Acta Cybernetica **26** (2024) 529–542.

Quadratic Displacement Maps for Heightmap Rendering*

Mátyás Kiglics[ab], Gábor Valasek[ac], Csaba Bálint[ad], and Róbert Bán[ae]

Abstract

We present a higher-order representation of heightfields by constructing unbounding revolved parabolas about every texel of the height texture. These surfaces of revolution do not intersect the interior of the volume defined by the heightfield. We present a simple generation algorithm and show that these maps can be rendered by computing intersections between lines and parabolas in the plane. We compare its quality and performance with cone step mapping.

Keywords: computer graphics, parallax mapping, cone step mapping, quadric tracing

1 Introduction and Related Work

Heightmaps are two-dimensional textures that store elevation values at each sample position. These textures are mapped onto simplified base geometries, and the base shapes are transformed by displacing their points by the corresponding elevations along a direction. This direction is usually the unit normal of an interpolated tangent frame over the surface. Geometrically, the heightfield describes a variable radius offset of the coarse geometry, as shown in Figure 1.

There are two main approaches to the implementation of the above transformation [8]. A geometric one takes the vertices of the simplified base shape and translates them according to the heightmap. This mesh-based heightmap requires a sufficiently dense base geometry to accommodate the heightmap resolution. It also necessitates carefully crafted level-of-detail (LOD) variations of the base shape

*Supported by the ÚNKP-22 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.

[a]Eötvös Loránd University, Budapest, Hungary
[b]E-mail: kiglics@caesar.elte.hu, ORCID: 0000-0003-4957-7495
[c]E-mail: valasek@inf.elte.hu, ORCID: 0000-0002-0007-8647
[d]E-mail: csabix@inf.elte.hu, ORCID: 0000-0002-5609-6449
[e]E-mail: rob.ban@inf.elte.hu, ORCID: 0000-0002-8266-7444

DOI: 10.14232/actacyb.299736

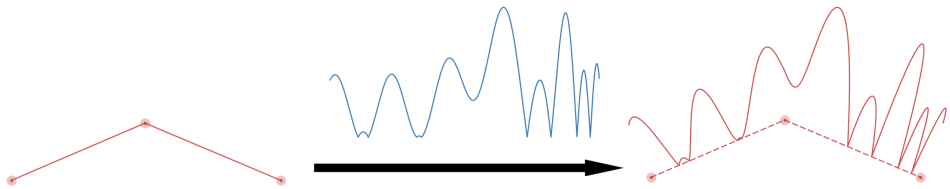


Figure 1: Heightfield (center, in blue) applied on top of a coarse geometry (left) yields a continuous, higher detail surface (right).

so that GPU performance is not wasted by rendering micro-triangles at a distance. Moreover, the transitions between the LODs should also be seamless.

The screen-space or per-fragment approach does not alter the raw geometry; instead, it casts a ray through each pixel of the base shape and alters the shading parameters according to a ray trace against the heightfield. Here, the displaced geometry is not stored explicitly; it only exists procedurally during ray traversal.

Initial screen-space techniques relied on the linear search along the ray to identify the ray-heightfield intersection [8]. Dummer proposed a conservative empty space skipping technique called cone step mapping to accelerate this process [4] with a different heightmap representation. Unbounding cones replaced the elevation values. These are the widest cones that are disjoint from the heightfield, have their apex on the heightfield surface, and their axes of symmetry are the tangent-space normals. Usually, these cones are stored with two scalars: the height of the apex and the tangent of the half-cone angle. Other numerical representations have been proposed as well that improve various numerical properties [5].

The current state-of-the-art in rendering performance is the relaxed cone map technique of Policarpo and Oliveira [7]. They extended Dummer's cone step mapping by replacing strictly conservative cones with relaxed cones that do not allow for more than one outside-inside transition between the ray and the heightmap. This approach guides the tracing inside the volume of the heightfield, so it requires a robust root refinement process, e.g., binary search, to find the surface point of the intersection.

Our paper proposes a generalization of cone maps by assigning a surface of revolution based on a parabola to each heightmap sample. This approach is a generalization of quadric maps [2]. The resulting surfaces are conservatively unbounding in the same sense as Dummer's, and we refer to them as unbounding revolved parabolas. Figure 2 illustrates several parabola cross-sections.

Section 2 introduces our proposed representation and specifies our cone-parabola hybrid model mathematically. We present a construction algorithm in Section 3 and a new tracing method for our data structure in Section 3.2. We propose generation and render time optimizations in Section 4. Section 5 contains our empirical results. We compared our proposed method with cone step mapping. Section 6 concludes this paper.

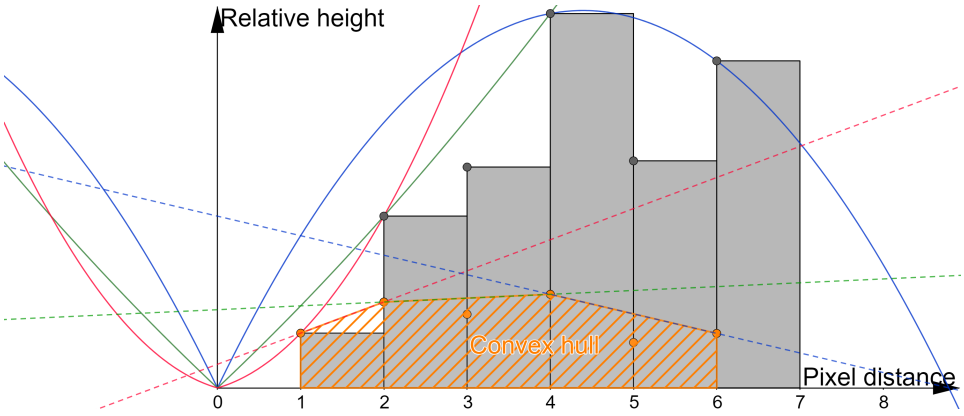


Figure 2: Distance–maximum height histogram for a given texel. The orange points are the height differences divided by the distance. The blue, red, and green lines form a convex boundary, and each line corresponds to a bounding parabola.

2 Quadratic displacement maps

The previously mentioned cone step mapping algorithms excel at rendering heightmaps in real-time; their most significant slowdown results from areas with high tangent slopes in the heightfield. Generally, the generated cones at these texels have a narrow opening angle, limiting the volume that can be skipped during raymarching, therefore, increasing the number of iterations required. While more steps taken per ray does not necessarily mean worse performance, the cost of multiple texture read queries on the GPU cause a significant amount of idle processing time, making the algorithm less effective.

We aim to reduce the number of iterations by generalizing the cones to conservative parabolic surfaces, thereby increasing the unbounding volume size where possible. Quadratic surfaces have a non-constant tangent that allows them not to be defined only by the closely surrounding height values.

However, surfaces defined by implicit quadratic equations may not be sufficient, as they cannot generally provide the necessary improvement in step size extension; hence we complicate the surface to consist of two parts. First, we define a cone with similar characteristics to Dummer's cones, although limiting the height of the cone to a predefined value which we specify as a ratio between the height of the texel and the maximal heightmap value. Then, we connect a revolution of the parabola to the edge of the cone to create a continuous surface, allowing the parabola to be defined by an independent parameter from the cone.

Our representation of the described surface consists of three parameters denoted by $a, b, c \in \mathbb{R}$. The first two values represent the coordiantes of a point on the plane relative to the position of the texel, defining a line segment as one side of the cone. The two-dimensional description is sufficient here due to the radial symmetry of the surface. The third value, c, specifies a parabola starting from (a, b) defined by

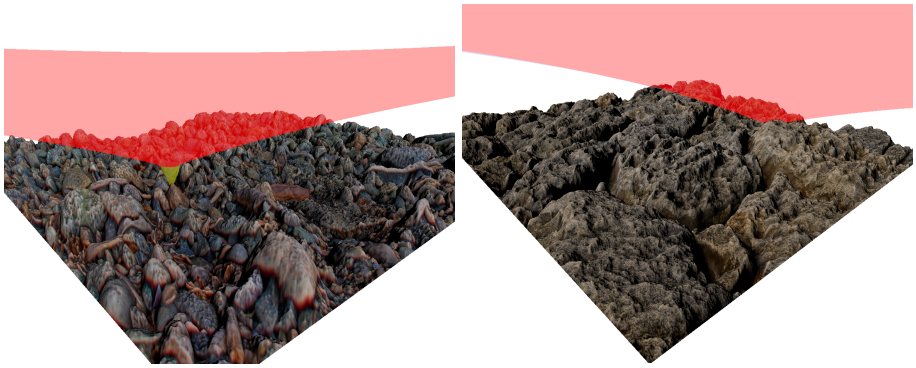


Figure 3: Generated quadratic surfaces with 0.5 (left) and 0.2 (right) height ratio for sample heightfields. The conic part on the left picture is colored yellow, and the quadratic parts are red on both sides.

the implicit equation

$$y = -(x-c)^2 + b + (c-a)^2 \qquad (a \geq 0, b \geq 0)\ .$$

Examples of such surfaces are shown in Figure 3. This representation requires storing three floating point values in addition to the height value for rendering. Thus, we equip the texture with four channels in our implementation.

3 Proposed algorithms

Our method traverses the empty space between higher heightmap elevations differently than similar techniques. We propose an algorithm for constructing unbounding parabolas and an efficient way to render heightmaps.

3.1 Generation of quadratic maps

Similarly to cone tracing, our ray tracing technique requires the revolved parabolas to be defined for every texel of the heightfield. For large textures, satisfying this condition demands time-efficient, parallelized construction of the unbounding surfaces with a small storage footprint.

First, for each (u, v) texel of the heightmap, we generate a radial function m_{uv} that returns the maximal relative height at a given distance from (u, v). An example of this function is shown in Figure 4. Let this function be defined for each $d \in \mathbb{N}$ integer pixel distance by

$$m_{uv}(d) = \max_{d \leq \left\|(x,y)-(u,v)\right\|_2 < d+1} \{h(x,y) - h(u,v)\} \qquad (x,y) \in \mathcal{D}_h.$$

After this transformation, we only need m_{uv} to find the optimal a, b parameters for a texel by advancing along the maximal height function, and in each step, we

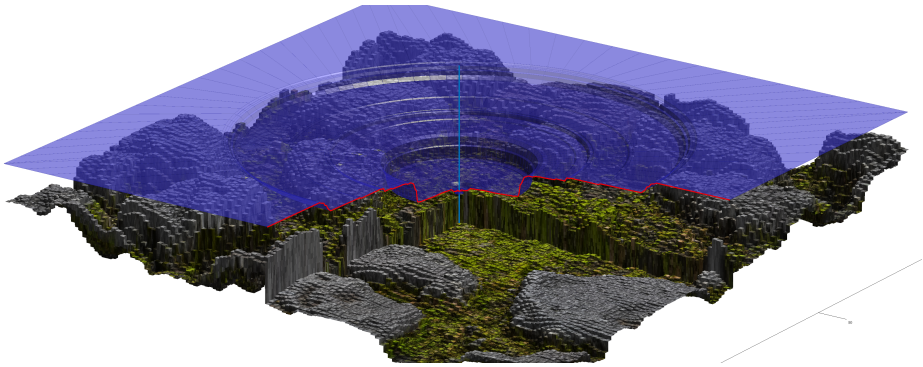


Figure 4: Generated radial function m_{uv} (red line) for a heightfield around the axis of the current texel (blue line). The value of the function at a given distance from (u, v) is the maximum height of the texels with the same distance.

choose $a \leftarrow i$ and $b \leftarrow \max\{i \cdot \frac{b}{a}, m_{uv}(i)\}$, where $i \in \mathcal{D}_{m_{uv}}$. We keep progressing until $i \cdot \frac{b}{a}$ exceeds the value of the height ratio parameter or i reaches the maximal distance. These steps can be pictured as searching for a cone which is a revolution of the line segment to (a, b) and does not intersect the heightfield but has the smallest slope possible. The result yields a finite-sized cone defined by the (a, b) point relative to the texel, as visualized in Figure 5.

With given values of a and b, we connect a revolved parabola to the top of the cone. Since these parabolas can be defined sufficiently in many ways, we choose a single equation to reduce the number of required parameters to one. This representation allows us to store all data for a parabola efficiently in four channels of a single texture. Let the implicit equation of the parabola be

$$y = -(x - c)^2 + b + (c - a)^2, \tag{1}$$

where $c \in \mathbb{R}^+$ remains to be determined. The global maximum of this curve is at c. Increasing this value guarantees that every point on the parabola between a and c will rise; satisfying our initial condition of avoiding intersection with the heightfield is trivial. This also means that the optimal value of c can be found by fitting a parabola to each point of m_{uv} and finding their global maximum. Thus, we solve the quadratic equations

$$m_{uv}(j) = -(j - c_j)^2 + b + (c_j - a)^2 \qquad j \in (a + 1, a + 2, \dots) \cap \mathcal{D}_{m_{uv}}$$

for c_j and let $c = \max c_j$.

After determining the a, b, and c parameters for each (u, v) texel, we include $h(u, v)$ and store the four floating point values in a texture to accelerate ray tracing. The complexity of the algorithm for a texture of size $N \times N$ is $\Theta(N^4)$ because the generation of m_{uv} requires checking all texels. However, since these calculations are independent, they can be parallelized.

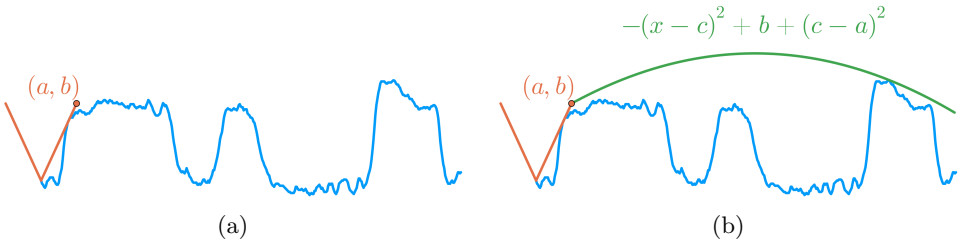


Figure 5: The two parts of generation of parabolas. First, we find values a and b, which define a cone (orange). Then, we calculate a parabola from (a, b) above the heightfield, defining an additional c parameter (green).

3.2 Rendering

During rendering, we cast rays from the geometry surface toward the heightfield, and our goal is to find their intersection using the quadratic maps described above. The raymarching algorithm is identical to the cone step mapping apart from the ray-geometry intersection calculations.

Let us define a ray starting from $\mathbf{p}_0$ and direction $\mathbf{v}$ by $\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}$, where $t \geq 0$. In each step of raymarching, we load the four values (a, b, c, and h) from the texel below $\mathbf{p}(t_i)$ (t_i being the current ray parameter), and find t where $\mathbf{p}(t)$ intersects the quadratic surface. We can simplify the problem into two dimensions due to the symmetry of the revolved geometries while obtaining the same results. Additionally, we only have to account for one parabola because the travel direction of the ray is known within the plane.

The entire curve is stitched together from two curve segments. The line segment and the parabolic curve share a single (a, b) point in the two-dimensional representation. When looking for an intersection with the ray, we separate these two parts and look at the line first.

Let us define $\mathbf{s}(\alpha) = \alpha(a, b)$, where an $\mathbf{s}(\alpha)$ point of the line is on the segment only if $\alpha \in [0, 1]$. Then, let the intersection of the two lines $\mathbf{p}(t)$ and $\mathbf{s}(\alpha)$ be (x, y), we have $\alpha = \frac{x}{a}$. If $\alpha \leq 1$, we have $t = \left\|\mathbf{p}(t_i) - (x, y)\right\|$.

If $\alpha > 1$, then substituting $\mathbf{p}_0 + t\mathbf{v}$ into Equation (1), we get

$$\mathbf{p}_{0_y} + t\mathbf{v}_y = -(\mathbf{p}_{0_x} + t\mathbf{v}_x - c)^2 + b + (c - a)^2 \tag{2}$$

a quadratic equation of t. If there is no real solution, then we terminate with no intersection; otherwise, let t_1, t_2 be two, not necessarily different, roots, thus $t = \min\{t_1, t_2\}$. The correctness of choosing the smaller value is because $\mathbf{v}_y < 0$ and $\mathbf{p}(t_i) > -c^2 + b + (c - a)^2$ holds by definition. For the latter inequality, it is important that it only holds if the previous $\alpha > 1$ is also true, although solving (2) would be unnecessary. Algorithm 1 formalizes this method, and Figure 6 shows two examples.

Algorithm 1 Tracing of quadratic map

Input: $\mathbf{p}_0$ ray origin, $\mathbf{v}$ ray direction, $a_{ij}, b_{ij}, c_{ij}, h_{ij}$ parabola parameters stored in a texture
Output: t distance along the ray

$\mathbf{p} \leftarrow \mathbf{p}_0; s \leftarrow 0$
while $s < steps \wedge t > \epsilon \wedge \mathbf{p}$ above heightfield **do**
  $(i, j) \leftarrow$ texel coordinates of $\mathbf{p}$
  $(x, y) \leftarrow$ intersection point of the ray and the line to (a_{ij}, b_{ij})
  $\alpha \leftarrow \frac{x}{a}$
  if $\alpha \leq 1$ **then** ▷ Intersected the line
    $t \leftarrow \|\mathbf{p} - (x, y)\|$
  else ▷ Check intersection with parabola
    $A \leftarrow \mathbf{v}_x^2$
    $B \leftarrow \mathbf{v}_y - 2 \cdot c_{ij} \cdot \mathbf{v}_x$
    $C \leftarrow \mathbf{p}_y - h_{ij} - b_{ij} - a_{ij}^2 + 2 \cdot a_{ij} \cdot c_{ij}$
    $t \leftarrow solveQuadratic(A, B, C)$
  end if
  $\mathbf{p} \leftarrow \mathbf{p} + t \cdot \mathbf{v}$ ▷ Step along the ray
  $s \leftarrow s + 1$ ▷ Increase step count
end while
return t

4 Further optimizations

The introduced algorithms above perform similarly to the classic cone step mapping method in both runtime and error metrics. Naturally, there is always room for improvement, and we have made some optimizations that we deemed necessary.

4.1 Convex bounds

We have proposed an algorithm for constructing conservative parabolas for a texel on a heightfield in Section 3. The generation consists of two steps: first, we calculate the values of the radial function m_{uv}, and while this is the more costly of the two parts in terms of performance, it requires further research. Second, we compute the a, b, c parameters of the parabola by iterating through every possible value of this function.

Since this iteration is linear in texture size, it can be more efficient to reduce the number of values using the same method with less repetition. The mentioned reduction is made by computing the upper convex bound of the m_{uv} values, excluding a significant number of possible parameters from the search. Upper convex bounds can be constructed in linear time according to [1].

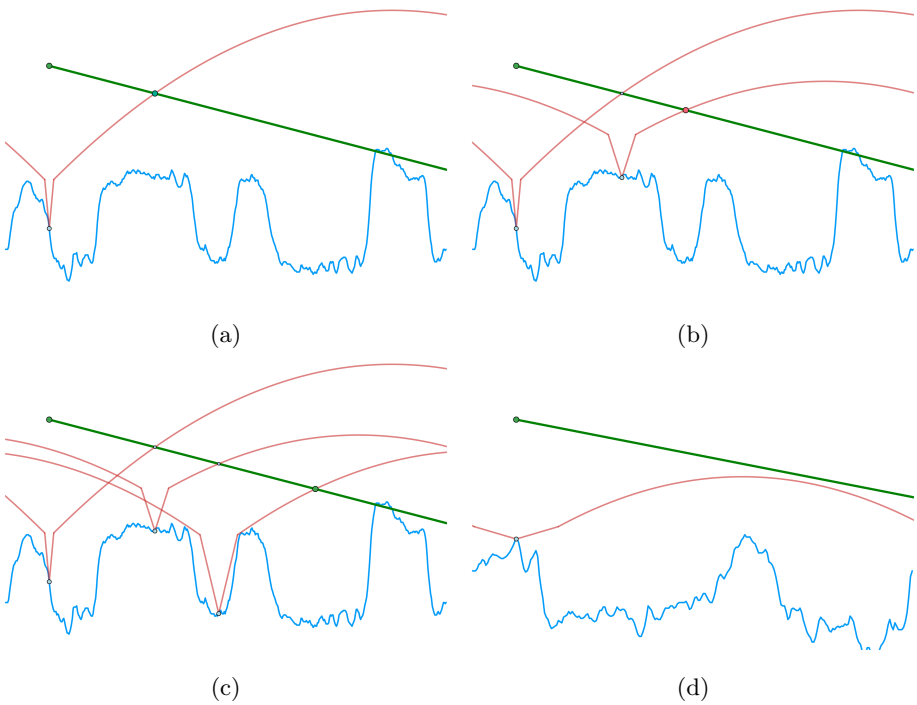


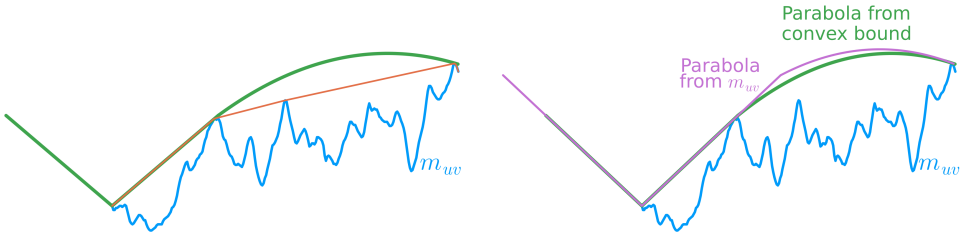
Figure 6: Ray tracing using quadratic maps. The first three pictures (a-c) show three tracing steps by reading the stored parabola in the current texel and calculating its intersection with the ray, resulting in the position of the next iteration. In picture (d), the ray misses the parabola, thus taking an arbitrarily large step.

This optimization, though not changing the overall complexity of the solution, allows to more efficiently separate the two phases of generation and reduces the required number of memory access queries. Additionally, the construction of the convex bound can be further accelerated and even performed directly from the heightfield, thus skipping the costly m_{uv} generation.

Note that with these changes, we will not always have the same parameters as a result, as demonstrated in Figure 7; however, it is guaranteed to preserve the conservative property of the parabolas.

4.2 Numerical stability

Time efficiency and numerical precision are critical during rendering to have the best results in the shortest possible time. It is known that finding the roots time-efficiently with minimal numerical error is a difficult task. Since we solve a quadratic equation in every iteration, we have to ensure that we do so in a numerically stable way.



(a) Example for parabola (green) generated using upper convex bound (orange) of the heightfield.

(b) Difference between a parabola generated from the radial m_{uv} function (purple) and from convex bound (green).

Figure 7: Parabola generated from convex bound

Blinn [3] has published a method to solve a general quadratic equation using homogeneous coordinates. Though robust and has a low error rate, it relies on several condition elevations, which can be time-consuming for real-time rendering. However, we can restrict the coefficients by considering how the values are computed.

Using the notations of Algorithm 1, it is guaranteed that $A \geq 0$ and $B < 0$, since $\mathbf{v}_x > 0, \mathbf{v}_y < 0$ and $c_{uv} > 0$ by definition. These inequalities allow writing a single conditional operator to yield the sufficient root of the equation, that is, checking if the root is real or not. The optimized quadratic equation solver is in Algorithm 2.

Algorithm 2 Numerically stable quadratic equation solver for parabolic maps

Input: $A \geq 0, B < 0, C \in \mathbb{R}$ coefficients
Output: smaller real root of $Ax^2 + Bx + C = 0$

$B \leftarrow \frac{B}{2}$
$M_1 = C$
$M_2 = -B + \sqrt{B^2 - AC}$
$x = \frac{M_1}{M_2}$

if $x \in \mathbb{R}$ **then**
    return x
end if
return ∞

5 Testing and results

The proposed method aims to reduce the number of steps taken along the rays during real-time heightfield rendering, thus lowering the GPU processing time of a single image. In this section, we compare the algorithms to Dummer's cone step

mapping by distance taken per iteration and runtime of frame rendering. While the current state-of-the-art method is the relaxed cone stepping, it fundamentally differs from our proposed conservative technique. Quadric steps do not require refinement, as we are not entering the surface during rendering. Thus, the preferred choice of comparison is the cone step mapping.

These algorithms were implemented and tested in the Falcor framework by NVIDIA [6] with 1920×1080 screen resolution on a GeForce GTX 1060M GPU. The listed results are the average values of renders on 9 different 1024×1024 sized heightmaps from Figure 8.

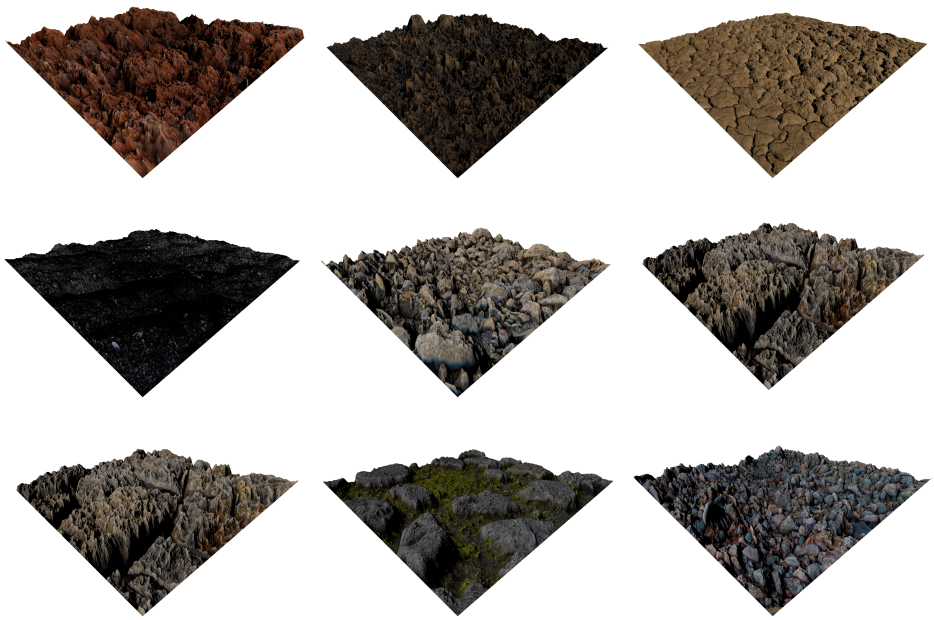


Figure 8: Heightmap samples used for testing the algorithms. Every texture has the same resolution of 1024×1024 pixels.

The height ratio parameter of the constructed quadratic surfaces is 0.2 in the following sections since this value seems to provide the best performance across our testing.

5.1 Step size and error

Both methods were analyzed by their performance compared to the same *ground-truth* image, a result of 200 iterations of linear search corrected with 20 steps of refinement. The absolute error for a ray is measured as the difference from this value. The rendered images are viewed from the same 8 camera positions that differ in incidence angle.

As shown in Figure 9, by increasing the number of maximal iterations, the total absolute error decreases for both algorithms as it is expected. In tests where the camera angle was below 45°, taking quadric steps usually gave significantly longer advancements along the ray because of the broad upper sections of the parabolas. Viewing the scene from higher than 45°, we lose some of this improvement; hence, the rays quickly reach the bottom part of the cones, which are similar in the two methods. Above 48 iterations, both algorithms seem to halt by reaching the height-field surface or leaving the geometry, so the difference between their errors becomes insignificant.

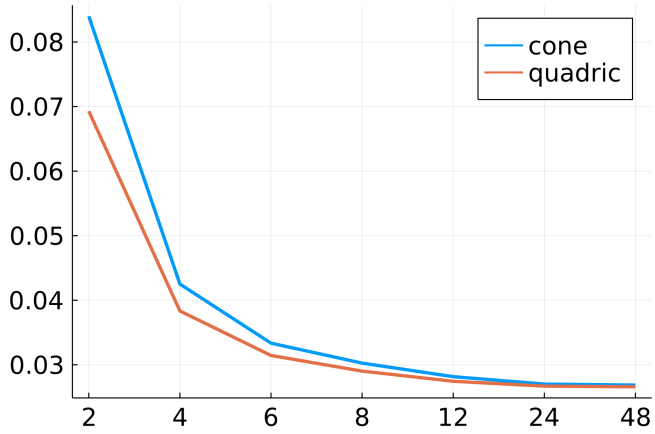


Figure 9: Average error in the distance taken on the ray (vertical axis) by iteration count (horizontal axis). Quadric stepping has generally lower error than cone step mapping, more significantly for fewer steps.

Figure 10 compares the average number of steps taken. Both algorithms follow a decreasing tendency by increasing the angle of the camera. Below 45° degrees, quadric mapping generally requires fewer iterations to converge. However, above 45° degrees, the method slows down as it approaches the surface.

5.2 Render time

We compared the two methods by average rendering speed for various heightfields from several camera angles and iteration numbers. Due to the composite nature of our representation and the fact that we have to resolve intersections with two different geometries, a single step of parabola tracing is computationally more expensive than that of cone step mapping. However, faster convergence properties allow for taking fewer steps, making our method more performant.

Our test results indicate that we can achieve better performance for view angles below 45° on all textures and maximal iterations. Although less noticeable, the overall mean rendering time for all angles is also reduced according to Table 1.

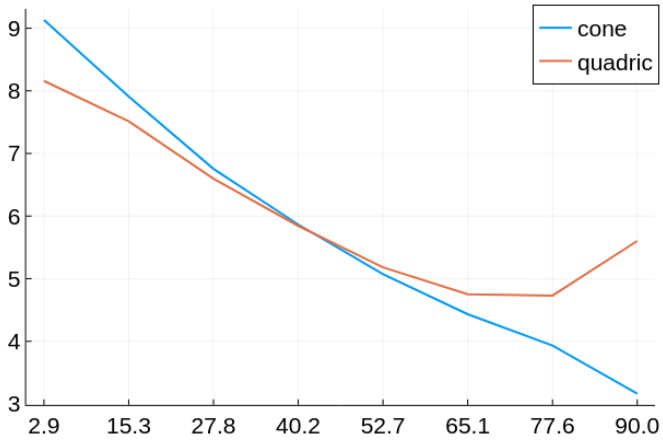


Figure 10: Average number of iterations before terminating of cone step mapping and quadric stepping (vertical axis). Up until 45°, quadric stepping performs better in general. The horizontal axis shows the angle in degrees.

Table 1: Average difference of render times between a single cone step and quadric steps in milliseconds. Negative values (highlighted) mean faster rendering for a quadric step. For angles below 45°, the sum of the values is −0.667, while for all values, it is −0.32, which means a faster average render time of a single image.

Angle	Cone minus Quadric render time (ms)						
	4 iters	6 iters	8 iters	12 iters	24 iters	48 iters	200 iters
2.9	**-0.023**	**-0.100**	**-0.047**	**-0.107**	**-0.093**	**-0.123**	**-0.260**
15.3	0.010	0.000	**-0.020**	**-0.010**	**-0.017**	**-0.023**	**-0.043**
27.8	0.027	0.017	**-0.003**	0.020	0.013	0.007	**-0.007**
40.2	0.023	0.017	0.013	0.027	0.023	0.013	0.000
52.7	0.020	0.013	0.010	0.023	0.020	0.010	0.010
65.1	0.013	0.010	0.010	0.017	0.020	0.010	0.000
77.6	0.010	0.007	0.010	0.017	0.010	0.000	0.000
90.0	0.010	0.010	0.013	0.027	0.027	0.010	0.010
Total:	0.090	**-0.027**	**-0.013**	0.013	0.003	**-0.097**	**-0.290**

There is less than 1% difference in the average runtime ratio between the two methods, where cone step mapping performed better as in Figure 11.

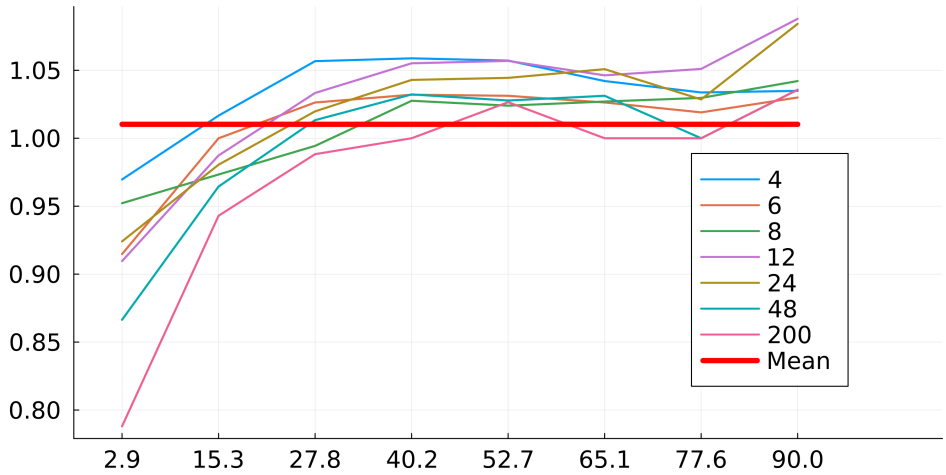


Figure 11: The average render speed of quadric steps compared to the cone stepping. The ratio (vertical axis) is below 1 for smaller angles (faster by $5-20\%$), and above 30°, it becomes $1-5\%$ slower. The lines represent different measures with varying numbers of maximum iterations.

6 Conclusion

We proposed a method for efficient real-time ray tracing of heightfields, utilizing revolved parabolas stored in a four-channeled texture. We introduced algorithms for generating these parabolas and rendering the surface with additional optimizations.

The algorithms were compared to the cone step mapping technique in extended testing by convergence speed and frame render time. The tests showed that our method performed better in both metrics when the camera view angle was low and produced similar results otherwise. The slowdown can be originated from the arithmetic costs of a single ray-parabola intersection computation that we plan to optimize in the future.

The main improvement of our method showed in faster convergence of rays, which is achieved by taking longer steps in most of the iterations. This indicates that it can be efficient for rendering heightfields with more expensive queries such as procedural textures. We plan to explore these possibilities in the future.

Currently, the generation algorithm of the parabola maps demands high memory and computing capacity, which requires further optimization. We are currently experimenting with alternative methods for construction that could radically reduce the resources needed.

References

[1] Andrew, A. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979. DOI: 10.1016/0020-0190(79)90072-3.

[2] Bálint, C. and Kiglics, M. A geometric method for accelerated sphere tracing of implicit surfaces. *Acta Cybernetica*, 25(2):171–185, 2021. DOI: 10.14232/actacyb.290007.

[3] Blinn, H. How to solve a quadratic equation? *IEEE computer Graphics and Applications*, 25(6):76–79, 2005. DOI: 10.1109/MCG.2005.134.

[4] Dummer, J. Cone step mapping: An iterative ray-heightfield intersection algorithm, 2006. URL: https://www.scribd.com/document/57896129/Cone-Step-Mapping.

[5] Halli, A., Saaidi, A., Satori, K., and Tairi, H. Per-pixel displacement mapping using cone tracing. *International Review on Computers and Software*, 3(5):1–11, 2008.

[6] Kallweit, S., Clarberg, P., Kolb, C., Davidovič, T., Yao, K.-H., Foley, T., He, Y., Wu, L., Chen, L., Akenine-Möller, T., Wyman, C., Crassin, C., and Benty, N. The Falcor rendering framework, 2022. URL: https://github.com/NVIDIAGameWorks/Falcor.

[7] Policarpo, F. and Oliveira, M. Relaxed cone stepping for relief mapping. *GPU Gems 3*, 3:409–428, 2007. URL: https://developer.nvidia.com/gpugems/gpugems3/part-iii-rendering/chapter-18-relaxed-cone-stepping-relief-mappingl.

[8] Szirmay-Kalos, L. and Umenhoffer, T. Displacement mapping on the GPU — state of the art. *Computer Graphics Forum*, 27(6):1567–1592, 2008. DOI: 10.1111/j.1467-8659.2007.01108.x.