not all the techniques mentioned in Chapter 2 were investigated, such as maximum mipmaps and tracing distance fields directly. These methods should be investigated to determine their performance characteristics and viability. A different error metric that takes missed geometry or perceived error into account might give a better indication of which methods work the best.

CNS relies on a new intersection method that is slower than CSM. It might be possible to optimize the intersection method further to make CNS a viable alternative to CSM.

Method 2 of the MCNS defined in Section 3.4 was also not evaulated. As the first method did not perform that well, it was assumed that the second method will probably also not perform very well. Method 2 does allow for selecting the correct cone to intersect instead of trying all the cones, thus it may perform better than Method 1.

Screen space ray tracing might be able to use some of the accelerated techniques, such as CSM to accelerate ray traversal, if the generation of the cone map can be either made much faster or approximated.

Relief mapping is a bridge between rasterization techniques and ray tracing. It might be possible to render larger sections of scenes on GPUs using ray tracing where rasterization struggles. One idea might be to render bodies of water using ray tracing only, ray tracing all the geometry below the surface of the water, or rendering complex geometry like grass with ray tracing.

Bibliography

[1] Saif Ali, Jieping Ye, Anshuman Razdan, and Peter Wonka. Compressed facade displacement maps. In *IEEE Transactions on Visualization and Computer Graphics*, volume 15, pages 262–273, mar 2009.

[2] James F. Blinn. Simulation of wrinkled surfaces. *ACM SIGGRAPH Computer Graphics*, 12(3):286–292, aug 1978.

[3] Zoe Brawley and Natalya Tatarchuk. Parallax Occlusion Mapping: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing. In W Engel, editor, *ShaderX3: Advanced Rendering with DirectX and OpenGL*, pages 135–154. Charles River Media, 2005.

[4] Ying Chieh Chen and Chun Fa Chang. A prism-free method for silhouette rendering in inverse displacement mapping. *Computer Graphics Forum*, 27(7):1929–1936, oct 2008.

[5] Yu-Jen Chen and Yung-Yu Chuang. Anisotropic Cone Mapping. *Proceedings of 2009 APSIPA*, 1:9–12, 2009.

[6] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 137–145, New York, New York, USA, jul 1984. ACM Press.

[7] William Donnelly. Per-Pixel Displacement Mapping with Distance Functions. In *GPU Gems 2*, chapter 8, pages 123–137. Pearson Education, Inc., 2005.

[8] J Dummer. Cone step mapping: An iterative ray-heightfield intersection algorithm. *Dostupno na http://www. lonesock. net/files/*, 2006.

[9] Alex Evans. Fast Approximations for lighting of Dynamic Scenes. *Advanced Real-Time Rendering in 3D Graphics and Games SIGGRAPH 2006*, pages 153 – 171, 2006.

[10] Wolfgang Heidrich, Katja Daubert, Jan Kautz, and Hans-Peter Seidel. Illuminating micro geometry based on precomputed visibility. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, pages 455–464, New York, New York, USA, 2000. ACM Press.

[11] Inigo Quilez. Sphere - soft shadow, 2014.

[12] Stefan Jeschke, Stephan Mantler, Michael Wimmer, J. Kautz, and S. Pattanaik. Interactive smooth and curved shell mapping. *Rendering Techniques 2007*, 6:351–360, 2007.

[13] Takashi Kanai and Masahiro Fujita. Hardware-assisted relief texture mapping. *Eurographics 2002 Short paper presentations*, 2:257–262, 2002.

[14] Tomomichi Kaneko, Toshiyuki Takahei, Masahiko Inami, Naoki Kawakami, Yasuyuki Yanagida, Taro Maeda, and Susumu Tachi. Detailed Shape Representation with Parallax Mapping. In *In Proceedings of the ICAT 2001*, pages 205–208, 2001.

[15] Jan Kautz, Wolfgang Heidrich, and Katja Daubert Mpi. Bump Map Shadows for OpenGL Rendering. 2000.

[16] Alexander Kharlamov, Iain Cantlay, and Yuri Stepanenko. Next-generation speedtree rendering. In *GPU Gems*, volume 3, chapter 4, pages 69–92. Addison-Wesley Professional, 2007.

[17] Lo Wei Lee, Shih Wei Tseng, and Wen Kai Tai. Improved relief texture mapping using minmax texture. In *Proceedings of the 5th International Conference on Image and Graphics, ICIG 2009*, pages 547–552. IEEE, sep 2010.

[18] Frederico A. Limberger, Victor C. Schetinger, and Manuel M. Oliveira. Meta-Relief Texture Mapping with Dynamic Texture-Space Ambient Occlusion. In *Brazilian Symposium of Computer Graphic and Image Processing*, volume 2015-Octob, pages 1–8, 2015.

[19] Jianxin Luo, Guyu Hu, and Guiqiang Ni. Dual-space ray casting for height field rendering. *Computer Animation and Virtual Worlds*, 25(1):45–56, jan 2014.

[20] Michael Mara and Morgan Mcguire. Fast Global Illumination Approximations on Deep G-Buffers. *NVIDIA Technical Report NVR-2014-001*, 2014.

[21] Nelson L. Max. Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer*, 4(2):109–117, mar 1988.

[22] Morgan Mcguire and Michael Mara. Efficient GPU Screen-Space Ray Tracing. *Journal of Computer Graphics Techniques*, 3(4):73–85, 2014.

[23] Morgan Mcguire and Max Mcguire. Steep Parallax Mapping. *I3D 2005 Poster*, pages 2005–2005, apr 2005.

[24] K Ohrn. Different Mapping Techniques for Realistic Surfaces. 2008.

[25] Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief texture mapping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, pages 359–368, New York, New York, USA, 2000. ACM Press.

[26] Koichi Onoue, Nelson Max, and Tomoyuki Nishita. Real-time rendering of bumpmap shadows taking account of surface curvature. In *Proceedings - 2004 International Conference on Cyberworlds, CW 2004*, pages 312–318. IEEE, 2004.

[27] David W. Paglieroni and Sidney M. Petersen. Height distributional distance transform methods for height field ray tracing. In *ACM Transactions on Graphics*, volume 13, pages 376–399. ACM, oct 1994.

[28] Fábio Policarpo, Manuel M. Oliveira, and João L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *ACM SIGGRAPH 2005 Papers on - SIGGRAPH '05*, volume 24, page 935, New York, New York, USA, 2005. ACM Press.

[29] Fábio Fabio Policarpo and Manuel M. Oliveira. Relaxed cone stepping for relief mapping. *GPU Gems 3*, 3(Dummer):409–428, 2007.

[30] Inigo Quilez. Free penumbra shadows for raymarching distance fields, 2010.

[31] Eric Risser, Musawir Shah, and Sumanta Pattanaik. Interval Mapping. *University of Central Florida Technical Report*, 2005.

[32] Ryan Ingram. Building and Rendering SimCity (2013), 2013.

[33] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, pages 327–334, 2000.

[34] Peter-Pike J. Sloan and Michael F. Cohen. Interactive Horizon Mapping. In Bernard Péroche and Holly Rushmeier, editors, *Rendering Techniques '00 (Proc. Eurographics Workshop on Rendering)*, pages 281–286. Springer Vienna, Vienna, 2000.

[35] Tiago Sousa, Wenzel Carsten, and Chris Raine. The Rendering Technologies of Crytek 3. In *Gdc*, 2013.

[36] Laszlo Szirmay-Kalos and Tamas Umenhoffer. Displacement mapping on the GPU state of the art. *Computer Graphics Forum*, 27(6):1567–1592, sep 2008.

[37] Natalya Tatarchuk. Dynamic parallax occlusion mapping with approximate soft shadows. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games - SI3D '06*, volume 1, page 63, New York, New York, USA, 2006. ACM Press.

[38] Art Tevs, Ivo Ihrke, and Hans-Peter Seidel. Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. *Symposium on Interactive 3D Graphics and Games*, page 183, 2008.

[39] Michal Valient. Killzone: Shadow Fall Demo Postmortem. *Sony Devstation*, 2013.

[40] Terry Welsh. Parallax mapping with offset limiting: A per-pixel approximation of uneven surfaces. *Infiscape Corporation*, pages 1–9, 2004.

[41] B Wronski. The future of screenspace reflections, 2014.

[42] Bartłomiej Wroński. Assassin's Creed 4: Black Flag: Road to next-gen graphics. *Game Developers Conference*, 2014.

Appendix A

Acronyms

ACM anistropic cone mapping. 15, 53, 54, 56

AO ambient occlusion. 2, 45, 46, 51, 57

CNS cone normal stepping. 2, 35, 39–41, 45, 46, 54, 56, 57

CPU central processing unit. 5–7, 17, 28, 53

CSM cone step mapping. iv, 2, 3, 13–15, 17, 22, 26, 34–42, 45, 46, 53–57

DDA Digital Differential Analyzer. 20, 54

GCC GNU compiler collection. 28

GPU graphics processing unit. 6, 7, 17, 18, 28, 37, 53

HDDT height distributional distance transform. 15, 17

LOD level of detail. 13

MCNS multiple cone normal stepping. 35, 41, 56, 57

MSE mean squared error. 28, 31, 41

POM parallax occlusion mapping. 19

QCSM quad cone step mapping. 14, 15, 34, 37, 38, 41, 42, 53, 54, 56

RCNS relaxed cone normal stepping. 2, 35, 39–42, 46, 54, 56

RCS relaxed cone stepping. 2, 14, 34, 38–42, 46, 53, 54, 56

Index