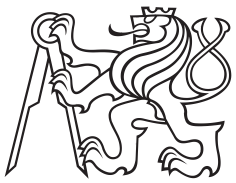
Master's thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Visibility of Triangulated Surface Illuminated by Flat Light Panel

Bc. Lukáš Koucký
Cybernetics and Robotics

10.5.2014
Supervisor: Ing. Vladimír Smutný

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Lukáš K o u c k ý

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: Visibility of Triangulated Surface Illuminated by Flat Light Panel

Guidelines:

1. Study triangulated surfaces, geometrical optics, geometry of cameras.
2. Design algorithms for determining visibility of triangulated glossy surface under prescribed illumination.
3. Implement and test the algorithms.
4. Document and evaluate all results.

Bibliography/Sources:

[1] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes: Computer Graphics-Principles and Practice, Addison-Wesley, 1990.
[2] Jiří Žára, Bedřich Beneš, Petr Felkel: Moderní počítačová grafika, Computer Press, Praha, 1998.
[3] Leila De Floriani, Paola Magillo: Visibility Algorithms on Triangulated Digital Terrain Models, Dipartimento di Informatica e Scienze dell'Informazione Universita di Genova. International Journal of Geographical Information Systems, Volume 8, Issue 1, 1994.
[4] Constantinos Perikleous Tsirogiannis: Analysis of Flow and Visibility on Triangulated Terrains. TU Eindhoven, 2011.

Diploma Thesis Supervisor: Ing. Vladimír Smutný

Valid until: the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Lukáš K o u c k ý

Studijní program: Kybernetika a robotika (magisterský)

Obor: Robotika

Název tématu: Viditelnost triangulovaného povrchu osvětleného plošným osvětlovačem

Pokyny pro vypracování:

1. Seznamte se s problémy triangulace povrchu, geometrickou optikou, geometrií kamer.
2. Navrhněte algoritmy pro určení viditelnosti a osvětlení jednotlivých trojúhelníků lesklého triangulovaného povrchu.
3. Algoritmy implementujte a vyzkoušejte.
4. Postupy a výsledky zdokumentujte a zhodnoťte.

Seznam odborné literatury:

[1] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes: Computer Graphics-Principles and Practice, Addison-Wesley, 1990.
[2] Jiří Žára, Bedřich Beneš, Petr Felkel: Moderní počítačová grafika, Computer Press, Praha, 1998.
[3] Leila De Floriani, Paola Magillo: Visibility Algorithms on Triangulated Digital Terrain Models, Dipartimento di Informatica e Scienze dell'Informazione Universita di Genova. International Journal of Geographical Information Systems, Volume 8, Issue 1, 1994.
[4] Constantinos Perikleous Tsirogiannis: Analysis of Flow and Visibility on Triangulated Terrains. TU Eindhoven, 2011.

Vedoucí diplomové práce: Ing. Vladimír Smutný

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Acknowledgement / Declaration

I would like to express my gratitude to my supervisor ing. Vladimír Smutný for his time, patient guidance and precious advices.

My biggest thanks goes to my family for their never ending support throughout whole study.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Lukáš Koucký
V Praze, 9. května 2014

Abstrakt / Abstract

Algoritmus předložený v této práci je navržen pro zjistění viditelnosti triangulovaného lesklého povrchu osvětleného plošným osvětlovačem. Algoritmus je schopen rozpoznat části zkoumaného povrchu viditelné kamerou a zpřesnit triangulaci. Tento proces nemení rozměry zkoumaného povrchu protože zpřesněná triangulace leží v té původní a činní jí tak detailnější a přesnější z pohledu kamery. Algoritmus nalezne části povrchu osvětlené zdrojem světla, které odrází obraz do kamery. Výstupem algoritmu je označkovaný triangulovaný povrch, narozdíl od tradičních algoritmů, jejichž výstupem je rasterový obraz.

Algoritmus byl otesován na 3D modelech s triangulovaným povrchem s rozličnými pozicemi a vlastnostmi kamer a osvětlení, kde prokázal svoji korektnost.

Klíčová slova: počítačové vidění, počítačová grafika, triangulovaný povrch, viditelnost, reflexe

Algorithm proposed in this thesis is designed for visibility determining of triangulated glossy surface illuminated by flat light source. Algorithm is able to recognize parts of examine surface visible by camera and refine triangulation. This process does not change dimensions of examine surface because refine triangulation lies in original one making it more detailed and more precise from camera perspective. Algorithm compute parts of surface illuminated by light source reflecting light to the camera. Output of the algorithm is labelled triangulation unlike traditional algorithms where the output is raster visibility image.

Algorithm was tested on 3D models with triangulated surface with various camera and light source positions and properties which proves algorithms correctness.

Keywords: computer vision, computer graphics, triangulated surface, visibility, reflection

/ Contents

Tables / Figures

Chapter 1
Introduction

Computing visibility is a fundamental problem in computer graphic. Rendering scene without dividing surface into parts that are visible and not visible for camera will end up in incorrect result. Two distinct goals in visibility determining are algorithm correctness and efficiency. For efficient visibility determining are used conservative algorithms. Their purpose is to distinct parts of scene that are likely visible from those that are definitely not visible. They are fast but does not guarantee accurate results. Correct algorithms on the other hand are responsible for exact visibility determining for given camera. Because conservative algorithms results can be obtained much more quickly they are commonly used to reduce amount of data later processed by exact algorithms and thus speed up the whole rendering process.

In this thesis is propose algorithm for determining parts of triangulated surface which reflects light from the light source to the camera. We call such situation when camera detect reflection on examine surface *visible reflection*. Algorithm uses both exact and conservative approaches. Main difference from common visibility algorithms is that the output is refined labelled triangulated surface compared to raster visibility image of standard approaches. Figure 1.1 shows difference between raster image and our approach. This opens opportunity to study surface visibility from more cameras at time and can be used, for example, to distribute cameras and light sources around the object so that every single triangle reflects light to at least one camera.

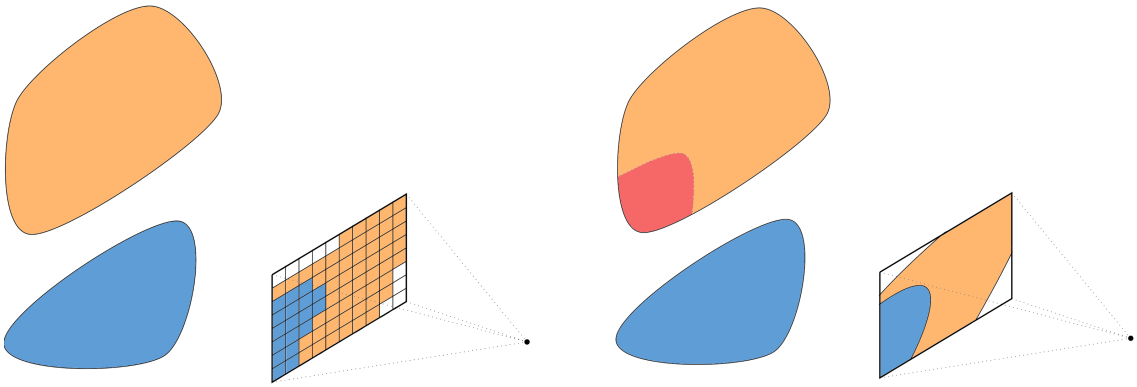


Figure 1.1. Difference between raster image and our approach. On the left is scene with two surfaces (orange and blue) and camera capturing raster image. On the right is the same scene, this time with proposed approach. Note the red area that is hidden for camera behind blue surface. Orange surface is divided into visible orange part and not visible red part.

1.1 Problem statement

The problem that algorithm designed in this thesis solves is following. Imagine scene with object with glossy mirror like surface S. The surface is surrounded with n cameras

$C_1 \dots C_n$ and m light sources $G_1 \dots G_m$. Algorithm computes visible reflection of every light source from every camera and the purpose is to cover whole examine surface with visible reflections. It is necessary to represent surface, cameras and light sources in form of mathematical model in order to process this problem with computer algorithm.

How to determine visibility by Geometrical optics is cover in Section 2.2. Camera is represented by pinhole camera model described in Section 2.3. The light source is represented as a flat surface that is pointed towards the examine surface. And finally the examine surface is represented by triangulated 3D model. Properties of triangulated model are described in Section 2.1.

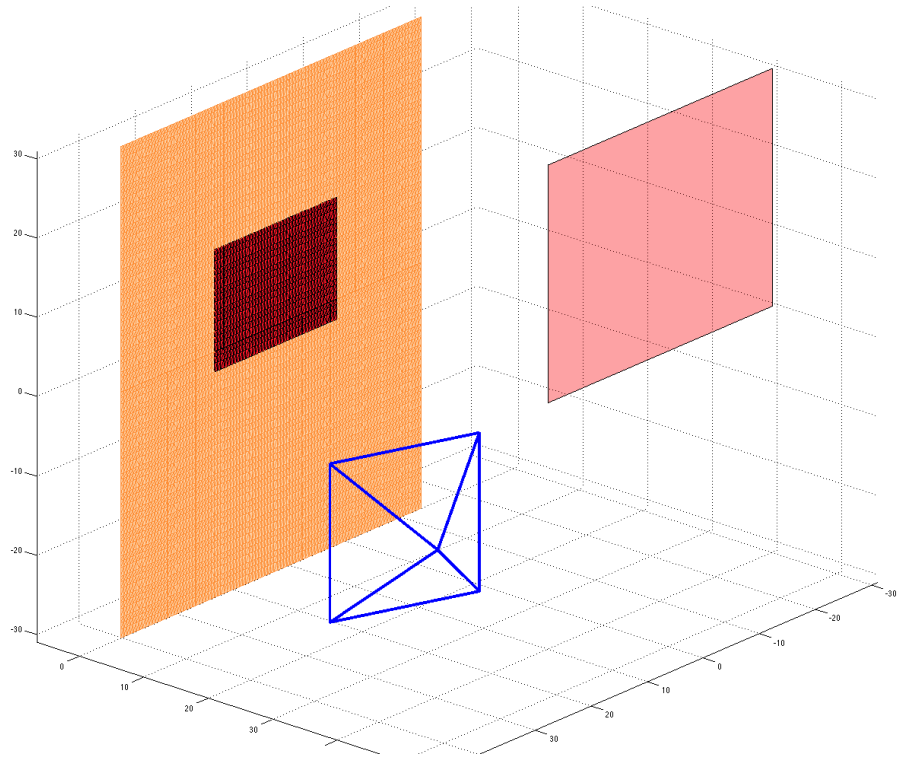


Figure 1.2. Scene with camera (blue), light source (light red) and examined surface (orange). Reflection visible by camera is marked red on examine surface.

For purposes of this thesis it is sufficient to design algorithm that examine surface with one camera and one light in a scene as shown in Figure 1.2. With this simplification is possible to cover whole surface with visible reflections by running the algorithm multiple time with different camera and light source parameters and positions and thus fulfil purpose of designed algorithm. Algorithm have to solve five main problems in order to correctly compute visibility reflection. They are:

- Select parts of examine surface facing towards the camera.
- Select parts of examine surface inside the field of view of the camera.
- Find parts of examine surface hidden behind another part of surface.
- Refine triangulation.
- Detects reflection of light source on visible parts of examine surface.

Solution to these five problem is discuses in Chapter 3.

1.2 Related work

Computer graphics knows many algorithms for visibility determining. Most common ones are raster algorithms with depth memory like *z-buffer* or *Depth-buffer*. Their principle is simple but powerful. For every pixel p in the image measure distance to each object that intersect straight line going from camera center through p. Colour of object that is closer to the camera is then used in final image for pixel p. Depth memory algorithm are widely use in real time scene rendering. Modification of depth memory algorithms is *depth line* algorithm which evaluates whole line at a time and thus offer compromise between memory usage and used time [1].

Because for large scenes could be memory and time expensive to compare every object in scene and look for the one that is closest to camera, computer graphics uses algorithms like *Backface culling*, that removes parts of objects facing away from camera. More about this algorithm can be found in Section 3.5.

List priority algorithms sometimes called *Painter's algorithm* [1] sorts scene by distance from camera and then paints closer objects over more distant ones. This is not very efficient but improvements like Binary Space Partitioning helps to efficiently calculate visibility.

Popular technique in space sorting is *Binary Space Partitioning* (BSP). BSP recursively partition space into convex subsets by hyperplanes. Binary Space Partition tree or *BSP tree* is the data structure used to represent partitioning. Root of the tree contains all the space and partition plane that divides space into two subsets. Whole scene is then divided into front facing and back facing convex areas [2].

Area subdivision algorithms subdivide image into four rectangle areas and then subdivides every rectangle into smaller ones until rectangles are at pixel size or each cover only one triangle. [3].

Hidden point removal purposed by Katz, Tal and Basri [4] works with point cloud without reconstructing the surface or estimating normals. They compute convex hull of whole object from given viewpoint and then removing hidden points. The proposed operator is however not completely accurate and relays on initial settings.

Chapter 2
Theoretical background

2.1 Surface triangulation

Triangulation is a word with many meanings. In finance triangulation stands for a strategy where trader exploits situation when three pairs of currencies are not balanced against each other [5]. In chess it is a move that forces opponent player to abandon a blockade and let the other player to move into his position [6]. In psychology it describes situation when one person communicates with another indirectly through third person [7]. And list of meanings goes on and on [8]. In this thesis meaning of triangulation is surface subdivision into collection of connected non-overlapping triangles [9]. This can be understood as approximation of curvilinear surfaces. The whole surface is covered with triangles

2.1.1 Triangle

Triangle is a basic geometry shape with three vertices and three edges. Each edge connects two different vertices. The only limitation for vertices is not to lie on straight line - the three vertices must not be collinear. Each triangle have three angles with sum 180 degrees. On Figure 2.1 are angles named α, β and γ. For reasons described later it is often beneficial to avoid triangles with small angles in triangulation. The circumscribed circle or circumcircle of a triangle is a unique circle through all vertices of triangle and it will play its role in Section 2.1.3.

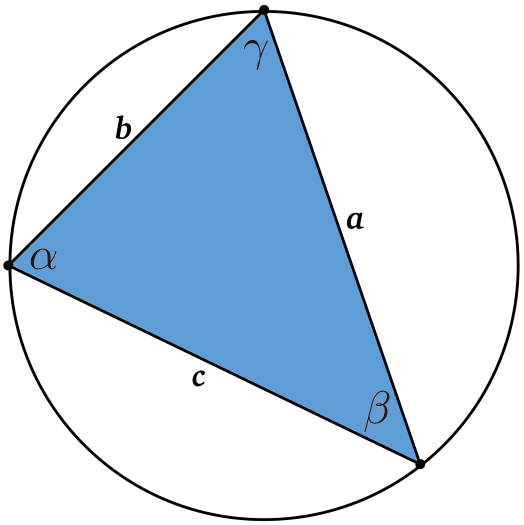


Figure 2.1. Triangle with circumcircle.

2.1.2 Triangulation

In general we can call any collection of triangles triangulation. However for practical reasons we are interested in triangulations that meets certain requirements. To describe those requirement several definitions need to be established. When constructing triangulation, we usually start with given collection of vertices

$$V = \{v_i\}, i = 1, \ldots, n,$$

and a domain Ω that contains all points in V. Figure 2.2 shows domain Ω with ten vertices and corresponding triangulation of these vertices. This example in 2D is still valid in 3D space.

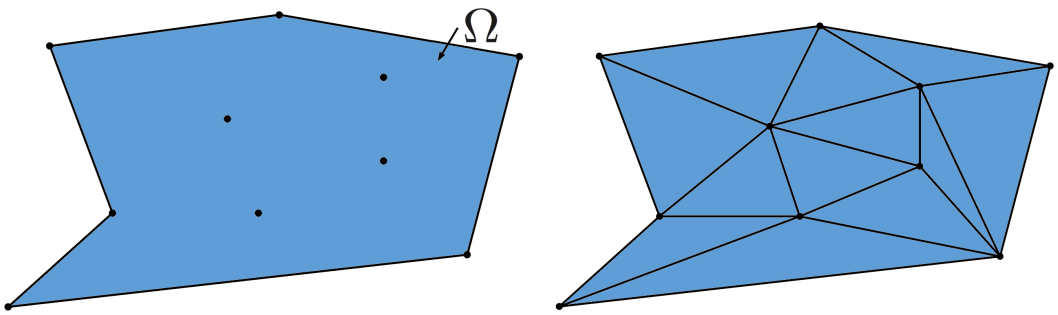


Figure 2.2. Domain Ω with ten vertices on the left and triangulation on the right.

Triangulation Δ is collection of triangles $t_{i,j,k}$ and triangle $t_{i,j,k}$ have vertices v_i, v_j and v_k. Triple (i, j, k) represents triangle $t_{i,j,k}$ so that set of triples I_Δ represents all the triangles in triangulation. Triple

$$t = (i, j, k) \in I_\Delta$$

for some integers i, j and k refers to triangle $t_{i,j,k}$ in triangulation Δ. Requirements for triangulation by [9] are following:

1. No triangle $t_{i,j,k}$ in a triangulation Δ is degenerated, that is, if $(i, j, k) \in I_\Delta$, then v_i, v_j and v_k are not collinear.
2. The interiors of any two triangles in Δ do not intersect, that is, if $(i, j, k) \in I_\Delta$ and $(\alpha, \beta, \gamma) \in I_\Delta$ then

$$Int(t_{i,j,k}) \cap Int(t_{\alpha,\beta,\gamma}) = \emptyset.$$

3. The boundaries of two triangles can only intersect at common edge or at common vertex.

2.1.3 Delaunay triangulation

Triangulation from collection of vertices can be achieved by various algorithms. They are usually focussing on creating optimal triangulation in a sense to avoid "poorly shaped" triangles. Such triangle is elongated or almost degenerated. When we want to avoid poorly shaped triangles we may say that triangulation is "good" if it consists of triangles that are close to being equiangular. More specifically, if we compare all possible triangulations constructed from the same collection of vertices, we may prefer

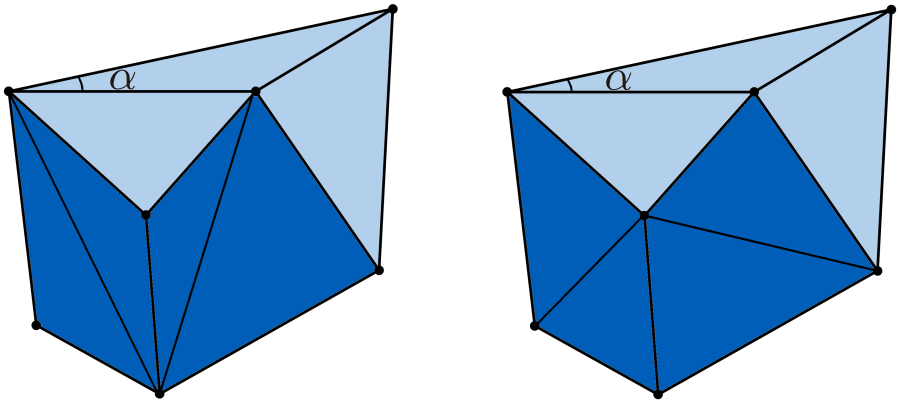


Figure 2.3. Two triangulations of the same point set. Triangulation on the right is Delaunay triangulation.

one that has triangle with largest minimal angle. Such criterion is known as *MaxMin* angle criterion [9].

Figure 2.3 shows two different triangulations of the same point set. Triangulation Δ_a is on the left and Δ_b on the right side. Only difference is in swapping two of the edges that effects area in dark blue color. Both triangulations satisfy MaxMin angle criterion because minimal angle is in both α. But only Δ_b is Delaunay triangulation, this is determine by the Circle Criterion.

Circle Criterion: *A Delaunay triangulation Δ of a set of vertices V in plane is a triangulation, where the interior the circumcircle of any triangle in Δ contains no point from T* [9].

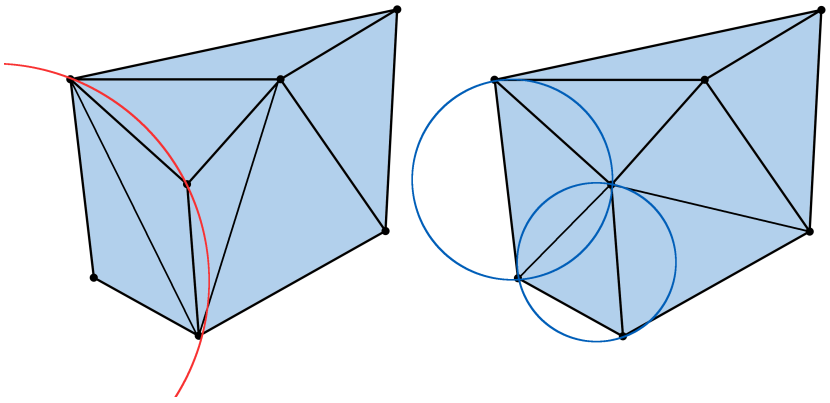


Figure 2.4. Two triangulations of the same point set with circumcircles.

Figure 2.4 shows the same example of triangulations as Figure 2.3. This time there are three circumcircle highlighted. Red one on the left (only part of this circumcircle is visible on Figure) shows why this is not Delaunay triangulation. Inside the red circumcircle is another vertex which violates the Circle Criterion. Blue circumcircles on the right satisfies this criterion. Circle Criterion can be generalized on Delaunay triangulation in 3D by considering circumscribed spheres.

2.2 Geometrical optics

Physics describes light as electromagnetic radiation. When we study light in environment with dimensions much larger than lights wave length we can use *Geometrical optics* as a tool for light propagation description. Geometrical optics or ray optics describes light propagation in form of line that is known as *ray*. Ray is idealize model of light that points in direction of energy flow [10].

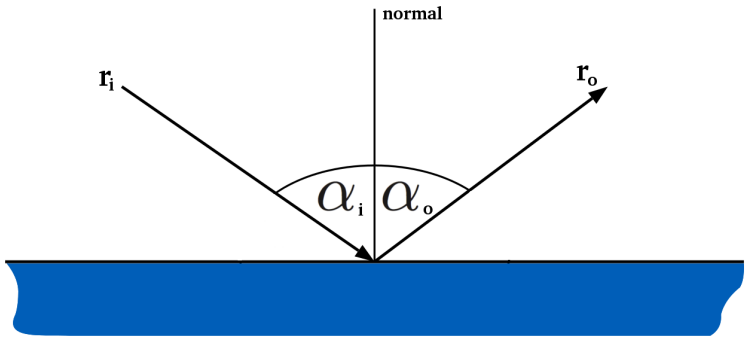


Figure 2.5. Reflection on glossy surface. Incident ray r_i makes with normal angle α_i which is same as α_o that makes reflected ray r_o with normal.

Figure 2.5 shows reflection of light on glossy surface described by geometrical optics. Incident ray r_i makes with normal same angle as reflected ray r_o. Such reflection is called *specular reflection* [10].

Ray can serve as a good instrument to define visibility. We can say that point A is *visible* from point B if a ray can travel from point A to point B without colliding with another object.

Figure 2.6. Visibility described by ray. On the left is point A visible from B because ray r can pass from A to B without colliding with another object. On the right is point A not visible from B because before reaching A, ray r hits obstacle.

2.3 Camera geometry

A camera could be understood as mapping between the 3D world points an a 2D image points. This could be represented by a 3×4 matrix. This matrix is called *camera projection matrix* and usually is labelled P. One of the simplest and most commonly used camera models is pinhole camera model [11].

2.3.1 Pinhole camera model

Pinhole camera projects 3D world points on *image plane* (see Figure 2.7). Center of projection is *camera center* C. A point in space with coordinates $X = (X, Y, Z)^T$ is projected on as point x on the image plane where a ray connecting point X with camera center C meets the image plane.

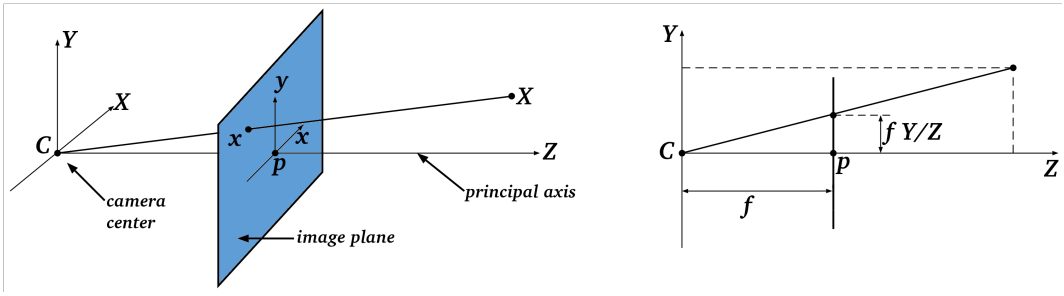


Figure 2.7. Pinhole camera geometry. Ray connecting point X and camera center C maps point X where the ray meets image plane.

Camera model is represented by *camera projection matrix* P. Using homogeneous coordinates, projection matrix maps points in space $X = (X, Y, Z, 1)^T$ to image points $x = (x, y, 1)$ by

$$x = PX \tag{1}$$

We can decompose camera matrix to

$$P = K[R|t] \tag{2}$$

Where K is *calibration matrix*, R is *rotation matrix* representing the orientation of the camera coordinate frame, $t = -RC$ and C is *camera center* [12].

Chapter 3
Algorithm design

The purpose of a proposed algorithm is determining visibility of triangulated glossy surface and reflection of illuminating light source. Input is a model with triangulated surface, camera pointed at the model and finally a light source illuminating the model.

- **Light source** - As a light source is considered flat light panel. Real world example of such light source can be monitor or display. Since surface of examine model is glossy or mirror like, it is capable to reflect not only active light source that emits light, but also a *passive source*. As a passive source that does not radiates light can serve any picture or object which reflected image will be visible on examine surface by camera.
- **Examined surface** - is represented by 3D model with triangulated surface. Such model is formed from vertices and triangles. Triangulated is described in detail in Section 2.1.
- **Camera model** - Pinhole camera model described in Section 2.3.1 is used as a camera representation.

All items are described more detailed in following sections.

3.1 Light source

Light source G is represented by coordinates of corners. In this thesis was used rectangle light source ((G_1, G_2, G_3, G_4) on Figure 3.1) but algorithm will work with any polyhedron. Light source does not carry any information about which direction is facing, it is however assume that since the goal is to detect reflection, light source is facing towards examine surface.

3.2 Examined model

As mention before, examine surface is 3D model with triangulated surface. Such surface consists of *vertices* and *triangles*. Every vertex v is described by two vectors. First vector called *position* holds 3D coordinates of v. Second vector *normal* describes normal of vertex v. Normal of vertex v is given by the mean of normals of all triangles that are formed by v. Both normals and vertices positions are stored in separated arrays of the same length. Triangle is represented by a triple of integers, where each integer is an index to the array of vertices.

Vertices and triangles are stored in two structures. Format of structures is shown in Tables 3.1 and 3.2.

Structure of vertices contains five array of same length, in this example n elements. First three arrays contains coordinates. Position stores coordinates of vertices in space, position2D stores coordinates on image plane and normal stores vertices normals. The remaining two arrays, visible and reflection, are logical arrays. They are initially set to zeros. If the vertex is determine to be visible, its position in visible array is set to 1.

position	$\begin{pmatrix} x_1, & x_2, & \cdots & x_n \\ y_1, & y_2, & \cdots & y_n \\ z_1, & z_2, & \cdots & z_n \end{pmatrix}$
normal	$\begin{pmatrix} x_{n1}, & x_{n2}, & \cdots & x_{nn} \\ y_{n1}, & y_{n2}, & \cdots & y_{nn} \\ z_{n1}, & z_{n2}, & \cdots & z_{nn} \end{pmatrix}$
position2D	$\begin{pmatrix} x_1, & x_2, & \cdots & x_n \\ y_1, & y_2, & \cdots & y_n \end{pmatrix}$
visible	$(v_1, \quad v_2, \quad \cdots \quad v_n)$
reflection	$(v_1, \quad v_2, \quad \cdots \quad v_n)$

Table 3.1. Structure of vertices.

vertices	$\begin{pmatrix} v_{11}, & v_{12}, & \cdots & v_{1m} \\ v_{21}, & v_{22}, & \cdots & v_{2m} \\ v_{31}, & v_{32}, & \cdots & v_{3m} \end{pmatrix}$
normal	$\begin{pmatrix} x_{n1}, & x_{n2}, & \cdots & x_{nm} \\ y_{n1}, & y_{n2}, & \cdots & y_{nm} \\ z_{n1}, & z_{n2}, & \cdots & z_{nm} \end{pmatrix}$
visible	$(t_1, \quad t_2, \quad \cdots \quad t_m)$
partly_visible	$(t_1, \quad t_2, \quad \cdots \quad t_m)$
reflection	$(t_1, \quad t_2, \quad \cdots \quad t_m)$

Table 3.2. Structure of triangles.

Same principle is with array reflection, is concrete vertex reflect light from light source to camera.

Structure of triangles is based on the same principles as previous structure, with length of m elements. Array vertices stores id's of vertices that forms concrete triangle. Meaning that integers store in this array are direct reference to arrays in structure of vertices. Structure normal contains triangles normal, it determine as mean of its vertices normals. Last three arrays are logical in a same sense as in structure of vertices. Visible determines that whole triangle is visible from camera perspective. If only one or two of triangles vertices are visible, partly_visiblee is set to 1. Reflection is set to 1 if all three vertices of triangle projects reflection.

3.3 Camera model

Last input for algorithm is a camera model. Algorithm uses pinhole camera model represented by *camera projection matrix* P described in Section 2.3. Matrix P is used for mapping points from 3D world coordinates to 2D image coordinates. If the world coordinates X are homogeneous coordinates then homogeneous 2D image coordinates x can be computed as $x = PX$ described in Section 2.3.1.

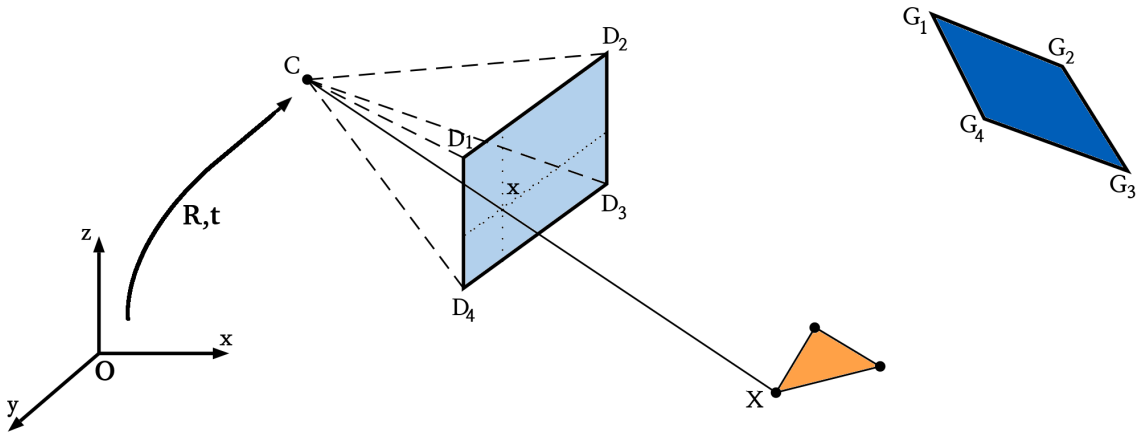


Figure 3.1. Description of camera

3.4 Structure of algorithm

Algorithm consists of five parts. Each part is described in detail in following sections. Structure of algorithm is described in Algorithm 3.1 below.

Algorithm 3.1	Main algorithm
input:	camera and light source properties 3D model with triangulated surface
1.	Find triangles facing away from camera (*Backface culling*).
2.	Find triangles in cameras field of view and refine triangles partly visible.
3.	Find hidden triangles.
4.	Retriangulate triangles that are labelled as partly visible in step 3.
5.	Find visible triangles that reflect light from light source into camera.

3.5 Backface culling

Backface culling selects triangles facing towards camera and labels them as visible. Since every vertex have its normal, which expresses which direction is vertex facing, finding triangles that are facing towards camera is done by method called Backface Culling [13]. Let v_i be the any vertex of model and v_n its normal. This defines plane that passes through v_i and has normal v_n. Camera can see vertex if it is on positive side of that plane. Camera can be represented as camera center point C. If vector from v_i to C forms with v_n angle smaller than 90° than v_i is visible. This describes Equation (1).

$$(C - v_i) \cdot v_n > 0 \tag{1}$$

Algorithm 3.2 Backface culling

input: structure of vertices V
structure of triangles T
camera structure *camera*

1. ***for each*** vertex v_i of V and its normal v_{ni}
2. ***if*** $(C - v_i) \cdot v_{ni} > 0$ ***then***
3. $v_{visible\,i} = 1$
4. ***endif***
5. ***endfor***
6. ***for each*** triangle t_i of T
7. ***if*** $v_{1visible} \& v_{2visible} \& v_{3visible} == 1$ ***then***
8. $t_{visible\,i} = 1$
9. ***endif***
10. ***endfor***

When all three vertices of triangle fulfils this condition, triangle is front facing camera and it is labelled as visible. But condition like this is not sufficient to completely decide about visibility. Figure 3.2 shows two examples of such insufficiency. On Figure 3.2 is camera pointed at object with eight faces. Blue and orange faces are marked as visible by backface culling, red ones are marked as not visible. Triangle can be facing camera but another part of model can cover it, like face F and D highlighted orange. Face F is in fact invisible from camera perspective because it is covered by A and H. Face D is even more problematic. Part of it is covered by face H and part is out of field of view, just like face B. Face E is completely out of field of view and thus not visible for camera. Following parts of algorithm covers these problems.

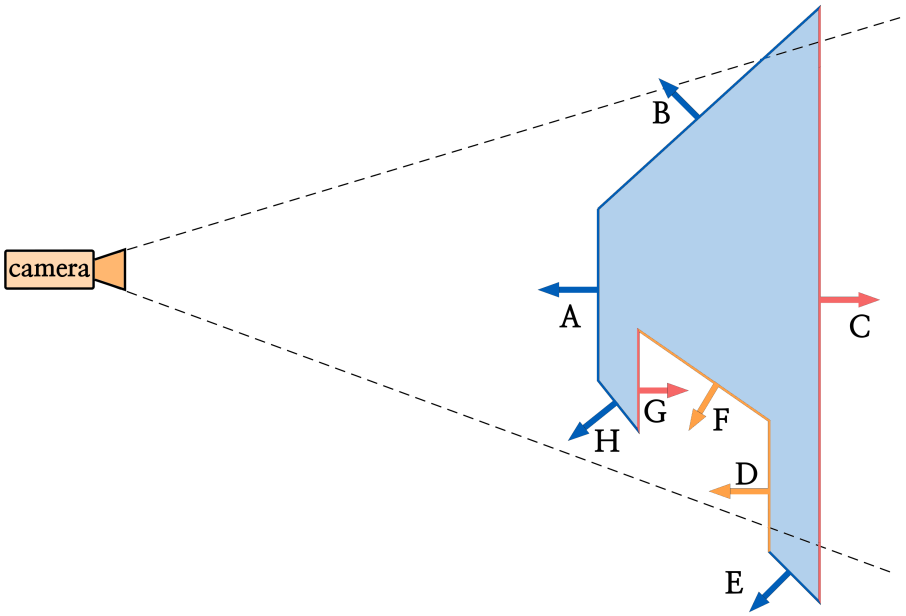


Figure 3.2. Backface culling.

3.6 Triangles in field of view

Backface culling removes triangles that are facing away from camera. But triangles facing camera out of field of its view are marked as visible and this is not correct. To

find such triangles is task of this part of algorithm. This process is in literature called clipping [13, 2].

At first is determine which vertices are in field of view and which are not. The field of view can be imagine as a pyramid with apex in camera center C and four sided base given by the image plane, see Figure 3.3 where is pyramid formed from C, D_1, D_2, D_3 and D_4. To determine whether is vertex inside pyramid and thus inside field of view are at first computed four planes that creates sides of pyramid. Each plane is defined by three points, they are $p_{c12} = (C, D_1, D_2)$ for first plane, $p_{c23} = (C, D_2, D_3)$ for second, $p_{c34} = (C, D_3, D_4)$ for third and $p_{c41} = (C, D_4, D_1)$ for last plane. Every plane has its normal n_{cxy}, they are

$$\begin{aligned} n_{c12} &= ((D_1 - C) \times (D_2 - C)) \\ n_{c23} &= ((D_2 - C) \times (D_3 - C)) \\ n_{c34} &= ((D_3 - C) \times (D_4 - C)) \\ n_{c41} &= ((D_4 - C) \times (D_1 - C)) \end{aligned} \tag{2}$$

With description of corner points of image plane D_1, D_2, D_3 and D_4 clockwise when looking towards camera, normals $n_{c12}, n_{c23}, n_{c34}$ and n_{c41} are pointing away from each other. Any point p is on the plane p_{cxy} if $(p - C) \cdot n_{cxy} = 0$. Point is above the plane if $(p - C) \cdot n_{cxy} > 0$ and under plane if $(p - C) \cdot n_{cxy} < 0$. Because normals of all planes on surface of pyramid points away from each other, if any point is under all four planes we can say that such point is inside pyramid. Pyramid in this case represents field of view, that is why every vertex v is inside field of view if it satisfy all four following conditions:

$$\begin{aligned} (v - C) \cdot n_{c12} &< 0 \\ (v - C) \cdot n_{c23} &< 0 \\ (v - C) \cdot n_{c34} &< 0 \\ (v - C) \cdot n_{c41} &< 0 \end{aligned} \tag{3}$$

After computing these conditions for all visible points, array *visible* in vertices structure is updated. Then are processed whole triangles. If all three points of triangle are not visible, triangle is not visible. If one ore two vertices are visible, only part of triangle is in the field of view. This situation is illustrated on Figure 3.3. Three triangles created by vertices $(v_8, v_7, v_6), (v_8, v_6, v_5)$ and (v_6, v_5, v_4) are completely in field of view. Triangle formed with vertices (v_1, v_2, v_3) is completely out of field of view and thus not visible for camera. But triangles created by vertices (v_5, v_4, v_3) and (v_4, v_3, v_2) are partly visible. Intersections between edges of triangles and plane p_{c23} are red points i_1, i_2 and i_3.

For every partly visible triangle are computed intersections of all edges containing not visible vertices with all four planes. As mention before point p is on a plane p_{cxy} if $(p - C) \cdot n_{cxy} = 0$. Let edge be a line segment from vertex v_1 to vertex v_2. Such line segment can be described by equation

$$p = v_1 + d(v_2 - v_1), d \in \leq 0, 1 \geq \tag{4}$$

Intersection between line segment and plane is in point p when

$$(v_1 + d(v_2 - v_1) - C) \cdot n_{cxy} = 0 \tag{5}$$

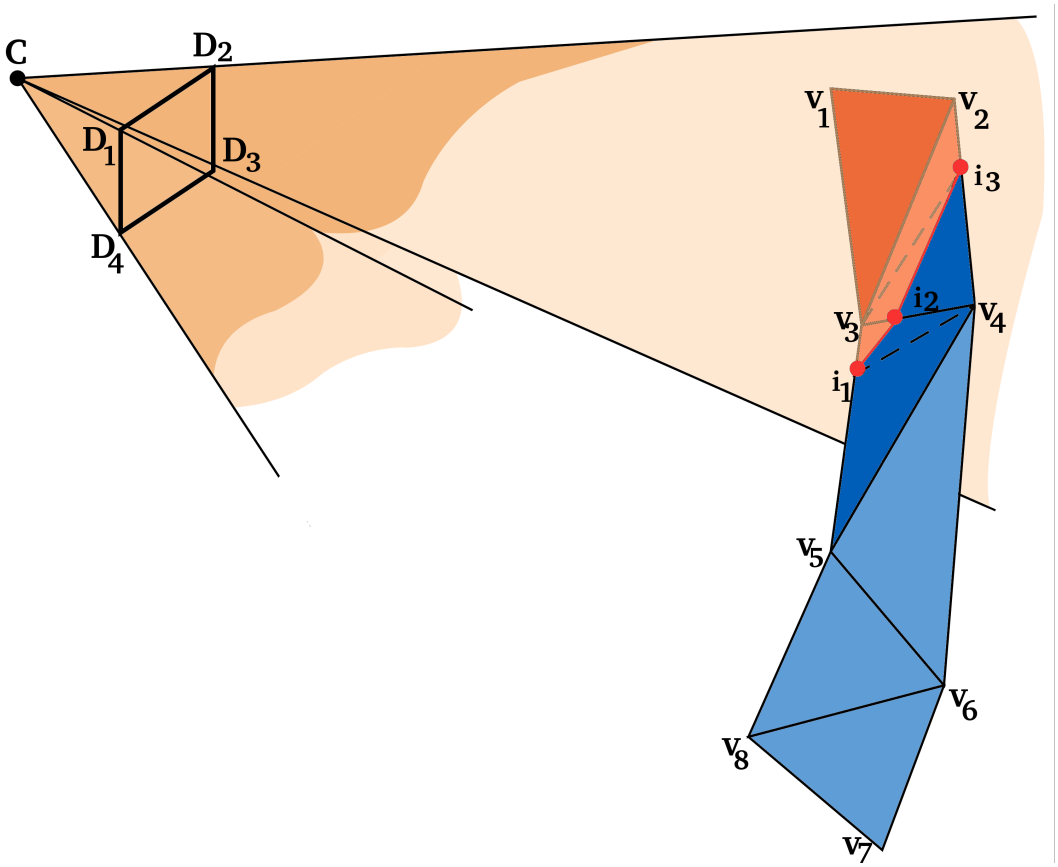


Figure 3.3. Clipping of triangles partly in field of view.

Adjusting previous formula to compute unknown d to

$$\begin{aligned} d(v_2 - v_1) \cdot n_{cxy} + (v_1 - C) \cdot n_{cxy} &= 0 \\ d(v_2 - v_1) \cdot n_{cxy} &= (C - v_1) \cdot n_{cxy} \\ d &= \frac{(C - v_1) \cdot n_{cxy}}{(v_2 - v_1) \cdot n_{cxy}} \end{aligned} \tag{6}$$

If $d \in \leq 0, 1 \geq$ point of intersection is $p = v_1 + d(v_2 - v_1)$.

Once point of intersection are computed algorithm checks whether these points already exist in array of vertices positions. If not adds them to structure of arrays and creates two or three new triangles from partly visible one [2]. Figure 3.4 shows why two or three.

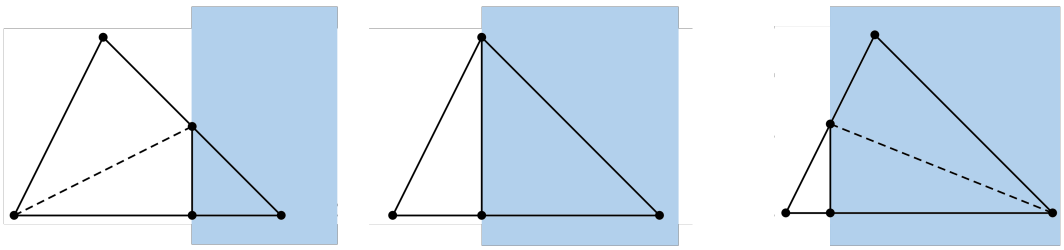


Figure 3.4. Possible triangle splitting.

As Figure 3.4 illustrates, dividing line can split triangle into two smaller triangles if it not passes through one of vertices, it other cases is triangle split into three new

triangles. Partly visible triangle is thrown away and new triangles are immediately tested on visibility in field of view and than put into structure of triangles. Pseudo code of this algorithm is below in Algorithm 3.3 box.

Algorithm 3.3 Triangles in cameras field of view

input: structure of vertices V
structure of triangles T
camera structure *camera*

1. Find equations for planes describing field of view pyramid.
2. ***for each*** visible triangle t_i of T
3. Determine whether vertices of t_i are in field of view.
4. ***if*** all 3 vertices of t_i are out of field of view ***then***
5. $t_{visiblei} = 0$
6. ***else if*** 1 or 2 vertices of t_i are out of field of view ***then***
7. Find intersection between four planes and triangle.
8. Create new triangles so that they are either fully inside field of view or fully out of field of view.
9. Add new triangles to T and throw away t_i.
10. ***endif***
11. ***endfor***

Figure 3.5 illustrates the same scene as Figure 3.2 but this time after Backface culling and removing of triangles out of field of view. Face E was out of field of view and now is marked in red color as not visible. Because face B was partly out of field of view algorithm divided this face into two new faces. Visible I and not visible J. Similar situation occurred with face D. It was also divided into two new faces, but face K is still partly covered by face H. This problem will solve next part of algorithm.

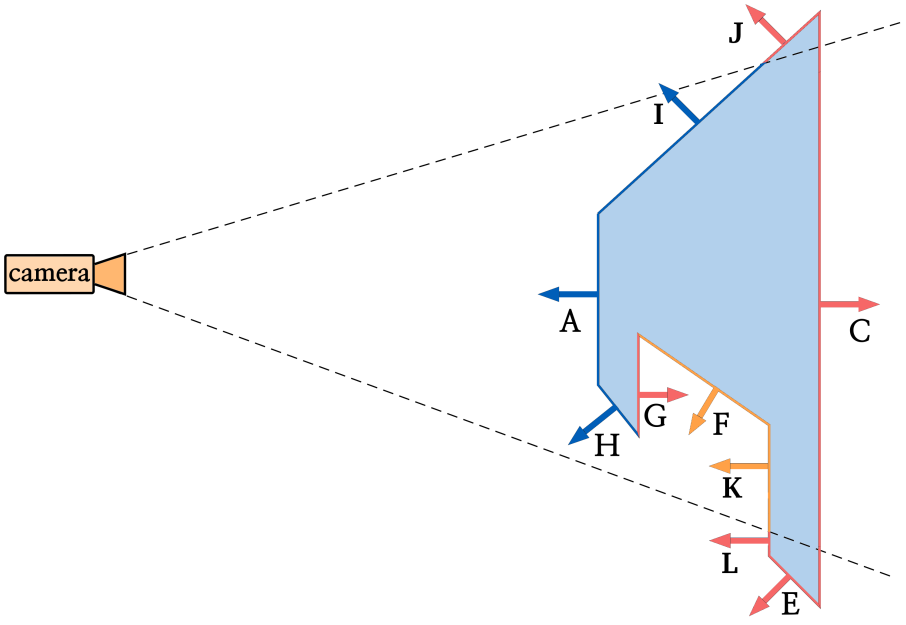


Figure 3.5. Discarding of faces out of field of view.

3.7 Hidden triangles

Figure 3.5 shows face K that is partly visible and partly behind face H. Also face F is still marked as visible but from Figure 3.5 is clearly visible that camera is unable to see that face from its position. Following part of algorithm deals with this problem. Every visible triangle $t_{visible}$ is tested whether another triangles lay between it and camera. Firstly are for every visible vertex $v_{visible}$ found triangles closer to camera than $v_{visible}$. Once we have all triangles that are closer to camera and therefore can lie between camera and $v_{visible}$ rest of computation is done in 2D projection on image plane of camera. There is tested whether point $v_{visible}$ is inside any triangle closer to camera. This would mean that such triangle is between camera and $v_{visible}$. Barycentric coordinates are used for determining whether point is inside triangle or not. Figure3.6 shows subdivision of triangle for calculating barycentric coordinates.

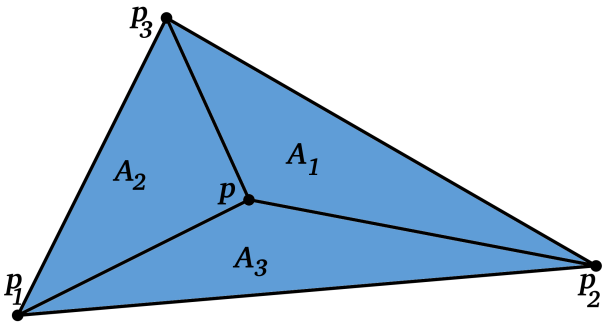


Figure 3.6. Subdivision of triangle for calculating barycentric coordinates.

Area of triangle with vertices v_1, v_2 and v_3, $(v_i = (x_i, y_i))$ is given by

$$A(v_1, v_2, v_3) = \frac{1}{2} \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}. \tag{7}$$

Any point p in plane can be handle ass linear combination of the three vertices of triangle v_1, v_2 and v_3 as

$$p = b_1 v_1 + b_2 v_2 + b_3 v_3, \tag{8}$$

if $b_1 + b_2 + b_3 = 1$. Than the triple (b_1, b_2, b_3) is known as barycentric coordinates of p with respect to v_1, v_2 and v_3. Barycentric coordinates can be expressed geometrically as ratios of area and total area of triangle is $A = A_1 + A_2 + A_3$, then

$$b_1 = \frac{A_1}{A}, b_2 = \frac{A_2}{A}, b_3 = \frac{A_3}{A}. \tag{9}$$

If $b_1, b_2, b_3 < 1$ then p is inside the triangle. If $b_i = 0$ than p lies on the triangle edge opposite to p_i. If $b_i = 1$ than p coincides with p_i [9].

Algorithm 3.4 Hidden triangles

input: structure of vertices V
structure of triangles T
camera structure *camera*

```
1.  for each visible vertex v_i of V
2.      for each triangle t_j of T closer to C then v_i
3.          if v_i is inside t_j in 2D projection on image plane then
4.              v_visiblei = 0
5.              break
6.          endif
7.      endfor
8.  endfor
9.  for each triangle t_i of T
10.     if all 3 vertices of t_i are not visible then
11.         t_visiblei = 0
12.     else if 1 or 2 vertices of t_i are not visible then
13.         t_partly_visible_i = 1
14.     endif
15. endfor
```

3.8 Retriangulation

Previous part marked triangles that are partly visible. This part of algorithm is named *Retriangulation*. Meaning that original triangulation is *REfine*. Retriangulation splits all partly visible triangles into smaller once, that are either completely visible or completely not visible. This does not change dimensions of surface in any way. Only one triangle at a time is retriangulated and new triangulation lies in original. Whole process is very similar to dividing triangles that are partly in field of view, described in Section 3.6. Difference is that in this case, one triangle can be divided into many more than just two or three. Situation is illustrated on Figure 3.7.

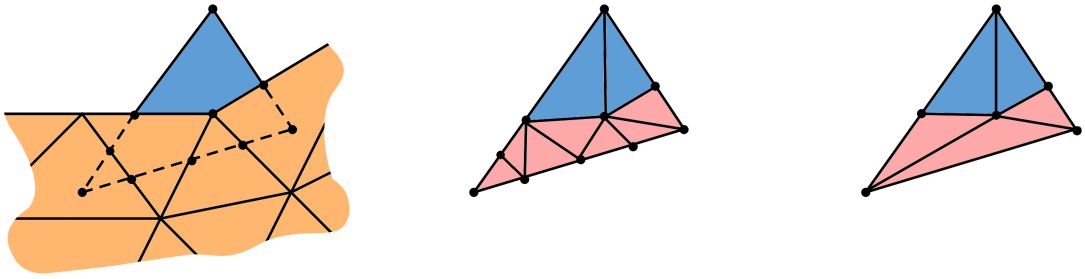


Figure 3.7. Retriangulation.

On Figure 3.7 is captured one triangle in three stages of retriangulation. On the left side is blue triangle partly hidden behind orange triangles. This is when all intersections with covering triangles are computed. Intersections are marked in image as black dots. In the middle is the same triangle now without the wall of orange triangles. Triangle is divided into smaller triangles so that whole new triangle is completely visible (blue triangles) or completely not visible (red triangles). This creates a large amount of new

triangles. In this example is one triangle split into nine smaller ones. Because not visible triangles (red ones) will not be later used in any way there is no need to store all of them. To improve memory effectiveness another triangulation is made. This time not from all intersection points, but only from points of visible triangles plus three vertices of original triangle. This operation creates triangulation on the right side of image where amount of new triangles is reduced to five.

Algorithm itself works partly with 2D projection on image plane and partly with 3D scene. At first are for every partly visible triangle $t_{partly\_visible}$ found triangles closer C than $t_{partly\_visible}$. Let them be T_{closer}. Then is $t_{partly\_visible}$ tested with every triangle from T_{closer} for intersection in 2D projection on image plane. There is used same method as in 3.6 for line segment intersection. Every edge of $t_{partly\_visible}$ is tested for intersection with every edge of every triangle in T_{closer}. If intersection is found for some triangle $t_j \in T_{closer}$ it is possible that any vertex of t_j is inside t_i (note that this is still in 2D projection). Again barycentric coordinates are used to find out whether any point is inside triangle.

Every intersection point and every point inside triangle $t_{partly\_visible}$ is stored in *RetriangulationPoints*. Once every triangle from T_{closer} is tested for intersection with $t_{partly\_visible}$ we have several points for Retriangulation in *RetriangulationPoints*. *RetriangulationPoints* is then supplement by three points forming triangle $t_{partly\_visible}$ and Delaunay triangulation is used on set of points in *RetriangulationPoints*. This creates first Retriangulation, usually with a lot of new triangles hidden. That is why new triangles are tested for visibility in a way described in 3.7. Thanks to subdivision of $t_{partly\_visible}$, new triangles are either visible or not visible. Vertices of visible triangles and three vertices from $t_{partly\_visible}$ are then used for second Delaunay triangulation. Triangles previously labelled as visible stays visible and amount of not visible triangles is reduced.

Since the intersection of triangles is done in 2D on image plane, we receive intersection points also in 2D on image plane. To determine world coordinates of intersection point, similar method as in 3.6 is used. Equations (4), (5) and (6) solve intersection between line (edge of partly visible triangle) and plane (side of pyramid). This time is line represented as a ray from camera passing through intersection point in 2D and plane is represented by triangle $t_{partly\_visible}$. Situation is captured on Figure 3.8. Orange triangle (triangle from T_{closer}) is between camera and blue triangle ($t_{partly\_visible}$). Intersection points x_1 and x_2 are on image plane with point x_3 which is vertex inside $t_{partly\_visible}$. Rays from camera passes through x_1, x_2 and x_3 creating new points X_1, X_2 and X_3 on blue triangle $t_{partly\_visible}$.

Once partly visible triangle $t_{partly\_visible}$ is Retriangulated, $t_{partly\_visible}$ is thrown away. New triangles are added to structure of triangles and new points are added to structure of vertices. Whole process in form of pseudo code is in Algorithm 3.5 below.

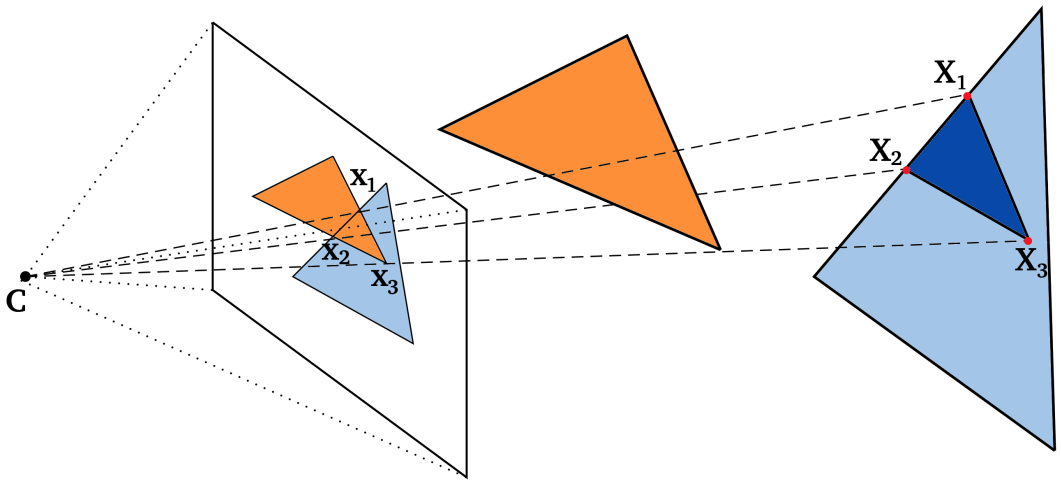


Figure 3.8. Computing 3D coordinates of intersection point.

Algorithm 3.5 Retriangulation

input: structure of vertices V
structure of triangles T
camera structure *camera*

1. ***for each*** partly visible triangle t_i of T
2. Store coordinates of t_i vertices into *RetriangulationPoints*
3. ***for each*** triangle t_j of T closer to C then t_i
4. ***if*** t_j intersect t_i in 2D projection on image plane ***then***
5. Add points of intersection into *RetriangulationPoints*
6. ***if*** Any vertex of t_j is inside t_i in 2D projection ***then***
7. Add these vertices into *RetriangulationPoints*
8. ***endif***
9. ***endif***
10. ***endfor***
11. Compute 3D positions of new points in *RetriangulationPoints*.
12. Use Delaunay triangulation on *RetriangulationPoints*.
13. Determine visibility of new triangles.
14. Use Delaunay triangulation only on three vertices from original triangle and all vertices of new visible triangles.
15. Determine visibility of new triangles, add new triangles to T and throw away t_i.
16. Add new points to V.
17. ***endfor***

3.9 Reflection detection

Once every triangle on examine surface is either visible for not visible for camera, reflection of light source on examine surface is examine. Algorithm looks for reflection of flat panel light source that can capture camera. Reflection of course can can be capture only on visible triangles. Principle of reflection detection is in tracking a ray from camera to visible visible vertex, where normal is known. This allows to compute reflection of such ray and then determine whether reflected ray hits flat light panel.

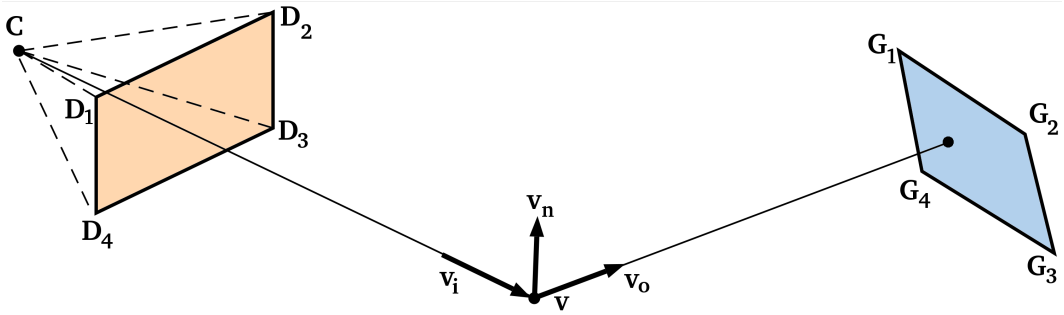


Figure 3.9. Notation in reflex detection.

Once we know that ray can reflect light from camera to light panel, we can assume that light can also be emitted from light panel to camera on the same path.

Figure 3.9 illustrates such situation. Ray from C to v described with vector v_i is reflected as ray v_o on vertex v to flat light panel $G = (G_1, G_2, G_3, G_4)$. Vector v_o is specular reflection of v_i, computed as [10]

$$v_o = (I - 2v_n v_n^T)(v - C), \tag{10}$$

where I is identity matrix 3×3. Once it is determine for every visible vertex whether it can reflex light from light panel to camera, next step finds whole triangles that can reflect light to camera. Important is also to compute whether ray have free path from surface to light panel or whether it hit some obstacle. Figure 3.10 show principle of such computation.

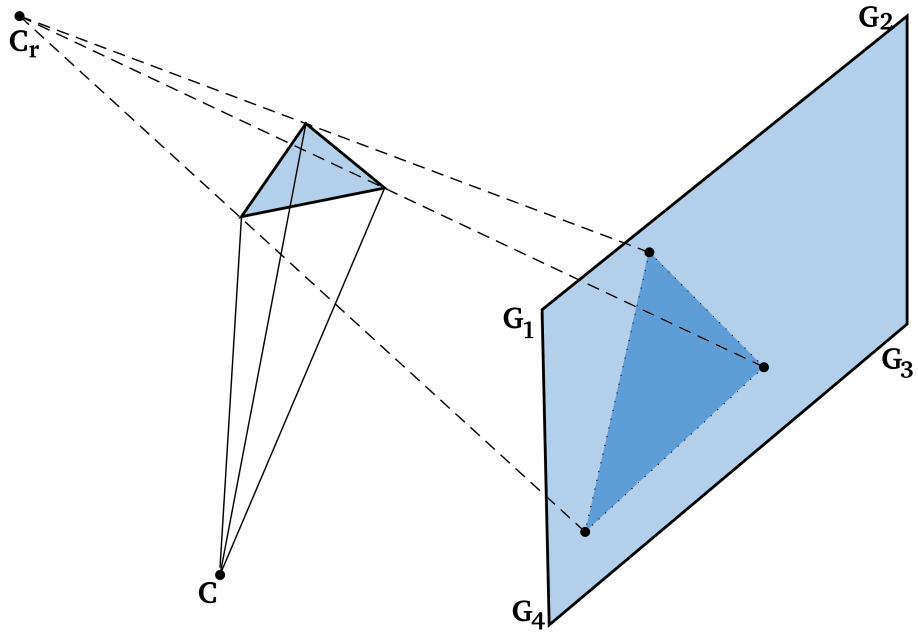


Figure 3.10. Triangle with reflection.

This give us opportunity to create pyramid with apex in C_r and base given by the t_r. Similar procedure as in 3.6 is then run to determine points inside pyramid. This time not all points will be tested but only points between two planes. First plane is given by t_r and the second by G. That creates five conditions. Three for point inside pyramid and two for point between planes. If any point satisfy all five conditions than

there is an obstacle in way for rays and triangle does not reflects light from light panel to camera. Pseudo code of reflection detection is below in box Algorithm 3.6.

Algorithm 3.6 Reflection detection

input: structure of vertices V
structure of triangles T
camera structure *camera*
light source

```
1.  for each visible vertex v_vis of V
2.      Compute ray v_i from C to v_vis and reflected ray v_o.
3.      if v_o passes through rectangle G then
4.          v_reflection_i = 1
5.      endif
6.  endfor
7.  for each visible triangle t_i of T
8.      if all 3 vertices of t_i have v_reflection == 1 then
9.          t_reflection_i = 1
10.         Create reflected camera C_r behind t_vis.
11.         for each triangle t_j of T between G and t_i
12.             if any t_j is inside pyramid (C, t_i) then
13.                 t_reflection_i = 0
14.                 break
15.             endif
16.         endfor
17.     endif
18. endfor
```

Chapter 4
Implementation

Algorithm described in Chapter 3 was implemented in Matlab R2013a from Mathworks company [1]). Inputs for the main script `reflex_main` that covers whole algorithm described in Algorithm 3.1 are:

- Camera matrix P.
- Coordinates of corners of image plane (D_1, D_2, D_3, D_4) (as shown on Figure 3.1).
- Coordinates of corners of light source (G_1, G_2, G_3, G_4) (also on Figure 3.1).
- Triangulated surface in Waveform OBJ formate

We use OBJ formate because it is open format for 3D representation of geometric data. OBJ is easy to open in text editor and understood its structure and many freeware software supports this standard. Another benefit is that OBJ format stores vertices normal and positions which is used in computation. To load model into Matlab is used Waveform OBJ toolbox [14]. Toolbox loads all information about model available, but not all of them are needed. Format contains information about textures and materials used on model. These informations are not used, because they have no impact on models visibility. Needed data are processed into two structures. One for vertices and the other for triangles. Format of structures is described in Tables 3.1 and 3.2.

Script `reflex_main` calls five scripts one by one. They are `backfaceCulling` for Backface Culling, `inFOV` for determining triangles in field of view and splitting those on the edge, `hiddenTriangles` to select triangles hidden behind another part of surface, `triangulate` for retriangulation, and finally `reflex` that computes which visible triangles reflects the light from the light source into the camera. Each script have structure of vertices V, structure of triangles T and camera as input. Only `reflex` have also parameters of light source as input. All scripts returns V and T as output with updated information about visibility and reflection.

[1]) http://www.mathworks.com

Chapter 5
Testing

Algorithms were implemented and tested in Matlab R2013a on object designed to cover all problems that ware stated in 1.1. This object is rotated curve shown on Figure 5.1.

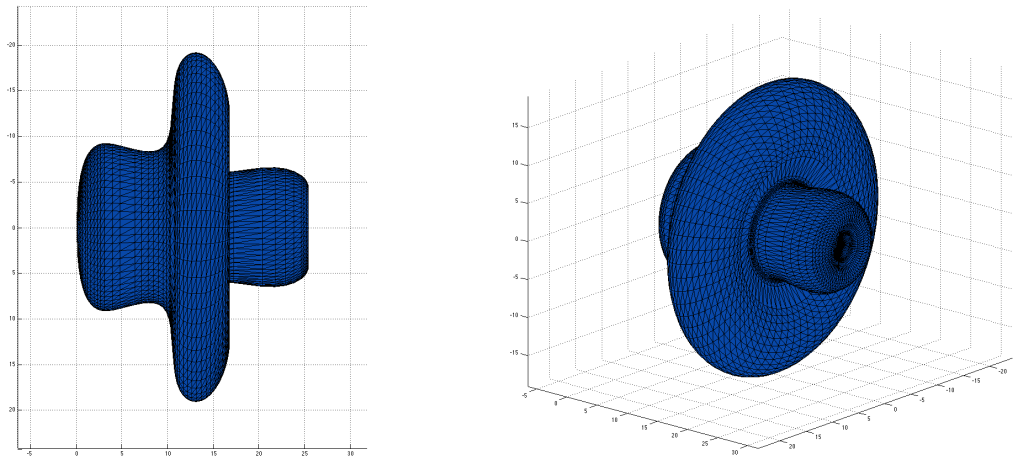


Figure 5.1. Objects of testing - rotated curve with 8064 triangles. On the left is top view and on the right side view.

5.1 Backface culling

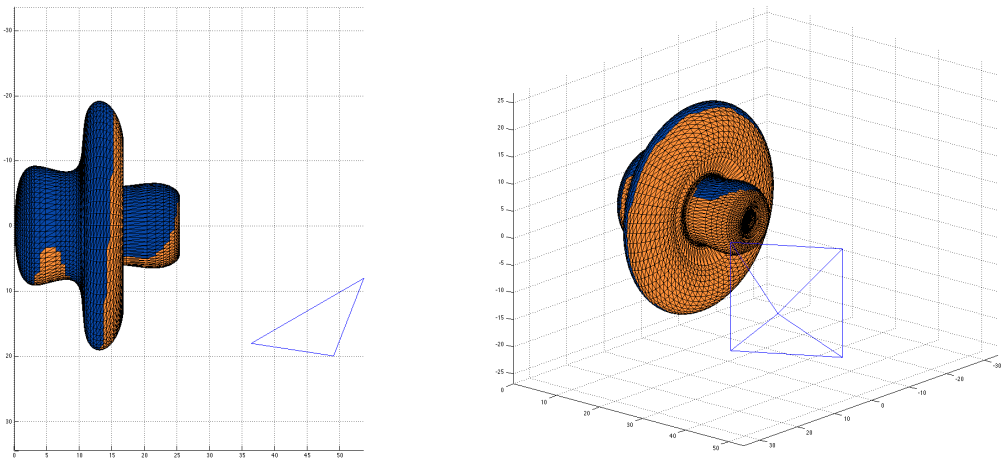


Figure 5.2. Backface culling. On the left is top view and on the right side view.

Figure 5.2 shows test object with camera from two views. Camera is represented by triangle with blue edges on the left and four triangles with blue edges creating pyramid representing field of view on the right side. Model is covered with two colors. Blue color marks triangles that are not visible for camera after backfaace culling and orange marks triangles that are visible. As mention in chapter 4 triangles out of filed of view (visible on the left side of Figure) are still marked as visible so the triangles covered by another part of model. This is expected and correct result.

5.2 Triangles in field of view

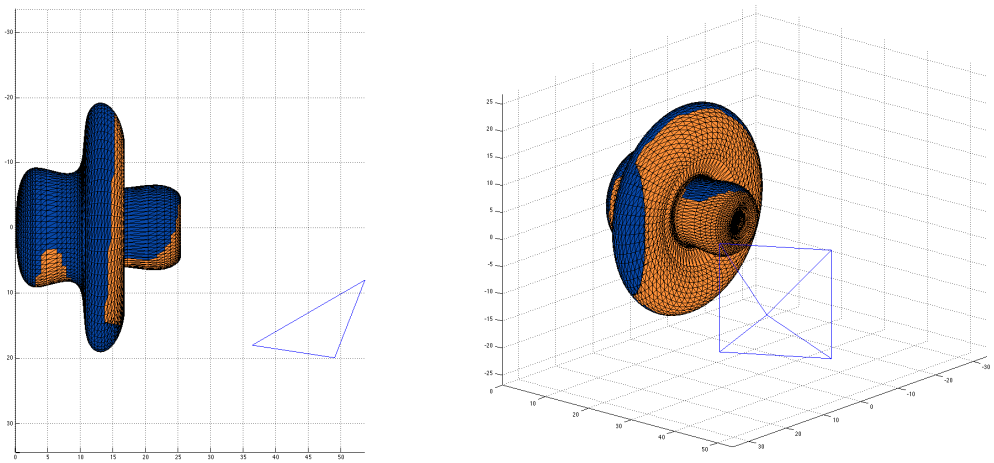


Figure 5.3. Triangles in field of view. Same views as in previous image.

Triangles that were out of field of view are now marked blue as not visible. Triangles on the edge of field of view are split into fully visible and completely not visible triangles. On the right of Figure 5.3 is clearly visible dividing line. More extreme example is on Figure 5.4 where field of view pyramids edge is passing through model.

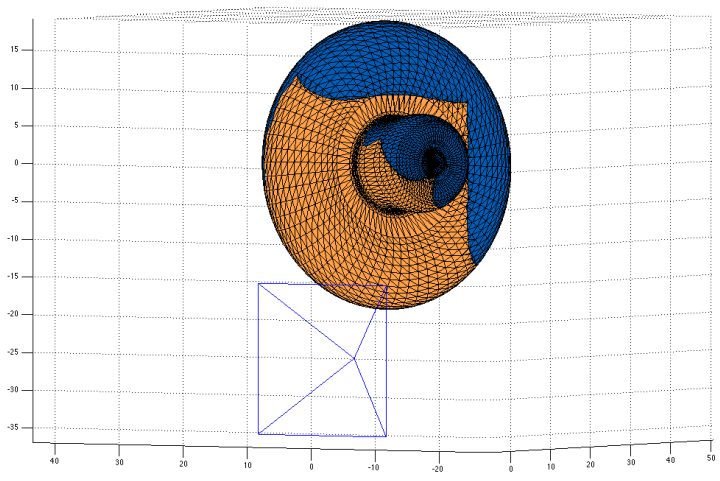


Figure 5.4. Another example of triangles in field of view.

5.3 Covered triangles

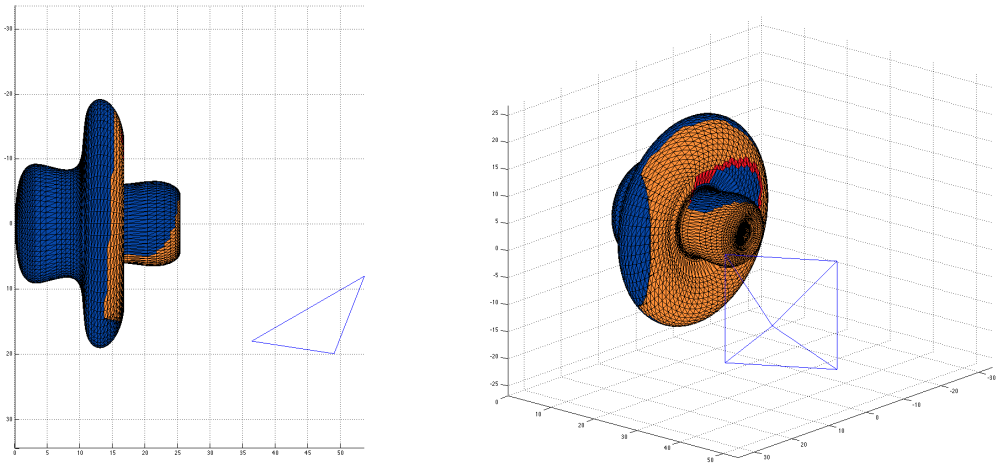


Figure 5.5. Covered triangles. Same views as in previous image.

On Figure 5.5 is tested object after running algorithm for removing triangles covered behind another part of model. It is clear that all triangles that were hidden are now blue coloured and thus not visible. But thanks to part of object that cast shadow, there are now few triangles partly visible and partly covered. These triangles are marked in red color and will be split into smaller triangles in new Retriangulation section.

5.4 Retriangulation

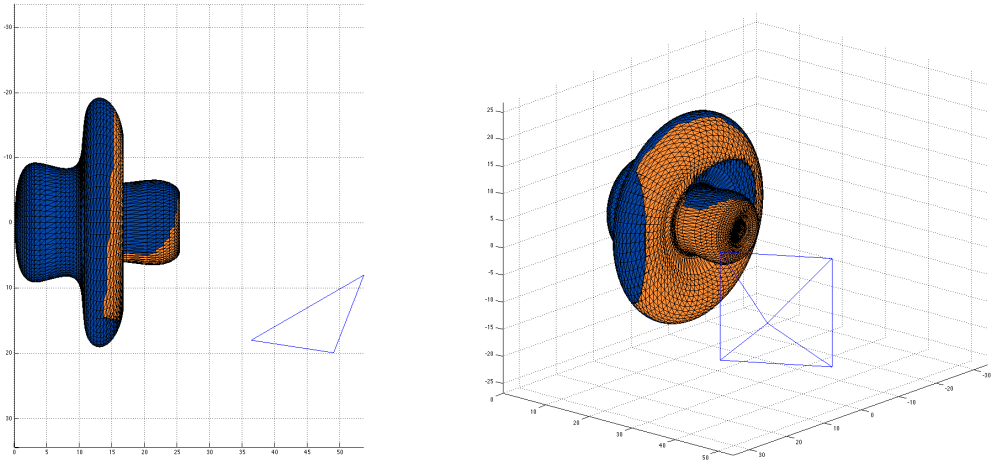


Figure 5.6. Retrianulation. Same views as in previous image.

Figure 5.6 captures situation after retriangulation of partly visible triangles. The edge between visible and not visible part of model is now clean and model is ready for reflection detection.

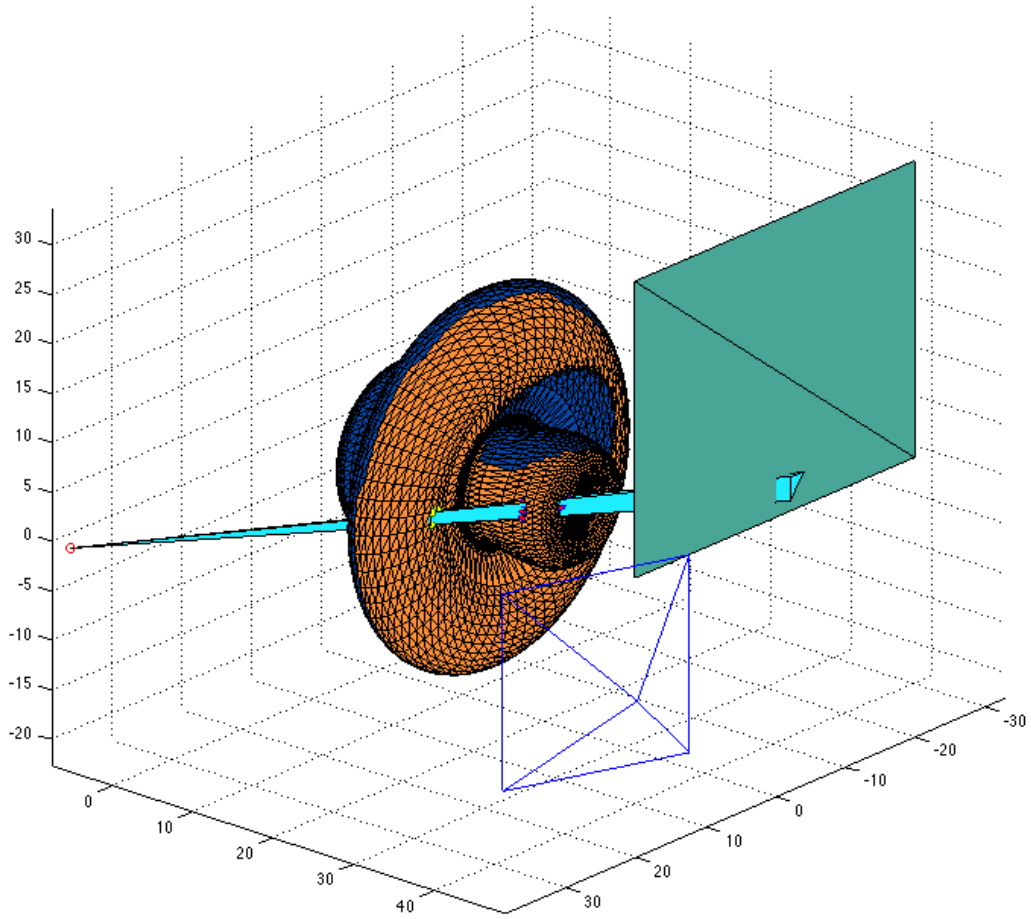


Figure 5.7. Reflection detection algorithm with pyramid from C_r through tested triangle.

5.5 Reflection

Figure 5.7 shows work of algorithm for reflection detection when looking for obstacles in a rays path. Reflected camera center C_r is on the left in red circle and obstacles in a path of rays are painted with pink dots.

Figure 5.8 shows reflection detected on examine surface. Surface is very curvy and therefore reflection is visible only in small part of surface. See Figure 1.2 where is reflection detected on flat surface and more clearly visible.

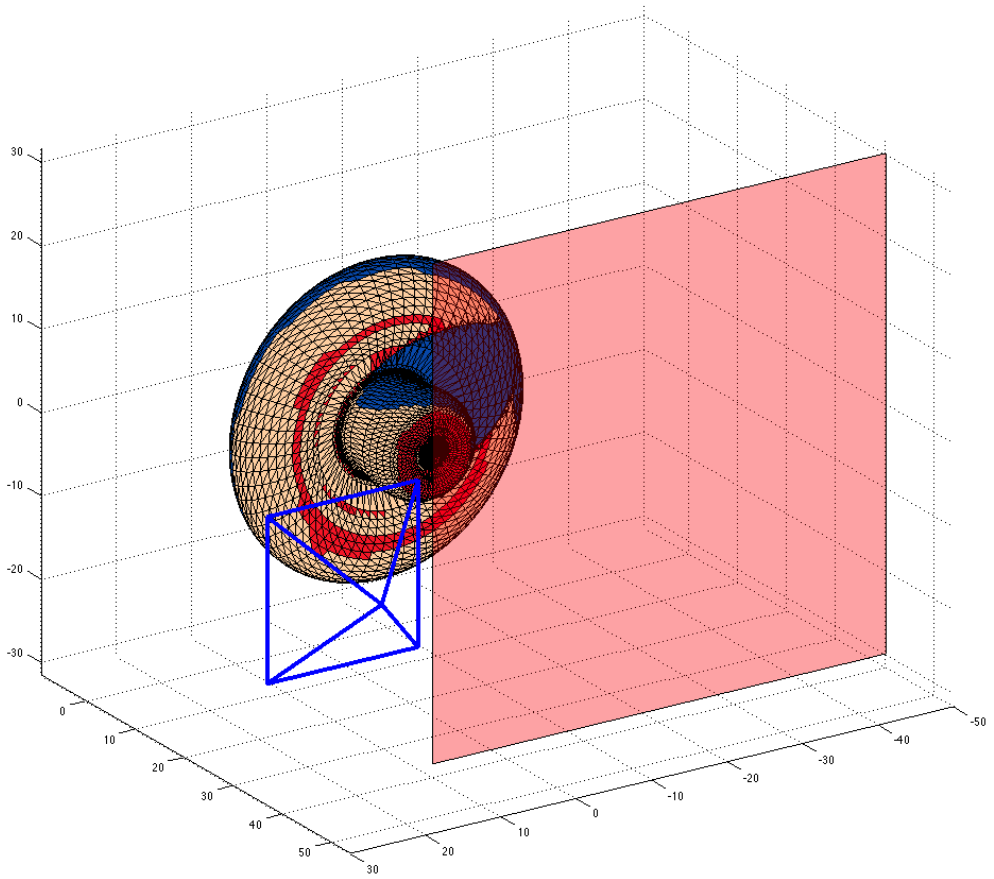


Figure 5.8. Reflection (in red color) of light source (red see through rectangle) on examine surface.

Chapter 6
Conclusion

In this thesis was designed and implemented algorithm for determining visibility and reflection on glossy triangulated surfaces. In the Section 1.1 ware stated five problem that needs to be solve in order to consider proposed algorithm successful. Problem with determining triangles facing towards camera was solved using method called Backface culling. Second problem was to locate parts of surface inside field of view. This problem was solved. Each triangle outside of field of view is labelled as not visible and triangles on the edge are refine to two or three new triangles that are either fully visible or fully not visible. Third stated problem was to determine parts of surface hidden behind another part of surface. Algorithm that finds every hidden triangle was designed and test proves its correctness. Removing hidden faces leaves certain triangles partly visible and partly hidden. To deal with this problem and solve fourth stated problem was design retriangulation algorithm that refine triangulation and leave whole examine surface with only completely visible and completely not visible triangles. And finally to label every triangle that reflect light from light source into the camera and thus solve last problem, was design algorithm that follows rays from camera into the light source. If the light ray have free path from camera into the light source, ray can also travel the opposite direction.

Chapter 3 describes algorithm design, Chapter 4 briefly describes implementation of designed algorithm in Matlab and Chapter 5 shows results of implementation. Proposed algorithm satisfy all goals that ware stated and offers tool to study visibility and reflection on glossy surfaces. In comparison to the traditional algorithm does not work in limited raster but refines examine surface and as its output produce labelled triangulation. Algorithm can be modified for more than one camera and more than one light source, creating interesting tool that can be used to cover whole surface with visible reflections. Example of usage is to let algorithm arrange cameras and light sources around model of real glossy object and then use this arrangement in real life study of reflection on surface when looking for possible flaws like scratches or bumps. Such flows would resolute in different light reflection than algorithm computes.

References

[1] Jiří Sochor, Jiří Žára, and Bedřich Beneš. *Algoritmy počítačové grafiky*. ČVUT Praha, 2-nd edition, 1996.

[2] David Eberly. *3D game engine design : a practical approach to real-time computer graphics*. Morgan Kaufmann, San Francisco, 2001.

[3] Jiří Bittner and Peter Wonka. Visibility in computer graphics. *Visibility Survey*, 2003.

[4] Sagi Katz, Ayellet Tal, and Ronen Basri. Direct visibility of point sets. *SIGGRAPH*, 26, 2007.

[5] Patrick O'Beirne. Managing risk in euro currency conversion. *Cutter IT Journal*, 11:22–29, 1998.

[6] Pal Benko. Endgame lab: Triangulation. *Chess Life*, November:48–49, 2010.

[7] Margaret S. Mahler. On human symbiosis and the vicissitudes of individuation. *The International Journal of Psychoanalysis*, October:740 – 763, 1967.

[8] Wikipedia. Triangulation (disambiguation) — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Triangulation_(disambiguation)`. Online, accessed: 10-April-2014.

[9] Øyvind Hjelle and Morten Dæhlen. *Triangulations and applications*. Springer Verlag, S.l, 2010.

[10] David Halliday. *Fundamentals of Physics*. Wiley, 2013.

[11] Simon Jeremy Damion Prince. *Computer vision : models, learning, and inference*. Cambridge University Press, 2012.

[12] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics - Principles and Practice*. Addison-Wesley, 2-nd edition, 1990.

[13] John Hughes. *Computer graphics - principles and practice*. Addison-Wesley, Upper Saddle River, New Jersey, 3-rd edition, 2014.

[14] Dirk-Jan Kroon. Wavefront obj toolbox. `http://www.mathworks.com/matlabcentral/fileexchange/27982`. Online, accessed: 3-February-2014.

Appendix A
Content of enclosed CD

On enclosed CD can be found following directories:

- **Thesis** - Contains this thesis in PDF.
- **Source** - Contains source code of implementation in Matlab.
- **Figures** - Contains figures shown in Chapter 5 from algorithm testing.

In source folder are prepared five scripts that can be run in Matlab. They will produce results shown in Chapter 5. They are `test_bc` for Backface culling test, `test_fov` for test of algorithm to determine triangles in field of view, `test_hidden` for hidden triangles determining, `test_retriangulation` for retriangulation test and `test_all` for complete algorithm test.