IJMMS 2004:21, 1121–1132
PII. S0161171204211012
http://ijmms.hindawi.com

REFLECTION CURVES—NEW COMPUTATION AND RENDERING TECHNIQUES

DAN-EUGEN ULMET

Received 1 November 2002

Reflection curves on surfaces are important tools for free-form surface interrogation. They are essential for industrial 3D CAD/CAM systems and for rendering purposes. In this note, new approaches regarding the computation and rendering of reflection curves on surfaces are introduced. These approaches are designed to take the advantage of the graphics libraries of recent releases of commercial systems such as the OpenInventor toolkit (developed by Silicon Graphics) or Matlab (developed by The Math Works). A new relation between reflection curves and contour curves is derived; this theoretical result is used for a straightforward Matlab implementation of reflection curves. A new type of reflection curves is also generated using the OpenInventor texture and environment mapping implementations. This allows the computation, rendering, and animation of reflection curves at interactive rates, which makes it particularly useful for industrial applications.

2000 Mathematics Subject Classification: 65D17, 65D18, 51P05, 78A05.

1. Introduction. My interest in the subject of reflections arose during the time I spent at the Mercedes Benz company as a CAD/CAM system analyst and programmer. Reflection curves (RCs) on surfaces are important tools in CAD/CAM systems. In the automobile industry, "reflection lines" (RLs), a special case of RC, correspond to the reflection of a set of parallel linear lights placed in the vicinity of the car body, as can be seen in Figure 1.1. Several other impressive images can be found on the homepage of Dassault Systems and the CAD/CAM system CATIA.

In the CAGD literature (the theory behind the algorithms that are used for the development of CAD/CAM software systems), Klass (see [6]) was one of the first to describe an industrial implementation of RLs in the CAD/CAM system SYRKO developed by Mercedes Benz. In the meantime, RLs are implemented in several industrial CAD/CAM systems.

Another important reflection issue in the automobile industry is the display of the reflections of the dashboard lights on the windshield in the dark. A simple approach is to view the dashboard as a plane surface and the lights as primitive curves like ellipses, circles, and line segments and neglect their width. Even though we will not deal with dashboard reflections here, they can be analyzed with techniques similar to those included in this note.

Physically accurate RCs or RLs require a considerable implementation effort due to the complicated mathematical algorithms which are needed. A series of other related concepts like isophotes (see [11]), a different type of RLs introduced by Kaufmann and

FIGURE 1.1. Reflections. Photo courtesy of the DaimlerChrysler company.

Klass (see [5]), or highlight lines (see [1, 2]) were implemented in CAD/CAM systems. These objects are mathematically less complicated and easier to implement; they are also useful surface interrogation tools. All types of RLs are important characteristic curves on surfaces and their smoothness and curvature properties (see [12]) give information about the overall quality of the underlying surface.

In this note, several new approaches to RCs and RLs and related concepts are introduced. These approaches are all designed to take the advantage of the graphics packages included in recent releases of the OpenInventor toolkit (see [13]) or in Matlab (see [9]). First, "physical reflection lines" (PRLs) are reviewed and a new implementation in Matlab is outlined. Next, a type of simplified "raytracing reflection lines" (RRLs) and a straightforward Matlab implementation are presented.

Next, we describe a new way of viewing RLs (which are curves on a surface) as contour curves (contour reflection lines) (CRLs) of a certain function defined in the parameter domain of the surface. This function captures the significant geometric information of the scene (surface, curve, and viewpoint). One advantage of this new method is the fact that contour curves are well-known tools and extensively used in rendering, and several robust computation algorithms are available (see [1] and the references therein). For our implementation, we use the contouring routines in Matlab.

Finally, a new type of RLs is introduced by a combination of texture and environment mapping, leading to what we will call "texture reflection lines" (TRLs). We were inspired to introduce this new type of RLs by the graphical features provided in the OpenInventor toolkit. The OpenInventor seems to become the new industrial standard in graphics development and rendering. Its features enabled us to implement the computation, rendering, and animation of RLs at interactive rates, which is particularly interesting for industrial applications. Though TRLs look quite straightforward from the theoretical point of view, they were considered useful by practicioners in an early experimental stage of a new release of the reflection software.

State-of-the-art graphics hardware is OpenGL accelerated and handles reflections by efficient raytracers.

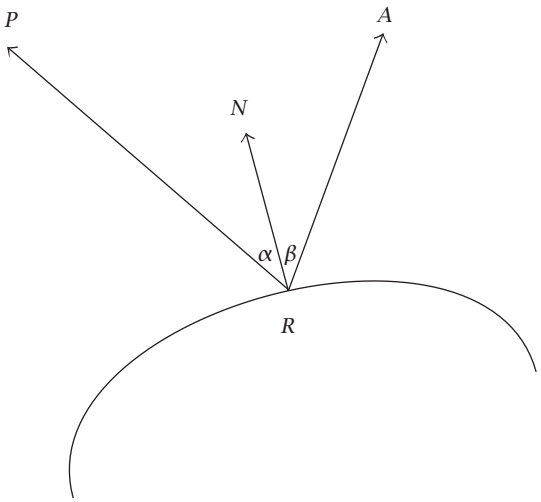

FIGURE 2.1. The geometry of reflection.

The use of Matlab and OpenInventor has been employed in order to reduce the implementation time of the various concepts. Future research will also address the evaluation of these methods regarding their suitability for the correction (see [2, 6, 7, 10]) and aesthetical analysis of RLs.

2. The reflection equation. Figure 2.1 shows a viewpoint/camera point A, a light point P on a light line, the corresponding reflection point R, and the normal $R\vec{N}$ to the surface. We will briefly review the derivation of the reflection equation described in [4]. The condition of reflection in the sense of geometrical optics is

$$\alpha = \beta, \tag{2.1}$$

where $\alpha = \widehat{NRP}$, $\beta = \widehat{NRA}$. We will also use the following notation in this section: $\vec{a} = R\vec{A}$, $\vec{p} = R\vec{P}$, and $\vec{n} = R\vec{N}$.

Start by setting $\cos\alpha = \cos\beta$. Then we have

$$\frac{\vec{p}\cdot\vec{n}}{|\vec{p}|\,|\vec{n}|} = \frac{\vec{a}\cdot\vec{n}}{|\vec{a}|\,|\vec{n}|}. \tag{2.2}$$

The following relation holds for the three planar unit vectors:

$$\frac{\vec{p}}{|\vec{p}|} + \frac{\vec{a}}{|\vec{a}|} = \mu\frac{\vec{n}}{|\vec{n}|} \tag{2.3}$$

for some real scalar μ.

It follows that

$$\frac{\vec{p}\cdot\vec{n}}{|\vec{p}|\,|\vec{n}|} + \frac{\vec{a}\cdot\vec{n}}{|\vec{a}|\,|\vec{n}|} = \mu, \tag{2.4}$$

where by "$\cdot$" we denote the scalar product of vectors.

Combine (2.4) and (2.2) to get

$$\mu = 2\frac{\vec{p}\cdot\vec{n}}{|\vec{p}|\,|\vec{n}|} = 2\frac{\vec{a}\cdot\vec{n}}{|\vec{a}|\,|\vec{n}|}, \tag{2.5}$$

and substitute into (2.3) to derive the "reflection equation"

$$\frac{\vec{p}}{|\vec{p}|} + \frac{\vec{a}}{|\vec{a}|} = 2\frac{\vec{a}\cdot\vec{n}}{|\vec{a}|\,|\vec{n}|}\cdot\frac{\vec{n}}{|\vec{n}|}. \tag{2.6}$$

The reflection equation can be approached in several different ways. This will be detailed in the following sections. In Section 3, the equation is treated as a nonlinear equation. Section 4 uses a simplified type of raytracing and linear algebra to generate the RLs. The theory of these first two sections is well known. However, the implementation in a higher programming language as Fortran or C/C++ is quite time consuming. We report on fast and straightforward implementations, which take advantage of Matlab's built-in matrix-oriented computations and 3D graphics. Hereby, the stage is also set for Section 5, which is dedicated to a new approach to RLs based on implicit curves and the countour functions in Matlab. Finally, Section 6 offers an approach to and an implementation of a type of RLs generated by texture and environment mapping using the OpenInventor graphics package developed by Silicon Graphics.

3. Physical reflection lines (PRLs). In real physics, the ray travels from the light source P to the eye A after being reflected on the surface. In this approach, the points P and A are considered known and the reflection point R has to be computed.

We assume that the surface is given in parametric form $\vec{r} = \vec{r}(u,v)$. Then the three vectors involved in the reflection scene can be represented as $\vec{n} = \vec{n}(u,v)$, $\vec{p} = \vec{p}(u,v)$, and $\vec{a} = \vec{a}(u,v)$, and the reflection equation as the nonlinear equation

$$h(u,v) := \frac{\vec{p}}{|\vec{p}|} + \frac{\vec{a}}{|\vec{a}|} - 2\frac{\vec{a}\cdot\vec{n}}{|\vec{a}|\,|\vec{n}|}\cdot\frac{\vec{n}}{|\vec{n}|} = 0. \tag{3.1}$$

To solve this equation in a numerically stable manner, we transform it into a nonlinear minimization problem.

Define $f(u,v) := |h(u,v)|^2$. Then (3.1) is equivalent to the minimization of the nonnegative scalar function $f(u,v)$.

This method provides physically "correct" reflections. However, the computation requires the solution of nonlinear minimization problems of two variables for light points P on the parallel light lines. Then the reflection points R usually have to be fitted to an approximating spline curve (of the RL) on the underlying spline surface. In industrial CAD/CAM systems, the approximating spline is needed for the purpose of subsequent geometry processing (such as surface smoothing by the correction of the RLs). This requires a tracing routine of the reflection points and a spline approximation routine. The programming effort in a higher programming language such as Fortran and C/C++ is considerable. An implementation in the CAD/CAM system SYRKO of the Daimler-Chrysler company was extensively used in the design of Mercedes automobiles.

If only the simulation of the RLs by the display of a set of reflection points corresponding to a fine sampling of light points is desired, then at least a visual inspection of

the RLs is possible with a reasonable implementation effort. This is especially straightforward if a "very high programming language" such as Matlab is employed: the built-in minimization routine *fmins* can solve the reflection equation (3.1) with a degree of accuracy which obviously depends on the quality of the Matlab routine and the considered test surfaces. The routine *fmins* uses a Nelder-Mead simplex search and allows tolerances (for the free vector variable and the function) and the maximum number of iteration steps as inputs (see [9]).

4. Simplified raytracing lines (RTLs). We consider a sample of points R on the surface. We trace back the rays $\vec{AR}$ from the eye/camera point A to the sample points R and check if the reflected rays $\vec{RP}$ hit one of the light lines in the light plane. If this is the case, the tested sample point R is a reflection point of that particular light line and hence belongs to the corresponding RL. If the sampling of the surface is dense enough, we can simulate the RLs by marking the points on the surface that pass the intersection test between the rays and the light lines.

A ray $\vec{RP}$ starts in R and has a direction $\vec{s}$ that can be derived from the reflection equation (2.6):

$$\vec{s} = \frac{\vec{p}}{|\vec{p}|} = 2\frac{\vec{a}\cdot\vec{n}}{|\vec{a}|\,|\vec{n}|}\cdot\frac{\vec{n}}{|\vec{n}|} - \frac{\vec{a}}{|\vec{a}|}. \tag{4.1}$$

Consider also a point B on a light line with direction $\vec{c}$. Then the 3D distance between the ray and the light line (regarded as lines) is

$$d = \frac{|(\vec{s}\times\vec{c})\cdot\vec{b}|}{|\vec{s}\times\vec{c}|}, \tag{4.2}$$

where $\vec{b} = \vec{RB}$ (by "×", we denote the cross product).

Because A, R (and N) are considered known in this setting, the computation of the distance d using (4.1) and (4.2) is a straightforward linear algebra. The intersection test ray/light line should be $d \le \text{eps}$, where eps is some tolerance or the radius of the light line (which is usually a cylinder).

Note that in this simplified raytracing approach, we do not consider an image plane and multiple reflections. This is reasonable when we focus our attention on the surface alone and not on the entire scene, and also if the topology of the surface is not very "exotic." The RLs obtained in this way do not have of course the quality of the excellent commercial or public domain raytracers, but still reveal qualitative information about the surface, while keeping the implementation effort and the computation time to a minimum. The implementation is particularly straightforward in Matlab due to the array data structures generated by a rectangular sampling of the surface. Using Matlab's built-in operations for higher-dimensional arrays (available since Matlab Release 5) and the routines for sparse matrices [9], which occur in the ray/light line intersection step, the geometric computations can be completely parallelized.

This section was also included to introduce notation and set the stage for a new approach to RLs, which will be addressed in the following section.

5. Contour reflection lines (CRLs). As in Section 4, we suppose that A, $R(u,v)$ (and $N(u,v)$) are known for (u,v) in the parameter domain of the surface. In this section, we will use an orthogonal coordinate system with the origin in $O(0,0,0)$ and the directions of the axes $\vec{\imath} = (100)'$, $\vec{\jmath} = (010)'$, and $\vec{k} = (001)'$. Vectors $\vec{x} = (xyz)'$ will be represented in this basis as $\vec{x} = x\vec{\imath} + y\vec{\jmath} + z\vec{k}$ and the corresponding points X with $\vec{x} = \vec{OX}$ as $X(x,y,z)$. This will keep the subsequent computations simple and the results as explicit as possible.

Consider now the parametric equation of the plane of the light lines ("light plane") $z = z_0 = \text{constant}$:

$$\vec{x} = x\vec{\imath} + y\vec{\jmath} + z_0\vec{k}, \quad x,y \in \mathbb{R}. \tag{5.1}$$

The light lines are situated in the light plane parallel to the y-axis, with $x = c$. Their parametric equation is

$$\vec{x_c} = c\vec{\imath} + y\vec{\jmath} + z_0\vec{k}, \quad y \in \mathbb{R}. \tag{5.2}$$

Denote the vector $\vec{OR}$ by $\vec{r} = \vec{r}(u,v)$.

Follow the ray $\vec{AR}$ to the surface. When the ray hits the surface in R, it will be reflected to a ray with the parametric equation

$$\vec{x}(u,v) = \vec{r}(u,v) + t\vec{s}(u,v), \quad t \in \mathbb{R}, \tag{5.3}$$

where the direction $\vec{s} = \vec{s}(u,v)$ is given by the right-hand side of (4.1).

The intersection point $P(u,v)$ of the ray (5.3) with the light plane (5.1) can be determined by solving the linear equation $\vec{x}(u,v) = \vec{x_c}$ (all the vectors are known at this time, while t or y needs to be determined),

$$\vec{r}(u,v) + t\vec{s}(u,v) = c\vec{\imath} + y\vec{\jmath} + z_0\vec{k}. \tag{5.4}$$

Equation (5.4) has a unique solution $\vec{OP} = \vec{p}(u,v)$ for all $R(u,v)$ for which the plane (5.1) and the ray (5.3) are not parallel. This is the case when $\vec{s}(u,v)\cdot\vec{k} \neq 0$. We compute the solution.

By (scalar) multiplication of the vector equation (5.4) with the basis vector $\vec{k}$, we have

$$\vec{r}\vec{k} + t(\vec{s}\vec{k}) = z_0. \tag{5.5}$$

From (5.5), we determine the intersection parameter

$$t = \frac{z_0 - \vec{r}\vec{k}}{\vec{s}\vec{k}}, \tag{5.6}$$

and from (5.3), the intersection

$$\vec{p} = \vec{r} + \frac{z_0 - \vec{r}\vec{k}}{\vec{s}\vec{k}}\vec{s}. \tag{5.7}$$

Now, if the intersection point P lies on one of the light lines $x = c$, that means that the corresponding point R belongs to the RL of that particular light line. This is the

case when the x-coordinate of P is equal to c, that is, $x = \vec{p}\vec{\imath} = c$. Using the expression for $\vec{p}$ found in (5.7), this means

$$x = \vec{r}\vec{\imath} + \frac{z_0 - \vec{r}\vec{k}}{\vec{s}\vec{k}}(\vec{s}\vec{\imath}) = c. \tag{5.8}$$

This equation can be rewritten as

$$\begin{aligned} \frac{(\vec{r}\vec{\imath})(\vec{s}\vec{k}) - (\vec{r}\vec{k})(\vec{s}\vec{\imath}) + z_0(\vec{s}\vec{\imath})}{\vec{s}\vec{k}} &= c, \\ \frac{(\vec{s}\times\vec{r})\vec{\jmath} + z_0(\vec{s}\vec{\imath})}{\vec{s}\vec{k}} &= c, \end{aligned} \tag{5.9}$$

or finally

$$\frac{[\vec{s}\vec{r}\vec{\jmath}] + z_0(\vec{s}\vec{\imath})}{\vec{s}\vec{k}} = c, \tag{5.10}$$

where by $[\vec{x}\vec{y}\vec{z}] = (\vec{x}\times\vec{y})\cdot\vec{z}$, we denote the mixed product of three vectors.

The implicit equation (5.10) represents the preimage of the RL corresponding to the light line $x = c$ in the parameter domain of the parametric surface. It can be viewed as a contour line of the function on the left-hand side of (5.10). Hence the following new result can be stated.

Theorem RL. *The RL of the "light line" $x = c$ situated in the light plane $z = z_0$ is the image of the mapping of the implicit curve $f(u,v) = c$ from the parameter domain of the surface to the surface. The function $f(u,v)$ is defined by*

$$f(u,v) = \frac{[\vec{s}\vec{r}\vec{\jmath}] + z_0 \cdot (\vec{s}\vec{\imath})}{\vec{s}\vec{k}}, \tag{5.11}$$

where $\vec{r} = \vec{r}(u,v)$ is the parametrization of the surface and $\vec{s} = \vec{s}(u,v)$ is defined in (4.1).

Loosely speaking, the following interesting statement can be made.

Corollary 5.1. *The RLs of the light lines $x = c$ are the contour curves $f(u,v) = c$.*

Figure 5.1 shows five RLs in the parameter domain of the surface. Note that some RLs split into two parts. Figure 5.2 displays the real RLs after the mapping from the 2D parameter domain to the surface in 3D domain. For the generation of these figures, we used a Matlab program that generates an RL which passes through an interactively given point on the surface. The five points corresponding to the five RLs and their counterparts in the parameter domain are also displayed in the two figures.

By reasoning in a similar way as in the derivation of (5.8) and the subsequent refinements, we can also determine an expression for $y = \vec{x}\vec{\jmath}$ and state a more general result for the reflection of implicit curves on a parametric surface. Suppose the curve is given by an implicit equation of the type $g(x,y) = c$ in the light plane $z = z_0$. Then the following new result holds.

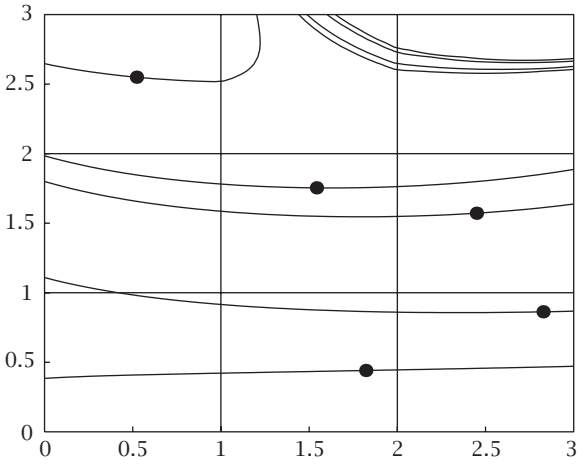

FIGURE 5.1. RLs as contours in 2D domain.

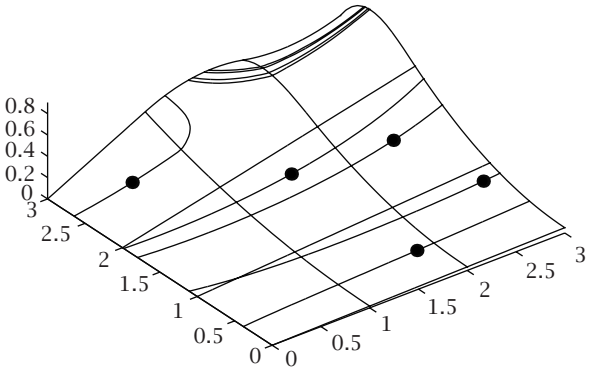

FIGURE 5.2. RLs on the surface.

THEOREM RC. *The RC of the "light curve" $g(x,y)=c$ situated in the light plane $z=z_0$ is the image of the mapping of the implicit curve $f(u,v)=c$ from the parameter domain of the surface to the surface. The function $f(u,v)$ is defined by*

$$f(u,v)=g\left(\frac{[\vec{s}\vec{r}\vec{j}]+z_0(\vec{s}\vec{i})}{\vec{s}\vec{k}},\frac{[\vec{r}\vec{s}\vec{i}]+z_0(\vec{s}\vec{j})}{\vec{s}\vec{k}}\right), \tag{5.12}$$

where $\vec{r}=\vec{r}(u,v)$ is the parametrization of the surface and $\vec{s}=\vec{s}(u,v)$ is defined in (4.1).

EXAMPLE 5.2. The reflection of the circle $x^2+y^2=c^2$ (e.g., the speedometer on the dashboard of a car) is the mapping of the implicit curve

$$\left([\vec{s}\vec{r}\vec{j}]+z_0(\vec{s}\vec{i})\right)^2+\left([\vec{r}\vec{s}\vec{i}]+z_0(\vec{s}\vec{j})\right)^2=c^2\cdot(\vec{s}\vec{k})^2 \tag{5.13}$$

from the parameter domain to the surface (e.g., the windshield of the car).

Remarks. (1) The geometry of the scene is captured by the function $f(u,v)$ which depends on the surface parametrized by $\vec{r} = \vec{r}(u,v)$, the viewpoint A contained in the expression of $\vec{s}$ (see (4.1)), the direction of the light lines $\vec{j}$, and the light plane with the normal $\vec{k}$. The result can be easily generalized for more general directions of the light lines and the light plane; the function $f(u,v)$ will be a little more complicated in this case.

(2) The result that "RCs are contour curves" is interesting from a theoretical point of view. For practical purposes, it is particularly useful due to the possibility of implementation of available numerical contouring algorithms (see [1] and the references therein). An even more straightforward implementation is possible in Matlab; by using the built-in *contourc* and *contour* Matlab functions for the computation and the rendering of the contour curves, we obtain the desired RCs on the surface. Figures 5.1 and 5.2 were generated with a Matlab program implemented by Ulrich Reif.

6. Texture reflection lines (TRLs). We were inspired to investigate this type of RLs as a surface interrogation tool by the graphical features included in the OpenInventor toolkit [13] developed by Silicon Graphics. This analysis uses the OpenInventor texture and environment mapping implementation and has some kind of experimental flavour. Though this approach is trivial from the theoretic point of view, it has not been used extensively in industrial implementations of RLs yet. Some practicioners used it in an early stage of surface interrogation. Kinks were detected quite accurately and that was our main purpose.

Roughly speaking, our OpenInventor program works like this. First, a set of big circles are generated on an "environment" sphere using texture mapping (by "projection" of a texture of parallel lines). The north pole of the sphere is situated in the upper part of the scene. This pattern is then "projected" on the geometry within the sphere environment (e.g., the surfaces) by environment mapping. (See [8] for details.) The program was developed under Windows and ported to UNIX (tested on SGI IRIX and HP-UX).

The pictures in Figures 6.1 and 6.2 show TRLs on C^1 and C^2 rotation surfaces. The generating curves of the surfaces were created by cubic C^1 and C^2 point interpolation [3]. The topology of RLs is sometimes intriguing and not easy to comprehend, particularly for this type of RLs; the main issue, however, is their sensitivity to the smoothness of the surface, which is revealed by the existence or nonexistence of kinks for the C^1 and C^2 surfaces, respectively.

The display of the differences between C^1 and C^2 continuity depends on the quality of the OpenInventor texture mapping implementation [8]. In some cases, the results are not always convincing to a visual examination. They can be emphasized by zooming relevant regions and viewing them from an appropriate direction. This applies particularly to kinks. Kinks of the TRLs can be seen in the left upper part of the C^1 surface in Figure 6.1; none are seen on the TRLs of the C^2 surface in Figure 6.2 as expected. These differences can be emphasized even more in our program by toggling between the two figures (actually by toggling the interpolation method).

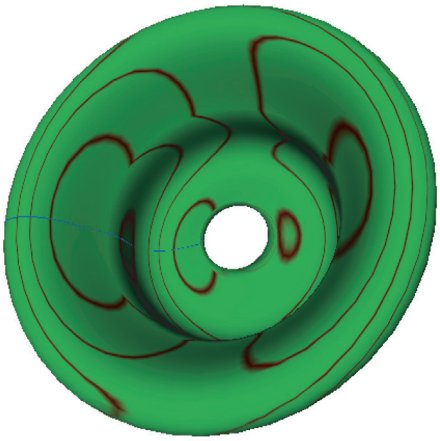

FIGURE 6.1. RLs with kinks.

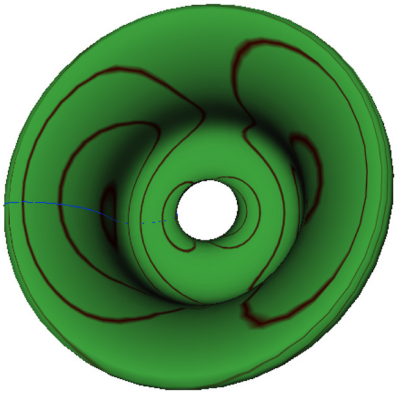

FIGURE 6.2. Smooth RLs.

7. A brief comparison of methods. Several tests showed that the Matlab-based solutions are more accurate, particularly the one using the main result of this note: "RCs are contour curves." For the implementation, we took great advantage of the Matlab library function contourc. The OpenInventor solution is accurate in the large, being able to detect kinks for most of the tested surfaces. But it did not perform as good as the Matlab-based solution. It seems desirable to combine the strengths of the OpenInventor graphics with the above-mentioned mathematical result. Therefore, I recommend an OpenInventor (C++) implementation of RLs as contours of the function(s) given in Theorems RL and RC.

8. Conclusion. In this note, a series of new concepts and approaches related to reflection lines (RLs) on free-form surfaces were presented. Physical reflection lines (PRLs)

and simplified raytracing reflection lines (RRLs) were reviewed. New straightforward implementations in Matlab which use built-in Matlab functions were described. A new theoretical result was derived, which links RLs to contour curves (CRLs); this result is applied to a Matlab implementation which uses Matlab routines for the computation and rendering of CRLs. The result can also be used for implementations based on available contouring algorithms if a Matlab-independent environment is desired. The computational part of the Matlab program can also eventually be "translated" into C/C++ code using the recently released Matlab to C/C++ converter and Math Library. Finally, a new type of RLs, texture reflection lines (TRLs), is generated using the OpenInventor texture and environment mapping implementations. This allows the computation, rendering, and animation of reflection curves (RCs) at interactive rates in a high-end graphics environment, which makes it particularly interesting for industrial applications.

The new concepts and techniques described in this note are suited for industrial applications in different environments and subject to various quality requirements. Future research will eventually address the suitability of the different types of RLs for surface smoothing and aesthetical analysis.

Acknowledgments. This research is part of a project initiated at the University of Esslingen with the cooperation of Prof. Dr. Binh Pham from the Qeensland University of Technology and Prof. Dr. Ulrich Reif from the Technische Hochschule Darmstadt, who implemented "contour reflection lines" in Matlab in a more general theoretical setting than the one described in Section 5. This project was partly supported by the German Academic Exchange Program DAAD and the LARS Program of the State of Baden Wuerttemberg for the OpenInventor Implementation by Michael Arnold. We also acknowledge the interesting interviews and practical demonstrations of several professionals from the CAD/CAM Development Department of the DaimlerChrysler Company, particularly the team of Dr. Ekkehart Kaufmann, one of the authors of [5], which gave us a valuable insight on the usefulness of reflection techniques for design work.

References

[1] K.-P. Beier and Y. Chen, *Highlight-line algorithm for realtime surface-quality assessment*, Computer-Aided Design **26** (1994), no. 4, 268–277.

[2] Y. Chen, K.-P. Beier, and D. Papageorgiou, *Direct highlight line modification on NURBS surfaces*, Comput. Aided Geom. Design **14** (1997), no. 6, 583–601.

[3] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, 4th ed., Academic Press, Boston, 1996.

[4] J. Hoschek and D. Lasser, *Grundlagen der geometrischen Datenverarbeitung* [*Foundations of Geometric Data Processing*], B. G. Teubner, Stuttgart, 1989.

[5] E. Kaufmann and R. Klass, *Smoothing surfaces using reflection lines for families of splines*, Computer-Aided Design **20** (1988), no. 6, 312–316.

[6] R. Klass, *Correction of local surface irregularities using reflection lines*, Computer-Aided Design **12** (1980), no. 2, 73–77.

[7] J. Loos, G. Greiner, and H.-P. Seidel, *Computing surface geometry from isophotes and reflection lines*, Tech. Report 4, University of Erlangen, Erlangen, Germany, 1997.

[8] J. Neider, T. Davis, and M. Woo, *Open GL Programming Guide*, Addison-Wesley, Massachusetts, 1993.

[9] E. Pärt-Enander, A. Sjöberg, B. Melin, and P. Isaksson, *The MATLAB Handbook*, Addison-Wesley, Harlow, 1996.

[10] B. Pham and Z. Zhang, *Correction of reflection lines using genetic algorithms*, Second Asia-Pacific Conference on Simulated Evolution and Learning (Canberra, 1998), Lecture Notes in Computer Science, vol. 1585, Springer-Verlag, Heidelberg, 1999, pp. 26–33.
[11] T. Poeschl, *Detecting surface irregularities using isophotes*, Comput. Aided Geom. Design **1** (1984), no. 1, 163–168.
[12] H. Theisel and G. Farin, *The curvature of characteristic curves on surfaces*, IEEE Computer Graphics and Applications **17** (1997), no. 6, 88–96.
[13] J. Wernecke, *The Inventor Mentor*, Addison-Wesley, Massachusetts, 1994.

Dan-Eugen Ulmet: Mathematical Institute, University of Applied Sciences, 73728 Esslingen, Germany
E-mail address: dan-eugen.ulmet@fht-esslingen.de