REAL-TIME RENDERING OF CUT DIAMONDS

paper category: process

(author 1) (author 2)

1 Abstract

We present a method to create in real time computer-generated images of cut diamonds of convex polyhedral shape. The method is based on beam tracing and models the complete underlying physics of Fresnel's equations and polarization tracing. To achieve high computational efficiency, we linearize the non-linear refraction and take advantage of the convex shape.

We analyze the scaling of our method with increasing depth of recursive beam tracing. We compare results produced by our method with those generated by ray tracing. For typical scenes and image resolutions of approximately 800-by-800 pixels, the speed-up factor of our method compared to ray tracing is about 1000 to 5000, depending on recursion depth, with the obtained images being almost identical.

Our method produces photo-realistic images, which we demonstrate by comparison of photographs taken under carefully controlled lighting conditions to images produced by our method when simulating the same conditions.

2 Introduction and Motivation

We describe a highly efficient method for real-time and near-photorealistic rendering of cut diamonds. Figure 1 shows high-quality images produced by our method.

Our work was motivated by the desire to develop an interactive tool supporting the rapid generation of imagery of cut diamonds. The primary objective was to devise a novel approach that is as efficient as possible and can support truly interactive rendering of diamonds, allowing a user to fully explore the "parameter space" given by geometry, lighting and camera. We have achieved this objective. The system that we have created allows a user to specify and change easily the parameters controlling optical/geometrical diamond characteristics and the settings for our camera viewing model. Our main objectives in designing our system were to devise an algorithm that

- is in accordance with the physical laws of reflection and refraction (Snell's law and Fresnel's equations), and
- leads to a computationally highly efficient implementation producing high-quality renderings.

We briefly review the essential physical laws underlying our approach; describe how these laws are implemented in the design of our algorithm; and compare the results produced by our prototype with those obtained by ray tracing and with actual digital photographs.

In computer graphics, ray tracing is the proto-typical method used for generating digital images of three-dimensional scenes. Effects like reflection and refraction can be modelled well with a ray tracing approach. Nevertheless, to generate "correct" ray-traced images it is crucial to embed the whole physics of reflection and refraction coefficients in one's ray tracing algorithm - and in the

Fig. 1: Renderings of diamonds obtained with our method

definition of all objects constituting a scene. In general, such a rigorous, physics-based approach is currently not applicable to scenes containing thousands of complicated objects with different optical properties. Our work was motivated by the goal to demonstrate that it is possible to produce "correct" images of cut diamonds, and to document that this goal can be achieved on today's commodity PCs using standard graphics cards. With our current implementation it is possible for a user to obtain high-quality renderings of cut diamonds at interactive frame rates, i.e. he/she can vary rendering or diamond parameters and obtain a rendering instantly.

Our approach to efficient rendering of cut diamonds is based on the principle of beam tracing, see [Heckbert and Hanrahan 1984]. Ray tracing, being an image-/screen-space approach requires us to shoot rays through every screen pixel; beam tracing can lead to substantial accelerations over standard ray tracing by using entire "bundles" (beams) of rays instead. We have adapted beam tracing for cut diamond rendering.

There are numerous ray tracing packages that use Snell's law to compute refractions. However, the reflection and refraction coefficients, given by Fresnel's equations, are not always implemented, as their computation is expensive. To compare the efficiency of the implementation of our method with a typical commonly used ray tracing package, we chose to use MegaPOV [MegaPOV], an extension of the popular ray tracer 'Persistence of Vision' [POV-Ray], for comparison, as it makes use of Fresnel's equations. We have compared computational efficiency and quality of rendering results. Subject to the chosen image resolution, it has turned out that our implementation can produce results highly similar to those created with MegaPOV - with speed-up factors of several orders of magnitudes. To validate that our implementation of the physical laws and our mathematical simplifications are also in agreement with physical reality, we have performed comparison with digital photographs. Our experiments have confirmed that our implementation can produce highly realistic renderings.

3 Method

We consider a convex polyhedron, in the remainder of this paper referred to as *stone*, made of a material of isotropic index of refraction. This definition comprises almost all relevant cases, as with the exception of the "heart" shape, all shapes commonly used for jewellery are convex polyhedra. Both diamond and cubic zirconia, the best commercial substitute for diamond, are materials with an isotropic index of refraction. The stone's faces will be referred to as *facets*. Due to the polyhedron's convexity, all facets are convex polygons.

First, we describe the individual components of our method, and then we show how they are used in the main algorithm.

3.1 Beam tracing

Ray tracing is known to produce very realistic images. With the appropriate models for local interaction of light and objects, many physical effects can be simulated with high accuracy. Unfortunately, ray tracing is also very slow, as every pixel of the rendered image has to be computed individually.

Beam tracing [Heckbert and Hanrahan 1984] builds upon the idea of enlarging the sampling area. As in the case of ray tracing, light is traced backwards, from the observer's eye to the objects in the scene and finally towards the light sources. Instead of infinitesimally thin rays (point sampling), beams with finite extension are used (area sampling). In this paper, a *beam* is understood as a cone with convex polygonal cross-section (Fig. 2). A point **v** (the location of the observer's eye) and a convex polygon *p* define a beam.

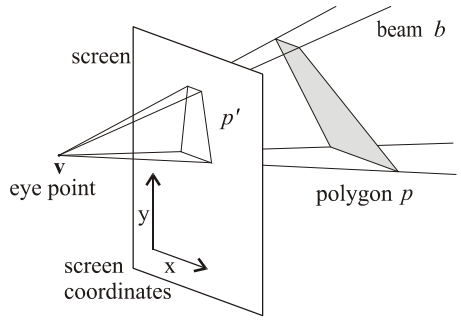


Fig. 2: A beam *b* is defined by an eye point **v** and a polygon *p*. Instead of using *p*, we use the screen-projected polygon *p'* to define the beam *b*.

For a given beam, there is no unique polygon that defines the beam. Any intersection of the beam with a plane can be used. By establishing a reference plane (the "screen") with a coordinate system and using the intersection of screen and beam as defining polygon, we can define beams in terms of two-dimensional screen coordinates. In OpenGL [Shreiner 1999], the mapping sc_{OpenGL} that converts three-dimensional object points to two-dimensional screen coordinates is

$$sc_{OpenGL} = pd \circ mv \circ hg$$

with

$$hg : \begin{pmatrix} x & y & z \end{pmatrix} \mapsto \begin{pmatrix} x & y & z & 1 \end{pmatrix}$$

the transformation to homogeneous coordinates,

$$mv : \mathbf{x} \mapsto \mathbf{M} \cdot \mathbf{x}, \ \mathbf{M} \in \mathbf{R}^4 \times \mathbf{R}^4$$

the product of the 'modelview' and 'projection' matrices and

$$pd_{OpenGL} : \begin{pmatrix} x & y & z & w \end{pmatrix}^t \mapsto \tfrac{1}{w} \begin{pmatrix} x & y & z \end{pmatrix}^t$$

the perspective division. We use the same kind of mapping, but in a slightly modified notation and omitting the depth component in the perspective division, as we will not need it. Our mapping *sc* to screen coordinates is

$$sc = pd \circ cm$$

with

$$cm : \mathbf{x} \mapsto \mathbf{A} \cdot \mathbf{x} + \mathbf{b}, \ \mathbf{A} \in \mathbf{R}^3 \times \mathbf{R}^3, \mathbf{b} \in \mathbf{R}^3$$

and

$$pd : \mathbf{x} \mapsto \frac{1}{\mathbf{x}^t \cdot \mathbf{w} + c} \begin{pmatrix} x \\ y \end{pmatrix}, \mathbf{w} \in \mathbf{R}^3, c \in \mathbf{R}.$$

For given OpenGL modelview and projection matrices, we can compute $\mathbf{A}, \mathbf{b}, \mathbf{w}$ and c to yield the same mapping.

In other words, we use OpenGL-like screen coordinates to define beams and compute intersections of polygons with beams. However, this is not done by the OpenGL subsystem, but directly in the algorithm.

The concept of beams is well suited for scenes consisting of convex polygons, as the intersection of a beam and a convex polygon yields a convex polygon, which is used to define a child beam with the same origin [Ghazanfarpour and Hasenfratz 1998]. In this framework, reflections are easily incorporated, as they are linear transformations, transforming polygons into polygons and beams into beams.

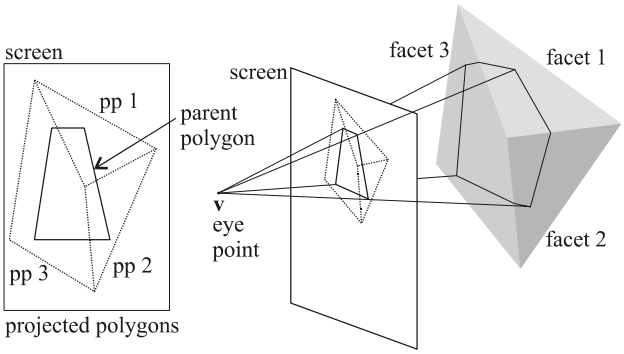


Fig. 3: The intersection of beam with facets. The computation of the intersection is done in terms of screen coordinates, which reduces the calculation of the beam-polygon-intersection to the intersection of two convex polygons.

3.2 The beam subdivision

As explained in the preceding section, we define beams in terms of two-dimensional screen coordinates. This simplifies the calculation of the intersection of a beam with a facet, as it can be done in screen coordinates. Figure 3 shows the three-dimensional arrangement and the corresponding screen-projected scene. As the mapping *sc* consists of a linear transformation and a perspective division, the projection to screen transforms convex polygons (the facets) into convex polygons (the "projected polygons" in Fig. 3). As the "parent polygon" that defines the beam is convex too, we have to compute the intersection of two convex polygons, which is easy and can be done rather fast.

In order to avoid the projection of facets that do not yield an intersection with the parent polygon, we start with a facet that is known to yield an intersection. We determine a suitable start facet by constructing a ray in the center of the beam and determining the facet where it leaves the stone.

While actually computing the intersection, all facets adjacent to the facet currently treated are marked for processing, if they must have a non-vanishing intersection with the beam. This can easily be determined from the mutual position of the polygons' edges.

When all facets marked for processing have been projected and their intersections computed, the subdivision of the beam is finished, as (1) there are no further facets with non-vanishing intersection, as the stone is known to be convex, and (2) the sum of the child beams is equal to the beam itself, as the surface of the stone is closed. The second statement can be easily understood by realizing that when tracing the beam, we are inside the stone, thus there is no way of leaving the stone without crossing its surface.

3.3 The reflection mapping r_f

The reflection upon a facet f with plane equation $\mathbf{x}^t \cdot \mathbf{n}_f - c_f = 0, \mathbf{n}_f \in \mathbf{R}^3, c_f \in \mathbf{R}$ is given by the mapping

$$r_f : \mathbf{R}^3 \to \mathbf{R}^3, \mathbf{x} \mapsto \left(\mathbf{1} - 2\mathbf{n}_f \cdot \mathbf{n}_f^t\right) \cdot \mathbf{x} + 2c_f \mathbf{n}_f .$$

As it depends only on the plane parameters, it can be rewritten in the more convenient linear form $r_f : \mathbf{x} \mapsto \mathbf{A}_f \mathbf{x} + \mathbf{b}_f$, and the matrix $\mathbf{A}_f$ and the vector $\mathbf{b}_f$ stored for every facet at the start of the beam trace algorithm.

3.4 The refraction mapping

On interfaces of materials with different index of refraction, light gets refracted according to Snell's law, given as

$$n_1 \sin\alpha_1 = n_2 \sin\alpha_2 .$$

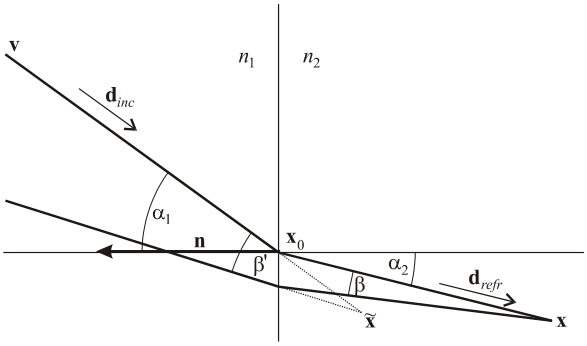


Fig. 4: The refraction of an incident ray on a plane interface between media with indices of refraction $n_1 < n_2$. For a viewer at position **v**, light emitted from a point **x** inside medium 2 seems to come from a virtual image point $\tilde{\mathbf{x}}$.

We consider a plane interface of two materials with indices of refraction $n_1 < n_2$ (Fig. 4). For a viewer at point **v**, light originating at point **x** inside material 2 seems to come from the virtual image point $\tilde{\mathbf{x}}$. For a fixed viewing position **v** and in the limit $\beta \to 0$, i.e., for small opening angles around the main ray, the relation between **x** and $\tilde{\mathbf{x}}$ is well defined. We refer to this mapping as

$$m_{refr} : \mathbf{R}^3 \to \mathbf{R}^3, \mathbf{x} \mapsto \tilde{\mathbf{x}} .$$

For non-vanishing opening angles β, there is no single virtual image point. In optics, the resulting image faults are called "aberrations". In this paper, we neglect the effects of aberration.

As m_{refr} is nonlinear and computationally expensive, we linearize it by Taylor series expansion. Using the Taylor series of m_{refr} at point $\mathbf{x}_0$

$$m_{refr}(\mathbf{x}) = m_{refr}(\mathbf{x}_0) + m'_{refr}(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0) + \cdots$$

we define a linear projection *t* ("transmission")

$$t : \mathbf{R}^3 \to \mathbf{R}^3, \mathbf{x} \mapsto \mathbf{A}_t \cdot \mathbf{x} + \mathbf{b}_t$$

with

$$\mathbf{A}_t := m'_{refr}(\mathbf{x}_0), \quad \mathbf{b}_t := m_{refr}(\mathbf{x}_0) - m'_{refr}(\mathbf{x}_0) \cdot \mathbf{x}_0 .$$

The development point $\mathbf{x}_0$ has to be on the interface in order to assert continuity: As points in medium 1 do not get refracted (the light does not cross the refracting boundary between medium 1 and 2), they are mapped onto themselves. In order to yield an overall continuous mapping, *t* must map the points in the interface plane onto themselves, just as m_{refr} does. This is asserted by choosing a development point on the interface. The most suitable choice for $\mathbf{x}_0$ is the center of the facet.

At the interface, m_{refr} is continuous but its derivative is not continuous, thus, strictly speaking, the term $m'_{refr}(\mathbf{x}_0)$ is a single-sided derivative.

For the computation of $m'_{refr}(\mathbf{x}_0)$, we are free to choose a suitable base $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ and construct $m'_{refr}(\mathbf{x}_0)$ from the directional derivatives along the base vectors. For $\mathbf{b}_1$ and $\mathbf{b}_2$, we use the plane vectors of the interface plane, as it is very simple to compute the directional derivatives for them.

The choice of $\mathbf{b}_3$ and the computation of the directional derivative is explained in Figure 4. From the direction of incidence $\mathbf{d}_{inc}$, defined by **v** and $\mathbf{x}_0$, and Snell's law, we obtain the direction $\mathbf{d}_{refr}$ of the refracted ray. We choose a virtual image point $\tilde{\mathbf{x}} = \mathbf{x}_0 + \varepsilon \mathbf{d}_{inc}$ with using a small value for ε and a small angle β' to define a second ray, compute the refraction of this second ray and get the point **x** as the intersection point of the two

refracted rays. Thus, the directional derivative along $\mathbf{b}_3 := \mathbf{x} - \mathbf{x}_0$ is $\tilde{\mathbf{b}}_3 = \tilde{\mathbf{x}} - \mathbf{x}_0$, and we finally get the derivative $m'_{refr}(\mathbf{x}_0)$ as

$$m'_{refr}(\mathbf{x}_0) = \begin{pmatrix}\mathbf{b}_1 & \mathbf{b}_2 & \tilde{\mathbf{b}}_3\end{pmatrix} \cdot \begin{pmatrix}\mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3\end{pmatrix}^{-1}.$$

Both $\mathbf{A}_t$ and $\mathbf{b}_t$ are therefore fixed, and the mapping t is determined. This computation must be done only once for every facet directly visible from the viewer.

3.5 Fresnel's equations

Snell's law describes the refraction angle, but not the intensities of the reflected and the refracted light components. These intensities are given by Fresnel's equations [Jackson 1999].

Using the quantities and sign conventions as shown in Figure 5, for the light component with an electric field orthogonal to the plane of incidence, the reflection and transmission coefficients are given by

$$r_{12,orth} := \frac{E_{1,orth}}{E_{0,orth}} = \frac{n_1 \cos\alpha_1 - n_2 \cos\alpha_2}{n_1 \cos\alpha_1 + n_2 \cos\alpha_2}$$
$$t_{12,orth} := \frac{E_{2,orth}}{E_{0,orth}} = \frac{2n_1 \cos\alpha_1}{n_1 \cos\alpha_1 + n_2 \cos\alpha_2}.$$

For the electric field parallel to plane of incidence these coefficients are given by

$$r_{12,par} := \frac{E_{1,par}}{E_{0,par}} = \frac{n_2 \cos\alpha_1 - n_1 \cos\alpha_2}{n_2 \cos\alpha_1 + n_1 \cos\alpha_2}$$
$$t_{12,par} := \frac{E_{2,par}}{E_{0,par}} = \frac{2n_1 \cos\alpha_1}{n_2 \cos\alpha_1 + n_1 \cos\alpha_2}.$$

In the case of total internal reflection, the reflection coefficients become complex. Their absolute value is one, as all light gets reflected. Thus, these coefficients have the form $r_{TIR,orth} = e^{i\delta_{orth}}$ and $r_{TIR,par} = e^{i\delta_{par}}$. For our application, we do not need the absolute phase, as we consider only incoherent light. Only the phase difference $\delta = \delta_{orth} - \delta_{par}$ is relevant. The equation for δ is given in [Born and Wolf 1999].

For the computation of the reflected and transmitted intensities, the polarization state has to be known. Algorithms that do not track the polarization state usually assume unpolarized light and use the reflection and transmission coefficients

$$r_{12,unpol} = \tfrac{1}{2} r_{12,orth} + \tfrac{1}{2} r_{12,par}$$
$$t_{12,unpol} = \tfrac{1}{2} t_{12,orth} + \tfrac{1}{2} t_{12,par}.$$

3.6 Polarization tracking

As the reflection and refraction coefficients given by Fresnel's equations differ for the parallel and orthogonal component, the polarization state of light changes due to the reflections and refractions. Completely unpolarized light can get completely linearly polarized by a single reflection (the angle of incidence for this case is known as Brewester's angle). As there are many reflections and refractions to take into account when tracing light through the stone, the polarization state of the light must be tracked. There are several ways to represent the polarization state of light: The Stokes parameters, the coherence matrix and the Jones calculus. Each of these ways has its own distinct advantages [Chipman 1995, Wolff and Kurlander 1990].

For the application of cut-diamond rendering, the most suitable representation is the coherence matrix: It can handle partially polarized light and lends itself to easy transformation. For a given light ray, an orthogonal coordinate system is established with the

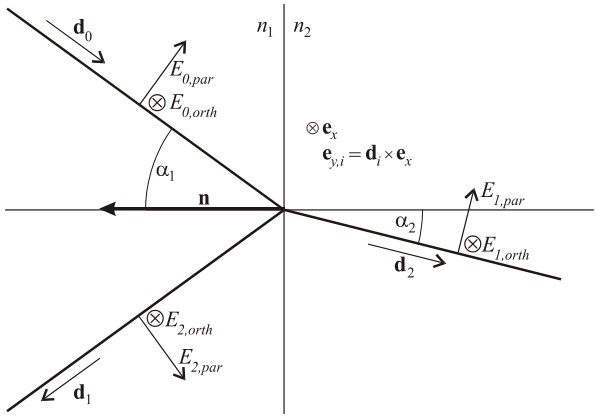


Fig. 5: Adopted sign convention for electic field used in Fresnel's equations.

z-axis corresponding to the direction of the light propagation. The coherence matrix **J** is defined as

$$\mathbf{J} = \left\langle \begin{pmatrix}E_x \\ E_y\end{pmatrix} \cdot \begin{pmatrix}E_x & E_y\end{pmatrix}^* \right\rangle = \begin{pmatrix}\langle E_x E_x^* \rangle & \langle E_x E_y^* \rangle \\ \langle E_y E_x^* \rangle & \langle E_y E_y^* \rangle\end{pmatrix},$$

with E_x and E_y denoting the components of the electric field with respect to the coordinate system, and the angular brackets denoting time averaging. The coherence matrix does not include E_z, as there is no electric field in the direction of propagation.

The light intensity is given by the trace of the coherence matrix $tr\,\mathbf{J} = J_{xx} + J_{yy}$, while the determinant gives the degree of polarization. For details on how to interpret the coherence matrix, see [Born and Wolf 1999].

In order to apply the reflection and transmission coefficients, the coordinate system must be rotated such that its x-axis is aligned with the directions orthogonal to the plane of incidence and the y-axis parallel to the plane of incidence. With respect to a coordinate system that is rotated around the z-axis by the angle φ, the electric field is given as

$$\begin{pmatrix}E'_x \\ E'_y\end{pmatrix} = \underbrace{\begin{pmatrix}\cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi\end{pmatrix}}_{\mathbf{m}_{rot} :=} \cdot \begin{pmatrix}E_x \\ E_y\end{pmatrix}.$$

Now it is possible to apply the transmission or reflections coefficients. The reflected part is given as

$$\begin{pmatrix}E_x \\ E_y\end{pmatrix}_{refl} = \underbrace{\begin{pmatrix}r_o & 0 \\ 0 & r_p\end{pmatrix}}_{\mathbf{m}_{refl} :=} \cdot \begin{pmatrix}E'_x \\ E'_y\end{pmatrix} = \mathbf{m}_{refl} \cdot \mathbf{m}_{rot} \cdot \begin{pmatrix}E_x \\ E_y\end{pmatrix}$$

or, in the case of total internal reflection,

$$\mathbf{m}_{refl} = \begin{pmatrix}e^{i\delta} & 0 \\ 0 & 1\end{pmatrix}$$

with δ the phase difference defined in section "Fresnel's equations". The transmitted part is

$$\begin{pmatrix}E_x \\ E_y\end{pmatrix}_{trans} = \underbrace{\sqrt{\frac{\cos\alpha_2}{\cos\alpha_1}}\begin{pmatrix}t_o & 0 \\ 0 & t_p\end{pmatrix}}_{\mathbf{m}_{trans} :=} \cdot \begin{pmatrix}E'_x \\ E'_y\end{pmatrix} = \mathbf{m}_{trans} \cdot \mathbf{m}_{rot} \cdot \begin{pmatrix}E_x \\ E_y\end{pmatrix}.$$

Actually, the cos factor does not belong to the transformation of the electric field, but to the transformation of the light power (energy per time): The light power for the sampling area changes proportionally to the ratio $\cos\alpha_2 / \cos\alpha_1$ due to the change of the

beam's cross-section. Instead of storing this factor and applying it when computing the light power, we apply its root in the transformation of the electric field. As a consequence, when computing the coherence matrix from the transformed electric field, it has already been taken into account.

3.7 Depth of field

The imaging of small objects tends to go hand in hand with a considerable depth-of-field: Due to their size, small objects emit little light. In order to capture enough light for a photograph the entry lens must be large, i.e., must have large numerical aperture. A large numerical aperture yields a small depth of field.

As cut diamonds usually are relatively small, most photographs of cut diamonds exhibit a pronounced depth of field. We are used to this depth of field in diamond pictures, and it is this phenomenon that makes "crisp" pictures look unrealistic to the human observer. For added realism, we must simulate depth of field.

There is a straightforward approach to simulate depth of field: We can position the (pin-hole) camera at several locations in the lens (with adjusted viewing direction) and calculate the average of the images. This approach amounts to rendering the same scene several times with a slightly modified camera, and computing the average of all pictures. In OpenGL, this calculation is done in the accumulation buffer [Shreiner 1999], as its increased color depth avoids rounding errors. For a smooth-looking depth of field, many camera positions are necessary, typically 30 to 200.

3.8 Parallel and perspective refraction

The accumulated projection transformation m_{acc} for every beam is of the form $\mathbf{x} \mapsto \mathbf{A} \cdot \mathbf{x} + \mathbf{b}$, with $\mathbf{A} \in \mathbf{R}^3 \times \mathbf{R}^3, \mathbf{b} \in \mathbf{R}^3$. After application of this projection, the perspective division *pd* is applied. For every intersection test, this transformation must be done for all corners of the facet to be projected. Considering overall computational cost, this step requires the execution of a relatively large number of floating-point operations.

We can reduce this computational cost by using "parallel refraction" instead of the refraction mapping *t* defined above. The idea is depicted in Figure 6. Instead of using *t*, we use the mapping

$$t_{parallel} : \mathbf{R}^3 \to \mathbf{R}^3,\ \mathbf{x} \mapsto \mathbf{A}_t \mathbf{x} + \mathbf{b}_t,$$

with

$$\mathbf{A}_t := \mathbf{1} - \frac{\mathbf{n} \cdot \mathbf{d}_{refr}^t}{\mathbf{d}_{refr}^t \cdot \mathbf{n}},\ \mathbf{b}_t := c \frac{\mathbf{d}_{refr}}{\mathbf{d}_{refr}^t \cdot \mathbf{n}}$$

where $\mathbf{n}$ and c are the parameters of the facet's implicit plane equation $\mathbf{x}^t \cdot \mathbf{n} - c = 0$. This mapping projects all points in the direction $\mathbf{d}_{refr}$ onto the facet plane. Instead of using the overall projection $pd \circ cm \circ t$, we use the projection

$$pd \circ z_{reconst} \circ red_{2d} \circ cm \circ t_{parallel},$$

taking advantage of the possible reduction to two-dimensional space,

$$red_{2d} : \mathbf{R}^3 \to \mathbf{R}^2, (x \quad y \quad z)^t \mapsto (x \quad y)^t,$$

which is compensated by an ensuing "reconstruction mapping" defined as

$$z_{reconst} : \mathbf{R}^2 \to \mathbf{R}^3, (x \quad y)^t \mapsto (x \quad y \quad f_{reconst}(x, y))^t,$$

which re-computes the z-component that is lost in red_{2d}. Reconstruction with the function $f_{reconst}$ is possible due to the fact that all points mapped by $cm \circ t_{parallel}$ lie in a plane defined by the facet plane equation, transformed linearly by *cm*.

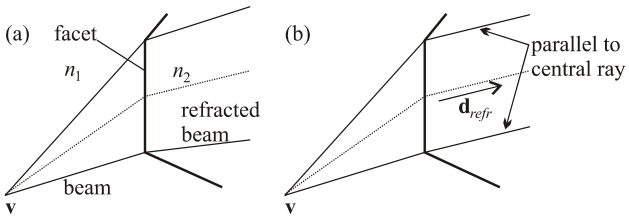


Fig. 6: (a) Perspective refraction and (b) parallel refraction.

We split the overall projection *sc* into two parts, given by

$$sc = \underbrace{pd \circ z_{reconst}}_{:= pd_{reconst}} \circ \underbrace{red_{2d} \circ cm \circ t_{parallel}}_{=: m_{red}},$$

using a modified perspective division $pd_{reconst} : \mathbf{R}^2 \to \mathbf{R}^2$ and a linear mapping $m_{red} : \mathbf{R}^3 \to \mathbf{R}^2,\ \mathbf{x} \mapsto \mathbf{A}_{red} \cdot \mathbf{x} + \mathbf{b}_{red}$ with $\mathbf{A}_{red} \in \mathbf{R}^2 \times \mathbf{R}^3, \mathbf{b}_{red} \in \mathbf{R}^2$.

The crucial idea for the reduction of computational cost is to use the mapping m_{red} instead of *sc* for the intersection test. This approach reduces the number of floating-point calculations for the linear part by one third and leaves out the perspective division completely. Only during the last stage of our algorithm, when we draw to the screen, do we compute the actual screen coordinates by $pd_{reconst}$. As many more coordinates are projected for the intersection test than coordinates drawn on the screen, the delayed perspective division reduces the number of floating-point operations as well.

To summarize: Instead of using screen coordinates, we use pre-perspective-division coordinates resulting from m_{red} for the computation of the intersections. This approach reduces the number of floating-point operations by more than one third.

In principle, a problem can occur during the reconstruction function $f_{reconst}$, as it can lead to a division by zero. In practice, the probability of this situation to occur is so small that it can be neglected. Of course, due to the missing perspective division, polygons that are benign in screen coordinates can be almost degenerate in pre-perspective-division coordinates. However, the intersection test for the subdivison of beams must be able cope with almost degenerate polygons, and as the opposite situation is just as likely to occur (polygons benign in pre-perspective-division coordinates, but almost degenerate in screen coordinates), parallel refraction is numerically not more sensitive than perspective refraction.

The only real issue that remains is this one: Due to the missing perspective division, sometimes the orientation of projected polygons is wrong (i.e., the polygon's corners are ordered in mathematical negative instead of positive sense). As the intersection algorithm relies on a fixed orientation, for polygons oriented wrongly, we use red_{2d} with the y-component reversed, perform all computations with this projection, and finally reverse the y-component when drawing, i.e., just before the perspective division $pd_{reconst}$.

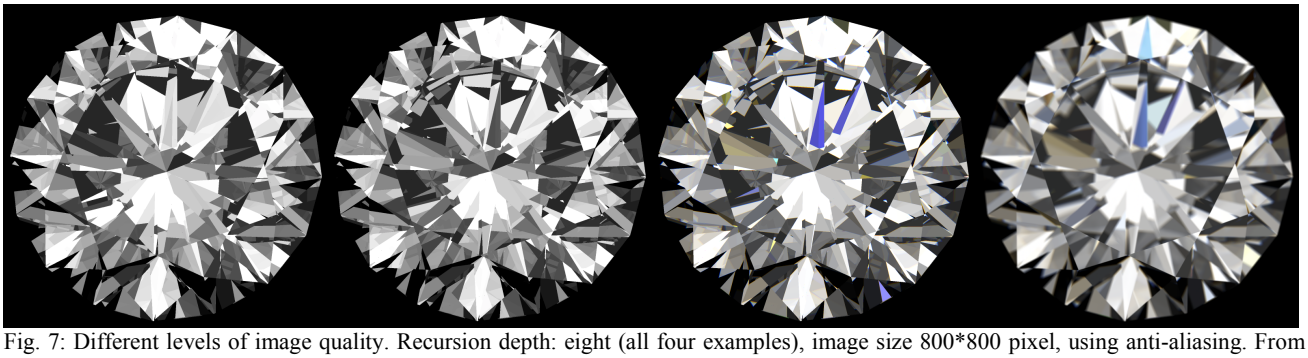


Fig. 7: Different levels of image quality. Recursion depth: eight (all four examples), image size 800*800 pixel, using anti-aliasing. From left to right: (a) Rendered with parallel refraction (65.5 ms); (b) rendered with perspective refraction (80.9 ms); (c) same as (b), with addition of spectral sampling (three wavelengths) (236 ms); (d) same as (c), with addition of depth of field (91 camera positions, 65.2 s).

3.9 The main algorithm

for (every facet f directly visible) {

- create beam of depth 1 by projecting facet f according to $pd \circ cm$, the projection to screen coordinates;
- create local coordinate system and compute change $\mathbf{m}_{rot,camera}$ from local to camera coordinate system;
- compute direction and coefficients of reflection;
- compute polarization transformation for outbound (reflected) beam: $p_{acc,out} = \mathbf{m}_{rot,camera} \circ \mathbf{m}_{refl} \circ \mathbf{m}_{rot,light\,source}$
- apply $p_{acc,out}$ to light emanating from light source, compute and store coherence matrix;
- for the computation of the child beams, add refraction mapping t_f to projection and store accumulated projection $m_{acc} = cm \circ t_f$;
- set polarization transformation to reflect change of coordinate system and refraction coefficients: $p_{acc} = \mathbf{m}_{rot,camera} \circ \mathbf{m}_{trans}$;

}

for (d=2 to maximal recursion depth) {

for (parent = every beam of depth d-1) {

- project facets according to parent's accumulated projection mapping $m_{acc,parent}$, followed by pd;
- use projected facets to split parent beam into child beams;
- for (every child beam) {
  - compute direction of refraction;
  - create local coordinate system and compute change $\mathbf{m}_{rot}$ from local to parent coordinate system;
  - compute polarization transformation for outbound (refracted) beam $p_{acc,out} = p_{acc,parent} \circ \mathbf{m}_{rot} \circ \mathbf{m}_{trans} \circ \mathbf{m}_{rot,light\,source}$
  - apply $p_{acc,out}$ to light emanating from light source, compute and store coherence matrix;
  - for the computation of child beams, add reflection mapping r_f to projection and store accumulated projection $m_{acc} = m_{acc,parent} \circ r_f$;
  - add change of coordinate system and reflection coefficients to polarization transformation $p_{acc} = p_{acc,parent} \circ \mathbf{m}_{rot} \circ \mathbf{m}_{refl}$

} } }

for (every beam computed) {

- use lighting model to determine light intensity for outbound direction;
- multiply intensity with coherence matrix of outbound direction;
- use resulting intensity to render polygon on screen;

}

4 Results

To evaluate the performance of our method, we have tested it on a PC running at 2GHz, having 512MByte of main memory and an NVidia Ti 4200 graphics card. We have used the graphics card's hardware anti-aliasing (two-by-two super-sampling) to reduce aliasing effects.

4.1 Performance and image quality

Figure 7 shows a typical image obtained with fixed recursion depth, but for different quality levels. The fastest version is parallel refraction, which yields images that are qualitatively correct, but not quantitatively correct (see also Fig. 9f).

To satisfy average rendering demands, in terms of image quality, the best choice is perspective refraction. It is "nearly" correct and has good performance. A further simplification of the refraction leads to visible deviation from the correct image (Fig. 9f), while reducing rendering times by just 5% for small and 25% for large recursion depths (Fig. 8).

Using spectral sampling, i.e., rendering the same scene for different wave lengths, the color effects due to diamond's dispersion [Edwards and Philipp 1985] appear. The rendering times increase linearly with the number of wavelengths used.

The simulation of depth of field increases computational cost linearly with the number of sampling points. As a smooth depth of field usually requires 30 to 200 sampling points, this option is very expensive.

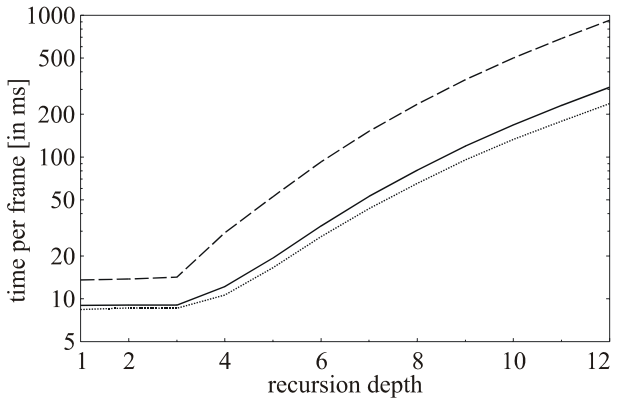


Fig. 8: Scaling of rendering times per frame with recursion depth for the scene shown in Fig. 7, with parallel refraction (dotted line), perspective refraction (solid line) and perspective refraction with three-wavelength spectral rendering (dashed line)

4.2 Beam and ray tracing

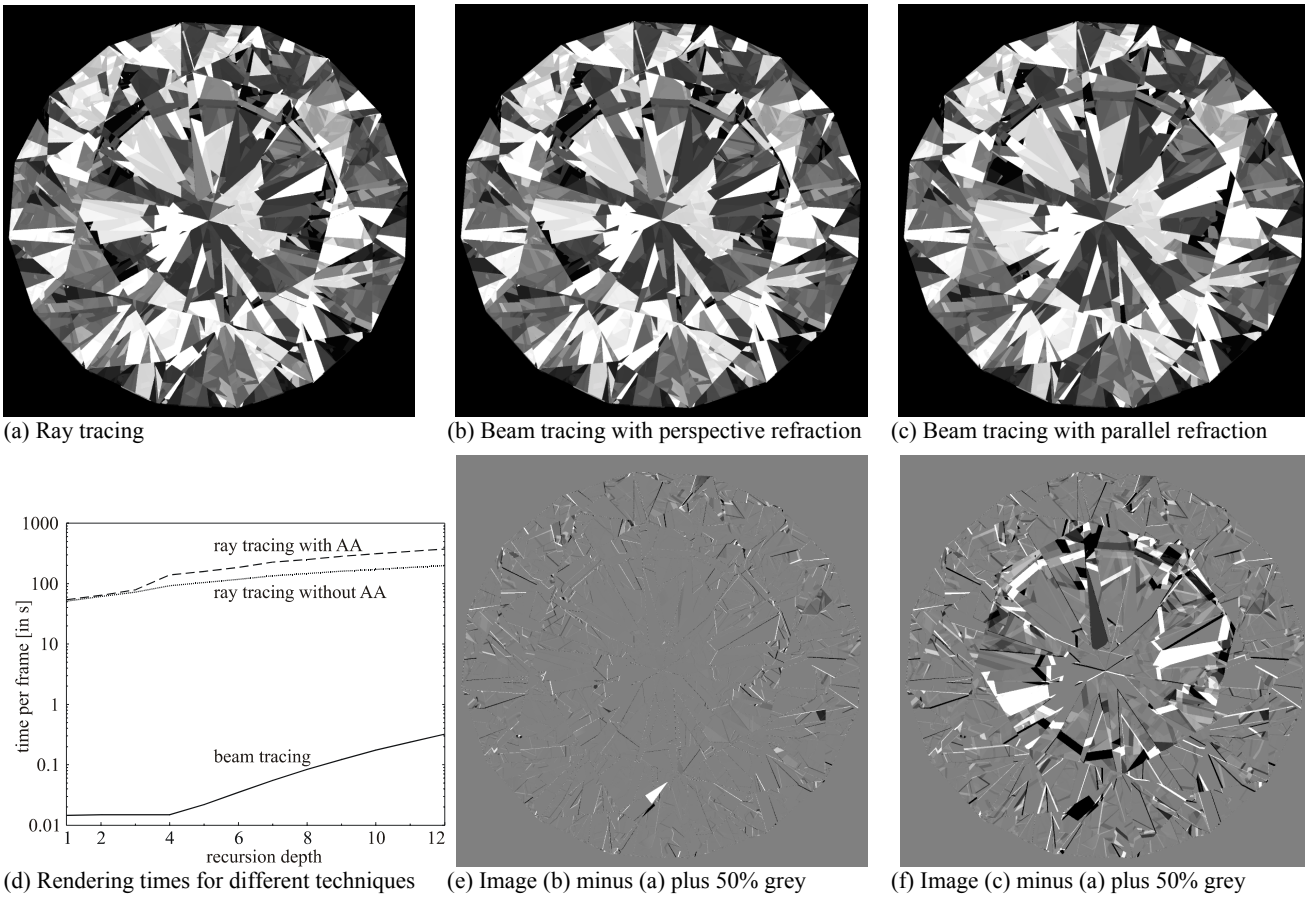
The de facto standard for rendering techniques is usually ray tracing. In this section, we compare ray-traced images with images rendered by our method. For this purpose, the ray tracer used for comparison must be capable of handling Fresnel's equations correctly. Polarization handling is desirable, but not necessary, as we can disable polarization tracking in our method.

We chose to use the ray tracer POV-Ray 3.1 as it is powerful, readily available and an extension called MegaPOV 0.7 is available that includes Fresnel's equations. As it is open-source, we could verify the correct handling of Fresnel's equations in the code. Unfortunately, it does not track polarization, so we had to disable polarization tracking in our method in order to assert comparability. This was done by using the coefficients for unpolarized light given at the end of section "Fresnel's equations" for both the parallel and orthogonal component.

In order to compare ray tracing with our method, we must use a scene that can be set up both by the ray tracers's scene language and our method. While the cut could be converted easily for the ray tracer, the lighting had to be simplified in order to guarantee that both MegaPOV and our method performed the same calculations.

The lighting used for Figure 9 consists of three planes and two spheres. All five objects have homogeneous color and are lit by pure ambient light, thus every point on an objects emits light with the same intensity. The intensities of the five objects are all different. Rendering is done for a single wavelength, i.e., a single index of refraction. Therefore, no colors are visible, as they emerge from dispersion, i.e., a wave length dependent index of refraction.

Beam tracing has the advantage of reducing aliasing [Ghanzanfarpour and Hasenfratz 1998], as it is an area-sampling method in contrast to ray tracing, which is a point-sampling method. In ray tracing, the sampling for the image pixels is done in software. An improvement of image quality by anti-aliasing increases the computational cost (Fig. 9d), whereas for beam tracing, the anti-aliasing can be done in hardware, resulting in a negligible performance reduction.

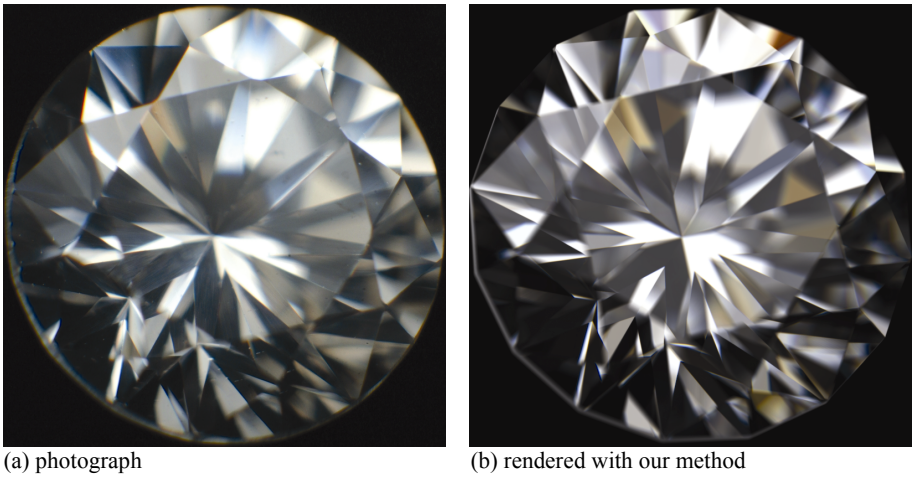


(a) Ray tracing (b) Beam tracing with perspective refraction (c) Beam tracing with parallel refraction

(d) Rendering times for different techniques (e) Image (b) minus (a) plus 50% grey (f) Image (c) minus (a) plus 50% grey

Fig. 9: Comparison of rendering techniques. All images have a size of 800 x 800 pixel and make use of anti-aliasing. (a) through (c) show the same scene, rendered with ray tracing, beam tracing with perspective refraction and beam tracing with parallel refraction. In order to emphasize the non-linear effects of the refraction, the camera location was chosen close the stone (the distance between camera and stone being less than six times the stone's diameter). As all 3 images closely resemble each other, (e) and (f) show the differences between the beam- and the ray-traced images. Fig 9(e) demonstrates that the linearization used for perspective refraction is a very good approximation to the correct non-linear refraction.

Fig 9(d) shows the rendering times per frame for beam tracing with perspective refraction (solid line), ray tracing without anti-aliasing (dotted line) and ray tracing with anti-aliasing (dashed line). For beam tracing, disabling anti-aliasing changes frame rates so little that, in this graph, it would yield the same line as anti-aliased beam tracing. The performance advantage of beam tracing against ray tracing ranges from 1100 to 9300 for high image quality (with anti-aliasing) and from 600 to 6200 for reduced image quality (without anti-aliasing)



(a) photograph (b) rendered with our method

Fig. 10: Comparison of photograph with result produced by our method. (a) Photograph of a cubic zirconia stone of "round brilliant" shape, taken with a microscope under well-controlled lighting conditions. (b) Image rendered by our method with the scene parameters matching the physical microscope setup. Depth of field is enabled with superposition of 217 individual frames. Rendering time: 200.3 s

4.3 Beam tracing and photography

While the comparison with ray tracing is convincing from a computer graphics point of view, the ultimate test is the comparison with real photographs.

The simulation of a real scene with our method is only feasible if the scene is well defined. The most critical part is the lighting. We built a dedicated box with a well-defined window for lighting. The stone was placed at a fixed position in the box and viewed with a microscope. A digital reflex camera was used, as it yields a linear connection between light intensity and the RGB values of the image pixels. The pictures taken with the camera were not altered in any way, they were only cropped to the relevant size.

As no real diamond of appropriate size was available, we used a stone made of cubic zirconia, the best commercial substitute for diamond. Using the index of refraction of cubic zirconia [Wood and Nassau 1982], the data of the microscopes objective, the dimensions of the lighting window and the position of the stone under the microscope's objective, we rendered the same scene with our method.

Figure 10 shows the photograph and the result obtained by our method. Almost all of the reflexes seen in the photograph can be clearly identified in the digitally produced duplicate; many of the colored reflexes have the same color. The differences can be attributed for the most part to the fact that the stone's cut is not perfect, as the location and size of the individual reflexes depend very delicately on the angles between the individual facets.

Though the rendering was done with just three wavelengths (620, 540 and 445 nm for the red, green and blue component, respectively), the colors are qualitatively correct. Apparently, this rather coarse approximation of the full spectral distribution already yields a good approximation to the real image.

5 Conclusions and possible extensions

We have described a method for the rendering of cut diamonds. It is possible to generate near-photorealistic images of cut diamonds, at high resolution and high recursion depth, with our implementation of this method. Today's commodity PCs equipped with contemporary graphics cards are sufficient to generate these images. Concerning the rigorous evaluation of our method we have compared our resulting images with those obtained with a popular ray tracing package and with actual photographs of a cut stone. In summary, our evaluation procedures have demonstrated that our implementation compares very favorably with experiments. The key contributions and specific strengths of our novel approach are:

- Our method is extremely efficient, much more efficient in general than existing competitive techniques we are aware of
- our method does not require any pre-computation steps
- our method is physically correct, as it takes into account Snell's law, Fresnel's equations and polarization
- our method can be used to produce highly realistic renderings of cut diamonds.

Possible extensions are the handling of colored stones by incorporation of depth tracking and the stones' extinction coefficients and the simulation of photographic artefacts like "bleeding" due very strong reflexes. A potentially very interesting application for our method is the mathematical optimization of diamond cuts, as it can be used to construct optimization functions, e.g. for "brilliance" and "fire", which, due to our method being an area sampling method, are continuous and thus lend themselves to numerous optimization algorithms.

6 References

BORN, M. AND WOLF, E. 1999. *Principles of Optics*, 7th ed. Cambridge University Press, Cambridge, U.K.

CHIPMAN, R. A. 1995. Mechanics of polarization ray tracing. *Optical Engineering 34*, 6, 1636-1645

EDWARDS, D. F. AND PHILIPP H. R. 1985. Cubic Carbon (Diamond). In *Handbook of Optical Constants of Solids*. Academic Press, Orlando, Florida. Palik E. D., Ed., 665-673

GHAZANFARPOUR, D. AND HASENFRATZ, J.-M. 1998. A beam tracing method with precise antialiasing for polyhedral scenes. *Computers & Graphics* 22, 1, 103-115.

HECKBERT, P. S. AND HANRAHAN, P. 1984. Beam tracing polygonal objects. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84), 18*, 3, ACM, 119-127.

JACKSON, J. D. 1999. *Classical Electrodynamics*, third edition. John Wiley & Sons, New York.

MEGAPOV. Version 0.7. http://megapov.inetart.net/

POV-RAY. http://www.povray.org/

SHREINER D. 1999. *OpenGL Reference Manual*, third edition. Addison Wesley Longman, Reading, Massachusetts

WOLFF, L. B. AND KURLANDER, D. J. 1990. Ray Tracing with Polarization Parameters. *IEEE Computer Graphics & Applications 10*, 6, 44-55.

WOOD, D. L. AND NASSAU, K. 1982. Refractive Index of cubic zirconia stabilized with yttria. *Applied Optics 21*, 16, 2978-2981.