Realtime Shading of Folded Surfaces

B.Ganster R. Klein M. Sattler R. Sarlette

{**ganster, rk, sattler, sarlette**}**@cs.uni-bonn.de**

University of Bonn
Institute of Computer Science II
Computer Graphics
Römerstrasse 164
53117 Bonn, Germany

Abstract

In this paper we present a new, simple, and efficient way to illuminate folded surfaces with extended light sources in realtime including shadows. In a preprocessing step we compute the parts of the surrounding environment that are visible from a set of points on the surface and represent this information in a binary visibility map.
The illumination of the scene, i.e., incoming light from different directions, is encoded in an environment map. This way, extended light sources and complex illumination conditions of the surface can be simulated. The binary visibility information stored in the visibility maps is used during runtime to calculate the incoming and outgoing radiance in the direction of the viewer for each sample point. Various reflection models like the Phong or Lafortune model can be incorporated into these calculations. After computing the radiance values in each sample point, the surface is shaded using simple Gouraud interpolation.
Using the pre-computed visibility information, the whole shading and even the change of lighting conditions can be simulated in realtime by modifying the environment map. As an extension to the environment map we support additional point light sources whose parameters and positions can also be manipulated in realtime. Several examples of mesh illumination and shading are demonstrated.

keywords: folded surfaces, cloth shading, shadowing, illumination

1 Introduction & previous work

In this work we aim at real time shading of surfaces such as cloth, which possibly contain holes and complex folds, under realistic illumination conditions. The surface

is given as a triangle mesh. The appearance of a surface point is given by the radiance leaving the point in the direction of the viewer. According to the rendering equation [8, 12], this radiance is obtained by integrating the incoming radiance over all incoming directions at the vertex v. These directions are typically represented by a hemisphere, $H(v)$, centered around v's surface normal.

The problem of shading folded surfaces, especially cloth, was addressed by Stewart [1]. The main idea of his algorithm is a preprocessing step, which computes 3D *visibility cones* for each vertex point, which are used to determine the parts of the environment "seen" by this point. The cones are stored for each vertex and used to evaluate the direct primary irradiance at runtime by doing several intersection tests. The local illumination model described in [9] is used to calculate the resulting irradiance value.

Stewart reduces the calculation of the 3D visibility cones to a number of 2D visibility computations by slicing a polygonal mesh with parallel planes. If the illumination comes from a point light source, it is sufficient to test whether the point lies in the visibility cone. If a uniform diffuse area light source illuminates the surface, the area light source is intersected with the visibility cone and a contour integral around the boundary of the part of the source inside the cone yields the direct primary irradiance [4]. If the surface is illuminated by a uniform diffuse spherical source that surrounds the surface, a contour integral can be applied to the boundary of the visibility cone in the same manner as that of the area source. Although the results presented by Stewart are convincing, the necessary computation of intersection areas and the evaluation of the integrals prevent the use of complex shaped light sources and changing lighting conditions in real time. To overcome these problems we propose to use *binary visibility maps* instead of the Stewart's visibility cones. These binary visibility maps are computed in a preprocessing step as follows: we proceed by considering a finite set of directions on the hemisphere $H(v)$ belonging to a vertex v. For each such direction we determine whether the environment is visible or occluded by the surface, and accordingly set the binary value at the corresponding position in the visibility map. Following the work of Stewart [1] we neglect the incoming light from directions occluded by the surface itself. Only light from directions in which the outside environment is visible contributes to the radiance of a surface point. To discretize the directions we evaluated three different models: a hemicube, a single plane and a subdivision of the hemisphere into rectangles using spherical coordinates. This way, we can accurately handle all extended light sources whose projection onto the hemisphere, the hemicube or the single plane can be approximated with sufficient accuracy by the visibility map. To illuminate the surface we encode realistic lighting conditions in a global environment map. During the rendering process, the radiance values stored in this environment map are used to calculate the outgoing radiance in direction of the viewer for each vertex of the mesh. For the computation various reflectance models of the surface can be applied. The next section contains an overview over the algorithm. Section 3 deals with the preprocessing phase, explains the visibility tests and the methods and models that were used. The rendering part is explained in section 4. In section 5 benchmarks of the preprocessing step and the rendering are presented.

Example images can be found in the appendix.

2 Algorithm outline

This section gives a brief outline of our algorithm:

- Preprocessing
  - For each vertex v of the mesh we compute the visibility map by rendering the scene using v as eye point and the normal n of the mesh in v as viewing direction. Pixels of the visibility map not covered by the mesh encode their corresponding direction in the hemisphere.
  - The visibility maps are computed and stored for each vertex.
- At runtime
  - The radiance with respect to the observers' position is calculated for each vertex using a standard reflection model. The vertex visibility map is used to determine whether the radiance from a certain direction contributes to its radiance value.
  - The positions of point light sources are transformed into the coordinate system of the visibility map. It is subsequently decided whether the point light source contributes to the illumination of the vertex or not. This can easily be performed using the visibility map.
  - Finally, we assign the calculated radiance values to their vertices and perform a Gouraud interpolation.

3 Preprocessing

During the preprocessing phase we generate the visibility maps for each vertex of the mesh.

3.1 Computation of visibility maps

We compare three different ways to encode a visibility map: the hemicube [2], a single plane [3] and the hemisphere discretized into a rectangular grid. To obtain the visibility map in a vertex v the mesh is projected by a central projection with center v to one of these models (see figure 2). The hemicube, the hemisphere and the single plane are centered around v's surface normal n. The single plane is oriented perpendicularly to n. If the environment cannot be seen in a certain direction, the visibility of this direction in the visibility map is set to the RGB value $(0, 0, 0)$, otherwise the direction itself is encoded in an RGB value. As we restrict ourselves to a cube model of the environment map, the RGB value (x, y, n) is given by the x and y coordinate

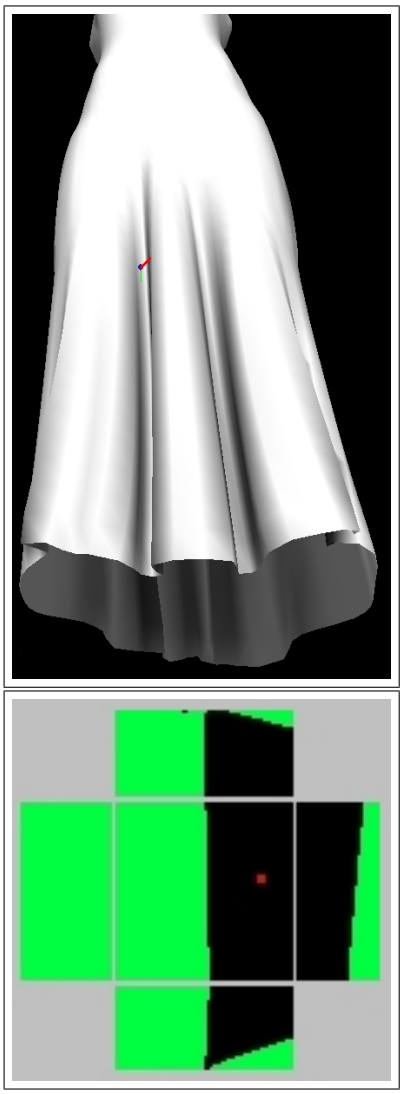

Figure No 1: **Visibility test**. The top image shows the mesh with a vertex (blue dot) inside a fold. The vertex is marked by its red normal. The bottom image shows the visibility map of this vertex. The model used for the visibility map is a hemicube with a resolution of 64×64 pixels for the top side of the cube and 64×32 for the sides. The figure shows the unfolded hemicube. For simplicity in this picture the directions in which the environment can be seen are coded by black and not by the direction itself. The directions where the environment cannot be seen are drawn in green. The red dot shows a projected point light source. In this example the light source can be seen by the specified vertex. Therefore the vertex is lit.

in the picture of the n-th side of the cube of the environment map. The numbering of the sides is as follows: The top face is numbered 0, the bottom face 5, the front face 1, the right face 2, the back face 3 and the left face 4.
Figure 1 shows a sample mesh and the corresponding visibility map for one vertex. A hemicube model is used with a resolution of 64×64 for the top and 64×32 for the sides. The top image shows the mesh with a vertex (blue dot) inside a fold. The vertex is marked by its red normal. The bottom image shows the unfolded hemicube for the above mentioned vertex. For simplicity in this picture the directions in which the environment can be seen are coded by black and not by the direction itself. The directions where the environment cannot be seen are drawn in green. The red dot shows a projected point light source. In this example the light source can be seen by the specified vertex. Therefore the vertex is lit.
The resolution of the visibility maps is $n \times m$. We experimented with values $n, m \in [4, 256]$. The resolution can be defined by the user. For good results we recommend $n \times m \geq 64$, as the possible resolution of the outgoing radiance in a vertex is limited by this number in case of diffuse lighting of the environment. For a resolution less than 64 this results in blotchy images. The central projection for all three models can be computed using standard ray tracing. For the hemicube and the single plane standard OpenGL rendering can be applied. However, using standard OpenGL rendering to project the mesh onto the sphere is complicated and involves specific hard to implement clipping steps. Details on this can be found in [10].
For a static mesh the visibility map is fixed and is stored together with the coordinates and the material parameters (reflectance parameters) of a point.

3.2 Comparison of Methods

Let **v** be the vertex of the mesh we determine the visibility map for. We evaluated the following methods for generating the visibility maps.

1. *OpenGL-Rendering*: We rendered the mesh using a triangle stripped display list of all triangles in the mesh.

2. *OpenGL-Rendering with triangle pre-selection*: In a first step, all triangles of the mesh are sorted into a three-dimensional grid. Then, triangle strips are generated for all triangles contained in a certain cell using a straightforward stripping algorithm. Finally, the triangle strips are stored in display lists. During the rendering of the mesh, we call the display lists of only those voxels that are within a given distance from the vertex **v**. This distance is given in units of the maximum dimension of the mesh. See table 1.

3. *Raytracer*: For every discretized direction of the visibility map we test all triangles of the mesh for an intersection with the ray leaving the point in that direction.

Method	Model	Resolution	Running time (sec)	Distance (ums)
Triangle Rasterizer	HC	8x8	28.4	2.0
	HC	16x16	30.0	
	SP	8x8	4.3	
	SP	16x16	5.0	
	HS	16x4	51.5	
	HS	32x8	54.7	
Triangle Rasterizer with triangle preselection	HC	8x8	15.2	1/6
	HC	16x16	17.0	
	SP	8x8	3.1	
	SP	16x16	3.7	
	HS	16x4	15.0	
	HS	32x8	20.6	
Raytracer with triangle preselection	HC	8x8	120.0	1/6
	HC	16x16	457.9	
	SP	8x8	32.2	
	SP	16x16	104.3	
	HS	16x4	33.3	
	HS	32x8	110.6	
Raytracer with grid traversal	HC	8x8	27.1	2.0
	HC	16x16	103.8	
	SP	8x8	9.3	
	SP	16x16	46.1	
	HS	16x4	7.2	
	HS	32x8	26.8	

Table No 1: Overview of the running times of the visibility calculation routines. Abbreviations used: HC= hemicube, SP= single plane, HS= hemisphere. UMS=units of mesh size.

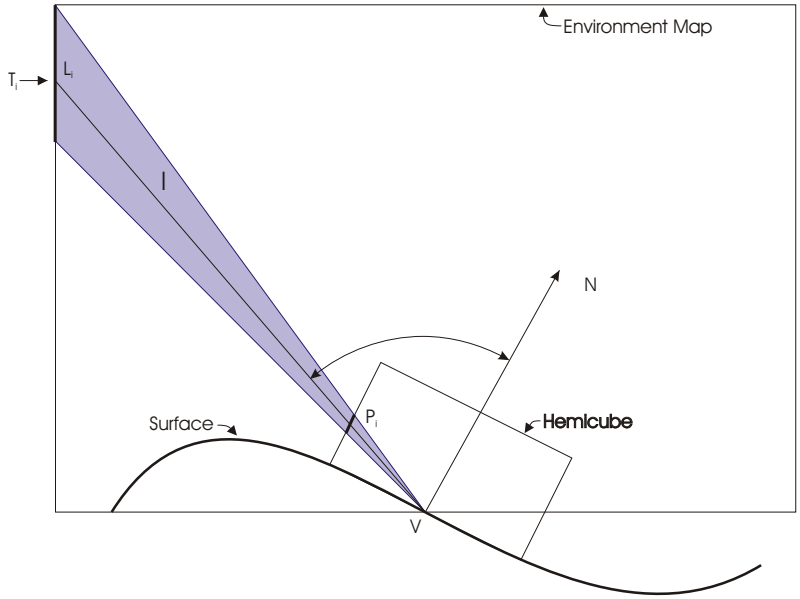

Figure No 2: The mapping between the *binary visibility* map of vertex v and the global environment map T_i is encoded in each pixel P_i of the visibility map itself. The environment patch T_i emits L_i in the direction I. θ is the angle between the surface normal N in v and the direction I of the incoming radiance. For simplicity, only a side view is shown.

4. *Raytracer with triangle preselection*: Similarly to OpenGL rendering with triangle pre-selection, we sort all triangles of the mesh into a three-dimensional grid and test only triangles in gris cells close to **v**.

5. *Raytracer with grid traversal*: We test only triangles lying in grid cells passed by the ray.

Table 1 summarizes the run times for all combinations of methods and models. The computations were performed on an Intel Celeron 800 MHz machine with a NVIDIA TNT2 graphics card. To evaluate the core speed of the visibility calculation routines, we do not include the rendering of the environment map into the measurements, i.e. we do not encode the directions. The table demonstrates quite clearly that OpenGL rendering with triangle preselection in singleplane mode and raytracer with grid traversal in hemisphere mode are the fastest techniques for the preprocessing step.

The singleplane model is efficient, however it has the disadvantage that the aperture angle must be less than 180 degrees, so only part of the half space is evaluated.

Therefore, the model may miss some small folds.

Using the Hemicube model is relatively slow compared to the singleplane model, because five pictures must be rendered for each point (the other models only need to render one image). On the other hand, it is more accurate than the single plane model and in contrast to the hemisphere it can be hardware-accelerated.

The table also shows that the OpenGL runtime only slightly depends on the resolution of the images that are generated. The situation is radically different in case of the raytracer, the double resolution needs twice as many rays.

4 Realtime Rendering

During the realtime rendering the outgoing radiances have to be computed for every vertex of the mesh.

4.1 Illuminating the surface using an environment map

As already mentioned, we restrict ourselves to cube maps. For the results the 24bit RGB pictures are generated by hand (see Appendix). For real applications they can be generated using high dynamic range images from the real world environments. The resolution of the environment map is adapted to the resolution of the visibility map in such a way that one texel in the environment map corresponds to approximately one pixel of our visibility map.

4.1.1 Calculating the outgoing radiance

The outgoing radiance in vertex v at the surface location x in direction of the viewer has to be computed. According to the rendering equation [3] the amount of incident light reflected towards the viewer has to be gathered. For this purpose, the incident radiance of each pixel is weighted by the Δ-form factor of the pixel itself and then used as incoming radiance of a reflection model. The Δ-form factors are derived from the rendering equation as following [12]:

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_S f_r(x, x' \rightarrow x, \vec{\omega}) L_i(x, x' \rightarrow x) V(x,x') G(x,x') dA' \quad (1)$$

with:
L_o: outgoing radiance [$Wm^{-2}sr^{-1}$]
L_e: emitted radiance [$Wm^{-2}sr^{-1}$]
f_r: BRDF [sr^{-1}]
L_i: incident radiance [$Wm^{-2}sr^{-1}$]
$\vec{\omega}'$: incidence direction
$\vec{n}$: normal at the surface location x
x': another surface location
$\vec{n}'$: normal at x'

dA': differential area at x'
$(x' \rightarrow x)$: radiance leaving x' in the direction towards x
S: set of all surface points

$$V(x,x') = \begin{cases} 1 & : \quad x \text{ and } x' \text{ are mutually visible} \\ 0 & : \quad \text{otherwise} \end{cases} \tag{2}$$

$$G(x,x') = \frac{(\vec{\omega}' \cdot \vec{n}')(\vec{\omega}' \cdot \vec{n})}{\|x' - x\|^2} \tag{3}$$

For the evaluation of the hemicubes we assume, that every surface in the model is Lambertian, so that the reflected radiance is constant in all directions. This reduces equation (8) to:

$$\begin{aligned} B(x) &= B_e(x) + \int_S f_{r,d}(x) B(x') V(x,x') G(x,x')\, dA' \\ &= B_e(x) + \frac{\rho_d(x)}{\pi} \int_S B(x') V(x,x') G(x,x')\, dA' \end{aligned} \tag{4}$$

B: radiosity (outgoing) $[Wm^{-2}]$
ρ_d: diffuse reflectance for a Lambertian surface ($\rho_d = \pi f_{r,d}(x)$)

Discretizing:

$$B_i = B_{e,i} + \rho_i \sum_{j=1}^{N} B_j F_{ij} \tag{5}$$

The form factor F_{ij} from differential area dA_i to differential area dA_j is

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{V(x,x') G(x,x')}{\pi}\, dA_j\, dA_i \tag{6}$$

Delta form factors for the hemicube pixels: (A hemicube pixel covers the area ΔA, the visibility information ist encoded into the pixels):

$$\Delta FF = \frac{G(x,x')}{\pi}\, \Delta A \tag{7}$$

For the hemicube model the Δ-form factors for top and side faces are computed analog to [11].
The contribution of one pixel in the visibility map of x is computed using the radiance stored in the corresponding texel of the environment map. This radiance is weighted by the Δ-form factor of the pixel and then used as input of a local illumination model, which is Lambertian reflection in our algorithm. Note that for real-world environment maps no Δ-form factor has to be applied to the outgoing radiance stored in the environment map, since it is already encoded in the corresponding real-world picture. Due

to the superposition property of light the total amount of radiance leaving the vertex in direction of the viewer is then easily obtained by summing the contributions of outgoing radiance of all pixels of the visibility map not marked as occluded. Note that the incident radiance corresponding to a pixel is taken from the environment map using the texel of the environment map encoded in the visibility map, see Figure 2. Note, that the Δ-form factors have to be computed only once and can be reused for every vertex. After calculating the radiance values for all vertices, the mesh can be rendered using a standard OpenGL-Renderer with Gouraud interpolation.
For the point light source any desired illumination model can be incorporated into the algorithm.

4.1.2 Dynamic environment maps

Due to the above calculations, our algorithm allows the dynamic modification of the illumination condition in realtime by using different cube maps. For example, if we want to rotate the surface in the environment, a rotated cube map is generated.

5 Results

Figure 3 and 5 show the static mesh of a cloth illuminated by virtual light encoded in hand made cubic environment maps (figures 4,6). The environment maps faces represent uniformly emitting light sources. In all pictures, the relative position of the model with respect to the environment map is fixed. In order to visualize the effect of the different lighting conditions, the whole scene including the environment map is rotated. We take snapshots from several viewing positions. Note how the radiance especially in the foldings descents from the illuminated to the dark side. Folds pointing towards the light source are fully illuminated. The situation is even more apparent if we use the faces of the environment map as colored light sources, see figure 6. Figure 8 shows a textured mesh of parts of the Grand Canyon based on satellite altitude data with 66049 vertices. In the left image pre-computed shadows calculated with a hemicube model are used. In the right image no illumination is used. The visual impression of the left images reveals a greater depth impression due to the self-shadowing in the faults.
Due to the vertex based shading it is necessary that the resolution of the triangle mesh provides sufficiently high fidelity. An additional point light source is used in figure 8 with the Utah teapot mesh. The spout casts a shadow on the pot (left image).
The images in figure 9 show a spaceship consisting of 18720 vertices. Self-shadowing is visible in the propulsion units, at the cockpit and under the wings.

6 Conclusions and Future Work

The algorithm presented is able of illuminating folded surfaces with extended light sources in realtime. The illumination conditions can be changed at runtime.
So far our meshes cannot be deformed in realtime. This would require a complete new set of visibility maps at every frame. At the moment, our preprocessing step needs at least about 3 seconds of runtime, using the hemiplane model, which is not enough to achieve interactive frame rates. A possible optimization could be to update only the parts of the mesh which have changed. Furthermore, it might be possible to further exploit graphics hardware acceleration for the preprocessing step. Future versions of our method should use high dynamic range images for the environment maps, which, at this state, consist only of RGB images. This will increase the realism of effects simulated by the environment illumination. Moreover, a local illumination model described by Stewart and Langer [9] can be applied to estimate secondary irradiance. The use of this model yields perceptually acceptable shading without resorting to an expensive global illumination step.

7 Acknowledgements

We would like to thank Markus Wacker from the University of Tuebingen for the mesh we used in our examples and our colleague Marcin Novotni for useful hints and corrections. Some of the used models were provided by www.3DCAFE.com.

References

[1] A. James Stewart. *Computing Visibility from Folded Surfaces*, Elsevier Preprint, 1999.

[2] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Fundamentals of Interactive Computer Graphics*, Addison Wesley, second edition. 1990.

[3] Micheal F. Cohen, John R. Wallace. *Radiosity and Realistic Image Syynthesis*, Morgan Kaufmann Publishers, Inc. 1993.

[4] Peter Shirley. *Realistic Ray Tracing*, A K Peters. 2000.

[5] T. Whitted. *An Improved Illumination Model for Shaded Display*, Communications of the ACM, vol.23, no. 6. 1980.

[6] F. C. Crow. *Shadow Algortihms for Computer Graphics*, SIGGRAPH 77. 1977.

[7] L. Williams. *Casting Curve Shadows on Curved Surfaces*, SIGGRAPH 78. 1978.

[8] J. T. Kajiya. *The rendering equation*, Computer Graphics, vol. 20, no.4. 1986.

[9] A. James Stewart. and M. S. Langer *Towards accurate recovery of shape from shading under diffuse lighting*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no.9, 1997.

[10] B. Ganster *Efficient cloth shading*, Diploma Thesis, University of Bonn, 2002.

[11] Michael F. Cohen and Donald P. Greenberg *The Hemi-Cube: A radiosity solution for complex environments*, SIGGRAPH 85 Proc., vol. 19, no.3, 1985.

[12] H.W. Jensen *Realistic Image Synthesis Using Photon Mapping*, A K Peters, 2001.

8 Appendix

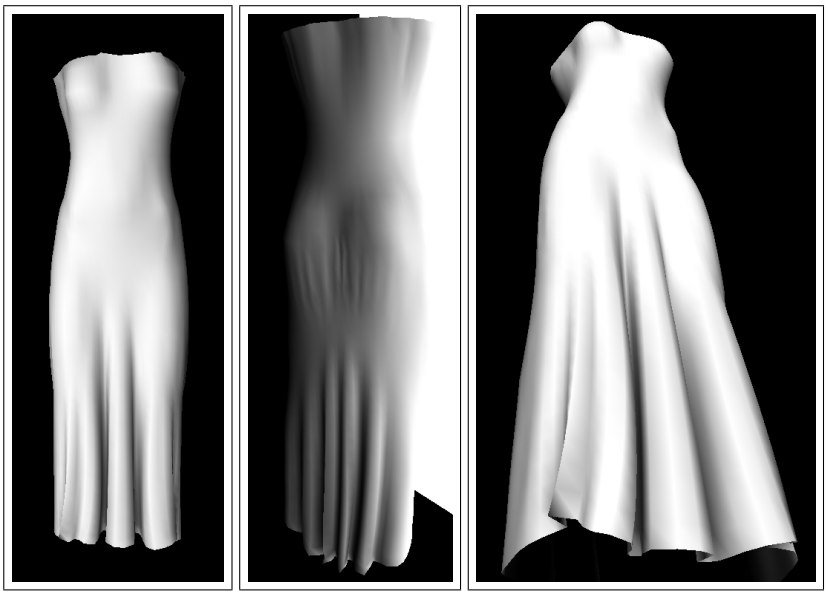

Figure No 3: These three images show a folded dress consisting of 3120 vertices. The left image shows a frontal view of the dress. Due to the pre-computed shadowing the folds in the lower part of the dress are clearly visible. In the center image, the back of the dress is shown, slightly rotated against the front side of the environment cube. The right image is rendered using another point of view, showing the folds in more detail.

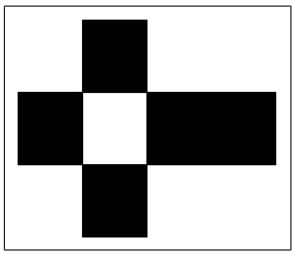

Figure No 4: Environment cube map used for the rendering of the above images, with only the front side white.

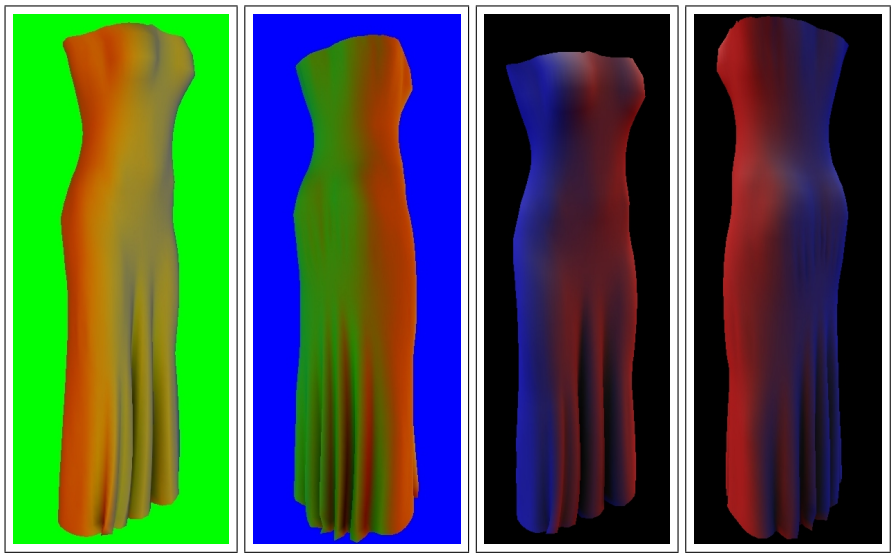

Figure No 5: These images show the effects of the usage of the environment faces as colored light sources. The mesh used, is the same as in figure 3. The corresponding environment maps are shown in figure 6. The reflection of the different light sources can be distinguished from each other in the folds.

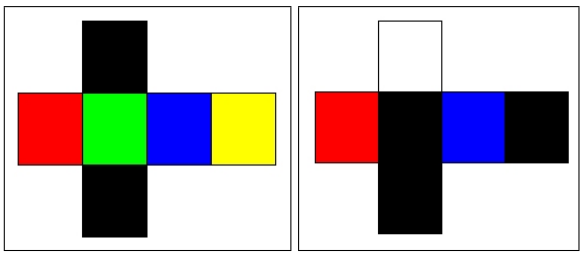

Figure No 6: Environment cube maps used for the rendering of the above images. For the two left images top and bottom are black. Sides are red, green, blue and yellow. For the two right images left and right are blue and red, the top is white and the rest is colored black.

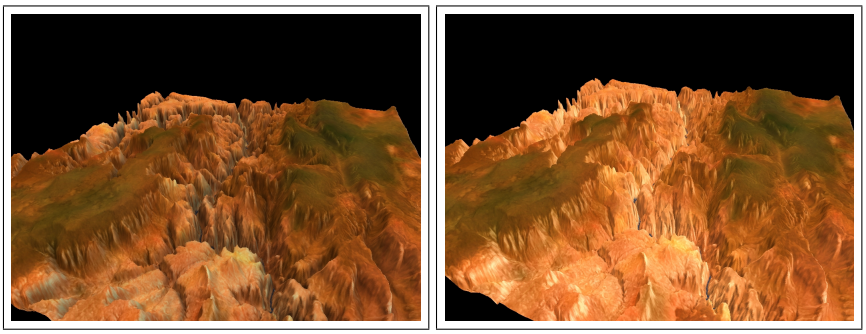

Figure No 7: These images show a textured mesh of parts of the Grand Canyon based on satellite altitude data with 66049 vertices. In the left image pre-computed shadows calculated with a hemicube model are used. In the right image no illumination is used. The visual impression of the left images reveals a greater depth impression due to the self-shadowing in the faults.

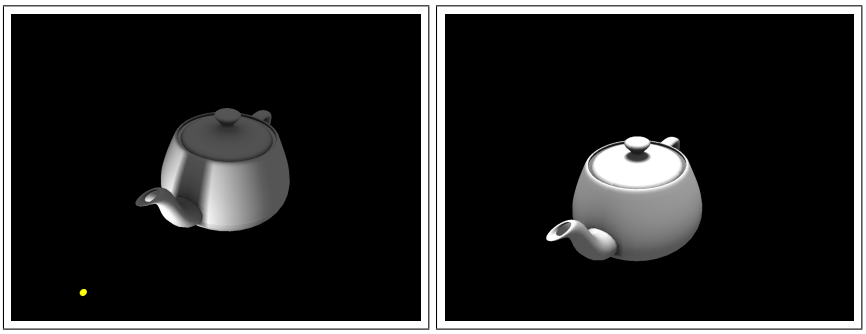

Figure No 8: The Utah teapot. The mesh consist out of 3907 vertices and is illuminated with a point light source (yellow dot) in front of the teapot (left image). The light source visibility is calculated on a per vertex basis, as described in the paper. The right image shows the teapot with the light source moved above it. Self shadowing is also visible in both images.

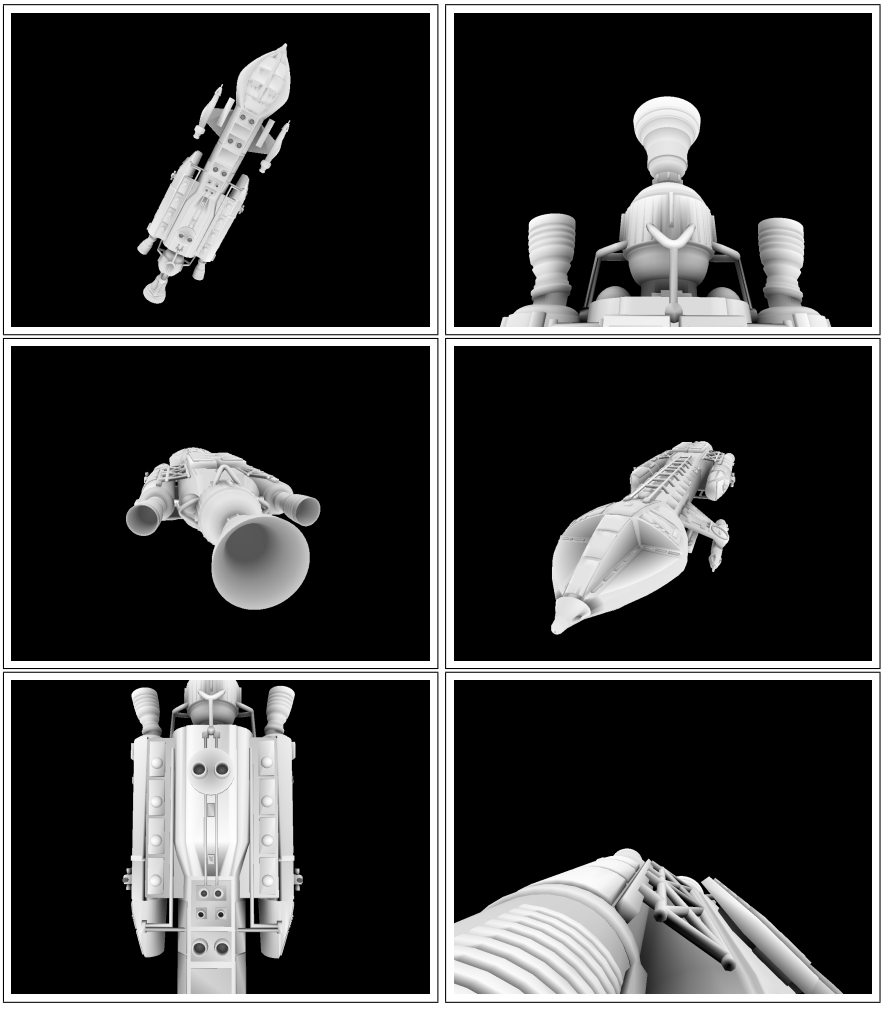

Figure No 9: These images show a spaceship model with 18720 vertices, provided by www.3DCAFE.com. A hemicube model with a resolution of 64×64 pixels for the top of the cube and no additional point light sources were used. Self-shadowing is visible in the propulsion units, at the cockpit and under the wings.