Quick cone map generation on the GPU

Gábor Valasek and Róbert Bán

Eötvös Loránd University, Hungary
{valasek, rob.ban}@inf.elte.hu

Abstract

We propose an efficient conservative cone map generation algorithm that has $\Theta(N^2 \log N)$ complexity for textures of dimension $N \times N$ in contrast to the $\Theta(N^4)$ complexity of brute-force approaches. This is achieved by using a maximum mip texture of a heightmap to process all texels during the search for cone apertures, resulting in real-time generation times. Furthermore, we show that discarding already visited regions of neighboring mip texels widens the obtained cones considerably while still being conservative. Finally, we present a method to increase cone aperture tangents further at the expense of conservativeness. We compare our methods to brute-force and relaxed cone maps in generation and rendering performance.

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing; Shape modeling;***

1. Introduction and Related Work

Variation in lighting due to fine geometric details is ubiquitous in realistic rendering. Its realization, however, is a difficult challenge. A brute-force display of such details, i.e. rendering dense triangle meshes, is impractical due to limited triangle throughput and the inefficiencies that result from rasterizing small triangles.

Blinn proposed to use perturbed surface normals [Bli78] to achieve the appearance of geometrically rich shapes without increasing the actual geometric complexity of the rendered model. These perturbations were computed from a heightmap imposed over the coarse geometry, essentially prescribing a displacement or offset along the surface normal.

Algorithms to render the actual offset surface per pixel were later developed, see [SKU08] for a detailed survey on these methods. In general, these aim to compute the intersection between the view ray and the heightfield and considerably increase realism as they also account for the parallax effect due to the variations in height.

Approximate algorithms may yield unstable images, where the visibility of various details may change depending on the current view. Cone step mapping [Dum06] is a conservative technique in the sense that it guarantees to find the first intersection and it is also efficient due to its ability to skip over empty spaces.

As such, cone maps are robust and efficient means to render mesostructural details over macrostructure geometries. However, generating cone maps is oftentimes delegated to brute-force algorithms, resulting in $\Theta(N^4)$ complexity for $N \times N$ images. As displacement map sizes increase even in real-time applications, this becomes a limiting factor.

Halli et al. [HSST08] propose a $\Theta(N^2)$ algorithm for cone map construction, however, the theta-notation hides a constant multiplier that is at least as large as the number of distinct height values in a heightmap, i.e. 256 for an 8 bit worst-case, and it relies on a series of linear time distance transforms.

We give a brief overview of the relevant notation and displacement mapping algorithms in Section 2. In Section 3, we propose a GPU-friendly conservative cone map generator algorithm that has a complexity of $\Theta\left(N^2 \log N\right)$ for $N \times N$ textures. It relies on a maximum mipmap pyramid of the heightfield texture. Instead of comparing every texel to every other texel directly to find the narrowest cone, the mip pyramid is used to compute a conservative cone from a given texel to all texels of a region, similarly to how [DT07] approximated horizon visibility.

This generates narrower cones than the naive algorithm, which can be either addressed by a relaxed cone step mapping render algorithm or by heuristically altering the cone aperture computation in our algorithm. These are validated in Section 4.

2. Per-pixel heightfield rendering

At every fragment, the heightfield represents a displacement in the $(\boldsymbol{p};\boldsymbol{t},\boldsymbol{b},\boldsymbol{n})$ tangent space, where $\boldsymbol{p} \in \mathbb{E}^3$ is the world space position of the current fragment, $\boldsymbol{t},\boldsymbol{b} \in \mathbb{R}^3$ are the unit tangent and bitangent vectors, and $\boldsymbol{n} \in \mathbb{R}^3$ is the unit surface normal. The displacement is applied along $\boldsymbol{n}$. The XY coordinates of any point can be used as texture coordinates to sample the heightfield texture and the Z coordinate is the current ray height.

For every rasterized fragment of the primitive, we transform the view ray into tangent space and compute its intersection with the

DOI: 10.2312/egs.20221021

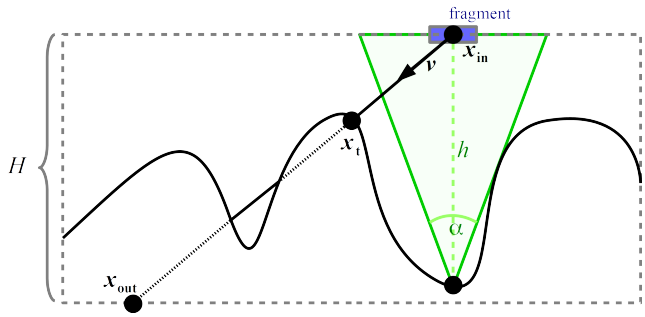

Figure 1: *Illustration of downward displacement. The view ray is intersected with the virtual extruded primitive (gray dashed box) to establish entry- ($\boldsymbol{x}_{in}$) and exit points ($\boldsymbol{x}_{out}$) of the ray into the volume that contains the heightfield. A cone (in green) is represented by the h height of its apex and a function of its α aperture, usually $\tan\frac{\alpha}{2}$.*

primitive and its translate along the surface normal by some prescribed maximum displacement height $H > 0$. This yields an entry point $\boldsymbol{x}_{in}$ and an exit position $\boldsymbol{x}_{out}$ between which we are looking for the first intersection, $\boldsymbol{x}_t$, between the $\boldsymbol{r}(t) = \boldsymbol{x}_{in} + t \cdot (\boldsymbol{x}_{out} - \boldsymbol{x}_{in})$, $t \in [0,1]$ ray and the heightfield, see Figure 1. It depends on convention whether positive heightfield values correspond to displacements in the direction of the normal ('upward displacement') or in the opposite direction to the normal ('downward displacement').

Linear search is a constant step ray march along $\boldsymbol{r}(t)$, that is, it looks for the first $i \in \{0,1,\ldots,N\}$ such that $f(\frac{i}{N}).z < 0$, where $f(t) = \boldsymbol{r}(t).z - h(\boldsymbol{r}(t).xy)$ and h is the heightfield. An optional root refinement may be used, such as binary search, to find a more exact location for the intersection between $f(\frac{i-1}{N}) > 0$ and $f(\frac{i}{N}) < 0$. This method may fail to report the first intersection, or any intersection at all, depending on step size.

The main idea of cone step mapping [Dum06] is to use unbounding cones over the heightfield to make intersection computations both more efficient and robust. These cones do not intersect with the interior of the heightfield, thus, the ray-heightfield intersection is resolved by iterative ray-cone intersection computations. Figure 1 illustrates such a cone.

A cone map entry akin to [Dum06, PO07] represents a tangent space cone by a $(h, \tan\frac{\alpha}{2})$ pair (or $(h, \sqrt{\tan\frac{\alpha}{2}})$, for a better distribution of digits), where h is the height of the apex of the cone and α is its aperture. It is argued in [HSST08] that this restricts the maximum cone apertures with unsigned normalized textures, and instead they propose to store $h\tan\frac{\alpha}{2}$, i.e. the maximum radius of the cone within the heightfield bounding volume, unnormalized.

Regardless of the choice of exact representation, a cone map texture can be used to obtain an interpolated cone over arbitrary points along the ray and the intersection between the ray and the interpolated cone is the next guess for the ray-heightfield intersection at each iteration.

For every texel of the heightmap, the cone can be computed by finding the smallest aperture between the current texel and any of all other texels, resulting in a $\Theta(N^4)$ algorithm for $N \times N$ textures.

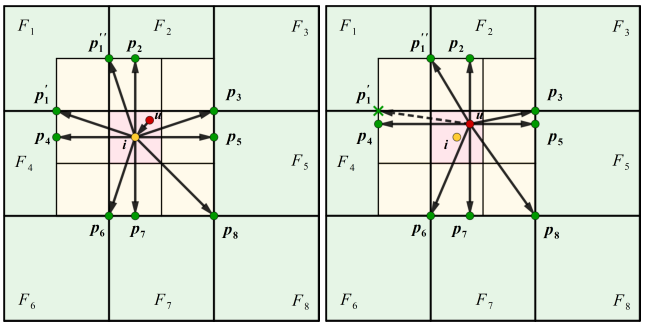

Figure 2: *Cone calculation on mip level k in a 3×3 neighborhood. The level 0 location of the current texel is denoted by $\boldsymbol{u}$. The texel at mip level $k-1$ that contains $\boldsymbol{u}$ has its center at $\boldsymbol{i}$ and we have already resolved its neighborhood, shown in yellow. Thus, we only have to consider the univisited parts of the 3×3 F_i neighborhood at level k, shown in green. The distance calculation itself is decomposed into two steps. First, we compute the level k closest points on the unvisited region boundaries to $\boldsymbol{i}$, that is, between the green regions and $\boldsymbol{i}$ (left). Second, we translate the results to mip level 0 by re-evaluation the distance between the footpoints and $\boldsymbol{u}$ (right).*

3. Region growing cone map generation

We propose a method that traverses a maximum mip pyramid of the heightfield to conservatively process all texels. In general, this results in tighter cones than the naive brute-force algorithm but at a much reduced generation time. Algorithm 1 summarizes this.

Algorithm 1 Naive quick cone map generation

1: **Input**: M max mip map generated from an $N \times N$ heightfield
2: **Output**: R result $N \times N$ cone map
3: **for** $\forall \boldsymbol{u} \in M[0]$ texel **do**
4: $h \leftarrow M[0][\boldsymbol{u}]$, $minTan \leftarrow 1$ // height of $\boldsymbol{u}$, narrowest tangent
5: **for** $\forall k$ level in M **do**
6: $\boldsymbol{i} \leftarrow \lfloor \frac{\boldsymbol{u}}{2^k} \rfloor$ // coordinates of the level k texel containing $\boldsymbol{u}$
7: **for** $\forall \boldsymbol{n} \in M[k] : \|\boldsymbol{n} - \boldsymbol{i}\|_\infty = 1$ level k neighbour **do**
8: $m \leftarrow M[k][\boldsymbol{n}]$ // maximum height within $\boldsymbol{n}$
9: **if** $m > h$ **then** // update current minimal cone
10: $F \leftarrow \{\boldsymbol{t} \in \mathbb{N}^2 \mid \exists i,j \in \{0,\ldots,2^k-1\} : \boldsymbol{t} = 2^k\boldsymbol{n} + \begin{bmatrix} i \\ j \end{bmatrix}\}$
11: $\boldsymbol{p} \leftarrow$ getClosest($\boldsymbol{u}, F$) // at level 0
12: $minTan \leftarrow \min\{minTan, \frac{\|\boldsymbol{p}-\boldsymbol{u}\|_2}{N(m-h)}\}$
13: **end if**
14: **end for**
15: **end for**
16: $R[\boldsymbol{u}] \leftarrow (h, minTan)$
17: **end for**

Note, however, that this algorithm is overly conservative about texels close to mip level boundaries. The more levels a texel is neighboring at their border, the narrower the estimated cone becomes, as the maximum heights from the higher mips will keep accumulating next to the initial texel due to Step 11.

We propose to improve on this by also considering what portions

of a given texel at the current mip level have been already covered previously and adjust the closest point accordingly. In particular, Figure 2 shows that certain regions of the coarser mip level can be discarded from being candidates to closest points. Algorithm 2 summarizes this approach.

Algorithm 2 Improved quick cone map generation

1: **Input**: M max mip map generated from an $N \times N$ heightfield
2: **Output**: R result $N \times N$ cone map
3: **for** $\forall \boldsymbol{u} \in M[0]$ texel **do**
4: $h \leftarrow M[0][\boldsymbol{u}]$, $minTan \leftarrow 1$, $V \leftarrow \{\boldsymbol{u}\}$ // set of visited texels
5: **for** $\forall k$ level in M **do**
6: $\boldsymbol{i} \leftarrow \lfloor \frac{\boldsymbol{u}}{2^k} \rfloor$ // coordinates of the level k texel containing $\boldsymbol{u}$
7: **for** $\forall \boldsymbol{n} \in M[k] : \|\boldsymbol{n} - \boldsymbol{i}\|_\infty = 1$ level k neighbour **do**
8: $F \leftarrow \{\boldsymbol{t} \in \mathbb{N}^2 \mid \exists i, j \in \{0, \ldots, 2^k - 1\} : \boldsymbol{t} = 2^k \boldsymbol{n} + \begin{bmatrix} i \\ j \end{bmatrix}\}$
9: $U \leftarrow F \setminus V$ // $\boldsymbol{n}$ is $2^k \times 2^k$ texels, U are unvisited so far
10: **if** $U \neq \emptyset$ **then**
11: $m \leftarrow M[k][\boldsymbol{n}]$ // maximum height within $\boldsymbol{n}$
12: **if** $m > h$ **then** // update current minimal cone
13: $\boldsymbol{p} \leftarrow$ getClosest($\boldsymbol{u}$, U) // at level 0
14: $minTan \leftarrow \min\{minTan, \frac{\|\boldsymbol{p}-\boldsymbol{u}\|_2}{N(m-h)}\}$
15: **end if**
16: **end if**
17: **end for**
18: **end for**
19: $R[\boldsymbol{u}] \leftarrow (h, minTan)$
20: **end for**

Our main observation is that Step 13 can be resolved by using the indices of the texels and the current mip level and F need not be represented explicitly. See Figure 2 for details.

For both algorithms, a common option to widen the cones is to replace the getClosest function by using the center of the texel of the current mip level. This may yield incorrect cones but since this error is proportional to the mip level, and that in turn is proportional to how far away we are from the currently processe texel, this can generally be mitigated by using a root refinement between the last two estimates after the initial cone step mapping. We refer to this as the texel center heuristic.

Figure 5 illustrates the results of the texel center heuristic applied to both the naive and improved algorithms.

4. Test results

The source code of our implementation is available at https://github.com/Bundas102/falcor-conemap.

4.1. Generation and render times

We compared the performance of the brute-force cone map (BFCM) generation algorithm [Dum06] with the relaxed cone map method [PO07] using $M = 64$ (RCM), our improved Algorithm 2 (QI) and its texel center heuristic modification from Section 3 (QC). The measurements are in milliseconds, taken on a notebook NVIDIA GeForce 2060 RTX and a desktop AMD RX 5700. The average overhead of using the improved algorithm Alg. 2 over Alg. 1 was about 6.52% on NVIDIA and 18.7% on AMD.

AMD RX 5700	BFCM	RCM	QI	QC	Mip
256^2	27	71	0.028	0.022	0.026
512^2	389	688	0.11	0.085	0.034
1024^2	N/A	N/A	0.44	0.36	0.074
2048^2	N/A	N/A	1.85	1.56	0.23
4096^2	N/A	N/A	8.33	6.87	0.84

NV 2060	BFCM	RCM	QI	QC	Mip
256^2	58.11	145.15	0.088	0.08	0.1
512^2	863.4	1563.24	0.27	0.26	0.13
1024^2	N/A	N/A	1.18	1.09	0.2
2048^2	N/A	N/A	4.91	4.69	0.41
4096^2	N/A	N/A	21.33	20.46	1.26

Our methods are running at real-time rates up to 2K resolution and produce cone maps at interactive speeds at 4K. Neither GPU could compute the brute-force [Dum06] and relaxed [PO07] cone maps in a single dispatch 1K resolution and up. We did not optimize the quick cone map generation code, neither the naive, multipass maximum mip generation. We did let the brute-force generation to early out should there be no chance to improve the cone aperture.

As Alg. 2 is conservative and the center heuristic is behaving similarly empirically, both methods can use a secant step for root refinement. Relaxed cone maps may guide rays into the interior of the geometery, which we resolved by binary search. Measurements were done on 1920×1080 resolution with 512×512 heightmaps using 32 and 200 cone map steps followed by 5 binary (RCMb) or 1 secant (RCMs, BFCM, QI, QC) root refinement steps. Timings are in milliseconds.

Steps	GPU	BFCM	RCMs	RCMb	QI	QC
32	**AMD 5700**	0.18	0.17	0.19	0.20	0.19
200	**AMD 5700**	0.23	0.2	0.21	0.26	0.25
32	**NV 2060**	0.45	0.41	0.45	0.48	0.46
200	**NV 2060**	0.53	0.46	0.49	0.62	0.59

On 32 steps, relaxed cone maps are about 7% faster to render than our approximations with secant search root refinement. When applying the same root refinement, relaxed cone maps become 11%-17% faster to render. Allowing 200 steps pronounces the efficiency of the wider cones found in relaxed cone maps, as renders using our cone maps take about 17%-27% more time.

We experimented with relaxing the step sizes by a constant multiplier on the quick cone maps. Similarly to relaxed cones, this necessitates binary search root refinement to achieve equal visual quality and did not yield significant performance improvements.

4.2. Cone aperture comparison

A particularly difficult aspect of any screen-space displacement technique is to overcome the presence of unstable shapes. These may be the result of incorrect intersections, e.g. not reporting the first one, or unconverged render iterations that reached the iteration cap without finding an intersection. The former may be handled by conservative empty space skipping, while the latter requires as large steps as possible.

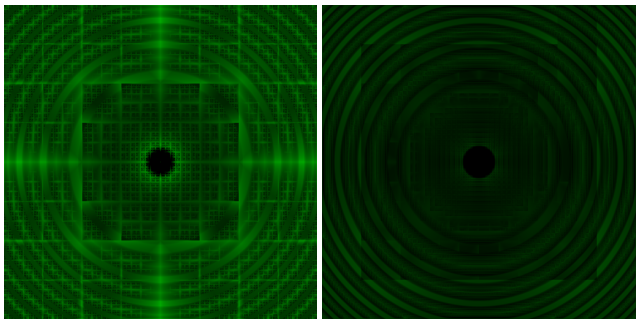

Figure 3: *Comparison between naive (left) and improved (right) quick cone map half aperture tangents with respect to a* 512×512 *naive ground truth. Brighter green denotes larger difference. The improved quick cone map algorithm yields wider cones.*

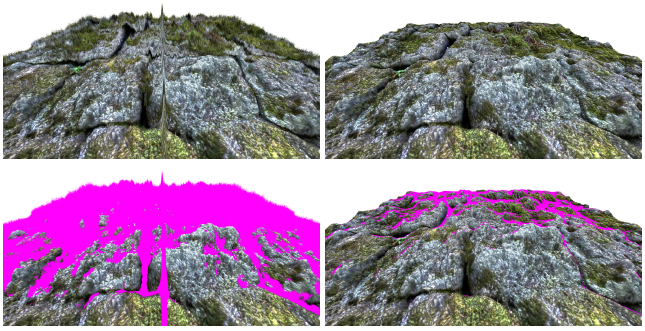

Figure 4: *Comparison of renders using the results of naive (left) and improved (right) quick cone map generation. Both cone maps are conservative. Magenta is not converged.*

Both Algorithm 1 and 2 may yield tighter cones than the brute-force algorithm. This means that cone step mapping may not be able to skip over the same span of empty spaces on these maps.

In general, Algorithm 2 generates at least as large apertures as Algorithm 1, and our tests show that these are in general considerably wider. See Figure 3 for the comparison of the computed tangents with respect to a ground truth obtained from a brute-force algorithm, and Figure 4 illustrates the two resulting cone maps in a render scenario.

5. Conclusions

We presented two algorithms to compute cone maps on the GPU using a maximum mip map. Their GPU implementation is straightforward and they can be executed at real-time rates, even for large resolution textures. We showed that Algorithm 2 produces larger cone apertures at the expense of a 10% increase in generation times.

The proposed algorithms generate cone maps that are conservative approximations to the ones obtained by the brute-force approach [Dum06]. As this incurs a performance penalty upon rendering, we proposed the texel center heuristic to increase the computed cone apertures in exchange of the strictly conservative property. Nevertheless, our tests on rendering textured terrain elements

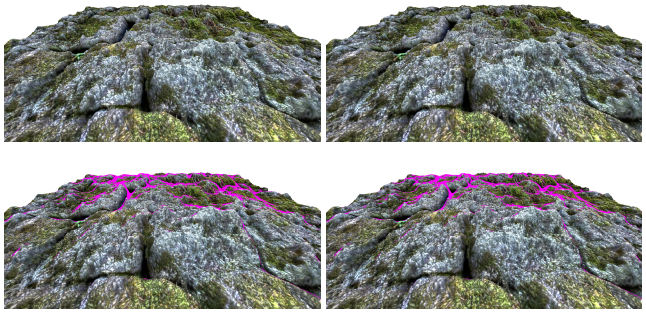

Figure 5: *Comparison of the naive (left) and improved (right) quick cone map generation results that use the texel centers as maximum candidates instead of the closest points of the mip texels. Magenta is not converged.*

showed that binary search is still not necessary for root refinement. These cone maps were at most 7% slower to render than relaxed cone maps with binary search in low iteration counts.

Both the proposed improved conservative and texel center heuristic algorithms are well suited for interactive or even real-time cone map generation scenarios. In more render performance oriented applications we suggest using the texel center heuristic maps.

Acknowledgement

Supported by the ÚNKP-21-3 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund. We would like to thank Visual Concepts for providing the AMD GPU used in the tests.

References

[Bli78] BLINN J. F.: Simulation of wrinkled surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1978), SIGGRAPH '78, Association for Computing Machinery, p. 286–292. URL: https://doi.org/10.1145/800248.507101, doi:10.1145/800248.507101. 1

[DT07] DACHSBACHER C., TATARCHUK N.: Prism parallax occlusion mapping with accurate silhouette generation. In *Symposium on Interactive 3D Graphics and Games poster* (2007). 1

[Dum06] DUMMER J.: Cone step mapping: An iterative ray-heightfield intersection algorithm. 2006. URL: http://www.lonesock.net/papers.html. 1, 2, 3, 4

[HSST08] HALLI A., SAAIDI A., SATORI K., TAIRI H.: Per-pixel displacement mapping using cone tracing. *International Review on Computers and Software 3* (09 2008), 1–11. 1, 2

[PO07] POLICARPO F., OLIVEIRA M. M.: Relaxed cone stepping for relief mapping. In *GPU Gems 3* (2007). URL: https://developer.nvidia.com/gpugems/gpugems3/part-iii-rendering/chapter-18-relaxed-cone-stepping-relief-mapping. 2, 3

[SKU08] SZIRMAY-KALOS L., UMENHOFFER T.: Displacement mapping on the gpu - state of the art. *Comput. Graph. Forum 27*, 6 (2008), 1567–1592. URL: http://dblp.uni-trier.de/db/journals/cgf/cgf27.html#Szirmay-KalosU08. 1