CEUR-WS.org/Vol-3027/paper8.pdf

Reliable GPU-Based Methods and Algorithms of Implementation Dynamic Relief Shadows in Virtual Environment Systems

Petr Timokhin [1] and Mikhail Mikhaylyuk [1]

[1] *Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences", Nakhimovskii pr. 36/1, Moscow, 117218, Russia*

Abstract

The paper considers the task of real-time rendering of dynamic relief shadows based on ray casting using origin relief data - detailed height map. The solution proposed is based on looking for shadow rays - sun rays, whose tracks on height map are passed through heights occluding the light. GPU-based methods and algorithms for extracting such rays using an accelerating data structure - a mipmap of maximum and minimum relief heights are developed. This structure provides an effective acceleration of shadow rays extraction by skipping long sections of sun ray tracks that are not involved in relief shadowing. In algorithms developed precise traversing such a data structure is implemented, as well as texture filtering is taking into account, which allows the formation of "torn" shadow edges to be prevented. The solution created was implemented in software complex and a number of comparative shadow visualization tests was conducted. The results of the research can be used in virtual environment systems, video simulators, scientific visualization, educational applications, etc.

Keywords

Relief shadows, height map, ray casting, GPU, virtual environment system

1. Introduction

One of the most important tasks of modern virtual environment systems is modelling of realistic light-shadow appearance [1, 2], in particular, dynamic relief shadows. This is especially demanded for space, air and ground video simulators [3, 4], where, in order to instill right skills to trainees, dynamic shadows should be of high quality and rendered in real-time (at least 25 times per second).

For such systems, an important feature of developing shadow algorithms is their reliability, i.e. the ability to construct correct shadows for any permissible directions of light sources, as well as the absence of freezes of visualization loop in case of unintentional errors in input data.

From the point of ensuring the correctness of obtained shadows, the best results are derived by algorithms based on ray casting. However, this approach requires the implementation of a per pixel ray processing loop, that, firstly, is associated with heavy time costs, and, secondly, carries potential risks of hanging the visualization system (obtaining an infinite loop). In this paper, we propose algorithms of original implementation of GPU-accelerated ray casting, which ensure reliable real-time relief shadow visualization.

2. Related work

In the field of real-time shadow rendering, a significant part of researches is devoted to the development of shadow mapping approach. One of the most common is the method of cascaded shadow mapping [5] (or parallel-split shadow maps [6]), in which a number of shadow maps of the same size for the foreground, middle and background are created from light source position (often up to 4 splits). Owing to good performance/shadow quality ratio, this method is, in fact, standard for many gaming

GraphiCon 2021: 31st International Conference on Computer Graphics and Vision, September 27-30, 2021, Nizhny Novgorod, Russia
EMAIL: webpismo@yahoo.de (P. Timokhin); mix@niisi.ras.ru (M. Mikhaylyuk)
ORCID: 0000-0002-0718-1436 (P. Timokhin); 0000-0002-7793-080X (M. Mikhaylyuk)
© 2021 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

applications. At the same time, the method requires careful adjustment of the shadowing algorithm for a target virtual scene in order to avoid the effect of "blocky" boundaries of shadows (especially elongated ones), as well as other undesirable artifacts (truncation of shadows, erroneous self-shading, etc.). To overcome these disadvantages, solutions based on percentage-closer filtering [7], nonlinear shadow mapping [8, 9], irregular Z-buffer shadow mapping [10], etc. were proposed.

Second major approach is based on constructing and rendering of accurate polygonal model of shadow volume for each object in the scene [11-13]. Having knowledge about object geometry, shadow volumes allow to obtain high-quality shadows at any distance from light source. Also supporting stencil buffers and other required features by modern graphics cards make this approach quite attractive for real-time rendering. The limitation of this approach is tightly bound to its geometric advantage: to solve silhouette detection task, a number of restrictions on the allowable geometry representations is imposed. This leads to complicating data structures and degradation of scaling ability while increasing scene complexity. If the scene comprises extended object with intricate silhouette (as the relief), then the implementation of shadow volumes can easily consume the performance of the entire visualization system.

It is important to note that both at first and second approaches, the reliability of relief shadow rendering will depend not only on shadow algorithm, but also on the reliability of the methods of constructing and rendering of a 3D relief model. These methods may contain hidden defects or optimizations that are invisible during the visualization of the 3D model, but revealing when shadows are rendered. In "Results" section we demonstrate such situation using a simple example.

A completely different principle of shadow rendering is used in the third approach, which involves working directly with the primary data of 3D model, – a height map of its surface [14-15]. This approach is based on looking up intersections of light source rays with object height map (shadow ray casting), which allows physically correct shadows from objects of complex shape to obtained, independently of 3D-model construction and visualization methods. However, brute-force ray casting implementation [16] including loop checking all map texels lying on ray path, is extremely time-consuming for real-time visualization. In this regard, there arises the task of developing effective and reliable methods and algorithms of accelerating ray casting, which preserve the quality of origin ray casting shadows and ensure guaranteed completion of ray processing loops.

The authors of the research [17] parallelize on the GPU execution of ray processing loops using the CUDA hardware-software architecture, however, implemented uniform sampling of height map along the ray can lead to missing some texels and loss of shadow quality. In [18], the task of relief mapping based on ray casting is considered, where looking up ray - height map intersections is accelerated by means of pre-calculated cone map. A significant limitation is great increase of cone map construction time depending on height map size (about 15 minutes for 512^2 and about 8 hours for 1024^2 [19]). The paper [19] describes the task of real-time visualization of dynamic height maps based on ray casting accelerated using a maximum mipmaps calculated "on the fly" on the GPU. The authors of the work [20] extended these ideas of ray casting acceleration to shadow rendering, proposing own implementation algorithm. As the authors mention, their algorithm suffers from insufficient accuracy leading to appearing fake shadows and missed terrain parts, which has to be compensated by adding some pre-displacement to terrain vertices.

Our work takes its origins from our earlier researches on ray casting implementation [21, 22]. Based on the experience gained, in this work, a core of our shadow ray casting system was developed – a reliable and accurate algorithm of traversing height map texels along the ray, adapted for the GPU. The key features of the algorithm are original "transit points" method for accurate height map sampling, which prevents texel missing (and potential loss of shadow information), as well as performs an incremental integer transition from texel to texel, which ensures guaranteed progress along the ray and prevents freezes of ray processing loop due to machine error in the representation of real numbers. An important advantage of our algorithm over [16] is no dependency from steps of previous iterations, which allows current step along the ray to be effectively and accurately changed.

During the development of shadow ray casting acceleration method, we, regardless studies [19, 20], logically came up to a data structure similar to maximum mipmap [19], where not only maximum heights are stored, but minimum heights too, as well as improved construction method is used, providing better stitching the mipmap with height map. Checking maximum heights allow large parts of height map, not involving in shadow formation, to be effectively skipped, while checking minimum heights allow ray processing to be earlier completed with shadow formation. If ray casting algorithm contains any inaccuracies or errors, the second check will give fake shadows that are immediately noticeable on the image. This unique property of our accelerating data structure allowed many inaccuracies of ray-casting algorithm to be identified and fixed, and, ultimately, a reliable shadow ray casting system performing entirely on the GPU, to be created. The proposed solution is implemented in C++ using the GLSL shading language and the OpenGL graphics library.

3. Our definitions and models

First, we will describe definitions of detailed height map, shadow ray and shadow coefficient, as well as some necessary mathematical background, used in our solution.

Detailed height map. Let there be relief section (for simplicity - a unit square), where the height above sea level for each point is known. Height map is a single-channel $2^m \times 2^m$ texture, where height values (in grayscale), sampled by the step of $1 / (2^m - 1)$, are written. Figure 1 shows an example of constructing 4×4 height map. In our work height values are extracted from map by means of bilinear interpolation [23]. So, for an arbitrary height map point with texture coordinates s, $t \in [0, 1]$, the following height expression can be written

$$h(s,t) = \big(h_{i,j}(1-s) + h_{i,j+1}s\big)(1-t) + \big(h_{i+1,j}(1-s) + h_{i+1,j+1}s\big)t, \tag{1}$$

where $h_{i,j}$, $h_{i,j+1}$, $h_{i+1,j}$, $h_{i+1,j+1}$ are height values of four nearest texels forming a square around the point under consideration (see Figure 1). As it can be seen, interpolated height values will be the closer to origin heights, the higher is m detail degree of height map. Height maps with m close to screen resolution (Full HD, 4K) will be referred to as *detailed* ones ($m = 10,..., 12$).

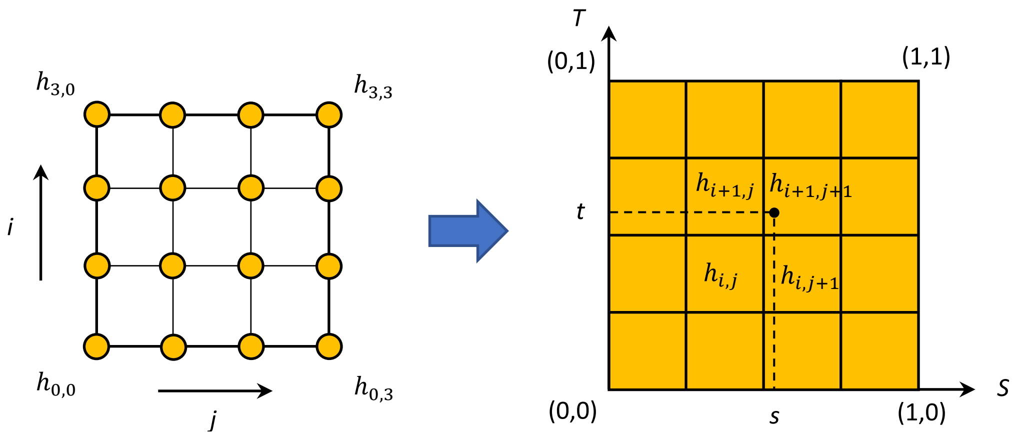


Figure 1: Constructing relief height map

In this paper, detailed height maps of relatively small terrain areas are considered, so that for shadow construction task the curvature of sea level surface can be neglected, i.e. considered as flat. An example is the Puget Sound height map [24], being mentioned in many papers related to terrain modeling.

Shadow ray. Let there be relief model given by height map and the normal $\boldsymbol{n}$ to sea level surface, as well as infinite light source (the sun) given by direction $\boldsymbol{s}$ to the sun (unit vector). The coordinates of vectors $\boldsymbol{s}$ and $\boldsymbol{n}$ are set in relief model Object Coordinate System (OCS). Let's choose some two points A and B of relief model, lying in the same plane with vectors $\boldsymbol{s}$ and $\boldsymbol{n}$, and determine the shadow at point A caused by the relief from point B (see Figure 2a). Denote by b_{shadow} boolean flag of shadow presence at point A (0 - no, 1 - yes). Cast the ray $\boldsymbol{v}$ from point A to point B. Figure 2a shows that the point A will be in the shadow, if the vector $\boldsymbol{s}$ is directed no higher than the ray $\boldsymbol{v}$. Define this using mixed product p [25] of the following vectors

$$p = \boldsymbol{r} \cdot (\boldsymbol{v} \times \boldsymbol{s}), \tag{2}$$

where $\boldsymbol{r} = \boldsymbol{s} \times \boldsymbol{n}$ is unit vector, complementing vectors $\boldsymbol{s}$ and $\boldsymbol{n}$ to the right triple. Then the expression of b_{shadow} flag can be written as

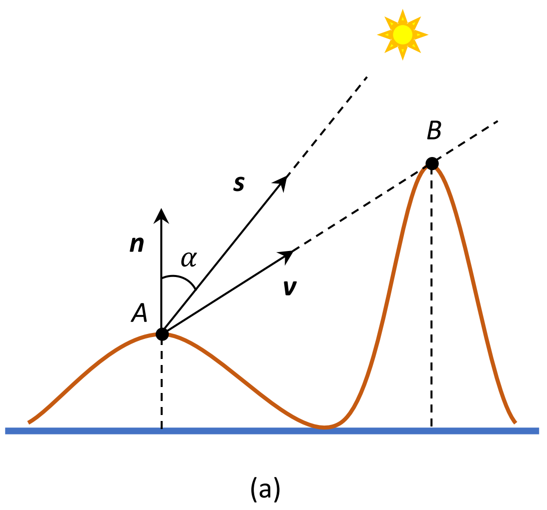
$$b_{shadow} = \begin{cases} 1, if\ p \le 0\ and\ (\boldsymbol{s} \cdot \boldsymbol{n}) \ne 1, \\ 0, otherwise. \end{cases} \tag{3}$$

If b_{shadow} is 1, then $\boldsymbol{v}$ is *shadow ray*.

Shadow coefficient. Let the point A is in shadow (b_{shadow} is 1). Introduce $k_{shadow} \in [k_{dark}, k_{bright}]$ - *shadow coefficient* of the point A, where k_{dark} corresponds to the darkest shadow, and k_{bright} - to the brightest one, and $k_{dark}, k_{bright} \in [0, 1]$. Define k_{shadow} as $cos(\alpha)$ (see Fig. 2a) reduced to the range $[k_{dark}, k_{bright}]$:

$$k_{shadow} = k_{dark} + (k_{bright} - k_{dark})(\boldsymbol{s} \cdot \boldsymbol{n}). \quad (4)$$

Figure 2b shows a plot of $k_{shadow}(\alpha)$ with $k_{dark} = 0.05$, $k_{bright} = 0.4$. As it can be seen from the plot, shadows become darker as the sun approaches the horizon ($\alpha \to \pi/2$), and, conversely, shadows are brighter, when the sun is at zenith ($\alpha \to 0$), what is explained by the contribution of ambient light. Note, that thresholds k_{dark} and k_{bright} are specified based on brightness range used for virtual scene rendering. This is especially actual for modern visualization systems, where rendering is performed in high dynamic range [26].



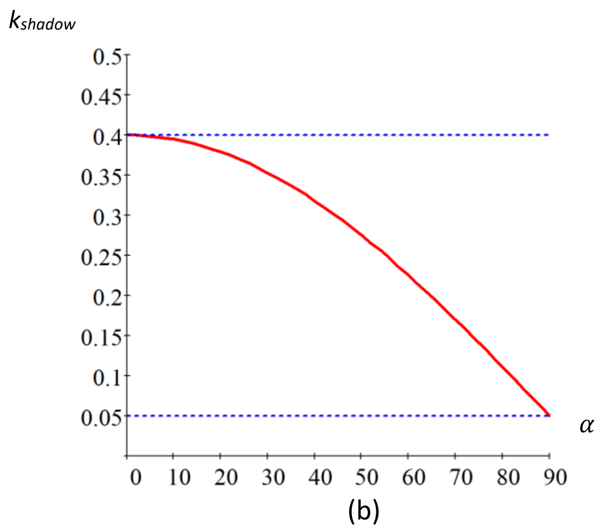


Figure 2: Determination of the shadow at point A, (a) and the plot of shadow coefficient $k_{shadow}(\alpha)$, (b)

4. Proposed solution

Consider the task of constructing shadows on the image of relief 3D model, synthesized at visualization stage. Let's choose an arbitrary pixel of this image. Chosen pixel corresponds to some point P of relief model and point P' on height map. Denote by s' the track of sun ray drawn from the point P' (in texture space), and by T - the set of texels intersected by the track s' (see Figure 3). Each $T_{i,j}$th texel has corresponded interpolated point $P_{i,j}$ of relief model (honestly, each (i, j)th section of the track s' has corresponded interpolated curve on relief model, however, this can be neglected on detailed height map). Denote by $\boldsymbol{v}_{i,j}$ the ray casted from point P to point $P_{i,j}$. As it can be seen, the point P will be in the shadow, if at least one of all $\boldsymbol{v}_{i,j}$th rays is shadow ray. Then to solve the task, every pixel of relief image, having $\boldsymbol{v}_{i,j}$th shadow ray, should be extracted, and its brightness decreased by k_{shadow} times.

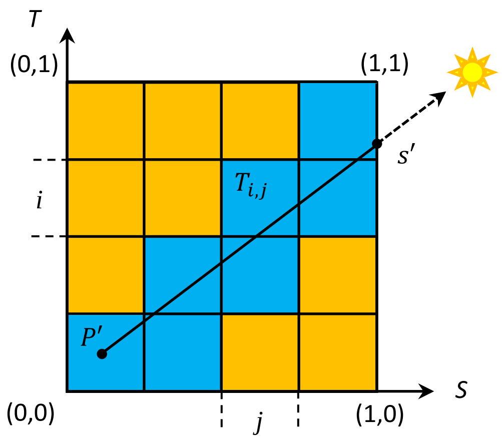


Figure 3: «Footprint» of sun ray track

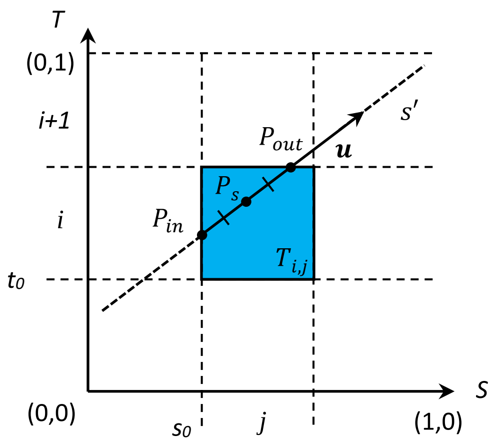


Figure 4: Transit points

In this work, pixels of relief image that to be shadowed, are extracted completely on the GPU, in parallel, independently of each other using developed fragment shader. This shader checks the texels of the "footprint" of the track s' either until the first $T_{i,j}$th pixel having shadow ray $\mathbf{v}_{i,j}$th will be detected, or until the "footprint" will end. This checking includes the following steps: (a) extracting height value from the $T_{i,j}$th texel; (b) constructing $\mathbf{v}_{i,j}$th ray and checking its belonging to shadow rays, and (c) moving to the next $T_{i,j}$th texel. Step (b) can be found in the previous Section, so we focus on steps (a) and (c).

To avoid the formation of "torn" shadow edges, we extract height value from the $T_{i,j}$th texel using bilinear interpolation (see Eq. (1)) and take each sampling point P_s as the average of two *transit points*: input point P_{in} and output point P_{out} of the track s' in the $T_{i,j}$th texel (see Figure 4). As it can be seen from the figure, the point P_{out} of the $T_{i,j}$th texel will be the point P_{in} for the $T_{i+1,j}$th texel, thus, when checking each $T_{i,j}$th texel, it is enough to calculate only one transit point P_{out}. Denote by $\mathbf{u}$ unit vector of track s' direction, then coordinates of the point P_{out} can be written in parametric form $P_{out} = P_{in} + \mathbf{u}t$. The calculation of t parameter, as well as shifts (i_{offset}, j_{offset}) of next texel along the track (step (c)), are implemented in developed algorithm *A1*.

Algorithm A1

```
1. Write texture coordinates array $K$ for 4 corners of $T_{i,j}$th texel:
        $K[4] = \{ \{s_0, t_0\}, \{s_0 + ds, t_0\}, \{s_0, t_0 + dt\}, \{s_0 + ds, t_0 + dt\} \}$,
   where $ds = 1.0 / w$, $dt = 1.0 / h$, and $w$, $h$ are height map width and height (in texels).
2. If $(|u_x| \le \varepsilon)$ and $(|u_y| \le \varepsilon)$, then   // $\varepsilon$ - machine error of real numbers.
        $t = -1$; $(i_{offset}, j_{offset}) = (0, 0)$; exit the algorithm.
3. Calculate the number $n_{corn}$ of the corner of $T_{i,j}$th texel, which is intersected by track $s'$:
        $n_{corn} = b_0 + 2b_1$, where $b_0$, $b_1$ are boolean flags, $b_0 = (|u_x| \ge 0)$, $b_1 = (|u_y| \ge 0)$.
4. Write the difference $dP = K[n_{corn}] - P_{in}$.
5. If $(|u_x| \le \varepsilon)$, then: $t = |dP_y|/|u_y|$; $(i_{offset}, j_{offset}) = (\mathrm{sign}(u_y), 0)$; exit the algorithm.
6. If $(|u_y| \le \varepsilon)$, then: $t = |dP_x|/|u_x|$; $(i_{offset}, j_{offset}) = (0, \mathrm{sign}(u_x))$; exit the algorithm.
7. Write $tVec = dP/u$ and boolean flags $b_3 = (tVec_x \le tVec_y)$, $b_4 = (tVec_y \le tVec_x)$.
8. $t = \min(tVec_x, tVec_y)$;   $(i_{offset}, j_{offset}) = (\mathrm{sign}(u_y) \cdot b_4, \mathrm{sign}(u_x) \cdot b_3)$.
End of the algorithm.
```

In practice, visiting all texels is not always necessary and effective, since height maps almost always contain large areas (plains, valleys, etc.) not involved in shadow constructing (provided not very low sun altitude). We localize such areas and skip them when our checking algorithm reaches such area border. To localize the areas, the mipmap $\{h_{max}, h_{min}\}$ of maximum and minimum heights is created. It is an ordered set of two-channel textures (R and G components) with the following properties:

1) For 0th map (the first one):

- dimensions are two times smaller than height map - $2^{m-1} \times 2^{m-1}$ (similar to classic mipmaps [27]);
- each (i_0, j_0)th texel stores h_{max}, h_{min} values sampled from height map using 3×3 matrix (see Fig. 5a);

2) For kth map (except 0th):

- dimensions are two times smaller than $(k$-1)th map, but 1×1 map is not created;
- each (i_k, j_k)th texel stores h_{max}, h_{min} values sampled from $(k$-1)th map using 2×2 matrix (see Fig. 5b).

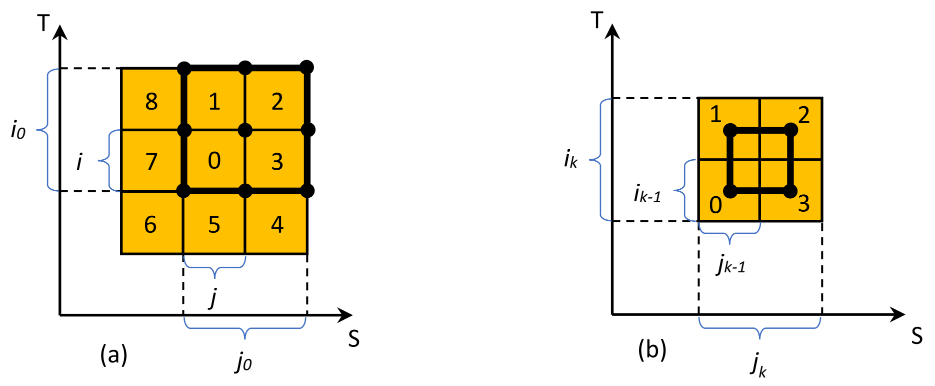


Figure 5: Sampling matrices for constructing mipmap $\{h_{max}, h_{min}\}$: (a) 3×3, (b) 2×2

Note, that in 3 × 3 matrix sampling points are upper right corners of 0,…,8th texels of height map, and sampling is performed using bilinear interpolation (GL_LINEAR, see Eq. (1)). In 2 × 2 matrix sampling points are centers of 0,…,3th texels of (k-1)th map, and sampling is performed using the "nearest neighbor" principle (GL_NEAREST). The mipmap $\{h_{max}, h_{min}\}$ is constructed "on the fly" on the GPU by means of developed fragment shader once before visualization.

Let the mipmap $\{h_{max}, h_{min}\}$ contain just 2 maps (for simplicity). Introduce the following definitions (see Figure 6):

- l is level of detail (LOD): 0 is height map, 1 and 2 are 0th and 1st maps of mipmap $\{h_{max}, h_{min}\}$;
- (i, j) are (row, column) numbers of start texel at 0th LOD;
- $(i_{cur,l}, j_{cur.l})$, $(i_{next,l}, j_{next,l})$ are (row, column) numbers of current and next texel along the ray at lth LOD;
- P_{out} is output point of the ray in start texel;
- $P_{out,l}$ is output point of the ray in $(i_{cur,l}, j_{cur.l})$th texel;
- P is the point on relief model (in OCS), corresponding to P';
- $P_{min,l}$, $P_{max,l}$ are the minimum and the maximum points of relief model (in OCS), corresponding to $(i_{cur,l}, j_{cur.l})$th texel (at 0th LOD $P_{min,l}$ equals to $P_{max,l}$).

The progress along the ray, accelerated by means of the mipmap $\{h_{max}, h_{min}\}$, is implemented in our developed algorithm *A2*.

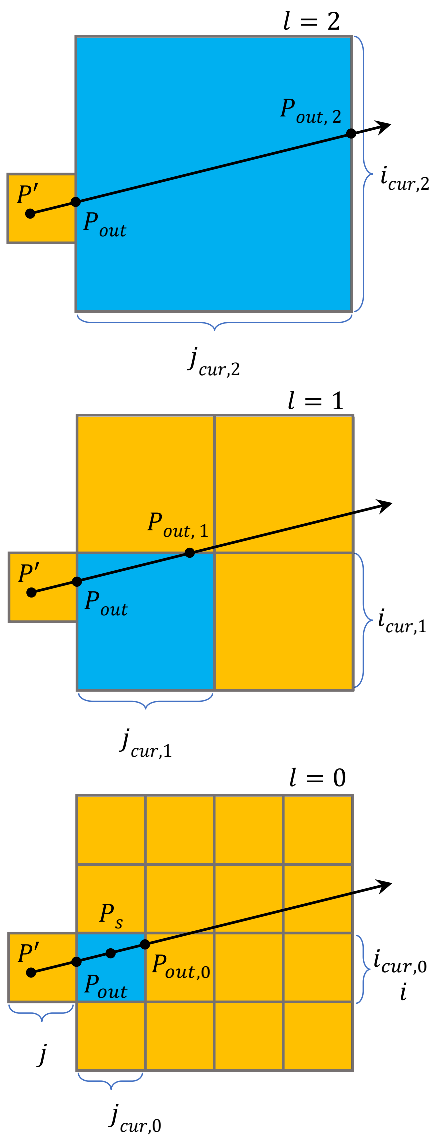


Figure 6: Progress along the ray at different LODs

Algorithm A2

```
1. Initialization.
      Calculate $P$ and $(i, j)$, as well as $k_{shadow}$ by equation (4).
      Set $P_{in} = P'$ in the algorithm A1.
      Calculate $P_{out}$ and $(i_{next,0}, j_{next,0})$ by means of algorithm A1.
      Set $P_{in} = P_{out}$ in the algorithm A1. Set $l = 2$, $k = 1$.
2. Ray-casting loop.
   Do loop while at least one of $(i_{cur,0}, j_{cur.0})$ will go beyond height map dimensions:
      Calculate $(i_{cur,l}, j_{cur,l})$ from $(i_{next,0}, j_{next,0})$.
      Calculate $P_{out,l}$ and $(i_{next,l}, j_{next,l})$ by means of algorithm A1.
      If $l > 0$, then do minimum height test:
            Calculate $P_{min,l}$ based on $P_{out,\,l}$ and $h_{min}$ sampled from $(i_{cur,l}, j_{cur,l})$th texel.
            Calculate for the ray $\boldsymbol{v} = (P_{min,l} - P)$ the flag $b_{shadow}$ by equations (2) and (3);
            If $b_{shadow}$ equals 1, then $k = k_{shadow}$, exit the loop.
      Do maximum height test:
            If $l$ equals to 0, then:
                  Calculate $P_{max,0}$ based on $P_s$ (see Fig. 4) and $h$ sampled from $(i_{cur,0}, j_{cur,0})$th texel.
            Otherwise:
                  Calculate $P_{max,l}$ based on $P_{out,\,l}$ and $h_{max}$ sampled from $(i_{cur,l}, j_{cur,l})$th texel.
                  Calculate for the ray $\boldsymbol{v} = (P_{max,l} - P)$ the flag $b_{shadow}$ by equations (2) and (3);
            If $b_{shadow}$ equals to 1, then:
                  If $l$ equals to 0, then $k = k_{shadow}$, exit the loop.
                  Otherwise: $l = l$-1.
            Otherwise:
                  Set $(i_{cur,l}, j_{cur,l}) = (i_{next,l}, j_{next,l})$. Calculate $(i_{cur,0}, j_{cur.0})$ from $P_{out,l}$.
                  Set $P_{in} = P_{out,l}$ in the algorithm A1. Set $l = 2$.
3. Set pixel color: $C = kC$.
End of the algorithm.
```

5. Results

Based on methods and algorithms proposed, a software complex for real-time rendering dynamic relief shadows was developed. The complex is a part of the VirSim virtual environment system [28], developed at the Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences".

We carried out a number of comparative shadow visualization tests using common cascaded shadow mapping method (CSM) varied in configurations, and our shadow ray casting implementation. For this, we integrated our solution into well-known NVidia CSM implementation, available at [5]. To stress-test both methods, we created two tricky height maps: "spikes" with 5 elongated thin cones (see Figure 7) and "steps", which yields a 3D model with non-closed surface (see Figure 8). Moreover, multiple tests were carried out using classic Grand Canyon height map of size 2K×2K (see Figure 9) and Puget Sound height map of sizes 1K×1K, 2K×2K and 4K×4K (see Figure 10), available at [24].

Figure 7 shows examples of cases where "spike"-test reveals hidden CSM method drawbacks: (a-c) some shadows are torn off from spikes or are passed through them; (e-g) sawtooth shadow edges are appeared (when increasing CSM resolution, edge appearance looks better, but sawtoothness remains still noticeable); (i) conical shape of spike shadow is lost (even at 4K CSM). Our ray casting implementation produces accurate spike shadows with smooth edges in all the above cases (d, h, j).

Figure 8 illustrates results of reliability test where shadows of steps model, having front (visible) faces only from one side, are visualized. Due to enabled back face culling optimization, CSM method failed to produce some of the shadows (left image). In our implementation shadows are produced completely (right image), because object's height map is directly used, and there is no dependency on 3D model construction and visualization methods.

As tests on the Grand Canyon and the Puget Sound height maps showed, dynamic shadow loss is possible in CSM method at long (see Figure 9a) or middle (see Figure 10a, along the lake) distances due to insufficiently close/distant location of intermediate clipping planes. For the same reason, a sawtooth silhouette of elongated mountain shadows may appear (see Figure 10c). In our ray casting implementation all relief shadows are accurately drawn with high detailing, regardless of distances they are located at and sun altitude (see Figures 9b, 10b, 10d).

All tests described were performed at 1920×1080 screen resolution on personal computer (Intel Core i7-6800K 3.40GHz, 16Gb RAM, Windows 10 Pro) equipped with graphics card NVidia GeForce RTX 2080 (2944 cores, 8 GB VRAM, 16x anisotropic filtering, 8x anti-aliasing). Table 1 contains results of performance measurement of the Puget Sound shadow rendering for both methods.

Table 1
Performance (in frames per second) of CSM with 4 splits/4K (first), and our ray-casting implementation (second) for different resolutions of Puget Sound height map and different sun altitudes

	82°	41°	20°	10°	3°
1024^2	486; 98	463; 52	471; 40	584; 32	591; 25
2048^2	182; 34	178; 18	182; 16	185; 12	182, 10
4096^2	83; 22	76; 10	80; 8	81; 6	80; 6

As it is seen from Table 1, in considered sun altitude range our solution can provide real-time frame rates for height maps up to 1024^2. To overcome this, in the future we plan to conduct an extended research using modern NVidia RTX technology of hardware-software ray tracing acceleration.

6. Acknowledgements

The publication is made within the state task of Federal State Institution “Scientific Research Institute for System Analysis of the Russian Academy of Sciences” on “Carrying out basic scientific researches (47 GP)” on topic No. FNEF-2021-0012 “Virtual environment systems: technologies, methods and algorithms of mathematical modeling and visualization. 0580-2021-0012” (Reg. No. 121031300061-2).

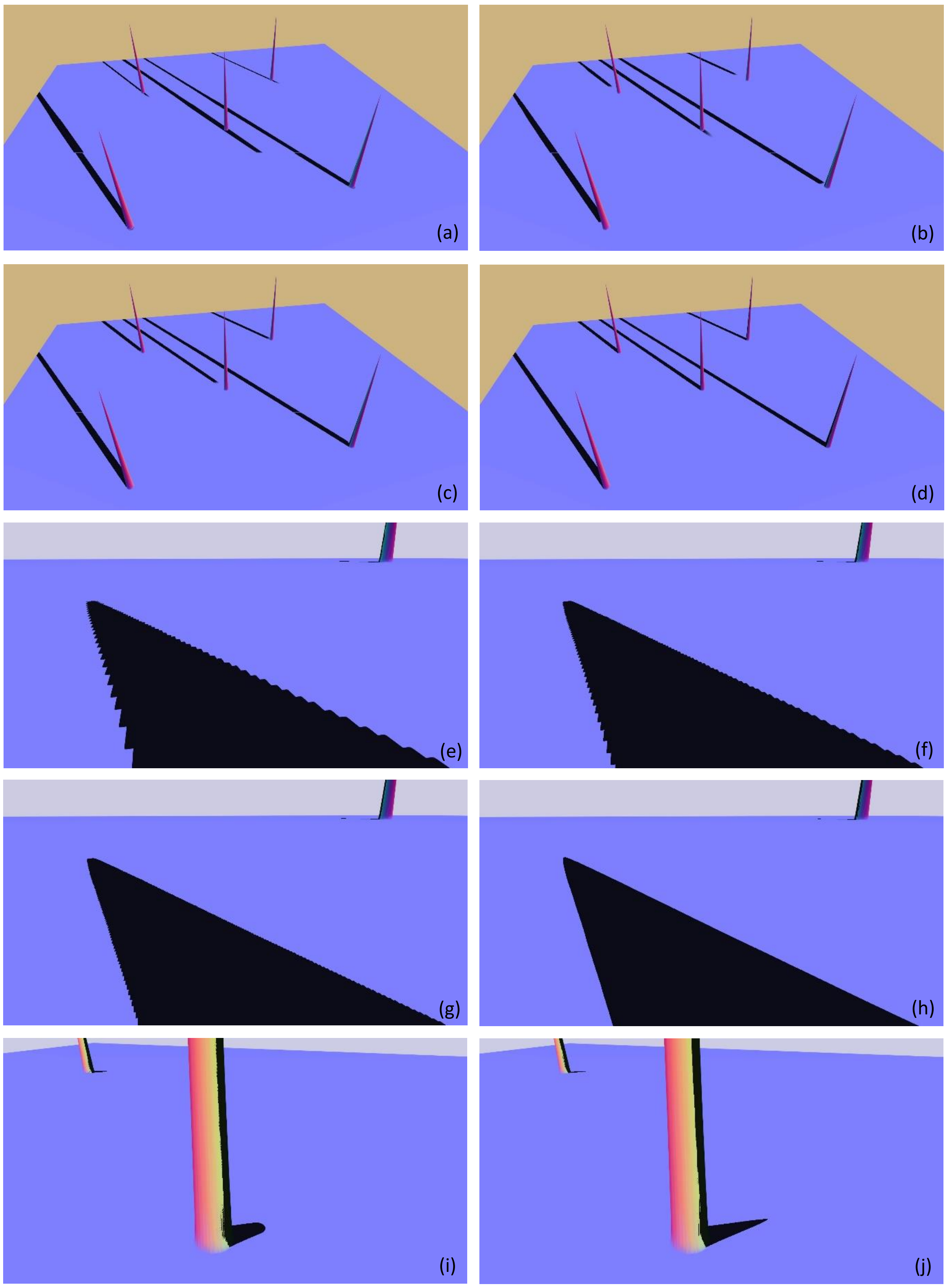


Figure 7: Testing the quality of shadow rendering methods on "spikes" model using cascaded shadow maps [5] with 4 splits and map resolution of 1K (a, e), 2K (b, f) and 4K (c, g, i), and using our ray casting implementation (d, h, j)

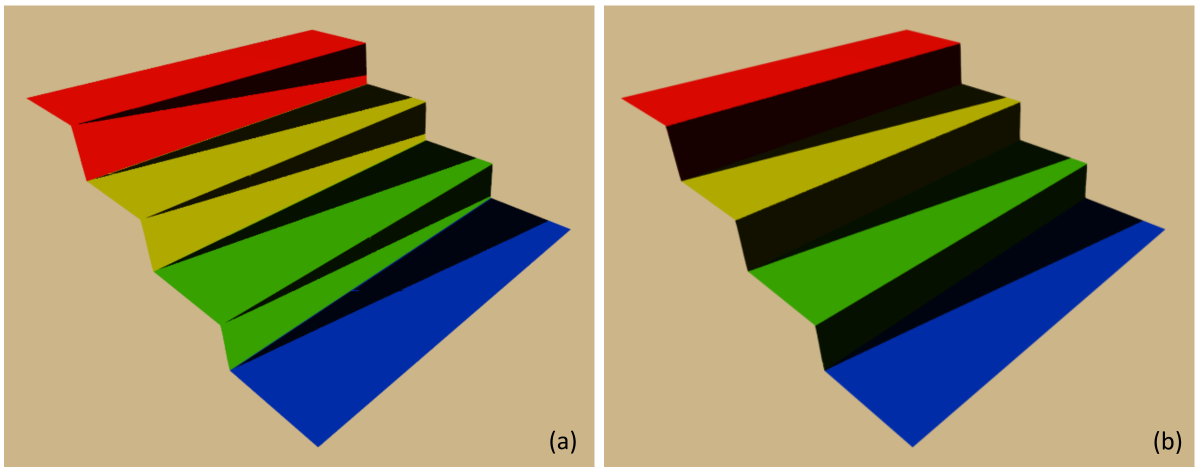


Figure 8: Testing the reliability of shadow rendering methods on "stairs" model with non-closed surface: (a) cascaded shadow maps (4 splits, 4K), (b) our ray casting implementation

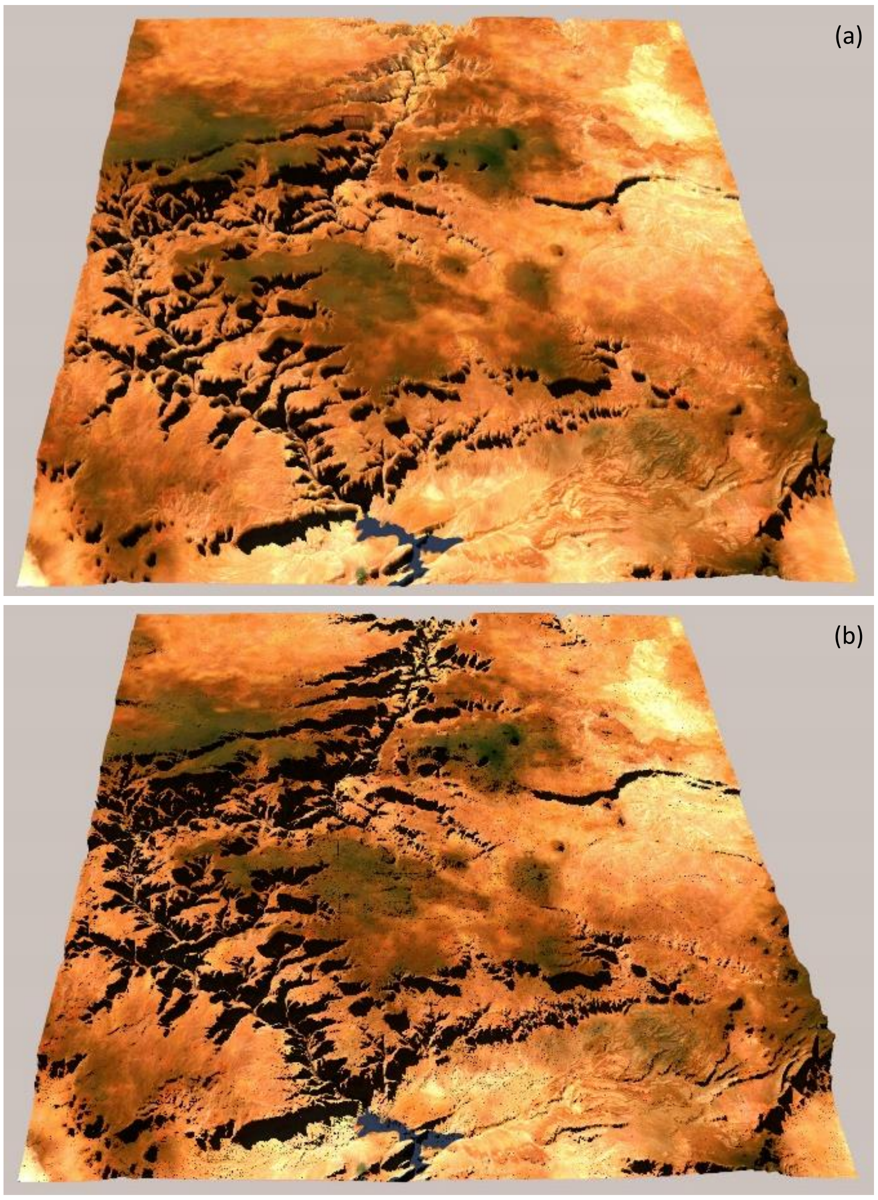


Figure 9: Shadows rendered for Grand Canyon height map [24] using (a) cascaded shadow maps (4 splits, 4K) and (b) our ray casting implementation

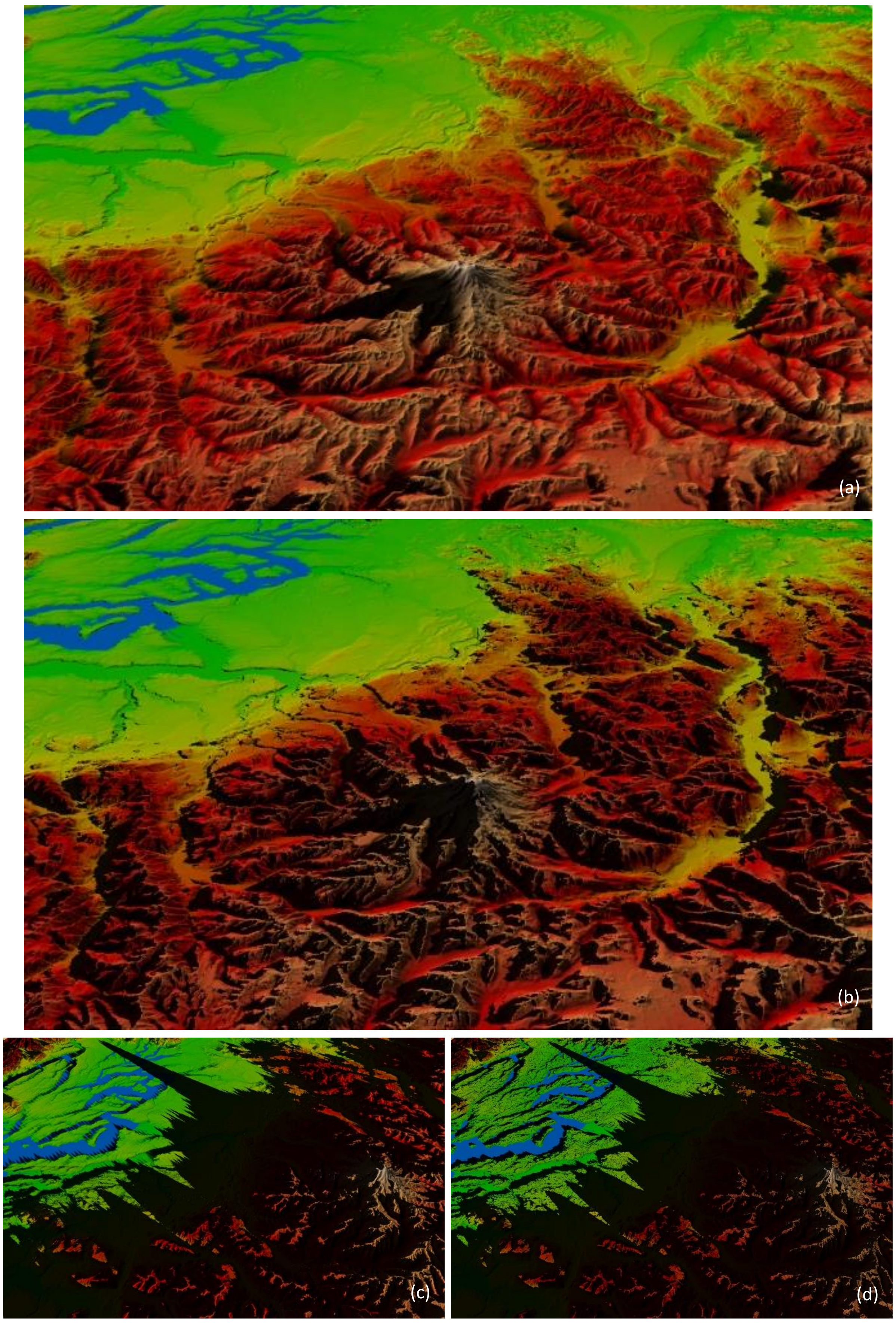


Figure 10: Shadow rendered for Puget Sound height map [24] using (a, c) cascaded shadow maps (4 splits, 4K) and (b, d) our ray casting implementation

7. References

[1] V.A. Frolov, A.G. Voloboy, S.V. Ershov, V.A. Galaktionov, The current state of the methods for calculating global illumination in tasks of realistic computer graphics, Trudy ISP RAN/Proc. ISP RAS 33.2 (2021): 7-48 (in Russian). doi:10.15514/ISPRAS–2021–33(2)–1.

[2] V.V. Sanzharov, V.A. Frolov, V.A. Galaktionov, Survey of Nvidia RTX Technology, Programming and Computer Software 46.4 (2020): 297-304. doi:10.1134/S0361768820030068.

[3] M.V. Mikhaylyuk, P.Y. Timokhin, A.V. Maltsev, A Method of Earth Terrain Tessellation on the GPU for Space Simulators, Programming and Computer Software 43.4 (2017): 243-249. doi:10.1134/S0361768817040065.

[4] B.Kh. Barladian, A.G. Voloboy, L.Z. Shapiro, N.B. Deryabin, I.V. Valiev, S.V. Andreev, Yu.A. Solodelov, V.A. Galaktionov, Safety critical visualization of the flight instruments and the environment for pilot cockpit, Scientific Visualization 13.1 (2021): 124-137. doi:10.26583/sv.13.1.09.

[5] R. Dimitrov, Cascaded shadow maps, Developer Documentation, NVIDIA Corp., 2007. URL: http://developer.download.nvidia.com/SDK/10/opengl/samples.html#cascaded_shadow_maps.

[6] Z. Fan, H. Sun, L. Xu, K. L. Lee, Parallel-Split Shadow Maps for Large-Scale Virtual Environments, in: Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and its Applications (VRCIA ’06), Association for Computing Machinery, New York, NY, 2006, pp. 311–318. doi:10.1145/1128923.1128975.

[7] M. Bunnell, F. Pellacini, Shadow Map Antialiasing, in: R. Fernando (Ed.), GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics, 1st. ed., Addison-Wesley, 2004, pp. 185–192. URL: https://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch11.html.

[8] P. Rosen, Rectilinear texture warping for fast adaptive shadow mapping, in: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D 2012), 2012, pp. 151–158. doi:10.1145/2159616.2159641.

[9] A.E. Lefohn, S. Sengupta, J.D. Owens, Resolution-Matched Shadow Maps, ACM Transactions on Graphics 26.4 (2007): article 20. doi:10.1145/1289603.1289611.

[10] G.S. Johnson, W.R. Mark, C.A. Burns, The Irregular Z-Buffer and its Application to Shadow Mapping, Department of Computer Sciences and Texas Advanced Computing Center of the University of Texas at Austin, 2004. URL: https://www.cs.utexas.edu/ftp/techreports/tr04-09.pdf.

[11] V. Forest, L. Barthe, M. Paulin, Realistic soft shadows by penumbra-wedges blending, in: Proceedings of the 21st ACM Siggraph/Eurographics symposium on Graphics hardware (GH ’06), 2006, pp. 39–46. doi:10.1145/1283900.1283907.

[12] M. Stich, C. Wächter, A. Keller, Efficient and Robust Shadow Volumes Using Hierarchical Occlusion Culling and Geometry Shaders, in: H. Nguyen (Ed.), GPU Gems 3, Addison-Wesley Professional, 2007, pp. 239–256. URL: https://developer.nvidia.com/gpugems/gpugems3/part-ii-light-and-shadows/chapter-11-efficient-and-robust-shadow-volumes-using.

[13] Z. Fu, H. Zhang, R. Wang, Z. Li, P. Yang, B. Sheng, L. Mao, Dynamic Shadow Rendering with Shadow Volume Optimization, in: N. Magnenat-Thalmann et al. (Eds.), Advances in Computer Graphics, Proceedings of the 37th Computer Graphics International Conference, CGI 2020, LNCS, vol. 12221, Springer Nature Switzerland, 2020, pp. 96–106. doi:10.1007/978-3-030-61864-3_9.

[14] Z. Brawley, N. Tatarchuk, Parallax Occlusion Mapping: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing, in: W. Engel (Ed.), ShaderX3: Advanced Rendering with DirectX and OpenGL, 1st. ed., Charles River Media, 2004, pp. 135–154.

[15] T. Sakalauskas, Hybrid Terrain Shadow Ray Casting, in: S. Cunningham, V. Skala (Eds.), Proceedings of the 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG ’2008), University of West Bohemia, Plzen, 2008, pp. 175–182. URL: http://wscg.zcu.cz/wscg2008/Papers_2008/short/!_WSCG2008_Short_final.zip.

[16] J. Amanatides, A. Woo, A Fast Voxel Traversal Algorithm for Ray Tracing, in: Proceedings of the 8th European Computer Graphics Conference and Exhibition (Eurographics ’87), Amsterdam, 1987, pp. 3–10. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.3443.

[17] T. Aslandere, M. Flatken, A. Gerndt, A Real-Time Physically Based Algorithm for Hard Shadows on Dynamic Height-Fields, in: Proceedings of 12. Workshop der GI-Fachgruppe on Virtuelle und Erweiterte Realität, Aachen Verlag, Bonn, 2015, pp. 101–112. URL: https://elib.dlr.de/101497/.

[18] F. Policarpo, M.M. Oliveira, Relaxed Cone Stepping for Relief Mapping, in: H. Nguyen (Ed.), GPU Gems 3, Addison-Wesley Professional, 2007, pp. 409–428. URL: https://developer.nvidia.com/ gpugems/gpugems3/part-iii-rendering/chapter-18-relaxed-cone-stepping-relief-mapping.

[19] A. Tevs, I. Ihrke, H.-P. Seidel, Maximum Mipmaps for Fast, Accurate, and Scalable Dynamic Height Field Rendering, in: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games (I3D ’08), New York, NY, 2008, pp. 183–190. doi:10.1145/1342250.1342279.

[20] D. Jung, F. Schrempp, S. Son, Optimally Fast Soft Shadows on Curved Terrain with Dynamic Programming and Maximum Mipmaps, 2020. URL: https://arxiv.org/pdf/2005.06671.pdf.

[21] M.V. Mikhaylyuk, D.A. Kononov, D.M. Loginov, The method for calculating the set of cells in 2D square grid intersected by a given ray, Trudy NIISI RAN/Proc. SRISA RAS 9.1 (2019): 44-48 (in Russian). doi:10.25682/NIISI.2019.1.0006.

[22] P.Yu. Timokhin, K.D. Panteley, E.M. Vozhegov, A.M. Trushin, Constructing on the GPU maximum intensity projection of 3D scalar fields of digital core material model, Trudy NIISI RAN/Proc. SRISA RAS 9.3 (2019): 58-65 (in Russian). doi:10.25682/NIISI.2019.3.0008.

[23] M. Segal, K. Akeley, The OpenGL Graphics System: A Specification, Version 4.6, Core Profile, The Khronos Group Inc., 2006-2019. URL: https://www.khronos.org/registry/OpenGL/specs/gl/ glspec46.core.pdf.

[24] Large Geometric Models Archive (Puget Sound, Grand Canyon), Georgia Institute of Technology, URL: https://www.cc.gatech.edu/projects/large_models/.

[25] H. Lass, Vector and Tensor Analysis, International series in pure and applied mathematics, 1st. ed., McGraw-Hill Book Company, Inc., New York, NY, 1950.

[26] M.V. Mikhaylyuk, P.Y. Timokhin, M.A. Torgashev, The Method of Real-Time Implementation of Tone Mapping and Bloom Effect, Programming and Computer Software 41.5 (2015): 289-294. doi:10.1134/S0361768815050084.

[27] E. Lengyel, Mathematics for 3D Game Programming and Computer Graphics, 3rd. ed., Course Technology, Boston, MA, 2012.

[28] M.V. Mikhaylyuk, A.V. Maltsev, P.Y. Timokhin, E.V. Strashnov, B.I. Kryuchkov, V.M. Usov, The VirSim Virtual Environment System for the Simulation Complexes of Cosmonaut Training, Scientific Journal Manned Spaceflight 4 (2020): 72-95 (in Russian). doi:10.34131/MSF.20.4.72-95. URL: http://www.gctc.ru/media/files/Periodicheskie_izdaniya/ppk_2020_4_total_37/5_stat.a_mihailuk_pq.pdf