SCI INSTITUTE
TECHNICAL REPORT

A Ray Tracing based Virtual Reality Framework for Industrial Design

Ingo Wald, Carsten Benthin[‡◇], Alexander Efremov[†], Tim Dahmen[◇], Johannes Günther[†], Andreas Dietrich[‡], Vlastimil Havran[†], Hans-Peter Seidel[†], Philipp Slusallek[‡]

UUSCI-2005-009

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

December 6, 2005

Abstract:

Computer Aided Design (CAD) and Virtual Reality (VR) are becoming increasingly important tools for industrial design applications. Unfortunately, there is a huge and growing gap between what data CAD engineers are working on, what rendering quality is needed by designers and executives to faithfully judge a design variant, and what rendering capabilities are offered by commonly available VR frameworks. In particular, existing VR systems cannot currently cope with the accuracy demanded by CAD engineers, nor can they deliver the photorealistic rendering quality and reliability required by designers and decision makers. In this paper, we describe a ray tracing based virtual reality framework that closes these gaps. In particular, the proposed system supports direct ray tracing of trimmed freeform surfaces even for complex models of thousands of patches, allows for accurately simulating reflections and refraction for glass and car paint effects, offers support for direct integration of measured materials via Bidirectional Texture Functions (BTFs), and even allows for soft environmental lighting from high dynamic range environment maps. All of these effects can be delivered interactively, and are demonstrated on a real-world industrial model, a complete Mercedes C-class car.

[◇] Carsten Benthin and Tim Dahmen area associated with inTrace GmbH, Saarbrücken, Germany

[†] Alexander Efremov, Vlastimil Havran, and Hans-Peter Seidel are members of the Max Planck Institute für Informatik, Saarbrücken, Germany

[‡] Carsten Benthin, Andreas Dietrich, and Philipp Slusallek are affiliated with Saarland University, Saarbrücken, Germany

THE UNIVERSITY OF UTAH

A Ray Tracing based Virtual Reality Framework for Industrial Design

Ingo Wald[×◇] Carsten Benthin[‡◇] Alexander Efremov[†] Tim Dahmen[◇] Johannes Günther[†]
Andreas Dietrich[‡] Vlastimil Havran[†] Hans-Peter Seidel[†] Philipp Slusallek[‡]

[×]SCI Institute
University of Utah
Salt Lake City, UT, USA

[†]Computer Graphics Group
MPI Informatik
Saarbrücken, Germany

[‡]Computer Graphics Group
Saarland University
Saarbrücken, Germany

[◇]inTrace GmbH
Schützenstrasse 3-5
Saarbrücken, Germany

Figure 1: *Several example screenshots from our framework, demonstrated on a complex Mercedes C-Class model: a.) The model consists of 320,000 Bézier patches and thousands of trimming curves that are directly and interactively ray traced without triangulation. b.) The model with ray traced shaders for e.g. glass and car paint, put into some nice environment made up of 200,000 triangles and a captured HDR environment map. Note the accurate reflections, the refraction through the glass, and the smooth shadows from environment lighting. c.) The realistic interior appearance is achieved via a shader supporting Bidirectional Texture Functions (BTFs) of measured samples from the corresponding real-world materials. d.) All these effects work together seamlessly, as can be seen on this view through the side window. At slightly reduced quality during interaction (see below), these views render at 20+, 14, 1.5, and 4.8 frames per second at* 640×480 *pixels, respectively.*

Abstract

Computer Aided Design (CAD) and Virtual Reality (VR) are becoming increasingly important tools for industrial design applications. Unfortunately, there is a huge and growing gap between what data CAD engineers are working on, what rendering quality is needed by designers and executives to faithfully judge a design variant, and what rendering capabilities are offered by commonly available VR frameworks. In particular, existing VR systems cannot currently cope with the accuracy demanded by CAD engineers, nor can they deliver the photorealistic rendering quality and reliability required by designers and decision makers.

In this paper, we describe a ray tracing based virtual reality framework that closes these gaps. In particular, the proposed system supports direct ray tracing of trimmed freeform surfaces even for complex models of thousands of patches, allows for accurately simulating reflections and refraction for glass and car paint effects, offers support for direct integration of measured materials via Bidirectional Texture Functions (BTFs), and even allows for soft environmental lighting from high dynamic range environment maps. All of these effects can be delivered interactively, and are demonstrated on a real-world industrial model, a complete Mercedes C-class car.

1. Introduction

Computer Aided Design and Virtual Reality are becoming increasingly important tools for industrial design applications. In particular large and high-end engineering projects such as cars or airplanes are already engineered almost entirely digital, as the cost for building physical mockups of such objects is very high. In practice however, this digital design is not as simple as it might first seem, as there are several problems and complication arising from conflicting demands of the different groups involved in such a project, namely CAD engineers, designers, VR specialists, and company executives.

CAD Engineers work on the raw geometric data of the model, usually using freeform data such as NURBS Surfaces [PT97]. The main objective of the CAD engineer is to model the individual geometric components of the car, and to perform evaluations like stress analysis, crash tests, or assembly simulation. For that reasons, CAD engineers are mostly interested in the *highest possible geometric ac-*

curacy, as e.g. an assembly simulation can easily produce wrong results when working on approximated data. Photorealistic rendering quality is usually not an objective for CAD designers – in fact, most of their tools do not even support reasonable material properties or even texture coordinates.

Designers: In contrast to CAD engineers, *designers* are usually interested in producing *photorealistic* results. As designers are responsible for the eventual look of the car to the customer, they strongly depend on the ability to *predict* how the model will eventually look in reality. For this reason, designers are usually interested in the highest rendering quality possible, and are particularly interested in realistic surface appearance, lighting effects like shadows, (accurate!) reflection and refraction, and if possible even global illumination.

Decision makers use visualizations – supplied by the designers or VR specialists – to evaluate and judge different variants of a model. As a decision for or against a certain variant may have significant financial consequences, decision makers require that what they see in VR is faithful to reality, and that both the geometry as well as the model appearance is as accurate as possible. Additionally, they are often neither computer specialists nor graphics experts, and thus have to take their decisions solely based on what they are being shown by the VR specialists.

VR Specialists have the task of taking the data prepared by CAD engineers and designers, and generating an interactive visualization for the decision makers. Unfortunately, the above-mentioned goals – high accuracy, high realism, and high performance – are in stark conflict to each other: In order to satisfy the interactivity constraints, virtually all of todays VR systems are built on triangle rasterization. With that however, rendering complete models at full accuracy is not possible, as freeform surfaces cannot be rendered directly, and the amount of triangles generated by a high-quality tessellation is in the order of tens of millions of triangles. Thus, VR specialists usually deliver their presentation on specially prepared "VR models" of the real data. In the best case, this involves a lot of various tools and manual effort for converting the model, tesselating, simplifying, removing invisible parts, and for fixing polygon orientations, degeneracies, and surface cracks, etc. In the worst case, this leads to costly remodeling a completely new, simpler version of the original CAD model.

Apart from lack of *geometric* accuracy, existing VR tools also fail to deliver the realism required by designers and decision makers. Whereas designers often make use of offline rendering processes, VR specialists have to deliver realtime frame rates, and thus often rely on approximations, manual model tuning, and "hand-painting" special textures to create reasonably nice images. Here as well, this "model preparation" step involves many different tools, complex workflows, and lots of manual effort. In practice, such model preparation is usually measured in "person weeks".

1.1. Limitations and Demands of Industrial VR

This process of working on specially prepared VR models is currently state of the art in industry, but has several important drawbacks: First, preparing these special VR models takes time, so changes to the original model may take several days before they can be shown in a VR presentation. This often leads to decision makers looking at outdated model variants. Apart from this "latency" issue, the personnel cost for the same tedious model preparation has to be spent anew for each iteration cycle. Finally, the qualitative limitations of existing, rasterization-based VR systems usually fail to deliver the realistic appearance that designers and decision makers depend on, potentially leading to suboptimal decisions, or requiring the construction of costly physical mockups.

1.2. Outline

In this paper, we describe a ray tracing based Virtual Reality framework that starts to close the gap between engineers, designers, and Virtual Reality. In particular, the proposed system supports direct ray tracing of trimmed freeform surfaces even for complex models of thousands of trimmed patches, allows for accurately simulating reflections and refraction for glass or car paint effects, allows for soft environmental lighting from high-dynamic range environment maps, and even offers support for direct integration of measured materials via Bidirectional Texture Functions (BTFs). All of these effects can be delivered interactively, and will be demonstrated on a real-world industrial dataset, a complete Mercedes C-class model (see Figure 1).

In the following, these individual components will be discussed and assembled step by step: Section 2 briefly sketches the freeform ray tracing module we use for rendering the base geometry. Following this, Section 3 discusses the glass and car paint shaders used for generating a realistic outside appearance, while Section 4 describes how smooth shadows from environmental lighting are generated. Section 5 then discusses how a high-quality car interior is achieved by using ray traced Bidirectional Texture Functions (BTFs), followed by some notes on our hardware setup and overall system performance in Section 6. Finally, in Section 7 we conlude with a discussion of some limitations – and potential extensoins – of our approach.

2. Freeform Ray Tracing for Real-World Data Sets

Before we can look into realistic surface appearance, we first have to be able to render the original model. In practice, this usually means supporting NURBS surfaces, as these – due to useful geometric properties and a compact representation – are the de-facto standard used by CAD engineers. In the last two decades researchers have proposed several approaches for ray tracing NURBS surfaces [SB86, PSL*99, MCFS00, WSC01], but due to the high cost of evaluating the NURBS equations this is usually

too slow for interactive performance. More recently Benthin et al. [BWS04] have presented a framework for interactive ray tracing of bicubic Bézier patches, that allowed for interactive performance even on a single CPU. Essentially, we build on a (significantly modified) variant of that framework.

2.1. NURBS to Bicubic Bézier Conversion

Since we only supports bicubic patches, we first convert the NURBS surface to an arbitrary-degree, rational Bézier representation, which can be done without any loss of accuracy [PT97]. This is achieved by increasing the multiplicity of each knot in the knot vectors of both parameter directions to the order of the NURBS surface in the corresponding parameter direction [PT97, Efr05]. The given model fortunately contains only non-rational surfaces, so no special handling of the rational part was necessary.

As our framework only supports non-rational Bézier patches of degree 3×3, the degree of each Bézier patch must then be either reduced or elevated, depending on the initial degree of the patch. This degree reduction of course may lead to a certain loss of accuracy, which is handled by a user-specified tolerance threshold: If the degree of a Bézier patch cannot be reduced without preserving the patch geometry below the tolerance threshold, the initial Bézier patch must be subdivided in the direction where the Bézier reduction step failed. Based on a given error threshold, this degree reduction is then applied recursively to each of the obtained subpatches. The same strategy can also be used for converting the contour trimming curves, which are given by (2D) NURBS curves in the surface's parameter domain.

In our example of the C Class, the original model consists of a total of 69,067 NURBS surfaces with 392,491 2D NURBS curves that form 73,749 trimming contours. Converting first yields 308,095 Bézier patches of arbitrary degree, which after degree reduction with error threshold yields 319,340 bicubic Bézier patches (see Figure 2). The resulting number of trimming curves in the model is 1.46 million, i.e., an average of 5 trimming curves per patch.

Obviously, approximating NURBS surfaces by bicubic Bézier patches also involves a certain loss of accuracy, and thus seems to conflict with our goal of working on the original CAD data. Nevertheless, typical NURBS ray tracing algorithms [SB86, MCFS00, PSL*99] are approximative in nature as well. Additionally, we already achieve significantly higher accuracy than a triangulation, as 320,000 smooth Bézier patches obviously achieves a significantly higher accuracy as representing the same model with a million triangles only. In particular, smooth Bézier patches allows for zooming onto the surface without eventually seeing the triangular discretization at the silhouettes. Additionally, as the trimming curves are fully integrated into the rendering process, we do not have to cope with all the problems that usually arise from finding nice triangulations along the trimming contours. Finally, the main strength of that approach is its full automatic: Whereas triangulation often require manual user intervention – i.e., for tuning parameters and fixing cracks, degeneracies etc in the triangulated surface – our framework is a fully automatic batch process that requires no user intervention at all.

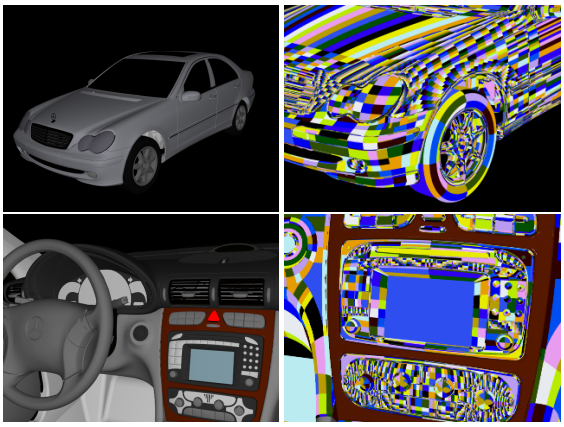

Figure 2: *The Mercedes C class model used in our experiments consists of 69,067 trimmed NURBS patches with 392,491 2D NURBS trimming curves, which we represent using 319,340 million trimmed Bézier patches with 1.4 million Bézier trimming curves. Left: Final model. Right: Color-coded Bézier patches showing the amount of geometric complexity. Note that the colored regions do not represent a tesselation, but a smooth Bézier patch each. On a single PC, the views on the left render at 0.86 and 3.01 frames per second at* 640×480 *pixels, respectively.*

2.2. Efficiently Ray Tracing the Freeform Model

Once the Bézier representation is generated, that representation is fed into the Bézier ray tracing plugin. Though this framework builds on earlier work on Benthin et al. [BWS04], this original framework eventually turned out to be quite problematic for a real-world data set such as the C class model. While this original framework was already used for scenes with thousands of patches, the C class model turned out to be much more complicated, mainly due to the excessive number of trimming curves, multiple patches overlapping and intersecting themselves, and untrimmed patches being very large in relation to their eventual trimmed counterparts. For example, the high accuracy requirements resulted in a high patch refinement level that got quite costly. Second, most of the traversal steps were performed on the patch level, which – as overlapping patches have to be intersected one after another – resulted in a huge amount of traversal steps. This is particularly annoying if these costly traversals then result in a hitpoint outside the trimming domain. Finally, the original system already supported trimming curves, but was optimized for only one or two trimming curves per patch, and could not cope with the huge amount of trimming curves in the C class model.

As a result, the existing system was completely re-

engineered. It now combines the original approach of Benthin et al. [BWS04] with some ideas from ray-tracing NURBS by Martin et al. [MCFS00]: Instead of performing a fixed number of de Casteljau subdivisions during traversal [BWS04], we now also use Newton-Iteration for computing the ray-patch intersections. In order to obtain good start values for the Newton Iteration, we follow Martin at al. [MCFS00] and – before rendering – subdivide the Bézier patches into multiple "sub-patches", that are then organized in an additional index structure. Generating subpatches for our 319,340 Bézier patches yields 1.2 million bicubic subpatches, whose accuracy clearly exceeds the roughly 1 million triangles in the triangulated counterpart of that model.

Instead of using a bounding volume hierarchy for these subpatches we use kd-trees, which allows for reusing the fast traversal algorithms as proposed by Wald et al. [Wal04]. In particular for regions where multiple patches overlap, we now no longer have to traverse each of these patches separately, but rather perform a single traversal in the 3D kd-tree, and for each encountered subpatch only have to perform the final Newton Iteration.

Once a hitpoint on the (untrimmed) Bézier patch is found, the trimming curves have to be evaluated. As in the given model some patches have up to several *hundred* trimming curve segments, an additional hierarchy was required for determining the trimming curves as well, which is realized by yet another (2D) kd-tree. Of course, we follow Martin et al., and determine completely trimmed patches before building the 3D kd-tree. Also, when building the kd-tree we are taking care to consider the trimming curves when computing the bounding box of the subpatch, which minimizes the number of subpatch intersections that later on get trimmed anyway.

Even with all these optimizations, ray tracing the C class model is still quite costly, due the huge geometric complexity and the large number of trimming curves (see Figure 2). Nevertheless, we still achieve interactive performance: On one dual 2.43GHz Opteron PC, we achieve 0.86 respectively 3.01 frames per second for the interior respectively exterior view shown in Figure 2. Note that the interior view in Figure 2 is one of the most costly views at all, due to the huge number of patches on the steering wheel, radio, and air vents.

3. Realistic Surface Appearance

Since we can now accurately render the original CAD geometry we geometry, we next focus on realistic material descriptions. Because ray tracing accurately simulates the physics of light transport and automatically accounts for global effects like reflections etc., we can fully concentrate on local material descriptions, which are realized via individual ray traced surface shaders.

Figure 3: *As most of a car's outside body consists of car paint, a realistic representation of that material is quite important. Left: Car with a typical Phong shader, illuminated by the HDR environment map. Right: With a ClearCoat shader that simulates Fresnel effects in the car paint and computes the resulting reflections. Note the accurate reflections (as opposed to a reflection map), as well as the varying reflectivity depending on the viewing angle.*

3.1. Car Paint

As most of a car's outside is made up of car paint, a realistic appearance of this material is obviously important. Car paint can have a wide range of appearances, including as diverse ingredients as pearlescence and sparkling effects [EKM01]. However, not all of these effects are equally important. One of the most obvious effects of car paint is its high specularity, which usually results in the car reflecting reflecting the environment. In typical VR systems, this is usually achieved by a highly specular Phong or ClearCoat [Sil98] shader with an appropriately chosen environment map.

Unfortunately, reflection maps usually result in significant artifacts, due to their "infinite distance" assumption that is violated by nearby geometry. This is particularly the case for highly curved regions such as at the fenders or door handles. Additionally, the infinite distance assumption makes reflection maps notoriously hard to use for the interior, where special reflection maps have to be computed for each object supposed to be be reflective. Furthermore, reflection maps do not allow for self-reflections, which is particularly problematic for e.g. the hood, which when standing in front of a car usually reflects the roof and windscreen (see Figure 3). Reflection maps must also account for reflection of nearby geometry such as the street or nearby cars. Thus reflection maps must be carefully constructed anew for each environment, and often have to be manually edited using PhotoShop to look good. This process can become quite costly.

As our framework is built on a ray tracer, these limitations obviously can be removed by computing real, accurate reflections. In order to improve realism, we usually surround the car by fully modeled environments. We still use an environment map for distant geometry such as the sky, but – due to a ray tracers good scalability in geometric complexity – can represent a large part of the environment by real geometry. Of course, we can easily switch between different environments during runtime, and all reflections will be fully accurate and correct any time, without *any* manual effort.

The reflectivity of car paint varies depending on the an-

gle at which the surface is seen, which cannot be captured with a standard Phong material model. For this reason, we use a shader based on a slightly more accurate variant of the "ClearCoat" model [Sil98], which essentially simulates a small layer of glass (the transparent coating) on an otherwise Phong-style material. The angular dependent reflectivity then is a result of the Fresnel factors for the coating. As all physical formulas for that are well known, implementing this "car paint" shader was straightforward.

Obviously, this relatively simple model for car paint is still not perfect. For example, we currently support neither glossy reflections nor pearlescence or sparkles, either. However, in practice the current set of effects has shown to be sufficient. The difference between our model and a Phong shader can be seen in Figure 3. Note that though these effects seem very subtle (in particular for still images and when printed on paper), there was explicit user demand for these features.

3.2. Glass

Of even higher importance for design reviews, in particular for cars, is Glass. Many other materials can – at least after enough manual tuning and hand-painting of textures – still be reasonably well represented by appropriately textured Phong models or specially designed programmable surface shaders. For glass however, this is not the case.

Glass is recursive in nature, as the color seen by an incoming ray hardly depends on the hit object at all, but almost entirely on what is seen in the reflected and refracted directions. Thus, rendering realistic glass with anything but a recursive ray tracing is next to impossible. For this reason, in typical VR systems glass is usually modeled by a simple mostly-transparent plane, with a slight gray-blue tint to make the glassy object visible at all.

This simplistic model is still state of the art in most of todays VR systems, but is mostly useless for reliable design decisions. In particular for glass, even very subtle effects, such as its slight reflectivity, can have a critical influence. For example, reflections of bright parts in the windshield or side windows can have significant security issues (e.g., glare, occlusion, or distraction). Accurately simulating such effects – in particular during interaction, where different configurations can be evaluated from different views – are extremely important for designers. In practice this is particularly important for the head and rear lights of the cars, due to the complex optical light paths inside these objects. However, a physically correct simulation of glass is impossible with current state-of-the-art VR systems.

For a ray tracer on the other side, glass is a pretty straightforward material to compute: It requires neither shadow rays nor costly texture operations, its physical behavior is well understood, and essentially only at most two new rays (for reflection and refraction) have to be shot. For this reason, Glass simulation was one of the first industrial applications

Figure 4: *Realistic simulation of glass effects has a strong impact on the level of realism of a rendered image. Left: Rendering glass materials as semi-transparent surfaces as usually done in rasterization-based systems. Right: With a physically correct, ray traced glass shader. Top: View from a drivers seat (effect slightly emphasized to reproduce on paper). Bottom: View from the outside.*

for realtime ray tracing [BWDS02]. Essentially, we use exactly the same implementation, except for a few minor optimizations and integration into a new framework. All the effects described in [BWDS02] – reflection, refraction, Fresnel terms, and termination of low-contributing paths – are supported in our current framework as well.

The impact of a realistic glass simulation can be seen in Figure 4, which compares the rendering quality of our ray traced glass shader with the quality as achieved by a semi-transparent plane as used in standard VR systems. Unfortunately, we cannot demonstrate this effect for the lights as well – though our system handles lights perfectly well [BWDS02], geometric data for the lamps unfortunately was not available for this model.

4. Smooth Shadows and HDR Environmental Lighting

Apart from materials, a realistic appearance of a car also also depends strongly on the incident illumination. For data sets such as cars the most natural source of illumination is its surrounding environment. For a clear, sunny sky, such illumination can be simulated by just placing a single directional light source into the respective direction.

However, in reality illumination usually is much more complex, resulting in smooth shadows and other effects. Because soft shadows are costly to compute, typical VR systems either use sharp shadows only (if at all), or confine themselves to a "hand-painted" shadow texture placed below the car. Of course, both methods again involve manual effort, have to be re-done for every change of the environment, often create inconsistencies between the shadows shown and the shadows as expected by the given environment, and generally fail to produce realistic appearance.

Figure 5: *Smooth environmental lighting: Left: Using 3 samples only, producing sharp shadows. Center: With interleaved sampling and discontinuity buffering, at ∼8 frames/s. Right: After accumulation. Also note the shadows in the background.*

4.1. Discretizing Environmental Lighting

For this reason, we have chosen to equip our framework with a module for environmental lighting from an HDR environment map in the spirit of [KK03, ARBJ03]. In particular, this module is designed to work fully automatically, and without any user invention except for specifying the environment map to be used. The respective "light shader" will use the same (high-dynamic range) environment map that is also used for rays that do not hit any geometry. Thus, the shadows always stay consistent with the chosen environment.

Unfortunately, producing smooth environmental lighting by randomly sampling an environment map often produces Monte Carlo noise [ARBJ03]. It is also quite costly due to the large number of samples required to reduce the noise. For that reason, we have chosen to discretize the illumination from the environment by (automatically) placing "virtual directional lights" in the spirit of [ARBJ03, KK03] that are then used to illuminate the scene (see Figure 5 below).

In order to support progressive accumulation (see below) for each loaded environment map we generate N such samples in a progressive way, i.e., one can always take the first k of these N samples, and – by simply scaling their power by $\frac{N}{k}$ – can use these k samples to get a coarser but nonetheless consistent representation of the environmental illumination.

4.2. Interleaved Sampling and Discontinuity Buffering

For reasonable quality of the environmental lighting, at least 20 to 40 samples would have to be computed, and really high-quality images would require even more samples. Unfortunately, due to the high cost of tracing the corresponding shadow rays, even using only 20 samples per pixel is not affordable during interaction.

To maintain interactive performance, we use interleaved sampling and discontinuity buffering as originally proposed for the Instant Global Illumination method [WKB*02, BWS03]. In that approach, not every pixel computes every shadow sample, but every other pixel in a 3×3 or 5×5 pattern uses a different set of shadow samples, whose results are then combined in an a-posteriori filtering step. As this filtering step only filters the irradiance (and not the final pixel colors), and additionally restricts filtering to pixels whose hitpoints pass certain continuity criteria, blurring over material or geometric discontinuities is minimized. Note that this technique was also already used in [KK03].

Using this method, the effective number of samples used per pixel after filtering is 9 respectively 25 times the number of shadow rays cast for each individual pixel, achieving nearly the same quality as when actually using that many samples per pixel. Thus, a reasonably good quality can be achieved even during interaction, where only a few (2–5) samples are affordable per pixel (see Figure 5).

As soon as camera motion stops, we perform the usual trick of progressively improving image quality by computing successive images of the same viewpoint with new random samples, and accumulating the resulting images.

In summary, with only 3–5 samples we achieve reasonably smooth shadows for environmental lighting even during interaction, and high-quality shadows after accumulating only a few frames (see Figure 5). In practice, this has shown to not be a problem, as the coarser shadow quality without accumulation is less perceptible during interaction. Note that the shadows computed by our method not only comprise the shadows of the car cast onto the floor, but, of course, also include the shadows the environment and the car casts onto themselves, respectively.

5. Support for Measured Real-World Materials using Bidirectional Texture Functions

While the outside appearance can be reasonably well handled with the techniques described above, the cars interior is more complex, including materials such as cloth, (structured) plastic, carpet, metal, leather, and wood. Most automotive companies have extremely high quality requirements for realistically rendering "their" materials, which cannot be satisfied using a common Phong reflection model with textures. Even bump mapping can only insufficiently capture the intricate lighting effects happening at the microstructure level of these materials. Thus, there is huge demand to *acquire* the surface appearance of samples of the *real-world* materials, and use those during rendering.

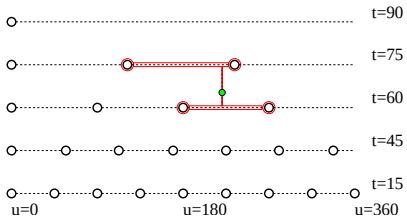

Figure 6: *Quadrilinear interpolation for reconstructing a smooth representation from the sampled BTF using successive linear interpolation in each dimension. This example shows a two-dimensional example, while the BTF interpolation across two directions is four-dimensional.*

5.1. Bidirectional Texture Functions

Unfortunately, measured *BRDF* data assumes a homogeneous surface, and cannot capture the complex surface patterns of the materials we are most interested in. Thus, Dana et al. [DvGNK99] have proposed to sample the texture of the material for many different light- and viewing-directions, yielding the so-called Bidirectional Texture Functions (BTFs). This approach has later been refined by Meseth et al. and Müller et al. [MMS*04], who proposed to compress the BTFs using clustered principle component analysis (PCA) [MMK03], and who presented an automatic BTF acquisition setup.

For the given Mercedes C-Class model, all of the surfaces had already been scanned by Bonn University in the course of the RealReflect project [Rea, MMS*04], and the resulting BTF data has been made available to us, courtesy of DaimlerChrysler AG. These BTFs are originally given as 256×256 pixel textures for every combination of the 81×81 samples of the viewing and incident lighting directions. Due to the HDR acquisition process, each of these $256 \times 256 \times 81 \times 81$ samples contains an RGB float triple. This huge amount of data (12GB per material) is compressed using clustered PCA [MMK03] with 32 clusters and 8 components per cluster. In addition to a reference to one of the PCA clusters, each of the 256×256 texels contain the 8 float weights for reconstructing the PCA, while each of the 32 clusters contains eight 81×81-dimensional base components, resulting in only roughly 24 Megabytes per material. During rendering, the BTF data for each $(u, v, \omega_i, \omega_o)$ sample is then decompressed on the fly from that PCA data.

5.2. Smooth Reconstruction using Quadrilinear Interpolation

Unfortunately the data set contains only values for 81×81 discrete (ω_i, ω_o) pairs. Thus, for any two given view and light direction during rendering, some value has to be reconstructed from these spares samples. With only 81 samples on the hemisphere, simply taking the nearest available sample yields severe discretization artifacts. To obtain visually pleasing results, it is therefore required to smoothly interpolate the data from multiple adjacent sample directions.

The BTFs are sampled on a hemisphere, which is discretized in θ direction into 15 degree steps, at $\theta = 0, 15, 30, 45, 60, 75$ degrees. The BTF does not actually represent a discretized BRDF $f_r(x, \omega_i, \omega_o)$, but rather its cosine-weighted counterpart $f_r(x, \omega_i, \omega_o)\cos\theta_i$. Thus, for the $\theta = 90$ ring all samples are zero, and are not stored.

The ϕ discretization on each such θ ring is chosen proportional to $\sin(\theta)$, ranging from 24 15-degree steps for $\theta = 75$, to a single sample at the pole (see Figure 6). This highly irregular sampling complicates bilinear interpolation (see Figure 6). We experimented with both a triangulating of the sample points, and with weighted distance interpolation, but have finally chosen to first linearly interpolate on each of the two nearest θ rings, and then to interpolate the result in θ direction, yielding a smooth reconstruction.

Unfortunately this bilinear interpolation is very costly: First, determining the correct sample indices and weights is costly, since many special cases have to be considered for each lookup (e.g., the missing samples for $\theta > 75$). Second, extracting a sample from the BTF dataset requires a costly PCA-decompression for each sample (i.e., accumulation of several terms addressed though many indirections). Third, we have to smoothly interpolate for *both* in- and outgoing direction, and thus have to perform a quadrilinear interpolation, i.e., we have to perform these costly PCA lookups 16 times for each pair of directions, and filter the resulting 16 values. Finally, as the incident direction ω_i varies for each light source, this costly procedure has to be performed anew for *every* light source. As a result, the BTF shader is extremely costly, and the weighting of the light samples is often more expensive than shooting the shadow ray itself.

5.3. Interior Lighting

As discussed in the previous section, we use environmental lighting or the car to produce a realistic model appearance on the outside. For the interior of the car, one would actually have to compute a full global illumination solution. In principle, it would be possible to use an interactive global illumination algorithm as proposed in [WKB*02, BWS03]. These methods however are specially optimized for mostly-diffuse scenes, and do not easily work for as complex shaders (glass, BTFs) as used in our framework. Furthermore, the complex lighting in the car interior would require too many virtual light source for achieving reasonable quality, which would probably not allow for interactive performance any more.

Precomputing global illumination using a Radiosity method [CW93] is not helpful either, as Radiosity is a directionally independent quantity, and is thus useless for illuminating BTFs. Precomputed Radiance Transfer methods [SKS02, KSS02, DAK*04] might be a reasonably alternative, but have never been applied to such models, yet.

For that reason, we have chosen to *not* yet compute any

Figure 7: *Comparison of using BTFs vs. usual textured Phong surfaces by a view into the car's cockpit. Left: Textured Phong. Right: Using measured BTFs. Top: Entire cockpit. Bottom: Zoom onto the wood and leather.*

global illumination in the car interior, but to confine ourselves to the *direct* environmental illumination only, plus a manually placed point light that "emulates" indirect lighting. In practice, most of the illumination patterns (such as shadows) are due to direct illumination, anyway, so the level of realism is still sufficiently high even without supporting indirect illumination yet. Nevertheless, in order to remove this final limitation in physical correctness, we are already looking into precomputing global illumination in a directionally dependent way. Preliminary results are available and look promising, but are not yet available for practical use, and will not be discussed in this paper.

5.4. Results

While the high cost of the BTFs is obviously a strong disadvantage, the increased level of realism makes more than up for that. Figure 7 shows a comparison of the measured BTF material versus a nicely textured Phong material. Note that the textures on the textured Phong model have already been provided with the original model. These are also photographs from the original materials and have been optimized to produce the highest image quality. Even in comparison to this already high quality, BTFs further increase the quality, in particular for cloth, wood, and leather (see Figure 7). Though this effect can hardly be seen on printed paper, it becomes clearly apparent during interaction, in particular when the viewing and/or lighting directions change. To our knowledge, this is the first time that BTFs have been ray traced or used in Virtual Reality at all.

6. Final Integration and Overall Results

In the preceding sections, we have discussed all the individual components of our framework, starting with directly ray tracing the complex, trimmed freeform geometry of the car, over various ray traced surface shaders (including glass and car paint), an efficient method for computing environmental illumination, and support for measured materials using bidirectional texture functions.

Using a ray tracing based framework, combining all these individual effects works mostly automatic: For example, a view from the inside will not only show the BTFs directly, but these surfaces will be correctly reflected in the mirrors or in reflections off the glass. Similarly, the environment – which is to a large degree modeled by real geometry as well – will correctly cast smooth shadows like the car as well, and will be correctly visible through the glass shaders as well as correctly reflected in the car paint, on the mirrors, etc. Some examples of the final rendering quality can e.g. be seen in Figure 8, and on the color page at the end. Note that though these images look exactly like offline renderings, they can be computed at interactive rates.

As our entire framework is realized via shaders and plugins into the OpenRT engine [Wal04], all the features of an existing OpenRT-based VR application are available for our framework as well. For example, the user can define and use lighting, geometry, and shading scenarios, can specify and interactively edit cutting planes, surface shaders, and light sources, can switch variants and move objects, etc.

6.1. Hardware Setup

As already discussed in the previous sections, the targeted level of accuracy and quality does not come for free: For as complex a model as the C class model, the freeform ray tracing incurs a high computational cost (see Section 2), and the complex shaders used in our framework further add to this cost. Furthermore, due to the complexity of the used shaders (e.g., glass and BTFs) we cannot use the fast packet-traversal code described in [WSBW01], and rather have to use the slower single-ray traversal code. Finally, the extensive use of secondary and shadow rays for computing reflections, refractions, and environmental illumination additionally multiplies the rendering cost. Thus, in order to achieve interactive performance, we use the parallelization framework of the OpenRT engine, and run our system on multiple PCs.

As a dedicated ray tracing cluster unfortunately was not available for our experiments, we have taken the commodity PCs available in our lab, and have used those for our experiments. In particular, we use a mix of 4 dual-3GHz Xeon machines, 1 quad-2.4GHz Opteron, and 9 various dual-Opteron machines ranging from 1.8 to 2.4 GHz (30 CPUs total). All machines are connected via multiple switches, some via 100MBit Fast Ethernet, others via Gigabit. Neither the machines not the network have been available for the ray tracer exclusively, and have been in use by others as well.

6.2. Final Performance

Even under these suboptimal conditions, we achieve interactive performance of 1.5 to 20+ frames per second at

Figure 8: *The final system. Top: Driver's view of the interior. Bottom: Outside view. Note the smooth shadows, the correct reflections in the car's paint, the subtle glass effects in the windshield, as well as the BTF interior. These views run at 1.5 and 10+ frames per second at* 640×480 *pixels, respectively. Left: During camera motion. Right: After accumulating several frames.*

640×480 pixels, depending on viewpoint and complexity of features seen. In particular, the views in Figure 8 can be rendered at 1.5 and 10+ frames per second, respectively, and even the accumulated performance is reached after only a few frames. Note that the interior view is one of the most expensive views in the entire car. Though this could not be tested, it can be expected that a dedicated ray tracing cluster, with better network and available for exclusive use by the ray tracer, would yield even better results.

Though the compute power accumulated for our experiments at first seems quite considerable, for industrial standards it is actually not significant. For example, a dedicated ray tracing cluster of $\sim$50 dual-Opteron nodes has recently been set up at a major german car manufacturer, where it is used to drive a 3200×1200 pixel PowerWall for ray tracing based design reviews. We estimate the compute power of such a setup to be sufficient for running our complete framework also in fullscreen mode.

7. Discussion

In this paper, we have presented a novel VR framework that is entirely based on realtime ray tracing, and that particularly targets a very high level of realism. The presented framework offers direct ray tracing of freeform surfaces, high-quality rendering of car paint and glass materials, smooth shadows from high-dynamic range environment maps, and support for measured materials using BTFs.

7.1. Comparison to Existing VR Solutions

In comparison to existing rasterization-based VR solutions, the proposed system offers a number of advantages. Directly ray tracing a freeform model allows for significantly improved geometric accuracy of up to 1.2 million smooth Bézier subpatches. Apart from this pure geometric complexity, directly ray tracing the trimmed Bézier patches removes the need for generating a complex triangulation of the model, which significantly simplifies the VR workflow.

Using ray traced shaders for e.g. glass and car paint delivers *accurate* and reliable images, and for the first time allows for *predictive rendering* even during interactive design reviews. Similarly, smooth environmental illumination greatly exceeds the realism previously available in VR applications.

All of these features have previously been possible only during offline rendering and are usually not available in practical VR systems at all. Though some of these features are already being worked on also for rasterization hardware (e.g. [MMK03, GBK04]), but few of such research results are actually used in industry. Finally, we believe our system to be the only one to support *all* of these effects at the same time and in a fully integrated way.

7.2. Shortcomings and Limitations

Even though the presented system qualitatively exceeds existing VR solutions, there are also several open issues.

First of all, due to the high computational cost of the freeform geometry, complex surface shaders, and illumination effects, our framework is currently restricted to relatively low resolutions (640×480) and frame rates ($<$ 10 fps), while already requiring a considerable amount of aggregate compute power. On the other hand, hardware cost is often one of the lesser problems for industrial applications, and much larger clusters than the one used in our experiments have already been installed for similar purposes. On such a hardware platform, it should even be possible to reach fullscreen resolutions at interactive frame rates. Even so, the compute power required for driving a fully interactive VR setup (including high-resolution output devices) is still significant, and - in particular - considerably higher than that for typical non-ray traced VR systems.

Also on the quality side, there is lot of room for improvements: For example, the simple logarithmic tone mapping used in our system works well in practice, but is far from "state of the art" in offline rendering technology, where much better tone mapping algorithms are being used. Though these tone mapping techniques are well-known, in particular the global operators are tricky to implement in a distributed ray tracing system.

Similarly, the car paint shader used in our system is a rather trivial shader. Much better car paint models – often generated via acquisition of real car paint BRDFs – are well known.

Though the quality far exceeds typical VR setups, we do not offer all the rendering effects that other offline renderers use. In particular, not supporting global illumination – in

particular in the interior of the car - is a severe limitation. Offline rendering packages typically provide global illumination effects, and thus generate far more accurate and "photorealistic" images than we do.

Developments into these areas are already under way. For example, a prototypical implementation of a precomputation-based technique for rendering the given model with full global illumination (including caustics, highlights, glossyness, BTFs, illumination through the windshields etc) is already available [Wal05]. Right now however this method is not yet fully integrated into the given system, and some incompatibilities exist. Similarly, in a related project Günther et al. have shown that much higher realism for car paint can be realized with a specially designed shader that is based on acquisition, digitalization, and fitting of real-world car paint samples [GCG*05]. Both the DIRmap-technique as well as Günther et al.'s measured car paint shader have been implemented in the OpenRT system, and adding them should only be a question of invested programming time. So far, however, this integration has not yet taken place, as end-user demand for adding these two features has been quite limited.

7.3. Summary

In this paper, we have proposed a system designed for ray tracing based VR applications. Though the system still leaves room for improvements – in both performance and quality – it for the first time allows for ray traced design reviews even for complex models composed of thousands of freeform surfaces. In particular typical ray tracing effects that are notoriously hard to simulate under VR – i.e., reflections, refraction, shadowing effects, and Glass – can be simulated accurately and interactively.

Advanced shading technology – in particular Global Illumination, better tone mapping, and a better car paint model – are already being developed for this system, and – using a fully plug-n-play enabled ray tracing engine – should be relatively easy to integrate into the final framework.

In summary, we reach neither the quality standards of high-end offline renderers, not the interactivity of standard VR applications. We do, however, offer significantly more realistic images than VR applications that are at least close to offline renderings – and still at interactive rates.

Acknowledgements

This paper would not have been possible without the gracious support of DaimlerChrysler AG, who have provided the C Class Model in both triangulated and NURBS version, and who have granted permission to use the measured BTF data. Second, we would like to thank the RealReflect project, in particular the Computer Graphics Group at Bonn University, for providing the BTF data and help in numerous ways. Finally, we have to thank the system administration groups of the MPI and Saarland University for mustering and providing the required compute power.

References

[ARBJ03] AGARWAL S., RAMAMOORTHI R., BELONGIE S., JENSEN H. W.: Structured Importance Sampling of Environment Maps. *Computer Graphics (Proceedings of ACM SIGGRAPH) 22*, 3 (2003), 605–612.

[BWDS02] BENTHIN C., WALD I., DAHMEN T., SLUSALLEK P.: Interactive Headlight Simulation – A Case Study of Distributed Interactive Ray Tracing. In *Proceedings of the 4th Eurographics Workshop on Parallel Graphics and Visualization (PGV)* (2002), pp. 81–88.

[BWS03] BENTHIN C., WALD I., SLUSALLEK P.: A Scalable Approach to Interactive Global Illumination. *Computer Graphics Forum 22*, 3 (2003), 621–630. (Proceedings of Eurographics).

[BWS04] BENTHIN C., WALD I., SLUSALLEK P.: Interactive Ray Tracing of Free-Form Surfaces. In *Proceedings of Afrigraph* (November 2004), pp. 99–106.

[CW93] COHEN M. F., WALLACE J. R.: *Radiosity and Realistic Image Synthesis*. Morgan Kaufmann Publishers, 1993.

[DAK*04] DMITRIEV K., ANNEN T., KRAWCZYK G., MYSZKOWSKI K. O., SEIDEL H.-P.: A CAVE System for Interactive Modeling of Global Illumination in Car Interior. In *ACM Symposium on Virtual Reality Software and Technology* (2004), pp. 137–145.

[DvGNK99] DANA K. J., VAN GINNEKEN B., NAYAR S. K., KONDERINK J. J.: Reflectance and Texture of Real-World Surfaces. *ACM Transactions on Graphics 18*, 1 (1999), 1–34.

[Efr05] EFREMOV A.: *Efficient Ray Tracing of Trimmed NURBS Surfaces*. Master's thesis, Computer Science Department, University of Saarland, 2005.

[EKM01] ERSHOV S., KOLCHIN K., MYSZKOWSKI K.: Rendering Pearlescent Appearance Based on Paint-Composition Modeling. In *Computer Graphics Forum, Proceedings of Eurographics 2001* (2001), pp. 227–238.

[GBK04] GUTHE M., BALÁZS A., KLEIN R.: Real-time out-of-core trimmed NURBS rendering and editing. In *Vision, Modeling and Visualisation 2004* (November 2004), pp. 323–330.

[GCG*05] GÜNTHER J., CHEN T., GOESELE M., WALD I., SEIDEL H.-P.: Efficient Acquisition and Realistic Rendering of Car Paint. In *Proceedings of 10th International Fall Workshop - Vision, Modeling, and Visualization (VMV) 2005* (Erlangen, Germany, November 2005), Greiner G., Hornegger J., Niemann H.,, Stamminger M., (Eds.), Akademische Verlagsgesellschaft Aka, pp. 487–494.

[KK03] KOLLIG T., KELLER A.: Efficient Illumination by

High Dynamic Range Images. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering* (2003), Eurographics Association, pp. 45–50.

[KSS02] KAUTZ J., SLOAN P.-P., SNYDER J.: Fast, Arbitrary BRDF Shading for Low-Frequency Lighting Using Spherical Harmonics. In *Rendering Techniques* (2002), pp. 301–308. (Proceedings of Eurographics Workshop on Rendering).

[MCFS00] MARTIN W., COHEN E., FISH R., SHIRLEY P.: Practical Ray Tracing of Trimmed NURBS Surfaces. *Journal of Graphics Tools 5* (2000), 27–52.

[MMK03] MÜLLER G., MESETH J., KLEIN R.: Compression and Real-Time Rendering of Measured BTFs using Local PCA. In *Vision, Modeling and Visualisation 2003* (November 2003), pp. 271–280.

[MMS*04] MÜLLER G., MESETH J., SATTLER M., SARLETTE R., KLEIN R.: Acquisition, Synthesis and Rendering of Bidirectional Texture Functions. In *Eurographics 2004, State of the Art Reports* (2004), pp. 69–94.

[PSL*99] PARKER S., SHIRLEY P., LIVNAT Y., HANSEN C., SLOAN P.-P.: Interactive Ray Tracing. In *Proceedings of Interactive 3D Graphics* (1999), pp. 119–126.

[PT97] PIEGL L., TILLER W.: *The NURBS book, 2nd edition*. Springer-Verlag, Inc., 1997.

[Rea] REALREFLECT: The Real Reflect Project. http://www.realreflect.org.

[SB86] SWEENEY M., BARTELS R.: Ray Tracing Free-Form B-Spline Surfaces. *IEEE Computer Graphics and Applications 6*, 3 (1986), 41–49.

[Sil98] SILICON GRAPHICS, INC.: ClearCoat360. http://-www.sgi.com/products/software/clearcoat, 1998.

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. In *Proceedings of ACM SIGGRAPH* (2002), pp. 527–536.

[Wal04] WALD I.: *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004. Available at http://www.mpi-sb.mpg.de/~wald/PhD/.

[Wal05] WALD I.: *High-Quality Global Illumination Walkthroughs using Discretized Incident Radiance Maps*. Tech. Rep. UUSCI-2005-010, SCI Institute, University of Utah, 2005. available at http://www.sci.utah.edu/~wald.

[WKB*02] WALD I., KOLLIG T., BENTHIN C., KELLER A., SLUSALLEK P.: Interactive Global Illumination using Fast Ray Tracing. *Rendering Techniques* (2002), 15–24. (Proceedings of the 13th Eurographics Workshop on Rendering).

[WSBW01] WALD I., SLUSALLEK P., BENTHIN C., WAGNER M.: Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum 20*, 3 (2001), 153–164. (Proceedings of Eurographics).

[WSC01] WANG S., SHIH Z., CHANG R.: An Efficient and Stable Ray Tracing Algorithm for Parametric Surfaces. *18th Journal of Information Science and Engineering* (2001), 541–561.

Examples

Figure 9: *Progressively turning on the features of our framework: a) Freeform model, in plain gray Phong. b) After adding an environment and assigning the car paint shader, but still with hard shadows and without glass shaders. c) After adding glass and computing environmental illumination, after accumulation.*

Figure 10: *Interior of the car modelled via bidirectional texture functions (BTFs). a) Entire cockpit (without external lighting). b) Zoom onto some wood and leather materials. c) In comparison, the view with textured Phong materials. The distortions in both variants result from distorted texture coordinates supplied with the model.*

Figure 11: *Exterior views of the car. Note the smooth shadows and accurate reflections. a) Distant view. b) Closeup, during user interaction. c) After accumulating several frames.*

Figure 12: *a) Reflections of the interior in the windshield and side windows, when evaluating effects like glare, occlusion, or distraction (effects artificially emphasized to reproduce on paper). b) Zoom onto the mirror, during interaction. c) After accumulating several frames.*