Souhrnná zpráva za rok 2012 z projektu "Vývoj softwaru v oblasti CAD systémů, 3D grafiky a vizualizace grafických scén"

Jan Pečiva

Faculty of Information Technology, Brno University of Technology

Abstract

High quality visualizations with realistic lighting is challenging topic in computer graphics. This report presents Lexolights architecture for visualization of virtual scenes when high fidelity lighting is required. The shader architecture providing per-pixel lighting is described, followed by the discussion of shadow techniques used in computer graphics today and their suitability for high quality lighting. Then, multipass concept used in Lexolights is presented as the multipass seems to be a robust approach for crossing all the limits of the maximum number of lights. Finally, our approach for rendering of models with huge texture data sets is presented.

1. INTRODUCTION

High quality visualizations became more and more important in computer graphics and simulations. They allow for proper light simulation in the virtual scene and allow close-to-reality perception of simulated virtual environment.

The close-to-reality visualization or photorealistic visualization provides the better perception of the simulated environment and provide clearer idea, for example, to a customer about the real product, he is going to get, already in simulation phase of the product development. Such applications may include industry design, architecture, civil and mechanical engineering, and CAD applications in general.

2. STATE OF THE ART

Much of the research of photorealistic visualizations was done using raytracing approach [Whitted 1980]. Because raytracing is generally slow and performance expensive, number of optimizations were developed [Arvo 1989][Kajiya 1986][Lafortune 1993][Jensen 1995] [Jensen 2001][Veach 1997]. Ray raytracing is generally slow and performance expensive, but it is able to simulate advanced light behaviour, thus providing better visual results.

These approaches were used by number of software tools, such as POV-Ray, Yafaray, and Radiance ray-tracing software system [Larson 1998]. Some tools attempted real-time raytracing by using huge cluster of computers [Muuss 1987], or recent multi-core systems [Valich 2008]. Other people used huge computing power of GPUs [Purcell 2002][Horn 2007][Gunther 2007][Shih 2009][Garanzha 2010]. Finally, Nvidia came with its own GPU-accelerated ray tracing API called OptiX [Nvidia 2011].

Although GPU raytracing seems a promising approach for real-time high quality visualizations, it is currently capable of processing about 100Mrays/second [Ludvigsen 2010] on scenes with 100K triangles (roughly said). It is still a level of magnitude behind the processing power of traditional rendering paradigm used, for instance, by OpenGL based visualization applications.

Because of performance advantage of traditional OpenGL rendering paradigm, many people prefer it over ray tracing approaches. Recent developments in OpenGL shader architecture [Segal 2010] allows to reach high level of realism even without ray tracing and other photorealistic approaches.

[Loviscach 2004] attempted to create photorealistic renderer for Cinema 4D modelling software using traditional OpenGL pipeline empowered by shaders. [Peciva 2011] does the same for POV-Ray ray tracing software, emulating POV-Ray core functionality in shaders.

This report describes the architecture for close-to-photorealistic visualization and light simulation that was implemented and tested in Lexolights visualization tool. The architecture was verified on number of CAD models while number of challenges was identified and this report will describe the solutions. The difficulties appear particularly when going from laboratory models to general models produced by the real users that does not have size or physical limits and often brings very surprising constructions or data organization. The challenges are analysed and described. The system is built using OpenSceneGraph visualization library based on OpenGL. The system can be used as a visualization front-end for a simulation software when high level of scene realism is required or for the simulation of light conditions and light distribution in the virtual scene.

3. OVERVIEW

Our toolkit is composed of the three components:

1. Lexocad – free CAD oriented modelling tool (www.lexocad.com)

2. Lexolights – open source visualization tool based on OpenSceneGraph (lexolight.sourceforge.net)

3. Inventor import – plugin for OpenSceneGraph allowing robust import of models in Open Inventor file format [Wernecke 1994]

Lexocad is our modelling tool used to create 3D models. These models are imported using Open Inventor plugin to OpenSceneGraph data structures. The data are processed and taken for visualization by our Lexolights visualization tool.

This report focuses just on Lexolights visualization tool that can be easily used as a visualization front-end for any kind of simulation requiring high fidelity visualizations.

3.1 Visual Fidelity

Why to deal with visual fidelity? The figures 1 and 2 are simple examples of the problem. The traditional OpenGL visualization paradigm based on triangles exhibits the problems of correct lighting (figure 1) practically on all CAD models containing lights. To go around the problem shaders and per-pixel lighting techniques need to be used. However, even these do not provide correct scene lighting that would include shadowing and natural light scene distribution. This report investigates the challenges of high quality lighting and describes Lexolights architecture to make a proper visualization with lighting.

3.2 Per-Pixel Lighting

Nowadays, per-pixel lighting is a "must" when making high quality lighting in the graphics scene. The shader architecture was developed for this purpose as a part of OpenGL and DirectX (GLSL [Kessenich 2010] and HLSL [MSDN 2011] shading language). Many projects including computer games are using pre-built shaders that are used for scene rendering doing per-pixel lighting.

However, such approach is difficult to follow in highly dynamic simulations and advanced CAD applications as the graphics scene is not known in advance. Thus, the shaders can not be built-in the application, but they have to be generated dynamically, based on the scene requirements.

Lexolights contains shader generator capable of creation of required vertex and fragment shaders performing advanced per-pixel lighting based on Phong lighting model [Phong 1973] currently.

One particular problem with shaders is their compilation cost when they are translated to the machine code of graphics processor. As the compilation time of each shader takes tens of milliseconds according to our experience, it can be very effective to use the cache of already compiled shader programs and reuse them whenever possible. Our experience shows that amount of reused shaders is quite high on the most CAD models and the speed up is sometimes quite noticeable.

3.3 Shadows

Shadows are very important for human perception. The brain is using shadow information for depth estimation and other "3D feeling". The absence of shadows in the scene gives the impression of artificial not real scene. Thus, shadows are very important in computer graphics.

In ancient times of computer graphics, precomputed shadows were used. They were performance cheap as they were static and built in the scene textures. However, nearly all simulations deal with dynamic scenes that changes through the time. For dynamic shadows, two major approaches are used nowadays: shadow maps and shadow volumes.

Shadow maps [Williams 1978] are very popular today because they are fast and allow for special filtering effects like soft shadows. On the other side, they suffer by shadow map resolution problems, like perspective aliasing (see figure 3). There were attempts to optimize the resolution distribution [Stamminger 2002][Wimmer 2004][Zhang 2006]. The optimizations are sufficient in many cases, but they do not remove the problem. Another

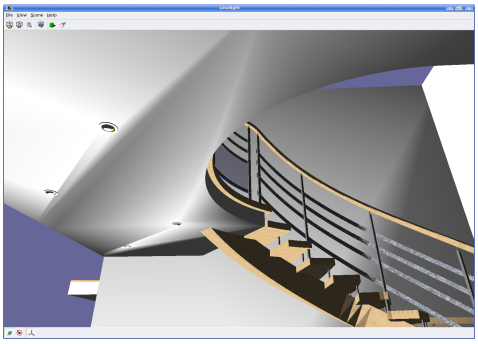


Figure 1: Lighting artifacts on a scene using old-fashioned OpenGL Gouraud shading with advanced light setups

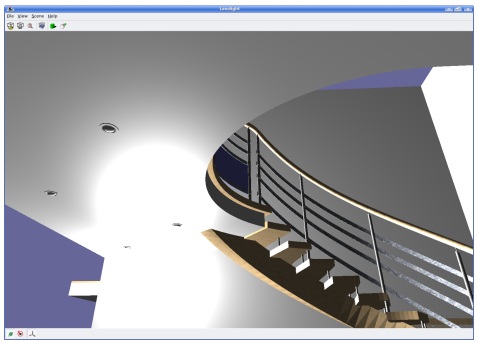


Figure 2: Per-pixel lighting in a scene rendered using shader technology

Figure 3: Shadow artefacts when using shadow map technique

issue with shadow maps is the memory consumption. Each light source may allocate, for instance, 2k x 2k depth texture, occupying 16MiB of video memory. When we are dealing with tens or hundreds of light sources, this may become an issue. Thus, we will probably want to switch to multipass solution when rendering too many lights that would consume too much of available video memory.

Shadow volumes [Crow 1977] use stencil buffer to create shadow effect. However, as there is only one stencil buffer per render target available (with exception of multi-layered rendering), multipass usually need to be used and each light must be rendered separately. The result of each light is blended to the final image. Our experience shows that shadow volumes are roughly ten times slower than shadow maps, but it very depends on the scene. On the other side, shadow volumes are rendered with pixel precision, or sub-pixel precision when multisampling is used. From visual point of view, they are often considering better option over shadow maps. There were also attempts to create soft shadows using shadow volumes [Assarsson 2003].

From the point of performance, it is possible to render even 4k lights, but without shadows, just by storing them, for example, as a texture data and iterate over them in the shader. This way, all light contributions will be computed in one pass. For more than 4k lights, we can use, for instance, texture arrays to push limits much further. Anyway, shadows are too important for human perception, so they should not be ignored for high quality visualizations.

Shadow maps have the limit of concurrently rendered lights given by the amount of textures that can be accessed from the shaders and by available texturing memory. In general, a good bet is about 32, 16 or 8 lights per rendering pass if using 2k x 2k shadow textures.

Shadow volumes usually allows only one light to be rendered per pass. It noticeably limits the performance, but shadow volumes can be considered very stable shadow technique that works correctly on the most of the models while shadow maps suffer from various problems. For example, very large models with tiny details that user may want to zoom on: such details will not cast proper shadows because of shadow map resolution while shadow volumes usually handle such cases without problems. In conclusion, as the visualized model is not known in advance and it does not have size and detail limits, it is not safe to use shadow maps for visualization. Thus Lexolights is using shadow volumes by default while the user may still manually switch to other shadow technique if he knows that sufficient results will be provided to him.

3.4 Silhouette shadow volumes

The method finds silhouette edges by looping over every edge in the model. Each edge is processed in parallel in Tessellation Control Shader where multiplicity is computed. An input patch primitive is composed of two vertices that describe an edge, one integer that contains number of opposite vertices and n opposite vertices, see Figure 4. Because patch the size must be constant, some positions are not used.

A vertex buffer of model has to be extended by En vertices, which is the number of edges in the model. We used element buffer to reduce memory requirements, as can be seen in Fig. 5.

Byungmoon's algorithm [Kim et al., 2008], as in its core proposal, has a flaw that multiplicity is not calculated in a deterministic way. In older approach [Peciva et al., 2013], it was fix this by calculating multiplicity per triangle and if the 3 results troughout all 3 edges were not consistent, we discarded the triangle from further processing, because it meant that the triangle is almost parallel to the light and does not cast a shadow. We further improved this approach - multiplicity is now computed only once for each opposite vertex using reference edge. A choice of reference edge has to be the same for all occurrences of a triangle. This can be achieved for ex-

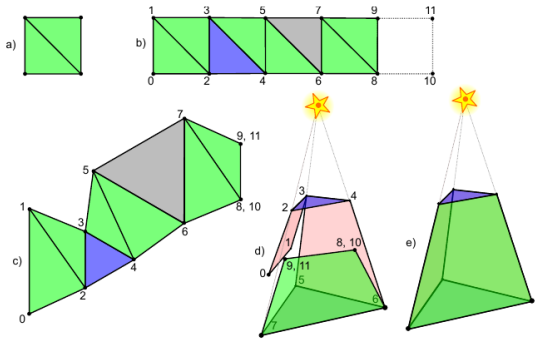


Figure 3: Single-pass per triangle method, a full shadow volume is created in a single pass. One point is added to the triangle in order to form a quad *a*) which is then tessellated using factors (1, 5, 1, 5),(5, 1) *b*). Points 10 and 11 are merged with 8, 9. Light cap is visualized as blue, dark cap grey *c*). Then we join points 0-7, 1-5, 2-9, 4-8 and push points 5, 6, 7 to infinity *d*) to make the volume *e*).

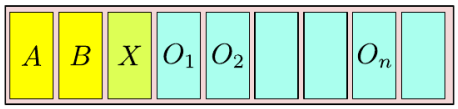


Figure 4: Input patch for tessellation control shader

ample by introducing vertex ordering - Equations 1 and Algorithm 2.

$$\begin{aligned} \mathbf{A} < \mathbf{B} &\Leftrightarrow \text{Greater}(\mathbf{A},\mathbf{B}) < 0 \\ \mathbf{A} = \mathbf{B} &\Leftrightarrow \text{Greater}(\mathbf{A},\mathbf{B}) = 0 \\ \mathbf{A} > \mathbf{B} &\Leftrightarrow \text{Greater}(\mathbf{A},\mathbf{B}) > 0 \end{aligned} \tag{1}$$

Data: Vertices $\mathbf{A},\mathbf{B}$
Result: Result r of comparison
$\mathbf{S} = \text{sgn}(\mathbf{A}-\mathbf{B})$;
$\mathbf{K} = (4,2,1)$;
$r = \mathbf{S}\cdot\mathbf{K}$
Algorithm 2: Function Greater$(\mathbf{A},\mathbf{B})$ used for vertex ordering.

In order to guarantee consistency, reference edge of a triangle in our algorithm is constructed using smallest and larges vertex of a triangle, as in Algorithm 2. More options for such method are available, but evaluation per each triangle occurance must be consistent in order to get correct results.

To simulate behaviour of Byungmoon's algorithm (edge casts a quad as many times as it has multiplicity), we tessellate the casted quad from the edge using inner tessellation levels $(Multiplicity \cdot 2 - 1, 1)$ and then we bend the tessellated quad in evaluation shader in a way to create m overlapping quads, as seen in Fig. 1, which demonstrates edge A-B having multiplicity of 3.

The procedure of multiplicity calculation is described in Algorithm 3 and 4.

Data: Edge $\mathbf{A},\mathbf{B}$, $\mathbf{A} < \mathbf{B}$, set $\mathfrak{O}$ of opposite vertices $\mathbf{O}_i \in \mathfrak{O}$, light position $\mathbf{L}$ in homogeneous coordinates
Result: Multiplicity m
$m = 0$;
for $\mathbf{O}_i \in \mathfrak{O}$ **do**
  if $\mathbf{A} > \mathbf{O}_i$ **then**
    $m = m + CompMultiplicity(\mathbf{O}_i,\mathbf{A},\mathbf{B},\mathbf{L})$;
  else
    if $\mathbf{B} > \mathbf{O}_i$ **then**
      $m = m - CompMultiplicity(\mathbf{A},\mathbf{O}_i,\mathbf{B},\mathbf{L})$;
    else
      $m = m + CompMultiplicity(\mathbf{A},\mathbf{B},\mathbf{O}_i,\mathbf{L})$;
    end
  end
end
Algorithm 3: Modified algorithm for computation of final multiplicity of edge $\mathbf{A},\mathbf{B}$

Data: Vertices $\mathbf{A},\mathbf{B},\mathbf{C}$; $\mathbf{A} < \mathbf{B} < \mathbf{C}$; light position $\mathbf{L}$ in homogeneous coordinates
Result: Multiplicity m for one opposite vertex
$\mathbf{X} = C - A$;
$\mathbf{Y} = (l_x - a_x l_w, l_y - a_y l_w, l_z - a_z l_w)$;
$\mathbf{N} = \mathbf{X} \times \mathbf{Y}$;
$m = \text{sgn}(\mathbf{N}\cdot(\mathbf{B}-\mathbf{A}))$;
Algorithm 4: $CompMultiplicity(\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{L})$ function used in algorithm 3

After tessellation, we have to transform tessellation coordinates into vertex position of the shadow volume quad in the evaluation shader. The algorithm for its implementation is described in Algorithm 5 and Equations 2.

$$\begin{aligned} \mathbf{A} &= (a_x, a_y, a_z, 1)^T \\ \mathbf{B} &= (b_x, b_y, b_z, 1)^T \\ \mathbf{C} &= (a_x - l_x, a_y - l_y, a_z - l_z, 0)^T \\ \mathbf{D} &= (b_x - l_x, b_y - l_y, b_z - l_z, 0)^T \end{aligned} \tag{2}$$

Because caps are not generated, this method can also be used with simpler z-pass algorithm.

2.3 Implementation

All our methods were implemented in Lexolights, an open-source multi-platform program based on OpenSceneGraph and Delta3D, using OpenGL.

Single-pass per-triangle method suffers from inconsistent rasterization of two identical triangles at the same depth but with different winding - depth of fragments from both triangles differs, which resulted in z-fighting artiffacts. We had to manually push the front cap's

Data: Vertices $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}$, tessellation coordinates $x,y \in \langle 0,1\rangle$ and multiplicity m
Result: Vertex $\mathbf{V}$ in world-space
$\mathbf{P}_0 = \mathbf{A}$;
$\mathbf{P}_1 = \mathbf{B}$;
$\mathbf{P}_2 = \mathbf{C}$;
$\mathbf{P}_3 = \mathbf{D}$;
$a = \text{round}(x \cdot m)$;
$b = \text{round}(y)$;
$id = a \cdot 2 + b$;
$t = (id \mod 2) \hat{} (\lfloor id/4 \rfloor \mod 2)$;
$l = \lfloor (id+2)/4 \rfloor \mod 2$;
$n = t + l \cdot 2$;
$\mathbf{V} = \mathbf{P}_n$;
Algorithm 5: This algorithm transforms tessellation coordinates into the vertex of side of shadow volume. Vertices $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}$ are computed using Equation 2.

fragments into depth of 1.0f, so they would fail the depth test, otherwise we observed self-shadowing artiffacts. Bypassing early depth test in rasterization due to assigning depth values in fragment shader causes significant performance loss over two-pass method. This method served as a basis for silhouette-based approach.

For caps generation in silhouette-based method, we used gemetry shader and multiplicity calculation, using which we calculated triangle's orientation towards light source via reference edge. It was also necessary for keeping discarding calculations consistent throughout the rendering process of shadow volumes.

Because tessellation factors are limited, at the time of writing, to 64, there is also a limit of maximum multiplicity per edge that this algorithm is able to process. Acording to equation to calculate tessellation factor $Multiplicity \cdot 2 - 1$, maximum multiplicity of an egde is 32, which should be more than enough for majority of models. But for example well-known Power Plant model (12M triangles) has some edges, which have multiplicity of 128. In that case, they would have to be splitted into more edges.

3 EXPERIMENTS

We compared our methods against already available shadow volumes implementations on modern hardware - robust geometry shader implementation and standard shadow mapping, using which we also tried to evaluate performance against Sintorn's AFSM [Sintorn et al., 2008]. We also tested two-pass per-triangle method against similar geometry shader implementation. For shadow volumes approaches, z-fail was used; shadow map resolution was set to 8k x 8k texels.

Testing platform had following configuration: Intel Xeon E3-1230V3, 3.3 GHz; 16GiB DDR3; GPUs: AMD Radeon R9 280X 3 GiB GDDR5, nVidia GeForce GTX 680 2 GiB GDDR5; Windows 7 x64; driver version: 13.12 (AMD), 334.89 (nVidia).

Spheres10x10	R280				G680			
Triangles	TS		GS		TS		GS	
32400	984	**995**	490	484	739	**825**	542	540
67600	921	**963**	488	487	624	**667**	494	513
102400	615	**729**	484	479	491	**555**	372	402
360000	203	233	270	**272**	218	**228**	131	135
1081600	72	88	104	**110**	82	**94**	46	49
1440000	56	72	84	**91**	67	**81**	36	39
1960000	34	41	59	**62**	49	**58**	26	28

Table 1: Performance of two determinism methods measured in FPS on a scene with 10x10 spheres at different triangle count.

3.1 Testing Scenes

We created a camera fly-through in two testing scenes, each having one point light source.

- Sphere scene: synthetic scene containing adjustable number of spheres (typically 100) with configurable amount of detailness. Fly-through has 16 seconds.
- Crytek Sponza: popular model used to evaluate computer graphics algorithms. 262 267 triangles, 40 seconds.

3.2 Results

Majority of our tests was performed on a sphere scene with adjustable amount of geometry. First, we made a flythrough in a scene containing 100 spheres with different amount of triangles per scene, the results can be seen in Table 1 and graph in Fig. 5.

On GTX680, tessellation using reference edge is the fastest, no matter the number of triangles, although the performance gaps gets smaller with increasing number of triangles in scene. R9 280X showed different results, tessellation was more than 2x faster when the scene contained only 32K triangles but at approximately 300K, geometry shader method took lead.

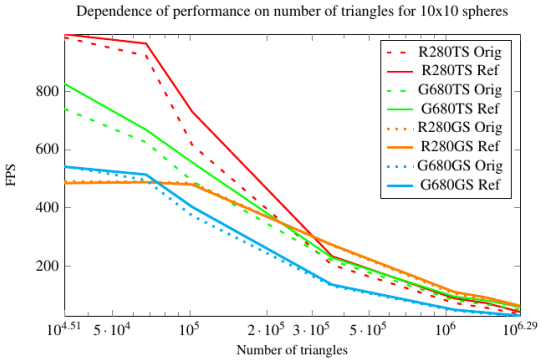


Figure 5: Dependence of performance (FPS) on number of triangles on a scene with 10x10 spheres using original and new deterministic method.

Spheres 1M	R280				G680			
Objects	TS		GS		TS		GS	
1	73	92	111	**120**	106	**134**	55	60
4	74	94	113	**121**	101	**126**	53	58
25	64	76	97	**101**	76	**88**	46	50
64	68	76	**90**	89	61	**66**	40	43
100	64	70	**84**	82	58	**62**	39	42
240	58	55	**70**	64	**50**	49	35	36
399	53	48	**61**	54	**36**	36	27	28
625	43	38	**53**	46	**29**	27	22	22
851	40	44	46	**50**	24	**25**	19	20
1250	35	**37**	28	31	**19**	19	16	16
2500	**23**	19	15.1	15.4	**12**	11	10.8	10.1
3116	21.2	**21.5**	12.8	12.5	11.1	**11.2**	9.1	9.2
3920	**15.7**	14	10.1	10.12	**9.1**	8.7	7.7	7.5
5100	**14.8**	14.2	7.8	7.75	**8.2**	8.2	6.7	6.8
15600	**7.45**	6.45	3.07	3.14	**10.5**	9.1	3.6	3.6

Table 2: Dependence on number of objects for spheres scene with 1M triangles. Bold values represent the fastest algorithm/implementation for respective number of objects, per GPU.

We further extended this test to performance dependency on number of objects in a scene while maintainig constant amount of geometry. This measurement was carried out on Sphere scene, having 1 million triangles (with deviation max 2%) in every case. No hardware instancing was used, every object was drawn via separate draw call. Results can be seen in Table 2 and graph in Figure 6.

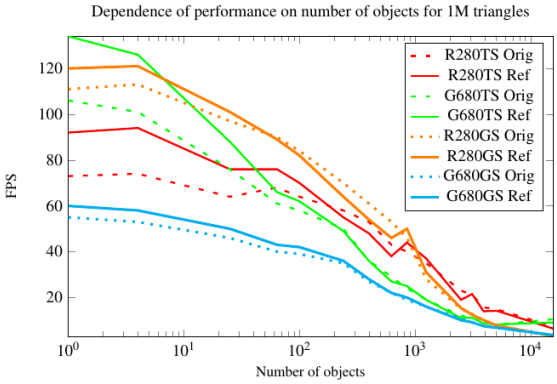


Figure 6: Dependence on number of objects for spheres scene with 1M triangles.

Contrary to previous measurements, tessellation was faster on R9 280X, starting from 10^3 objects, although reference edge was faster only in 40% cases. Moreover, as can be seen in Fig. 6, there is a slight increase in FPS in both geometry shader and tessellation implementations at about 1000 objects on Radeon. On GTX680, tessellation method was faster in every case; eferenge edge provided increased performance only in a half of measurements, but in all other cases the difference was only 1-3 FPS.

Sintorn in his AFSM paper Sintorn et al. [2008] stated that his per-pixel precise shadow maps are 3-times slower than standard 8Kx8K shadow mapping. In order to evaluate our algorithm against AFSM, we conducted a measurement against shadow mapping having resolution metioned above, results of which are in table 3 and graph 7.

Spheres10x10	R280		G680	
Triangles	TS	SM	TS	SM
32400	**995**	252	**825**	245
67600	**963**	250	**667**	237
102400	**729**	244	**555**	225
360000	**233**	219	**228**	190
1081600	88	**168**	94	**135**
1440000	72	**155**	81	**115**
1960000	41	**120**	58	**103**

Table 3: Shadow Mapping vs Tessellation Silhouettes, 10x10 sphere scene, FPS

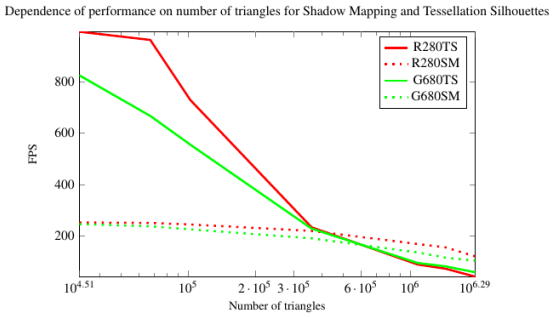


Figure 7: Shadow Mapping vs Tessellation Silhouettes on a scene with 10x10 spheres, measured in frames per second.

Not only we managed to outperform shadow mapping with triangle count up to ~400K triangles, but at almost 2M triangles our method was on par or faster than AFSM - R9 280X dropped to 34% of SM performance whereas GTX680 was only 44% slower than 8K shadow mapping.

We also compared silhouette methods with two-pass per-triangle tessellation implementation and 8K shadow mapping (only on sphere scene, our framework does not support omnidirectional shadow mapping) on Crytek Sponza scene, results in table 4 and graph 8.

One can observe that per triangle tessellation method is even faster than than both geometry shader methods running on Sponza scene. It is also worth noting that per-triangle geometry-shader-based method provides more performance on this scene than silhouette-based approach. On GTX680, the difference between silhouette and per-triangle tessellation method is 122%, whereas on R9 280X card it is only faster by 27%.

With increased amount of geometry in our synthetic test scene, the situation turns around in favor to silhouette methods. Also performance difference between shadow mapping and tessellation on Radeon drops under 1/3 ratio, but GeForce is still able to maintain 43% of SM performance.

4 CONCLUSIONS

We have developed new methods for computing shadow volume silhouettes using tessellation shaders.

	R280		G680	
Method	Spheres	Sponza	Spheres	Sponza
TS Triangle	5.8	102	7.9	83
TS Silhouette	23.7	130	32	185
GS Triangle	3.1	51	4.9	73
GS Silhouette	34	49	14.8	62
SM	93	0	74	0

Table 4: Overall comparison of GS, TS methods and classic 8K shadow mapping on testing scenes - Sponza, and Spheres with 4M triangles. Shadow mapping was not evaluated on Sponza scene (zeros).

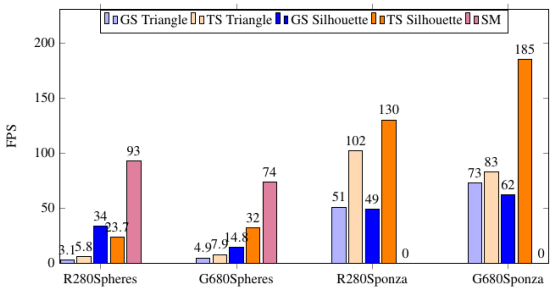


Figure 8: Overall comparison of methods on testing scenes - Sponza and Spheres with 4M triangles.

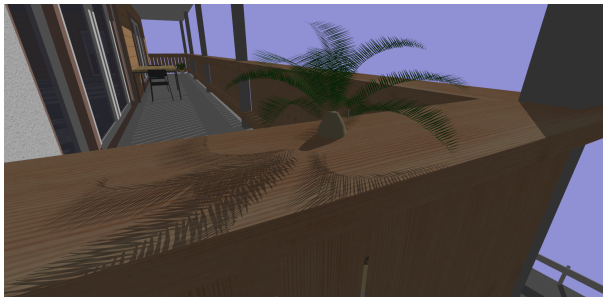
Our two-pass per-triangle tessellation method is, in some cases, faster than silhouette algorithm implemented in geometry shader, but loses performance as geometry amount in the scene grows. Compared to geometry shader per-triangle implementation, it was faster in every measurement.

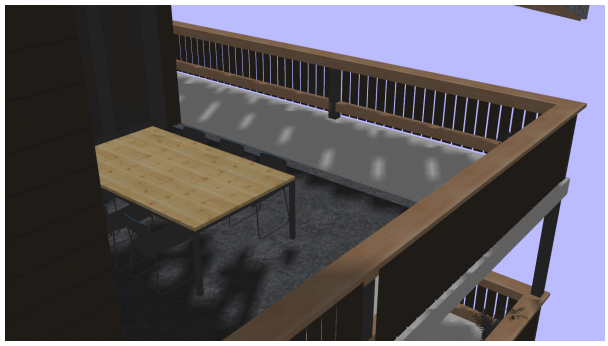
The silhouette method is more efficient, and as we have proven in our measurements, mostly in scenes with higher amount of geometry. GeForce GTX680 benefited mostly from this algorithm, being faster than geometry shader silhouette method. As for Radeon R9 280X, geometry shader method is more suitable. Tessellation method on Radeon proved to be faster in Sponza scene, but our synthetic tests on sphere scene showed that it's performance is dominant only up to ~300K of triangles when having multiple objects in the scene, or only up to 15K triangles when only a single detailed object was drawn. In less detailed scenes it was able to outperform nVidia-based card, but only up to aforementioned 300K triangles.

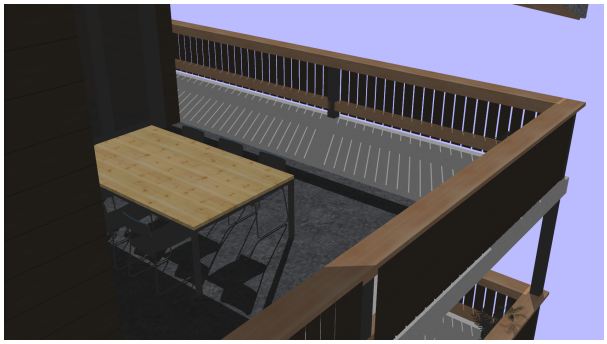
Our robust algorthm was sped up by using a novel method of multiplicity computation, which was able to provide up to 31% performance gain in tessellation method (13.5% in average), maximum speedup in geometry shader was 10.7% with average of 3.4%.

In comparison to standard SM and Sintorn's Alias-Free Shadow Maps (AFSM), our tessellation method provides better performance than 8K shadow maps up to ~400K triangles and then fall to 43% performance of shadow mapping at 4M triangles on GeForce, 34% on Radeon, which is on par or better than AFSM (it's 3-times slower than 8K SM) and is also simplier to implement.

In the future, we would like to see an arbitrary $\pm$ stencil operation in hardware, configurable in shaders, which would allow us to increase the speed of our method even more, due to a lower number of triangles being drawn. We also want to evaluate more hardware platforms and explore GPGPU potential in the filed of shadow vol-







3.4 Multipass Rendering

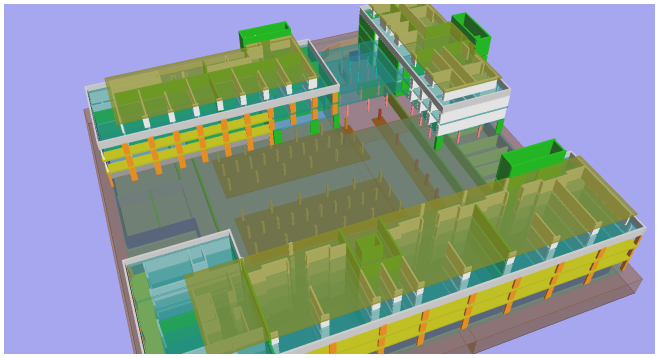


Figure 4: Model of Vivaldi shopping center visualized using many transparent parts

Since the rendering of all lights in one pass seems difficult, particularly when using hundreds of lights or when using shadow volumes, multipass may needs to be utilized.

As Lexolights is using multipass whenever needed, i.e. in shadow volumes mode, it is activated when there is more than one light. We used OpenSceneGraph RenderBin concept while each RenderBin represents one pass. The RenderBins are ordered and scheduled by their bin number.

The purpose of the first pass in Lexolights is to render scene with ambient light only. Lexolights combines this pass with the second pass and renders ambient light and the first light together to optimize for performance.

The first pass produces z-values to the depth buffer and color information for the ambient light with addition of the first light. Following passes appends just the light contributions of other lights, iterating through all of them and blending them to the final image. Blending equation is set to FUNC_ADD and z-buffer depth function to EQUAL or LEQUAL. All opaque objects can be rendered this way.

Transparent objects are more difficult to visualize. They are usually rendered after all opaque objects, because their color contribution is combined with the opaque objects already present in the frame buffer. They are usually rendered with activated blending and with z-test on but with disabled z-buffer updates. Moreover, transparent objects should be sorted based on distance and rendered in back-to-front, or rarely front-to-back, order. For object sorting, we direct all transparent objects to "depth sorted" render bins of OpenSceneGraph that are sorted automatically. The render bin number is set in such a way that it is rendered after all other opague render bins. Figures 4 and 6 shows some of the visual results.

3.5 Large Data Sets

Large simulations and big CAD models tends sometimes to grow over the memory limits of even high-end computers. On visualization part, the memory limits that usually need to be addressed is texturing memory.

The figure 5 shows model of the whole Switzerland whose textures occupy 1.3 GiB of disk space stored as jpg compressed data. It is nearly not possible to decompress all of them and hold them in the memory of standard computer of today. When viewing the model, usually only small part of the model requires highest detail while lower details can be used with increasing distance. For that purpose, we introduced number of texture details and utilized on demand loading based on distance from the camera.

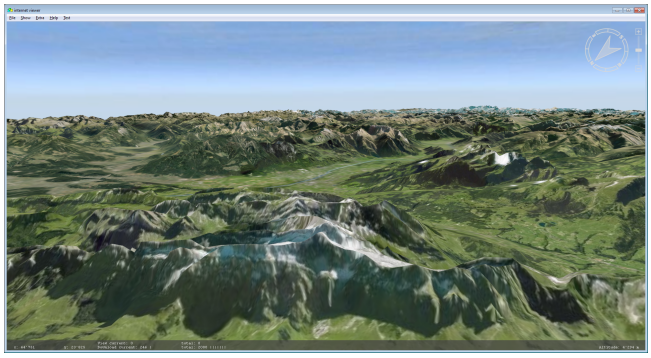


Figure 5: Visualization of Switzerland model that contains 1.3GiB jpg texture data

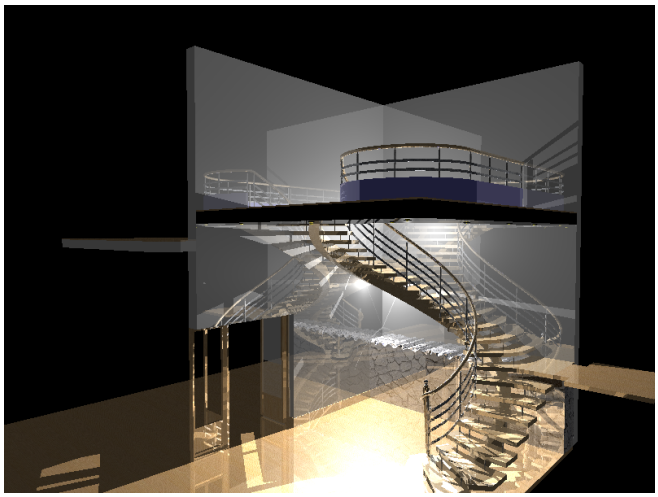


Figure 6: Visualization of stairs with reflective floor and walls made of glass

On demand loading brings challenges of smooth visualization. At first, it looked like jpg decompression can be a problem as it requires noticeable amount of CPU time. However, scheduling the decompression to the second CPU easily resolved the problem Since the most of desktop computers today have at least two CPU cores, the proposed solution should not be a problem.

More difficult problem appeared afterwards. On demand loading means that a large amount of data needs to be transmitted from main memory through the bus to the graphics card each time we want to increase the detail of any texture in the model. For instance, 4k x 4k RGB texture takes 48 MiB and sending it to graphics card by OpenGL caused rendering stalls of hundreds of milliseconds according to our experience. At the end, we developed two solutions. One is based on texture "sub-loading". Sub-loading loads the texture in several chunks, avoiding rendering stalls by distributing the load to a number of frames. The second solution is based on hardware supported compressed textures. Our decompressed jpg data are compressed again by one of texture compressions supported by the graphics card, in our case DXT1 in sufficient. Then, data are sent to the graphics card in compressed form, reducing the bus load by the factor of 4 for 16-bit textures and 8 for 32-bit textures. With such optimizations, it is possible to smoothly visualize models with very large texture sets, including our 1.3 GiB model. This was verified even on mid and entry level computers of today.

4 CONCLUSION

We described the Lexolights architecture for close-to-photorealistic visualization of large number of light sources. It can be used as a visualization front-end for simulation systems, CAD modelling, architecture, and industry simulations. Moreover, it can be used for light simulations, estimation of visual impression of particular model, or construction or verification of amount of light for example in a new construction of a school as there must be enough light throughout classrooms to avoid damage of eyes of the children.

The report described the light rendering architecture based on OpenSceneGraph using multipass technique, used shadow techniques, and optimizations related to large model visualizations.

For the future work, we want to further enhance visual quality and build visualization system that would approach photorealistic quality. For that purpose, we are considering even the use of raytracing for some special materials that can be used in the scene.

REFERENCES

ARVO, J. AND KIRK, D. 1989. A survey of ray tracing acceleration techniques. In An introduction To Ray Tracing, A. S. Glassner, Ed. Academic Press Ltd., London, UK, 201-262.

ASSARSSON, U. AND AKENINE-MÖLLER, T. 2003. A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, vol. 22, no. 3, pp. 511-520, July 2003.

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

GARANZHA, K., LOOP, C., 2010. Fast Ray Sorting and Breadth-First Packet Traversal for GPU Ray Tracing, Computer Graphics Forum 29, 2. Proceedings of Eurographics 2010, Norrköping, Sweden.

GUNTHER, J., POPOV, S., SEIDEL, H., AND SLUSALLEK, P. 2007. Realtime Ray Tracing on GPU with BVH-based Packet Traversal. In Proceedings of the 2007 IEEE Symposium on interactive Ray Tracing (September 10 - 12, 2007). IEEE/Eurographics Symposium on Interactive Ray Tracing. IEEE Computer Society, Washington, DC, 113-118. DOI= http://dx.doi.org/10.1109/RT.2007.4342598

HORN, D. R., SUGERMAN, J., HOUSTON, M., AND HANRAHAN, P. 2007. Interactive k-d tree GPU raytracing. In *Proceedings of the 2007 Symposium on interactive 3D Graphics and Games* (Seattle, Washington, April 30 - May 02, 2007). I3D '07. ACM, New York, NY, 167-174. DOI=

http://doi.acm.org/10.1145/1230100.1230129

JENSEN, H. W., CHRISTENSEN, N. J. 1995. Photon maps in Bidirectional Monte Carlo Ray Tracing of Complex Objects". *Computers & Graphics* 19 (2), pages 215—224.

JENSEN, H. W. 2001. Realistic Image Synthesis Using Photon Mapping. A. K. Peters, Ltd.

KAJIYA, J. T. 1986. The rendering equation. SIGGRAPH Comput. Graph. 20, 4 (Aug. 1986), 143-150. DOI= http://doi.acm.org/10.1145/15886.15902

KESSENICH, J. 2010. The OpenGL Shading Language (Language Version 4.1). The Khronos Group Inc. Available at: http://www.opengl.org/documentation/specs/

LAFORTUNE, E.P. AND WILLEMS, Y.D., Bi-directional Path Tracing, Computer Graphics Proc., Alvor (Portugal), 1993, pp. 145-153.

LARSON, G. W. AND SHAKESPEARE, R. 1998 *Rendering with Radiance: the Art and Science of Lighting Visualization*. Morgan Kaufmann Publishers Inc.

LOVISCACH, J. 2004 Emulating an Offline Renderer by 3D Graphics Hardware, WSCG 2004, 269-276.

LUDVIGSEN, H. AND ELSTER, A. 2010. Real-Time Ray Tracing Using NVIDIA OptiX. EuroGraphics, Norrköping, May 3-7, 2010.

MUUSS, M. J. 1987. RT & REMRT: Shared Memory Parallel and Network Distributed Ray-tracing Programs. *Proceedings of 4th Computer Graphics Workshop*, Cambridge, MA, USA, October 1987. Usenix Association, pp 86-98.

MSDN 2011. HLSL. Microsoft documentation. Available at: http://msdn.microsoft.com/en-us/library/bb509561%28v=vs.85%29.aspx

NVIDIA 2011. *NVIDIA OptiX 2*. Nvidia website: http://developer.nvidia.com/object/optix-home.html

PECIVA, J., ZEMCIK, P., NAVRATIL, J. 2011 Mimicking POV-Ray Photorealistic Rendering with Accelerated OpenGL Pipeline, Proceedings of WSCG 2011 (Pilsen, Jan 30 – Feb 2, 2011).

PHONG, B. T. 1973 *Illumination for Computer-Generated Images.* Ph.D. Thesis. UMI Order Number: AAI7402100., The University of Utah.

PURCELL, T. J., BUCK, I., MARK, W. R., AND HANRAHAN, P. 2002. Ray tracing on programmable graphics hardware. In Proceedings of the 29th Annual Conference on Computer Graphics and interactive Techniques (San Antonio, Texas, July 23 - 26, 2002). SIGGRAPH '02. ACM, New York, NY, 703-712. DOI= http://doi.acm.org/10.1145/566570.566640

SEGAL, M., AKELEY, K. 2010. *The OpenGL Graphics System: A Specification (Version 4.1)*. The Khronos Group Inc. Available at: http://www.opengl.org/documentation/specs/

SHIH, M., CHIU, Y., CHEN, Y., AND CHANG, C. 2009. Real-Time Ray Tracing with CUDA. In Proceedings of the 9th international Conference on Algorithms and Architectures For Parallel Processing (Taipei, Taiwan, June 08 - 11, 2009). A. Hua and S. Chang, Eds. Lecture Notes In Computer Science, vol. 5574. Springer-Verlag, Berlin, Heidelberg, 327-337. DOI= http://dx.doi.org/10.1007/978-3-642-03095-6_32

STAMMINGER, M. AND DRETTAKIS, G. 2002. Perspective shadow maps. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '02). ACM, New York, NY, USA, 557-562. DOI=10.1145/566570.566616 http://doi.acm.org/10.1145/566570.566616

VALICH, T. 2008. *Intel converts ET: Quake Wars to ray tracing*. TG Daily. (June 12, 2008), Available online at http://www.tgdaily.com/html_tmp/content-view-37925-113.html .

VEACH, E. AND GUIBAS, L. J. 1997. Metropolis light transport. In Proceedings of the 24th Annual Conference on Computer Graphics and interactive Techniques International Conference on Computer Graphics and Interactive Techniques. ACM Press/Addison-Wesley Publishing Co., New York, NY, 65-76. DOI= http://doi.acm.org/10.1145/258734.258775

WERNECKE, J. 1994. The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, Release 2, Addison-Wesley, ISBN-13: 978-0201624953

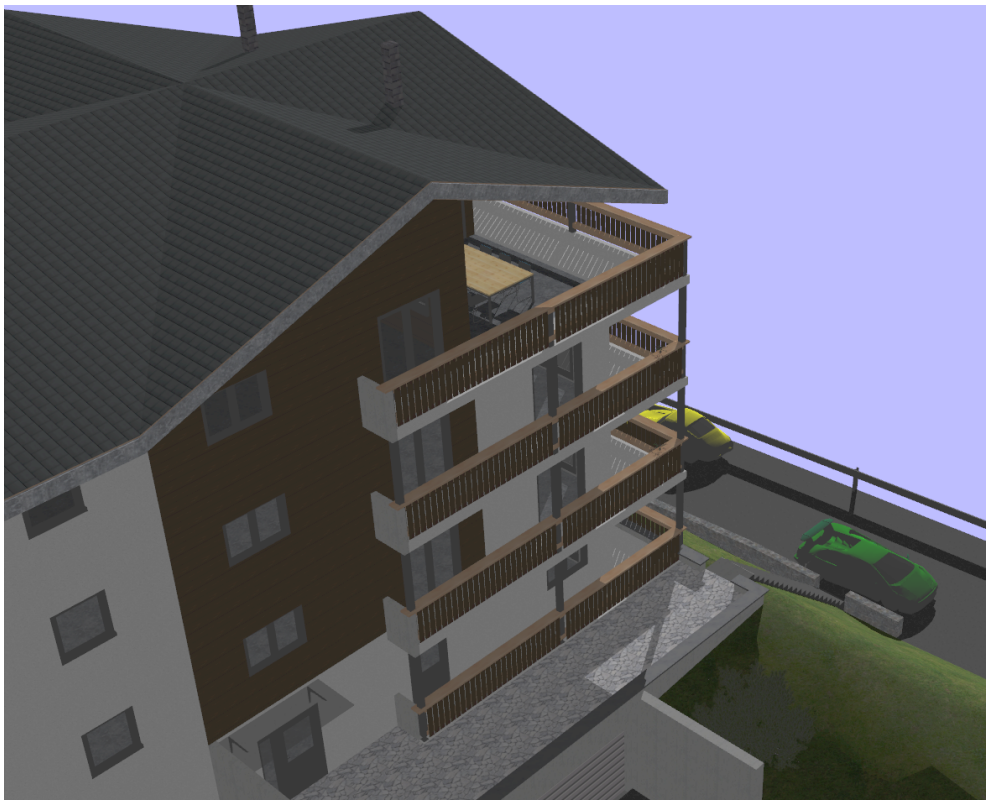
WHITTED, T. 1980. An improved illumination model for shaded display. *Commun. ACM* 23, 6 (Jun. 1980), 343-349. DOI= http://doi.acm.org/10.1145/358876.358882

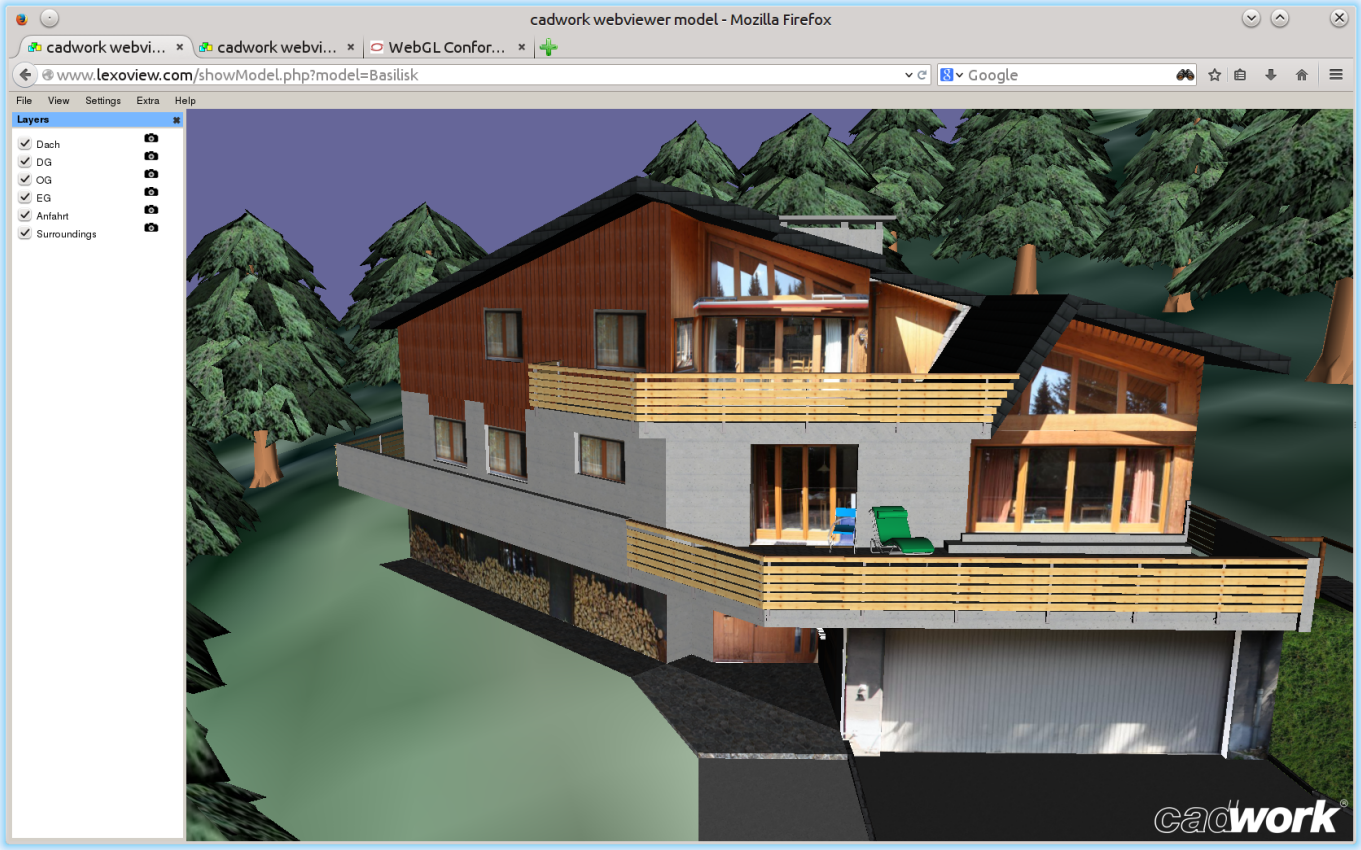
WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. SIGGRAPH Comput. Graph. 12, 3 (Aug. 1978), 270-274. DOI= http://doi.acm.org/10.1145/965139.807402

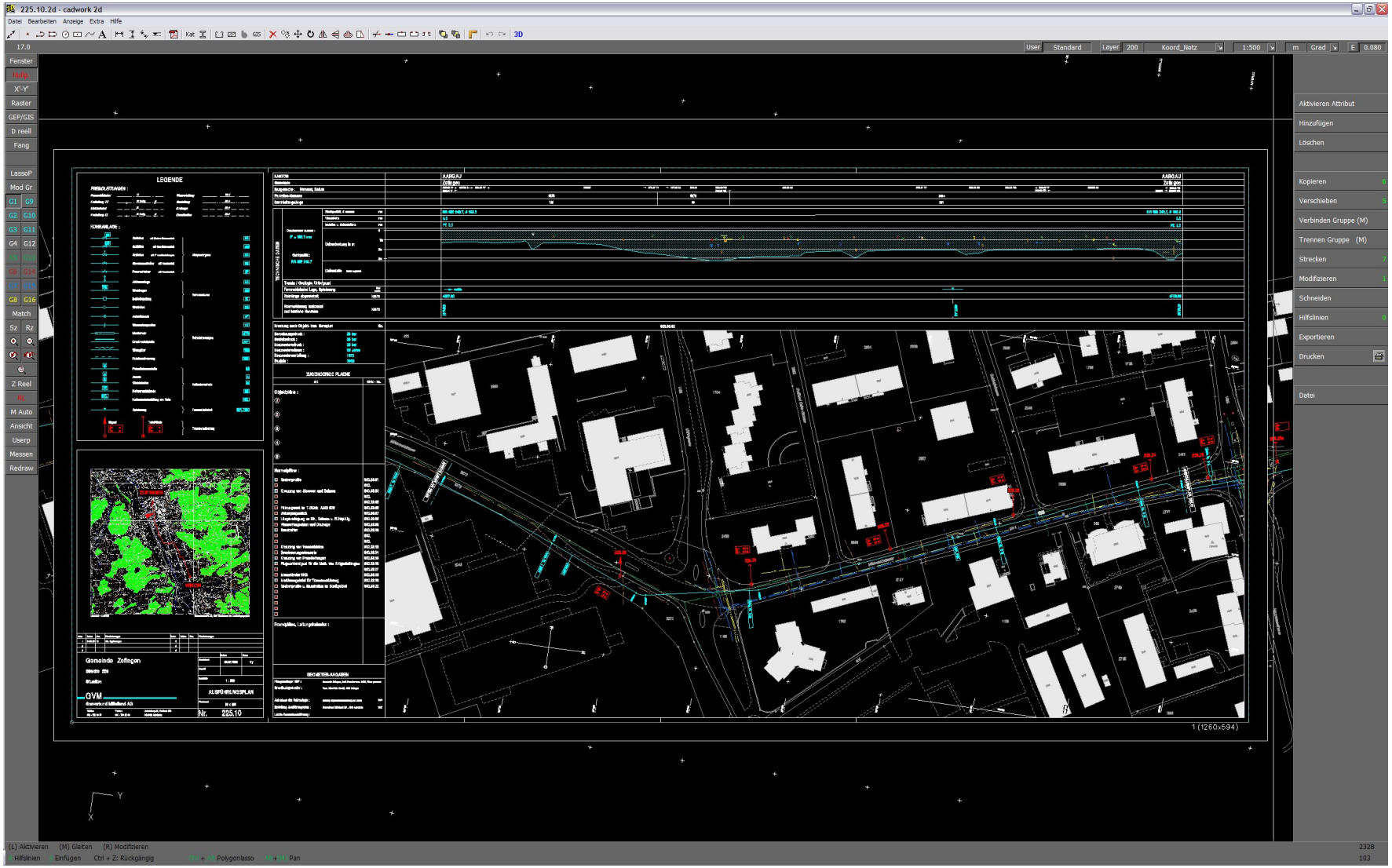
WIMMER, M., SCHERZER, D., PURGATHOFER, W. 2004. Light Space Perspective Shadow Maps, In Rendering Techniques 2004 (Proceedings Eurographics Symposium on Rendering), p. 143-151. June 2004.

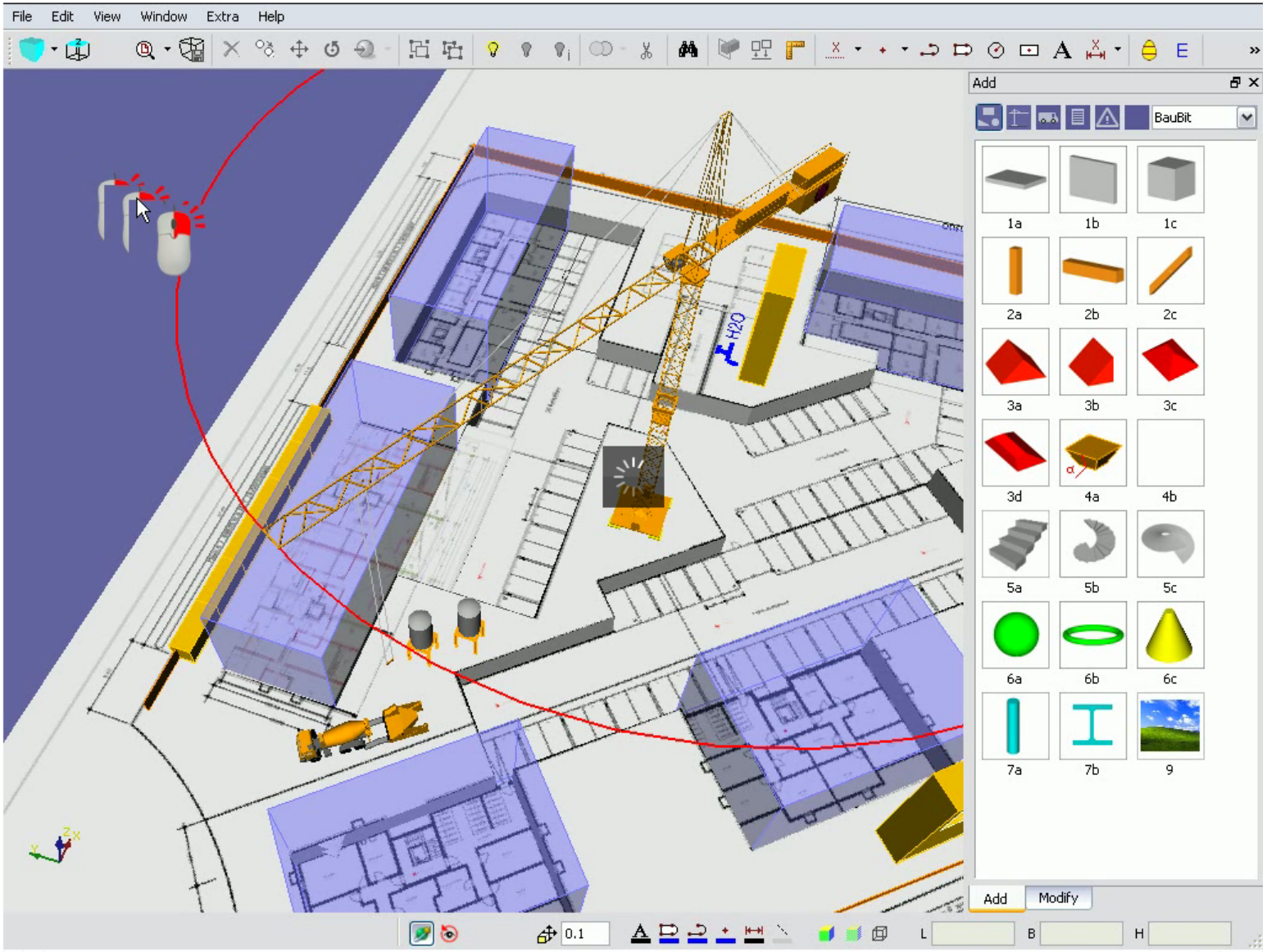
ZHANG, F., SUN, H., XU, L. AND LUN, K. L. 2006. Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications* (VRCIA '06). ACM, New York, NY, USA, 311-318. DOI=10.1145/1128923.1128975 http://doi.acm.org/10.1145/1128923.1128975

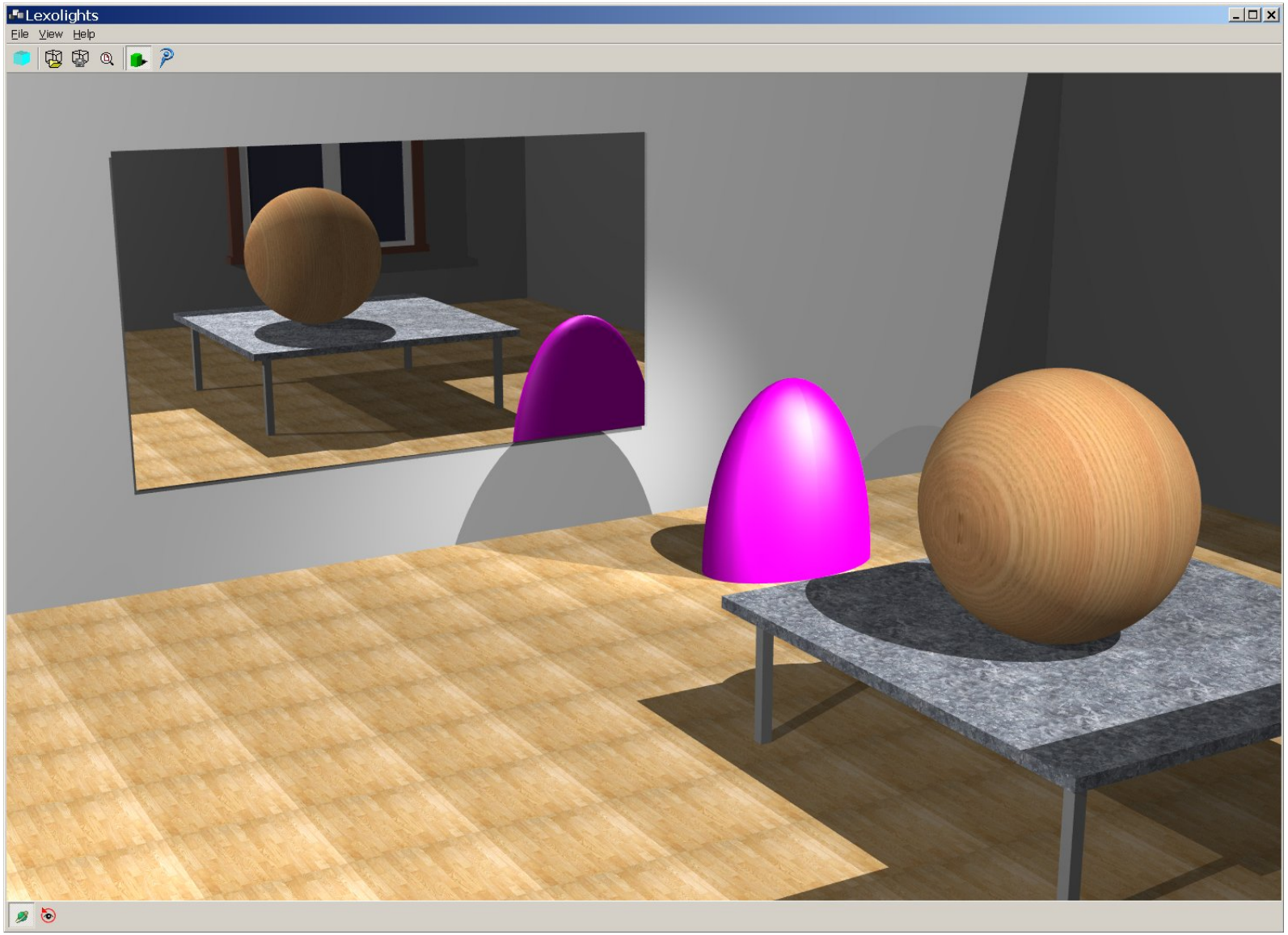
Image Gallery

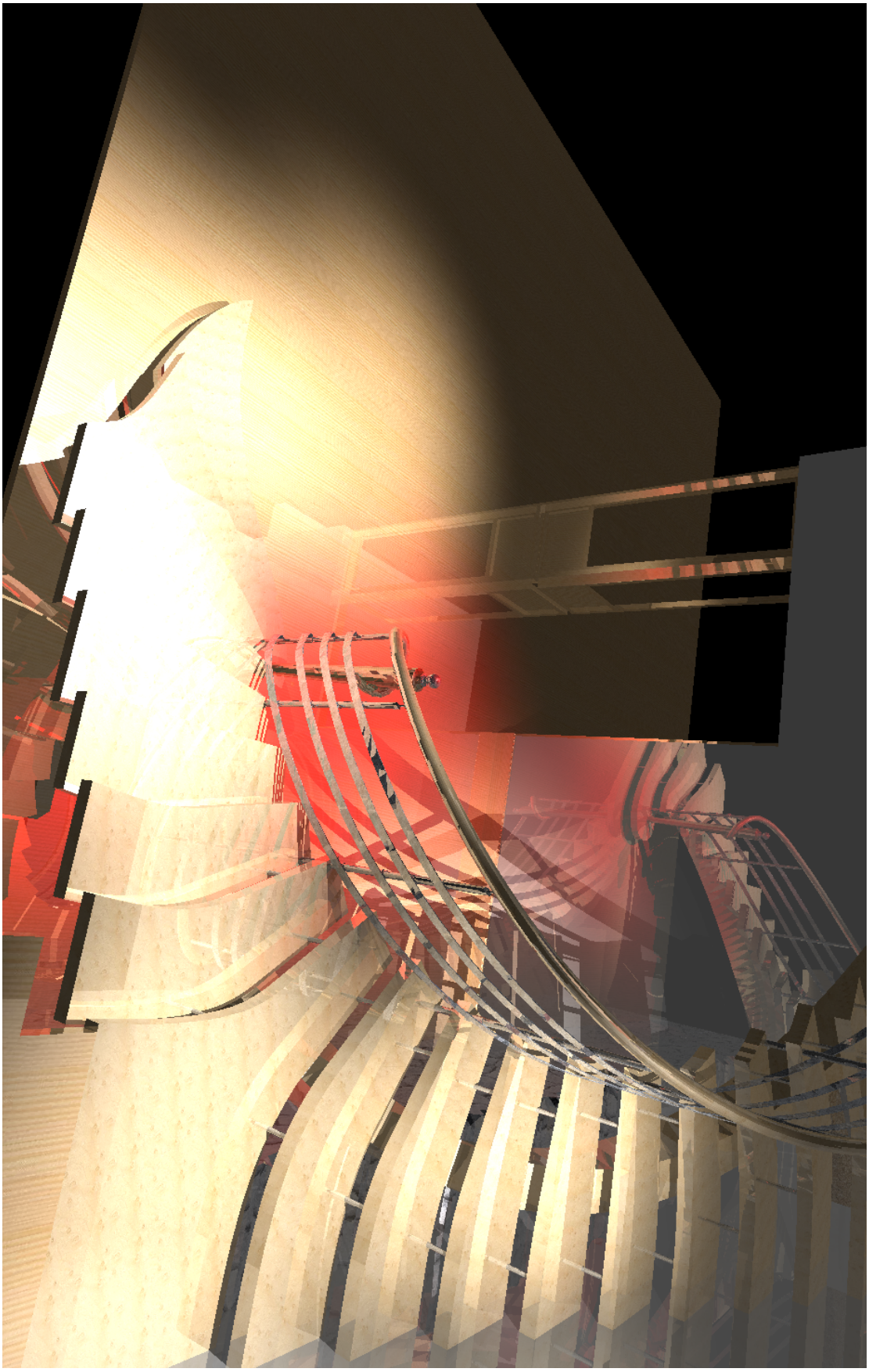


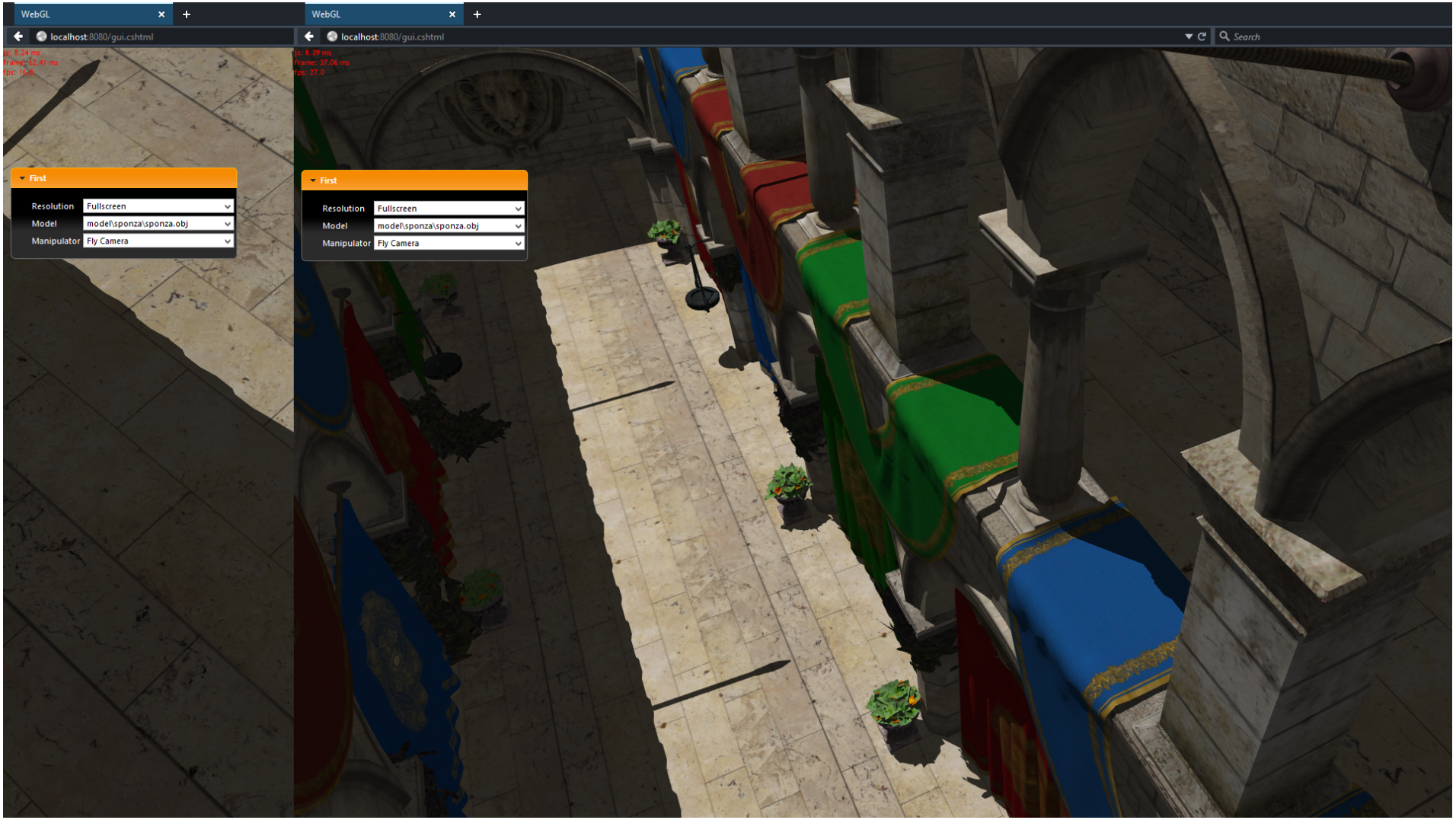

cadwork webviewer model - Mozilla Firefox
cadwork webvi...
cadwork webvi...
WebGL Confor...
www.lexoview.com/showModel.php?model=Basilisk
Google
File
View
Settings
Extra
Help
Layers
Dach
DG
OG
EG
Anfahrt
Surroundings
cadwork


225.10.2d - cadwork 2d
Datei Bearbeiten Anzeige Extra Hilfe
Fenster
Raster
Aktivieren Attribut
Hinzufügen
Löschen
Kopieren
Verschieben
Verbinden Gruppe (M)
Trennen Gruppe (M)
Strecken
Modifizieren
Schneiden
Hilfslinien
Exportieren
Drucken
Datei
LEGENDE
Gemeinde Zofingen
GVM
Nr. 225.10


File
Edit
View
Window
Extra
Help
Add
BauBit
1a
1b
1c
2a
2b
2c
3a
3b
3c
3d
4a
4b
5a
5b
5c
6a
6b
6c
7a
7b
9
H2O
Add
Modify
0.1
L
B
H


Lexolights
File View Help




WebGL
localhost:8080/gui.cshtml
First
Resolution
Fullscreen
Model
model\sponza\sponza.obj
Manipulator
Fly Camera
WebGL
localhost:8080/gui.cshtml
Search
First
Resolution
Fullscreen
Model
model\sponza\sponza.obj
Manipulator
Fly Camera