Souhrnná zpráva za rok 2013 z projektu "Vývoj softwaru v oblasti CAD systémů, 3D grafiky a vizualizace grafických scén"

Jan Pečiva

Faculty of Information Technology, Brno University of Technology

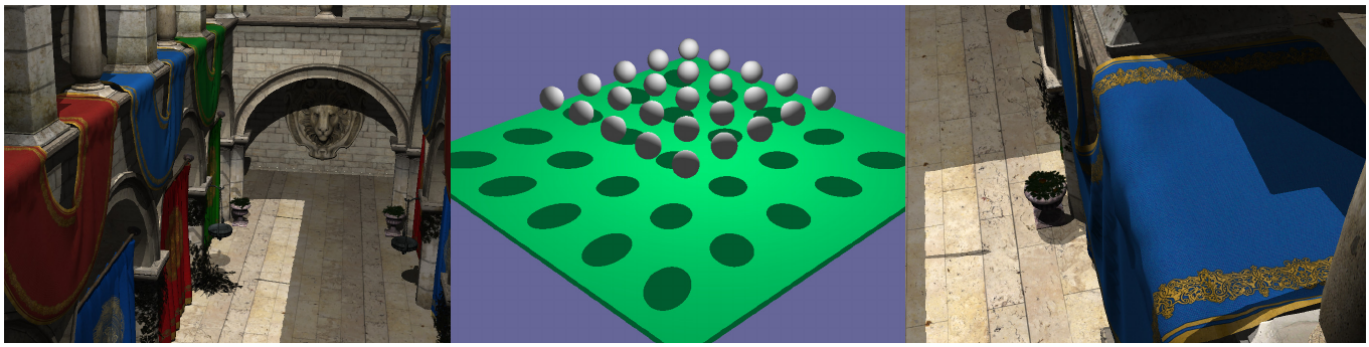


Figure 1: *Precise shadows shown in Sponza and sphere scenes. These scenes were used for tests of our proposed algorithms.*

Abstract

Silhouette-based shadow volume rendering provides per-sample precise shadow visualization unlike other approaches, such as shadow maps. In modern hardware, performance of the silhouette-based methods is even comparable to those of shadow maps. Traditionally, the shadow volume methods require the scene objects to be closed watertight 2-manifolds to achieve good performance. Such restriction is not acceptable for many applications, such as CAD, architecture rendering or other technical applications. The proposed algorithm works with arbitrary triangle meshes. We implemented the algorithm on number of platforms, tested it on variety of graphics hardware and evaluated which implementation approach is suitable under various circumstances. We also propose the current hardware improvements to allow for further optimizations.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture/Image Generation—Display Algorithms Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Shadows provide an important visual cue in the virtual scene visualizations. Shadow volumes [Cro77] and shadow mapping [Wil78] are two major approaches for shadow rendering used today. A widespread idea exists, that shadow mapping is superior in performance over shadow volumes. We are going to show that this is not true nowadays and that our shadow volume-based algorithms implemented on the contemporary graphics hardware are capable of outperforming shadow mapping or at least of delivering similar performace while being more precise and artefact free.

The most important property of shadow volumes is their capability to deliver per-sample correct visual results while shadow maps are prone to number of visual artifacts. Per-sample correctness is nowadays often required in CAD, technical and architecture visualizations where visual artifacts are often not acceptable. Most of the shadow mapping methods suffer from visual imperfections that might pop up in some scenes. In game industry, visual artifacts can be supressed by careful design of scenes. But again, this is usually not an option in some other areas. The correct scene visualization might be important in these areas while adapting the scene for the purpose of correct shadow visualization

might be undesirable. Thus, this paper presents the shadow volume algorithms that provide precise per-sample correct results while providing similar performance to the shadow mapping as shown in this paper by the performance measurements.

Shadow volumes were originally proposed by [Cro77]. [Hei91] implemented them on then-new stencil buffer hardware and his approach became known as *z-pass* method. Because z-pass approach was not completely robust, [Car00] and [BS99] proposed *z-fail* approach to overcome its limitations. Finally, [EK02] presented robust z-fail algorithm that should theoretically provide correct visual results for any arbitrary scene.

Shadow mapping [Wil78] was another method for the shadow computation. With the hardware support [Kil01] it started to become very popular method as it provided much better performance than shadow volumes at that time. However, shadow mapping techniques are prone to visual artifacts [Llo07]. Tremendous amount of research was done to eliminate these artifacts, for instance [SD02], [WSP04], [ZSXL06] and many others. However, the developed techniques often improve the visual result to some extent but does not avoid the sources of the artifacts themselves. Erik Sintorn introduced sample-correct method based on shadow mapping [SEA08]. As he stated, the method is three times slower than standard shadow mapping with resolution 8192x8192 for the screen resolution 512x512.

As shadow volumes are capable of providing artifact-free visual results, there were many attempts to optimize them, focusing especially to reduce rasterization work of large shadow volume extents. [Len02] and [MHE*03] tried to use attenuated lights and to use scissor test to clamp rendering to only necessary part of the screen. CC Shadow Volumes [LWGM04] reduce fill-rate by (1) culling shadow casters that are completely in shadow or whose shadows are not visible to the eye and (2) they clamp shadow volumes in a way that the places without shadow receivers does not consume fill-rate. Recently, [SOA11] attempted to implement shadow volumes hierarchical rasterization on CUDA, getting interesting performance results.

Many researchers exploited the fact that the shadow volume can be extruded using object's silhouette edges instead of extruding each triangle, usually greatly reducing rasterization work. [Ber86] described algorithm for finding silhouette edges on manifold meshes. [BS03] was the first to show the silhouette algorithm implemented entirely in graphics hardware. [MU04] described how to find silhouette and extrude it entirely in vertex shader using a specially precomputed mesh. [vW05] showed optimized silhouette construction using SSE2 instructions. [SWK07] implemented silhouette algorithm using then-new geometry shader.

Many silhouette algorithms require 2-manifold triangle meshes to work correctly. To be able to do the same with arbitrary model, more general algorithms were proposed. The one described in [AW04] requires mesh to be orientable only. [KKT08] showed algorithm that should work correctly with any arbitrary mesh. [PSM*13] discovered that the algorithm suffers from visual artifacts due to limited numerical precision of both GPU and CPU computing units while he proposed robust silhouette solution and verified it on several hardware platforms.

2. Proposed algorithm

Our proposed algorithm is based on silhouette shadow volumes. It constructs a silhouette edge set for each shadow casting geometry. The silhouette edge set is constructed in the similar way to the one in [KKT08]. For each edge, a plane is constructed using the edge and the light position (or light direction vector in the case of directional light). The plane splits the scene into two subspaces. We iterate through adjacent triangles of the edge and compute the difference in number of triangles between both subspaces. This number is called *multiplicity*. Its absolute value states how many times the edge should be extruded from the light source and its sign designates the vertex winding of the extruded quads. The quads will be called shadow volume (SV) sides in the paper.

Formally, the edge multiplicity computation is described in Algorithm 1. The first novel contribution of the paper is the visibility test performed at the very beginning of the algorithm in conjunction with exploitation of the deterministic multiplicity calculation.

Data: Edge $\bar{A},\bar{B}$, $\bar{A}<\bar{B}$, set $\mathfrak{O}$ of opposite vertices $\bar{O}_i \in \mathfrak{O}$, light position $\bar{L}$ in homogeneous coordinates
Result: Multiplicity m
$\bar{C} = (A_xL_w - L_x, A_yL_w - L_y, A_zL_w - L_z, 0)^T$;
$\bar{D} = (B_xL_w - L_x, B_yL_w - L_y, B_zL_w - L_z, 0)^T$;
if $!IsSideVisible(\bar{A},\bar{B},\bar{C},\bar{D},\bar{L})$ **then**
  return;
end
$m = 0$;
for $\bar{O}_i \in \mathfrak{O}$ **do**
  if $\bar{A} > \bar{O}_i$ **then**
    $m = m + CompMultiplicity(\bar{O}_i,\bar{A},\bar{B},\bar{L})$;
  else
    if $\bar{B} > \bar{O}_i$ **then**
      $m = m - CompMultiplicity(\bar{A},\bar{O}_i,\bar{B},\bar{L})$;
    else
      $m = m + CompMultiplicity(\bar{A},\bar{B},\bar{O}_i,\bar{L})$;
    end
  end
end

Algorithm 1: Algorithm for multiplicity computation of edge $\bar{A},\bar{B}$ and set of opposite vertices $\mathfrak{O}$.

As input, Algorithm 1 takes the edge with ordered start

and end points and the set of opposite vertices that, together with edge vertices, form the adjacent triangles.

The multiplicity contribution of particular adjacent triangle can be described as in Algorithm 2.

Data: Vertices $\bar{A},\bar{B},\bar{C}$, where $\bar{A}<\bar{B}<\bar{C}$ and light position $\bar{L}$
Result: Multiplicity m for one opposite vertex
$\bar{N}=(\bar{C}-\bar{A})\times(L_x-A_xL_w,L_y-A_yL_w,L_z-A_zL_w)$;
$m=sgn(\bar{N}\cdot(\bar{B}-\bar{A}))$;
Algorithm 2: Algorithm for multiplicity computation of reference edge $\bar{A},\bar{C}$ and one opposite vertex $\bar{B}$

The visibility test consists of the two parts. The first part tests all four edges of SV side against the view frustum. If any edge is visible, the side have to be rendered on the screen. Otherwise, we proceed to the second step. In the second step, we determine if the side is completely outside of the view frustum or whether it covers the whole screen.

Both parts of the visibility test are performed in clip space because of number of advantages. First of all, clip space coordinates are available in OpenGL pipeline before passing the geometry to the rasterizer, thus they are available without additional computation cost. As the second, visibility tests can be performed very easily in clip space as will be shown on the following paragraphs.

A point in clip space $\bar{P}=(P_x,P_y,P_z,P_w)$ is visible if simple conditions 1 are true.

$$\begin{aligned}\bar{P}_i &> -\bar{P}_w\\ \bar{P}_i &< \bar{P}_w\\ \pm\bar{P}_i+\bar{P}_w &> 0\\ i &\in \{x,y,z\}\end{aligned} \tag{1}$$

A point $\bar{X}(t),t\in\langle 0,1\rangle$ on a line segment $\bar{A},\bar{B}$ is given by the equation 2

$$\bar{X}(t)=\bar{A}+t\cdot(\bar{B}-\bar{A}) \tag{2}$$

A line segment in clip space is visible if conditions 3 are true. We can combine conditions 1 and equations 2 to obtain final conditions for a line segment:

$$\begin{aligned}\bar{X}(t)_i &> -\bar{X}(t)_w\\ \bar{X}(t)_i &< \bar{X}(t)_w\\ \pm\bar{X}(t)_i+\bar{X}(t)_w &> 0\\ i &\in \{x,y,z\}\end{aligned} \tag{3}$$

If the parameter $t\in\langle 0,1\rangle$ exists that makes all conditions 3 true, than the line segment is visible. Algorithm 3 shows interval determination of parameter t values. It starts with interval $\langle 0,1\rangle$ and moves its bounds for each condition 3. If minimal value of parameter t is greater than maximal value of parameter t, a line segment is not visible. This visibility test is performed on all four edges of a SV side. The visibility test is depicted in Figure 2.

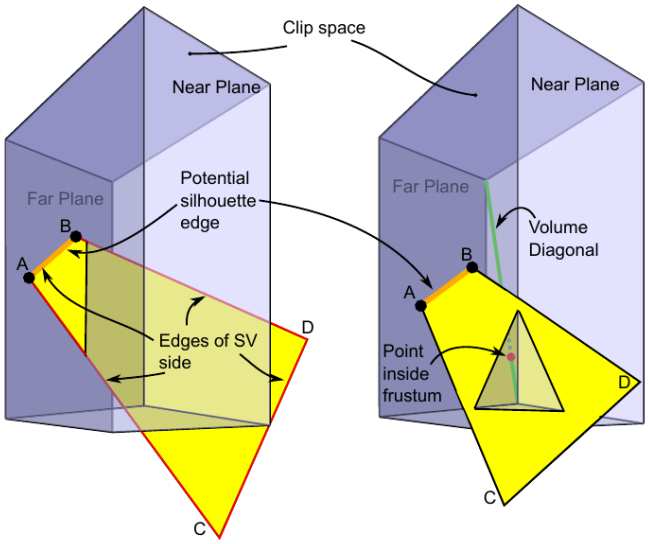


Figure 2: *The image shows the visibility test. The visibility test is performed in clip space. It is split into the two parts. The First part (left) tests visibility of four edges of a SV side. The second part (right) tests visibility of a SV side if all its edges are invisible.*

The second part of visibility test uses the whole side quad. None of side quads edges could be visible and still affect the screen. A side quads could cover the whole screen or some part of it, see right part of Figure 2. A point $\bar{X}(t,s),t,s\in\langle 0,1\rangle$ on side quad $\bar{A},\bar{B},\bar{C},\bar{D}$ in clip space is given by the equation 4

$$\begin{aligned}\bar{X}(t,s)&=\bar{A}+t\cdot(\bar{B}-\bar{A})+s\cdot((\bar{C}+t\cdot(\bar{D}-\bar{C}))-(\bar{A}+t\cdot(\bar{B}-\bar{A})))\\ \bar{X}(t,s)&=\bar{A}+t\cdot(\bar{B}-\bar{A})+s\cdot(\bar{C}-\bar{A})+t\cdot s\cdot((\bar{D}-\bar{C})-(\bar{B}-\bar{A}))\\ \bar{X}(t,s)&=\bar{A}+t\cdot(\bar{B}-\bar{A})+s\cdot(\bar{C}-\bar{A})\\ \bar{X}(t,s)&=\bar{E}+t\cdot\bar{F}+s\cdot\bar{G}\end{aligned} \tag{4}$$

If a quad is visible but its edges are not visible, there must exist intersection point of quad and one volume diagonal of view frustum than lies inside frustum, see right part of Figure 2. Points on these diagonals have special property - their components are equal or negative of others, see Figure 2. We can express this property by the equations 5:

$$\bar{X}(t,s)_x\cdot R_x=\bar{X}(t,s)_y\cdot R_y=\bar{X}(t,s)_z\cdot R_z \tag{5}$$

Vector $\bar{R}=(R_x,R_y,R_z)$ represents a diagonal, see Figure 2. There is a simple close form solution for the equations 5 than finds parameters t,s. The algorithm can be seen in Figure 5. In order to decrease the number of computations, only one

Data: Edge $\bar{A}, \bar{B}$ in clip space
Result: True if the Edge $\bar{A}, \bar{B}$ is visible
$\bar{M}_1 = (A_x + A_w, A_y + A_w, A_z + A_w)$;
$\bar{M}_2 = (-A_x + A_w, -A_y + A_w, -A_z + A_w)$;
$\bar{N}_1 = (B_x + B_w, B_y + B_w, B_z + B_w)$;
$\bar{N}_2 = (-B_x + B_w, -B_y + B_w, -B_z + B_w)$;
$\bar{V}_1 = \bar{N}_1 - \bar{M}_1$;
$\bar{V}_2 = \bar{N}_2 - \bar{M}_2$;
$Left = 0$;
$Right = 1$;
for $i \in \{x, y, z\}$ **do**
  for $j \in \langle 1, 2 \rangle$ **do**
    if $V_{j,i} == 0$ **then**
      if $M_{j,i} < 0$ **then**
        return $false$;
      end
    else
      if $V_{j,i} > 0$ **then**
        $Left = max(Left, \frac{-M_{j,i}}{V_{j,i}})$;
      else
        $Right = min(Right, \frac{-M_{j,i}}{V_{j,i}})$;
      end
    end
  end
end
return $Left \leq Right$;

Algorithm 3: Algorithm *isEdgeVisible* which determines edge's position against view frustum

diagonal has to be tested. We choose the diagonal and vector $\bar{R}$ by using normal vector of SV sides. Algorithm 4 finds a diagonal id than can be transform to vector $\bar{R}$.

Data: Points of edge $\bar{A}, \bar{B}$, light position $\bar{L}$ in homogeneous coordinates, model view projection matrix $\mathbf{M}$
Result: Diagonal Id
$\bar{u} = (B_x - A_x, B_y - A_y, B_z - A_z)^T$;
$\bar{v} = (L_x - A_x, L_y - A_y, L_z - A_z)^T$;
$\bar{n} = \bar{u} \times \bar{v}$;
$\bar{s} = \mathbf{M} \cdot (n_x, n_y, n_z, 0)^T$;
$\bar{U} = \lfloor (\bar{1} + \text{sgn}((s_x, s_y, s_z)))/\bar{2} \rfloor$;
if $U_z == 1$ **then**
  $\bar{U} = \bar{1} - \bar{U}$;
end
return $U_x + U_y \cdot 2 - 1$;

Algorithm 4: Algorithm *getDiagonalId* for computation of id of volume diagonal.

The final visibility test can be seen in Algorithm 6.

Next proposed improvement is reduction of number of render-passes and reusing computation of multiplicity.

Data: Edge $\bar{\mathbf{A}}, \bar{\mathbf{B}}$, light position $\mathbf{L}$ in homogeneous coordinates, model view projection matrix $\mathbf{M}$
Result: True if the quad is visible
$\bar{a} = A_{xyz}$;
$\bar{b} = B_{xyz}$;
$\bar{c} = C_{xyz}$;
if $Diag \geq 0$ **then**
  $a[Diag] = -a[Diag]$;
  $b[Diag] = -b[Diag]$;
  $c[Diag] = -c[Diag]$;
end
$\bar{m} = (a_x - a_y)$;
$\bar{n} = (b_x - b_y)$;
$\bar{o} = (c_x - c_y)$;
$\bar{p} = (a_x - a_z)$;
$\bar{q} = (b_x - b_z)$;
$\bar{r} = (c_x - c_z)$;
$\bar{d} = (q \cdot o - n \cdot r)$;
$\bar{t} = \frac{(m \cdot r - p \cdot o)}{d}$;
$\bar{l} = \frac{-(m \cdot q - p \cdot n)}{d}$;
$X = A + t \cdot B + l \cdot C$;

Algorithm 5: Algorithm *isFullVisible* which determ quad's position against view frustum

Data: Points of quad $\bar{A}, \bar{B}, \bar{C}, \bar{D}$, light position $\bar{L}$ in homogeneous coordinates, model view project matrix $\mathbf{M}$
Result: True if the side $\bar{A}, \bar{B}, \bar{C}, \bar{D}$ is visible
$m = 0$;
$\bar{A}_c = \mathbf{M} \cdot \bar{A}$;
$\bar{B}_c = \mathbf{M} \cdot \bar{B}$;
$\bar{C}_c = \mathbf{M} \cdot \bar{C}$;
$\bar{D}_c = \mathbf{M} \cdot \bar{D}$;
$Diagonal = getDiagonalId(\bar{A}, \bar{B}, \bar{L}, \mathbf{M})$;
if $isFullVisible(\bar{A}_c, \bar{B}_c - \bar{A}_c, \bar{C}_c - \bar{A}_c, Diagonal)$ **ther**
  return $true$;
end
$R1 = isEdgeVisible(\bar{A}_c, \bar{B}_c)$;
$R2 = isEdgeVisible(\bar{A}_c, \bar{C}_c)$;
$R3 = isEdgeVisible(\bar{B}_c, \bar{D}_c)$;
$R4 = isEdgeVisible(\bar{C}_c, \bar{D}_c)$;
return $R1 \vee R2 \vee R3 \vee R4$;

Algorithm 6: Algorithm *isQuadVisible* for determina whether side quad is visible or not.

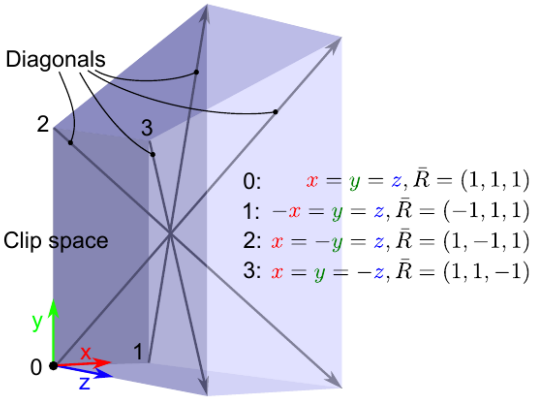


Figure 3: *The image shows volume diagonals of frustum in clip space and their ids. Each diagonal has its equation.*

Sides and caps are usually rendered separately. We propose method based on geometry shader that renders sides and caps together. This approach reduces number of passes and computation of multiplicity can be reused for caps. We extend a set of opposite vertices for each edge with flags. A opposite vertex with set flag forms front and back caps. Front caps can introduce self-shadowing artefacts on some GPUs. The winding of caps (and sides) has to be sometimes changed in orderer to guarantee the same orientation of every shadow volume in scene. Self-shadowing artefacts are caused by different rasterization of two identical triangles with different winding.

We propose method that shifts front cap to to infinity. The shifting can be seen on figure 2. The shifting ensures that the front cap always fails the depth test. Let $A = (a_x, b_y, c_z, d_w)$ is vertex of front cap in clip space. The shifted vertex B toward infinity from camera can be computed according to equation 6

$$B = (b_x, b_y, b_z, b_w) = (a_x, a_y, a_w, a_w) \tag{6}$$

3. Implementation Details and Issues

We have implemented our algorithm on various platforms. Vertex shader implementation brought our algorithm to WebGL and mobile platforms that supports vertex and fragment shaders only at the moment. Other platfoms uses mainly the GPU: geometry shaders, tessellation shaders, compute shaders, OpenCL. We have also tested CPU.

A potential disadvantage of our approach is the need for geometry pre-processing because for each edge, opposite vertices need to be identified and respective information collected and stored in VBOs. However, the preprocess is only made once at the application start up. This also leads to increased memory consumption. In an extreme cases such as popular Power Plant model consisting of 12.7M triangles, the model data might not fit into a conventional graphics

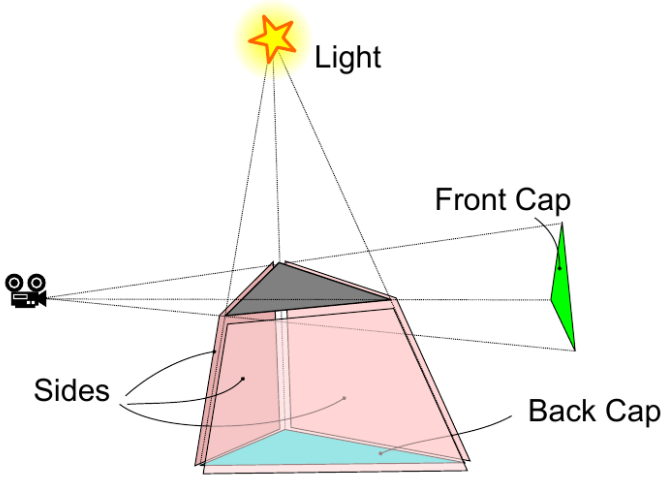


Figure 4: *The image shows the visibility test. The visibility test is done in clip space. It is split into two parts. The first part (left) tests visibility of four edges of a SV side. The second part (right) tests visibility of a SV side if every edge of it is invisible.*

card memory, because about 5GB might be required. Further optimization would be needed to address the issue. Per-triangle methods, on the other hand, don't require preprocessing work, since they don't need any information about neighbouring triangles.

We also looked into Intel's integrated graphics segment, where OpenGL support is not as good as on dedicated cards. There were issues with our tessellation method, whose GLSL program compiled successfully but linking failed on Intel HD 4600 despite the fact that a simple tessellation program worked and the chip and drivers support OpenGL 4.2 and several extensions from 4.3 and 4.4. The problem was reported to the manufacturer.

4. Experiments

Graphic processors of various manufacturers differ in their architectures. Even if focusing on particular manufacturer, the architecture evolve over the time in graphics processor generations. The different generations have different performance characteristics and have different feature set. As a result, particular implementation of an algorithm might be the most efficient on one graphics processor while other graphics processors might match better with other algorithm implementations.

In order to properly evaluate performance of our novel optimizations for silhouette shadow volumes, we developed number of implementations utilizing various parts of OpenGL graphics pipeline, starting from geometry shaders available widely on all Shader Model 4.0 capable graphics cards and finishing with compute shaders of Shader

Model 5.0 and OpenCL implementation. Moreover, we implemented reference cpu implementation and special implementation for vertex shaders based on ideas of [MU04]. Vertex shader implementation might seem a step back, but it opens the possibility to use our algorithms on OpenGL ES 2.0 and 3.0 capable mobile platforms.

We made extensive tests of our novel optimizations using number of our implementations. We selected high-end cards of various gpu generations of AMD and Nvidia and integrated solutions found in Intel microprocessors. We excluded too recent graphics cards and limited ourselves to graphics processors since introduction of unified shader pipeline.

For testing, we have chosen three scenes: fly-though Crytek Sponza (400k triangles) as a game-like scene, sphere scene with increasing number of triangles and sphere scene with one million triangles but divided to various number of separate scene objects. The last test stresses the fragmentation level of the scene, causing increased number of VBOs to be allocated, more draw calls and high number of small chuncks of data scheduled for processing on graphics card.

Test setup: Intel Xeon E3-1230V3, 16GiB DDR3, Windows 7 x64, driver version: 14.3 Beta (AMD, HD 5000 and newer), 13.9 (AMD HD 2900XT and HD 4890), 334.89 (nVidia). For integrated Intel GPUs, we used the same configuration as above except the processor Core i7 4770K for HD Graphics 4600 and Core i7 2700K for HD Graphics 3000 using 10.18.10.3496 drivers.

Our test suite consists of Lexolights open-source multiplatform application, based on OpenSceneGraph and Delta3D libraries and a light-weight testing framework directly utilizing OpenGL. All tests were carried out in FullHD (1920x1080) resolution in fullscreen mode. Lexolights, including our new optimizations, is available as open-source at http://sourceforge.net/projects/lexolights.

Our algorithm, as described above, generates and draws as many quads per edge as is the *multiplicity* of the edge. This way, stencil value is incremented/decremented *multiplicity*-times. Graphic processors from AMD offer an OpenGL extensions which can be used to speed up shadow volume rendering by more sophisticated stencil operations. One of such extensions is GL_AMD_stencil_operation_extended, which allows arbitrary stencil value addition, value replacement and binary operations. Using this extension, we do not need to render SV sides *multiplicity*-times. Instead, we separate these quads into a set of VBOs and each is then rendered with different stencil increment. This greatly reduces fillrate. The second extension, GL_AMD_shader_stencil_value_export, allows writing arbitrary value to stencil buffer in fragment shader. Compared to GL_AMD_stencil_operation_extended, there is no need to separate quads into separate buffers, stencil value addition is determined in shaders.

Triangles	CPU	PCPU	VS	GS	TS	CS	OCL	SM4k	SM8k	CTR	GSTR	TSTR	GS2M
102000	77.9	69	638	501	679	368	145	867	296	13.5	111	168	982
360000	25.9	21.8	203	230	216	214	101	721	274	4.1	31.7	68.5	560
640000	13.3	13.1	120	144	135	162	79	530	244	2.2	18.1	37.8	393
1000000	7.8	7.8	75.5	96.4	83.5	121	53.9	384	206	2	10.2	22.7	259
1400000	6.5	6.8	65.6	91.5	69	128	60.2	353	198	2	7.5	15.9	267
1960000	4.9	4.9	11.5	59.6	50.7	81	39.2	269	169	2	5.5	10.8	167

Table 1: *Dependence of performance (FPS) on number of triangles on R290x for different methods for sphere scene.*

Triangles	CPU	PCPU	VS	GS	TS	CS	OCL	SM4k	SM8k	CTR	GSTR	TSTR	GS2M
102000	88.1	75.2	299	401	454	292	117	544	295	14.6	102	151	666
360000	28.8	27.5	95.8	170	180	146	92.8	330	242	4.6	37.1	72.3	342
640000	16.9	16.5	54	109	118	90.5	74.9	257	206	2.6	22.7	42.1	236
1000000	10.2	9.9	33.9	70.1	81	58.4	58.7	201	172	2	13.7	24.8	163
1400000	7.8	7.9	12.7	64.9	71.2	54.2	56	176	149	2	11.2	19.7	143
1960000	5.9	5.8	5.8	43	48.9	38.2	40.1	149	120	2	7.6	12.8	99

Table 2: *Dependence of performance (FPS) on number of triangles on GTX780 for different methods for sphere scene.*

4.1. Results

In the first test, we compared our algorithms on two most current graphic cards (Radeon R9 290X, GeForce GTX 780) in all our methods and several popular test scenes.

bla bla, tabulka/graf

Secondly, we tested dependencies of increasing amount of geometry while maintaining constant number of objects in a scene on FPS, and also dependence of increasing number of scene objects on FPS while keeping triangle amount constant. We used synthetic Sphere scene for this purpose, because we can easily configure amount of objects/geometry in the scene.

Followingly, we selected popular Crytek Sponza scene, baked into 29 VBOs and without textures, and tested several GPUs on all our methods. Including classic shadow mapping with shadow map resolution of 4kx4k and 8kx8k. As can be seen in tables 5, 6, 8k shadow mapping was outperformed by non-CPU methods on AMD, taking the fastest method of each platform into account. Moreover, all our methods provided better shadow quality than 8k shadow mapping and also generated omnidirectional shadows from point light source, whereas shadow mapping was used only from one direction - in theory, it would be even slower in omnidirectional form and 4K shadow mapping would also lose it's performance advantage.

Strong point of two latest generations (otazka je ci generacie, kedze maju rovnaku architekturu) of nVidia chips, GTX 680 and 780, tak pouzij vyraz family (both Kepler

Objects	CPU	PCPU	VS	GS	TS	CS	OCL	SM4k	SM8k	CTR	GSTR	TSTR	GS2M
1	6	7	89.5	128	108	197	97.2	490	254	0.6	3	5.8	288
16	8.4	8.2	80.4	107	135	103	53.3	428	218	0.8	3	4.3	230
100	8.2	8	63.4	70	78.3	76.7	48.6	284	169	1.1	3.1	4	143
625	6.3	6.3	49.6	43.2	48.3	34.3	0	97.5	35.3	1.2	4	5	92.3
1250	4.4	5	12.4	14.1	40.1	15.8	22.8	105	36.2	1.2	4	4.9	83.4
2500	3.7	3.7	10.5	9.7	19.3	8.3	18	95.3	40	0.7	4.4	4.7	35.1
5100	2.9	2.6	9.1	7.3	15.6	4.9	11.7	71.6	31.7	0.7	5	5.2	21.4
15600	1.8	1.3	5.9	2.8	6.9	1.6	0	34.8	21.2	0.8	6.8	6.4	9.59

Table 3: *Dependence of performance (FPS) on number of objects on R290x for different methods for sphere scene.*

Objects	CPU	PCPU	VS	GS	TS	CS	OCL	SM4k	SM8k	CTR	GSTR	TSTR	GS2M
1	8.3	7.5	38.3	105	111	100	133	482	303	1.1	5.6	5.6	243
16	9.8	9.5	33.7	76.3	67.3	79.5	76.7	239	178	1.1	5.1	5.2	132
100	9.7	9	31.3	49.3	58.5	41.1	41.2	133	133	1.2	3.2	3.4	93.6
625	7.1	6.2	19.7	19.5	21.8	17.1	11.7	55.8	56.4	1.1	2.8	2.7	33.7
1250	6.1	5.1	16	15	18	12.7	7.3	40.2	39.7	1	2.1	2.2	24.2
2500	4.5	3.8	7.9	6.8	7.5	5	3.7	28.4	28	0.7	1.9	1.9	12.3
5100	4	3	8.4	6	8.1	4.1	2	21.2	20.9	0.6	2.2	2.2	10.6
15600	2.9	2	5.5	2.9	5.2	1.8	1.7	15.7	15.6	0.9	3.5	3.5	7.5

Table 4: *Dependence of performance (FPS) on number of objects on GTX780 for different methods for sphere scene.*

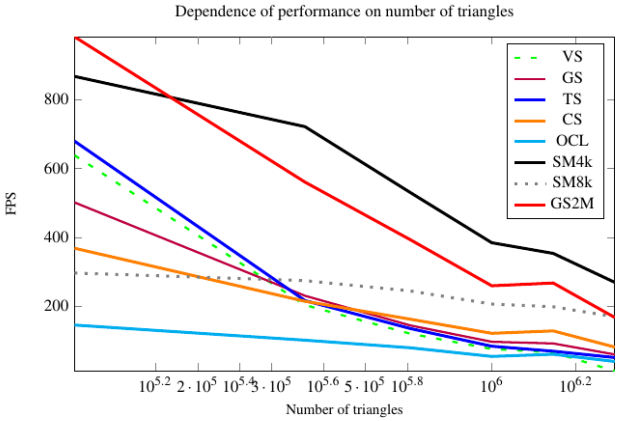


Figure 5: *Dependence of performance (FPS) on number of triangles on R290x for different methods for sphere scene.*

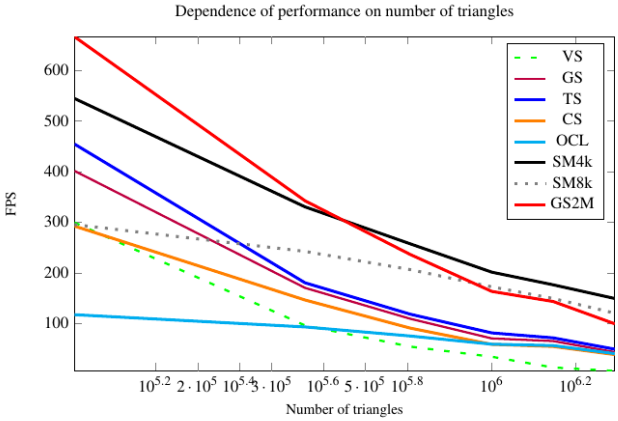


Figure 6: *Dependence of performance (FPS) on number of triangles on GTX780 for different methods for sphere scene.*

architecture), is tessellation, whereas for two latest AMD generations, R9 280X and 290X, compute shader is best suited, tightly backed by geometry shader implemetation. CPU implementations were generally slower when rendered by AMD cards. Contrary to that, on GeForce 8800Ultra, being the slowest from our nVidia lineup, the CPU implementation was the fastest.

Vertex shader implementation never provided the fastest solution on any GPU. It happened to be faster than 8k shadow mapping, but its targets are embedded GPUs sup-

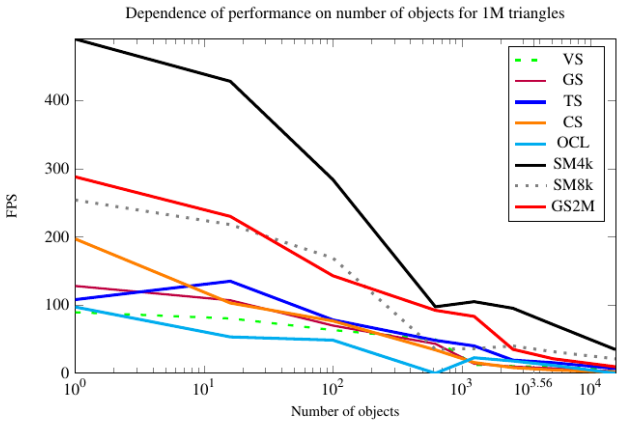


Figure 7: *Dependence of performance (FPS) on number of objects on R290x for different methods for sphere scene.*

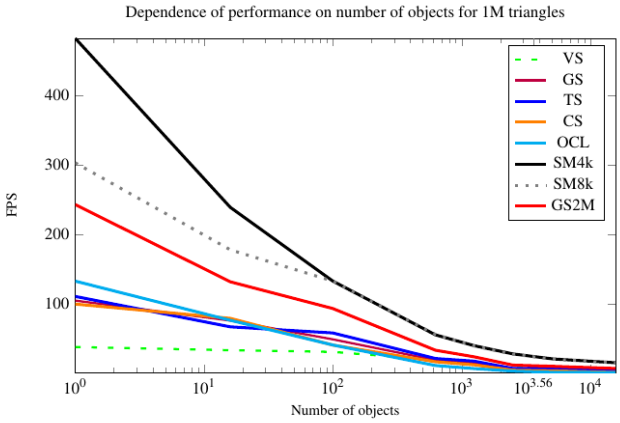


Figure 8: *Dependence of performance (FPS) on number of objects on G780 for different methods for sphere scene.*

porting OpenGL ES 2.0. This standard does not support geometry shader or other new pipeline stages, introduced in OpenGL 3 or newer.

Next, we tested two top GPUs in synthetic Sphere scene, where we first increased the amount of triangles per sphere while keeping their number constant (100 spheres, 10 by 10). Tables 1, 2 and graphs **??**, **??** contains results of this test. It can be seen that the best method to use on GeForce GTX 780 is tessellation, no matter the geometry amount. Compute shaders are practical choice when using latest AMD graphics hardware.

For comprehension, we measured also per-triangle methods. Tessellation-based per-triangle method provided the best results on both tested GPUs, while the CPU-based is basically the most naive algorithm of shadow volumes implementation, implemented for comparison only and not recommended for use in real applications.

Provided we have 2-manifold model, we can use faster

2-manifold implementation, which speeds up shadow volume rendering by XY% on Radeon and XY% in average, compared to the fastest respective shadow volume method per test case. We also tested classic shadow mapping, which keeps the same behaviour as in previous tests. Tables 3, 4 and graphs 7, 8 shows results from test with increasing number of objects in a scene. This measurement shows different results - tessellation is no longer domain of GeForce, but of Radeon. GTX780 does not have a dominant implementation in this test, but tessellation was still able to be the fastest in 37.5% of test cases. As mentioned in section 4, we also wanted to evaluate performance when using AMD's proprietary stencil extensions. Results of this test can be seen in tables 5 6 and graphs 9, 10. We tested AMD's extensions on Sponza scene.

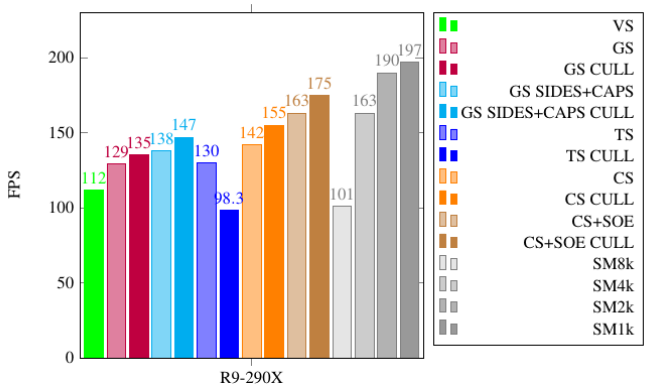


Figure 9: *Comparison of methods on Sponza scene with our new approach - visibility test. Darker columns depict methods that use visibility test. The measurements were carried out on R290x. Sides+caps method renders sides and caps together in one pass using our projection of front cap. Soe is extension GL_AMD_stencil_operation_extended that allows us to increment stencil buffer by greater numbers.*

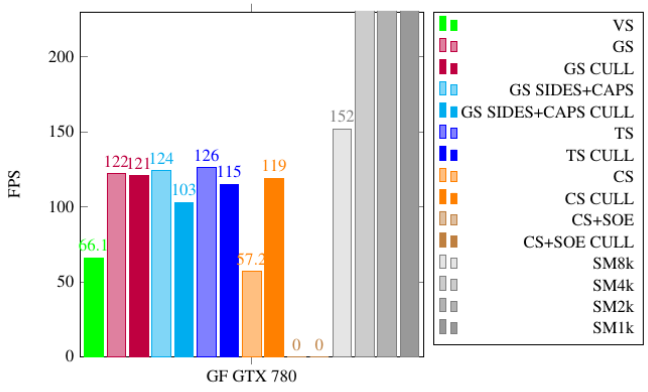


Figure 10: *Comparison of methods on Sponza scene with our new approach - visibility test. The measurements were carried out on GTX 780.*

GPU	VS	GS	GSa	GSb	GSab	GSc	GSac	GSbc	GSabc
GF GTX 780	66.1	122	121	n/s	n/s	124	103	n/s	n/s
R9-290X	112	129	135	37.9	37.8	138	147	36.4	36.8
R9-280X	67.0	108	110	27.6	27.5	114	119	27.3	27.6
GF GTX 680	42.4	90.9	87.4	n/s	n/s	n/a	71.5	n/s	n/s
GF GTX 580	48.3	89.4	93.5	n/s	n/s	91.7	82.8	n/s	n/s
GF GTX 480	42.4	78.0	81.1	n/s	n/s	n/a	n/a	n/s	n/s
R HD 5850	3.9	35.3	32.6	10.9	10.8	36.7	39.7	10.9	10.6

Table 5: *The performance of new approaches on Sponza scene. Letters a,b,c stands for visibility test, AMD's extension: GL_AMD_shader_stencil_value_export and rendering sides and caps together using shifting of front cap to infinity.*

GPU	TS	TSa	TSb	TSab	CS	CSa	CSd	CSAd	SM8k	SM4k	SM2k	SM1k
GF GTX 780	126	115	n/s	n/s	57.2	119	n/s	n/s	152	251	318	343
R9-290X	130	98.3	40.7	37.6	142	155	163	175	101	163	190	197
R9-280X	96.2	71.2	29.7	27.4	115	123	131	140	84	161	211	228
GF GTX 680	97.0	83.8	n/s	n/s	51.4	94.1	n/s	n/s	114	198	254	278
GF GTX 580	76.8	53.2	n/s	n/s	41.5	84.3	n/s	n/s	113	184	239	248
GF GTX 480	67.8	47.5	n/s	n/s	37.4	74.7	n/s	n/s	95.7	161	201	215
R HD 5850	20.2	8.1	9.4	6.3	33.1	36.5	37.7	39.5	22.7	52.4	78.4	89.6

Table 6: *The performance of new approaches on Sponza scene. Letters a,b,d stands for visibility test, AMD's extension: GL_AMD_shader_stencil_value_export and AMD's extension: GL_AMD_stencil_operation_extended.*

5. Conclusion

We presented a very fast algorithm for rendering hard shadows, impelmented in several pipeline stages and thoroughly tested on both modern and older hardware.

References

[AW04] ALDRIDGE G., WOODS E.: Robust, geometry-independent shadow volumes. In *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (New York, NY, USA, 2004), GRAPHITE '04, ACM, pp. 250–253. URL: http://doi.acm.org/10.1145/988834.988877, doi:10.1145/988834.988877. 2

[Ber86] BERGERON P.: A general version of crow's shadow volumes. *IEEE Computer Graphics and Applications 6* (1986), 17–28. 2

[BS99] BILODEAU B., SONGY M.: Real time shadows. *Creativity 1999* (1999). 2

[BS03] BRABEC S., SEIDEL H.-P.: Shadow volumes on programmable graphics hardware. *Computer Graphics Forum (Eurographics) 2003* (2003), 433–440. 2

[Car00] CARMACK J.: unpublished correspondence, 2000. 2

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1977), SIGGRAPH '77, ACM, pp. 242–248. URL: http://doi.acm.org/10.1145/563858.563901, doi:10.1145/563858.563901. 1, 2

[EK02] EVERITT C., KILGARD M. J.: Practical and robust stenciled shadow volumes for hardware-accelerated rendering. 2

[Hei91] HEIDMANN T.: Real shadow real time. IRIS Universe, pp. 28–31. 2

[Kil01] KILGARD M. J.: Shadow mapping with today's opengl hardware, 2001. URL: https://developer.nvidia.

com/sites/default/files/akamai/gamedev/docs/ShadowMaps_CEDEC_E.pdf. 2

[KKT08] KIM B., KIM K., TURK G.: A shadow-volume algorithm for opaque and transparent nonmanifold casters. *J. Graphics Tools 13*, 3 (2008), 1–14. 2

[Len02] LENGYEL E.: The mechanics of robust stencil shadows, 2002. URL: http://www.gamasutra.com/view/feature/2942/the_mechanics_of_robust_stencil_.php. 2

[Llo07] LLOYD B.: *Logarithmic Perspactive Shadow Maps*. 2007. URL: http://www.cs.unc.edu/~blloyd/dissertation/. 2

[LWGM04] LLOYD D. B., WENDT J., GOVINDARAJU N. K., MANOCHA D.: Cc shadow volumes. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2004), EGSR'04, Eurographics Association, pp. 197–205. URL: http://dx.doi.org/10.2312/EGWR/EGSR04/197-205, doi:10.2312/EGWR/EGSR04/197-205. 2

[MHE*03] MCGUIRE M., HUGHES J. F., EGAN K., KILGARD M., EVERITT C.: *Fast, Practical and Robust Shadows*. Tech. rep., NVIDIA Corporation, Austin, TX, Nov 2003. URL: http://developer.nvidia.com/object/fast_shadow_volumes.html. 2

[MU04] MCGUIRE M., UNIVERSITY B.: *Chapter 9. Efficient Shadow Volume Rendering*. Addison-Wesley Professional (April 1, 2004), 2004. 2, 6

[PSM*13] PEČIVA J., STARKA T., MILET T., KOBRTEK J., ZEMČÍK P.: Robust silhouette shadow volumes on contemporary hardware. In *Conference Proceedings of GraphiCon'2013* (2013), GraphiCon Scientific Society, pp. 56–59. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10408. 2

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. *ACM Trans. Graph. 21*, 3 (July 2002), 557–562. URL: http://doi.acm.org/10.1145/566654.566616, doi:10.1145/566654.566616. 2

[SEA08] SINTORN E., EISEMANN E., ASSARSSON U.: Sample based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2008) 27*, 4 (2008), 1285–1292. 2

[SOA11] SINTORN E., OLSSON O., ASSARSSON U.: An efficient alias-free shadow algorithm for opaque and transparent objects using per-triangle shadow volumes. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 153:1–153:10. URL: http://doi.acm.org/10.1145/2070781.2024187, doi:10.1145/2070781.2024187. 2

[SWK07] STICH M., WÄCHTER C., KELLER A.: Efficient and robust shadow volumes using hierarchical occlusion culling and geometry shaders. In *GPU Gems 3* (2007), Nguyen H., (Ed.), Addison Wesley Professional, pp. 239–256. 2

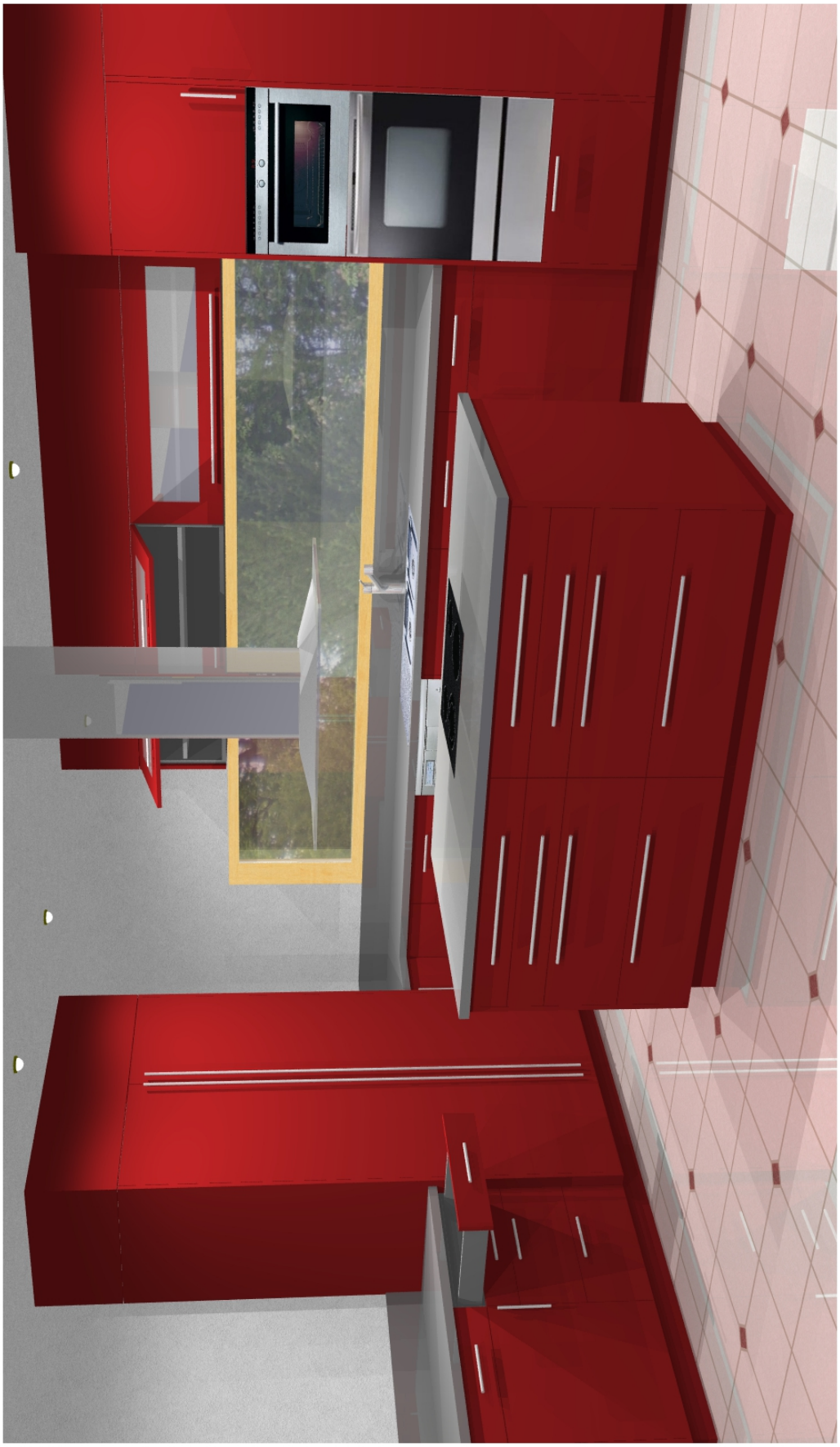
[vW05] VAN WAVEREN J.: Shadow volume construction, 2005. 2

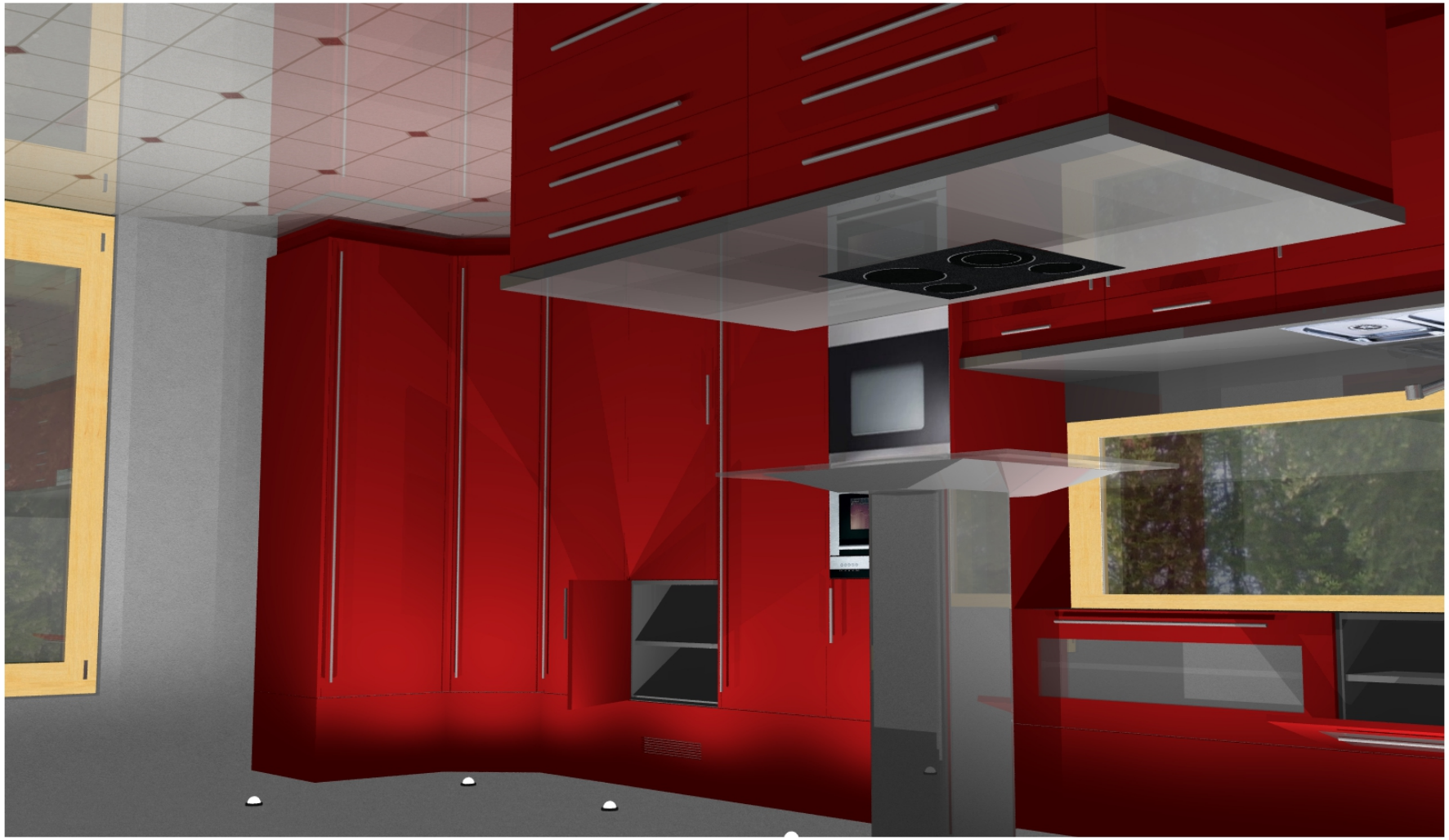
[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (Aug. 1978), 270–274. URL: http://doi.acm.org/10.1145/965139.807402, doi:10.1145/965139.807402. 1, 2

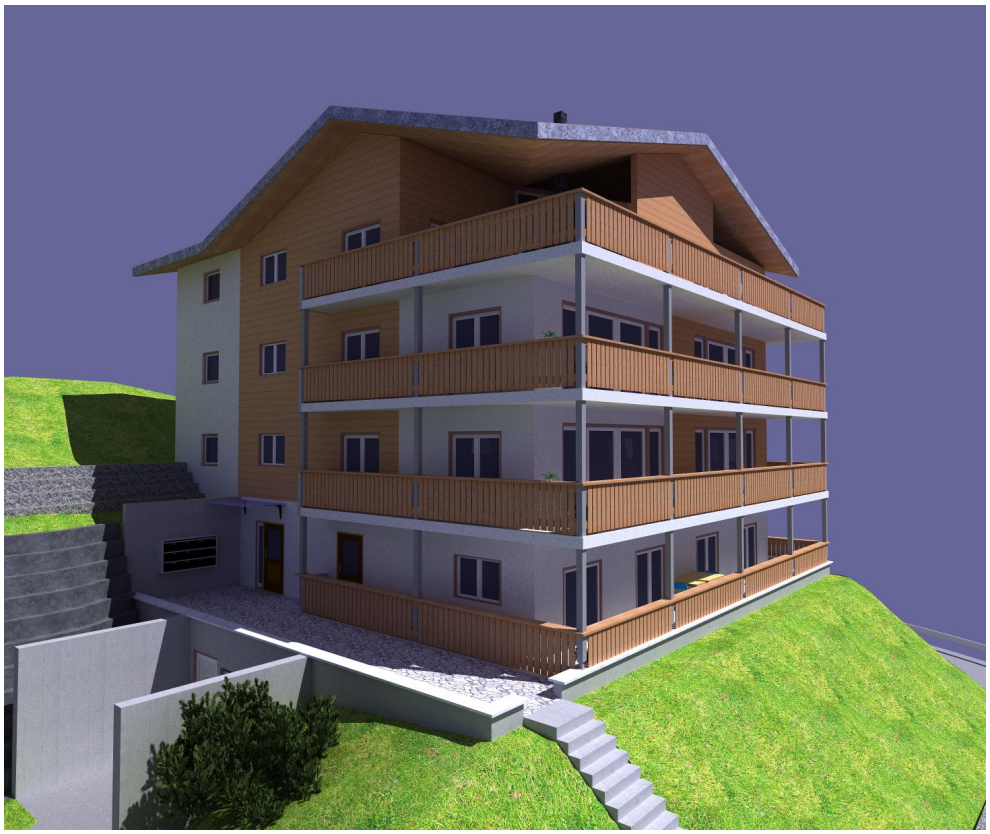
[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2004), EGSR'04, Eurographics Association, pp. 143–151. URL: http://dx.doi.org/10.2312/EGWR/EGSR04/143-151, doi:10.2312/EGWR/EGSR04/143-151. 2

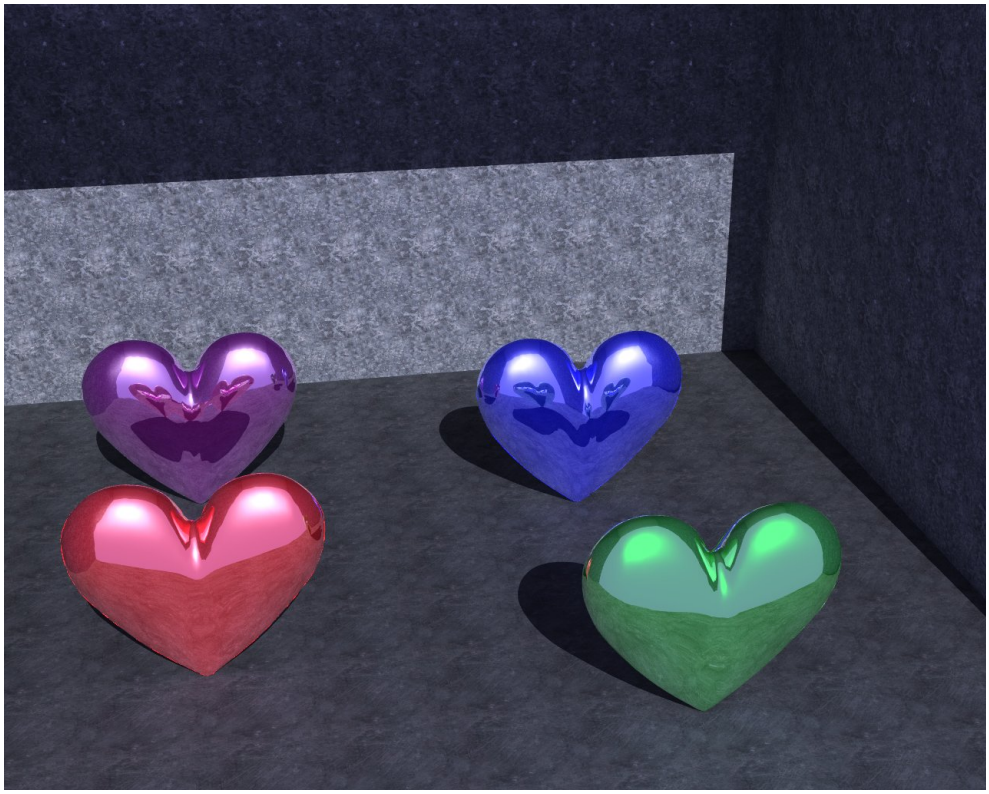
[ZSXL06] ZHANG F., SUN H., XU L., LUN L. K.: Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications* (New York, NY, USA, 2006), VRCIA '06, ACM, pp. 311–318. URL: http://doi.acm.org/10.1145/1128923.1128975, doi:10.1145/1128923.1128975. 2

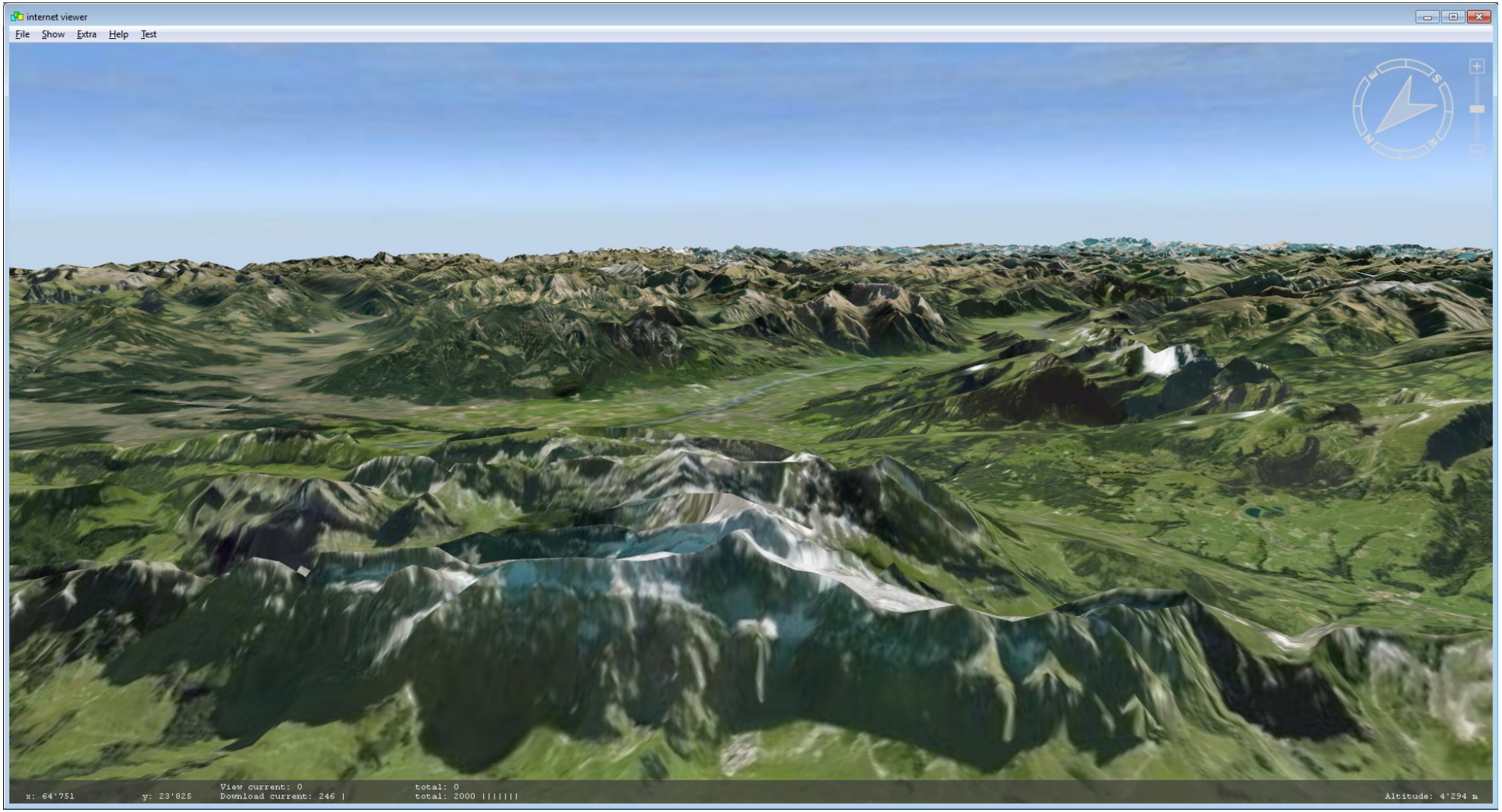
Image Gallery

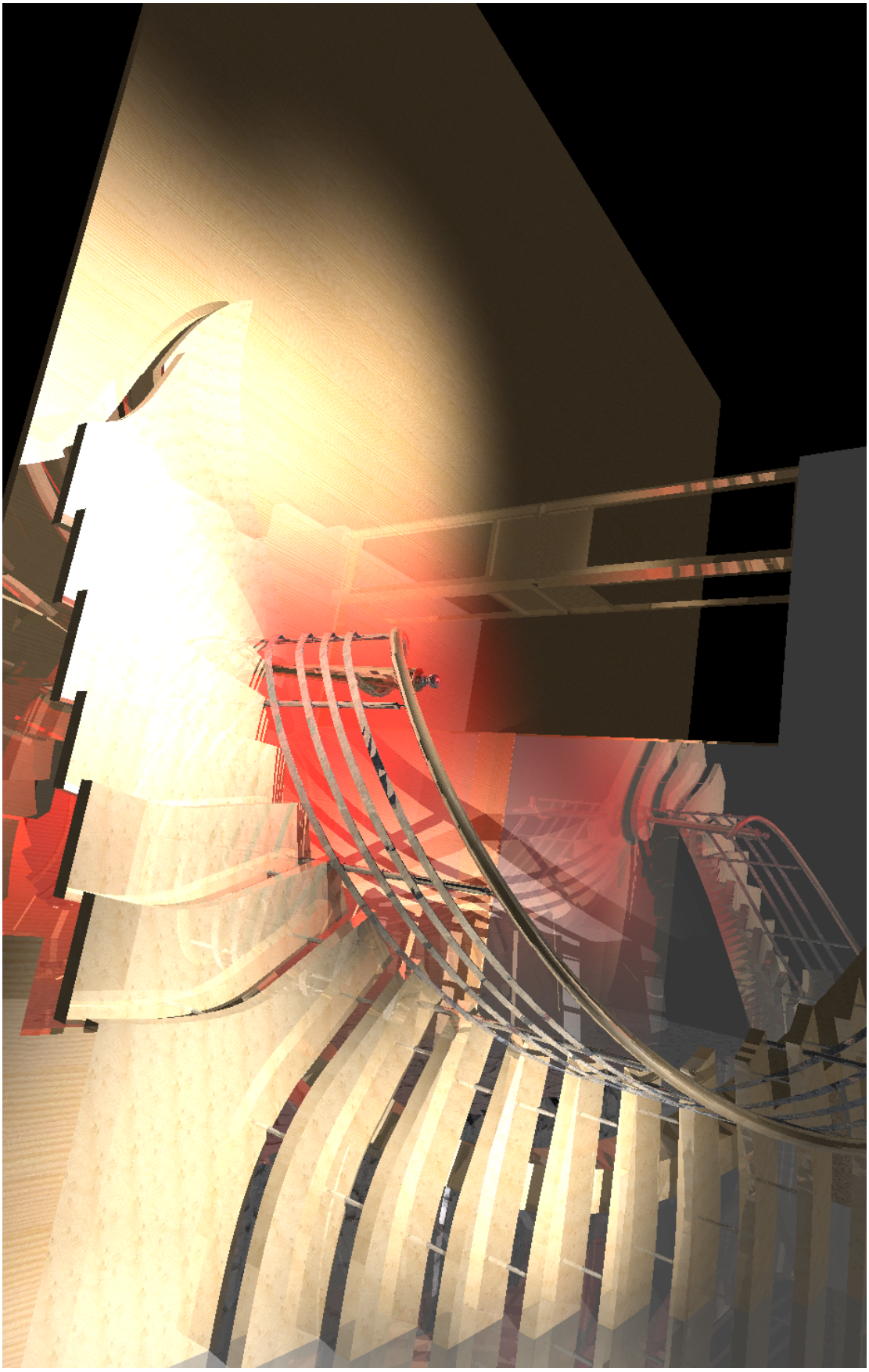


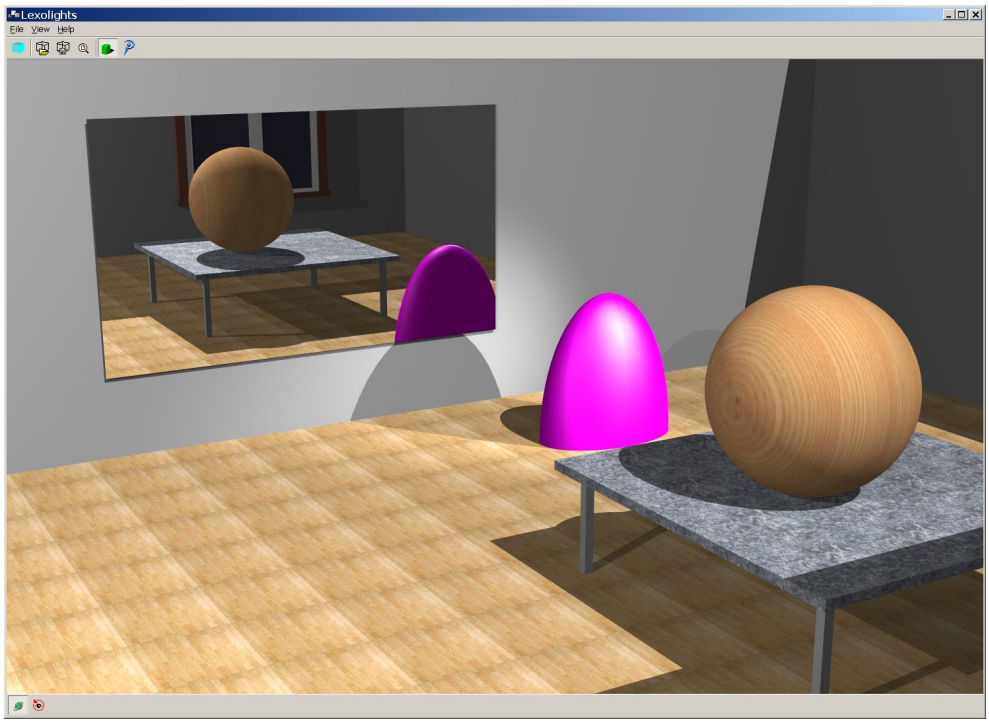


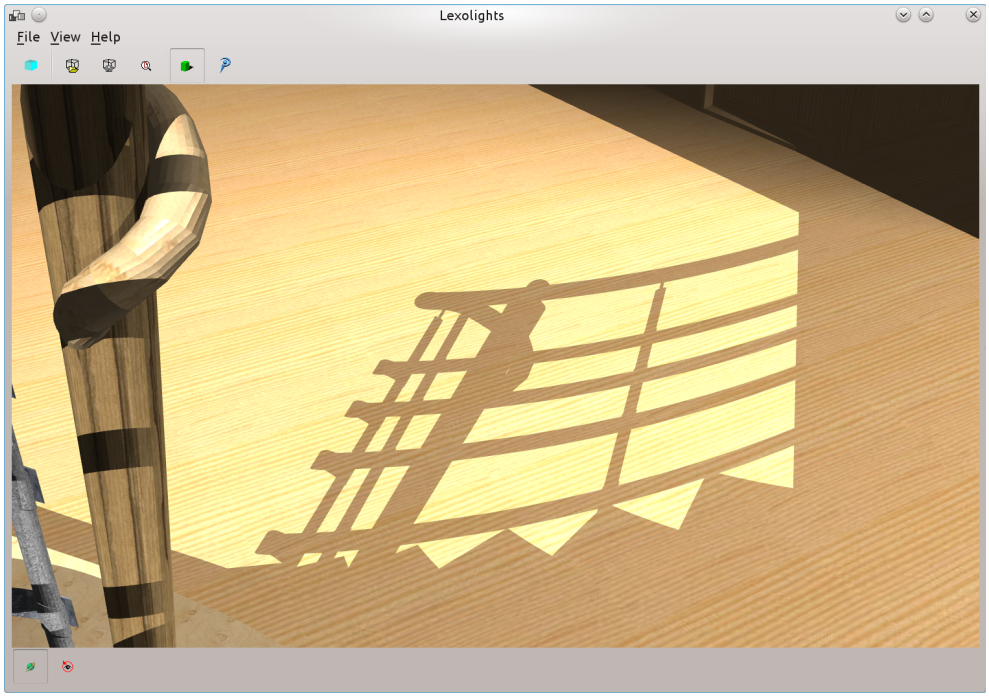





internet viewer
File Show Extra Help Test
x: 64'751 y: 23'825
View current: 0
Download current: 246 |
total: 0
total: 2000 |||||||
Altitude: 4'294 m




Lexolights
File View Help


Lexolights
File View Help